CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Visualisation scientifique parallèle de gros volumes de données

December 2017

Jean M. Favre, CSCS

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Presentation of CSCS

CSCS, home of “Piz Daint”, the Swiss flagship for national HPC Service

- Cray XC40/XC50
- #3 of the Top500 list in november 2017
- 5320 hybrid nodes (Intel Xeon E5-2690 v3/Nvidia Tesla P100)
- 1788 multi-core nodes (Intel Xeon E5-2695 v4)
- Piz_daint_one_of_the_most_powerful_supercomputers_in_the_world

Outline of the presentation

Best practices in parallel visualization

- Parallelization on the node (SMP)
- Understanding, and fine-tuning the I/O. Managing the pipeline.
- MPI-based parallelization. The do’s and dont’s
- Parallel Rendering libraries
- *in-situ* visualization
- Conclusion

Footnote

Mes commentaires et divagations sur le thème sont le fruit de longues années passées a observer les méthodes utilisées par de nombreux utilisateurs du CSCS et a ma passion pour promouvoir une utilisation sensée et la plus efficace possible de deux solutions de visualisation open-source,

ParaView: www.paraview.com

VisIt: https://wci.llnl.gov/simulation/computer-codes/visit

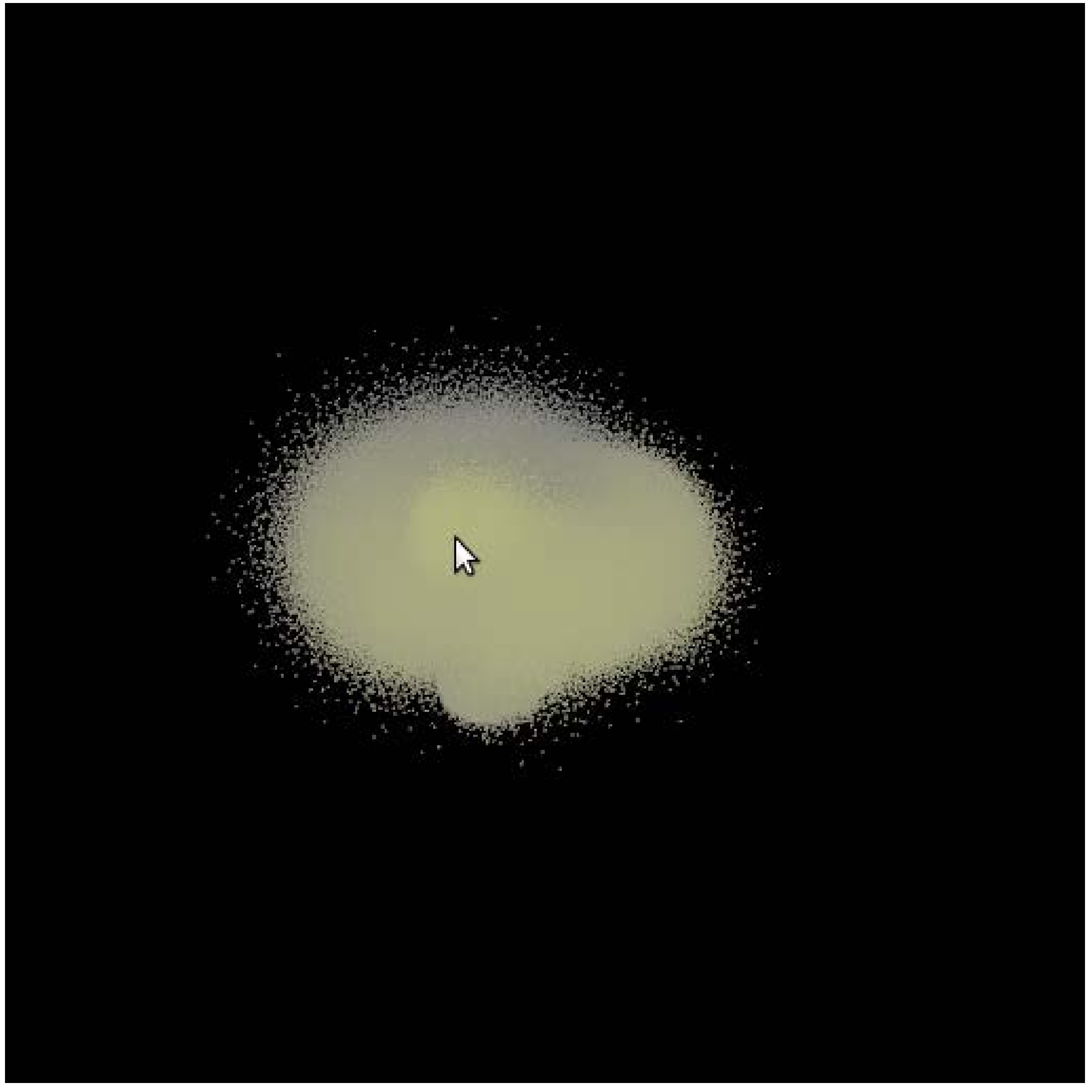
Parallel visualization is managed in two ways

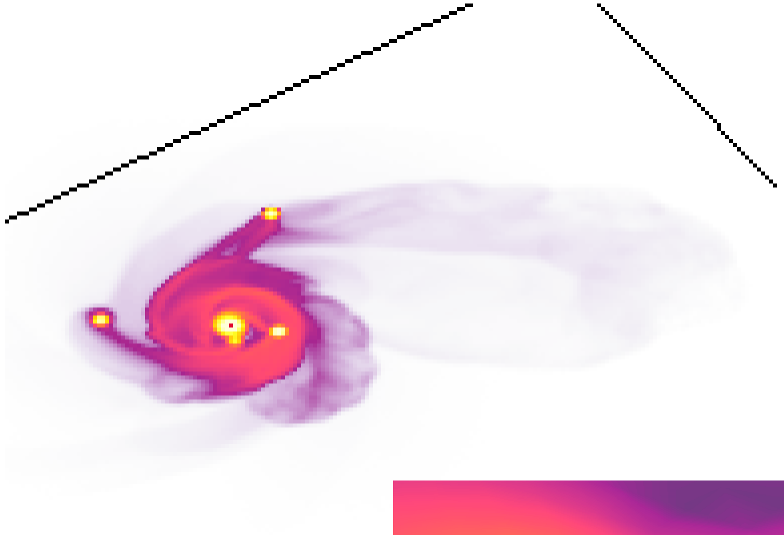
- Client-server and batch mode execution of MPI-based data filtering and rendering engines are a must for big data.
  - How big is “big”?

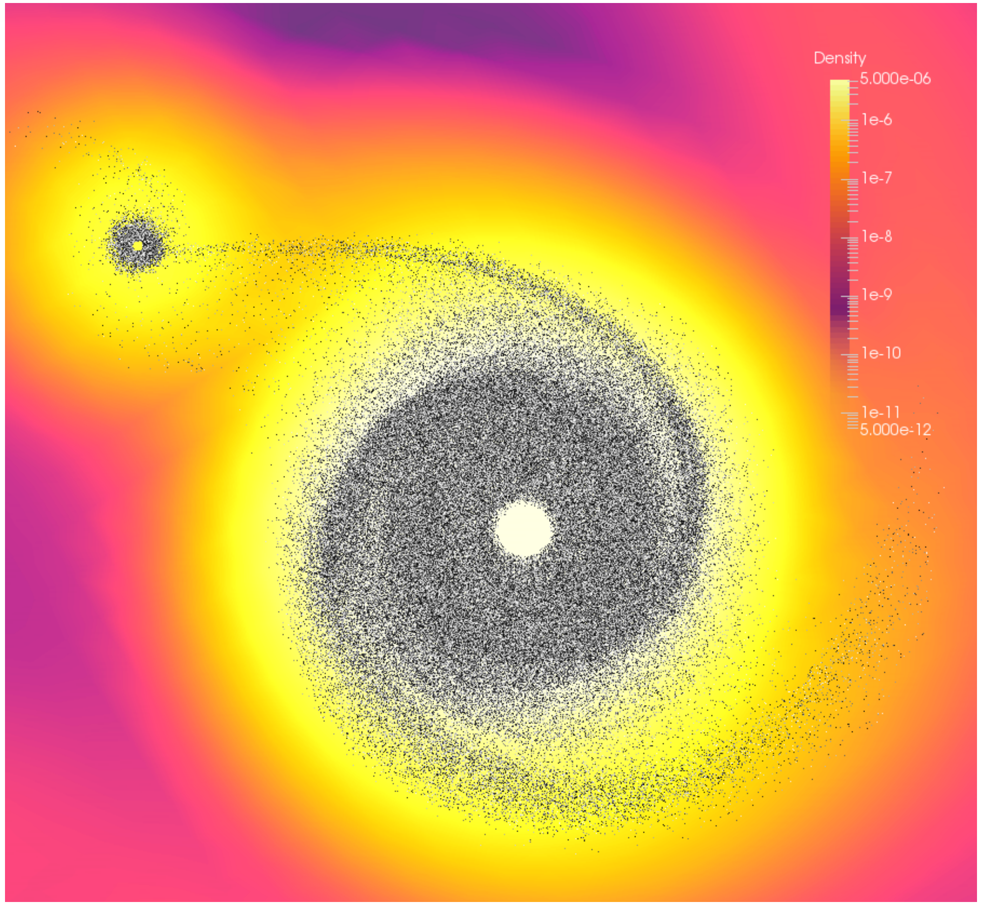
- Yet, many users still use the desktop version of VisIt or ParaView. A quick review of on-the-node parallelism is of actuality.

SMP parallelism

SPH Particle clouds to slices







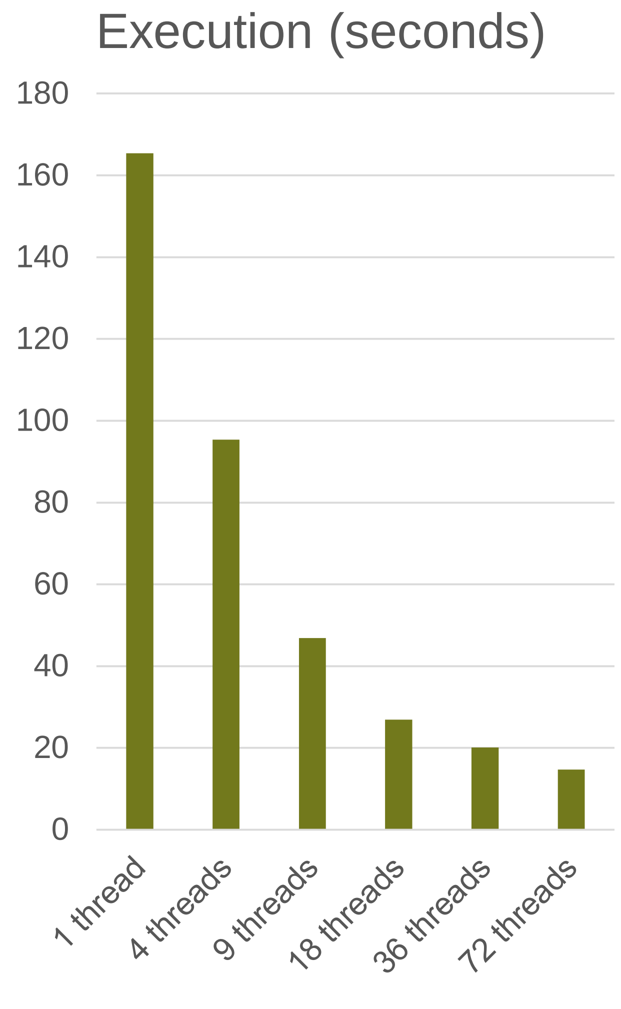
SPH data interpolation

- The vtkSPHInterpolator filter uses SPH (smooth particle hydrodynamics) kernels to interpolate a data source onto an input structure.

- A Point locator is a crucial part of the execution path, to accelerate queries about points and their neighbors.

- The execution of a plane interpolation has been tested on a multi-core node, using parallelism-on-the-node with Intel TBB.

- Compute node: dual socket Intel® Xeon® E5-2695 v4 @ 2.10GHz (18 cores)



VTK’s SMPTools

Shared memory parallel algorithms were kick-started in 2013

vtkDepthImageToPointCloud, vtkShepardMethod, vtkGaussianSplatter, vtkCheckerboardSplatter, vtkImageHistogram, vtkImageDifference, vtkStatisticalOutlierRemoval, vtkPointOccupancyFilter,

vtkVoxelGrid, vtkExtractHierarchicalBins, vtkPCACurvatureEstimation, vtkSPHInterpolator, vtkRadiusOutlierRemoval, vtkPointDensityFilter, vtkSignedDistance, vtkExtractPoints, vtkFitImplicitFunction, vtkUnsignedDistance, vtkPCANormalEstimation, vtkPointInterpolator, vtkPointCloudFilter, vtkHierarchicalBinningFilter, vtkMaskPointsFilter, vtkPointInterpolator2D, vtkExtractSurface, vtkDensifyPointCloudFilter, vtkFlyingEdgesPlaneCutter, vtkSimpleElevationFilter, vtkVectorDot, vtkFlyingEdges3D, vtkPlaneCutter, vtkVectorNorm, vtkFlyingEdges2D, vtkElevationFilter, vtkSampleImplicitFunctionFilter, vtkSortDataArray, vtkSortFieldData, vtkStaticPointLocator, vtkStaticCellLocator

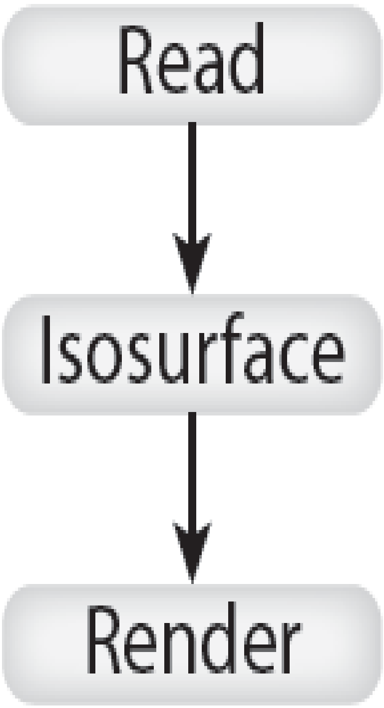
As of VTK-8, VTK-m is now opening to multiple back-ends and different implementations (CUDA,)

A side-note on visualization pipelines

The Visualization Pipeline

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 threee types of modules: *sources*, *filters* and *sinks*.»

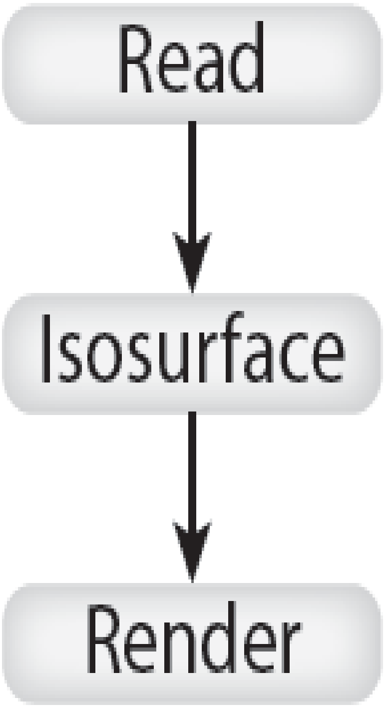


Visualization Pipeline: Definitions

- Modules are functional units, with 0 or more inputs ports and 0 or more output ports.

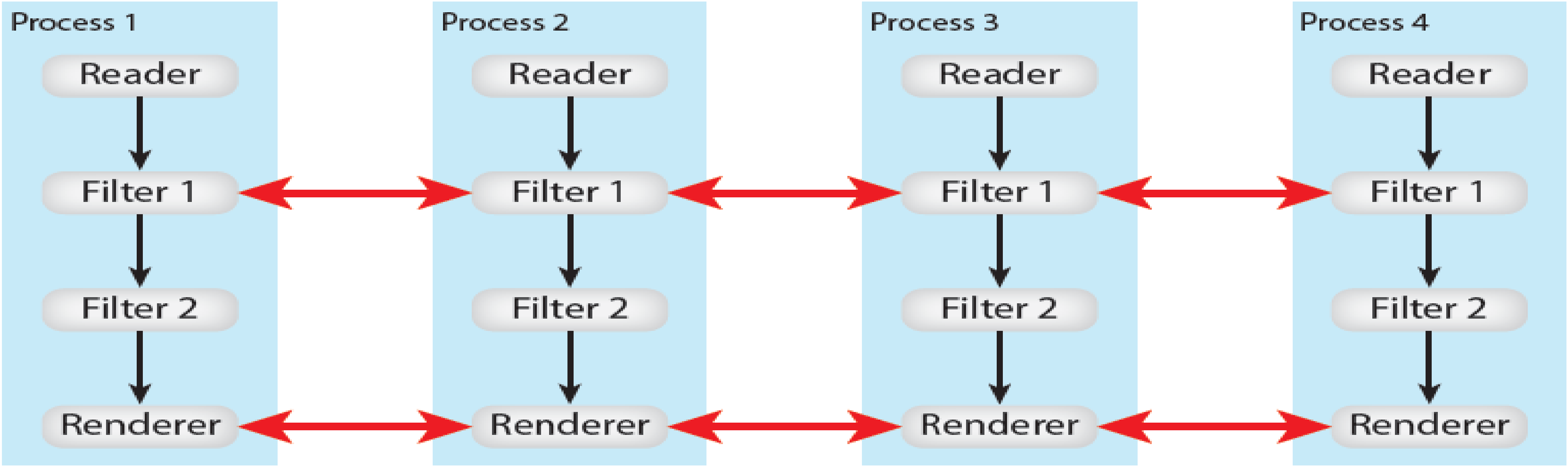
- Connections are directional attachments between input and output ports.

- Execution management is inherent in the pipeline
  - Event-driven
  - Demand-driven



Visualization Pipeline: 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. GhostCellGenerator)



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

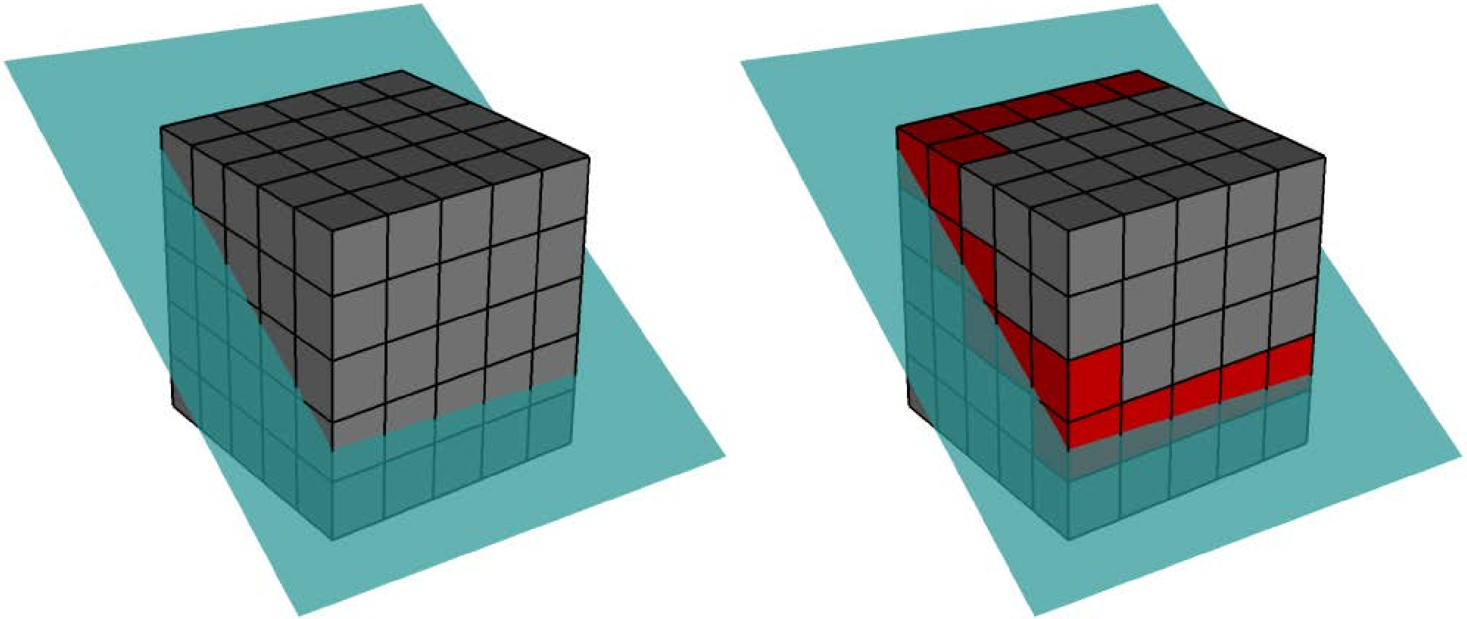
Understanding, and fine-tuning the I/O

The parallel I/O before the visualization

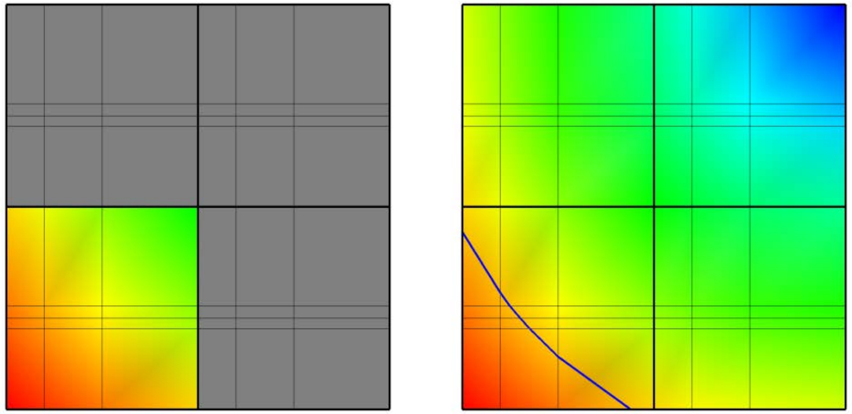
- Did you chose your data format?
- Did you chose the visualization application?
- Do you know how the file(s) is being read? Distributed among parallel tasks?

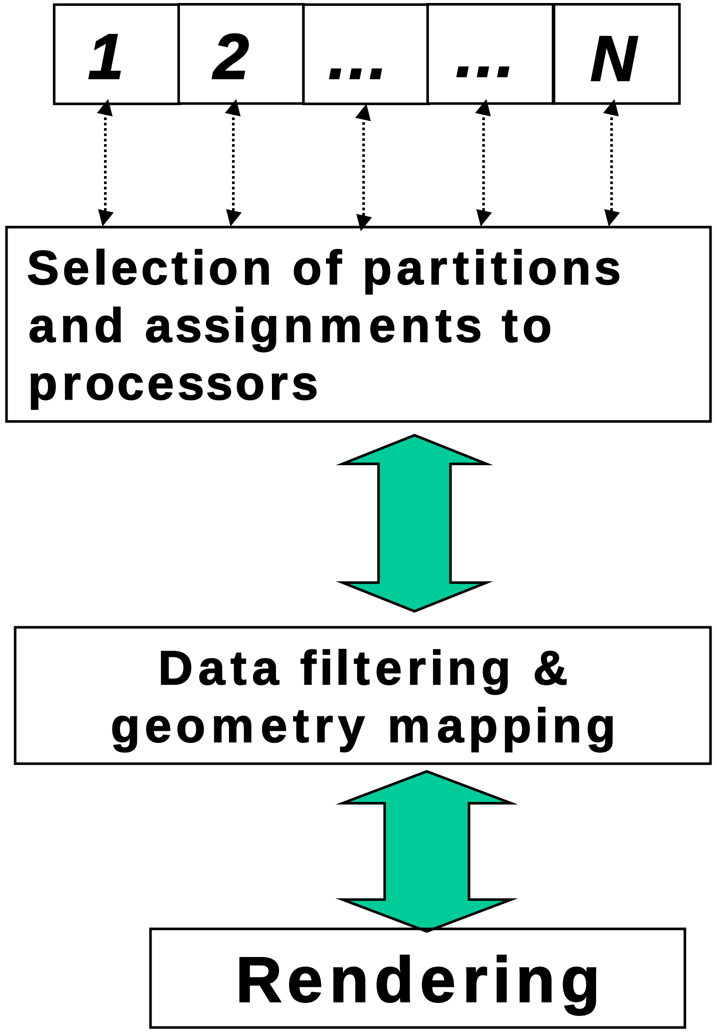
Does your file format enable *load-on-demand*?

- VisIt will inquire about spatial extents, and if available, the visualization pipeline is by-passed for the extents outside the range



Does your file format enable *load-on-demand*?



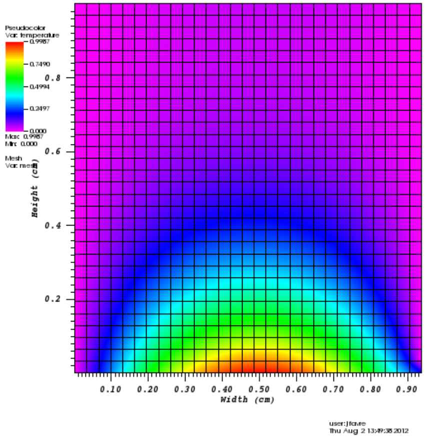


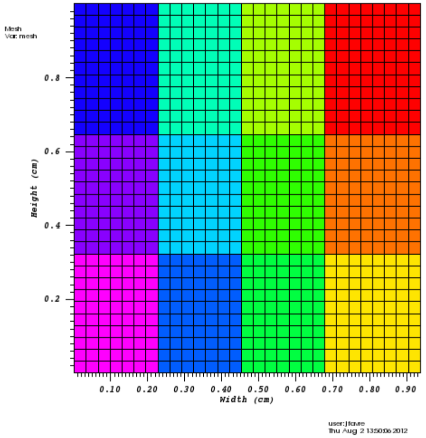
Data extents (min & max) are examined and the visualization pipeline is by-passed for those outside the range

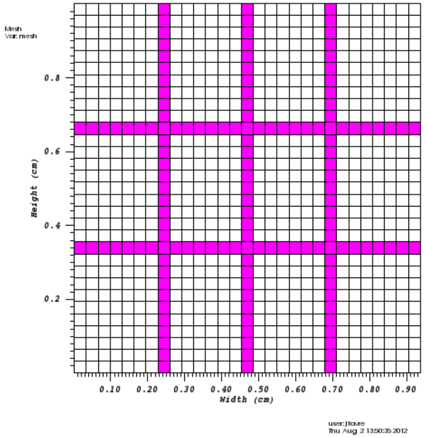
MPI tasks, ghost-cells, hyperslabs

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

Example: a 12-processor run

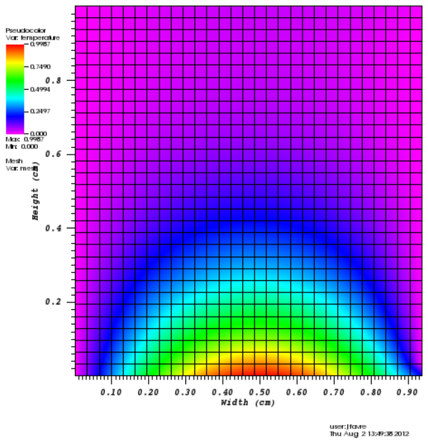


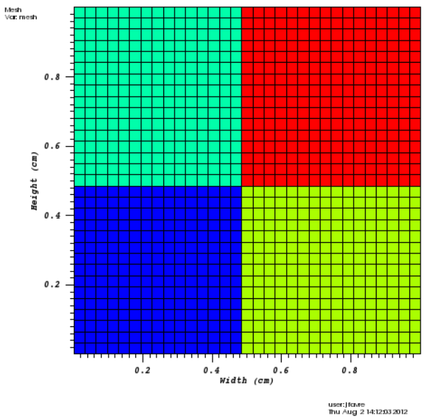


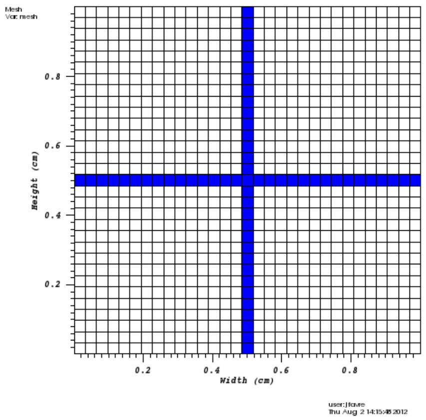


MPI tasks, ghost-cells, hyperslabs

Example: a 4-processor run



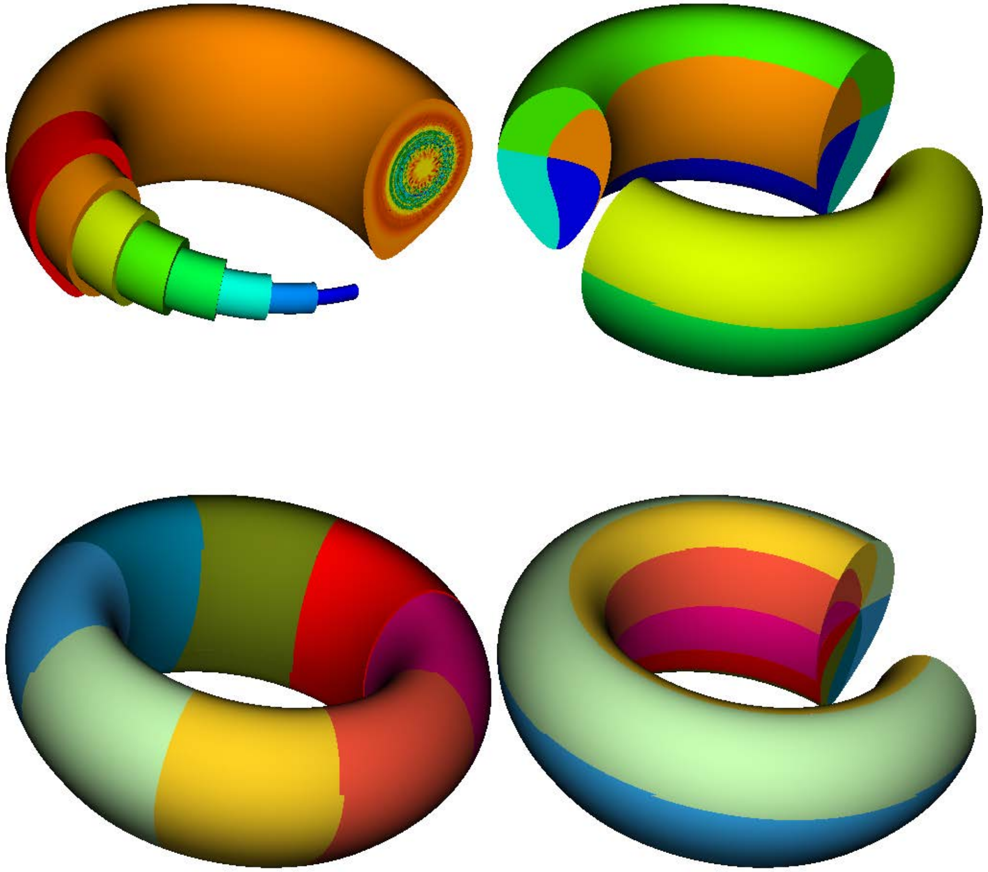




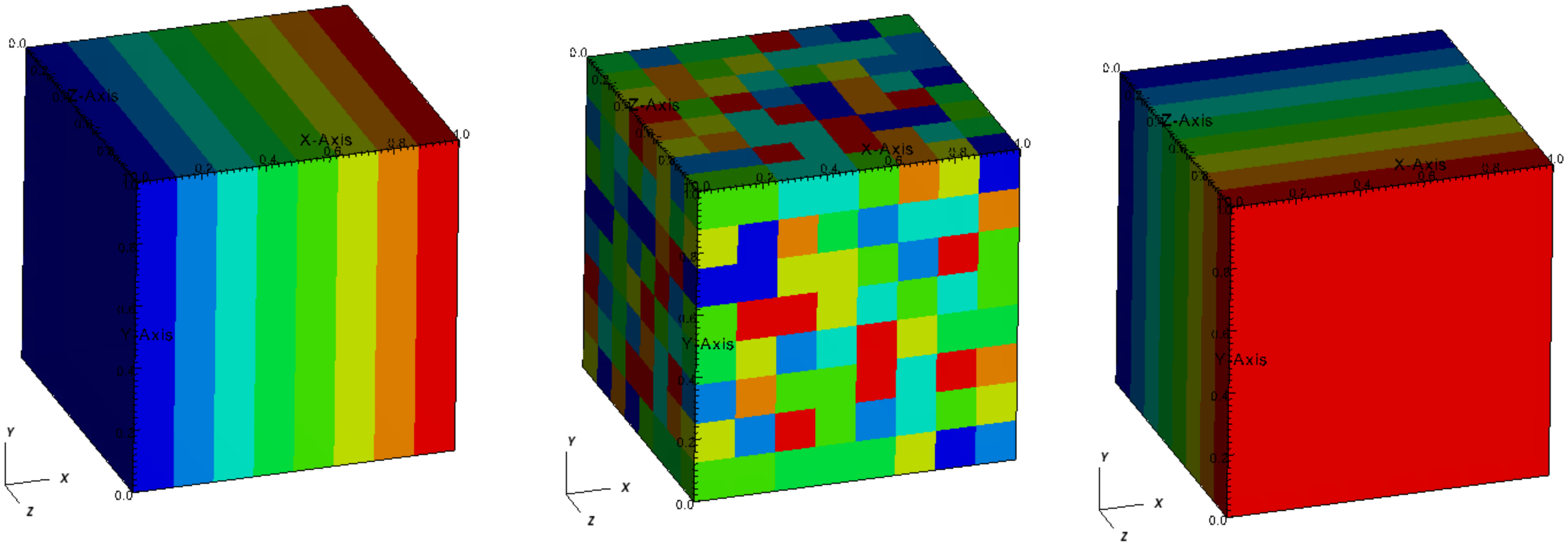
Example with a plasma simulation output

Four different read modes are implemented:

- radial,
- toroidal,
- poloidal,
- kd-tree



Example reading a BOV file in VisIt



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

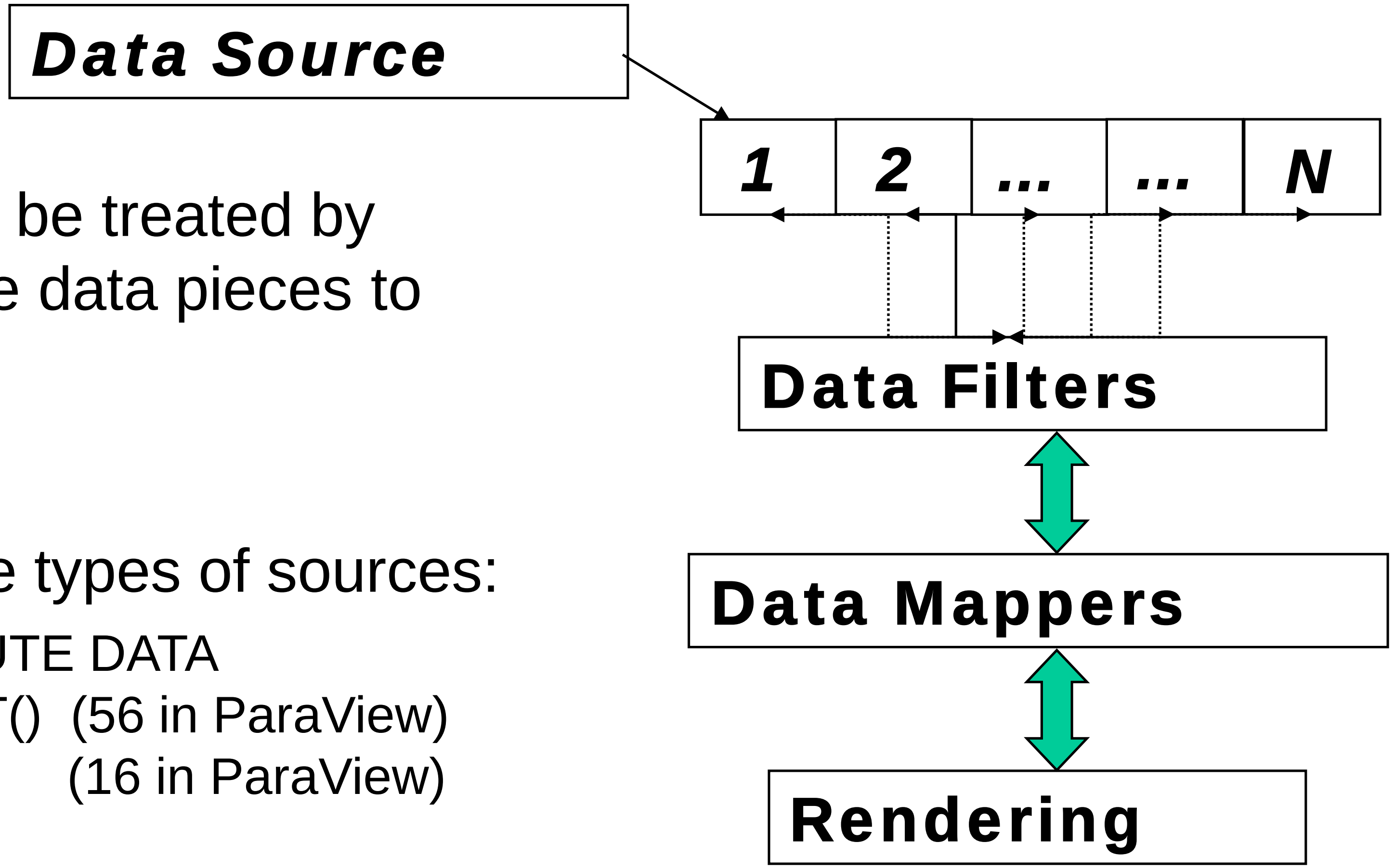
Cube dimension = 640x640x640

Bricklets = 80x80x80

Divide_brick = true

Modes: stride = 8, random, block

Distributed data and Streaming



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

- VTK differentiates between three types of sources:
  - DON'T KNOW HOW TO DISTRIBUTE DATA
  - CAN_HANDLE_PIECE_REQUEST() (56 in ParaView)
  - CAN_PRODUCE_SUB_EXTENT() (16 in ParaView)

Wiki article: VTK-Parallel_Pipeline

Question for the audience

VTK distinguishes between two formats:

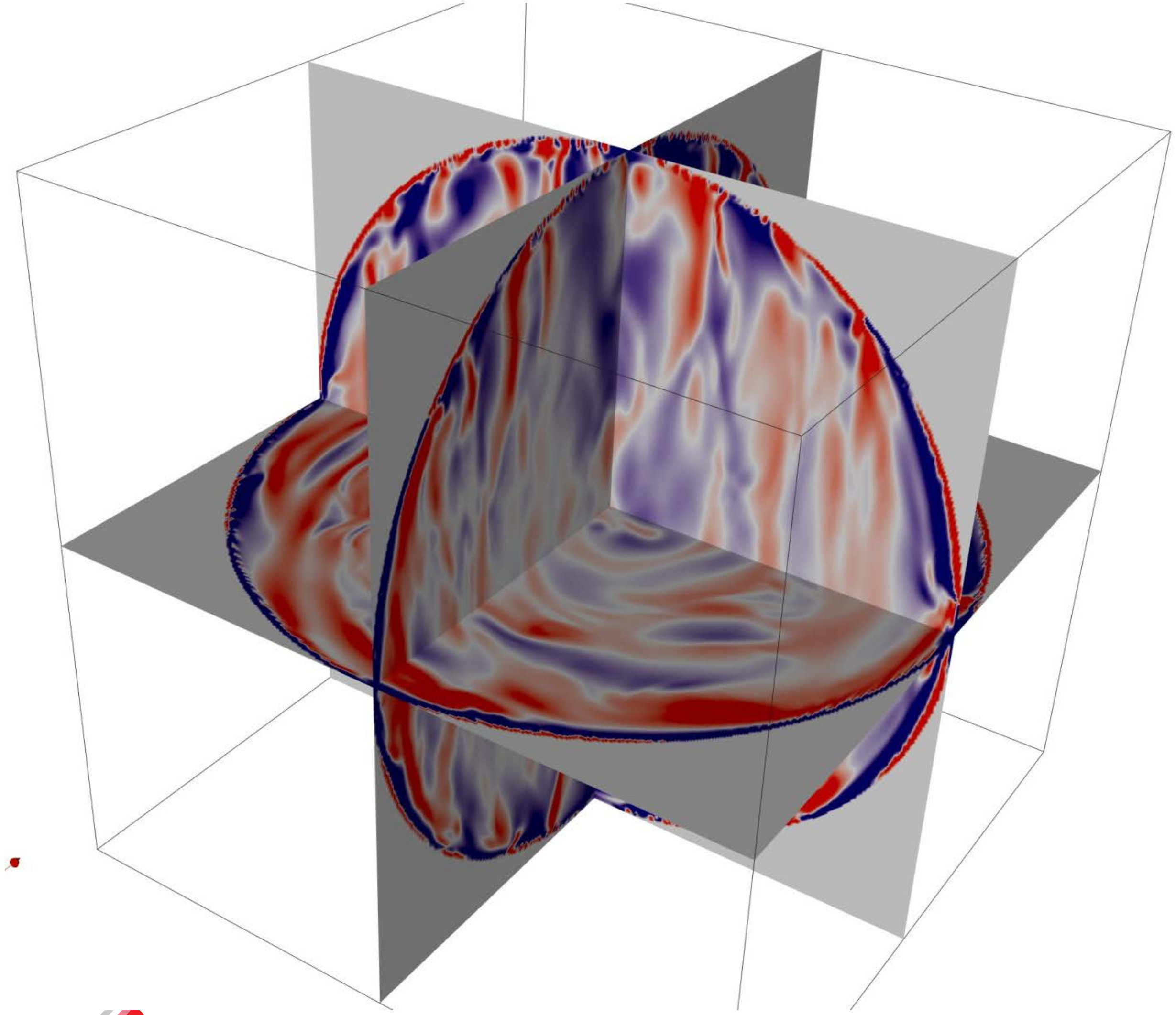
- Legacy format (*.vtk)
- XML-based format (more modern, enable distributed storage, compression, etc.)

Which format enable transparent distribution when read in parallel?

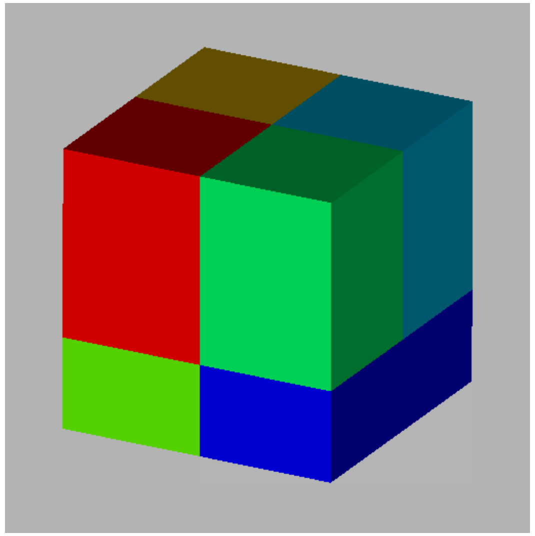
1. The XML Binary Image format *.vti ?
2. The XML Binary Unstructured Mesh format *.vtu ?

- 1
- 2
- Both 1) and 2)

Structured grids are split by IJK Extents



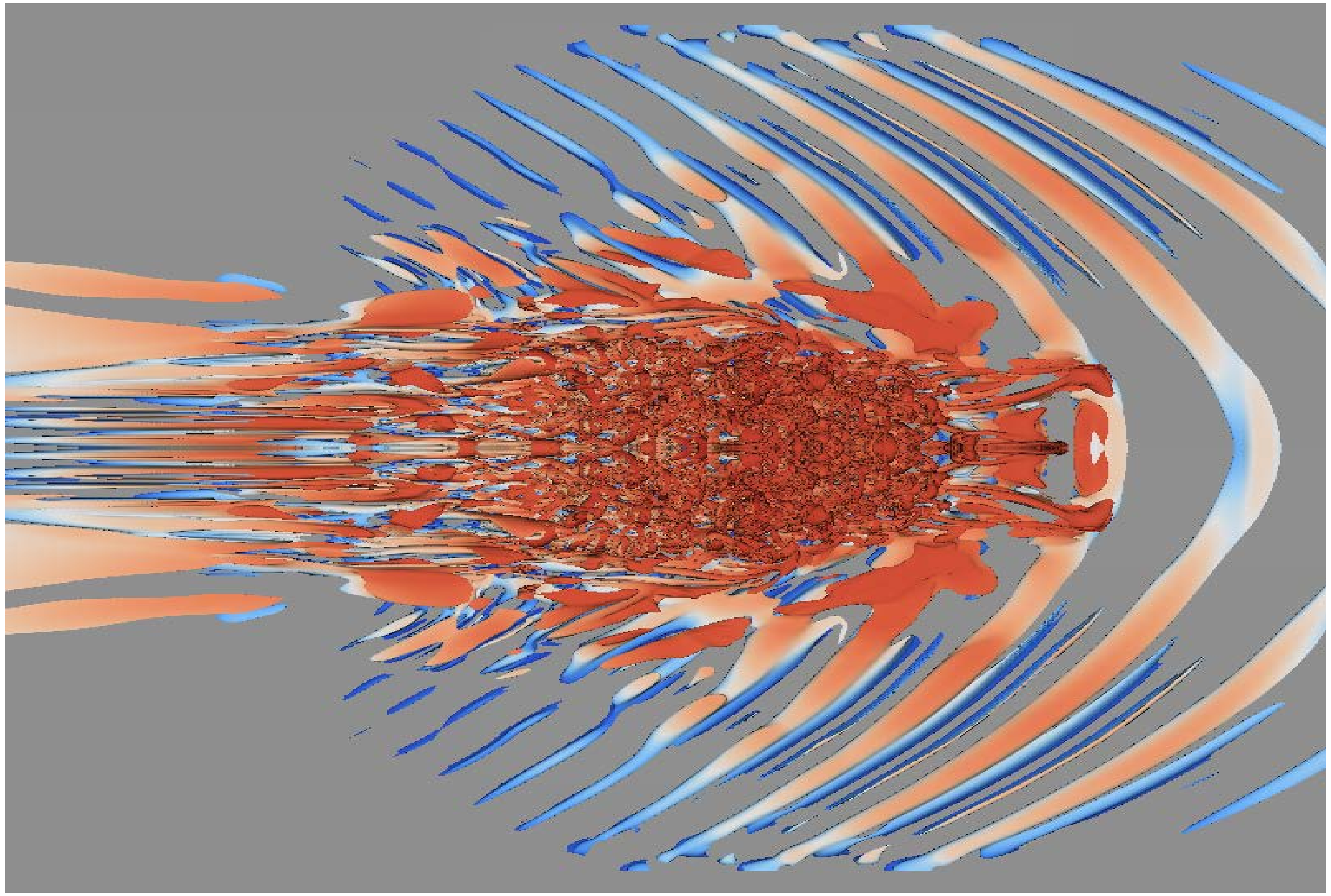
Parallel processing will enable requests for any subsets, including ghost-cells



XML output example with ghost cells

```
<VTKFile type="PStructuredGrid" version="0.1">
  <PStructuredGrid WholeExtent="0 65 0 65 0 65" GhostLevel="1">
    <Piece Extent=" 0 17  0 17 0 65"      Source="d0372_00.vts"/>
    <Piece Extent="16 33  0 17 0 65"      Source="d0372_01.vts"/>
    <Piece Extent="32 49  0 17 0 65"      Source="d0372_02.vts"/>
    <Piece Extent="48 65  0 17 0 65"      Source="d0372_03.vts"/>
    <Piece Extent=" 0 17 16 33 0 65"      Source="d0372_04.vts"/>
    <Piece Extent="16 33 16 33 0 65"      Source="d0372_05.vts"/>
    <Piece Extent="32 49 16 33 0 65"      Source="d0372_06.vts"/>
….
</PStructuredGrid>
</VTKFile>
```

Example for a Rectilinear Grid

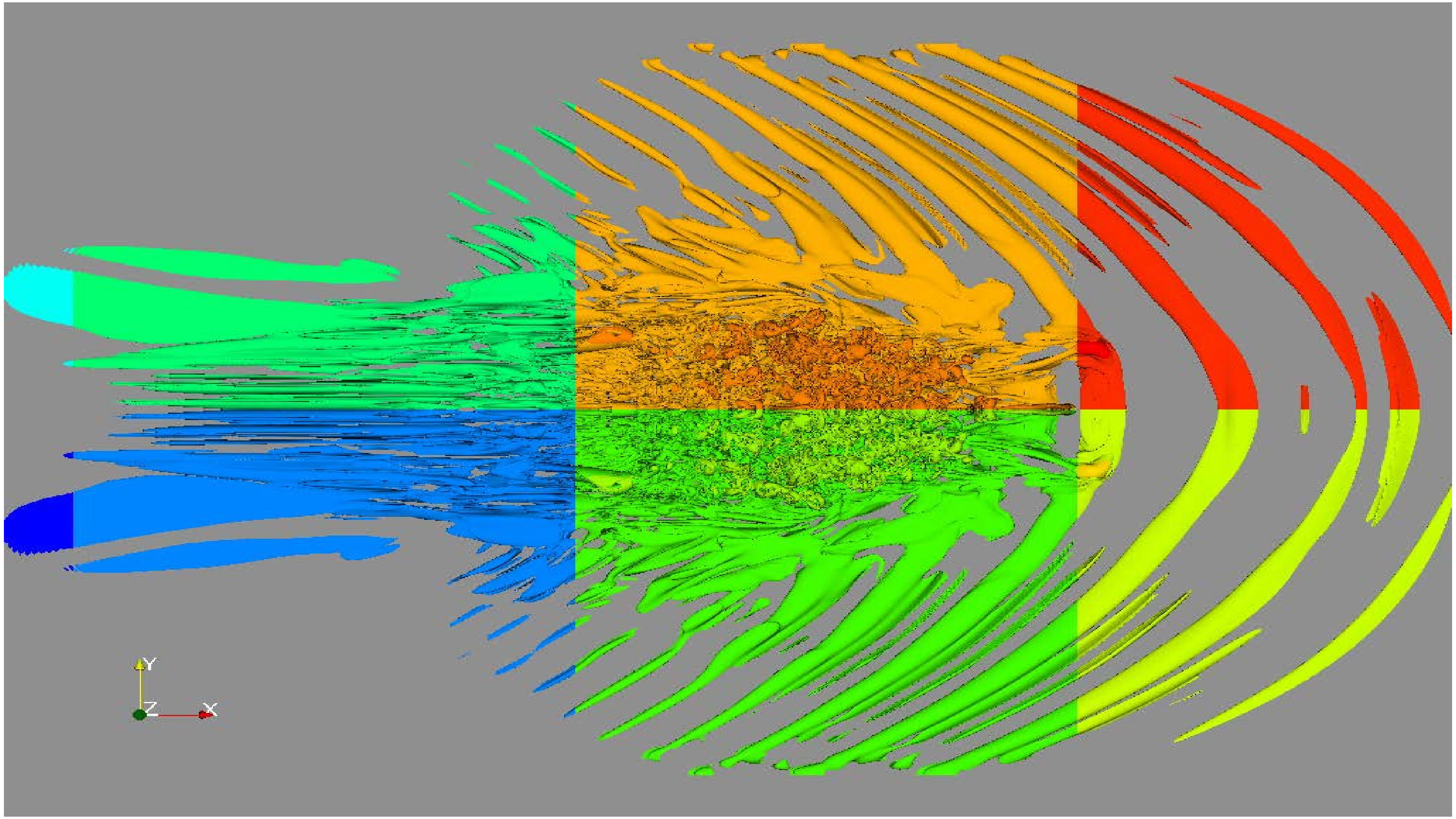


Assumption:

ParaView can read the data on any number of processors

Yes….but

Running on 8 pvservers



Optimizing the reading order (X, Y or Z)

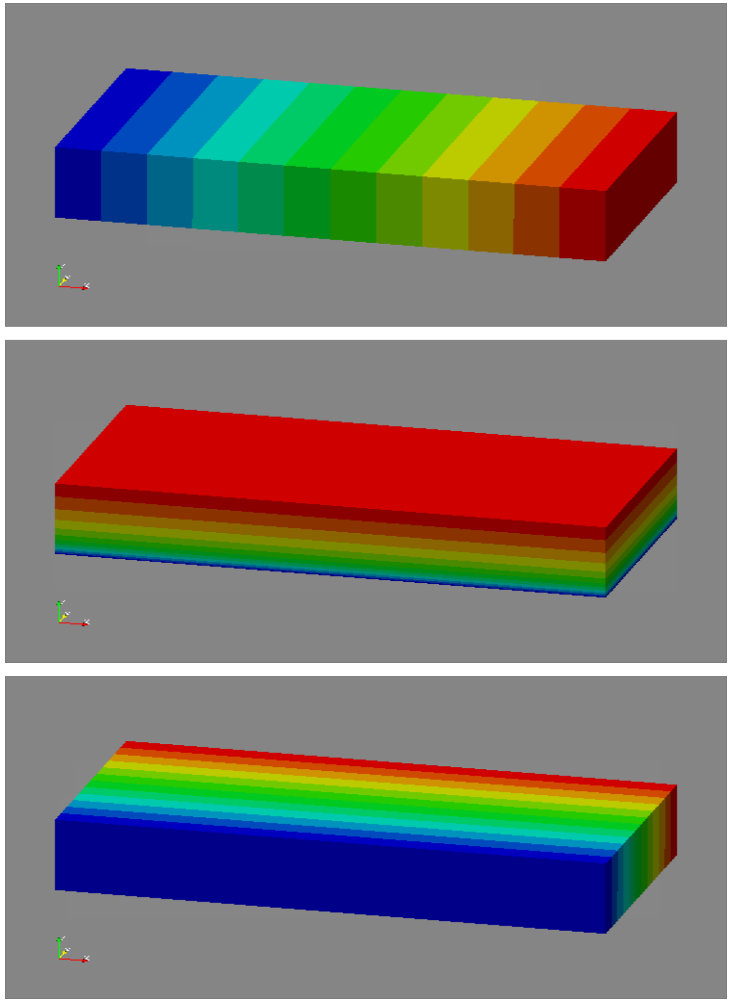
Reading 15 Gb of data with 12 cpus, with HDF5 hyperslabs

X hyperslabs: average read: 430 secs

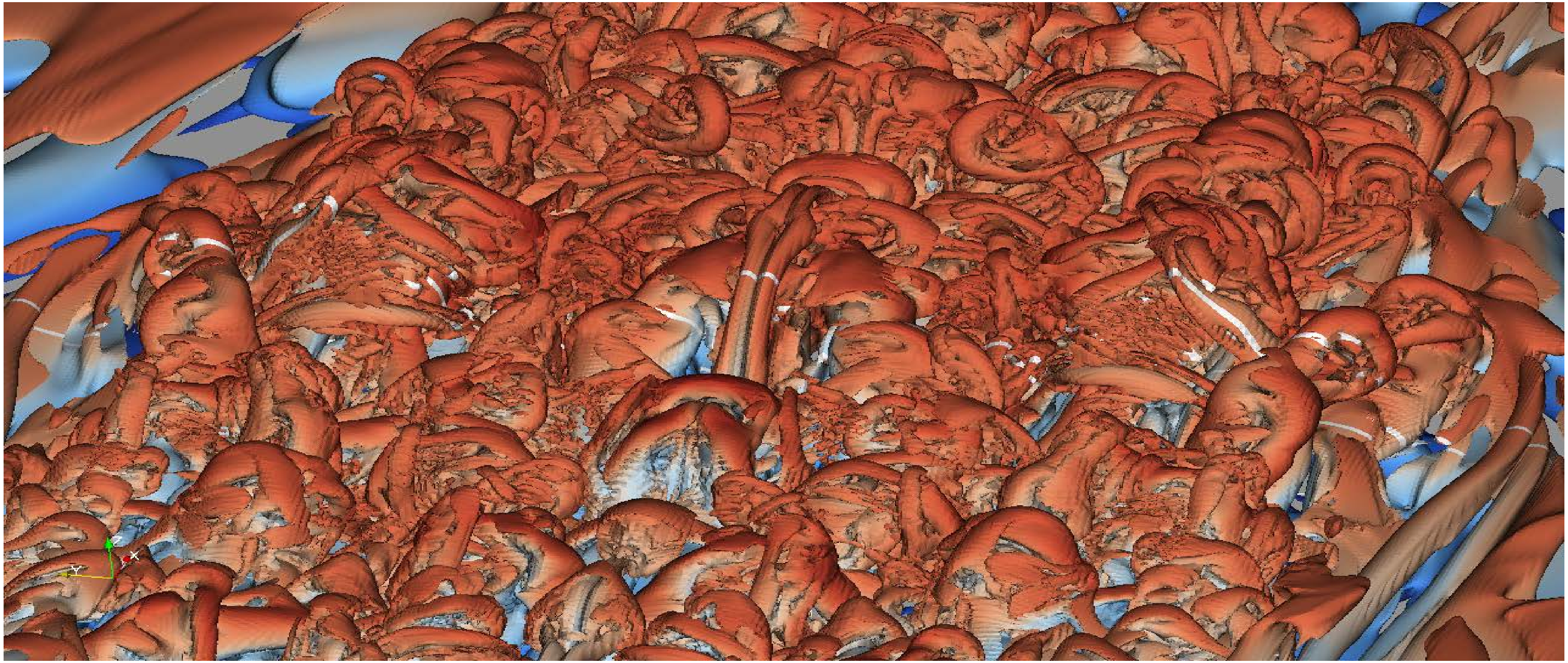
Y hyperslabs: average read: 142 secs

Z hyperslabs: average read: 36 secs

Parallel Visualization is ALL about file I/O ☺

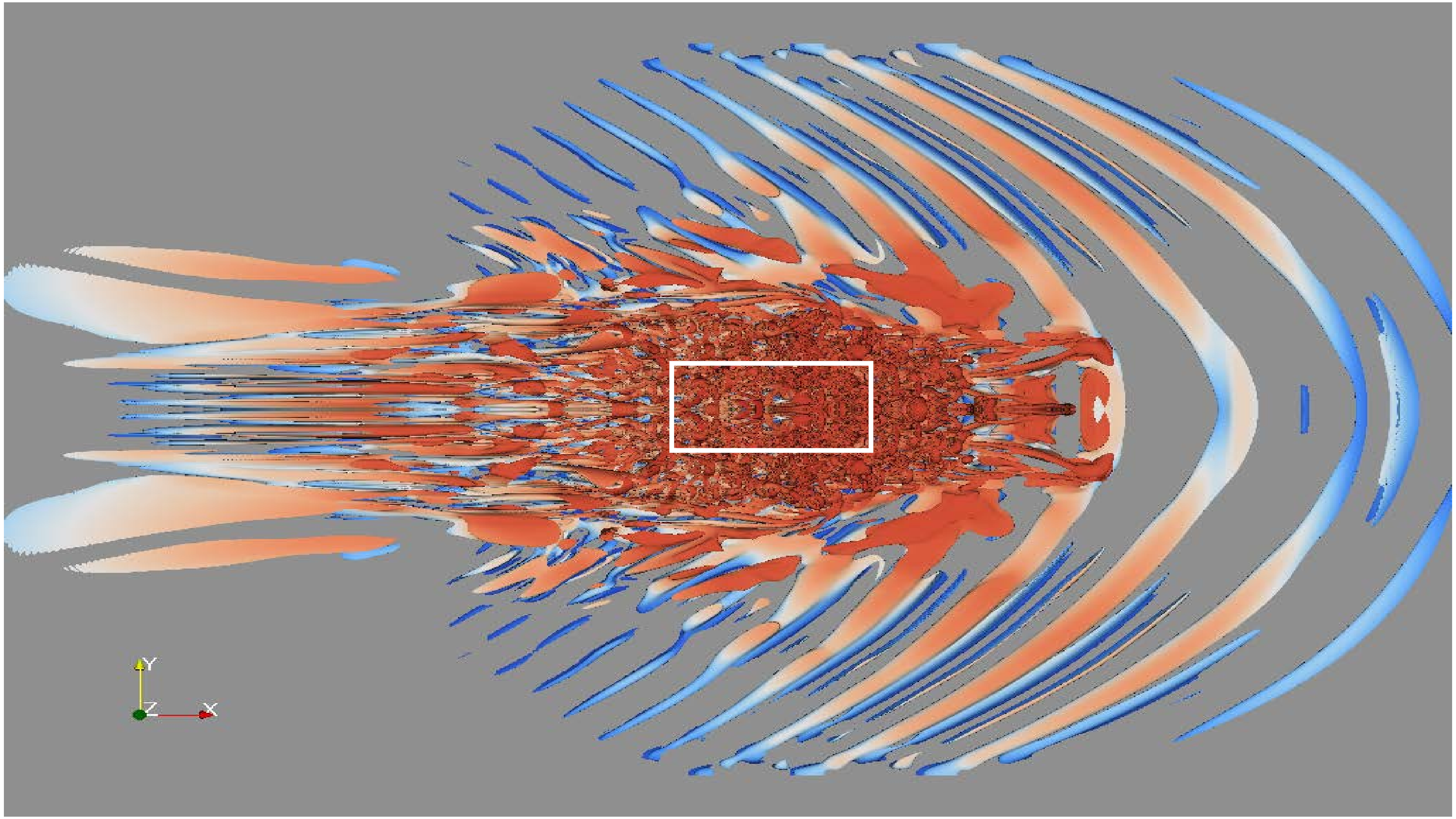


Zooming into the interesting zone



How much data was read, isosurfaced, and never displayed in this picture?

Adjusting the Data Extents…

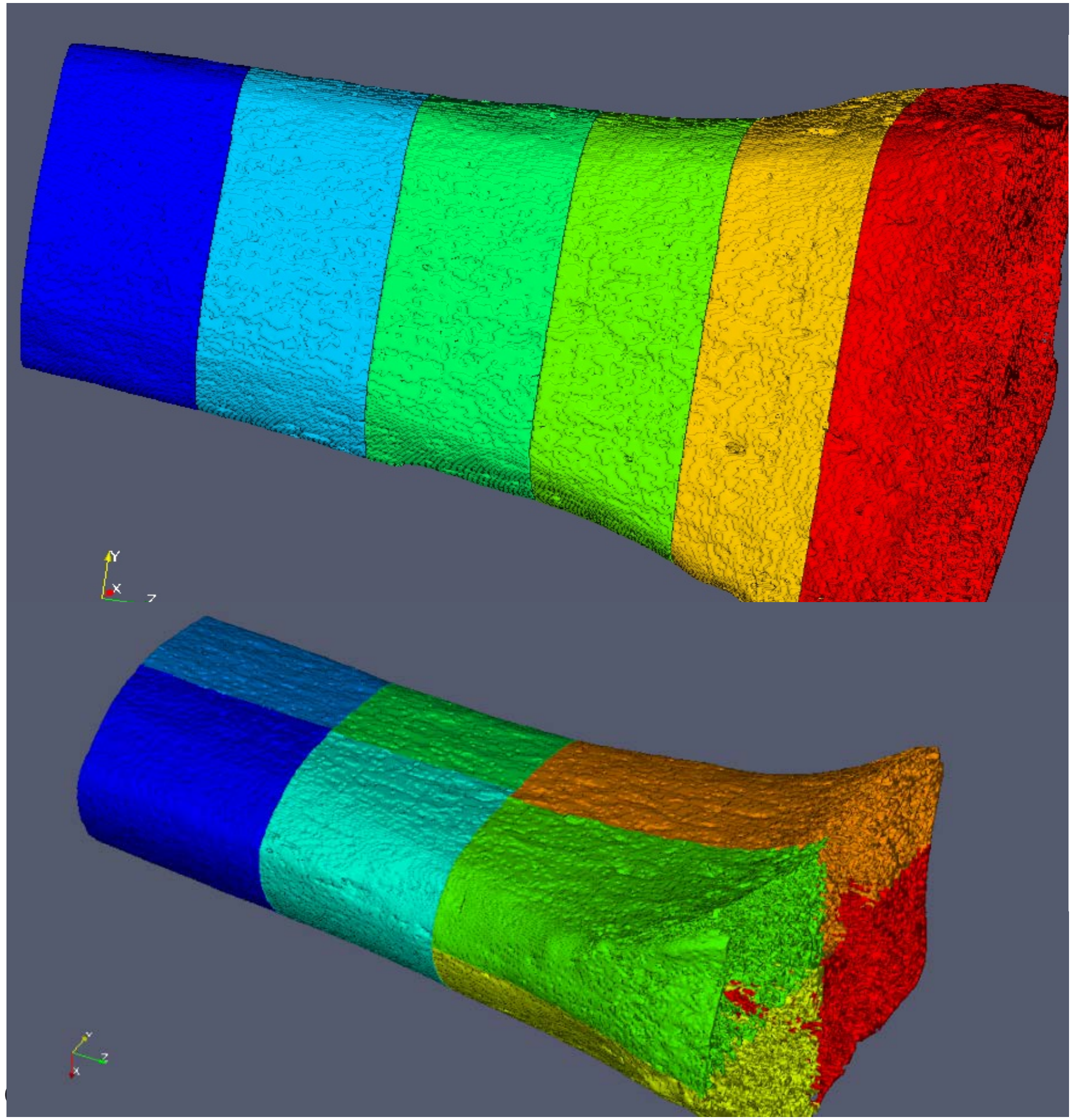


Reading much less data

display only 1/40-th of the data volume

25 millions instead of one billion cells

Unstructured grids are split into N pieces

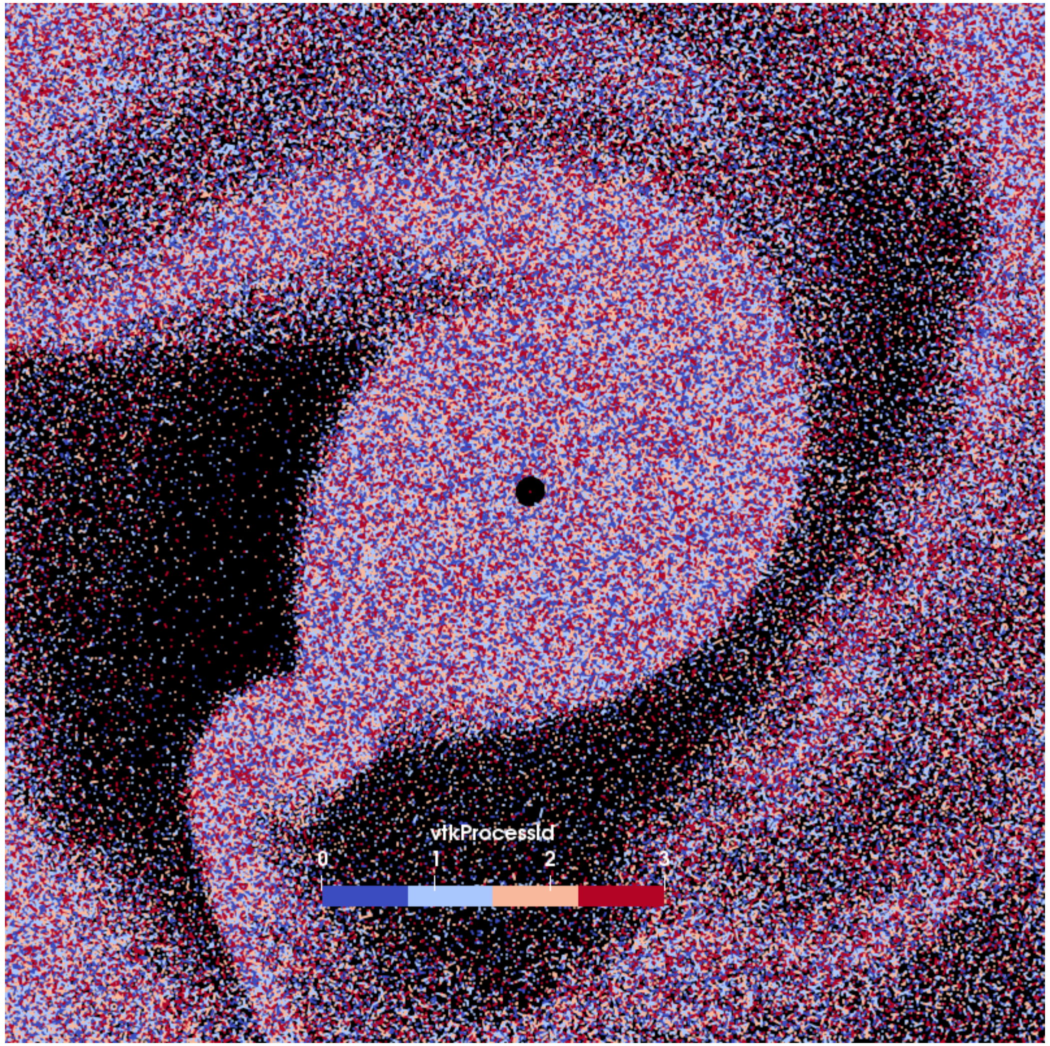


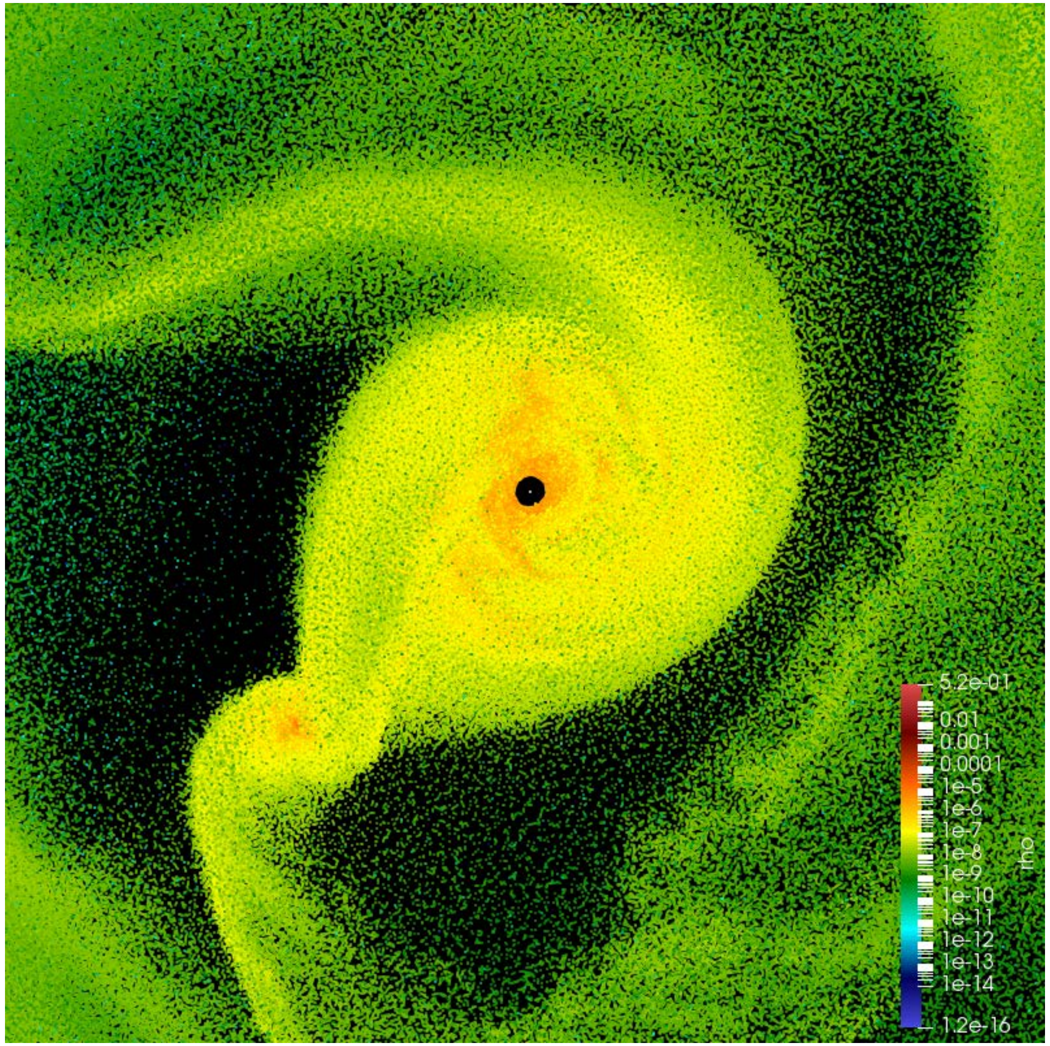
- The meaning of “pieces” can vary
- If ghost cells cannot be generated by the reader, ParaView has two filters

- D3,
- GhostCellGenerator.

ETH zürich

Reading particle data





Eschew simplicity

- If the I/O is too simple, there might be a very high cost…

- Two examples:

- **Example 1: ASCII output**

Example 2: a finite element code stores its results in Xdmf/HDF5

- Mixed elements
- Connectivity given

```
<Topology TopologyType="Mixed" Dimensions="3">
<DataItem Dimensions="18" NumberType="Int" Precision="8" Format="XML">
7 0 1 2 3 4          # code for pyramid
8 1 5 6 2 7 8        # code for wedge
6 5 9 11 10          # code for tetra
</DataItem>
</Topology>
```

Unfortunately, the Connectivity required by VTK is different

```
CONNECTIVITY = np.array([5, 0,1,2,3,4,
                         6, 1,5,6,2,7,8,
                         4, 5, 9, 11, 10])

VTK_PYRAMID = 14, VTK_WEDGE = 13, VTK_TETRA = 10

CELL_TYPES = np.array([VTK_PYRAMID, VTK_WEDGE, VTK_TETRA],
dtype=np.ubyte)

CELL_OFFSETS = np.array([0, 6, 13])

output.SetCells(CELL_TYPES, CELL_OFFSETS, CONNECTIVITY)
```

reader must convert and re-shuffle

Read from HDF5 file:

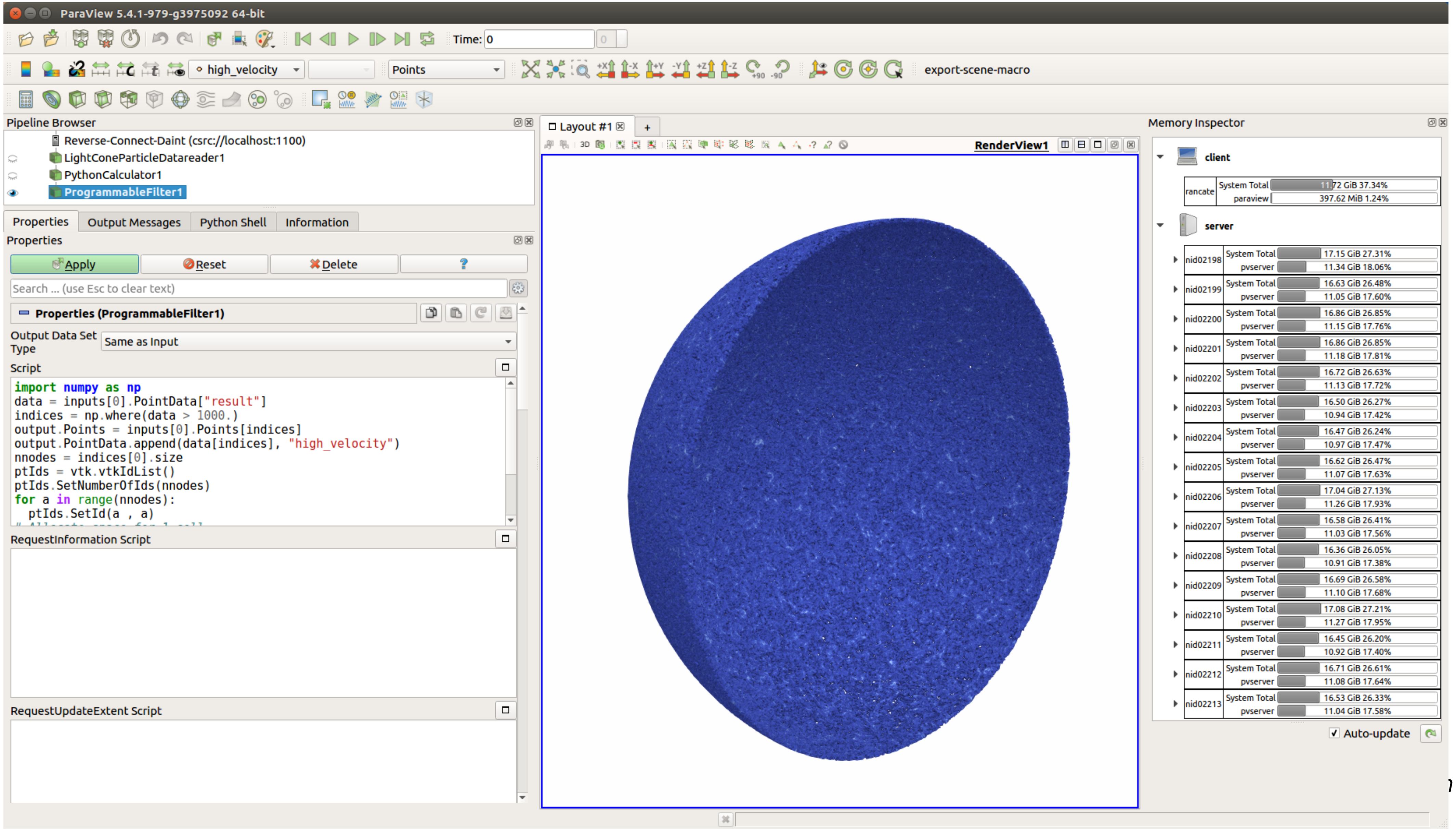
CONNECTIVITY = [7, 0,1,2,3,4,
8, 1,5,6,2,7,8,
6, 5, 9, 11, 10]

Converted to

CONNECTIVITY = [5, 0,1,2,3,4,
6, 1,5,6,2,7,8,
4, 5, 9, 11, 10]

Parallel data filtering

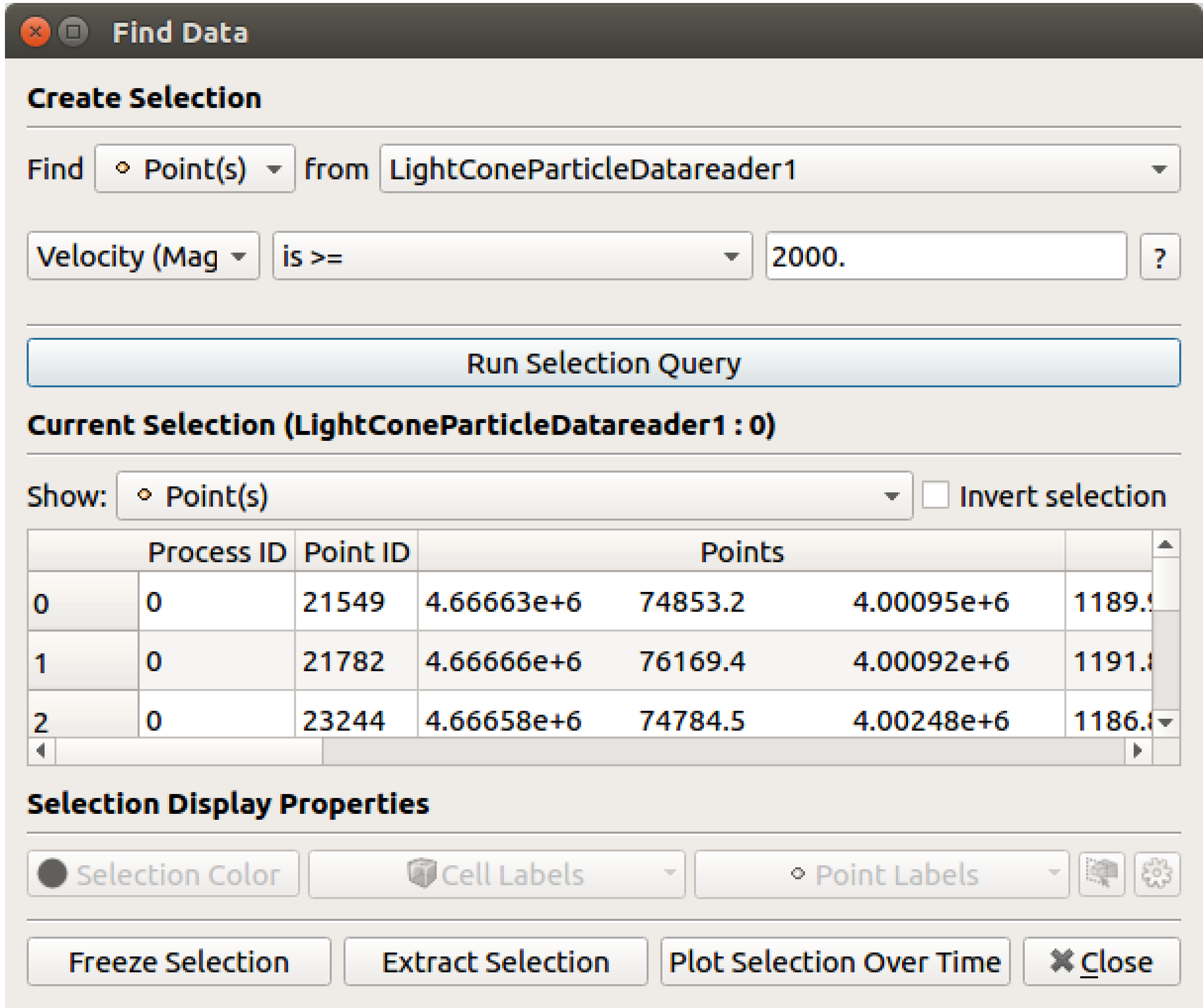
Visualization of a large hemisphere of points (3.8 Billion particles)



CSCS

Visualization of a large hemisphere of points

- 3,873,074,670 particles
- 16 nodes (64GB RAM, 16GB GPU RAM) are a minimum
- The standard way of using ParaView is doomed for immediate failure:
  - Read the data, display the data
  - Calculate the magnitude of velocity, display the data
  - Find the high velocity particles, display the data



CSCS

Visualization of a large hemisphere of points

Solution on N pvservers:

- Each of the N paraview server holds one poly-vertex cell with 1/N of the particles
- Do not display the data.

- Calculate the magnitude of velocity

  numpy.linalg.norm(inputs[0].PointData["Velocity"], axis=1)

- Extract the high velocity particles => 16 cells with 377 M particles (1/10th of original size)

  data = inputs[0].PointData["mag_velocity"]
  indices = np.where(data > 1000.)
  output.Points = inputs[0].Points[indices]
  output.PointData.append(data[indices], 'high_velocity')

  => Render these particles with OSPRay and shadows for high visual perception

Can I use the VTK-numpy interface in parallel?

How do I find the global min and max of a scalar field?

- Use numpy algorithms directly, and use mpi4py to do the proper reduction

- Use VTK's algorithms module.

```
from vtk.numpy_interface import algorithms as algs
_min = algs.min(data)
```

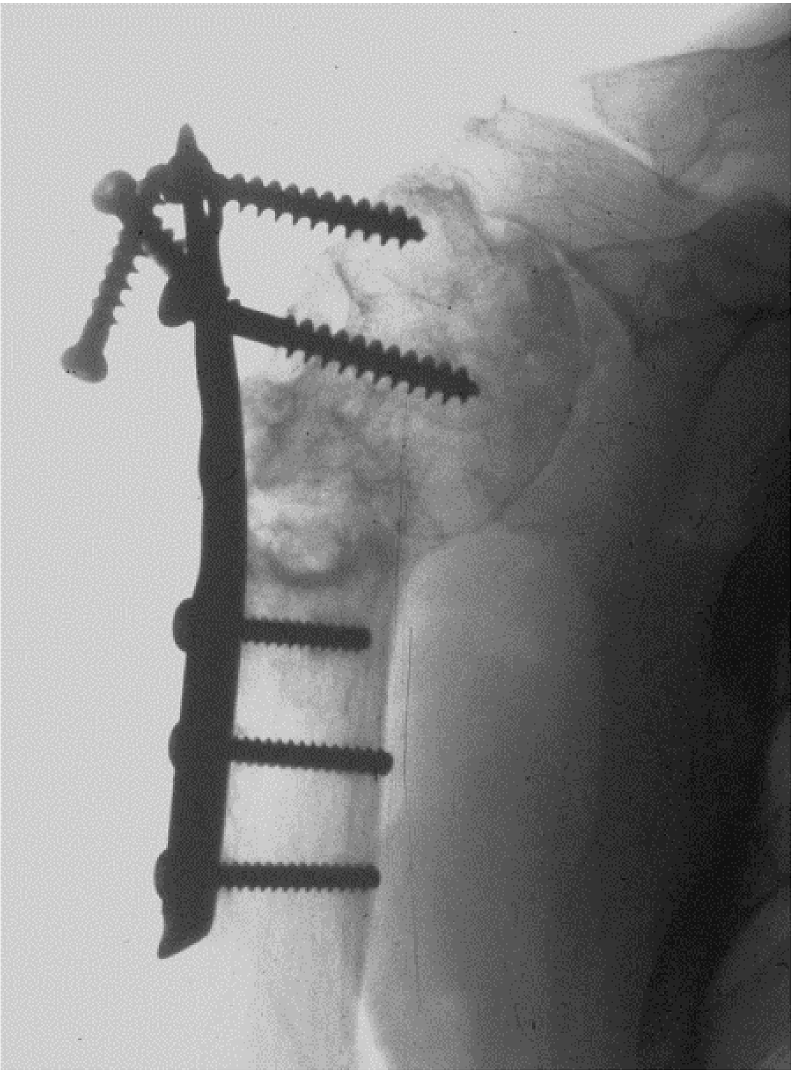
All algorithms in the numpy_interface.algorithms module work in parallel.

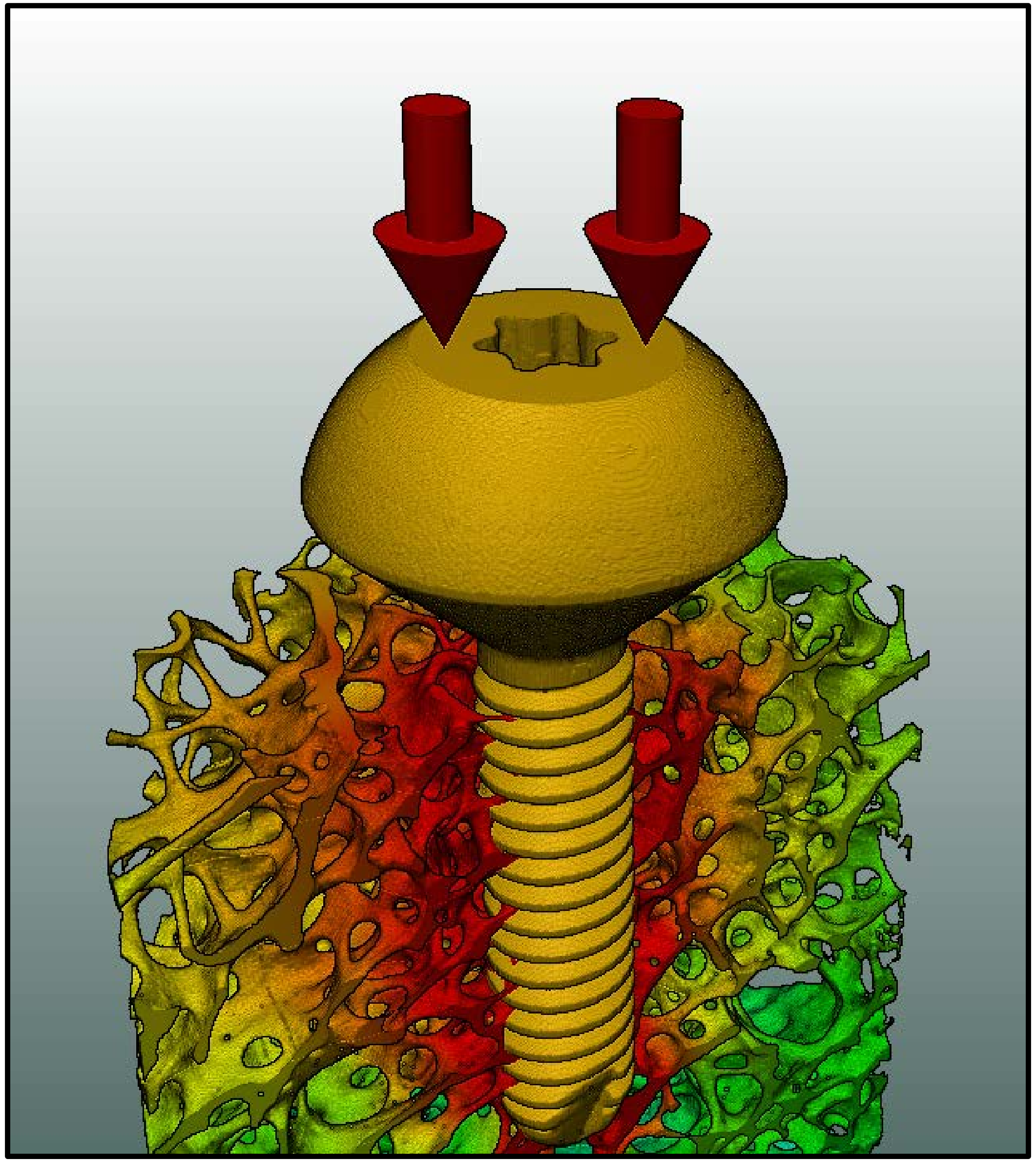
Moving down the visualization pipeline

- In previous slides, we focused on parallel data filtering, **without seeing** any data

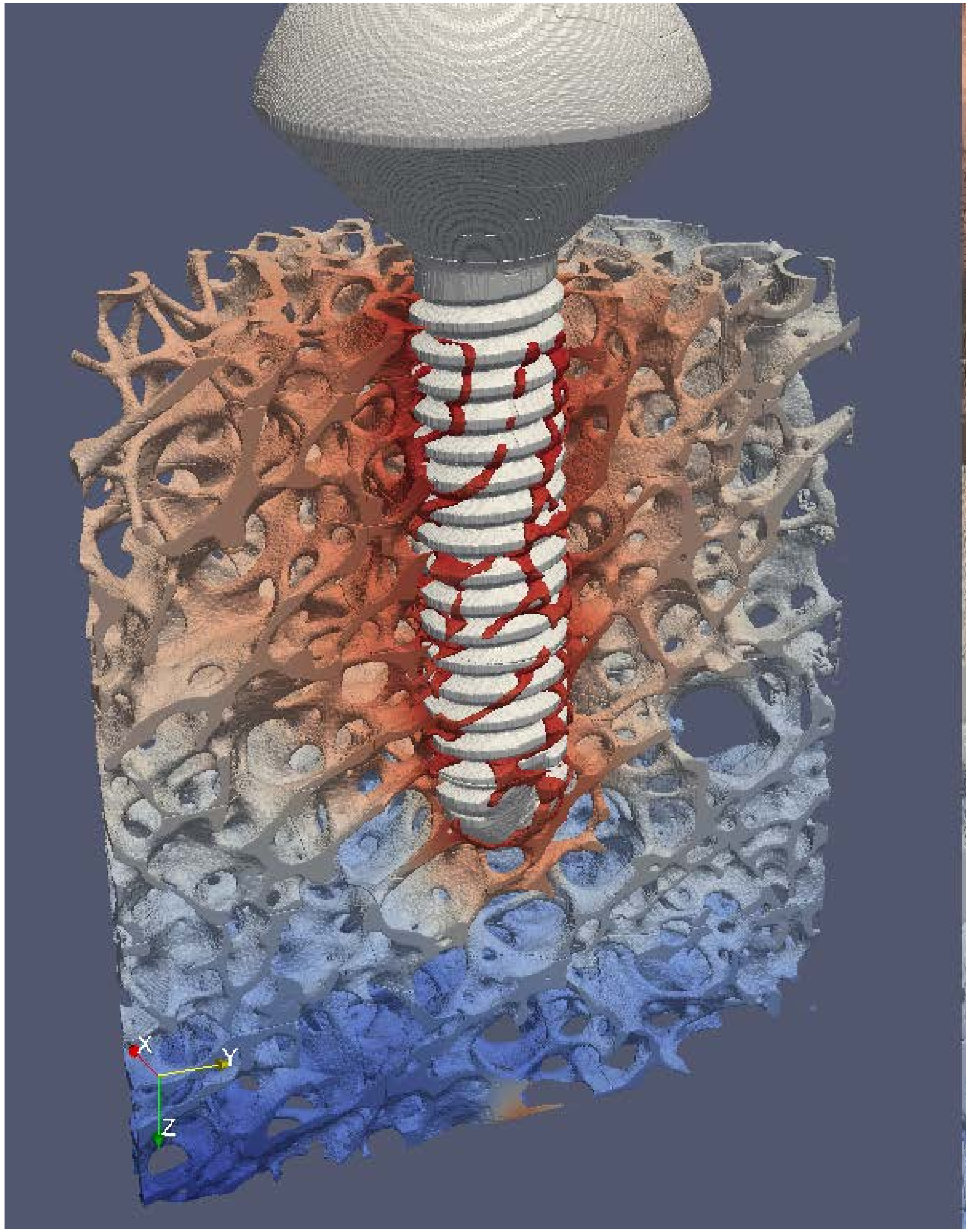
- We now move down the pipeline, and examine data **visually**.

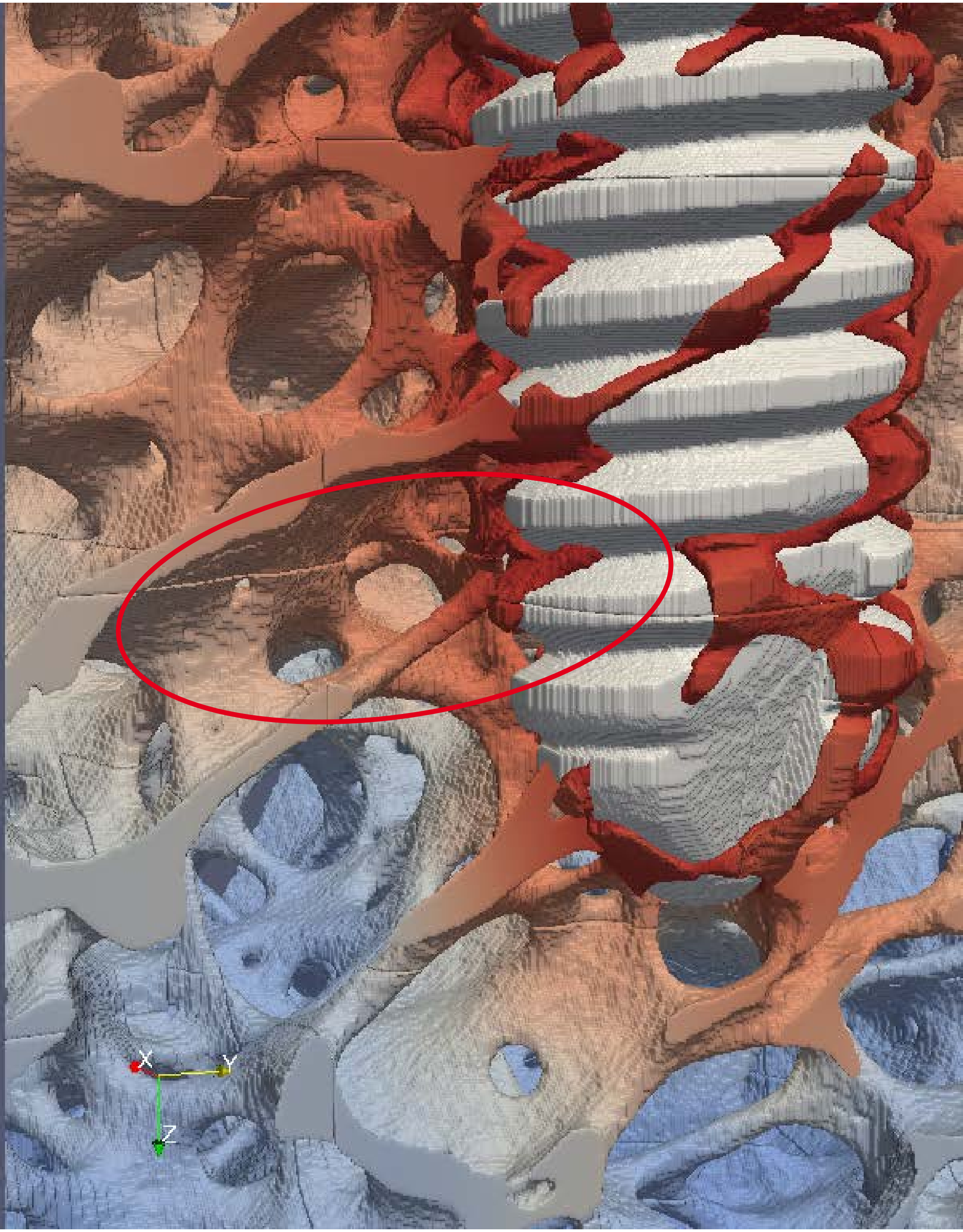
Very large geometry creation and rendering





CSCS





CSCS

The visualization pipeline is created interactively, adding more modules

Parallel Reader

↓

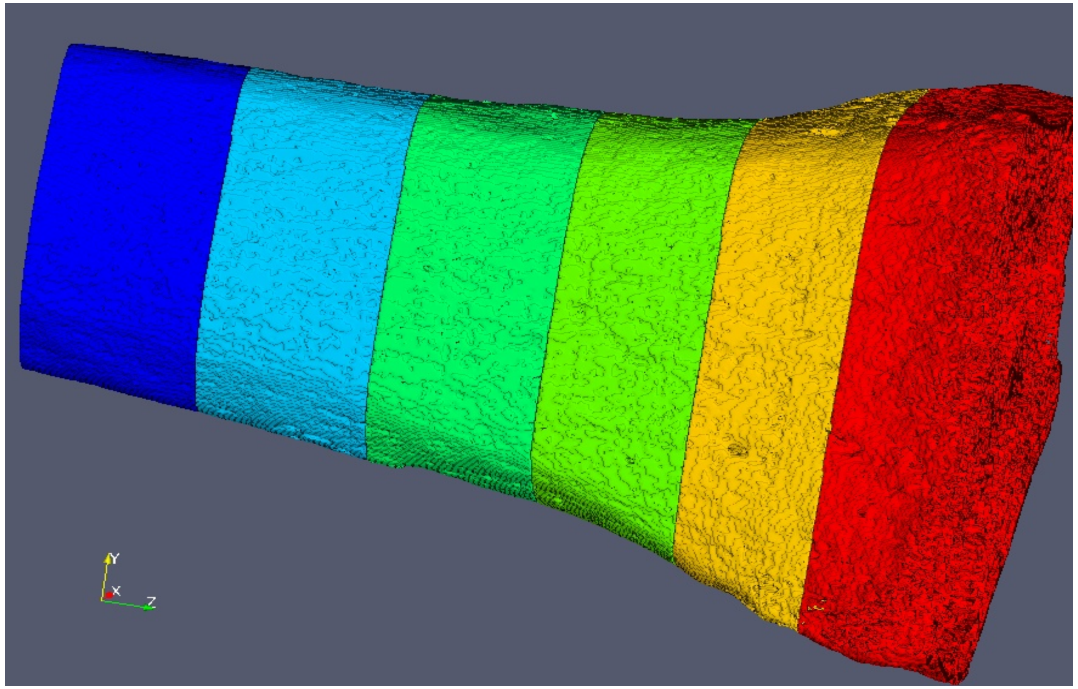
Parallel Surface Extraction

↓

Parallel Smoother

↓

Parallel Image Generation



The artifacts due to discontinuities are resolved thanks to ghost cells

Parallel Reader

↓

Ghost Cell Generation

↓

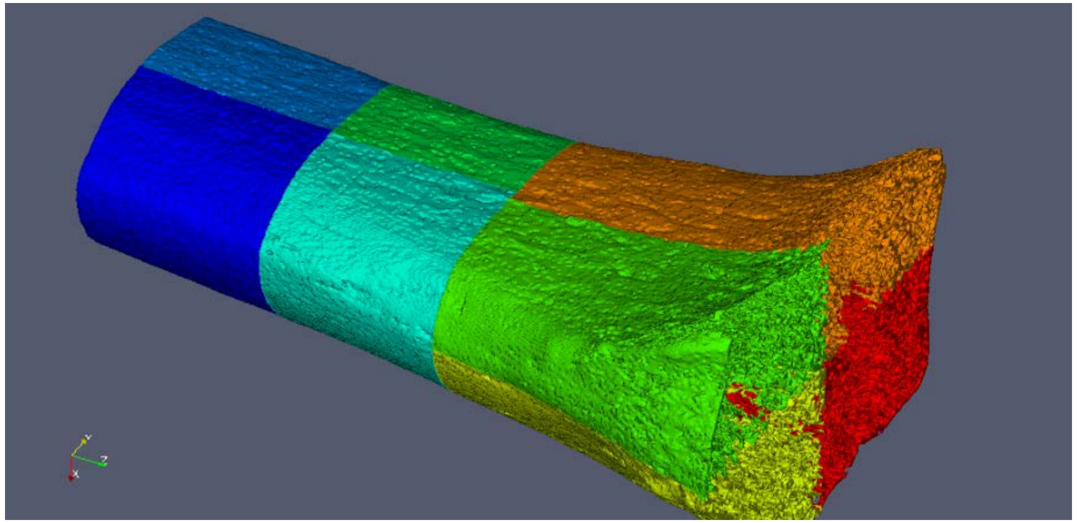
Parallel Surface Extraction

↓

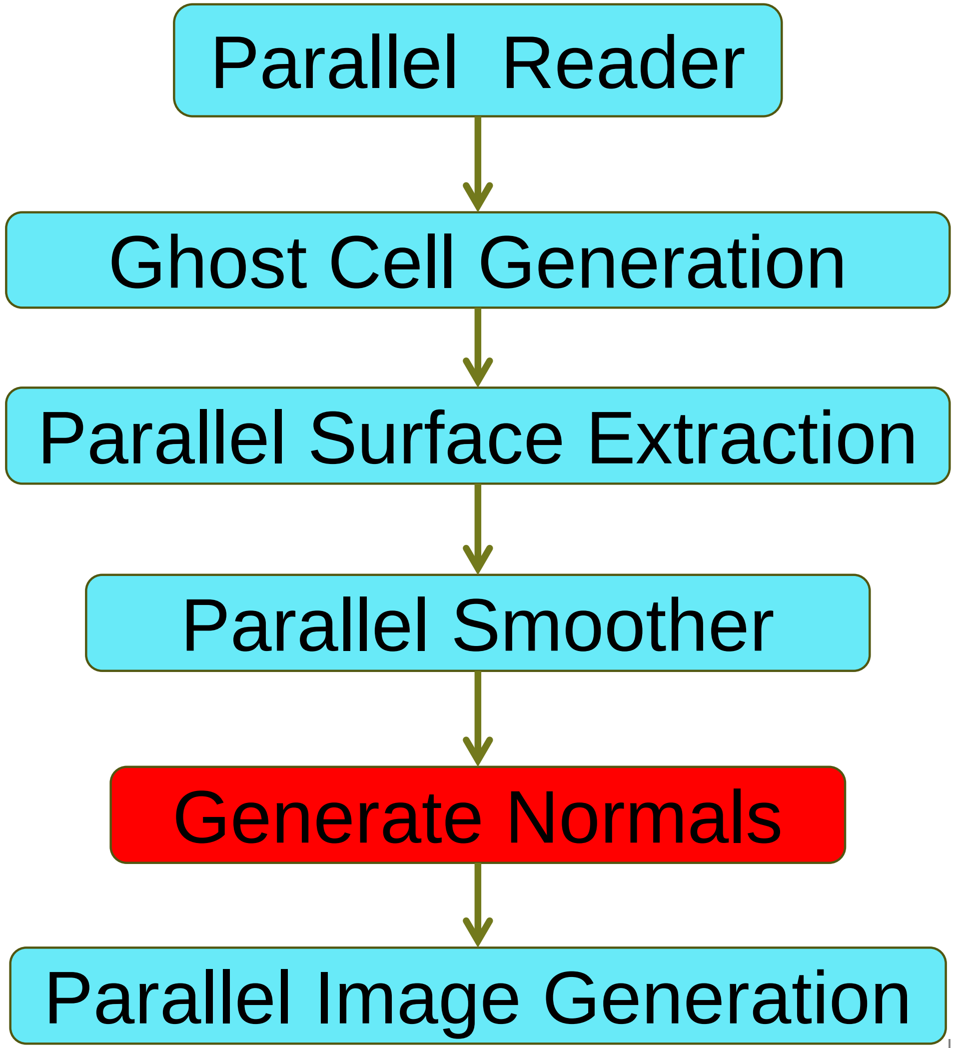
Parallel Smoother

↓

Parallel Image Generation

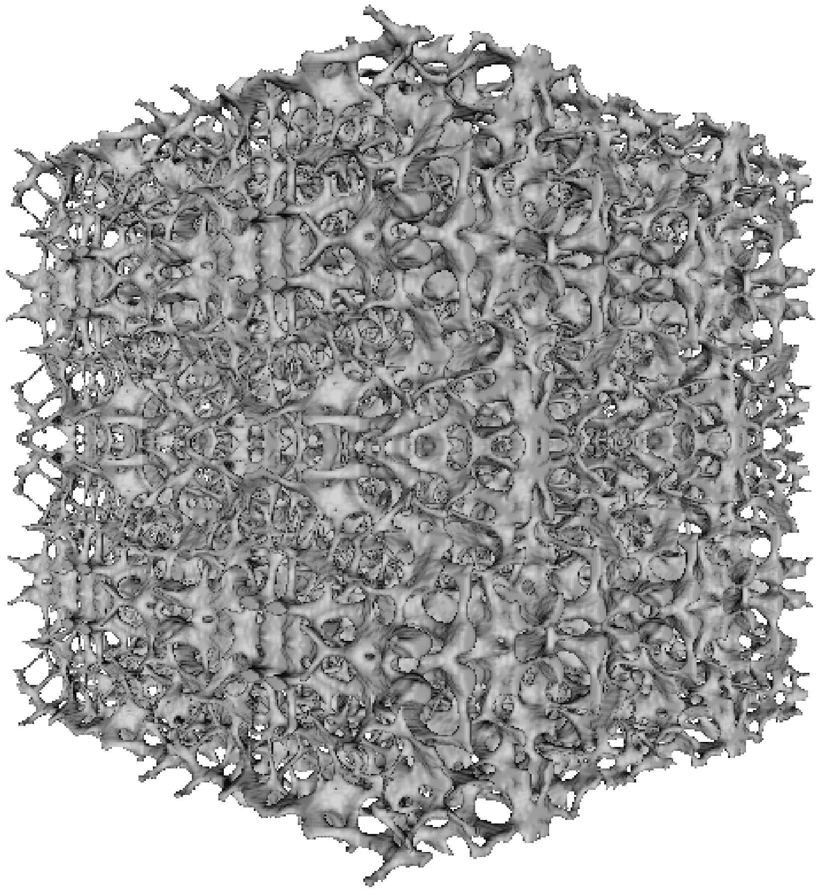


There remains visual boundaries due to illumination artifacts

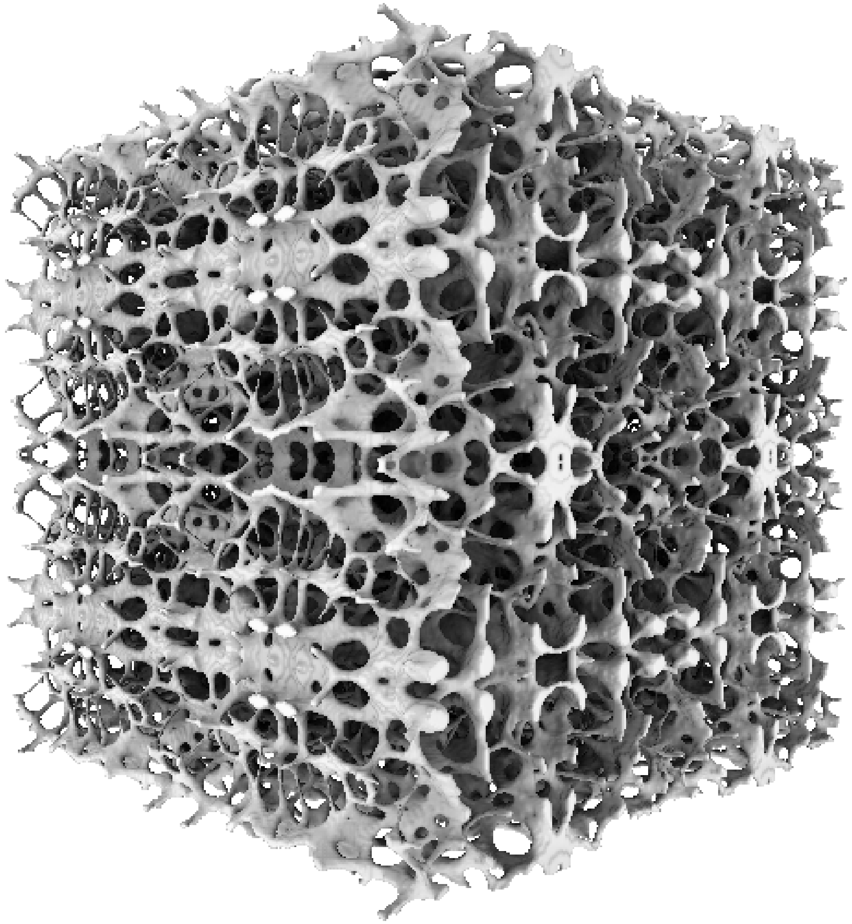


After 2 re-execution of the visualization pipelines….

We have this!



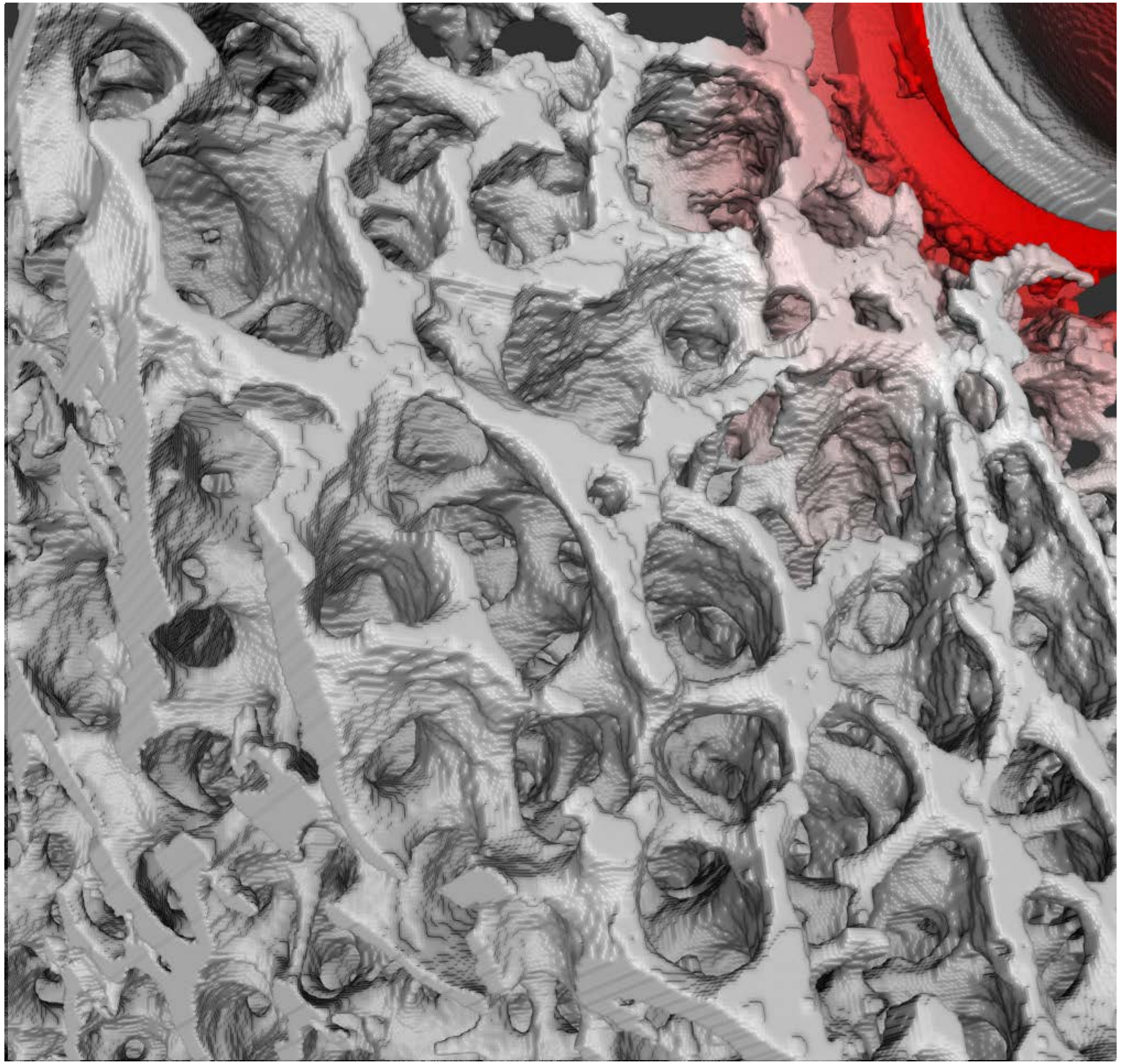
But we want this!



Need a rendering library to do shadows and ambient light occlusion

Must use ray-tracing techniques to do visibility computing

- NVIDIA OptiX available (only) in VTK
- Intel OSPRay integrated in ParaView



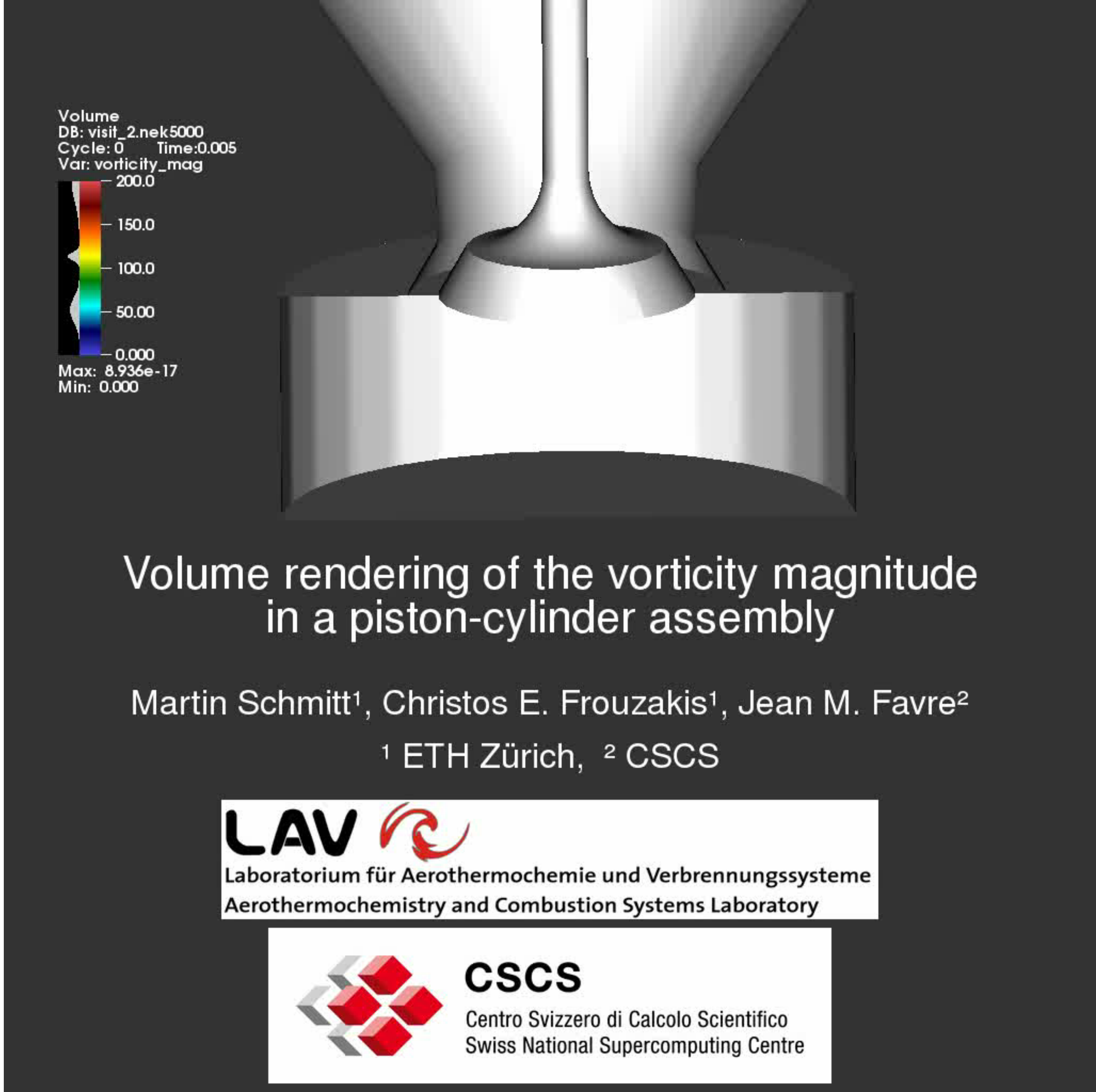
Volume rendering strategies

Understand the visualization technique and its implementation

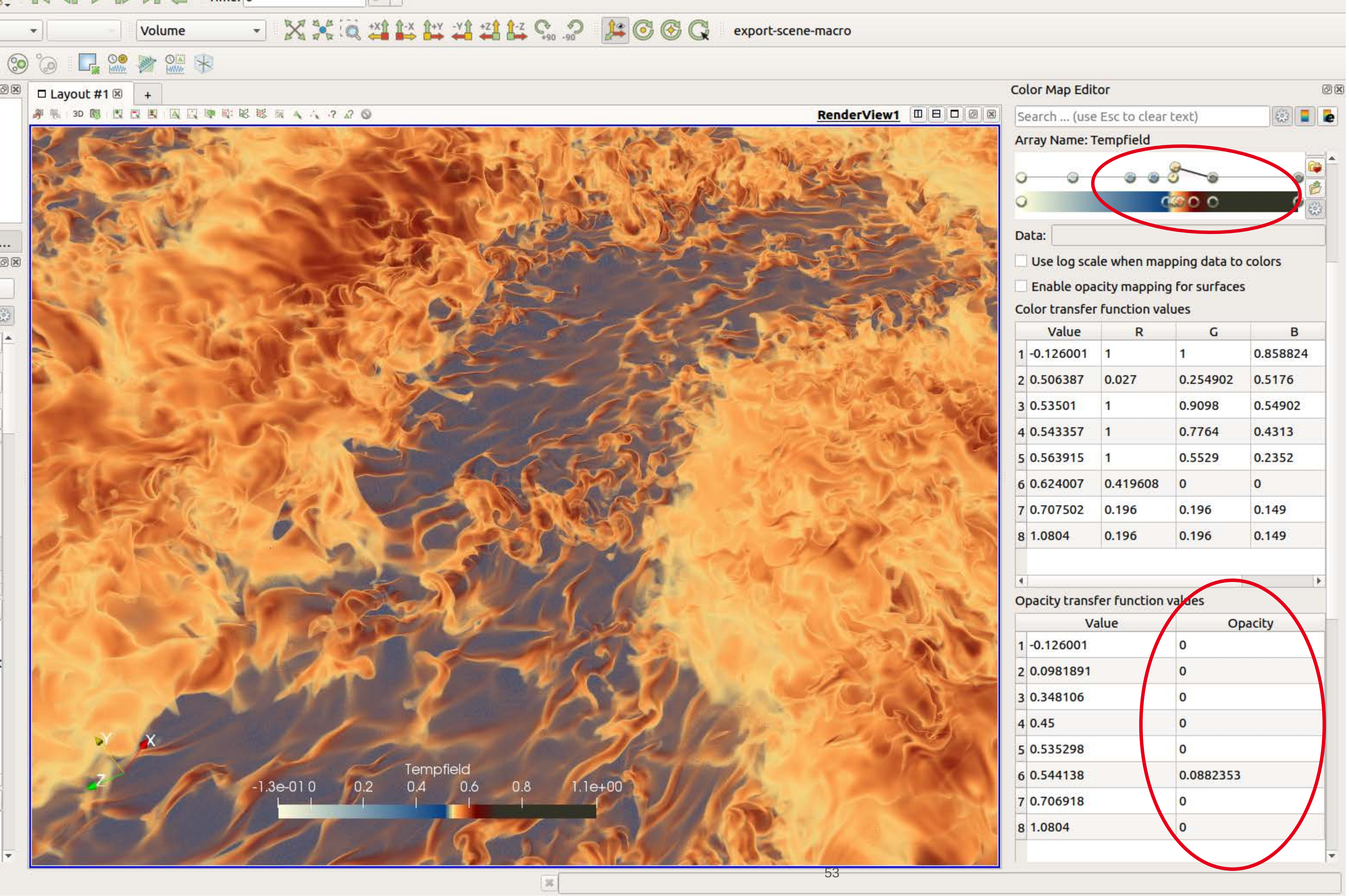
heard too many times:

"Volume rendering does **not** work!!!"

"Volume rendering crashes **all** the time!!!"

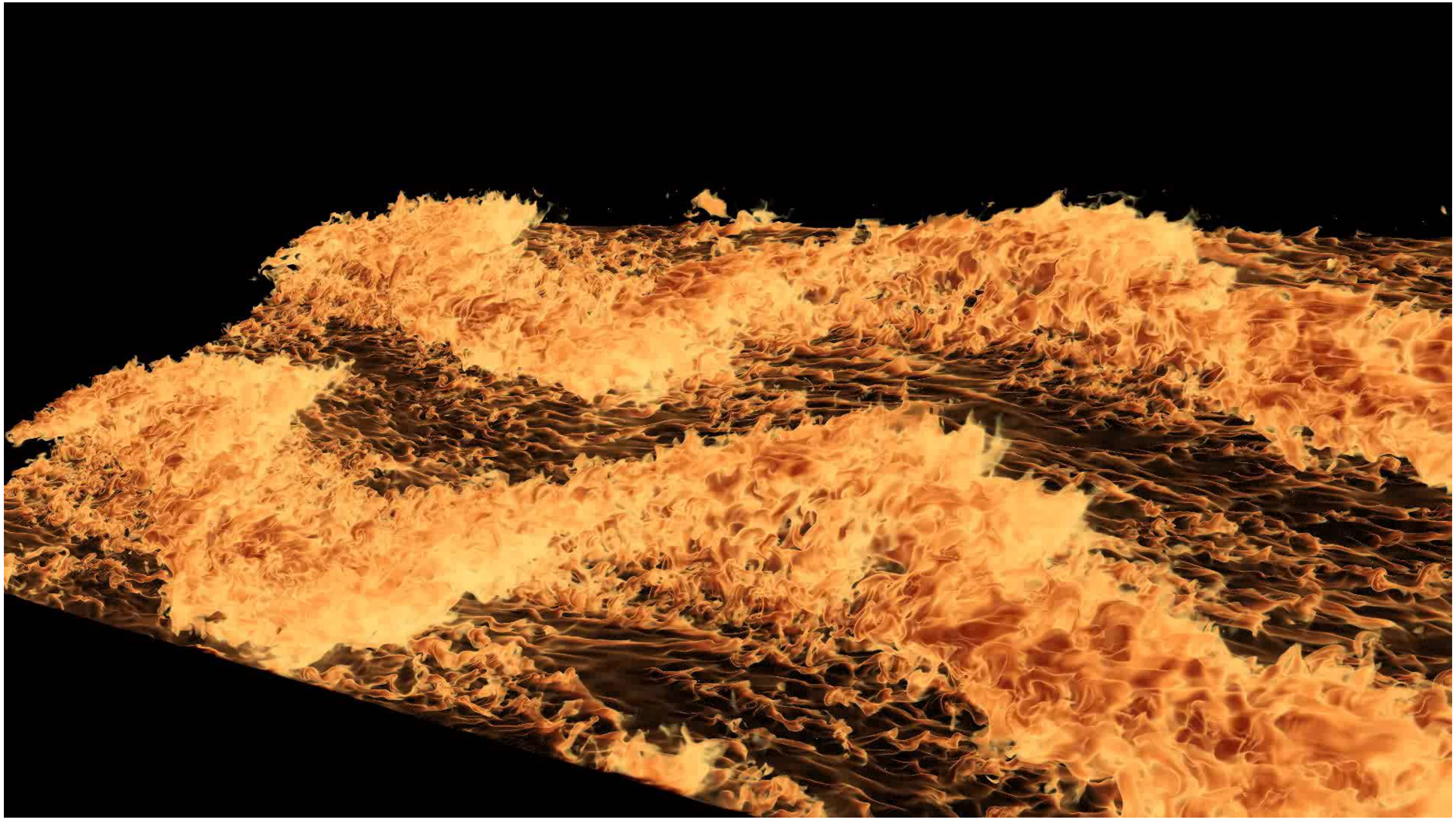


cscs



Volume rendering is a technique which requires a very light hand…

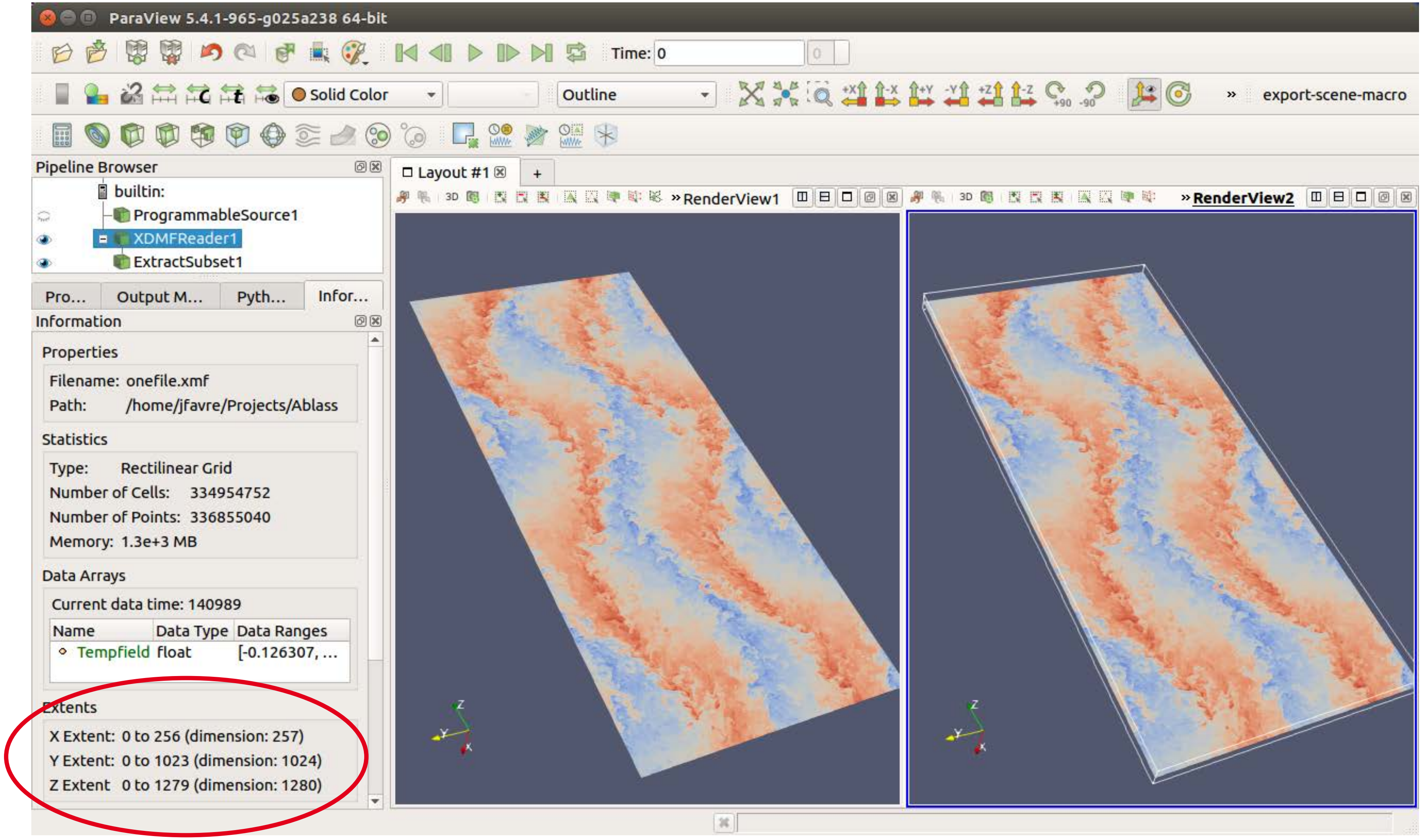
DNS of a sheared thermal convection



From the 2017 Gallery of Fluid Motion, APS Division of Fluid Dynamics
https://gfm.aps.org/meetings/dfd-2017/59b69eceb8ac316d38841b9c

3D grid
257*1024*1280

Occupies 1.3 GB

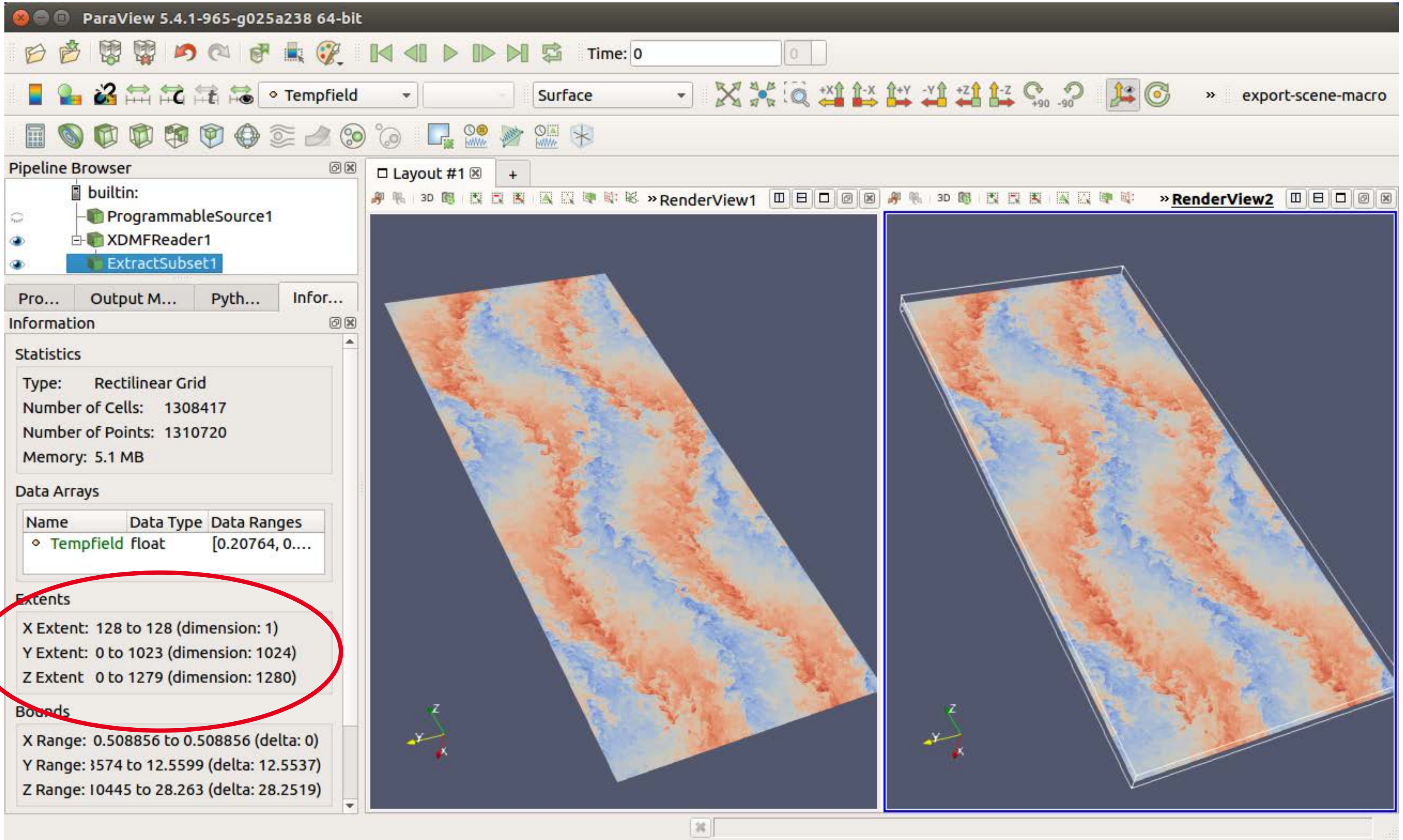


3D grid
257*1024*1280

Occupies 1.3 GB

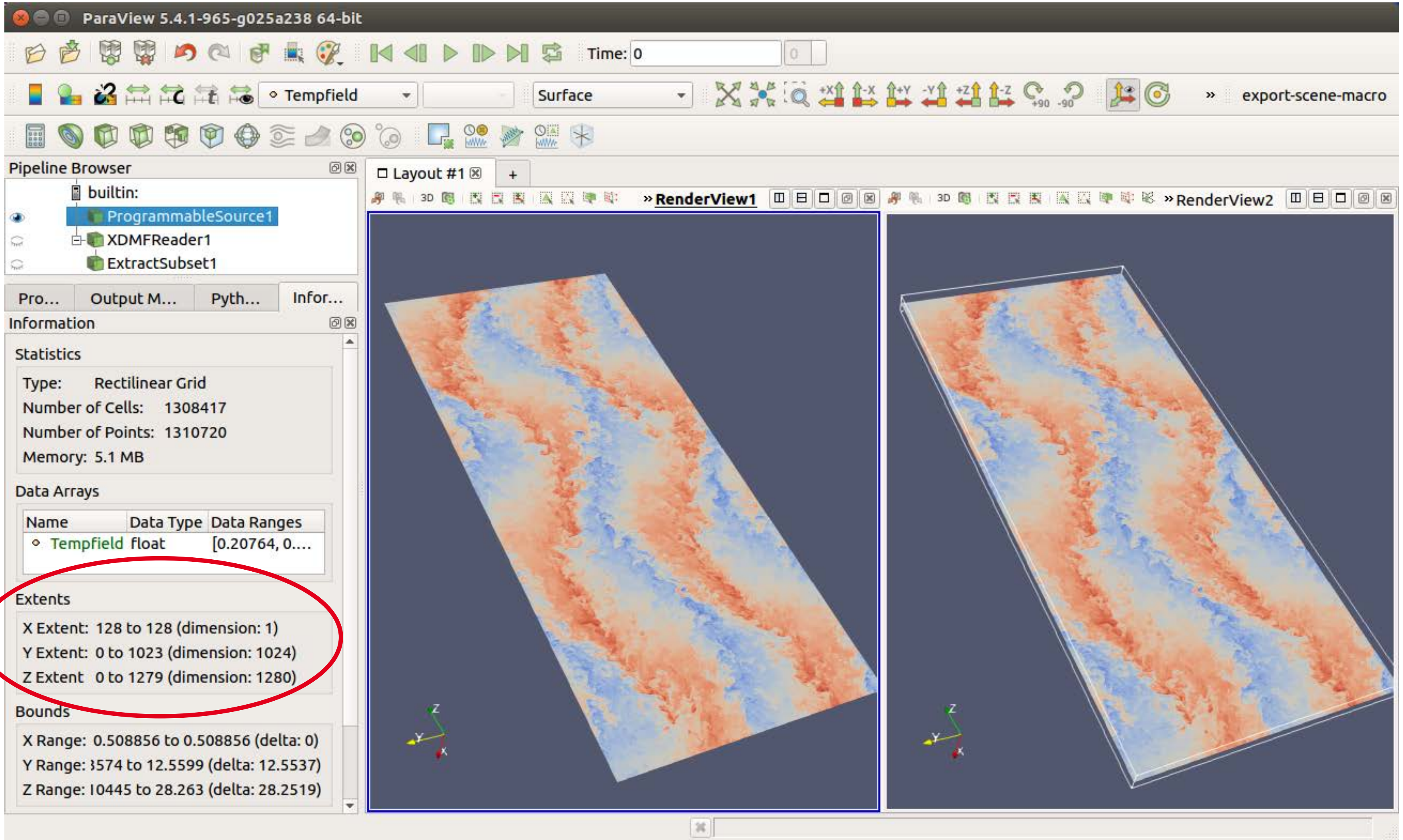
A single slice through it occupies 5.2 MB

Total = 1.3GB



Only read a 1D grid
1*1024*1280

Occupies 5.2 MB

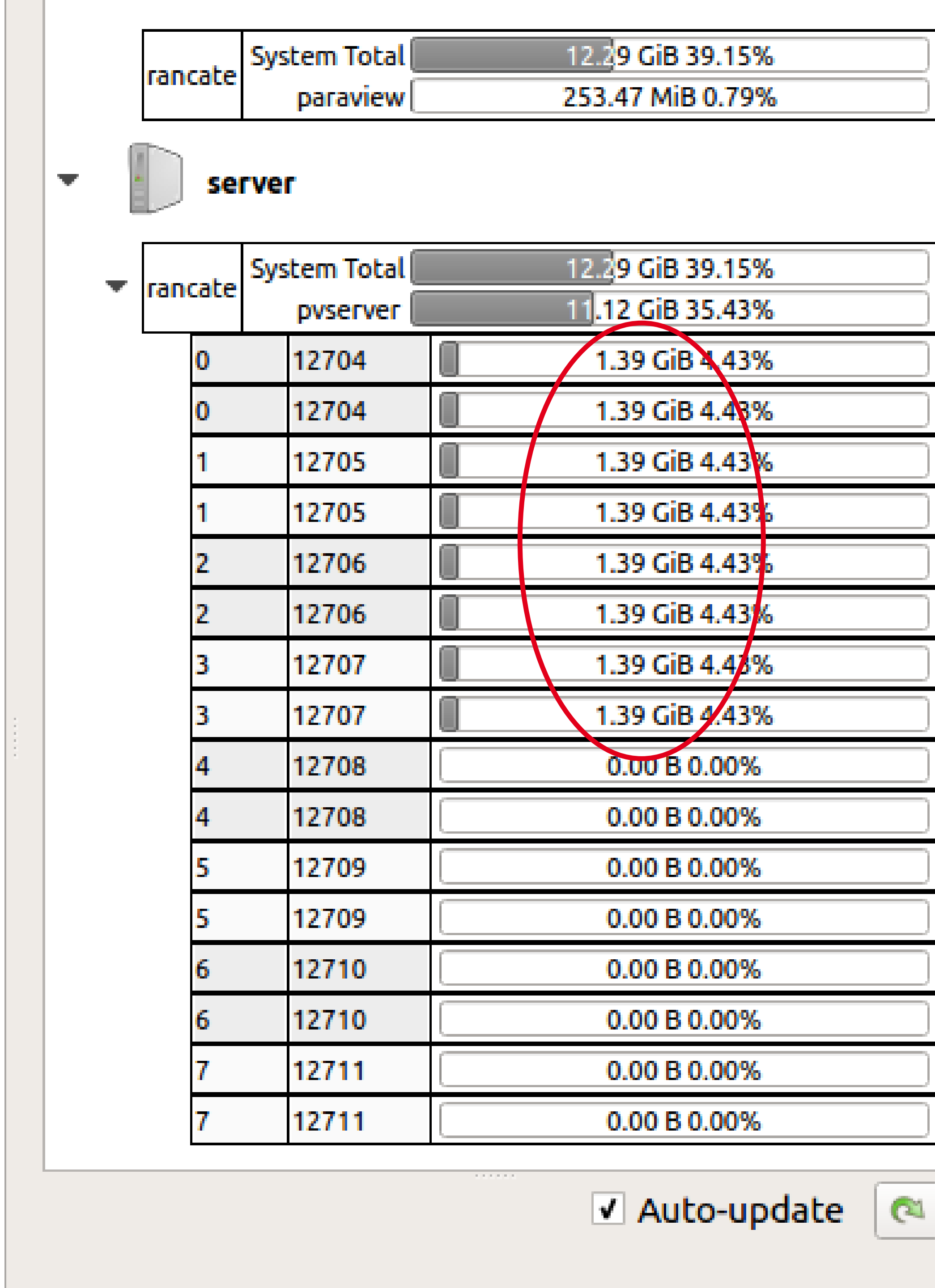


Total = 5.2 MB

Substitute XDMF with h5py

- Reduced space I/O operations are easily implemented with numpy/h5py

- ParaView’s XDMF3 reader does not distribute data among parallel servers. It **duplicates** the data everywhere.



Details about the rendering

- The computation took place on a rectilinear grid.
- But ParaView implements Volume Rendering of rectilinear data using hexahedra cells, i.e using the very slow unstructured cell volume rendering

- Re-sampled the data to a regular, cartesian grid…and while doing that, change the resolution from 257x1024x1280 to 256x1024x1280

- In summary…the data I/O strategy was developed on-the-fly during several weeks of testing, benchmarking.
- The final movie production took place in a few hours with temporal and spatial parallelism.

Volume rendering was tested with three parallel libraries

- Kitware's native GPU-based renderer
- NVIDIA IndeX GPU-based renderer
- Intel OSPRay CPU-based renderer (12-core Broadwell (two threads per core))

- Running on 16 nodes:
  - ParaView's native renderer runs at 106 frames/sec
  - Index IndeX runs at 45 frames/sec
  - OSPRay runs at 21 frames/sec

Volume rendering was tested with three parallel libraries

- Running on 1 node, with a 1024^3 volume:
  - ParaView converts 32-bit floats to 16-bit integers

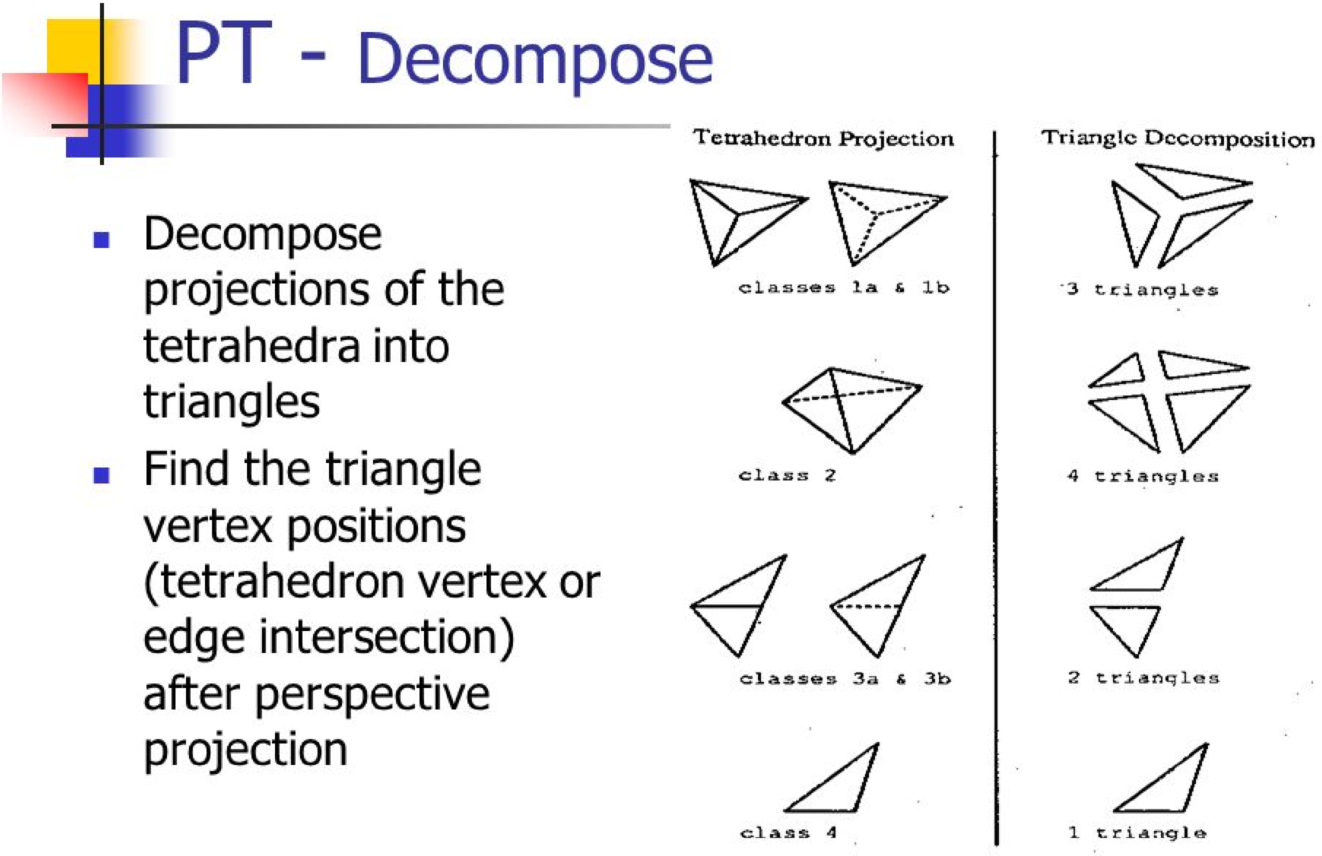
- Memory consumption by GPU (see nvidia-smi)
  - Kitware's native renderer: 2722 MiB
  - NVIDIA Index Volume Renderer: 15948 MiB

GPU-based rendering will scale across nodes, at the condition that it fits in memory

Intel OSPRay CPU-based renderer can use a lot more memory (thus read much bigger data). It's on the CPU.

Volume rendering of Unstructured grid

- The Projected Tetrahedra mapper is almost 30 years old.
- Although today's implementation are very sophisticated, using OpenGL shaders and other GPU optimizations techniques, it remains a slow rendering technique because of the sheer size of the triangles in the scene



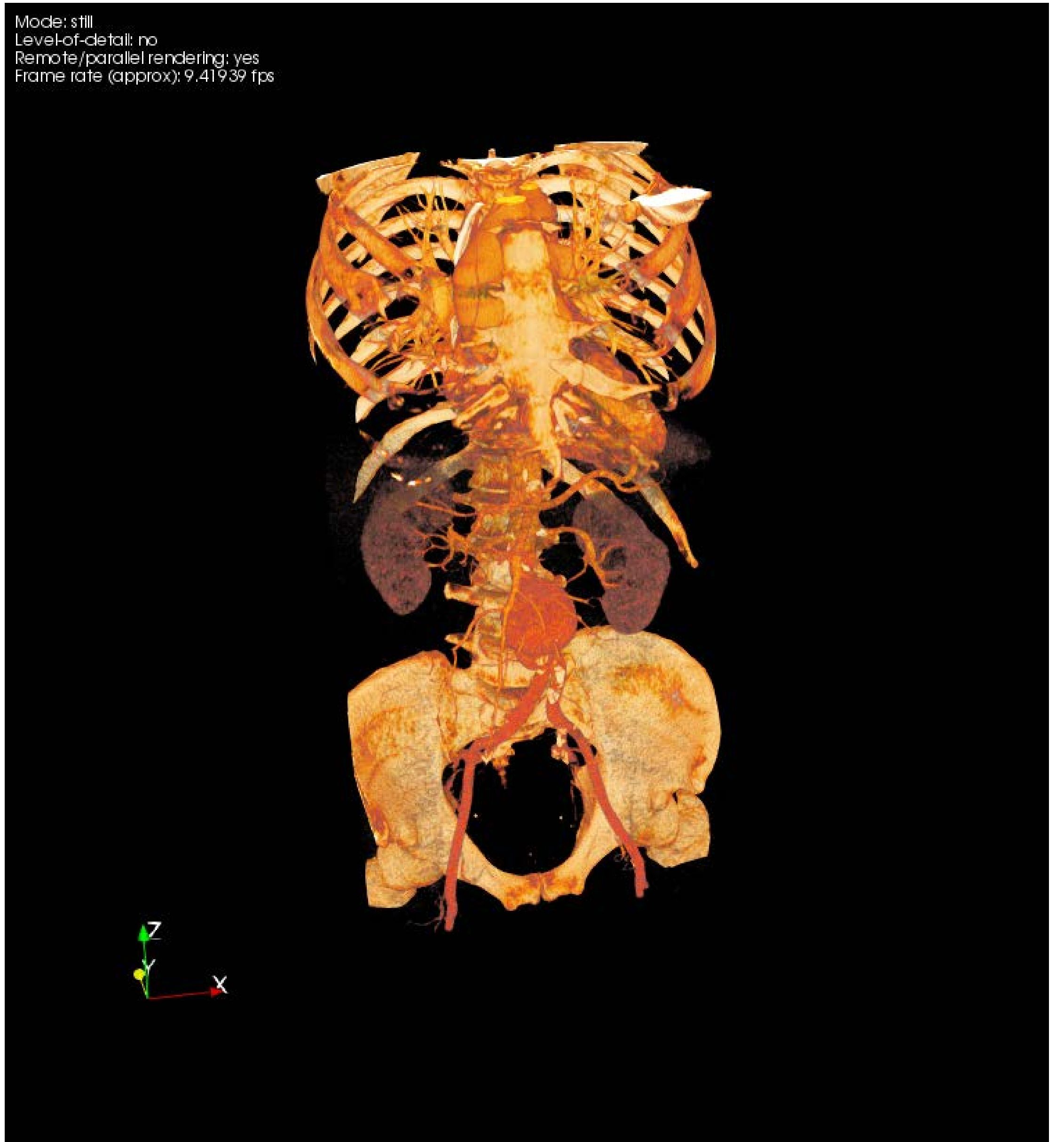
Volume rendering Unstructured grids with NVIDIA's IndeX

Although currently limited to 32-bit floats, I was able to volume render a mesh of 770 million tetrahedra, in real time, using 64 GPUs

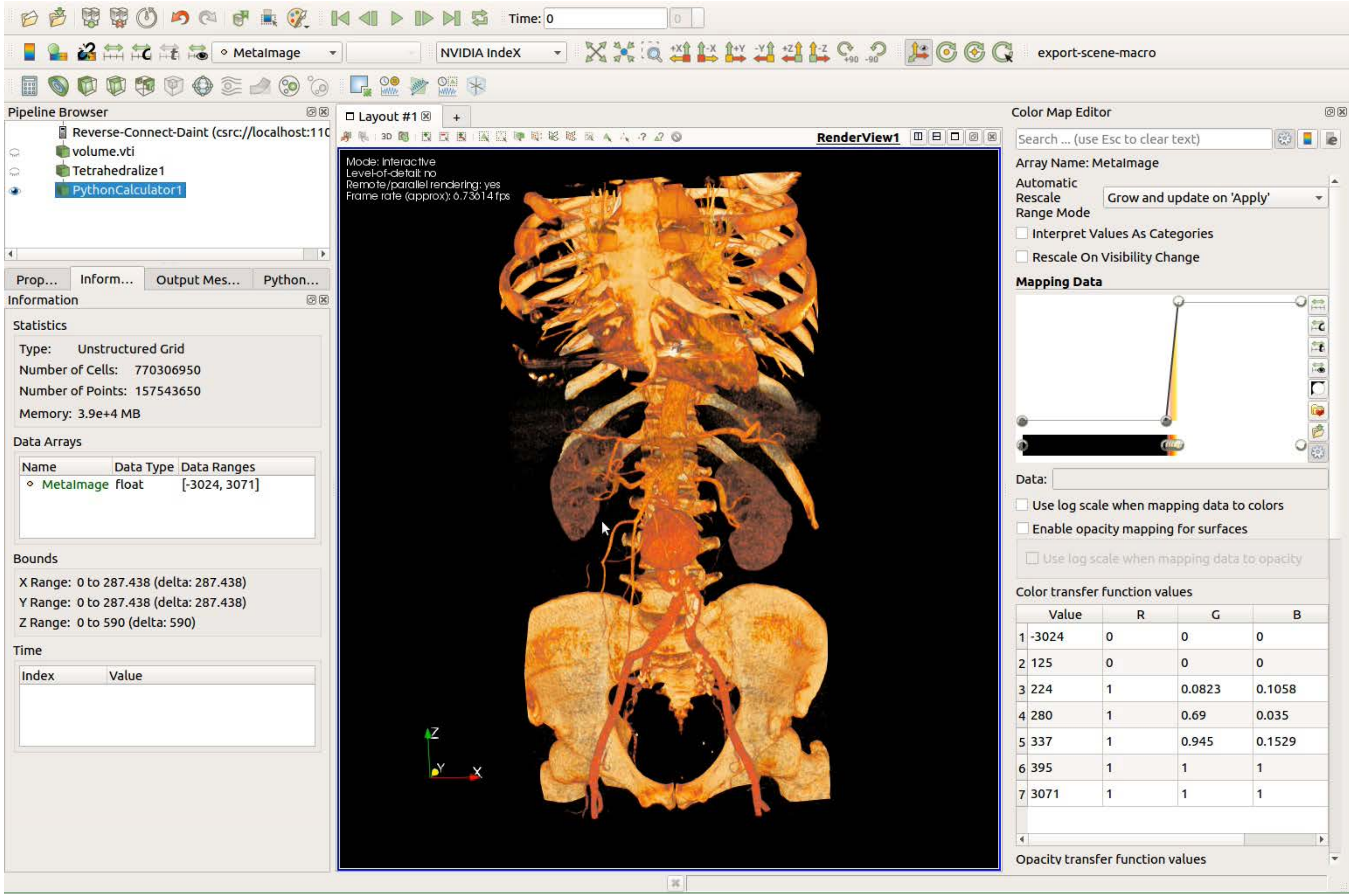
https://developer.nvidia.com/index

http://www.nvidia.com/object/index-paraview-plugin.html

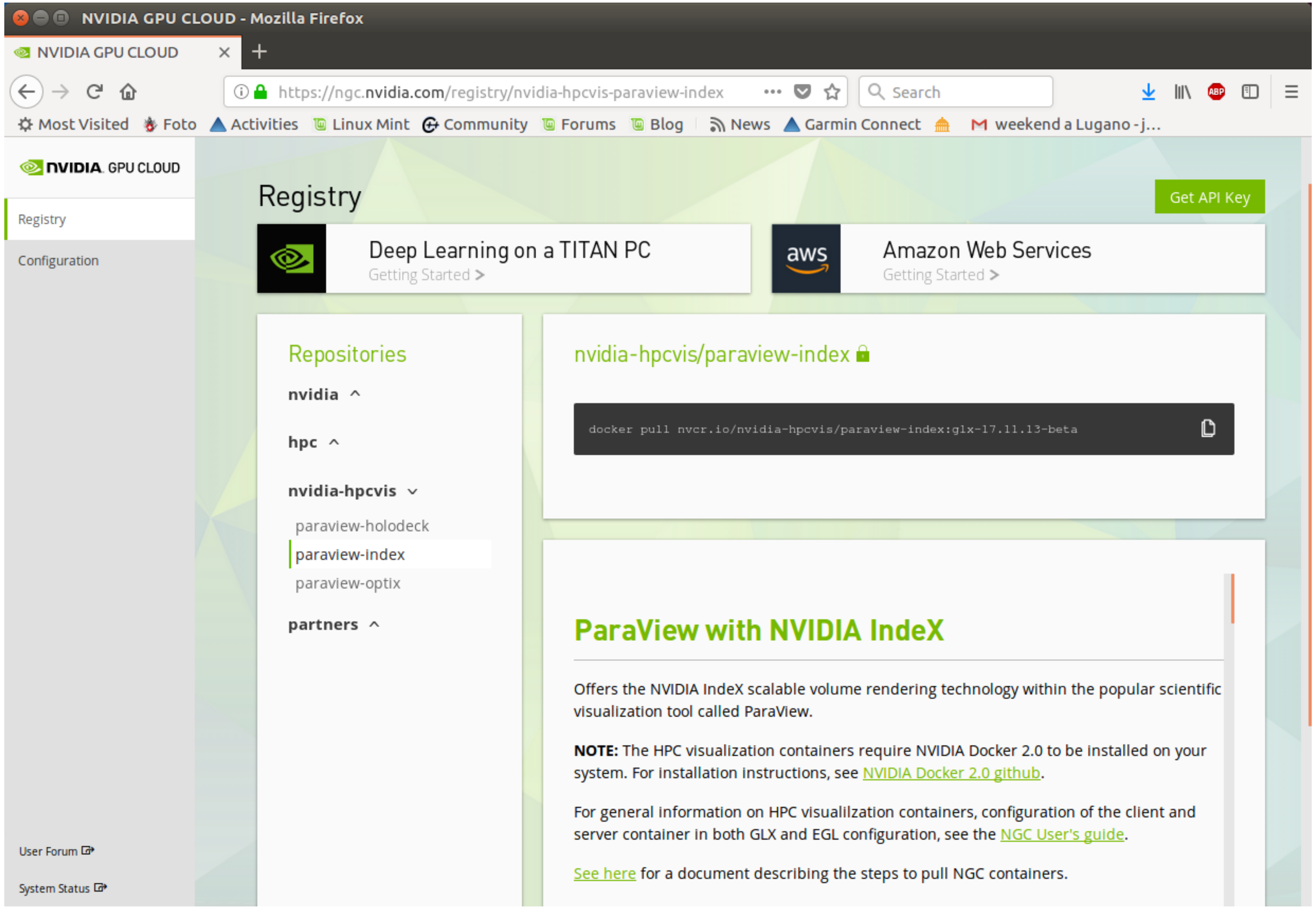
ftp://ftp.cscs.ch/out/jfavre/NVIDIA/IndexVolumeRendering.avi



Real time screen capture

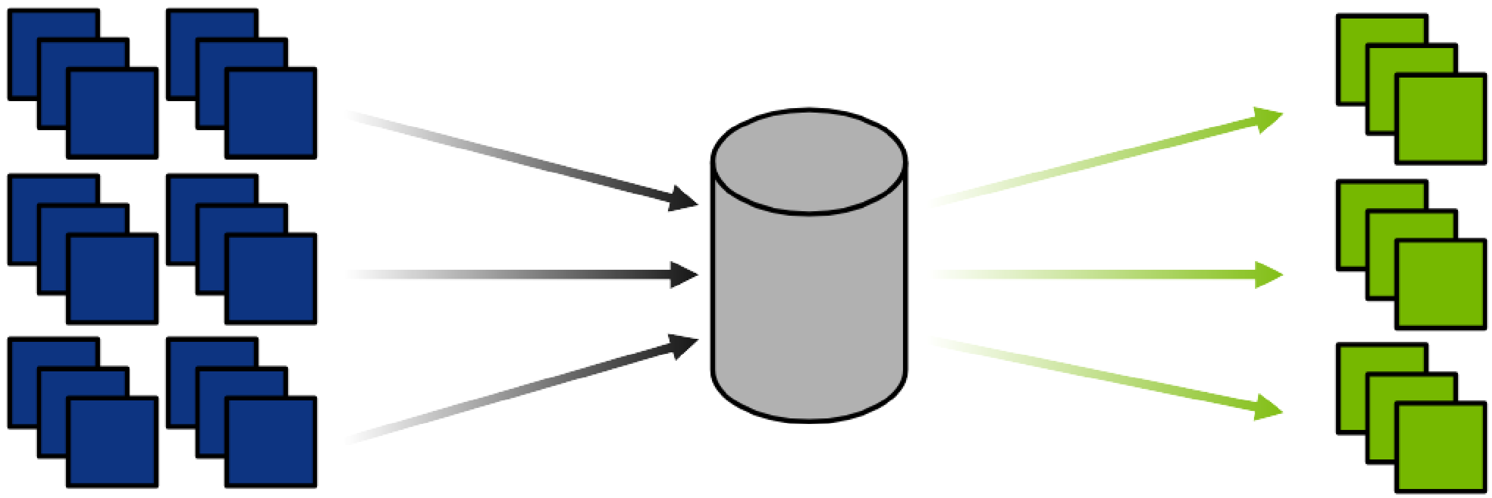


Curious to try?



Where is IndeX leading us?

TRADITIONAL VISUALIZATION PIPELINE

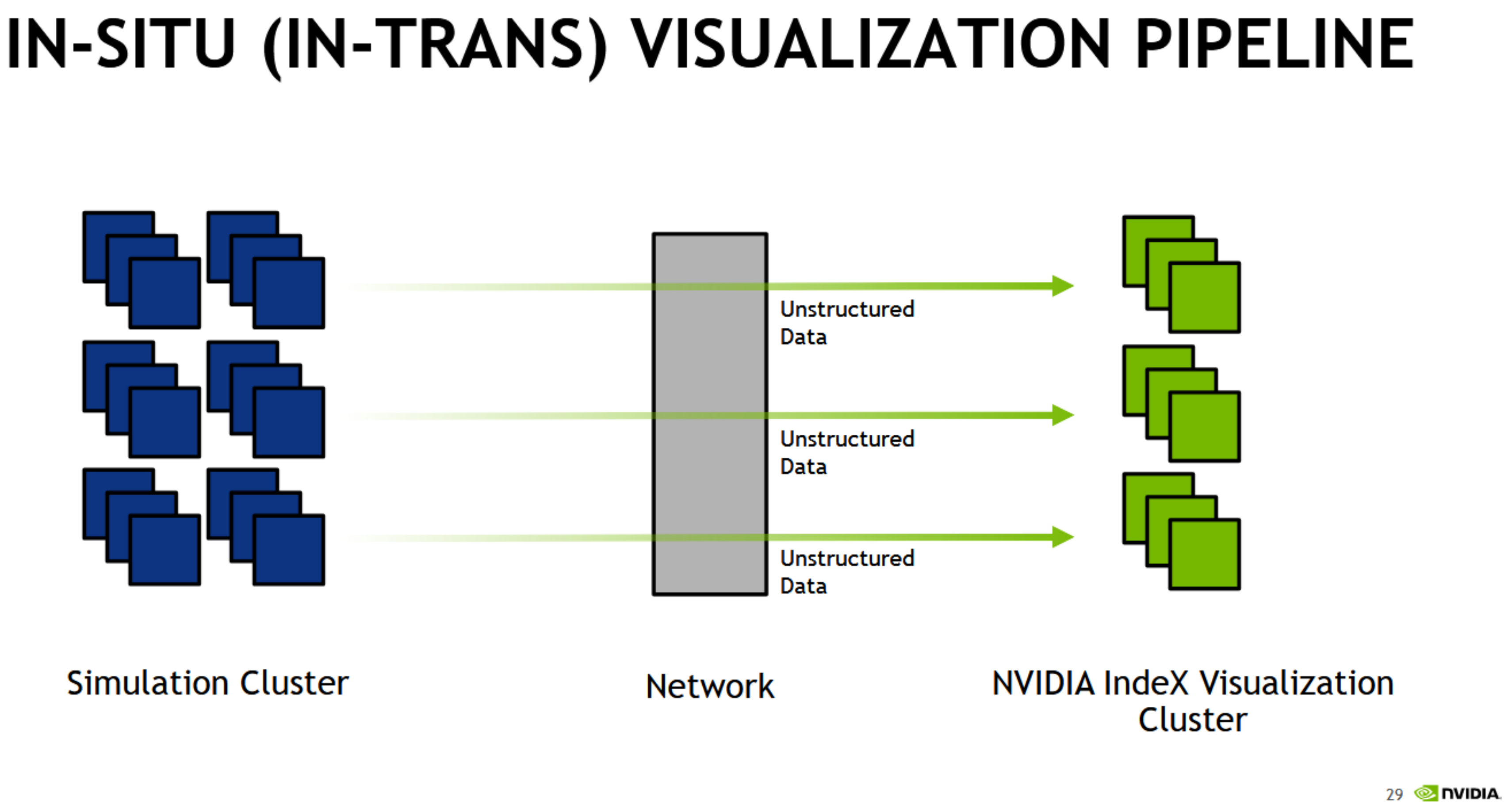


Simulation Cluster

Data Storage
e.g. Unstructured Data

NVIDIA IndeX Visualization Cluster

Where is IndeX leading us?



- A library such as NVIDIA’s Index would sit on the “end” of the Visualization pipeline, i.e. the rendering side

- A more general approach is to be able to embed the standard visualization (filters and geometry mappers) closer to the simulation’s data.

When there is too much data…

- Several strategies are available to mitigate the data problem:
- read less data:
    - multi-resolution,
    - on-demand streaming,
- out-of-core, etc...

- Do no read data from disk but from memory:

***in-situ* visualization**

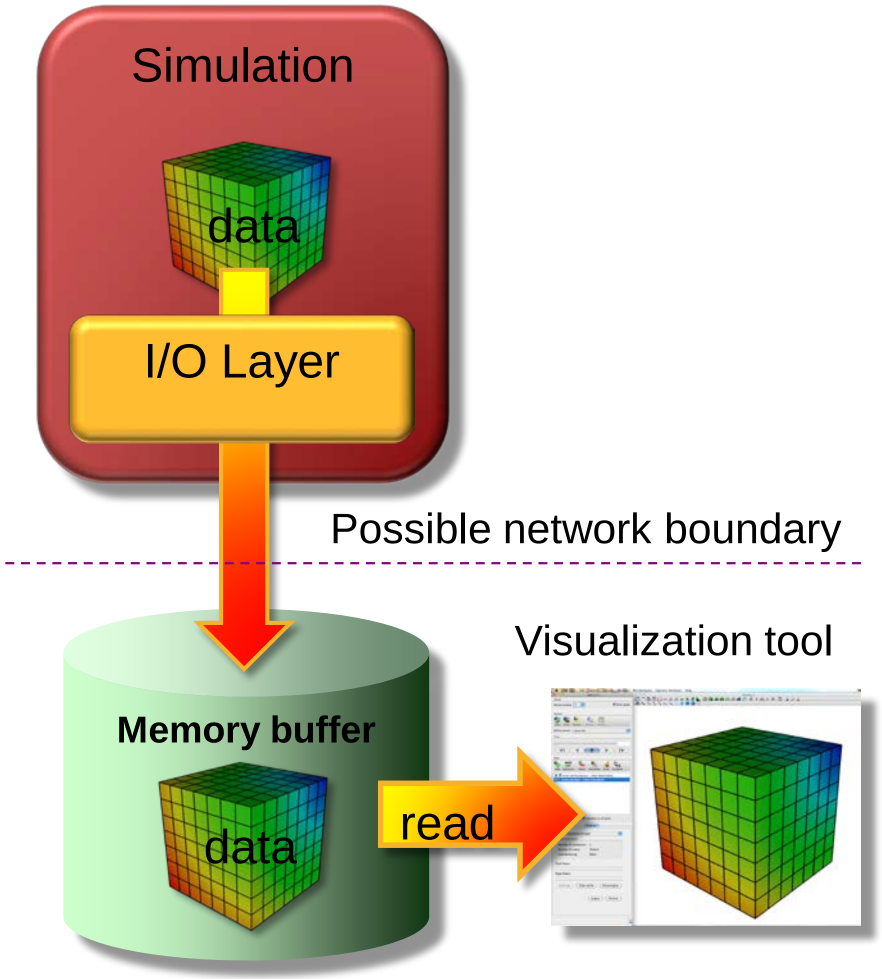
in-situ visualization

Instrument parallel simulations to:

- Eliminate (or reduce) I/O to and from disks
- Use all grid data with or without ghost-cells
- Have access to all time steps, all variables
- Use the available parallel compute nodes, or a secondary resource

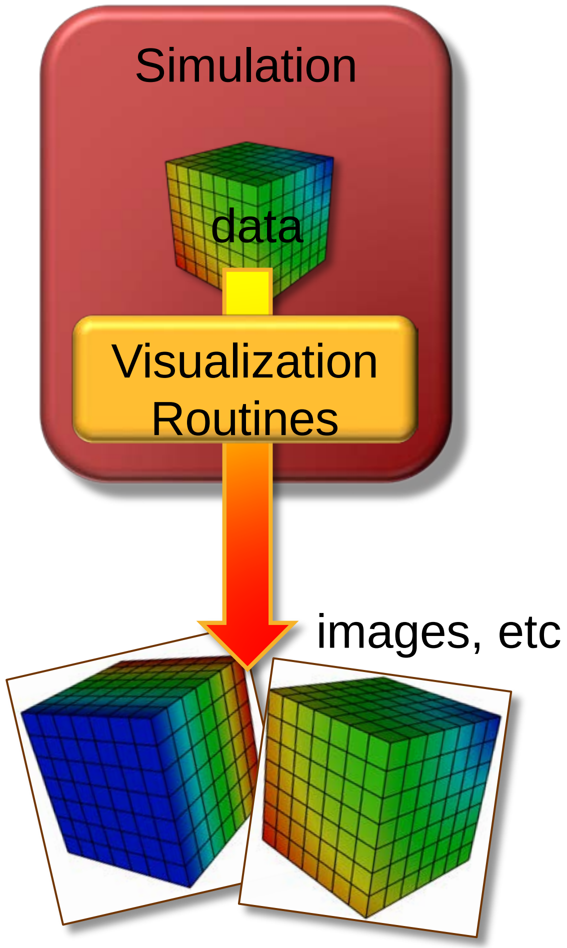
Loosely Coupled in-situ Processing (old definition)

- I/O layer stages data into secondary memory buffers, possibly on other compute nodes
- Visualization applications access the buffers and obtain data
- Separates visualization processing from simulation processing
- Copies and moves data



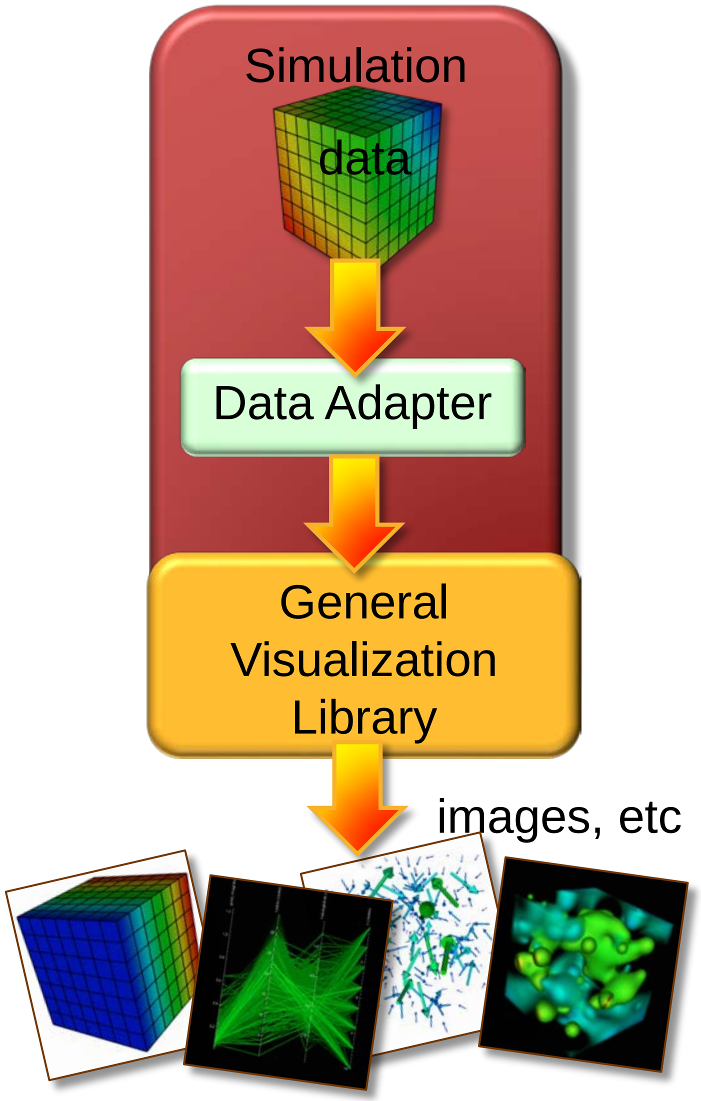
Tightly Coupled *Custom* in-situ Processing (old definition)

- Custom visualization routines are developed specifically for the simulation and are called as subroutines
  - Create best visual representation
  - Optimized for data layout
- Tendency to concentrate on very specific visualization scenarios
- *Write once, use once*



Tightly Coupled *General* in-situ Processing (old definition)

- Simulation uses data adapter layer to make data suitable for general purpose visualization library
- Rich feature set can be called by the simulation
- Operate directly on the simulation's data arrays when possible
- *Write once, use many times*



The In Situ Terminology Project
project lEADER: Hank Childs

Next 2 slides thanks to Hank Childs

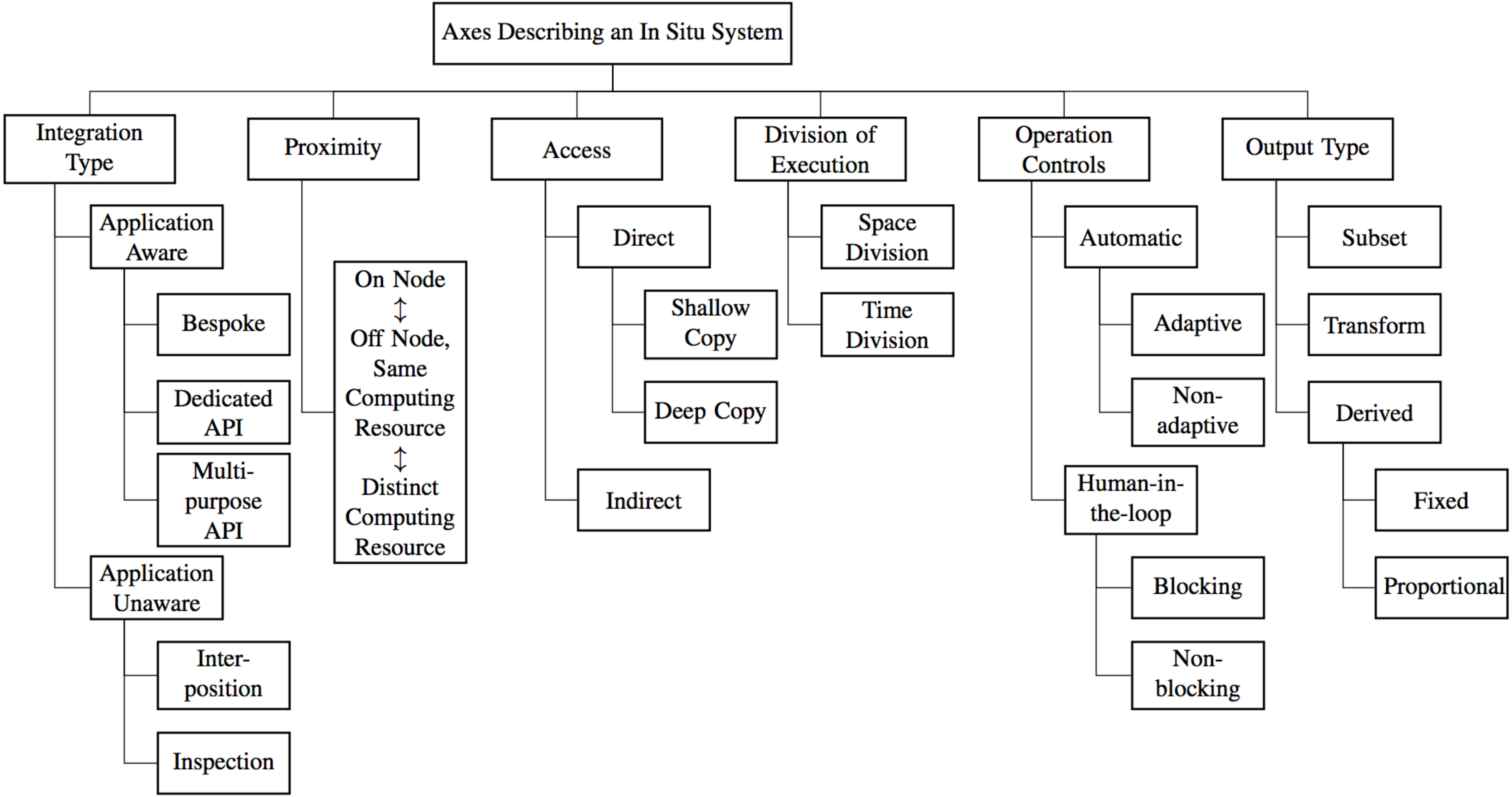
CSCS

ETH zürich

Project Participants (to date)

Participants

- Hasan Abbasi, ORNL
- Sean Ahern, CEI
- Jim Ahrens, LANL
- Marco Ament, Karlsruhe Institute of Technology (KIT)
- Andy Bauer, Kitware
- Janine Bennett, SNL-CA
- Wes Bethel, LBNL
- Peer-Timo Bremer, LLNL & Univ. of Utah
- Eric Brugger, LLNL
- Chun-Ming (Jimmy) Chen, The Ohio State University
- Hank Childs, Univ. of Oregon & LBNL
- Amit Chourasia, SDSC
- Joseph Cottam, Indiana Univ.
- Matthieu Dorier, Argonne
- Soumya Dutta, The Ohio State University
- Earl Duque, Intelligent Light
- Jean Favre, CSCS
- Tom Fogal, NVIDIA
- Steffen Frey, Stuttgart
- Berk Geveci, Kitware
- Cyrus Harrison, LLNL
- Bernd Hentschel, RTWH-Aachen
- Joseph Insley, Argonne
- Chris Johnson, Univ. of Utah
- Aaron Knoll, Univ. of Utah
- Scott Klasky, ORNL
- James Kress, Univ. of Oregon
- Matt Larsen, Univ. of Oregon & LLNL
- Laura Lediaev, Univ. of Utah
- Jay Lofstead, SNL-NM
- Kwan-Liu Ma, UC Davis
- Jeremy Meredith, ORNL
- Ken Moreland, SNL-NM
- Paul Navratil, UT-Austin
- Patrick O'Leary, Kitware
- Manish Parashar, Rutgers
- Valerio Pascucci, Univ. of Utah
- John Patchett, LANL
- Tom Peterka, ANL
- Steve Petruzza, Univ. of Utah
- David Pugmire, ORNL
- Michel Rasquin, Cenaero
- Silvio Rizzi, Argonne
- David Rogers, LANL
- Franz Sauer, UC Davis
- Dave Semeraro, UT-Austin
- Han-Wei Shen, The Ohio State University
- Rob Sisneros, NCSA
- Venkat Vishwanath, ANL
- Chaoli Wang, Notre Dame
- Ingo Wald, Intel
- Gunther Weber, LBNL
- Daniel Weiskopf, Stuttgart
- Brad Whitlock, Intelligent Light
- Matt Wolf, Georgia Tech
- Hongfeng Yu, UN-L
- Sean Ziegeler, DoD



Axes Describing an In Situ System
Integration Type
Application Aware
Bespoke
Dedicated API
Multi-purpose API
Application Unaware
Inter-position
Inspection
Proximity
On Node
↕
Off Node, Same Computing Resource
↕
Distinct Computing Resource
Access
Direct
Shallow Copy
Deep Copy
Indirect
Division of Execution
Space Division
Time Division
Operation Controls
Automatic
Adaptive
Non-adaptive
Human-in-the-loop
Blocking
Non-blocking
Output Type
Subset
Transform
Derived
Fixed
Proportional

In-situ references

- ParaView Catalyst
- VisIt libsim
- ADIOS and GLEAN both provide tools for in situ I/O and some analysis
- The SENSEI project
  - enables connection of simulation data sources to visualization and analysis back ends
  - data model enables viz & analysis codes to access data through a unified API

SENSEI – http://www.sensei-insitu.org/

Software repo – https://gitlab.kitware.com/sensei/sensei

GLEAN – https://www.alcf.anl.gov/glean

ADIOS – https://www.olcf.ornl.gov/center-projects/adios/

VisIt/Libsim – https://www.visitusers.org/index.php?title=Category:Libsim

ParaView Catalyst – http://www.paraview.org/in-situ/

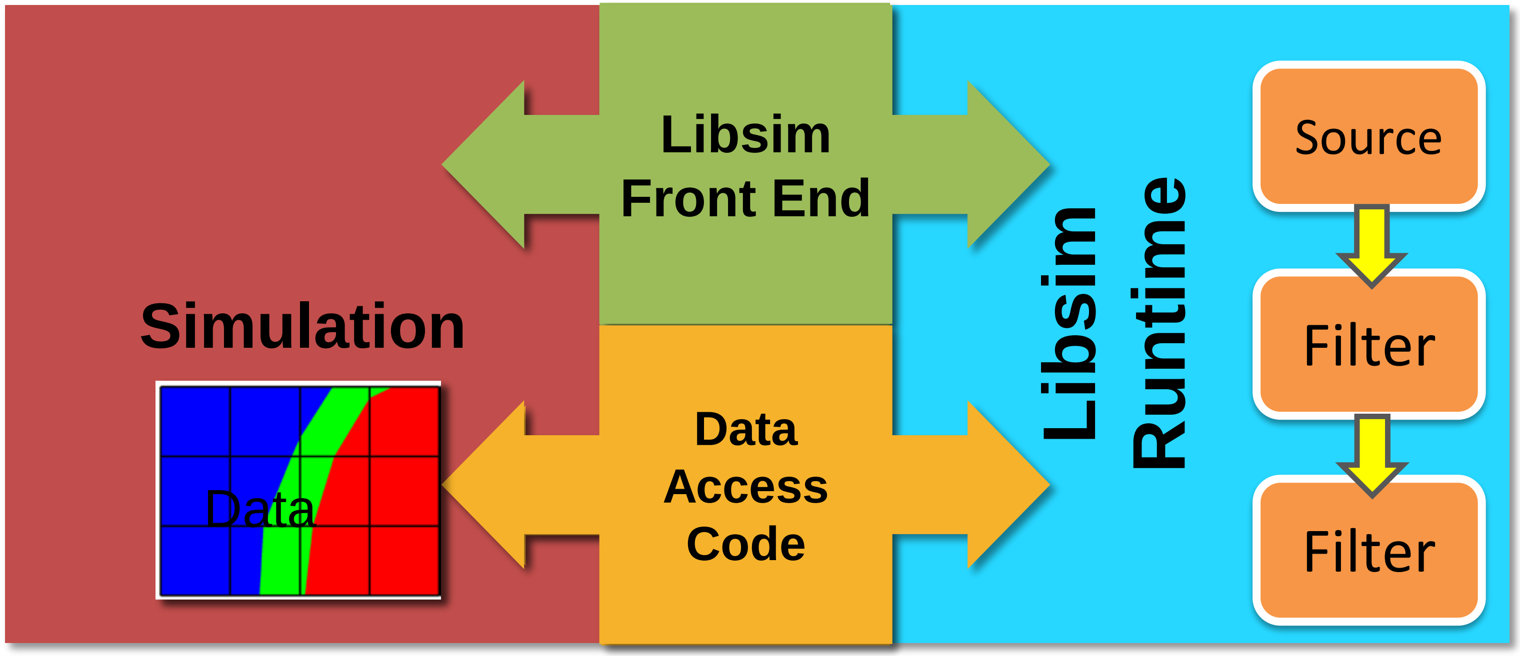
Using the in-situ terminology defined earlier

Visit's libsim, can be defined as being

- Application-aware
- On-the-node
- With direct access (shallow copy or deep copy) to the data
- Using Time-division
- With human-in-the-loop (blocking) or batch execution
- Providing different outputs (subset, transforms, derived, etc)

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.



In situ - interactive - Processing Workflow

1. The simulation code launches and starts execution
2. The simulation regularly checks for connection attempts from visualization tool
3. The visualization tool connects to the visualization
4. The simulation provides a description of its meshes and data types
5. Visualization operations are handled via Libsim and result in data requests to the simulation

Instrumenting a Simulation

Additions to the source code are usually minimal, and follow three incremental steps:

Initialize Libsim and alter the simulation's main iterative loop to listen for connections from VisIt.

Create *data access callback* functions so simulation can share data with Libsim.

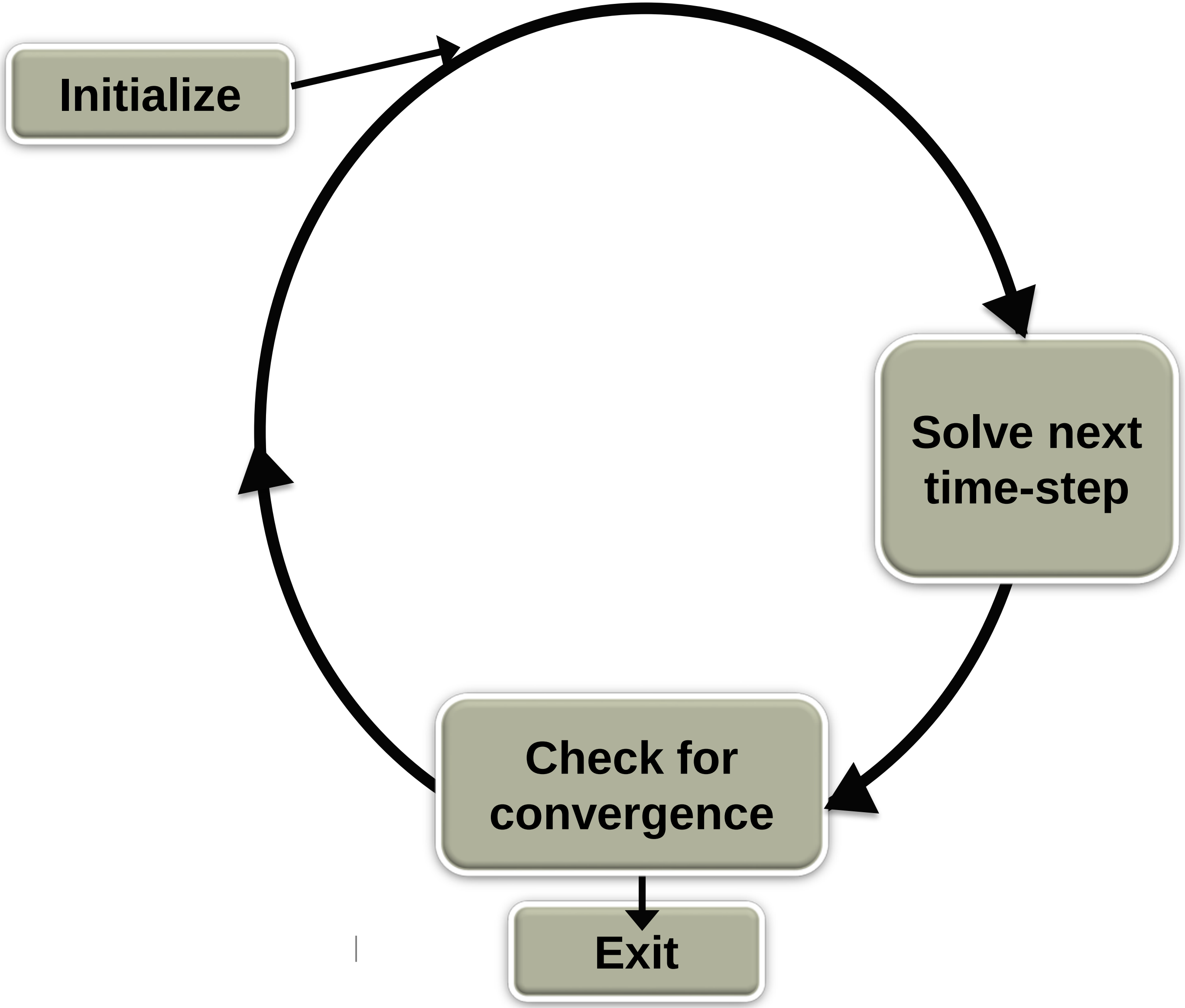
Add control functions that let VisIt steer the simulation.

Instrumenting Application’s flow diagram (before and after

Connection to the visualization library is optional

Execution is *step-by-step* or in *continuous* mode

Live connection can be closed and re-opened at later time

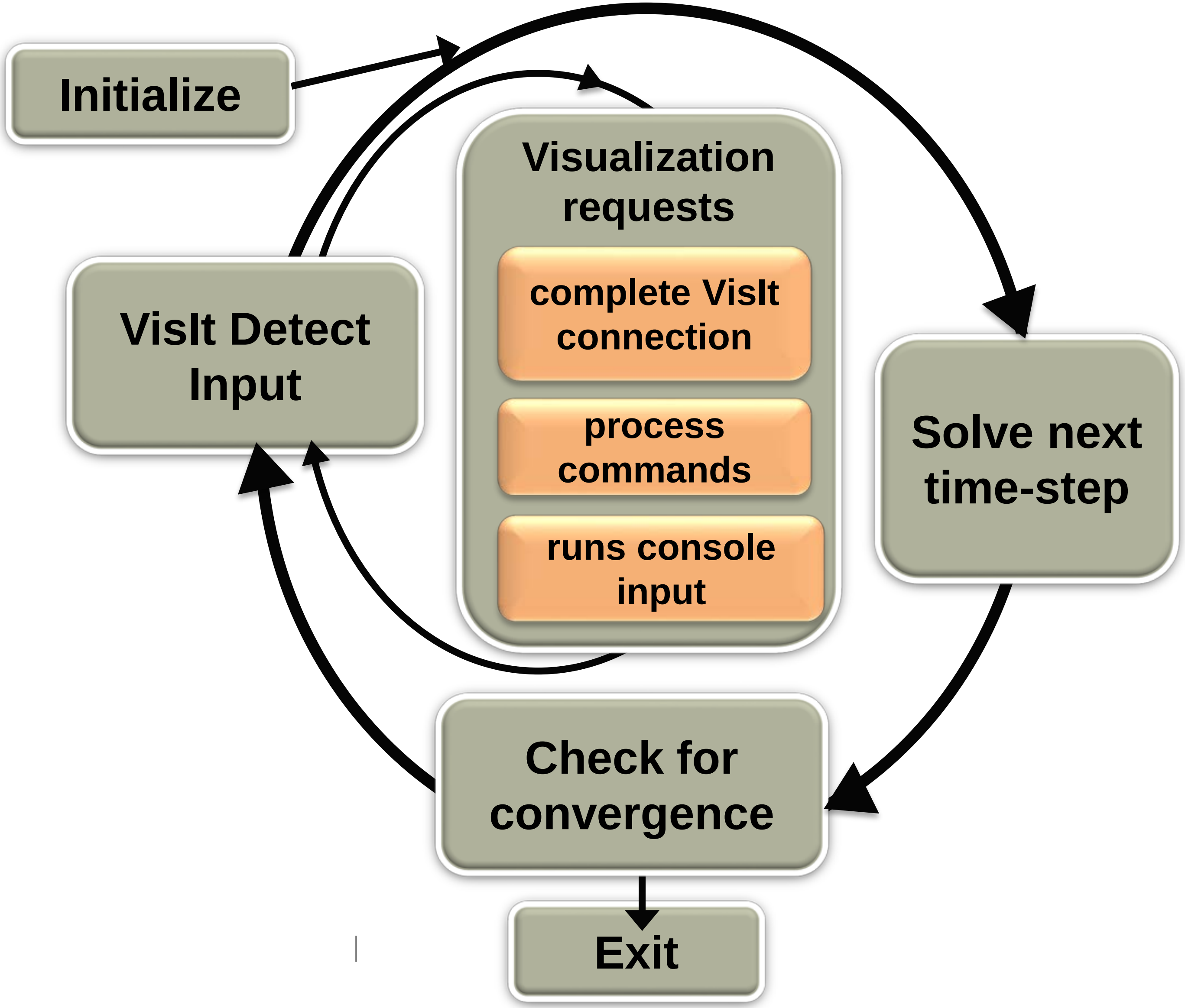


VisIt in-the-loop

- Libsim opens
  a socket and writes out
  connection parameters

VisItDetectInput checks for:

- Connection request
- VisIt commands
- Console input



In situ – batch - Processing Workflow

1. The simulation code launches and starts execution
2. The simulation explicitly loads the libsim runtime library
3. Visualization operations are handled via Libsim and will be processed at the end of (each, or at regular intervals) compute iteration.

Details on the wiki.

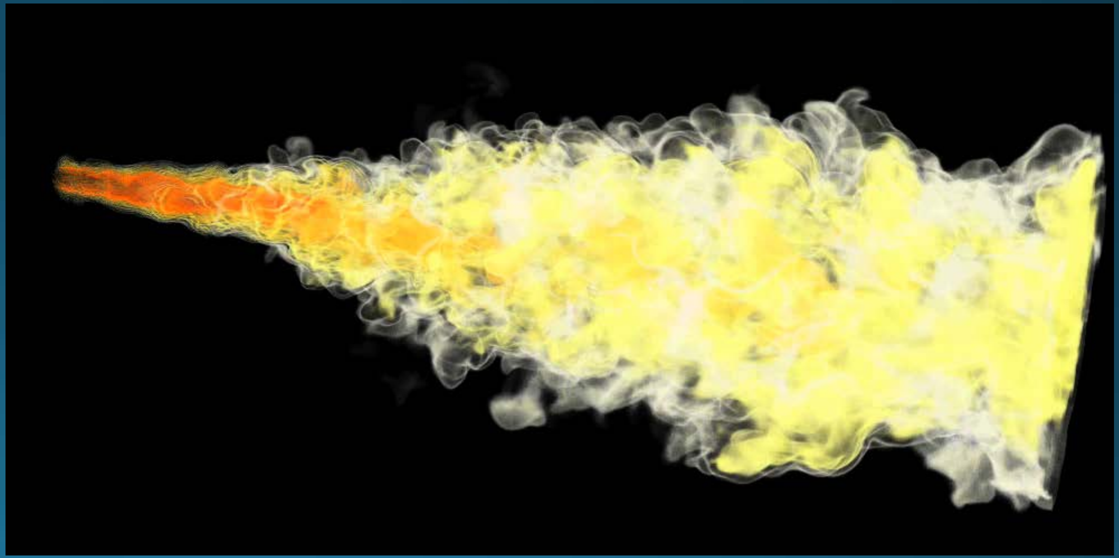
The Uintah – VisIt coupled workflow

Allen R. Sanderson, Alan Humphrey,
John Schmidt, Chuck Hansen, Martin Berzins
Scientific Computing and Imaging Institute,
The University of Utah, Salt Lake City, USA

SIAM CSE 2017

Uintah

- The Uintah software suite is a set of libraries and applications for simulating and analyzing complex chemical and physical reactions.
- http://www.uintah.utah.edu/projects.html
- **Clean Energy from Fossil Fuels –** model various energy technologies from traditional air-fired coal, oxy-fired coal/natural gas, fluidized bed coal combustion and coal gasification to more exotic coal technologies such as chemical looping and under ground thermal treatment.



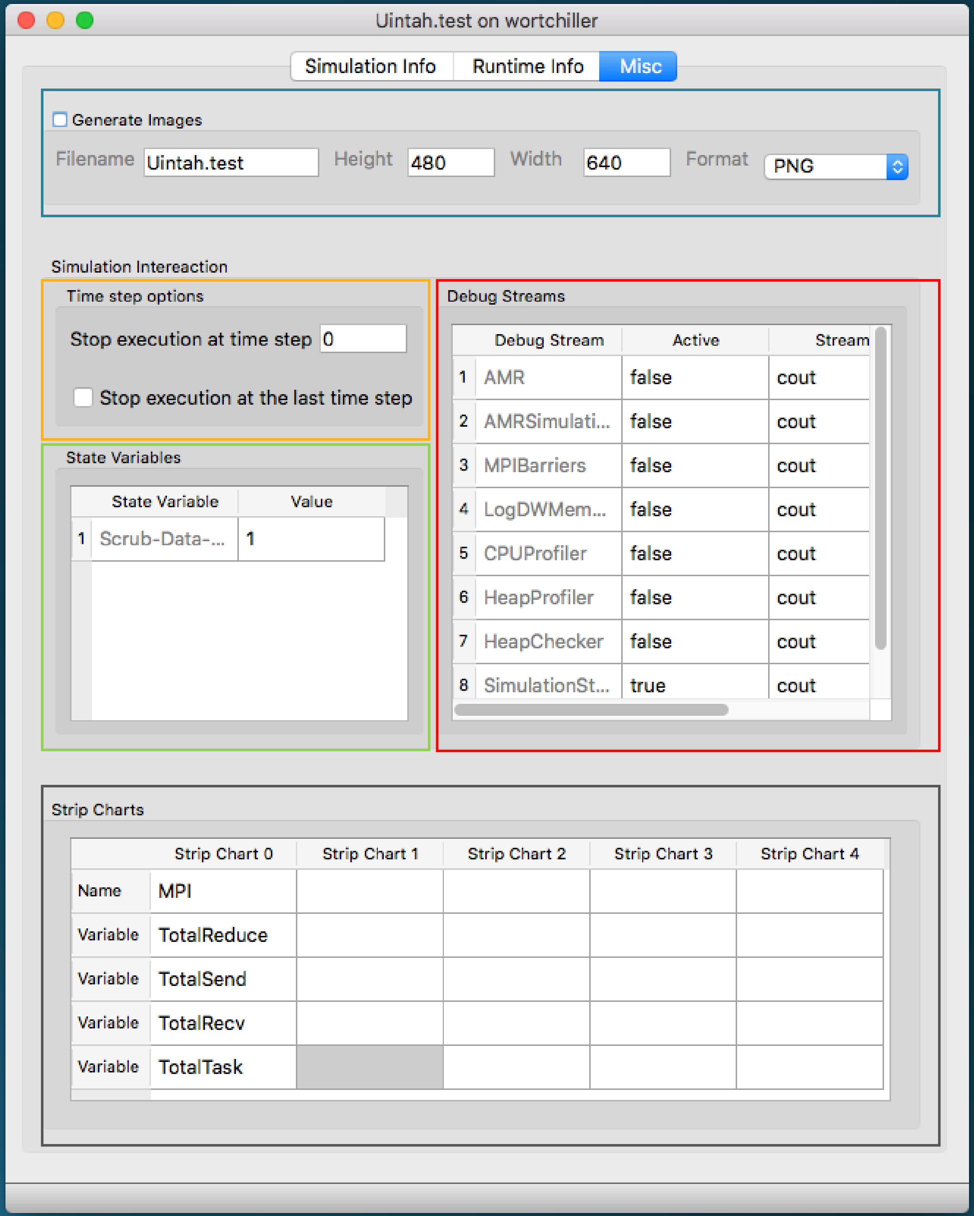
VisIt – libsim

- Allows a connection between the application and VisIt via sockets.
  - Dynamically loads VisIt's runtime libraries that allows the simulation to act as VisIt's compute engine.
  - Uses a middle layer to move application data to VisIt.

- Using VisIt's libsim as part of the work flow for the:
  - **Runtime layer** where computer scientists develop and maintain infrastructure – i.e. memory, mpi scheduling.
  - **Application layer** where domain scientists develop and maintain the meshing and physics.
  - **User layer** where scientists visualization and analyze the simulation results.

VisIt – Custom UI

- Nice to have features.
  - Generate image frames automatically
    - Used when data is too large to dump to disk.
  - Stopping the execution at specific time steps.
    - Useful for debugging
  - Global state variables
    - Controlling Data Warehouse intermediate variables – used for visual debugging.
  - Controlling debug streams.
    - Direct pointer link is shared with libsim.
  - Strips Charts
    - Allows for the monitoring of global values over the life time of the simulation.



Parallel visualization as an iceberg

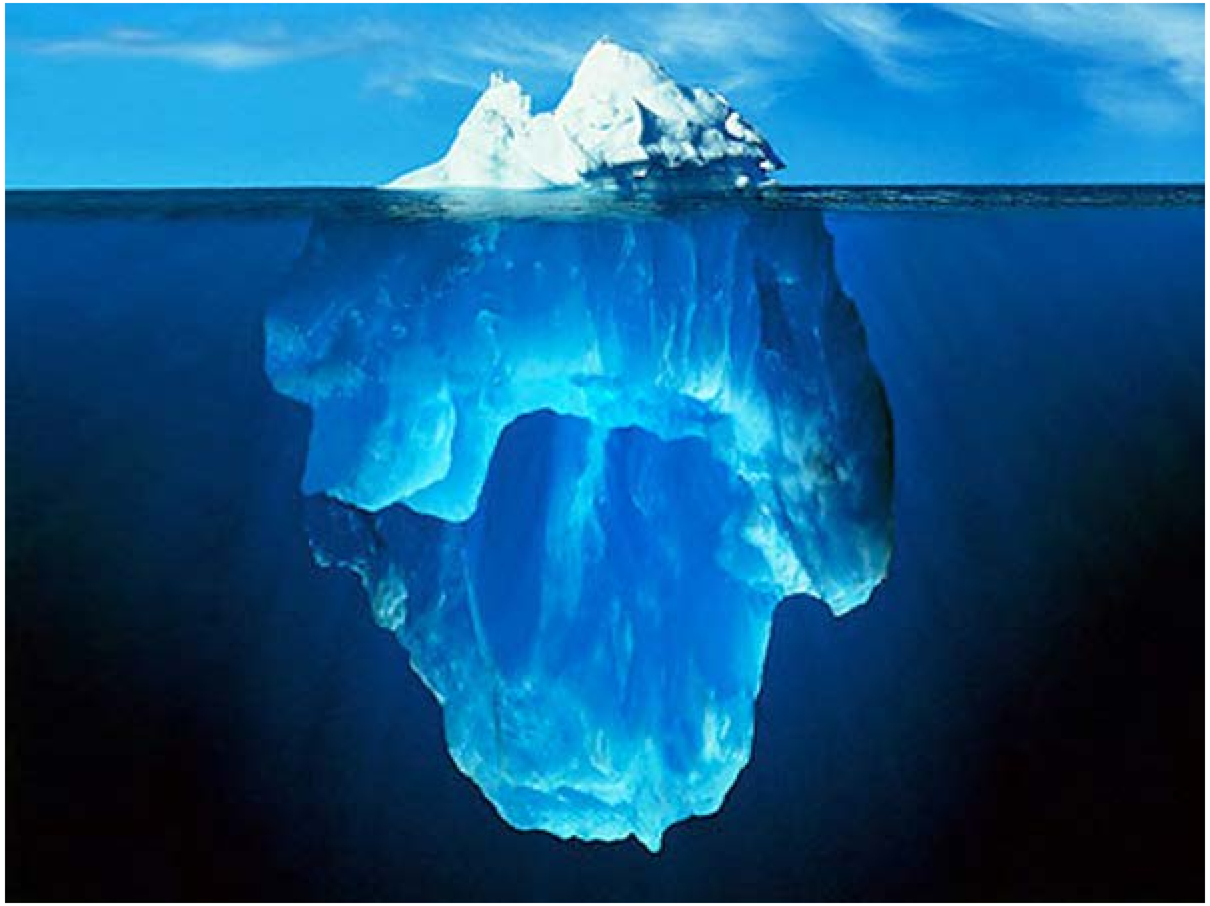
Les I/O parallèles

La gestion des pipelines de visualization

Le format des données internes

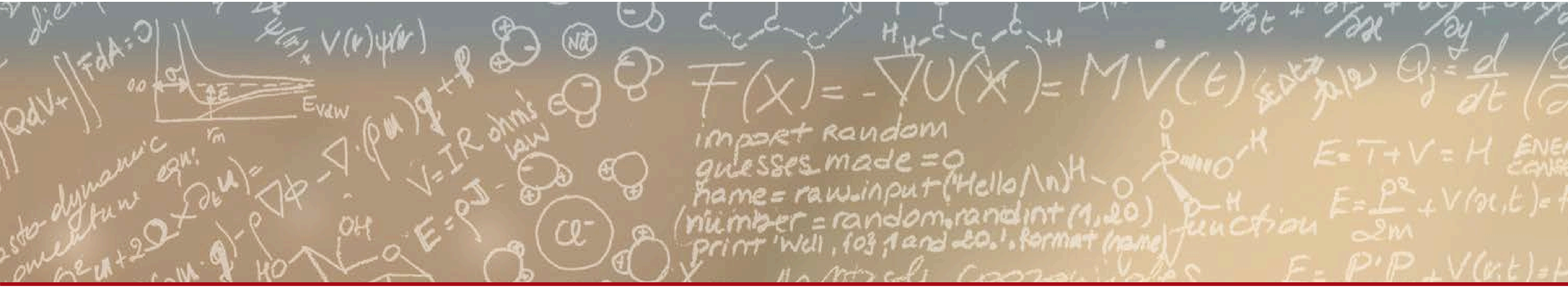
La gestion des resources mémoire (CPU et GPU)

Le couplage in-situ des codes de simulation



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



End

Thank you for your attention.