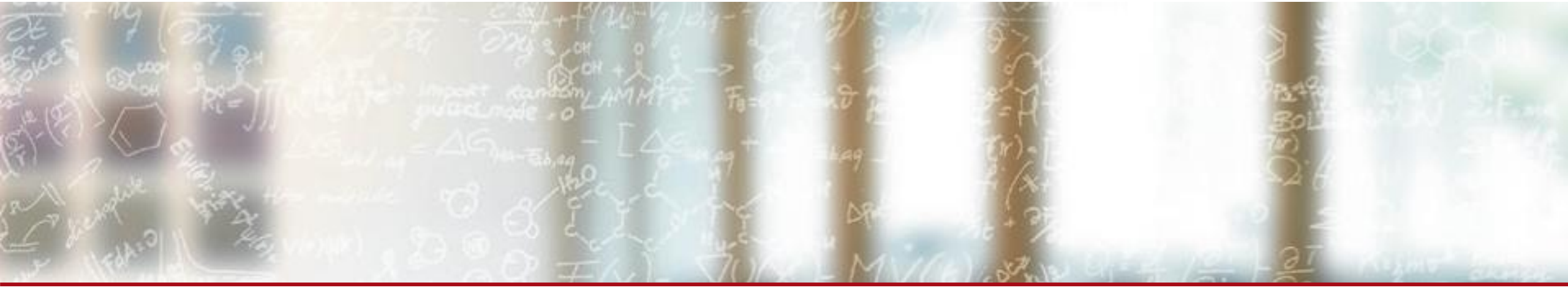
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Embracing new solutions for in-situ visualisation

P'Con 2021

Jean M. Favre,
Senior Visualization Software Engineer
CSCS

Thank to different teams

- P’Con organizers
- Ugo Varetto, Chief Technology Officer at Pawsey
- Yathu Sivarajah, Visualization leader at Pawsey

- Thanks to the many actors in the visualization and in-situ dev teams:
- Kitware, Berkeley Lab, ANL, LLNL, Los Alamos, Sandia, and many others

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

CSCS, Swiss National Supercomputing Center

“Piz Daint” at CSCS, the Swiss flagship for national HPC Service

- Cray XC40/XC50
- 5704 hybrid nodes (Intel Xeon E5-2690 v3/NVIDIA Tesla P100)
- 1813 multi-core nodes (Intel Xeon E5-2695 v4)

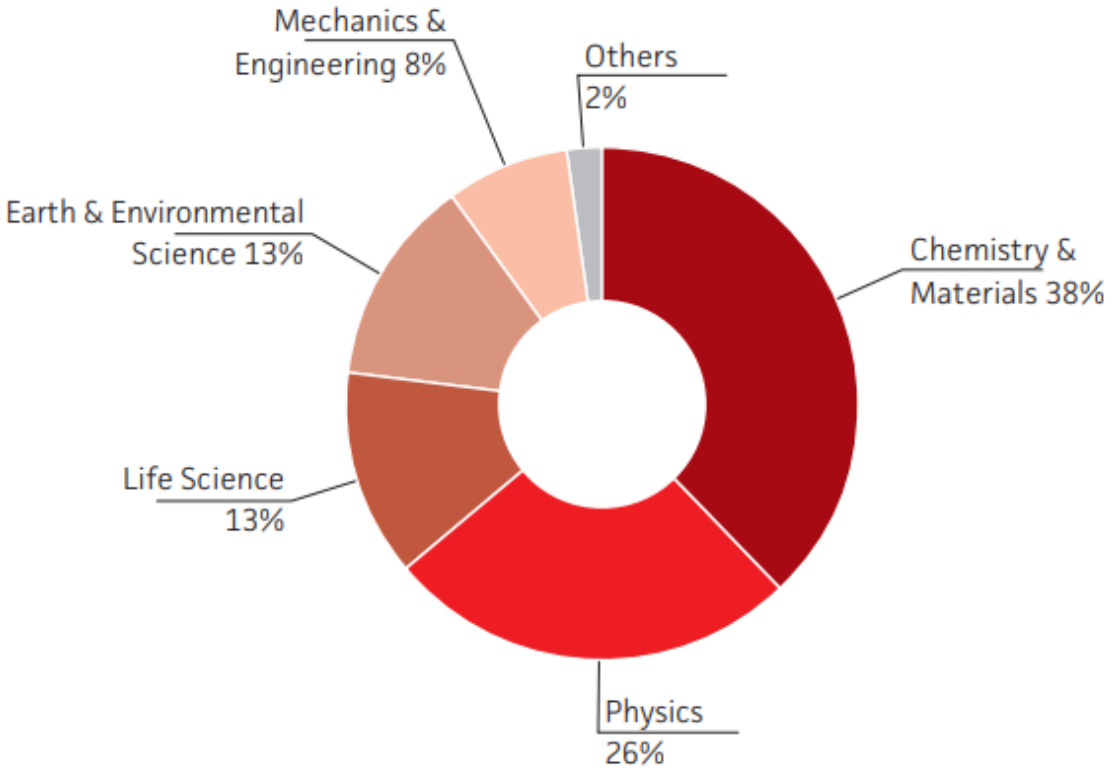
Coming online in 2023, the ‘Alps’ system infrastructure

will replace CSCS’s Piz Daint and serve as a general-purpose system.

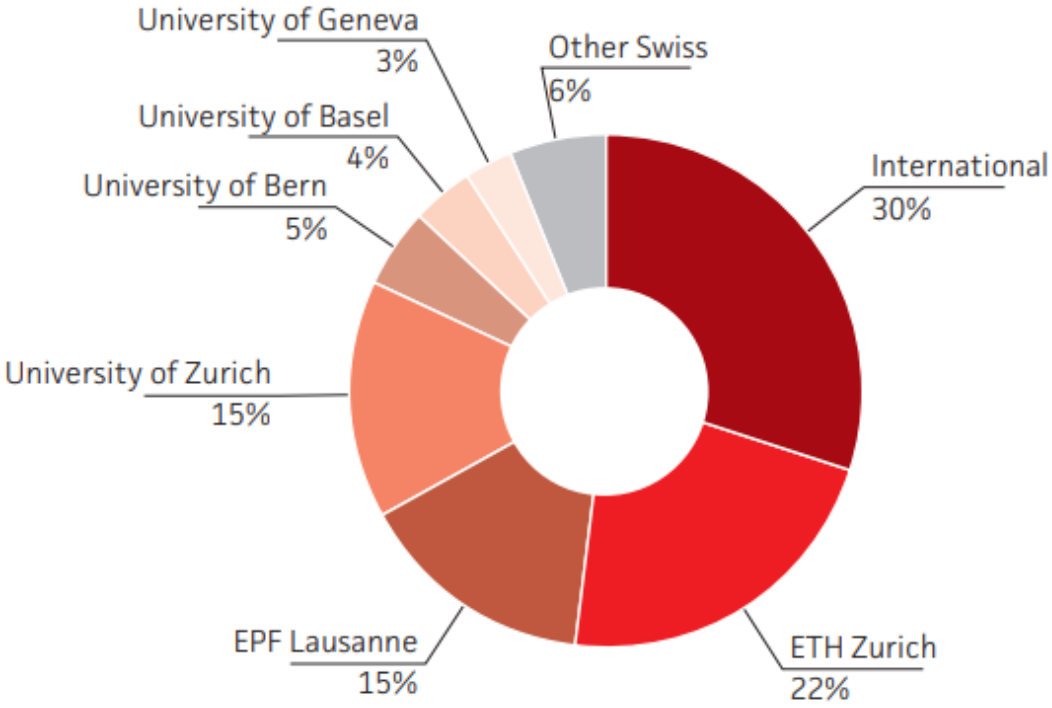
Link

CSCS serves a wide range of application and user communities

User Lab Usage by Research Field



User Lab Usage by Institution



Overview

- What is in-situ visualization, why do we need it? What solutions are available to implement it?
- Some examples from the previous decade (libSim, Catalyst)
- A few words about Sensei (from the SC21 tutorial)

- Replay some of the presentations from the most recent Ascent tutorial at SC21

- Present the ParaView-Catalyst and Ascent solutions, which share a common data definition layer based on Conduit
- Cinema

The past

- 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

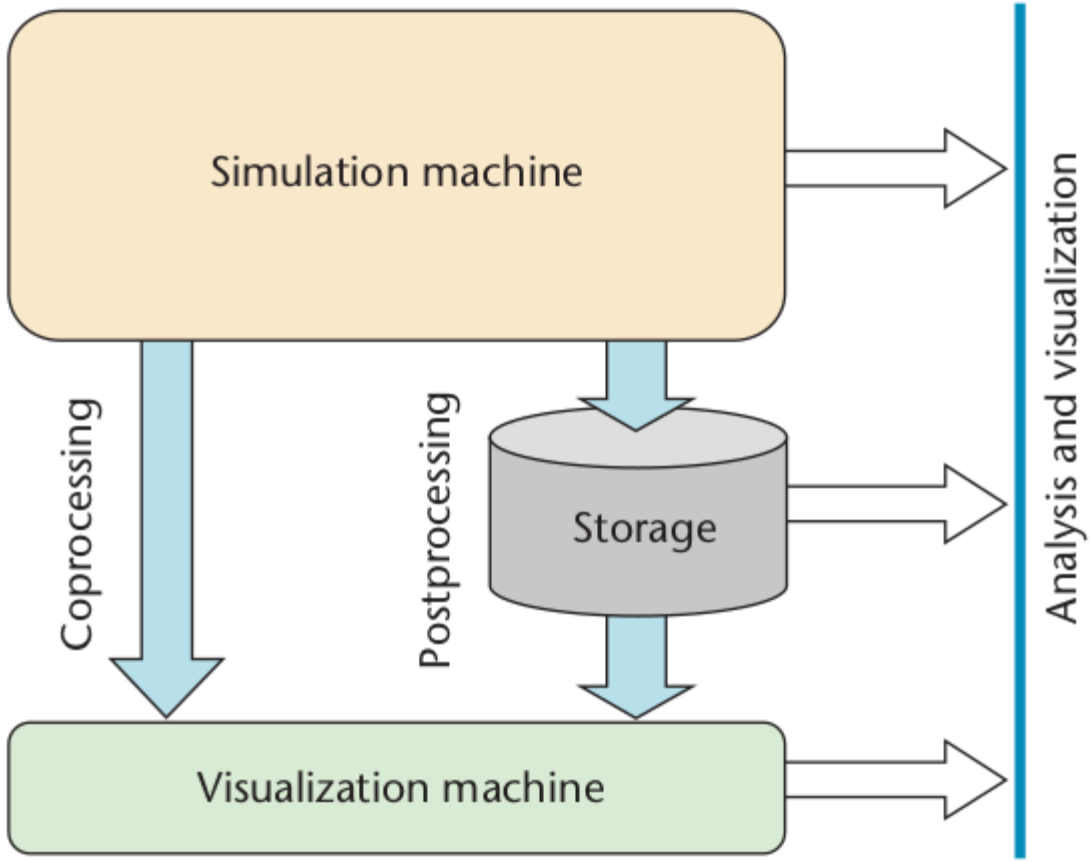


Figure taken from "*In Situ Visualization at Extreme Scale: Challenges and Opportunities*", Kwan-Liu Ma, IEEE CG&A, nov/dec 2009

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*

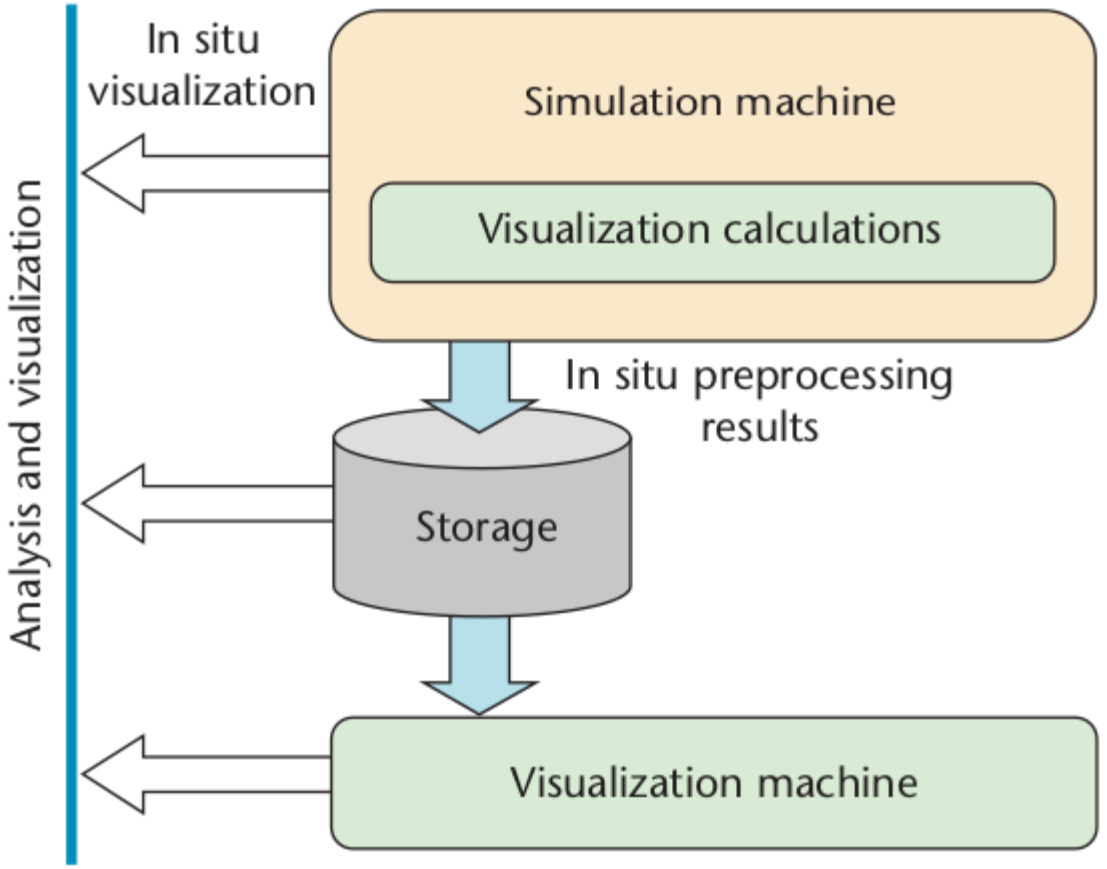


Figure taken from "*In Situ Visualization at Extreme Scale: Challenges and Opportunities*", Kwan-Liu Ma, IEEE CG&A, nov/dec 2009

In-situ visualization 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

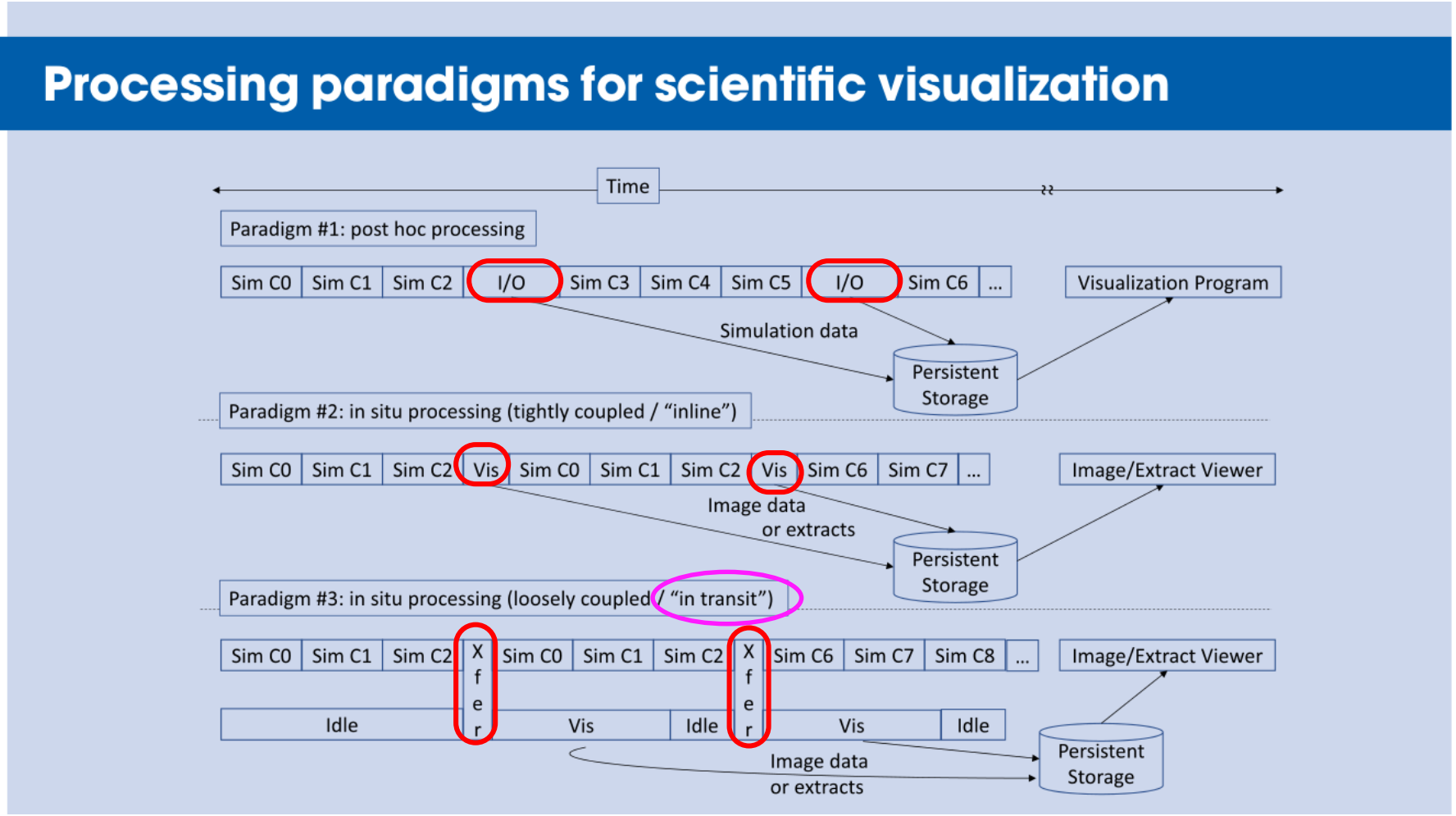


Figure taken from "*In Situ Visualization for Computational Science*", Hank Childs et al., IEEE CG&A, nov/dec 2019

Does “in-situ” mean “in place”?

- The data is already in the processor’s memory space, without touching the disks

- If simulation data is moved to a distinct set of resources (nodes dedicated to visualization), we are still analysing data “in place”, but is it “in-situ”?

- There has been quite a few variants on the terminology:
  - Co-processing, concurrent processing, run-time visualization

Many definitions and colloquial use for “in-situ”

- An exhaustive panorama of the different systems in use was created :
- "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 were best 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

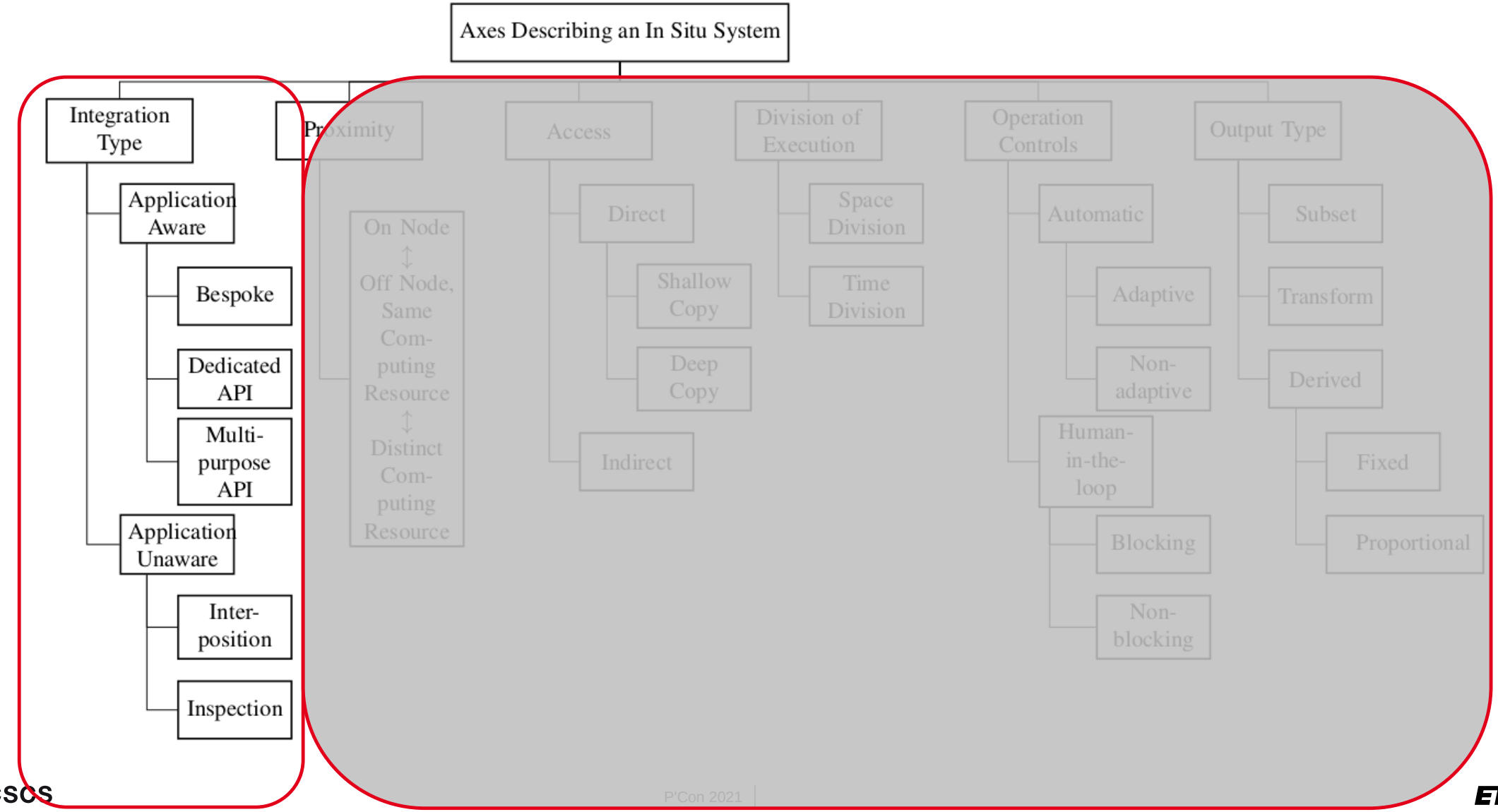
  How does the simulation give access to the data?
- division of execution:

  how compute resources are shared between simulation and in situ routines.
- operation controls:

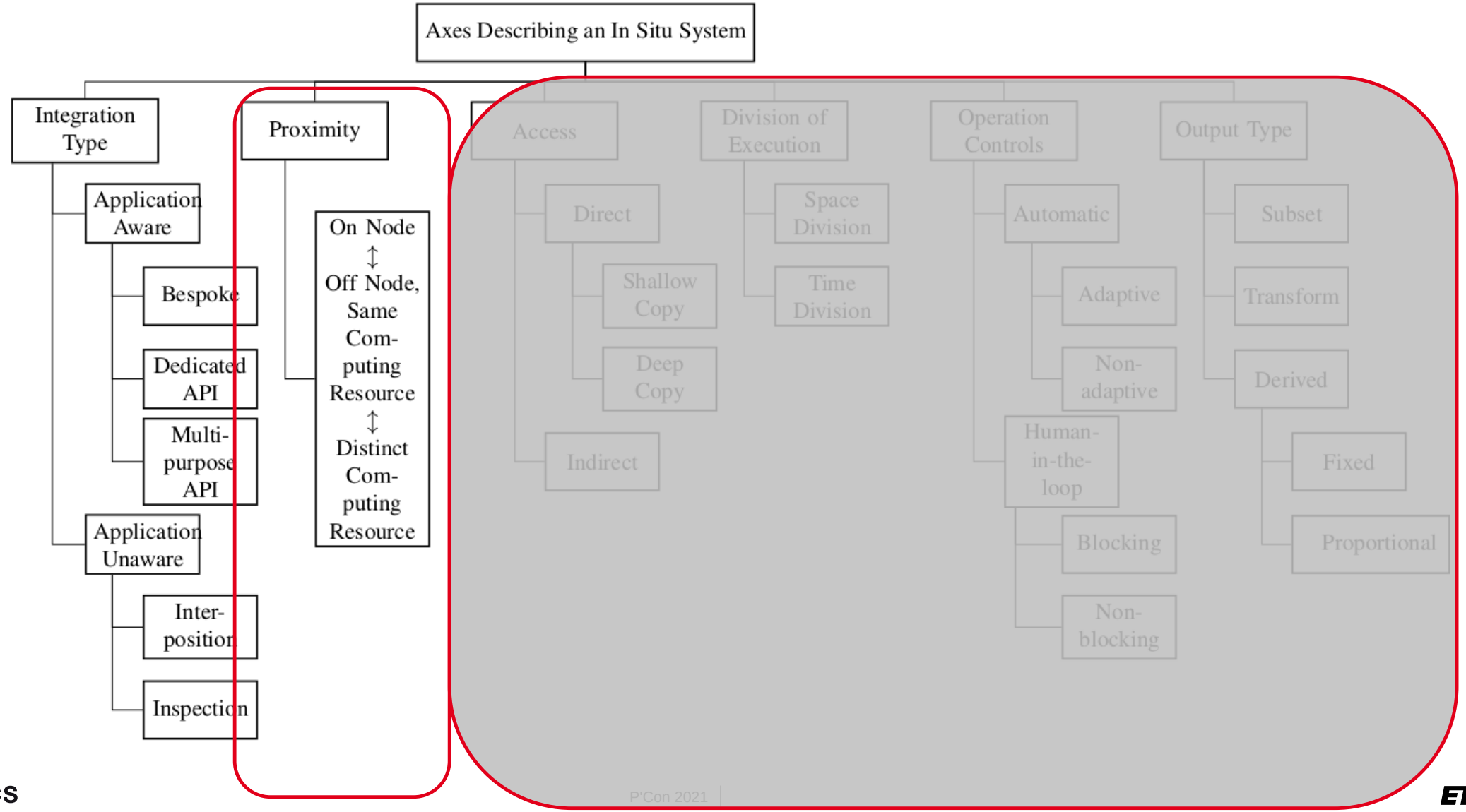
  the mechanism for selecting which operations are executed during run-time
- output type

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

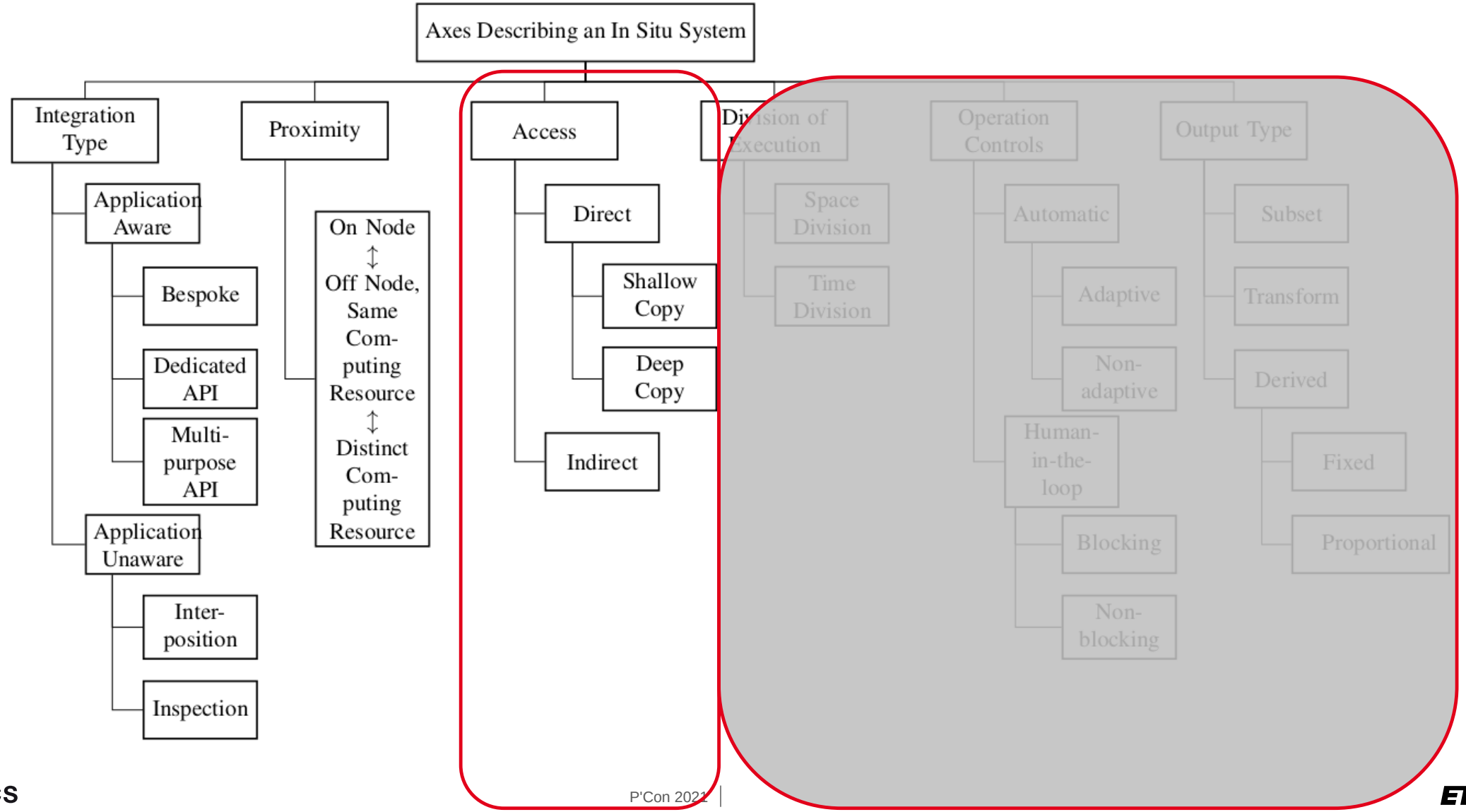
Integration Type



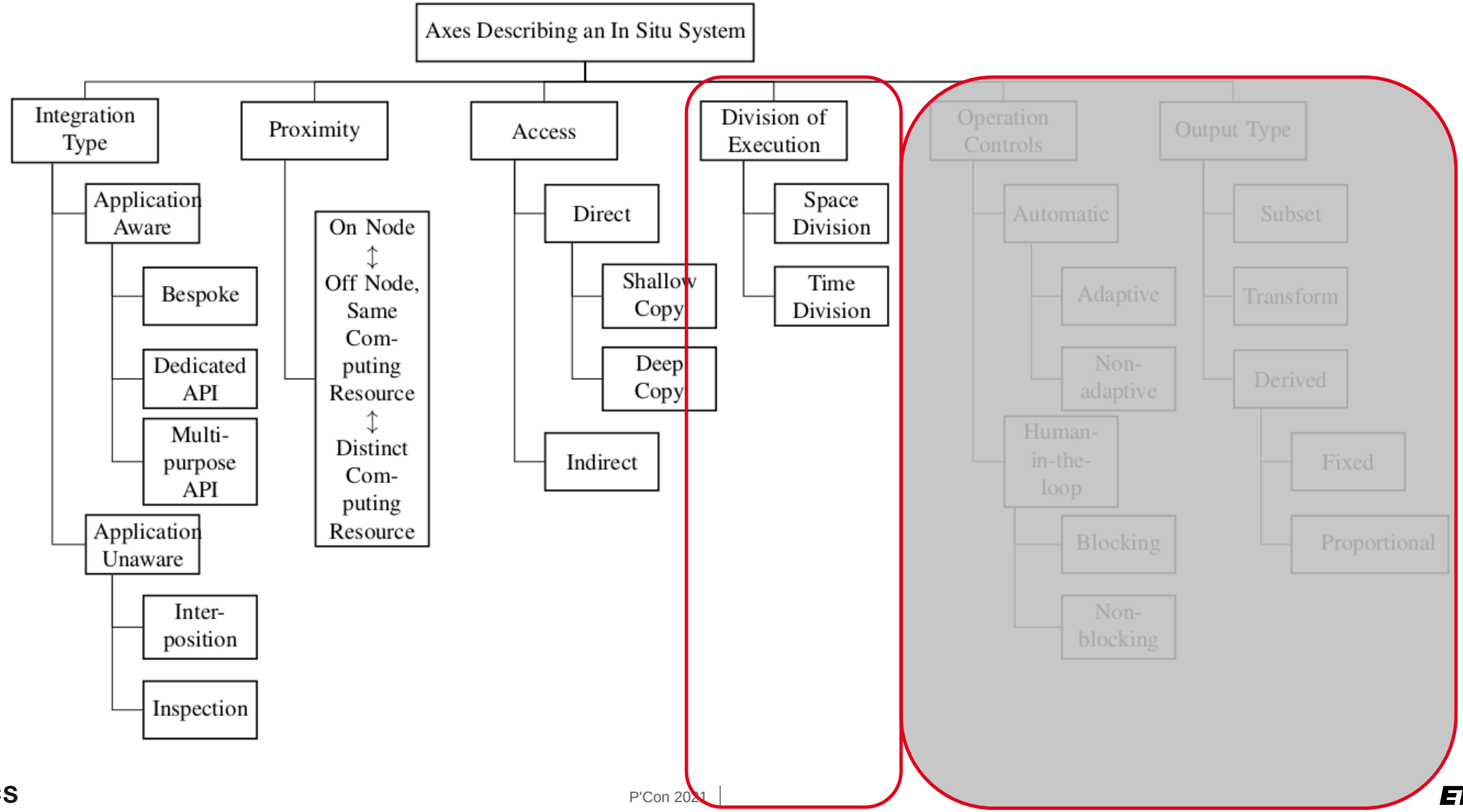
Proximity


Axes Describing an In Situ System
Integration Type
Application Aware
Bespoke
Dedicated API
Multi-purpose API
Application Unaware
Inter-position
Inspection
Proximity
On Node
↕
Off Node, Same Com-puting Resource
↕
Distinct Com-puting Resource
Access
Direct
Shallow Copy
Deep Copy
Indirect
Division of Execution
Space Division
Time Division
Operation Controls
Automatic
Adaptive
Non-adaptive
Human-in-the-loop
Blocking
Non-blocking
Output Type
Subset
Transform
Derived
Fixed
Proportional

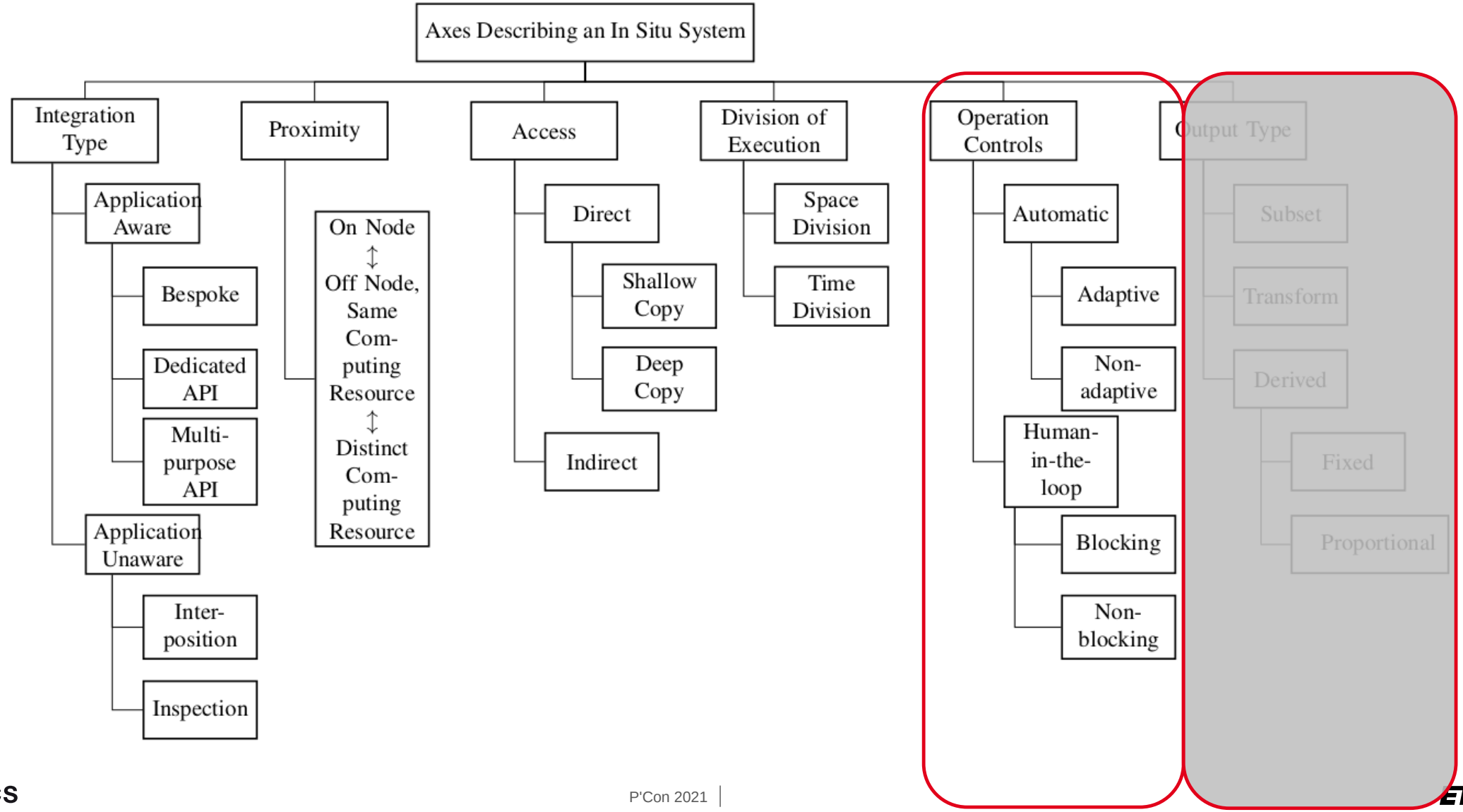
Access


Axes Describing an In Situ System
Integration Type
Application Aware
Bespoke
Dedicated API
Multi-purpose API
Application Unaware
Inter-position
Inspection
Proximity
On Node
Off Node, Same Com-puting Resource
Distinct Com-puting Resource
Access
Direct
Shallow Copy
Deep Copy
Indirect
Division of Execution
Space Division
Time Division
Operation Controls
Automatic
Adaptive
Non-adaptive
Human-in-the-loop
Blocking
Non-blocking
Output Type
Subset
Transform
Derived
Fixed
Proportional

Division of Execution

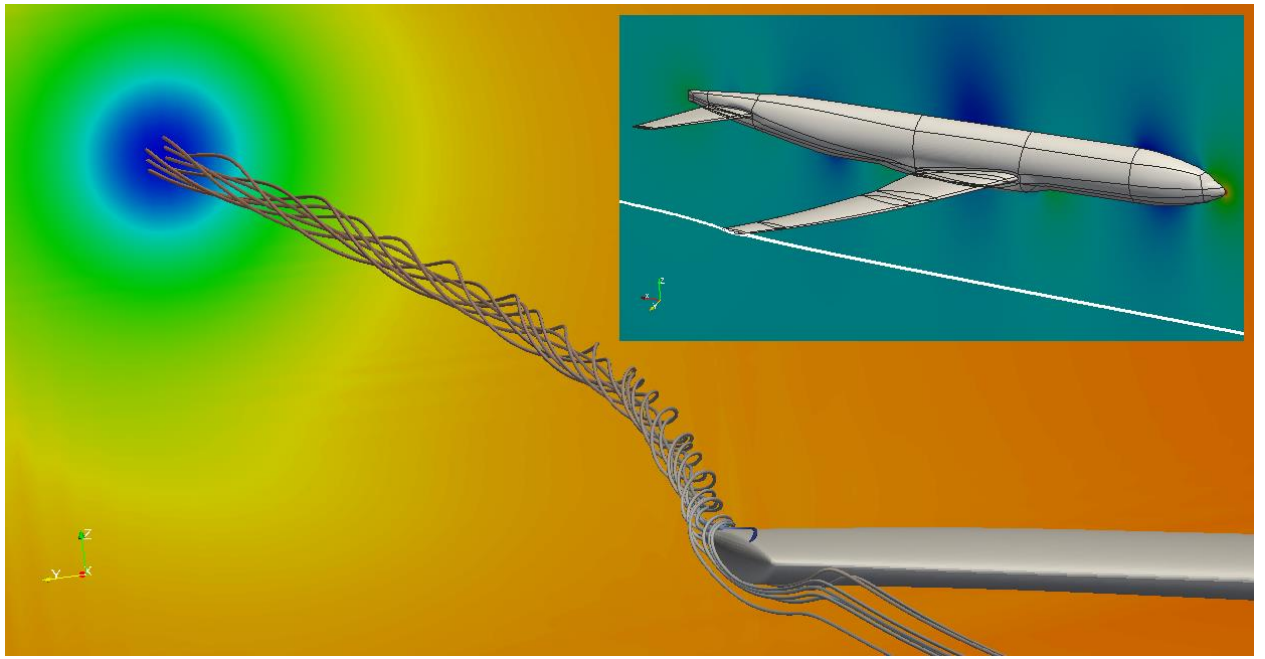


Operation Controls



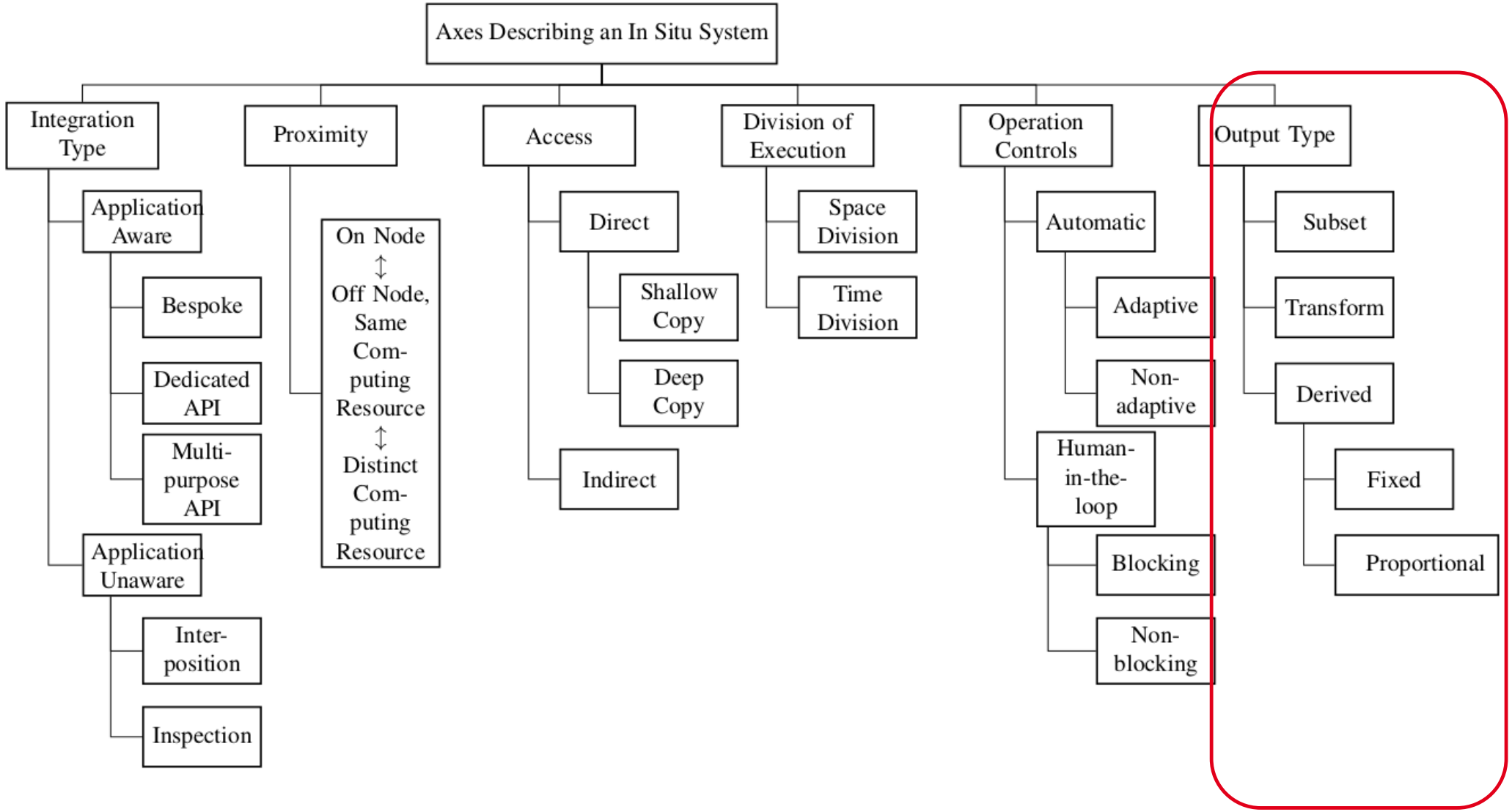
Feature extraction?

- What's a feature?
- Some small[er] data worth of interest because of domain knowledge



- Without "interactive" data exploration, it can be difficult to know a-priori which features to extract, or how to tune parameters.
- Knowing when to trigger a potentially expensive calculation would also be handy.

Output types



Outcome

- In that paper, 15 existing systems were reviewed. Half of them implement a Time Division, with On Node proximity.
  - The simulation advances, then pauses and hands control to the in situ system which completes its operations, hands control back, and so on
  - For example: Ascent, ParaView-Catalyst, Visit/Libsim

- Other examples, ADIOS, SENSEI, which can operate in different modes, sending also their data to distinct vis resources

- ADIOS, for example, can integrate with other workflow or data analytics systems without detailed knowledge of the underlying software and hardware stack
- ADIOS allows users to combine data storage, data staging, data compression, and/or data reduction (ZFP, SZ, BZip2 (compression), FlexPath, Dataspace (data staging), and coupling with ParaView, Visit, Ascent)

SENSEI

SENSEI provides a generic API to enable a “write-once, run-anywhere” environment. This approach focuses on data proximity and portability, runtime selection between running On Node or Off Node

SENSEI provides access to a diverse set of in situ analysis back-ends and transport layers such as ADIOS, Libsim, Ascent, Catalyst etc , through a simple API and data model.

Simulations instrumented with the SENSEI API can process data using any of these back-ends interchangeably. The back-ends are selected and configured at run-time via an XML configuration file.

For more details, refer to the SC2021 tutorial notes

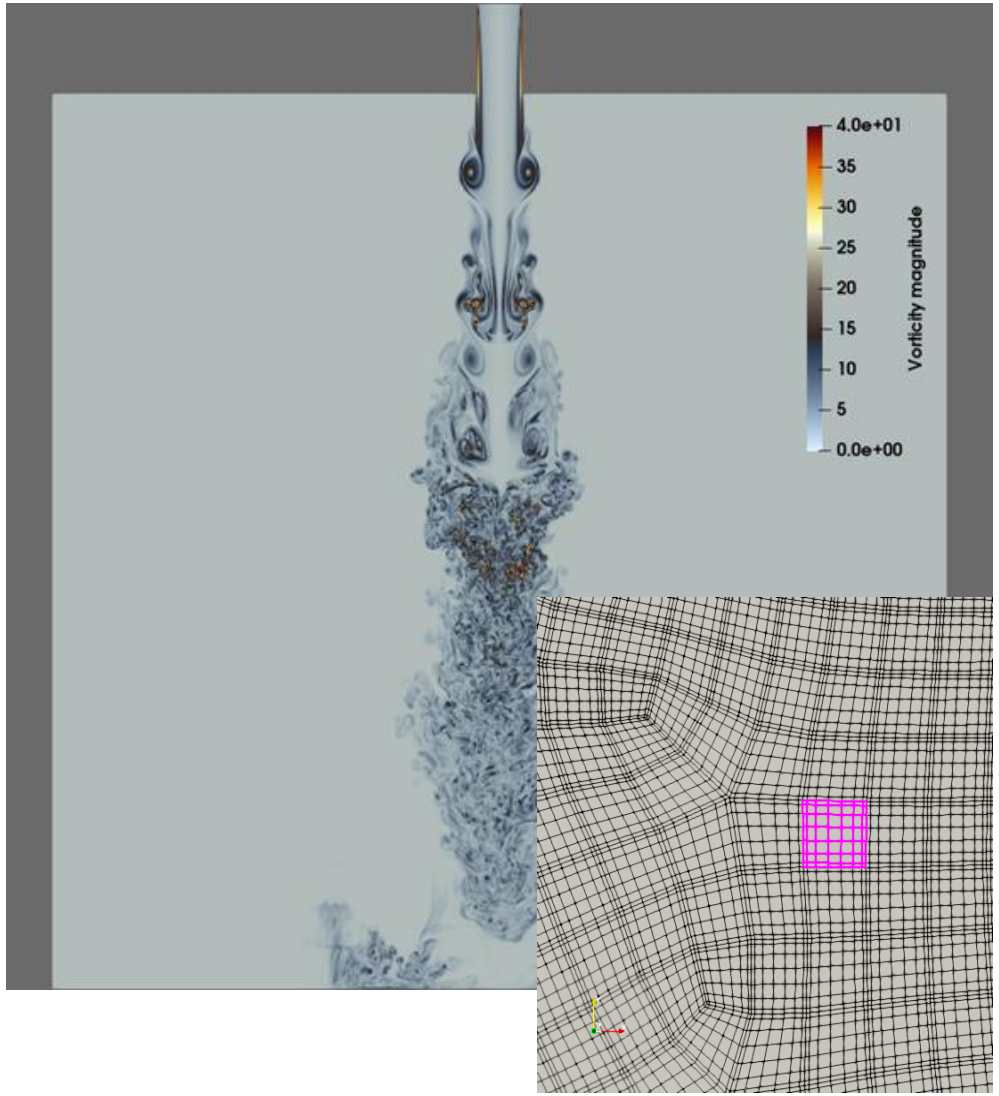
What about my [the solver] internal data model?

- Do the standard visualization applications support all the data structures I use in my code?
  - => Use Data Adaptors
- Are our standard visualization applications ready to handle new alternate data representations?
  - => Use Data Converters

- In recent years, we have seen new ways of thinking about data simulations (run multiphysics code, run ensembles, use ML).
- => new data, perhaps quite different from the traditional "mesh of gridded points"

Even the mesh of gridded points can raise challenges

- Take the Nek5000 solver for aerothermochemistry, combustion and fluid dynamics problems.
- Simulations use spectral elements and n-th order polynomials.
- For GTC 2021, we needed to use NVIDIA IndeX, a state-of-the-art volume renderer for unstructured data.
- In a post-hoc scenario, we were forced to convert spectral elements to a tetrahedral mesh.
- See the full 3D movie here

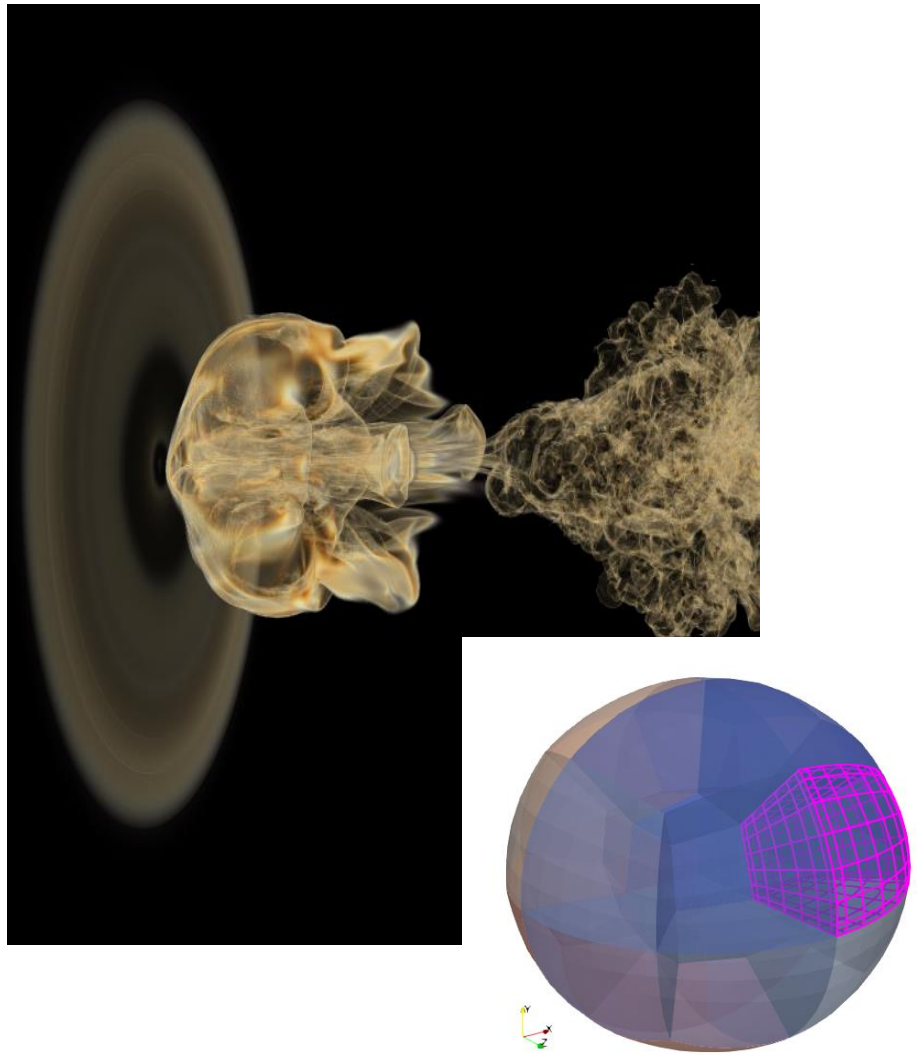


NVIDIA IndeX capable of volume rendering billions of cells

Running on Piz Daint with Cray compute nodes and NVIDIA P100 GPUs

We looked at two cases:

	Medium resolution	High resolution
# spectral elements	444,320 (8-th order)	584,304 (12-th order)
# of hexahedra	152,401,760	777,708,624
# of tetrahedra	914,410,560	4,666,251,744
# of GPUs	16-32	64



Exotic element type support coming soon…to the visualization side

Example:

- Add Discontinuous Galerkin Finite Element Mesh Support to VTK/ParaView
- Issue #21120

A showcase and practical experience

Feedback on my first real exercise of code instrumentation

Instrument a solver,

and use three different environments to interface with a visualization package:

1. Use ParaView Catalyst and replay ParaView Python scripts
   - We will “only” need to describe the data layout in memory
2. Use Ascent and its own renderer
   - Must describe not only the data, but also what to do with it and how to create images
3. Use Ascent to pass “data” to a ParaView Python plugin
   - Tell Ascent that we will use a ParaView script

ParaView, a multi-purpose scientific visualization application

- ParaView is an open-source, multi-platform data analysis and visualization application, developed by Kitware.
- ParaView was developed to analyze extremely large datasets using distributed memory computing resources.
- ParaView can run:
  - on the desktop
  - in client-server mode for interactive exploration of data,
  - in batch mode with Python-driven scripts
  - in-situ scripts when coupled with a simulation code

Catalyst

- Catalyst is an API specification developed for simulations (and other scientific data producers) to analyze and visualize data *in situ*.

- It includes the following:

  - A light-weight implementation of the Catalyst API, currently called a stub library
  - An SDK for developers to develop implementations of the Catalyst API to perform custom data processing and visualization tasks.

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

- The Catalyst API uses Conduit for describing data and other parameters which can be communicated between a simulation and Catalyst.

Conduit first example

cd /local/apps/ascent/install/examples/ascent/tutorial/ascent_intro/notebooks

jupyter lab 02_conduit_basics.ipynb

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

First Method. Use ParaView Catalyst

Catalyst : a VTK-based API vs. a Conduit-based API

Original API (ParaView before v5.9)

- Writing a data adaptor requires an intimate understanding of VTK. VTK Mesh types, Data types.
- Knowledge of how to efficiently transform simulation data structures to VTK is a must.
- Not for the faint-at-heart
- Not for a simulation code developer, without a hobby for Visualization software development

Catalyst : a VTK-based API vs. a Conduit-based API

Original API (ParaView before v5.9)

- Writing a data adaptor requires an intimate understanding of VTK. VTK Mesh types, Data types.
- Knowledge of how to efficiently transform simulation data structures to VTK is a must.
- Not for the faint-at-heart
- Not for a simulation code developer, without a hobby for Visualization software development

(with v5.9 and onwards)

- Writing an adaptor requires a discreet knowledge of Conduit, which I acquired in one afternoon.
- Catalyst is a lightweigth API and a simulation can be linked against its stub implementation, without requiring a ParaView SDK
- A ParaView 5.9+ build provides a Catalyst API implementation, which is now referred to as *ParaView-Catalyst*
- Catalyst in ParaView 5.9: Blog article

For the rest of the discussion, we will now assume a ParaView 5.9+ environment and the use of the new Conduit-based API

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

The ParaView Extractors

- Extractors are items in the visualization pipeline that can save data or images at a user-chosen temporal frequency.
  - Data Extractors
  - Image Extractors

- Using Extractors, no custom code per iteration is really necessary in the majority of cases. One can simply use Extractors to save out images from views or data extracts from filters and other data producers. (*more on that later*)

The ParaView Catalyst Python scripts

- Python scripts, which are written by the ParaView application, given a representative template input file, and a set of visualization filters interactively tuned by the user in an *offline* fashion [not connected to a running solver]
- We create a visualization pipeline with the numerous visualization filters and rendering options available in ParaView, and we add “Extractors”
- ParaView provides hybrid parallelism out-of-the-box
  - MPI-enabled
  - TBB multi-threading
  - CUDA-enabled filters
- Some [or most of the] ParaView visualization code will be tightly integrated with the solver memory and execution space.

The ParaView Catalyst Python scripts

- Can be generated from the GUI, and later fine-tuned
- 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

The Protocol is rather simple:

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

- Defines the protocol for **catalyst_execute()** and includes information about Catalyst channels i.e. ports on which data is made available to in situ processing as well as the actual data from the simulation

- Defines the protocol for **catalyst_finalize()**

ParaView-Catalyst Blueprint

Similar to the Conduit Mesh Blueprint.

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

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

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

Example: Instrument an SPH simulation package with ParaView Catalyst

- 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++17 headers-only code with no external software dependencies. The parallelism is currently expressed via the following models: MPI, OpenMP, CUDA and HIP.

- We started instrumenting an SPH-EXA application with ParaView-Catalyst

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_cpp::Node exec_params;
// using ParaView Catalyst Blueprint
auto channel = exec_params["catalyst/channels/grid"];
// using Conduit Mesh Blueprint to define the mesh
channel["type"].set("mesh");
auto mesh = channel["data"];
mesh["coordsets/coords/type"].set("explicit");
mesh["coordsets/coords/values/x"].set_external(sim.x);
mesh["coordsets/coords/values/y"].set_external(sim.y);
mesh["coordsets/coords/values/z"].set_external(sim.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(sim.N);
std::iota(conn.begin(), conn.end(), 0);
mesh["topologies/mesh/elements/connectivity"].set(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(sim.rho);
```

Use one of conduit's relay protocol to save data to disk?

```
import conduit.relay as relay


if relay.io.about()["protocols/hdf5"] == "enabled":
relay.io.save(mesh, "/dev/shm/foo.hdf5")
```

```
h5ls -r /dev/shm/foo.hdf5
/                        Group
/coordsets               Group
/coordsets/coords        Group
/coordsets/coords/type   Dataset {9}
/coordsets/coords/values Group
/coordsets/coords/values/x Dataset {4}
…
/fields                  Group
/fields/rho              Group
/fields/rho/values       Dataset {4}
….
/topologies              Group
/topologies/mesh         Group
/topologies/mesh/coordset Dataset {7}
/topologies/mesh/elements Group
/topologies/mesh/elements/connectivity Dataset {4}
```

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(“detecting a condition at timestep”, node.timestep)
    # Extract particles whose Density is greater than threshold
  else:
    # use ALL particles
```

What about “Operations Control”?

Human in the loop

ParaView has a very efficient client-server mode for post-hoc visualization

- Uses a powerful, remote server to do the heavy work:
  - Parallel I/O
  - Parallel filtering
  - Parallel Image Composition
- Has a very efficient, threshold-based, Image Delivery engine
- The results of the different visualization operations stay on the server (Mbytes, Gbytes, Tbytes (choose your flavour))
- Only the image gets sent over the network to the client (1024x1024 pixels, RGB).

ParaView-Catalyst uses the same engine

- Open a port on a reverse tunnel to my desktop
  - ssh -R 22222:localhost:22222 daint103.cscs.ch

  - Tell my desktop client to send a request for connection on that port
  - my code instrumentation listens to incoming calls with four additional lines of Python code:

```
from paraview import catalyst
options = catalyst.Options()
options.EnableCatalystLive = 1
options.CatalystLiveURL = 'daint103:22222'
```

Demonstration of Catalyst Live

Other examples: LULESH instrumented for ParaView Catalyst

ASC Proxy Apps

Instrumented with the VTK-based Catalyst

Instrumented with the Conduit-based Catalyst

Thanks to Utkarsh Ayachit (Kitware)

Second Method. Use Ascent

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

"*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

Ascent slides from the recent SC2021 tutorial

Thanks go to Cyrus Harrison and colleagues.

Ascent

Ascent is 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)

Ascent tutorial first example

cd /local/apps/ascent/install/examples/ascent/tutorial/ascent_intro/notebooks

jupyter lab 01_ascent_first_light.ipynb

Ascent tutorial pages

- Tutorial link
- SC2021 tutorial

- docker run -p 8888:8888 -t -i alpinedav/ascent-jupyter

Ascent tutorial examples,

Generating time dependent data

- jupyter lab 03_conduit_blueprint_mesh_examples.ipynb

Rendering images with Scenes

- jupyter lab 04_ascent_scene_examples.ipynb

Transforming data with Pipelines

- jupyter lab 05_ascent_pipeline_examples.ipynb

Application: Instrument the SPH-EXA simulation package with Ascent

Trivial !

- Reuse the Conduit mesh node definition from the ParaView Catalyst code

- Add a basic scene definition

About 100 lines of code. Total!

Running with different visualization pipelines

In its basic form, we have hard-wired a default scene description into the Ascent bridge code

- At run-time, we can supply different scenes, with additional pipelines
- If present, a file called "ascent_actions.json" will be read, overriding the actions previously set

Let’s add a scene description

```
"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]
                       }
```

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"
                    }
                },
```

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",
```

Third Method. Use Ascent and ParaView Python scripts

Ascent, passing data to a ParaView script

- Ascent Documentation

- The instrumented simulation sends a tree structure (json like) that describes the simulation data using the Conduit Blueprint Mesh specification.
- A ParaView plugin constructs one of the following datasets: vtkImageData, vtkRectilinearGrid, vtkStructuredGrid or vtkUnstructuredGrid
- This data is converted to a VTK format using shallow copies for data arrays

Ascent, passing data to a ParaView script

- Ascent's "action" node is:

```
"action": "add_extracts",
   "extracts":
   {
     "e1":
     {
       "type": "python",
        "params":
        {
        "file": "paraview-vis.py“
        }
     }
 }
```

- The ParaView script describes a "standard" ParaView pipeline and can save images and any other derived data to disk

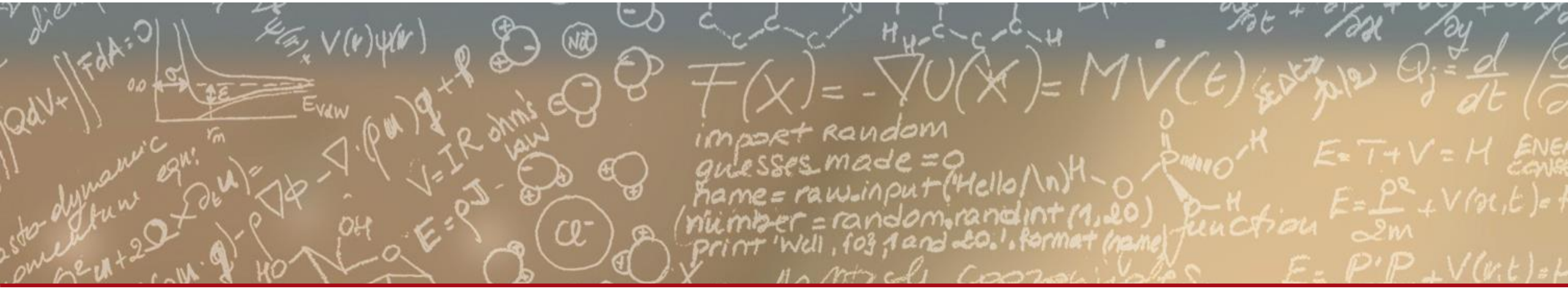
Summary

- I have passed the original early adopter phase
- A lot of ground still has to be covered, to evaluate performance, memory consumption, visualization pipeline creation and tuning, and re-usability

- For the first time in several decades of developing data formats plugins, guiding code developers on how to most effectively store their data to disk, I don’t have to worry about this anymore.
- The Conduit-based interfaces bring a very easy-to-use, and very intuitive interface to data in memory. The bridges to ParaView or Ascent are now hiding all the complexity from the code developers.

Resources

- My slides
- https://dav.lbl.gov/events/SC21_SENSEI_Ascent_Tutorial/



Thank you for your attention.