SOME ASSEMBLY REQUIRED

IN-SITU VISUALIZATION WITH AMREX

Juliana Kwan (DAMTP, Cambridge)

PROJECT OVERVIEW

- ExCALIBUR project: *"In-situ visualization and unified programming across accelerator architectures at exascale"*
- Aims:
  - Convert our existing numerical relativity code (GRChombo) to run on GPUs for the new generation of HPC clusters (GRTeclyn).
  - Incorporate in-situ visualization into GRTeclyn for analysis and dissemination purposes on exascale systems.

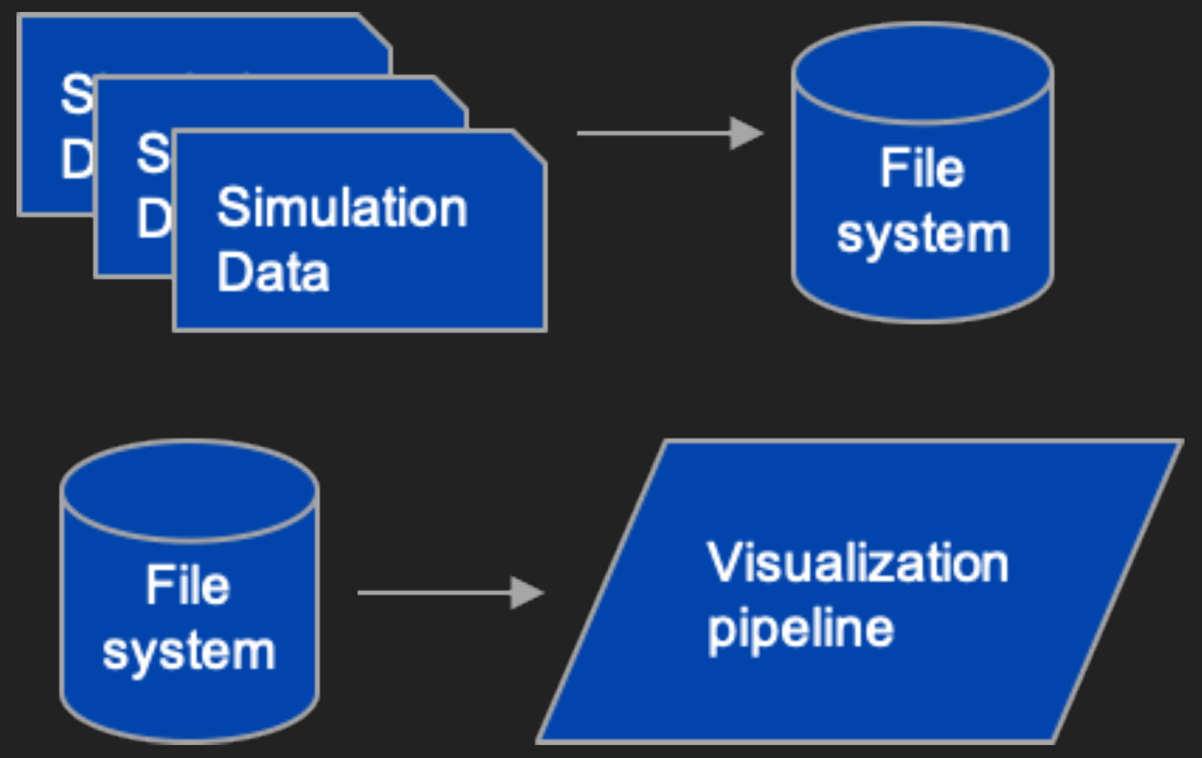
THE GRTL COLLABORATION

- Testing the Limits of General Relativity - collaboration between Queen Mary University of London, King's College, and Cambridge.

- Intel collaborators as well: Carson Brownlee, Dave DeMarle, Maxwell Cai
- Our GitHub: https://github.com/GRTLCollaboration/GRTeclyn
- http://www.grchombo.org/

WHY IN-SITU VISUALIZATION?

- Storage systems have not kept up with advances in flop rates.
- Aim: produce visualizations on the fly to reduce demand on read/write. Also great for debugging.
- GRChombo’s implementation used an earlier version of Catalyst requiring a direct implementation of VTK containers.
- ParaView (among others) can hook into AMReX to capture data from AMR mesh for visualization and analysis.



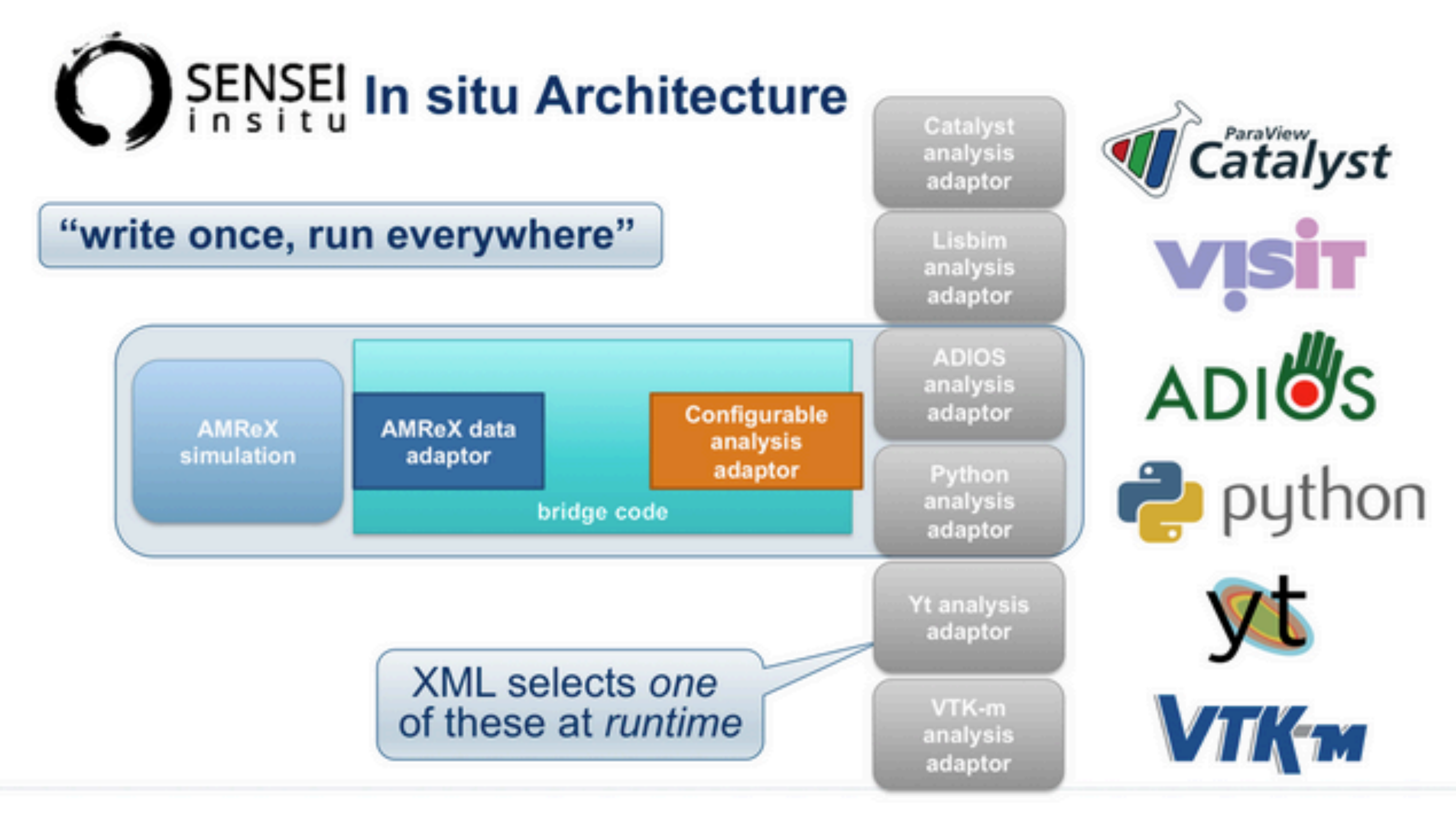
Post-Processing

In-situ processing

AMREX BACKENDS FOR INSITU-VIZ

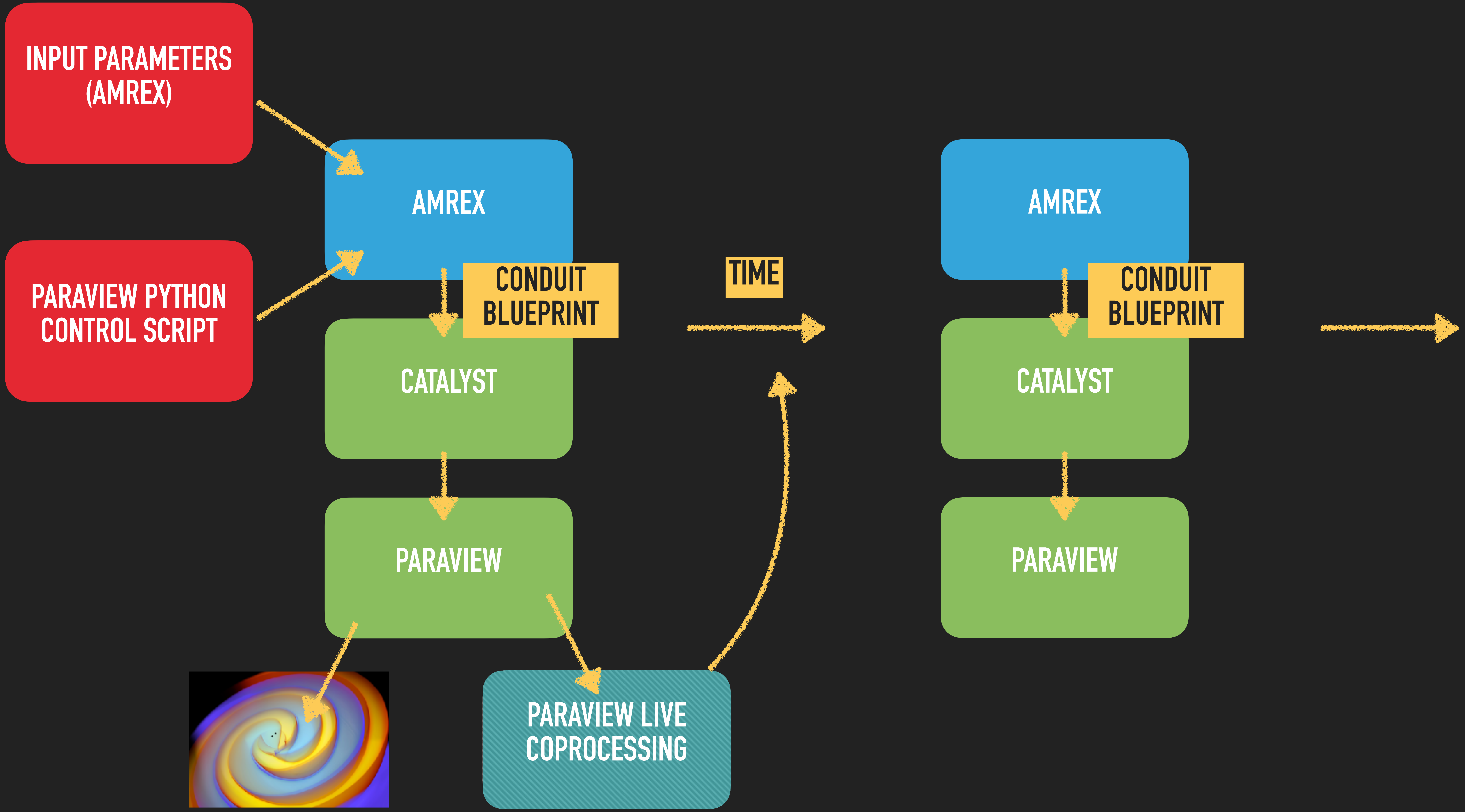
- AMReX provides several capabilities for in-situ viz via:
  - Ascent: https://github.com/alpine-dav/ascent
    - Uses Conduit Blueprint nodes to receive AMReX data
  - SENSEI:
    - Catch all framework that interfaces with ParaView

      Catalyst, yt, ADIOS, VTK-m



- We specifically want to use ParaView with its distributed raytracing capabilities in OSPRay.
- ParaView versions 5.9+ are compatible with ParaView Catalyst (also known as Catalyst 2)
- AMReX data can be directly ported using AMReX's existing Conduit Blueprint backend

OUR IMPLEMENTATION WITHIN AMREX



PARAVIEW CATALYST

- Catalyst API consists of 5 main function calls, of which we will use 3:
    - catalyst_initialize:

Python script that contains instructions for ParaView, e.g. what to plot, camera viewing angle

I pass this in with the AMReX parameter file

Runtime options for ParaView, e.g. - - enable-live

These are optional

The directory to the Catalyst library, also defined in AMReX parameter file

```
void Initialize(std::string filename, std::vector<std::string> catalyst_options,
                std::string paraview_impl_dir)
{
    // Populate the catalyst_initialize argument based on the "initialize"
    // protocol
    // https://docs.paraview.org/en/latest/Catalyst/blueprints.html#protocol-initialize
    conduit_cpp::Node node;

    // This is the Python script that controls ParaView
    node["catalyst/scripts/script/filename"] = filename;


    for (auto const &i : catalyst_options)
    {
        conduit_cpp::Node list_entry =
            node["catalyst/scripts/script/args"].append();
        list_entry.set(i);
    }
    node["catalyst_load/implementation"] = "paraview";

    // This is the directory to the catalyst library
    node["catalyst_load/search_paths/paraview"] = paraview_impl_dir;

    catalyst_status err = catalyst_initialize(conduit_cpp::c_node(&node));
    if (err != catalyst_status_ok)
    {
        amrex::Print() << "Failed to initialize Catalyst: " << err << std::endl;
    }
    else
    {
        amrex::Print() << "Initialized Catalyst: " << err
                       << " with options: " << std::endl;
        for (auto const &i : catalyst_options)
        {
            amrex::Print() << i << std::endl;
        }
    }
}
```

PARAVIEW CATALYST

- Catalyst API consists of 5 main function calls, of which we will use 3:
  - catalyst _execute:

conduit not conduit_cpp

What timestep is it?
(optional)

Name of channel must match input channel in ParaView python script

Only 3 options: “mesh”, “multimesh”, or “amrmesh”, even if you have particle data

```
void Execute(int verbosity, int cycle, double time, int output_levs,
             const amrex::Vector<int> level_steps,
             const amrex::Vector<amrex::Geometry> &geoms,
             const amrex::Vector<amrex::IntVect> &ref_ratios,
             const amrex::Vector<const amrex::MultiFab *> &mfs)
{
    // Populate the catalyst_execute argument based on the "execute" protocol
    // https://docs.paraview.org/en/latest/Catalyst/blueprints.html#protocol-execute

    // Note that this is conduit not conduit_cpp for compatibility with AMReX's
    // Conduit Blueprint backend
    conduit::Node exec_params;

    // State: Information about the current iteration. All parameters are
    // optional for catalyst but downstream filters may need them to execute
    // correctly.

    // add time/cycle information
    auto &state = exec_params["catalyst/state"];
    state["timestep"].set(cycle);
    state["time"].set(time);

    // Channels: Named data-sources that link the data of the simulation to the
    // analysis pipeline in other words we map the simulation datastructures to
    // the ones expected by ParaView.  In this example we use the Mesh Blueprint
    // to describe data see also bellow.

    // The name of the channel must match the name of the input source used in
    // the python script passed to ParaView
    auto &channel = exec_params["catalyst/channels/input"];

    /*   Set the channel's type to "amrmesh" for ParaView 5.12 or
     * "multimesh" for older ParaView versions. */

    channel["type"].set("amrmesh");

    // now create the mesh.
    // when using multimesh, the additional meshes must be named e.g grid/domain
    // for this to be a valid conduit node.

    auto &mesh = channel["data"];
```

PARAVIEW CATALYST

- Catalyst API consists of 5 main function calls, of which we will use 3:

  - catalyst _execute:

This converts AMReX MFs into Conduit Blueprint format

If running in verbose mode, print out a summary of the Catalyst node, including:

- grid dimensions
- parent/child relationships
- list of field names and abbreviated field values

```
    // The name of the channel must match the name of the input source used in
    // the python script passed to ParaView
    auto &channel = exec_params["catalyst/channels/input"];

    /*   Set the channel's type to "amrmesh" for ParaView 5.12 or
     * "multimesh" for older ParaView versions. */

    channel["type"].set("amrmesh");

    // now create the mesh.
    // when using multimesh, the additional meshes must be named e.g grid/domain
    // for this to be a valid conduit node.

    auto &mesh = channel["data"];

    // The variable names
    // Note that MultiLevelToBlueprint makes no distinction about the component
    // number and the ordering of the names passed in
    for (auto const &index : StateVariables::names)
    {
        varnames.push_back(index);
    }

    amrex::MultiLevelToBlueprint(output_levs, mfs, varnames, geoms, time,
                                 level_steps, ref_ratios, mesh);

    if (verbosity > 0)
    {
        exec_params.print();
    }

    catalyst_status err = catalyst_execute(conduit::c_node(&exec_params));
    if (err != catalyst_status_ok)
    {
        amrex::Print() << "Failed to execute Catalyst: " << err << "\n";
    }
    else
    {
        amrex::Print() << "Running Catalyst at timestep: " << cycle << "\n";
    }
}
```

PARAVIEW CATALYST

- Catalyst API consists of 5 main function calls, of which we will use 3:
    - catalyst _finalize:

```
// Although no arguments are passed for catalyst_finalize  it is required in
// order to release any resources the ParaViewCatalyst implementation has
// allocated.
void Finalize() {
  conduit_cpp::Node node;
  catalyst_status err = catalyst_finalize(conduit_cpp::c_node(&node));
  if (err != catalyst_status_ok) {
    amrex::Print() << "Failed to finalize Catalyst: " << err << "\n";
  } else {
    amrex::Print() << "Finalized Catalyst: " << err << "\n";
  }
}
} // namespace CatalystAdaptor
```

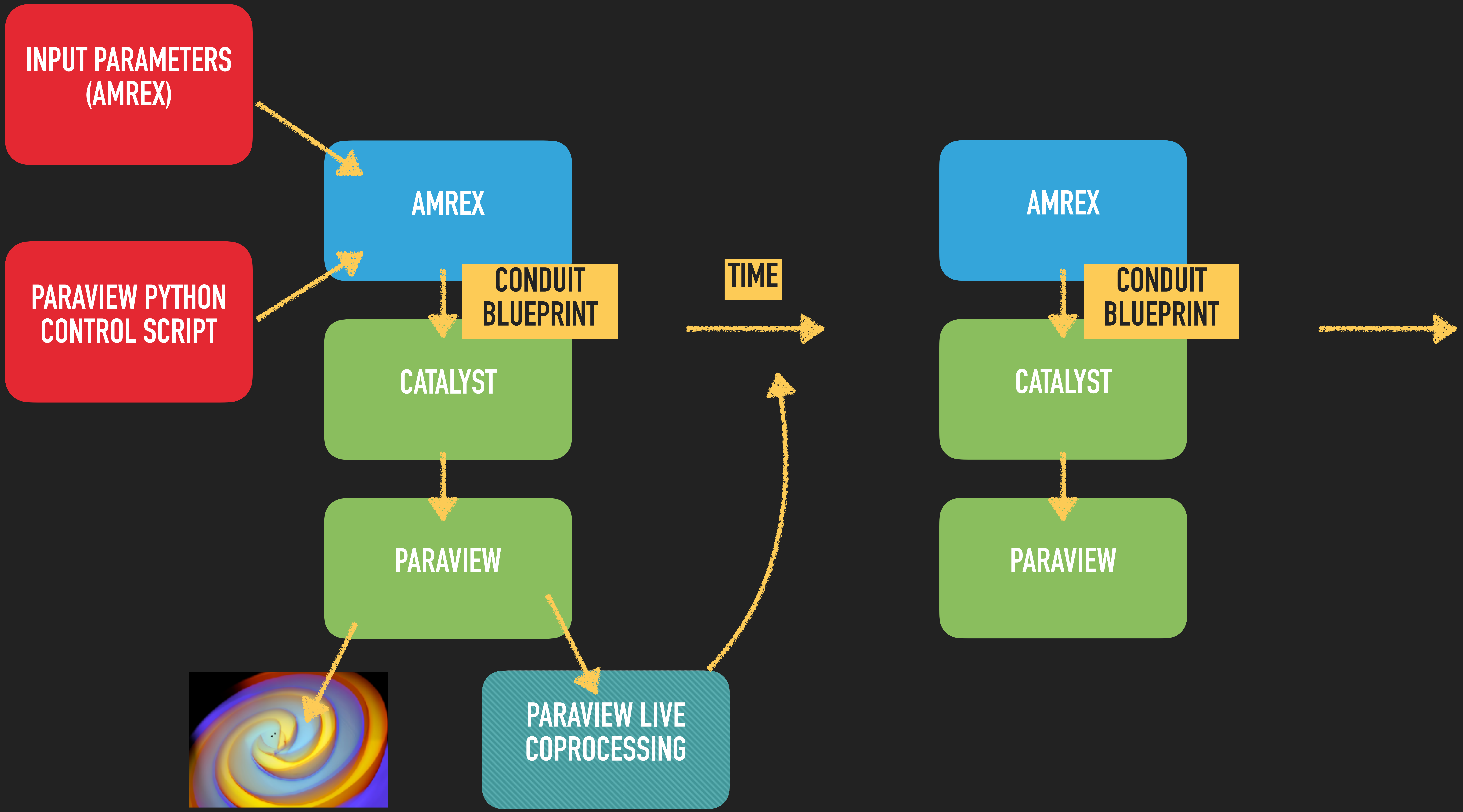
BUILDING WITH AMREX/GRTECLYN

- MultiFabToConduitBlueprint is defined in <AMReX_Conduit_Blueprint.h>
  - Must set USE_CONDUIT = TRUE to build these
- Initialize (Finalize) are called before (after) any time stepping is done
- ParaView Catalyst's execute is called during specificPostTimeStep()
  - Pointers to MultiFabs at each level are collected and passed through to Catalyst
- Also set USE_CATALYST=TRUE as a makefile option to use ParaView Catalyst
  - This requires CONDUIT_DIR and PARAVIEW_DIR environment variables to be set also
- ParaView parameters are set in the input file, e.g. Catalyst implementation path, extra flags to ParaView

MULTIFAB TO CONDUIT BLUEPRINT

- Written by Cyrus Harrison (LLNL) and Matt Larsen (LLNL)
- Must provide: a list of MultiFabs, geometries, variable names
- Conduit needs to know:
  - min/max and spacing of grid in both physical and grid coordinates and topology
  - ordering of AMR levels
  - where the data is stored
  - unique numbering of grids
- Some improvements could be made though:
  - Must cycle through each MF and look for parent/child relationships by refining the coarse grid
  - Variable names are assumed to be in order

OUR IMPLEMENTATION WITHIN AMREX



INTERFACING WITH PARAVIEW: A SIMPLE EXAMPLE

- A Python script controlling ParaView must be passed into AMReX as well as the usual AMReX parameter file.
- A minimal example (from the ParaView catalyst examples directory):

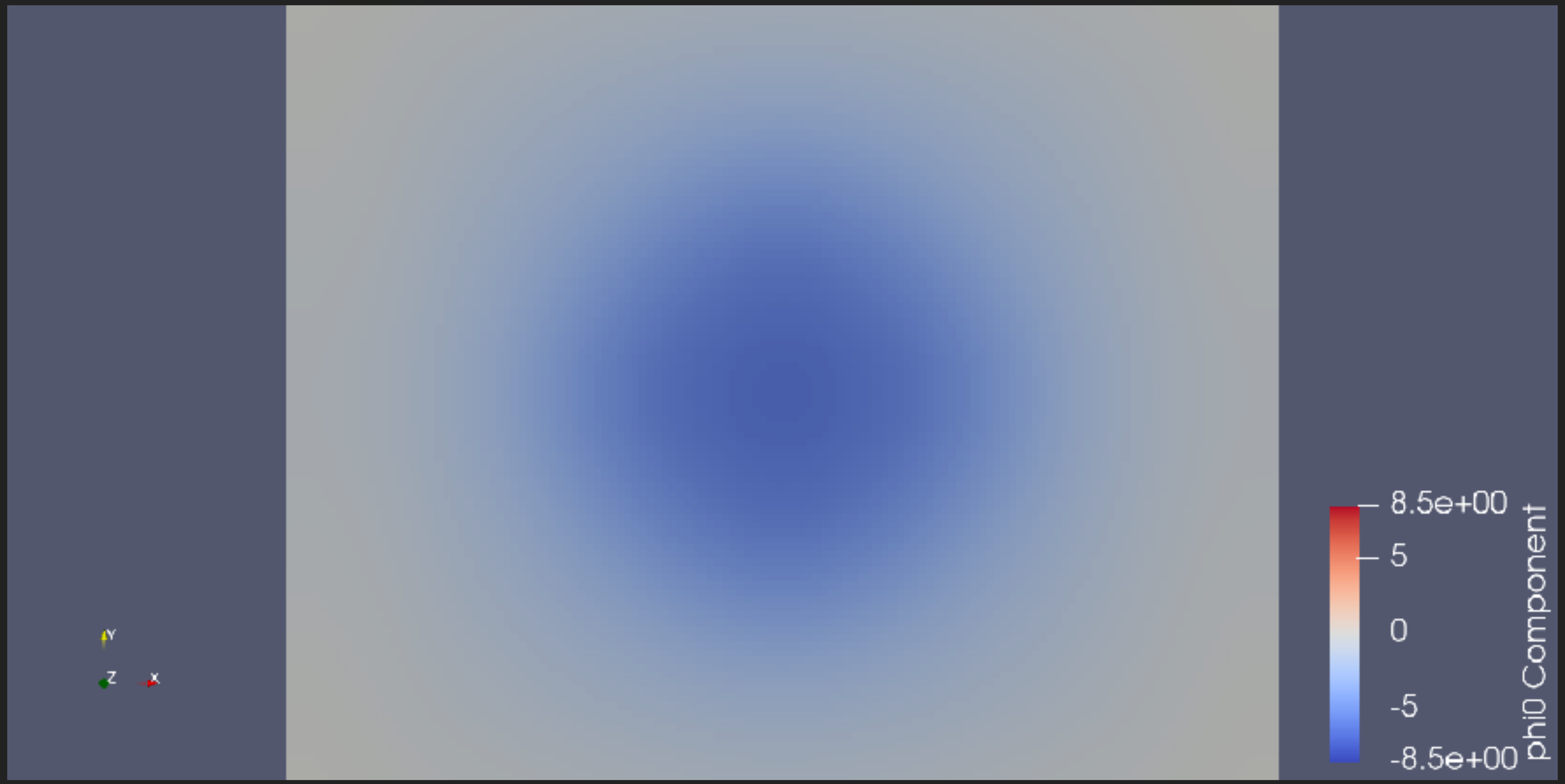
```python
from paraview.simple import *

# Greeting to ensure that ctest knows this script is being imported
print("executing catalyst_pipeline")

# registrationName must match the channel name used in the
# 'CatalystAdaptor'.
producer = TrivialProducer(registrationName="grid")

def catalyst_execute(info):
    global producer
    producer.UpdatePipeline()
    print("-----------------------------------")
    print("executing (cycle={}, time={})".format(info.cycle, info.time))
    print("bounds:", producer.GetDataInformation().GetBounds())
    print("velocity-magnitude-range:", producer.PointData["velocity"].GetRange(-1))
    print("pressure-range:", producer.CellData["pressure"].GetRange(0))

    # access the node pass through catalyst_execute from the simulation
    # make sure that CATALYST_PYTHONPATH is in your PYTHONPATH
    node = info.catalyst_params
    print(f"{node=}")
```

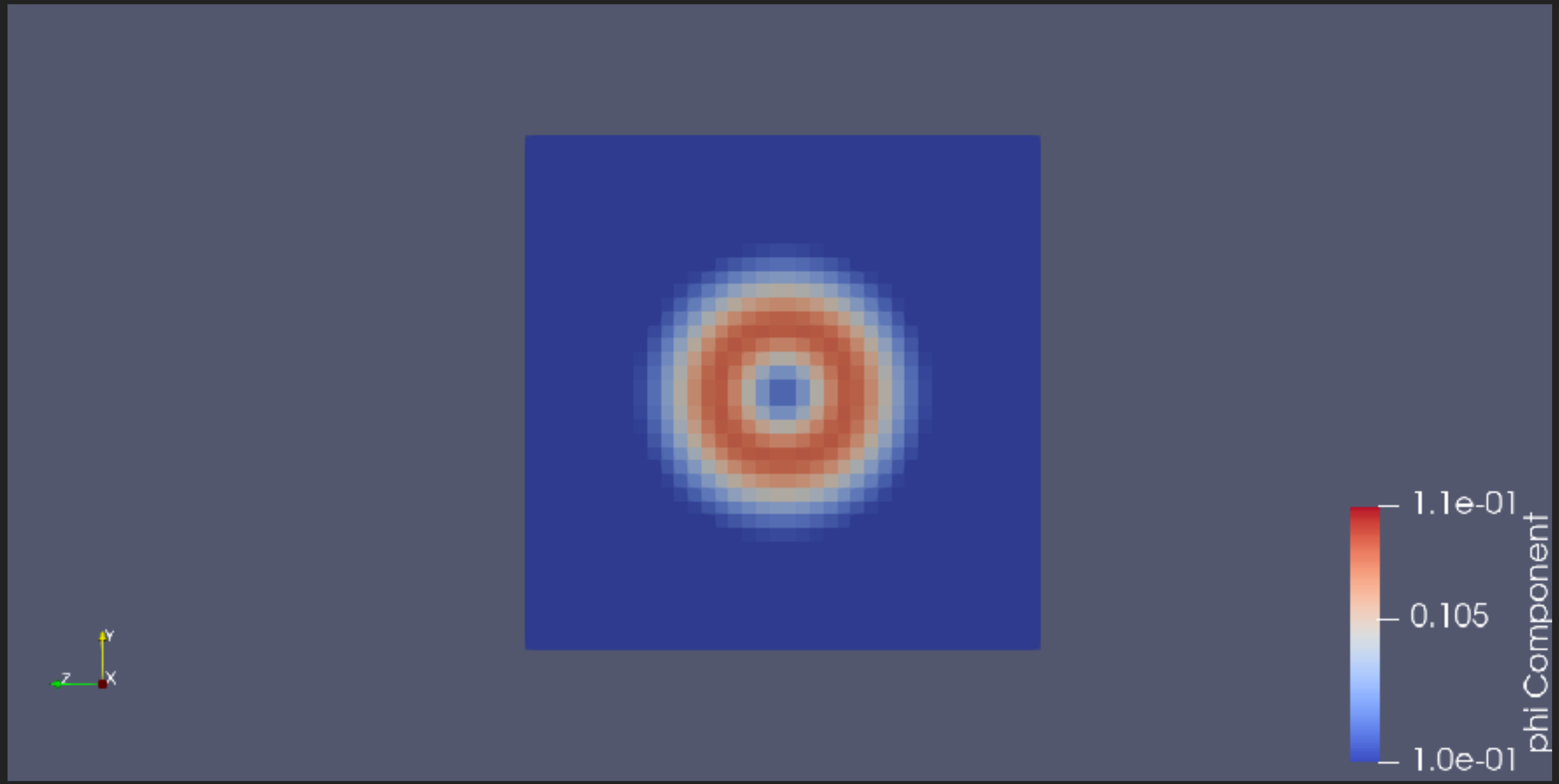
INTERFACING WITH PARAVIEW: A MORE USEFUL EXAMPLE

```
pNG1 = CreateExtractor('PNG', renderView1, registrationName='PNG1')
# trace defaults for the extractor.
pNG1.Trigger = 'TimeStep'

# init the 'PNG' selected for 'Writer'
pNG1.Writer.FileName = 'main_test_{timestep:06d}.png'
pNG1.Writer.ImageResolution = [1600,800]
pNG1.Writer.Format = 'PNG'

SetActiveSource(pNG1)

# ------------------------------------------------------------------------------
# Catalyst options
options = catalyst.Options()
options.GlobalTrigger = 'TimeStep'
options.ExtractsOutputDirectory = "/home/dc-kwan1/rds/rds-dirac-dp002/dc-kwan1/AMReX/wave/insitu-viz/paraview_5p12"

if "--enable-live" in catalyst.get_args():
  options.EnableCatalystLive = 1
  options.CatalystLiveTrigger = 'TimeStep'
  options.CatalystLiveURL = "localhost:22222"


# Greeting to ensure that ctest knows this script is being imported
print("executing catalyst_pipeline")

def catalyst_execute(info):
    global producer, grid
    producer.UpdatePipeline()
    print(producer.GetDataInformation().GetDataAssembly())


    print("-----------------------------------")
    print("executing (cycle={}, time={})".format(info.cycle, info.time))

    SaveExtractsUsingCatalystOptions(options)
```

AND AFTERWARDS…?

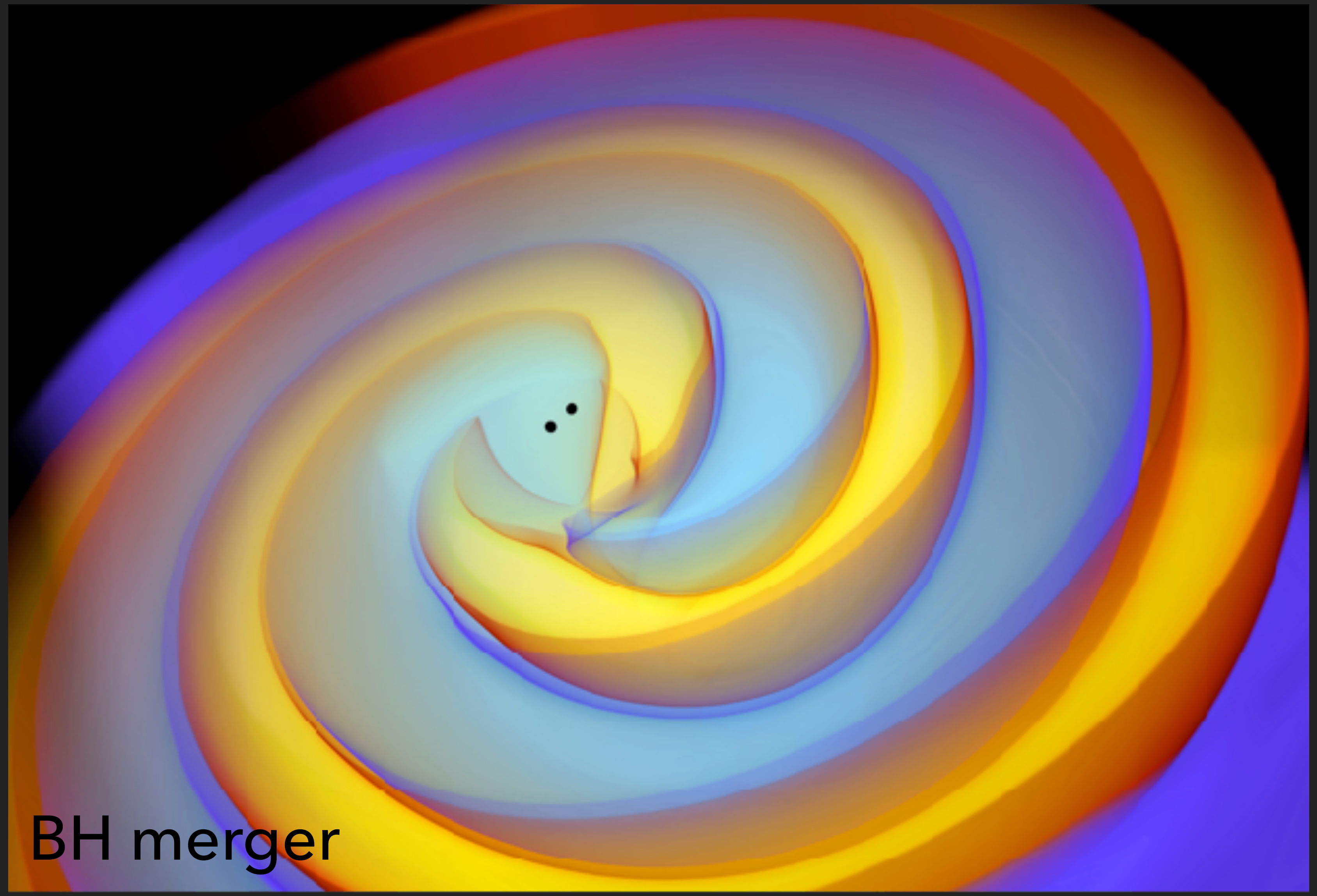
- Use ffmpeg to convert snapshots (*.png files) to movies
  - ffmpeg -r <frame rate> -i <filenames> movie.mp4



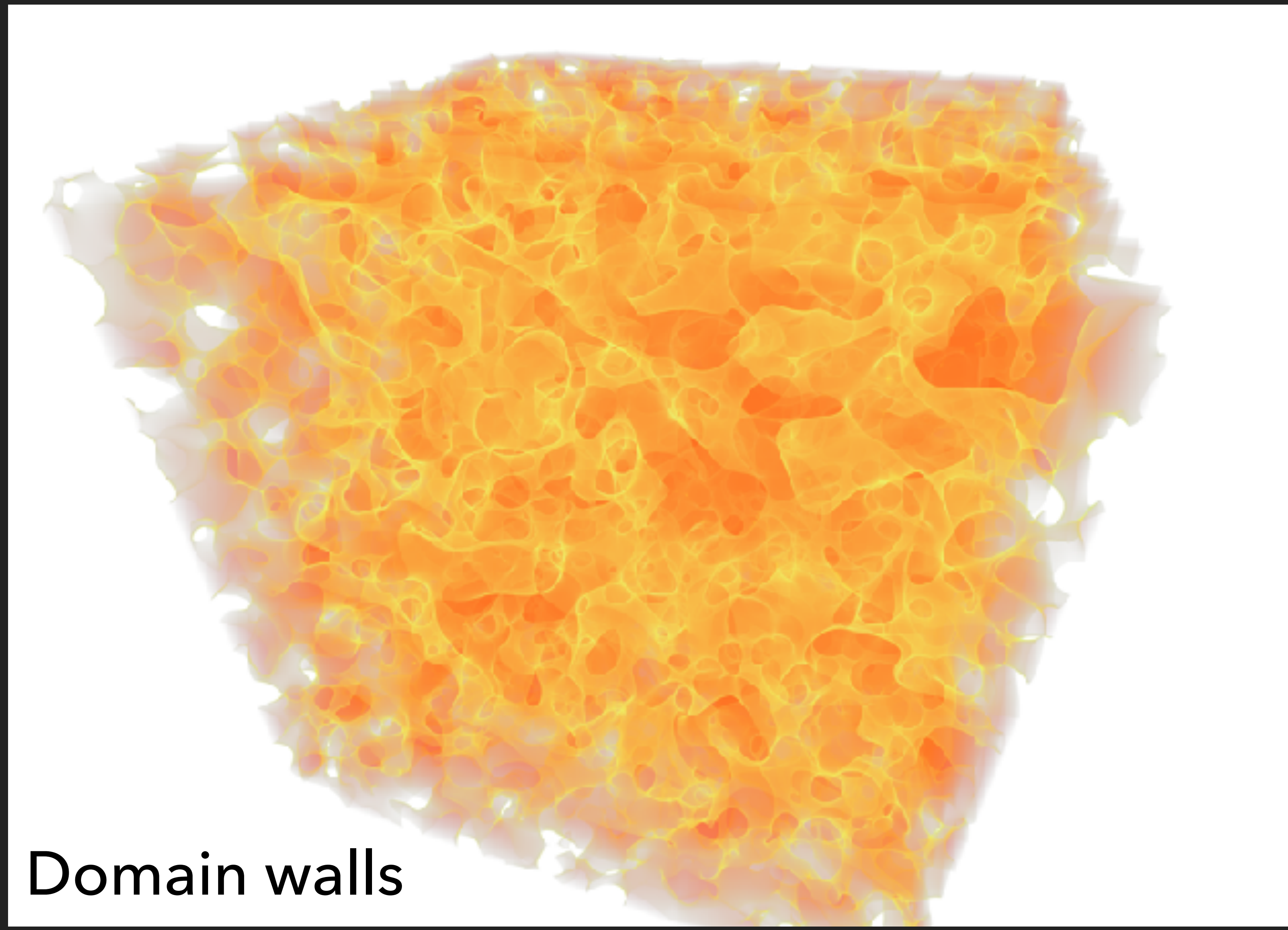


FUTURE DEVELOPMENTS: OSPRAY/WOMBAT

- MPI distributed volume rendering within ParaView is done by Intel OSPRay.
- OSPRay can raytrace through the AMR data structure by using Wombat to break up the grids into convex patches.
- Wombat is needed to correctly order the grid data along the line of sight for distributed ray tracing over several processors.
- Wombat is still a work in progress.



BH merger



Domain walls