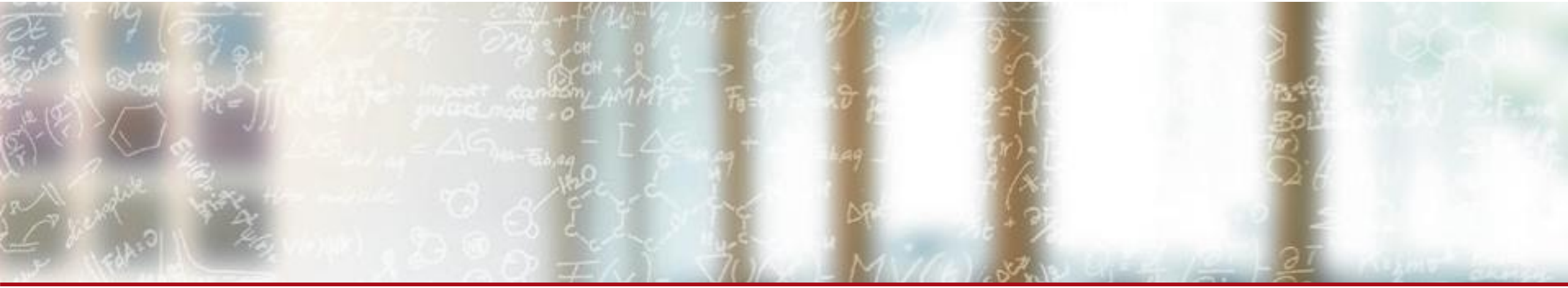
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



in-situ visualization solutions come to the rescue

PASC 2023

Jean M. Favre, CSCS

June 27, 2023

Abstract

Scientific visualization and analysis are key ingredients in HPC simulation workflows. For decades, the dominant paradigm has been post-hoc visualization; simulation codes iterate and save files to disk, giving the domain scientists the opportunity to read the data back at a later time. However, in recent years where data volumes are growing at an ever-diverging rate as compared to I/O subsystems, this traditional paradigm has been greatly stressed out. In-situ processing helps mitigate these I/O bottlenecks, enabling simulation and visualization calculations to run in-memory, at higher spatial and temporal resolution, while the simulation is running, limiting, or avoiding the transfer of raw data to disks. We will introduce two open-source libraries, “ParaView Catalyst” and “Ascent”, enabling in-situ data analysis and visualization. In-situ libraries can analyze, and transform data, render images, and export results in real-time, but require careful planning because we miss the exploratory, “what-if” analysis process enabled by the interactive visualization applications we are accustomed to. Doing in-situ visualization represents a paradigm shift that will impact the scientist's workflow. We will summarize our early adoption of in-situ visualization scenarios with examples from different simulation fields.

Outline

- Motivations
- A brief taxonomy of in-situ visualization solutions
- Remarks about file formats
- Introduction to Conduit
  - Conduit + Catalyst
  - Conduit + Ascent
- Introduction to ADIOS and Fides
  - Engines (in particular SST)
  - Adios plugins (a ParaView plugin)
- Catalyst + Adios + SST => Catalyst + ParaView
- Summary

Motivation

- What is *in-situ* visualization, why do we need it? What solutions are available to implement it?
- See recent book "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 cited above)

Post-hoc visualization

- For decades, the dominant paradigm has been *post-hoc* visualization
- Simulation codes iterate, and save data at regular time intervals.
  - Visualization and domain scientists can then read the data back from storage and interactively explore the data without time constraints

"Without I/O, no visualization is possible"

The true cost of doing I/O is an aggregate of the solver's I/O phase and the many iterations of visualization sessions.

Post-hoc visualization

Even if scientists could afford to keep most of the data for analysis, they must transfer the data to a machine with sufficient capacity and processing power:

➔ Very high data transfer
➔Visualization machine needs to be almost as powerful as the supercomputer
➔The alternative: use smaller temporal and spatial subsets

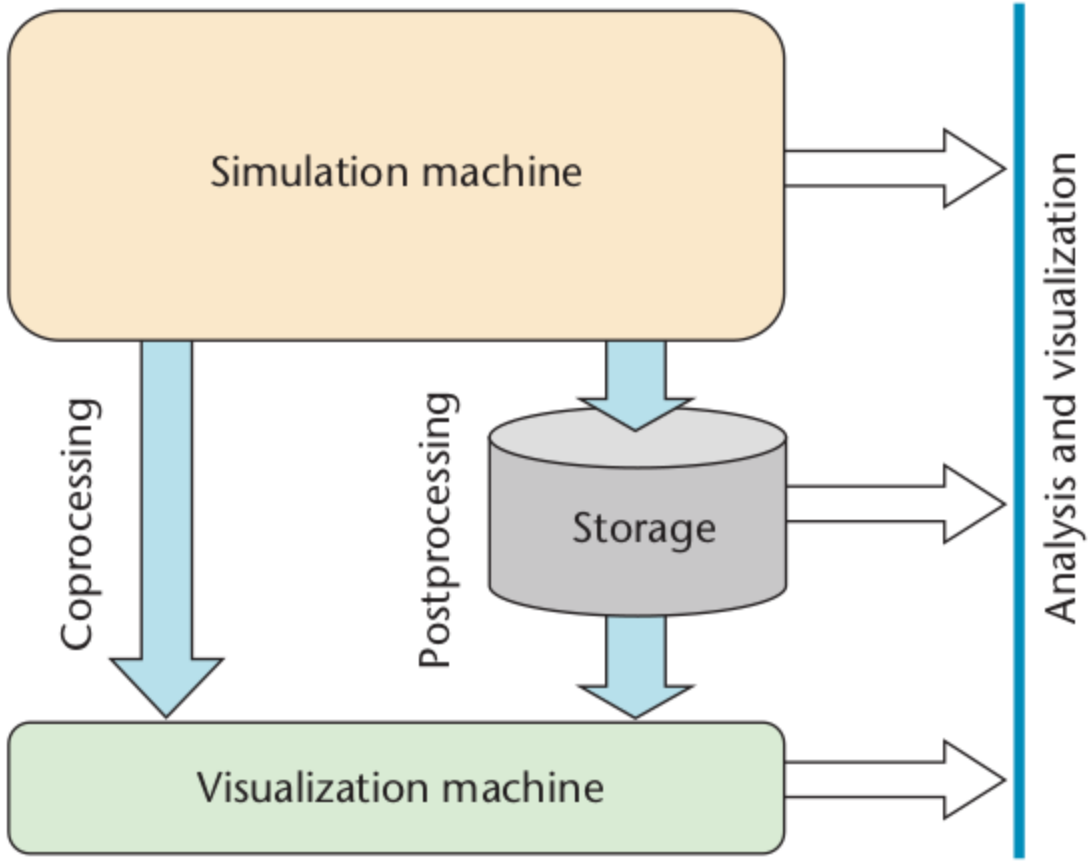
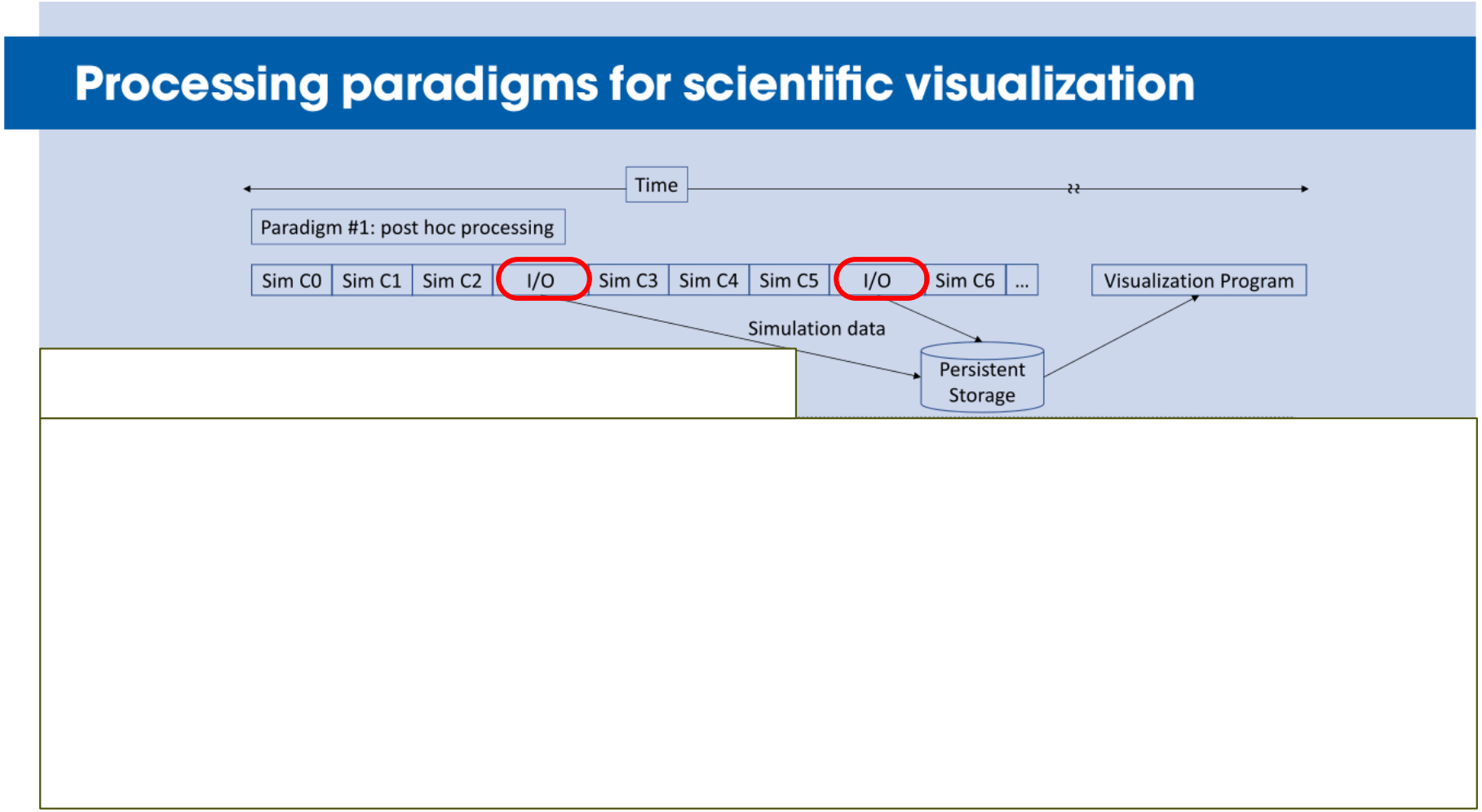


Figure taken from "*In Situ Visualization at Extreme Scale: Challenges and Opportunities*", Kwan-Liu Ma, IEEE CG&A, nov/dec 2009

Post-hoc visualization



in situ visualization

Instrument the code such that both the simulation and visualization calculations run on the same hardware

This runtime co-processing can render images directly or extract features -- *which are much smaller than the original raw data*

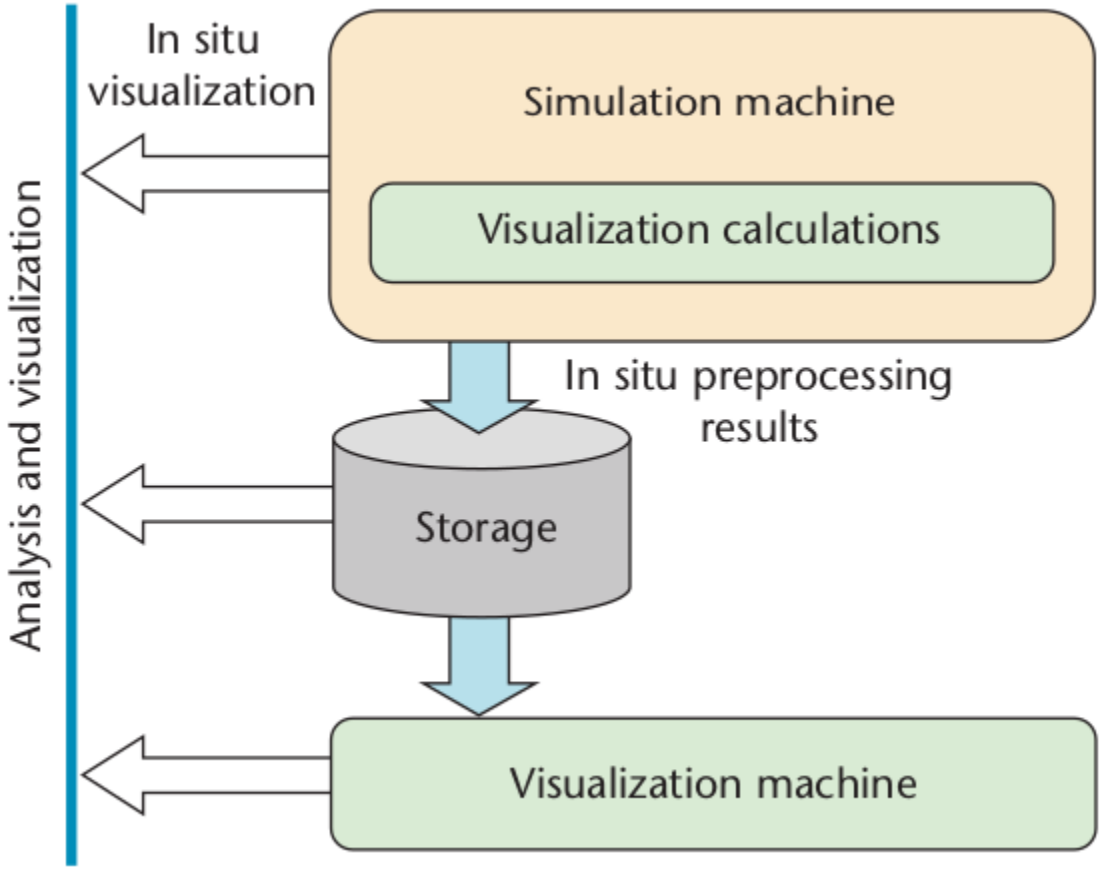
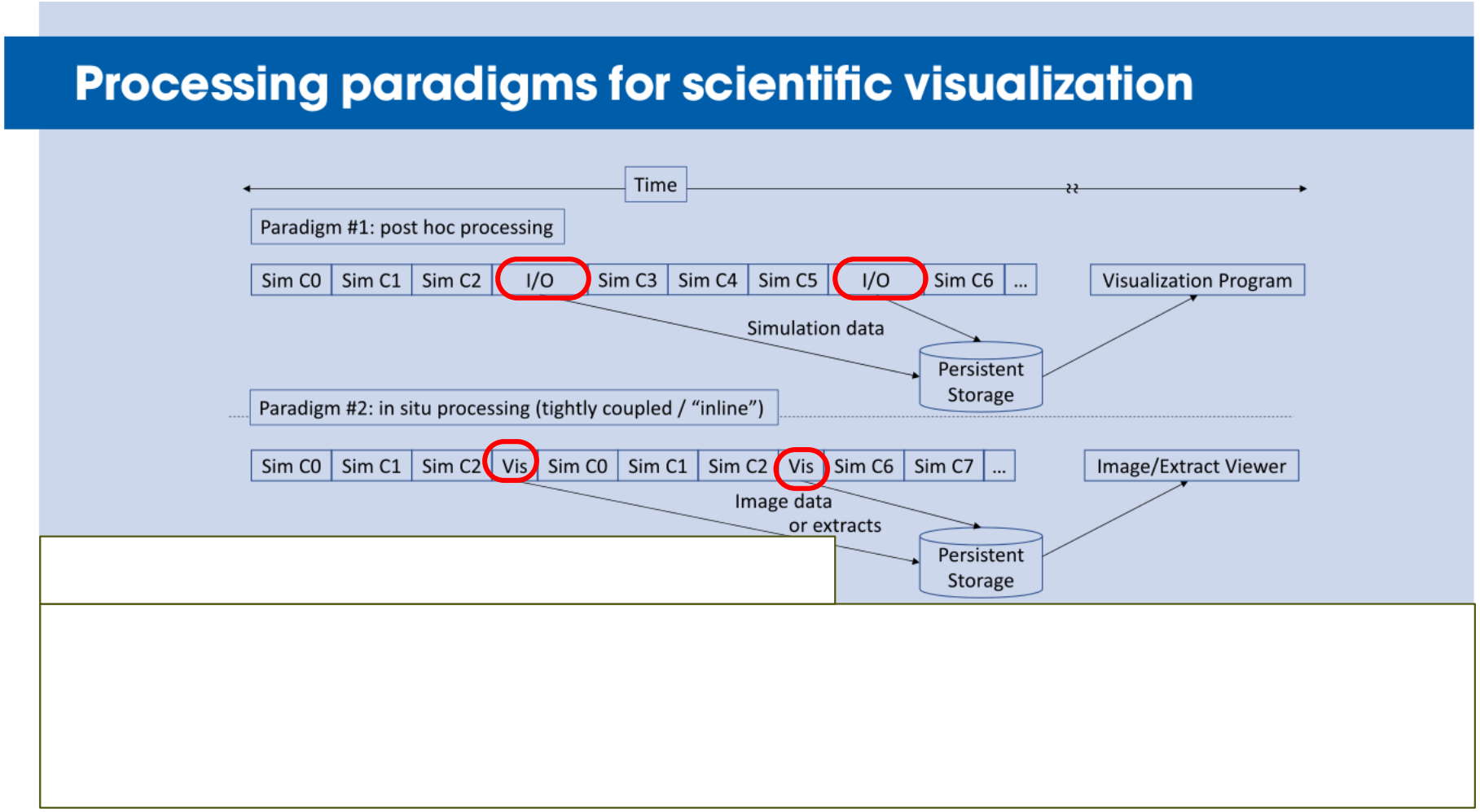


Figure taken from "*In Situ Visualization at Extreme Scale: Challenges and Opportunities*", Kwan-Liu Ma, IEEE CG&A, nov/dec 2009

In-situ visualization



In-situ visualization has raised quite a few questions

- 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?
- Since less data would be effectively stored to disk, should we augment it with ancillary data?
- Can we provide a generic abstraction to describe the data and mesh structures?

A third paradigm also emerged: in-transit visualization

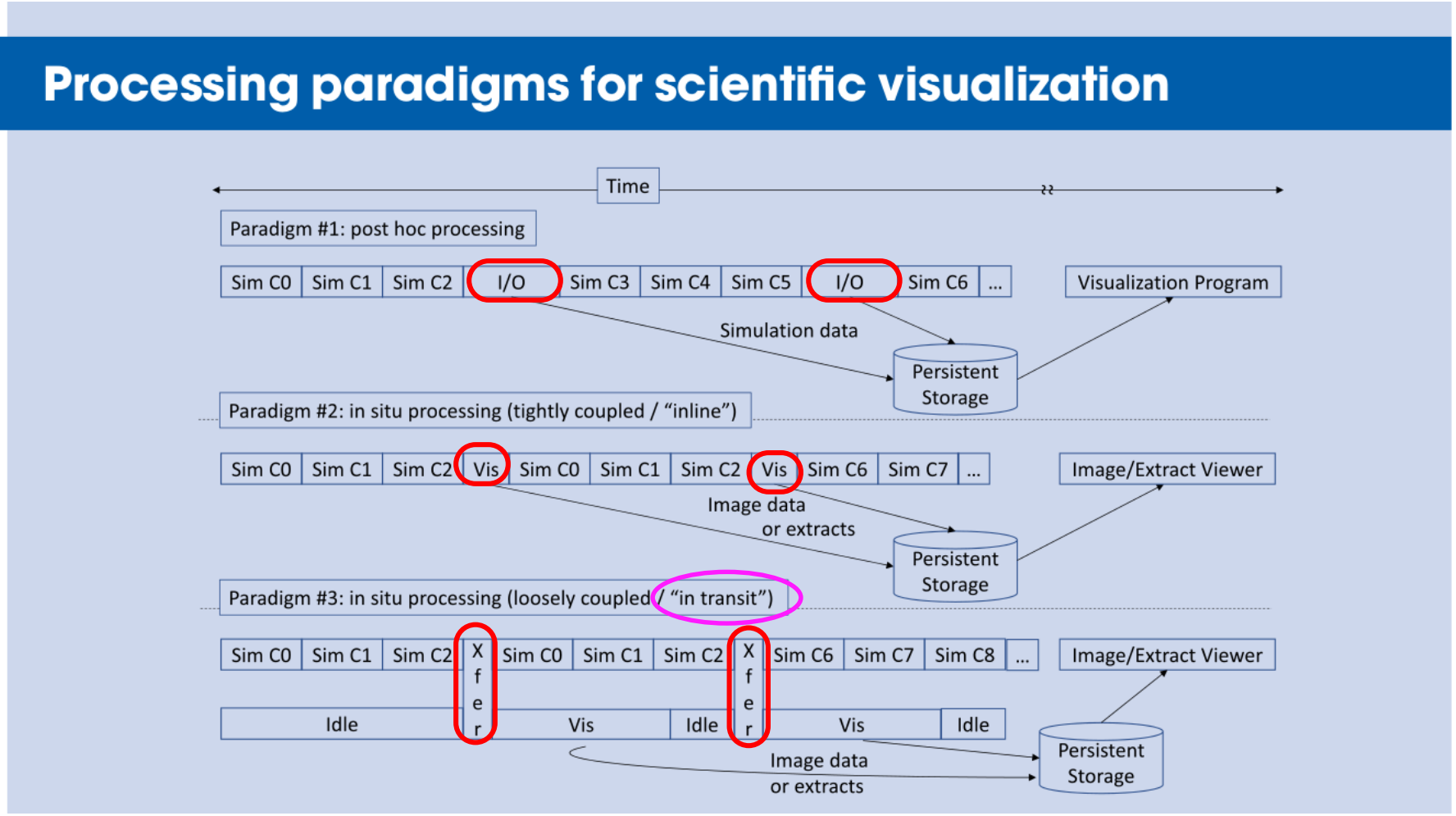


Figure taken from "*In Situ Visualization for Computational Science*", Hank Childs et al., IEEE CG&A, nov/dec 2019

Many definitions and colloquial use for “in-situ”

Overall motivation: process data in the processor’s memory space, without touching the disks, even if data is moved to a distinct set of resources (in-transit)

- Co-processing, concurrent processing, run-time visualization, “in-situ”, “in place”, etc..

- "A Terminology for In Situ Visualization and Analysis Systems“, Hank Childs et al, International Journal of High Performance Computing Applications, 34(6):676–691

http://cdux.cs.uoregon.edu/pubs/ChildsIJHPCA.pdf

- For the scope of this paper, “in situ processing” was defined to be:

“**processing data as it is generated**”

in situ systems are described via multiple, distinct axes

- Integration type

  How visualization and analysis code is integrated with the simulation code?

- Proximity

  How close is the visualization code from the data?

- Access

  How does the simulation give access to the data?

- Division of execution:

  how compute resources are shared between simulation and in situ routines.

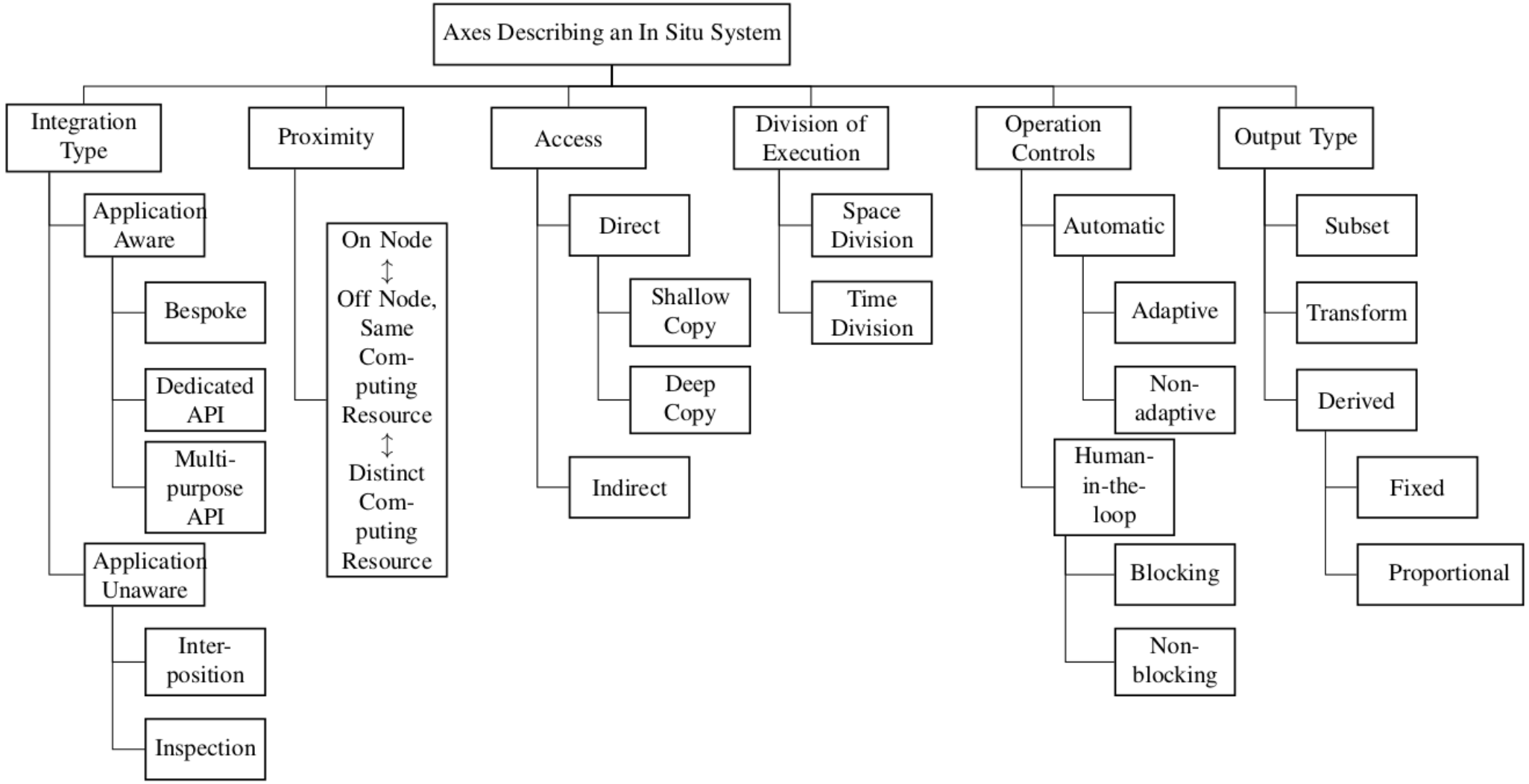
- Operation controls:

  the mechanism for selecting which operations are executed during run-time

- Output type

  which types of operations are performed on the simulation data before it is output.

(fig taken from the paper)



What about my [the solver] internal data model?

Do the standard visualization apps support all the data structures I use in my code?

- Use Data Adaptors (e.g. NEK5000 spectral code => converted to hexahedra in a VTK plugin)
- File formats in traditional post-hoc visualization *most-often* preserve, document, the nature of the grid. See for example the family of VTK XML-based file formats (*.pvti, *.pvtu, *.pvtp, …)
- Some others don’t.
- Some I/O libraries require *ad-hoc* conventions for proper data discovery (e.g. HDF5 Gadget)

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Introduction to Conduit

Conduit: Simplified Data Exchange for HPC Simulations

- Conduit is an open source project from Lawrence Livermore National Laboratory that 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.

- Conduit provides a convention to describe computational simulation meshes. This is called the Mesh Blueprint.
- Illustration of Mesh Blueprint examples

- Ascent and Catalyst use Conduit for describing data and other parameters which can be communicated between a simulation and the visualization apps.

Conduit: super simple and intuitive interface to build nodes

```
import conduit
n = conduit.Node()
n[“key"] = "data"
print(n)
```

```
--------------------------------

key: "data"
```

```
n = conduit.Node()
n[“key"] = "data"
n["a/b/c"] = "d"
n["a"]["b"]["e"] = 64.0
print(n)
```

```
--------------------------------

key: "data"
a:
  b:
    c: "d"
    e: 64.0
```

Data ownership in Conduit

The *Node* class provides two ways to hold data, the data is either **owned** or **externally described**:

documentation

```
vals = numpy.zeros((5,),dtype=numpy.float64)
n = conduit.Node()
n["v_owned"].set(vals)
n["v_external"].set_external(vals)
```

Conduit: A uniform mesh example

```
Node mesh;
// create the coordinate set
mesh["coordsets/coords/type"] = "uniform";
mesh["coordsets/coords/dims/i"] = 3;
mesh["coordsets/coords/dims/j"] = 3;
// 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/spacing/dx"] = 10.0;
mesh["coordsets/coords/spacing/dy"] = 10.0;
```

Conduit: A uniform mesh example

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

// add a simple element-associated field
mesh["fields/ele_example/association"] =  "element";
// reference the topology this field is defined on by name
mesh["fields/ele_example/topology"] =  "topo";
// set the field values, for this case we have 4 elements
mesh["fields/ele_example/values"].set(DataType::float64(4));
```

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Ascent: an *in situ* visualization and analysis library based on **Conduit**

Ascent

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

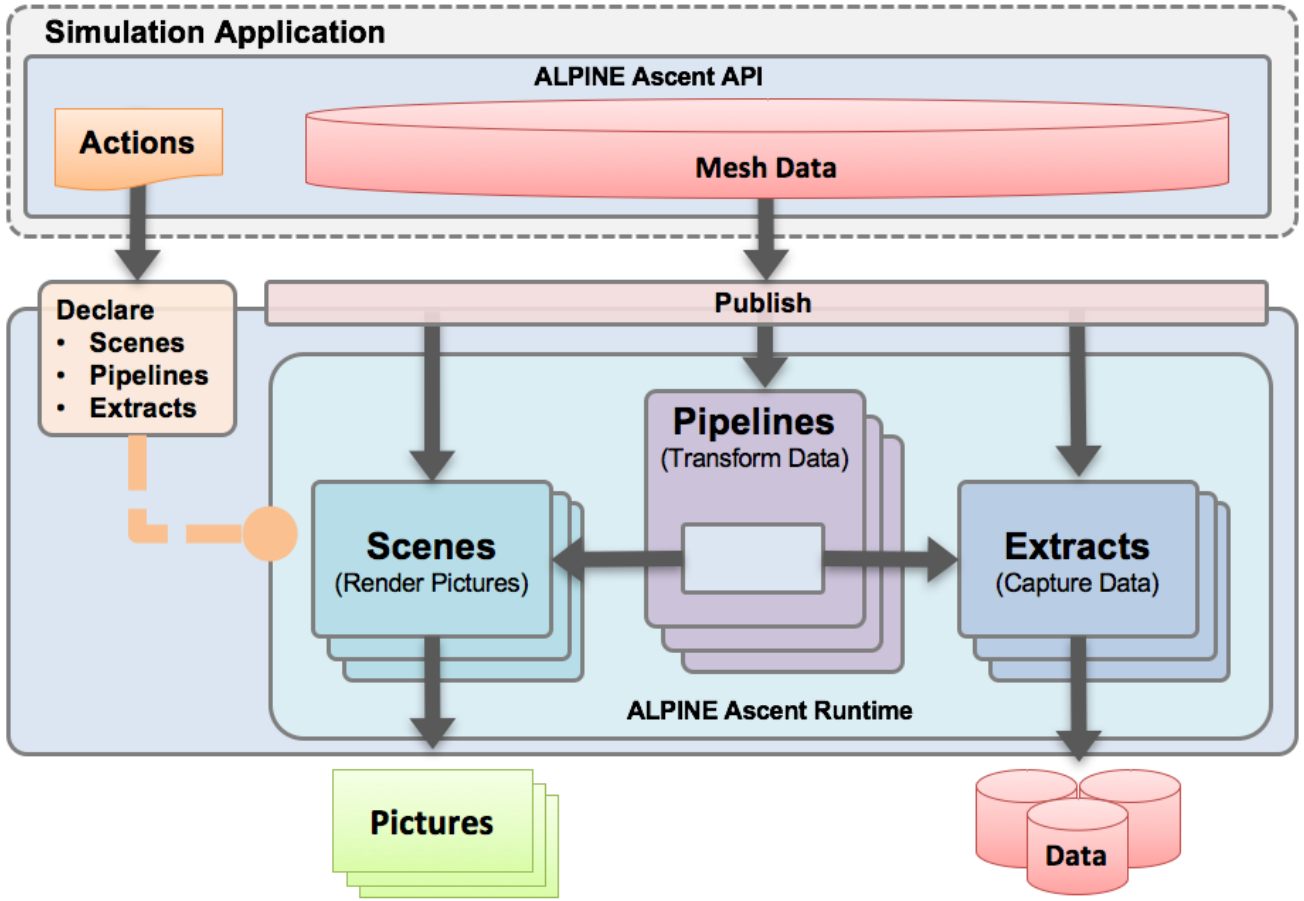
- Supports: Making Pictures, Transforming Data, and Capturing Data for use outside of Ascent
- Young effort, yet already includes most common visualization operations
- Provides a simple infrastructure to integrate custom analysis
- Provides C++, C, Python, and Fortran APIs
- Ref

"*The ALPINE in situ infrastructure: Ascending from the ashes of strawman*", M. Larsen et al., Proc. 3rd Workshop In Situ Infrastructures Enabling Extreme Scale Anal. Vis. Denver, CO, USA, Nov. 12–17, 2017

Ascent

based on several components:

- 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)



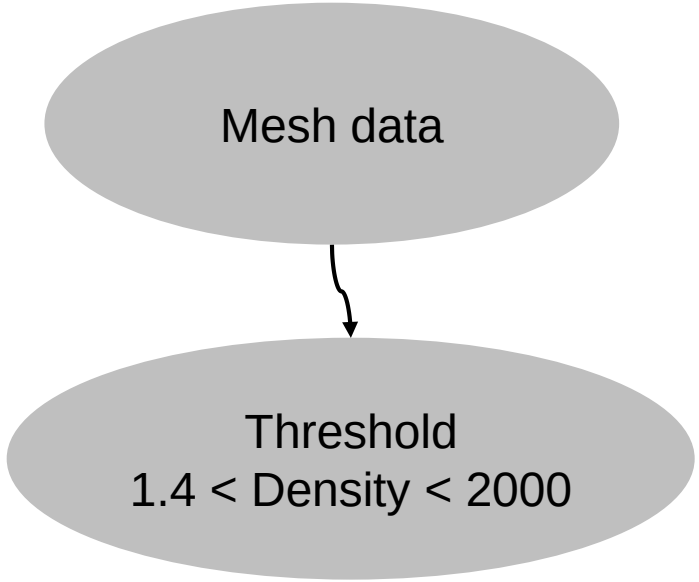
A scene description (JSON or YAML)

```
"action": "add_scenes",
        "scenes": {
            "s1": {
                "plots": {
                    "p1": { "type": "pseudocolor",
                            "field": "Density“ } },
                "renders": {
                    "r1": {
                     "image_prefix": “DensityImage.%05d",
                     "camera": {
                      "look_at": [0, 0, 0],
                      "position": [-2.17, 1.79, 1.80],
                      "up": [0.44, 0.84, -0.30]
                     }
```

```
-
  action: "add_scenes"
  scenes:
    s1:
      plots:
        p1:
          type: "pseudocolor"
          field: "Density"
    renders:
        r1:
          image_prefix: “DensityImage.%05d"
          camera:
            azimuth: 30
            elevation: 11
```

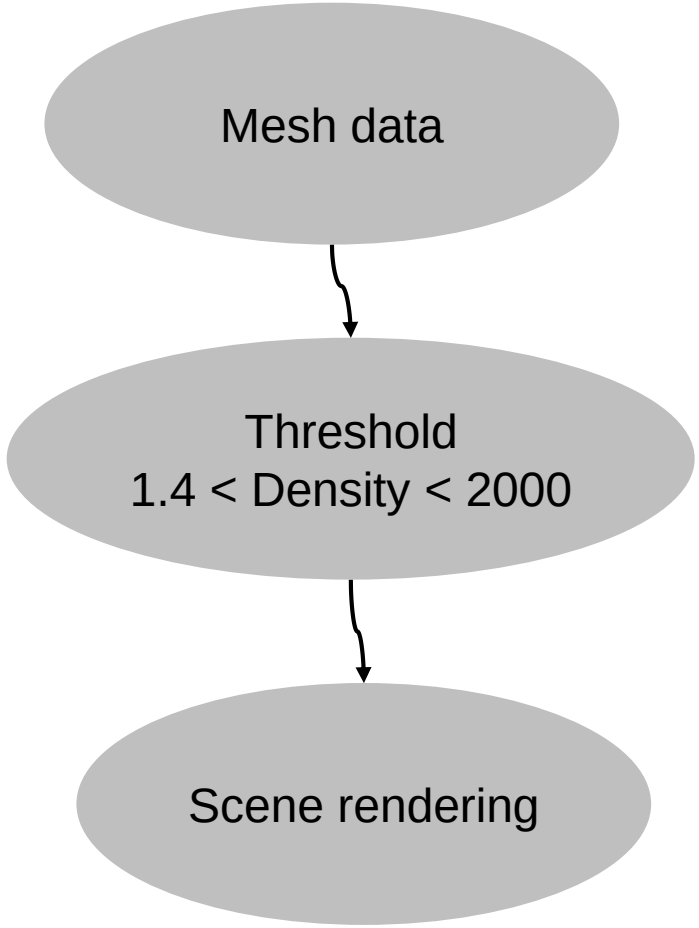
Let’s add a pipeline description

```
"action": "add_pipelines",
    "pipelines": {
        "pl1": {
            "f1": {
                "type": "threshold",
                "params": {
                    "field": "Density",
                    "min_value": 1.4,
                    "max_value": 2000
                }
            }
        }
    }
```



The scene description is refined with the new pipeline

```
"action": "add_scenes",
    "scenes": {
      "s1": {
        "plots": {
          "p1": {
            "type": "pseudocolor",
            "pipeline": "pl1",
            "field": "Density"
          }
        },
```



Example: Instrument an SPH simulation package with Ascent

- The smooth particle hydrodynamics (SPH) technique is a purely Lagrangian method. SPH discretizes a fluid in a series of interpolation points whose distribution follows the mass density of the fluid.

- PASC, the Swiss Platform for Advanced Scientific Computing initiative, supports the SPH-EXA project developing an SPH library.
- SPH-EXA is a C++20 headers-only code with no external software dependencies. The parallelism is currently expressed via the following models: MPI, OpenMP, CUDA and HIP.

Instrument the SPH-EXA simulation package with Ascent

- Define a Conduit mesh definition
- Define a Conduit scene definition

  About 150 lines of code. Total!

Using Conduit, a particle set is trivially described [the coordinates]

```
particle_set = """
  coordsets:
  coords:
  type: "explicit"
  values:
    x: [0.0, 10.0, 20.0, 30.0]
    y: [0.0, 10.0, 20.0, 30.0]
    z: [0.0, 10.0, 20.0, 30.0]
"""
```

```
conduit::Node mesh;
mesh["state/cycle"].set_external(&d.iteration);
mesh["state/time"].set_external(&d.ttot);
mesh["coordsets/coords/type"] = "explicit";

mesh["coordsets/coords/values/x"].set_external(&d.x);

mesh["coordsets/coords/values/y"].set_external(&d.y);

mesh["coordsets/coords/values/z"].set_external(&d.z);
// The heavy-data is available via shallow-copy links
```

Using Conduit, a particle set is trivially described [the topology]

```
particle_set = """
  topologies:
    mesh:
      type: "unstructured"
      elements:
      shape: "point"
      connectivity: [0, 1, 2, 3]
      coordset: "coords"
"""
```

```
mesh["topologies/mesh/type"].set("unstructured");
mesh["topologies/mesh/elements/shape"].set("point");
mesh["topologies/mesh/coordset"].set("coords");

std::vector<int> conn(N); // N is # of particles
std::iota(conn.begin(), conn.end(), 0);

mesh["topologies/mesh/elements/connectivity"].set_external(conn);
```

Using Conduit, a particle set is trivially described [the solution fields]

```
particle_set = """
fields:
  rho:
    association: "vertex"
    values: [-1, -2, -3, -4]
    topology: "mesh"
    volume_dependent: "false"
    units: "g/cc"
"""
```

```
auto fields = mesh["fields"];
// Density scalar field
fields["rho/association"].set("vertex");
fields["rho/topology"].set("mesh");
fields["rho/volume_dependent"].set("false");
// Conduit supports shallow copy
fields["rho/values"].set_external(&d.rho);
```

Catalyst: an API specification developed for simulations (and other scientific data producers) to analyze and visualize data in situ

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

enum catalyst_status **catalyst_initialize**(const conduit_node* params);

enum catalyst_status **catalyst_finalize**(const conduit_node* params);

enum catalyst_status **catalyst_execute**(const conduit_node* params);

Optional:

enum catalyst_status **catalyst_results**(conduit_node* params);

enum catalyst_status **catalyst_about**(conduit_node* params);

https://catalyst-in-situ.readthedocs.io

ParaView Catalyst

- 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. Simulations that can use the Mesh Blueprint to describe their data can directly use ParaView's Catalyst implementation for in situ analysis and visualization.

- ParaView-Catalyst
- ParaView-Catalyst Blueprint

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Conduit + ParaView Catalyst

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]
- ParaView provides hybrid parallelism out-of-the-box
  - MPI-enabled
  - SMP multi-threading
  - CUDA-enabled filters
- The Python scripts are completely interchangeable between the batch-mode ParaView execution (reading data from disk), and the in-situ execution

Catalyst scripts (batch-mode vs. in-situ mode)

```
grid = OpenDataFile(registrationName='grid',
 filename=['/LULESH/datasets/data_000009.vtpd'])


renderView1 = GetRenderView()
rep = Show()
ColorBy(rep, ['POINTS', 'velocity'])
Render()


 # execute in batch-mode with data read from disk
 from paraview.simple import
      SaveExtractsUsingCatalystOptions
 SaveExtractsUsingCatalystOptions(options)
```

```
grid = TrivialProducer(registrationName='grid‘)


 renderView1 = GetRenderView()
 rep = Show()
 ColorBy(rep, ['POINTS', 'velocity'])


 v = CreateExtractor('VTPD', grid)
 v.Trigger = 'TimeStep'
 v.Trigger.Frequency = 30
 v.Writer.FileName =
‘data_{timestep:06d}.vtpd‘
```

ParaView-Catalyst Blueprint

- Defines the options accepted by **catalyst_initialize()**; these include things like ParaView Python scripts to load, directories to save data

```
node["catalyst/scripts/script/filename"] = …
```

- Defines the protocol for **catalyst_execute()** and includes information about Catalyst channels i.e. ports on which data is made available

```
node["catalyst/state/cycle"] =
node["catalyst/state/time"] =

node["catalyst/channels/grid/type"] = "mesh"
node["catalyst/channels/grid/data"] =
```

Defines the protocol for **catalyst_finalize()**

Code instrumentation - before

The Catalyst glue code for the SPH-EXA solver is 144 lines of code

Enabling in-situ visualization can be optionally compiled

```
int main(int argc, char** argv)
{
  MPI_Init_and_Code_Init();

  for (d.iteration = 0; d.iteration <= maxStep; d.iteration++)
  {
  Solve_For_Each_Timestep();

  }

  return exitSuccess();
}
```

Code instrumentation - after

- The Catalyst glue code for the SPH-EXA solver is 144 lines of code
- The execution driver is instrumented with 4 lines of code
- Total: 148 lines of code

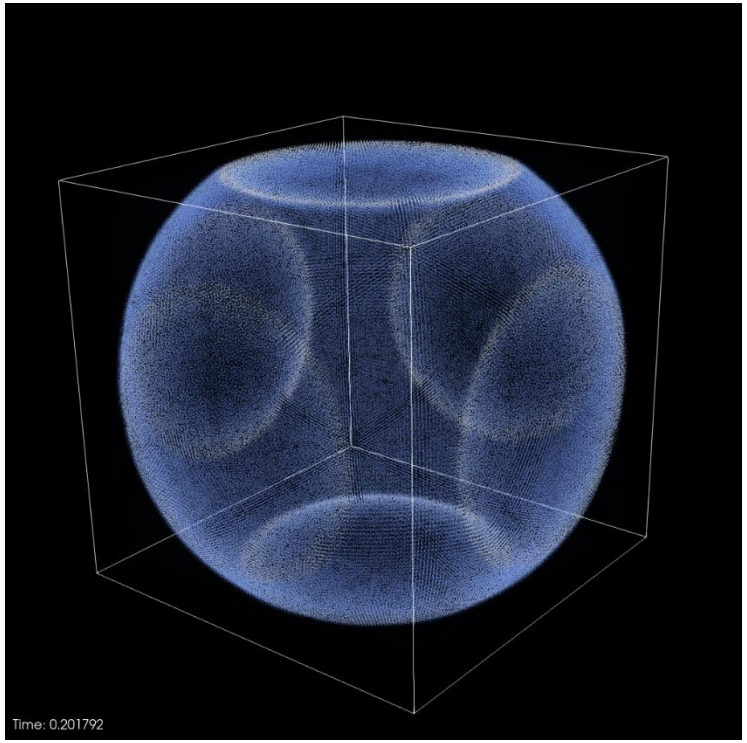
```
#include "CatalystAdaptor.h"
int main(int argc, char** argv)
{
  MPI_Init_and_Code_Init();
  CatalystAdaptor::Initialize(argc, argv);
  for (d.iteration = 0; d.iteration <= maxStep; d.iteration++)
  {
  Solve_For_Each_Timestep();
  CatalystAdaptor::Execute(d, domain.startIndex());
  }
  CatalystAdaptor::Finalize();
  return exitSuccess();
}
```

What about “Operations Control”?

Adaptive Control

- Choose different pathes of execution based on queries/triggers
- The catalyst_execute(ConduitNode) script can be customized

```
def catalyst_execute(node):
  threshold = 1.4
  if reader.PointData[“Density”].GetRange()[1] > threshold:
     print(“density at timestep”, node.timestep)
     # Extract particles where Density > threshold
  else:
     # use ALL particles
```



ParaView pipeline vs. Ascent 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
```

```
"action": "add_pipelines",
        "pipelines": {
          "pl1": {
            "f1": {
              "type": "threshold",
[…]
"action": "add_scenes",
        "scenes": {
          "s1": {
            "plots": {
              "p1": {
                "type": "pseudocolor",
                "pipeline": "pl1",
[…]
        "renders": {
              "r1": {
                "image_prefix": "ThresholdImage.%05d",
```

A look in the direction of ADIOS

ADIOS

- Adaptable Input Output System: summary
  - Extreme scale I/O: YES!
  - A file-only I/O library: NO!
- ADIOS Engines:
  - BP5
  - Sustainable Staging Transport (SST)

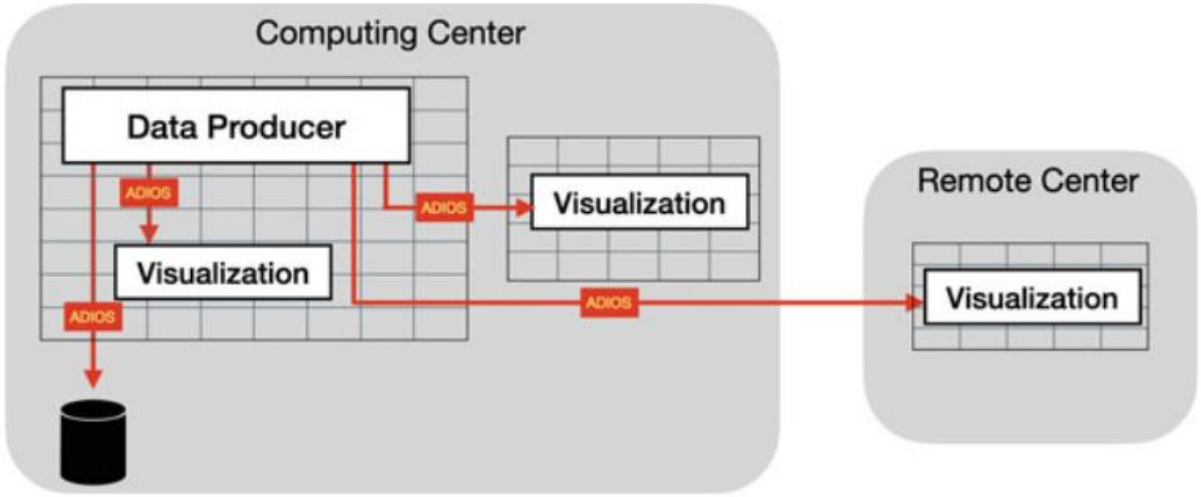


Figure taken from "*The Adaptable IO System (ADIOS)*", book chapter in In Situ Visualization for Computational Science

ADIOS I/O Abstraction

- “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 for output.

N.B. Missing are descriptions of mesh types…….. We’ll follow up on that…

N.B. There is nothing in the ADIOS interface that prescribes how to handle the data

ADIOS SST Engine

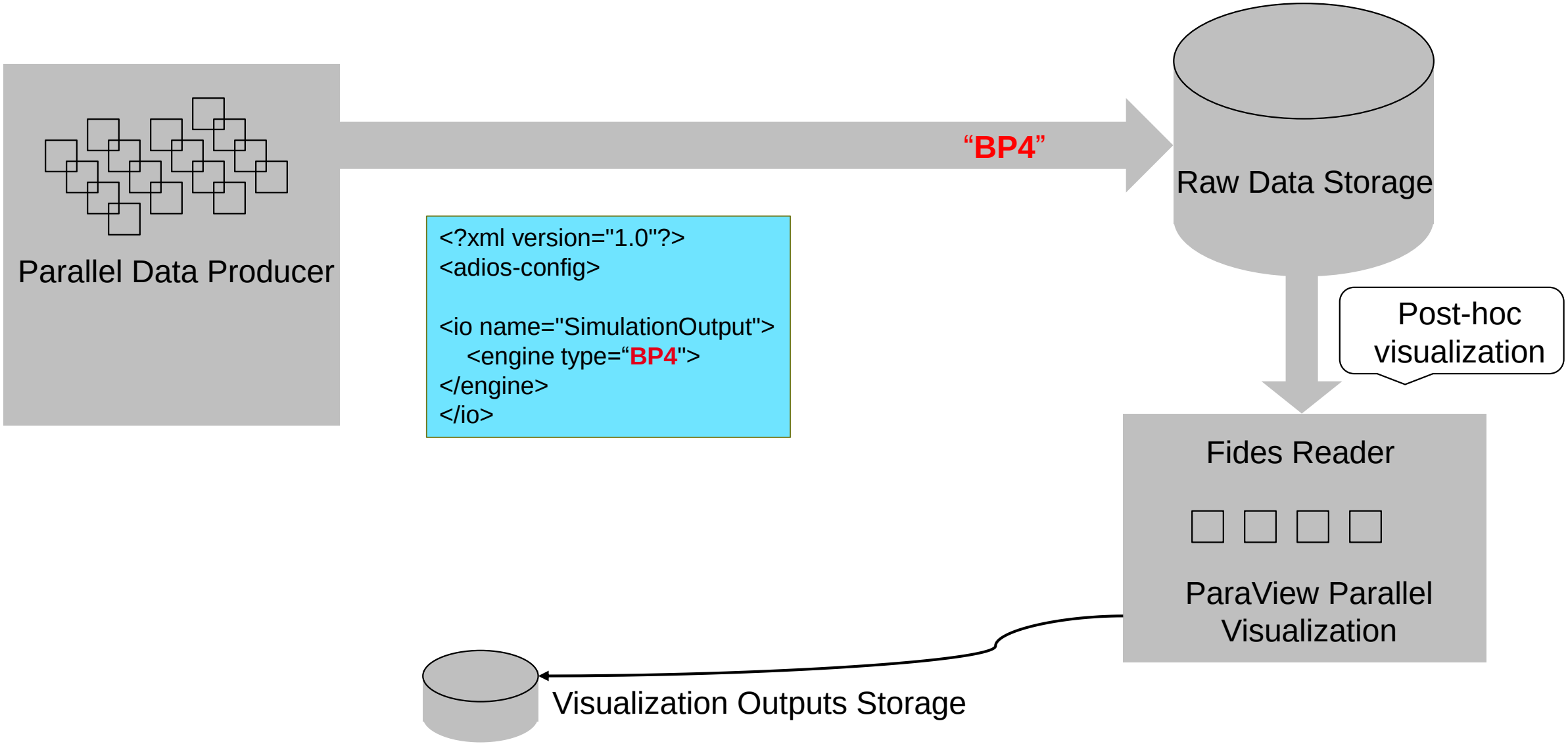
- SST allows direct connection of data producers and consumers via the ADIOS2 write/read API

- the SST buffering policy can be configured at run-time

- SST readers and writers **do not necessarily move in lockstep**, but depending upon the queue length parameters and queueing policies specified, differing reader and writer speeds may cause **one or the other side to wait for data to be produced or consumed**, or **data may be dropped if allowed** by the queueing policy

- SST supports full MxN data distribution

Post-hoc Visualization with ADIOS2 and Fides



Fides: an ADIOS Schema

- Schemas provide the ability to annotate the semantics of the array-based layout of data in ADIOS.
- They provide the meaning of each data array, and the relationship between groups of arrays:
  - Coordinates arrays
  - Connectivity arrays
- Fides is a library that uses a JSON data model to map ADIOS2 data arrays to VTK-m 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

- An ADIOS description of Variable and Attributes is **augmented** with attributes describing the supported data model

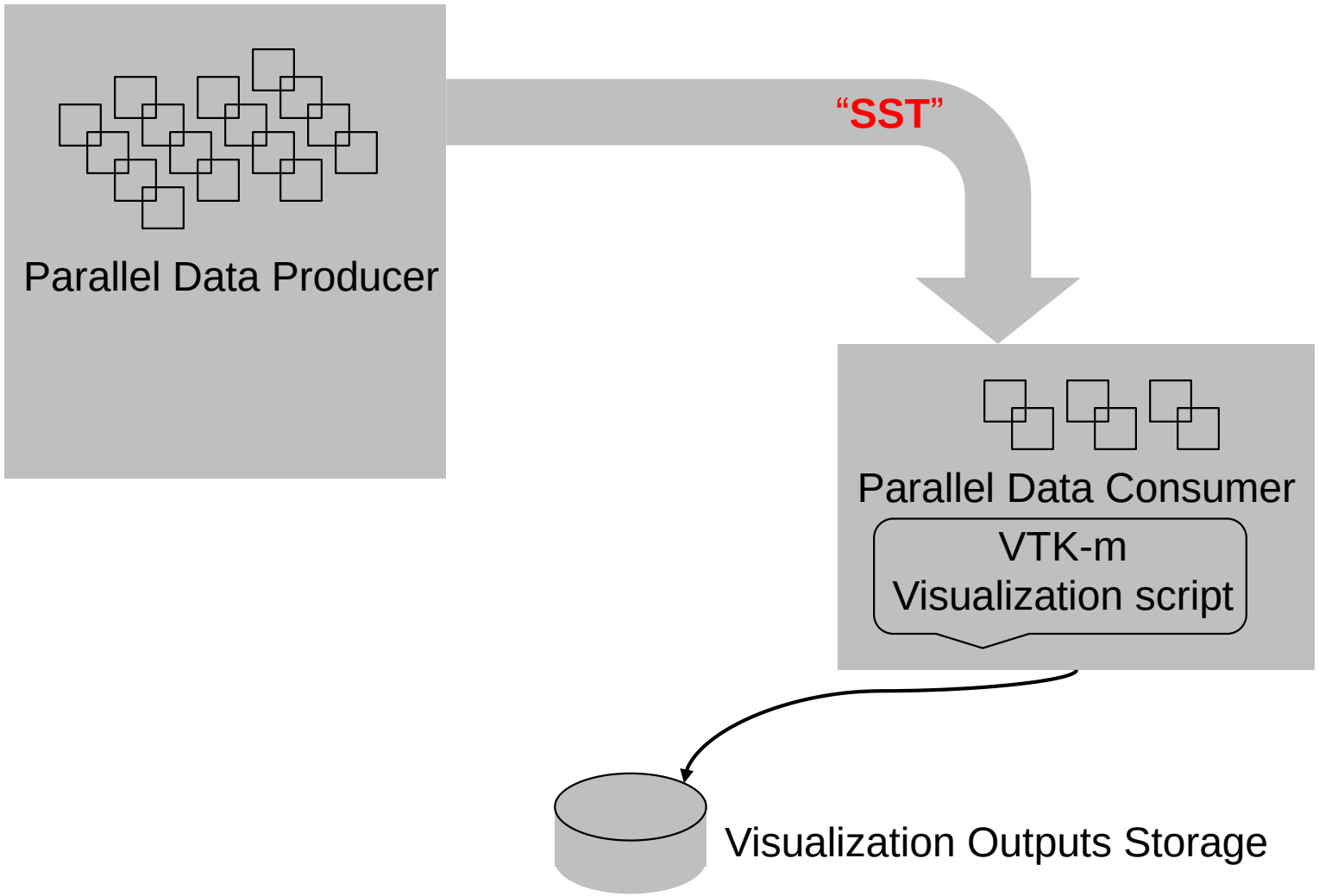
Uniform Data Model

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

In-transit Visualization with ADIOS2




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

<io name="SimulationOutput">
    <engine type="SST">
    <parameter key="RendezvousReaderCount“
                    value="1"/>
     <parameter key="QueueLimit“
                     value="5"/>
     <parameter key="QueueFullPolicy“
                    value="Block"/>
     </engine>
</io>
```

VTK-m facts

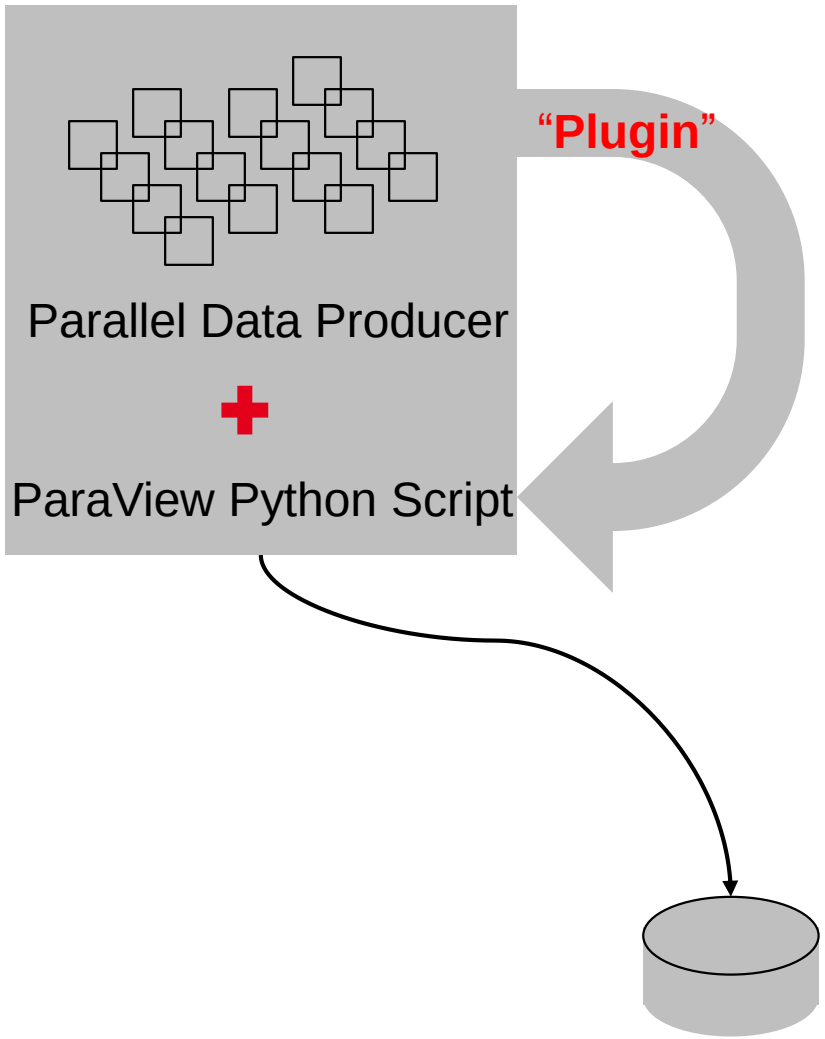
- Advances in processor technology include ever greater numbers of cores, hyperthreading, accelerators with integrated blocks of cores, and special vectorized instructions, all of which require more software parallelism to achieve peak performance
- VTK-m is a visualization toolkit for multi-/many-core architectures with support for finer threading typical in HPC today

- VTK-m has its own self-contained lightweight rendering package
- Rendering done by VTK-m's rendering classes is performed offscreen
- VTK-m data model differs from the traditional VTK data model!

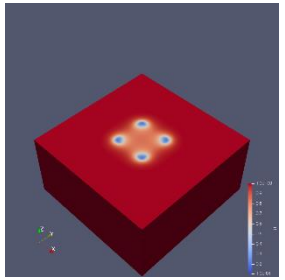
ADIOS Plugin

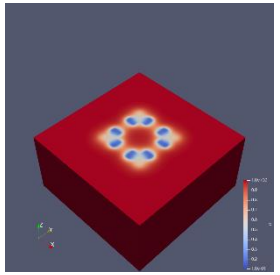
- ADIOS has the ability for users to load their own engines and operators through plugins
- As of v2.9, ADIOS has a **ParaView** “plugin”
  - Uses Catalyst and the Fides JSON description
  - Uses the “inline” engine
  - The Inline engine provides in-process communication between writers and readers, avoiding the copy of data buffers.
  - This engine is focused on the N → N case
  - Data are not copied to a file or to another buffer

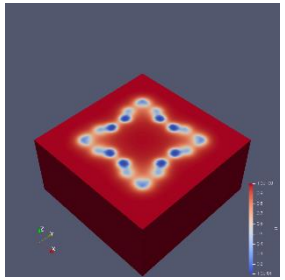
In-situ Visualization with ADIOS2

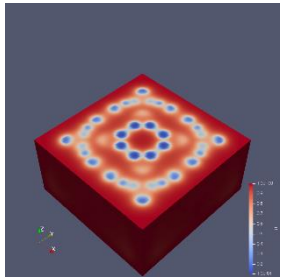



```
<?xml version="1.0"?>
<adios-config>

 <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>
```

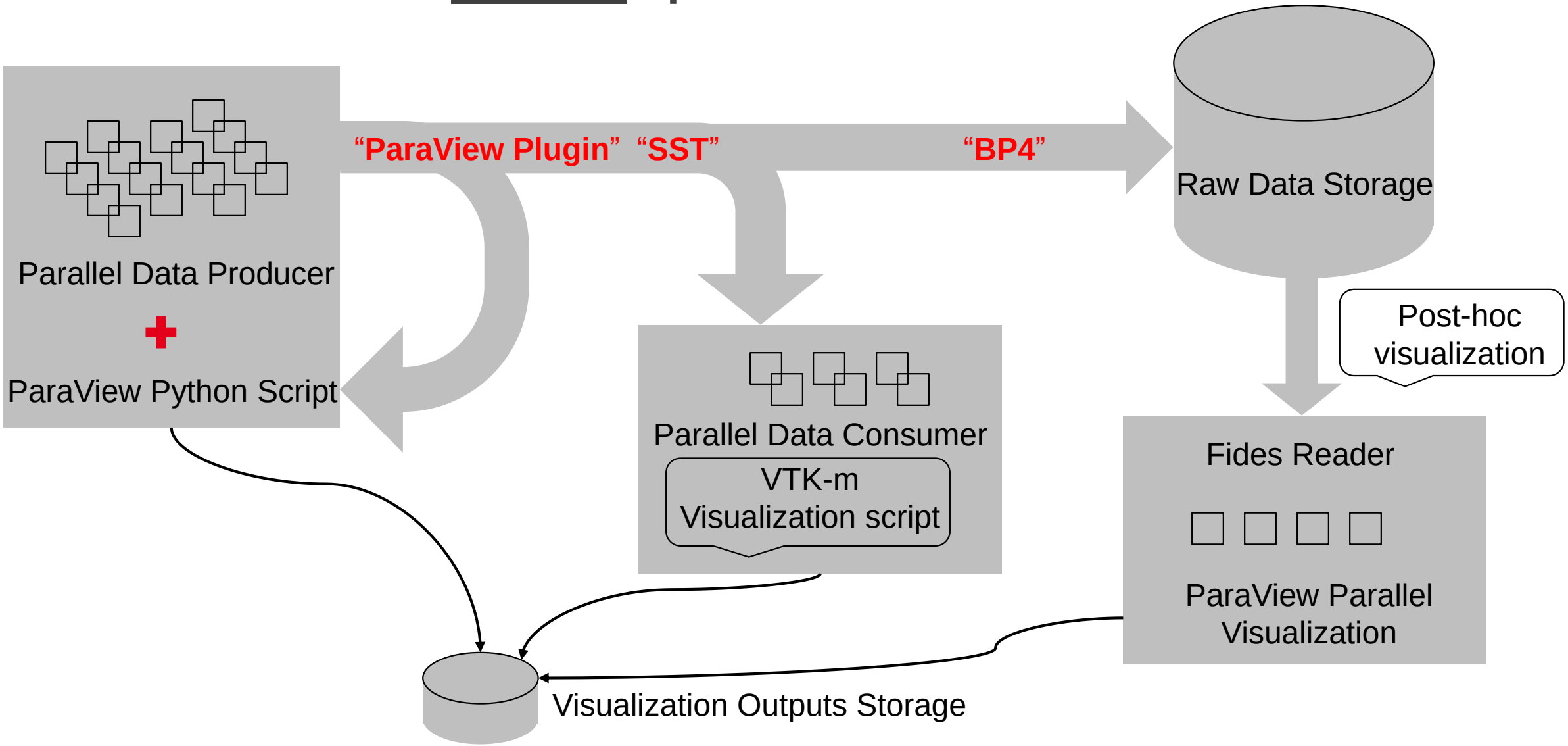








Run-time selection for multiple options for Visualization



User-defined visualization programs (pipelines)

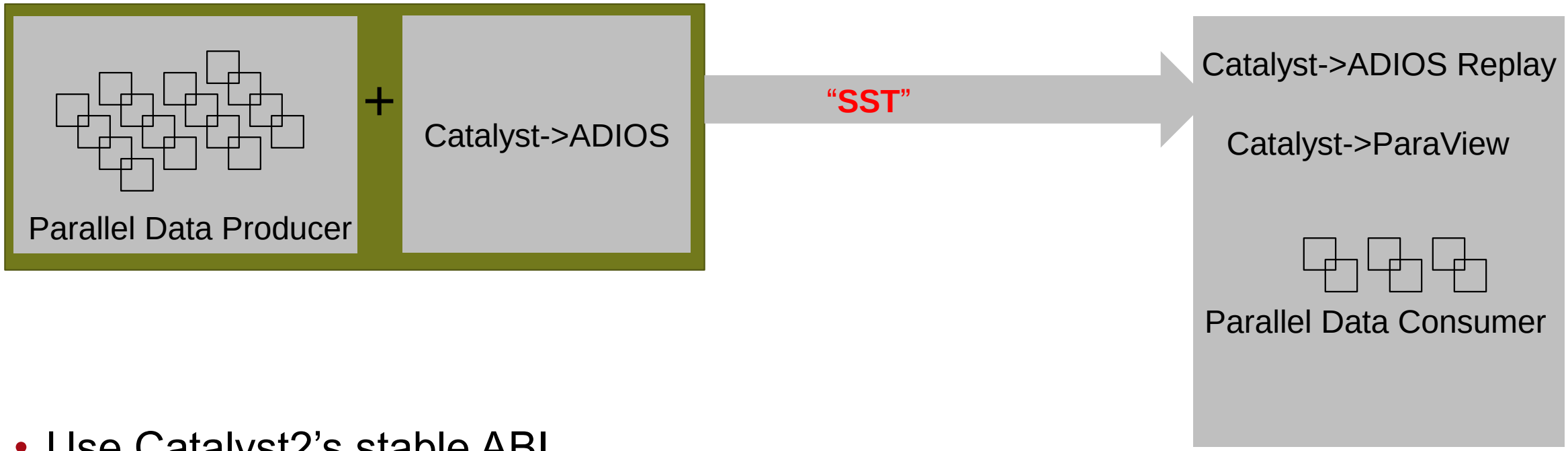
VTK-m

```
fides::io::DataSetReader reader("fides.json");
std::unordered_map<std::string, std::string> paths;
paths[source_name] = std::string("gs.bp");
fides::DataSourceParams parms;
parms["engine_type"] = "SST";
reader.SetDataSourceParameters(“source”,
std::move(parms));
vtkm::cont::PartitionedDataSet output =
reader.ReadDataSet(paths, sels);
vtkm::Vec3f origin(3.15, 3.15, 3.15), normal(0., 0., 1.);
vtkm::filter::contour::ClipWithImplicitFunction clip;
clip.SetImplicitFunction(vtkm::Plane(origin, normal));
clip.SetInvertClip(1);
clip.SetFieldsToPass("U");
vtkm::cont::DataSet outputData = clip.Execute(inputData);
```

ParaView Python

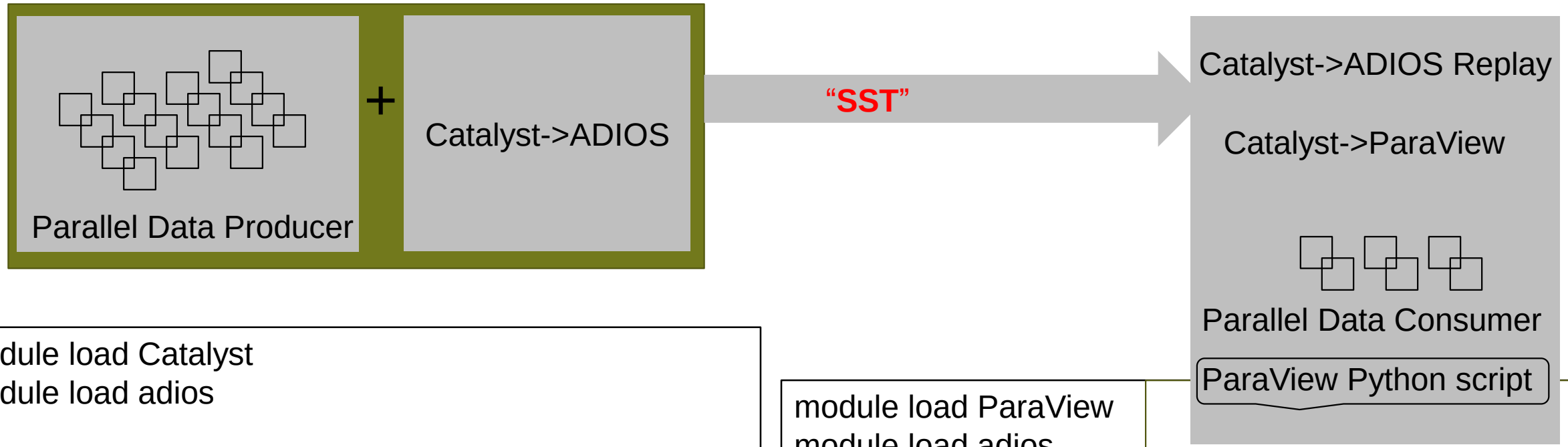
```
mesh = TrivialProducer(registrationName='fides')
clip = Clip(registrationName="clip1", Input=mesh)
clip.ClipType = 'Plane'
clip.ClipType.Origin = [3.15, 3.15, 3.15]
clip.ClipType.Normal = [0.0, 0.0, 1.0]
clipDisplay = Show(clip)
ColorBy(clipDisplay, ('POINTS', 'U'))


camera = GetActiveCamera()
camera.Roll(-45)
camera.Elevation(-45)
```

A variation on the Catalyst API



- Use Catalyst2's stable ABI
- Replace the Catalyst-ParaView implementation by a Catalyst-ADIOS encapsulator (work by the Kitware folks)
- https://gitlab.kitware.com/paraview/adioscatalyst

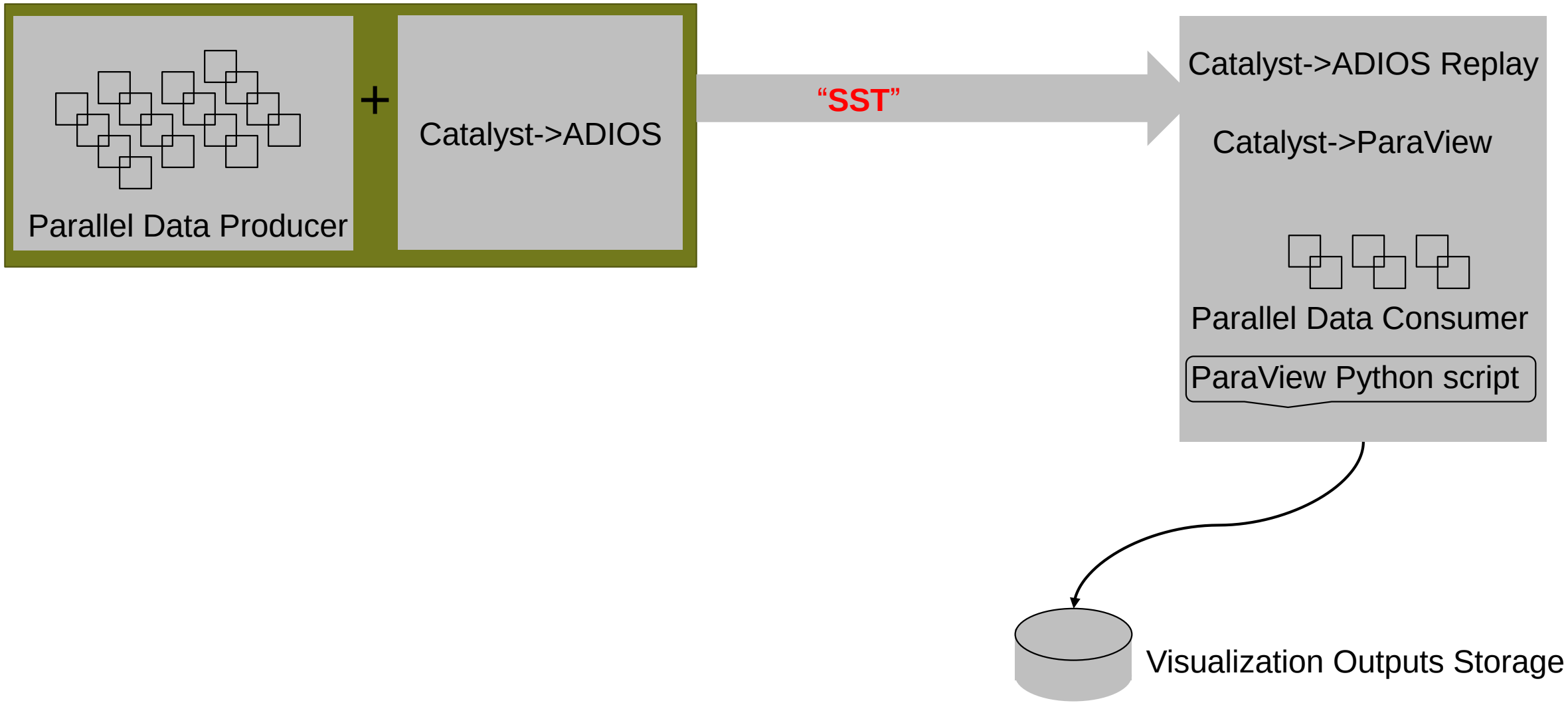
A variation on the Catalyst API




```
module load Catalyst
module load adios

export CATALYST_IMPLEMENTATION_NAME=adios
export
CATALYST_IMPLEMENTATION_PATHS=/users/jfavre/
Projects/InTransit/adioscatalyst/build/lib64/catalyst

srun Data_Producer adios2.xml paraview_script.py
```

```
module load ParaView
module load adios

export
CATALYST_IMPLEMENTATION_NAME=paraview
export
CATALYST_IMPLEMENTATION_PATHS=/paraview-
install/lib64/catalyst

srun AdiosReplay adios2.xml
```

A variation on the Catalyst API



Other references

- This is by far not an exhaustive panorama of in-situ/in-transit solutions.
- They are only the ones I have had time to try, and used to instrument some applications. Quite a bit ParaView, or VTK-m centric

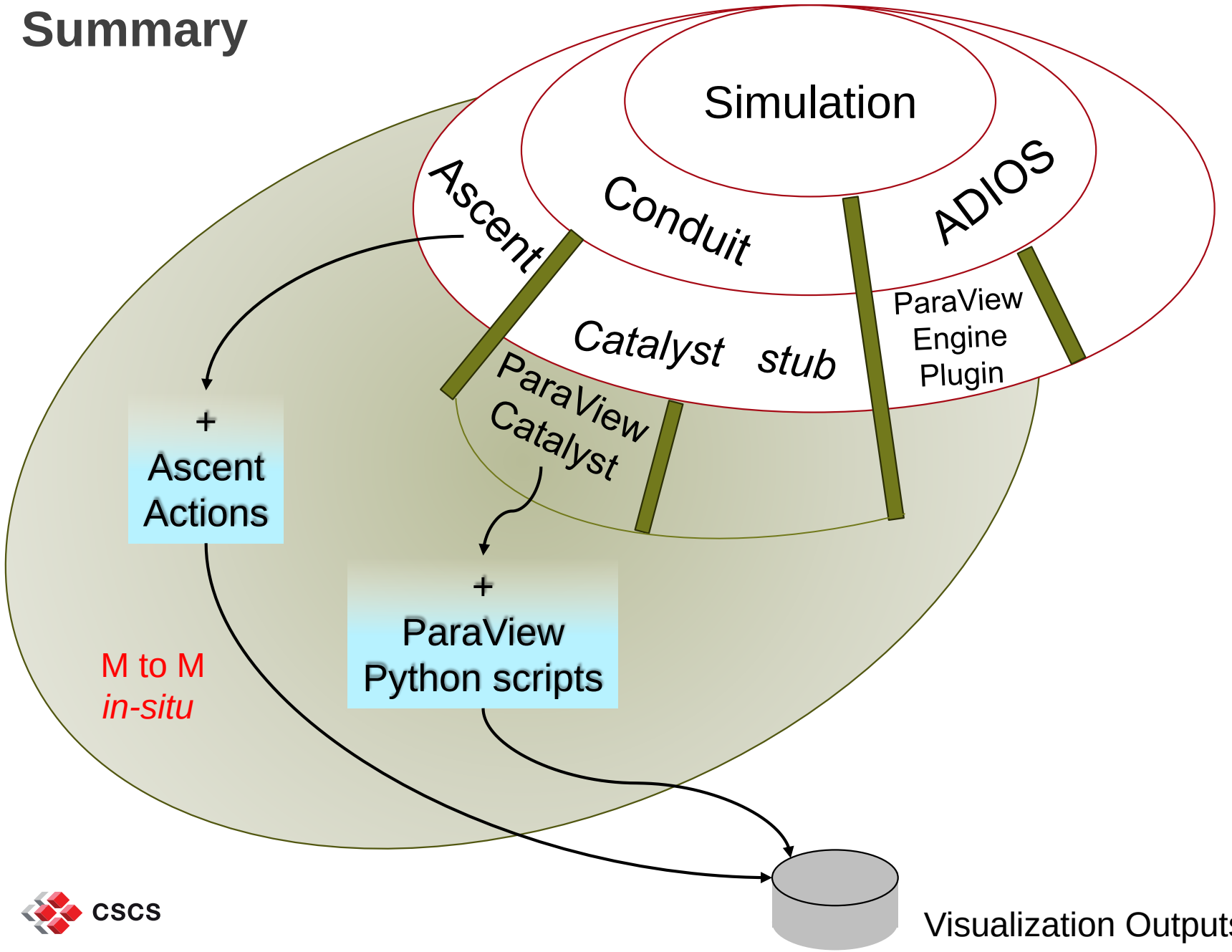
- Look at SENSEI (a generic data interface and a data model) to serve data to Libsim, Catalyst2, or ADIOS…

- look at the WOIV workshops at ISC, …

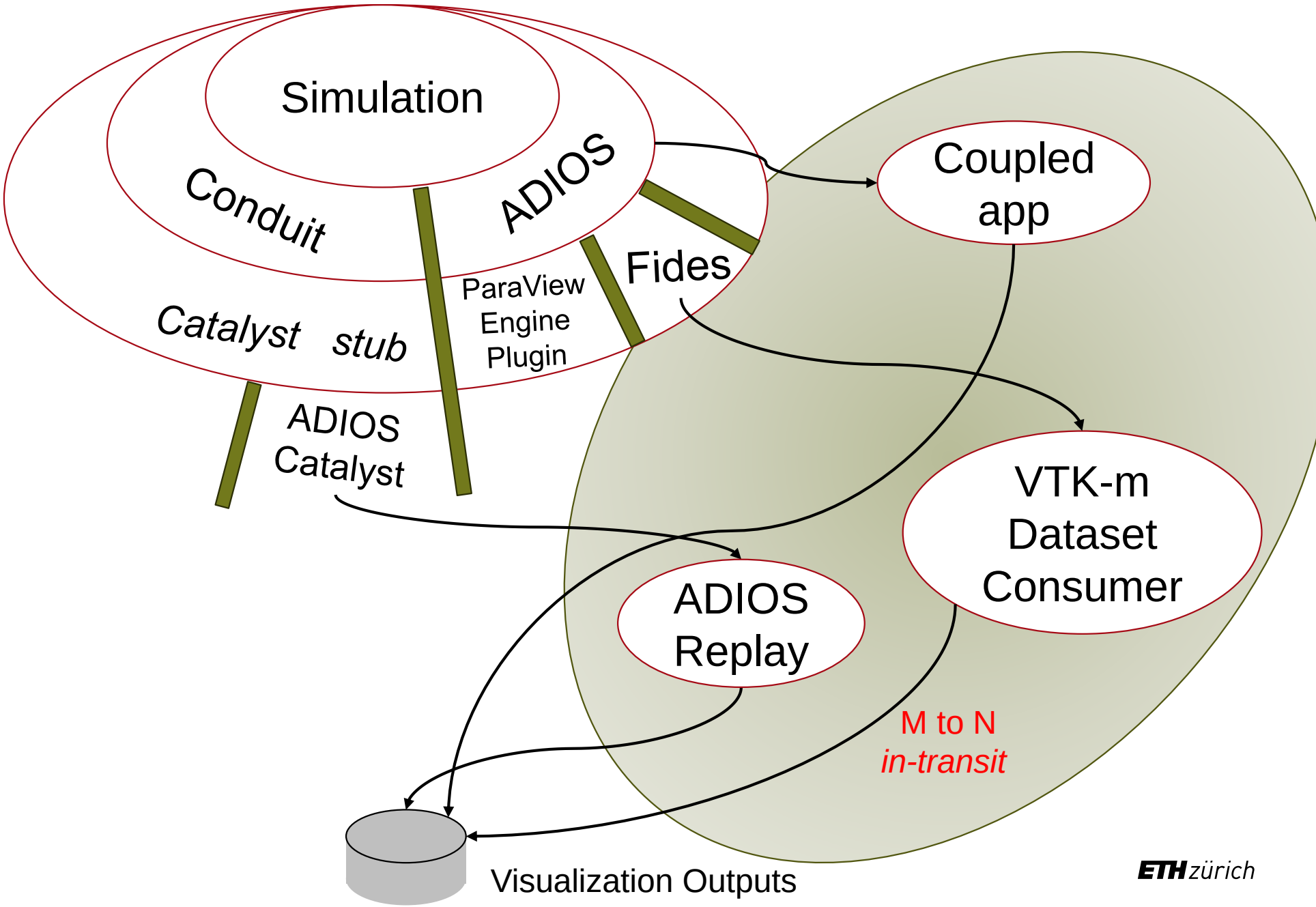
Summary

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

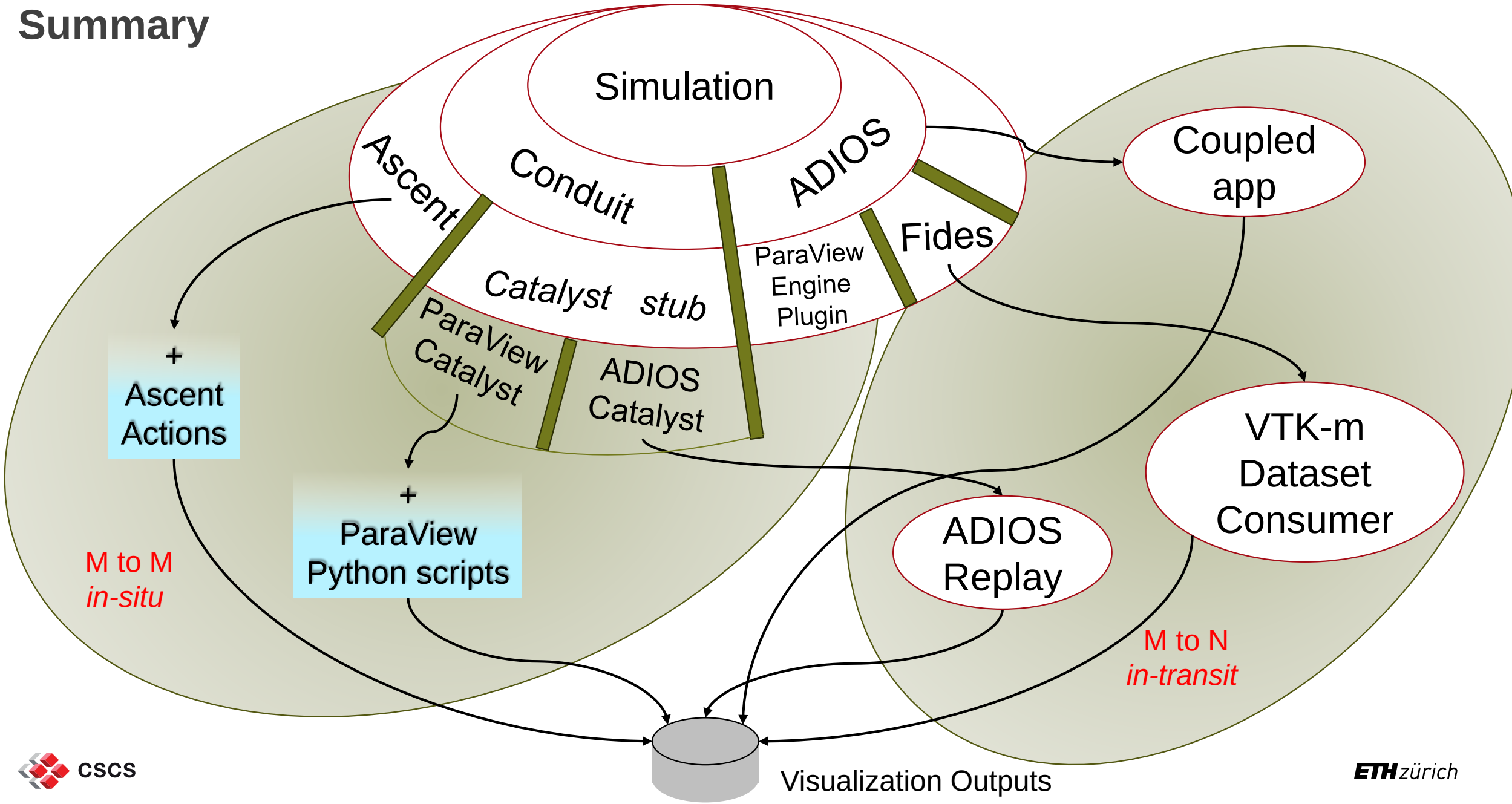
Summary


Simulation
Ascent
Conduit
ADIOS
ParaView
Engine
Plugin
Catalyst stub
ParaView
Catalyst
+
Ascent
Actions
+
ParaView
Python scripts
M to M
in-situ
Visualization Outputs

Summary


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

Summary


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

Availability

- https://ascent.readthedocs.io/en/latest/QuickStart.html#public-installs-of-ascent
- Ascent, Catalyst, ADIOS + Fides at CSCS on Piz Daint and ALPS pre-system dev platform

In practice

- https://ascent.readthedocs.io/en/latest/ExampleIntegrations.html
- Link to Gray-Scott Fides demo
- Link to Adios-Catalyst demo
- Link to my in-situ visualization tutorial examples

- Up-coming tutorial:

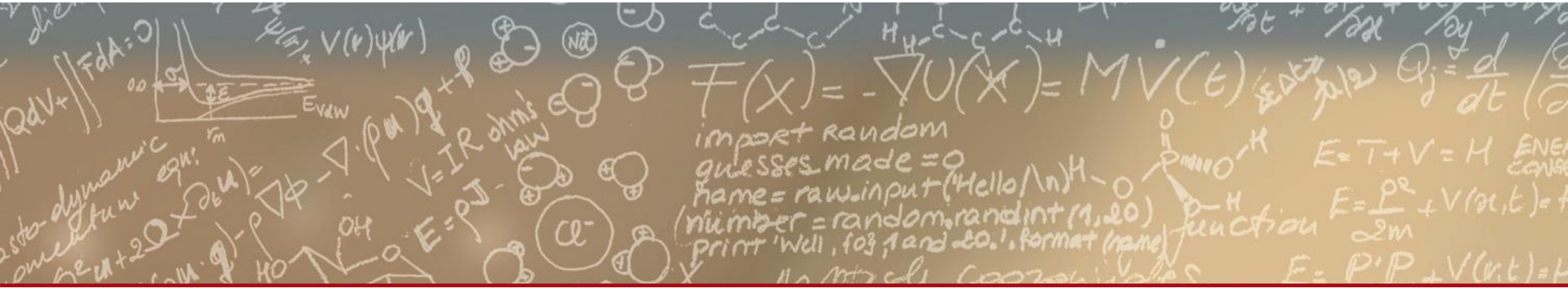
In-situ Analysis and Visualization with Ascent and ParaView Catalyst,

accepted for presentation at SC23

“Cyrus Harrison, Jean M. Favre, Corey Wetterer-Nelson, Nicole Marsaglia”

Acknowledgments

- Caitlin Ross, François Mazen, Kitware
- Cyrus Harrison, LLNL
- Hank Childs, U of Oregon



Thank you for your attention.