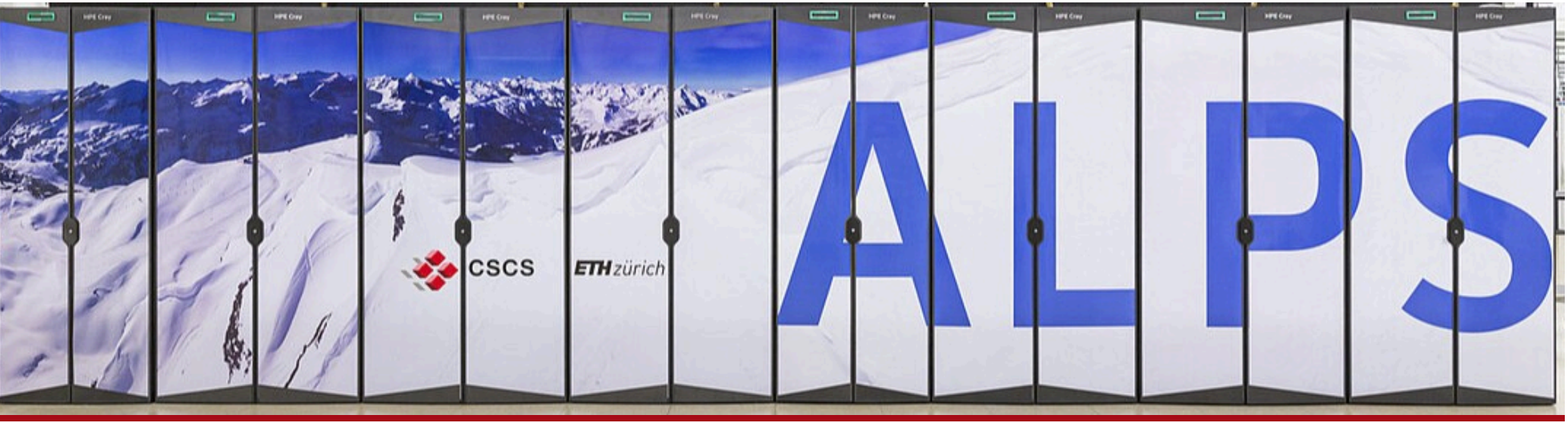
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Big Data Visualization: challenges, solutions, short-comings, and new ideas!

Dr. Jean M. Favre
Senior Visualization Software Engineer

HPMDV'26: First International Workshop on High Performance Massive Data Visualization

Take a step back and review several Visualization projects

- Fluid Dynamics
- Combustion
- Climatology
- Astrophysics

Looking forward and report on new developments

- Make a wish list
- Motivate the use of co-processing
- Highlight one showcase in Astrophysics

Big scientific data visualization

Characteristics:

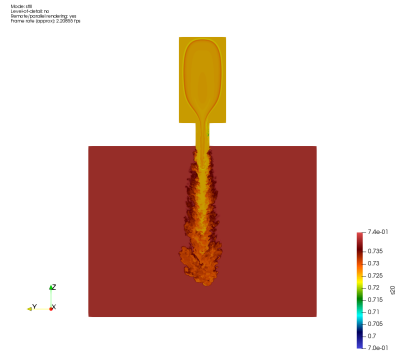
- 3D or n-D,
- Time-series,
- Unknown to the human eye, or
- An everyday reality (no artistic freedom allowed).

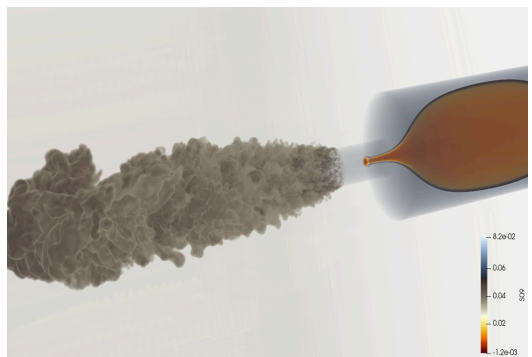
Simple recipes to deal with massive data:

- Subsampling,
- slicing

Drawbacks:

- miss finer details (temporal or spatial)
- Perception (and understanding!) will be reduced

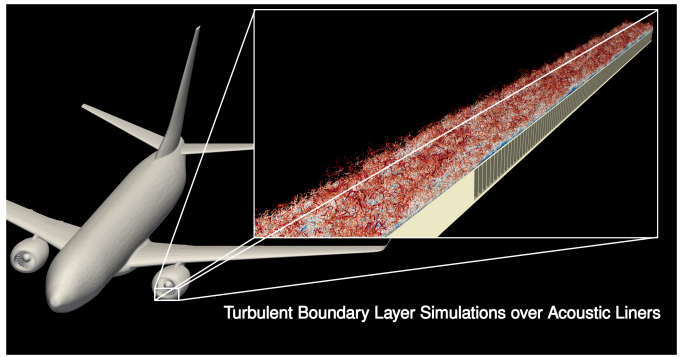


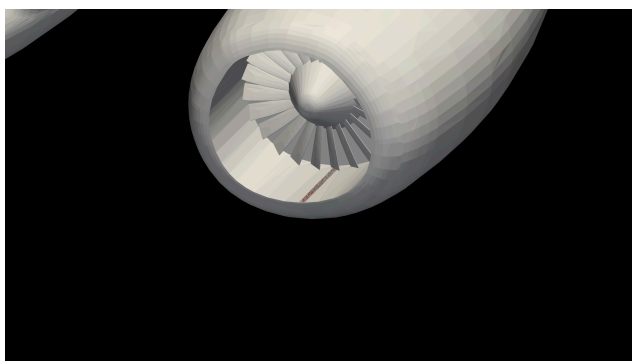


CFD: High resolution grids, many data I/O formats

- Regular structured grids (DNS, etc…) seem trivial to handle, but
- I/O interfaces cannot be taken for granted
  - I/O performance must be profiled!
  - I/O can be tuned
  - Use proven I/O libraries

CFD: High resolution structured grids: easy to read?

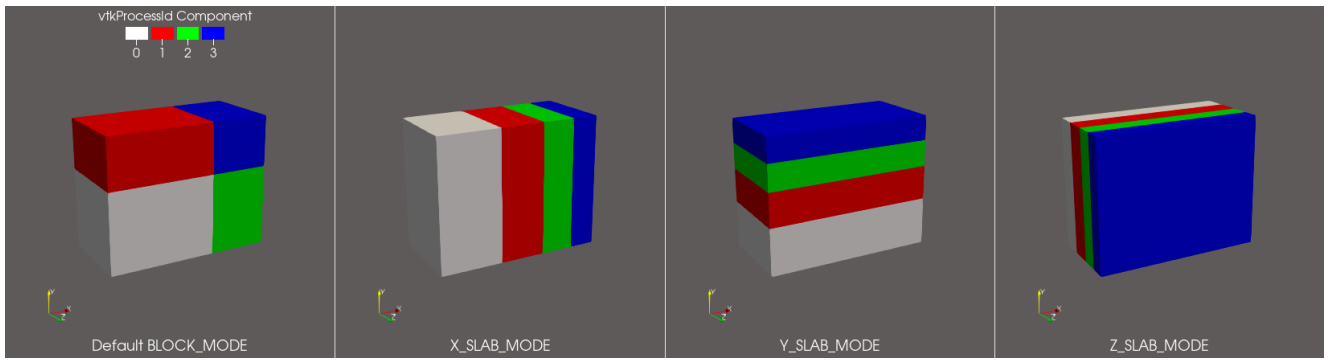




- Use HDF5 dataset and read via an Xdmf layer
- Grid dimensions are 21504 x 448 x 1120
- ParaView uses Kd-tree partitioning of the 3D space and all is well

- It works, but performance is very poor!

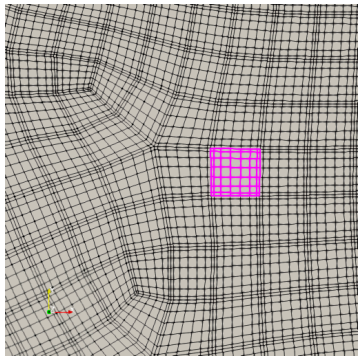
CFD: High resolution structured grids can be read much faster

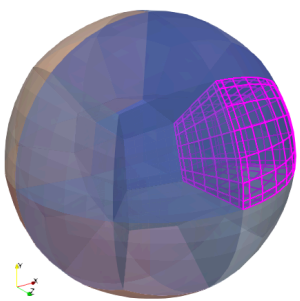


Override ParaView's default space-decomposition scheme

- Enforce Z-slab reading
- Provide non-fragmented I/O and a 10x increase in performance
- What about Kitware's new VTKHDF file format?

Combustion: High resolution spectral elements





Most visualization techniques require conversion to unstructured grids!

- See Nek5000-support-in-VTK-and-ParaView

Challenges:

To my knowledge, NVIDIA IndeX is still the only parallel volume renderer supporting unstructured data

Volume rendering animation

High resolution means finer details

- What a shame it would be to downgrade/undersample,

  or to take rendering shortcuts when we have a completely **natural** phenomenon under study.
- Take something as simple as **water**

Original waterfall

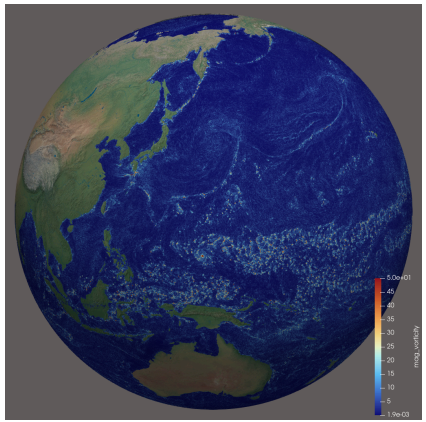
Natural waterfall

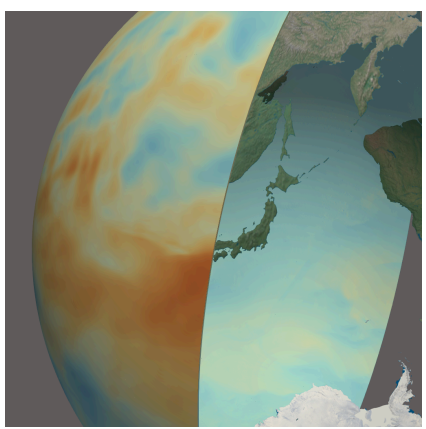
A whole Earth at a time: (Atmospheric data)

ICON 2.5 km resolution *atmospheric only*

- Spherical Shell
  - 83,886,080 triangles
  - 1 GH200 node (4 GPUs) with 8.8 GB per GPU
- 3D full model
  - 10,066,329,600 3D wedges (5 billion points)
  - 4 GH200 nodes (16 GPUs) with 275 GB per GPU

- Our ambition: Track hurricanes from 2D to 3D

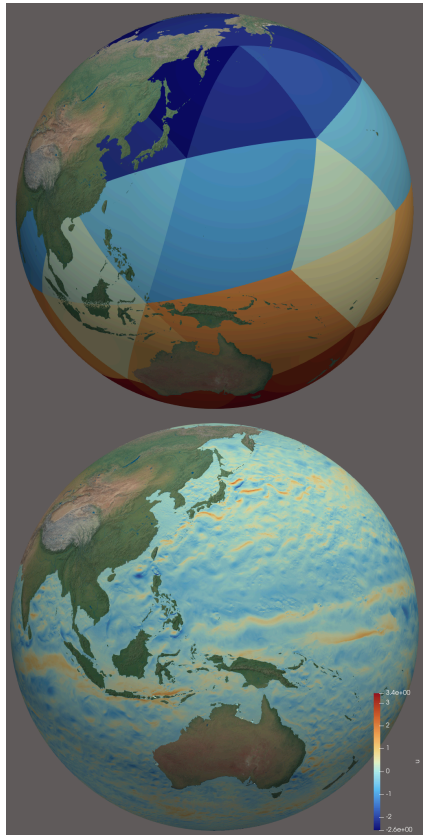




A whole Earth at a time: (Ocean circulation)

ICON 1.25 km resolution (R2B11) *ocean only*

- Spherical Shell
  - 237,416,516 triangles
  - A single GH200 node is required
- 3D full model
  - ????? 3D wedges
  - unable to read the 3D model with up to 128 nodes

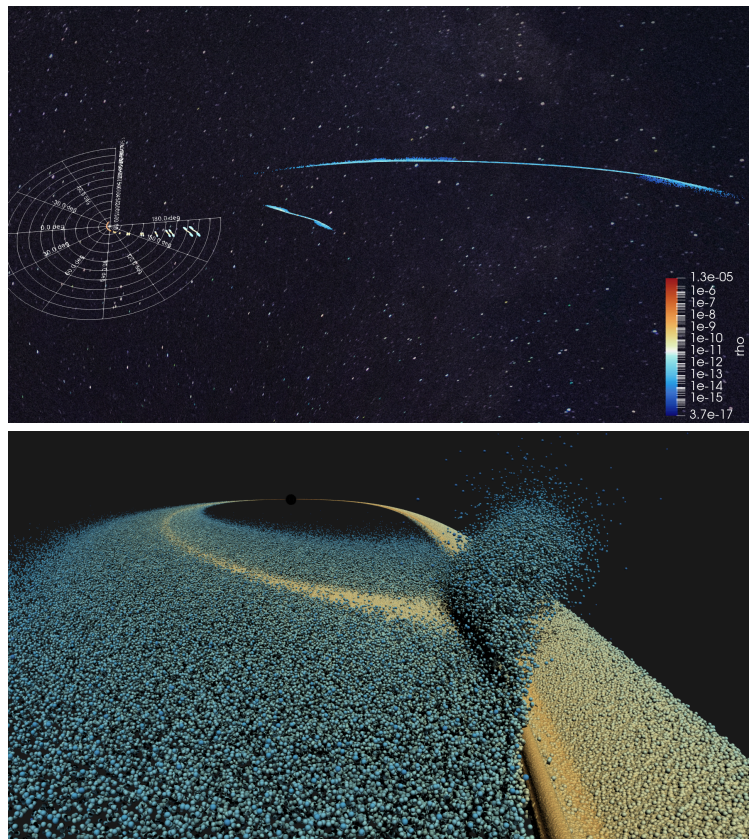


Beyond the Earth…to a Black Hole

A Tidal Disruption Event (TDE) was simulated on CSCS ALPS with a 10-billion particles dataset (code SPH-EXA)

Challenges of the post-hoc visualization:

- Data size: each snapshot is 807 GB on disk
- Poor temporal saving to disk
- Very large *spatial travel*
- Very large *object distortion*
- Camera tracking is hard
- Memory consumption is **non-negligible**

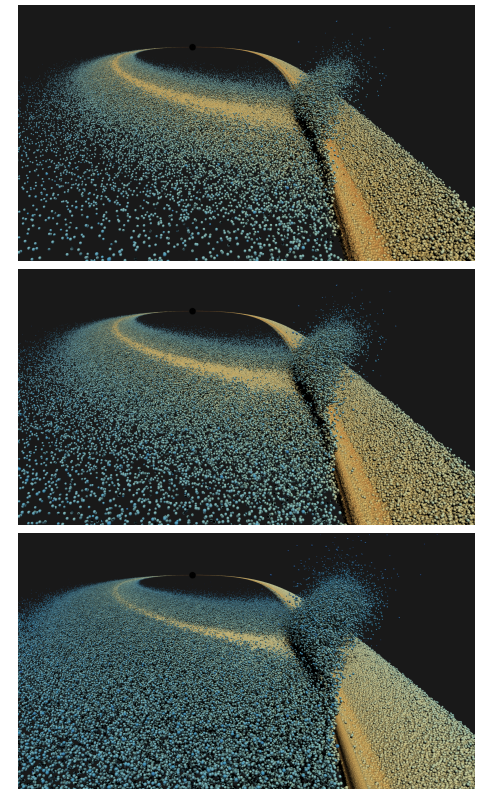


Recipes

- New I/O code:

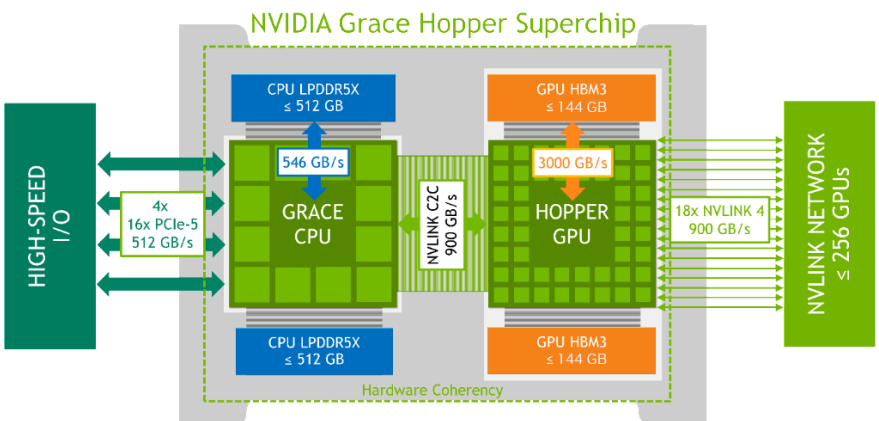
```
rho = timedata['rho'][begin:end].astype('f4')
polyVertex = vtk.vtkAffineIdTypeArray()
polyVertex.ConstructBackend(1,0)
polyVertex.SetNumberOfTuples(Number_of_Particles) verts =
vtk.vtkCellArray() verts.SetData(1, polyVertex)
output.SetVerts(verts)
```

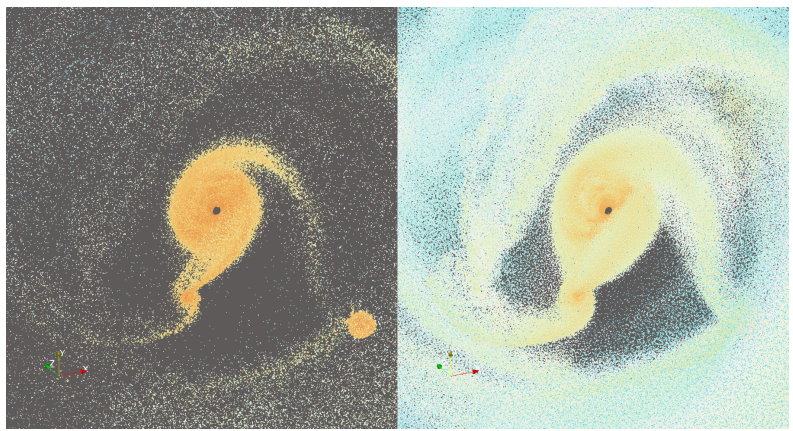
- Histogram sampling introduced in Ascent (later pushed to ParaView)
  - On the right: 1B, 2B and 10B particles
  - Very fast...for quick prototyping
  - but for the closer look, use full resolution
- Raytracing to gain 3d depth perception



Viskores Histogram Sampling

- Histogram sampling can be used with MPI-distributed data
  - in multi-threaded mode on the Grace CPU, or
  - with CUDA on the Hopper GPU
- Execution times are comparable, in the order of 3.5 seconds per 1.25 billion particles per device





The TDE simulated and visualized on CSCS ALPS

ParaView on NVIDIA GH200 nodes

- will re-visit this showcase when we discuss co-processing

Summary of our requirements

- super-fast, super-smart I/O
- No 2D cutting/slicing if we can (use full resolution)
- photorealistic rendering to provide
  - enhanced contrast
  - images as close to the reality

How have we worked in all these past years?

- prototyped visualization scenarios from *lower* to *full* spatial resolution
  - e.g: We have ParaView Python scripts ready
- Learned the intricate details of file I/O (seek-and-read...)

- What if we could re-use all of this *know-how* ?

Adopt new processing workflows

- Eliminate the slowest part of the workflow (disk I/O)
- Promote in-situ visualization workflow!

LLNL Ascent

```
https://ascent.readthedocs.io/en/latest/index.html
```

Kitware ParaView Catalyst

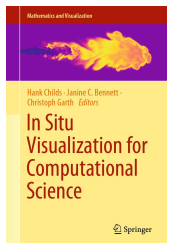
```
https://kitware.github.io/paraview-catalyst/
```

Viskores: Accelerating the Visualization Toolkit for Massively Threaded Architectures

```
https://viskores.readthedocs.io/en/latest/
```

ADIOS2 and Fides for in-transit visualization

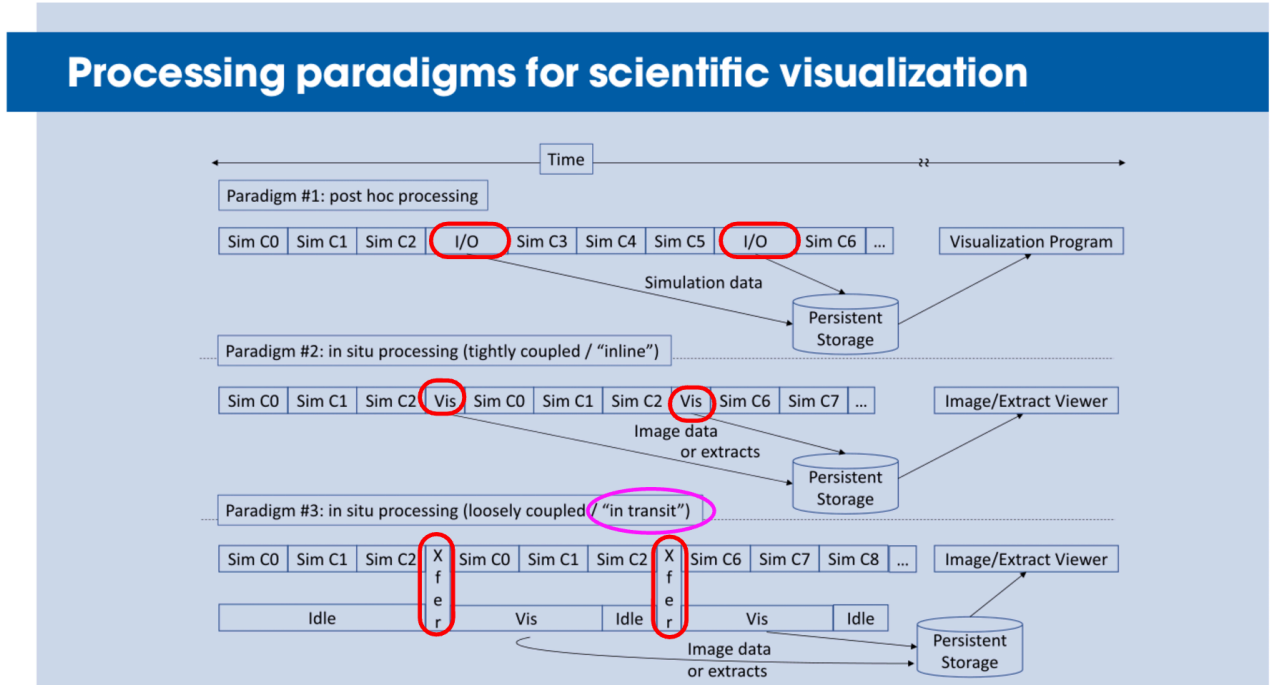
What is in-situ visualization ?



In Situ Visualization for Computational Science

"Oak Ridge National Laboratory saw three generations of leading supercomputers-Jaguar (2009) to Titan (2012) to Summit (2018)- yield a 100X increase in computer power (from 1.75 petaFLOPS to more than 175 petaFLOPS), but only a 10X increase in filesystem performance (from 240 GB/s to 2.5TB/s)" (from the book)

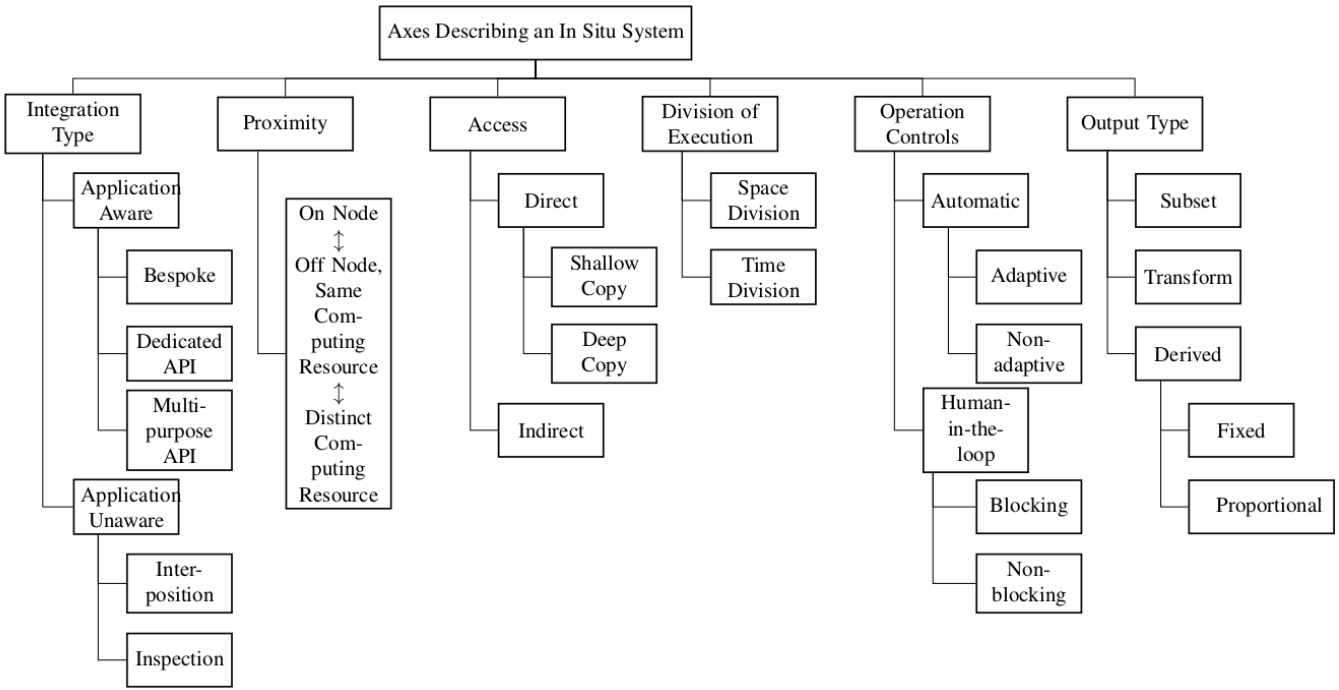
in-situ and in-transit visualization workflows



In-situ visualization raises quite a few challenges

- Sharing physical resources and domain decomposition?
- What % of time can we afford to "do visualization" vs. "advance the solver"?
- Which feature extraction and visualization tasks are best suited for on-the-fly processing?
- Can we provide a generic abstraction to describe the data and mesh structures?

In situ workflows have multiple dimensions



Need to start from the simulation code internal data model

Provide conventions to provide data bridges promoting zero-copy memory space sharing

- **Conduit**, from LLNL provides an intuitive model for describing hierarchical scientific data in C++, C, Fortran, and Python. It is used for data coupling between packages in-core, serialization, and I/O tasks.
- `Ascent` and `Catalyst` use `Conduit` for describing data and other parameters which can be communicated between a simulation and the visualization apps.
- **Fides** enables workflows to seamlessly integrate simulation and visualization by providing a data model in JSON that describes the mesh and fields.
- Using this data model, Fides maps `ADIOS2 data arrays` (from files or streams).

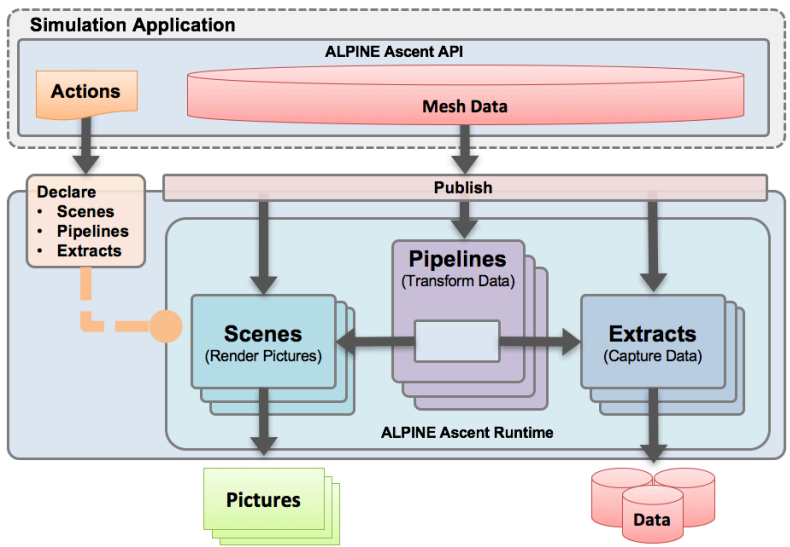
Ascent: an *in situ* visualization and analysis library

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

- Supports: Making Pictures, Transforming Data, and Capturing Data for use outside of Ascent
- Provides a simple infrastructure to integrate custom analysis
- Provides C++, C, Python, and Fortran APIs
- References:
  - Ascent: A Flyweight In Situ Library for Exascale Simulations, M. Larsen et al., 2022
  - The ALPINE in situ infrastructure: Ascending from the ashes of strawman, M. Larsen et al.

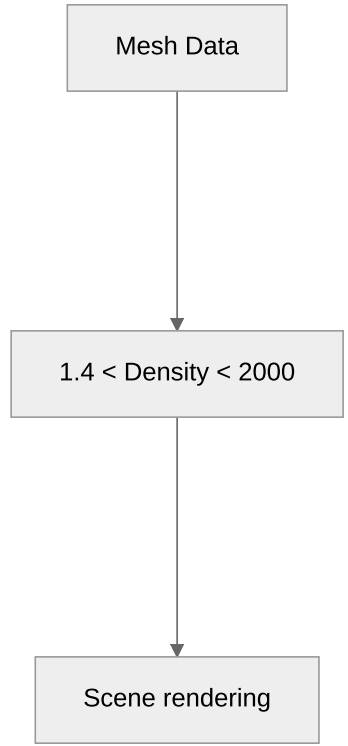
Ascent

- The Conduit Mesh Blueprint
- Runtimes providing analysis, rendering and I/O
  - Runtimes will execute a number of `actions`, defined by Conduit Nodes
- Data Adaptors (internal)



Ascent's programming interface is YAML-based, interpreted at run-time

```yaml
-
  action: "add_pipelines"
  pipelines:
    pl1:
      f1:
        type: "threshold"
        params:
          field: "Density"
          min_value: 1.4
          max_value: 2000.0
-
  action: "add_scenes"
  scenes:
    s1:
      plots:
        p1:
          type: "pseudocolor"
          pipeline: "pl1"
          field: "Density"
```



ParaView Catalyst: an in-situ API with support for an ABI interface

ParaView-Catalyst is an implementation of the Catalyst `in situ API` that uses ParaView for data processing and rendering.

ParaView-Catalyst supports a subset of the **Mesh Blueprint**.

The ParaView Catalyst Python scripts

- An interactive session with the ParaView application with a representative template input file, and a set of visualization filters can be tuned by the user in an *offline* fashion [not connected to a running solver]
- The Python scripts are completely interchangeable between the batch-mode ParaView execution (reading data from disk), and the in-situ execution

ParaView Catalyst pipeline vs. Ascent pipeline

```python
# ParaView Python pipeline

renderView1 = CreateView('RenderView')

selection=SelectPoints()
selection.QueryString = "Density ⩾ 1.4"

extractSelection = ExtractSelection()
thresholdDisplay = Show(extractSelection)
ColorBy(thresholdDisplay, ['POINTS', 'Density'])

pNG1 = CreateExtractor('PNG', renderView1)
pNG1.Trigger = 'TimeStep'
pNG1.Writer.FileName =
    'threshold_{timestep:06d}{camera}.png'
pNG1.Trigger.Frequency = 100
```

```yaml
// Ascent YAML pipeline
-
  action: "add_scenes"
  scenes:
    s1:
      plots:
        p1:
          type: "pseudocolor"
          field: "Density"
          pipeline: "pl_threshold"
      renders:
        r1:
          image_prefix: "datasets/threshold_.%05d"
-
  action: "add_pipelines"
  pipelines:
    pl_threshold:
      f1:
        type: "threshold"
        params:
          field: "Density"
          min_value: 1.4
          max_value: 2000
```

in short

- *ParaView Catalyst* and *Ascent* use *Conduit*-based meta-data descriptions
- Usability is somewhat similar, and they share the main drawback:
  - An N-to-N mapping from simulation tasks to visualization tasks
  - but viz. algorithms most often don't scale with the same granularity of simulations
  - The simulation waits while visualization steps take place.

N.B.

- The 10-billion particle Tidal Disruption Event ran on 16 GH200 nodes
- Raytraced images ran on 2 GH200 nodes (4 samples/pixel and shadows) not using the GPUs, but using all 72 threads of each Grace CPU.

Both workflows have demonstrated giga,tera-scale support

What is new?

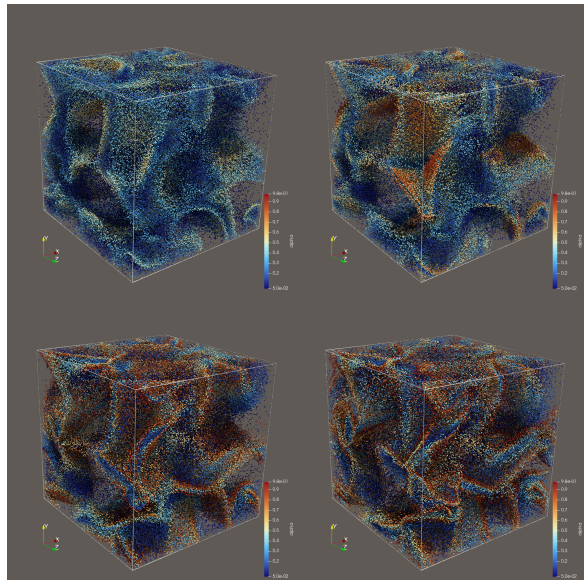
- GPU-resident data support (with zero-copy) was missing in *ParaView Catalyst* when I last presented at the ISC25 in-situ visualization workshop

```
Favre, J.M., Piccinali, JG. (2026). Issues and Challenges of Deploying in-Situ Visualization for SPH Codes.
In: Neuwirth, S., Paul, A.K., Weinzierl, T., Carson, E.C. (eds) High Performance Computing. ISC High Performance 2025.
Lecture Notes in Computer Science, vol 16091. Springer, Cham.
https://doi.org/10.1007/978-3-032-07612-0_8
```

- It is now becoming a reality, thanks to Kitware's great commitment!

Our first public claim of GPU-resident in-situ ParaView support

- Tested two demonstrators from the 2025 CSCS CUDA Summer University program
  - ParaView Catalyst (version > v6.0.1)
- Got ambitious and moved to a full HPC application: SPH-EXA CUDA-enabled
  - Catalyst Adaptor is ~ 100 lines of code
  - adopted vtkAffineIdTypeArray
  - some rendering options are still an issue
  - standard SPH-EXA benchmarks with up to 512M particles on multi-node, multi-gpu configurations



Moving from N-to-N to N-to-M process allocations

ADIOS

- `AD` aptable `I` nput `O` utput `S` ystem: summary
  - Extreme scale I/O: `YES!`
  - A file-only I/O library: `NO!`
- ADIOS Engines:
  - BP5
  - Sustainable Staging Transport (SST)

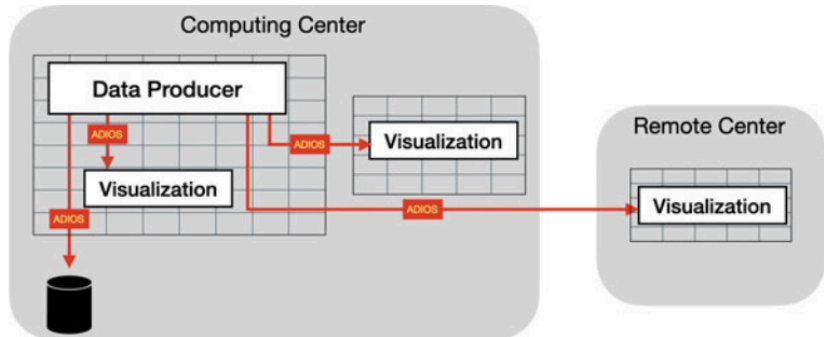


Figure taken from `The Adaptable IO System (ADIOS)`, book chapter in In Situ Visualization for Computational Science

ADIOS I/O Abstraction and the SST Engine

- *Variables* (n-dimensional distributed arrays of a particular type)
- *Attributes* (labels associated with individual variables or the entire output data set).
- *Steps* specify when the data is available.

- There is nothing in the ADIOS interface that prescribes how to handle the data

- SST allows direct connection of data producers and consumers
- The buffering policy is configured at run-time,
- Readers and writers `do not necessarily move in lockstep`,
- `One or the other side can wait for data to be produced or consumed`, or `data may be dropped`
- SST supports full M-to-N data distribution

Data bridge semantics is provided by Fides

- Fides is an *ADIOS Schema* providing the meaning of each data array, and the relationship between groups of arrays, e.g.:
  - Coordinates arrays
  - Connectivity arrays
- Fides is a library mapping ADIOS2 data arrays to Viskores datasets.
- Simulations already using ADIOS2 do not need to make any changes to the way their data is written/streamed by ADIOS.

Fides: an ADIOS Schema

- Support for pre-defined grid types has:
  - Uniform Grid
  - Rectilinear Grid
  - Unstructured Grid
  - Single cell type Unstructured Grid
  - XGC

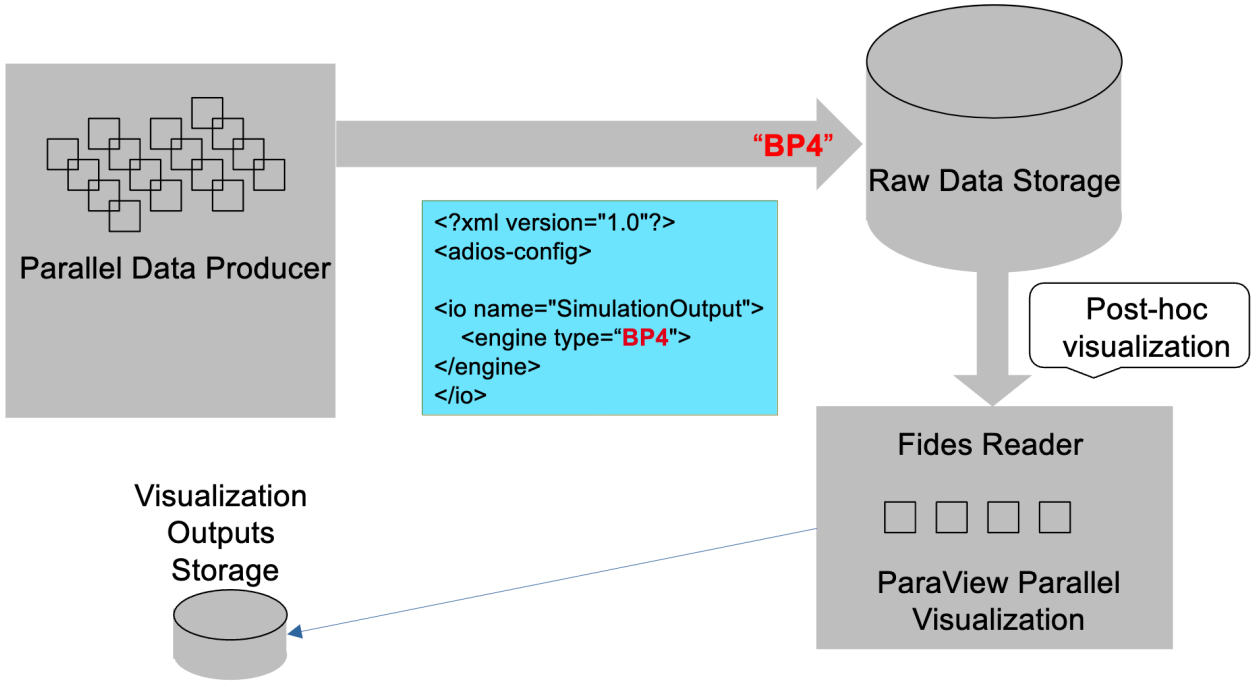
Uniform Data Model

The data model uses uniform point coordinates for the coordinate system, needing the origin and spacing to be specified. The cell set is structured based on the dimensions of the data.

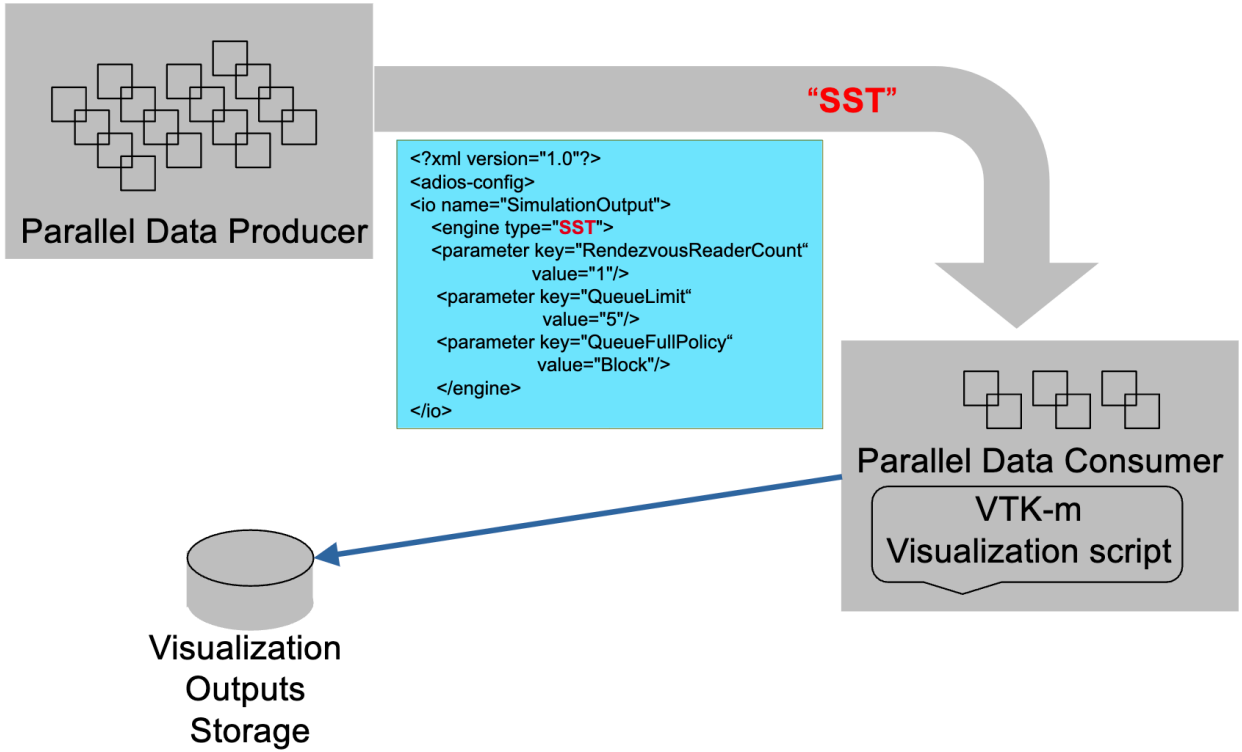
Attributes

Attribute Name	Possible types/values	Def
`Fides_Data_Model`	string: `uniform`	non
`Fides_Origin`	3 integer or floating points	non
`Fides_Spacing`	3 integer or floating points	non
`Fides_Dimension_Variable`	string: name of variable to use for determining dimensions	non

Post-hoc Visualization with ADIOS2 and Fides



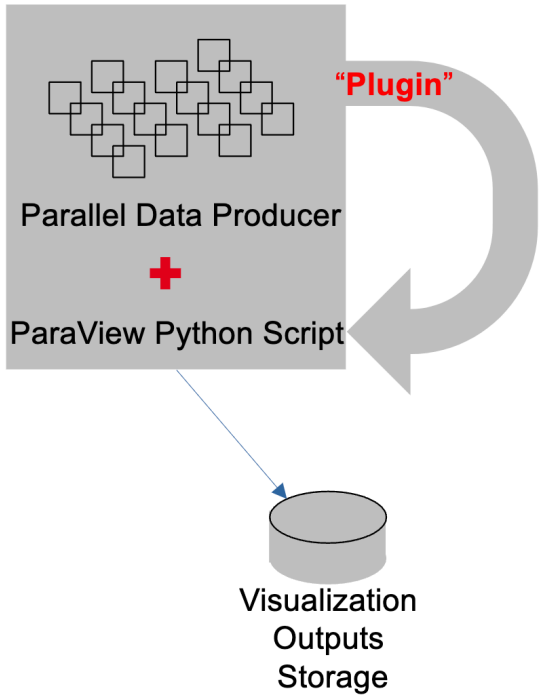
In-transit Visualization with ADIOS2



Using and ADIOS Plugin is yet another alternative

- ADIOS has the ability for users to load their own engines and operators through plugins
- As of v2.9, ADIOS has a ParaView `plugin`
  - Uses Catalyst and the Fides JSON description
  - Uses the `inline` engine
  - The Inline engine provides in-process communication between writers and readers, avoiding the copy of data buffers.
  - This engine is focused on the N-to-N case

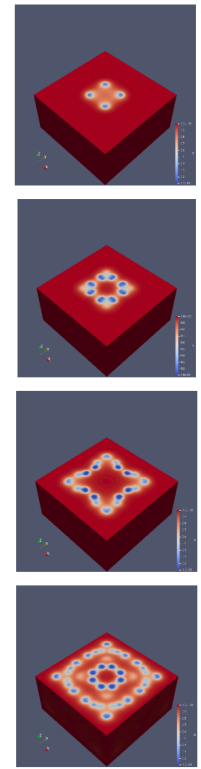
In-situ Visualization with ADIOS2



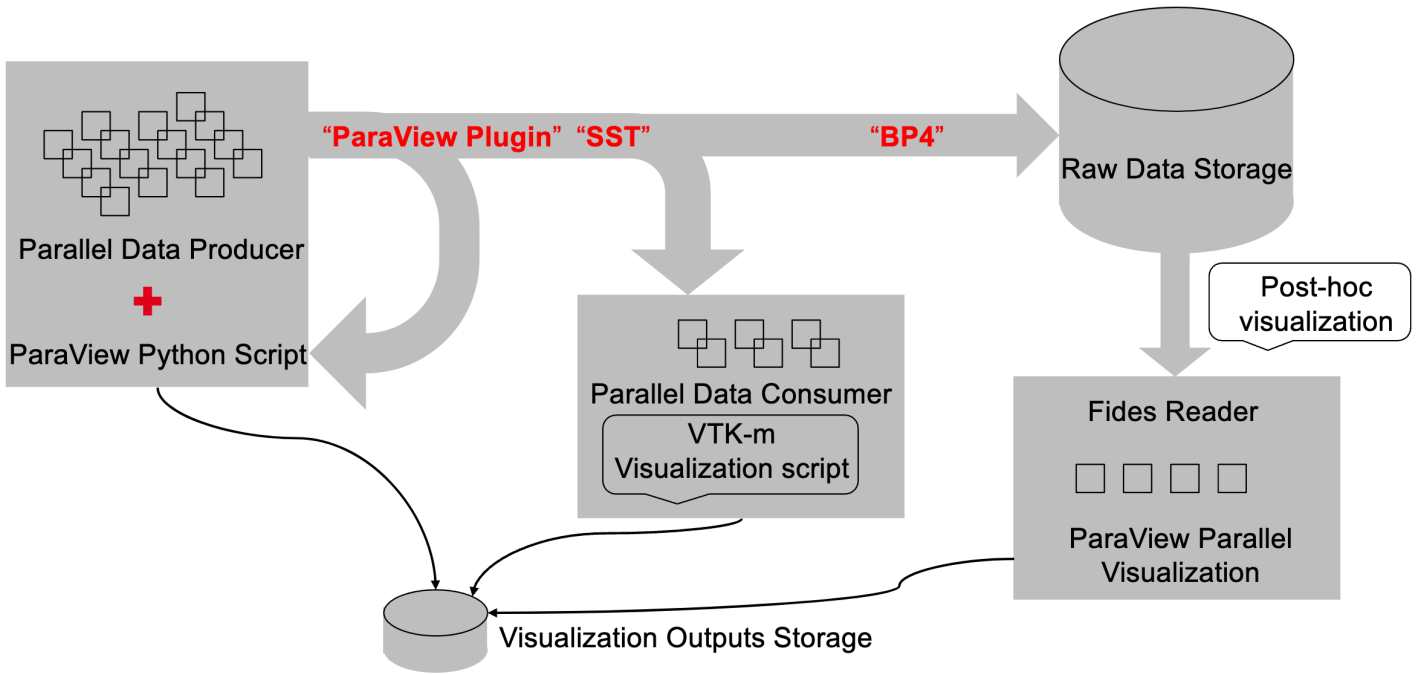

```
<?xml version="1.0"?>
<adios-config>

 <io name="SimulationOutput">
      <engine type="plugin">
      <parameter key="PluginName" value="fides"/>
      <parameter key="PluginLibrary"

value="ParaViewADIOSInSituEngine"/>
          <!-- ParaViewFides engine parameters -->
          <parameter key="DataModel“
                        value="gs-catalyst-fides.json"/>
          <parameter key="Script“
                        value=“pvParaViewScript.py"/>
      </engine>
   </io>
```



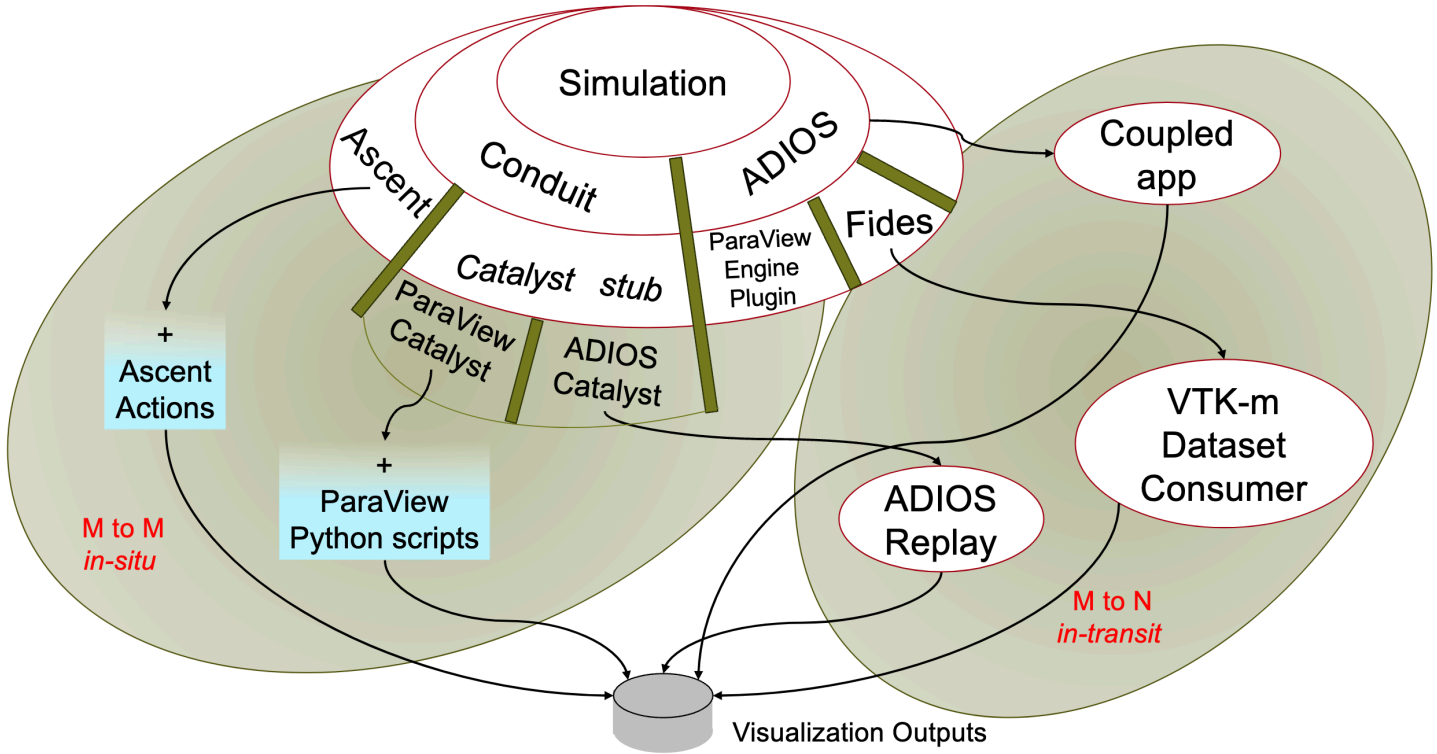
Run-time selection for multiple options for Visualization



Process tuning is mandatory everywhere

- Reviewed definitions of `post-hoc`, `in-situ`, and `in-transit` visualization
- Introduced two Conduit-based solutions providing filtering and rendering engines.
  - `ParaView Catalyst`, and
  - `Ascent`,
- Introduced `ADIOS` and `Fides`
- Focused on the ADIOS ParaView engine plugin
- Focused on the ADIOS SST engine to drive, either
  - A Viskores visualization and rendering program, or
  - A Catalyst Python program for ParaView
- The `in-transit` concept needs tuning, to properly balance compute and viz resources

Summary


Simulation
Conduit
Ascent
ADIOS
Fides
ParaView
Engine
Plugin
Catalyst stub
ParaView
Catalyst
ADIOS
Catalyst
Coupled
app
+
Ascent
Actions
+
ParaView
Python scripts
VTK-m
Dataset
Consumer
ADIOS
Replay
M to M
in-situ
M to N
in-transit
Visualization Outputs

Summary and Q&A

ワークショップの主催者の皆様、ご招待いただきありがとうございます。

and Thanks to the audience