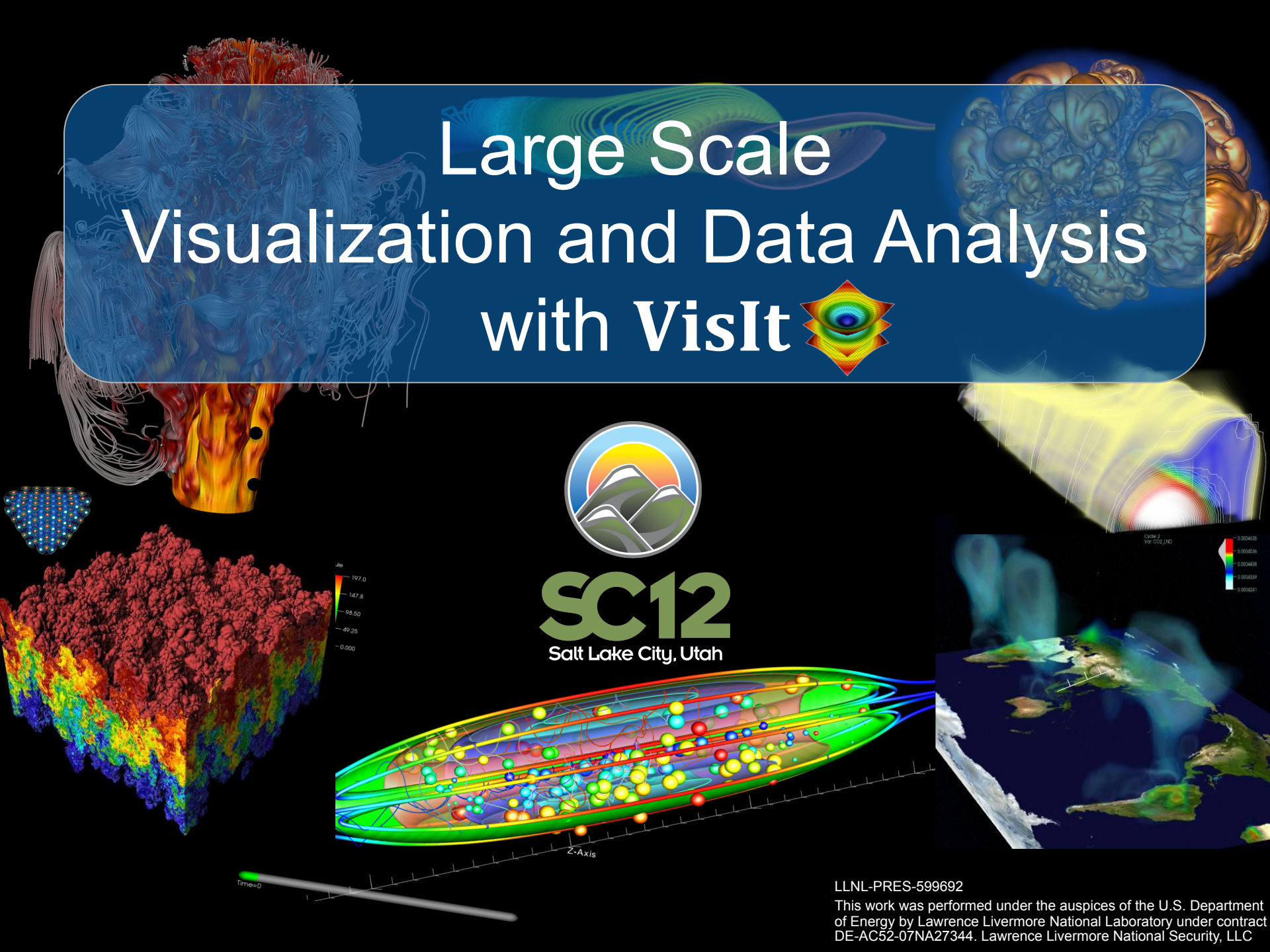
Large Scale Visualization and Data Analysis with VisIt



LLNL-PRES-599692

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Tutorial Speakers

Cyrus Harrison



Jean Favre



CSCS

Hank Childs



UCDAVIS
UNIVERSITY OF CALIFORNIA

Dave Pugmire

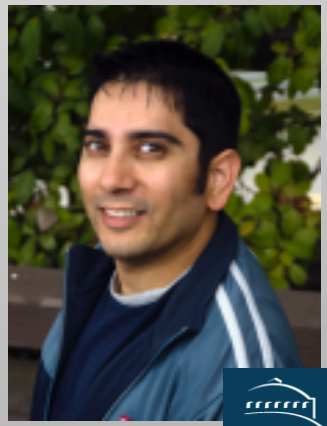


OAK RIDGE
National Laboratory

Brad Whitlock



Hari Krishnan

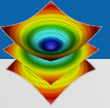


Afternoon Session (Advanced) [1/2]

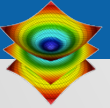
- **Client-server (10 min)**
- **Parallelism in VisIt** (10 min)
- **Advanced Topics:**
  - **Named Selections** (10 min)
  - **Streamline Computation** (20 min)
  - **Material Interface Reconstruction** (5 min)
- ***<Break>*** *(30 min)*

Afternoon Session (Advanced) [2/2]

- **Customizing VisIt:**
  - **Writing a file format reader** (30 min)
  - **Writing an analysis operator** (15 min)
  - **Writing a Python query** (10 min)
  - **Creating custom Python applications** (15 min)
- **In situ processing** (35 min)
- **Wrap-up and Questions** (20 min)

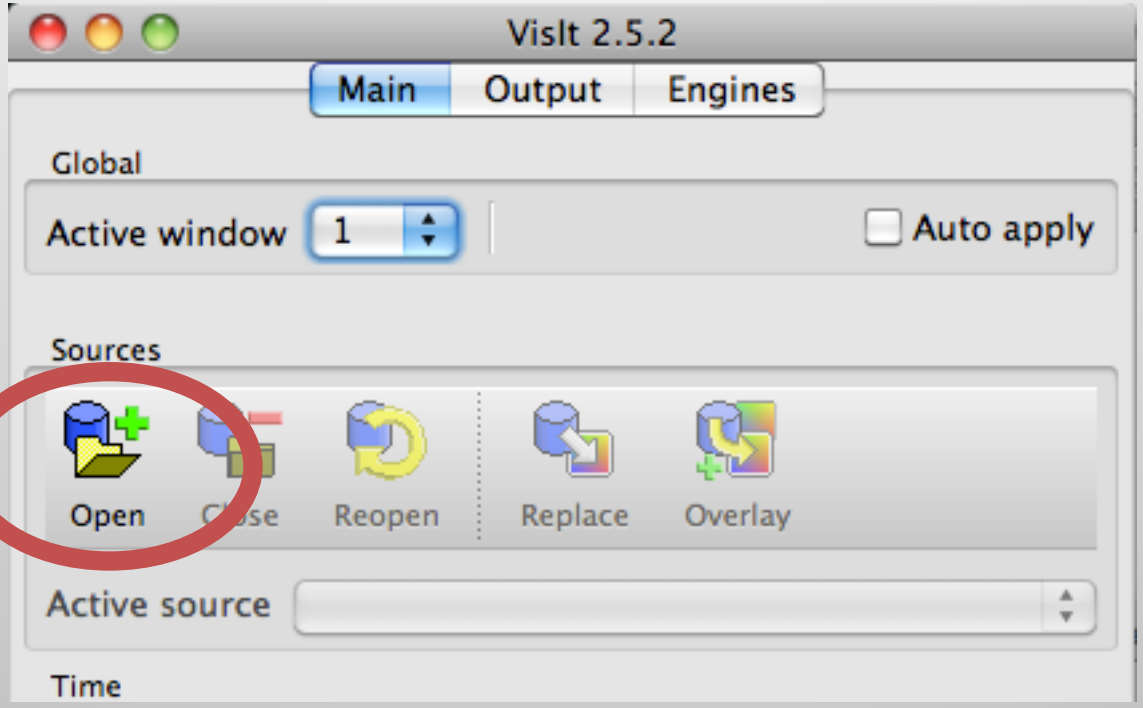


Afternoon Session

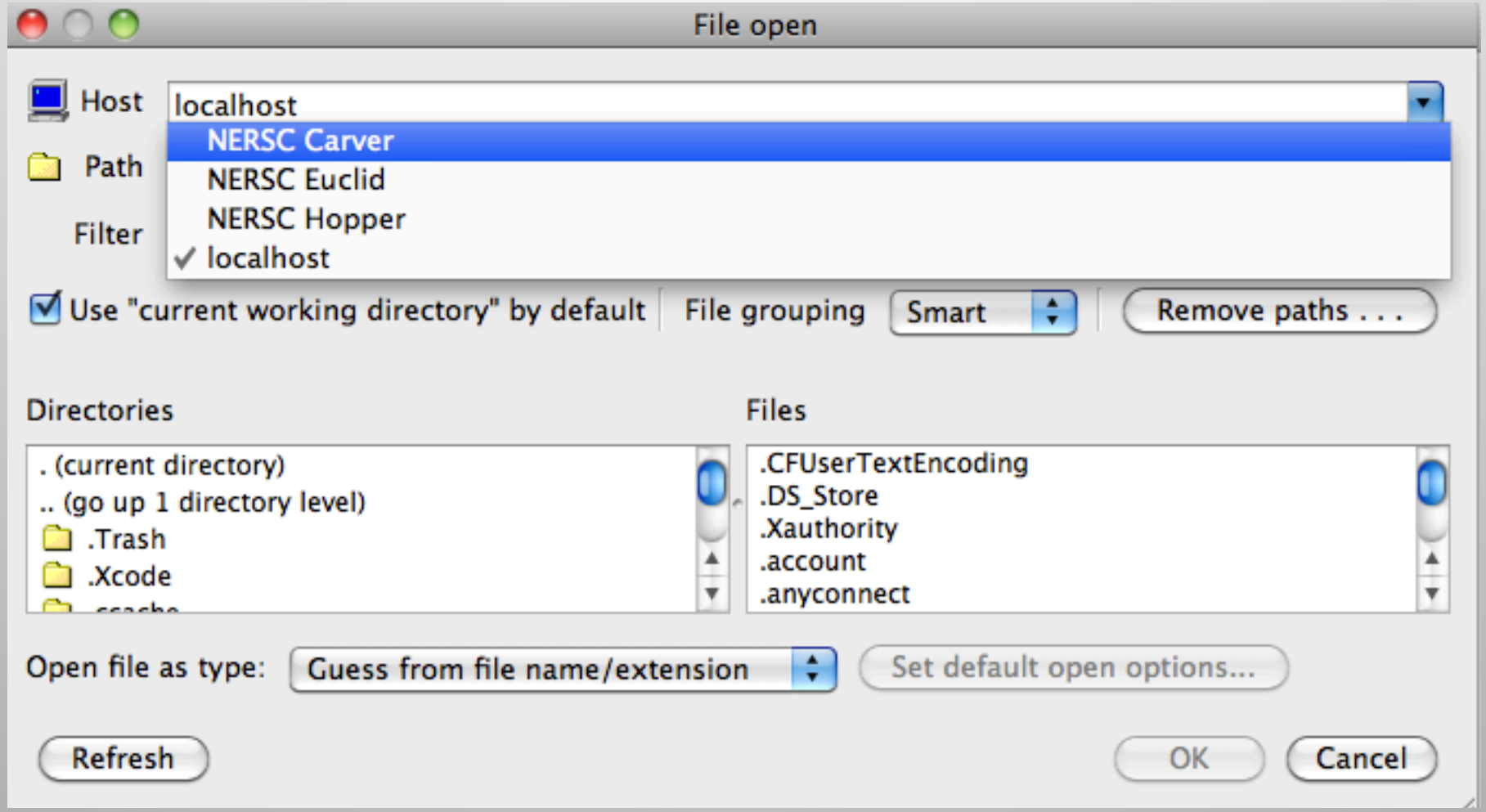


Using VisIt in client-server mode

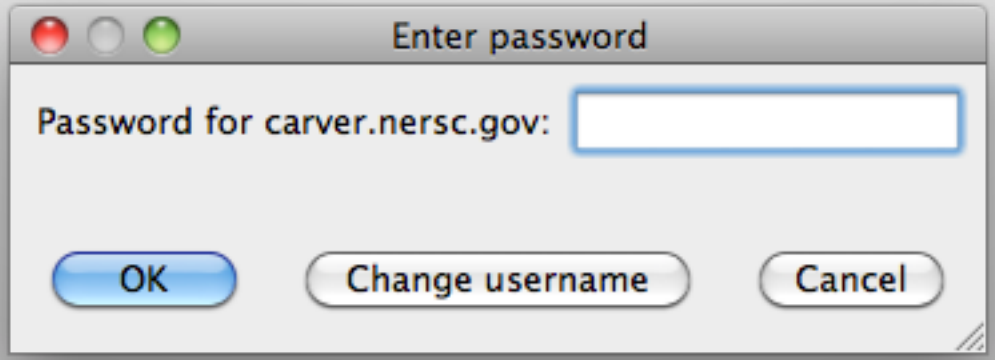
Client-Server Example: Connecting to NERSC's Carver Cluster



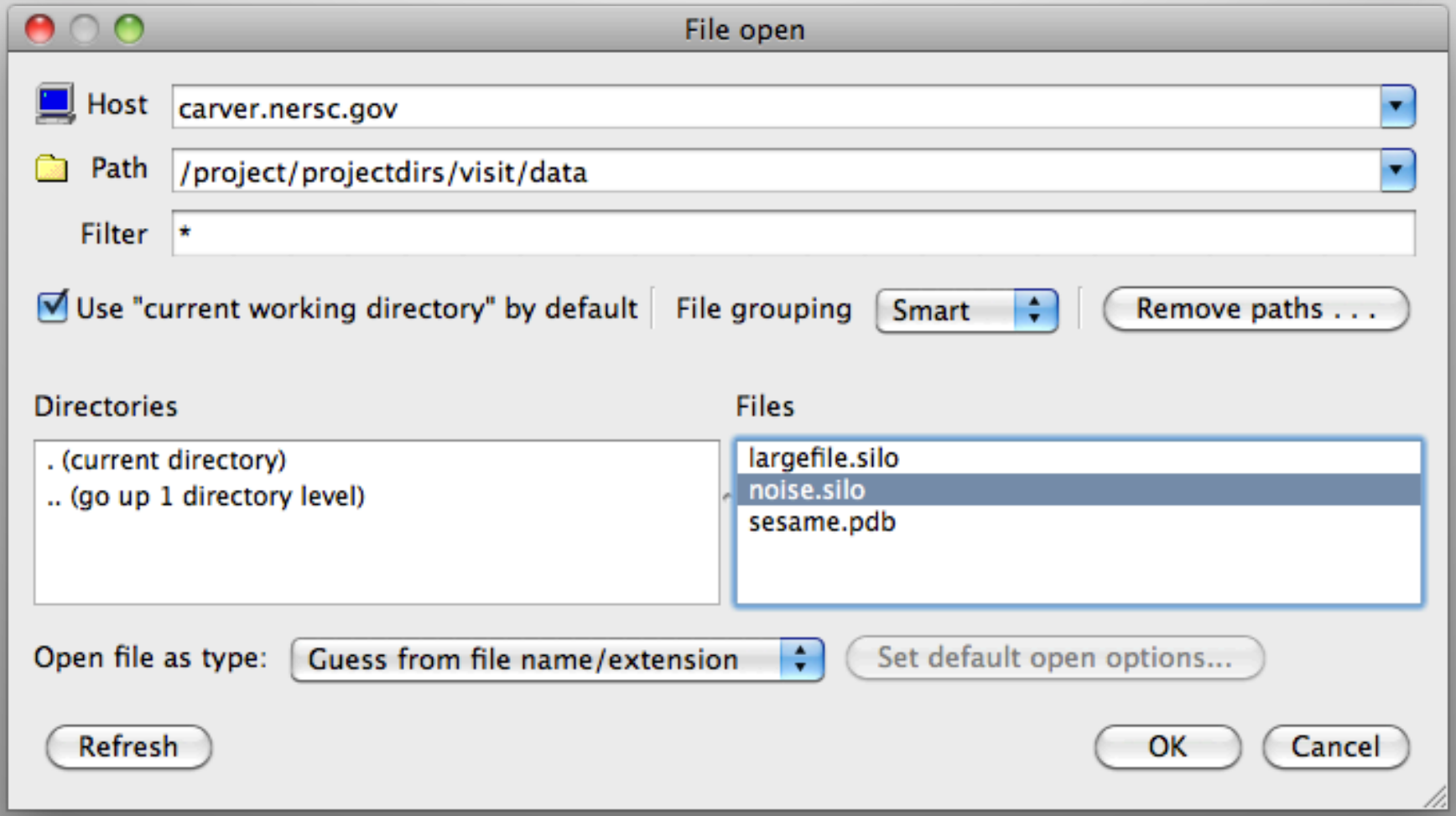
Client-Server Example: Connecting to NERSC's Carver Cluster



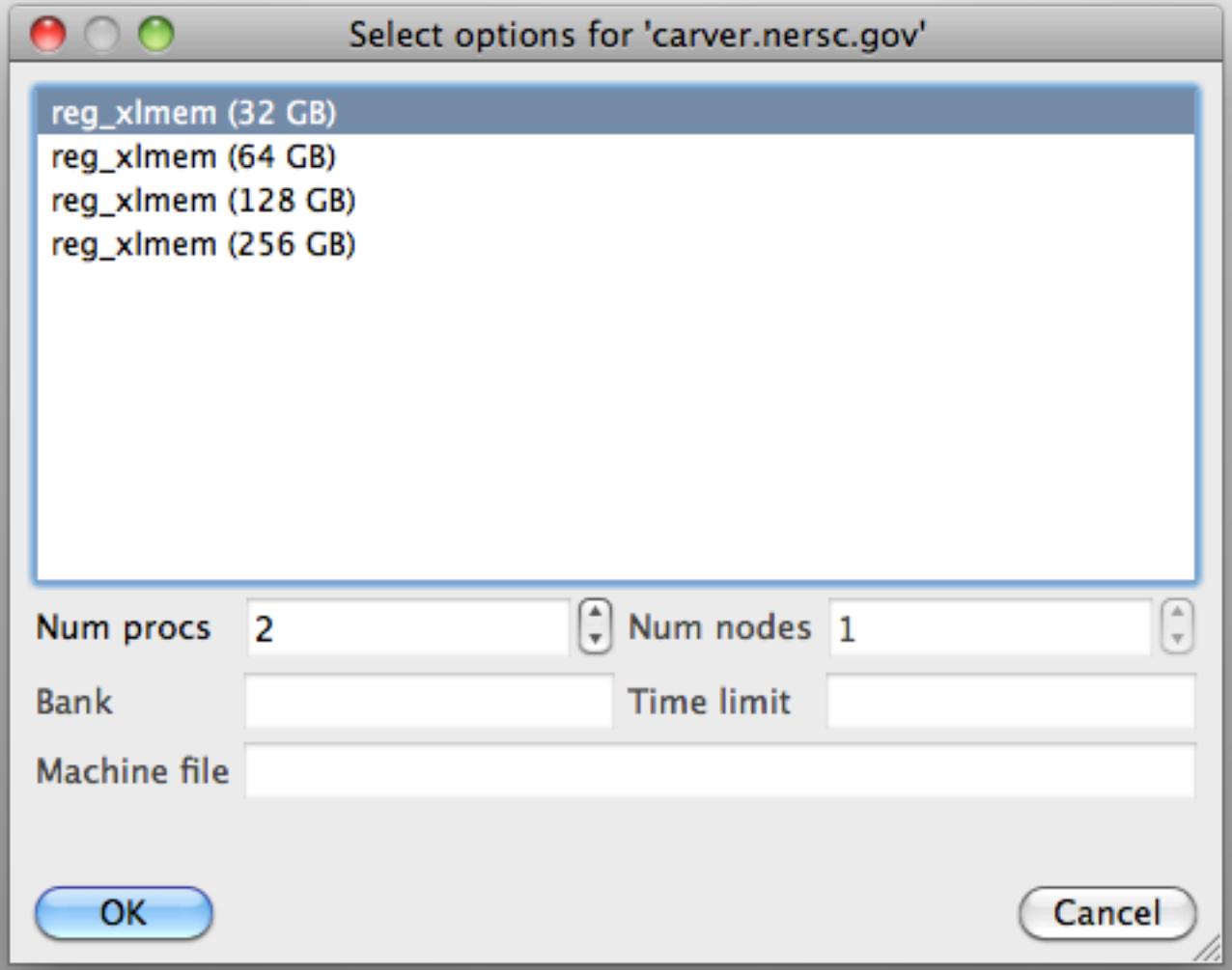
Client-Server Example: Connecting to NERSC's Carver Cluster

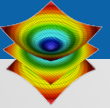


Client-Server Example: Connecting to NERSC's Carver Cluster



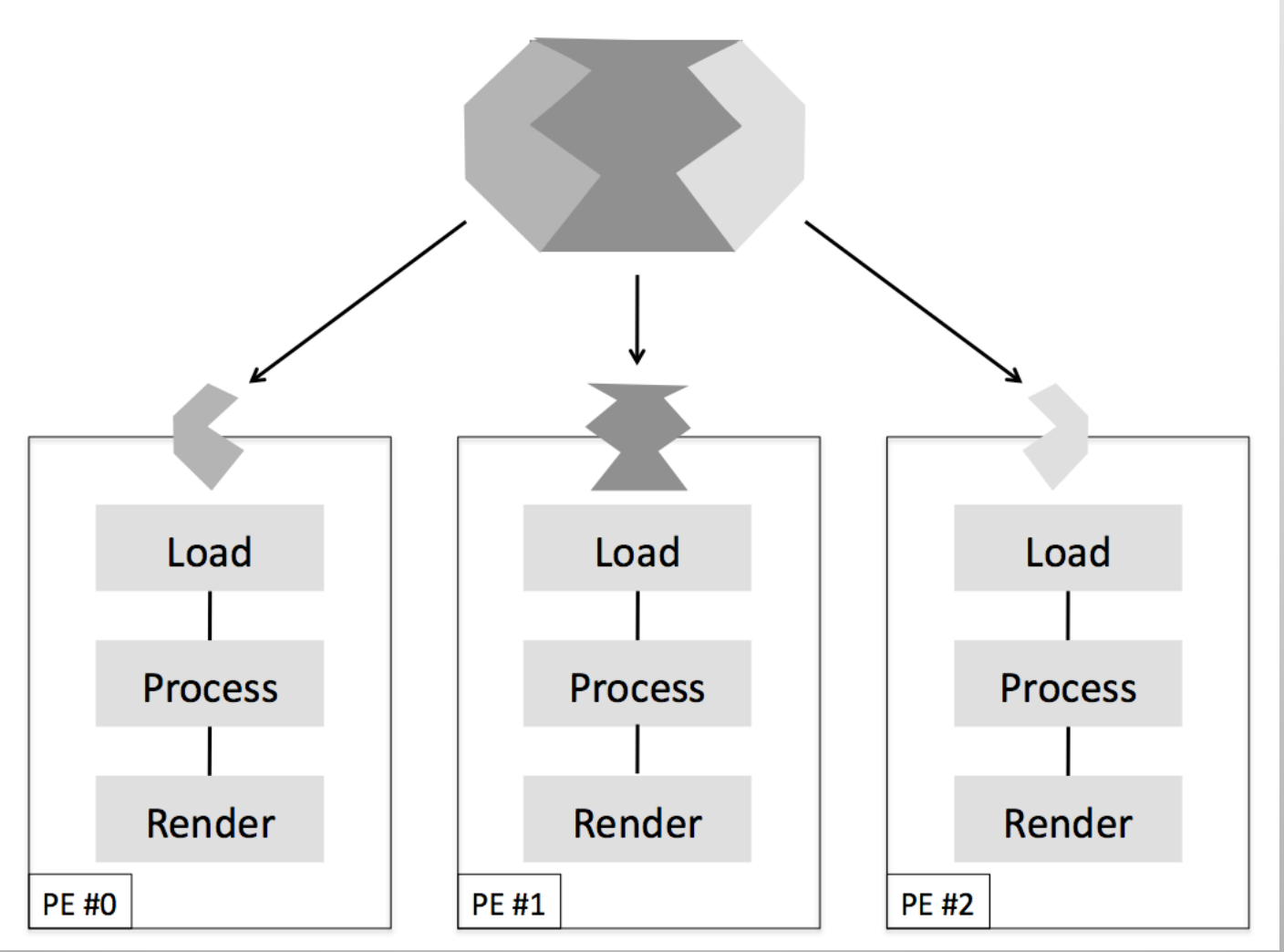
Client-Server Example: Connecting to NERSC's Carver Cluster





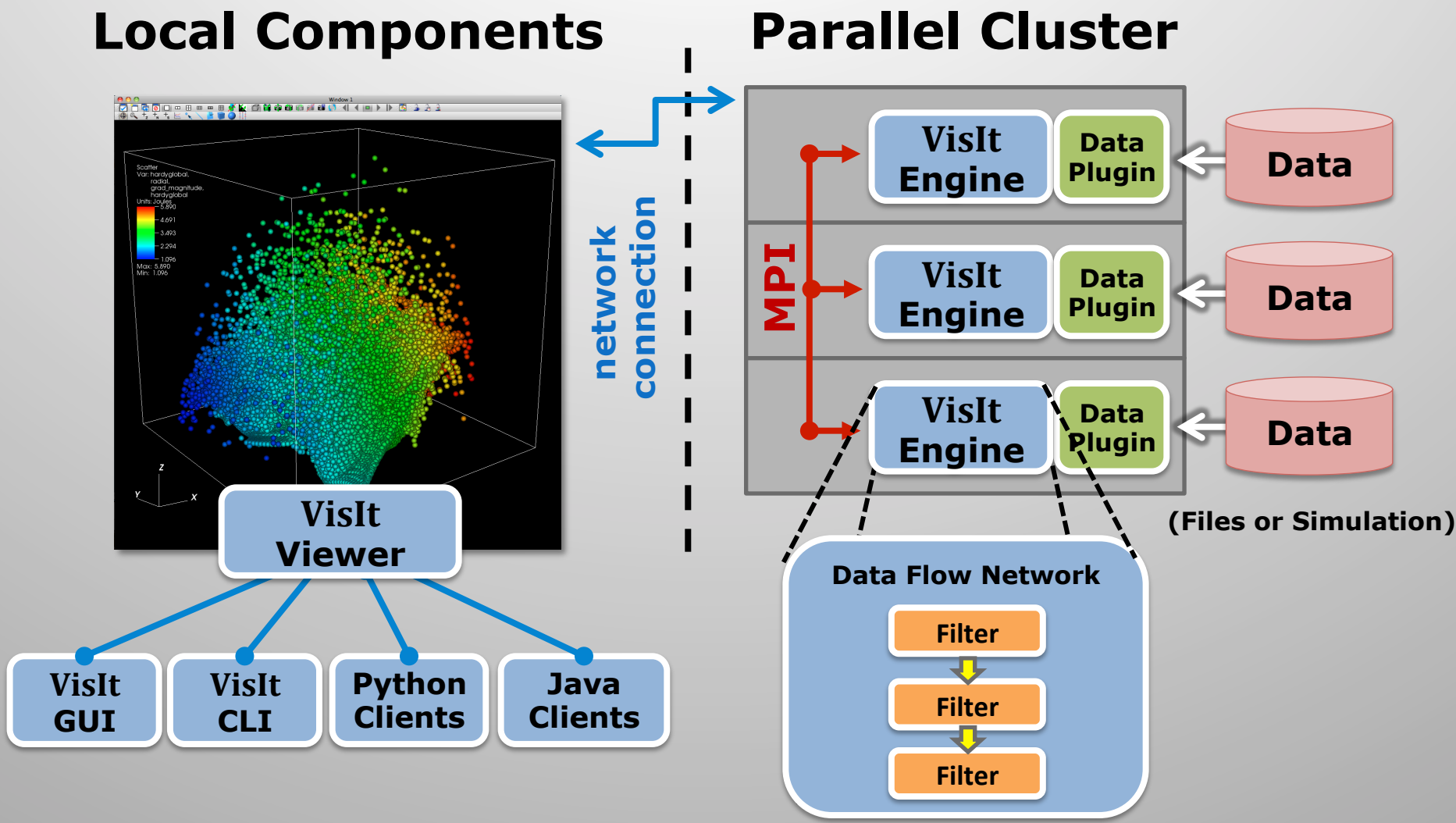
Data parallelism in VisIt

Data parallelism is the dominant paradigm for processing.



PE = Processing Element

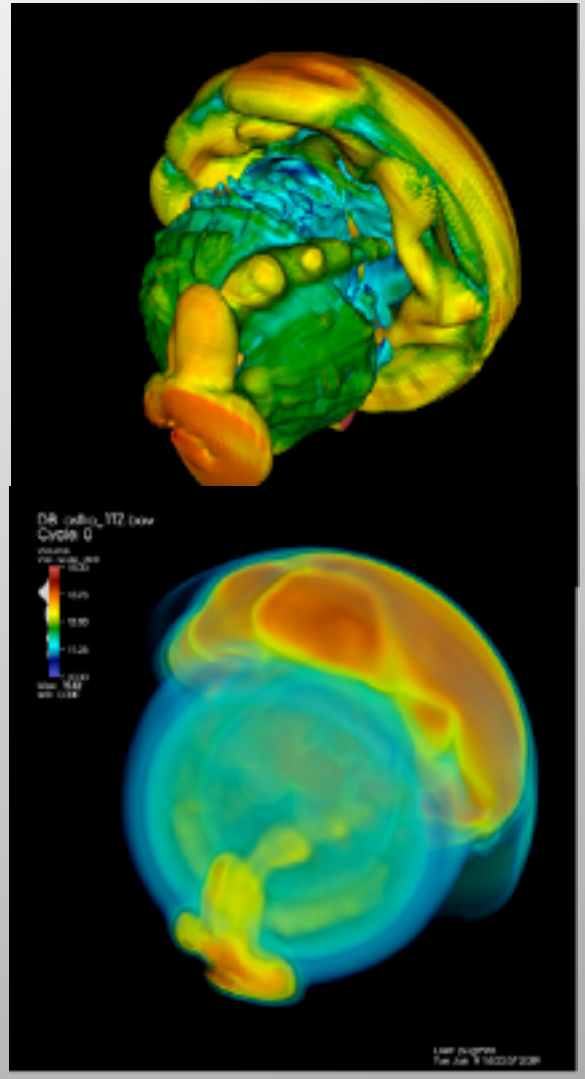
VisIt employs a parallelized client-server architecture.



VisIt has demonstrated good performance at unprecedented scale.

Weak scaling study: ~62.5M cells/CPU

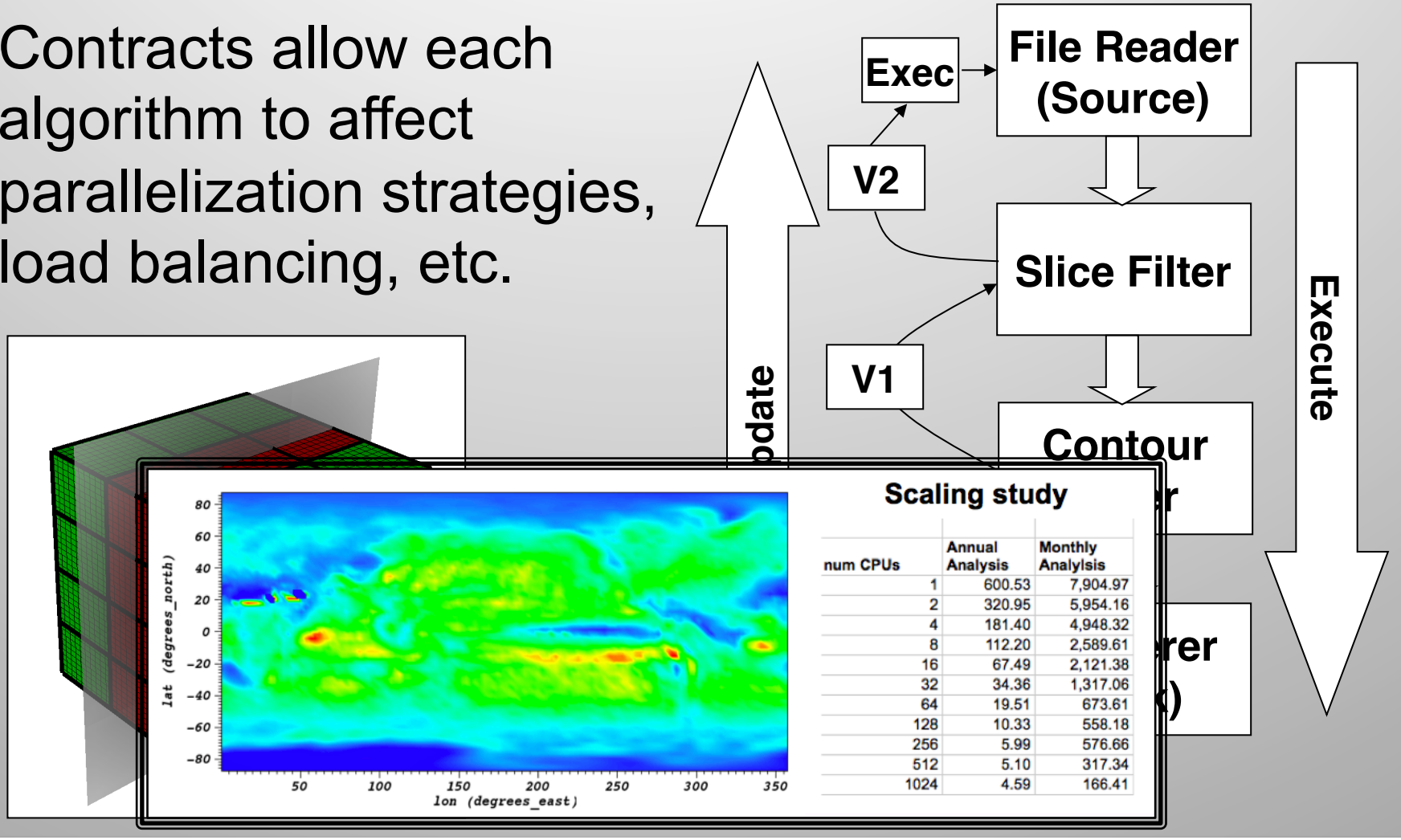
Machine	Architecture	Problem Size	# of Cores
Dawn	BG/P	$15{,}871^3$ (4 T cells)	64K
Franklin	Cray XT4	$12{,}596^3$ (2 T cells)	32K
JaguarPF	Cray XT5	$12{,}596^3$ (2 T cells)	32K
Juno	X86_64	$10{,}000^3$ (1 T cells)	16K
Franklin	Cray XT4	$10{,}000^3$ (1 T cells)	16K
Ranger	Sun	$10{,}000^3$ (1 T cells)	16K
Purple	IBM P5	$8{,}000^3$ (0.5 T cells)	8K



Two trillion hexahedron data set,
rendered in VisIt
by Dave Pugmire
on Oak Ridge Jaguar machine

VisIt's unique contract-based system enables techniques beyond simple data parallelism.

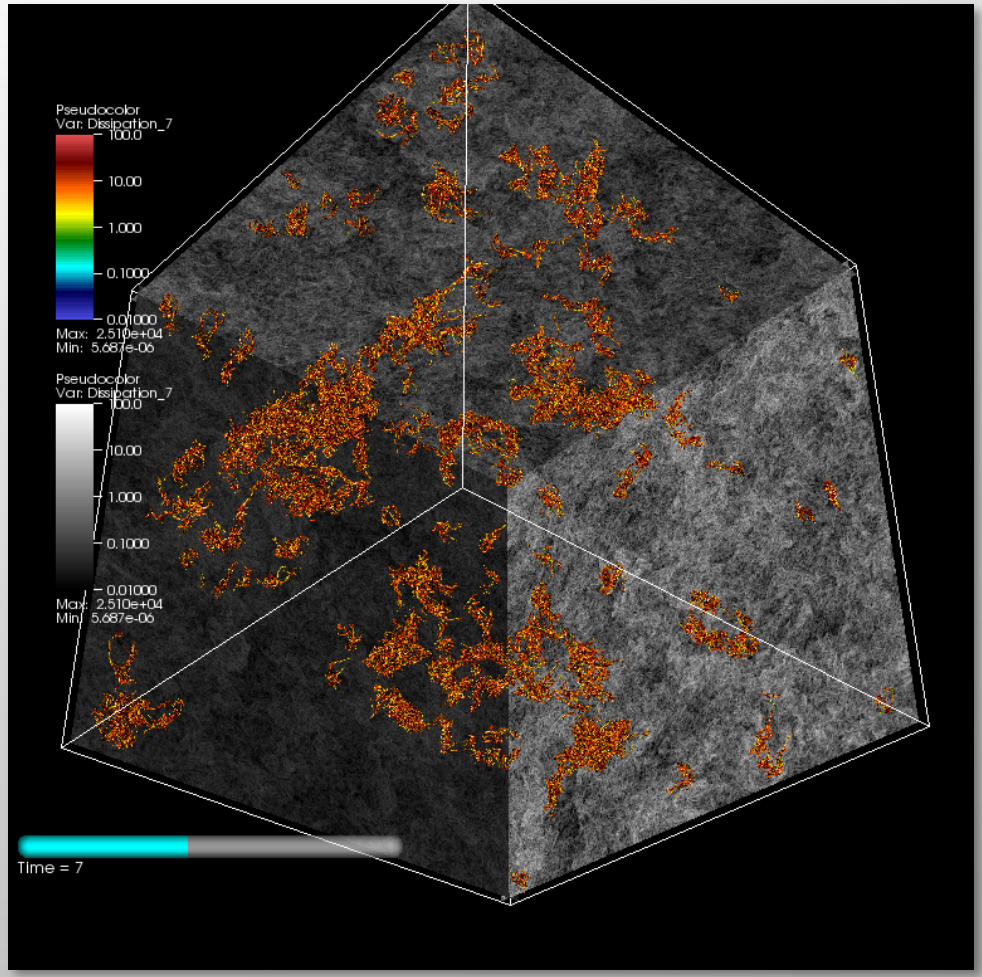
- Contracts allow each algorithm to affect parallelization strategies, load balancing, etc.



num CPUs	Annual Analysis	Monthly Analysis
1	600.53	7,904.97
2	320.95	5,954.16
4	181.40	4,948.32
8	112.20	2,589.61
16	67.49	2,121.38
32	34.36	1,317.06
64	19.51	673.61
128	10.33	558.18
256	5.99	576.66
512	5.10	317.34
1024	4.59	166.41

Visualizing and Analyzing Large-Scale Turbulent Flow

- Detect, track, classify, and visualize features in large-scale turbulent flow.
- Analysis effort by Kelly Gaither (TACC), Hank Childs (LBNL), & Cyrus Harrison (LLNL).
- Stresses two algorithms that are difficult in a distributed memory parallel setting:
  1. Can we identify connected components?
  2. Can we characterize their shape?

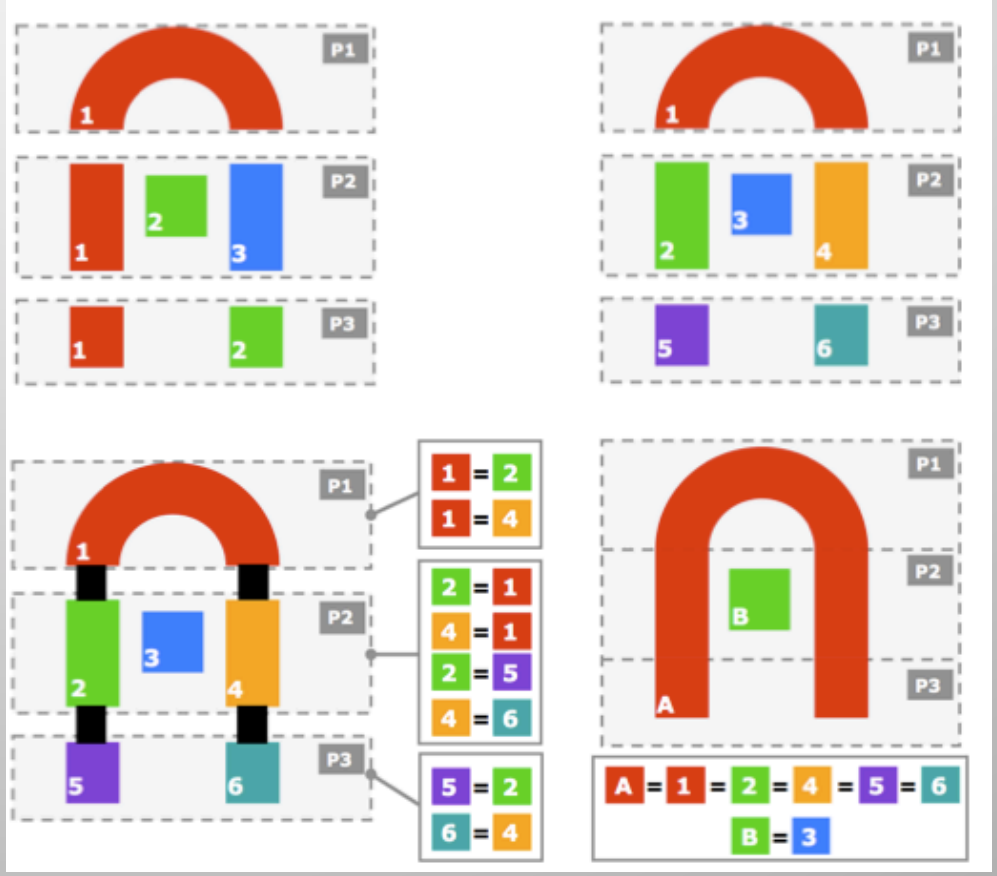


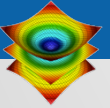
Data courtesy of P.K. Yeung and Diego Donzis

VisIt calculated connected components on a 4K^3 turbulence data in parallel using TACC's Longhorn machine. Two million components were initially identified and then the map expression was used to select only the components that had total volume greater than 15.

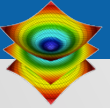
Identifying connected components in parallel is difficult.

- Hard to do efficiently
- Tremendous bookkeeping problem.
- 4 stage algorithm that finds local connectivity and then merges globally.

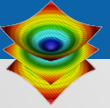




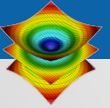
Advanced Topics:
Named Selections



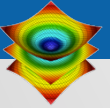
Advanced Topics:
Streamline Computation



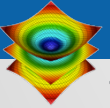
Advanced Topics:
Material Interface Reconstruction



<Break>



Customizing VisIt:
Writing a file format reader



VisIt's Data Model

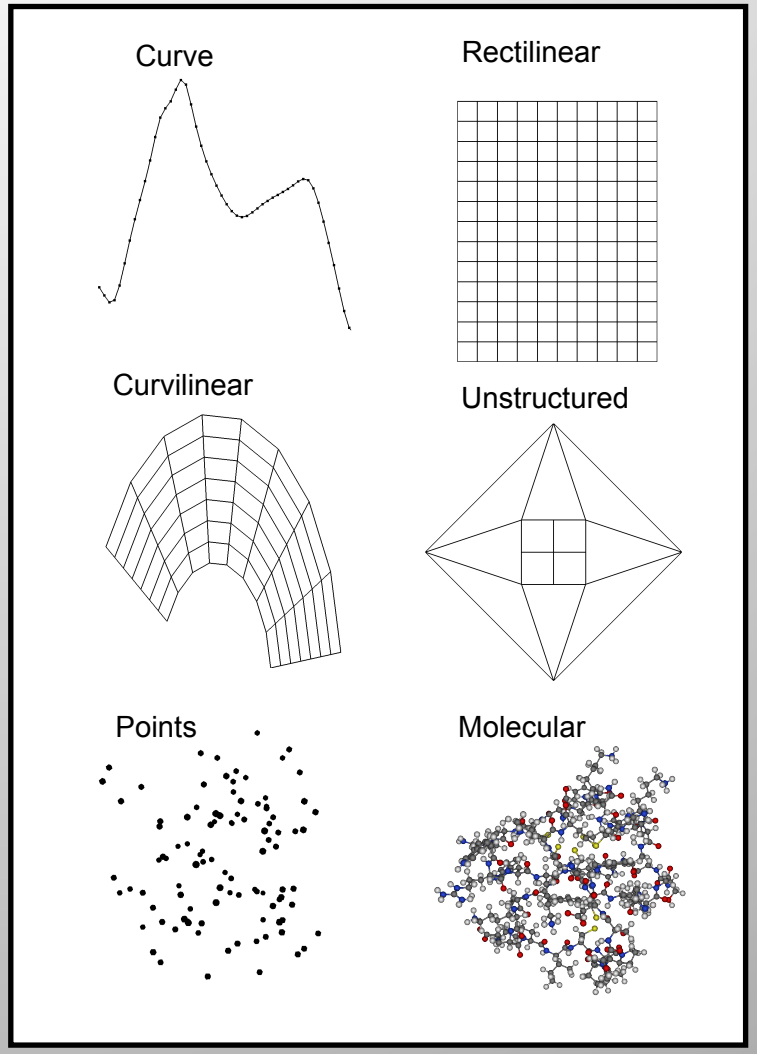
In order to start, we must understand VisIt's Data Model

- A very rich data model
  - Closer to the "computational model"
- Internally implemented with VTK
- Many conventions built on top of VTK

Meshes

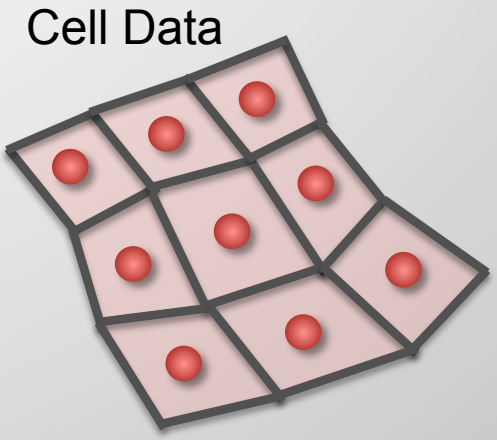
- All data in VisIt lives on a mesh
- Discretizes space into points and cells
  - (1D, 2D, 3D) + time
  - Mesh dimension need not match spatial dimension *(e.g. 2D surface in 3D space)*
- Provides a place for data to be located
- Defines how data is interpolated

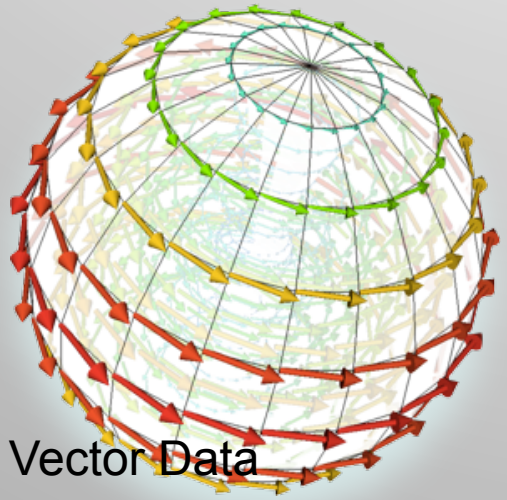
Mesh Types

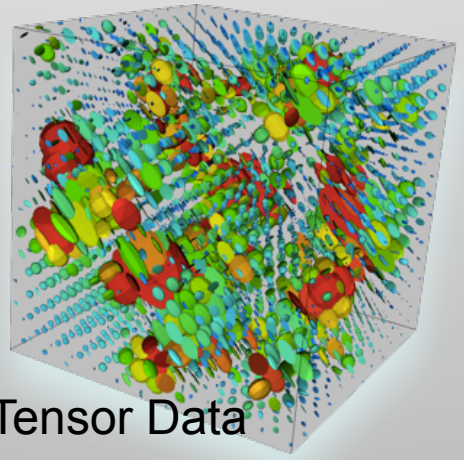


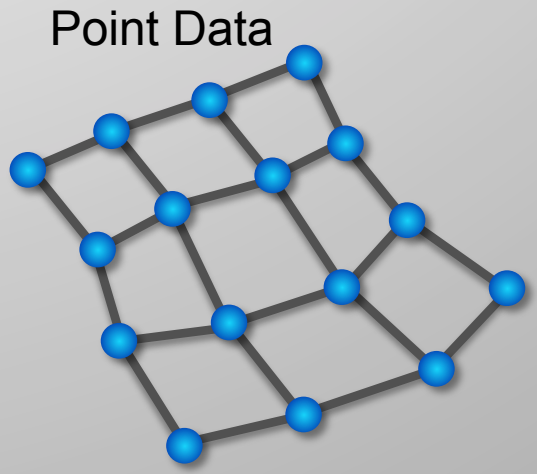
Variables

- Scalars, Vectors, Tensors
- Sits on 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)



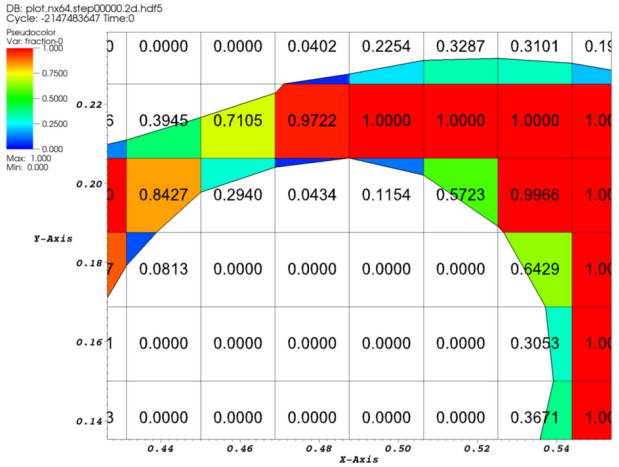

Vector Data

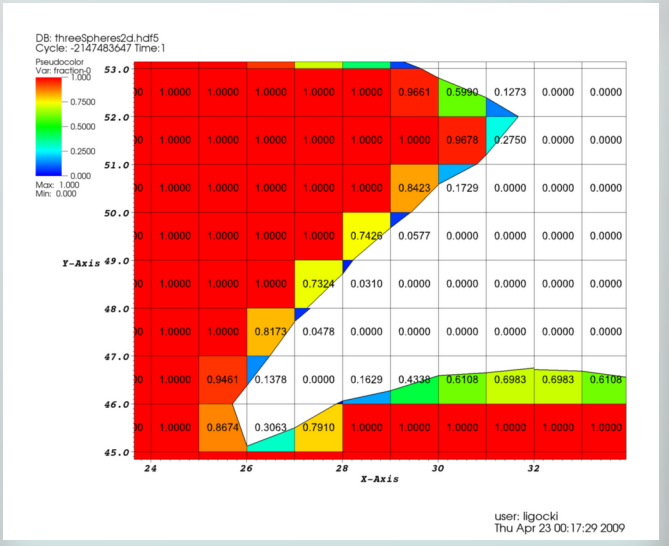

Tensor Data

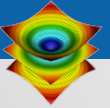


Materials

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





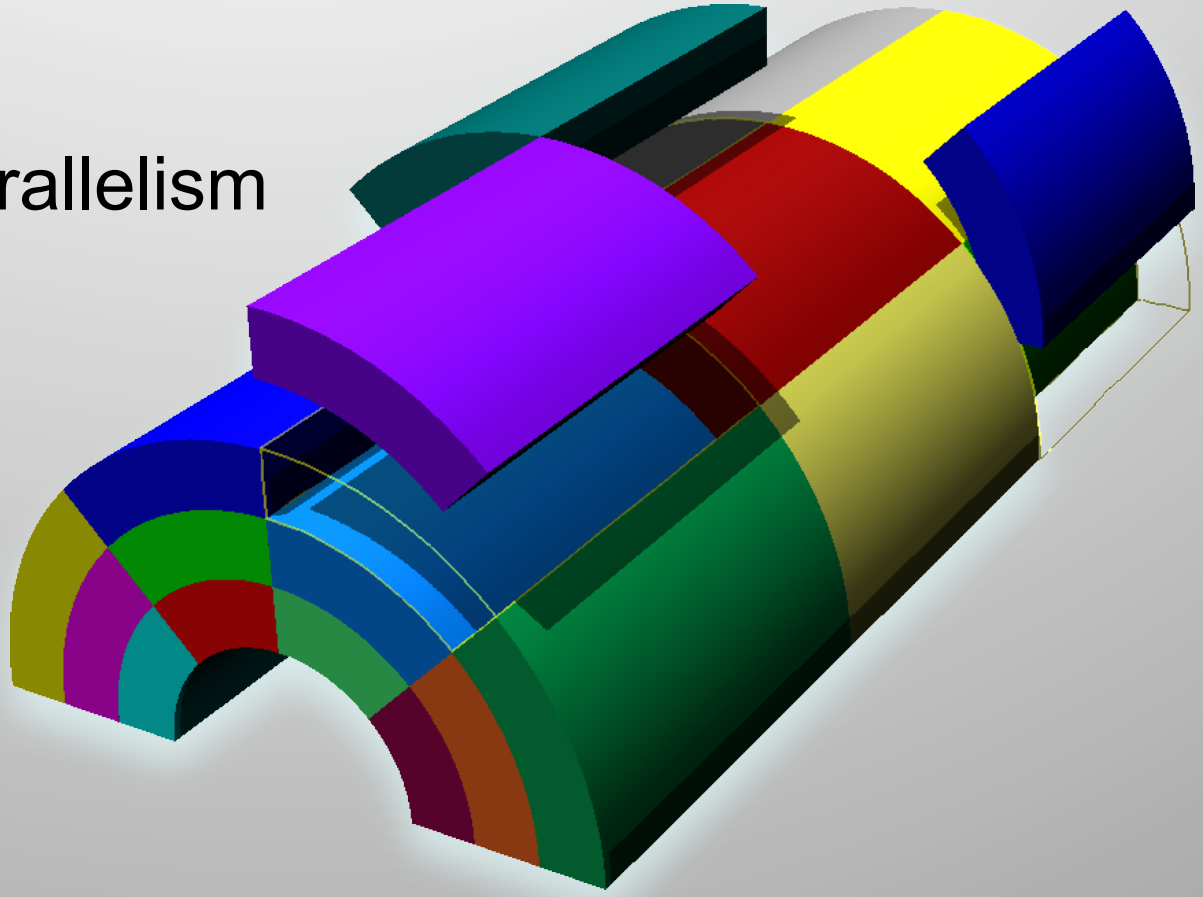


Species

- Similar to materials, describes sub-grid variable composition
  - Example: *Material "Air" is made of species "N_2" ,"O_2", "Ar", "CO_2", etc.*
- Used for mass fractions
- Generally weights other scalars (e.g. partial pressure)

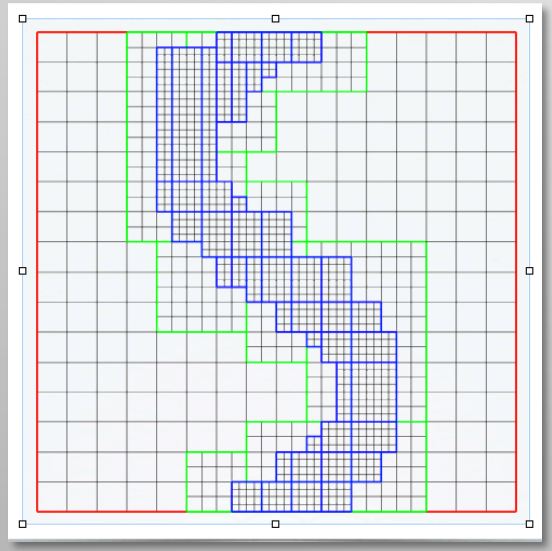
Parallel Meshes

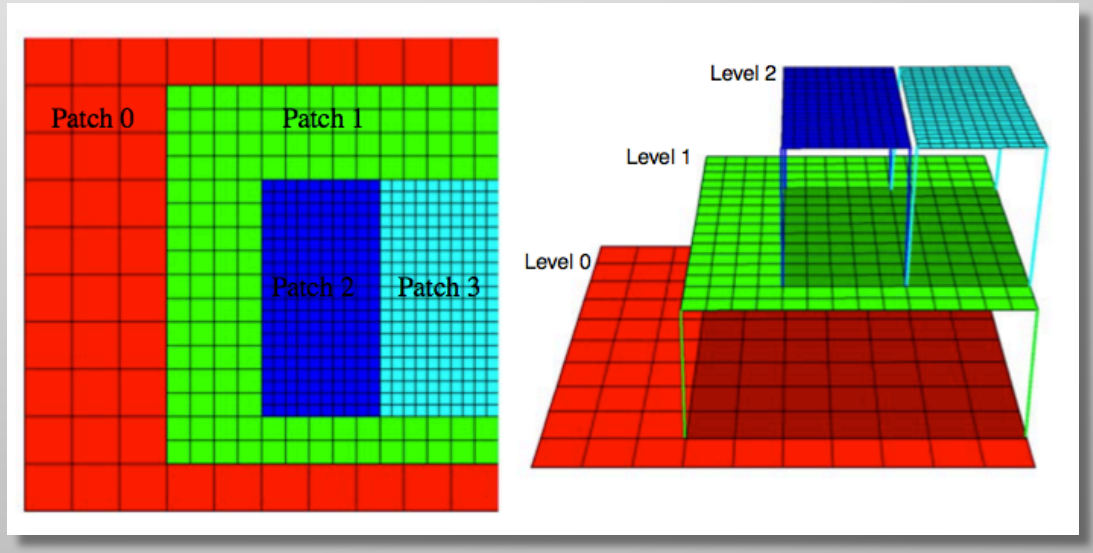
- Provides aggregation for meshes
- A mesh may be composed of large numbers of mesh “blocks”
- Allows data parallelism



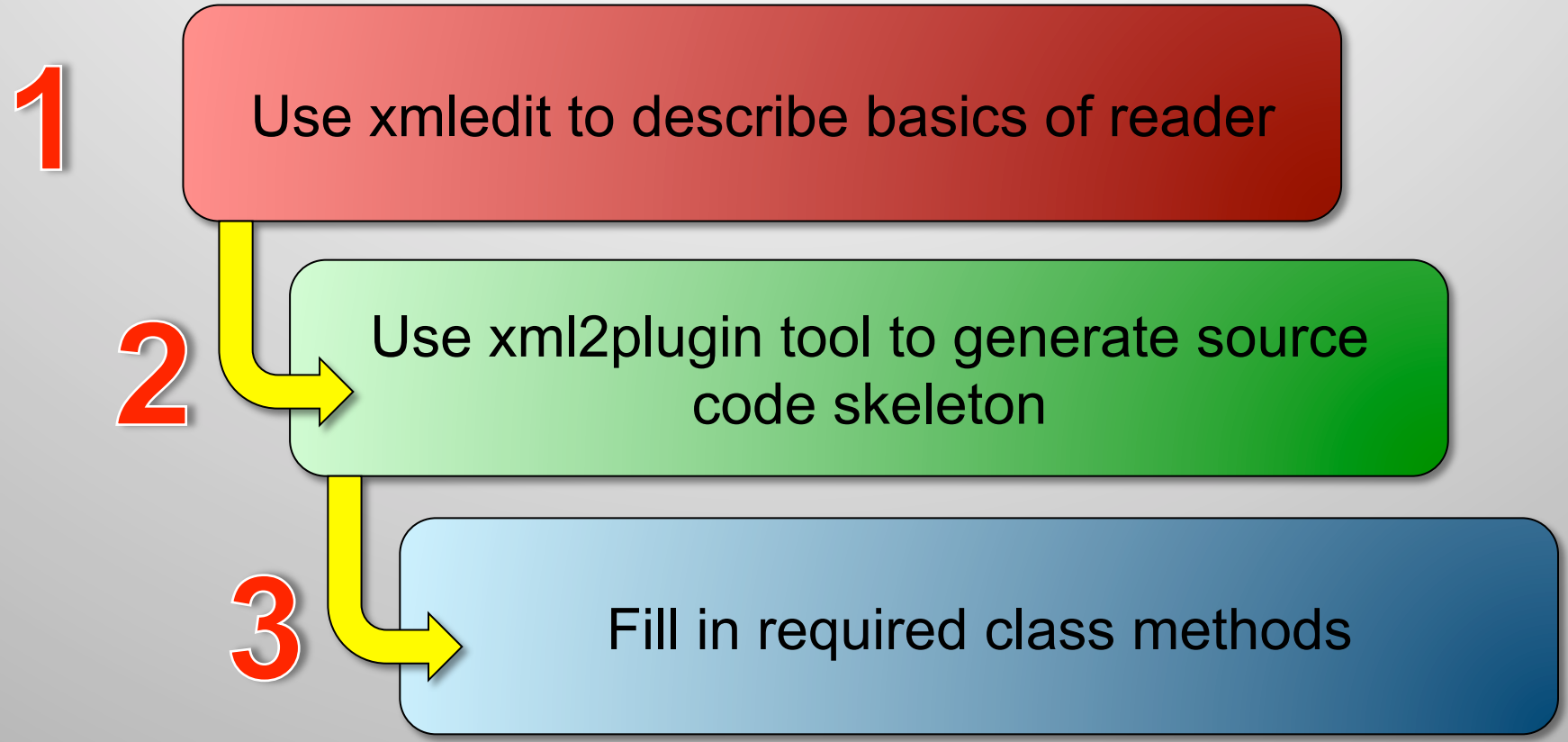
AMR meshes

- Mesh blocks can be associated with patches and levels
- Allows for aggregation of meshes into AMR hierarchy levels





Three Basic Steps to Create Reader



We will make an "XYV" plugin

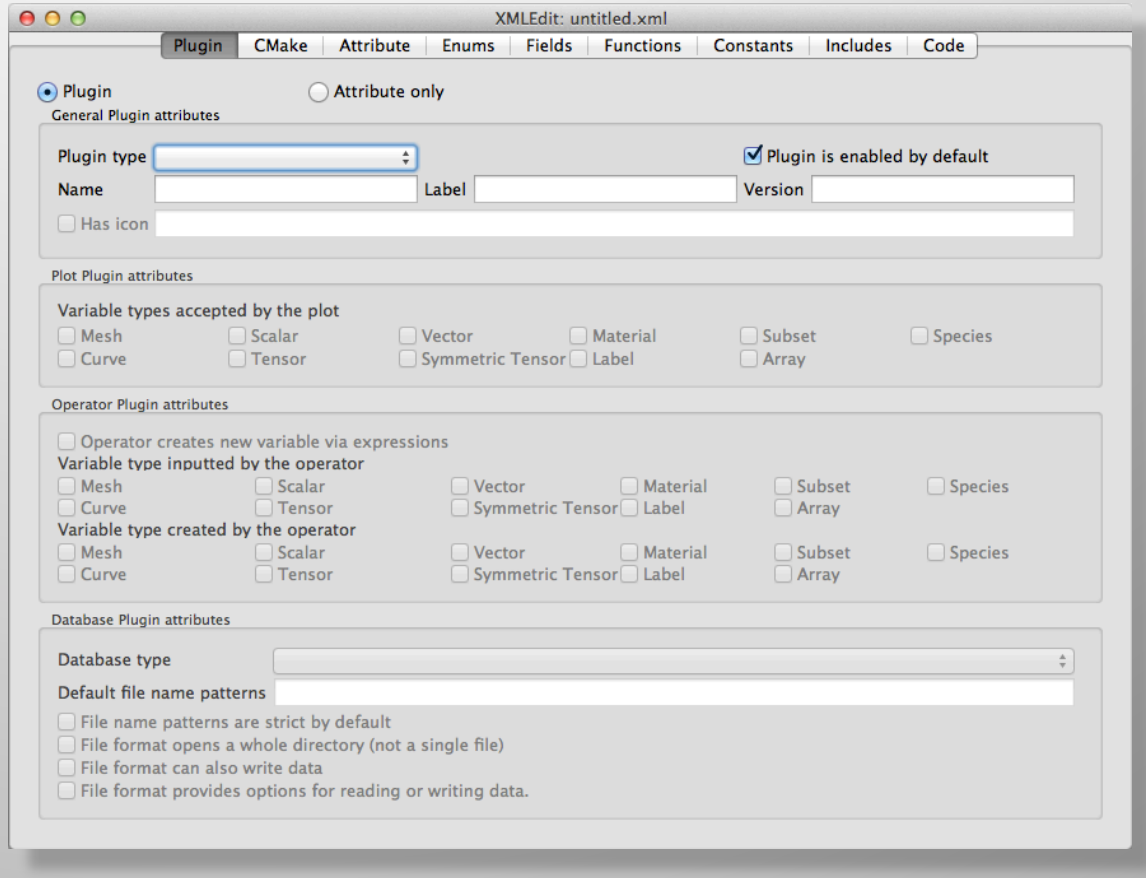
- Reads single text file of 2D points with data
- Each time step exists in a separate file
- One header line that gives the names of the variables
- All lines after header contain 3 floating point numbers: *x y v*

data0000.ieee

```
# x y v
4.23753518051 -6.28432838795 0.521763945345
-2.27336977284 1.23918450369 1.42646054811
-4.03918487303 -0.286620182048 0.522498245438
0.639292855503 -2.96498998734 0.90514247344
-5.7708632707 0.854749967412 0.595733223819
4.22884876129 -2.7545817101 1.33946270708
-1.50187023469 7.40731971096 2.06474912551
-1.84358988266 -1.69603038028 2.11343808748
8.28952594174 -5.27925078513 1.3699229709
3.33254051205 -8.38358598386 0.593437056273
-2.6929805368 6.78330995203 2.40600790499
5.50405350169 -4.06063996957 1.17023275219
-0.991625825469 0.175361206959 0.634724902132
-5.59374324464 2.45526494403 1.17599040687
8.25784898862 -5.02401404284 1.43114782698
-8.5629769287 1.24561359521 0.592438175926
-0.530331303097 8.03279085595 1.69648857654
-0.133894656327 -0.0422984084432 0.892558277144
-0.754588638463 -3.91619940375 2.04051844594
3.02755189408 -5.95624038806 0.512772994571
4.93565142025 0.0354412788418 2.49976798792
1.64271535366 -1.80298026979 0.701480018933
-3.96181246308 7.75087469904 2.4882836195
2.0537216104 5.2904698864 0.603919612071
```

XmlEdit

- GUI tool to edit plugin XML descriptions
- Used to define type of database, filename extensions, etc.
- Creates XML file that describes your reader

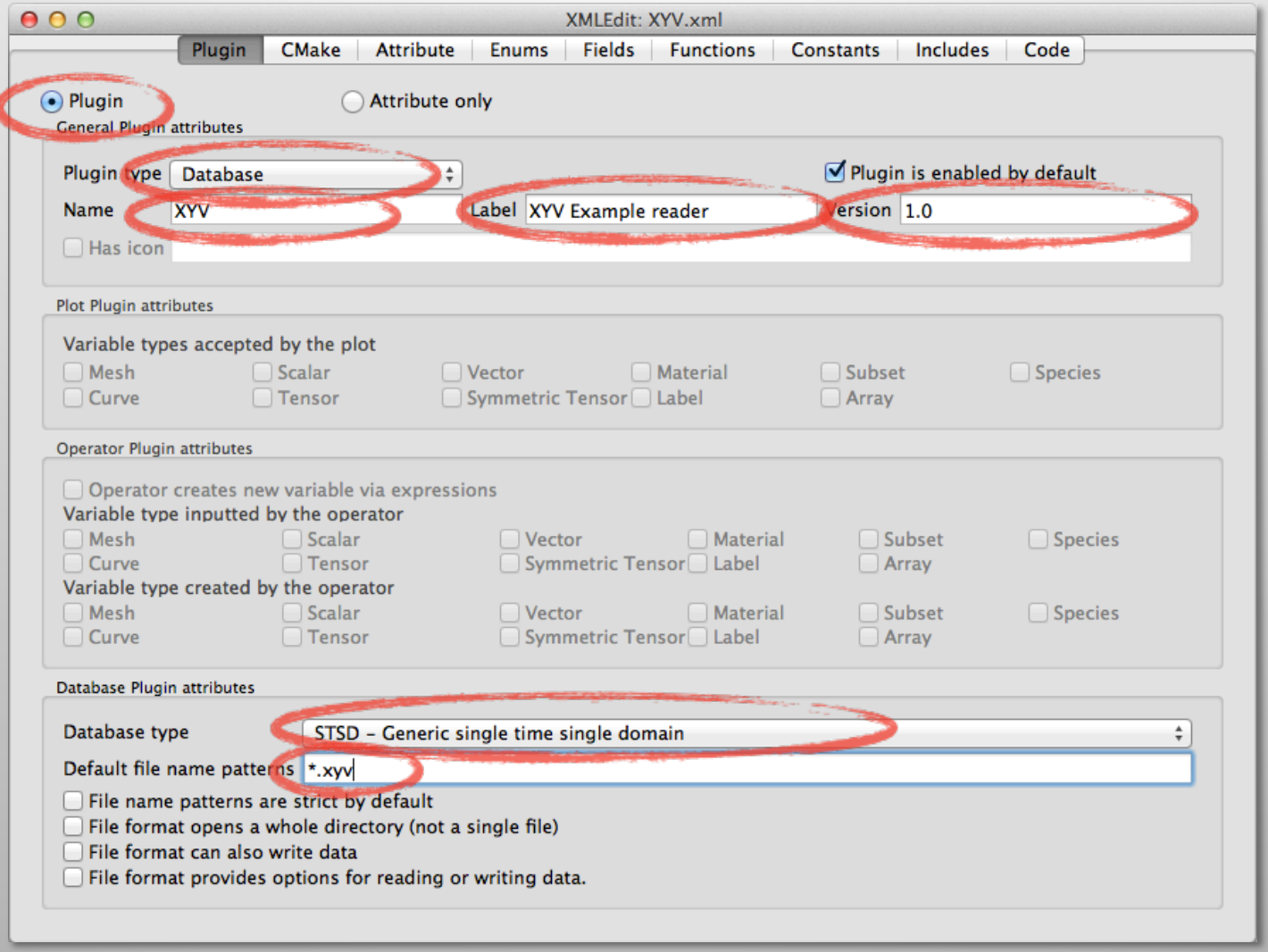


Types of Database Plugins

- Two axes: domains and time
- Moving down and to the right adds complexity

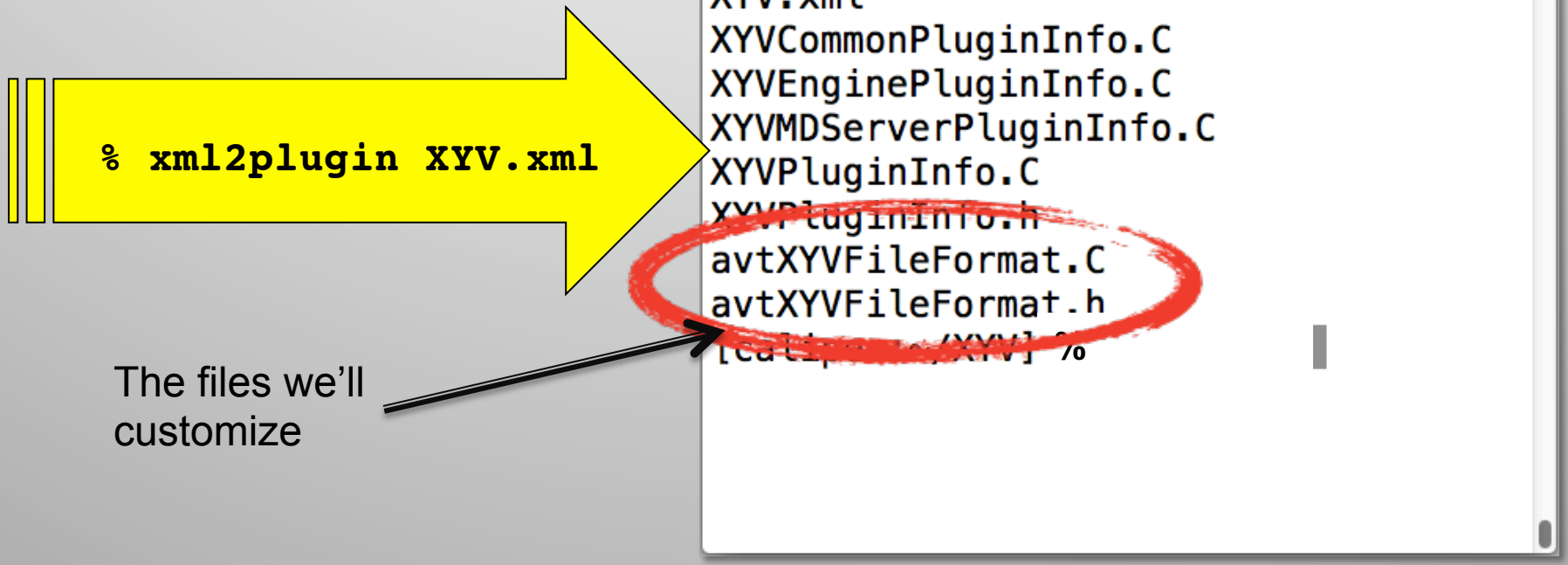
	Single Domain	Multiple Domain
Single Timestep	STSD	STMD
Multiple Timesteps	MTSD	MTMD

Run XmlEdit and save XYV.xml



Generate Plugin Skeleton

- xml2plugin takes your XML file and generates many files

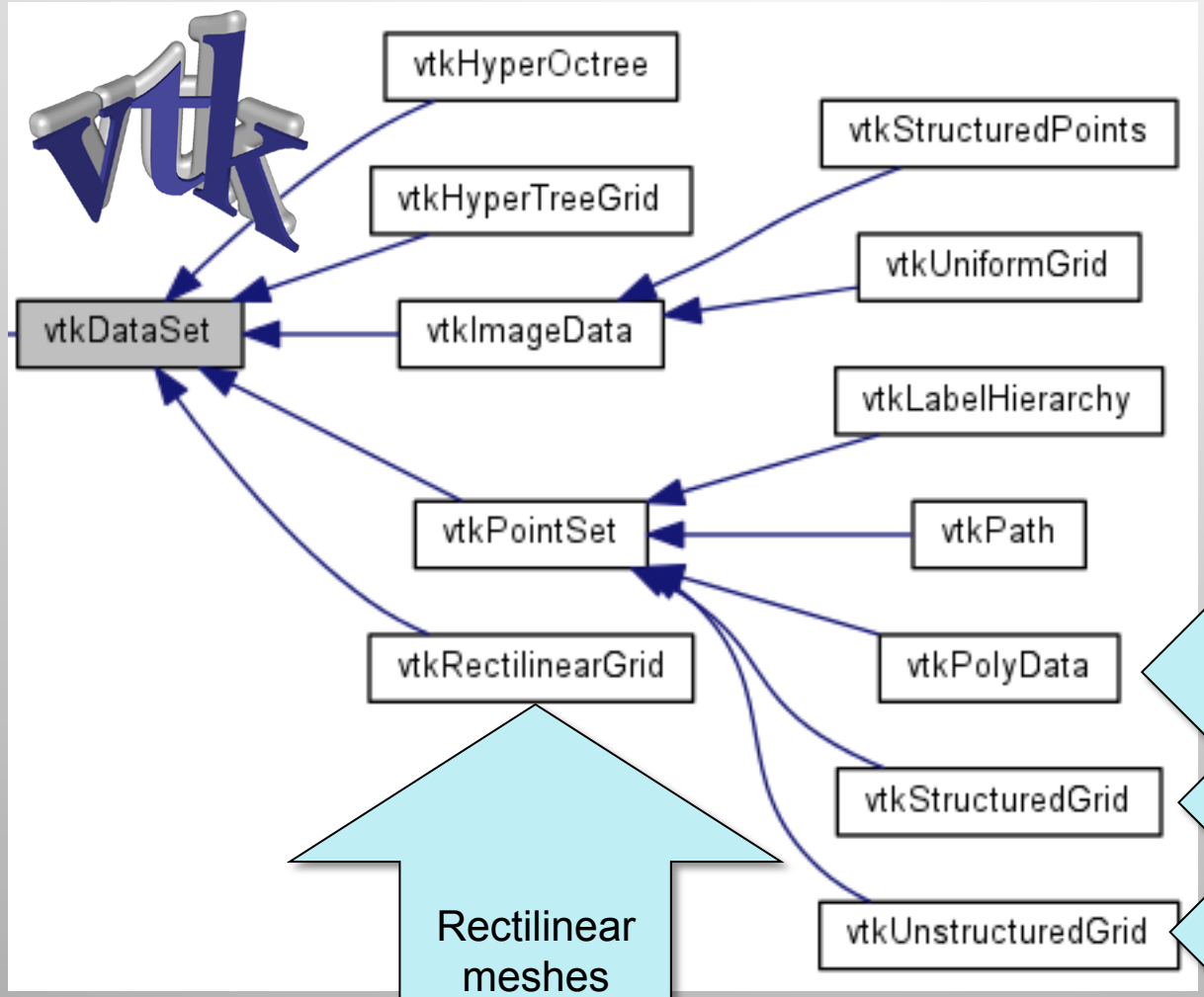


Fill in methods for avtXYVFileFormat

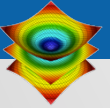
- You need to write code to read the file and populate VTK objects
- See ***Getting Data Into VisIt*** manual
- Templates to create VTK objects at:
  - http://portal.nersc.gov/svn/visit/trunk/src/tools/DataManualExamples/DatabasePlugin/

Method	Description
Constructor	Class constructor
Destructor	Class destructor
PopulateDatabaseMetaData	Fill in metadata that describes the meshes and variables present in the data file
GetMesh	Return vtkDataSet subclass containing the mesh
GetVar	Return vtkDataArray subclass containing variable
GetVectorVar	Just return GetVar()

vtkDataSet is the base class for many meshes in VTK



Create and fill out the appropriate VTK class for the mesh you want to return from your plugin



The Plan

- Add vtkUnstructuredGrid called *grid* to contain the X,Y point mesh
- Add vtkFloatArray called *var* to contain the V value
- Read the file in a ReadData() method
- Call ReadData() from GetMesh() and return reference to *grid*
- Call ReadData() from GetVar() and return reference to *var*

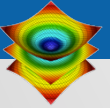
PopulateDatabaseMetaData

```
void
avtXYVFileFormat::PopulateDatabaseMetaData(avtDatabaseMetaData *md)
{
    ReadHeader();

    avtMeshMetaData *mesh = new avtMeshMetaData;
    mesh->name = "mesh";
    mesh->meshType = AVT_POINT_MESH;
    mesh->blockOrigin = 0;
    mesh->spatialDimension = 2;
    mesh->topologicalDimension = 0;
    mesh->hasSpatialExtents = false;
    md->Add(mesh);

    if (varFound)
        AddScalarVarToMetaData(md, varname, "mesh", AVT_NODECENT, NULL);
}
```

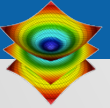
Describe the mesh



GetMesh

```
vtkDataSet *
avtXYVFileFormat::GetMesh(const char *meshname)
{
    ReadData();

    // Update the pointer count and return;
    grid->Register(NULL);
    return grid;
}
```



GetVar

```
vtkDataArray *
avtXYVFileFormat::GetVar(const char *varname)
{
    ReadData();

    // Set the data array to our read variable.
    vtkDataArray *rv = var;

    // Adjust the pointer count and return.
    rv->Register(NULL);
    return rv;
}
```

Custom method: ReadData

```
void avtXYVFileFormat::ReadData()
{
}
```

Read all lines in file
Store values in x,y,v vectors

Use x,y vectors to make vtkPoints

Use vtkPoints to make vtkUnstructuredGrid

Use v vector to make vtkFloatArray

Custom method: ReadData

```
void avtX
{
    U
    Use vt
    Us
}
```

```
vector<float> x;
vector<float> y;
vector<float> v;
char line[1024];
debug4 << filename << ": Processing all lines" << endl;
while (!ifile.eof())
{
    debug4 << filename << ": Reading line" << endl;
    ifile.getline(line, 1024);
    float xv, yv, vv;
    if (varFound)
        sscanf(line, "%f %f %f", &xv, &yv, &vv);
    else
        sscanf(line, "%f %f", &xv, &yv);
    x.push_back(xv);
    y.push_back(yv);
    if (varFound)
        v.push_back(vv);
}
```

Custom method: ReadData

```
void avtX
{
    U
    Use vt
    Us
}
```

```
int npts = x.size();

// Create vtkPoints structure for the points.
vtkPoints *points = vtkPoints::New();
points->SetNumberOfPoints(npts);
for (int i = 0; i < npts; i++)
    points->SetPoint(i, x[i], y[i], 0);
```

Custom method: ReadData

```
void avtX
{
    U
    Use vt
    Us
}
```

```
// Now create a vtkUnstructuredGrid for the real mesh.
grid = vtkUnstructuredGrid::New();
grid->SetPoints(points);
points->Delete();

// Grids have to have cells to define them.
grid->Allocate(npts);
vtkIdType oneVertex[1];
for (int i = 0; i < npts; i++)
{
    oneVertex[0] = i;
    grid->InsertNextCell(VTK_VERTEX, 1, oneVertex);
}
```

Custom method: ReadData

```
void avtX
{
    U
    Use vt
    Us
}
```

```
    // If we have a variable, get it.
    if (varFound)
    {
        var = vtkFloatArray::New();
        var->SetNumberOfTuples(npts);
        for (int i = 0; i < npts; i++)
            var->SetTuple1(i, v[i]);
    }

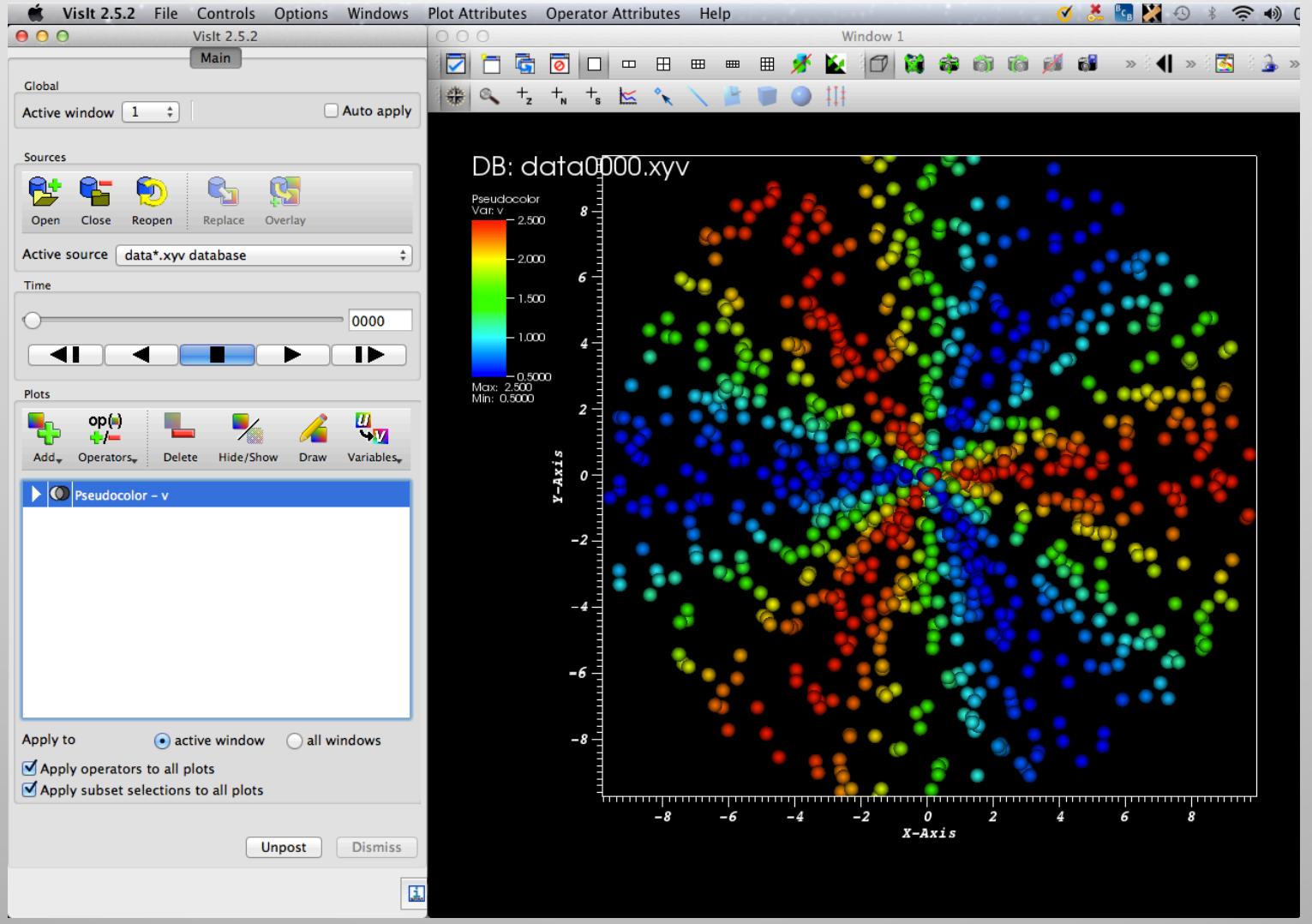
    fileRead = true;
}
```

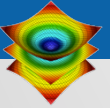
Build and Run

```
Build commands:
% cd XYV
% cmake .
% make
```

- Make will automatically install your plugin to your ~/.visit directory
- VisIt will load your plugin at launch

Run

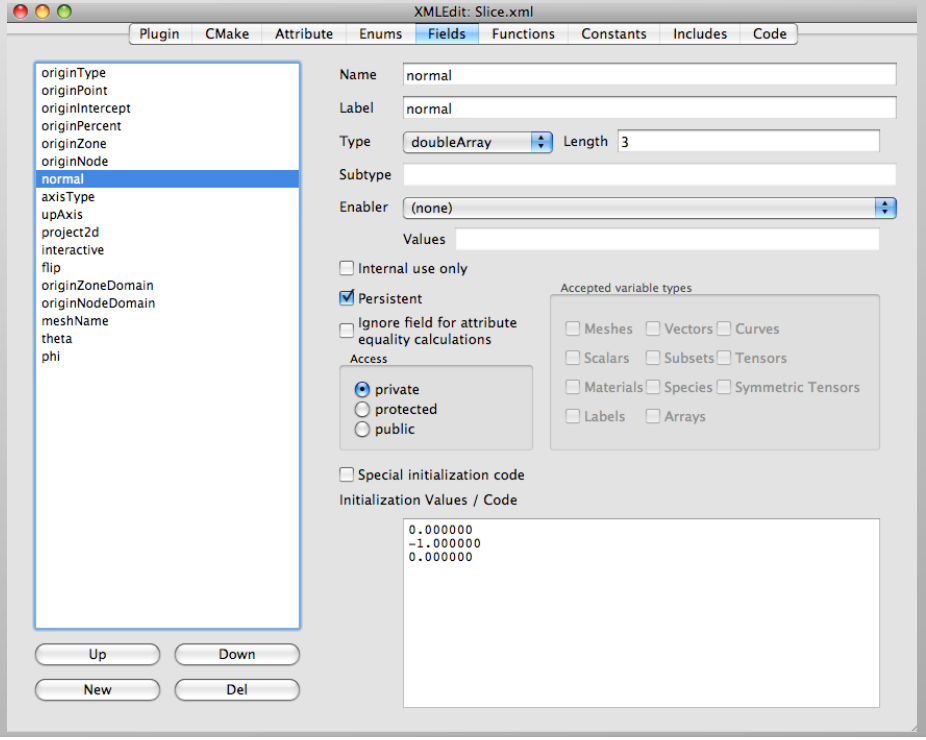

VisIt 2.5.2
File
Controls
Options
Windows
Plot Attributes
Operator Attributes
Help
VisIt 2.5.2
Main
Global
Active window 1
Auto apply
Sources
Open
Close
Reopen
Replace
Overlay
Active source data*.xyv database
Time
0000
Plots
Add
Operators
Delete
Hide/Show
Draw
Variables
Pseudocolor – v
Apply to
active window
all windows
Apply operators to all plots
Apply subset selections to all plots
Unpost
Dismiss
Window 1
DB: data0000.xyv
Pseudocolor
Var: v
2.500
2.000
1.500
1.000
0.5000
Max: 2.500
Min: 0.5000
Y-Axis
X-Axis



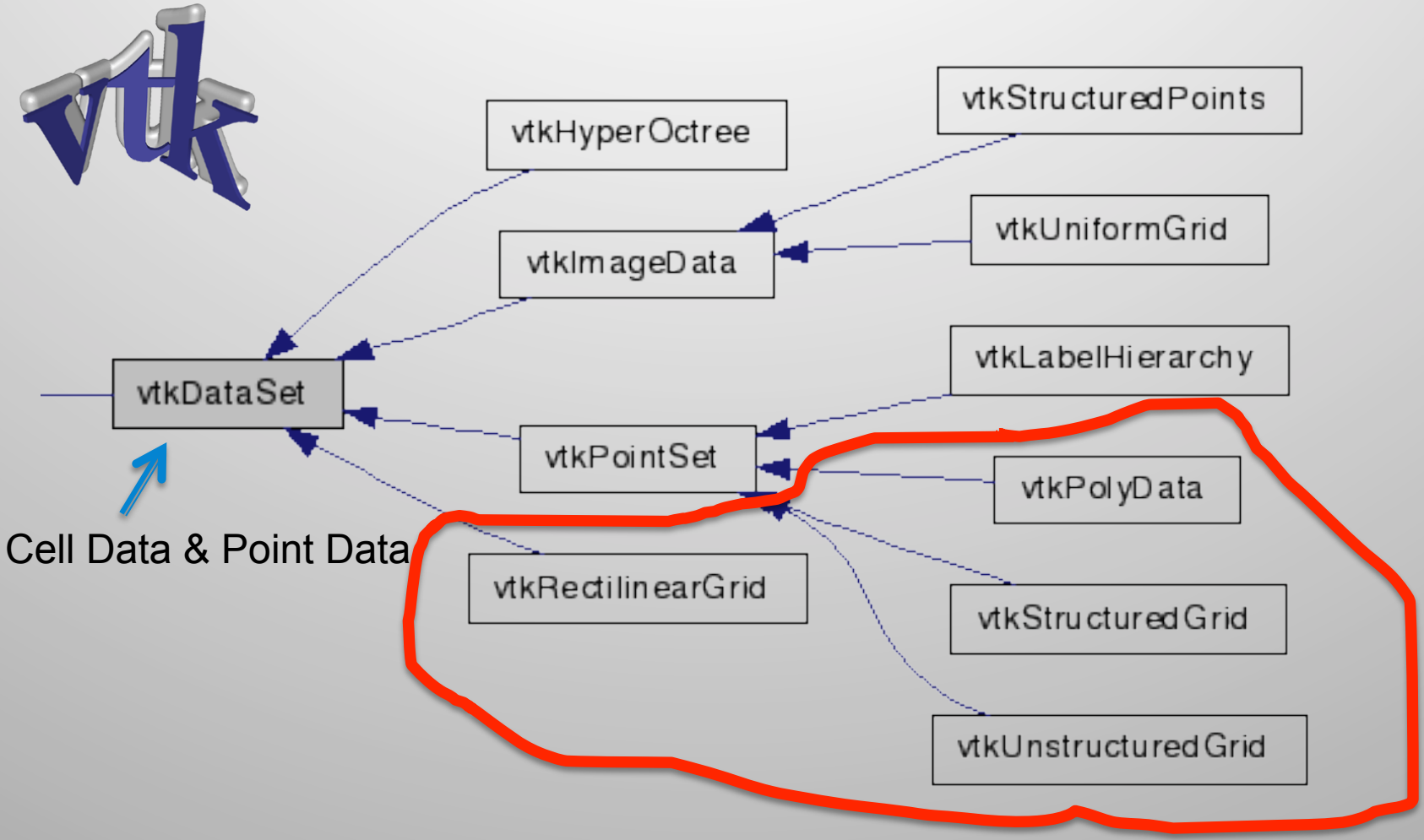
Customizing VisIt:
Writing an analysis operator

Writing a plugin for VisIt

- 4 Elements:
  - Defining attributes
  - User interface (Qt, Python, etc)
  - Formatting as a shared library VisIt can load
  - Data manipulation code to carry out operation (typically using VTK)
- VisIt's XML tools code generate the first three.

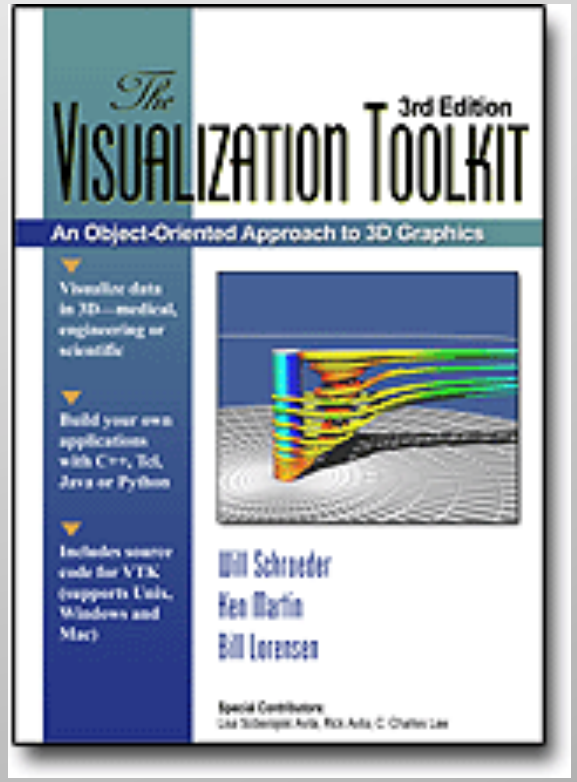


Data model


vtkHyperOctree
vtkStructuredPoints
vtkImageData
vtkUniformGrid
vtkDataSet
vtkLabelHierarchy
vtkPointSet
vtkPolyData
Cell Data & Point Data
vtkRectilinearGrid
vtkStructuredGrid
vtkUnstructuredGrid

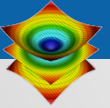
VTK Resources

- ww.vtk.org
  - Download (source and binaries)
  - Documentation
  - Mailing lists
  - Links
  - FAQ, Search
- ww.kitware.com
  - VTK Textbook
  - VTK User's guide
  - Mastering CMake



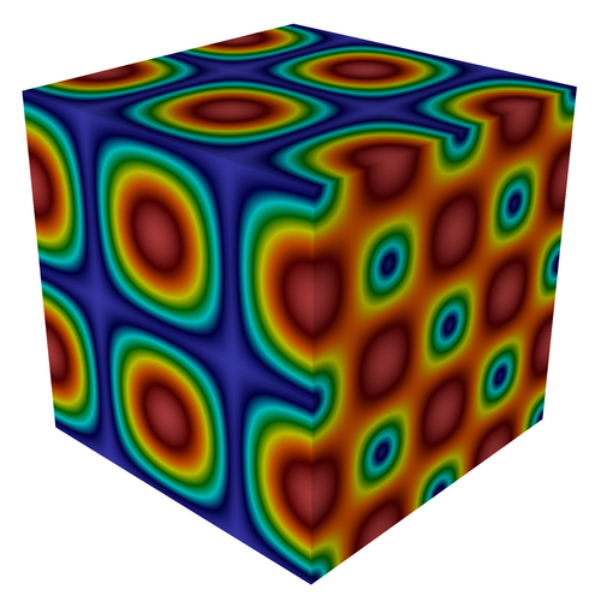
Our example plugin

- Operator plugin: AnalyticSlice
- Goal: construct a distance function and isosurface by that distance function
- User interface:
  - $Ax^2 + Bx + Cy^2 + Dy + Ez^2 + Fz = G$

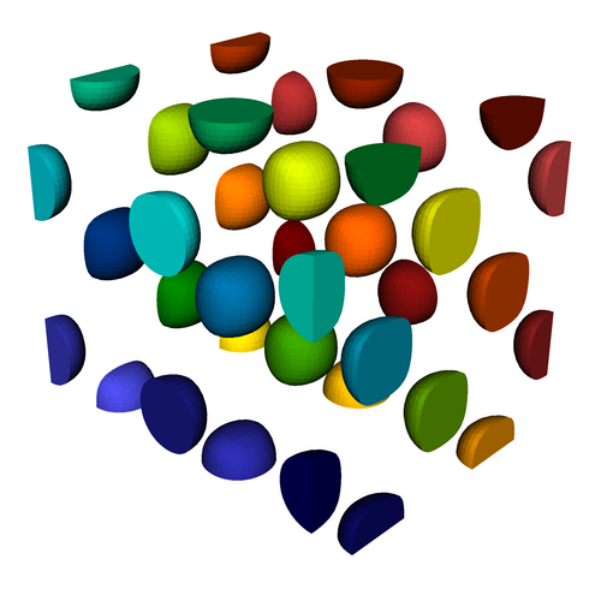


Customizing VisIt: Writing a Python Query

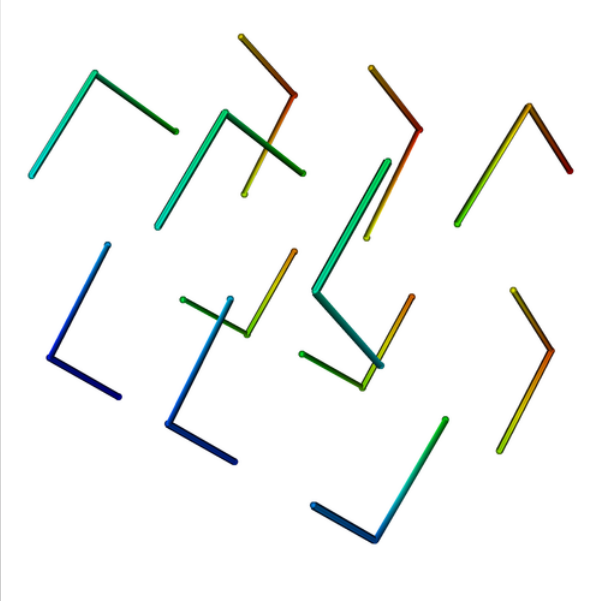
Example: Post-processing connected components of a sub-volume.



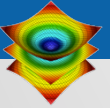
Velocity magnitude



High velocity components

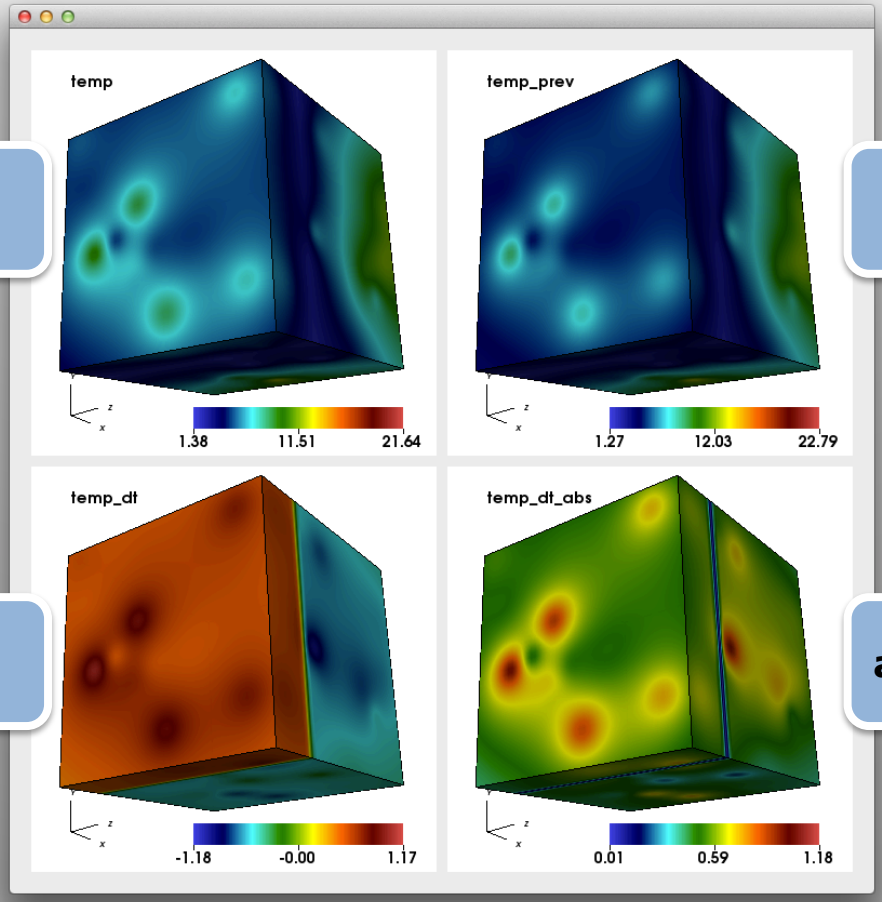


Closest components by centroid



Customizing VisIt:
Creating custom Python applications

Example: Using PySide to create a UI which shows time differences in a scalar field.

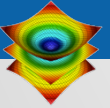


Scalar field at current time step

Scalar field at previous time step

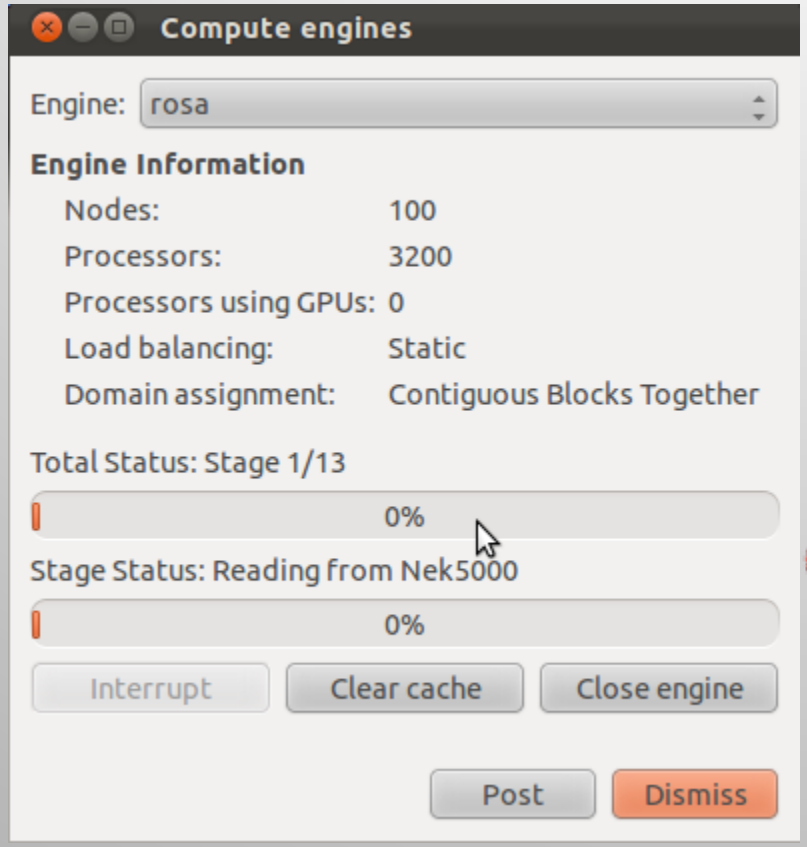
Field difference

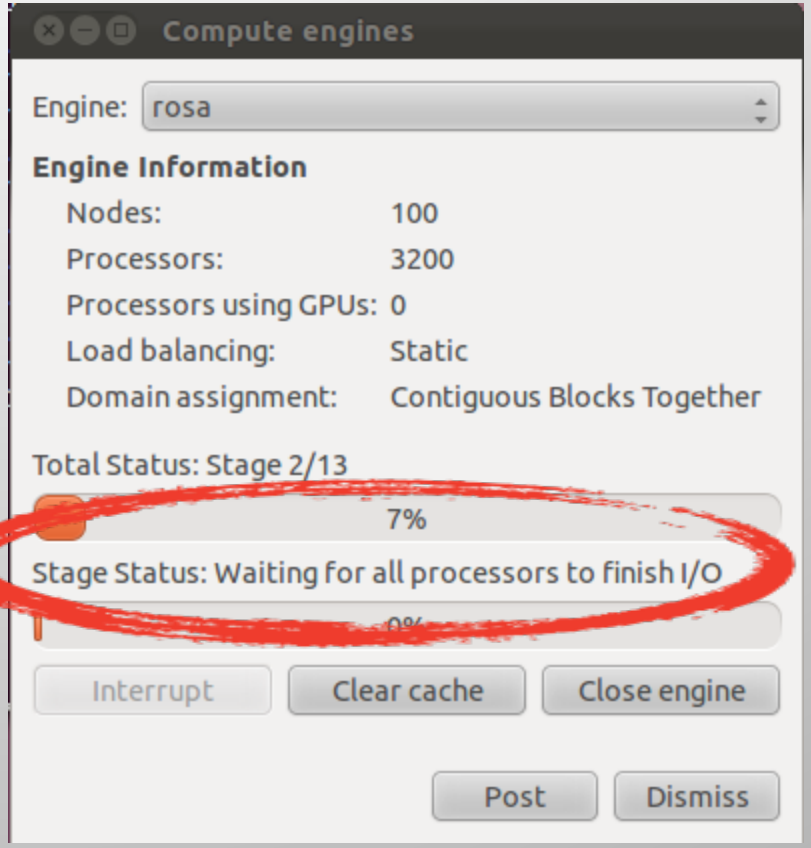
abs(Field difference)



In situ processing

Typical situation in HPC…





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...**

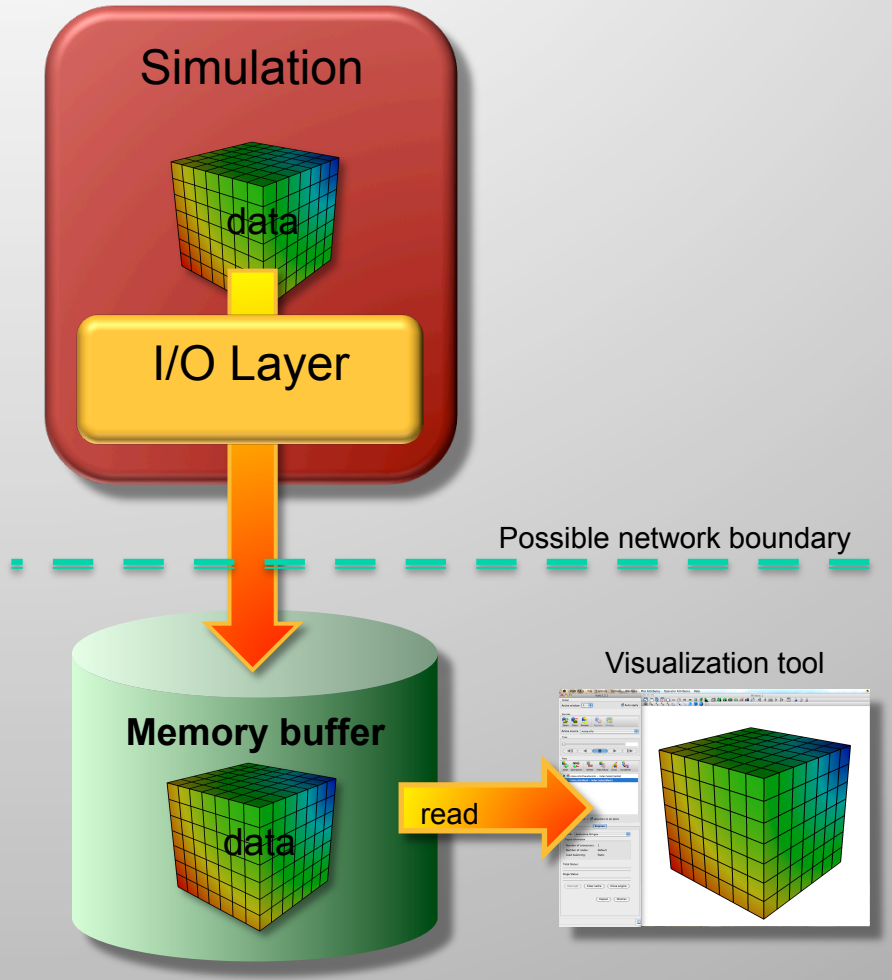
Do not read data from disk but from memory:
***In situ* visualization**

In situ Processing Strategies

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

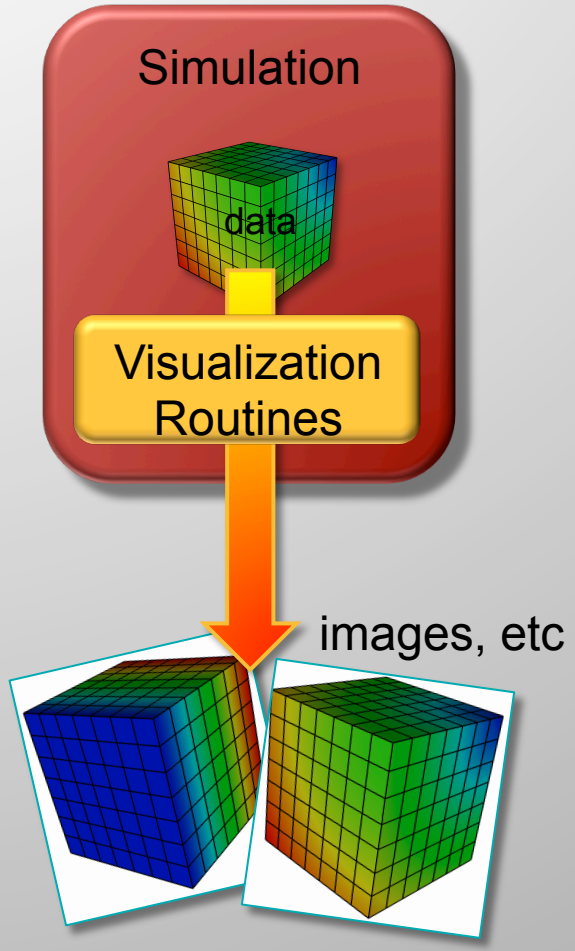
Loosely Coupled *in situ* Processing

- 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



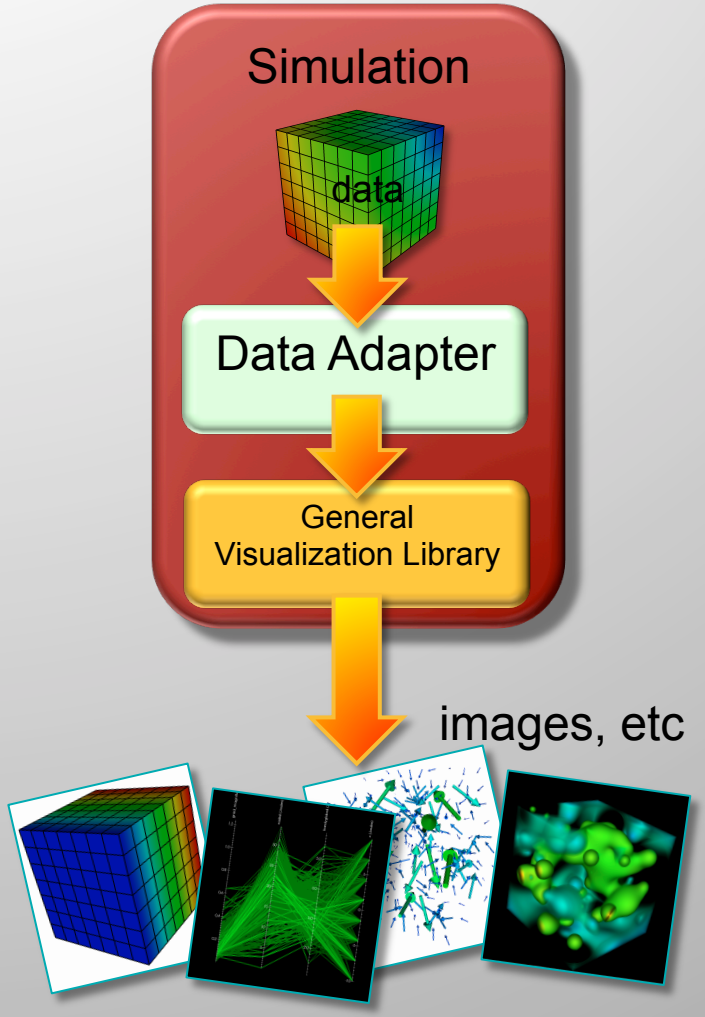
Tightly Coupled Custom *in situ* Processing

- **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***



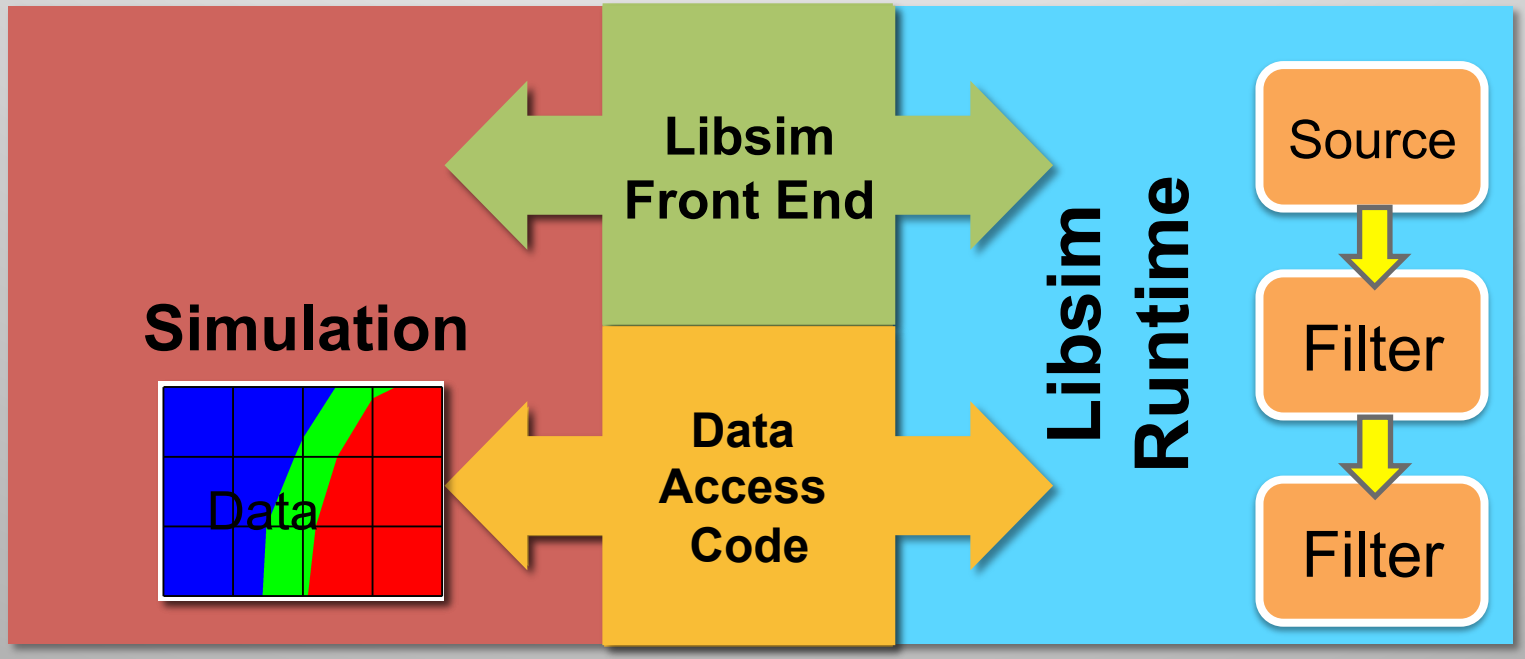
Tightly Coupled General *in situ* Processing

- **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***



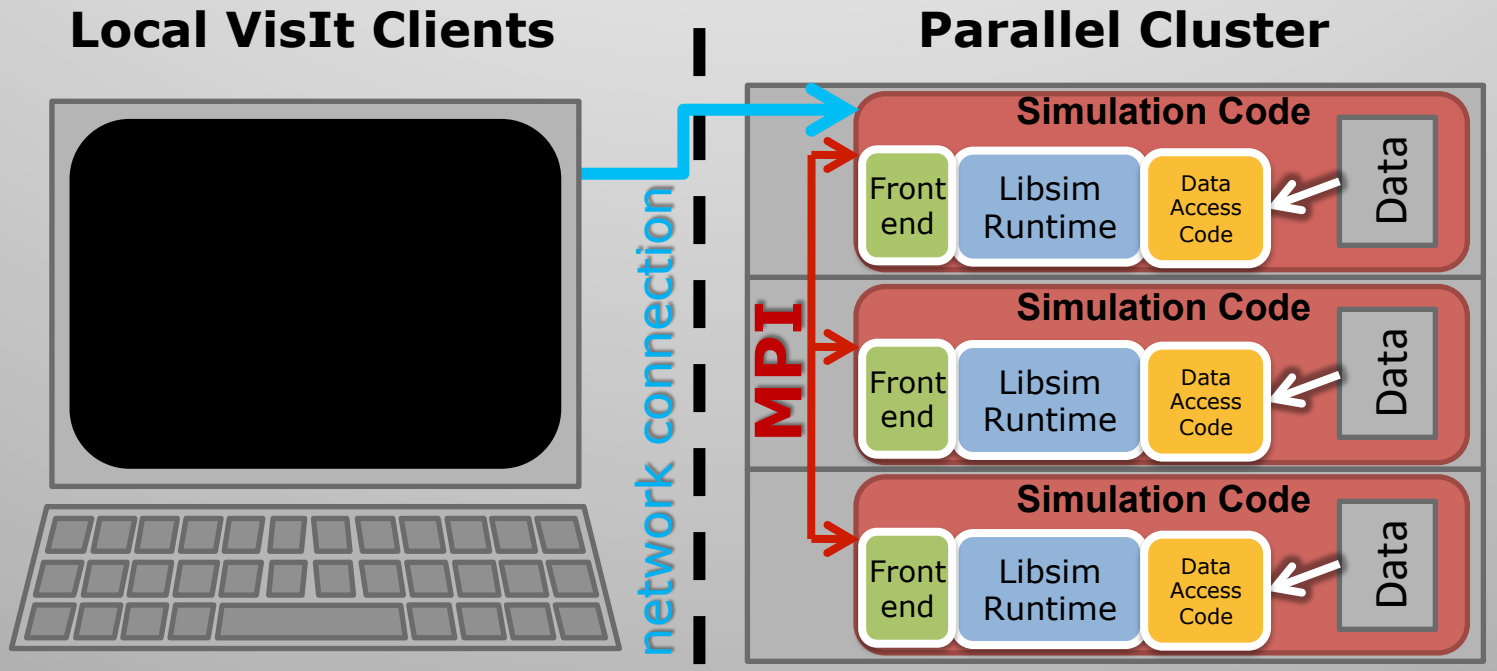
VisIt's libsim implements a tight coupling

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

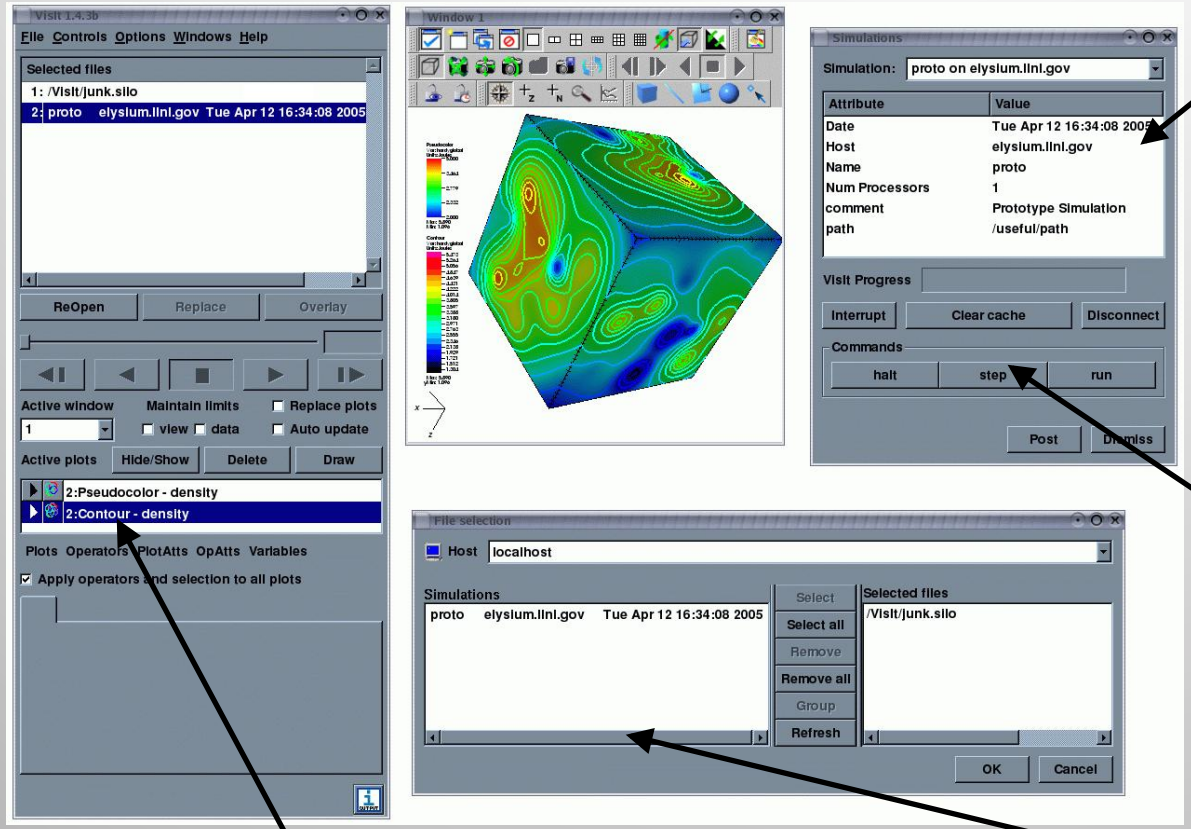


Libsim in practice

- Front-end library lets VisIt connect
- Runtime library processes the simulation's data
- Runtime library obtains data on demand through user-supplied *Data Access Code* callback functions



Libsim from the client side



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

Control commands exposed by the code are available here

All of VisIt's existing functionality is accessible

Users select simulations to open as if they were files

In situ Processing Workflow

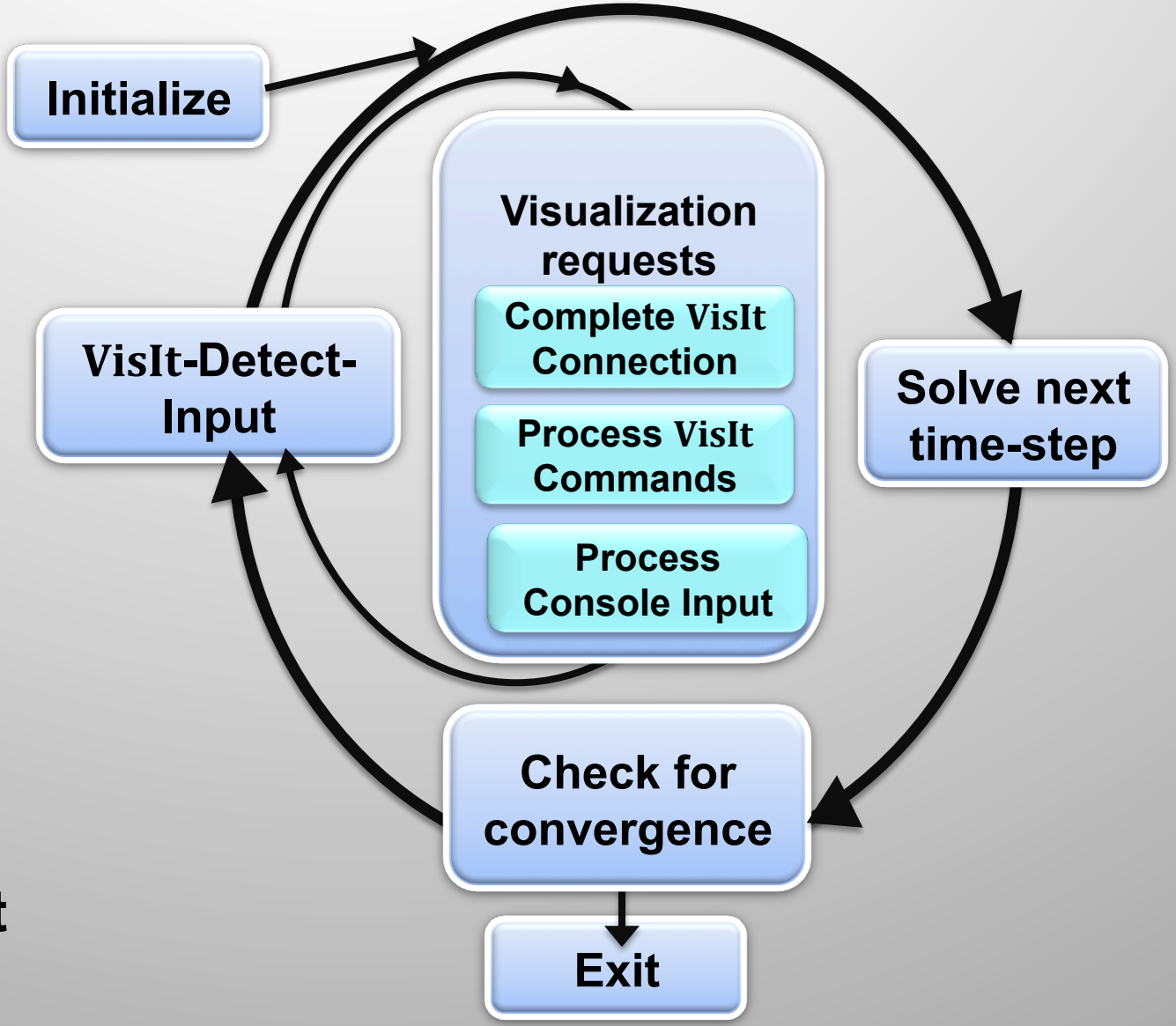
1. The simulation code launches and starts execution
2. The simulation regularly checks for connection attempts from visualization tool
3. The visualization tool connects to the simulation
4. The simulation provides a description of its meshes and data types
5. Visualization operations are handled via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

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



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.

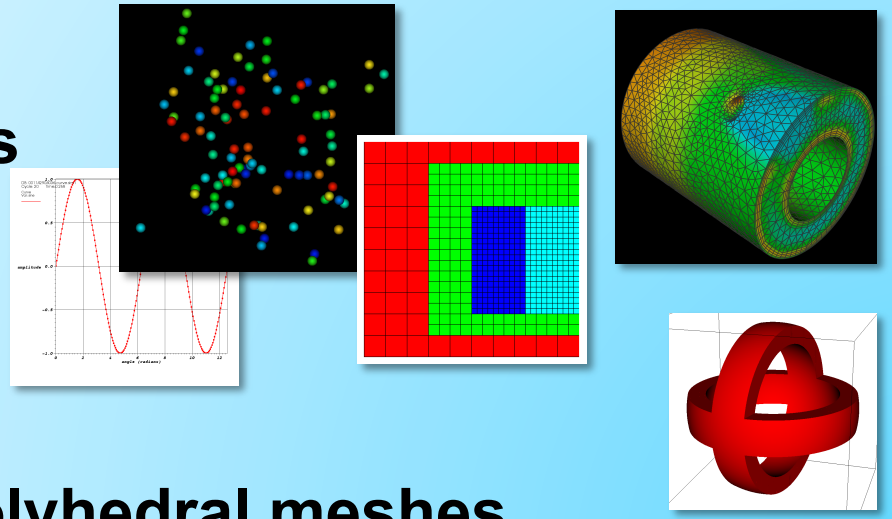
Add control functions that let VisIt steer the simulation.

Data-access callbacks

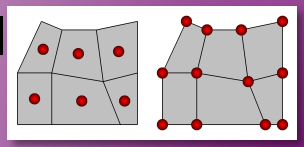
- VisIt requests data *on demand* through data access callback functions
  - Return actual pointers to your simulation's data (nearly zero-copy)
  - Return alternate representation that Libsim can free
  - Written in C, C++, Fortran, Python

Supported Data Model

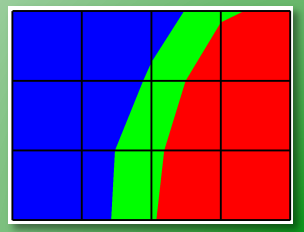
- **Mesh Types**
  - Structured meshes
  - Point meshes
  - CSG meshes
  - AMR meshes
  - Unstructured & Polyhedral meshes



- **Variables**
  - 1 to N components
  - Zonal and Nodal



- **Materials**
- **Species**

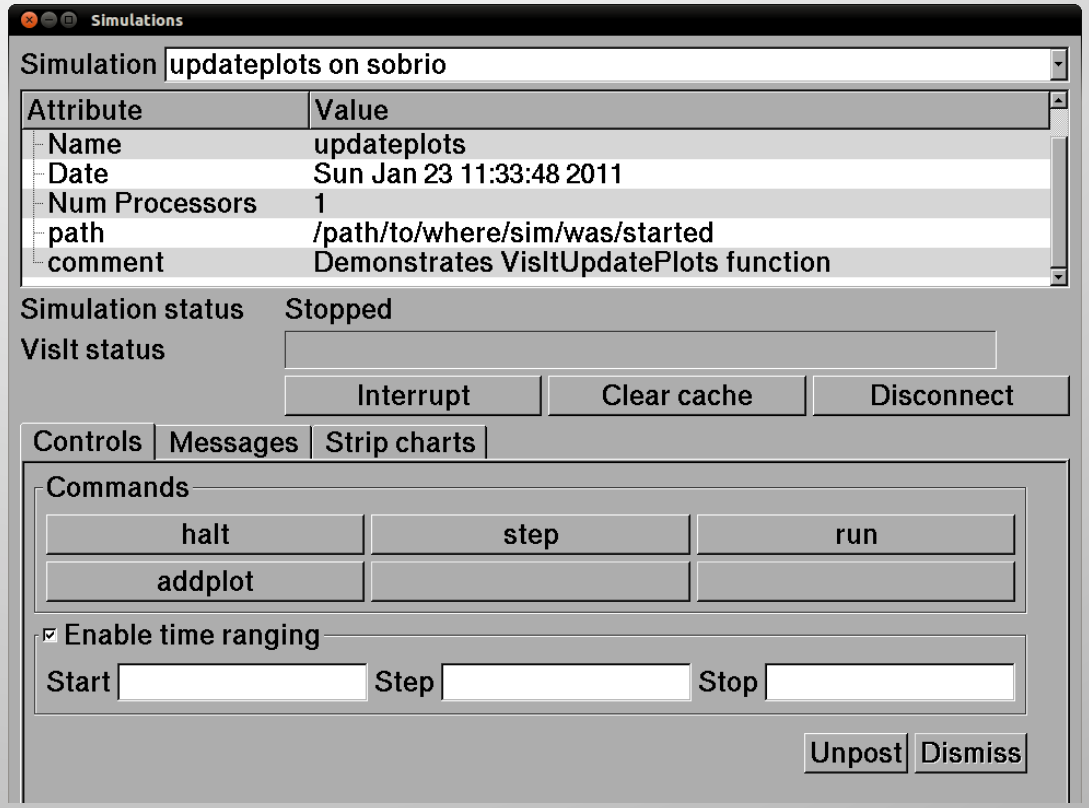


Examples are provided for all these types

- *Getting Data Into VisIt* Manual
  - https://wci.llnl.gov/codes/visit/manuals.html
  - http://portal.nersc.gov/svn/visit/trunk/src/tools/DataManualExamples/Simulations
- http://www.visitusers.org

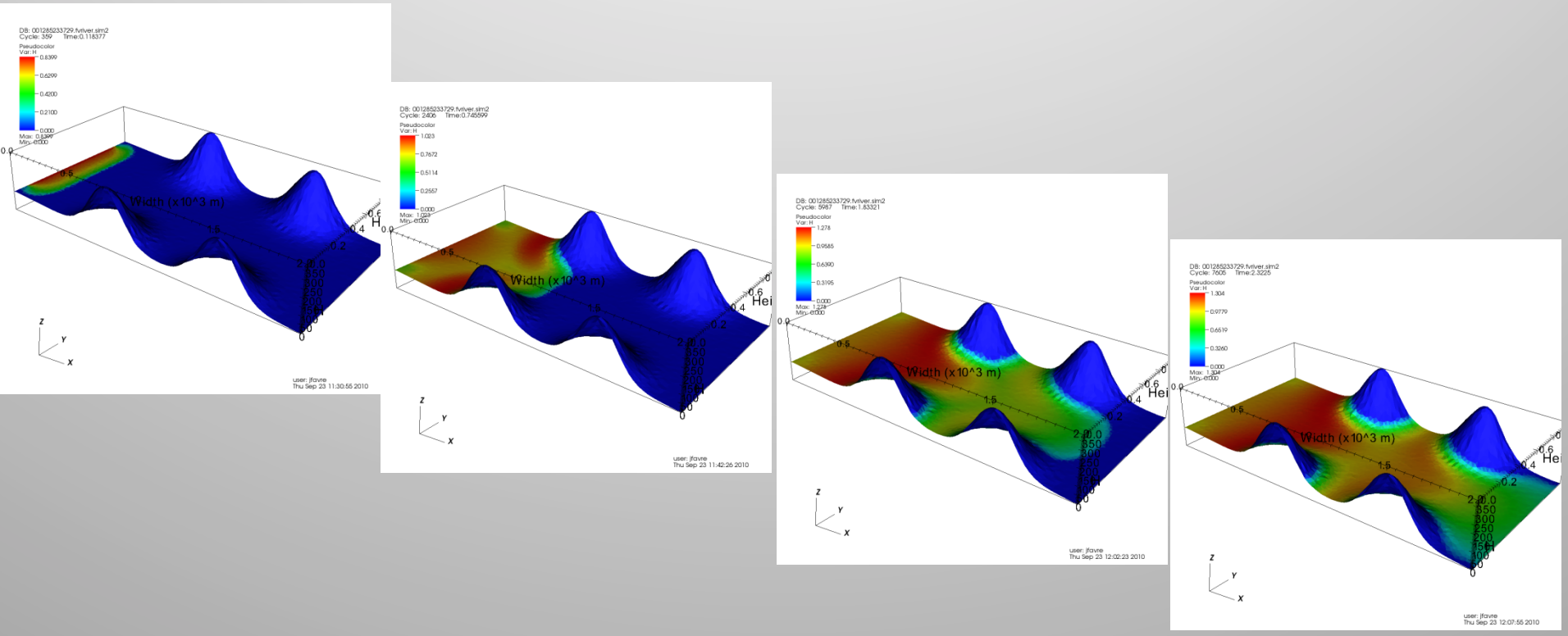
Adding Control Functions

- The simulation provides commands to which it will respond
- Commands generate user interface controls in Simulations Window



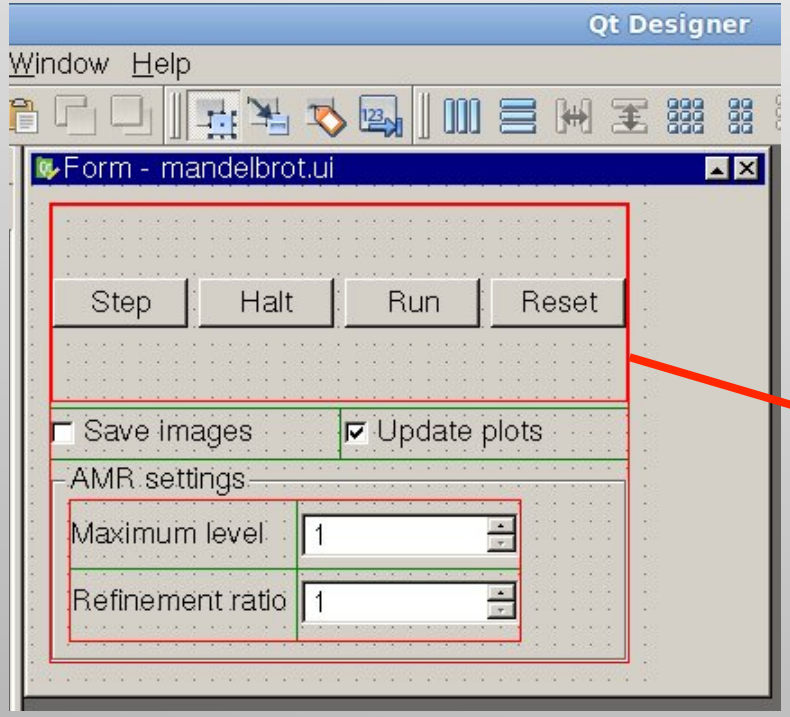
VisIt can control the running simulation

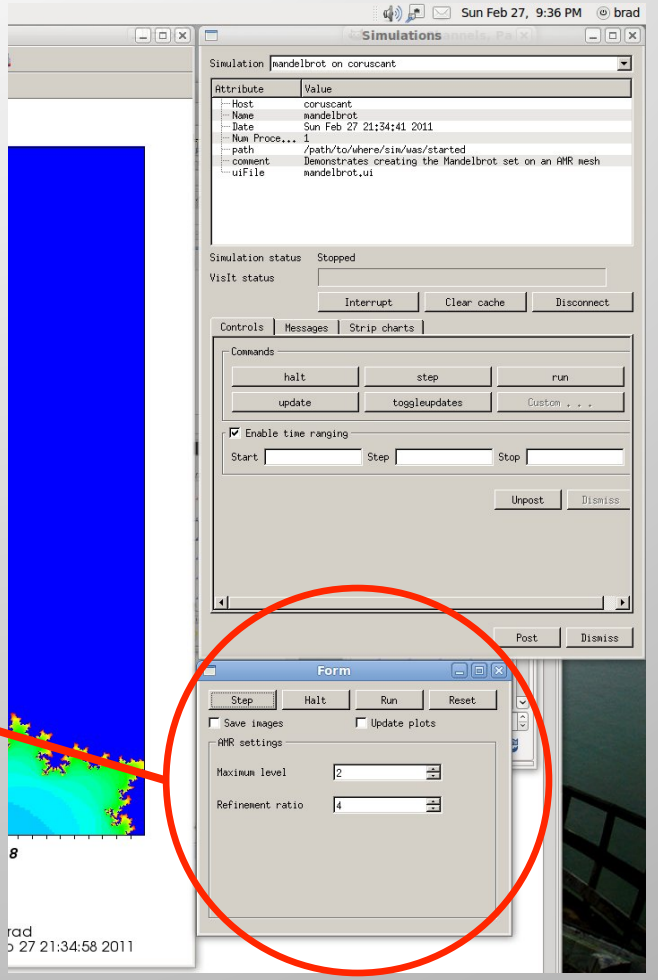
- Connect and disconnect at any time while the simulation is running
- “Halt”, “Step”, “Run”, “Update”, “others…”



Custom User Interfaces

- **Simulation can provide UI description for more advanced computational steering**





Details on the instrumentation

- The C and Fortran interfaces for using libsim 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()`
  - `GetVariable()`, etc

Main program

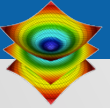
```
int main(int argc, char **argv)
{
    SimulationArguments(argc, argv);



    read_input_deck();
    mainloop();
}
```

Main program is augmented

```
int main(int argc, char **argv)
{
    SimulationArguments(argc, argv);
    VisItSetupEnvironment();
    VisItInitializeSocketAndDumpSimFile();
    read_input_deck();
    mainloop();
}
```

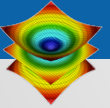


mainloop() will be augmented

```
void mainloop(simulation_data *sim)
{
    int blocking, visitstate, err = 0;
    do
    {

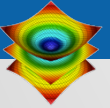
    simulate_one_timestep(sim);

    } while(!sim->done && err == 0);
}
```



main loop is augmented

```
void mainloop(simulation_data *sim)
{
    do
    {
    blocking = (sim->runMode == SIM_RUNNING) ? 0 : 1;
    visitstate = VisItDetectInput (blocking, -1);
    if(visitstate == 0) {
       simulate_one_timestep(sim);
       }
}
```



simulate_one_timestep() remains untouched

```
if (visitstate == 0) {

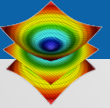
        simulate_one_timestep(sim);

        }
```

This call should also be provided as an entry point from the controlling GUI to single-step into the execution

Simulation main loop

```
else if (visitstate == 1) {
  if (VisItAttemptToCompleteConnection() == VISIT_OKAY)
  {
      sim->runMode = SIM_STOPPED;
      VisItSetCommandCallback(ControlCallback, sim);
      VisItSetGetMetaData(SimGetMetaData, sim);
      VisItSetGetMesh(SimGetMesh, sim);
  }
}
```



Simulation main loop

```
else if (visitstate == 2) {
    if(VisItProcessEngineCommand() == VISIT_ERROR)
    {
   /* Disconnect on an error or closed connection. */
        VisItDisconnect();
   /* Start running again if VisIt closes. */
        sim->runMode = SIM_RUNNING;
     }
  }
} while(!sim->done && err == 0);
}
```

Callbacks are added to advertize the data

`visitcommandcallback()`

`visitgetmetadata()`

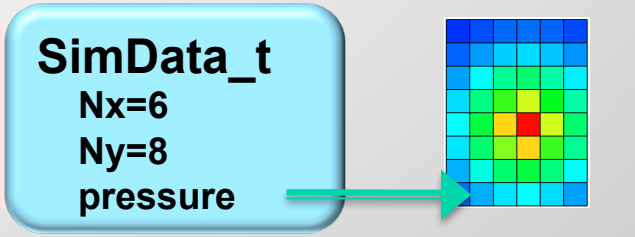
- Mesh name
- Mesh type
- Topological and spatial dimensions
- Units, labels
- Variable names and location (cell-based, node-based)
- Variable size (scalar, vector, tensor)
- Commands which will be understood (next(), halt(), run(), …)

`visitsimgetmesh(), visitgetvariable()`

Sharing Data Example

```
// Example Data Access Callback
visit_handle
GetVariable(int domain, char *name,
void *cbdata)
{
    visit_handle h = VISIT_INVALID_HANDLE;
    SimData_t *sim = (SimData_t *)cbdata;
    if(strcmp(name, "pressure") == 0)
    {
        VisIt_VariableData_alloc(&h);
        VisIt_VariableData_setDataD(h,
            VISIT_OWNER_SIM,
            1, sim->nx*sim->ny,
            sim->pressure);
    }
    return h;
}
```

Pass simulation buffer to Libsim



Updating plots

- Call `VisItTimeStepChanged()` when you want to tell VisIt that the simulation time step has changed
  - This causes your `GetMetaData` callback to be called and new metadata to be sent to VisIt
- Call `VisItUpdatePlots()` when you want to update VisIt's plots
  - An update message gets sent to VisIt asking it to update its plots
  - The update message causes all of the plots to be re-executed
  - This causes new data to be requested from the simulation via the data access code callbacks

Parallel considerations

- Libsim can instrument parallel simulations too
  - See updateplots.c example in VisIt source code
- More setup in your main function after `VisItSetupEnvironment`
  - Call `VisItSetParallel, VisItSetParallelRank`
  - Provide broadcast callbacks that let VisIt use your MPI implementation to broadcast integers and strings
- Only rank 0 calls `VisItInitializeSocketAndDumpSimFile`
- `VisItDetectInput` must be called only on rank 0 so its results must be broadcast to the other processors
- When rank 0 calls `VisItProcessEngineCommand`, it broadcasts the results to the other processors

The people behind Libsim

- Jeremy and Brad were the initial developers
- Brad is now leading the effort

Brad Whitlock
Lawrence Livermore National Laboratory

Jeremy S. Meredith
Oak Ridge National Laboratory

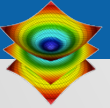
http://portal.nersc.gov/svn/visit/trunk/docs/Presentations/EGPGV2011_InSituPaper.pdf

Summary of *in situ* visualization

Parallel visualization is very mature and can run like a supercomputer simulation; likewise, it is too often limited by I/O.

By coupling Simulations and Visualization, we inherit all the development of a mature visualization app. such as VisIt, to perform *in situ* visualization.

<Live demonstration>



Wrap-up and Questions

Thank you for attending!

Tutorial Survey: http://sc12.supercomputing.org/survey

- Presenters:
  - Cyrus Harrison — cyrush@llnl.gov
  - Jean Favre — jfavre@cscs.ch
  - Hank Childs — hchilds@lbl.gov
  - Dave Pugmire — pugmire@ornl.gov
  - Brad Whitlock — whitlock2@llnl.gov
  - Hari Krishnan — hkrishnan@lbl.gov
- User resources:
  - Main website: http://www.llnl.gov/visit
  - Wiki: http://www.visitusers.org
  - Email: visitusers@ornl.gov
- Development resources:
  - Email: visit-developers@ornl.gov
  - SVN: http://portal.nersc.gov/svn/visit