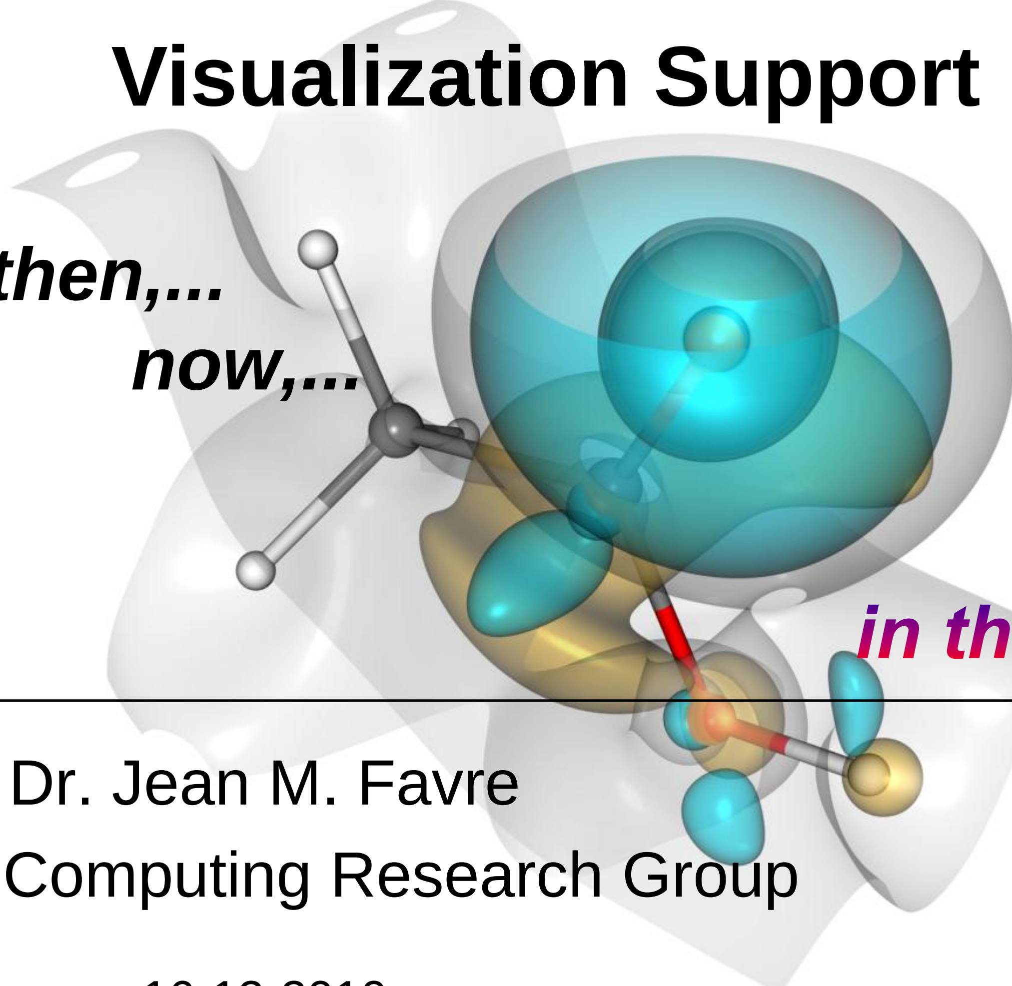
CSCS

Swiss National Supercomputing Centre

Visualization Support



then,...

now,...

in the future…

Dr. Jean M. Favre

Scientific Computing Research Group

16-12-2010

ETH

Eidgenössische Technische Hochschul

Swiss Federal Institute of Technology

In this talk,

- Summarize 15 years of experience as Visualization Scientist
- Give examples of “Development”
- Offer some fundamental principles
- Present some parallel visualization challenges

and,

- Present current activities (in-situ visualization)
- Offer some thoughts about the future.

Education / Tutorials

- Data formatting (HDF5, NetCDF)
- ParaView (in and outside of Switzerland)
- VisIt (proposal for ISC'2011)
- AVS/Express (with CINECA)
- Molekel (real-time movie capture)

- Parallel I/O (PHDF5, NetCDF4, ADIOS, MPI-I/O)
- New techniques (parallel coordinates, information visualization, in-situ visualization)

What I have pushed very hard

- Standard data formats (HDF5 and others)
- No double copies, i.e. native interface.
  - The Visualization software ingests the data *as-is.*
- *Parallel Reading* is a necessary and obligatory first step in the parallel visualization pipeline.
- Repeatable visualization (script-based with history)

- The customer is king:
  - Data interpretation is the scientist's *raison d'être*
  - My role is [only] to provide all possible ways to interpret the data and do "scientific discovery"

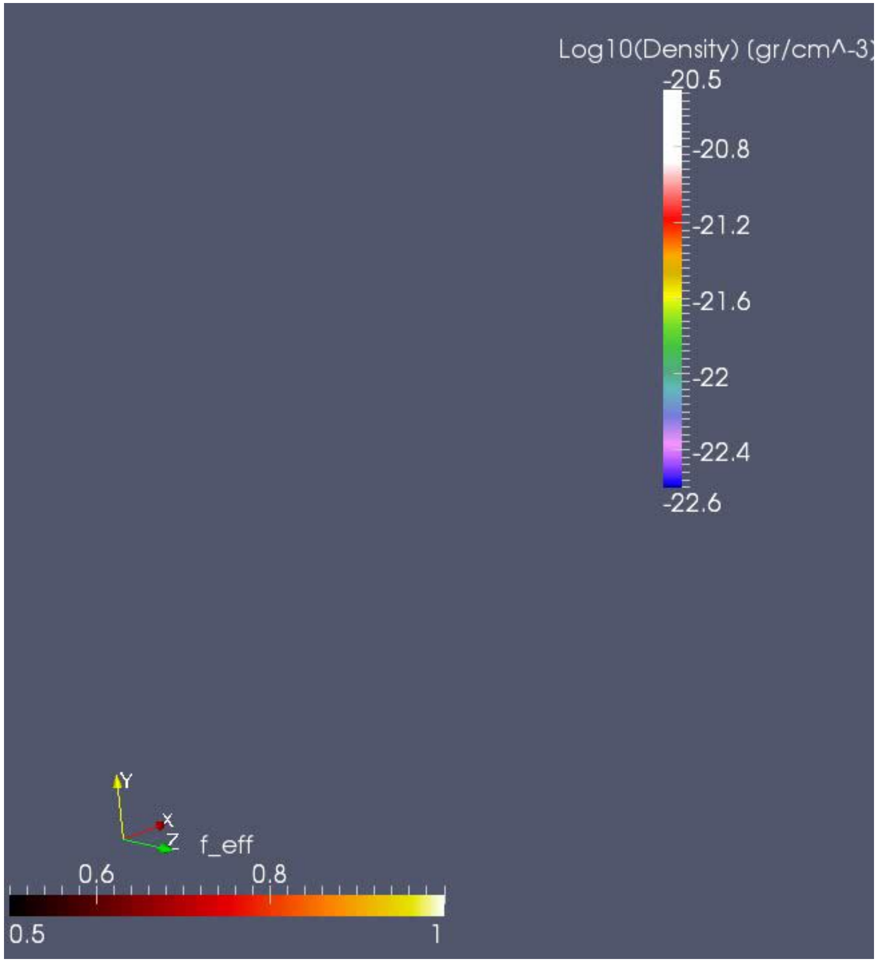
Data Interpretation is the scientist's *raison d'être*

How can the visualization scientist promote this?

- Learn the basics of the application field
- Confirm the analysis with quantitative data
- Minimize the risks of mis-interpretation (visual)

- Provide intuitive tools, designed especially for the data on hand

Confirm visual analysis with quantitative data



Supersonic turbulence in shock-bound slabs by Doris Folini and Rolf Walder, ENS-Lyon

Confirm visual analysis with quantitative data

“average upstream density is 14 particles per cm^3… average compression from "upstream" to "within the slab" is about 20… average particle density within the slab of 2x10^{14} = 280 particles per cm^3… One particle corresponds to about 1.6e^{-24} gram in our simulations… average density within the slab of 1.6e^{-24} x 280 = 4.48e^{-22} gram per cm^3

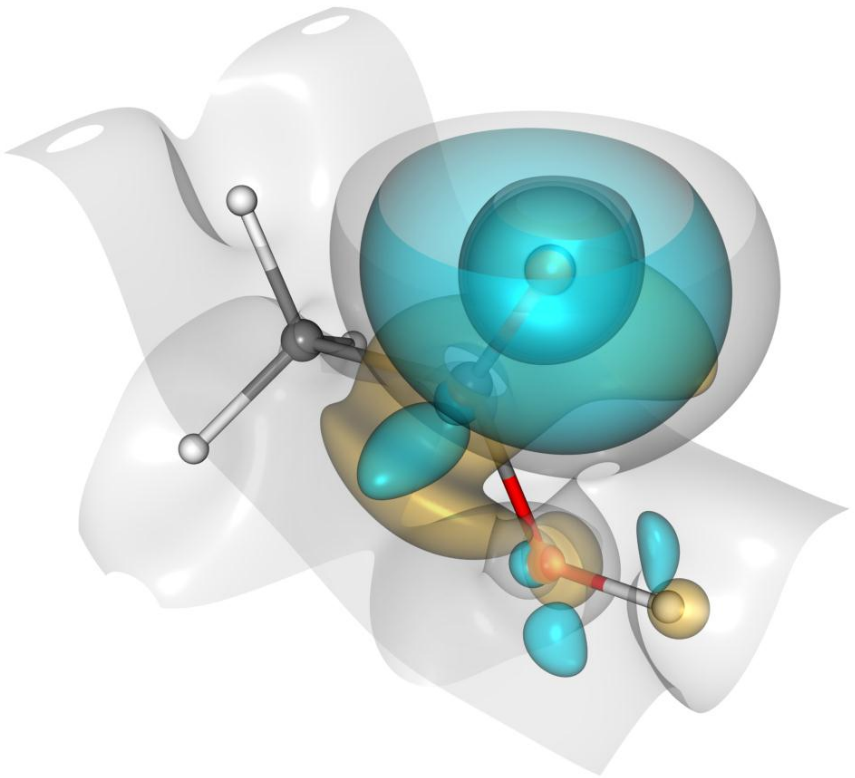
Multiplying this average density with the total volume of the slab at the last time (3.5e^{49} cm^3), I get 3.5e^{49} * 4.48e^{-22} = 1.57e^{28} grams

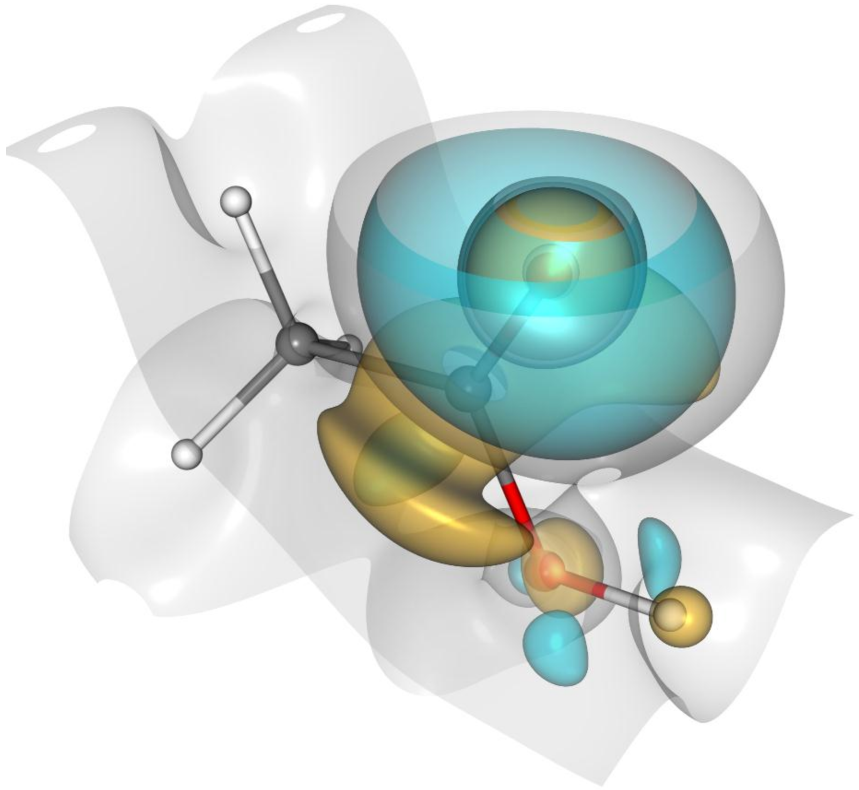
Minimize the risks of mis-interpretation

- In 15 years at CSCS, I have never put music on a scientific video
- No ColorfulFluidDynamics (with one exception)
- Provide spatial and temporal coherence
- Explain to the scientist what the visual representation “really” means.

- Examples:

Transparency is not a trivial to implement…



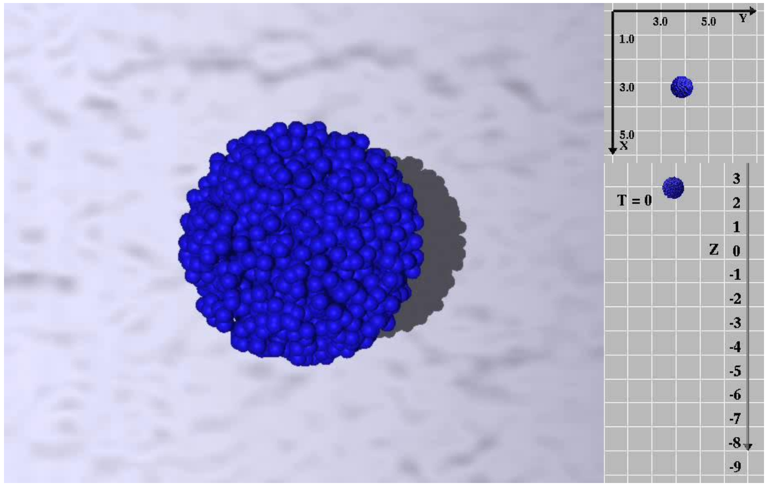


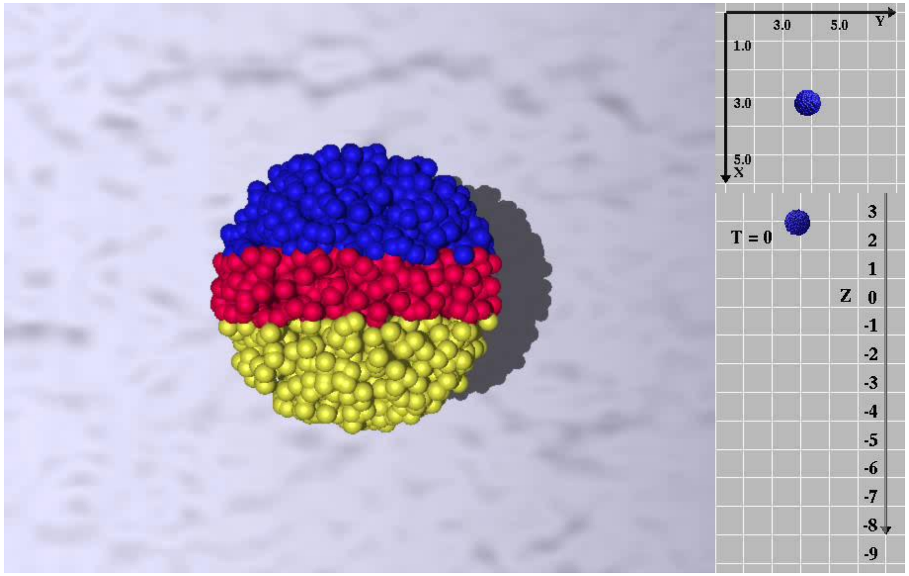
With Depth Peeling

Images by Ugo Varetto, CSCS

Issues with Camera Animation

- Difficult to follow an object at varying speed
- How much time has elapsed?
- How far have we travelled?
- Motion with respect to a fixed point of reference
- Motion of the particles with respect to each other



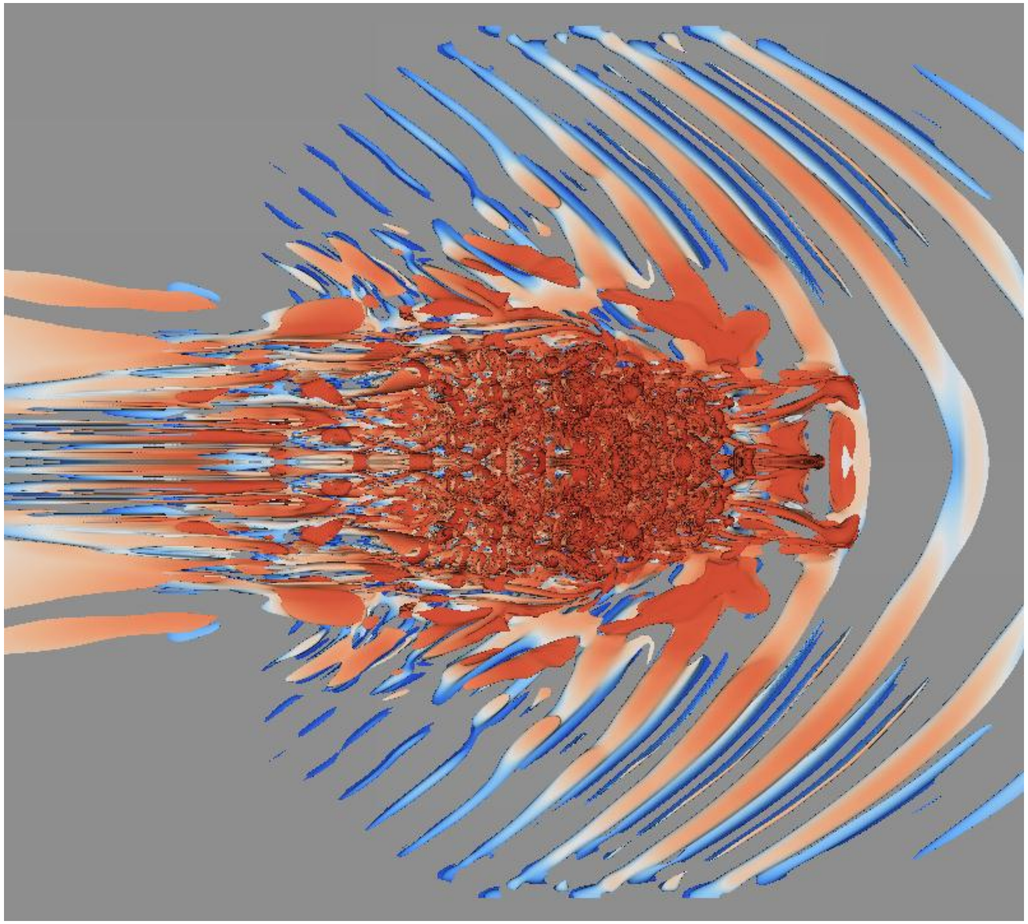


Provide intuitive tools, akin to the data

Molekel, a success story at CSCS

- Over 100K downloads in the past 4 years,
- Over 150 citations in Google Scholar in 2010

Issues with multi-billion cell DNS grids



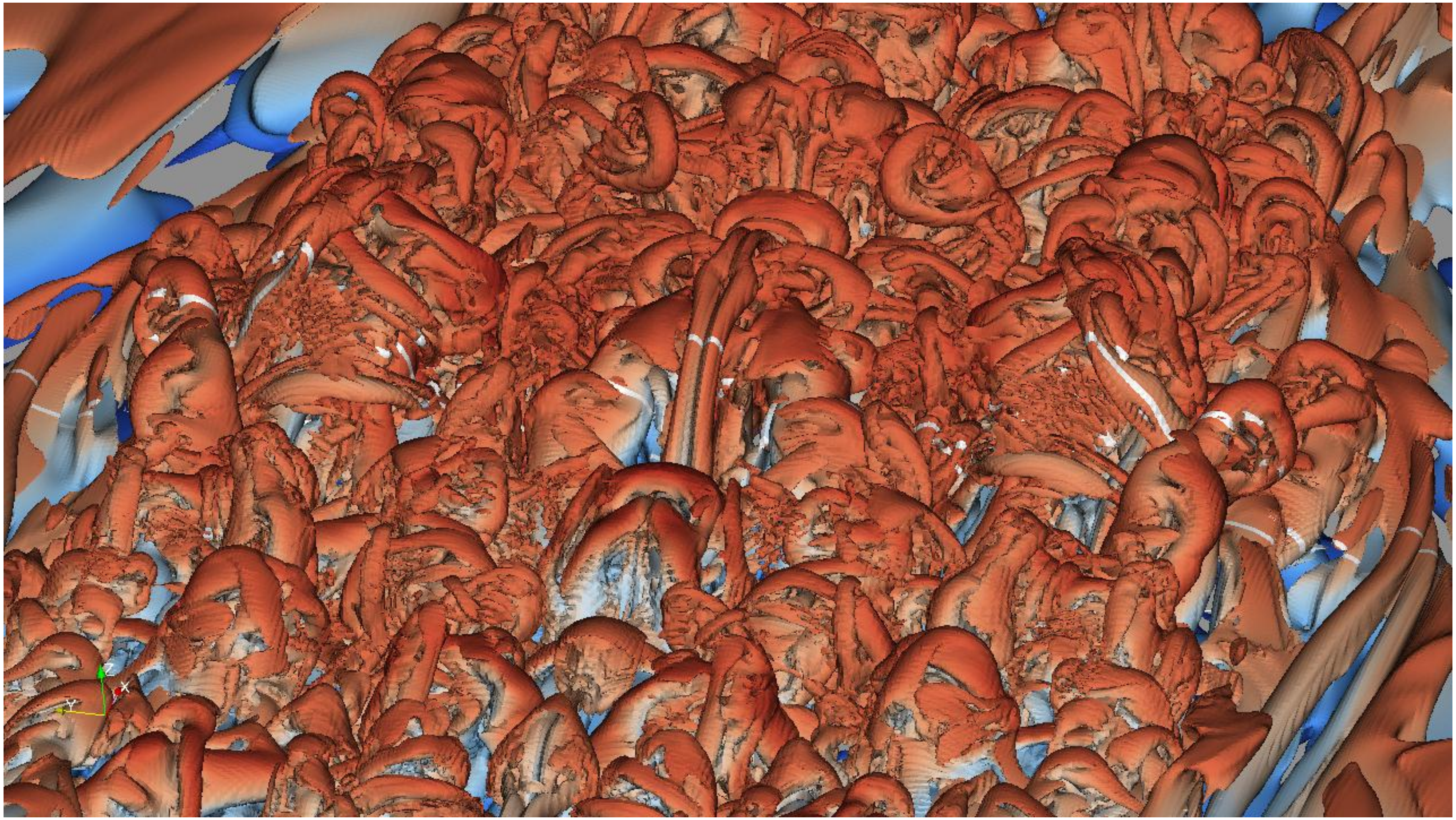
Users of DNS simulations want to see

“the data at full resolution”

Yes,

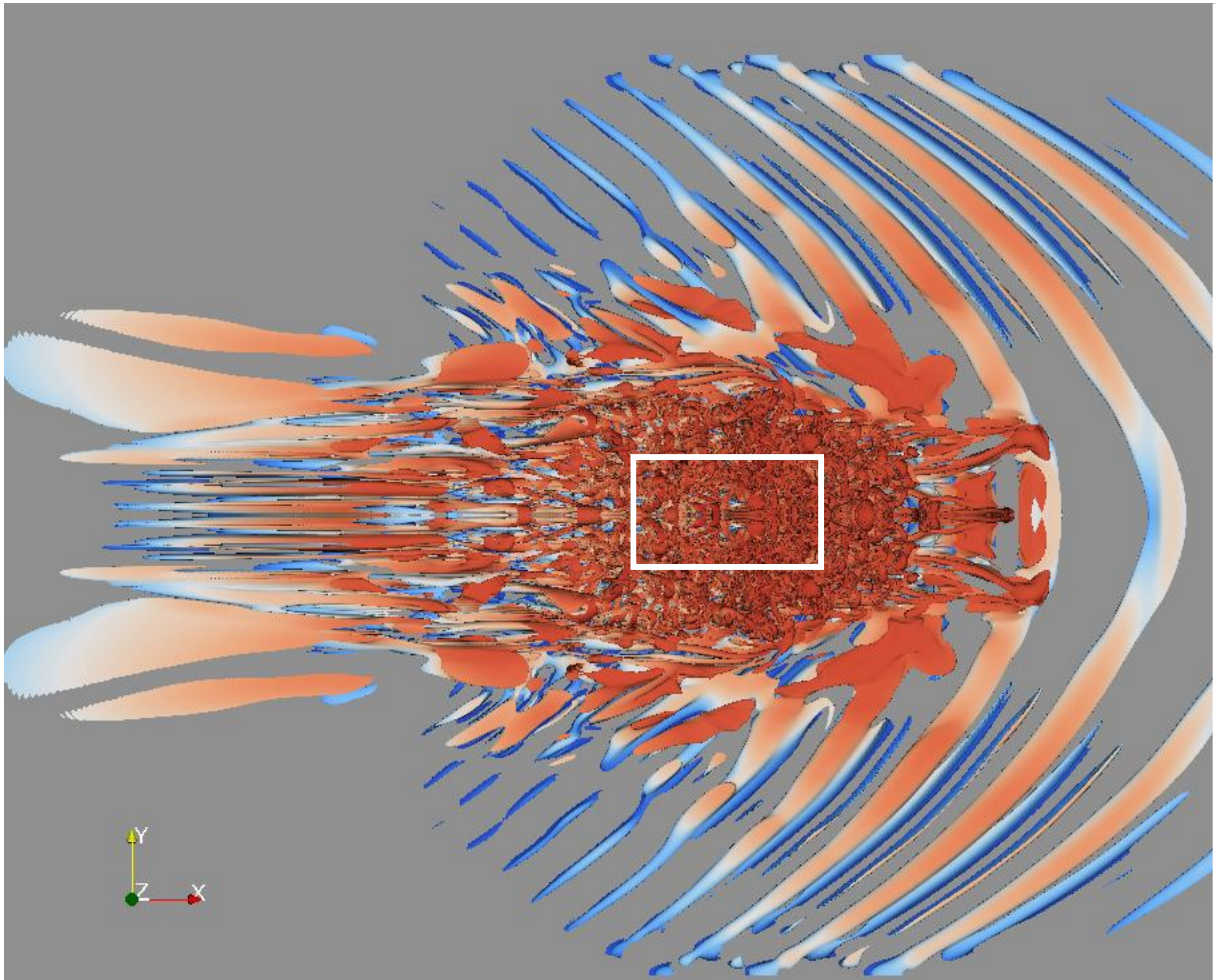
but implemented with care

Contribution of 1 cell in the X direction ?



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

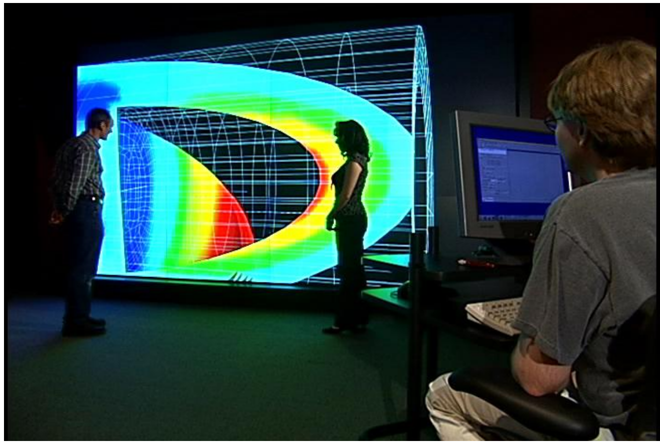
How can we see the details of the flow?



ftp://ftp.cscs.ch/out/jfavre/Kleiser/DNS.Zoom.v1.07.avi

Three options:

1. Image Wall ($$$$)



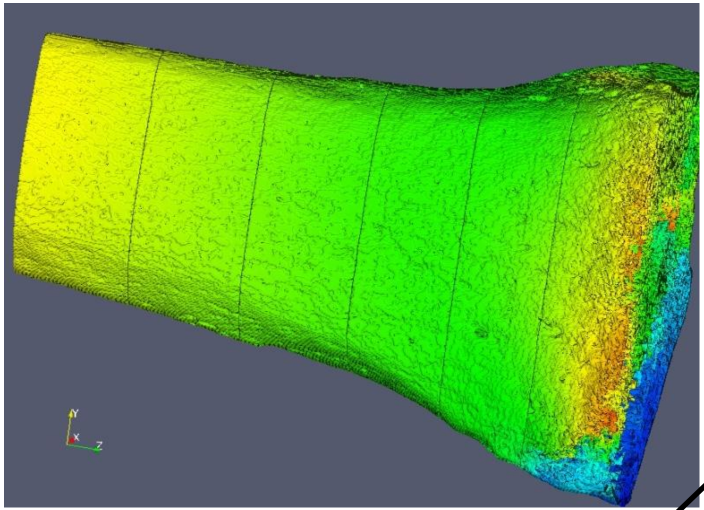
2. Zoom in (wasted I/O)

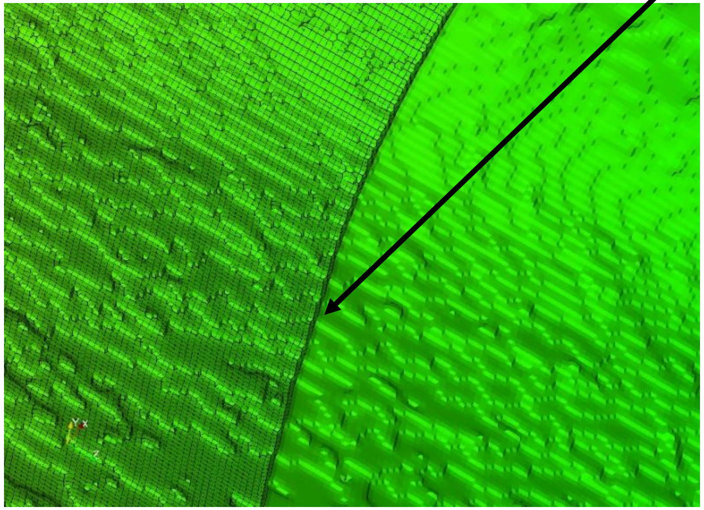
3. Read less

display only (1÷40) of the data volume

Implement multi-resolution or data streaming

Accept to trade-off speed for data quality

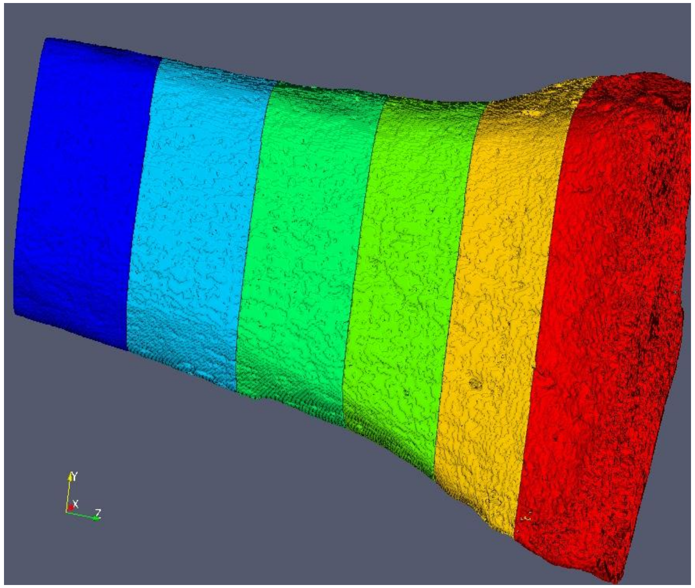


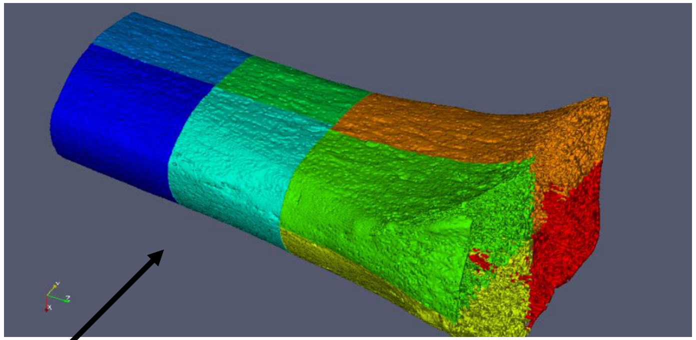


Implementation:
Very fast I/O reading

Symptoms:
Visual artifacts are caused by a non-physical boundary

Solution:
Slow I/O but partitions with reconstructed ghost-cells

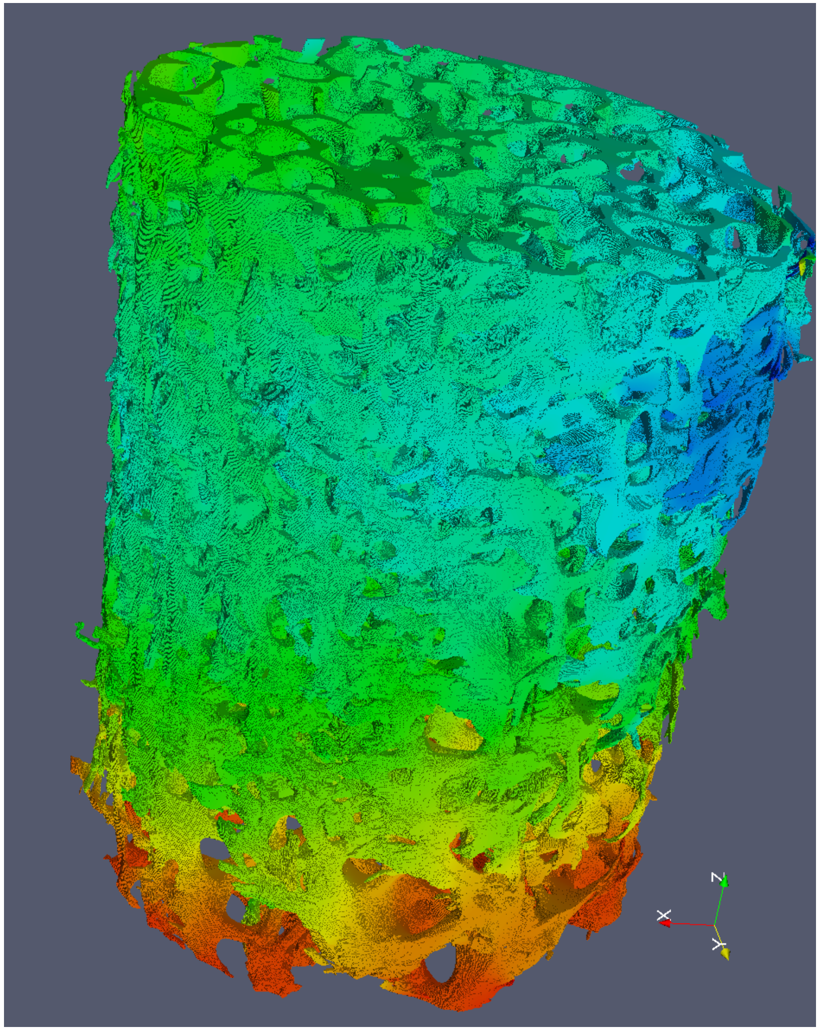




The biggest data we tried was still "interactive"

- 409,387,412 hexahedra and 447,294,209 nodes on 32 GPUs

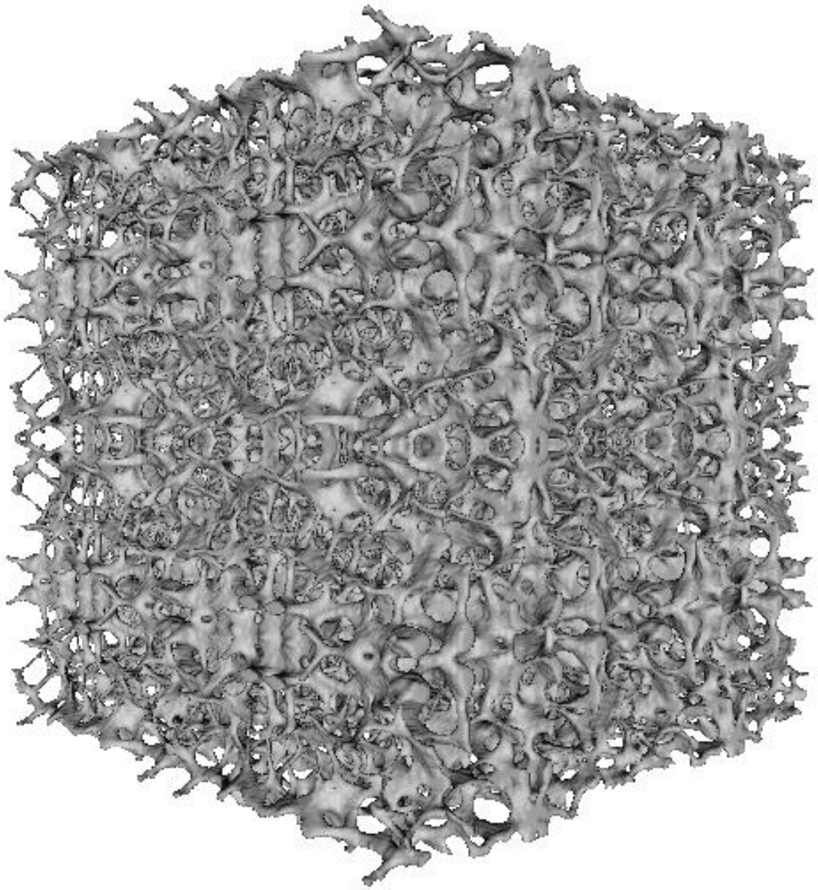
- but, we now have too much data on the screen and it is not very easy to see…

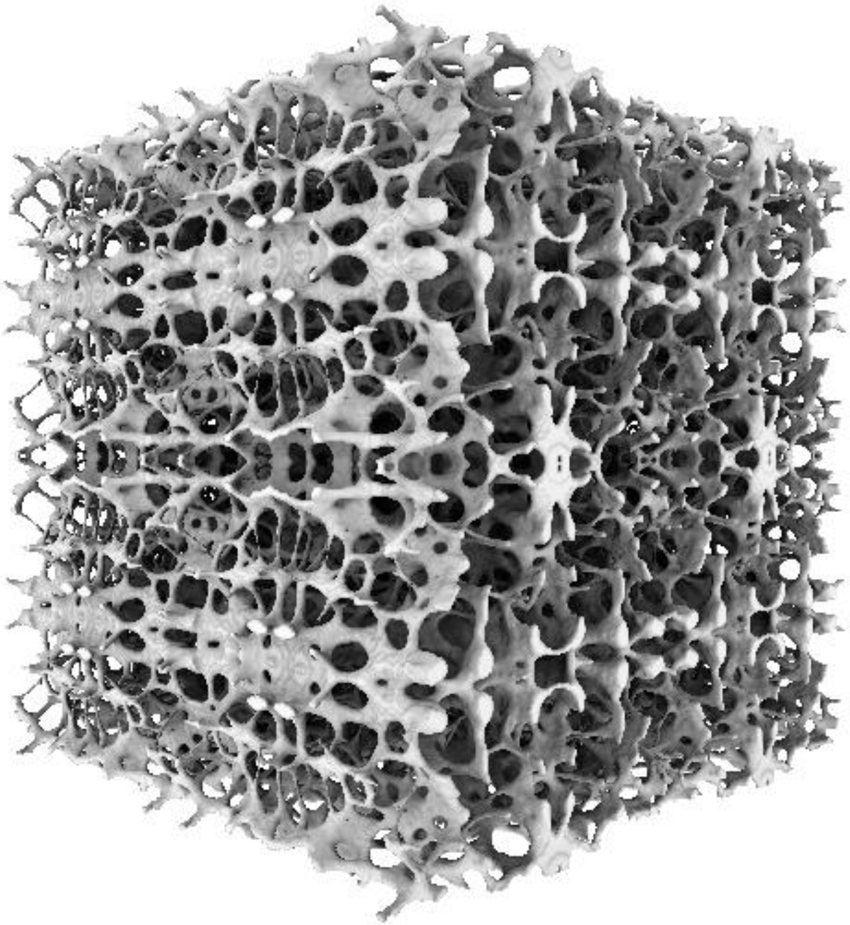


Challenge:

Can we increase realism?

Increase realism with Ambient Light Occlusion



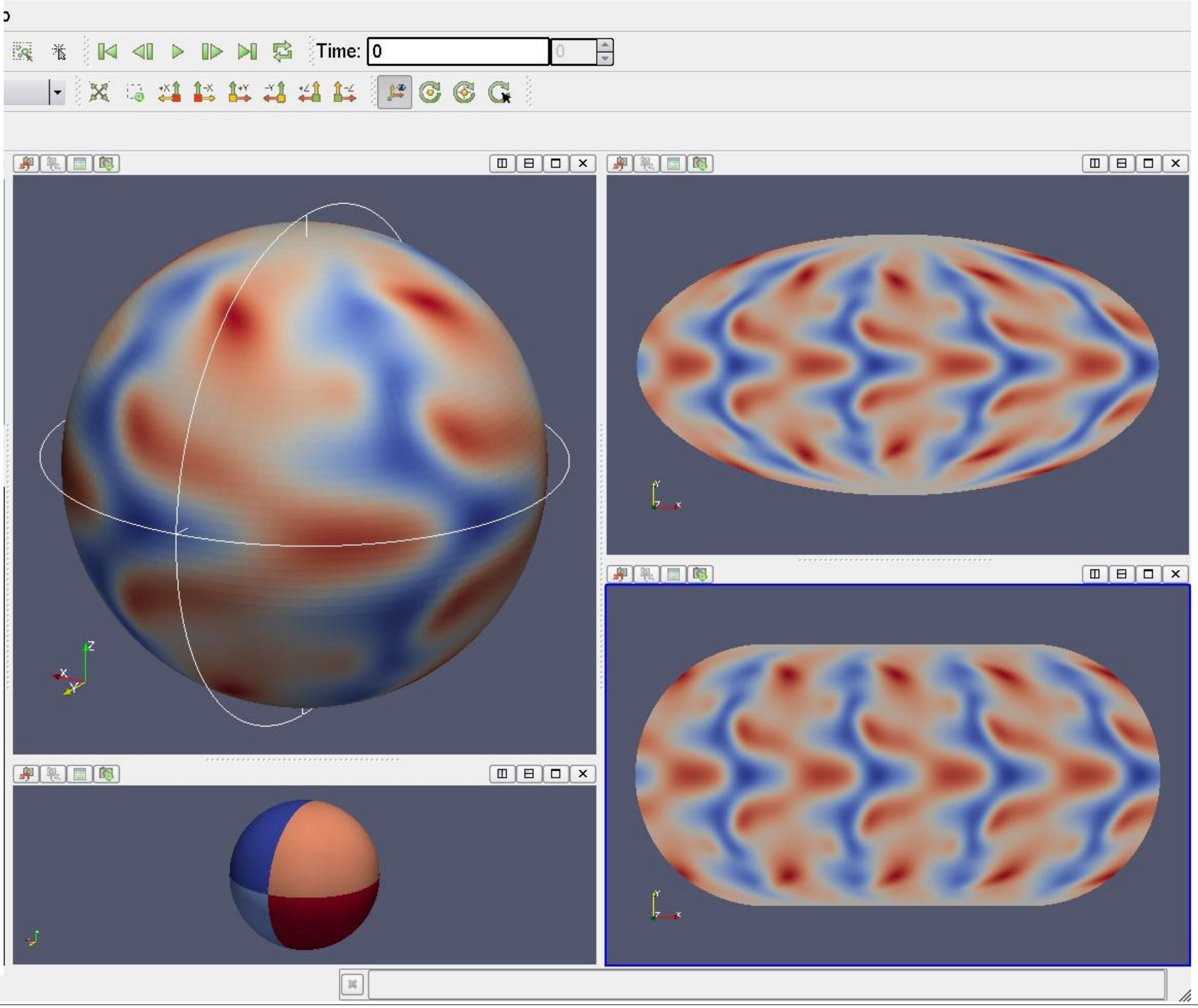


Images by Ugo Varetto, CSCS

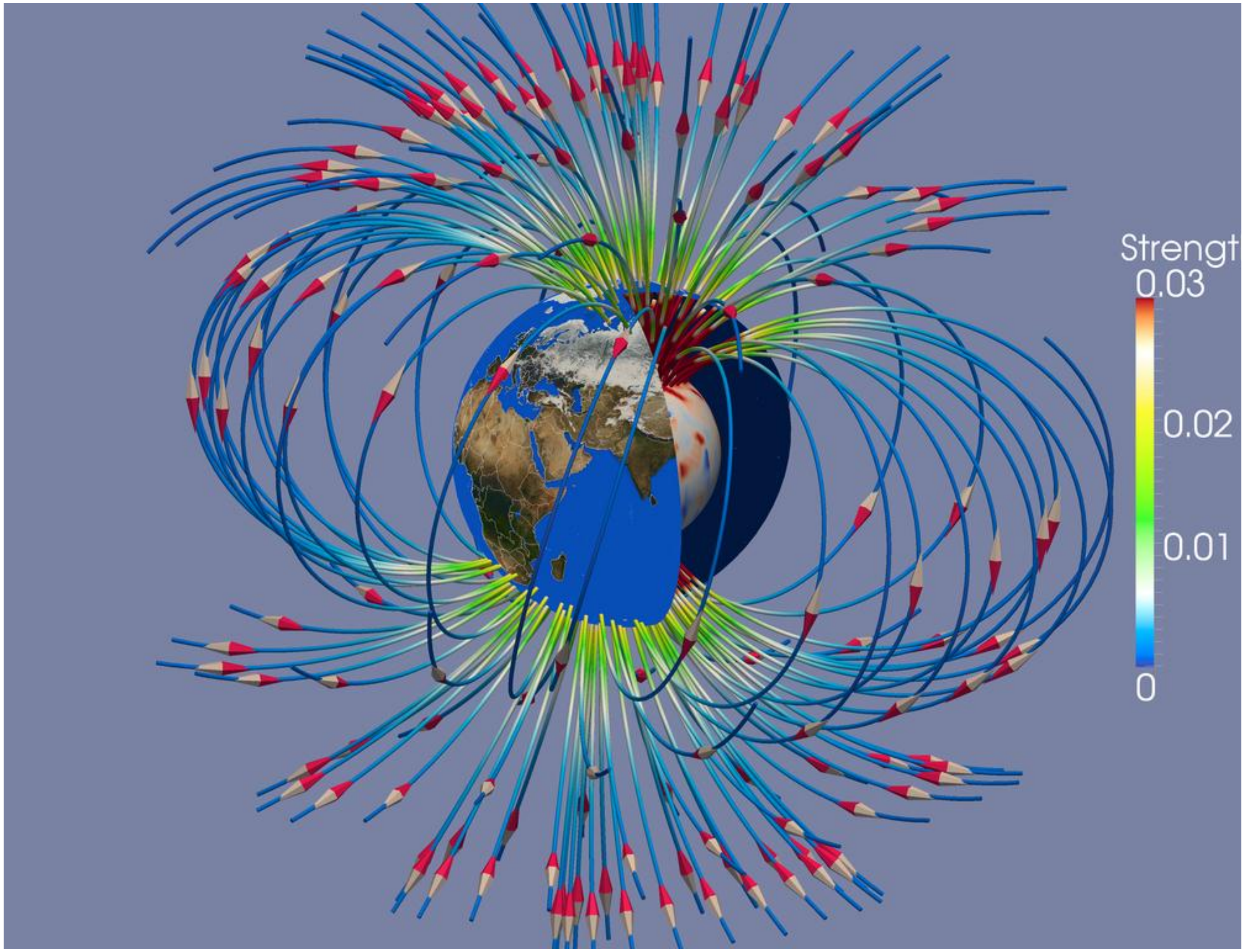
Parallel support for spherical grids in geo-physics

Implemented in ParaView a spherical to Cartesian mapping

and access to the PROJ.4 cartographic projections

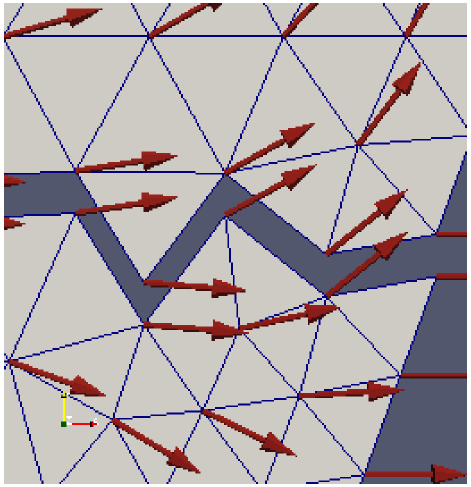


Parallel support for spherical grids in geo-physics



The best partitioning and I/O implementation should of course be scalable,…

Parallel support for spherical grids in geo-physics

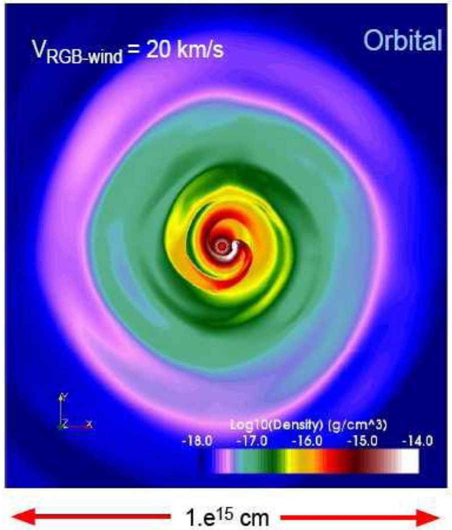


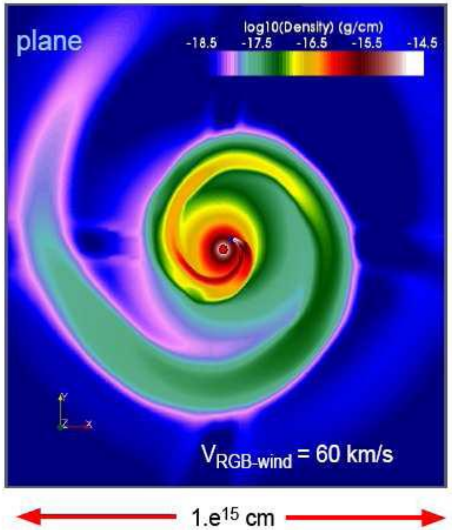
But must also,

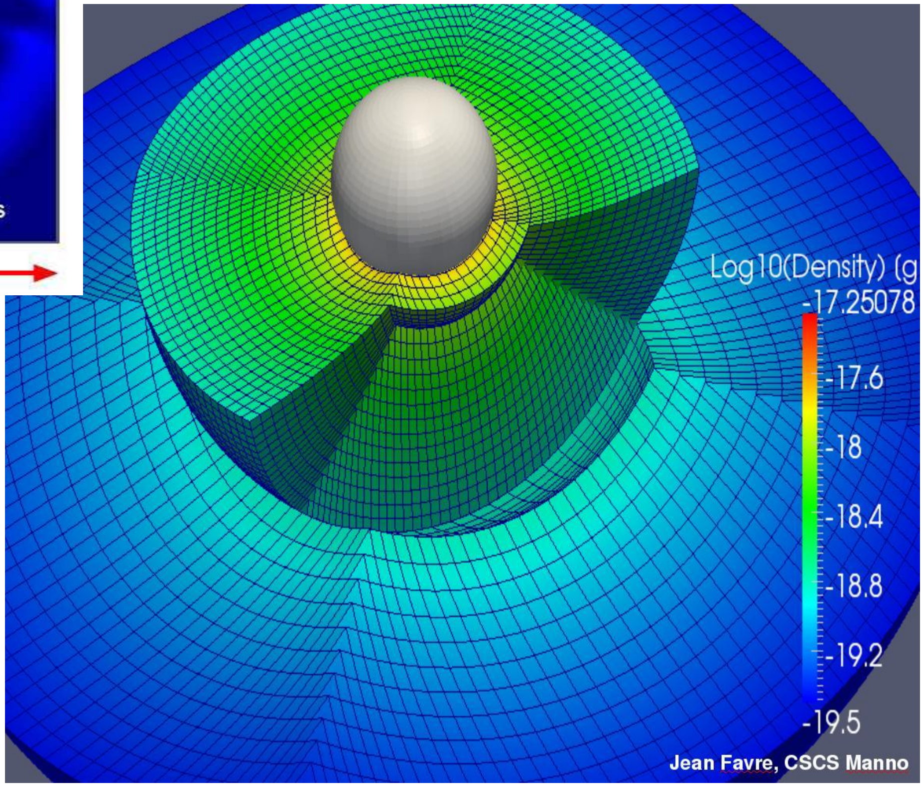
preserve data continuity,

and avoid redundancy.

Parallel support for astrophysics







Implementation of general mappings of grids to better support star geometries and the associated hyperbolic (and others) data spaces

Pause…

And a shift of paradigm

One action item in our HP2C efforts

The HP2C platform aims at developing applications to run at scale and make efficient use of the next generation of supercomputers. Presently this will be the generation of computing technologies available in 2013 timeframe.

Replace *post-processing* by *in-situ*

- Parallel simulations are now ubiquitous
- The mesh size and number of time steps are of unprecedented size
- The traditional ***post***-processing model "*compute-store-analyze*" does not scale because I/O to disks is the slowest component

Consequences:

- Datasets are often under-sampled on disks
- Many time steps are never archived
- It takes a supercomputer to re-load and visualize supercomputer data

Increase Data Locality!

- Many cores (22000 here)
- Fast I/O
- Software rendering
- Fast switch

But,

- Little memory

in-situ (parallel) visualization

Could I instrument parallel simulations to communicate to a subsidiary visualization application/driver?

- Eliminate I/O to and from disks
- Use all my grid data with ghost-cells
- Have access to all time steps, all variables
- Use my parallel compute nodes

- Don’t invest into building a GUI, or a new visualization package

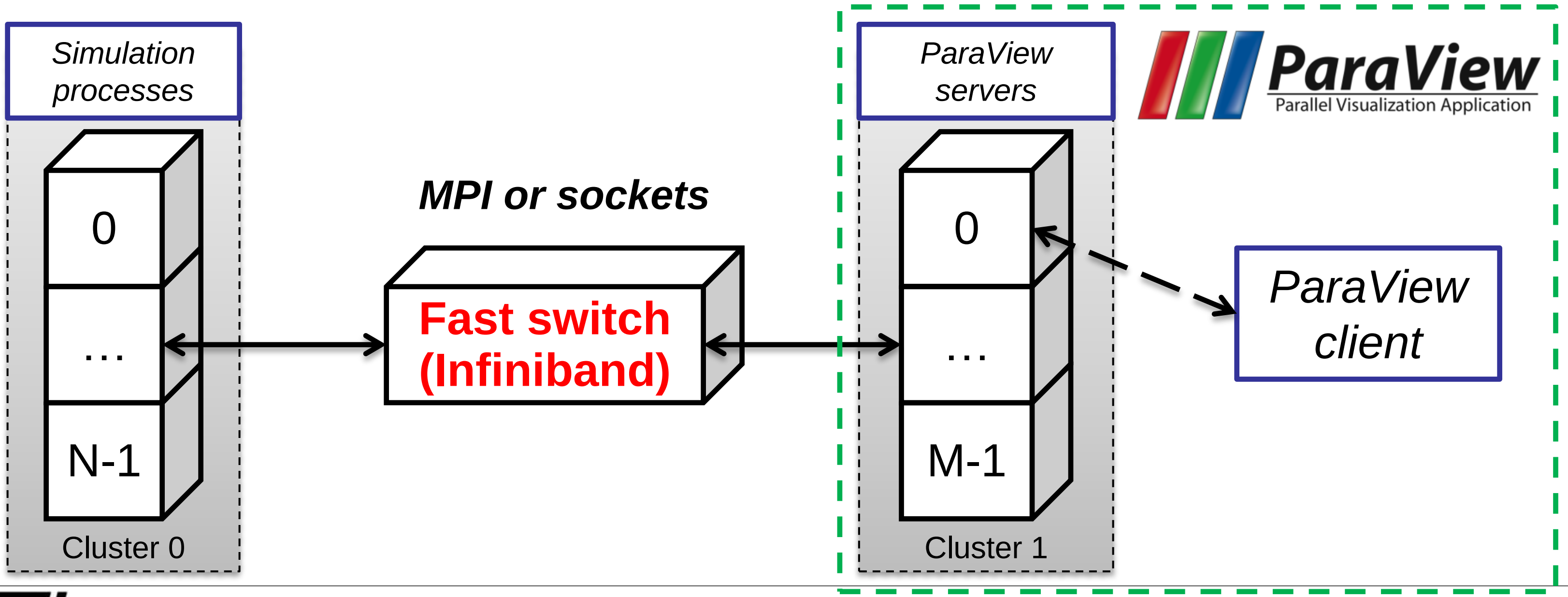
in-situ (parallel) visualization

We are currently prototyping two methods

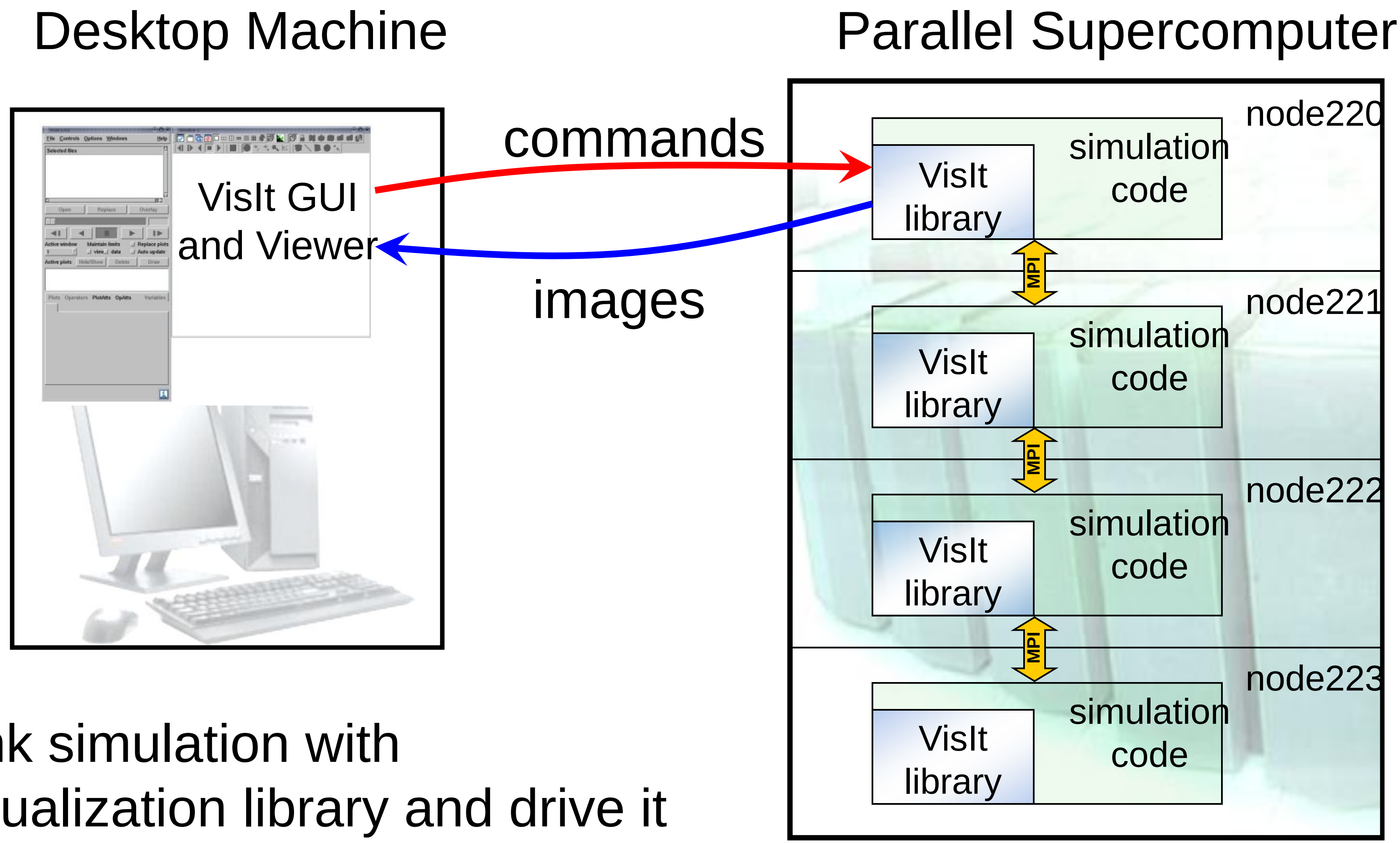
- **Parallel Data transfer to Distributed Shared Memory**
  - Computation and visualization physically separated
  - developed by JB/JS, publicly available on HPCforge
- **Co-processing**
  - Computation and visualization on the same nodes

First Method: ParaView

- Wire the supercomputer to the visualization cluster
- See some published work by my colleagues John Biddiscombe and Jerome Soumagne

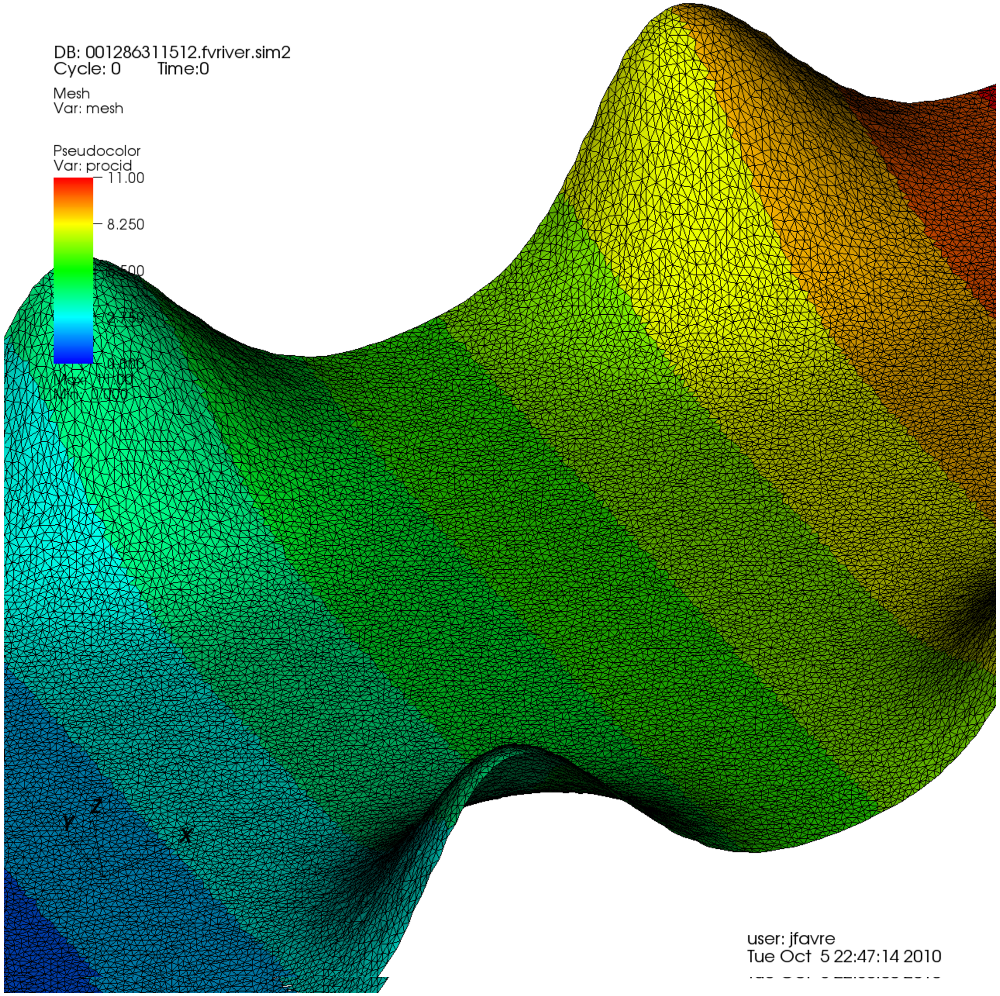


Second Method: VisIt



Link simulation with visualization library and drive it from a GUI

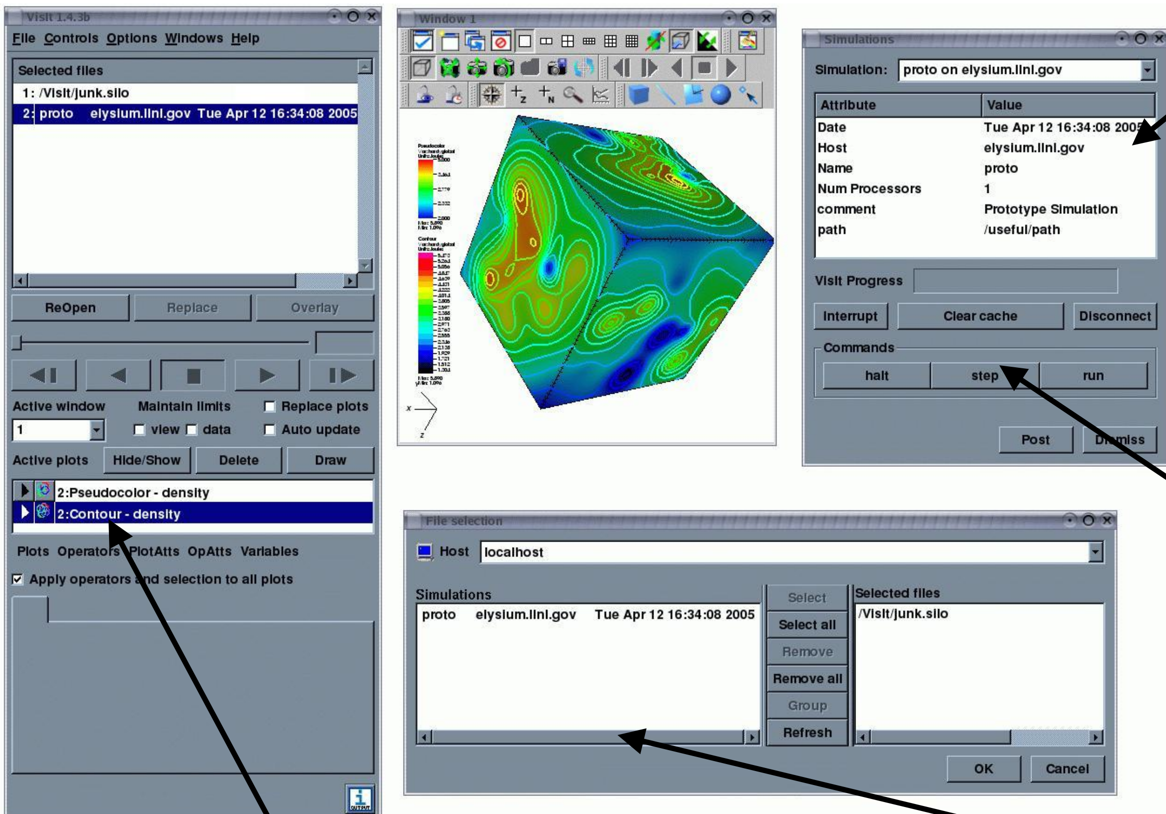
A short reminder about ghost-cells



Ghost- or halo-cells are usually not saved in solution files because the overhead can be quite high, and we need to be independent of the # of processors

When we couple simulation and visualization *in-situ*, the ghost cells are available, for free!

VisIt https://wci.llnl.gov/codes/visit



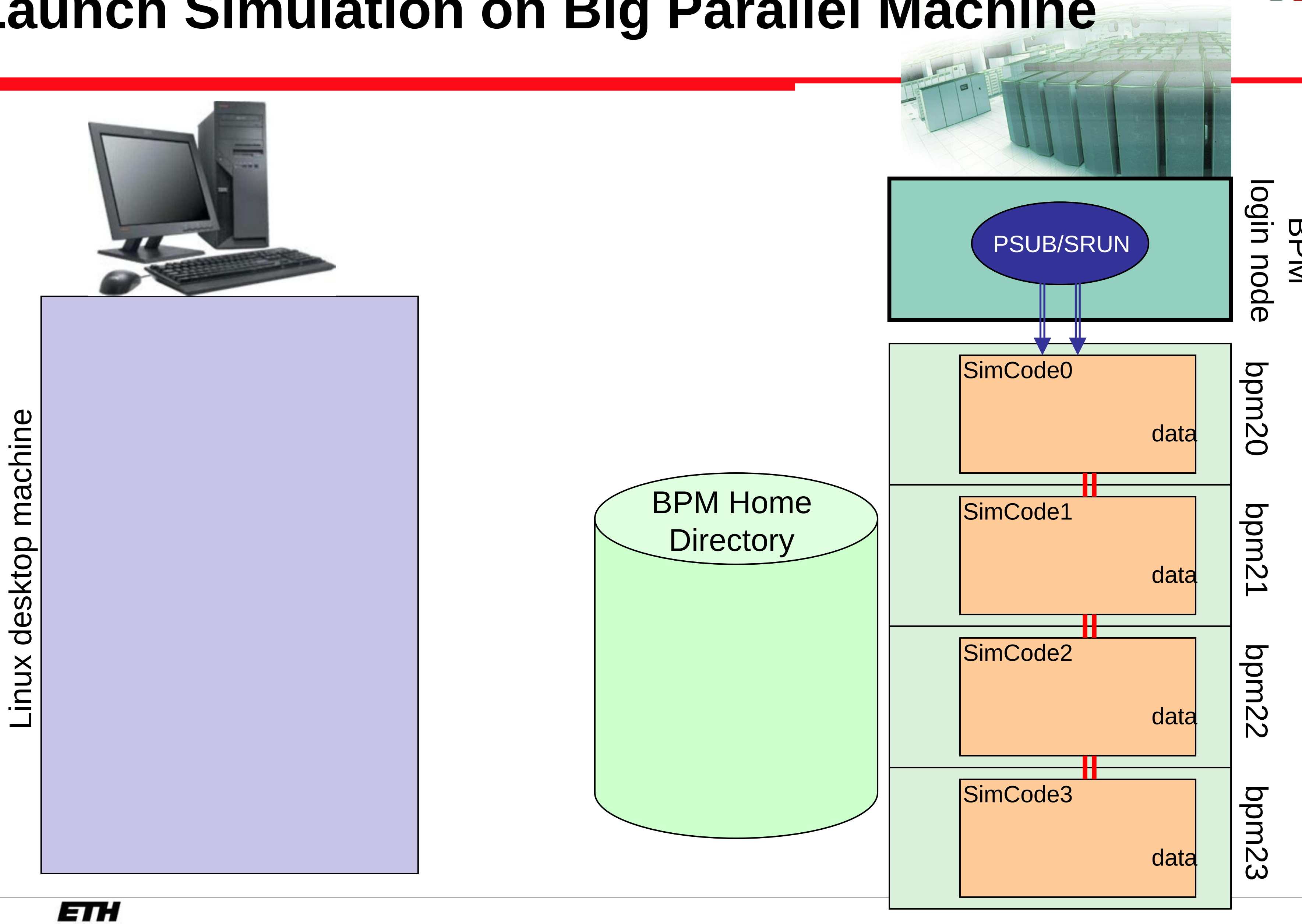
The Simulation's window shows meta-data about the running code

Control commands exposed by the code are available here

Users select simulations to open as if they were files

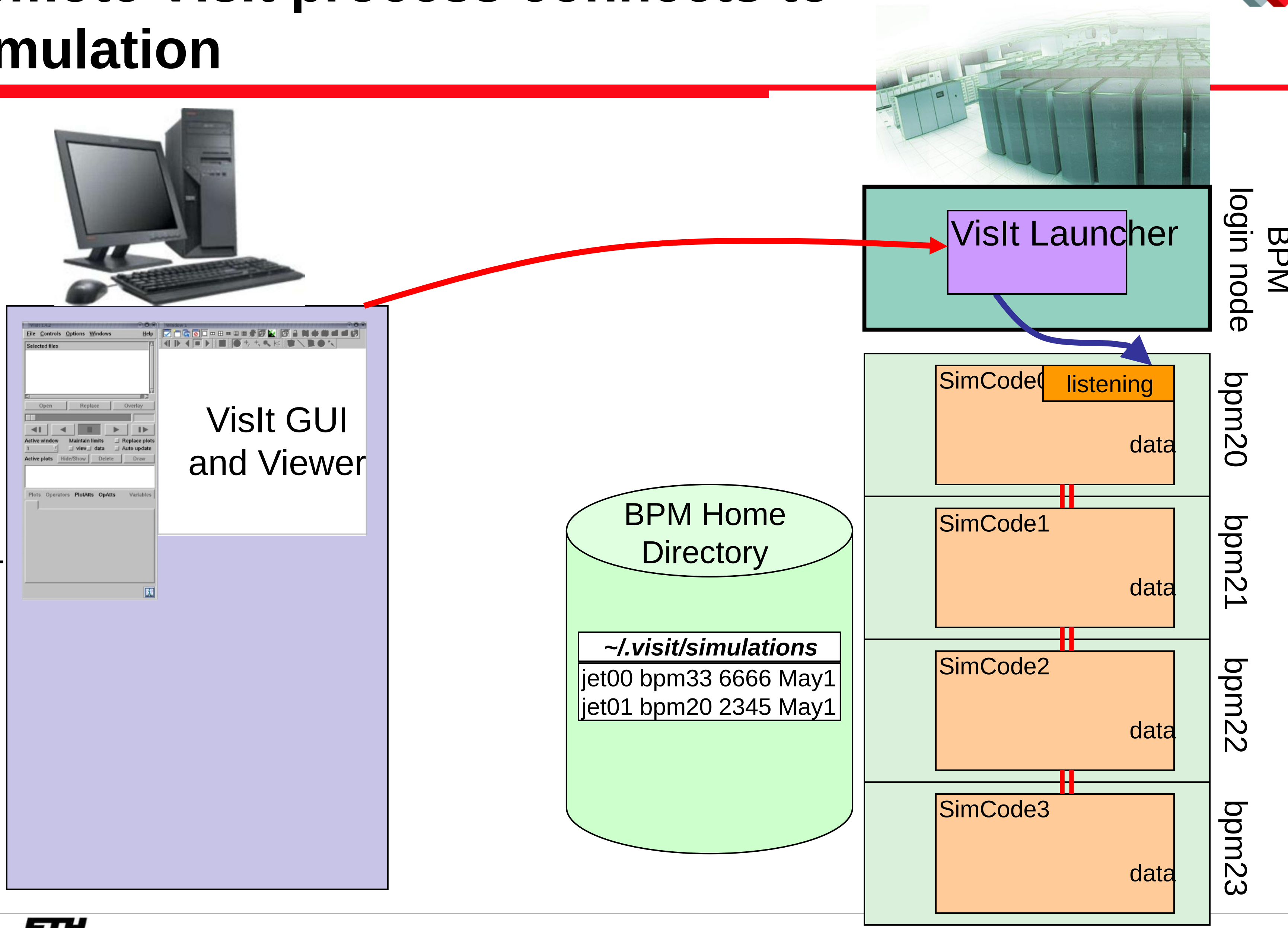
All of VisIt's existing functionality is accessible

Launch Simulation on Big Parallel Machine

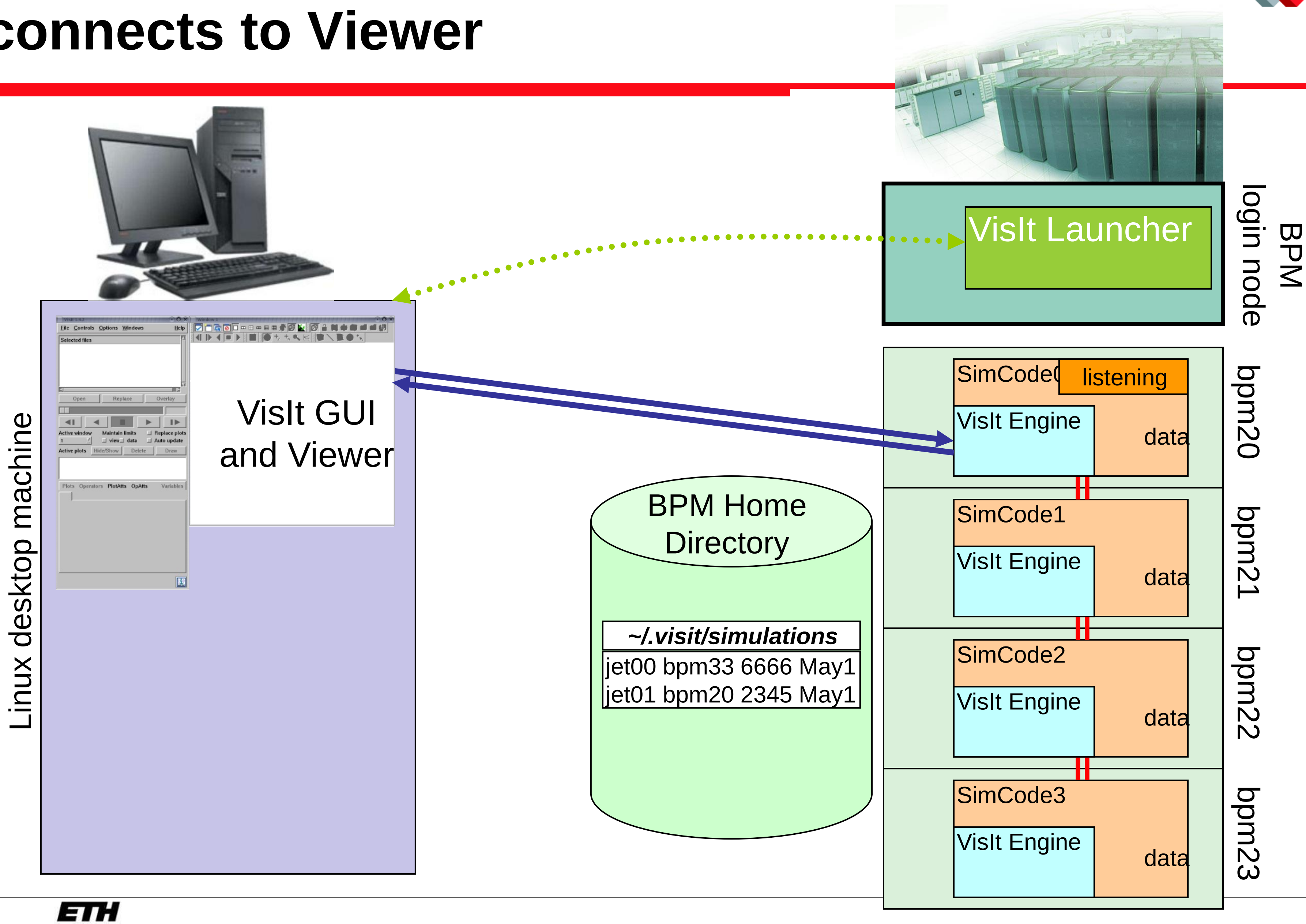


Remote VisIt process connects to Simulation

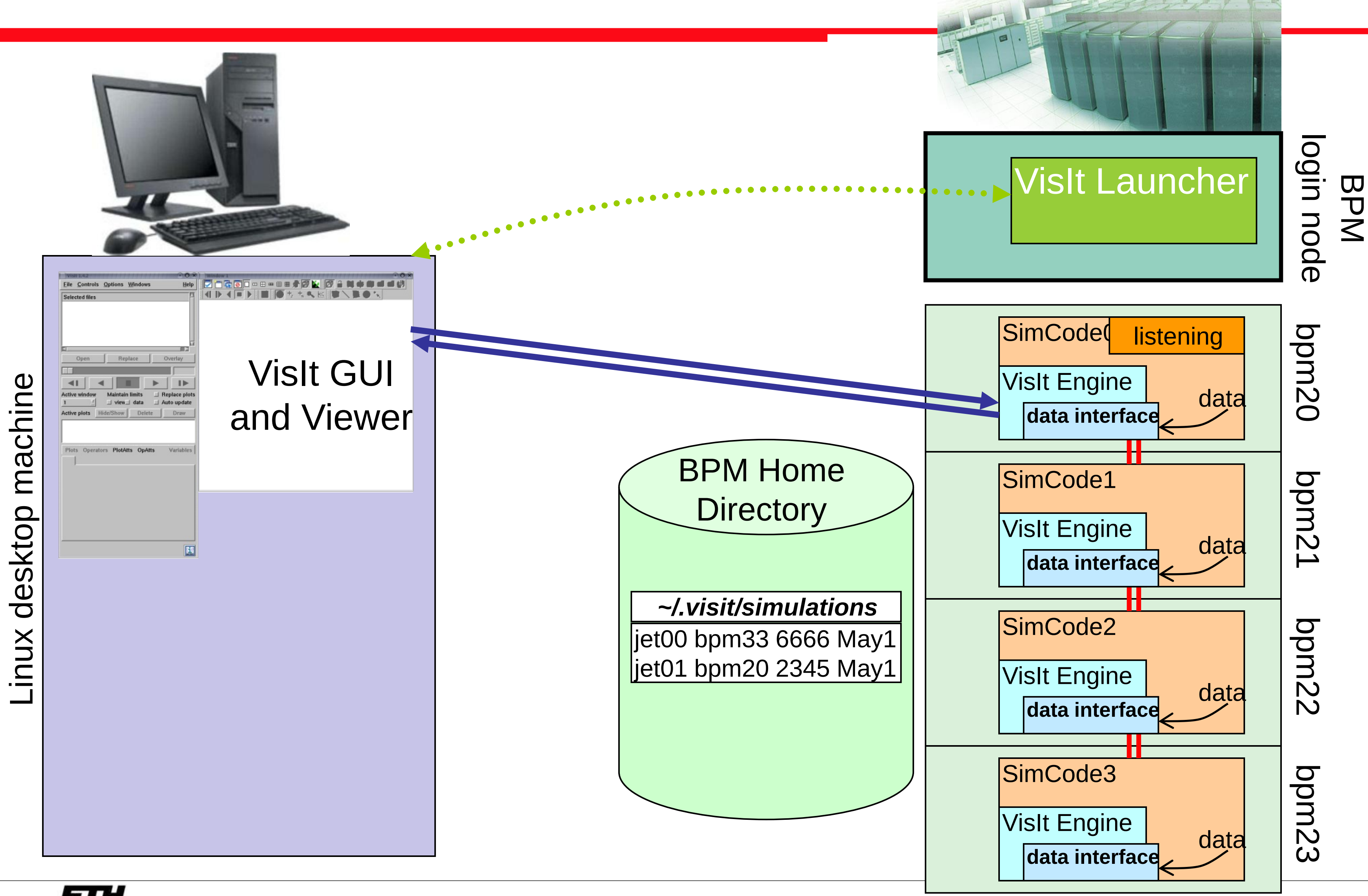
CSCS
Swiss National Supercomputing Centre



Simulation *becomes* engine, connects to Viewer



VisIt requests pull Data from Simulation



Some details on the API

- The C and Fortran interfaces for using SimV2 are identical, apart from calling different function names

- The VisIt Simulation API has just a few functions
  - set up the environment
  - open a socket and start listening
  - process a VisIt command
  - set the control callback routines
- The VisIt Data API has just a few callbacks
  - GetMetaData()
  - GetMesh()
  - GetScalar(), etc

Callbacks are added to advertize the data

visitcommandcallback()

visitgetmetadata ()

- Mesh name
- Mesh type
- Topological and spatial dimensions
- Units, labels
- Variable names and location (cell-based, node-based)
- Variable size (scalar, vector, tensor)
- Commands which will be understood (next(), halt(), run(), …)

visitsimgetmesh()

How much impact in the source code?

The **best suited** simulations are those allocating large (contiguous) memory arrays to store mesh connectivity, and variables

Memory pointers are used, and the simulation (or the visualization) can be assigned the responsibility to de-allocate the memory when done.

How much impact in the source code?

The **least suited** are those emphasizing the Object Oriented principles to a maximum

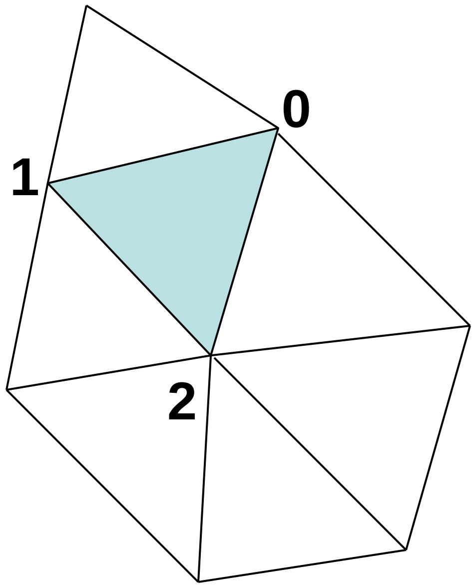
Data points spread across objects require a new memory allocation to gather the data before passing it to the Vis Engine

:

```
TYPE Element
        REAL(r8) :: x(3)
        REAL(r8) :: y(3)

        REAL(r8) :: h
        REAL(r8) :: u
        REAL(r8) :: zb(3)

END TYPE Element
```



The VisIt *in-situ* library provides many features

- Access to scalar, vector, tensor arrays, and label
- CSG meshes
- AMR meshes
- Polyhedra
- Material species
- Ability to save images directly from the simulation
- Interleaved XY, XYZ coordinate arrays

- See a short introductory article to be published next week at EPF-L

Advantages compared to saving files

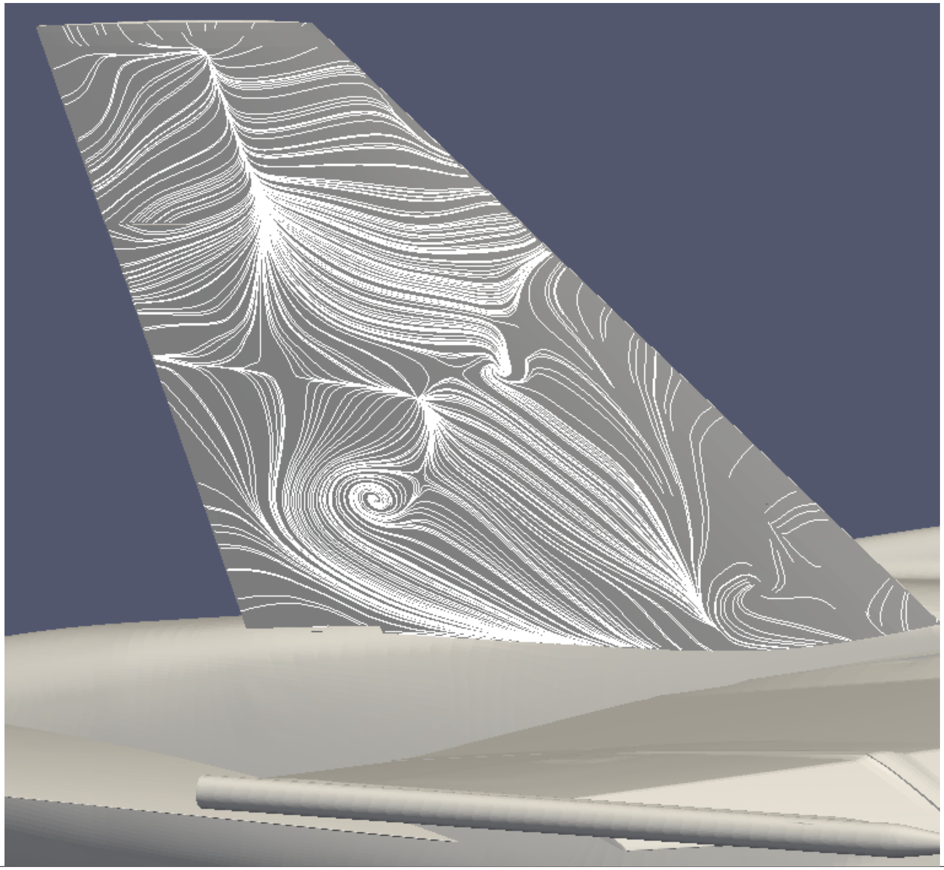
- The greatest bottleneck (disk I/O) is eliminated
- Not restricted by limitations of <u>any file format</u>
- No need to reconstruct ghost-cells from archived data
- All time steps are potentially accessible
- All problem variables can be visualized
- Internal data arrays can be exposed or used
- Parallel compute nodes are already allocated

- The simulation can watch for a particular event and trigger the update of the VisIt plots

Summary

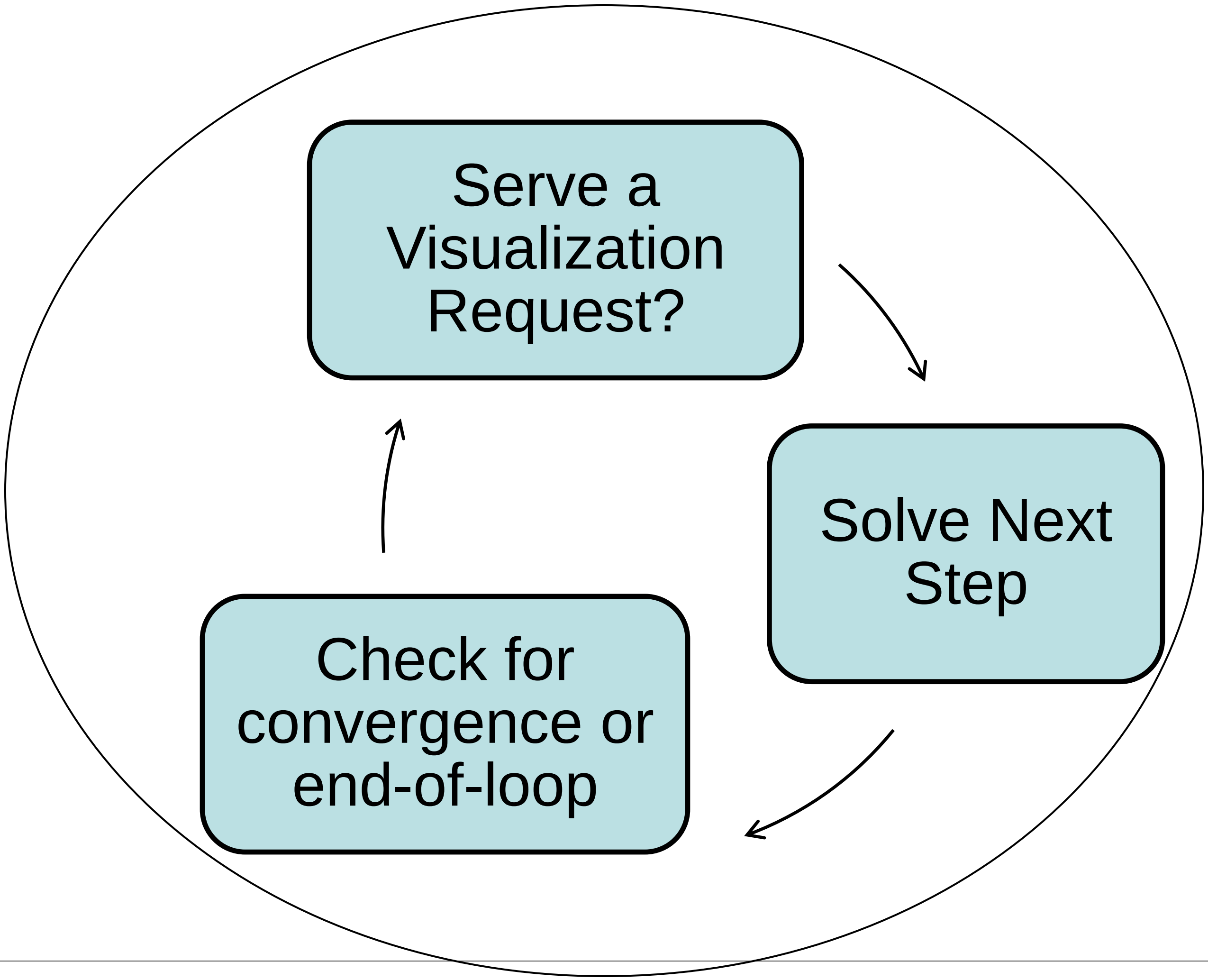
In the past, we focused on raw data => images

<DataArray type="UInt8" Name="types" format="appended" RangeMin="" RangeMax="" offset="5948" />
</Cells>
</Piece>
</UnstructuredGrid>
<AppendedData encoding="base64">
_AQAAPAAAAFwAAAA==eJwVzzEoRHE
Ax/H/YDAYbjAYDDcYDIYbDAbluQwGww
0Gg+EGg8Fwg8FgeEm6JF2SLkkvSZekS
9J1SS9Jl6RL0iXpjUaj0Uf9PvOvbwgfxRA+
+SIjngohREZMQkpGmA4hR54CESXKVIip
UqNOQoMmLVI6dOmR8c0PvwT/ffQzQI5
BhhgmzwijjFFgnAkmiZhhljlKzLPAImWWW
GaFCqussf7fzgabbFFlmx12qbHHPgfUOe
SIYxJOOOWMBudccEmTK665oUWbW+5
IueeBRzo88cwLXV55451e8Q8G5lcqAQA
AAACAAABABQAAgQIAAA

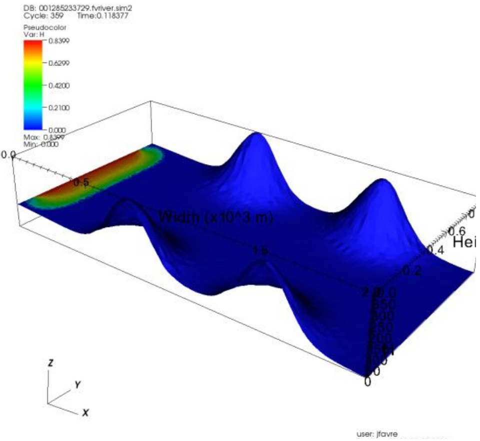


We are now adding a new interaction paradigm

mega-,
giga-,
peta-,
exa-scale
simulations
can now be
coupled with
visualization



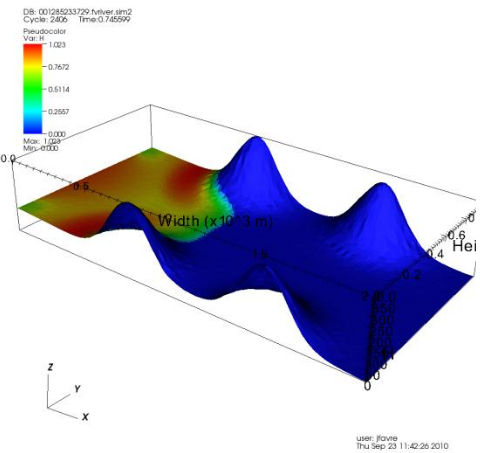
now focus on source code => live images

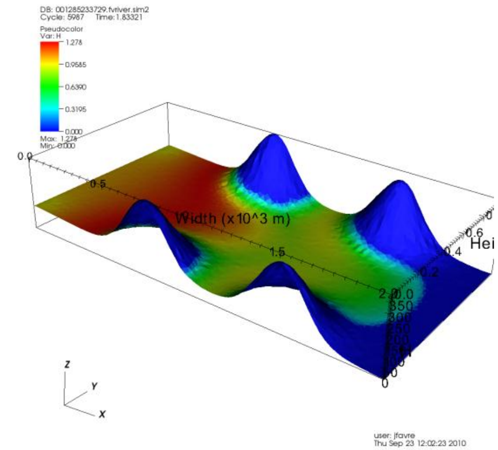


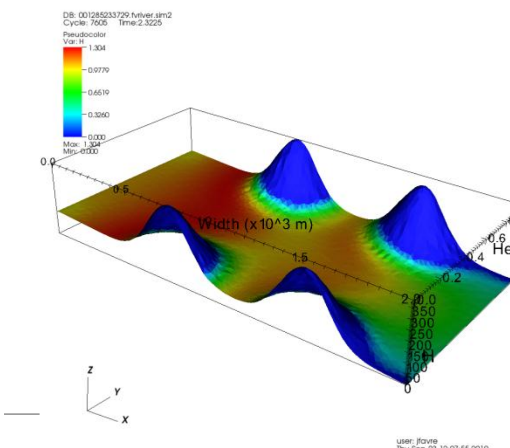

```
REAL, DIMENSION(:), ALLOCATABLE :: cx
ALLOCATE( cx(numNodes) , stat=ierr)

DO iElem = 1, numElems+numHalos
        DO i = 1, 3
          cx(ElementList(iElem)%lclNodeIDs(i))
= ElementList(iElem)%x(i)
        END DO
END DO

err = visitvardatasetf(x,
VISIT_OWNER_COPY, 1, numNodes, cx)
```







Conclusion

We have seen a few examples of carefully crafted visualizations

Parallel visualization is mature, but is **very** limited by I/O

In-situ visualization is an attractive strategy to mitigate this problem, but will require an even stronger collaboration between the application scientists and the visualization scientist