



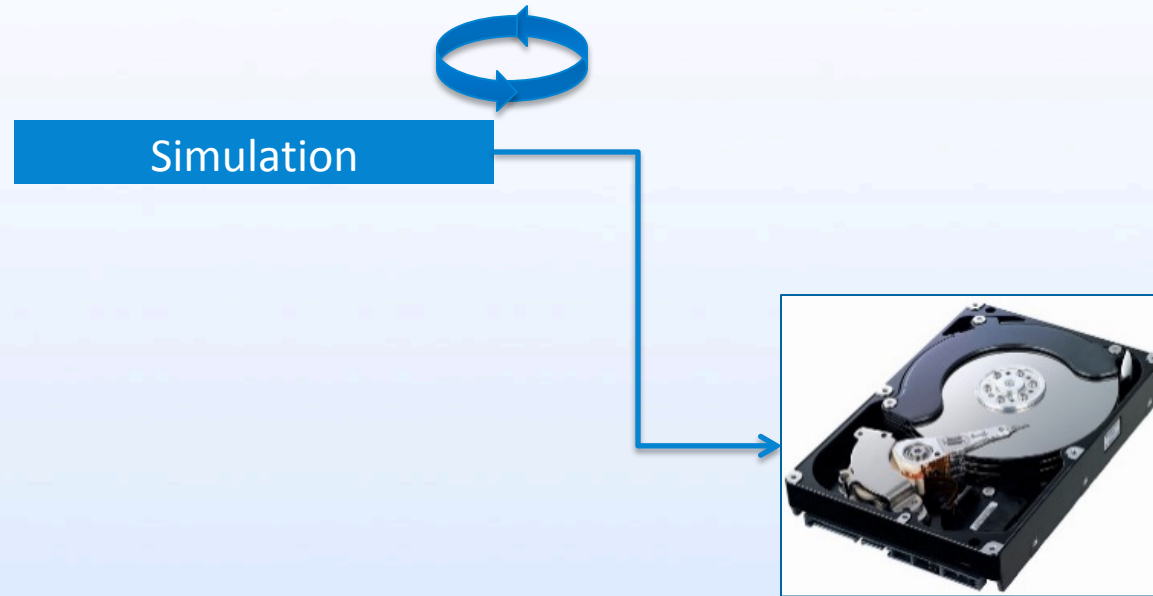
# In Situ ParaView - Catalyst Deep Dive

Dave DeMarle

IXPUG SDVis Workshop

May 2018

# Traditional/Post Hoc Analysis



# Traditional/Post Hoc Analysis

- sometime later...

# Traditional/Post Hoc Analysis



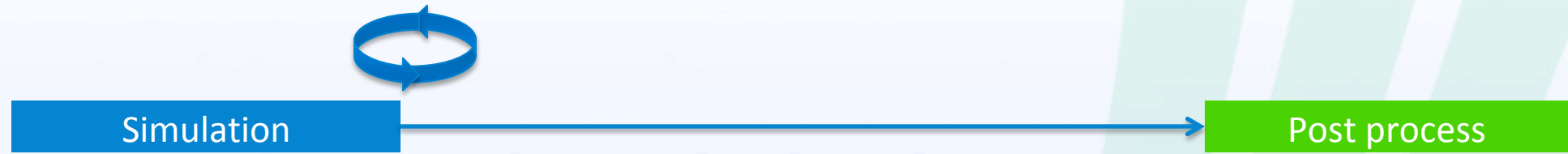
Post process



# Traditional/Post Hoc Analysis

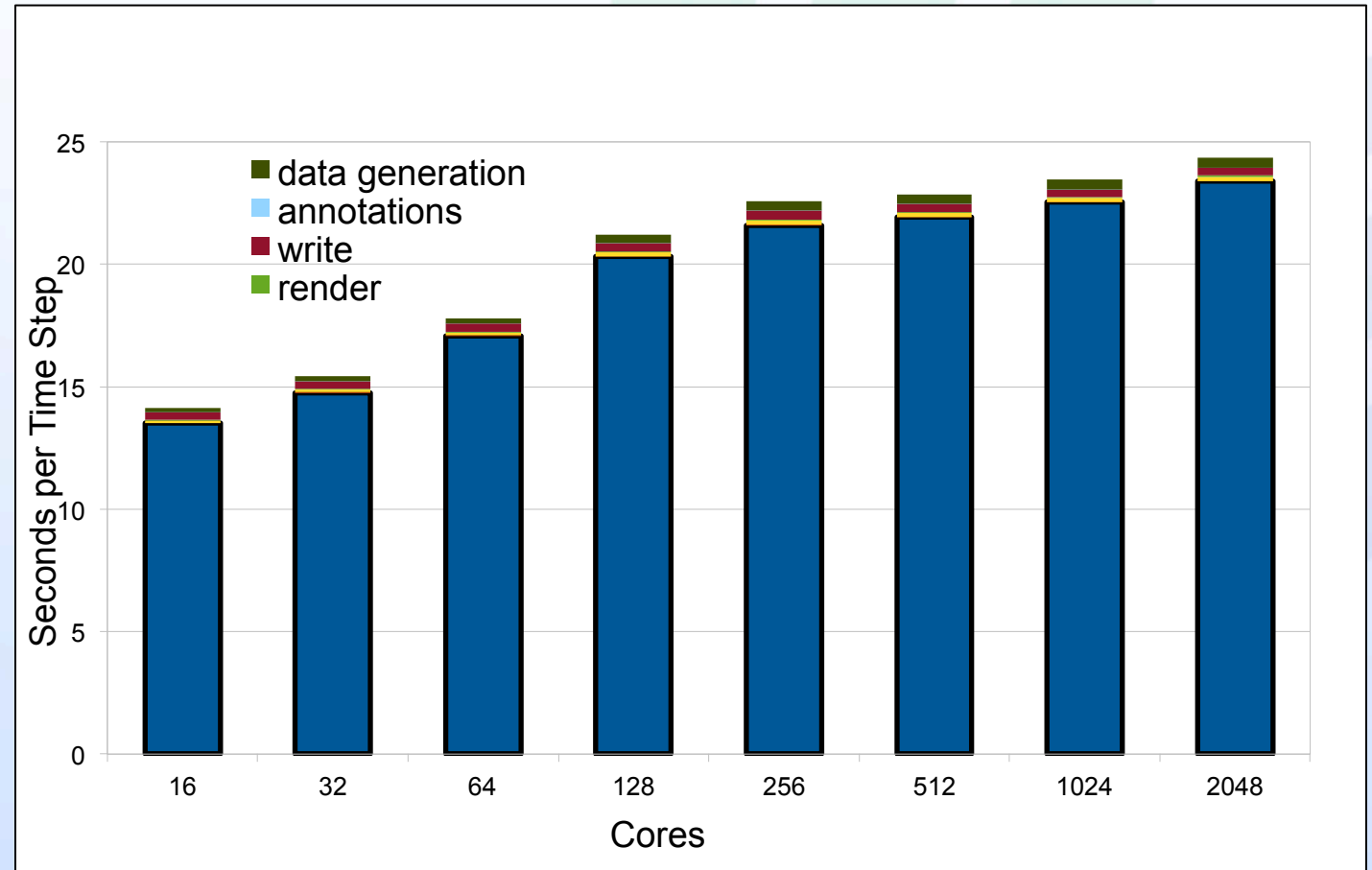


# In Situ Analysis



# Small Run-Time Overhead

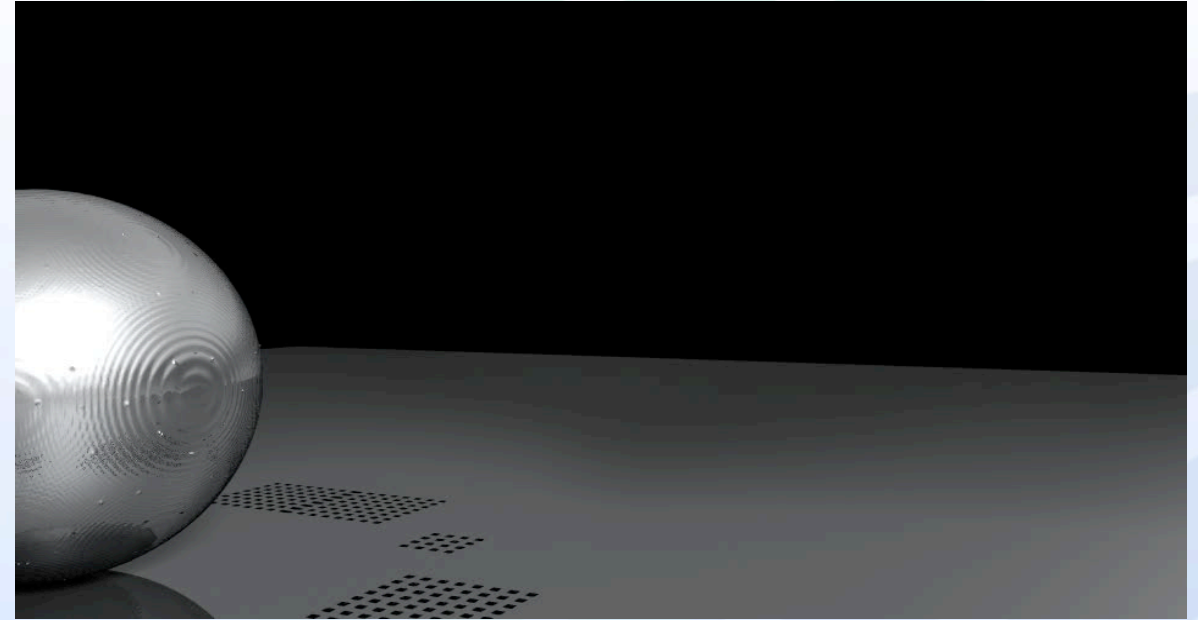
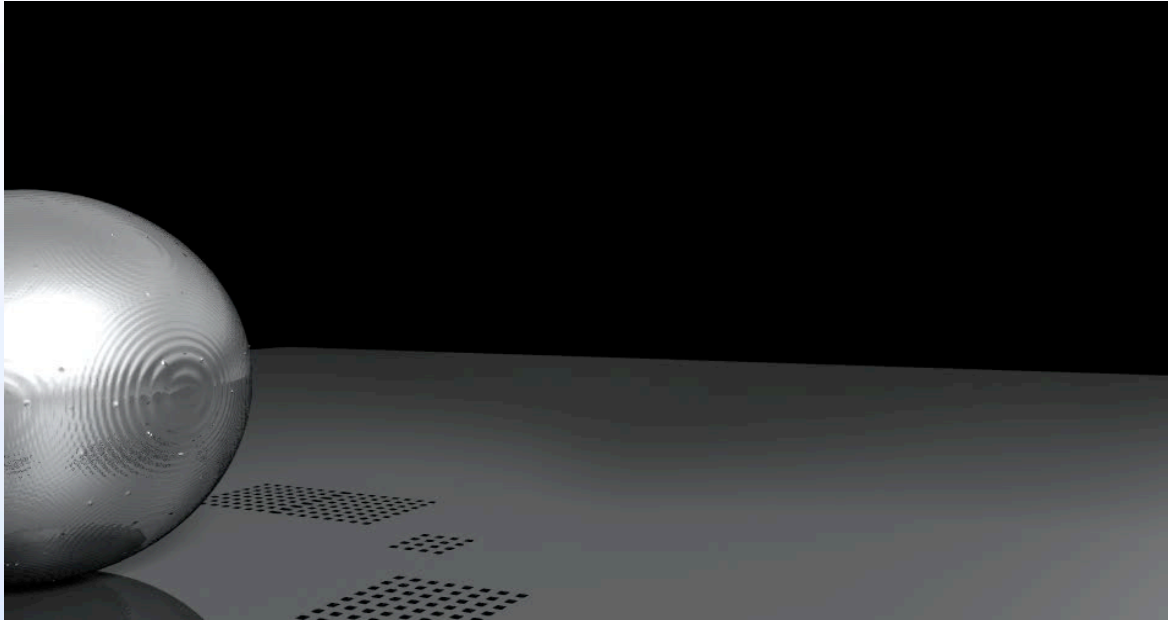
- Small initialization and finalization times
- Scalable analysis and visualization algorithms
- Reduced amount of IO
  - More complex IO patterns



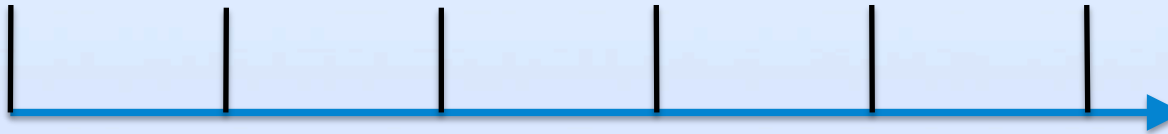
XRAGE (LANL) simulation

<https://datascience.lanl.gov/data/papers/2013-2.pdf>

# Access to More Data



Dump Times



Post-processing

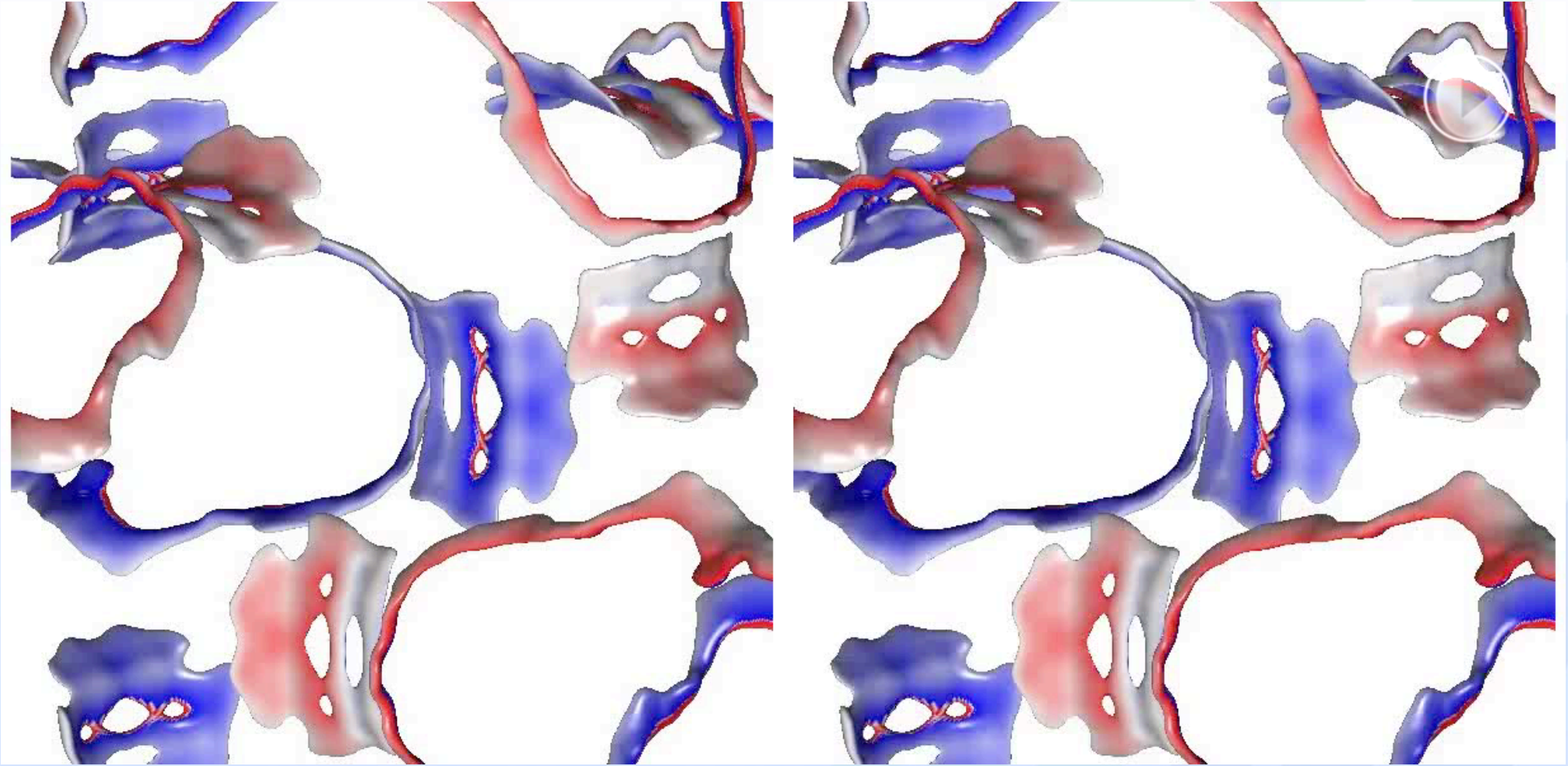


*In situ* processing

CTH (Sandia) simulation with roughly equal data stored at simulation time

*Reflections and shadows added in post-processing for both examples*

# Better Insight



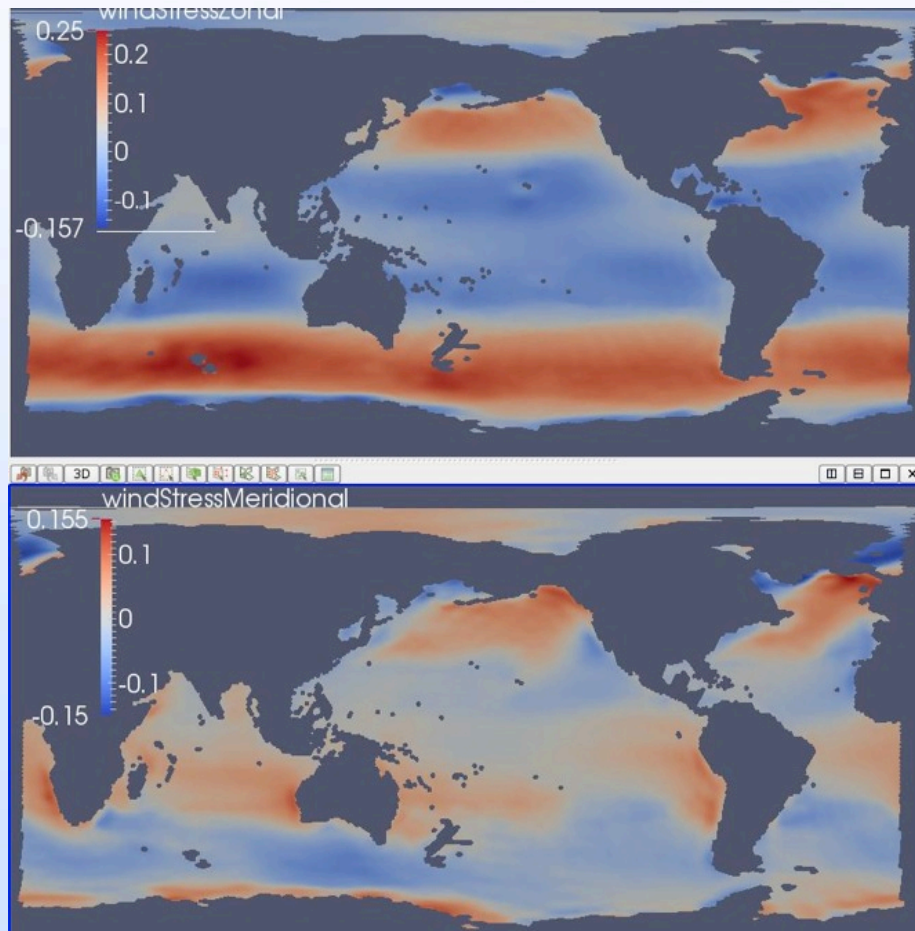
Full dump every 400 time steps versus *in situ* every 25 time steps

Animation courtesy Sean Ziegler (PETTT/Engility)

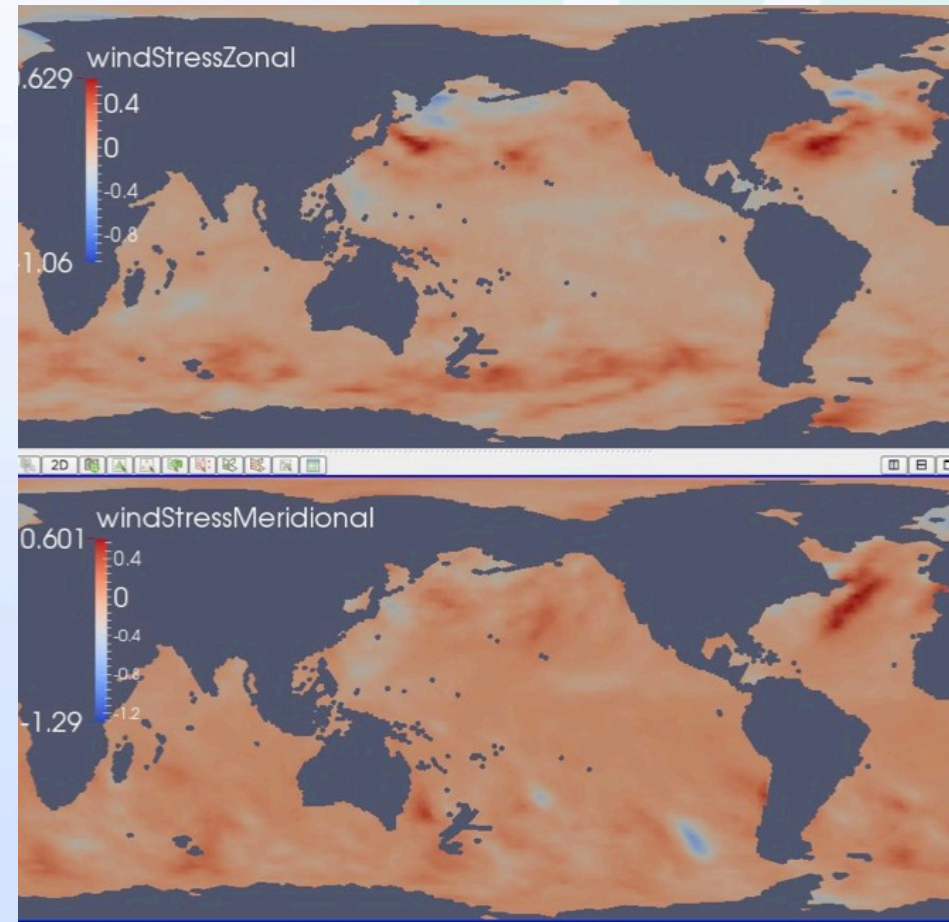


# Quick and Easy Run-Time Checks

Expected wind stress field at the surface of the ocean



Wind stress in new run, quick glance indicates using wrong wind stress

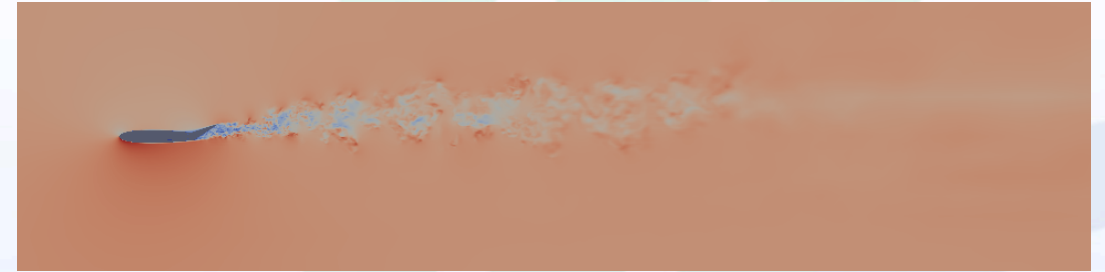


MPAS-O (LANL) simulation

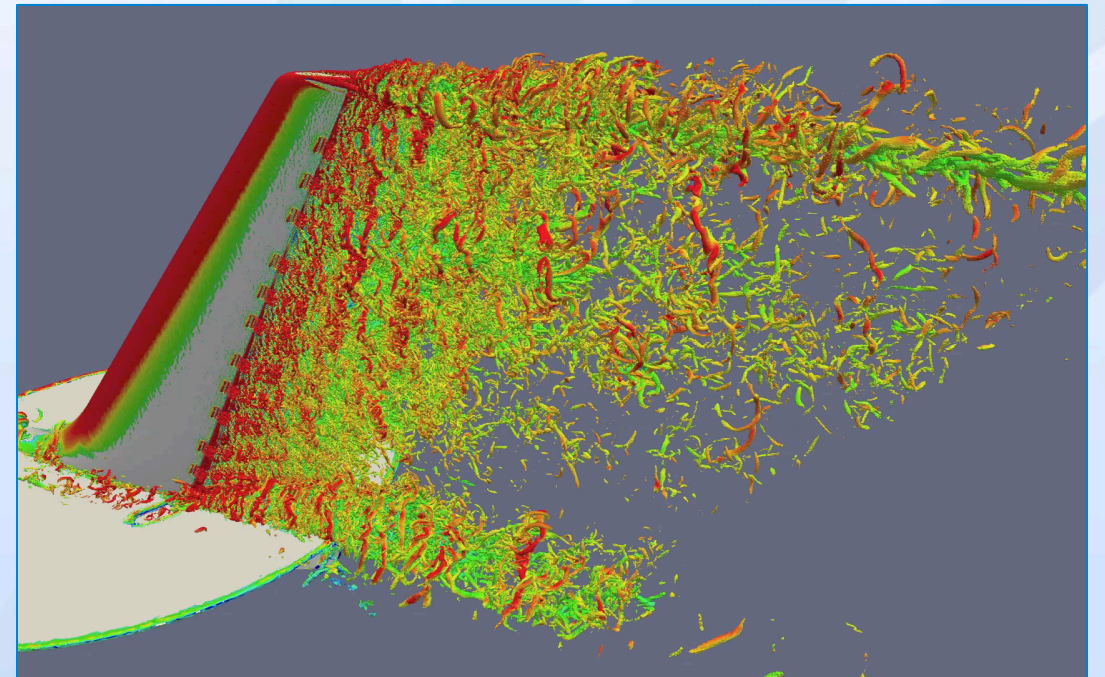
# Going to Exascale

- Highly parallel algorithms (if possible)
- Avoid file IO issues
- Freeze Python
- Process 0 reads Python script and broadcasts to other processes
- Don't generate .pyc files
- Static libraries
- Simulation appropriate levels of parallelism

1M MPI ranks on Mira@ANL (BG/Q)

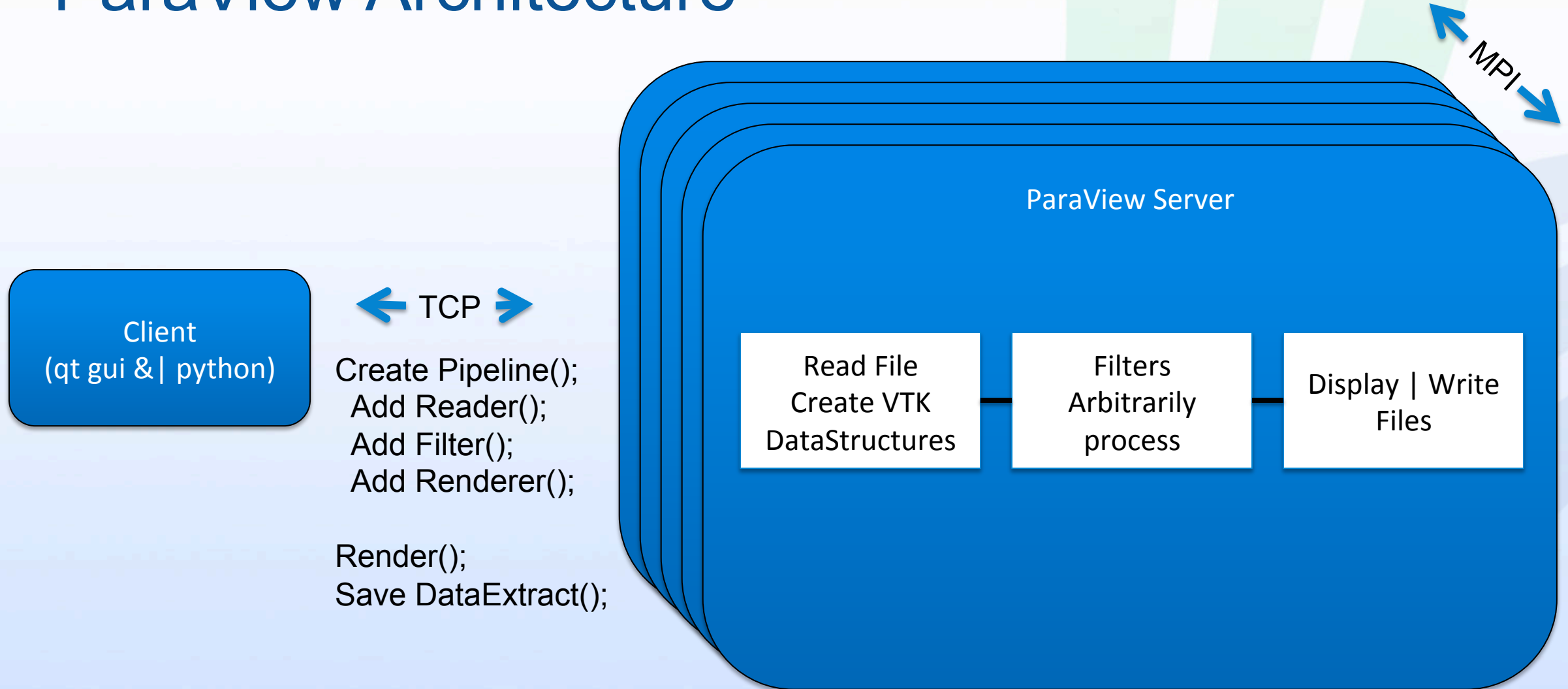


256K MPI ranks on Mira@ANL (BG/Q)



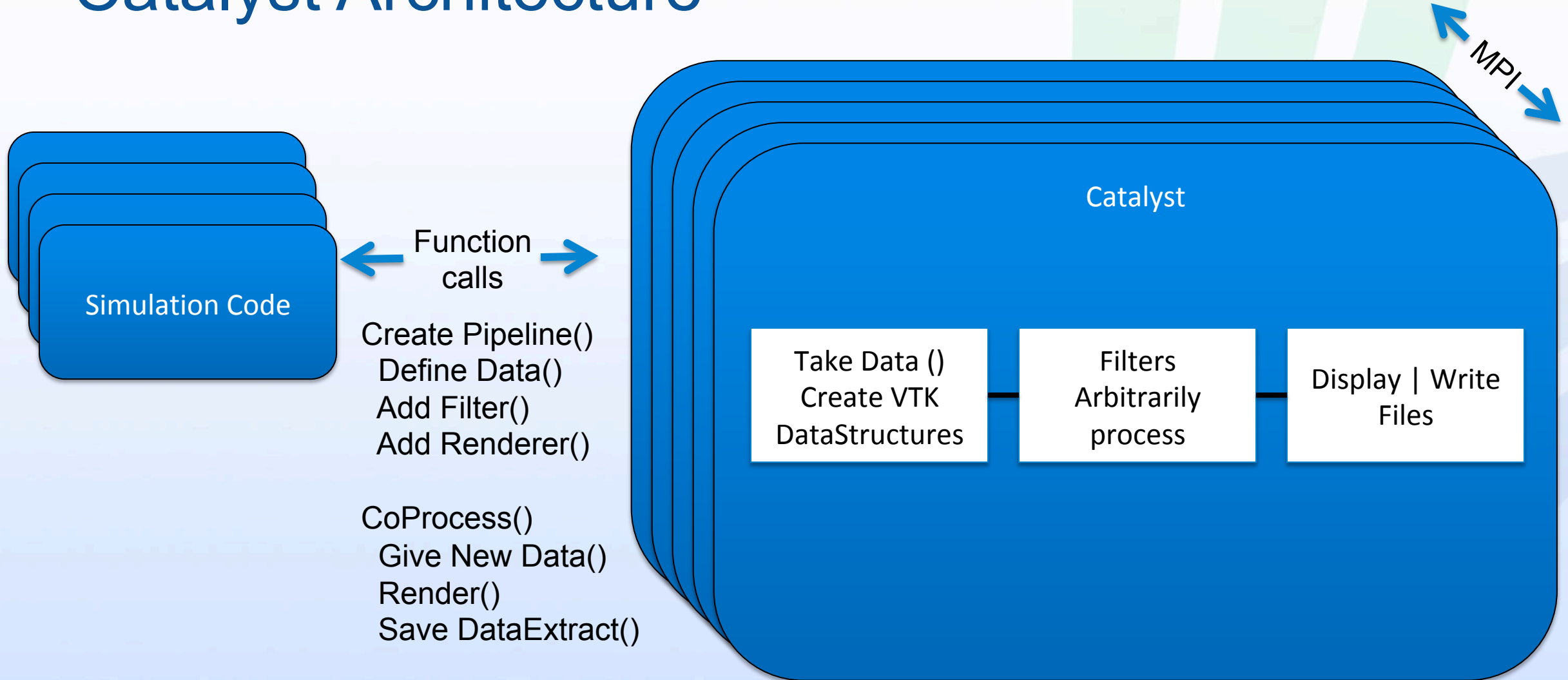
Movie courtesy Michel Rasquin (Cenaero/ UC Boulder)

# ParaView Architecture





# Catalyst Architecture

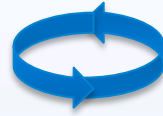


# Motivation Continued



Bespoke -  
Why not just hardcode plot routines?

# Motivation Continued



Simulation

plot

Why not just hardcode plot routines?

Why not code in assembler?

Take advantage of ParaView - an extremely capable, flexible, scalable library with tens of thousands of developer hours behind it.

# Motivation Continued



Why not use an extremely capable, flexible, scalable library with tens of thousands of developer hours behind it?

and millions of lines of code

millions of lines of code?  
memory overhead?  
unfamiliar syntax?

# Since 2009 we've been steering ParaView

- Millions of Lines of Code?
  - VTK 6.0 Modularity : Catalyst Editions (even custom)
  - Boiled down to three entry points to add to simulation
- Memory Overhead?
  - Editions again
  - Zero copy arrays
- Unfamiliar Syntax?
  - Documentation
  - User Level Controls : choose nitty gritty details or high level record/play
    - Can code at VTK level (c++ or Python)
    - Can record python scripts in GUI and run them
    - Can encapsulate as Domain Specific Commands (Sparta) in Input Deck

# All of ParaView in my Sim?!



!@^#

- The entire reason for VTK 6.0
- a.k.a. modularization
  - 19 kits -> 160 modules (== libs)
  - Remove unused code by deleting directories
    - 100MB->5MB
  - Add code by dropping in directories
  - Dependency scripts traverse includes
- At Catalyst level - called “*Editions*”

Base

Base + Essentials

Base + Essentials + Render

Base + Essentials + Render + Python

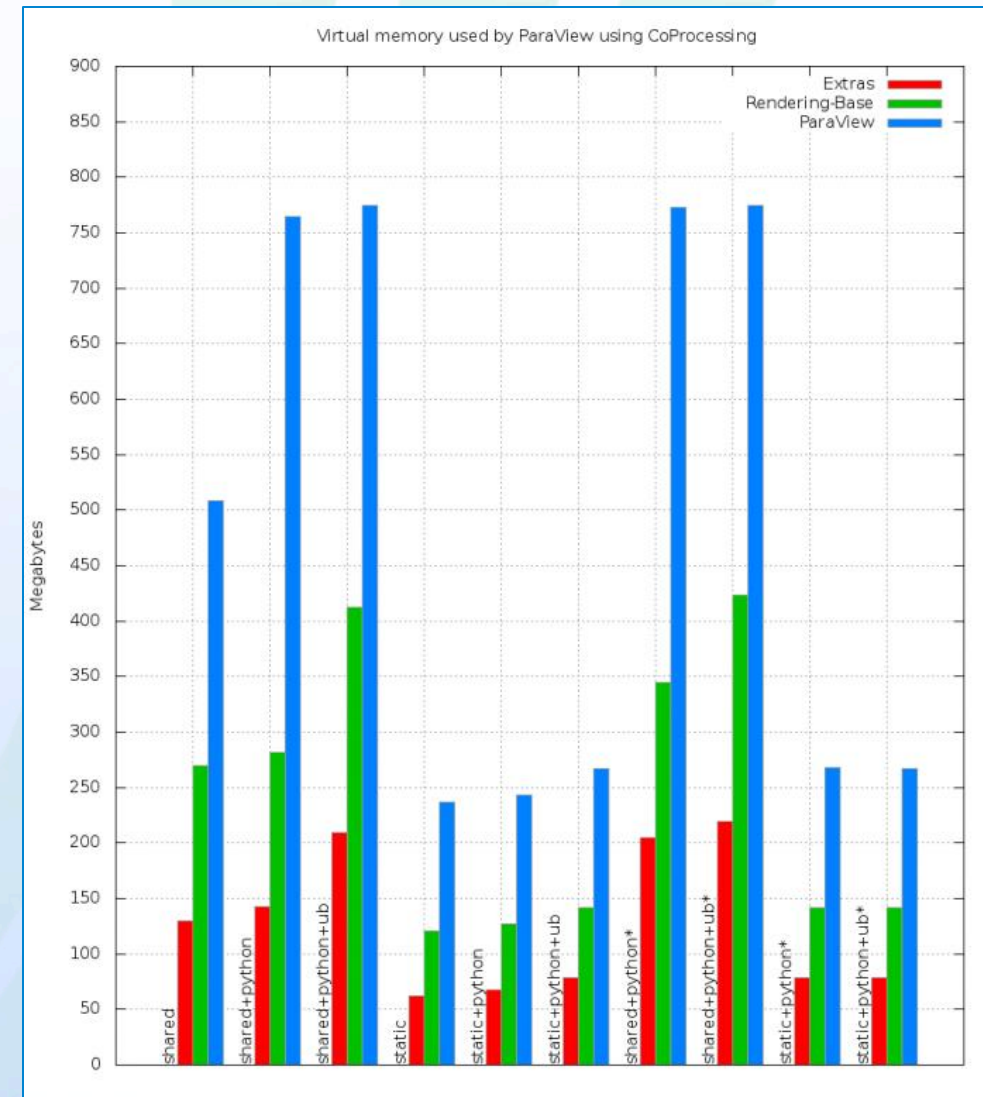
Custom

Maker scripts that build source tree

See 4.1 in Catalyst Users Guide

# Efficient Memory-Wise

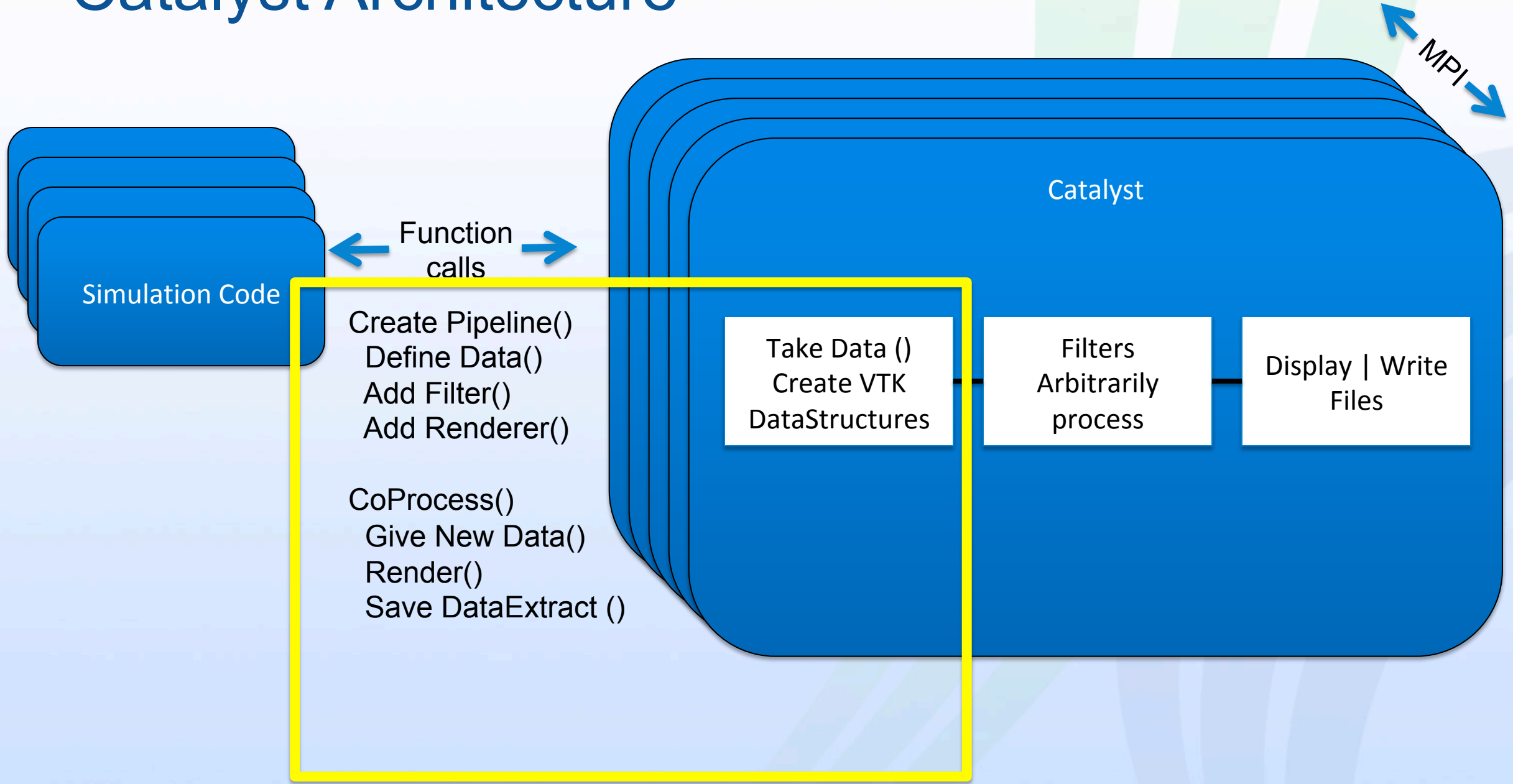
- Try to use simulation data structure memories (read only)
  - Catalyst support structure-of-arrays and array-of-structures memory layouts
  - Pipeline architecture ensures data isn't modified by Catalyst
- Only create objects that are needed when they're needed
- Small library size
  - Catalyst editions – versions without linking to unneeded parts of VTK/ParaView, Python and/or rendering components
- UH3D using Catalyst editions
  - 400 MB extra memory footprint with full ParaView
  - 40 MB extra memory footprint with Catalyst edition with rendering support



<https://blog.kitware.com/paraview-catalyst-editions-what-are-they/>

<https://blog.kitware.com/why-is-paraview-using-all-that-memory/>

# Catalyst Architecture





# How to make an Adaptor

1. Link to Catalyst Library
2. Insert three calls into simulation
  - `vtkCPPProcessor::Initialize()`
  - `vtkCPPProcessor::CoProcess()`
  - `vtkCPPProcessor::Finalize()`
3. Translate Simulation Data to VTK Data Structures
  - `vtkCPInputDataDescription::SetGrid(vtkDataObject *)`
4. Define Pipeline/Results to gather

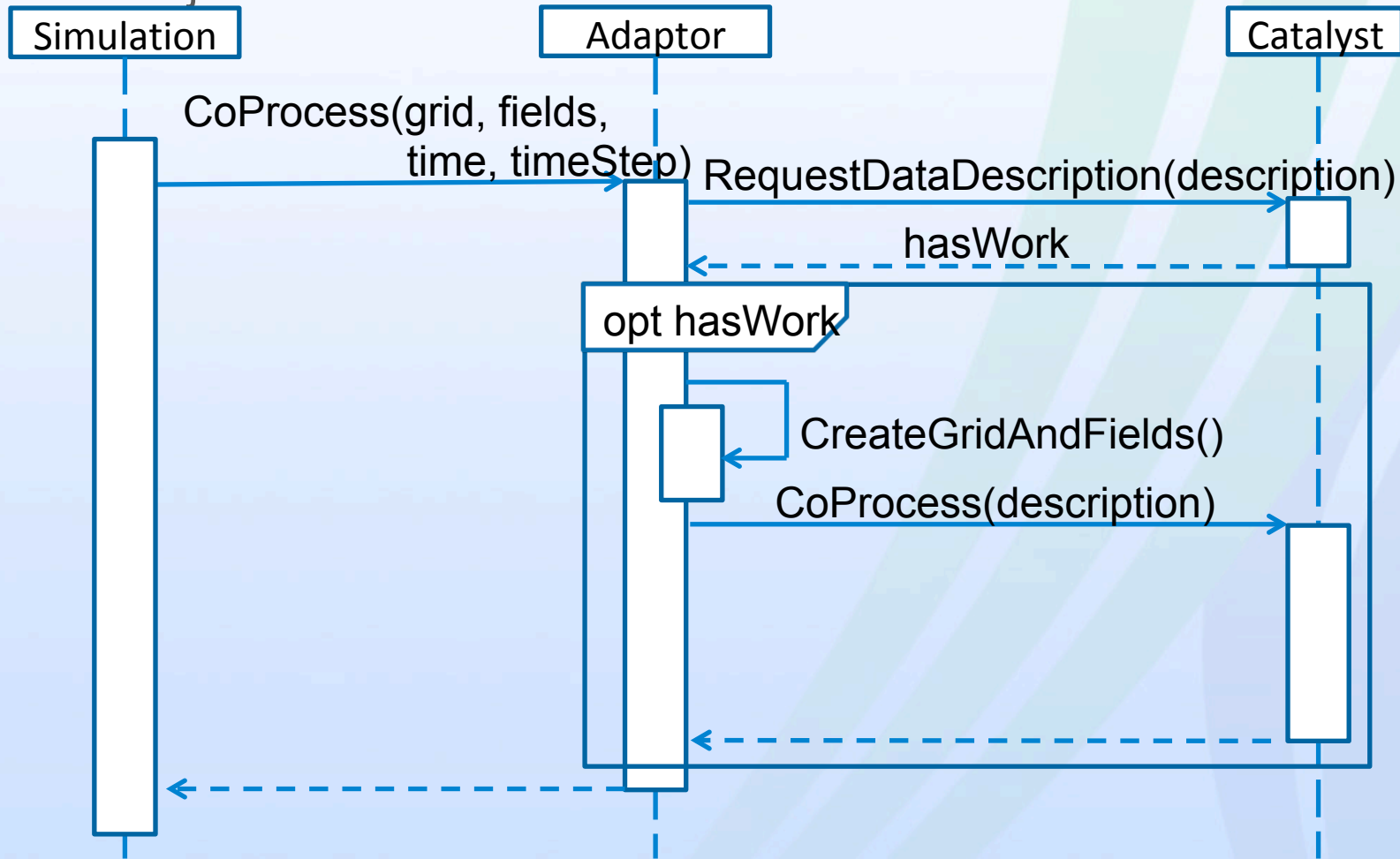
Use generic option and let simulation user do this

# API for the Developer

- `vtkCPDataDescription`
  - A means to transfer information from sim to ParaView (->) and back (<-)
  - -> named map of Grids for which new data is ready
    - “name” : `vtkCPInputDataDescription`
  - -> Time
  - <- Is Data necessary at this time?
- `vtkCPInputDataDescription` (CPIDD)
  - A container for a grid
  - Adaptor is responsible for populating CPIDD's `vtkDataObject`

# Efficient Compute-Wise

- Catalyst called every time step
- Negligible compute time if no output is requested
- Only create VTK objects if there is *in situ* work



# Populating vtkDataObjects

- See Section 3.2 of the Catalyst User Guide v 2.0

- Data Structures

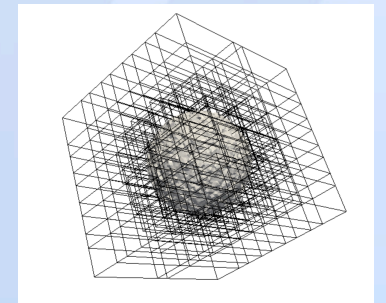
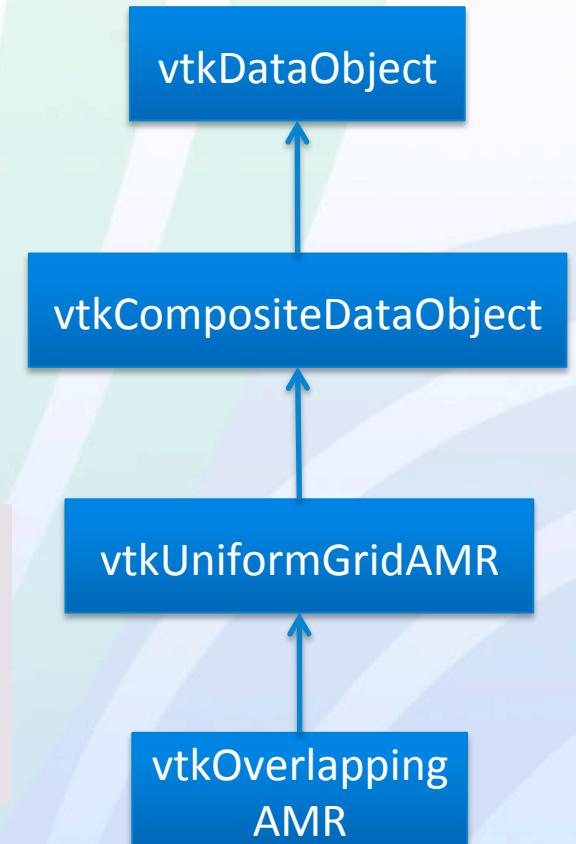
- Geometry, Connectivity, Values

- Arrays

```
vtkFloatArray* arr = vtkFloatArray::New();  
arr->SetName("an array");  
float* values = new float[300];  
arr->SetArray(values, 300, 0, vtkDoubleArray::VTK_DATA_ARRAY_DELETE);  
arr->SetNumberOfComponents(3);
```

- ZeroCopy Arrays/Grids

- <http://www.vtk.org/Wiki/VTK/InSituDataStructures>
  - vtkMappedDataArray (old)
  - vtkGenericDataArray ← vtkSOADDataArray



# API for the Developer continued

- `vtkCPPipeline`
  - a ParaView pipeline with arbitrary contents
  - `vtkCPPythonScriptPipeline`
    - Takes in a python script that defines the pipeline
  - Or derive your own subclass and manually create VTK/PV filters

# API for the Developer continued

- `vtkCPPProcessor` - Manages the whole server
  - Sim runs visualization through `CPPProcessor` calls
  - Has 0 or more `vtkCPDataDescriptions`
  - Has 0 or more `vtkCPPipelines`
  - `Initialize()`
    - Create Pipeline(s)
  - `CoProcess()`
    - Use `CPDataDescriptions` to ask if Pipelines need to run
    - If yes:
      - Populate their `CPIDD`'s with new data
      - Call Update (actually `CoProcess`) on Pipeline
  - `Finalize()`

# Small Code Footprint



- Typically 3 calls between simulation code and adaptor
  - Initialize()
    - MPI communicator (optional)
    - Add analysis scripts
  - CoProcess()
    - Does the work (potentially)
  - Finalize()
- Information provided by solver to adaptor
  - Time, time step, (optional) force output
  - Grids and fields

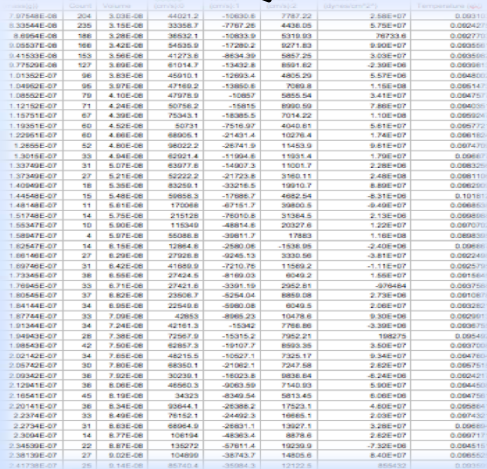


The screenshot displays the OpenFOAM 2.2.0 graphical user interface. The main window shows a 3D visualization of a turbulent flow field, likely a wake flow, with a complex mesh structure. A color bar at the bottom indicates the velocity magnitude in cm/s, ranging from 0 (blue) to 800,000 (red). The GUI includes a top menu bar, a toolbar, and a left sidebar with various tool options.

```
# Create the reader and set the filename.
reader = servermanager.sources.Reader(FileNames=path)
view = servermanager.CreateRenderView()
repr = servermanager.CreateRepresentation(reader, view)
reader.UpdatePipeline()
dataInfo = reader.GetDataInformation()
pDInfo = dataInfo.GetPointDataInformation()
arrayInfo = pDInfo.GetArrayInformation("displacement")
if arrayInfo:
    # get the range for the magnitude of displacement9
    range = arrayInfo.GetComponentRange(-1)
    lut = servermanager.rendering.PVLookupTable()
    lut.RGBPoints = [range[0], 0.0, 0.0, 1.0,
                     range[1], 1.0, 0.0, 0.0]
    lut.VectorMode = "Magnitude"
    repr.LookupTable = lut
    repr.ColorArrayName = "displacement9"
    repr.ColorAttributeType = "POINT_DATA"
```

Simulation

A diagram illustrating a simulation process. At the top, the word "Simulation" is written in a black serif font. Below it, a large, light-red rounded rectangle with a thin red border contains the word "Catalyst" in a black serif font. An arrow points from the left edge of the slide into the left side of the "Catalyst" box. At the bottom center, a black line extends downwards and then branches out into three separate lines, suggesting a continuation or expansion of the process.



The top plot displays two variables: Temperature (Temp, black line) and Pressure (Pres, orange line) against distance. The x-axis ranges from 0.00 to 2.00. The y-axis for Temp ranges from 300.00 to 1000.00, and for Pres from 0.02 to 0.03. Temp decreases from approximately 920 to 550, while Pres increases from approximately 0.018 to a peak of 0.028 at distance 1.00, then decreases to 0.02 at distance 2.00.

The bottom plot is a bar chart showing the distribution of a variable across different distance intervals. The x-axis labels are 324.15, 448.15, 572.15, 696.15, and 820.15. The y-axis ranges from 0.00 to 5000.00. The bars show a sharp peak at 324.15 (approx. 4600), followed by a significant drop at 448.15 (approx. 1100), and then much lower values for the remaining intervals.

A 3D visualization of a crack in a concrete slab. The crack is represented by a red, elongated, and irregular shape. A blue mesh is overlaid on the crack, showing its internal structure. The crack is surrounded by a light blue, textured material. The entire scene is set against a dark blue background.

## Rendered Images



# Creating a Catalyst Python Script

- Open a representative data set in ParaView GUI
- Setup pipeline
- Define the outputs
  - Load in the Catalyst Script Generator Plugin
  - Add writers to Pipeline - Data Extracts
  - (within Export Script Dialog) Export Views - Rendered Images
- Save Script

# Create Pipeline 1 - load plugin

Only if ParaView < 5.5

View Sources Filters **Tools** Catalyst Macros Help

Create Custom Filter...  
Add Camera Link...  
Manage Custom Filters...  
Manage Links...  
**Manage Plugins...**  
Record Test...  
Play Test...  
Lock View Size  
Lock View Size Custom...  
Timer Log  
Output Window  
Python Shell  
Start Trace

Local plugins are automatically searched for in /media/ssddrive/BUILDS/ParaView/debug/bin/plugins.

Remote Plugins

Name	Property
------	----------

Local Plugins

Name	Property
NonOrthogonalSource	Not Loaded
SciberQuestToolKit	Not Loaded
QuadView	Not Loaded
pvNektarReader	Not Loaded
GMVReader	Not Loaded
AnalyzeNifTIO	Not Loaded
H5PartReader	Not Loaded
EyeDomeLightingView	Not Loaded
MobileRemoteControl	Not Loaded
UncertaintyRendering	Not Loaded
PacMan	Not Loaded
CatalystScriptGeneratorPlugin	Not Loaded
Version	
Location	/media/ssddrive/BUILDS/P
Required Plugins	
Status	Not Loaded
Auto Load	<input checked="" type="checkbox"/>
Moments	Not Loaded
StreamingParticles	Not Loaded
ArrowGlyph	Not Loaded
SurfaceLIC	Not Loaded
SierraPlotTools	Not Loaded
RGBZView	Loaded
SLACTools	Not Loaded
PointSprite_Plugin	Not Loaded
vtkPVInitializerPlugin	Loaded

Load New ... Load Selected Remove

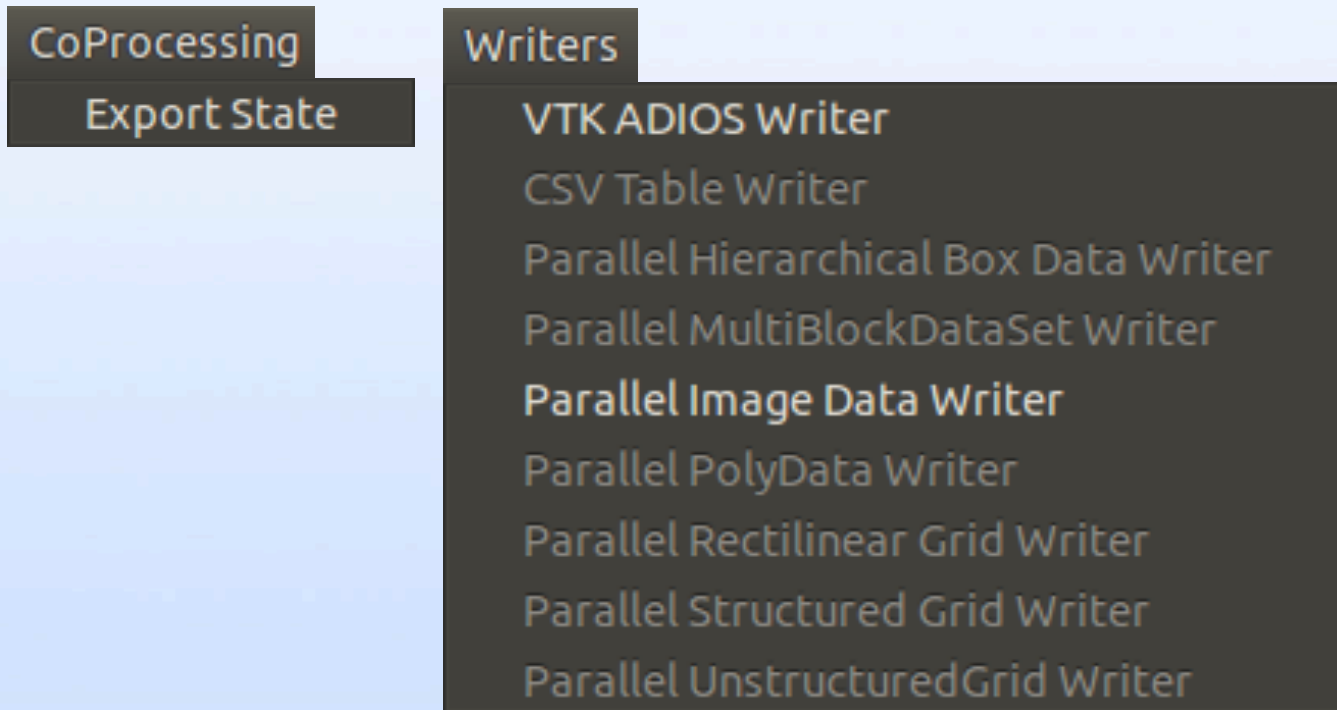
Load New ... Load Selected Remove

Close

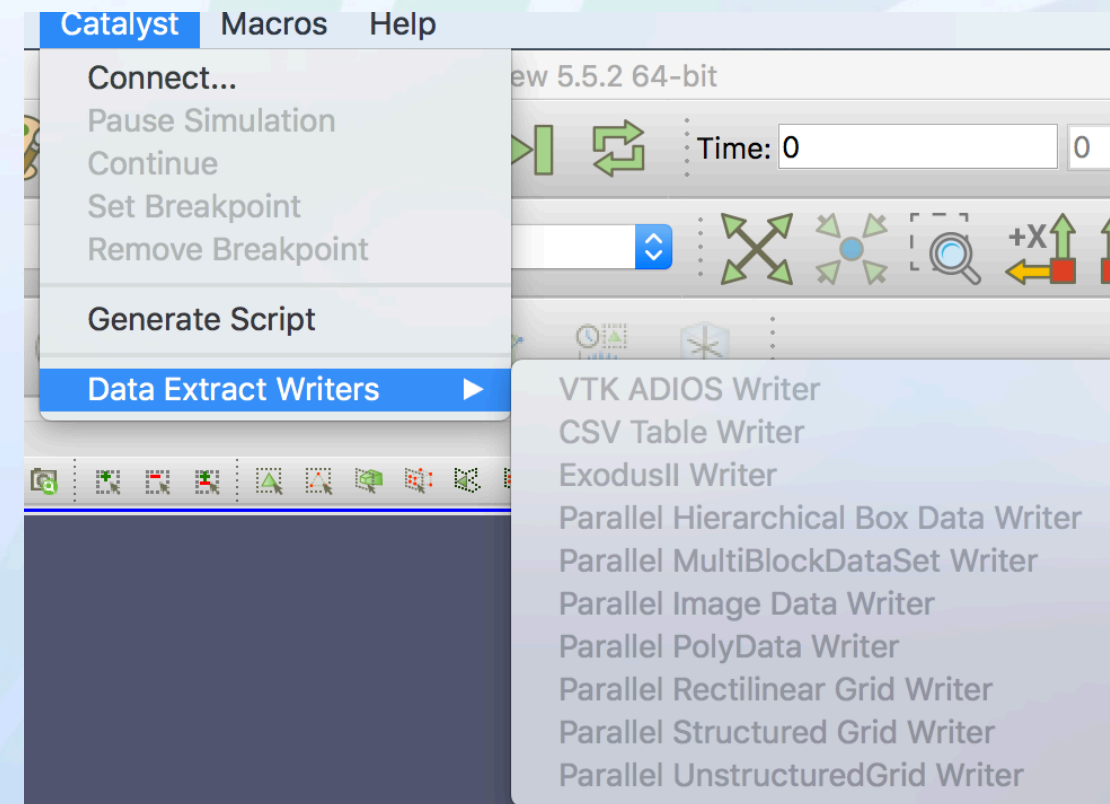
# Plugin adds two new menus

Two new menu items specific to creating Catalyst Python scripts

< PV 5.5




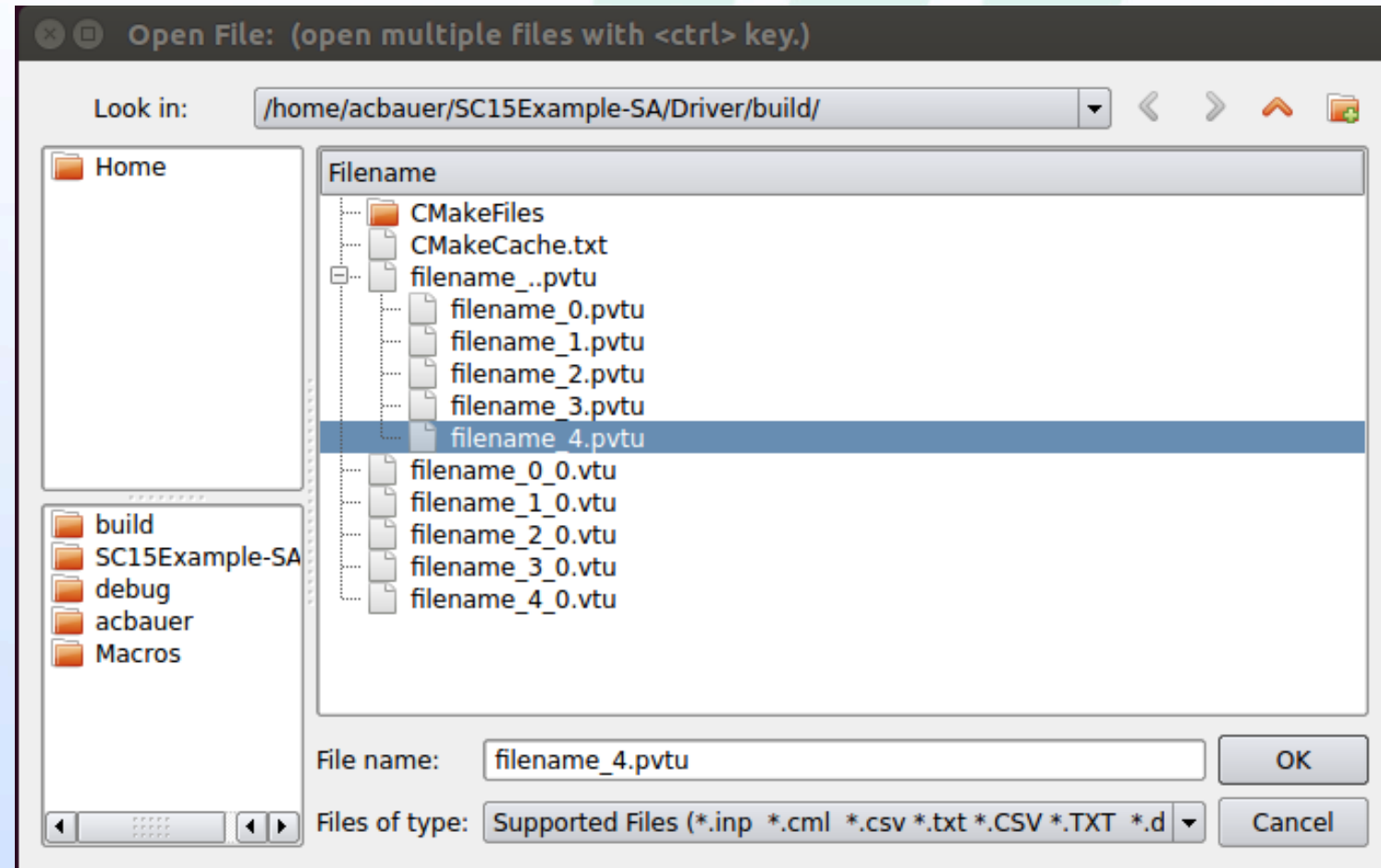
> PV 5.5



# Create Pipeline

Load “representative” data set

- **File→Open...**
  - Shortcut 
  - Choose /home/catalystuser/ filename\_4.pvtu



# Define Writers

- Only valid writers available in **Writers** menu
- Parameters:
  - **File Name** – %t gets replaced with time step
  - **Write Frequency**

VTK ADIOS Writer

CSV Table Writer

Parallel Hierarchical Box Data Writer

Parallel MultiBlockDataSet Writer

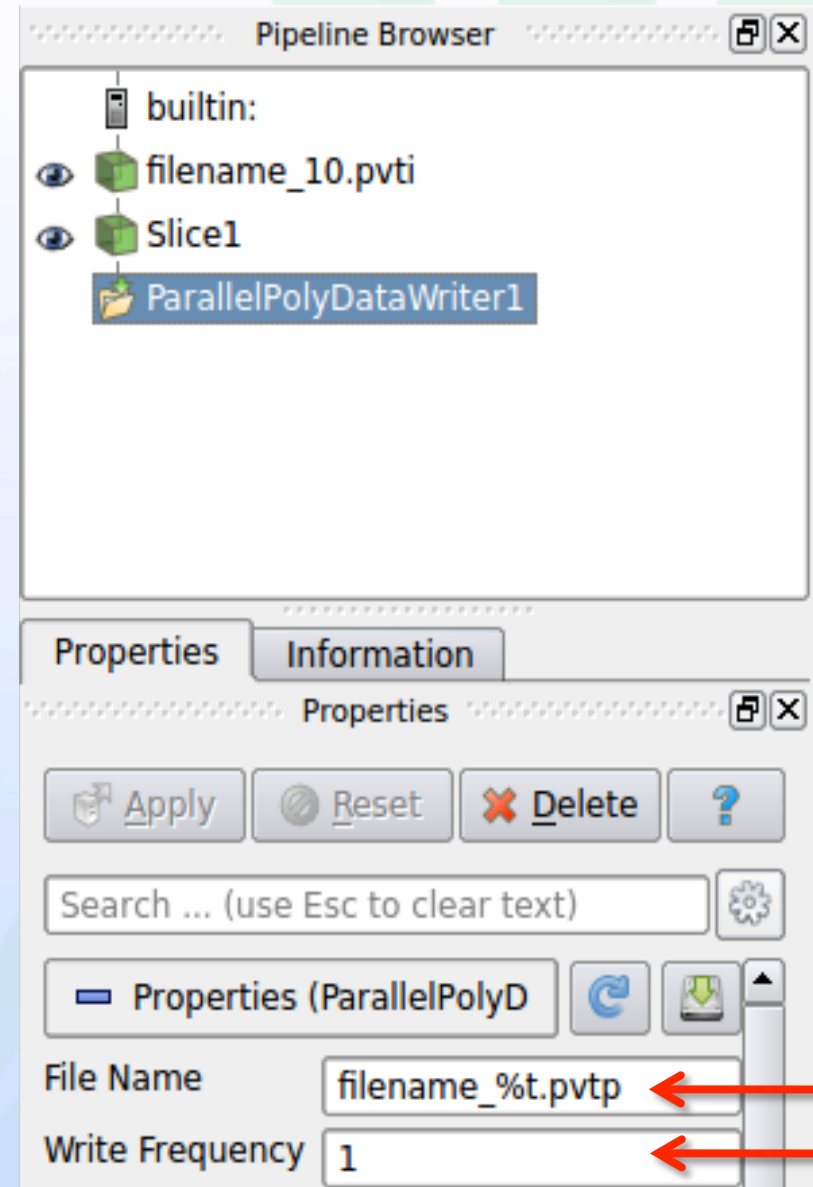
**Parallel Image Data Writer**

Parallel PolyData Writer

Parallel Rectilinear Grid Writer

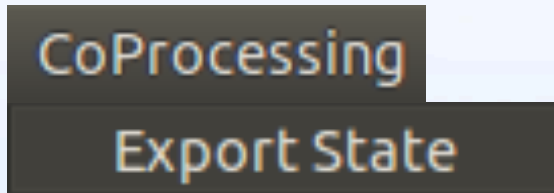
Parallel Structured Grid Writer

Parallel UnstructuredGrid Writer

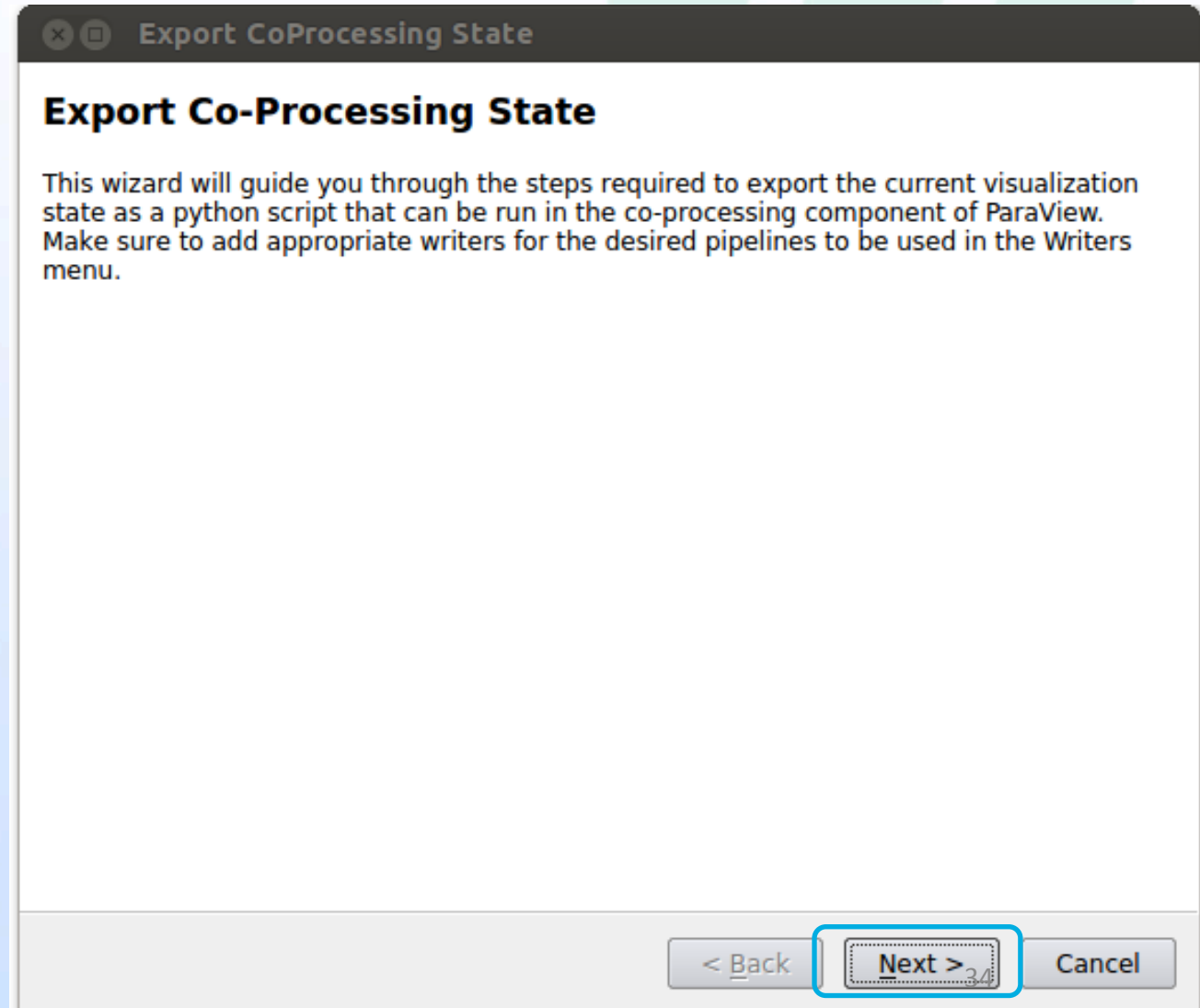
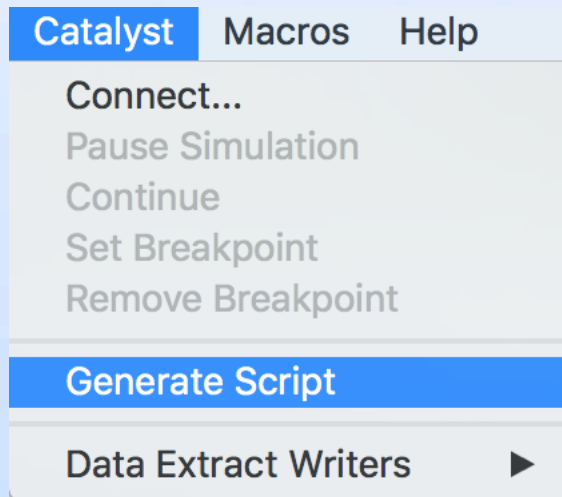


# Export the Pipeline (Script)

< PV 5.5



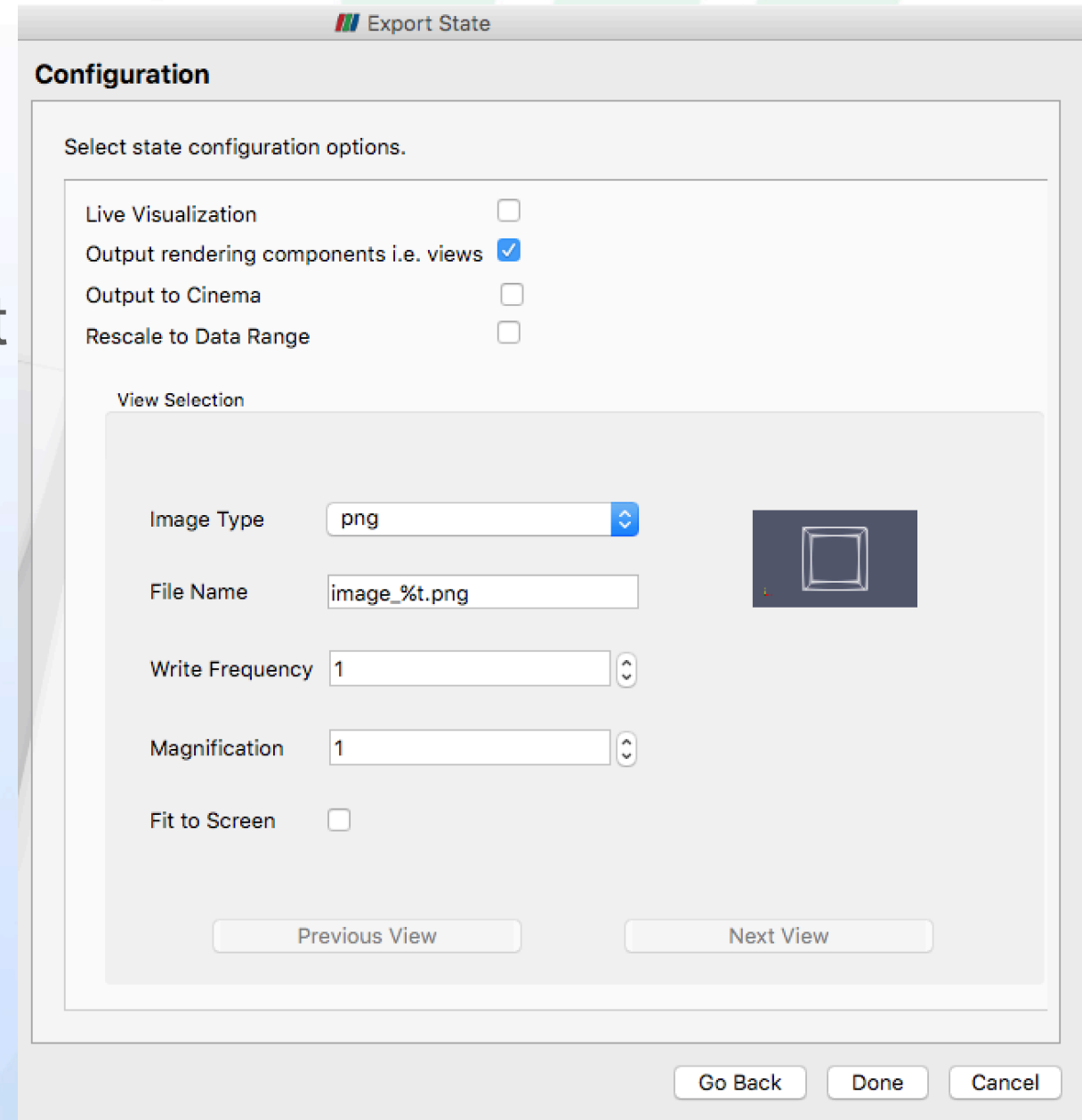
> PV 5.5



# Configure Renders

- Click “Output rendering components i.e. views”
- Choose which Windows to export  
Previous View, Next View
- Set Render Frequency

In PV  $\geq 5.6$  will UI will be streamlined:



The screenshot shows a 'Configuration' dialog box titled 'Export State'. It contains a section 'Select state configuration options.' with four checkboxes: 'Live Visualization' (unchecked), 'Output rendering components i.e. views' (checked), 'Output to Cinema' (unchecked), and 'Rescale to Data Range' (unchecked). Below this is a 'View Selection' section with a preview window showing a wireframe cube. The settings in this section are: 'Image Type' set to 'png', 'File Name' set to 'image\_%t.png', 'Write Frequency' set to '1', 'Magnification' set to '1', and 'Fit to Screen' (unchecked). At the bottom of the 'View Selection' section are two buttons: 'Previous View' and 'Next View'. At the very bottom of the dialog are three buttons: 'Go Back', 'Done', and 'Cancel'.

Export State

### Configuration

Select state configuration options.

- Live Visualization ☐
- Output rendering components i.e. views ☒
- Output to Cinema ☐
- Rescale to Data Range ☐

View Selection

Image Type: png

File Name: image\_%t.png

Write Frequency: 1

Magnification: 1

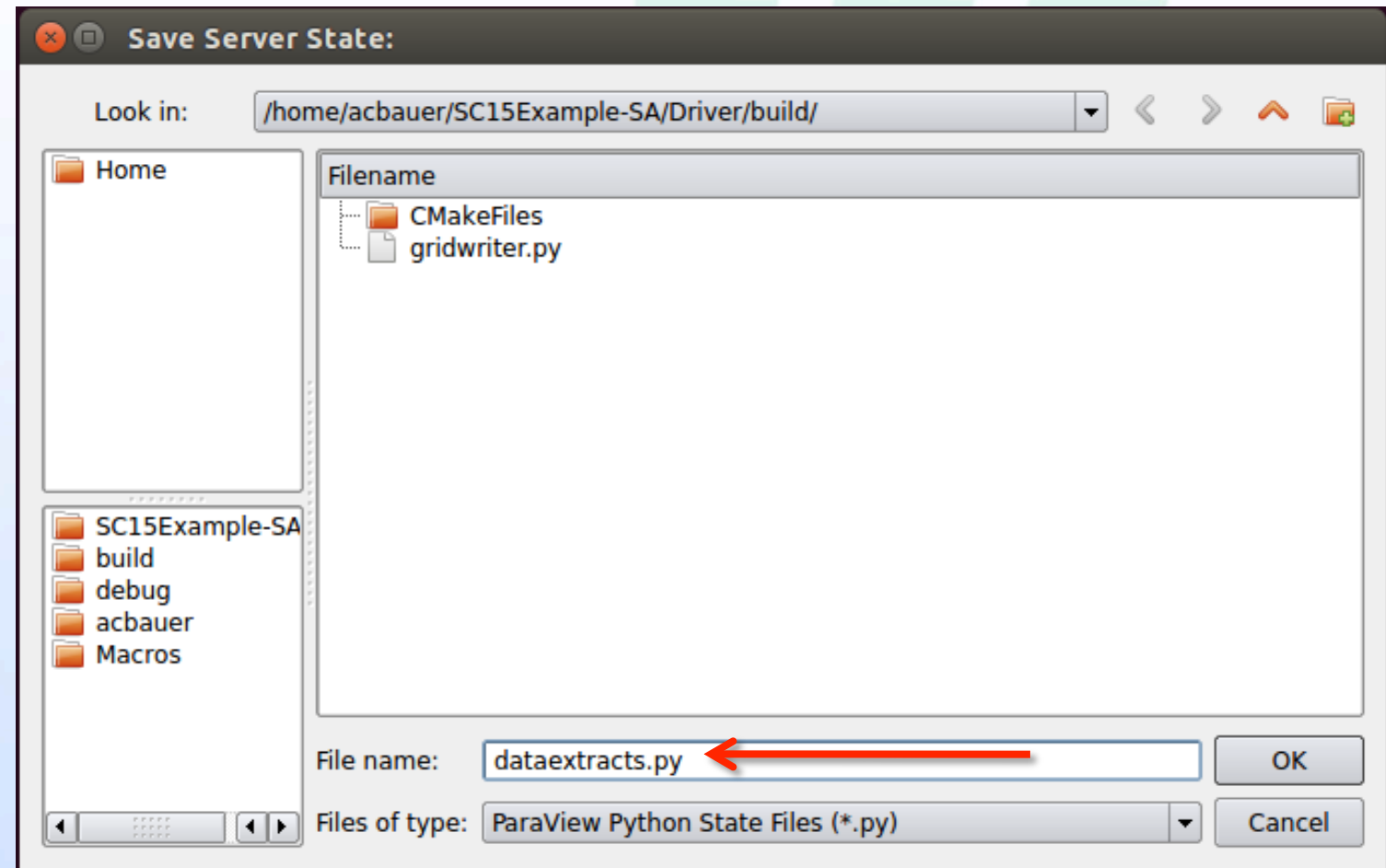
Fit to Screen ☐

Previous View Next View

Go Back Done Cancel

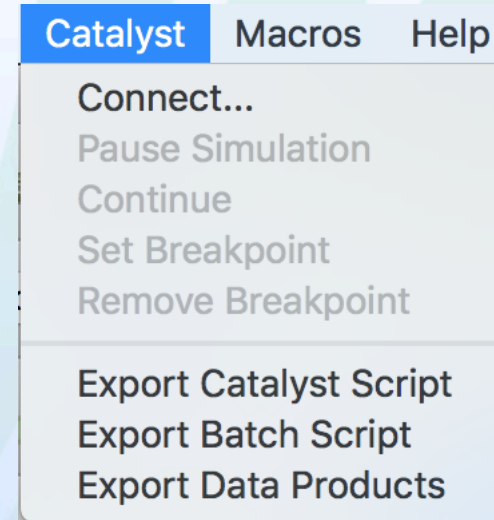
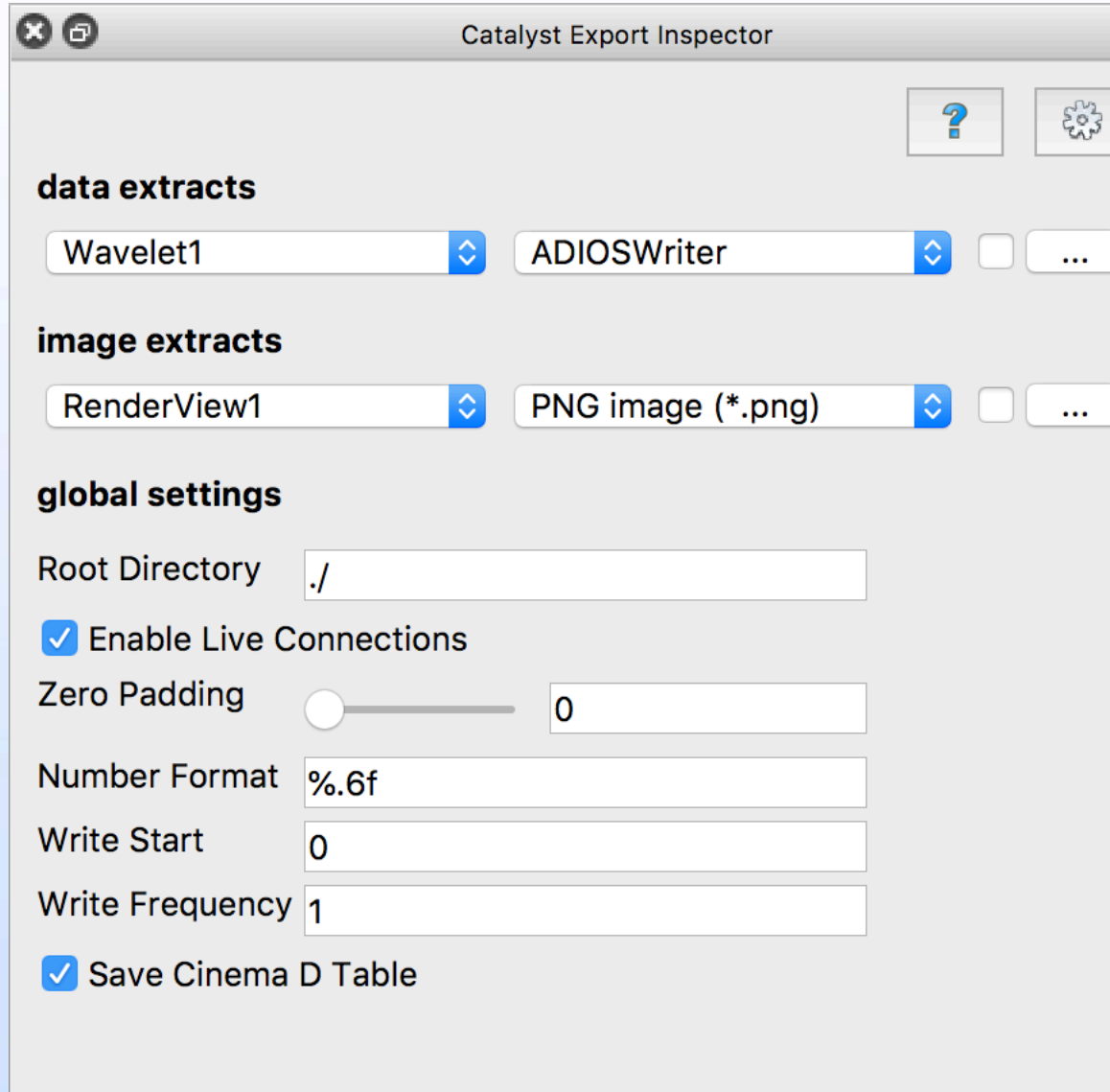
# Save Script

- Click on **Finish**
- Save scripts as dataextracts.py





# LANL's ParaView 5.6 refactoring



# About the Catalyst Script

- Generated script will look something like this
- Pass script (or scripts) as an argument to simulation run.
  - Sim will run, periodically produce extracts and images
- Need to change vis?
  - Regenerate script and rerun
  - (or just edit it)

```
try: paraview.simple
except: from paraview.simple import *

cp_writers = []

def RequestDataDescription(datadescription):
    "Callback to populate the request for current timestep"
    timestep = datadescription.GetTimeStep()

    input_name = 'input'
    if (timestep % 1 == 0):
        datadescription.GetInputDescriptionByName(input_name).AllFieldsOn()
        datadescription.GetInputDescriptionByName(input_name).GenerateMeshOn()
    else:
        datadescription.GetInputDescriptionByName(input_name).AllFieldsOff()
        datadescription.GetInputDescriptionByName(input_name).GenerateMeshOff()

def DoCoProcessing(datadescription):
    "Callback to do co-processing for current timestep"
    global cp_writers
    cp_writers = []
    timestep = datadescription.GetTimeStep()

    input = CreateProducer( datadescription, "input" )

    ParallelMultiBlockDataSetWriter1 = CreateWriter( XMLMultiBlockDataSetWriter, "filename_%.vtm", 1 )

    for writer in cp_writers:
        if timestep % writer.cpFrequency == 0:
            writer.FileName = writer.cpFileName.replace("%t", str(timestep))
            writer.UpdatePipeline()

    if timestep % 1 == 0:
        renderviews = servermanager.GetRenderViews()
        imagefilename = ""
        for view in range(len(renderviews)):
            fname = imagefilename.replace("%v", str(view))
            fname = fname.replace("%t", str(timestep))
            WriteImage(fname, renderviews[view])

    # explicitly delete the proxies -- we do it this way to avoid problems with prototypes
    tobedeleted = GetProxiesToDelete()
    while len(tobedeleted) > 0:
        Delete(tobedeleted[0])
        tobedeleted = GetProxiesToDelete()

def GetProxiesToDelete():
    iter = servermanager.vtkSMPProxyIterator()
    iter.Begin()
    tobedeleted = []
    while not iter.IsAtEnd():
        if iter.GetGroup().find("prototypes") != -1:
            iter.Next()
            continue
        proxy = servermanager._getPyProxy(iter.GetProxy())
        proxygroup = iter.GetGroup()
        iter.Next()
```

# Some other ways to define pipeline

Lower level :

By hand at VTK/PV level

- Derive directly from vtkCPPipeline  
Instead of vtkCPPythonScriptPipeline
- No change to DataDescription and  
CoProcessor use model
- Create VTK or PV pipeline manually

Higher Level:

By hand in simulation input deck

CATALYST BLOCK

---

```
1 begin results output
2   database type = catalyst
3
4   begin catalyst
5     #save jpg images instead of png's
6     image format = jpg
7
8     #make them HD (720p) resolution
9     image size = 1280 720
10  end catalyst
11 end results output
```

---

**Listing 5.** Imageset Command Example

---

```
1 begin results output
2   database type = catalyst
3
4   begin catalyst
5     #slice plane through the origin normal to the X axis
6     slice = center 0 0 0 normal 1 0 0
7   end catalyst
8 end results output
```

---

**Listing 6.** Slice Shortcut Command Example

# Honorable Mention : Live

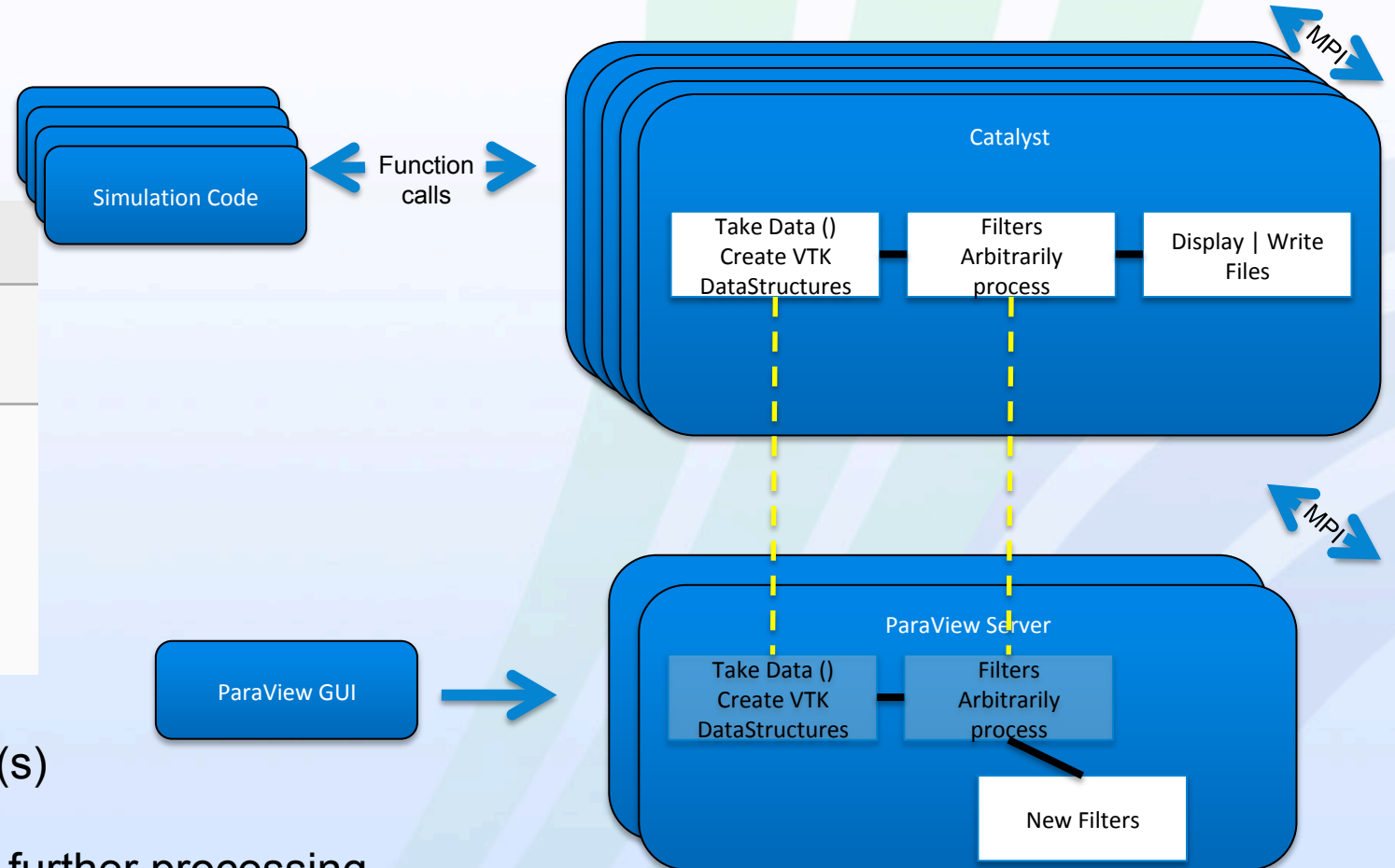
- Live

## Configuration

Select state configuration options.

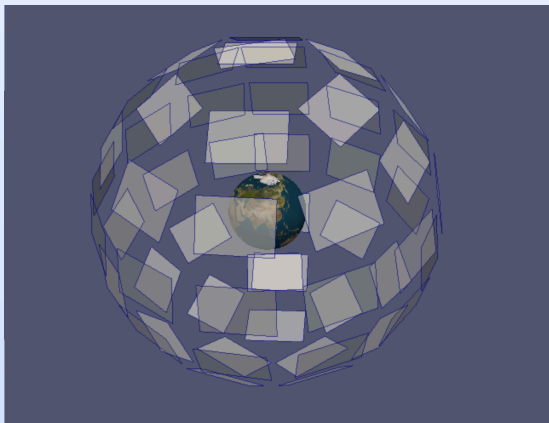
- |  |                                     |
|--|-------------------------------------|
| Live Visualization                     | <input checked="" type="checkbox"/> |
| Output rendering components i.e. views | <input type="checkbox"/>            |
| Output to Cinema                       | <input type="checkbox"/>            |

- Get references to simulation's pipeline(s)
- See data change as it evolves
- Optionally transfer designated data for further processing
- Optionally control settings of filters
- Optionally set breakpoints in simulation



# Honorable Mention

- Cinema -
  - Image based parameter exploration
  - Tell sim what vis parameters you want to inspect
  - In situ save image for all Combinations
  - <http://cinemascience.org>
  - <http://cinemaviewer.org>



## Configuration

Select state configuration options.

- Live Visualization ☐
- Output rendering components i.e. views ☒
- Output to Cinema ☒
- Rescale to Data Range ☐

## View Selection

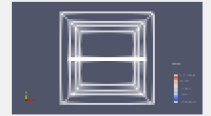
Image Type

File Name

Write Frequency

Magnification

Fit to Screen ☐



## Cinema Options

- Composite ☒
- Disable Float ☐
- Try Direct Float ☒
- Camera Type

Phi  Theta  Roll

Track Object

## Cinema Track Selection

- builtin:
- Wavelet1
- Slice1

Arrays

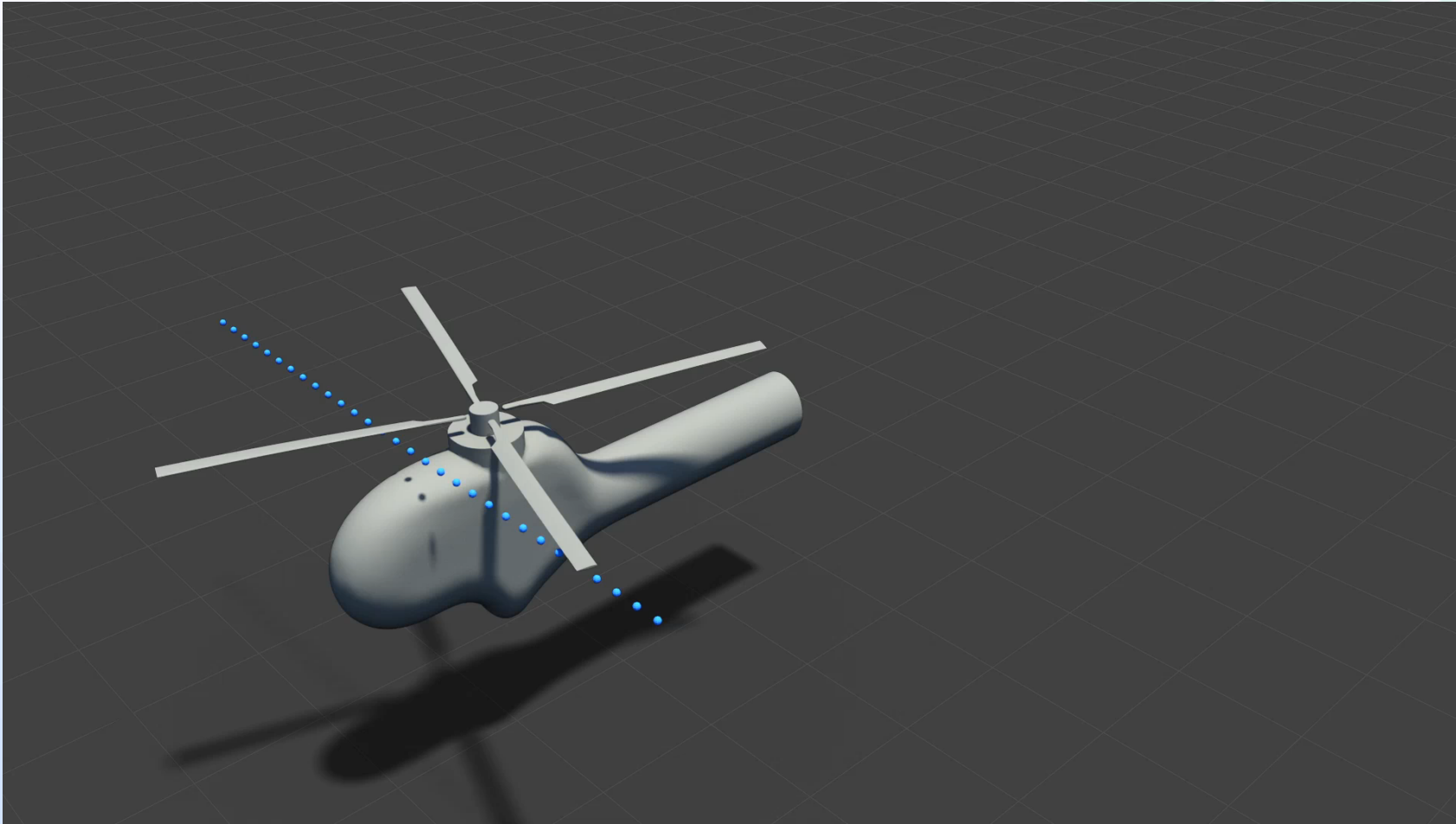
## Slice1

Value Range: [-17.3205, 17.3205]

1	-17.3205	<input type="button" value="+"/>
2	-13.471499999999999	<input type="button" value="-"/>
3	0.6224000000000000	<input type="button" value="X"/>

# Thank You!

Further information: [andy.bauer@kitware.com](mailto:andy.bauer@kitware.com)



Movie courtesy Mike Stephens (DAAC)