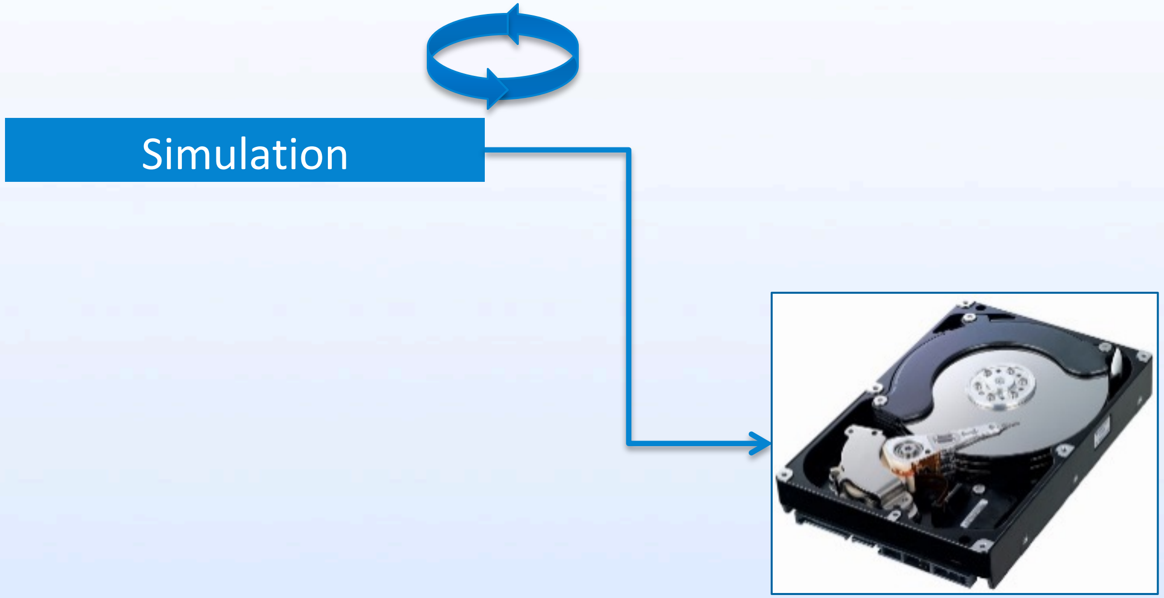
In Situ ParaView - Catalyst Deep Dive

Dave DeMarle

IXPUG SDVis Workshop

May 2018

Traditional/Post Hoc Analysis



Traditional/Post Hoc Analysis

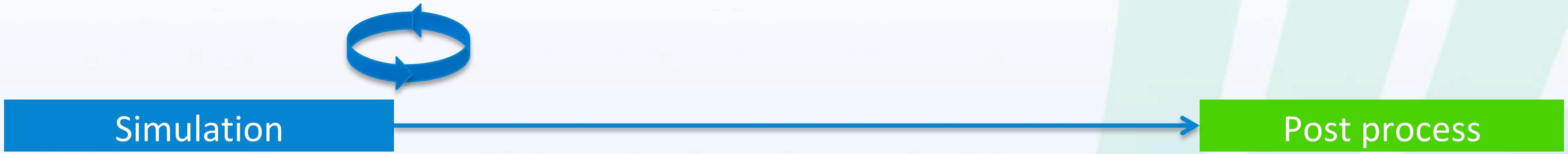
- sometime later…

Traditional/Post Hoc Analysis

Post process

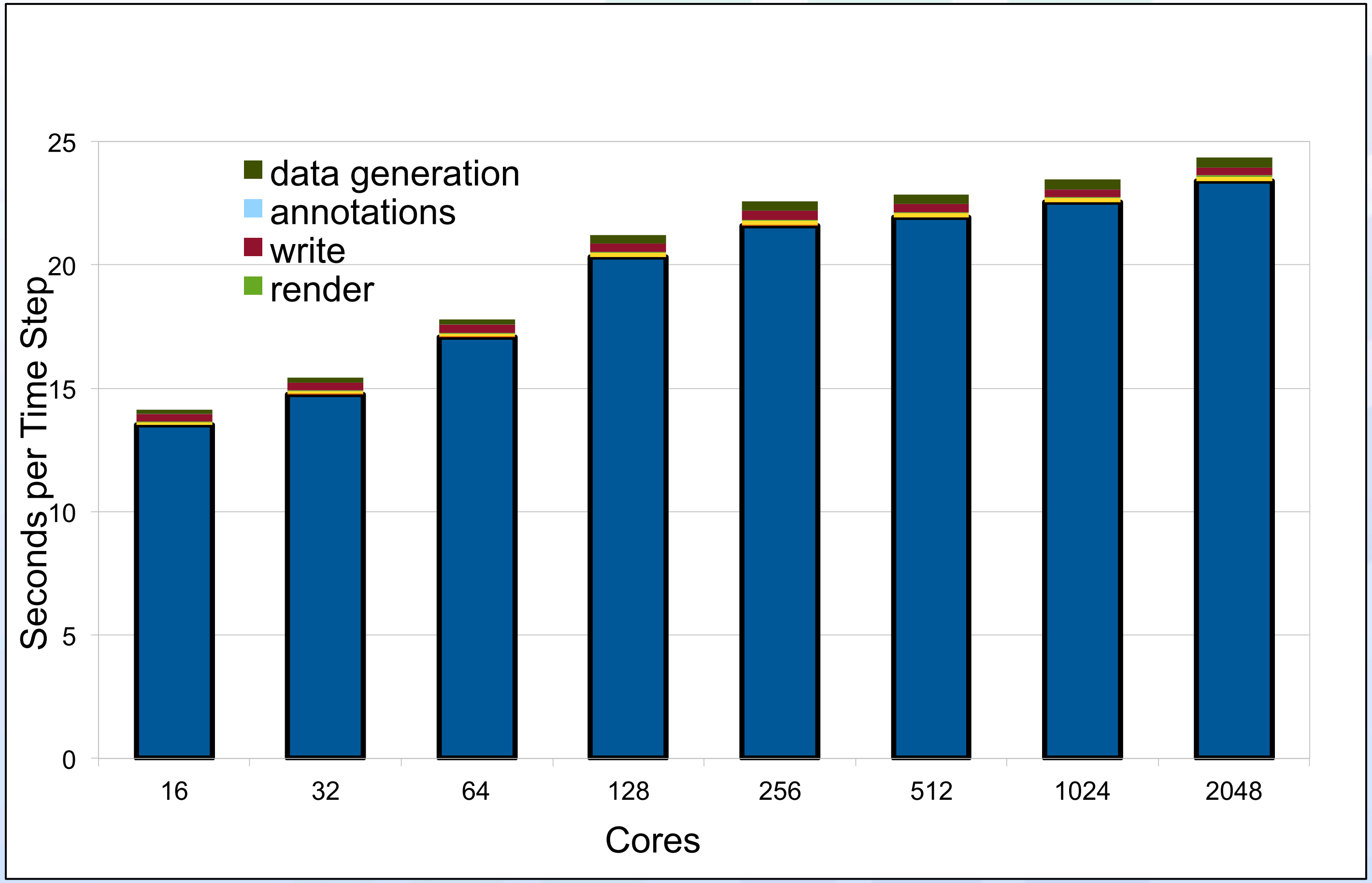
Traditional/Post Hoc Analysis

In Situ Analysis



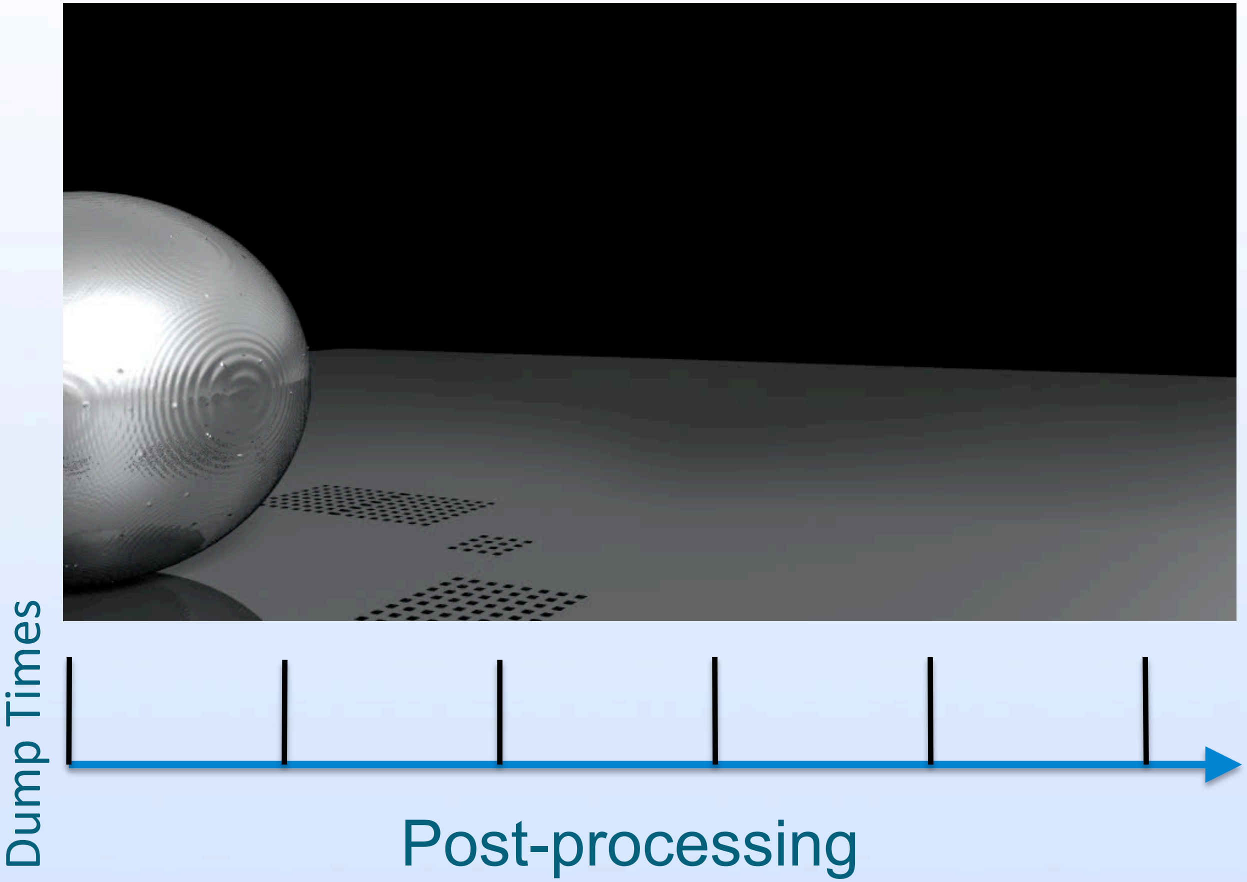
Small Run-Time Overhead

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

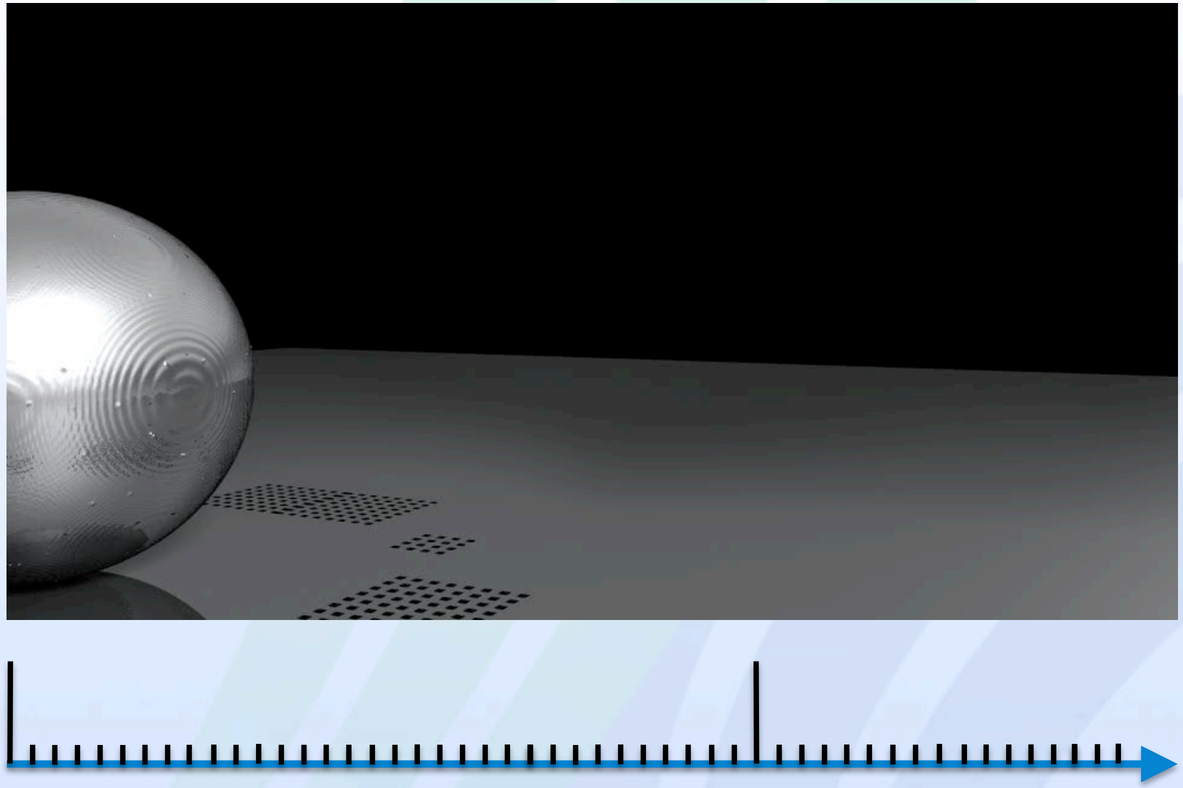


XRAGE (LANL) simulation
https://datascience.lanl.gov/data/papers/2013-2.pdf

Access to More Data



Post-processing

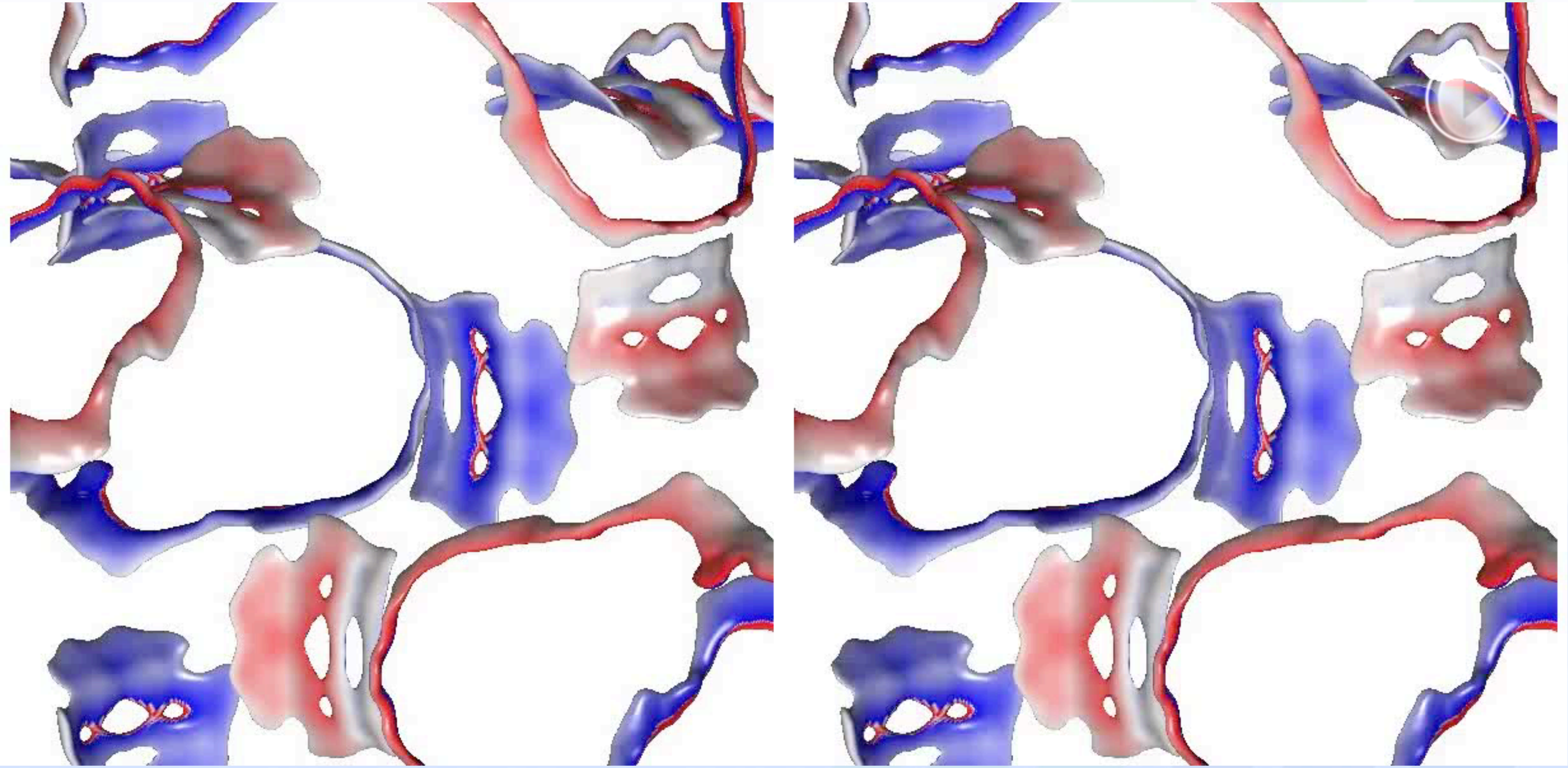


In situ processing

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

Reflections and shadows added in post-processing for both examples

Better Insight

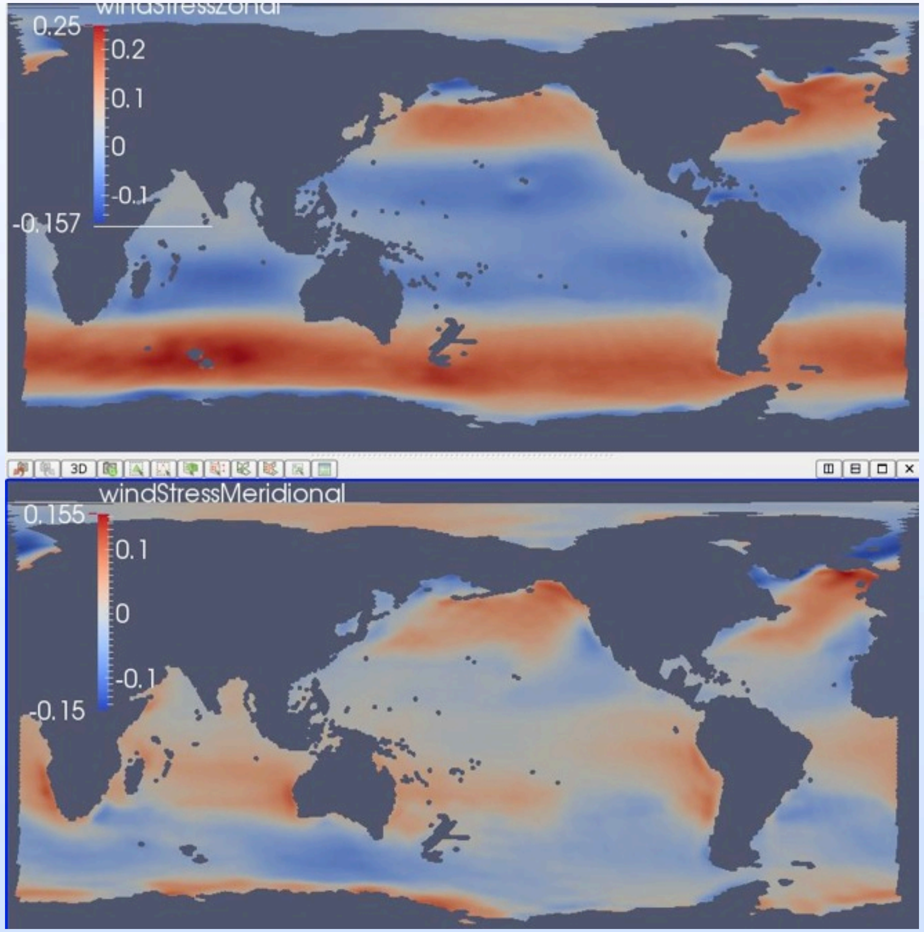


Full dump every 400 time steps versus *in situ* every 25 time steps

Animation courtesy Sean Ziegeler (PETTT/Engility)

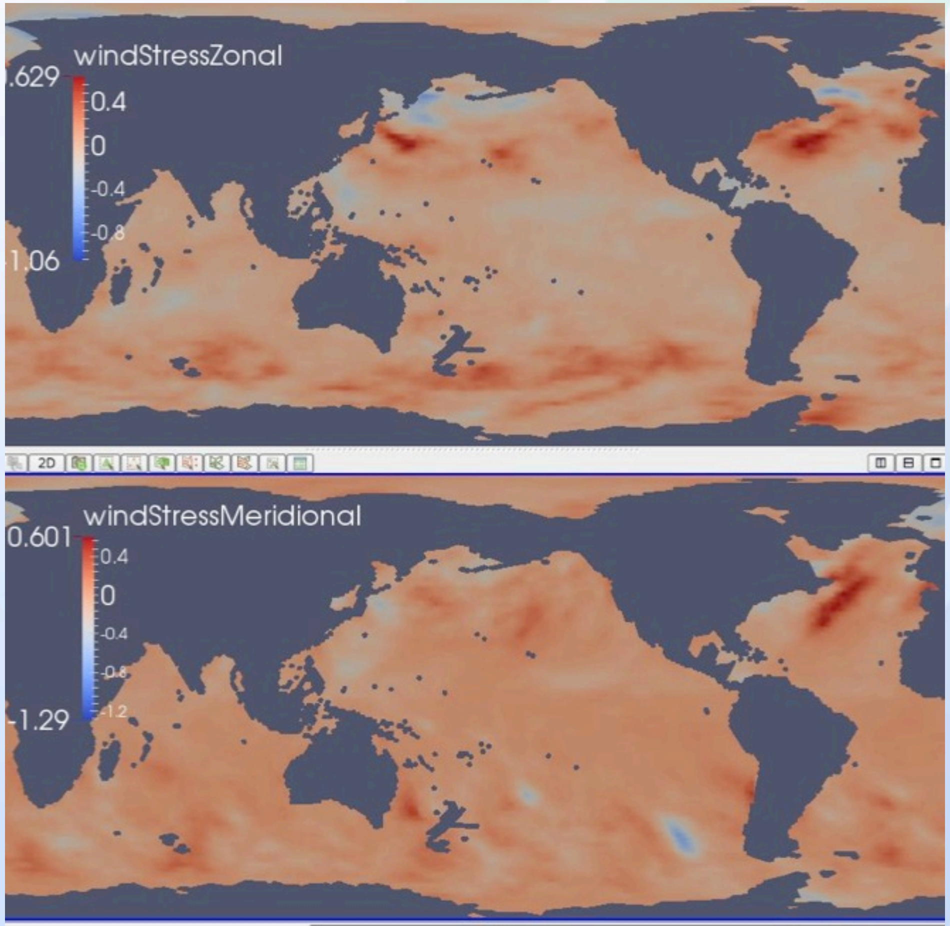
Quick and Easy Run-Time Checks

Expected wind stress field at the surface of the ocean



MPAS-O (LANL) simulation

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

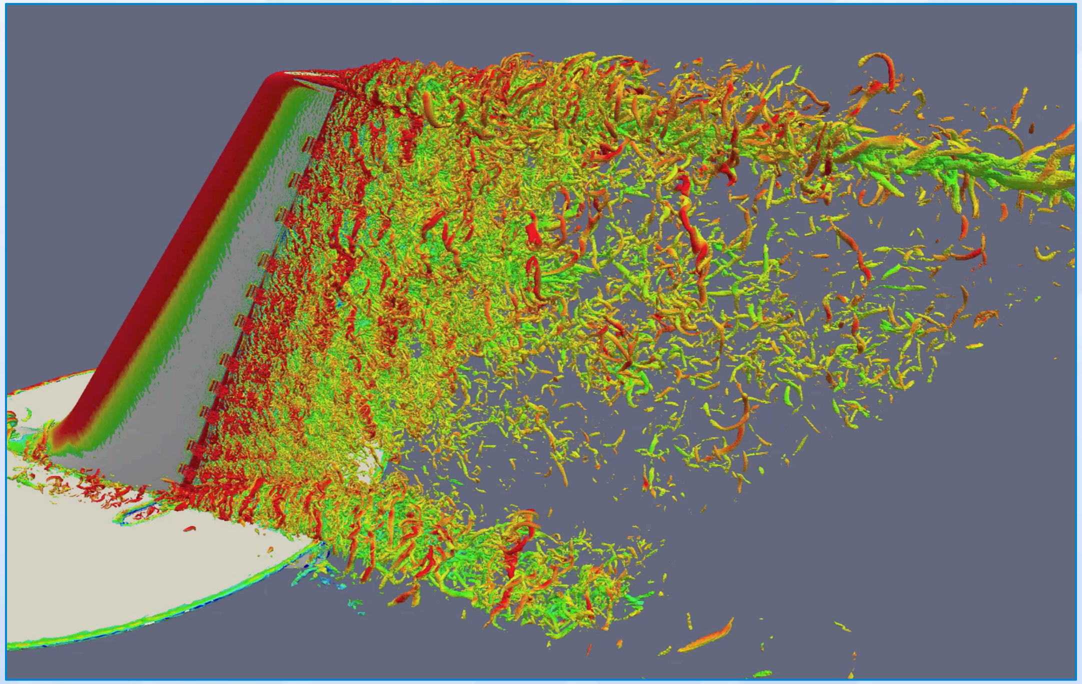


Going to Exascale

- 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

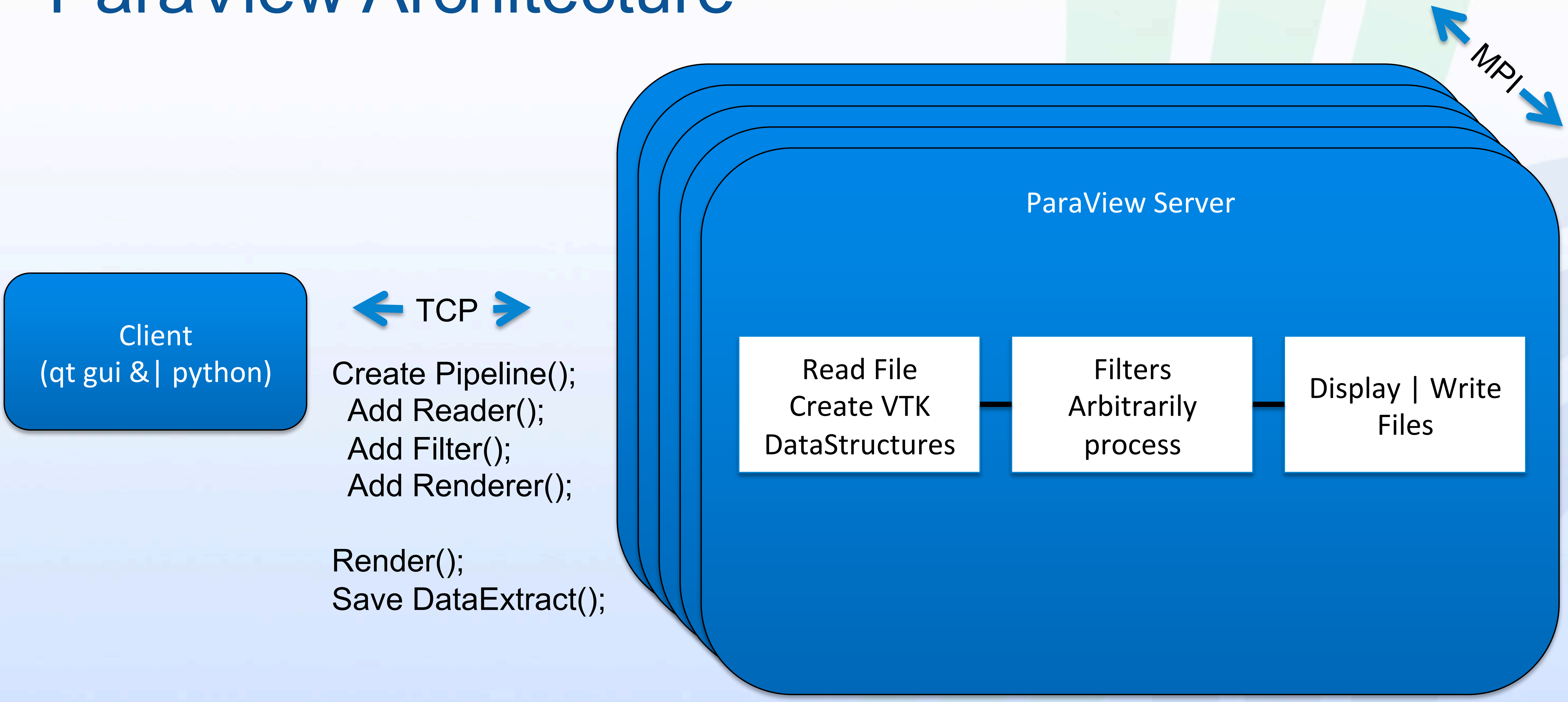
1M MPI ranks on Mira@ANL (BG/Q)

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

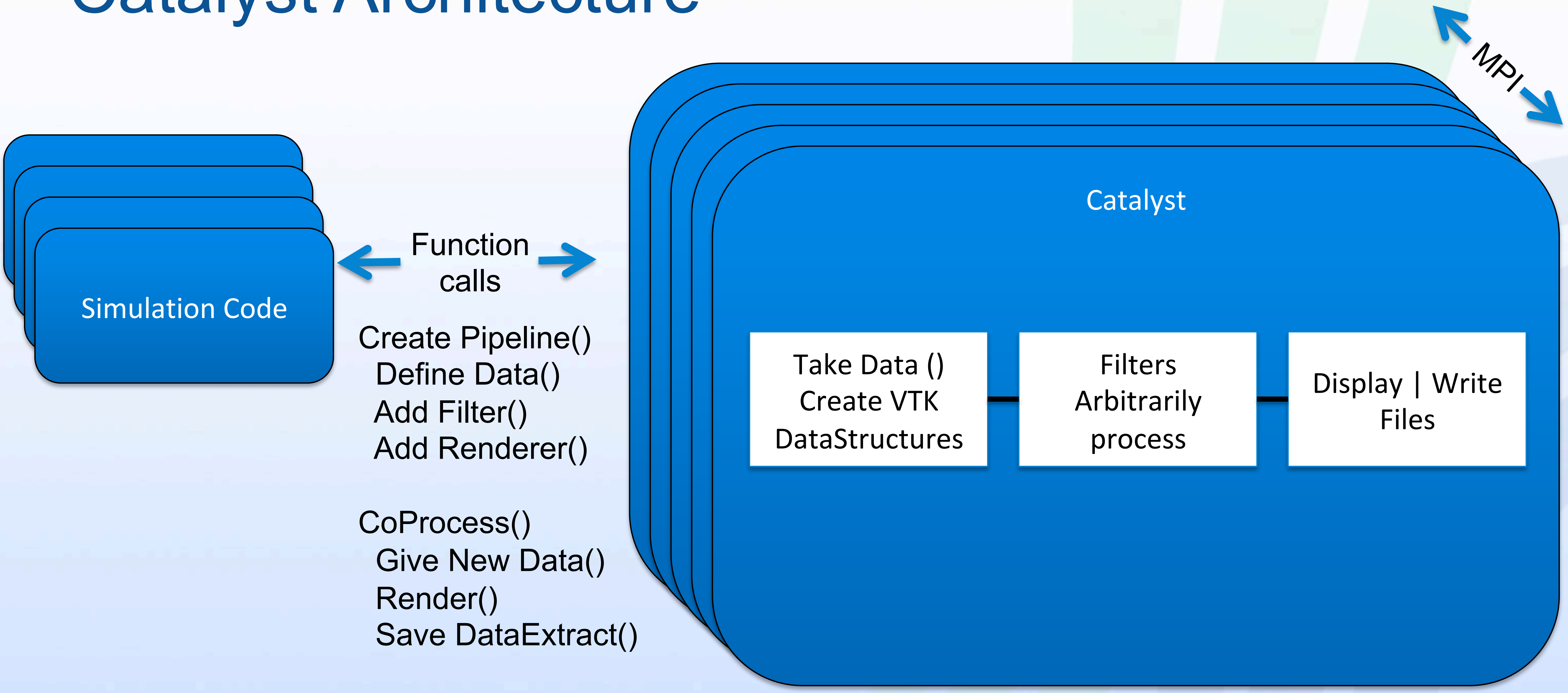


Movie courtesy Michel Rasquin (Cenaero/ UC Boulder)

ParaView Architecture



Catalyst Architecture



Motivation Continued

Bespoke -
Why not just hardcode plot routines?

Motivation Continued

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

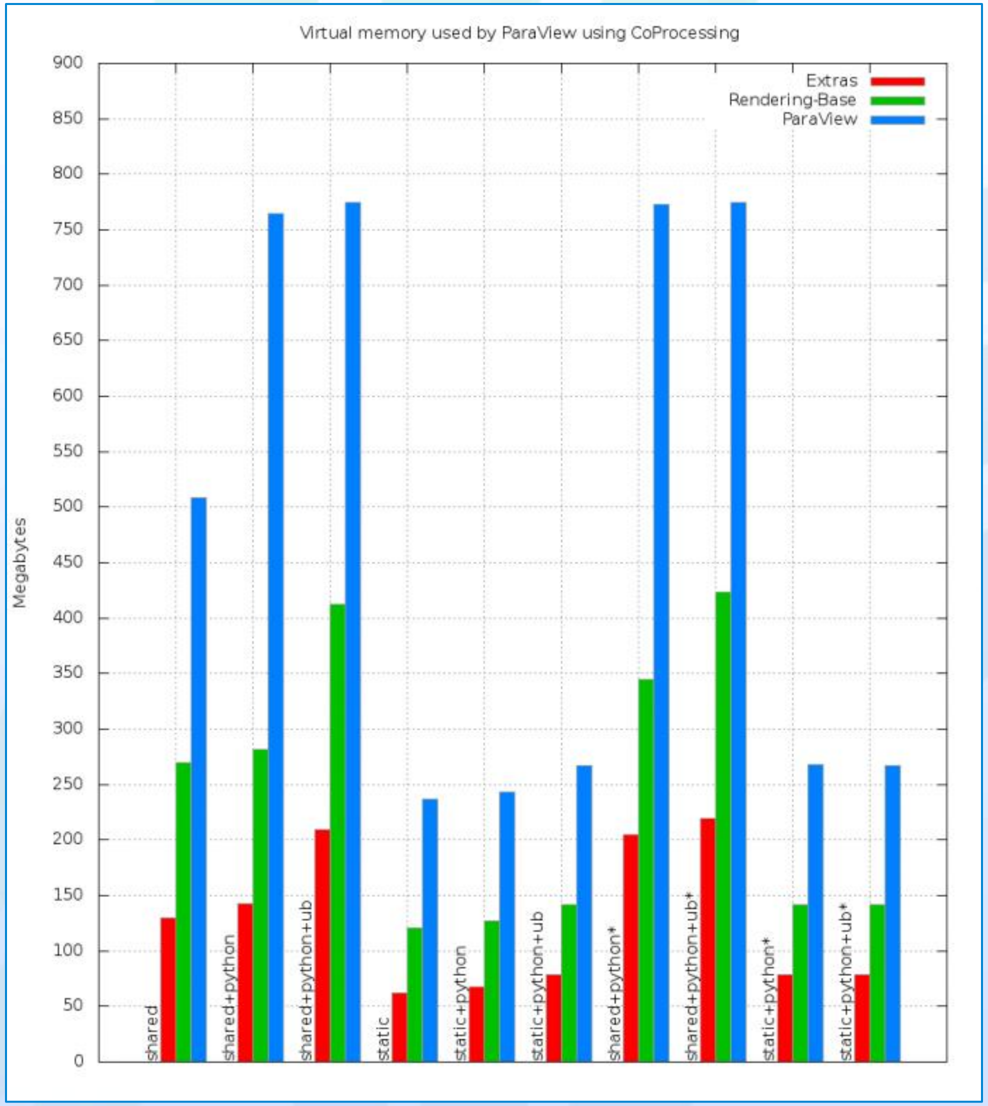
Custom

Maker scripts that build source tree

See 4.1 in Catalyst Users Guide

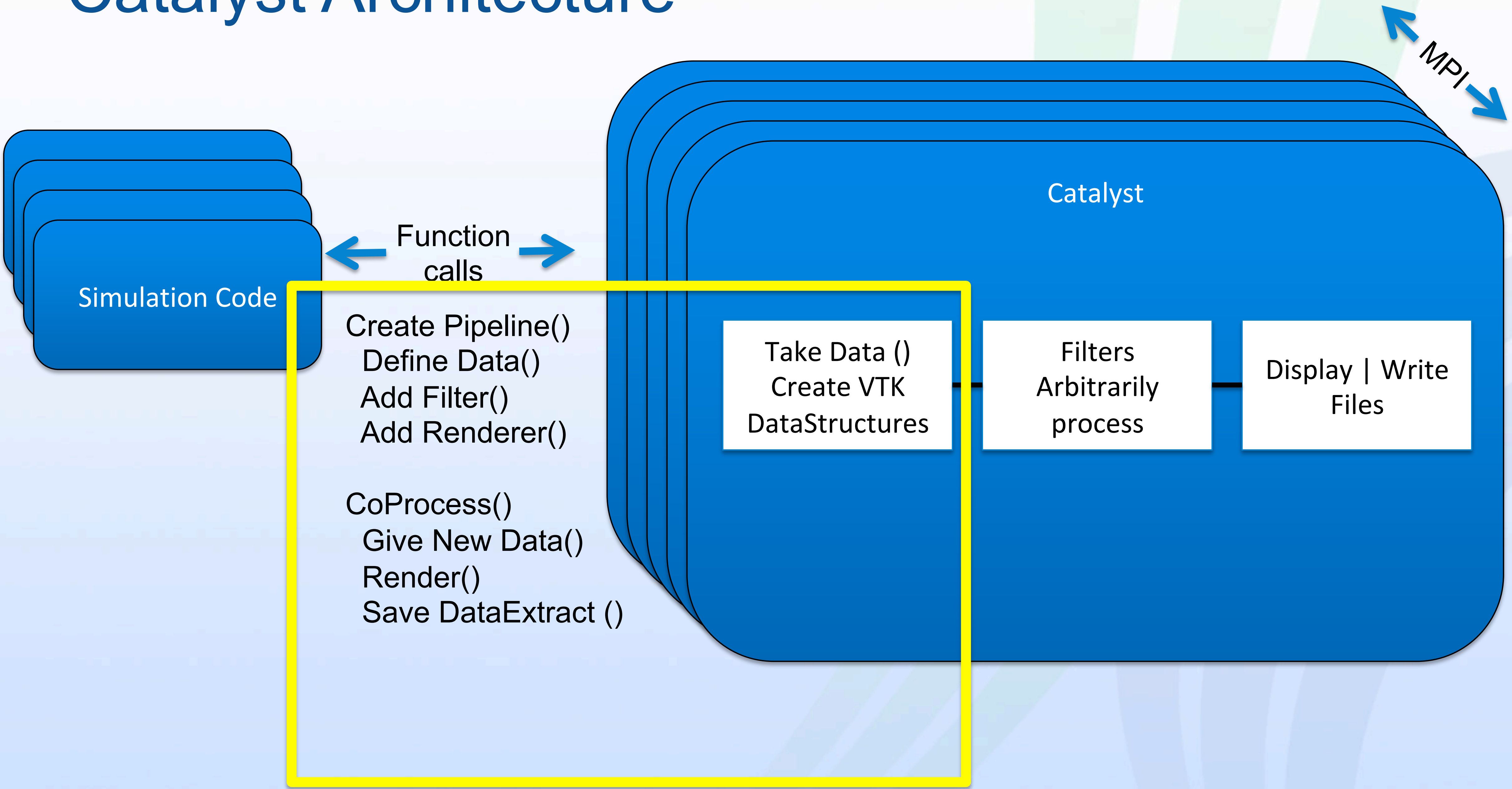
Efficient Memory-Wise

- 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
   - vtkCPProcessor::Initialize()
   - vtkCPProcessor::CoProcess()
   - vtkCPProcessor::Finalize()
3. Translate Simulation Data to VTK Data Structures
   - vtkCPInputDataDescription::SetGrid(vtkDataObject *)

4. Define Pipeline/Results to gather

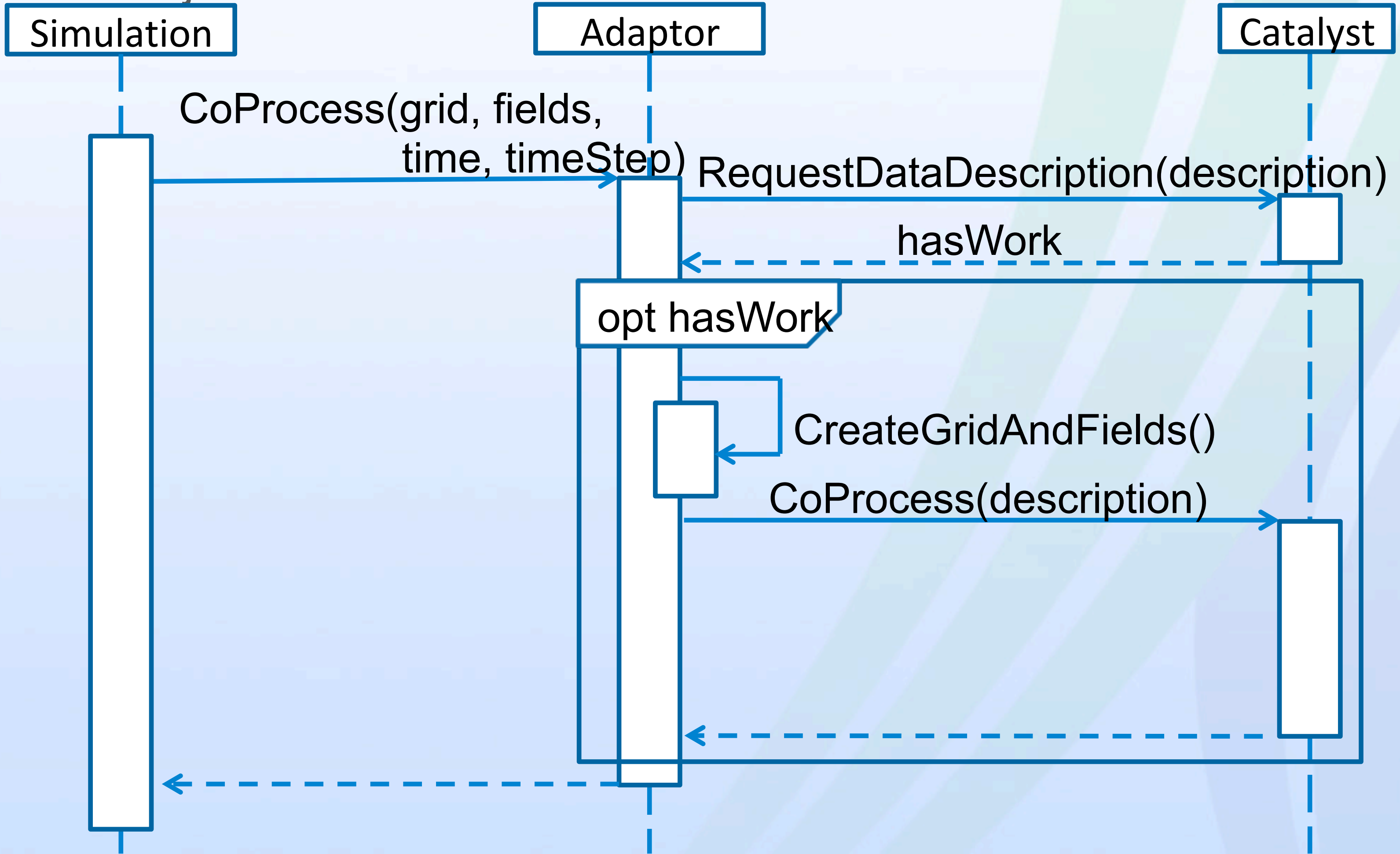
   Use generic option and let simulation user do this

API for the Developer

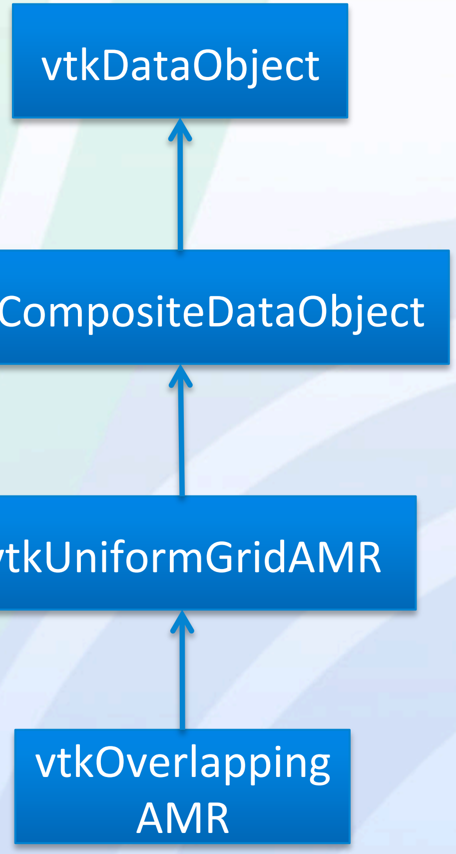
- vtkCPDataDescription
  - A means to transfer information from sim to ParaView (->) and back (<-)
  - -> named map of Grids for which new data is ready
    - "name" : vtkCPInputDataDescription
  - -> Time
  - <- Is Data necessary at this time?
- vtkCPInputDataDescription (CPIDD)
  - A container for 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



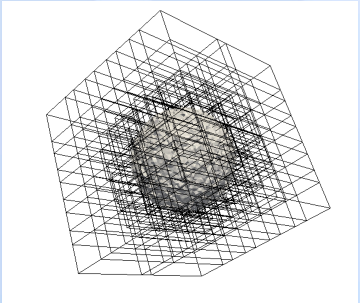
Populating vtkDataObjects



- 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 ← vtkSOADataArray



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

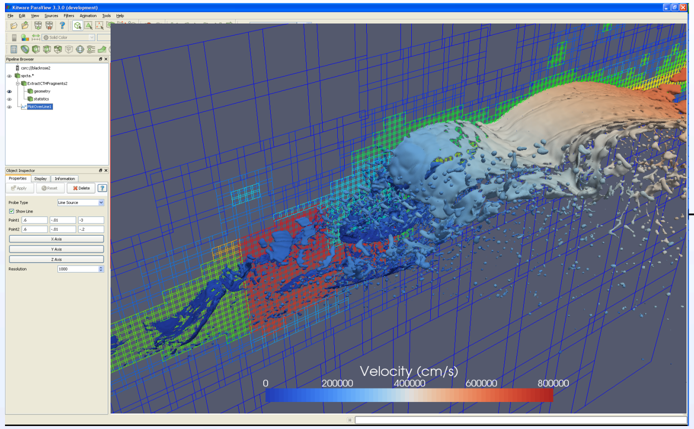
API for the Developer continued

- vtkCPProcessor - Manages the whole server
  - Sim runs visualization through CPProcessor 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

User Workflow



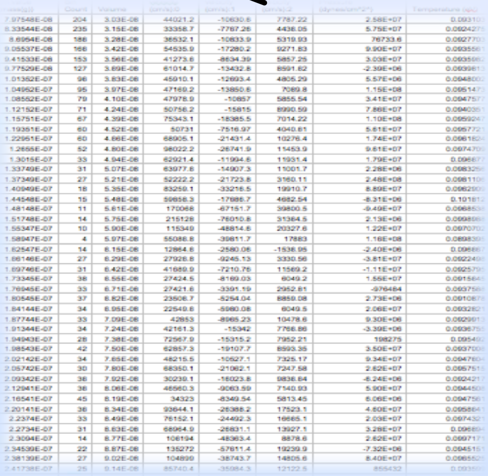
Script Export

```
# 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("displacement9")
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"
```

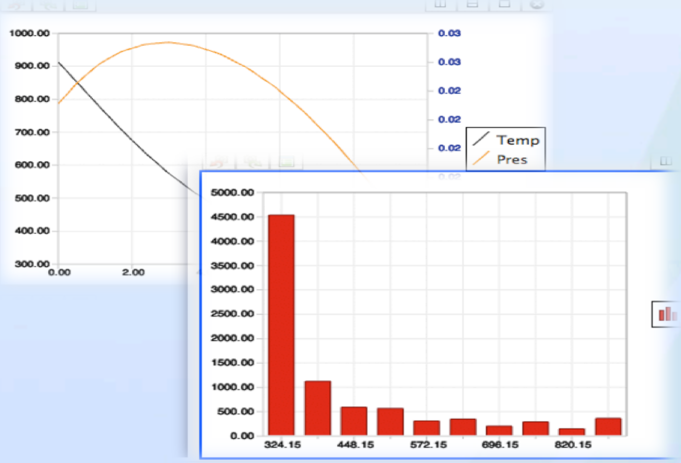
Augmented script in input deck

Simulation

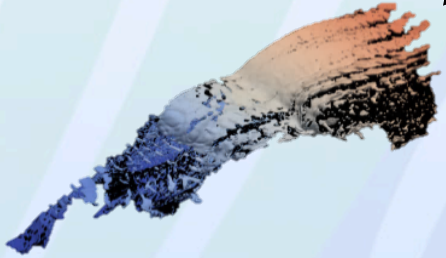
Catalyst



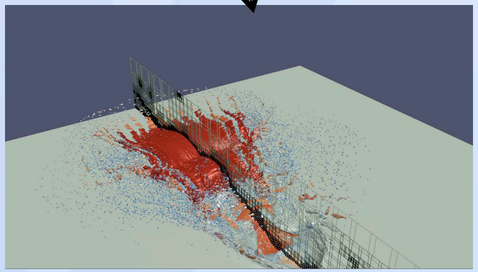
Statistics



Series Data



Polygonal Output with Field Data



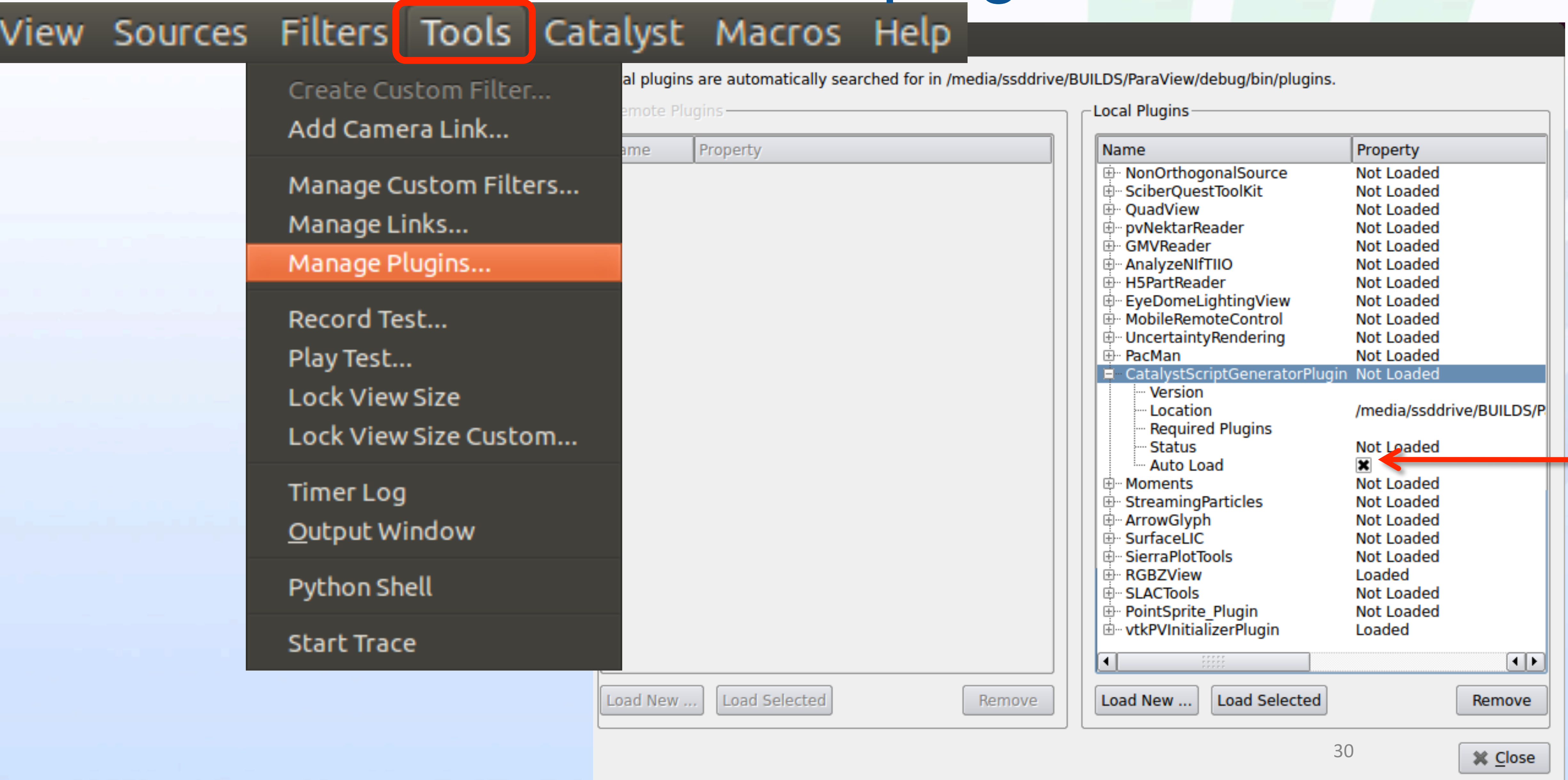
Rendered Images

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~~

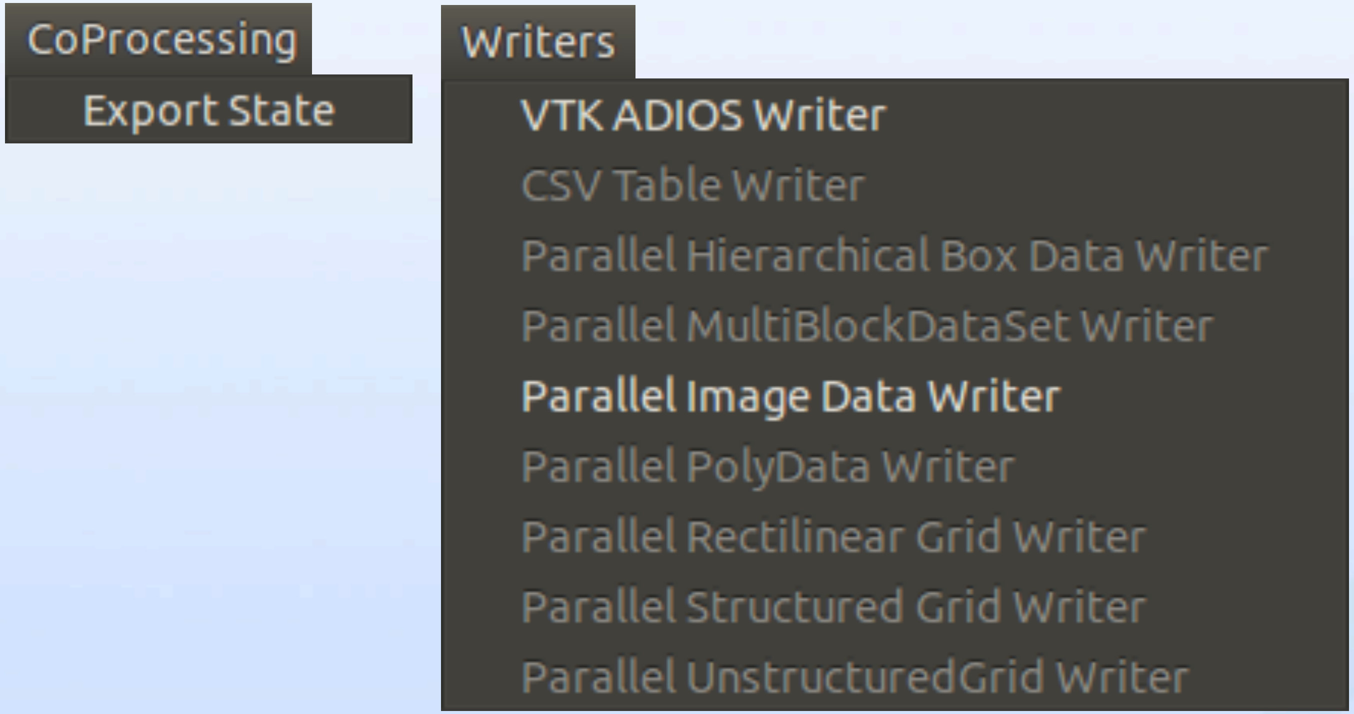
Only if ParaView < 5.5



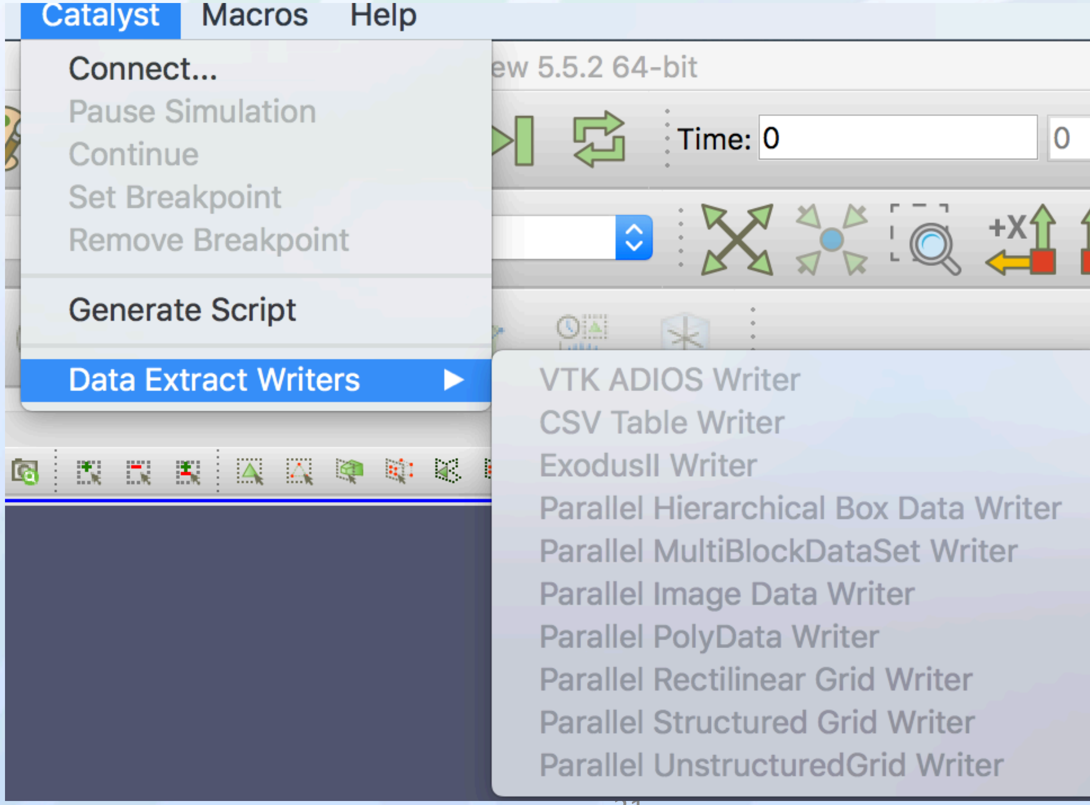
~~Plugin adds two new menus~~

~~Two new menu items specific to creating Catalyst Python scripts~~

< PV 5.5



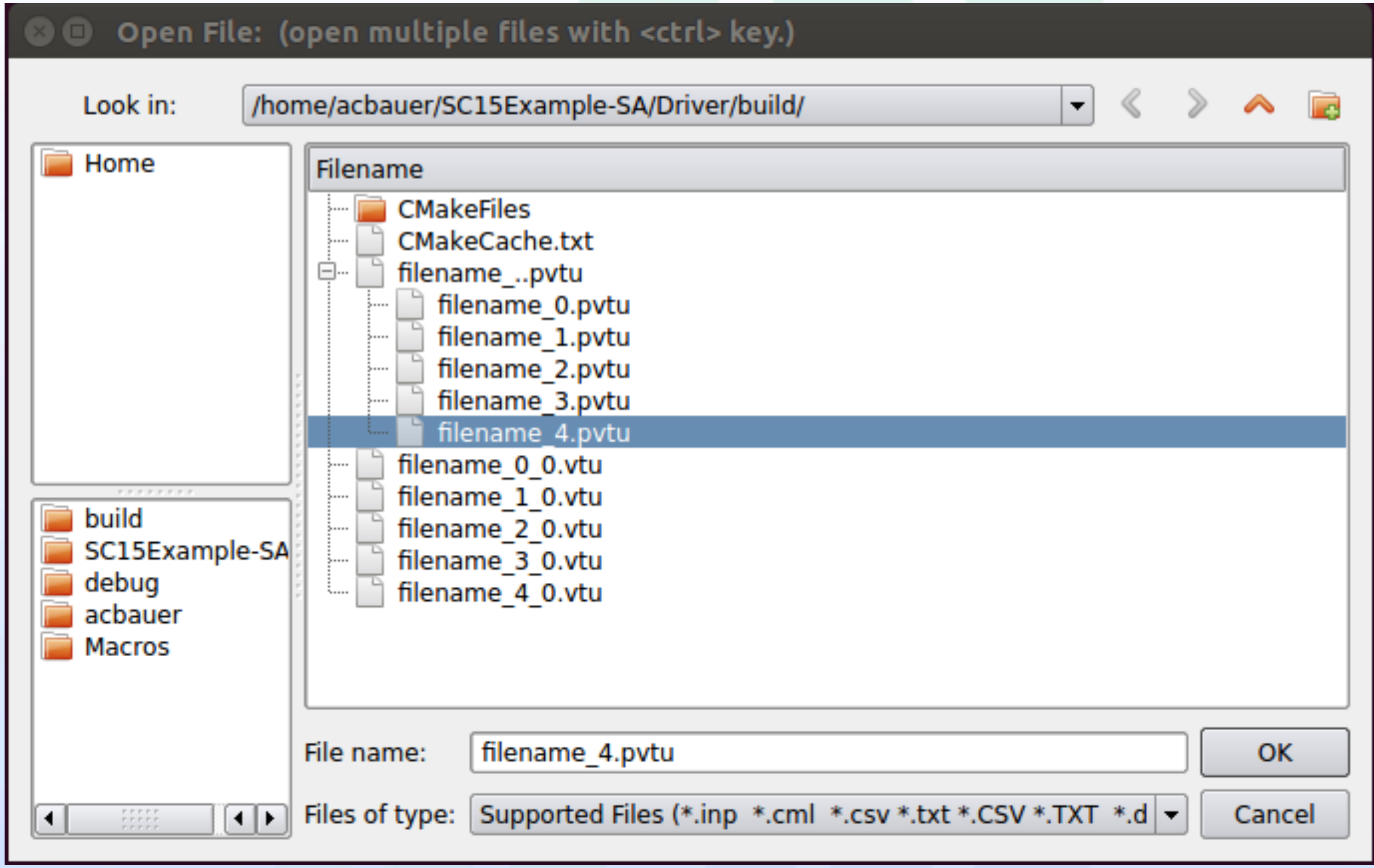
> PV 5.5



Create Pipeline

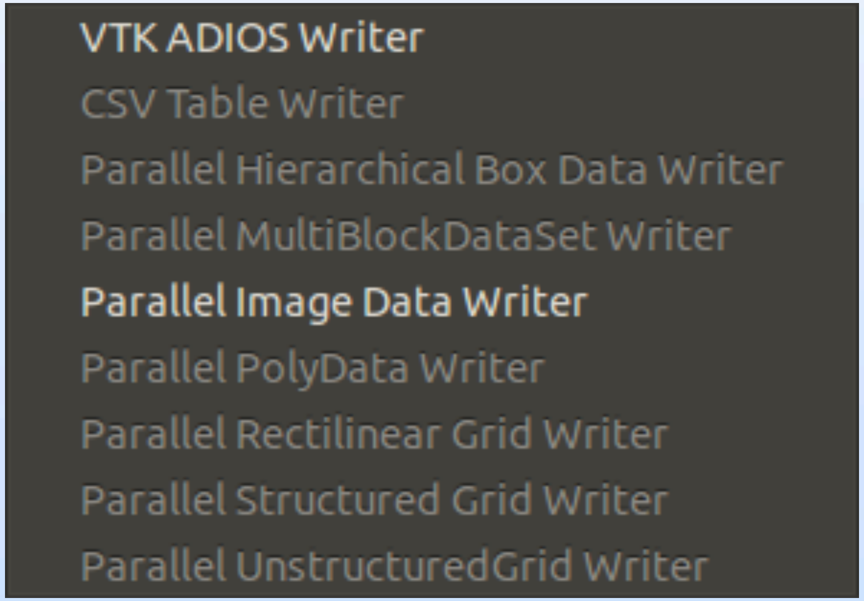
Load “representative” data set

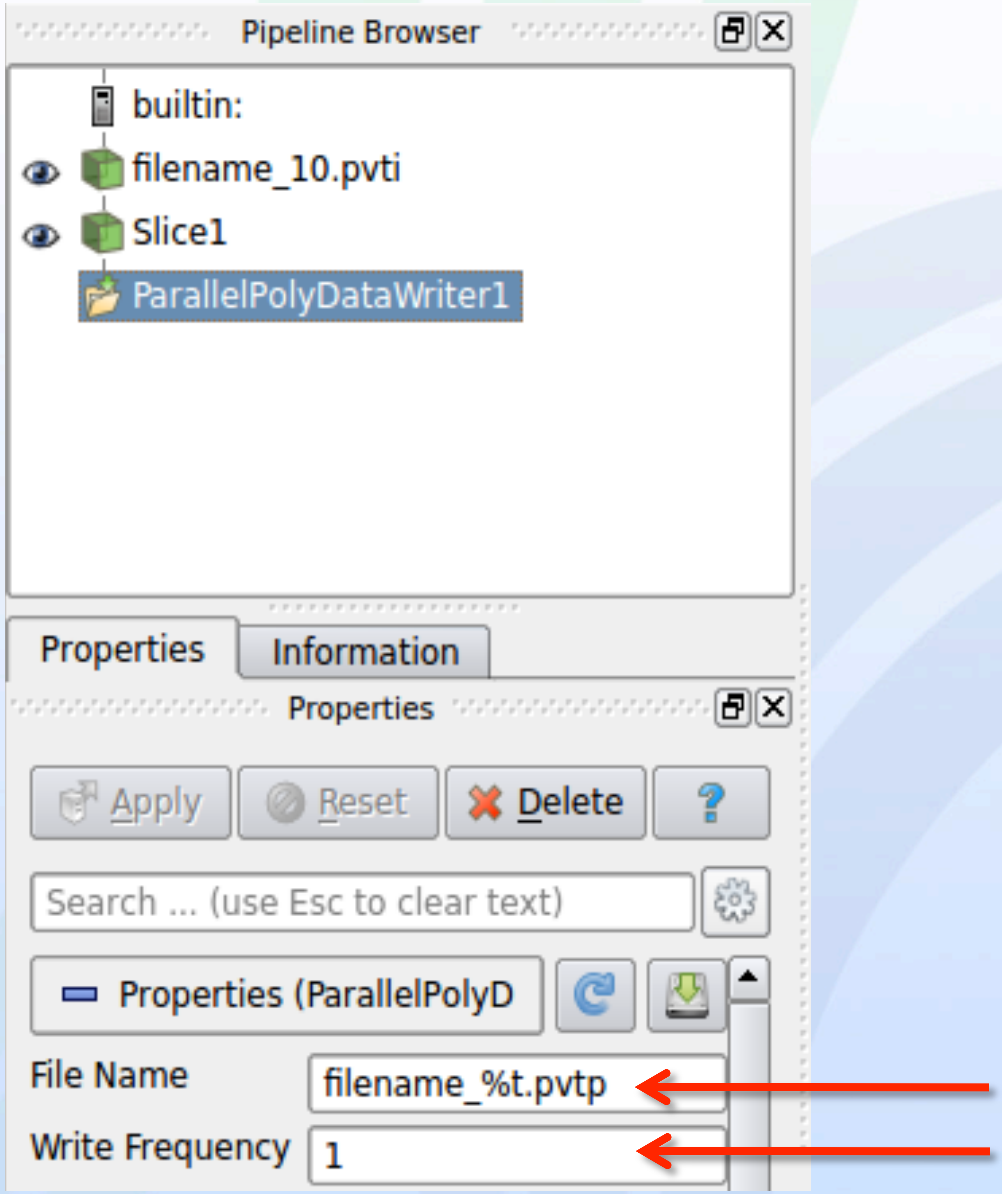
- **File→Open…**
  - Shortcut 
  - Choose /home/catalystuser/filename_4.pvtu



Define Writers

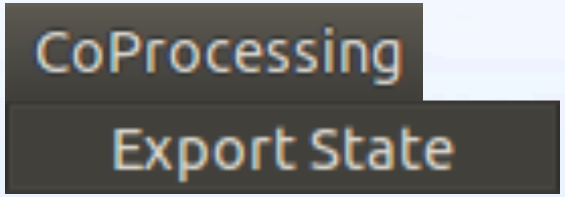
- Only valid writers available in **Writers** menu
- Parameters:
  - **File Name** – %t gets replaced with time step
  - **Write Frequency**



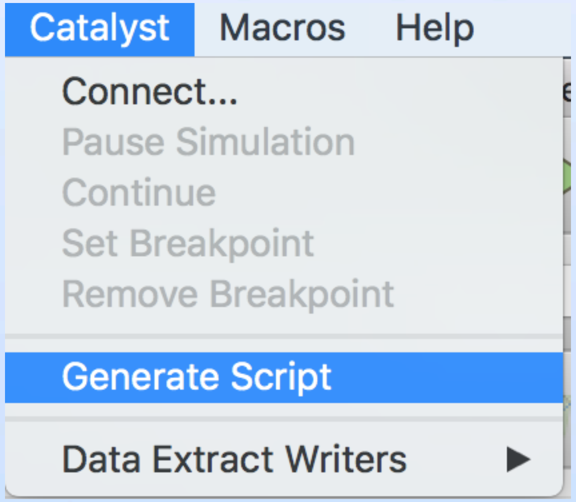


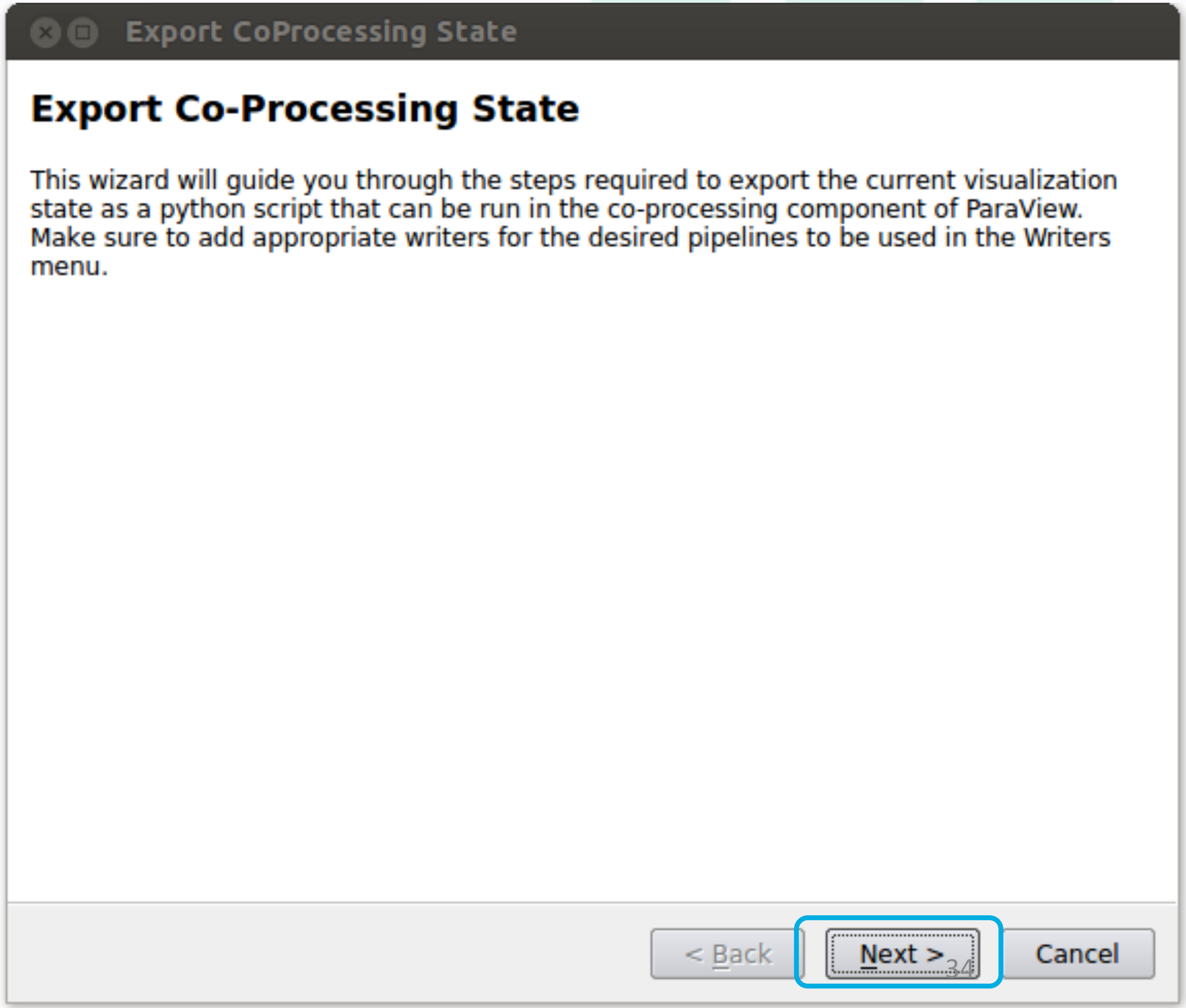
Export the Pipeline (Script)

< PV 5.5



> PV 5.5

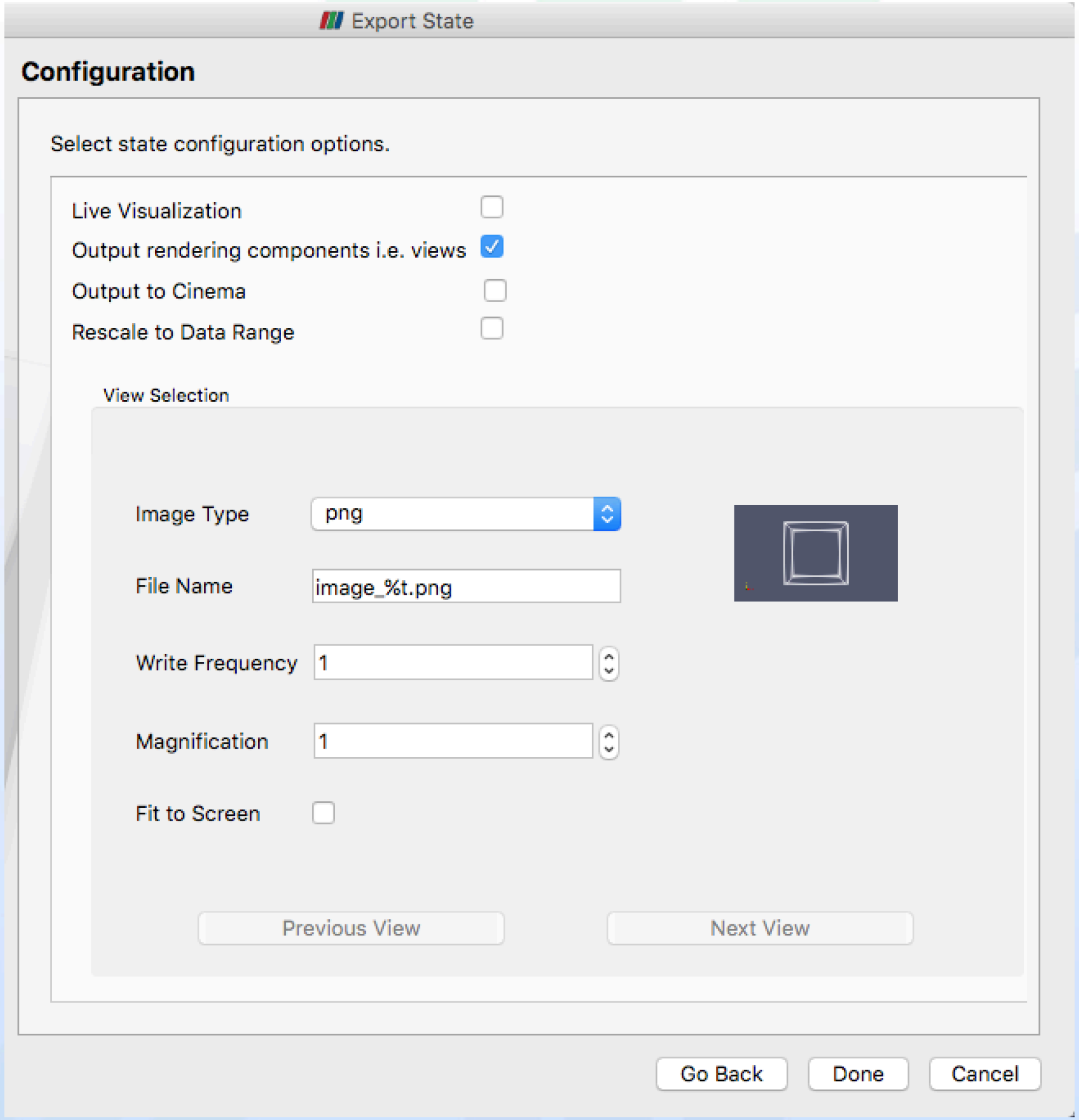




Configure Renders

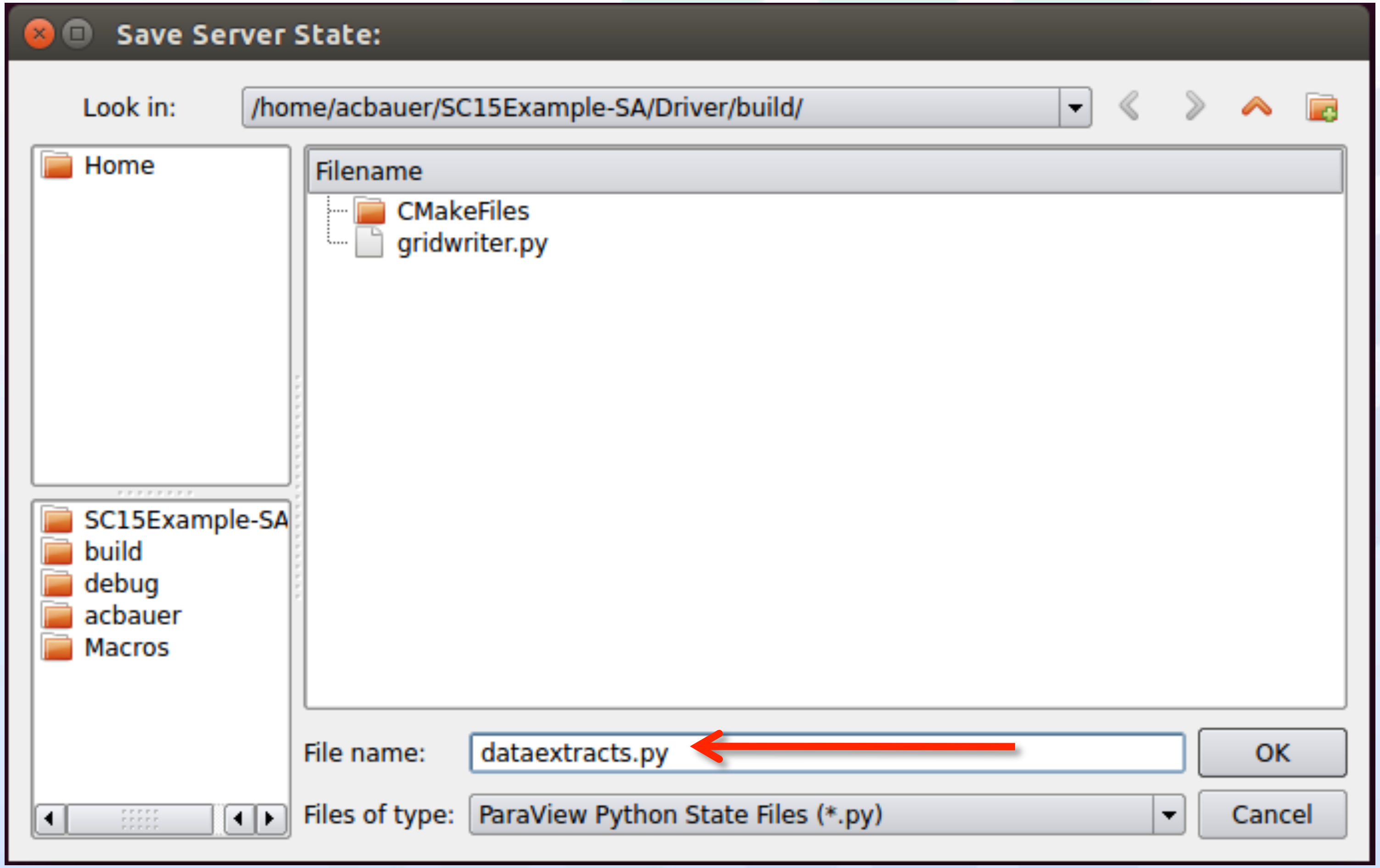
- Click "Output rendering components i.e. views"
- Choose which Windows to export
  Previous View, Next View
- Set Render Frequency

In PV >= 5.6 will UI will be streamlined:

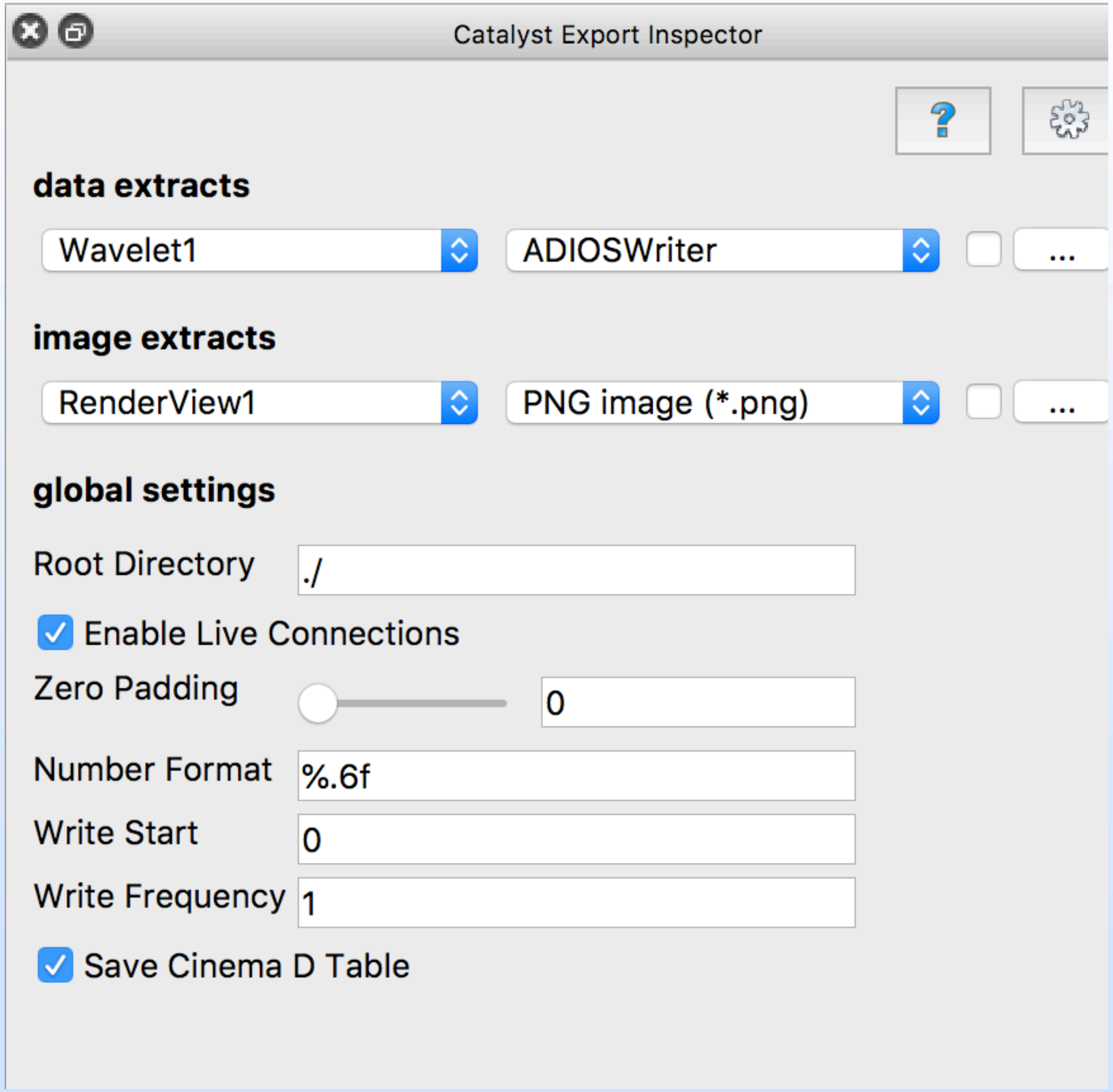


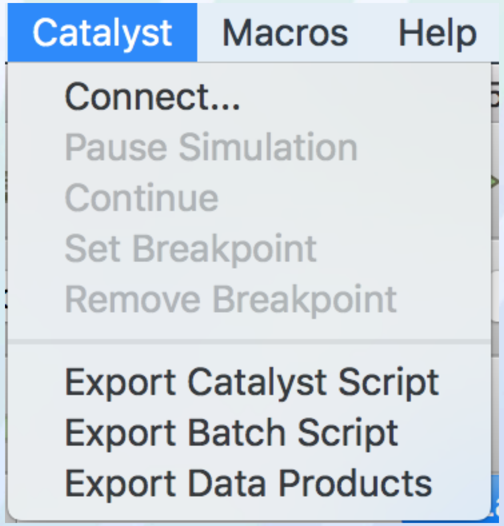
Save Script

- 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( XMLMultiBlockDataWriter, "filename_%t.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.vtkSMProxyIterator()
    iter.Begin()
    tobedeleted = []
    while not iter.IsAtEnd():
      if iter.GetGroup().find("prototypes") != -1:
         iter.Next()
         continue
      proxy = servermanager._getPyProxy(iter.GetProxy())
      proxygroup = iter.GetGroup()
```

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

```
begin results output
  database type = catalyst

  begin catalyst
    #save jpg images instead of png's
    image format = jpg

    #make them HD (720p) resolution
    image size = 1280 720
  end catalyst
end results output
```

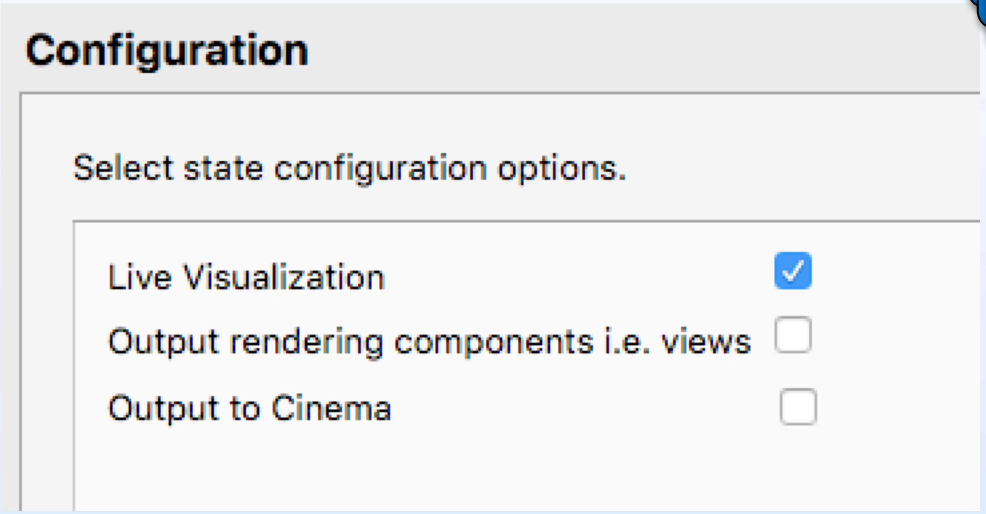
Listing 5. Imageset Command Example

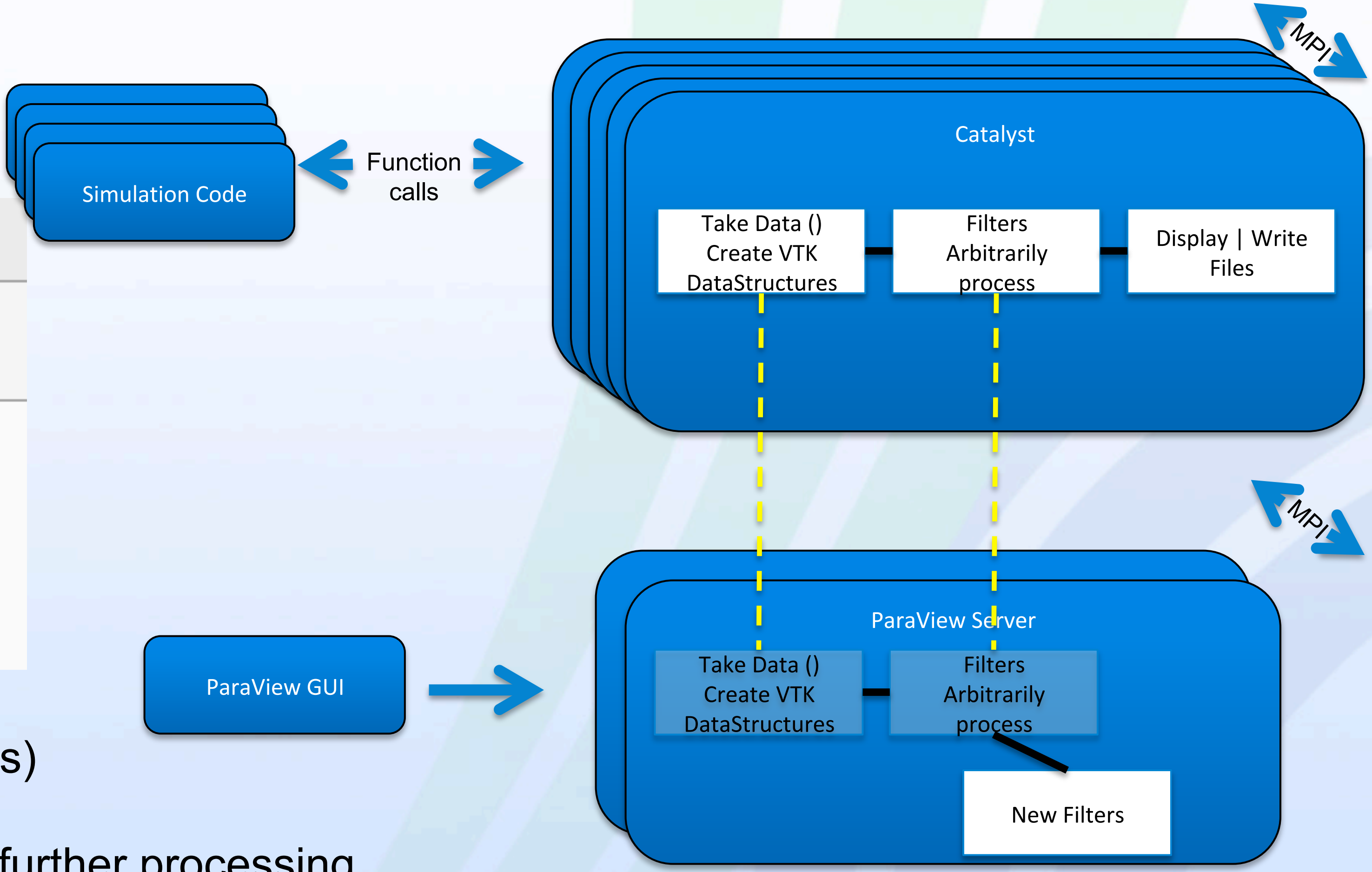
```
begin results output
  database type = catalyst

  begin catalyst
    #slice plane through the origin normal to the X axis
    slice = center 0 0 0 normal 1 0 0
  end catalyst
end results output
```

Listing 6. Slice Shortcut Command Example

Honorable Mention : Live

- Live

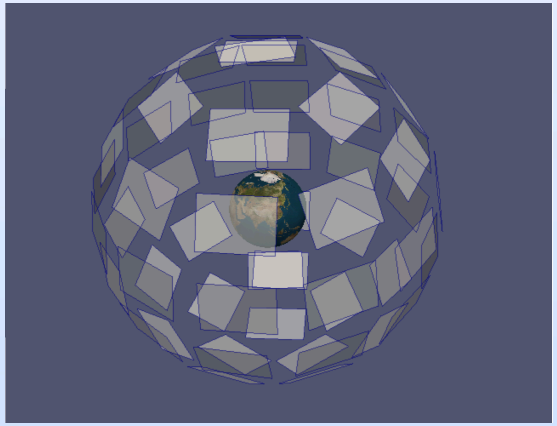


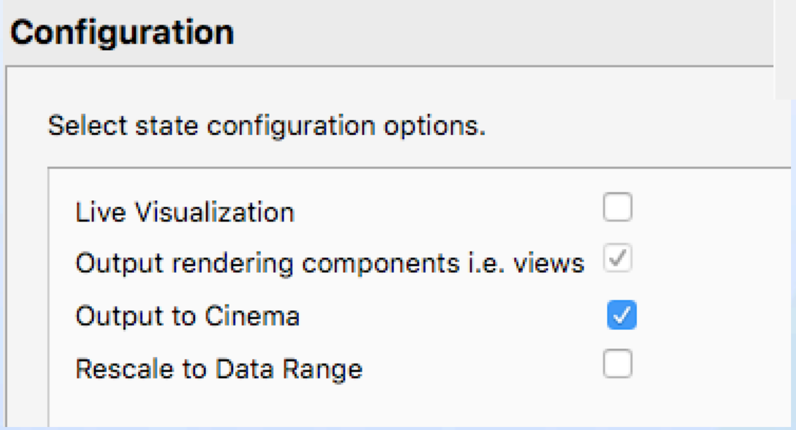


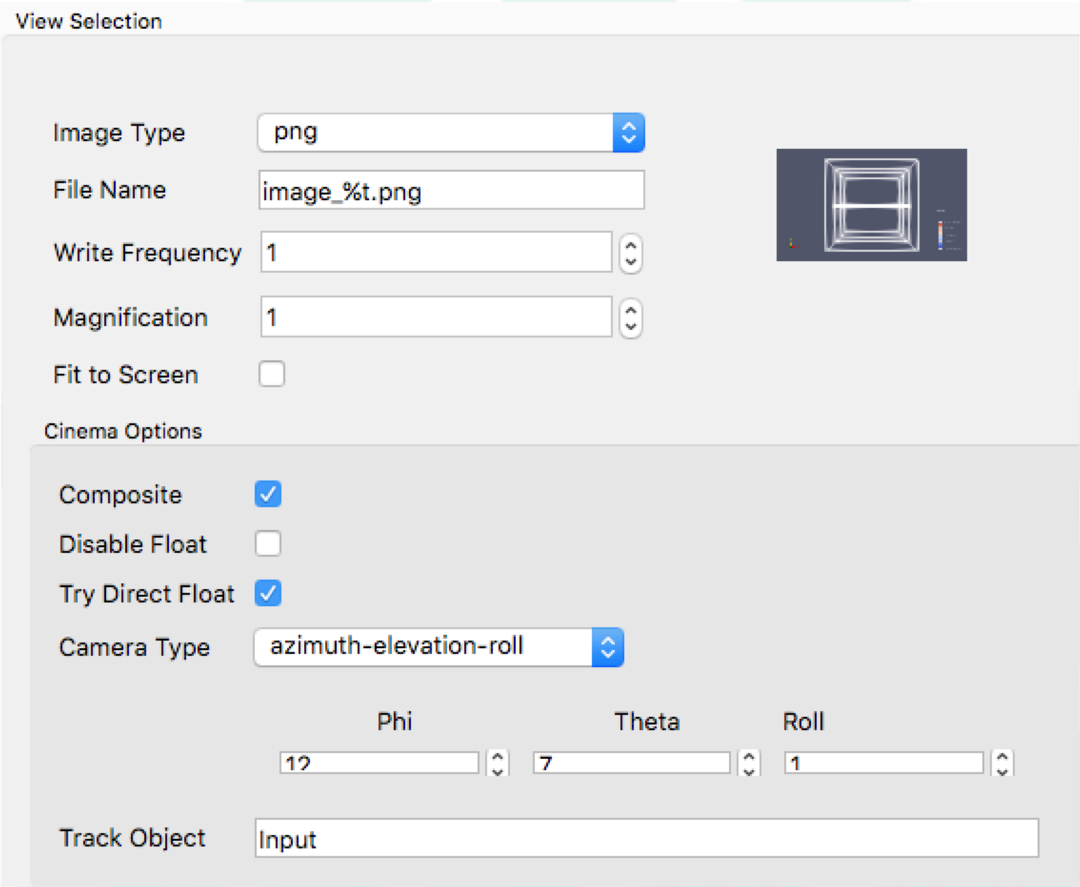
- Get references to simulation's pipeline(s)
- See data change as it evolves
- Optionally transfer designated data for further processing
- Optionally control settings of filters
- Optionally set breakpoints in simulation

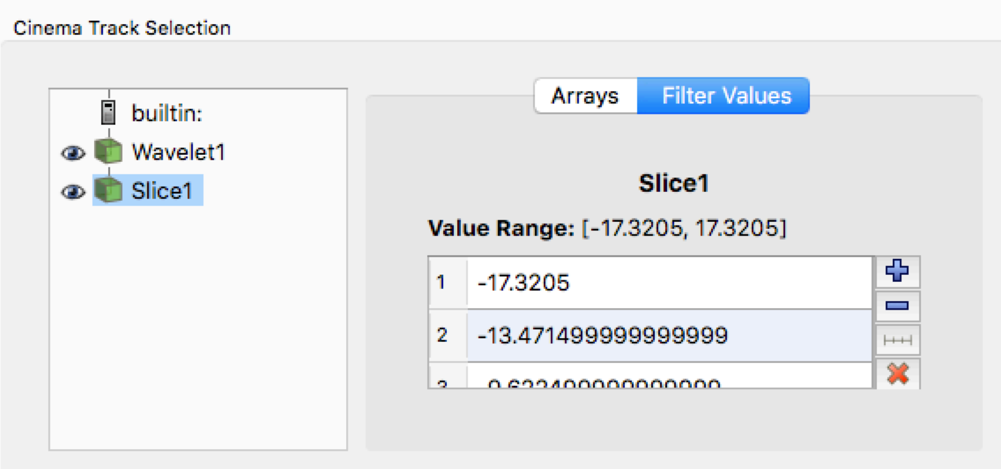
Honorable Mention

- 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



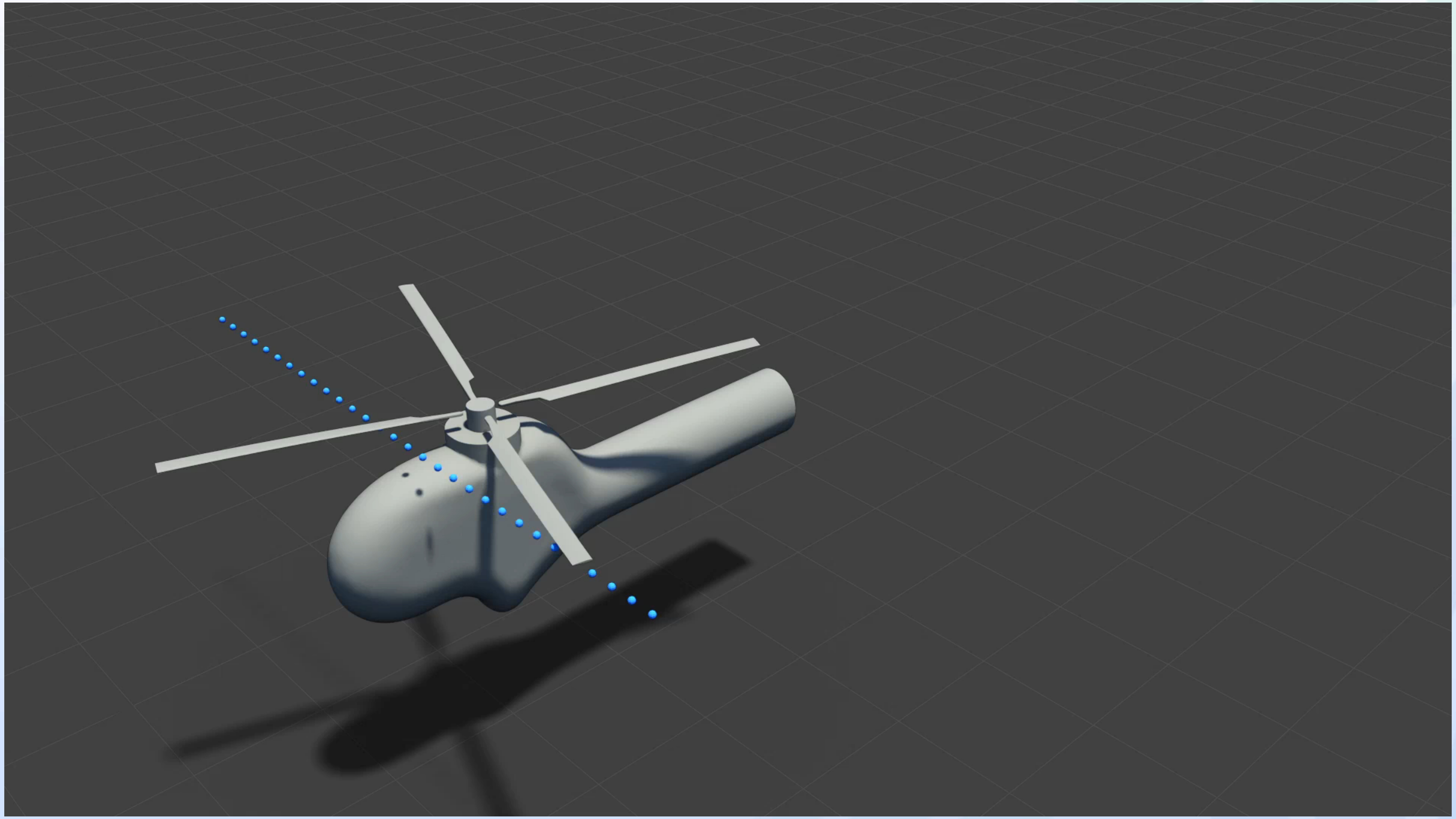






Thank You!

Further information: *andy.bauer*@kitware.com



Movie courtesy Mike Stephens (DAAC)