



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Grid data visualization and VisIt

International School of Computational Astrophysics

Jean M. Favre, CSCS

May 25-27, 2016

Outline

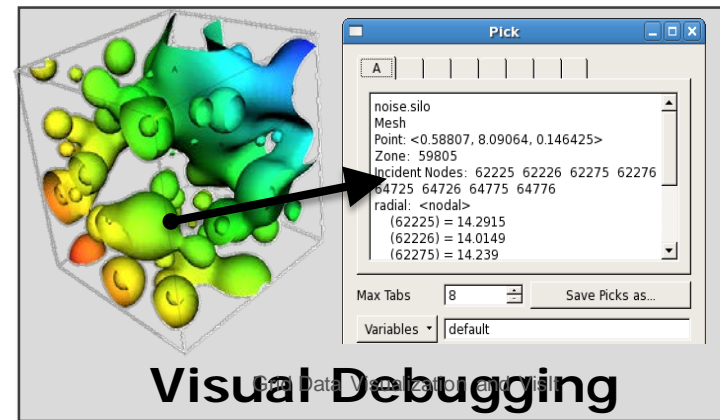
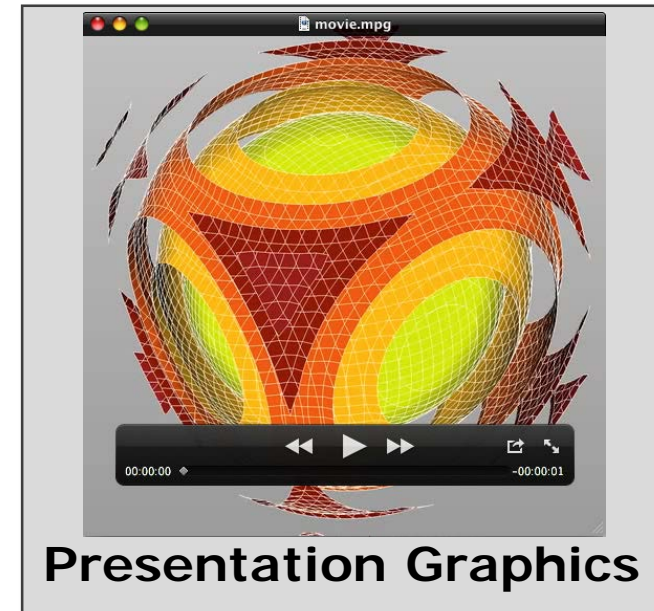
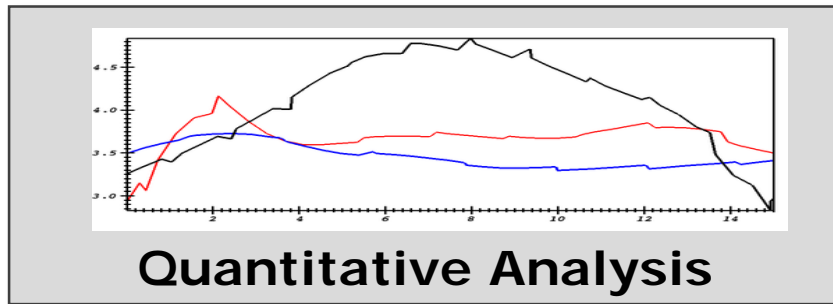
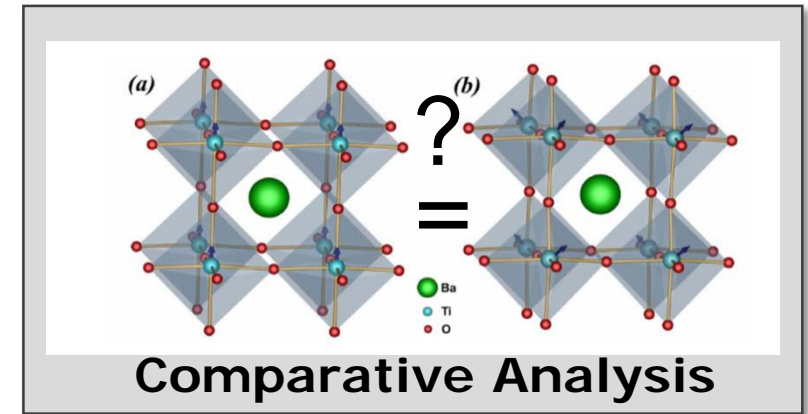
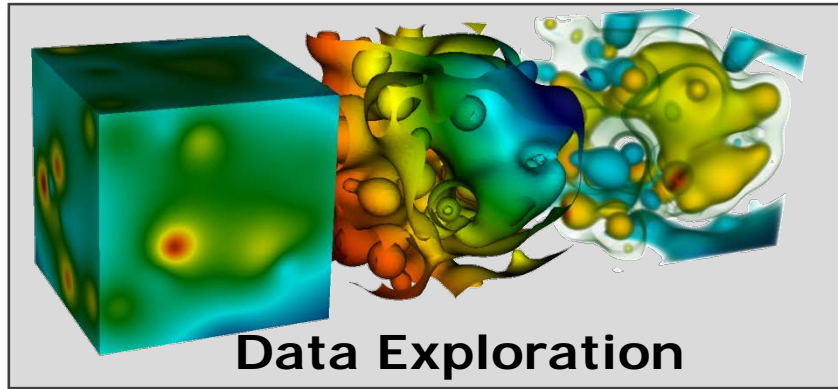
- Introduction to VisIt (10 min)
- Data format and I/O libs (15 min)
- AMR Vis (e.g. Chombo) (15 min)
- Pause (5 min)
- Python scripting (10 min)
- Filtering and Analysis (20 min)
- Intro to In-situ Visualization (15 min)

Plan

- Re-use parts of tutorials which are on-line on the very rich Wiki of visitusers.org
- Give some live demonstration
- Give a general idea of the numerous and various features of VisIt. The time is very short to be exhaustive.

- Fasten your seatbelt. I am going to go fast. I apologize. 😊
- I will be available as much as possible, even after the exercises to address your questions. Do not hesitate to come to me.

Visualization is many complementary things





CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

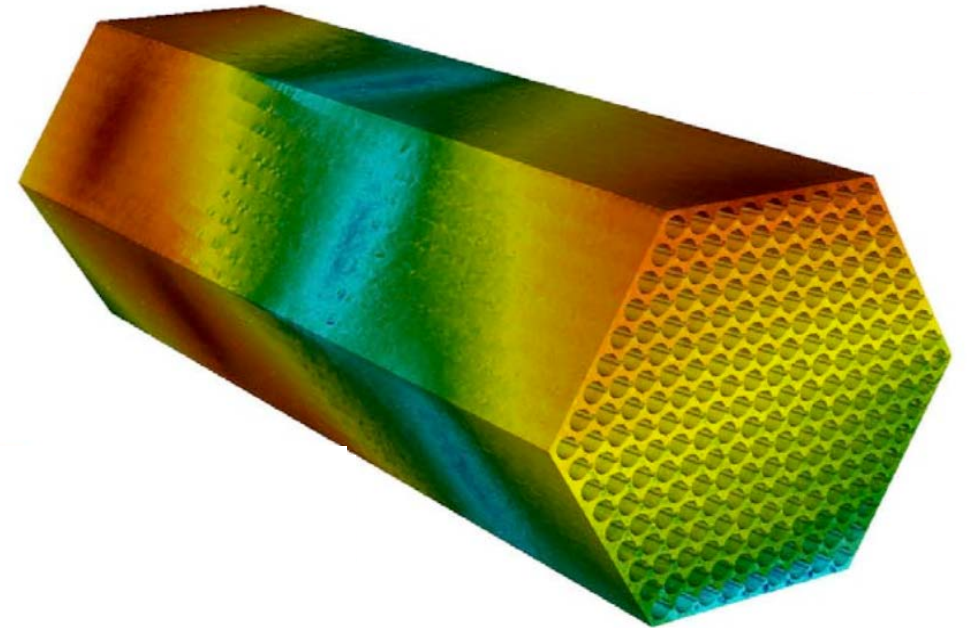
The VisIt Visualization Pipeline (borrowed from the SC13 tutorial)

Terminology

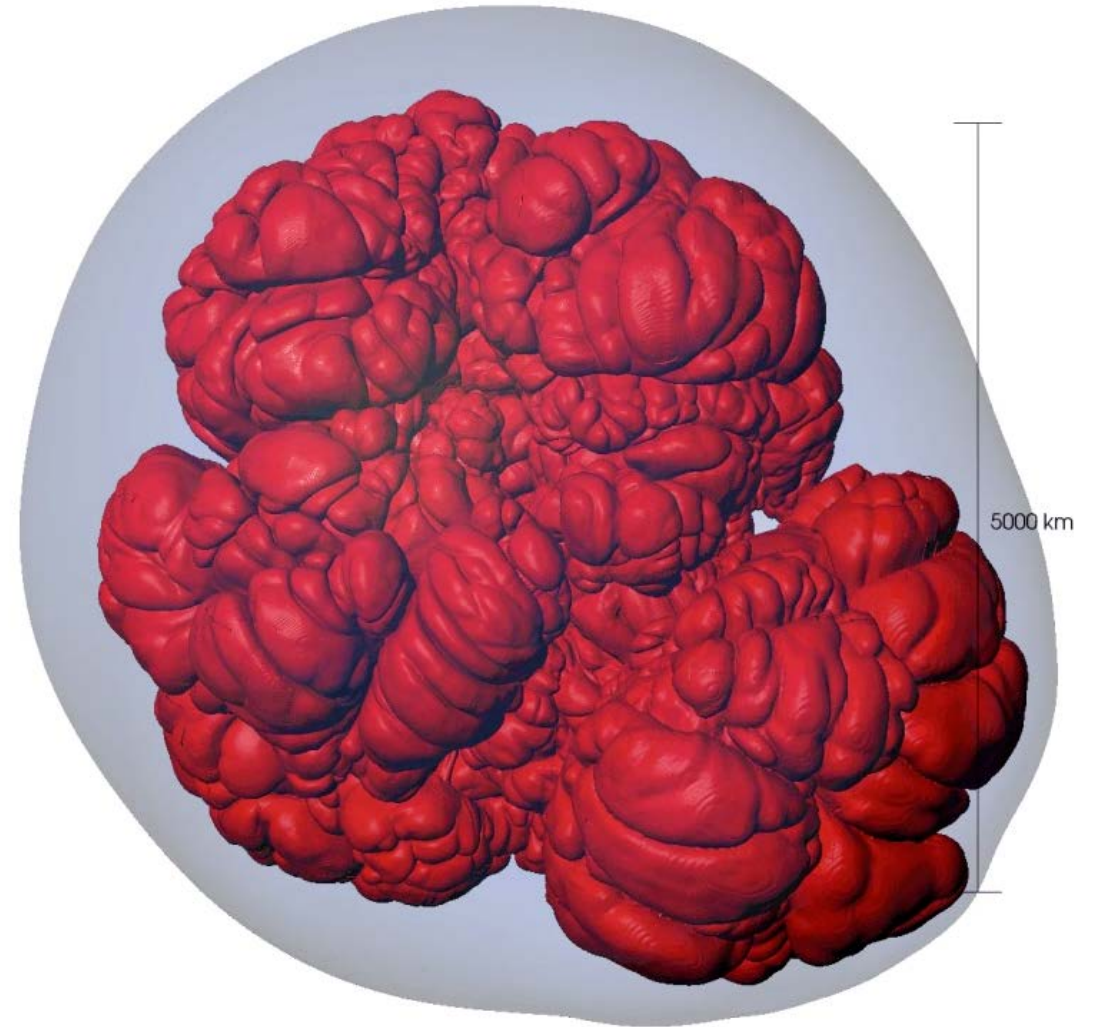
- Meshes: discretization of physical space
 - Contains “zones” / “cells” / “elements”
 - Contains “nodes” / “points” / “vertices”
 - VisIt speak: zone & node
- Fields: variables stored on a mesh
 - Scalar: 1 value per zone/node
 - Example: pressure, density, temperature
 - Vector: 3 values per zone/node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

Pseudocolor

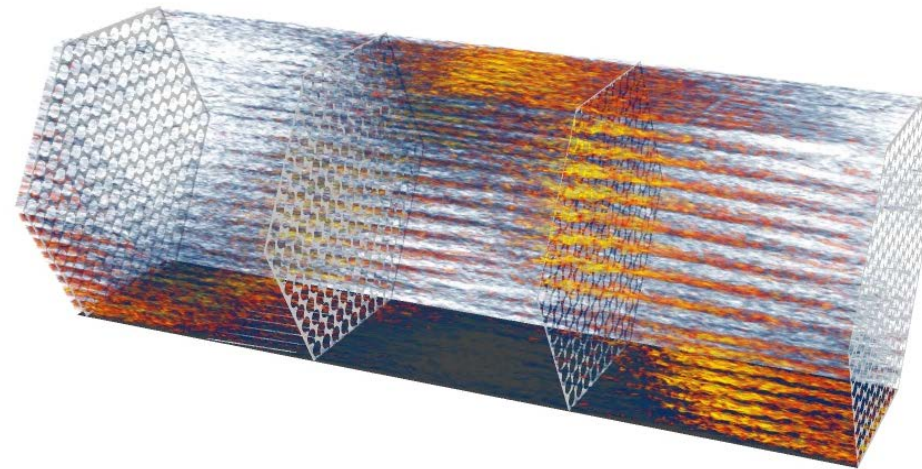
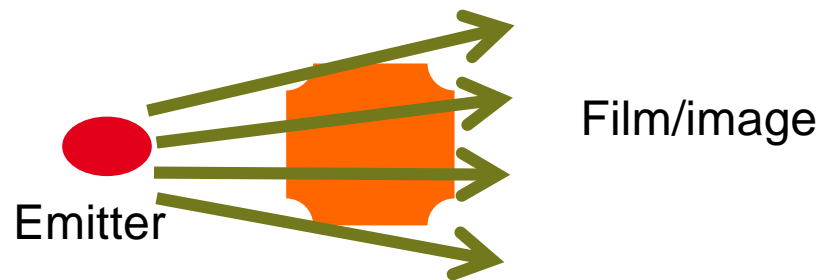
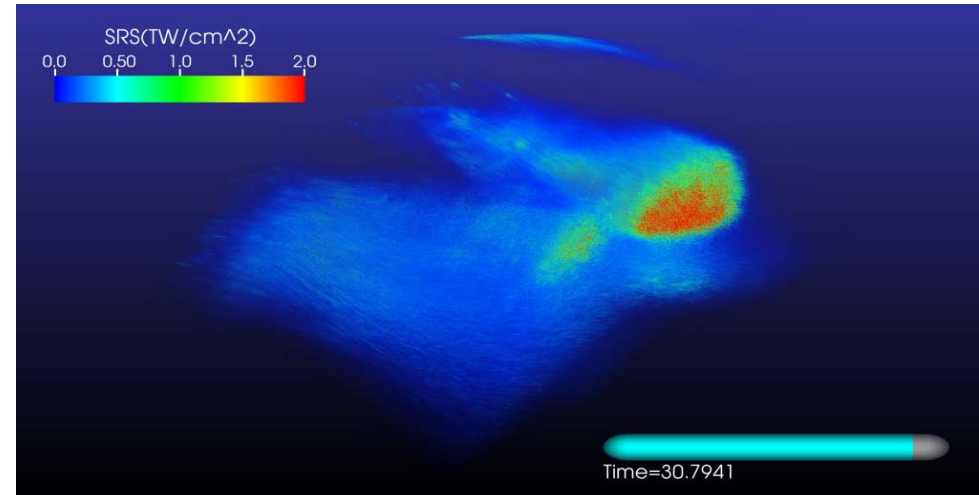
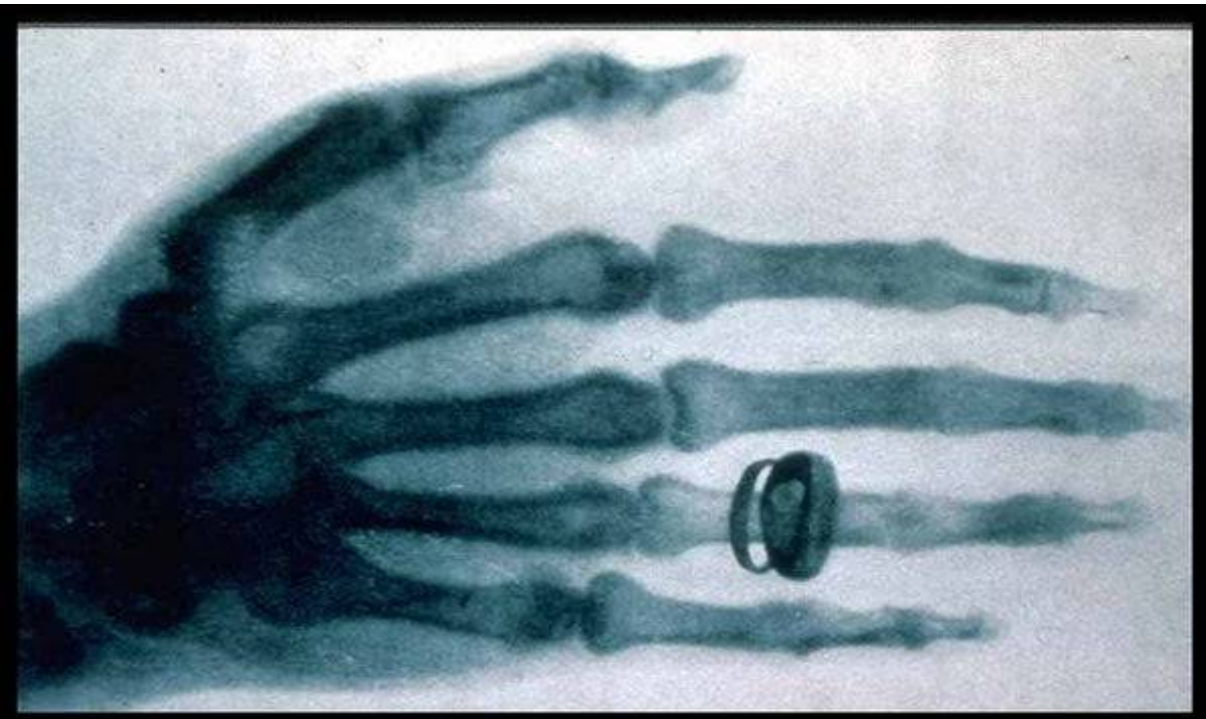
- Maps scalar fields (e.g., density, pressure, temperature) to colors.



Contour / Isosurface



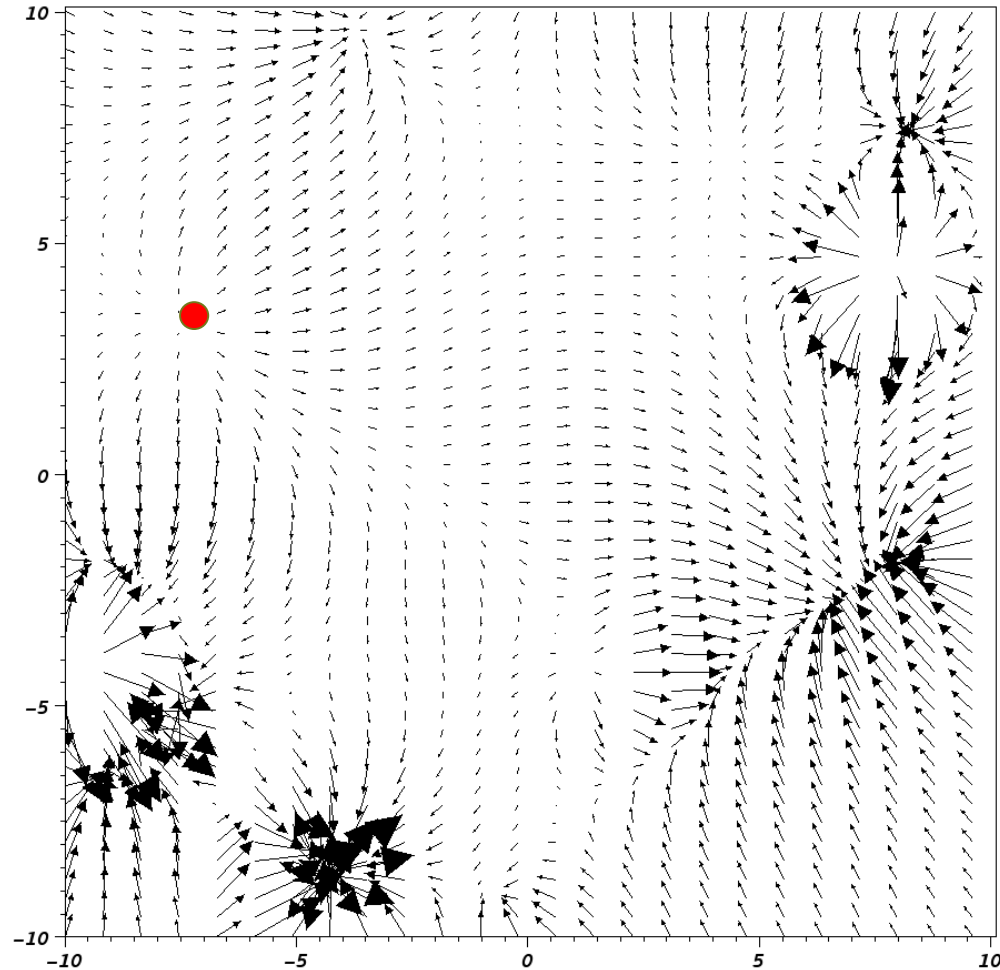
Volume rendering



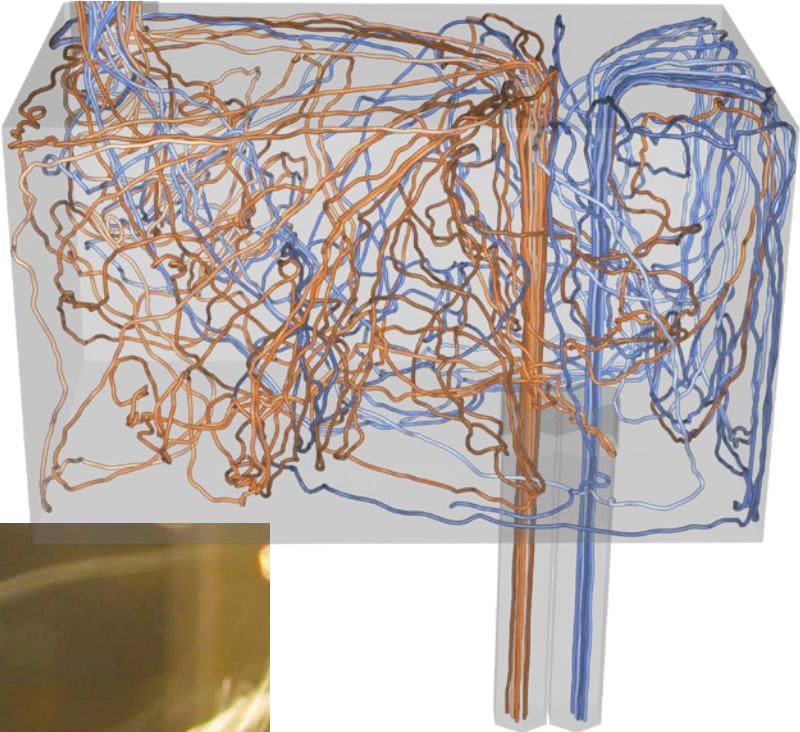
Particle advection: the foundation of flow visualization

- 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



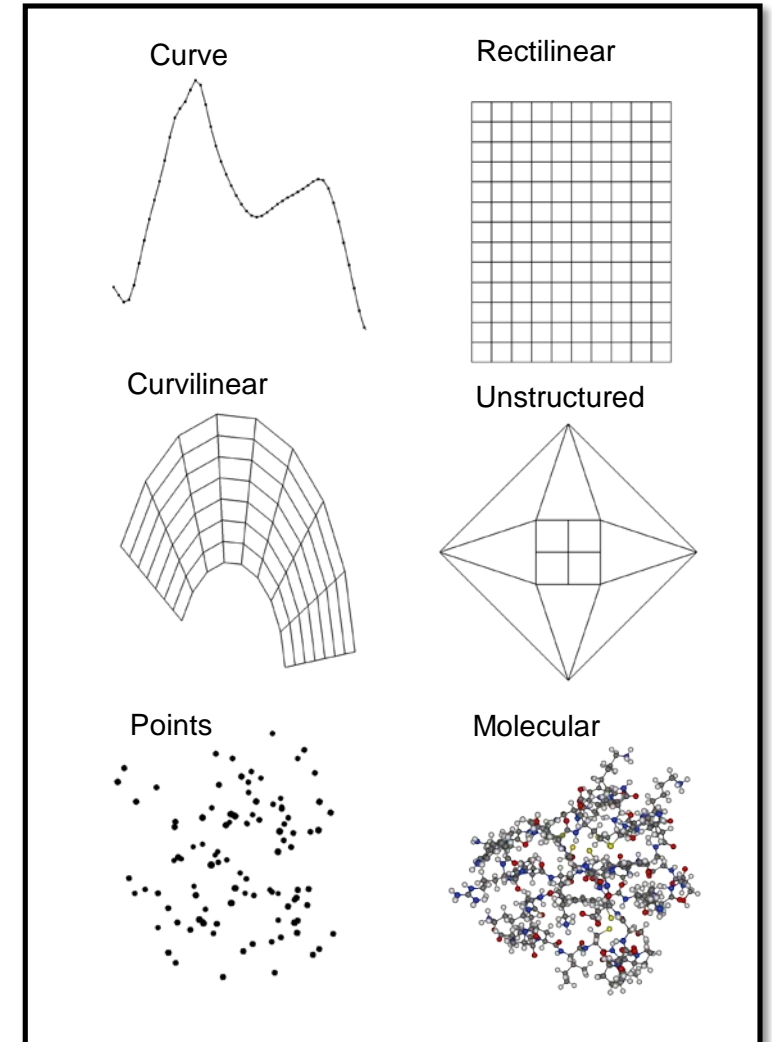
Streamlines



Meshes

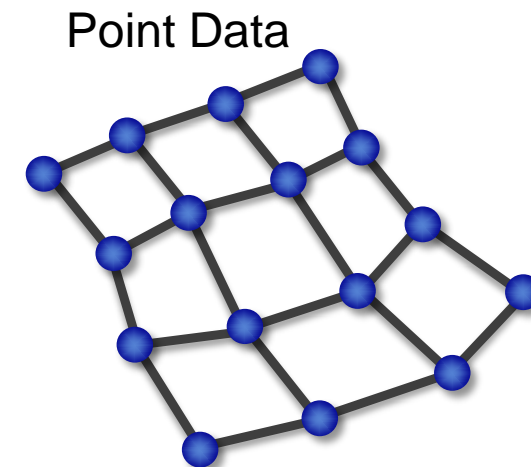
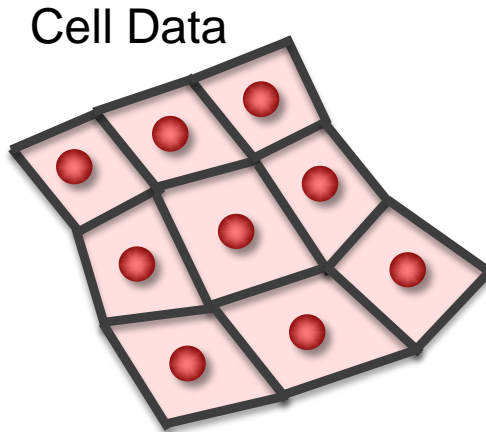
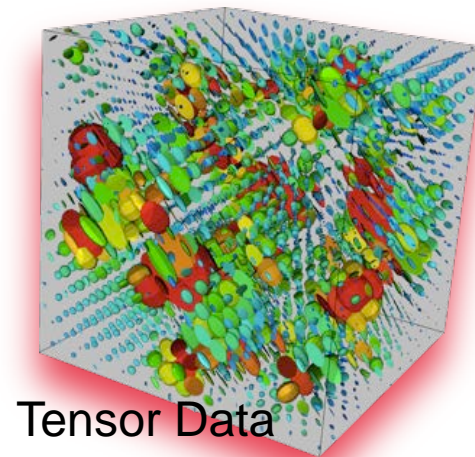
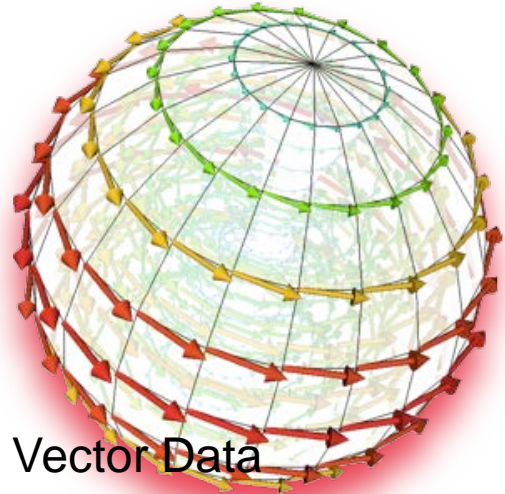
- All data in VisIt lives on a mesh
- Discretizes space into points and cells
 - (1D, 2D, 3D) + time
 - Mesh dimension need not match spatial dimension (*e.g. 2D surface in 3D space*)
- Provides a place for data to be located
- Defines how data is interpolated

Mesh Types



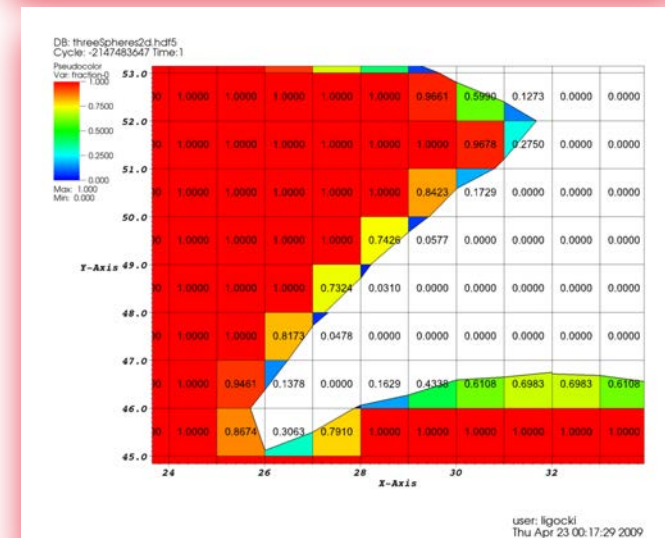
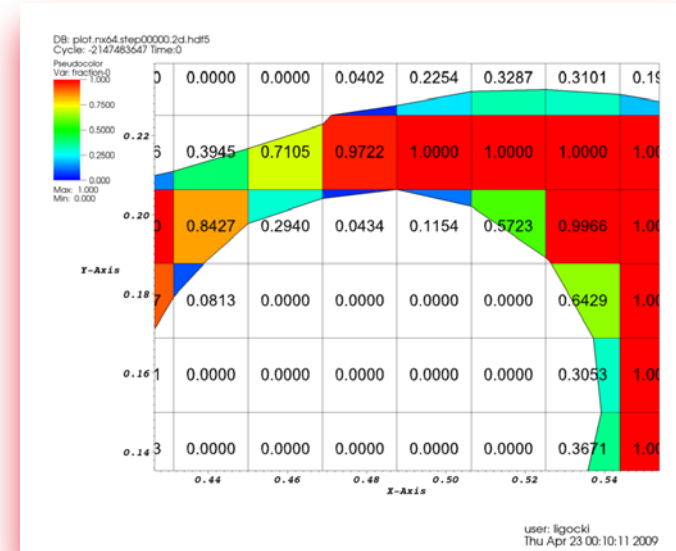
Variables

- 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

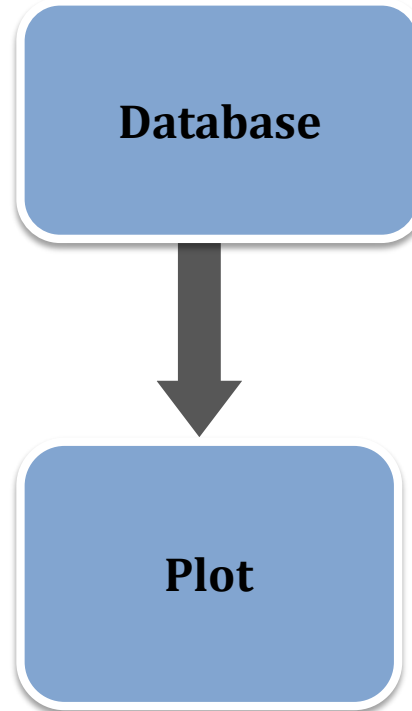
- Similar to materials, describes sub-grid variable composition
 - Example: *Material “Air” is made of species “N₂”, “O₂”, “Ar”, “CO₂”, etc.*
- Used for mass fractions
- Generally used to weight other scalars (e.g. partial pressure)

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

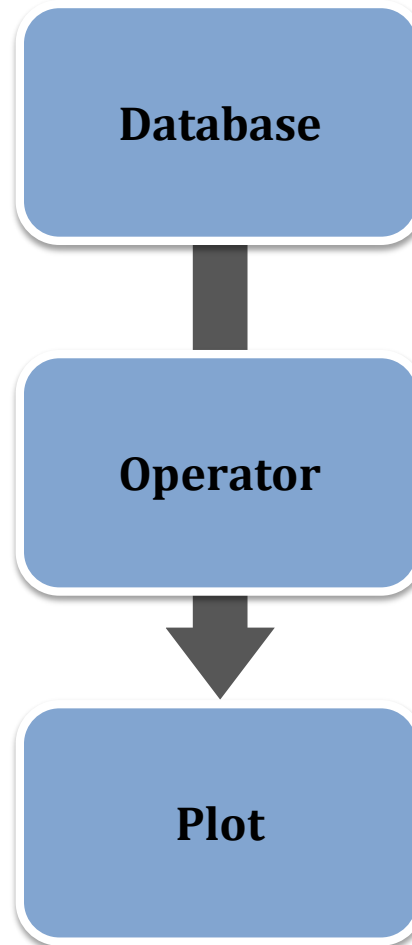


Open a database, which reads from a file (example: open file1.hdf5)

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



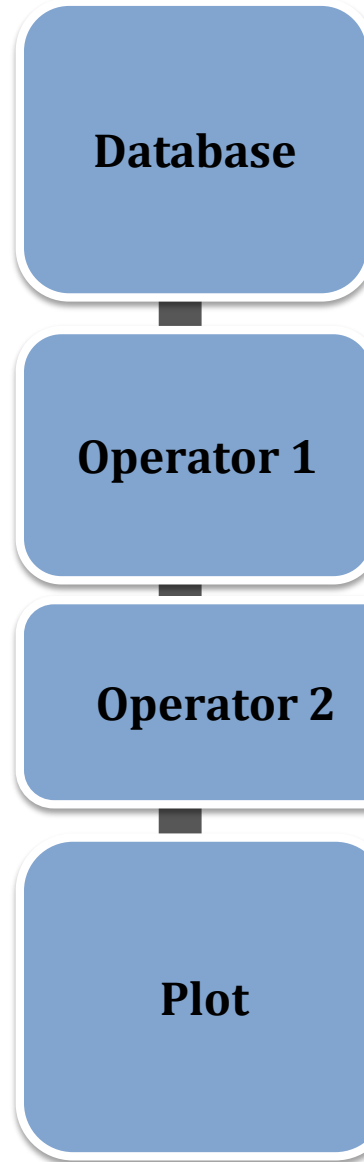
Open a database, which reads from a file (example: open file1.hdf5)

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)

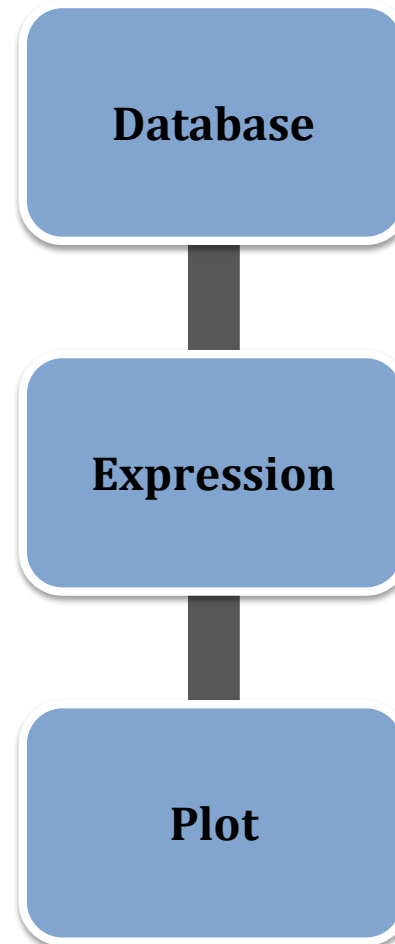
Apply an operator to transform the data (example: Slice operator)

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



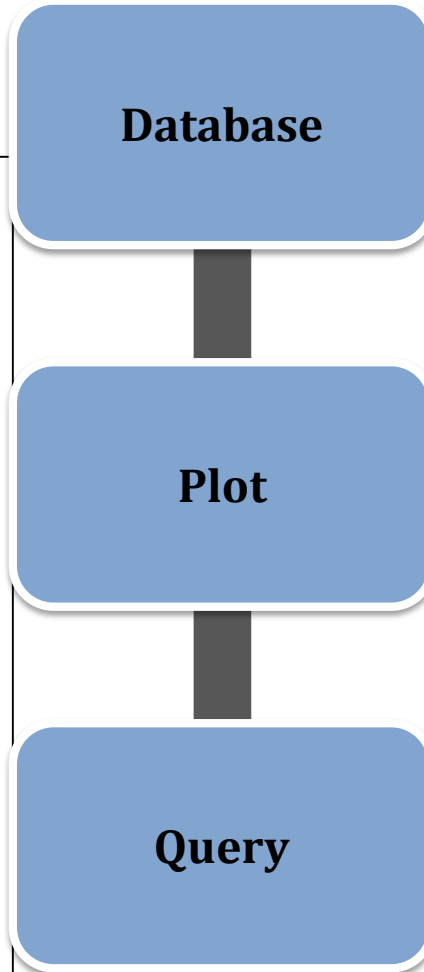
Open a database, which reads from a file (example: open file1.hdf5)

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

Plot the expression variable (example: Pseudocolor plot)

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)

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

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

Data format and I/O libs

Prelude

Data formats

- Interface between simulations and visualization
- Many formats exist. Pick the most appropriate
- High level libraries (HDF5, netCDF, ...)
- Raw data vs. self.described formats
- 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 of the number of processors (tasks)
- Self-documented

Data formats

- Community specific
 - CGNS, CCSM, NEK5000, H5Part, Chombo, ENZO, RAMSES,
- Ad-hoc
 - ~~Make up your own~~
 - Many formats exist. Choose the most appropriate
 - High level libraries (HDF5, netCDF, ...) and usage conventions

I/O strategies

- Serial I/O: each process writes its own output.
 - Make sure each single file can be read independently
 - Have a method to combine them all (without file concatenation)
- Parallel I/O: a collective operation writes a single file, each process placing data at the correct offset.
- Transient data: use one of the above method, for every N timesteps.

Parallel processing, but serial I/O:

**Each process writes its own file independently,
A 64x64x64x4 block of floats**

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

Each file can also be gzipped

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

A very simple raw data (mono-block) file, but still many challenges

Read it in VisIt. Visualize in parallel?

- Find erroneous data. Where? i,j coordinates? rank ? value? neighbors?
- Plot node's value over time?
- Find mean, at current time, a mean-over-time?
- Find first timestep when “condition is true”
- Compare serial output with:
 - OpenMP output ?
 - MPI output ?
 - CUDA output ?
- Compare solution at time T1, with solution at time T2?
- Compare solution on grids of different resolutions?

Raw-binary data

- What is good about it?

Performance! Subsetting!

- What is bad about it?

What's inside? variable name? resolution? Endianness? Precision? Intended use?
Who created it? When? Which version of code? Which compiler? Architecture?

Meta-data versus raw-data

- We'll see two ways to define the meta-data necessary to be able to make sense of the binary data
- BOV format (starting at page 9 of the manual)
- Xdmf format

Brick of Values (BOV) format read by VisIt

VisIt can read raw binary data with the following header file

BOV version: 1.0

serial I/O output file

TIME: 0.01

DATA_FILE: output.80.000.bin

DATA SIZE: 64 64 1

DATA_ENDIAN: LITTLE

DATA FORMAT: DOUBLE

VARIABLE: phi

CENTERING: nodal

BRICK SIZE: 0.5 0.5 1.0

BRICK ORIGIN: 0.0 0.0 0.0

The raw data can also be read by numpy

```
import numpy as np
```

```
phi = np.fromfile("output.bin", dtype=np.float64, count=-1, sep="")
```

What are the dimensions of this array?

```
phi.shape => (16384,)
```

We will need to use it as an array of dimensions (128, 128) later on...

Use independent (serial) I/O

- Create one file per process

Brick of Values (BOV) format read by VisIt

VisIt puts all the serial pieces together with the following header

BOV version: 1.0

serial I/O output files, recombined

TIME: 0.01

DATA_FILE: benchmark.200.%03d.bin

DATA SIZE: 128 128 1

DATA_BRICKLETS: 64 64 1

DATA FORMAT: DOUBLE

DATA_ENDIAN: LITTLE

VARIABLE: phi

BRICK ORIGIN: 0.0 0.0 0.0

BRICK SIZE: 1.0 1.0 1.0

Xdmf format read by VisIt

Xdmf provides the “Data model”, i.e. the intended use of the data

Its size, shape, dimensions, variable names, etc...

Data format refers to the raw data to be manipulated. Information like number type (float, integer, etc.), precision, location, rank, and dimensions completely describe any dataset regardless of its size.

The description of the data is separate from the values themselves.

We refer to the description of the data as **Light** data and the values themselves as **Heavy** data. Light data is small and can be passed between modules easily. Heavy data may be potentially enormous.

Example: a three dimensional array of floating point values may be the X,Y,Z geometry for a grid or calculated vector values. Without a data model, it is impossible to tell the difference.

Xdmf enables a richer description than BOV

- A Domain can have one or more *Grid* elements. Each Grid contains a *Topology*, *Geometry*, and zero or more *Attribute* elements. Topology specifies the connectivity of the grid while Geometry specifies the location of the grid nodes. Attribute elements are used to specify values such as scalars and vectors that are located at the node, edge, face, cell center, or grid center.
- **Example: Structured**
 - 2DSMesh - Curvilinear
 - 2DRectMesh - Axis are perpendicular
 - 2DCoRectMesh - Axis are perpendicular and spacing is constant
 - 3DSMesh
 - 3DRectMesh
 - 3DCoRectMesh

Xdmf format can describe the raw data

```
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>
<Xdmf xmlns:xi="http://www.w3.org/2003/XInclude" Version="2.2">
  <Domain>
    <Grid Name="Mesh" GridType="Uniform">
      <Topology TopologyType="3DCORECTMESH" Dimensions="1 128 128"/>
      <Geometry GeometryType="ORIGIN_DXDYDZ">
        <DataItem Name="Origin" NumberType="Float" Dimensions="3" Format="XML">0. 0. 0.</DataItem>
        <DataItem Name="Spacing" NumberType="Float" Dimensions="3" Format="XML">1. 1. 1.</DataItem>
      </Geometry>
      <Attribute Name="phi" Active="1" AttributeType="Scalar" Center="Node">
        <DataItem Dimensions="1 128 128" NumberType="Float" Precision="8" Format="Binary">output.bin</DataItem>
      </Attribute>
    </Grid>
  </Domain>
</Xdmf>
```

A high-level I/O library HDF5

An HDF5 file is a container for storing a variety of scientific data and is composed of two primary types of objects

- HDF5 group: a grouping structure containing zero or more HDF5 objects, together with supporting metadata
- HDF5 dataset: a multidimensional array of data elements, together with supporting metadata

Any HDF5 group or dataset may have an associated attribute list. An HDF5 attribute is a user-defined HDF5 structure that provides extra information about an HDF5 object.

Working with groups and datasets is similar in many ways to working with directories and files in UNIX. As with UNIX directories and files, an HDF5 object in an HDF5 file is often referred to by its full path name (also called an absolute path name).

- / signifies the root group.
- /foo signifies a member of the root group called foo.
- /foo/zoo signifies a member of the group foo, which in turn is a member of the root group.

The raw data can also be re-written as HDF5

```
import numpy as np
```

```
import h5py
```

```
phi = np.fromfile("output.bin", dtype=np.float64, count=-1, sep="")
```

```
out = h5py.File("output.h5", "w")
```

```
g_id = out.create_group("data")
```

```
g_id.create_dataset("phi", (128, 128), np.double, phi)
```

```
out.close()
```

Use “hdfview”, “h5ls -r”, “h5dump -d data/phi”

The raw data can also be converted as HDF5

We need a Data Description Language (DDL) input file:

```
HDF5 "output.h5" {  
  DATASET "/data/phi" {  
    DATATYPE H5T_IEEE_F64LE  
    DATASPACE SIMPLE { ( 128, 128 ) / ( 128, 128 ) }  
    DATA {  
    }  
  }  
}
```

```
H5import output.bin -c ddl.txt -o output.h5
```

See [online](#) reference

Advantages of HDF5

Self-described (but we're still missing the meaning of the data array)

```
In [11]: input = h5py.File("output.h5", "r")
```

```
In [12]: input.values()
```

```
Out[12]: [<HDF5 group "/data" (1 members)>]
```

```
In [13]: input['data']
```

```
Out[13]: <HDF5 group "/data" (1 members)>
```

```
In [14]: input['data'].values()
```

```
Out[14]: [<HDF5 dataset "phi": shape (128, 128), type "<f8">]
```

```
In [15]: phi = input['data']['phi']
```

```
In [16]: phi.shape
```

```
Out[16]: (128, 128)
```

Xdmf format can also describe the HDF5 data

```
<?xml version="1.0" ?>
```

```
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd" []>
```

```
<Xdmf xmlns:xi="http://www.w3.org/2003/XInclude" Version="2.2">
```

```
<Domain>
```

```
<Grid Name="Mesh" GridType="Uniform">
```

```
<Topology TopologyType="3DCORECTMESH" Dimensions="1 128 128"/>
```

```
<Geometry GeometryType="ORIGIN_DXDYDZ">
```

```
<DataItem Name="Origin" NumberType="Float" Dimensions="3" Format="XML">0. 0. 0.</DataItem>
```

```
<DataItem Name="Spacing" NumberType="Float" Dimensions="3" Format="XML">1. 1. 1.</DataItem>
```

```
</Geometry>
```

```
<Attribute Name="phi" Active="1" AttributeType="Scalar" Center="Node">
```

```
<DataItem Dimensions="1 128 128" NumberType="Float" Precision="8"
```

```
Format="HDF5">output.h5:/data/phi</DataItem>
```

```
</Attribute>
```



```
</Grid>
```

The Xdmf format can describe a time series of datasets

- Reading a time varying Raw file series

Parallel I/O

Data formats and Parallelism

- MPI-IO
 - Raw data parallelism
 - Some can be read by VisIt (BOV format)
- ADIOS
 - Raw data but complexity is hidden
- HDF5, NetCDF
 - content-discovery is possible, but semantic is not given
- SILO
 - Poor man's parallelism (1 file per process + metafile) but strong semantic

SILO

- <https://wci.llnl.gov/simulation/computer-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

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.

Summary 1

- Documenting how data files are generated, and what is the intended purpose of the raw data is of utmost importance.
- Think:
 - long-term
 - data sharing
 - multiple post-processing tools
- Raw binary data is fine. As long as it is augmented with some meta-data headers such as BOV, or Xdmf, or numpy reading commands.

Summary 2

- Higher level libraries such as HDF5 enable parallel I/O but most importantly, provide self-documentation about the nature of the raw data
- Yet, interpretation is application-dependent
- There exist ‘conventions of usage’ of HDF5, or netcdf.
 - Pixie, Chombo,
 - H5Part,
 - “CF”
- HDF5 and Xdmf are often used together.

Summary 3

- There exists many formats used by each communities
 - Molecular science
 - Fluid dynamics
 - Astronomy, ...
- Your first thought should be to see if you can re-use such formats.
- Warning. Just because you have a visualization software which runs in parallel does not guarantee that you can read data in parallel...



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

AMR data handling

Chombo data format with HDF5

- Chombo provides a set of tools for implementing finite difference and finite volume methods for the solution of partial differential equations on block-structured adaptively refined rectangular grids
- It uses HDF5 to store its data solution
- Storage is organized by levels of refinement
- For each level, we find
 - An array of bounding boxes
 - A single array of field values (the concatenation of all field values for all patches)
 - An array of offsets for each patch and each variable
- Some global attributes

Chombo data format with HDF5

DB: output.hdf5
Time: 0.12345

Mesh
Var: Mesh

Pseudocolor
Var: density

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

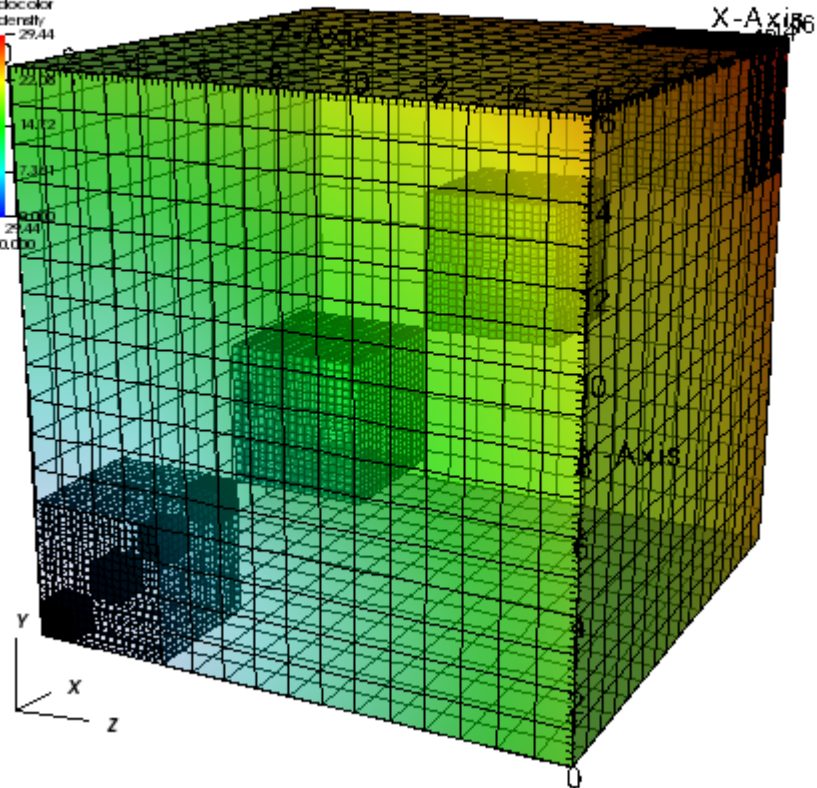
Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000

Max: 29.44
Min: 0.000



DB: output.hdf5
Time: 0.12345

Mesh
Var: Mesh

Pseudocolor
Var: density

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

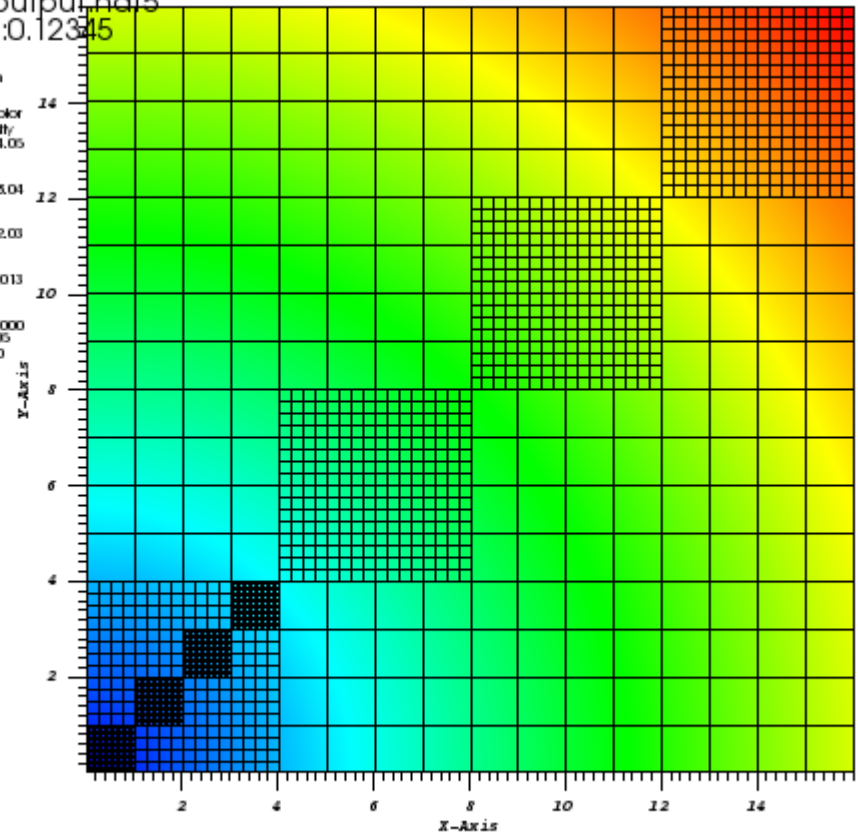
Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000

Max: 24.05
Min: 0.000



Chombo data format with HDF5

- Any code could actually output data using Chombo's convention
- This ensures compatibility with VisIt
- Alternatively, one can write a VisIt-specific reader plugin (for example A-MAZE)
 - It requires an understanding of the internal data structures of VisIt.

With AMR data storage, several opportunities for efficient processing

- AMR Dual Grid and Stitch Cells
- Multi-res Control
- Subsetting

- VisIt's execution is demand-driven, which means it will only pull in the data needed to execute a particular pipeline. If you subset BEFORE clicking "Draw", it will only read the selected data.

- Astro data is [often] very big. You absolutely must understand the two-pass execution mode of the VisIt pipeline to scale your visualization.



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
 - => exercises, or questions from you in the second session

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)

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)`

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:

$$\begin{aligned} &(\text{conn_cmfe}(\langle[-1]id:varname\rangle, \text{meshname}) + \\ &\text{conn_cmfe}(\langle[-2]id:varname\rangle, \text{meshname}) + \\ &\text{conn_cmfe}(\langle[-3]id:varname\rangle, \text{meshname})) / 3. \end{aligned}$$

`[-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 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 a set of moving particles:

- which particles become accelerated,
- which locations exhibit high velocities during an extended timeframe,
- which particles reach a local maximum energy, or which particles change their state

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 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.

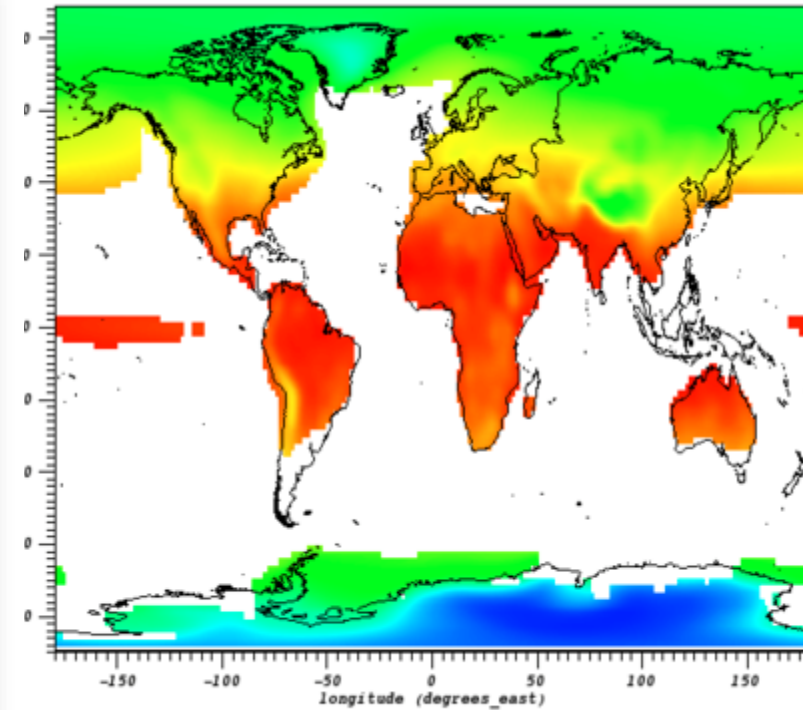
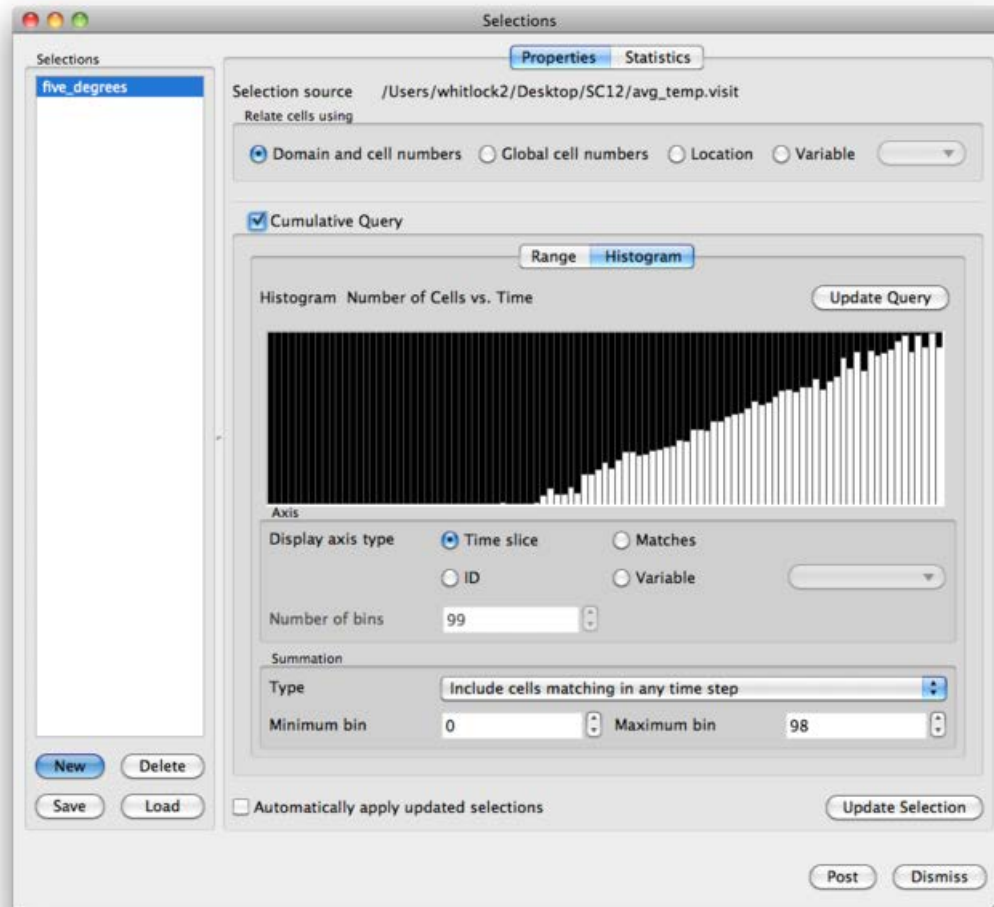
Question:

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



Cumulative query example in climatology

- Detailed explanation on the [wiki](#)



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

In-situ visualization

In-situ visualization

- Motivations
- *In-situ* visualization
 - In-situ processing strategies
 - VisIt's libsim library
 - Enable visualization in a running simulation
 - Source code instrumentation

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**
- **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 but from memory:
***in-situ* visualization**

in-situ (parallel) visualization

Instrument parallel simulations to:

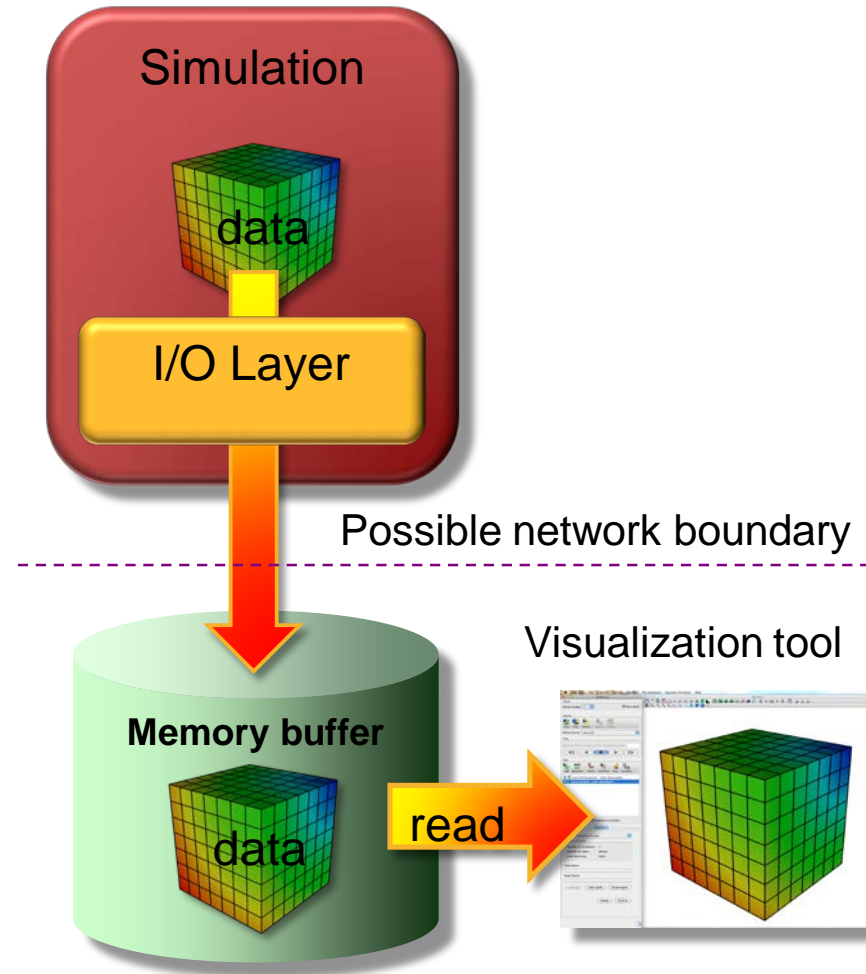
- Eliminate I/O to and from disks
- Use all grid data with or without ghost-cells
- Have access to all time steps, all variables
- Use the available parallel compute nodes
- Maximize features and capabilities
- Minimize code modifications to simulations
- Minimize impact to simulation codes
- Allow users to start an in-situ session *on demand* instead of deciding before running a simulation
 - Debugging
 - Computational steering

in-situ Processing Strategies

In Situ Strategy	Description	Negative Aspects
Loosely coupled 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
Tightly coupled 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)
Hybrid	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

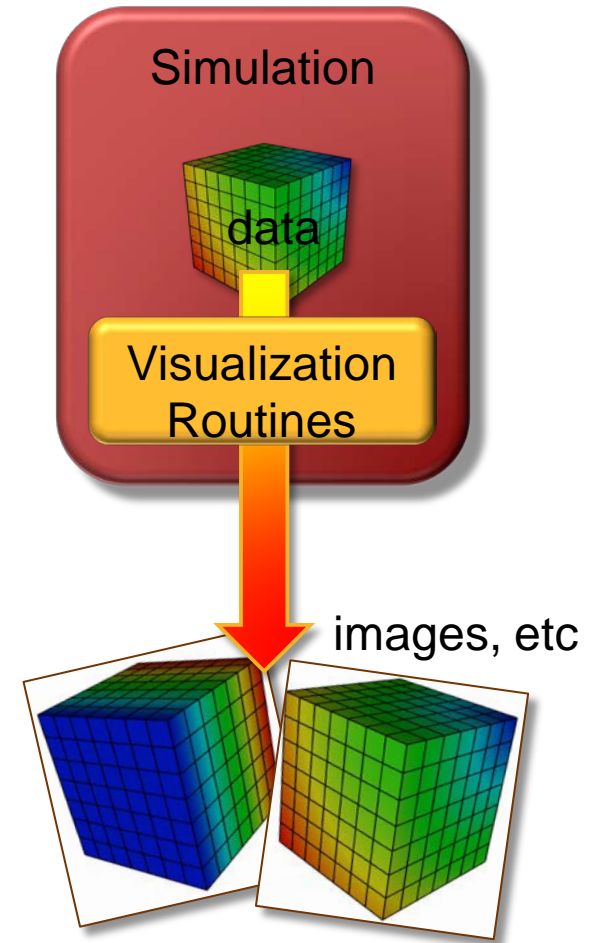
Loosely Coupled in-situ Processing

- 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



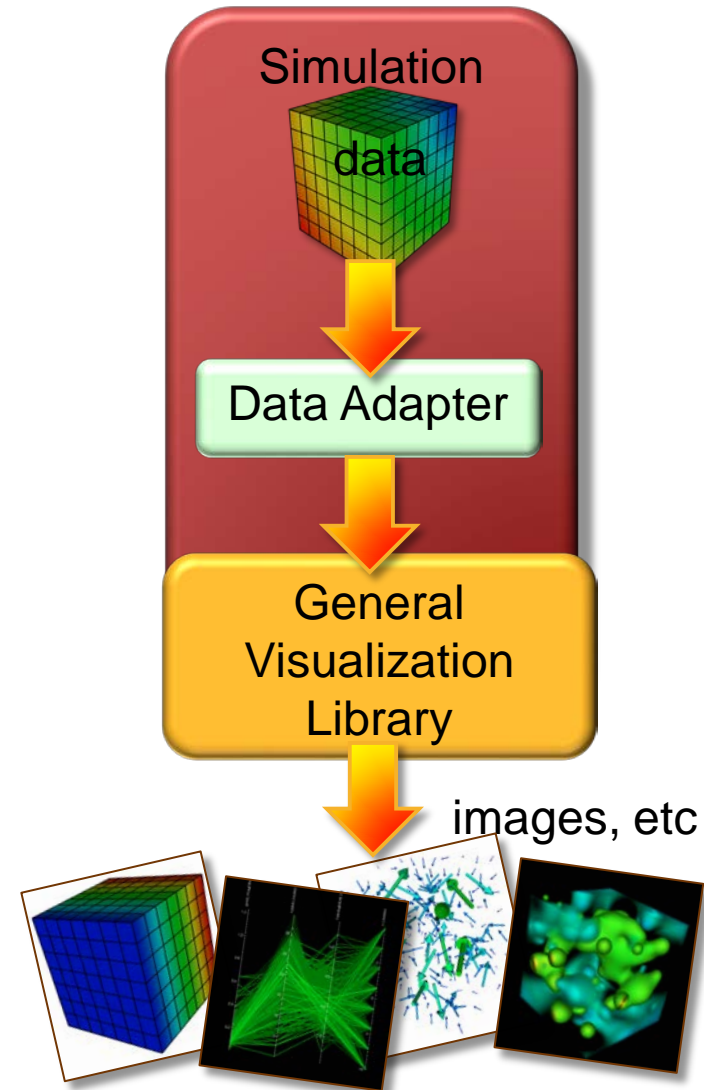
Tightly Coupled *Custom* in-situ Processing

- 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*



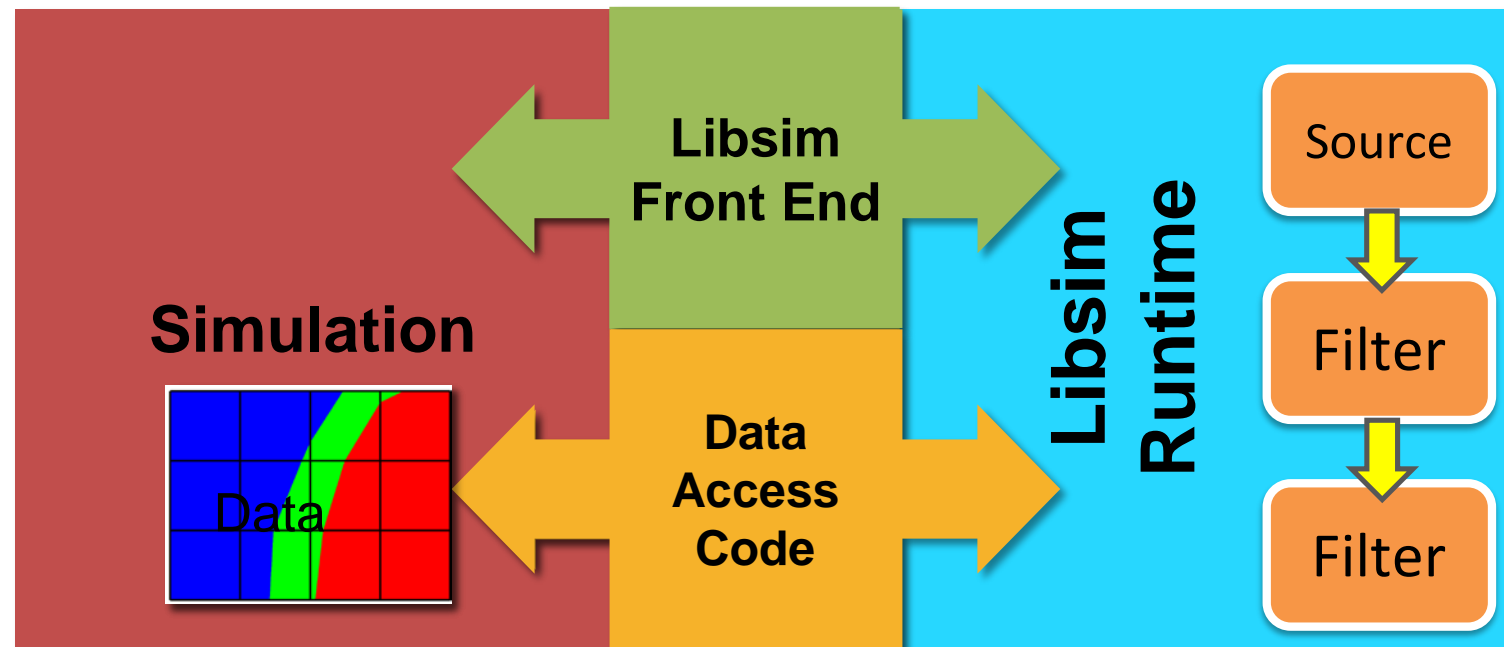
Tightly Coupled *General* in-situ Processing

- 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*



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 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.

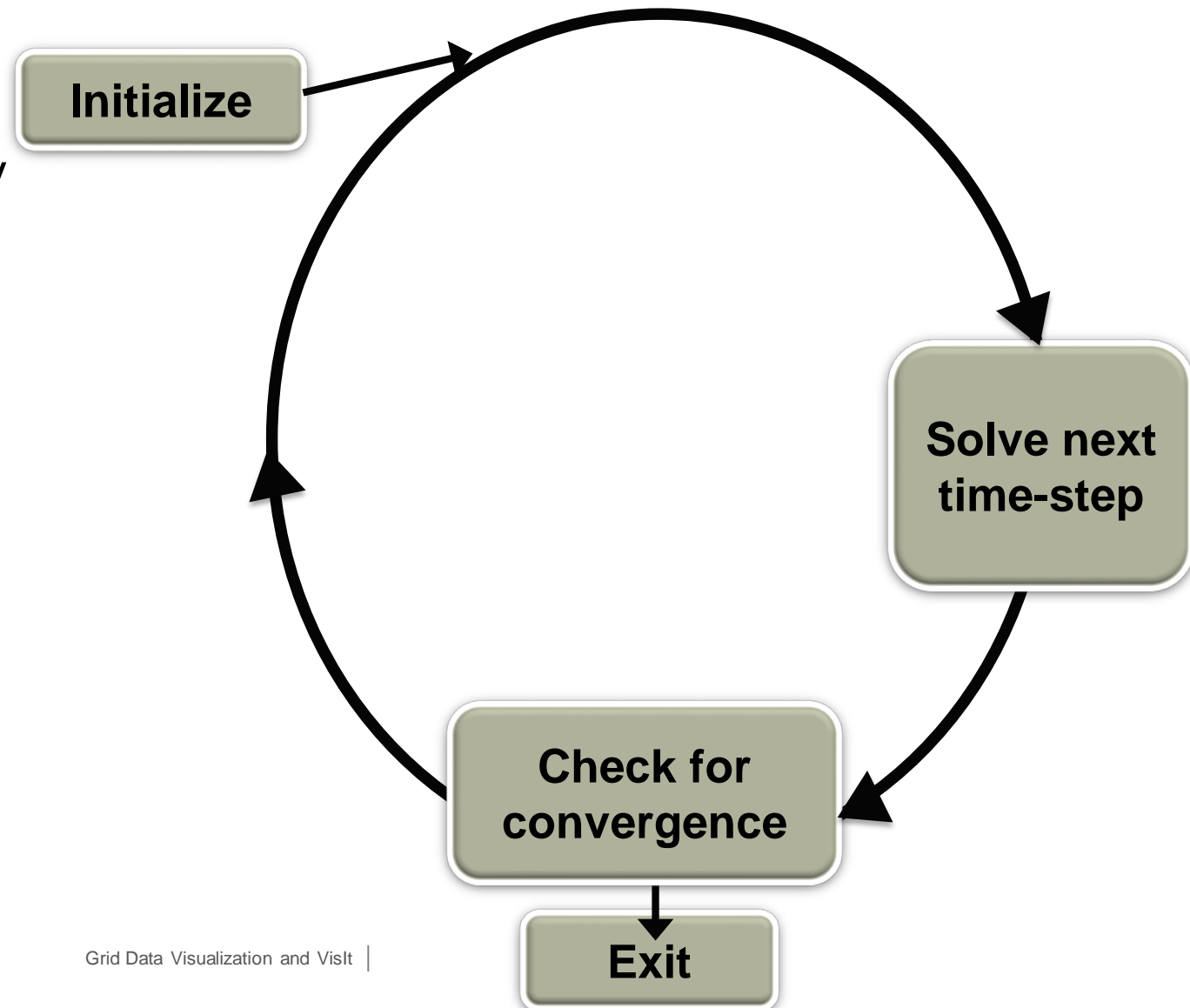
Add control functions that let VisIt steer the simulation.

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

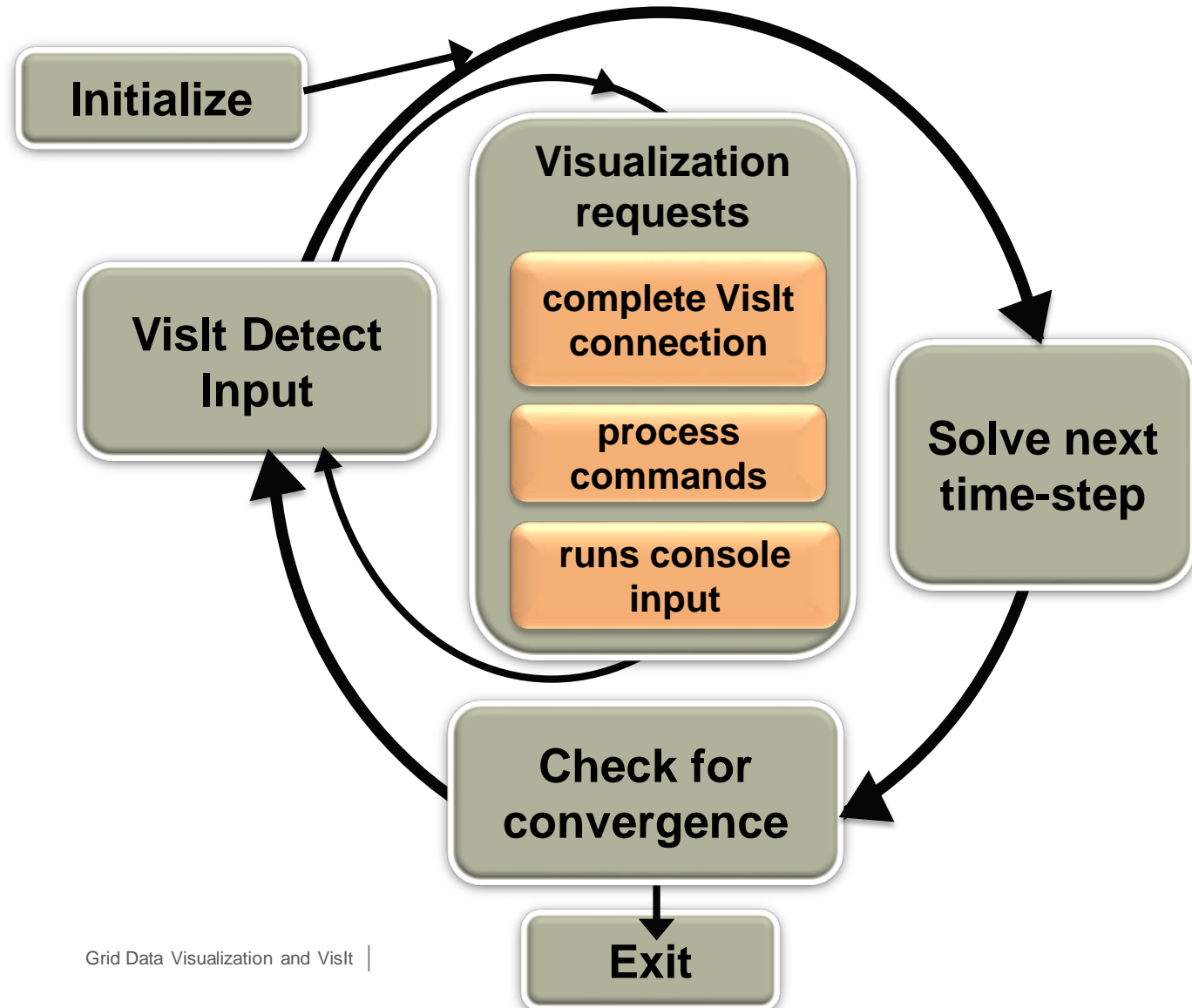


Visit in-the-loop

- Libsim opens a socket and writes out connection parameters

VisitDetectInput checks for:

- Connection request
- Visit commands
- Console input



Summary

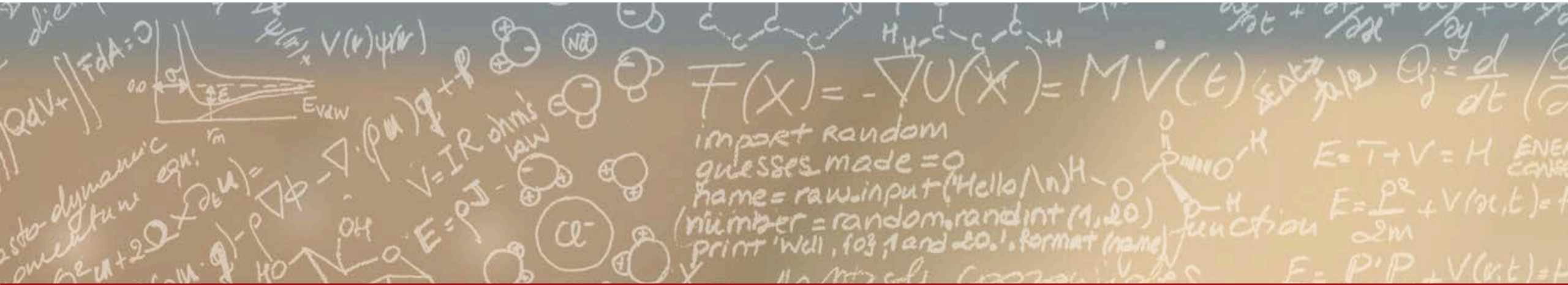
- VisIt has a very rich set of database plugins
 - Try to re-use a supported file format
- The python interface is the way to automatize visualization and analysis tasks
 - Move to batch mode
 - Move to parallel execution
- Expressions and queries enable us to go much beyond simply “3D graphics”
- Most of the above is also available in the in-situ interface.



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention.