

# Conduit

## Simplified Data Exchange for HPC Simulations

<https://github.com/llnl/conduit>

LANL Data Science at Scale Summer School

Wednesday July 21st, 2021

Cyrus Harrison (cyrush@llnl.gov)

Joe Ciurej

Matt Larsen

(and other Conduit contributors)



# Acknowledgements

---

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.  
Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

#### Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

# Abstract

---

- Conduit (<https://github.com/llnl/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 to share data in-memory, for serialization, and for I/O.
- Conduit was built around the concept that an in-memory data description capability simplifies common tasks in the HPC simulation eco-system and helps connect software components. Sharing simulation meshes across software components is one of the primary use cases for Conduit. To support this Conduit provides the Mesh Blueprint ([https://llnl-conduit.readthedocs.io/en/latest/blueprint\\_mesh.html](https://llnl-conduit.readthedocs.io/en/latest/blueprint_mesh.html)), a set of hierarchical conventions for describing mesh-based simulation data.
- This talk introduces Conduit, the Mesh Blueprint, and the motivation for these tools.

# References (Presentations and Related Papers)

---

- [“The Conduit Mesh Blueprint: Drafting a New Way to Share Simulation Meshes”](#)  
DOE Computer Graphics Forum Talk, April 2019
- [“The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman”](#)  
In Proceedings of ISAV 2017 (SC17) Workshop, Denver CO, November 2017
- SciPy 2016 talk on Conduit ([https://youtu.be/3\\_GKjeRUPKg](https://youtu.be/3_GKjeRUPKg))
- [“Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes”](#)  
In Proceedings of ISAV 2015 (SC15) Workshop, Austin TX, November 2015

# References (Tutorials)

---

- Conduit User Tutorials (C++ & Python)  
<https://llnl-conduit.readthedocs.io/en/latest/conduit.html>
  
- Ascent Conduit Intro (C++ & Python)  
[https://ascent.readthedocs.io/en/latest/Tutorial Intro Conduit Basics.html](https://ascent.readthedocs.io/en/latest/Tutorial%20Intro%20Conduit%20Basics.html)

# This talk is about Conduit and the Mesh Blueprint

---

- Conduit provides a set of tools focused on in-memory data description to aid with sharing data across the HPC ecosystem
- The Mesh Blueprint is a set of hierarchical conventions (built using Conduit) to describe mesh-based simulation data both in-memory and via files

# Why should you be interested in these topics?

---

- General use of Conduit may make life easier if you are writing HPC simulation or data analysis software
  
- The Mesh Blueprint is being adopted as a data interface to DOE supported open source in situ visualization tools:
  - Ascent (<http://ascent-dav.org/>)
  - Catalyst (<https://catalyst-in-situ.readthedocs.io>)
  - SENSEI (<https://sensei-insitu.org/>)

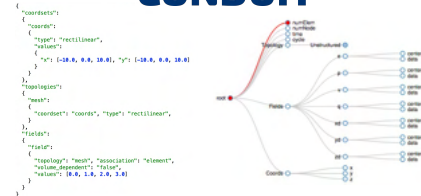
---

# Introduction to Conduit

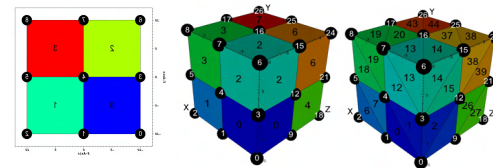


# Conduit provides intuitive APIs for in-memory data description and exchange

- **Provides an intuitive API for in-memory data description**
  - Enables *human-friendly* hierarchical data organization
  - Can describe in-memory arrays without copying
  - Provides C++, C, Python, and Fortran APIs
- **Provides common conventions for exchanging complex data**
  - Shared conventions for passing complex data (e.g. *Simulation Meshes*) enable modular interfaces across software libraries and simulation applications
- **Provides easy to use I/O interfaces for moving and storing data**
  - Enables use cases like binary checkpoint restart
  - Supports moving complex data with MPI (serialization)



Hierarchical in-memory data description



Conventions for sharing in-memory mesh data

<http://software.llnl.gov/conduit>  
<http://github.com/llnl/conduit>

Website and GitHub Repo

# The heart of Conduit is a hierarchical variant type: *Node*

---

- Let's look at some examples of how to create a *Node*
- [https://ascent.readthedocs.io/en/latest/Tutorial Intro Conduit Basics.html](https://ascent.readthedocs.io/en/latest/Tutorial%20Intro%20Conduit%20Basics.html)
- (Demo of Ascent Tutorial Conduit Basics Jupyter Notebook)

# Conduit API Example: Create a tree of data arrays

```
// create a "Node", the primary object in conduit
conduit::Node n;
// init our Node hierarchy with a few data arrays
n["coordsets/coords/values/x"] = {0.0,1.0,2.0};
n["coordsets/coords/values/y"] = {0.0,1.0,2.0};
n["fields/density/values"] = {1.0,1.0,1.0,1.0};
n.print();
```

**Foreshadowing:**  
These don't have to be  
Contiguously allocated arrays

```
coordsets:
  coords:
    values:
      x: [0.0, 1.0, 2.0]
      y: [0.0, 1.0, 2.0]
  fields:
    density:
      values: [1.0, 1.0, 1.0, 1.0]
```

# Conduit API Example:

## Mixing externally allocated and Conduit owned data

```
// you can mix external and conduit owned data within a  
// Node hierarchy  
std::vector<conduit::float64> vel_u(9,1.0);  
std::vector<conduit::float64> vel_v(9,1.0);  
  
// use Node::external to init the "u" and "v" nodes of the  
// tree to point to the same memory location of the source vectors.  
n["fields/velocity/values/u"].set_external(vel_u);  
n["fields/velocity/values/v"].set_external(vel_v);
```

# Conduit API Example:

## Mixing externally allocated and Conduit owned data

```
// show the elements of the "u" array  
n["fields/velocity/values/u"].print();
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

```
// change the first element of the u array (via the vector)
```

```
vel_u[0] = 3.14159;
```

```
// show the elements of the "u" array again  
n["fields/velocity/values"].print();
```

```
u: [3.14159, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]  
v: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

# Conduit API Example:

## Mixing externally allocated and Conduit owned data

```
// mixed ownership semantics allow you to organize,  
// extend, and annotate existing data  
n["coordsets/coords/type"] = "rectilinear";  
n["fields/density/units"] = "g/cc";  
n["fields/velocity/units"] = "m/s";  
  
n.print();
```

```
coordsets:  
  coords:  
    values:  
      x: [0.0, 1.0, 2.0]  
      y: [0.0, 1.0, 2.0]  
    type: "rectilinear"  
fields:  
  density:  
    values: [1.0, 1.0, 1.0, 1.0]  
    units: "g/cc"  
  velocity:  
    values:  
      u: [3.14159, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]  
      v: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]  
    units: "m/s"
```

# We aim to provide intuitive C++, Python, and Fortran APIs

## C++

```
std::vector<float64> den(4,1.0);

conduit::Node n;
n["fields/density/values"] = den;
n["fields/density/units"] = "g/cc";

Node &n_den = n["fields/density"];

float64 *den_ptr = n_den["values"].value();
std::string den_units = n_den["units"].as_string();

n_den.print();

std::cout << "\nDensity (" << den_units << "):\n";
for(index_t i=0; i < 4; i++)
{
    std::cout << den_ptr[i] << " ";
}

std::cout << std::endl;
```

## Python

```
import conduit
import numpy as np

den = np.ones(4,dtype="float64")

n = conduit.Node()
n["fields/density/values"] = den
n["fields/density/units"] = "g/cc"

n_density = n.fetch("fields/density")

den_vals = n_density["values"]
den_units = n_density["units"]

print(n_density)

print("\nDensity ({0}):\n{1}").format(den_units,
                                     den_vals)
```

## Fortran

```
do i = 1,4
    den(i) = 1.0
enddo

n = conduit_node_obj_create()
call n%set_path_ptr("fields/density/values", den, 4_8)
call n%set_path("fields/density/units", "g/cc")

n_den = n%fetch("fields/density")

call n_den%fetch_path_as_float64_ptr("values", d_arr)

n_den_units = n_den%fetch("units")
units_len = n_den_units%number_of_elements()

call n_den_units%as_char8_str(units)

call n_den%print()

print *, "Density (", (units(i),i=1,units_len), "):"

do i = 1,4
    write (*,"(f5.2,1x)",advance="no") d_arr(i)
enddo
print *

call conduit_node_obj_destroy(n)
```

# We aim to provide intuitive C++, Python, and Fortran APIs

## C++

```
values: [1.0, 1.0, 1.0, 1.0]  
units: "g/cc"
```

```
Density (g/cc):  
1 1 1 1
```

## Python

```
values: [1.0, 1.0, 1.0, 1.0]  
units: "g/cc"
```

```
Density (g/cc):  
[1. 1. 1. 1.]
```

## Fortran

```
values: [1.0, 1.0, 1.0, 1.0]  
units: "g/cc"
```

```
Density (g/cc):  
1.00 1.00 1.00 1.00
```



# The heart of Conduit is a hierarchical variant type: *Node*

## A *Node* acts as one of following basic roles:

- *Object*: An ordered associative array mapping names to children
- *List*: An ordered list of unnamed children
- *Leaf*: Scalar or 1D Array of bitwidth-specified primitives:
  - *Signed Integers*: int8, int16, int32, int64
  - *Unsigned Integers*: uint8, uint16, uint32, uint64
  - *Floating Point Numbers*: float32, float64
  - *Strings*: char8\_str
- *Empty*: No data

Experience with NumPy and JSON motivated Conduit's data model

## Beyond setup and access methods, *Node* also provides methods to help you easily process and compare trees

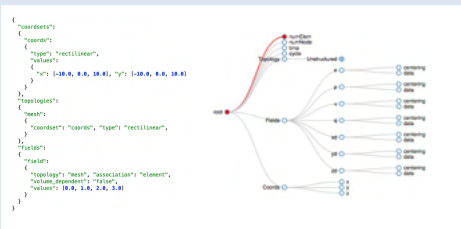
---

- Inspect the Memory layout of
  - Leaves
  - Entire Trees
- Print Tree Summaries
- Calculate Leaf Summary Metrics
- Compact / Serialize Trees
- Iterate over Children
- Calculate Tree Differences (``diff`` with adjustable tolerance)
- Parse and Create
  - YAML
  - JSON

# Conduit's *Relay* and *Blueprint* libraries provide features built on top of Conduit's core data model

## Conduit

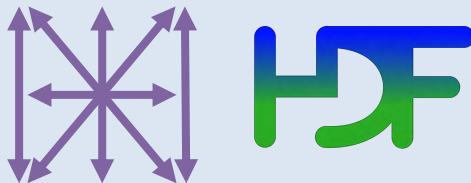
Implements interfaces to Conduit's in-memory data model



- Core Objects
- YAML/JSON parsing
- Basic I/O
- Basic transforms

## Relay

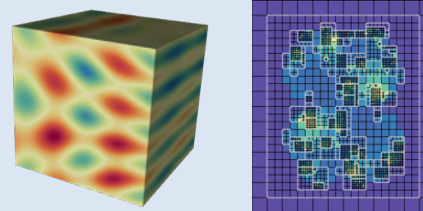
Provides advanced I/O features built on top of Conduit's data model



- File-based I/O: HDF5, Silo
- MPI
- WebSockets
- ZFP

## Blueprint

Supports shared higher level conventions for using Conduit to represent data



- Computational Meshes
- Multi-component Arrays
- One-to-many Relations
- Example Meshes
- Mesh Transforms

# Projects are now leveraging Conduit to support a wide range of capabilities

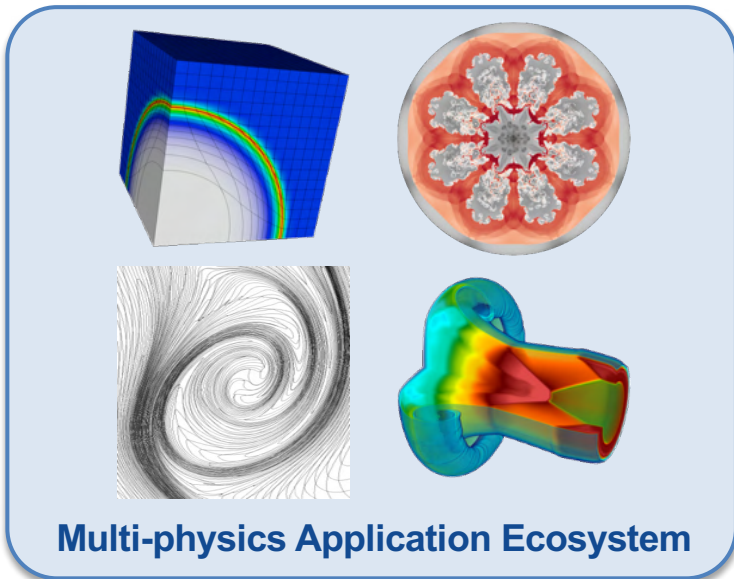
- Creating in-memory data stores
- Checkpoint restart of simulation data
- Node/YAML/JSON based:
  - Data APIs
  - User APIs
- Tree + Path-based I/O and partial I/O
- Moving complex data with MPI
- Distributing work with MPI
- Sharing ad-hoc data between programs written in multiple languages (both in-memory and via files)
- Sharing simulation mesh data (both in-memory and via files)

---

# What motivated Conduit?

# Predictive simulation requires advanced modeling tools and access to parallel computational horsepower

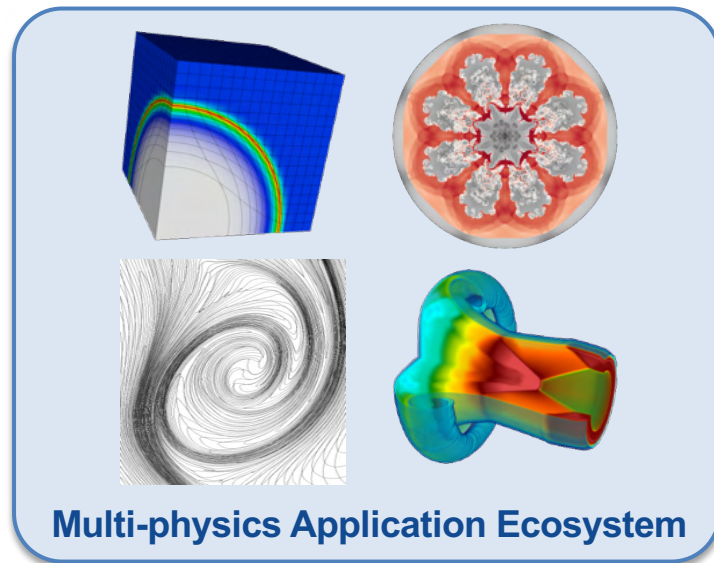
At LLNL, Weapon Simulation and Computing (WSC) develops an ecosystem of applications that provide validated physics modeling capabilities and fields the world-class HPC platforms needed to fully utilize these applications.



# LLNL's WSC Computational Physics (CP) program develops production HPC multi-physics simulation applications

## Snapshot of Scale and Complexity of LLNL's WSC CP Program:

- Development Efforts
  - ~120 Developers  
(Physicists, Engineers, Computer Scientists, Software Quality, etc ...)
  - ~15 project teams
  - ~15 – 30+ year application lifetimes
  - ~12 million lines of production code across projects  
(with more than 100 third-party dependencies)
- Diversity of Programming Languages and HPC Architectures
  - C++/C, Fortran, Python, Lua
  - Distributed Memory plus Multi-core or Many-core
- Diversity of Data
  - Scalars, Arrays, Tables, Contours, CAD Geometry, Meshes



CP's efforts are a microcosm of the broader HPC simulation community

# The software ecosystem supporting HPC multi-physics simulations is very complex

## Multi-physics Simulation Applications

### CS Infrastructure

- Input Parsing
- Steering
- Communication
- Parallelism Abstractions
- I/O
- In Situ Coupling

### Physics Packages

- Hydrodynamics
- Chemistry
- Thermal radiation
- *{and many more ...}*

### Physics Libraries

- Material Properties
- Material Models

### Numerical Libraries

- Linear Algebra
- Finite Elements

## Workflow Applications

### Problem Setup and Meshing

- Computational Geometry
- Mesh Generation
- Mesh Decomposition
- Mesh Overlay

### Visualization and Analysis

- Mesh Rendering
- Feature Extraction
- Simulated Diagnostics

### Uncertainty Quantification

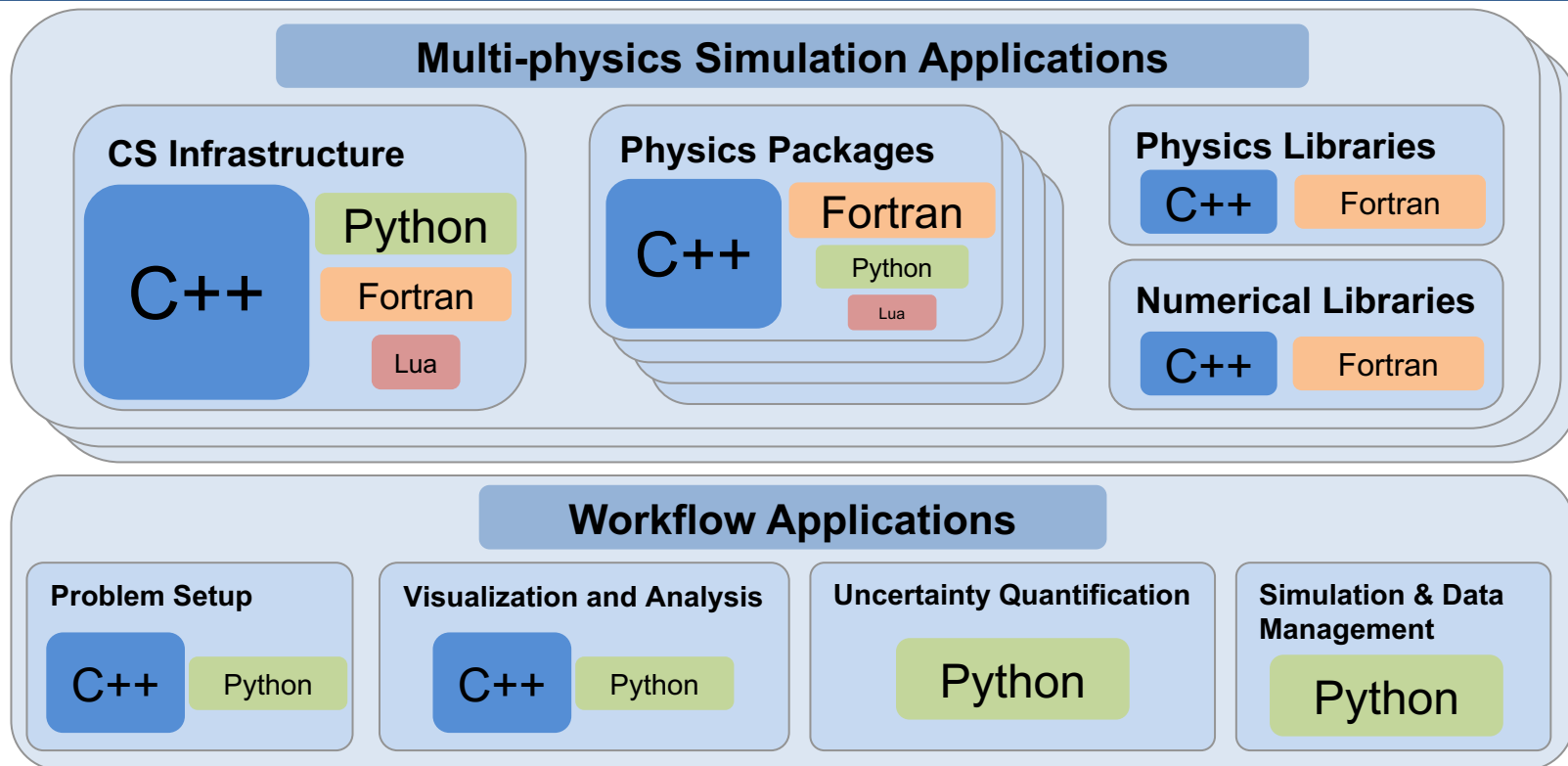
- Ensemble Generation
- Parametric Studies
- Statistical Models

### Simulation & Data Management

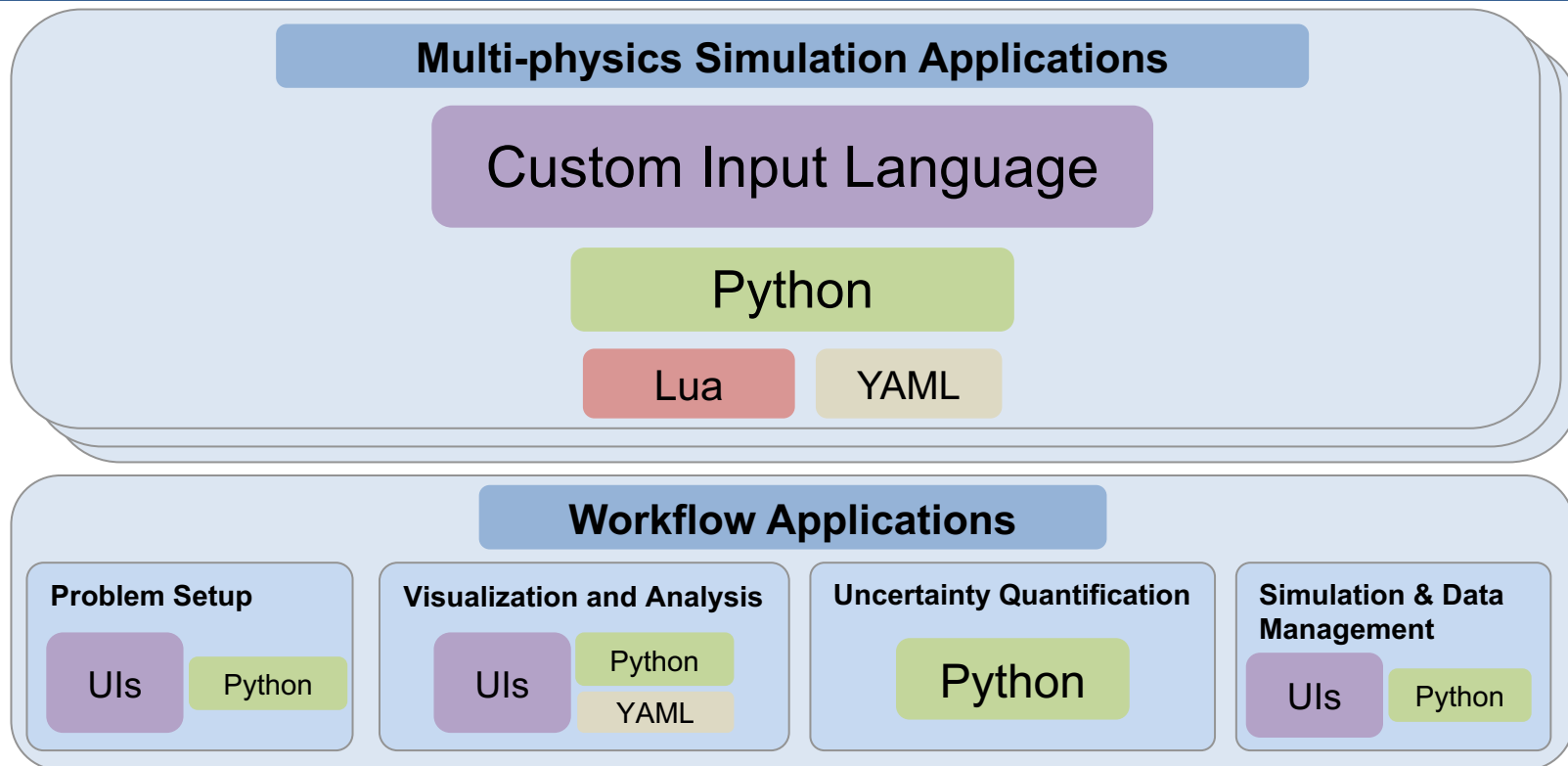
- Workflow Capture
- Data Organization
- Provenance



# A mix of C++, Python, Fortran and Lua are used to develop applications









# A mix of custom input languages, UIs, Python and Lua are used to run and script applications



# We are several years into a strategic focus to develop a modular software foundation for our ecosystem

Many of our investments are open source: <https://software.llnl.gov>

	RAJA	Performance Portability Layer (C++)
	MFEM	Modular parallel library for finite element methods
	AXOM	Core software components for HPC simulations
	Ascent	Flyweight in situ viz and analysis for HPC simulations
	Spack	A flexible package manager for HPC
	Conduit	Simplified Data Exchange for HPC Simulations

# Sharing data effectively is a key challenge to build our modular ecosystem

## How do we share data across these components?

- Traditional Support:
  - Simulations leverage their own bespoke in-memory mesh data models
  - File-based I/O libraries evolved into defacto interfaces
  - File-based I/O is acceptable for coarse-grain and low-frequency data sharing
- Future Goals:
  - Simplify connecting more components
  - Broadly support in-memory sharing

We need a flexible in-memory sharing solution to meet our goals

# We identified three key requirements to simplify in-memory data description

- Describe numeric primitives
  - Scalars, strided arrays, etc with explicit precision
- Support mixed memory ownership semantics
  - Enable zero-copy where feasible
  - Play friendly with existing data structures in multiple languages
- Enable higher level conventions
  - Hierarchical context (“key/value” + “paths” are a great interface ...)
  - Human readable descriptions

These requirements inspired Conduit

# We designed Conduit with software ecosystem logistics in mind

- Embodies our vision of data description as a core capability
  - Provide an in-memory foundation on which to build serialization, I/O, and messaging features
- Completely runtime focused
  - Avoids incompatible (or unshareable) code-generation solutions
- Provides a multi-language data model
  - APIs for C++, Python, C, and Fortran
  - YAML/JSON friendly

Philosophy: Share data without massive code infrastructure

---

# The Mesh Blueprint

# Many tools are adopting the Mesh Blueprint as an interface

---

- The Mesh Blueprint is being adopted as a data interface to DOE supported open source in situ visualization tools:
  - Ascent (<http://ascent-dav.org/>)
  - Catalyst (<https://catalyst-in-situ.readthedocs.io>)
  - SENSEI (<https://sensei-insitu.org/>)
- VisIt has a database reader for Blueprint-flavored HDF5/JSON/YAML fields
- MFEM supports converting their mesh data structures to/from Blueprint trees
- AMReX supports converting their mesh data structures to/from Blueprint trees



# The software ecosystem supporting HPC multi-physics simulations is very complex

## Multi-physics Simulation Applications

### CS Infrastructure

- Input Parsing
- Steering
- Communication
- Parallelism Abstractions
- I/O
- In Situ Coupling

### Physics Packages

- Hydrodynamics
- Chemistry
- Thermal radiation
- *{and many more ...}*

### Physics Libraries

- Material Properties
- Material Models

### Numerical Libraries

- Linear Algebra
- Finite Elements

## Workflow Applications

### Problem Setup and Meshing

- Computational Geometry
- Mesh Generation
- Mesh Decomposition
- Mesh Overlay

### Visualization and Analysis

- Mesh Rendering
- Feature Extraction
- Simulated Diagnostics

### Uncertainty Quantification

- Ensemble Generation
- Parametric Studies
- Statistical Models

### Simulation & Data Management

- Workflow Capture
- Data Organization
- Provenance

# The software ecosystem supporting HPC multi-physics simulations is very complex

## Multi-physics Simulation Applications

### CS Infrastructure

- Input Parsing
- Steering
- Communication

### Physics Packages

- Hydrodynamics
- Chemistry
- Thermal radiation

### Physics Libraries

- Material Properties
- Material Models

“Sharing” mesh data requires agreement on how to represent meshes ...

## Workflow Applications

### Problem Setup and Meshing

- Computational Geometry
- Mesh Generation
- Mesh Decomposition
- Mesh Overlay

### Visualization and Analysis

- Mesh Rendering
- Feature Extraction
- Simulated Diagnostics

### Uncertainty Quantification

- Ensemble Generation
- Parametric Studies
- Statistical Models

### Simulation & Data Management

- Workflow Capture
- Data Organization
- Provenance

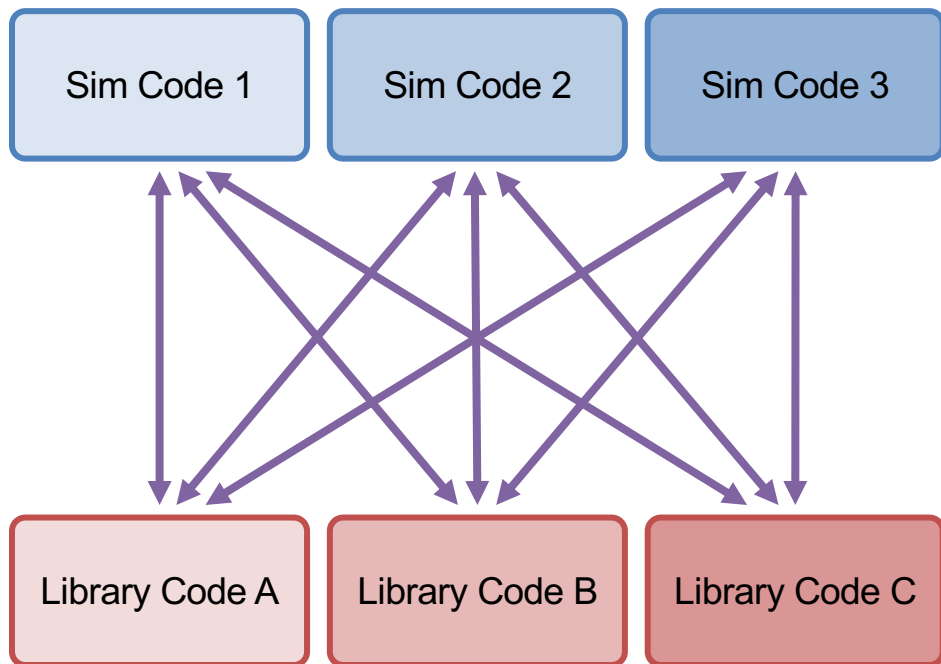
# Components of the ecosystem implement and leverage a wide range of mesh data structures and APIs

- Many simulation applications leverage their own bespoke in-memory mesh data models.
- Other tools leverage a range of mesh-focused toolkits, frameworks, and APIs including: VTK, VTK-m, MFEM, and SAMRAI
- Many powerful analysis tools are mesh agnostic (NumPy, PyTorch, etc) and recasting bespoke mesh data into these tools is a barrier to wide use

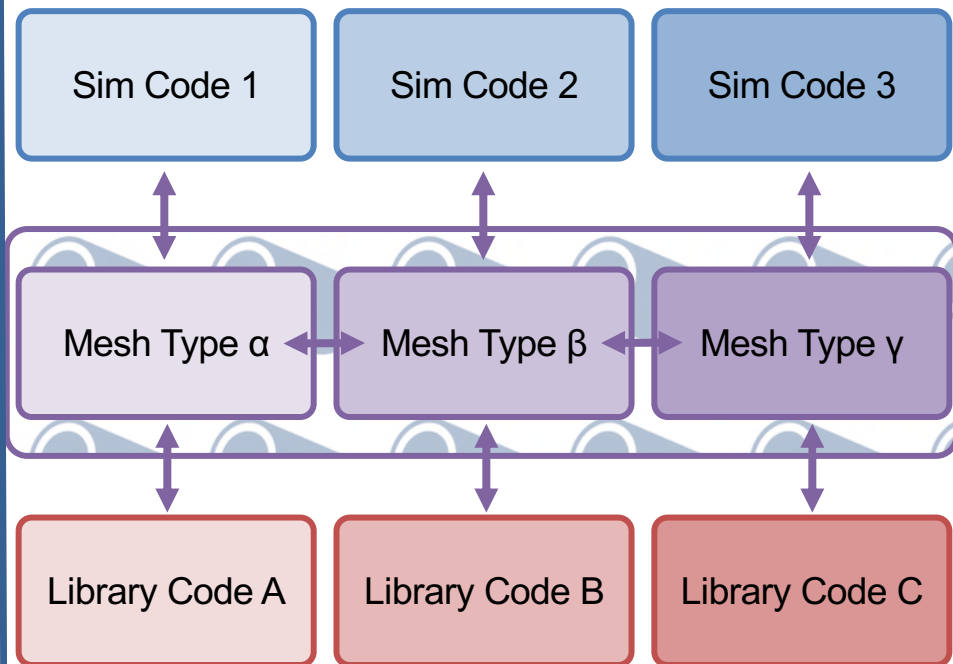
We never expect a single full fledged API to emerge that will cover all use cases across the ecosystem

# The Mesh Blueprint aims to provide an abstract, multi-hub interface to simplify interoperability

## Pre-Conduit/Blueprint Infrastructure

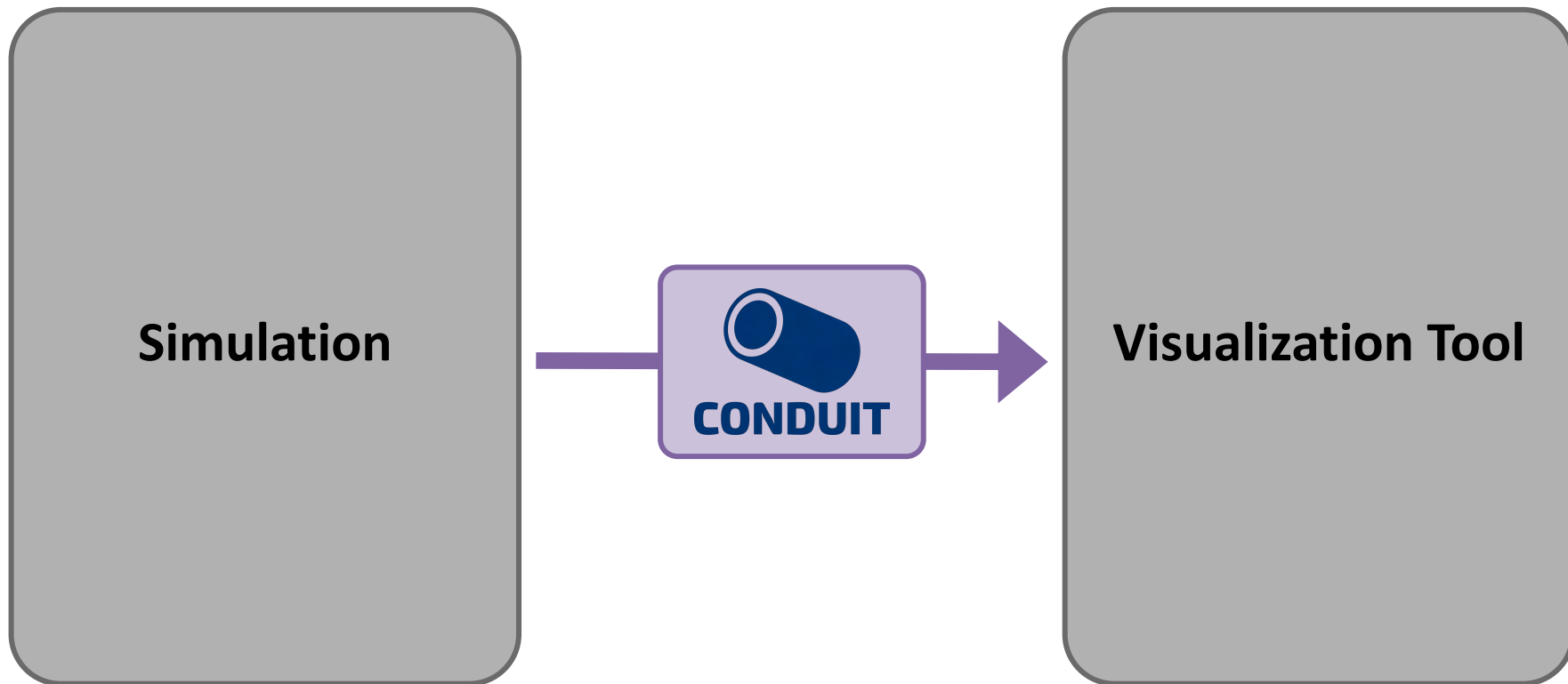


## Post-Conduit/Blueprint Infrastructure



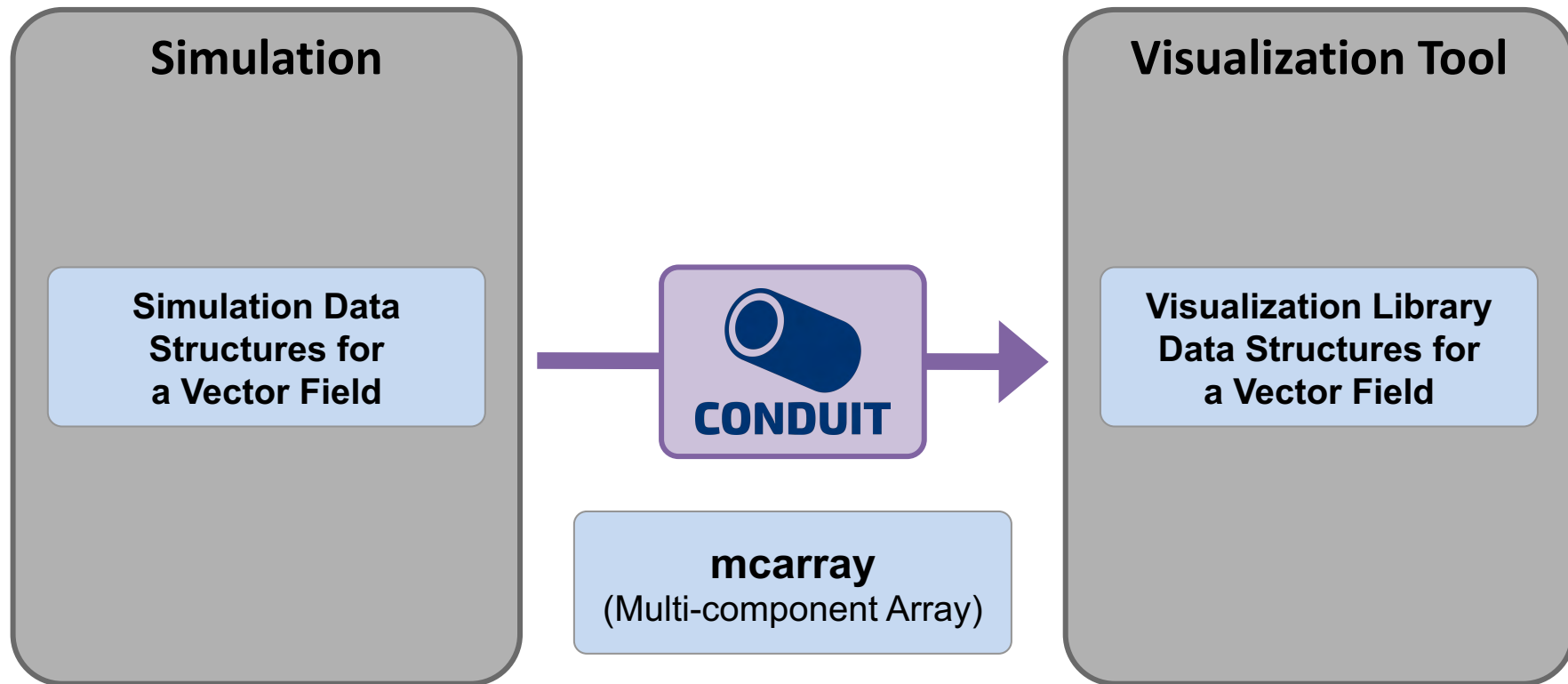
# Data description simplifies connecting software components

## Example: Sharing a Vector Field



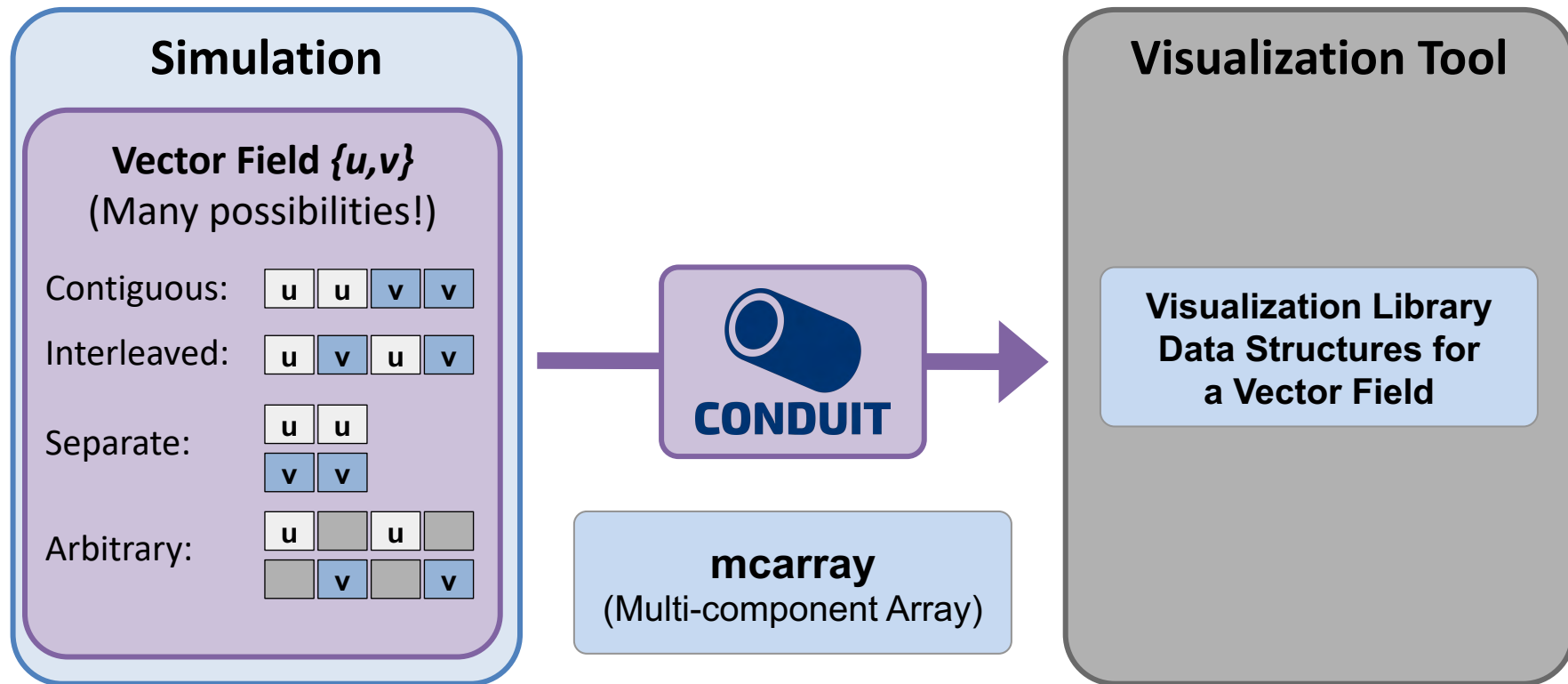
# Data description simplifies connecting software components

## Example: Sharing a Vector Field



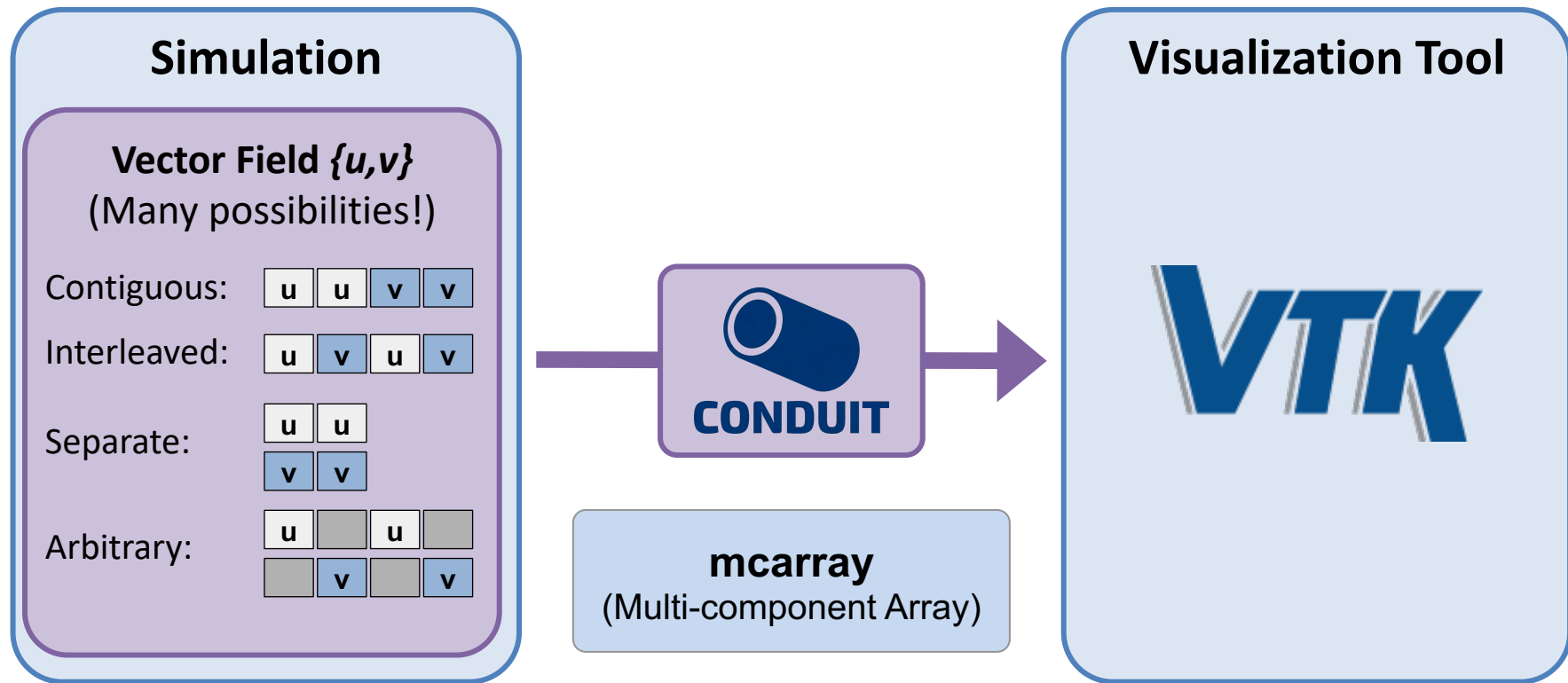
# Data description simplifies connecting software components

## Example: Sharing a Vector Field



# Data description simplifies connecting software components

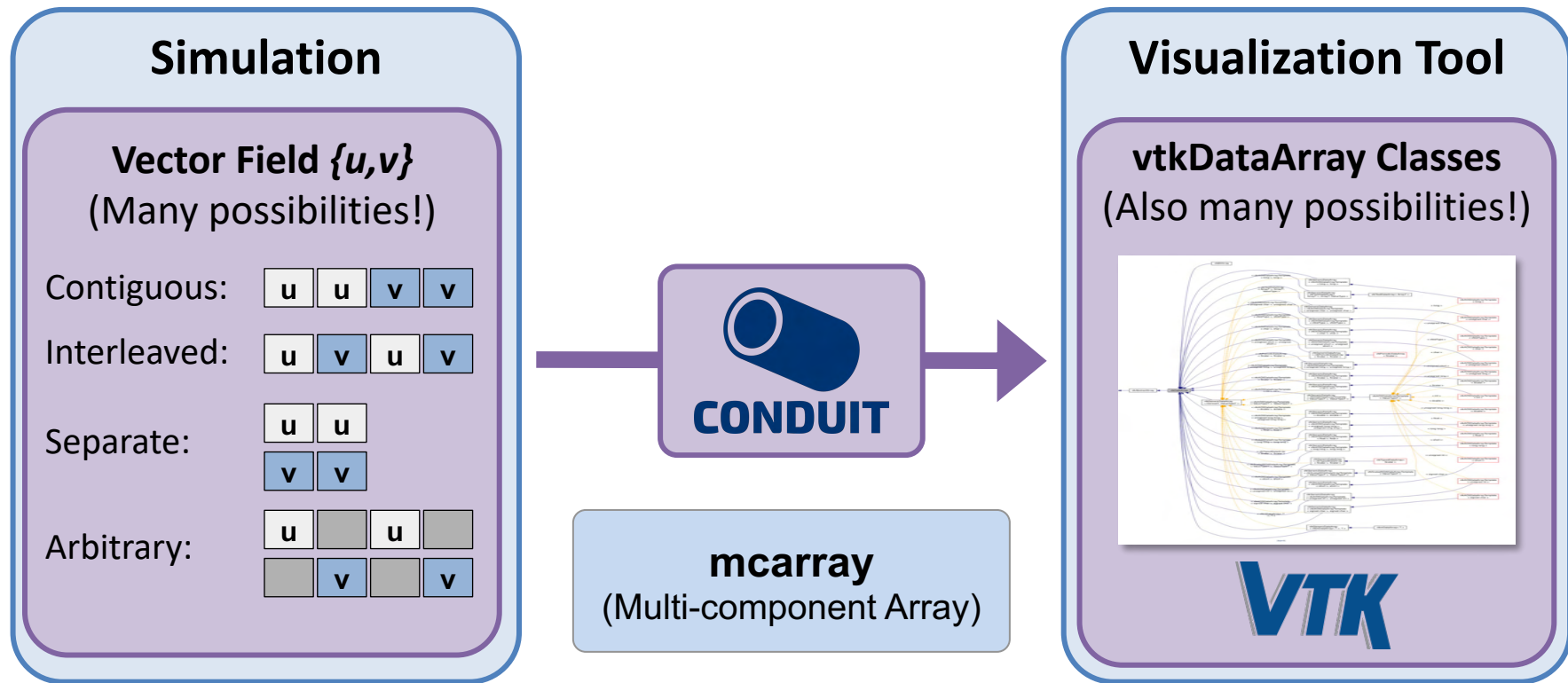
## Example: Sharing a Vector Field





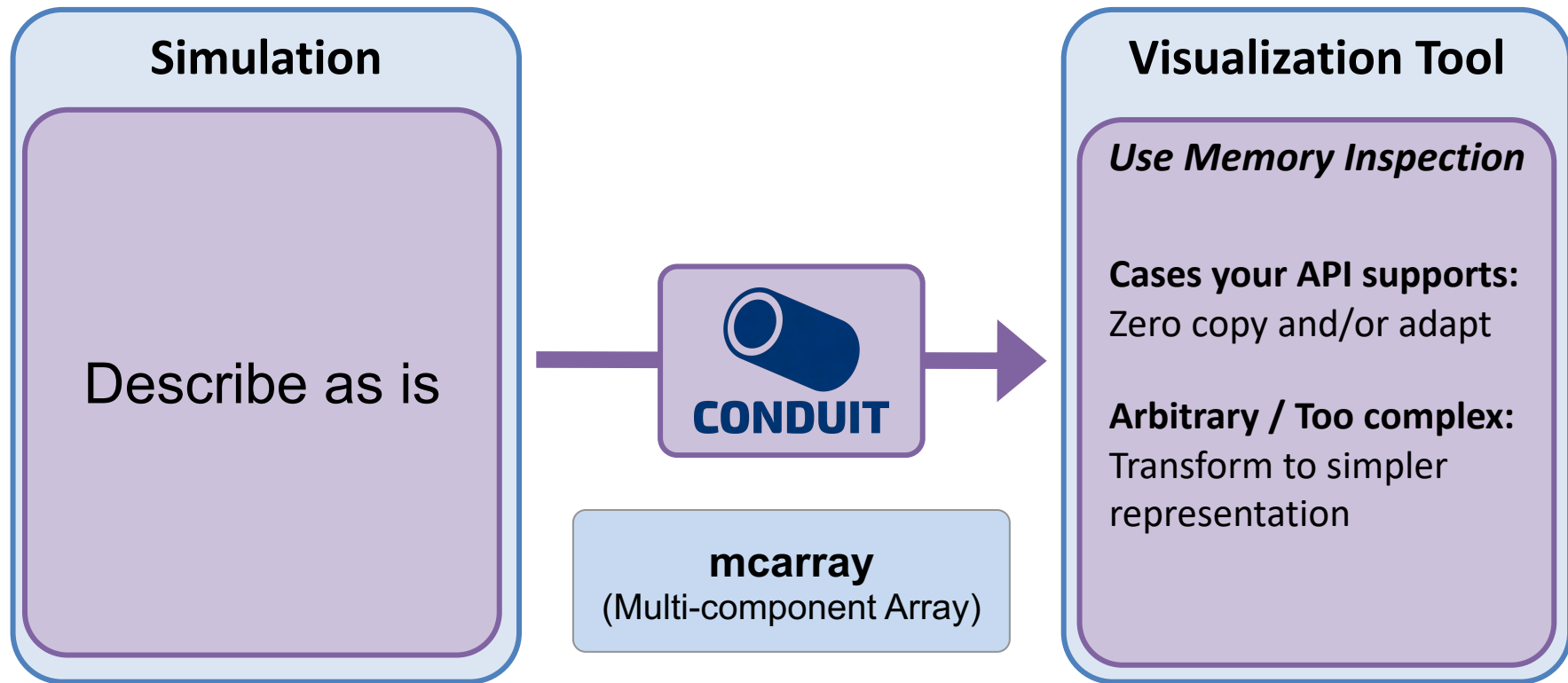
# Data description simplifies connecting software components

## Example: Sharing a Vector Field



# Data description simplifies connecting software components

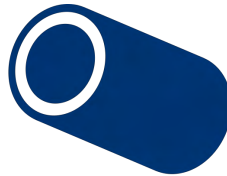
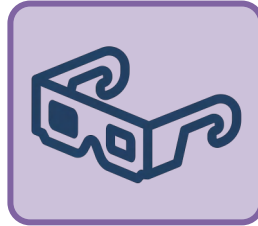
## Example: Sharing a Vector Field



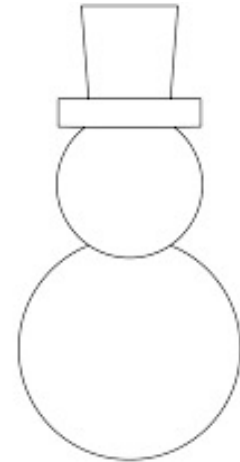
# Takeaway: Conduit makes you feel safe



**Simulation and Tool  
Data Structures in the Wild**

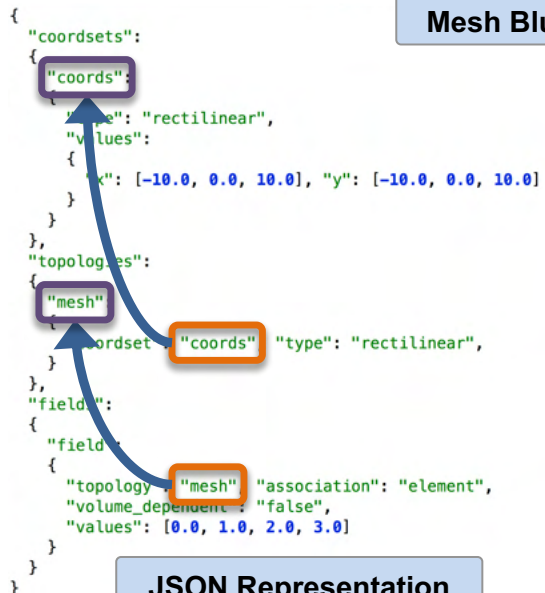


**CONDUIT**

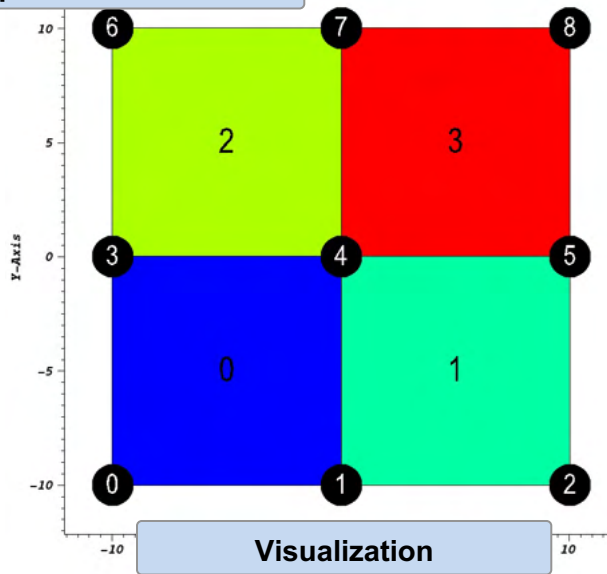


**View through  
Conduit Lens**

# The Mesh Blueprint is a set of conventions that outline a hierarchical structure (or schema) to describe mesh data

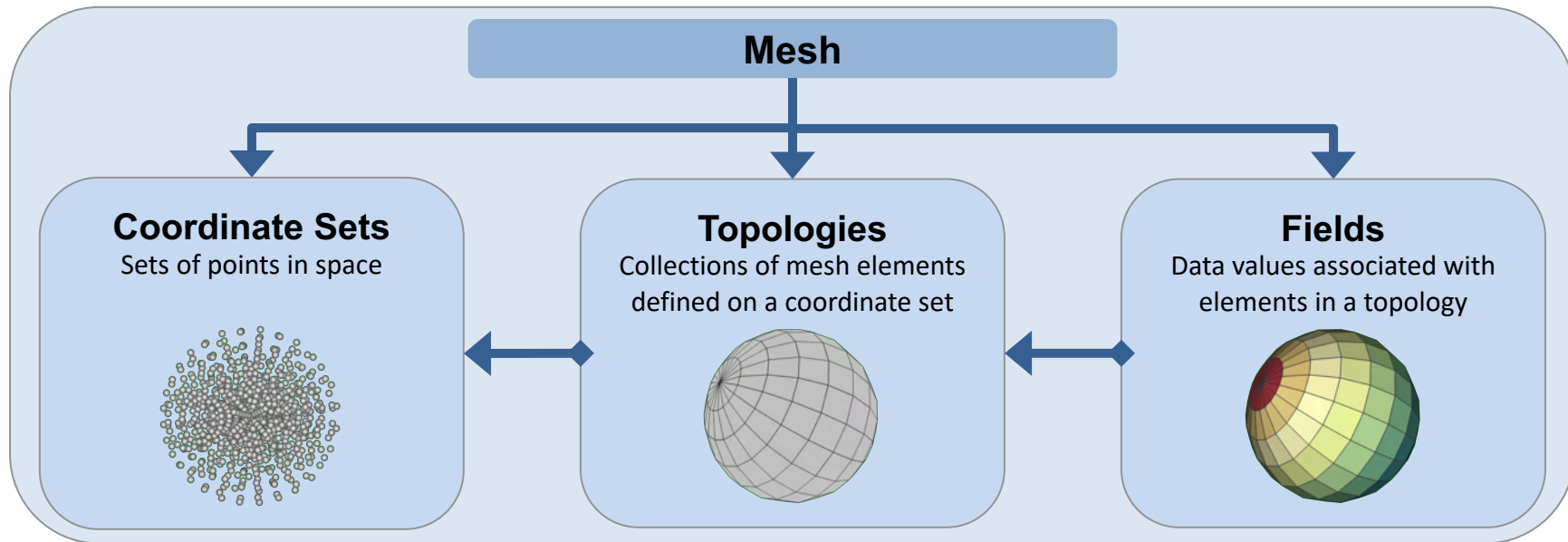


Mesh Blueprint Example: A Simple Rectilinear Grid



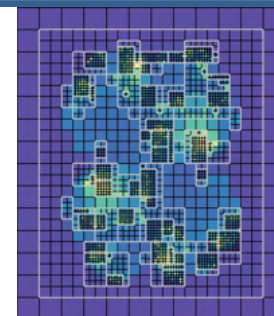
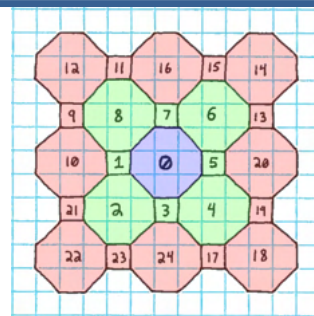
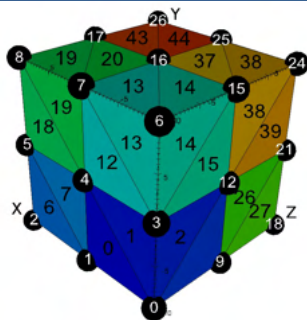
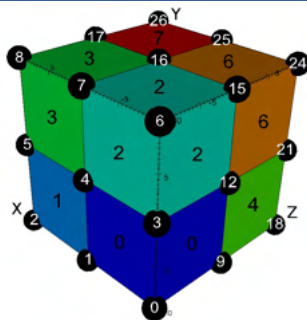
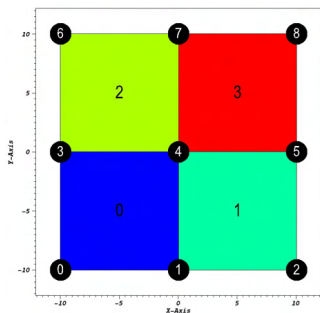
Conduit provides an in-situ data layer to minimize unnecessary data copying/movement and Blueprint provides a standard schema for communicating mesh data

# The Mesh Blueprint supports mesh constructs common in several full featured mesh data models

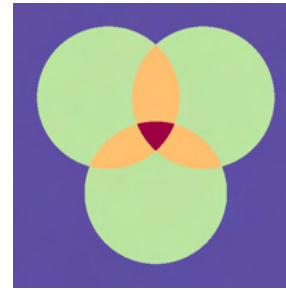
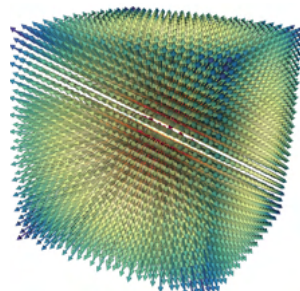
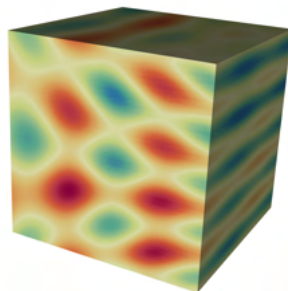


Ideas were shaped by surveying projects including: ADIOS, BoxLib, Chombo, Damaris, EAVL, Exodus, ITAPS, MFEM, SAF, SAMRAI, Silo, VisIt's AVT, VTK, VTK-m, Xdmf.

# We are steadily filling out the Blueprint to cover the wide range of mesh descriptions required by the ecosystem



*Topologies:* 1D/2D/3D - Uniform, Rectilinear, Structured, Unstructured, Polygonal, Polyhedral, AMR



*Fields:* Scalar, Vector, Multi-material

[https://lnl-conduit.readthedocs.io/en/latest/blueprint\\_mesh.html](https://lnl-conduit.readthedocs.io/en/latest/blueprint_mesh.html)

# We are steadily filling out the Blueprint to cover the wide range of mesh descriptions required by the ecosystem

## ▪ Coordinate Sets

- 1D/2D/3D
- Cartesian, Cylindrical, Spherical
- Implicit: Uniform, Rectilinear
- Explicit

## ▪ Topologies

- Implicit: Points, Uniform, Rectilinear, Structured
- Unstructured [Points, Lines, Quads, Tris, Tets, Hexs]
- Optional MFEM Grid Function support
- Arbitrary Polygonal and Polyhedral
- Unstructured heterogenous element shapes (Planned for future)

## ▪ Fields

- Vertex or Element associated
- Multi-component field arrays
- Optional MFEM Grid Function Basis support
- Sparse representations for multi-material field arrays
- Multi-dimensional field arrays (Planned for future)

## ▪ Domain Decomposition Info

- Basic State Info [Domain Ids]
- Domain Adjacency Info for Unstructured Meshes
- Domain Adjacency Info for Structured Meshes
- Nesting Info for Block-Structured AMR Meshes

# The Mesh Blueprint supports many different styles of mesh metadata and supplies transform functions therebetween

## Topologies

Structured (Rectilinear):

<b>X Coords</b>	0	1
<b>Y Coords</b>	0	1

**Topology** (implicit)

Unstructured:

<b>X Coords</b>	0	1	0	1
<b>Y Coords</b>	0	0	1	1
<b>Topology</b>	0	1	3	2

## Fields/Materials

Element Dominant:

<b>A VFs</b>	a 1	a 2	0	0
<b>B VFs</b>	0	b 1	b 2	0
<b>C VFs</b>	0	0	c 1	c 2

Material Dominant:

<b>A VFs</b>	a 1	a 2	<b>A EIDs</b>	0	1
<b>B VFs</b>	b 1	b 2	<b>B EIDS</b>	1	2
<b>C VFs</b>	c 1	c 2	<b>C EIDS</b>	2	3

## Everything In Between

Bit Width:

<b>int</b>	0x00000000 0	0xDEADBEEF F
<b>long</b>	0x00000000 0	0xDEADBEEF F

Data Ordering:

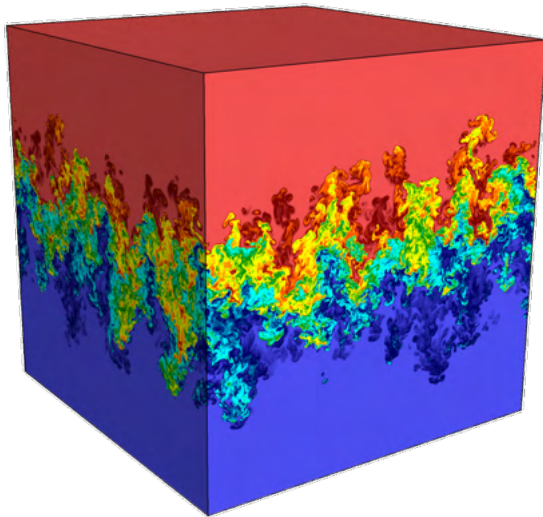
<b>Contig.</b>	x 1	...	x N	y 1	...	y N
<b>Interl.</b>	x 1	y 1	...	...	x N	y N

Data Indirection:

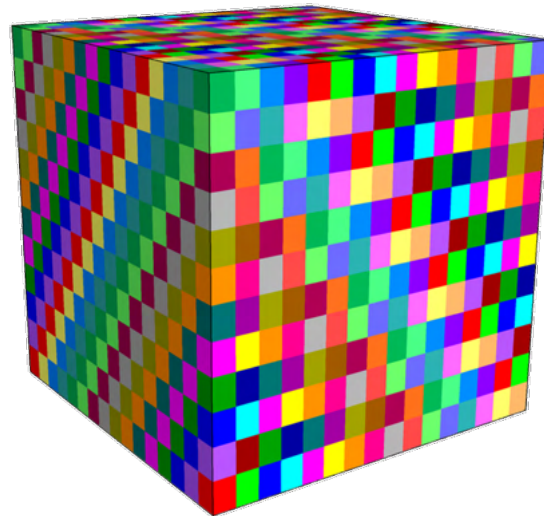
<b>sparse</b>	v N	...	v 1	...	v 2	...
<b>indices</b>	i1	i2	...	...	...	i N
<b>compact</b>	v 1	v 2	...	...	...	v N



# The structure of the Blueprint is designed with distributed-memory parallelism in mind



**Full Dataset**

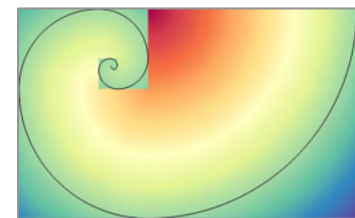
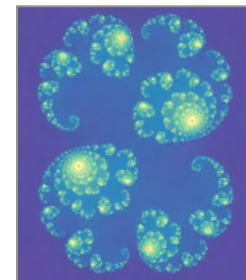
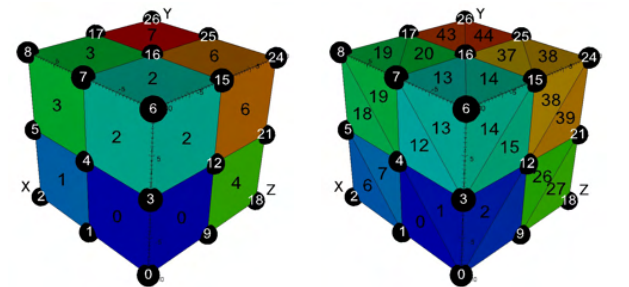


**Domain Decomposition**

Any info required to describe to domain decomposition, nesting, or abutment is local

# The Conduit Blueprint library facilitates using Mesh Blueprint data via three important capabilities

- Methods which verify if data conforms to blessed conventions at runtime
  - Provides detailed information for non-conforming data
- Methods that apply basic data transforms to conforming data, including:
  - Coordinate Set and Topology transforms (e.g. Implicit Uniform to Explicit Coordinates)
  - Memory layout transforms (e.g. Contiguous to Interleaved to array layouts)
- Methods that generate mesh examples, aiming to cover the range of supported meshes



Mesh Blueprint Examples generated by the Conduit Blueprint Library

# Demo: Creating Simple Blueprint Meshes

---

- Let's look at some examples of how to create and verify Blueprint Meshes
- <https://ascent.readthedocs.io/en/latest/Tutorial Intro Conduit Blueprint.html>
- (Demo of Ascent Tutorial Conduit Blueprint Jupyter Notebook)

# Conduit API Example:

## Our first example was almost a Blueprint Mesh ...

```
// extend our example to a blueprint compliant mesh
n["topologies/topo/type"] = "rectilinear";
n["topologies/topo/coordset"] = "coords";

n["fields/density/association"] = "element";
n["fields/density/topology"] = "topo";

n["fields/velocity/association"] = "vertex";
n["fields/velocity/topology"] = "topo";

n.print();

conduit::Node info;
if(conduit::blueprint::mesh::verify(n,info))
{
    std::cout << "Mesh Blueprint Verify Success!" << std::endl;
}
else
{
    std::cout << "Mesh Blueprint Verify Failure!" << std::endl;
    info.print();
}

conduit::relay::io::blueprint::save_mesh(n, "my_mesh_yaml", "yaml");
```

```
coordsets:
  coords:
    values:
      x: [0.0, 1.0, 2.0]
      y: [0.0, 1.0, 2.0]
    type: "rectilinear"
fields:
  density:
    values: [1.0, 1.0, 1.0, 1.0]
    units: "g/cc"
    association: "element"
    topology: "topo"
  velocity:
    values:
      u: [3.14159, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
      v: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
    units: "m/s"
    association: "vertex"
    topology: "topo"
topologies:
  topo:
    type: "rectilinear"
    coordset: "coords"
```

Mesh Blueprint Verify Success!

# Conclusion: We built Conduit to simplify data description and sharing across software components

---

- Conduit focuses on in-memory description and other use cases are built on that foundation
- Conduit and the Mesh Blueprint support LLNL WSC's strategic push to develop a modular simulation software ecosystem
- The Mesh Blueprint is being adopted as a data interface to DOE supported open source visualization tools
- Conduit Github: <https://github.com/llnl/conduit>
- Cyrus Contact Email: [cyrush@llnl.gov](mailto:cyrush@llnl.gov)

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.  
Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.



**Disclaimer**

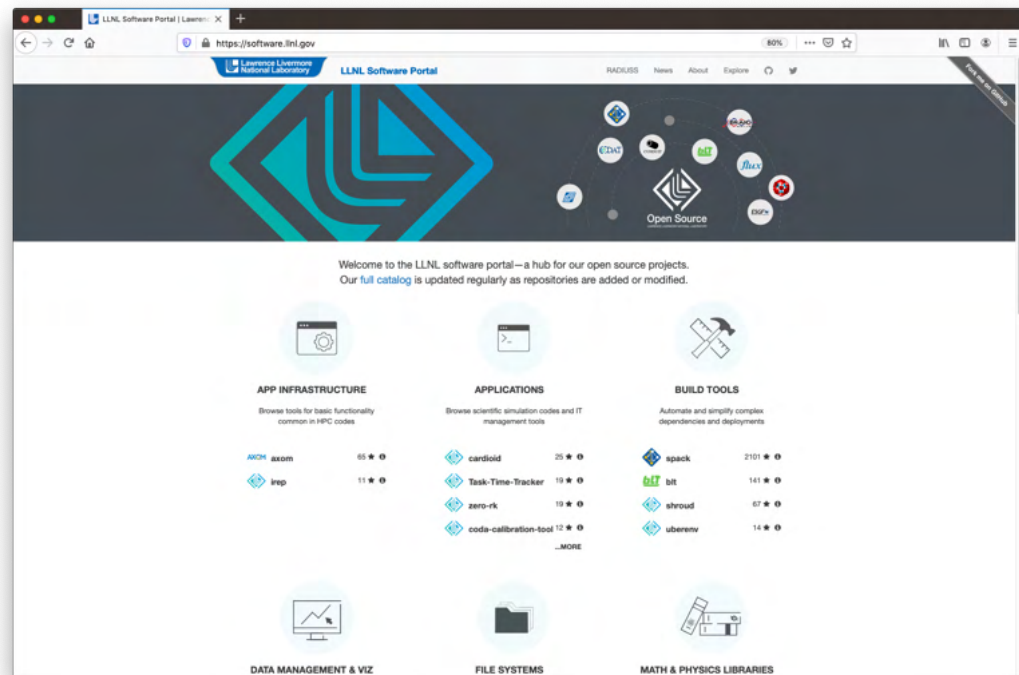
This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

---

# Backup slides

# LLNL programs actively develop and leverage open source software

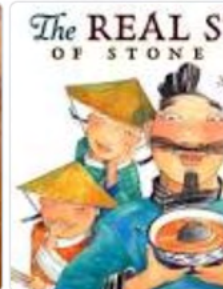
- DOE ASC(I) and ASCR have explicitly embraced and promoted open source software development since 2002
- Not all our software projects can be open sourced, but open source is a key part of our development strategy
- Open source and open development amplify: Collaboration, Adoption, and Competition



<https://software.llnl.gov>



# Creating the Mesh Blueprint followed the flavor of a Stone Soup Model



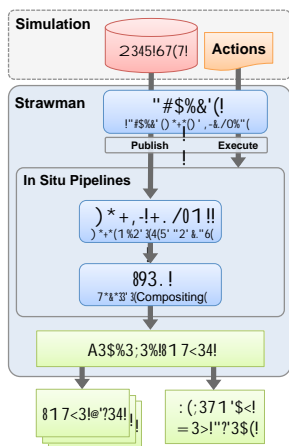
**Stone Soup** is a European folk story in which hungry strangers convince the people of a town to each share a small amount of their food in order to make a meal that everyone enjoys, and exists as a moral regarding the value of sharing.

[en.wikipedia.org](https://en.wikipedia.org/wiki/Stone_Soup) › wiki › Stone\_Soup ▼

[Stone Soup - Wikipedia](https://en.wikipedia.org/wiki/Stone_Soup)

# The Blueprint started with experiments using Conduit to describe meshes for visualization

- Summer 2015: The VisIt team expanded these ideas to create a mesh interface for a new in-situ visualization proxy called “Strawman”



!"#\$%&'%()\*"+, #&%\$-\*. \$/01#- /#2\$-

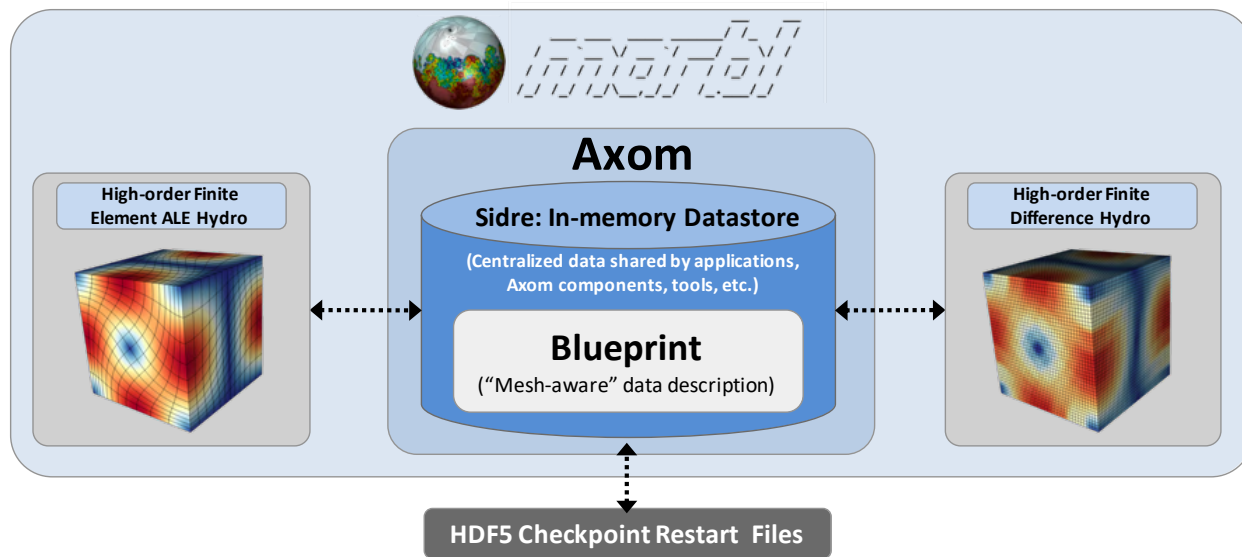
3(451#2\*\$- (6-\$\*+, \*78+9-\$8-%, : ; \*6%#%\*

3(5<1\$%#1+( \*, +\$\*=1>\*?\*\$+@A\* . <<

!"#\$%&' %()\*+\*, %#-.\*/('0#1\*2031%405%#06(\*% (7\*+ (%48303\*/(9\$%3#\$1-#1\$: \*96\$\*; 14#0<=. 830-3\*\*'0' 14%#06(\*>67: 3?  
/( \*=\$6-: : 70(@3\*69\*/"+2\*ABCD"E">CDEF6\$H3. 6IJ\*+13#0(\*KLM6N: ' 0: \$\*ABCD

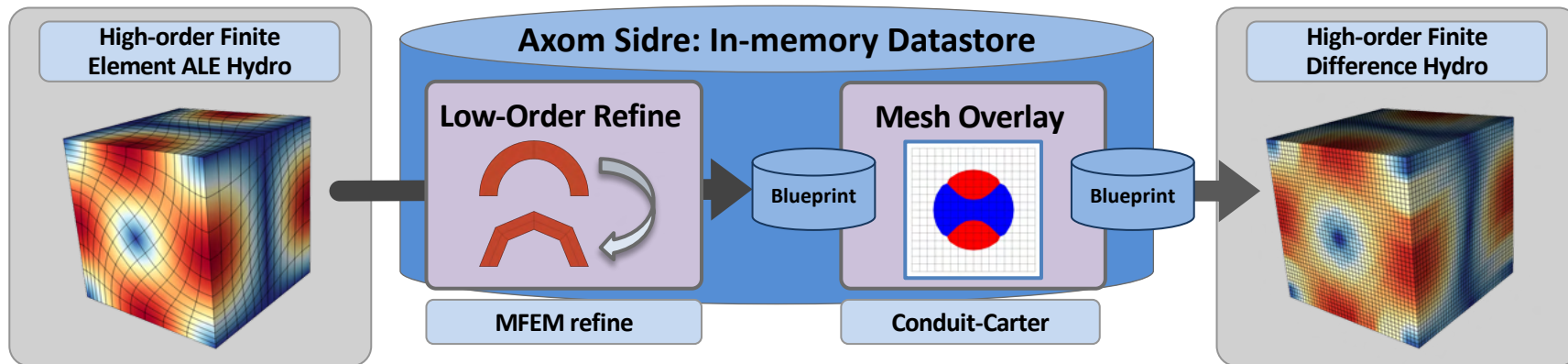
# Success with visualization use cases helped demonstrate that the Blueprint was viable for use directly in simulations

- Fall 2015 – Fall 2016: The Axom and MARBL teams adopted and helped expand Blueprint to describe meshes for checkpoint restarts supporting both of MARBL's hydrodynamics packages (Supported an ATDM FY16 L2)



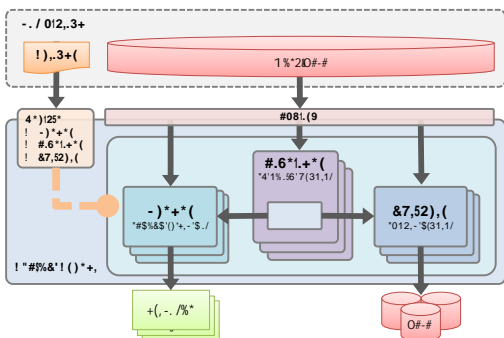
# Success with visualization use cases helped demonstrate that the Blueprint was viable for use directly in simulations

- Fall 2016 – Fall 2017: A Blueprint interface for the Carter mesh overlay tool was developed, which enabled in-situ mesh overlay from MARBL's high-order ALE hydrodynamics package to MARBL's high-order finite difference hydrodynamics package (Supported an ATDM FY17 L2)



# Adoption of the Blueprint in MARBL bolstered the case for using Blueprint in the ALPINE ECP project

- Fall 2017 – Present: Blueprint is the mesh interface for Ascent, an in-situ visualization and analysis infrastructure developed as part of the ALPINE ECP project



**Ascent Software Architecture**

**In-situ render from MARBL**

**Evolved from “Strawman”**

!K. : \*+P= /MQ\* / (\*\*0#1\* / (9\$%3#\$1-#1\$:) \*+3-: (70(@\*9\$6' \*#. : \*+3. : 3\*69\*\*\*#\$\$&' %(?  
/(\*=\$6-: : 70(@\*3\*69\*/"+2\*ABCR"E">CRF\*G6\$H3. 61J\*S: (N: \$\*>TJM6N: ' O: \$\*ABCR

[.##13\)UU@0#. 1OV-6' U%410\(:<7%NU%3-: \(#  
.##13\)UU%3-: \(#V\\$: %7#: : 76-3V06U](#)