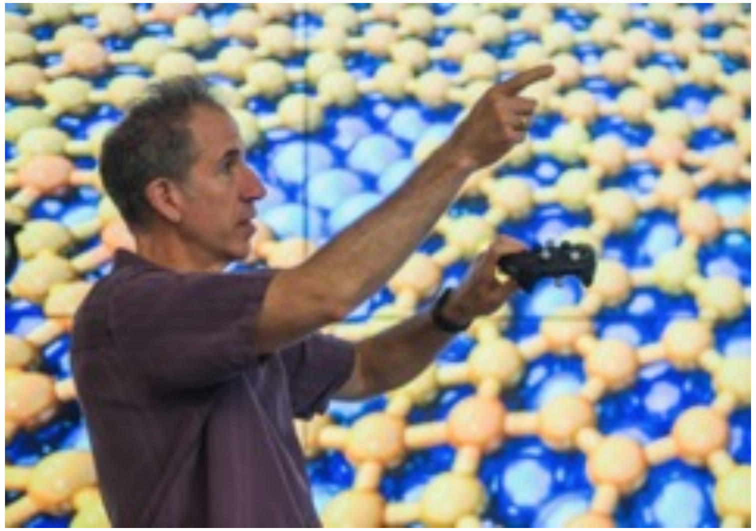
October 29-31, 2024

ALCF Hands-on HPC Workshop

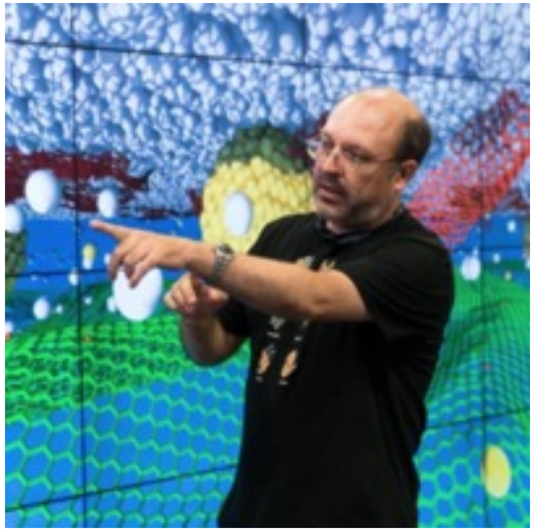
Data Analysis and Visualization

Joseph Insley
Silvio Rizzi
Victor Mateevitsi
Argonne Leadership Computing Facility

ALCF Visualization and Data Analytics Group



Joe Insley
Team Lead



Silvio Rizzi
Computer Scientist

Victor Mateevitsi
Assistant Computer Scientist

Janet Knowles
Principal Software Engineering Specialist

Geng Liu
Postdoctoral Appointee

Here's the plan…

- Examples of visualizations
- Visualization tools and formats
- Data representations
- Visualization for debugging
- In Situ Visualization and Analysis

ALCF Systems Evolution

Exascale

34 PF

15.6 PF

11.7 PF

10 PF

557 TF

5.7 TF

IBM BG/L
2004

Intrepid
IBM BG/P
2007

Mira
IBM BG/Q
2012

Theta
Intel-Cray XC40
2017

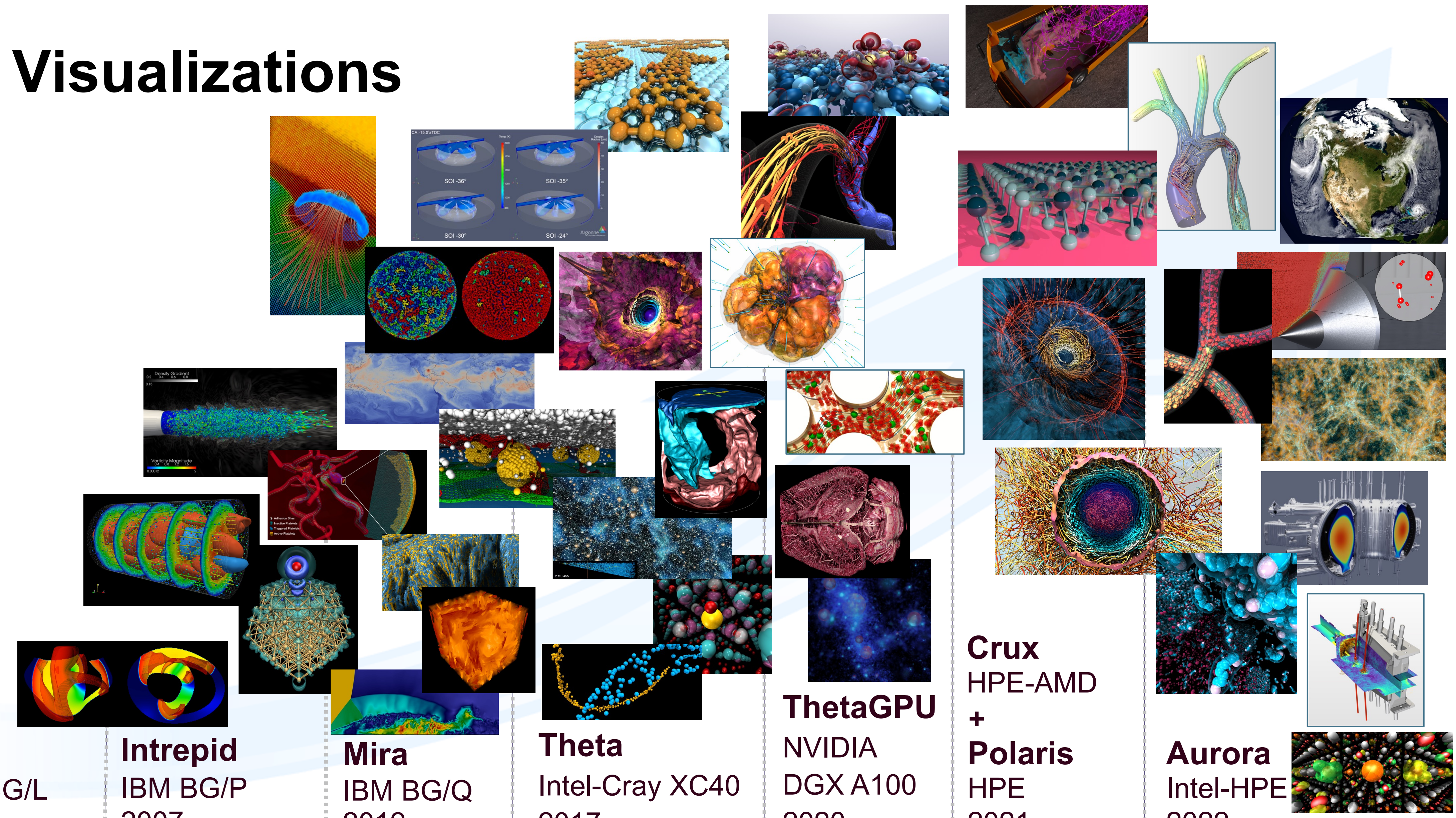
ThetaGPU
NVIDIA
DGX A100
2020

Crux
HPE-AMD
+
Polaris
HPE
2021

Aurora
Intel-HPE
2022

JLSE (2013) **AI Testbed** (2020) **Edge Testbed** (2021)

ALCF Visualizations



IBM BG/L
2004

Intrepid
IBM BG/P
2007

Mira
IBM BG/Q
2012

Theta
Intel-Cray XC40
2017

ThetaGPU
NVIDIA
DGX A100
2020

Crux
HPE-AMD
+
Polaris
HPE
2021

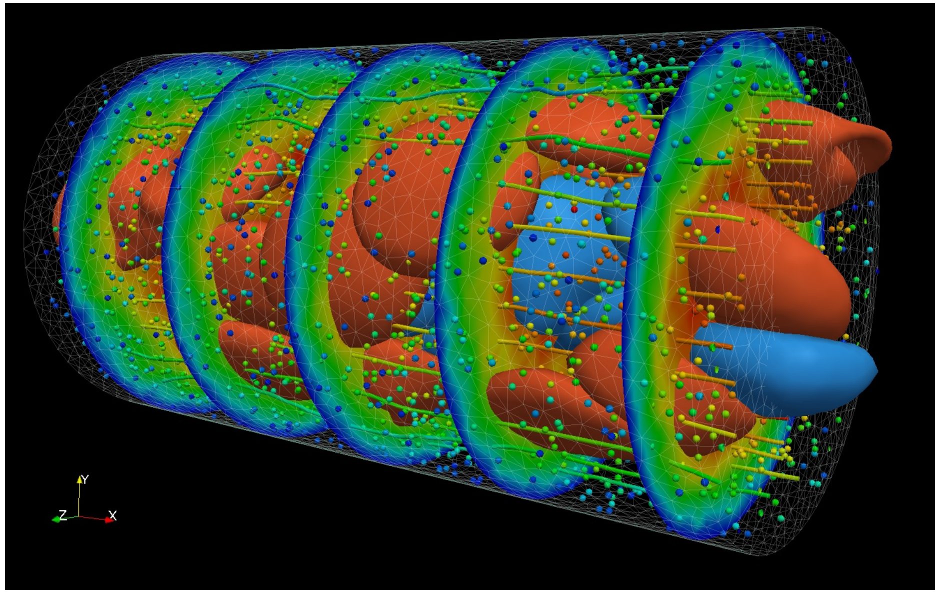
Aurora
Intel-HPE
2022

JLSE (2013) **AI Testbed** (2020) **Edge Testbed** (2021)

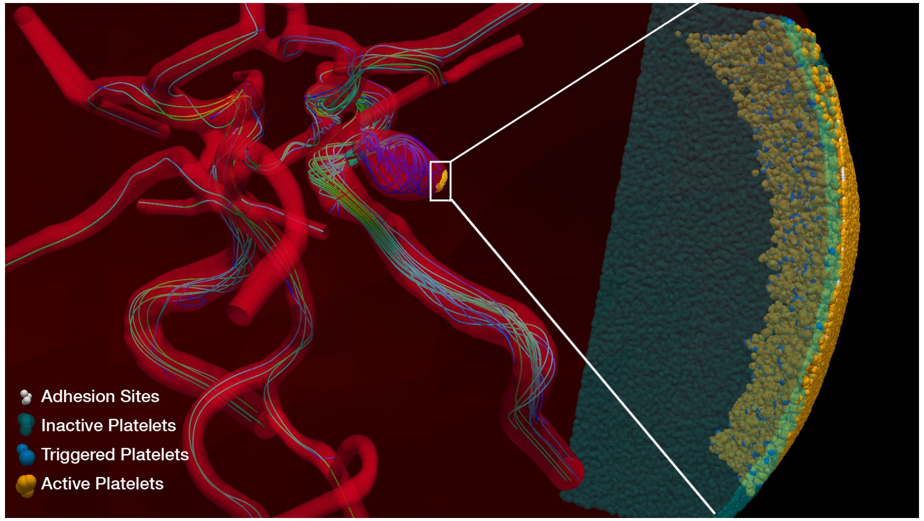
Multi-Scale Simulation / Visualization Arterial Blood Flow

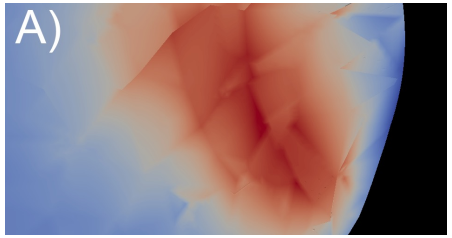
PI: George Karniadakis, Brown University

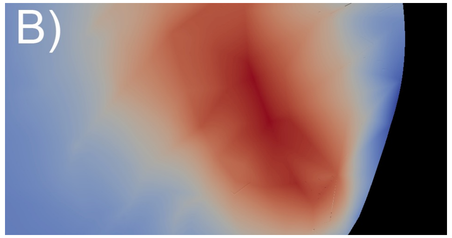
2010



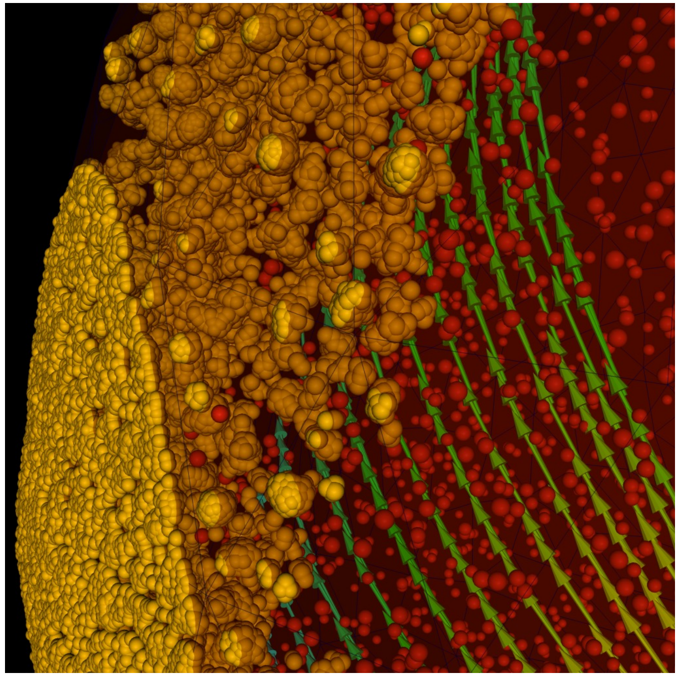
2011



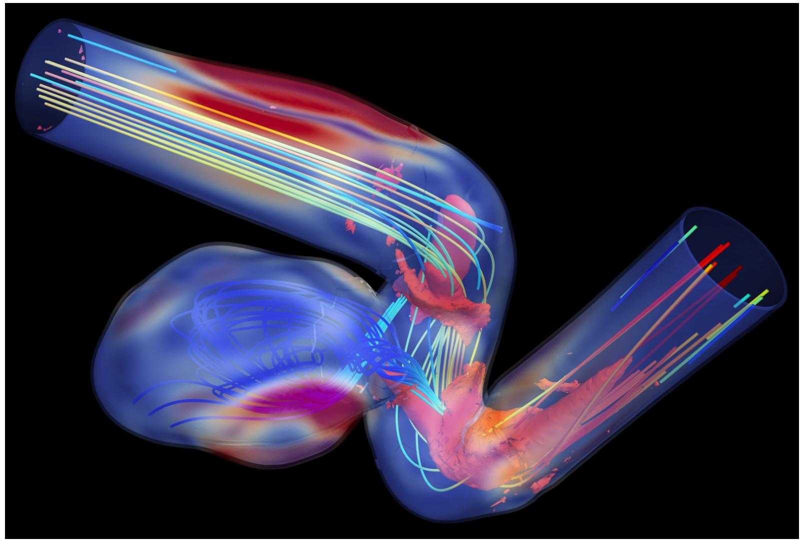




2012

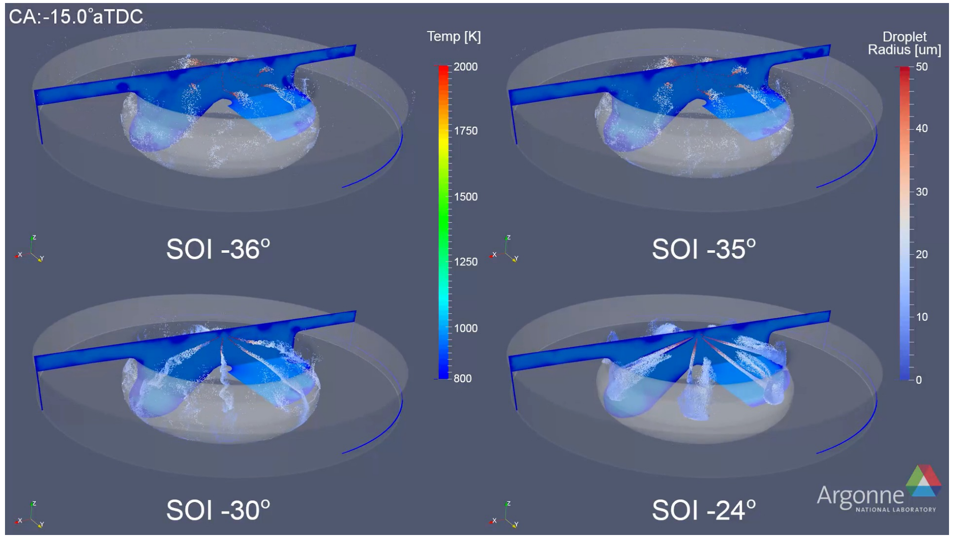


2014

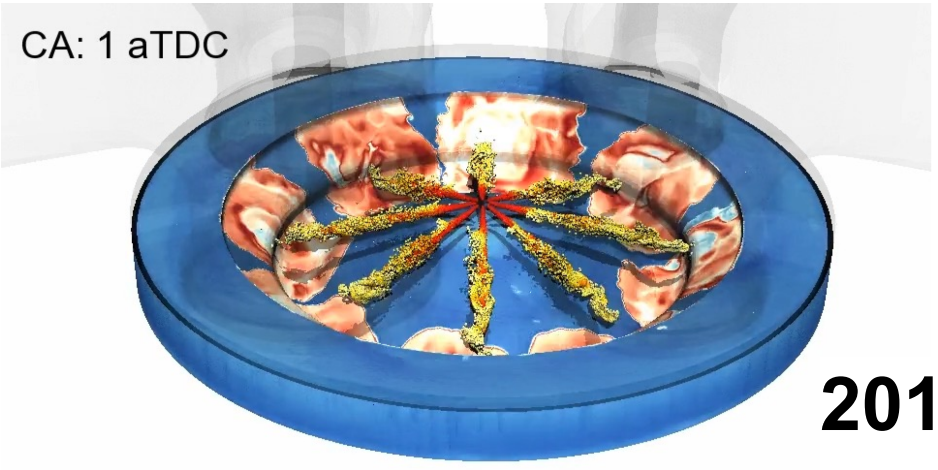


Engineering / Combustion / Biofuels

PI: Sibendu Som, Argonne National Laboratory

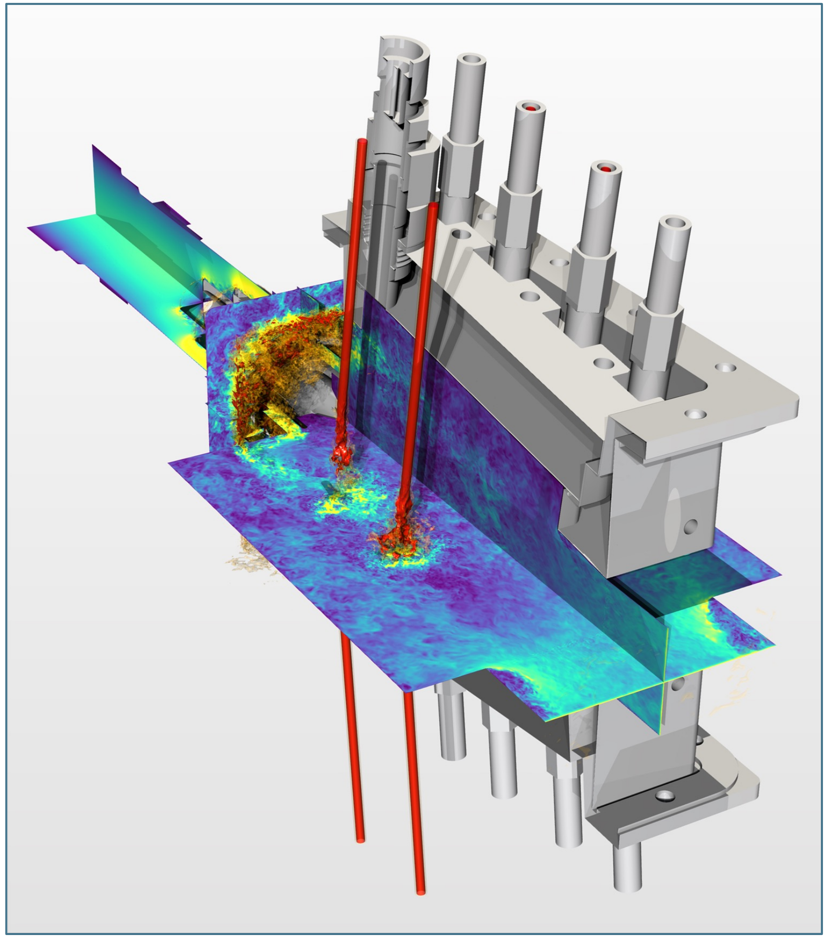


2015



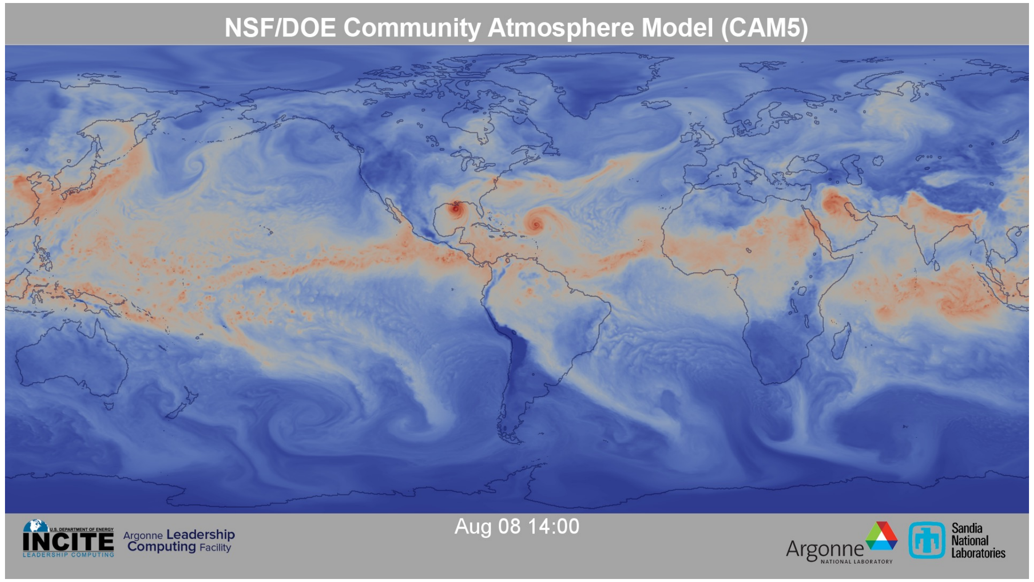
2017

2021



2023

Climate



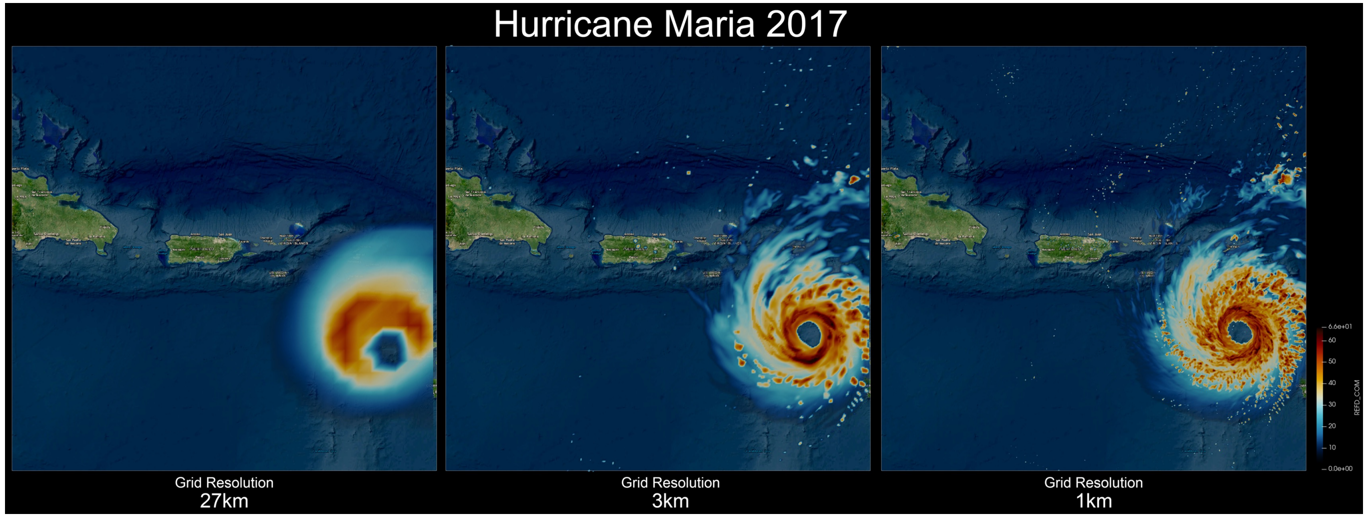
PI: Warren Washington, National Center for Atmospheric Research

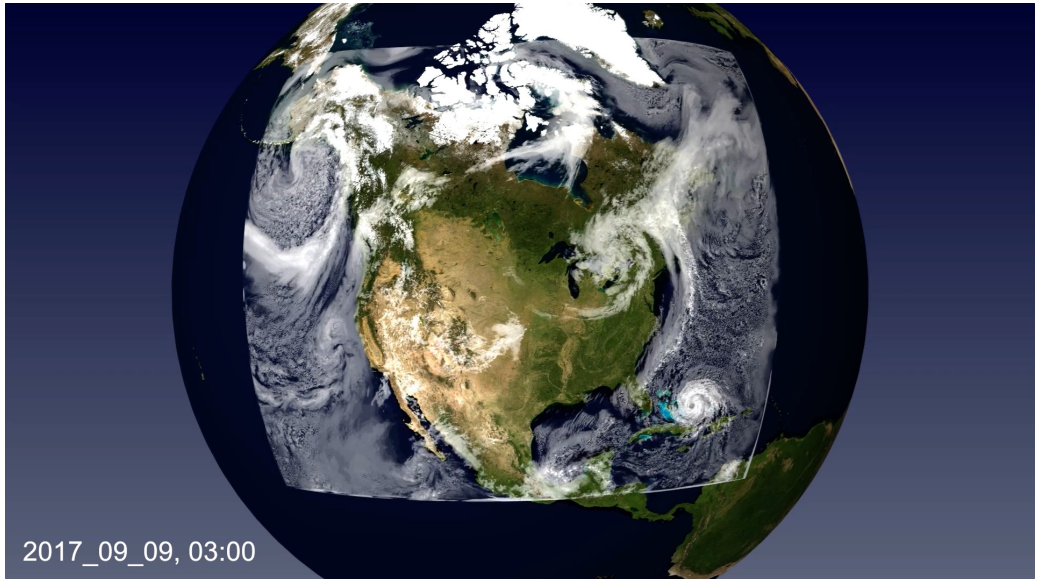
2012

PI: Rao Kotamarthi, Argonne National Laboratory

2024

2022

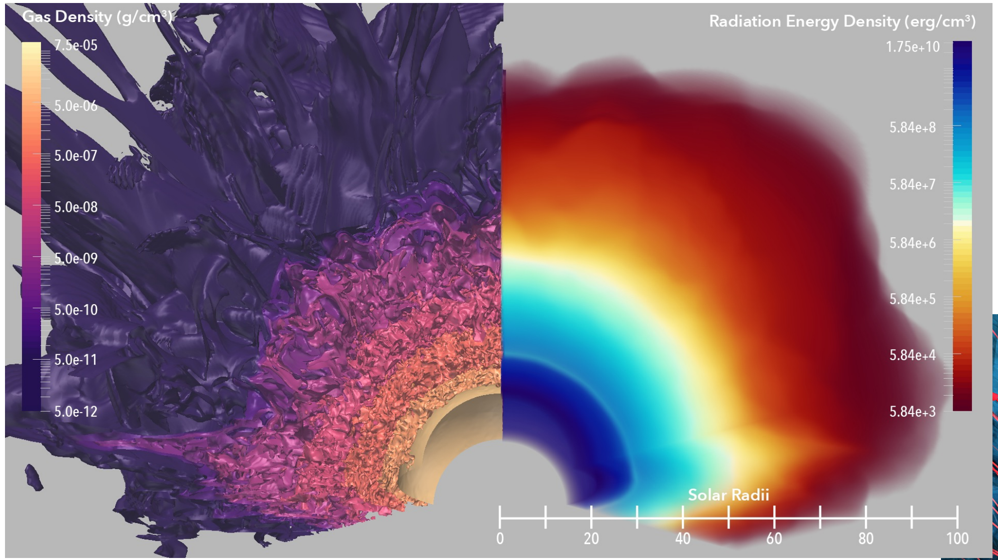




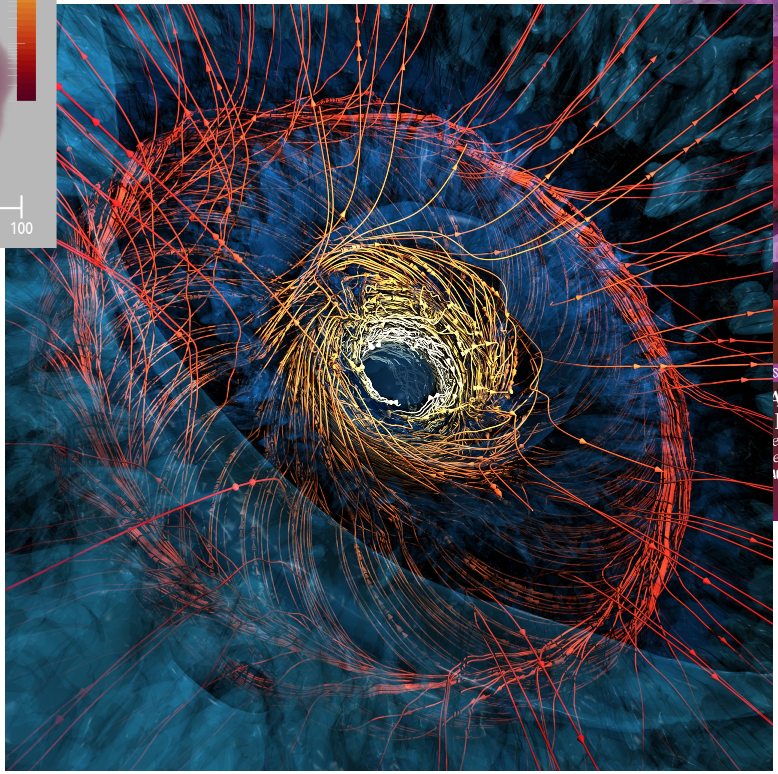
Physics: Stellar Radiation

PI: Lars Bildsten, University of California, Santa Barbara

2018



2017

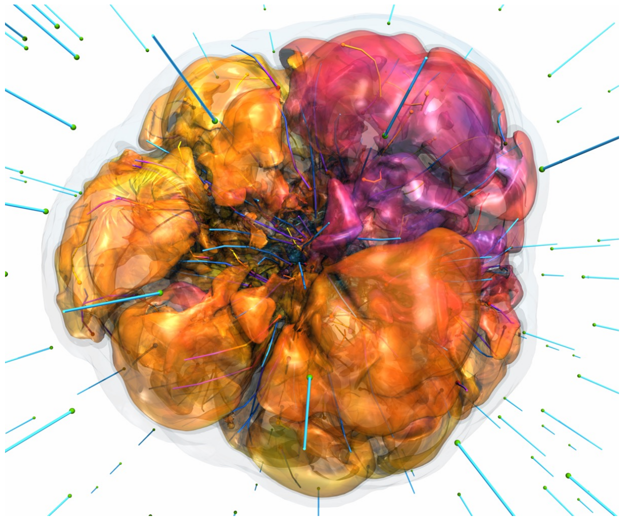


2021

Argonne NATIONAL LABORATORY

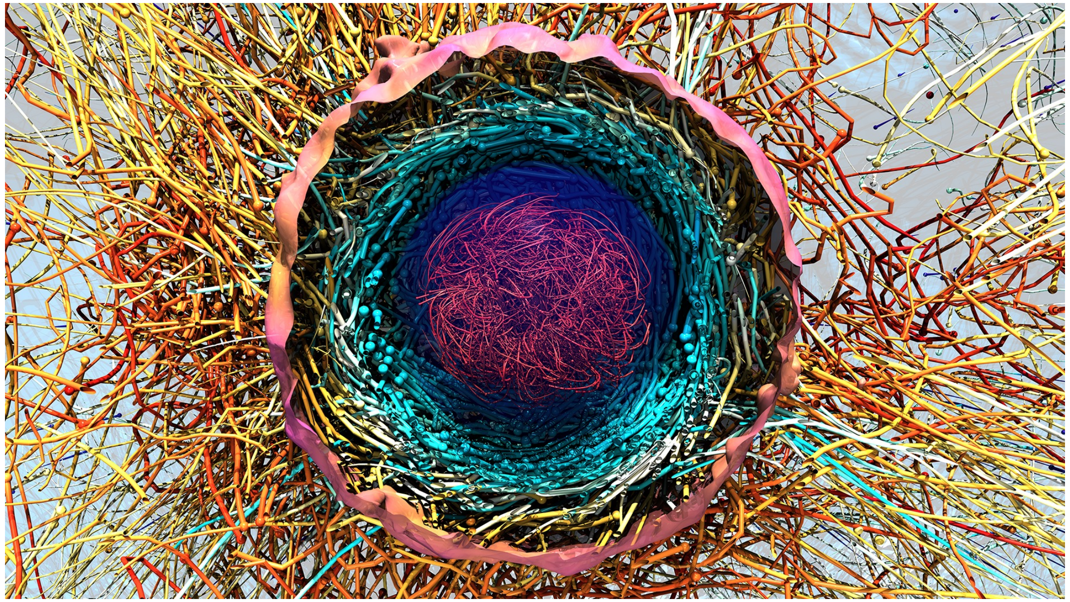
Astrophysics

PI: Adam Burrows, Princeton University

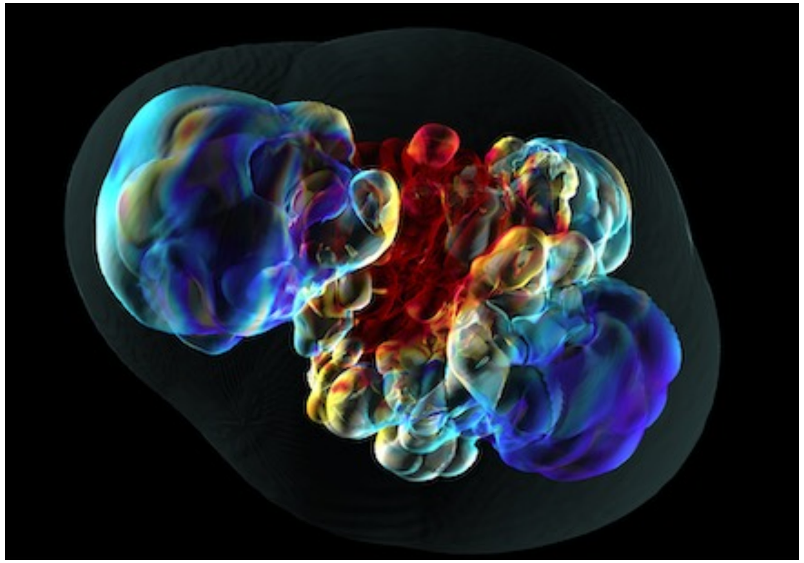


2019

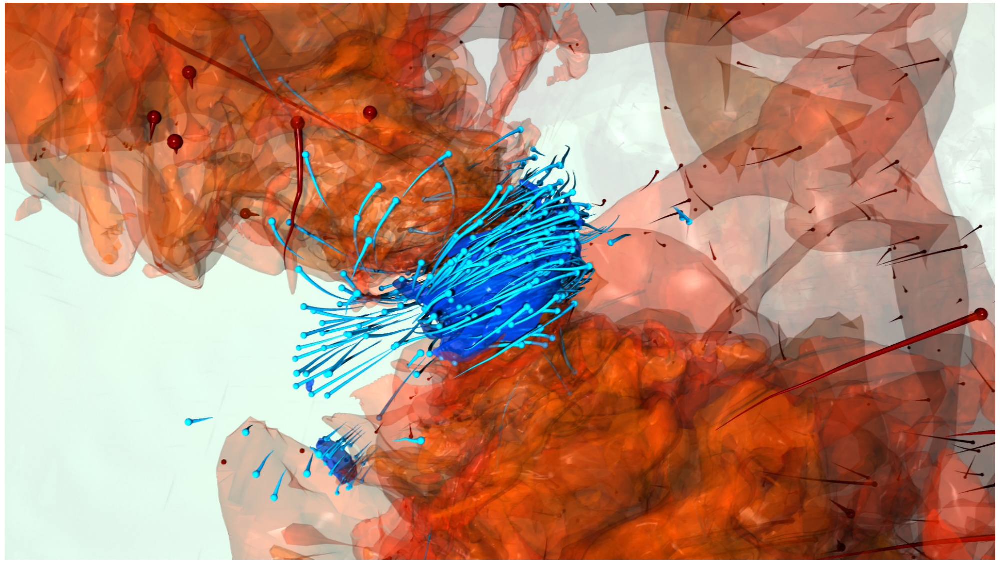
2021



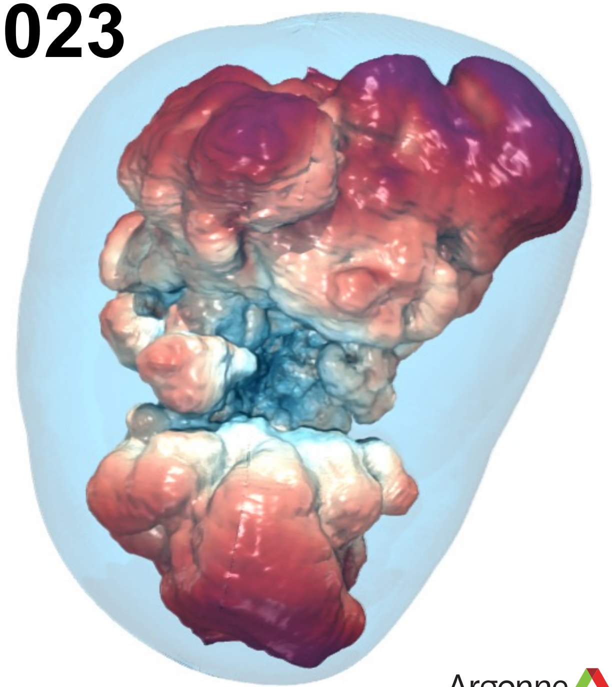
2022



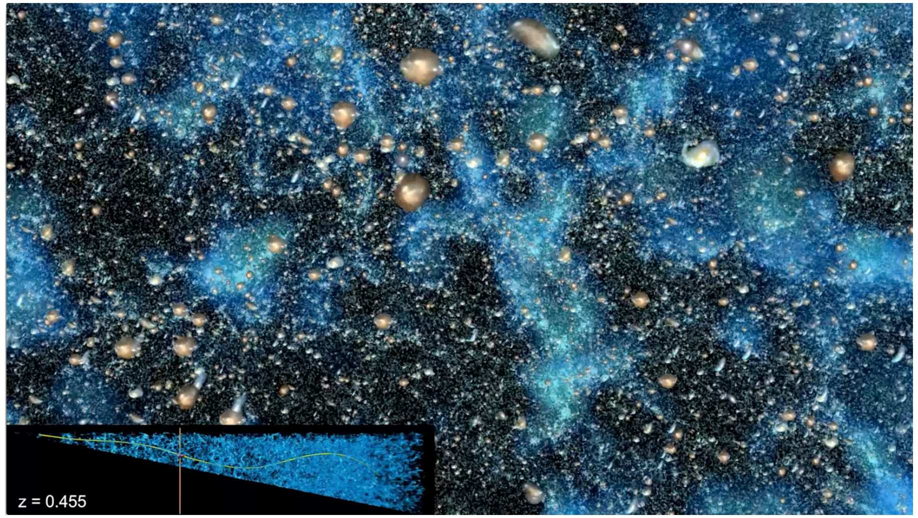
2023



2023

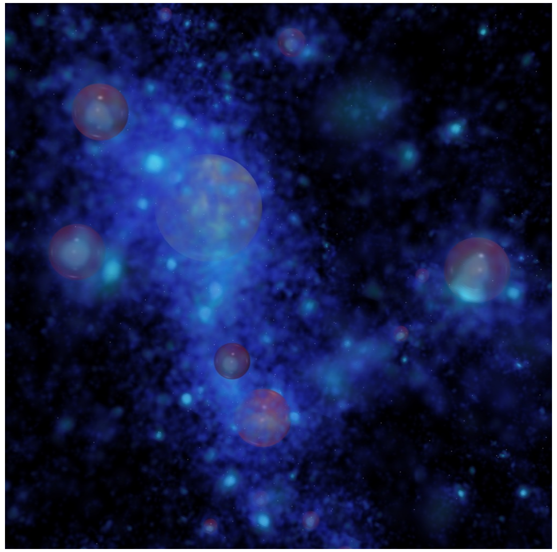


HACC: Cosmology

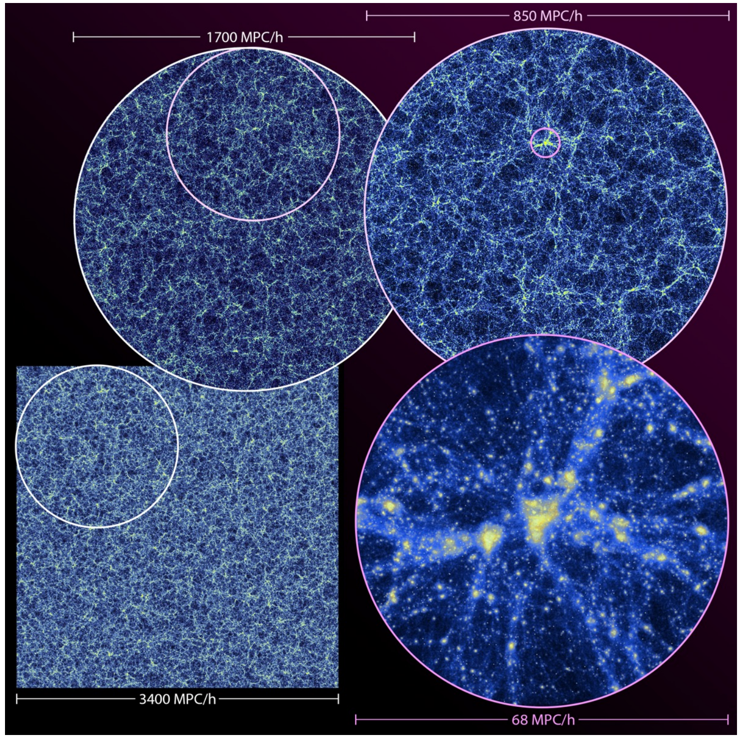


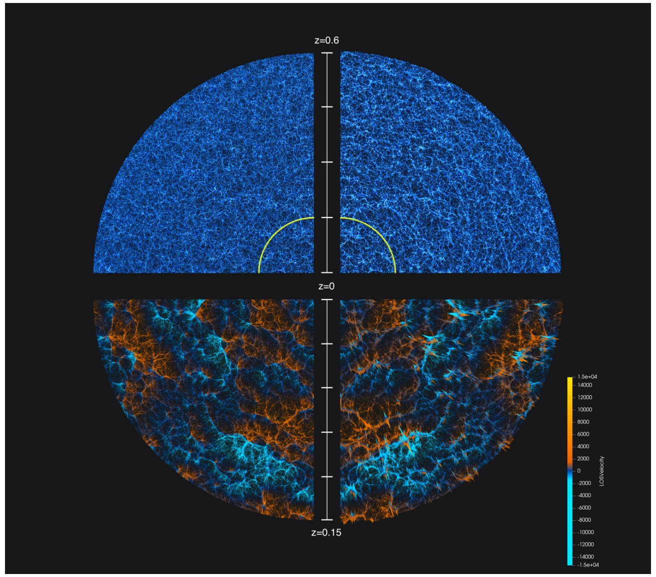
2018

2020

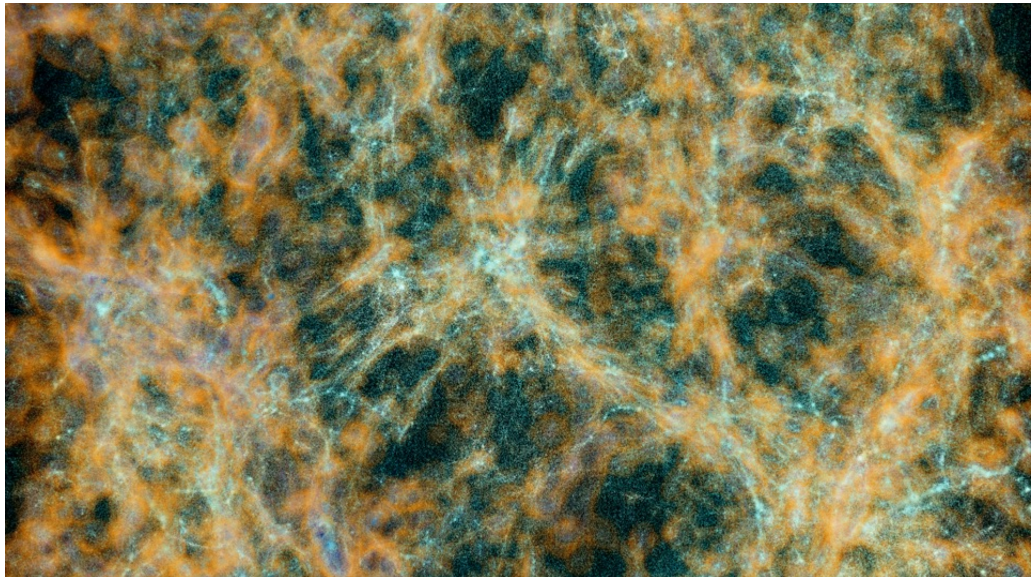


2020





2021



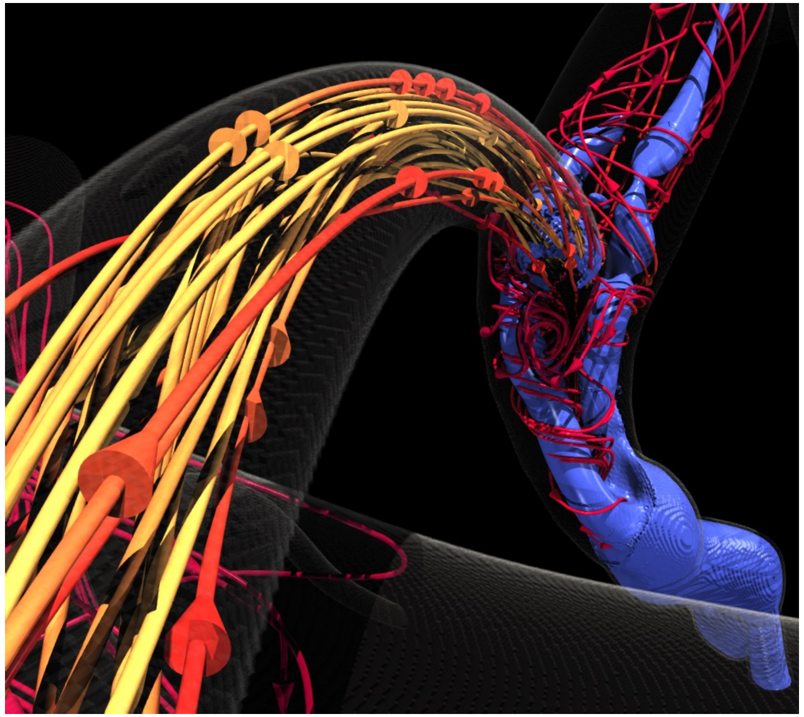
2023

Computed and Rendered on Aurora

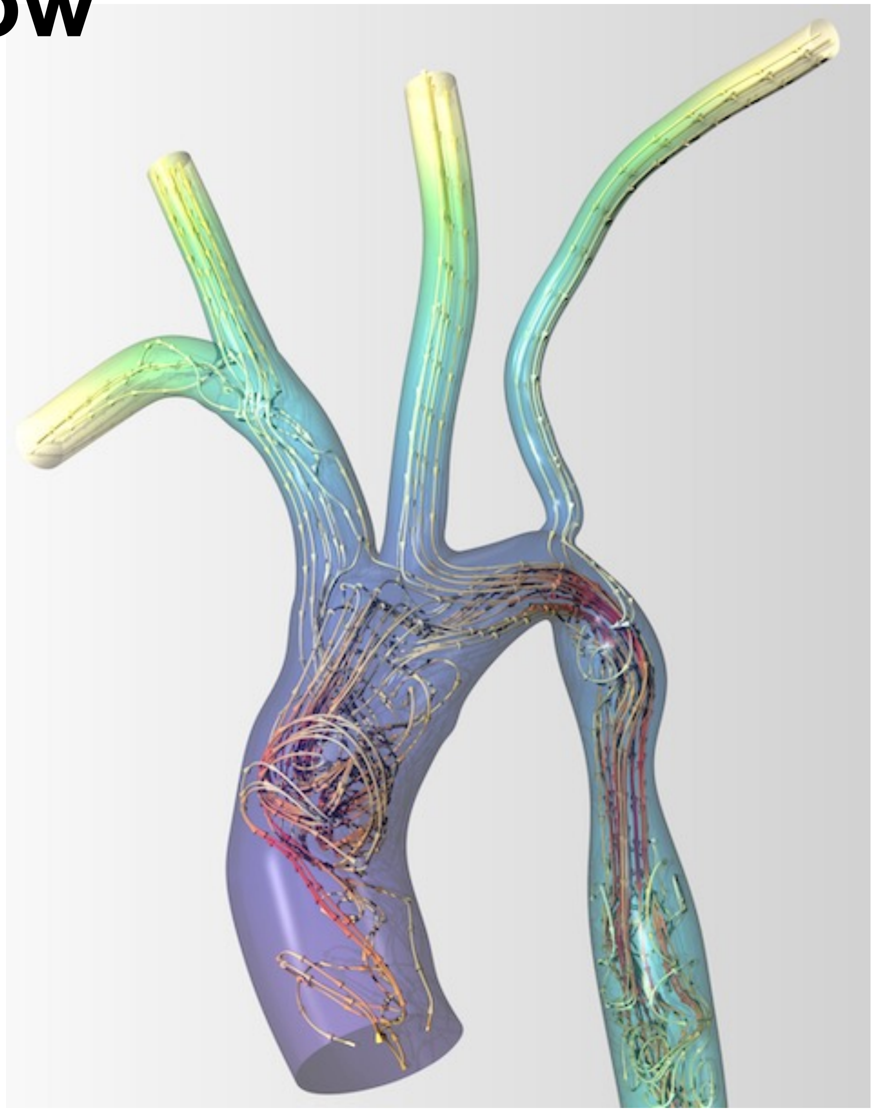
PI: Salman Habib and HACC Team, Argonne National Laboratory

Arterial Blood Flow

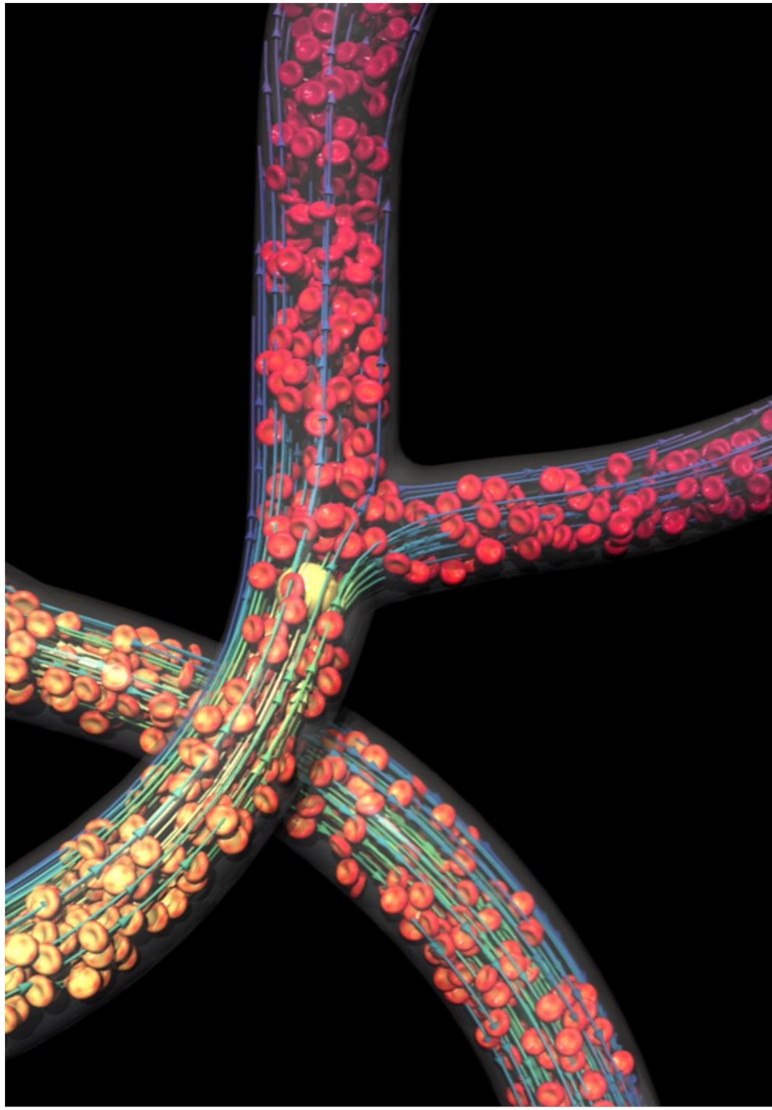
PI: Amanda Randles, Duke University



2020

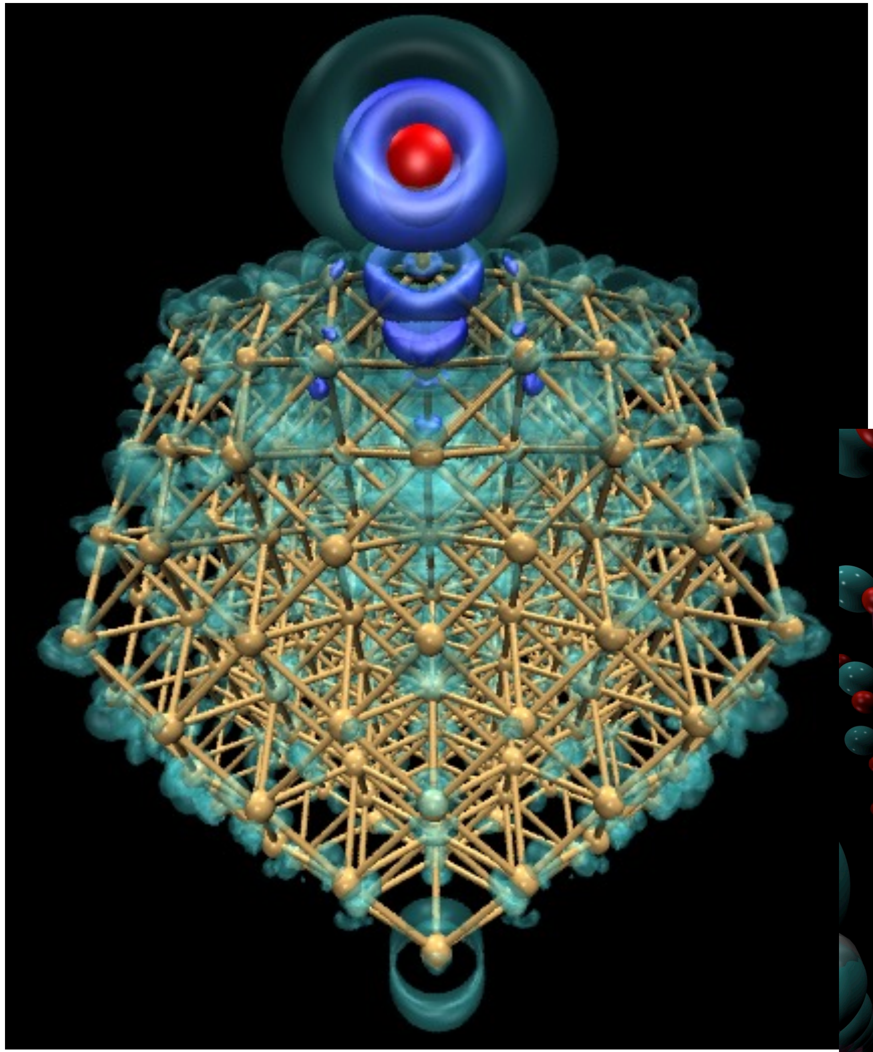


2023 Rendered on Aurora

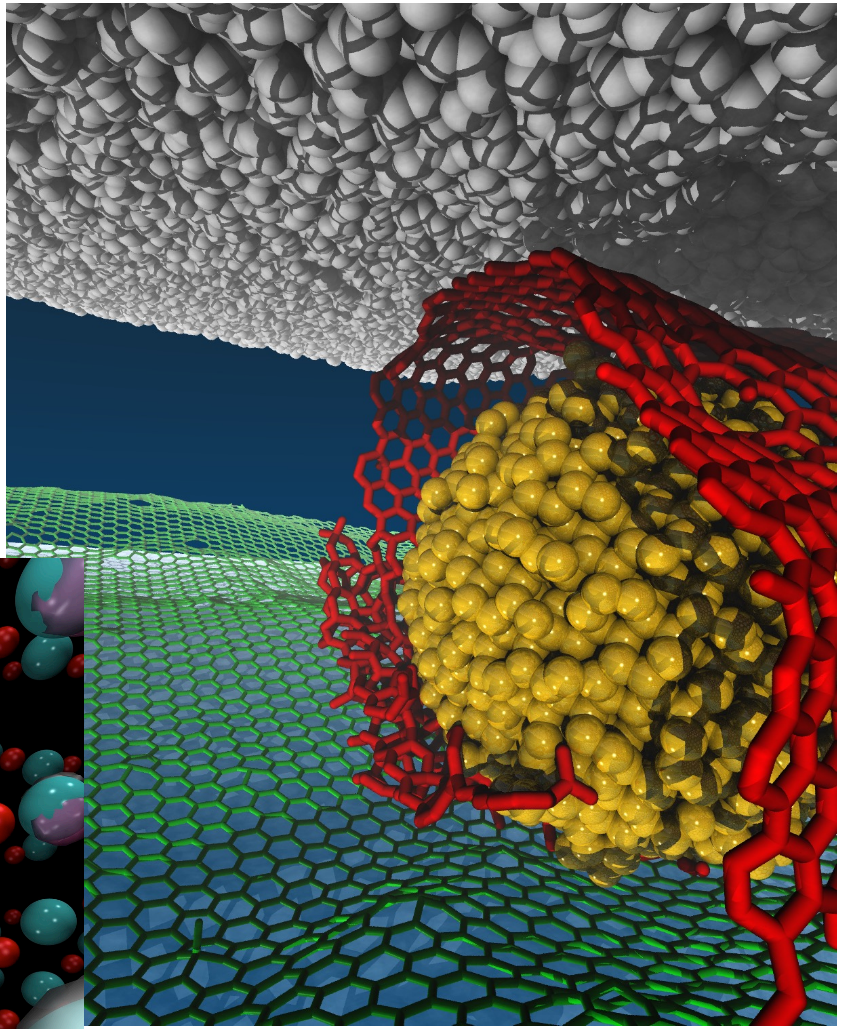


2023 Rendered on Aurora

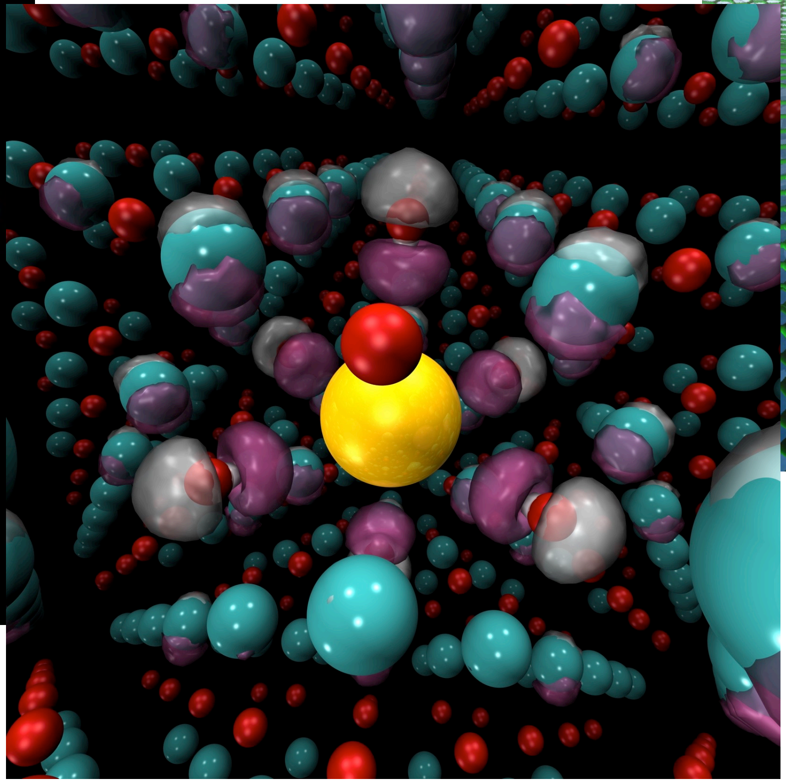
Materials Science / Molecular



Data courtesy of: Jeff Greeley, Nichols Romero, Argonne National Laboratory



Data courtesy of: Subramanian Sankaranarayanan, Argonne National Laboratory



Data courtesy of: Paul Kent, Oak Ridge National Laboratory, Anouar Benali, Argonne National Laboratory

Visualization Resources, Tools, and Data Formats

VISUALIZATION RESOURCES

Polaris

SOPHIA

All Sorts of Tools

Visualization Applications
– VisIt
– ParaView
– EnSight

Domain Specific
– VMD, PyMol, Ovito, Vapor

APIs
– VTK: visualization
– ITK: segmentation & registration

Analysis Environments
– Matlab
– Parallel R

Utilities
– GnuPlot
– ImageMagick

Available on Cooley

ParaView & VisIt vs. vtk

ParaView & VisIt

- General purpose visualization applications
- GUI-based
- Client / Server model to support remote visualization
- Scriptable / Extendable
- Built on top of vtk (largely)
- *In situ* capabilities

vtk

- Programming environment / API
- Additional capabilities, finer control
- Smaller memory footprint
- Requires more expertise (build custom applications)

Data File Formats (ParaView & VisIt)

- VTK
- Parallel (partitioned) VTK
- VTK MultiBlock (MultiGroup, Hierarchical, Hierarchical Box)
- Legacy VTK
- Parallel (partitioned) legacy VTK
- EnSight files
- EnSight Master Server
- Exodus
- BYU
- XDMF
- PLOT2D
- PLOT3D
- SpyPlot CTH
- HDF5 raw image data
- DEM
- VRML
- PLY
- Polygonal Protein Data Bank
- XMol Molecule
- Stereo Lithography
- Gaussian Cube
- Raw (binary)
- AVS
- Meta Image
- Facet
- PNG
- SAF
- LS-Dyna
- Nek5000
- OVERFLOW
- paraDIS
- PATRAN
- PFLOTRAN
- Pixie
- PuReMD
- S3D
- SAS
- Tetrad
- UNIC
- VASP
- ZeusMP
- ANALYZE
- BOV
- GMV
- Tecplot
- Vis5D
- Xmdv
- XSF

Data Representations

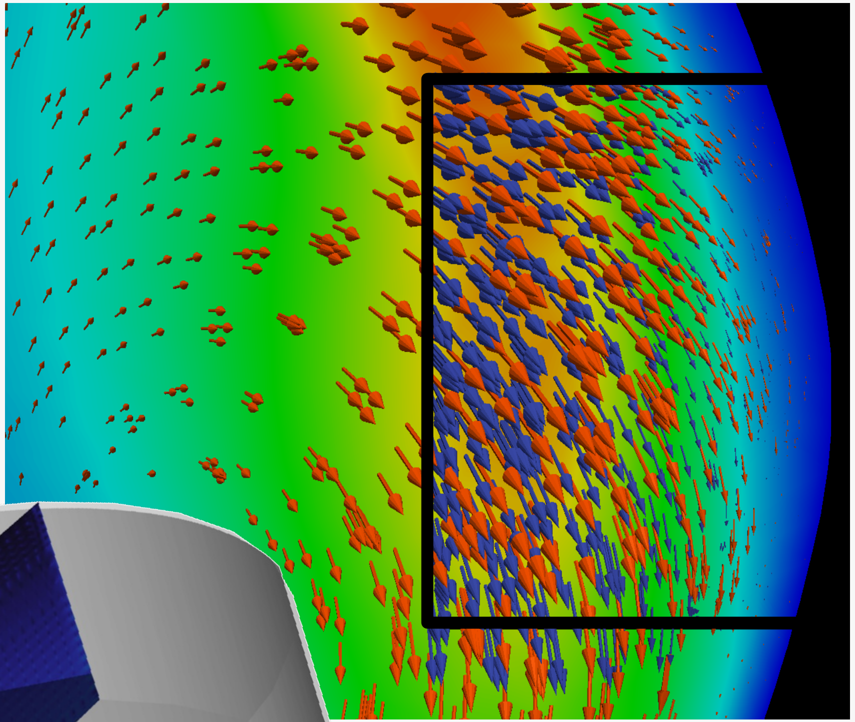
Data Representations: Cutting Planes

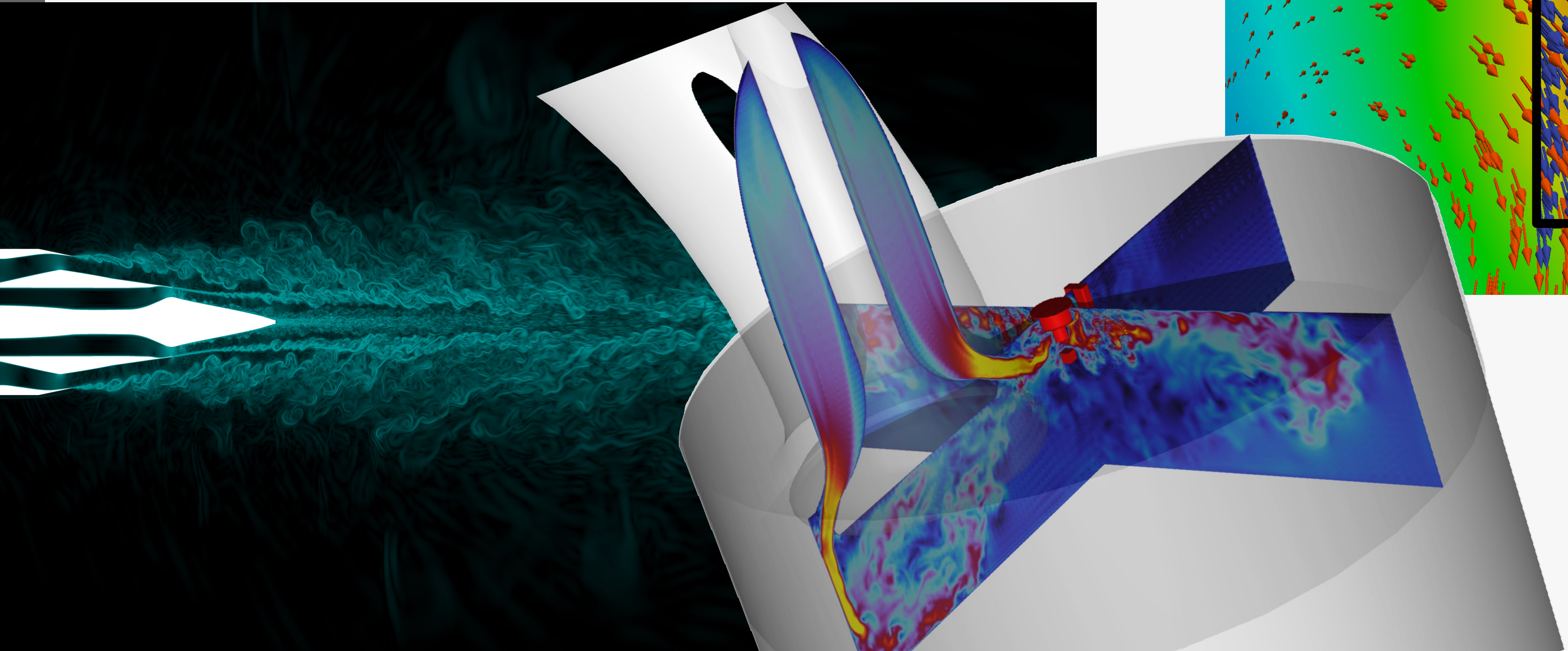
Slice a plane through the data

– Can apply additional visualization methods to resulting plane

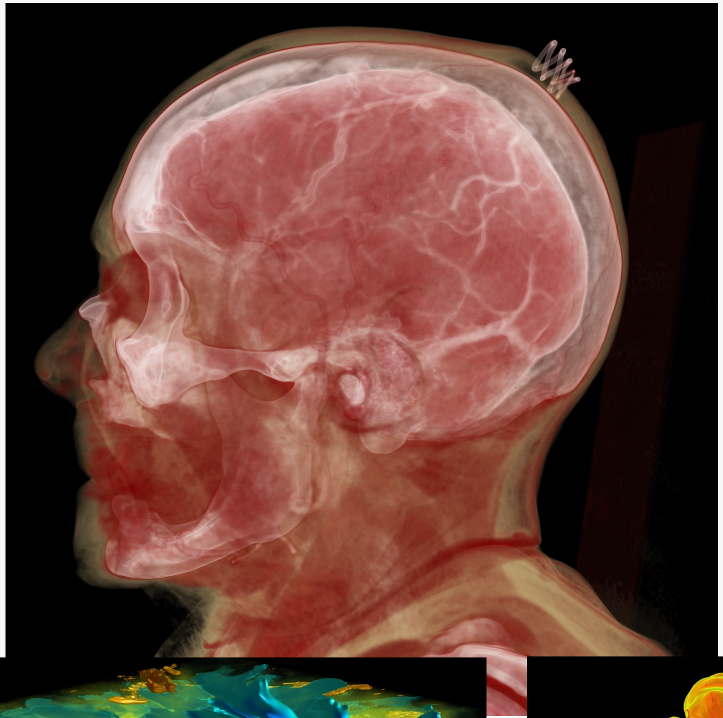
VisIt & ParaView & vtk good at this

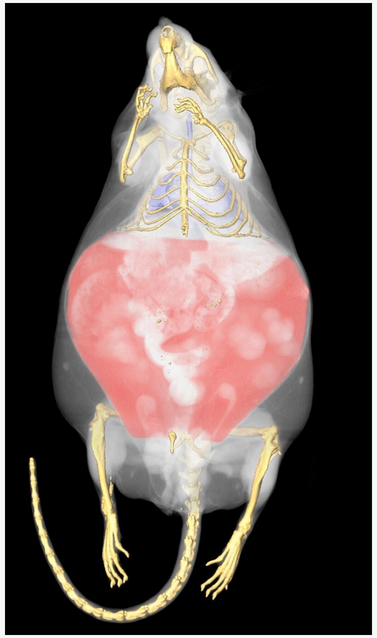
VMD has similar capabilities for some data formats

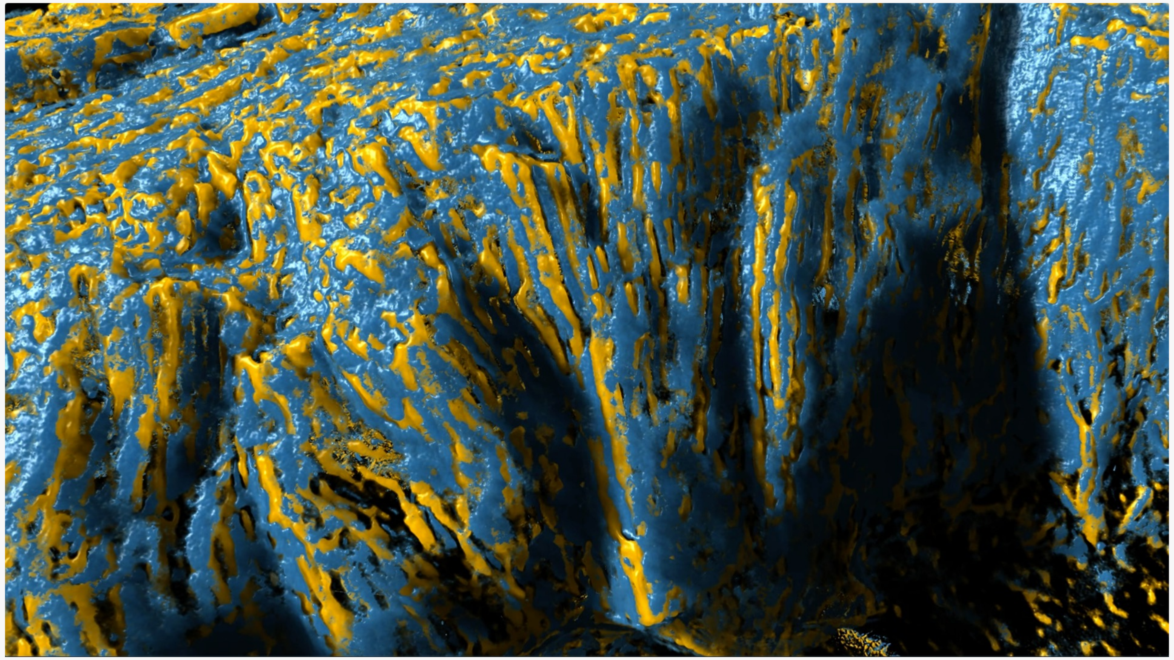


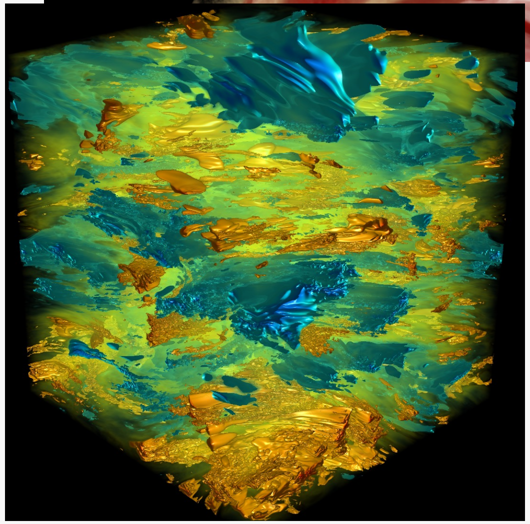


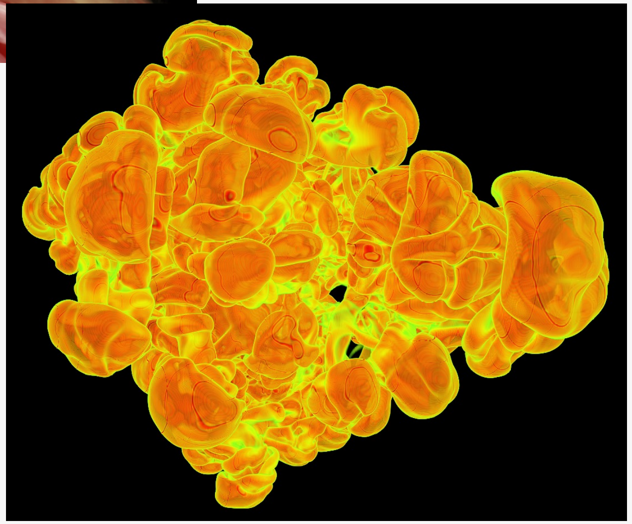
Data Representations: Volume Rendering

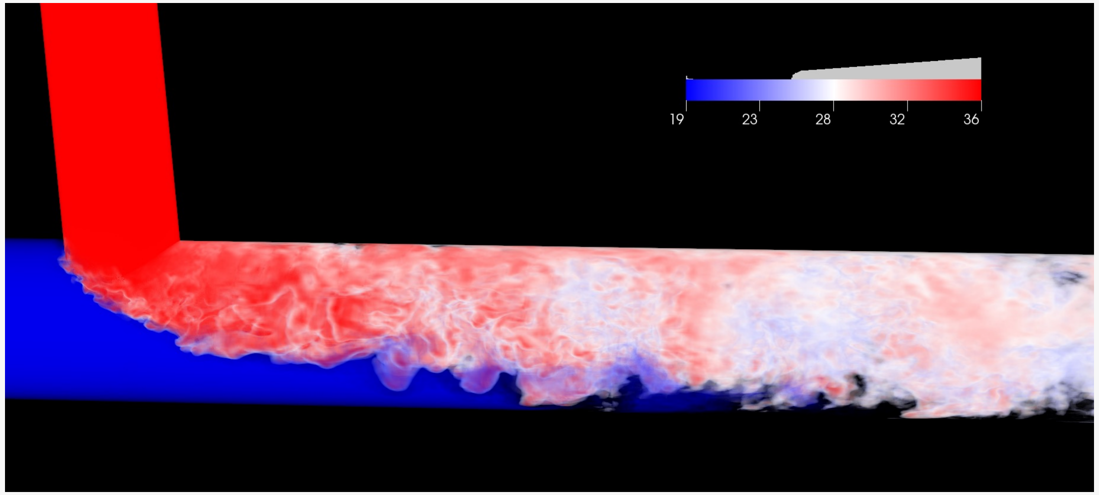












Data Representations: Contours (Isosurfaces)

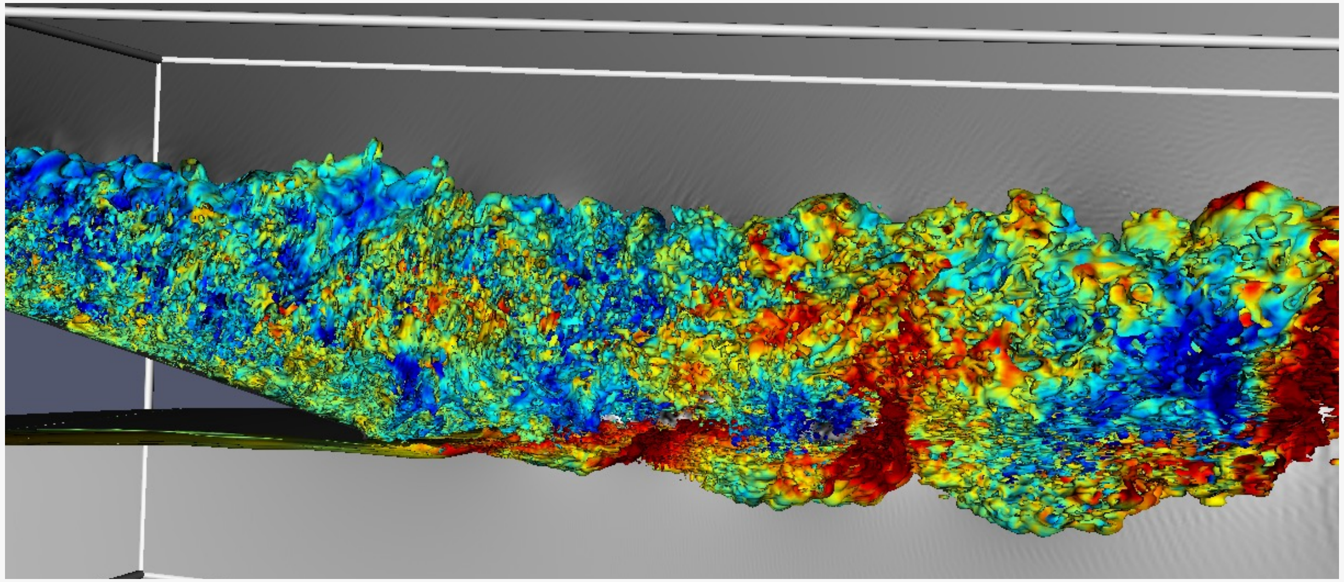
A Line (2D) or Surface (3D), representing a constant value

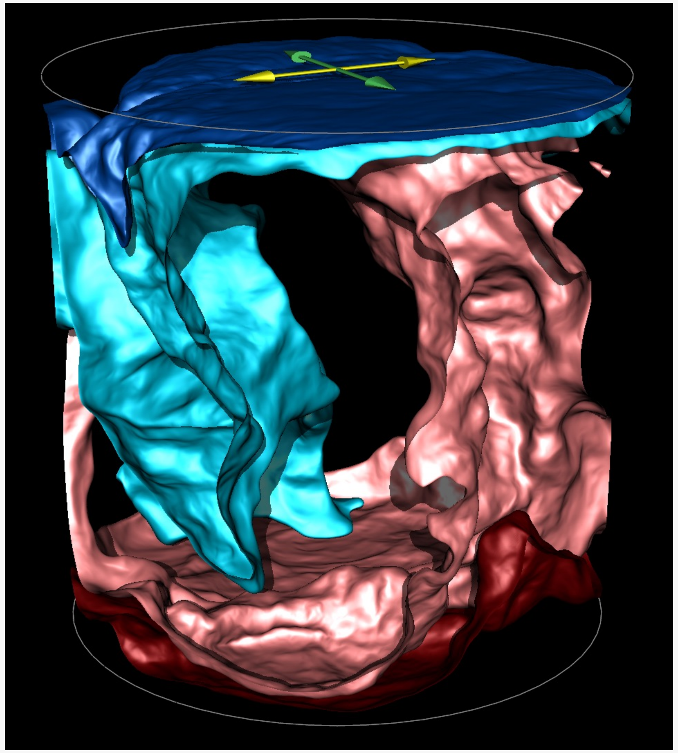
VisIt & ParaView:

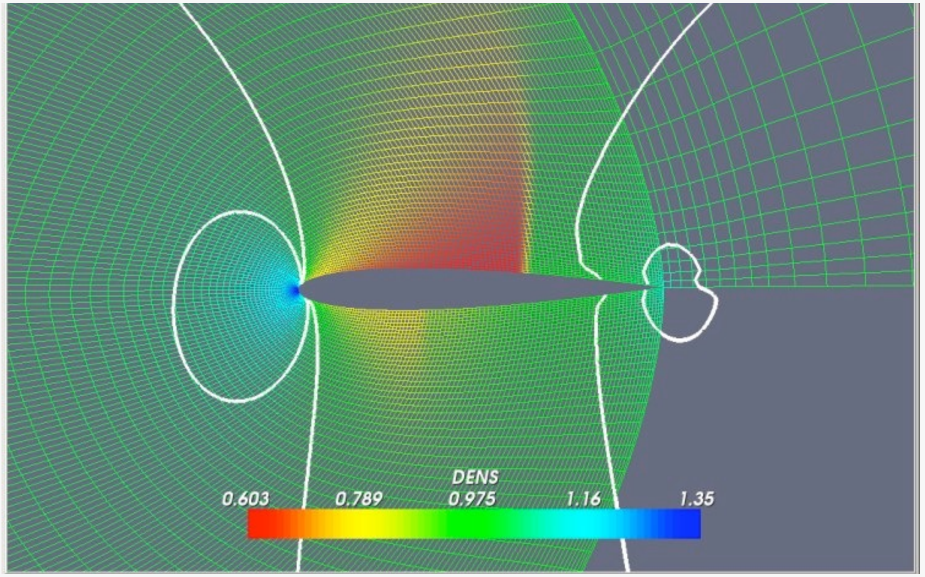
– good at this

vtk:

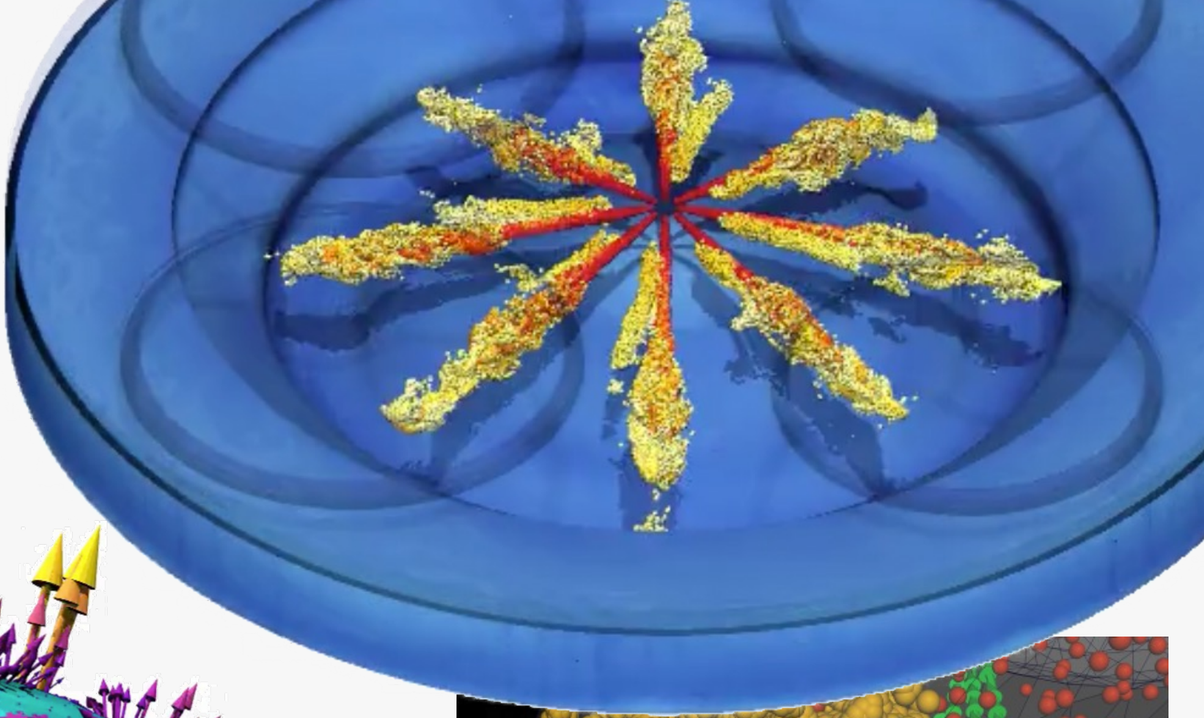
– same, but again requires more effort







Data Representations: Glyphs



2D or 3D geometric object to represent point data

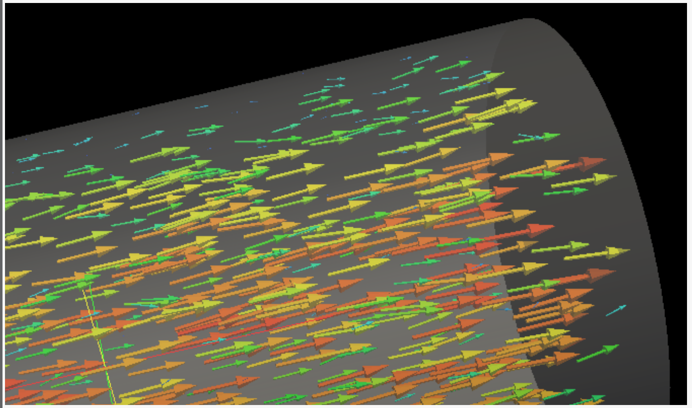
Location dictated by coordinate

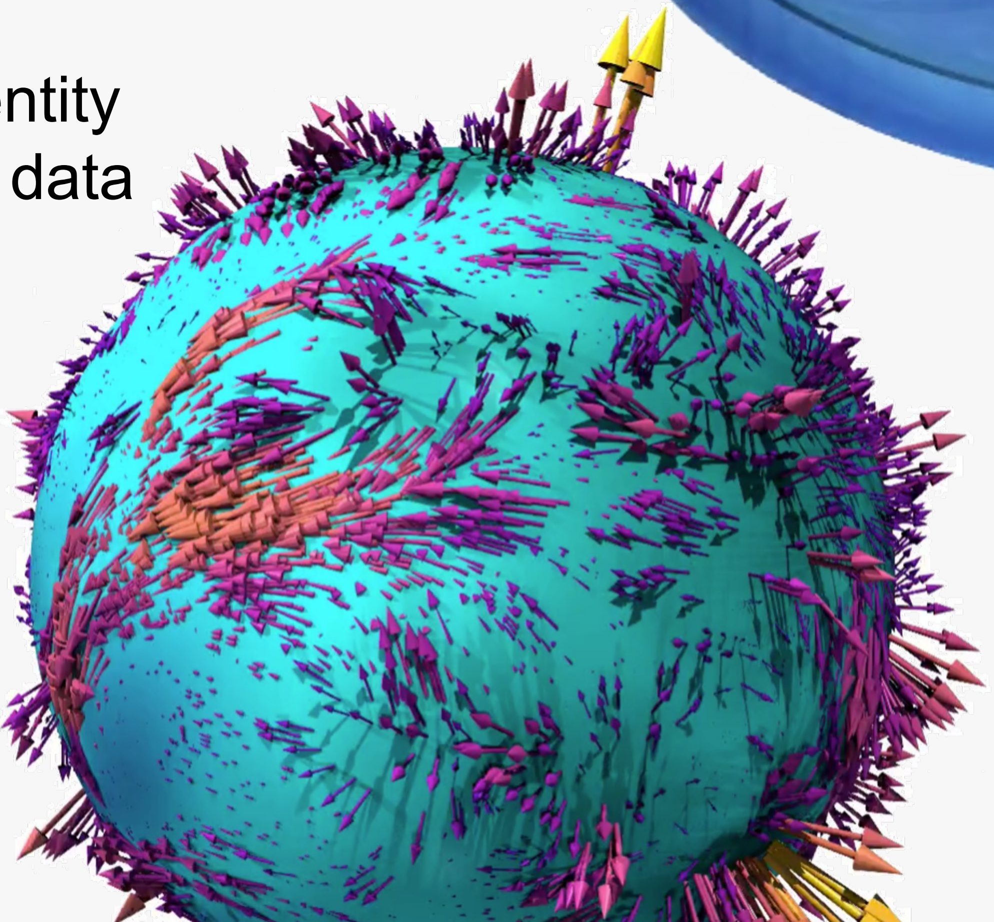
– 3D location on mesh

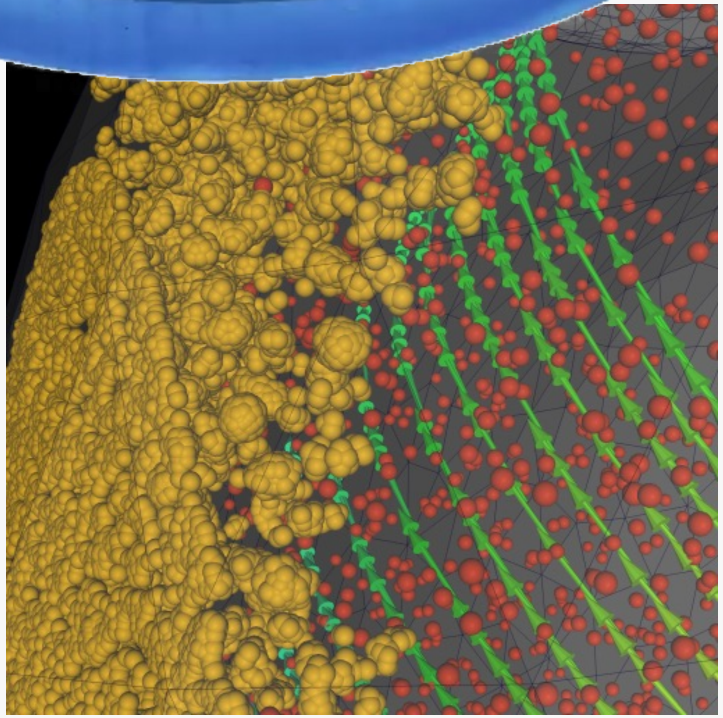
– 2D position in table/graph

Attributes of graphical entity dictated by attributes of data

– color, size, orientation





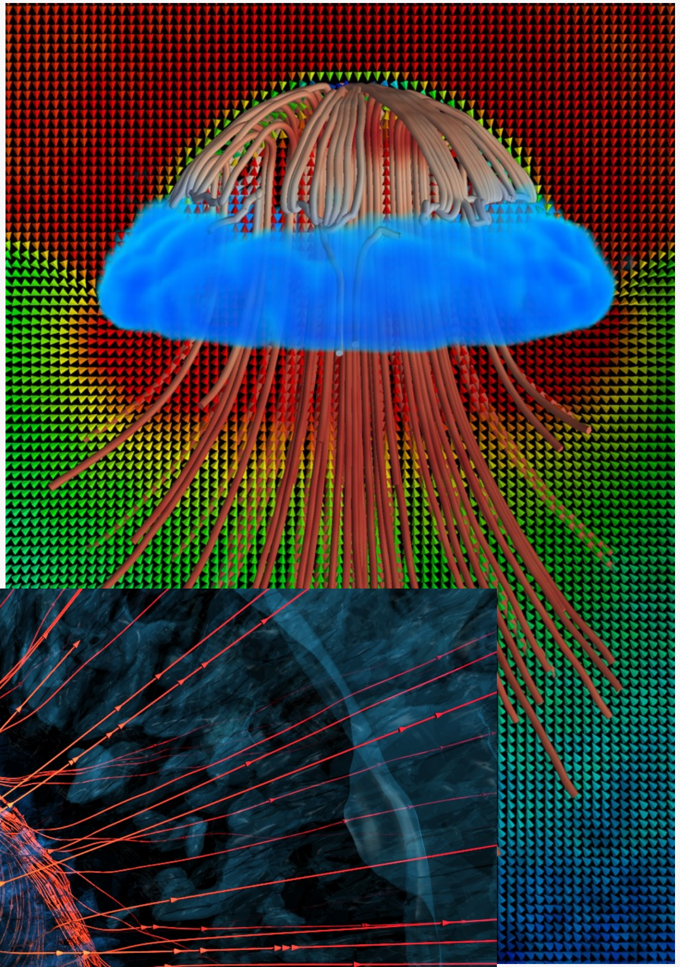


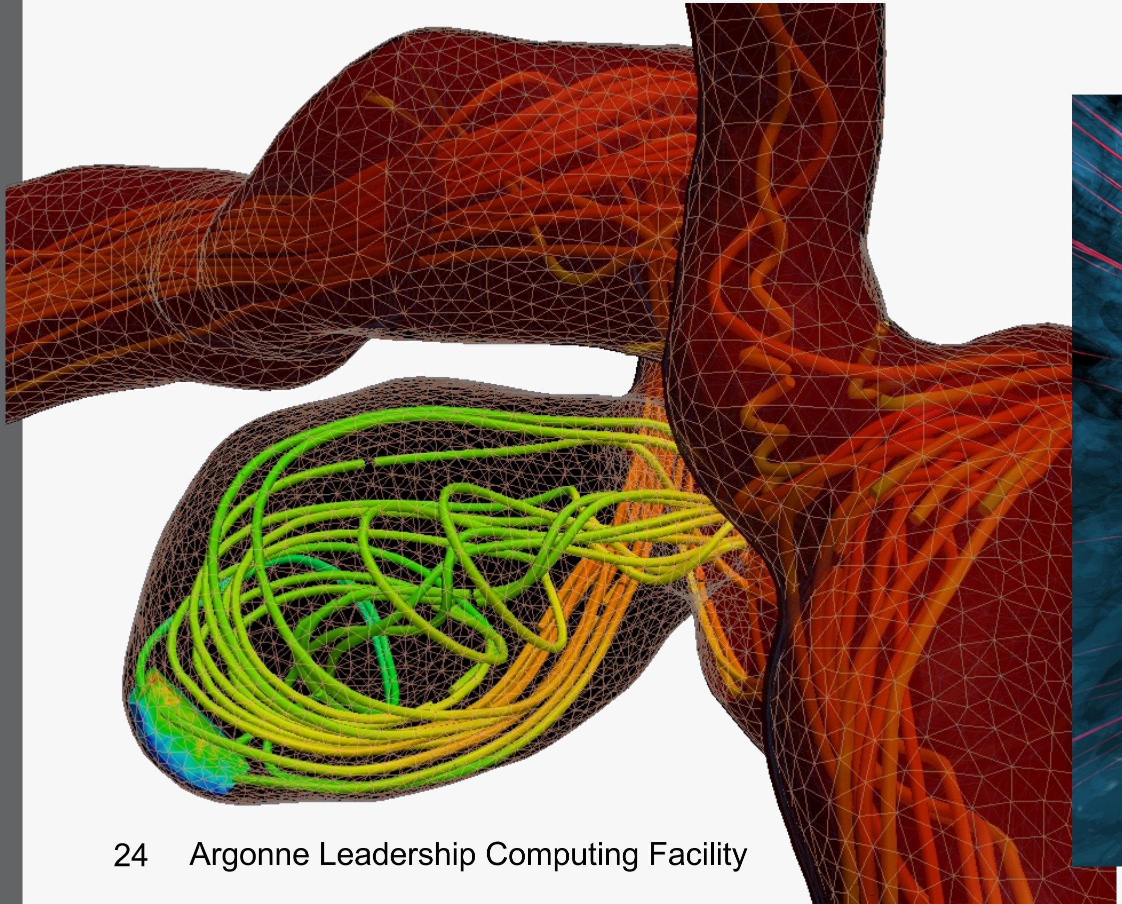
Data Representations: Streamlines

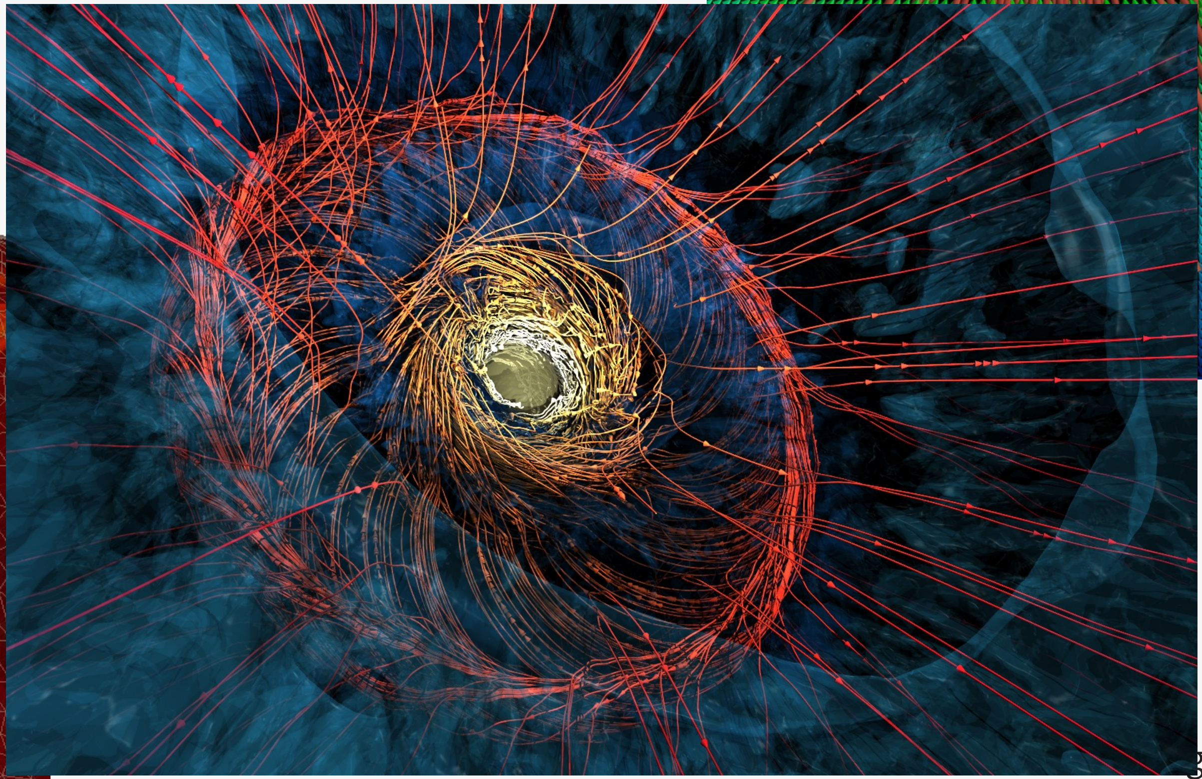
From vector field on a mesh (needs connectivity)

– Show the direction an element will travel in at any point in time.

VisIt & ParaView & vtk good at this





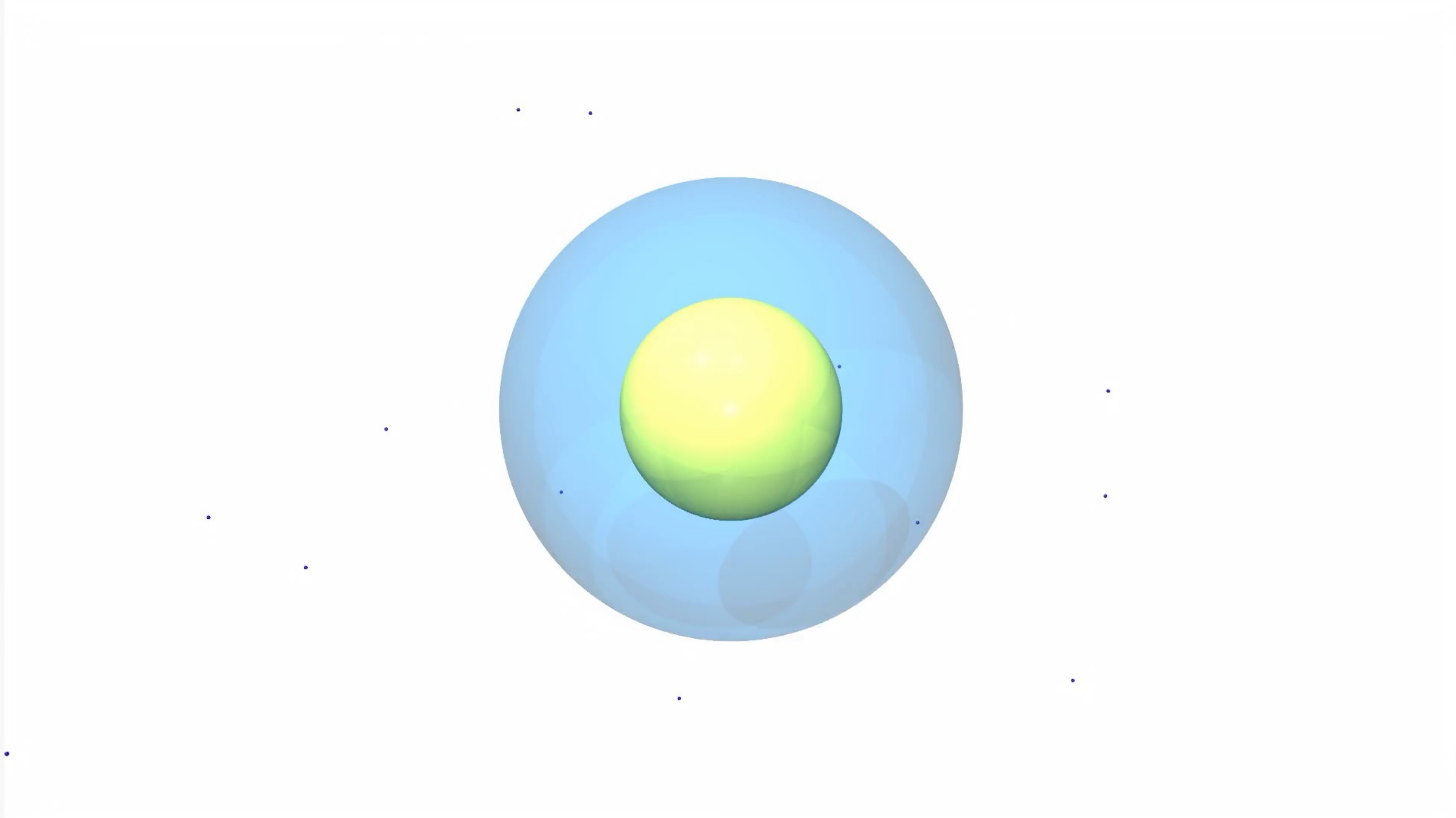


Data Representations: Pathlines

From vector field on a mesh (needs connectivity)

– Trace the path an element will travel over time.

VisIt & ParaView & vtk good at this



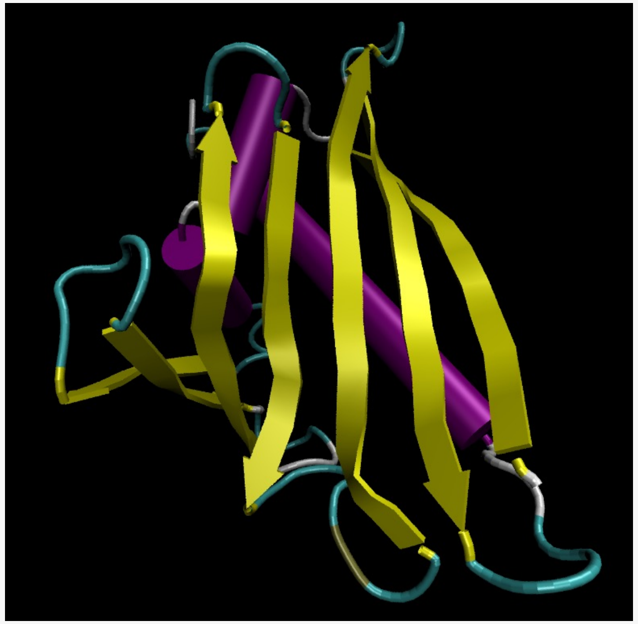
Data Representations: Pathlines

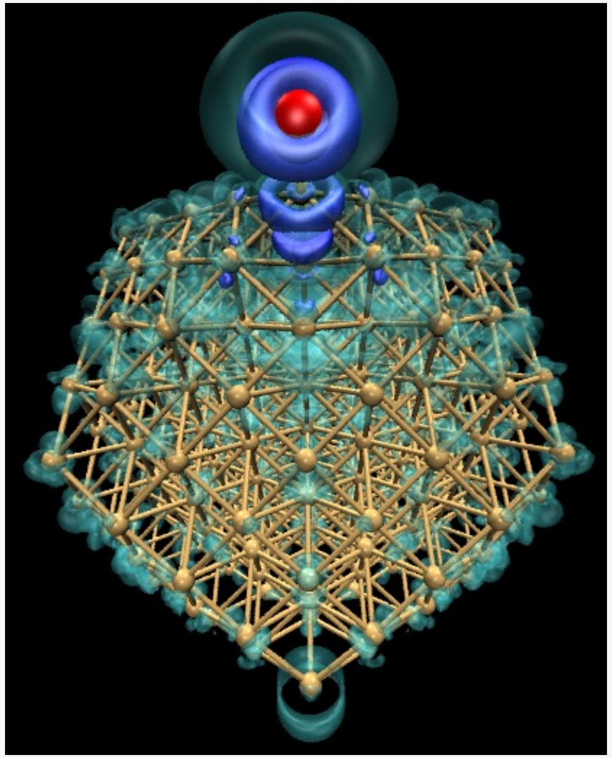
From vector field on a mesh (needs connectivity)

– Trace the path an element will travel over time.

VisIt & ParaView & vtk good at this

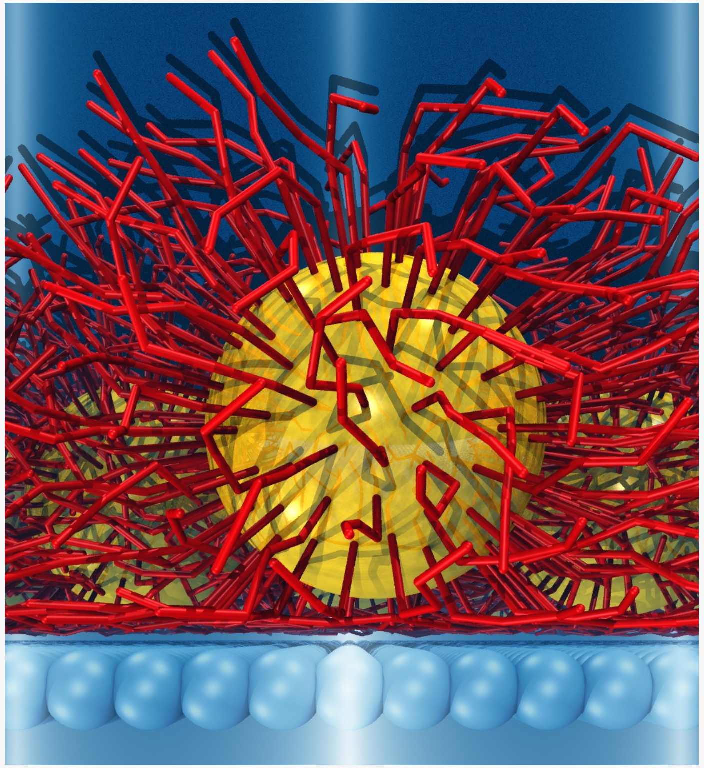
Molecular Dynamics Visualization

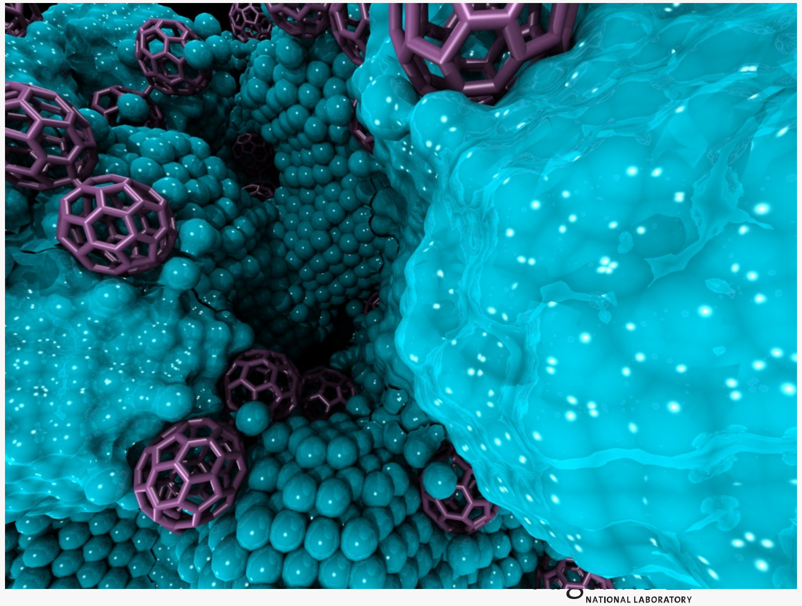




VMD:

- Lots of domain-specific representations
- Many different file formats
- Animation
- Scriptable





VisIt & ParaView:

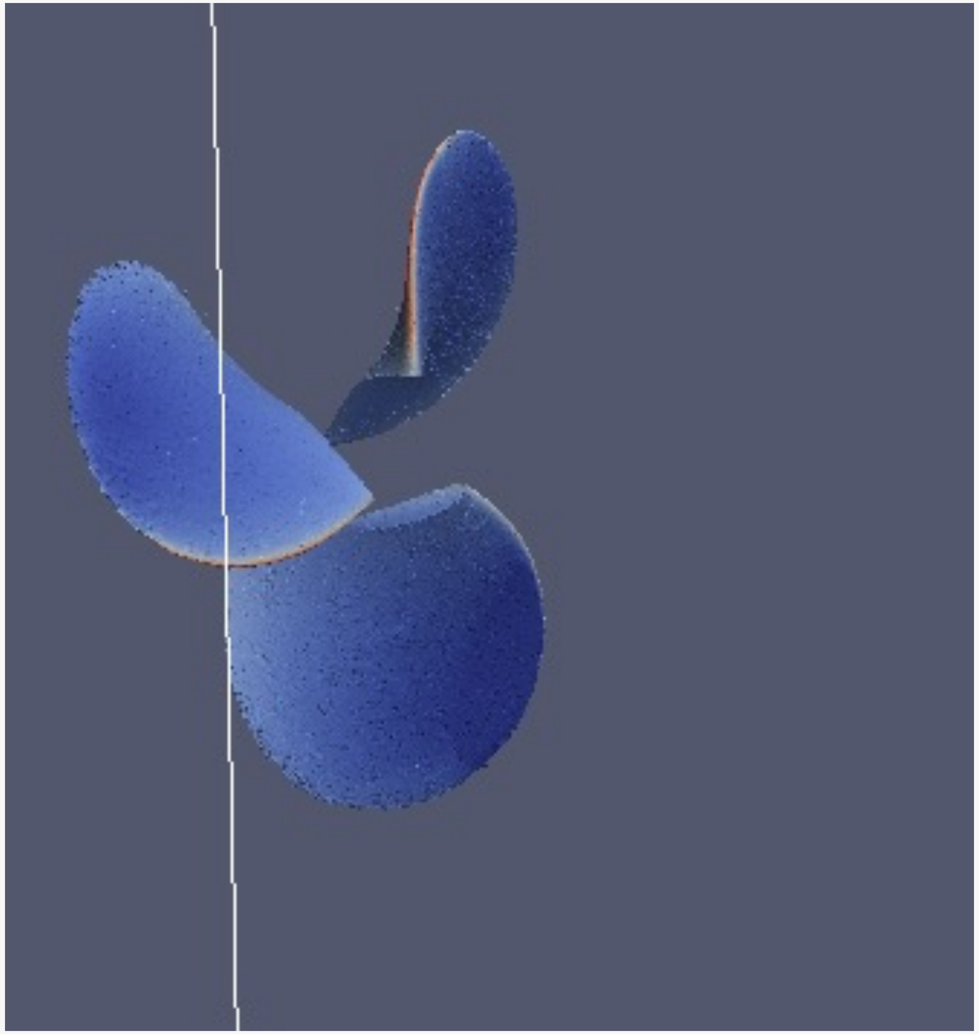
- Limited support for these types of representations, but improving

VTK:

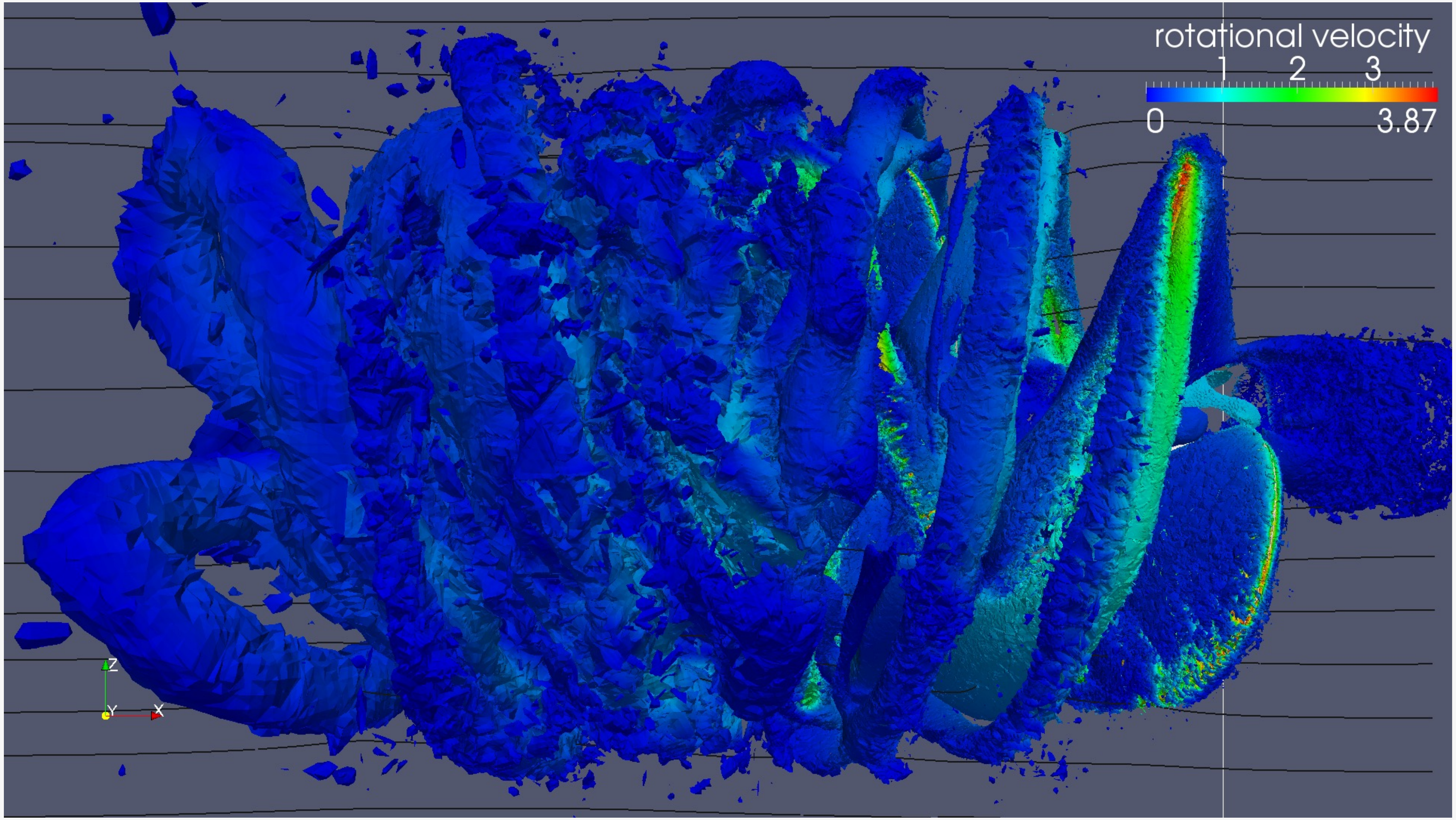
- Anything’s possible if you try hard enough

Visualization for Debugging

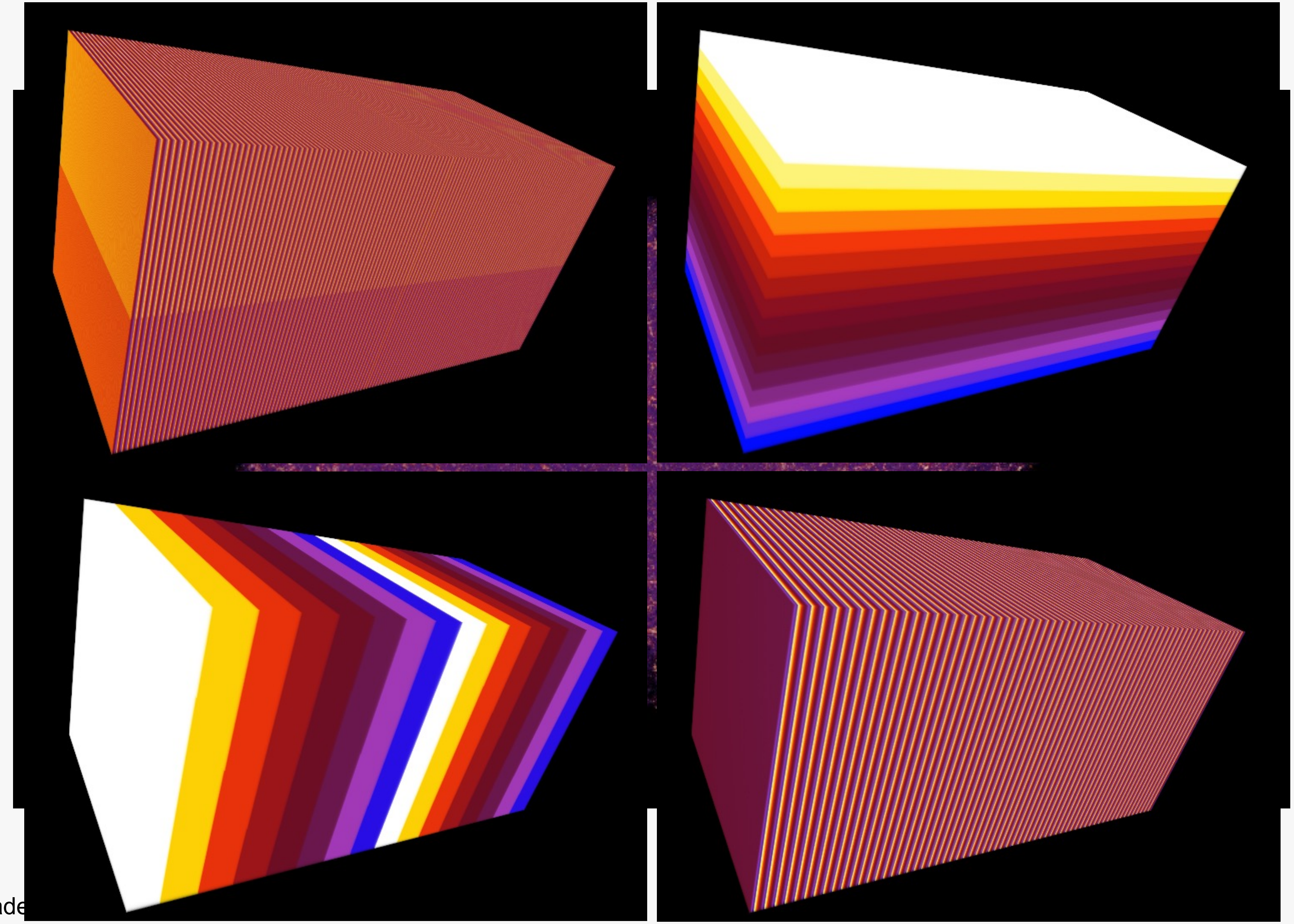
Visualization for Debugging



Visualization for Debugging



Visualization as Diagnostics: Color by Thread ID



In Situ Visualization and Analysis

Five orders of magnitude between compute and I/O capacity on Titan Cray system at ORNL

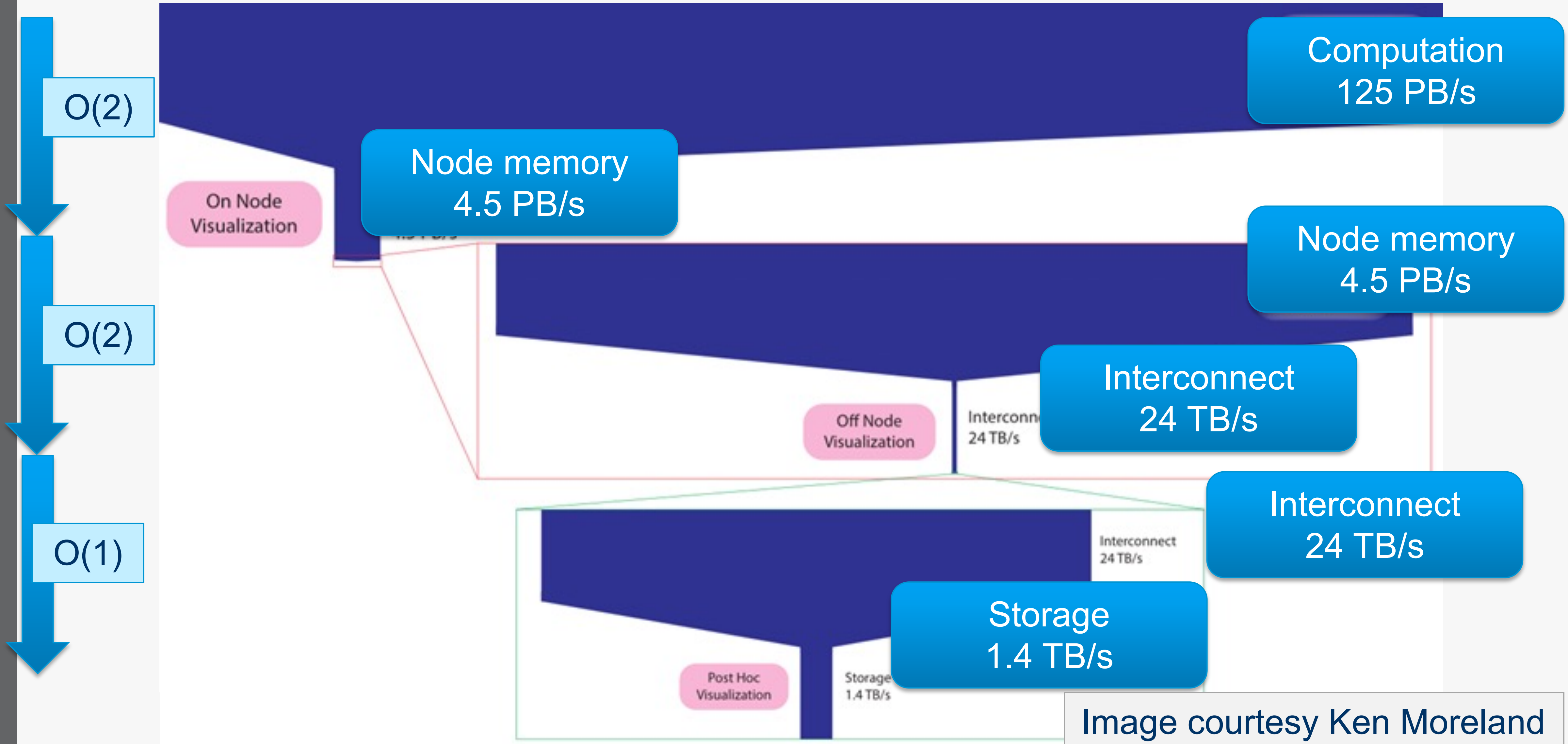


Image courtesy Ken Moreland

In Situ vis and Analysis Problem:

FLOPS to I/O Bottleneck

– Frontier
  - Peak Performance: 1.6 EF
  - Storage: 2-4x Summit's I/O 2.5TB/s. At best 10TB/s
  - 5 orders of magnitude difference

– Aurora
  - Peak Performance: 1.012 EF
  - Storage: 31TB/s
  - 5 orders of magnitude difference

Problem

I/O is too expensive

Scientists cannot save every timestep, and/or resolution

Lost cycles: simulation waits while I/O is happening

Lost discoveries: scientists might miss discoveries

Solution: *In situ* visualization and analysis

What is *IN SITU*

Traditionally visualization and analysis happens post hoc

– aka: Data gets saved to the disk, scientist opens it after the simulation has ended

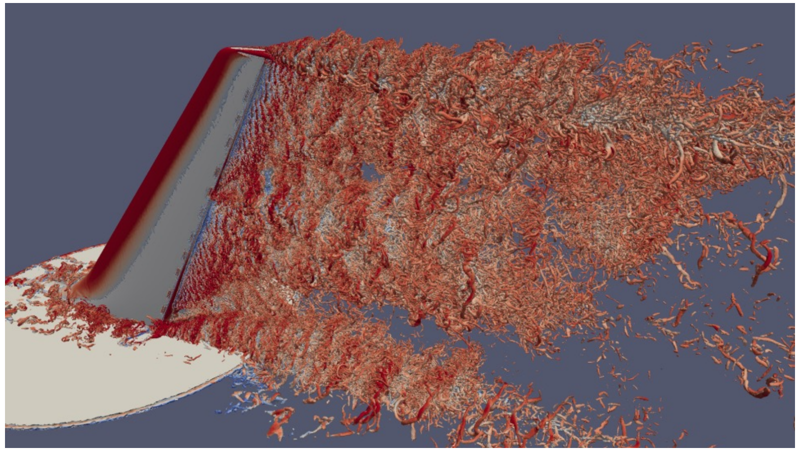
In situ

– Data gets visualized/analyzed **while** in memory.

– If zero-copy used, there is no data movement

– Ideally the data is on the GPU and stays on the GPU

In Situ

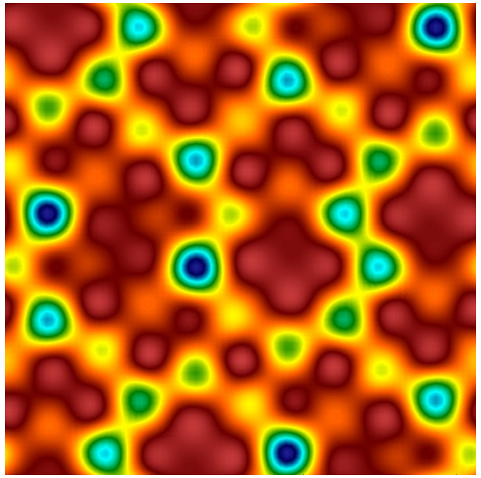


~2014

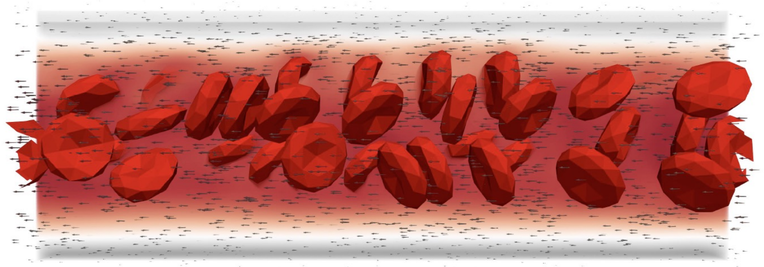
PHASTA, Catalyst,
Ken Jansen

2018

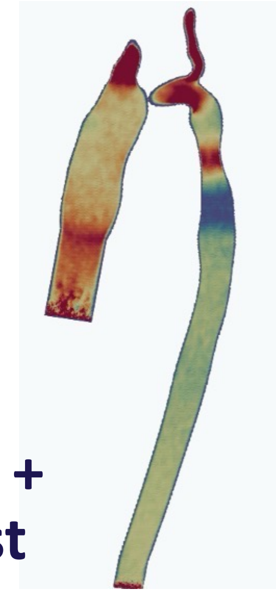
Nek5000,
SENSEI



2021 - 2024

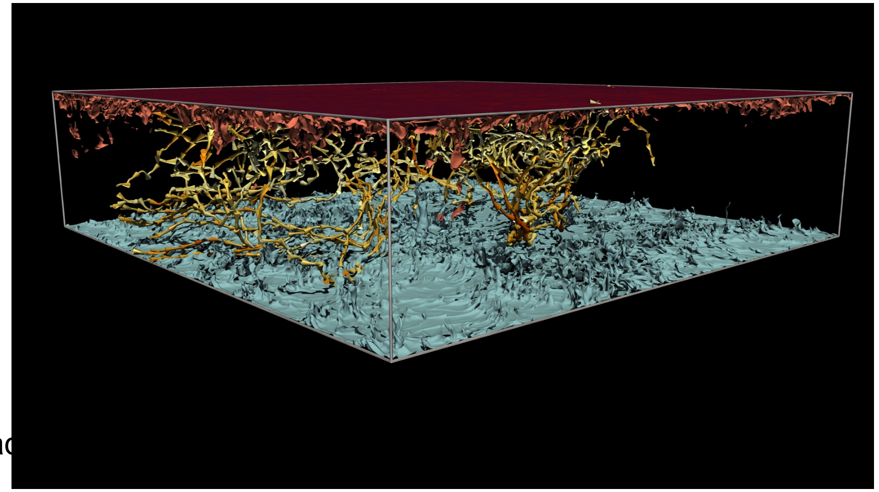


Palabos+LAMMPS,
SENSEI + Catalyst,
bi-directional



2019

SENSEI +
Catalyst



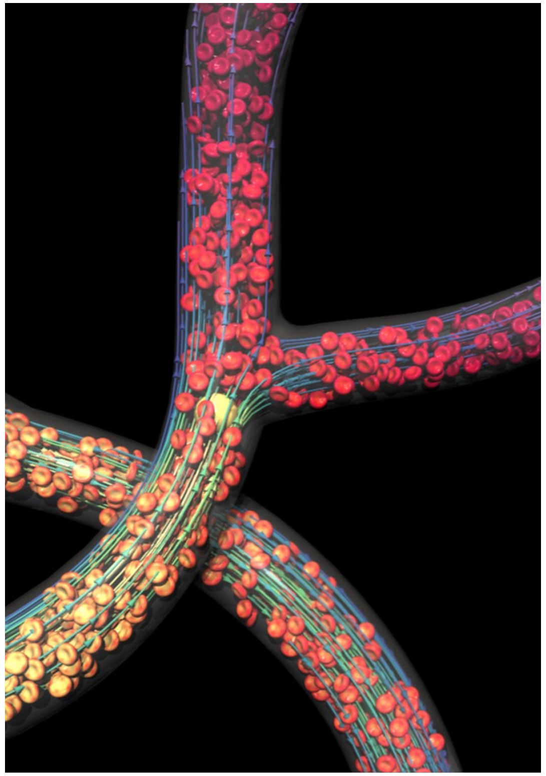
2024

nekRS,
Ascent +
Catalyst

HARVEY

Ascent +
Catalyst

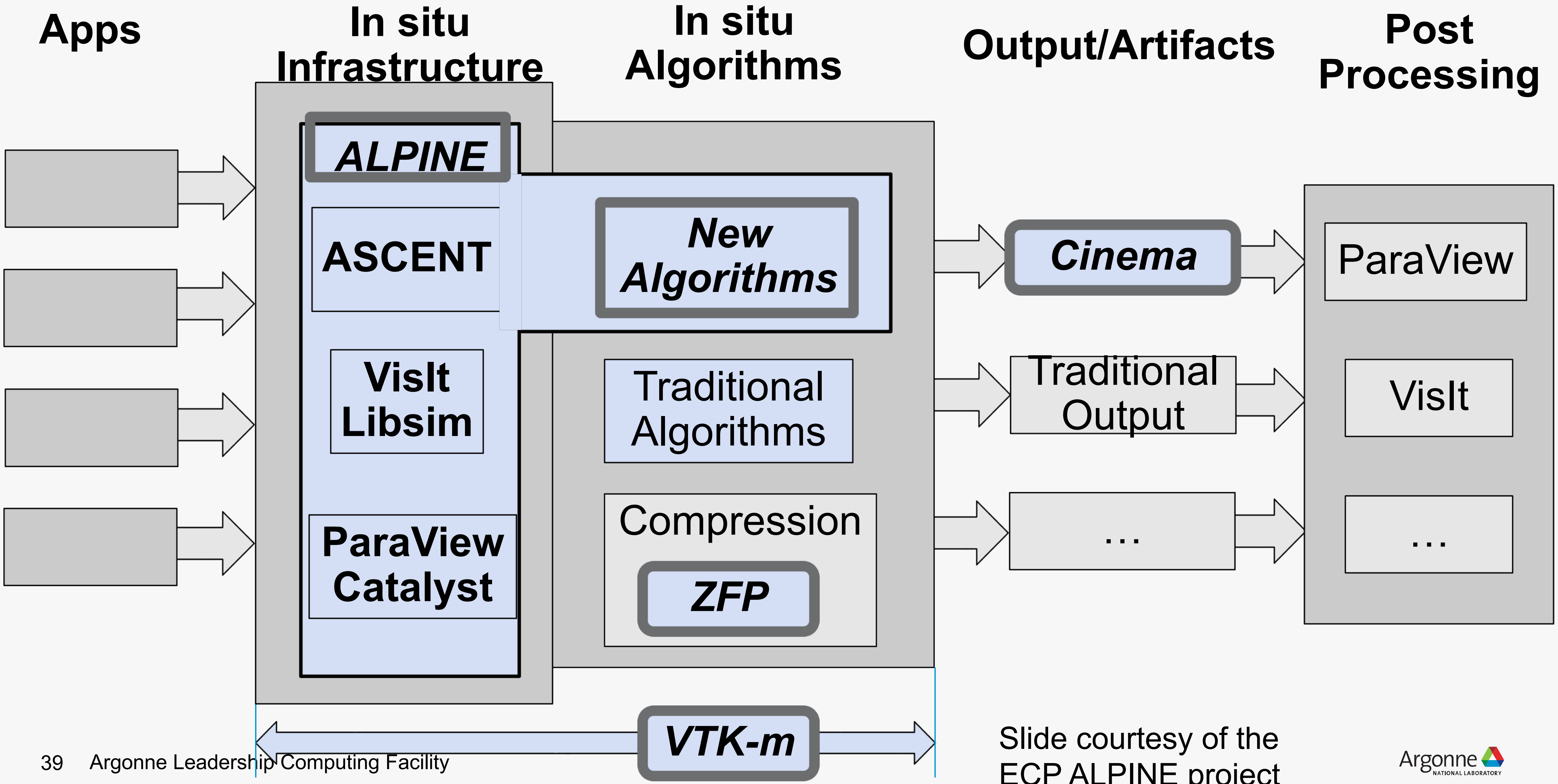
2024



In Situ Frameworks and Infrastructures at ALCF

Name	Description
ASCENT	The Ascent in situ infrastructure is designed for leading-edge supercomputers, and has support for both distributed-memory and shared-memory parallelism.
ParaView/Catalyst	*In situ* use case library, with an adaptable application programming interface (API), that orchestrates the delicate alliance between simulation and analysis and/or visualization tasks
Cinema	Cinema is an innovative way of capturing, storing, and exploring both extreme scale scientific data and experimental data. It is a highly interactive image-based approach to data analysis and visualization that promotes investigation of large scientific datasets.
SmartSim	SmartSim is a software framework that facilitates the convergence of numerical simulations and AI workloads on heterogeneous architectures

Exascale Computing Project
Software Technology Data and Visualization



Slide courtesy of the ECP ALPINE project

Ascent

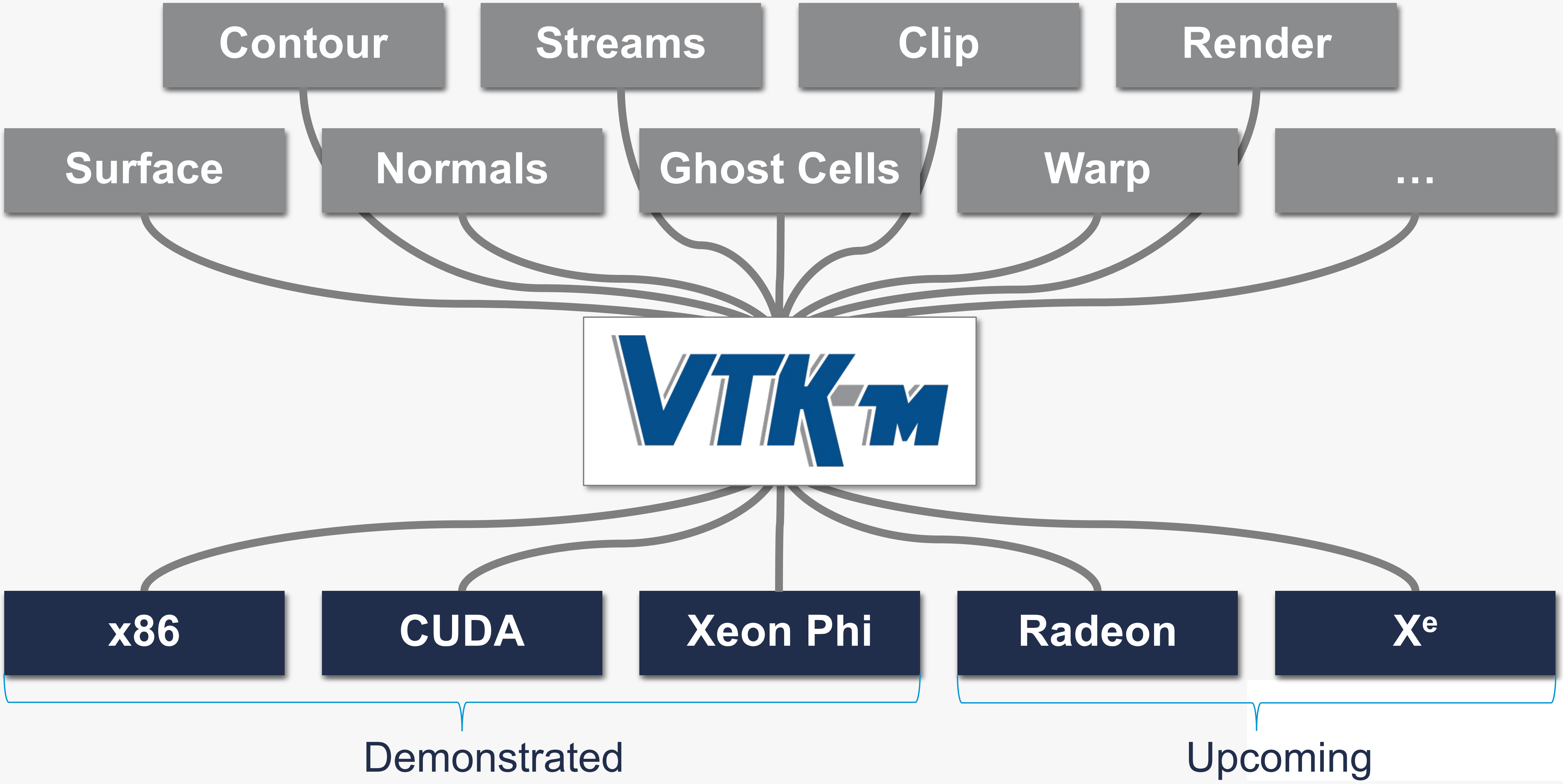
- Flyweight design, minimizes dependencies
- Data model based on Conduit from LLNL
- Vis and analysis algorithms implemented in VTK-m

```
//
// Run Ascent
//

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

Slide courtesy of the Ascent team

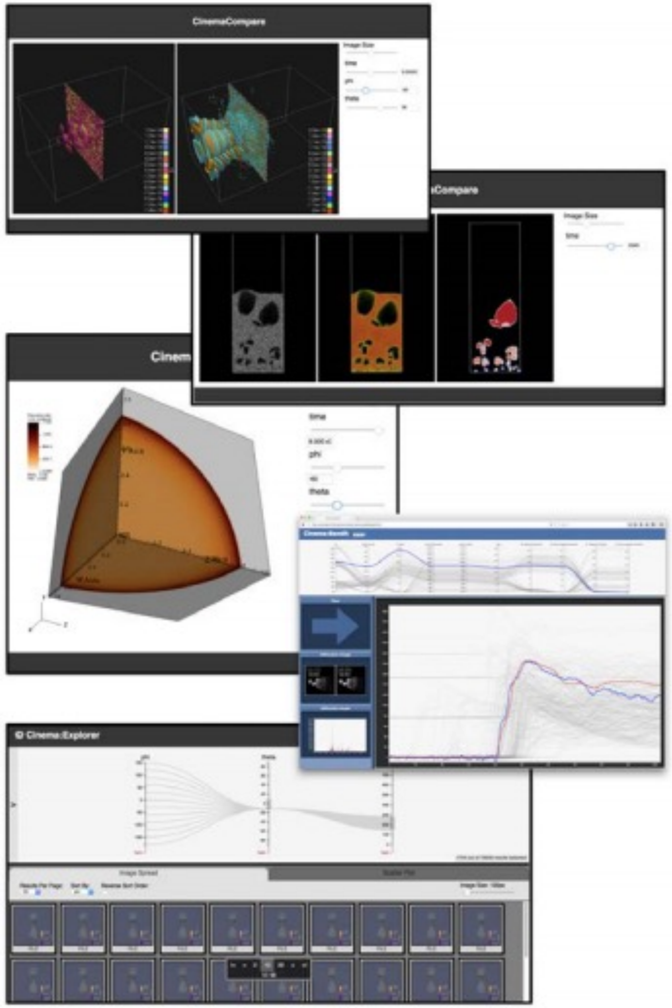
VTK-m's main thrust: a write-once-run-everywhere framework



Slide courtesy of the ECP VTK-m project

What is Cinema?

- **Cinema** is part of an integrated workflow, providing a method of extracting, saving, analyzing or modifying and viewing complex data artifacts from large scale simulations.
  - If you're having difficulty exploring the complex results from your simulation, Cinema can help.
- **The Cinema 'Ecosystem'** is an integrated set of writers, viewers, and algorithms that allow scientists to export, analyze/modify and view Cinema databases.
  - This ecosystem is embodied in widely used tools (**ParaView, VisIt, Ascent**) and the database specification.



Slide courtesy of Los Alamos National Lab and the CINEMA project

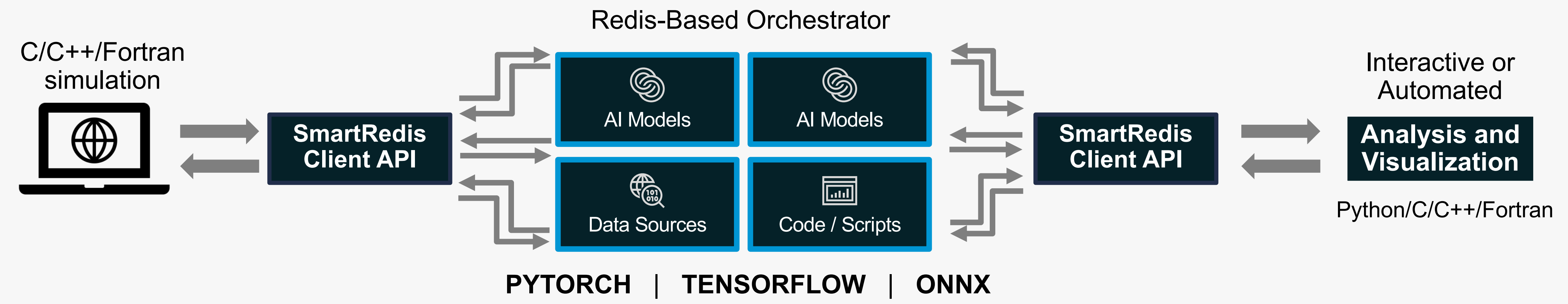
SmartSim Overview

The **SmartSim open-source library** enables scientists, engineers, and researchers to embrace a **“data-in-motion” philosophy** to accelerate the convergence of **AI/data science** techniques and **HPC simulations**

SmartSim enables **simulations** to be used as **engines** within a system, **producing data,** consumed by other services enable **new applications**

- Embed **machine learning** training and inference with **existing** in Fortran/C/C++ **simulations**
- **Communicate** data **between** C, C++, Fortran, and Python **applications**
- Analyze and visualize **data streamed** from **HPC applications** while they are **running**
- **Launch**, **configure**, and **coordinate** complex simulation, analysis, and visualization **workflows**

All of these can be done without touching the filesystem, i.e. **data-in-motion**



slide courtesy of the HPE SmartSim team

Bridging Gaps in Simulation Analysis through a General-Purpose, Bidirectional Steering Interface

Background

ASCENT

Easy-to-use flyweight in situ visualization and analysis library

Uses Conduit for describing data

Supports zero-copy GPU-GPU

Uses VTK-m as a toolkit for scientific visualization algorithms

Has a python and jupyter interface

Background
Bidirectional steering

Libsim

Drawbacks
- Limited steering functionality
- Do not allow sending back modified data
- Require changes in the code of the GUI version, thus making development more difficult

Motivation

Interviews with scientists

7 scientists at Argonne National Laboratory

- Computational Fluid Dynamics
- Computational Physics
- Water Resource Engineering
- Environmental Science
- Computational Biology
- Fluid Structure Interactions

Motivation

Interview Insights

Steering

- Change parameters (velocity, temperature, pressure, etc.)
- Adjust resolution
- Add/modify geometry
- Modify simulation data

Exploration and Discovery

- Checking on simulation status
- Interactively exploring data and changing filters/cameras
- Trial and error

External Notifications

- Be alerted when things have gone wrong
- Manually investigate interesting phenomena

Contributions

General Purpose Steering Interface

Build upon Ascent

- Lightweight.
- Scalable.
- Convenient instrumentation with existing simulation codes.

Let users define their own steering behaviors

- Function callbacks give users total control over a simulation.

Bidirectional interactive interface

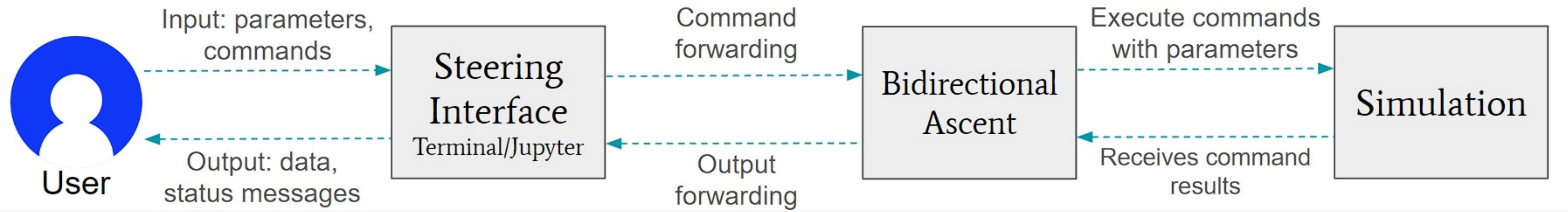
- Use Ascent’s existing remote Jupyter notebook capabilities.

BONUS CONTRIBUTION

Shell commands

```yaml
-
  action: "add_commands"
  commands:
    c1:
      params:
        shell_command: echo "Unstable!" | mail -s "Unstable Simulation" $email
        mpi_behavior: "root"
```

Can run on the root node, or on all nodes


Input: parameters, commands
Output: data, status messages
User
Steering Interface
Terminal/Jupyter
Command forwarding
Output forwarding
Bidirectional Ascent
Execute commands with parameters
Receives command results
Simulation

Comparison of In Situ Frameworks by Steering Capability

Steering Type	Catalyst	SENSEI	Ascent	Ascent w/ bidirectional
Simulation Variables	Yes	Partial	Partial	Yes
Internal Commands	No	No	No	Yes
External Commands	No	No	No	Yes
Modifying data	No	No	No	Yes

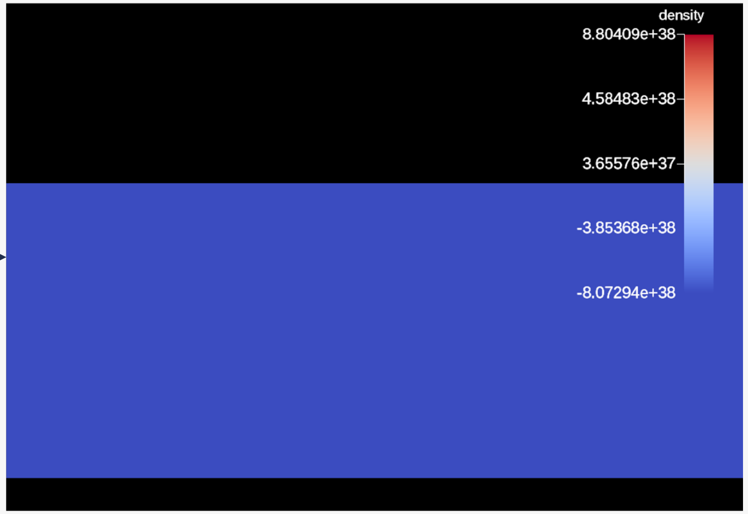
Use case

Instability

Input data and initial conditions

Lattice Boltzmann Simulation

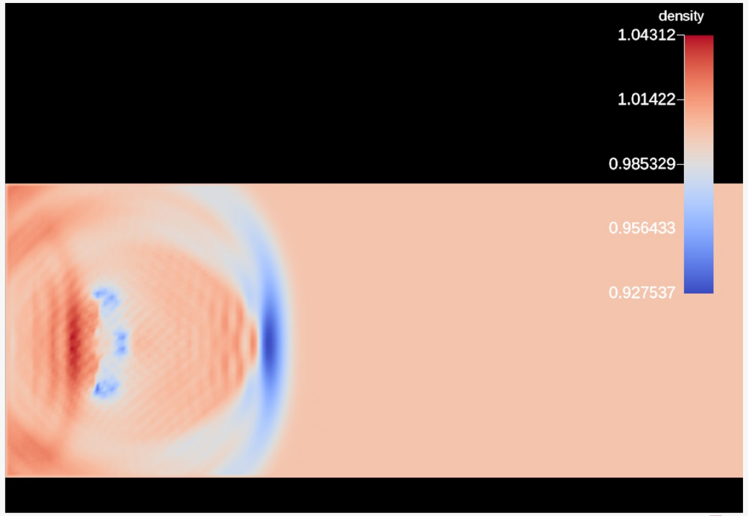
Resolution too low



Jupyter Notebook

User

Increase resolution



USE case

Workflow

1. Detect instability: simulation
2. Pause simulation: Ascent
3. Notify the user: Ascent

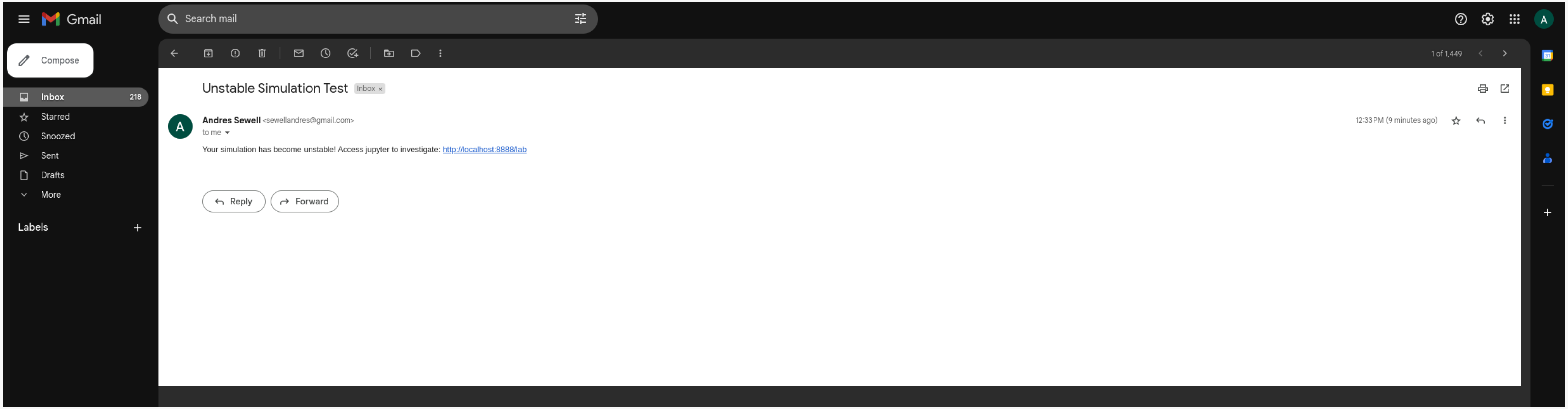
```yaml
-
  action: "add_commands"
  commands:
    c1:
      params:
        callback: "setStability"
        mpi_behavior: "all"
-
  action: "add_triggers"
  triggers:
    t1:
      params:
        callback: "isStable"
        actions_file : "stable_actions.yaml"
    t2:
      params:
        callback: "isUnstable"
        actions_file : "unstable_actions.yaml"
```

```yaml
-
  action: "add_commands"
  commands:
    c1:
      params:
        shell_command: echo "Unstable!" | mail -s "Unstable Simulation" $email
        mpi_behavior: "root"
```

Example Ascent actions

USE CASE

Workflow



Argonne NATIONAL LABORATORY

USE CASE

Workflow

4. Access the steering interface:
5. Restore checkpoint:
6. Increase number of timesteps:
7. Resume the simulation:

```
# Connect to our simulation instance via Ascent, pausing it
%connect
```

```
# This is a function callback that takes no parameters and returns no output
execute_callback("restoreStableState", conduit.Node(), conduit.Node())
```

```
# This is a function callback that takes one parameter and returns no output
params = conduit.Node()
params["timesteps"] = 1000
execute_callback("setTimeSteps", params, conduit.Node())
```

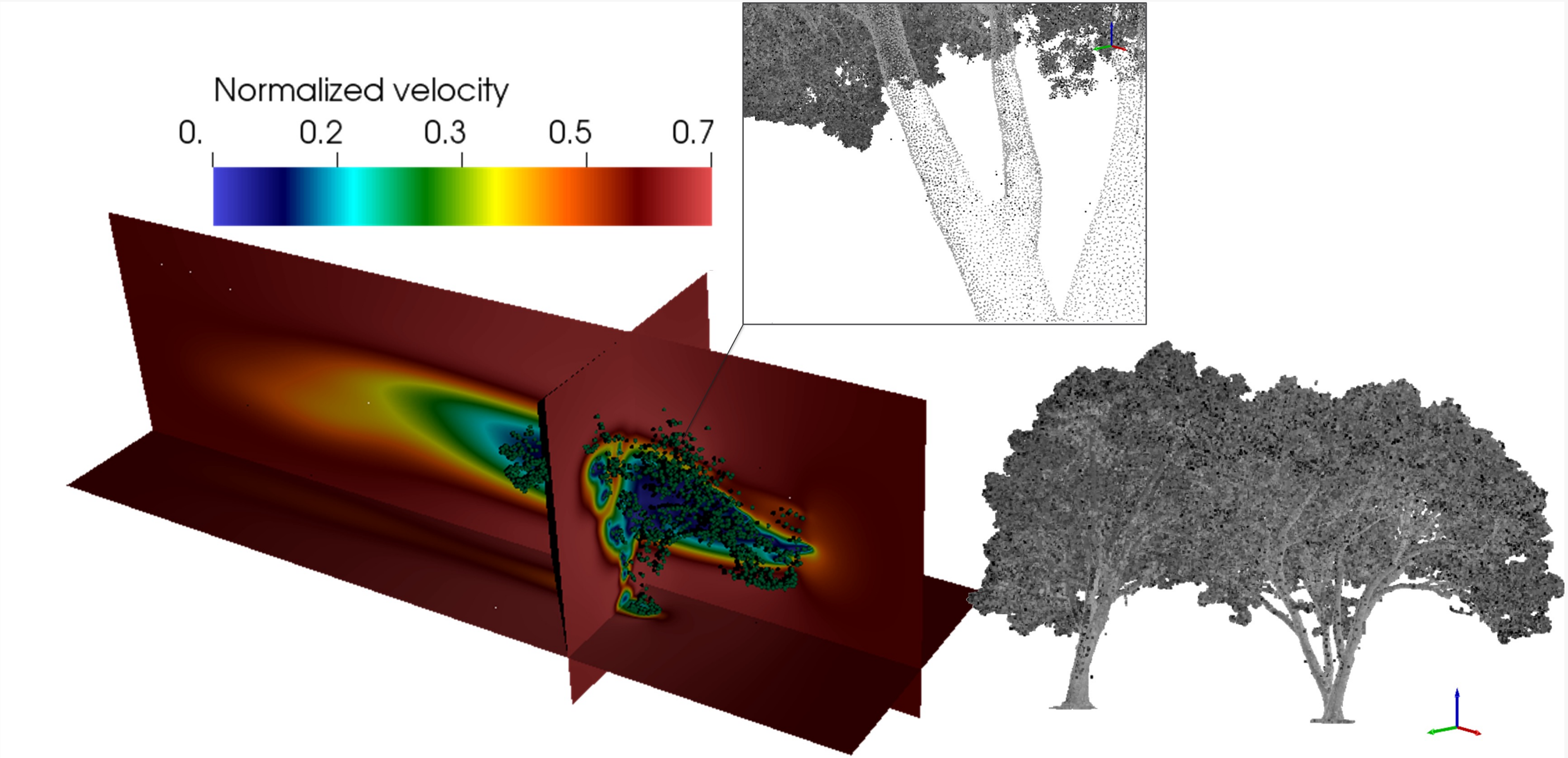
```
# This is a function callback that takes no parameters and does return an output
output = conduit.Node()
execute_callback("setTimeSteps", conduit.Node(), output)
print(output)
```

```
# Disconnect from the simulation, resuming it
%disconnect
```

Example Jupyter notebook

56

Simulation of wind across a city



ISAV24: Bridging Gaps in Simulation Analysis through a General Purpose, Bidirectional Steering Interface with Ascent

Simulation of wind across a city

Use case

- LIDAR of trees in a city
- Simulating wind and effect of trees

Problem

- Case takes 20-30 mins to load
- Scientist examines result and then modifies lidar resolution
- This is a trial and error scenario. They don’t know a priori what numbers they should select

Simulation of wind across a city

Bidirectional solution

– Load case once (20-30 mins)
– Real-time bidirectional steering
    • Sees the results immediately

Time without steering (rough estimates)

– 20 times trial and error x 30 minutes = 600 compute minutes
– Scientist need to wait in the queue again

Time with steering (rough estimates)

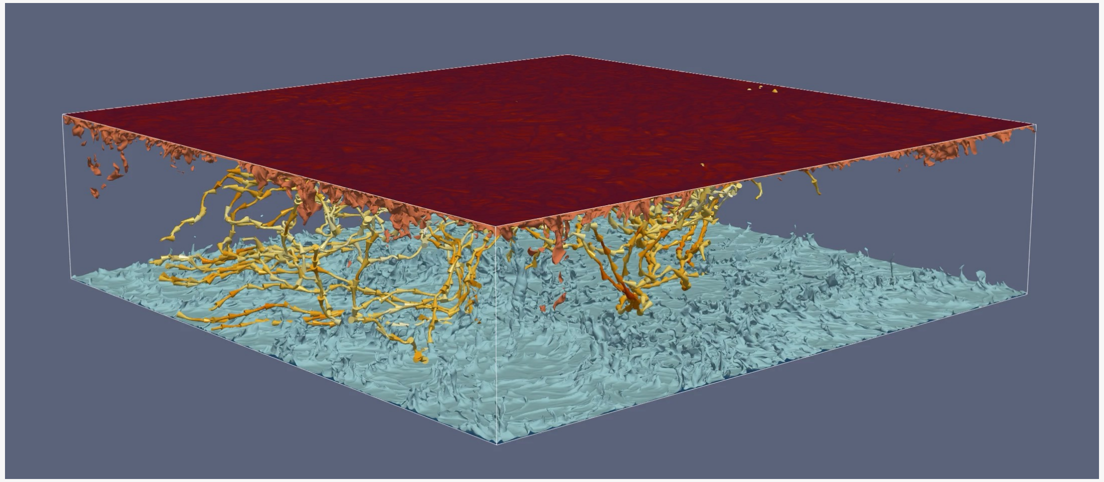
– Load case once x 30 minutes = 30 compute minutes
– 20 times trial and error x 2 minutes = 40 compute minutes
– Total: 70 compute minutes

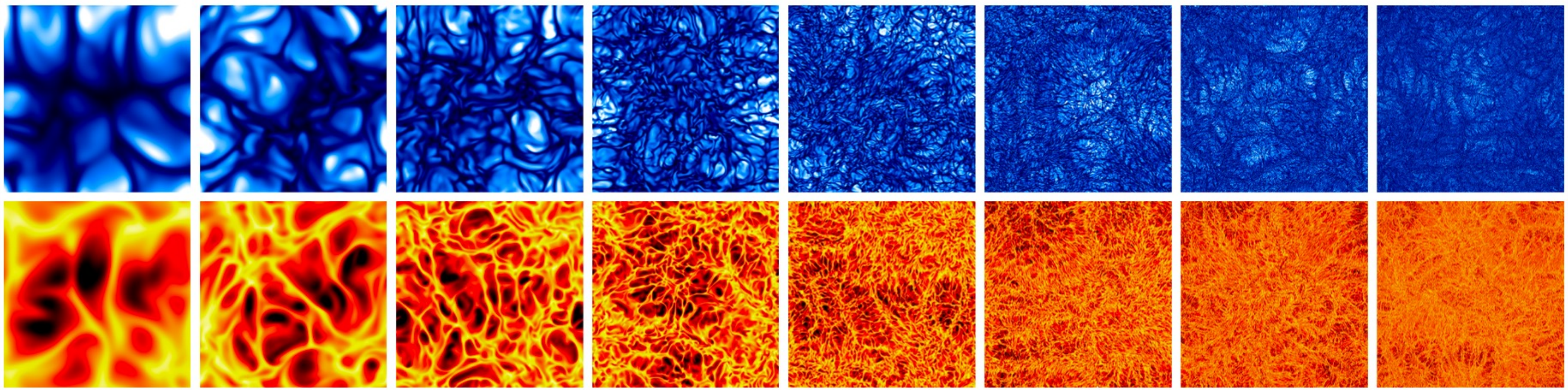
A peak into the (not so distant) future

LARGE SCALE ASCENT RUNS

Ascent

- GPU – GPU
- More realistic rendering
- ParaView integration





QUESTIONS?

Joe Insley
insley@anl.gov

Silvio Rizzi
srizzi@anl.gov

Victor Mateevitsi
vmateevitsi@anl.gov