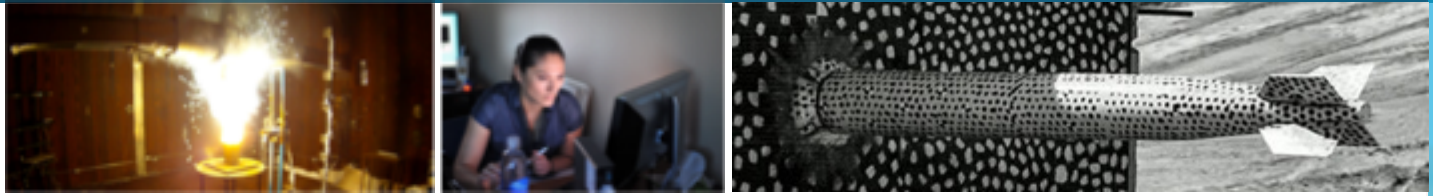
Addressing data center storage diversity in HPC applications using Faodel



Craig Ulmer, Shyamali Mukherjee, Gary Templet
Scott Levy, Jay Lofstead, ***Patrick Widener***
Todd Kordenbrock
patrick.widener@sandia.gov

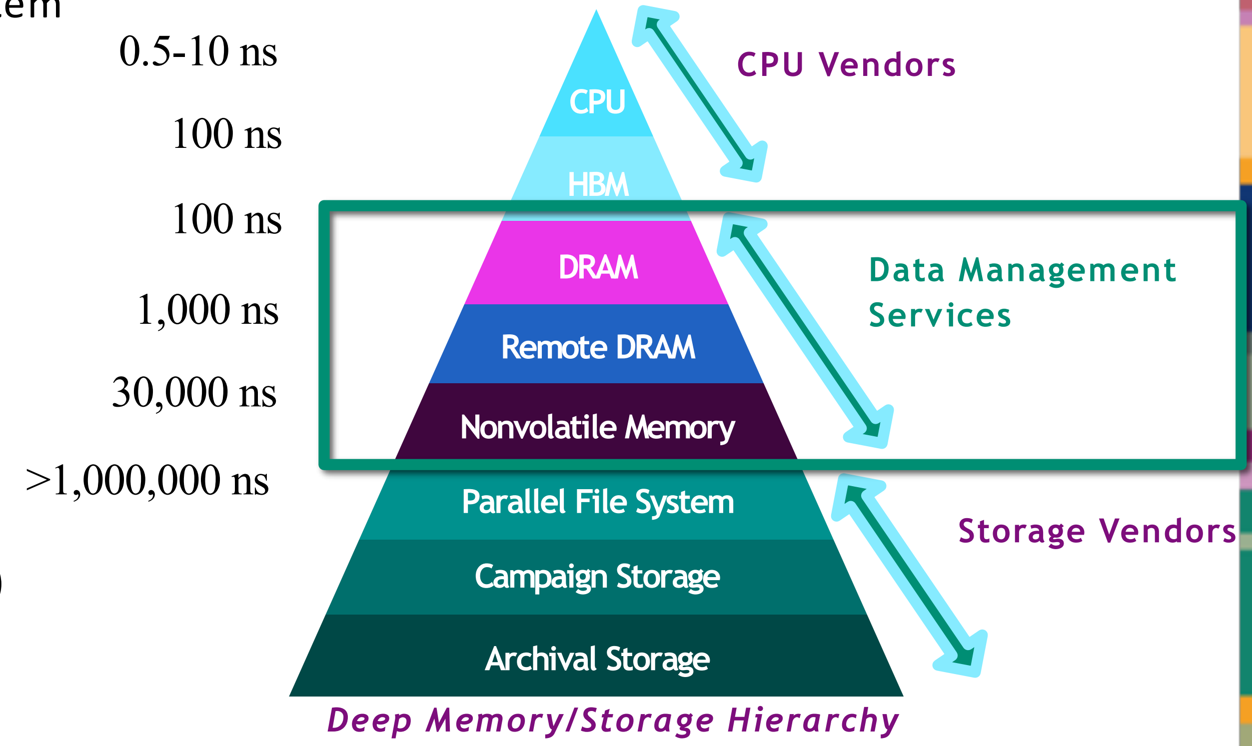
Sandia National Laboratories
Albuquerque NM / Livermore CA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND 2017-12459C, 2018-5192C, 2018-1096PE

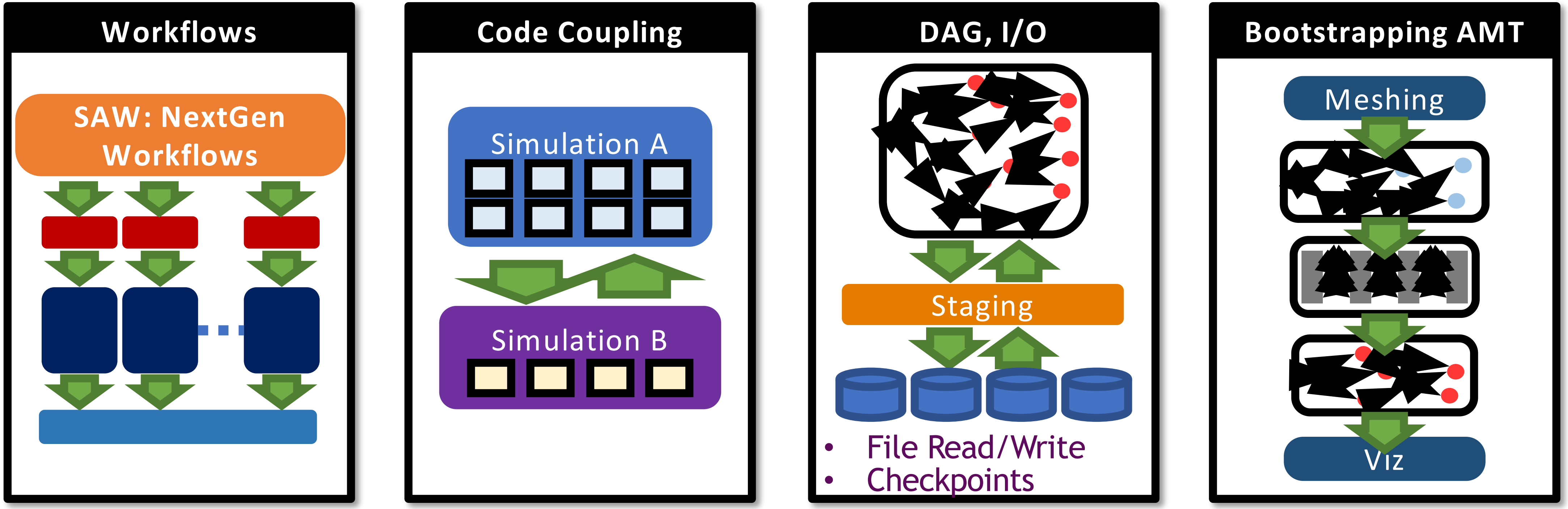
HPC applications face evolving data management needs

- Much of Sandia National Laboratories HPC is large-scale simulation of physical systems
  - Climate modeling, combustion, materials engineering, stockpile assurance
- Data center storage will be a focal point for application evolution
  - Simulate / output / analyze cycle
  - Integration point has traditionally been the storage system
  - Scale-up, scale-out on same platform
- Changes aren't permanent, but change is
  - Impedance mismatches between data capture / production vs. storage
  - Applications want flexible and resilient data storage, but want complexity hidden
  - Storage hierarchies growing deeper and more complex
  - Barriers to integration with analytics / viz / other downstream processing (file formats, storage locations)
  - Support for workflows and portable analytics (potentially on same platform)



Need for Data Management Services at Exascale

Traditional HPC *Asynchronous Many-Task*



= **Data Management Service**

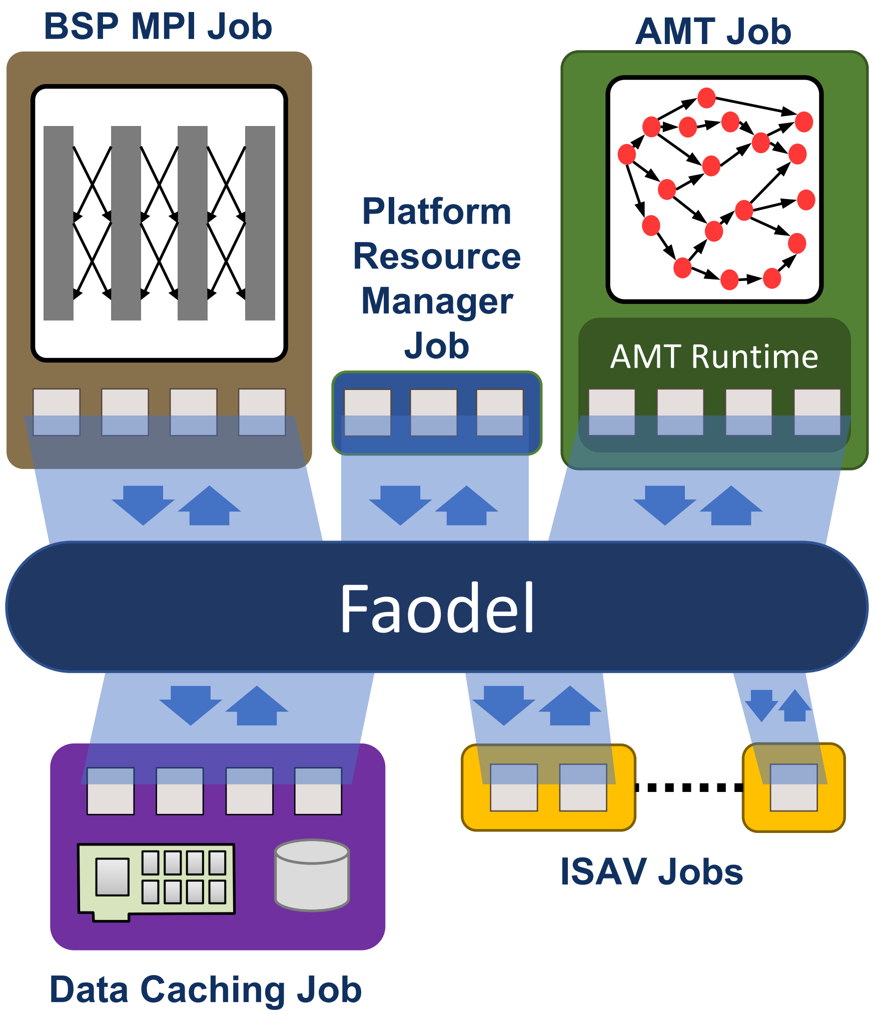
Currently we lack a single good way to implement these capabilities

What should a data management service look like?

- Requirements
  - Job-to-Job Communication
  - Coexist with MPI and AMT
  - Asynchronous and Event-Driven
  - Support Sandia's APIs and Platforms
  - Modern C++ primitives (lambdas, futures)

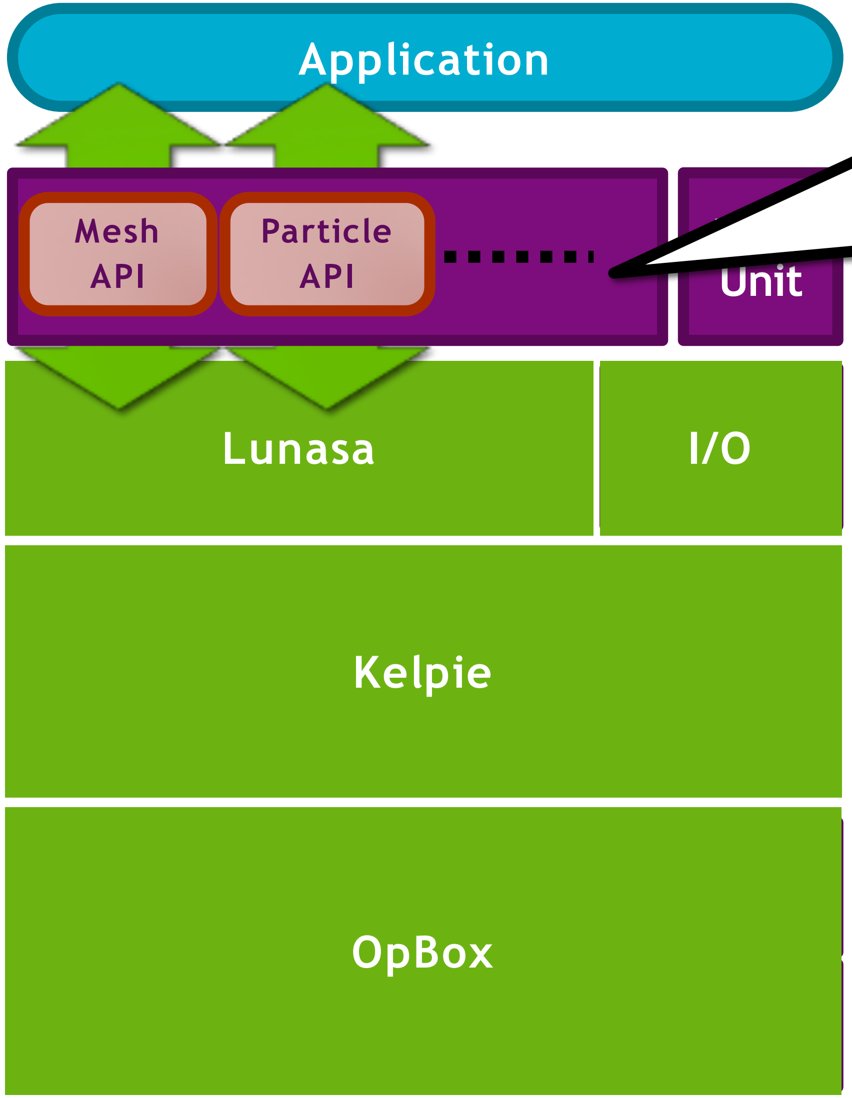
Existing software for data management services?

Domain	Examples	Issues
AMT Frameworks	Charm++, Legion, Uintah	Lack job-to-job Framework lock-in
RDMA Libraries	GASnet, Mercury, Nessie, libfabric, UCX, Converse…	Too low-level Only target Client/Server
Code Coupling	DataSpaces	Focused on staging Good, but need more capabilities



Faodel Architecture

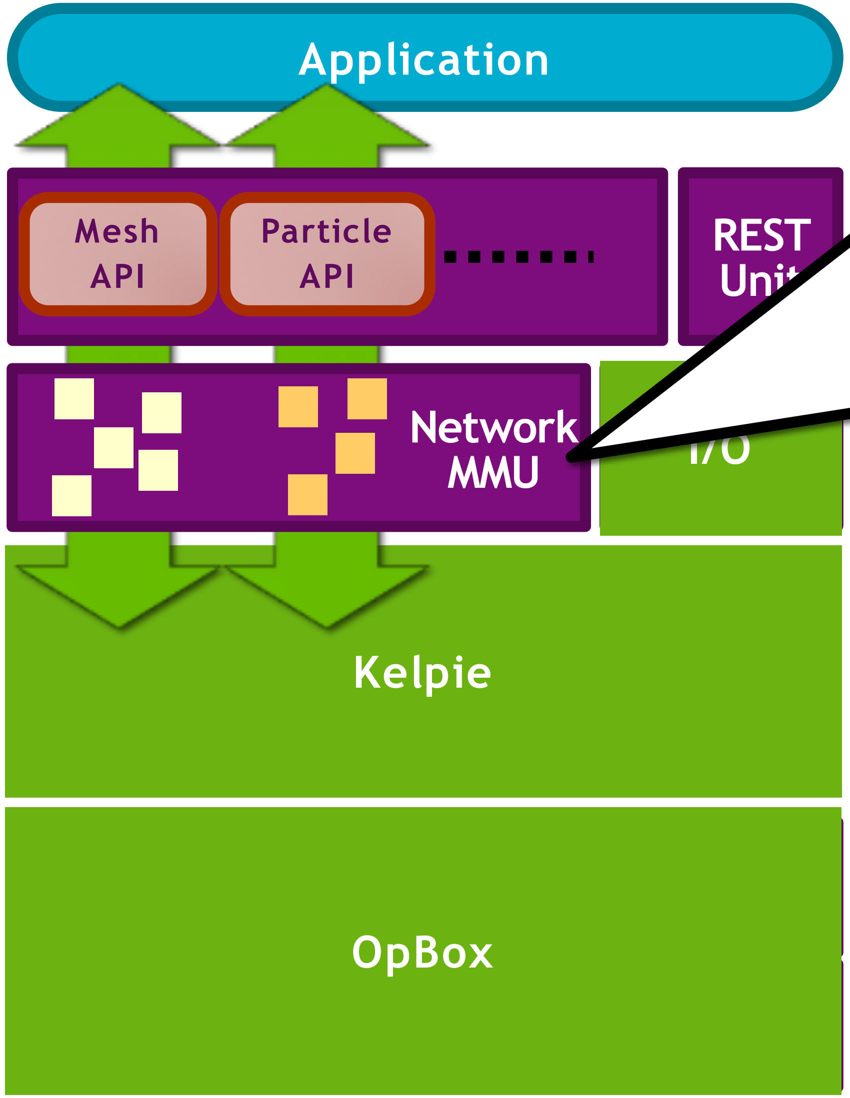
Faodel Component Structure



Data Interface Modules (DIMs)

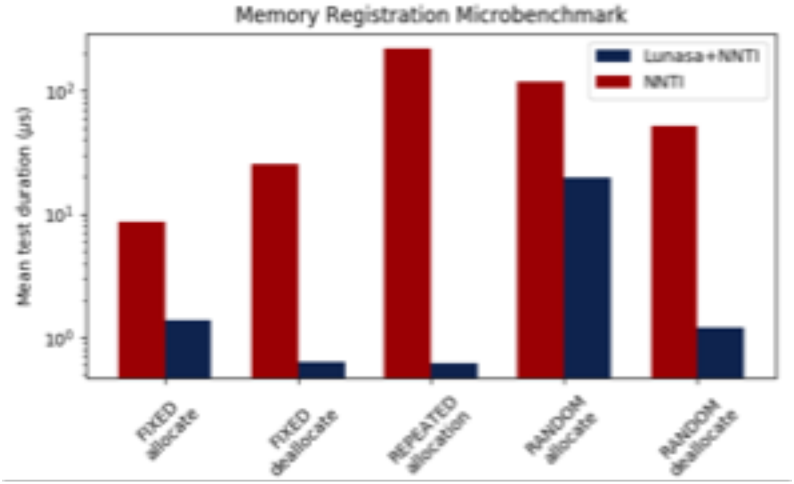
- No single API for all datasets
- Develop new modules for each dataset
- Top: Implement familiar user API
- Bottom: Faodel calls

Faodel Component Structure

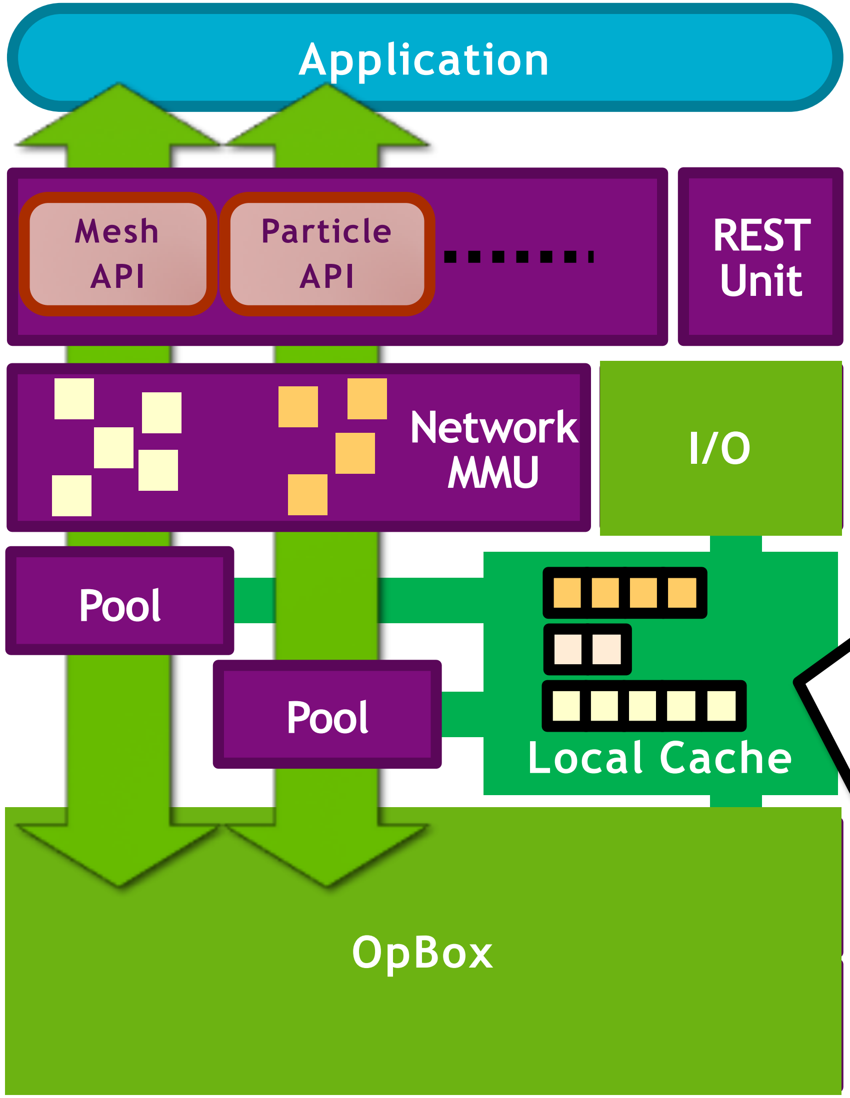


Lunasa: Network Memory Management

- Network memory requires *registration*
- Registration can be expensive
- Suballocate memory with tcmalloc

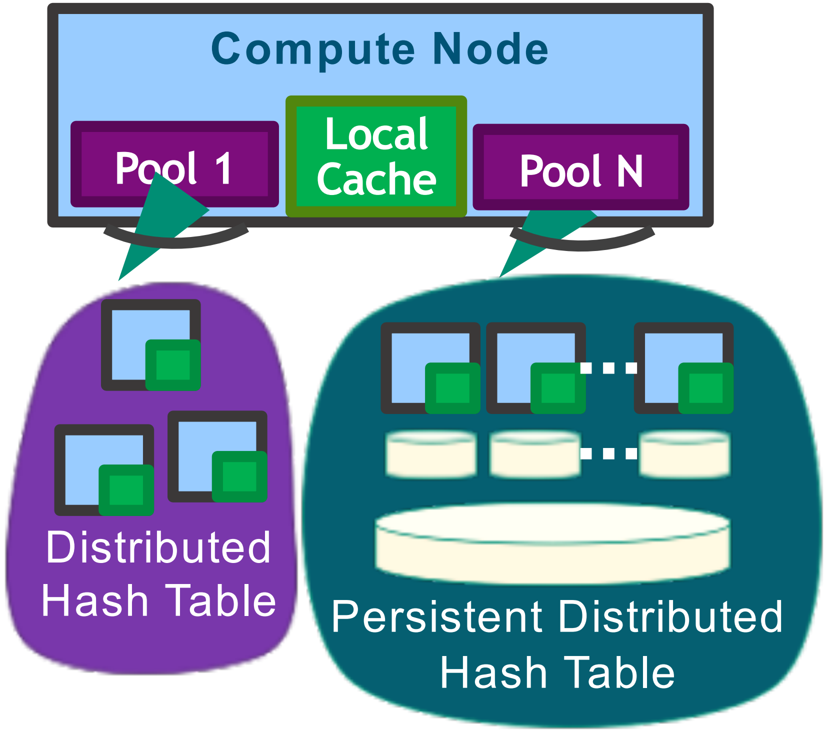


Faodel Component Structure

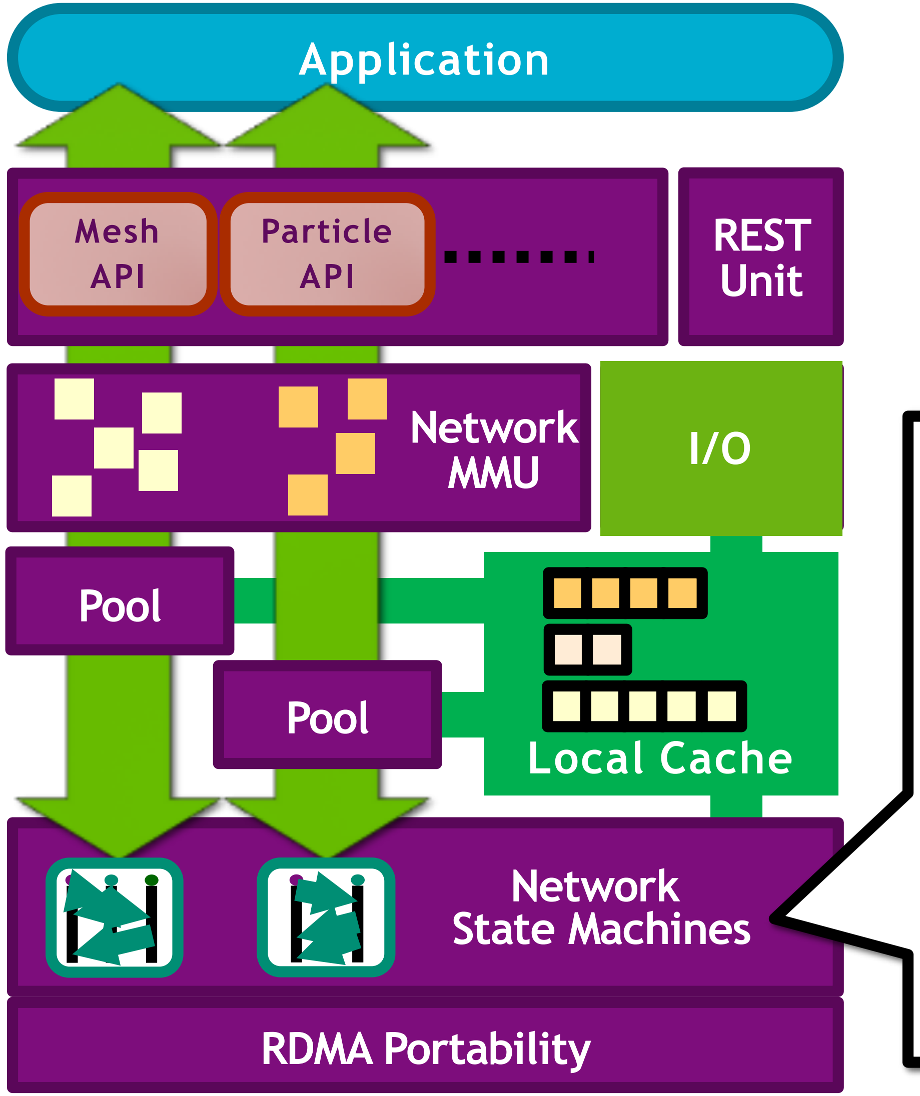


Kelpie: Distributed Key/Blob Service

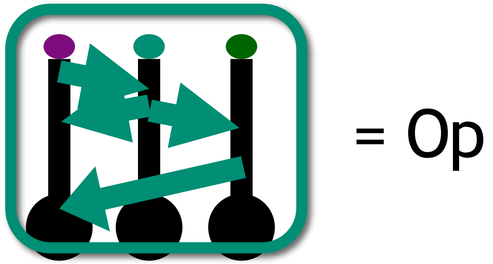
- User-controlled *Local Cache*
- Leave callbacks for objects
- “Pool” controls object distribution



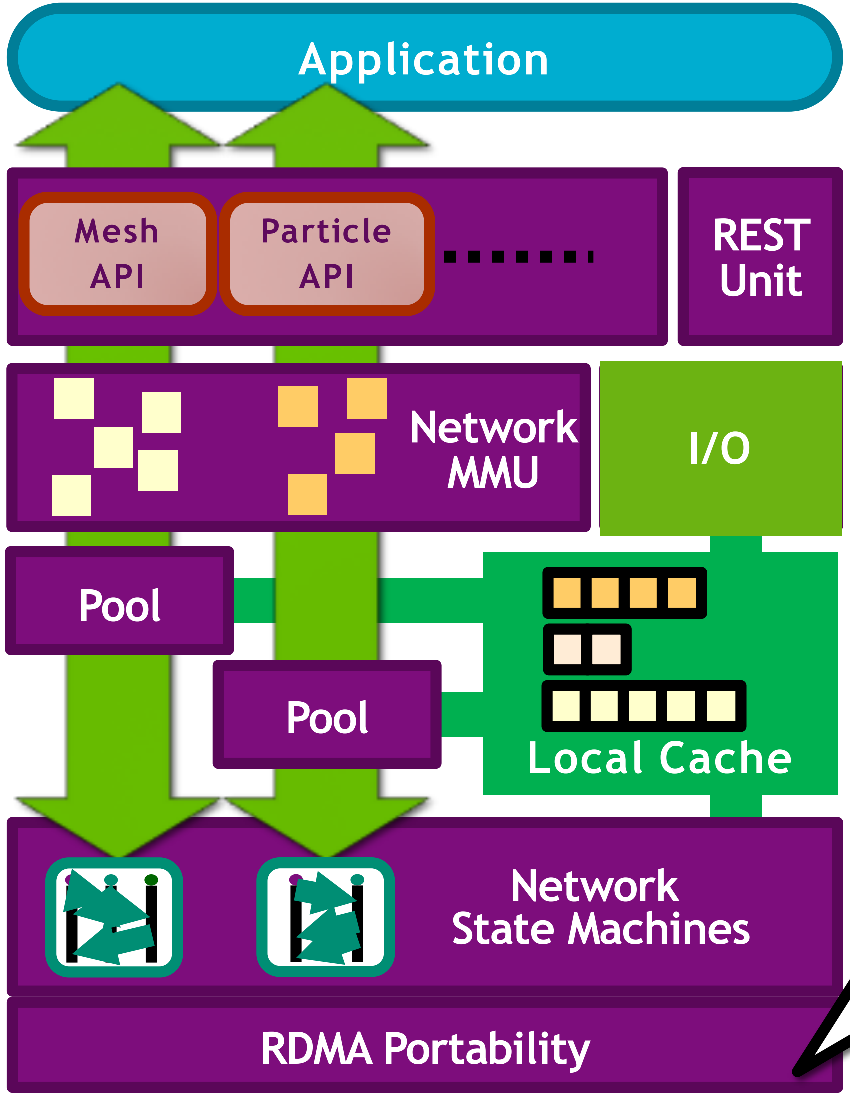
Faodel Component Structure



OpBox: Network State Machines
- RPCs insufficient
- Implement transfers in *state machines*
- More clarity, better error handling
- OpBox manages progress via Ops

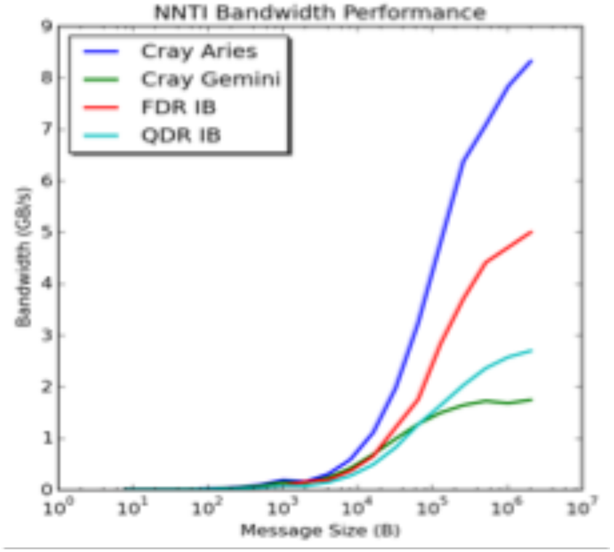


Faodel Component Structure

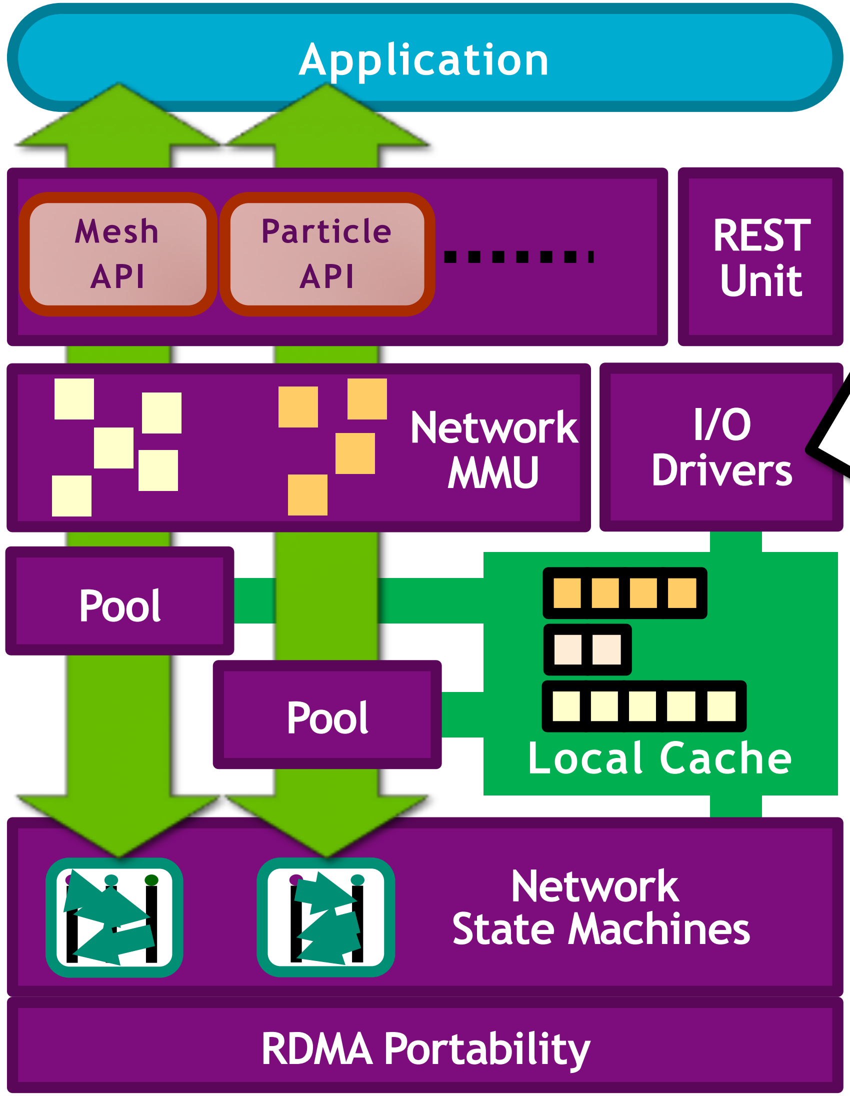


RDMA Portability

- Low-level network transfers
- Support NNTI or libfabric

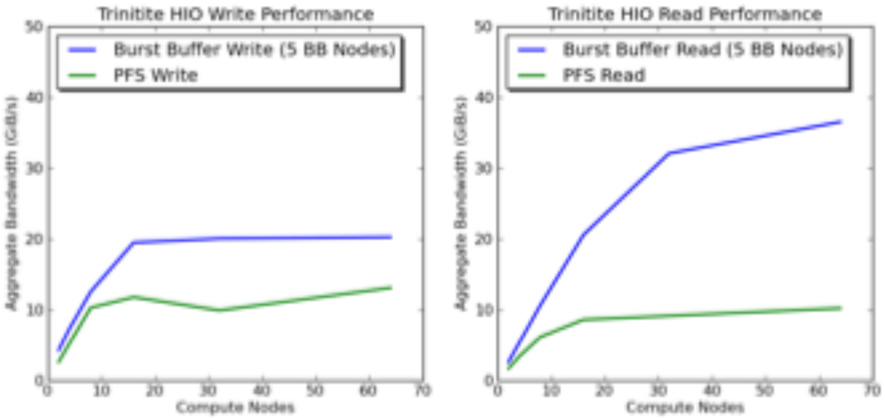


Faodel Component Structure



I/O Drivers

- Interface to Burst Buffers, NVMe, PFS
- Currently use libHIO from LANL
- Support for XC40 DataWarp and PFS



Faodel Use Cases

Faodel Use Case: Kelpie Producer / Consumer Example

```
void
produce( const size_t ds, const size_t item_count )
{
  dht = kelpie::Connect( url );

  for( const size_t i = 0; i < item_count; i++ ) {

    kelpie::Key k;

    k.K1( std::to_string( mpi_rank ) );
    k.K2( std::to_string( i ) );

    lunasa::DataObject ldo ( 0, ds );

    dht.Publish( k, ldo );
   }
}
```

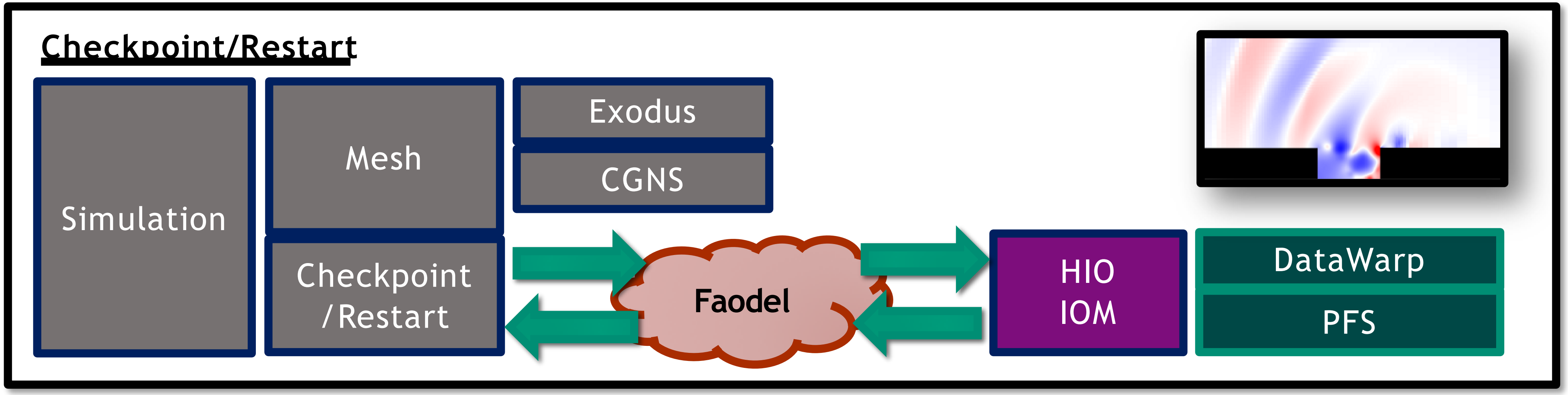
```
void
consume( const size_t ds, const size_t item_count )
{
  dht = kelpie::Connect( url );

  for( const size_t j = 0; j < item_count; j++ ) {

     kelpie::Key k;

     k.K1( std::to_string( mpi_rank ) );
     k.K2( std::to_string( j ) );

     lunasa::DataObject ldo1;

     dht.Need( k, &ldo1 );
    }
}
```

URL-based naming scheme for resource groups (for example, processes implementing a DHT)

Fine-grain control over keys and therefore hashing performance

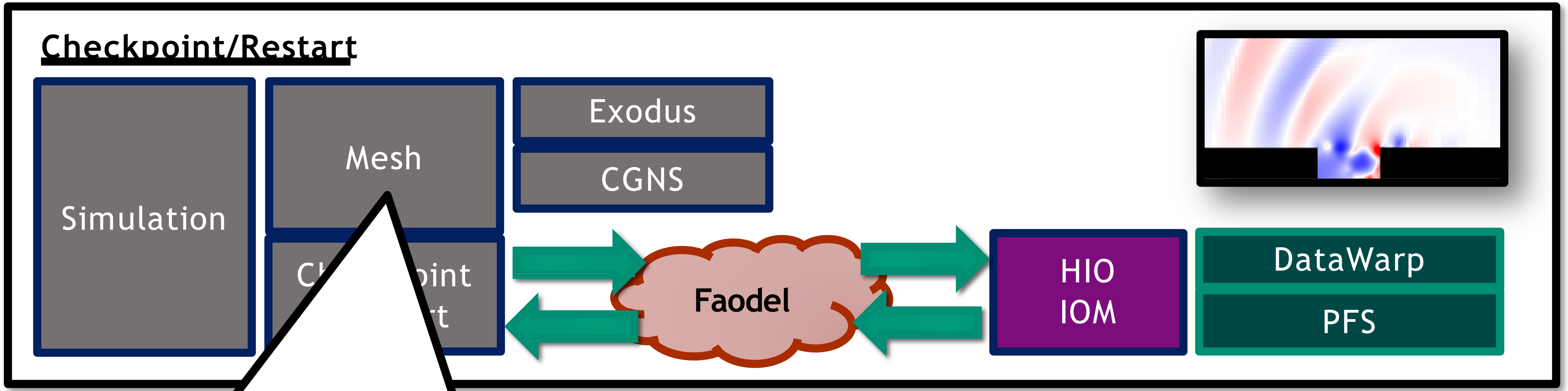
Event-based API
Publish, Want, Need

Faodel Use Case: I/O Modules for Checkpoint – Restart



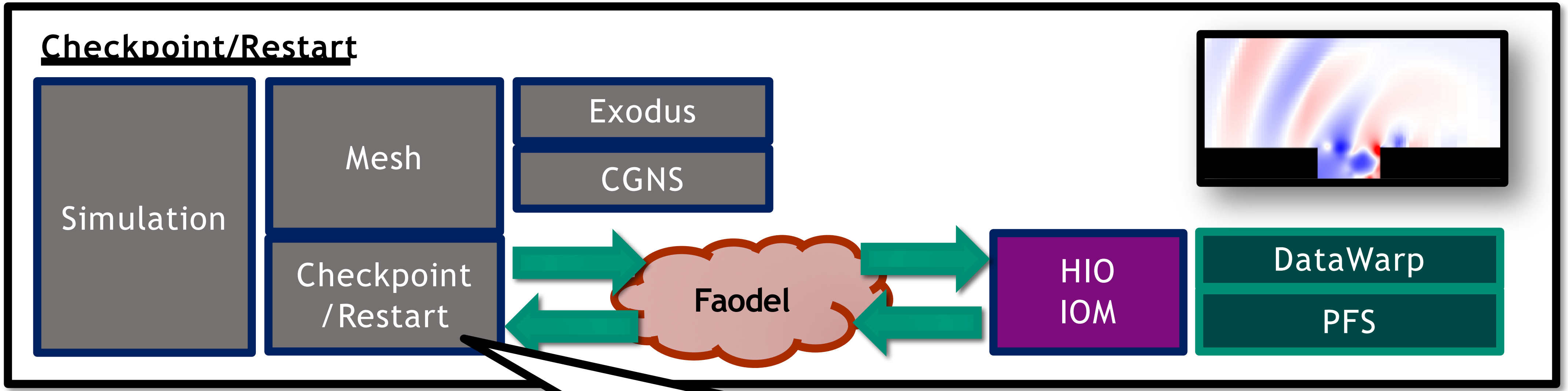
- Adding checkpoint/restart capabilities to an existing aerosciences CFD simulation code
  - Inputs are structured and unstructured meshes
- Primary restart use case is to ”bridge” long-running problems across job allocations

Faodel Use Case: I/O Modules for Checkpoint – Restart



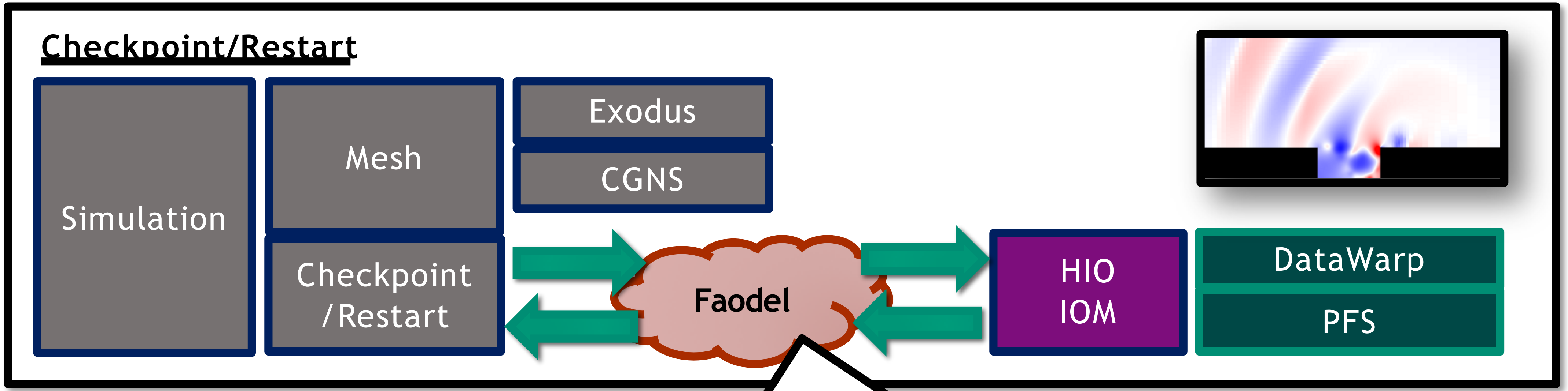
- Mesh description handled by existing file / container formats
- Exodus (NetCDF) and CGNS historically popular
  - Tied to file system
  - Complicated API, interdependent metadata updates during I/O
- Frequently the mesh structure doesn't represent the problem (which is what needs to be memo-ized)

Faodel Use Case: I/O Modules for Checkpoint – Restart



- *Solution state* is what must be checkpointed
- Often makes sense to represent independently of mesh
  - Significant space savings possible
  - Organize representation for specific cases – restart, viz, analysis
- Many times only 1 or 2 checkpoints are necessary
  - … as opposed to writing all to a filesystem-hosted library

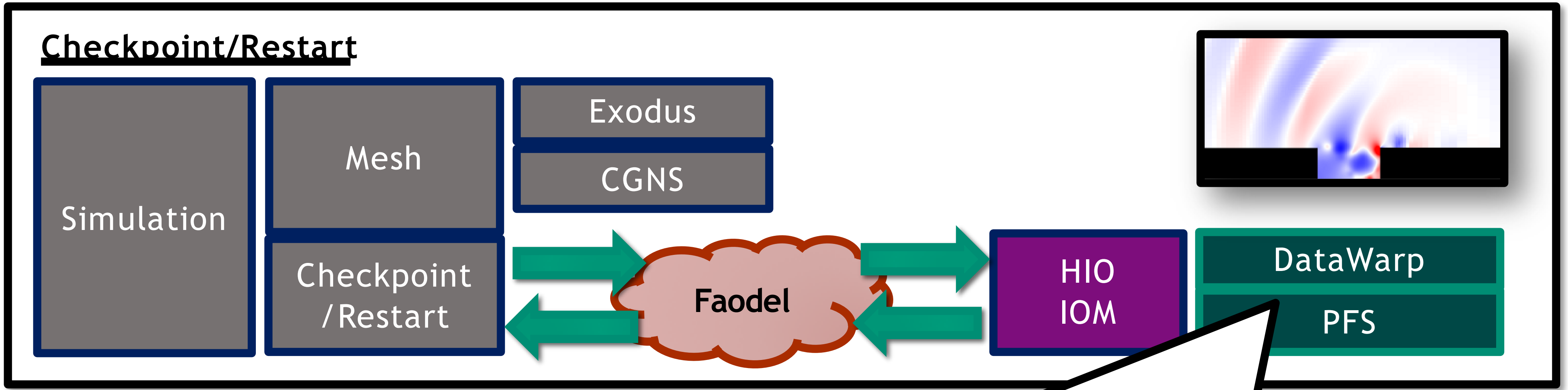
Faodel Use Case: I/O Modules for Checkpoint – Restart



- Simulation chooses a set of keys to represent desired semantics
  - Sometimes just arrays of state variables
- Values stored in LDOs allocated through Lunasa
- Kelpie stores LDOs in desired pool structure (e.g. DHT)
- LDO contents (the checkpoint) distributed among DHT nodes

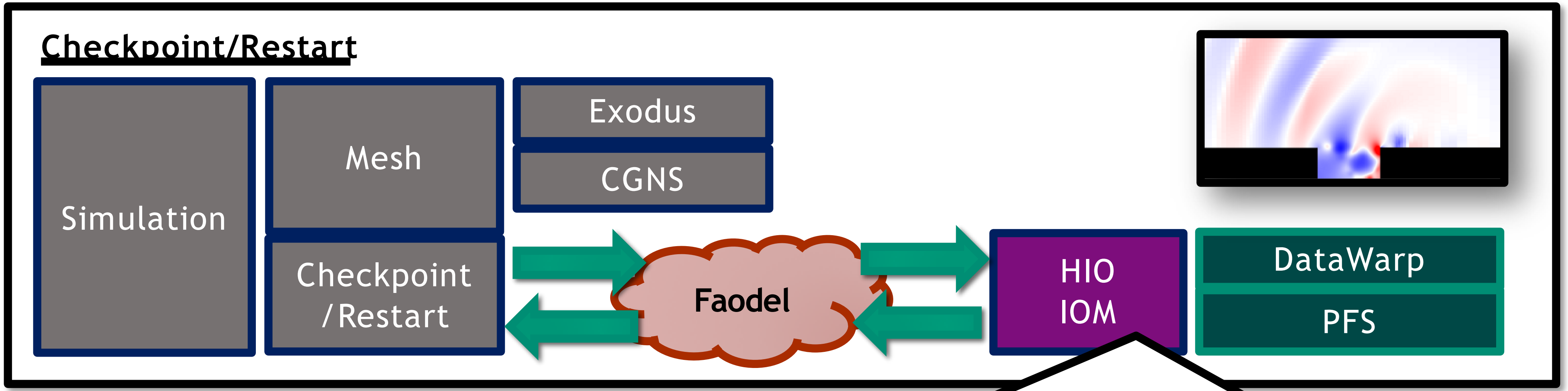
Checkpoint contents have to end up on stable storage eventually

Faodel Use Case: I/O Modules for Checkpoint – Restart



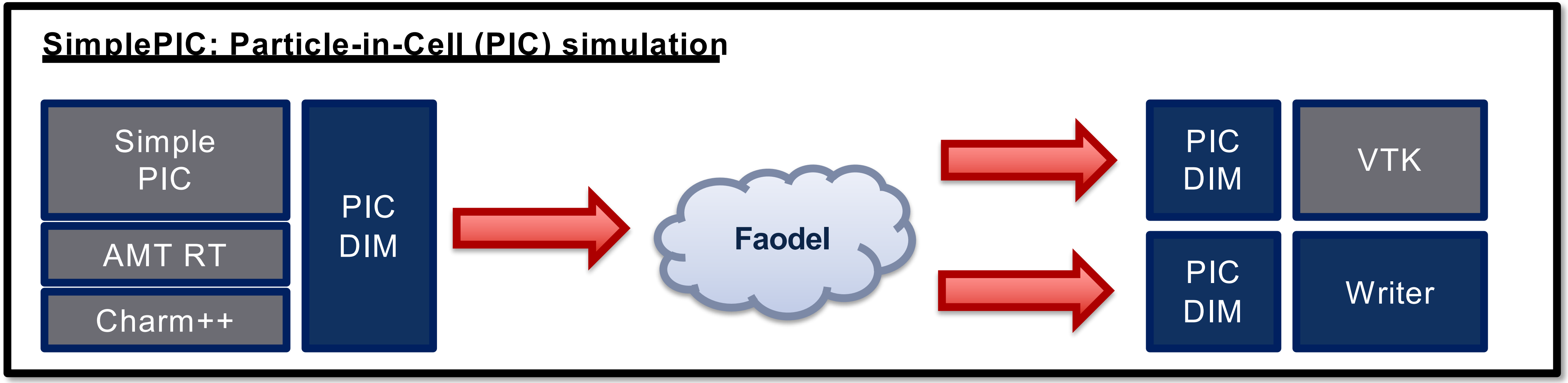
- Application developers would like to use "burst-buffer" storage
  - Fast I/O for checkpoint
  - Background "trickle" to PFS
  - Potentially, preferentially retain some data at burst-buffer
- Targets are new systems which will have some type of near-line fast storage
- But they do *not* want to manage this process themselves if they don't have to

Faodel Use Case: I/O Modules for Checkpoint – Restart



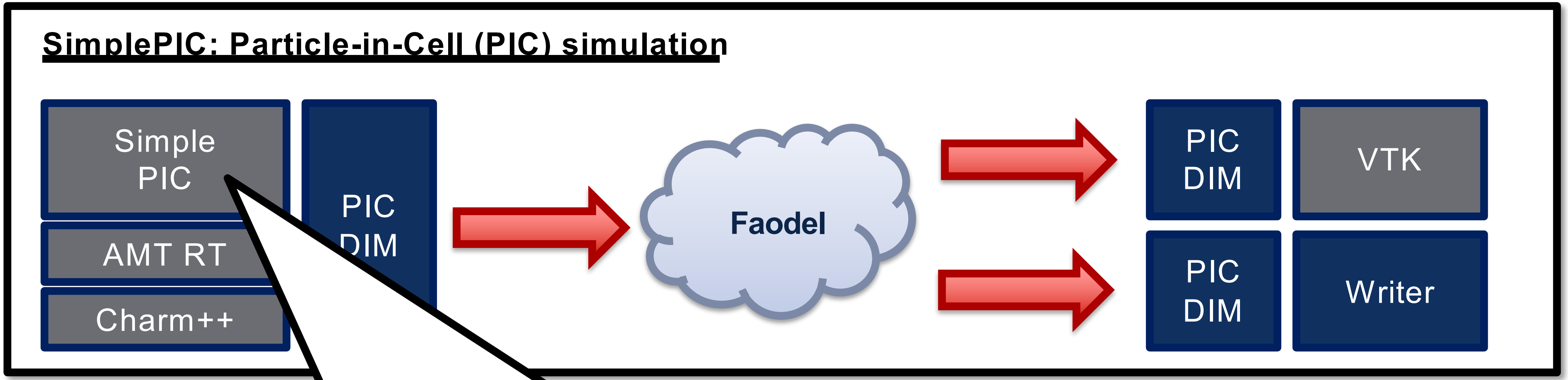
- The role of the I/O Module
  - Mediate between K/V structure and stable storage APIs
  - Still need explicit interaction with job scheduler
- At intervals:
  - Faodel supplies a set of keys to the IOM attached to each DHT node to be persisted
  - IOM writes to stable storage as configured
- HIO library can write to either DataWarp (Cray burst-buffer) or PFS
- Also have an IOM that writes directly to DataWarp
- Performance is mixed - we continue to investigate causes

Faodel Use Case: Data Interface Modules for Data Access Diversity



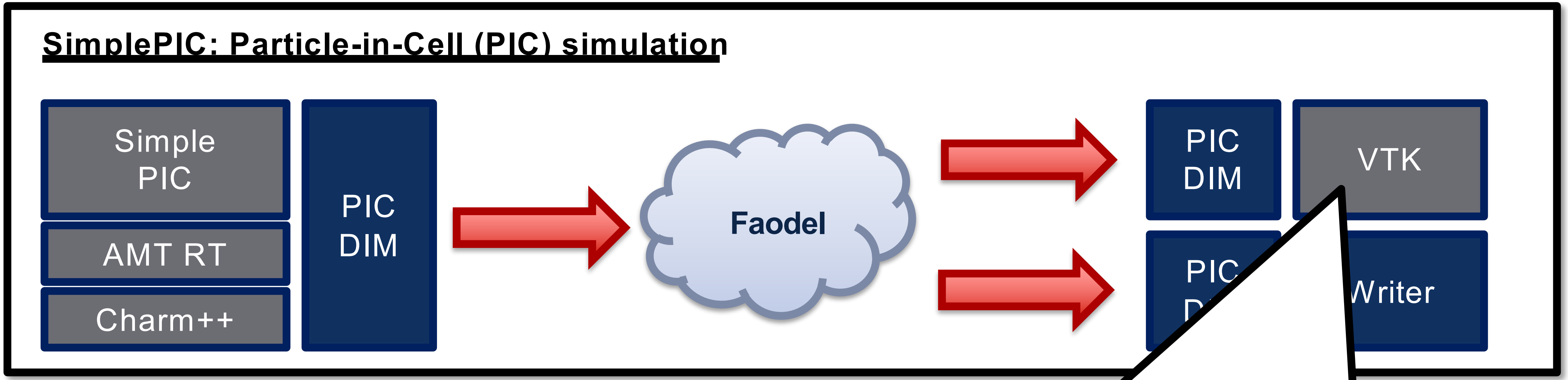
- PIC methods simulate EM fields using high-fidelity meshes, tracing particles as they migrate
- Particle motion causes imbalance in the mesh distribution across compute nodes
- SimplePIC: asynchronous many-task reference implementation to explore load-balancing tradeoffs
- More particles → wider range of testing possibilities

Faodel Use Case:
Data Interface Modules for Data Access Diversity



- Current simulations track 1 billion particles on 30M element mesh
- Data sizes
  - 8 features per particle (~128 bytes)
  - ~128 GB per timestep
  - Normal timesteps perform sampling to minimize output
  - Full checkpoints include all data + some extra features

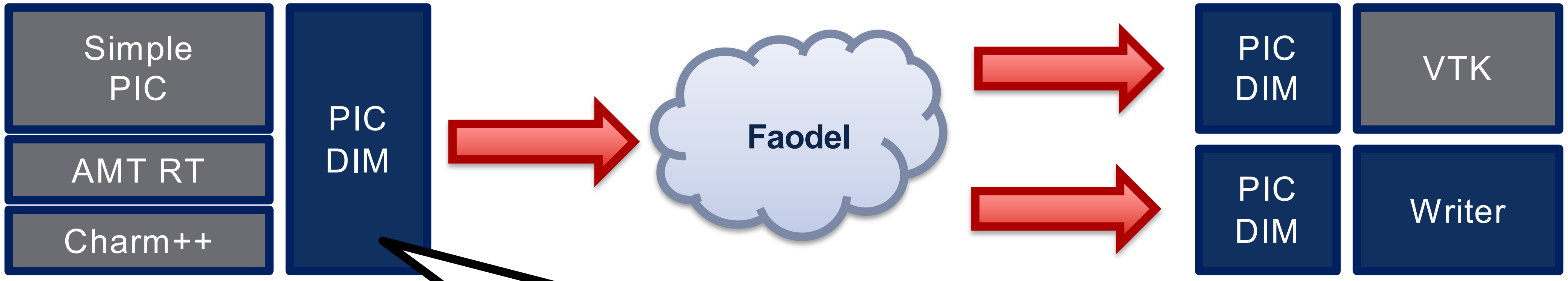
Faodel Use Case: Data Interface Modules for Data Access Diversity

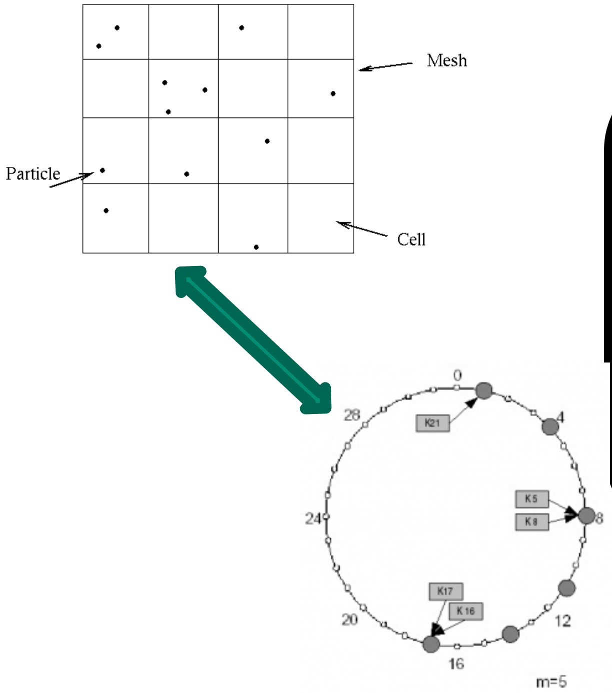


- *In situ* analysis / visualization tools are necessary
  - Summarize simulation conditions in ways meaningful to users
  - Load balancing means tools need particle-access mechanism
- Typical ISAV tasks
  - # of particles “close” to regions of interest
  - # of particles exceeding threshold velocity
  - Image rendering to monitor simulation modeling
- These actions aren’t well-supported by on-disk storage formats
  - Hard to “split” common formats to take advantage of memory hierarchy
  - Intrusive coding required for viz tasks

Faodel Use Case:
Data Interface Modules for Data Access Diversity

SimplePIC: Particle-in-Cell (PIC) simulation





Faodel Particle Data Interface Module

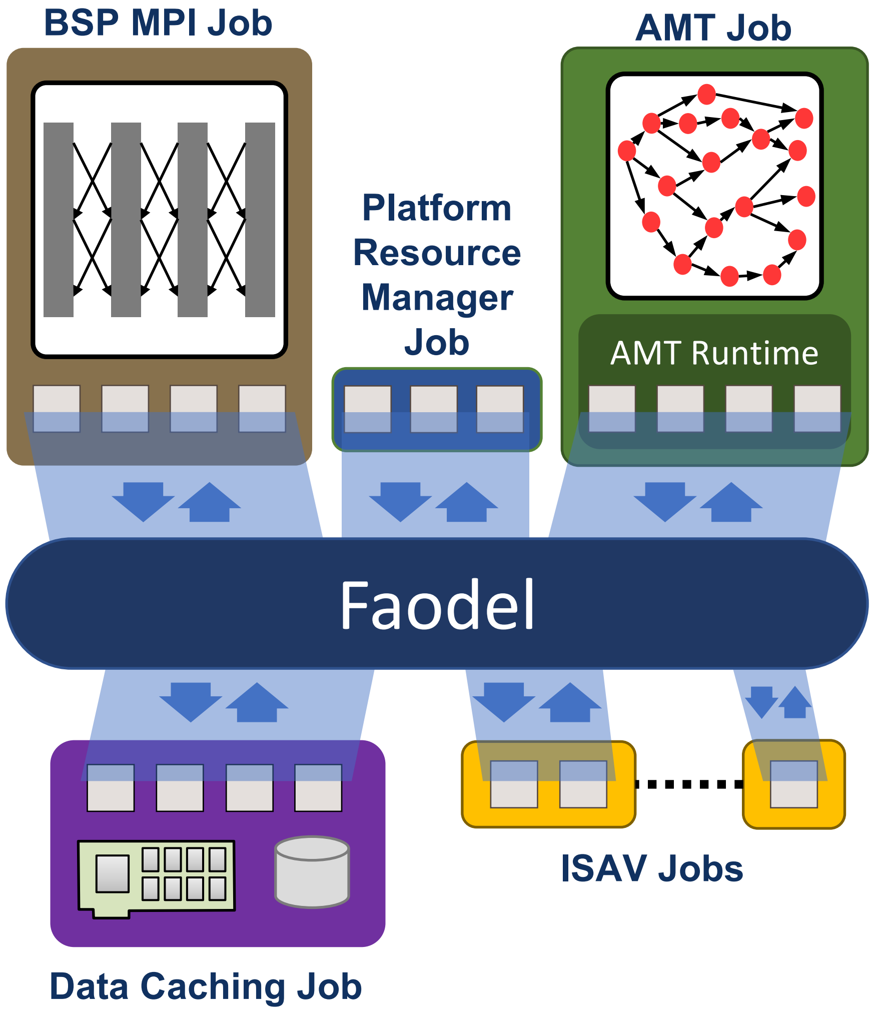
- Faodel & Particle DIM serve both producers and consumers of PIC data
- A DIM is a data-exchange contract: what data is exchanged, not how (multiple DIMs possible)
- Translates app data semantics into Faodel K-V pairs
- Migration, indexing and query support appropriate to the application and available storage capabilities

Faodel Future Use Case: Cooperating with Kokkos on Memory Management

- Kokkos is a Sandia open-source library providing programming abstractions which support *performance portability*
  - Integrated mapping of thread parallel computations and N-d array data onto manycore architectures
- Many Sandia applications have adopted Kokkos containers ("View")
- Faodel manages user memory for network transfer using Lunasa LDOs
- Can we provide an expressive, performant way to map from View ←→ LDO?
  - Relatively complex integration with Kokkos memory management
- If successful, potential for reducing I/O cost in AMT applications that rely on accelerators
  - Also may be able to inform Kokkos on-node data layouts via Kelpie-hosted metadata

Conclusion

- Faodel provides data management tools & services for computational science applications
- Faodel is a promising integration point for managing data in complex storage hierarchies
  - … while providing applications with abstractions
- Our group is currently working on additional use cases for evaluation purposes
  - We care about performance and scalability
  - We care more about uptake among users
- An alpha public release of Faodel is available: `https://github.com/faodel/faodel`



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

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 initiative.