Rabbit Storage for El Capitan

Fast I/O through Big, Pointy Teeth

Brian Behlendorf – behlendorf@llnl.gov
Olaf Faaland – faaland1@llnl.gov

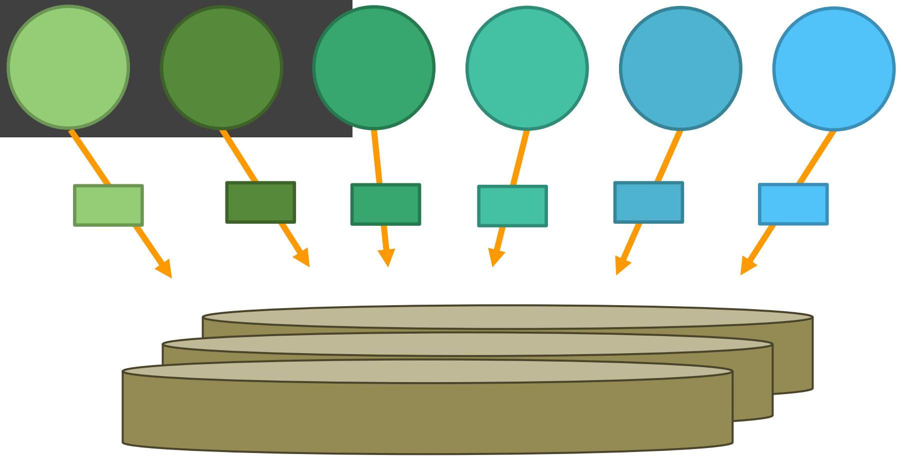
LUG May 2023

Storage systems’ job is getting more challenging

- El Capitan reflects general changes in HPC
- Compute rates growing faster than I/O rates and storage capacity
  - Sierra 125 petaFLOPs
  - El Capitan 2 exaFLOPs (projected)
  - Not so easy to make the file system 16x as fast and 16x as large
- Need to support traditional HPC I/O workloads
  - Defensive I/O / checkpoints / reading input files / writing output
  - Existing codes need be able to run with minimal to no modifications
- Also need to support new computing paradigms on our systems
  - Machine learning, AI
  - Complex simulation workflows
  - In situ analysis and data reduction
  - I/O libraries and tools
- What can we do?

Traditional HPC workloads – the usual suspects

```
int main(int argc, char* argv[]) {
  MPI_Init(argc, argv);

  read_input();

  for(int t = 0; t < TIMESTEPS; t++)
  {
    /* ... Do work ... */
    /* ... Synchronization ... */

      write_output();

  }

  MPI_Finalize();
  return 0;
}
```



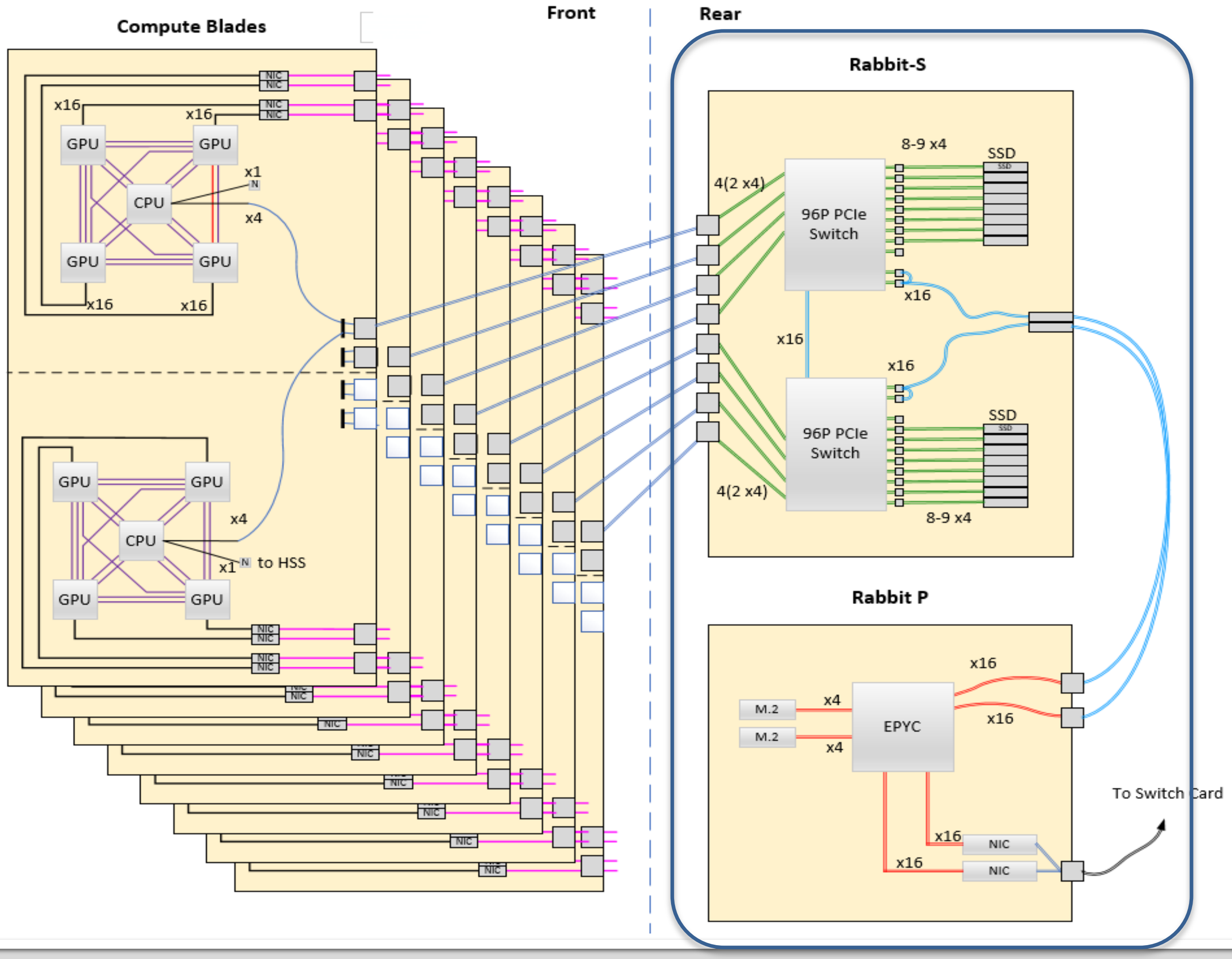
- Lustre works great, it's what it was built for!
  - But there are some things to be aware of
- Within one simulation, processes perform file operations simultaneously, increasing load and contention on the global parallel file system
- Multiple simulations running at the same time compete in ways mysterious to users
- Past burst buffer implementations not widely adopted
- What can be done:
  - Good: do less I/O
  - Better: faster filesystem built with SSDs
  - Better still: usable burst buffer (AKA file system per running application)

Machine learning training workloads are stressing the I/O system in new ways for HPC systems

- Machine learning training performs a large number of small, random reads
- Parallel file systems are not designed for this workload - local low-latency storage is a better fit
- Example: LBANN crashed Sierra with a large-scale training run with 1000 trainers
- What can be done:
  - Good: optimize for parallel file systems
  - Better: direct PCIe attached block storage
  - Best: low-latency shared storage on multiple nodes

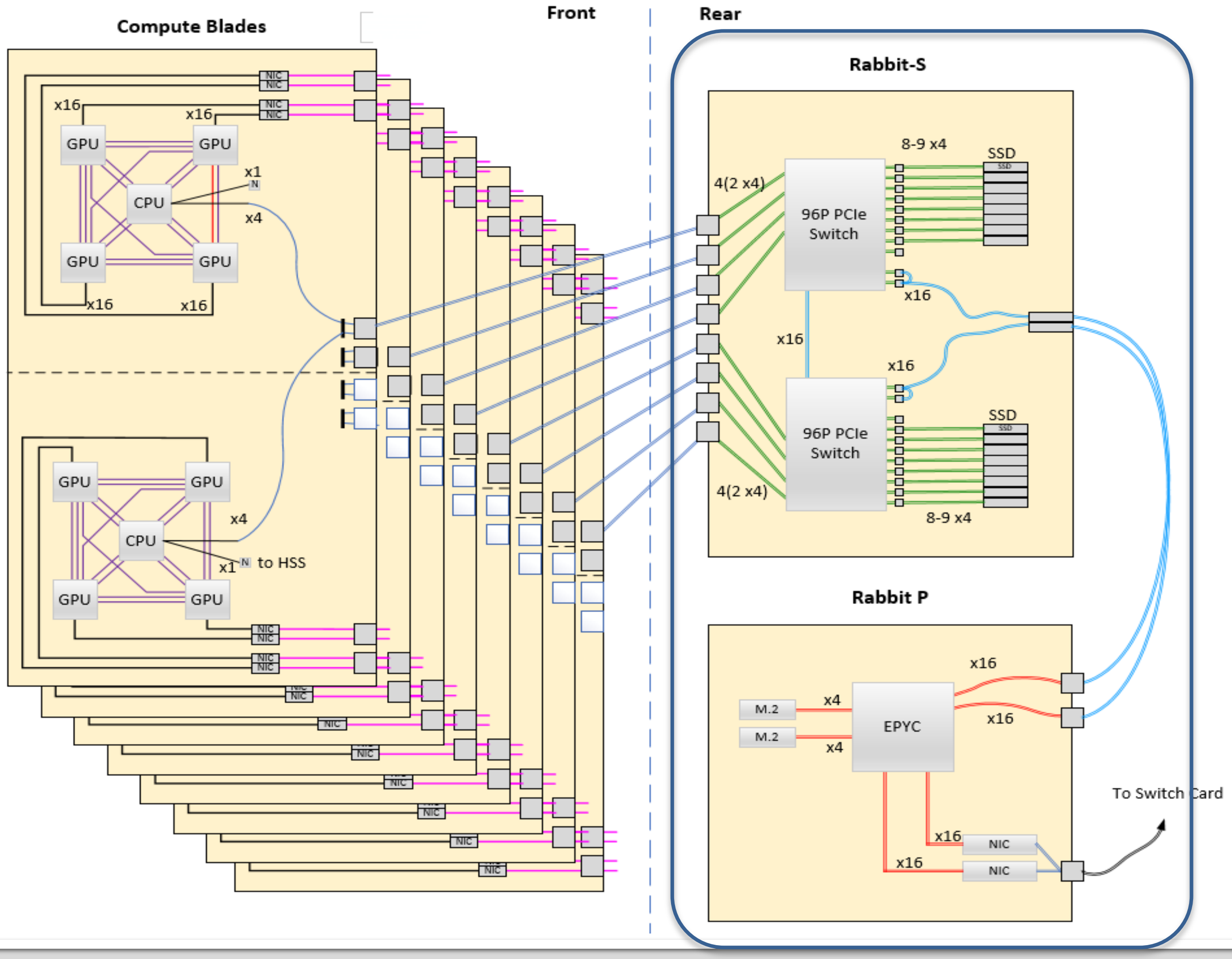
LBANN: Livermore Big Artificial Neural Network Toolkit:
https://github.com/LLNL/lbann

Rabbit module: in-chassis local storage++



- Architecture which can support both usage models with shared storage hardware
- One Rabbit module per compute chassis
  - 18 SSDs in the Rabbit-S storage enclosure
  - Rabbit-P AMD Epyc based storage node
- SSDs directly PCIe attached to the Rabbit-P node and compute blades
- Rabbit-P node connected to the high-speed interconnect

Rabbit module: in-chassis local storage++



OK, we have the hardware, but ...

How is access controlled?

How is it prepared for use and torn down?

How does the user request it?

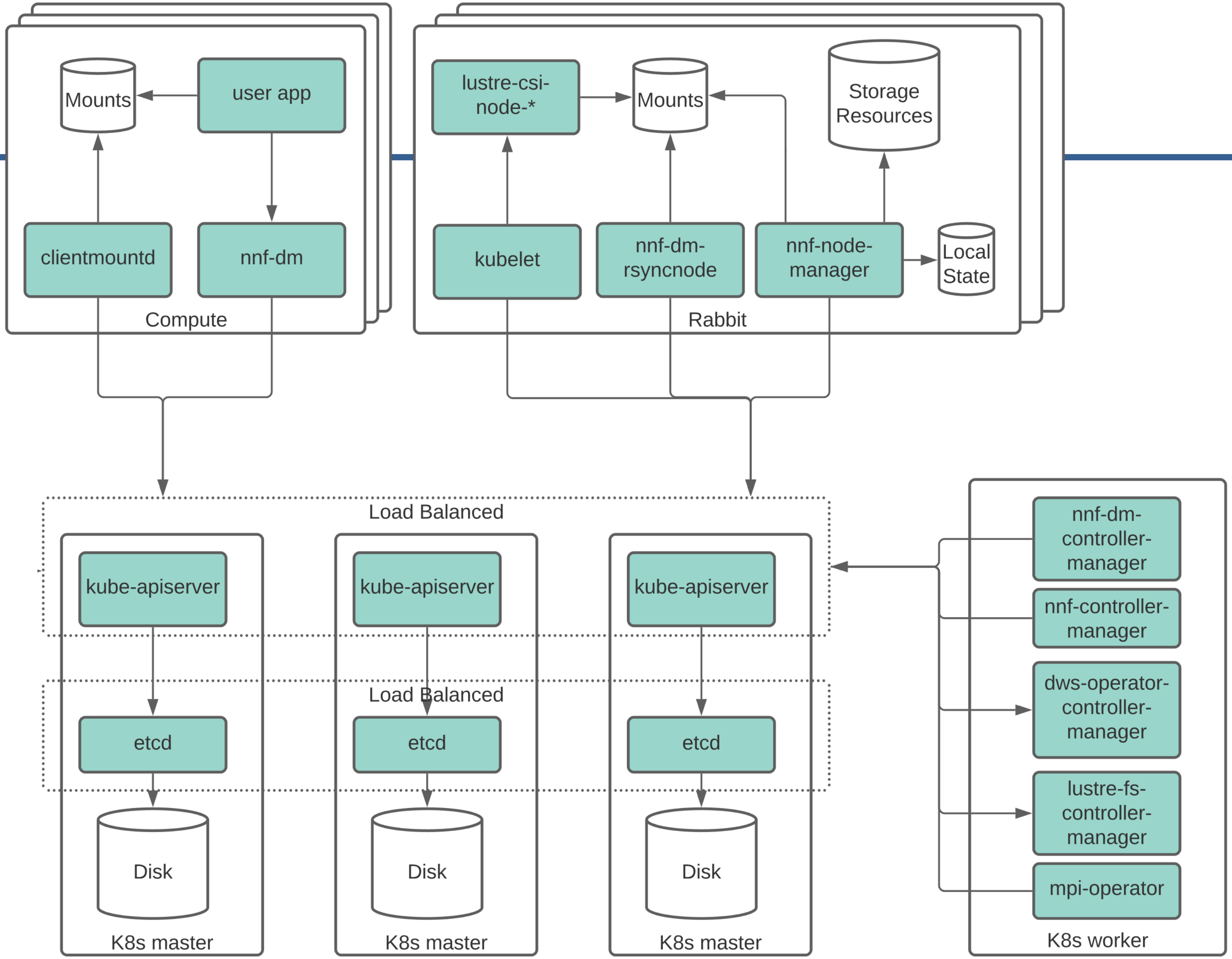
Operating system stack

- Nodes run Tri-Lab Operating System Stack (TOSS) *
  - Linux distribution focused on HPC clusters
  - All the good bits from RHEL plus more, including …
    - ZFS and Lustre
    - Kernel with addition drivers necessary (minimal changes)
    - Flux, Kubernetes, and Rabbit (NNF) software

* https://dl.acm.org/doi/10.5555/3433701.3433754

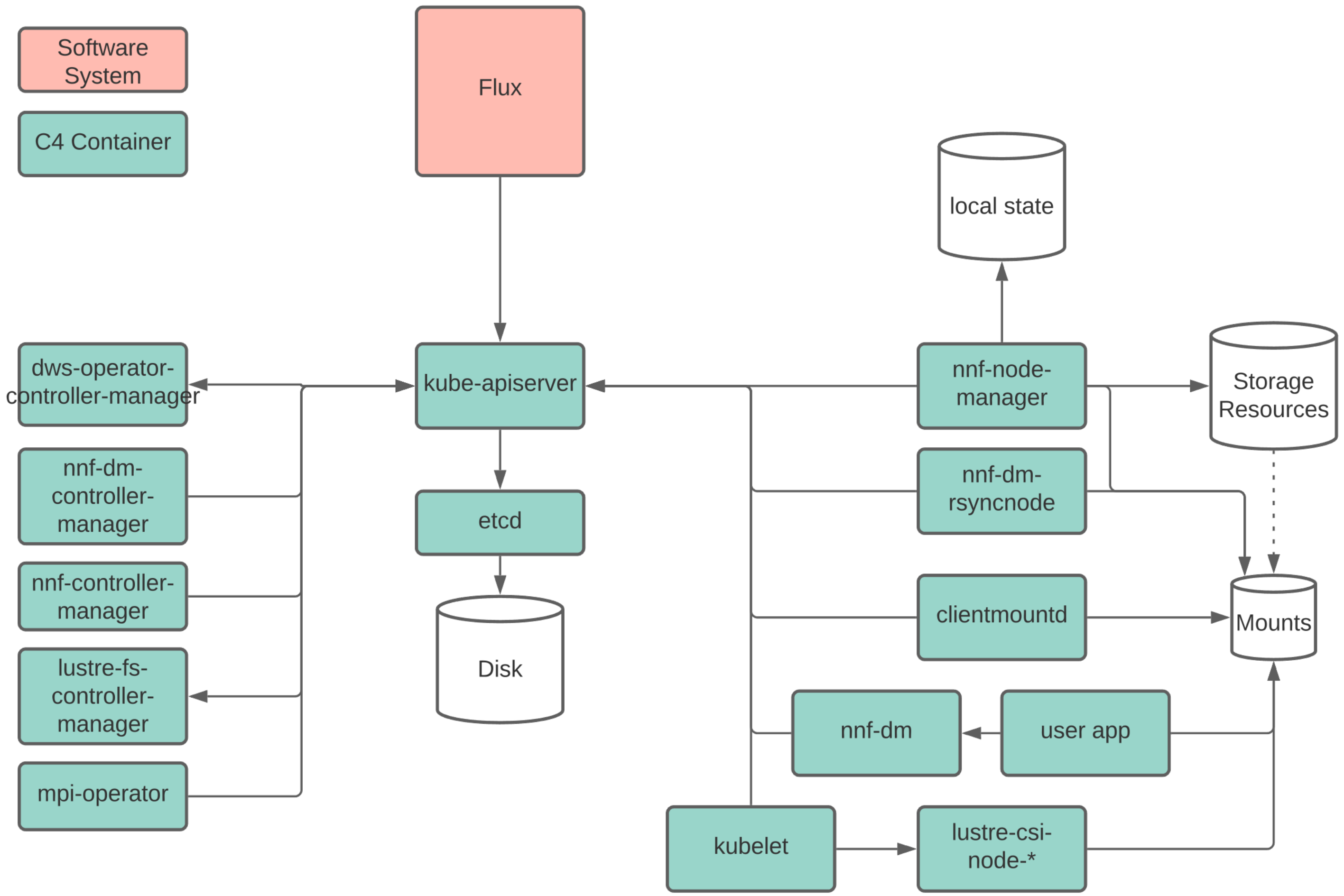
Rabbit storage orchestration stack

- Managed by HPE provided containers
  - Reconfigures PCIe fabric
  - Creates / destroys NVMe namespaces
  - Creates / destroys filesystems
  - Performs asynchronous data staging
  - Rabbit storage state maintained in etcd
  - Interprets job provided storage requests
- Developed for a Kubernetes cluster
  - K8s deployed on TOSS (stateless)
  - Integrated with Ansible
  - K8s on rabbit and worker nodes
  - Redundant K8s master nodes
- Services running on compute nodes
  - Filesystem mount daemon
  - Data movement daemon



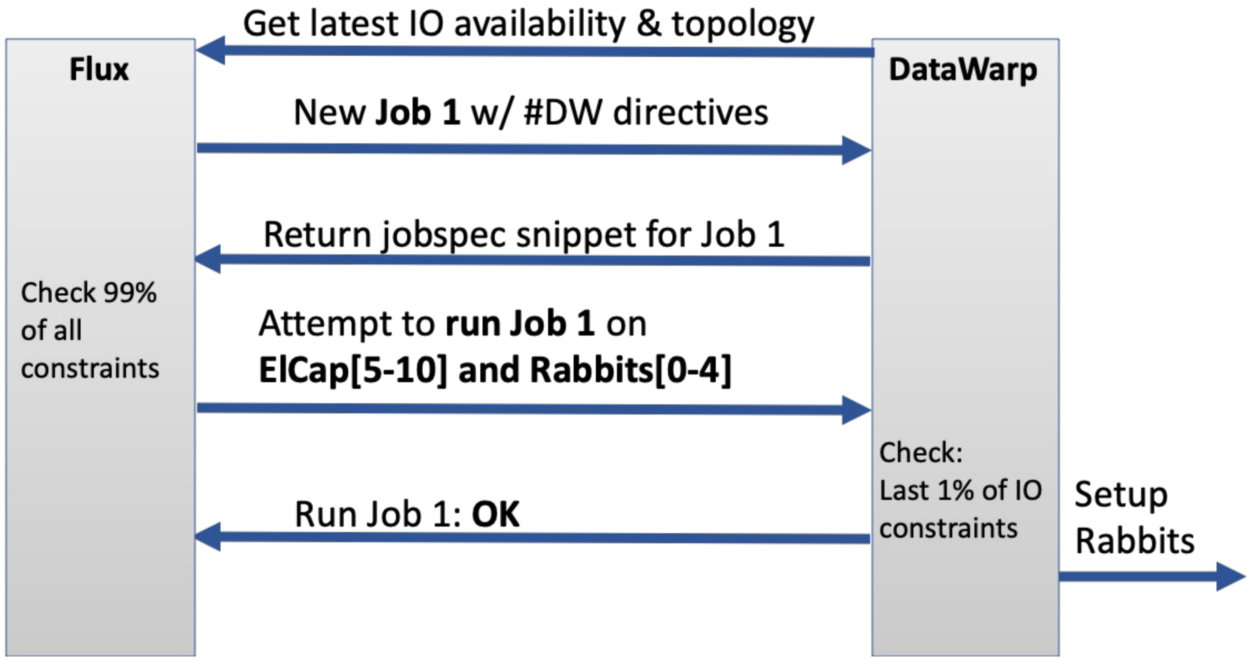
Flux resource manager allocates compute and storage

- Flux uses NNF interfaces to allocate and control resources
- Rabbit Kubernetes's interfaces were co-designed with HPE to support Flux
  - Exposed resource granularity
  - Resource discovery interface
  - Resource request translation interface
  - Resource allocation interface
  - Performance degradation notification
- Requires unique scheduling capabilities not provided by traditional HPC schedulers
  - Locality aware scheduling for compute and storage resources



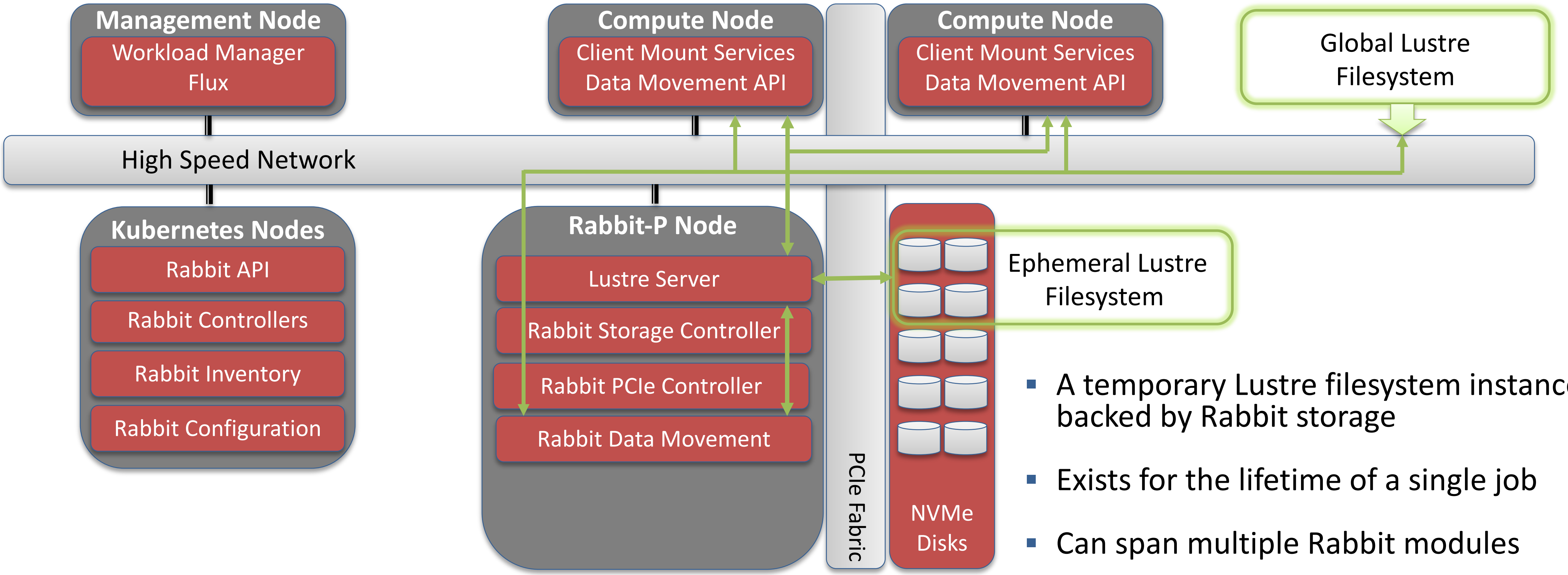
Dynamic reconfiguration between node-local and job-local storage

- DataWarp directives (#DW)
  - *jobdw* – ephemeral filesystem
  - *stage_in* – copy data in before job
  - *stage_out* – copy data out after job
  - *create_persistent* – persistent filesystem
  - *destroy_persistent*
  - *persistentdw*
  - *container* – request user container
- Directives provided to Flux as part of of the user's job script
- Multiple directives can be specified



A Model of Interaction Between DataWarp and Flux

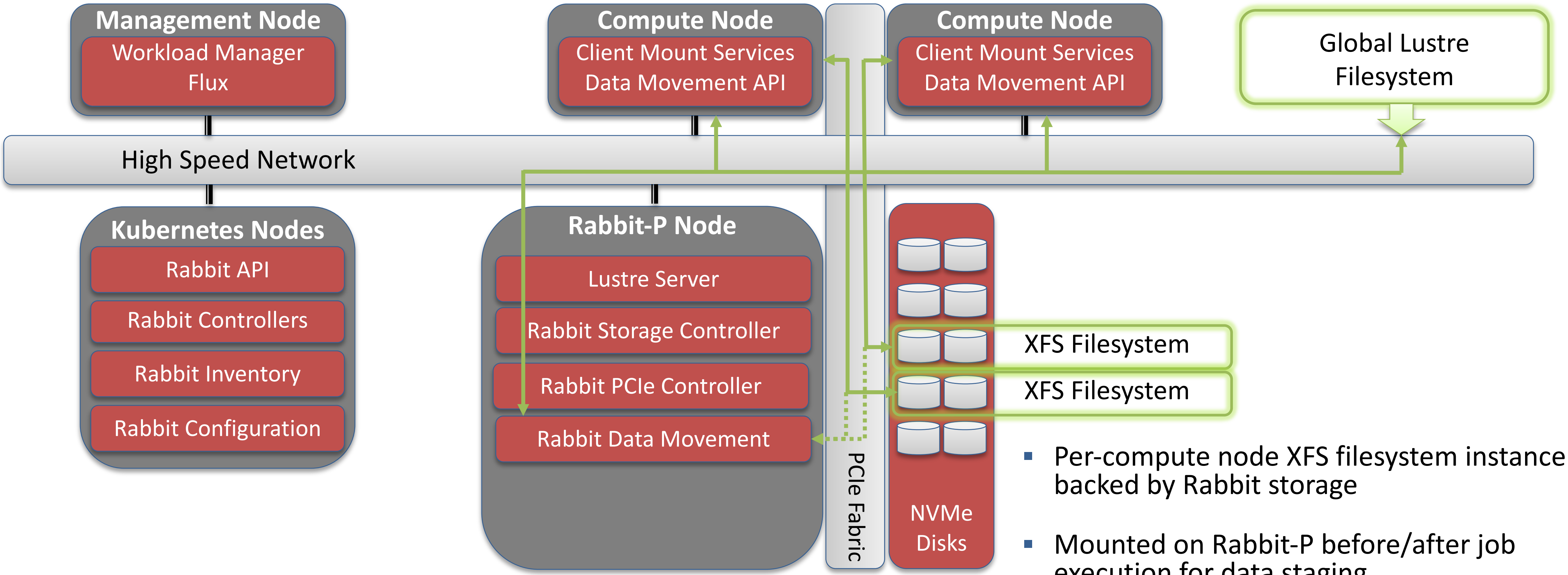
Ephemeral Lustre and data staging



- A temporary Lustre filesystem instance backed by Rabbit storage
- Exists for the lifetime of a single job
- Can span multiple Rabbit modules
- Data stage-in/out from the Rabbit-P

```
#DW jobdw type=lustre capacity=100TB name=lustre-job1 profile=metadata
#DW stage_out source=$DW_JOB_lustre-job1 dest=/p/lustre1/<username>/data/
```

Ephemeral direct attached local filesystem and data staging

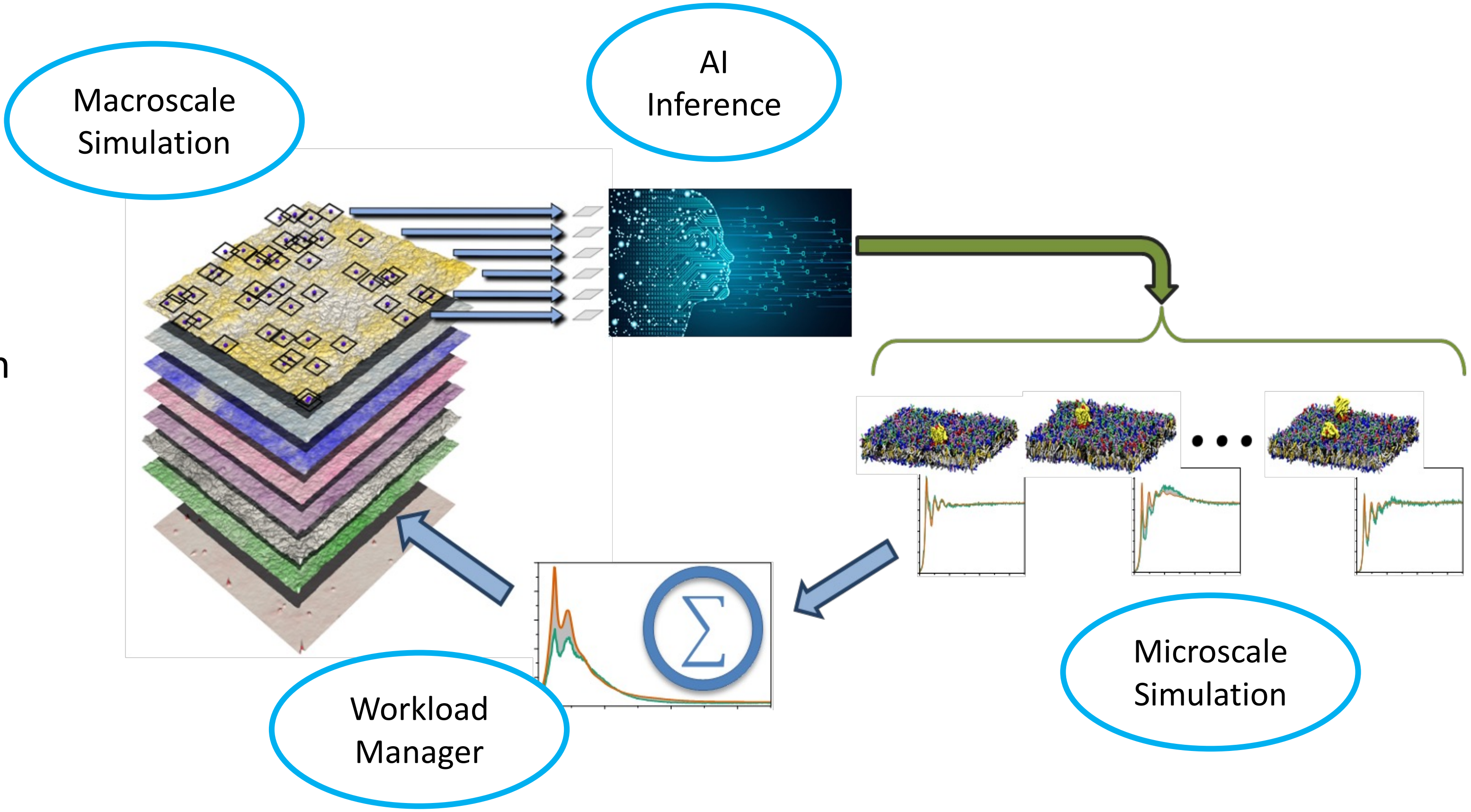



```
#DW jobdw type=xfs capacity=500GB name=xfs-job1
#DW stage_in source=/p/lustre1/<username>/data dest=$DW_JOB_xfs-job1
```

- Per-compute node XFS filesystem instance backed by Rabbit storage
- Mounted on Rabbit-P before/after job execution for data staging
- Remounted on compute during job execution

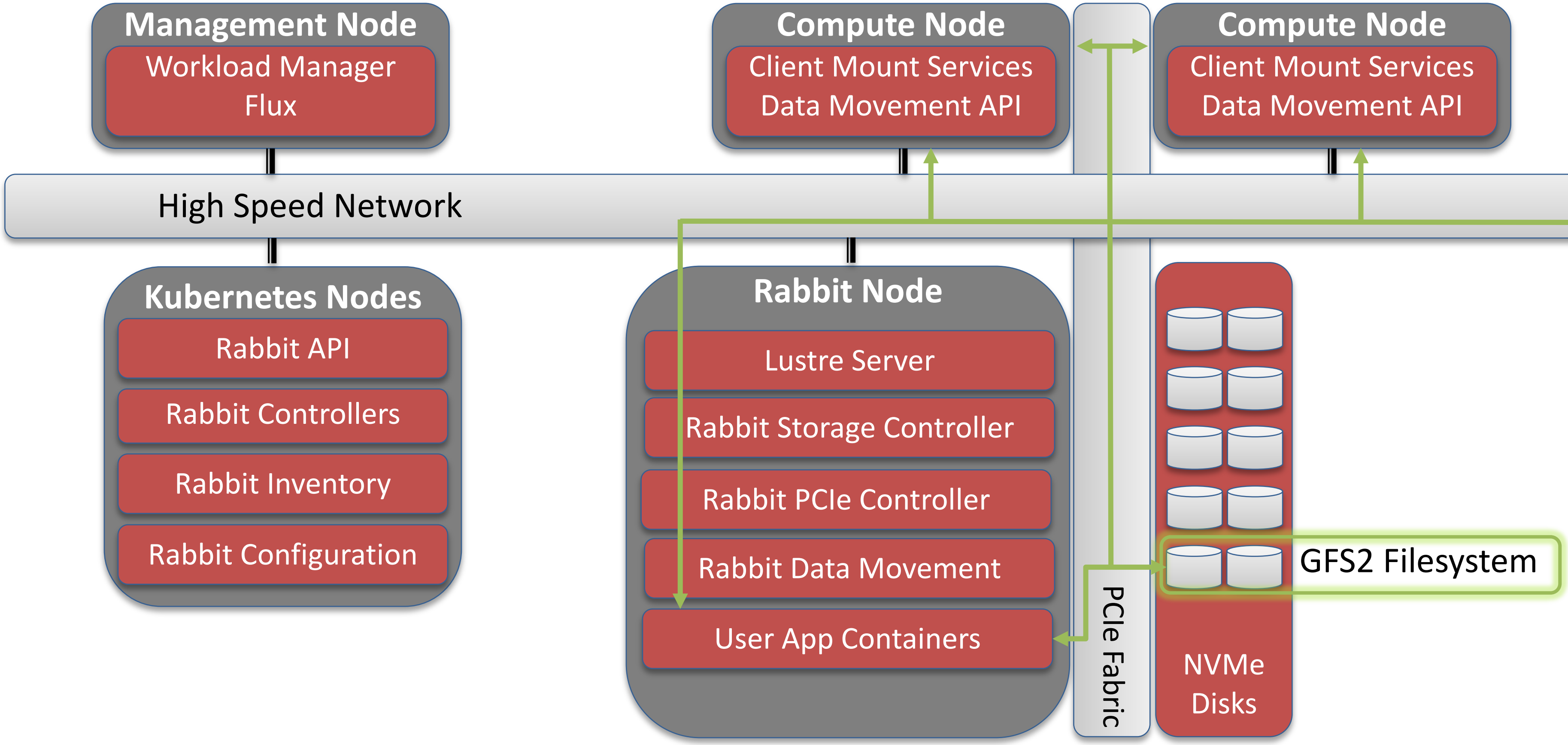
More complex workflows can take advantage of containers and local shared storage

- Some complex workflow I/O patterns can be problematic for parallel file systems
  - Random small read/writes I/O
  - Smaller, temporary files passed between stages are non-optimal and can create bottlenecks
  - Problematic when the data needs to be shared between nodes in a workflow
- Portions of a workflow can be containerized to run on a rabbit node
  - Access to shared node-local storage on compute and rabbit node via GFS2



MuMMI: Machine-Learned Modeling Infrastructure: https://github.com/mummi-framework, Di Natale, SC’19

Complex storage workflows using a user container




```
#DW jobdw type=gfs2 capacity=500GB name=gfs2-job1
#DW container name=job1 profile=foo JOB_DW_foo-local-storage=gfs2-job1
```

- Shared direct PCIe attached storage
  - Avoids swamping the HSN with data movement
- Containers on the Rabbit can be used for:
  - SCR for data movement
  - I/O libraries and tools
  - Data management, reduction, migration
  - In situ analysis
  - Machine learning training

Scalable Checkpoint / Restart (SCR) Library support

Currently working to add Rabbit support to SCR

- Plan to support all three ephemeral file system types (Lustre, xfs, GFS2)
  - When using Lustre and xfs, data movement to permanent file system will be via threads running on compute nodes
  - When using GFS2, goal is to do data movement on rabbits in a container
- An "out-of-the-box" option to utilize the Rabbits for some applications
- An example implementation for reference purposes

https://github.com/LLNL/scr
https://computing.llnl.gov/projects/scalable-checkpoint-restart-for-mpi

Applications can gradually adopt new strategies to support their I/O needs

1. Applications can still just write/read to the Lustre capacity tier (Will work but will not have scalable bandwidth compared to Rabbits.)

2. Applications allocate file systems on Rabbit using HPE-provided options for faster performance than global Lustre capacity tier

3. Applications utilize software libraries / tools integrated with Rabbit to provide even better performance and portability

4. Workflow, analytics frameworks, I/O middleware can be ported to the Rabbits and provide better performance and new capabilities to codes

Progress So Far

- Early Access System with Rabbit prototype hardware installed at Livermore
- Containerized HPE Rabbit software deployed to TOSS-based Kubernetes environment
  - Starting to test out delivered functionality
- Hardware / software shakeout, evaluation, and testing underway
  - Uncovered several hardware / firmware issues
  - Iterated with HPE through software issues via github
- Continued integration work (Flux, TOSS, Kubernetes, monitoring, etc)
- Additional prototype Rabbits being installed now

Rabbit Resources

- Rabbit software has been open sourced and is available at GitHub
  - Using PRs, Issues, Actions, and container repositories
- Documentation:
  - https://nearnodeflash.github.io
- Source code and container repositories:
  - https://github.com/NearNodeFlash/nnf-dm
  - https://github.com/NearNodeFlash/nnf-sos
  - https://github.com/NearNodeFlash/nnf-deploy
  - https://github.com/NearNodeFlash/lustre-fs-operator
  - https://github.com/HewlettPackard/lustre-csi-driver
- Entertainment:
  - https://en.wikipedia.org/wiki/Rabbit_of_Caerbannog

Flux Framework Resources

- Flux software has been open sourced and is available at GitHub
  - Using PRs, Issues and Actions
- Documentation:
  - https://flux-framework.readthedocs.io/en/latest/
- Source code repositories:
  - https://github.com/flux-framework/flux-core
  - https://github.com/flux-framework/flux-coral2
  - https://github.com/flux-framework/flux-sched

Thank you!

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