CSCS

Swiss National Supercomputing Centre

In-situ Visualization
Computational Steering

Dr. Jean M. Favre

Scientific Computing Research Group

30-09-2011

Outline

- *In-situ* visualization with VisIt's libsim
  - Examples for all grid types
  - The 2D Jacobi solver with parallel partitioning
  - Source code instrumentation
  - Specify ghost-nodes
  - Single stepping through the execution
  - Debugging

- Computational Steering
  - The mandelbot example
  - A custom GUI widget

In-situ Visualization - Motivations

Having a real-time monitoring capability on all supercomputing resources is essential **to avoid wasting valuable time on computational resources**…

Techniques such as **in situ analysis and online data reduction** and transformation for **reducing the demands on the storage system** must be pursued…

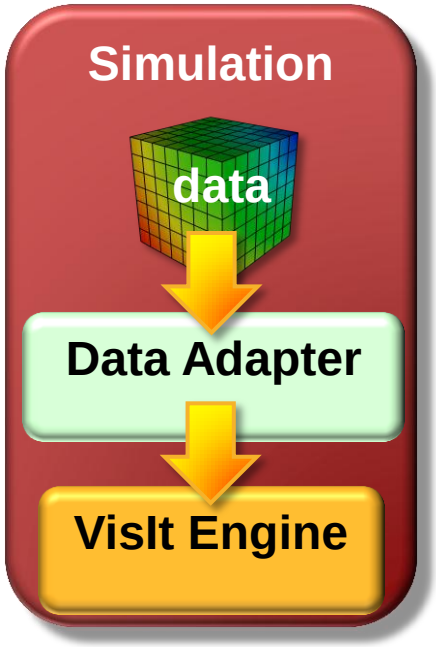
Run-time reduction of the raw simulation data before visualization and interactive discovery will be routine…

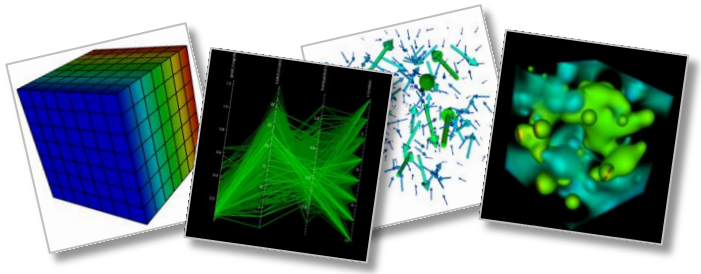
In-situ Visualization with libsim

- The libsim allows the visualization of simulation data in situ to avoid the high costs of I/O associated with writing and then reading the data
- Simulation codes are instrumented (source code is added) to create an interface to the full feature set of VisIt
- Libsim implements a tight-coupling, sharing the memory space and the execution thread of the simulation

In-situ Visualization

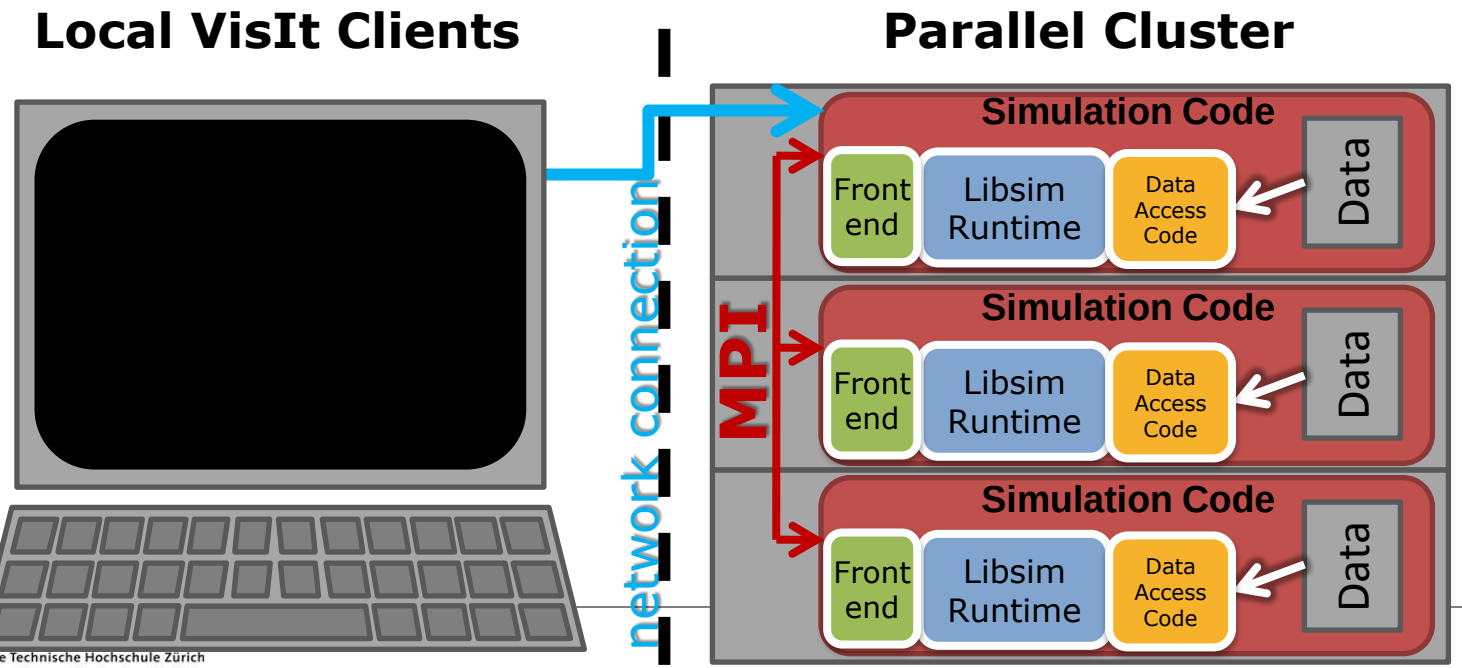
- Simulations use a data adapter layer to make the data suitable for VisIt's engine process
- Operate directly on the simulation's data arrays when possible





In-situ Visualization

- Front end library lets VisIt connect to the simulation
- The GUI client requests data on demand
- VisIt's engine gains access to the data through user-supplied *Data Access Code* callback functions



Code instrumentation

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.

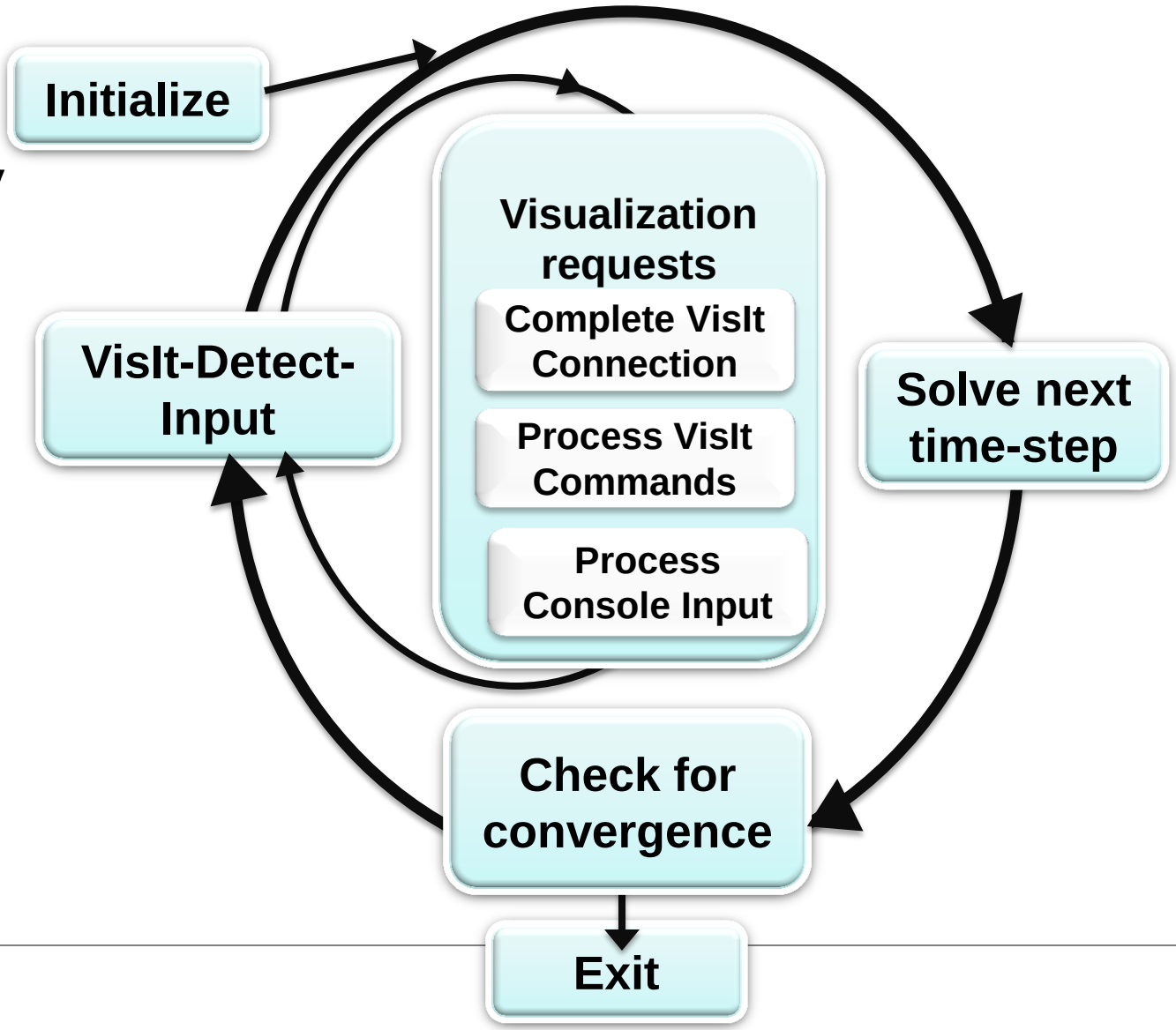
Add control functions that let VisIt steer the simulation.

1) Adapt the main iterative loop

Connection to the visualization library is optional

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

Live connection can be closed and re-opened later



2) Create *data access callback*

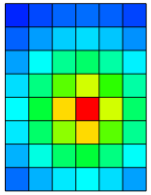
```
GetMetaData(void *cbdata) {
simulation_data *sim = (simulation_data *) cbdata;
VisIt_SimulationMetaData_setCycleTime(
                                        md, sim->cycle, sim->time);
VisIt_MeshMetaData_setName(mmd, "mesh2d");
VisIt_MeshMetaData_setMeshType(mmd,
                                  VISIT_MESHTYPE_RECTILINEAR);
VisIt_MeshMetaData_setTopologicalDimension(mmd, 2);
VisIt_MeshMetaData_setSpatialDimension(mmd, 2);
VisIt_MeshMetaData_setNumDomains(mmd, sim->par_size);
```

grid mesh

```
VisIt_VariableMetaData_setName(vmd, "pressure");
VisIt_VariableMetaData_setMeshName(vmd, "mesh2d");
VisIt_VariableMetaData_setType(vmd,
                                    VISIT_VARTYPE_SCALAR);
VisIt_VariableMetaData_setCentering(vmd,
                                  VISIT_VARCENTERING_ZONE);
VisIt_SimulationMetaData_addVariable(md, vmd); }
```

data fields

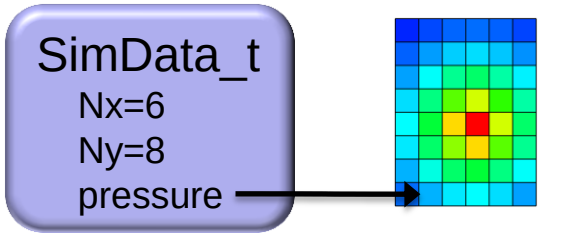
Simulation Buffer



2) Create *data access callback*

```
// 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;
}
```

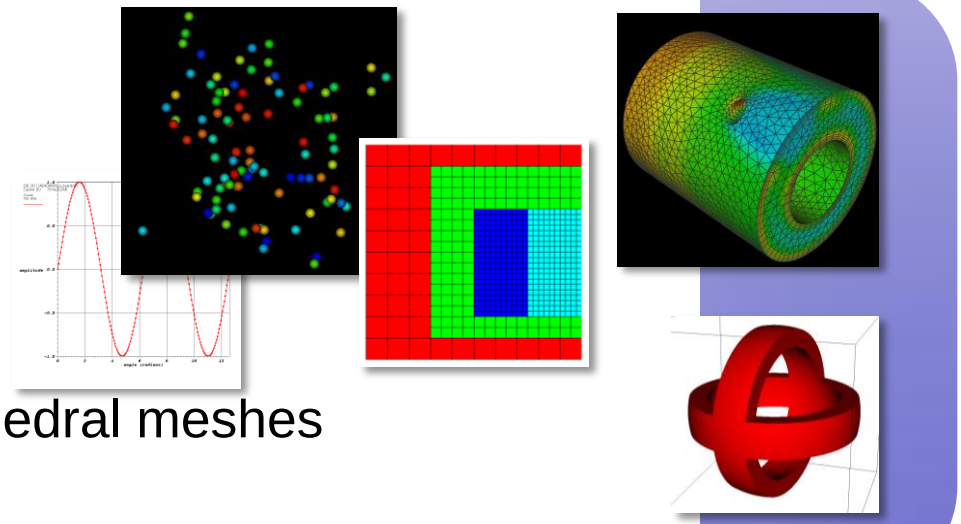
Simulation Buffer



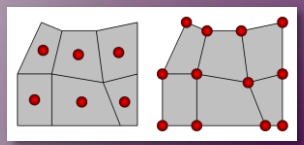
Pass simulation buffer to Libsim

2) *Data access enable all mesh types*

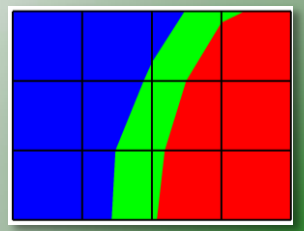
- Mesh Types
  - Structured meshes
  - Point meshes
  - CSG meshes
  - AMR meshes
  - Unstructured & Polyhedral meshes



- Variables
  - 1 to N components
  - Zonal and Nodal

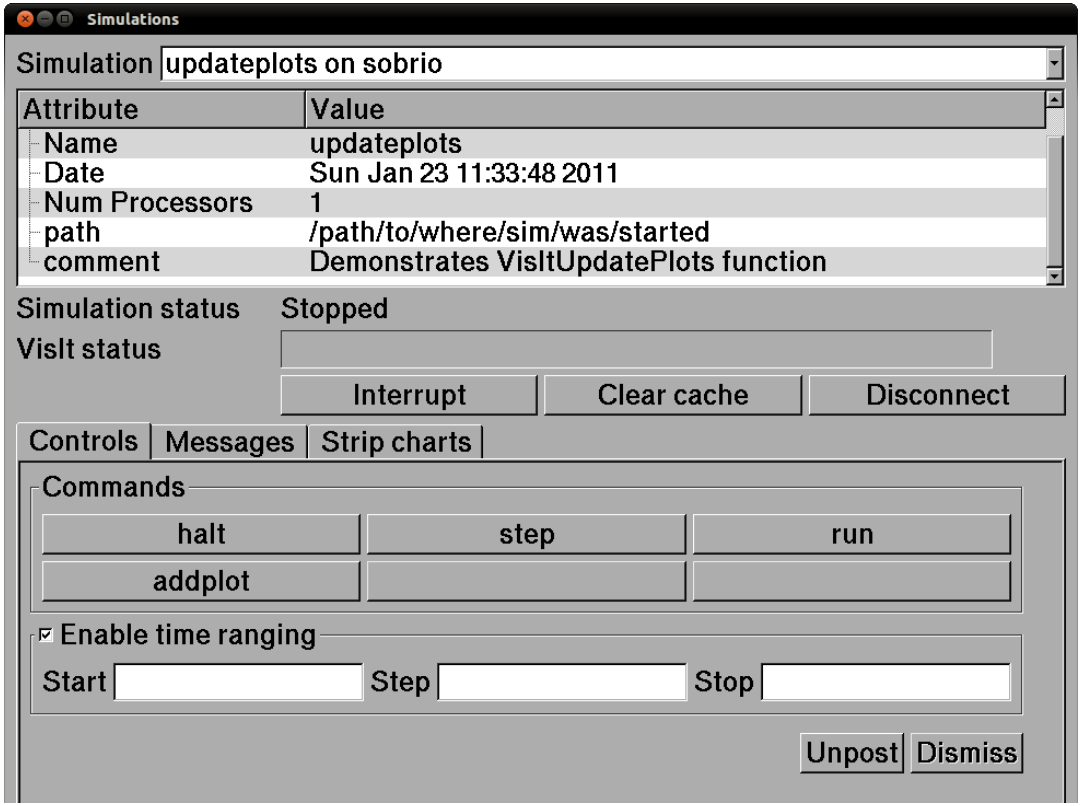


- Materials
- Species



3) Add control functions for steering

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



Compilation details

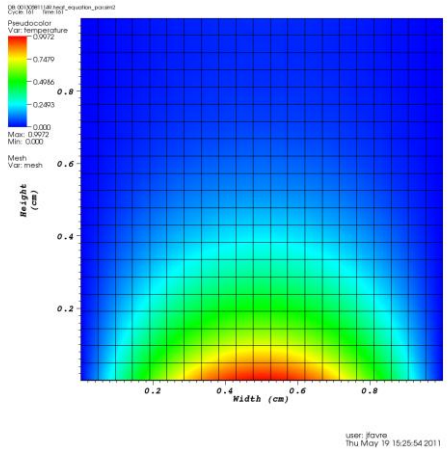
- Run ccmake and check:
- VISIT_DATA_MANUAL_EXAMPLES = ON
- If compiling with VISIT_FORTRAN = ON, the silo plugin will give you problems =>
- Reconfigure and install silo without the –disable-fortran option

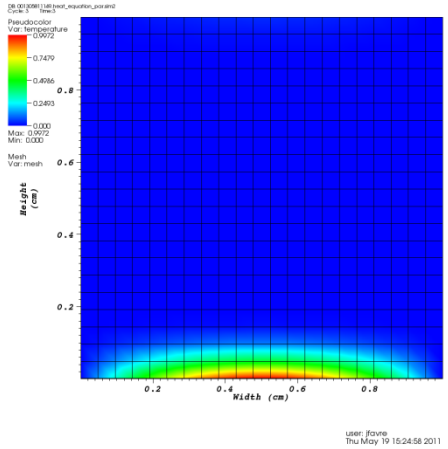
Solving a PDE and visualizing the execution

Full source code solution is given here:

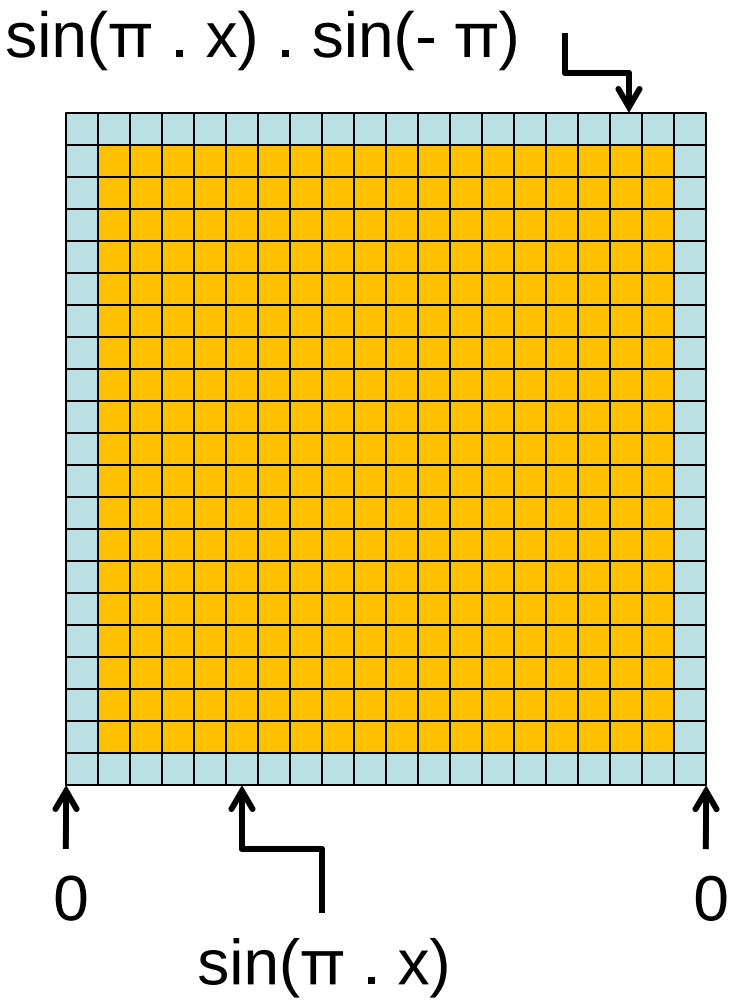
- http://portal.nersc.gov/svn/visit/trunk/src/tools/DataManualExamples/Simulations/contrib/pjacobi/

- C, F90 and Python subdirectories





A PDE with fixed boundary conditions



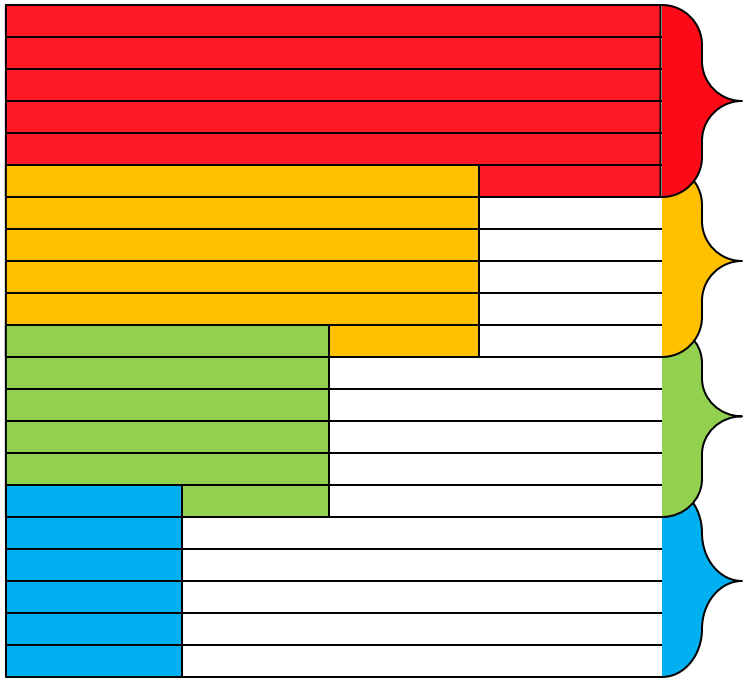
Update grid with solver

Fixed boundary conditions

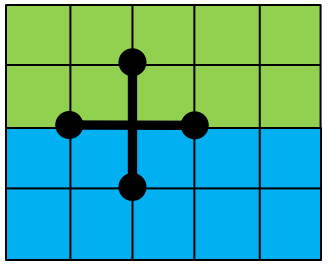
Laplace equation: $\Delta u = 0$

Total grid size is (m+2)*(m+2)

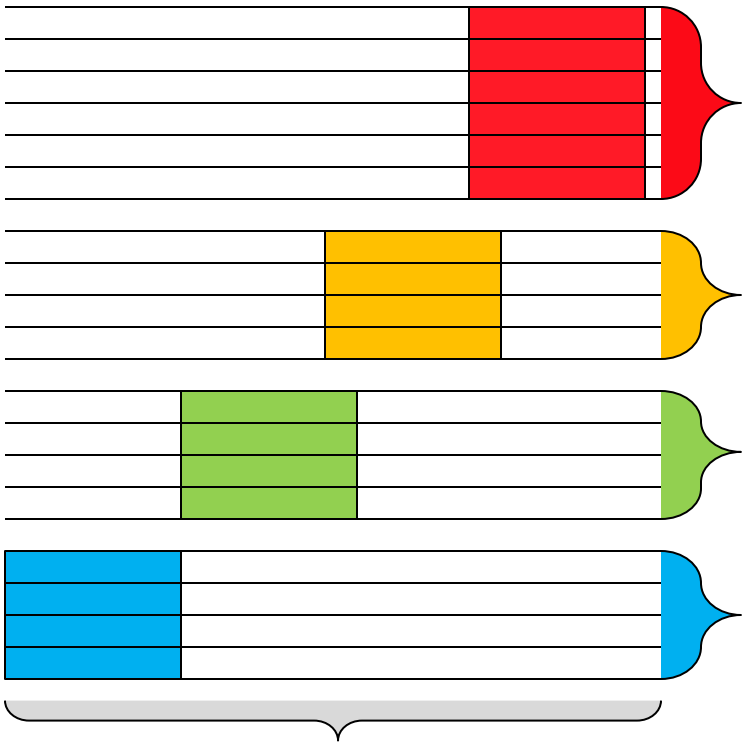
2D grid partitioning and initialization



- The grid is partitioned along the Y direction
- Locall grid size is (m+2)*(mp+2)
- Boundary conditions are set
- A single line of ghost-nodes insure that the 5-point stencil is continuous across MPI task boundaries



I/O patterns



(m+2) columns to write

Check-pointing and restart

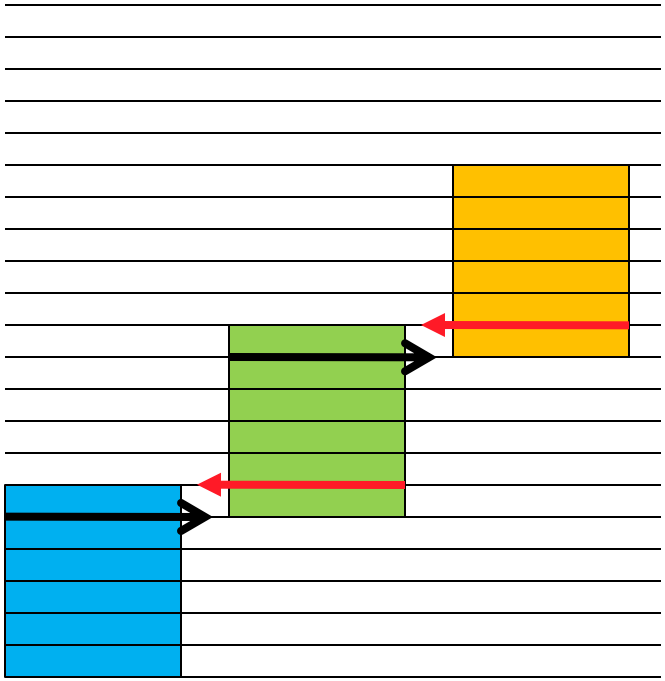
(mp+2) grid lines to read/write

(mp) grid lines to read/write

(mp) grid lines to read/write

(mp) grid lines to read/write

Ghost data exchange

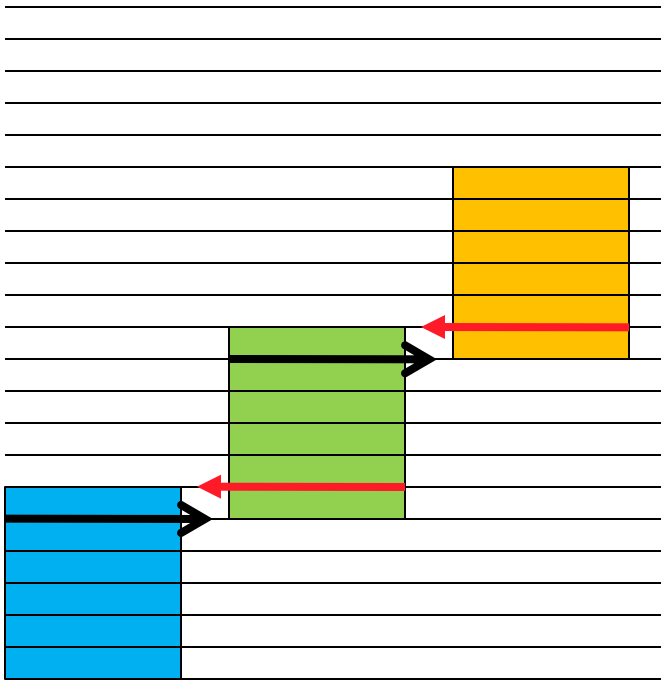


Overlap Send and Receive

Proc. 0 does not receive from “below”

Proc. (N-1) does not send “above”

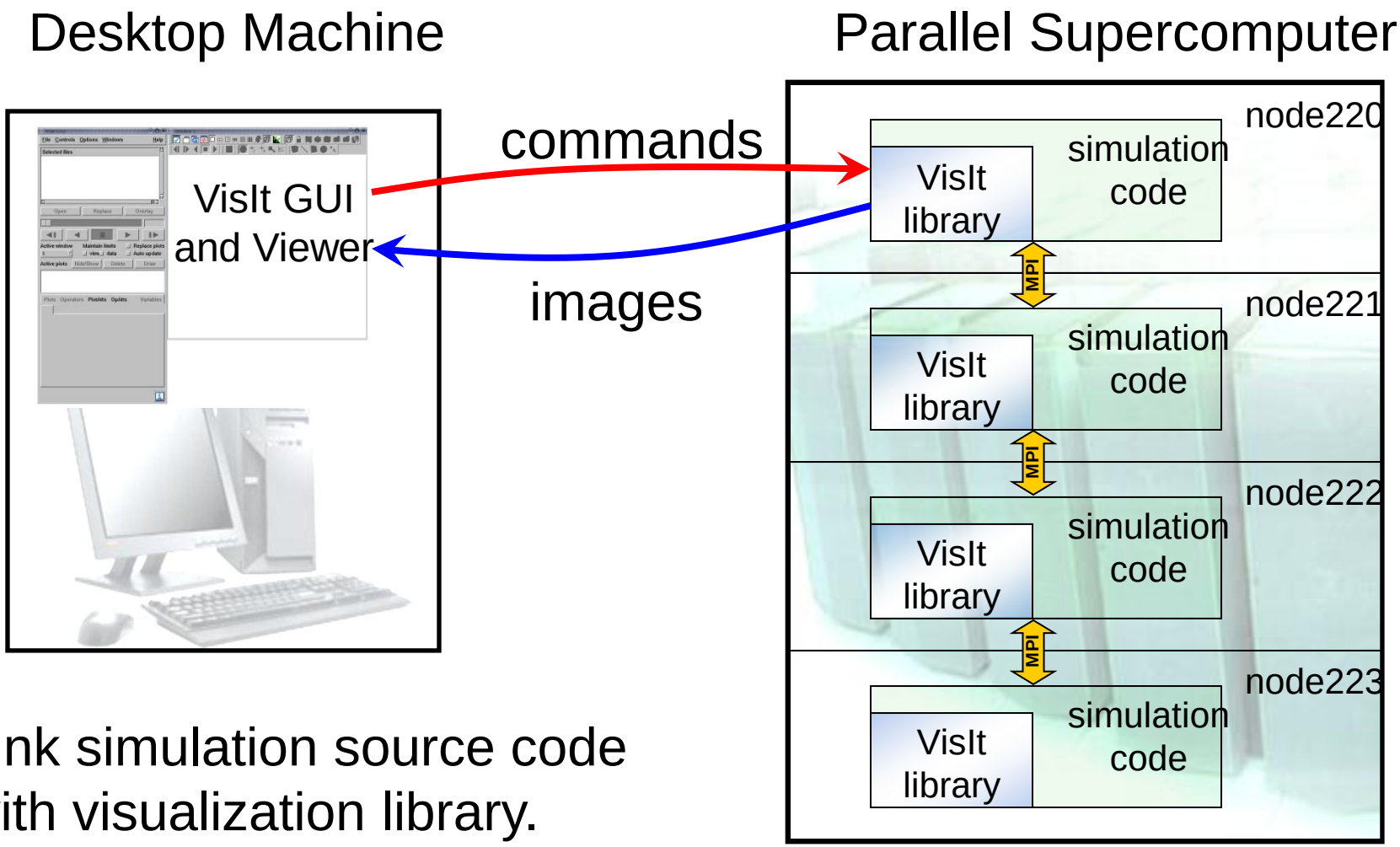
Iteration tasks for one timestep



1. oldTemp <= Temp
2. Execute stencil operation to evaluate new temperature array “Temp”
3. Exchange ghost data in Temp
4. Visualize Temp

“oldTemp” is the old timestep
“Temp” is the new timestep

VisIt's libsim implements a tight coupling



- Link simulation source code with visualization library.
- Data is shared via pointers.

The source code needs to be instrumented

1. The execution flow needs to check for Visualization Requests

2. Once connected, the simulation code needs to advertize what data/meshes are available, and

3. Provide pointers to data, or wrap data into the expected form/shape

Source code examples are instrumented with:

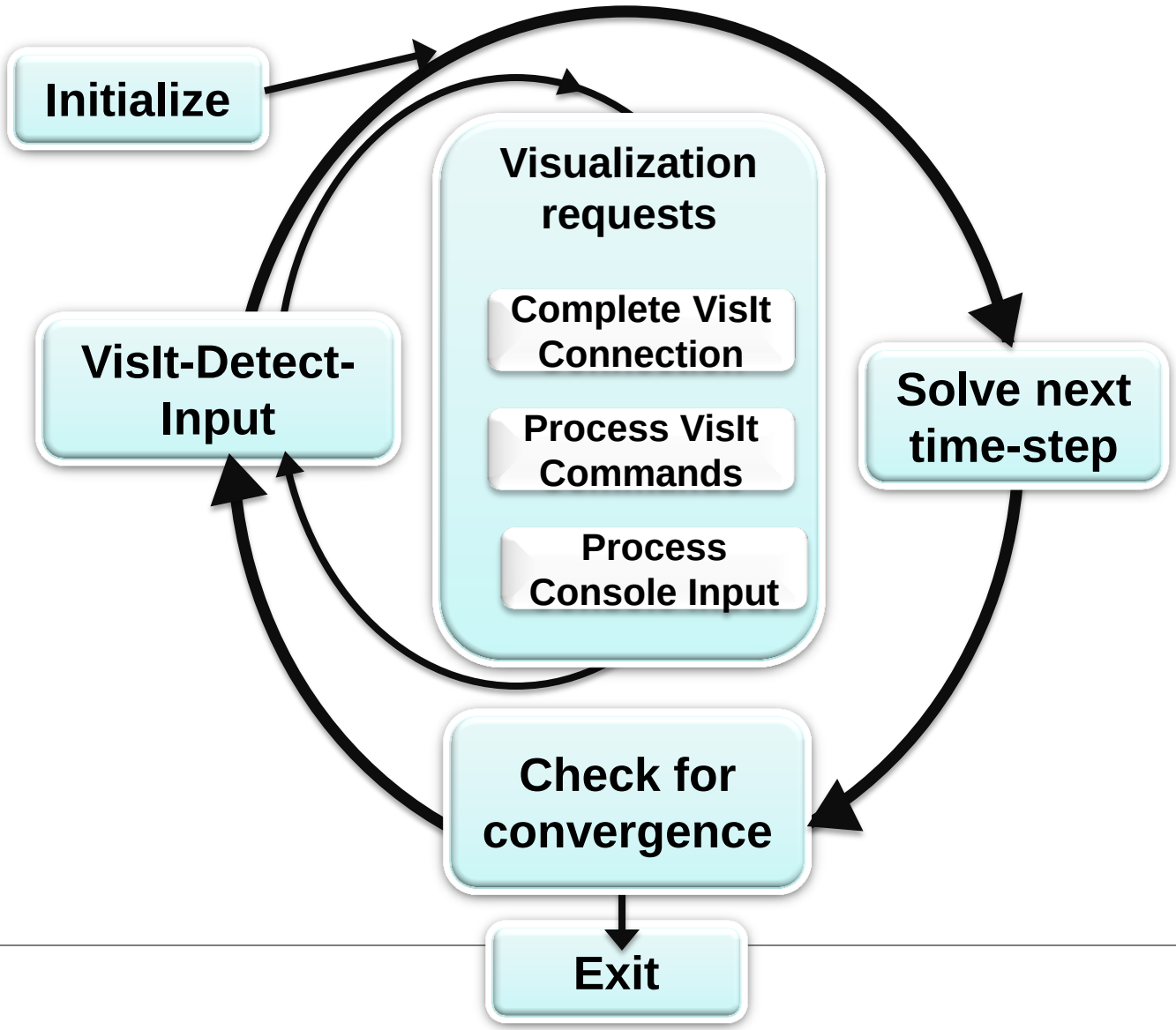
```
#ifdef _VISIT_
#endif
```

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



Step-by-step or continuous execution

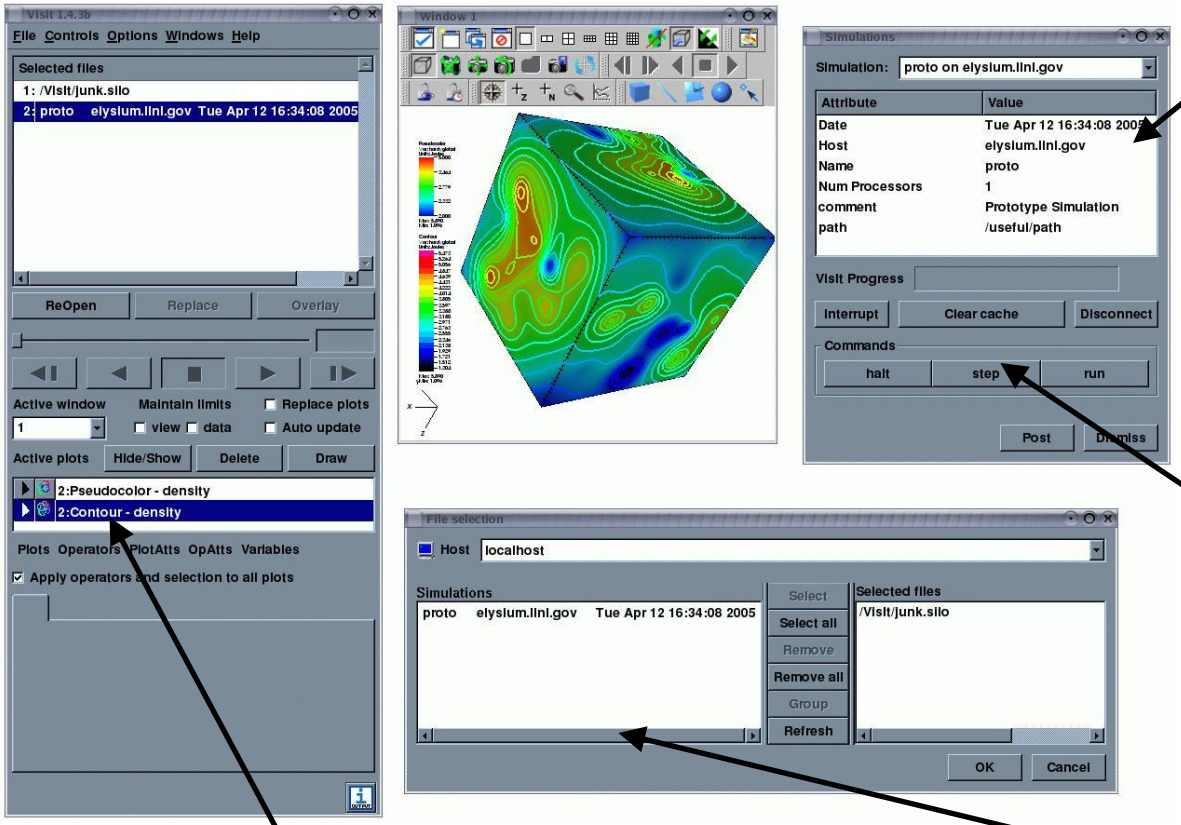
- A simulation would normally not wait for a connection and execute as fast as possible.

These examples however, pause immediately, so they won't finish before you have time to connect!

The call visitdetectinput(bool blocking, -1) instructs the simulation to wait for a connection at init time.

The examples also block after each timestep so you have time to request multiple plots.

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



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

Control commands exposed by the code are available here

Users select simulations to open as if they were files

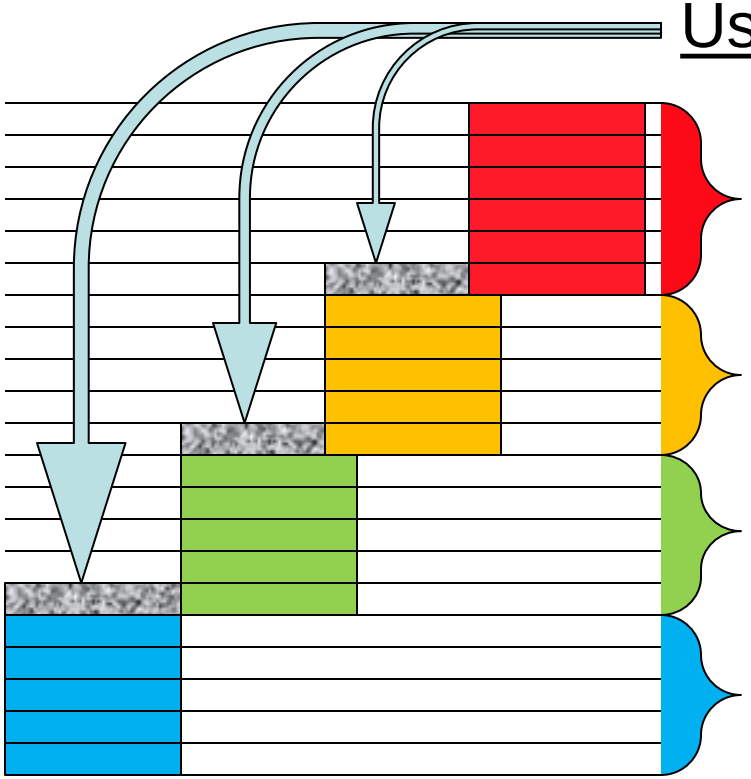
All of VisIt's existing functionality is accessible

Data sharing

- The VisIt Data API has just a few callbacks
  - GetMetaData()
  - GetMesh()
  - GetScalar(), GetVector()

  - Each MPI rank provides the full mesh/data (with ghost regions) marked in a way similar to HDF5 hyperslabs or MPI_Type_create_subarray().

grid mesh for *in situ* graphics



Use ghost-nodes to prevent overlaps

(mp+2) lines to send

(mp+1) lines to send

(mp+1) lines to send

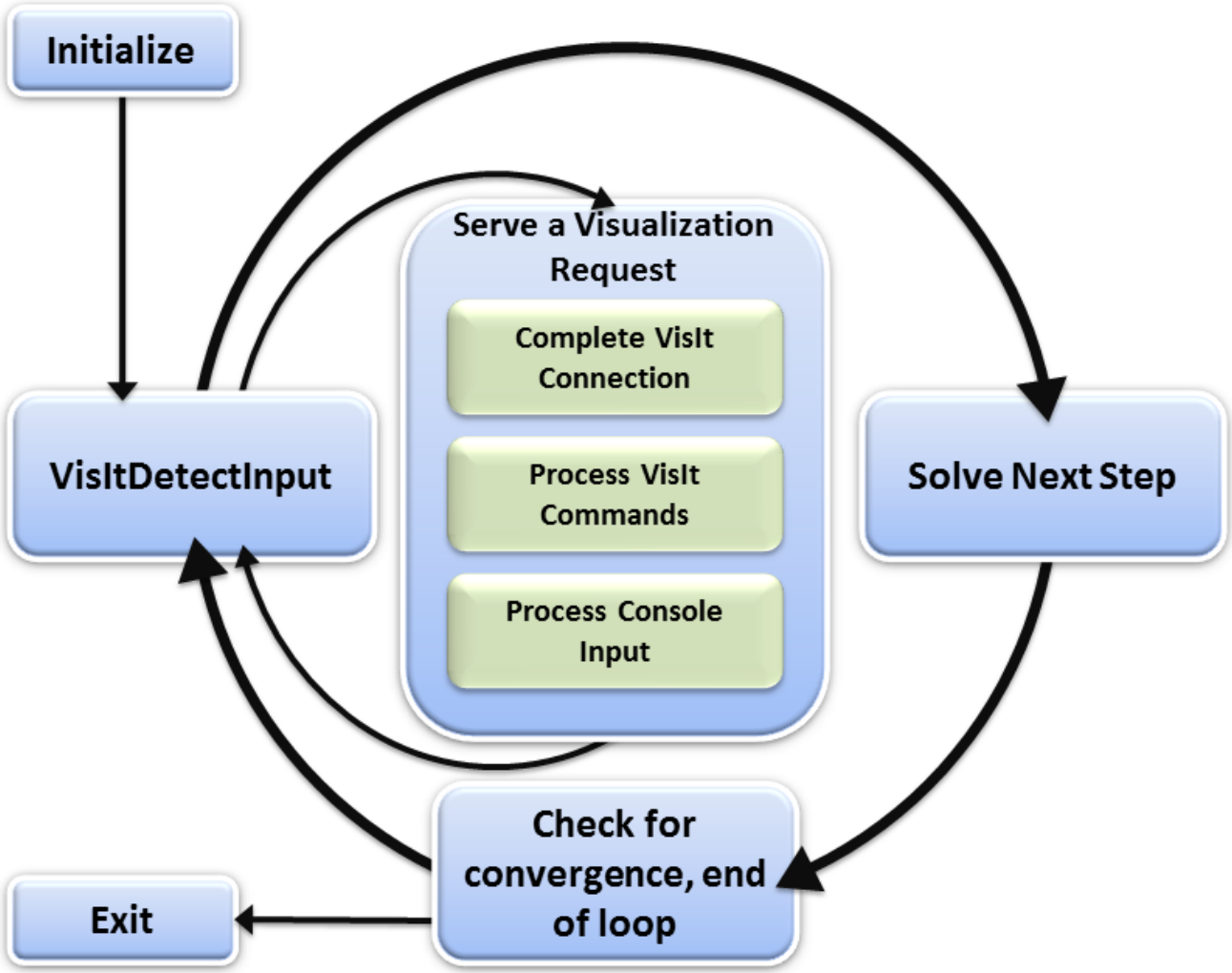
(mp+1) lines to send

VisitRectMeshSetRealIndices(h, minRealIndex, maxRealIndex)

At least two entry points to the execution

Execution of the next step can be triggered by:

- normal program flow,
- on-demand single-stepping from the GUI,
- console input.



Exercise 1.1

Compiler et executer une simulation pjacobi, en C ou en F90

Avec sortie fichier (use BOV reader):

mpiexec –n 4 pjacobi

Avec couplage in-situ:

mpiexec –n 4 pjacobi_visit
*cd $HOME/.visit/simulations and ouvrez le fichier *sim2*
Visualiser la variable Temperature

Exercise 1.2

Faite une copie (-r) dans votre repertoire prive

/softs/VisIt/Exercise/pjacobi

Lire le fichier README.txt

rm /tmp/Jacobi.bin, execute pjacobi, visualize the result using the BOV reader

https://wci.llnl.gov/codes/visit/2.0.0/GettingDataIntoVisIt2.0.0.pdf

Page 9-12

Exercise 1.3

A quelle iteration sommes nous?

Visualizer les ghost-zones, le Processor Id

Visualizer l'iteration courante ET celle d'avant.

Creez une expression pour visualizer la difference entre les 2 pas de temps.

Vous devez ecrire du code supplementaire.

Exercise 2: Computational Steering

Lancer la demo mandelbrot in

build_directory/tools/DataManualExamples/Simulations

Se connecter, et utiliser la ligne de commande pour:

Changer la resolution du maillage
Avancer, step-by-step

At the console: command> help	Open the custom widget from the GUI panel

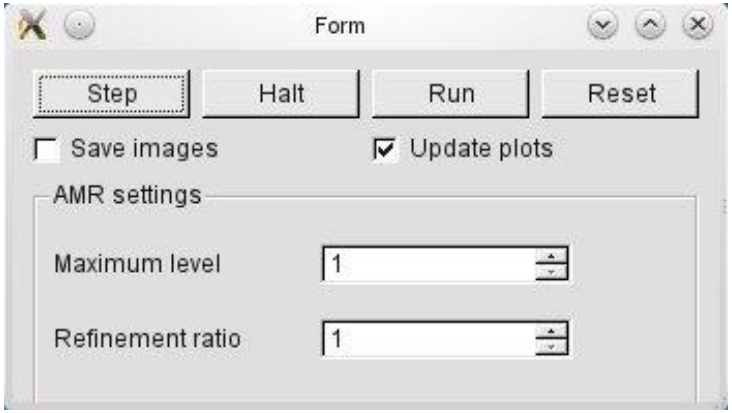
Exercise 2: Computational Steering

Copier ce fichier dans $HOME/.visit/ui

http://portal.nersc.gov/svn/visit/trunk/src/tools/DataManualExamples/Simulations/mandelbrot.ui

The main loop (see mandelbrot.C) defines which callbacks to associate with the button's actions (clicked(), valueChanged(), etc…)

The programmer writes these callbacks to modify the simulation's parameters.



The merits of libsim

- The greatest bottleneck (disk I/O) can be eliminated
- Not restricted by limitations of any file format
- No need to reconstruct ghost-cells from archived data
- All time steps are potentially accessible
- All problem variables can be visualized
- Internal data arrays can be exposed or used
- Step-by-step execution will help you debug your code and your communication patterns

- Custom GUI widgets enable computational steering

The end

Brad Whitlock (LLNL), Jeremy Meredith (ORNL), Hank Childs (NERSC) have contributed greatly.

Reading:

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

Merci de votre attention