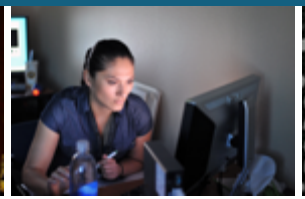
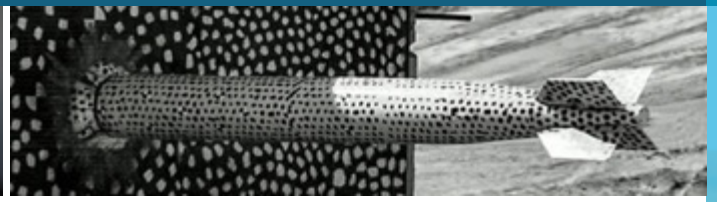
Mediating data center storage diversity for HPC applications with FAODEL





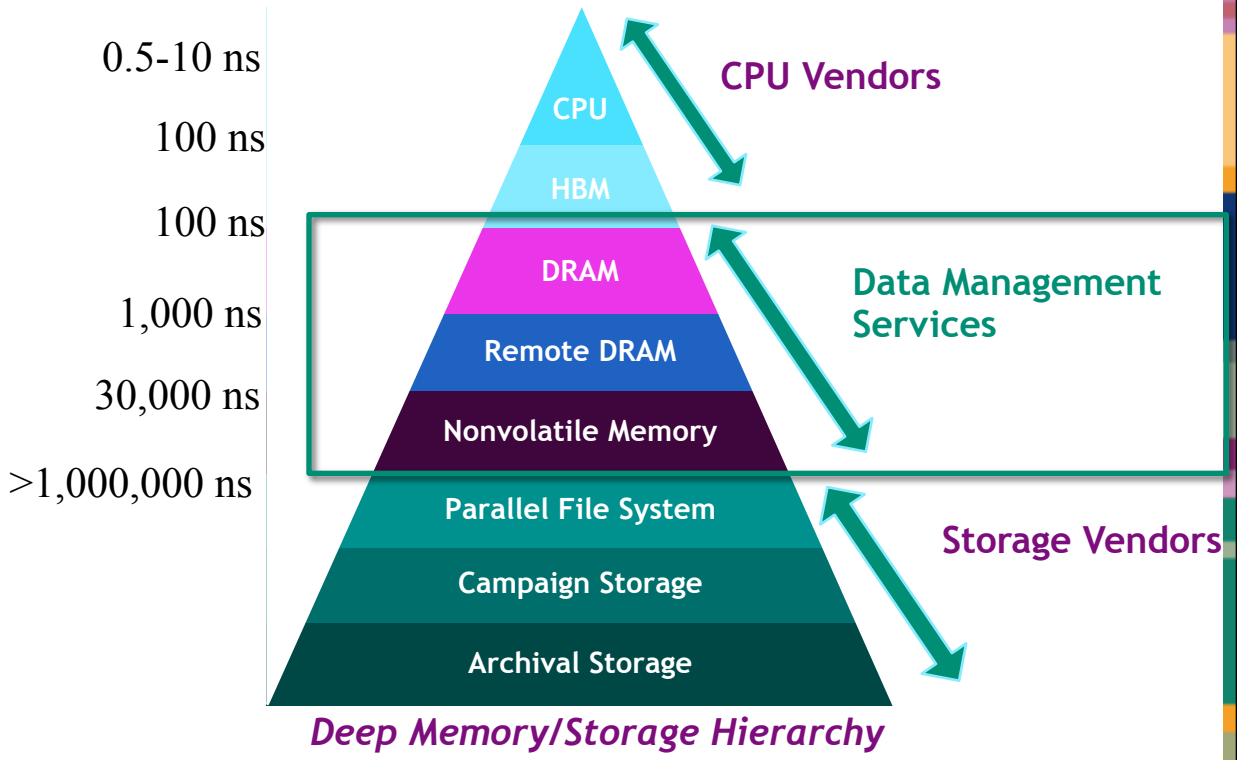
Patrick Widener, Craig Ulmer, Scott Levy, Gary Templet, Todd Kordenbrock

Sandia National Laboratories
Albuquerque NM / Livermore CA

SAND2019-6668C

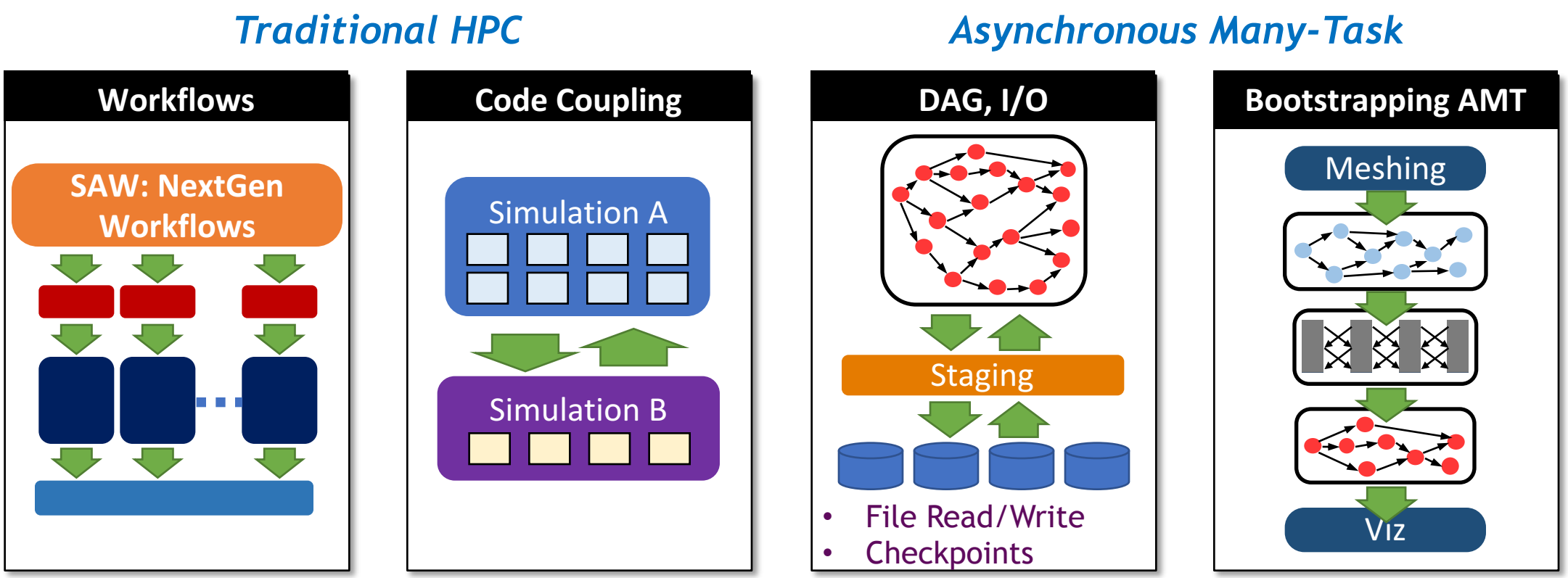
HPC applications face evolving data management needs

- Data center storage will be a focal point for HPC 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)



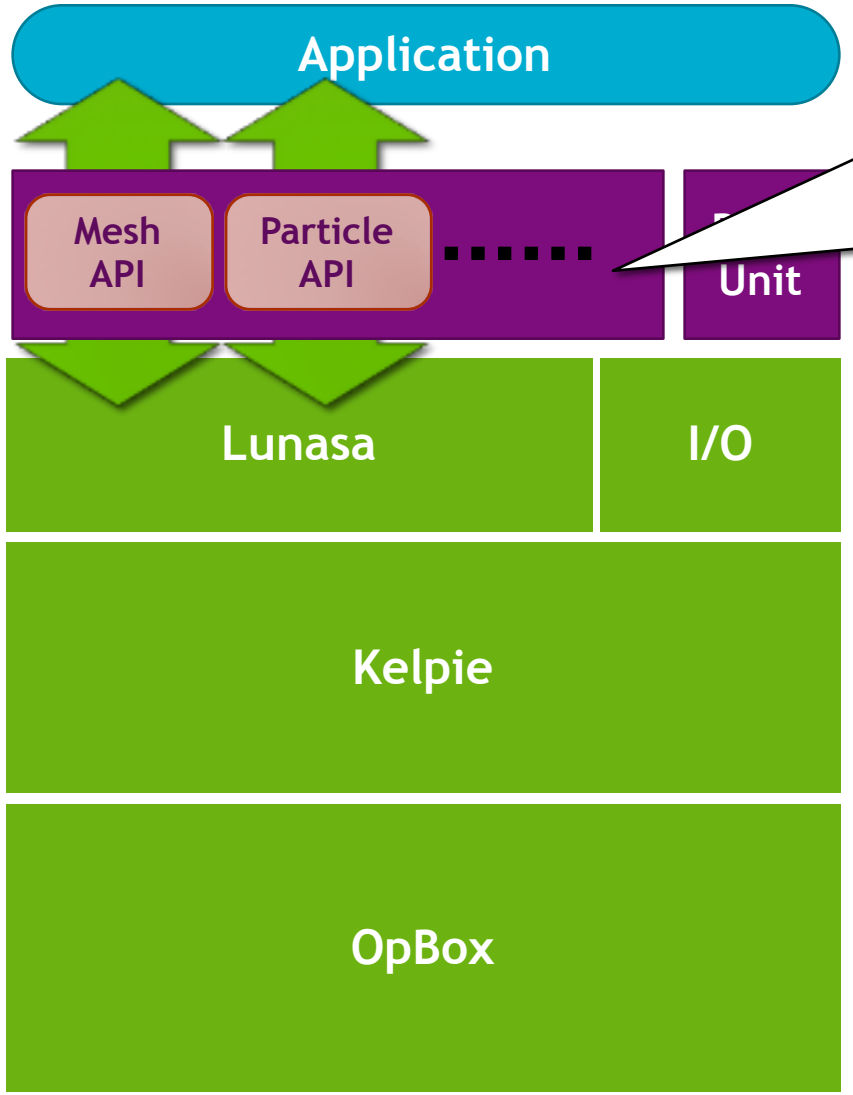
Deep Memory/Storage Hierarchy

Need for Data Management Services at Exascale



We are implementing these capabilities in the FAODEL service

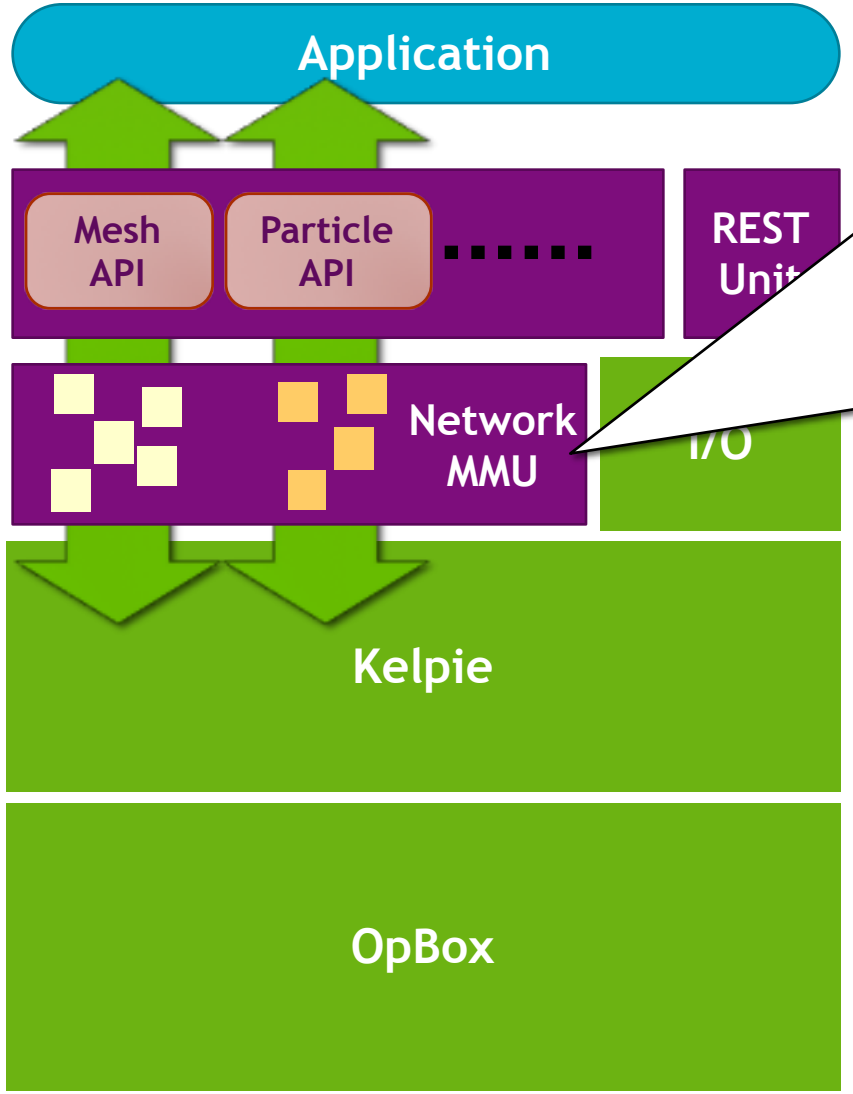
Faodel Component Structure



Data Interface Modules (DIMs)

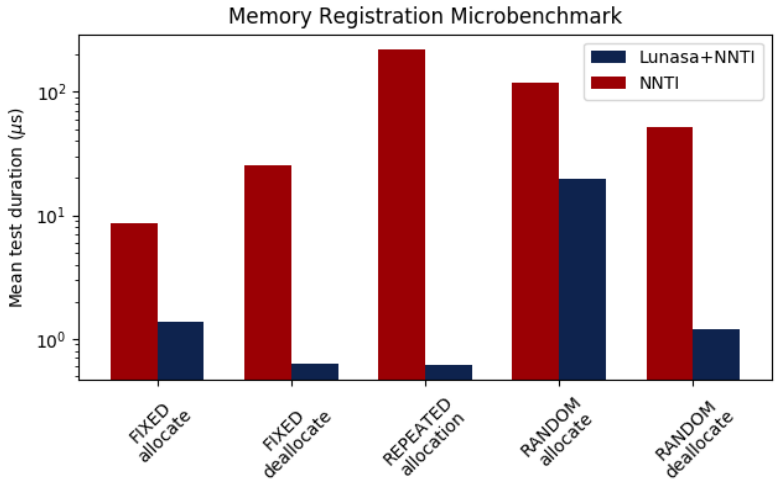
- No single API for all datasets
- Develop new modules for each dataset
- Top: Implement familiar user API
- Bottom: Faodel calls

Faodel Component Structure

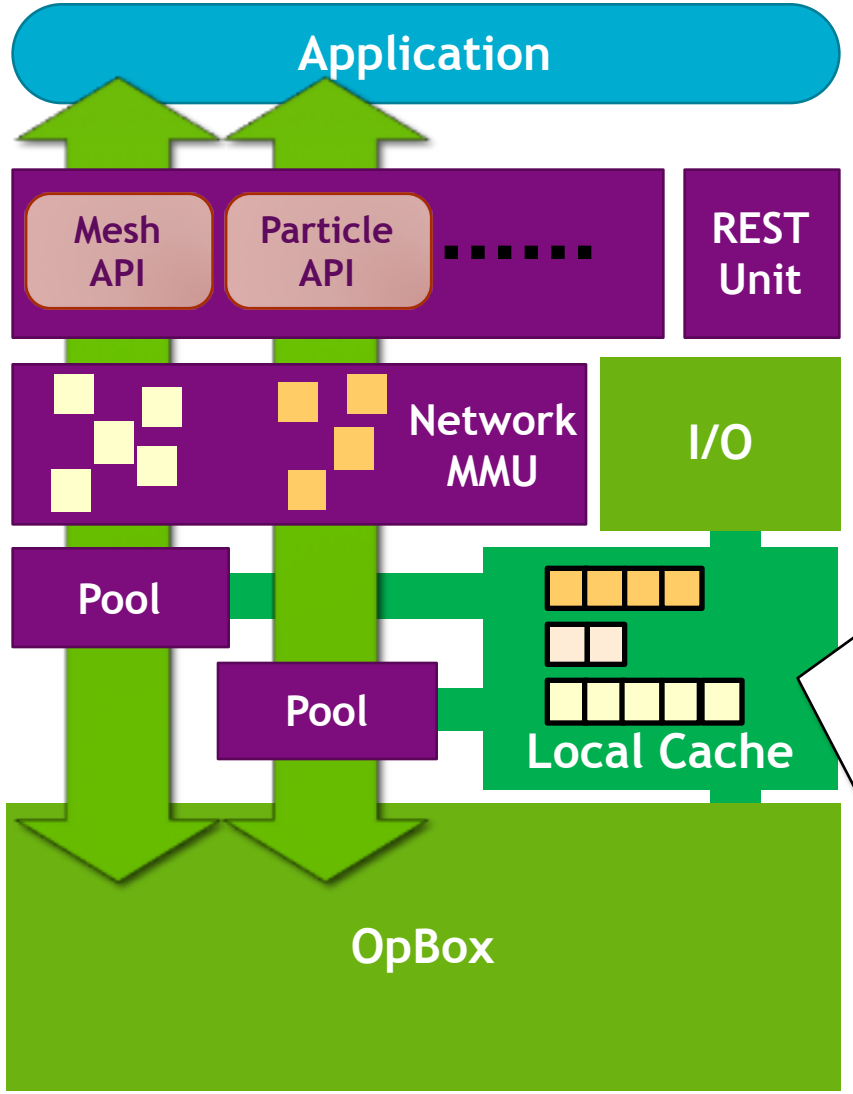


Lunasa: Network Memory Management

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

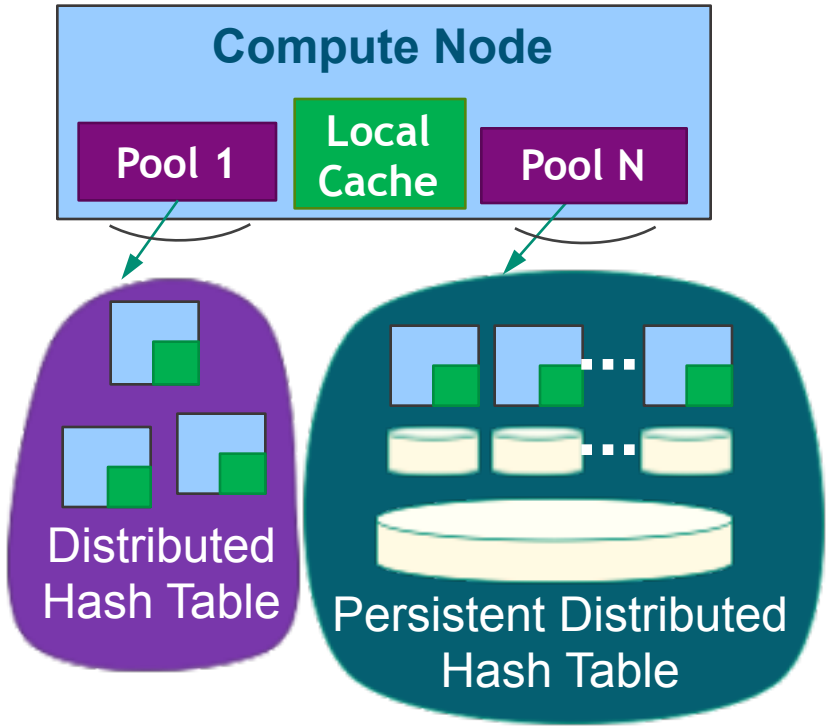


Faodel Component Structure

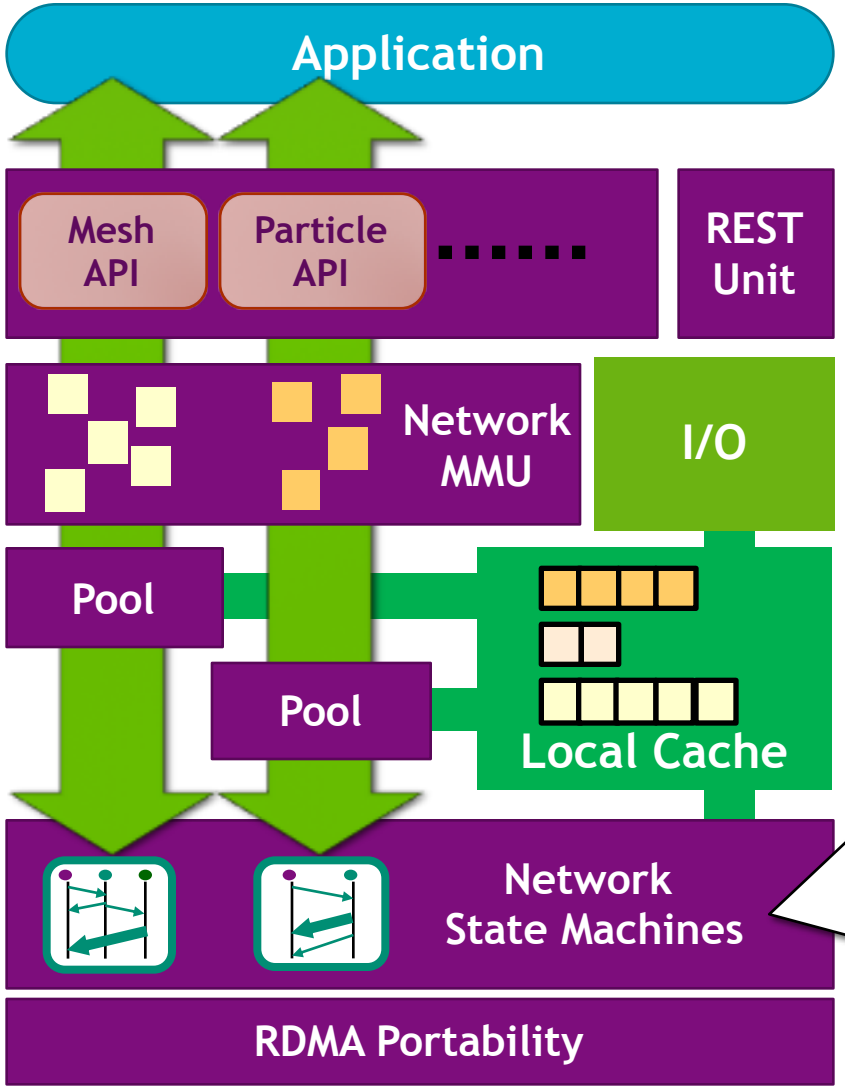


Kelpie: Distributed Key/Blob Service

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

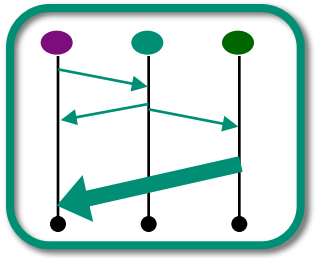


Faodel Component Structure



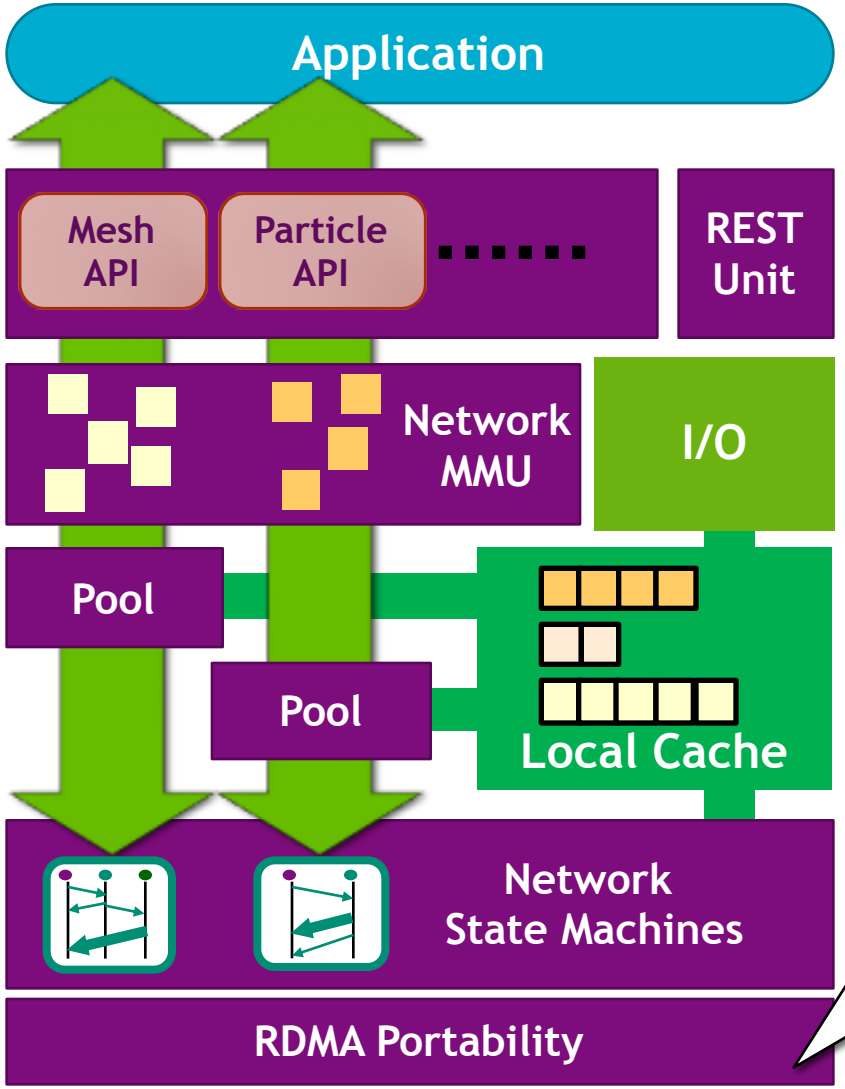
OpBox: Network State Machines

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



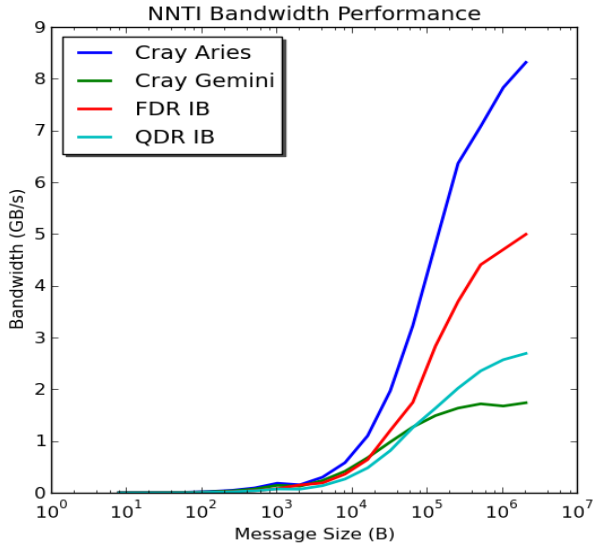
= Op

Faodel Component Structure

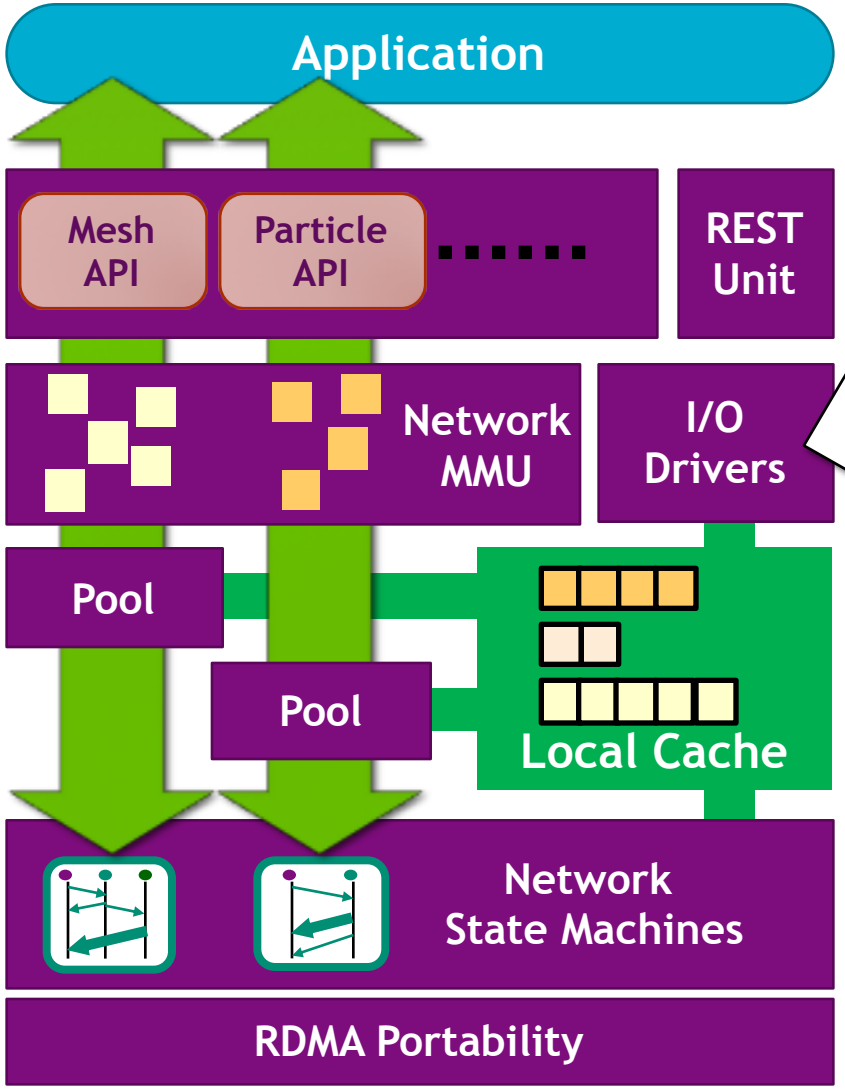


RDMA Portability

- Low-level network transfers
- Support NNTI or libfabric

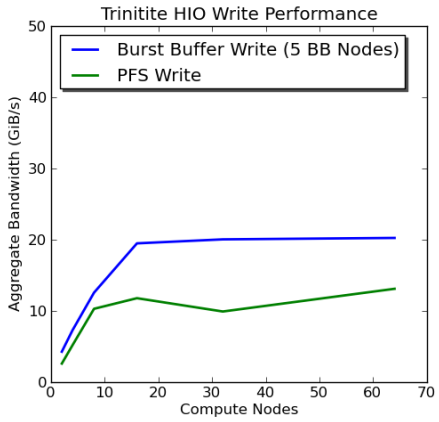


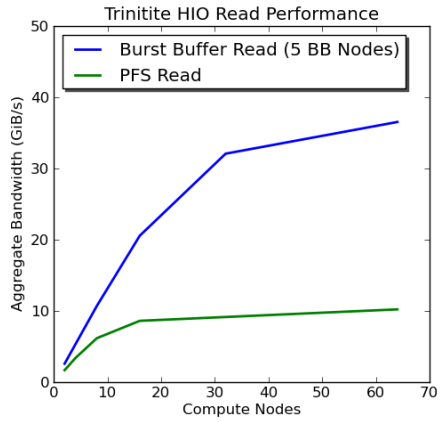
Faodel Component Structure



I/O Drivers

- Interface to Burst Buffers, NVMe, PFS
- Support for XC40 DataWarp and PFS





I/O Modules in FAODEL

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

I/O Modules (IOMs) in FAODEL

FAODEL was designed as an in-memory service

- Essentially a way to leverage node memory and fast networks in a cluster

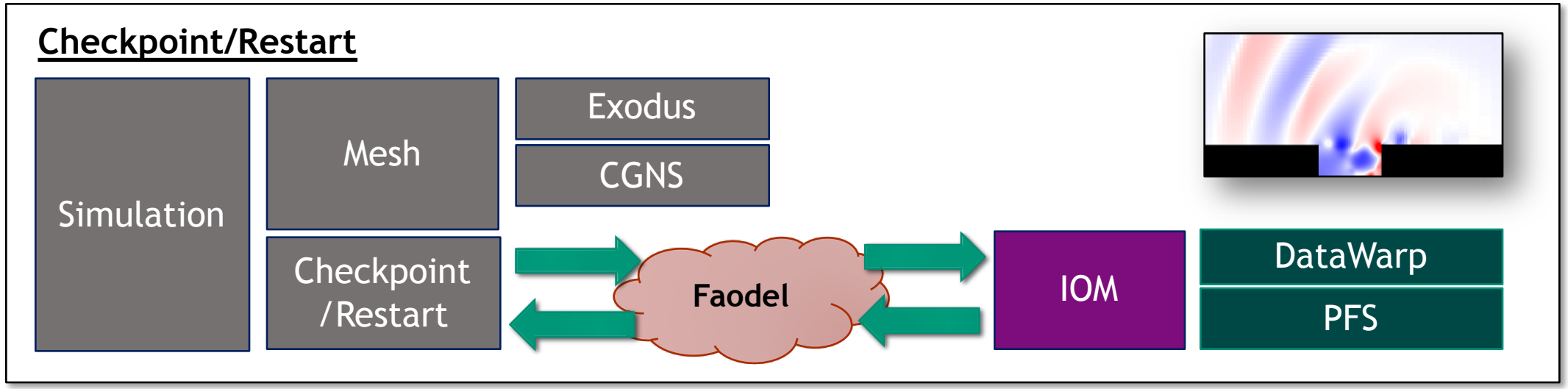
Application users wanted a way to interact with persistent storage

- Either extract from FAODEL and store, or FAODEL learns about storage

IOMs are an attempt at a middle path

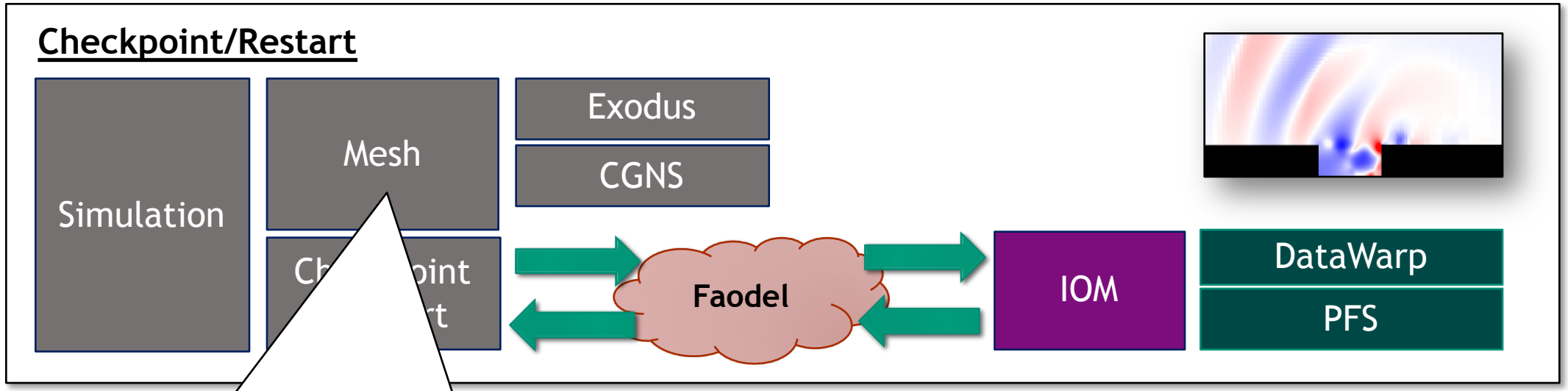
- Provide FAODEL with an abstraction of persistent storage
- Straightforward extension to many kinds of storage systems

I/O Modules for Checkpoint – Restart



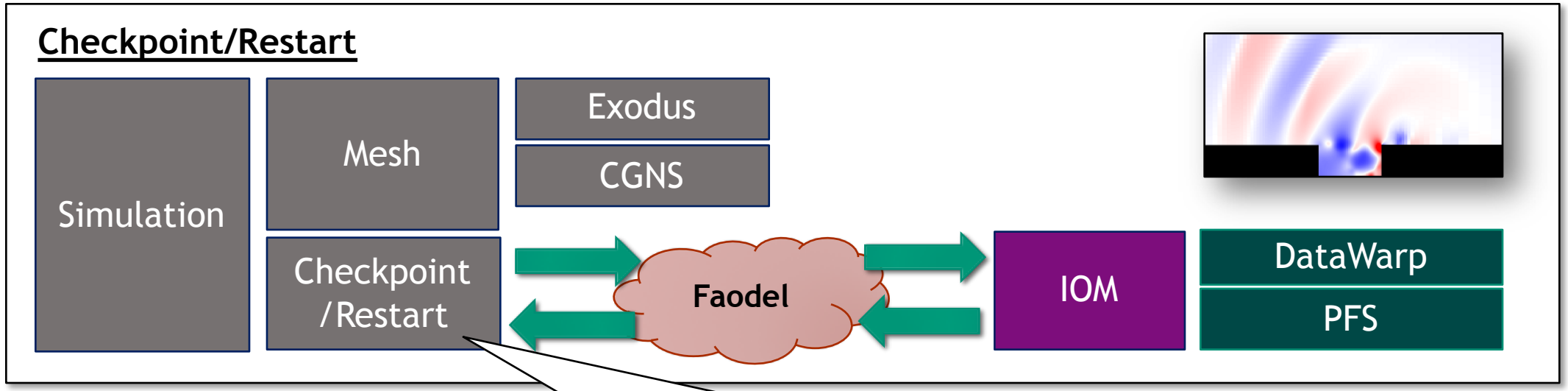
- Adding checkpoint/restart capabilities to an existing aerosciences CFD simulation code
  - Inputs are structured and unstructured meshes
- Primary restart use case is to ”bridge” long-running problems across job allocations

I/O Modules for Checkpoint – Restart



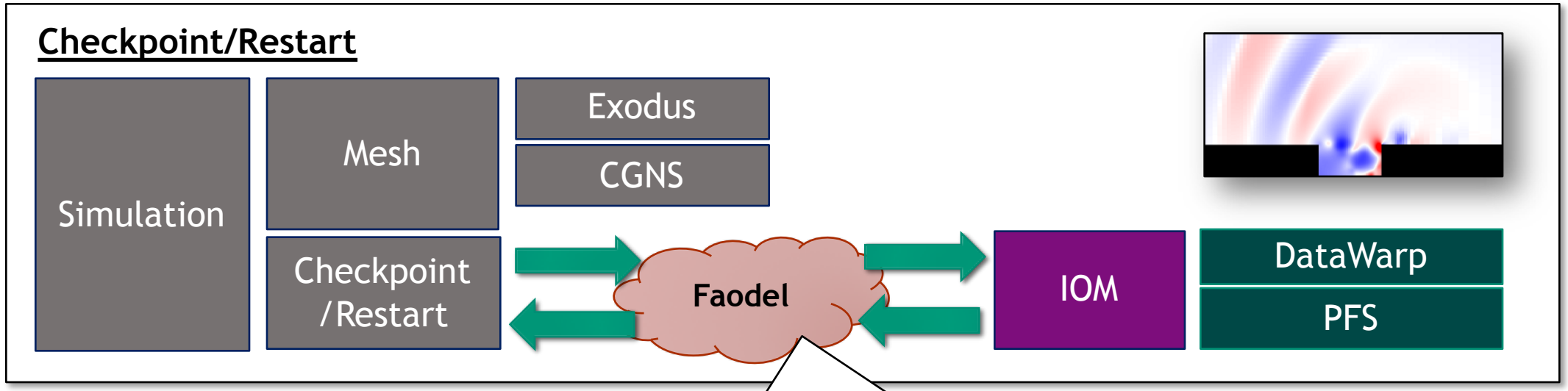
- Mesh description handled by existing file / container formats
- Exodus (NetCDF) and CGNS historically popular
  - Tied to file system
  - Complicated API, interdependent metadata updates during I/O
- Frequently the mesh structure doesn't represent the problem (which is what needs to be memo-ized)

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

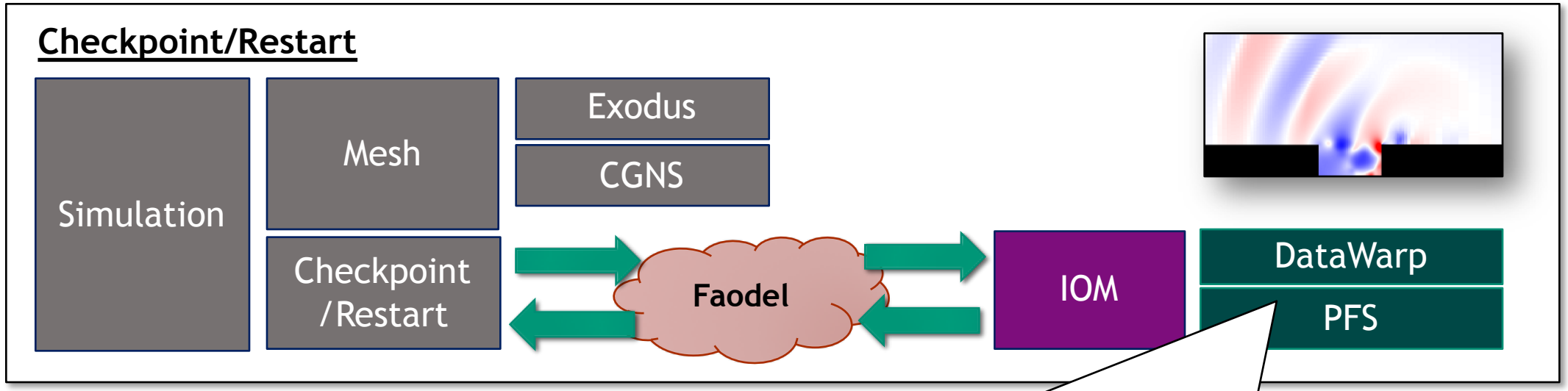
I/O Modules for Checkpoint – Restart



- 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

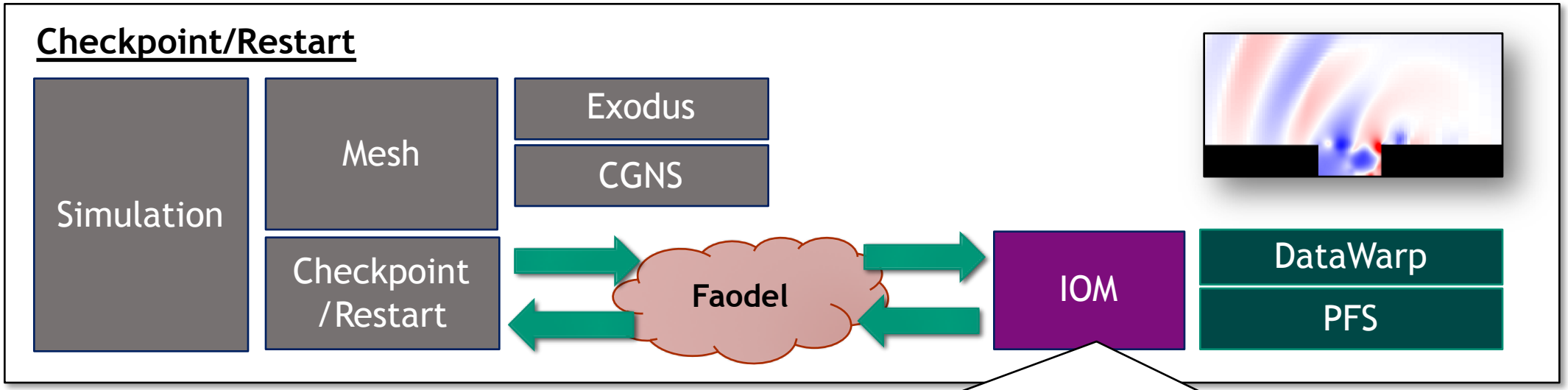
Checkpoint contents have to end up on stable storage eventually

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

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
- IOM can write to either DataWarp (Cray burst-buffer) or PFS

Expanding IOMs: dynamically mediating among storage destinations

FAODEL is an abstraction over the storage hierarchy

Not all application data has the same “shape”, though

- Metadata and control-plane content is very different from experiment results
- We want to match data shape and usage to appropriate storage tools
- We have implemented IOMs for HDF5, LevelDB, and Cassandra

Part of the IOM rationale is to enable flexible data sharing

- Reduce data transformation by storing persistently in the right place the first time
- If analytics tools use a column store, let’s just use that directly (Cassandra)

We address a shared data center environment where multiple services and APIs are available and where we cannot dictate which ones are used

Dynamically mediating among storage destinations

How to provide applications with a naturally expressive way to select an IOM?

We made it possible to attach semantics to the object namespace in FAODEL

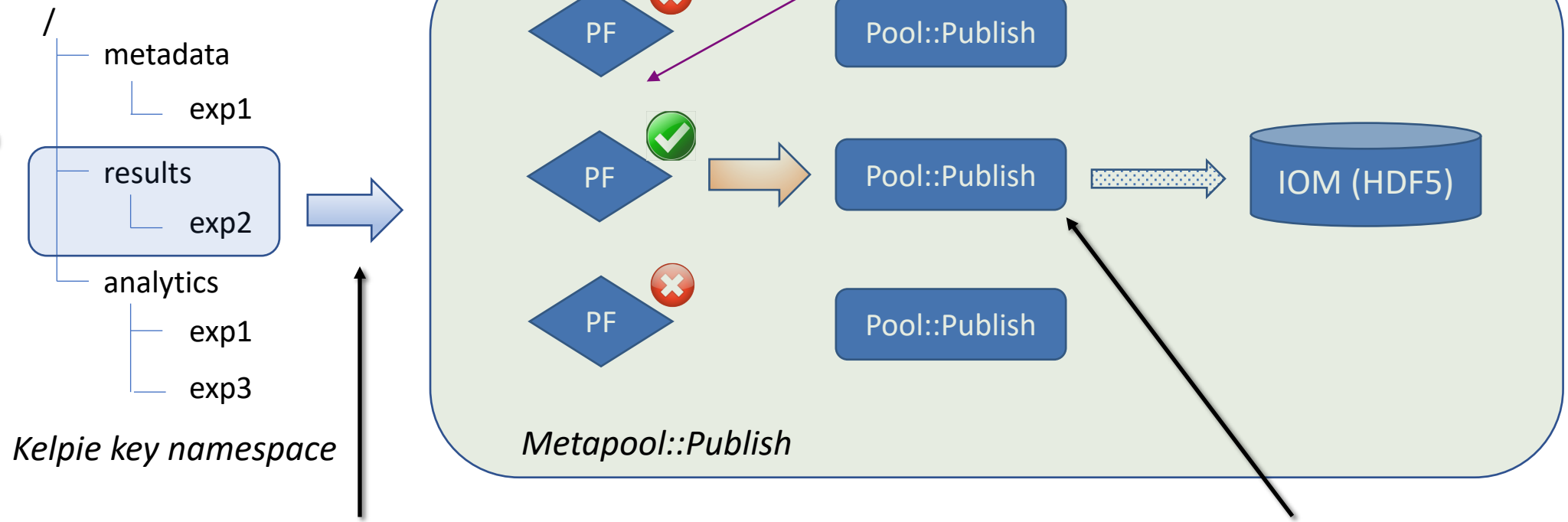
The Metapool class

- Interface resembles the Pool class (Pool manages an IOM)
- Extra methods for managing a set of a Pools and implementing semantics
- Multiple possibilities here, but to start with we use this to select IOMs

Metapool high-level view

Partition functions control which managed Pool is identified as a delegate

Application wants to Publish() "results" data at /results/exp2



Metapool associates a partition of the key namespace with an IOM (storage)

Publish() / Want() requests are delegated to a Pool based on what part of the key namespace they address

Defining partition functions

```
hdf5_dht = kelpie::Connect( "ref:/myapp/results" );
leveldb_dht = kelpie::Connect( "ref:/myapp/metadata" );
cassandra_dht = kelpie::Connect( "ref:/myapp/analytics" );

metapool.Manage( hdf5_dht,
[]( const kelpie::Key& k ) {
  if( k.K1.substr( 0, 8 ) != "/results" ) return false;
  return true;
} );

metapool.Manage( leveldb_dht,
[]( const kelpie::Key& k ) {
  if( k.K1.substr( 0, 9 ) != "/metadata" ) return false;
  return true;
} );

metapool.Manage( cassandra_dht,
[]( const kelpie::Key& k ) {
  if( k.K1.substr( 0, 10 ) != "/analytics" ) return false;
  return true;
} );
```

Separate Pool constructed for each storage modality

App provides partition functions as C++ lambdas through the Metapool::Manage() method

PF is given the Key of the Publish request. Can take other actions.

Using the Metapool interface

- Metapool iterates its collection of partition functions
- First true response forwards the Publish request to the associated Pool

```
for( int i = 0; i < 25; i++ ) {
  kelpie::Key k;

  k.K1( "/metadata/" + random_string( 10 ) );

  lunasa::DataObject ldo( 0, 256, lunasa::DataObject::AllocatorType::eager );
  metapool.Publish( k, ldo );
}

for( int i = 0; i < 25; i++ ) {
  kelpie::Key k;

  k.K1( "/results/" + random_string( 10 ) );

  lunasa::DataObject ldo( 0, 256 * 1e6, lunasa::DataObject::AllocatorType::eager );
  metapool.Publish( k, ldo );
}
```

Other Metapool policies possible: random Pool selection, non-binary PF response, weighted PF response, not-just-PF

FAODEL service design allows such policies to be changed dynamically & by third party

Evolution of the Metapool concept

Metapool provides benefits at a macro scope
- Matching data to storage service characteristics
- Raw performance is governed by the storage service itself

Handling of IOMs can evolve in different ways depending on available/installed services

Namespace semantics can also be used for different things
- Caching behavior, TTL, protection mechanism, others?
- This is defined by applications, so overloaded namespace components is for them to resolve

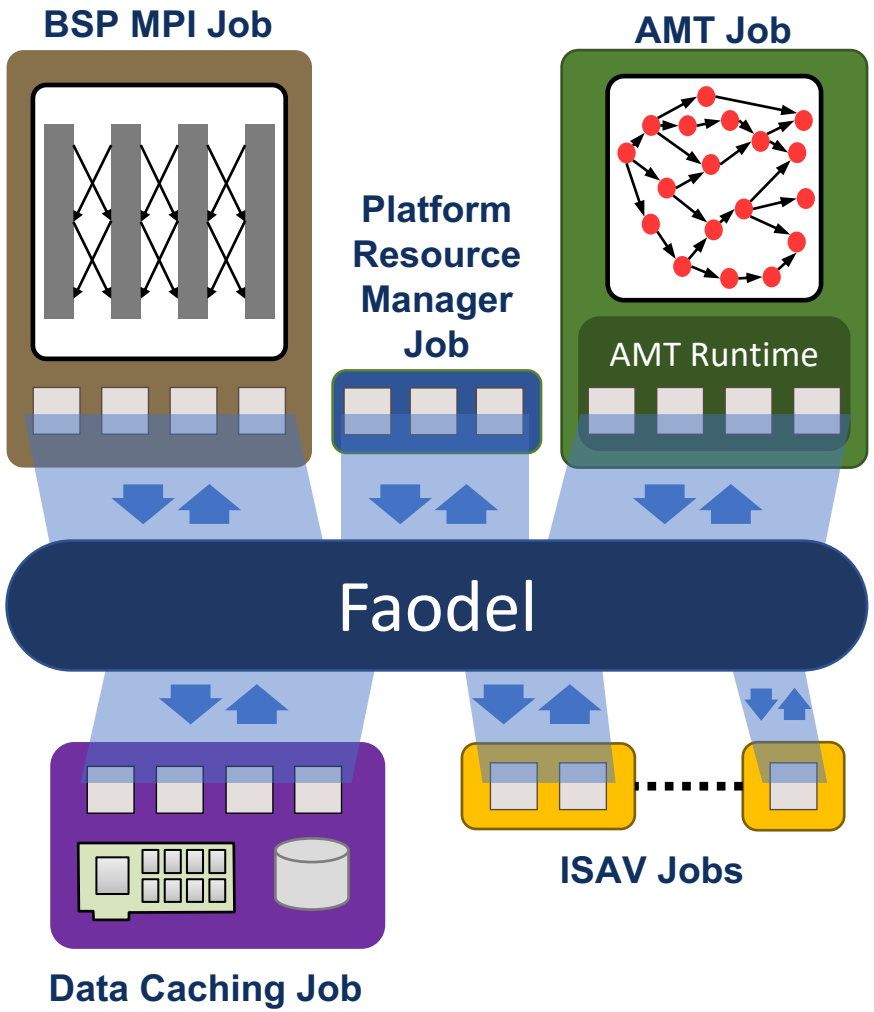
Ability to change mediation behavior externally opens up policy management possibilities

Perhaps a good thing to keep storage destinations opaque
- Users match the known “shape” of data with a set of performance/capability characteristics

Metapool is an example of FAODELs usefulness as a mediation point
- … between applications & storage, within workflows

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
- This exploration of IOM capabilities is an example of extending those 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
- A 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.