



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



# Scientific Data Visualization with VisIt

ANF Visualisation et données

Jean M. Favre, CSCS

30 novembre, 2016

# Outline

- Premiere partie (9h30-10h45)
  - Introduction to Visit 9h30-9h45
  - Basics [Tutorial](#)
  - Python scripting [Tutorial](#) 9h45-10h00
  - Data Analysis (Queries and Expressions) [Tutorial](#) 10h00-10h45
- Deuxieme partie 11h-12h30
  - Blood Flow Aneurysm [Tutorial](#) 11h-11h45
  - Client-Server and parallel execution 11h45-12h00
  - The *In situ* Terminology 12h-12h30
- Apres-midi (14h-15h30)
  - couplage aux calculs scientifiques (libsim)
  - Code instrumentation
  - Connecting to a live simulation
  - Batch-mode simulation+visualization

# Plan

- Use parts of the wiki tutorials which are on-line and up-to-date on [visitusers.org](http://visitusers.org)
- Do one long exercise/tutorial
- Run in-situ examples

# The very short summary and objectives of the full day (1)

## Visualize data post-mortem, interactively

- Load data from disk
- AddPlot("Pseudocolor", "temperature")
- AddOperator("Isosurface")
- DrawPlots()
- SaveWindow()

## Visualize data post-mortem, batch mode

- Repeat the same operations with multiple datasets, different compute resources.

# The very short summary and objectives of the full day (2)

## Visualize data in-situ, interactively

- Connect to a live simulation
- AddPlot("Pseudocolor", "temperature")
- AddOperator("Isosurface")
- DrawPlots()
- SaveWindow()

## Visualize data in-situ, batch mode

- Run the simulation and execute visualization calls, export data products contemporaneously.
- VisItAddPlot("Pseudocolor", "temperature")
- VisItAddOperator("Isosurface")
- VisItDrawPlots()
- VisItSaveWindow()

# Tutorial preparation

## Select and download a VisIt release binary for your platform

<https://wci.llnl.gov/simulation/computer-codes/visit/executables>

## Extract or Install the release

Linux/Unix:

untar the tarball

Mac

Open the DMG file and copy the VisIt app bundle to your desktop

Windows:

Run the VisIt installer program and follow the prompts.

## Run VisIt to test your installation

Linux/Unix

In your shell, run

>path/to/visit/bin/visit

Mac:

Double click the VisIt app bundle

Or, in your shell, run

>path/to/VisIt.app/Contents/Resources/bin/visit

Windows:

Launch VisIt from the start menu (or equivalent)

## Download the tutorial sample data:

[http://visitusers.org/index.php?title=Tutorial\\_Data](http://visitusers.org/index.php?title=Tutorial_Data)

## Untar or unzip the visit\_tutorial\_data file

```
if /usr/bin/test -z "$userenv_visit_212"; then
```

```
if /usr/bin/test -z "$PATH"
```

```
then
```

```
PATH=/local/apps/VisIt/2.12/current/linux-x86_64/bin:/local/apps/VisIt/2.12/bin; export PATH
```

```
else
```

```
PATH=/local/apps/VisIt/2.12/current/linux-x86_64/bin:/local/apps/VisIt/2.12/bin:$PATH; export PATH
```

```
fi
```

```
if /usr/bin/test -z "$LD_LIBRARY_PATH"
```

```
then
```

```
LD_LIBRARY_PATH=/usr/lib/nvidia-367:/local/apps/VisIt/2.12/current/linux-x86_64/lib; export LD_LIBRARY_PATH
```

```
else
```

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/local/apps/VisIt/2.12/current/linux-x86_64/lib; export LD_LIBRARY_PATH
```

```
fi
```

```
export PYTHONPATH=/local/apps/VisIt/2.12/current/linux-x86_64/lib:  
/local/apps/VisIt/2.12/current/linux-x86_64/lib/site-packages
```

```
userenv_visit_212=y; export userenv_visit_212
```

```
export VISITHOME=/local/apps/VisIt/2.12
```

```
export VISITARCHHOME=/local/apps/VisIt/2.12/current/linux-x86_64
```

```
fi
```

# Get copies of python simulation examples

---

<http://visit.ilight.com/svn/visit/trunk/src/tools/DataManualExamples/Simulations/>

[batch.py](#)

[updateplots.py](#)

# ipython and VisIt

---

```
export PYTHONPATH=/local/apps/VisIt/2.12/current/linux-x86_64/lib/site-packages
```

```
ipython
```

```
from visit import *
```

```
Launch()
```

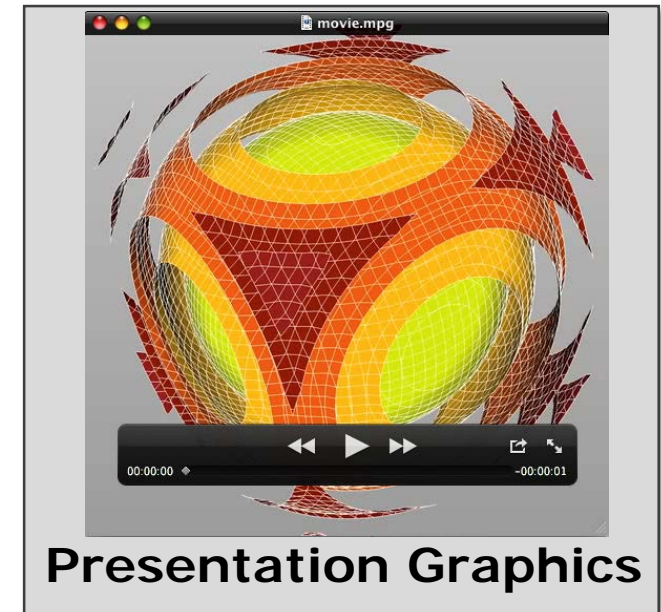
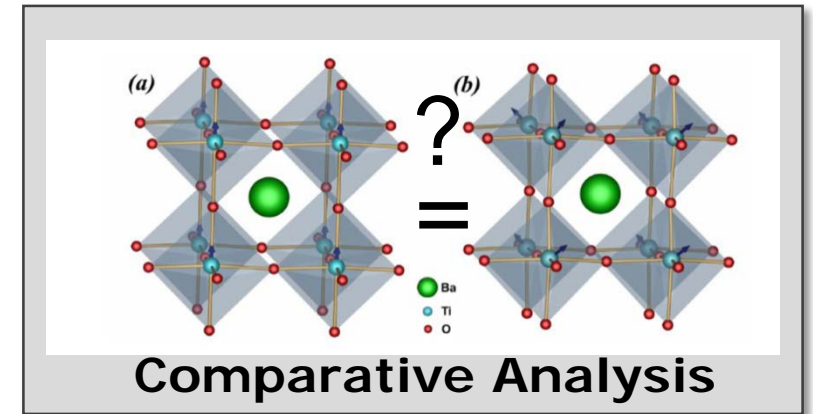
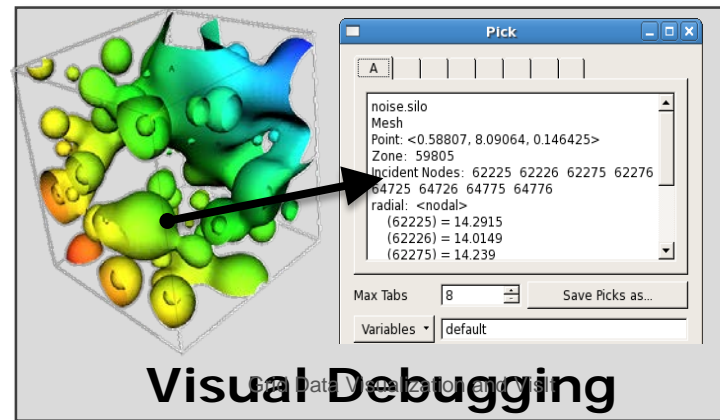
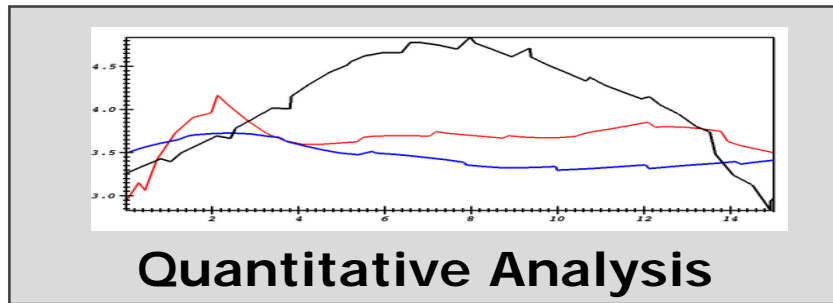
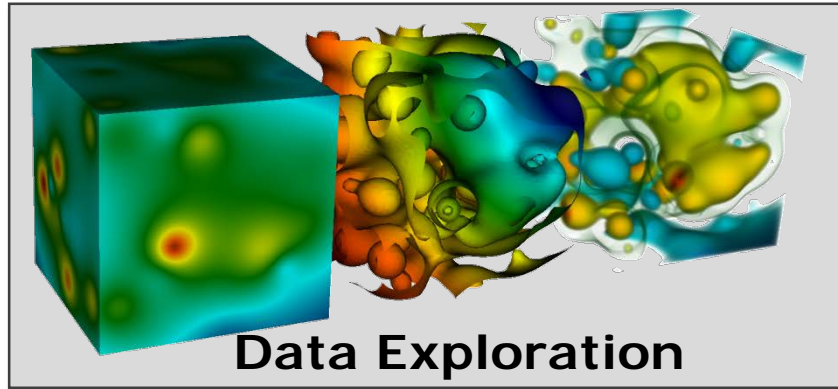
```
OpenGUI()
```



# Intro to VisIt

---

# Visualization is many complementary things





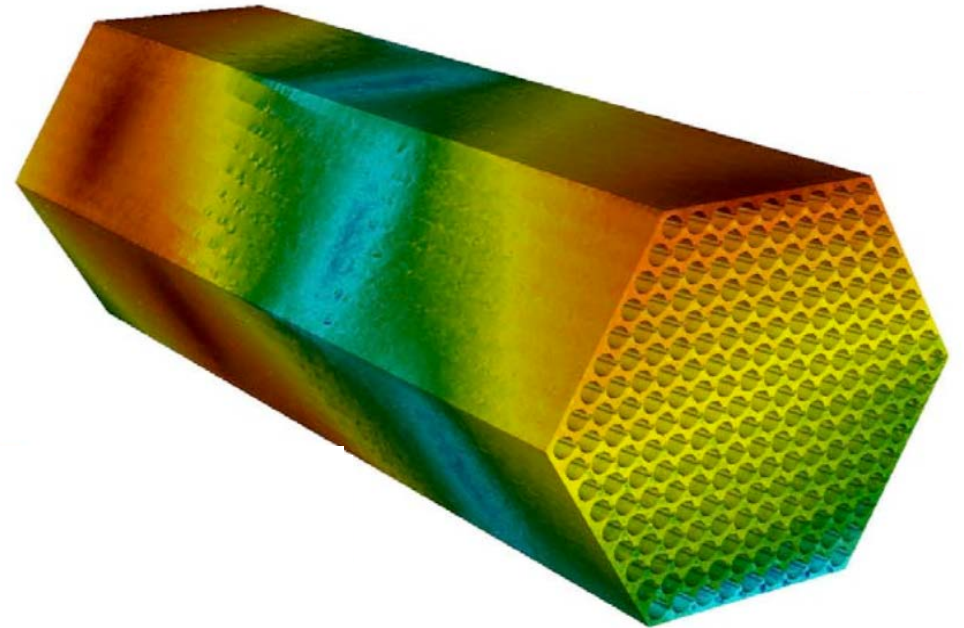
# Terminology

- Meshes: discretization of physical space
  - Contains “zones” / “cells” / “elements”
  - Contains “nodes” / “points” / “vertices”
    - VisIt speak: zone & node
- Fields: variables stored on a mesh
  - Scalar: 1 value per zone/node
    - Example: pressure, density, temperature
  - Vector: 3 values per zone/node (direction)
    - Example: velocity
      - Note: 2 values for 2D, 3 values for 3D
  - More fields discussed later...

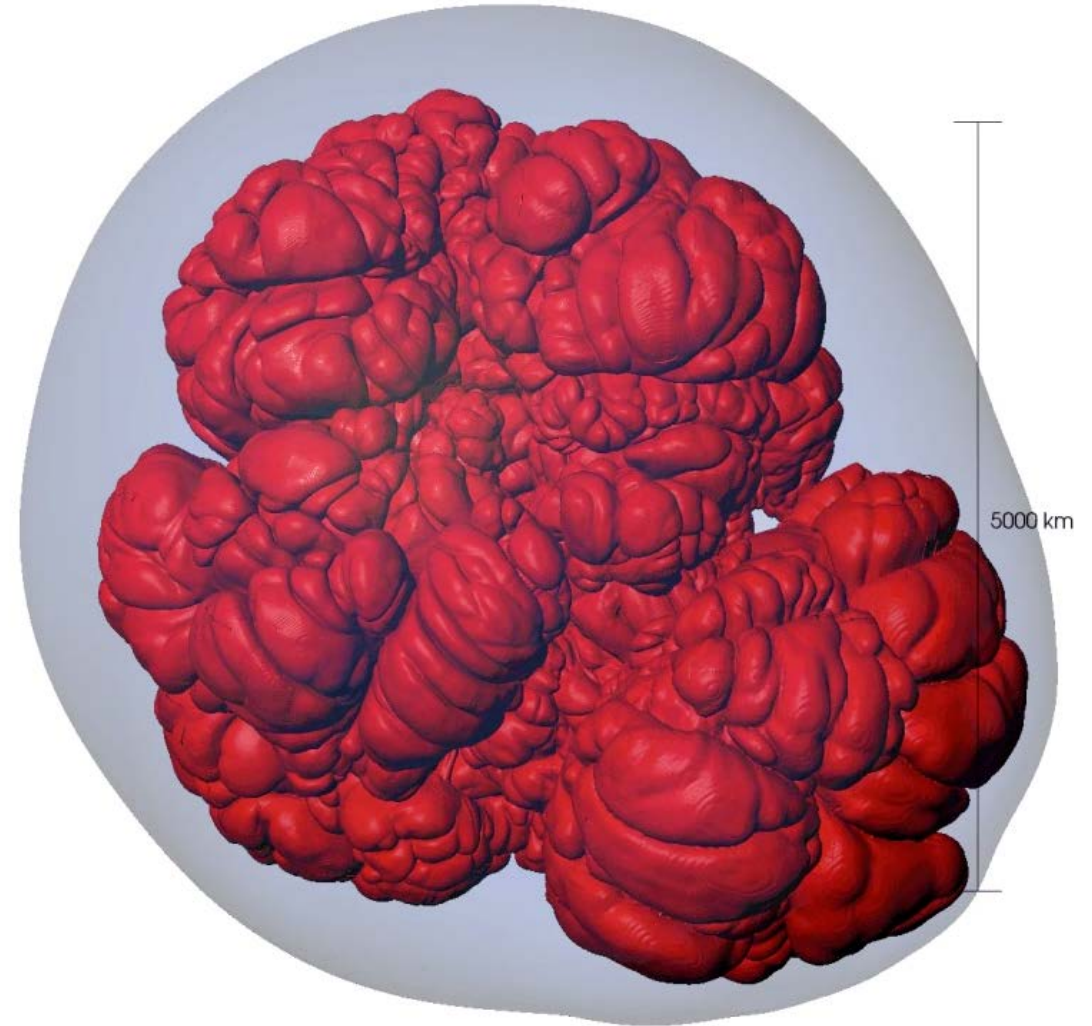


# Pseudocolor

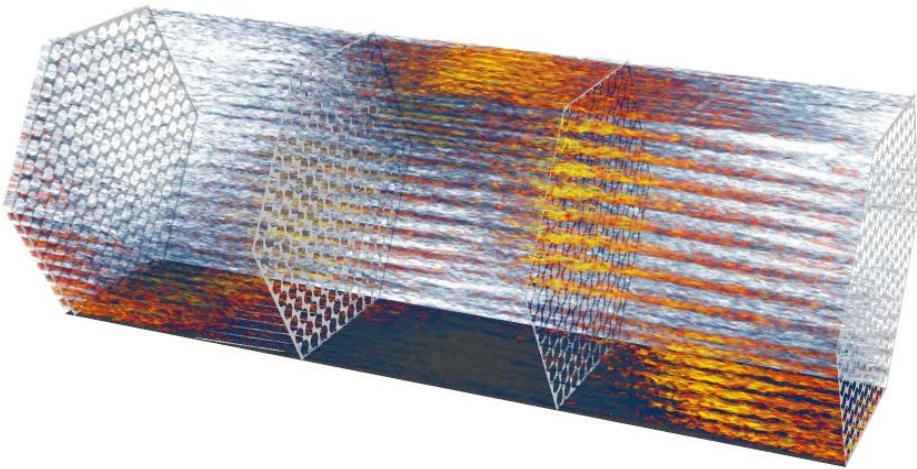
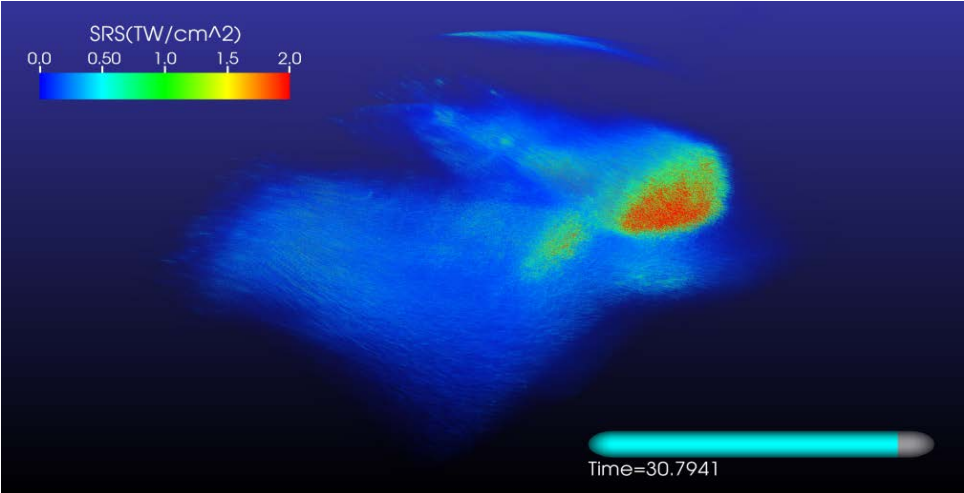
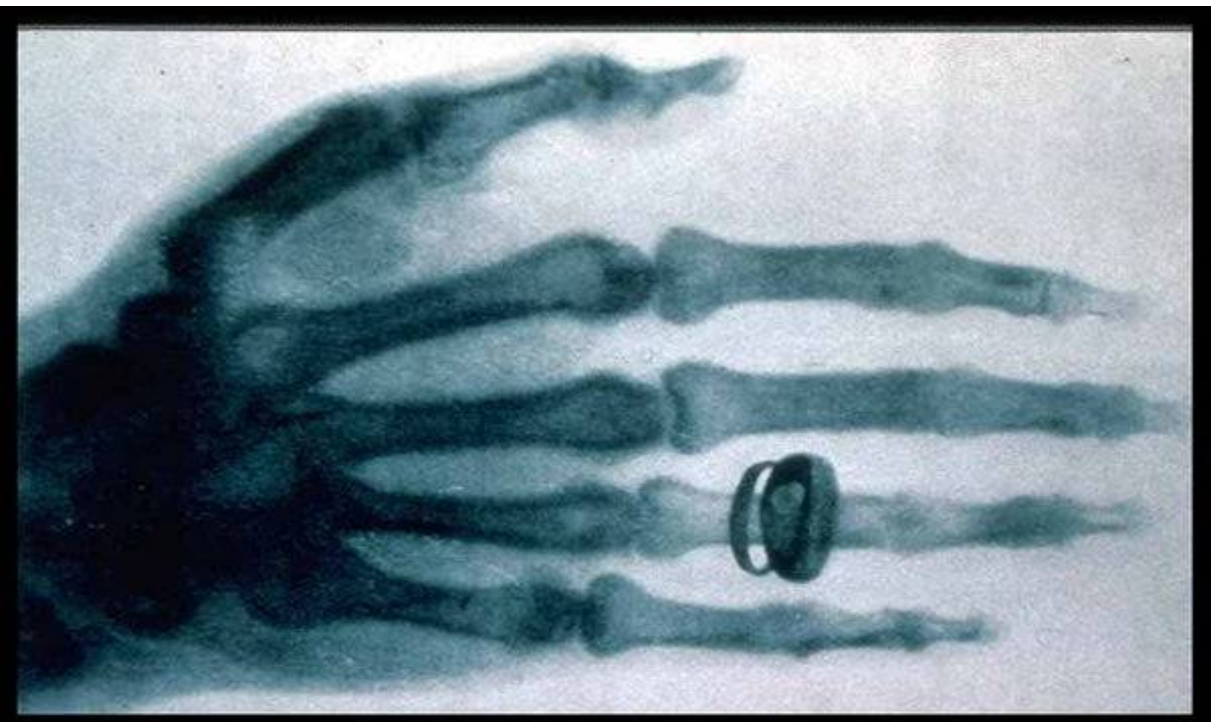
- Maps scalar fields (e.g., density, pressure, temperature) to colors.



# Contour / Isosurface



# Volume rendering

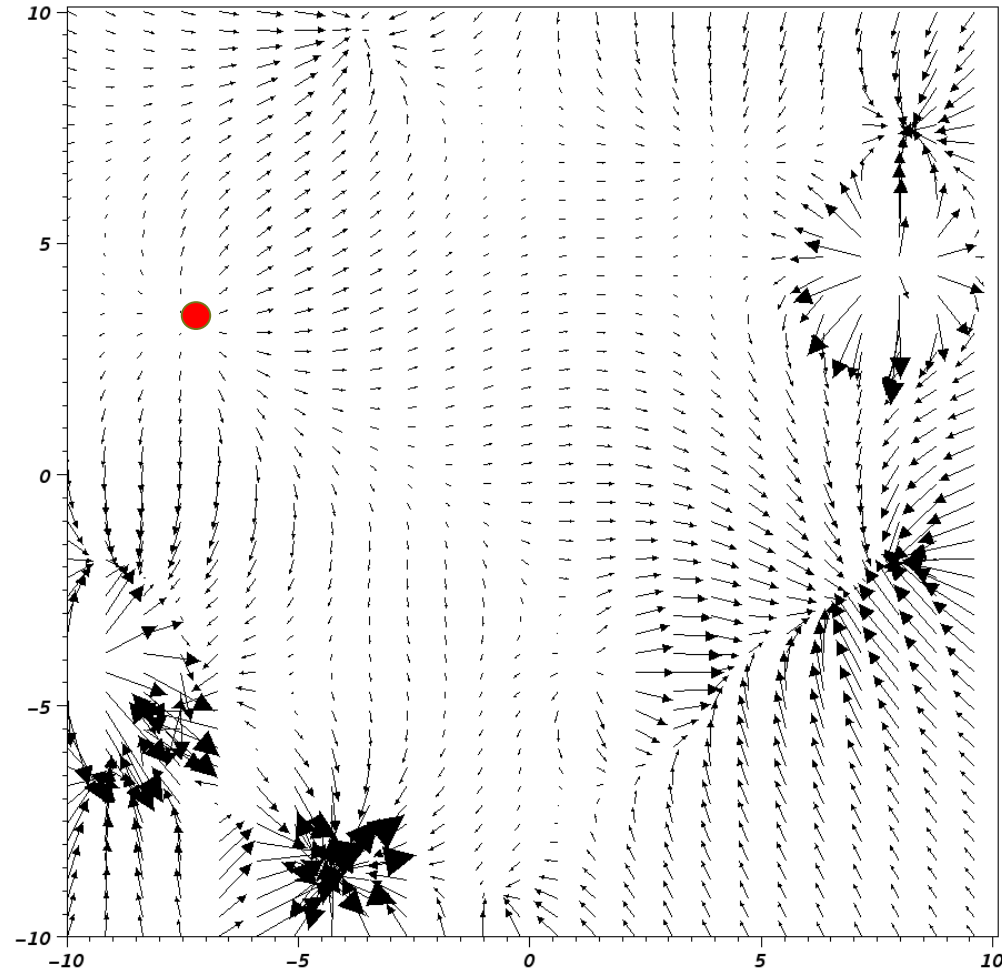




# Particle advection: the foundation of flow visualization

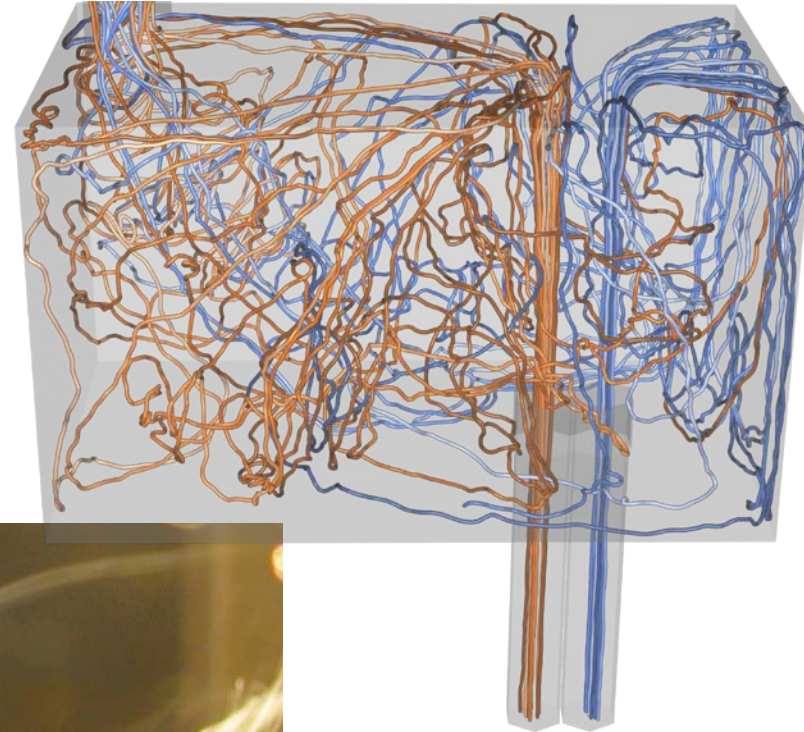
- Displace massless particle based on velocity field
- $S(t)$  = position of curve at time  $t$ 
  - $S(t_0) = p_0$ 
    - $t_0$ : initial time
    - $p_0$ : initial position
  - $S'(t) = v(t, S(t))$ 
    - $v(t, p)$ : velocity at time  $t$  and position  $p$
    - $S'(t)$ : derivative of the integral curve at time  $t$

**This is an ordinary differential equation**





# Streamlines

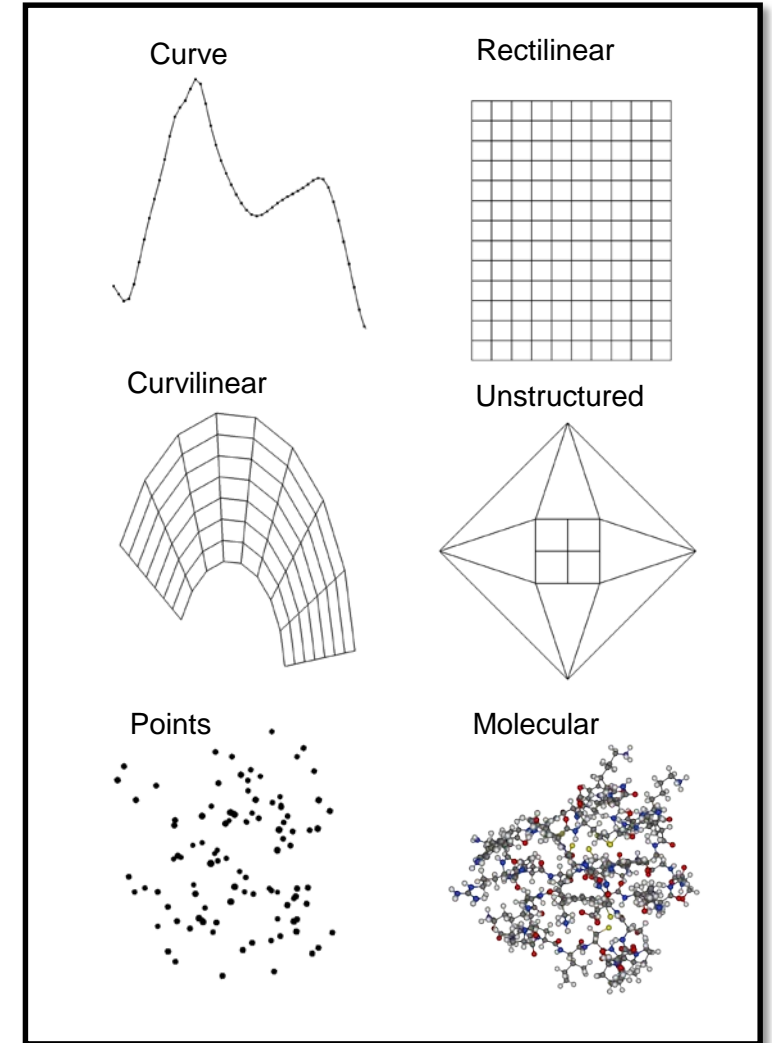


Grid Data Visualization and VTK

# Meshes

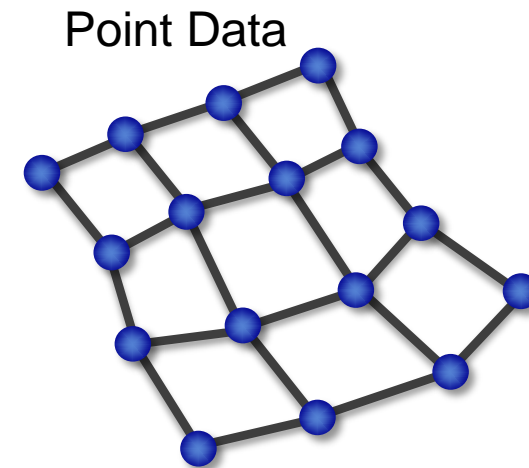
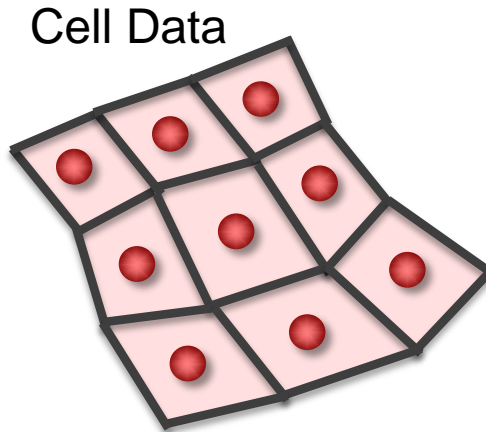
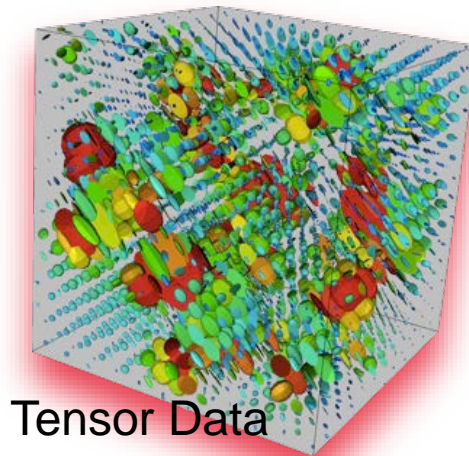
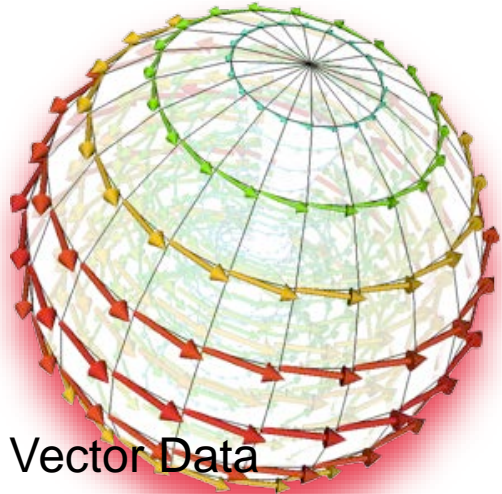
- All data in VisIt lives on a mesh
- Discretizes space into points and cells
  - (1D, 2D, 3D) + time
  - Mesh dimension need not match spatial dimension (*e.g. 2D surface in 3D space*)
- Provides a place for data to be located
- Defines how data is interpolated

## Mesh Types



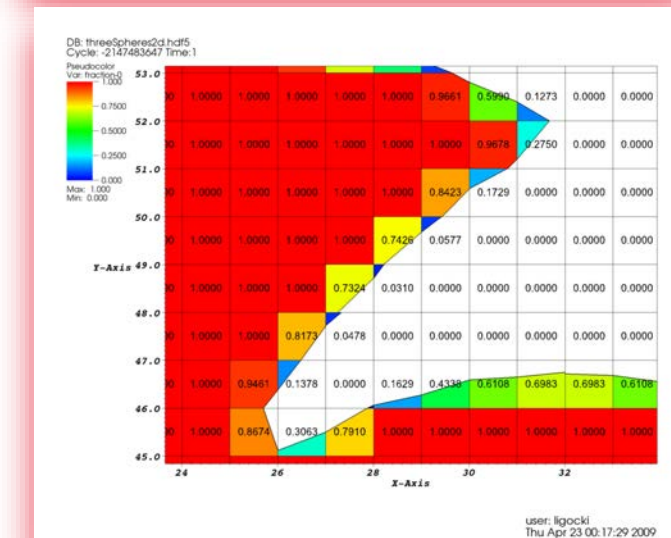
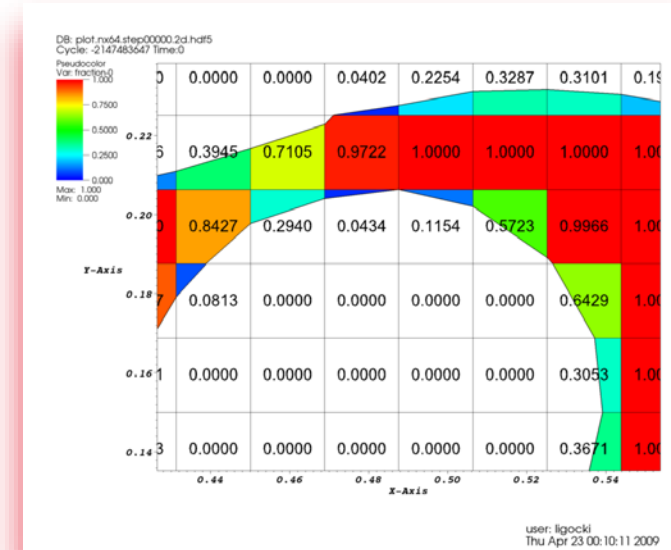
# Variables

- Scalars, Vectors, Tensors
- Associated with points or cells of a mesh
  - Points: linear interpolation
  - Cells: piecewise constant
- Can have different dimensionality than the mesh (e.g. 3D vector data on a 2D mesh)



# Materials

- Describes disjoint spatial regions at a sub-grid level
- Volume/area fractions
- VisIt will do high-quality sub-grid material interface reconstruction



# Species

- Similar to materials, describes sub-grid variable composition
  - Example: *Material “Air” is made of species “N<sub>2</sub>”, “O<sub>2</sub>”, “Ar”, “CO<sub>2</sub>”, etc.*
- Used for mass fractions
- Generally used to weight other scalars (e.g. partial pressure)

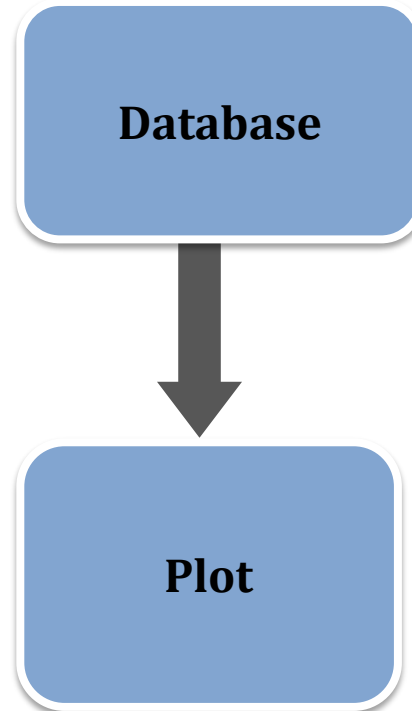
## VisIt's core abstractions

- **Databases:** How datasets are read
- **Plots:** How you render data
- **Operators:** How you manipulate data
- **Expressions:** Mechanism for generating derived quantities
- **Queries:** How to access quantitative information



# Examples of VisIt Pipelines

- Databases: how you read data
- Plots: how you render data
- Operators: how you transform/manipulate data
- Expressions: how you create new fields
- Queries: how you pull out quantitative information

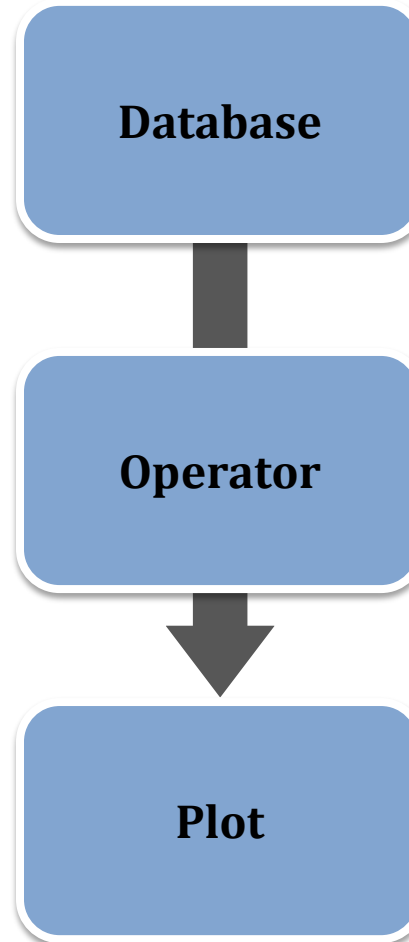


Open a database, which reads from a file (example: open file1.hdf5)

Make a plot of a variable in the database (example: Volume plot)

# Examples of VisIt Pipelines

- Databases: how you read data
- Plots: how you render data
- Operators: how you transform/manipulate data
- Expressions: how you create new fields
- Queries: how you pull out quantitative information



Open a database, which reads from a file (example: open file1.hdf5)

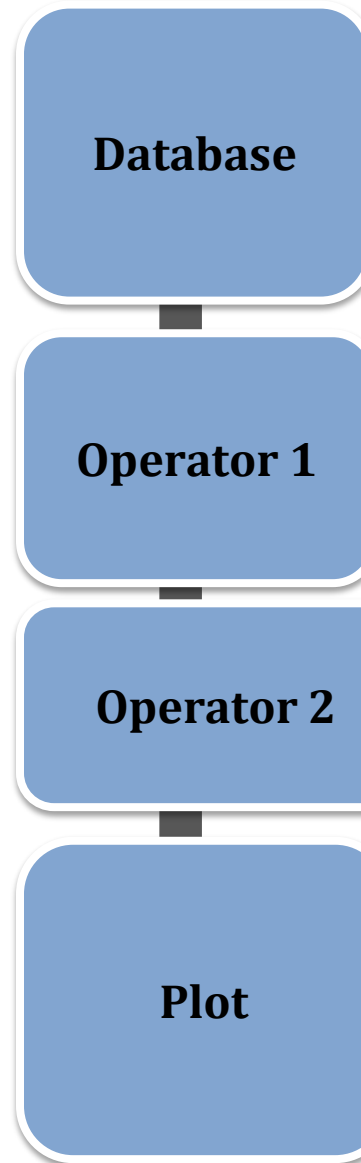
Apply an operator to transform the data (example: Slice operator)

Plot a variable in the database (example: Pseudocolor plot)



# Examples of VisIt Pipelines

- Databases: how you read data
- Plots: how you render data
- Operators: how you transform/manipulate data
- Expressions: how you create new fields
- Queries: how you pull out quantitative information



Open a database,  
which reads from a file  
(example: open file1.hdf5)

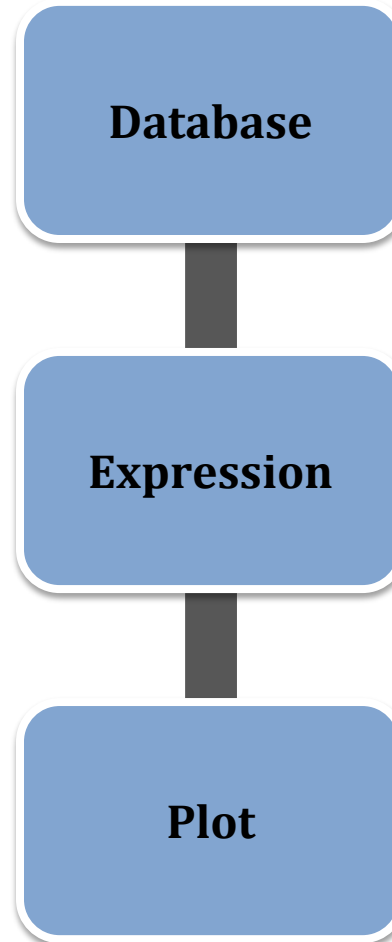
Apply an operator to  
transform the data  
(example: Slice operator)

Apply a second operator to  
transform the data  
(example: Elevate operator)

Plot a variable in the  
database  
(example: Pseudocolor plot)

# Examples of VisIt Pipelines

- Databases: how you read data
- Plots: how you render data
- Operators: how you transform/manipulate data
- Expressions: how you create new fields
- Queries: how you pull out quantitative information



Open a database, which reads from a file (example: open file1.hdf5)

Create derived quantities from fields in the file (example: magnitude(velocity))

Plot the expression variable (example: Pseudocolor plot)

# Expressions and Queries

**Expressions** in VisIt create new mesh variables from existing ones. These are also known as *derived quantities*. VisIt's expression system supports only derived quantities that create a new mesh variable defined over the *entire* mesh.

**Queries** define single numerical values

# Expressions and Queries examples

Given a mesh on which a variable named **pressure** is defined, an example of a very simple expression is

$2.0 * \text{pressure}$

On the other hand, suppose one wanted to sum (or integrate) pressure over the entire mesh. Such an operation is not an expression in VisIt because it does not result in a new variable defined over the **entire** mesh. In this example, summing pressure over the entire mesh results in a single, scalar, number, like 25.6.

Such an operation is supported by VisIt's Variable Sum Query.

# Expressions

## Predefined expressions

- VisIt defines several types of expressions automatically. For all vector variables from a database, VisIt will automatically define the associated magnitude expressions.
- For unstructured meshes, VisIt will automatically define mesh quality expressions.
- For any databases consisting of multiple time states, VisIt will define time derivative expressions.

# Derived quantities. Expressions

combine velocity components into a vector

- $\{ vx, vy, vz \}$

Extract X coordinate of a mesh

- `coords( mesh )[0]`

Gradient, vorticity, divergence, etc...

- `gradient( temperature )`

Conditional, relational and logical

- `if( gt ( temperature, 25.), <then-var>, <else-var> )`
- `ge(pressure, 0.8)`

Time-based

- `time_index_at_minimum( temperature, [, start-time-index, stop-time-index, stride])`
- `first_time_when_condition_is_true ( pressure_big, 100, 1, 71, 1)`

## Examples of VisIt Pipelines

- Databases: how you read data
- Plots: how you render data
- Operators: how you transform/manipulate data
- Expressions: how you create new fields
- Queries: how you pull out quantitative information

Database

Open a database, which reads from a file (example: open file1.hdf5)

Plot

Plot a field from the file (example: density + Pseudocolor plot)

Query

Extract quantitative information (example: integrate density to find mass)

# Examples of VisIt Pipelines

- **Databases:** how you read data
- **Plots:** how you render data
- **Operators:** how you transform/manipulate data
- **Expressions:** how you create new fields
- **Queries:** how you pull out quantitative information

**Database**

Open a database,  
which reads from a file  
(example: open file1.hdf5)

**Expression**

Create derived quantities from  
fields in the file  
(example: magnitude(velocity))

**Operator 1**

Apply an operator to transform  
the data  
(example: Slice operator)

**Operator 2**

Apply a second operator to  
transform the data  
(example: Elevate operator)

**Plot**

Plot a field  
(example: speed + Pseudocolor  
plot)

**Query**

Extract quantitative information  
(example: maximum speed over  
cross-section)





**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# Python scripting in VisIt

---

# Python scripting

- Easy
- A great time-saver
- Built by examples, or in incremental mode
- Not complete. Needs some fine-tuning to move from interactive use to batch mode

## See the Wiki

[VisIt-tutorial-Python-scripting](#)

The wiki refers to a file called “example.silo”. It is actually a clone of “noise.silo” (from the distribution)

# More expressions

Cross-mesh field evaluation (CMFE) are a more advanced type of Expressions

# Cross-mesh field evaluation (CMFE)

- Use the expressions seen earlier and add connectivity-based, or position-based evaluations
- The expressions to evaluate **p** from file **a.00000** with a default value of **0** onto the mesh **mesh\_3d** are:
  - `pos_cmfe(<a.00000:p>, mesh_3d, 0)`
  - `conn_cmfe(<a.00000:p>, mesh_3d)`
- The expressions can be complicated, but there is a GUI wizard to assist you in writing the correct syntax
- Examples:

# CMFE Example 1

Difference between two datasets:

N.B. (variable "Density" exists in file f1, on "my\_mesh")

- `den_diff = "Density - conn_cmfe(<f2:den>, my_mesh)"`

Make a Pseudocolor plot of `den_diff`

Query with "Weighted Variable Sum". This will integrate the differences (meaning using volume weighting in 3D or area-weighting in 2D)

## CMFE Example 2

Make an average over the previous three time slices:

$$\begin{aligned} & (conn\_cmfe(<[-1]id:varname>, meshname) + \\ & conn\_cmfe(<[-2]id:varname>, meshname) + \\ & conn\_cmfe(<[-3]id:varname>, meshname) ) / 3. \end{aligned}$$

`[-1]id` -> 'i' means index (as opposed to 'c' for cycle or 't' for time), 'd' means "delta".

## CMFE Example 2

Better yet. VisIt has a pre-defined expression:

*average\_over\_time(<varname> [, "pos\_cmfe", <fillvar-for-uncovered-regions>] [, start-index, stop-index, stride])*

*e.g.,*

*average\_over\_time(temp\_F , 0, 98, 1)*



# Query-driven Analysis

---

# Query-driven Analysis

Query-driven analysis based on single timestep queries is a versatile tool for the identification and extraction of temporally persistent and instantaneous data features.

But there is more...

Temporal tracking and refinement of selections based on information from multiple timesteps can support detailed analysis of the temporal evolution of data features.

⇒ Do Cumulative Queries

# Query-driven Analysis

Cumulative Queries open the way to refinement (secondary queries):

1. select records that match the primary query most or least frequently,
2. select only records that match the primary query within a given time frame or at timesteps with a particularly high or low number of matches,
3. refine the query based on information of data values that have not been used in query but which show interesting trends with respect to the data subset retrieved by the query.

# Query-driven Analysis

Example in climatology (earth surface temperature over 100 years):

- see the areas on the globe that have warmed the most and have an idea how the warming has progressed.
- see which regions have most frequently been warmer by at least 5 degrees

# Time query and histogram-based queries

## Example:

- Given a climate dataset with the Earth's surface temperature for a period of 100 years.

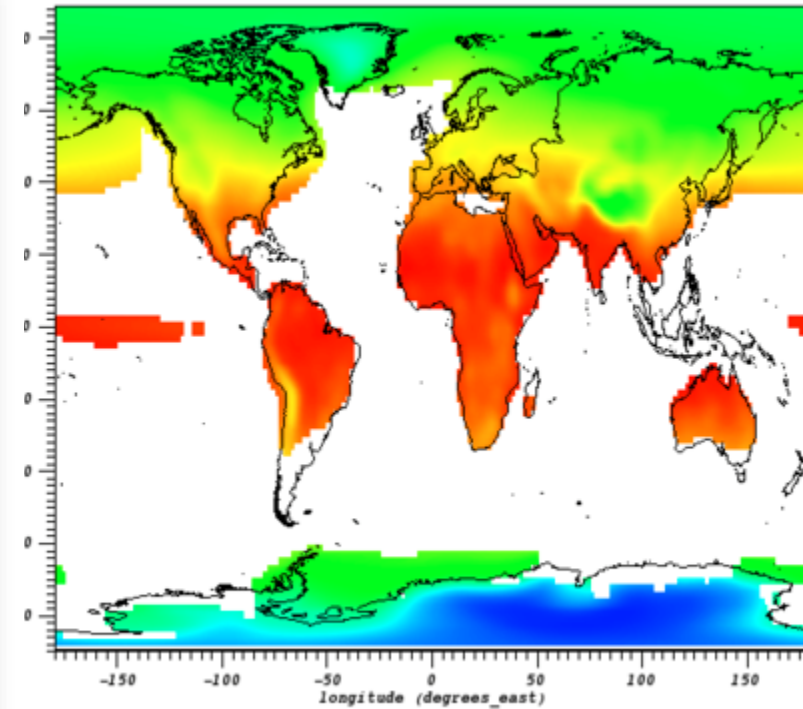
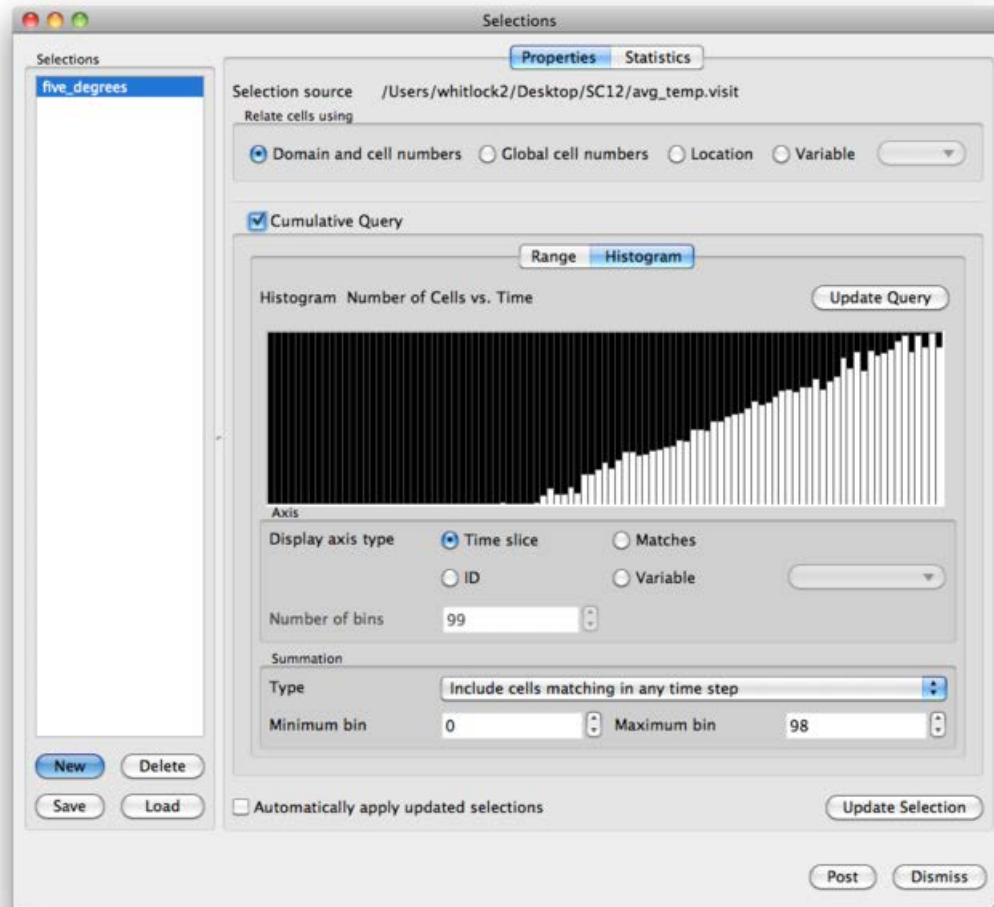
## Question:

Which areas on the globe have warmed the most and how has the warming progressed?

Compute the difference of each year vs. the first year;

Find which regions have most frequently been warmer by at least 5 degrees.

# Live demonstration



# Python code to do cumulative queries

```
OpenDatabase("localhost:/local/data/tutorial_data/continent.shp", 0)
AddPlot("Mesh", "polygon", 1, 0)
DrawPlots()
OpenDatabase("localhost:/local/data/tutorial_data/avg_temp.visit", 0)
CreateNamedSelection("five_degrees")
UpdateNamedSelection("five_degrees")
SaveNamedSelection("five_degrees")
AddPlot("Pseudocolor", "temp_F", 1, 0)
DrawPlots()
SetTimeSliderState(98)
ApplyNamedSelection("five_degrees")
```



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# Client Server and parallel execution

---



# Client Server and parallel execution

- Large data usually reside on the server-side (the supercomputer)
- There is no room, or no time, to copy it over locally
- A VisIt server will run close to the data, and communicate to a remote client
- You will need a host profile

## Opening a remote connection via python

```
args = ("-sshtunneling", "-np", "24", "-nn", "1", "-l", "srun",  
        "-dir", "/apps/daint/UES/6.0.UP02/VisIt/2.12", "-t", "00:05:00",  
        "-la", "--partition=mc -C mc")
```

```
host = "daint103.login.cscs.ch"
```

```
OpenComputeEngine(host, args)
```

```
OpenDatabase(host+":/apps/daint/UES/6.0.UP02/VisIt/2.12/data/multi_ucd3d.silo")
```

# Opening a remote connection to CINES

```
args = "-np", "24", "-nn", "1", "-l", "srun",  
        "-dir", "/scratch/cnt0022/cra6960/SHARED/VisIt/2.12", "-t", "00:05:00",  
        "-la", "--mpi=pmi2 -K1 --resv-ports")
```

```
host = "occigen.cines.fr"
```

```
OpenComputeEngine(host, args)
```

```
OpenDatabase(host+":/scratch/cnt0022/cra6960/SHARED/jet3d.amr5", 0,  
             "AMAZE_1.0")
```



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# In-situ visualization

---

# In-situ visualization

- TERMINOLOGY
  - In-situ processing strategies
- VisIt's libsim library
  - Source code instrumentation
  - Examples, Exercises

# Facts

- Parallel simulations are now ubiquitous
- The mesh size and number of timesteps are of unprecedented size
- The traditional ***post***-processing model “*compute-store-analyze*” does not scale

## Consequences:

- **Datasets are often under-sampled**
  - **Loss of precision in feature-tracking**
- **Many time steps are never archived**
- **It takes a supercomputer to re-load and visualize supercomputer 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 not read data from disk process it as it is being generated

***in-situ* visualization**

# in-situ (parallel) 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

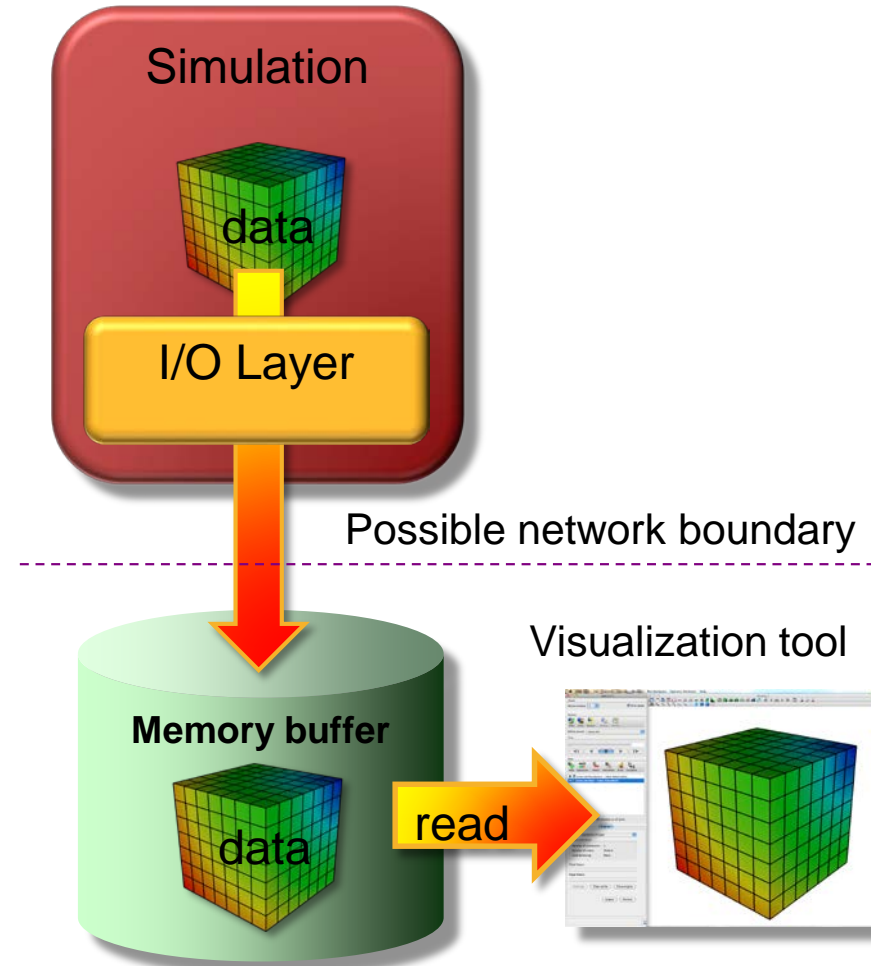


# in-situ Processing Strategies (old definitions)

In Situ Strategy	Description	Negative Aspects
<b>Loosely coupled</b> a.k.a. “Concurrent processing”	Visualization and analysis run on concurrent resources and access data over network	1) Data movement costs 2) Requires separate resources
<b>Tightly coupled</b> a.k.a. “Co-processing”	Visualization and analysis have direct access to memory of simulation code	1) Very memory constrained 2) Large potential impact (performance, crashes)
<b>Hybrid</b>	Data is reduced in a tightly coupled setting and sent to a concurrent resource	1) Complex 2) Shares negative aspects (to a lesser extent) of others

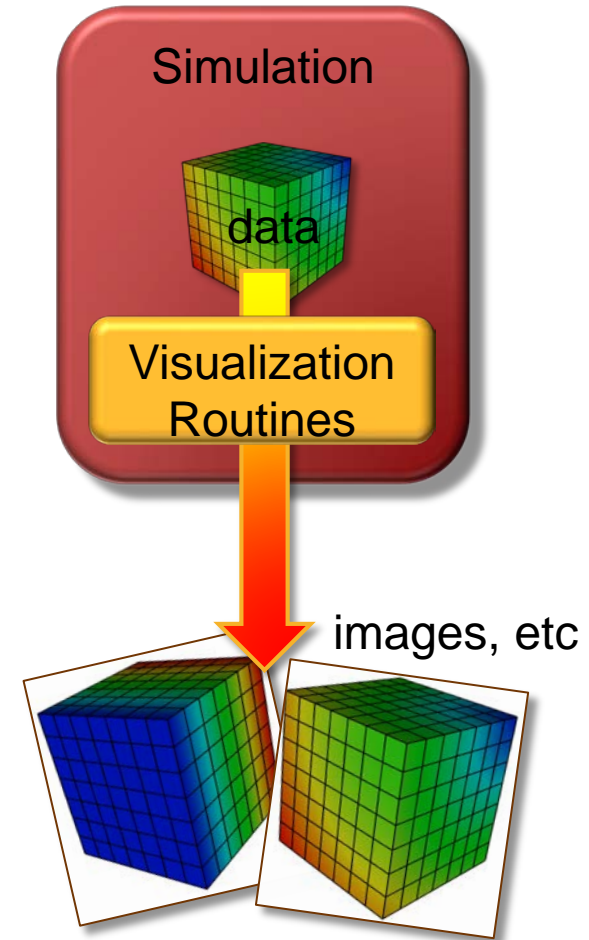
# Loosely Coupled in-situ Processing (old definition)

- 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



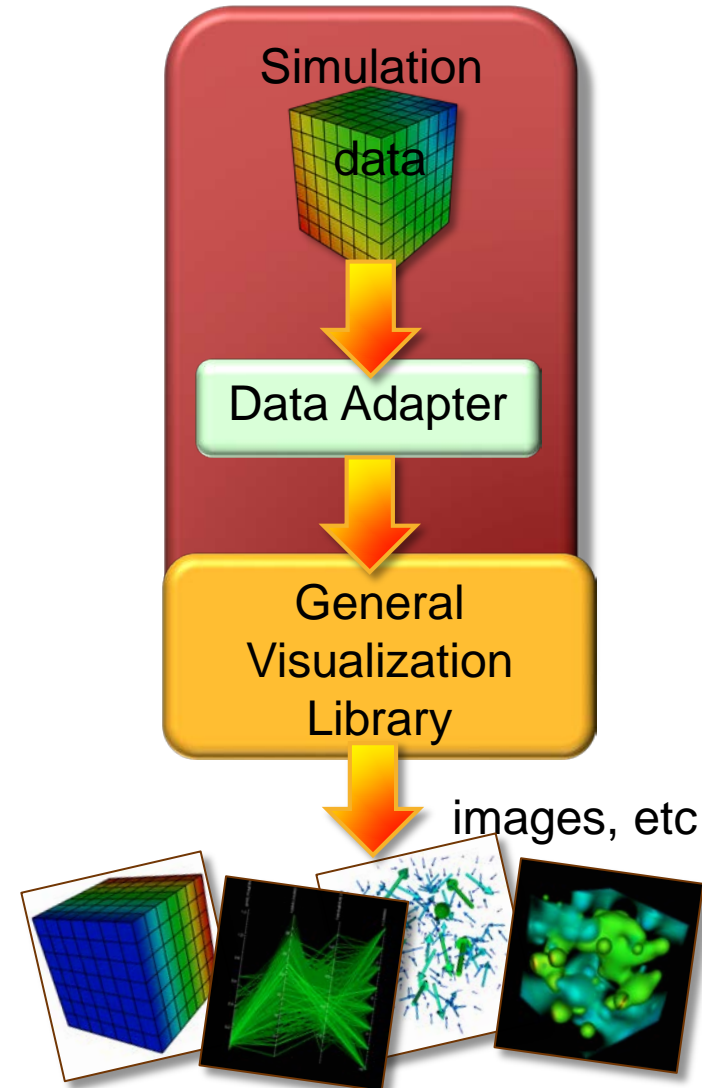
# Tightly Coupled *Custom* in-situ Processing (old definition)

- 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 above examples are by far not sufficient to clearly distinguish between all cases.
- There is now a rich and varied eco-system of in-situ implementations and the need has been felt to derive a taxonomy of the current solutions.
- => The in-situ Terminology Project
  - Led by Hank Childs, U. of Oregon
  - With contributions from a large group
  - See also the Eurovis2016 STAR report “In Situ Methods, Infrastructures and Applications on High Performance Computing Platforms”, A. Bauer et al.



# THE IN SITU TERMINOLOGY PROJECT

## PROJECT LEADER: HANK CHILDS

# Motivation

- Current terms are not descriptive enough
  - ▣ E.g., Tightly-coupled / loosely coupled mean different things to different people
  - ▣ In transit?
- Causes confusions for stakeholders, funding agencies, and even among visualization researchers
- Goal: build a community effort to standardize terminology
- Additional benefit: better understanding of the richness of the in situ space



# 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 Lediae, 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



# Approach

- Observation: lots of orthogonal dimensions that describe an in situ system
- Task 1: Identify those dimensions
- Task 2: Identify options for those dimensions
- So how is an in situ system described?
  - ▣ → description of in situ system is done by identifying which options they have chosen for a given dimension

# Axes (i.e., the orthogonal dimensions)

---

- Integration Type
- Proximity
- Access
- Division of Execution
- Operation Controls
- Output Type

# #1: Integration Type

---

- Application-Aware
- Application-Unaware

# #1: Integration Type

## □ Application-Aware

- ▣ Bespoke: custom vis and analysis written specifically for a single simulation code and tailored to its needs
- ▣ Dedicated API: the framework is dedicated to visualization and analysis, and so the simulation code is aware that interactions with this API are for the purpose of visualization and analysis
- ▣ Multi-purpose API: the scope of the framework is data, meaning that it includes visualization and analysis, but that it also might include I/O or data movement between components

# #1: Integration Type

## □ Application-Unaware

- Interposition: placing a dynamically-loaded library containing symbols known to the simulation code into the place of the original library that the simulation code was expecting (LD\_PRELOAD)
- Inspection: inspect memory to infer patterns in data layout and automatically add in situ processing (debuggers do this)
- ???

## #2: Proximity

---

- On Node
- Off Node, Same Computing Resource
- Different Computing Resource

# #2: Proximity

- On Node ← use memory hierarchy
  - ▣ registers
  - ▣ L1-cache
  - ▣ L2-cache
  - ▣ L3-cache
  - ▣ Random-access memory
  - ▣ NUMA accesses
  - ▣ NV-RAM
  - ▣ Local disk

## #2: Proximity

- Off Node, Same Computing Resource
  - ▣ Quantify number of hops?
- Different Computing Resource
  - ▣ Strains usage of term in situ?



# #3: Access

- Direct Access: in situ routines run in the same logical memory space as the simulation code
- Indirect Access: in situ routines run in a distinct logical memory space from the simulation code

# #3: Access

- Direct Access: in situ routines run in the same logical memory space as the simulation code
  - ▣ Deep Copy: in situ routines make a copy of its input data from the simulation
  - ▣ Shallow Copy: adapt data structures to those of the simulation code

# Proximity + Access

	On Node Proximity	Off Node / Distinct Resources Proximity
Direct Access	Yes! “in situ”...	This is possible! (PGAS) Is it “tightly coupled”?
Indirect Access	This makes sense! Is it “in transit”?	Yes! “in transit”...

# #4: Division of Execution

- Space Division: subset of the compute resources are devoted exclusively to in situ routines
- Time Division: no compute resources are devoted exclusively to in situ routines
  - ▣ Some (or all) of the compute resources alternate between advancing the simulation and visualization and analysis.

# #5: Operation Controls

- Automatic: users select which operations to perform in advance of the calculation
- Human-in-the-Loop: stakeholders modify which visualization and analysis routines are executed in situ

# #5: Operation Controls

- Automatic: users select which operations to perform in advance of the calculation
  - ▣ Adaptive: the in situ routines can adapt which operations are performed as the simulation executes
  - ▣ Non-adaptive: in situ routines are static

# #5: Operation Controls

- Human-in-the-Loop: stakeholders modify which visualization and analysis routines are executed in situ
  - ▣ Blocking: simulation can pause when waiting for guidance from a stakeholder
  - ▣ Non-blocking: simulation will not pause to wait for guidance from a stakeholder

# #6: Output Type

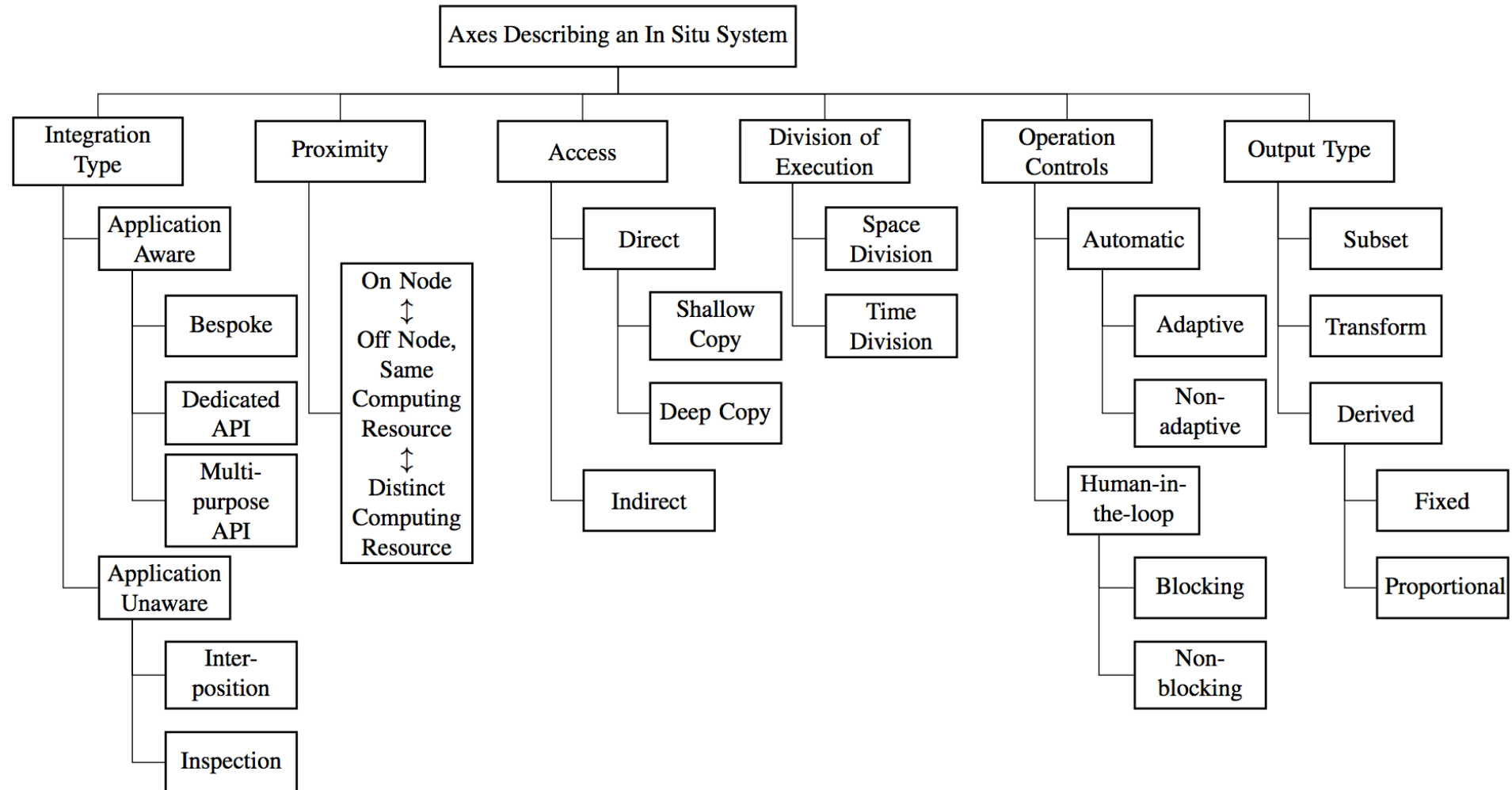
- Subset: selectively pick elements of the data
- Transform: perform an operation
  - ▣ Examples: wavelet compression, Lagrangian flow (?)
- Derived: Generate new data of a different nature than the input
  - ▣ Examples: statistics, isosurfaces, topological analyses, rendering, indexing
  - ▣ Two sub-types:
    - Derived: fixed
    - Derived: proportional



# What about “hybrid in situ”?

- 1) Transform data as simulation is generating it
  - 2) Send results to distinct resources
  - 3) Apply vis algorithms to transformed data on distinct resources
- For example, is the access direct or indirect?
    - ▣ Answer: direct in (1), but indirect in (3)
  - Solution: identify systems and score each system separately

# All categories & options



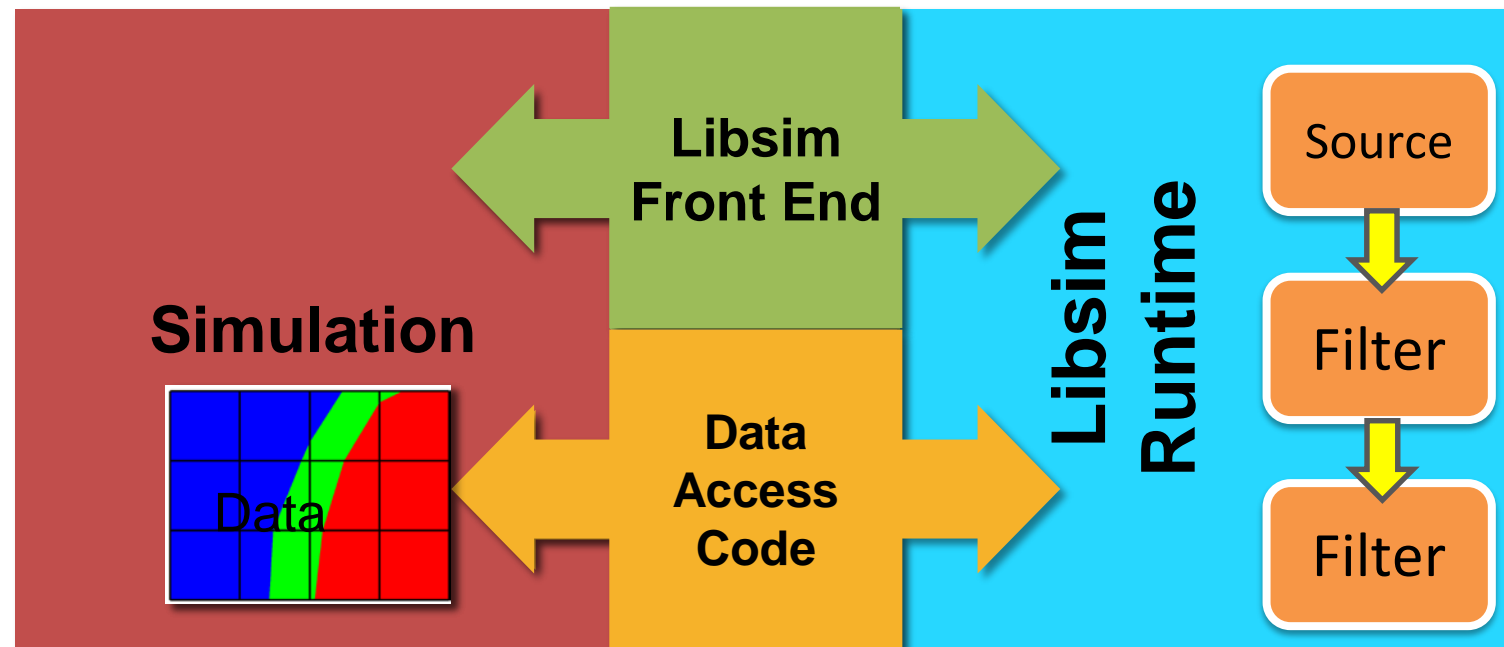
# Using the in-situ terminology defined earlier

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.

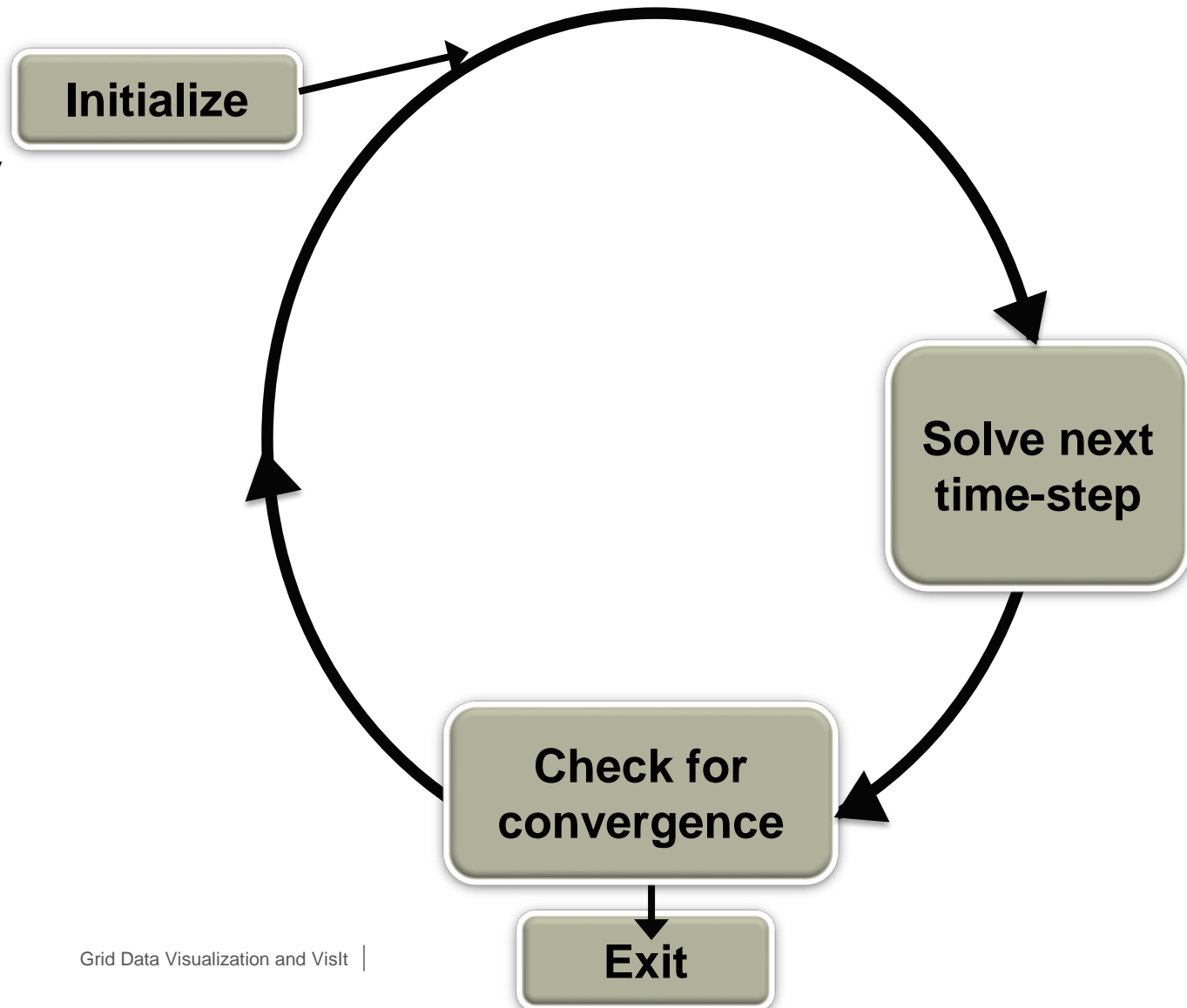
Add control functions that let VisIt steer the simulation.

# 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

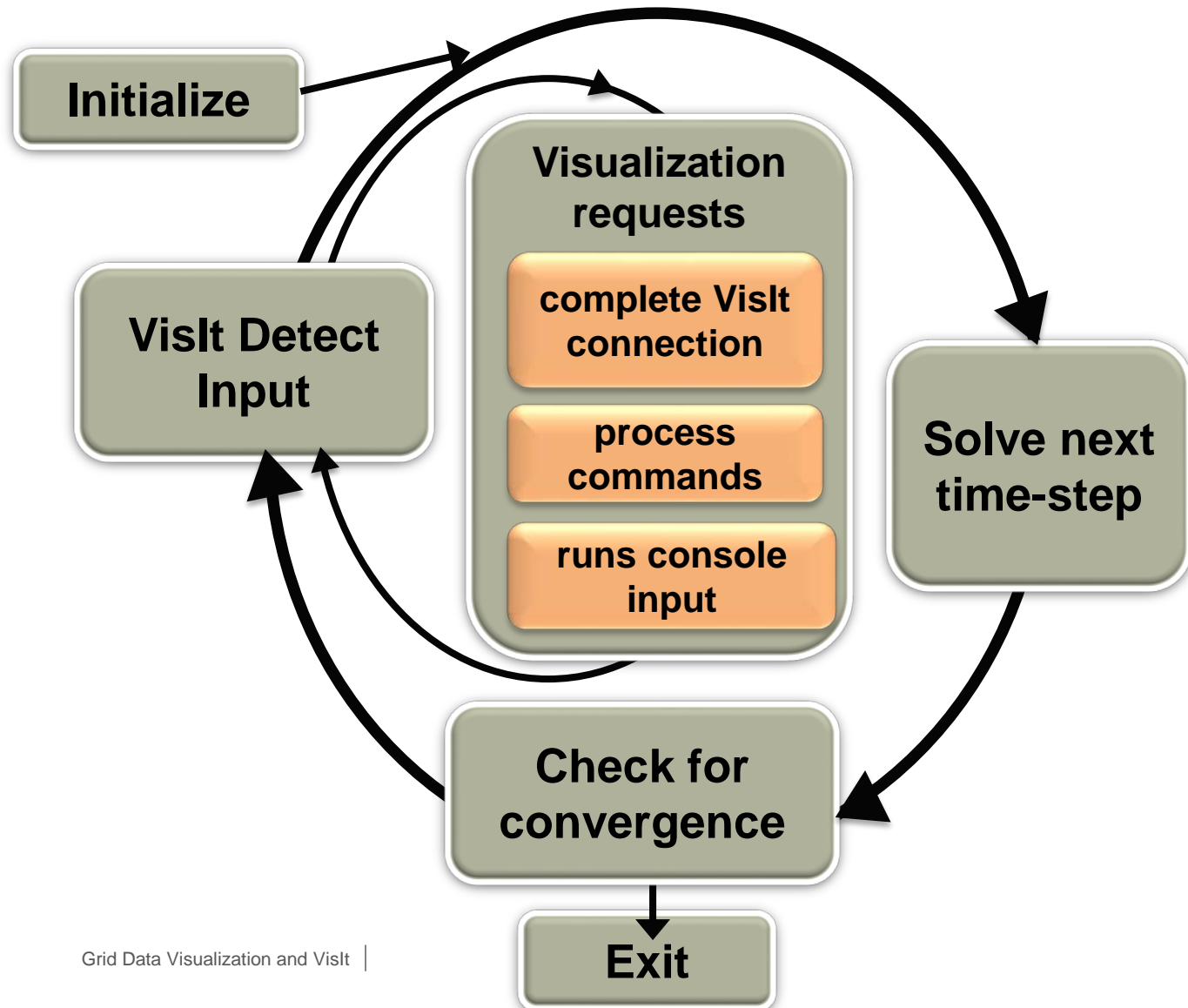


# VisIt in-the-loop

- 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.

# Exercises

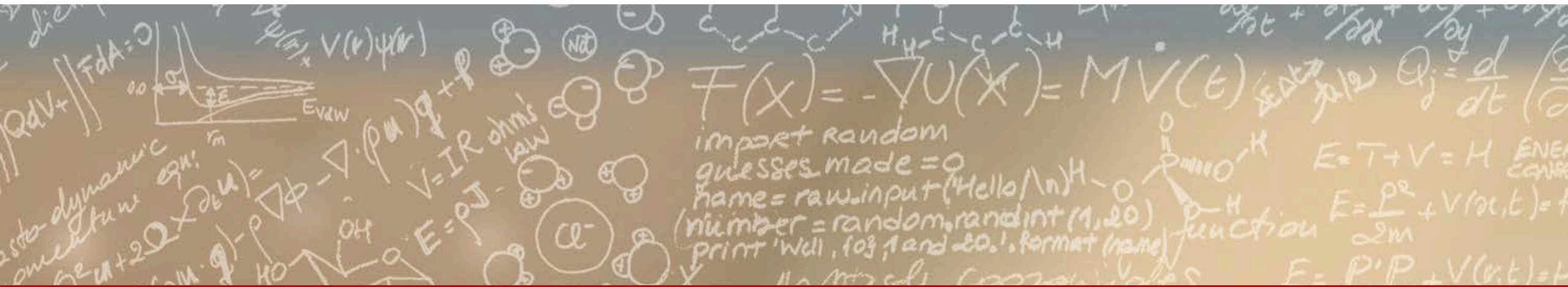
- Parallel simulations will only run if VisIt itself has been compiled with `–parallel`
- The `updateplots.{c,py}` are the easiest to demonstrate `libsim`
- The 2D Jacobi example (serial or parallel) is more advanced.
- Download <ftp://ftp.cscs.ch/out/jfavre/VisIt/InSitu/>



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



**Thank you for your attention.**