CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Open-source Scientific Visualization: VisIt and ParaView

Jean M. Favre
Visualization Task Leader

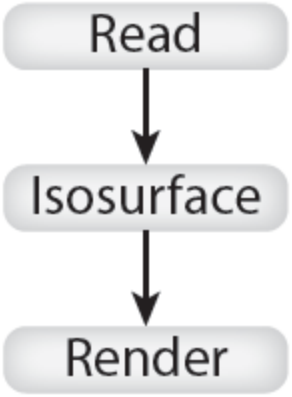
November 2013

Agenda

- **Visualization pipelines**
  - Parallel pipelines
  - Rendering modes
- **Data formats, parallel I/O and parallel visualization**
- **Remote, client-server, parallel viz**
- **Demonstrations with ParaView**
- **Demonstrations with VisIt**

Visualization Pipelines: Introduction

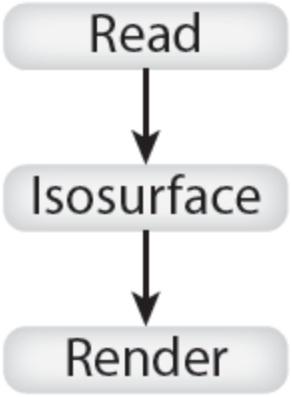
- From a survey article by Ken Moreland, IEEE Transactions on Visualizations and Computer Graphics, vol 19. no 3, March 2013



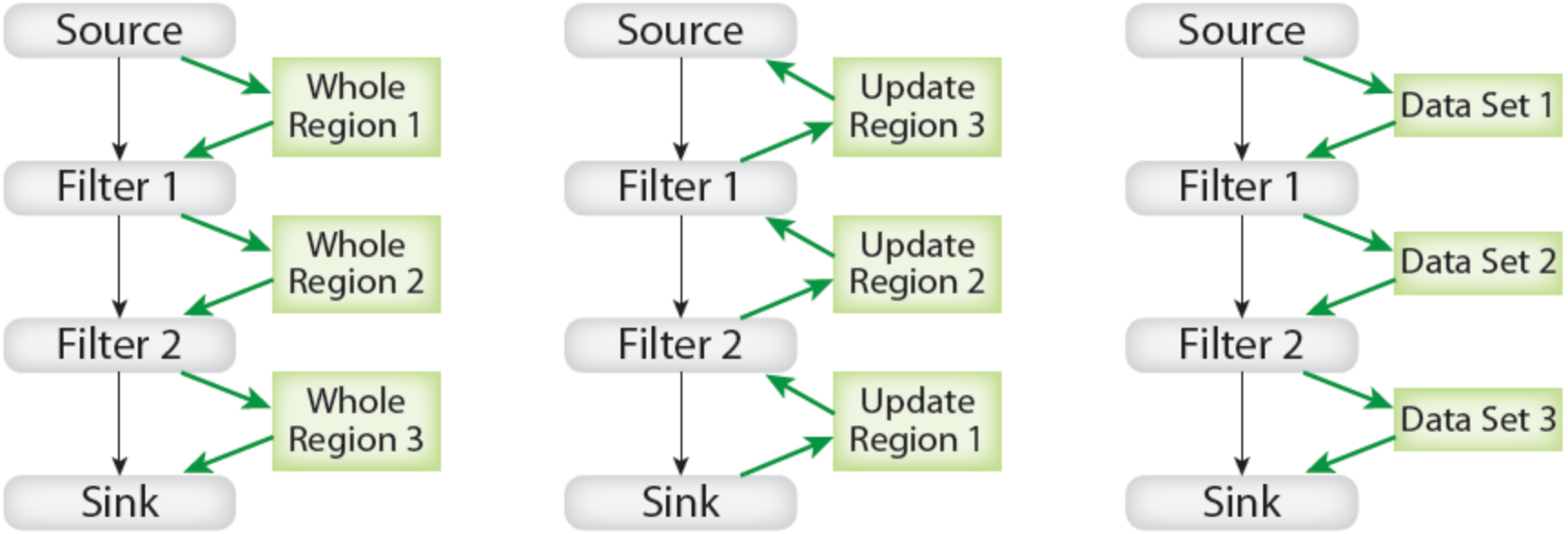
A visualization pipeline embodies a *dataflow network* in which computation is described as a collection of executable *modules* that are connected in a directed graph representing how data moves between modules. There are three types of modules: *sources*, *filters*, and *sinks*.

Visualization Pipelines: Definitions

- **Modules** are functional units, with 0 or more inputs ports and 0 or more output ports.
- **Connections** are directional attachments between input and output ports.
- **Execution** management is inherent in the pipeline
  - Event-driven
  - Demand-driven

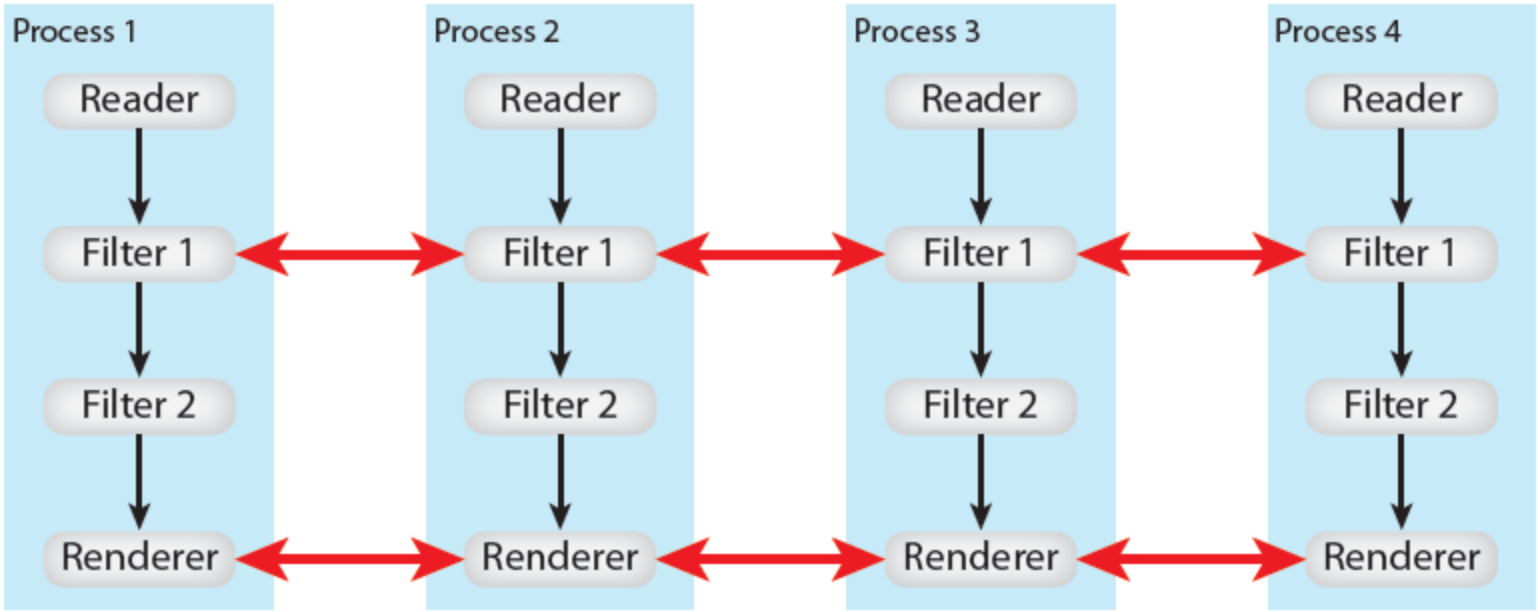


Visualization Pipelines: Metadata



- **1st pass: Sources describe the region they can generate.**
- **2nd pass: The application decides which region the sink should process.**
- **3rd pass: The actual data flow thru the pipeline**

Visualization Pipelines: Data Parallelism

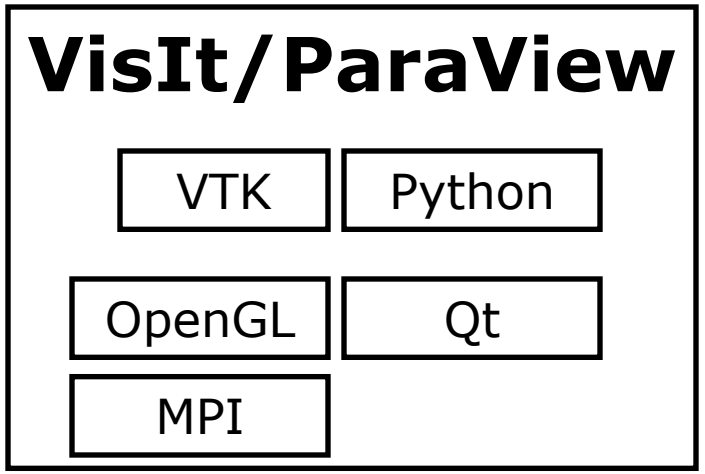


- **Data parallelism partitions the input data into a set number of pieces, and replicates the pipeline for each piece.**
- **Some filters will have to exchange information (e.g. streamlines)**
- **A special rendering phase will be needed.**

VisIt and ParaView are based on VTK

The Visualization ToolKit (VTK) is an open source, freely available software system for 3D computer graphics, image processing, and visualization.

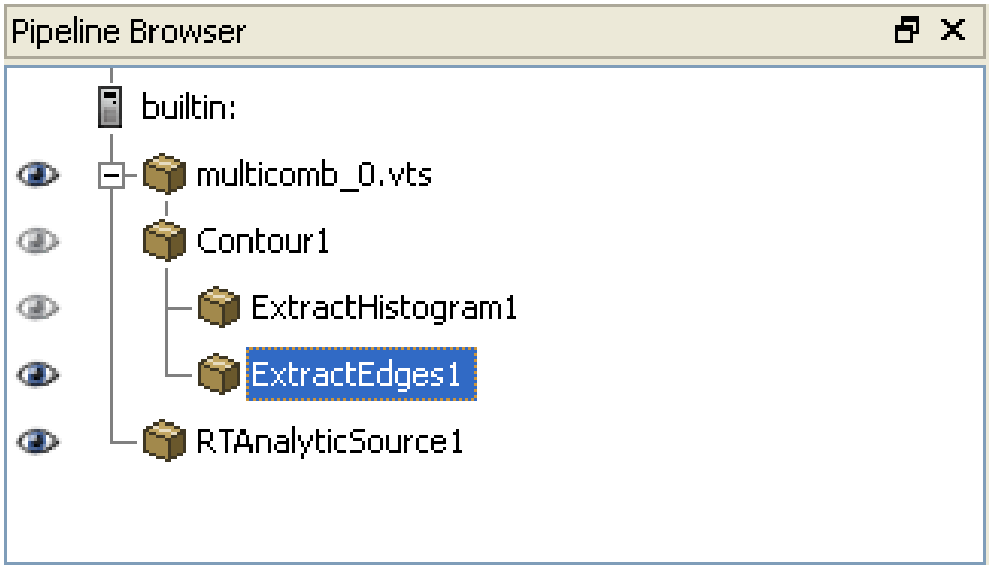
VisIt and ParaView are end-user applications based on VTK, with support for:

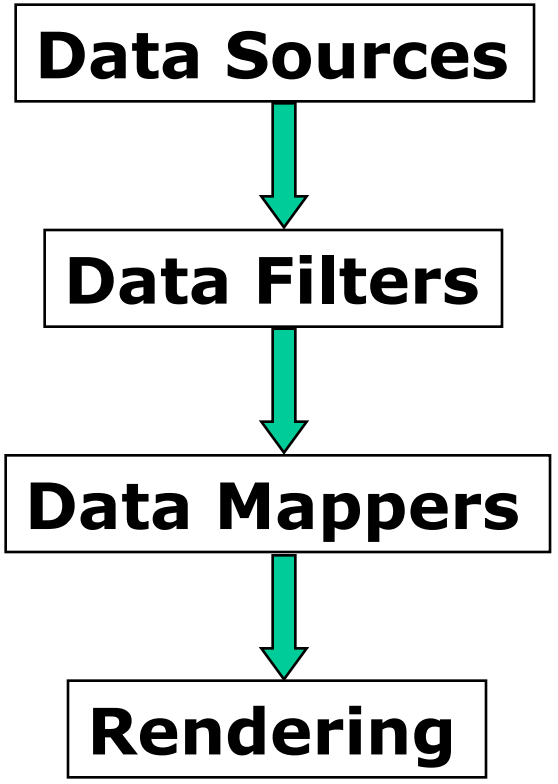


- Parallel Data Archiving
- Parallel Reading
- Parallel Processing
- Parallel Rendering
- Single node, client-server, MPI cluster rendering

The VTK visualization pipeline (1)

VTK's main execution paradigm is the *data-flow*, i.e. the concept of a downstream flow of data






```
Filter.SetInputConnection(Source.GetOutputPort())
Mapper.SetInputConnection(Filter.GetOutputPort())
```

Examples of Filters/Sources


Contour


Cut


Clip


Threshold


Extract grid


Warp vector


Stream lines


Integrate flow


Surface vectors


Glyph


Calculator


Pick cell


Probe


Group


Ungroup


AMR outline


AMR extract part


AMR surface


Wavelet


Measure


Fractal


Sphere

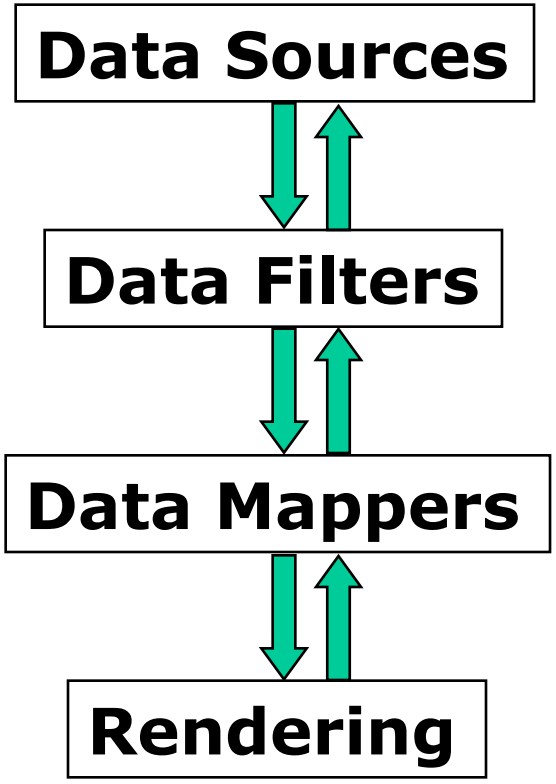

Superquadric

The VTK visualization pipeline (2)

- **VTK extends the *data-flow* paradigm**
- **VTK acts as an *event-flow* environment, where data flow downstream and events (or information) flow upstream**

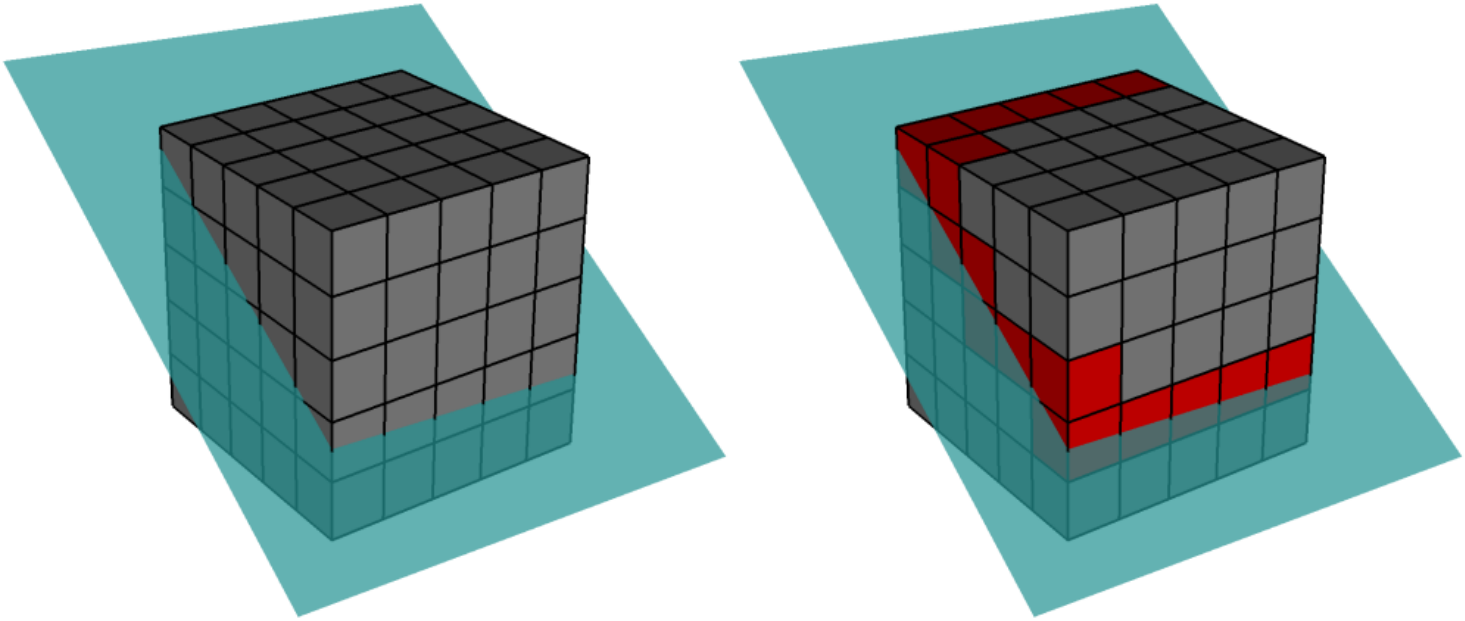
ParaView's Rendering drives the execution:
view.StillRender()

VisIt defines its own meta-data package, called "Contracts"

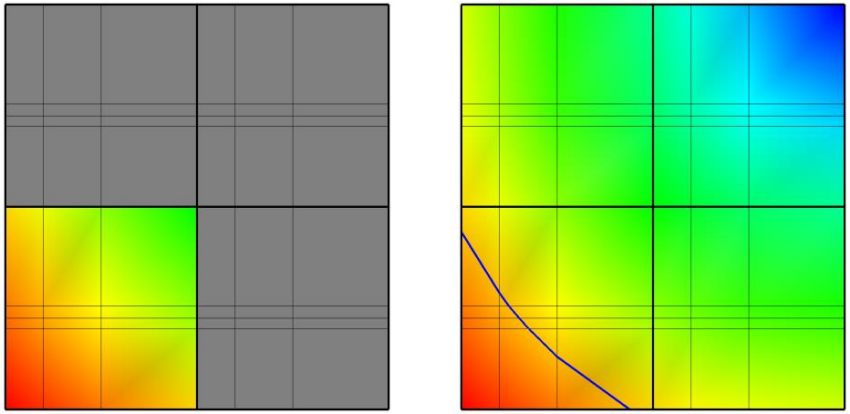


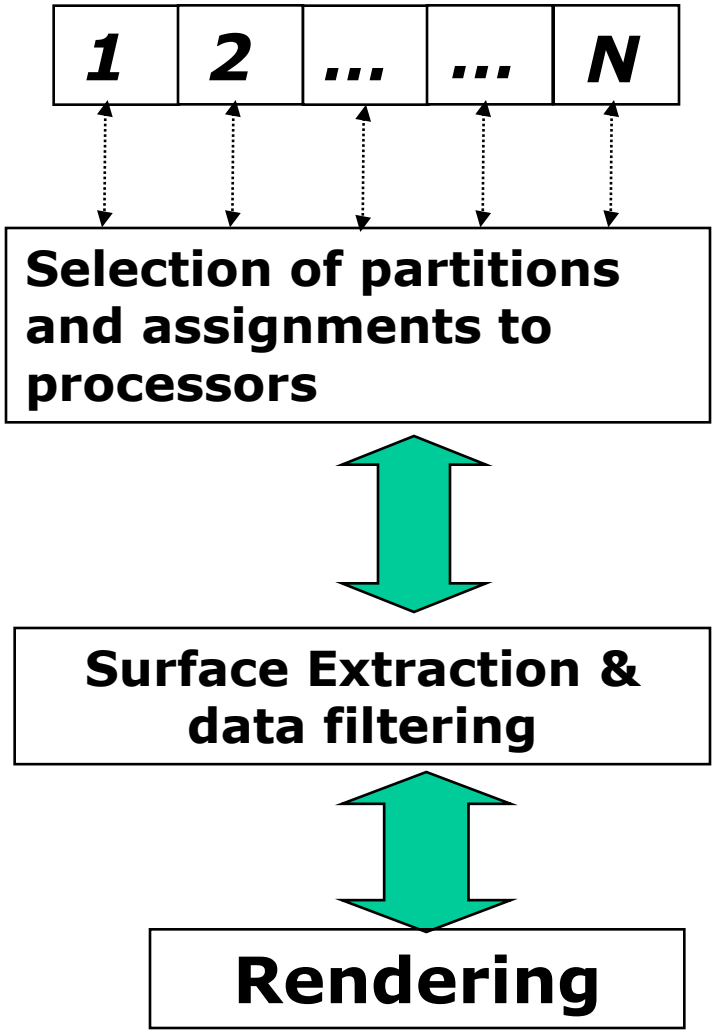
Example of a VisIt Contract

Spatial extents are examined and the visualization pipeline is by-passed for those outside the range



Example of a VisIt Contract



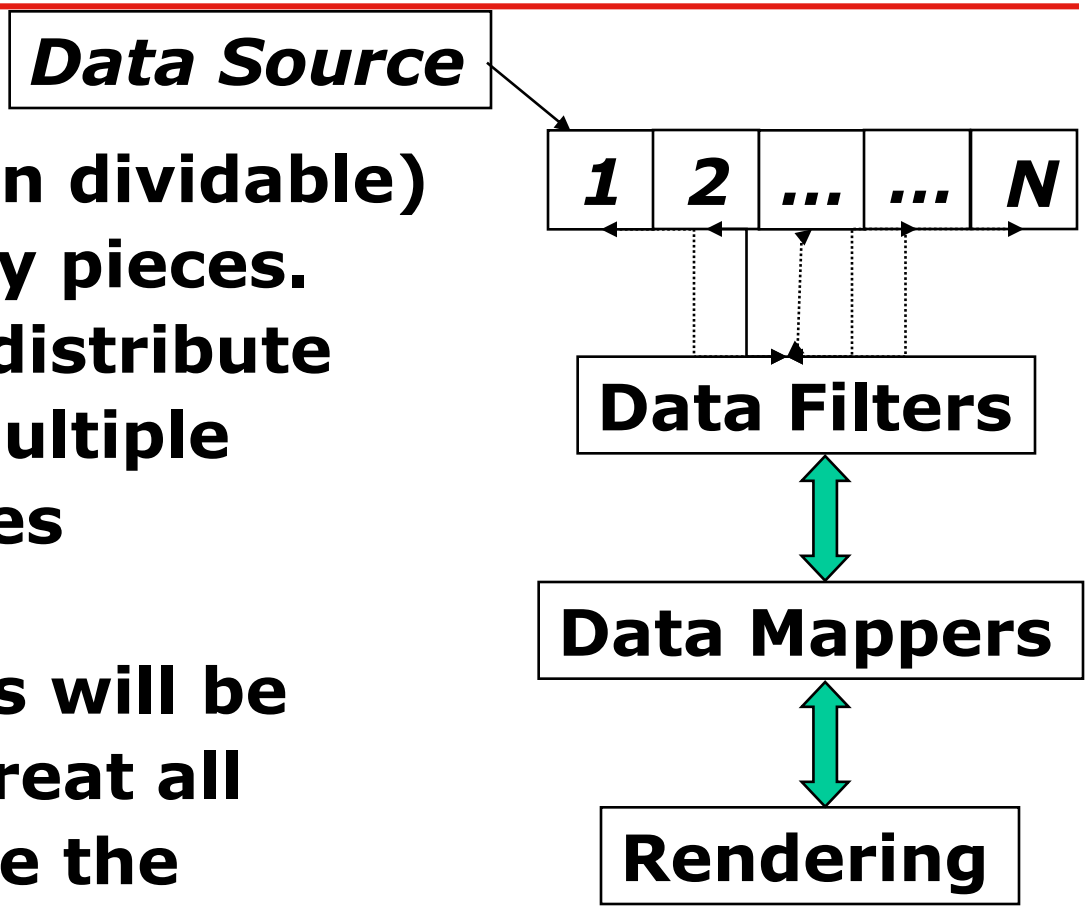


Data extents (min & max) are examined and the visualization pipeline is by-passed for those outside the range

The VTK visualization pipeline (3)

- **Large data (when dividable) can be treated by pieces. The Source will distribute data pieces to multiple execution engines**

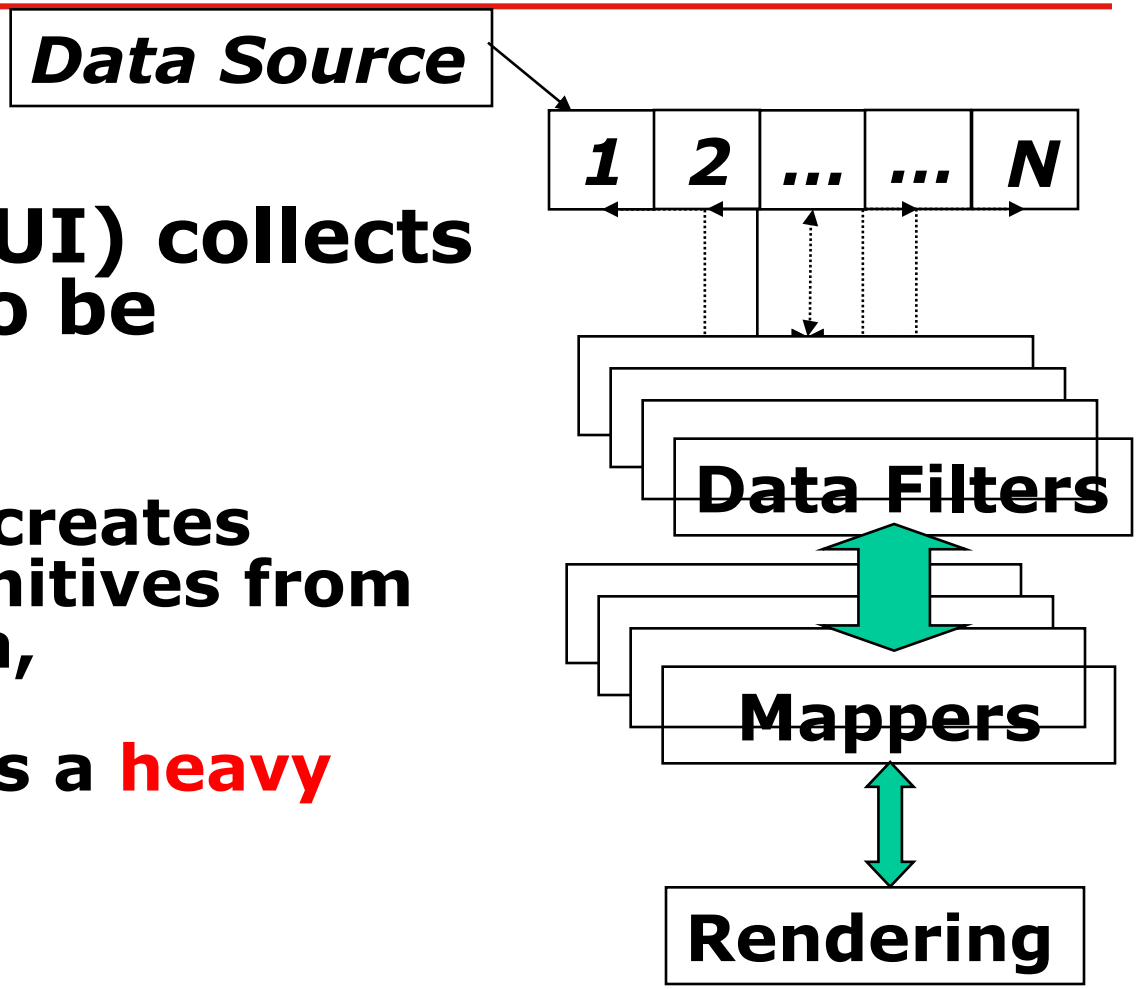
- **Parallel pipelines will be instantiated to treat all pieces and create the graphics output. This is hidden from the user.**



First Rendering option

The client (GUI) collects all objects to be rendered

- **Each pipeline creates rendering primitives from its partial data,**
- **The client does a heavy rendering**

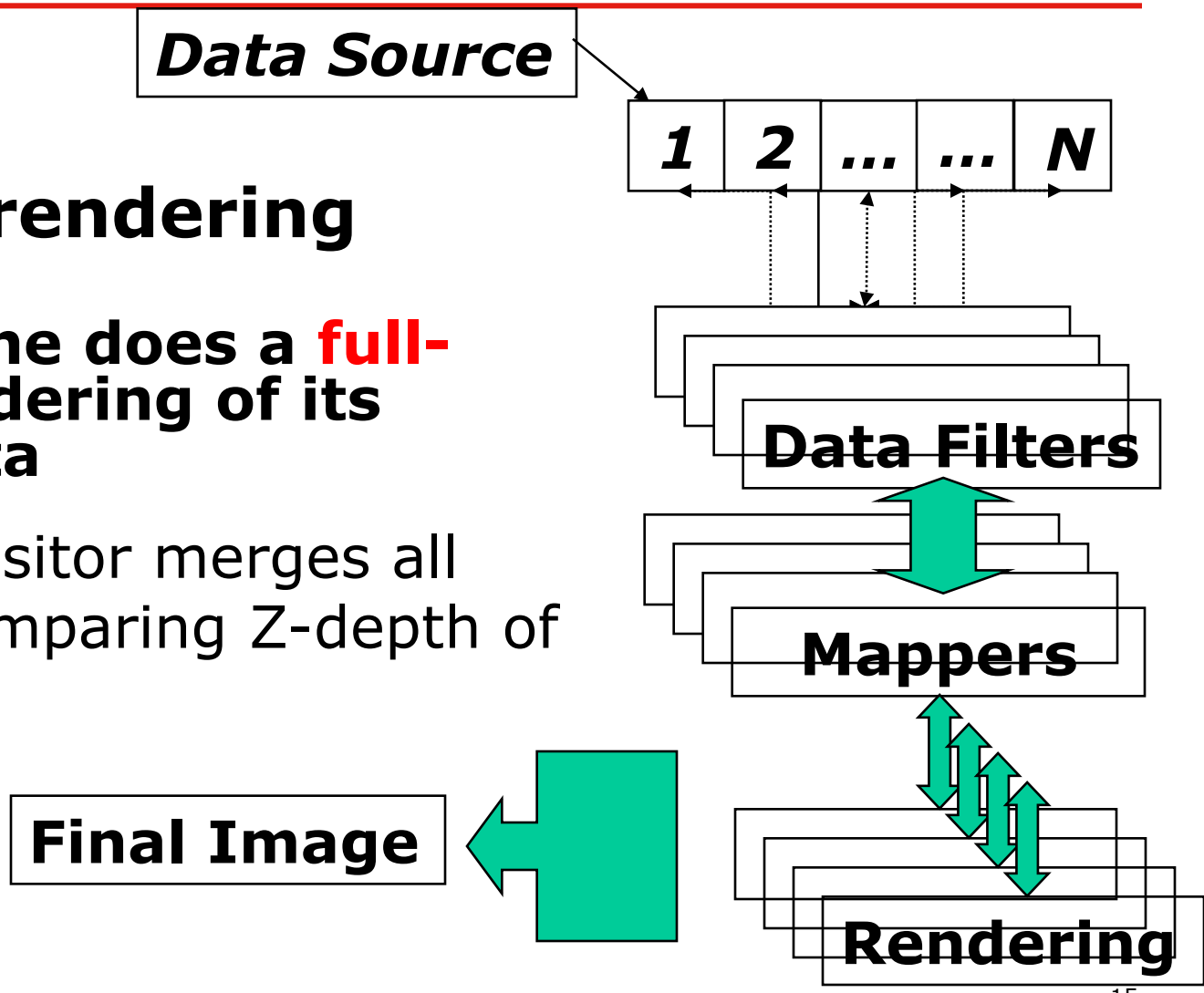


Second Rendering option

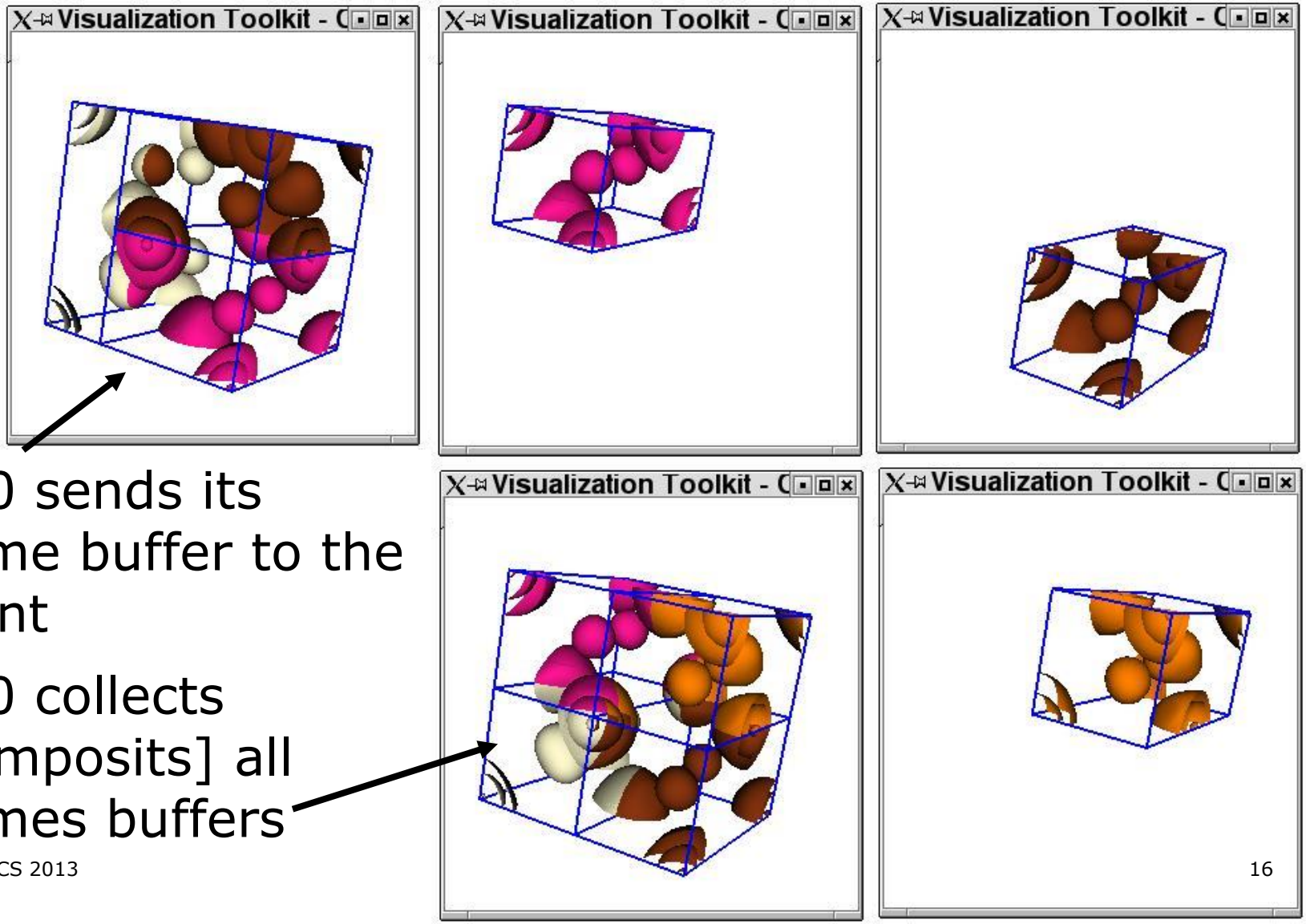
Sort-last rendering

Each pipeline does a full-frame rendering of its partial data

An image compositor merges all images by comparing Z-depth of all pixels



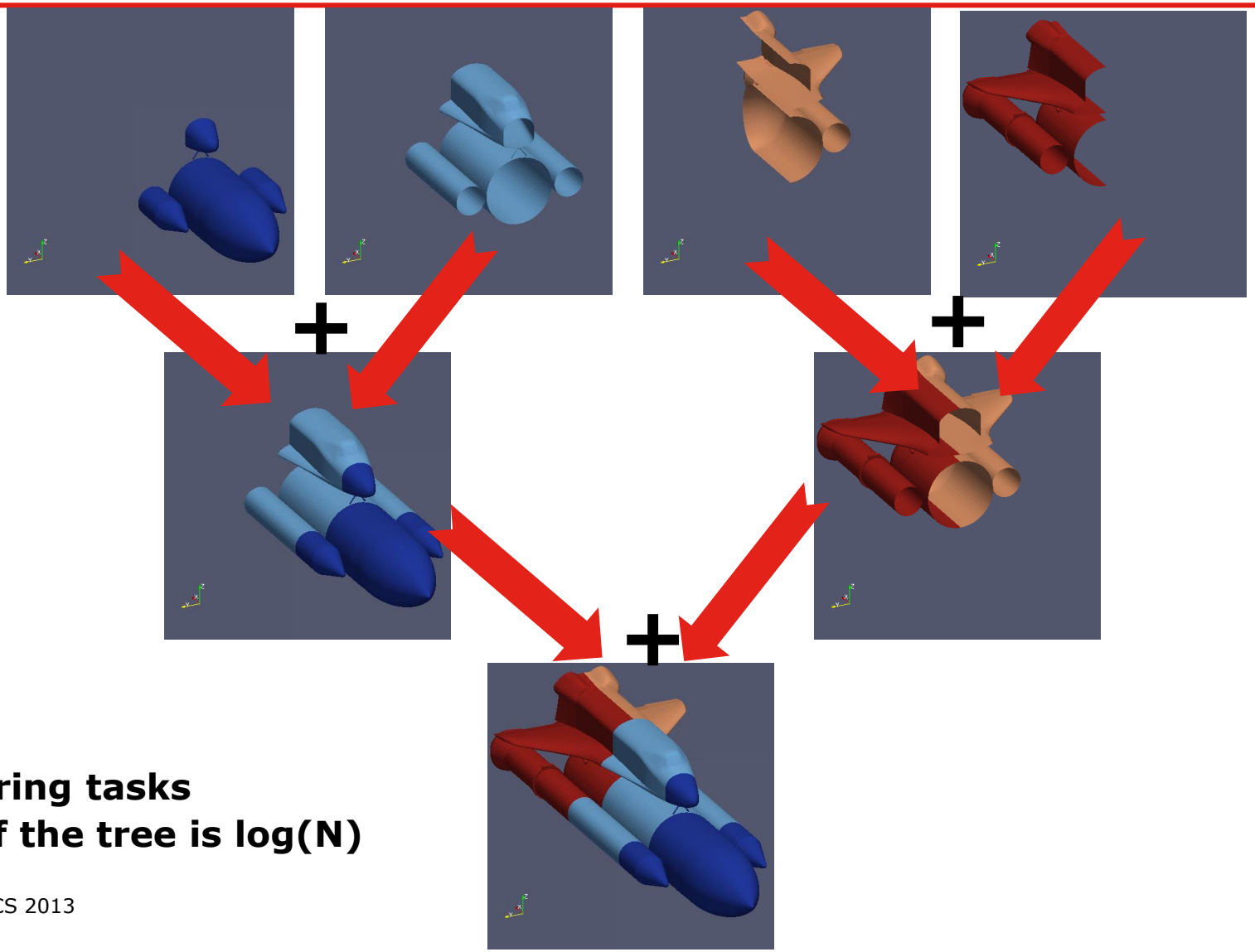
Sort-last rendering [pixel compositing]



Node 0 sends its frame buffer to the client

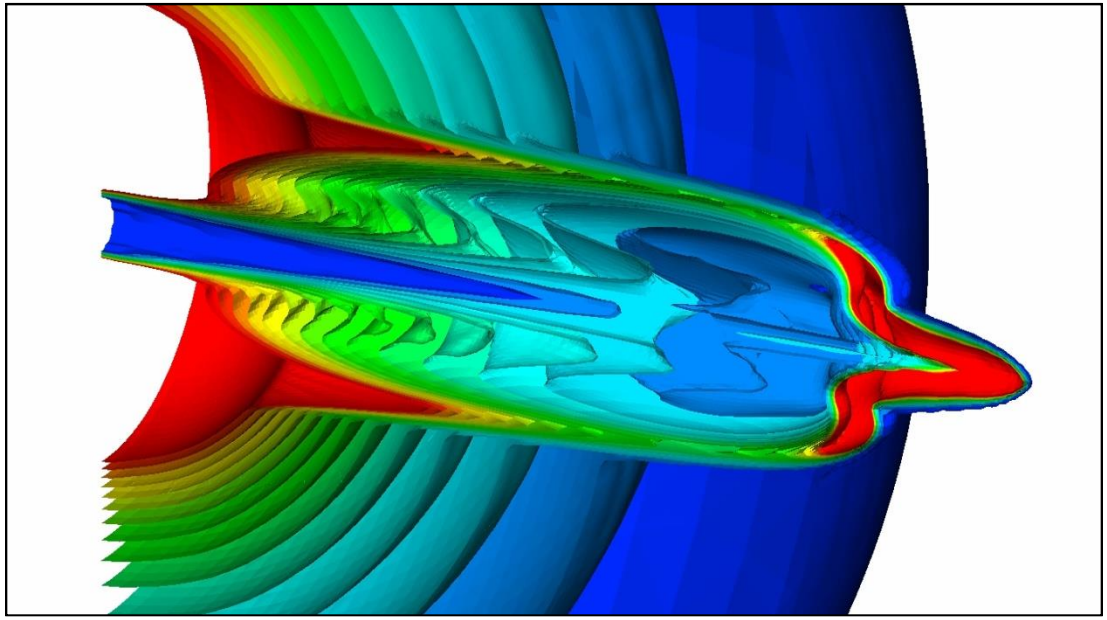
Node 0 collects [composits] all frames buffers

Sort-last rendering [pixel compositing]



- **N rendering tasks**
- **Depth of the tree is log(N)**

Arbitrary (or adaptive) 3-D data partitioning



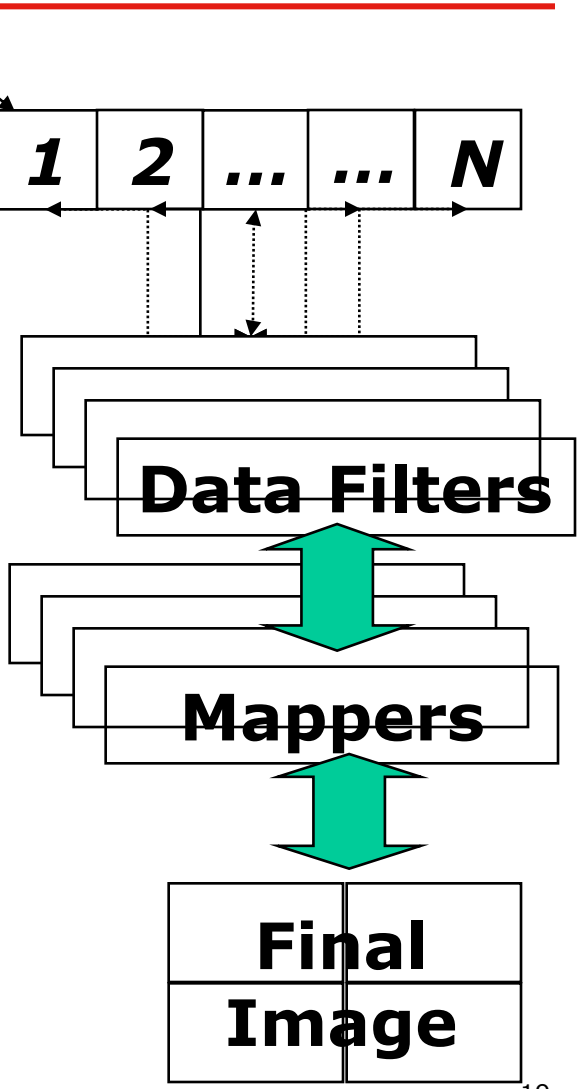
Is the final image order-independent?

A sort-last compositing enables complete freedom in data partitioning. Each pixel carries its color & depth

Third Rendering option

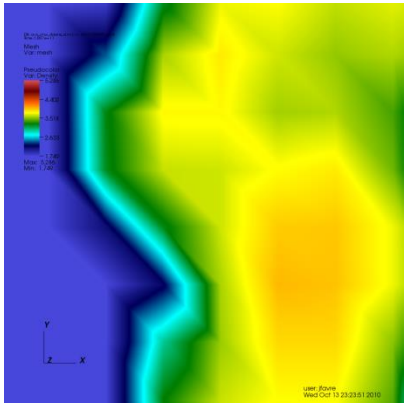
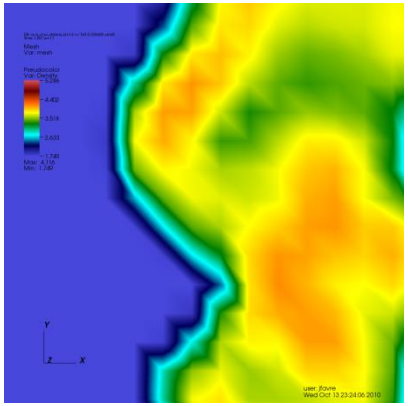
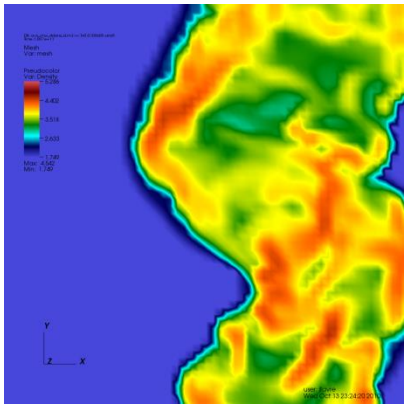
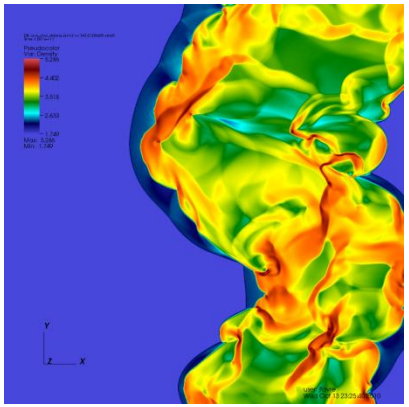
Tiled-Display

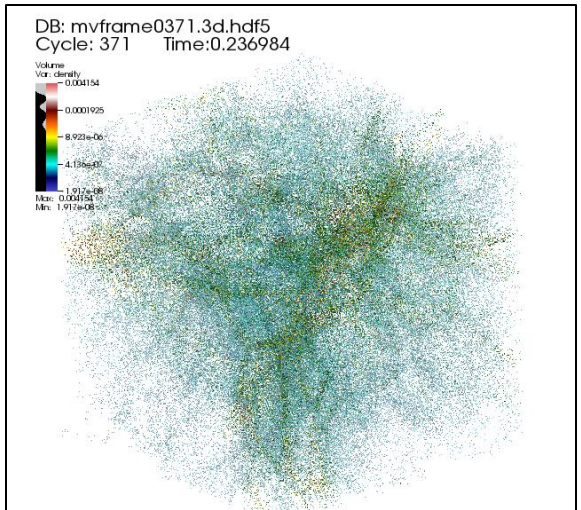
Each renderer does a partial-frame rendering of the full data

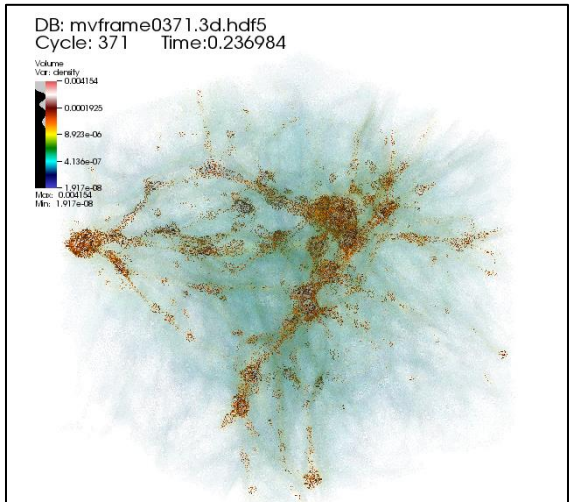


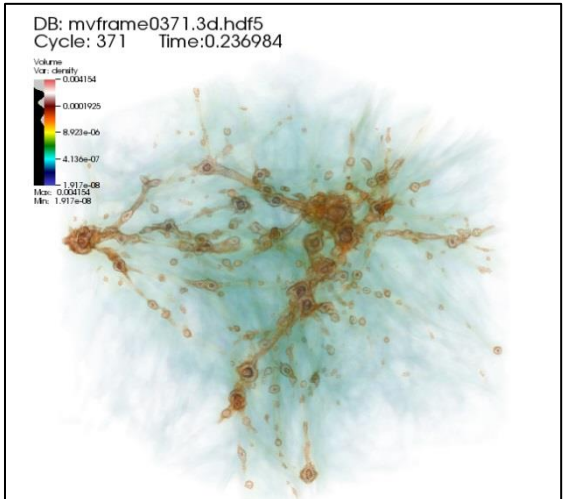
When there is too much data…

Adaptive resolution processing should be used







When large data require distributed processing

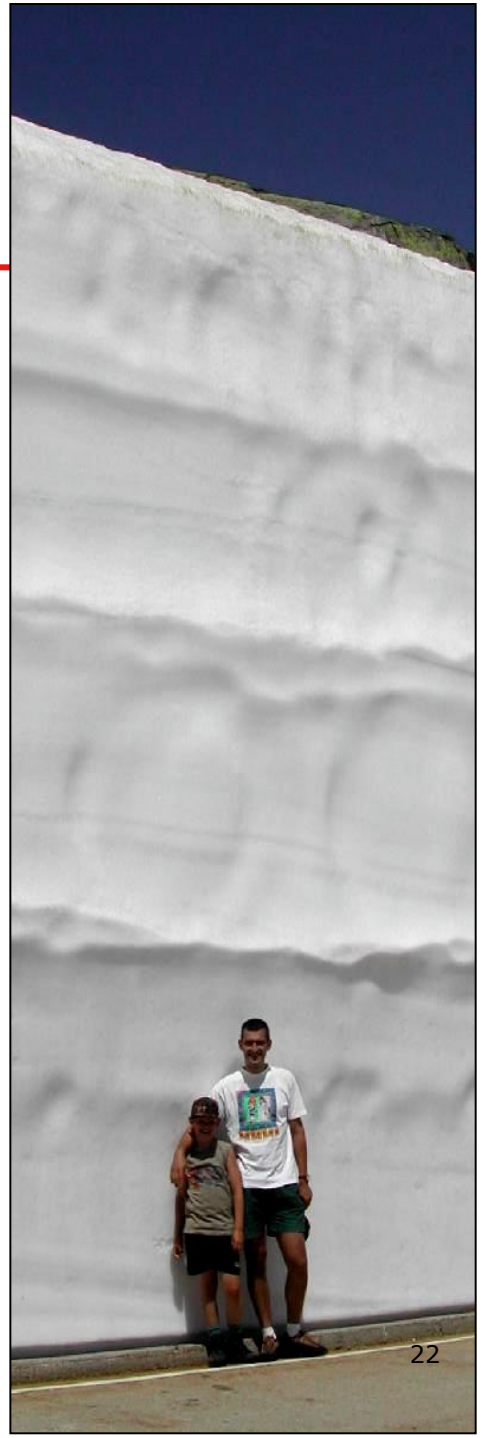
- **Sub-sampling can help prototype a visualization**
  - As long as the data format/reader supports it. (see the Xdmf reader in ParaView)
- **Piece-wise processing (on a single node)**
  - Data streaming (when the whole visualization will not fit in memory)
- **Distributed processing (on multiple nodes)**
  - Parallel file I/O
  - Parallel processing
  - Parallel rendering

Sub-sampling, streaming or multi-pass…

- **The snow removal was done in about 5 passes**

Data Streaming = Divide and conquer

- **Load datasets of <u>any size</u> by splitting the volumes in pieces**
- **Process the split data**



Example: Digital Elevation Model

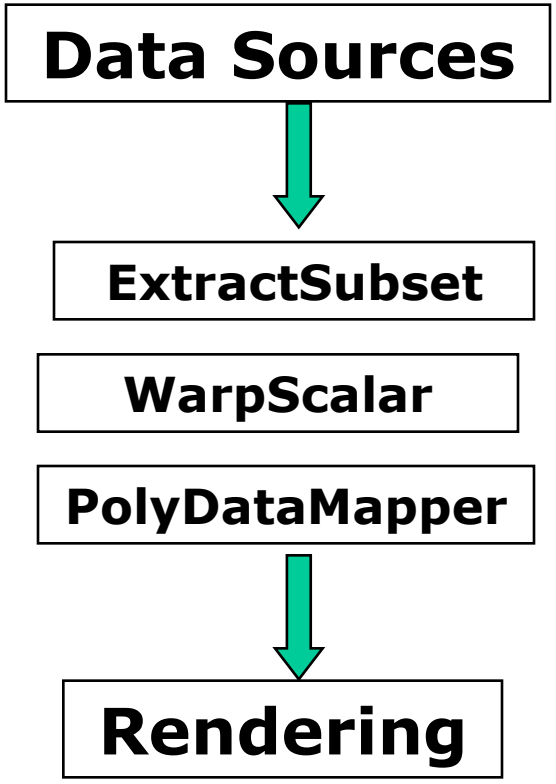
The VTK file header =>

```
# vtk DataFile Version 3.0
European DEM File
BINARY
DATASET STRUCTURED_POINTS
DIMENSIONS 8319 7638 1
ORIGIN 0 0 0
SPACING 1 1 1
POINT_DATA 63540522
```

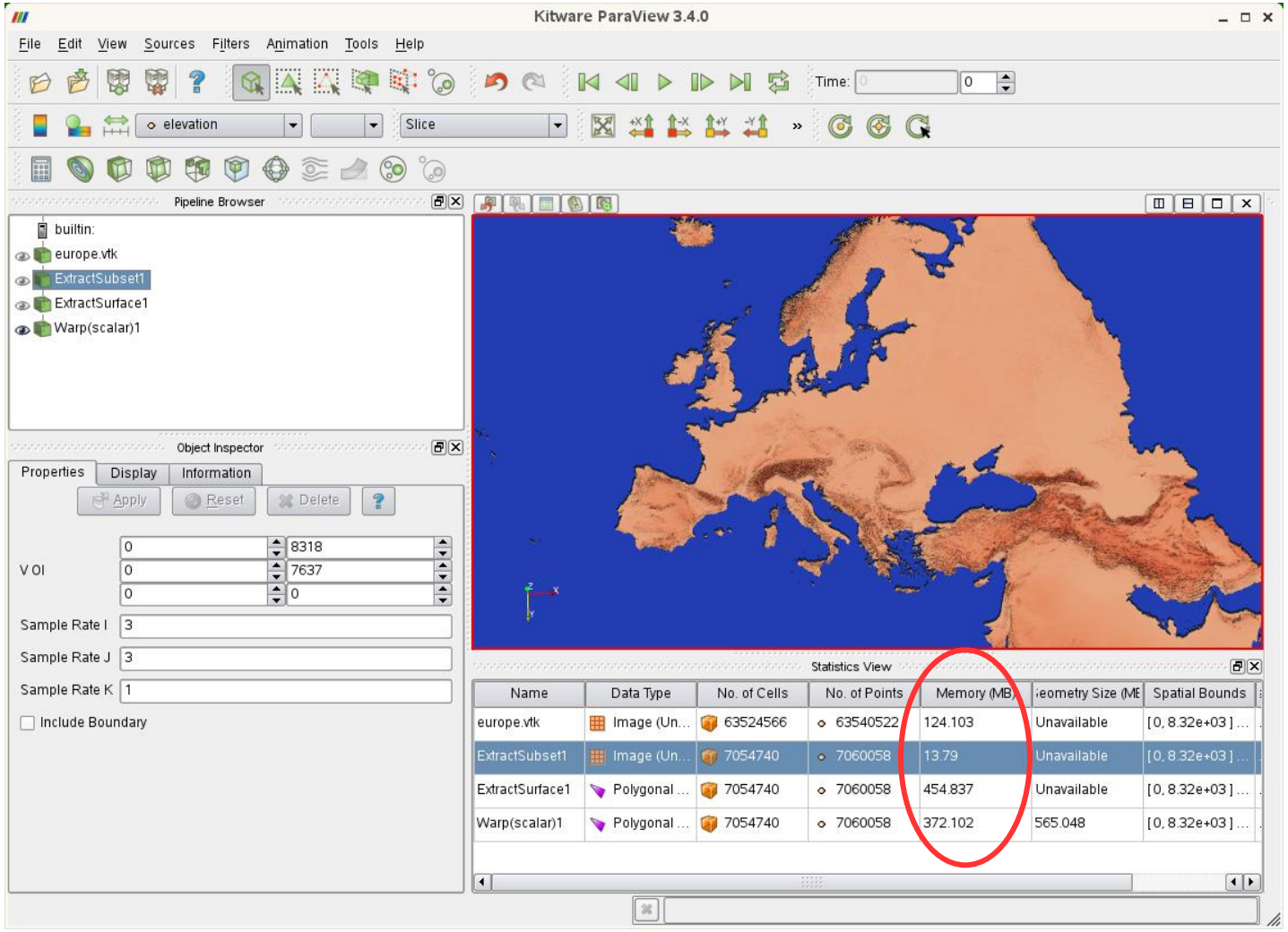
Use sub-sampling when data are too big

Warning: 64 millions points are first read in memory, then sub-sampled

The memory footprint can still be huge



Memory usage blows-up down the pipeline...



© CSCS 2013

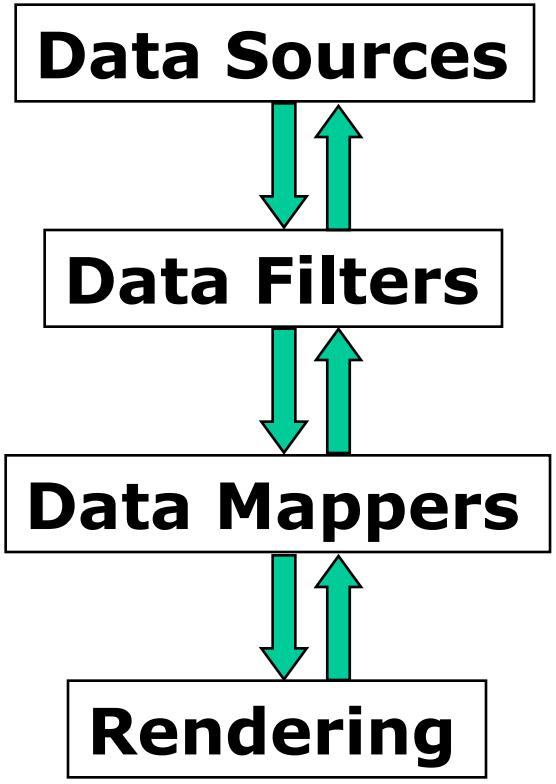
Data Streaming in VTK

- **Data larger than memory can be easily treated**
- Piece by piece
- Releasing or re-using memory after each subset
- Optionally accumulating sub-object representations for the final image

- **The upstream filters should be prepared to handle piece requests of any size**
- **Each filter can translate the piece request**

Reminder: VTK pipeline

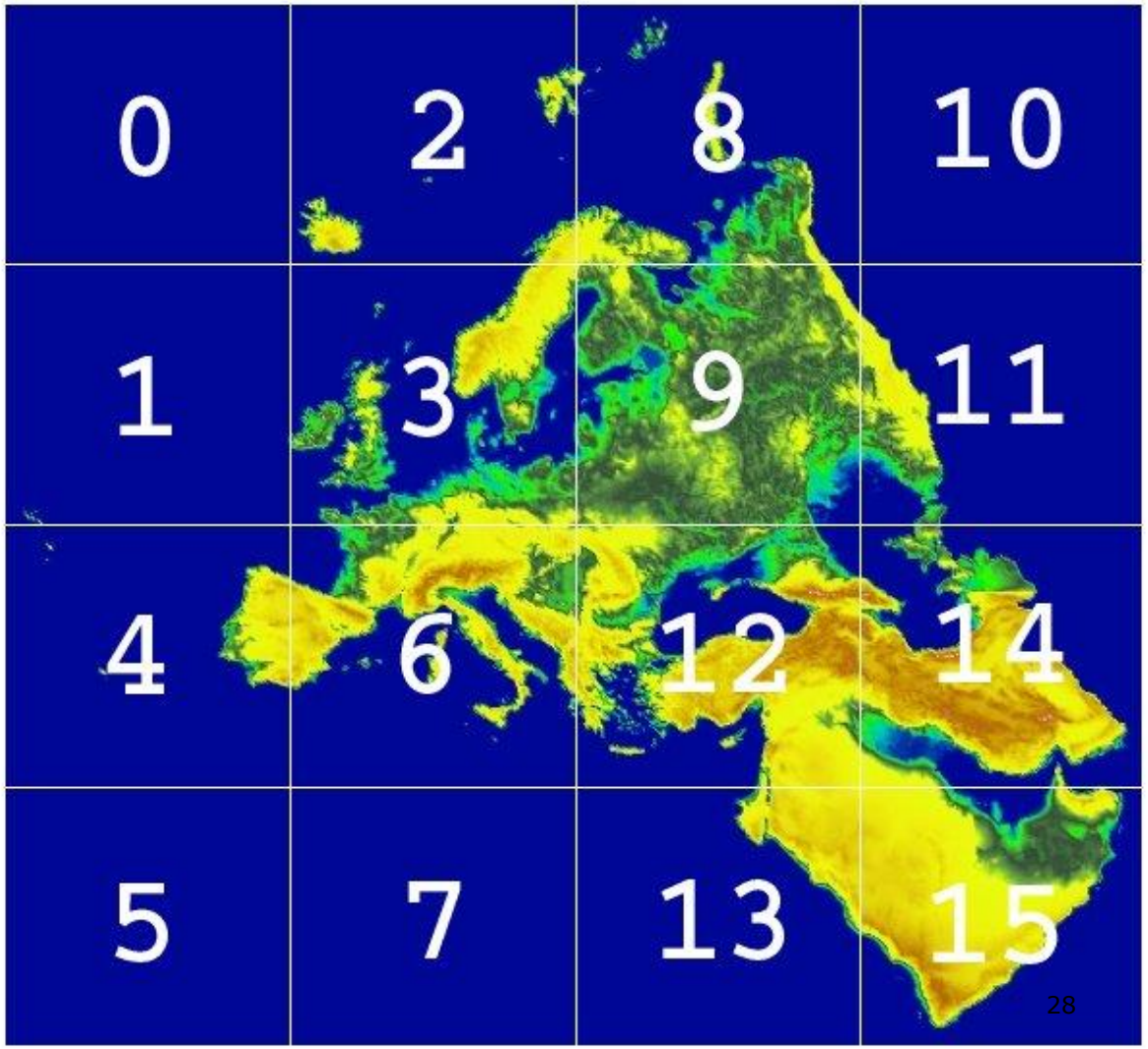
- The VTK pipeline enables a two-way exchange of data & information.
- The renderer drives the request for data updates.



- First pass: Get general bounds information, without reading the data
- Second pass: Decide how much to sub-divide [The Extent Translator], and process pieces

The Extent Translator

The Extent Translator does a binary subdivision of the data and let the user access pieces one at a time

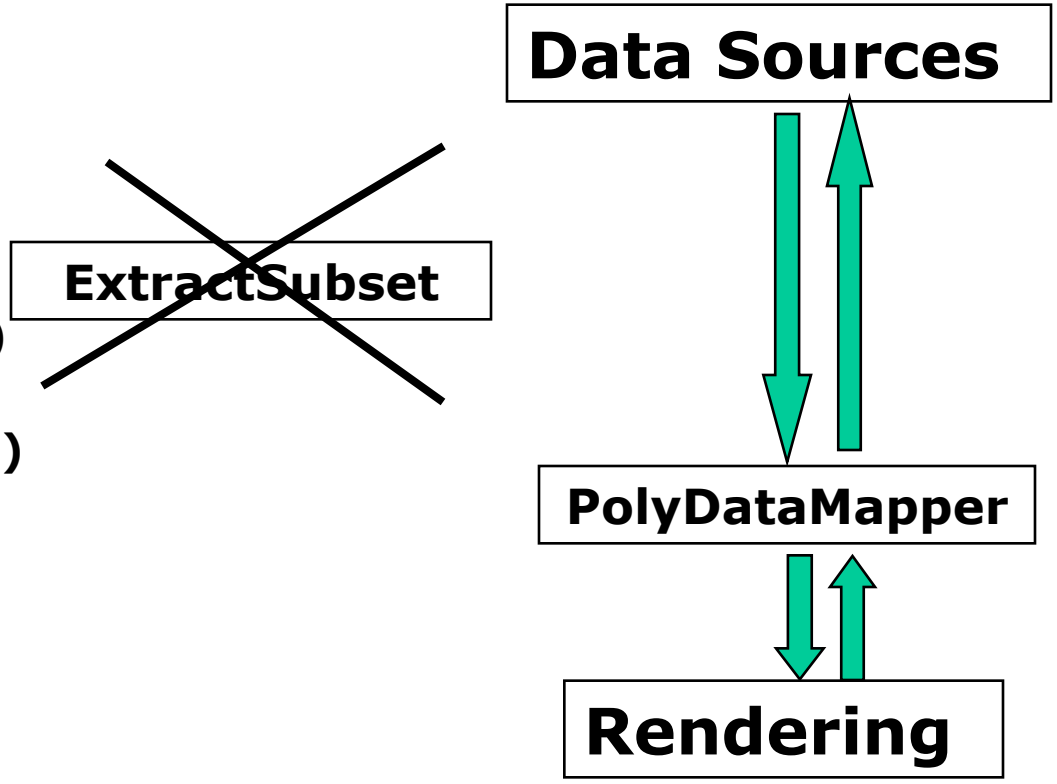


Streaming the data

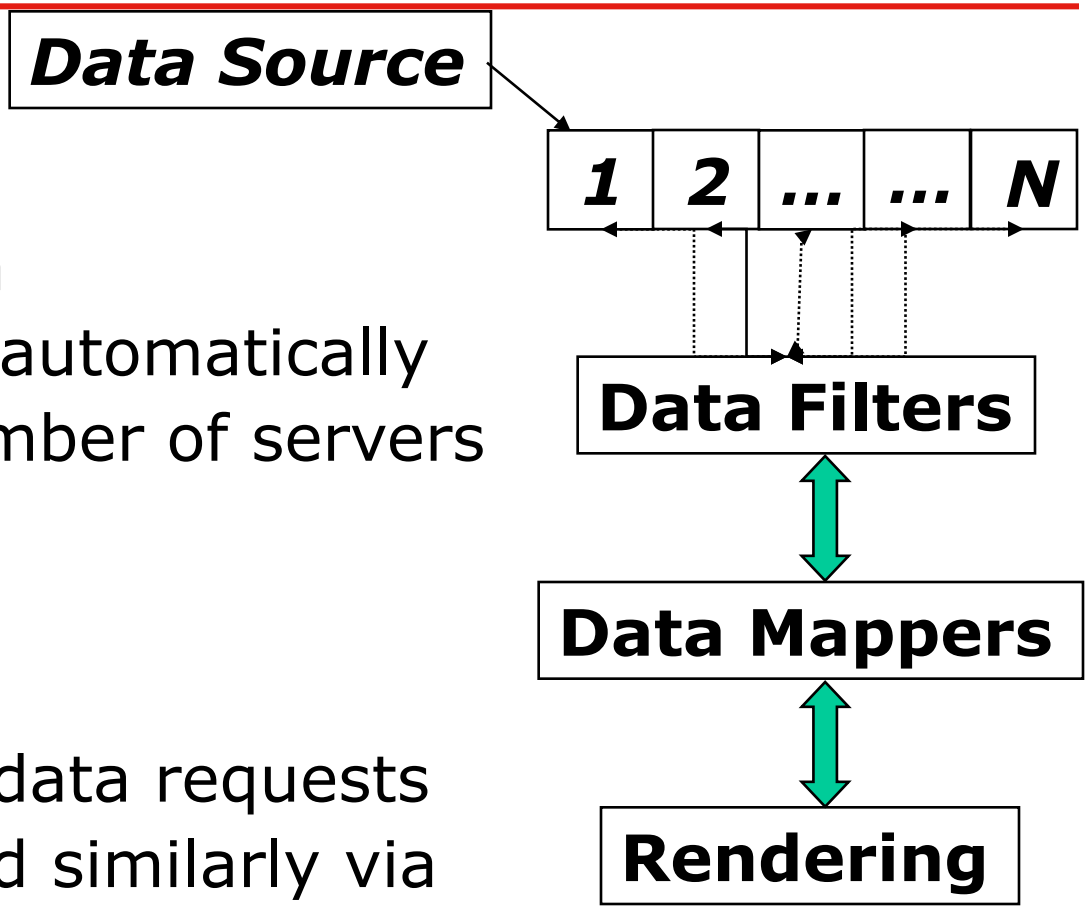
```
mapper = vtkPolyDataMapper()

mapper.SetNumberOfPieces(64)

mapper.SetPiece(0)
```



The Vis pipeline is "under the hood"



Data Parallelism

- data are divided automatically based on the number of servers available

Transient Data

- time-dependent data requests are also managed similarly via the two-way pipeline data exchange

Summary

- VisIt and ParaView hide all the pipeline management
- Meta-data are paramount to let the pipeline schedule the most efficient processing

The real questions are:

Can you provide data that can be distributed?

Is the distribution "piece invariant"?

Data formats, parallel I/O and visualization

Prelude

Data formats

- Interface between simulations and visualization
- Many formats exist. Pick the most appropriate
- High level libraries (HDF5, netCDF, …)
- Make up your own
- Parallel I/O

Data formats

Purpose of I/O

- Archive results to file(s)
- Provide check-point / restart files
- Analysis
- Visualization
- Debugging simulations

Requirements

- Fast, parallel, selective
- Independent off # of processors
- Self-documented

Data formats

- **Community specific**
  - CGNS, CCSM, NEK5000, H5Part
- **Ad-hoc**
  - Make up your own. No!
  - Many formats exist. Choose the most appropriate
  - High level libraries (HDF5, netCDF, …)

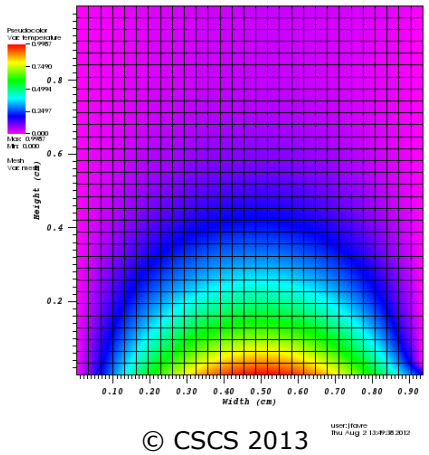
Data formats and Parallelism

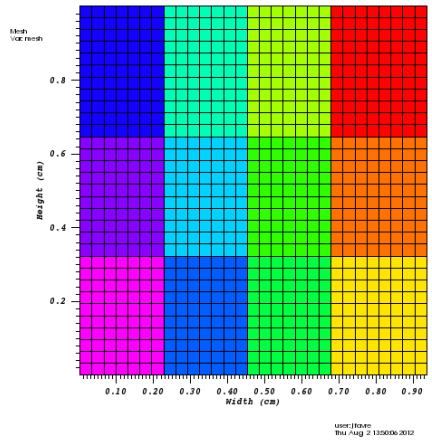
- **MPI-IO**
  - Raw data parallelism
  - The BOV format can be read by VisIt and ParaView
- **ADIOS**
  - Raw data but complexity is hidden
- **HDF5, NetCDF**
  - content-discovery is possible, but semantic is left-as-an-exercise.
- **SILO**
  - Poor man's parallelism (1 file per process + metafile) but strong semantic

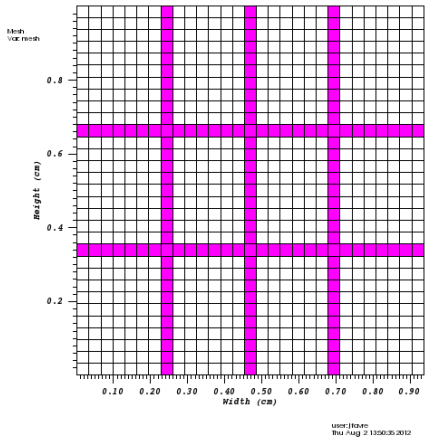
MPI tasks, ghost-cells, hyperslabs

- **Grids are sub-divided with ghost regions**
- **Ghost cells/nodes are usually not archived**
- **The User (You) is responsible for managing the subdivisions and know what to archive**

Example: a 12-processor run

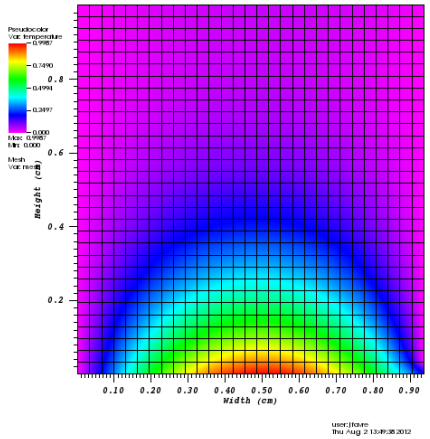


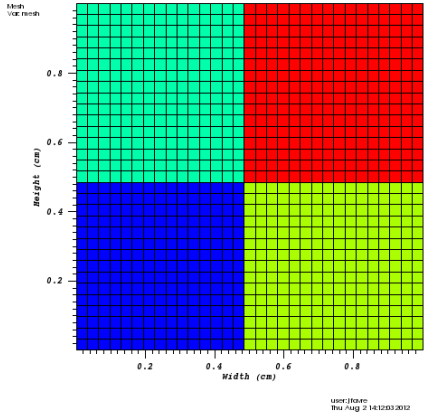


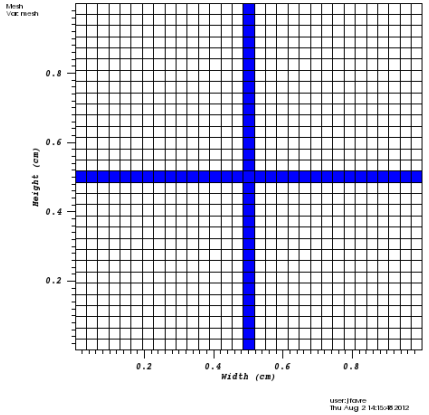


MPI tasks, ghost-cells, hyperslabs

Example: a 4-processor run

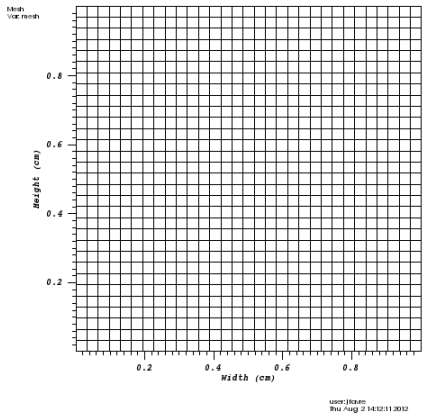






MPI tasks, ghost-cells, hyperslabs

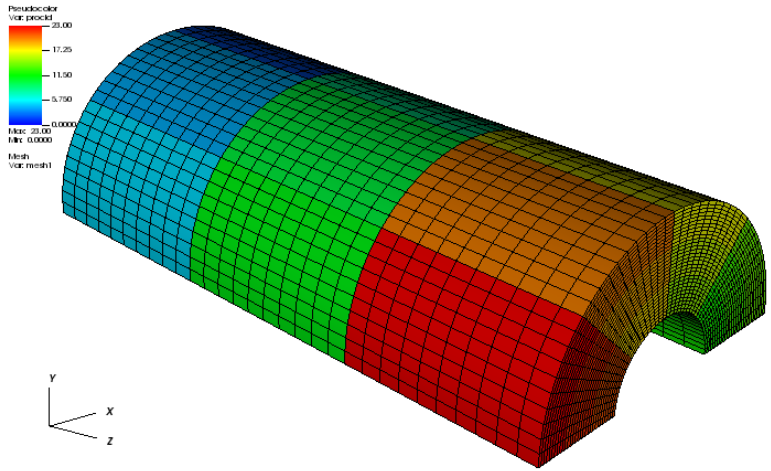
The goal of (parallel) I/O:
Present a uniform grid storage/display

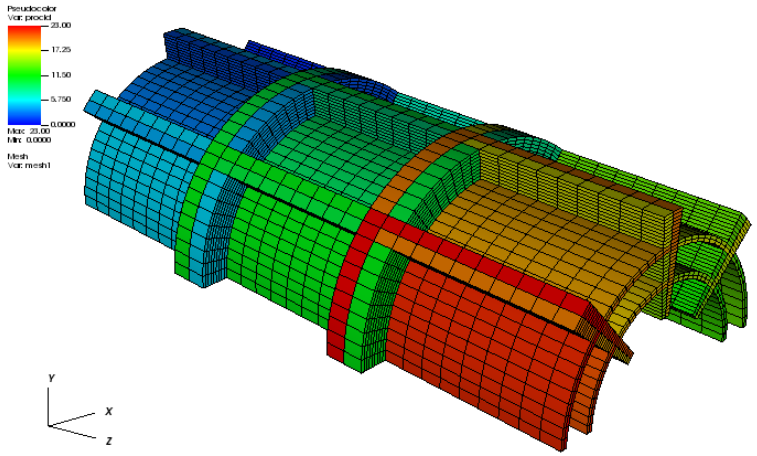


The Visualization software (ParaView or VisIt) will do its own subdivision and re-construct ghost-zones – when necessary

MPI tasks, ghost-cells, hyperslabs

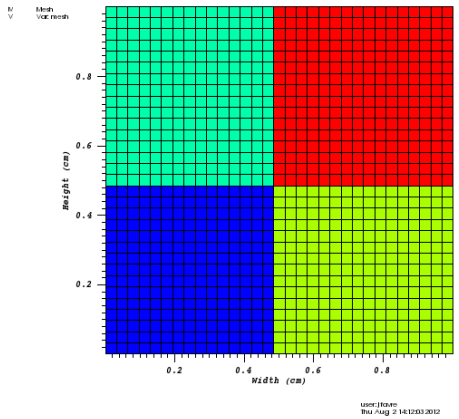
The Visualization software should know how to re-construct ghost-zones – when necessary



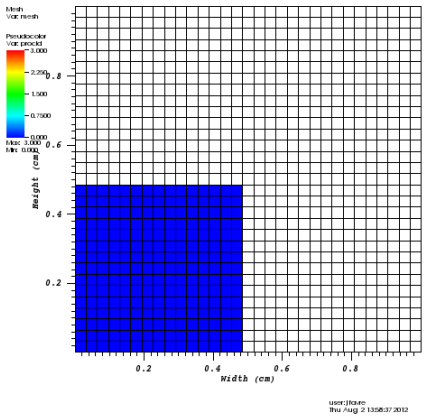


MPI tasks, ghost-cells, hyperslabs

Def: a hyperslab, is a subset in n-D of a larger grid. Parallel I/O is a composition (superposition) of multiple hyperslabs.



Each processor must know where each piece fits in the global mesh



Data formats. Parallelism

- **Once you know the IJK extents of all your hyperslabs, you can use**
  - MPI-IO, or
  - HDF5, or
  - NetCDF, or
  - ADIOS

SILO Data Format

- https://wci.llnl.gov/codes/silo

- **A very versatile data format. The "Getting Data Into VisIt" manual covers how to create files of this type. In addition, there are many code examples here**

- http://portal.nersc.gov/svn/visit/trunk/src/tools/DataManualExamples/CreatingCompatible

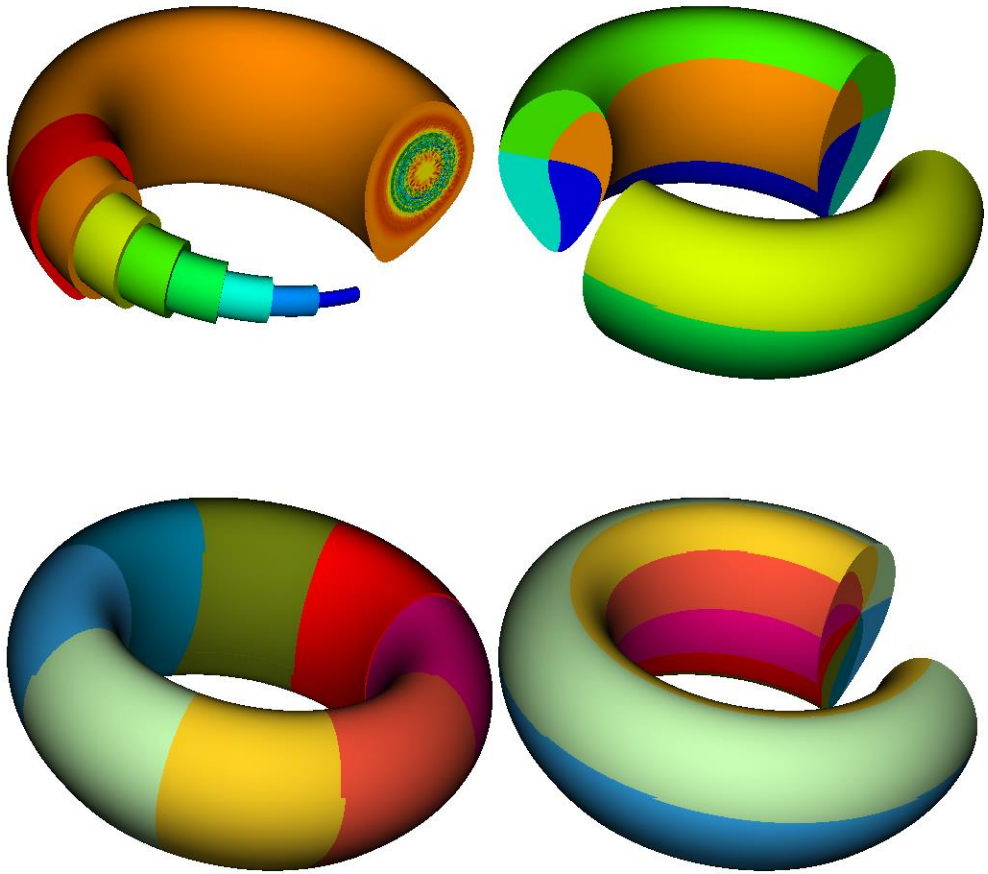
SILO Data Format

From the User Manual:

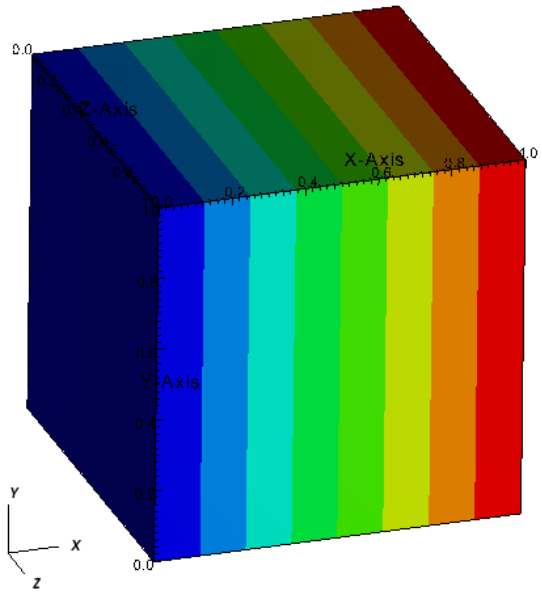
- **Silo is a serial library. Nevertheless, it (as well as the tools that use it like VisIt) has several features that enable its effective use in parallel with excellent scaling behavior.**

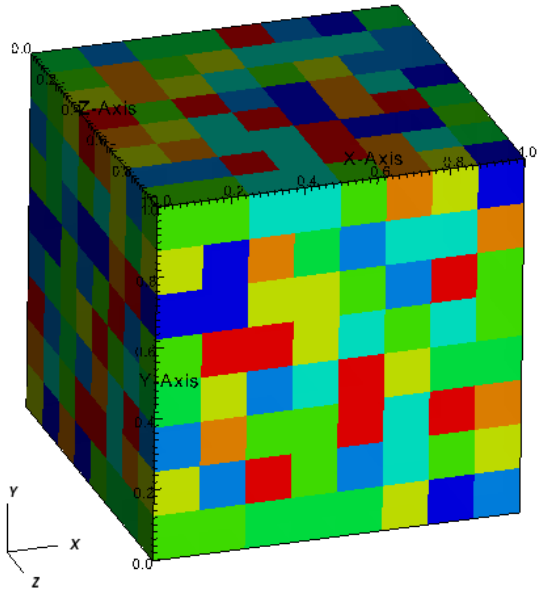
Pixie (HDF5) Data Format

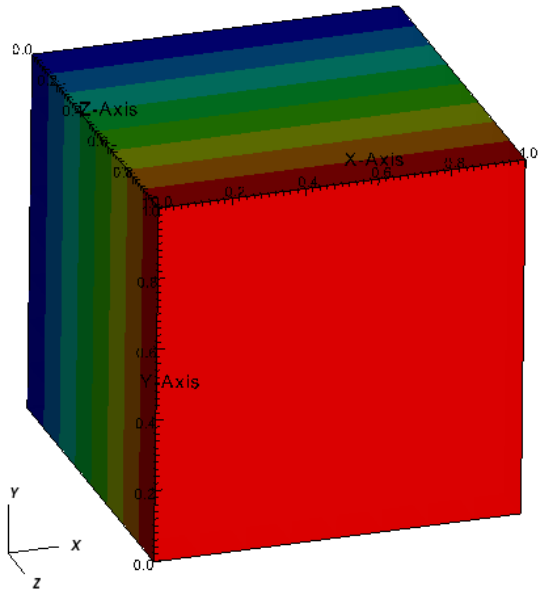
Modes:
radial, toroidal, poloidal, kd-tree sub-divisions



Data Parallelism by example: BOV format







Read a single block in a single file, but split the block in pieces

Cube dimension = 640x640x640

Bricklets = 80x80x80

Divide_brick = true

Modes: stride = 8, random, block

Data Parallelism by example: BOV format

Alternatively, each process writes its own file independently,
A 64x64x64x4 block of floats

```
1048576 2013-11-25 14:04 benchmark.00000.0004.bof
1048576 2013-11-25 14:04 benchmark.00001.0004.bof
1048576 2013-11-25 14:04 benchmark.00002.0004.bof
1048576 2013-11-25 14:04 benchmark.00003.0004.bof
1048576 2013-11-25 14:04 benchmark.00004.0004.bof
1048576 2013-11-25 14:04 benchmark.00005.0004.bof
```

Each file can also be gzipped

```
22177 2013-11-25 14:04 benchmark.00000.0004.bof.gz
22094 2013-11-25 14:04 benchmark.00001.0004.bof.gz
22398 2013-11-25 14:04 benchmark.00002.0004.bof.gz
21958 2013-11-25 14:04 benchmark.00003.0004.bof.gz
21838 2013-11-25 14:04 benchmark.00004.0004.bof.gz
22220 2013-11-25 14:04 benchmark.00005.0004.bof.gz
  422 2013-11-25 14:26 benchmark.0004.bov
```

Data Parallelism by example: BOV format

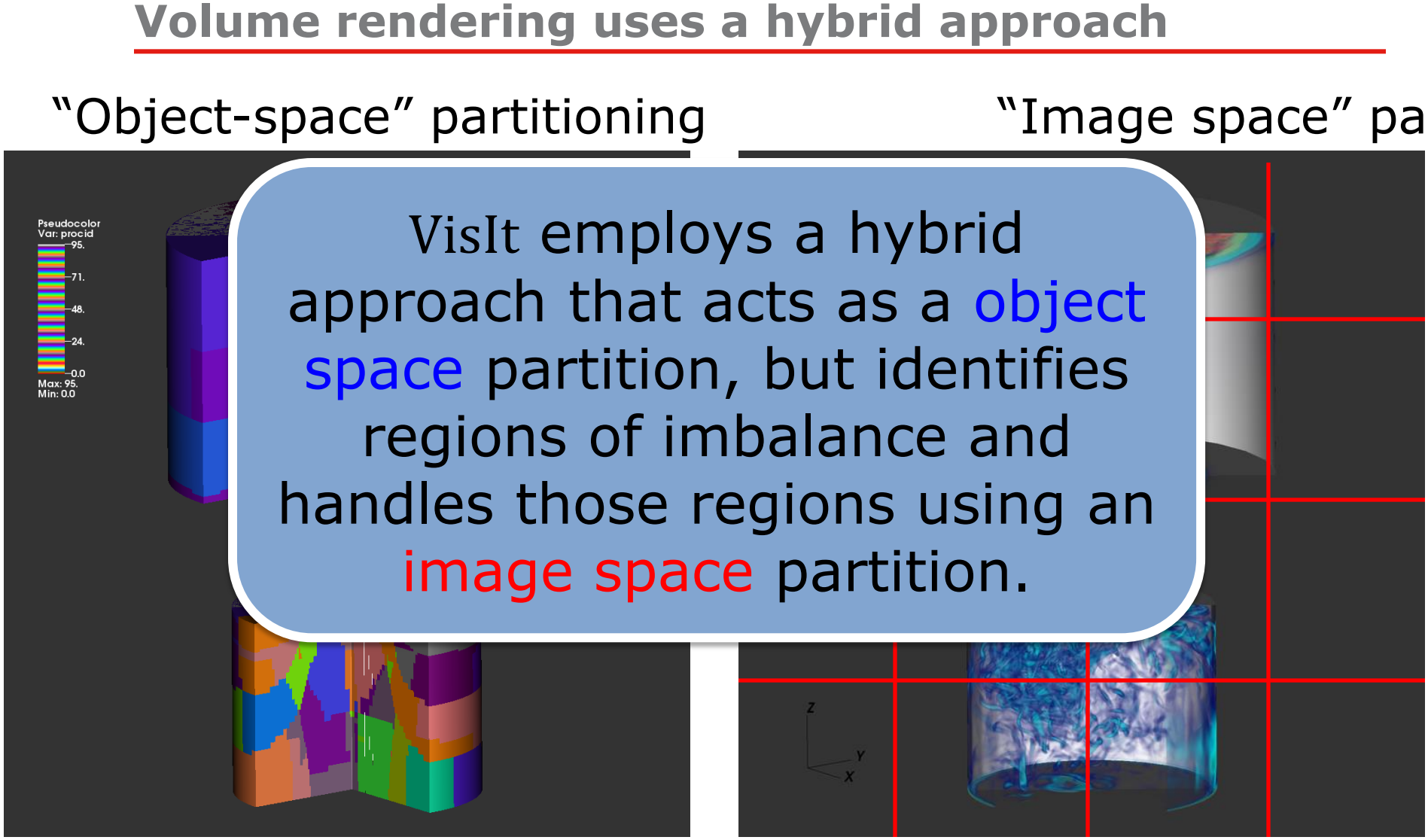
VisIt can put the pieces together (serially, or in parallel) with the following meta-file "benchmark.0004.bov"

```
# BOV version: 1.0
# I/O benchmark program
DATA_FILE: benchmark.%05d.04.bof.gz
DATA SIZE: 192 128 64
DATA_BRICKLETS: 64 64 64
DATA FORMAT: FLOAT
VARIABLE: node_data
BRICK ORIGIN: 0.0 0.0 0.0
BRICK SIZE: 3.0 2.0 1.0
```

Volume rendering uses a hybrid approach

"Object-space" partitioning

"Image space" par



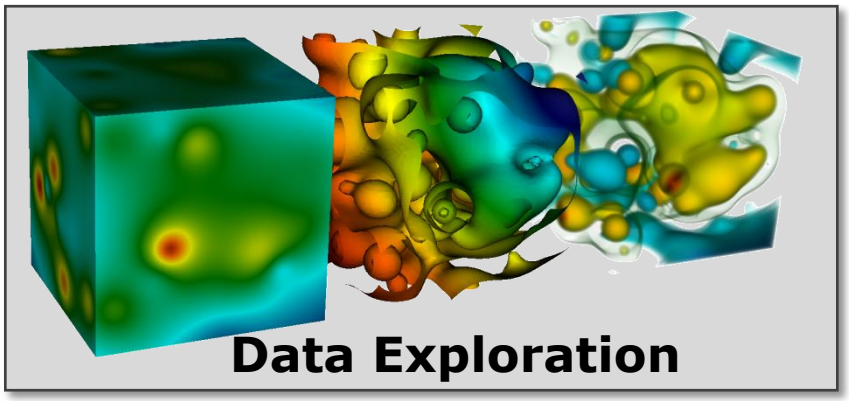
VisIt employs a hybrid approach that acts as a object space partition, but identifies regions of imbalance and handles those regions using an image space partition.

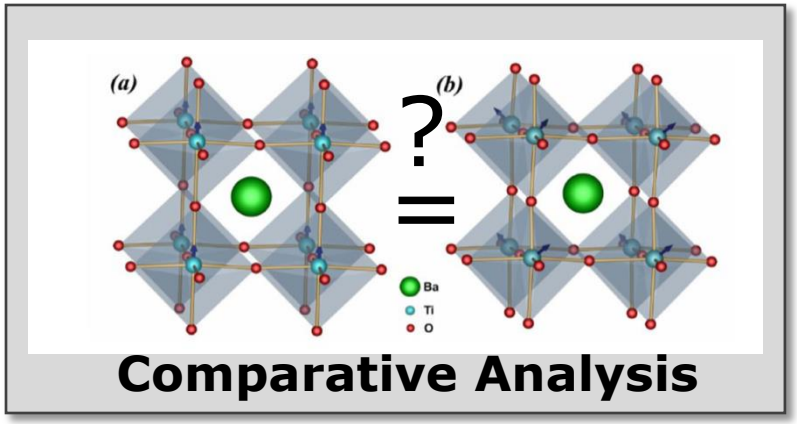
Scientific Visualization

- **Why visualization?**

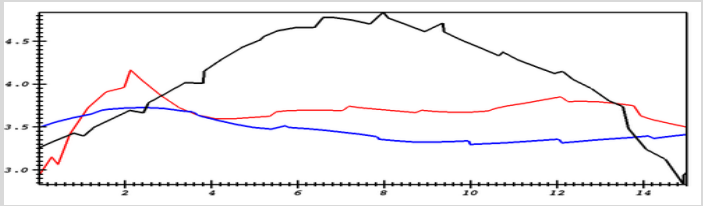
- **How to:**
  - **Remote Visualization**
  - **Client server**
  - **Parallel Visualization**
  - **In-situ Visualization**

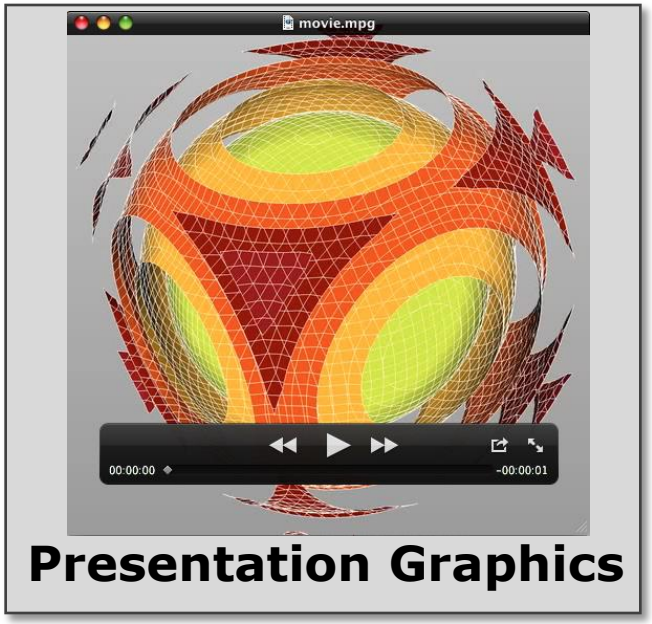
Visualization is many complementary things


Data Exploration

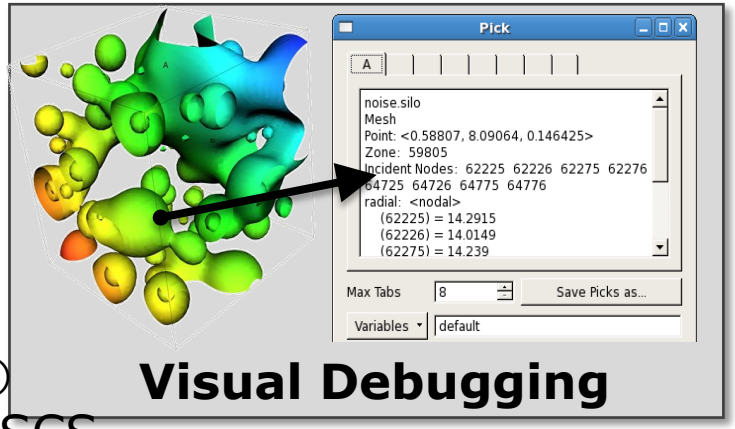


Comparative Analysis

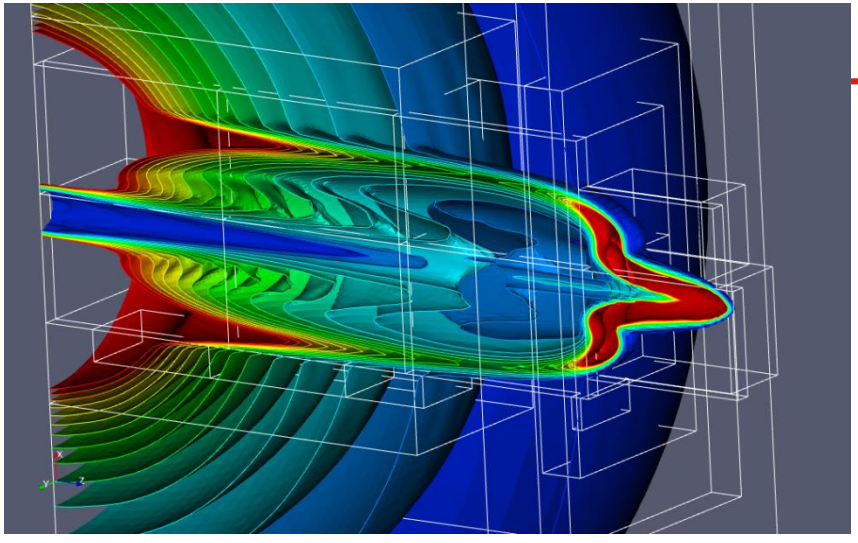

Quantitative Analysis

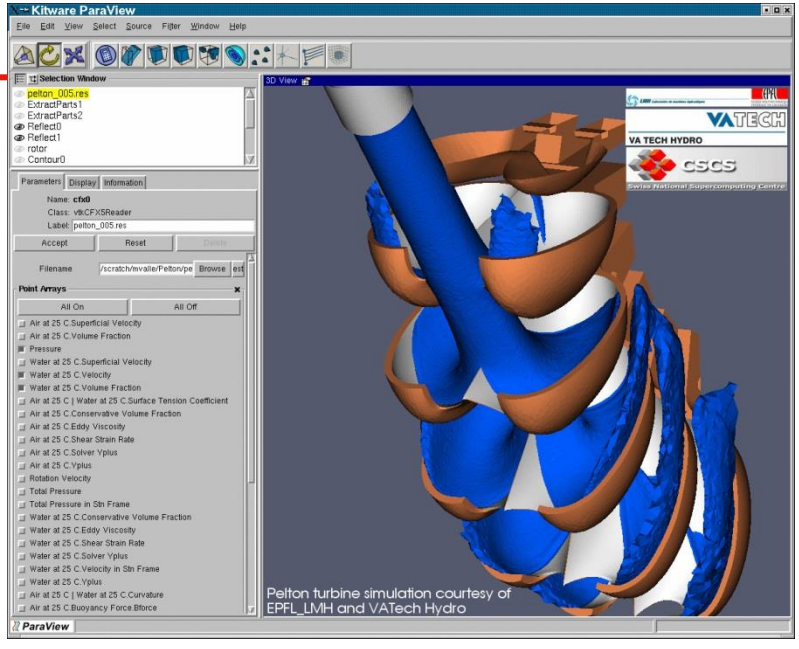


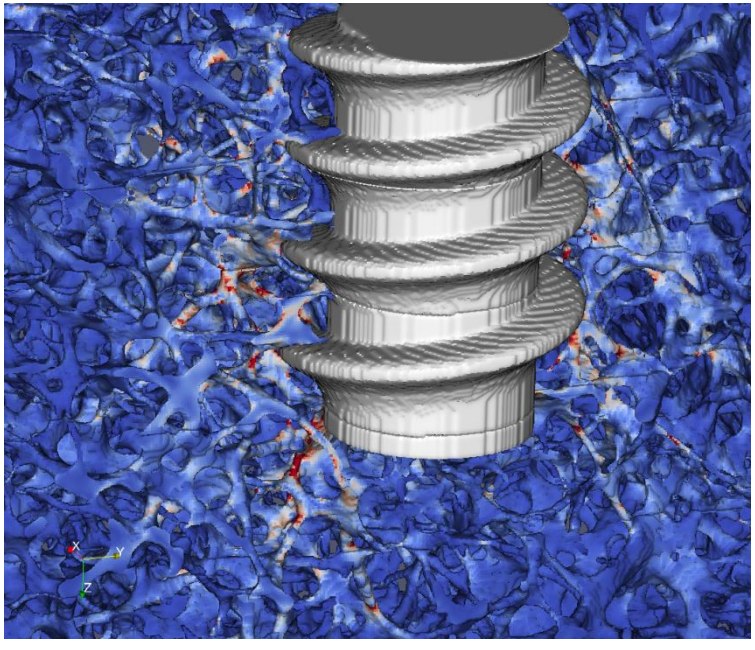
Presentation Graphics



Visual Debugging







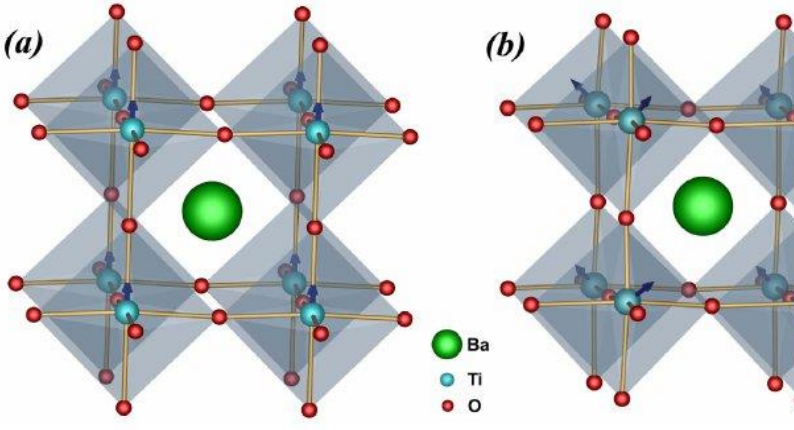


Figure 2. Displacement pattern of Ti atoms in tetragonal $BaTiO_3$. (a) Average pic
shiftings along the direction of tetragonal distortion. (b) Local, instantaneous Ti sh
directions close to the body diagonal. Ti displacements are represented as bold blu

Scientific Visualization: Two modes

- **Post-mortem**

This does not mean you can start thinking about it [The Visualization] after the simulations are done. You should plan it before running your code…

- **Live, a.k.a. in-situ visualization**

Simulation and visualization codes run at the same time, on a shared resource, or a distributed set of machines. An advanced topic, … (see demo)

Scientific Visualization

VisIt and ParaView support two modes of execution:

1. **Interactive imaging, analysis, query…**

Requires a GUI, to test and try multiple visual representations.

2. **A batch-oriented movie-making process**

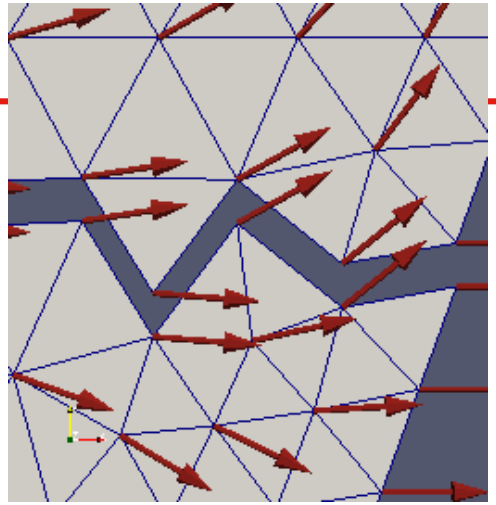
Requires a script (python), to enable reproducible visualizations, and the support of time series, or multiple experiments

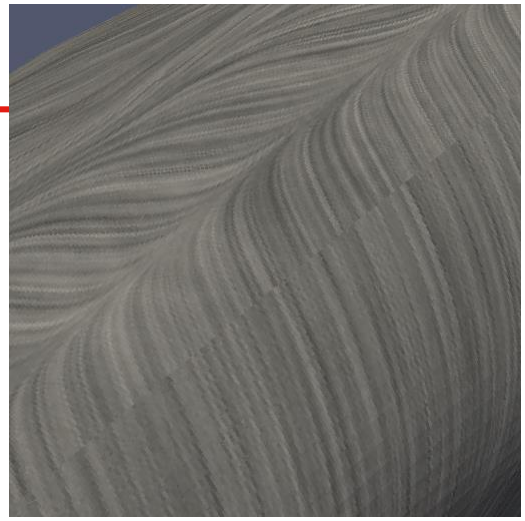
Visualization: Client Server

ParaView and VisIt use the client-server concept:

- **A client (optional) runs the GUI**
- **A server, embedded and local (by default), or remote and/or parallel, does the real work:**
  - I/O
  - Data analysis
  - Image generation

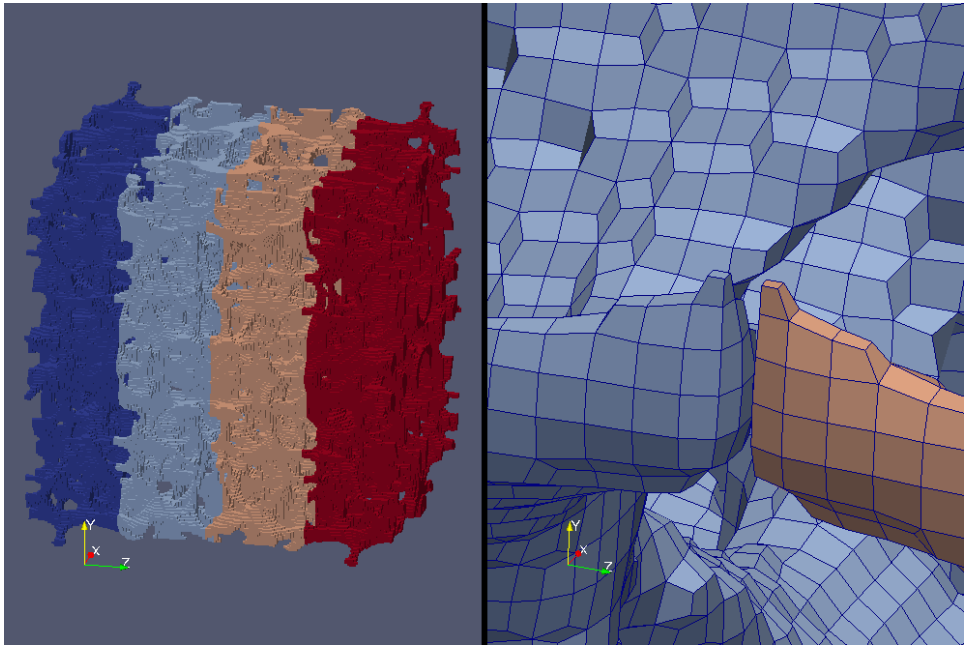
Parallel Visualization





Parallelism is a must for big data.

Parallelism is the source of many problems.



Parallel Visualization

Should we bother?

Yes!

Interactive visualization is necessary to gain insight from exploration.

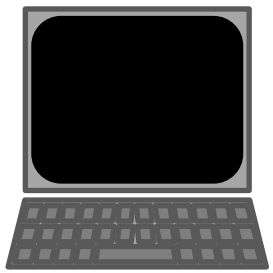
Yes!

Parameter tuning should be fast.

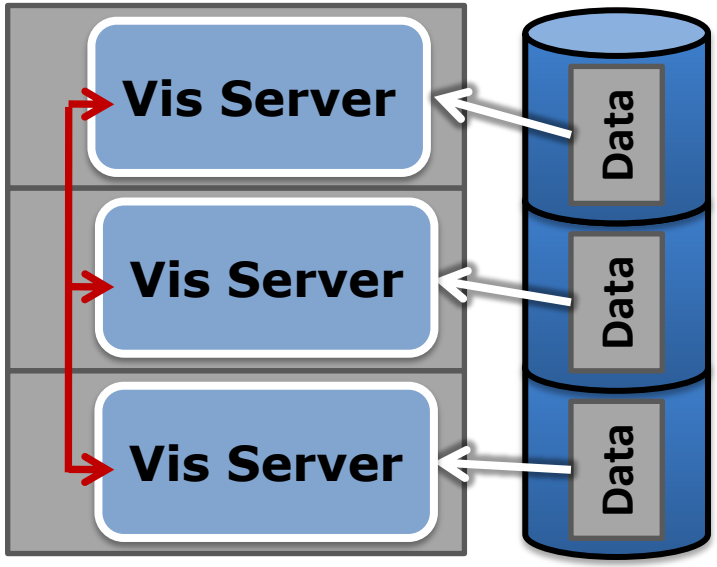
Client and Remote Servers: Direct connections

- **A client app can request a direct connection to a parallel visualization server (thru a firewall)**
- **ParaView and VisIt use ssh tunnels to establish connections**

Local (remote) Clients



Parallel server at CSCS



Supercomputer or graphics cluster?

VisIt Launcher

```
#SBATCH --nodes=12
#SBATCH --ntasks=96
#SBATCH --gres=gpu:2
#SBATCH --exclusive

mpirun -np 96 engine_par
-sshtunneling
-hw-accel  -display :0.%l
-host castor0
-port 15129 -key 709adcfbf2
```

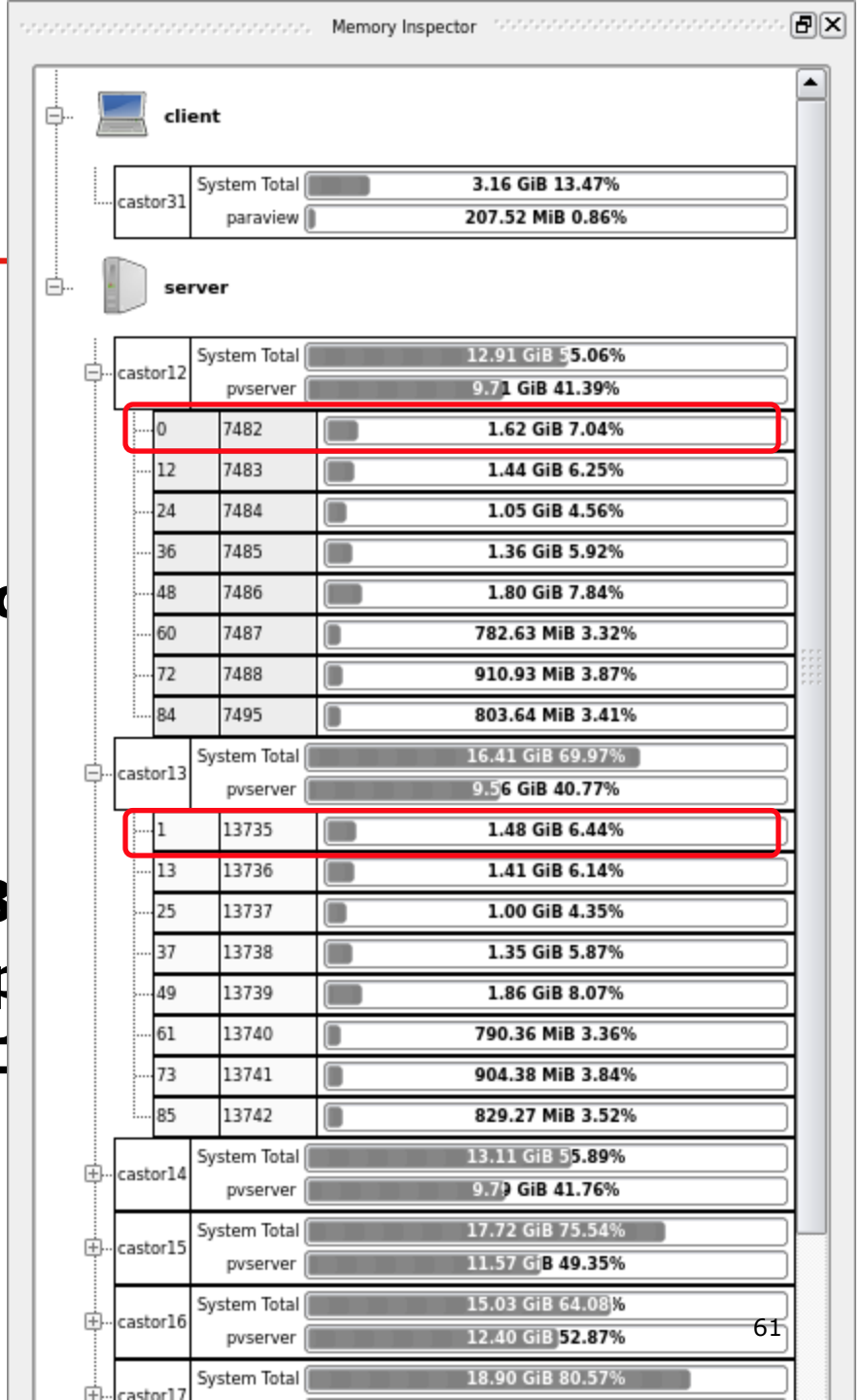
```
00: Creating (HW-based) display
01: Creating (HW-based) display
02: Creating Mesa (SW-based) display
03: Creating Mesa (SW-based) display
04: Creating Mesa (SW-based) display
05: Creating Mesa (SW-based) display
06: Creating Mesa (SW-based) display
07: Creating Mesa (SW-based) display
08: Creating (HW-based) display
09: Creating (HW-based) display
10: Creating Mesa (SW-based) display
11: Creating Mesa (SW-based) display.
12: Creating Mesa (SW-based) display
13: Creating Mesa (SW-based) display
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ParaView Launcher

```
#SBATCH --nodes=12
#SBATCH --ntasks=96
#SBATCH --gres=gpu:2
#SBATCH --distribution=c
#SBATCH --exclusive

mpiexec -binding rr -ppn
:0.0 pvserver -rc -ch=148
-n 48 -env DISPLAY :0.1 p
ch=148.187.19.45 -sp=1
```



From the Supercomputing'13 Showcase

No interactivity for 150 millions cells

HDF5 Reader

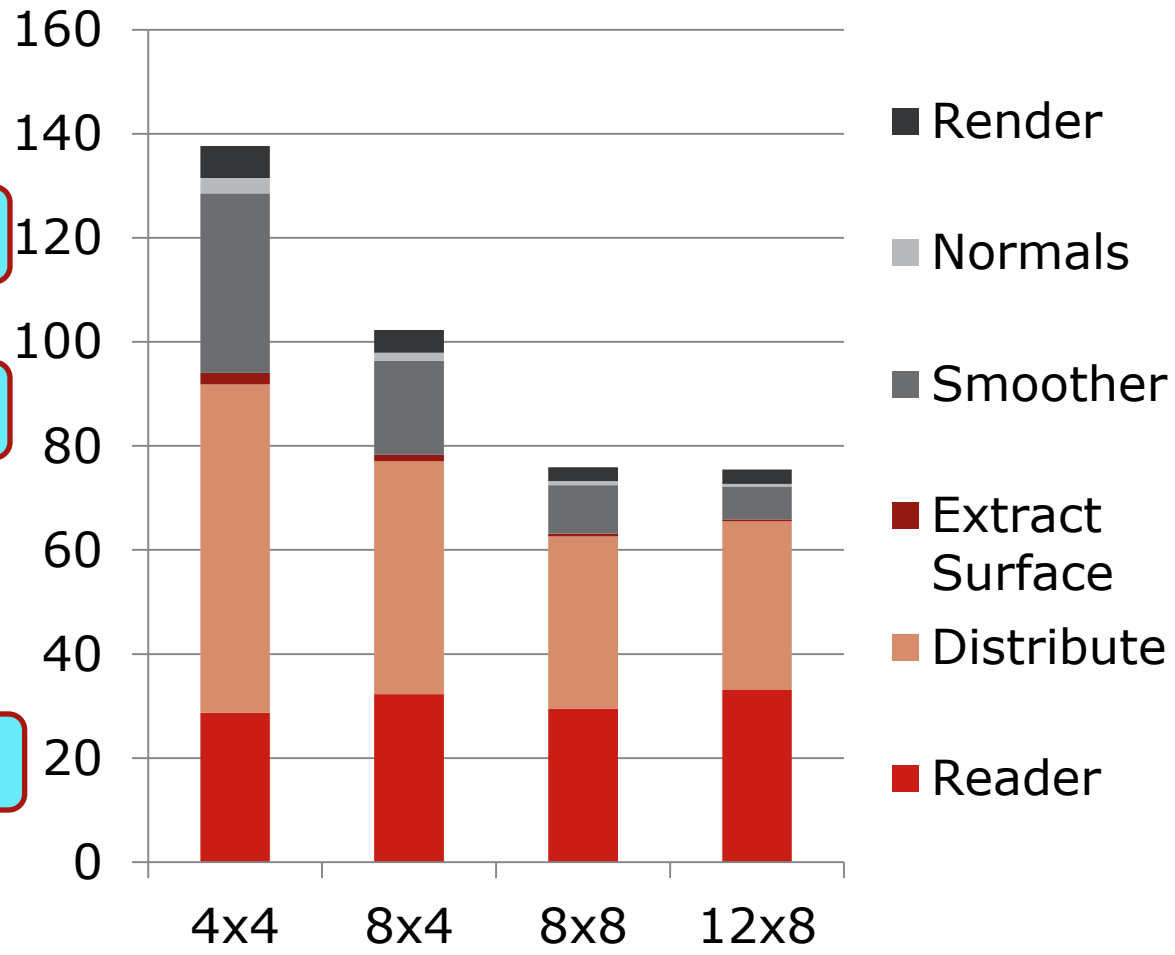
Distribute Data

Extract Surface

Smoother

Generate Normals

Still Render



Visualization without a client

ParaView and VisIt can run a server-only job

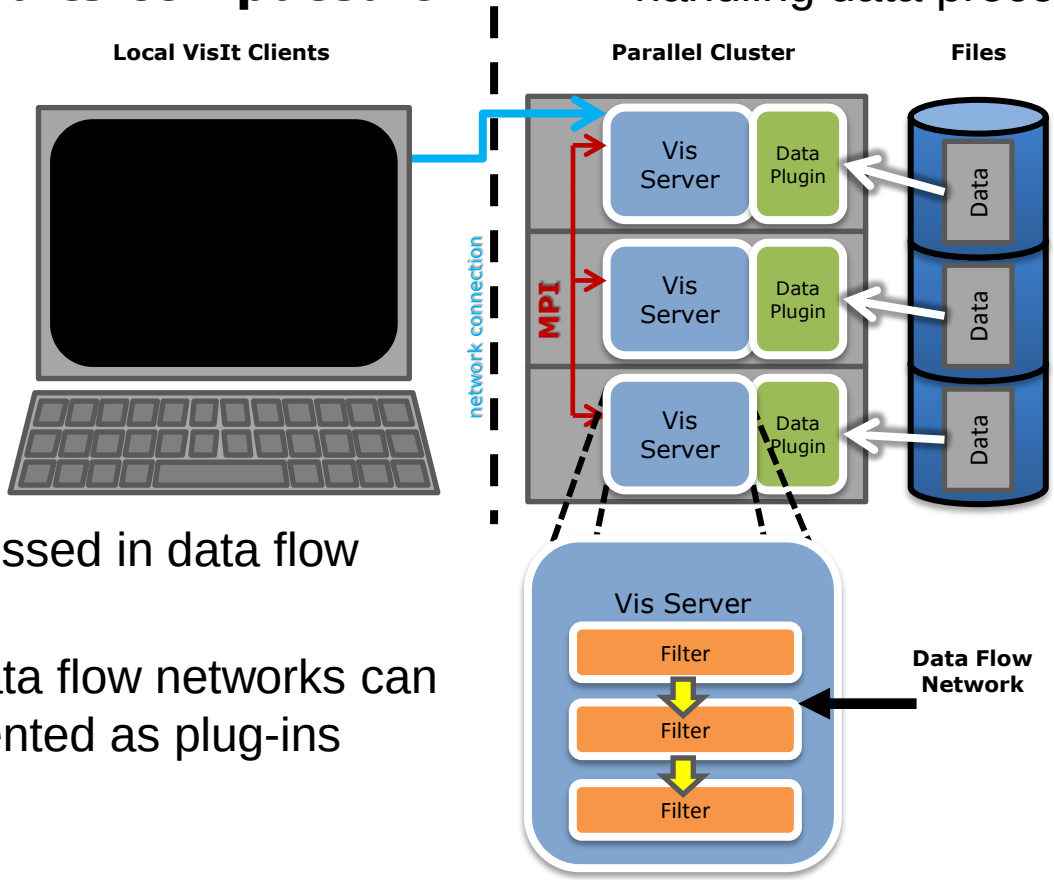
- **Ideal for batch-processing**
- **A python script is necessary**

- **While creating a new visualization (with the client), one can save the corresponding python commands to construct the pipelines**

- **Python programs do not include any explicit parallel programming! (that's easy!)**

In-situ Visualization

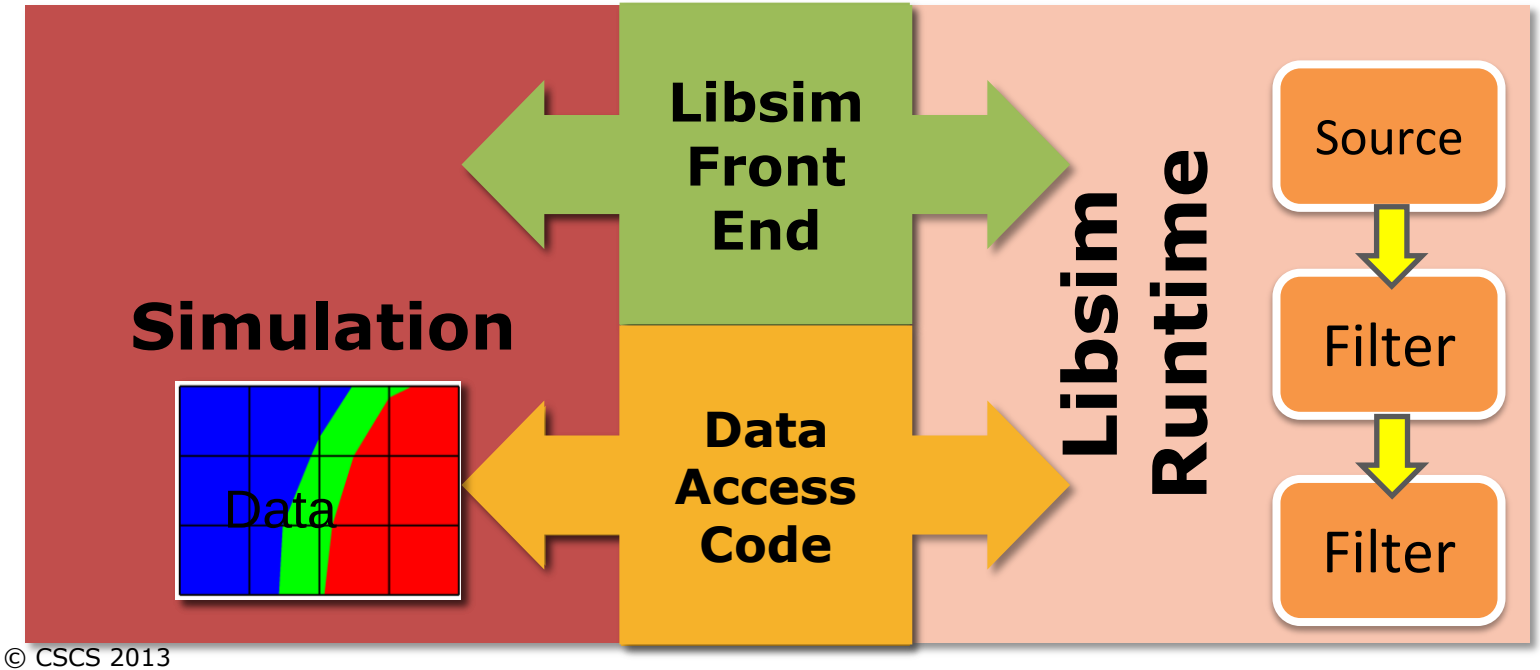
- **Clients runs locally and display results computed on the server**
  - Server runs remotely in parallel, handling data processing for client



- Data processed in data flow networks
- Filters in data flow networks can be implemented as plug-ins

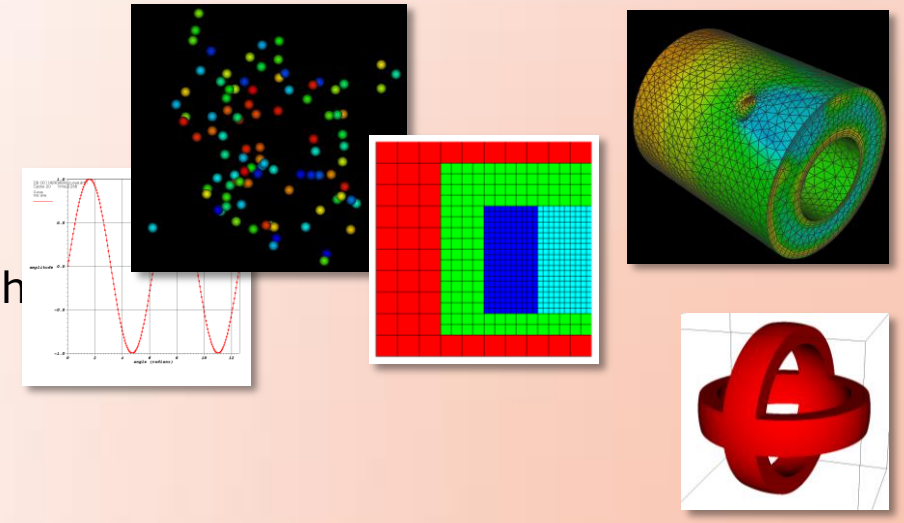
Coupling of Simulations and VisIt

Libsim is a VisIt library that simulations use to enable couplings between simulations and VisIt. Not a special package. It is part of VisIt.

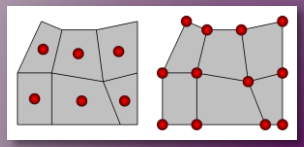


Data model for in-situ visualization

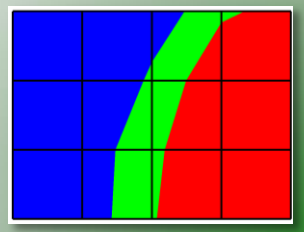
- **Mesh Types**
- Structured meshes
- Point meshes
- CSG meshes
- AMR meshes
- Unstructured & Polyhedral mesh



- Variables
  - 1 to N components
  - Zonal and Nodal



- Materials
- Species



© CSCS 2013

Summary

- **Visit and ParaView support visualization with a focus on large data typically output by simulations in HPC.**
- **Remote, client-server, parallel, interactive and batch oriented executions are used daily.**
- **A data format which supports distributed access is essential.**

VisIt and ParaView tutorials on-line

We will now follow with several demonstrations of the VisIt and ParaView applications

http://visitusers.org/index.php?title=VisIt_Tutorial

http://www.paraview.org/Wiki/images/5/5d/ParaViewTutorial41.pdf

Thank you for your attention.