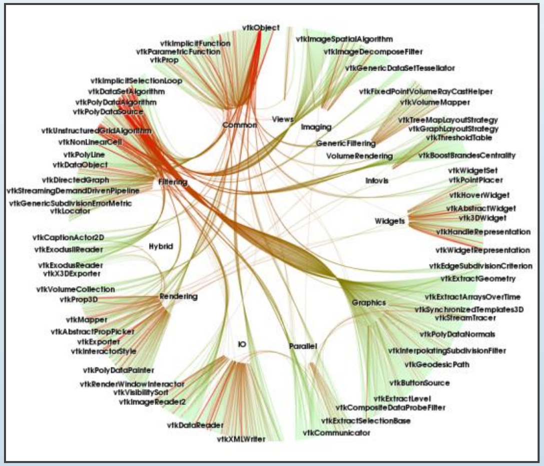
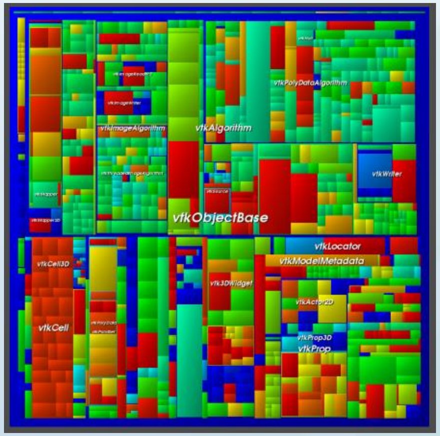
# VTK + ParaView Using NetCDF & VTK

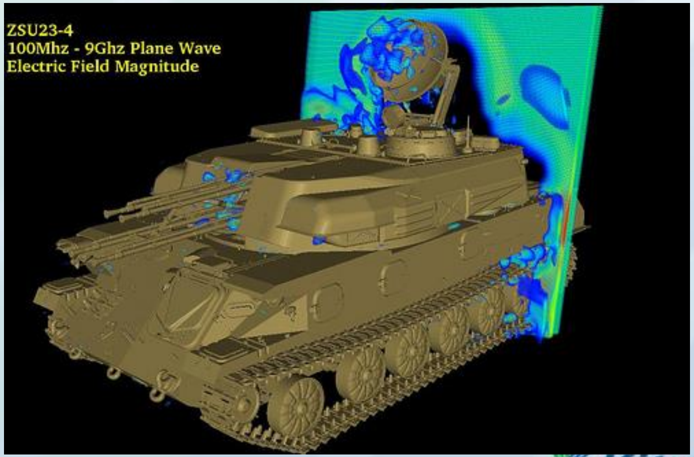
Andrew Bauer
andy.bauer@kitware.com

# The Visualization Toolkit (VTK)

- www.vtk.org
- Started in 1993 at GE
- Visualization Library
  - Written in C++ (+8.2 million LOC) – BSD License
  - Automatic binding for Java, TCL, Python
  - Portable by design: Linux, Windows, Mac OSX, Solaris…
- Very active community: 4000+ users on the mailing list

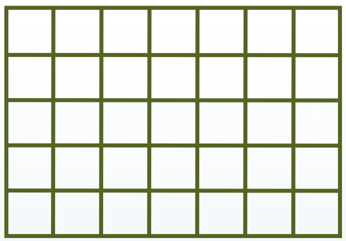




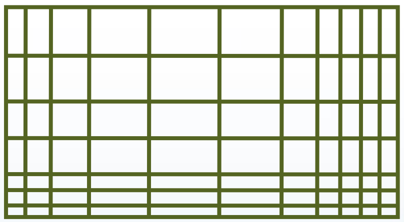



# VTK Data Types

vtkImageData

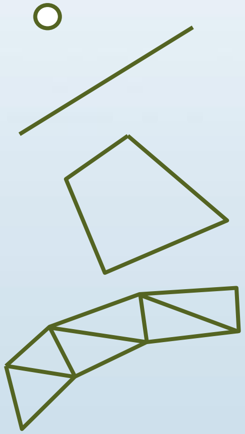


vtkRectilinearGrid

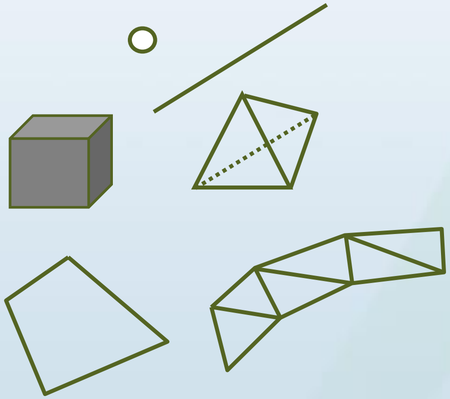


vtkStructuredGrid

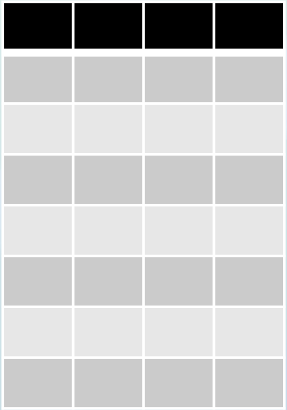
vtkPolyData



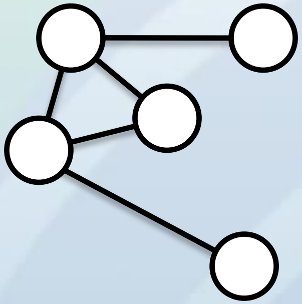
vtkUnstructuredGrid



vtkTable

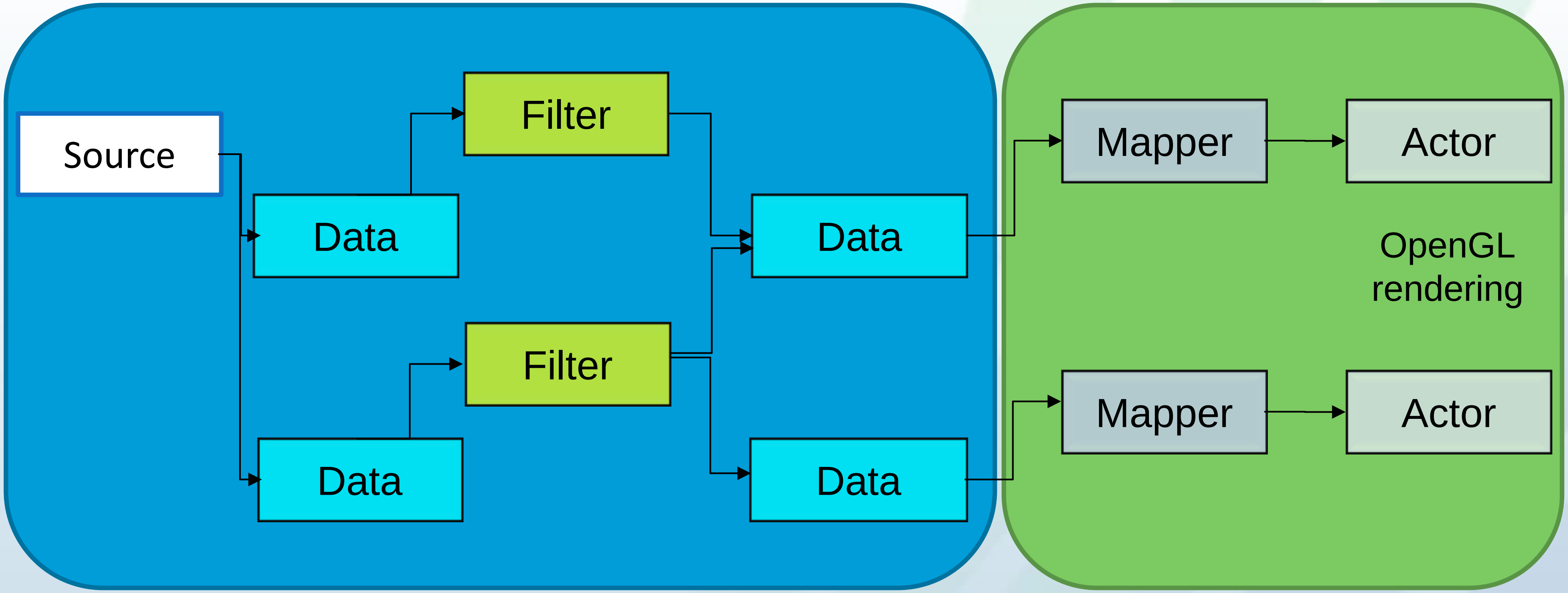


vtkGraph

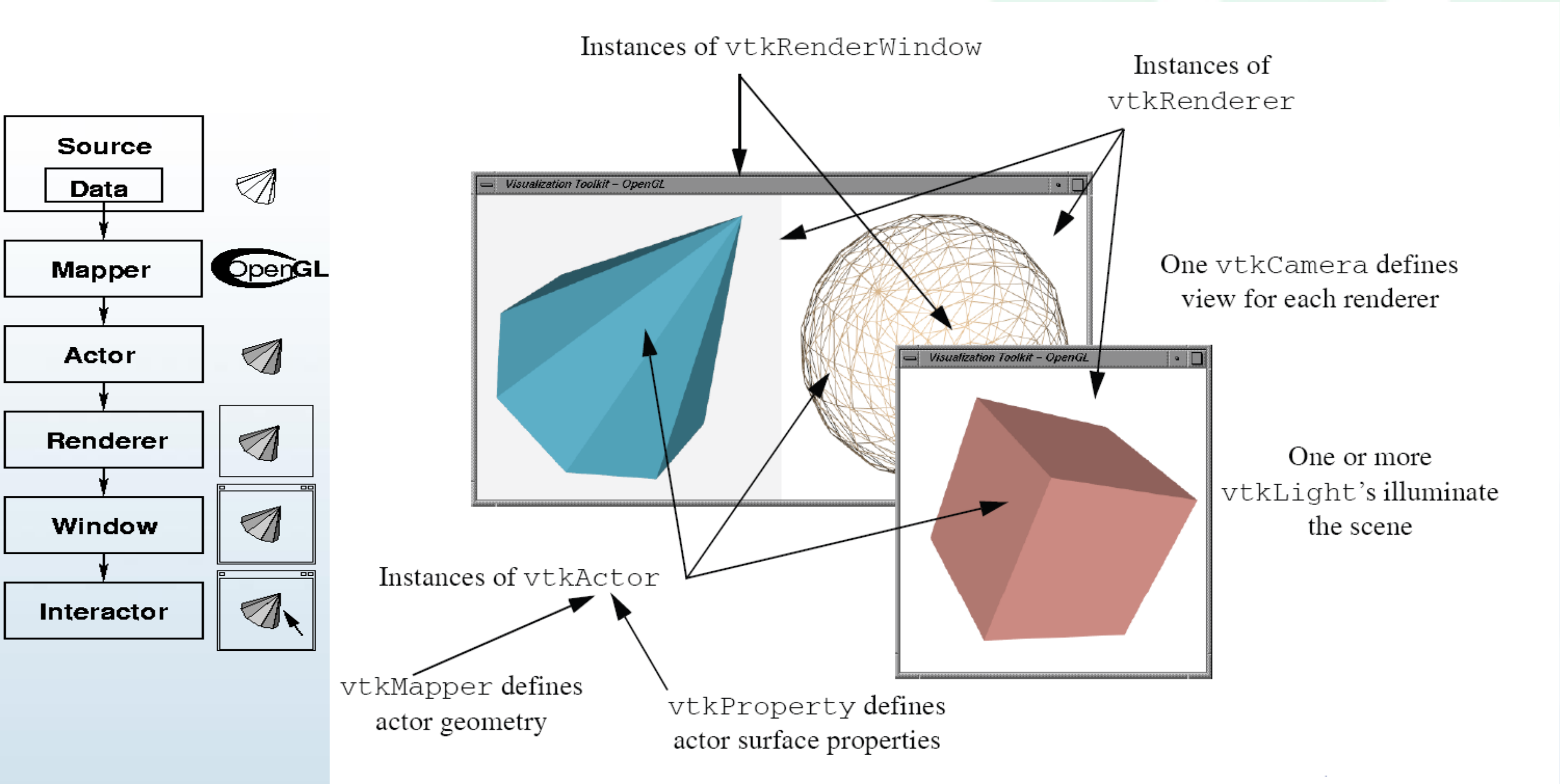


# VTK Pipeline Architecture

- A sequence of algorithms that operate on data objects to generate geometry that can be rendered by the graphics engine or written to a file



# The VTK Graphics Subsystem



# VTK with Python

- VTK distribution
  - www.vtk.org
  - vtkpython standalone python interface (Python 2.7)
- Anaconda
  - https://store.continuum.io/cshop/anaconda/
- Compilation from source
  - www.vtk.org (via CMake)

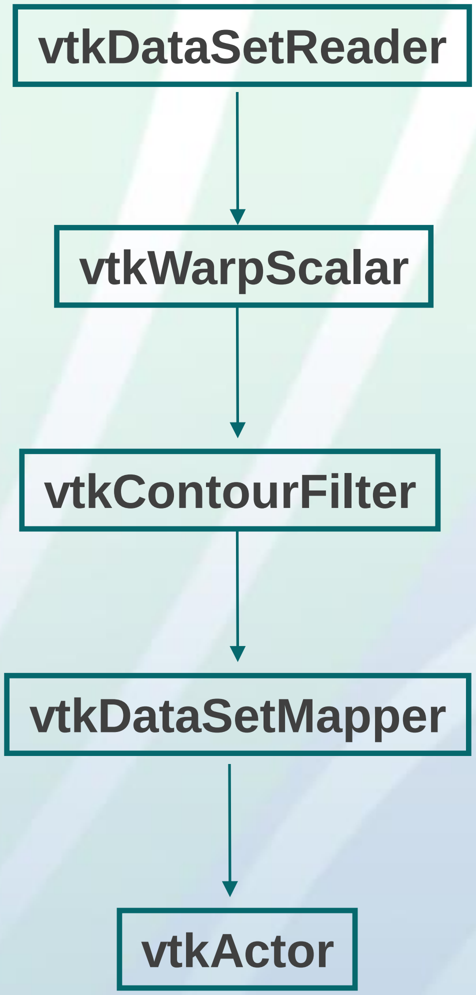
# Why Use VTK with Python

- Get direct access to VTK objects
  - Can inspect data objects and values
  - Can set data objects and values

- Access to VTK's filters
  - Read data
  - Operate on data
  - Write data
  - View data

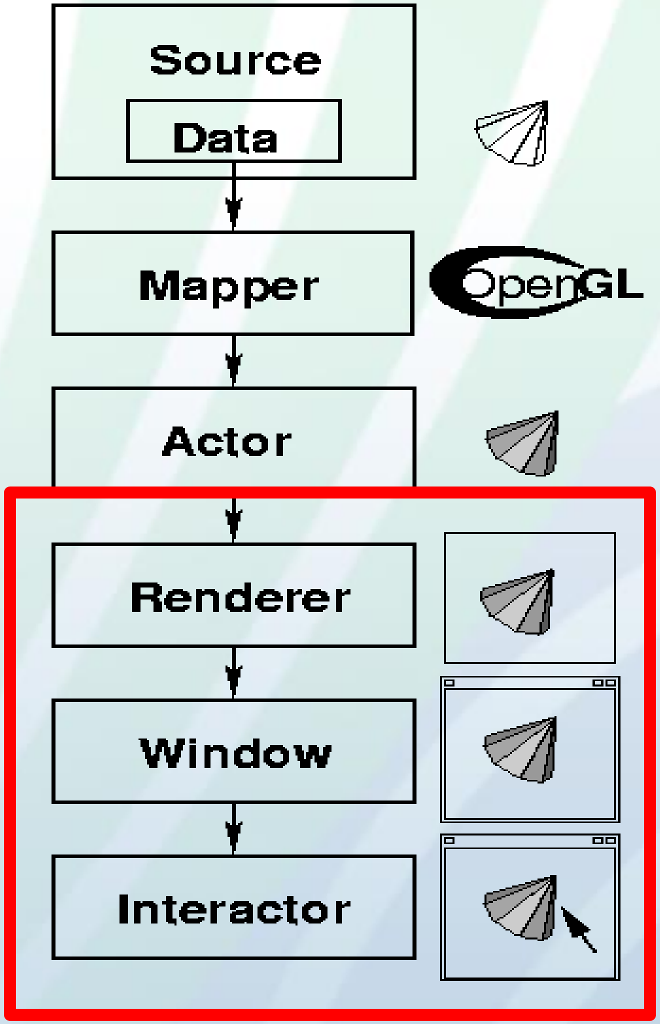
# Tutorial – Display and Interact with DEM Data

- Input Data
  - Digital Elevation model
  - Standard Geospatial File Format
  - SaintHelens.vtk
  - Structured Grid with an elevation scalar field
- We want to be able to
  - Load data
  - Represent the elevation → 3D
  - Represent isolines of elevation
  - Clip the data



# Step 1 – Creating the Display

```
# First thing first
import vtk

# Make a window on our display
renwin = vtk.vtkRenderWindow()
# Define the size of the initial window
renwin.SetSize(500,500)
# Create a renderer
renderer = vtk.vtkRenderer()
renderer.SetBackground2(1,1,1)
renderer.SetGradientBackground(1)
# Create an interactor
iren = vtk.vtkRenderWindowInteractor()
# Add the renderer to the render window
renwin.AddRenderer(renderer)
# Set the interactor to the render window
renwin.SetInteractor(iren)
# Show the (empty)
windowrenwin.Render()
```

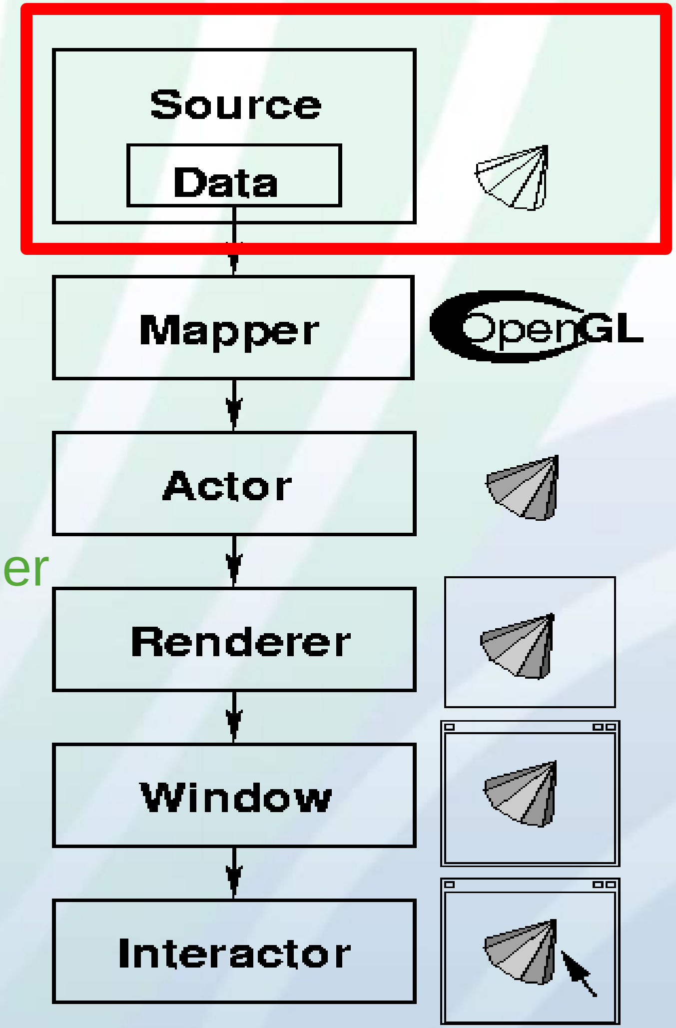


# Step 1b – Enabling interactions

```
# Initialize the interactor
iren.Initialize()
# Update the rendering pipeline
renwin.Render()
# Start the UI event loop
iren.Start()
```

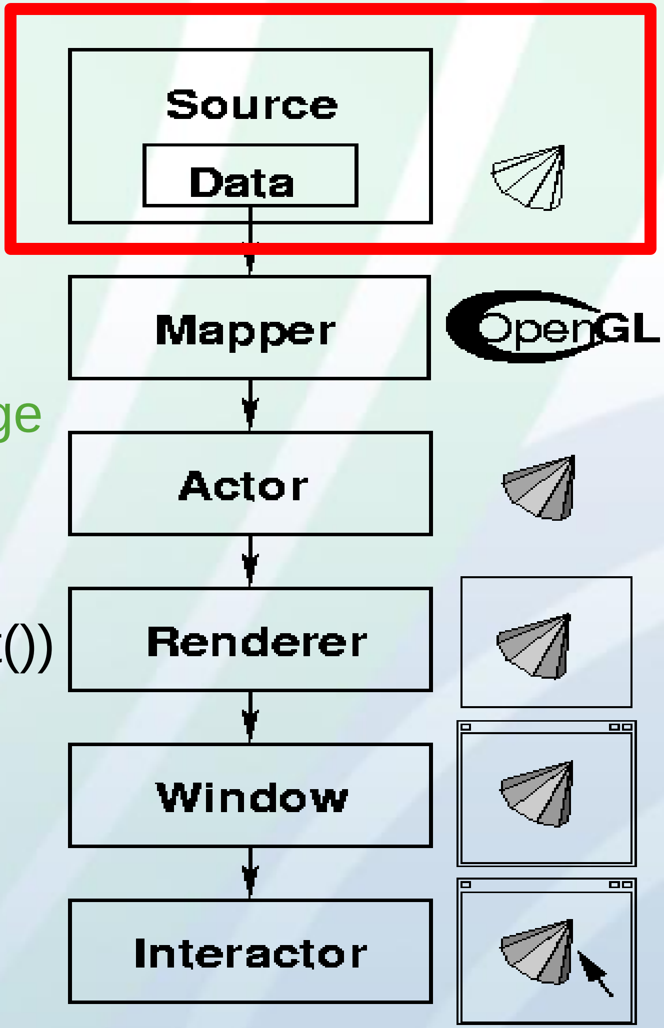
# Step 2 – Read the DEM file

```
# Creating a reader
reader = vtk.vtkDataSetReader()
filename = "SaintHelens.vtk"
reader.SetFileName(filename)
# Updating the pipeline
reader.Update()
# Printing information about the output of the reader
id = reader.GetOutput()
print(id.GetClassName())
print(id.GetBounds())
print id.GetPointData().GetArray(0).GetRange()
```



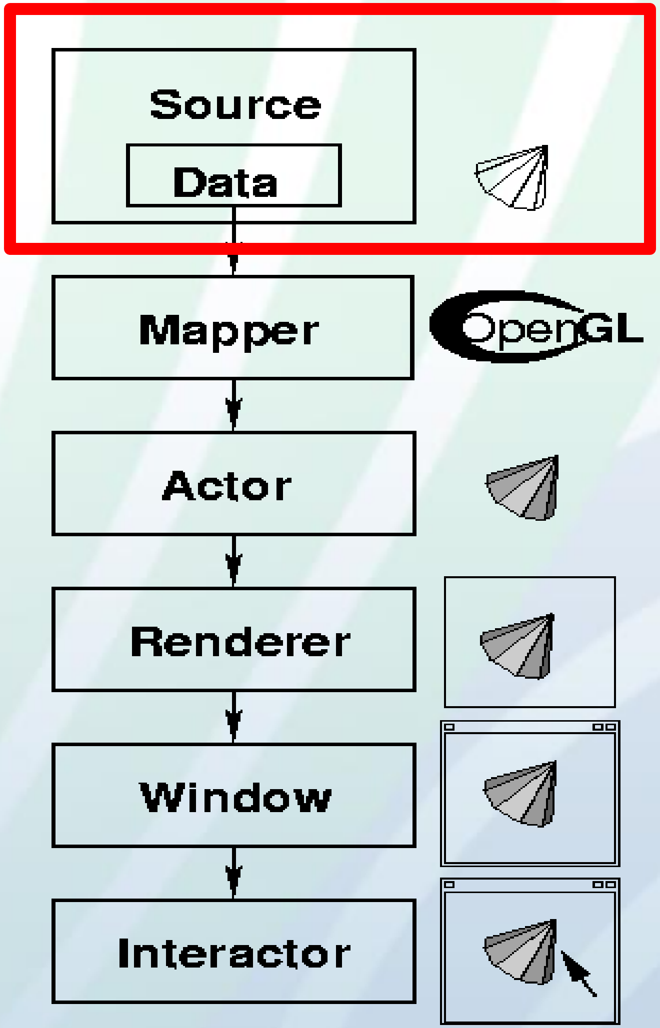
# Step 3 – Creating the mapper

```
# Create the mapper
mapper = vtk.vtkDataSetMapper()
# Set the visisbility on the scalar
mapper.ScalarVisibilityOn()
# Set the scalar range to have a nice blue-red range
mapper.SetScalarRange(682.0, 2543.0)
# Connect the mapper to the reader
mapper.SetInputConnection(reader.GetOutputPort())
```



# Step 4 – Creating the Actor

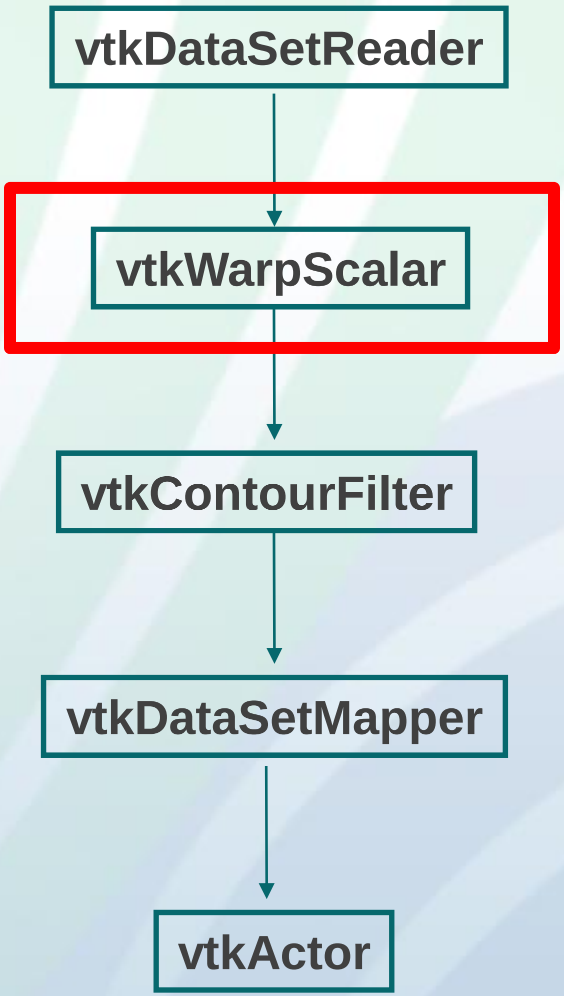
```
# Create the actor
actor = vtk.vtkActor()
# Sets the mapper to the actor
actor.SetMapper(mapper)
# Add the actor to the renderer
renderer.AddViewProp(actor)
# Perform rendering
renwin.Render()
# Center the camera to see the full scene
renderer.ResetCamera()
# Final rendering
renwin.Render()
```




Kitware

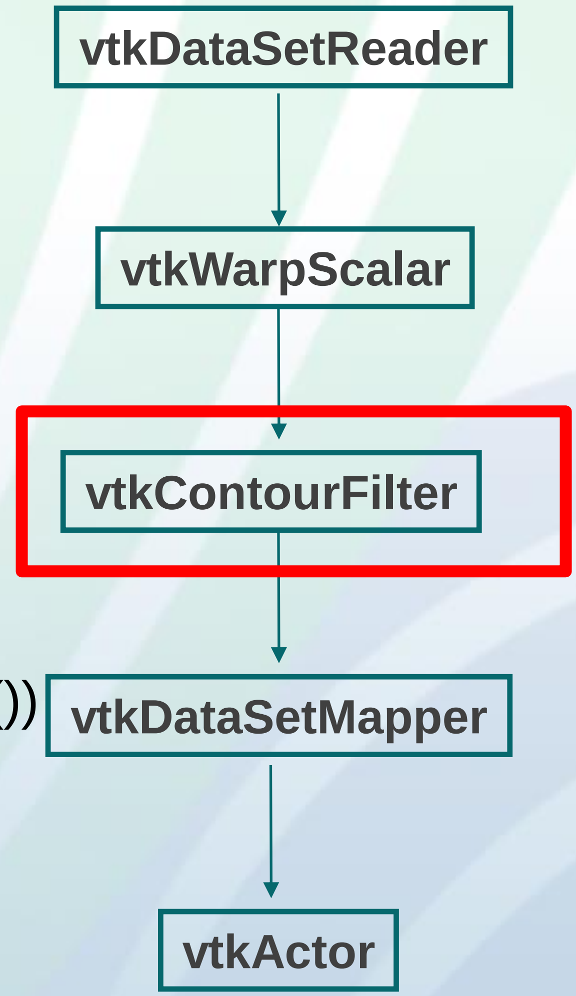
# Step 5 – Elevating our surface to 3D

```python
# We elevate the surface with the vtkWarpScalarFilter
warp = vtk.vtkWarpScalar()
# Connect the filter to the reader
warp.SetInputConnection(reader.GetOutputPort())
# Update the pipeline
warp.Update()
print(warp.GetOutput().GetBounds())

# Show the elevated surface
mapper.SetInputConnection(warp.GetOutputPort())
renwin.Render()
renderer.ResetCamera()
renwin.Render()
```



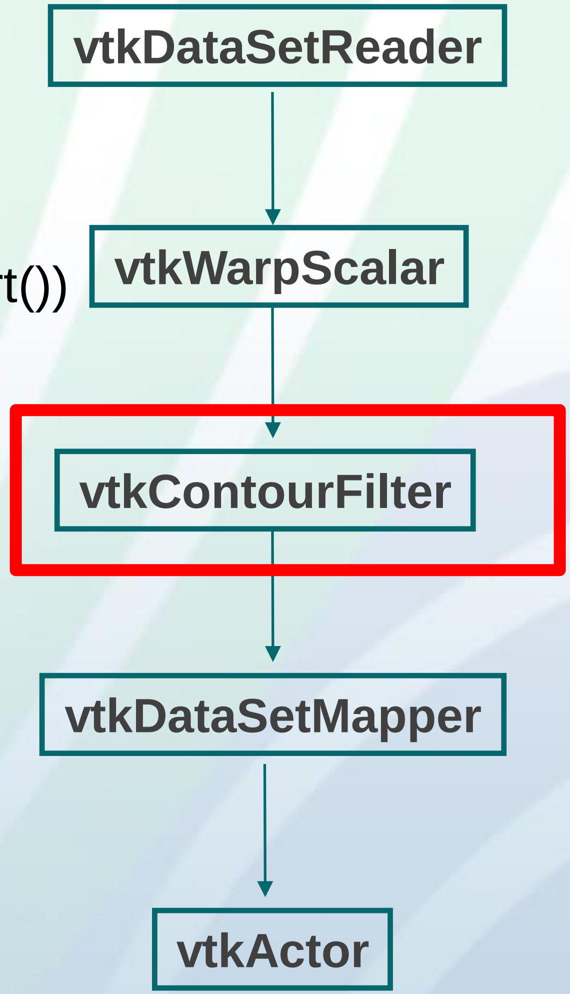
# Step 5 – Drawing iso-surfaces

```
# Draw the iso-surface
isosurface = vtk.vtkContourFilter()
# Generate 10 iso-surfaces along the elevation
isosurface.GenerateValues(10,682.0,2543.0);
# Connect the filter to the reader
isosurface.SetInputConnection(warp.GetOutputPort())
# Update the pipeline
isosurface.Update()

mapper.SetInputConnection(isosurface.GetOutputPort())
renwin.Render()
renderer.ResetCamera()
renwin.Render()
```



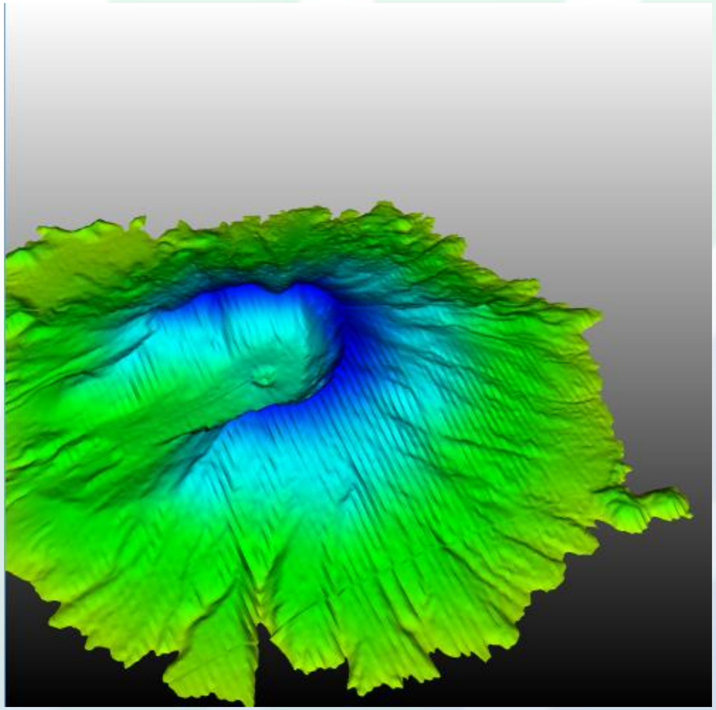
# Step 6 – Exercise - Display iso-surface and data

```
# Render everything into one
mapper.SetInputConnection(warp.GetOutputPort())
# Create a new mapper
mapperiso = vtk.vtkDataSetMapper()
mapperiso.SetInputConnection(isosurface.GetOutputPort())
# Create the actor
actoriso = vtk.vtkActor()
actoriso.SetMapper(mapperiso)
# Disable the scalar coloring
mapperiso.ScalarVisibilityOff()
# Set the color to black
actoriso.GetProperty().SetColor(0,0,0)
# Set the line width larger
actoriso.GetProperty().SetLineWidth(2)
# add the actor to the rendering
renderer.AddViewProp(actoriso)
```



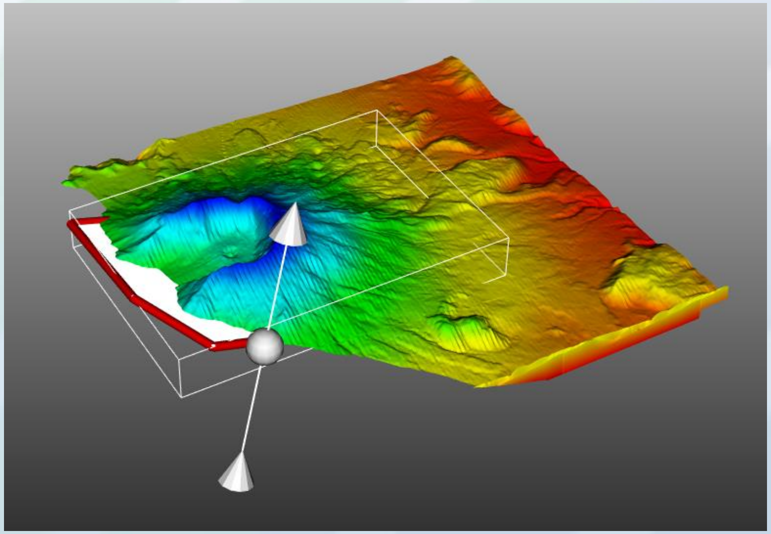
# Step 7 – Clipping the data

```
# Remove first the iso actor
renderer.RemoveViewProp(actoriso)

# Clip the data
cf = vtk.vtkClipDataSet()
# Set the clipping plane
plane = vtk.vtkPlane()
cf.SetClipFunction(plane)
print plane
# Set the plane origin
plane.SetOrigin(560000,5120000,2000)
# Connect the pipeline
cf.SetInputConnection(warp.GetOutputPort())
mapper.SetInputConnection(cf.GetOutputPort())
```



# Step 8 – Clipping Widget interaction

```
# Creates an implicit plane widget
widget = vtk.vtkImplicitPlaneWidget()
widget.PlaceWidget(warp.GetOutput().GetBounds())
widget.SetOrigin([plane.GetOrigin()[x] for x in 0,1,2])
widget.SetNormal([plane.GetNormal()[x] for x in 0,1,2])
widget.SetInteractor(iren)

# Connects the interaction event to the plane
def cb(obj,event):
    global plane
    obj.GetPlane(plane)
widget.AddObserver("InteractionEvent", cb)
widget.SetEnabled(1)
widget.DrawPlaneOn()
widget.TubingOn()
renwin.Render()
```



# VTK and NumPy – Step 1 VTK to NumPy

```
# Example of VTK array to NumPy
import vtk
vtkarray = vtk.vtkDoubleArray()
vtkarray.InsertNextValue(1)
vtkarray.InsertNextValue(2)
from vtk.util.numpy_support import vtk_to_numpy
numpyarray = vtk_to_numpy(vtkarray)
```

# VTK and NumPy – Step 2 NumPy to VTK

```
# Example of NumPy to VTK array
import vtk
from vtk.util.numpy_support import numpy_to_vtk
import numpy
numpyarray = numpy.zeros(5)
vtkarray = numpy_to_vtk(numpyarray)
```

# Improved VTK - NumPy Integration

```
import vtk
from vtk.numpy_interface import dataset_adapter as dsa
from vtk.numpy_interface import algorithms as algs
reader = vtk.vtkMPASReader()
reader.SetFileName('MPASReader.nc')
# Have reader examine what information is available
reader.UpdateInformation()
# Print out available arrays
for i in range(reader.GetNumberOfCellArrays()):
    print reader.GetCellArrayName(i)
# Don't read in the ke cell data array
reader.SetCellArrayStatus('ke', 0)
reader.Update()
```

# Improved VTK - NumPy Integration (continued)

```
# Wrap the reader output to make simplify access
wrappedreader = dsa.WrapDataObject(reader.GetOutput())
print wrappedreader.PointData.keys()
print wrappedreader.CellData['vorticity']
vorticity = wrappedreader.CellData['vorticity']
# Perform some operations on the data
wrappedreader.CellData.append(vorticity + 1, 'vorticity plus one')
print algs.max(vorticity)
print algs.max(wrappedreader.CellData['vorticity plus one'])
```

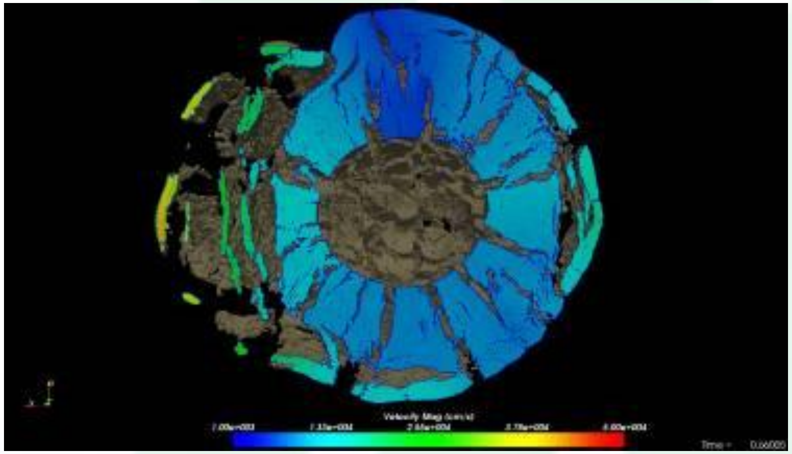
# Grid Aware Algorithms

```
import vtk
from vtk.numpy_interface import dataset_adapter as dsa
from vtk.numpy_interface import algorithms as algs
# Create a dataset
w = vtk.vtkRTAnalyticSource()
t = vtk.vtkDataSetTriangleFilter()
t.SetInputConnection(w.GetOutputPort())
t.Update()
# Compute gradient of RTData array
ugrid = dsa.WrapDataObject(t.GetOutput())
print algs.gradient(ugrid.PointData['RTData'])
```

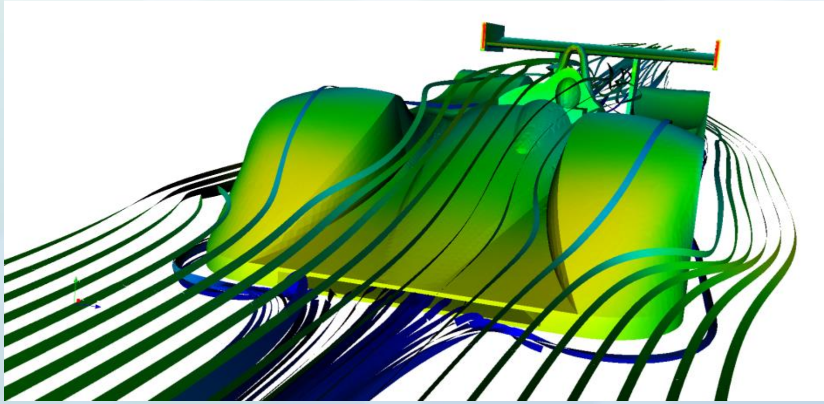
# PARAVIEW

# ParaView

- www.paraview.org
- OpenSource (BSD)
- Based on VTK
- C++/Qt
- Python support
- Very active community (HPC wire award)
- Multi-core support (MPI)
- Co-Processing (in-situ)
- More than 50 data readers

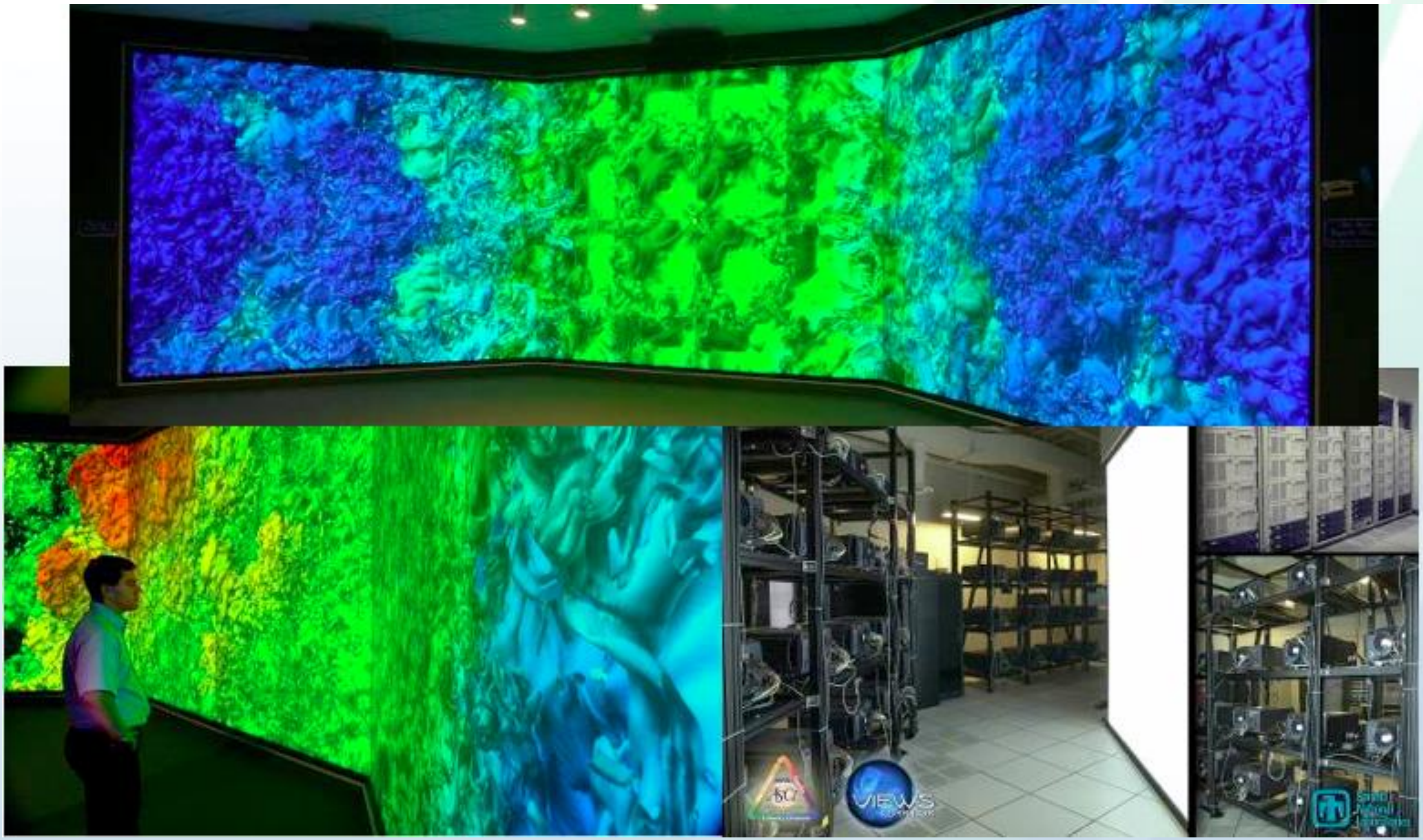


1 billion cell asteroid detonation simulation

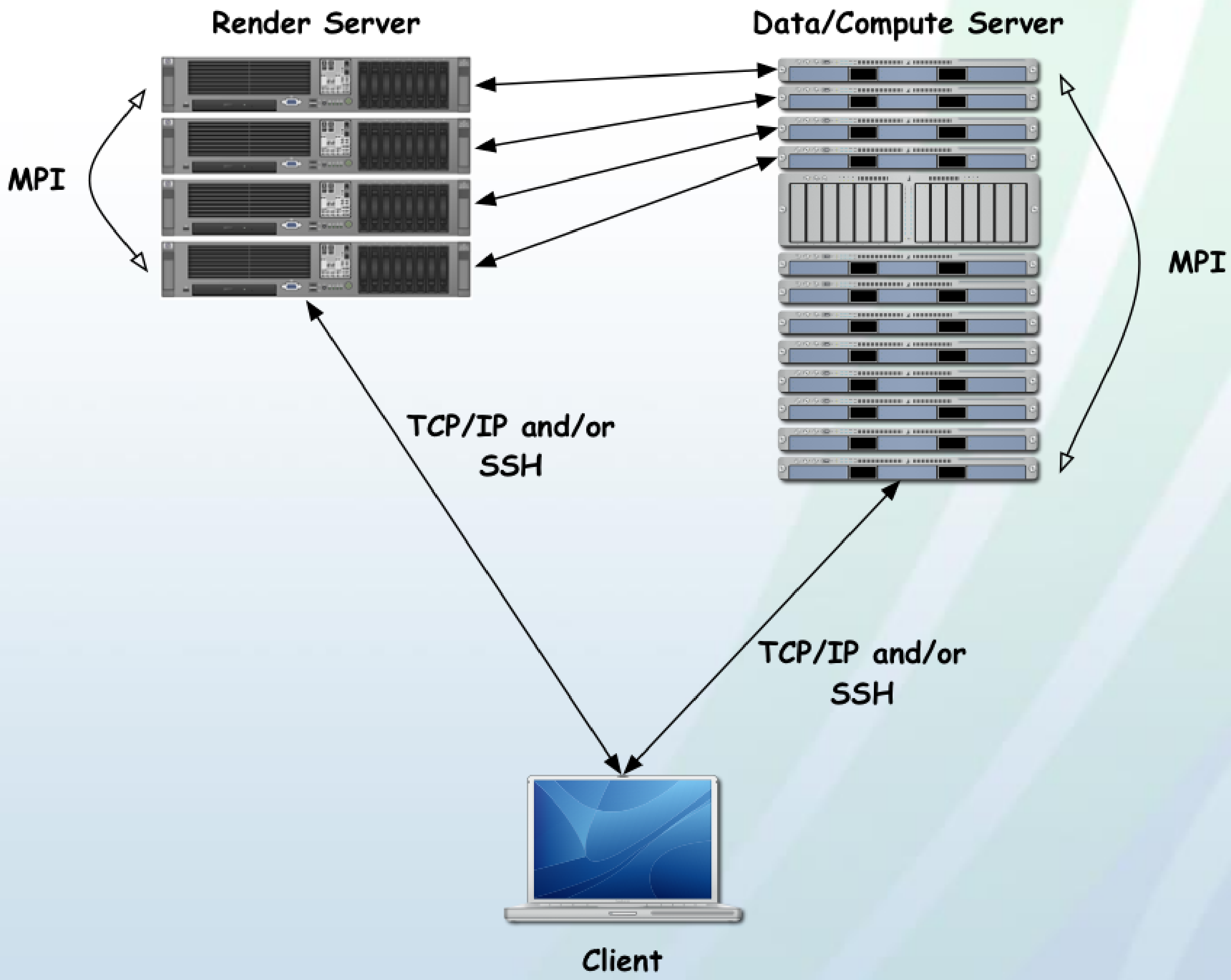


Kitware

# ParaView in Use – Immersive Visualization



# ParaView Architecture




Kitware

# Why Use ParaView with Python

- Client-server architecture
- Easy to use in parallel
  - Image compositing & rendering
  - Avoid common VTK mistakes (e.g. parallel XML formats need process 0 to have field data arrays for meta-data file)
- More filters than in VTK
- Meta-data instead of data
  - Bounds, ranges, sizes
- GUI shortcuts
  - Trace option to mirror GUI operations in Python
  - GUI widgets to execute Python scripts

# Creating a simple visualization

- Create a cone

```
>>> myCone = Cone()
```

- Get documentation on cone and its properties

```
>>> help(myCone)
```

- Examine a property

```
>>> myCone.Center
```

- Change a property

```
>>> myCone.Center = [0,0,1]
```

- Show the result

```
>>> Show(myCone)
>>> Render()
```

# Creating a simple visualization

- Apply a filter

```
>>> myClip = Clip()
```

- Show the result

```
>>> Show(myClip)
>>> Render()
```

- Forgot to hide the cone

```
>>> Hide(myCone)
>>> Render()
```

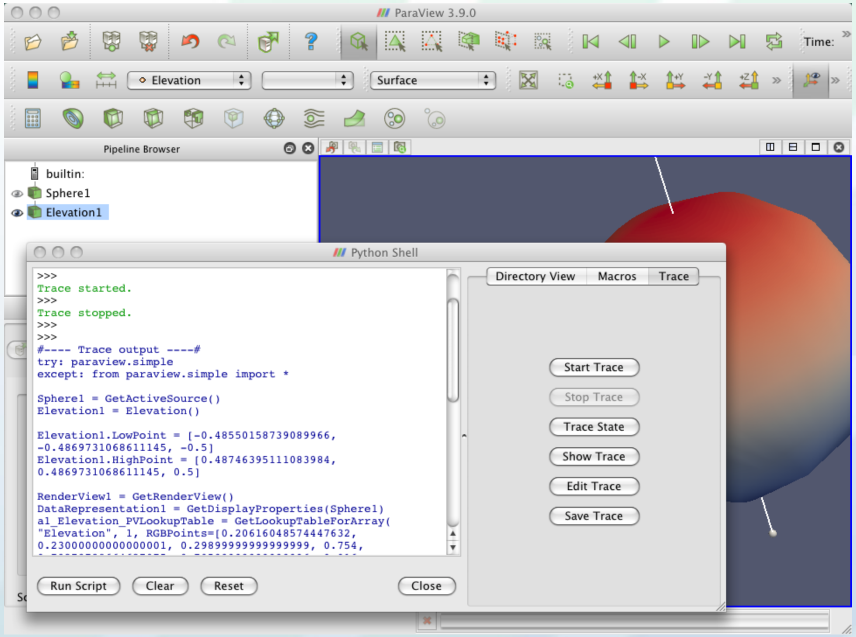
- Change camera

```
>>> cam = GetActiveCamera()
>>> cam.GetPosition()
>>> cam.SetPosition(-1,0,2.5)
>>> Render()
```

Kitware

# Trace – record python script from GUI

- Tools → Start Trace
  - Starts recording GUI actions
- Tools → Stop Trace
  - Finishes recording
  - Brings up script editor
- File → Save
  - Writes script to file
- File → Save as Macro…
  - Saves script and adds button to menu that calls it
- Macros
  - Execute and manage macros you've created



# More information

- VTK
  - http://www.vtk.org
  - Documentation: http://www.vtk.org/doc/nightly/html/
  - Examples in Python: http://www.vtk.org/Wiki/VTK/Examples/Python

- ParaView
  - http://www.paraview.org
  - ParaView Python Scripting: http://www.paraview.org/Wiki/ParaView/Python_Scripting

# NetCDF with ParaView

- Run pvpython

```
>>> from paraview.simple import *
>>> reader = NetCDFMPASreader(FileName='MPASReader.nc')
>>> # Get information about what's in the file
>>> reader.UpdatePipelineInformation()
>>> print reader.PointArrayStatus
>>> print reader.TimestepValues
>>> # Reader in only some of the arrays
>>> reader.PointArrayStatus = ['latCell', 'lonCell', 'xCell', 'nEdgesOnCell', 'areaCell']
>>> # Read in the requested data
>>> reader.UpdatePipeline()
>>> # Change time steps
>>> animation = GetAnimationScene()
>>> animation.GoToLast()
>>> animation.GoToPrevious()
```

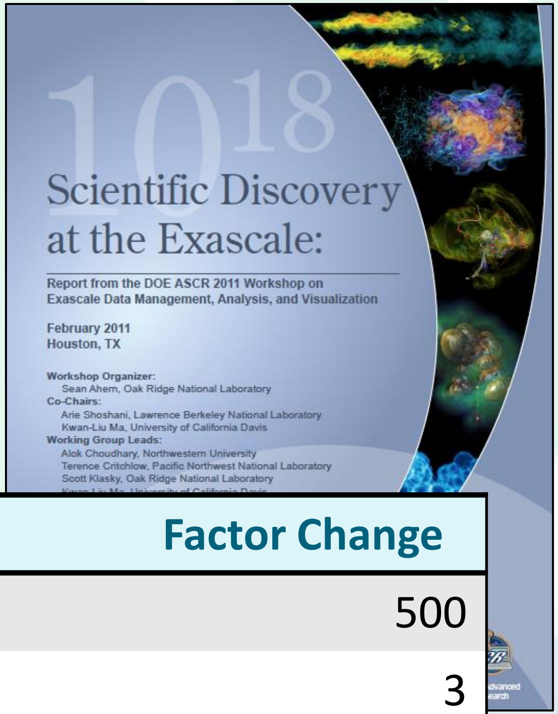
# Client-Server

- First run pvbatch, use mpirun for parallel server

- Run pvpython

```
>>> from paraview.simple import *
>>> Connect("localhost")
>>> s = Sphere()
>>> pid = ProcessIdScalars()
>>> Show()
>>> Render()
```

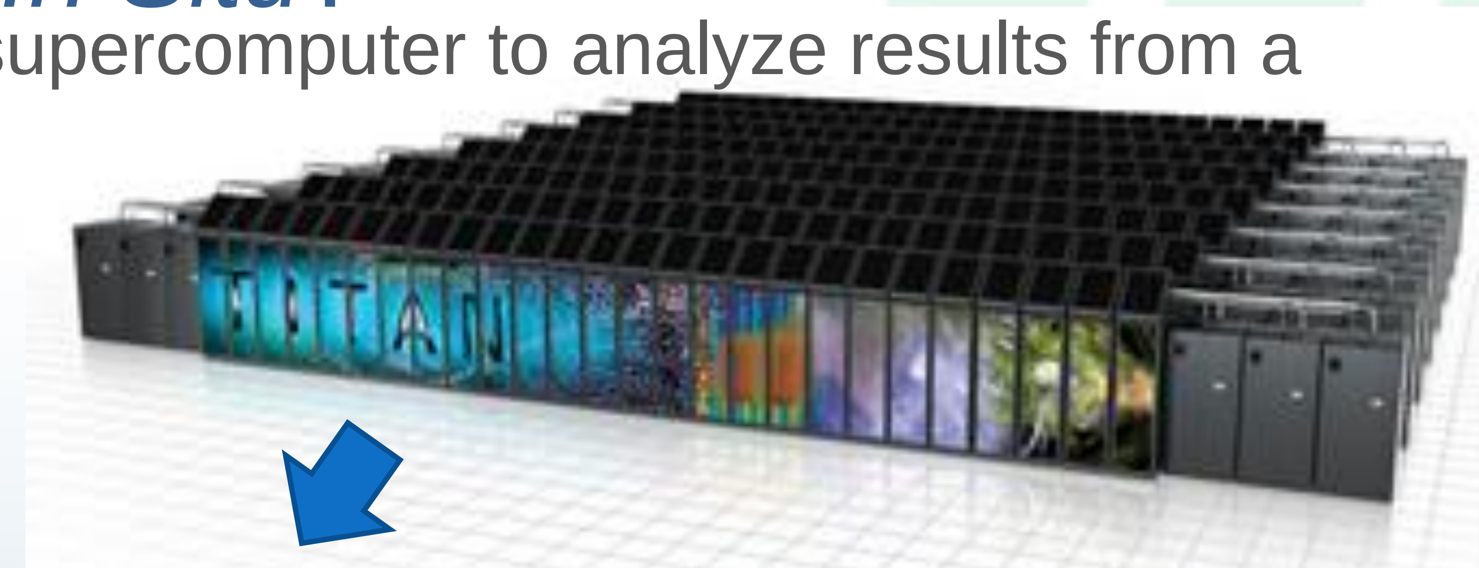
# Why *In Situ*?




| System Parameter | 2011 | "2018" | | Factor Change |
|---|---|---|---|---|
| System peak | 2 PF | 1 EF | | 500 |
| Power | 6 MW | ≤20 MW | | 3 |
| System Memory | 0.3 PB | 32-64 PB | | 33 |
| Node Performance | 0.125 TF | 1 TF | 10 TF | 8-80 |
| Node Concurrency | 12 | 1,000 | 10,000 | 83-830 |
| Network BW | 1.5 GB/s | 100 GB/s | 1,000 GB/s | 66-660 |
| System Size (nodes) | 18,700 | 1M | 100k | 50 |
| Total Concurrency | 225 K | 10 B | 100 B | 40k-400k |
| Storage Capacity | 15 PB | 300-1,000 PB | | 20-67 |
| I/O BW | 0.2 TB/s | 20-60 TB/s | | 100-300 |

# Why *In Situ*?

Need a supercomputer to analyze results from a hero run

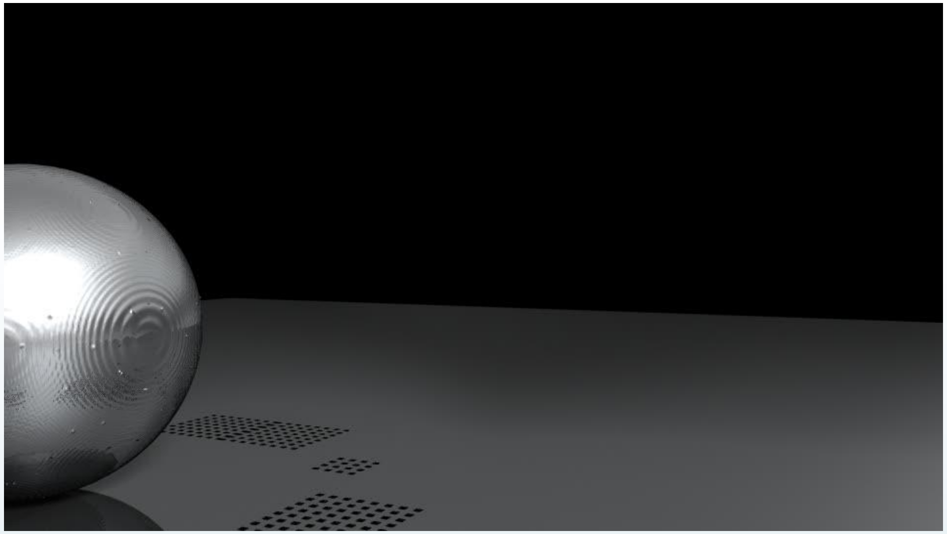


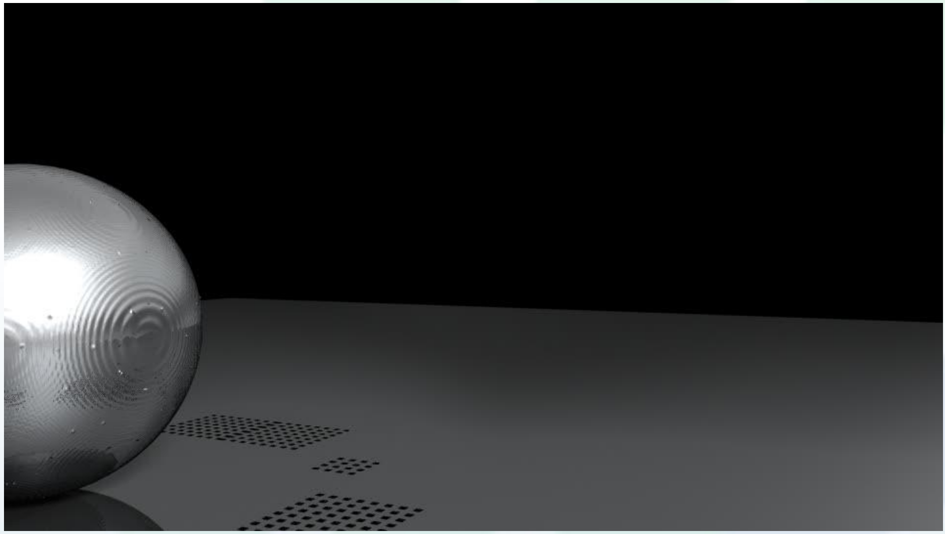

2 orders of magnitude difference between each level

# ParaView Catalyst

- Goals:
  - Provide flexible analysis options in addition to traditional 'post-processing' options
    - Integrated workflow with widely used tool set
    - Old options still available
  - Increase the value of data saved to permanent storage
    - Increased fidelity in time/space
  - Create framework for sampling and visualization R&D
    - Large scale sampling
    - Novel visualization options (ParaView Cinema, etc.)
  - Optimize I/O
    - Will continue to be under heavy constraints …

# Access to More Data





Dump Times

Post-processing

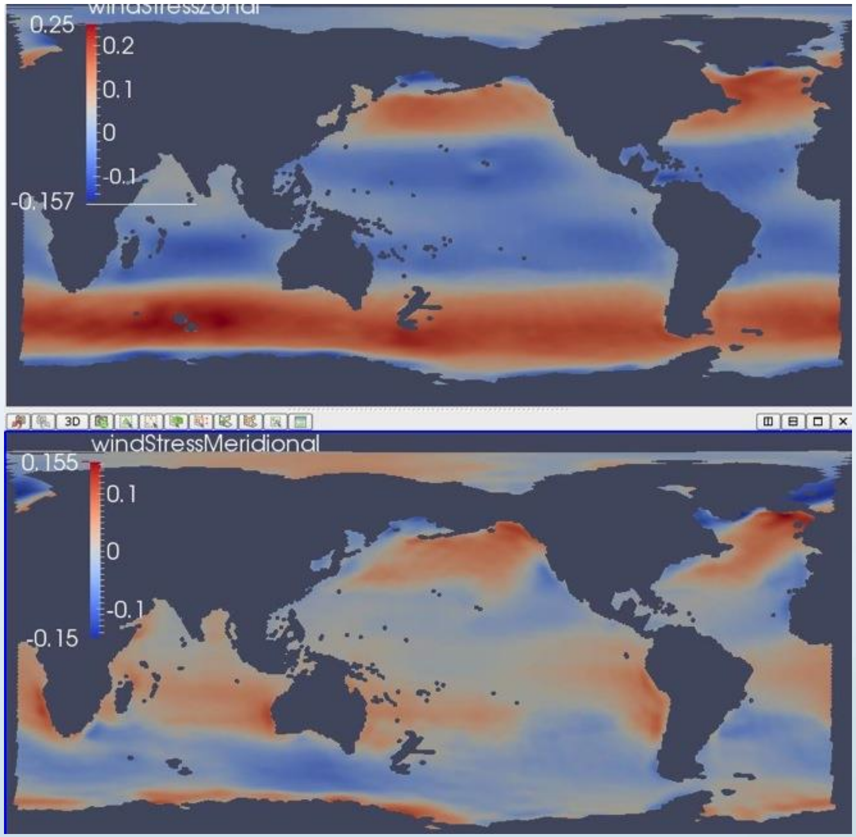
*In situ* processing

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

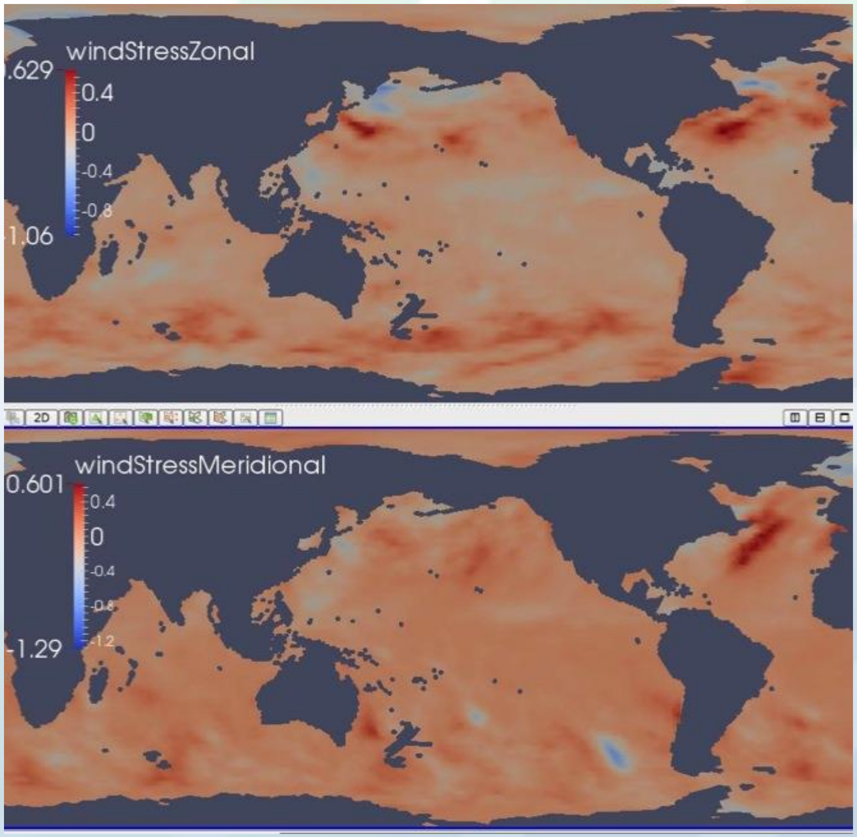
*Reflections and shadows added in post-processing for both examples*

# Quick and Easy Run-Time Checks

Expected wind stress field at the surface of the ocean

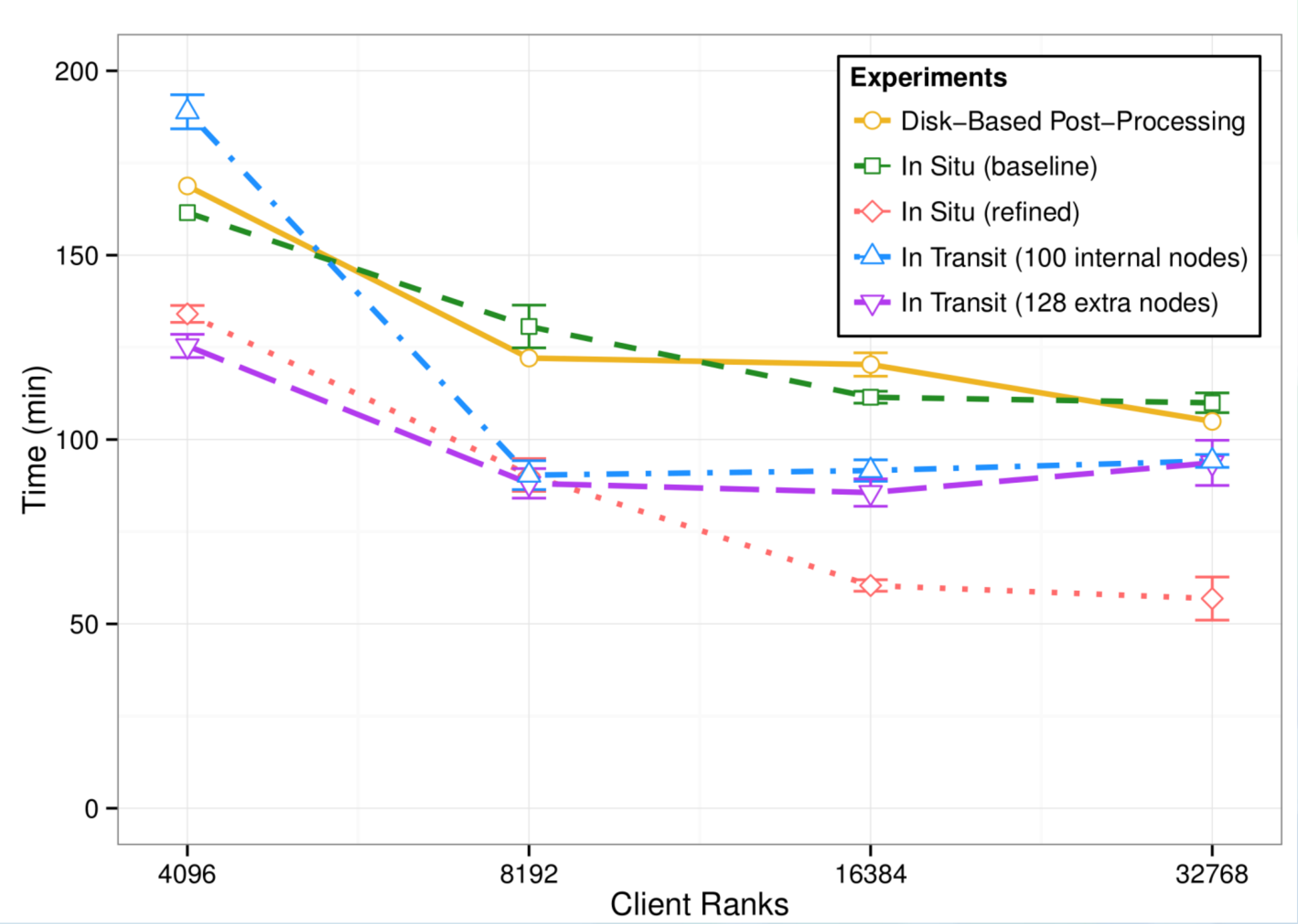



Wind stress in new run, quick glance indicates we are using wrong wind stress




MPAS-O (LANL) simulation

# Faster Time to Solution

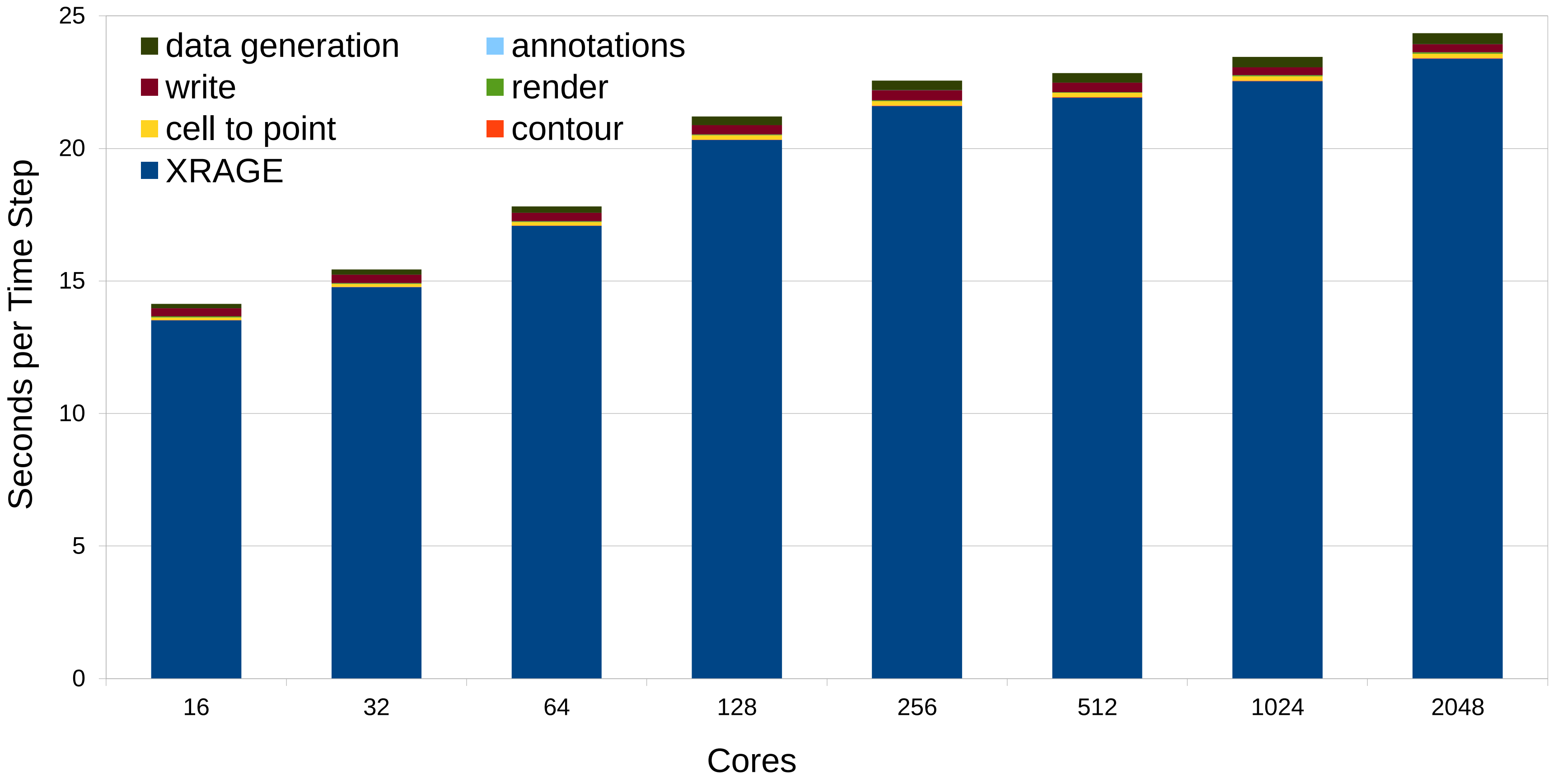



CTH (Sandia) simulations comparing different workflows



Kitware

# Small Run-Time Overhead




XRAGE (LANL) simulation

# Reduced File IO Costs

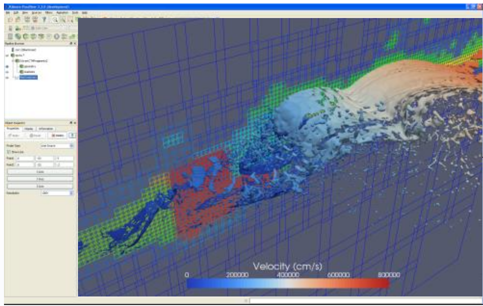
| Time of Processing | Type of File | Size per File | Size per 1000 time steps | Time per File to Write at Simulation |
|---|---|---|---|---|
| Post | Restart | 1,300 MB | 1,300,000 MB | 1-20 seconds |
| Post | Ensight Dump | 200 MB | 200,000 MB | > 10 seconds |
| *In Situ* | PNG | .25 MB | 250 MB | < 1 second |

XRAGE (LANL) simulation



Kitware

# 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

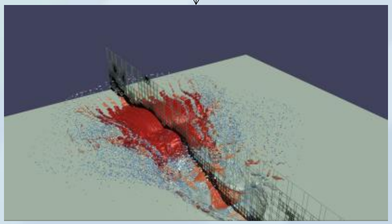
Catalyst

Output Processed Data

Statistics

Series Data

Polygonal Output with Field Data

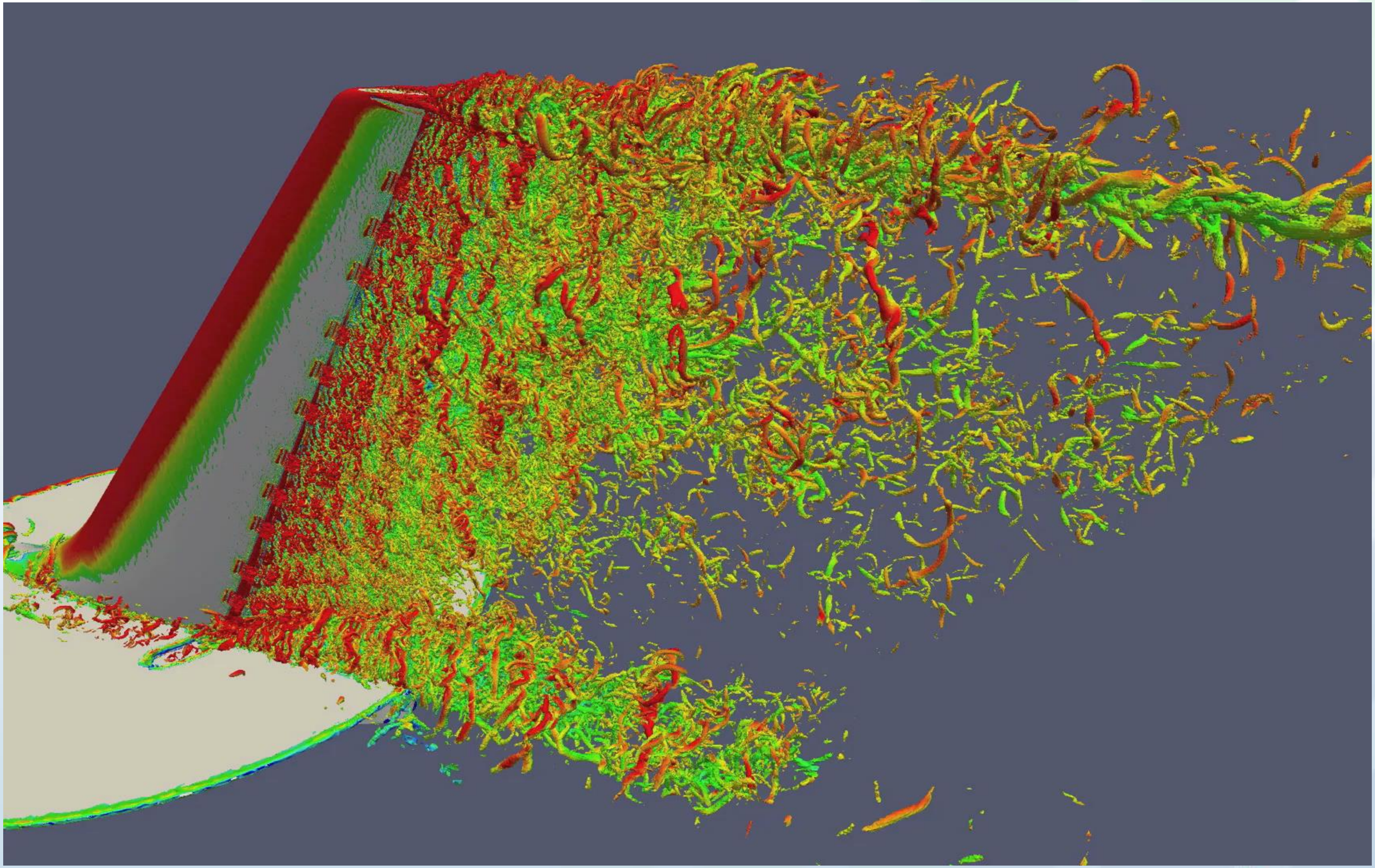


Rendered Images

# ParaView Catalyst with PHASTA

- Parallel Hierarchic Adaptive Stabilized Transient Analysis (PHASTA)
  - Joint development at UC Boulder, RPI, Argonne
  - Fully-implicit, stabilized, semi-discrete finite element method for the transient, incompressible Navier-Stokes partial differential equation (PDE)
- 1.3 billion element unstructured mesh
- 256K MPI processes run on 128K cores
  - Argonne BG/L (Mira)
  - Using 2 of 4 hardware threads per core
- Python driven output from Catalyst

# Flow Visualization: Full Span and Wake



**Credit: Michel Rasquin (ANL) and Ken Jansen (UC Boulder)**

Kitware

# Thank You!

Andrew Bauer
andy.bauer@kitware.com