



In Situ Visualization and Analysis with Ascent

Hank Childs, University of Oregon
Matthew Larsen, Lawrence Livermore National Laboratory
Cyrus Harrison, Lawrence Livermore National Laboratory
Kenneth Moreland, Sandia National Laboratories
David Rogers, Los Alamos National Laboratory

2019 Exascale Computing Project Annual Meeting


HELD AT

**Royal Sonesta Houston Galleria
Houston, Texas**

January 14–17, 2019 – ECP Annual Meeting

JANUARY 14-18, 2019 - INDIVIDUAL OR GROUP MEETINGS

Big Picture

-  **Ascent** is a library for in situ visualization and analysis
 - Made by the developers of ParaView and visit
 - Will include pathways to both of those tools in 2019
- Emphasis on:
 - “Flyweight” approach (both API and runtime)
 - Exascale architectures (both many-core architectures and distributed memory parallel)

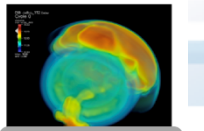
Visualization and Analysis for HPC: Current Status

- Developed popular, open source tools (ParaView, VisIt) based on the Visualization ToolKit library (VTK)
 - Widespread usage in DOE and >1 million downloads worldwide
 - Hundreds of person years of effort
- Three major problems for exascale:
 - 1) Many-core architectures (as current VTK-based investments are primarily only MPI parallelism)
 - 2) I/O limitations will require in situ processing
 - 3) Artifacts created will require new methods of analysis and vis

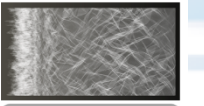
Scientific Visualization and Analysis Ecosystem for Large Data (VTK, ParaView*, VisIt*, Cinema, SENSEI, and VTK-m*)

- Problem
 - DOE simulation scientists generate petabytes of data that they need to understand
- Solution
 - Developed general-purpose, scientific visualization and analysis libraries and tools
 - designed with parallelism in mind to operate on world's largest data sets
 - collaborative open-source development model engaging multiple National Laboratories, universities, and industry
- Impact
 - Leading visualization and analysis solutions for Department of Energy scientists
 - Used on all ASCR supercomputing facilities, and worldwide in most HPC facilities
 - Over 1 million downloads of software worldwide
 - Creates capability for DOE to look at data from the world's largest simulations
 - Steve Langer (LLNL) – "We rely on this software for gaining an understanding of complex and spatial variations in backscattered light simulations using pF3D, where data may consist of over 400 billion zones per time step."

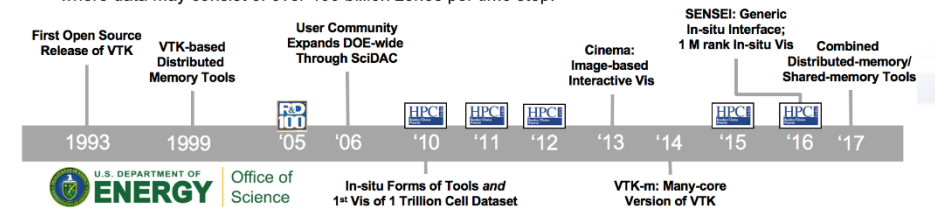
* ECP funded



Core collapse supernova from GeneASIS.



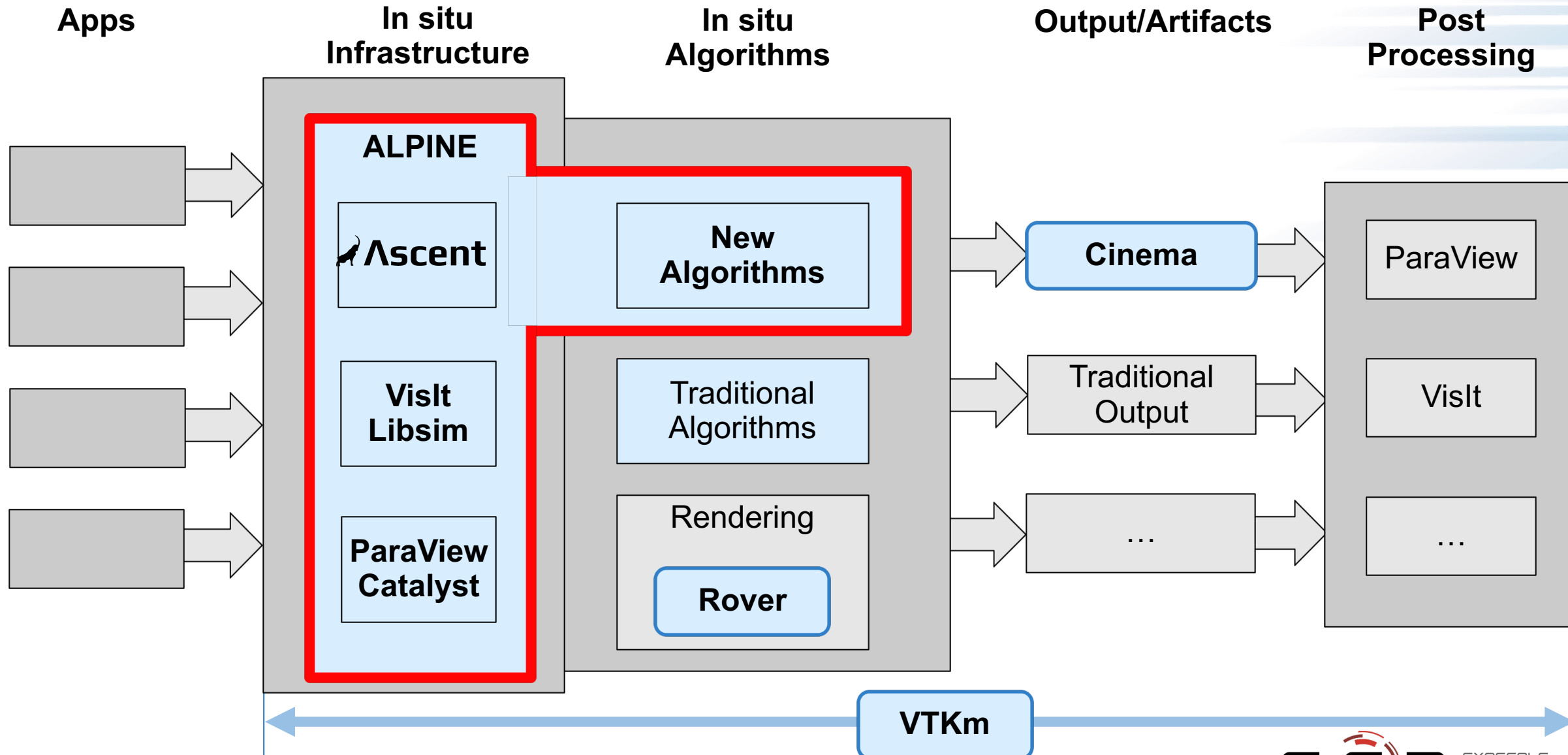
Novel visualization of ocean flow data, designed to show 'mixing barrier' (left).




ASCR highlight slide for VTK-based tools.

ECP VTK-m project is focused on problem #1.
ECP ALPINE is focused on problem #2.
ECP ALPINE and ATDM Cinema impact #3.
Our approaches are complementary and coordinated.

ECP DAV is an integrated workflow



This tutorial

- 2 hours:  **Ascent**
 - Overview
 - How to use? (get hands dirty / walk out with understanding on how to integrate)
 - Examples of advanced usage (what it can do)
- 1 hour: other ECP vis technologies (can interact with Ascent or work standalone)
 - Cinema
 - VTK-m
 - In situ algorithms

Tutorial Team

Contributor	Hank Childs 	Matthew Larsen 	Cyrus Harrison 	Kenneth Moreland 	David Rogers 
Affiliation				 Sandia National Laboratories	
ECP Projects	ALPINE Deputy PI, VTK-m	ALPINE, Rover	ALPINE	VTK-m PI	Cinema Co-PI
Software Projects	VisIt, VTK-m, Ascent	Ascent, VTK-m, Conduit, VisIt	VisIt, Ascent, Conduit	VTK-m, ParaView, Ice-T	Cinema, ParaView

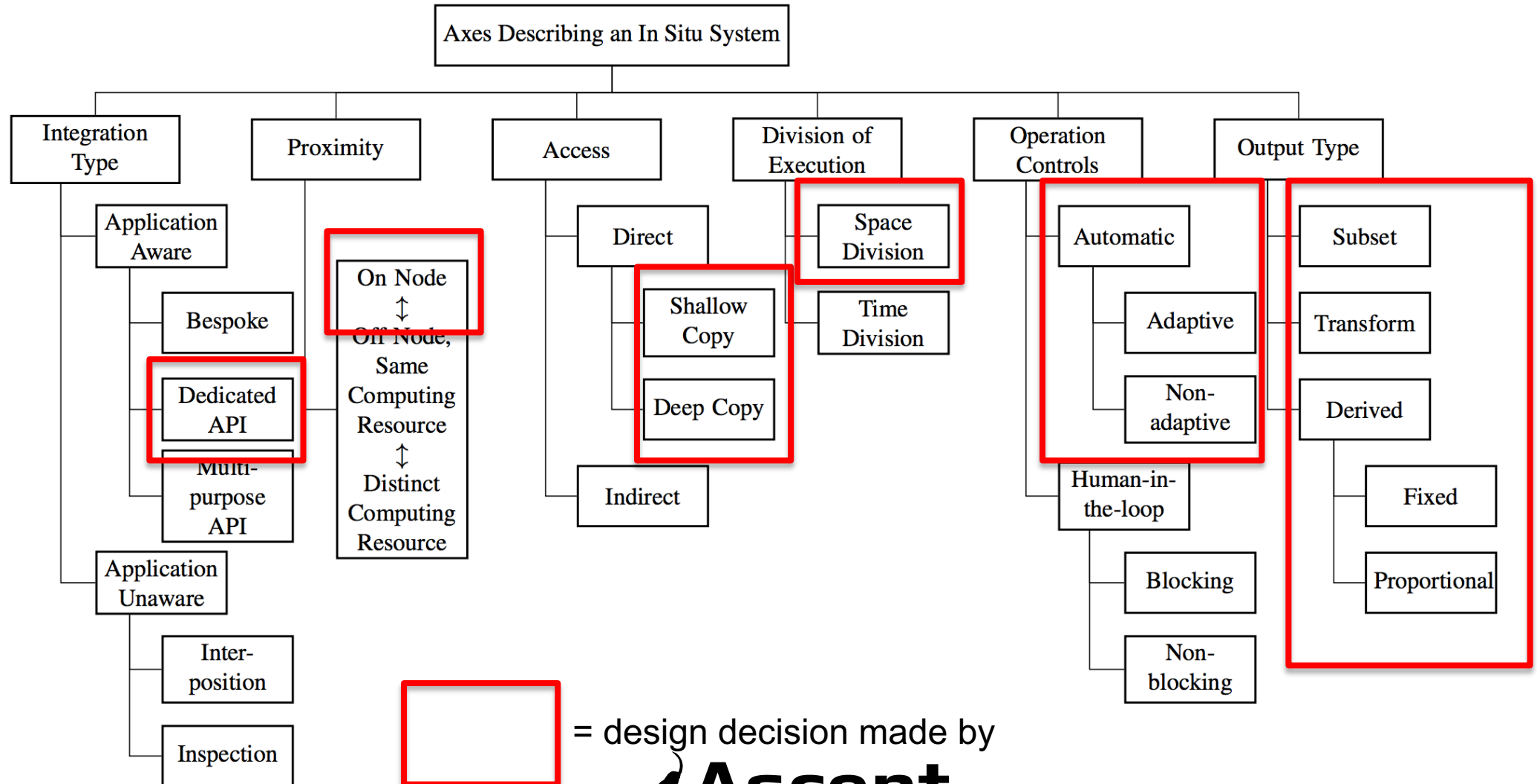
This tutorial

- 2 hours: Ascent
 - Overview
 - How to use? (get hands dirty / walk out with understanding on how to integrate)
 - Examples of advanced usage (what it can do)
- 1 hour: other ECP vis technologies
 - Cinema
 - VTK-m
 - In situ algorithms

There are multiple flavors of in situ processing (1/2)

- Ascent follows a traditional “tightly coupled” / “in line” in situ approach:
 - It is a library that you link into your simulation code
 - It uses the same resources that your simulation uses
 - It is designed for execution to alternate between your simulation and visualization/analysis
 - This approach can be used to simplify data ownership issues
 - But Ascent can also make a copy of your data
- Ascent focuses on flyweight processing:
 - It does not need to make a copy of your data for its own purposes
 - It has minimal dependencies on other software (i.e., small binary size)

There are multiple flavors of in situ processing (2/2)



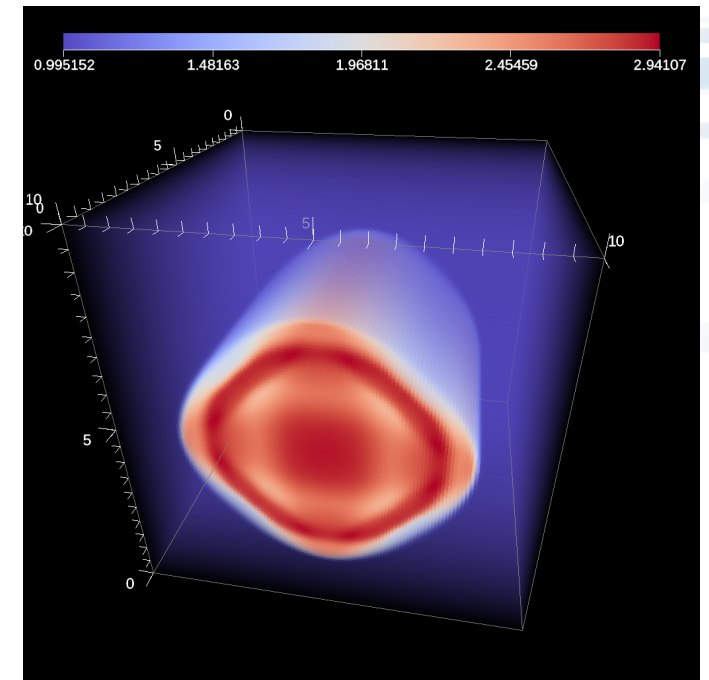
Ascent is designed for the exascale.

- In situ processing:
 - Will be important part of exascale
 - Need both traditional algorithms and new algorithms
 - New algorithms:
 - No human in the loop
 - Data reduction
- Architectures:
 - Utilizing VTK-m for shared memory parallelism
 - Handles distributed memory parallelism via MPI

Ascent is an easy to use flyweight in-situ visualization and analysis library for HPC simulations

Project Info:

- Website + Docs: <http://ascent-dav.org>
- GitHub Repo: <https://github.com/Alpine-DAV/ascent>
- Email Help: help@ascent-dav.org
- Supported Languages: C++, Python, C, Fortran
- License: BSD Style
- Builds with Spack <https://spack.io/>



Example in-situ rendering
created using Ascent

Ascent focuses on ease of use and efficient in-situ execution

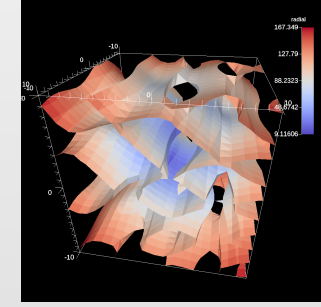
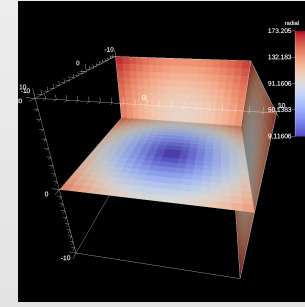
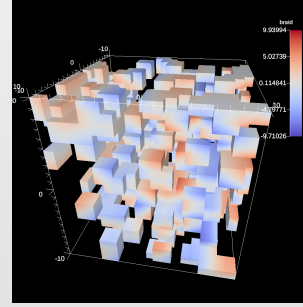
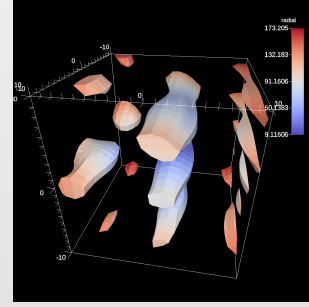
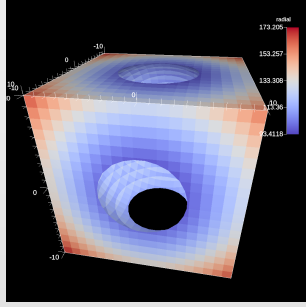
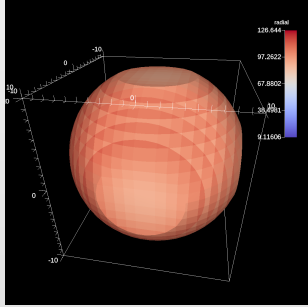
Ascent Delivers

- An easy to use API
 - Designed to enable three use cases
 - Making Pictures
 - Transforming Data
 - Capturing Data
 - Leverages Conduit (<http://software.llnl.gov/conduit>)
 - Underpins support for C, C++, Fortran, and Python
 - Simplifies handoff of mesh-based simulation data
 - Convention for specifying data called “Blueprint”
- A flyweight design
 - Efficient distributed-memory + many-core execution
 - Leverages MPI, VTK-m (<http://m.vtk.org/>)
 - Lower memory requirements than current tools
 - Less dependencies than current tools (ex: no OpenGL)

```
//  
// Run Ascent  
//  
Ascent ascent;  
ascent.open();  
ascent.publish(data);  
ascent.execute(actions);  
ascent.close();
```



Ascent is ready for common visualization use cases

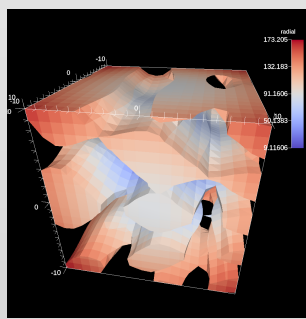
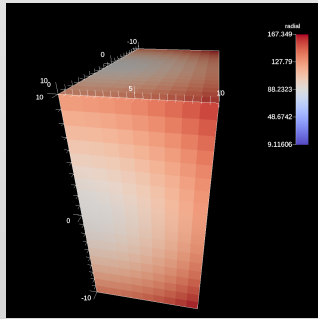


Iso-Volume

Threshold

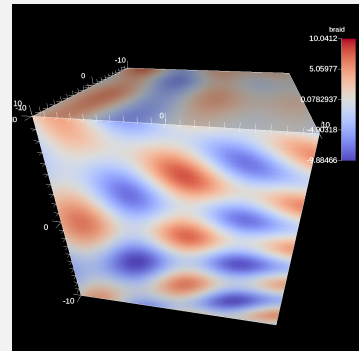
Slice

Contour

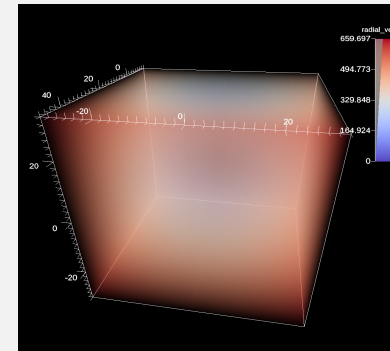


Clips

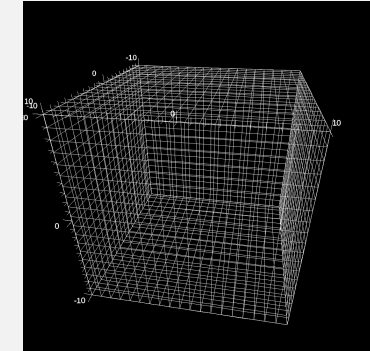
Rendering



Pseudocolor



Volume



Mesh

Ascent supports multiple languages and output types

- Language Bindings

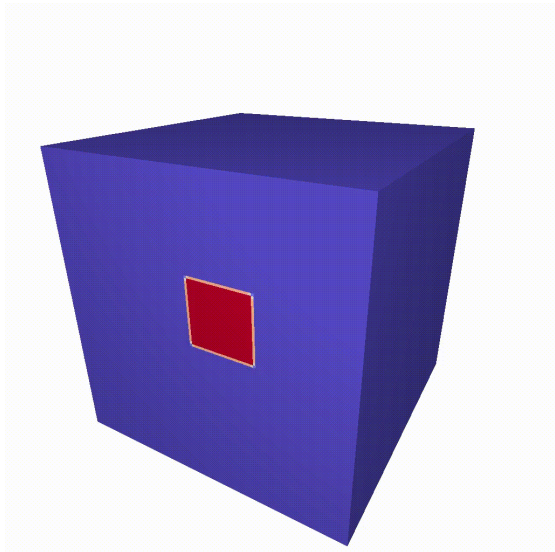


Fortran

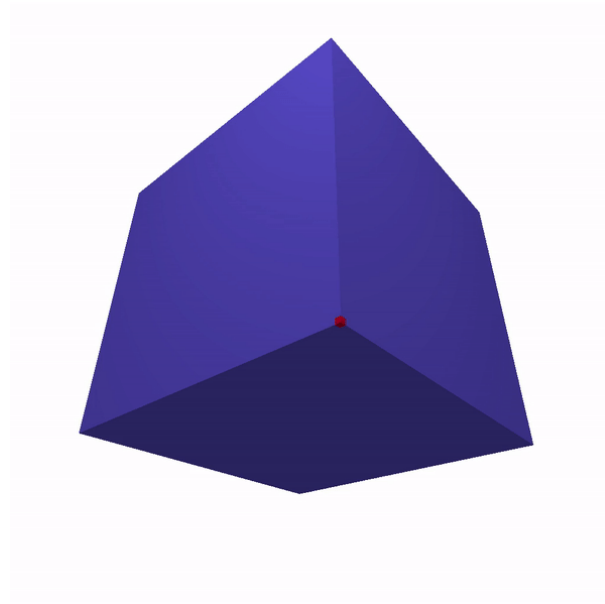
- Output Types



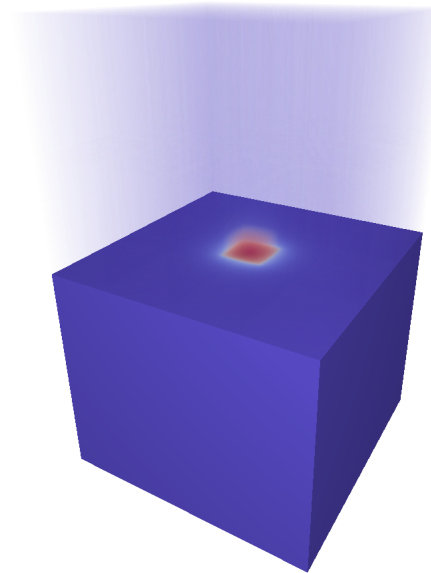
Ascent provides example integrations that also serve as built-in data sources



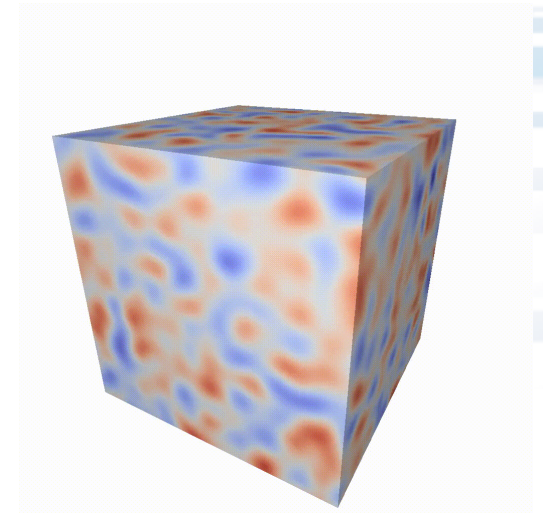
Cloverleaf3D



Lulesh



Kripke

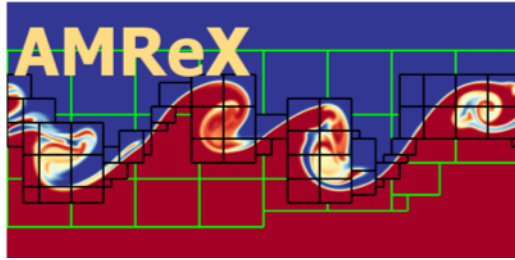


Smooth Noise

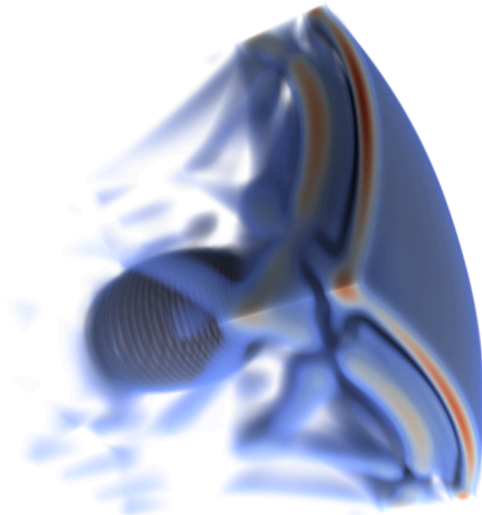
Ascent is being developed by ECP ALPINE (2.3.4.12)

Scope & Intent	R&D Themes	Delivery Process	Target ECP Users	Support Model
Deliver in situ visualization and analysis algorithms and infrastructure.	1) Automated in situ massive data reduction algorithms	Regular releases of software and documentation, open access to production software from GitHub	All ECP applications. Focused delivery for co-design centers applications.	Ongoing developer support. Dedicated email, issue tracking portals, comprehensive web-based documentation, regular tutorials.
	2) Portable, scalable, performant infrastructure			

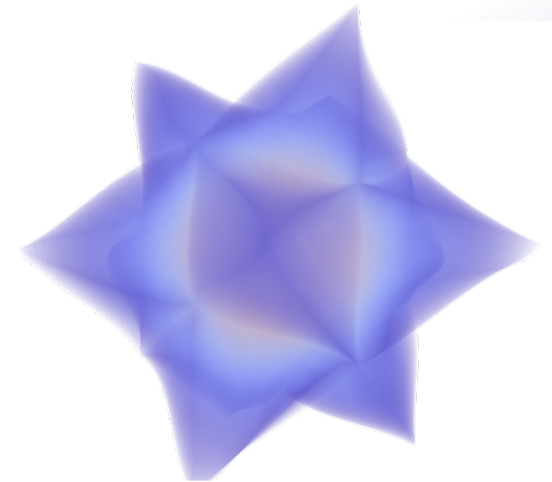
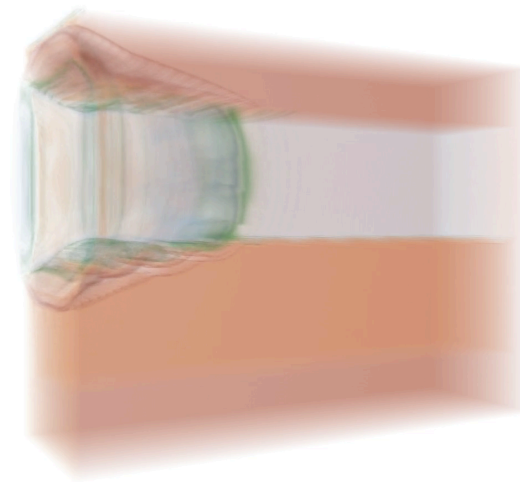
We are working to provide ECP Co-Design Centers easy paths to publish simulation mesh data to Ascent



We are developing AMReX functions to wrap AMR Grids and Particle Containers for use in Ascent



MFEM includes Conduit support which wraps MFEM High-order meshes for use in Ascent

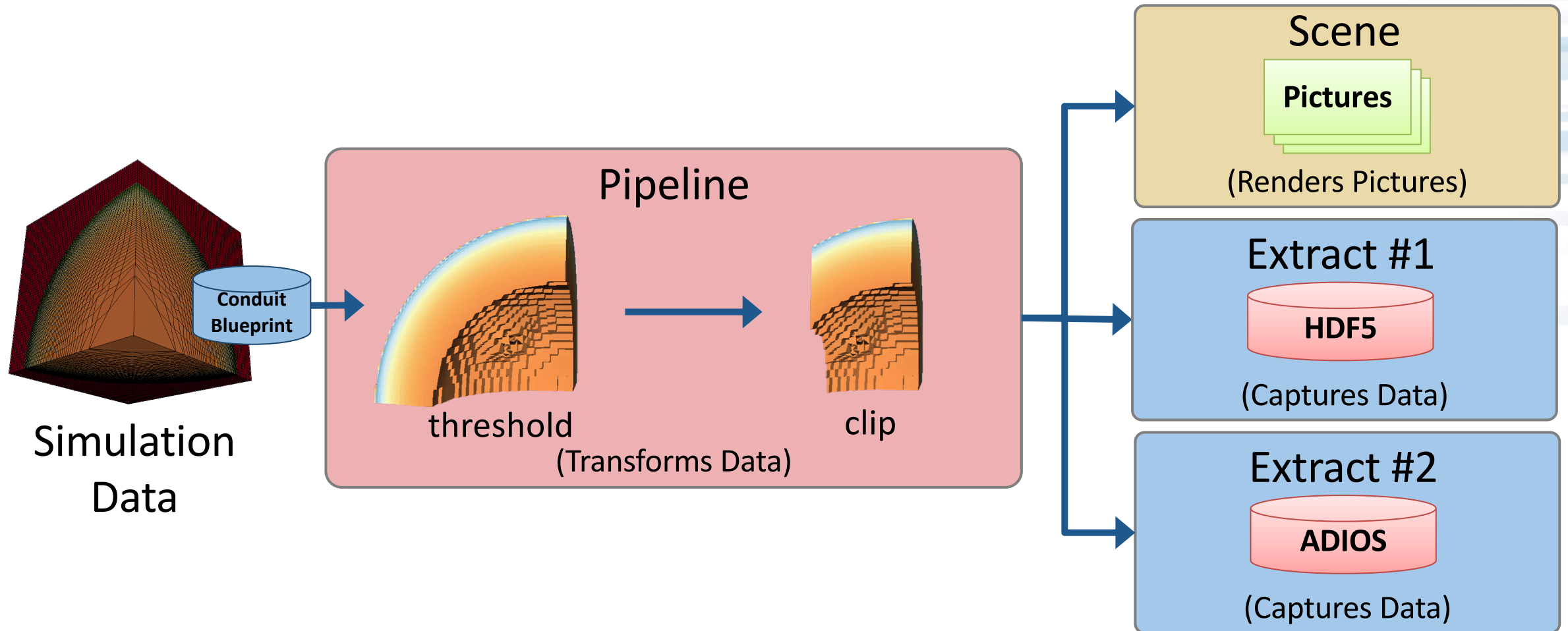


Ascent Concepts

Ascent's API is composed of three key concepts

- Pipelines (transform data):
 - Allows users to describe how they want to transform their data
- Scenes (make pictures):
 - Allows users to describe the pictures they want to create
- Extracts (capture data):
 - Allows users to describe how they want capture data

Ascent end-to-end conceptual example

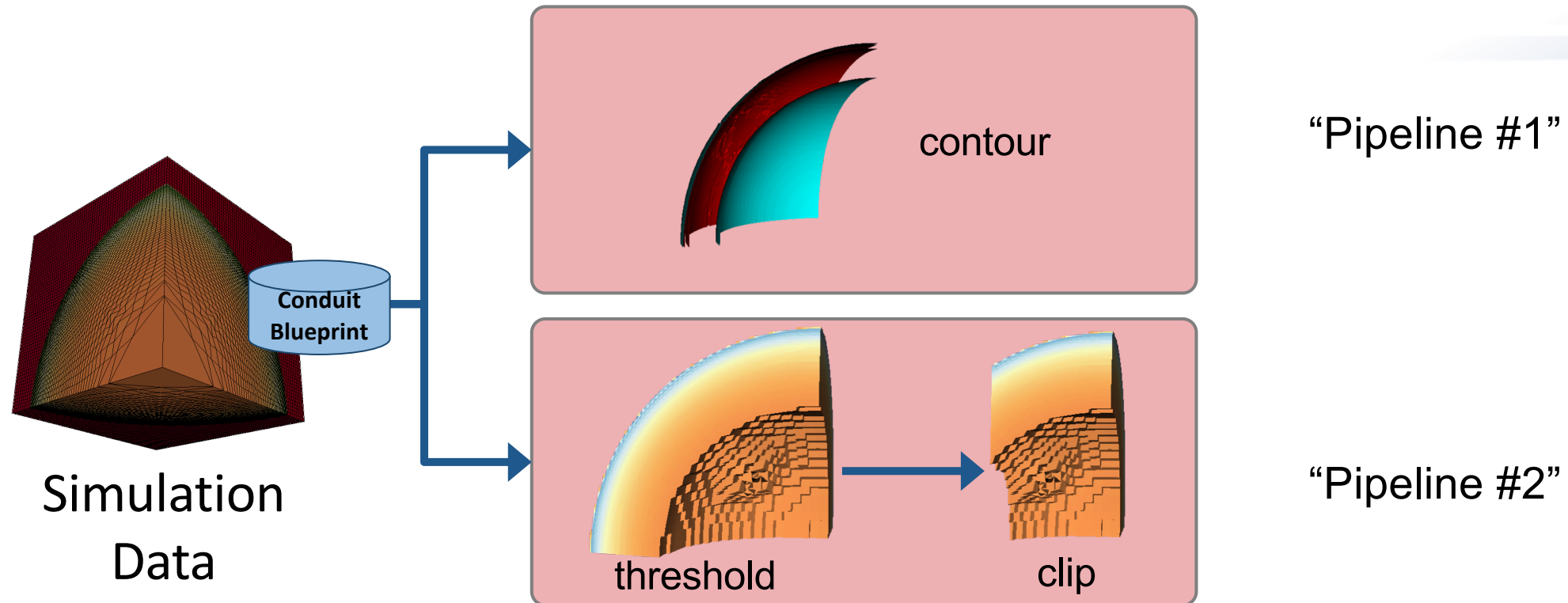


Ascent's API is composed of three key concepts

- **Pipelines (transform data):**
 - Allows users to describe how they want to transform their data
- **Scenes (make pictures):**
 - Allows users to describe the pictures they want to create
- **Extracts (capture data):**
 - Allows users to describe how they want capture data

A pipeline is a series data transformations (i.e., filters)

- Ascent allows an arbitrary number of pipelines to be described

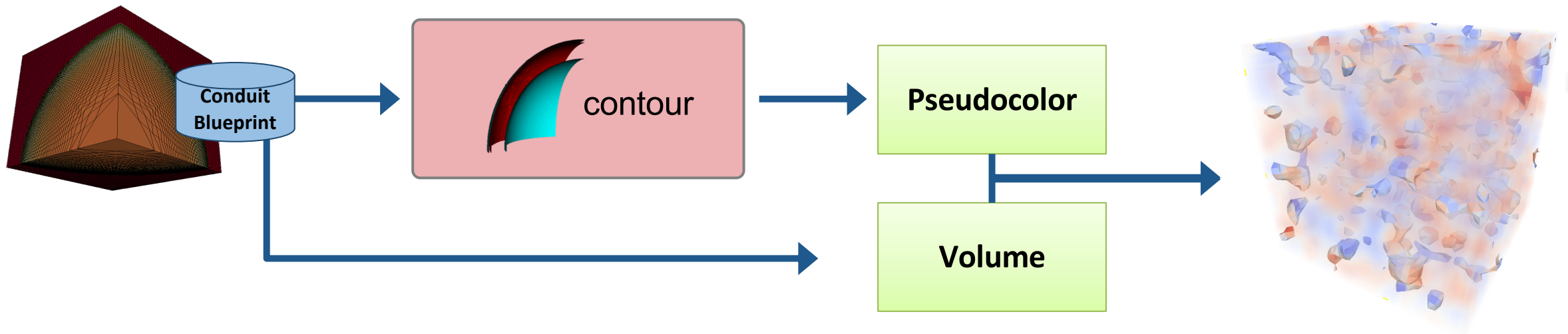


Ascent's API is composed of three key concepts

- Pipelines (transform data):
 - Allows users to describe how they want to transform their data
- Scenes (make pictures):
 - Allows users to describe the pictures they want to create
- Extracts (capture data):
 - Allows users to describe how they want capture data

A scene is a way to render pictures

- Contains a list of plots
 - E.g., volume, pseudocolor, and mesh
- Contains a list of camera parameters



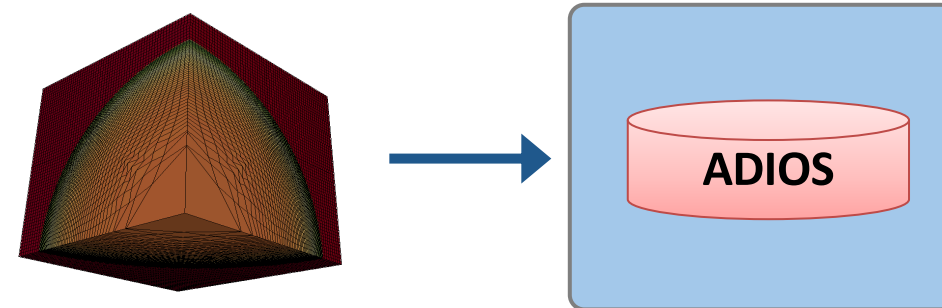
Ascent's API is composed of three key concepts

- Pipelines (transform data):
 - Allows users to describe how they want to transform their data
- Scenes (make pictures):
 - Allows users to describe the pictures they want to create
- Extracts (capture data):
 - Allows users to describe how they want capture data

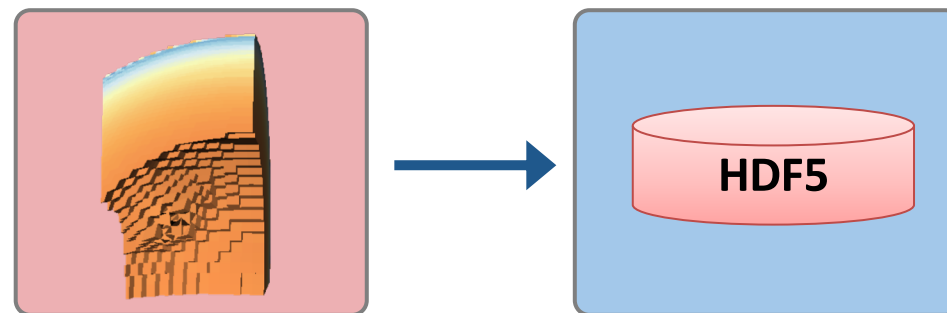
Extracts capture data for use outside of Ascent

- Examples:

- Export published simulation data to HDF5, ADIOS, etc



- Export pipeline results to HDF5, ADIOS, etc.



Currently supported extracts:

- Create Cinema databases
- Export to HDF5 files
- Publish to an embedded Python interpreter
- Publish to ADIOS (proof-of-concept)



This tutorial

- 2 hours: Ascent
 - Overview
 - How to use? (get hands dirty / walk out with understanding on how to integrate)
 - Tutorial Setup
 - Compiling Ascent into a simulation code
 - Hello World example
 - Conduit Blueprint: data model API
 - Specifying actions
 - Examples of advanced usage (what it can do)
- 1 hour: other ECP vis technologies

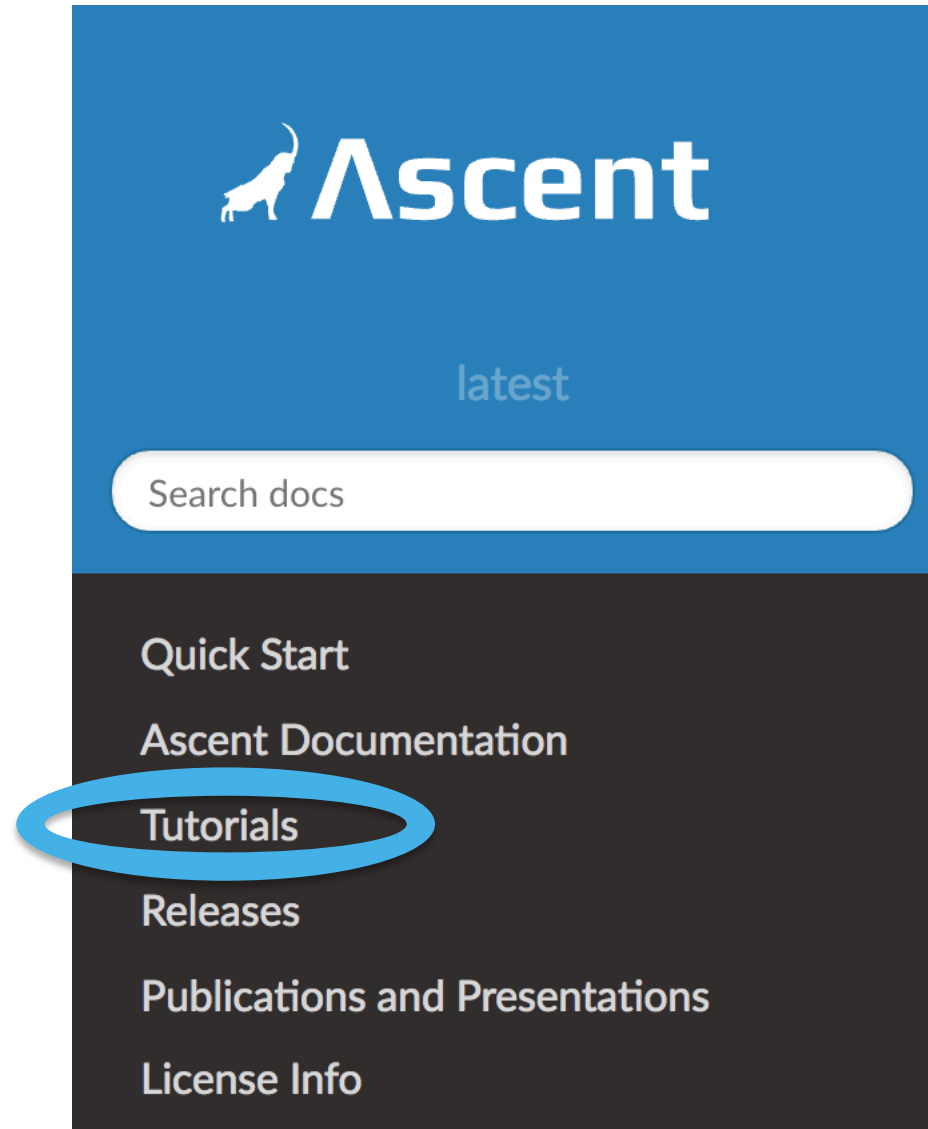
Tutorial Setup

Tutorial Setup – Building or Obtaining Ascent

- Game plan:
 1. Get your hands on an Ascent install:
 - Download Ascent source code and build using a script (uberenv + spack)
OR
 - Use an existing install on NERSC's Cori System
OR
 - Use our Docker image
 2. Test your ascent install against included example

Tutorial Setup – Tutorial Docs

- <http://ascent-dav.org>
- Click on “Tutorials”



Tutorial Setup – Quick Start (Build via script)

- ECP Tutorial Docs: <https://ascent.readthedocs.io/en/latest/Tutorials.html>

Tutorial Setup

Build and Install

To build and install Ascent yourself see [Quick Start](#).

Tutorial Setup – Quick Start (Build via script)

- Quick Start: <https://ascent.readthedocs.io/en/latest/QuickStart.html>

Quick Start

Installing Ascent and Third Party Dependencies

The quickest path to install Ascent and its dependencies is via [uberenv](#):

```
git clone --recursive https://github.com/alpine-dav/ascent.git
cd ascent
python scripts/uberenv/uberenv.py --install --prefix="build"
```

After this completes, `build/ascent-install` will contain an Ascent install.

For more details about building and installing Ascent see [Building Ascent](#). This page provides detailed info about Ascent's CMake options, [uberenv](#) and [Spack](#) support. We also provide info about [building for known HPC clusters using uberenv](#) and a [Docker example](#) that leverages Spack.

Tutorial Setup – Running on NERSC’s Cori

- ECP Tutorial Docs: <https://ascent.readthedocs.io/en/latest/Tutorials.html>

NERSC Cori Install

We have a public ascent install for use on NERSC’s Cori System. This install was built with the default intel compiler (18.0.1.163).

The install is located at `/project/projectdirs/alpine/software/ascent/ecp_2019/ascent-install`. You can copy the tutorial examples from this install and build them as follows:

```
cp -r /project/projectdirs/alpine/software/ascent/ecp_2019/ascent-install/examples/ascent/tutorial
cd ecp_2019
make ASCENT_DIR=/project/projectdirs/alpine/software/ascent/ecp_2019/ascent-install/
```

Tutorial Setup – Using Docker

- ECP Tutorial Docs: <https://ascent.readthedocs.io/en/latest/Tutorials.html>

Using Docker

If you have Docker installed you can obtain a Docker image with a ready-to-use ascent install from [Docker Hub](#).

Fetch the latest Ascent image:

```
docker pull alpinedav/ascent
```

After the download completes, create and run a container using this image:

```
docker run -p 8000:8000 -p 10000:10000 -t -i alpinedav/ascent
```

(The `-p` is used to forward ports between the container and your host machine, we use these ports to allow web servers on the container to serve data to the host.)

You will now be at a bash prompt in you container.

To add the proper paths to Python and MPI to your environment run:

```
source ascent_docker_setup.sh
```

The ascent source code is at `/ascent/src/`, and the install is at `/ascent/install-debug`.

Tutorial Setup – Example Program Source Code

- ECP Tutorial Docs: <https://ascent.readthedocs.io/en/latest/Tutorials.html>

Example Program Sources

You can find the tutorial example source code and a Makefile in your Ascent install directory under

`examples/ascent/tutorial/ecp_2019`.

Example Makefile:

- [↓ Makefile](#)

Basic Example:

- [↓ ascent_example1.cpp](#)

Conduit Examples:

- [↓ conduit_example1.cpp](#)
- [↓ conduit_example2.cpp](#)
- [↓ conduit_example3.cpp](#)
- [↓ conduit_example4.cpp](#)
- [↓ conduit_example5.cpp](#)

Conduit Blueprint Examples:

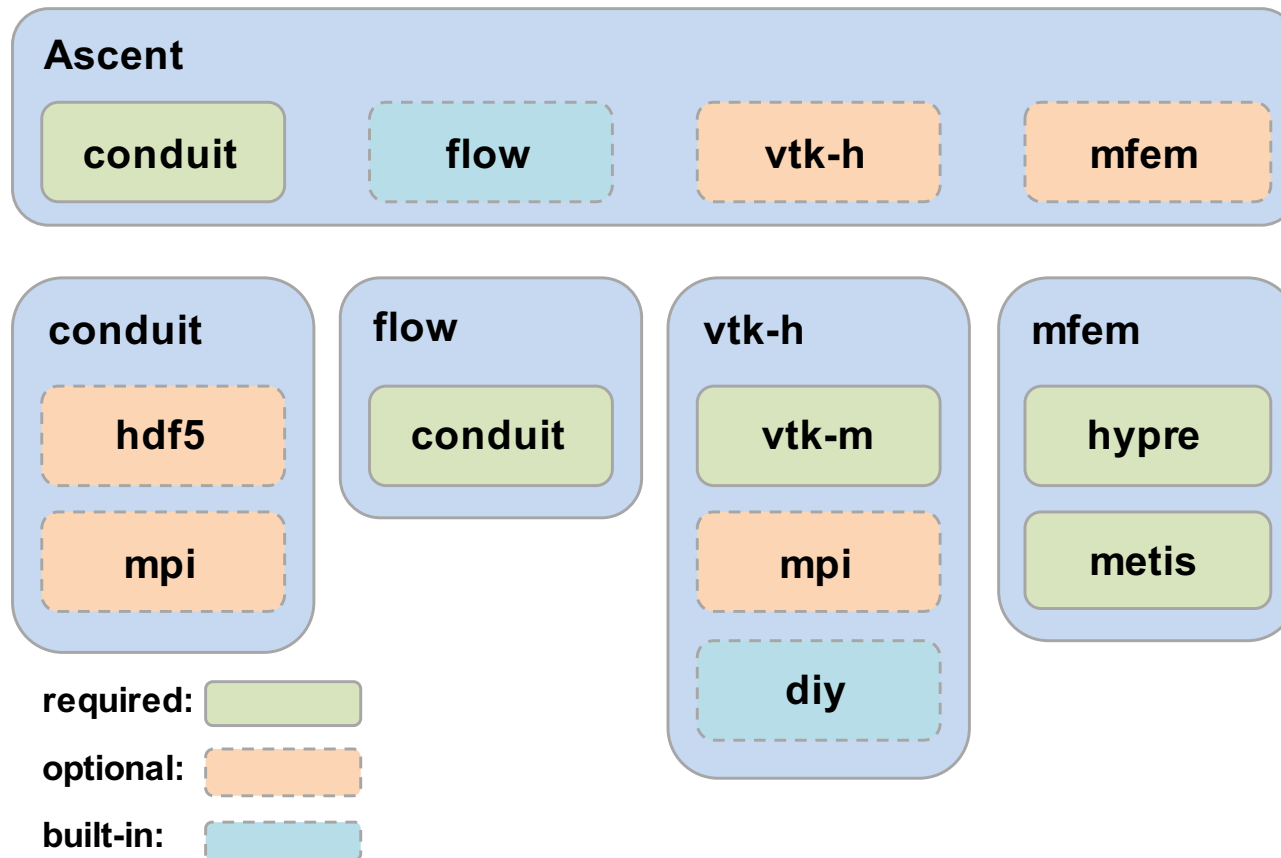
- [↓ blueprint_example2.cpp](#)
- [↓ blueprint_example2.cpp](#)

Scene Examples:

- [↓ ascent_scene_example1.cpp](#)
- [↓ ascent_scene_example2.cpp](#)
- [↓ ascent_scene_example3.cpp](#)
- [↓ ascent_scene_example4.cpp](#)

Sidebar: Ascent's Dependencies

Ascent depends on a few third party libraries:



- What are these libraries?

Libraries covered in this tutorial

- Conduit: library for sharing data
 - (Must understand to use Ascent)
- VTK-m: visualization library for many-core architectures (single node)

Libraries Ascent can utilize

- HDF5: I/O library
- MFEM: finite element discretization library

Libraries Ascent needs (but users don't need to know about)

- VTK-h: expands VTK-m with MPI
 - (h → “hybrid parallel”)
- flow: library for data flow
- diy: library for parallel communication

Tutorial Setup – Testing Ascent with `using-with-make` example

In your install, cd into *examples/ascent/using-with-make*:

```
using-with-make — -bash — 91x26
[harrison37@elder ascent (develop)]$
[harrison37@elder ascent (develop)]$ cd install-debug/examples/ascent/using-with-make/
[harrison37@elder using-with-make (develop)]$
[harrison37@elder using-with-make (develop)]$ ls
Makefile          ascent_render_example.cpp
[harrison37@elder using-with-make (develop)]$
```

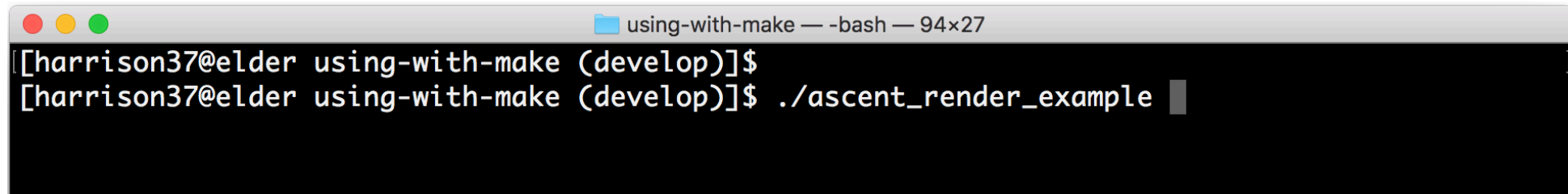

Tutorial Setup – Testing Ascent with `using-with-make` example

Run *make*:

```
using-with-make — -bash — 91x26
[harrison37@elder using-with-make (develop)]$ make
c++ -I /Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/conduit-master-dvy3cvpy7etcfkr3sefndfhshj3eymfe/include/conduit -I /Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/hdf5-1.8.21-lxiikq2xxiaj2y3w7wag6efxoivwf44x/include -I /Users/harrison37/Work/alpine/ascent/install-debug/include/ascent ascent_render_example.cpp -Wl,-rpath,/Users/harrison37/Work/alpine/ascent/install-debug/lib -Wl,-rpath,/Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/conduit-master-dvy3cvpy7etcfkr3sefndfhshj3eymfe -Wl,-rpath,/Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/hdf5-1.8.21-lxiikq2xxiaj2y3w7wag6efxoivwf44x/lib -Wl,-rpath,/Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/vtkh-ascent_ver-wsin76d2vmm6tzrh3s5wte5655xhh23p/lib -Wl,-rpath,/Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/vtkm-ascent_ver-vbrwhgn6yc4mkl3c7ogppxdzrm7pc7da/lib -L /Users/harrison37/Work/alpine/ascent/install-debug/lib -lascent -lflow -lrover -L /Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/vtkh-ascent_ver-wsin76d2vmm6tzrh3s5wte5655xhh23p/lib -lvtkh_rendering -lvtkh_filters -lvtkh_core -lvtkh_lodepng -lvtkh_utils -L /Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/vtkm-ascent_ver-vbrwhgn6yc4mkl3c7ogppxdzrm7pc7da/lib -lvtkm_cont-1.3 -lvtkm_rendering-1.3 -L /Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/conduit-master-dvy3cvpy7etcfkr3sefndfhshj3eymfe/lib -lconduit_blueprint -lconduit_relay -lconduit -L /Users/harrison37/Work/alpine/ascent/uberenv_libs/spack/opt/spack/darwin-highsierra-x86_64/clang-9.0.0-apple/hdf5-1.8.21-lxiikq2xxiaj2y3w7wag6efxoivwf44x/lib -lhdf5 -o ascent_render_example
[harrison37@elder using-with-make (develop)]$
```

Tutorial Setup – Testing Ascent with `using-with-make` example

Run *`./ascent_render_example`*:



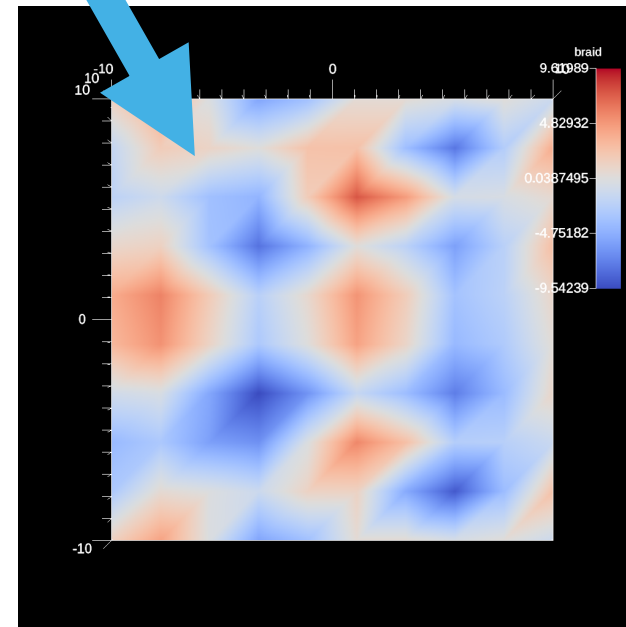
```
using-with-make — -bash — 94x27
[harrison37@elder using-with-make (develop)]$
[harrison37@elder using-with-make (develop)]$ ./ascent_render_example
```

Tutorial Setup – Testing Ascent with `using-with-make` example

Success == the Ascent Mascot and a rendered PNG image file



```
using-with-make -- -bash -- 73x35
[harrison37@elder using-with-make (develop)]$ ls -l
Makefile
ascent_render_example
out_ascent_render_3d100.png
[harrison37@elder using-with-make (develop)]$
```



Tutorial Setup – Testing Ascent with `using-with-make` example

Inside the `using-with-make` Makefile

```
ASCENT_DIR ?= ../../..

# See $(ASCENT_DIR)/share/ascent/ascent_config.mk for detailed linking info
include $(ASCENT_DIR)/share/ascent/ascent_config.mk

INC_FLAGS = $(ASCENT_INCLUDE_FLAGS)
LNK_FLAGS = $(ASCENT_LINK_RPATH) $(ASCENT_LIB_FLAGS)

main:
    $(CXX) $(INC_FLAGS) ascent_render_example.cpp $(LNK_FLAGS) -o ascent_render_example

clean:
    rm -f ascent_render_example
```

Tutorial Setup – Final Setup Notes

- Example for CMake-based build systems:
 - *examples/ascent/using-with-cmake*:
- Tutorial examples also build via *make*:
 - *examples/ascent/tutorial/ecp_2019*
- For many more details about configuring and building Ascent, see:
 - <https://ascent.readthedocs.io/en/latest/BuildingAscent.html>

This tutorial

- 2 hours: Ascent
 - Overview
 - **How to use? (get hands dirty / walk out with understanding on how to integrate)**
 - Tutorial Setup
 - **Compiling Ascent into a simulation code**
 - Hello World in Ascent
 - Conduit Blueprint: data model API
 - Specifying actions
 - Examples of advanced usage (what it can do)
- 1 hour: other ECP vis technologies

Compiling Ascent Into Your Simulation Code

Compiling Ascent Into Your Simulation Code

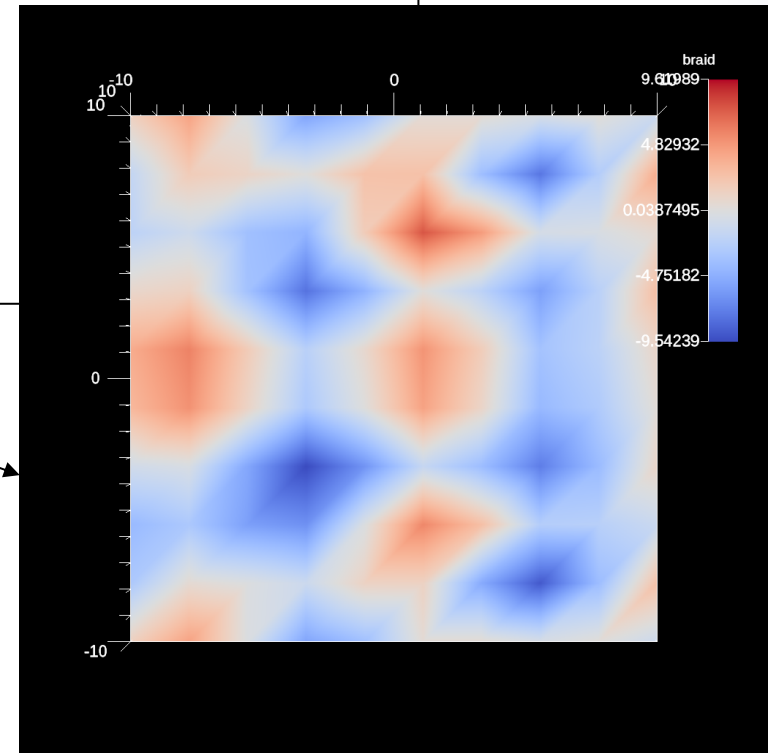
```
hank@alaska:~/sim_example$ cat Makefile
ASCENT_DIR=/home/users/hank/ascent/install-debug
CONDUIT_DIR=/home/users/hank/ascent/uberenv_libs/spack/opt/spack/linux-ubuntu18.04-x86_64/gcc-7.3.0/conduit-master-1mpqbcd6wzdsogygde5u3efb6xswadeo

INC_FLAGS=-I$(ASCENT_DIR)/include/ascent -I$(CONDUIT_DIR)/include/conduit
LINK_FLAGS=-L$(ASCENT_DIR)/lib/ -lascent -L$(CONDUIT_DIR)/lib/ -lconduit -lconduit_relay -lconduit_blueprint

main:
    $(CXX) $(INC_FLAGS) ascent_render_example.cpp $(LINK_FLAGS) -o ascent_render_example
```


Running the example

```
hank@alaska:~/sim_example$ make
g++ -I/home/users/hank/ascent/install-debug/include/ascent -I/home/users/hank/ascent/uberenv_1
ibs/spack/opt/spack/linux-ubuntu18.04-x86_64/gcc-7.3.0/conduit-master-lmpqbcd6wzdsogygde5u3efb
6xswadeo/include/conduit ascent_render_example.cpp -L/home/users/hank/ascent/install-debug/lib
/ -lascent -L/home/users/hank/ascent/uberenv_libs/spack/opt/spack/linux-ubuntu18.04-x86_64/gcc
-7.3.0/conduit-master-lmpqbcd6wzdsogygde5u3efb6xswadeo/lib/ -lconduit -lconduit_relay -lcondui
t_blueprint -o ascent_render_example
hank@alaska:~/sim_example$ echo $LD_LIBRARY_PATH
:/home/users/hank/ascent/uberenv_libs/spack/opt/spack/linux-ubuntu18.04-x86_64/gcc-7.3.0/condu
it-master-lmpqbcd6wzdsogygde5u3efb6xswadeo/lib:/home/users/hank/ascent/install-debug/lib/
hank@alaska:~/sim_example$ ./ascent_render_example
hank@alaska:~/sim_example$ ls -l
total 348
-rw-rw-r-- 1 hank hank    476 Dec 30 10:17 Makefile
-rwxrwxr-x 1 hank hank  14248 Dec 30 10:20 ascent_render_example
-rw-rw-r-- 1 hank hank   1130 Dec 30 10:19 ascent_render_example.cpp
-rw-rw-r-- 1 hank hank 313622 Dec 30 10:20 out_ascent_render_3d100.png
```



This code to generate this picture is covered next in a “hello world” example

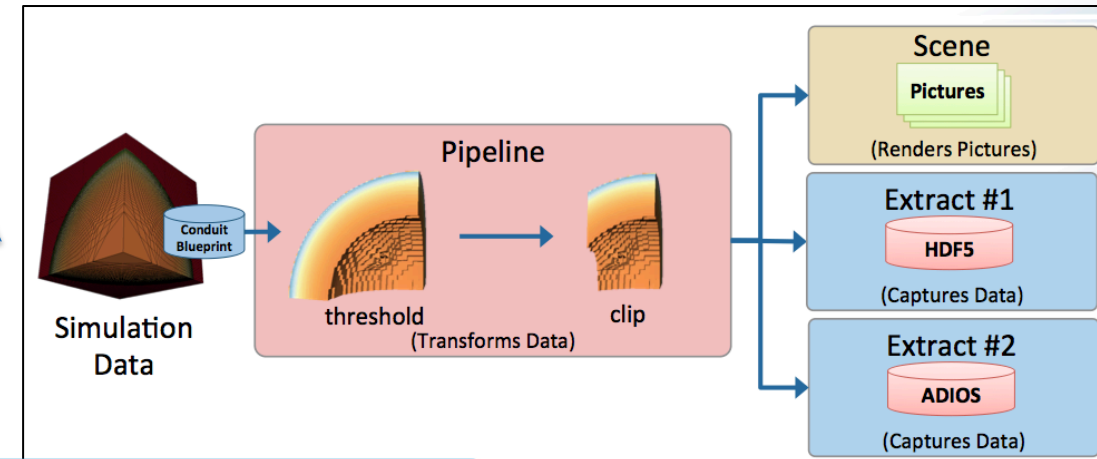
This tutorial

- 2 hours: Ascent
 - Overview
 - **How to use? (get hands dirty / walk out with understanding on how to integrate)**
 - Tutorial Setup
 - Compiling Ascent into a simulation code
 - **Hello World in Ascent**
 - Conduit Blueprint: data model API
 - Specifying actions
 - Examples of advanced usage (what it can do)
- 1 hour: other ECP vis technologies

Ascent: 4 function calls

- Open: initializes Ascent
 - What device to use, etc.
 - Often no arguments specified (use the default)
- Publish: share simulation data with Ascent
- Execute: give Ascent list of actions to execute
- Close: clean up

```
//  
// Run Ascent  
//  
Ascent ascent;  
ascent.open();  
ascent.publish(data);  
ascent.execute(actions);  
ascent.close();
```



Learning Ascent requires understanding Publish and Execute.
“data” and “actions” are set up via Conduit nodes.
(Open and Close are trivial.)

The example from the previous slide: ascent_render_example.cpp

```
hank@alaska:~/sim_example$ cat ascent_render_example.cpp
#include <iostream>
#include "ascent.hpp"
#include "conduit_blueprint.hpp"

using namespace ascent;
using namespace conduit;

int main(int argc, char **argv)
{
    Ascent a;

    // open ascent
    a.open();

    // create example mesh using conduit blueprint
    Node n_mesh;
    conduit::blueprint::mesh::examples::braid("hexs",
                                             10,
                                             10,
                                             10,
                                             n_mesh);

    // publish mesh to ascent
    a.publish(n_mesh);
```

Set up simple
data set using
Conduit routine

```
    // declare a scene to render the dataset
    Node scenes;
    scenes["s1/plots/p1/type"] = "pseudocolor";
    scenes["s1/plots/p1/field"] = "braid";
    // Set the output file name (ascent will add ".png")
    scenes["s1/image_prefix"] = "out_ascent_render_3d";

    // setup actions
    Node actions;
    Node &add_act = actions.append();
    add_act["action"] = "add_scenes";
    add_act["scenes"] = scenes;

    actions.append()["action"] = "execute";

    // execute
    a.execute(actions);

    // close ascent
    a.close();
}
```

Instruct Ascent
to do a
visualization

```
// |
// Run Ascent
// |

Ascent ascent;
ascent.open();
ascent.publish(data);
ascent.execute(actions);
ascent.close();
```

Next up for this tutorial: understand this code!
(i.e., how to make "n_mesh"/"data" & "actions")

Important Note:

Ascent Can Be Controlled In Multiple Ways

- All of the examples in this tutorial use C++
 - For publishing mesh data
 - For instructing Ascent to do visualization
- Other supported languages (not shown in the tutorial):
 - Fortran bindings
 - Python bindings
- Also JSON support
 - JSON is used only for instructing Ascent to do visualization
 - Workflow with JSON:
 - Create binary with simulation code and Ascent. Only Ascent instruction is to use JSON.
 - At runtime, direct simulation code to use appropriate JSON file.

This tutorial

- 2 hours: Ascent
 - Overview
 - **How to use? (get hands dirty / walk out with understanding on how to integrate)**
 - Tutorial Setup
 - Compiling Ascent into a simulation code
 - Hello World in Ascent
 - **Conduit Blueprint: data model API**
 - Specifying actions
 - Examples of advanced usage (what it can do)
- 1 hour: other ECP vis technologies

Simple Example: ascent_render_example.cpp

```
hank@alaska:~/sim_example$ cat ascent_render_example.cpp
#include <iostream>
#include "ascent.hpp"
#include "conduit_blueprint.hpp"

using namespace ascent;
using namespace conduit;

int main(int argc, char **argv)
{
    Ascent a;

    // open ascent
    a.open();

    // create example mesh using conduit blueprint
    Node n_mesh;
    conduit::blueprint::mesh::examples::braid("hexs",
                                             10,
                                             10,
                                             10,
                                             n_mesh);

    // publish mesh to ascent
    a.publish(n_mesh);
}
```

```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "braid";
// Set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "out_ascent_render_3d";

// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_scenes";
add_act["scenes"] = scenes;

actions.append()["action"] = "execute";

// execute
a.execute(actions);

// close ascent
a.close();
}
```

```
// |
// Run Ascent
// |

Ascent ascent;
ascent.open();
ascent.publish(data);
ascent.execute(actions);
ascent.close();
```

The code above makes a toy mesh.

Question: how can we publish your simulation data to Ascent?

Conduit & Blueprint



- Conduit is a library
 - Defines a model for:
 - describing hierarchical data
 - in C++, C, Fortran, and Python.
 - Used for data coupling between packages in-core, serialization, and I/O tasks
- Blueprint is a set of hierarchical conventions to describe mesh-based simulation data both in-memory and via files
- Relationship: data can be stored in Conduit using Blueprint conventions

Conduit basics

Conduit basics

- Primary object is called a “Node”
- Stores as key/value

```
#include <iostream>
#include "conduit_blueprint.hpp"

using namespace conduit;

int main(int argc, char **argv)
{
    Node n;
    n["key"] = "value";
    n.print();
}

hank@alaska:~/conduit_example$ make -q
hank@alaska:~/conduit_example$ ./conduit_example

{
  "key": "value"
}
```

Conduit basics

- Can store data hierarchically
- Paths look like Unix directory structure

```
#include <iostream>
#include "conduit_blueprint.hpp"

using namespace conduit;

int main(int argc, char **argv)
{
    Node n;
    n["dir1/dir2/val1"] = 100.5;
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "dir1":
  {
    "dir2":
    {
      "val1": 100.5
    }
  }
}
```

No need to use “dir” names...

```
#include <iostream>
#include "conduit_blueprint.hpp"

using namespace conduit;

int main(int argc, char **argv)
{
    Node n,
    n["a/b/val1"] = 100.5;
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "a":
  {
    "b":
    {
      "val1": 100.5
    }
  }
}
```

```
#include <iostream>
#include "conduit_blueprint.hpp"

using namespace conduit;

int main(int argc, char **argv)
{
    Node n;
    n["dir1/dir2/val1"] = 100.5;
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "dir1":
  {
    "dir2":
    {
      "val1": 100.5
    }
  }
}
```

conduit::Node::set

- set: method for creating an array in a conduit::Node

```
hank@alaska:~/conduit_example$ cat conduit_example.cpp
#include <iostream>
#include "conduit_blueprint.hpp"

using namespace conduit;

int main()
{
    Node n;
    int *A = new int [20];
    A[0] = 0;
    A[1] = 1;
    for (int i = 2 ; i < 20 ; i++)
        A[i] = A[i-2]+A[i-1];
    n["fib"].set(A, 20);
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "fib": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
233, 377, 610, 987, 1597, 2584, 4181]
}
```

Conduit::Node::set is smart about STL vectors

```
hank@alaska:~/conduit_example$ cat conduit_example.cpp
#include <iostream>
#include <vector>
#include "conduit_blueprint.hpp"

using namespace conduit;
using std::vector;

int main()
{
    Node n;
    vector<int> A(20);
    A[0] = 0;
    A[1] = 1;
    for (int i = 2 ; i < 20 ; i++)
        A[i] = A[i-2]+A[i-1];
    n["fib"].set(A);
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "fib": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
233, 377, 610, 987, 1597, 2584, 4181]
}
```

```
hank@alaska:~/conduit_example$ cat conduit_example.cpp
#include <iostream>
#include "conduit_blueprint.hpp"

using namespace conduit;

int main()
{
    Node n;
    int *A = new int [20];
    A[0] = 0;
    A[1] = 1;
    for (int i = 2 ; i < 20 ; i++)
        A[i] = A[i-2]+A[i-1];
    n["fib"].set(A, 20);
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "fib": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
233, 377, 610, 987, 1597, 2584, 4181]
}
```

For STL vectors, Conduit::Node::set can be called via the assignment operator (but it gets confused with non-vector arrays, i.e., pointers)

```
hank@alaska:~/conduit_example$ cat conduit_example.cpp
#include <iostream>
#include <vector>
#include "conduit_blueprint.hpp"

using namespace conduit;
using std::vector;

int main()
{
    Node n;
    vector<int> A(20);
    A[0] = 0;
    A[1] = 1;
    for (int i = 2 ; i < 20 ; i++)
        A[i] = A[i-2]+A[i-1];
    n["fib"].set(A);
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "fib": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
233, 377, 610, 987, 1597, 2584, 4181]
}
```

```
hank@alaska:~/conduit_example$ cat conduit_example.cpp
#include <iostream>
#include <vector>
#include "conduit_blueprint.hpp"

using namespace conduit;
using std::vector;

int main()
{
    Node n;
    vector<int> A(20);
    A[0] = 0;
    A[1] = 1;
    for (int i = 2 ; i < 20 ; i++)
        A[i] = A[i-2]+A[i-1];
    n["fib"] = A;
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "fib": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
233, 377, 610, 987, 1597, 2584, 4181]
}
```

conduit::Node::set_external

- `set_external`: another method for creating an array in a `conduit::Node`
 - But doesn't allocate new memory

	Method	Makes copy of your array
	<code>conduit::Node::set</code>	YES
	<code>conduit::Node::set_external</code>	NO

```
chank@alaska:~/conduit_example$ cat conduit_example.cpp
#include <iostream>
#include <vector>
#include "conduit_blueprint.hpp"

using namespace conduit;
using std::vector;

int main()
{
    Node n;
    vector<int> A(20);
    A[0] = 0;
    A[1] = 1;
    for (int i = 2 ; i < 20 ; i++)
        A[i] = A[i-2]+A[i-1];
    n["fib"].set_external(A);
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "fib": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,
610, 987, 1597, 2584, 4181]
}
```


Shallow Vs Deep Copy

```
chank@alaska:~/conduit_example$ cat conduit_example.cpp
#include <iostream>
#include <vector>
#include "conduit_blueprint.hpp"

using namespace conduit;
using std::vector;

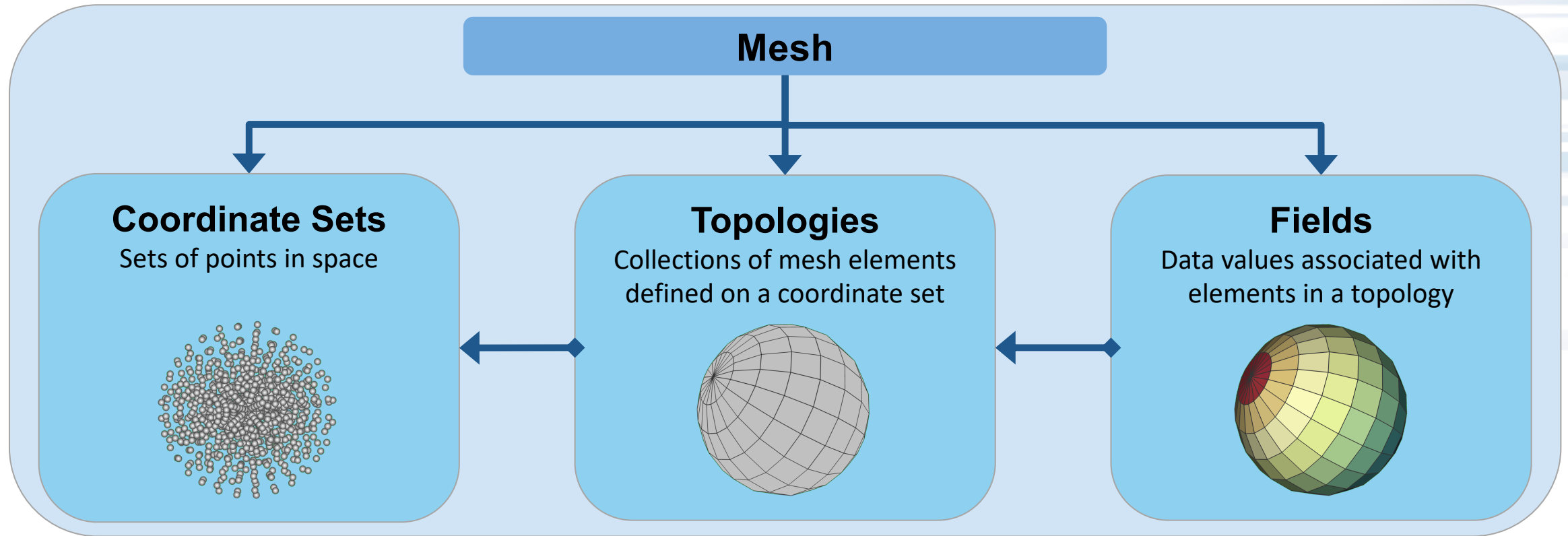
int main()
{
    Node n;
    vector<int> A1(20), A2(20);
    A1[0] = 0;  A2[0] = 0;
    A1[1] = 1;  A2[1] = 1;
    for (int i = 2 ; i < 20 ; i++)
    {
        A1[i] = A1[i-2]+A1[i-1];
        A2[i] = A2[i-2]+A2[i-1];
    }
    n["fib_deep_copy"].set(A1);
    // shal_copy --> shallow_copy
    n["fib_shal_copy"].set_external(A2);
    A1[10] = -1;
    A2[10] = -1;
    n.print();
}

hank@alaska:~/conduit_example$ ./conduit_example

{
  "fib_deep_copy": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
233, 377, 610, 987, 1597, 2584, 4181],
  "fib_shal_copy": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, -1, 89, 144,
233, 377, 610, 987, 1597, 2584, 4181]
}
```

Blueprint

The Mesh Blueprint supports mesh constructs common in several full featured mesh data models



Ideas were shaped by surveying projects including: ADIOS, BoxLib, Chombo, Damaris, EAVL, Exodus, ITAPS, MFEM, SAF, SAMRAI, Silo, VisIt's AVT, VTK, VTK-m, Xdmf.

Blueprint covers a wide range of mesh descriptions

- **Coordinate Sets**

- 1D/2D/3D
- Cartesian, Cylindrical, Spherical
- Implicit: Uniform, Rectilinear
- Explicit

- **Topologies**

- Implicit: Points, Uniform, Rectilinear, Structured
- Unstructured
[Points, Lines, Quads, Tris, Tets, Hexs]
- Optional MFEM Grid Function support
- Arbitrary Polygonal and Polyhedral
(Active development)
- Unstructured heterogenous element shapes
(Planned for future)

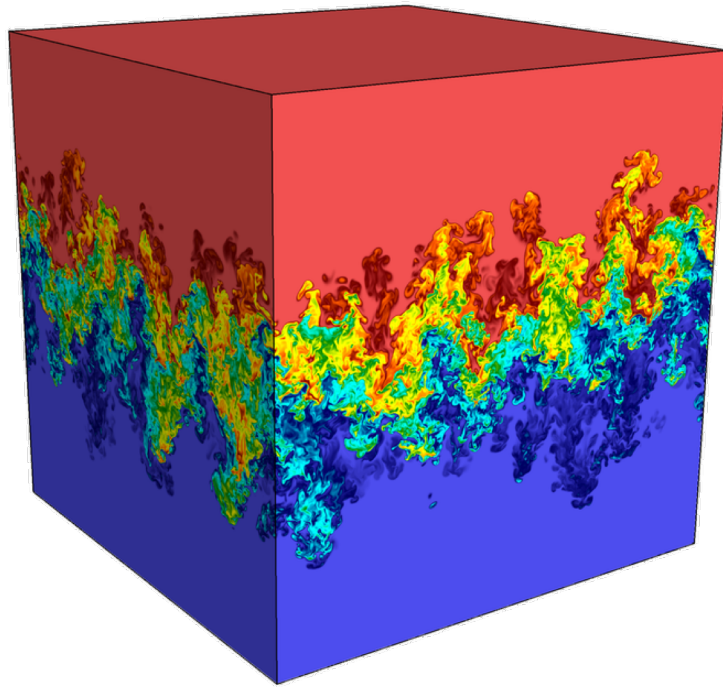
- **Fields**

- Vertex or Element associated
- Multi-component field arrays
- Optional MFEM Grid Function Basis support
- Multi-dimensional field arrays
(Planned for future)
- Sparse representations for field arrays
(Planned for future)

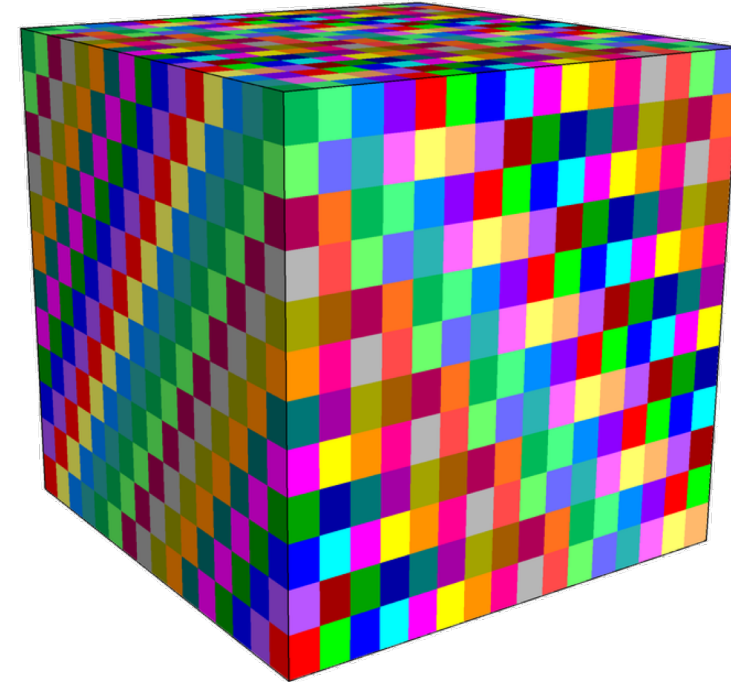
- **Domain Decomposition Info**

- Basic State Info [Domain Ids]
- Domain Adjacency Info for Unstructured Meshes
- Domain Adjacency Info for Structured Meshes
(Planned for future)
- Nesting Info for Block-Structured AMR Meshes
(Active development)

The structure of the Blueprint is designed with distributed-memory parallelism in mind



Full Dataset

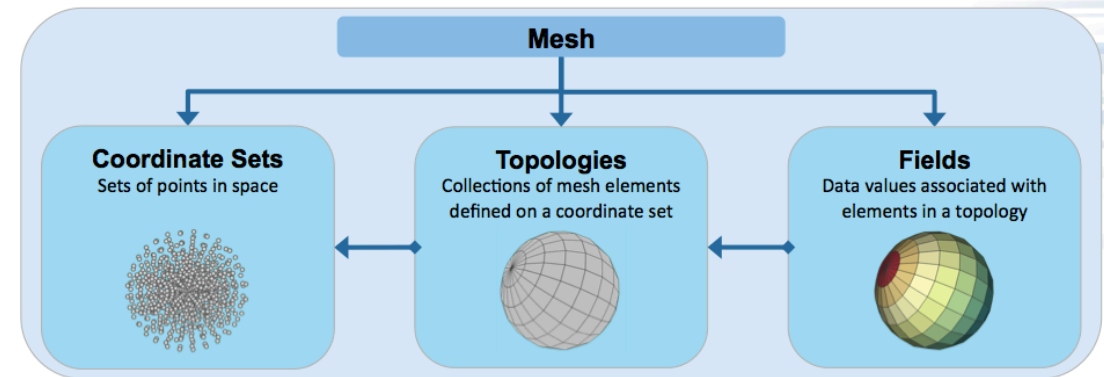


Domain Decomposition

Any info required to describe to domain decomposition, nesting, or abutment is local

Blueprint basics

- Set up three things for a Mesh:
 - Coordinate sets (points in space)
 - Topologies (elements defined on coordinate sets)
 - Fields (data values on the elements)



- Specifics of how to set these up depend on the mesh type

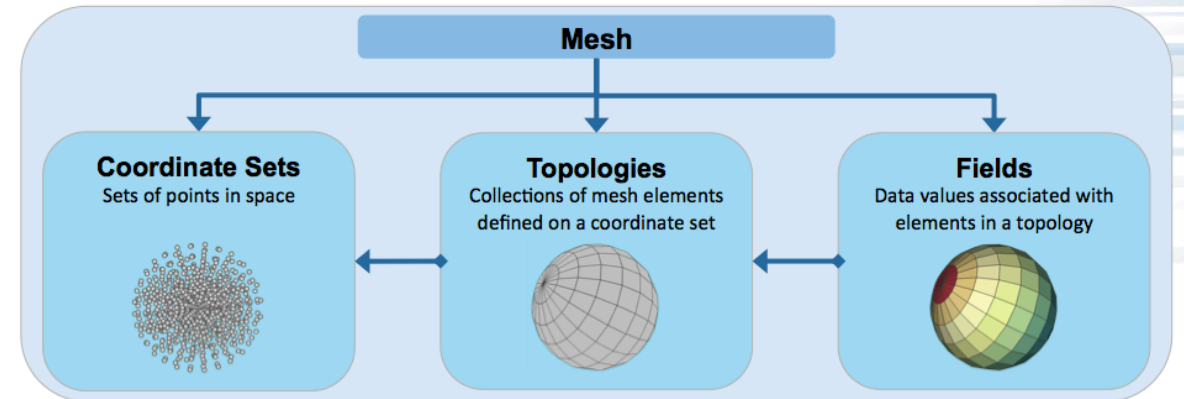
Blueprint syntax (1/3)

- Basic syntax:

```
Node mesh;  
mesh["<THING TO SET>/<VAR NAME>/<PROPERTY>"] = VALUE;
```

- **THING TO SET:** only 3 possible values

- “coordsets”
- “topologies”
- “fields”



- **VAR NAME:** you choose the variable name, and you can have multiple variables of each type
 - Multiple coordsets
 - Multiple topologies
 - Multiple fields

Blueprint syntax (2/3)

- Basic syntax:

```
Node mesh;  
mesh["<THING TO SET>/<VAR NAME>/<PROPERTY>"] = VALUE;
```

- Example:

```
Node mesh;  
mesh["fields/temperature/association"] = "elements";
```

- What does the example do?
 - Declares that the field named temperature has an association on the elements
 - (it is defined in the cell centers, not on the vertices)

Learning Blueprint means learning which keywords describe your data.
The list of keywords are available in the Blueprint documentation.

Blueprint syntax (3/3)

- Basic syntax:

```
Node mesh;  
mesh["<THING TO SET>/<VAR NAME>/<PROPERTY>"] = VALUE;
```

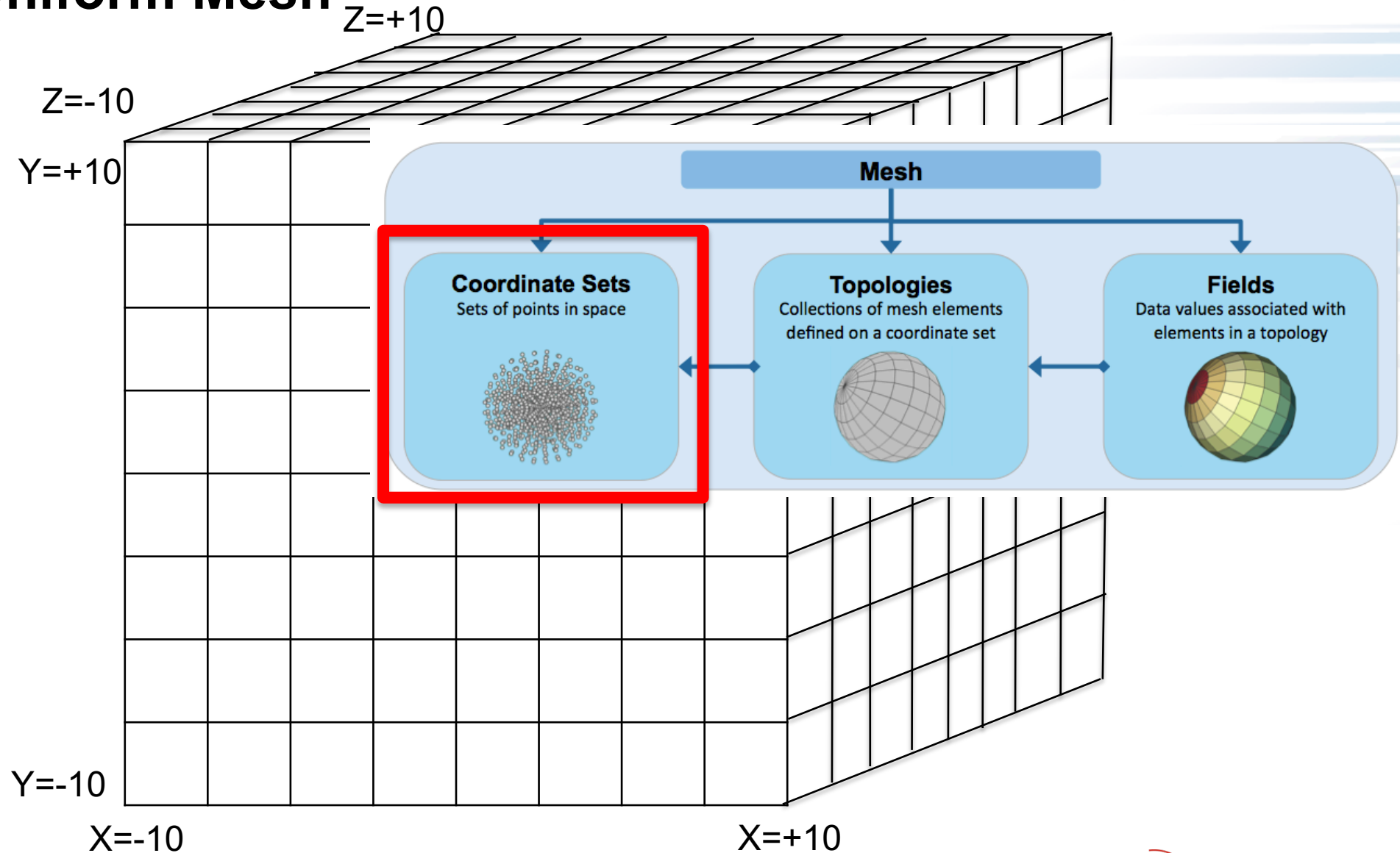
- Variant syntax:

```
Node mesh;  
mesh["<THING TO SET>/<VAR NAME>/<MAIN PROPERTY>/<SUB PROPERTY>"] = VALUE;
```

- This makes sense from a Conduit perspective, since Conduit stores data hierarchically.
- Concrete example:

```
Node mesh;  
mesh["coordsets/coords/origin/x"] = -30;  
mesh["coordsets/coords/origin/y"] = +10;  
mesh["coordsets/coords/origin/z"] = -10;
```

Example #1: Uniform Mesh



Points can be defined implicitly.

9 points per dimension:
{ -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10 }

By specifying
9 points in X,
9 points in Y,
9 points in Z
→ we can infer 9^3 points overall

Example #1: uniform mesh (1 of 3): coordsets

Declare a Conduit node object called mesh

Using “coordsets” keyword & declare a coordsets variable named “coords.”
Declare the coords variable has uniform spacing using the “uniform” keyword.
Note keywords appear in both the key (“coordsets”) and values (“uniform”)

More conventions for coordsets – “dims/i”, “dims/j”, “origin/x”, “spacing/dx”, etc.

Node mesh;

Will be a 9x9x9 mesh

```
int numPerDim = 9;
// create the coordinate set
mesh["coordsets/coords/type"] = "uniform";
mesh["coordsets/coords/dims/i"] = numPerDim;
mesh["coordsets/coords/dims/j"] = numPerDim;
mesh["coordsets/coords/dims/k"] = numPerDim;

// add origin and spacing to the coordset (optional)
mesh["coordsets/coords/origin/x"] = -10.0;
mesh["coordsets/coords/origin/y"] = -10.0;
mesh["coordsets/coords/origin/z"] = -10.0;
double distancePerStep = 20.0/(numPerDim-1);
mesh["coordsets/coords/spacing/dx"] = distancePerStep;
mesh["coordsets/coords/spacing/dy"] = distancePerStep;
mesh["coordsets/coords/spacing/dz"] = distancePerStep;
```

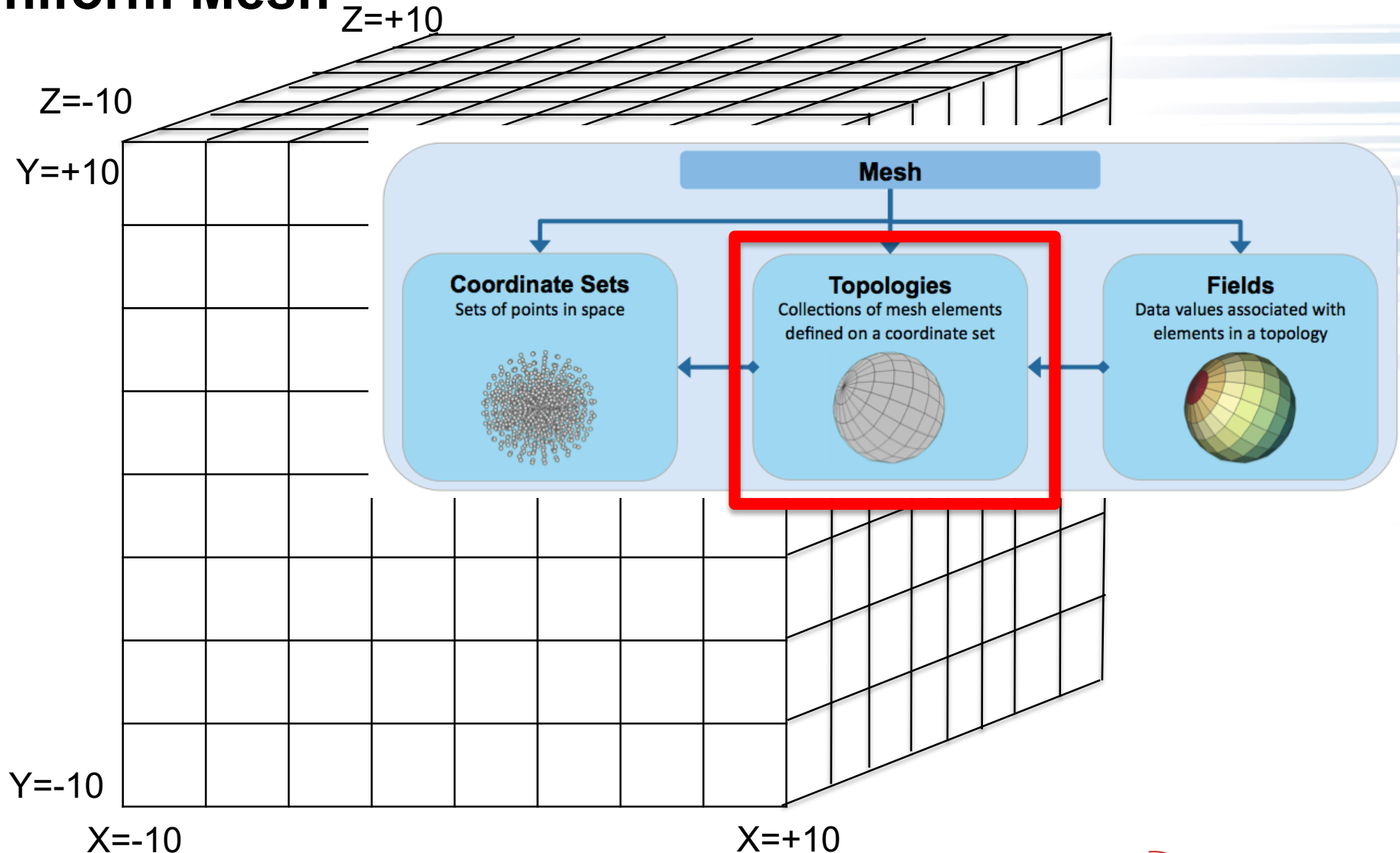
Example #1: Uniform Mesh

9 points per dimension:
{ -10, -7.5, -5, -2.5, 0,
2.5, 5, 7.5, 10}

Points can be defined implicitly.

By specifying
9 points in X,
9 points in Y,
9 points in Z
→ we can infer 9^3 points
overall

As far as cells, 8^3 cells,
with cells in between
each grouping of 8
points.



Example #1: uniform mesh (2 of 3): coordsets

Conventions for topologies.
“uniform” is used again.
This is because we are now
specifying uniform topology (where
previously it was coordinates)

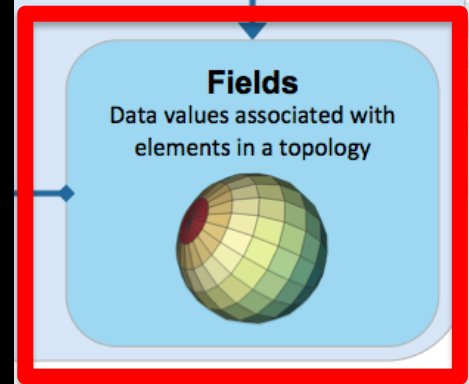
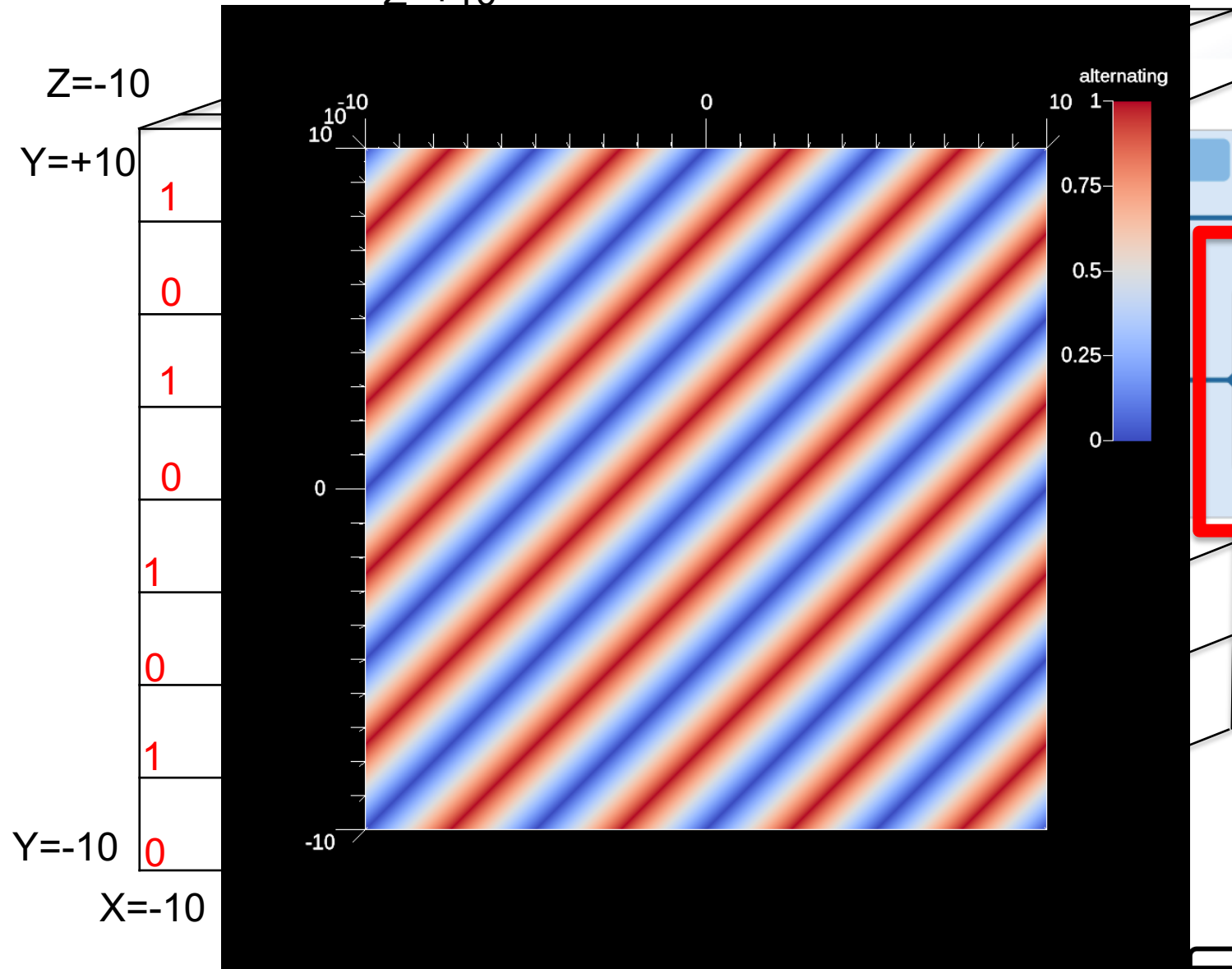
```
Node mesh;
```

```
int numPerDim = 9;  
// create the coordinate set  
mesh["coordsets/coords/type"] = "uniform";  
mesh["coordsets/coords/dims/i"] = numPerDim;  
mesh["coordsets/coords/dims/j"] = numPerDim;  
mesh["coordsets/coords/dims/k"] = numPerDim;  
  
// add origin and spacing to the coordset (optional)  
mesh["coordsets/coords/origin/x"] = -10.0;  
mesh["coordsets/coords/origin/y"] = -10.0;  
mesh["coordsets/coords/origin/z"] = -10.0;  
double distancePerStep = 20.0/(numPerDim-1);  
mesh["coordsets/coords/spacing/dx"] = distancePerStep;  
mesh["coordsets/coords/spacing/dy"] = distancePerStep;  
mesh["coordsets/coords/spacing/dz"] = distancePerStep;  
  
// add the topology  
// this case is simple b/c it's implicitly derived from the coordset  
mesh["topologies/topo/type"] = "uniform";  
// reference the coordinate set by name  
mesh["topologies/topo/coordset"] = "coords";
```

Coordset for this topology is “coords”

Example #1: Uniform Mesh $Z=+10$

Example:
field called
“alternating”
defined on vertices.
If the vertex ID is
even, then value is
0. Else value is 1.



Example #1: uniform mesh (3 of 3): fields

Set up trivial field

Using “fields” keyword to declare a field called “alternating.”
Declare its association is with “vertex” (not “element”).
Declare it belong on topology “topo.”
Use “set_external” to set values.

Verify function to make sure we set it up right

Publish data to ascent

```
int numVertices = numPerDim*numPerDim*numPerDim; // 3D
float *vals = new float[numVertices];
for (int i = 0 ; i < numVertices ; i++)
    vals[i] = ( (i%2)==0 ? 0.0 : 1.0);

mesh["fields/alternating/association"] = "vertex";
mesh["fields/alternating/topology"] = "topo";
mesh["fields/alternating/values"].set_external(vals, numVertices);

Node verify_info;
if(!blueprint::mesh::verify(mesh, verify_info))
{
    std::cout << "Verify failed!" << std::endl;
    verify_info.print();
}

// publish mesh to ascent
a.publish(mesh);
```

Code: blueprint_example1.cpp

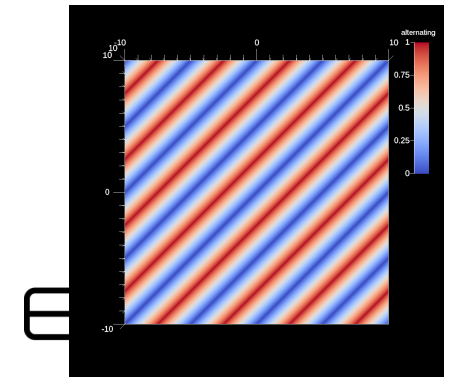
Coming up:

```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "alternating";
// Set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "out_ascent_render_uniform";
```

```
// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_scenes";
add_act["scenes"] = scenes;

actions.append()["action"] = "execute";

// execute
a.execute(actions);
```



Example for setting up uniform grid (all in one slide)

```
Node mesh;

int numPerDim = 9;
// create the coordinate set
mesh["coordsets/coords/type"] = "uniform";
mesh["coordsets/coords/dims/i"] = numPerDim;
mesh["coordsets/coords/dims/j"] = numPerDim;
mesh["coordsets/coords/dims/k"] = numPerDim;

// add origin and spacing to the coordset (optional)
mesh["coordsets/coords/origin/x"] = -10.0;
mesh["coordsets/coords/origin/y"] = -10.0;
mesh["coordsets/coords/origin/z"] = -10.0;
double distancePerStep = 20.0/(numPerDim-1);
mesh["coordsets/coords/spacing/dx"] = distancePerStep;
mesh["coordsets/coords/spacing/dy"] = distancePerStep;
mesh["coordsets/coords/spacing/dz"] = distancePerStep;

// add the topology
// this case is simple b/c it's implicitly derived from
mesh["topologies/topo/type"] = "uniform";
// reference the coordinate set by name
mesh["topologies/topo/coordset"] = "coords";
```

```
int numVertices = numPerDim*numPerDim*numPerDim; // 3D
float *vals = new float[numVertices];
for (int i = 0 ; i < numVertices ; i++)
    vals[i] = ( (i%2)==0 ? 0.0 : 1.0);

mesh["fields/alternating/association"] = "vertex";
mesh["fields/alternating/topology"] = "topo";
mesh["fields/alternating/values"].set_external(vals, numVertices);

Node verify_info;
if(!blueprint::mesh::verify(mesh, verify_info))
{
    std::cout << "Verify failed!" << std::endl;
    verify_info.print();
}

// publish mesh to ascent
a.publish(mesh);
```

```
//
// Run Ascent
//

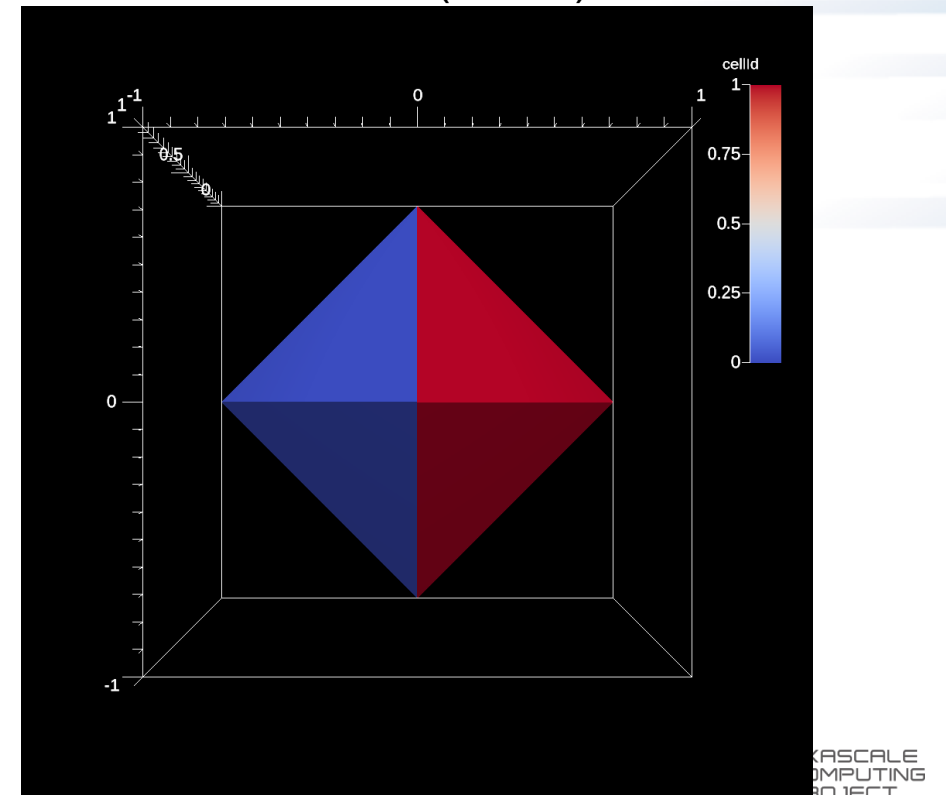
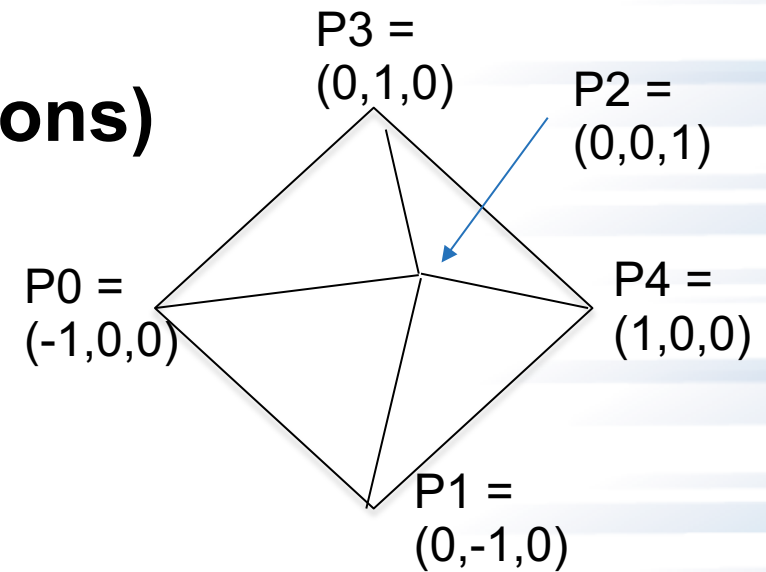
Ascent ascent;
ascent.open();
ascent.publish(data);
ascent.execute(actions);
ascent.close();
```


Unstructured Mesh Example (2 Tetrahedrons)

```
// create the coordinate set
double X[5] = { -1.0, 0.0, 0.0, 0.0, 1.0 };
double Y[5] = { 0.0, -1.0, 0.0, 1.0, 0.0 };
double Z[5] = { 0.0, 0.0, 1.0, 0.0, 0.0 };
mesh["coordsets/coords/type"] = "explicit";
mesh["coordsets/coords/values/x"].set_external(X, 5);
mesh["coordsets/coords/values/y"].set_external(Y, 5);
mesh["coordsets/coords/values/z"].set_external(Z, 5);

// create the topology
mesh["topologies/mesh/type"] = "unstructured";
mesh["topologies/mesh/coordset"] = "coords";
mesh["topologies/mesh/elements/shape"] = "tet";
int64 connectivity[8] = { 0, 1, 3, 2, 4, 3, 1, 2 };
mesh["topologies/mesh/elements/connectivity"]
    .set_external(connectivity, 8);

// create the field
const int numCells = 2;
float vals[numCells] = { 0, 1 };
mesh["fields/cellId/association"] = "element";
mesh["fields/cellId/topology"] = "mesh";
mesh["fields/cellId/volume_dependent"] = "false";
mesh["fields/cellId/values"].set_external(vals, 2);
```



Learning the Blueprint Interface:

<https://lnl-conduit.readthedocs.io/en/latest/blueprint.html#blueprint-interface>

Protocol Details

- marray
 - Protocol
 - Properties and Transforms
 - Examples
- mesh
 - Protocol
 - Coordinate Sets
 - Topologies
 - Topology Nomenclature
 - Association with a Coordinate Set
 - Optional association with a Grid Function
 - Implicit Topology
 - Explicit (Unstructured) Topology
 - Single Shape Topologies
 - Mixed Shape Topologies
 - Element Windings
 - Polygonal/Polyhedral Topologies
 - (Optional) Element Offsets
 - Material Sets
 - Fields
 - Topology Association for Field Values
 - Adjacency Sets
 - State

- Examples
 - basic
 - Uniform
 - Rectilinear
 - Structured
 - Tris
 - Quads
 - Polygons
 - Tets
 - Hexs
 - Polyhedra
 - braid
 - spiral
 - julia
 - polytess
 - miscellaneous
 - Outputting Meshes for Visualization
 - Detailed Uniform Example

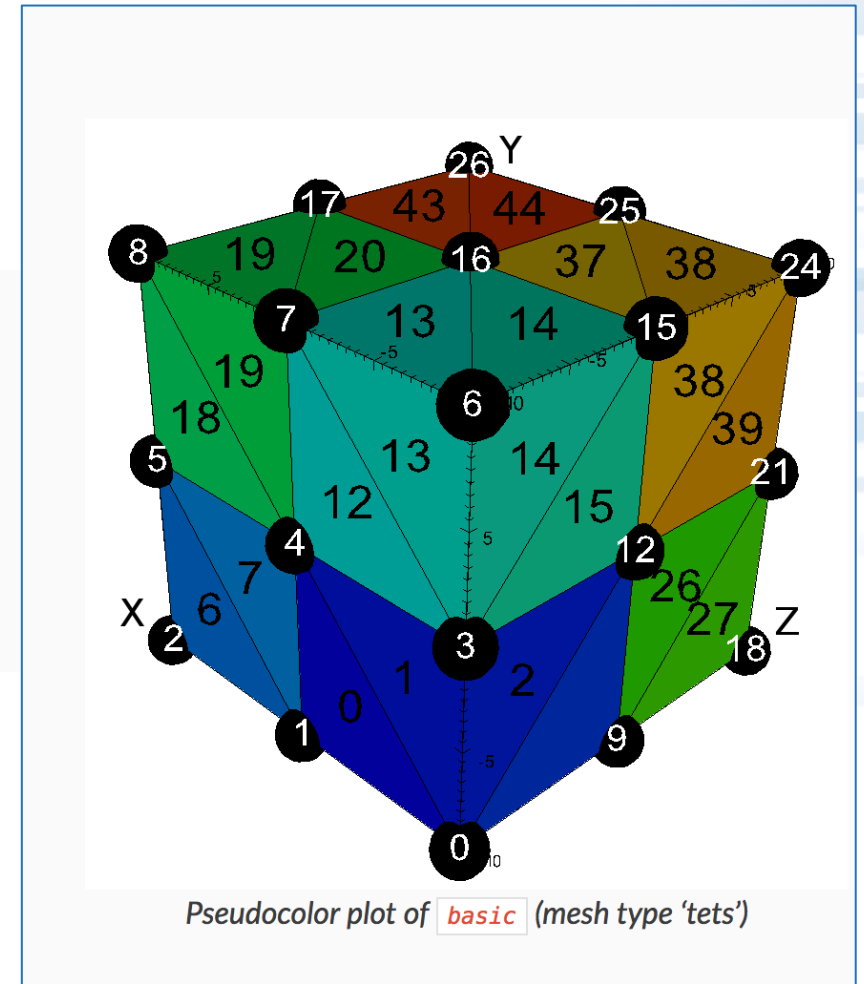
Learning the Blueprint Interface (tets):

https://lnl-conduit.readthedocs.io/en/latest/blueprint_mesh.html#tets

- Usage Example

```
// create container node
Node mesh;
// generate simple explicit tri-based 3d 'basic' mesh
conduit::blueprint::mesh::examples::basic("tets", 3, 3, 3, mesh);
// print out results
mesh.print();
```

- Blueprint has methods for creating all different mesh types
- You can then examine the results and see intended layout



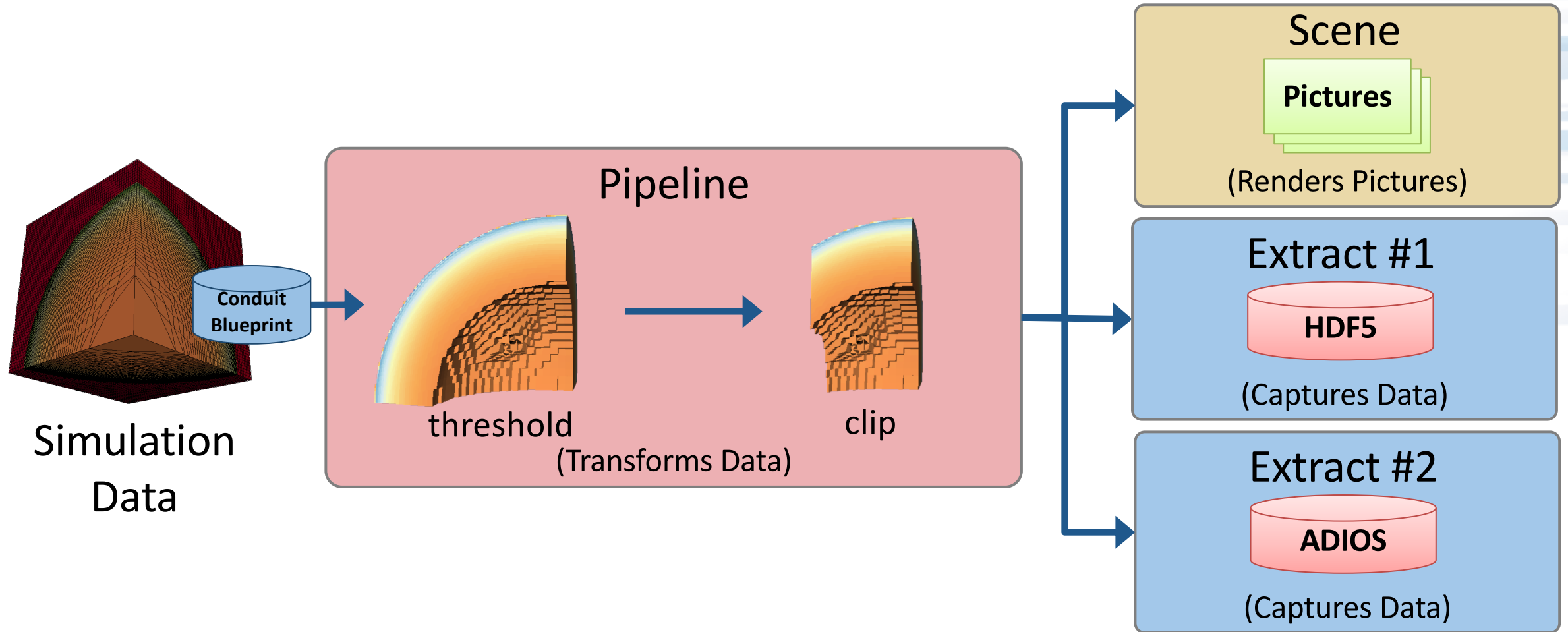
This tutorial

- 2 hours: Ascent
 - Overview
 - **How to use? (get hands dirty / walk out with understanding on how to integrate)**
 - Tutorial Setup
 - Compiling Ascent into a simulation code
 - Hello World in Ascent
 - Conduit Blueprint: data model API
 - **Specifying actions**
 - Examples of advanced usage (what it can do)
- 1 hour: other ECP vis technologies

Slide Repeat: Ascent's API is composed of three key concepts

- Pipelines (transform data):
 - Allows users to describe how they want to transform their data
- Scenes (make pictures):
 - Allows users to describe the pictures they want to create
- Extracts (capture data):
 - Allows users to describe how they want capture data

Slide Repeat: Ascent end-to-end conceptual example



How Does It Work?

- Answer: very similarly to publishing data
 - Create Conduit Nodes.
 - Be aware of keywords for keys and values
 - Connect the Conduit Nodes as “actions” for Ascent

```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "braid";
// Set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "out_ascent_render_3d";

// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_scenes";
add_act["scenes"] = scenes;

actions.append()["action"] = "execute";

// execute
a.execute(actions);
```


How Does It Work?

- Answer: very similarly to publishing data
 - Create Conduit Nodes.
 - Be aware of keywords for keys and values
 - Connect the Conduit Nodes as “actions” for Ascent

Known keyword. Tells Ascent you will be describing a plot.

You can have multiple plots. Refer to this plot as “p1.”

You can have multiple scenes. Refer to this scene as “s1.”

Known keyword. Tells Ascent the type of the plot.

```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "braid";
// Set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "out_ascent_render_3d";

// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_scenes";
add_act["scenes"] = scenes;

actions.append()["action"] = "execute";

// execute
a.execute(actions);
```

How Does It Work?

- Answer: very similarly to publishing data
 - Create Conduit Nodes.
 - Be aware of keywords for keys and values
 - Connect the Conduit Nodes as “actions” for Ascent

Known keyword. Tells Ascent you will be describing a plot.

You can have multiple plots. Refer to this plot as “p1.”

You can have multiple scenes. Refer to this scene as “s1.”

Known keyword. Tells Ascent the type of the plot

Tells Ascent that the “scenes” Node is of type “scene.”

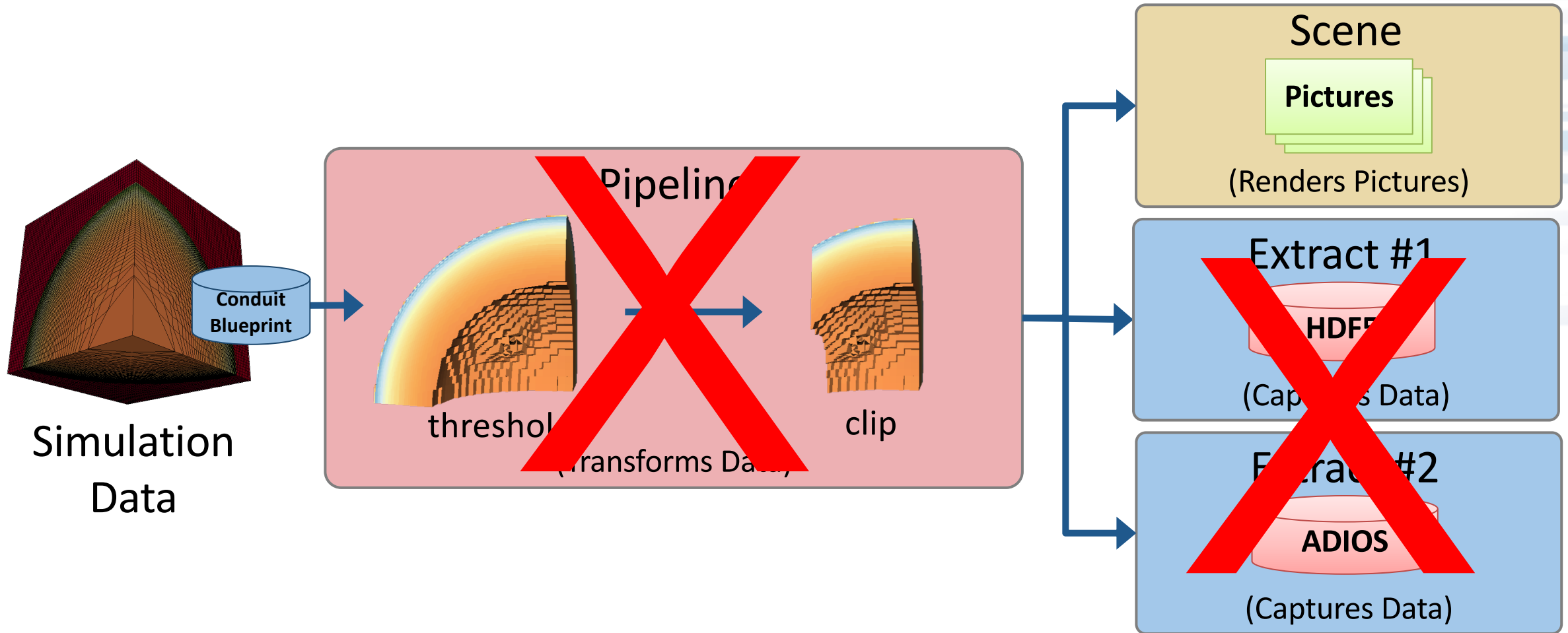
```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "braid";
// Set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "out_ascent_render_3d";

// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_scenes";
add_act["scenes"] = scenes;

actions.append()["action"] = "execute";

// execute
a.execute(actions);
```

Very Simple Example: empty Pipeline, no Extracts, one Scene



Scenes are composed of plots

- So far, three types of plots:
 - Pseudocolor:
 - map scalars to colors
 - if a volume, then only render the exterior faces of that volume
 - Volume rendering:
 - map scalars to color and transparency
 - only works on volumes, fails on other mesh types
 - Mesh:
 - draws outline of each element

Very Simple Example: no Pipeline, no Extracts, one Scene

Declare a Conduit node object called "scenes"

The first scene is called "s1."
The first plot is called "p1."

The plot p1 is a pseudocolor, indicated by setting its "type" (reserved keyword) to value "pseudocolor" (reserved keyword).

The name of the image to save is associated with the scene ("s1/image_prefix").

```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "braid";
// Set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "out_ascent_render_3d";

// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_scenes";
add_act["scenes"] = scenes;

actions.append()["action"] = "execute";

// execute
a.execute(actions);
```

Some standard methods for telling Ascent to carry out the scene

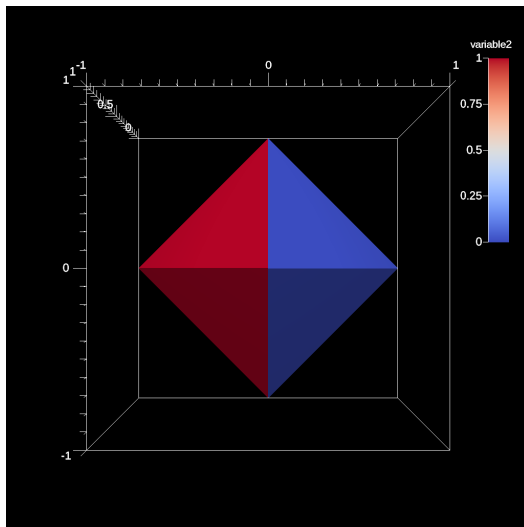
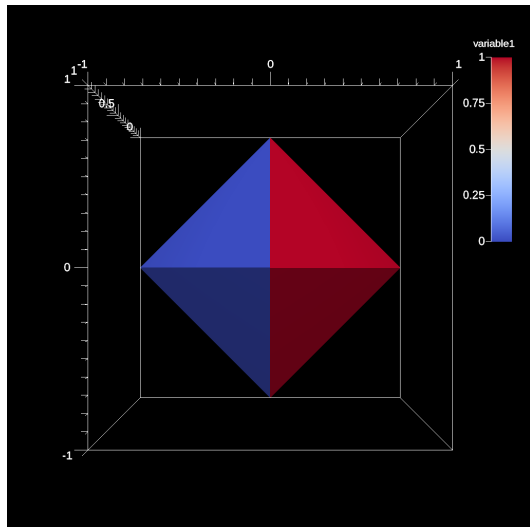
No Pipeline, no Extracts, two Scenes

Same:
declare a Conduit node object called "scenes"

The first scene is still called "s1",
The second scene is called "s2."

s2 is distinct from s1 since it is visualizing a
different variable (variable2 vs variable1).

The name of the image to save is
associated with the scene, and it
changes between the scenes as well.



```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "variable1";
scenes["s1/image_prefix"] = "out_ascent_render_var1";

scenes["s2/plots/p1/type"] = "pseudocolor";
scenes["s2/plots/p1/field"] = "variable2";
scenes["s2/image_prefix"] = "out_ascent_render_var2";
```

```
// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_scenes";
add_act["scenes"] = scenes;

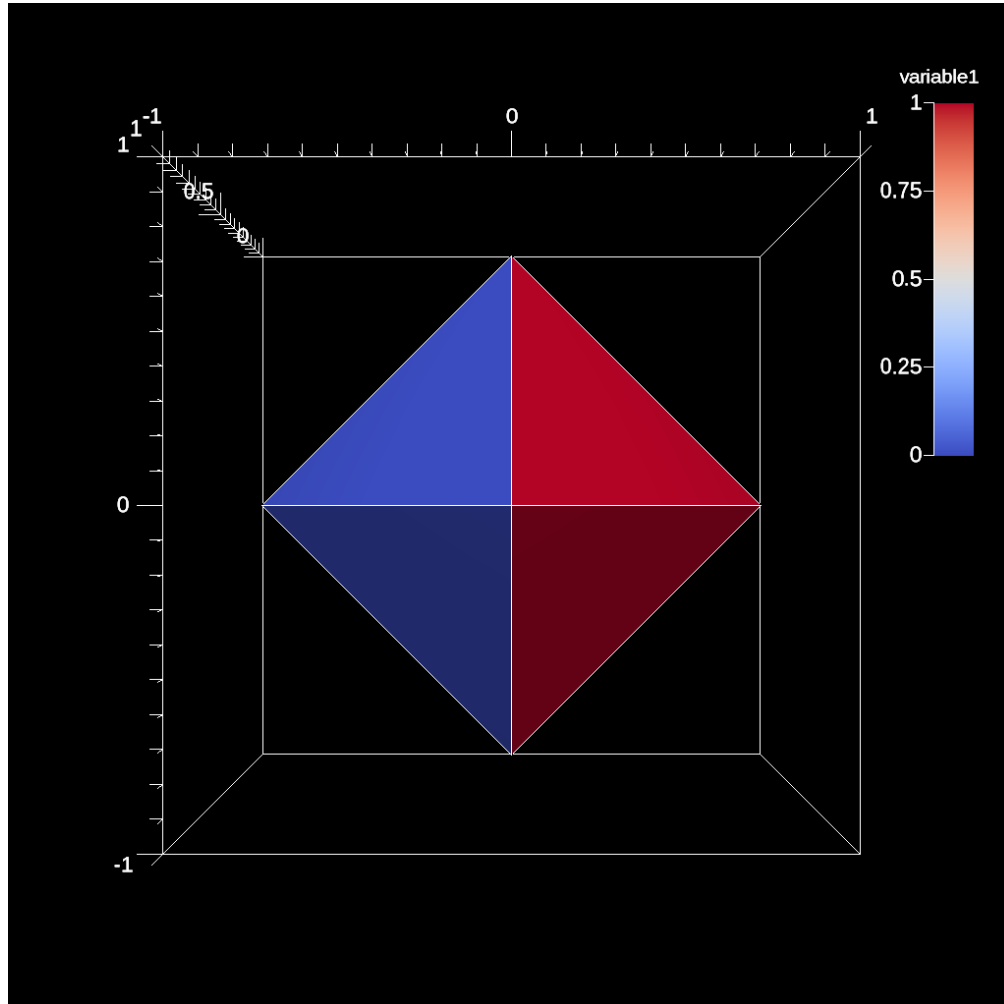
actions.append()["action"] = "execute";

// execute
a.execute(actions);
```

Didn't
change:
some
standard
methods
for telling
Ascent to
carry out
the scene

Code: ascent_scene_example1.cpp

No Pipeline, no Extracts, one Scene (but two plots in that Scene)



```
Node scenes;  
scenes["s1/plots/p1/type"] = "pseudocolor";  
scenes["s1/plots/p1/field"] = "variable1";  
scenes["s1/plots/p2/type"] = "mesh";  
scenes["s1/image_prefix"] = "render_two_plots";
```

Added a second plot, p2, with type mesh

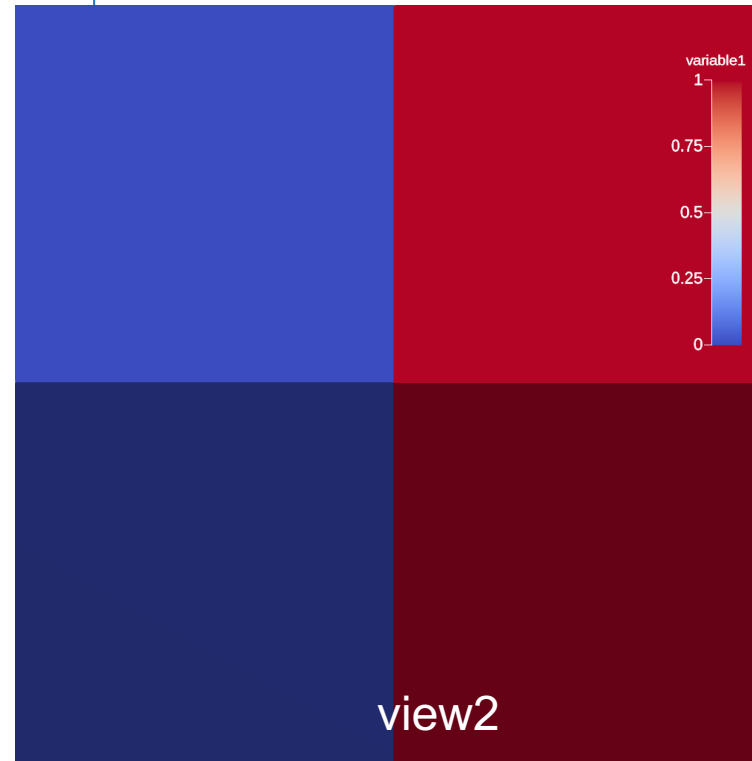
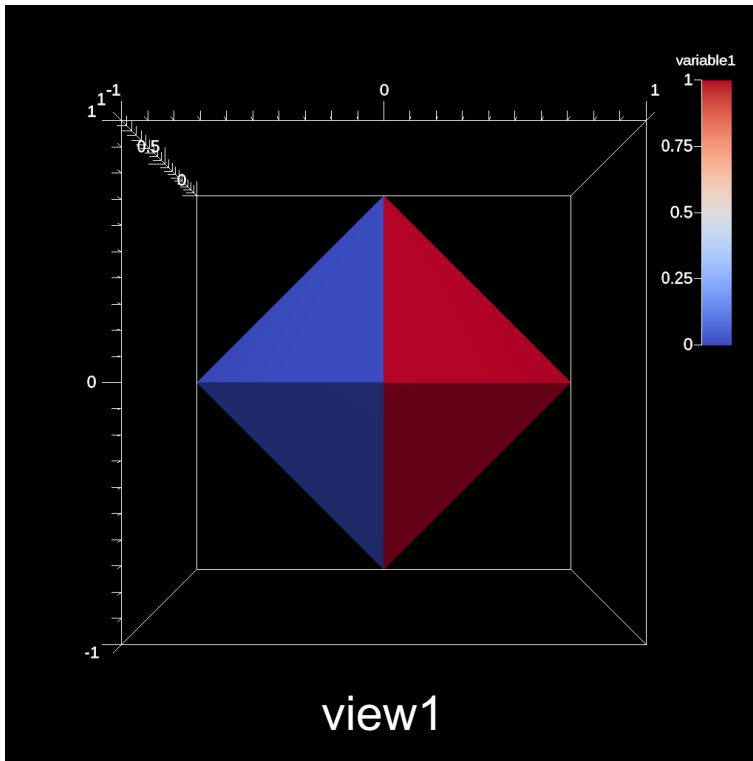
Code: ascent_scene_example2.cpp

Still no Pipeline or Extracts, one Scene but: set up multiple renders

Code: ascent_scene_example3.cpp

“renders” object controls rendering
multiple renders per scene OK
includes controls for camera and image size

```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "variable1";
//scenes["s1/image_prefix"] = "ascent_output";
scenes["s1/renderers/r1/image_name"] = "view1";
scenes["s1/renderers/r2/image_name"] = "view2";
scenes["s1/renderers/r2/camera/zoom"] = 3.0;
```



More on renders

```
conduit::l
scenes ["s
scenes ["s
scenes ["s
scenes ["s
scenes ["s
scenes ["s
scenes ["s
double vec
vec3[0] =
scenes ["s
vec3[0] =
scenes ["s
vec3[0] =
scenes ["s
scenes ["s
scenes ["s
scenes ["s
scenes ["s
scenes ["s
scenes ["s
scenes ["s
scenes ["s1/renders/r2/camera/far_plane"] = 33.1;
```

Additional Render Options

In addition to image and camera parameters, renders have several options that allow the users to control the appearance of images. Below is a list of additional parameters:

- `bg_color` : an array of three floating point values that controls the background color.
- `fg_color` : an array of three floating point values that controls the foreground color. Foreground colors indicate the color of annotations and mesh plot lines.
- `annotations` : controls if annotations are rendered or not. Valid values are `"true"` and `"false"`.
- `render_bg` : controls if the background is rendered or not. If no background is rendered, the background will appear transparent. Valid values are `"true"` and `"false"`.

```
"renders":
```

```
,0],
```

```
}
```

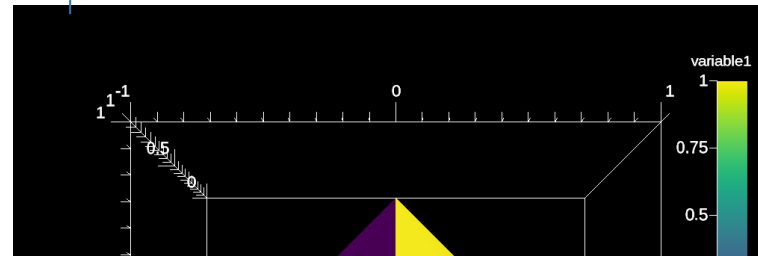
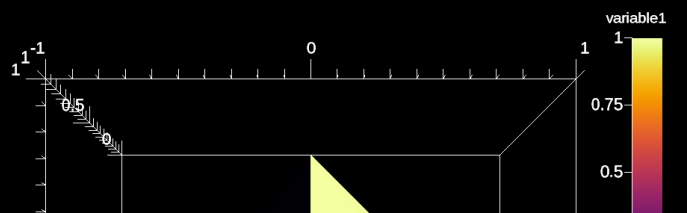
Still no Pipeline or Extracts, one Scene but: set color tables

Code: ascent_scene_example4.cpp

color_table/name being set
Another options: color_table/reverse

```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "variable1";
scenes["s1/plots/p1/color_table/name"] = "Viridis";
scenes["s1/image_prefix"] = "viridis";

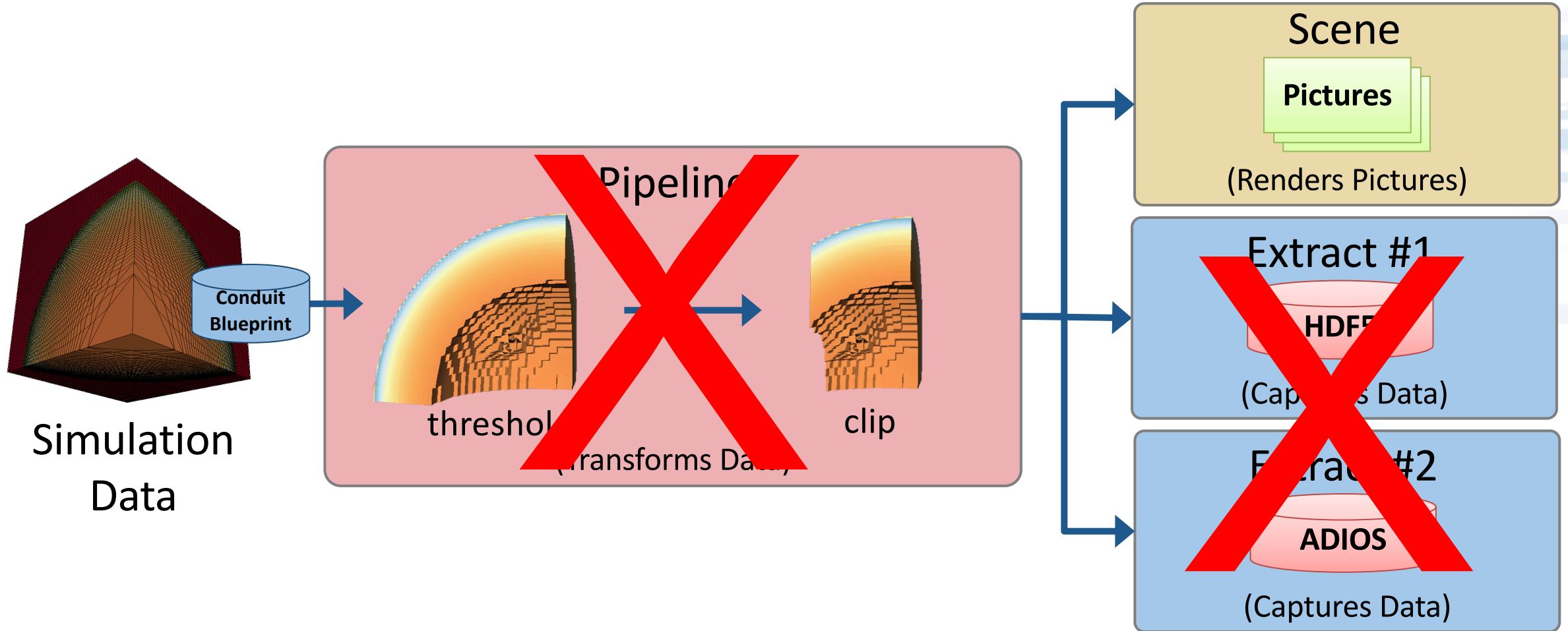
scenes["s2/plots/p1/type"] = "pseudocolor";
scenes["s2/plots/p1/field"] = "variable1";
scenes["s2/plots/p1/color_table/name"] = "Inferno";
scenes["s2/image_prefix"] = "inferno";
```



Here is an example of clamping the scalar values to the range [-0.5, 0.5].

```
conduit::Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "braid";
scenes["s1/plots/p1/min_value"] = -0.5;
scenes["s1/plots/p1/max_value"] = 0.5;
```

Now Let's Consider Examples With Pipelines



One Pipeline With One Filter, One Scene

```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/field"] = "braid";
// Set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "out_ascent_render_3d";

// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_scenes";
add_act["scenes"] = scenes;

actions.append()["action"] = "execute";

// execute
a.execute(actions);
```

Same as
previous
examples

Code: ascent_pipeline_example1.cpp

```
Node pipelines;
pipelines["p11/f1/type"] = "contour";
Node contour_params;
contour_params["field"] = "braid";
double iso_vals[2] = {0.2, 0.4};
contour_params["iso_values"].set_external(iso_vals,2);
pipelines["p11/f1/params"] = contour_params;

// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_pipelines";
add_act["pipelines"] = pipelines;
```

New code
for setting
up pipelines

```
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/pipeline"] = "p11";
scenes["s1/plots/p1/field"] = "braid";
// set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "isosurface";

Node &add_act2 = actions.append();
add_act2["action"] = "add_scenes";
add_act2["scenes"] = scenes;

actions.append()["action"] = "execute";
```

```
// execute
a.execute(actions);
```

One Pipeline With One Filter, One Scene

“contour” is a reserved word,
and it tells Ascent to use a
Contour filter

“p1/f1/type” → the type of the first filter (f1)
in the first pipeline (p1).
We could use any names besides p1 and f1.

Instantiate a new node, which will contain
parameters for the contour.

Set parameter values by setting Conduit keys
for specific keywords (“field”, “iso_values”)

Register the parameters with the pipeline

Tell the “actions” node to add our pipelines
(only one pipeline in this example)

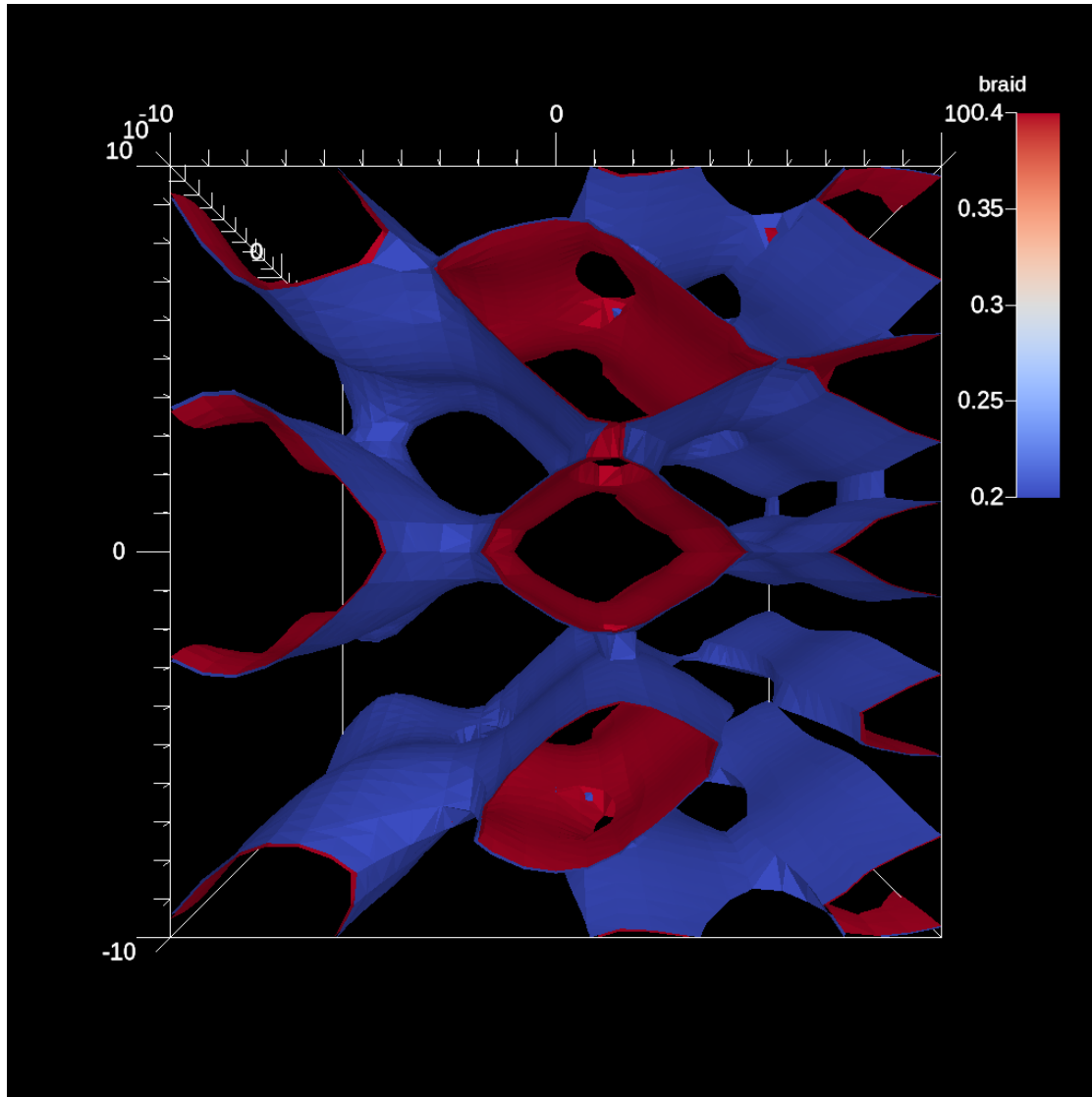
Adding the pipeline to the actions node allows
us to refer to it later

We get a second node to set up scenes
(do not reuse the first node)

```
Node pipelines;  
pipelines["p1/f1/type"] = "contour";  
Node contour_params;  
contour_params["field"] = "braid";  
double iso_vals[2] = {0.2, 0.4};  
contour_params["iso_values"].set_external(iso_vals,2);  
pipelines["p1/f1/params"] = contour_params;  
  
// setup actions  
Node actions;  
Node &add_act = actions.append();  
add_act["action"] = "add_pipelines";  
add_act["pipelines"] = pipelines;  
  
// declare a scene to render the dataset  
Node scenes;  
scenes["s1/plots/p1/type"] = "pseudocolor";  
scenes["s1/plots/p1/pipeline"] = "p1";  
scenes["s1/plots/p1/field"] = "braid";  
// set the output file name (ascent will add ".png")  
scenes["s1/image_prefix"] = "isosurface";  
  
Node &add_act2 = actions.append();  
add_act2["action"] = "add_scenes";  
add_act2["scenes"] = scenes;  
  
actions.append()["action"] = "execute";  
  
// execute  
a.execute(actions);
```

Code: ascent_pipeline_example1.cpp

One Pipeline With One Filter, One Scene: Output



```
Node pipelines;  
pipelines["p1/f1/type"] = "contour";  
Node contour_params;  
contour_params["field"] = "braid";  
double iso_vals[2] = {0.2, 0.4};  
contour_params["iso_values"].set_external(iso_vals,2);  
pipelines["p1/f1/params"] = contour_params;
```

```
// setup actions
```

```
Node actions;
```

```
Node &add_act = actions.append();
```

```
add_act["action"] = "add_pipelines";
```

```
add_act["pipelines"] = pipelines;
```

```
// declare a scene to render the dataset
```

```
Node scenes;
```

```
scenes["s1/plots/p1/type"] = "pseudocolor";
```

```
scenes["s1/plots/p1/pipeline"] = "p1";
```

```
scenes["s1/plots/p1/field"] = "braid";
```

```
// set the output file name (ascent will add ".png")
```

```
scenes["s1/image_prefix"] = "isosurface";
```

```
Node &add_act2 = actions.append();
```

```
add_act2["action"] = "add_scenes";
```

```
add_act2["scenes"] = scenes;
```

```
actions.append()["action"] = "execute";
```

```
// execute
```

```
a.execute(actions);
```

Code: ascent_pipeline_example1.cpp

One Pipeline With Two Filters, One Scene

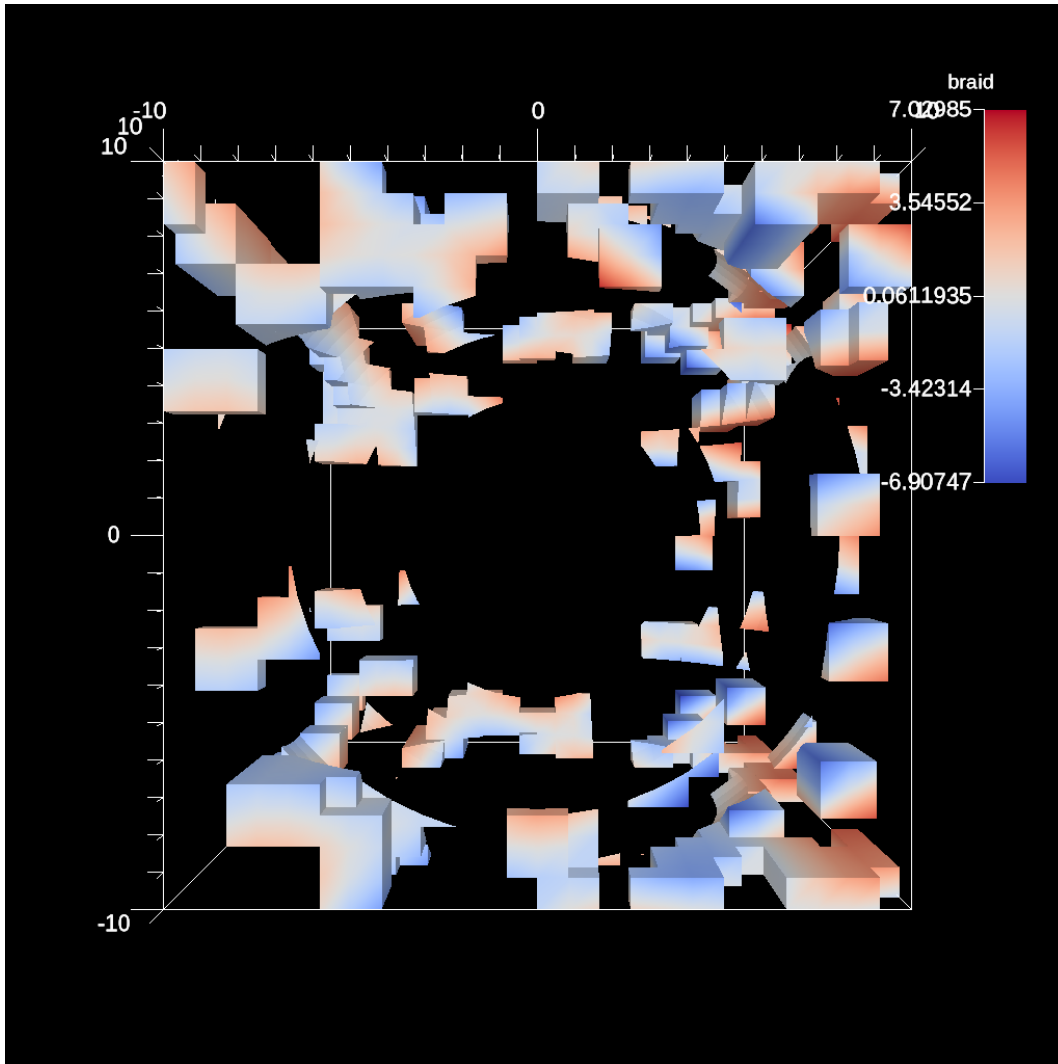
Set up first filter (Threshold).
Everything same as before

Set up second filter (Clip).
Important: Ascent knows there is a new filter,
since there is a new name ("f2", not "f1").
Also: Ascent knows it goes second because
we added it to the "pipelines" object second.

Everything same as before.
Adding a pipeline, and number of filters is
irrelevant.

```
pipelines["p11/f1/type"] = "threshold";  
// filter parameters  
conduit::Node thresh_params;  
thresh_params["field"] = "braid";  
thresh_params["min_value"] = 0.0;  
thresh_params["max_value"] = 0.5;  
pipelines["p11/f1/params"] = thresh_params;  
  
pipelines["p11/f2/type"] = "clip";  
// filter parameters  
conduit::Node clip_params;  
clip_params["topology"] = "mesh";  
clip_params["sphere/center/x"] = 0.0;  
clip_params["sphere/center/y"] = 0.0;  
clip_params["sphere/center/z"] = 0.0;  
clip_params["sphere/radius"] = 12;  
pipelines["p11/f2/params/"] = clip_params;  
  
// setup actions  
Node actions;  
Node &add_act = actions.append();  
add_act["action"] = "add_pipelines";  
add_act["pipelines"] = pipelines;
```

One Pipeline With Two Filters, One Scene: Output



```
pipelines["p11/f1/type"] = "threshold";  
// filter parameters  
conduit::Node thresh_params;  
thresh_params["field"] = "braid";  
thresh_params["min_value"] = 0.0;  
thresh_params["max_value"] = 0.5;  
pipelines["p11/f1/params"] = thresh_params;
```

```
pipelines["p11/f2/type"] = "clip";  
// filter parameters  
conduit::Node clip_params;  
clip_params["topology"] = "mesh";  
clip_params["sphere/center/x"] = 0.0;  
clip_params["sphere/center/y"] = 0.0;  
clip_params["sphere/center/z"] = 0.0;  
clip_params["sphere/radius"] = 12;  
pipelines["p11/f2/params/"] = clip_params;
```

```
// setup actions  
Node actions;  
Node &add_act = actions.append();  
add_act["action"] = "add_pipelines";  
add_act["pipelines"] = pipelines;
```


Two Pipelines, One Scene

Set up pipeline 1 (pl1), then pipeline 2 (pl2).

```
// pipeline 1
pipelines["pl1/f1/type"] = "contour";
conduit::Node contour_params;
contour_params["field"] = "braid";
double iso_vals[2] = {0.2, 0.4};
contour_params["iso_values"].set_external(iso_vals,2);
pipelines["pl1/f1/params"] = contour_params;

// pipeline 2
conduit::Node thresh_params;
pipelines["pl2/f1/type"] = "threshold";
thresh_params["field"] = "braid";
thresh_params["min_value"] = 0.0;
thresh_params["max_value"] = 0.5;
pipelines["pl2/f1/params"] = thresh_params;
pipelines["pl2/f2/type"] = "clip";
conduit::Node clip_params;
clip_params["topology"] = "mesh";
clip_params["sphere/center/x"] = 0.0;
clip_params["sphere/center/y"] = 0.0;
clip_params["sphere/center/z"] = 0.0;
clip_params["sphere/radius"] = 12;
pipelines["pl2/f2/params/"] = clip_params;
```

```
// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_pipelines";
add_act["pipelines"] = pipelines;

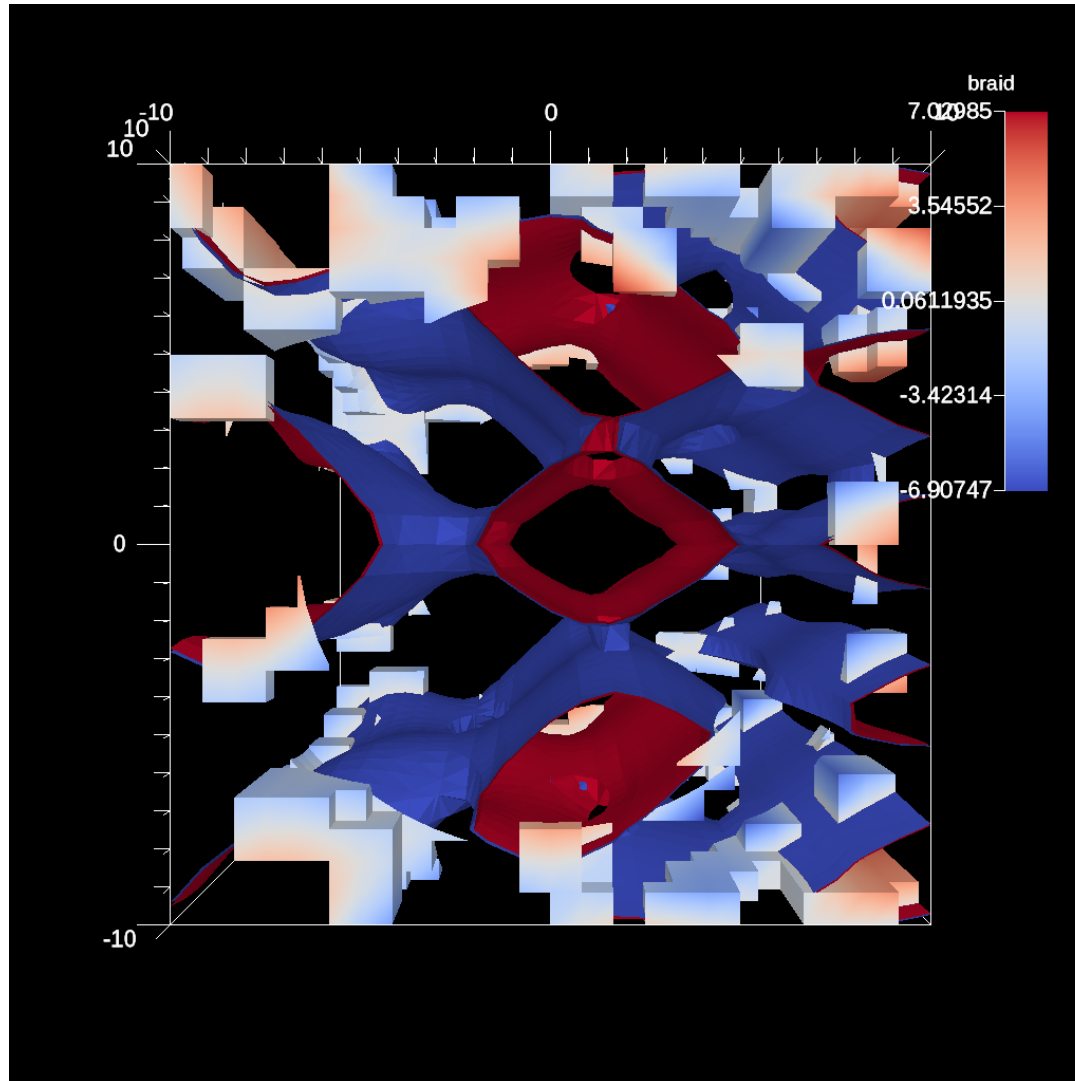
// declare a scene to render the dataset
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/pipeline"] = "pl1";
scenes["s1/plots/p1/field"] = "braid";
scenes["s1/plots/p2/type"] = "pseudocolor";
scenes["s1/plots/p2/pipeline"] = "pl2";
scenes["s1/plots/p2/field"] = "braid";
// set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "two_pipelines";

Node &add_act2 = actions.append();
add_act2["action"] = "add_scenes";
add_act2["scenes"] = scenes;

actions.append()["action"] = "execute";
```

Set up plot for each pipeline.

Two Pipelines, One Scene: Output



```
// pipeline 1
pipelines["p11/f1/type"] = "contour";
conduit::Node contour_params;
contour_params["field"] = "braid";
double iso_vals[2] = {0.2, 0.4};
contour_params["iso_values"].set_external(iso_vals,2);
pipelines["p11/f1/params"] = contour_params;
```

```
// pipeline 2
conduit::Node thresh_params;
pipelines["p12/f1/type"] = "threshold";
thresh_params["field"] = "braid";
thresh_params["min_value"] = 0.0;
thresh_params["max_value"] = 0.5;
pipelines["p12/f1/params"] = thresh_params;
pipelines["p12/f2/type"] = "clip";
conduit::Node clip_params;
clip_params["topology"] = "mesh";
clip_params["sphere/center/x"] = 0.0;
clip_params["sphere/center/y"] = 0.0;
clip_params["sphere/center/z"] = 0.0;
clip_params["sphere/radius"] = 12;
pipelines["p12/f2/params/"] = clip_params;
```

```
// setup actions
```

```
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_pipelines";
add_act["pipelines"] = pipelines;
```

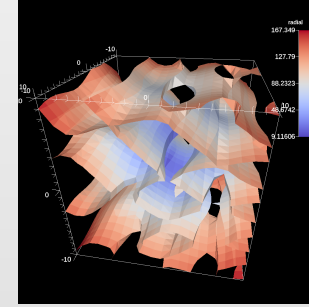
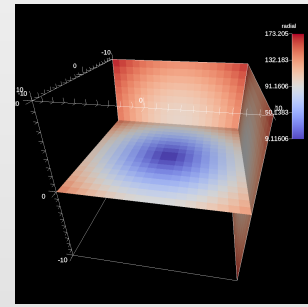
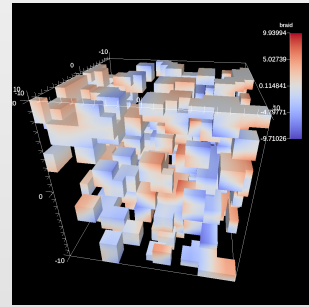
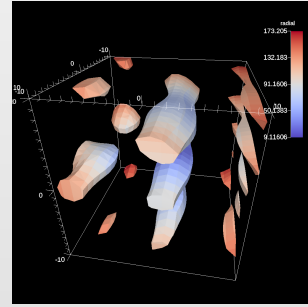
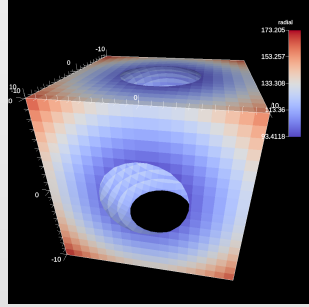
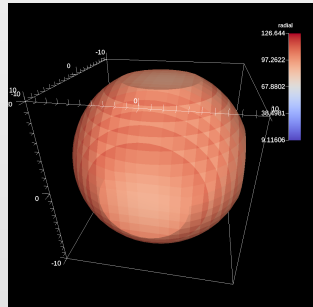
```
// declare a scene to render the dataset
```

```
Node scenes;
scenes["s1/plots/p1/type"] = "pseudocolor";
scenes["s1/plots/p1/pipeline"] = "p11";
scenes["s1/plots/p1/field"] = "braid";
scenes["s1/plots/p2/type"] = "pseudocolor";
scenes["s1/plots/p2/pipeline"] = "p12";
scenes["s1/plots/p2/field"] = "braid";
// set the output file name (ascent will add ".png")
scenes["s1/image_prefix"] = "two_pipelines";
```

```
Node &add_act2 = actions.append();
add_act2["action"] = "add_scenes";
add_act2["scenes"] = scenes;
```

```
actions.append()["action"] = "execute";
```

Supported Filters & Scenes

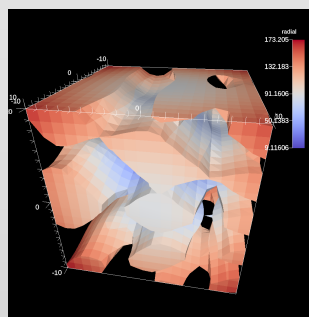
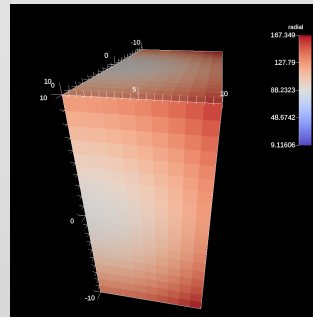


Iso-Volume

Threshold

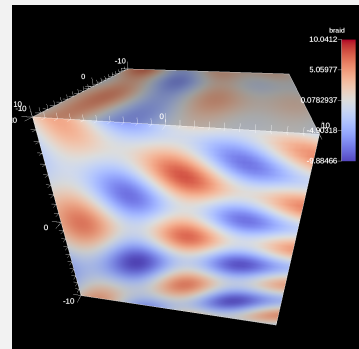
Slice

Contour

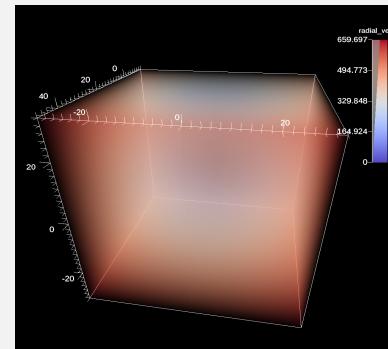


Clips

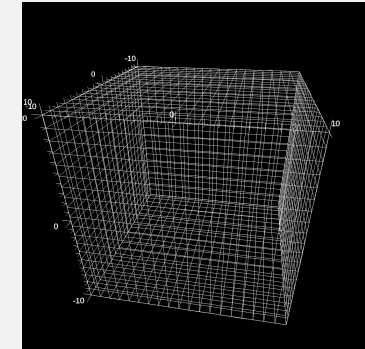
Rendering



Pseudocolor



Volume



Mesh

More information on Filters and Pipelines: <https://ascent.readthedocs.io/en/latest/Actions/Pipelines.html>

IsoVolume

IsoVolume is a filter that clips a data set based on a minimum and maximum value in a scalar field. All value outside of the minminum and maximum values are removed from the data set.

```
conduit::Node pipelines;  
// pipeline 1  
pipelines["p1/f1/type"] = "iso_volume";  
// filter knobs  
conduit::Node &clip_params = pipelines["p1/f1/params"];  
clip_params["field"] = "braid";  
clip_params["min_value"] = 5.;  
clip_params["max_value"] = 10.;
```

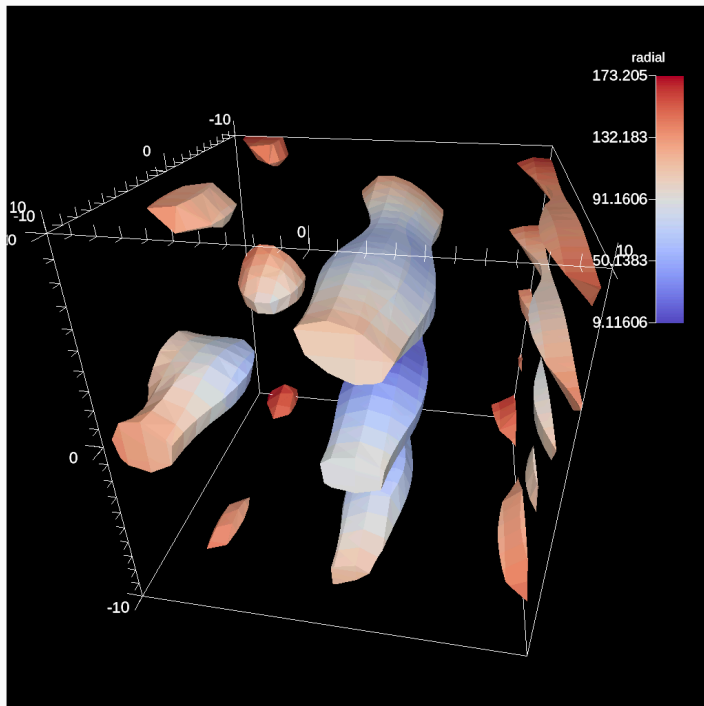


Fig. 20 An example of creating a iso-volume of values between 5.0 and 10.0.

Slice

The slice filter extracts a 2d plane from a 3d data set. The plane is defined by a point (on the plane) and a normal vector (not required to be nomalized).

```
conduit::Node pipelines;  
pipelines["p1/f1/type"] = "slice";  
// filter knobs  
conduit::Node &slice_params = pipelines["p1/f1/params"];  
slice_params["point/x"] = 0.f;  
slice_params["point/y"] = 0.f;  
slice_params["point/z"] = 0.f;  
  
slice_params["normal/x"] = 0.f;  
slice_params["normal/y"] = 0.f;  
slice_params["normal/z"] = 1.f;
```

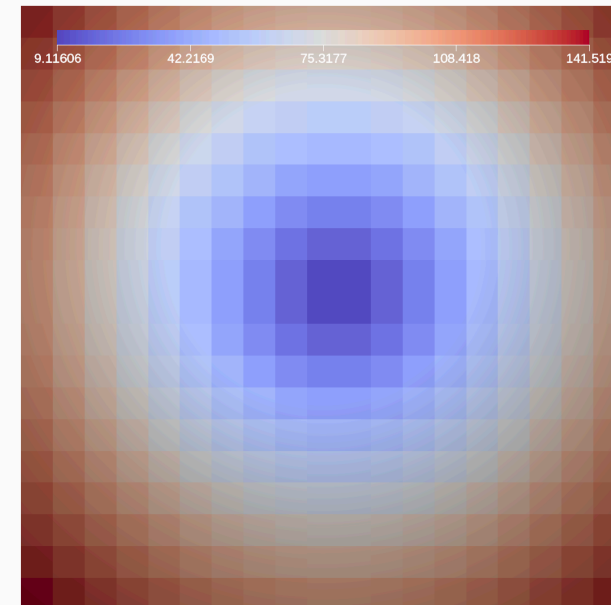
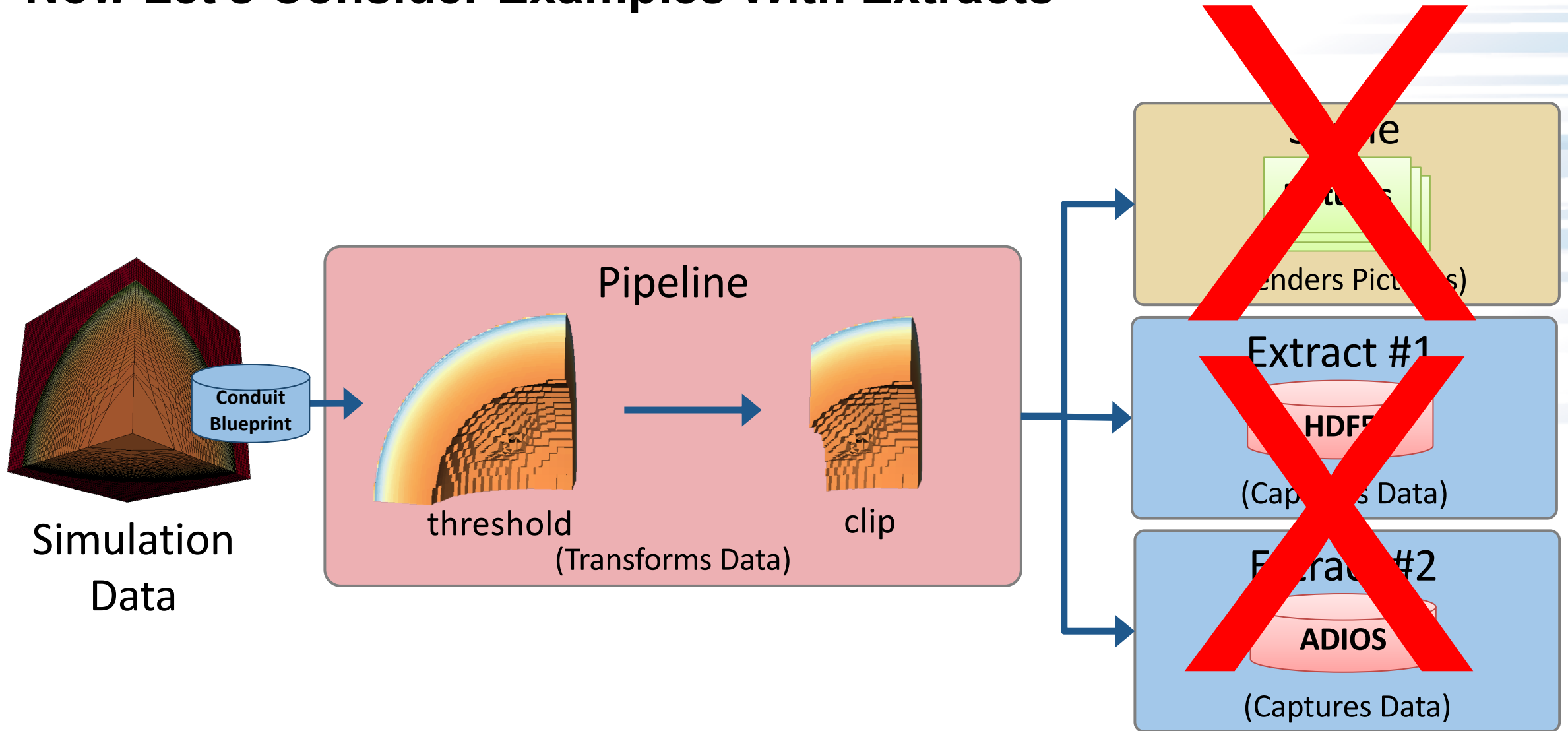


Fig. 11 An example image of the slice filter on a element-centered variable using the previous code sample.

Figure 11 shows an image produced from the slice filter. The full example is located in the file `slice test`.

Now Let's Consider Examples With Extracts



Extracts

- Extracts: an abstraction for capturing data.
 - “data capture” → sends data outside the Ascent infrastructure
- Currently supported extracts include:
 - Relay: leverages Conduit’s Relay library to do parallel I/O
 - Python: use a python script with NumPy to analyze mesh data
 - ADIOS: use ADIOS to send data to a separate resource
 - Cinema: output a Cinema database

No Pipeline, No Scene, One Extract

Specifies the extract should use the relay output

Set a parameter so the output is named "braid."

```
Node extracts;  
extracts["e1/type"] = "relay";  
extracts["e1/params/path"] = "braid";  
extracts["e1/params/protocol"] = "blueprint/mesh/hdf5";
```

Set a parameter so the output is HDF5 (not JSON)

```
// setup actions  
Node actions;  
Node &add_act = actions.append();  
add_act["action"] = "add_extracts";  
add_act["extracts"] = extracts;  
  
actions.append()["action"] = "execute";
```

New action: "add_extracts"

Outputs:

File: braid.cycle_0000100.root
Directory: braid.cycle_000100
File: domain_000000.hdf5 (in directory)

One Pipeline, No Scene, One Extract

```
Node pipelines;
pipelines["pl1/f1/type"] = "contour";
Node contour_params;
contour_params["field"] = "braid";
double iso_vals[2] = {0.2, 0.4};
contour_params["iso_values"].set_external(iso_vals,2);
pipelines["pl1/f1/params"] = contour_params;

// setup actions
Node actions;
Node &add_act = actions.append();
add_act["action"] = "add_pipelines";
add_act["pipelines"] = pipelines;

Node extracts;
extracts["e1/type"] = "relay";
extracts["e1/pipeline"] = "pl1";
extracts["e1/params/path"] = "braid_contour";
extracts["e1/params/protocol"] = "blueprint/mesh/hdf5";

// setup actions
Node &add_act2 = actions.append();
add_act2["action"] = "add_extracts";
add_act2["extracts"] = extracts;

actions.append()["action"] = "execute";

// execute
a.execute(actions);
```

Typical code for setting up a pipeline

Add a pipeline action (as per usual)

Set up extract

New step: declare the extract should capture the output of the pipeline "pl1."

Add a second action, for the extract

Outputs:
Same HDF5 file structure as previous slide, but the output is the isosurface, not the original data set.

Python Extract

- Python extracts can execute arbitrary Python code.
- The Python code uses Conduit's python interface to interrogate and retrieve mesh data.
- Code is executed on each MPI rank, and mpi4py can be used for collective communication.

```
conduit::Node extracts;  
extracts["e1/type"] = "python";  
extracts["e1/params/source"] = py_script;
```

```
import numpy as np  
from mpi4py import MPI  
  
# obtain a mpi4py mpi comm object  
comm = MPI.Comm.f2py(ascent_mpi_comm_id())  
  
# get this MPI task's published blueprint data  
mesh_data = ascent_data().child(0)  
  
# fetch the numpy array for the energy field values  
e_vals = mesh_data["fields/energy/values"]  
  
# find the data extents of the energy field using mpi  
  
# first get local extents  
e_min, e_max = e_vals.min(), e_vals.max()  
  
# declare vars for reduce results  
e_min_all = np.zeros(1)  
e_max_all = np.zeros(1)  
  
# reduce to get global extents  
comm.Allreduce(e_min, e_min_all, op=MPI.MIN)  
comm.Allreduce(e_max, e_max_all, op=MPI.MAX)  
  
# compute bins on global extents  
bins = np.linspace(e_min_all, e_max_all)  
  
# get histogram counts for local data  
hist, bin_edges = np.histogram(e_vals, bins = bins)  
  
# declare var for reduce results  
hist_all = np.zeros_like(hist)  
  
# sum histogram counts with MPI to get final histogram  
comm.Allreduce(hist, hist_all, op=MPI.SUM)
```

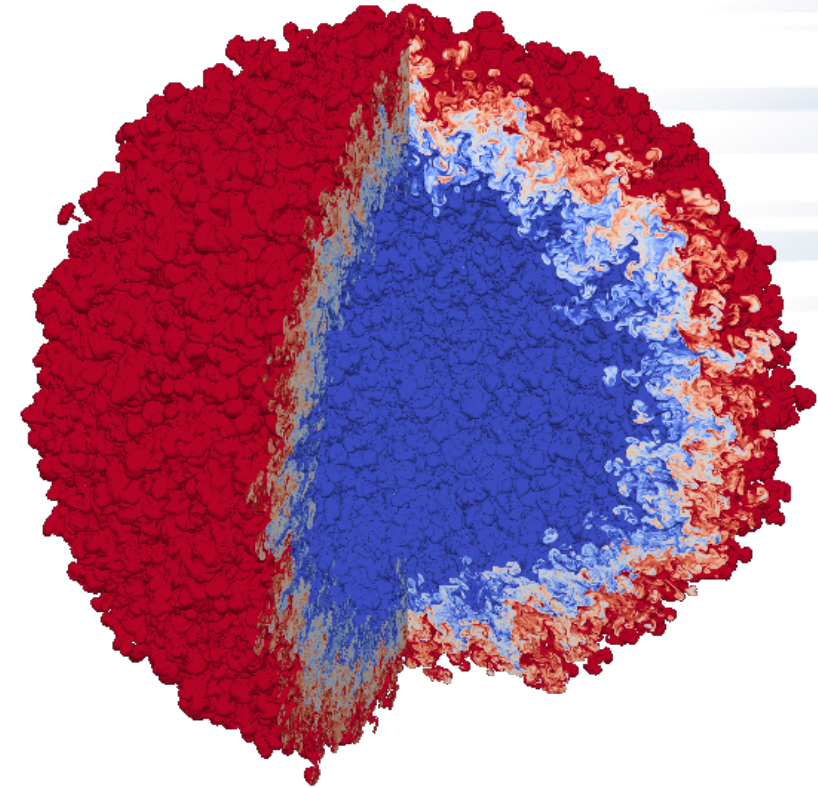
This tutorial

- 2 hours: Ascent
 - Overview
 - How to use? (get hands dirty / walk out with understanding on how to integrate)
 - **Examples of advanced usage (what it can do)**
- 1 hour: other ECP vis technologies
 - Cinema
 - VTK-m
 - In situ algorithms

LLNL ran a massive turbulent fluid mixing simulation on Sierra in October 2018 over 16,000 GPUs

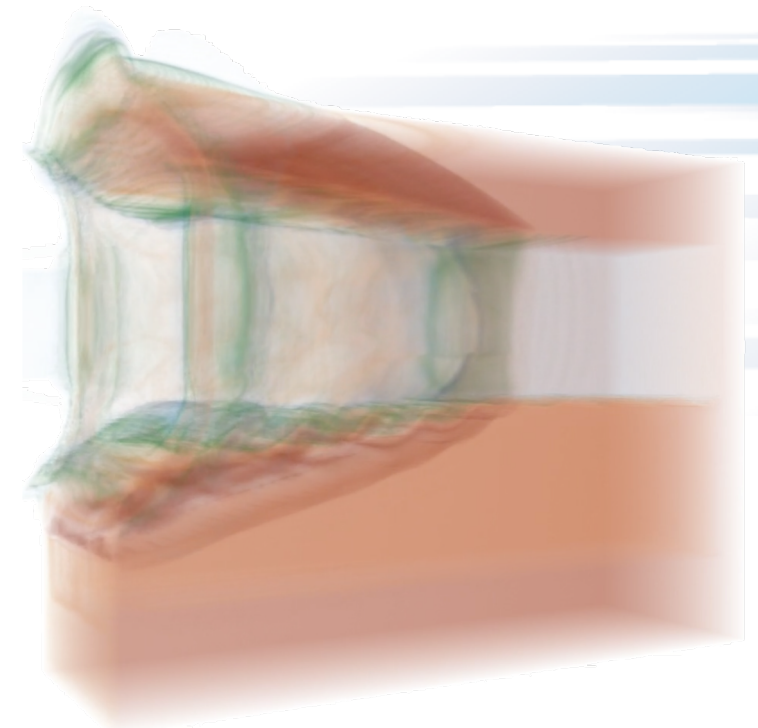
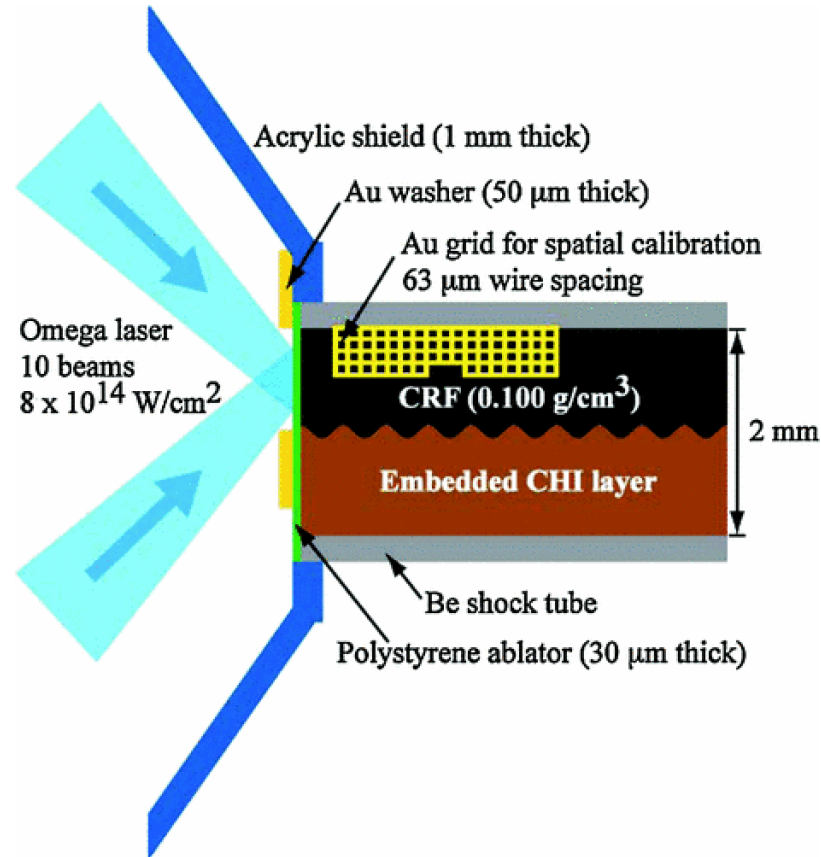
Highlights:

- The 97.8 billion element simulation ran across 16,384 GPUs on 4,096 Sierra Compute Nodes
- The simulation application used **CUDA** via **RAJA** to run on the GPUs
- Time-varying evolution of the mixing was visualized in-situ using **Ascent**, also leveraging 16,384 GPUs
- Ascent leveraged **VTK-m** to run visualization algorithms on the GPUs
- The last time step was exported to the parallel file system for detailed post-hoc visualization using **VisIt**



In-situ Visualization of
Mixing Layer

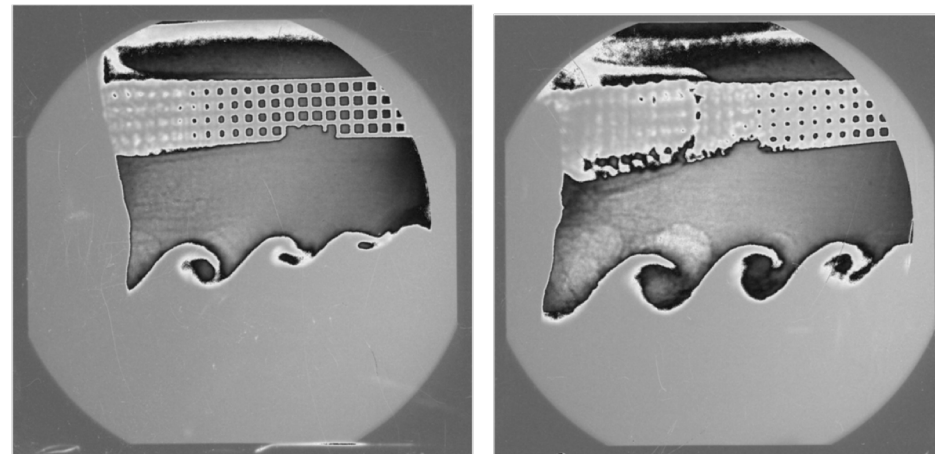
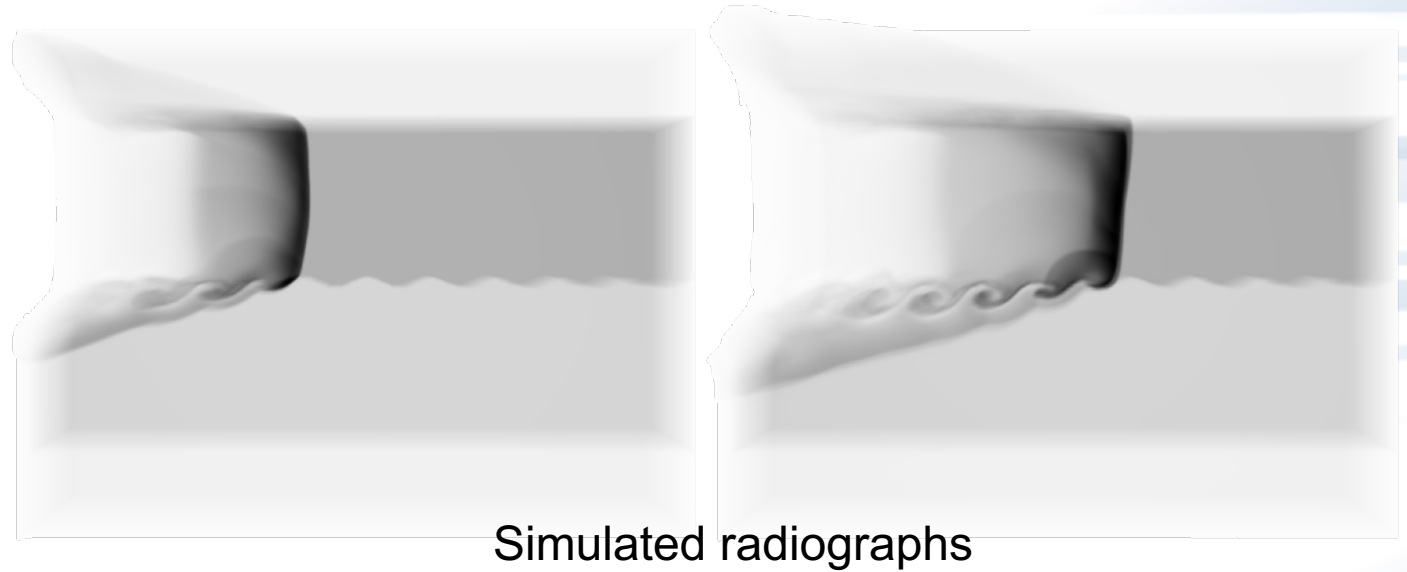
MARBL integration: blast-wave driven Kelvin-Helmholtz (big laser, tiny box simulation)



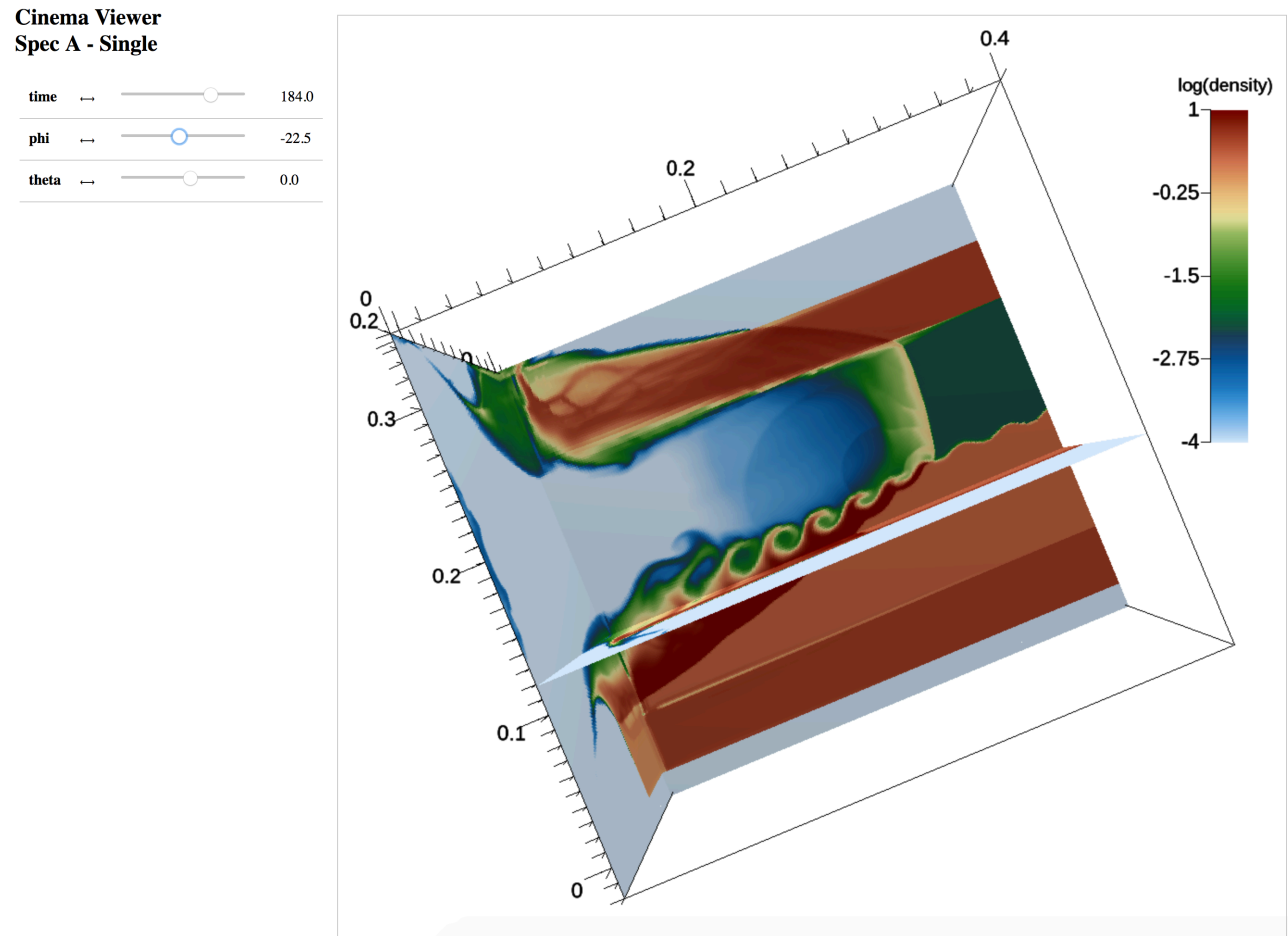
*Hurricane, O. A., et al. "Blast-wave driven Kelvin-Helmholtz shear layers in a laser driven high-energy-density plasma." *Astrophysics and Space Science* 336.1 (2011): 139-143.

ROVER is deployed in MARBL through Ascent

- In-situ radiography of a laser driven Kelvin-Helmholtz instability
- *2.3K MPI tasks*
- *120 hours wall time*
- *3.5M 3D Q2 elements, 100M quad points*
- *~20K RK2 timesteps*



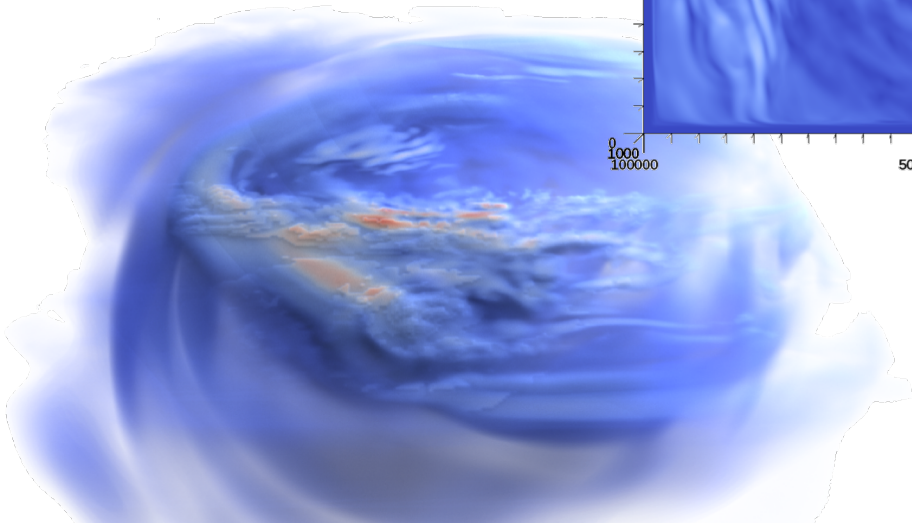
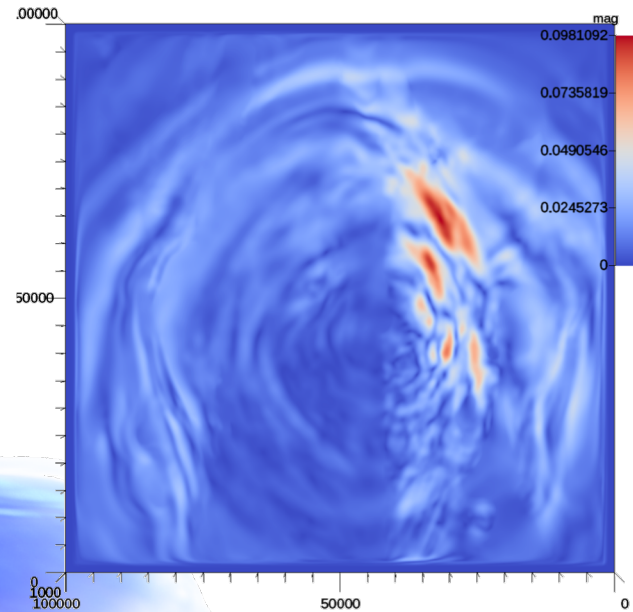
Data reduction: cinema database support



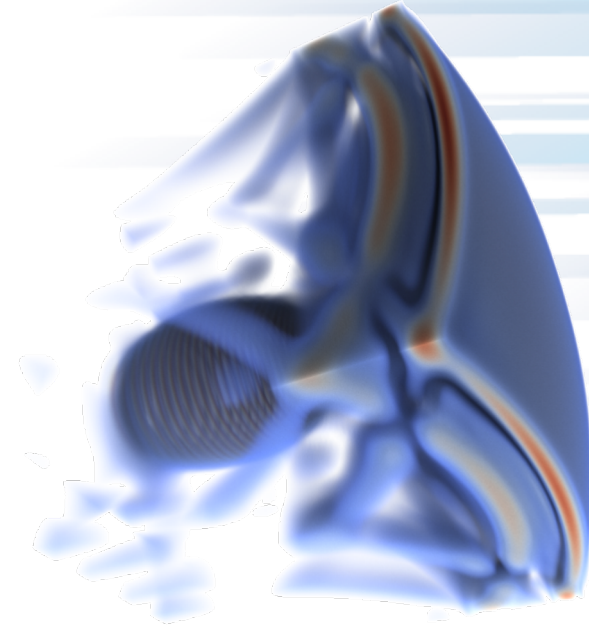
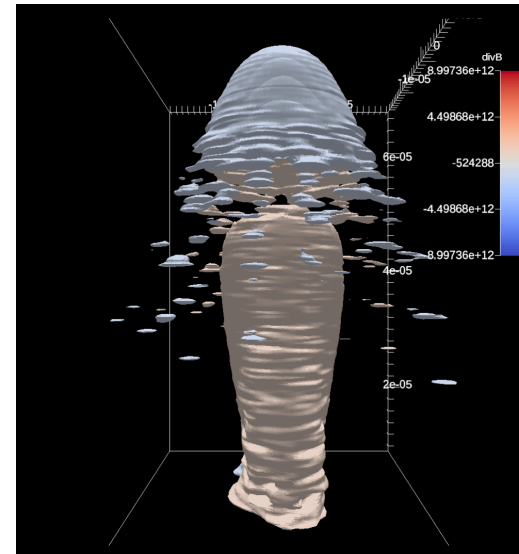
- http://portal.nersc.gov/project/visit/larsen/cinema/rad_kh/cinema.html
- <https://bit.ly/2VUOyYE>

Additional ECP connections

- SW4: seismology

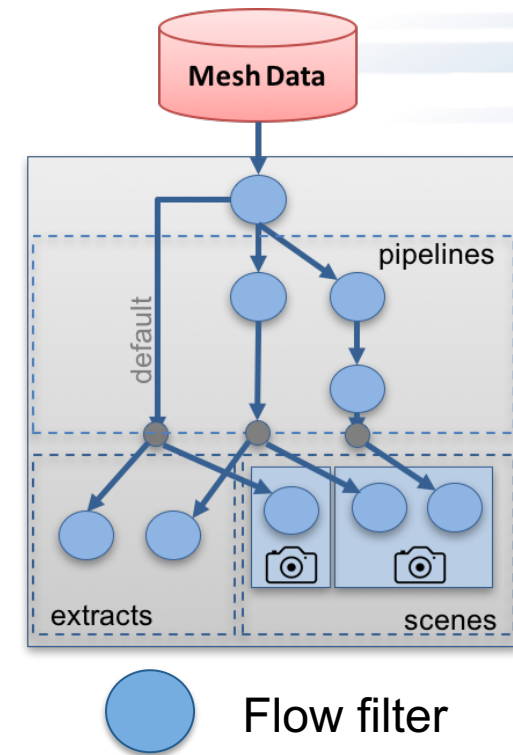


- WarpX: electromagnetic PIC code



Under development: Triggers

- Performing visualization every cycle takes time and resources away from the simulation
- We plan to add support for “Triggers”:
 - When X happens do Y
- Examples:
 - Entropy in energy reaches some threshold
 - Save data or render
 - Not enough node memory
 - Examine data flow network and make adjustments
 - Resample data to fit within constraints

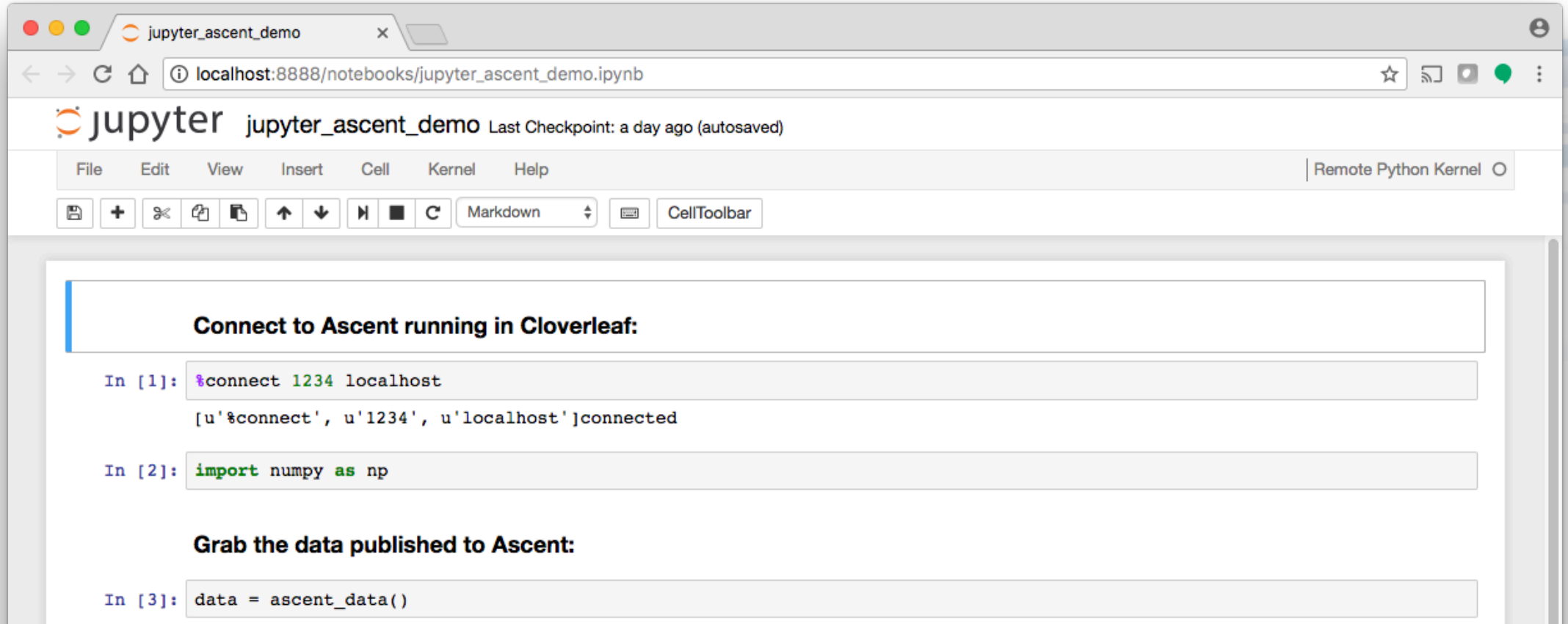


Under development: Jupyter Notebook Support



Ascent's Jupyter support will allow you to connect to a running simulation, access published data, run scripts, and yield back to the simulation.

Jupyter Notebook Demonstration



The screenshot shows a Jupyter Notebook interface in a web browser. The browser tab is titled "jupyter_ascent_demo" and the address bar shows "localhost:8888/notebooks/jupyter_ascent_demo.ipynb". The Jupyter logo and "jupyter_ascent_demo" are visible at the top, along with the text "Last Checkpoint: a day ago (autosaved)". The menu bar includes "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". A "Remote Python Kernel" button is on the right. The toolbar contains icons for file operations, navigation, and execution, with a "Markdown" dropdown and a "CellToolbar" button. The notebook content is as follows:

```
Connect to Ascent running in Cloverleaf:  
  
In [1]: %connect 1234 localhost  
[u'%connect', u'1234', u'localhost']connected  
  
In [2]: import numpy as np  
  
Grab the data published to Ascent:  
  
In [3]: data = ascent_data()
```

Jupyter Notebook Demonstration

data is a Conduit tree with Cloverleaf's mesh data, lets take a look at the state:

```
In [4]: print(data['state'])
```

```
{  
  "time": 0.313751481239519,  
  "domain_id": 0,  
  "cycle": 10  
}
```

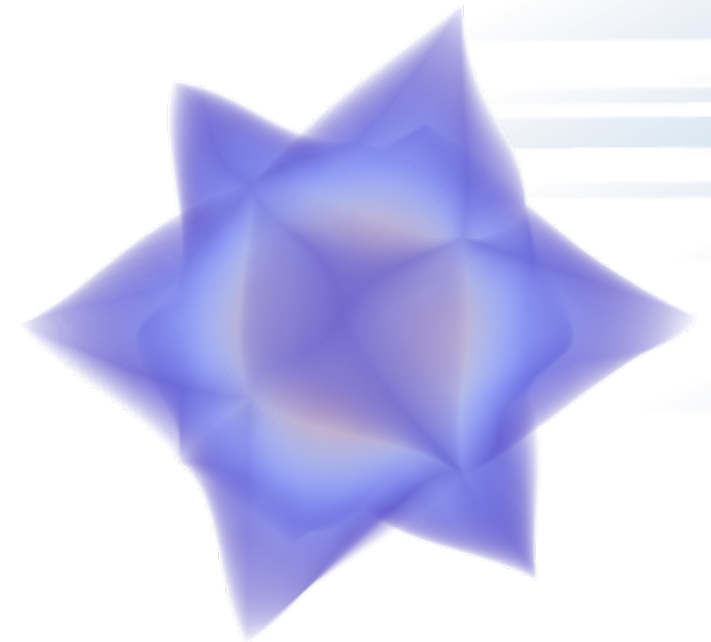
Next we list the names of the mesh fields:

```
In [5]: for f in data["fields"]:  
        print f.name()
```

```
density  
energy  
pressure  
velocity_x  
velocity_y  
velocity_z
```

Under development: Devil Ray to support native MFEM rendering

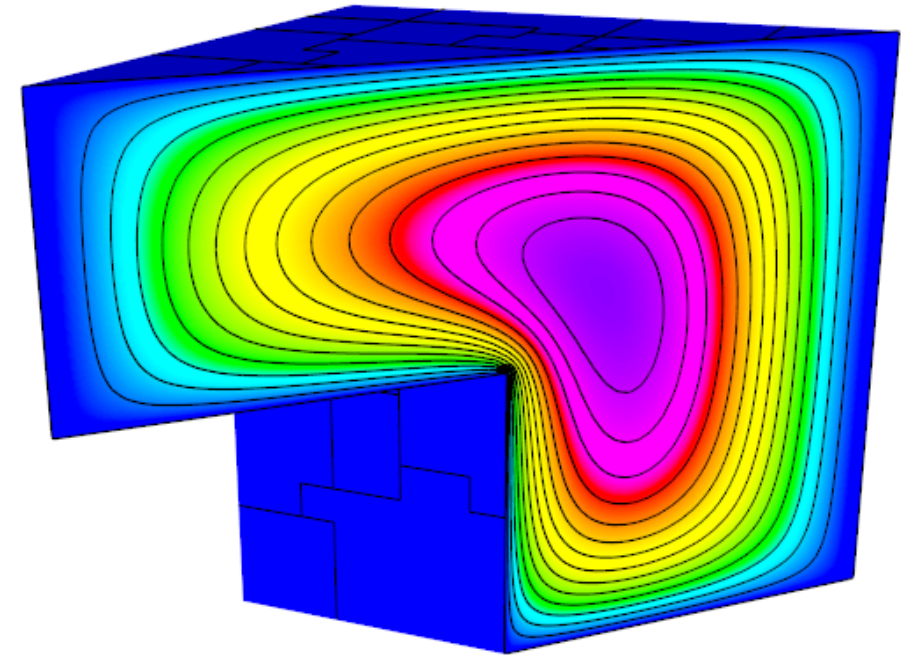
- Devil Ray is a library for direct ray tracing and volume rendering of high-order MFEM meshes
- Many-core capable, built using:
 - MFEM (<http://mfem.org/>)
 - RAJA (<https://github.com/LLNL/RAJA>)
 - Umpire (<https://github.com/LLNL/Umpire>)
- Devil Ray will be integrated as a component of Ascent



Example render of a high-order mesh using Devil Ray

Why do we need direct MFEM support?

- Traditional visualization refines high-order meshes into meshes with linear elements
- Increases memory usage
 - Might not have the memory in-situ
- Low-order refine can miss important features
 - Ex, high-order contours



Publications

- [A Flexible System For In Situ Triggers](#) Presented at the [ISAV 2018 Workshop](#), held in conjunction with SC 18, on November 12th 2018 in Dallas, TX, USA.
- [The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman](#) Presented at the [ISAV 2017 Workshop](#), held in conjunction with SC 17, on November 12th 2017 in Denver, CO, USA.
- [Performance Impacts of In Situ Wavelet Compression on Scientific Simulations](#) Presented at the [ISAV 2017 Workshop](#), held in conjunction with SC 17, on November 12th 2017 in Denver, CO, USA.
- [Projecting Performance Data Over Simulation Geometry Using SOSflow and Alpine](#) Presented at the [VPA 17 Workshop](#), held in conjunction with SC 17, on November 17th 2017 in Denver, CO, USA.
- [PaViz: A Power-Adaptive Framework for Optimal Power and Performance of Scientific Visualization Algorithms](#) Presented at the [EGPGV 2017 Symposium](#), held in conjunction with EuroVis 2017, on 12 June, Barcelona, Spain
- [Performance Modeling of In Situ Rendering](#) Presented at [SC16](#).
- [Optimizing Multi-Image Sort-Last Parallel Rendering](#) Presented at the [IEEE Symposium on Large Data and Analysis \(LDAV\) 2016](#), held in conjunction with IEEE Vis, on October 23rd 2016 in Baltimore, MD, USA.
- [Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes](#) Presented at the [ISAV 2015 Workshop](#), held in conjunction with SC 15, on November 16th 2015 in Austin, TX, USA.

Proof-of-concept: In-situ machine learning

- Python has a massive menu of data science tools
- Goal: Use Ascent to connect Python data science tools to HPC simulations
- Demonstrate ease of use:
 - Ascent provides curated simulation data that is easy to digest in python
 - Conduit Blueprint data published in Fortran, C, or C++ codes can be accessed as numpy arrays

What does using Python in Ascent look like?

```
import numpy as np
from mpi4py import MPI

# obtain a mpi4py mpi comm object
comm = MPI.Comm.f2py(ascent_mpi_comm_id())

# get this MPI task's published blueprint data
mesh_data = ascent_data()

# fetch the numpy array for the energy field values
e_vals = mesh_data["fields/energy/values"]

# find the data extents of the energy field using mpi

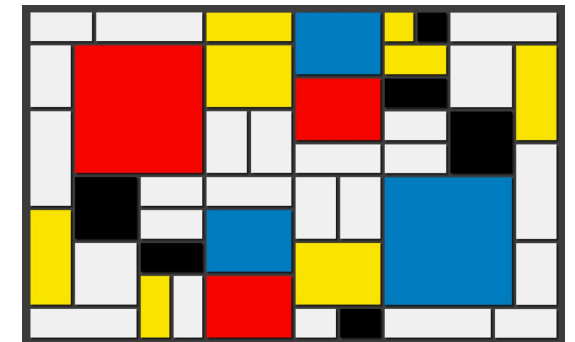
# first get local extents
e_min, e_max = e_vals.min(), e_vals.max()

# declare vars for reduce results
e_min_all = np.zeros(1)
e_max_all = np.zeros(1)

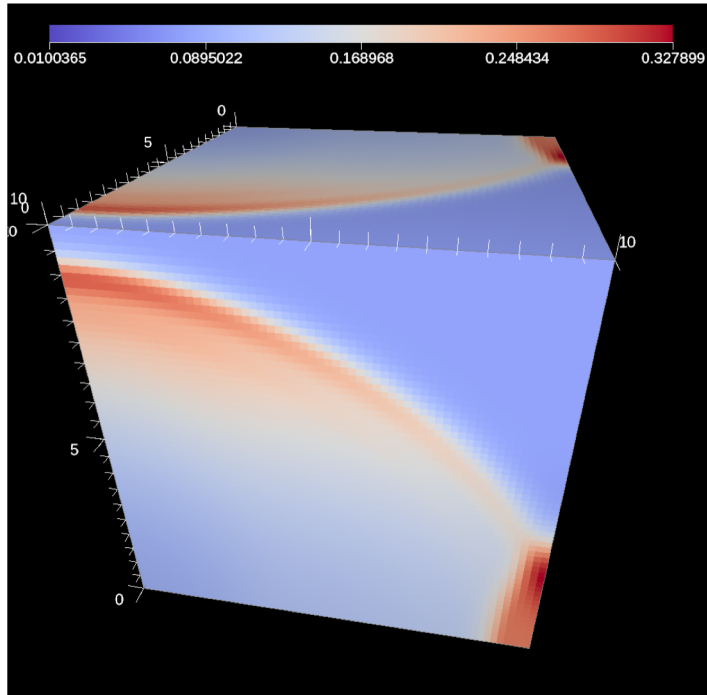
# reduce to get global extents
comm.Allreduce(e_min, e_min_all, op=MPI.MIN)
comm.Allreduce(e_max, e_max_all, op=MPI.MAX)
```


Custom Python Example: Distributed machine learning

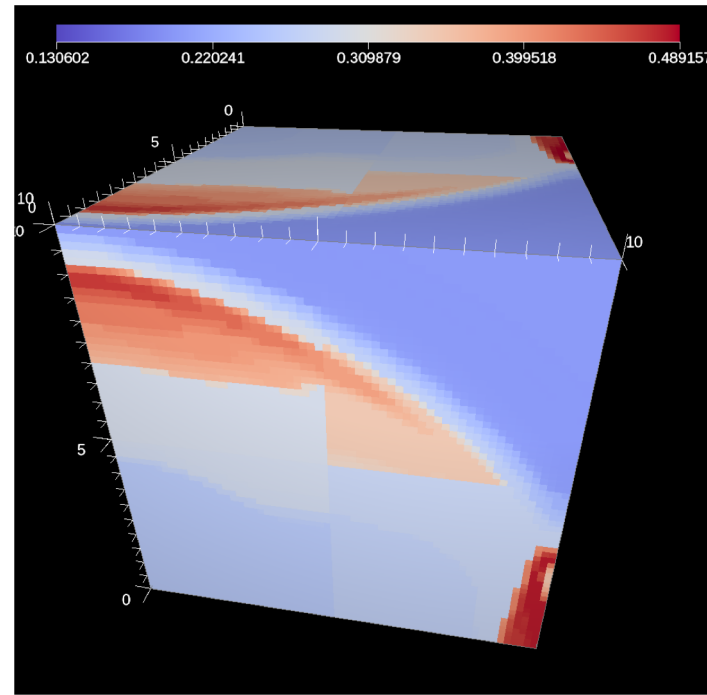
- Harvey Mudd Clinic
 - 2 semester senior project
 - 4 team members
- Investigate distributed machine learning
 - Naïve bayes
 - Random forest
 - Mondrian forest
- Demonstrate proof-of-concept in-situ
 - Ascent + Cloverleaf3D + python extract



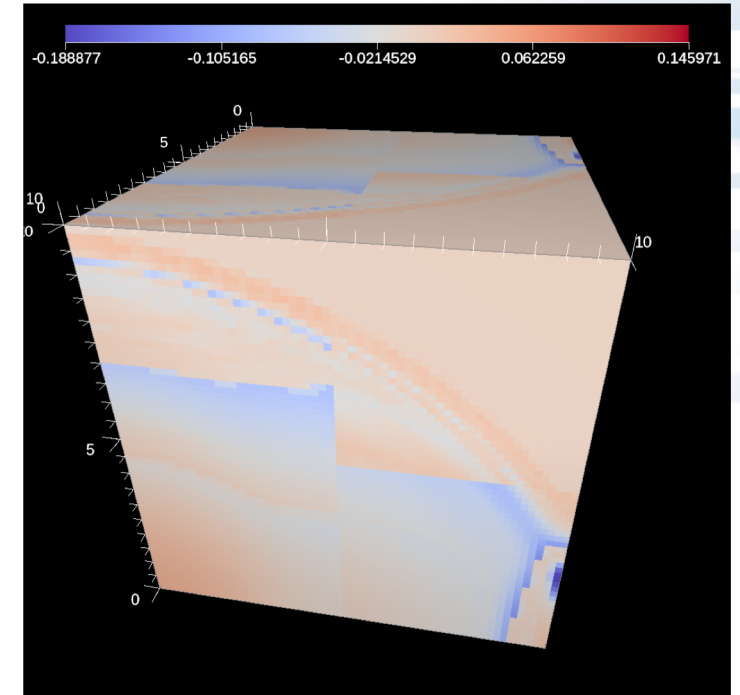
Proof-of-concept: In-situ distributed machine learning results



Actual Pressure



Predicted Pressure

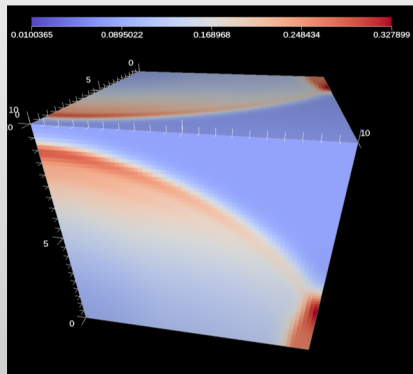


Difference

Inception: Using Ascent from a python extract

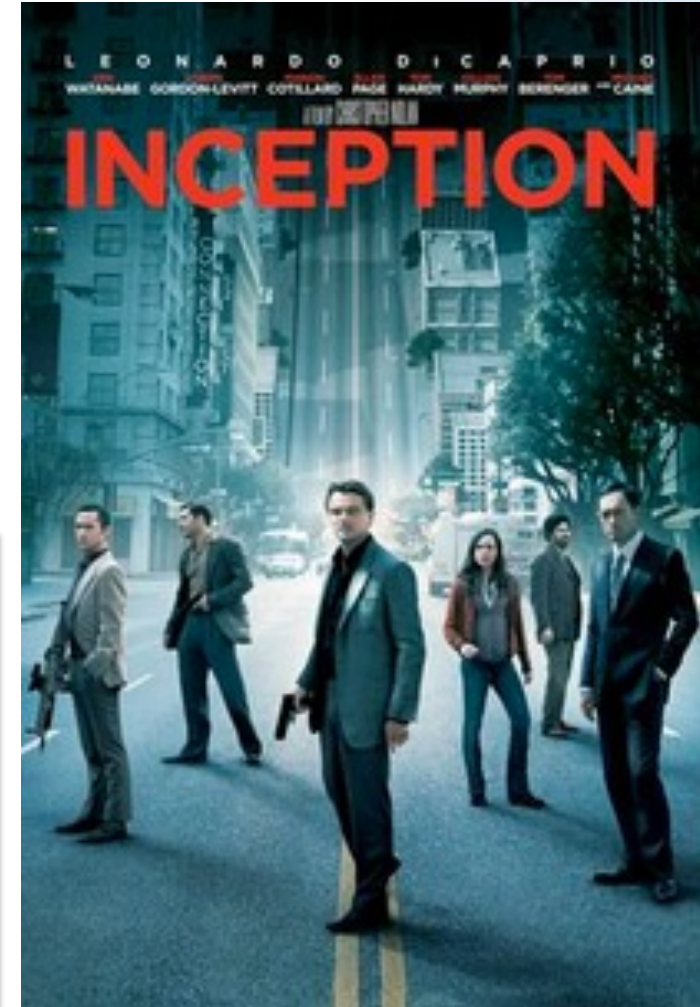
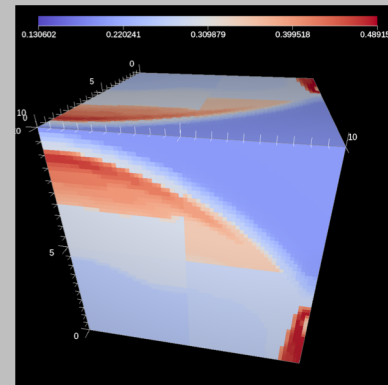
- We support python filters
 - Must edit the Ascent runtime to execute
 - How do I pass my data back to Ascent?
- Inside of a python extract
 - Create another instance of Ascent
 - Publish the new data set and actions

Ascent Level 1

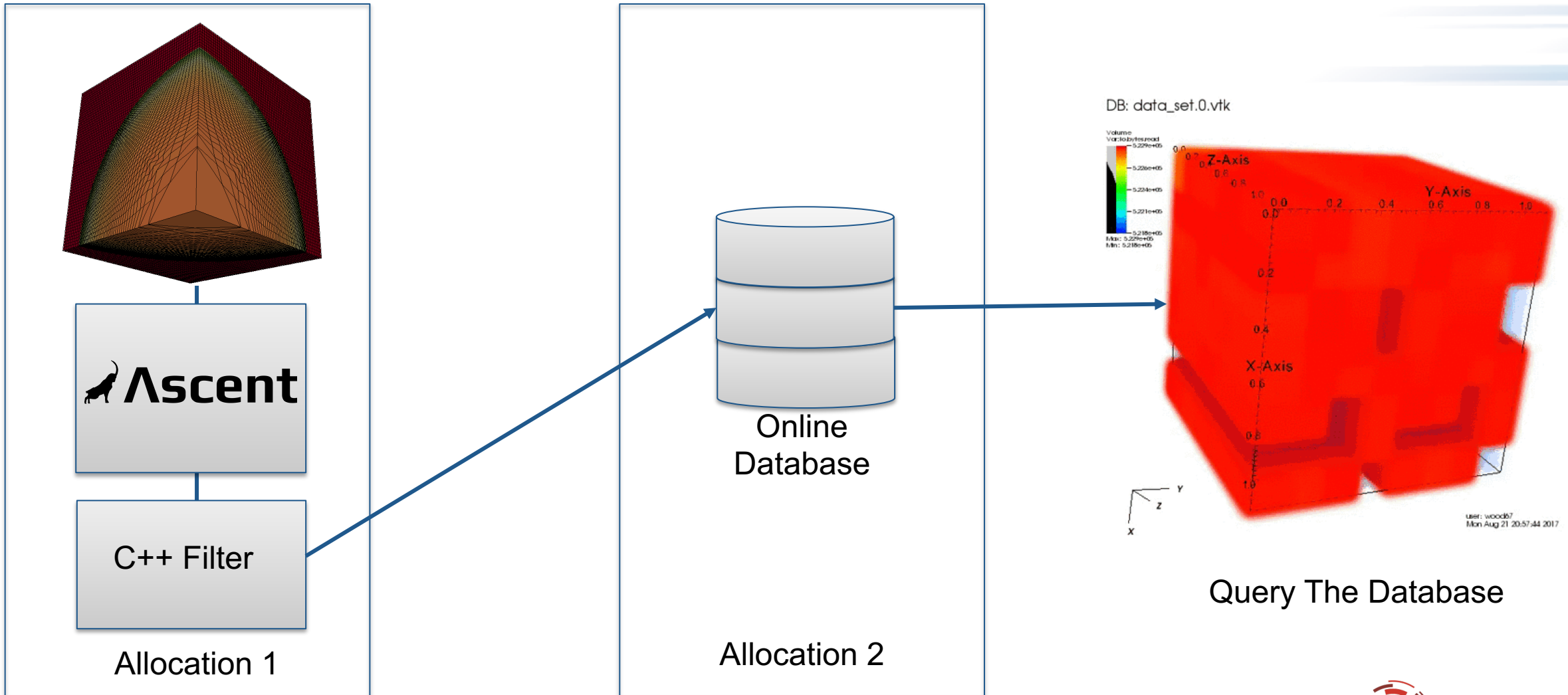


python
extract

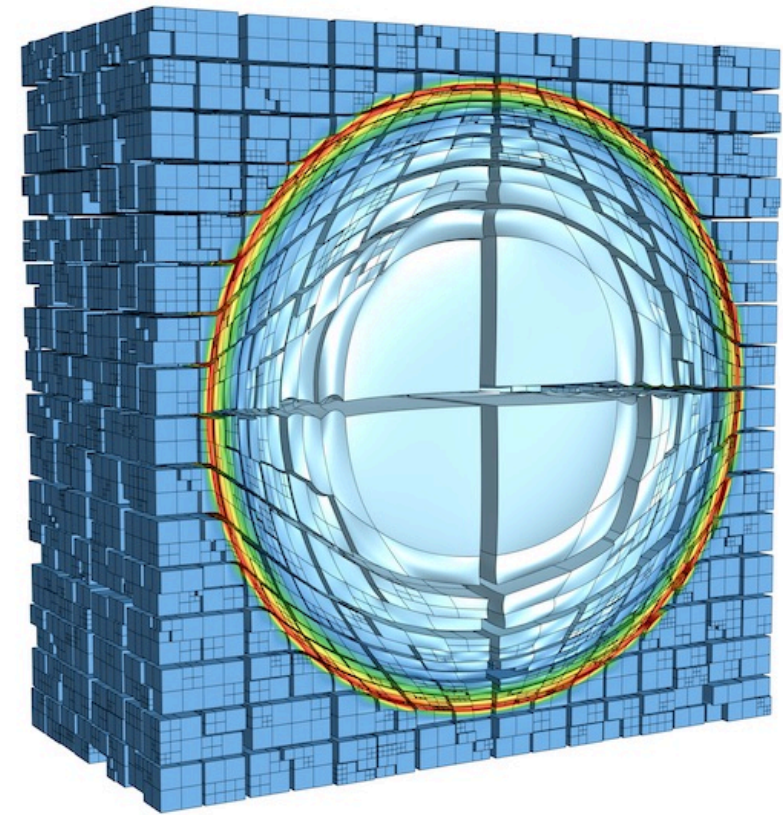
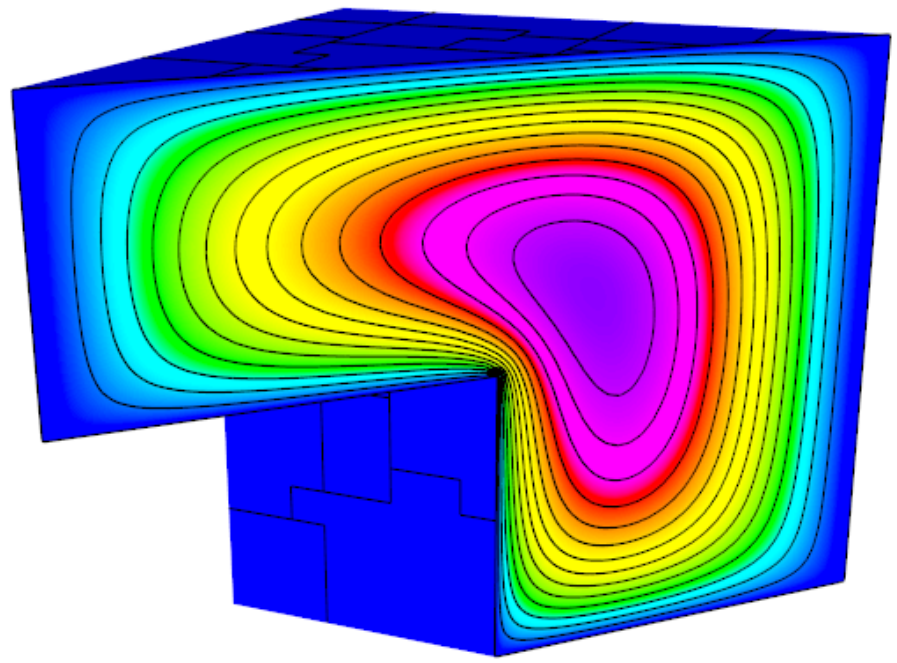
Ascent Level 2



Custom C++ Filter: Gathering performance + mesh data



Native MFEM rendering support (Beta Support)



Why is Ascent Important?

- Designed for batch-focused in-situ analysis
- Helps connect your data with other ecosystems
- Light weight
 - Streamlined API
 - Low dependency count
- Targeting unique capabilities
 - Rendering of high-order meshes
- Easy to use and extend
 - Lowers barriers to custom analysis



Ascent is ready for common visualization use cases

- GitHub Repo: <https://github.com/Alpine-DAV/ascent>
- Docs: <https://alpine-dav.github.io/ascent>
- Try it out using Docker:
 - `docker pull alpinedav/ascent`
 - More info: <http://ascent.readthedocs.io/en/latest/Tutorial.html>
- Primary Contacts:
 - **Matt Larsen** larsen30@llnl.gov
 - **Cyrus Harrison** cyrush@llnl.gov

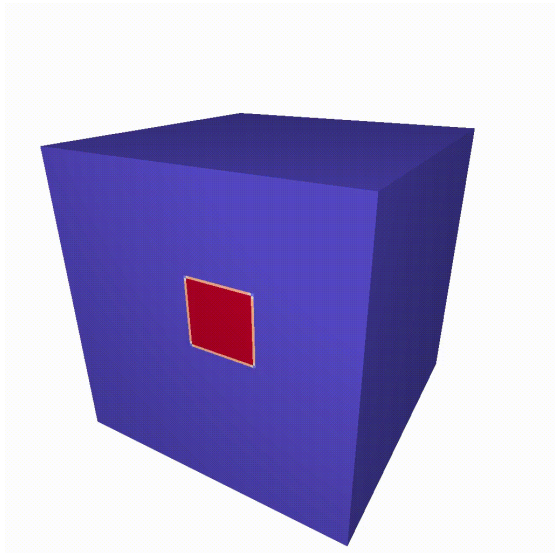


Acknowledgements

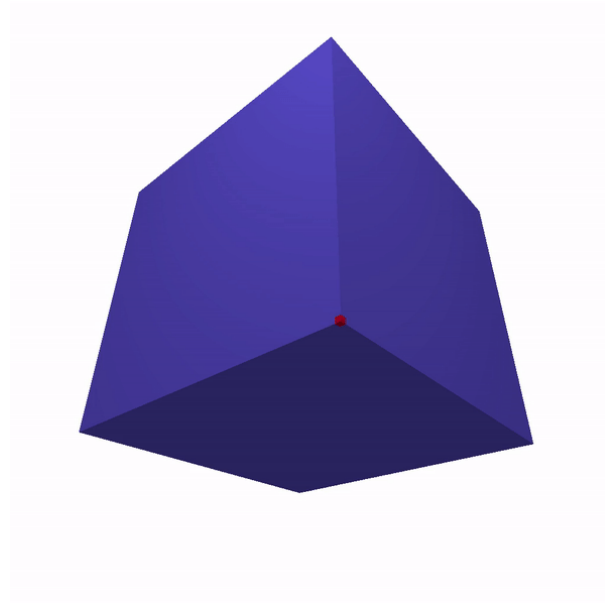
- This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy, Office of Science and the National Nuclear Security Administration.
- This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC (LLNL-CONF-737832)

Questions?

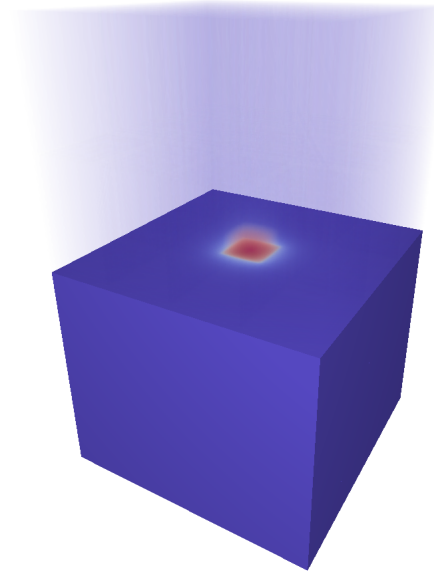
Proxy-applications included with Ascent



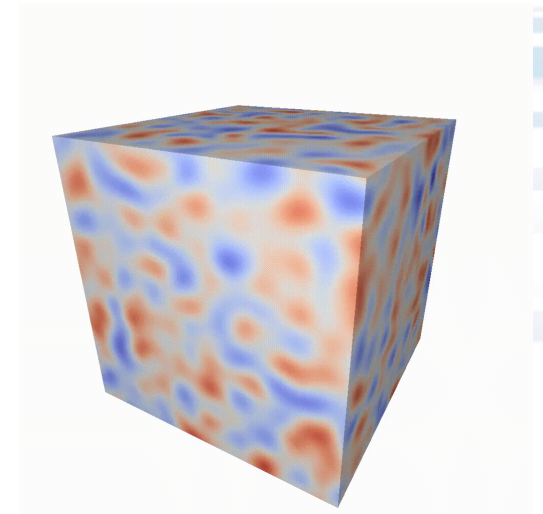
Cloverleaf3D



Lulesh



Kripke



Smooth Noise