BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

SC18 *In Situ* Analysis and Visualization with SENSEI

E. Wes Bethel, Junmin Gu, Burlen Loring, Dmitriy Morozov, *Gunther H. Weber*, John Wu (LBNL). *Nicola Ferrier*, Silvio Rizzi (ANL). Dave Pugmire, James Cress, Matthew Wolf (ORNL). Earl Duque, Brad Whitlock (Intelligent Light). Utkarsh Ayachit, David Thompson, Andrew Bauer, Patrick O'Leary (Kitware)

BERKELEY LAB

Argonne NATIONAL LABORATORY

OAK RIDGE National Laboratory

Intelligent Light

Kitware

Tutorial VM & web-site

The latest slides and VM can be obtained from after 5pm on Friday Nov

- www.sensei-insitu.org/tutorials/sc18.html

At the tutorial

- USB drive available which contains:
  - All demos shown here
  - A pdf of the slides for reference
    - Includes hidden slides with more details not covered here due to time restrictions

Outline

- Introduction to *In Situ* Analysis and Visualization
- SENSEI *In Situ* Data Interface
- Instrumenting data sources and endpoints (C++)
- SENSEI *In Situ* Demonstrations with Coupled Infrastructures
  - Data extracts with Libsim
  - Computational monitoring with ParaView Catalyst
  - Autocorrelation with ADIOS
  - Using SENSEI via Python
- *In Situ* Costs and Performance
- Closing thoughts

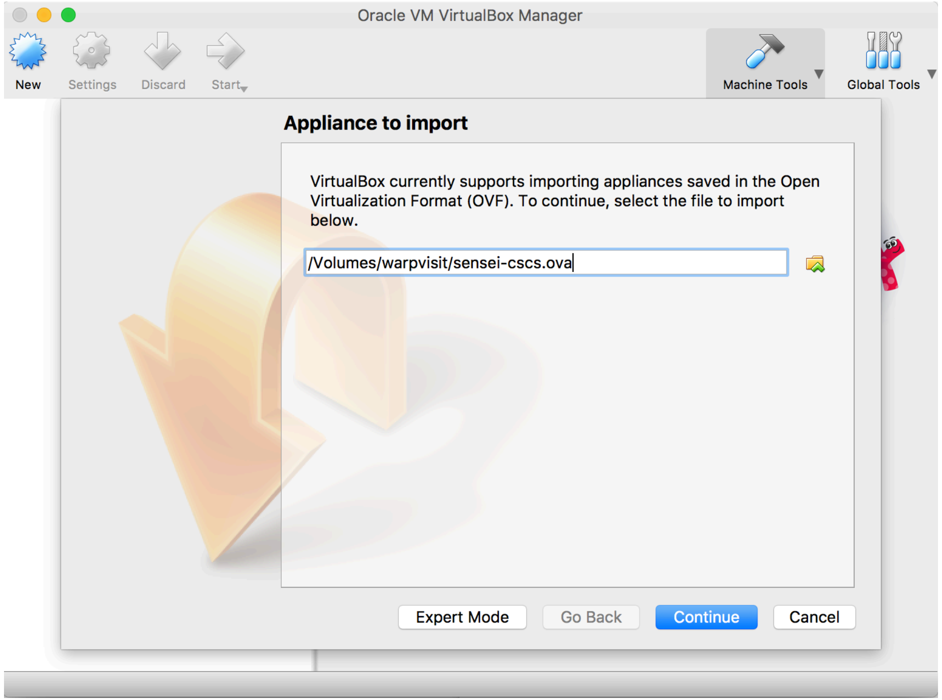
Setting up the VM

VirtualBox

- Download VirtualBox
- & VirtualBox extensions
- Update Guest additions in theVM, if your VirtualBox is not 5.2.16

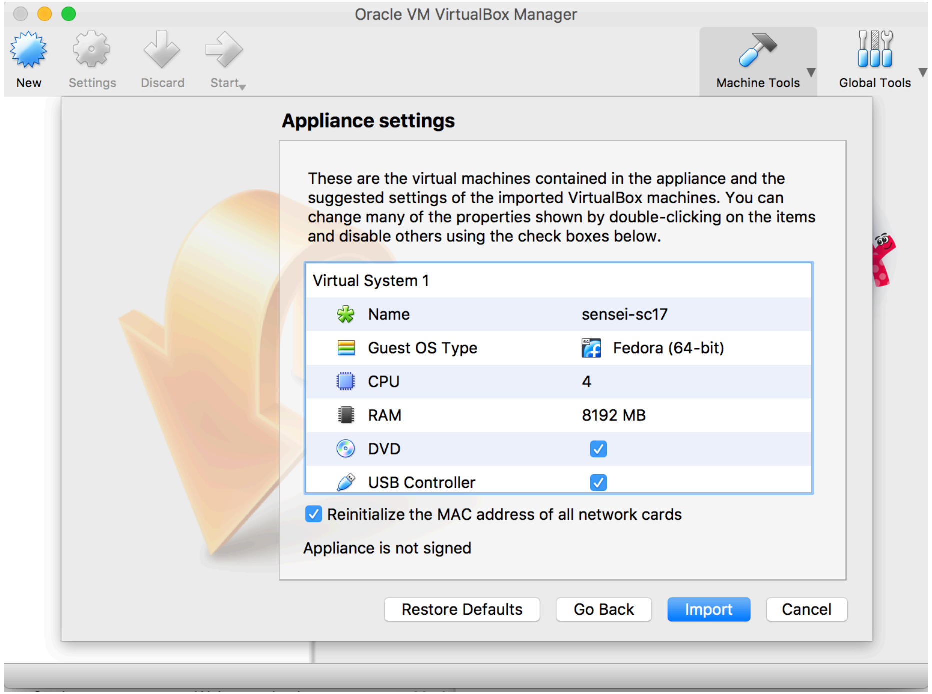
Import Appliance

- File->Import appliance
- locate sensei-sc18.ova



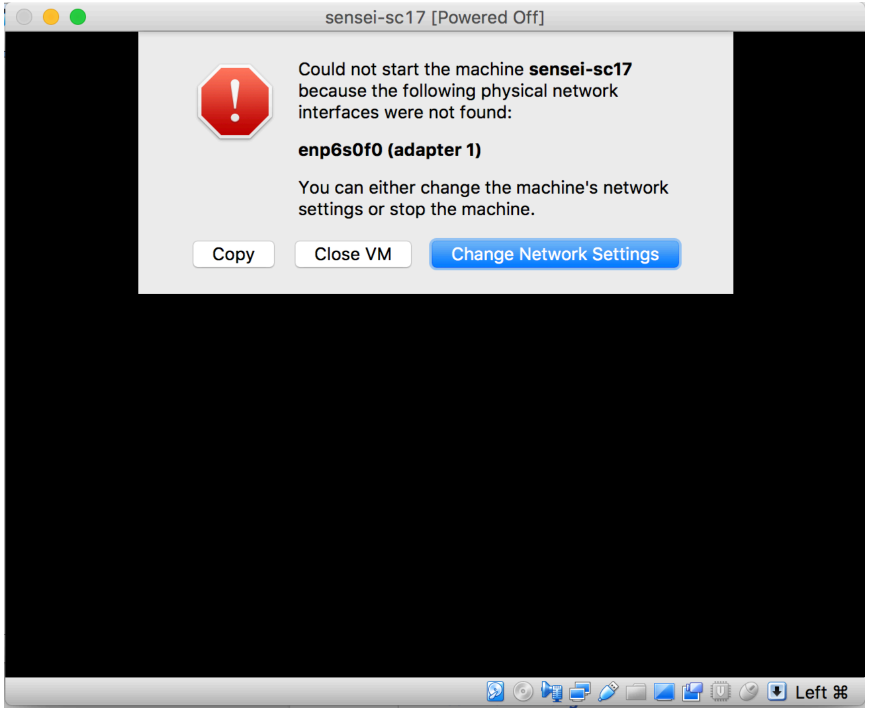
Import Appliance

- Check reinitialize mac address



Start the VM

- Start the VM
- Change network settings
- VirtualBox default should work



VM Layout

`~/sensei_insitu/software`

- ADIOS, ParaView, VisIt, VTK, SENSEI installs
- Use modules to select a SENSEI install. sensei/<version>-<backend>
  - `$ module load sensei/2.1.1-libsim`

`~/sensei_insitu/demos/sc18`

- demo codes and SENSEI miniapps used in the tutorial

VM access:
sensei
sc18_password

Getting started on cori

Demos on cori

- take an account, copy user name & password, cross it off the list and pass list on.
- fill out user agreement, turn them in before the break
- log in
  - `ssh -X <user name>@cori.nersc.gov`
- demos need to be run from scratch file system
  - `cd $SCRATCH`
  - `ln -s /project/projectdirs/m636/sensei_insitu $SCRATCH`

Starting jobs on cori

We have 40 nodes, one per account. to use the reservation add

```
--reservation=SC18_SENSEI
```

to salloc command

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

Introduction to in situ analysis

BERKELEY LAB · Argonne NATIONAL LABORATORY · OAK RIDGE National Laboratory · Intelligent Light · Kitware

Welcome! Why are we here?

Problem: FLOPS >> I/O, potential for lost science

Approach: do as much processing as possible while data still resident in memory?

Why This Tutorial? To inform you of issues involved, to show you what technologies are available and how to use them.

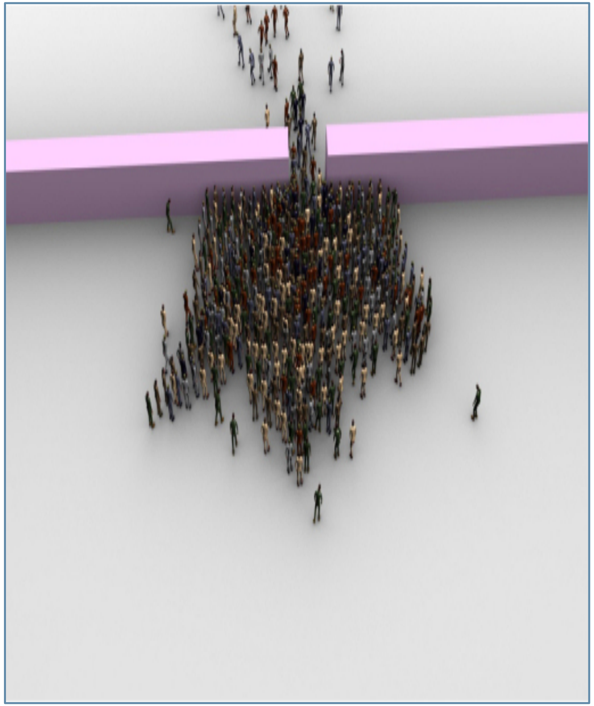
What are the problems?

Not enough I/O capacity on current HPC systems, and the trend is getting worse.

If there's not enough I/O, you can't write data to storage, so you can't analyze it: lost science.

Energy consumption: it costs a lot of power to write data to disk.

Opportunity for doing better science (analysis) when have access to full spatiotemporal resolution data.



Five orders of magnitude between compute and I/O capacity on Titan Cray system at ORNL

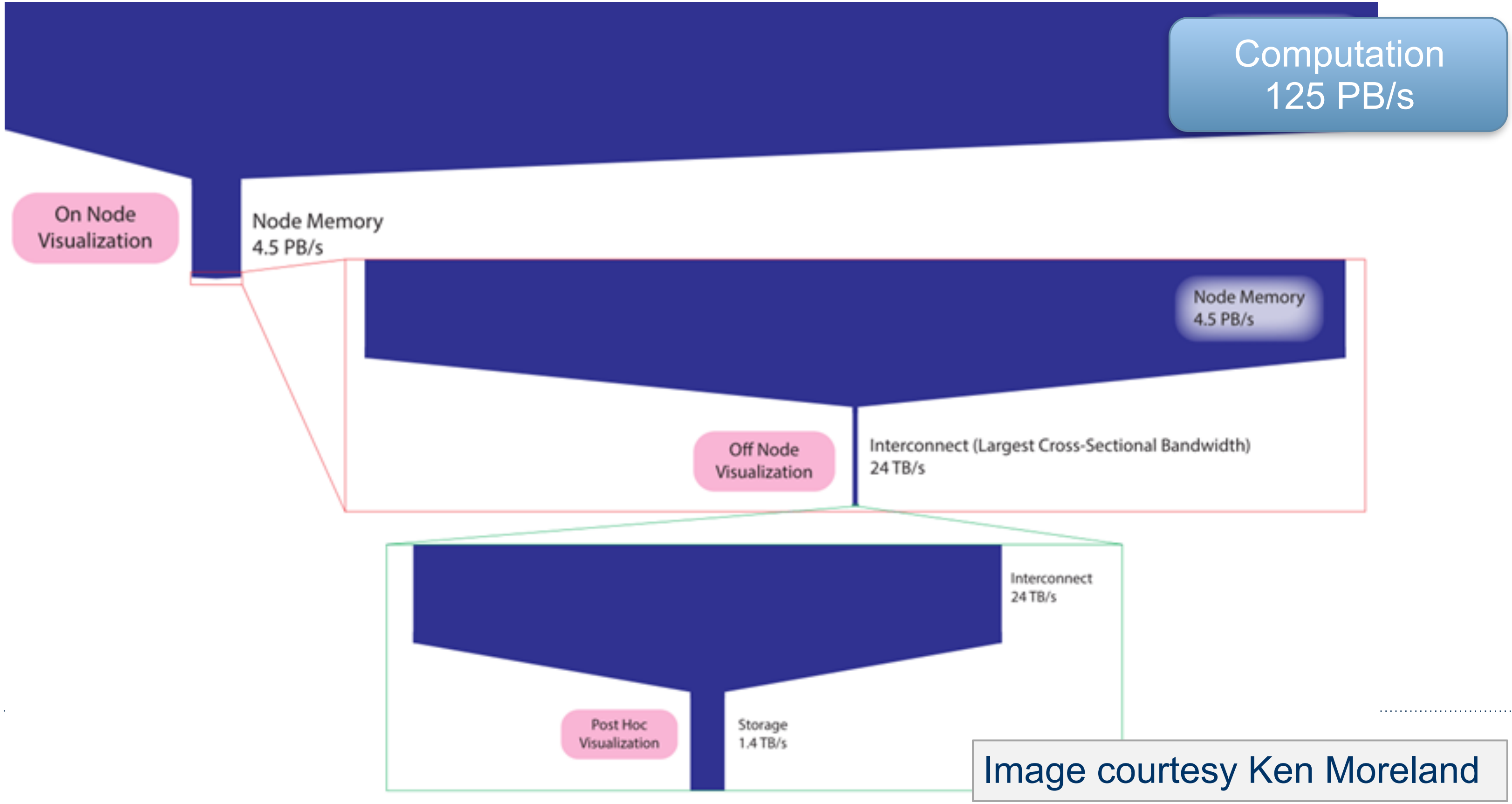


Image courtesy Ken Moreland

Five orders of magnitude between compute and I/O capacity on Titan Cray system at ORNL

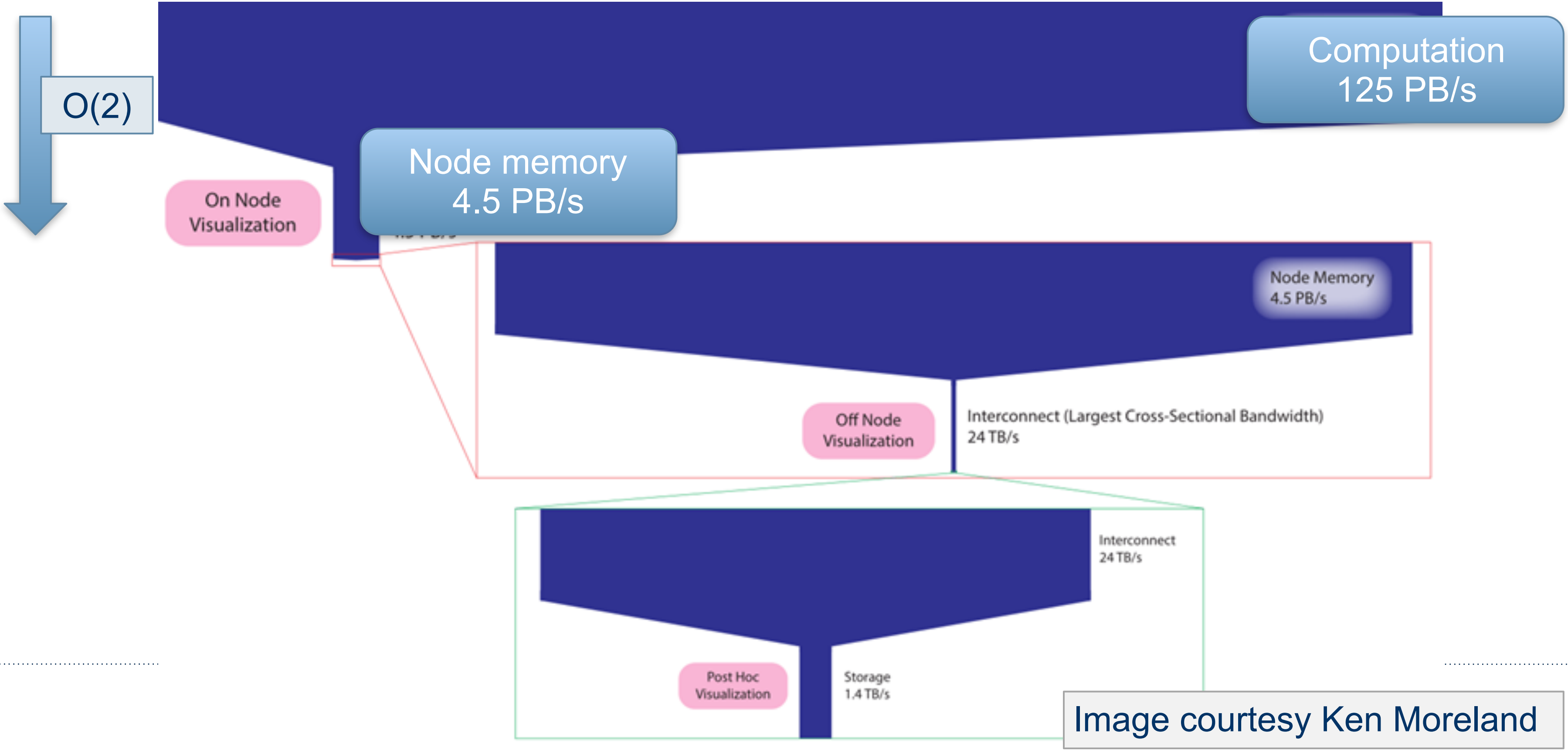


Image courtesy Ken Moreland

Five orders of magnitude between compute and I/O capacity on Titan Cray system at ORNL

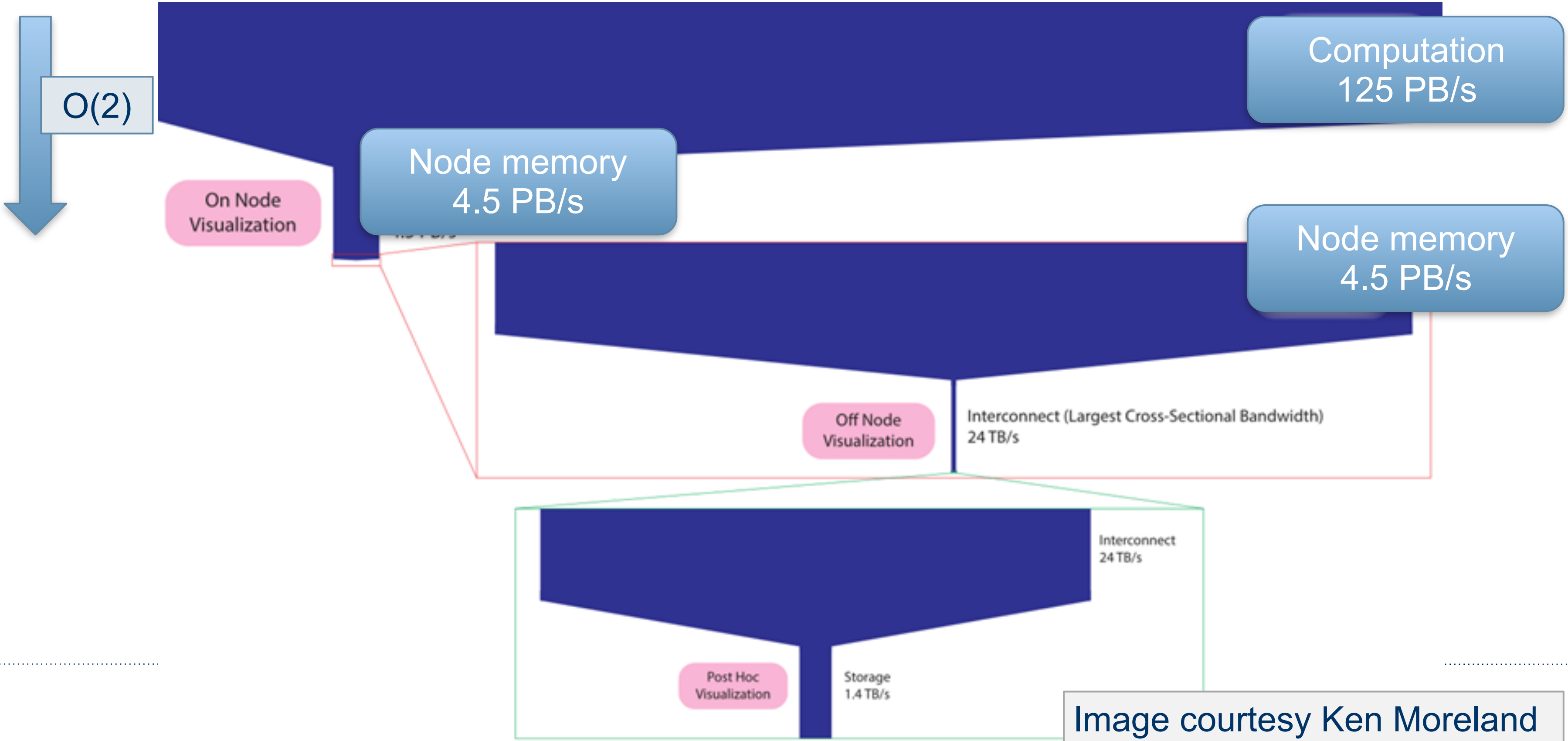


Image courtesy Ken Moreland

Five orders of magnitude between compute and I/O capacity on Titan Cray system at ORNL

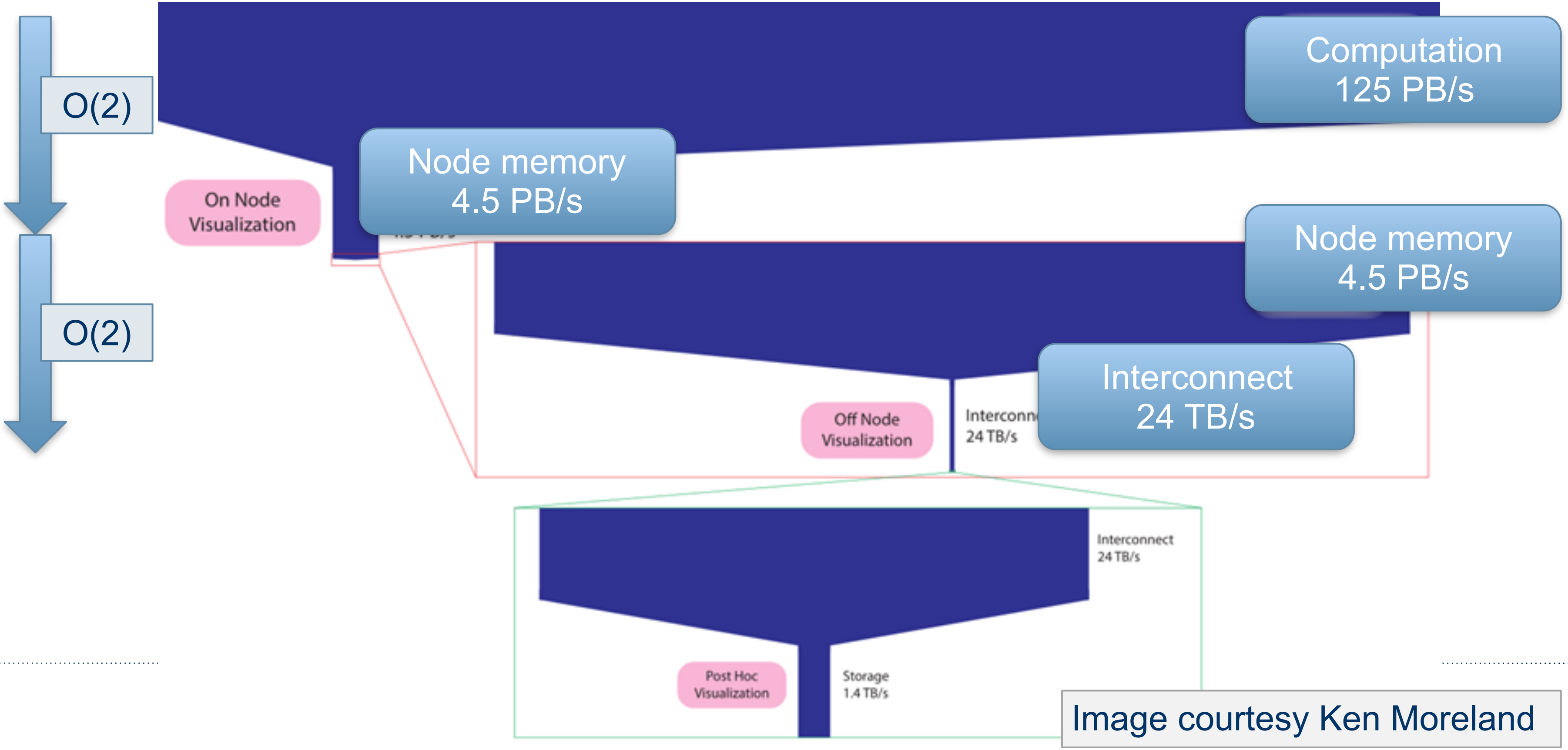


Image courtesy Ken Moreland

Five orders of magnitude between compute and I/O capacity on Titan Cray system at ORNL

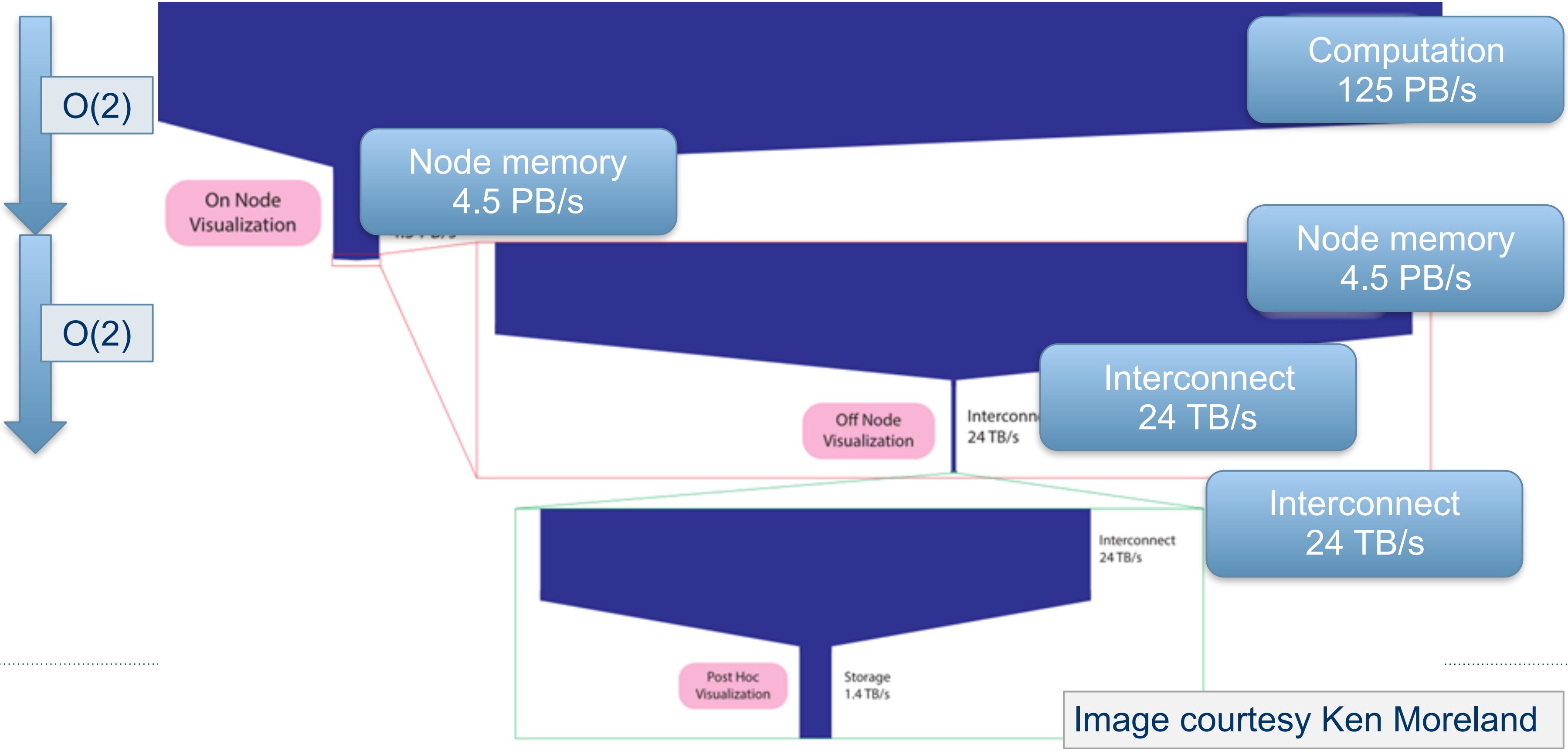


Image courtesy Ken Moreland

Five orders of magnitude between compute and I/O capacity on Titan Cray system at ORNL

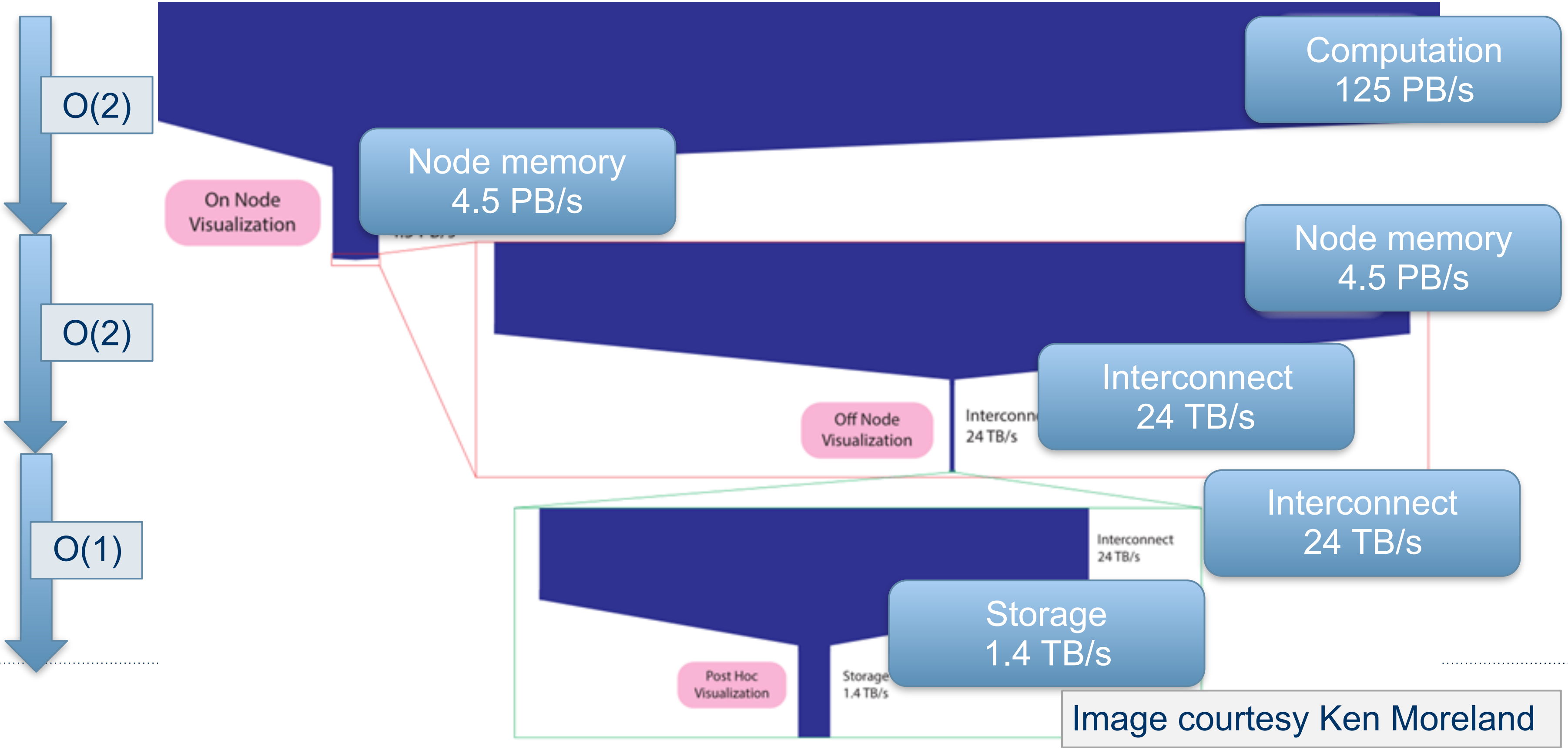


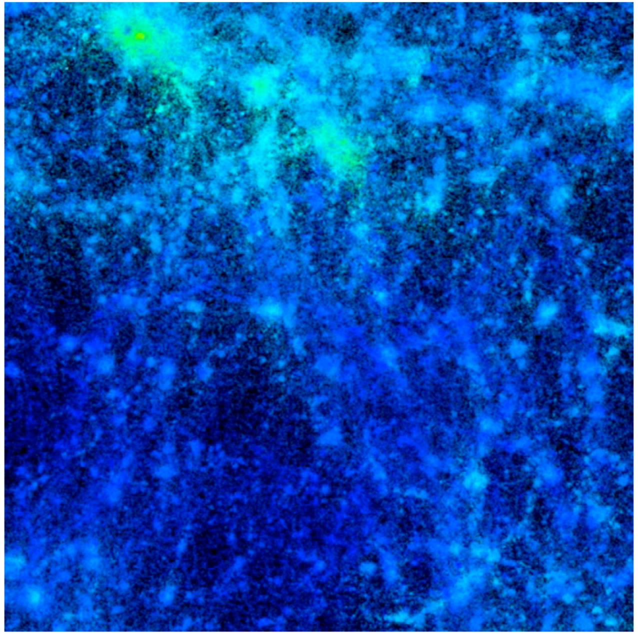
Image courtesy Ken Moreland

Trends in recent HPC systems

System attributes	NERSC Now	OLCF Now	ALCF Now	NERSC Upgrade	OLCF Upgrade	ALCF Upgrades	
Name Planned Installation	**Edison**	**TITAN**	**MIRA**	**Cori 2016**	**Summit 2017-2018**	**Theta 2016**	**Aurora 2018-2019**
System peak (PF)	2.6	27	10	> 30	150	>8.5	180
Peak Power (MW)	2	9	4.8	< 3.7	10	1.7	13
Total system memory	357 TB	710TB	768TB	~1 PB DDR4 + High Bandwidth Memory (HBM)+1.5PB persistent memory	> 1.74 PB DDR4 + HBM + 2.8 PB persistent memory	>480 TB DDR4 + High Bandwidth Memory (HBM)	> 7 PB High Bandwidth On-Package Memory Local Memory and Persistent Memory
Node performance (TF)	0.460	1.452	0.204	> 3	> 40	> 3	> 17 times Mira
Node processors	Intel Ivy Bridge	AMD Opteron Nvidia Kepler	64-bit PowerPC A2	Intel Knights Landing many core CPUs Intel Haswell CPU in data partition	Multiple IBM Power9 CPUs & multiple Nvidia Voltas GPUS	Intel Knights Landing Xeon Phi many core CPUs	Knights Hill Xeon Phi many core CPUs
System size (nodes)	5,600 nodes	18,688 nodes	49,152	9,300 nodes 1,900 nodes in data partition	~3,500 nodes	>2,500 nodes	>50,000 nodes
System Interconnect	Aries	Gemini	5D Torus	Aries	Dual Rail EDR-IB	Aries	2nd Generation Intel Omni-Path Architecture
File System	7.6 PB 168 GB/s, Lustre®	32 PB 1 TB/s, Lustre®	26 PB 300 GB/s GPFS™	28 PB 744 GB/s Lustre®	120 PB 1 TB/s GPFS™	10PB, 210 GB/s Lustre initial	150 PB 1 TB/s Lustre®

- NERSC:
  $\sim$ 12x flops,
  $\sim$ 4.5x I/O bandwidth
- ALCF:
  $\sim$ 18x flops,
  $\sim$ 3.3x I/O bandwidth
- OLCF:
  $\sim$ 5x flops,
  $\sim$ 1x I/O bandwidth

A real example



Real example:

Early science program at NERSC:

- 8192^3 element N-body + hydrodynamics simulation on NERSC's Cori.
- ~ 16M CPU hours
- ~ 256TB memory
- 20TB user scratch quota. (A double precision 8192^3 array is 4TB. Checkpoint has 14 arrays.)

What is *in situ* data analysis and visualization?

- **Post processing**: save to disk, then later, a separate analysis/vis program reads that data and operates on it.

What is *in situ* data analysis and visualization?

- **Post processing**: save to disk, then later, a separate analysis/vis program reads that data and operates on it.
- ***In situ* processing**: process data as it produced without writing to and reading from storage. Processed "in place".

What is *in situ* data analysis and visualization?

- **Post processing**: save to disk, then later, a separate analysis/vis program reads that data and operates on it.
- ***In situ* processing**: process data as it produced without writing to and reading from storage. Processed "in place".
  - Many flavors/terms: tightly coupled, loosely coupled, in transit, co-processing, etc.
  - Practical view: anything processed but not written to persistent storage is *in situ*

Generic processing sequence

```
1. initialize sim
2. do
3.   compute new state
4.   if do_io write plot file
5. while !done
6. finalize sim
```

Generic processing sequence w/ in situ

```
1. initialize sim
2. if do_insitu initialize in situ
3. do
4.   compute new state
5.   if do_io write plot file
6.   if do_insitu execute in situ
7. while !done
8. if do_insitu finalize insitu
9. finalize sim
```

Generic processing sequence w/ in situ

```
1. initialize sim
2. if do_insitu initialize in situ
3. do
4.   compute new state
5.   if do_io write plot file
6.   if do_insitu execute in situ
7. while !done
8. if do_insitu finalize insitu
9. finalize sim
```

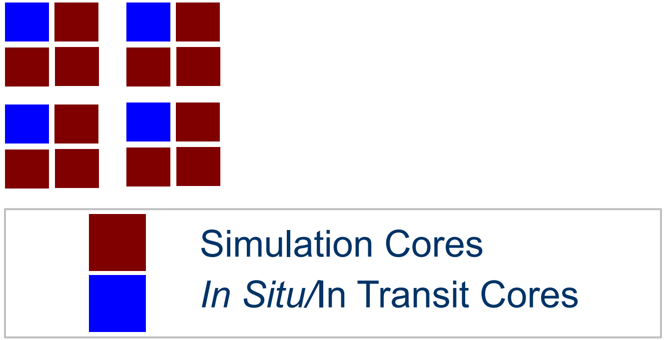
execute is where things get interesting

- shared address space zero copy data transfers to shared or unique compute resources
- staging transfer sends data to a de-coupled parallel job, potentially asynchronous, potentially different jobs size

In situ vs In transit

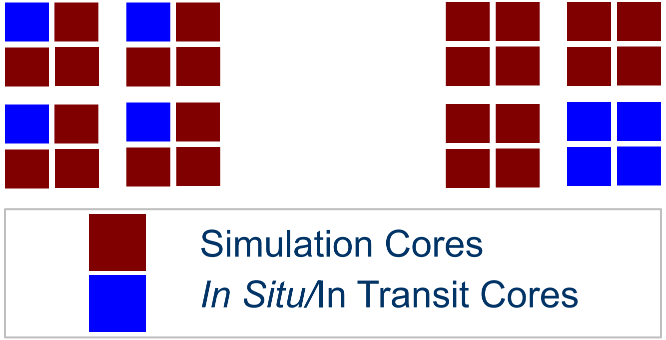
In situ vs In transit

<u>In situ</u> – no data movement: Simulation and *in situ* methods share memory



In situ vs In transit

In situ – *no data movement*: Simulation and *in situ* methods share memory



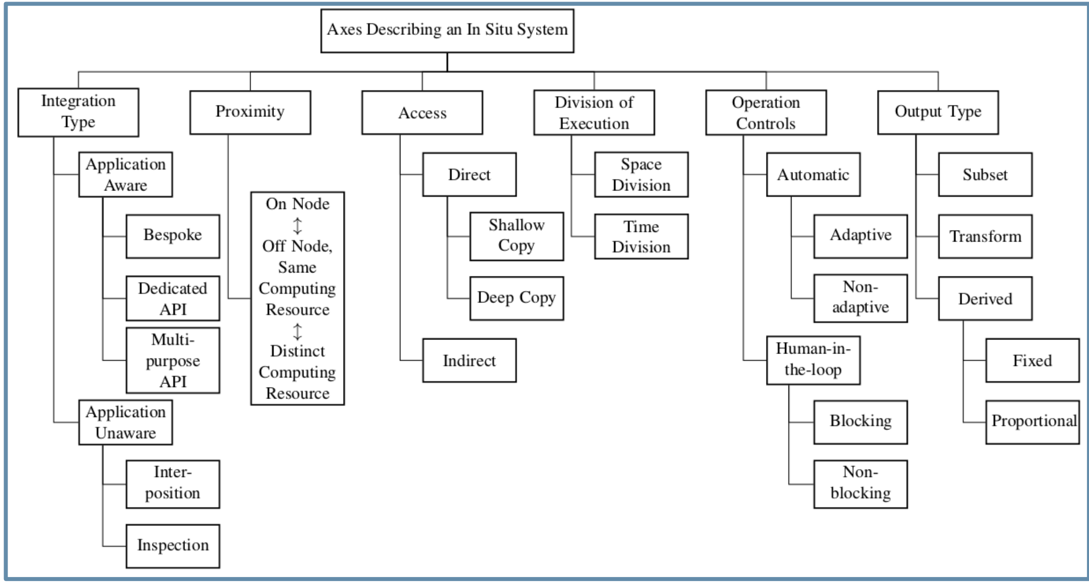
In transit – data is moved: Simulation and *in situ* methods do not share memory

The story is much more interesting than “in situ” vs. “in transit”

In situ vs. in transit is an oversimplification of a much richer problem space

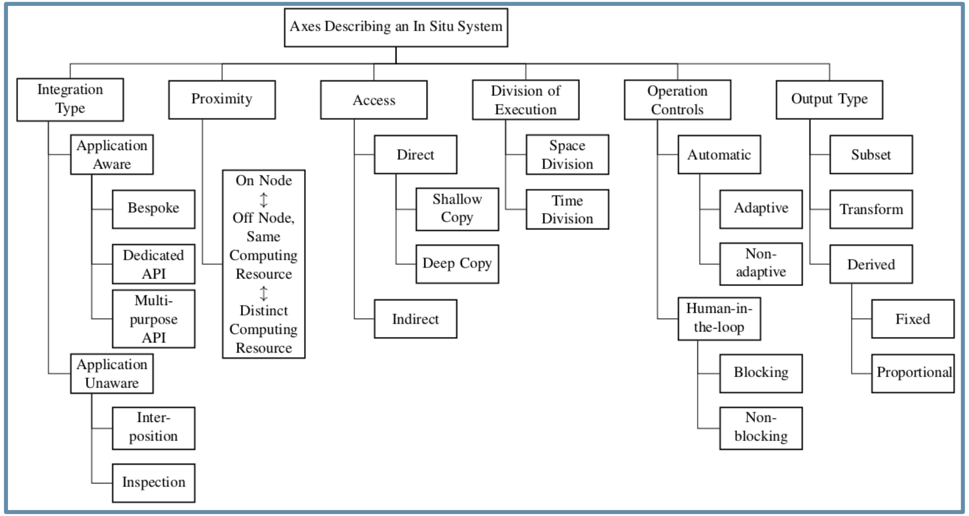
The “In Situ Terminology Project”

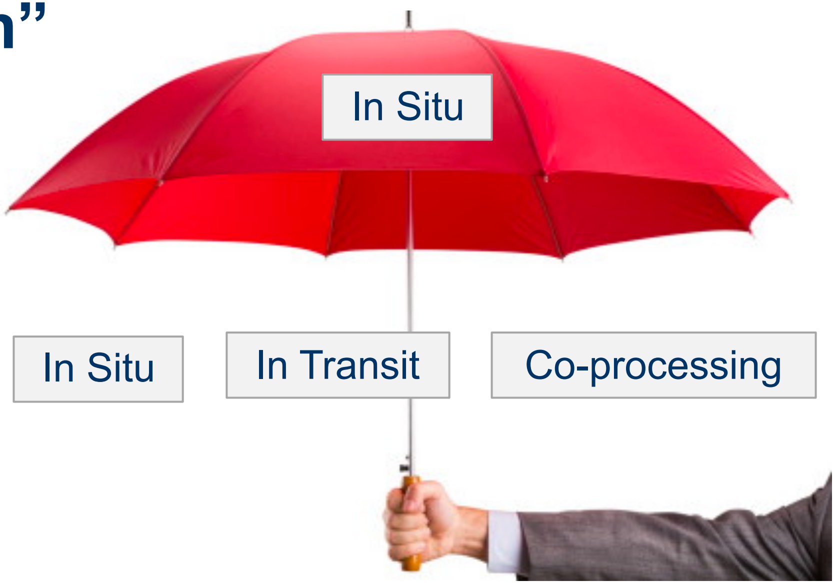
- A community effort (>50 participants)
- Identify “basis vectors” for describing aspects of in situ processing
  - Integration Type, Proximity, Access, Division of Execution, Operation Controls, Output Type



In situ: an "umbrella definition"

In situ is term that covers a lot of territory:





In Situ Terminology project:
http://ix.cs.uoregon.edu/~hank/insituterminology/
Community effort to identify basis vectors and name them.

In situ has been around a long time: ancient history

E. Zajac, CACM 7(3), Mar 1964.

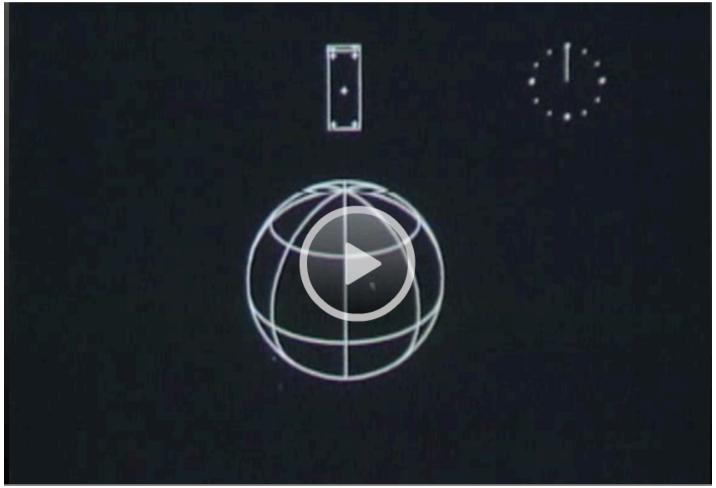
Direct-to-film process (simulation, calligraphic display exposes film) movie of a satellite orbiting a planet.

Is this *in situ?*

- Yes: no data ever landed on disk.

Why did he do it?

- "Standard practice" for that era, and many years that followed: direct-to-media more efficient.



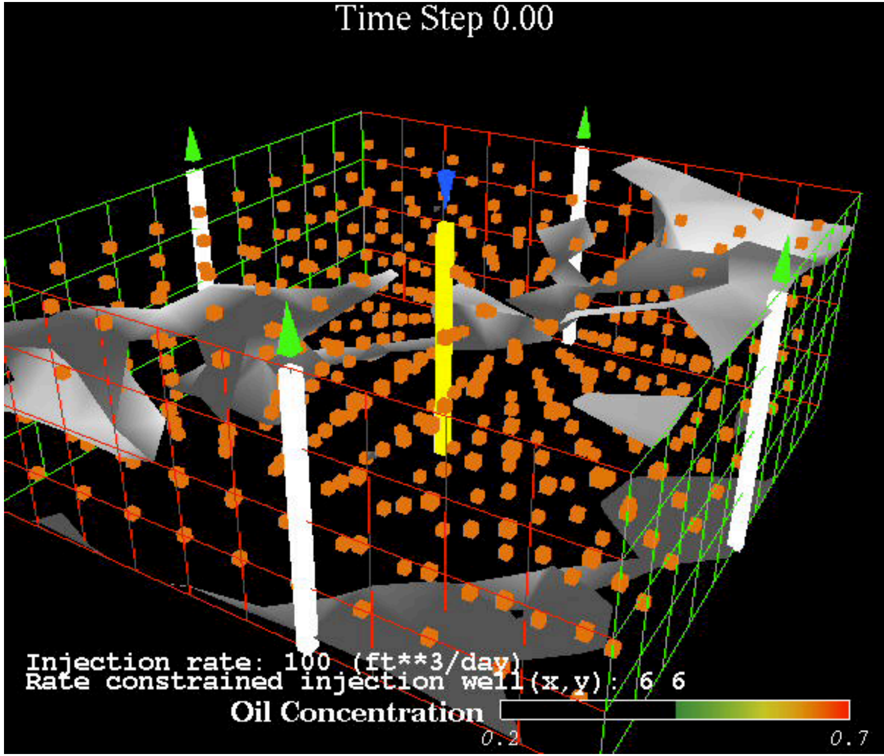
Link to movie page

The 1990s: the golden era of coprocessing

Main idea: systems/methods that support interactive computation, computational monitoring and steering.

Packages from this era (partial list):

- pV3: custom distributed memory code (Haimes)
- AVS: co-routine processing (serial, mostly)
- CUMULVS: distributed memory M-to-N visualization, steering (based on PVM) (Kohl, et al.)



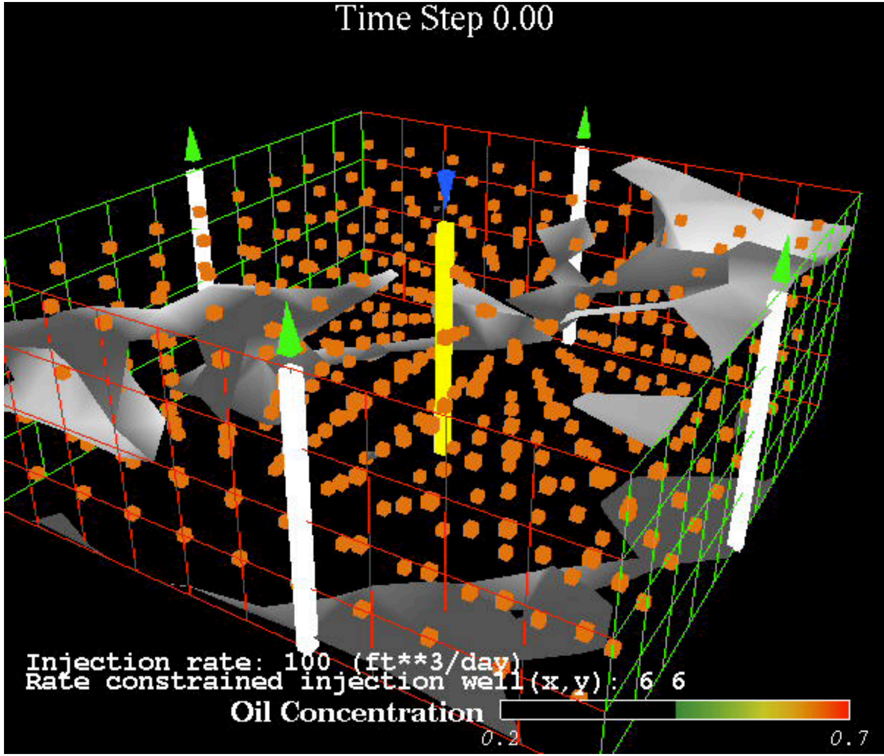
Bethel and Jacobsen (1994, 1995). Coupling a multi-phase reservoir simulator with AVS.

The 1990s: the golden era of coprocessing

Main idea: systems/methods that support interactive computation, computational monitoring and steering.

Packages from this era (partial list):

- pV3: custom distributed memory code (Haimes)
- AVS: co-routine processing (serial, mostly)
- CUMULVS: distributed memory M-to-N visualization, steering (based on PVM) (Kohl, et al.)



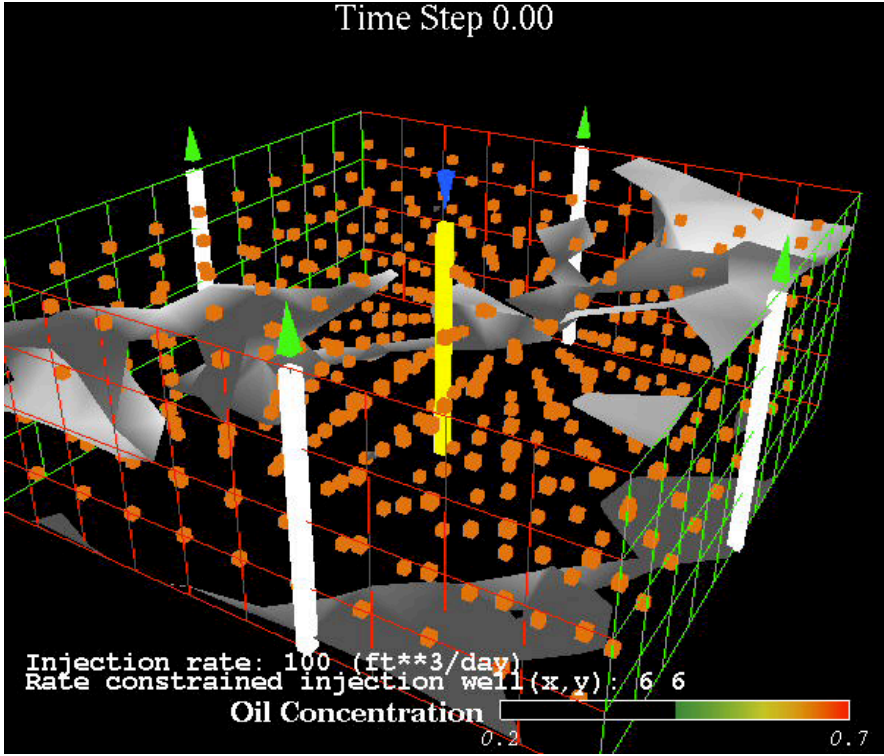
Bethel and Jacobsen (1994, 1995). Coupling a multi-phase reservoir simulator with AVS.

The 1990s: the golden era of coprocessing

Main idea: systems/methods that support interactive computation, computational monitoring and steering.

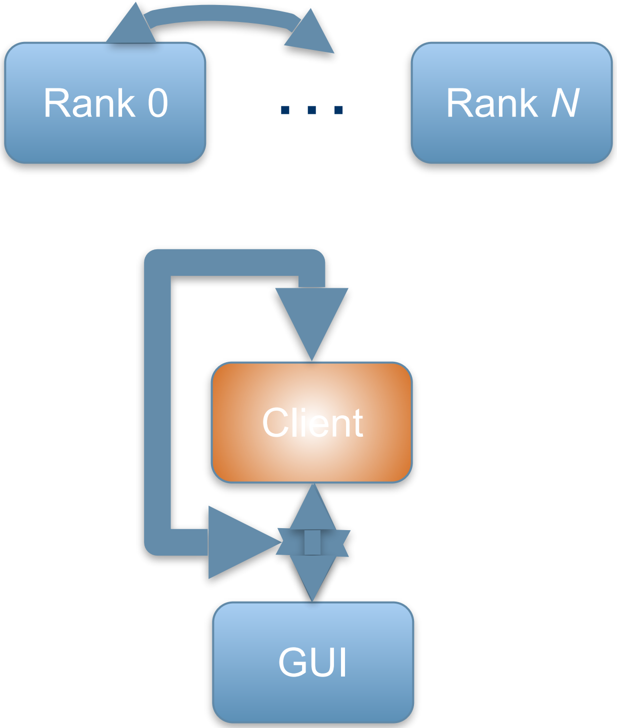
Packages from this era (partial list):

- pV3: custom distributed memory code (Haimes)
- AVS: co-routine processing (serial, mostly)
- CUMULVS: distributed memory M-to-N visualization, steering (based on PVM) (Kohl, et al.)



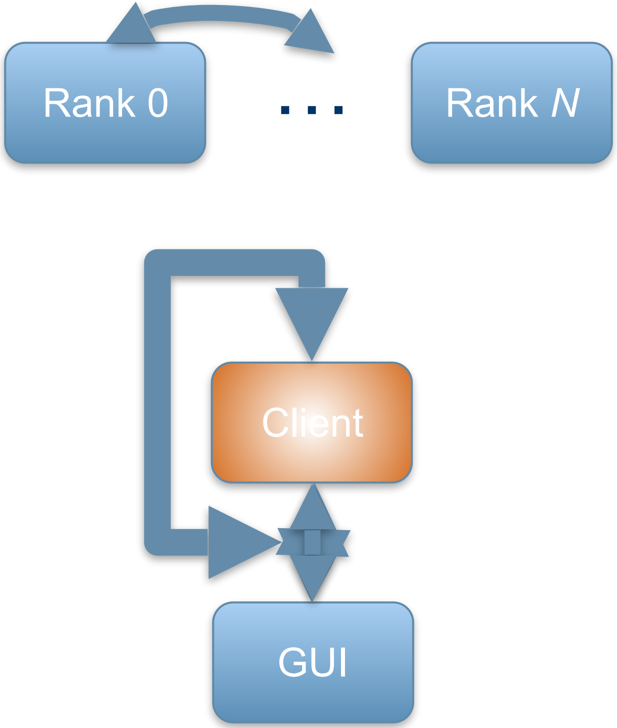
Bethel and Jacobsen (1994, 1995). Coupling a multi-phase reservoir simulator with AVS.

Common design patterns of 1990s

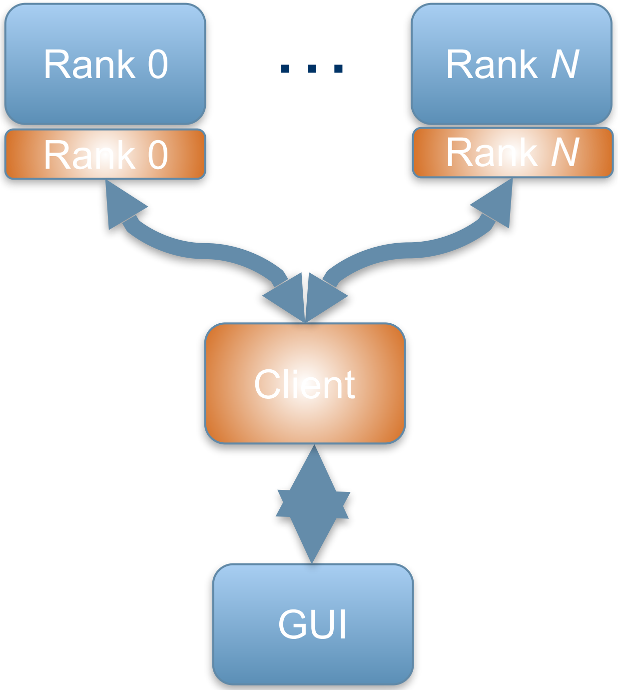


Many-to-one: AVS

Common design patterns of 1990s

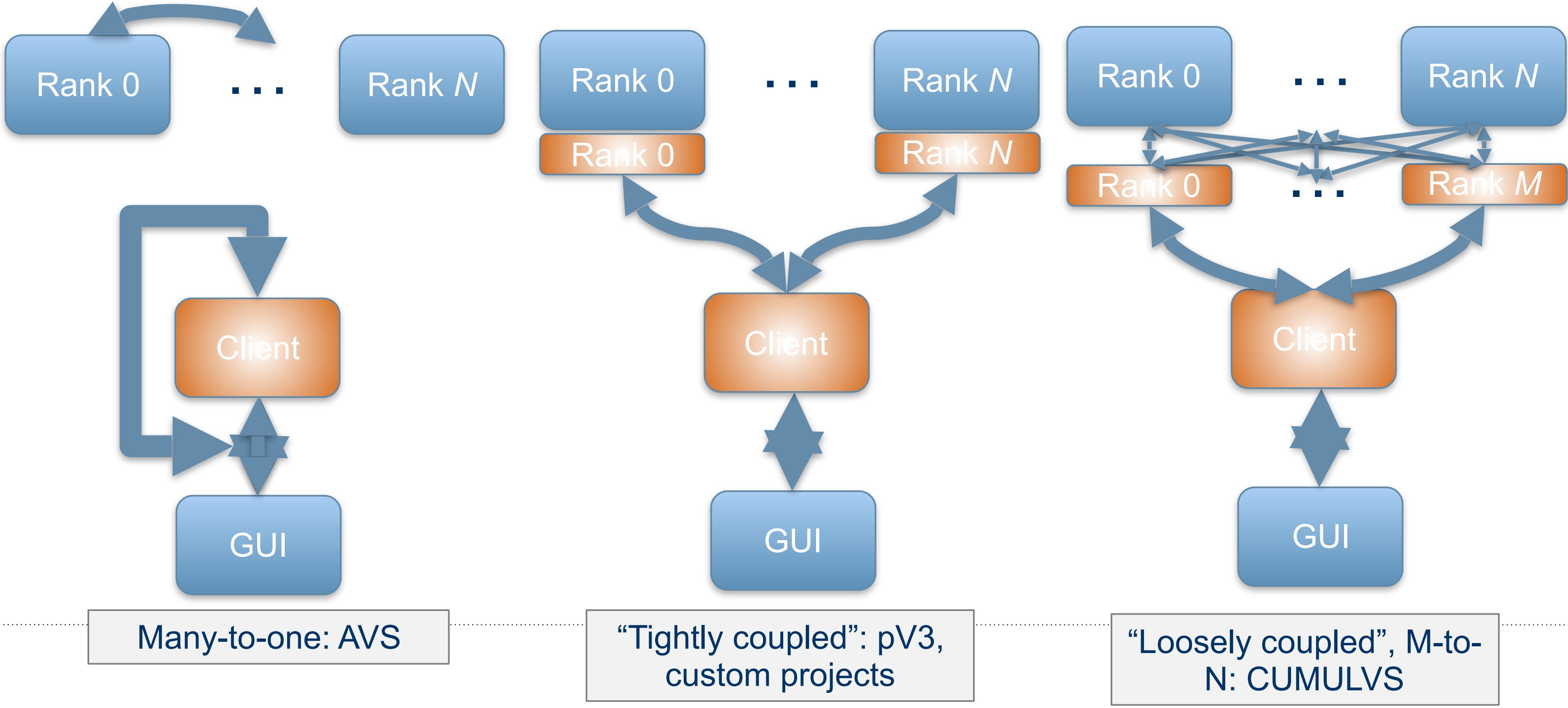


Many-to-one: AVS



“Tightly coupled”: pV3, custom projects

Common design patterns of 1990s



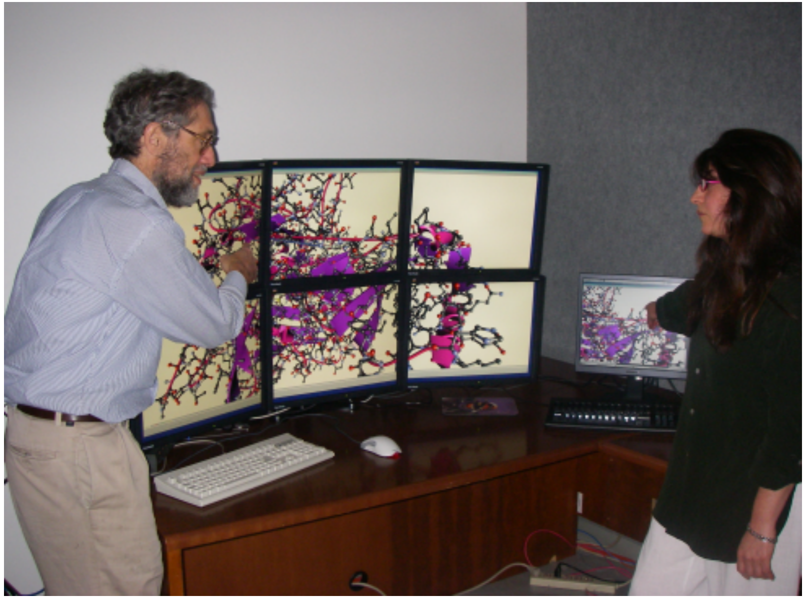
Computational steering – human in the loop

Main idea: rapid convergence

Example: protein structure prediction, find optimal-energy conformation from initial conditions (NP-hard problem)

Approach:

- parallel computations that minimize energy for individual conformations
- User can examine any of these, perform manual tweaks to get "unstuck" from local minimum, then resume calculations.



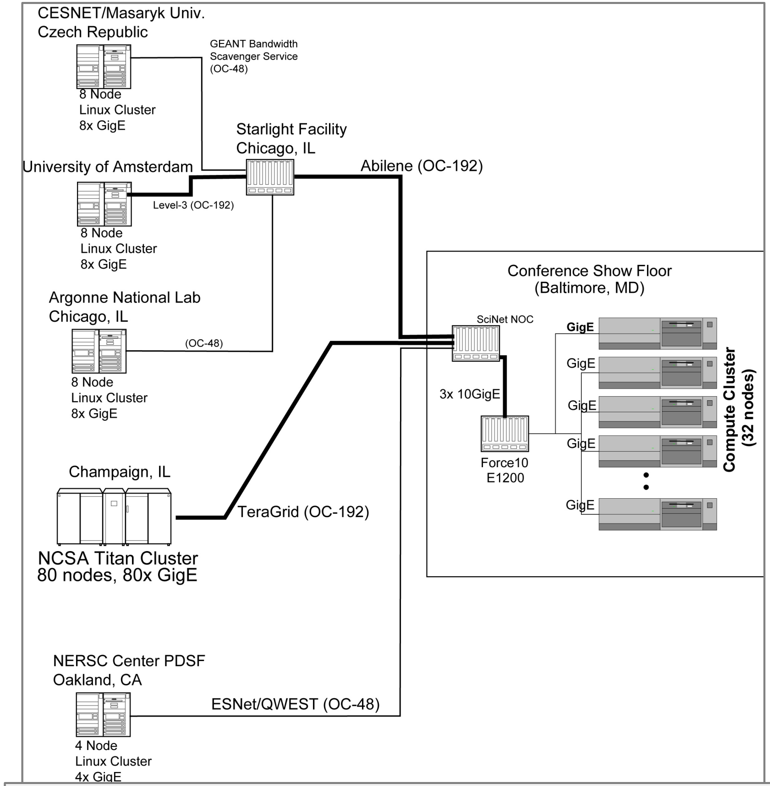
O. Kreylos, N. Max, B. Hamann, S. Crivelli, W. Bethel. *Interactive Protein Manipulation.* IEEE Vis 2003, Best Application Paper award.

Integrated computational environments

- Simplify building, running codes
- Many add-on capabilities for vis, analysis, debugging, data I/O, etc.

Examples: SCIRun, Cactus

Application (sample): parallel binary black hole merger computation, in transit vis wins SC Bandwidth Challenge (2000, 2001, 2002)



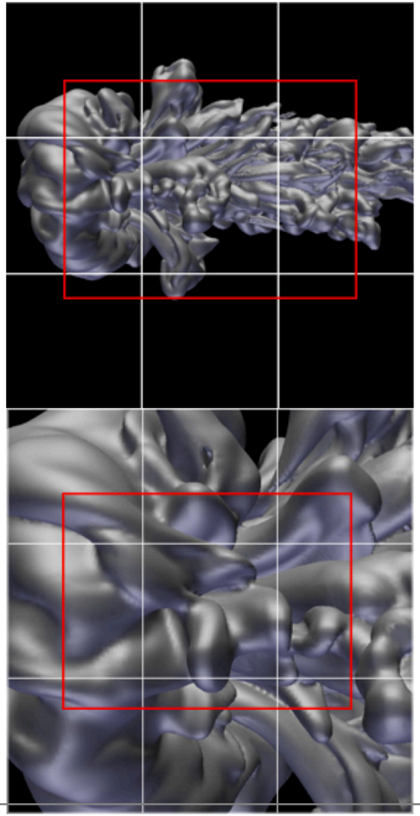
Resources used in SC 2002 Bandwidth Challenge, in transit workflow

Explorable extracts

Basic ideas:

- Overcome *in situ* primary weakness: know before you go.
- Use *in situ* computation to produce reduced-size datasets, e.g., images, data subsets, "extracts" like collections of features, etc.
- These "data extracts" are much smaller in size compared to doing full resolution data I/O.
- Use some post-processing tool to view/analyze/interact with these extracts.

Climate modeling example using Catalyst and Cinema in our STAR paper.



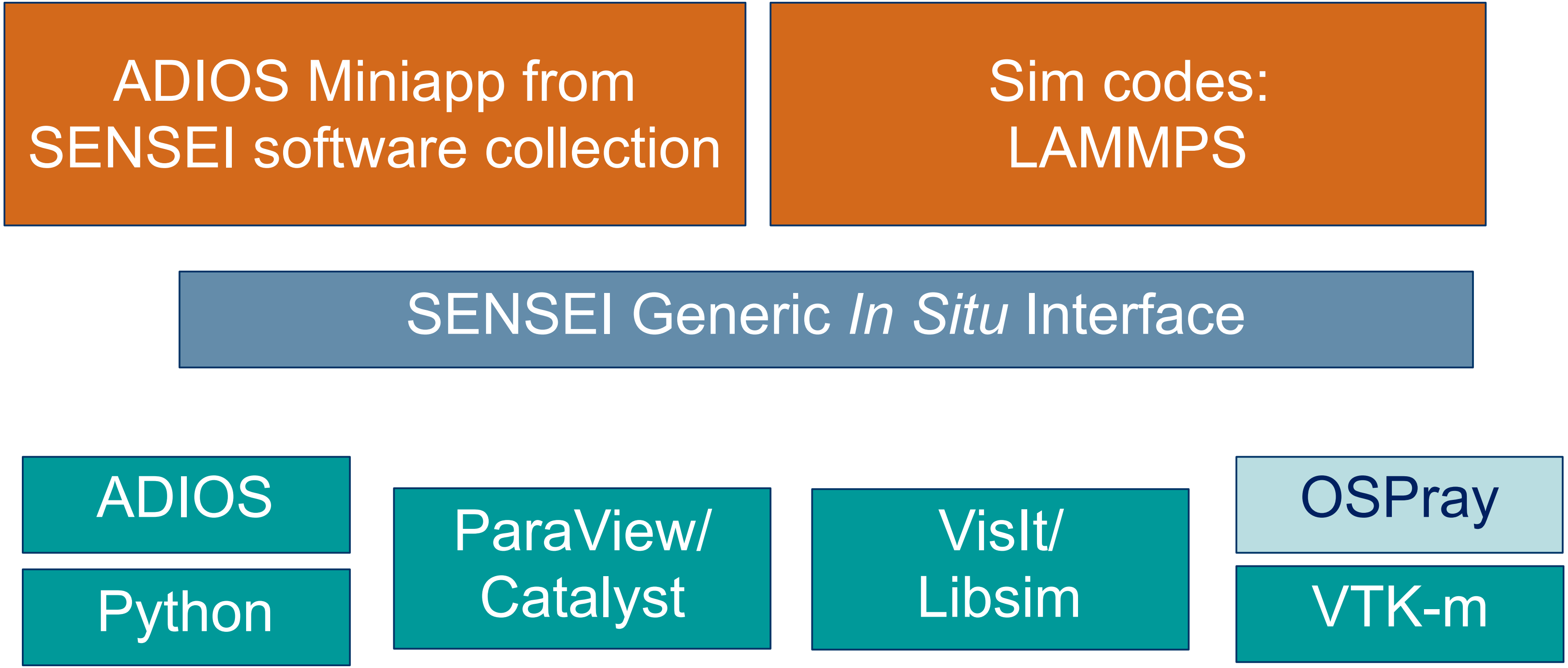
Chen et al., *Interactive, Internet Delivery of Visualization via Structured, Prerendered Multiresolution Imagery.* TVCG 14(2), 2008.

Bauer, et al., *In Situ Methods, Infrastructures, and Applications on HPC Platforms, a State-of-the-Art (STAR) Report, Computer Graphics Forum, 35(3), 2016.*

In situ projects over the years (approximate, partial)

- 1964: Zajac, direct-to-film animations
- 1990s: Code coupling, computational steering:
  - AVS
  - pV3
  - CUMULVS
- 2000s (early): Integrated Computational Environments:
  - SCIRun
  - CACTUS
- 2000s (late): Computing Extracts for Post Hoc Use
  - Multiresolution, precomputed images
  - Topology
  - Geometry
- Present day:
  - VisIt/Libsim, Paraview/Catalyst: scalable vis infrastructure accessible *in situ*
  - ADIOS: I/O library approach
  - SENSEI: generic *in situ* interface
  - Other nascent efforts

Roadmap of *In Situ* Software Infrastructure for Today



SENSEI System Overview

BERKELEY LAB

Argonne NATIONAL LABORATORY

OAK RIDGE National Laboratory

Intelligent Light

Kitware

In situ infrastructures

Relatively new. Until recently, **ad hoc**, **proof-of-concept prototypes.** However, several **production quality** *in situ* **infrastructures** have emerged

ADIOS provides tools for *in situ* **I/O** , **data movement** and **analysis**

- ADIOS allows simulations to adopt *in situ* techniques by **leveraging** their **advanced I/O infrastructures** that enable co-analysis pipelines **rather than changing the simulator**.
- The non-intrusive integration **provides resilience** to third party library bugs and possible jitter in the simulation.

ParaView and **VisIt** both provide tools for *in situ* **analysis** and **visualization**

- Can be **tightly** or **loosely** linked to a simulation, allowing the simulation to **share data** with Catalyst for analysis and visualization.
- Catalyst, Libsim, and ADIOS enable the **opposite flow of information**, sending data from the client to the simulation, enabling the possibility of *in situ* and/or **monitoring/simulation steering**.

Ascent an emerging in situ framework with an elegant data model, taking advantage of emerging **VTK-m** many core analysis and rendering capabilities


how to choose?

Can WE….

Enable use of any in situ framework?

Enable use of any analysis library/tool, even those not designed for in situ?

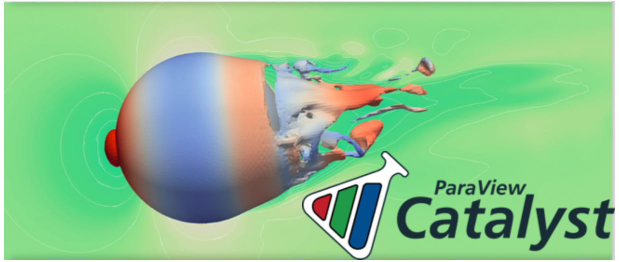
Develop analysis routines that are portable between codes?

Make it easy to use?

The *original* problem set

ADIOS

Libsim



www.olcf.ornl.gov/center-projects/adios
wci.llnl.gov/simulation/computer-codes/visit
www.paraview.org/in-situ

The *current* problem set

SENSEI seamlessly & efficiently enables in situ data processing with a diverse set of tools & libraries

Our approach

Data model

- The lingua franca allowing an analyses to access simulation data consistently across a variety of simulations

Data adaptor

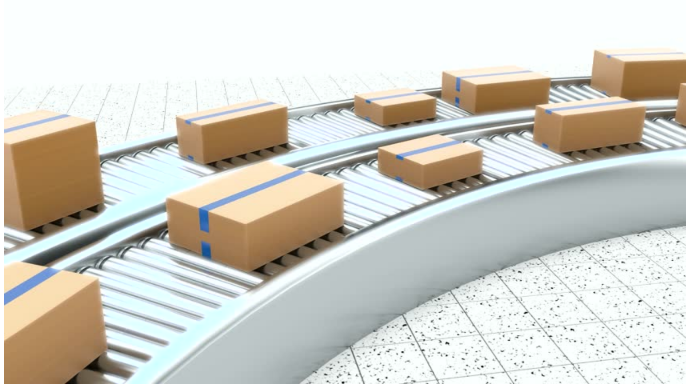
- Convert simulation data to/from the data model
- API for accessing the simulation data from the backend

Analysis adaptor

- Present the back-end data consumer to the simulation
- API for pushing data through the system from the sim

Library

- Providing off the shelf access to a diverse set of back-ends. eg Libsim, Catalyst, and ADIOS capabilities



Write once run everywhere

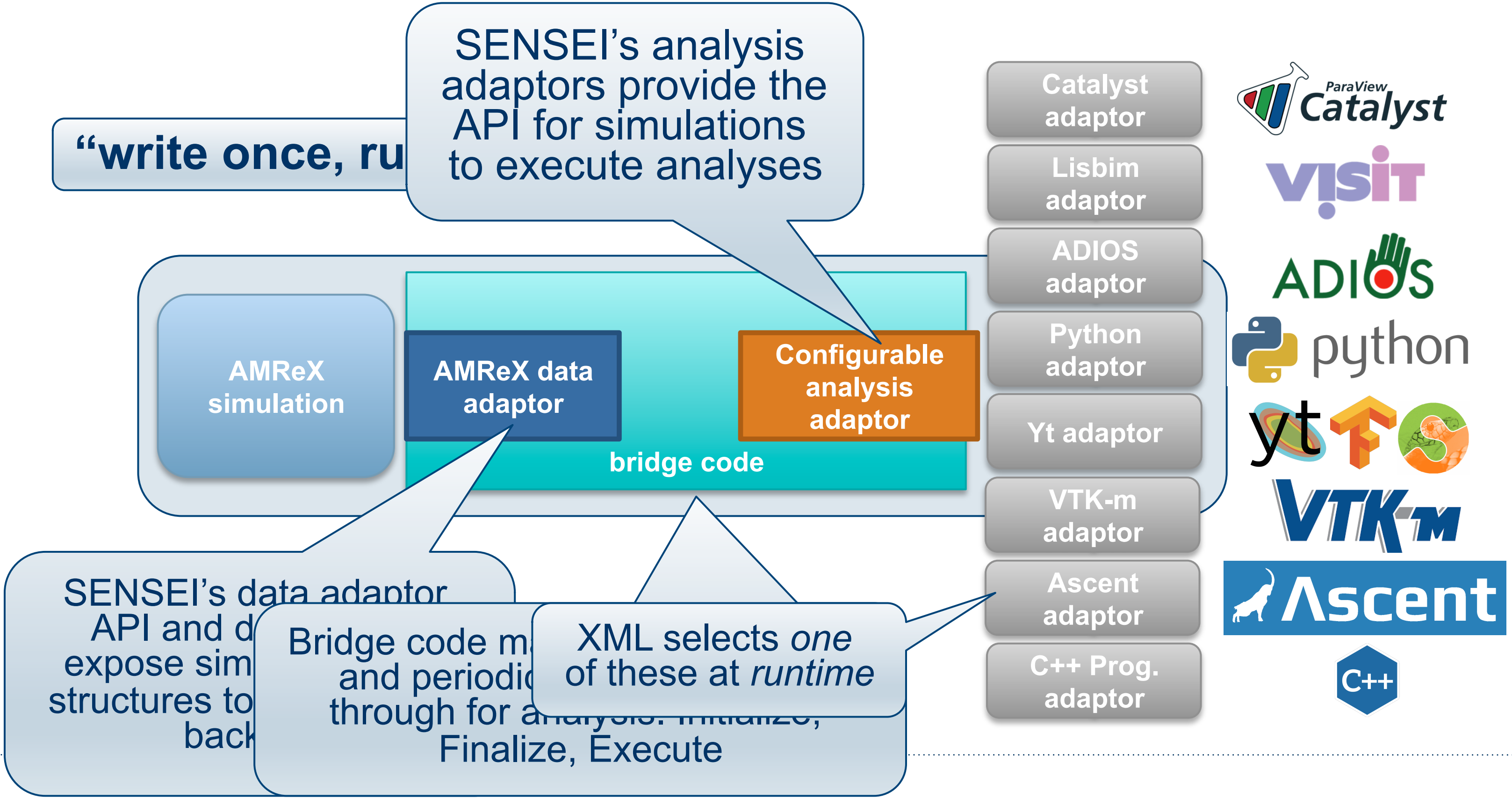
The **SENSEI API** enables connection of simulation data sources to visualization and analysis back ends

- From the perspective of the simulation, the back ends(analysis/vis codes) are interchangeable

The **SENSEI data model** enables viz & analysis codes to access data through a unified API.

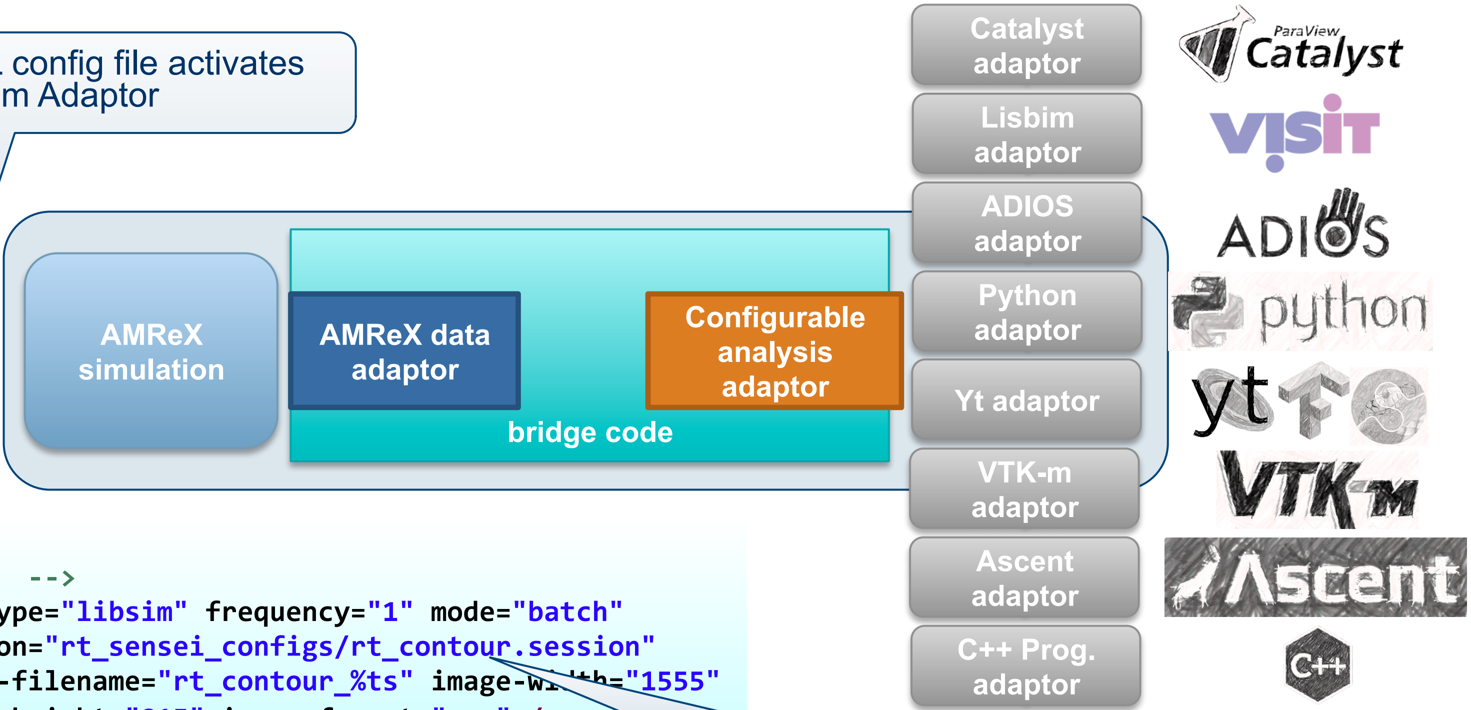
- From the perspective of the analysis/visualization code, data sources(simulations) are interchangeable

In situ Architecture



Use w/ VisIt

SENSEI XML config file activates the VisIt Libsim Adaptor

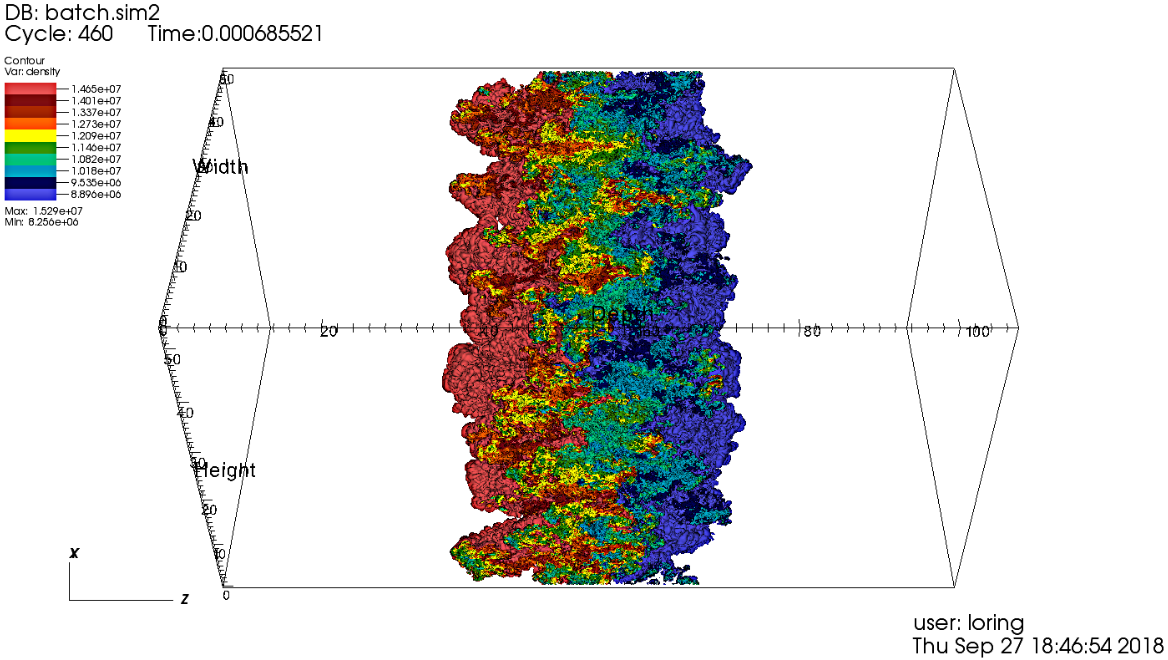



```xml
<sensei>
  <!-- libsim  -->
  <analysis type="libsim" frequency="1" mode="batch"
        session="rt_sensei_configs/rt_contour.session"
        image-filename="rt_contour_%ts" image-width="1555"
        image-height="815" image-format="png" />
</sensei>
```

Session file created in VisIt GUI configures VisIt

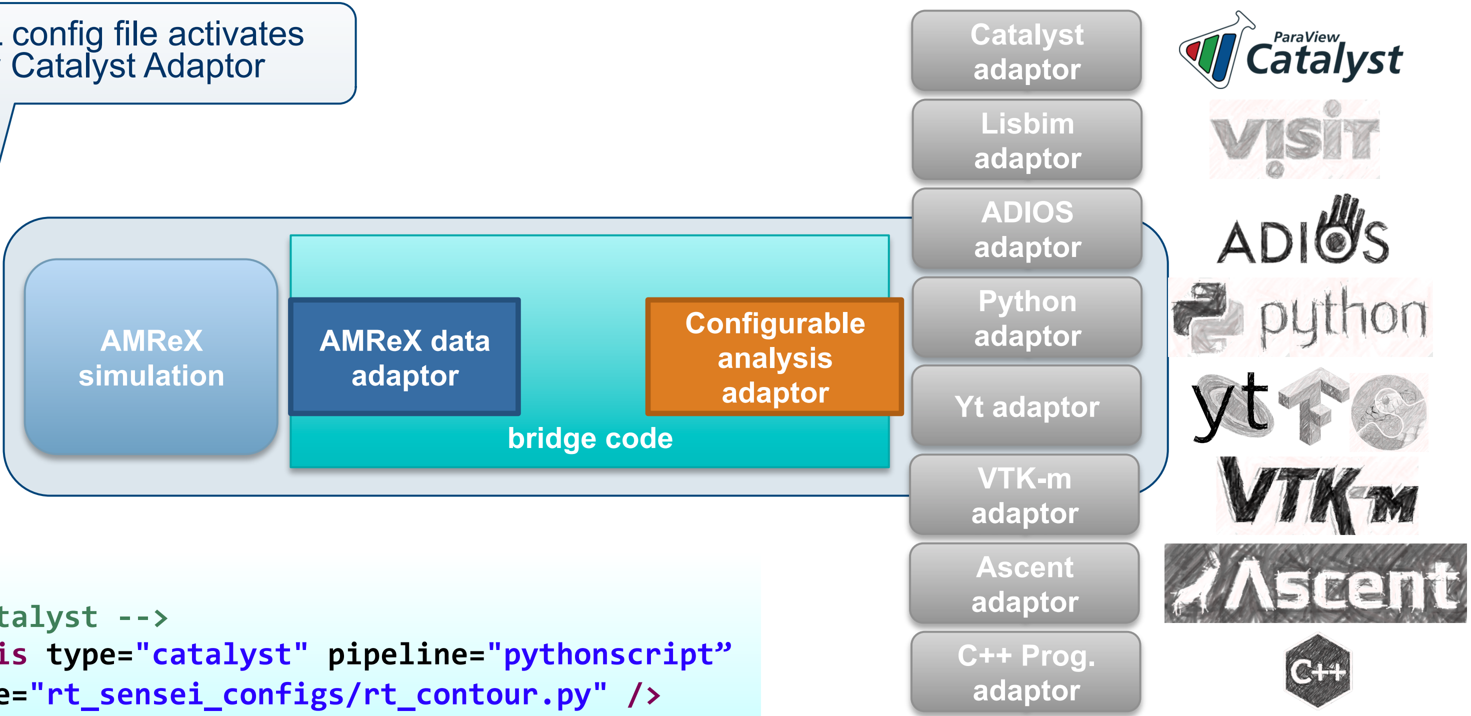
IAMR Rayleigh-Taylor Libsim

2048 Cores Cori Haswell



Use w/ ParaView Catalyst

SENSEI XML config file activates the ParaView Catalyst Adaptor

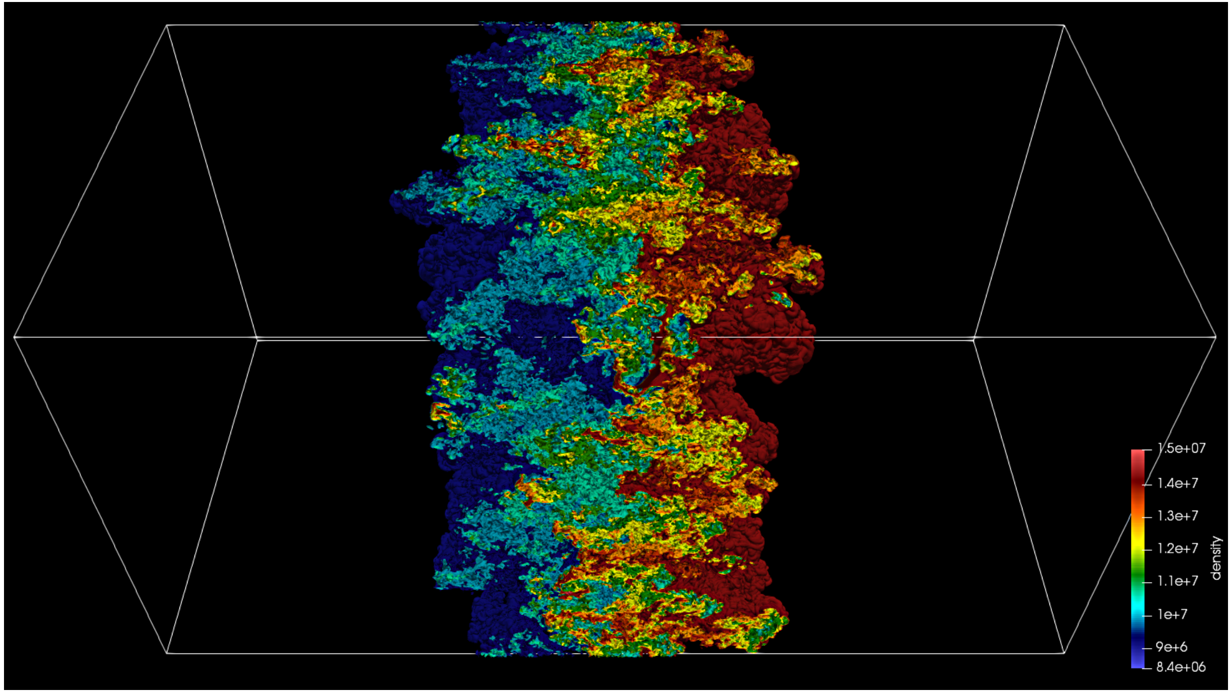



```xml
<sensei>
  <!-- catalyst -->
  <analysis type="catalyst" pipeline="pythonscript”
  filename="rt_sensei_configs/rt_contour.py" />
</sensei>
```

Catalyst python script created in ParaView GUI configures Catalyst

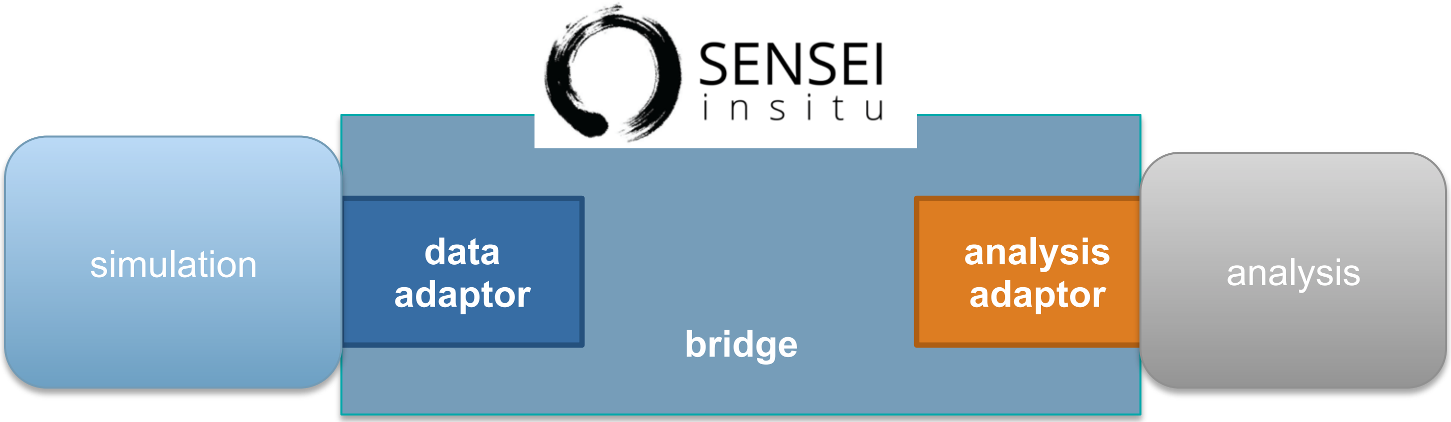
IAMR Rayleigh-Taylor Catalyst

2048 Cores Cori Haswell



SENSEI API's

DataAdaptor API



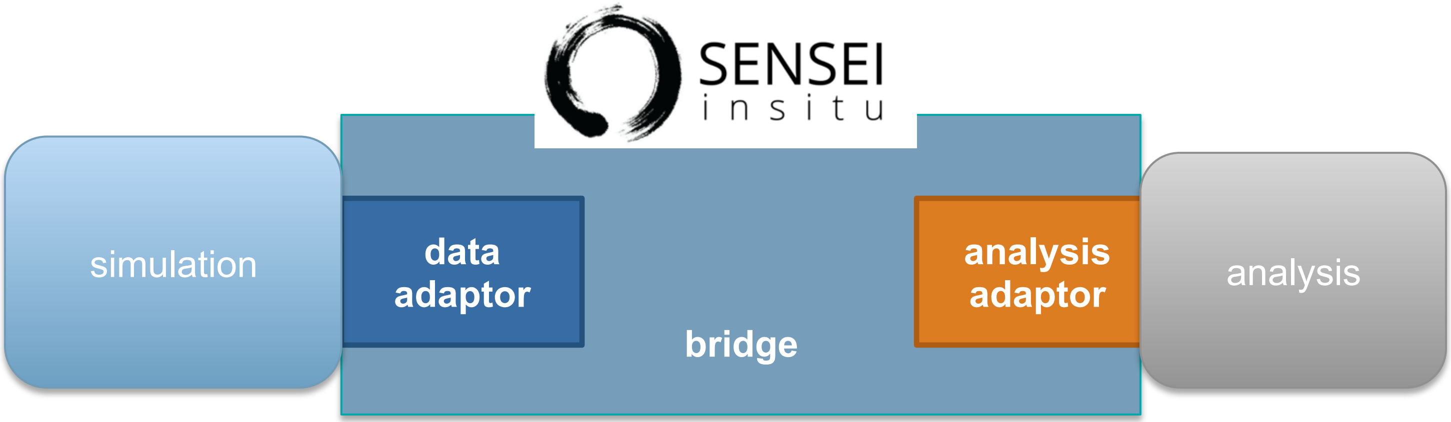
DataAdaptor API

- Provides the API through which data is accessed
- Converts simulation data structures into VTK data structures on demand
- Is used by the analysis adaptor to access simulation data on demand

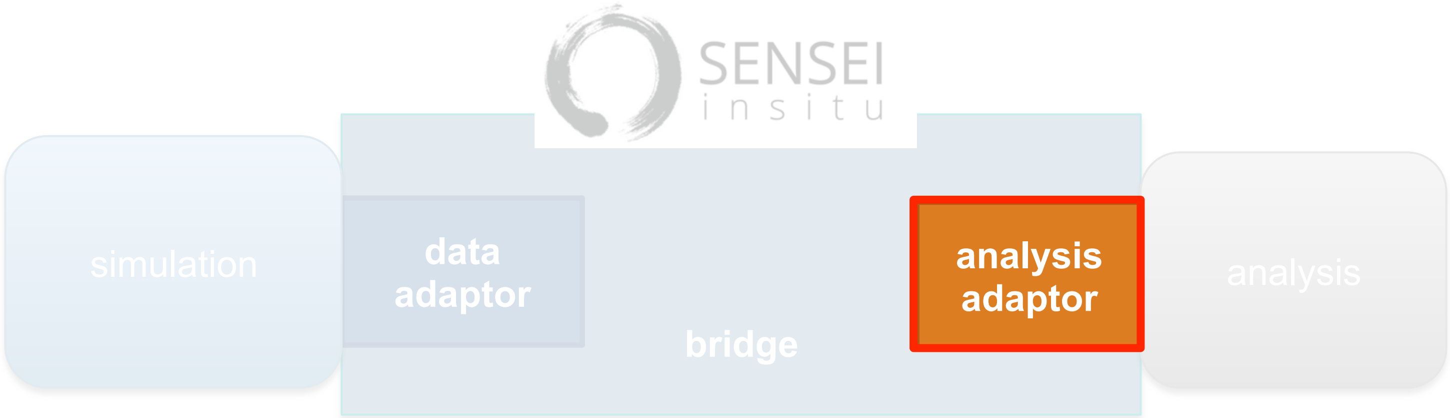
DataAdaptor API

```
/// @breif Gets the number of meshes a simulation can provide
virtual int GetNumberOfMeshes(unsigned int &numMeshes) = 0;

/// @breif Get the name of the i'th mesh
virtual int GetMeshName(unsigned int id, std::string &meshName) = 0;

/// @breif get a list of all mesh names
virtual int GetMeshNames(std::vector<std::string> &meshNames);

/// @brief Return the data object with appropriate structure.
virtual int GetMesh(const std::string &meshName, bool structureOnly,
  vtkDataObject *&mesh) = 0;

/// @brief Adds the specified field array to the mesh.
virtual int AddArray(vtkDataObject* mesh, const std::string &meshName,
  int association, const std::string &arrayName) = 0;

/// @brief Return the number of field arrays available.
virtual int GetNumberOfArrays(const std::string &meshName, int association,
  unsigned int &numberOfArrays) = 0;

/// @brief Return the name for a field array.
virtual int GetArrayName(const std::string &meshName, int association,
  unsigned int index, std::string &arrayName) = 0;

/// @brief Release data allocated for the current timestep.
virtual int ReleaseData() = 0;
```

AnalysisAdaptor API



AnalysisAdaptor API

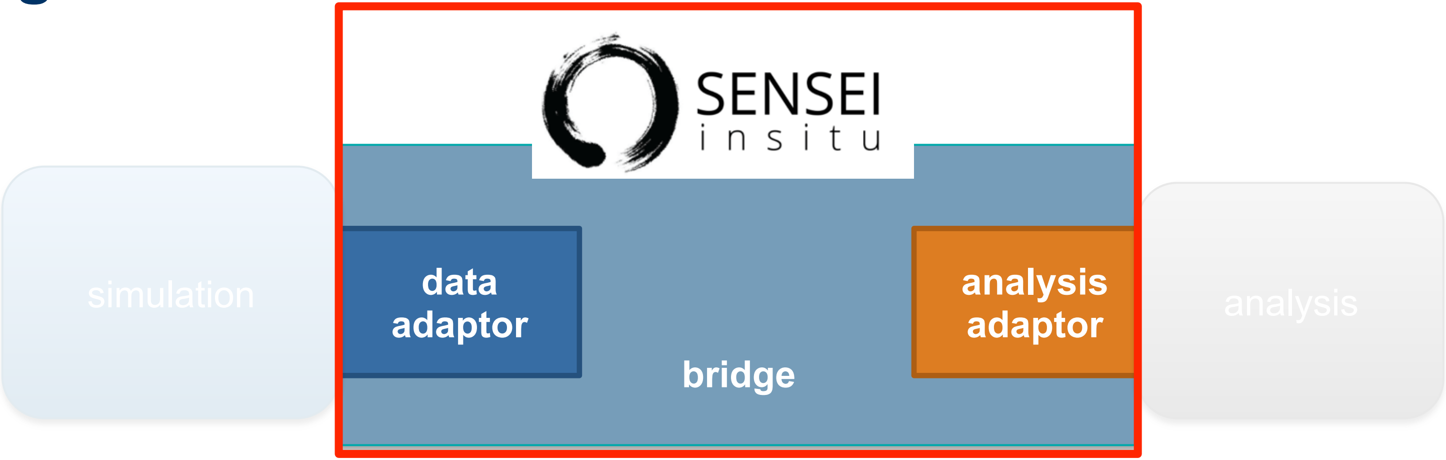


- Provides the API for driving the analysis
- Invoked by the bridge from the simulation when it is time for analysis
- A DataAdaptor instance is passed, which the analysis code uses to access simulation data structures

AnalysisAdaptor API

```
/// @brief AnalysisAdaptor is an abstract base class that defines
/// the analysis interface.
class AnalysisAdaptor : public vtkObjectBase
{
public:
   /// @brief Execute the analysis routine.
   virtual int Execute(DataAdaptor* data) = 0;

    /// @breif Finalize the analyis routine
    virtual int Finalize() = 0;
};
```

Bridge API



- Is part of the simulation code
- Is where you create, initialize, and manage your data and analysis adaptors
- Is where you execute the analyses adaptors as needed
- Typically consists of 3 functions: Initialize, Compute and Finalize

Simulation loop with bridge code

```
initialize sim
if do_insitu bridge::initialize
do
  compute new state
  if do_io write plot file
  if do_insitu bridge::execute
while !done
if do_insitu bridge::finalize
finalize sim
```

Run time configuration

Adaptors

- SENSEI Configurable analysis. Parses XML and creates and configures one of the other analysis adaptors interfacing to the back-ends (Libsim, Catalyst, ADIOS, custom, etc).
- Direct integration

Back-ends

- May expose control API via their SENSEI adaptor. In the Configurable analysis adaptor these are exposed via XML attributes.
- May be scriptable via their own Python bindings adding another layer of control.
- May be configured via "state" or "session" files.
- Special purpose

ConfigurableAnalysisAdaptor

- a meta analysis. a manager. it configures and invokes one or more of the other analysis adaptors
- XML specifies analyses and their run time options
- Supports ADIOS, Catalyst, Libsim, VTK I/O, and other data consumers
- In in transit use cases one XML configures the transport a second configures the analysis/backend

ConfigurableAnalysis XML

```xml
<sensei>
  <!-- Custom Analyses -->
  <analysis type="histogram" mesh="bodies" array="v" association="point"
    bins="10" enabled="0" />

  <!-- VTK XMLP I/O -->
  <analysis type="PosthocIO" mode="paraview" output_dir="./" enabled="0">
    <mesh name="bodies">
        <point_arrays> ids, m, v, f </point_arrays>
    </mesh>
  </analysis>

  <!-- CATALYST -->
  <analysis type="catalyst" pipeline="pythonscript"
    filename="../sensei/miniapps/newton/newton_catalyst.py" enabled="1" />

  <!-- LIBSIM -->
  <analysis type="libsim" plots="Pseudocolor" plotvars="ids"
    image-filename="newton_%ts" image-width="800" image-height="800"
    slice-project="1" image-format="png" enabled="0"/>
</sensei>
```

SENSEI Data Model

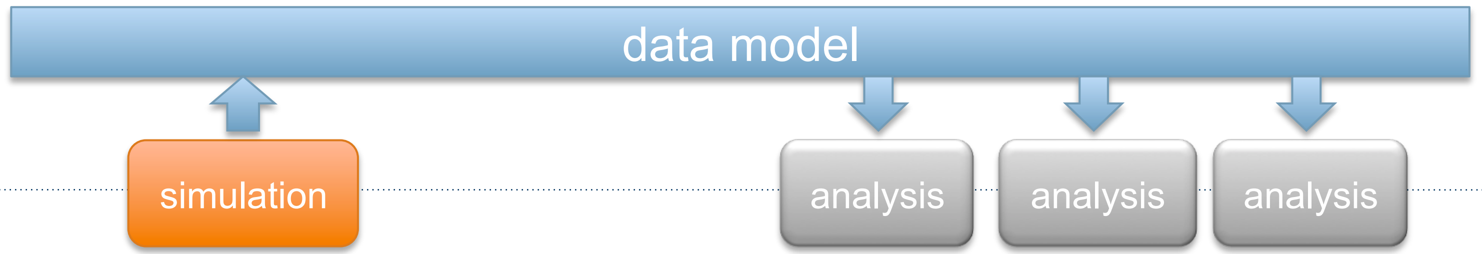
Data model roles

Challenges

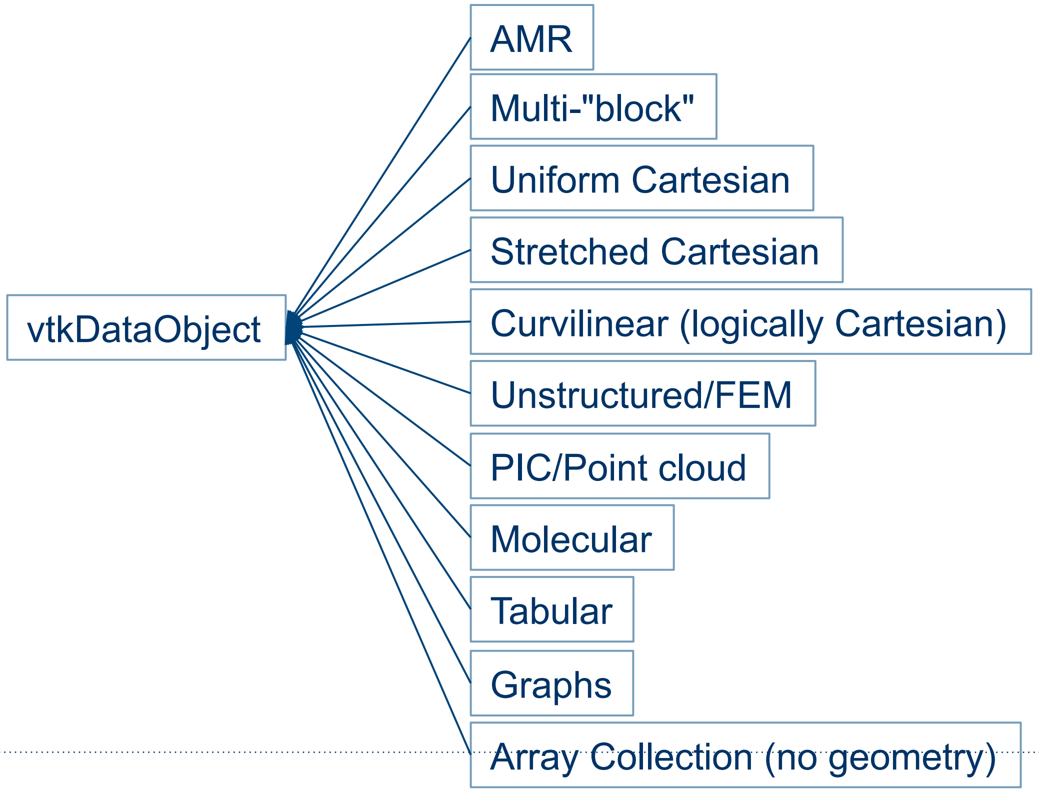
- large bodies of existing codes with purpose specific non standard data models can't talk to each other
- data needs are diverse

Solutions

- Agreement between simulation and analysis on a data model enables the exchange of data
- Normalization of data model enables a generic solution



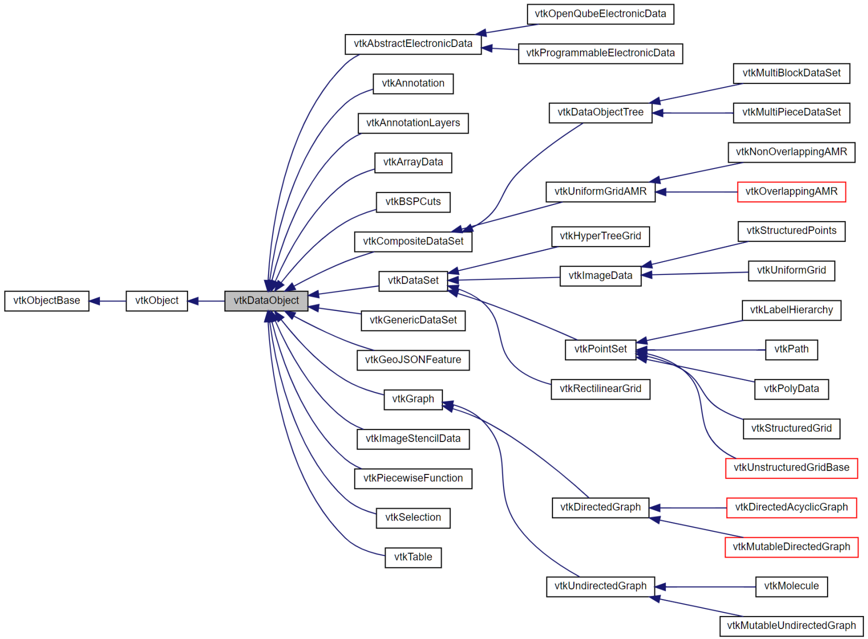
What simulation data types does SENSEI support?



- many more purpose specific and esoteric data types are supported by VTK
- **SENSEI has no explicit dependence on other parts of VTK such as i/o, filters, renderering, etc etc**

www.vtk.org

vtkDataObject – The key to passing data in SENSEI



- You can pass any of these classes derived from vtkDataObject through the SENSEI API
- Go to the link below. use the clickable class diagram to navigate / access documentation for the specific data object types

https://www.vtk.org/doc/nightly/html/classvtkDataObject.html

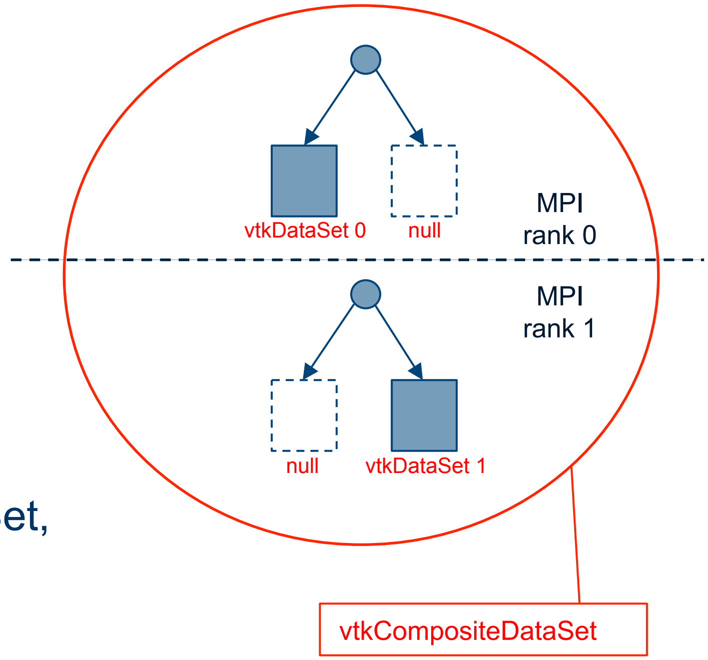
Distributed mesh based data in VTK

Composite Data

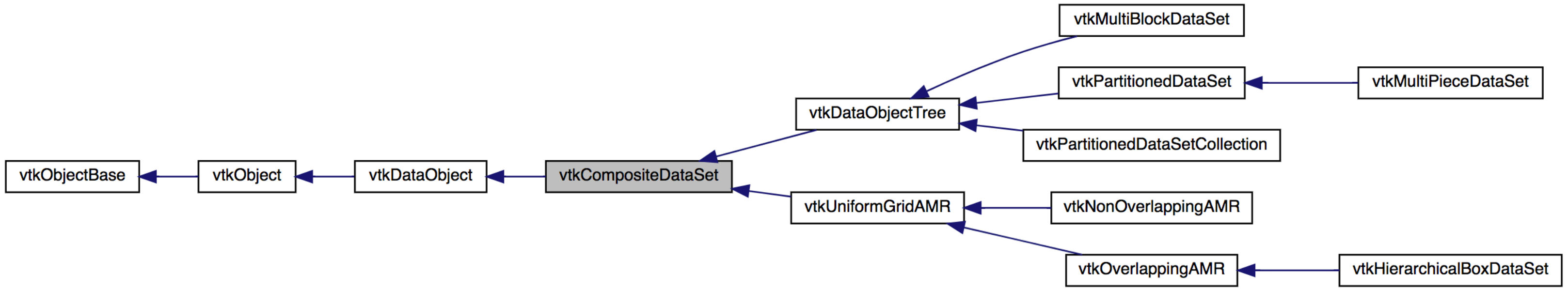
- Tree based data structures
- Think of as multi-block, blocks need not be Cartesian or rectangular
- Supports many blocks per rank
- Provides iterators to walk over local blocks
- Limited info about off rank blocks

Legacy Approach

- Each rank has a single instance of vtkDataSet, metadata identifies "piece" for unstructured, "extents" for Cartesian



vtkCompositeDataSet – Container for distributed data



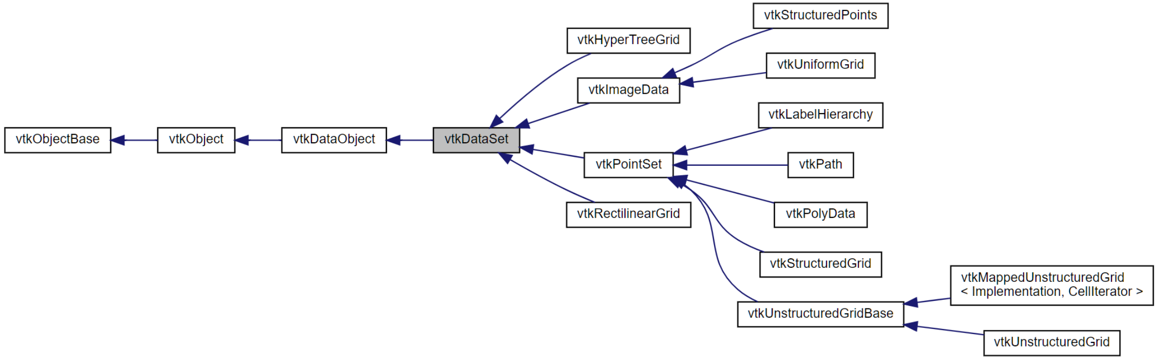
https://www.vtk.org/doc/nightly/html/classvtkCompositeDataSet.html

- Go to the above link. use the clickable class diagram to navigate / access documentation for the specific composite data object types
- Use `vtkCompositeDataIterator::NewIterator()` to get an iterator that can visit local blocks

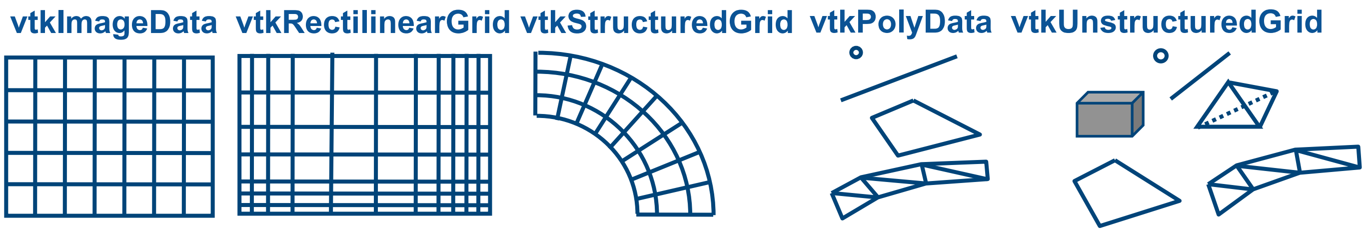
vtkCompositeDataIterator API

```
 // If SkipEmptyNodes is true, then nullptr(non-local) datasets will be skipped.
void SetSkipEmptyNodes (vtkTypeBool);

// Begin iterating over the composite dataset structure.
void InitTraversal ();

 // Begin iterating over the composite dataset structure.
void GoToFirstItem();

// Move the iterator to the next item in the collection.
void GoToNextItem();

//Test whether the iterator is finished with the traversal.
int IsDoneWithTraversal();

// Returns the current item.
vtkDataObject *GetCurrentDataObject();

// Flat index is an index to identify the data in a composite data set
unsigned int GetCurrentFlatIndex();
```

vtkDataSet – Leaves of the tree / legacy model

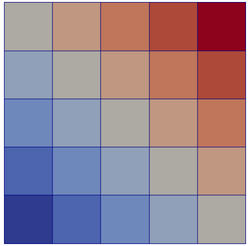


https://www.vtk.org/doc/nightly/html/classvtkDataSet.html

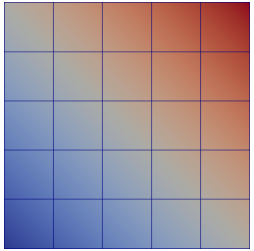


VTK's take on mesh based data

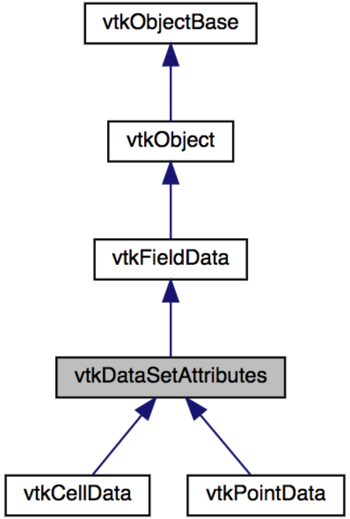
- Either point or cell centered, or no centering at all
  - `vtkPointData` – a collection of point centered arrays. Must have number of points elements
  - `vtkCellData` – a collection of cell centered arrays. Must have number of cells elements
  - `vtkFieldData` – a collection of arrays with no centering. Can be any lengtth
- Mesh/block dimensions are in units of points



Cell data



Point data



https://www.vtk.org/doc/nightly/html/classvtkDataSetAttributes.html

vtkDataArray – passing simulation data

- vtkFloatArray, vtkDoubleArray, vtkIntArray, etc are a façade hiding templates vtkAOSDataArrayTemplate<ValueTypeT>
- VTK's AOS type is the default for all arrays in VTK
- Supports zero copy, can take ownership of a pointer & free/delete when finished see XX::SetArray API
- Supports zero copy from alternative layouts, these are derived from vtkGenericDataArray<DerivedT, ValueTypeT>
  - eg SOA vtkSOADataArrayTemplate<ValueTypeT>

vtkDataArray – accessing data for analysis

- Supports accessing stored data via pointer
  - Avoid ~~`XX::GetVoidPointer`~~, this may make a deep copy if the layout is not VTK's default layout
  - Downcast to SOA or AOS type, `vtkAOSDataArrayTemplate<ValueTypeT>` or `vtkSOADataArrayTemplate<ValueTypeT>` and used typed API `XX::GetPointer`
  - If down casting fails, for instance a new layout is added, fall back to `XX::GetVoidPointer`
- Or use VTK's API for accessing tuples/values, these often are OK given modern optimizing compilers

Speed & Efficiency

zero copy layouts provide pointer equivalent performance

- Array of Structures (AOS)
  — single array with components interleaved

v=	x1	y1	z1	x2	y2	z2	…	xn	yn	zn

```
// VTK's default is AOS, no need to use vtkAOSDataArrayTemplate
vtkDoubleArray *aos = vtkDoubleArray::New();
aos->SetNumberOfComponents(3);
aos->SetArray(v, 3*n, 0);
```

- Structure of Arrays (SOA)
  — each component in its own arrays

vx=	x1	x2	x3	…	xn
vy=	y1	y2	y3	…	yn
vz=	z1	z2	z3	…	zn

```
// use the new SOA class
vtkSOADataArrayTemplate<double> *soa =
   vtkSOADataArrayTemplate<double>::New();
soa->SetNumberOfComponents(3);
soa->SetArray(0, vx, n, true);
soa->SetArray(1, vy, n);
soa->SetArray(2, vz, n);
```

Zero copy to VTK Arrays

Memory Layouts in VTK

- Array of Structures (AOS)

— Vectors/Tensors are a single array with components interleaved

v= | x1 | y1 | z1 | x2 | y2 | z2 | … | xn | yn | zn |

- Structure of Arrays (SOA)

— Each vector/tensor component in its own arrays

vx= | x1 | x2 | x3 | … | xn |

vy= | y1 | y2 | y3 | … | yn |

vz= | z1 | z2 | z3 | … | zn |

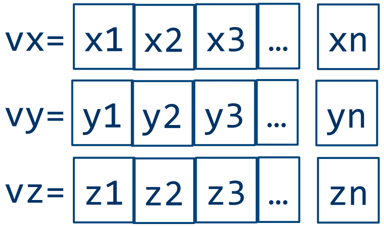
Zero copy with AOS (Array of Structures)

```
// VTK's default is AOS, no need to use vtkAOSDataArrayTemplate
vtkDoubleArray *aos = vtkDoubleArray::New();
aos->SetNumberOfComponents(3);
aos->SetArray(v, 3*nxy, 0);
aos->SetName("aos");

// add the array as usual
im->GetPointData()->AddArray(aos);
im->GetPointData()->SetActiveVectors("aos");

// give up our reference
aos->Delete();
```

v=	x1	y1	z1	x2	y2	z2	…	xn	yn	zn

Zero copy with SOA (structure of arrays)

```
// use the SOA class
vtkSOADataArrayTemplate<double> *soa = vtkSOADataArrayTemplate<double>::New();
soa->SetNumberOfComponents(3);
soa->SetArray(0, vx, nxy, true);
soa->SetArray(1, vy, nxy);
soa->SetArray(2, vz, nxy);
soa->SetName("soa");

// add to the image as usual
im->GetPointData()->AddArray(soa);
im->GetPointData()->SetActiveVectors("soa");

// git rid of our reference
soa->Delete();
```

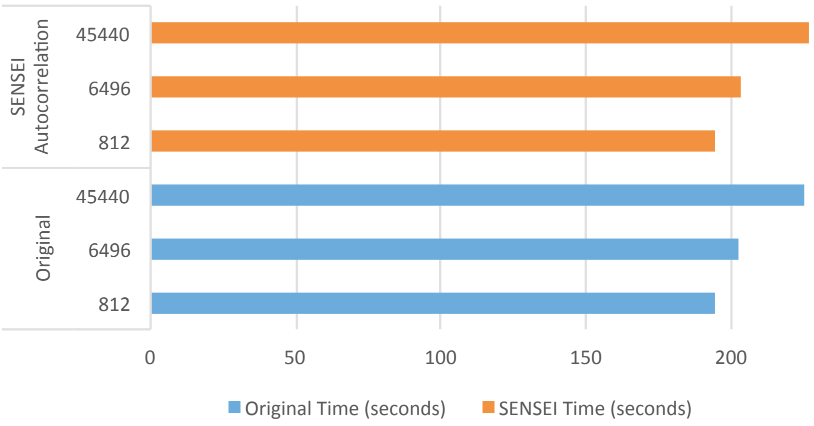


Overhead due to VTK data model

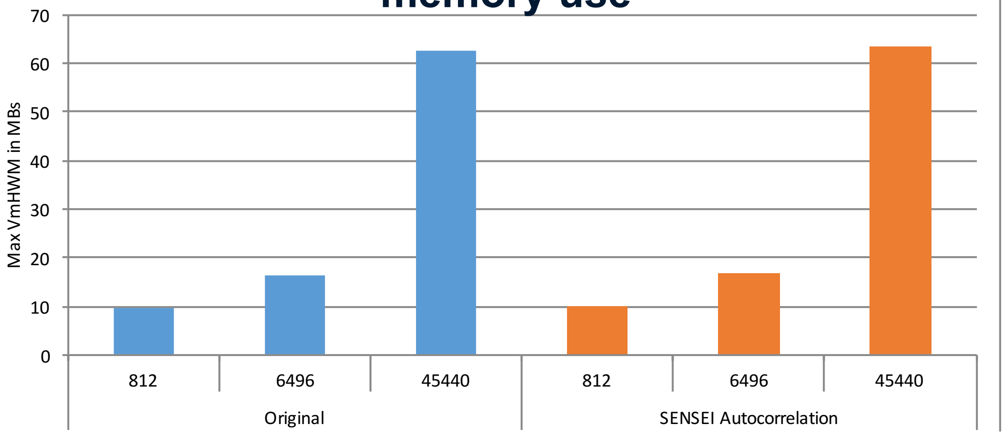
Run *Original* and *Baseline* configs, 3 levels of concurrency: 1K, 6K, 45K

- Original: subroutine called, Baseline: through SENSEI bridge

run time



memory use



Performance Analysis, Design Considerations, and Applications of Extreme-scale In Situ Infrastructures. SC16

Zero copy demo

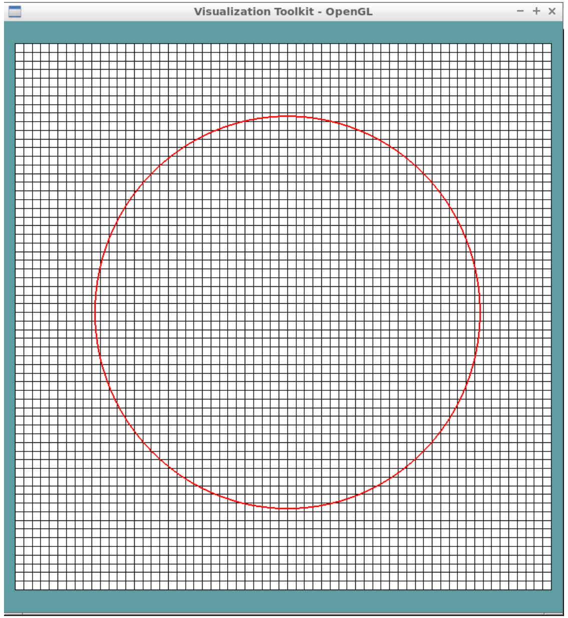
This demo shows how to do zero copy using AOS and SOA layouts

Zero-copy passes a vector field to the VTK stream line tracer

Vector field is tangent to concentric circles on a domain of -1 to 1 in x and y

Running the demo

```
$ cd ~/sensei_insitu/demos/sc18/zero_copy
$ vim zero_copy.cpp # view source code (optional)
$ ./zero_copy.sh
```



BERKELEY LAB

Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

SENSEI insitu

Instrumenting Data Sources and Endpoints with SENSEI

BERKELEY LAB

Argonne NATIONAL LABORATORY

OAK RIDGE National Laboratory

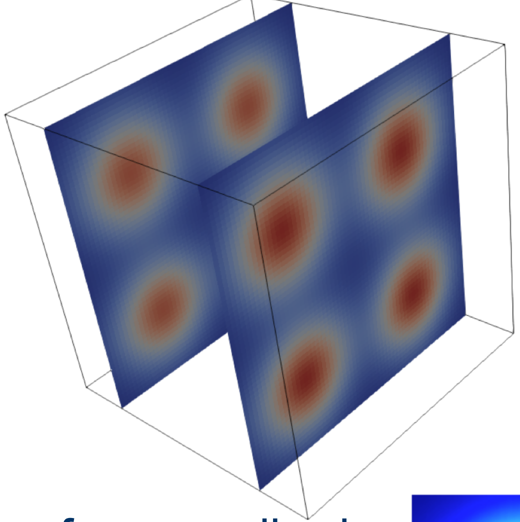
Intelligent Light

Kitware

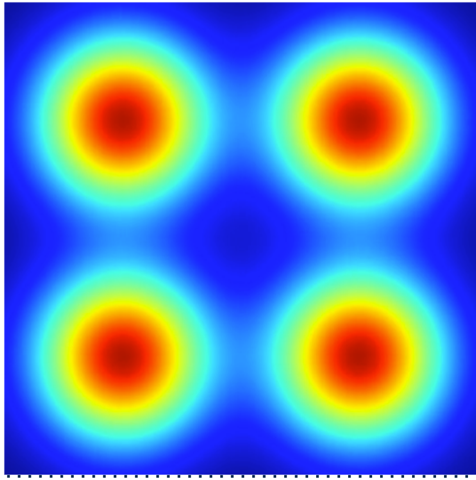
Instrumentation tasks

1. Data
   - Decide if you can use `sensei::VTKDataAdaptor`
   - Or write an adaptor derived from `sensei::DataAdaptor`
2. Analysis
   - Decide if you can use existing analyses: Libsim, Catalyst, Adios, etc
   - And/Or implement new analyses derived from `sensei::AnalysisAdaptor`
3. Bridge
   - Implement Initialize, Compute, and Finalize methods/functions
   - Instrument the simulation to call the bridge code at the right times

Oscillator miniapp overview



- MPI based C++ code that simulates a collection of periodic, damped, or decaying oscillators over a Cartesian grid.
- Unstructured grid also supported
- Each oscillator is convolved with a Gaussian of a prescribed width
- Can randomly place particles and advect them using an analytical velocity field
- Executable inputs are oscillator parameters, time resolution, length of the simulation, grid dimensions, grid partitioning, and number of random particles to generate



Instrumenting the oscillator mini-app to use SENSEI

- Create a class that derives from `sensei::DataAdaptor` and implements:
  - `virtual int GetNumberOfMeshes(unsigned int &numMeshes) = 0;`
  - `virtual int GetMeshName(unsigned int id, std::string &meshName) = 0;`
  - `virtual int GetMesh(const std::string &meshName, bool structureOnly, vtkDataObject *&mesh) = 0;`
  - `virtual int GetNumberOfArrays(const std::string &meshName, int association, unsigned int &numberOfArrays) = 0;`
  - `virtual int GetArrayName(const std::string &meshName, int association, unsigned int index, std::string &arrayName) = 0;`
  - `virtual int AddArray(vtkDataObject* mesh, const std::string &meshName, int association, const std::string &arrayName) = 0;`
  - `virtual int ReleaseData() = 0;`

Creating the VTK grid – GetMesh() method

```
int DataAdaptor::GetMesh(const std::string &meshName, bool structureOnly, vtkDataObject *&mesh)
{
    if (meshName != "mesh" && meshName != "ucdmesh" && meshName != "particles")
    {
        SENSEI_ERROR("the miniapp provides meshes named \"mesh\", \"ucdmesh\", and \"particles\""
                     " you requested \"" << meshName << "\"")
        return -1;
    }

    DInternals& internals = (*this->Internals);

    if (meshName == "ucdmesh")
    {
        .....
    }
    else if (meshName == "mesh")
    {
        if (!internals.Mesh)
        {
            internals.Mesh = vtkSmartPointer<vtkMultiBlockDataSet>::New();
            internals.Mesh->SetNumberOfBlocks(static_cast<unsigned int>(internals.CellExtents.size()));
            for (size_t cc=0; cc < internals.CellExtents.size(); ++cc)
                internals.Mesh->SetBlock(static_cast<unsigned int>(cc), this->GetBlockMesh(cc));
        }
        mesh = internals.Mesh;
    }
    else if (meshName == "particles")
    {
        ....
    }
    return 0;
}
```

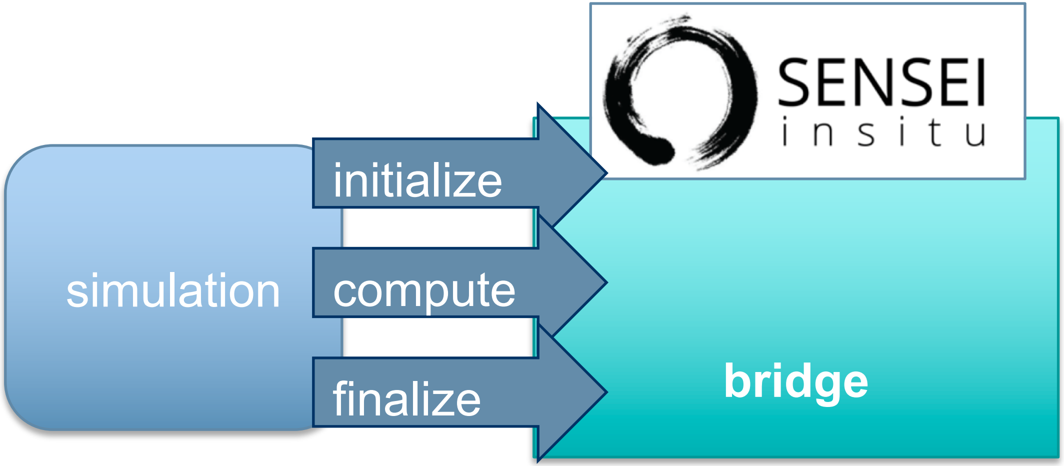
Creating the VTK cell data – AddArray() method

```
int DataAdaptor::AddArray(vtkDataObject* mesh, const std::string &meshName, int association, const std::string &arrayName)
{
    DInternals& internals = (*this->Internals);
    vtkMultiBlockDataSet* md = vtkMultiBlockDataSet::SafeDownCast(mesh);

    if ((meshName == "mesh" || meshName == "ucdmesh") && arrayName == "data" &&
        association == vtkDataObject::FIELD_ASSOCIATION_CELLS)
    {
        for (unsigned int cc=0, max=md->GetNumberOfBlocks(); cc < max; ++cc)
        {
          ....
        }
    }
    else if (meshName == "particles" && association == vtkDataObject::FIELD_ASSOCIATION_POINTS &&
             (arrayName == "uniqueGlobalId" || arrayName == "velocity" || arrayName == "velocityMagnitude"))
    {
        ....
    }
#ifndef NDEBUG
    else
    {
        SENSEI_ERROR("the miniapp provides a cell centered array named \"data\" "
                     "on meshes named \"mesh\" or \"ucdmesh\"; or point centered arrays named "
                     "\"uniqueGlobalId\", \"velocity\" and \"velocityMagnitude\" on a mesh named \"particles\"")
        return -1;
    }
#endif

    return 0;
}
```

Implementing the bridge to SENSEI

Typically 3 calls:

- Initialize()
  - Set the DataAdaptor
  - Initialize DataTimeStep
  - Specify what analysis will be done. For the Oscillator we use the ConfigurableAnalysis class.
- Compute()
  - For the Oscillator we do this with two calls: set_data() / set_particles() and analyze(), so that SENSEI may be disabled in benchmarks
- Finalize()



Initializing the bridge

```
void initialize(MPI_Comm comm, size_t window, size_t nblocks,
                size_t n_local_blocks, int domain_shape_x, int domain_shape_y,
                int domain_shape_z, int* gid, int* from_x, int* from_y, int* from_z,
                int* to_x, int* to_y, int* to_z, int* shape, int ghostLevels,
                const std::string& config_file)
{
    timer::MarkEvent mark("oscillators::bridge::initialize");

    (void)window;
    (void)comm;

    GlobalDataAdaptor = vtkSmartPointer<oscillators::DataAdaptor>::New();
    GlobalDataAdaptor->Initialize(nblocks, shape, ghostLevels);
    GlobalDataAdaptor->SetDataTimeStep(-1);

    for (size_t cc=0; cc < n_local_blocks; ++cc)
    {
        GlobalDataAdaptor->SetBlockExtent(gid[cc],
                                          from_x[cc], to_x[cc], from_y[cc], to_y[cc],
                                          from_z[cc], to_z[cc]);
    }

    int dext[6] = {0, domain_shape_x, 0, domain_shape_y, 0, domain_shape_z};
    GlobalDataAdaptor->SetDataExtent(dext);

    GlobalAnalysisAdaptor = vtkSmartPointer<sensei::ConfigurableAnalysis>::New();
    GlobalAnalysisAdaptor->Initialize(config_file);
}
```

Executing the in situ

```
void set_data(int gid, float* data)
{
    GlobalDataAdaptor->SetBlockData(gid, data);
}


void set_particles(int gid, const std::vector<Particle> &particles)
{
    GlobalDataAdaptor->SetParticles(gid, particles);
}


void analyze(float time)
{
    GlobalDataAdaptor->SetDataTime(time);
    GlobalDataAdaptor->SetDataTimeStep(GlobalDataAdaptor->GetDataTimeStep() + 1);
    GlobalAnalysisAdaptor->Execute(GlobalDataAdaptor.GetPointer());
    GlobalDataAdaptor->ReleaseData();
}
```

Finalizing the bridge

```
void finalize(size_t k_max, size_t nblocks)
{
    (void)k_max;
    (void)nblocks;

    GlobalAnalysisAdaptor->Finalize();

    GlobalAnalysisAdaptor = nullptr;
    GlobalDataAdaptor = nullptr;
}
```

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

VisIt

Data Extracts with VisIt/Libsim

BERKELEY LAB

Argonne NATIONAL LABORATORY

OAK RIDGE National Laboratory

Intelligent Light

Kitware

Libsim puts VisIt in situ

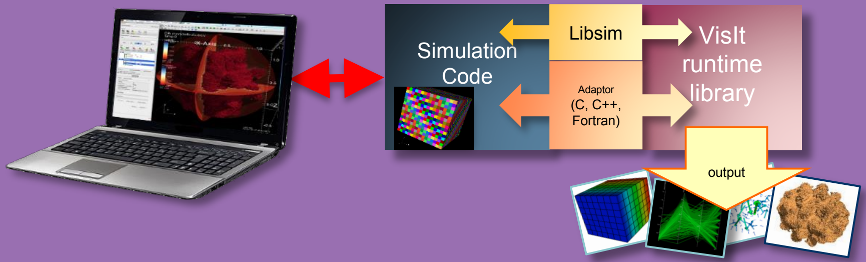
- VisIt provides Libsim, a library that simulations may use to let VisIt connect and access their data
- Avoids I/O and data movement
- Supports automated data product generation
- Also supports user-driven exploration of simulation data

VisIt

- Versatile open source software for visualizing and analyzing extreme scale simulation datasets

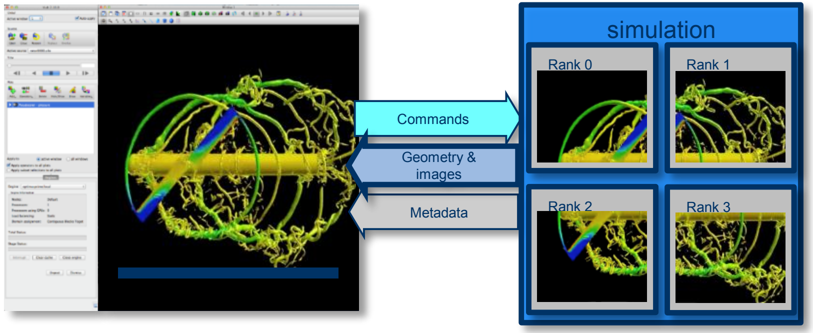
Libsim

- Enables simulations to perform data analysis and visualization in situ by applying VisIt algorithms to data.

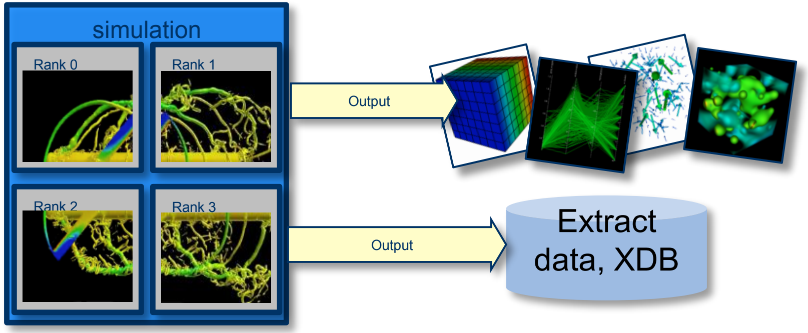


Libsim enables flexible workflows

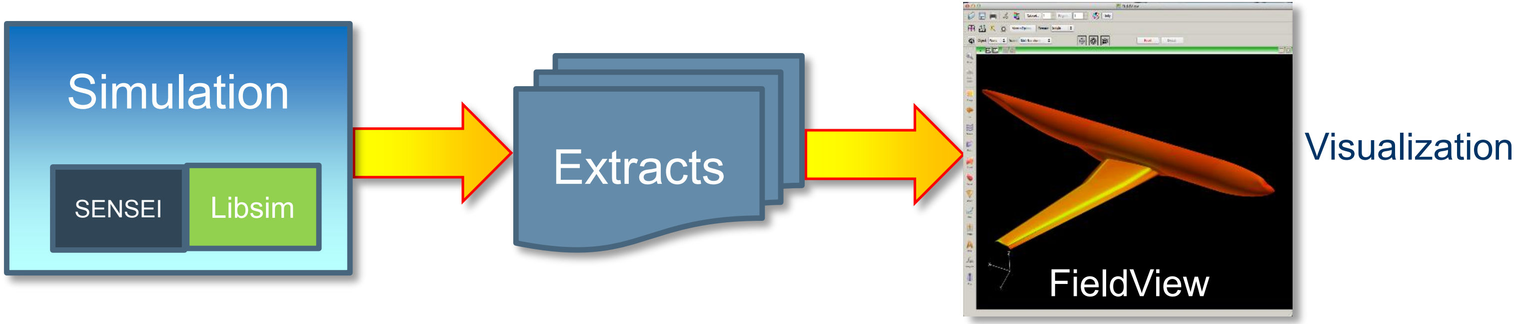
- Use the VisIt GUI to connect to your simulation and explore!
- Simulations are like any other data source



- Create automated routines to generate data in batch
- Program directly using Libsim
- Use VisIt session files



In Situ Extracts Workflow



Extracts contain the “interesting” stuff from the simulation

– Extracts are orders of magnitude smaller than volume data *(avoid I/O bottleneck)*

– **Provides enough geometry and field data that enables useful post-hoc exploration**

– Surface extracts stored in FieldView XDB format, VTK format, etc.

Flexible Extract Export with SENSEI

- Hard-coding plots and extracts limits flexibility
- SENSEI XML input file can select plots for extract creation and for rendering
  - Provides hints to Libsim
  - Specifies extracts, variables, files to write
  - Pass session file
  - Pass hints to connect interactively

```
<sensei>
 <!-- Libsim: Set up plots using session, save VTK files in batch  -->
  <analysis type="libsim"
          frequency="5"
          visitdir="/usr/common/software/sensei/visit"
          mode="batch"
          session="oscillator-ucdmesh.session"
          operation="export"
          filename="iso-ghost%ts"
          enabled="1"/>
</sensei>
```

Specifies which plots, output files, etc.

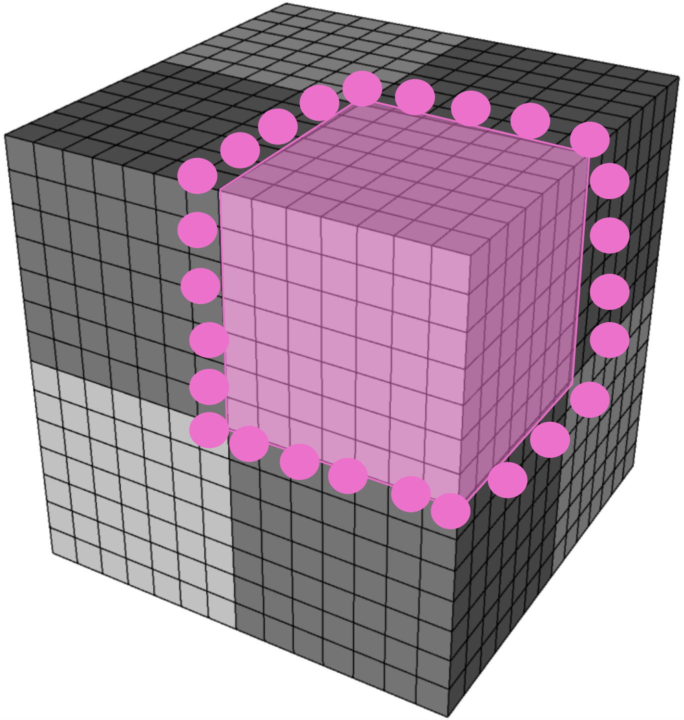
```
<sensei>
 <!-- Libsim Iso ucdmesh: connect VisIt interactively -->
  <analysis type="libsim"
          frequency="10"
          visitdir="/usr/common/software/sensei/visit"
          mode="interactive,paused"
          enabled="1"/>
</sensei>
```

SENSEI's Libsim Integration has Advanced

- Supports interactive connections using VisIt GUI
- Supports ghost data
- Supports unstructured meshes
- Use VisIt session files to produce visualizations in batch
  - Session files record all of the setup to make a nice visualization
  - Workflow: Connect interactively with VisIt -> set up plots -> save a session file -> rerun in batch using the session file to specify plots

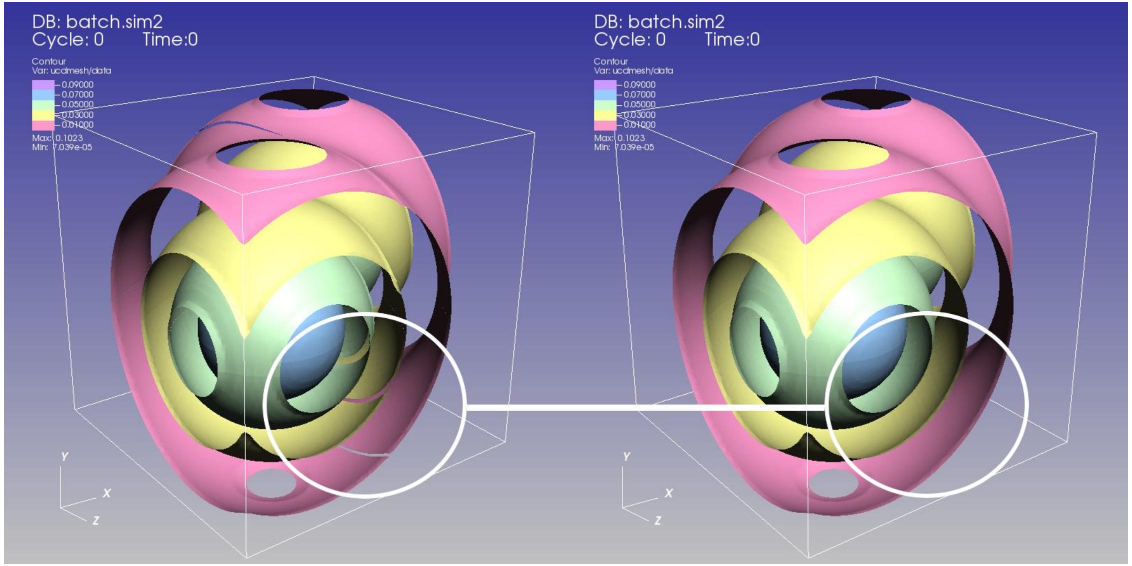
Ghost Data

- Simulations exchange ghost data (additional layers of cells/nodes) along processor boundaries to make sure enough information is present to calculate quantities that need neighbor values
- Ghost Data are marked as such so they can be used then they are needed and skipped when appropriate *(e.g. avoid double-counting in histogram)*



Ghost Data in Oscillators Mini-app

- SENSEI's Oscillators mini-app now supports ghost cells
- Enables isosurfaces of cell data to be continuous across domain boundaries
- Enabled using the –g # command line argument to generate a user-specified number of ghost levels



Isosurfaces without (left) and with (right) ghost cells

mpirun –np 4 oscillators –g 2 –f oscillator.xml –t 0.1 samples.osc

SENSEI API for Ghost Data

- The VTK data representing meshes and fields need to contain extra cells/nodes if ghost data are used
- Ghost data must also be marked as ghost
- SENSEI adds new methods in `sensei::DataAdaptor` that enables the adaptor to mark cells/nodes as ghost data
  - `virtual int GetMeshHasGhostCells(const std::string &meshName, int &nLayers);`
  - `virtual int AddGhostCellsArray(vtkDataObject* mesh, const std::string &meshName);`
  - `virtual int GetMeshHasGhostNodes(const std::string &meshName, int &nLayers);`
  - `virtual int AddGhostNodesArray(vtkDataObject* mesh, const std::string &meshName);`
  - The default implementations of these methods in indicate that no ghost data are present

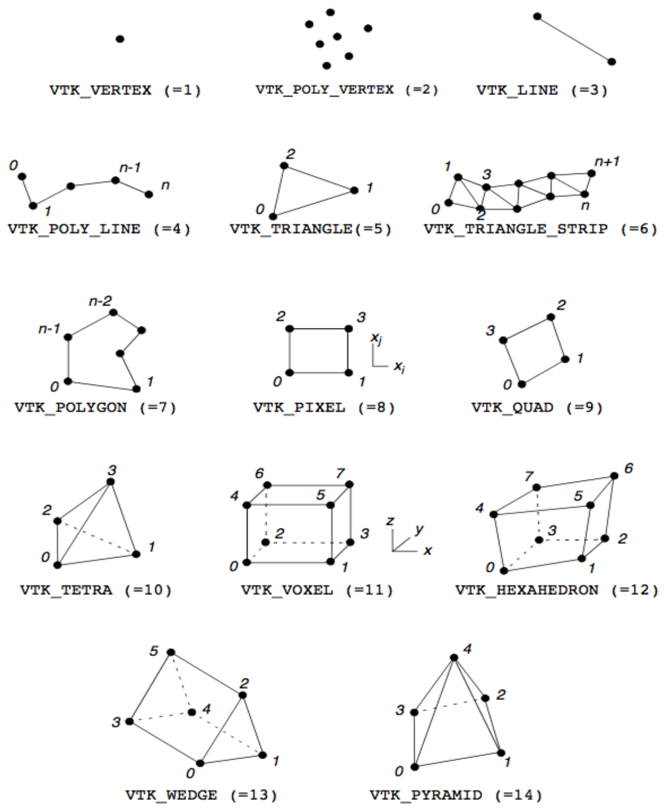
Ghost Data Encoding

- Ghost data arrays are *vtkUnsignedCharArray* objects that contain values for each cell or node
- The allowable values follow the conventions used in VisIt and ParaView
  - The array name must be *“vtkGhostType”*
  - 1=Ghost, 0=Real

```
//------------------------------------------------------------------------------
int DataAdaptor::GetMeshHasGhostCells(const std::string &/*meshName*/,
  int &nLayers)
{
  DInternals& internals = (*this->Internals);
  nLayers = internals.ghostLevels;
  return 0;
}

//------------------------------------------------------------------------------
int DataAdaptor::AddGhostCellsArray(vtkDataObject *mesh, const std::string &meshName)
{
  int retVal = 1;
  DInternals& internals = (*this->Internals);
  vtkMultiBlockDataSet* md = vtkMultiBlockDataSet::SafeDownCast(mesh);
  for (unsigned int cc=0, max=md->GetNumberOfBlocks(); cc < max; ++cc)
    {
    vtkSmartPointer<vtkImageData>& blockMesh = internals.BlockMesh[cc];
    vtkCellData *cd = (blockMesh? blockMesh->GetCellData() : NULL);
    if (cd != NULL)
      {
      if (cd->GetArray("vtkGhostType") == NULL)
        {
        vtkDataArray *g = CreateGhostCellsArray(cc); // Make vtkUnsignedCharArray.
        cd->AddArray(g);
        g->Delete();
        }
      retVal = 0;
      }
    }
  return retVal;
}
```

Unstructured Grid Support

- SENSEI represents unstructured grids using *vtkUnstructuredGrid*
  - Contains a set of points
  - Contains cells defined by connectivity (indices into the points)
- SENSEI's Libsim integration can now pass unstructured grids through to VisIt



Some of the unstructured grid cell types supported in VTK

Unstructured Grid Support in Oscillator

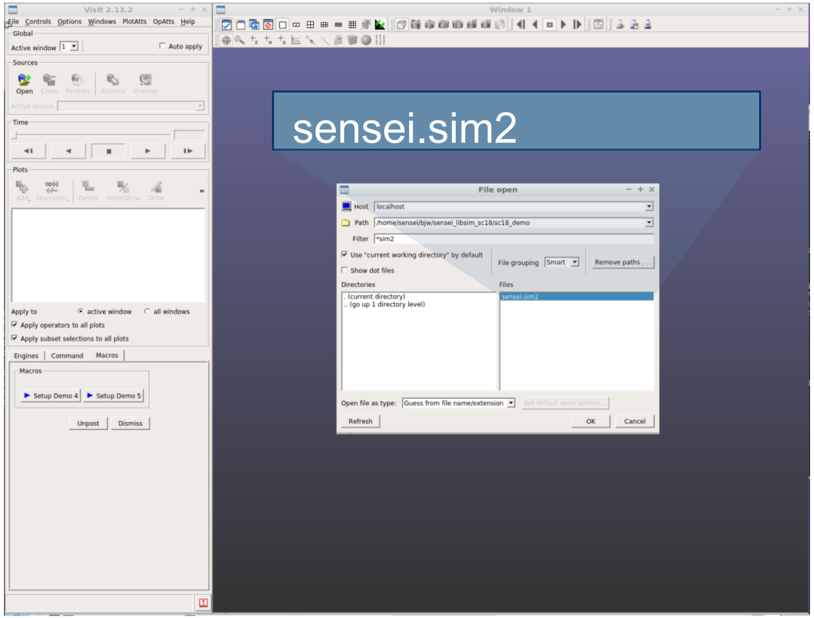
- Oscillator exposes a second mesh called *ucdmesh* that is an unstructured representation of its normal structured mesh
- The same fields are returned for both the structured and unstructured meshes

Adaptor Changes:

- GetNumberOfMeshes() returns 2
- GetMeshNames() returns “mesh” for index 0 and “ucdmesh” for index 1.
- GetMesh() returns the vtkUnstructuredGrid representation of the data for index 1

Connecting to a SENSEI simulation using VisIt

- Enable Libsim analysis in the SENSEI XML input file
  - Set the mode to "interactive" or "interactive,paused"
  - The paused mode blocks the simulation until VisIt connects and lets the simulation proceed using the controls in VisIt's Simulation window
- Libsim will write a file called *sensei.sim2*
- Open *sensei.sim2* in VisIt to connect

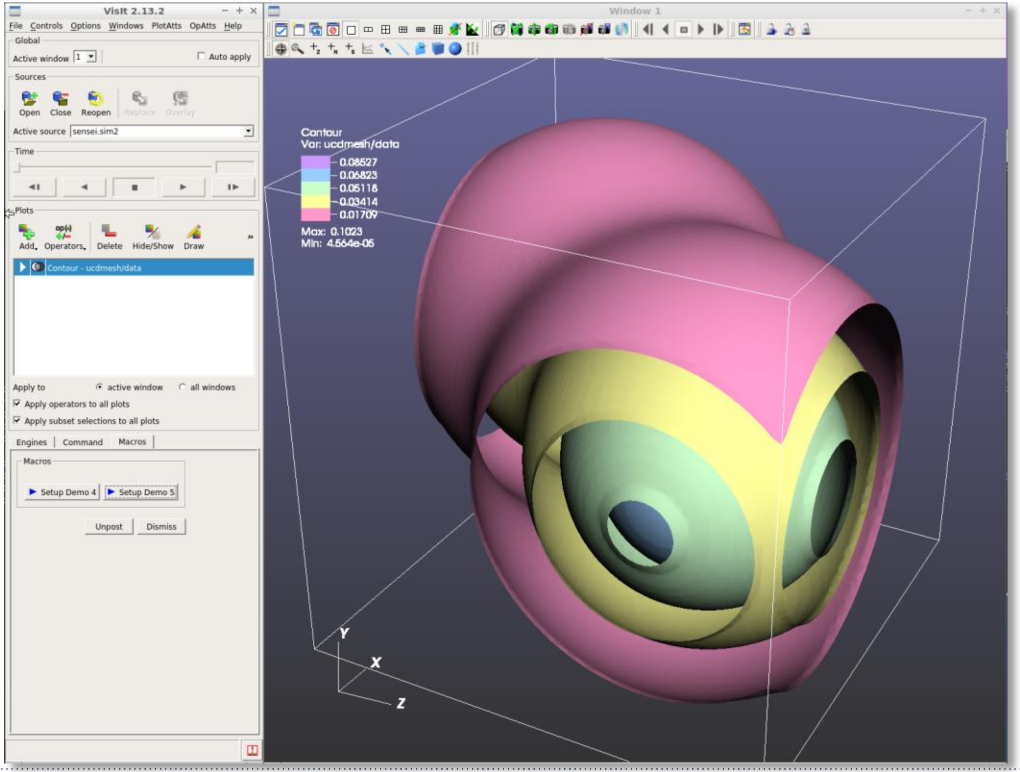


Connect interactively using VisIt GUI by opening sensei.sim2

Libsim Demo

Live Demo run on VM, or VM + cori.nersc.gov

- Run oscillator mini-app
- Show effects of ghost cells
- Use session files to produce extracts
- Run VisIt interactively
- Interactively connect to oscillator simulation



Libsim Demo: Procedure

- Run all on the VM
- Run using a combination of the VM and cori.nersc.gov
- Replace *USERNAME* with the token account login

SENSEI VM

```
%
%
% cd sensei_insitu/demos
% cd sc18/visit_libsim
% ./demo.sh 1 USERNAME
% ./demo.sh 2 USERNAME
% ./demo.sh 3 USERNAME
% ./demo.sh 4 USERNAME
% ./demo.sh 5 USERNAME
% ./demo.sh 6 USERNAME
```

Cori.nersc.gov

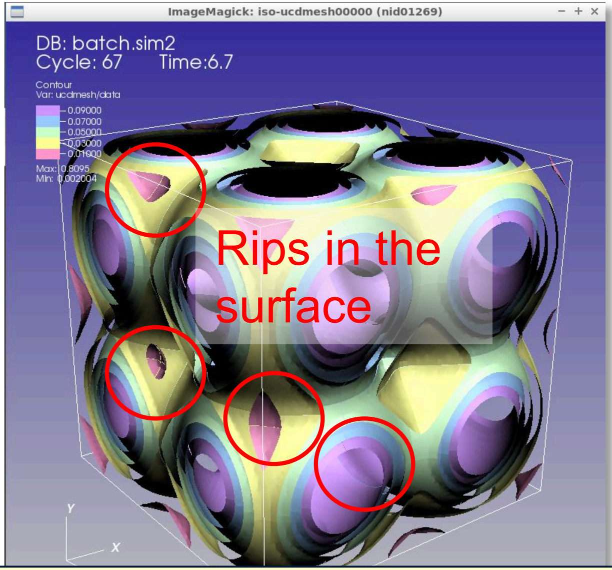
```
% cd /project/projectdirs
% cd m636
% cd sensei_insitu/demos
% cd sc18/visit_libsim
% ./demo.sh 1 USERNAME
% ./demo.sh 2 USERNAME
% ./demo.sh 3 USERNAME
```

If running on Cori, return to VM to run steps 4,5,6

*NOTE: when running on cori, the demo script will tell you to run an **salloc** command to allocate a node.*

Libsim Demo: Oscillator without ghost cells

- This part of the demo runs oscillator without ghost cells and renders pictures using a VisIt session file
- This can run in the VM or on Cori
- If running on Cori, run the salloc command printed by the demo.sh command and run again



% ./demo 1 USERNAME ⇦ Run oscillator, render images

% ./demo 2 USERNAME ⇦ Display images

Libsim Demo: Oscillator with ghost cells

- This part of the demo runs oscillator with ghost cells and saves isosurface extracts
- VisIt is then used to visualize the extracts
- Step 3 can run in the VM or on Cori
  - Step 3 writes out the directory where files are saved to the console
- Step 4 must be run in the VM

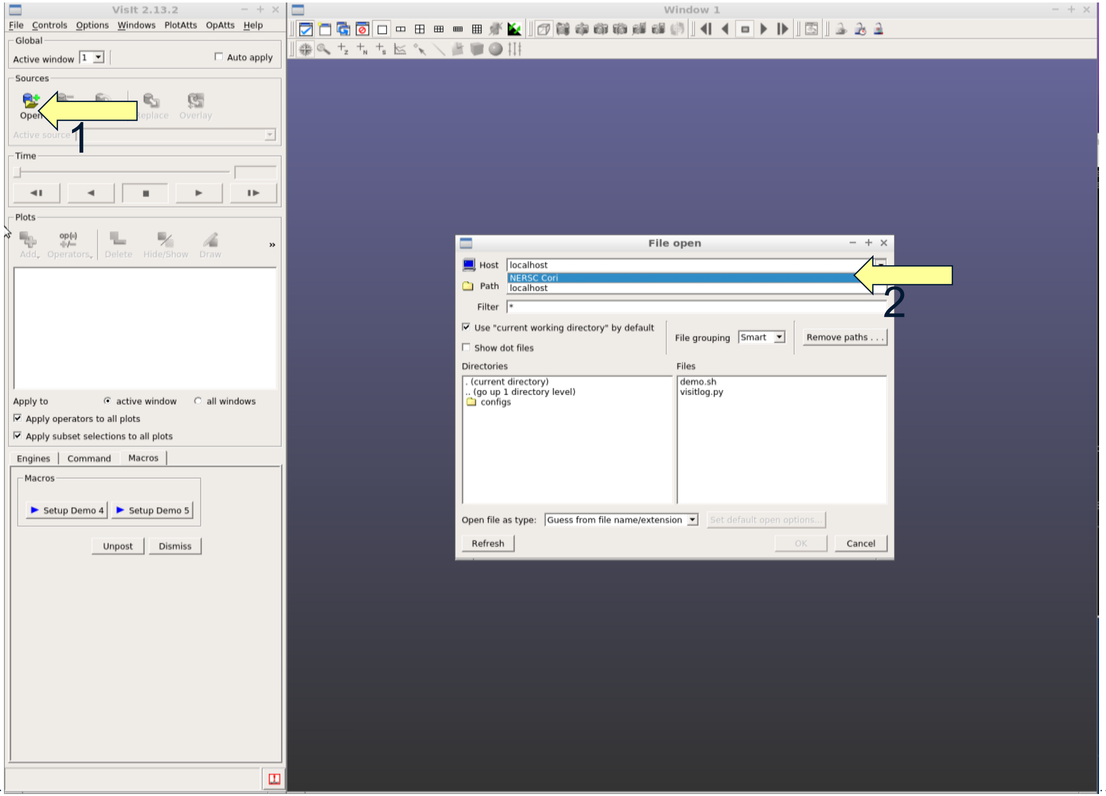
```
% ./demo 3 USERNAME
% ./demo 4 USERNAME
```

Run oscillator, make extracts

Open VisIt GUI

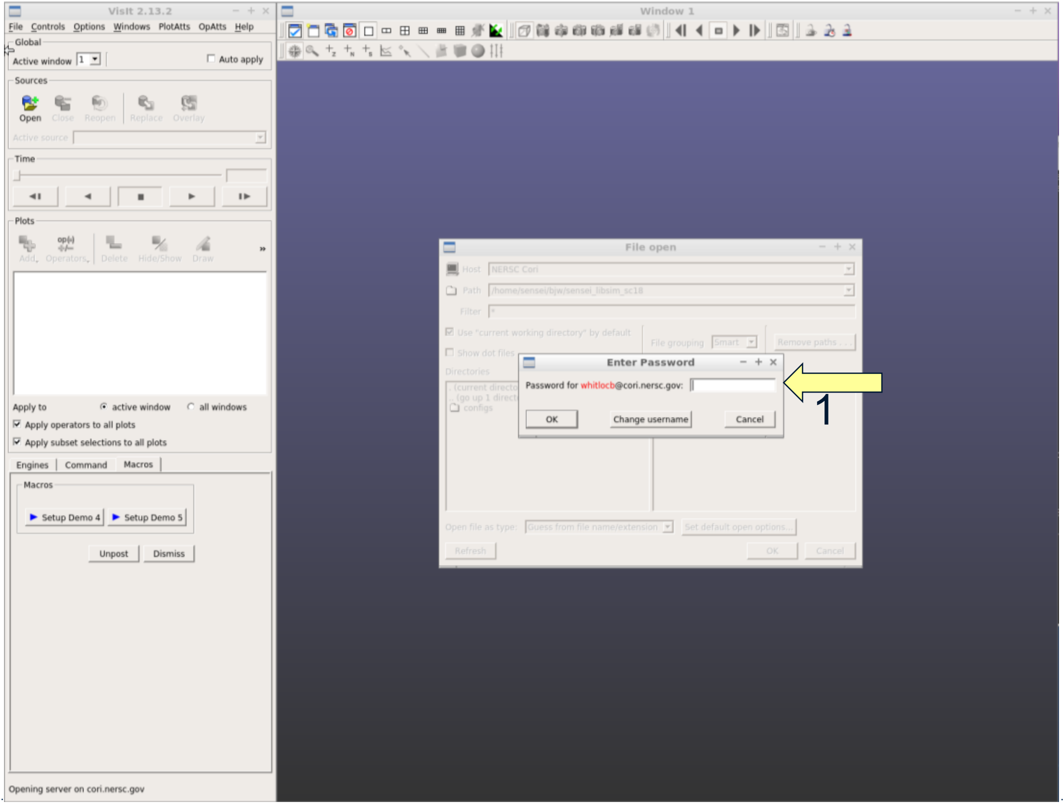
Libsim Demo: Client Server to Cori

- Step 4 opens the VisIt GUI
- Click the Open button in the Main window
- If Step 3 ran on Cori, select “NERSC Cori” from the host list to initiate a connection to Cori
- If Step 3 ran in the VM, skip the Cori only sections



Libsim Demo: Client Server to Cori - Password

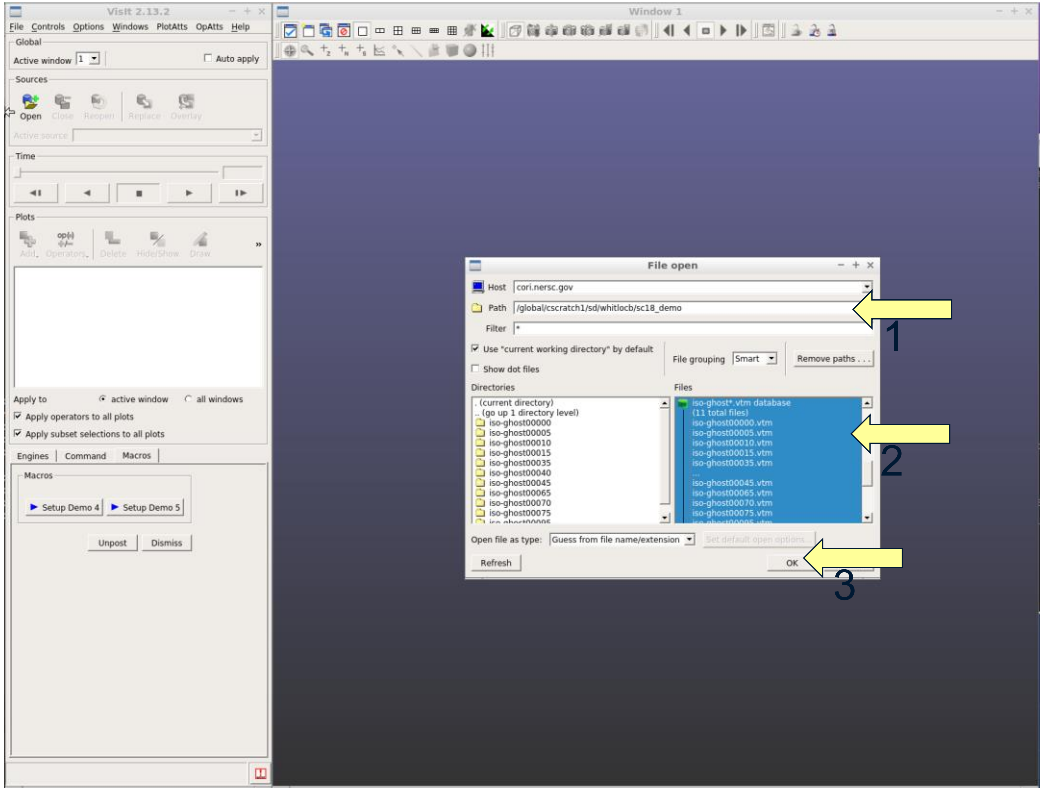
- When connecting to Cori, enter your Cori account password in VisIt's password window



Cori only

Libsim Demo: Select Files

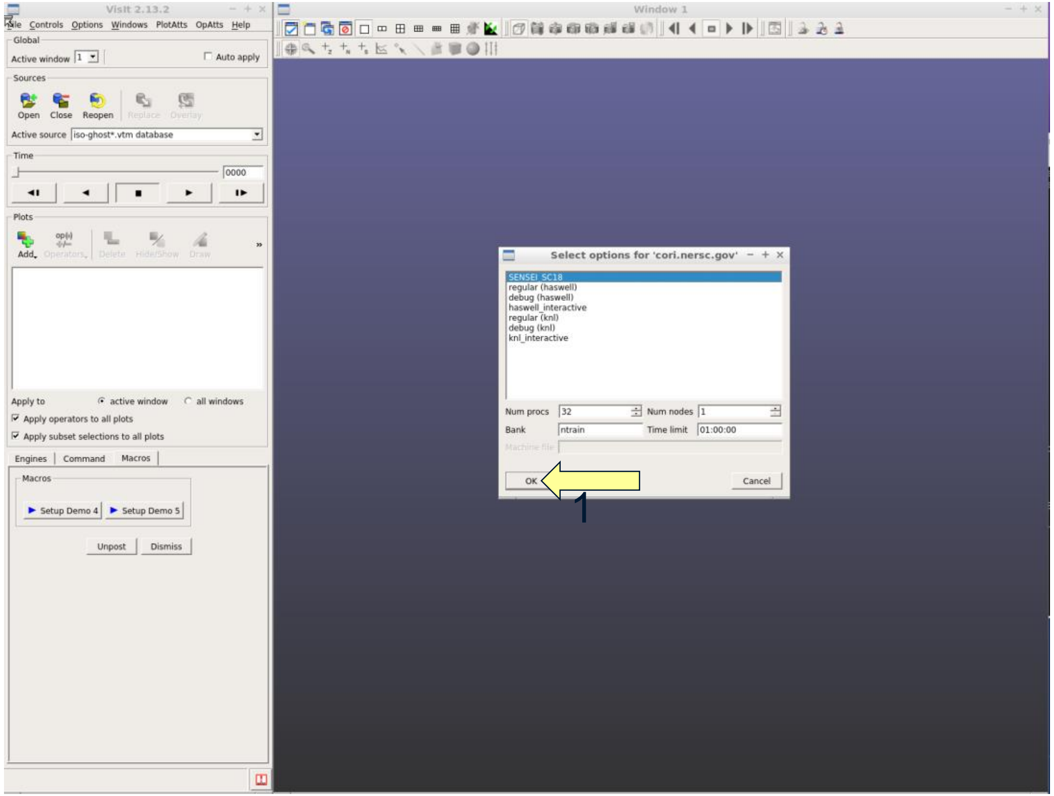
- Once connected, paste the directory name containing the files into the File Selection Dialog's **path** and press Enter
- Click on the "iso-ghost*.vtm" database
- Click OK



Libsim Demo: Engine Chooser

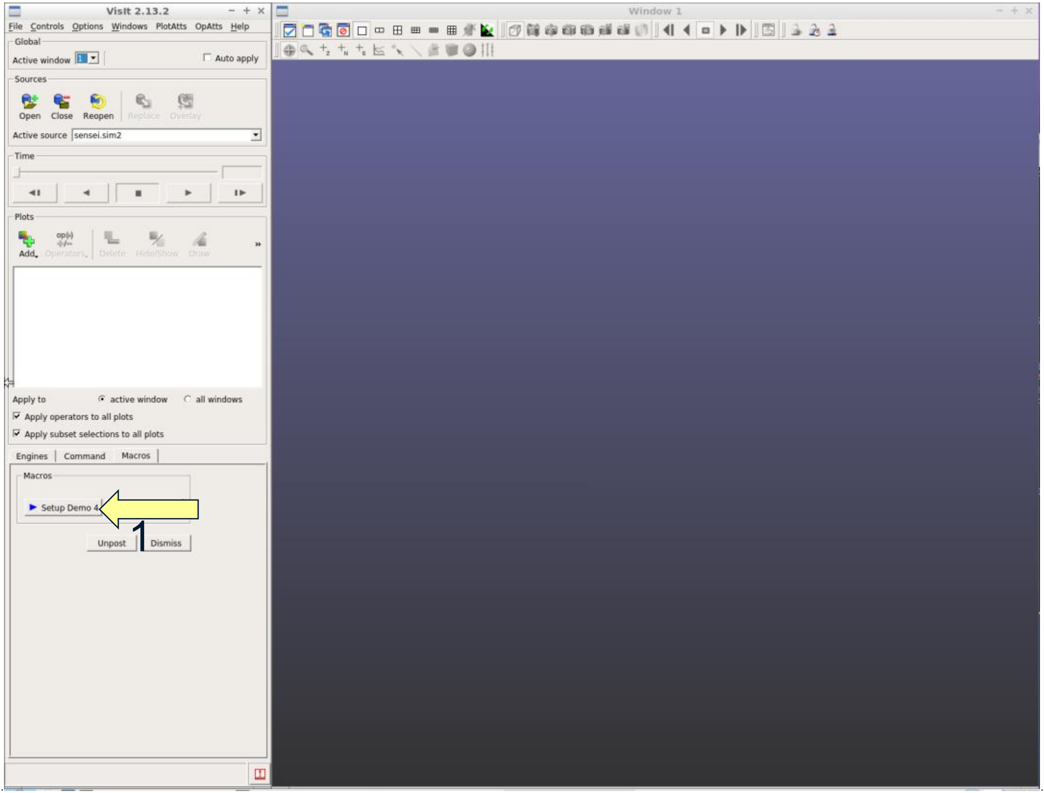
- If opening data that reside on Cori, VisIt will prompt you which *host profile* should be used to launch the VisIt compute engine
- The *SENSEI_SC18* profile should be selected so click OK

Cori only



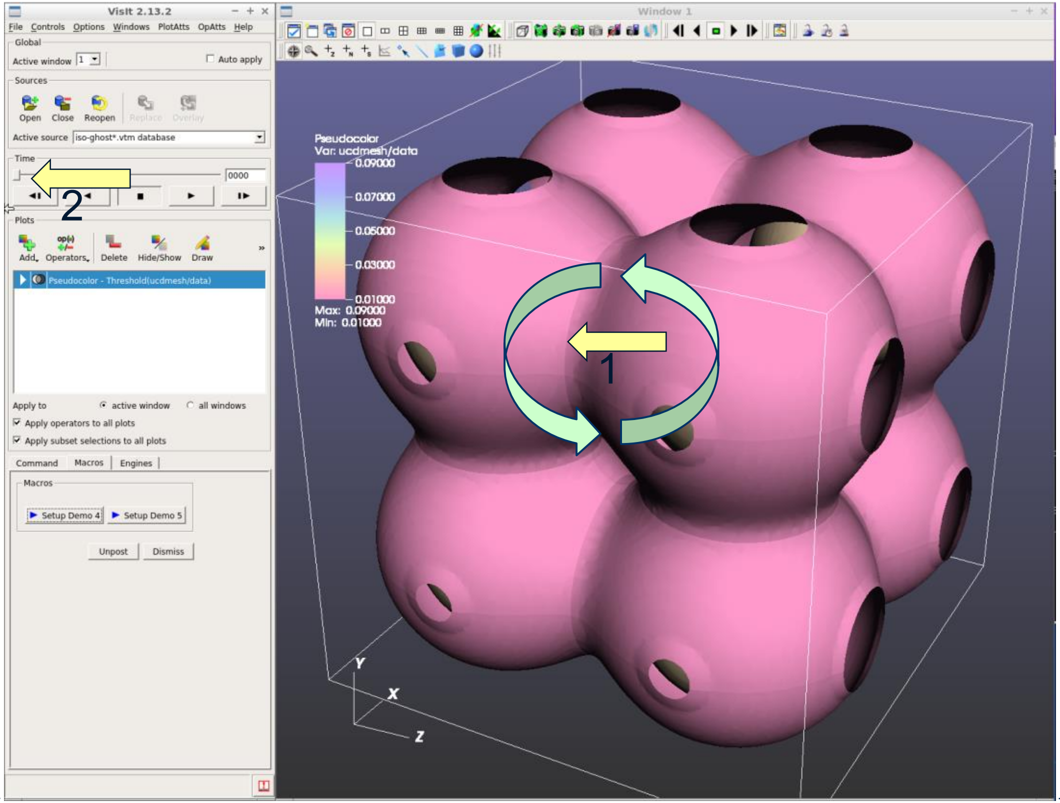
Libsim Demo: Set up Plots

- To set up plots based on the VTK extracts that SENSEI saved, click the *“Setup Demo 4”* button



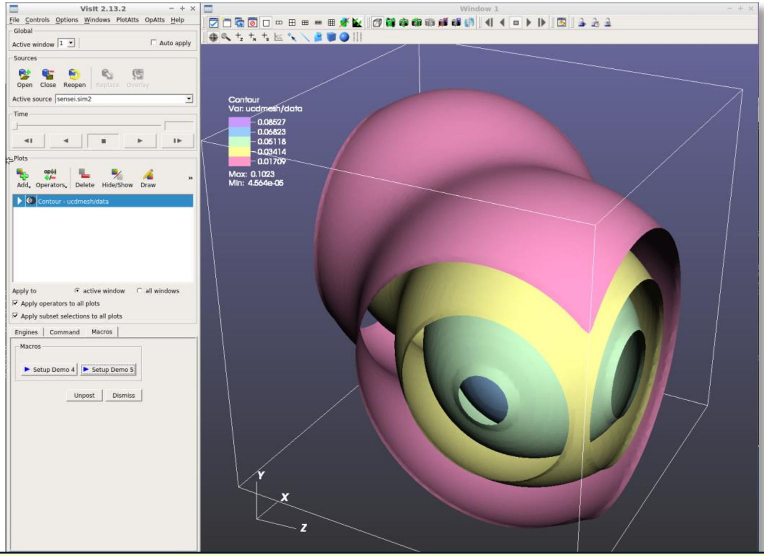
Libsim Demo: Interact with Plots

- When plots appear, note how the surfaces do not have gaps at domain boundaries
- Change the view by clicking/dragging on the plots
- Move the time slider
- Quit VisIt



Libsim Demo: Connect Interactively to Oscillator

- This part of the demo runs oscillator with ghost cells and waits for VisIt to connect
- We will plot data form oscillator interactively and watch it evolve
- Step 5 must be run in the VM

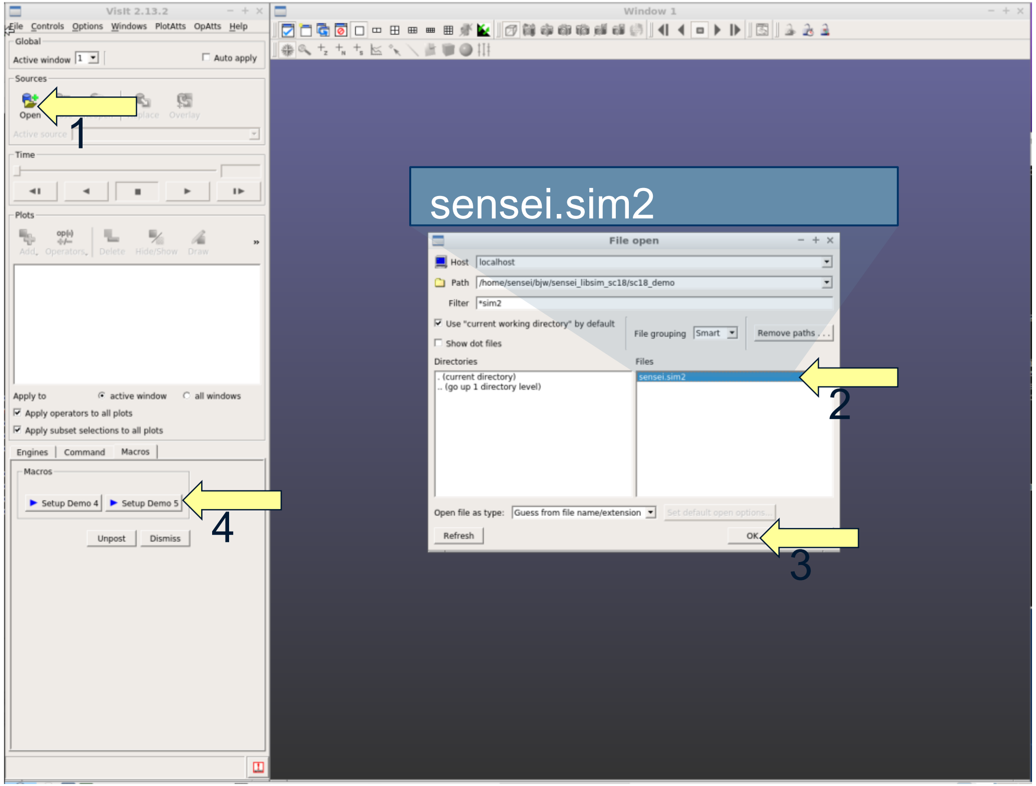


% ./demo 5 USERNAME ← Run oscillator, wait for VisIt

% ./demo 4 USERNAME ← Cleanup (at the end)

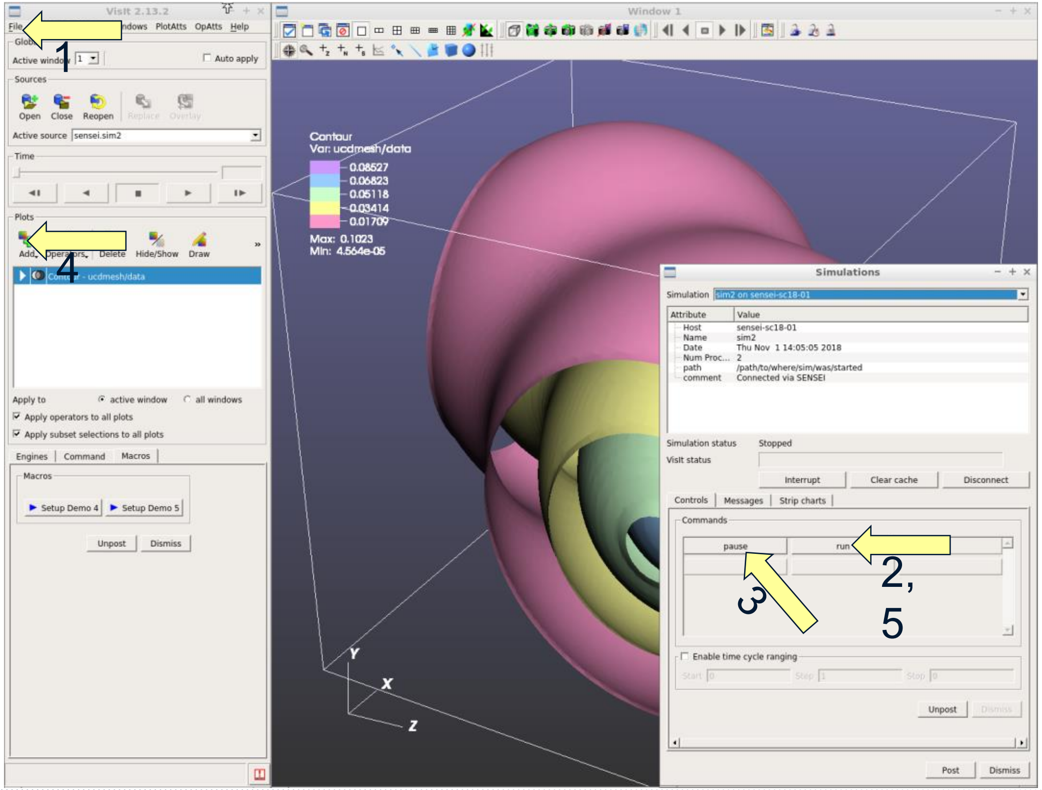
Libsim Demo: Connect Interactively to Oscillator

- Step 5 will open the VisIt GUI
- Open the File Selection window
- Select the “sensei.sim2” file
- Click OK
- VisIt will to connect to the oscillator
- Click the “Setup Demo 5” button to make plots



Libsim Demo: Let Oscillator Continue

- Click the File menu
- Open the Simulation window
- Click the “run” button in the Simulation window to let oscillator continue
- Pause the simulation
- Add other plots
- Let the simulation continue and watch it evolve



Libsim information

- Information about instrumenting a simulation can be found at the following sources:
- Getting Data Into VisIt (https://wci.llnl.gov/codes/visit/2.0.0/GettingDataIntoVisIt2.0.0.pdf)
- VisIt Example Simulations (http://visit.ilight.com/trunk/src/tools/DataManualExamples/Simulations)
- VisIt Wiki (http://www.visitusers.org)
- VisIt Email List (visit-users@email.ornl.gov)

SENSEI + Python

SENSEI is a powerful tool to connect simulations to visualization and analysis tools for in situ use. Here we show how to leverage this from a Python based simulation.

SENSEI's Python bindings

- SENSEI based on VTK but we use SWIG (Simple Wrapper Interface Generator) to generate Python bindings.
- VTK's Python wrapper generator, doesn't wrap many methods due to types it doesn't understand. Too purpose specific and inflexible.
- SWIG has extensive C++ compatibility and can be taught to play nice with VTK's wrapper generator
- Interface (.i) files control what gets wrapped. We wrap everything in SENSEI.
- Bound classes and API in Python have same names as in C++. Code looks and feels very C++ like.

For developers, extending or adding on to SENSEI

vtk.i : A SWIG interface file defining 2 macros:

1. `VTK_SWIG_INTEROP(vtk_t)`

- defines typemaps for using VTK wrapped VTK classes in SWIG generated API (tells SWIG how to play nice with VTK)

2. `VTK_DERIVED(derived_t)`

- enable SWIG memory management for wrapped classes derived from VTK classes (VTK has unique reference counting implementation)

✔ Pass a VTK class to SENSEI

✘ Pass a SENSEI class to VTK

Instrumenting Python Based Simulations

Integrating SENSEI in a simulation written in Python

1. Compile VTK with Python enabled. often a part of your chosen back-end. eg Catalyst, Libsim.
2. Compile SENSEI with Python features enabled
3. Write data adaptor using `sensei::ProgrammableDataAdaptor` or `sensei::VTKDataAdaptor`
4. Instrument your simulation, and bridge code. sets up the data adaptor and invoke analysis periodically through `sensei::ConfigurableAnalysis` adaptor.
5. Create any analysis specific run time configurations needed, eg. SENSEI XML files, Catalyst Python scripts, VisIt session files, etc..

Newton mini-app

N-body Gravitational Simulation. A single file, <400 lines.

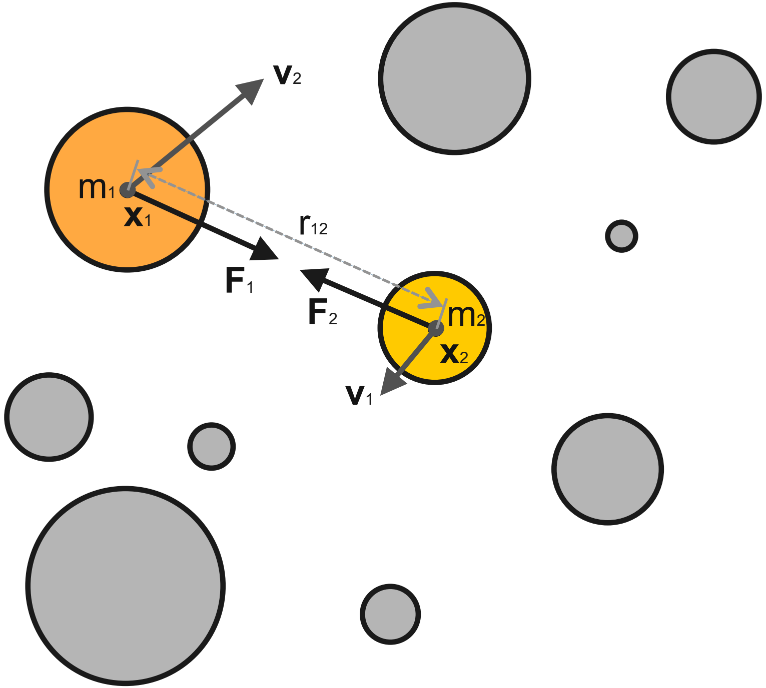
Solves Newton's law of gravitation

Velocity Verlet method

$F_i = F_j = G*m_i*m_j/r_{ij}**2$

$x_i' = v_i$

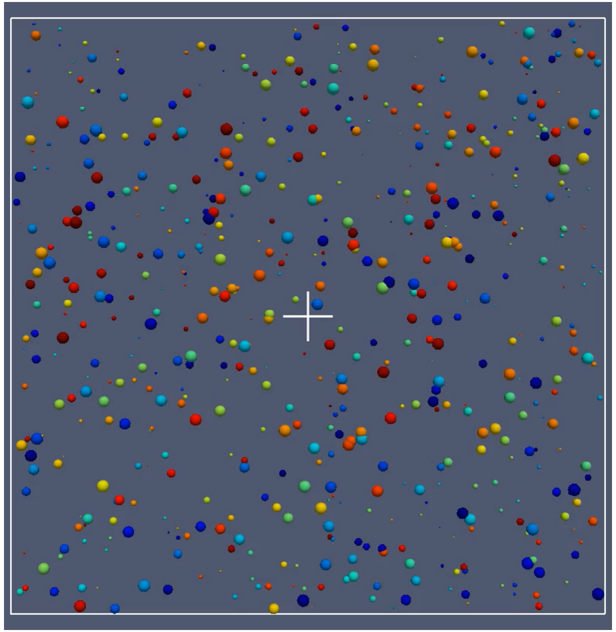
$v_i' = F_i/m_i$



BERKELEY LAB

Newton mini-app

- direct solver, O(N**2)
  - Velocity Verlet
    - » second order, symplectic, conserves momentum exactly, time reversible
- the simplest possible code
  - a single file, <400 lines, to better focus on use of SENSEI interface
  - a production quality code could easily be thousands of lines (see NBODY6 ~6K lines)



Instrumenting the simulation

```
if __name__ == '__main__':
    # parse the command line
    …

    # set up the initial condition
    n_bodies = args.n_bodies*n_ranks
    ic = uniform_random_ic(n_bodies, -5906.4e9, \
        5906.4e9, -5906.4e9, 5906.4e9, 10.0e24, \
        100.0e24, 1.0e3, 10.0e3)

    ids,x,y,z,m,vx,vy,vz,fx,fy,fz = ic.allocate()
    h = args.dt if args.dt else ic.get_time_step()

    # run the sim and analysis
    i = 1
    while i <= args.n_its:
        velocity_verlet(x,y,z,m,vx,vy,vz,fx,fy,fz,h)
        i += 1
```


BERKELEY LAB

Instrumenting the simulation

```python
# set up the initial condition
n_bodies = args.n_bodies*n_ranks
ic = uniform_random_ic(n_bodies, -5906.4e9, \
    5906.4e9, -5906.4e9, 5906.4e9, 10.0e24, \
    100.0e24, 1.0e3, 10.0e3)
ids,x,y,z,m,vx,vy,vz,fx,fy,fz = ic.allocate()
h = args.dt if args.dt else ic.get_time_step()

# create an analysis adaptor(bridge code)
adaptor = analysis_adaptor()
adaptor.initialize(args.analysis, args.analysis_opts)

# run the sim and analysis
adaptor.update(0,0,ids,x,y,z,m,vx,vy,vz,fx,fy,fz)
i = 1
while i <= args.n_its:
    velocity_verlet(x,y,z,m,vx,vy,vz,fx,fy,fz,h)
    adaptor.update(i,i*h,ids,x,y,z,m,vx,vy,vz,fx,fy,fz)
    i += 1

# finish up
adaptor.finalize()
```

Interface to SENSEI (aka the bridge)

```
class analysis_adaptor:
    def __init__(self):
        self.DataAdaptor = sensei.VTKDataAdaptor.New()
        self.AnalysisAdaptor = None

    def initialize(self, analysis, args=''):
        # select and configure SENSEI analysis adaptor
        …

    def finalize(self):
        if self.Analysis == 'posthoc':
            self.AnalysisAdaptor.Finalize()

    def update(self, i,t,ids,x,y,z,m,vx,vy,vz,fx,fy,fz):
        # convert simulation data to VTK
        # invoke the analysis
        …
```

– Our analysis adaptor bridge selects and configures and drives one of a number of SENSEI analysis adaptors

– Manages an instance of `sensei::VTKDataAdaptor` to which we will create and pass VTK objects to

BERKELEY LAB

Initializing the in situ analysis

```
def initialize(self, analysis, args=''):
    self.Analysis = analysis
    args = csv_str_to_dict(args)
    # Libsim
    if analysis == 'libsim':
        self.AnalysisAdaptor = sensei.LibsimAnalysisAdaptor.New()
        self.AnalysisAdaptor.AddPlots('Pseudocolor','ids', False,False, \
            (0.,0.,0.),(1.,1.,1.),sensei.LibsimImageProperties())
    # Catalyst
    elif analysis == 'catalyst':
        if check_arg(args,'script'):
            self.AnalysisAdaptor = sensei.CatalystAnalysisAdaptor.New()
            self.AnalysisAdaptor.AddPythonScriptPipeline(args['script'])
    # VTK I/O
    elif analysis == 'posthoc':
        if check_arg(args,'file','newton') and check_arg(args,'dir','./') \
            and check_arg(args,'mode','0') and check_arg(args,'freq','1'):
            self.AnalysisAdaptor = sensei.VTKPosthocIO.New()
            self.AnalysisAdaptor.Initialize(comm, args['dir'],args['file'], \
                [],['ids','fx','fy','fz','f','vx','vy','vz','v','m'], \
                int(args['mode']),int(args['freq']))
    # Configurable
    elif analysis == 'configurable':
        if check_arg(args,'config'):
            self.AnalysisAdaptor = sensei.ConfigurableAnalysis.New()
            self.AnalysisAdaptor.Initialize(comm, args['config'])

    if self.AnalysisAdaptor is None:
        status('ERROR: Failed to initialize "%s"\n'%(analysis))
        sys.exit(-1)
```

Select and configure one of the existing SENSEI analysis adaptors from command line arguments

- We are using Libsim, Catalyst, and VTKPosthocIO SENSEI analysis classes directly through the bindings
- SENSEI's Configurable analysis class also exposes these and more and is configurable via an XML file. Eg ADIOS

BERKELEY LAB

Invoking in situ back analysis

```
def update(self, i,t,ids,x,y,z,m,vx,vy,vz,fx,fy,fz):

    status('% 5d\n'%(i)) if i > 0 and i % 70 == 0 else None
    status('.')

    # construct VTK a dataset
    node = points_to_polydata(ids,x,y,z,m,vx,vy,vz,fx,fy,fz)
    mb = vtk.vtkMultiBlockDataSet()
    mb.SetNumberOfBlocks(n_ranks)
    mb.SetBlock(rank, node)

    # pass it to the data adaptor
    self.DataAdaptor.SetDataTime(t)
    self.DataAdaptor.SetDataTimeStep(i)
    self.DataAdaptor.SetDataObject(mb)

    # execute the in situ analysis
    self.AnalysisAdaptor.Execute(self.DataAdaptor)

    # free up memory
    self.DataAdaptor.ReleaseData()
```

1. create and pass Multi-block (tree based) dataset to SENSEI data adaptor
   – each rank is responsible for a leaf in the tree
2. pass time and step number to data adaptor
3. invoke the SENSEI analysis adaptor
4. release memory held in the adaptor

Create the VTK dataset

```
def points_to_polydata(ids,x,y,z,m,vx,vy,vz,fx,fy,fz):
    nx = len(x)
    # convert simulation to VTK data structures
    v_pts = to_vtk_points(nx,x,y,z)
    v_cells = to_vtk_cells(nx)
    v_ids = to_vtk_scalars(nx,'ids',ids)
    v_m = to_vtk_scalars(nx,'m',m)
    v_v,v_mv = to_vtk_vector(nx,'v',vx,vy,vz)
    v_f,v_mf = to_vtk_vector(nx,'f',fx,fy,fz)
    # package it all up in a poly data set
    pd = vtk.vtkPolyData()
    pd.SetPoints(pts)
    pd.GetPointData().AddArray(v_ids)
    pd.GetPointData().AddArray(v_m)
    pd.GetPointData().AddArray(v_v)
    pd.GetPointData().AddArray(v_mv)
    pd.GetPointData().AddArray(v_f)
    pd.GetPointData().AddArray(v_mf)
    pd.SetVerts(cells)
    return pd
```

Strategy

1. create VTK arrays
2. pass them to a VTK dataset

Who owns what?

- VTK uses reference counting. Python does too. Unfortunately they don't talk to each other without some extra code.
- Tell VTK to make a deep copy if the array goes out of scope

Dataset geometry

```
def to_vtk_points(nx,x,y,z):
    xyz = np.empty(3*nx, dtype=np.float32)
    xyz[::3] = x[:]
    xyz[1::3] = y[:]
    xyz[2::3] = z[:]
    vxyz = vtknp.numpy_to_vtk(xyz, deep=1)
    vxyz.SetNumberOfComponents(3)
    vxyz.SetNumberOfTuples(nx)
    pts = vtk.vtkPoints()
    pts.SetData(vxyz)
    return pts

def to_vtk_cells(nx):
    cids = np.empty(2*nx, dtype=np.int32)
    cids[::2] = 1
    cids[1::2] = np.arange(0,nx,dtype=np.int32)
    cells = vtk.vtkCellArray()
    cells.SetCells(nx, vtknp.numpy_to_vtk(cids, \
        deep=1, array_type=vtk.VTK_ID_TYPE))
    return cells
```

Strategy

1. create an empty array
2. interleave x,y,z components or cell length and point ids
3. pass new array to VTK data structure

TODO – test new zero copy stuff from DG

Array based data

```
def to_vtk_scalars(nx,name,s):
    scalar = vtknp.numpy_to_vtk(s, deep=1)
    scalar.SetName(name)
    return scalar

def to_vtk_vector(nx,name,vx,vy,vz):
    # vector in interleaved layout
    vxyz = np.zeros(3*nx, dtype=np.float32)
    vxyz[::3] = vx
    vxyz[1::3] = vy
    vxyz[2::3] = vz
    vector = vtknp.numpy_to_vtk(vxyz, deep=1)
    vector.SetName('v')
    # magnitude
    mv = np.sqrt(vx**2 + vy**2 + vz**2)
    mag = vtknp.numpy_to_vtk(mv, deep=1)
    mag.SetName('mag%s'%(name))
    return vector,mag
```

Scalars

1. pass new array to VTK data structure

Vectors/Tensors

1. create an empty array
2. interleave components
3. pass new array to VTK data structure

TODO – test new zero copy stuff from DG

Writing a DataAdaptor in Python

Strategy

- Add a class that contains functions returning callbacks that implement the SENSEI data adaptor API
  - Closures enable class state to be accessed from the callbacks
- This class contains an instance of `sensei::ProgramableDataAdaptor` which has been initialized with your callbacks
- Set up call forwarding. when a non-existent member function is called, the call is forwarded to the `sensei::ProgramableDataAdaptor` instance

The Programable Data Adaptor

```
class ProgrammableDataAdaptor : public DataAdaptor
{
public:
  using GetNumberOfMeshesFunction = std::function<int(unsigned int&)>;

  /// Set the callable that will be invoked when GetNumberOfMeshes is called
  void SetGetNumberOfMeshesCallback(const GetNumberOfMeshesFunction &callback);

  /// @breif Gets the number of meshes a simulation can provide
  int GetNumberOfMeshes(unsigned int &numMeshes) override;


  using GetMeshNameFunction =
    std::function<int(unsigned int, std::string &)>;

  /// Set the callable that will be invoked when GetMeshName is called
  void SetGetMeshNameCallback(const GetMeshNameFunction &callback);

  /// @breif Get the name of the i'th mesh
  int GetMeshName(unsigned int id, std::string &meshName) override;

  .
  .
  .
  continues for all overrides in the data adaptor API

};
```

C++ class implementing SENSEI's DataAdaptor API that forwards incoming SENSEI API calls to user provided "callables"

SENSEI's Python bindings handle forwarding to user provided Python "callables"

Writing a Python DataAdaptor

```
class data_adaptor:
    def __init__(self):                       # set up data structures to capture sim data, and plumbing to ProgramableDataAdaptor instance
    …
    def __getattr__(self, *args):             # forward calls to ProgramableDataAdaptor instance
    …
    def base(self):                           # return PDA instance
    …
    def validate_mesh_name(self, mesh_name): # helper checks mesh name
    …
    def update(self, i,t,ids,x,y,z,
      m,vx,vy,vz,fx,fy,fz):                   # capture latest simulation data
    …
    def set_array_1(self, vals, name):        # Convert sim array into VTK scalar
    …
    def set_array_3(self, vx,vy,vz, name):    # Convert sim arrays into VTK vector
    …
    def set_geometry(self, x,y,z):            # Convert sim arrays into VTK Polydata
    …
    def get_number_of_meshes(self):           # get SENSEI API callback
    …
    def get_mesh_name(self):                  # get SENSEI API callback
    …
    def get_number_of_arrays(self):           # get SENSEI API callback
    …
    def get_array_name(self):                 # get SENSEI API callback
    …
    def get_mesh(self):                       # get SENSEI API callback
    …
    def add_array(self):                      # get SENSEI API callback
    …
    def release_data(self):                   # get SENSEI API callback
```

the purpose of this class:

1. provides callbacks implementing SENSEI data adaptor API
2. gives callbacks access to simulation state
3. installs the callbacks in the ProgrammableDataAdaptor

BERKELEY LAB

Writing a Python Data

```
def __init__(self):                     #
    # capture data from sim
    self.arrays = {}
    self.points = None
    self.cells = None
    # PDA plumbing. connect all the callbacks
    self.pda = sensei.ProgrammableDataAdaptor.New()
    self.pda.SetGetNumberOfMeshesCallback(self.get_number_of_meshes())
    self.pda.SetGetMeshNameCallback(self.get_mesh_name())
    self.pda.SetGetNumberOfArraysCallback(self.get_number_of_arrays())
    self.pda.SetGetArrayNameCallback(self.get_array_name())
    self.pda.SetGetMeshCallback(self.get_mesh())
    self.pda.SetAddArrayCallback(self.add_array())
    self.pda.SetReleaseDataCallback(self.release_data())

def __getattr__(self, *args):           # forward ca
    return self.pda.__getattribute__(*args)

def base(self):                         # return PDA ins
    return self.pda

def validate_mesh_name(
    if mesh_name != "bodies"
        raise RuntimeError('no
```

simulation state that can be accessed from callbacks

Install callbacks that implement SENSEI DataAdaptor API

forward calls to the PDA instance

get the PDA, it will be passed into the analysis

Python DataAdaptor

```
def get_number_of_meshes(self):          # get SENSEI API callback
    def callback():
        return 1
    return callback

def get_mesh_name(self):                  # get SENSEI API callback
    def callback(idx):
        if idx != 0: raise RuntimeError('no mesh %d'%(idx))
        return 'bodies'
    return callback

def get_number_of_arrays(self):           # get SENSEI API callback
    def callback(mesh_name, assoc):
        self.validate_mesh_name(mesh_name)
        return len(self.arrays.keys()) \
            if assoc == vtk.vtkDataObject.POINT else 0
    return callback

def get_array_name(self):                 # get SENSEI API callback
    def callback(mesh_name, assoc, idx):
        self.validate_mesh_name(mesh_name)
        return self.arrays.keys()[idx] \
            if assoc == vtk.vtkDataObject.POINT else 0
    return callback
```


BERKELEY LAB

Python DataAdaptor

```python
def get_mesh(self):                         # get SENSEI API callback
        def callback(mesh_name, structure_only):
            self.validate_mesh_name(mesh_name)
            # local bodies
            pd = vtk.vtkPolyData()
            if not structure_only:
                pd.SetPoints(self.points)
                pd.SetVerts(self.cells)
            # global dataset
            mb = vtk.vtkMultiBlockDataSet()
            mb.SetNumberOfBlocks(n_ranks)
            mb.SetBlock(rank, pd)
            return mb
        return callback

    def add_array(self):                        # get SENSEI API callback
        def callback(mesh, mesh_name, assoc, array_name):
            self.validate_mesh_name(mesh_name)
            if assoc != vtk.vtkDataObject.POINT:
                raise RuntimeError('no array named "%s" in cell data'%(ar
            pd = mesh.GetBlock(rank)
            pd.GetPointData().AddArray(self.arrays[array_name])
        return callback

    def release_data(self):                     # get SENSEI API callback
        def callback():
            self.arrays = {}
            self.points = None
            self.cells = None
        return callback
```

The closure pattern: a function that returns a function. The returned function can see/access data that is in the scope of the outer/returning function. here it gives us access to a reference to “self”, and simulation state stored therein.

BERKELEY LAB

Python DataAdaptor

```
def update(self, i,t,ids,x,y,z,m,vx,vy,vz,fx,fy,fz):
    # update the state arrays
    self.set_array_1(ids, 'ids')
    self.set_array_1(m, 'm')
    self.set_array_3(vx,vy,vz, 'v')
    self.set_array_3(fx,fy,fz, 'f')
    self.set_geometry(x,y,z)
    self.SetDataTime(t)      # fwd to PDA
    self.SetDataTimeStep(i)  # fwd to PDA
```

Python DataAdaptor

```
def set_array_1(self, vals, name):
    arr = vtknp.numpy_to_vtk(vals, 1)
    arr.SetName(name)
    self.arrays[name] = arr

def set_array_3(self, vx,vy,vz, name):
    # vector
    nx = len(x)
    vxyz = np.zeros(3*nx, dtype=vx.dtype)
    vxyz[::3] = vx
    vxyz[1::3] = vy
    vxyz[2::3] = vz
    vtkv = vtknp.numpy_to_vtk(vxyz, deep=1)
    vtkv.SetName(name)
    self.arrays[name] = vtkv
    # mag
    mname = 'mag%s'%(name)
    mv = np.sqrt(vx**2 + vy**2 + vz**2)
    vtkmv = vtknp.numpy_to_vtk(mv, deep=1)
    vtkmv.SetName(mname)
    self.arrays[mname] = vtkmv
```

Python DataAdaptor

```python
def set_geometry(self, x,y,z):
    # points
    nx = len(x)
    xyz = np.zeros(3*nx, dtype=x.dtype)
    xyz[::3] = x[:]
    xyz[1::3] = y[:]
    xyz[2::3] = z[:]
    vxyz = vtknp.numpy_to_vtk(xyz, deep=1)
    vxyz.SetNumberOfComponents(3)
    vxyz.SetNumberOfTuples(nx)
    pts = vtk.vtkPoints()
    pts.SetData(vxyz)
    self.points = pts
    # cells
    cids = np.empty(2*nx, dtype=np.int32)
    cids[::2] = 1
    cids[1::2] = np.arange(0,nx,dtype=np.int32)
    cells = vtk.vtkCellArray()
    cells.SetCells(nx, vtknp.numpy_to_vtk(cids, \
        deep=1, array_type=vtk.VTK_ID_TYPE))
    self.cells = cells
```


BERKELEY LAB

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

Python Analysis Backend

Python Analysis Backend

- Enable in situ analysis using all the power and simplicity of Python
- Rapid prototyping and design of diagnostics and numerical analysis
- Entirely independent of any other backend
- Can be coupled to simulations which have no knowledge of Python. for instance to a simulation written in Fortran

SENSEI Python Analysis Adaptor

```
namespace sensei {

class PythonAnalysis : public AnalysisAdaptor
{
public:
   void SetScriptFi  (const std::  ring &fi   );
   void SetInitializeS   ce(const std:   ing &

   int Initialize();

   int Execute(sensei::DataAdaptor        override;
   int Finalize() override;
};

}
```

sets a string containing Python code that is executed before the user provided Initialize function is called

Creates and initializes an embedded interpreter, loads the user script, runs the initialization source, invokes the user provided initialization

sets the path to the user provided Python script

A C++ Class t

lls to a
ython
tion

calls the user provided function, shuts down the embedded interpreter

User Provided Script Template

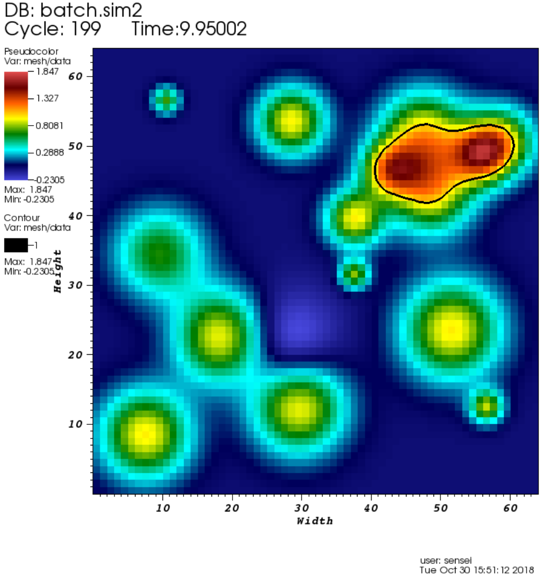
```
def Initialize():
    # your initialization code here
    return

def Execute(dataAdaptor):
    # your in situ analysis code here
    return

def Finalize():
    # your tear down code here
    return
```

Parallel Python code

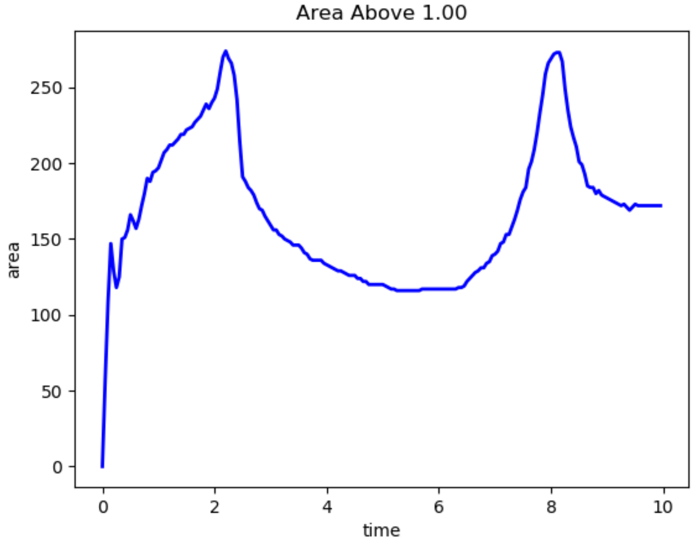
- SENSEI supports ghost zones using the masking conventions defined by VisIt (also used by VTK/ParaView) now. The mask array is named vtkGhostType
- SENSEI's MPI communicator , which may not be MPI_COMM_WORLD, is shared with the Python script via a global variable `comm`

Case Study: Chemical Reaction on 2D Substrate

Input Data: Proxy simulation of chemical reaction on a 2D substrate



Output of analysis: Area where reaction rate exceeds a threshold of 1.0



“Area above threshold” Source Code

```python
import numpy as np, matplotlib.pyplot as plt
from vtk.util.numpy_support import * from vtk import vtkDataObject, vtkCompositeDataSet

# default values of control parameters
threshold = 0.5
mesh = ''
array = ''
cen = vtkDataObject.POINT
out_file = 'area_above.png'
times = []
area_above = []

def pt_centered(c):
    return c == vtkDataObject.POINT

def Execute(adaptor):
    # get the mesh and arrays we need
    dobj = adaptor.GetMesh(mesh, False)
    adaptor.AddArray(dobj, mesh, cen, array)
    adaptor.AddGhostCellsArray(dobj, mesh)
    time = adaptor.GetDataTime()
    # compute area above over local blocks
    vol = 0.
    it = dobj.NewIterator()
    while not it.IsDoneWithTraversal():
        # get the local data block and its props
        blk = it.GetCurrentDataObject()
        # get the array container
        atts = blk.GetPointData() if pt_centered(cen) \
            else blk.GetCellData()
        # get the data and ghost arrays
        data = vtk_to_numpy(atts.GetArray(array))
        ghost = vtk_to_numpy(atts.GetArray('vtkGhostType'))
        # compute the area above
        ii = np.where((data > threshold) & (ghost == 0))
        vol += len(ii[0])*np.prod(blk.GetSpacing())
        it.GoToNextItem()
    # compute global area
    vol = comm.reduce(vol, root=0, op=MPI.SUM)
    # rank zero writes the result
    if comm.Get_rank() == 0:
        times.append(time)
        area_above.append(vol)

def Finalize():
    if comm.Get_rank() == 0:
        plt.plot(times, area_above, 'b-', linewidth=2)
        plt.xlabel('time')
        plt.ylabel('area')
        plt.title('area Above %0.2f'%(threshold))
        plt.savefig(out_file)
    return 0
```


```python
def Execute(adaptor):
    # get the mesh and arrays we need
    dobj = adaptor.GetMesh(mesh, False)
    adaptor.AddArray(dobj, mesh, cen, array)
    adaptor.AddGhostCellsArray(dobj, mesh)
    time = adaptor.GetDataTime()
    # compute area above over local blocks
    vol = 0.
    it = dobj.NewIterator()
        # compute the area above
        ii = np.where((data > threshold) & (ghost == 0))
        vol += len(ii[0])*np.prod(blk.GetSpacing())
        it.GoToNextItem()
    # compute global area
    vol = comm.reduce(vol, root=0, op=MPI.SUM)
    # rank zero writes the result
    if comm.Get_rank() == 0:
        times.append(time)
        area_above.append(vol)

def Finalize():
    if comm.Get_rank() == 0:
        plt.plot(times, area_above, 'b-', linewidth=2)
        plt.xlabel('time')
        plt.ylabel('area')
        plt.title('area Above %0.2f'%(threshold))
        plt.savefig(out_file)
    return 0
```

Configurable Analysis XML

```
<sensei>
  <analysis type="python" script_file="area_above.py" enabled="1">
    <initialize_source>
threshold=1.
mesh='mesh'
array='data'
cen=1
     </initialize_source>
  </analysis>
</sensei>
```

Python code that executes before user's Initialize function

path to the user provided Python script

Running the demo

This demo shows Python based analysis from a code written in C++. The surface area where the data exceeds a runtime specified threshold over a 2D domain is calculated at each update. At the end of the run, an image showing the calculation over time is produced.

VM

```
cd ~/sensei_insitu/demos/sc18/python
./oscillator_python.sh
```

Cori

```
cd $SCRATCH
salloc -N 2 -C haswell -t 01:00:00 \
    -q regular --reservation=SC18_SENSEI
./sensei_insitu/demos/sc18/adios/oscillator_python.sh
```


BERKELEY LAB

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

ADIOS

BERKELEY LAB

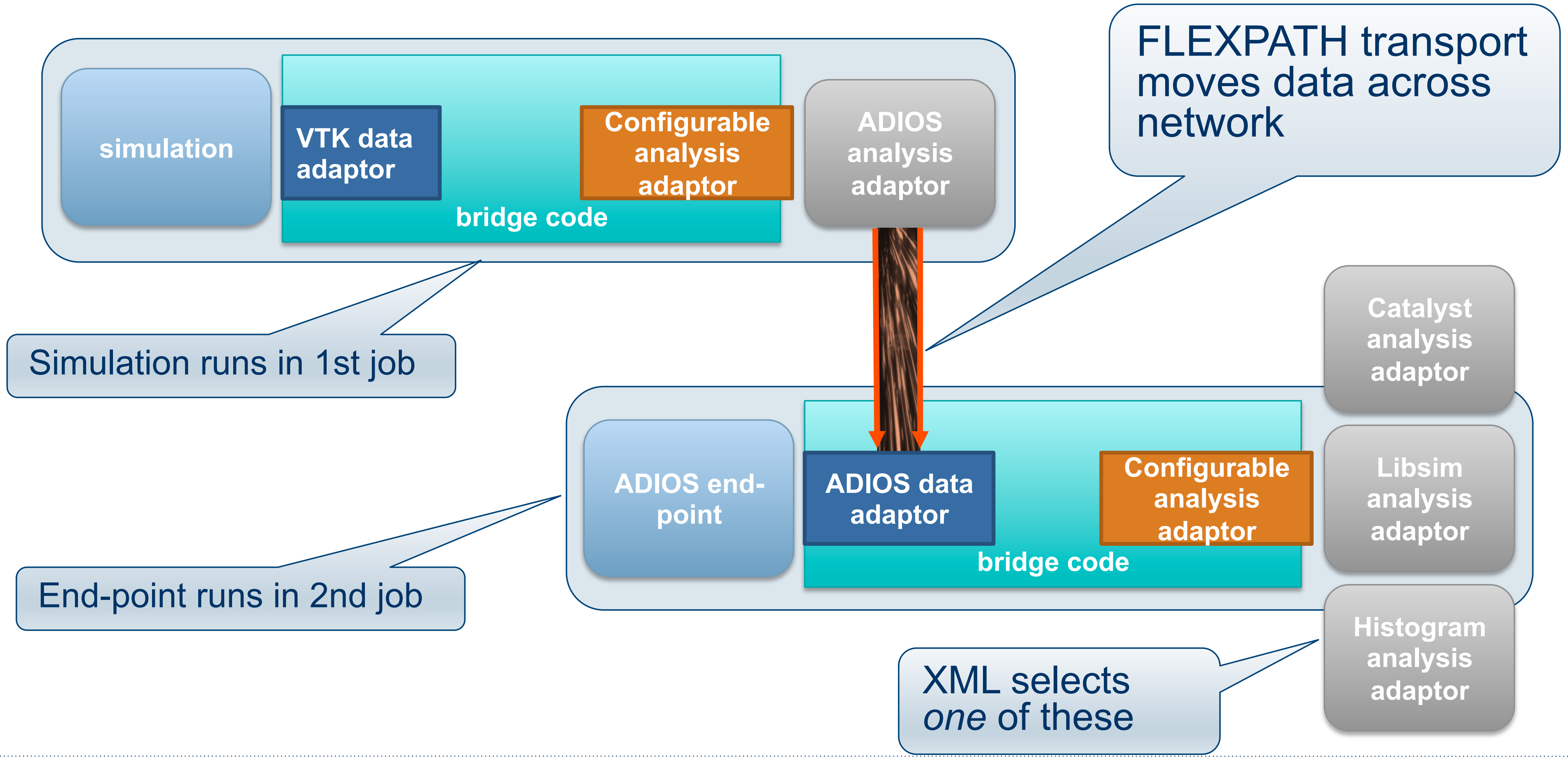
Argonne NATIONAL LABORATORY

OAK RIDGE National Laboratory

Intelligent Light

Kitware

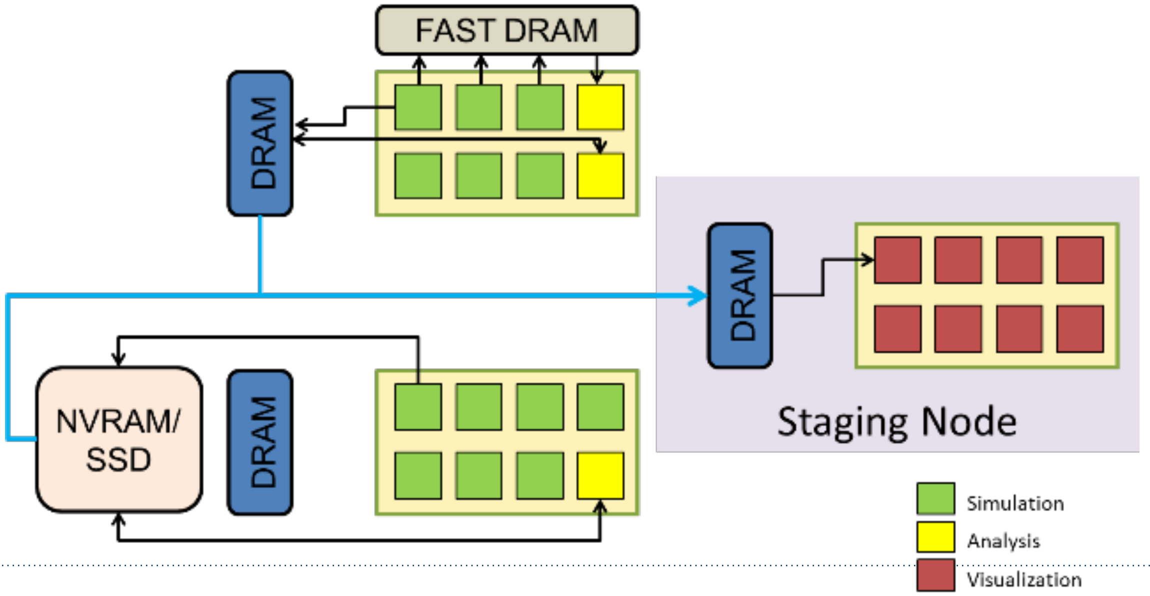
In transit Architecture

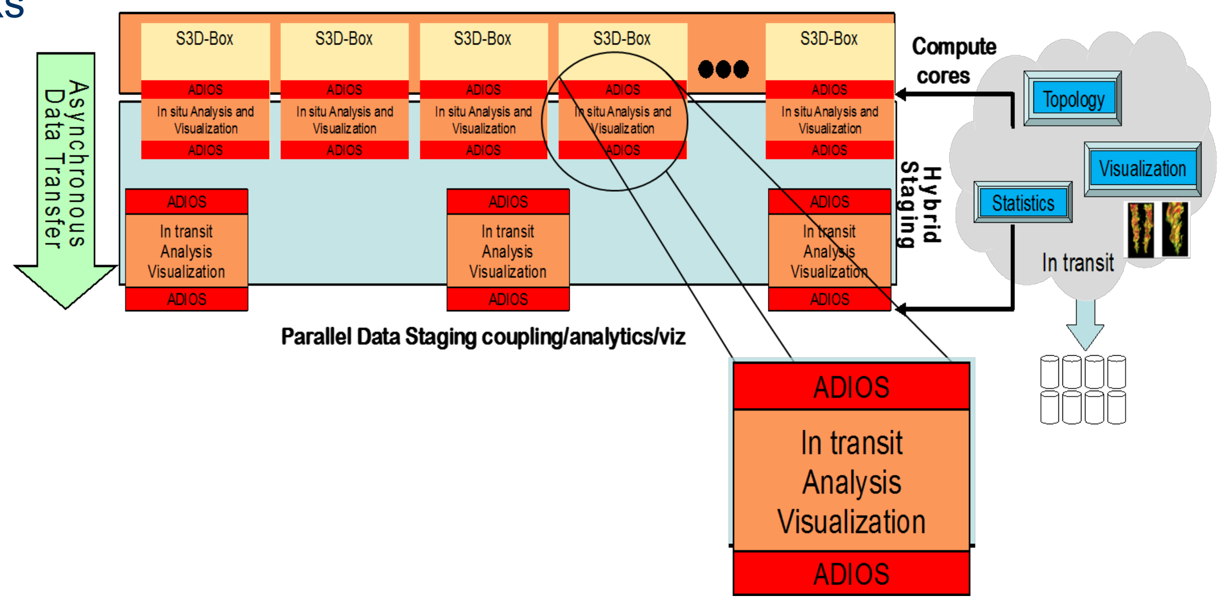


Data management tradeoffs at exascale → to hybrid staging

Explore node layout choices for data management

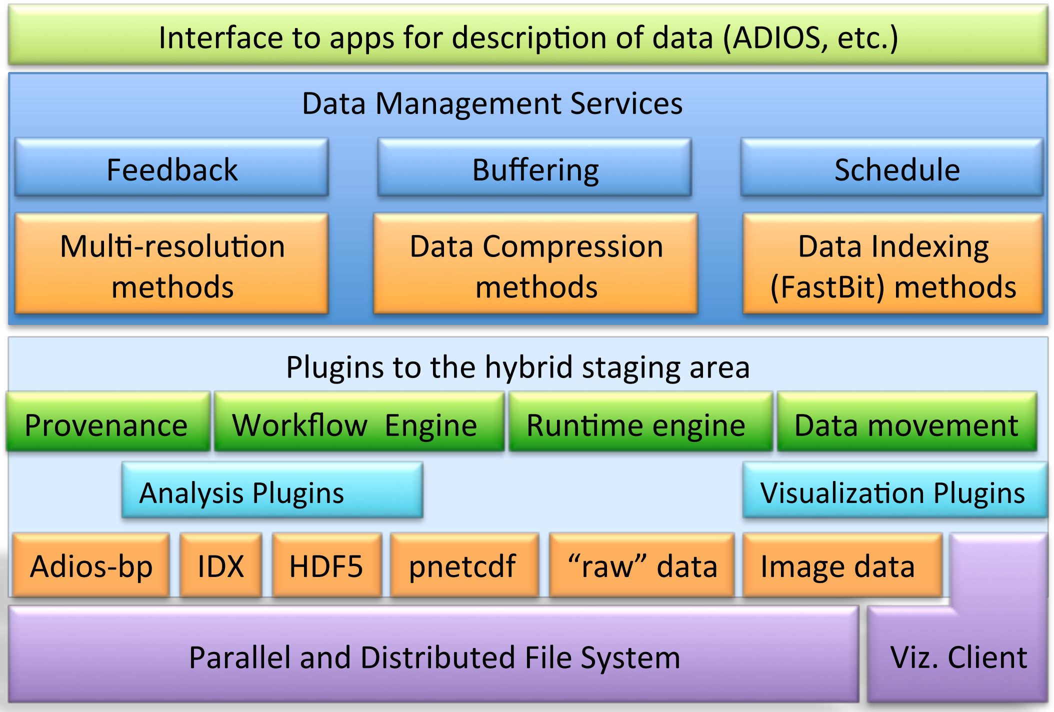
- Balance of memory size and speed
- Feedback for node designs with NVRAM, larger memory, on-chip NIC
- Network throughput and latency impact on SDMA tasks
- Placement of operations in concert with solver and network topology

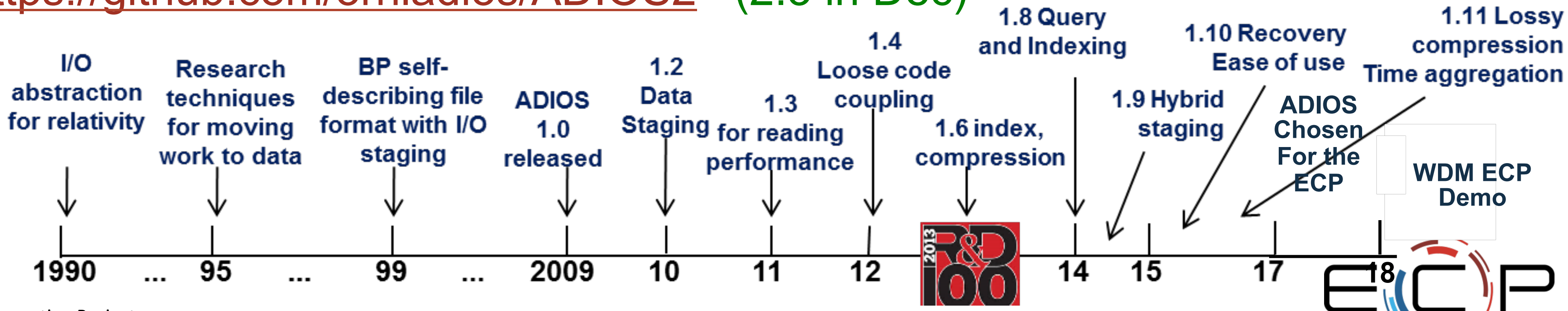




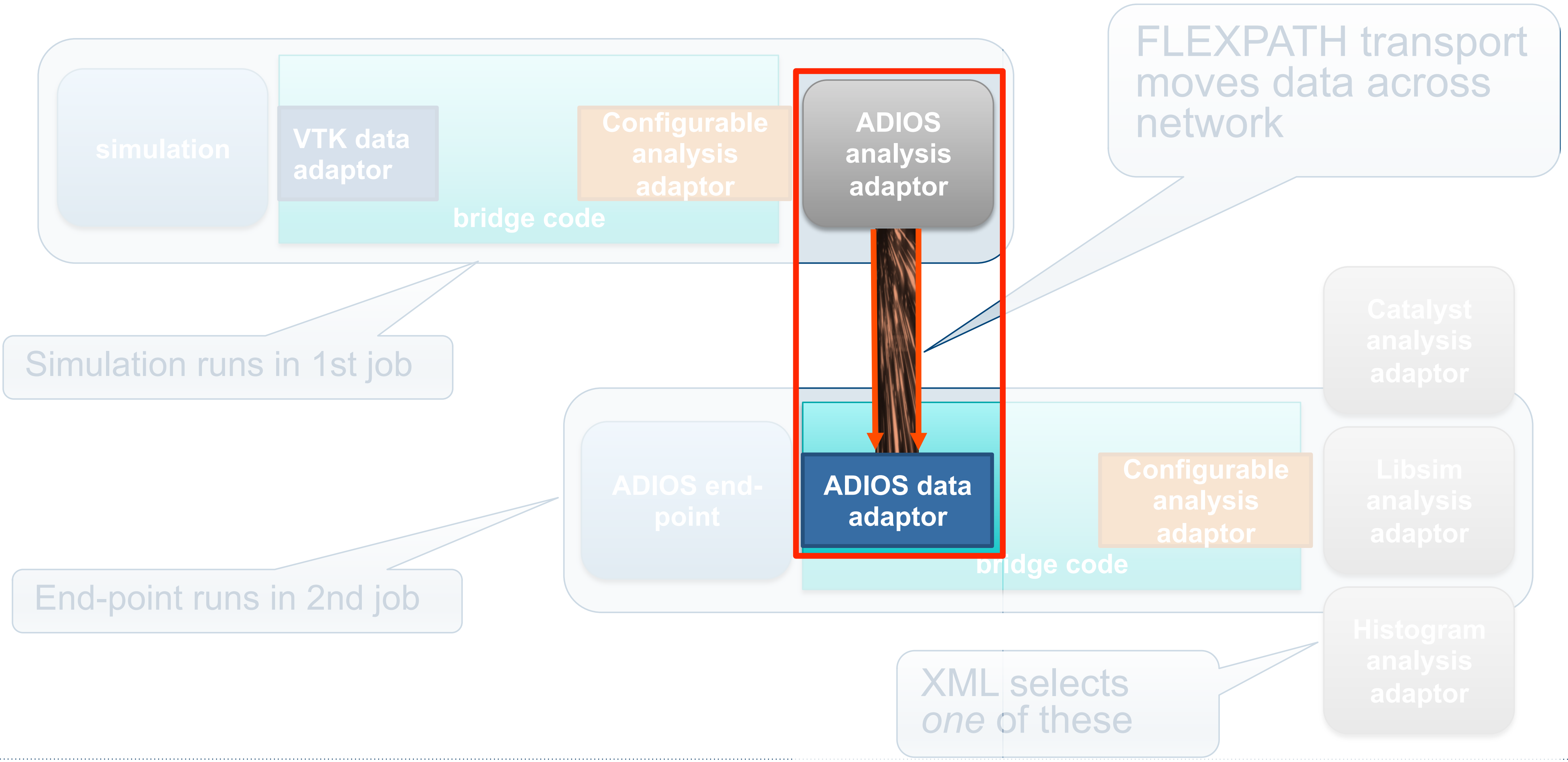
What is ADIOS

- An extendable framework that allows developers to *plug-in*
  - **I/O methods**: Aggregate, Posix, MPI
  - **Services**: Compression, Decompression
  - **Formats**: HDF5, netcdf, ADIOS-BP,…
  - **Plug-ins**: Analytic, Visualization
- Incorporates the "best" practices in the I/O middleware layer
- Bindings to F90, C++, C, Python, R, Java, Matlab
- https://csmd.ornl.gov/adios, https://github.com/ornladios/ADIOS, (1.13.1) https://github.com/ornladios/ADIOS2 (2.3 in Dec)

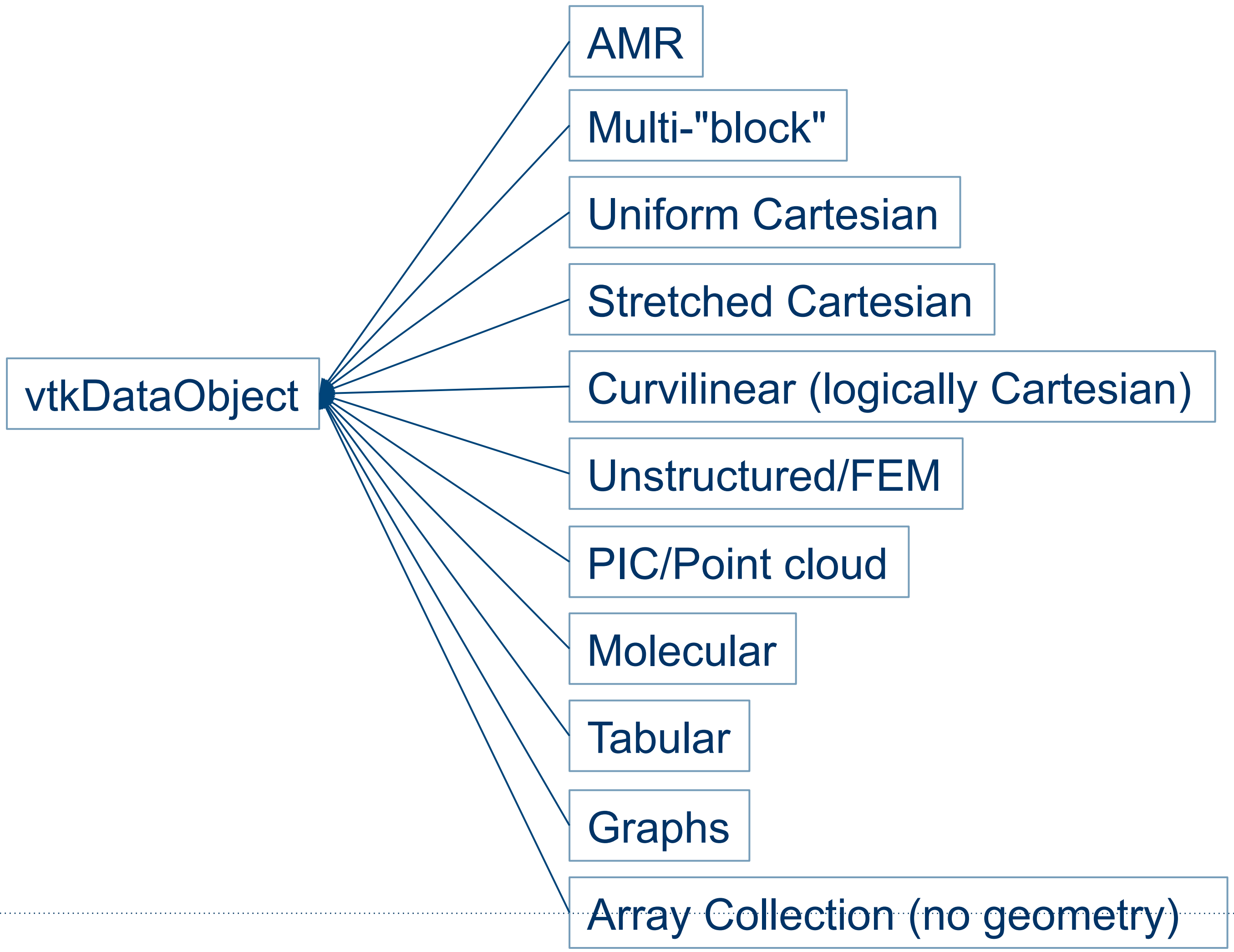




ADIOS Adaptors



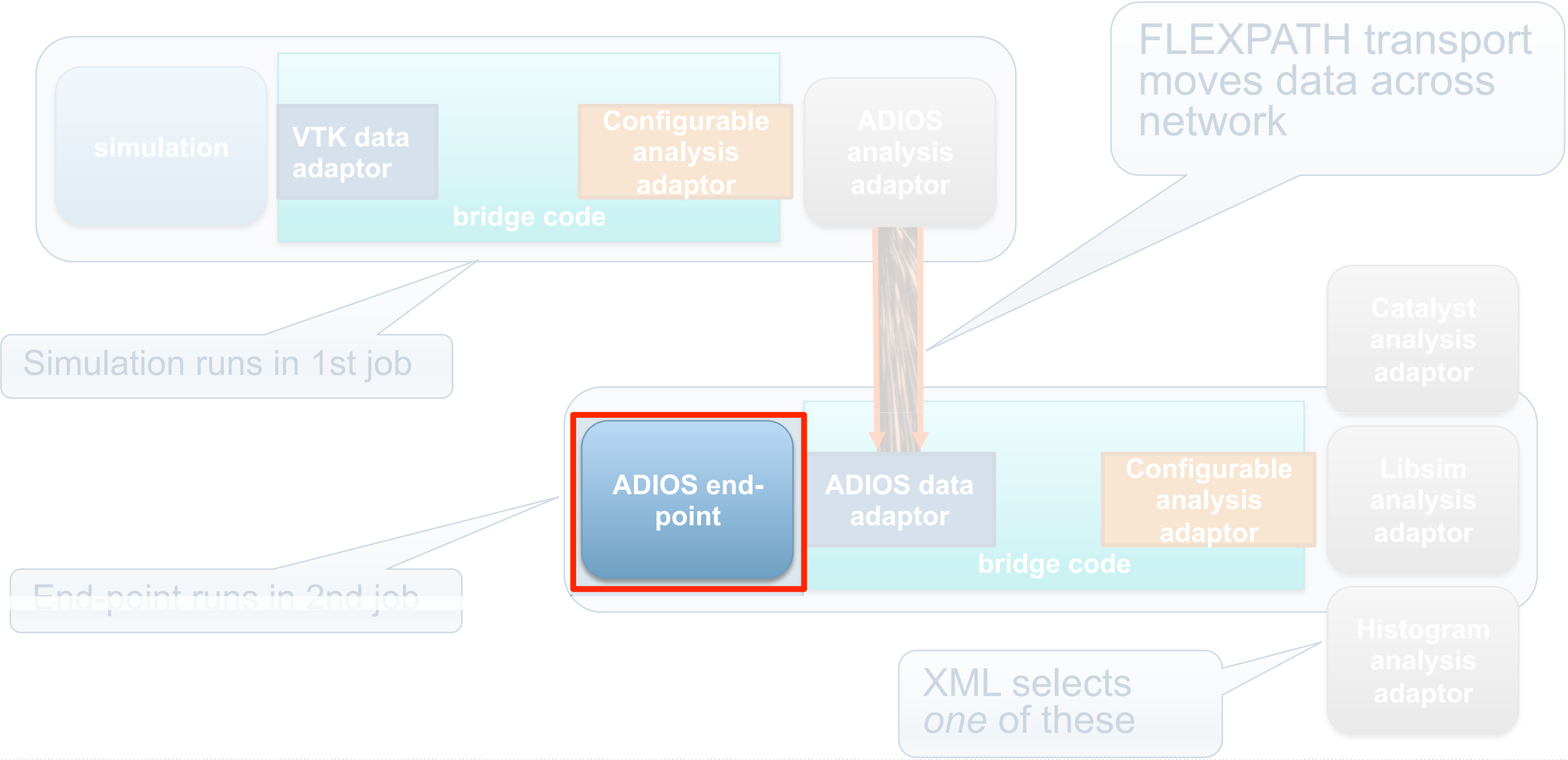
What simulation data types does SENSEI support?



- many more purpose specific and esoteric data types are supported by VTK
- **no explicit dependence on other parts of VTK such as i/o, filters, renderering, etc etc**

www.vtk.org

End-Point


simulation
VTK data adaptor
Configurable analysis adaptor
bridge code
ADIOS analysis adaptor
FLEXPATH transport moves data across network
Simulation runs in 1st job
Catalyst analysis adaptor
ADIOS end-point
ADIOS data adaptor
Configurable analysis adaptor
bridge code
Libsim analysis adaptor
End-point runs in 2nd job
Histogram analysis adaptor
XML selects
one of these

In transit demo

The demo runs 2 parallel MPI jobs, in the first the oscillator sends data through the ADIOS Analysis adaptor. In the second the end point uses the ADIOS data adaptor to receive.

SENSEI XML is displayed in cyan along with mpiexec/srun commands in white. The first job's output is displayed in red, the second job's output in green

srun "`-r X`" argument tells to start the job on node X

```
cd $SCRATCH
salloc -N 2 -C haswell -t 01:00:00 \
    -q regular --reservation=SC18_SENSEI
./sensei_insitu/demos/sc18/adios/in_transit_libsim.sh
```

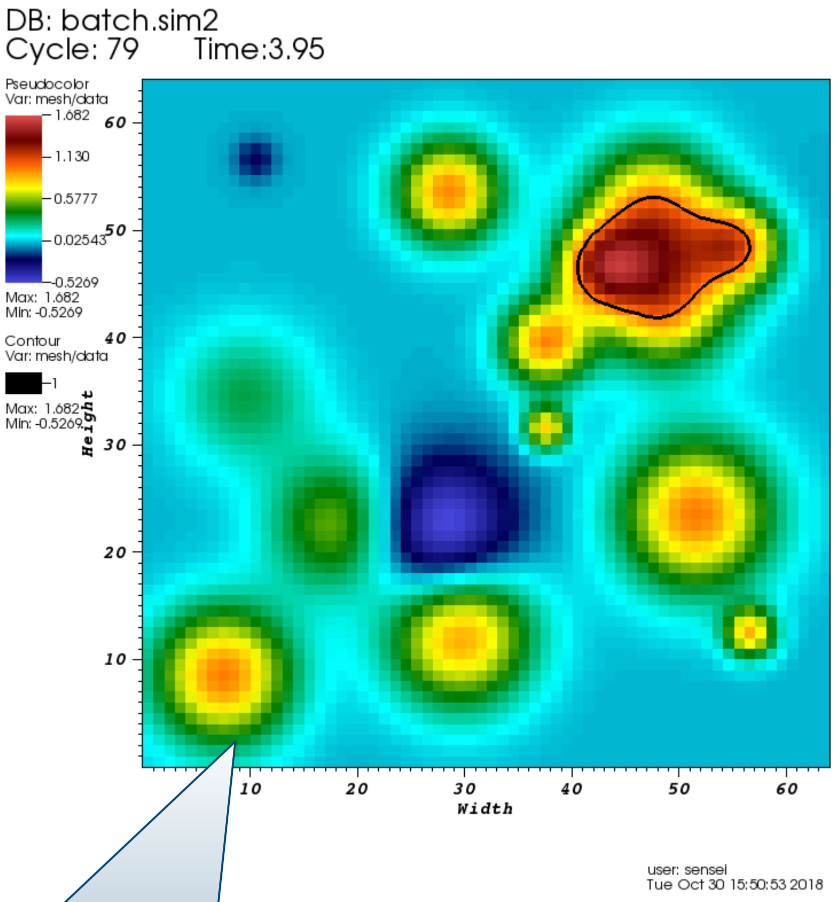
In transit demo (VM)

The demo runs 2 parallel MPI jobs, in the first the oscillator sends data through the ADIOS Analysis adaptor. In the second the end point uses the ADIOS data adaptor to receive.

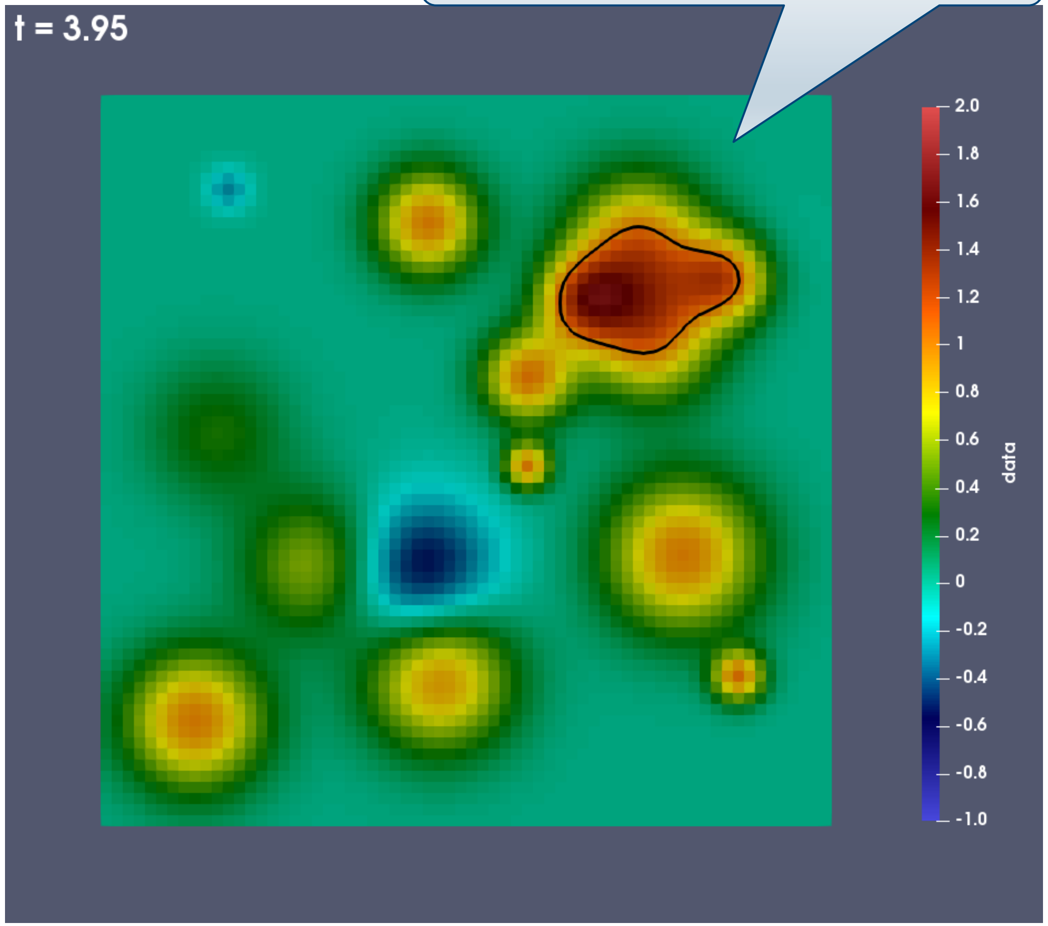
SENSEI XML is displayed in cyan along with mpiexec/srun commands in white. The first job's output is displayed in red, the second job's output in green

```
cd ~/sensei_insitu/demos/sc18/adios
./in_transit_libsim.sh
```

Demo output

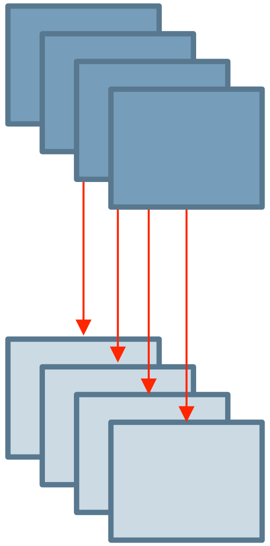


rendered with libsim

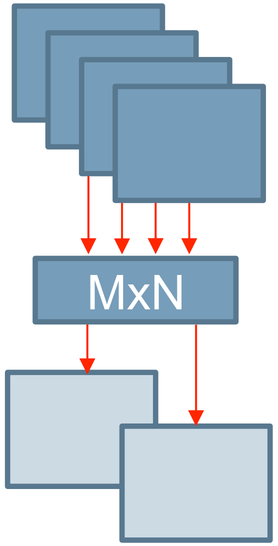


rendered with catalyst

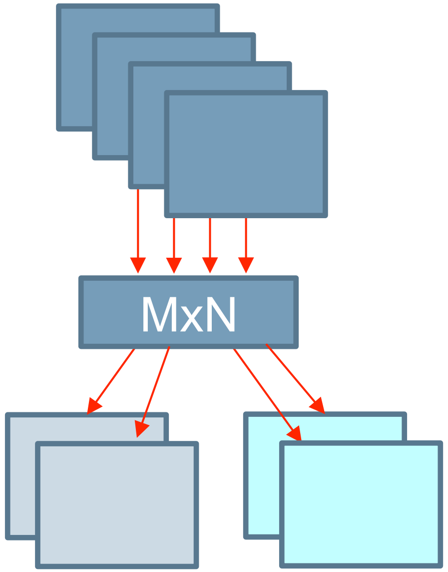
Design and execution patterns



N producer ranks,
N consumer ranks
Unidirectional
data movement/
control
(N:N)



M producer ranks,
N consumer ranks
Unidirectional
data movement/
control
(M:N)



M producer ranks,
N1 and N2
consumer ranks,
Unidirectional
data movement/
control
(M:<N1, N2>)

Research focus areas:

- MxN data redistribution
- Depth of copies
- Leveraging arch features like NVRAM for staging
- Leveraging 3rd party tools like TensorFlow for ML-based analytics
- Specific science app use case drivers

SENSEI In Situ Demonstrations

Computational Monitoring with ParaView/Catalyst

BERKELEY LAB · Argonne NATIONAL LABORATORY · OAK RIDGE National Laboratory · Intelligent Light · Kitware

SC18 Dallas, TX | hpc inspires.

Agenda

- Overview of ParaView/Catalyst Functionality
- Catalyst Editions
- Python Pipelines
- Live Connections for Computational Monitoring
- Demo / Exercise

ParaView & Catalyst

- Scaled to 10^6 MPI ranks on ALCF's Mira BG/Q
- SC16 visualization showcase winner generated animation using Catalyst
- HPCWire Best HPC Visualization Product or Technology
  - 2011 (VTK), 2012, 2014 (runner-up), 2016 Editor's Choice (ParaView)
  - 2015 Reader's Choice – tie (Paraview)
- Used on many HPC architectures: Cray, BlueGene, SGI, etc.

What Can Catalyst Do?

- Catalyst can save

  - A subset of your data (usu. only useful for small tests)
    - Scripts can determine when to start/stop saving data

  - A sequence of images
    - 1+ per timestep; multiple views are possible.

  - A Cinema database
    - A separate image per "actor", with per-pixel depth & scalar values.
    - Interactive post hoc re-coloring & composition of images via depth & scalar values.

What Can Catalyst Do?

- Two use cases:
  - Extremely low overhead with Catalyst Editions and a fixed visualization
    - Only compile portions of ParaView and VTK that you will use
    - Pipeline configured via C++
  - Extremely flexible visualizations with Catalyst Python scripts
    - ParaView can write a Python script you can customize
    - Change scripts on a per-job basis

Catalyst Editions

In depth: https://blog.kitware.com/paraview-catalyst-editions-what-are-they/
https://www.paraview.org/Wiki/Generating_Catalyst_Source_Tree

- Reduce the number of libraries built and linked to reduce startup time and memory overhead.
- Works with either static or dynamic library linkage.
- Especially important on large machines if dynamic linking is used as link loaders have much less work to do.
- Reduces both executable file size and resident memory usage, but reduces flexibility since some functionality will no longer be present.

Fixed Catalyst Pipelines

- SENSEI provides 2 example C++ pipelines:
  - A slice filter that saves an image of a slice through your data.
  - A particle renderer that uses ParaView's point-Gaussian renderer.
- These are examples if you decide the overhead of Python is too high.

Exercises 1 & 2

```
cd ~/sensei_insitu/demos/sc18/pv_catalyst          ← On the VM
cd /project/projectdirs/m636/sensei_insitu/        ← On Cori
    demos/sc18/pv_catalyst
./demo 0 username                                  ← On either
./demo 1 username
./demo 2 username
```

- Create a visualization of oscillator mini-app data using a fixed pipeline
- Configure the oscillator to use the Catalyst slice analysis
- Show output images

Python Pipelines

- Load a sample of your data in ParaView
  - May be a downsampled version, but
  - Should include all variables/attributes/fields you wish to analyze.

- Create a visualization pipeline in ParaView by filtering data
  - Successive filters generate subsetted or alternative forms of data without overwriting the original data, but they do consume memory.
  - Choose representation style and visual properties for data

- Export a Catalyst script with *Catalyst→Generate Script* (v5.5.2) or *Catalyst→Define Exports* and the Catalyst Export Inspector panel (v5.6.0).

Pipelines for ParaView 5.5.2

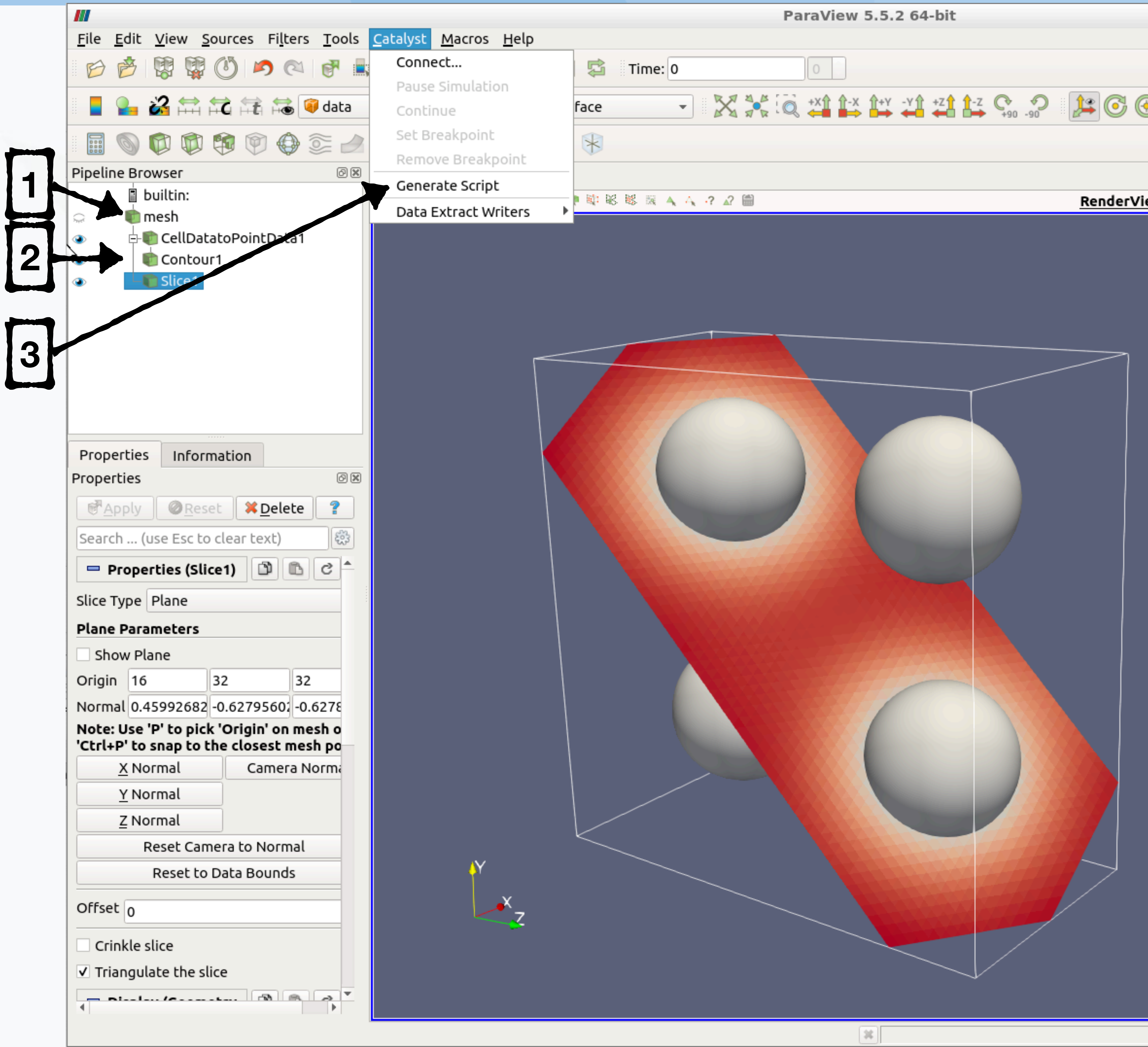
The following slides show how to create Python pipeline scripts using ParaView version 5.5.2, which is the version in the tutorial VM.

Creating a Python Pipeline

1 • Attach to Catalyst/Live or load an example dataset.

2 • Create a pipeline. Here we have averaged cell data to points, contoured, and sliced an example dataset.

3 • Then click "Catalyst→Generate Script".

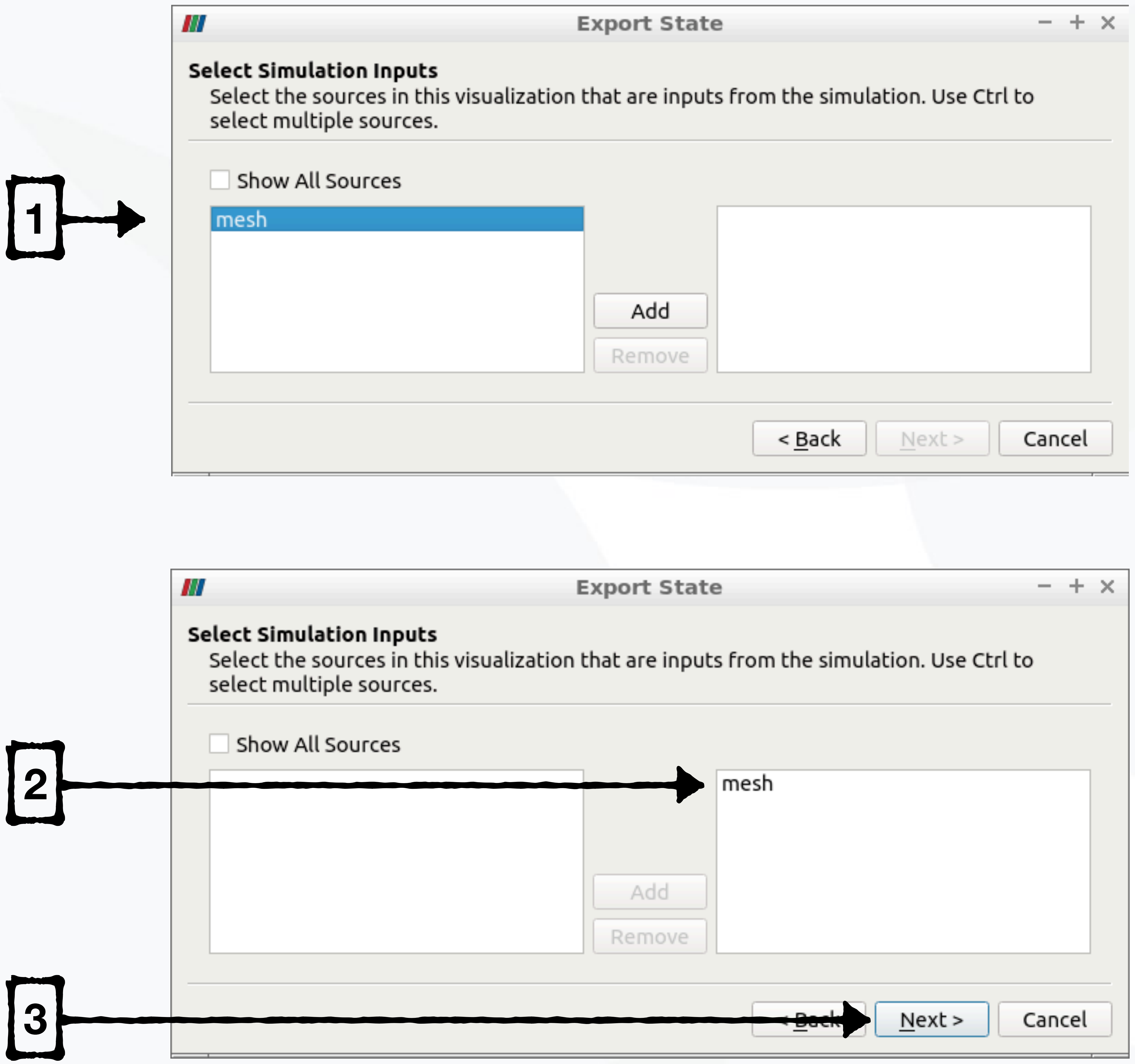


Creating a Python Pipeline

1 • Choose the datasets from ParaView that will be provided by your simulation via SENSEI.

2 • Click "Add" for each dataset.

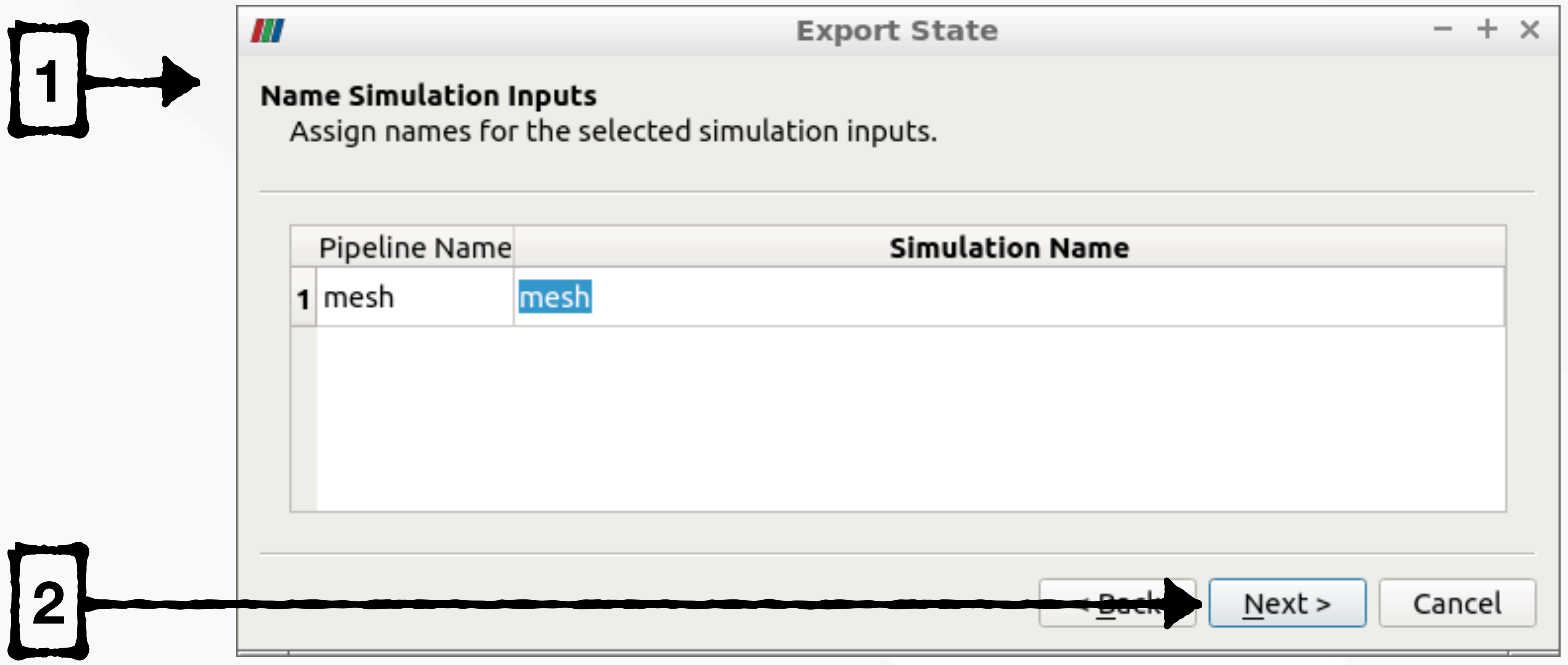
3 • Then click "Next".



Creating a Python Pipeline

1 • For each dataset from ParaView, set the "Simulation Name" used by SENSEI to identify that mesh. The names on the right should be mesh names from your data adaptor.

2 • Click "Next".



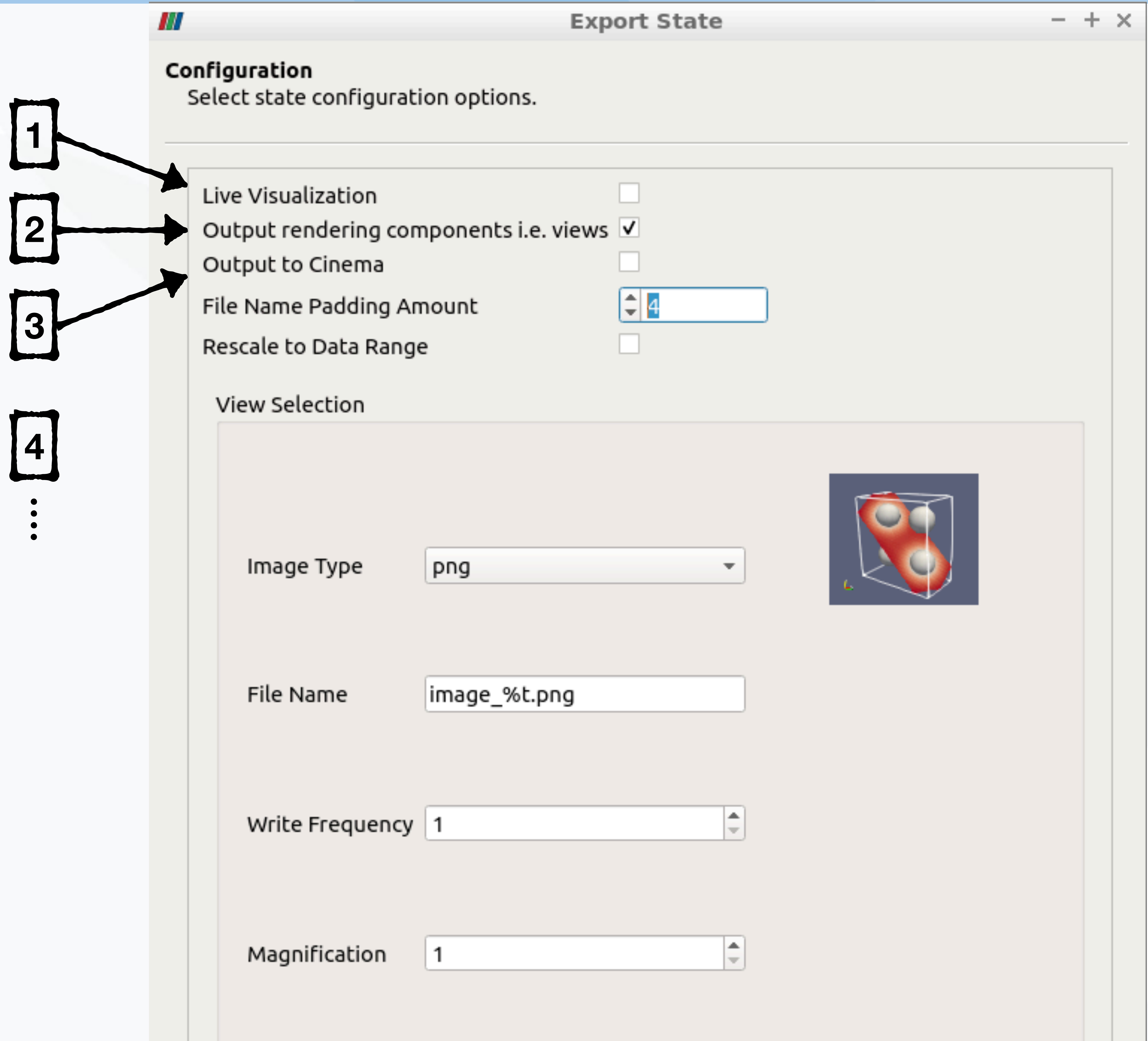
Creating a Python Pipeline

1 • "Live Visualization" will create a script that attempts to connect to ParaView at each timestep.

2 • "Output rendering..." will create a script that saves image sequences.

3 • "Output to Cinema" will create a script that saves composable depth images.

4 • Set other options; click "Finish".

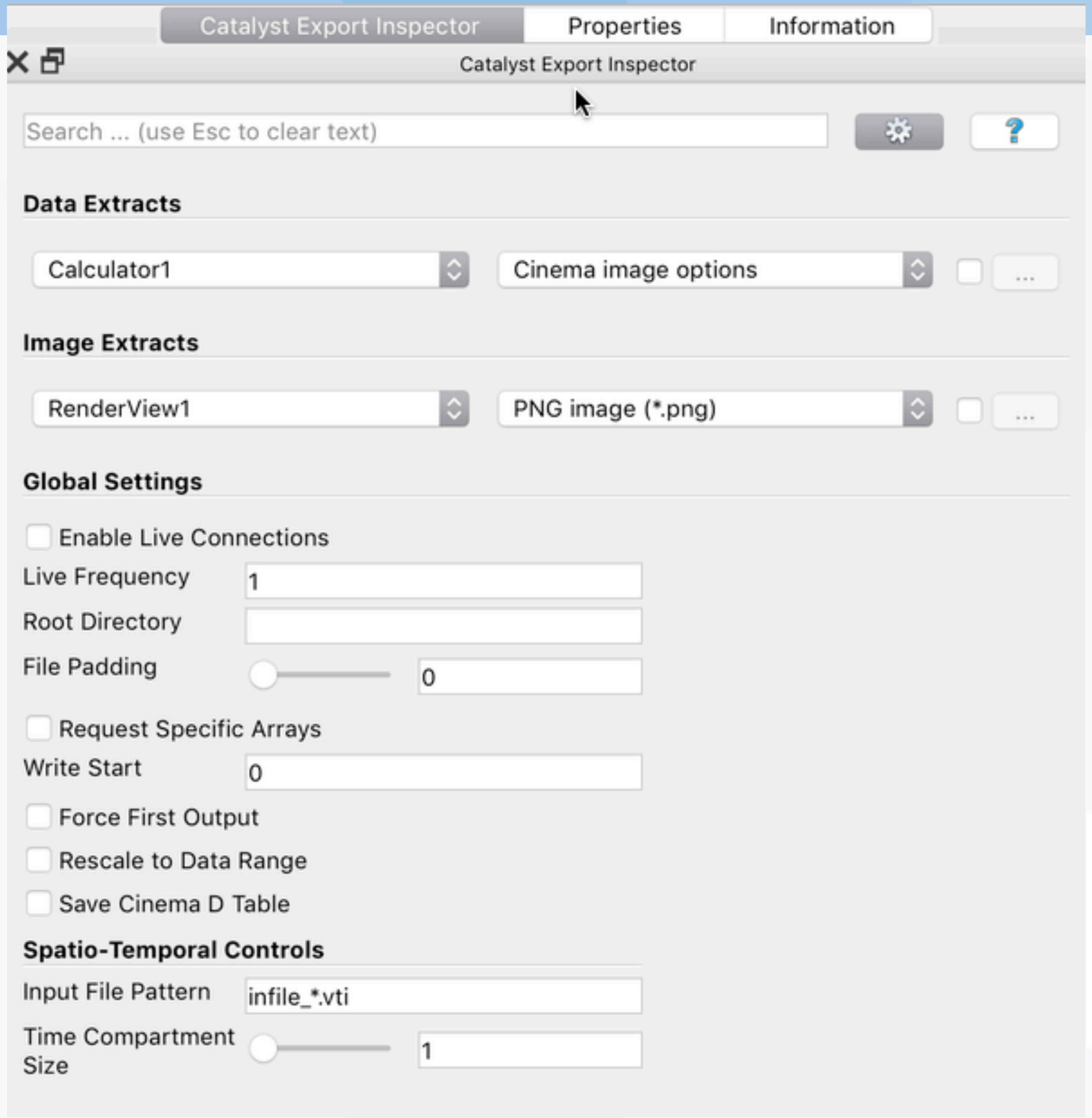


Pipelines for ParaView 5.6.0

The following slides show how to create Python pipeline scripts using ParaView version 5.6.0, which is soon to be released and significantly different/improved.

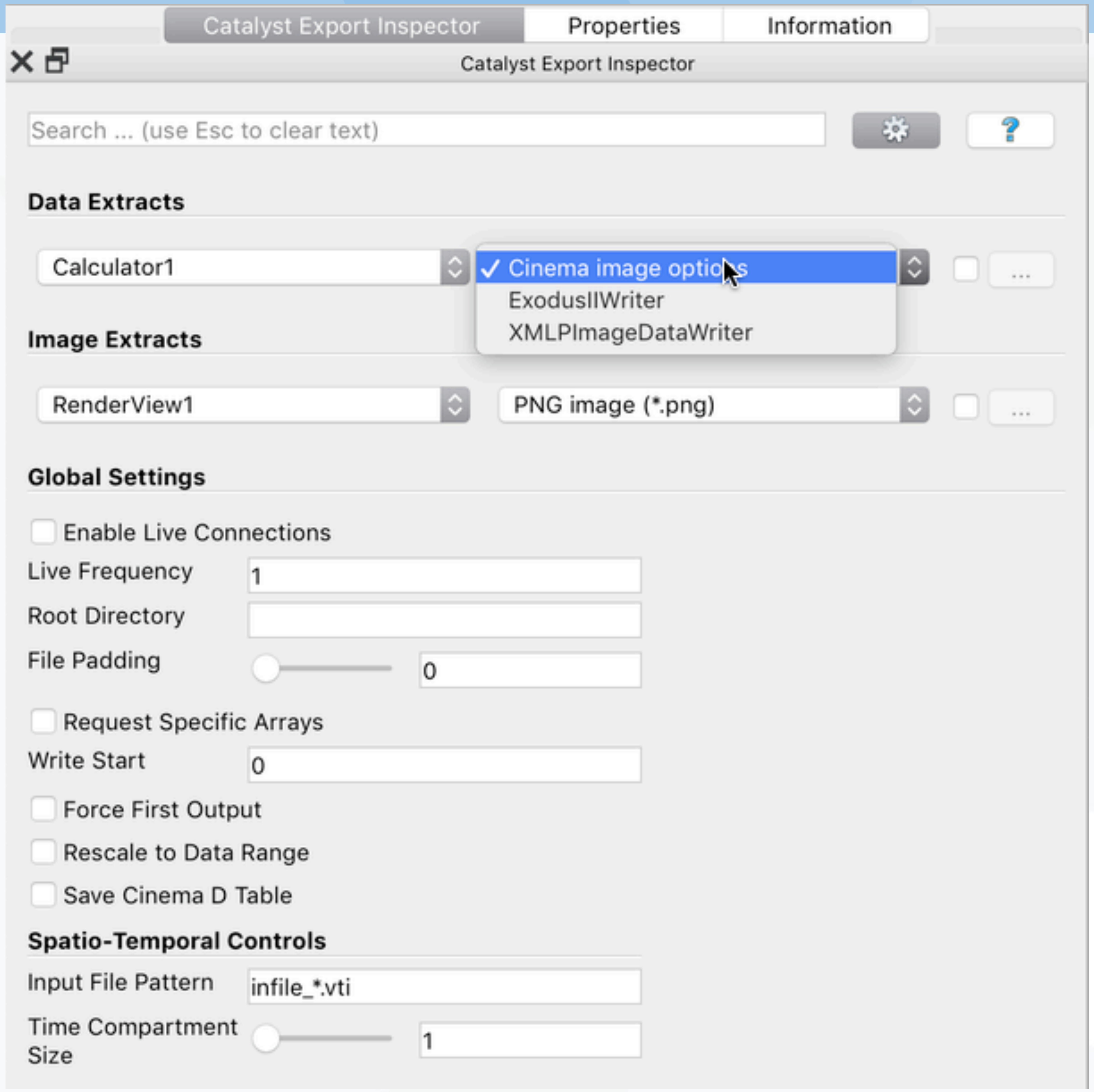
Python Pipelines

- The Catalyst Export Inspector panel can save
  - Data Extracts, which write filtered datasets using VTK's I/O libraries
  - Image Extracts, which render filtered data and save image sequences or Cinema databases
- The Enable Live Connections checkbox tells Catalyst to look for ParaView client connections



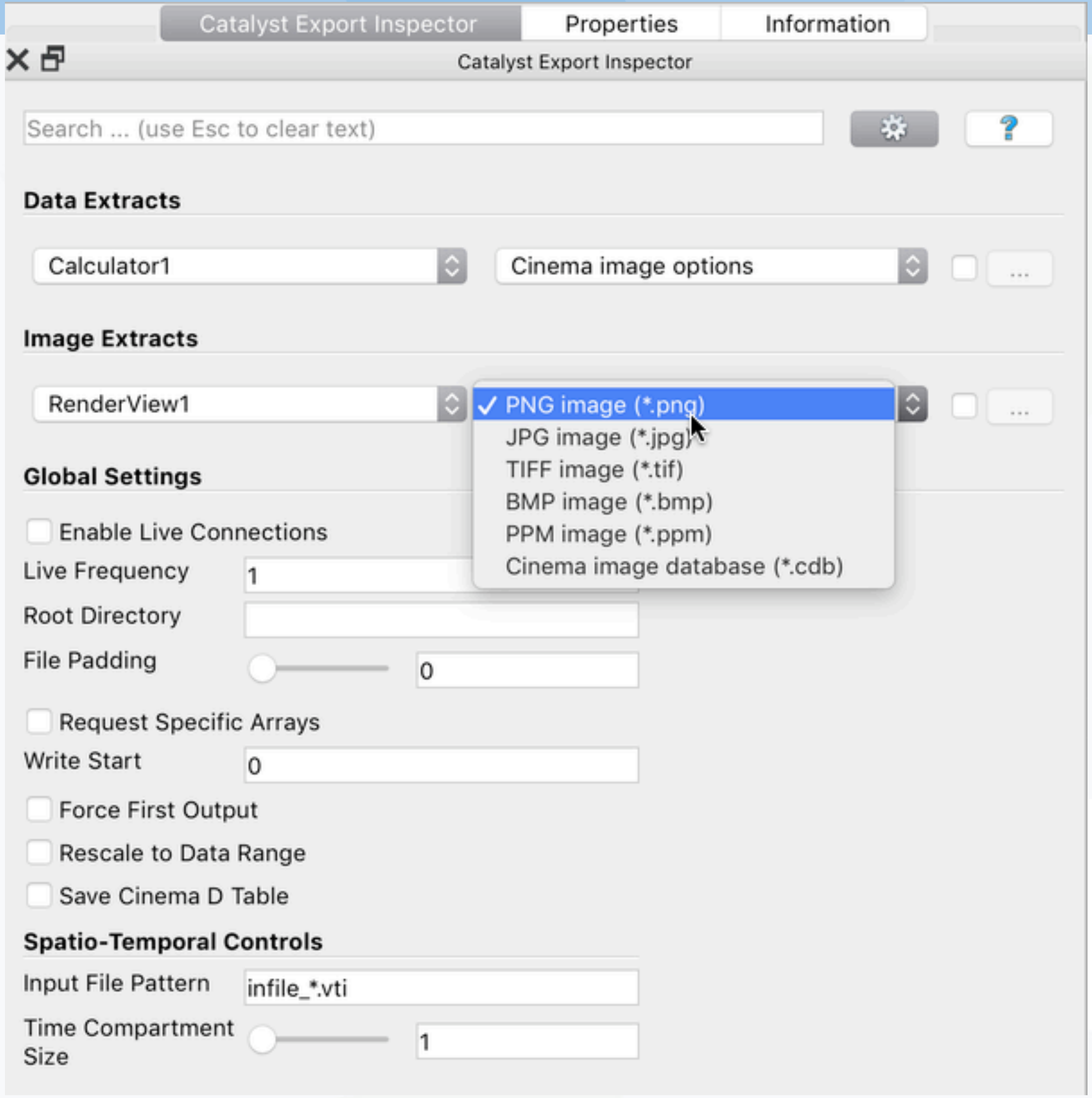
Python Pipelines

- The Catalyst Export Inspector panel can save
  - Data Extracts, which write filtered datasets using VTK's I/O libraries
  - Image Extracts, which render filtered data and save image sequences or Cinema databases
- The Enable Live Connections checkbox tells Catalyst to look for ParaView client connections



Python Pipelines

- The Catalyst Export Inspector panel can save
  - Data Extracts, which write filtered datasets using VTK's I/O libraries
  - Image Extracts, which render filtered data and save image sequences or Cinema databases
- The Enable Live Connections checkbox tells Catalyst to look for ParaView client connections



Python Pipelines

- Now configure SENSEI to run the Catalyst Python pipeline with an XML configuration file for SENSEI's ConfigurableAnalysis:

Exercises 3 & 4

```
cd ~/sensei_insitu/demos/sc18/pv_catalyst              ← On the VM
cd /project/projectdirs/m636/sensei_insitu/            ← On Cori
    demos/sc18/pv_catalyst
./demo 3 username                                      ← On either
./demo 4 username
```

- Demo:
  - Create a visualization of oscillator mini-app data
  - Save a Catalyst script
- Exercise
  - Configure the oscillator to use the Catalyst script
  - Run the oscillator again using the flexible, run-time pipeline

Live Connections

- With ParaView Live connections,
    - Catalyst will check for a ParaView client connection request at the beginning of each timestep.
    - If present, a TCP/IP connection between the client and simulation is used to bootstrap a connection between the simulation and ParaView's server (which may be running in parallel on the same or different nodes of the cluster).
    - Datasets are transmitted upon demand (by the GUI client) from the simulation to the ParaView server, where they can be filtered and rendered in parallel.

Live Connections

- To enable ParaView Live, edit your Catalyst pipeline Python script; change this:

```
# Enable Live-Visualization with ParaView and the update frequency
coprocessor.EnableLiveVisualization(False, 1)
```

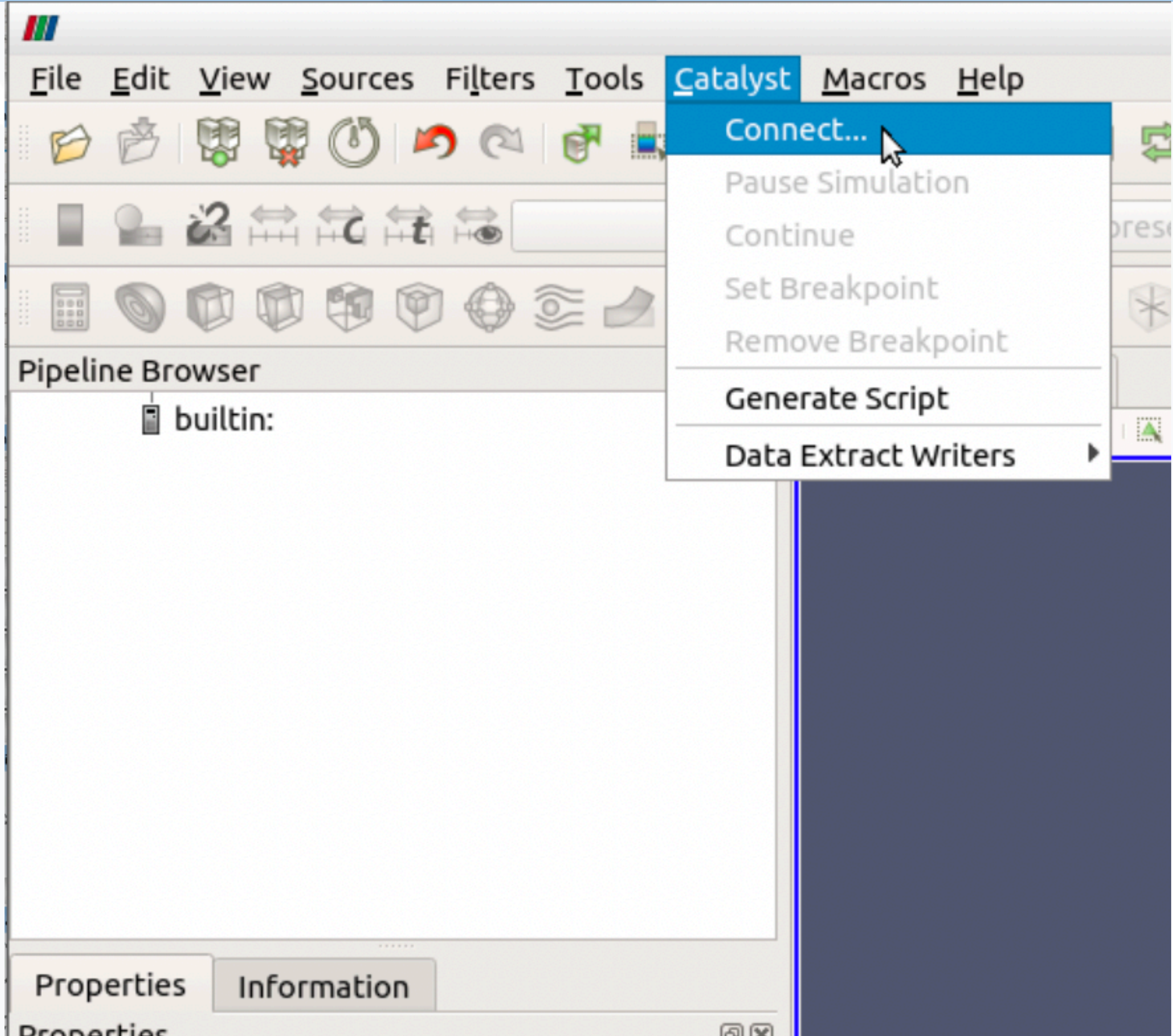
- to this:

```
# Enable Live-Visualization with ParaView and the update frequency
coprocessor.EnableLiveVisualization(True, 1)
```

Live Connections

- Before starting your simulation, run the ParaView client, connect to the remote server (if you want to perform parallel rendering), and tell ParaView to accept Catalyst connections.

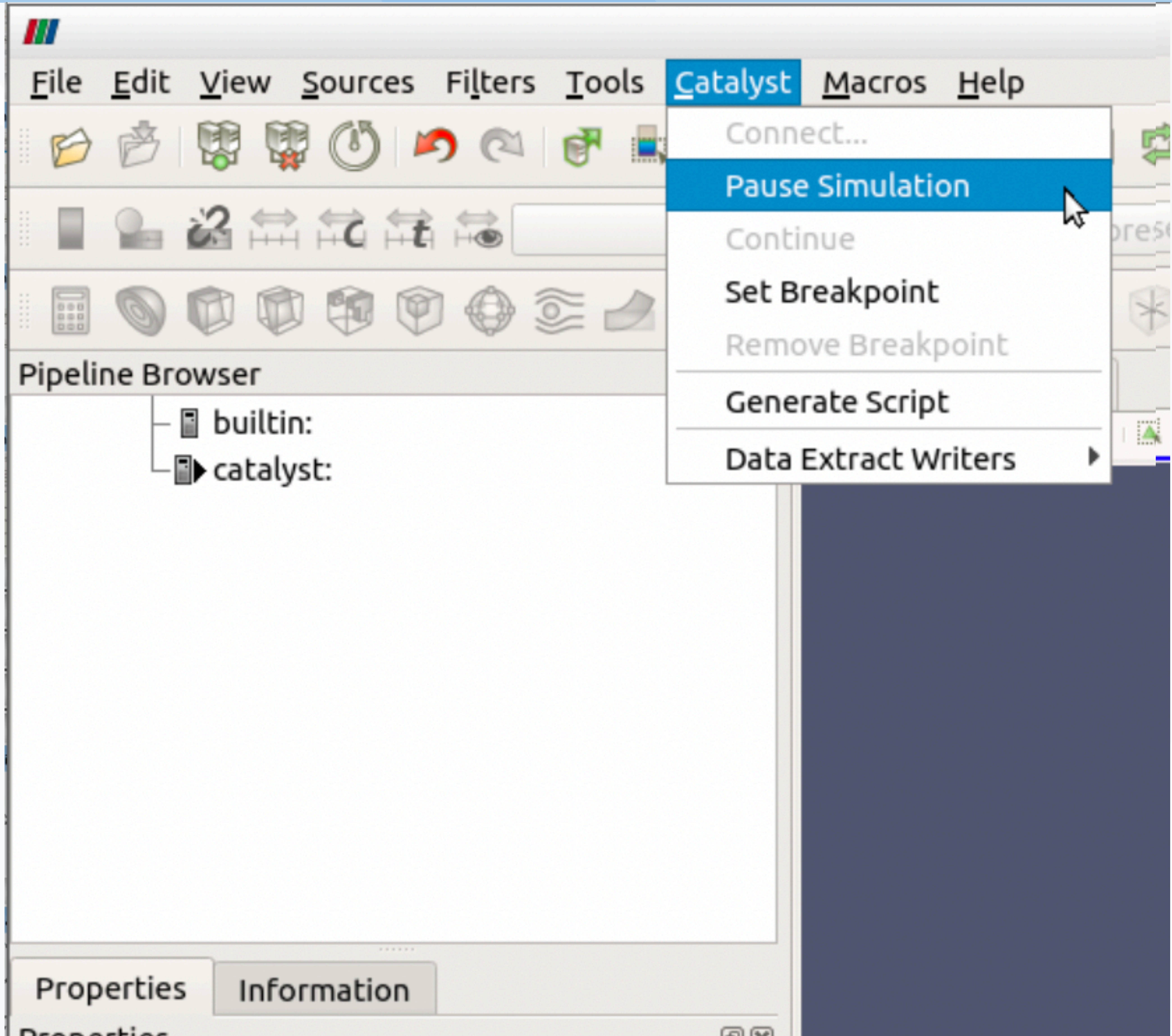
- You may also want to pause Catalyst, which will halt the simulation when it connects so you have an opportunity to configure ParaView.



Live Connections

- Before starting your simulation, run the ParaView client, connect to the remote server (if you want to perform parallel rendering), and tell ParaView to accept Catalyst connections.

- You may also want to pause Catalyst, which will halt the simulation when it connects so you have an opportunity to configure ParaView.



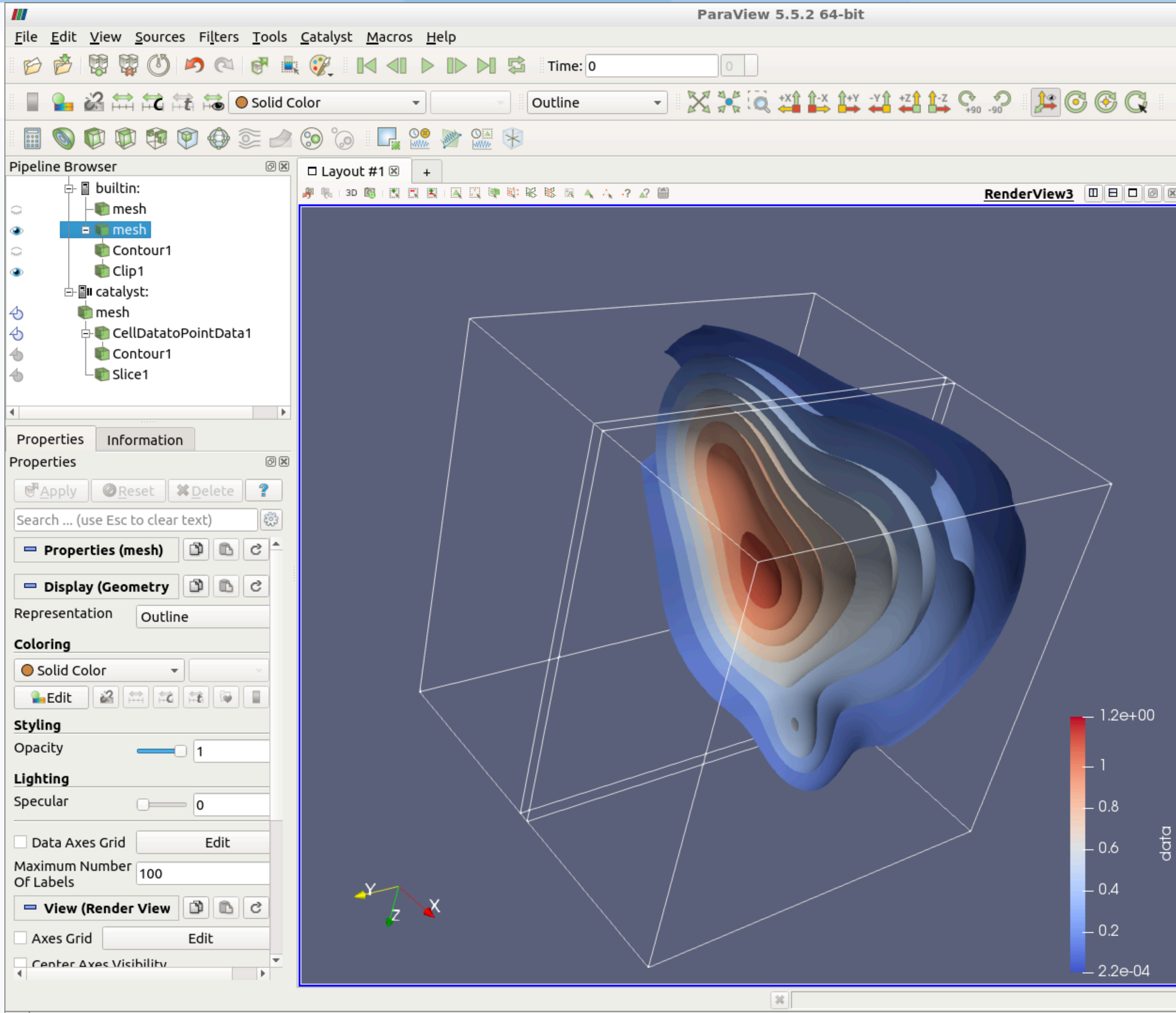
Live Connections

- The builtin or cori server is data present on ParaView's server process(es).
- The catalyst "server" is data present in the simulation.
- Clicking on catalyst pipelines will transfer the data to ParaView's server process(es).

Simulation running

Simulation paused

catalyst: **Simulation running; breakpoint set**



Demo / Exercises 5 & 6

```
cd ~/sensei_insitu/demos/sc18/pv_catalyst
./demo 5 username
./demo 6 username
```

This only runs on the VM

- Demo:
  - Edit a Catalyst script to enable Live connections
- Exercise
  - Run ParaView and accept connections from Catalyst
  - Run the oscillator and connect using ParaView Live

Getting Help

- User's Guide: http://www.paraview.org/paraview-guide
- Discourse Forum: https://discourse.paraview.org/
- Websites
  - http://www.paraview.org/
  - http://www.paraview.org/in-situ/
  - http://www.cinemascience.org/

SENSEI In Situ Demonstrations

Integrating VTK-m and Cinema into SENSEI

BERKELEY LAB · Argonne NATIONAL LABORATORY · OAK RIDGE National Laboratory · Intelligent Light · Kitware

SC18 Dallas, TX | hpc inspires.

Agenda

- Overview of VTK-m
- Requirements
- Instrumentation Examples
  - Direct access
  - vtkmlib from VTK
- Demo / Exercise

VTK-m

- VTK-m is a "m"any-core version of VTK that *also* integrates
  - new C++ features not available in 1993.
  - design changes based on the VTK community's experience.
- VTK-m is designed around *worklets* that evaluate a single point or cell.
- Algorithms in VTK-m are cross-compiled to run on Cuda and TBB.
- VTK-m datasets are structurally different than VTK data objects.

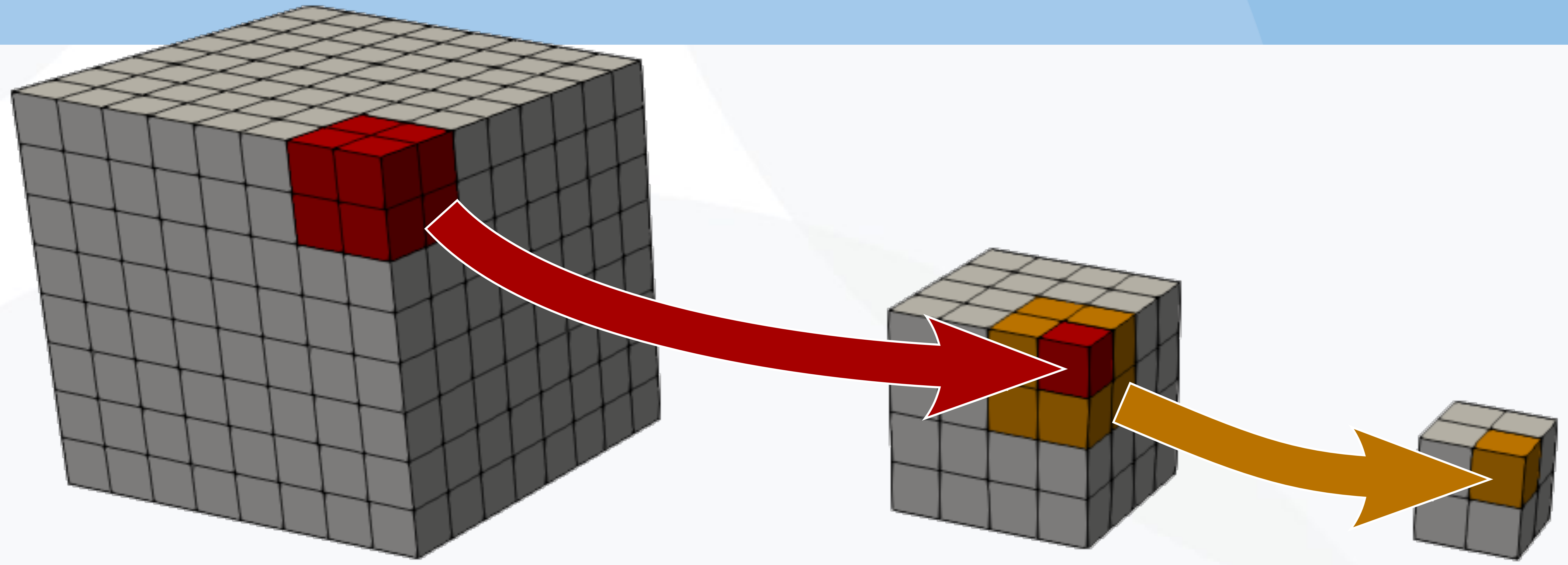
Requirements

- SENSEI is targeting the version of VTK-m that will ship with VTK 8.2.0.
- Since VTK 8.2.0 has not been released, the virtual machine for this tutorial comes with a build against a known-good version of VTK & VTK-m.

Instrumenting VTK-m

- Preferred: Use vtkmlib from `VTK/Accelerators/Vtkm/vtkmlib`
  - Construct VTK datasets from VTK-m datasets without copying large arrays.
  - Pass the resulting datasets to SENSEI's data adaptor.
- Direct access
  - Simply create vtkDataArray subclasses that reference external memory.
  - This is not recommended as it does not generalize.

Exercise: Haar wavelet



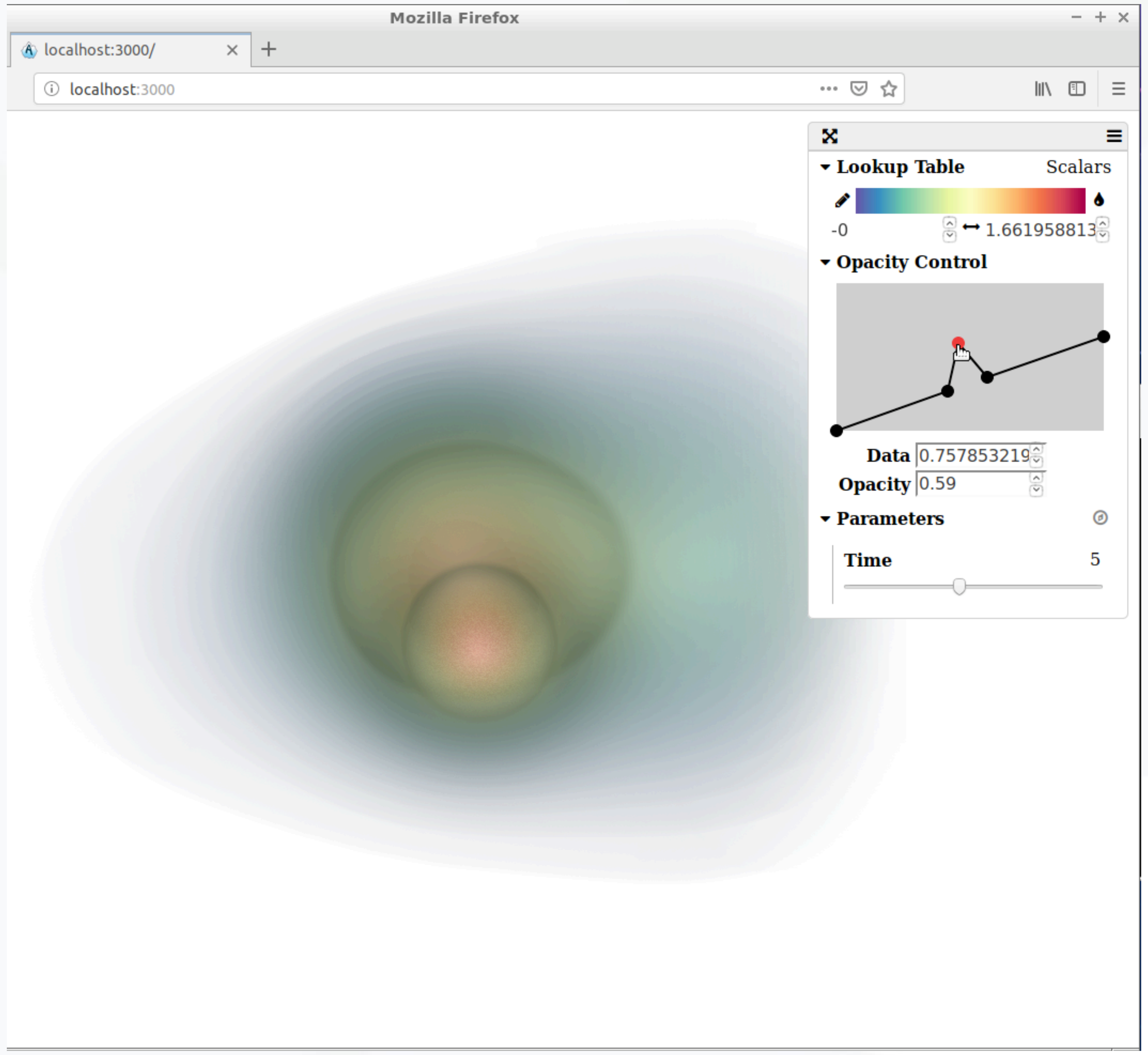
- The Haar wavelet basis is simple to compute; discarding the second set of coefficients halves the size of the dataset. Applying once along each coordinate axis cuts dataset size by 8.

- Applying the Haar and discarding part of its basis results in a low-spatial resolution dataset that is much smaller; it may serve as a global simulation summary over time, especially when combined with other techniques.

Demo / Exercise

```
cd ~/sensei_insitu/demos/sc18/vtk-m
./demo 1 username
./demo 2 username
```

This only runs on the VM



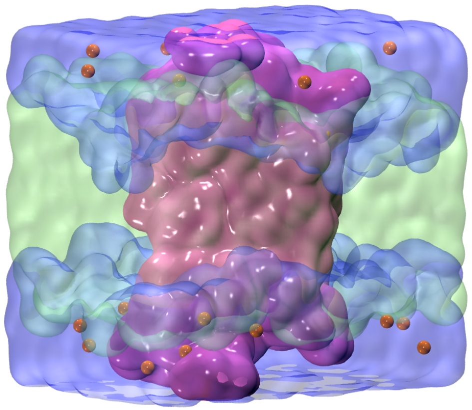
- Exercise
  - Run the oscillator, saving out Haar-transform-reduced datasets in Cinema format
  - Visualize the resulting data in a web browser using arctic-viewer.

Instrumenting LAMMPS with SENSEI

LAMMPS

- Large-scale Atomic/Molecular Massively Parallel Simulator
- Classical molecular dynamics code
- Runs on single processors or in parallel using message-passing techniques and a spatial-decomposition of the simulation domain
- Accelerated performance on CPUs, GPUs, and Intel Xeon Phis
- Distributed by Sandia National Laboratories

http://lammps.sandia.gov/



LAMMPS rhodopsin benchmark (32,000 atoms).
Courtesy Malakar et al. "Optimal scheduling of in situ analysis for large-scale scientific simulations." SC 2015.

Enabling in situ interactive visualization for large-scale molecular simulations

- LAMMPS is a good representative application of large scale molecular dynamics simulations
- We use LAMMPS as a library
  - No need to recompile or instrument LAMMPS original code
- Drive LAMMPS from a simple application instrumented with SENSEI
- Integrate OSPRay (Intel Software-Defined visualization) as an additional SENSEI infrastructure for interactive visualization
- Use libIS as a lightweight in transit library

W.Usher, S. Rizzi, I. Wald, J. Amstutz, J. Insley, V. Vishwanath, N. Ferrier, M.E. Papka, V. Pascucci. ***libIS: A Lightweight Library for Flexible In Transit Visualization***. ISAV 2018.

Data format

- LAMMPS particle format is basically x,y,z coordinates with additional fields like atom type or radius
- Add LAMMPS *fix/external* command in input file for LAMMPS to share pointers to its internal data after computing every timestep of the simulation

- Additional information here: ***Coupling LAMMPS to other codes***

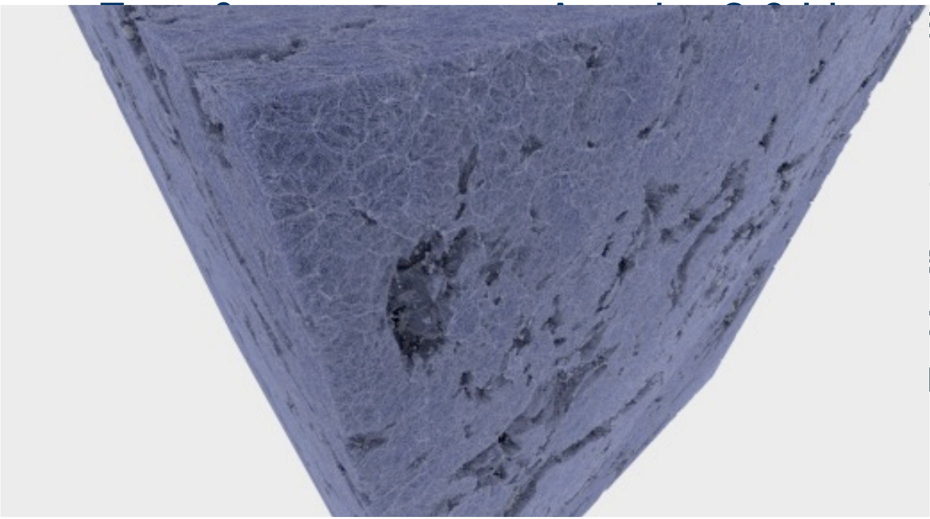
https://lammps.sandia.gov/doc/Howto_couple.html

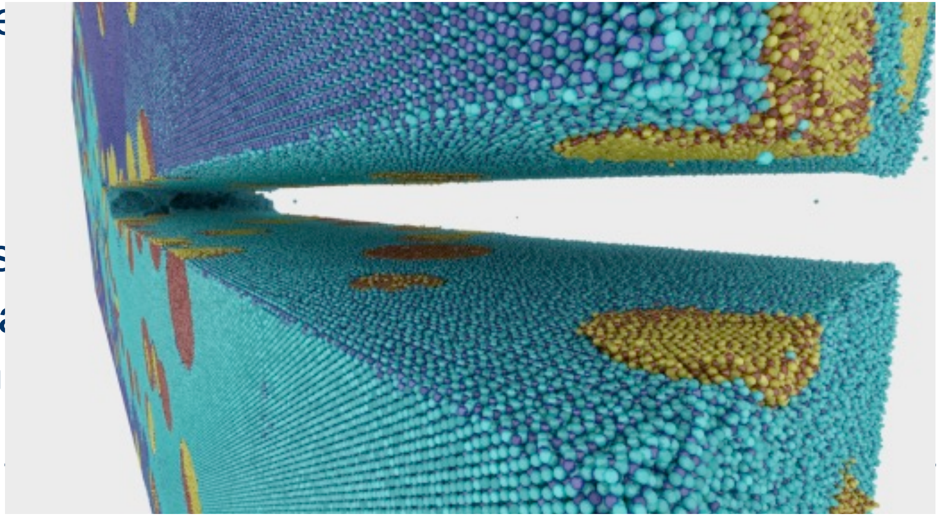
OSPRay

Wald, Ingo, Gregory P. Johnson, J. Amstutz, Carson Brownlee, Aaron Knoll, J. Jeffers, J. Günther, and P. Navratil. "OSPRay-A CPU Ray Tracing Framework for Scientific Visualization." IEEE transactions on visualization and computer graphics 23, no. 1 (2017): 931-940.

Ray tracer for interactive scientific visualization-style rendering

- Volumes, triangle meshes, non-polygonal geometry (spheres, cylinders,…)
- Ray traced shading effects for shadows, ambient occlusion

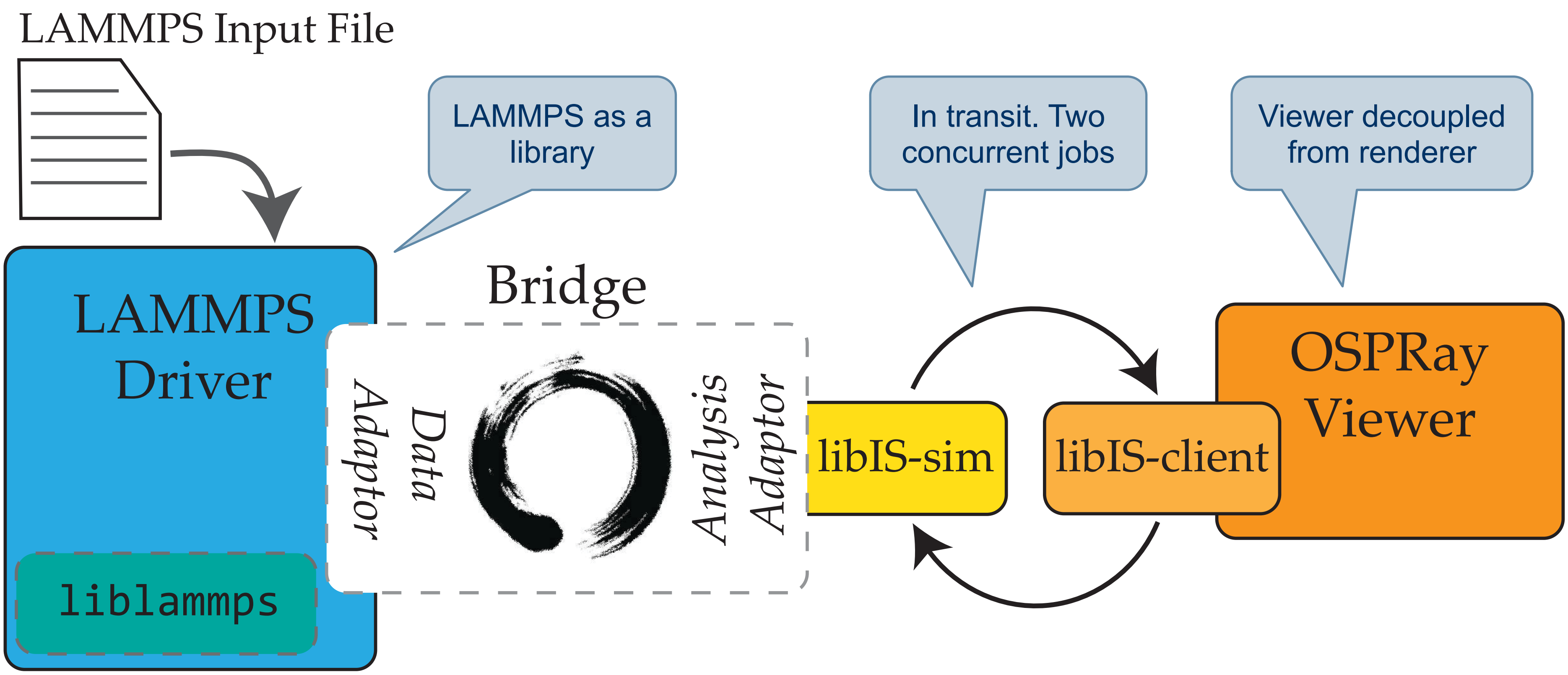




[Wald et al. '15]

Slide courtesy the OSPRay team

LAMMPS instrumentation with SENSEI and OSPRay



Callback function from LAMMPS (every timestep)

```
void LAMMPSCallback(void *ptr, bigint ntimestep,
                int nlocal, int *id, double **x, double **f)
{
    Info *info = (Info *) ptr;

    // extents
    double boxxlo = *((double *) lammps_extract_global(info->lmp,"boxxlo"));
    double boxxhi = *((double *) lammps_extract_global(info->lmp,"boxxhi"));
    double boxylo = *((double *) lammps_extract_global(info->lmp,"boxylo"));
    double boxyhi = *((double *) lammps_extract_global(info->lmp,"boxyhi"));
    double boxzlo = *((double *) lammps_extract_global(info->lmp,"boxzlo"));
    double boxzhi = *((double *) lammps_extract_global(info->lmp,"boxzhi"));

    // get pointer to atom types
    int    *type = (int *) lammps_extract_atom(info->lmp,"type");

    // update SENSEI bridge
    bridge::Set_data(nlocal, id, type, x, boxxlo, boxylo, boxzlo, boxxhi, boxyhi, boxzhi);

    // visualize
    bridge::Execute();
}
```

XYZ atom coords from LAMMPS

get atom types from LAMMPS

Update SENSEI bridge

Visualize

Materials Science with LAMMPS

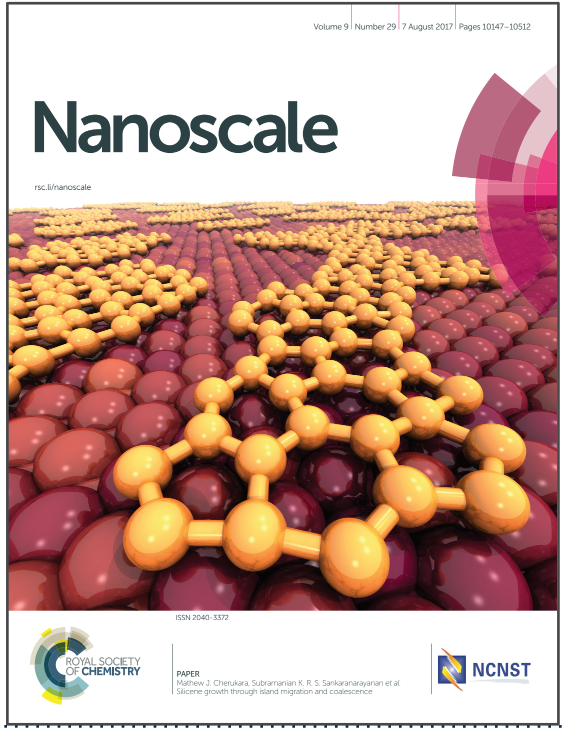
Silicene: Mono-layer Silicon / Iridium Substrate

- Massively-parallel classical molecular dynamics (MD) simulations with LAMMPS
- Various temperature conditions
- Varying rates of silicene deposition
- Characterize material structure and growth

Simulations were run on Mira at Argonne

162,000 iridium atoms

~6 Million total compute hours

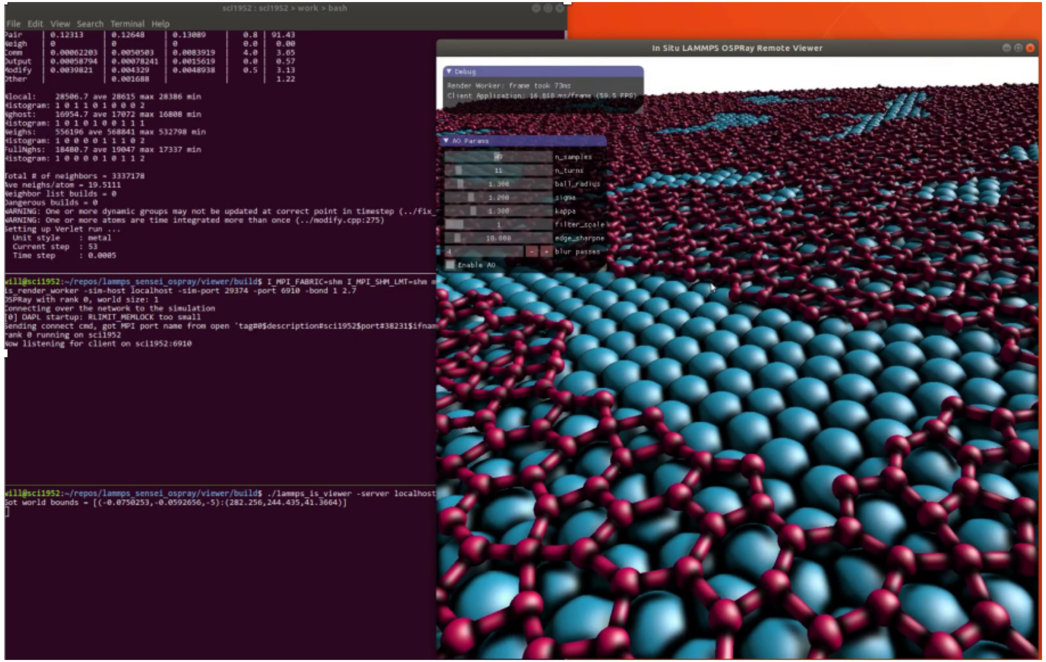


Cherukara, Mathew J., Badri Narayanan, Henry Chan, and Subramanian Sankaranarayanan. "Silicene growth through island migration and coalescence." Nanoscale 9, no. 29 (2017)

Slide courtesy Joe Insley, Argonne National Laboratory

Live demo

- Live demo on virtual machine
  - Running LAMMPS coupled to OSPRay for interactive visualization
  - Navigation: Use RIGHT click to zoom in/out, LEFT click to rotate



- Steps:

Open a terminal

```
% cd ~/sensei_insitu/demos/sc18/lammps
% ./silicene-demo-sc18.sh
```

BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY

SENSEI in situ

In Situ Costs and Performance

BERKELEY LAB
Argonne NATIONAL LABORATORY
OAK RIDGE National Laboratory
Intelligent Light
Kitware

What is the cost of *in situ* processing?

Concern: simulations want to use all available resources, so having an understanding of *in situ* resource utilization is useful.

In other words: In situ infrastructure must play nicely with simulation

Full details in SC16 paper: Utkarsh Ayachit, Andrew Bauer, Earl P. N. Duque, Greg Eisenhauer, Nicola Ferrier, Junmin Gu, Kenneth E. Jansen, Burlen Loring, Zarija Lukic, Suresh Menon, Dmitriy Morozov, Patrick O'Leary, Rateesh Ranjan, Michel Rasquin, Christopher P. Stone, Venkat Vishwanath, Gunther H. Weber, Brad Whitlock, Matthew Wolf, K. John Wu, and E. Wes Bethel, Performance Analysis, Design Considerations, and Applications of Extreme-scale In Situ Infrastructures. In Proceedings of SC16, November 2016.

Shared resources

- Initialization costs need to be monitored
  - Static build options important as HPC simulation size increases
  - Initialization costs do get amortized
- Finalization costs can be a factor for certain in situ algorithms
- Memory costs can be a factor
  - Shared memory usage for simulation and in situ arrays ("zero copy")
  - Request only needed arrays through the DataAdaptor's AddArray() method
  - Some analysis algorithms can require a lot of memory
  - Autocorrelation could potentially need to store full data at each time step. Use autocorrelation window size to reduce the amount of time steps stored

In situ compute

- In situ computation may not need to be done every time step
  - Lower fidelity time stepping output
  - Only when something "interesting" is happening

- Can still reduce output size
  - Image output is fixed size and independent of simulation size
  - Coarsen data extracts
  - Compute summary statistics (e.g. autocorrelation, histogram)

Three key performance analysis focus areas

One-time costs: initialization

- Some *in situ* setups may entail non-zero initialization costs, e.g.:
  - Per-rank config file processing

Recurring costs

- Execution time:
  - Different methods require differing amounts of computation
  - Algorithmic complexity at scale
  - *In situ* methods that use reductions
  - *In situ* vs. in transit tradeoffs
- Memory consumption
  - Temporal analysis methods must buffer more data

One-time costs: finalization

- Some *in situ* setups may entail non-trivial initialization costs, e.g.:
  - Global reductions
- Gives insights into ways to optimize

Measuring the cost of *in situ*

Two questions:

How much overhead associated with use of *in situ* methods, infrastructure (runtime, memory)?

Does this change with varying concurrency?

Additionally:

In situ and in transit configurations

In situ and *post hoc:* end-to-end comparison

U. Ayachit, A. Bauer, E. P. N. Duque, G. Eisenhauer, N. Ferrier, J. Gu, K. E. Jansen, B. Loring, Z. Lukic, S. Menon, D. Morozov, P. O'Leary, R. Ranjan, M. Rasquin, C. P. Stone, V. Vishwanath, G. H. Weber, B. Whitlock, M. Wolf, K. Wu, and E. W. Bethel. Performance Analysis, Design Considerations, and Applications of Extreme-scale In Situ Infrastructures. In Proceedings of SC16, November 2016.

Methodology for measuring cost of *in situ*

Miniapplication: data source (next slide)

In situ methods

- Histogram computation
- Autocorrelation computation (temporal analysis)
- Extract and render a 2D slice from a 3D volume

In situ infrastructures

- VisIt/Libsim
- ParaView/Catalyst
- ADIOS

Measure:

- Runtime and memory footprint
- At varying levels of concurrency
- One-time and recurring

Test Platform

Cori Phase I at NERSC
Cray XC system
1630 compute nodes
Dual 2.3Ghz 16-core Intel Haswell processors
128GB RAM/node

Concurrency levels of tests:
812 (~1K)
6496 (~6K)
45440 (~45K)

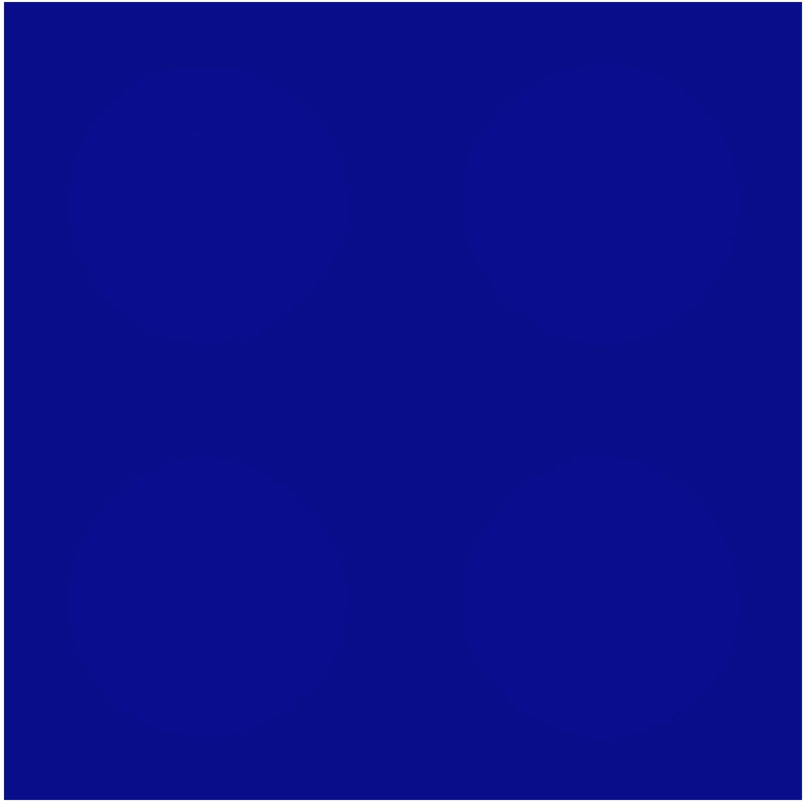
Miniapplication - oscillators

Bulk-synchronous parallel computation of periodic, damped oscillators (MPI-based app)

No interprocess communication - entirely analytic, embarassingly parallel

For *m* oscillators and per-rank grid size of N^3:

- Per-rank memory footprint: $2N^3$
- Per-rank complexity: mN^3



Miniapp configurations – *in situ* methods

Configuration	Intention
Original	Miniapp with no SENSEI interface, no I/O. Direct-coupling (subroutine call) to analysis methods Measure runtime/memory with no *in situ*
Baseline	Miniapp with the SENSEI interface enabled No analysis or I/O Measure overhead of *in situ* interface in isolation
Histogram	Miniapp+SENSEI interface+histogram computation No *in situ* infrastructures Compare performance to *Original, Baseline*
Autocorrelation	Miniapp+SENSEI interface+autocorrelation computation No *in situ* infrastructures Compare performance to *Original, Baseline*

Miniapp configurations – with *in situ* infrastructures

Configuration	Intention
Catalyst-slice	Miniapp + SENSEI interface + Catalyst Catalyst performs a 2D slice extraction of 3D volume Followed by parallel rendering, produces an image Compare to *Original, Baseline*
Libsim-slice	Miniapp + SENSEI interface + Libsim Libsim performs a 2D slice extraction of 3D volume Followed by parallel rendering, produces an image Compare to *Original, Baseline*
ADIOS-FlexPath	Miniapp + SENSEI interface + ADIOS/FlexPath In transit implementation of histogram, autocorrelation, Catalyst-slice Compare to *Original, Baseline*

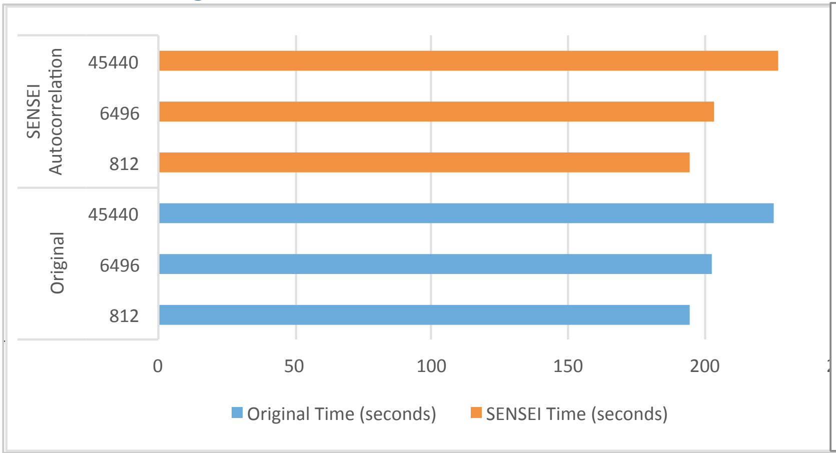
Measuring impact of SENSEI interface

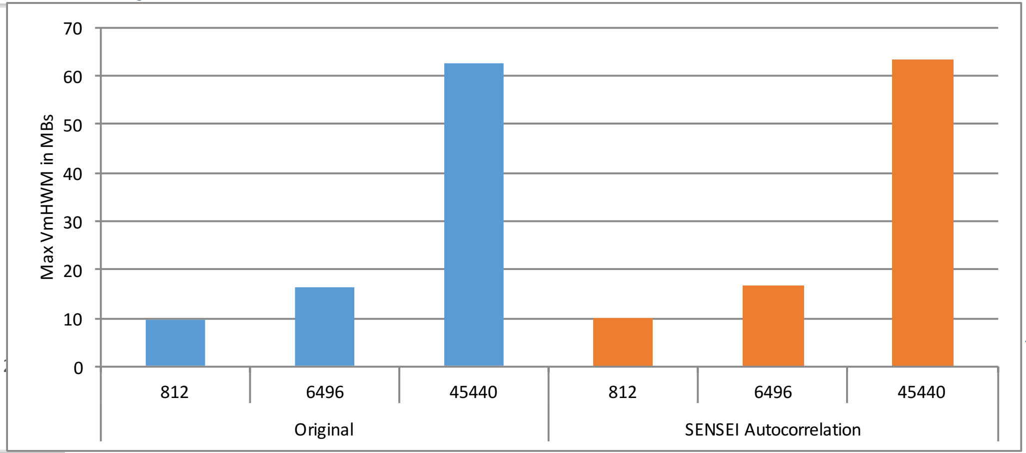
Run *Original* and *Baseline* configs, 3 levels of concurrency: 1K, 6K, 45K

- Original: miniapp + subroutine called autocorrelation
- Baseline: miniapp + SENSEI bridge to autocorrelation

Compare runtime (left), memory footprint (right)

No significant difference reflects zero-copy nature of the interface





Comparing *in situ* to *post hoc*

Post hoc configuration

- Simulation computes something
- Then writes results to disk
- Post hoc method reads from disk and performs analysis

In Situ configuration

- Simulation computes something
- Then *in situ* method computes something
- (No disk I/O involved)

Post hoc study concurrency

Simulation	Postprocess
812	82
6496	650
45440	4545

Weak-scaling Study

- Measure post hoc end-to-end cost
  - Sim writes, post hoc reads, processing
- Compare to *in situ* configurations
- Also measure time-to-solution for 100 timesteps

Post hoc: cost of writes

Baseline miniapp with the addition of parallel I/O

- VTK I/O, non-collective
- MPI-IO collective is slower (see the paper)
- This is not an I/O study. ☺ We used the fastest I/O approach we could get our hands on.

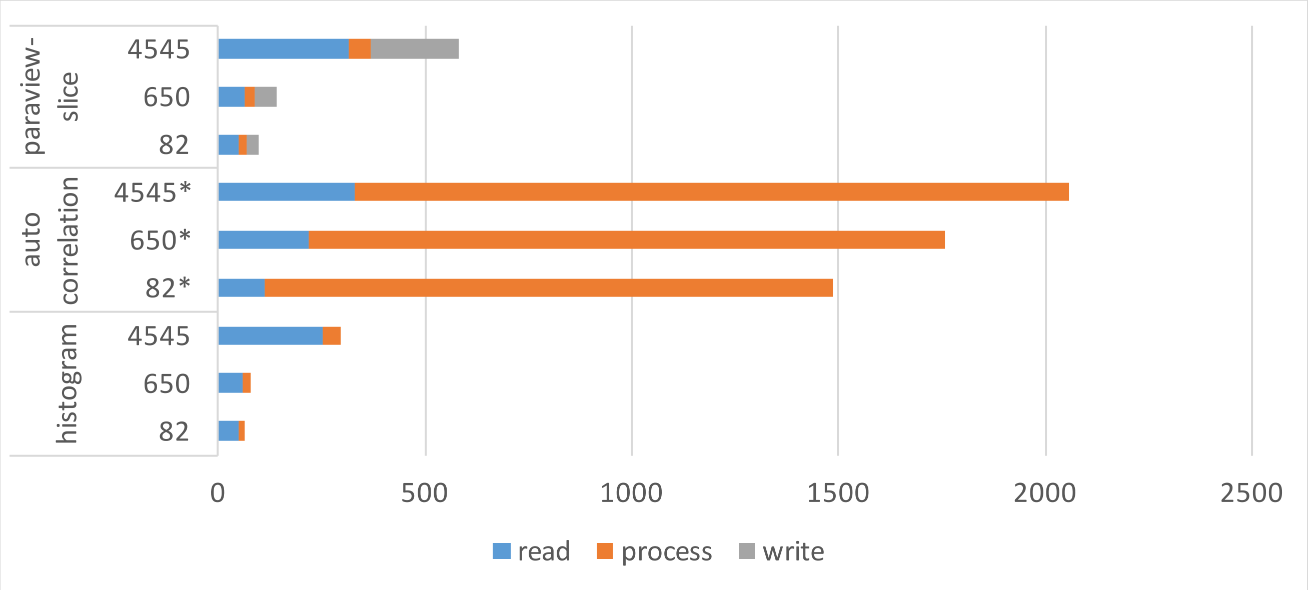
Weak-scaling: linear increase with problem size

I/O cost is significant at high concurrency

Cost of Writes

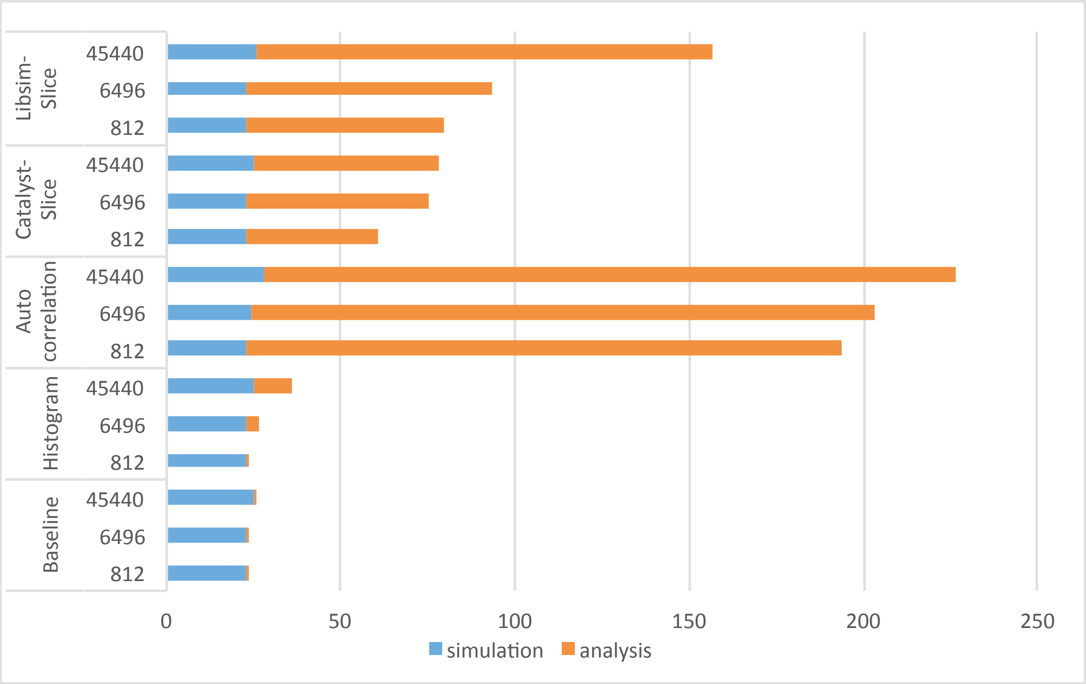
Concurrency	1 step	Aggregate
812	2 GB, 0.12s	0.2 TB, 12s
6496	16 GB, 0.67s	1.6 TB, 67s
45440	123 GB. 9.05s	12.3 TB, 905s

Post hoc: cost of reads + processing



Time required for reads, processing, and writing (results) for post hoc methods at varying level of concurrency.

In situ: time-to-solution



Post hoc vs. *in situ* time to solution

Configuration (45K)	*In Situ*	Post hoc: sim + write + read + process
Histogram	~40s	~1230s = ~25s + ~905s + ~300s + (a few secs)
Autocorrelation	~225s	~2930s = ~25s + ~905s + ~300s + ~1700s
Catalyst-slice	~80s	~1505s = ~25s + ~905s + ~300s + ~275s

Post hoc fixed costs (at 45K): about 1200s and 12.3 TB disk space

Fewer ranks for analysis processing results in longer analysis runtime (in this 1:10 configuration, which is typical for post hoc use cases)

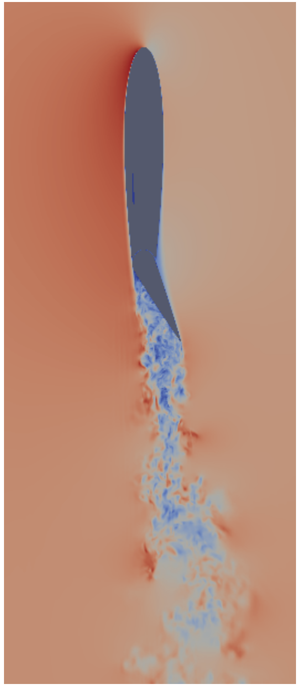
In situ at scale, Performance in the real world

BERKELEY LAB
Argonne NATIONAL LABORATORY
OAK RIDGE National Laboratory
Intelligent Light
Kitware

PHASTA: Computational Fluid Dynamics

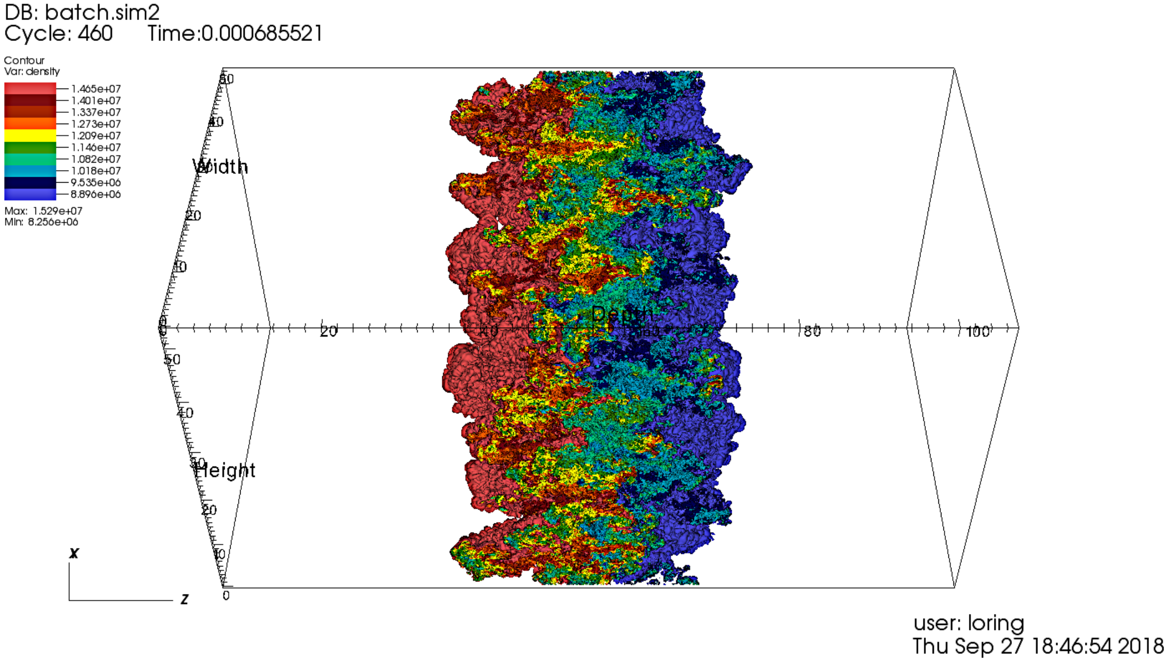
PHASTA from UC Boulder run on Mira@ANL

- Simulation of realistic geometry tail rudders and active flow control
- Coupled via SENSEI interface to Catalyst-slice, producing an output image
  - Field data, nodal coordinates: zero copy
  - Connectivity data: full copy
- Runs with 256K and 1M MPI ranks
  - 1M run was 4 times larger than any known *in situ* analysis run
  - Key technologies include reduced library size, simplified output specification and static linking using IBM XL compilers for fastest run times
  - In situ overhead: 8.2%, 33%, 13%
    - The 33% traced to zlib/PNG compression on rank 0



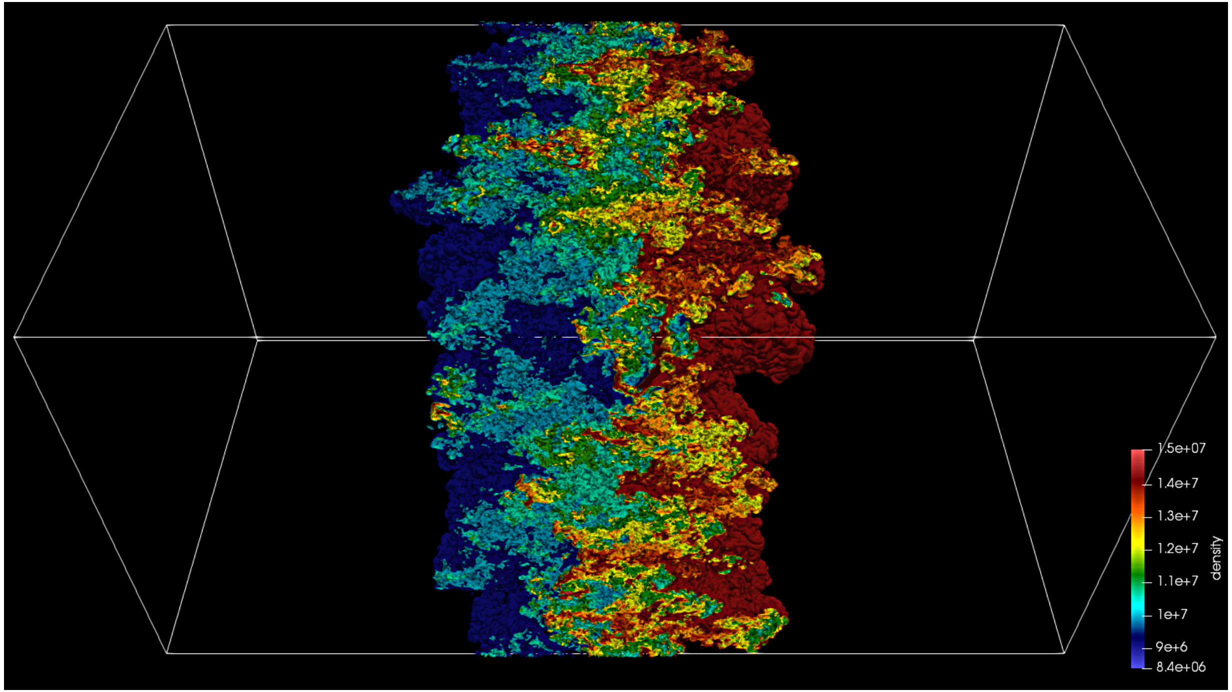
IAMR Rayleigh-Taylor Libsim

2048 Cores Cori Haswell



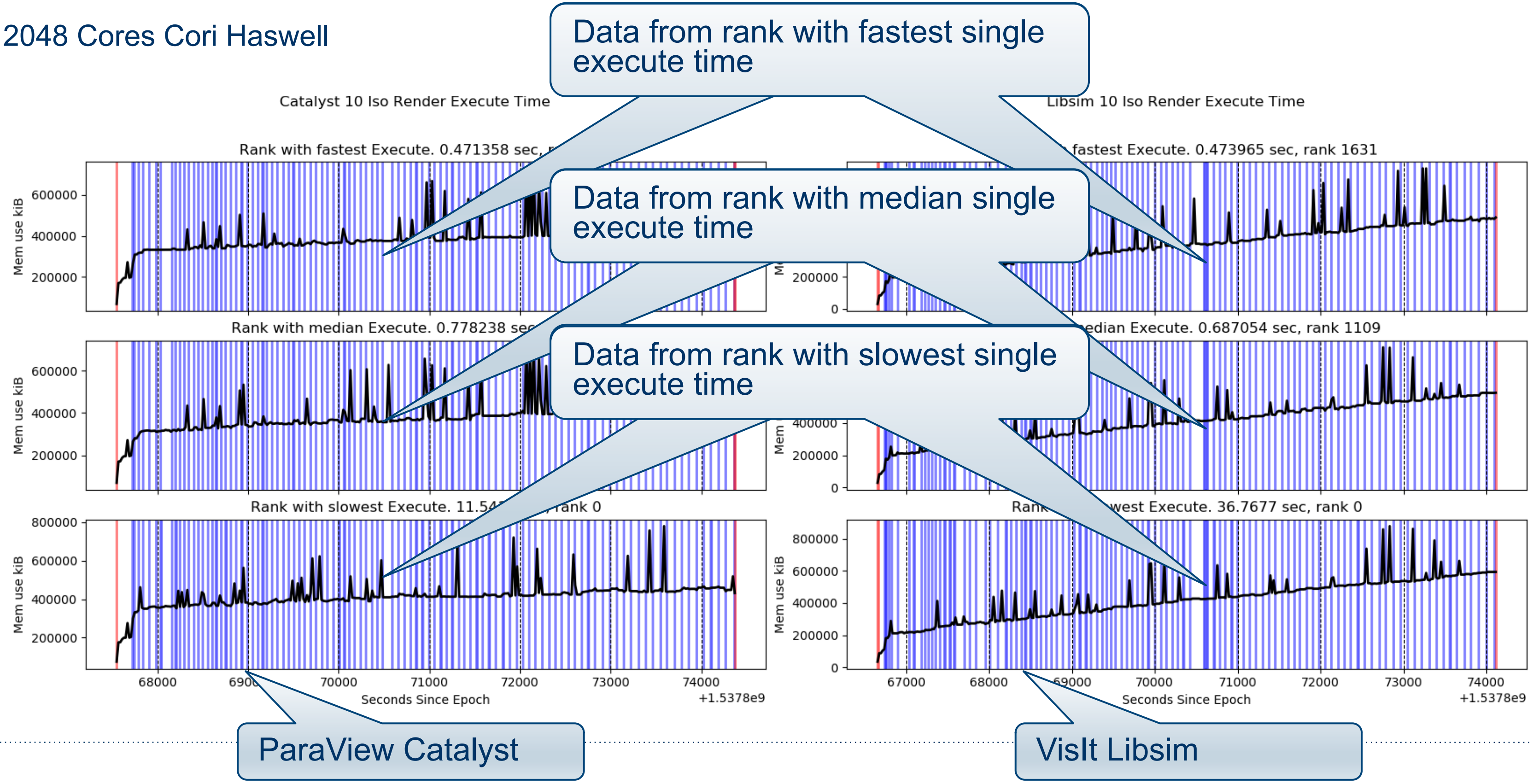
IAMR Rayleigh-Taylor Catalyst

2048 Cores Cori Haswell



Performance

2048 Cores Cori Haswell



SENSEI insitu

Wrapping Up

SC17 In Situ Tutorial Summary

- Why should you care about *in situ?*
  - Flops >> I/O; *in situ* is a viable approach for coping with this problem
- What *in situ* infrastructures are available?
- What about interfacing my sim code to them?
- What are the performance issues to be thinking about?

Links

- Main page – http://www.sensei-insitu.org/
- Software repo – https://gitlab.kitware.com/sensei/sensei
- ADIOS – https://www.olcf.ornl.gov/center-projects/adios/
- VisIt/Libsim – https://www.visitusers.org/index.php?title=Category:Libsim
- ParaView Catalyst – http://www.paraview.org/in-situ/

Tutorial evaluation

- Was this tutorial useful to you?
- Were there any subjects you'd like to see covered?
  - More of some?
  - Less of others?
- Please provide SC17 with tutorial feedback
  - https://submissions.supercomputing.org/eval.html
- Also, can provide feedback to us at:
  - Andy Bauer: andy.bauer@kitware.com
  - Wes Bethel: ewbethel@lbl.gov

Hidden

Conclusions and future work

Write once, use everywhere

Easy to add new analysis/frameworks

Understanding data transformation costs

Data Model: supporting arbitrary layouts for connectivity

Bigger runs – current best is 1Mi MPI processes on Mira@ALCF

More examples, tutorials, improved docs, etc.

SENSEI: Scalable Analysis Methods and *In Situ* Infrastructure for Extreme Scale Knowledge Discovery

BERKELEY LAB Argonne NATIONAL LABORATORY OAK RIDGE National Laboratory Kitware Intelligent Light

This work is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy, Office of Advanced Scientific Computing Research, under Contract No. DE-AC02-05CH11231, through the grant "**Scalable Analysis Methods and *In Situ* Infrastructure for Extreme Scale Knowledge Discovery**," program managers Dr. Lucy Nowell and Dr. Laura Biven.

Acknowledgment

This work is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy, Office of Advanced Scientific Computing Research, under Contract No. DE-AC02-05CH11231, through the grant "**Scalable Analysis Methods and *In Situ* Infrastructure for Extreme Scale Knowledge Discovery**," program managers Dr. Lucy Nowell and Dr. Laura Biven.

SENSEI: Scalable Analysis Methods and *In Situ* Infrastructure for Extreme Scale Knowledge Discovery


THE
END