Kitware

ParaView Catalyst:
Scalable *In Situ* Processing

David Rogers (LANL), Jeffrey Mauldin (Sandia),
Thierry Carrard (CEA), Andrew Bauer (Kitware)

November 17th, 2014

Agenda

- Introduction to ParaView Catalyst
- Catalyst for Users
  - New Functionalities in ParaView 4.2
- Catalyst for Developers

Goals

- Understand the flow of control and data when using ParaView Catalyst with a simulation code
  - Understand efficiency trade-offs
- Hands-on exercises for:
  - Creating a Python script to extract data and screenshots through ParaView Catalyst
  - Interfacing a simulation code with Catalyst
  - Creating VTK data structures to represent grid and field information from simulation data structures
- Running a simulation with all of the parts put together

Online Materials

- This presentation and VM virtual appliance at:
  - www.paraview.org/Wiki/ParaView/Catalyst/SC14_Tutorial/
  - SC14_Tutorial.ova
  - SC14_Tutorial.pdf
- Install VirtualBox
  - www.virtualbox.org
- Install ParaView 4.2 (in VM virtual appliance)
  - www.paraview.org/download

VirtualBox

- Run VirtualBox and go to “File→Import Appliance…”
- Click on “Open appliance…” and select “SC14_Tutorial.ova”, click on “Next >”
- Can adjust appliance settings as needed and click on “Import”
  - Suggest keeping defaults
- Click on “Start” to run VirtualBox with ParaView Catalyst tutorial examples
  - Can click on “OK” for warning messages

VirtualBox

- Use catuser account
  - Password is “cat2014”
  - Account has sudo privileges for installing your own development environment
- ParaView executables in default PATH
- ParaView OSMesa build is at /home/catuser/ParaView4.2/build_mesa

Online Help

- ParaView Catalyst User's Guide:
  - http://paraview.org/Wiki/images/4/48/CatalystUsersGuide.pdf
- Email list:
  - paraview@paraview.org
- Doxygen:
  - http://www.vtk.org/doc/nightly/html/classes.html
  - http://www.paraview.org/ParaView3/Doc/Nightly/html/classes.html
- Sphinx:
  - http://www.paraview.org/ParaView3/Doc/Nightly/www/py-doc/index.html
- Websites:
  - http://www.paraview.org
  - http://www.paraview.org/in-situ/
- Examples:
  - https://github.com/Kitware/ParaViewCatalystExampleCode

What is ParaView Catalyst?

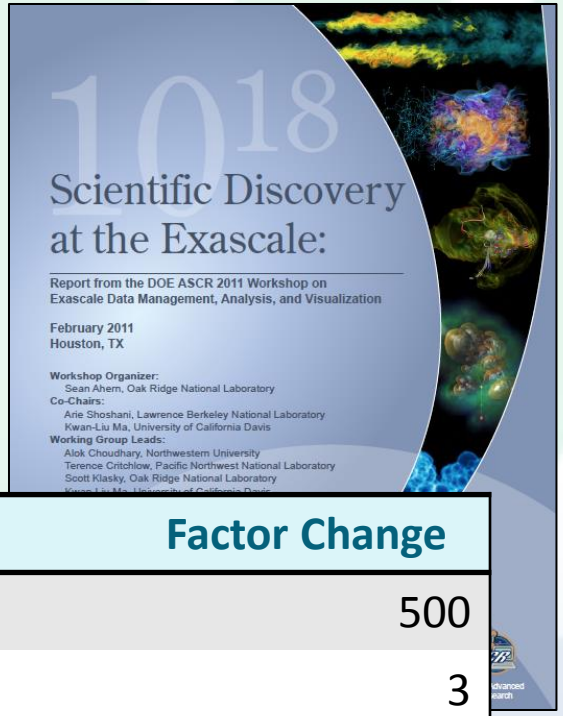
What is ParaView Catalyst?

- A set of *in situ* data analysis and visualization capabilities developed in response to current and near future data analysis challenges
  - Available today, open source
- Challenges
  - There is an explosion of scientific data
  - “data analysis infrastructure and storage system bandwidth have not scaled proportionately to processing power”

ParaView Catalyst: Goals

- 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 …

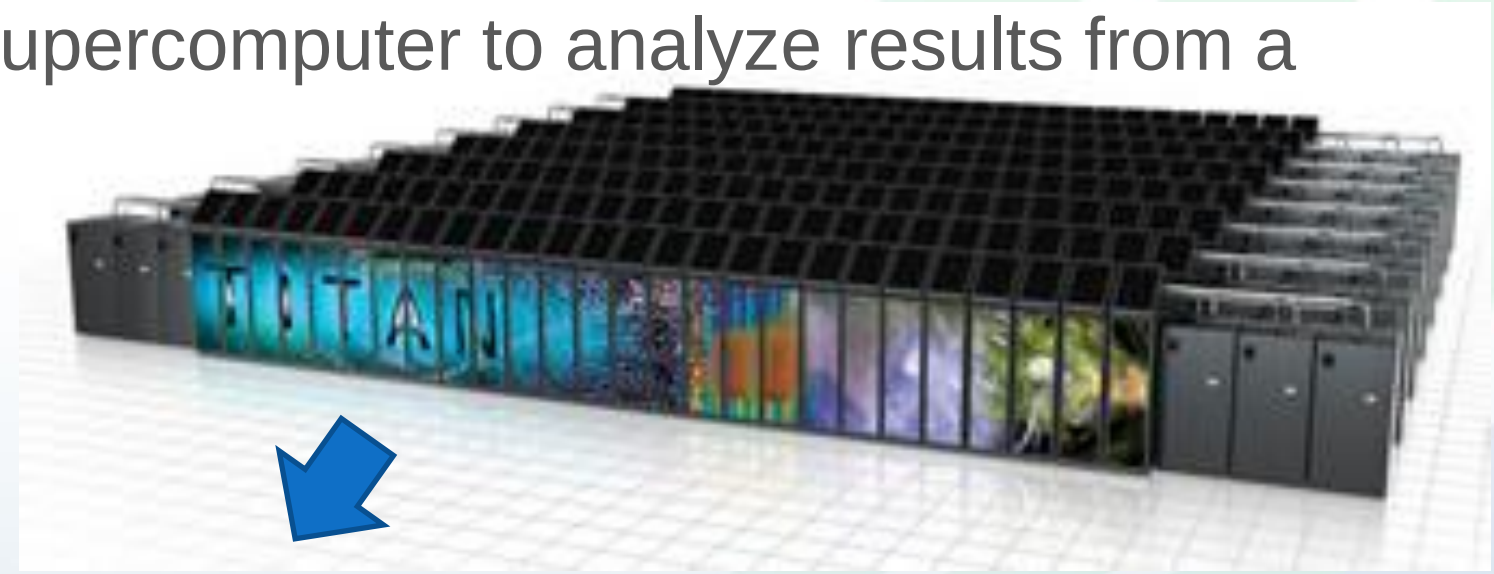
Why *In Situ*?



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*?

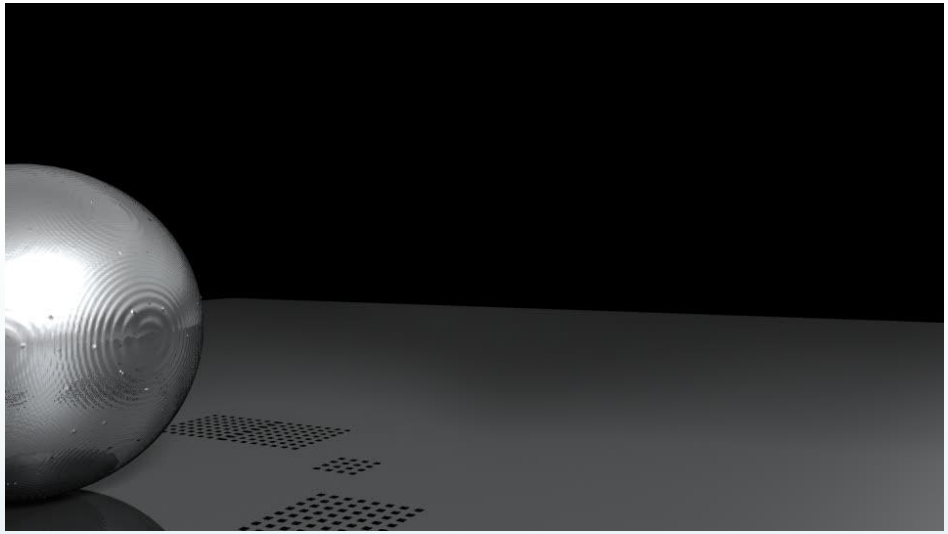
Need a supercomputer to analyze results from a hero run

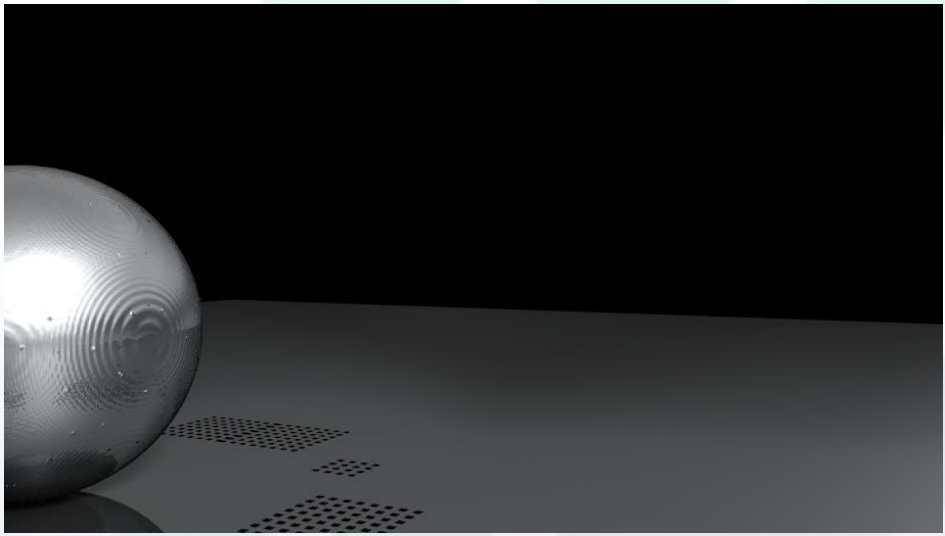


RHEA

2 orders of magnitude difference between each level

Access to More Data





Post-processing

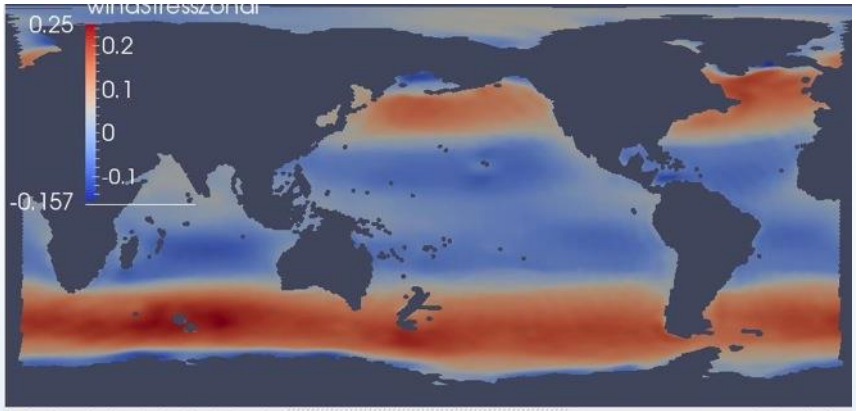
In situ processing

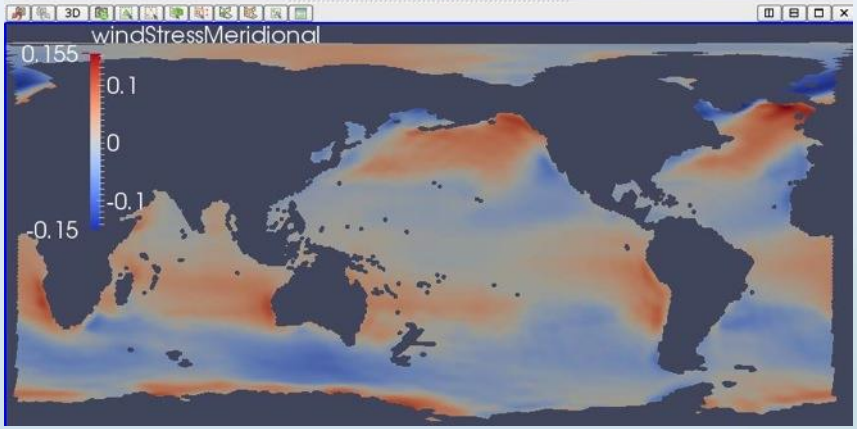
CTH (Sandia) simulation with roughly equal data stored at simulation time

Reflections and shadows added in post-processing for both examples

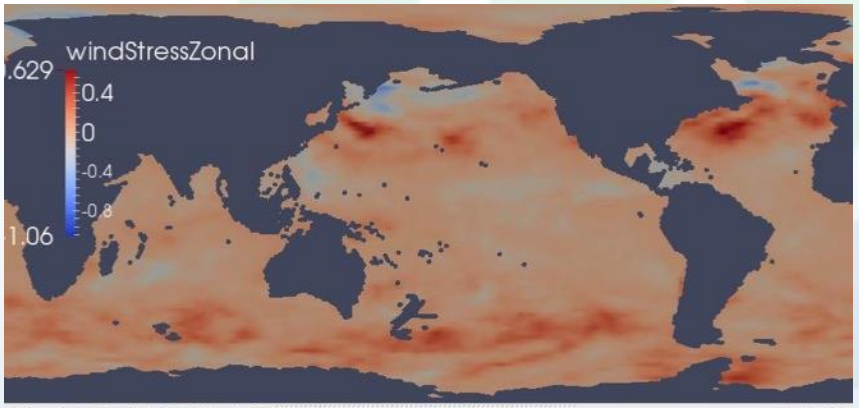
Quick and Easy Run-Time Checks

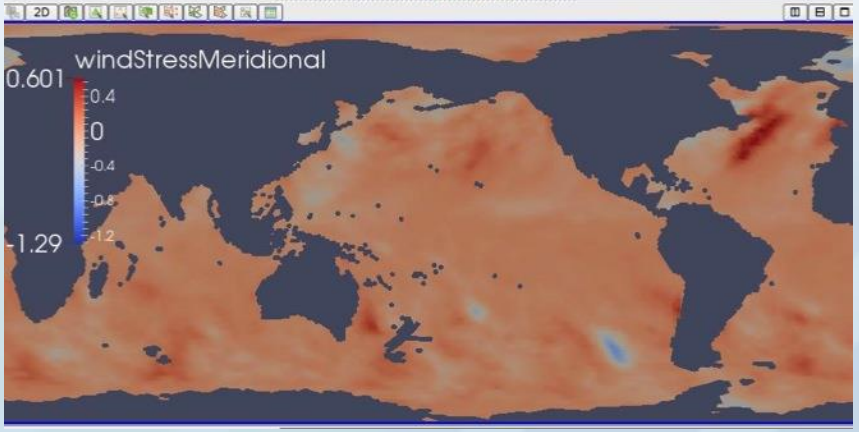
Expected wind stress field at the surface of the ocean





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

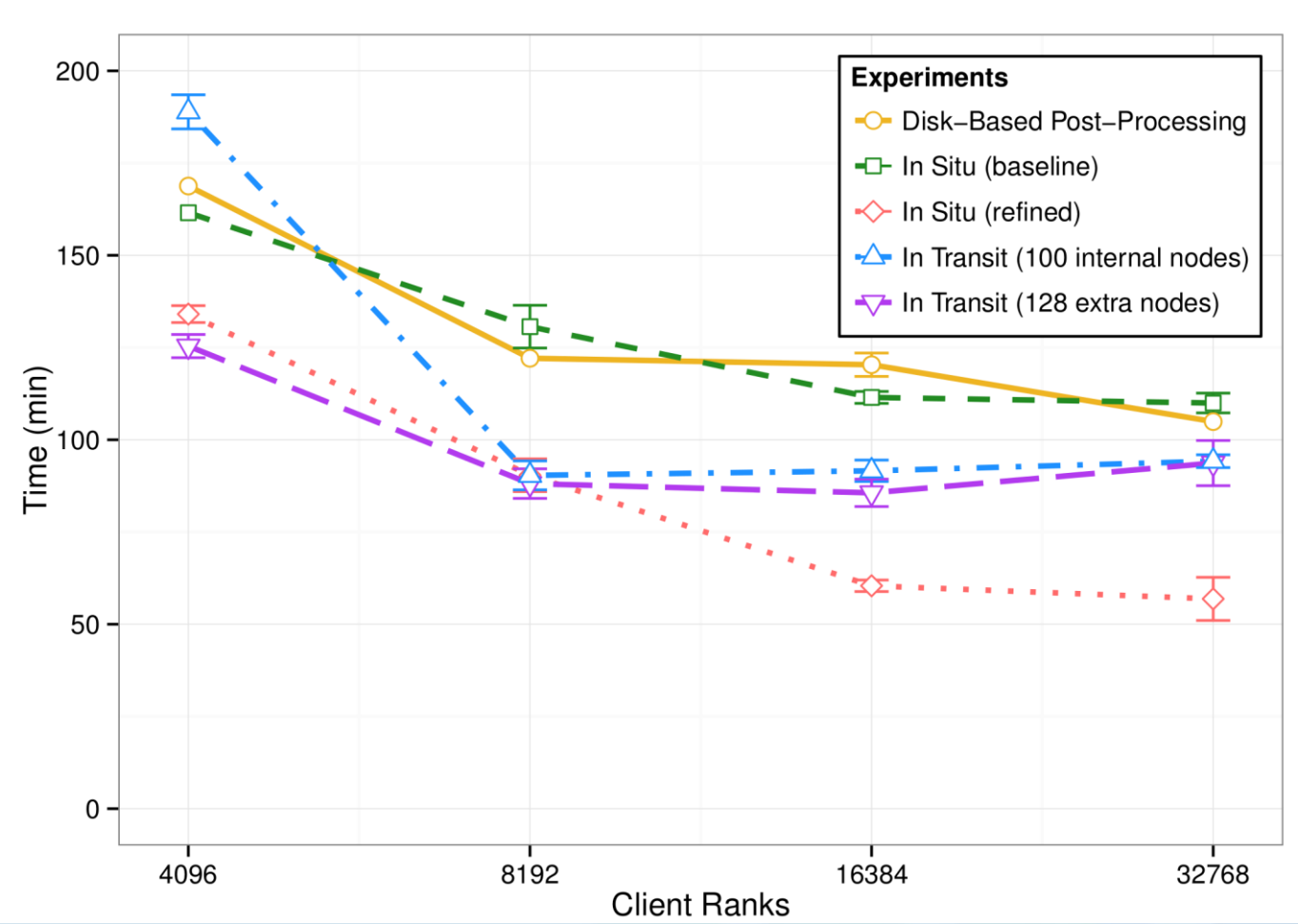




MPAS-O (LANL) simulation

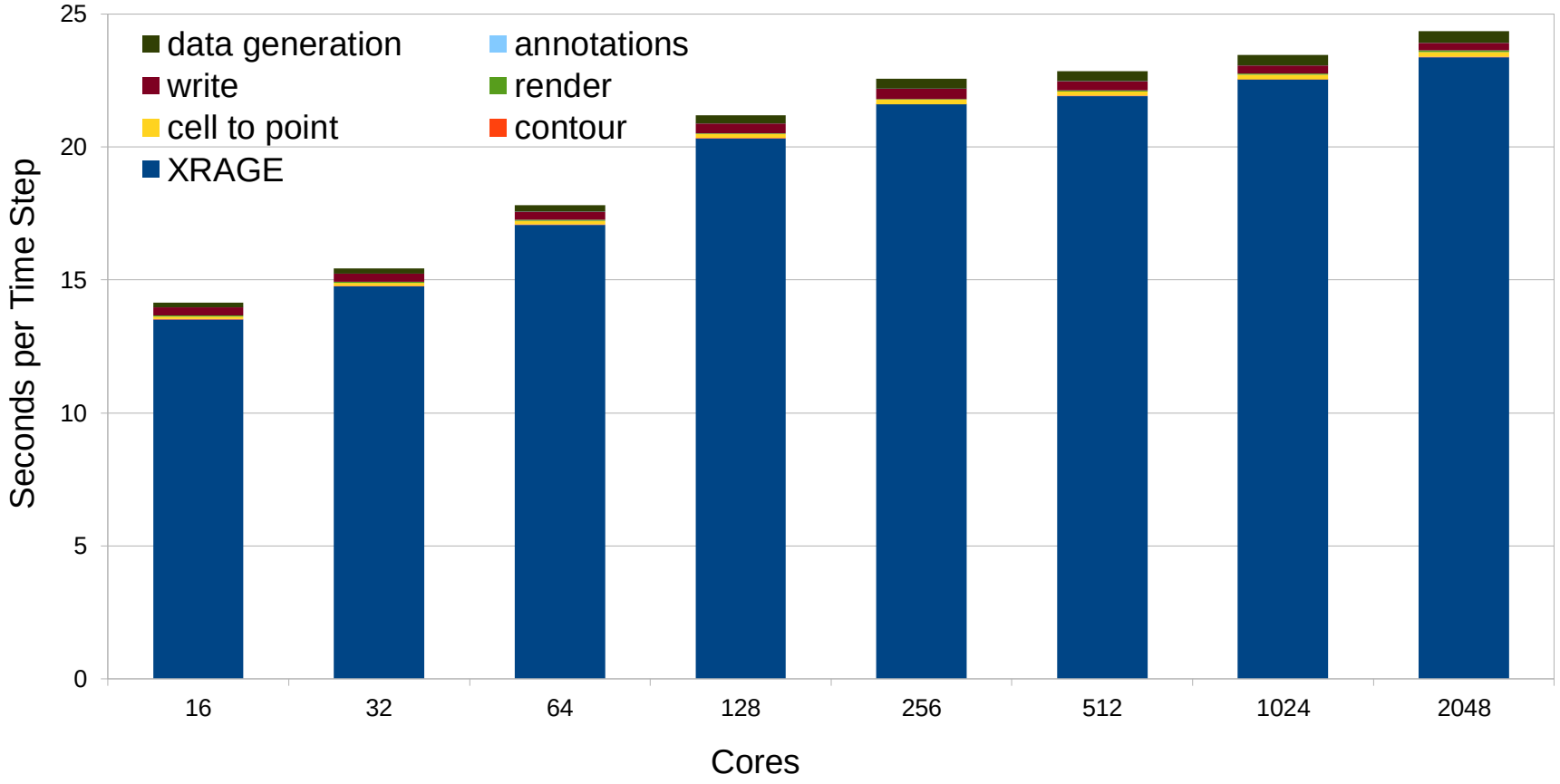
Kitware

Faster Time to Solution



CTH (Sandia) simulations comparing different workflows

Small Run-Time Overhead

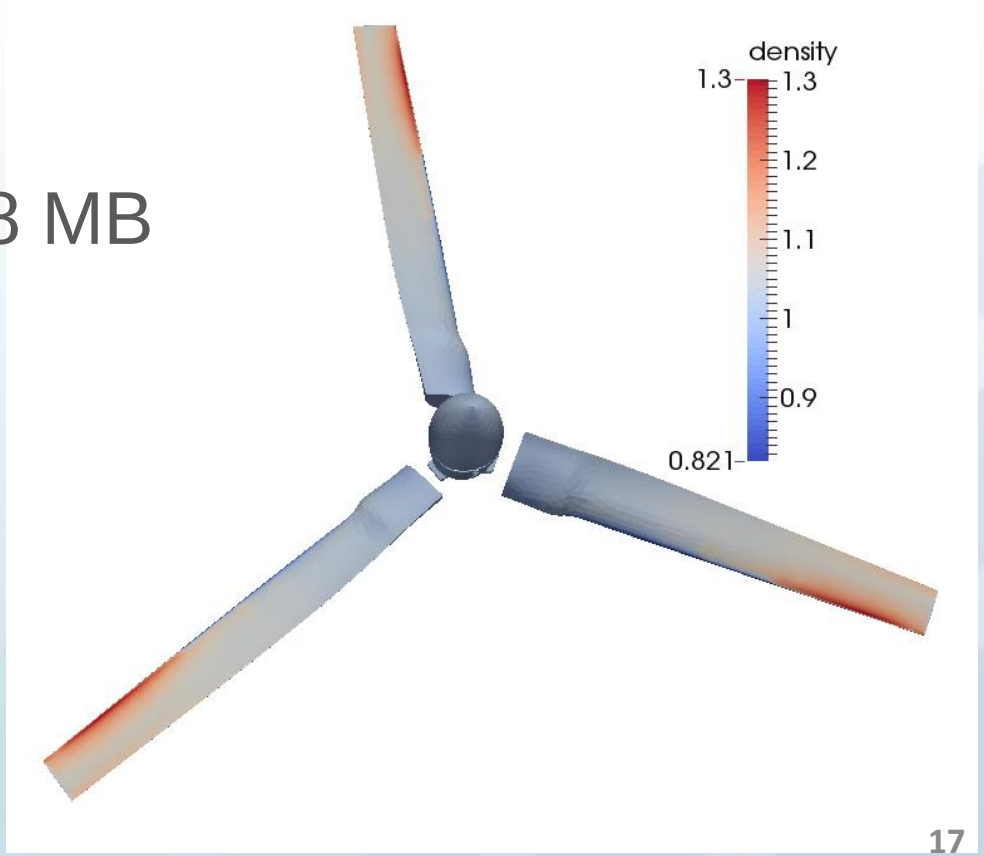


XRAGE (LANL) simulation

Reduced File Size

Rotorcraft simulation output size for a single time step

- Full data set – 448 MB
- Surface of blades – 2.8 MB
- Image – 71 KB



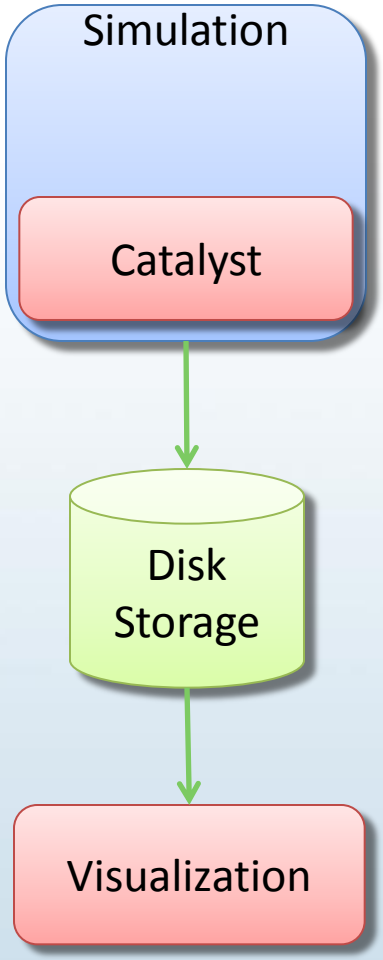
Helios (Army AFDD/AMRDEC) simulation

Reduced File IO Costs

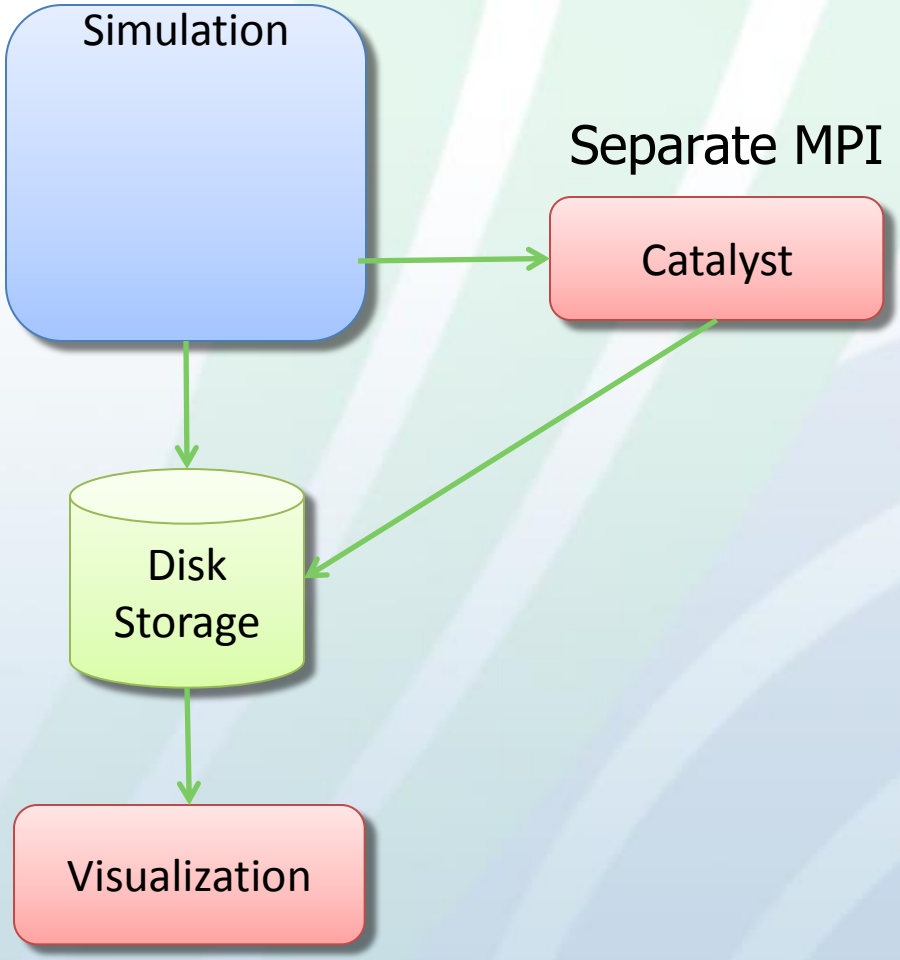
Time of Processing	Type of File	Size per File	Size per 1000 time steps	Time per File to Write at Simulation
Post	Restart	1,300 MB	1,300,000 MB	1-20 seconds
Post	Ensight Dump	200 MB	200,000 MB	> 10 seconds
In Situ	PNG	.25 MB	250 MB	< 1 second

XRAGE (LANL) simulation

Two Ways to Run in Batch



In Situ

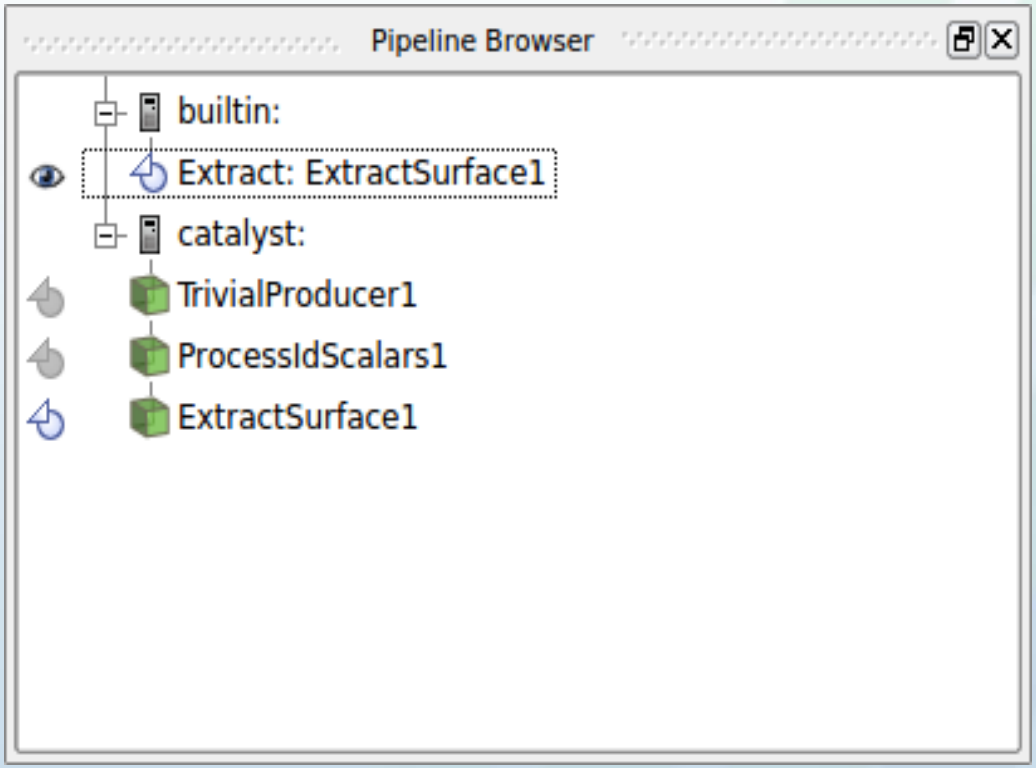


In Transit

Kitware

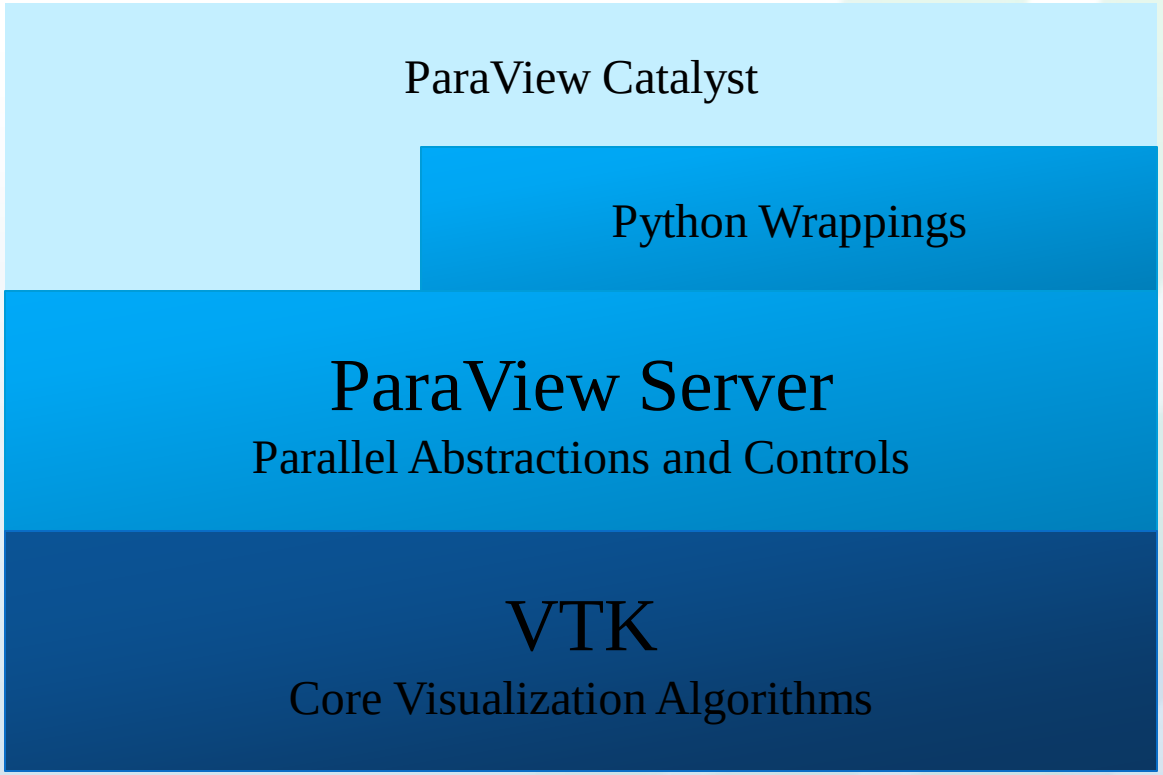
Run Interactively

- Interactive to see current results from simulation
- Access to pipeline on ParaView Catalyst server and client



Kitware

ParaView Catalyst Architecture



High Level View

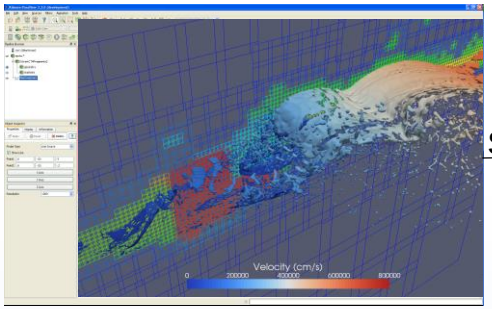
Simulation Users

- Knowledge of ParaView as a post-processing/analysis tool
  - Basic interaction with GUI Catalyst script generator plugin
  - Incremental knowledge increase to use the *in situ* tools from basic ParaView use
- Programming knowledge can be useful to extend the tools

Simulation Developers

- Pass necessary simulation data to ParaView Catalyst
- Need sufficient knowledge of both codes
  - VTK for grids and field data
  - ParaView Catalyst libraries
- Transparent to simulation users
- Extensible

User Perspective



Script Export

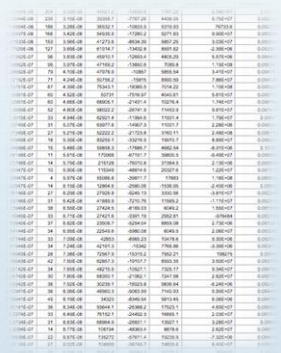
```
# Create the reader and set the filename.
reader = servermanager.sources.Reader(FileNames=path)
view = servermanager.CreateRenderView()
repr = servermanager.CreateRepresentation(reader,  view)
reader.UpdatePipeline()
dataInfo = reader.GetDataInformation()
pDinfo = dataInfo.GetPointDataInformation()
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

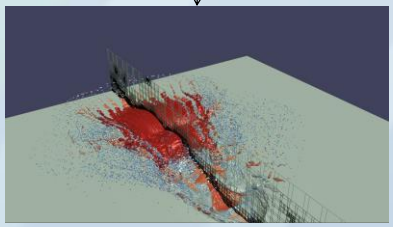
Output Processed Data



Statistics

Series Data

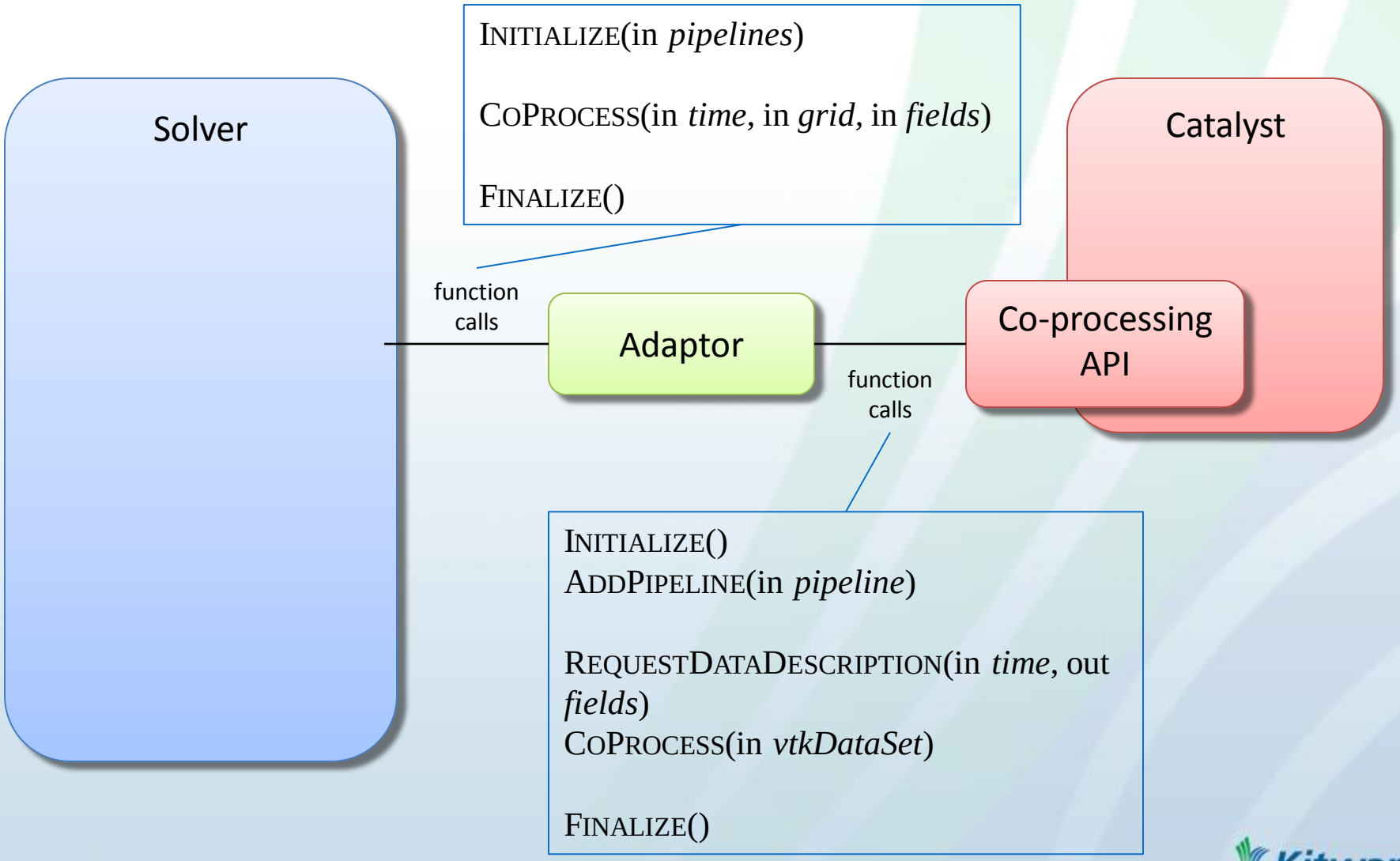
Polygonal Output with Field Data



Rendered Images

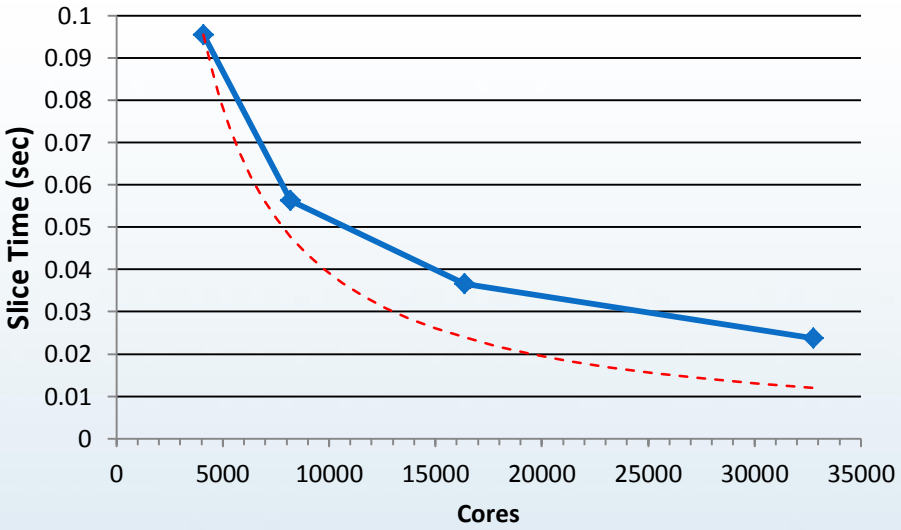
Kitware

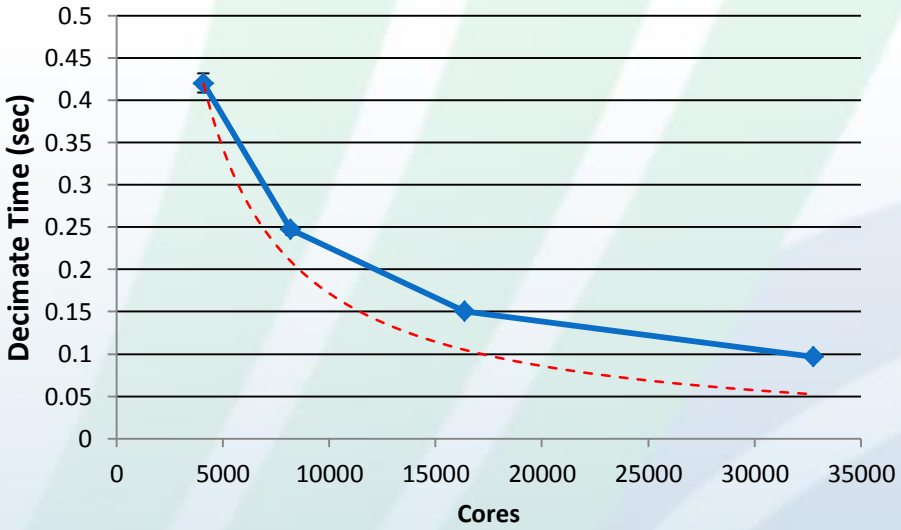
Developer Perspective



Kitware

User & Developer Perspective





Kitware

ParaView Catalyst for Simulation Users

Creating Catalyst Output

Two main ways:

- Create and/or modify Python scripts
  - ParaView GUI plugin to create Python scripts
  - Modification with knowledge of ParaView Python API
- Developer generated “canned” scripts
  - User provides parameters for already created Catalyst pipelines
  - User may not even need to know ParaView
  - See ParaView Catalyst User’s Guide

Create Python Scripts from ParaView

- Use the Catalyst script generator GUI plugin
- Similar to using ParaView GUI interactively
  - Setup desired pipelines
  - Ideally, start with a representative data set from the simulation
- Extra pipeline information to tell what to output during simulation run
  - Add in data extract writers
  - Create screenshots to output
  - Both require file name and write frequency

In Situ Demo

- Create a ParaView Catalyst Python pipeline script
  - Specify desired outputs from run
  - Export the script
- In the ~/miniFE-2.0_ref_Catalyst/src directory, run the script with miniFE
  - ./miniFE.x script_names=<file name>
- Examine results

Use Catalyst to Write Out Grid

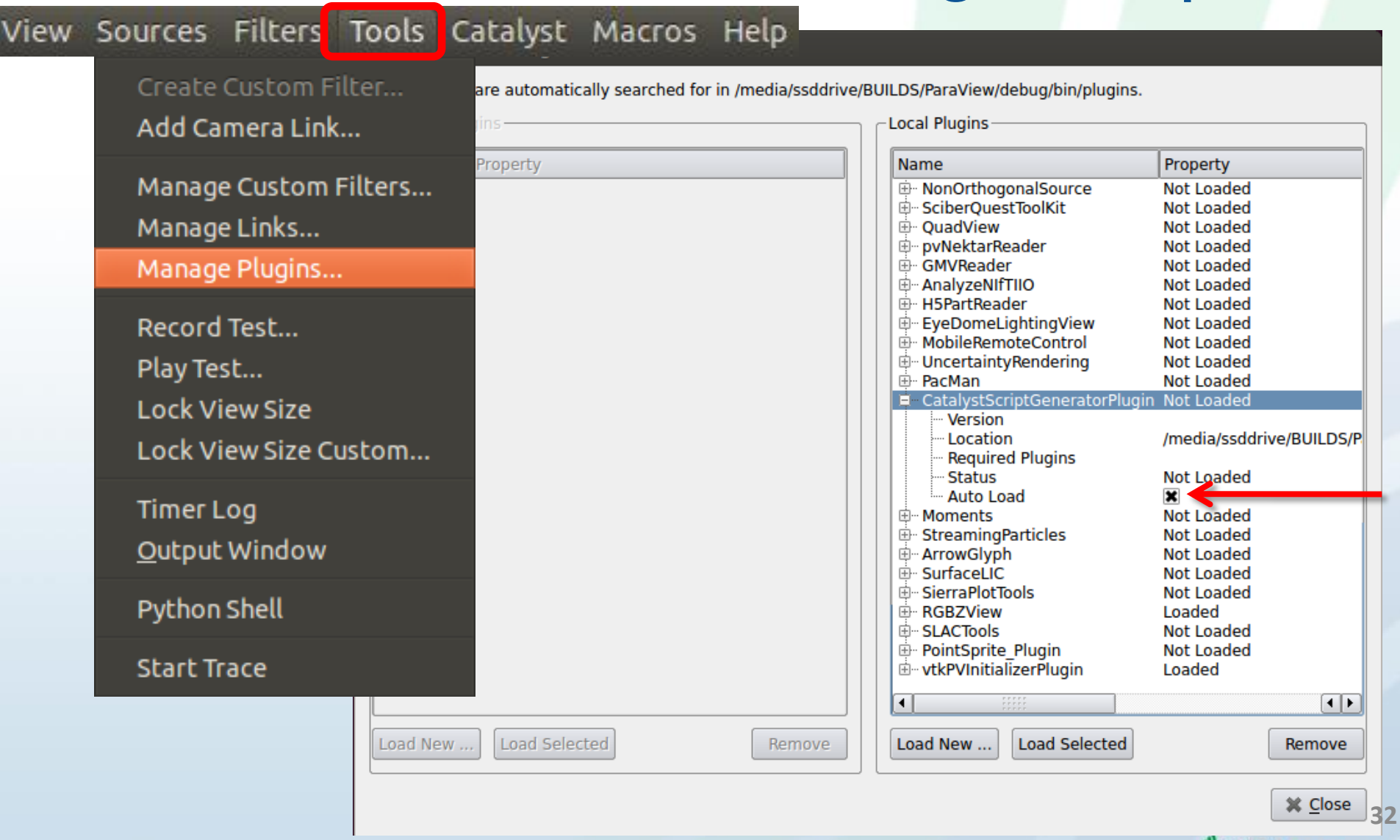
- Need something to start with
- Grid of the same type
- Fields of the same type
- The closer to the actual simulation the better

Run It!

- Go to ~/miniFE-2.0_ref_Catalyst/src

- First time use gridwriter.py
  - Simple Catalyst Python pipeline to write out the full grid and fields every time step
  - Run with “./miniFE.x script_names=gridwriter.py”
  - Output files will be filename_*.pvti
  - Default grid dimensions will be 11 points in each direction

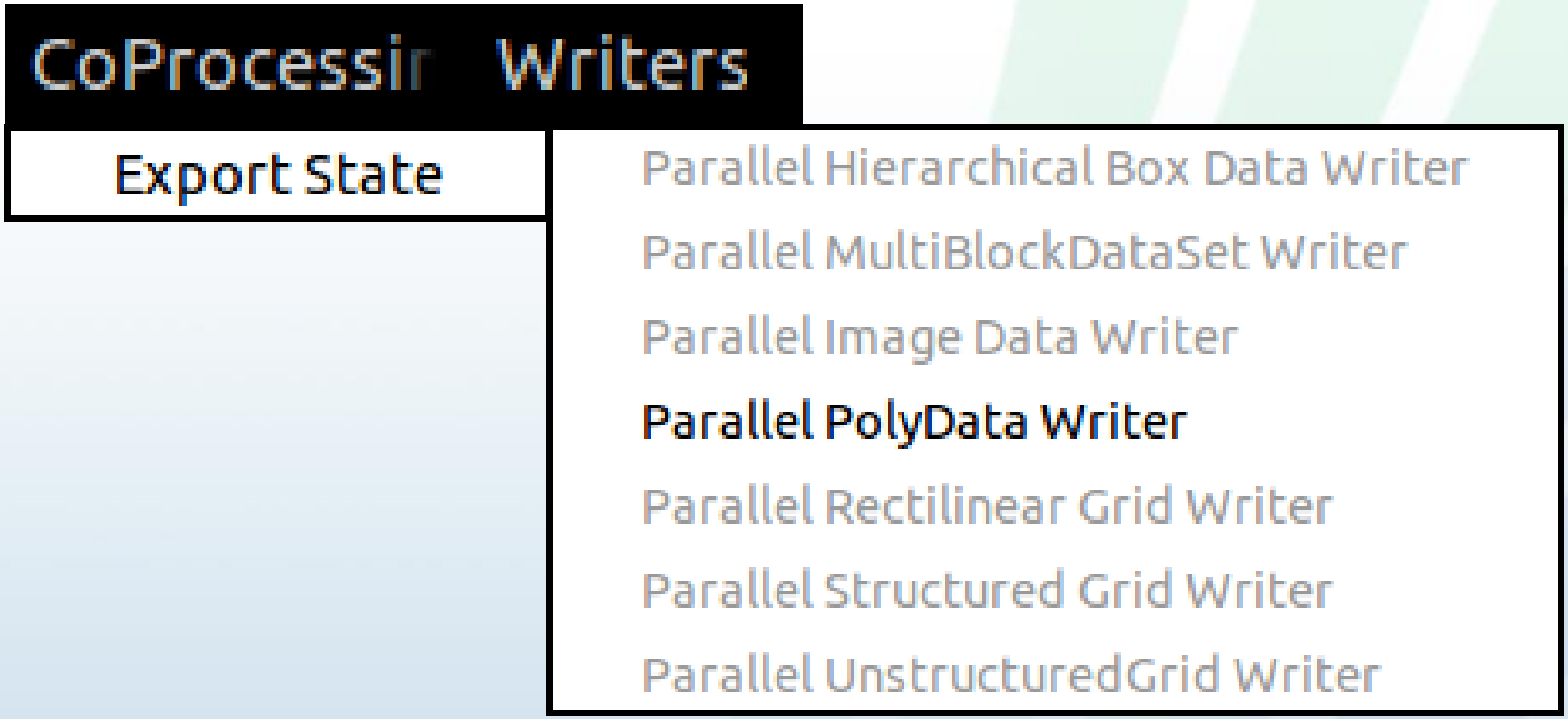
In Situ Demo – Load Plugin Step



Called CoProcessingPlugin for ParaView 4.1 and earlier

Kitware

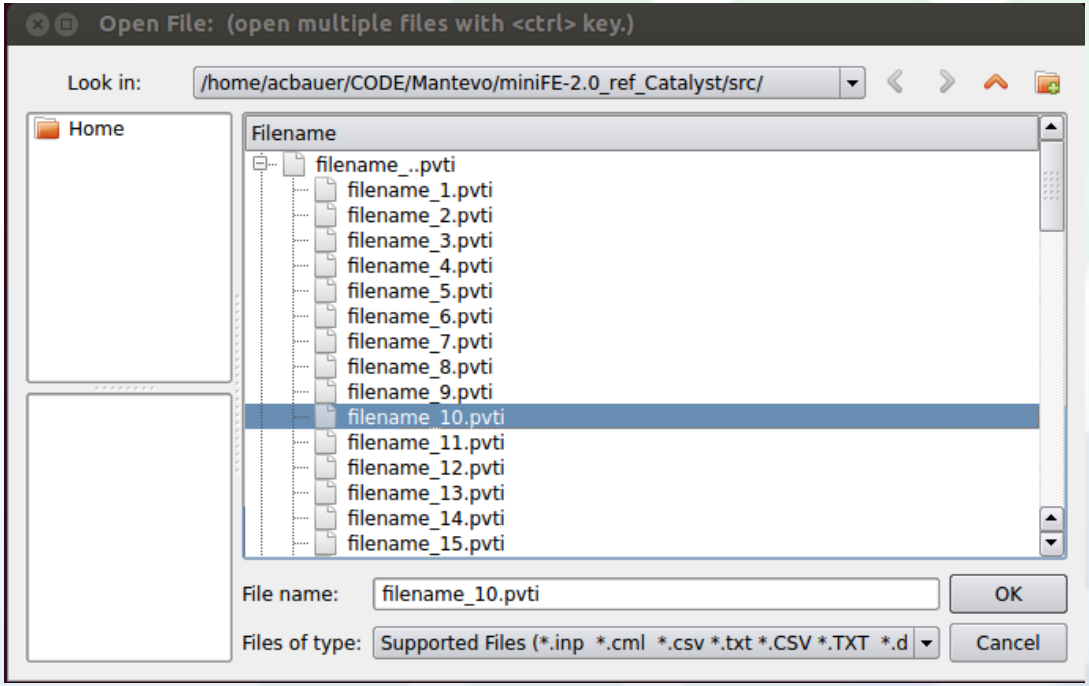
In Situ Demo – New Plugin Menus



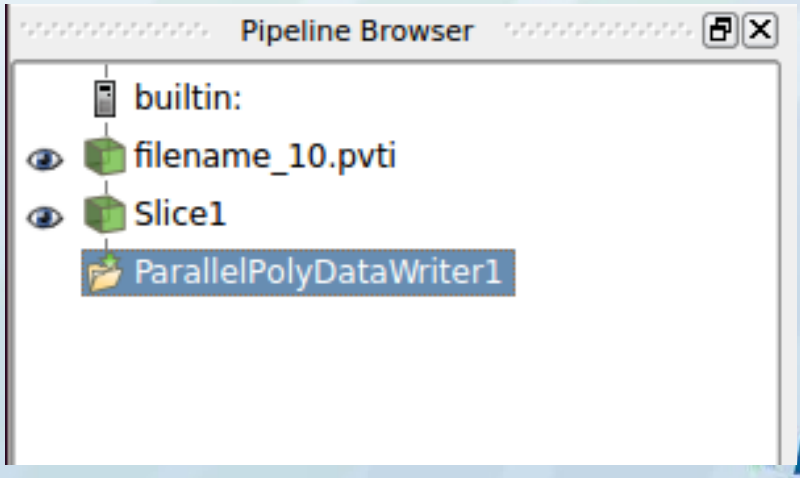
Kitware

In Situ Demo – Creating a Catalyst Python Script

- Load output file filename_*.pvti from previous run
  - Located in ~/miniFE-2.0_ref_Catalyst/src



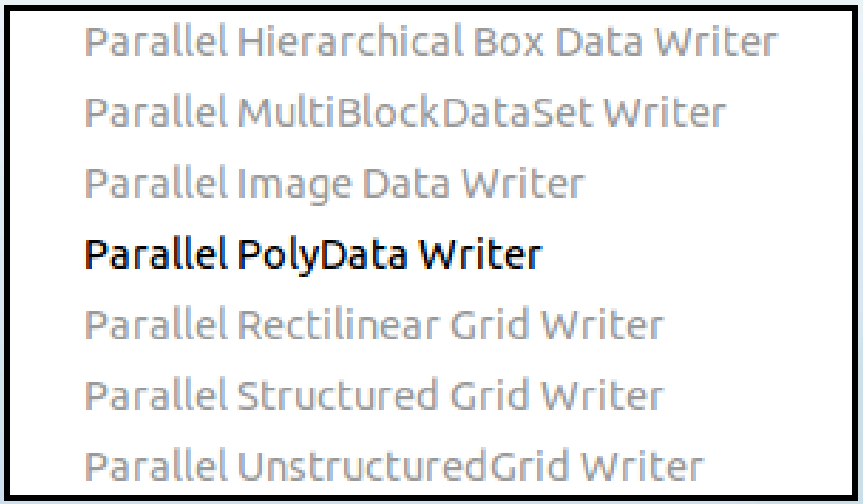
- Create desired pipeline

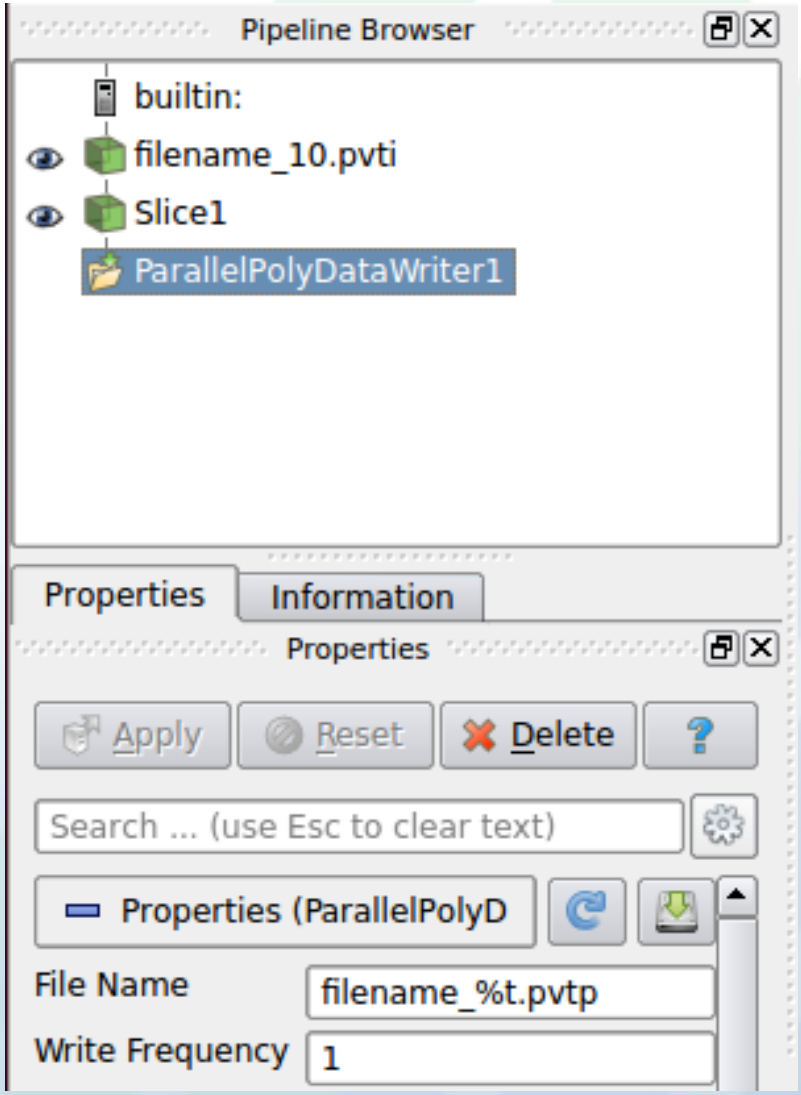


Kitware

In Situ Demo – Adding in Writers

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





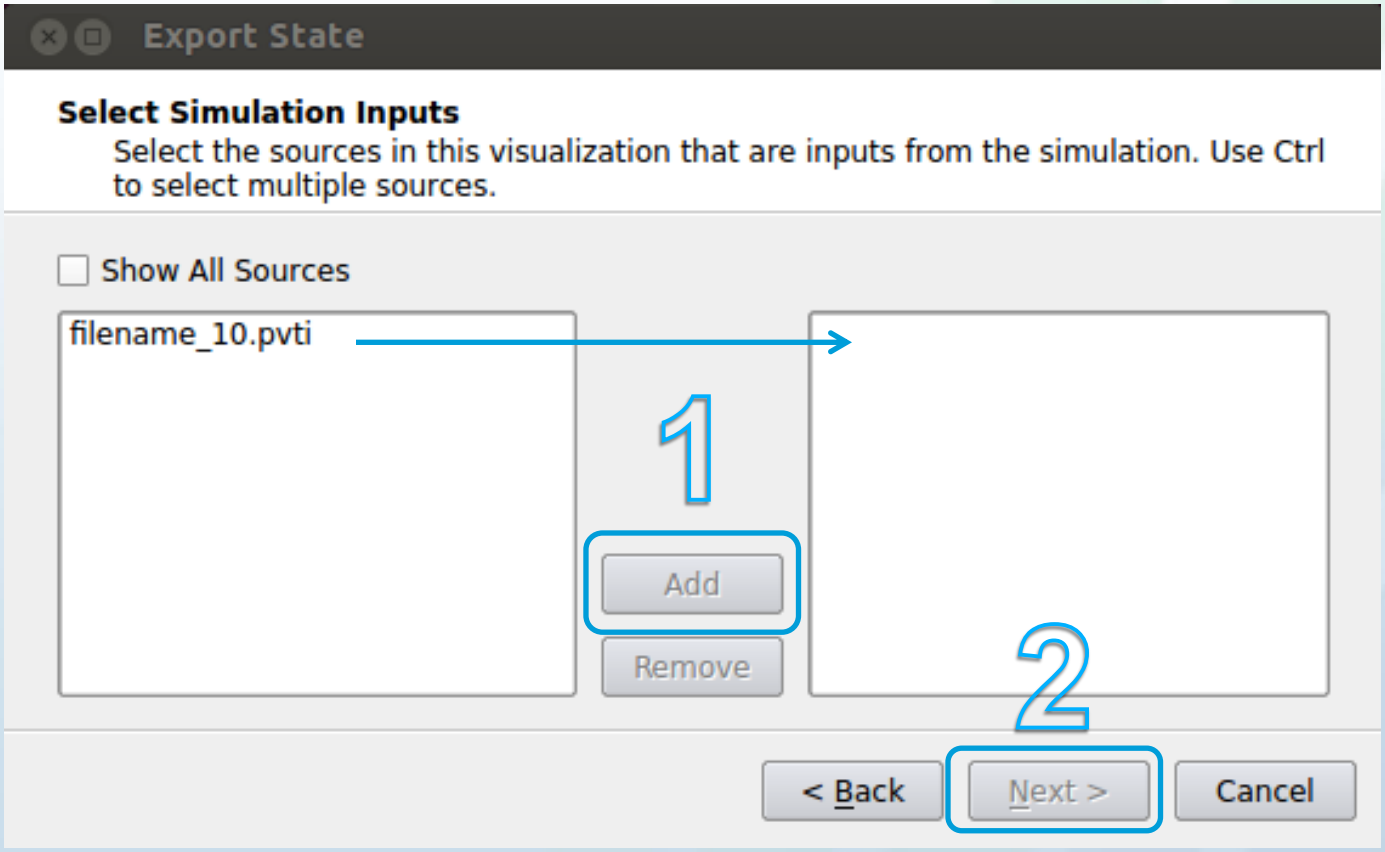
Kitware

In Situ Demo – Exporting the Script

Kitware

In Situ Demo – Select Inputs

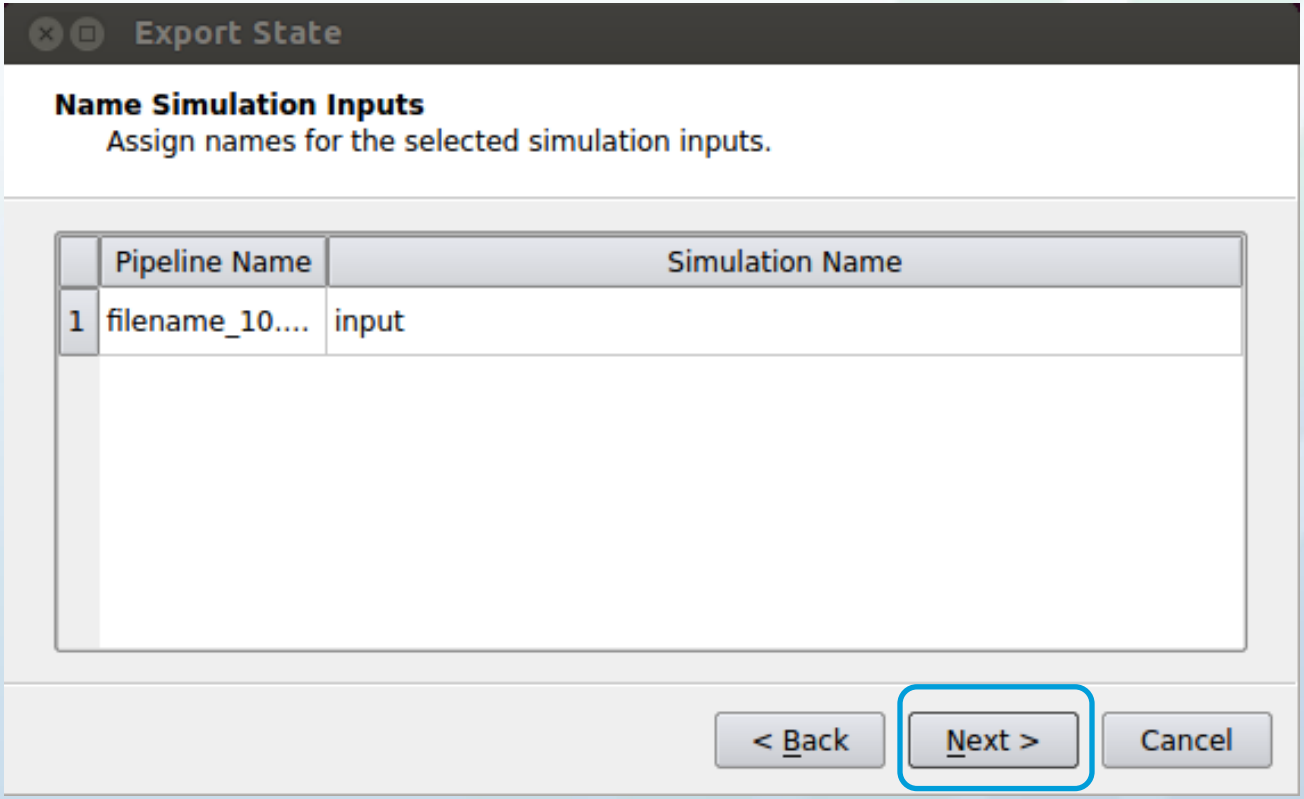
- Usually only a single input but can have multiple inputs
- Each pipeline source is a potential input



Kitware

In Situ Demo – Match Up Inputs

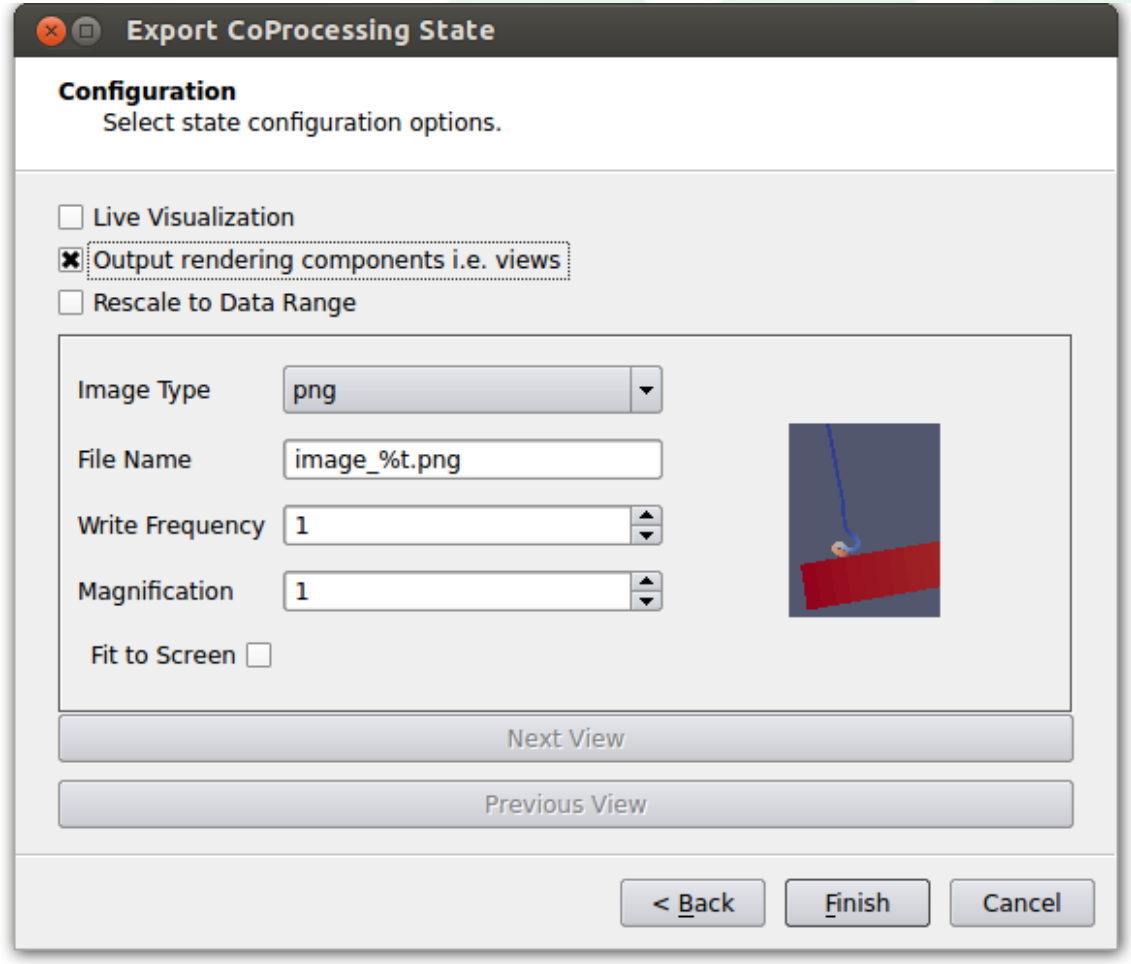
- Source name (e.g. “filename_10.pvti”) needs to be matched with string key in adaptor (e.g. “input” which is the default convention for single inputs)



Kitware

In Situ Demo – Generating an Image

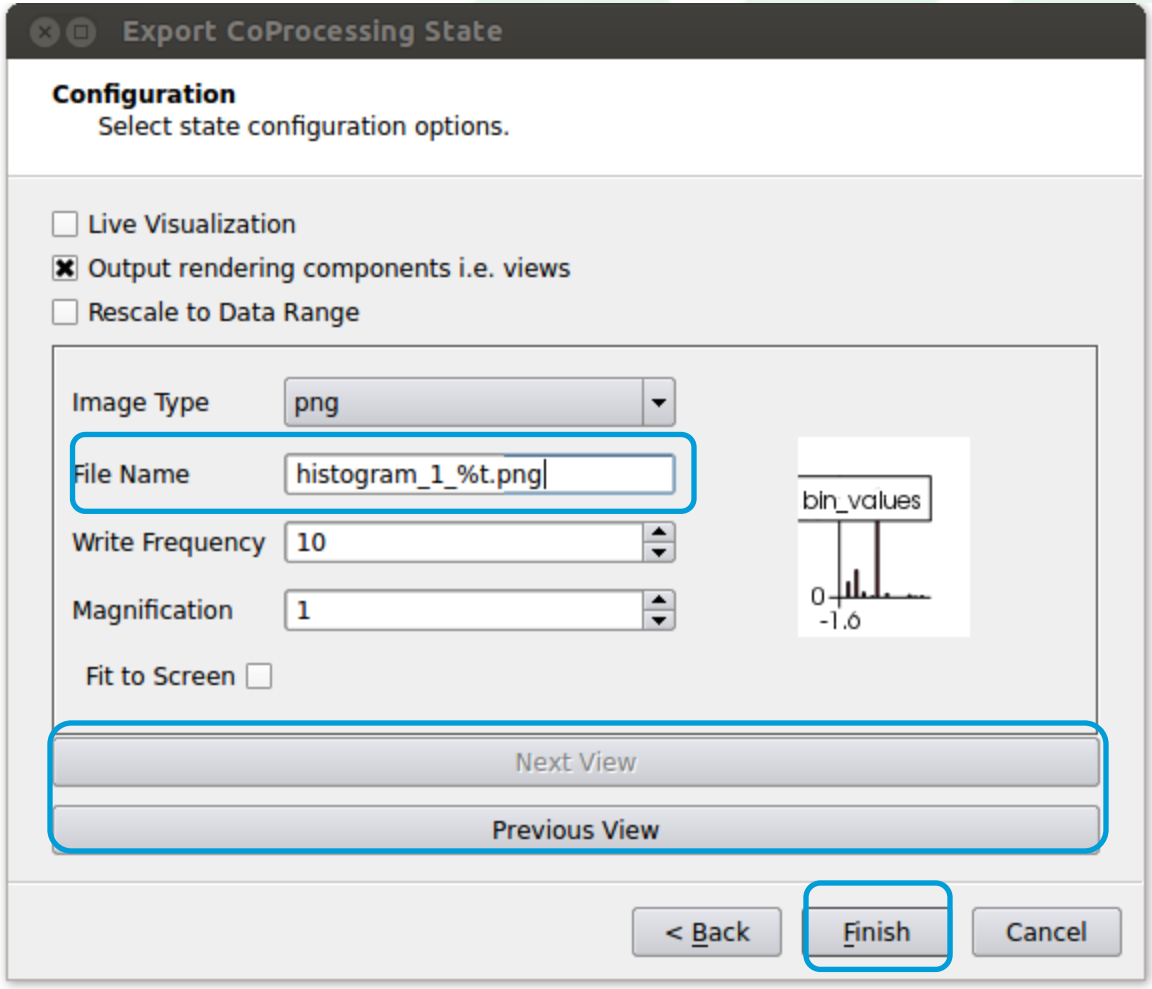
- Parameters/Options:
  - Live visualization
  - Rescale to Data Range (all images)
  - Individual images
    - Image Type
    - File Name
    - Write Frequency
    - Magnification
    - Fit to Screen
- %t gets replaced with time step



Kitware

In Situ Demo – Generating Two Images

- Requires two views in ParaViews GUI
  - Use Next View and Previous View to toggle between GUI views



Kitware

In Situ Demo – Write Out the Script

- Generated script will look something like this

```
def DoCoProcessing(datadescription):

  input = CreateProducer( datadescription, "input" )

  ParallelMultiBlockDataSetWriter1 = CreateWriter( XMLMultiBlockDataWriter, "filename_%t.vtm", 1 )
```

```
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))
                                ine()


                                nager.GetRenderViews()

                                renderviews)):
                                ame.replace("%v", str(view))
                                ace("%t", str(timestep))
                                renderviews[view])

                                oxies -- we do it this way to avoid problems with prototypes
                                Delete()
                                :

                                esToDelete()


    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()
      iter.Next()
      if proxygroup != 'timekeeper' and proxy != None and proxygroup.find("pq_helper_proxies") == -1 :
          tobedeleted.append(proxy)

    return tobedeleted

def CreateProducer(datadescription, gridname):
  "Creates a producer proxy for the grid"
  if not datadescription.GetInputDescriptionByName(gridname):
    raise RuntimeError, "Simulation input name '%s' does not exist" % gridname
  grid = datadescription.GetInputDescriptionByName(gridname).GetGrid()
  producer = TrivialProducer()
  producer.GetClientSideObject().SetOutput(grid)
  producer.UpdatePipeline()
  return producer


def CreateWriter(proxy_ctor, filename, freq):
    global cp_writers
    writer = proxy_ctor()
    writer.FileName = filename
    writer.add_attribute("cpFrequency", freq)
    writer.add_attribute("cpFileName", filename)
    cp_writers.append(writer)
    return writer
```

In Situ Demo – Run the Script

- Put the generated Python script in ~/miniFE-2.0_ref_Catalyst/src directory

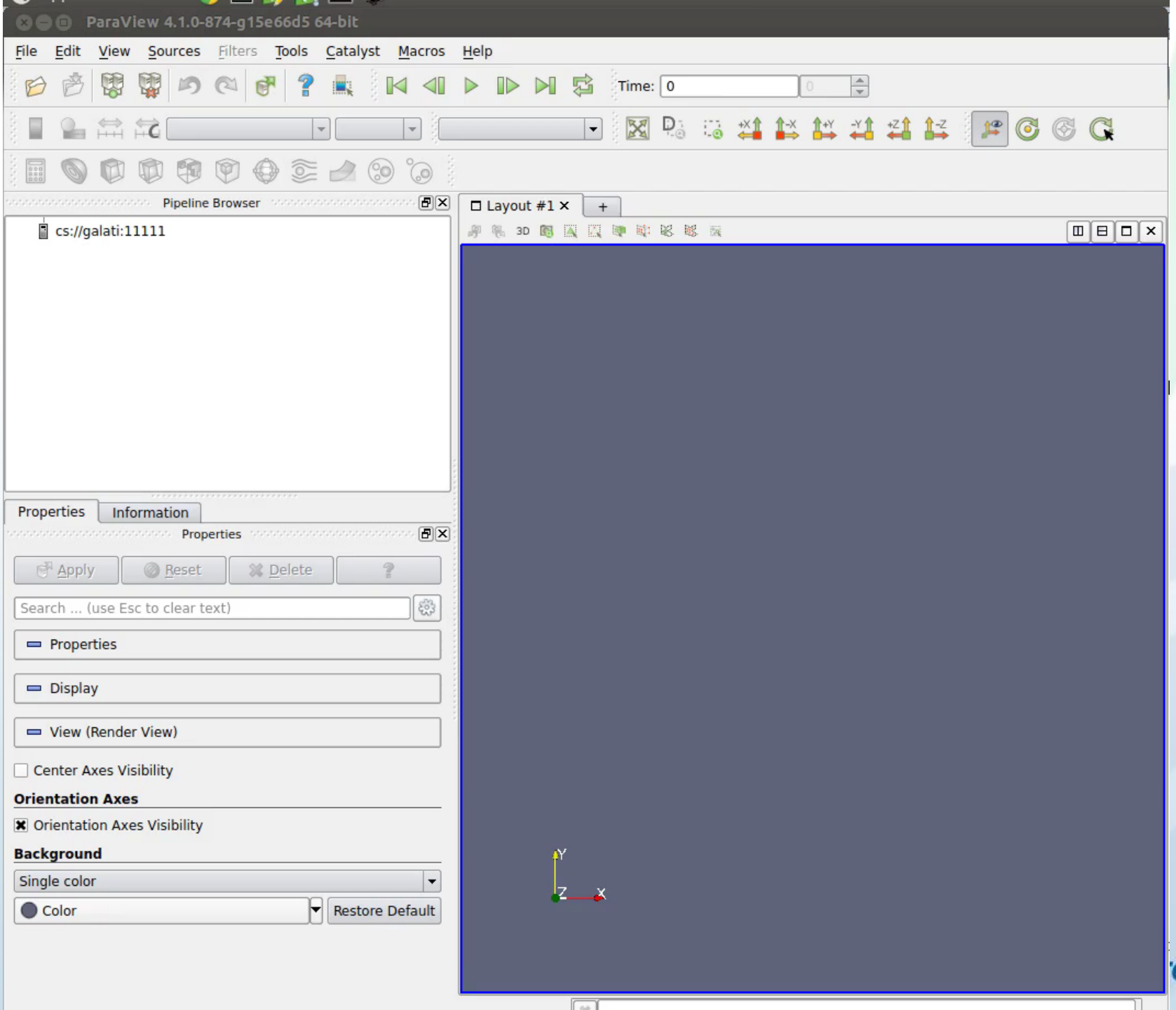
- From ~/miniFE-2.0_ref_Catalyst/src directory run with "./miniFE.x script_names=<outputscript>"
  - Can use MPI to run with more processes
  - Change grid size with nx=, ny=, nz= command line arguments
  - Run multiple scripts with comma separated list script_names=<script1>,<script2>,<script3>,…

Live *In Situ* Analysis and Visualization

- Everything before this was “batch”
  - Preset information with possible logic added
- Functionality for interacting with simulation data during simulation run
  - When exporting a Python script, select “Live Visualization”
  - During simulation run choose the “Tools→Connect to Catalyst” GUI menu item

Two Ways for Live *In Situ* Analysis and Visualization

- Without blocking
  - Simulation proceeds while the user interacts with the data from a specific time step
  - “At scale” mode
- With blocking (new in ParaView 4.2)
  - Simulation is blocked while the user interacts with the data
  - “Debugging” mode
- Can switch between interactive and batch during run as well as disconnecting from the run


ParaView 4.1.0-874-g15e66d5 64-bit
File Edit View Sources Filters Tools Catalyst Macros Help
Time: 0
Pipeline Browser
cs://galati:11111
Properties Information
Properties
Apply Reset Delete
Search ... (use Esc to clear text)
Properties
Display
View (Render View)
Center Axes Visibility
Orientation Axes
Orientation Axes Visibility
Background
Single color
Color
Restore Default
Layout #1
3D
Y
Z
X

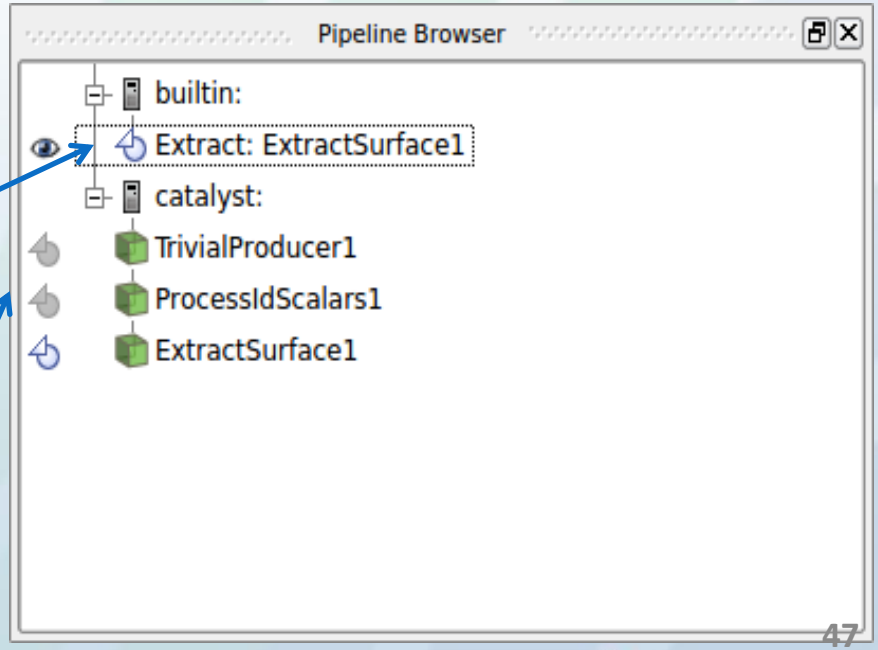
Live *In Situ* Example

- Need to “lengthen” the simulation time
  - Increase nx, ny and nz
- Run as before with “./miniFE.x script_names=<outputscript>”
  - Must have Catalyst “Live Visualization” enabled
- Start ParaView and select Tools→Connect to Catalyst
  - Select port (22222 is default)

Live *In Situ* Example

- Only transfer requested data from server (simulation run) to client
  - ExtractSurface1 is already getting extracted
- Use 

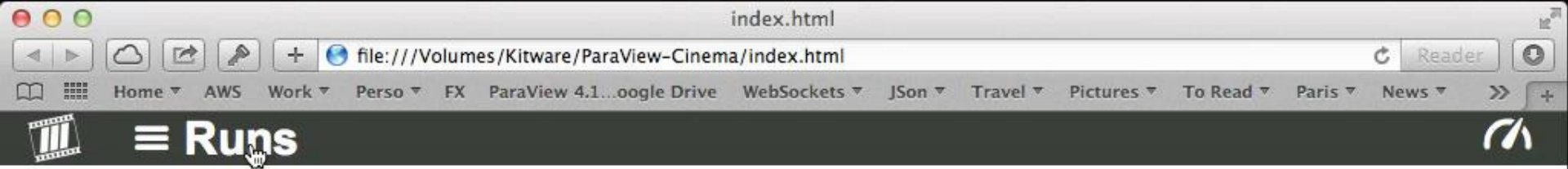
on client to stop transferring to client
- Click on  to transfer to client from Catalyst



Kitware

ParaView Cinema + Catalyst

- Image output is orders magnitude less than an entire data set
  - Helios: full data set – 448 MB, surface of blades – 2.8 MB, image – 71 KB
- Use Catalyst to make a set of images at each time step with small output changes (rotations, slice planes locations, iso-surfaced variables & values, etc.)
- Use a web-browser to view images later
- In ParaView 4.2 as beta functionality

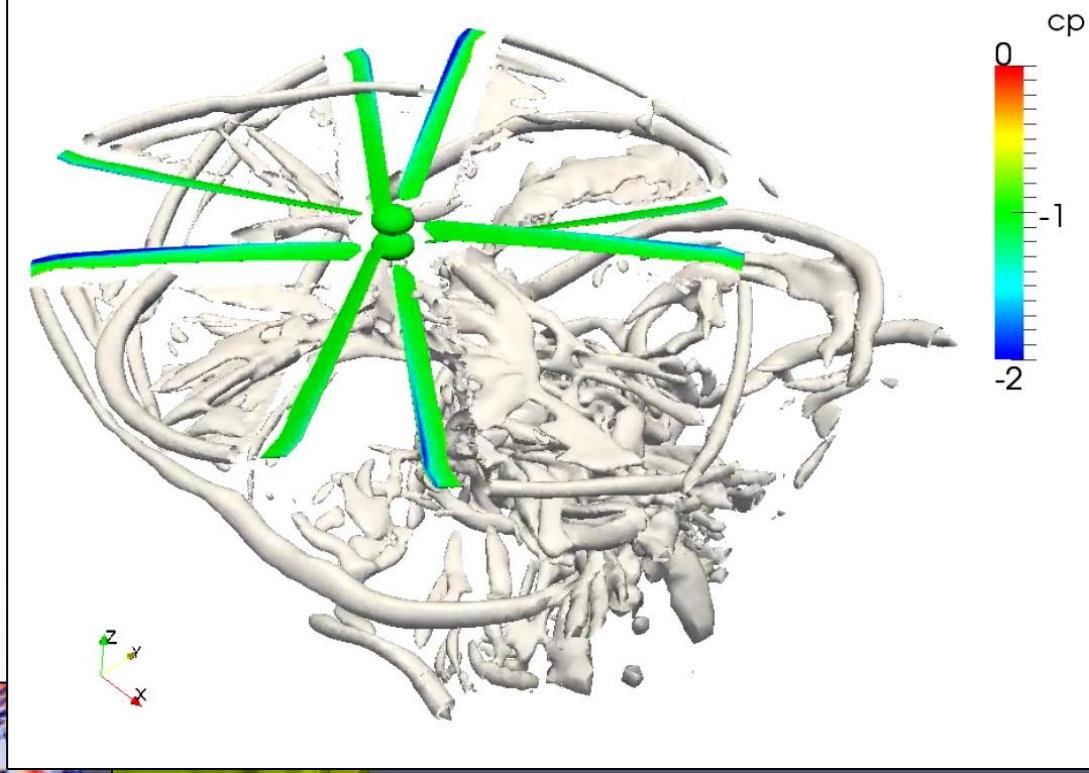


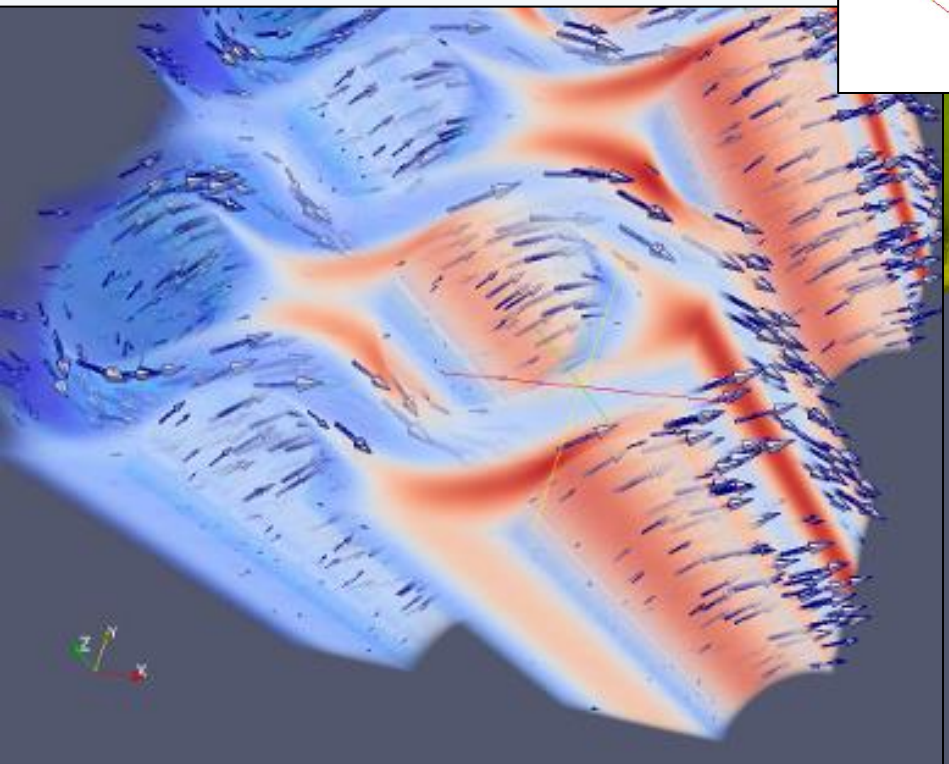
An Image-based approach to Extreme Scale In Situ Visualization and Analysis

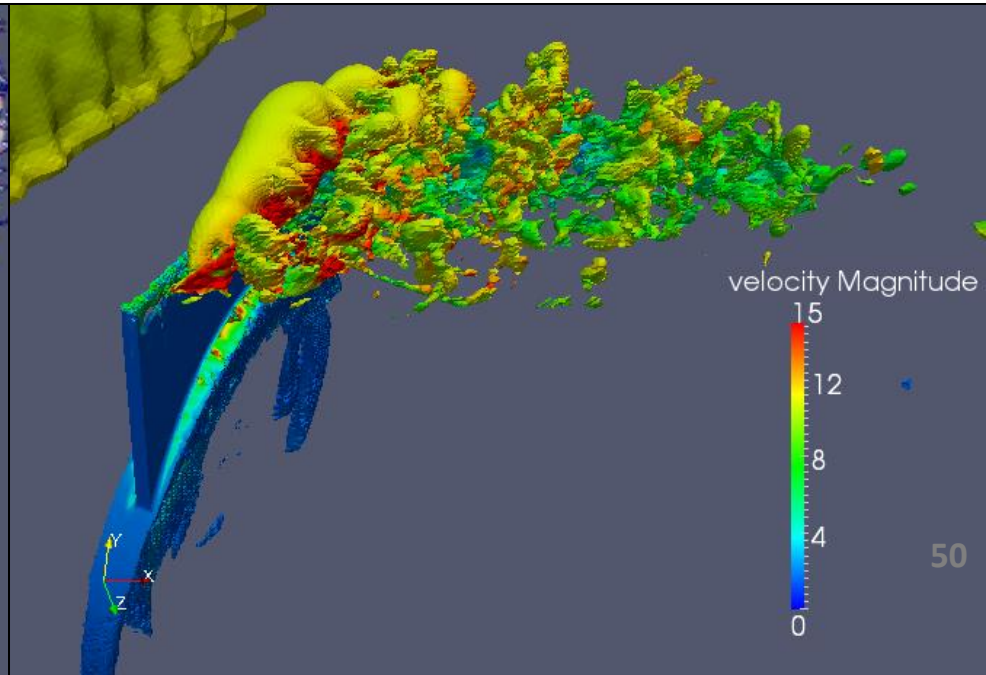
James Ahrens*, Sebastien Jourdain**, Patrick O'Leary**, John Patchett*, David H. Rogers*

* Los Alamos National Laboratory ** Kitware Inc.

Gratuitous Catalyst Output







Kitware

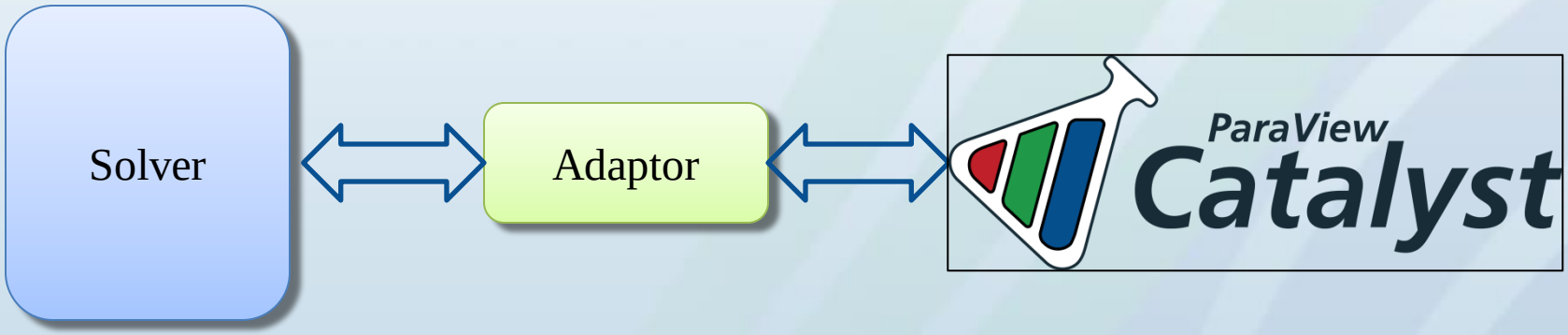
ParaView Catalyst for Simulation Developers

Developers Section Overview

- Most work is done in creating the adaptor between the simulation code and Catalyst
- Small footprint in main simulation code base
- Needs to be efficient both in memory and computationally
- Fortran, C++ and Python examples
  - Fortran – ~/Examples/CatalystExampleCode/FortranPoissonSolver
  - C++ – ~/Examples/miniFE-2.0_ref_Catalyst
  - Python – ~/Examples/CatalystExampleCode/PythonDolfinExample

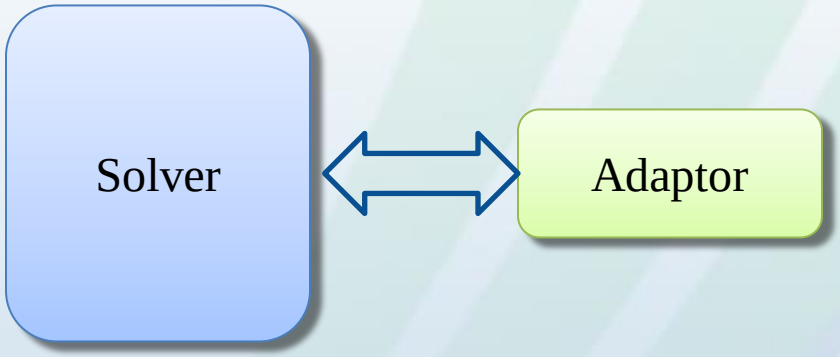
Interaction Overview

- Simulation has separate data structures from VTK data structures
- Use an adaptor to bridge the gap
  - Try to reuse existing memory
  - Also responsible for other interactions between simulation code and Catalyst



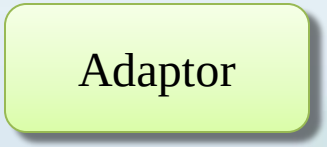
Interaction Overview

- Functional interface – typically 3 calls between simulation code and adaptor
  - Initialize
  - CoProcess
  - Finalize



Adaptor Overview

- Creates VTK data objects representing simulation data
- Creates Catalyst pipelines
  - Information on how to process VTK data objects to get desired output



Finalize Overview

- Last chance to do something
- Clean up any allocated memory

Information Needed By Adaptor During Initialization

- What pipelines to use?
  - Canned pipelines – pre-built pipelines that take in a few parameters (e.g. slice output with output file name, frequency and slice normal and origin)
  - Python pipelines – generated in the ParaView GUI and potentially edited
- Optionally
  - Information to construct grids and fields

Information Needed By Adaptor During CoProcess

- Time, time step
- Optionally
  - Information to construct grids and fields if not passed in during initialization step
  - Force all pipelines to execute (simulation may know something important has happened)
  - Other information to add in extra logic to pipelines

VTK Preliminaries

- Many objects in VTK are reused so we use reference counting to keep track of how many other objects are using them
  - New() – creates a new object in dynamic memory and sets reference count to 1
  - Delete() – reduces reference count by 1 and if reference count goes to 0 then object gets deleted
  - a->Set*(b), a->Add*(b) – b's reference count gets increased by 1

```
vtkDoubleArray* a = vtkDoubleArray::New(); // a's ref count = 1
vtkPointData* pd = vtkPointData::New();    // pd's ref count = 1
pd->AddArray(a);                           // a's ref count = 2
a->Delete();                               // a's ref count = 1
a->SetName("an array");                    // valid as a hasn't been deleted
pd->Delete();                              // deletes both pd and a
```

VTK Preliminaries (Python)

- The whole API is available in Python, though syntax is different
- All variables are references
- An object is deleted when no variable references it anymore
- No need for smart pointers

```
a = vtk.vtkDoubleArray()          # a's ref count = 1
pd = vtk.vtkPointData()           # pd's ref count = 1
pd.AddArray(a)                    # a's ref count = 2
a = None                          # array's ref count = 1 but a doesn't reference it
a.SetName("an array")             # invalid as a doesn't reference the array
pd.GetArray(0).SetName("an array")  # valid since array still exists
pd = None                         # deletes both pd and the array
```

Helpful VTK Objects

- vtkSmartPointer<vtkObject*>
  - Deals with creating and deleting vtkObjects
  - Increases reference count when set
  - Reduces reference count when it gets deleted or unset

```
vtkPointData* pd = vtkPointData::New();          // pd's ref count = 1
{
  vtkSmartPointer<vtkDoubleArray> a =
    vtkSmartPointer<vtkDoubleArray>::New();
  pd->AddArray(a);
}
// a no longer exists but the generated vtkDoubleArray does
vtkSmartPointer<vtkPointData> pd2 = pd;          // pd's ref count = 2
```


http://www.vtk.org/doc/nightly/html/classvtkSmartPointer.html

Helpful VTK Objects (2)

- vtkNew<vtkObject*>
  - Deals with creating and deleting vtkObjects
  - Reduces reference count when it gets deleted
  - Need GetPointer() when a method needs a vtkObject

```
vtkPointData* pd = vtkPointData::New();        // pd's ref count = 1
{
  vtkNew<vtkDoubleArray> a;
  pd->AddArray(a.GetPointer());       // need GetPointer() here and not just a
}
// a no longer exists but the generated vtkDoubleArray does
```

http://www.vtk.org/doc/nightly/html/classvtkNew.html

Writing the Initialization Method

- Assuming we will pass in data to create VTK grid and field data objects during CoProcess call
- ParaView classes to look at:
  - vtkCPProcessor – main manager object
  - vtkCPPipeline and vtkCPPythonScriptPipeline – object to control Catalyst pipeline (typically one object per pipeline)

http://www.paraview.org/ParaView3/Doc/Nightly/html/classvtkCPProcessor.html

http://www.paraview.org/ParaView3/Doc/Nightly/html/classvtkCPPythonScriptPipeline.html

vtkCPProcessor Initialization Step

- vtkCPProcessor
  - vtkCPProcessor* New()
    - Static constructor method that sets reference count to 1
  - void Delete()
    - Reduce reference count by 1 (not necessarily delete though)
  - int Initialize()
    - Returns 1 for success and 0 for failure
  - int AddPipeline(vtkCPPipeline* pipeline)
    - Pass in a vtkCPPipeline object, vtkCPProcessor will keep reference to pipeline so it won't get deleted
    - Returns 1 for success and 0 for failure
- Will go through other important vtkCPProcessor methods later

vtkCPProcessor Initialization Step

For ParaView 4.2, can initialize Catalyst with a separate MPI communicator

- Use “int Initialize(vtkMPICommunicatorOpaqueComm*)” instead of “int Initialize()”
- vtkMPICommunicatorOpaqueComm defined in vtkMPI.h
- Use constructor vtkMPICommunicatorOpaqueComm(MPI_Comm*)

vtkCPPipeline Initialization Step

- vtkCPPipeline
  - Abstract class – methods are usually only called by vtkCPProcessor
- vtkCPPythonScriptPipeline
  - Concrete class that derives from vtkCPPipeline
  - vtkCPPythonScriptPipeline* New()
    - Static constructor method that sets reference count to 1
  - int Initialize(const char *fileName)
    - fileName is the Python script pipeline which can include a relative or absolute path
    - Returns 1 for success and 0 for failure

vtkCPProcessor Finalize Step

- vtkCPProcessor
  - int Finalize()
  - Delete the vtkCPProcessor object
- If ParaView was built with DEBUG_LEAKS_ON, will get messages that VTK objects were not deleted before exiting.
  - Most likely from not deleting vtkCPProcessor, vtkCPipeline or VTK objects created in the adaptor

Configuring with CMake

- Configuring the build is simplified by using CMake
  - Determines dependencies automatically for compiling (location of header files & modules) and linking (location of libraries)
- Typically use ccmake for terminal-based interface
  - Often building on a remote machine
  - Command in build directory: ccmake <path to source dir> <preset configuration options>

Example 1

- Goals
  - Initialize Catalyst
  - Create a set of vtkCPPythonScriptPipelines
  - Finalize Catalyst
  - Only do Fortran90, C++ or Python example
- Hints
  - Function interfaces already specified with needed arguments
- Result
  - Nothing output yet from Catalyst

Fortran90 Example 1 – Build Hints

- Source code in ~/Examples/CatalystExampleCode/FortranPoissonSolver
- CMake build configuration
  - Create a build directory outside of the source directory at ~/FPSBuild
- From ~/FPSBuild directory do “ccmake ~/Examples/CatalystExampleCode/FortranPoissonSolver”
  - Uses CMAKE_PREFIX_PATH to find the location of the ParaView OSMesa build at ~/ParaView4.2/build_mesa
  - With ccmake, hit “c” to configure. May need to do this multiple times until acceptable values are entered
  - Once acceptable values are set, hit “g” to generate Makefiles
- Enter “make” at command line to build in ~/FPSBuild
  - Result is FortranPoissonSolver executable

Fortran90 Example 1 – Write It!

- Catalyst Interface in:
  - ~/ParaView4.2/ParaViewCoProcessing/Catalyst/CAdaptor.API.h
    - For initializing and finalizing Catalyst from Fortran or C code
  - ~/ParaView4.2/ParaView/CoProcessing/PythonCatalyst/CPythonAdaptorAPI.h
    - For adding in Python scripts from Fortran or C code
- Assume Python scripts for pipeline specifications are passed in as command line arguments
- Starting file:
  - ~/Examples/CatalystExampleCode/FortranPoissonSolver/FEFortranAdaptor.F90
  - Subroutine signatures:
    - initializecoprocessor()
    - finalizecoprocessor()

Fortran90 Example 1 – Executable

- Running the example
  - ./FortranPoissonSolver
  - Command line arguments is a list of Python scripts
    - E.g. ./FortranPoissonSolver script1.py script2.py
- Nothing extra output from run yet

Fortran90 Example 1 – A Solution

~/Examples/CatalystExampleCode/FortranPoissonSolver/example1/FEFortranAdaptor.F90

```
subroutine initializecoprocessor()
  implicit none
  integer :: ilen, i
  character(len=200) :: arg

  call coprocessorinitialize()
  do i=1, iargc()
    call getarg(i, arg)
    ilen = len_trim(arg)
    arg(ilen+1:) = char(0)
    call coprocessoraddpythonscript(arg, ilen)
  enddo
end subroutine initializecoprocessor
```

```
subroutine finalizecoprocessor()
  call coprocessorfinalize()
end subroutine finalizecoprocessor
```

C++ Example 1 – Build Hints

- Source code in ~/Examples/miniFE-2.0_ref_Catalyst/adapter/catalyst_adapter.cpp
- CMake build configuration
  - Create a build directory outside of the code directory at ~/Examples/miniFE-2.0_ref_Catalyst/adapter_build
  - From adapter_build directory do “ccmake ../adapter”
    - Uses CMAKE_PREFIX_PATH to find the location of the ParaView OSMesa build at ~/ParaView4.2/build_mesa
    - With ccmake, hit “c” to configure. May need to do this multiple times until acceptable values are entered
    - Once acceptable values are set, hit “g” to generate Makefiles
- Build Catalyst adapter as a library
  - Enter “make” at command line to build in ~/Examples/miniFE-2.0_ref_Catalyst/adapter_build
  - Result is libCatalystMiniFERefAdaptor.so

C++ Example 1 – Write It!

- Starting file ~/Examples/miniFE-2.0_ref_Catalyst/adapter/catalyst_adapter.cpp

- Function signatures:
  - int initialize(miniFE::Parameters& params)
    - Python script names in params.script_names which is type std::vector<std::string>
    - vtkCPProcessor* Processor in anonymous namespace
  - int finalize()
  - Driver code already calls these functions

- Assume Python scripts for pipeline specifications

C++ Example 1 – Executable

- libCatalystMiniFERefAdaptor.so library must already be built
- In ~/Examples/miniFE-2.0_ref_Catalyst/src
  - Enter “make” at command line to build
  - Result executable is miniFE.x
- Running the example
  - ./miniFE.x script_names=gridwriter.py
  - Command line arguments
    - nx,ny,nz for number of points in corresponding directions
      - Default of 10 for each, if nz not set uses ny value, if ny is not set uses nx value
    - script_names=<comma separated list of Python scripts>
- Nothing extra output from run yet

C++ Example 1 – A Solution

~/Examples/miniFE-2.0_ref_Catalyst/adapter/examples/
example1_catalyst_adaptor.cpp

```
void initialize(miniFE::Parameters& params)
{
  Processor = vtkCPProcessor::New();
  Processor->Initialize();
  for(std::vector<std::string>::const iterator
      it=params.script_names.begin();
      it!=params.script_names.end();it++) {
    vtkCPPythonScriptPipeline* pipeline =
      vtkCPPythonScriptPipeline::New();
    pipeline->Initialize(it->c_str());
    Processor->AddPipeline(pipeline);
    pipeline->Delete();
    }
 }
```

```
void finalize()
 {
  if(Processor)
    Processor->Finalize();
    Processor->Delete();
    Processor = NULL;
    }
 }
```

Python Example 1 – Build Hints

- Source code in ~/Examples/CatalystExampleCode/PythonDolfinExample
- CMake build configuration
  - Create a build directory outside of the source directory at ~/PythonBuild
- From ~/PythonBuild directory do “ccmake ~/Examples/CatalystExampleCode/PythonDolfinExample”
  - Uses CMAKE_PREFIX_PATH to find the location of the ParaView OSMesa build at ~/ParaView4.2/build_mesa
  - With ccmake, hit “c” to configure. May need to do this multiple times until acceptable values are entered
  - Once acceptable values are set, hit “g” to generate Makefiles
- Enter “make” at command line to build in ~/PythonBuild
  - Result is *run-yours.sh* executable script
  - ./run-yours.sh [coprocessingscript.py] [# iterations]
  - For instance : *./run-yours.sh gridwriter.py 10*

Python Example 1 – Write It!

- Uses a Python demo from **dolfin 1.3.0**
  http://fenicsproject.org/documentation/dolfin/dev/python
- Starting file (the one you will modify):
  ~/Examples/CatalystExampleCode/PythonDolfinExample/**simulation-yours.py**
- Import paraview, vtk and other needed packages
- Load the co-processing script
- To run the simulation
  - from ~/PythonBuild
  - ./run-yours.sh to run the version **you** modify
  - ./run-original.sh to run original (out of the box) demo
  - or ./run-catalyst-step#.sh to execute solution for step #

Kitware

Python Example 1 – A Solution

~/Examples/CatalystExampleCode/PythonDolfinExample/simulation-catalyst-step1.py

```
import sys
import paraview
import paraview.vtk as vtk
import paraview.simple as pvsimple
import vtkPVCatalystPython
import os
```

```
# initialize and read input parameters
paraview.options.batch = True
paraview.options.symmetric = True
# import user co-processing script
scriptpath, scriptname = os.path.split(sys.argv[1])
sys.path.append(scriptpath)
if scriptname.endswith(".py"):
    print 'script name is ', scriptname
    scriptname = scriptname[0:len(scriptname)-3]
try:
    cpscript = __import__(scriptname)
except:
    print sys.exc_info()
    sys.exit(1)
```

Kitware

Writing the CoProcess Method

- Assuming we will pass in data to create VTK grid and field data objects during CoProcess call
- ParaView classes to look at:
  - vtkCPProcessor – main manager object
  - vtkCPDataDescription

http://www.paraview.org/ParaView3/Doc/Nightly/html/classvtkCPDataDescription.html

Writing the CoProcess Method

Main section with 2 stages

- First stage
  - Check with Catalyst pipelines to see if there is anything to do
  - Should check after every time step
  - Should be very fast
- Second stage
  - Only done if at least one Catalyst pipeline wants to do something this time step
  - Construct needed input (i.e. VTK grids and fields)
  - Execute the pipelines that need to do something

Writing the CoProcess Method: First Stage

- ParaView classes to look at:
  - vtkCPProcessor – main manager object
  - vtkCPDataDescription – meta-description of what simulation is able to provide
    - Simulation state
    - Pipeline sources

vtkCPProcessor CoProcess Step: First Stage

- vtkCPProcessor
  - int RequestDataDescription(vtkCPDataDescription*)
    - Have the processor check if any Catalyst pipelines need to execute this time step
    - Returns 1 if a pipeline needs to execute and 0 otherwise
- vtkCPDataDescription
  - void AddInput(const char* name)
    - Specify source of pipelines
    - Convention for single input is name="input"
  - void SetTimeData(double time, vtkIdType timeStep)
    - Set the current simulation time and time step
  - void ForceOutputOn()/SetForceOutput() – optional
    - Specify that all Catalyst pipelines should execute

Example 2

- Goals
  - Create a vtkCPDataDescription object and set time and time step
  - Check if any co-processing needs to be done
- Hints
  - Function interfaces already specified with needed arguments
- Result
  - Nothing output yet from Catalyst

Fortran90 Example 2 – Write It!

- Subroutine signature:
  - runcoprocessor(dimensions, step, time, x)
- Starting file:
  - ~/Examples/CatalystExampleCode/
    FortranPoissonSolver/FEFortranAdaptor.F90
  - Only need to use step and time for this part

- Fortran interface at:

~/ParaView4.2/ParaView/CoProcessing/Catalyst/CAdaptorAPI.h

Fortran90 Example 2 – Executable

- Enter “make” at command line to build in ~/FPSBuild
  - Result is FortranPoissonSolver executable
- Running the example
  - ./FortranPoissonSolver
  - Command line arguments is a list of Python scripts
    - E.g. ./FortranPoissonSolver script1.py script2.py
- Nothing extra output from run yet

Fortran90 Example 2 – A Solution

~/Examples/CatalystExampleCode/FortranPoissonSolver/example2/FEFortranAdaptor.F90

```
subroutine runcoprocessor(dimensions, step, time, x)
  use iso_c_binding
  implicit none
  integer, intent(in) :: dimensions(3), step
  real(kind=8), dimension(:), intent(in) :: x
  real(kind=8), intent(in) :: time
  integer :: flag, extent(6)
  real(kind=8), DIMENSION(:), allocatable :: xcp(:)

  call requestdatadescription(step,time,flag)
  if (flag .ne. 0) then
     call getvtkextent(dimensions, extent)

    ! x is the array with global values, we need just this process's
    ! values for Catalyst which will be put in xcp
    allocate(xcp((extent(2)-extent(1)+1)*(extent(4)-extent(3)+1)*(extent(6)-extent(5)+1)))
    call getlocalfield(dimensions, extent, x, xcp)

    deallocate(xcp)
  end if
end subroutine runcoprocessor
```

C++ Example 2 – Write It!

- Starting file:
  - ~/Examples/miniFE-2.0_ref_Catalyst/adapter/
    catalyst_adapter.cpp
- Function signature:
  - void coprocess(const double spacing[3], const Box& global_box, const Box& local_box, std::vector<double>& minifepointdata, int time_step, double time, bool force_output);
  - Only need to use time_step, time and force_output for this part

C++ Example 2 – Build Hints

- Example 1 already took care of CMake configuration
  - Rebuild Catalyst adapter as a library
  - Enter “make” at command line to build in ~/Examples/miniFE-2.0_ref_Catalyst/adapter_build
  - Result is libCatalystMiniFERefAdaptor.so
- Running the example
  - In ~/Examples/miniFE-2.0_ref_Catalyst/src do ./miniFE.x script_names=gridwriter.py
  - Command line arguments
    - nx,ny,nz for number of points in corresponding directions
      - Default of 10 for each, if nz not set uses ny value, if ny is not set uses nx value
    - script_names=<comma separated list of Python scripts>
- Nothing extra output from run yet

C++ Example 2 – A Solution

~/Examples/miniFE-2.0_ref_Catalyst/adapter/examples/
example2_catalyst_adaptor.cpp

```
void coprocess(const double spacing[3], const Box& global_box,
  const Box& local_box, std::vector<double>& minifepointdata,
  int time_step, double time, bool force_output)
{
  vtkSmartPointer<vtkCPDataDescription> dataDescription =
      vtkSmartPointer<vtkCPDataDescription>::New();
  dataDescription->AddInput("input");
    dataDescription->SetTimeData(time, time_step);
  dataDescription->SetForceOutput(force_output);
  if(Processor->RequestDataDescription(dataDescription) == 0)
    {
     return;
     }
…
```

Python Example 2 – Write It!

- Function signature:
  - coProcess(grid, time, step)
    - grid : a VTK data object, in this example a *vtkUnstructuredGrid*
    - time : a real value defining the physical time
    - step : the timestep, i.e. the computation iteration number
- Starting file:
  - simulation-catalyst-step1.py (or your own solution)
  - Only need to use time and step for this part

Python Example 2 – A Solution

~/Examples/CatalystExampleCode/PythonDolfinExample/simulation-catalyst-step2.py

```
def coProcess(grid, time, step):
    # initialize data description
    datadescription = vtkPVCatalystPython.vtkCPDataDescription()
    datadescription.SetTimeData(time, step)
    datadescription.AddInput("input")
    cpscript.RequestDataDescription(datadescription)
    . . .
```

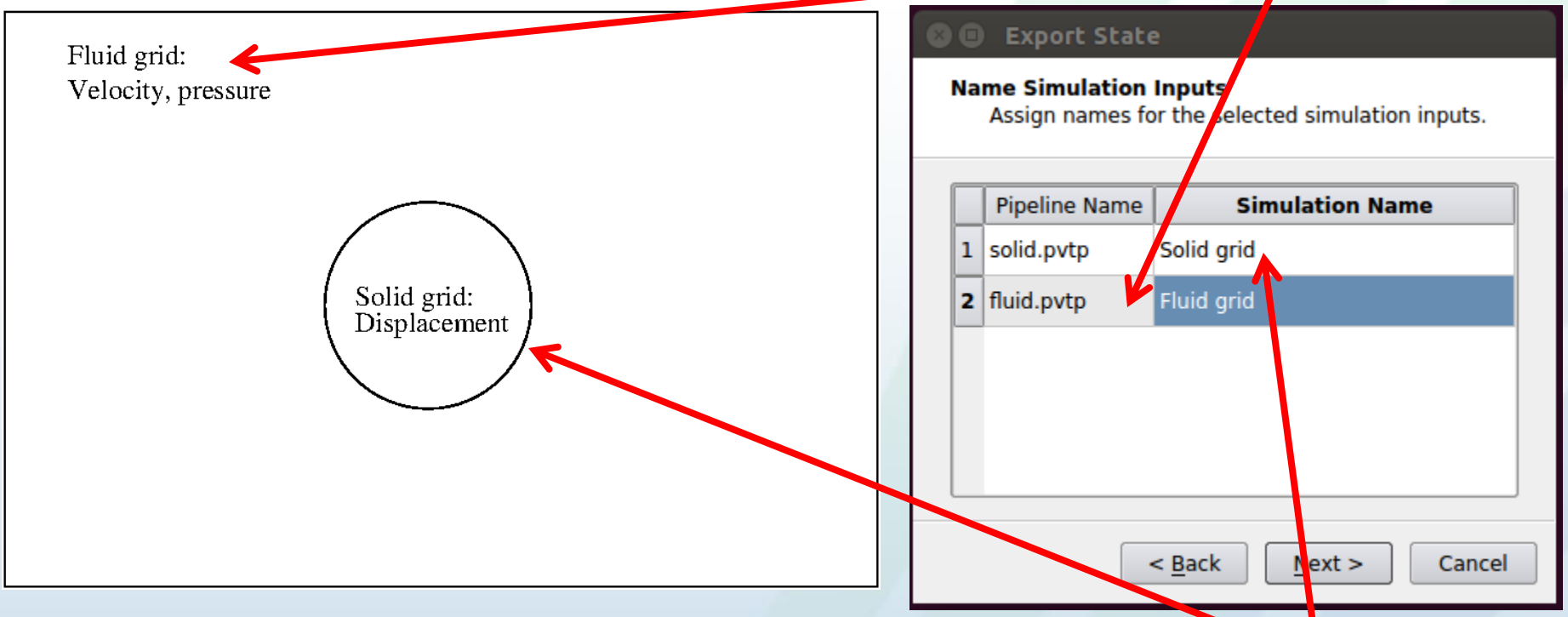
Writing the CoProcess Method: Second Stage

- ParaView classes to look at:
  - vtkCPProcessor – main manager object
  - vtkCPDataDescription – meta-description of what simulation is able to provide
  - vtkCPInputDataDescription – Catalyst description of what it needs in order to execute a pipeline
    - Each input type has a vtkCPInputDataDescription (e.g. one for every physics type in a multi-physics simulation, analysis and debug, etc.)

http://www.paraview.org/ParaView3/Doc/Nightly/html/classvtkCPInputDataDescription.html

vtkCPInputDataDescription “Map”

vtkCPDataDescription::GetInputDescriptionByName(“Fluid grid”)



vtkCPDataDescription::GetInputDescriptionByName(“Solid grid”)

Kitware

vtkCPProcessor CoProcess Step: Stage Two

- vtkCPProcessor
  - int CoProcess(vtkCPDataDescription*)
    - Have the co-processor execute the desired Catalyst pipelines
    - Returns 1 for success and 0 for failure
- vtkCPDataDescription
  - vtkCPInputDataDescription* GetInputDescriptionByName(const char* name)
    - Given an input name (“input”), get the input description object to determine what is needed
- vtkCPInputDataDescription
  - void SetGrid(vtkDataObject*)
    - Set the VTK data source object for a Catalyst pipeline

Example 3

- Goals
  - If Catalyst needs to do some co-processing, create a vtkImageData (C++ and Fortran90 examples) or vtkUnstructuredGrid (Python example) object (we will properly configure this object later)
  - Get the vtkCPInputDataDescription object from the vtkCPDataDescription object
  - Set the grid for the vtkCPInputDataDescription object
- Hints
  - The vtkCPInputDataDescription object is accessed from vtkCPDataDescription with the “input” key
- Result
  - Nothing output yet from Catalyst

Fortran90 Example 3 – Write It!

- Subroutine signature:
  - runcoprocessor(dimensions, step, time, x)
- Starting files:
  - ~/Examples/CatalystExampleCode/ FortranPoissonSolver/FEFortranAdaptor.F90
  - ~/Examples/CatalystExampleCode/ FortranPoissonSolver/FECxxAdaptor.cxx
- Fortran interface at: ~/ParaView4.2/ParaView/CoProcessing/Catalyst/ CAdaptorAPI.h
- Access to C++ Catalyst objects created from Fortran90 calls in vtkCPAdaptorAPI.h
  - static vtkCPDataDescription* GetCoProcessorData();
  - static vtkCPProcessor* GetCoProcessor();

Fortran90 Example 3 – Executable

- Enter “make” at command line to build in ~/FPSBuild
  - Result is FortranPoissonSolver executable
- Running the example
  - ./FortranPoissonSolver
  - Command line arguments is a list of Python scripts
    - E.g. ./FortranPoissonSolver script1.py script2.py
- Nothing extra output from run yet

Fortran90 Example 3 – A Solution (1/2)

~/Examples/CatalystExampleCode/FortranPoissonSolver/example3/FEFortranAdaptor.F90

```
subroutine runcoprocessor(dimensions, step, time, x)
  use iso_c_binding
  implicit none
  integer, intent(in) :: dimensions(3), step
  real(kind=8), dimension(:), intent(in) :: x
  real(kind=8), intent(in) :: time
  integer :: flag, extent(6)
  real(kind=8), DIMENSION(:), allocatable :: xcp(:)

  call requestdatadescription(step,time,flag)
  if (flag .ne. 0) then
    call needtocreategrid(flag)
    call getvtkextent(dimensions, extent)

    if (flag .ne. 0) then
      call createcpimagedata(dimensions, extent)
    end if
    …
  end if
end subroutine runcoprocessor
```

Fortran90 Example 3 – A Solution (2/2)

~/Examples/CatalystExampleCode/FortranPoissonSolver/example3/FECxxAdaptor.cxx

```
extern "C" void createcpimagedata_(int* dimensions, int* extent)
{
 if (!vtkCPPythonAdaptorAPI::GetCoProcessorData())
   {
   vtkGenericWarningMacro("Unable to access CoProcessorData.");
   return;
   }

 // The simulation grid is a 3-dimensional topologically and geometrically
 // regular grid. In VTK/ParaView, this is considered an image data set.
 vtkSmartPointer<vtkImageData> grid = vtkSmartPointer<vtkImageData>::New();

 // Name should be consistent between here, Fortran and Python client script.
 vtkCPPythonAdaptorAPI::GetCoProcessorData()->GetInputDescriptionByName("input")-
>SetGrid(grid);
}
```

C++ Example 3 – Write It!

- Starting file:
  - ~/Examples/miniFE-2.0_ref_Catalyst/adapter/ catalyst_adapter.cpp
- Still using function signature:
  - void coprocess(const double spacing[3], const Box& global_box, const Box& local_box, std::vector<double>& minifepointdata, int time_step, double time, bool force_output);
  - Create a vtkImageData object that we'll set up later
- Build & run steps
  - Rebuild libCatalystMiniFERefAdaptor.so in ~/Examples/miniFE-2.0_ref_Catalyst/adapter_build
  - In ~/Examples/miniFE-2.0_ref_Catalyst/src do./miniFE.x script_names=gridwriter.py

C++ Example 3 – A Solution

~/Examples/miniFE-2.0_ref_Catalyst/adapter/examples/
example3_catalyst_adaptor.cpp

```
void coprocess(const double spacing[3], const Box& global_box,
  const Box& local_box, std::vector<double>& minifepointdata,
  int time_step, double time, bool force_output)
{
  vtkSmartPointer<vtkCPDataDescription> dataDescription =
      vtkSmartPointer<vtkCPDataDescription>::New();
  dataDescription->AddInput("input");
    dataDescription->SetTimeData(time, time_step);
  dataDescription->SetForceOutput(force_output);
  if(Processor->RequestDataDescription(dataDescription) == 0)
    {
     return;
     }
  vtkNew<vtkImageData> grid;
  dataDescription->GetInputDescriptionByName("input")->SetGrid(grid.GetPointer());
  Processor->CoProcess(dataDescription);
…
```

Python Example 3 – Write It!

- Still using the same function signature

```
coProcess(grid, time, step)
```

- Pass the given grid to the data description
- Call coProcess after each computing step

Python Example 3 – A Solution

~/Examples/CatalystExampleCode/PythonDolfinExample/simulation-catalyst-step3.py

```
def coProcess(grid, time, step):
    # initialize data description
    datadescription = vtkPVCatalystPython.vtkCPDataDescription()
    datadescription.SetTimeData(time, step)
    datadescription.AddInput("input")
    cpscript.RequestDataDescription(datadescription)
    inputdescription = datadescription.GetInputDescriptionByName("input")
    if inputdescription.GetIfGridIsNecessary() == False:
        return
    inputdescription.SetGrid(grid)
    cpscript.DoCoProcessing(datadescription)
```

Finally, insert a call to coProcess in the compute loop …

```
while tstep < maxtimestep:
    . . .
    coProcess(ugrid,t,tstep)
    u0.assign(u1)
    t += dt
    tstep += 1
```

Kitware

Writing the CoProcess Method: Second Stage

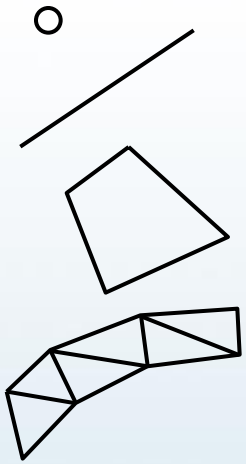
- VTK classes to look at:
  - Depends on simulation's grid type
  - Classes that derive from:
    - vtkDataObject for grids
    - vtkAbstractArray for fields
    - vtkFieldData for managing vtkAbstractArrays
  - Other useful classes:
    - vtkCellType – types of cells for unstructured grids (vtkPolyData and vtkUnstructuredGrid)
- Can look at readers or sources that provide the same type of VTK data object

Getting Data Into Catalyst

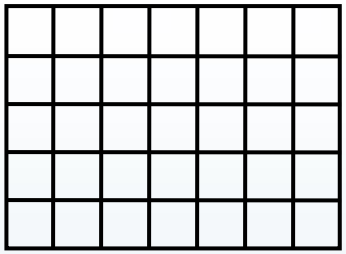
- Main object will derive from vtkDataObject
  - Grids that derive from vtkDataSet
  - Multiblock grids that contain multiple vtkDataSets
- Field (aka attribute) information
  - Point data – information specified for each point in a grid
  - Cell data – information specified for each cell in a grid
  - Field data – meta-data not associated with either points or cells
- All object groups are 0-based/C/C++ indexing

vtkDataSet Subclasses

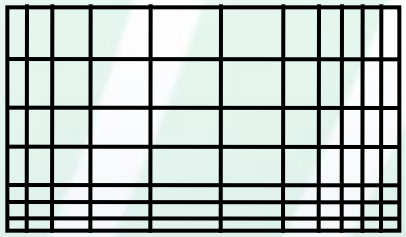
vtkPolyData



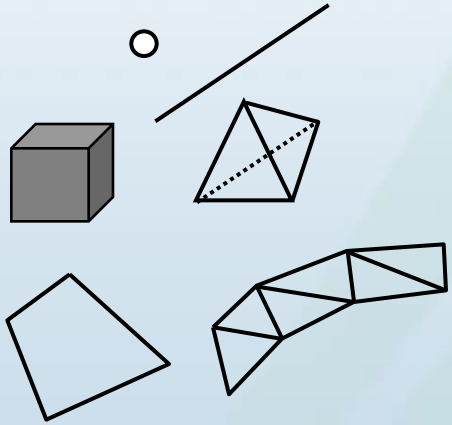
vtkImageData
vtkUniformGrid



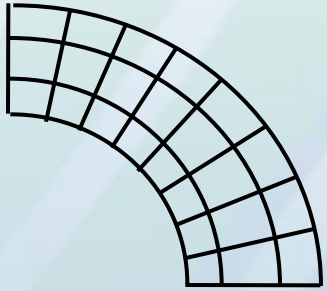
vtkRectilinearGrid



vtkUnstructuredGrid

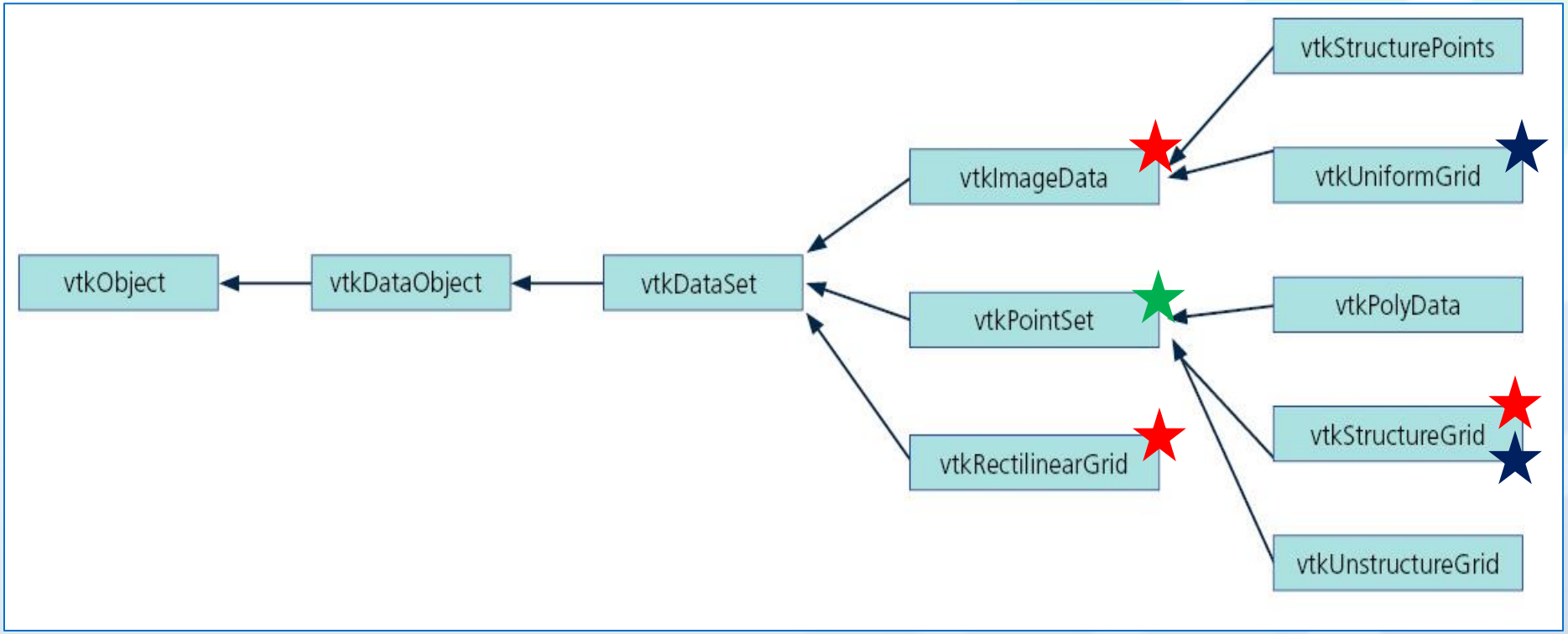


vtkStructuredGrid



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

vtkDataSet Class Hierarchy



★ Topologically regular grid

★ Irregular geometry

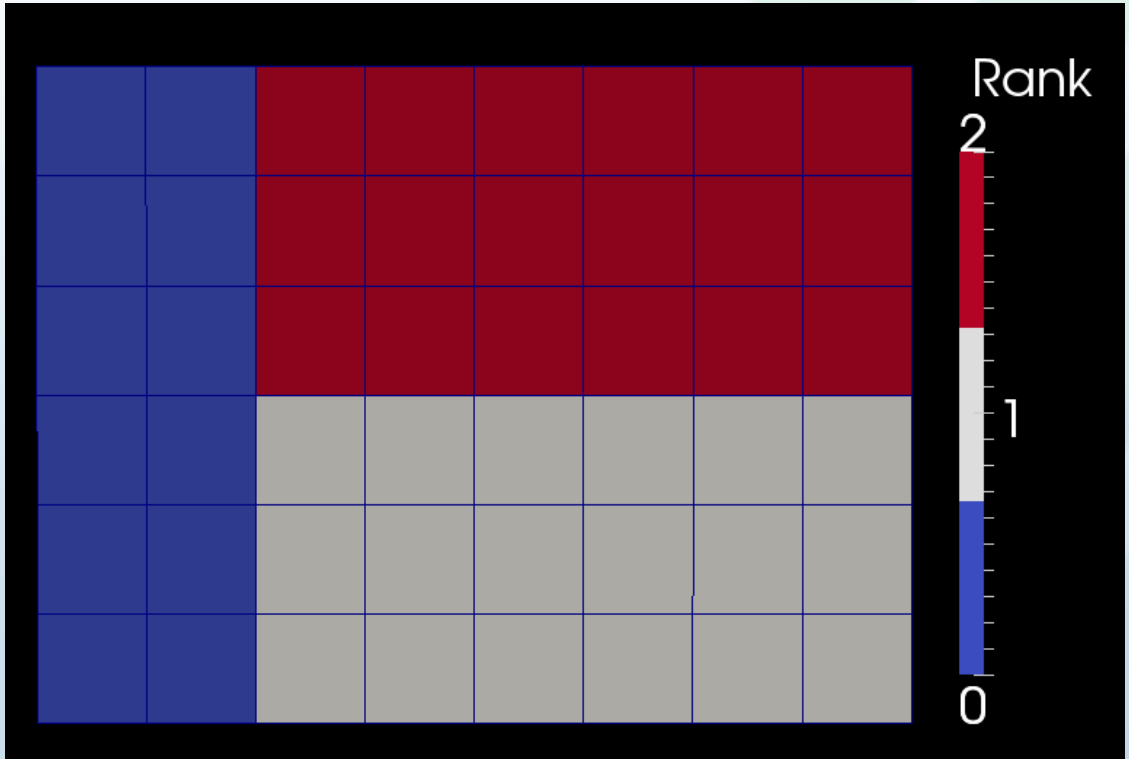
★ Supports blanking

Topologically Regular Grids

- vtkImageData/vtkUniformGrid, vtkRectilinearGrid and vtkStructuredGrid
- Topological structure defined by whole extent
  - Gives first and last point in each logical direction
  - Not required to start at 0
- Partitioning defined by extents
  - First and last point in each logical direction of part of the entire grid (xmin, xmax, ymin, ymax, zmin, zmax)
- Ordering of points and cells is fastest in logical x-direction, then y-direction and slowest in z-direction

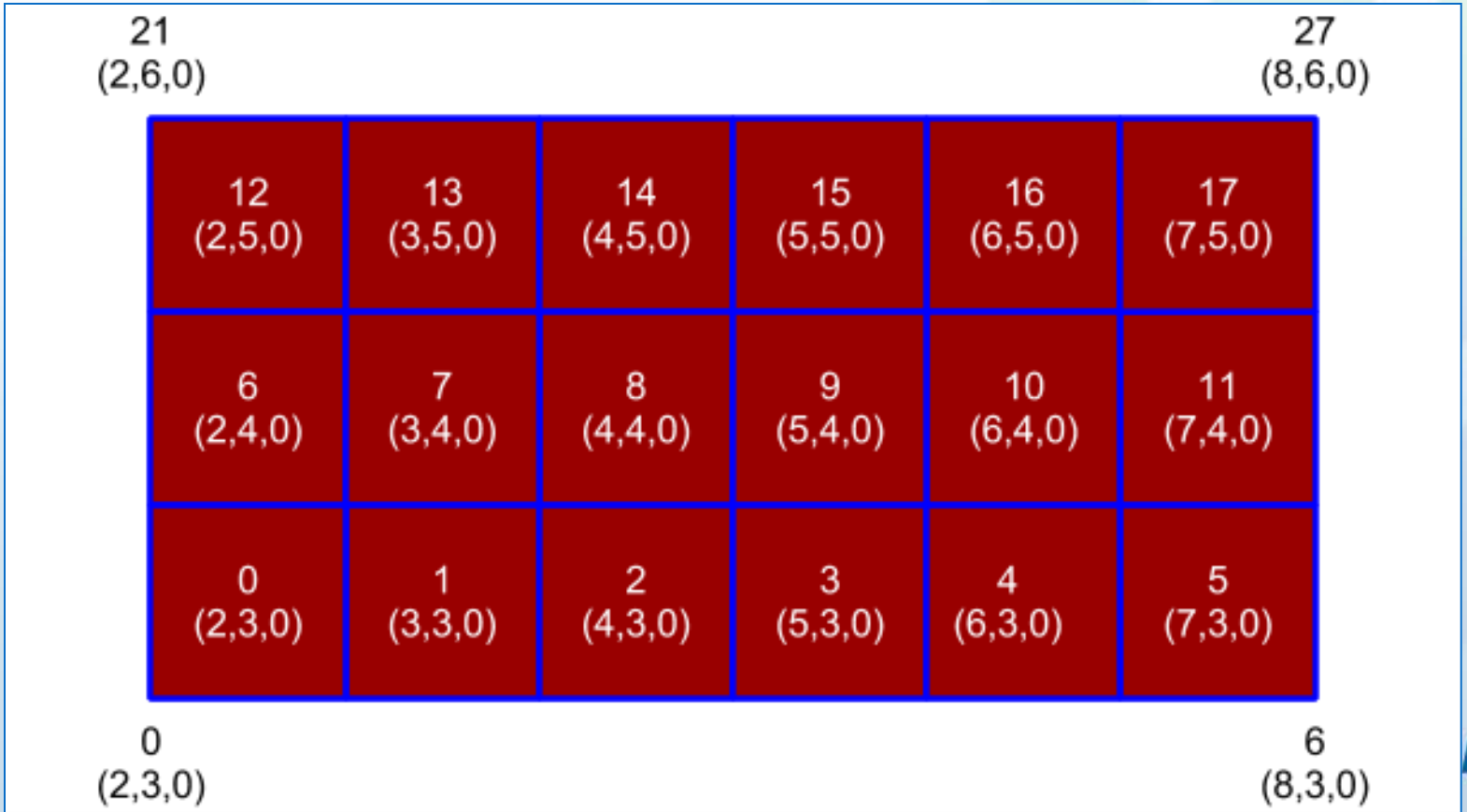
Extents

- Whole extent for all processes (0, 8, 0, 6, 0, 0)
- Extents different for each process
  - Rank 0: (0, 2, 0, 6, 0, 0), 21 points, 12 cells
  - Rank 1: (2, 8, 0, 3, 0, 0), 28 points, 18 cells
  - Rank 2: (2, 8, 3, 6, 0, 0), 28 points, 18 cells



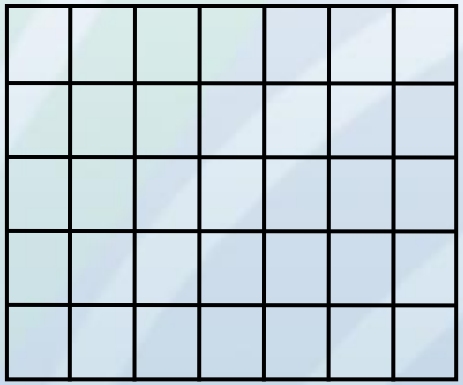
Extents

- Point and cell indices for extent of (2, 8, 3, 6, 0, 0)
  - From rank 2 in previous slide
  - 28 points and 18 cells



vtkImageData/vtkUniformGrid

- vtkCPInputDataDescription::SetWholeExtent() – total number of points in each direction
- SetExtent() – a process's part of the whole extent
- SetSpacing() – cell lengths in each direction (same for all processes)
- SetOrigin() – location of point at i=0, j=0, k=0 (same for all processes)



- vtkUniformGrid
  - Supports cell blanking
  - Currently written to file as vtkImageData

http://www.vtk.org/doc/nightly/html/classvtkUniformGrid.html

Example 4

- Goals
  - Build the partitioned VTK grid and set vtkCPInputDataDescription's whole extent for structured grids
- Result
  - The grid is now available to output from Catalyst

Fortran90 Example 4 – Write It!

- Function signature:
  - extern "C" void createcpimagedata_(int* dimensions,
    int* extent)
- Starting file:
  - ~/Examples/CatalystExampleCode/
    FortranPoissonSolver/FECxxAdaptor.cxx
  - Local extent is contained in extent:
    - Points range from extent[0] to extent[1] in X-direction, extent[2] to extent[3] in Y-direction, extent[4] to extent[5] in Z-direction
  - Global extent can be determined from dimensions:
    - dimensions only has the maximum extent for each direction
- Access to C++ Catalyst objects created from Fortran90 calls in vtkCPAdaptorAPI.h
  - static vtkCPDataDescription* GetCoProcessorData();
  - static vtkCPProcessor* GetCoProcessor();

Fortran90 Example 4 – Executable

- Enter “make” at command line to build in ~/FPSBuild
  - Result is FortranPoissonSolver executable
- Running the example
  - ./FortranPoissonSolver ~/Examples/CatalystExampleCode/SampleScripts/gridwriter.py
  - Will output filename_*.pvti files with vtkImageData
- Will only contain grid output

Fortran90 Example 4 – A Solution

~/Examples/CatalystExampleCode/FortranPoissonSolver/example4/FECxxAdaptor.cxx

```
extern "C" void createcpimagedata_(int* dimensions, int* extent)
{
  if (!vtkCPPythonAdaptorAPI::GetCoProcessorData())
    {
    vtkGenericWarningMacro("Unable to access CoProcessorData.");
    return;
    }

  // The simulation grid is a 3-dimensional topologically and geometrically
  // regular grid. In VTK/ParaView, this is considered an image data set.
  vtkSmartPointer<vtkImageData> grid = vtkSmartPointer<vtkImageData>::New();

  grid->SetExtent(extent[0]-1, extent[1]-1, extent[2]-1,
                  extent[3]-1, extent[4]-1, extent[5]-1);
  grid->SetSpacing(1./(dimensions[0]-1), 1./(dimensions[1]-1), 1./(dimensions[2]-1));

  // Name should be consistent between here, Fortran and Python client script.
  vtkCPPythonAdaptorAPI::GetCoProcessorData()->GetInputDescriptionByName("input")->SetGrid(grid);
  vtkCPPythonAdaptorAPI::GetCoProcessorData()->GetInputDescriptionByName("input")->SetWholeExtent(
    0, dimensions[0]-1, 0, dimensions[1]-1, 0, dimensions[2]-1);
}
```

C++ Example 4 – Write It!

- Starting file:
  - ~/Examples/miniFE-2.0_ref_Catalyst/adapter/ catalyst_adapter.cpp
- Still using function signature:
  - void coprocess(const double spacing[3], const Box& global_box, const Box& local_box, std::vector<double>& minifepointdata, int time_step, double time, bool force_output);
  - Geometry is unit box so Origin is at [0,0,0]
  - Local extent is contained in local_box:
    - Points range from local_box[0][0] to local_box[0][1] in X-direction, local_box[1][0] to local_box[1][1] in Y-direction, local_box[2][0] to local_box[2][1] in Z-direction
  - Global extent is contained in global_box:
    - Points range from global_box[0][0] to global_box[0][1] in X-direction, global_box[1][0] to global_box[1][1] in Y-direction, global_box[2][0] to global_box[2][1] in Z-direction

C++ Example 4 – Build & Execute

- Build step
  - Rebuild libCatalystMiniFERefAdaptor.so in ~/Examples/miniFE-2.0_ref_Catalyst/adapter_build
- Run step
  - In ~/Examples/miniFE-2.0_ref_Catalyst/src do ./miniFE.x script_names=gridwriter.py
  - Will output filename_*.pvti files with vtkImageData

C++ Example 4 – A Solution

~/Examples/miniFE-2.0_ref_Catalyst/adapter/examples/
example4_catalyst_adaptor.cpp

```
…
  vtkNew<vtkImageData> grid;
  int extent[6] = {local_box[0][0], local_box[0][1], local_box[1][0],
                   local_box[1][1], local_box[2][0], local_box[2][1]};
  grid->SetExtent(extent);
  grid->SetSpacing(spacing[0], spacing[1], spacing[2]);
  grid->SetOrigin(0, 0, 0);
  dataDescription->GetInputDescriptionByName("input")->SetGrid(grid.GetPointer());
  int wholeExtent[6] = {global_box[0][0], global_box[0][1], global_box[1][0],
                        global_box[1][1], global_box[2][0], global_box[2][1]};
  dataDescription->GetInputDescriptionByName("input")->
     SetWholeExtent(wholeExtent);
…
```

Python Example 4 – Write It!

- We treat only the unstructured grid case
- Be aware of efficiency issues with python
- Proposed solution is not optimal
- Write a function that parses a Dolfin mesh description and builds out a *vtkUnstructuredGrid*

Python Example 4 – A Solution

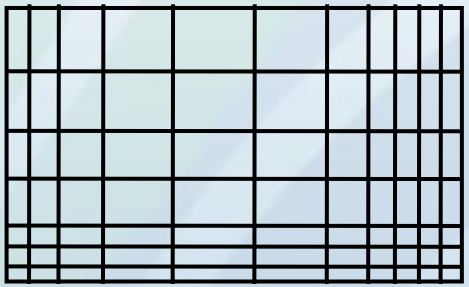
~/Examples/CatalystExampleCode/PythonDolfinExample/simulation-catalyst-step4.py

```
def Mesh2VTKUGrid(mesh):
    npoints = mesh.num_vertices() ; geom = mesh.geometry()
    pts=vtk.vtkPoints() ; pts.SetNumberOfPoints(npoints)
    for i in xrange(npoints):
        p=geom.point(i)
        pts.SetPoint(i,p.x(),p.y(),p.z())
    dim = mesh.topology().dim() ; ncells = mesh.num_cells()
    cells = vtk.vtkCellArray()
    cellTypes = vtk.vtkUnsignedCharArray()
    cellTypes.SetNumberOfTuples(ncells)
    cellLocations = vtk.vtkIdTypeArray()
    cellLocations.SetNumberOfTuples(ncells)
    loc=0
    for (cell,i) in zip(mesh.cells(),xrange(ncells)) :
        ncellpoints=len(cell) ; cells.InsertNextCell(ncellpoints)
        for cpoint in cell: cells.InsertCellPoint(cpoint)
        cellTypes.SetTuple1(i,vtkcelltypes[dim][ncellpoints])
        cellLocations.SetTuple1(i,loc) ; loc+=1+ncellpoints
    ugrid = vtk.vtkUnstructuredGrid()
    ugrid.SetPoints(pts) ; ugrid.SetCells(cellTypes,cellLocations,cells)
return ugrid
```

Kitware

vtkRectilinearGrid

- vtkCPInputDataDescription::SetWholeExtent() – total number of points in each direction
- SetExtents() – a process's part of the whole extent
- Set<X,Y,Z>Coordinates() – point coordinates in each direction
  - Only values for process's extents
  - Index starting at 0



http://www.vtk.org/doc/nightly/html/classvtkRectilinearGrid.html

vtkPointSet

- Abstract class for grids with arbitrary point locations
  - Derived classes: vtkStructuredGrid, vtkPolyData, vtkUnstructuredGrid
- SetPoints(vtkPoints*) – set the number and coordinates of a process's points
- vtkPoints
  - Class used to store actual point location array
  - C/C++ indexing
  - SetDataTypeTo<Int,Float,Double>()
  - SetNumberOfPoints() – allocation
  - SetPoint() – fast but doesn't do bounds checking
  - InsertPoint()/InsertNextPoint() – safe but does bounds checking and reallocation as needed

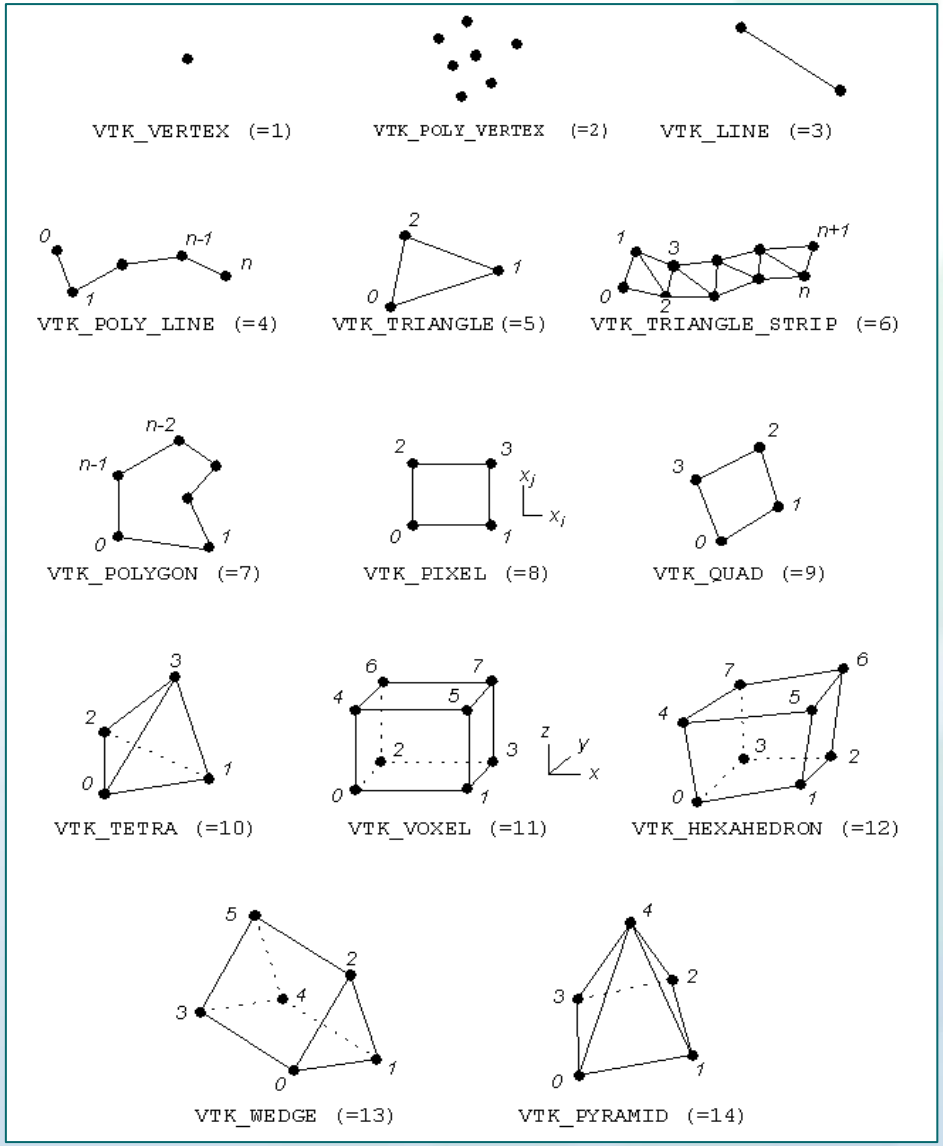
http://www.vtk.org/doc/nightly/html/classvtkPointSet.html

vtkStructuredGrid

- vtkCPInputDataDescription::SetWholeExtent() – total number of points in each direction
- SetExtents() – a process's part of the whole extent
- Ordering for points and cells
  - Fastest in x-direction
  - Next in y-direction
  - Slowest in z-direction
  - vtkPoints must be ordered in same manner

http://www.vtk.org/doc/nightly/html/classvtkStructuredGrid.html

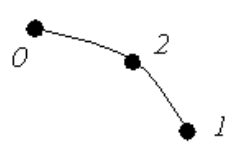
vtkCell Straight-Edged Types



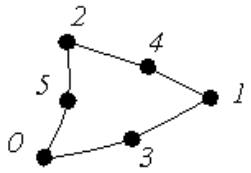
http://www.vtk.org/doc/nightly/html/classvtkCell.html

Kitware

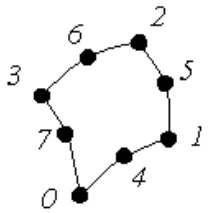
vtkCell Curvilinear-Edge Types



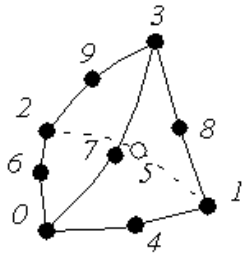
Quadratic Edge



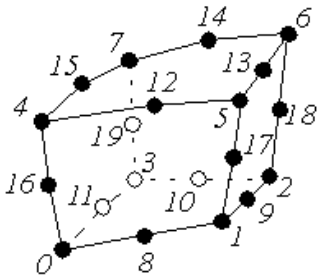
Quadratic Triangle



Quadratic Quadrilateral



Quadratic Tetrahedron

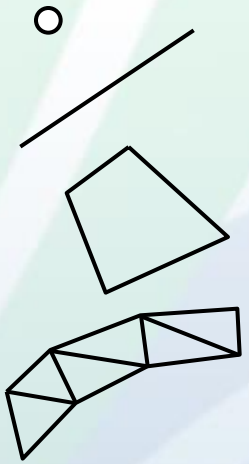


Quadratic Hexahedron

Kitware

vtkPolyData Cells

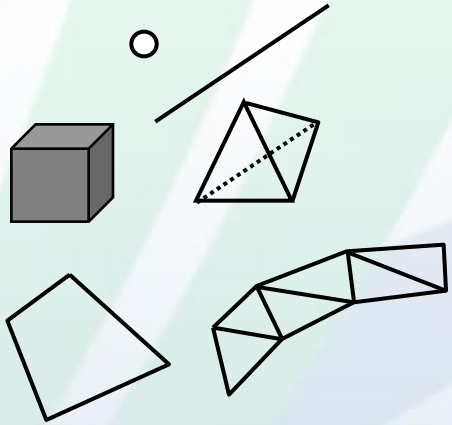
- Efficient way – build through vtkCellArray
  - vtkCellArray::SetNumberOfCells()
  - Done for verts, lines, polys, triangle strips
- Simple way – build through poly data API
  - Allocate()
  - InsertNextCell()
  - Squeeze() – reclaim unused memory
- Cells are ordered by their insertion order



http://www.vtk.org/doc/nightly/html/classvtkPolyData.html

vtkUnstructuredGrid Cells

- Order of cells is order of insertion
  - 0-based indexing
- Allocate()
- InsertNextCell()
- SetCells()
- Squeeze()
- Inserted points of cells are 0-based indexing



http://www.vtk.org/doc/nightly/html/classvtkUnstructuredGrid.html

Example 5 (C++) – Write It!

- Still using function signature:
  - void coprocess(const double spacing[3], const Box& global_box, const Box& local_box, std::vector<double>& minifepointdata, int time_step, double time, bool force_output);
- Hint: example5_catalyst_adaptor.cpp
  - void getpointcoordinate(const int indices[3], const double spacing[3], double coord[3]);
  - void getcellpointids(const Box& local_box, const int indices[3], vtkIdType pointids[8]);

Example 5 (C++) – A Solution

~/Examples/miniFE-2.0_ref_Catalyst/adapter/examples/
example5_catalyst_adaptor.cpp

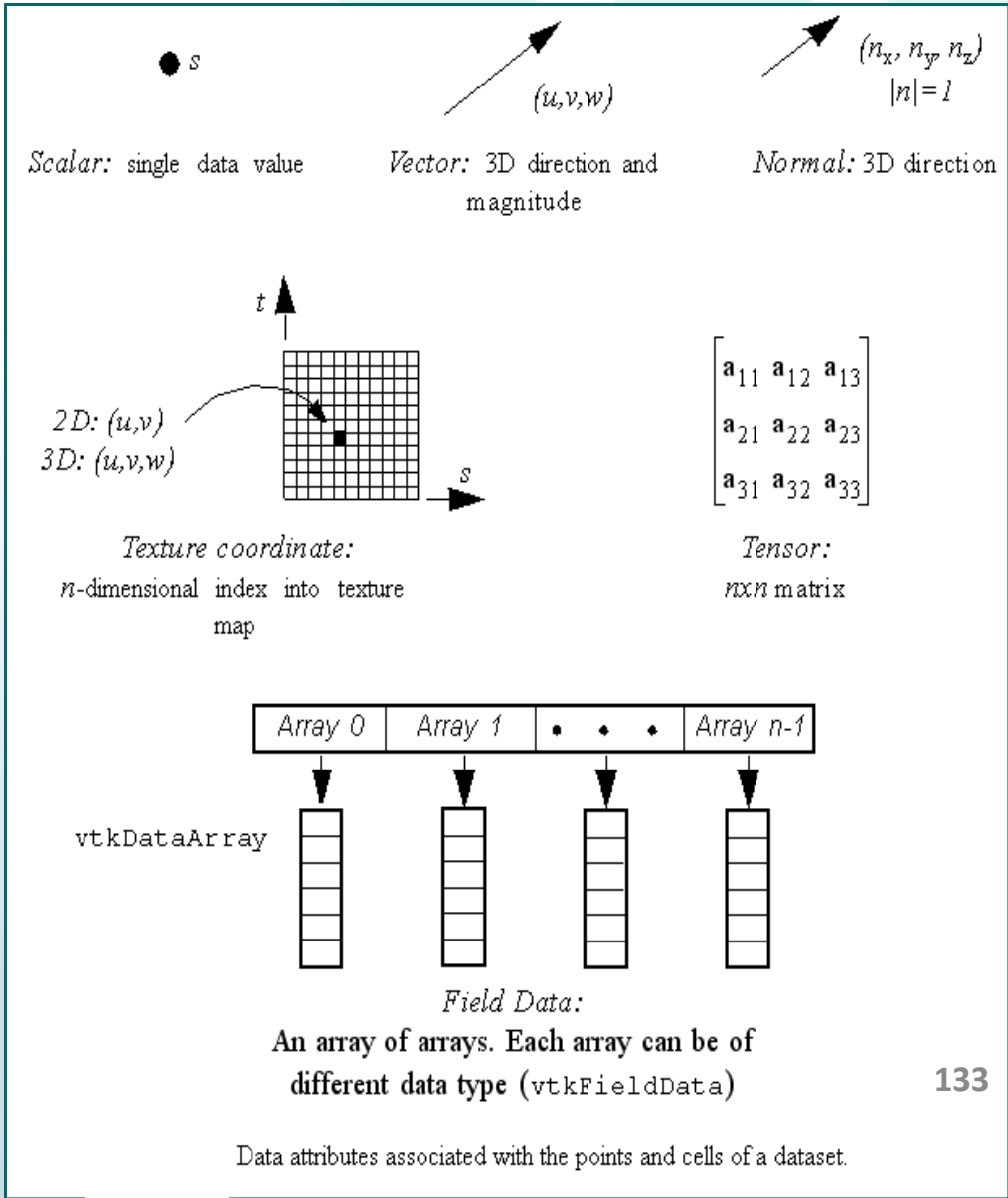
```
…
  vtkNew<vtkUnstructuredGrid> grid;
  grid->Initialize();
  vtkNew<vtkPoints> points;
  points->SetNumberOfPoints( (local_box[0][1]-local_box[0][0]+1) *
    (local_box[1][1]-local_box[1][0]+1)*(local_box[2][1]-local_box[2][0]+1) );
  double coord[3]; int indices[3]; vtkIdType id=0;
  for(int iz=local_box[2][0]; iz<=local_box[2][1]; ++iz)
      for(int iy=local_box[1][0]; iy<=local_box[1][1]; ++iy)
        for(int ix=local_box[0][0]; ix<=local_box[0][1]; ++ix) {
          int indices[3] = {ix, iy, iz};
          getpointcoordinate(indices, spacing, coord);
          points->SetPoint(id++, coord);
          }
    grid->SetPoints(points.GetPointer());
…
```

Example 5 (C++) – A Solution (2)

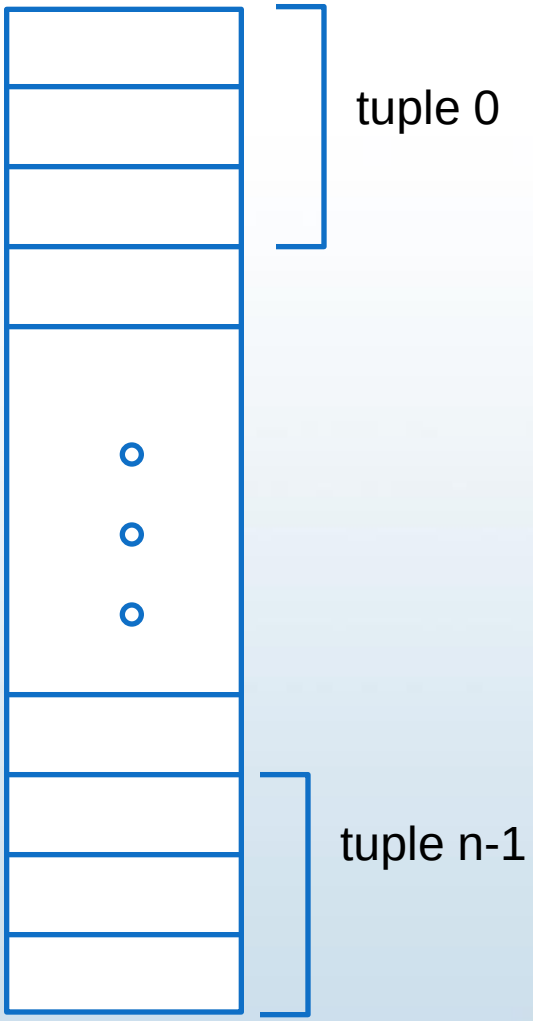
```
…
  grid->Allocate( (local_box[0][1]-local_box[0][0]) *
     (local_box[1][1]-local_box[1][0])*(local_box[2][1]-local_box[2][0]) );
    vtkIdType pointids[8];
    for(int iz=local_box[2][0]; iz<local_box[2][1]; ++iz)
      for(int iy=local_box[1][0]; iy<local_box[1][1]; ++iy)
        for(int ix=local_box[0][0]; ix<local_box[0][1]; ++ix) {
          int indices[3] = {ix, iy, iz};
          getcellpointids(local_box, indices, pointids);
          grid->InsertNextCell(VTK_HEXAHEDRON, 8, pointids);
        }
…
```

Field Information

- Store information defined over grids
- Stored in concrete classes that derive from vtkDataArray
  - vtkFloatArray
  - vtkIntArray
  - vtkDoubleArray
  - vtkUnsignedCharArray
  - …



vtkDataArray – Basis for vtkDataObject Contents



- An array of *n* tuples
- Each tuple has *m* components which are logically grouped together
- Internal implementation is a pointer to an n x m block of memory
- Data type determined by class

- Two APIs : generic and data type specific

http://www.vtk.org/doc/nightly/html/classvtkDataArray.html

vtkDataArray

- SetNumberOfComponents() – call first
- SetNumberOfTuples()
  - For point data must be set to number of points
  - For cell data must be set to number of cells
- SetArray()
  - If flat array has proper component & tuple ordering use existing simulation memory – most efficient
  - Can specify who should delete
  - VTK uses pipeline architecture so Catalyst libraries will NOT modify array values
- SetTupleValue() – uses native data type
- SetValue() – uses native data type
- SetName() – array descriptor, e.g. velocity, density

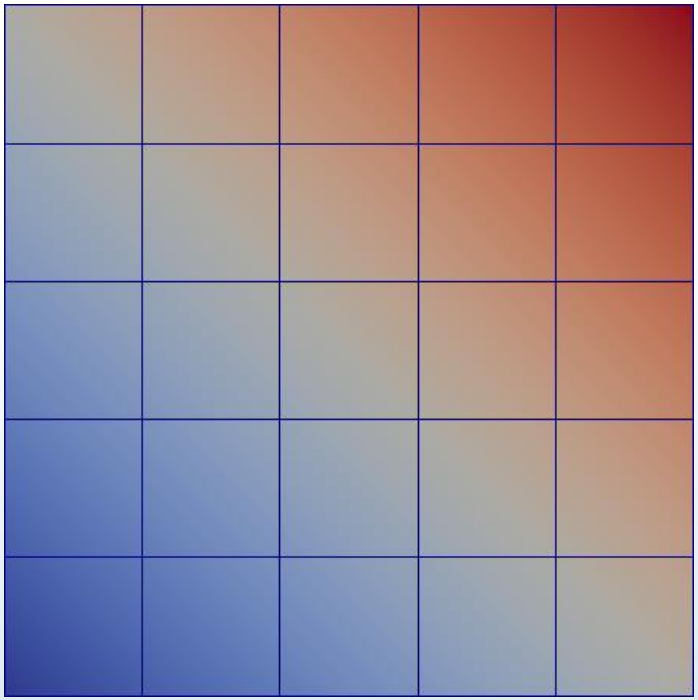
http://www.vtk.org/doc/nightly/html/classvtkDataArray.html

vtkFieldData

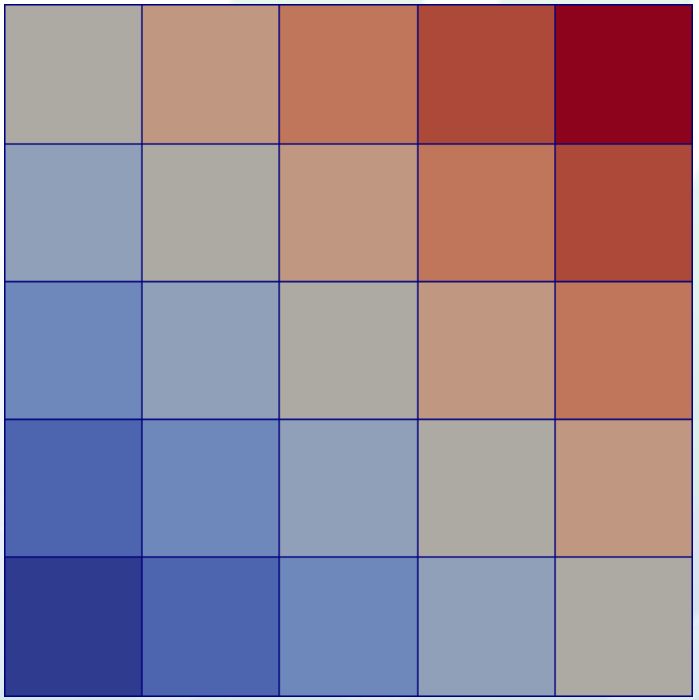
- Object for storing vtkDataArrays
- vtkDataSet::GetFieldData() – arrays not associated with points or cells
- Derived classes
  - vtkPointData – vtkDataSet::GetPointData()
  - vtkCellData – vtkDataSet::GetCellData()
- vtkFieldData::AddArray(vtkDataArray*)
- Specific arrays are normally retrieved by name from vtkFieldData
  - Uniqueness is not required but can cause unexpected results

http://www.vtk.org/doc/nightly/html/classvtkFieldData.html

Point Data and Cell Data



Point data – 36 values



Cell data – 25 values

Example 6

- Goals
  - Create the field data arrays
  - Associate the field data arrays with the grid's vtkPointData array container
- Hints
  - vtkDataSet::GetNumberOfPoints() can be used to determine the number of tuples in these arrays
- Result
  - Adapter is complete and will have all grid and field information available to output

Fortran90 Example 6 – Write It!

- Subroutine signature:
  - runcoprocessor(dimensions, step, time, x)
- Starting files:
  - ~/Examples/CatalystExampleCode/
    FortranPoissonSolver/FEFortranAdaptor.F90
  - ~/Examples/CatalystExampleCode/
    FortranPoissonSolver/FECxxAdaptor.cxx
- Use **xcp** vector for source in FEFortranAdaptor.F90
  - Already ordered for our grid's points
- Access to C++ Catalyst objects created from Fortran90 calls in vtkCPAdaptorAPI.h
  - static vtkCPDataDescription* GetCoProcessorData();
  - static vtkCPProcessor* GetCoProcessor();

Fortran90 Example 6 – Executable

- Enter “make” at command line to build in ~/FPSBuild
  - Result is FortranPoissonSolver executable
- Running the example
  - ./FortranPoissonSolver ~/Examples/CatalystExampleCode/SampleScripts/gridwriter.py
  - Will output filename_*.pvti files with vtkImageData
- Will contain grid and field output

Fortran90 Example 6 – A Solution (1/2)

~/Examples/CatalystExampleCode/FortranPoissonSolver/example6/FEFortranAdaptor.F90

```
subroutine runcoprocessor(dimensions, step, time, x)
 …
! x is the array with global values, we need just this process's
    ! values for Catalyst which will be put in xcp
    allocate(xcp((extent(2)-extent(1)+1)*(extent(4)-extent(3)+1)*(extent(6)-extent(5)+1)))
    call getlocalfield(dimensions, extent, x, xcp)

    ! adding //char(0) appends the C++ terminating character
    ! to the Fortran array
    call addfield(xcp,"solution"//char(0))
    call coprocess()
    deallocate(xcp)
  end if
end subroutine runcoprocessor
```

Fortran90 Example 6 – A Solution (2/2)

~/Examples/CatalystExampleCode/FortranPoissonSolver/example6/FECxxAdaptor.cxx

```
extern "C" void addfield_(double* scalars, char* name)
{
  vtkCPInputDataDescription* idd =
    vtkCPPythonAdaptorAPI::GetCoProcessorData()->GetInputDescriptionByName("input");
  vtkImageData* image = vtkImageData::SafeDownCast(idd->GetGrid());
  if (!image)
    {
    vtkGenericWarningMacro("No adaptor grid to attach field data to.");
    return;
    }
  // field name must match that in the fortran code.
  if (idd->IsFieldNeeded(name))
    {
    vtkSmartPointer<vtkDoubleArray> field = vtkSmartPointer<vtkDoubleArray>::New();
    field->SetName(name);
    field->SetArray(scalars, image->GetNumberOfPoints(), 1);
    image->GetPointData()->AddArray(field);
    }
}
```

C++ Example 6 – Write It!

- Starting file:
  - ~/Examples/miniFE-2.0_ref_Catalyst/adapter/
    catalyst_adapter.cpp
- Still using function signature:
  - void coprocess(const double spacing[3], const Box& global_box, const Box& local_box, std::vector<double>& minifepointdata, int time_step, double time, bool force_output);
- Use **vtkpointdata** vector for source
  - Already ordered for our grid's points

C++ Example 6 – Build & Execute

- Build step
  - Rebuild libCatalystMiniFERefAdaptor.so in ~/Examples/miniFE-2.0_ref_Catalyst/adapter_build
- Run step
  - In ~/Examples/miniFE-2.0_ref_Catalyst/src do ./miniFE.x script_names=gridwriter.py
  - Will output filename_*.pvti files with vtkImageData that now has field data attached

C++ Example 6 – A Solution

~/Examples/miniFE-2.0_ref_Catalyst/adapter/examples/
example6_catalyst_adaptor.cpp

```
…
  std::vector<double> vtkpointdata;
  getlocalpointarray(global_box, local_box, minifepointdata, vtkpointdata);
  vtkSmartPointer<vtkDoubleArray> myDataArray =
      vtkSmartPointer<vtkDoubleArray>::New();
  myDataArray->SetNumberOfComponents(1);
  myDataArray->SetName("myData");
  myDataArray->SetArray(&(vtkpointdata[0]), vtkpointdata.size(), 1);
  grid->GetPointData()->AddArray(myDataArray);
…
```

Python Example 6 – Write It!

- Write a function to add point and cell arrays to the grid
- Use Dolfin vector for source
  - Based on numpy vectors
  - Ordered as the cells in the grid
  - In case of multidimensional arrays (vector fields) components are serialized : x1,y1,z1, x2,y2,z2, in a 1D Dolfin vector

Python Example 6 – A Solution (1/2)

~/Examples/CatalystExampleCode/PythonDolfinExample/simulation-catalyst-step6.py

First, write a function to convert Dolfin vectors to n-component VTK arrays

```
def Values2VTKArray(values,n,name):
        ncomps=len(values)/n
        array=vtk.vtkDoubleArray()
        array.SetNumberOfComponents(ncomps)
        array.SetNumberOfTuples(n)
        i=0
        for x in values:
                array.SetValue(i,x)
                i+=1
        array.SetName(name)
        return array
```

Kitware

Python Example 6 – A Solution (2/2)

Then, we write a function adding arrays to the VTK grid

```
def AddFieldData(ugrid, pointArrays, cellArrays ):
    npoints = ugrid.GetNumberOfPoints()
    for (name,values) in pointArrays:
        ugrid.GetPointData().AddArray(Values2VTKArray(values,npoints,name))
    ncells = ugrid.GetNumberOfCells()
    for (name,values) in cellArrays:
        ugrid.GetCellData().AddArray( Values2VTKArray(values,ncells,name) )
```

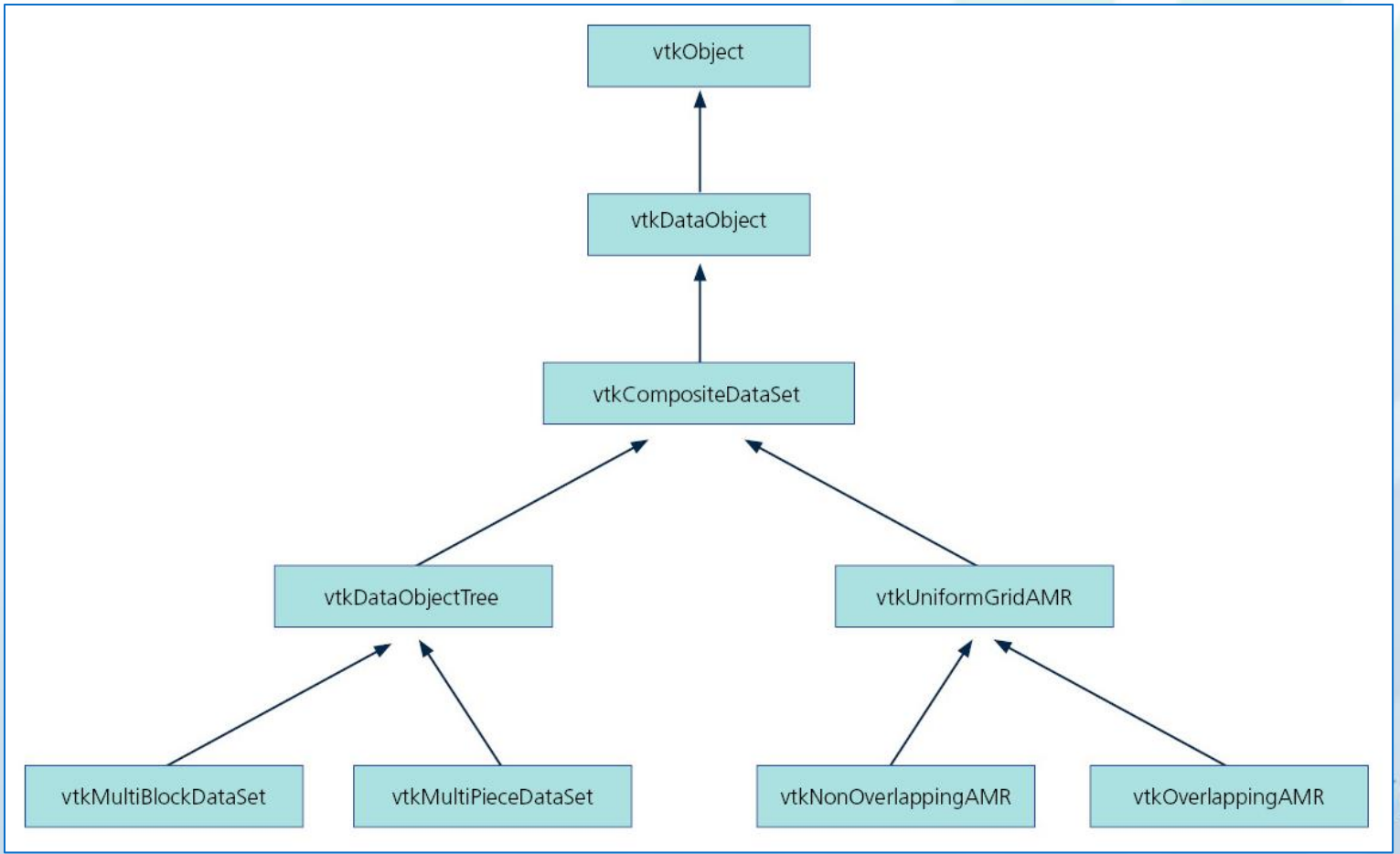
Finally, before calling *AddFieldData*, we request Dolfin to produce flattened representation of fields

```
while tstep < maxtimestep:
    . . .
    ugrid = Mesh2VTKUGrid( u1.function_space().mesh() )
    velocity = u1.compute_vertex_values()
    pressure = p1.compute_vertex_values()
    AddFieldData( ugrid, [ ("Velocity",velocity) , ("Pressure",pressure) ] , [] )
    coProcess(ugrid,t,tstep)
```

Kitware

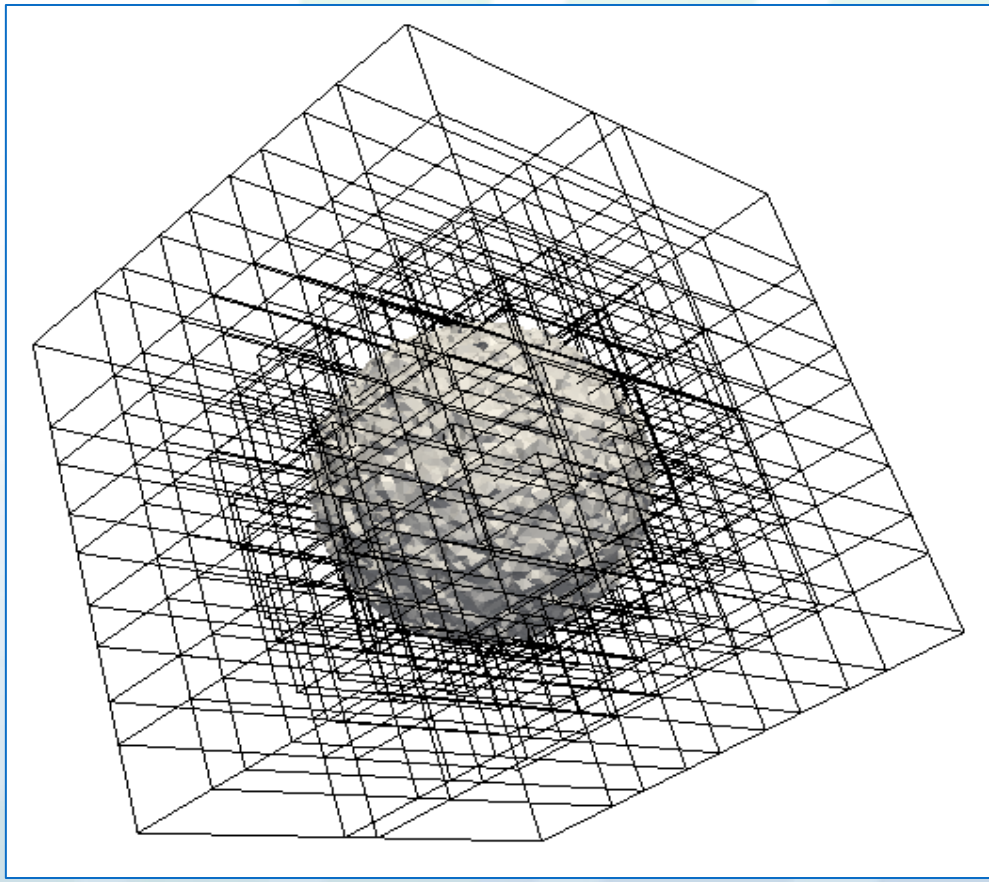
Multiple Grids

- Use something that derives from vtkCompositeDataSet to group multiple vtkDataSets together



vtkMultiBlockDataSet

- Most general way of storing vtkDataSets and other vtkCompositeDataSets
- Block structure must be the same on each process
- Block is null if data set is on a different process
- SetNumberOfBlocks()
- SetBlock()



http://www.vtk.org/doc/nightly/html/classvtkMultiBlockDataSet.html

vtkMultiPieceDataSet

- Way to combine multiple vtkDataSets of the same type into a single logical object
  - Useful when the amount of pieces is unknown *a priori*
- Must be contained in a vtkMultiBlockDataSet
- SetNumberOfPieces()
- SetPiece()

http://www.vtk.org/doc/nightly/html/classvtkMultiPieceDataSet.html

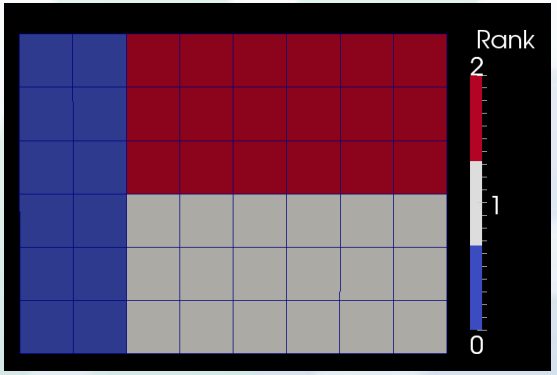
AMR Data Sets

- Classes derive from vtkUniformGridAMR
- Concept of grid levels for changing cell size locally
- Uses only vtkUniformGrids
  - vtkNonOverlappingAMR – no blanking used since no data sets overlap
  - vtkOverlappingAMR – vtkUniformGrids overlap and use blanking

http://www.vtk.org/doc/nightly/html/classvtkUniformGridAMR.html

Grid Partitioning

- For unstructured grids and polydatas
  - Single data set per process – use as is
  - Multiple data sets per process – choice of combining or not depends on memory layout



- For topologically structured grids
  - Must be partitioned into logical blocks
  - SetExtent()
    - Index of first and last point in each direction
    - Will be different on each process
  - vtkCPInputDataDescription::SetWholeExtent() – same on each process

Advanced Topics

- Zero-copy for VTK arrays
  - Reuse existing memory in layout different than standard VTK arrays
- Hard-coded C++ pipelines
  - Removes dependency on Python
  - Use simulation input parameters to remove requirement that users create pipelines in ParaView
- Ghost information
- CMake for automatic C++ and Fortran function name mangling

Building ParaView Catalyst

Build Considerations Overview

- Shared library build
  - Easier for development but can kill the IO system on HPC machines for large scale runs
- Static library build
  - Freezing Python allows it to be built statically into the executable instead of having it behave similar to shared libraries
- Mesa
  - When window manager isn't available or don't want render windows popping up for image generation

ParaView Catalyst Editions

- Doing a full ParaView build gives full Catalyst functionality
  - Also can add 150 MB or more to the executable size depending on ParaView configuration
- Most people use a small subset of VTK and ParaView filters for analysis and visualization
- Catalyst editions allow simpler ways to remove unneeded functionality
  - Can get Catalyst library size down to 15 to 30 MB depending on configuration

Preconfigured ParaView Catalyst Editions

- Base – a minimal set of functionality to build other editions from but not meant for use on its own
- Essentials – base + several of the most popular filters
- Extras – base + essentials + parallel XML writers and a couple more filters
- Rendering – coming soon…
- Python version of each available as well
- Available at www.paraview.org/download

Online Help

- ParaView Catalyst User's Guide:
  - http://paraview.org/Wiki/images/4/48/CatalystUsersGuide.pdf
- Email list:
  - paraview@paraview.org
- Doxygen:
  - http://www.vtk.org/doc/nightly/html/classes.html
  - http://www.paraview.org/ParaView3/Doc/Nightly/html/classes.html
- Sphinx:
  - http://www.paraview.org/ParaView3/Doc/Nightly/www/py-doc/index.html
- Websites:
  - http://www.paraview.org
  - http://www.paraview.org/in-situ/
- Examples:
  - https://github.com/Kitware/ParaViewCatalystExampleCode