SAND2023-05390C

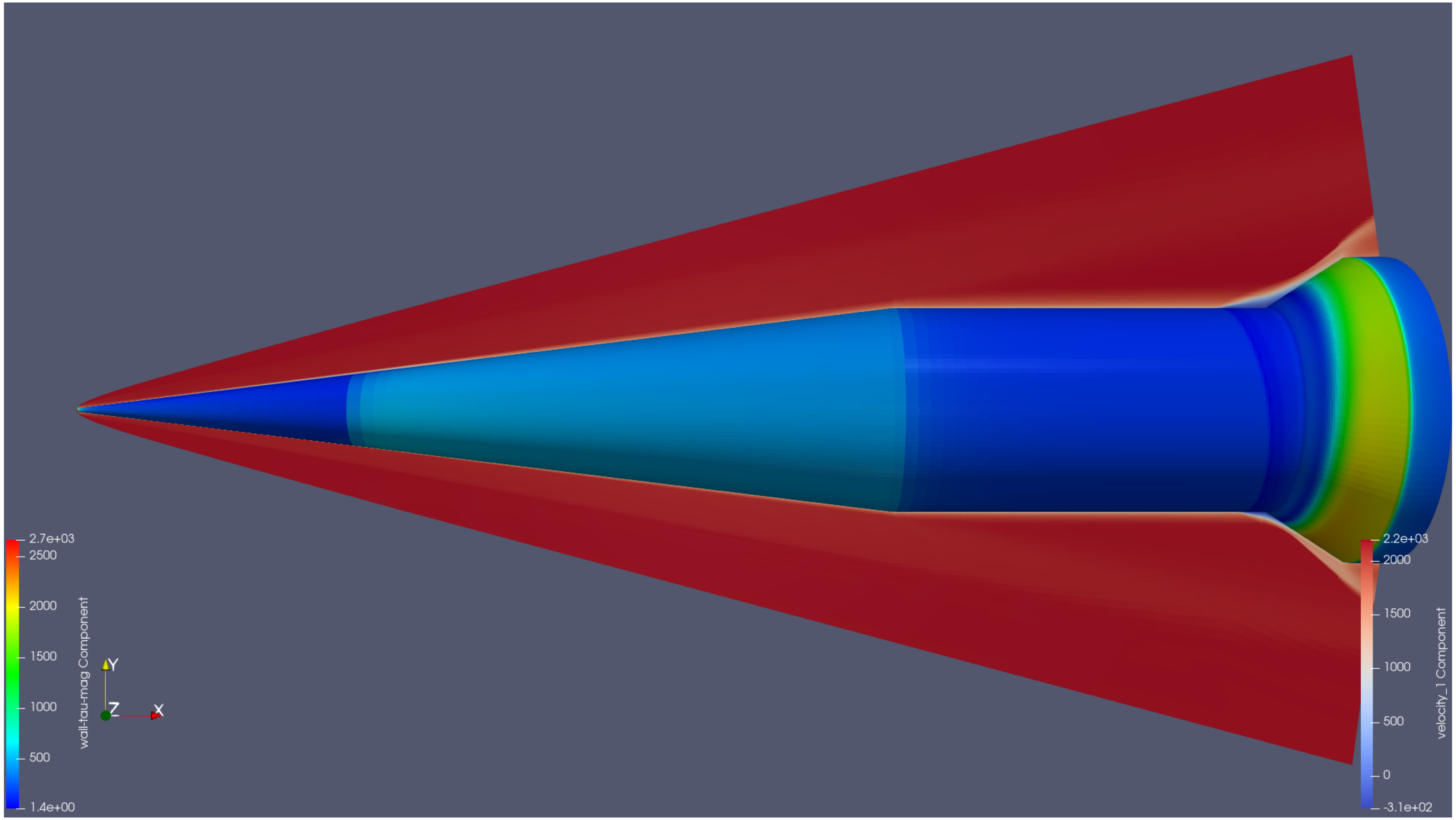
Sandia National Laboratories

Exceptional service in the national interest

DEVOPS FOR CATALYST IN SITU VISUALIZATION AND ANALYSIS AT SANDIA NATIONAL LABORATORIES

Jeff Mauldin and Thomas Otahal

NLIT Summit 2023, Milwaukee, WI


2.7e+03
2500
2000
1500
1000
500
1.4e+00
wall-tau-mag Component
Y
Z
X
2.2e+03
2000
1500
1000
500
0
-3.1e+02
velocity_1 Component

My managers 4+ years ago: “We think you should be the technical lead for in situ visualization at SNL”

Me: “You mean take the Kitware Catalyst product, based on the tens of thousands of lines of ParaView code, developed over decades, and integrate with an SNL simulation code, developed over decades, so they can do in situ visualization?”

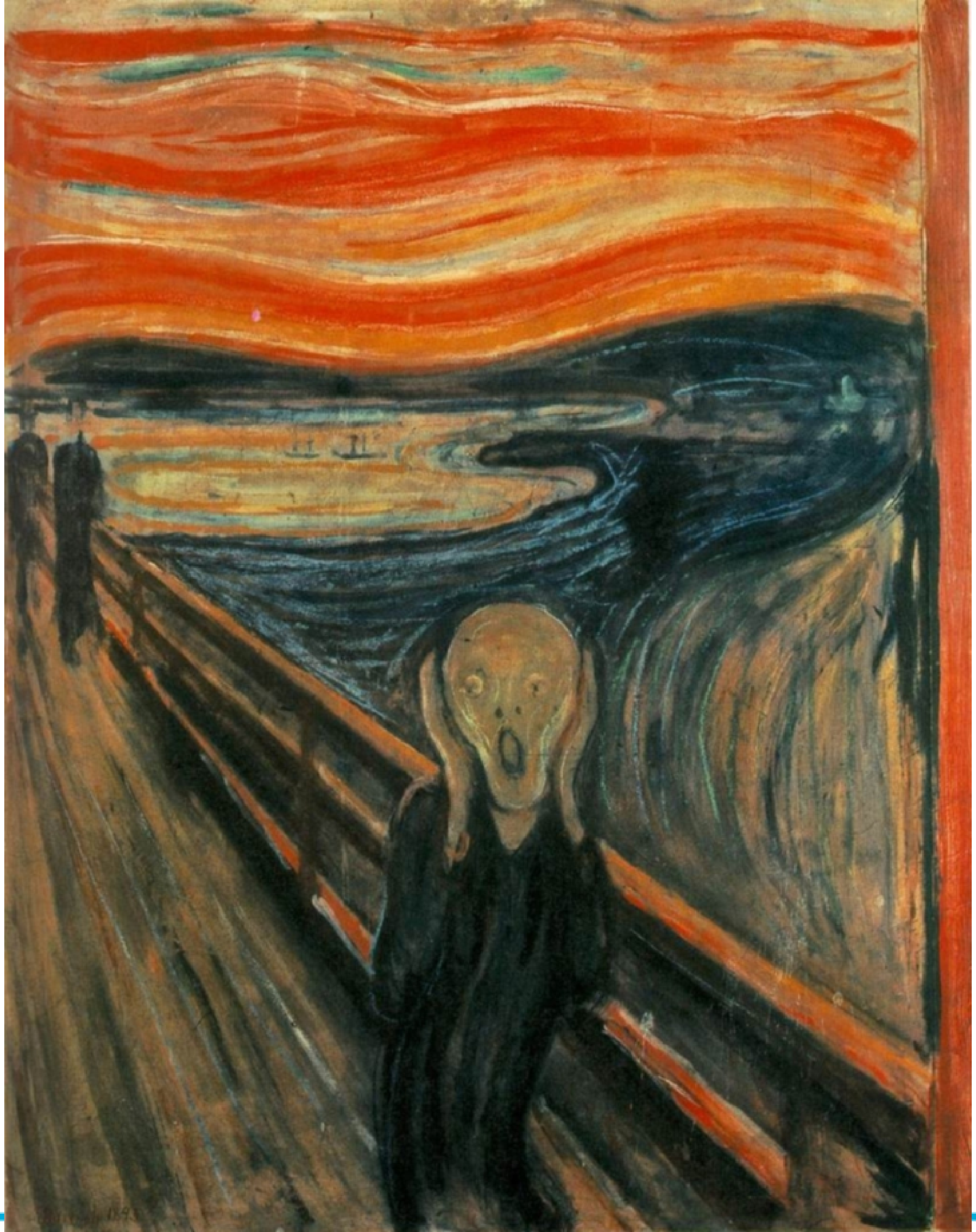
Managers: “Close. We want you to integrate with *several* SNL simulation codes. Maybe all of them!”

Me: “Ah.”

Managers: “You won’t have to do it yourself. Kitware makes Catalyst.   You can share the work at SNL with team members.”

Me: “Good. How much manpower will we have?”

Managers: “Plenty! I think we can give you half your time and a quarter of somebody else’s! A whole 0.75 FTE!”



INTRODUCTION

- Goal: Provide in situ visualization and analysis capabilities in various simulation codes, on various HPC platforms, to Analysts at Sandia National Laboratories (SNL), and also to clients outside SNL who use those codes.
- This presentation focuses on the DevOps aspects of what we are doing to try to succeed in this goal.
- Most of the work has been done by Jeff Mauldin and Tom Otahal, Members of the technical staff at SNL (More acknowledgements at the end!)

IN SITU VISUALIZATION AND PARAVIEW CATALYST

In situ visualization computes images and data extracts by directly coupling to the simulation code. This direct coupling gives the in situ software library access to simulation data structures and data fields as they are calculated. This confers several benefits over traditional post-processing workflows of simulation data:

- Reduced I/O costs since visualization images and data extracts are much smaller than full mesh output
- More frequent I/O is possible leading to higher frequency outputs
- Reduced disk space required to store output
- Potential reduction in end to end workflow length

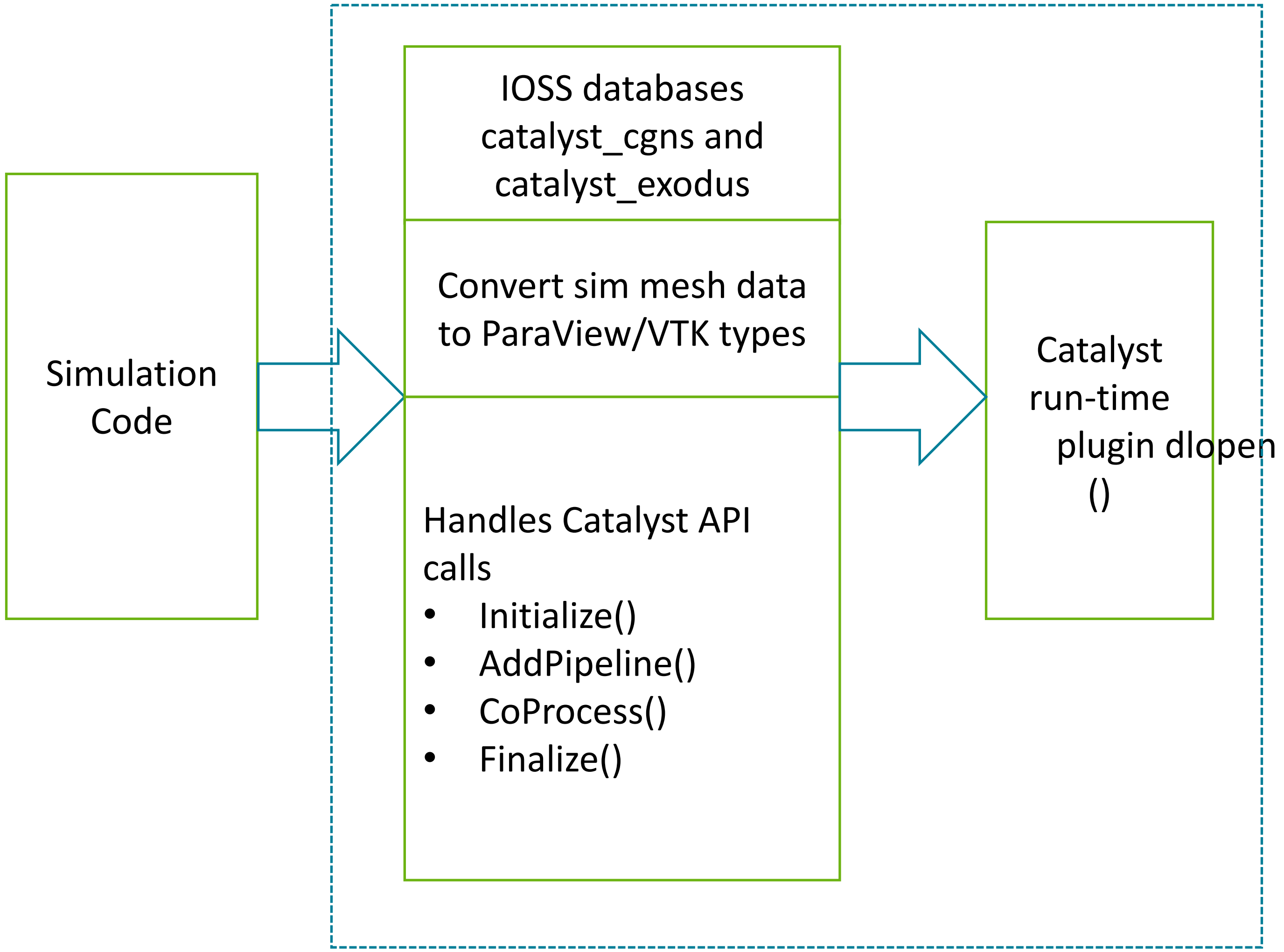
ParaView Catalyst is an in situ library version of the ParaView visualization software (Kitware Inc.). Simulation codes can access the visualization and analysis capabilities of ParaView at run-time by linking to ParaView Catalyst.

Catalyst API version 1.0 legacy documentation
https://www.paraview.org/files/catalyst/docs/ParaViewCatalystUsersGuide_v2.pdf

Catalyst API version 2.0 documentation
https://catalyst-in-situ.readthedocs.io/en/latest/introduction.html

CATALYST INTEGRATION WITH SEACAS IOSS

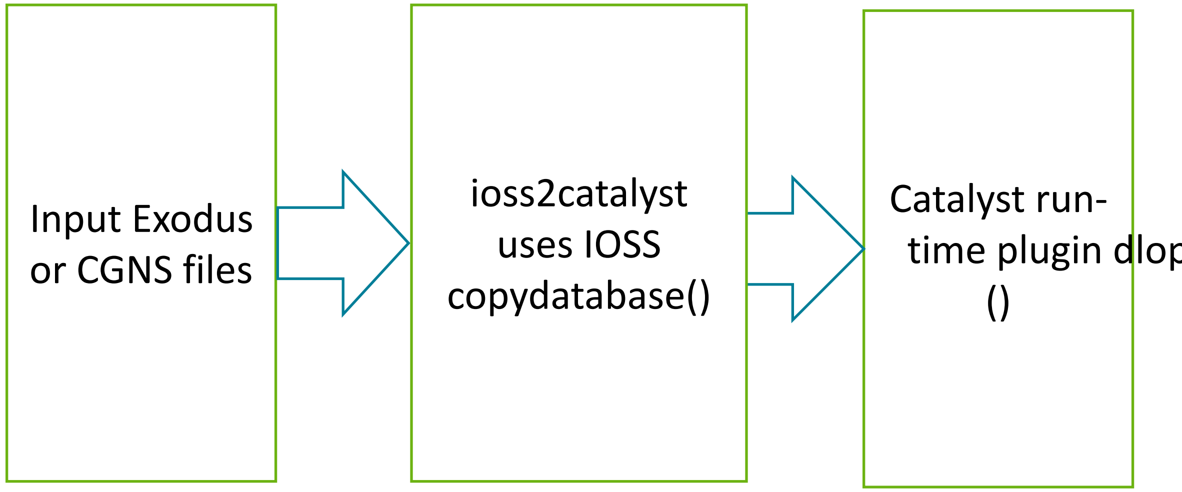
- SEACAS (Sandia Engineering Analysis Code Access System) contains IOSS (I/O Subsystem), which is a parallel I/O framework used by Sandia codes to read and write Exodus, CGNS, and other file formats
- Created catalyst_cgns and catalyst_exodus IOSS databases to output CGNS and Exodus data to Catalyst
- Simulation codes can access Catalyst through existing code paths to write file data through IOSS
- Catalyst run-time plugin loaded with dlopen() allows separate build chains for simulation codes and Catalyst (with same MPI and compilers)



Simulation Code
IOSS databases catalyst_cgns and catalyst_exodus
Convert sim mesh data to ParaView/VTK types
Handles Catalyst API calls
• Initialize()
• AddPipeline()
• CoProcess()
• Finalize()
Catalyst run-time plugin dlopen ()

CATALYST TEST SUITE IN IOSS

- Catalyst test suite uses program ioss2catalyst, which can be used from a test fixture or from the command line
- The ioss2catalyst program uses the IOSS function copydatabase() to read CGNS and Exodus files in parallel and send them through the appropriate IOSS Catalyst database
- The test suite does not require linking to the Sierra or SPARC simulation codes, since the ioss2catalyst program acts a stand-in for these programs
- Catalyst test suite contains 42 tests that exercise code paths for CGNS and Exodus simulation data conversion to Catalyst, Catalyst control parameters, usage logging, and Phactori Sierra input deck processing
- If the Catalyst test suite builds on runs on an HPC platform, users can have a high degree of confidence it will run from a calling simulation code


Input Exodus or CGNS files
ioss2catalyst uses IOSS copydatabase()
Catalyst run-time plugin dlop ()

INTEGRATING USER CONTROL OF CATALYST INTO SIMULATION ENGINES

Code level integration is via the IOSS/Catalyst run time plugin using dlope ()

Analyst level integration requires communication from the analyst to the Catalyst capability

- Analyst must have a way to create python for Catalyst to use
- Catalyst control syntax must be available to analyst in simulation input deck so that script and any other settings can be identified

This analyst level integration needs to be managed in the devops process

SIERRA ANALYST LEVEL INTEGRATION

- Works with Sierra SM (Solid Mechanics), Sierra heat transfer code, and Sierra fluid dynamics code
- Sierra input deck syntax is home grown and described with a set of xml files. Some newer Sierra codes are moving away from this syntax towards, e.g. YAML.
- Prior to any Catalyst integration Sierra input decks had a “results output” block which had the option of specifying an output database type of “exodusII” (for full multiblock volume mesh output)
- We added an output database type of “catalyst” which sends the mesh out to catalyst via the plugin rather than to the filesystem

CATALYST SCRIPT CREATION IN SIERRA

Initial method for creating Catalyst scripts was by using the ParaView gui and the python capture capability to create scripts (still available)

This Catalyst script creation capability from the ParaView GUI has become steadily more robust and easier to use in the ensuing years but still has some disadvantages

We integrated a Catalyst scripting language into the Sierra input deck

- Allows us to exhaustively test available capability (in contrast to python)
- Analyst do not have to do a script construction step involving ParaView
- Increased likelihood of adoption (simple behaviors are easy to create)

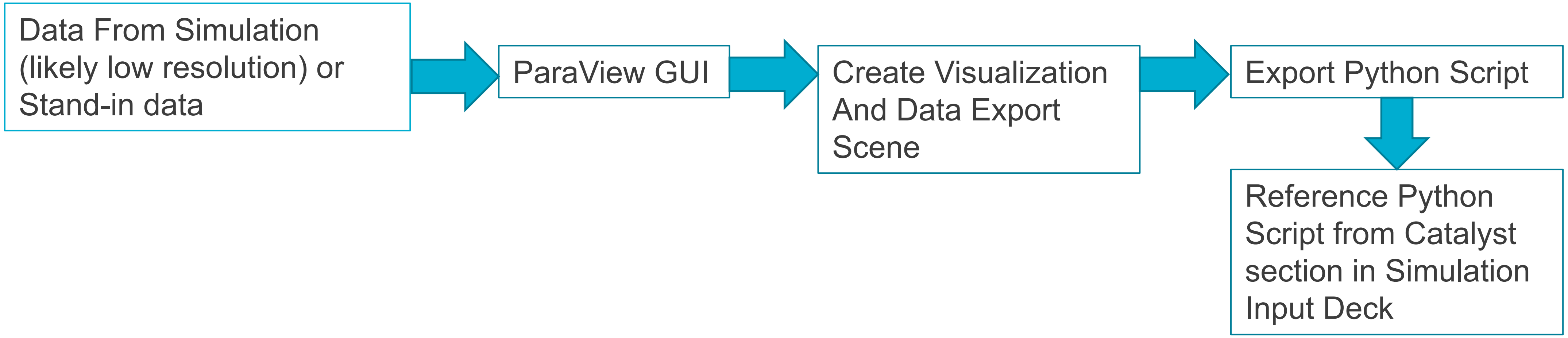
Scripting language named “Phactori” (backronym of “Paraview Higher level of AbstraCTiOn scRipting Interface”)

CATALYST IN SIERRA

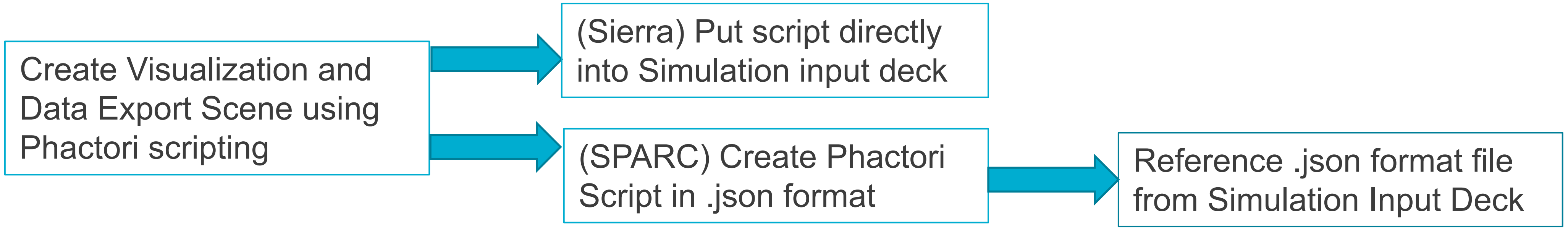
```
begin results output catalyst_example_output_results_block
       database type = catalyst
       At Step 0, Increment = 200
       element variables = von_mises as vonmises
       nodal Variables   = displacement as displ
       begin catalyst
         begin camera closeupCam
           look direction = -1 -1 -1
           look at relative distance = 0.5
         end camera closeupCam
         begin clip myClip
           relative point on plane = 0 0 0
           plane normal = -1 -1 -1
           side to keep = positive
         end clip myClip
         begin imageset myImageSet
           camera = closeupCam
           operation = myClip
           color by scalar = vonmises
           show edges = true
           image size = 1280 1024
           image basename = clipVM.
         end imageset myImageSet
       end
     end results output catalyst_example_output_results_block
```

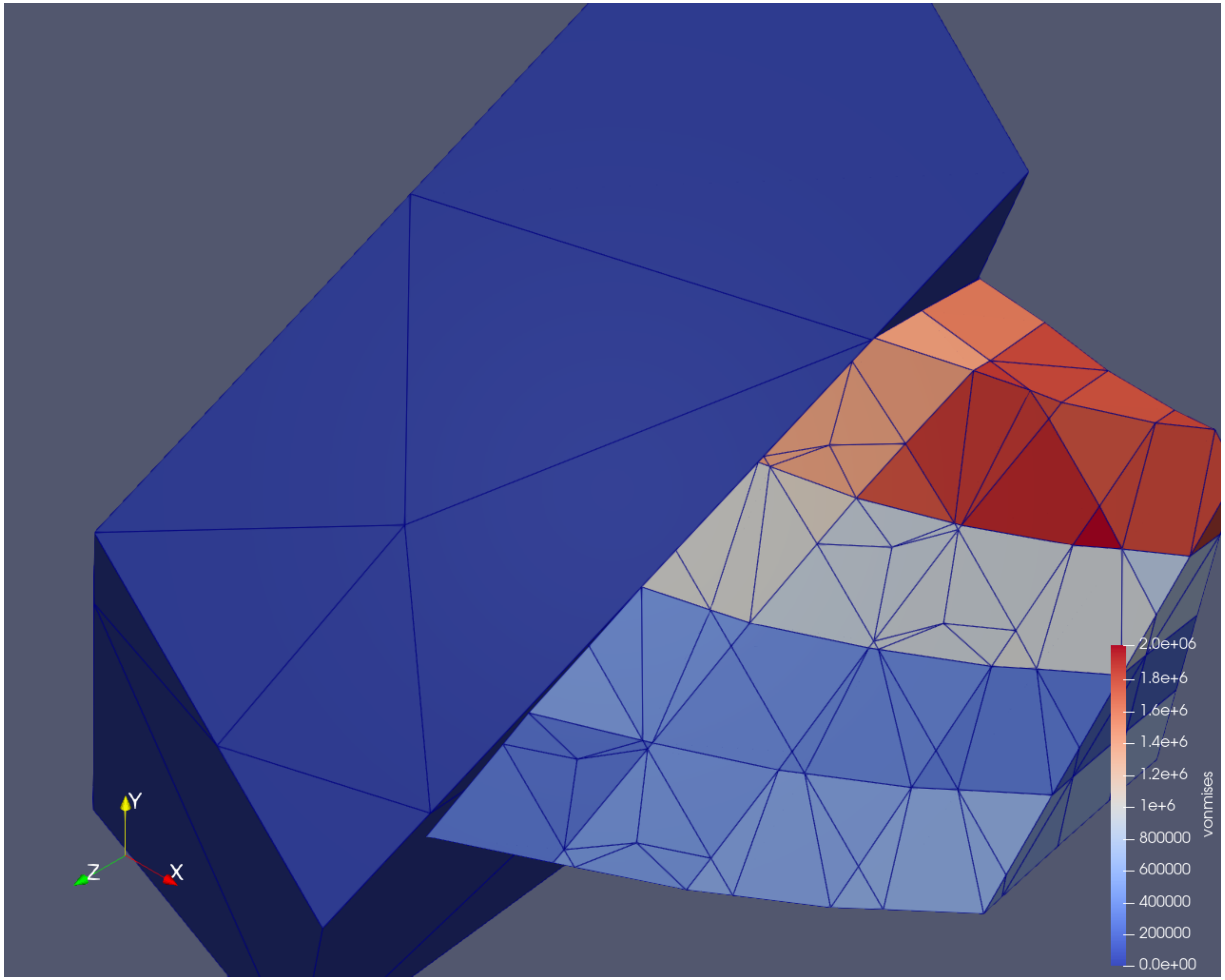
CATALYST SCRIPT CREATION

Catalyst Script Creation Using ParaView (SPARC and Sierra)


Data From Simulation (likely low resolution) or Stand-in data
ParaView GUI
Create Visualization And Data Export Scene
Export Python Script
Reference Python Script from Catalyst section in Simulation Input Deck

Catalyst Script Creation Using Phactori


Create Visualization and Data Export Scene using Phactori scripting
(Sierra) Put script directly into Simulation input deck
(SPARC) Create Phactori Script in .json format
Reference .json format file from Simulation Input Deck


Y
Z
X
2.0e+06
1.8e+6
1.6e+6
1.4e+6
1.2e+6
1e+6
800000
600000
400000
200000
0.0e+00
vonmises

CATALYST SCRIPT CREATION IN SPARC

As before, create Catalyst scripts using the ParaView GUI

Users may also create Catalyst scripts with Phactori. In Sierra Phactori is internally converted to JSON format. In SPARC users may write Phactori script directly in JSON in a file separate from the SPARC input deck

SPARC input deck syntax specified in YAML

SPARC had volume-post-processing and surface-post-processing output blocks; we added volume-runtime-processing and surface-runtime processing blocks

Additional blocks for sending multiple grids (e.g. volume and wall surface) to the same Catalyst script for simultaneous visualization and analysis

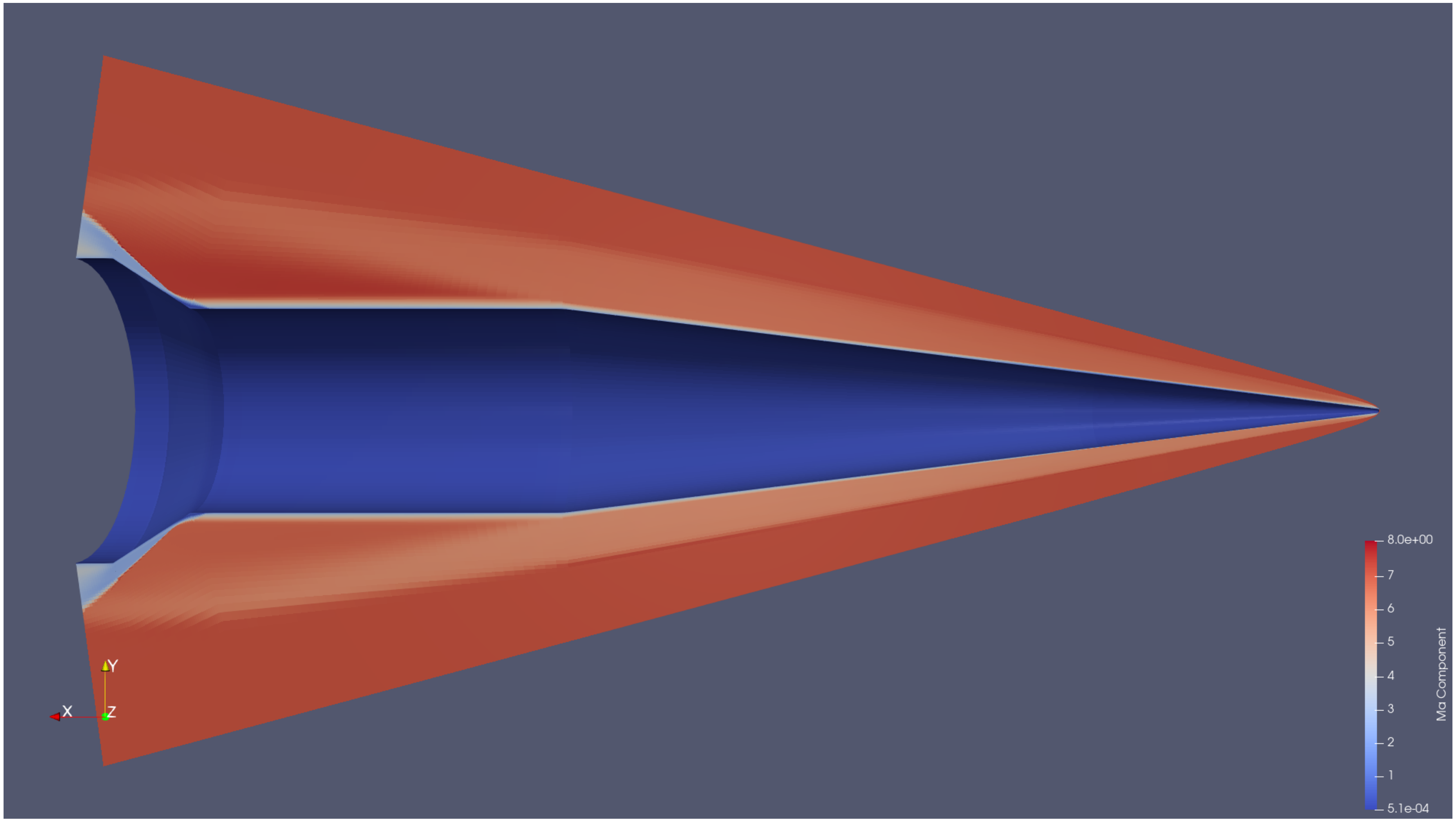
CATALYST IN SPARC

Catalyst Settings in SPARC Input Deck

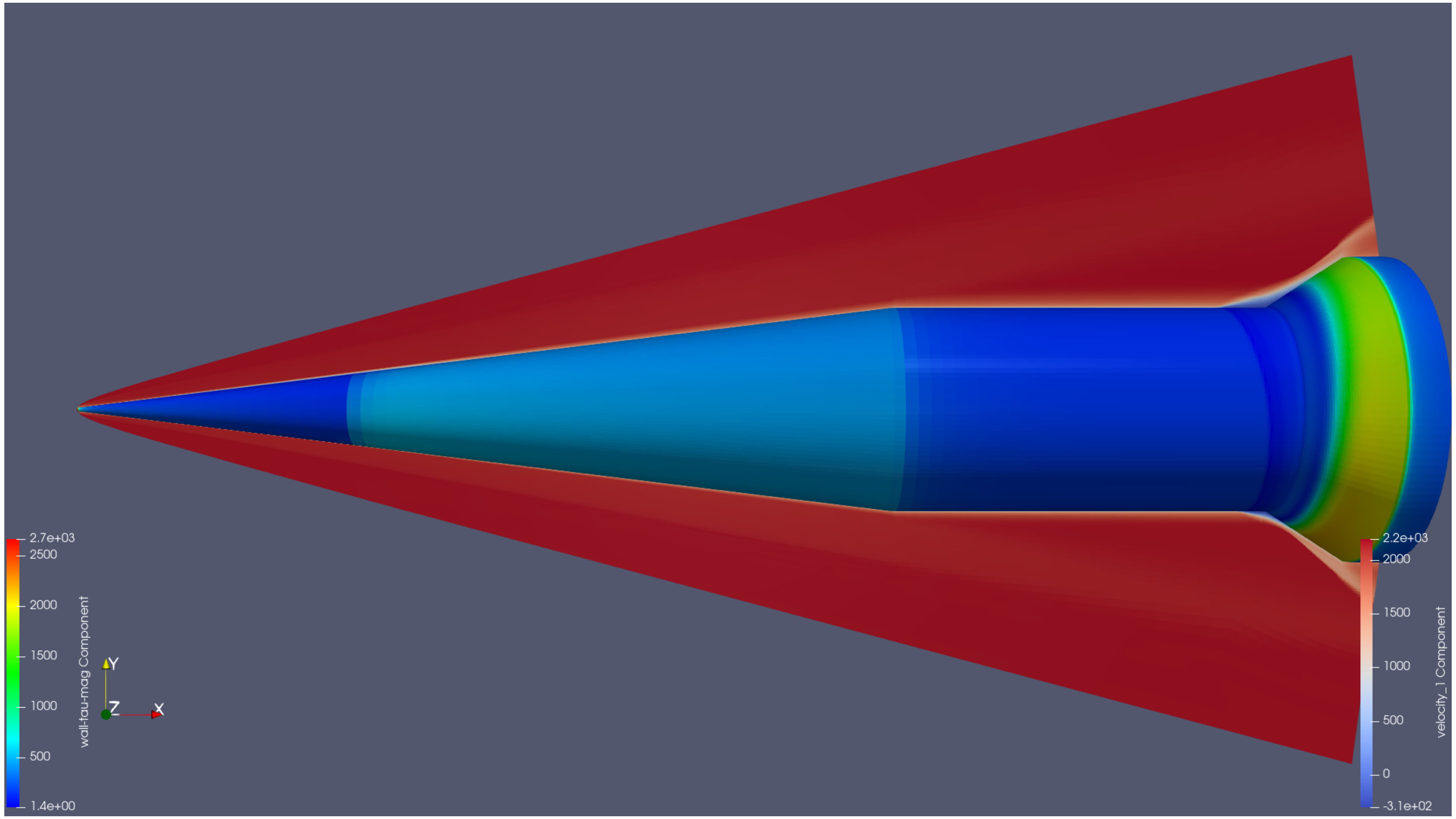
```
volume-runtime-processing:
[{
    datatype: cgns,
    step-start: 0,
    step-frequency: 10,
    variables:  [density, velocity,
temperature, temperature_vib, pressure, Ma,
total-enthalpy, nonlinear-residual,
turbulent-viscosity],
    runtime-control: {
      catalyst: {
        control-script: {
          script-type: phactori-json,
                script: catscript1.json
        }
      }
    }
  },
],
```

Phactori Script (JSON) Referred (catscript1.json)

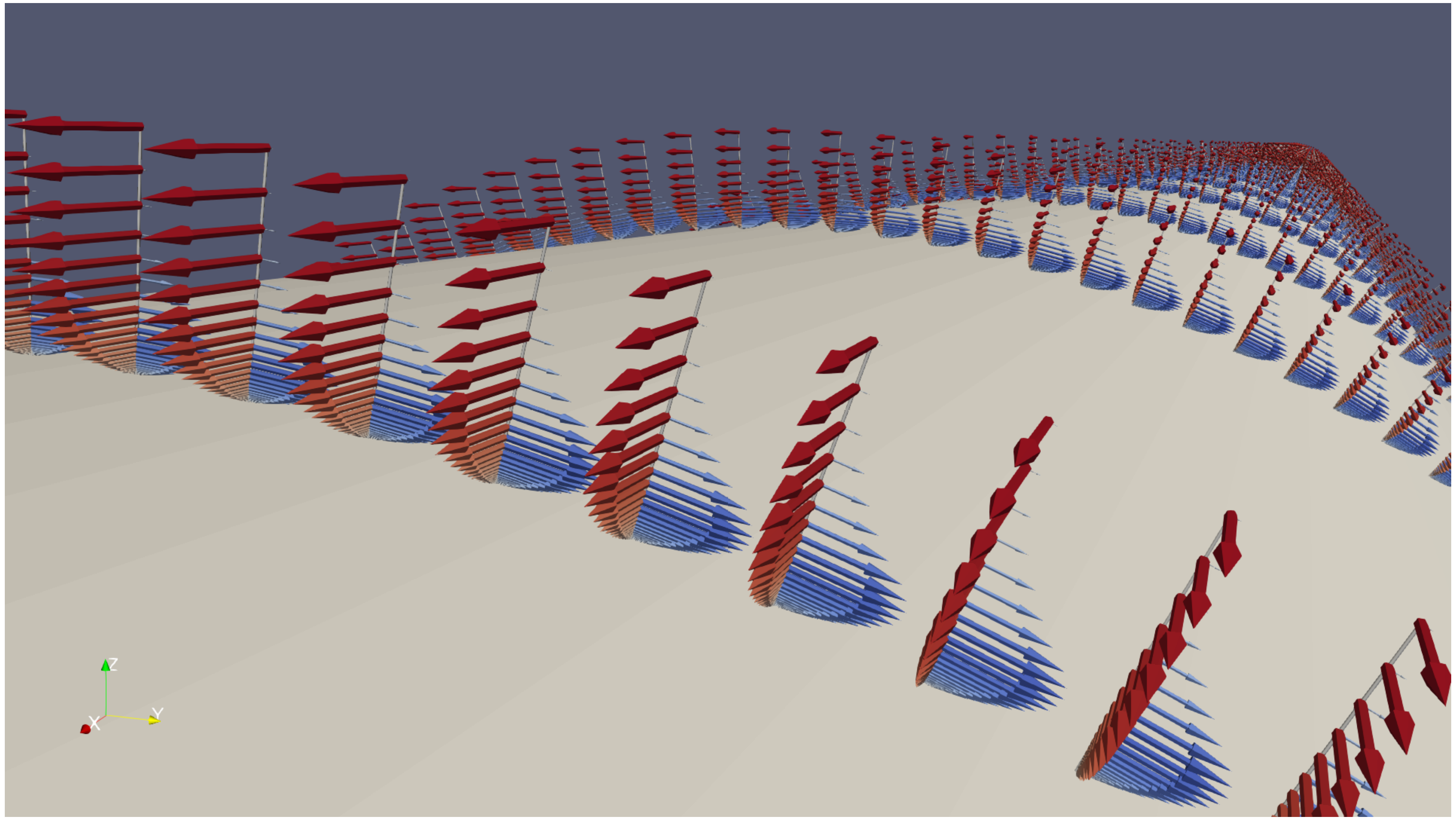
```
{
"camera blocks":{
  "camz2":{"camera type":"camera",
           "look direction":[0,0,1]},
},
"representation blocks":{
  "rep_Ma":{ "color by scalar":"Ma",
             "variable type":"element"},
},
"imageset blocks":{
  "is_Ma_camz2":{
    "camera":"camz2",
    "representation":"rep_Ma",
    "image basedirectory":"CatalystOutput",
    "image basename":"Ma_camz2."
  },
},
"operation blocks":{},
}
```


Y
X
Z
8.0e+00
7
6
5
4
3
2
1
5.1e-04
Ma Component

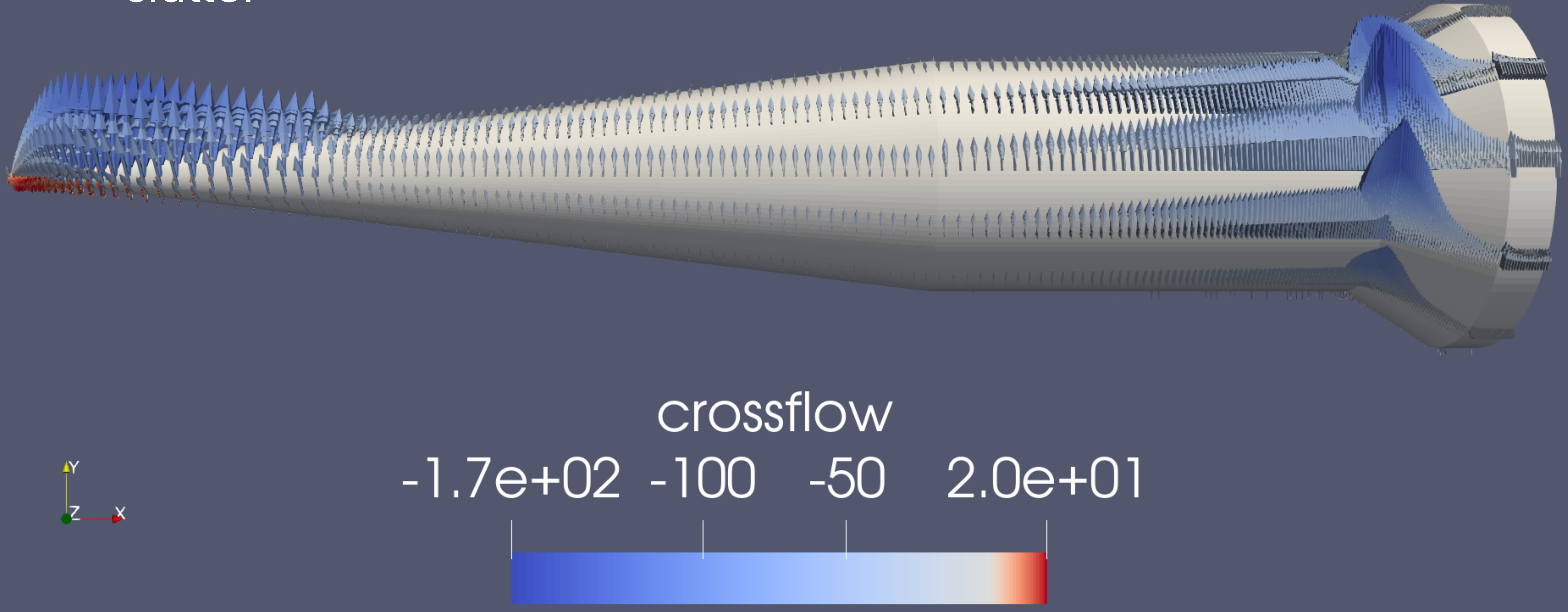
NEW SPARC COMPOSITE-RUNTIME-PROCESSING OUTPUT BLOCKS



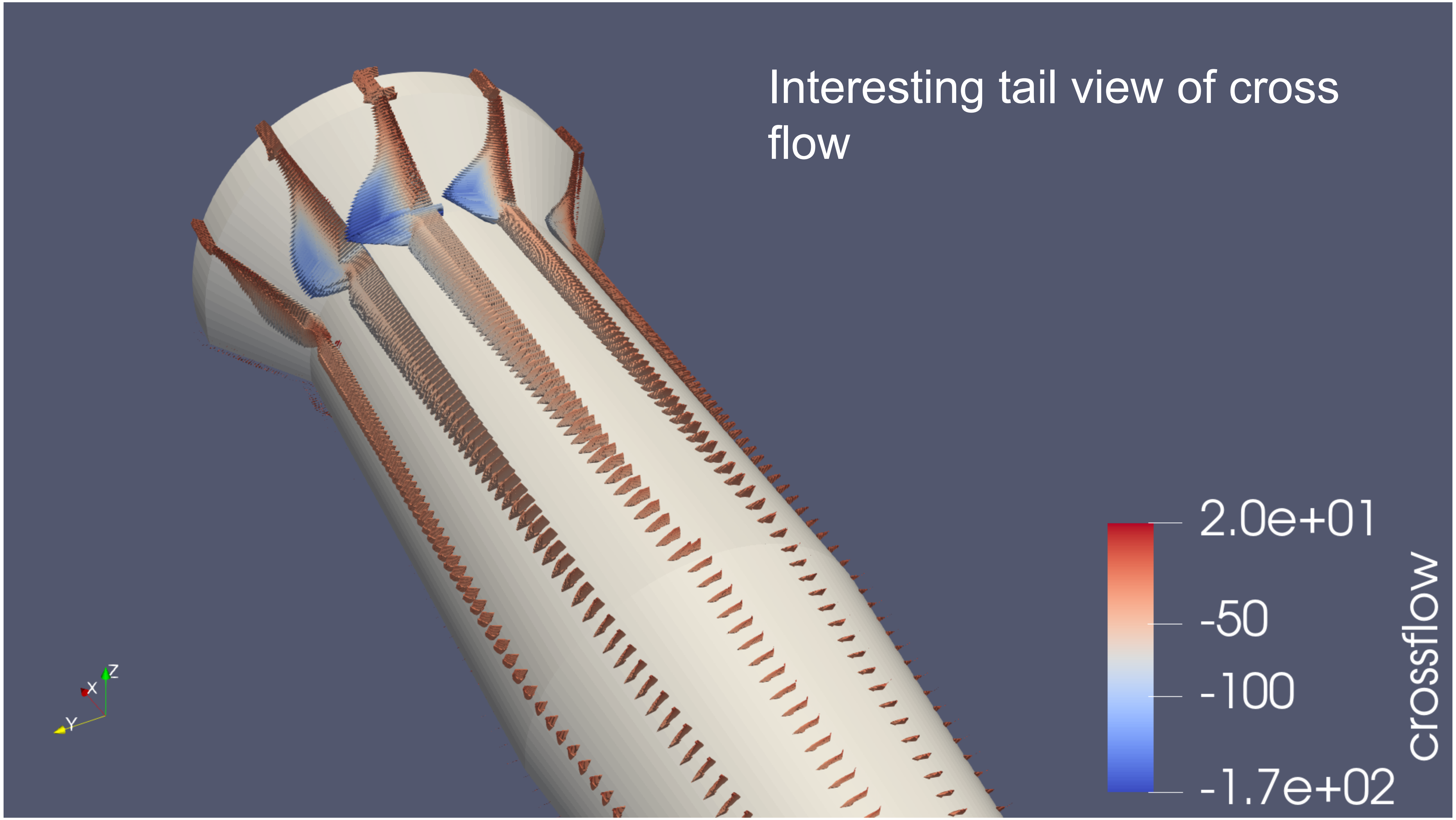
Towards In-Situ Analysis: Surface cross flow and surface tangential flow on body surface in constant non-symmetric incoming airflow. Sample points are positioned perpendicular to surface with nonlinear sample spacing. (Greg Weirs develop the ideas for this analysis/visualization with interactive ParaView, here you see results created with SPARC in-situ.)

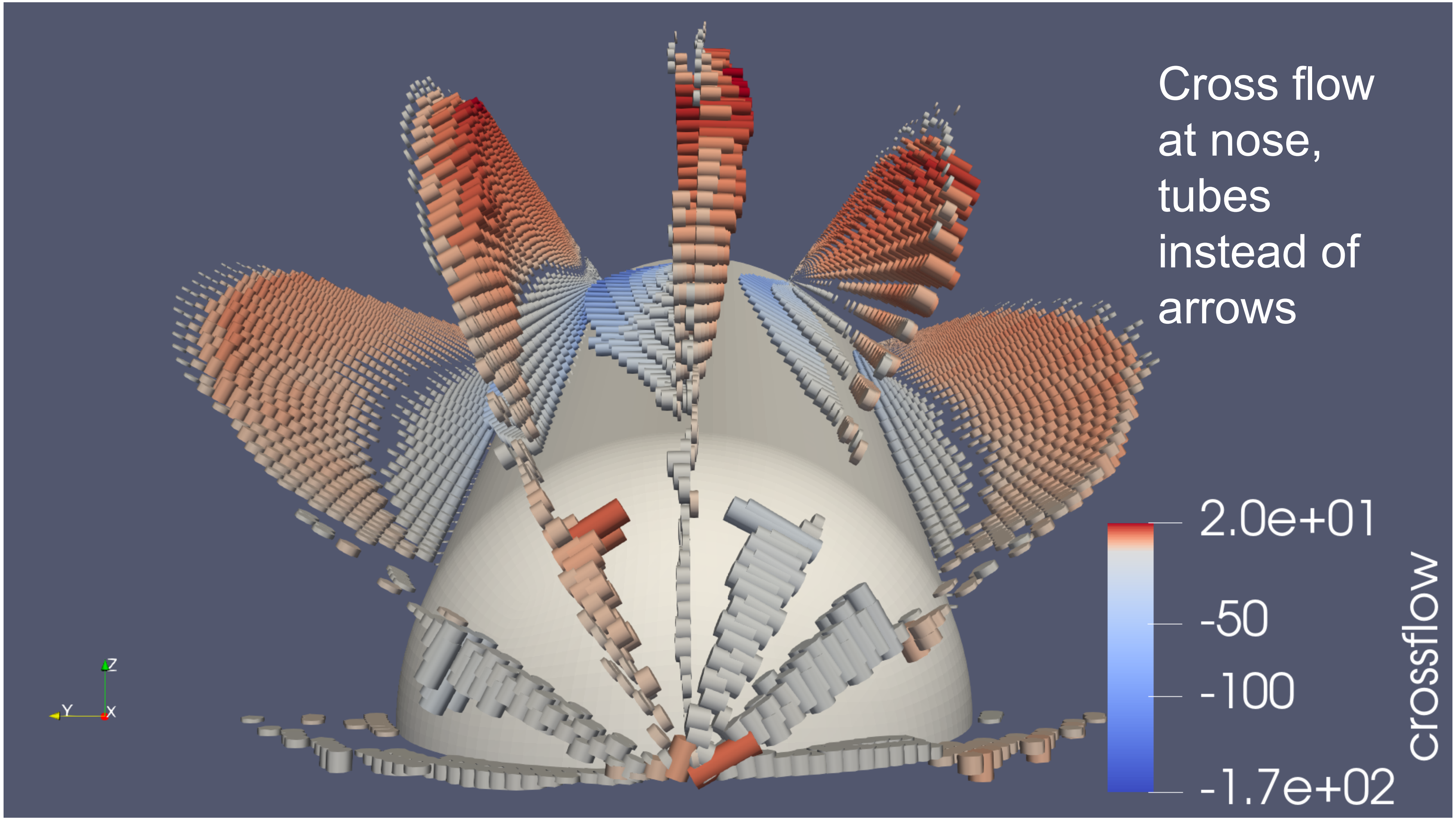


Crossflow only, arrows scaled to be visible from overview, samples from slices at 30-degree angles to avoid screen clutter



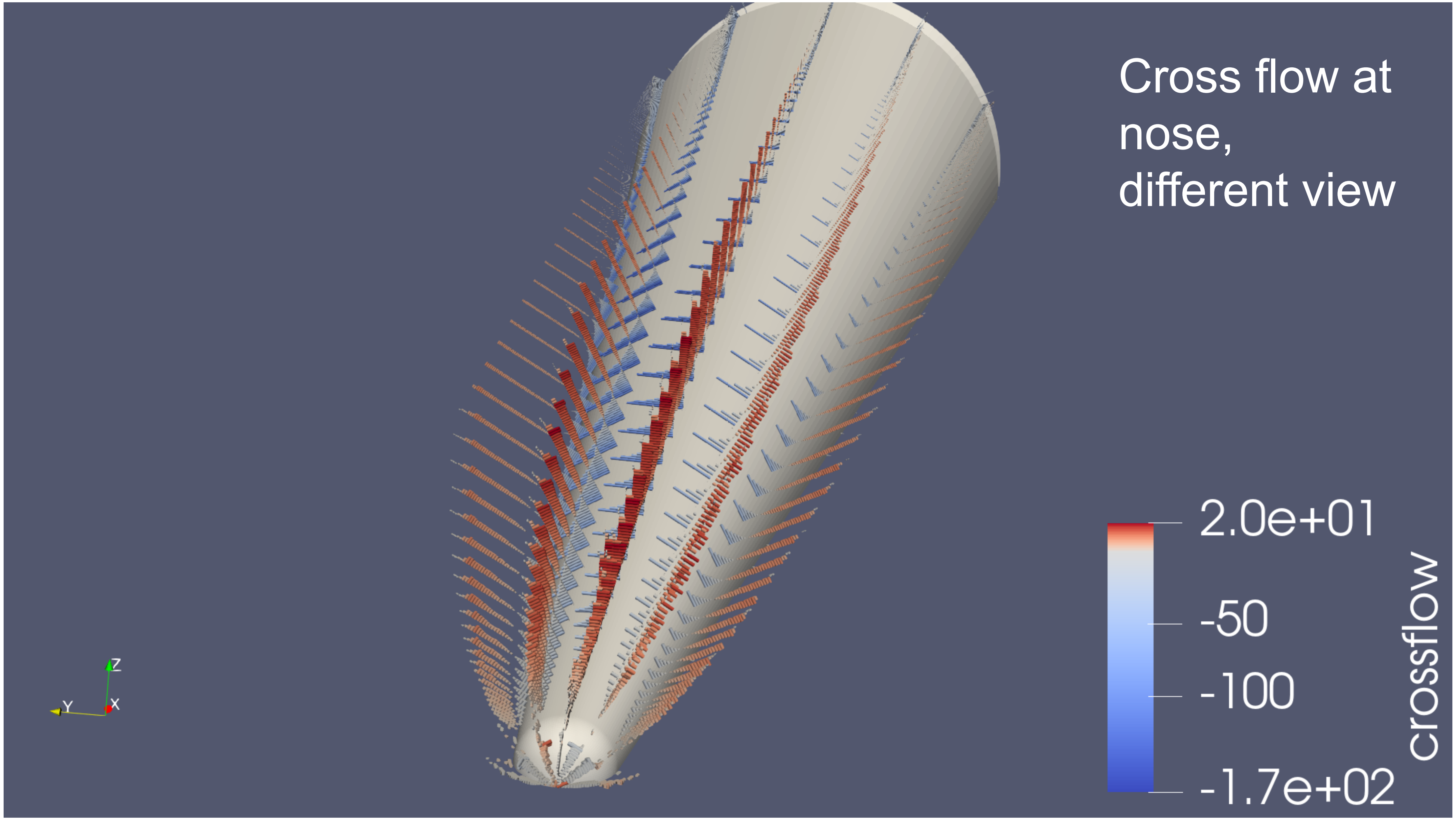
Interesting tail view of cross flow

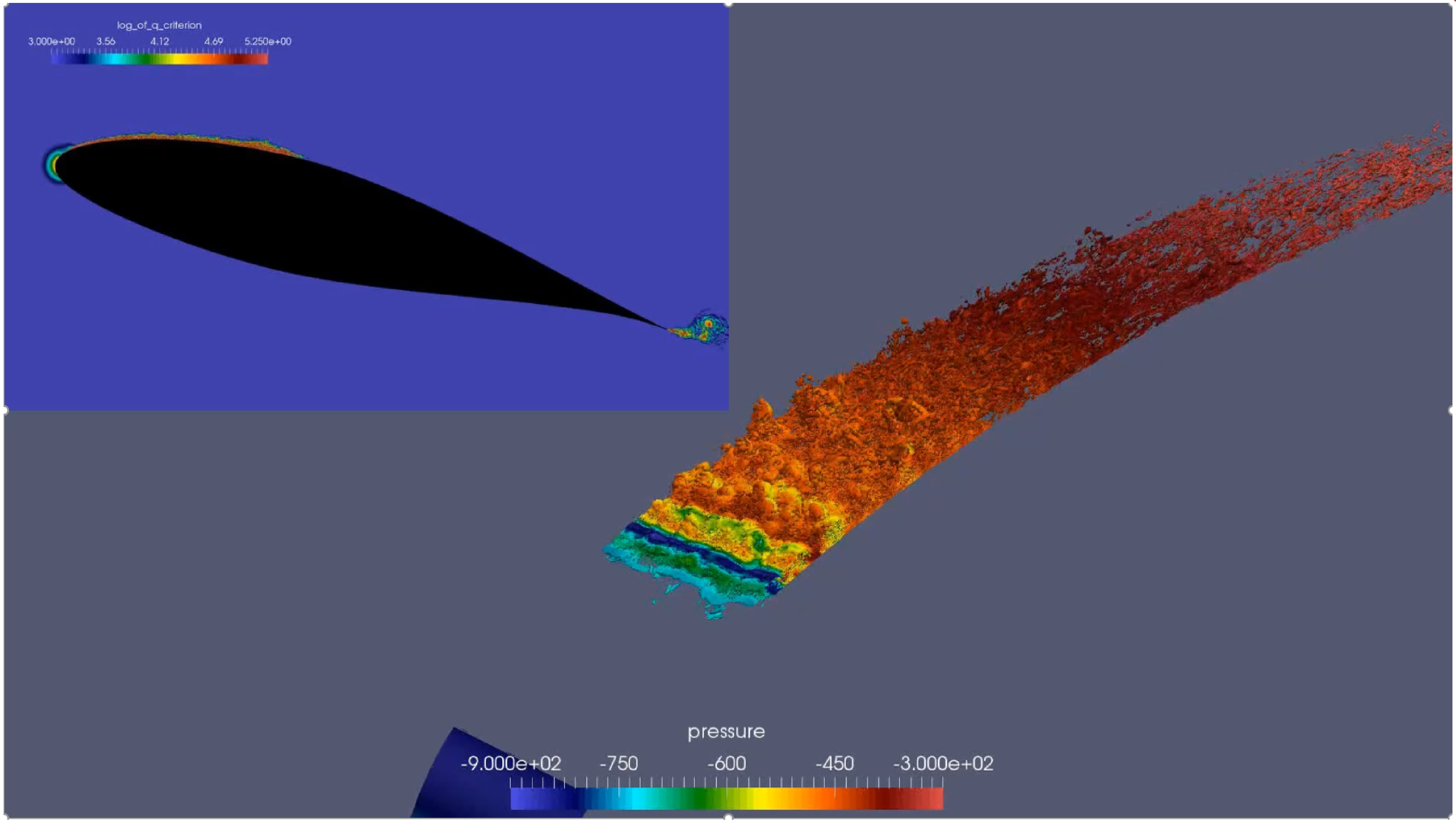




Cross flow at nose, tubes instead of arrows

Cross flow at nose, different view




log_of_q_criterion
3.000e+00
3.56
4.12
4.69
5.250e+00
pressure
-9.000e+02
-750
-600
-450
-3.000e+02

INTEGRATING WITH SIERRA AND SPARC DEVOPS

Part of our DevOps process has been integrated into the SPARC and Sierra DevOps processes

SPARC test suite has regression/integration tests which test the syntax of the Catalyst-related input deck syntax and test the integration with Catalyst by producing images

SIERRA has similar but less complete tests. Historically we only did a “smoke test” inside of SIERRA and did all the other testing outside their DevOps process. We are currently working on fully testing all the Sierra Catalyst input deck syntax inside of the Sierra test suite

Coordination with SPARC and Sierra DevOps teams is required and ongoing

Note: we can expand this a lot, should we?

DEVOPS NEED FOR CATALYST IN SPARC AND SIERRA

Deployment environment at SNL for Catalyst is complex:

Multiple HPC platforms with different architectures:

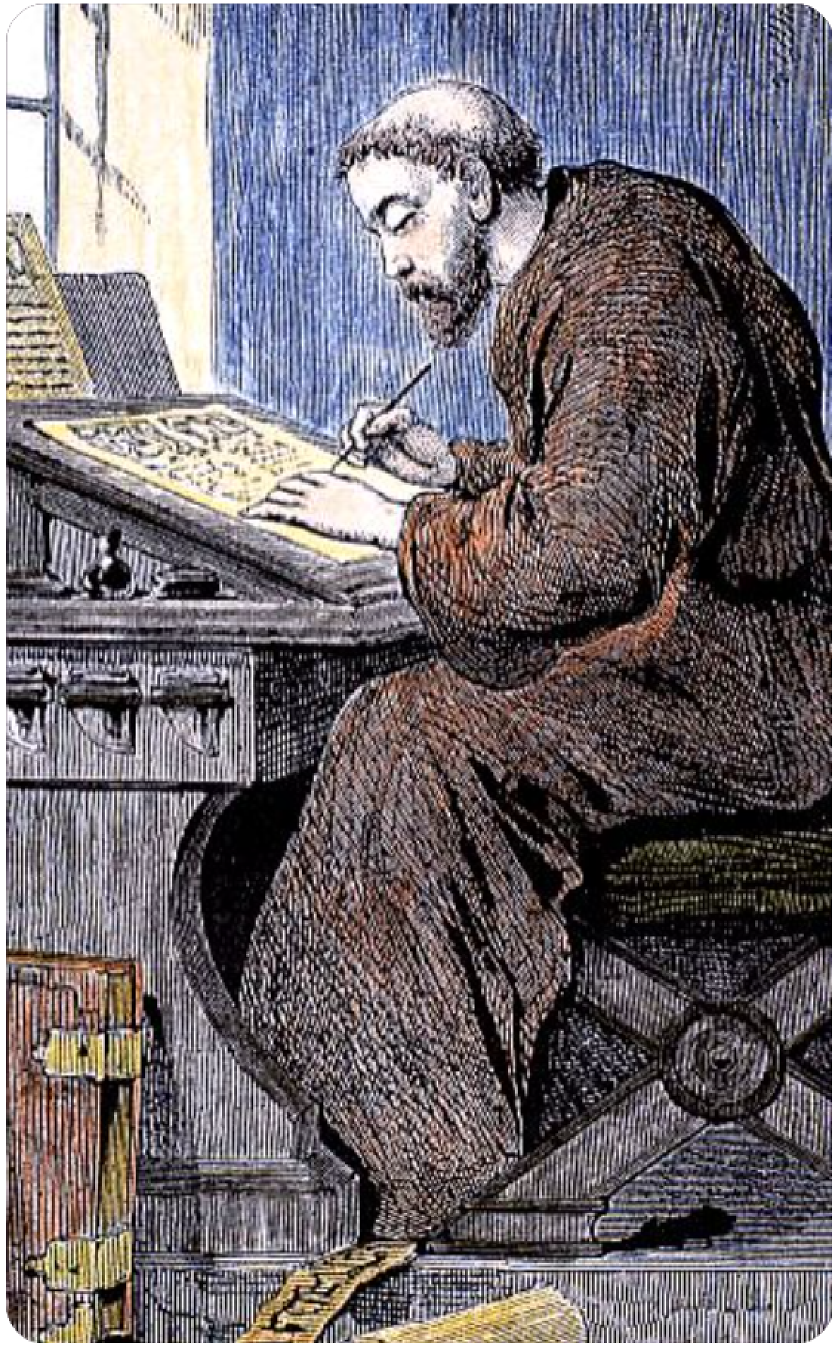
- TLCC2 and CTS-1: traditional multicore nodes on
- ATS-1: Intel Many Core Architecture with, e.g. 80 cores per chip (Trinity at LANL)
- ATS-2: Nodes with Nvidia GPUs (Sierra at LLNL)
- Vanguard ARM CPUs
- CEE: Common Engineering environment, non-HPC high performance workstations

SPARC and Sierra typically use different MPI versions and different compiler versions on each HPC and use different MPI versions from each other on each machine

SPARC and Sierra typically use different versions and builds of many TPLs

Catalyst must be configured, built, tested, and deployed for each HPC for each simulation code

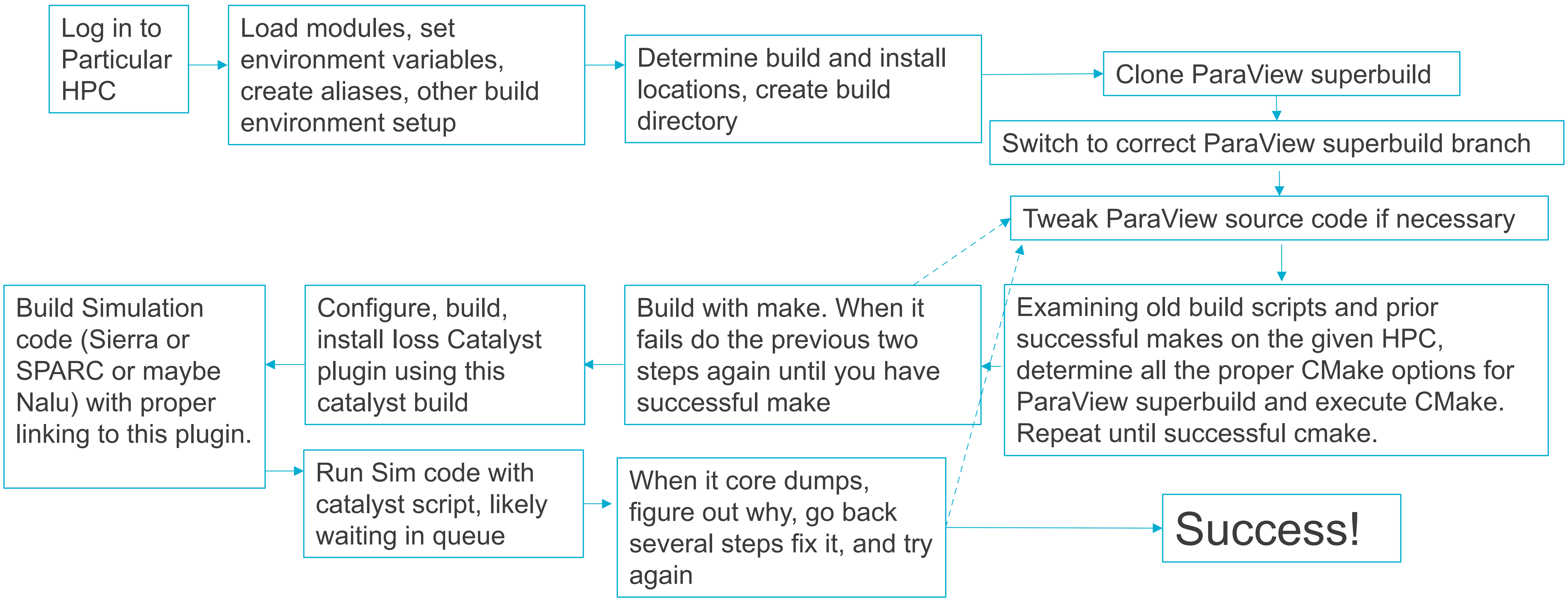
THE DARK AGES



THE DARK AGES

- Used ParaView superbuild to build ParaView/Catalyst on each HPC
- Hand tweaking of all CMake options, many machinations necessary on each HPC to get successful builds compatible with corresponding version of Sierra and Catalyst
- Knowledge of build details on each platform contained in various build scripts and engineer’s heads
- Testing done by Catalyst build engineer on Each machine by hand via running simulation codes with Catalyst Scripts
- We at least had a set of automated test via CMake that would run Sierra repeatedly with different autogenerated test Sierra input decks. Very useful but difficult to maintain and run.

THE DARK AGES (PARAVIEW SUPERBUILD ONLY)



Log in to Particular HPC

Load modules, set environment variables, create aliases, other build environment setup

Determine build and install locations, create build directory

Clone ParaView superbuild

Switch to correct ParaView superbuild branch

Tweak ParaView source code if necessary

Examining old build scripts and prior successful makes on the given HPC, determine all the proper CMake options for ParaView superbuild and execute CMake. Repeat until successful cmake.

Build with make. When it fails do the previous two steps again until you have successful make

Configure, build, install Ioss Catalyst plugin using this catalyst build

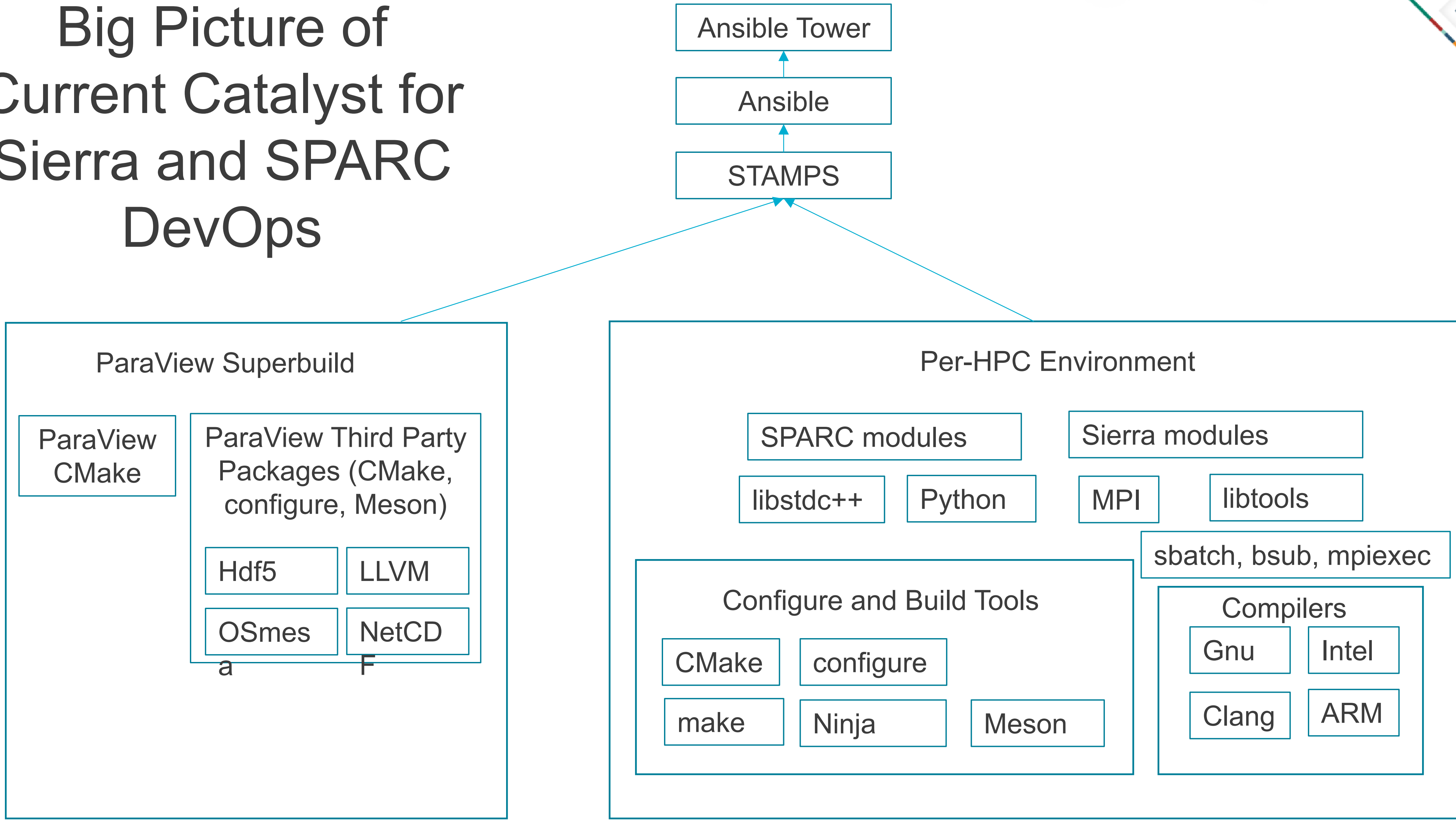
Build Simulation code (Sierra or SPARC or maybe Nalu) with proper linking to this plugin.

Run Sim code with catalyst script, likely waiting in queue

When it core dumps, figure out why, go back several steps fix it, and try again

Success!

Big Picture of Current Catalyst for Sierra and SPARC DevOps



STAMPS: THE HERALD OF THE NEW AGE

STAMPS: THE HERALD OF THE NEW AGE

- We built the Sandia Targeted Automatic Make ParaView System (STAMPS)
- Python script wrappers for ParaView Superbuild, which wraps ParaView CMake and its third party dependencies
- For each platform for each simulation code, platform specific CMake settings and any necessary third party package related alterations are contained in a python script specific to that platform and simulation code
- Top level build command to specify platform and locations of build and install directories (useful for both testing and deployment)
- Also builds IOSS catalyst plugin (for SPARC or Sierra as specified)
- Git repo to maintain system
- Likely we will continue to use STAMPS for a long time but we may eventually replace ParaView Superbuild with SPACK

STAMPS

Formerly building on each HPC using ParaView Superbuild required a lot of command line operations

- Lots of opportunity for error
- Time consuming
- Disorganized

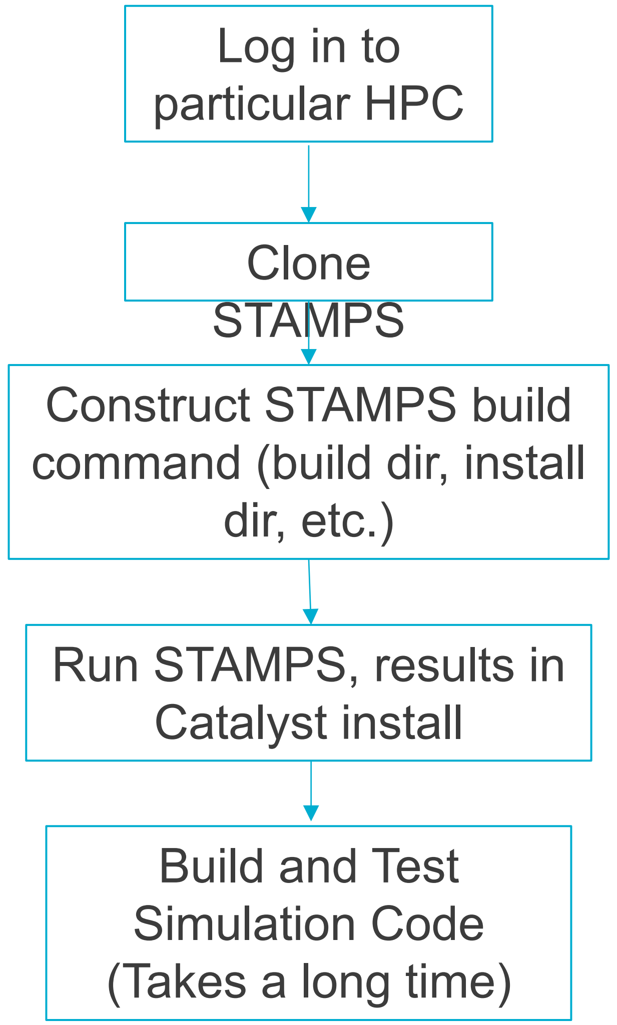
After STAMPS you could simply log on to the HPC, clone STAMPS, run the build command for the platform/code of interest, and walk away

- No user error
- Organized
- No monitoring of build process needed

STAMPS

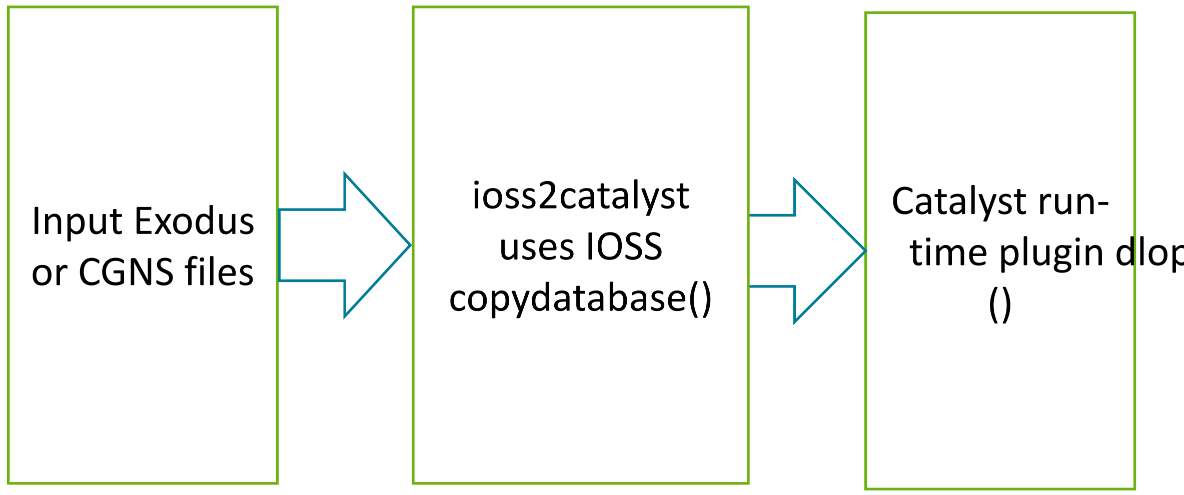
```
python3 /vscratch1/jamauld//stamps/stamps_paraview_build.py \
  --build_format ats2_pv_5_11_0_sparc \
  --build_directory /vscratch1/jamauld/build_dirs/build_1003_ats2_pv_5_11_0_sparc \
  --install_directory /vscratch1/jamauld/install_dirs/install_1003_ats2_pv_5_11_0_sparc \
  --build_ioss_catalyst_plugin \
  --build_ioss2catalyst_application \
  --number_of_build_processes 64
```

STAMPS WORKFLOW SIMULATION CODE


Log in to particular HPC
Clone STAMPS
Construct STAMPS build command (build dir, install dir, etc.)
Run STAMPS, results in Catalyst install
Build and Test Simulation Code (Takes a long time)

CATALYST TEST SUITE IN IOSS

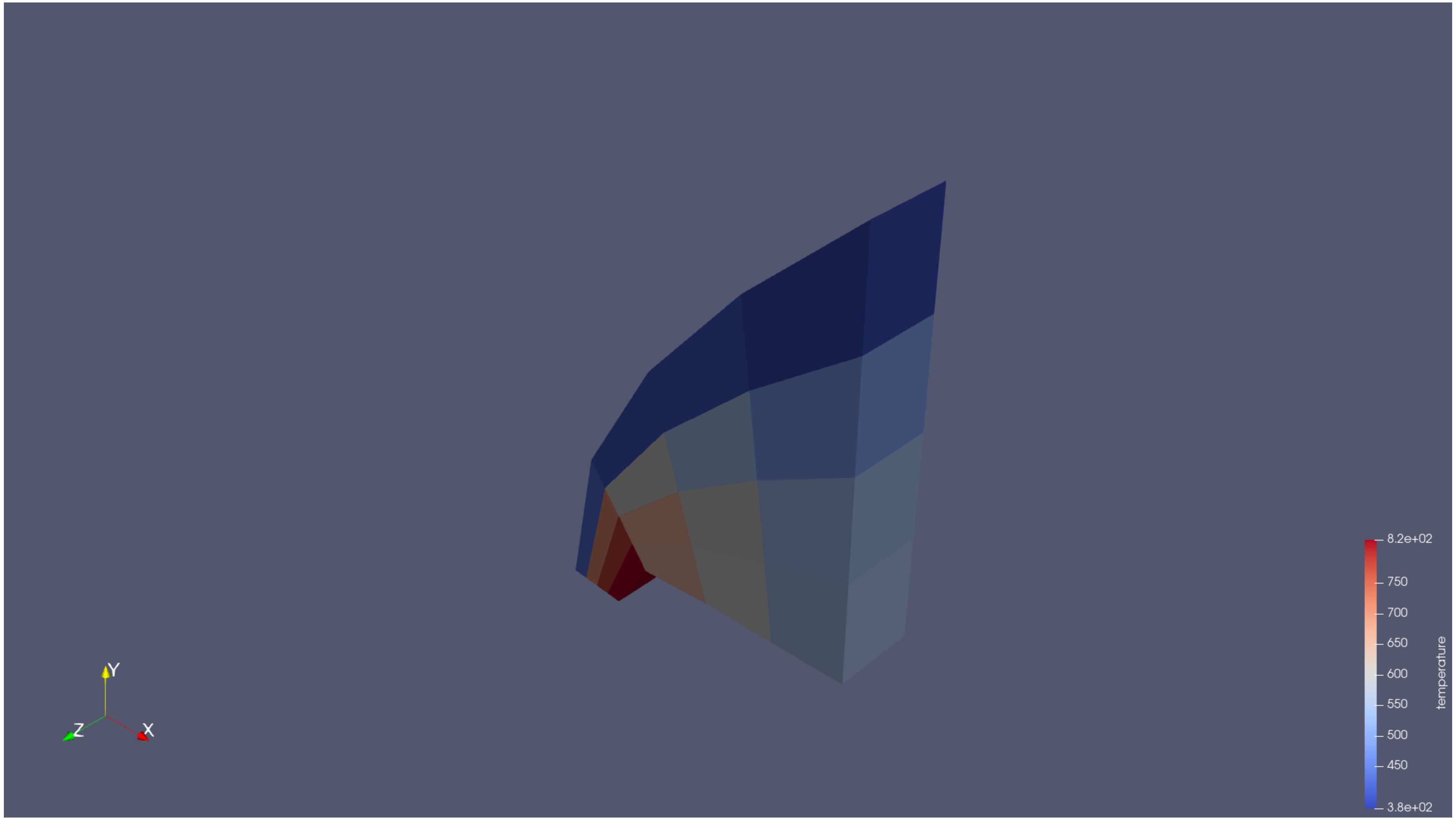
- Catalyst test suite uses program ioss2catalyst, which can be used from a test fixture or from the command line
- The ioss2catalyst program uses the IOSS function copydatabase() to read CGNS and Exodus files in parallel and send them through the appropriate IOSS Catalyst database
- The test suite does not require linking to the Sierra or SPARC simulation codes, since the ioss2catalyst program acts a stand-in for these programs
- Catalyst test suite contains 42 tests that exercise code paths for CGNS and Exodus simulation data conversion to Catalyst, Catalyst control parameters, usage logging, and Phactori Sierra input deck processing
- If the Catalyst test suite builds on runs on an HPC platform, users can have a high degree of confidence it will run from a calling simulation code


Input Exodus or CGNS files
ioss2catalyst uses IOSS copydatabase()
Catalyst run-time plugin dlop
()

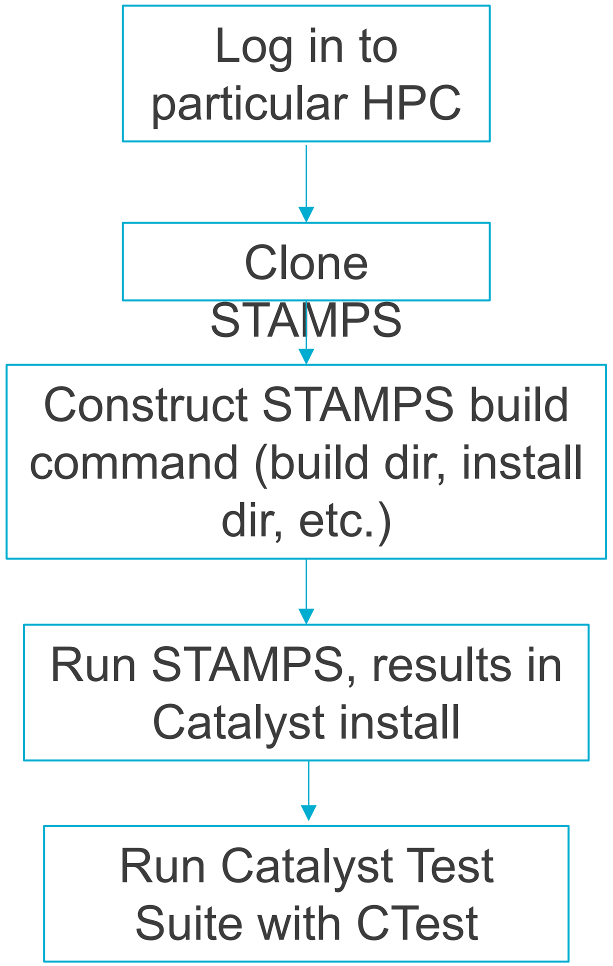
STAMPS RUNS SEACAS CATALYST TESTS

- As described, the SEACAS Catalyst test suite is absolutely vital to maintaining code reliability
- We added to STAMPS the options and other groundwork to allow us to turn on the test suite for the IOSS catalyst plugin.
  - SEACAS package required to run ioss catalyst plugin tests
  - Method of running CTest varies on each platform
- After quite a bit of work, on each HPC platform after the STAMPS build script is run we can go into the ioss_catalyst_plugin build directory and run ctest to test the result (or submit a job via sbatch or bsub to run ctest)
- Now the build engineer can run the build script and then run ctest to verify the build

STAMPS RUNS SEACAS CATALYST TESTS



STAMPS WORKFLOW WITH CATALYST TEST SUITE


Log in to particular HPC
Clone STAMPS
Construct STAMPS build command (build dir, install dir, etc.)
Run STAMPS, results in Catalyst install
Run Catalyst Test Suite with CTest

ANSIBLE FOR CATALYST CONFIGURE, BUILD, TEST, INSTALL

- While STAMPS is a vast improvement, we still have to log on to each HPC, check out STAMPS, run the build script, and then run CTest
- Could maintain a script on each HPC to do this automatically
- Ansible playbooks are more maintainable and consistent than shell scripts and have access to libraries for dealing with things like git and file creation (it’s easier to pass along understanding of an Ansible playbook than an ad-hoc shell script)

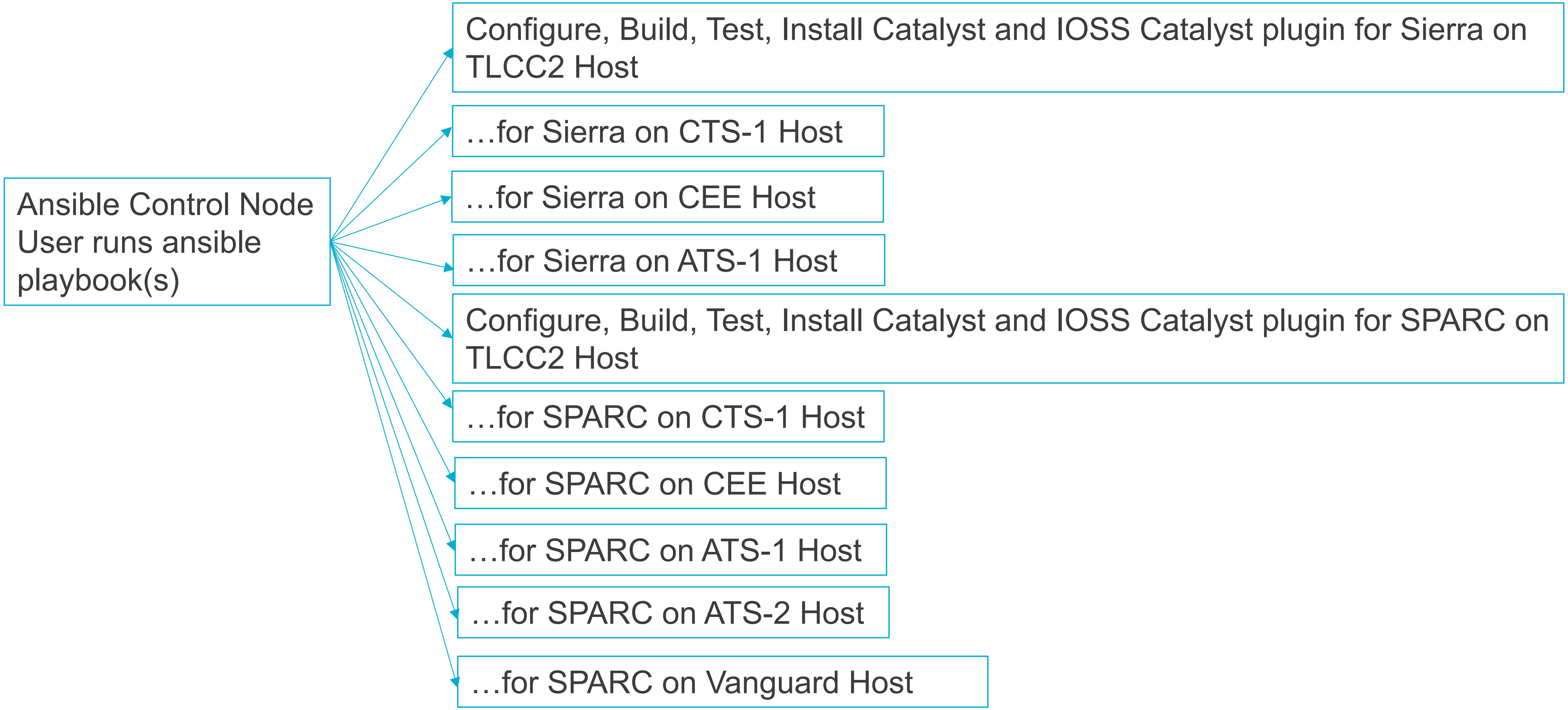
ANSIBLE FOR CATALYST CONFIGURE, BUILD, TEST, INSTALL

- The hardest piece turned out to be the testing
- Very inconsistent how testing must be done on the various platforms
  - Some you just run CTest where you build
    - Some you run CTest from a batch file
      - sbatch
      - Bsub
    - Sometimes you need to use mpiexec to run CTest on the allocated node, sometimes mpiexec isn’t available and you have to use srun
    - Other considerations I’ve probably forgotten but which are stored in the HPC-corresponding Ansible playbook (or playbook section)

ANSIBLE FOR CATALYST CONFIGURE, BUILD, TEST, INSTALL

- Now build engineer can, from a single location, run Ansible playbooks which configure, build, install, and test the Catalyst and IOSS catalyst plugin
- Engineer can run a single playbook or launch many of them simultaneously (I do this often). Compute burden on host system is low. I usually background the jobs and redirect their output to different files for each playbook run.
- Our Ansible playbooks can copy test results back to the launching system, but we usually turn off this step and let reporting go to cdash (see later).

ANSIBLE FOR CATALYST CONFIGURE, BUILD, TEST, INSTALL


Ansible Control Node
User runs ansible playbook(s)
Configure, Build, Test, Install Catalyst and IOSS Catalyst plugin for Sierra on TLCC2 Host
…for Sierra on CTS-1 Host
…for Sierra on CEE Host
…for Sierra on ATS-1 Host
Configure, Build, Test, Install Catalyst and IOSS Catalyst plugin for SPARC on TLCC2 Host
…for SPARC on CTS-1 Host
…for SPARC on CEE Host
…for SPARC on ATS-1 Host
…for SPARC on ATS-2 Host
…for SPARC on Vanguard Host

CDASH REPORTING

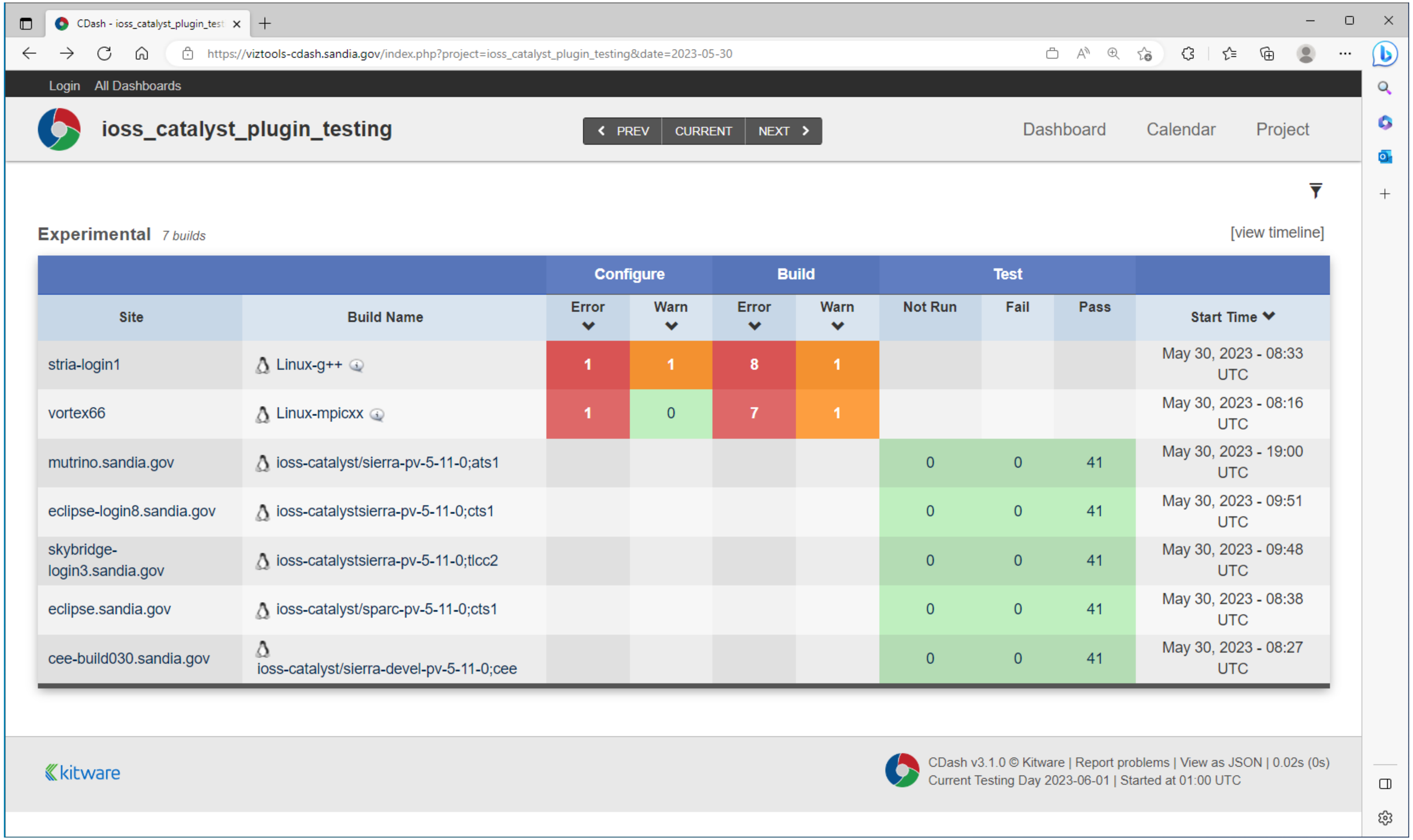
Initially Ansible playbooks would copy the test results back to the host system, and we could view the results from each HPC back on the host system

We were heading towards scheduled automatic launching of ansible playbooks and we didn’t want to have to look at test report files by hand

We added to the IOSS Catalyst plugin CTest the capability to report to CDash

After, again, a fair amount of work, we got each HPC to successfully report the CTest results to a CDash dashboard

CDASH REPORTING



CDash - ioss_catalyst_plugin_test

https://viztools-cdash.sandia.gov/index.php?project=ioss_catalyst_plugin_testing&date=2023-05-30

Login All Dashboards

ioss_catalyst_plugin_testing

Experimental *7 builds*

[view timeline]

		Configure		Build		Test			
Site	Build Name	Error	Warn	Error	Warn	Not Run	Fail	Pass	Start Time
stria-login1	Linux-g++	1	1	8	1				May 30, 2023 - 08:33 UTC
vortex66	Linux-mpicxx	1	0	7	1				May 30, 2023 - 08:16 UTC
mutrino.sandia.gov	ioss-catalyst/sierra-pv-5-11-0;ats1					0	0	41	May 30, 2023 - 19:00 UTC
eclipse-login8.sandia.gov	ioss-catalystsierra-pv-5-11-0;cts1					0	0	41	May 30, 2023 - 09:51 UTC
skybridge-login3.sandia.gov	ioss-catalystsierra-pv-5-11-0;tlcc2					0	0	41	May 30, 2023 - 09:48 UTC
eclipse.sandia.gov	ioss-catalyst/sparc-pv-5-11-0;cts1					0	0	41	May 30, 2023 - 08:38 UTC
cee-build030.sandia.gov	ioss-catalyst/sierra-devel-pv-5-11-0;cee					0	0	41	May 30, 2023 - 08:27 UTC

kitware

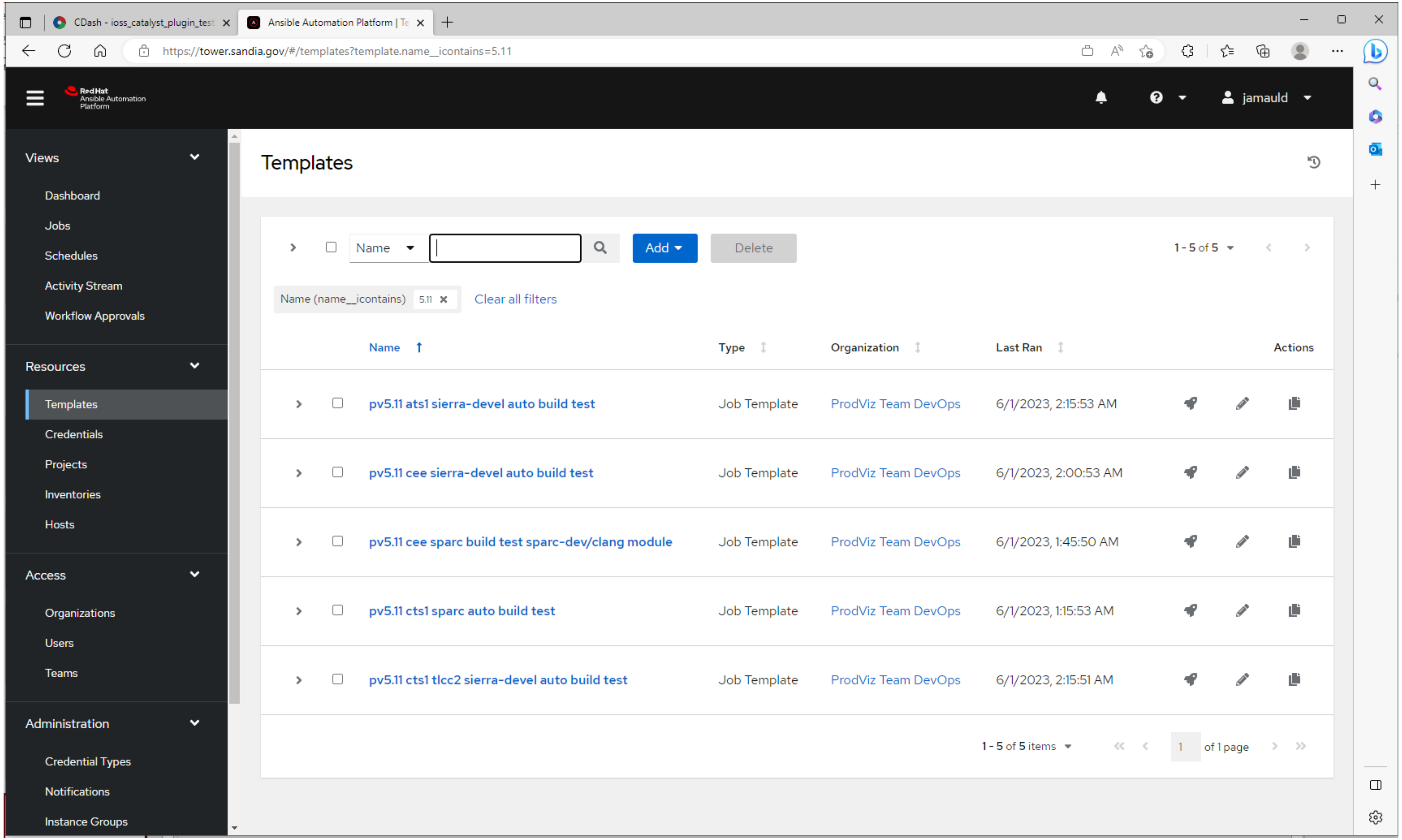
CDash v3.1.0 © Kitware | Report problems | View as JSON | 0.02s (0s)
Current Testing Day 2023-06-01 | Started at 01:00 UTC

SCHEDULED AUTOMATED BUILDING WITH ANSIBLE TOWER

- Now we had automated configure, build, install, test, and test report to CDash on each HPC via Ansible playbooks
- Rather than launching these jobs by hand via the “ansible-playbook” command, we wanted these jobs to launch automatically from a DevOps system
- We wanted to schedule nightly builds/tests on each HPC platform for each simulation Code

SCHEDULED AUTOMATED BUILDING WITH ANSIBLE TOWER

- We created an Ansible Tower environment (projects, templates, hosts, schedules, etc.)
- Nightly Scheduled jobs for configure, build and test, (and install test)
  - IOSS catalyst plugin for Sierra
    - Cee with sierra module
    - Cee with sierra-devel module
    - Cee with sparc module
    - TLCC2 and CTS-1 with sierra-devel module and sparc module
    - ATS-1 with sierra-devel module and sparc module
    - ATS-2 and Vanguard with sparc module
- Currently moving to ParaView 5.11 and still standing up automated testing for various platforms



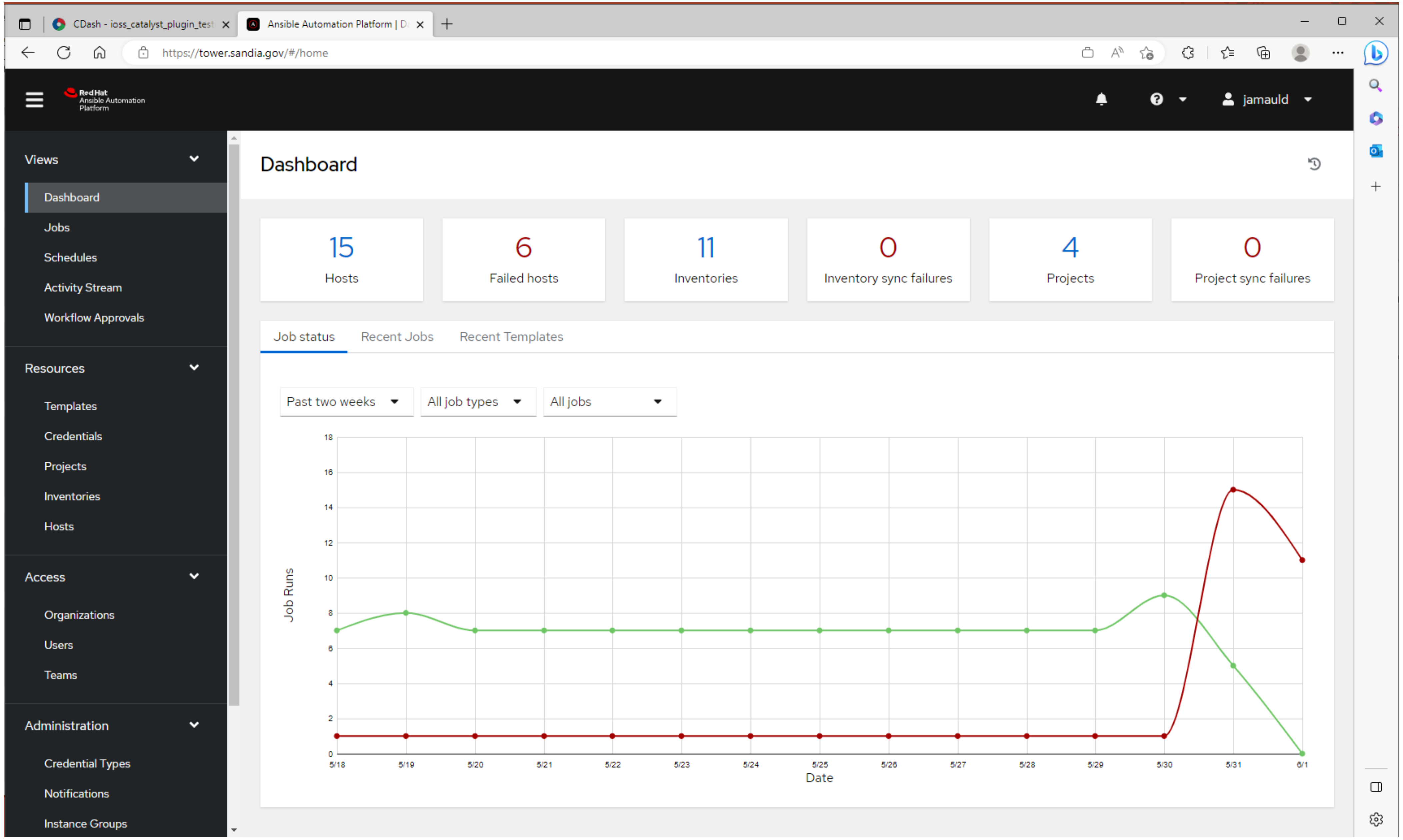
Templates

Name | Add | Delete | 1 - 5 of 5

Name (name__icontains) 5.11 | Clear all filters

Name	Type	Organization	Last Ran	Actions
pv5.11 ats1 sierra-devel auto build test	Job Template	ProdViz Team DevOps	6/1/2023, 2:15:53 AM	
pv5.11 cee sierra-devel auto build test	Job Template	ProdViz Team DevOps	6/1/2023, 2:00:53 AM	
pv5.11 cee sparc build test sparc-dev/clang module	Job Template	ProdViz Team DevOps	6/1/2023, 1:45:50 AM	
pv5.11 cts1 sparc auto build test	Job Template	ProdViz Team DevOps	6/1/2023, 1:15:53 AM	
pv5.11 cts1 tlcc2 sierra-devel auto build test	Job Template	ProdViz Team DevOps	6/1/2023, 2:15:51 AM	

1 - 5 of 5 items | 1 of 1 page


CDash - ioss_catalyst_plugin_test
Ansible Automation Platform | D
https://tower.sandia.gov/#/home

Dashboard

15	6	11	0	4	0
Hosts	Failed hosts	Inventories	Inventory sync failures	Projects	Project sync failures

RECENT PRACTICAL SUCCESS

Week of June 5-9 2023, all automated builds started failing

Dashboard was suddenly empty of successful results

Investigation revealed that there was a new system requirement for “git-lfs”, a large file system module for git

Altered ansible playbooks so that the git clone activity had the needed access to a git-lfs install

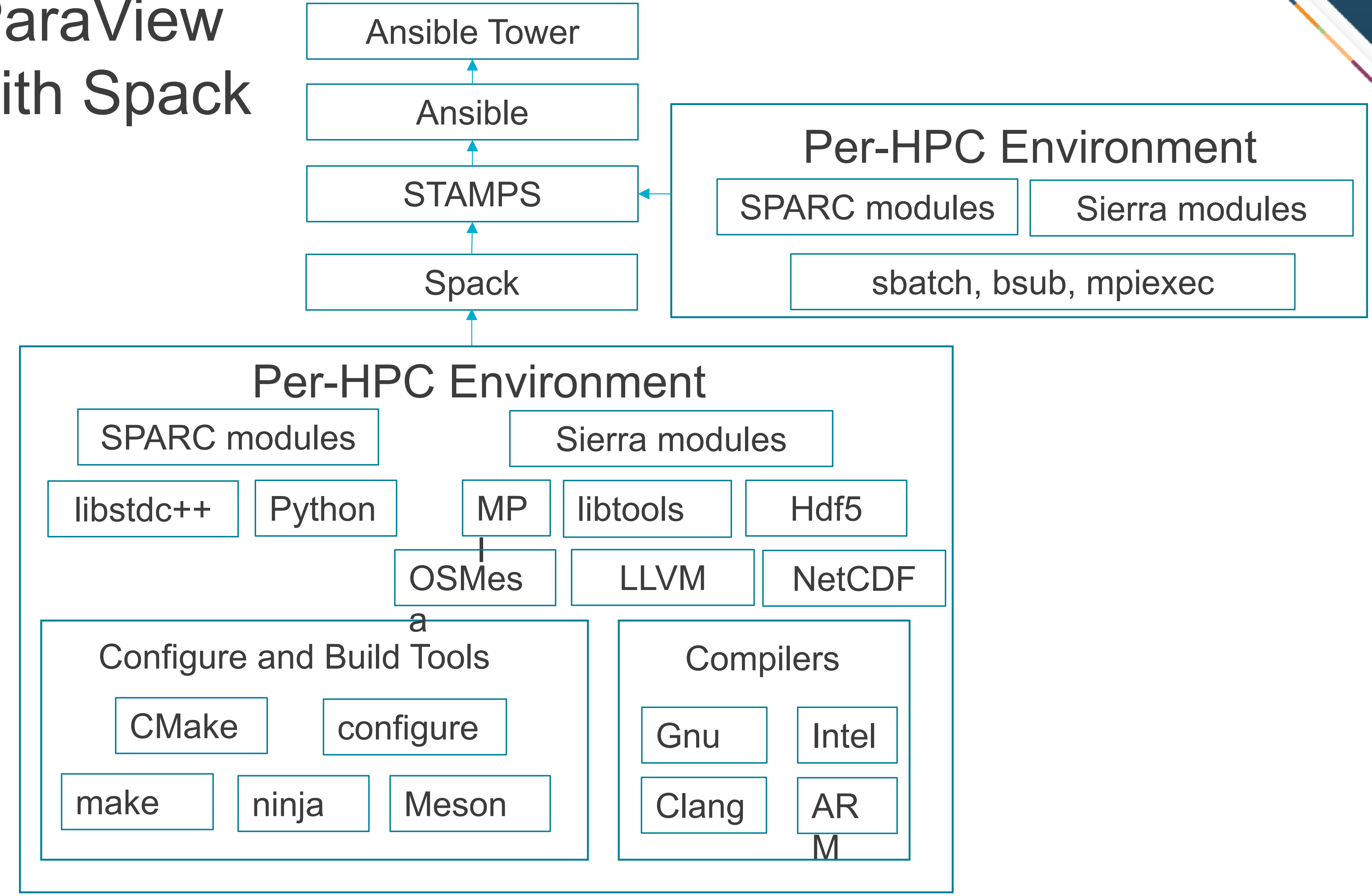
No users affected, discovery of problem was all automatic

Proximity in time and singularity of problem made fixing it much easier

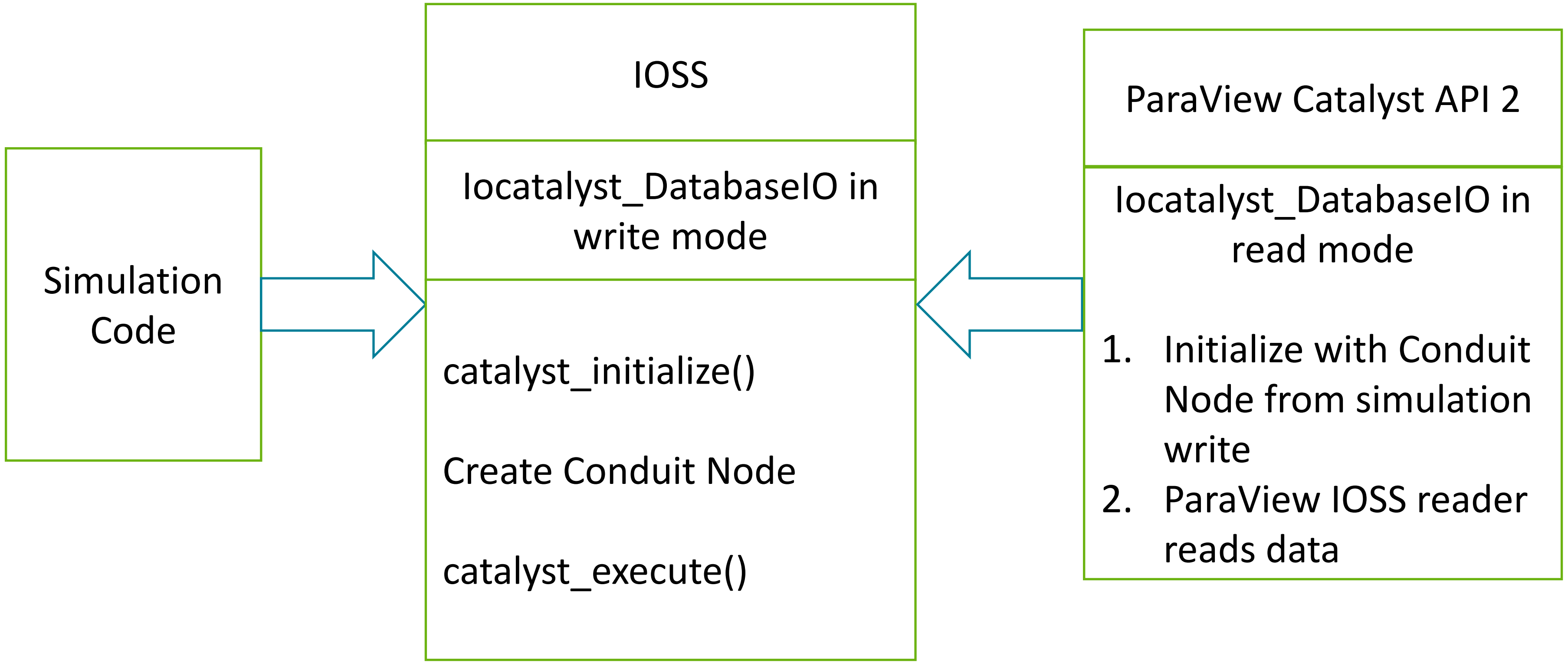
If problem had been undiscovered until a new ParaView release it would have been much harder to find and fix

A FEW THINGS IN THE FUTURE

Replacing ParaView Superbuild with Spack



CATALYST API 2 IN IOSS


Simulation Code
IOSS
Iocatalyst_DatabaseIO in write mode
catalyst_initialize()
Create Conduit Node
catalyst_execute()
ParaView Catalyst API 2
Iocatalyst_DatabaseIO in read mode
1. Initialize with Conduit Node from simulation write
2. ParaView IOSS reader reads data

- New Catalyst IOSS database class: Iocatalyst_DatabaseIO
- On IOSS write, creates a Conduit Node
- On IOSS read, accesses data from a Conduit Node
- ParaView has an IOSS reader
- Iocatalyst_DatabaseIO is just another IOSS database type like Exodus or CGNS
- Code paths in Catalyst and ParaView GUI are identical!

LESSONS LEARNED

- We did not and do not really have enough manpower for our requirements
- Productizing Catalyst in our environment is entirely hopeless without DevOps, but at least a tractable problem with DevOps
- Automated testing is key--difficult in a varied HPC environment but vital
- Catalyst has some unique challenges due to the way it integrates into simulation codes, and design, implementation, and DevOps work for each code is necessary
- Ansible is excellent and has some useful advantages over bespoke scripts, but if we had started a bit later we probably would have at least compared and considered gitlab runners

RELATED PUBLICATIONS

V. G. Weirs, E. M. Raybourn, R. Milewicz, K. Muollo, J. A. Mauldin, and T. J. Otahal, ‘Enabling Catalyst Adoption in SPARC’, in 2022 IEEE/ACM International Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), 2022, pp. 20–25.

J. A. Mauldin, T. J. Otahal, A. M. Agelastos, and S. P. Domino, ‘In-Situ Visualization for the Large Scale Computing Initiative Milestone’, in Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, Denver, Colorado, USA, 2019, pp. 1–5.

Templet, Jr., Gary J., Glickman, Matthew R., Kordenbrock, Todd Henry, Levy, Scott Larson Nicoll, Lofstead, Gerald Fredrick, Mauldin, Jeff, Otahal, Thomas Jay, Ulmer, Craig D., Widener, Patrick, and Oldfield, Ron A.. Data Services for Visualization and Analysis - ASC Level II Milestone (7186). United States: N. p., 2020. Web. doi:10.2172/1663267

CONCLUSION

- Catalyst still looks to be necessary as we move toward using Exascale, and can be useful at smaller scales
- Analysts and Simulation Code teams need ubiquitous availability
- Due to our DevOps implementation we are much closer to our goal
  - Provide in situ visualization and analysis capabilities in various simulation codes, on various HPC platforms, to Analysts at Sandia National Laboratories (SNL), and also to clients outside SNL who use those codes.
- Catalyst is (finally) attracting more interest and a few real users at SNL
- We need to continue our DevOps work which is ongoing to get to the point where SNL Code teams and Analysts can depend on Catalyst availability wherever the simulation codes reside
- We have received a little increased manpower (yay Alex!), but we probably still need additional help if we are committed to providing this capability long-term

SOME ACKNOWLEDGEMENTS

Jeff Mauldin and Tom Otahal did a great deal of the work presented, but there were tremendous contributions from many Technical Staff :

SPARC liaison and driving force Greg Weirs

Members of the 9326 Viz team including Alan Scott, Phil Smith, Dave Karelitz, Alex Pelletier, Warren Hunt, and Manoj Bhardwaj

IOSS mastermind Greg Sjaardema

Members of the SPARC community including (but not limited to): Travis Fisher, Derek Dinzl, Sam Browne, Jeff Fike

Members of the Sierra community including (but not limited to): Jesse Thomas, Sam Browne, Justin Lamb, Rebecca Nylen, Mark Merewether, Peter Grimmer, Douglas Vangoethem, George Orient

SOME ADDITIONAL CATALYST OUTPUT

LONGER MULTI-RESTART NALU RUN

Testing with Nalu

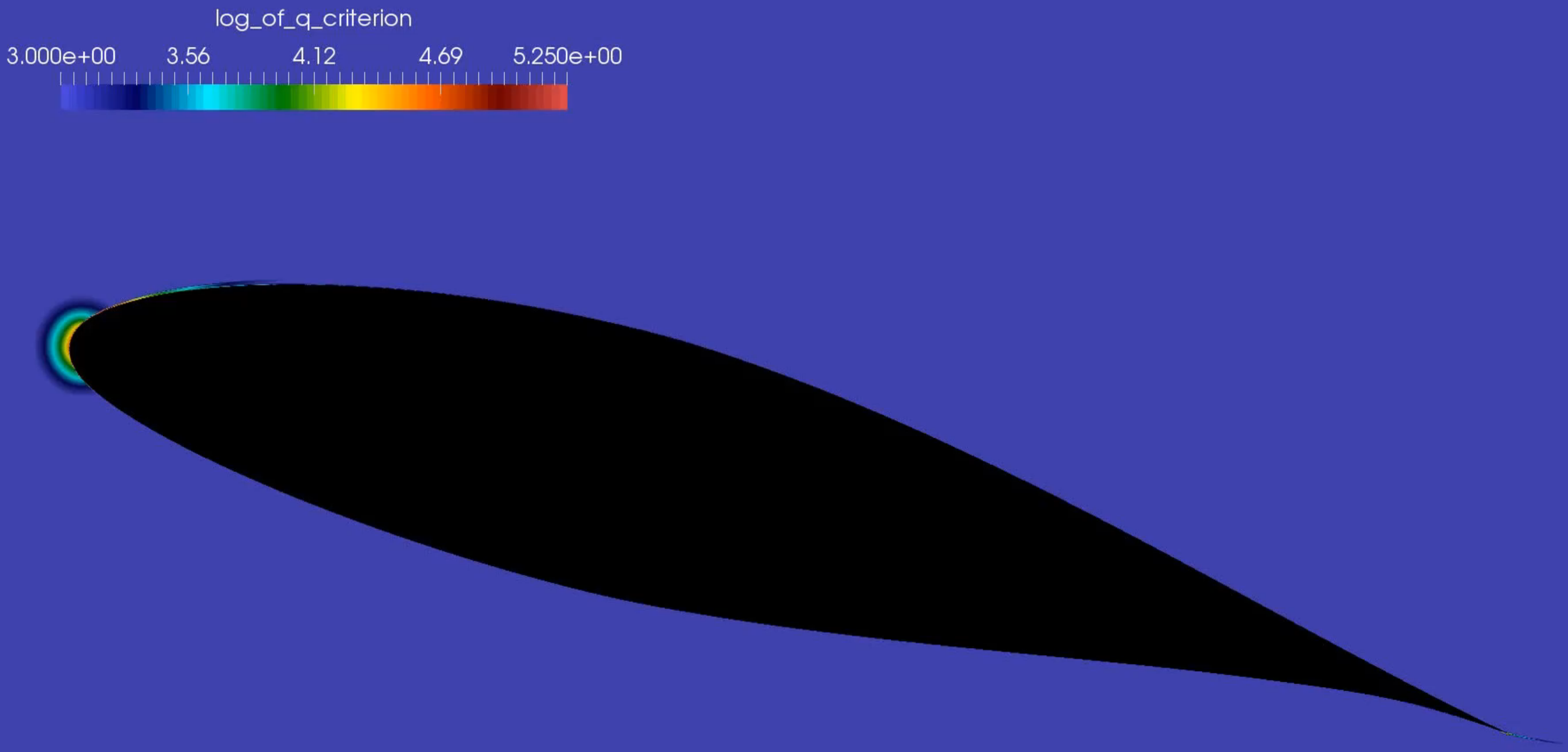
Wind turbine airfoil turbulence

Running on Mutrino, SNL small Trinity look-alike

2560 processors

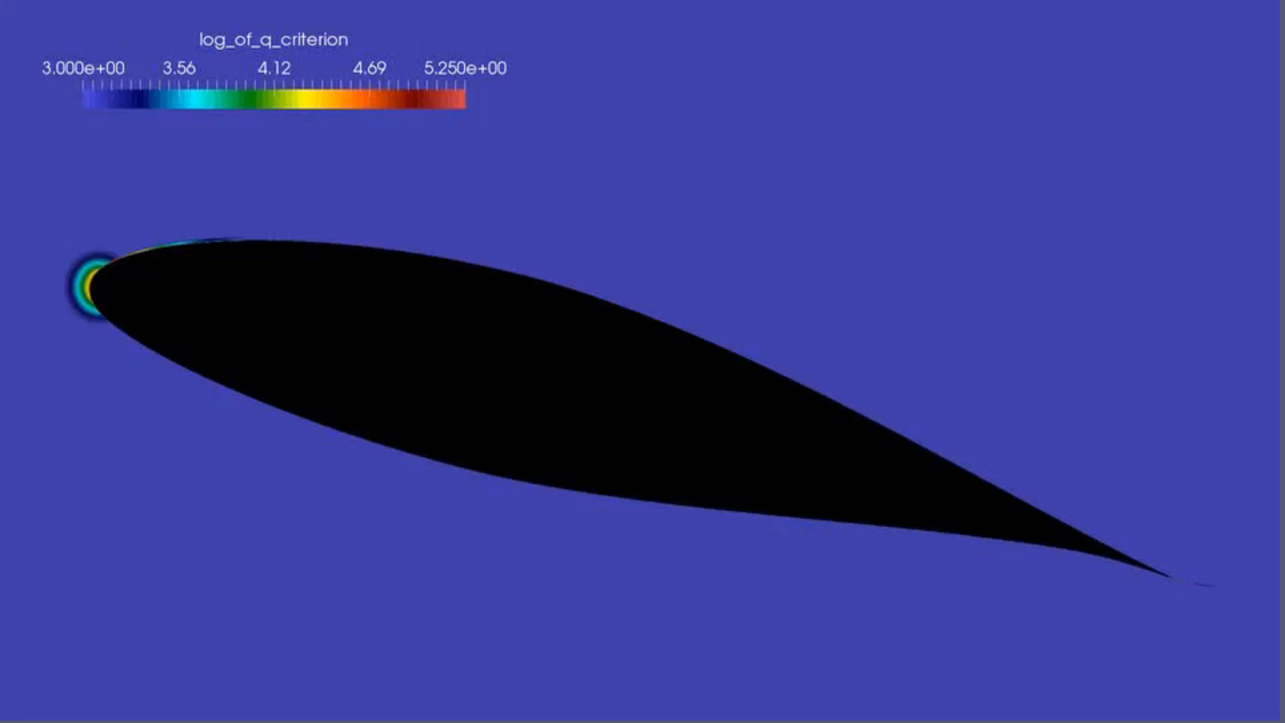
Around 50 overnight runs on Mutrino, primarily when system was not in heavy use by other analysts

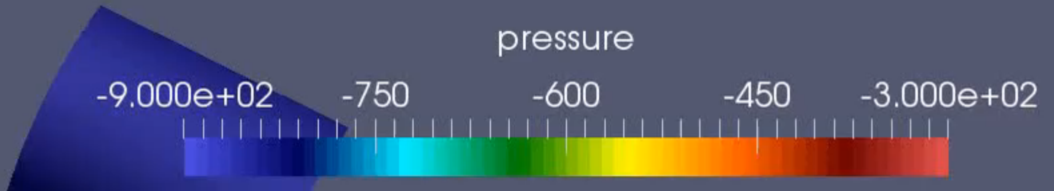
XY SLICE THROUGH MIDDLE
COLORED BY LOG BASE 10 OF Q-CRITERION


log_of_q_criterion
3.000e+00
3.56
4.12
4.69
5.250e+00

3D VIEW OF FRONT SECTION OF AIRFOIL
ISOSURFACE AT Q-CRITERION OF 1.0E6
COLORED BY PRESSURE

INCLUDES INSET OF XY SLICE COLORED BY
Q-CRITERION


log_of_q_criterion
3.000e+00
3.56
4.12
4.69
5.250e+00


pressure
-9.000e+02
-750
-600
-450
-3.000e+02