Introduction

Flux Tutorial at ECP Annual Meeting, Feb 6, 2020

Dong H. Ahn

LLNL-PRES- 804364

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

Lawrence Livermore National Laboratory

Today's Agenda

Time	Topic	Presenters
08:30AM – 09:00AM	Flux: Next-Generation Resource Management and Scheduling Infrastructure for Exascale Workflow and Resource Challenges	Ahn
09:00AM – 09:30AM	Flux APIs	Herbein
09:30AM – 10:00AM	Using Flux for scheduling MuMMI	Di Natale
10:00AM – 10:30AM	Coffee break	
10:30AM – 12:00PM	Hands-on exercises	Herbein, Ahn and Milroy
12:00PM – 01:30PM	Lunch	
01:30PM – 01:50PM	Using Flux to improve the usability, performance, and software complexity of UQP	Ahn
01:50PM – 02:05PM	Manage internal queue w/ Flux for common workflows	Agelastos (SNL)
02:05PM – 02:25PM	Flux with Kubernetes	Milroy
02:25PM – 03:00PM	Hands-on for advanced topics	Herbein and Ahn
03:00PM – 03:30PM	Coffee break	
03:30PM – 04:00PM	Graph-based resource model and tiered storage support	Ahn and Herbein
04:00PM – 04:30PM	Makeup time for hands-on	Ahn and Herbein

Flux: Next-Generation Resource Management and Scheduling Infrastructure for Exascale Workflow and Resource Challenges

Flux Tutorial at ECP Annual Meeting, Feb 6, 2020

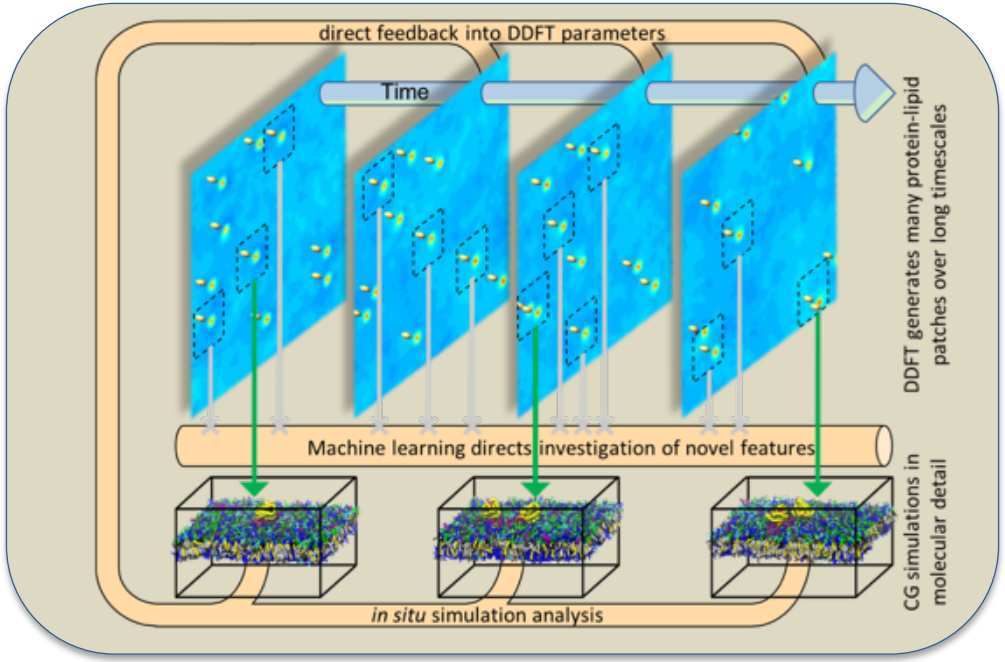
Dong H. Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, Stephen Herbein, Joseph Koning, Tapasya Patki, Thomas R. W. Scogland, Becky Springmeyer, and Michela Taufer

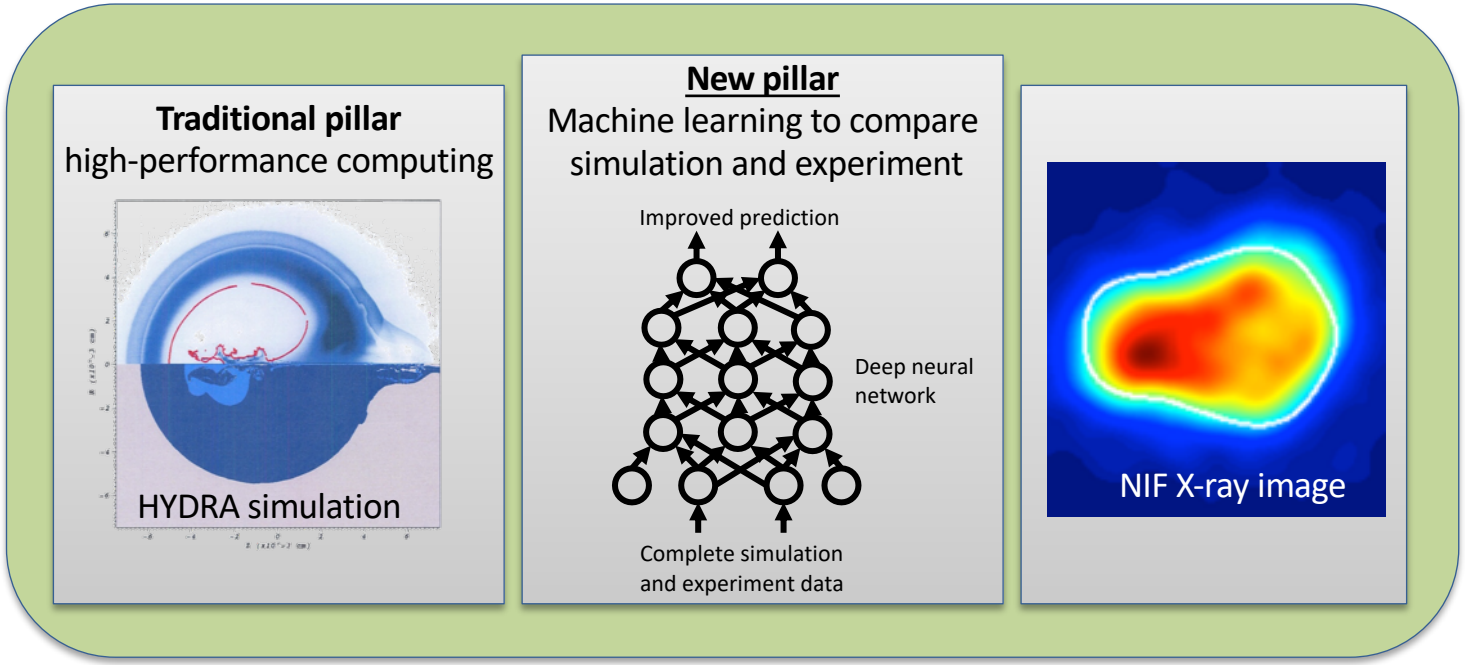
LLNL-PRES- 804364

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

Lawrence Livermore National Laboratory

Workflows on high-end HPC systems are undergoing significant changes.

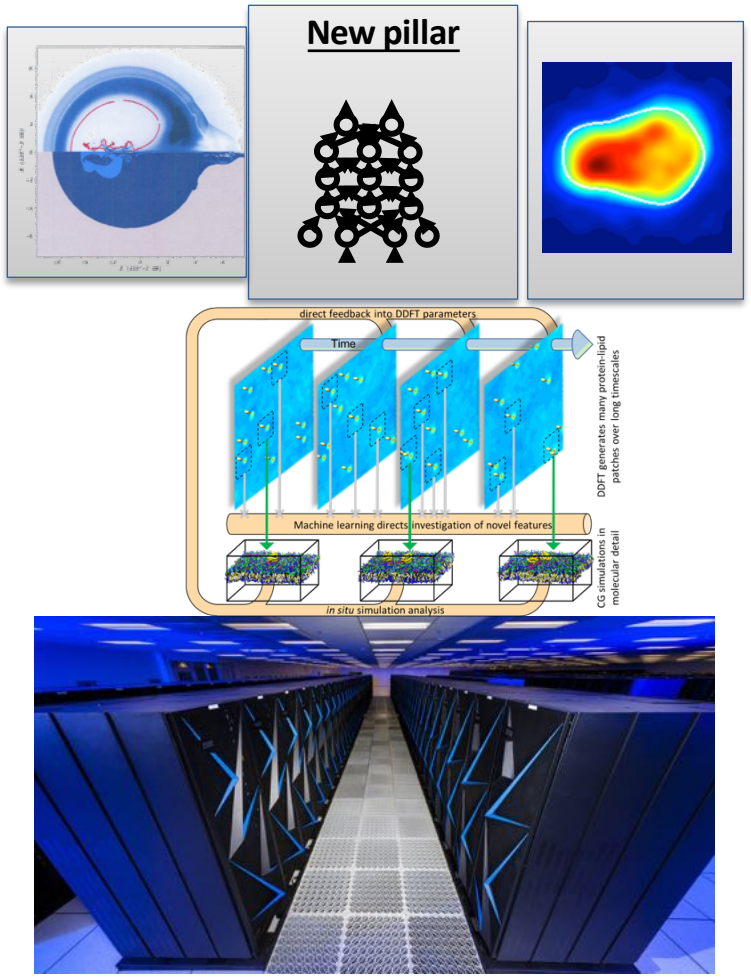




- Cancer Moonshot Pilot2 – co-schedule many elements and ML continuously schedules, de-schedules and executes MD jobs.
- In-situ analytics modules
- ~7,500 jobs simultaneously running

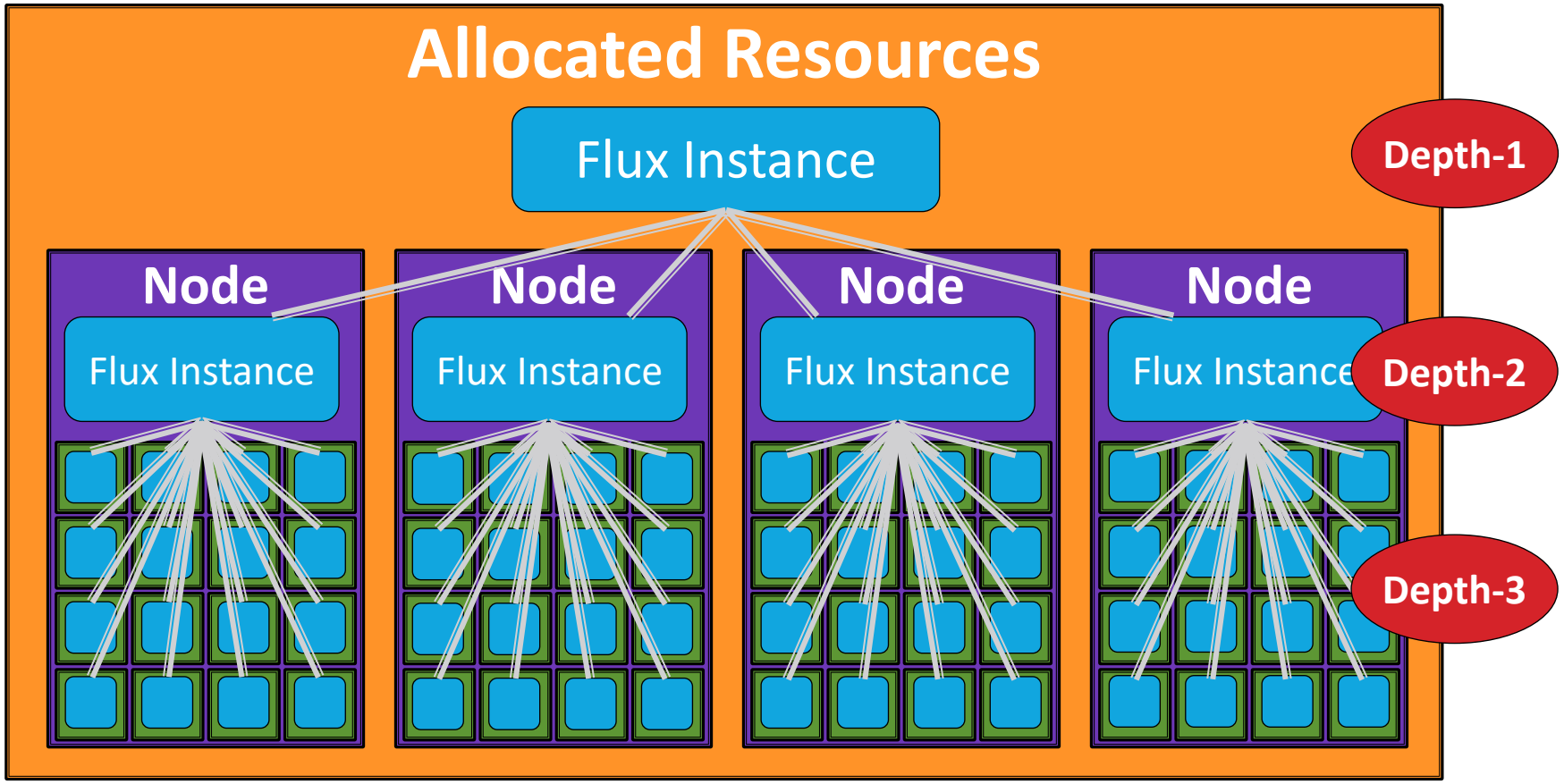
- Machine Learning Strategic Initiative (MLSI) – 1 billion short-running jobs!
- Similar needs for co-scheduling heterogenous components

Key challenges in emerging workflow scheduling include…



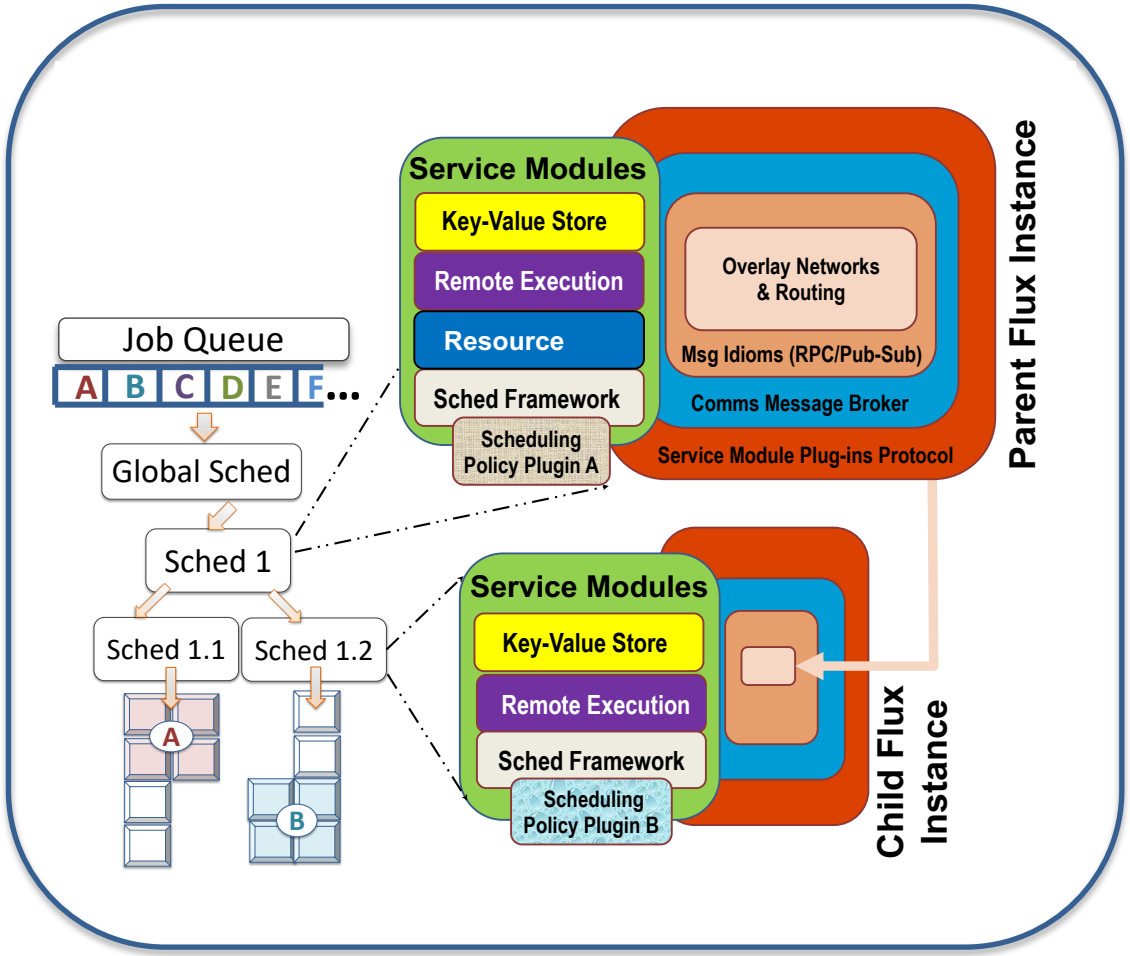
- Co-scheduling challenge
- Job throughput challenge
- Job communication/coordination challenge
- Portability challenge

Flux provides a new scheduling model to meet these challenges.



Our "Fully Hierarchical Scheduling" is designed to cope with many emerging workload challenges.

Flux is specifically designed to embody our fully hierarchical scheduling model.



Techniques

Scheduler Specialization

Scheduler Parallelism

Rich API set

Consistent API set

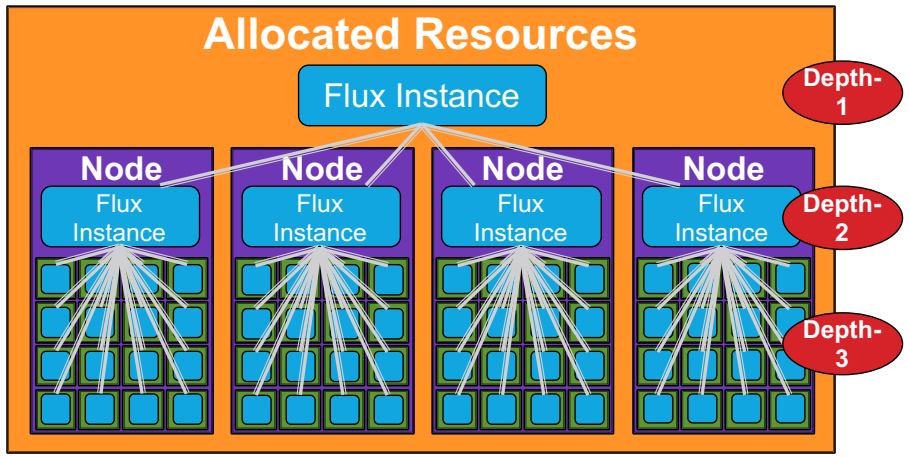
Challenges

- Co-scheduling challenge
- Job throughput challenge
- Job communication/coordination challenge
- Portability challenge

Scheduler specialization solves the co-scheduling challenge.

- Traditional approach
  - A single site-wide policy being enforced for all jobs
  - No support for user-level scheduling with distinct policies
- Flux enables both system- and user-level scheduling under the same common infrastructure.
- Give users the freedom to adapt their scheduler instance to their needs.
  - Instance owners can choose predefined policies different from system-level policies.
    - FCFS/backfilling
    - Scheduling parameters (queue depth, reservation depth etc)
  - Create their own policy plug-in

Scheduler parallelism solves the throughput challenge.



- The centralized model is fundamentally limited.
- Hierarchical design facilitates scheduler parallelism.
- Deepening the scheduler hierarchy allows for higher levels of scheduler parallelism
- Implementation used in our scalability evaluation:
  - Submit each job in the ensemble individually to the root
  - The jobs are distributed automatically across the hierarchy.

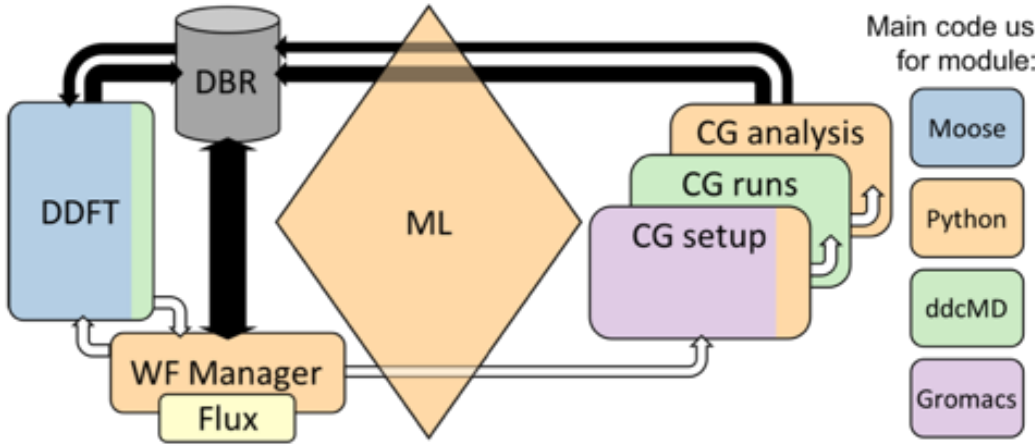
A rich API set enables easy job coordination and communication.

- Jobs in ensemble-based simulations often require close coordination and communication with the scheduler as well as among them.
  - Traditional CLI-based approach is too slow and cumbersome.
  - Ad hoc approaches (e.g., many empty files) can lead to many side-effects.
- Flux provides well-known communication primitives.
  - Pub/sub, request-reply, and send-recv patterns
- High-level services
  - Key-value store (KVS) API
  - Job status/control (JSC) API
  - KZ stdout/stderr stream API

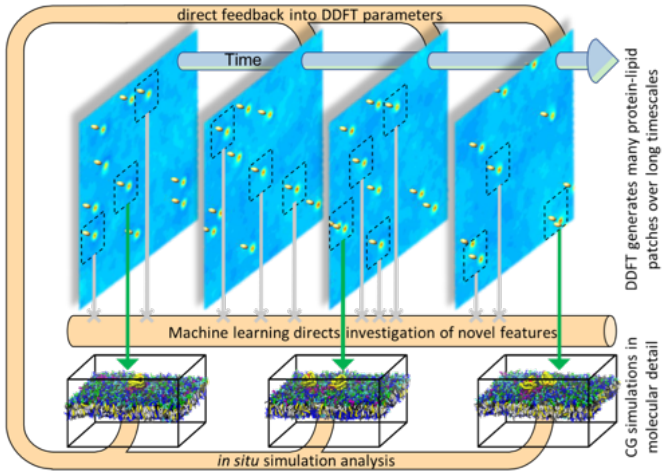
A consistent API set facilitates high portability.

- Flux's APIs are consistent across different platforms
- Effort for porting and optimizing Flux itself for a new environment is small
  - Linux
  - Require the lower-level system to provide the Process Management Interface (PMI)

Scheduler specialization addresses co-scheduling challenges in Cancer Moonshot Pilot2 on Sierra



Pilot2 workflow architecture



- The machine-learning module evaluates the top *n* candidate patches for MD simulations.
- Integrate Flux into Maestro workflow manager to start and stop jobs accordingly
- Maestro adapter to Flux
  - Specialize the policy to be non-node-exclusive scheduling for complex co-scheduling
  - At least 5 different logically separate jobs, each with CPU and/or GPU requirements on every node
  - Handle the volume of jobs using simple hierarchical scheduling

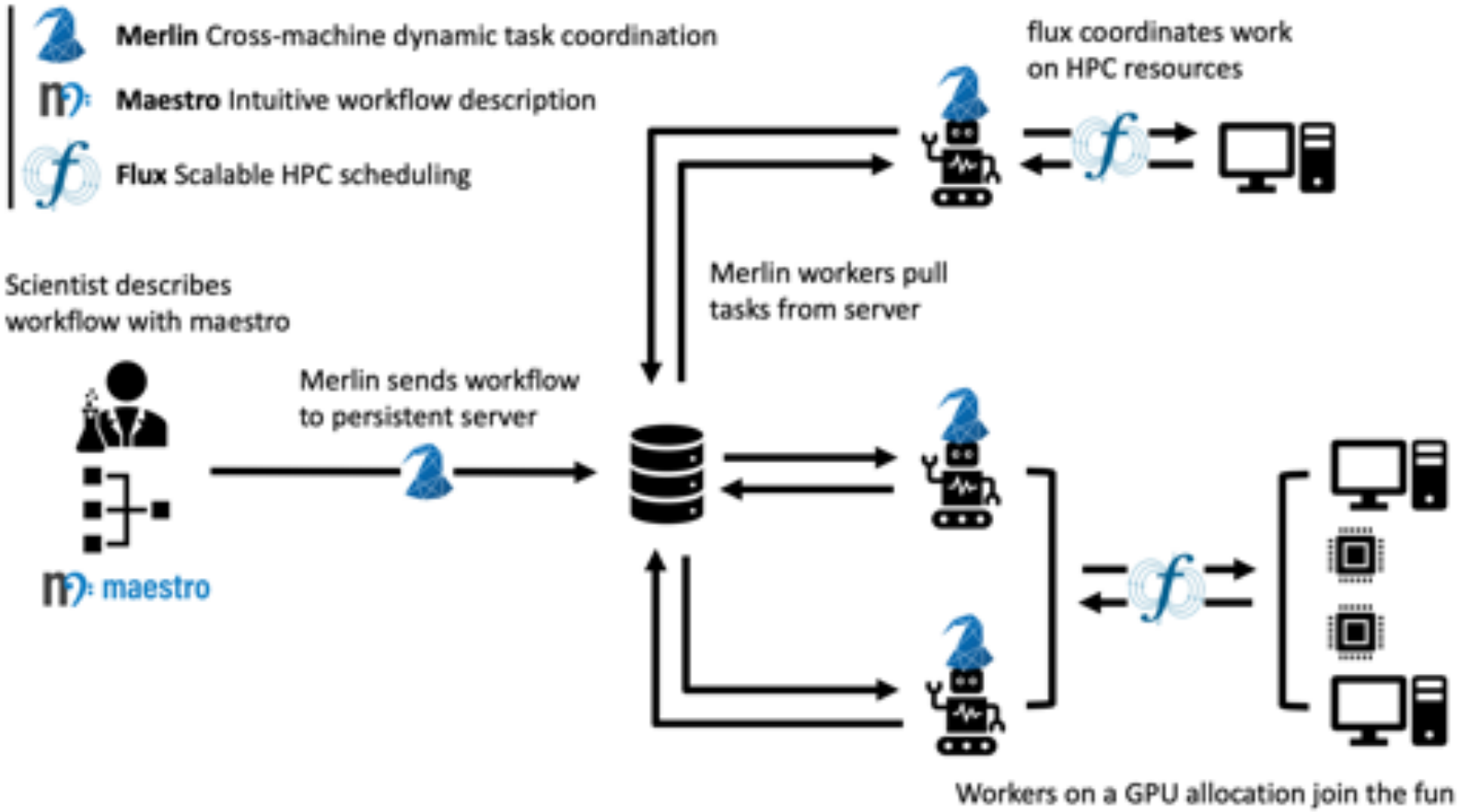
Franc Di Natale will present this SC19 best paper award winning work at 9:30AM today

Co-design with Flux is uniquely improving LLNL's UQ Pipeline tool in 3 unprecedented ways

- Enable UQP to perform quick exploratory experiments and to generate a large ensemble from it.
  - — Improve usability and performance.
- Simplify the software architecture.
  - — Reduce software complexity and increase portability.
- Position the ensemble manager for extremely high throughput workflows.
  - — Increase scalability and exascale readiness.

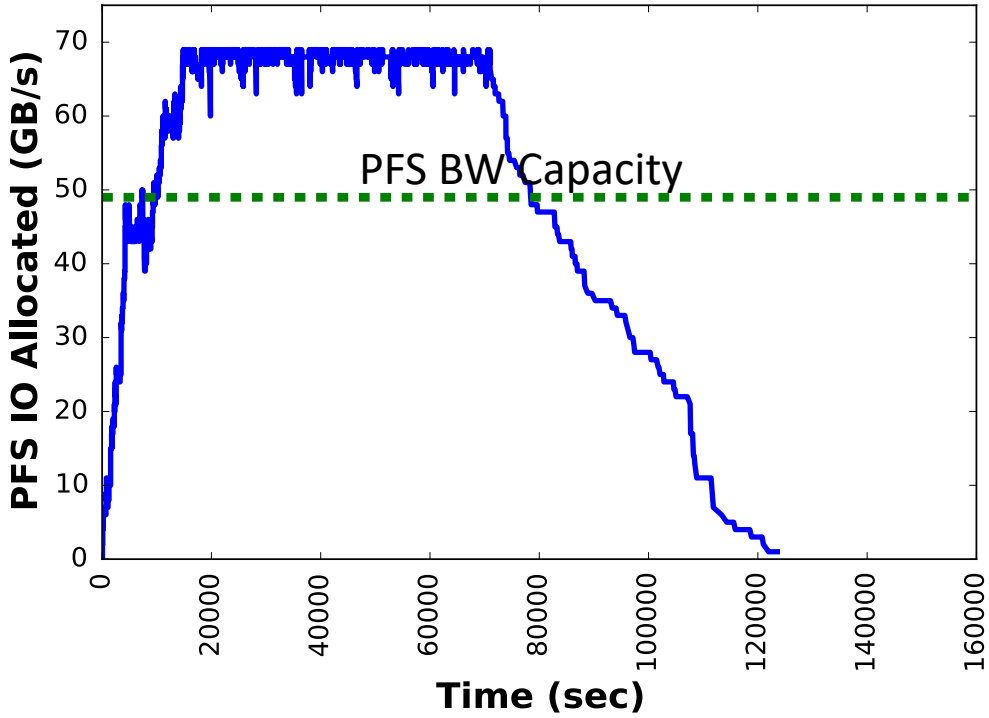
We will present more details
at 1:30PM today

MSLI team successfully ran 100Million short-running jobs thanks to Flux.



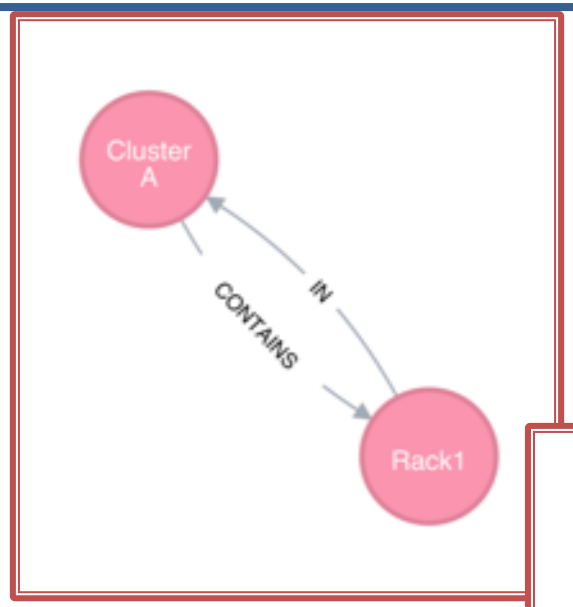
The changes in resource types are equally challenging.

- Problems are not just confined to the workload/workflow challenge.
- Resource types and their relationships are also becoming increasingly complex.
- Much beyond compute nodes and cores…
  - GPGPUs
  - Burst buffers
  - I/O and network bandwidth
  - Network locality
  - Power
- Will be worse for exascale computing.

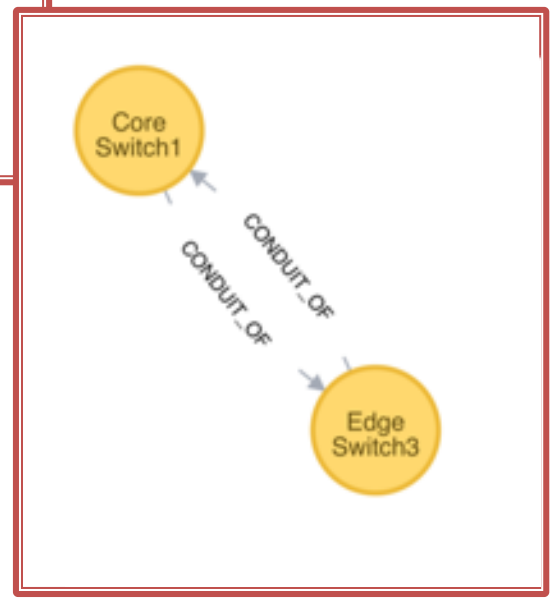


Flux uses a graph-based resource data model to represent schedulable resources and their relationships.

- A graph consists of a set of vertices and edges
  - Vertex: a resource
  - Edge: a relationship between two resources
- Highly composable to support a graph with arbitrary complexity
- The scheduler remains to be a highly generic graph code.



Containment subsystem



Network connectivity subsystem

We will present the graph-based resource modeling work at 3:30PM today

Flux Plan

Flux Tutorial at ECP Annual Meeting, Feb 6, 2020

Dong H. Ahn

LLNL-PRES- 804364

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

Lawrence Livermore National Laboratory

High-Level Update

- Two main drivers for last and this fiscal year
  - Deploy Flux natively on a moderate-size TOSS4 system
  - Demonstrate key exascale-computing capabilities in preparation for CORAL2 El Capitan
- ExaWorks: new ECP proposal

Recent highlights: we completed the planned development of our new execution system

- Goal is to implement our execution system that meets the requirements imposed by multi-user Flux and security while making its feature set on-par with our old system called wreck.
- We have been more or less on track:
  - Preliminary job shell (7/5/2019)
  - MPI Hello world (7/31/2019)
  - Shell plugin (08/16/2019)
  - Flux run prototype in review (8/16/2019)
  - I/O improvements (08/30/2019)
  - Task exit handling (09/13/2019)
  - Flux simulator prototype in review (11/26/2019)
- Some of our workflow customers are kicking the tires

Recent highlights: we completed the planned integration of our graph-based scheduler

- Goal is to make the feature set of our graph-based scheduler on-par with the old ad hoc system and integrate it with the new execution system
- We have been more or less on track:
  - Introduced qmanager to tie the new job-manager with the graph scheduler (level 0 integration) (7/10/2019)
  - Added HPC queuing policies (EASY/HYBRID/CONSERVATIVE) (7/26/2019)
  - Added satisfiability checks (8/8/2019)
  - Added performance variation-aware scheduling support (9/30/2019)
  - Added JGF reader for resiliency and Kubernetes (9/9/2019)
  - Resource graph update using JGF for resiliency (10/4 and 12/2/2019)
  - New staff training and prototype for Banking and accounting (12/31/2019)
- Some of our workflow customers are kicking the tires

Recent highlights: we completed a relatively low-cost design exploration

- System instance
  - Identified the need for a relatively low-cost design exploration
  - Added remote jobshell (a.k.a., brokerless compute nodes) prototype (12/12/2019)
  - Determined that this does not meet an important requirement (coordinated migration of the system instance)
  - In the process of penciling in a more detailed design (with brokers on compute nodes)
  - Added a few additional weeks to our schedule
- Multiuser capability
  - Added AddressSanitizer support to prepare for multi-user job launch
  - Will help write correct privileged code next quarter

Recent highlights: completed lots of some of less-appreciated efforts (harden, refactor and cleanup)

- People often underestimate the level of effort required to harden, refactor and cleanup code for production-quality software
- http://flux-framework.org/news

ExaWorks: a new component-based workflow toolbox R&D project proposed to ECP

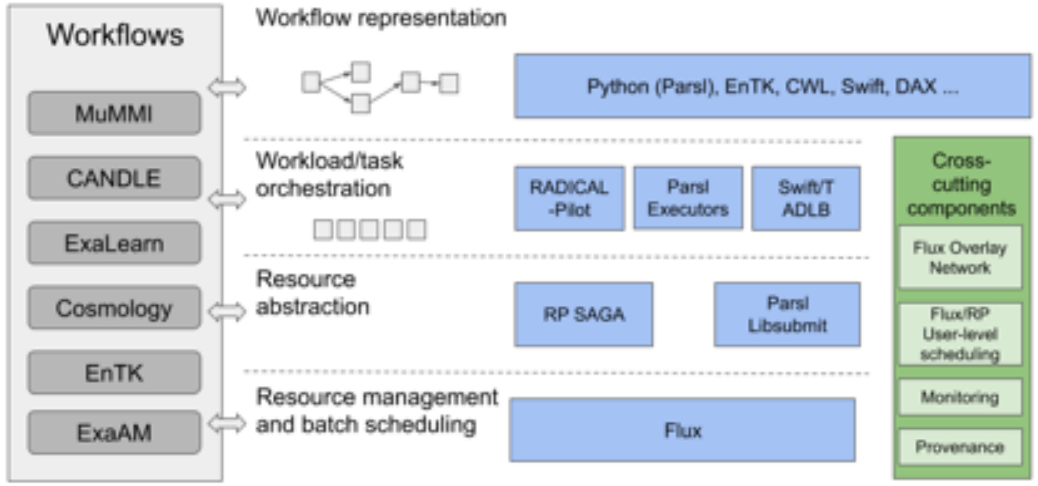


Figure 1: ExaWorks workflow components showing the four component layers (workflow representation, workload orchestration, resource abstraction, and resource management) and several cross-cutting components.

Science Driver	Scheduling	Performance	Communication	Challenges
Precision Medicine for Cancer	Macro scale coupled to many MD models; co-scheduling CPU/GPU tasks	36K concurrent tasks	Filesystem-based data interchange	Task coordination while minimizing filesystem utilization
CANDLE	Co-scheduling of single core and GPU tasks	Billions of short-duration tasks	coordination using properties in files	Throughput requirements at extreme scales
ExaSky	Container scheduling and long-tail runtimes	Simulating millions of sensors for thousands of visits	Cosmology databases and file formats	Monitoring resource usage; coupling simulation and analysis
ExaLearn	Co-scheduling of inference, simulation, and training tasks	Hundreds of thousands of short-duration tasks, thousands of simulation tasks, few training tasks	Communication between tasks, data services, and workflow	Extreme throughput, task heterogeneity, long running tasks with flexible communication
ASC	Large long-running tasks and short-running co-scheduled tasks integrated with ML	thousands of 3D runs, millions of 2D runs	Filesystem-based data interchange	Coupling 10's of apps to explore high dimensional parameter spaces with provenance and reproducibility
ExaAM	Ensembles of multi-scale models	Coupled models sharing volumetric field data	filesystem-based data interchange	Highly-variable task runtimes with co-scheduling

Table 1: Summary of exascale workflow requirements

Flux APIs for Scientific Workflows

Anthony Agelastos, Dong H. Ahn, Frank Di Natale,
Stephen Herbein, Dan Milroy, Tapasya Patki

November 8, 2019

LLNL-PRES- 804364

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

Lawrence Livermore National Laboratory

What is Flux?

- New Resource and Job Management Software (RJMS) developed at LLNL
- A way to manage remote resources and execute tasks on them

What is Flux?

- New Resource and Job Management Software (RJMS) developed at LLNL
- A way to manage **remote resources** and execute tasks on them

amazon web services™

Google Cloud

flickr: dannychamoro

What is Flux?

- New Resource and Job Management Software (RJMS) developed at LLNL
- A way to manage remote resources and **execute tasks** on them

Why Flux?

- Extensibility
  - Open source
  - Modular design with support for user plugins
- Scalability
  - Designed from the ground up for exascale and beyond
  - Already tested at 1000s of nodes & millions of jobs
- Usability
  - C, Lua, and Python bindings that expose 100% of Flux's functionality
  - Can be used as a single-user tool or a system scheduler
- Portability
  - Optimized for HPC and runs in Cloud and Grid settings too
  - Runs on any set of Linux machines: only requires a list of IP addresses or PMI

Portability: Running Flux

- Already installed on TOSS systems (including Sierra)
  - `spack install flux-sched` for everywhere else
- Flux can run anywhere that MPI can run, (via PMI – Process Management Interface)
  - Inside a resource allocation from: itself (hierarchical Flux), Slurm, Moab, PBS, LSF, etc
  - `flux start` **OR** `srun flux start` **OR** `flux bootstrap`
- Flux can run anywhere that supports TCP and you have the IP addresses
  - FLUX_CONF_DIR=/etc/flux `flux broker -Sboot.method=config`
  - `/etc/flux/conf.d/boot.toml:`

```
[bootstrap]
default_port = 8050
default_bind = "tcp://en0:%p”
default_connect = "tcp://e%h:%p”
hosts = [ { host="fluke0" }, { host = "fluke1" }, { host = "fluke2" } ]
```

Why Flux?

- Extensibility
  - Open source
  - Modular design with support for user plugins
- Scalability
  - Designed from the ground up for exascale and beyond
  - Already tested at 1000s of nodes & millions of jobs
- Usability
  - C, Lua, and Python bindings that expose 100% of Flux's functionality
  - Can be used as a single-user tool or a system scheduler
- Portability
  - Optimized for HPC and runs in Cloud and Grid settings too
  - Runs on any set of Linux machines: only requires a list of IP addresses or PMI

Usability: Submitting a Batch Job

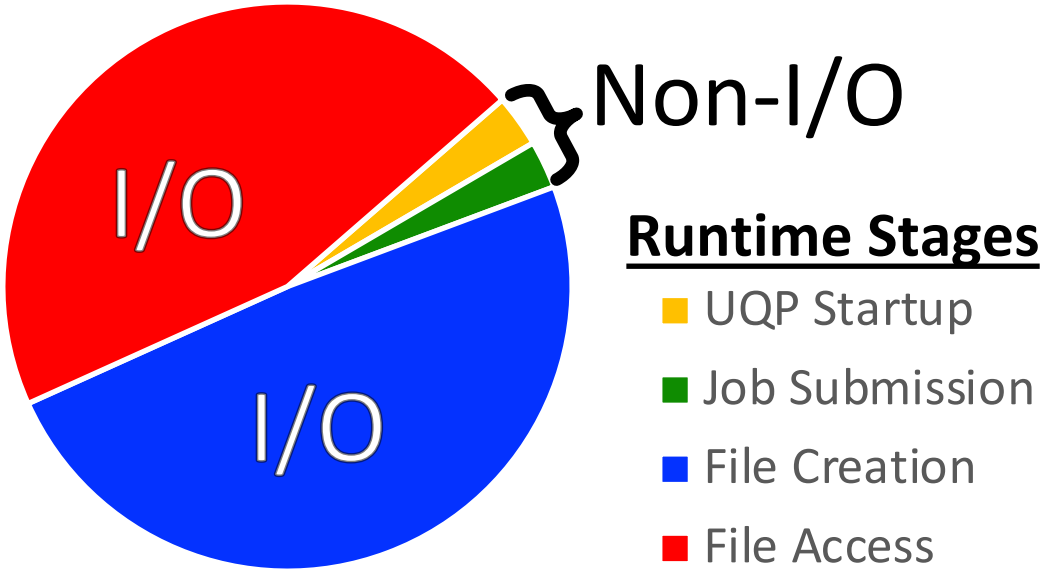
- Slurm
  - `sbatch –N2 –n4 –t 2:00 sleep 120`
- Flux CLI
  - `flux mini submit –N2 –n4 –t 2m sleep 120`
- Flux API:

```
import json, flux, job
from flux.job import JobspecV1

f = flux.Flux()
j = JobspecV1.from_command(command=["sleep", "120"],
                           num_nodes=2,
                           num_tasks=4)
j.set_duration(120)
resp = flux.job.submit(f, j)
```

Usability: Tracking Job Status

- CLI: slow, non-programmatic, inconvenient to parse
  - `watch squeue`
  - `watch flux job list`
- Tracking via the filesystem
  - `date > $JOBID.start; srun myApp; date > $JOBID.stop`

```
→ quota -vf ~/quota.conf
Disk quotas for herbein1:
Filesystem     used   quota  limit   files
/p/lscratchrza 760.3G n/a    n/a     8.6M
```



Usability: Tracking Job Status

- CLI: slow, non-programmatic, inconvenient to parse
  - `watch squeue`
  - `watch flux job list`
- Tracking via the filesystem
  - `date > $JOBID.start; srun myApp; date > $JOBID.stop`
- Job tracking via Flux:

```
# wait for next job to complete
fut = flux.job.wait(f)

# get completed job's info
(jobid, success, errstr) =
      flux.job.wait_get_status(fut)
```

Usability: Tracking Job Status

- CLI: slow, non-programmatic, inconvenient to parse
  - `watch squeue`
  - `watch flux job list`
- Tracking via the filesystem
  - `date > $JOBID.start; srun myApp; date > $JOBID.stop`
- Job tracking via Flux:

```
# wait for next job to complete
fut = flux.job.wait(f)

# get completed job's info
(jobid, success, errstr) =
      flux.job.wait_get_status(fut)
```

- Reactive, event-driven tracking via Flux:

```
f.event_subscribe("job-state")
f.msg_watcher_create(job_state_cb, "job-state"
).start()
f.reactor_run(f.get_reactor(), 0)

def job_state_cb (f, typemask, msg, arg):
    for jobid, state in msg.payload['transitions']:
        print("job", jobid, "changed to ", state)
```

Why Flux?

- Extensibility
  - Open source
  - Modular design with support for user plugins
- Scalability
  - Designed from the ground up for exascale and beyond
  - Already tested at 1000s of nodes & millions of jobs
- Usability
  - C, Lua, and Python bindings that expose 100% of Flux's functionality
  - Can be used as a single-user tool or a system scheduler
- Portability
  - Optimized for HPC and runs in Cloud and Grid settings too
  - Runs on any set of Linux machines: only requires a list of IP addresses or PMI

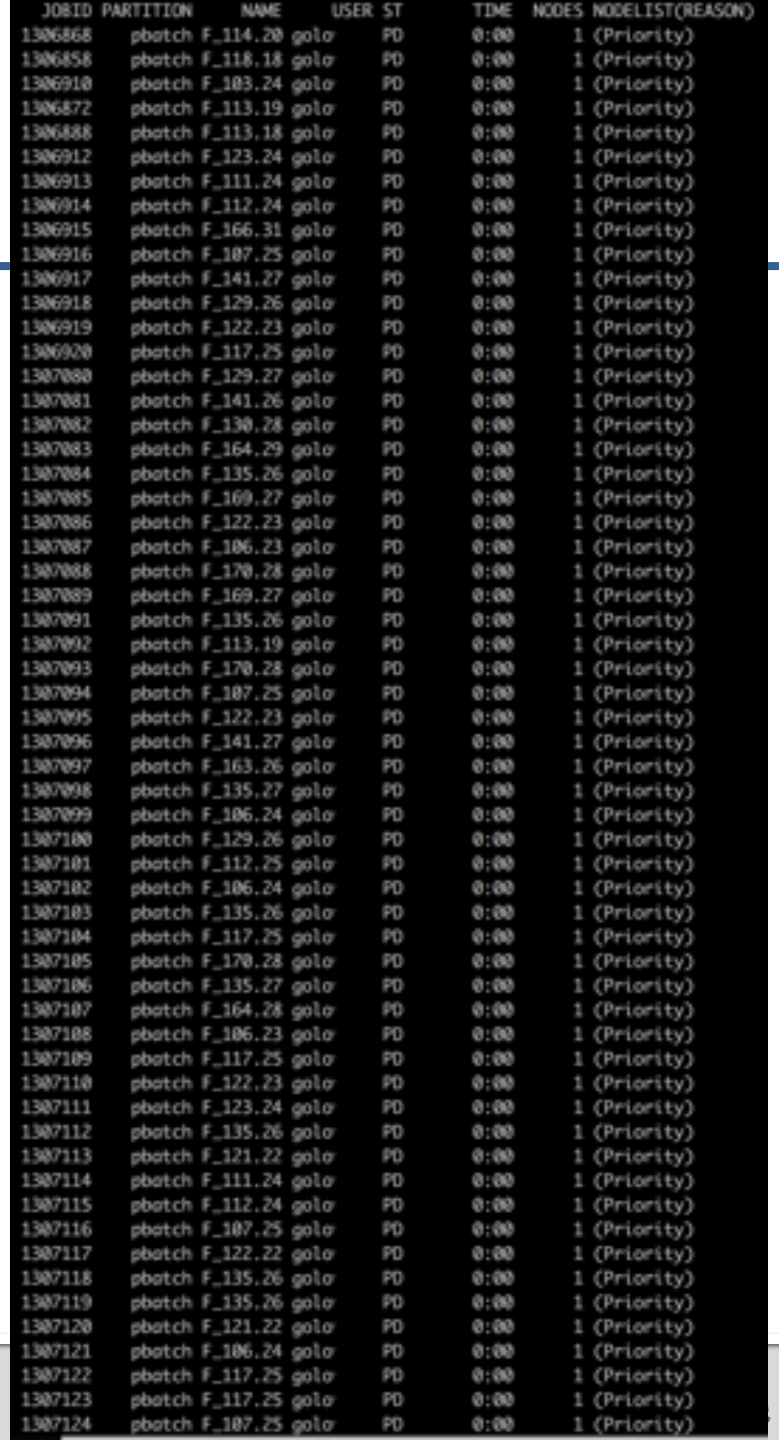
Scalability: Running Many Jobs

- Slurm
  - `find ./ -exec sbatch –N1 tar –cf {}.tgz {}\;`
    - Slow: requires acquiring a lock in Slurm, can timeout causing failures
    - Inefficient: uses 1 node for each task

Subject: Good Neighbor Policy

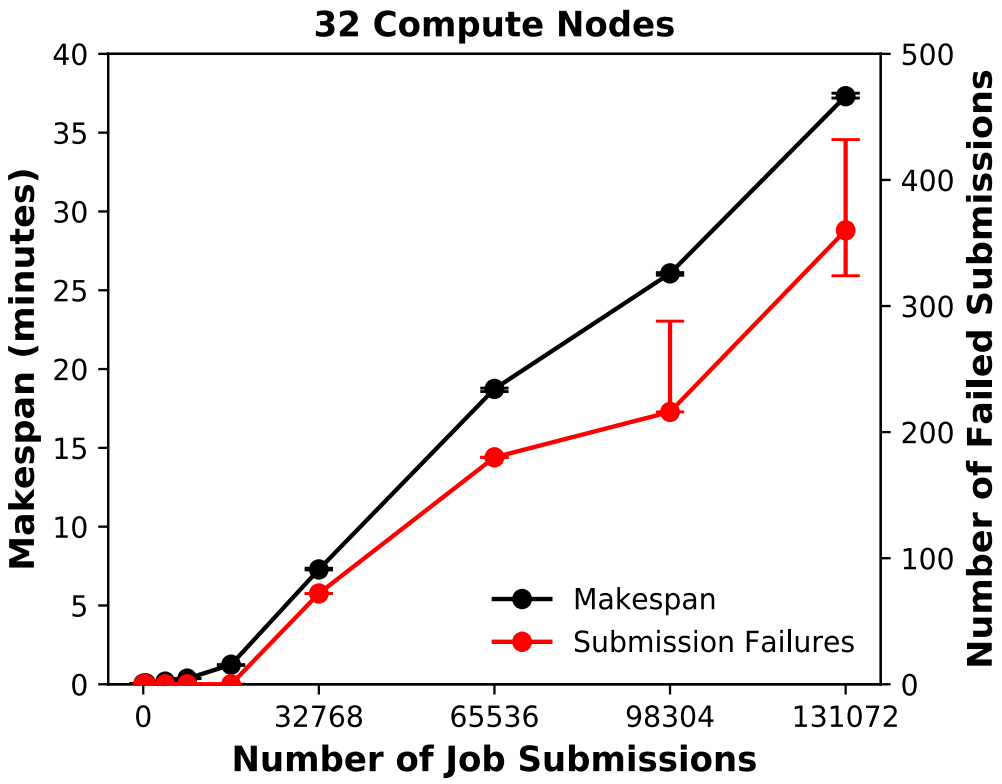
You currently have 271 jobs in the batch system on lamoab.

The good neighbor policy is that users keep their maximum submitted job count at a maximum of 200 or less. Please try to restrict yourself to this limit in the future. Thank you.



Scalability: Running Many Jobs

- Slurm
  - `find ./ -exec sbatch –N1 tar –cf {}.tgz {}\;`
    - Slow: requires acquiring a lock in Slurm, can timeout causing failures
    - Inefficient: uses 1 node for each task
  - `find ./ -exec srun –n1 tar –cf {}.tgz {}\;`
    - Slow: spawns a process for every submission
    - Inefficient: is not a true scheduler – can overlap tasks on cores



Scalability: Running Many Jobs

- Slurm
  - `find ./ -exec sbatch –N1 tar –cf {}.tgz {}\;`
    - Slow: requires acquiring a lock in Slurm, can timeout causing failures
    - Inefficient: uses 1 node for each task
  - `find ./ -exec srun –n1 tar –cf {}.tgz {}\;`
    - Slow: spawns a process for every submission
    - Inefficient: is not a true scheduler – can overlap tasks on cores
- Flux API:

```
flux start my_jobs.py
----------------------------
import flux, flux.job
from flux.job import JobspecV1

h = flux.Flux()
for f in os.listdir(‘.’):
    command = [“tar”, “-cf”, ”{}.tgz”.format(f), f]
    flux.job.submit(h, JobspecV1.from_command(command))
```

Scalability: Running Many Heterogeneous Jobs

- Slurm
  - No support for heterogeneous job steps in versions before 17.11
  - Limited support in versions after 17.11

https://slurm.schedmd.com/heterogeneous_jobs.html#limitations

Limitations

The backfill scheduler has limiations in how it tracks usage of CPUs and memory in the future. This typically requires the backfill scheduler be able to allocate each component of a heterogeneous job on a different node in order to begin its resource allocation, even if multiple compoents of the job do actually get allocated resources on the same node.

In a federation of clusters, a heterogeneous job will execute entirely on the cluster from which the job is submitted. The heterogeneous job will not be eligible to migrate between clusters or to have different components of the job execute on different clusters in the federation.

Job arrays of heterogeneous jobs are not supported.

The srun command's --no-allocate option is not supported for heterogeneous jobs.

Only one job step per heterogeneous job component can be launched by a single srun command (e.g. "srun --pack-group=0 alpha : --pack-group=0 beta" is not supported).

The sattach command can only be used to attach to a single component of a heterogeneous job at a time.

Heterogeneous jobs are only scheduled by the backfill scheduler plugin. The more frequently executed scheduling logic only starts jobs on a first-in first-out (FIFO) basis and lacks logic for concurrently scheduling all components of a heterogeneous job.

Heterogeneous jobs are not supported with Slurm's select/serial plugin.

Heterogeneous jobs are not supported on Cray ALPS systems.

Heterogeneous jobs are not supported on IBM PE systems.

Slurm's PERL APIs currently do not support heterogeneous jobs.

The srun --multi-prog option can not be used to span more than one heterogeneous job component.

The srun --open-mode option is by default set to "append".

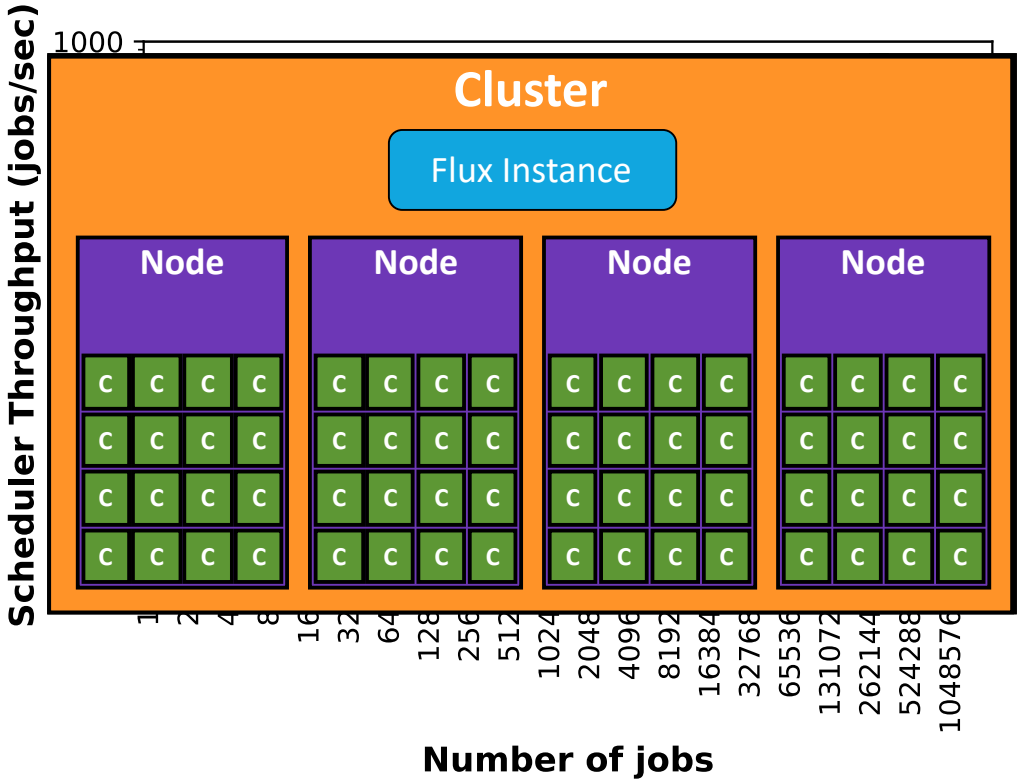
Scalability: Running Many Heterogeneous Jobs

- Slurm
  - No support for heterogeneous job steps in versions before 17.11
  - Limited support in versions after 17.11
- Flux API

```
flux start my_jobs.py
-----------------------------
import flux, flux.job
from flux.job import JobspecV1

h = flux.Flux()
for f in os.listdir('.'):
    command = ["tar", "-I", "pigz", "-cf", "{}.tgz".format(f), f]
    ncores = os.path.getsize(f) / 1024**3
    flux.job.submit(h, JobspecV1.from_command(command, cores_per_task=ncores))
```

Scalability: Running Millions of Jobs

- Single Flux Instance
  - `flux start my_workflow.py`
- Statically Partitioned Flux Instances
  -
```
for x in $(seq 1 $num_nodes); do
        flux submit -N1 flux start  \
             my_workflow_$x.py
done
```
- Flux Hierarchy
  -
```
flux-tree –N $num_nodes \
        –T ${num_nodes}  \
        -J $num_jobs -- jobspec.yaml
```
  -
```
flux-tree –N $num_nodes \
        –T ${num_nodes}x${cores_per_node} \
        -J $num_jobs -- jobspec.yaml
```

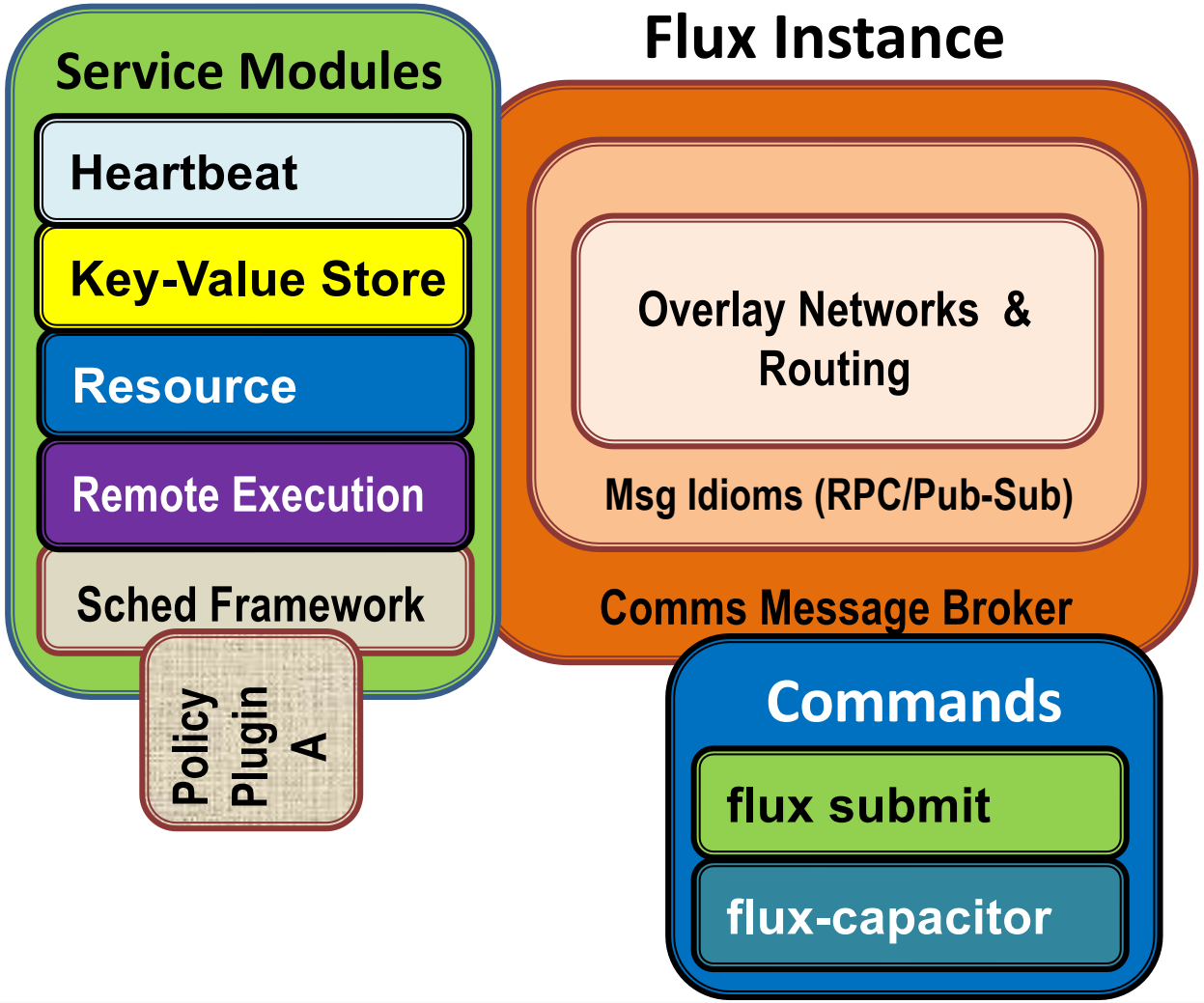


Why Flux?

- Extensibility
  - Open source
  - Modular design with support for user plugins
- Scalability
  - Designed from the ground up for exascale and beyond
  - Already tested at 1000s of nodes & millions of jobs
- Usability
  - C, Lua, and Python bindings that expose 100% of Flux's functionality
  - Can be used as a single-user tool or a system scheduler
- Portability
  - Optimized for HPC and runs in Cloud and Grid settings too
  - Runs on any set of Linux machines: only requires a list of IP addresses or PMI

Extensibility: Modular Design

- At the core of Flux is an overlay network
  - Built on top of ZeroMQ
  - Supports RPCs, Pub/Sub, Push/Pull, etc
- Modules provide extended functionality (i.e., services)
  - User-built modules are loadable too
  - Some modules also support plugins
- External tools and commands can access services
  - User authentication and roles supported



Extensibility: Creating Your Own Module

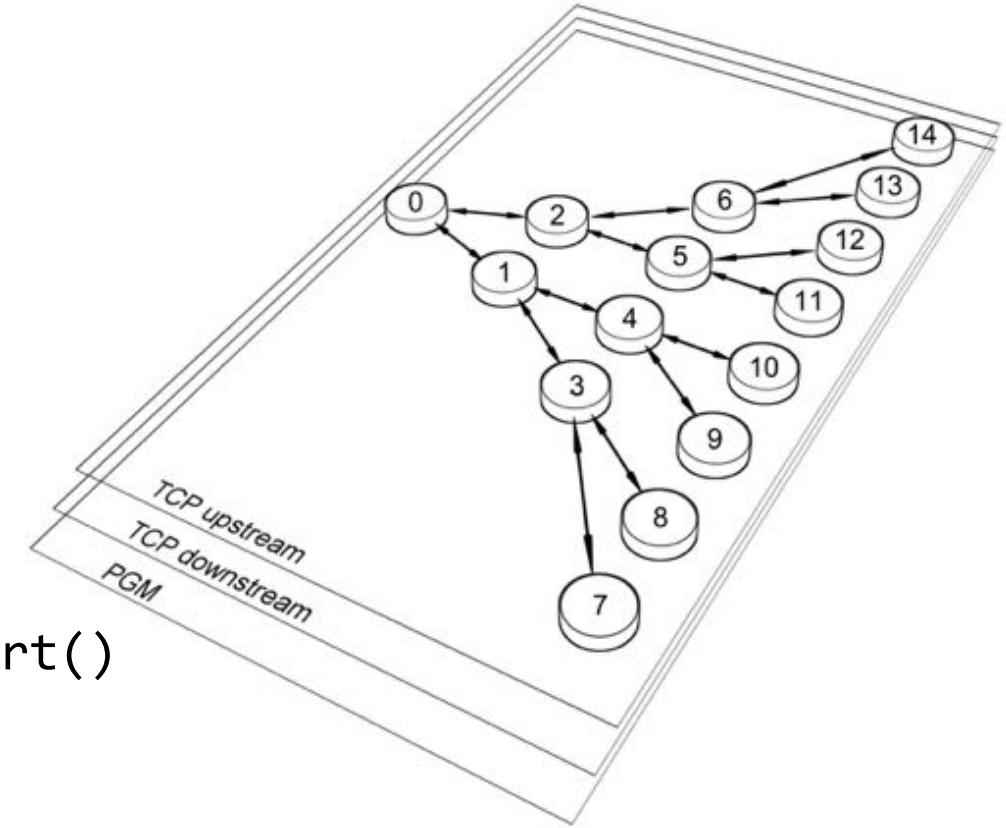
- Register a new service “pymod.new_job” that ingests jobs and responds with a Job ID

```
import itertools, json, flux

def handle_new_job(f, typemask, message, arg):
    job_queue, job_ids = arg
    job_queue.append(message.payload)
    response = {‘jobid’ : job_ids.next()}
    f.respond(message, 0, response)

def mod_main(f, *argv):
    f.msg_watcher_create(handle_new_job, topic="pymod.new_job",
                         args=([], itertools.count(0))).start()

    f.reactor_run(f.get_reactor(), 0)
```

- Load using `flux module load pymod --module=path/to/file.py`

Extensibility: Flux's Communication Overlay

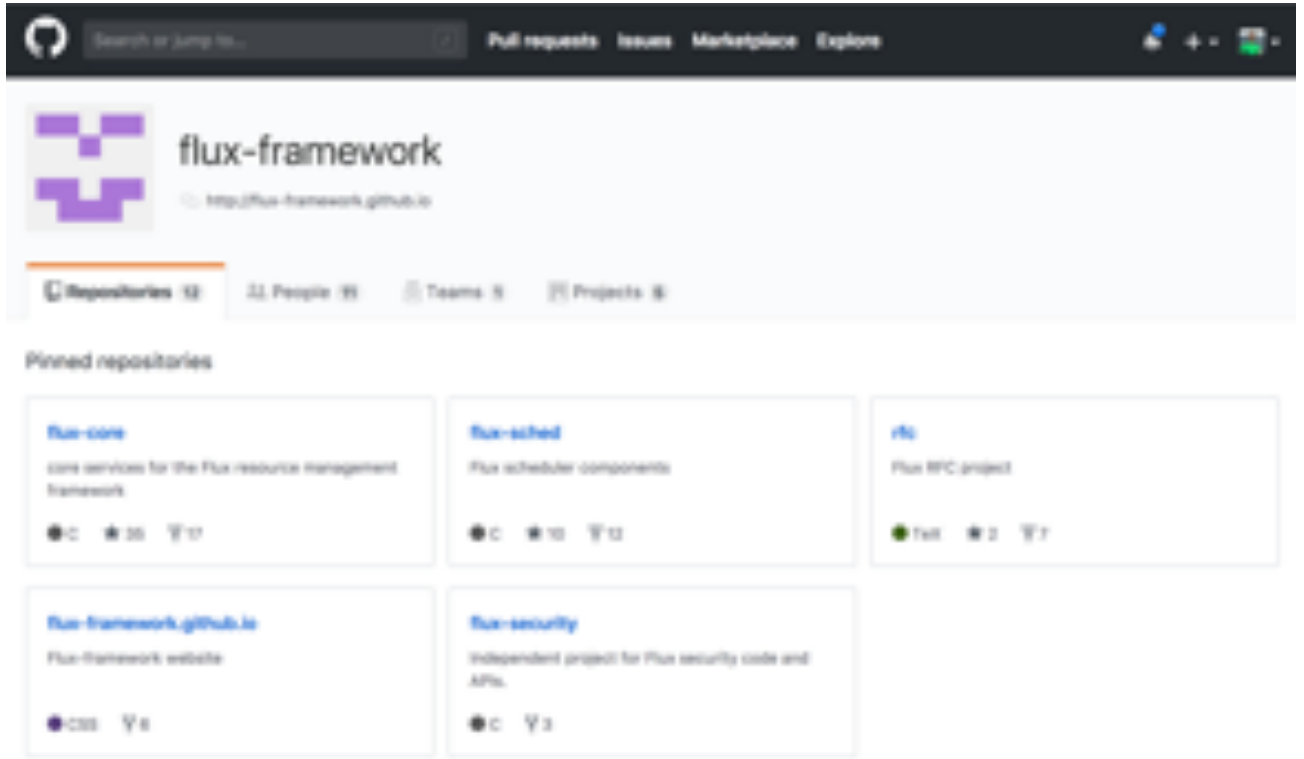
- Connect to a running flux instance
  - `f = flux.Flux()`
- Send an RPC to a service and receive a response
  - `resp = f.rpc("pymod.new_job", payload)`
    `jobid =resp.get()['jobid']`
- Subscribe to and publish an event
  - `f.event_subscribe("node_down")`
    `f.msg_watcher_create(node_down_cb,`
    `topic="node_down").start()`
  - `f.event_send("node_down")`



Extensibility: Open Source

- Flux-Framework code is available on GitHub
- Most project discussions happen in GitHub issues
- PRs and collaboration welcome!

GitHub



Thank You!

A Massively Parallel Infrastructure for Adaptive Multiscale Simulations:

Modeling RAS Initiation Pathway for Cancer

ECP Annual Meeting 2020

February 6, 2020

Francesco Di Natale
Computer Scientist
JDACS4C Pilot 2

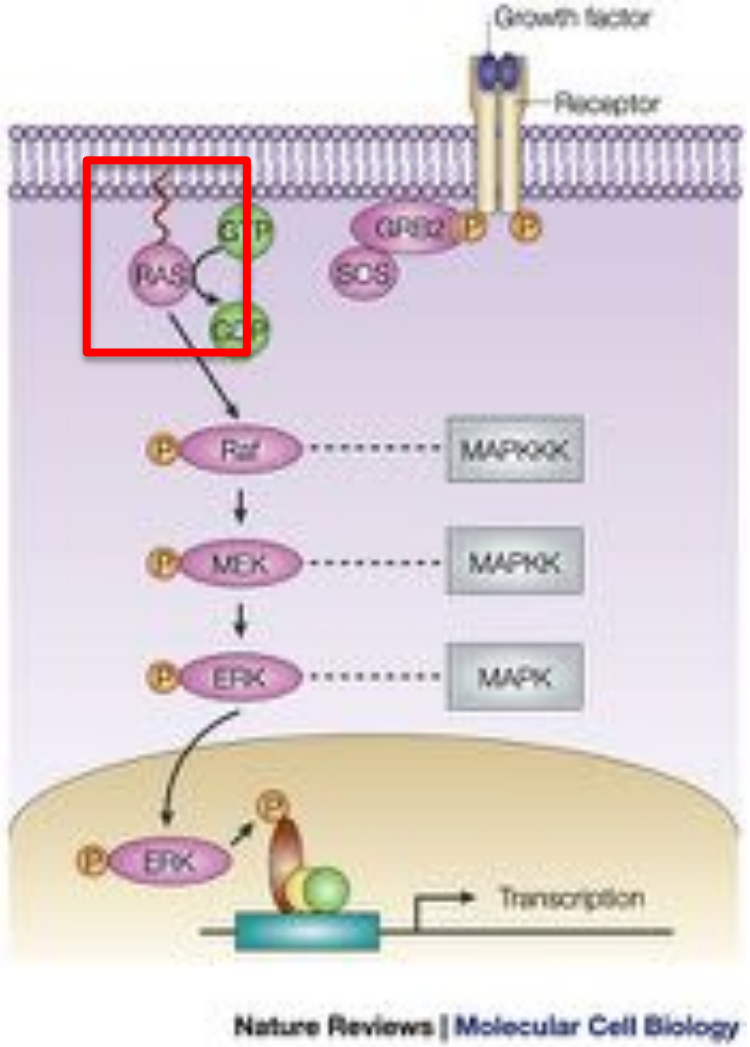
LLNL-PRES- 804364

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

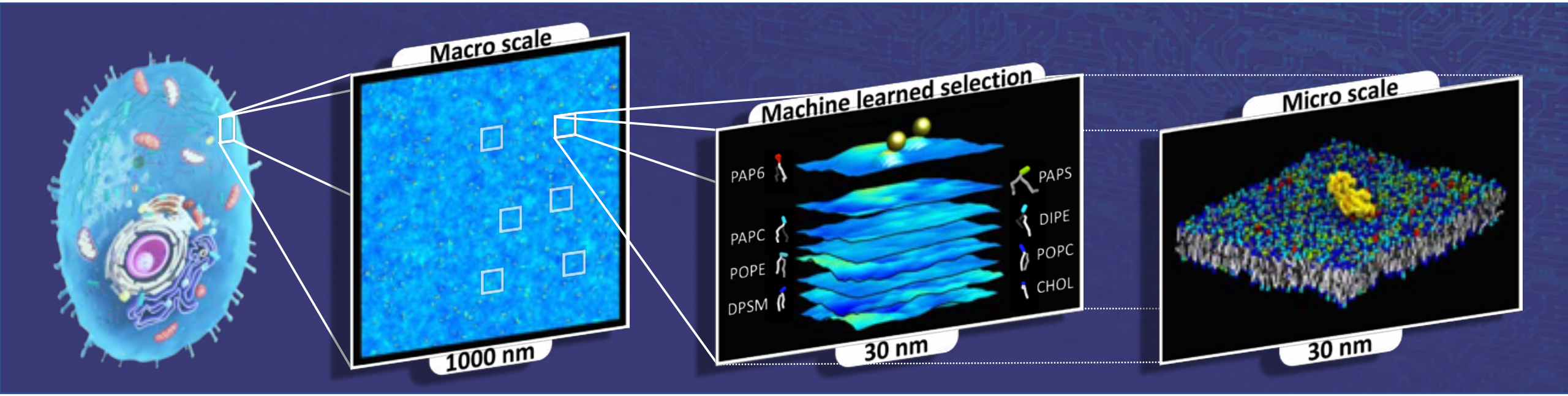
Lawrence Livermore National Laboratory

Resolving RAS dynamics is paramount to understanding cancer

- Mutated RAS protein is involved in 30% of all human cancers
  - 95% of pancreatic; 45% of colorectal; 35% of lung;
- RAS sits on the cell membrane and acts as an “on switch” for cell growth
- Until recently there were no known inhibitors and RAS is often considered undruggable
- Activation and aggregation are dependent on the local membrane
- Multiscale simulations are required to understand membrane dynamics and RAS-membrane interactions
  - micro: molecular level detail of the biological processes
  - macro: experimentally relevant length- and time-scales



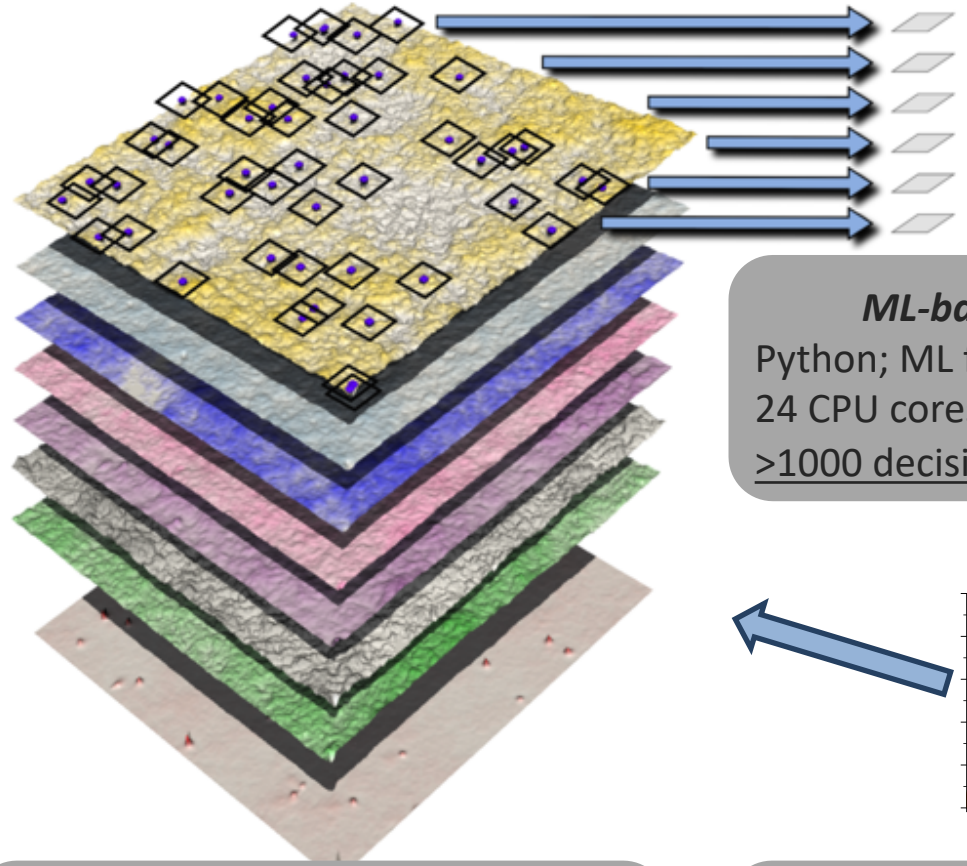
MuMMI implements a complex workflow to enable a new genre of multiscale simulation for cancer research



Multiscale Machine-Learned Modeling Infrastructure (MuMMI)

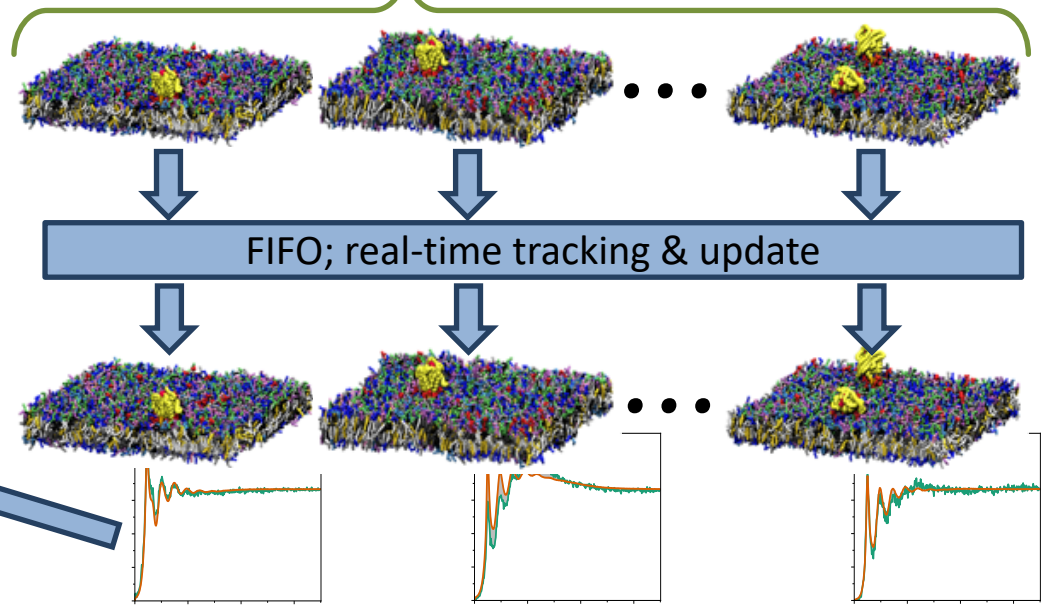
- Novel framework coupling multiple scales using a hypothesis driven selection process.

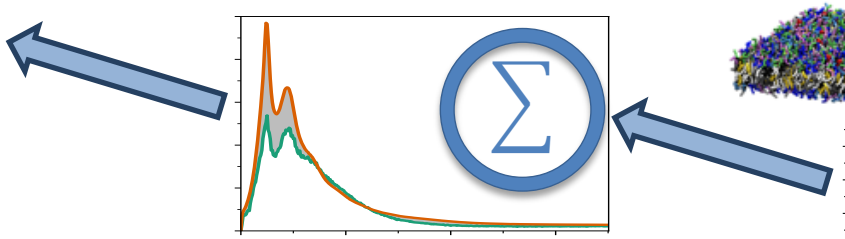
MuMMI implements a complex and dynamic workflow



ML-based Selection
Python; ML frameworks; FAISS
24 CPU cores
>1000 decisions per minute

CG Setup
Python; Custom; GROMACS
24 CPU cores each
1.5 hr each





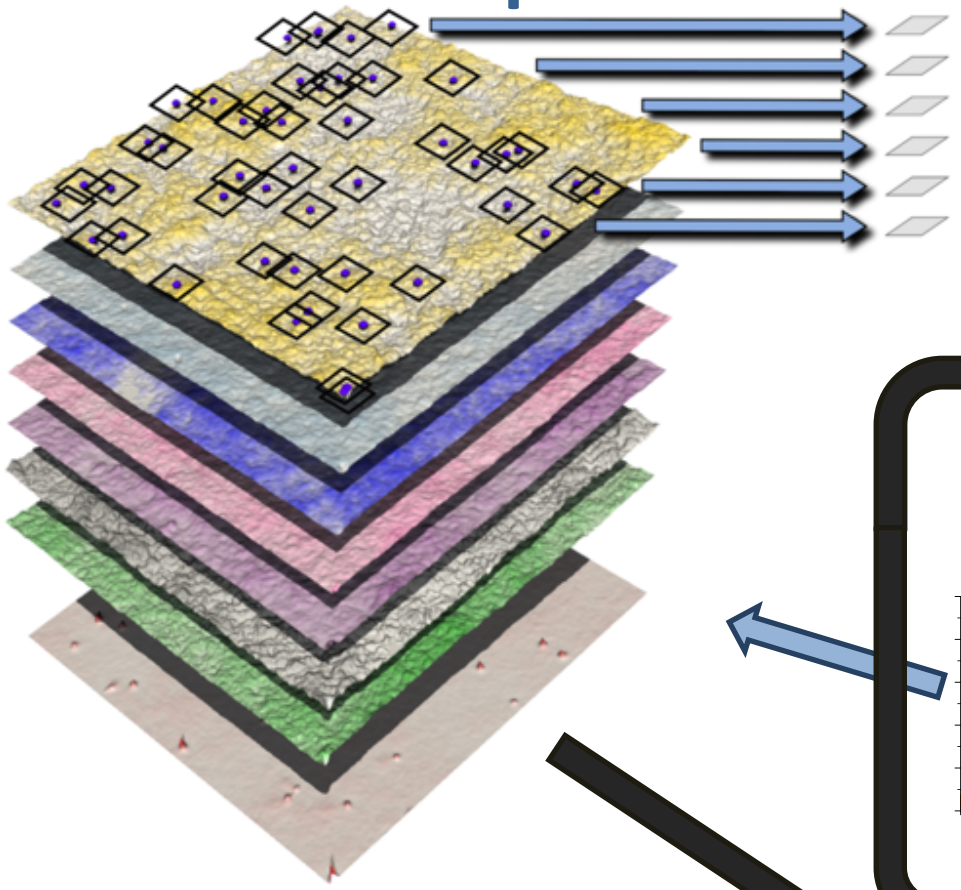
Single Macro Simulation
C++ (with MPI); MOOSE; ddcMD
24 CPU cores/node; 2400 MPI tasks
242 GB per day

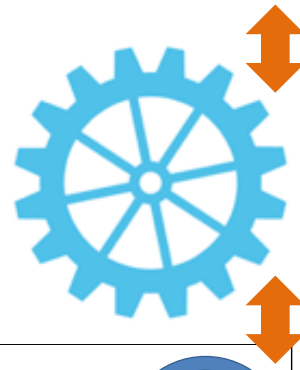
Analysis Aggreg. & Feedback
Python; Custom
24 CPU cores
120,000 reads per cycle

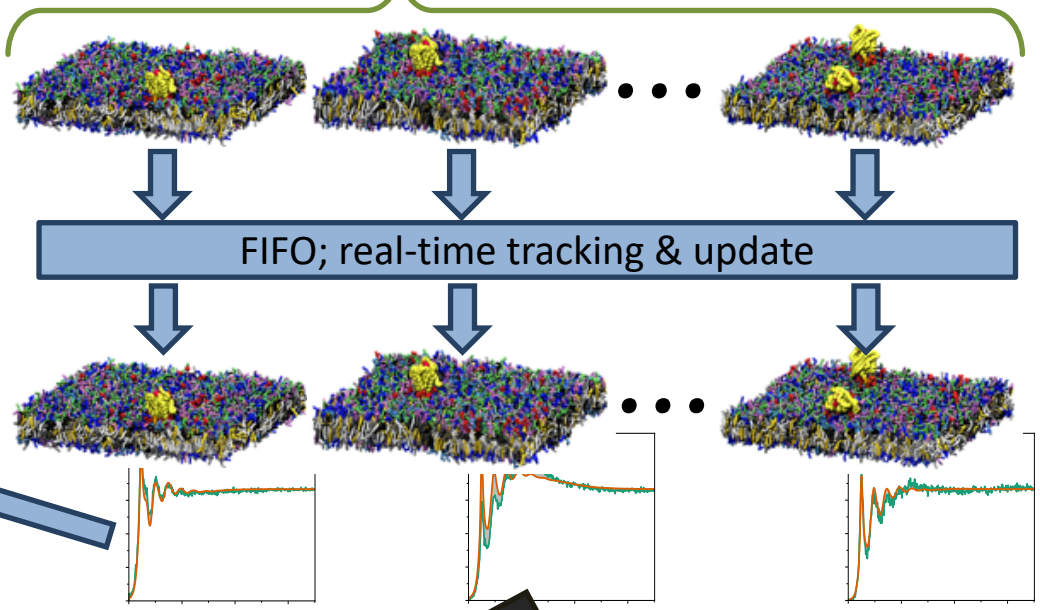
CG Simulation
C++ (with CUDA); ddcMD
1 GPU + 1 CPU core each
1.04 μs and ~6.5 GB per day

In situ CG Analysis
Python; Custom
3 CPU cores each
>2K frames per day

The workflow is responsible for managing various components in real time







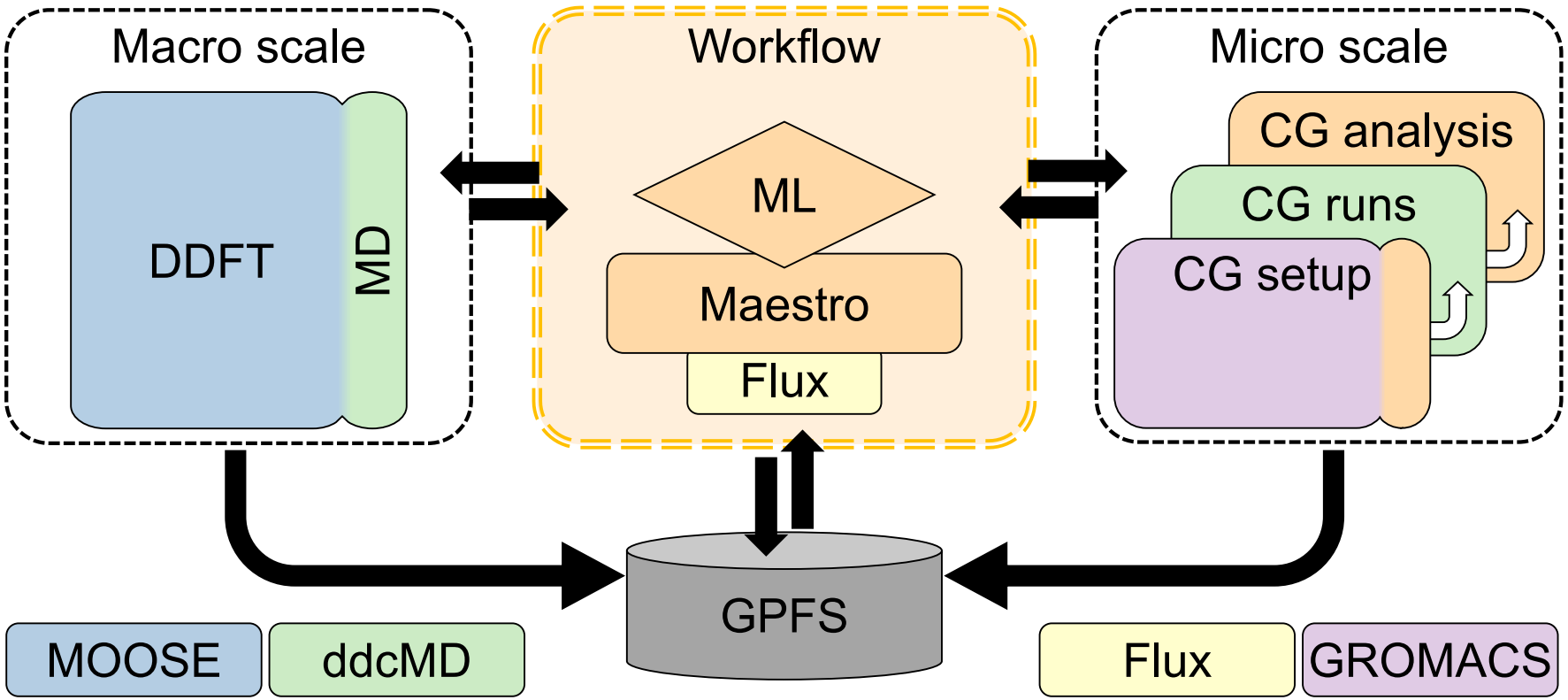
FIFO; real-time tracking & update

GPFS
1400 accesses per minute

In-memory communication

File system-based communication

The MuMMI workflow is built on a diverse set of independent tools and software



The overall workflow combines three high-level components to enable efficient execution at scale

- Custom in-memory workflow manager
  - highly configurable
  - on-the-fly task orchestration
  - coordinated checkpoint and fault recovery
- A new scheduling engine extending Flux [Dong et. al]
  - hierarchical scheduling of all available resources
  - co-location across heterogenous hardware components
  - high throughput, low latency processing of tens of thousands of jobs
- Data management framework
  - in situ analysis
  - asynchronous coordination
  - efficient archival

The scheduling engine efficiently provisions heterogenous resources across all of Sierra – 172K CPUs, 16K GPUs

- Existing scheduling engines proved not flexible enough
  - static and flat scheduling do not scale to handle the number of required jobs
  - schedulers typically allocate by node, therefore do not support dynamic workflows
  - high turn around time especially at high job counts

- We utilize Flux to provide
  - low-latency scheduling of thousands of jobs
  - hierarchical scheduling within large user-space allocation
  - explicit task pinning to maximize memory bandwidth

The workflow submits to Flux via an abstract interface provide by the Maestro Workflow Conductor

- Maestro is an open source, Python tool for running workflows on various schedulers.
- You can find Maestro on GitHub: https://github.com/LLNL/maestrowf
- Maestro implements an interface to Flux:

```
# NOTE: These libraries are compiled at runtime when an allocation
# is spun up.
self.flux = __import__("flux")
self.kvs = __import__("flux.kvs", globals(), locals(), ["kvs"])

# NOTE: Host doesn"t seem to matter for FLUX. sbatch assumes that the
# current host is where submission occurs.
self.add_batch_parameter("nodes", kwargs.pop("nodes", "1"))
self._addl_args = kwargs.get("args", [])

….
```

The workflow submits to Flux via an abstract interface provide by the Maestro Workflow Conductor

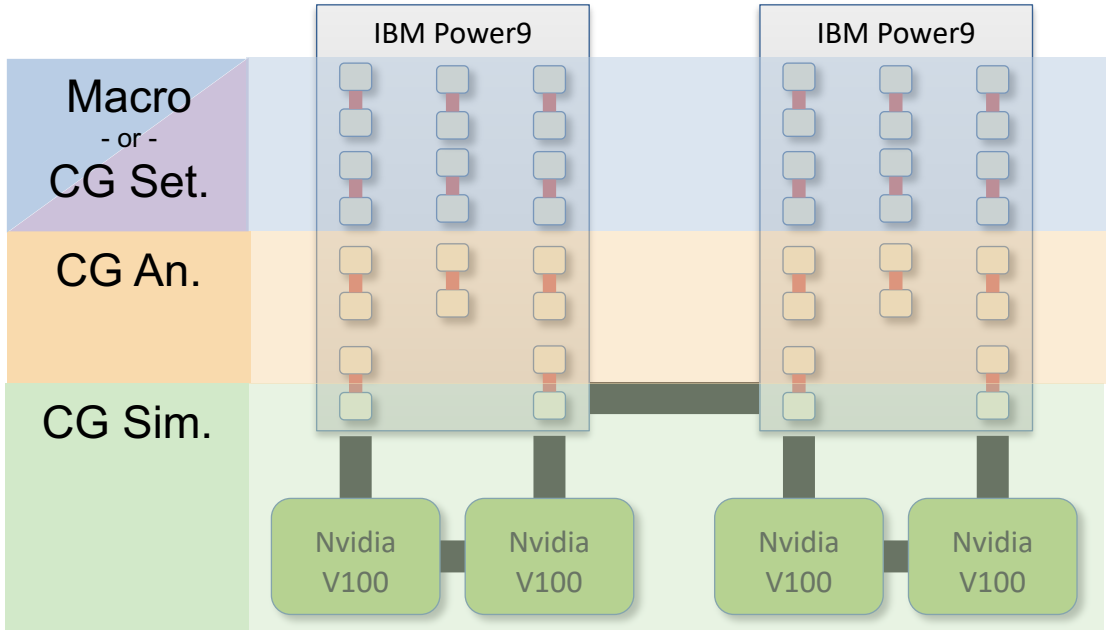
```
# Generate a job spec
jobspec = {
"nnodes": step.run["nodes"],
"ntasks": nodes,
"ncores": ncores,
"ngpus": ngpus,
"environ": get_environment(),
"options": {
    "stdio-delay-commit": 1
},
"opts": {
    "nnodes": nodes,
    "ntasks": nodes,
    "cores-per-task": cores_per_task,
    "tasks-per-node": 1,
},
"cwd": cwd,
"walltime": walltime,
}
```

```
wrapper = step.run.get("wrapper", "")
if step.run["nodes"] > 1:
    if wrapper == "":
      jobspec["cmdline"] = \
        ["flux", "broker", path]
    else:
      jobspec["cmdline"] = \
        ["flux", "broker", wrapper, path]
else:
    if wrapper == "":
      jobspec["cmdline"] = [path]
    else:
      jobspec["cmdline"] = [wrapper, path]

if self.h is None:
    self.h = self.flux.Flux()

resp = self.h.rpc_send(
    "job.submit", json.dumps(jobspec))
```

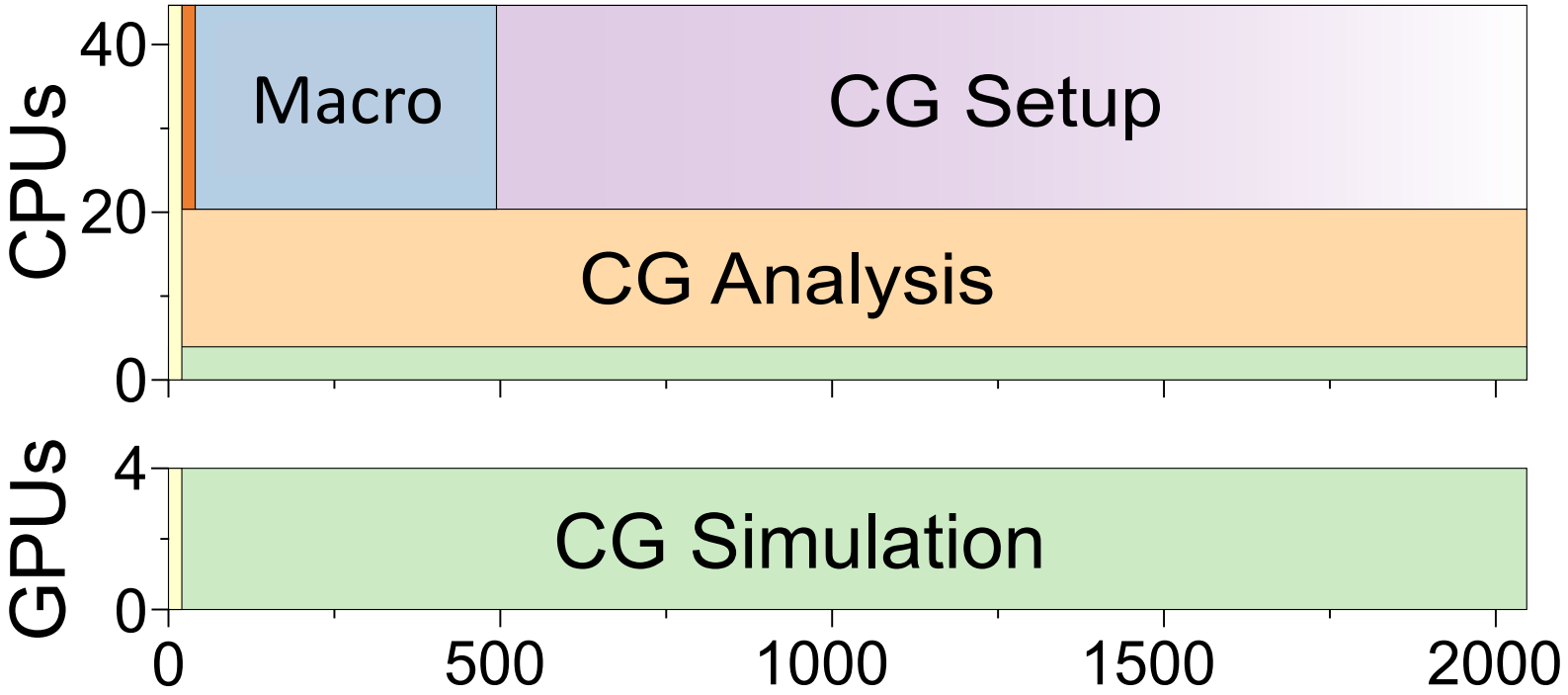
Flux allows nodes to be sub-divided to handle a diverse range of tasks

- Each subbroker manages a single node
- MuMMI utilizes binding scripts to pin processes
  - CG Simulations are pinned closer to the GPUs
  - CG Setup are swapped out as the task finishes
  - Macro simulations stay pinned to their nodes
- Flux allows for MuMMI to pin tasks because the user owns the nodes in an allocation

Sierra Node

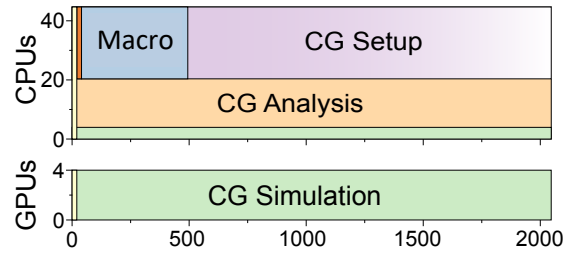


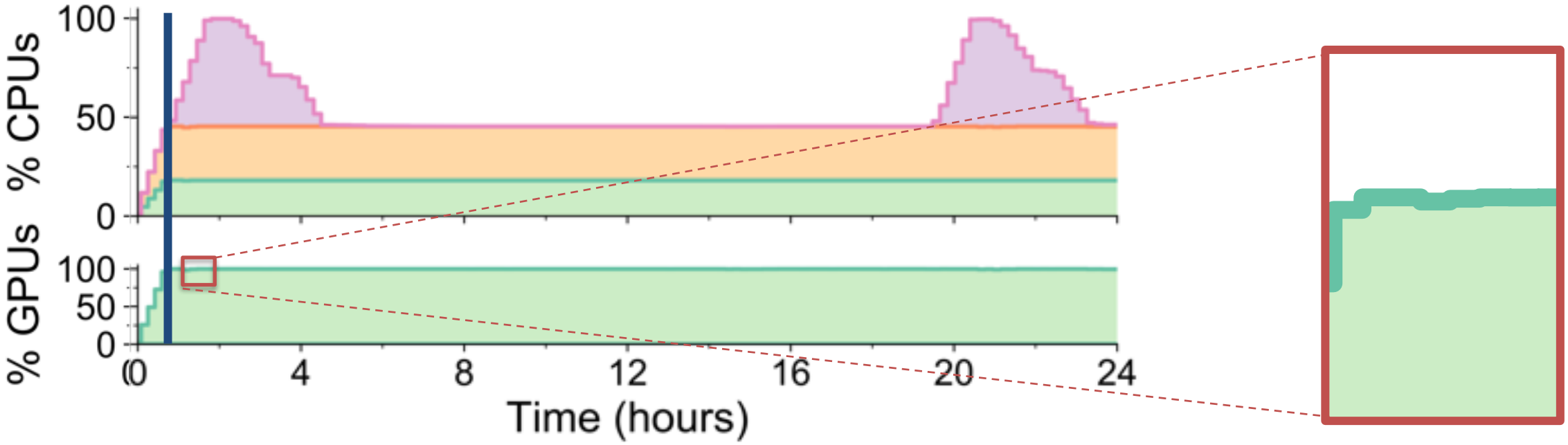
The scheduling engine efficiently provisions heterogenous resources across all of Sierra – 172K CPUs, 16K GPUs



The high through-put, low latency scheduling enables fast restarts and consistent utilization of all resources

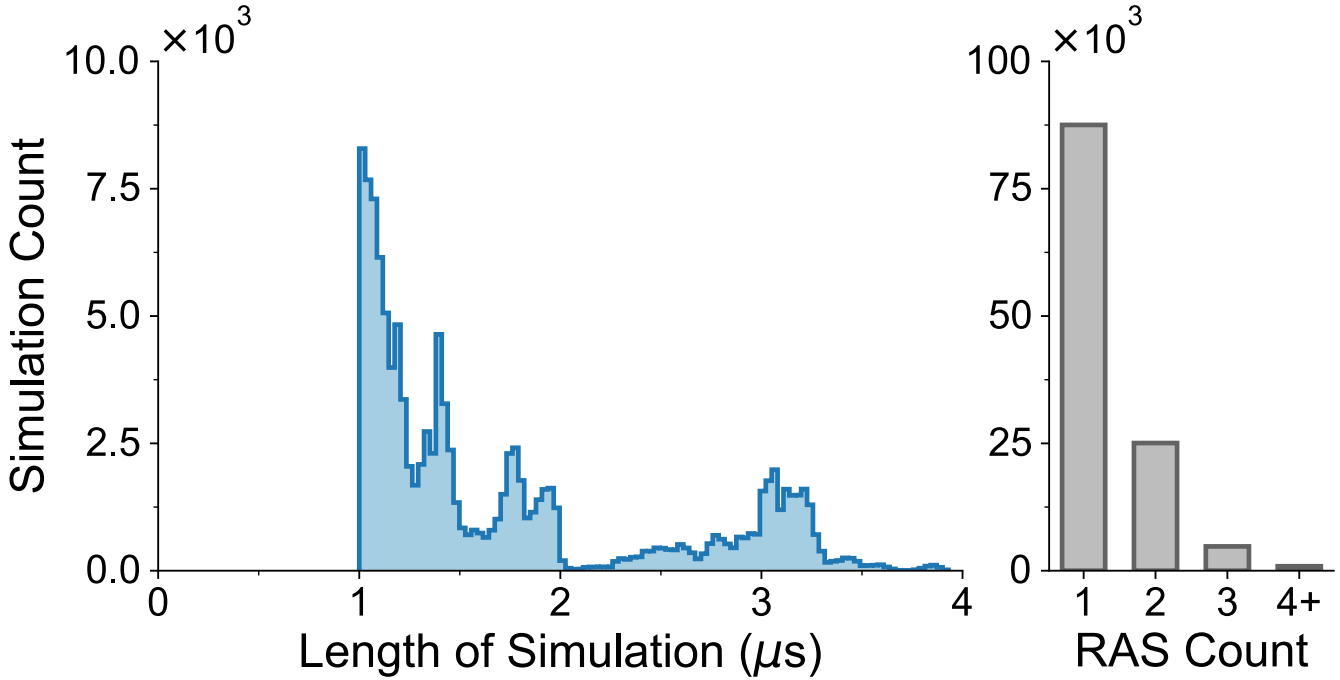
- Hierarchical scheduling allows us to reach steady state in ~45 minutes (newer versions will reduce turnaround time)
- Depending on the scientific hypothesis we utilize **>95%** of the available compute





A scientific campaign of unprecedented scale with simulations automatically selected by specified criteria

- About 120,000 unique micro simulations in total, with ~206 ms of simulated time.
- **Orders of magnitude** bigger than typical large campaigns



- 5.6 x 10^6 GPU hours consumed over ~14.5 days of wall-time

JDACS4C Pilot 2 team:
A multi-faceted problem requires a multi-disciplinary team

Ryan Berg, Harsh Bhatia, Timo Bremer, Tim Carpenter, Gautham Dharuman, Francesco Di Natale, Jim Glosli, Helgi Ingólfsson, Piyush Karande, Felice Lightstone, Shusen Liu, Corey McNeish, Adam Moody, Joseph Moon, Tomas Oppelstrup, Tom Scogland, Liam Stanton, Shiv Sundram, Michael Surh, Fred Streitz, Brian Van Essen, Xiaohua Zhang

Animesh Agarwal, Angel Garcia, Sandrasegaram Gnanakaran, Nick Hengartner, Jeevapani Hettige, Cesar Lopez, Chris Neale, Sumantra Sarkar, Tim Travers, Art Voter

Arvind Ramanathan

Debsindhu Bhowmik, Christopher Stanley

Debanjan Goswami, Gulcin Gulten, Frantz Jean-Francois, Dwight Nissley, Rebika Shrestha, Andy Stephen, Dhirendra Simanshu, Tommy Turbyville, Que Van

Bruce D'Amora, Changhoan Kim, Claudia Misale, Lars Schneidenbach, Sara Kokkila Schumacher

Flux Hands-on (Installing and Building)

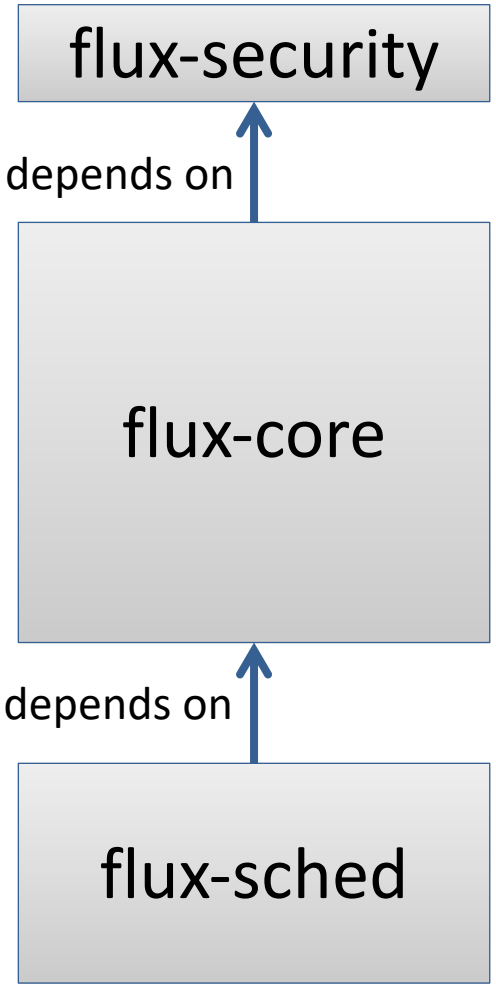
Anthony Agelastos, Dong H. Ahn, Frank Di Natale,
Stephen Herbein, Dan Milroy, Tapasya Patki

LLNL-PRES- 804364

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

Lawrence Livermore National Laboratory

Poly Repo Summary



- Contains setuid binary & libraries for multi-user support

- Resource Manager
- Core services:
  - Key-Value Store
  - Job Manager
  - Simple Scheduler
- Python Bindings
- Most Front-End CLIs

- Graph-based Resource Database
- Queue Manager with advanced scheduling policies

Docker Images (for users that want to give Flux a quick spin)

```
> docker pull fluxrm/flux-sched:latest
> docker run –ti fluxrm/flux-sched:latest bash
$ sudo /sbin/runuser -u munge /usr/sbin/munged
$ flux start
$ flux mini run hostname
4e94515812e9
```

Spack installation (for users that want a quick side-installation)

- Install `spack`
  - `git clone https://github.com/spack/spack.git`
- Setup `spack`
  - `export SPACK_ROOT=/home/patki1/flux-tutorial/spack/`
  - `export PATH=$SPACK_ROOT/bin:$PATH`
  - `source $SPACK_ROOT/share/spack/setup-env.sh`
- Install `flux-core` and `flux-sched`
  - `spack install flux-core@master`
  - `spack install flux-sched@master`

Spack installation (for users that want a quick side-installation)

- Install `environment-modules` for easy access (or `lmod`)
  - `spack install environment-modules`
  - `source `spack location -i environment-modules`/Modules/init/bash*`
- Flux can now be loaded and unloaded as a module, `spack` sets correct paths
  - `spack load flux-core@master`
  - `spack load flux-sched@master`
  - `$$ which flux`
  - `$$ /home/patki1/flux-tutorial/spack/opt/spack/linux-ubuntu18.04-x86_64/gcc-7.3.0/flux-core-master-5vtsnto3hbhllffc7g6nkht2at7r3hfp/bin/flux`
  - `spack unload flux-core@master`
  - `spack unload flux-sched@master`

*Similar for `tcsh` or other shells

Installation from source (for developers)

- Clone `flux-core`
https://github.com/flux-framework/flux-core.git

- Install dependencies

- If using ubuntu, the following may be needed:

```
sudo ln -s posix_c.so /usr/lib/x86_64-linux-gnu/lua/5.1/posix.so
```

```
./autogen.sh
./configure --prefix=<path-to>/flux-install/
make –j
make check
make install
```

redhat	ubuntu	version
autoconf	autoconf	
automake	automake	
libtool	libtool	
libsodium-devel	libsodium-dev	>= 1.0.14
zeromq4-devel	libzmq3-dev	>= 4.0.4
czmq-devel	libczmq-dev	>= 3.0.1
jansson-devel	libjansson-dev	>= 2.6
lz4-devel	liblz4-dev	
hwloc-devel	libhwloc-dev	>= v1.11, < 2.0
sqlite-devel	libsqlite3-dev	>= 3.0.0
yaml-cpp-devel	libyaml-cpp-dev	>= 0.5.1
lua	lua5.1	>= 5.1, < 5.3
lua-devel	liblua5.1-dev	>= 5.1, < 5.3
lua-posix	lua-posix	
python-devel	python-dev	>= 2.7
python-cffi	python-cffi	>= 1.1
python-six	python-six	>= 1.9
python-yaml	python-yaml	>= 3.10.0
python-jsonschema	python-jsonschema	>= 2.3.0
asciidoc	asciidoc	
asciidoctor	asciidoctor	>= 1.5.7
aspell	aspell	
valgrind	valgrind	
mpich	mpich	

Test `flux-core` installation

```
$$ flux keygen
Saving /home/herbein1/.flux/curve/client
Saving /home/herbein1/.flux/curve/client_private
Saving /home/herbein1/.flux/curve/server
Saving /home/herbein1/.flux/curve/server_private

$$ flux start --size=2
flux-start: warning: setting --bootstrap=selfpmi due to --size option

$$ flux mini run -n2 hostname
ip-172-31-23-210
ip-172-31-23-210

$ flux jobs -a
       JOBID USER     NAME       STATE    NTASKS TIME
861510041600 herbein1 hostname   INACTIVE      2 0.31s

$$ flux ping --count=2 kvs
kvs.ping pad=0 seq=0 time=0.421 ms (DADF8!511D5!0!0)
kvs.ping pad=0 seq=1 time=0.458 ms (DADF8!511D5!0!0)
```

Test `flux-core` installation

```
$$ flux module list
Module                     Size Digest  Idle  S Service
job-manager             1304128 02A5E34   23  S
job-ingest              1205960 D4A524B   21  S
kvs                     1558368 5F5DC5A    0  S
barrier                 1124120 4E47443   23  S
aggregator              1141112 3E7DD17   19  S
job-exec                1273448 0ECD779   23  S
userdb                  1122368 34E0A75   19  S
cron                    1202688 E6A15D3    0  S
sched-simple            1236424 B297382   23  S sched
content-sqlite          1130168 8284A49   23  S content-backing
job-info                1302648 A490E12    8  S
connector-local         1110664 62924A9    0  R
kvs-watch               1299800 A31D222   23  S
```

Installation from source (for developers)

- Clone `flux-sched`

https://github.com/flux-framework/flux-sched.git

- Install `flux-core`, install dependencies

- Export `PKG_CONFIG_PATH`

```
export PKG_CONFIG_PATH=<path-to>/flux-install/lib/pkgconfig:$PKG_CONFIG_PATH
```

```
./autogen.sh
./configure --prefix=<path-to>/flux-install/
make –j
make check && make install
```

```
libhwloc-dev >= 1.11.1
libboost packages == 1.53 or > 1.58
      - libboost-dev
      - libboost-system-dev
      - libboost-filesystem-dev
      - libboost-thread-dev
      - libboost-graph-dev
      - libboost-regex-dev
libxml2-dev >= 2.9.1
```

Test `flux-sched` installation

```
$$ flux keygen # if you didn't run keygen while testing flux-core
$$ flux start --size=2
$$ flux module remove sched-simple
$$ flux module load resource
$$ flux module load qmanager

$$ flux module list
Module                   Size Digest  Idle  S Service
cron                  1202688 C983CCA    0  S
barrier               1124120 0D89A2E   19  S
job-info              1302648 9E06142   16  S
job-ingest            1205960 3402F27   14  S
qmanager              1088544 F13B4EA   16  S sched
job-exec              1273448 0934B49   18  S
kvs-watch             1299800 EDAA28F   16  S
aggregator            1141112 D2C34C5   18  S
userdb                1122368 57F855E   18  S
kvs                   1558368 74CC3B0    0  S
resource             18210448 9CFBCFD   22  S
job-manager           1304128 C49E095   16  S
connector-local       1110664 4B13F7C    0  R
content-sqlite        1130168 BB634BE   16  S content-backing
```

Test `flux-sched` installation

```
$$ flux mini submit -n2 sleep 0
32812040192000

$$ flux job info 32812040192000 R
{"version":1,"execution":{"R_lite":[{"rank":"0","node":"quartz2300","children":{"core":"34-35"}}]}}

$$ flux job eventlog 32812040192000
1579158133.589118 submit userid=51249 priority=16 flags=0
1579158133.606256 depend
1579158133.616810 alloc note=""
1579158133.634276 start
1579158133.723823 finish status=0
1579158133.732106 release ranks="all" final=true
1579158133.733731 free
1579158133.733747 clean
```


Lawrence Livermore
National Laboratory

Flux: Hands-on Workflow Examples

Anthony Agelastos, Dong H. Ahn, Frank Di Natale,
Stephen Herbein, Dan Milroy, Tapasya Patki

February 2020

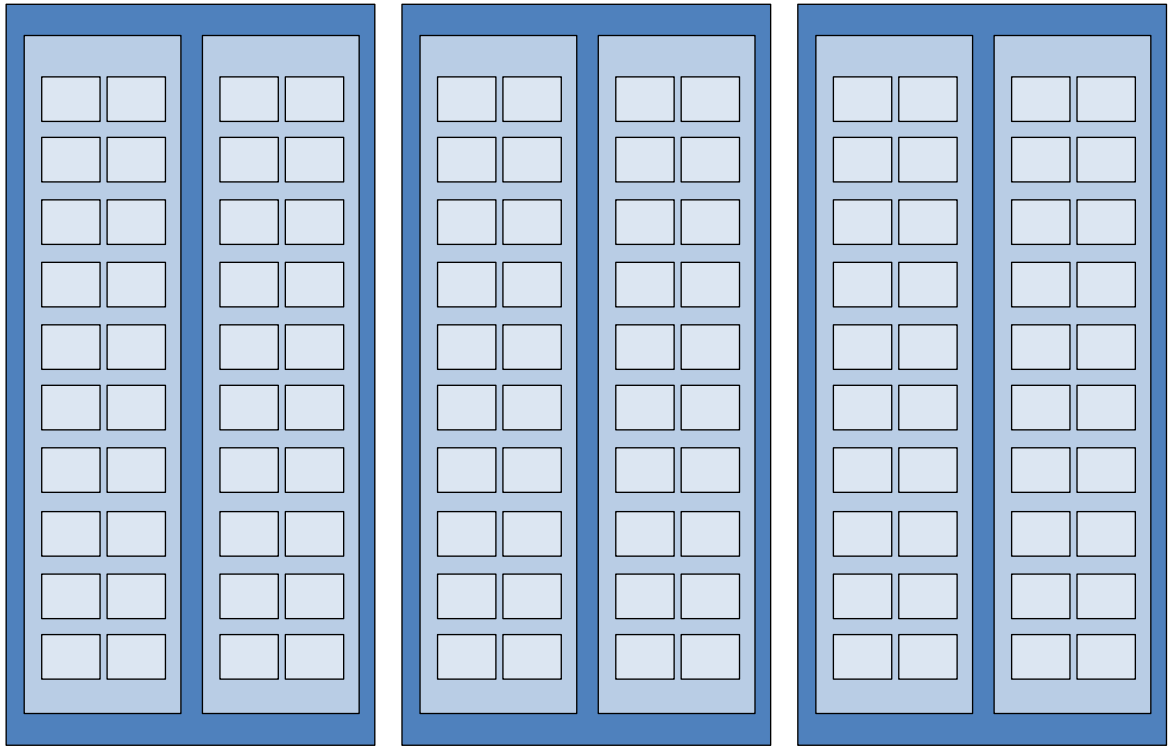
LLNL-PRES- 804364

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

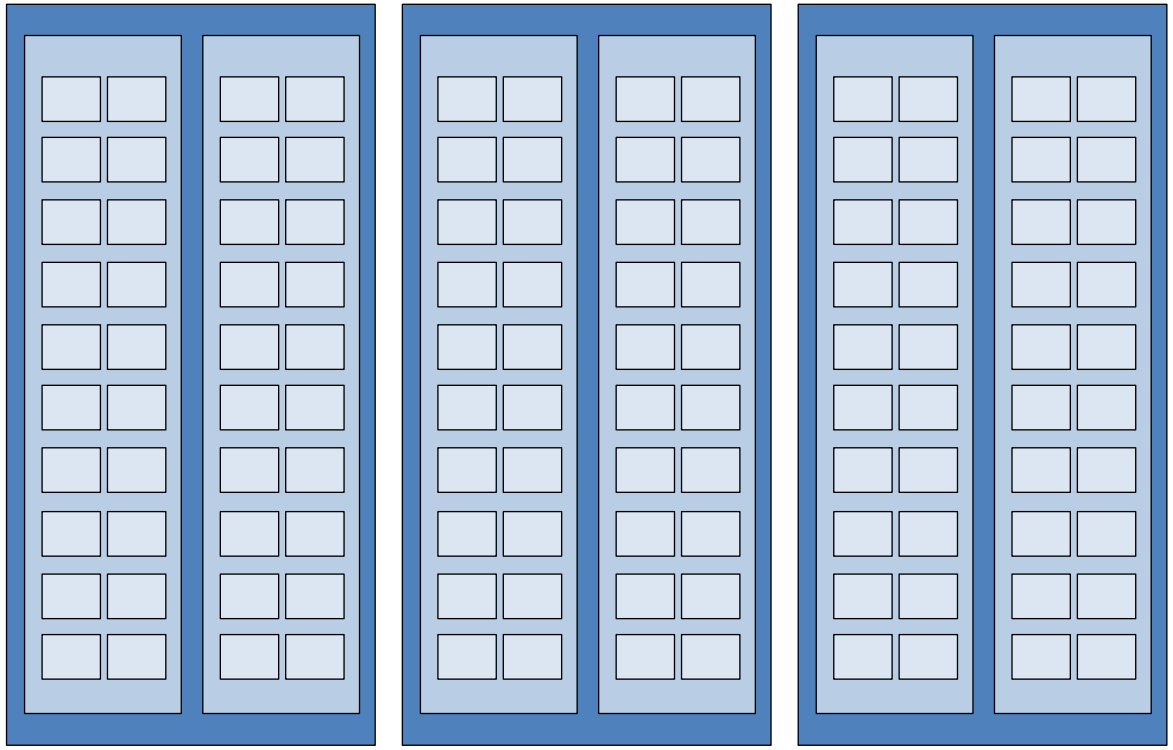
Lawrence Livermore National Laboratory

Example 1: Partitioning vs Overlapping Scheduling - CLI

Partitioning Scheduling



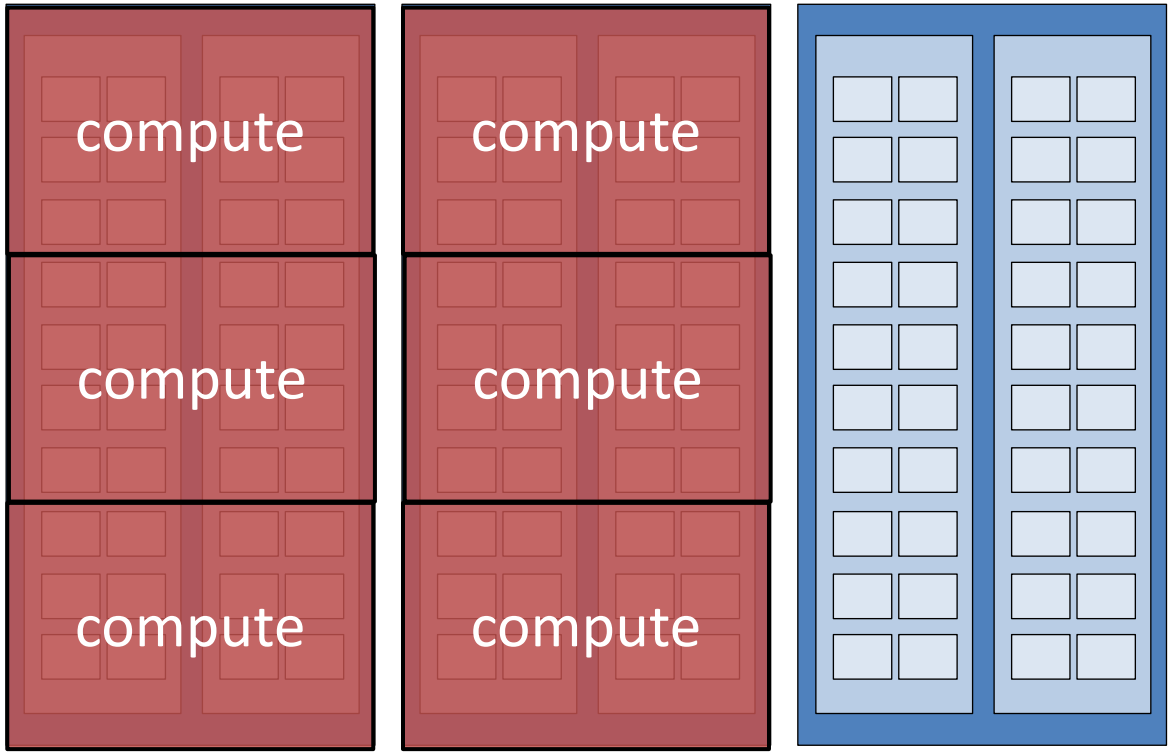
Overlapping Scheduling



Example 1: Partitioning vs Overlapping Scheduling - CLI

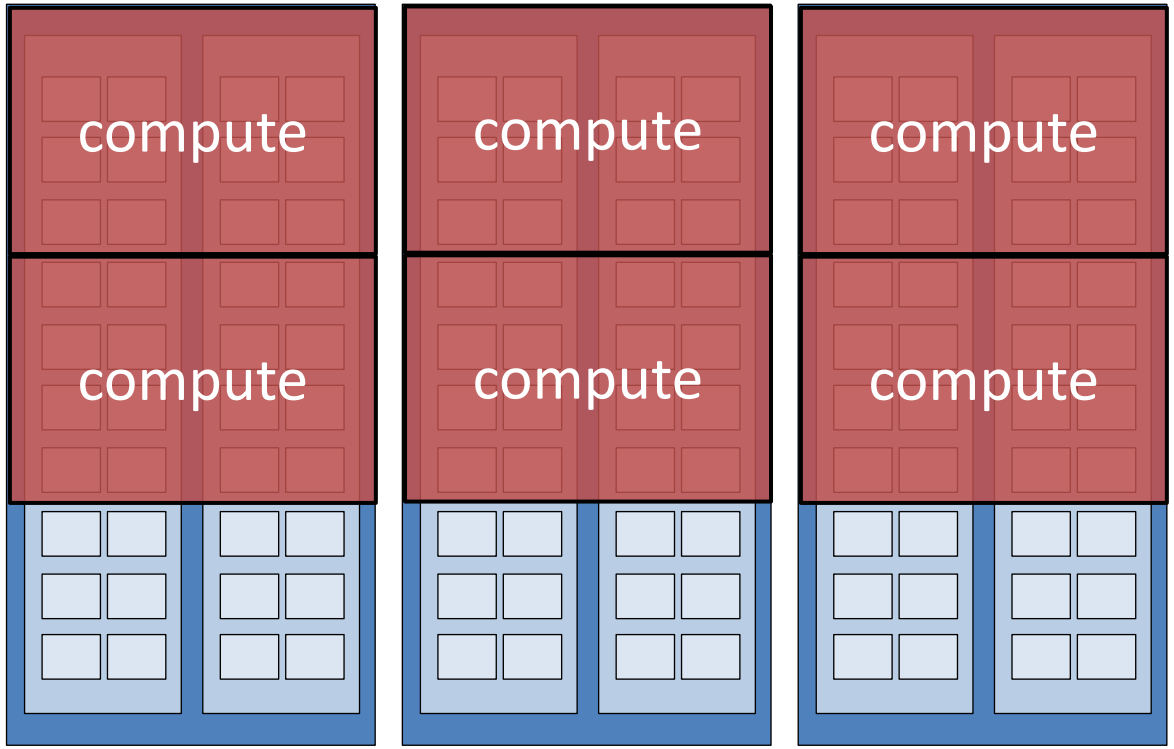
Partitioning Scheduling

flux mini submit –N 2 –n 6 ./compute.py



Overlapping Scheduling

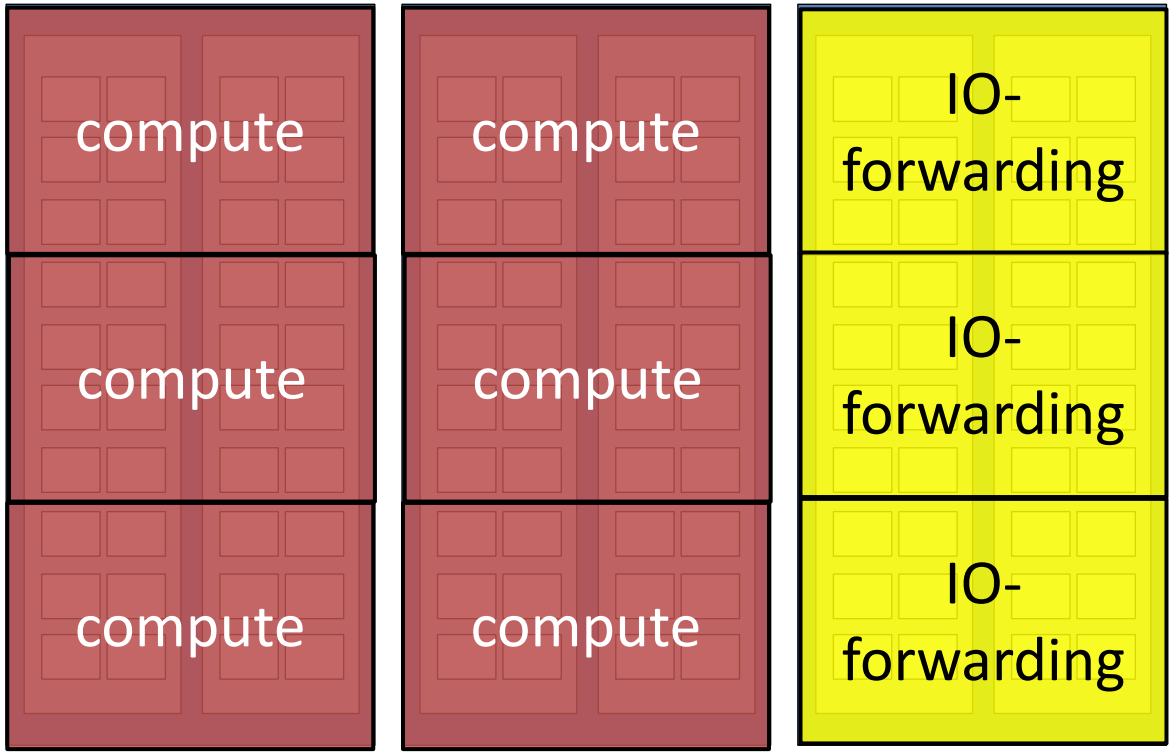
flux mini submit –N 3 –n 6 ./compute.py



Example 1: Partitioning vs Overlapping Scheduling - CLI

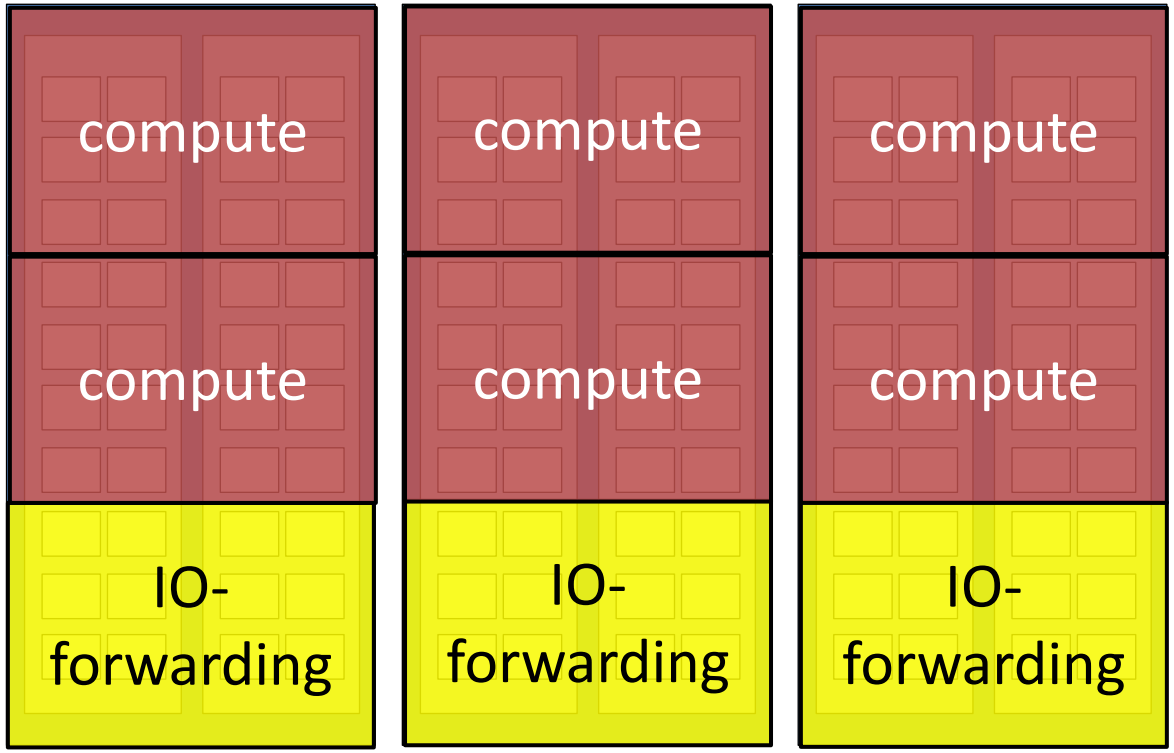
Partitioning Scheduling

flux mini submit –N 2 –n 6 ./compute.py
flux mini submit –N 1 –n 3 ./io-forwarding.py



Overlapping Scheduling

flux mini submit –N 3 –n 6 ./compute.py
flux mini submit –N 3 –n 3 ./io-forwarding.py

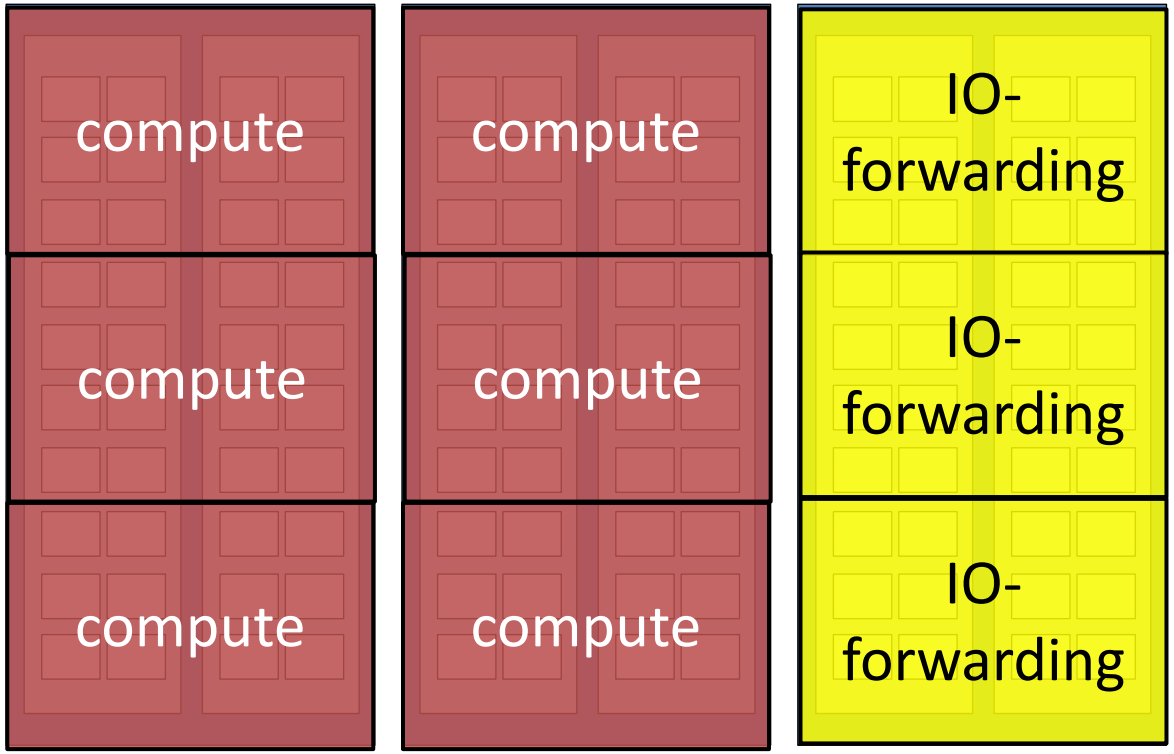


Example 2: Partitioning vs Overlapping Scheduling - Python

Partitioning Scheduling

submitter.py:

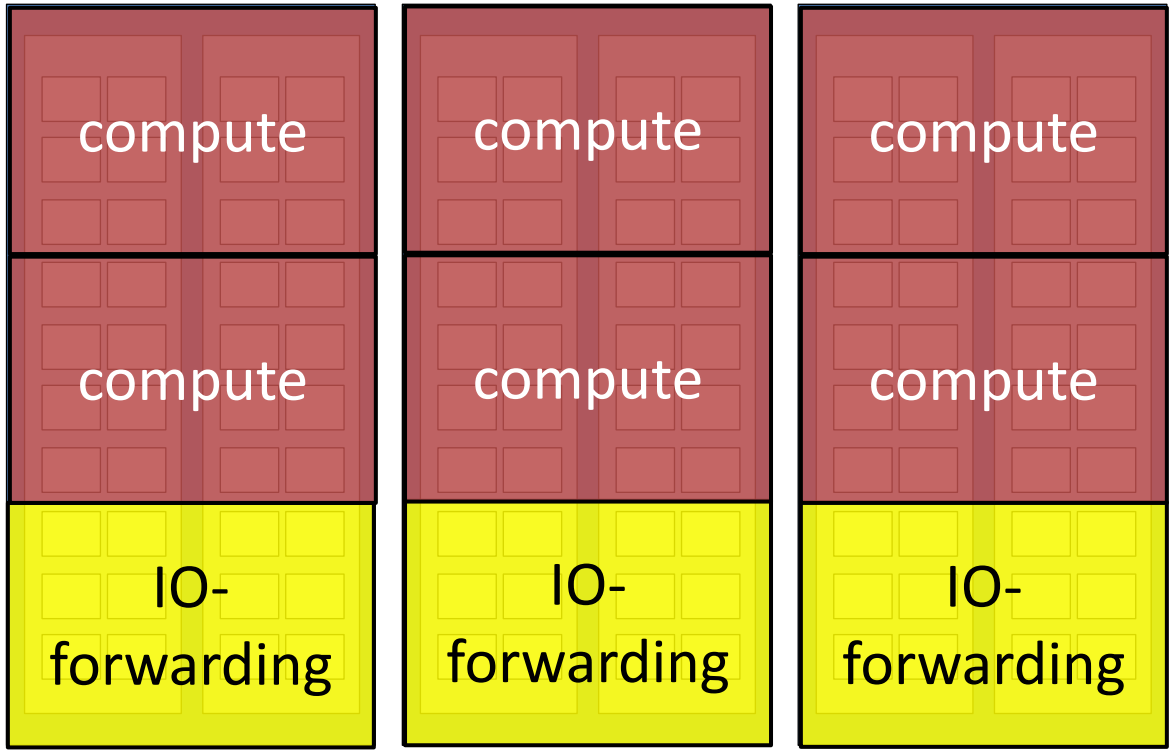
```
flux.job.JobspecV1.from_command(
    command=["./compute.py"],
    num_tasks=6, num_nodes=2)
```



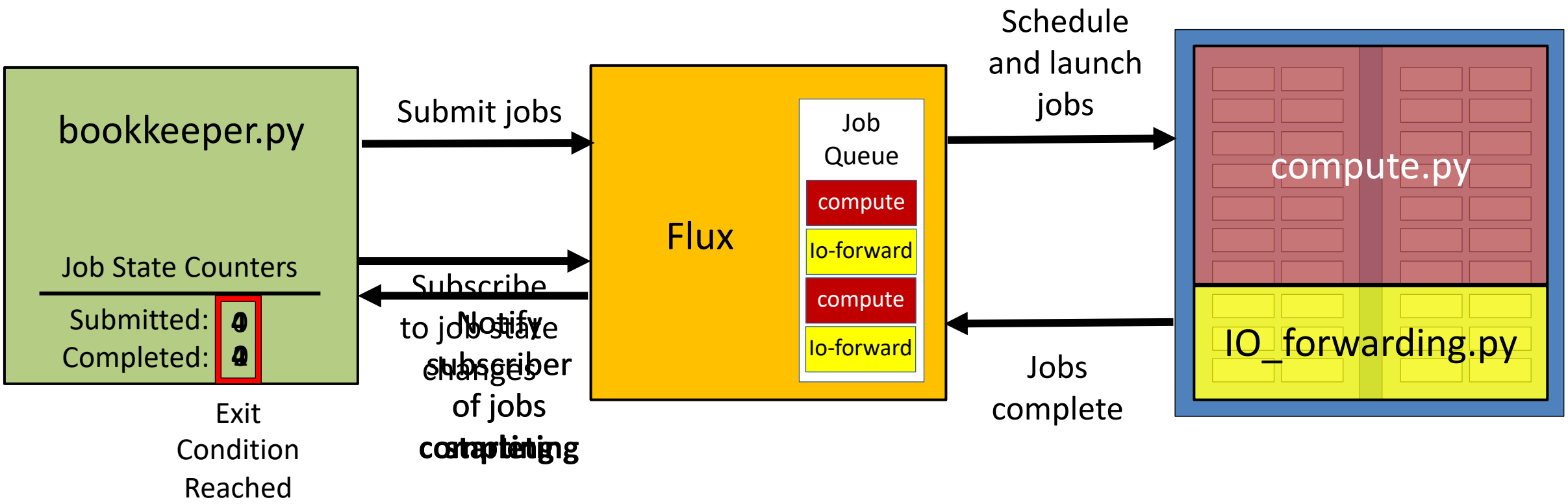
Overlapping Scheduling

submitter2.py:

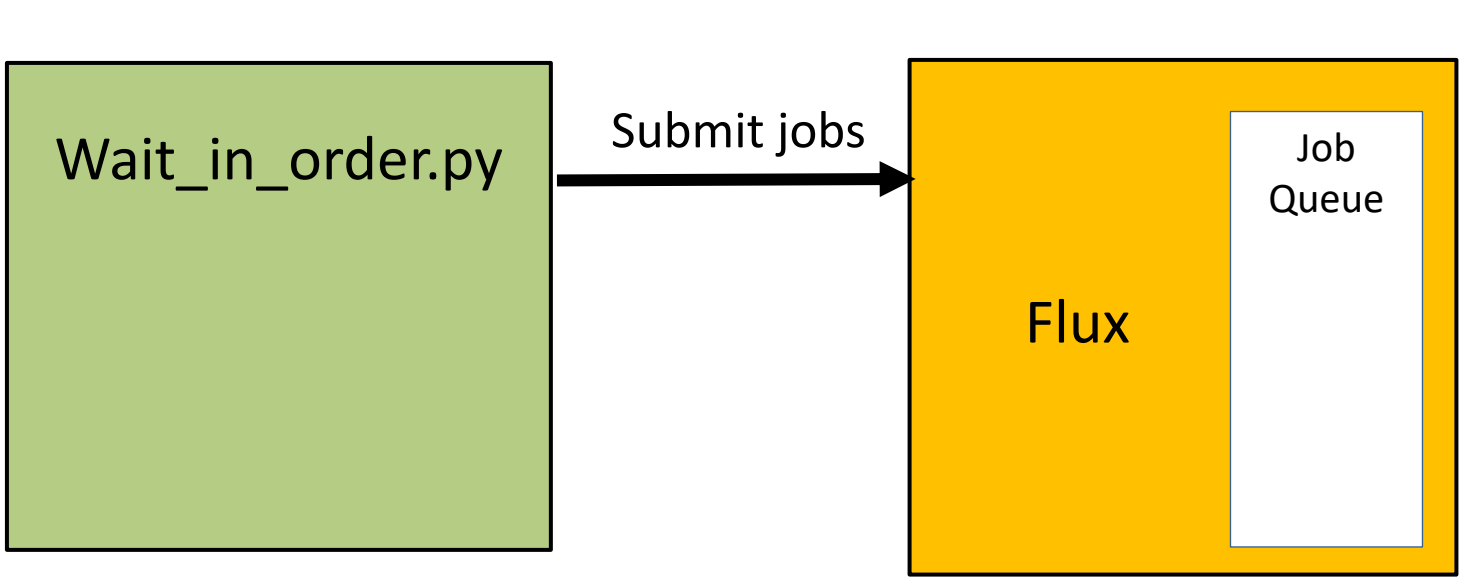
```
flux.job.JobspecV1.from_command(
    command=["./compute.py"],
    num_tasks=6, num_nodes=3)
```

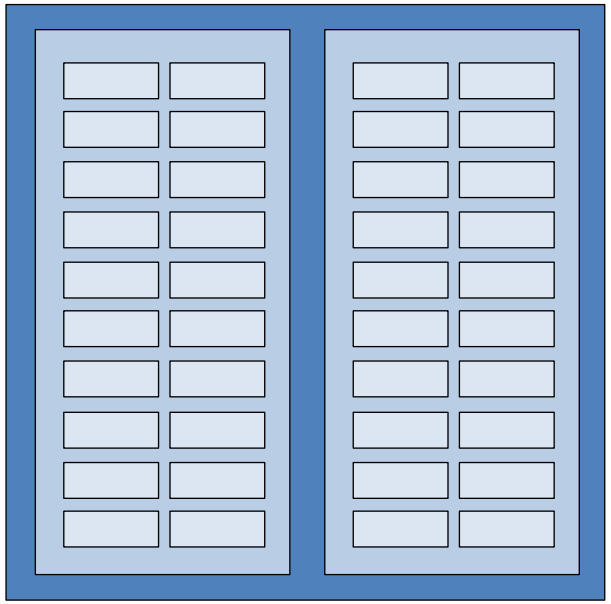


Example 7: Job Status & Control API

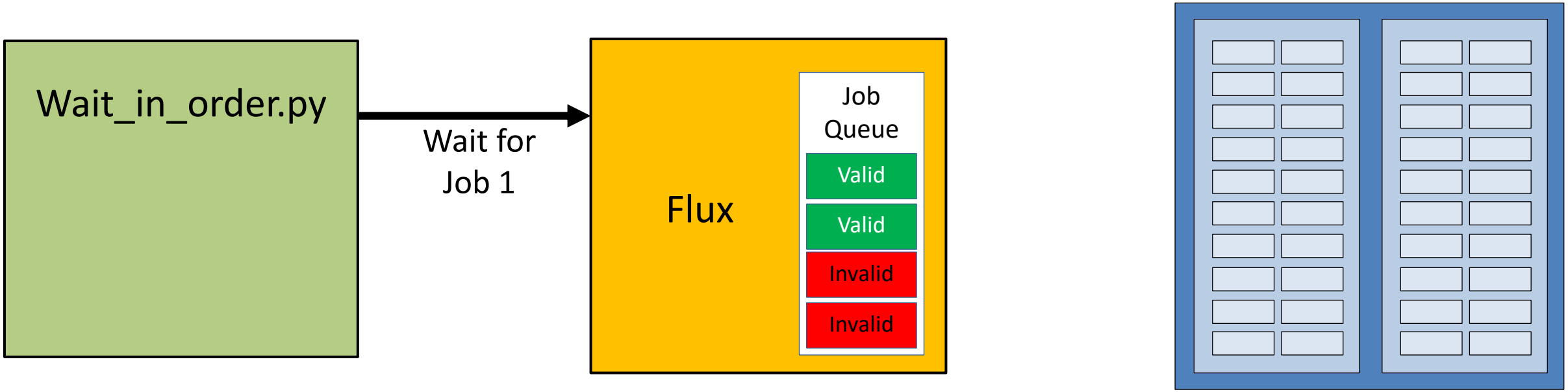


Example 10: Wait in Order

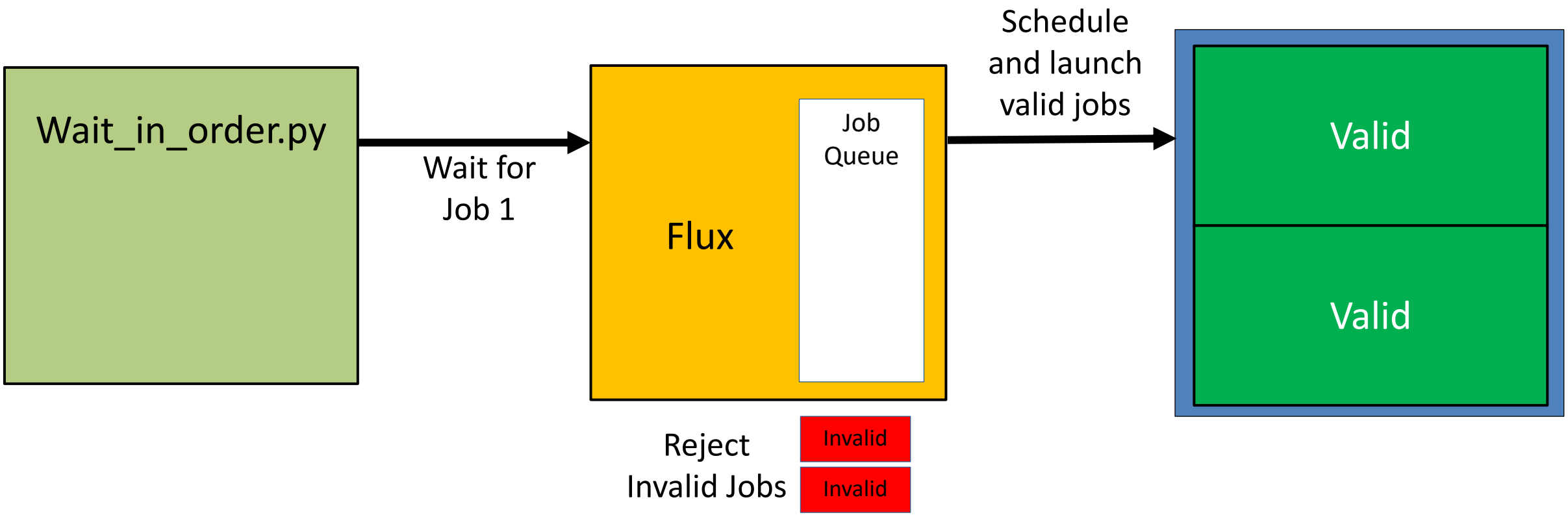




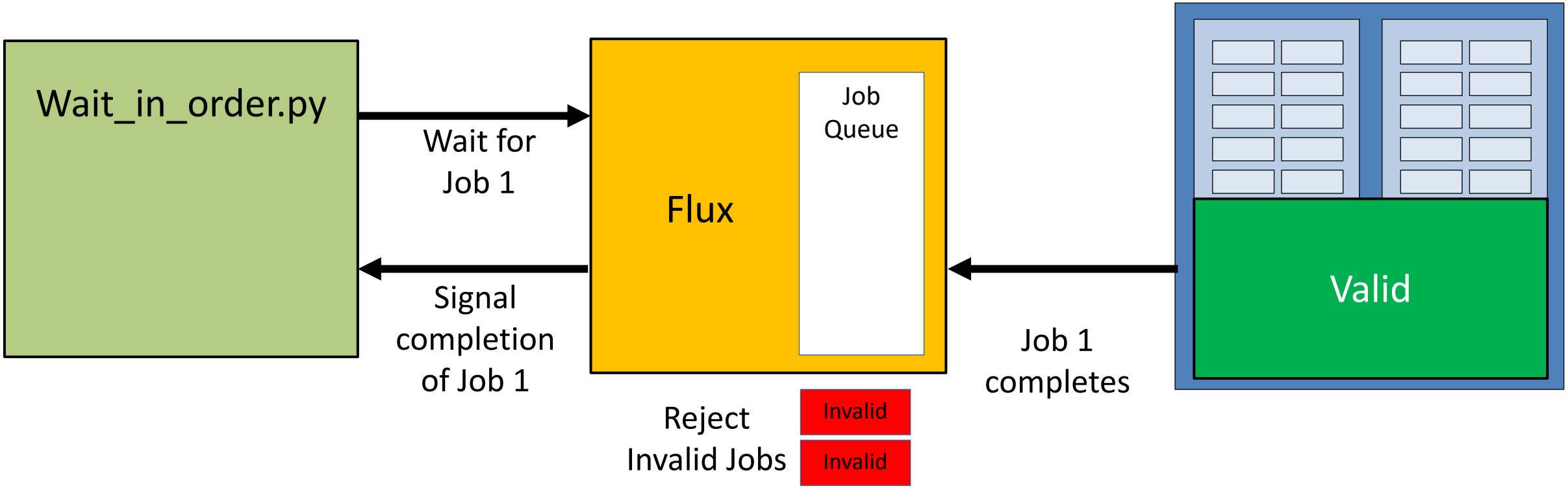
Example 10: Wait in Order



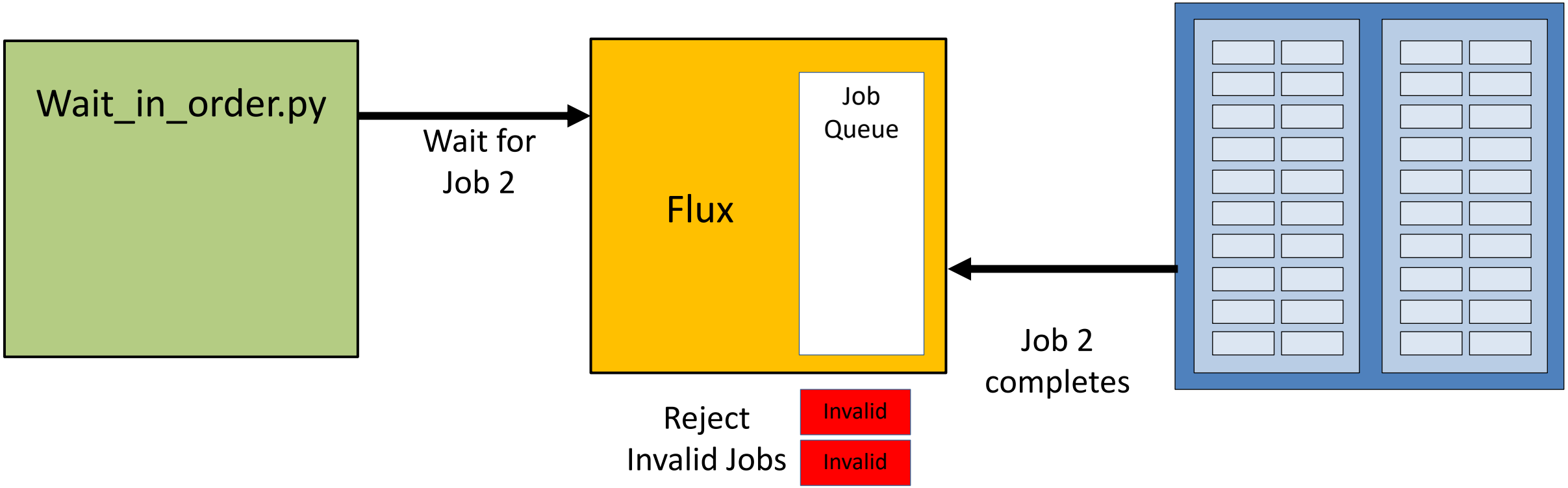
Example 10: Wait in Order



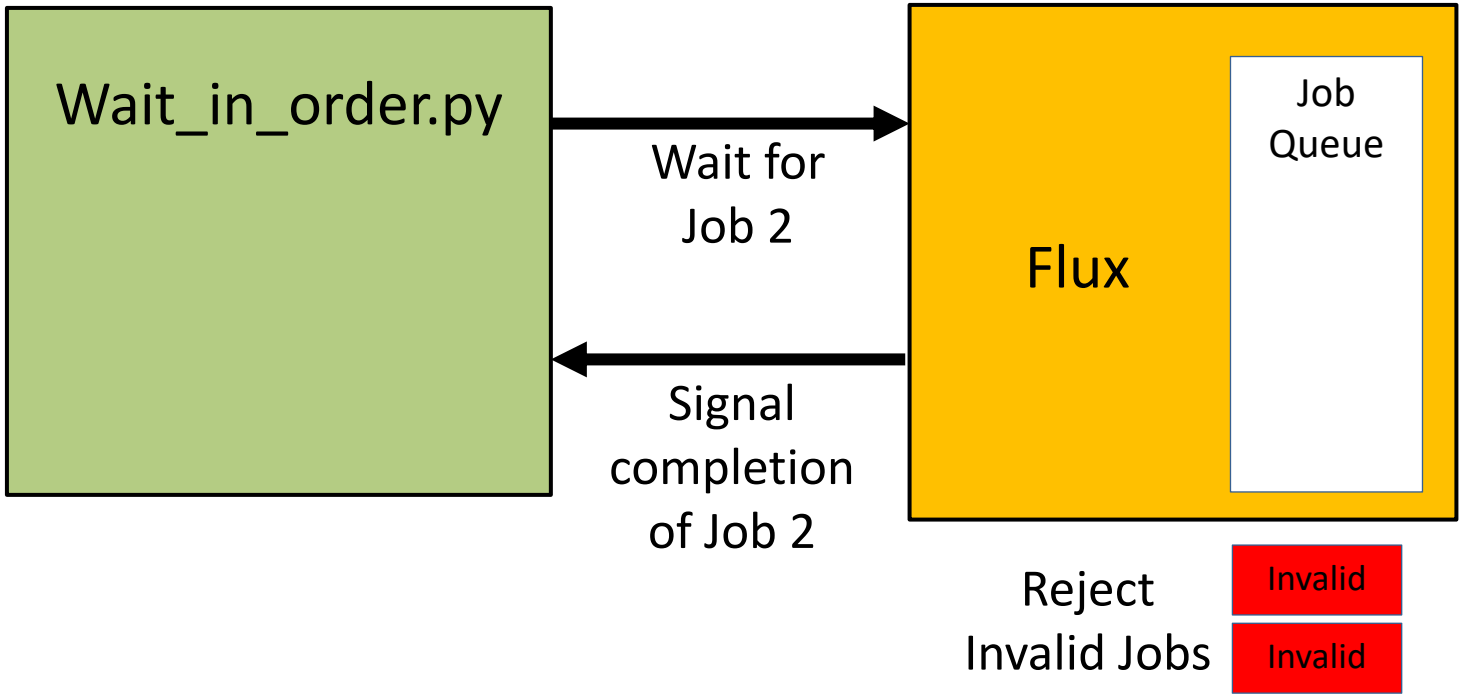
Example 10: Wait in Order

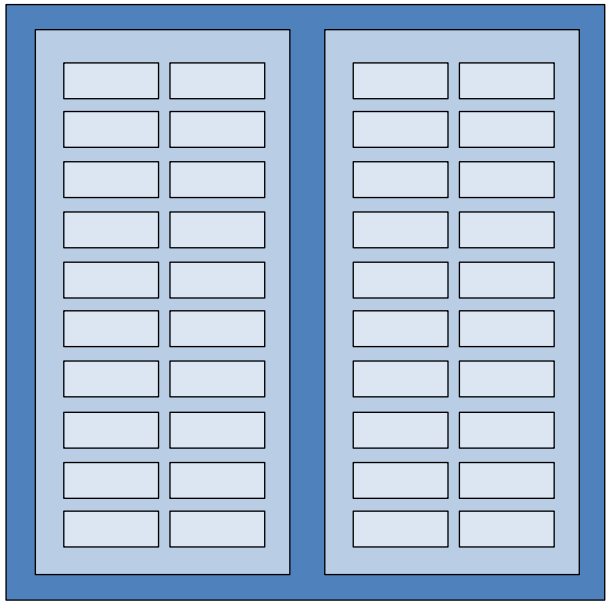


Example 10: Wait in Order

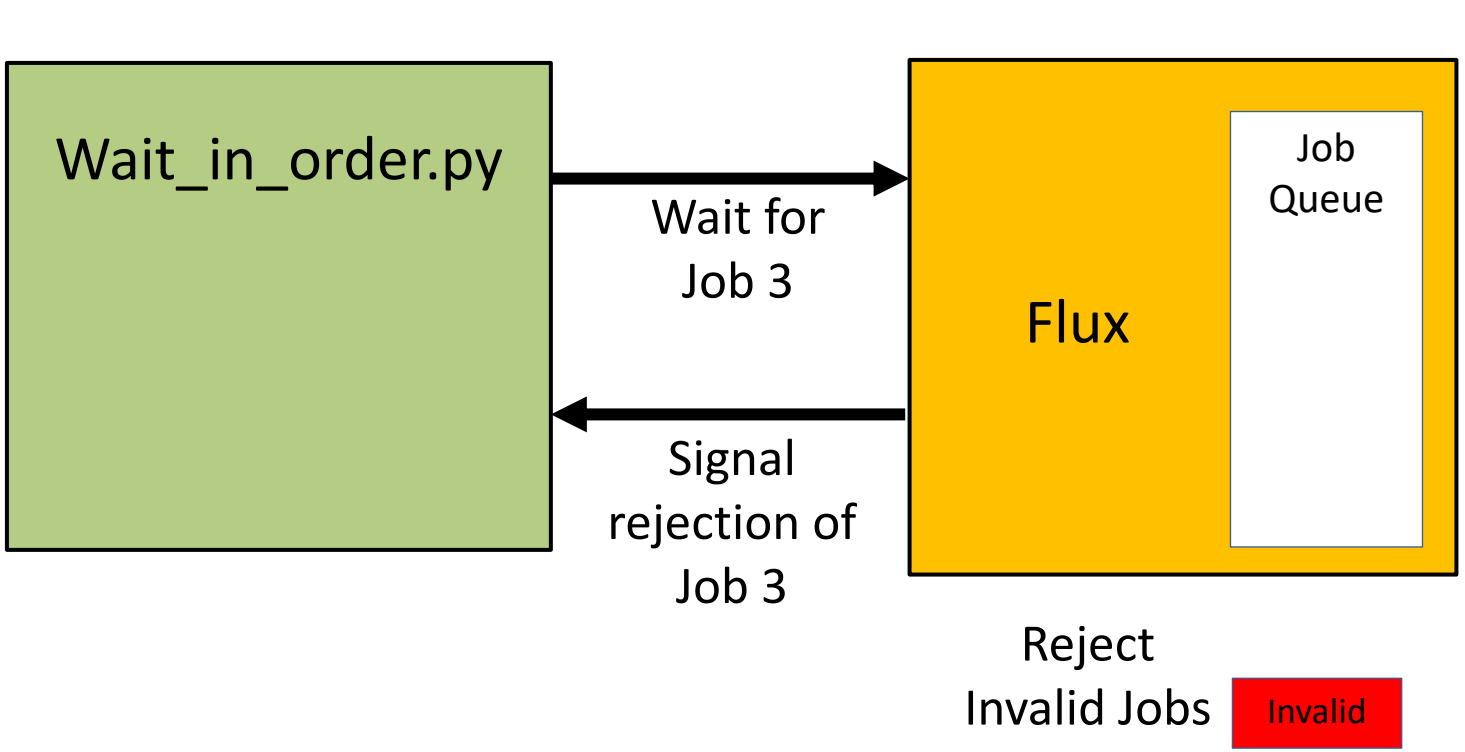


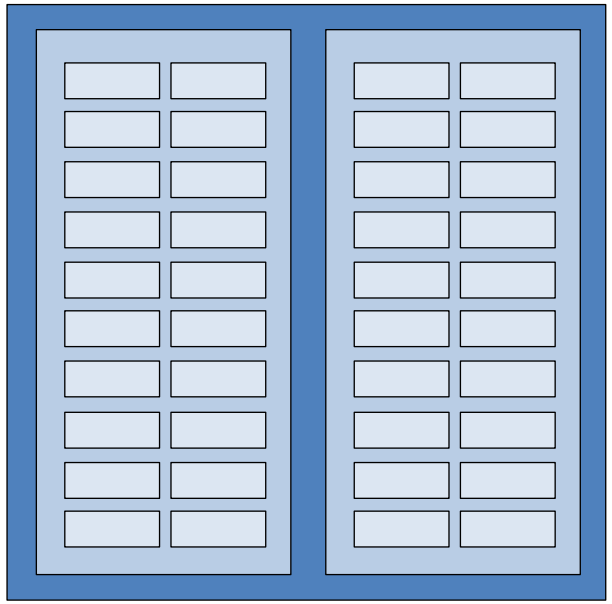
Example 10: Wait in Order



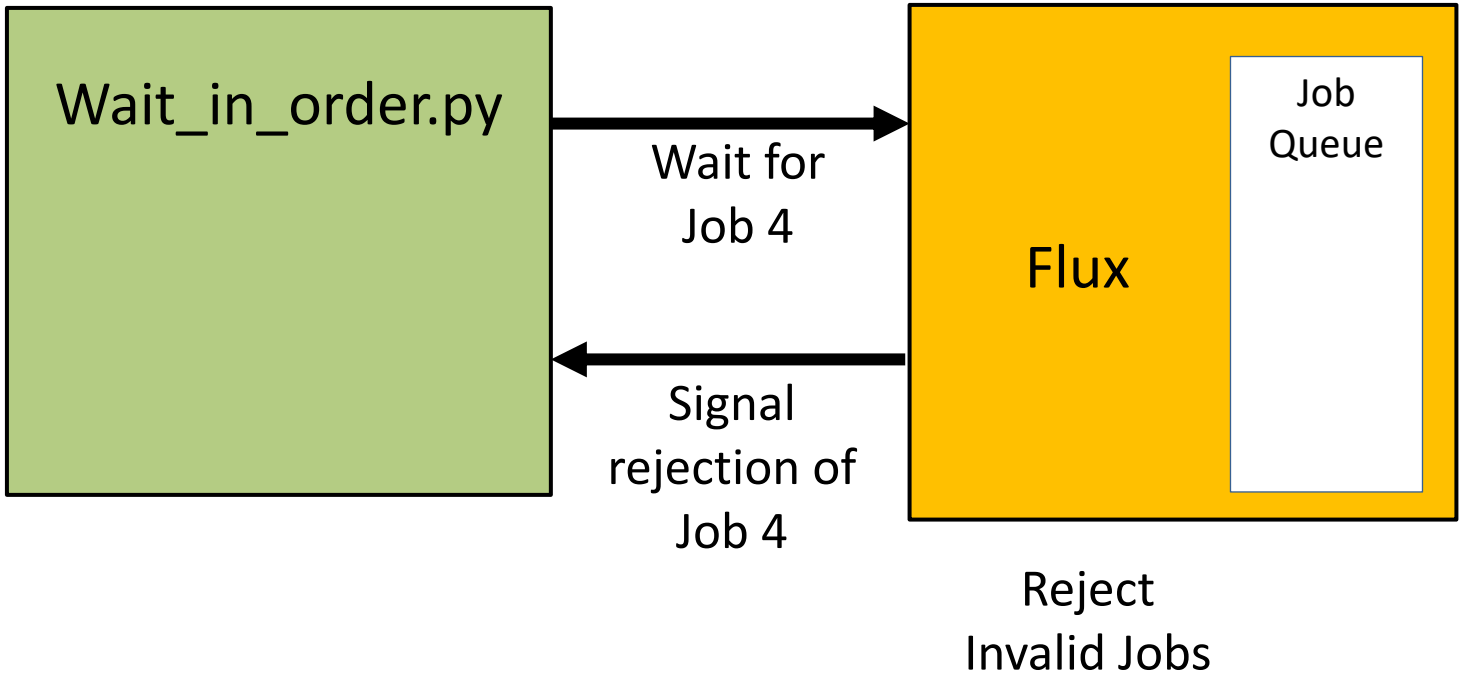


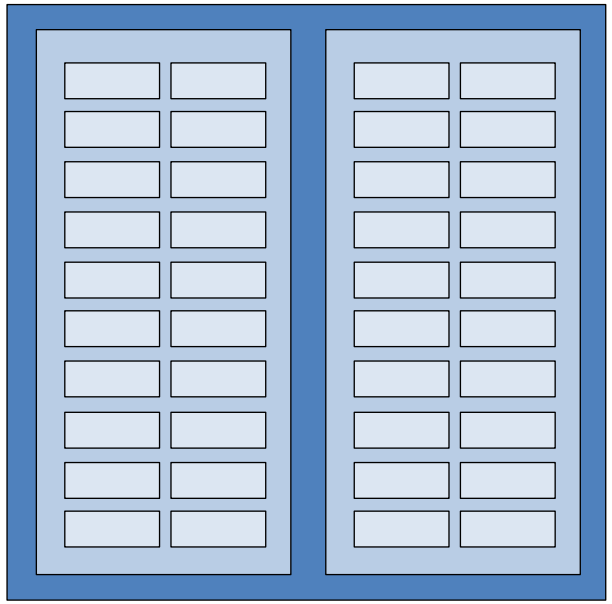
Example 10: Wait in Order





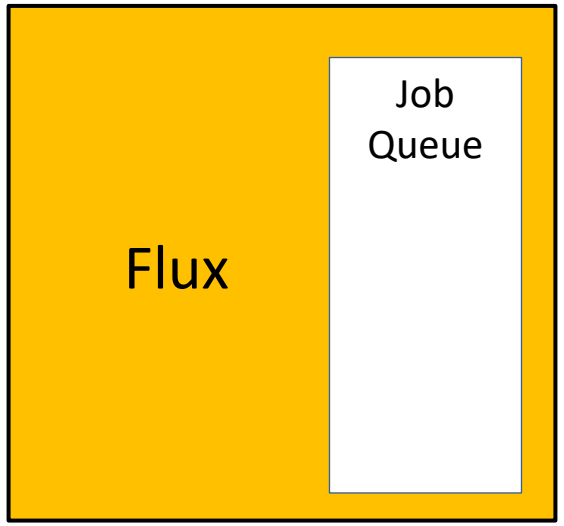
Example 10: Wait in Order

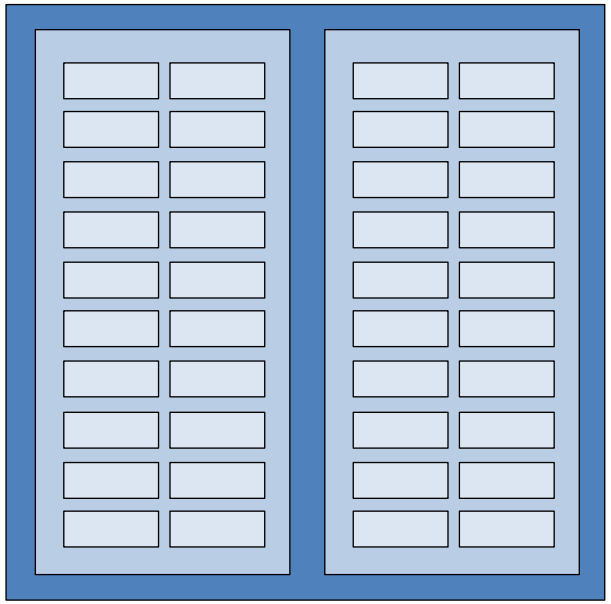




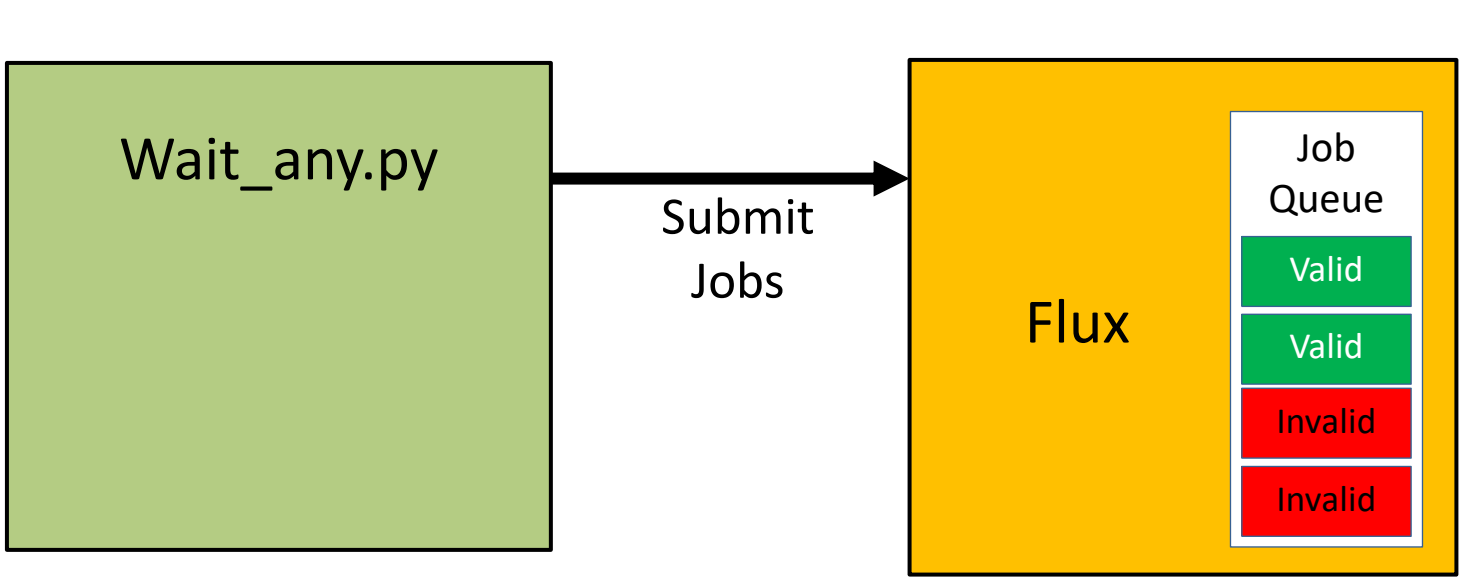
Example 10: Wait in Order

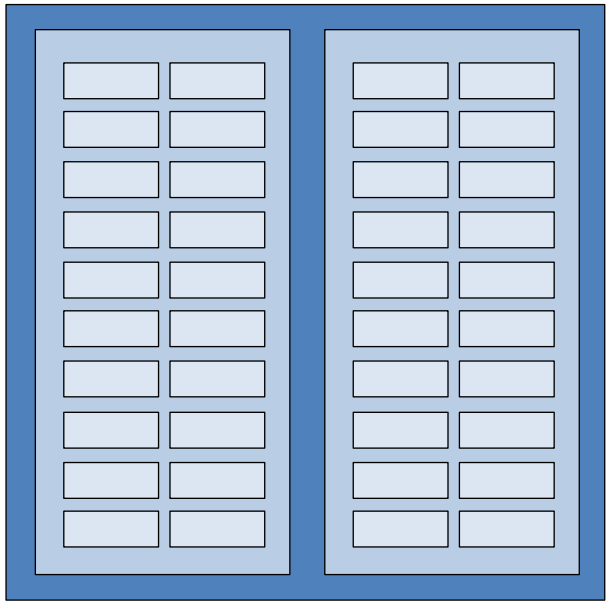
Wait_in_order.py



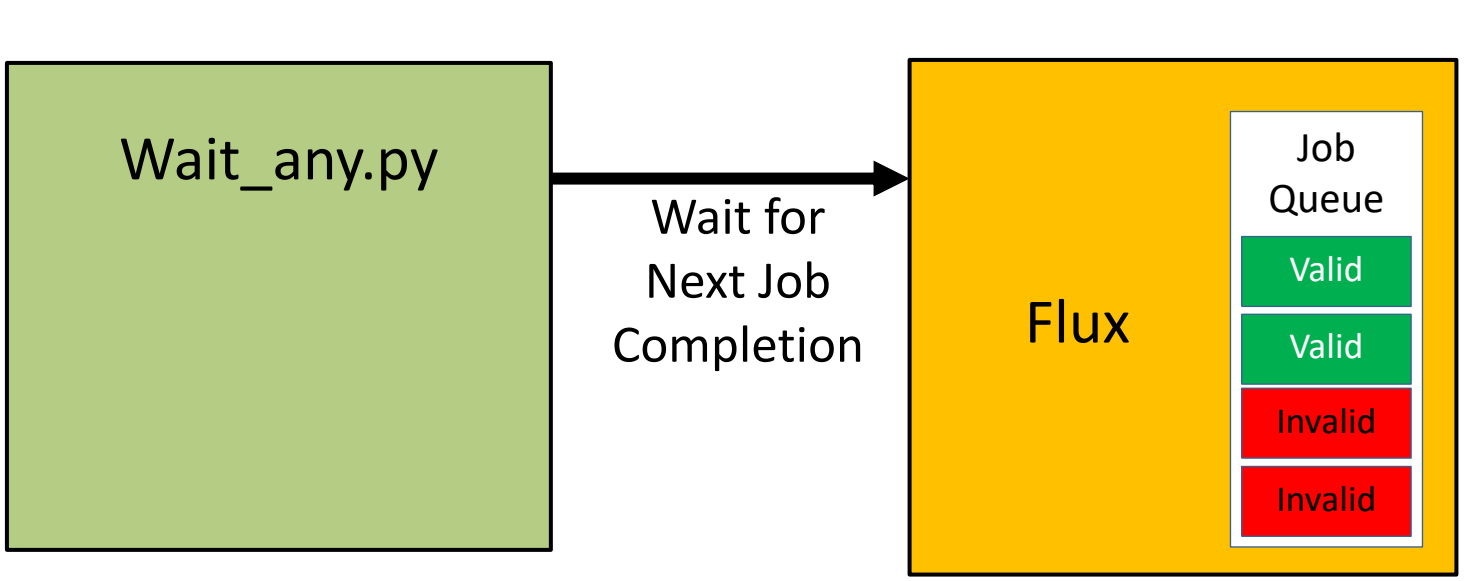


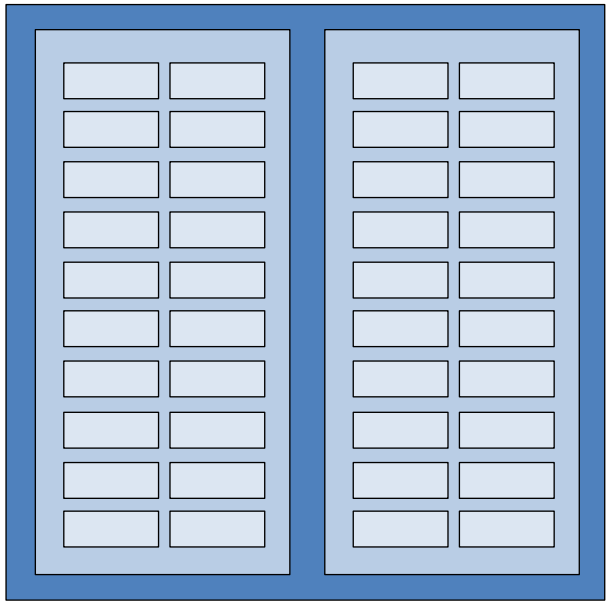
Example 10: Wait Any



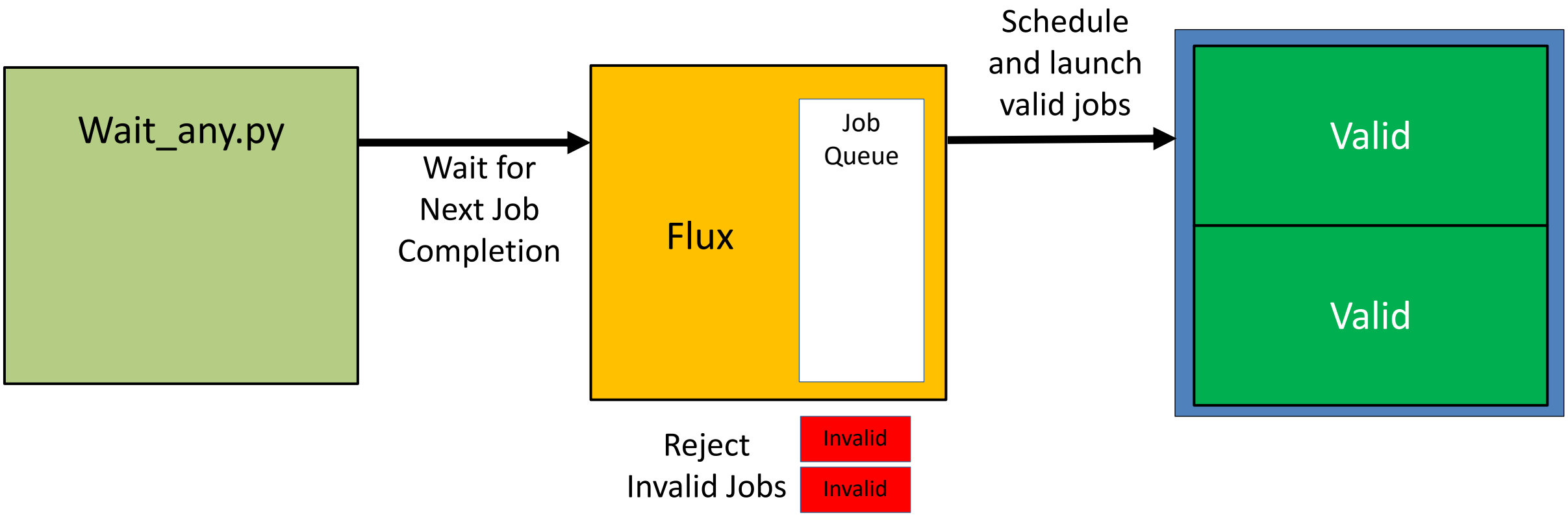


Example 10: Wait Any

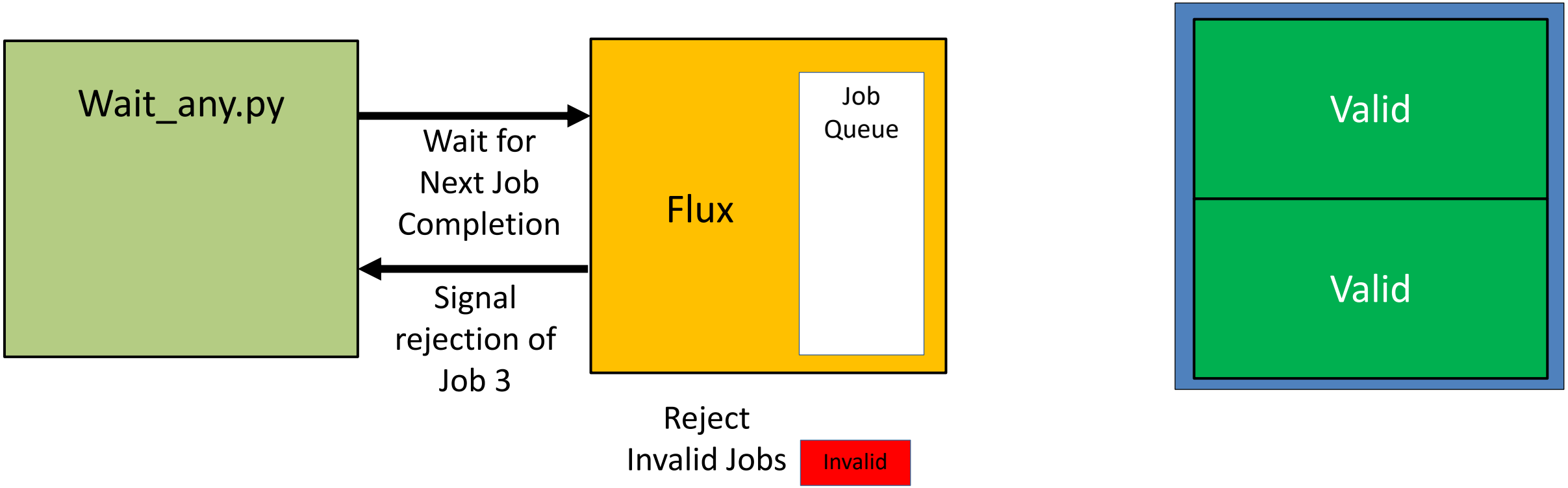




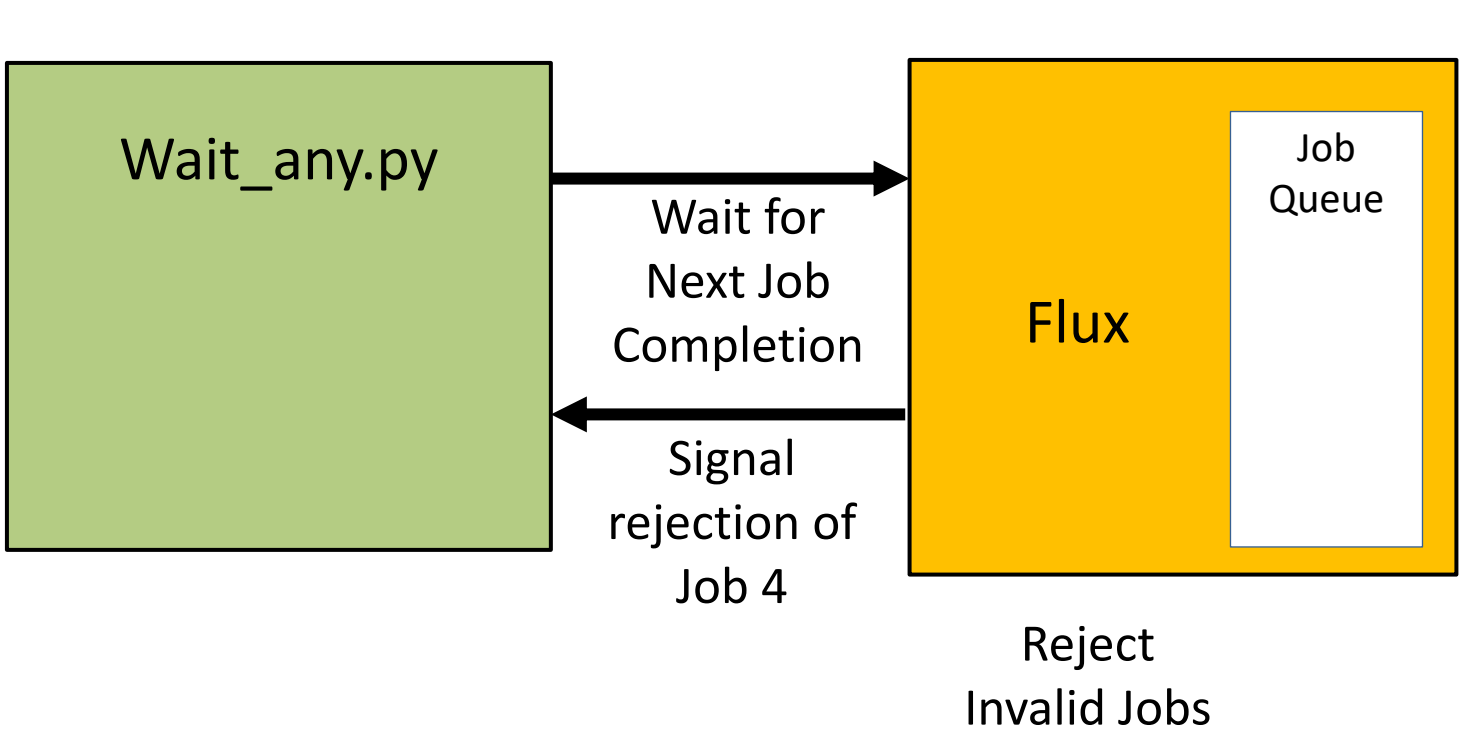
Example 10: Wait Any

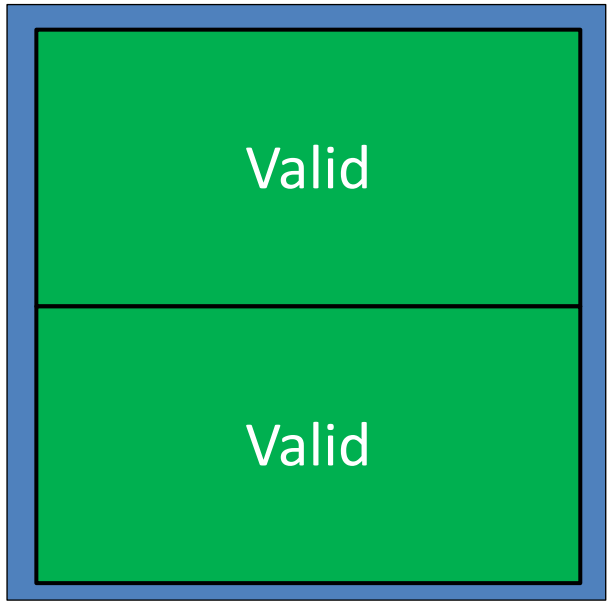


Example 10: Wait Any

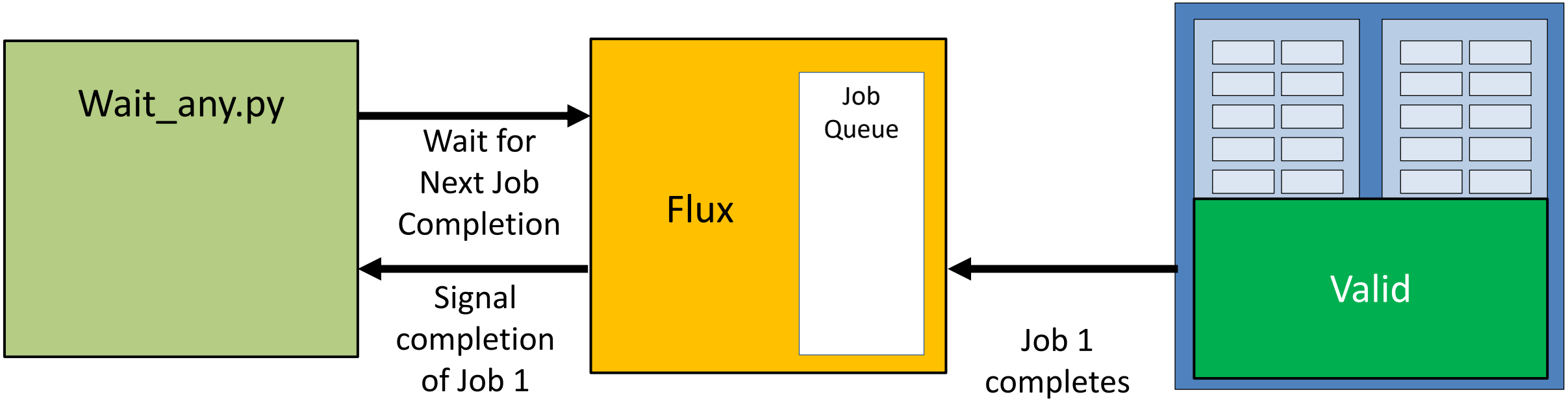


Example 10: Wait Any

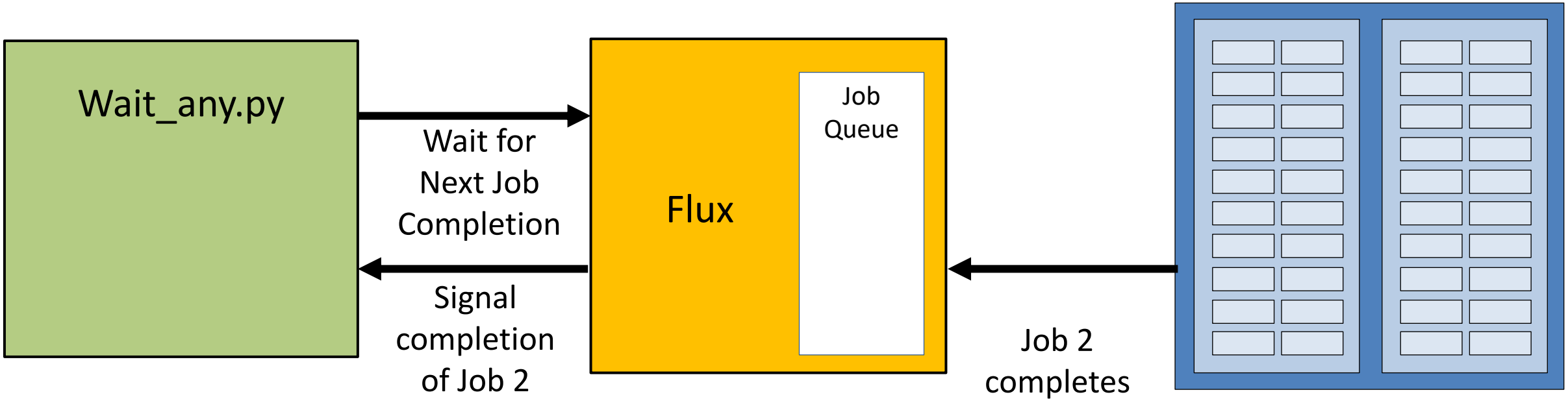




Example 10: Wait Any

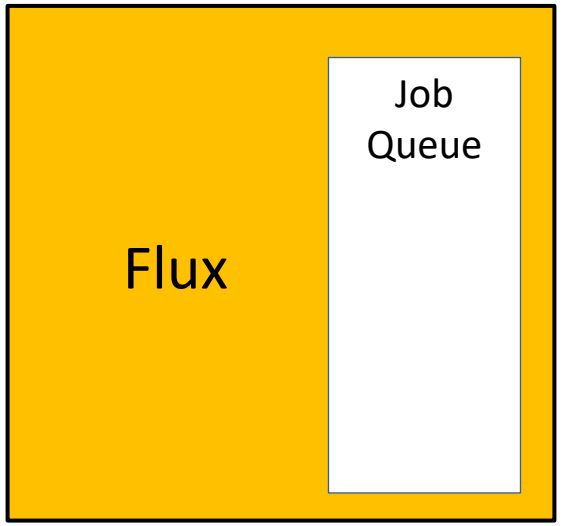


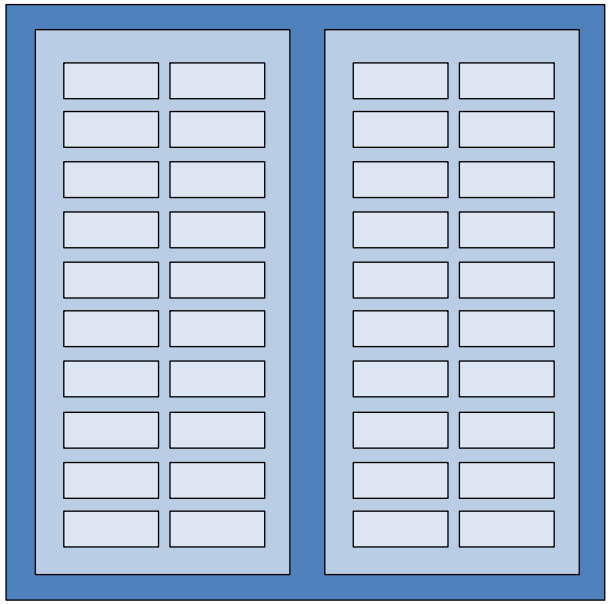
Example 10: Wait Any



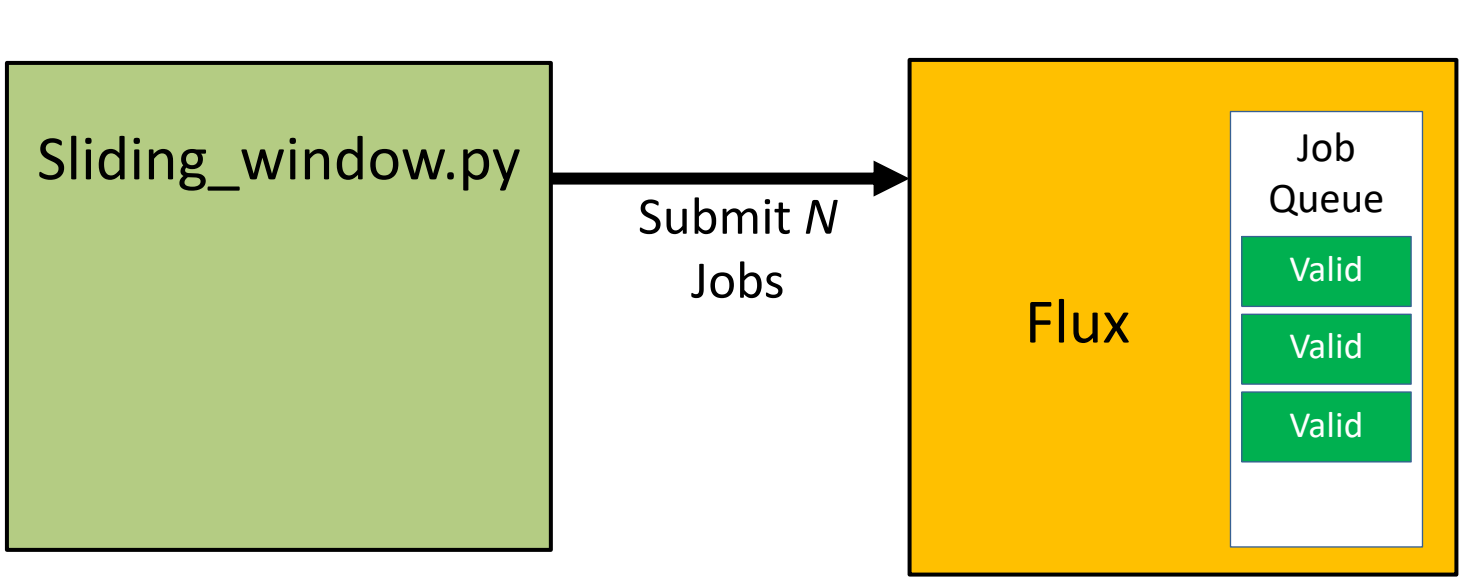
Example 10: Sliding Window

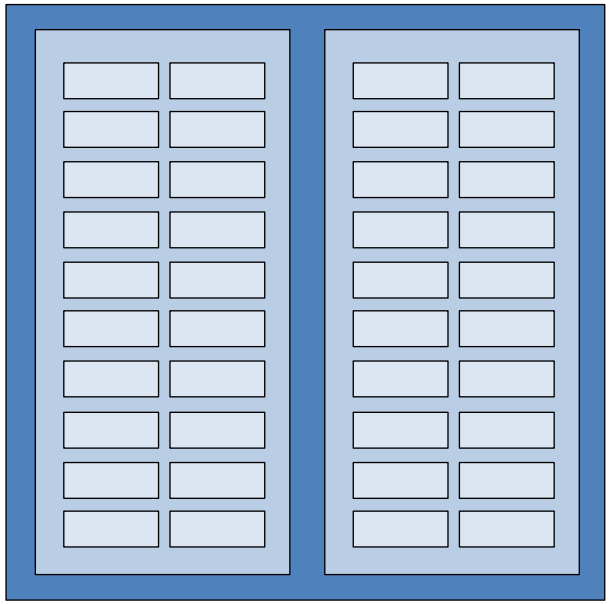
Sliding_window.py



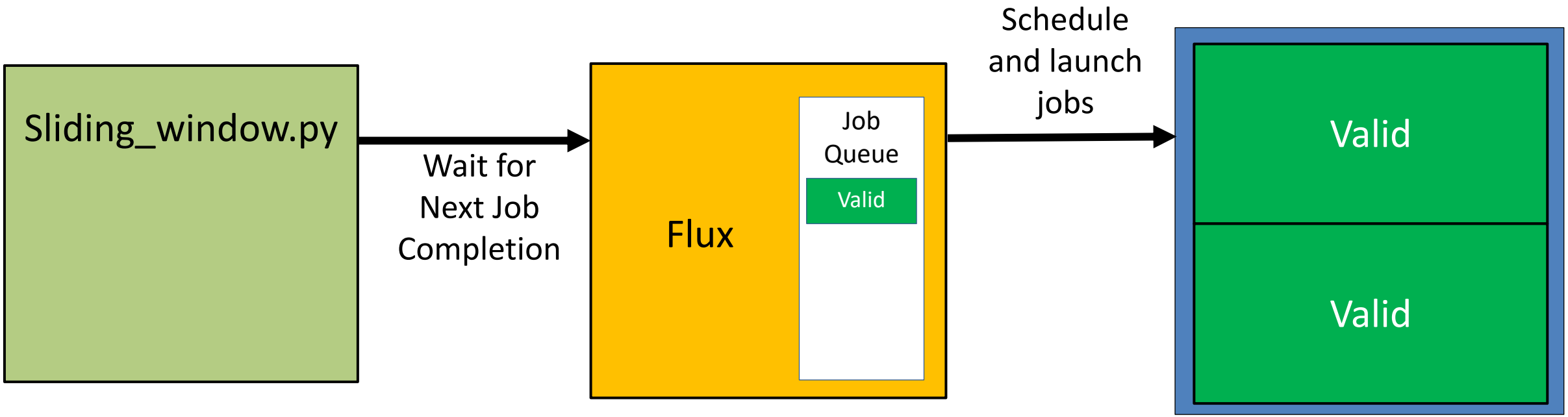


Example 10: Sliding Window

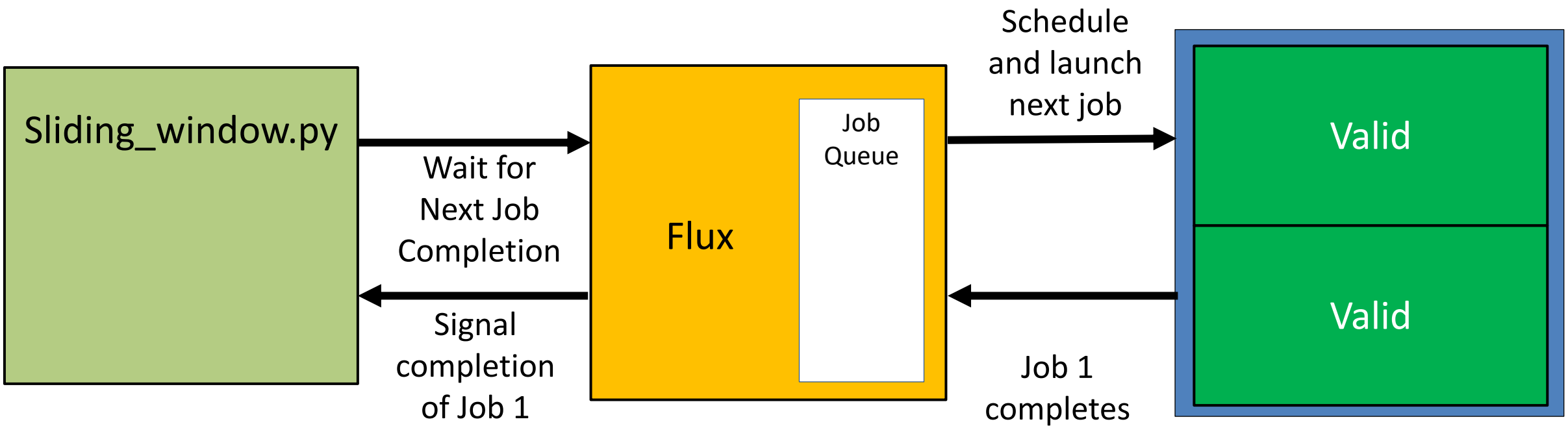




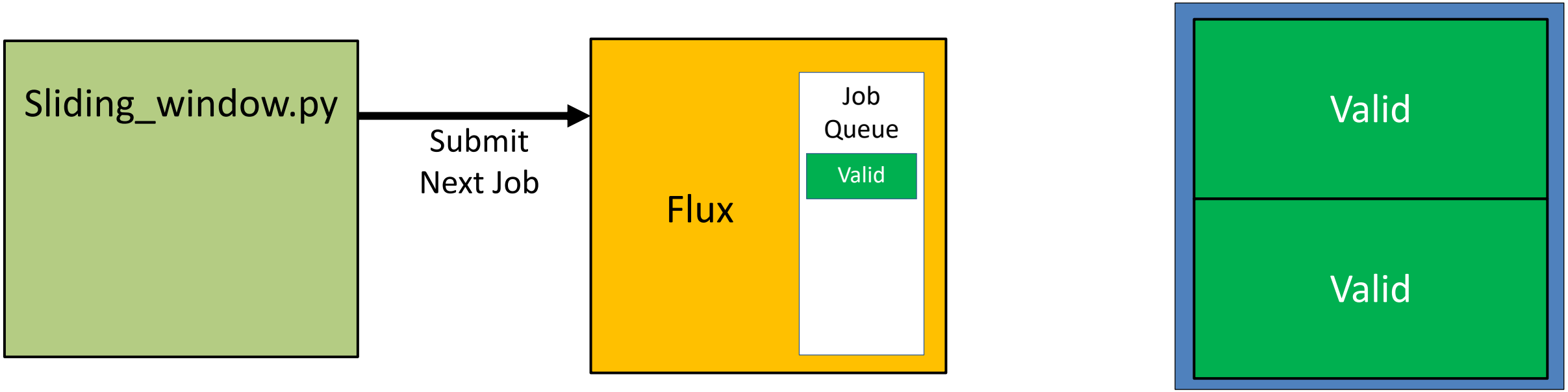
Example 10: Sliding Window



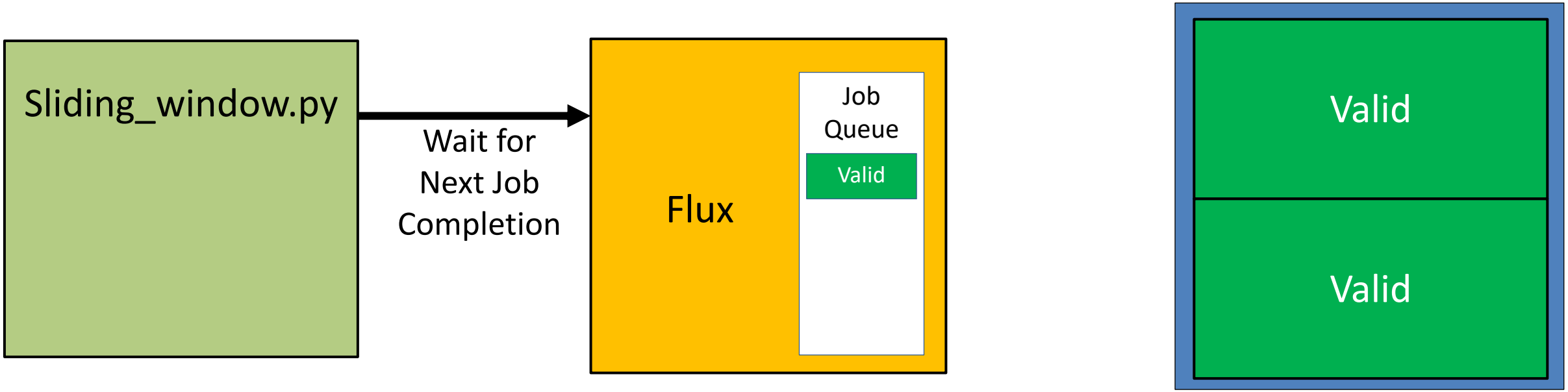
Example 10: Sliding Window



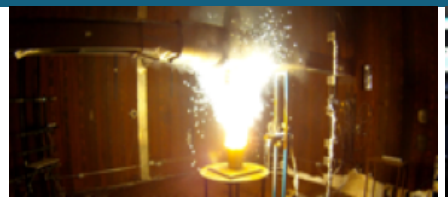
Example 10: Sliding Window

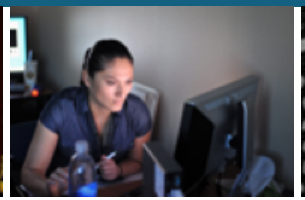


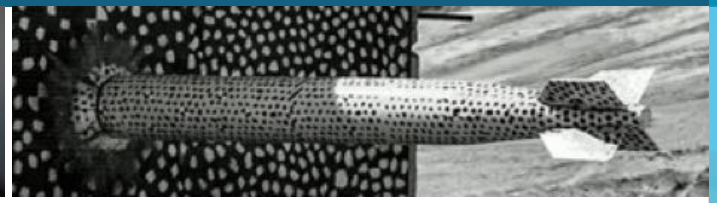
Example 10: Sliding Window



Manage Internal Queue w/ Flux for Common Workflows







Slides & Content Generated by

Anthony Agelastos, Gary Lawson, Dong Ahn, Mark Grondona, Stephen Herbein

Slides Presented by

LLNL Flux Team

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. Approved for unclassified, unlimited release. SAND2020-1164 PE

Acknowledgments

This presentation would not have been possible without the following direct contributions

LLNL
- Dong Ahn
- Jim Garlick
- Mark Grondona
- Stephen Herbein
- Daniel Milroy
- Tapasya Patki
- Francesco Di Natale

SNL
- Anthony Agelastos
- Gary Lawson

… with motivation/need exhibited from the following indirect contributions

SNL
- Micheal Glass (ASC IC/ATDM)
- Brian Adams (ASC Dakota)
- Paul Wolfenbarger (ASC CompSim DevOps)
- Richard Drake (ASC CompSim DevOps)
- Steve Monk (FOUS)
- Ann Gentile (FOUS)

Executive Summary

There are many workflows that require launching and processing jobs in bulk.

- Testing, Uncertainty Quantification (UQ), Verification and Validation (V&V), optimization, parameter study, and Design of Experiments (DoE) are typical activities that require bulk processing.

The activities above are lumped into the following 2 categories.

Unit & Regression Testing

Development teams have tens of thousands of test cases that, ideally, would be run for every commit (many times per day) using low-to-medium scale resources.

Ensemble Processing

Analysts may need to have thousands-to-millions of fast test cases for a study using low-to-large resources.

These cases, individually, are fast but their quantity exceeds what is reasonable for submitting to traditional HPC queues which are optimized for lower numbers of longer-running jobs.

Executive Summary (contd.)

Purpose

- This presentation provides a demonstration of using Flux to address these use cases.

Method

- A simple test case was created for the demonstration.
- A modern Flux installation was created (the version bundled with TOSS is stale).
- Caveats, experiences, results, and methods for extending to more complicated problems are disseminated.
- Discuss observed barrier to entry from someone _not_ on the development team.
- The example is simple to help prospective users understand Flux's small barrier to entry for evaluation.

Results

- Flux installation was robust and easy.
- Getting Flux working as an internal queue for problems is very fast and reliable.
- Information supporting both activity categories is presented.

Installing Flux

Experiences & Notes

- Instructions for "developer" and Spack builds were followed and resulted in success *on the 1st try*.
  - Intel compiler versions 16, 19 & GNU compiler versions 4.8, 8.2 were tried and all resulted in a build of Flux on TOSS.
- Certain OpenMPI versions contain bugs and cause issues with Flux.
  - Bug: https://github.com/open-mpi/ompi/issues/6730
  - PR: https://github.com/open-mpi/ompi/pull/6764
  - Currently, avoid most versions >= 3.1.0 (issue confirmed on 3.1.0, 3.1.3, 4.0.0, 4.0.1)
  - Test case was performed with OpenMPI 1.10.
- Test suite worked as expected and displayed passing/failing tests.

Compiling and installing Flux was easy & reliable on TOSS/RHEL.

Unit & Regression Test Case: Description

`mpi_greet_after_sleep` is a simple MPI application that does the following:

- `MPI_Barrier()` (syncs ranks after initialization)
- `sleep(20)` (sleeps for 20 sec.)
- `get_time_of_day()` (gets current date & time)
- `MPI_Barrier()` (syncs ranks)
- rank 0 process spits out location and timing info

```
$ srun -N 2 -n 72 ./mpi_greet_after_sleep
Hello WORLD! I am node ama42 with 72 ranks
Started at: Mon Jan 27 22:13:55 2020
Finished at: Mon Jan 27 22:14:15 2020
```

Simple application is used to create example test case

Unit & Regression Test Case: Using with Flux CLI

Get an allocation on 4 nodes

- `salloc –N 4 --time=4:00:00`

Tell Flux you want backfill

- `export FLUX_QMANAGER_OPTIONS="queue-policy=easy"`

Start Flux upon the 4 nodes with 1 Flux instance per node

- `srun --pty --mpi=none -N 4 -n 4 flux start -o,-S,log-filename=mpi_greet_after_sleep.log`

Submit a case

- `flux mini submit -N 4 -n 144 -t 30 ./mpi_greet_after_sleep >>submit_id 2>&1`

Get job output if desired

- `flux job attach \`cat submit_id\``

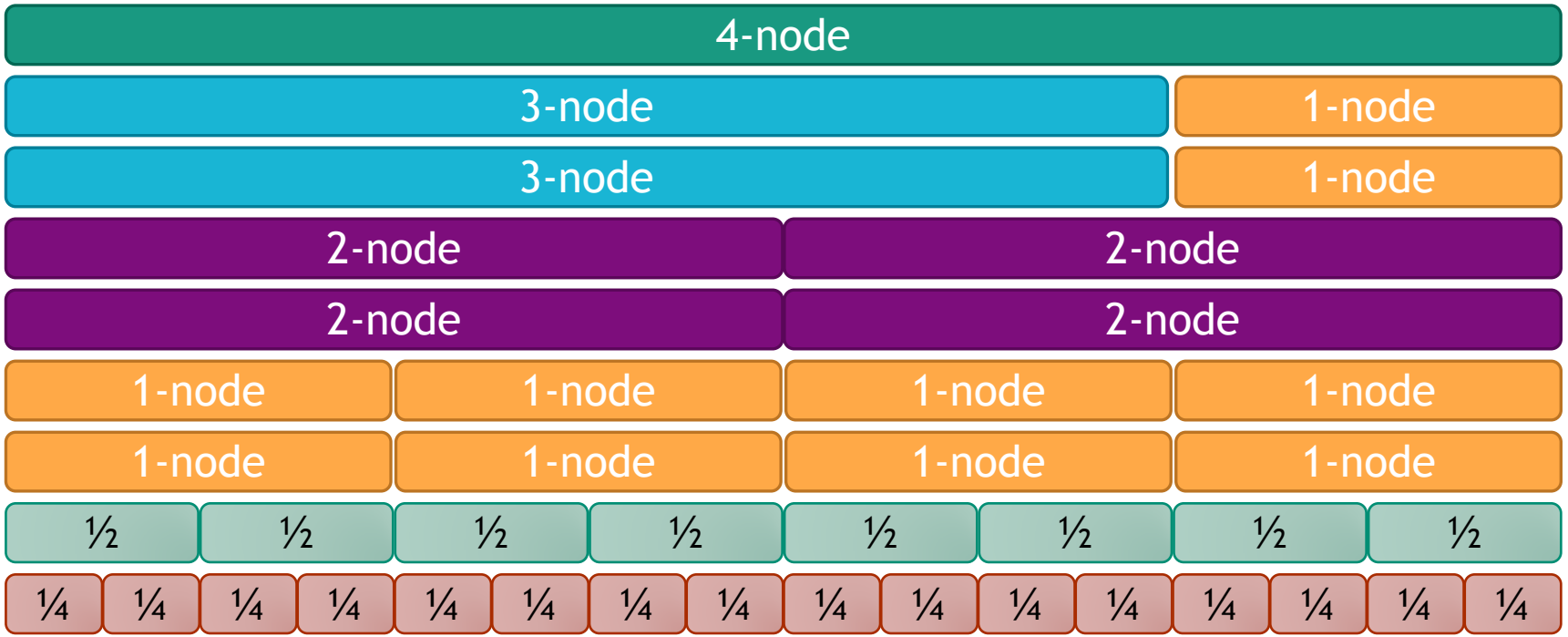
This is a high-level overview of the ***entire*** process! There are not many steps!

Unit & Regression Test Case: Workload

The workload contains the following 41 jobs:

- 1x 4-node job
- 2x 3-node jobs
- 4x 2-node jobs
- 10x 1-node jobs
- 8x ½-node jobs
- 16x ¼-node jobs

The workload was designed to take ≅3:00 if the entire 4-node allocation was wholly utilized.

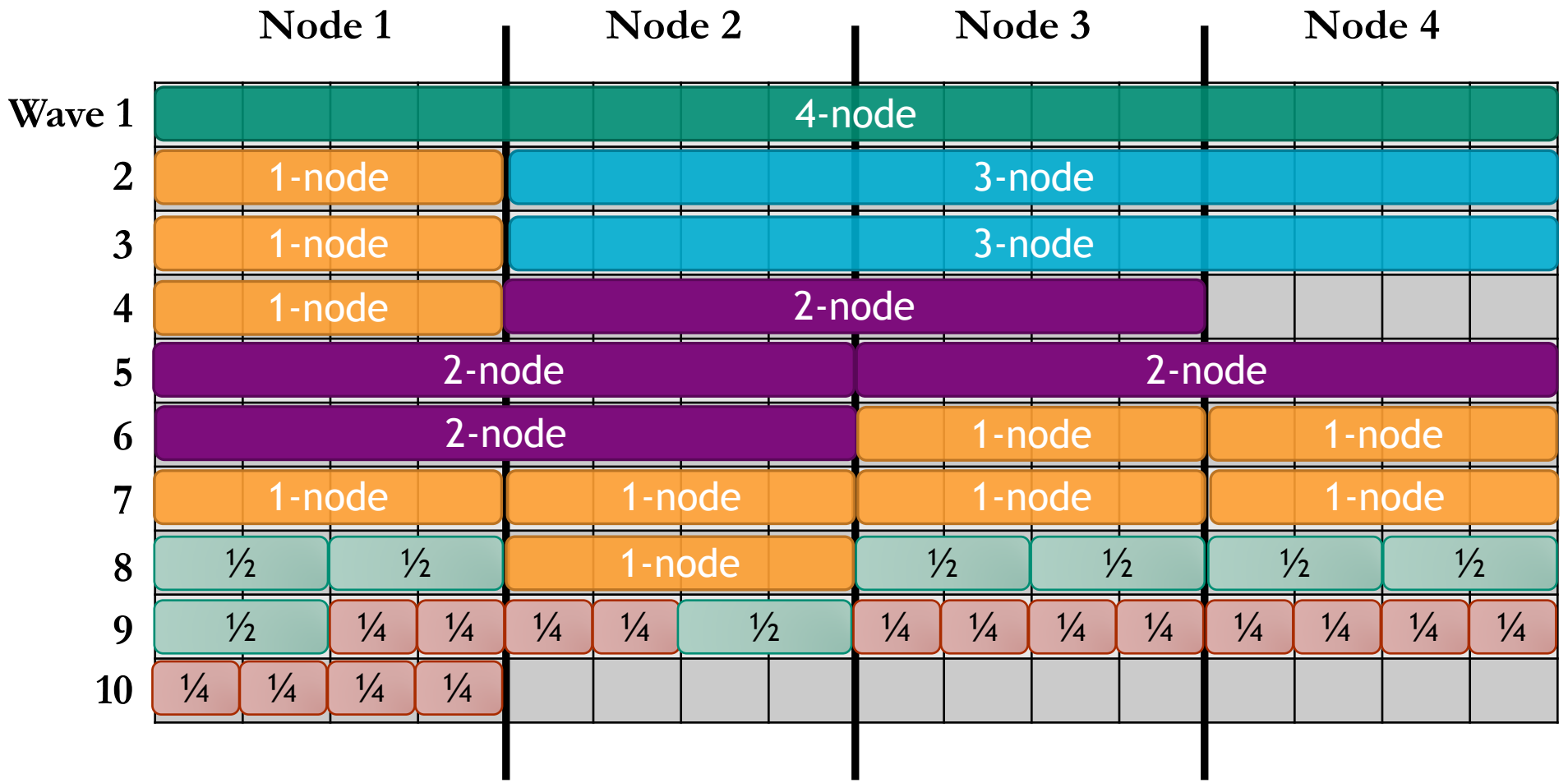


Workload contains jobs that only require a fraction of a node's resources up to 4 nodes.

Unit & Regression Test Case: Ordered Submission

Submission Order (*# nodes*)

1.	4	22.	½
2.	3	23.	½
3.	3	24.	½
4.	2	25.	½
5.	2	26.	¼
6.	2	27.	¼
7.	2	28.	¼
8.	1	29.	¼
9.	1	30.	¼
10.	1	31.	¼
11.	1	32.	¼
12.	1	33.	¼
13.	1	34.	¼
14.	1	35.	¼
15.	1	36.	¼
16.	1	37.	¼
17.	1	38.	¼
18.	½	39.	¼
19.	½	40.	¼
20.	½	41.	¼
21.	½		

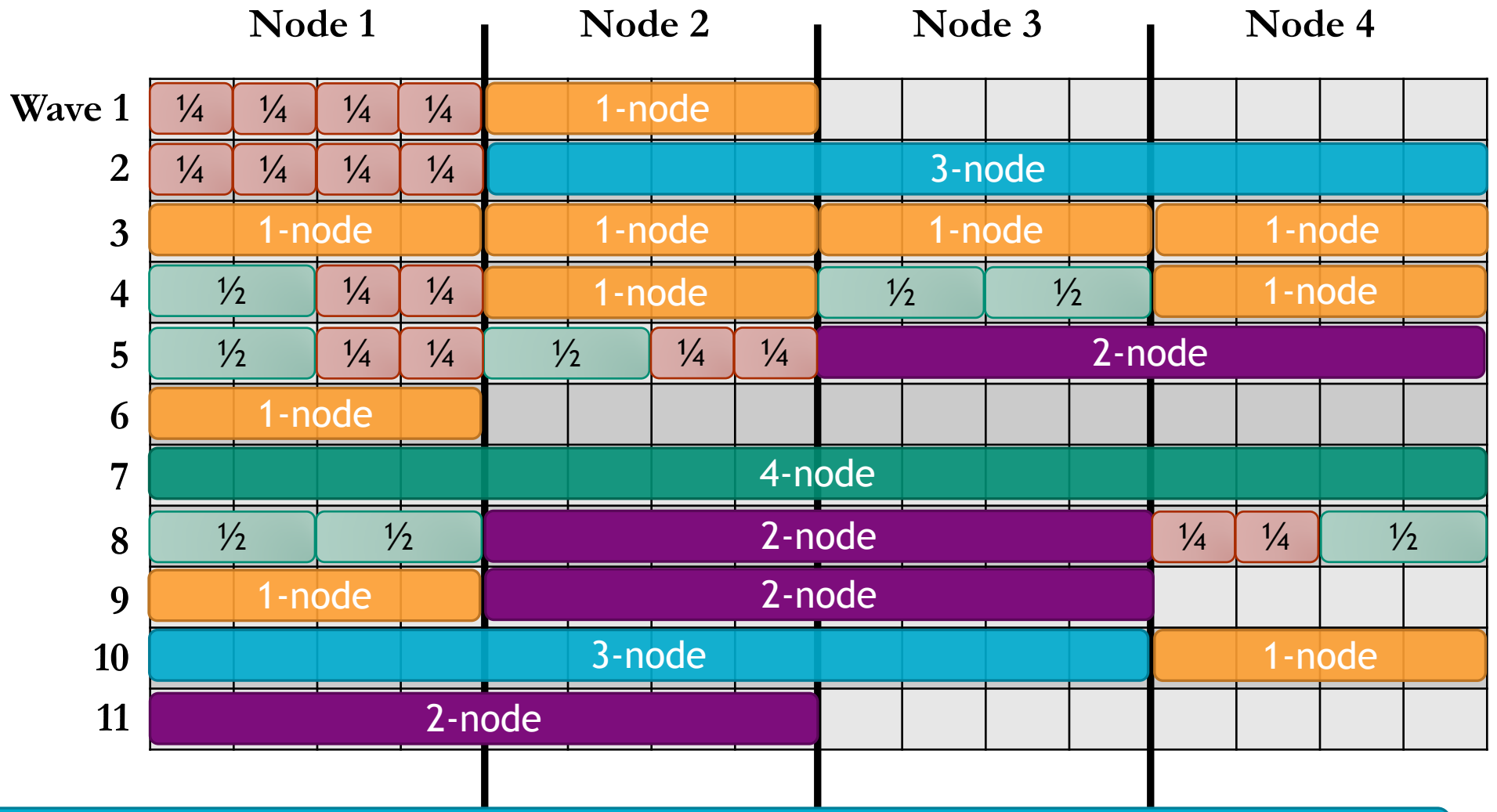


The packing was nearly optimal; runtime variations and inconsistent start times are cause for extra wave.

Unit & Regression Test Case: Random Submission

Submission Order (*# nodes*)

1.	¼	22.	½
2.	1	23.	2
3.	¼	24.	¼
4.	¼	25.	½
5.	¼	26.	1
6.	3	27.	4
7.	¼	28.	¼
8.	1	29.	¼
9.	¼	30.	¼
10.	¼	31.	2
11.	1	32.	½
12.	1	33.	½
13.	1	34.	¼
14.	¼	35.	2
15.	½	36.	¼
16.	¼	37.	1
17.	1	38.	3
18.	½	39.	1
19.	1	40.	½
20.	¼	41.	2
21.	½		



Packing was less optimal due to queue depth optimization + job start/stop not synced.

Unit & Regression Test Case: Random Submission (contd.)

Extra Specialization

- Can specify queue depth optimization parameter via:
  `FLUX_QMANAGER_OPTIONS="queue-policy=easy queue-params=queue-depth=<#>"`
- Queue depth = 2 had 12 waves
- Queue depth = 20 had 11 waves
- Queue depth = 41 had 10 waves
- Queue depth = 100 had 10 waves
- More detailed discussion regarding this at https://github.com/flux-framework/flux-sched/issues/572

Simple tuning of the queue depth reduced the amount of waves down to 10.

Enhancing the Test Case: Submitting Scripts

OLD: Submit a case (directly calling application `mpi_greet_after_sleep`)

- `flux mini submit -N 4 -n 144 -t 30 ./mpi_greet_after_sleep`

NEW: Submit a case (calling script `internal_script.sh`)

- `flux mini submit -N 4 -n 4 flux start ./internal_script.sh`

NEW: Within script (`internal_script.sh`) execute parallel application with this

- `flux mini run -N 4 -n 144 -t 30 ./mpi_greet_after_sleep`

Scripts can be run as well and each script can run different sized applications that Flux will manage.

Limitations with Large Ensemble Processing

Flux Interface

- Flux supports a command line interface (which was leveraged for the technology demonstration herein).
- Flux also provides a Python API and other methods for ingesting jobs.
  - Python API is the preferred method for interacting with Flux for medium-to-complicated workflows.

Issue with Submitting Large Ensembles

- Methods described herein have a "slow" ingest rate of 4-8 job/s. This would require almost 2 days for 1 million jobs, which may not work for such a large ensemble. ☹
- Thankfully, they have bulk loaders and methods directly accessible with their Python API to drastically increase this to over 600 job/s (needing about 30 minutes to submit a full stack of 1 million jobs). ☺
  - Refer to discussions within https://github.com/flux-framework/flux-sched/issues/573 for more details.

Flux has a diverse set of accessor methods that can tailor to many workflows.

Conclusions & Future Work

Conclusions

- Flux was easy to install, run, and extend to suit the simple example herein.
- Flux is very easy to use for managing internal queues and provides good behaviors right out of the box with minimal tinkering.
- Flux is the most comprehensive solution tested to date for dynamic workflows consisting of heterogeneous jobs.

Future Work

- SNL Dakota team is crafting a dynamic, heterogeneous optimization case that involves real simulation codes being driven by Dakota. Flux will be put in the loop to manage the resource allocations as a more expanded example case to share with the broader Dakota user base.
- Data herein will be disseminated to various teams whose testing strategies can be enhanced with Flux.

Using Flux to Improve the Usability, Performance, and Software Complexity of UQ Pipeline

Flux Tutorial at ECP Annual Meeting, Feb 6, 2020

Dong H. Ahn, Stephen Herbein, James Corbett, and David M. Domyancic

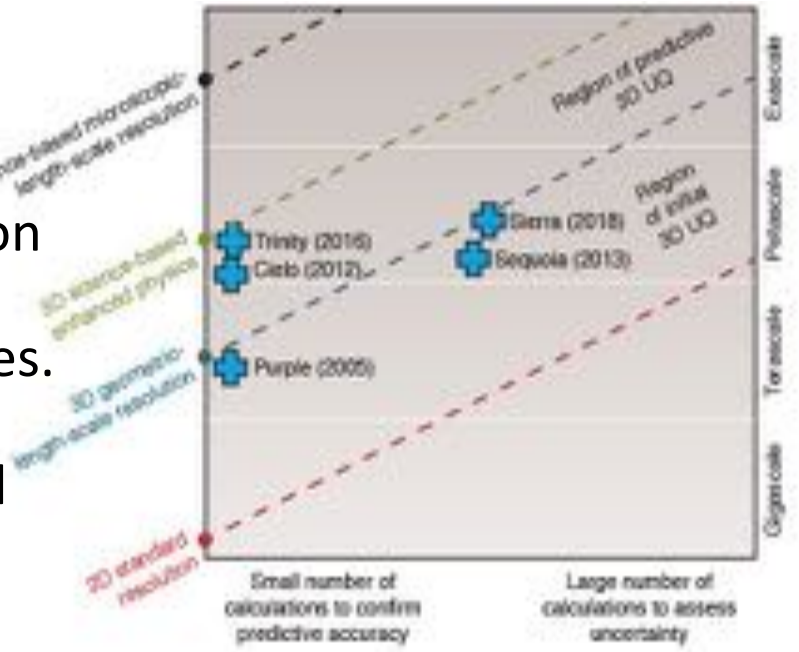
LLNL-PRES- 804364

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

Lawrence Livermore National Laboratory

Uncertainty Quantification (UQ) workflows are critically important for the DOE.

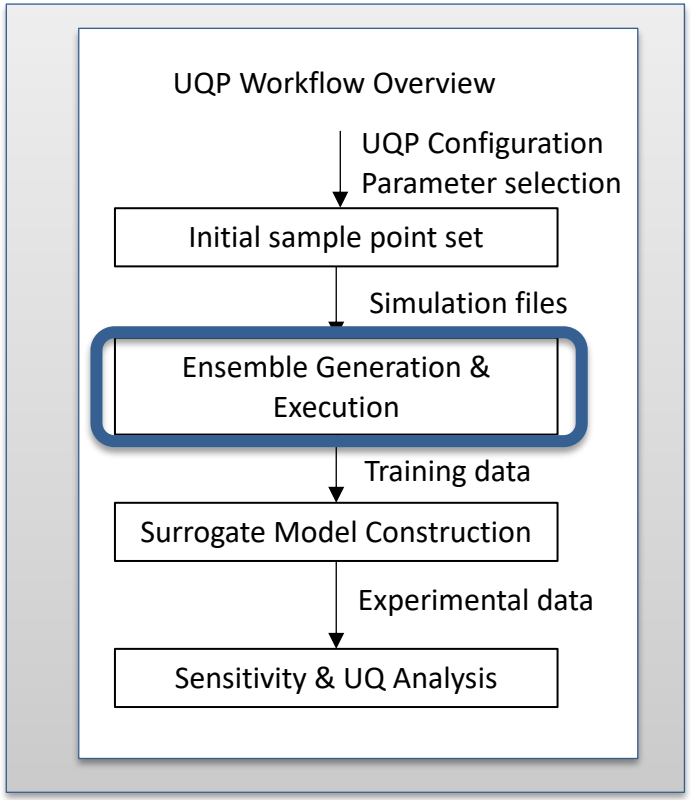
- UQ is the quantitative characterization and reduction of uncertainty in simulation applications
- UQ studies to understand our simulation codes.
  - Evaluate input parameter impacts on outcomes.
  - Assess likelihood of certain outcomes.
  - Make predictions about untested experiments with uncertainty based on relevant previously completed experiments



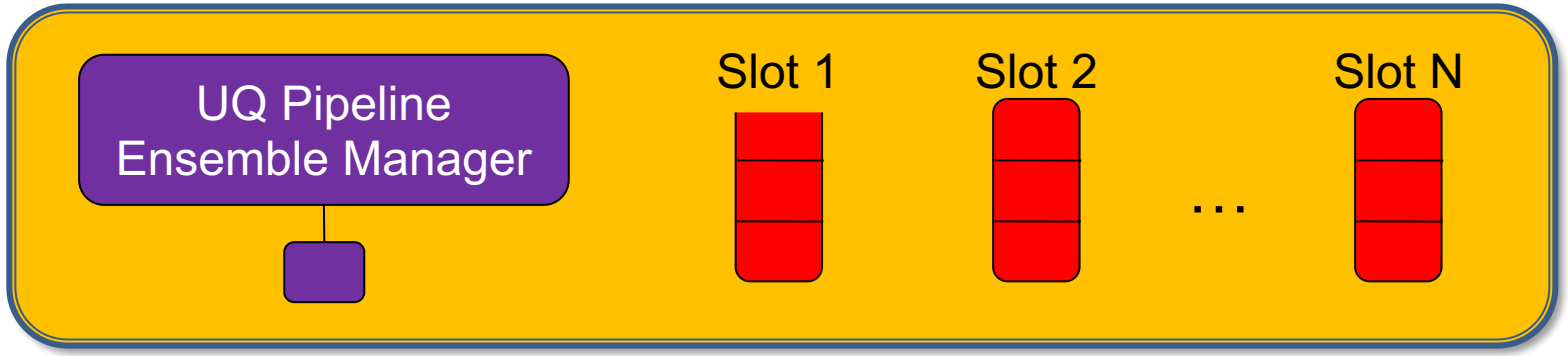
https://wci.llnl.gov/simulation/computer-codes/uncertainty-quantification

UQ Pipeline (UQP): LLNL's marquee UQ workflow management system.

- Account for ~50.9 million CPU hours each year at LLNL.
- Used in daily for WCI and NARAC programmatic work
- Composed of multiple capabilities
  - Sampling methods
  - Ensemble generation and management
  - Surrogate model methods
  - Sensitivity Analysis (quantify the influential input parameters)
  - UQ analysis methods
- The pipeline's sampling and ensemble generation capabilities are also used to generate simple parameter studies.



The UQP ensemble manager is the enabling component for efficiently running ensembles.

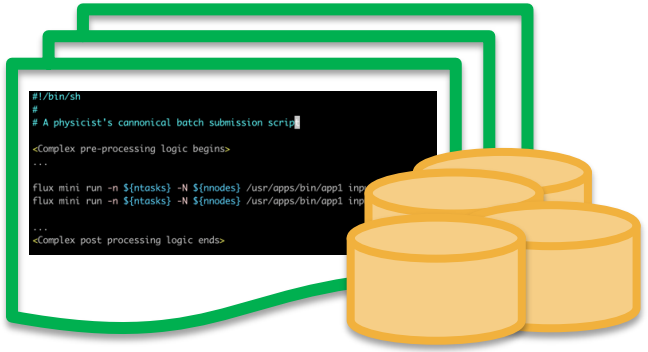


- Executes on the first node and subdivides remaining allocated nodes into N uniform “slots”.
- Each slot executes one of M ensemble simulations using the parallel job launcher (e.g., srun).
- The ensemble manager dynamically manages the slots.
- The ensemble manager can restart/terminate simulations, reduce/disable slots, or terminate the entire ensemble.
- Users can pass command directives back to the ensemble manager (e.g. restart, terminate, …).

UQP desires to improve its ensemble manager by solving three concrete problem using Flux.

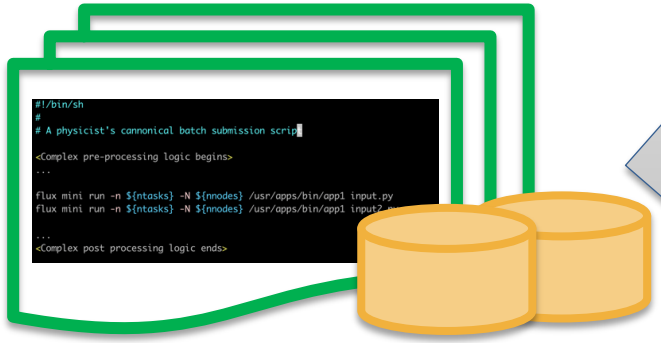
- An ability to perform quick exploratory experiments and to generate a large ensemble from it.
  - by allowing for running users' batch scripts directly
  - Improve usability and performance.
- Simplify the software architecture.
  - Reduce software complexity and increase portability.
- Position the ensemble manager for extremely high throughput workflows.
  - Increase scalability and exascale readiness.
- Detail follows...

Flux uniquely enables UQP to map better to the representative workflows of UQ scientists.



- Previously simulated results and batch submission scripts
  - Simulation campaign
  - UQ jobs

- Quick exploratory simulation experiments on some changes
  - No high-throughput support and other tool support



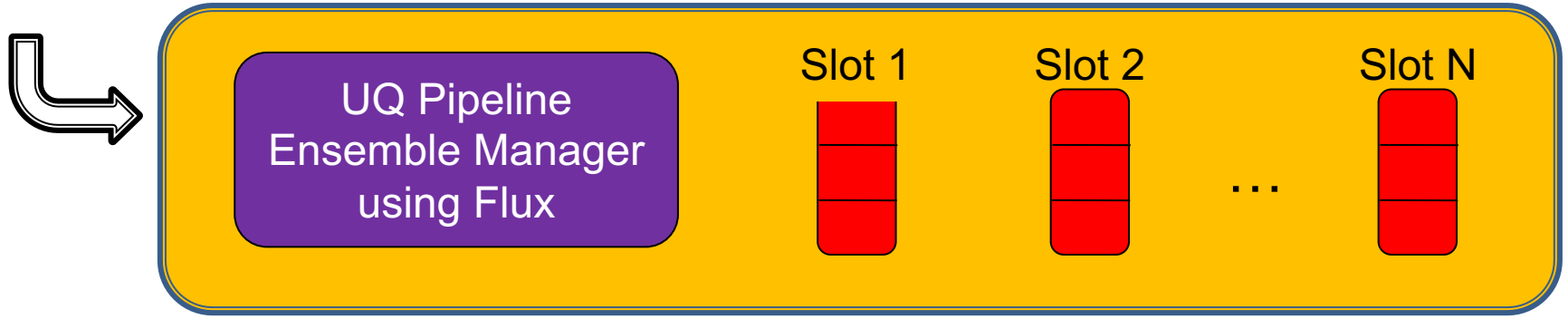
- Large ensemble
  - Replicate the logic in batch script
  - Script rewrite to UQP non-trivial

- No explicit support for the next cycle

Our fully hierarchical scheduling model naturally maps to the representative UQ workflows.

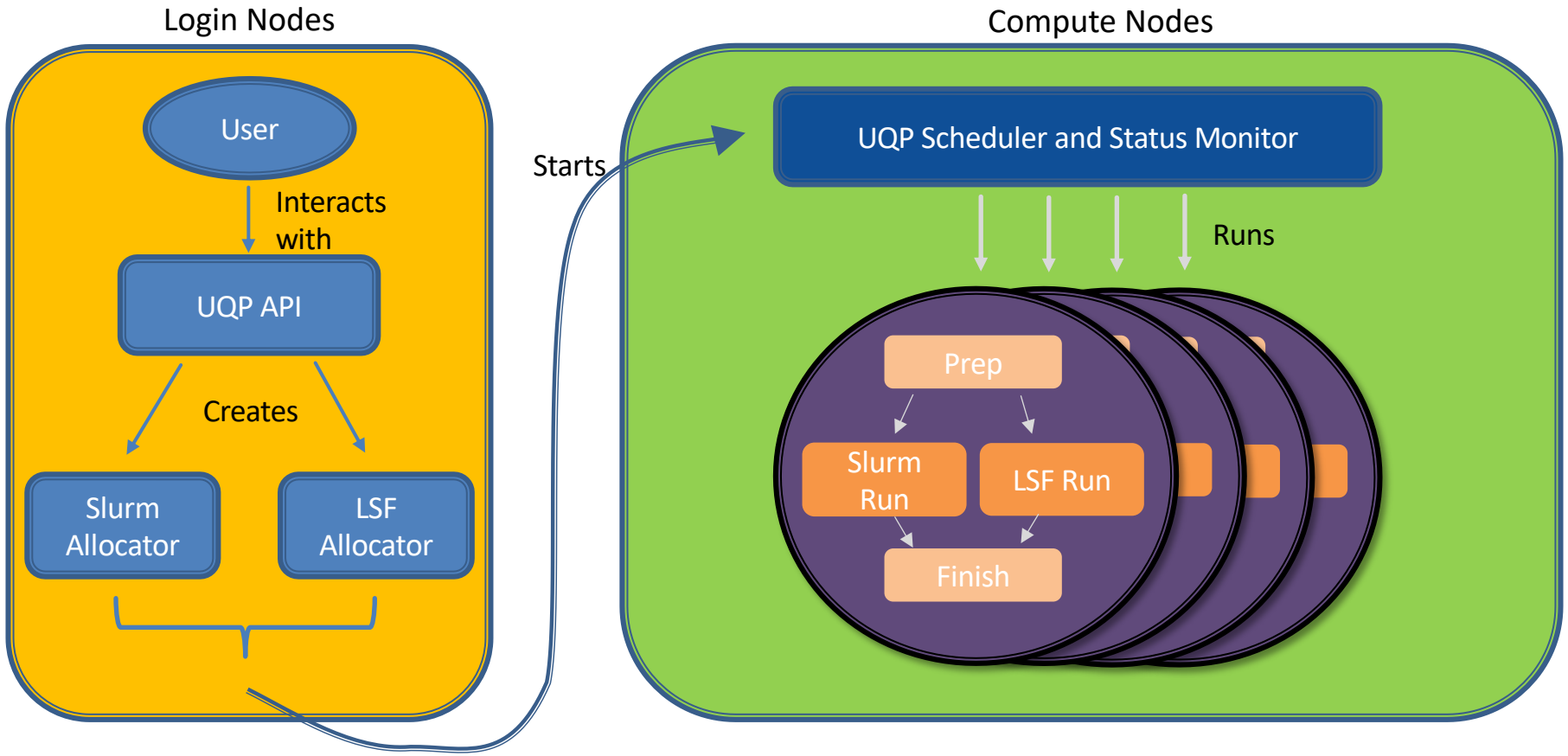
```
#!/bin/sh
#
# A physicist's cannonical batch submission scrip

<Complex pre-processing logic begins>
...

flux mini run -n ${ntasks} -N ${nnodes} /usr/apps/bin/app1 input.py
flux mini run -n ${ntasks} -N ${nnodes} /usr/apps/bin/app1 input2.py

...
<Complex post processing logic ends>
```

script.sh

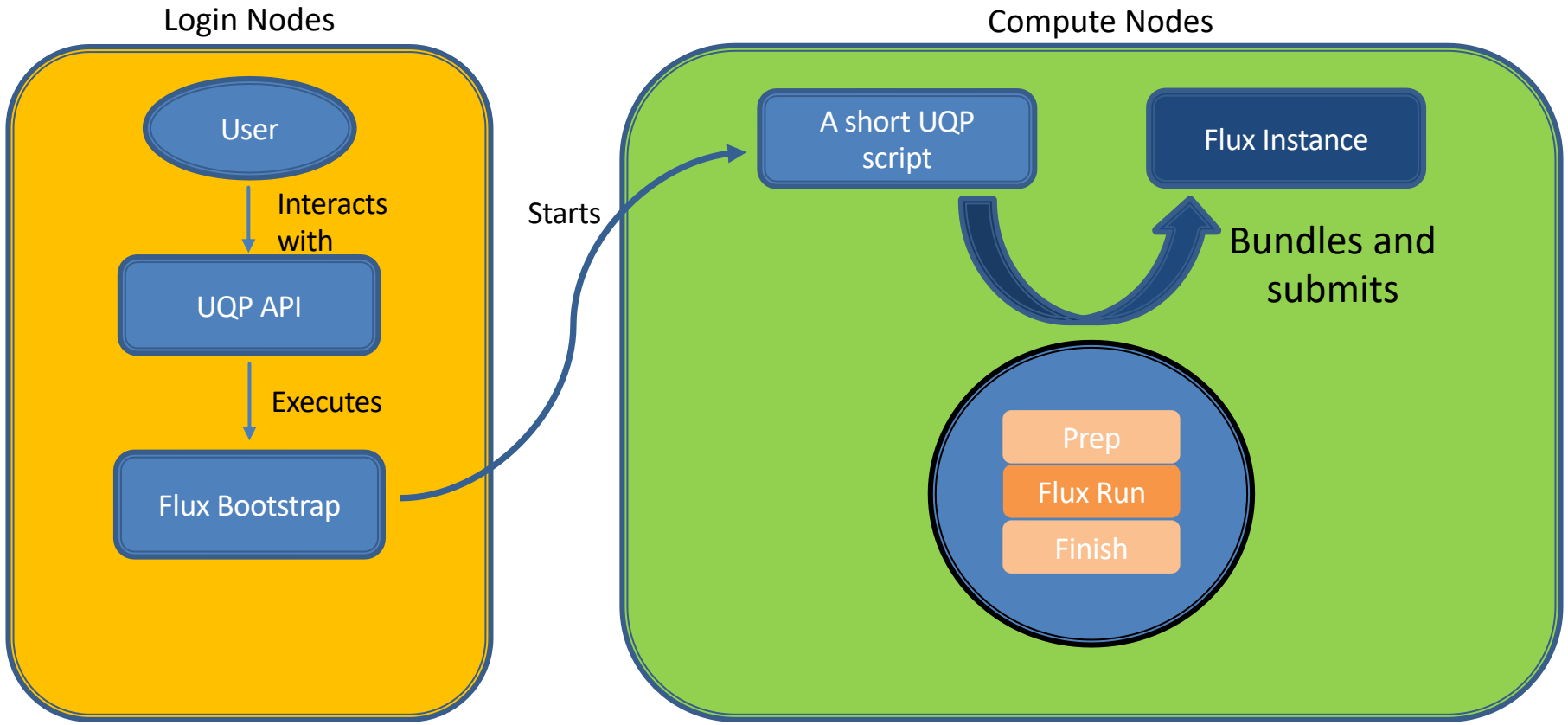


- As simple as: *% flux mini submit -N 1 -n1* ***flux start ./script.sh***
- A prototype implemented to get more user feedback

The software architect of the ensemble manager with the traditional batch scheduling approach.

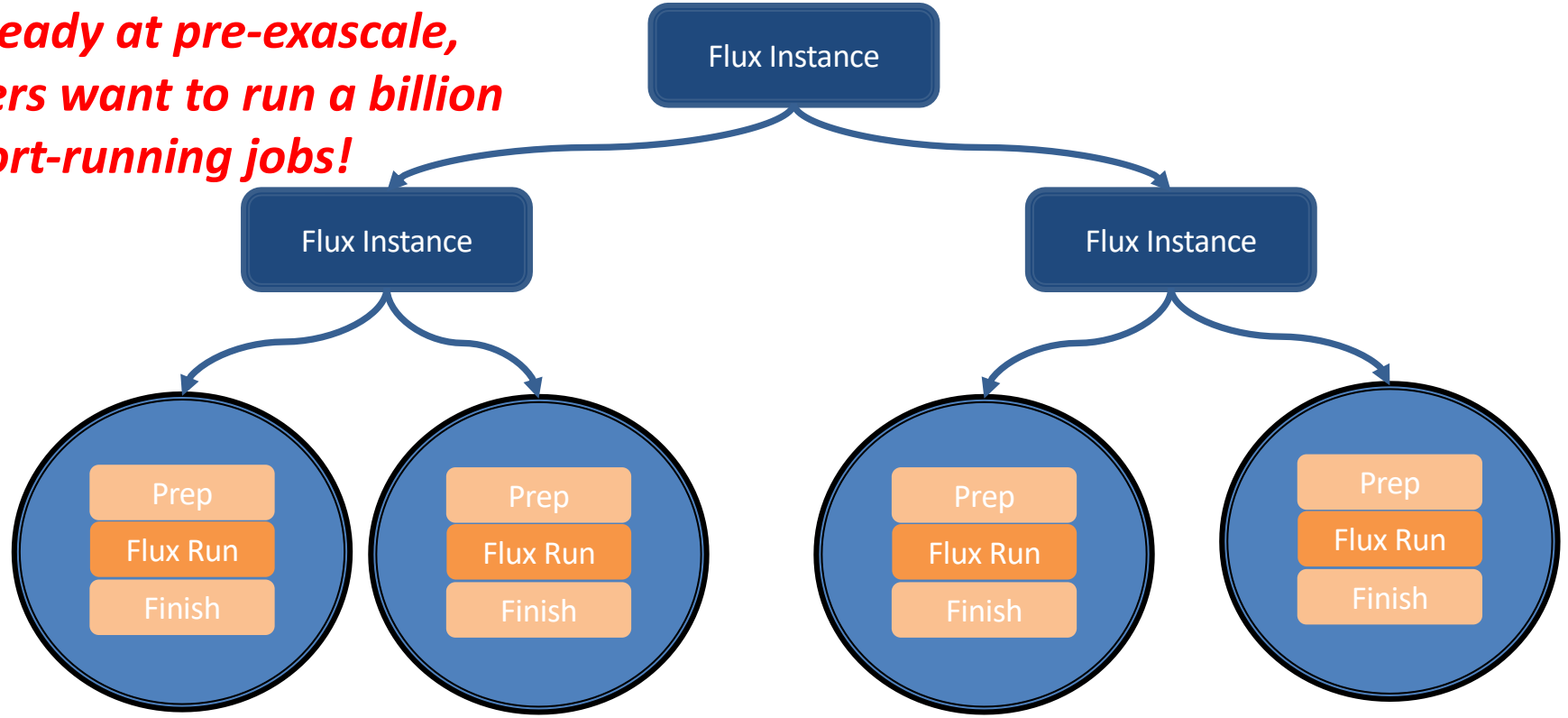


Flux significantly reduces the software complexity of UQP and improves its portability.



Flux positions UQP for extremely high-throughput workflows towards exascale computing.

Already at pre-exascale, users want to run a billion short-running jobs!



- As simple as: `flux tree -N 4 –n4 –T2 -J4 ./script.sh`
- E.g. deeper tree by specifying the hierarchical policy (-T 2x2x2)

The novel scheduling model of Flux is uniquely enabling UQP.

- DOE is increasingly interested in uncertainty quantification (UQ).
- UQP is a highly important workflow management tool for LLNL.
- Flux-UQP co-design is enabling UQP in 3 unprecedented ways.
  - Enable UPQ to run users' batch scripts with no modification.
  - Simplify the software architecture of UQP.
  - Position UPQ for extremely high throughput workloads.

Future Flux developments: not nebulous, but a bit cloudy

2020 ECP Annual Meeting Tutorial

Anthony Agelastos, Dong Ahn, Francesco Di Natale, Stephen Herbein, **Daniel Milroy**, Tapasya Patki

02/06/2020

LLNL-PRES-804364

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

Lawrence Livermore National Laboratory

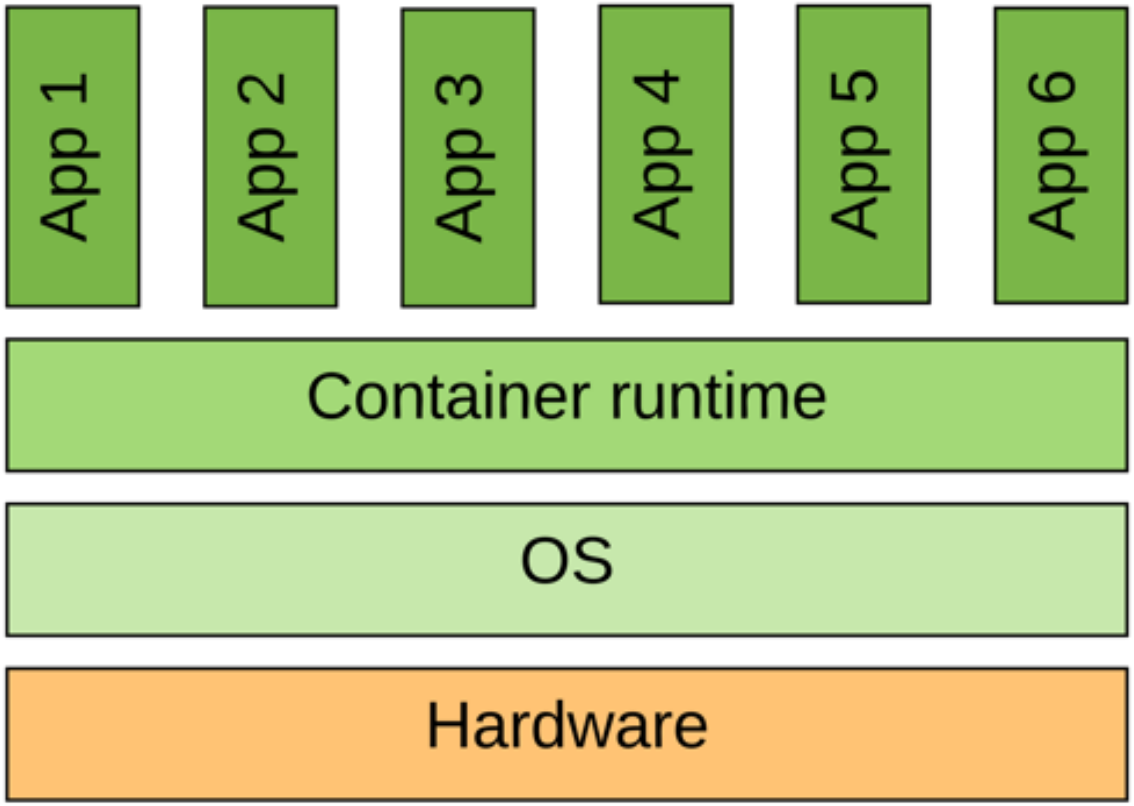
Outline

- Containers are becoming common in HPC
- Kubernetes (k8s): container management
- Challenges for converged k8s-HPC
- Flux in the cloud

Containerized computing in widespread use

A lot of interest in containers because:

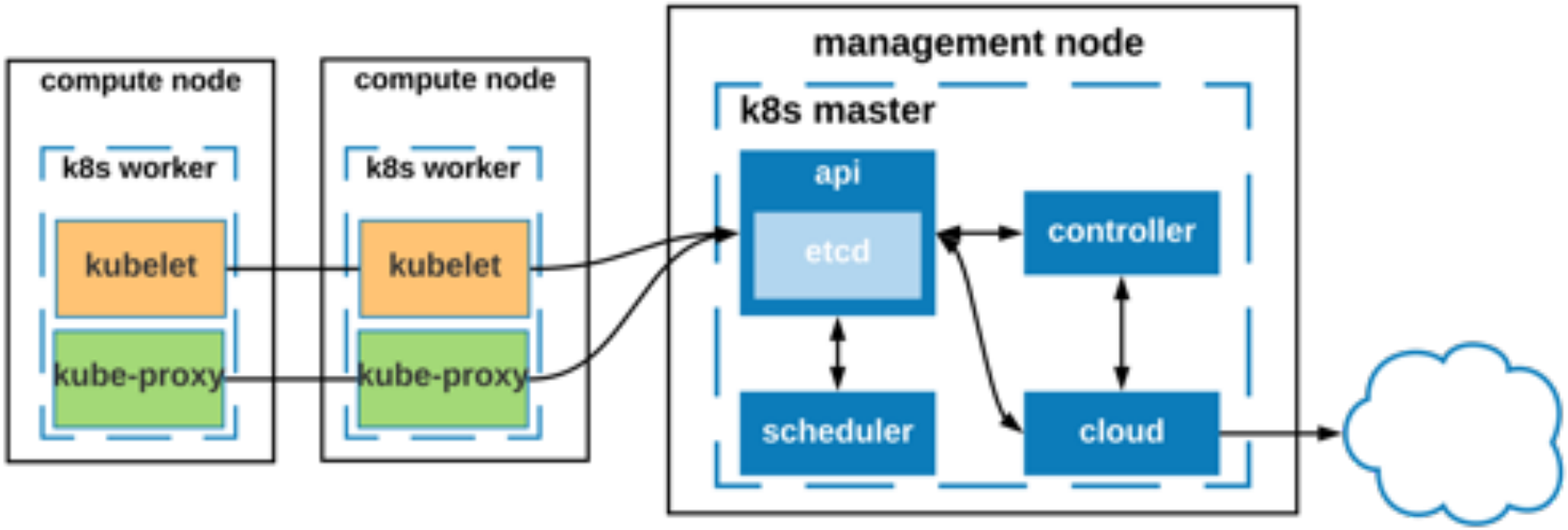
- portable
  - packages software and dependencies
- charge by use
- lightweight
  - share OS kernel
  - no need for OS virtualization
- low/negligible performance impact
  - (CANOPIE 2019, etc.)
- elasticity



Managing containers

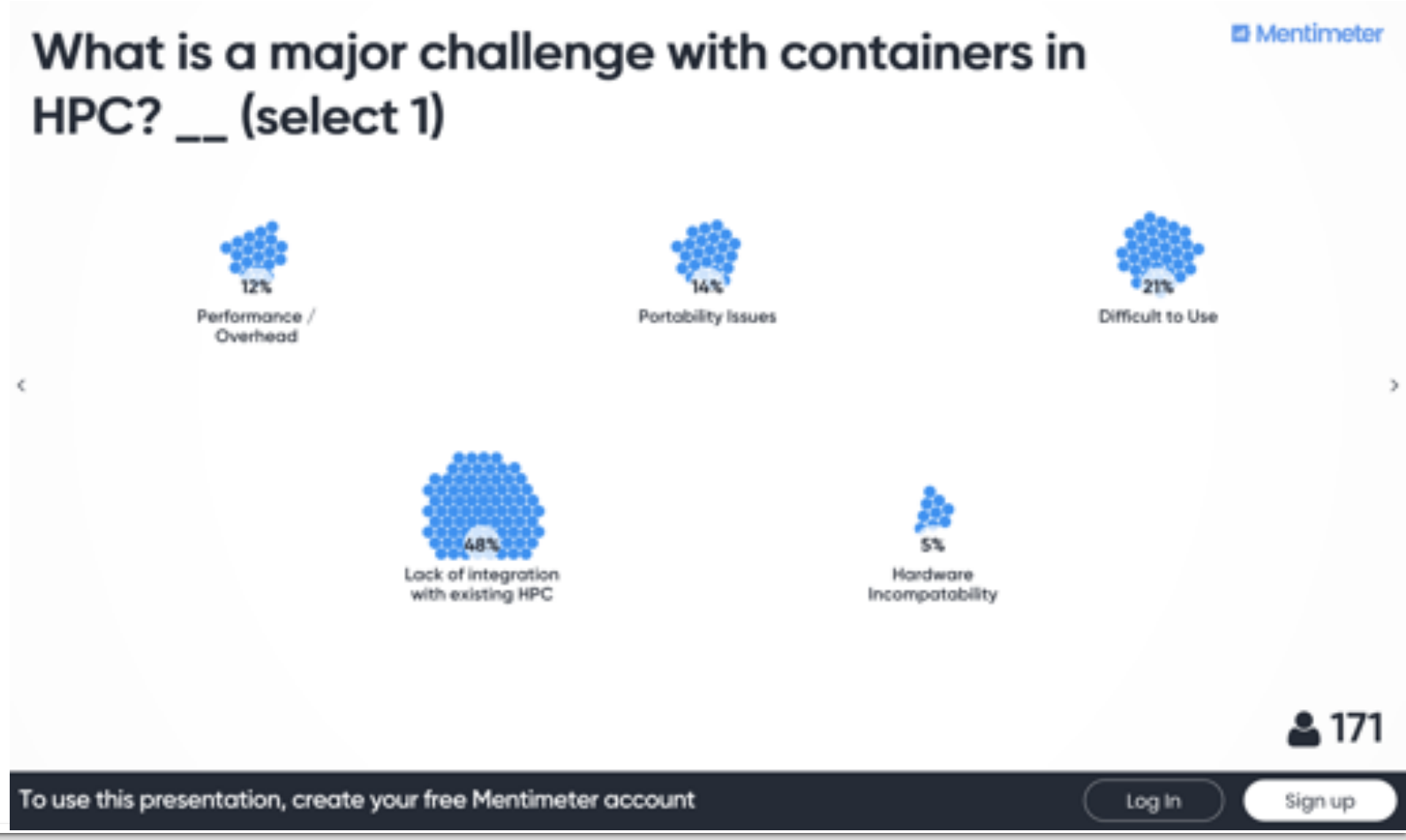
"Container orchestration" via Kubernetes

- manage container lifecycle
- manage container networking
- declarative management
- auto scaling
- cloud integration

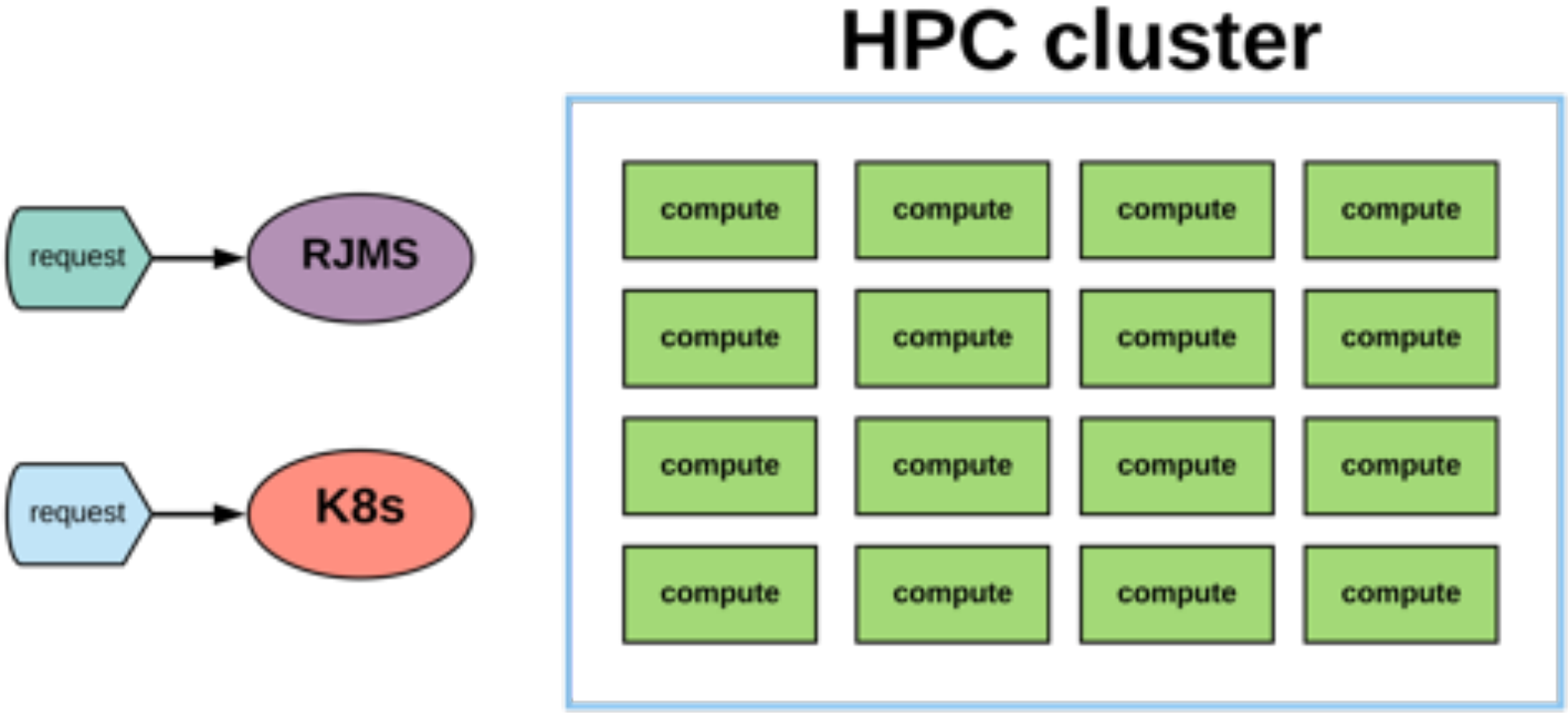


Substantial, growing interest + HPC

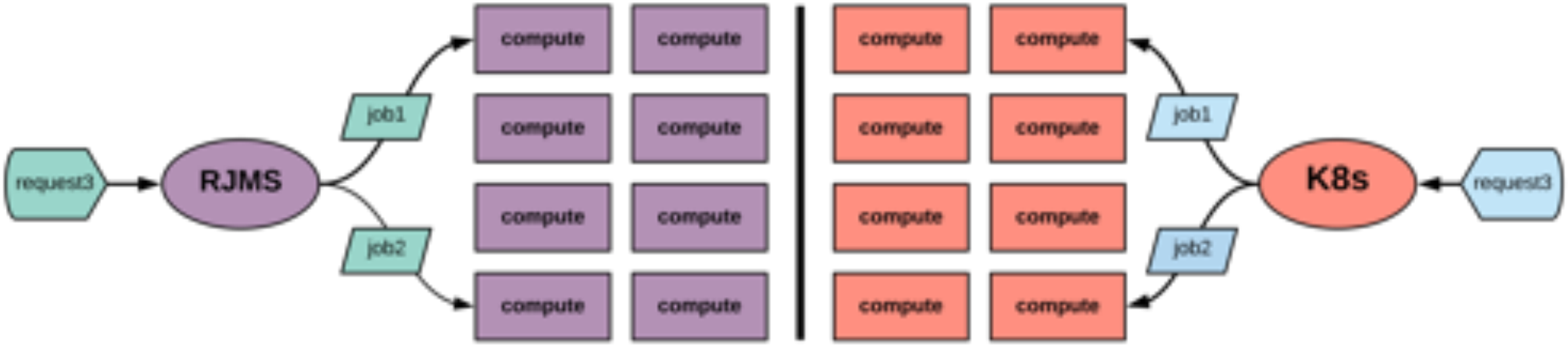
- KubeCon + CloudNativeCon NA 2019
  - 12,000 attendees (50% increase from 2018, 200% increase from 2017)
- Containers in HPC (SC19 BoF)



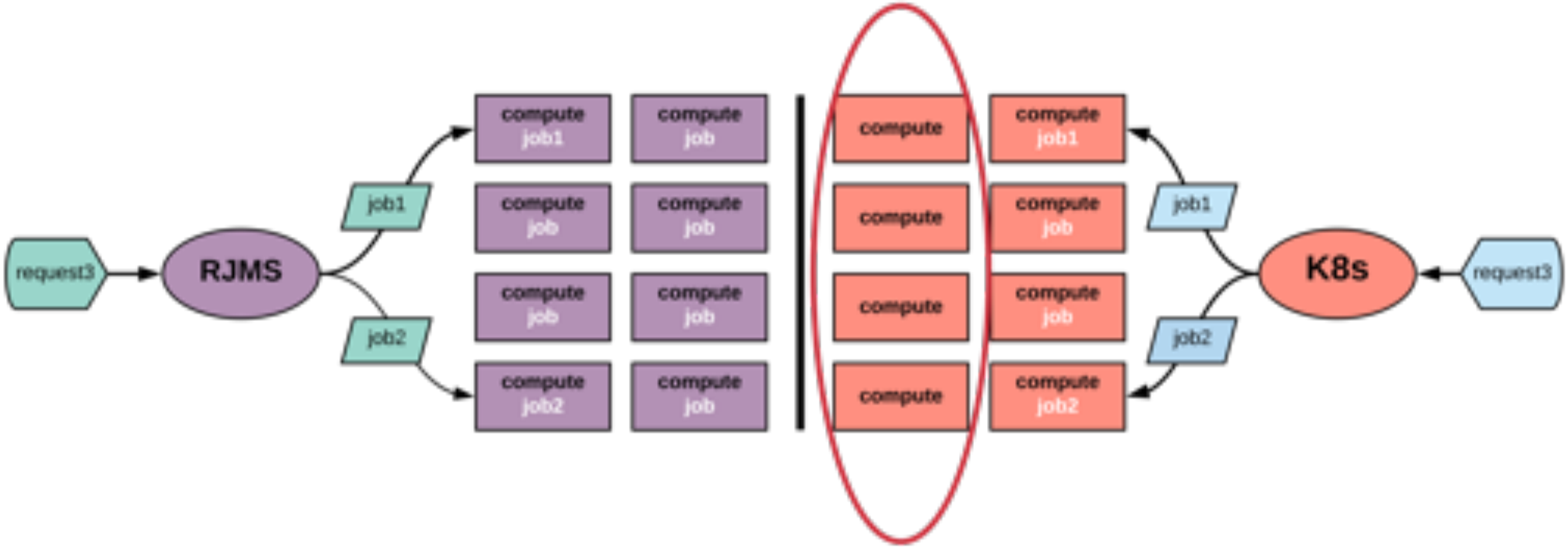
Converged k8s and HPC



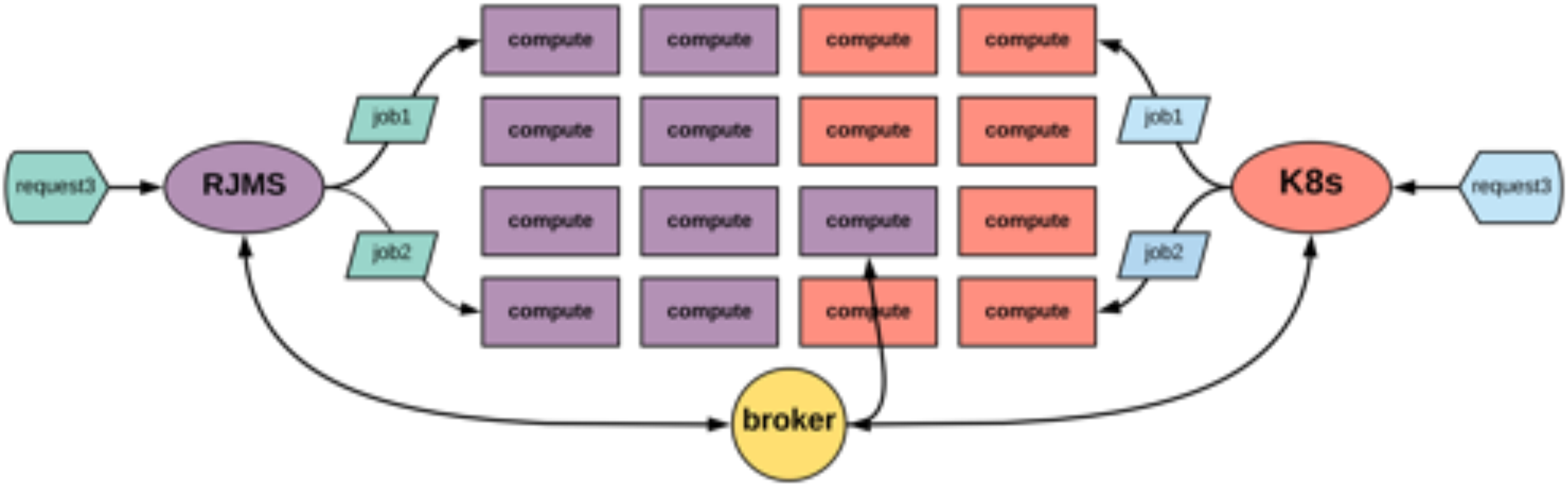
Converged k8s and HPC: static partitions



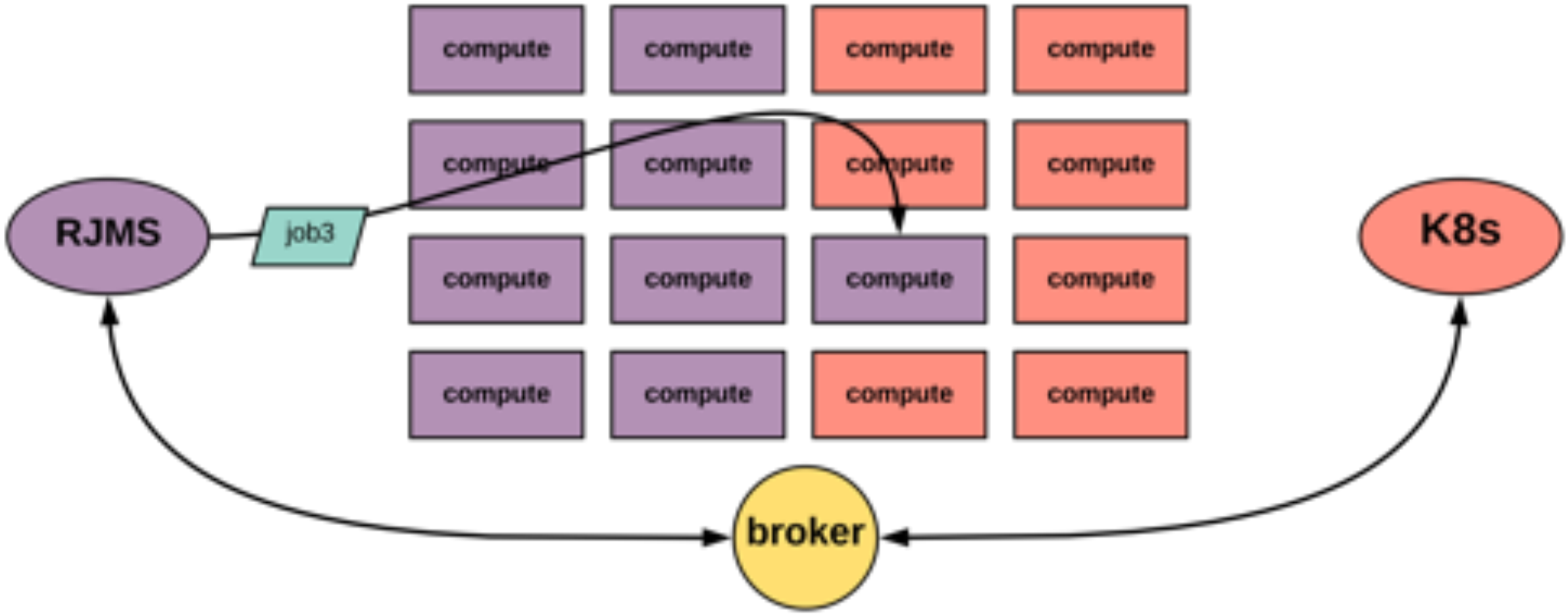
Static partitions challenge: inefficiency



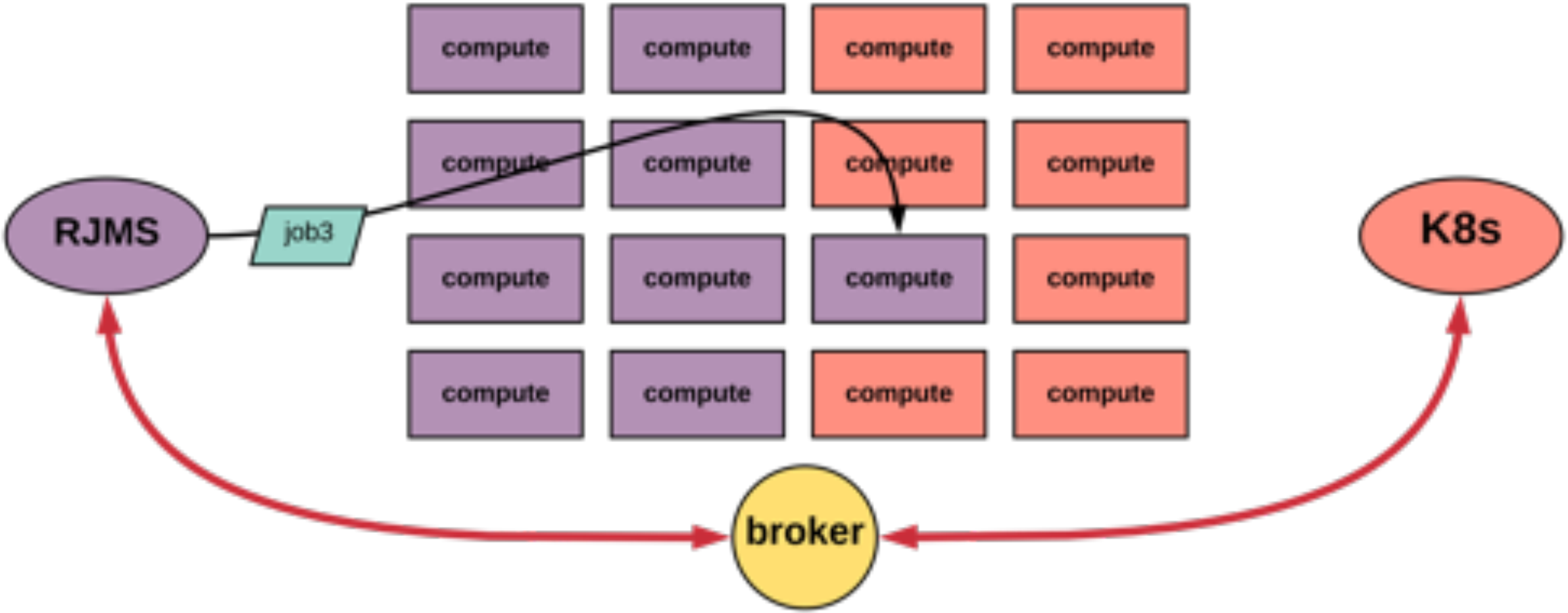
Converged k8s and HPC: resource broker



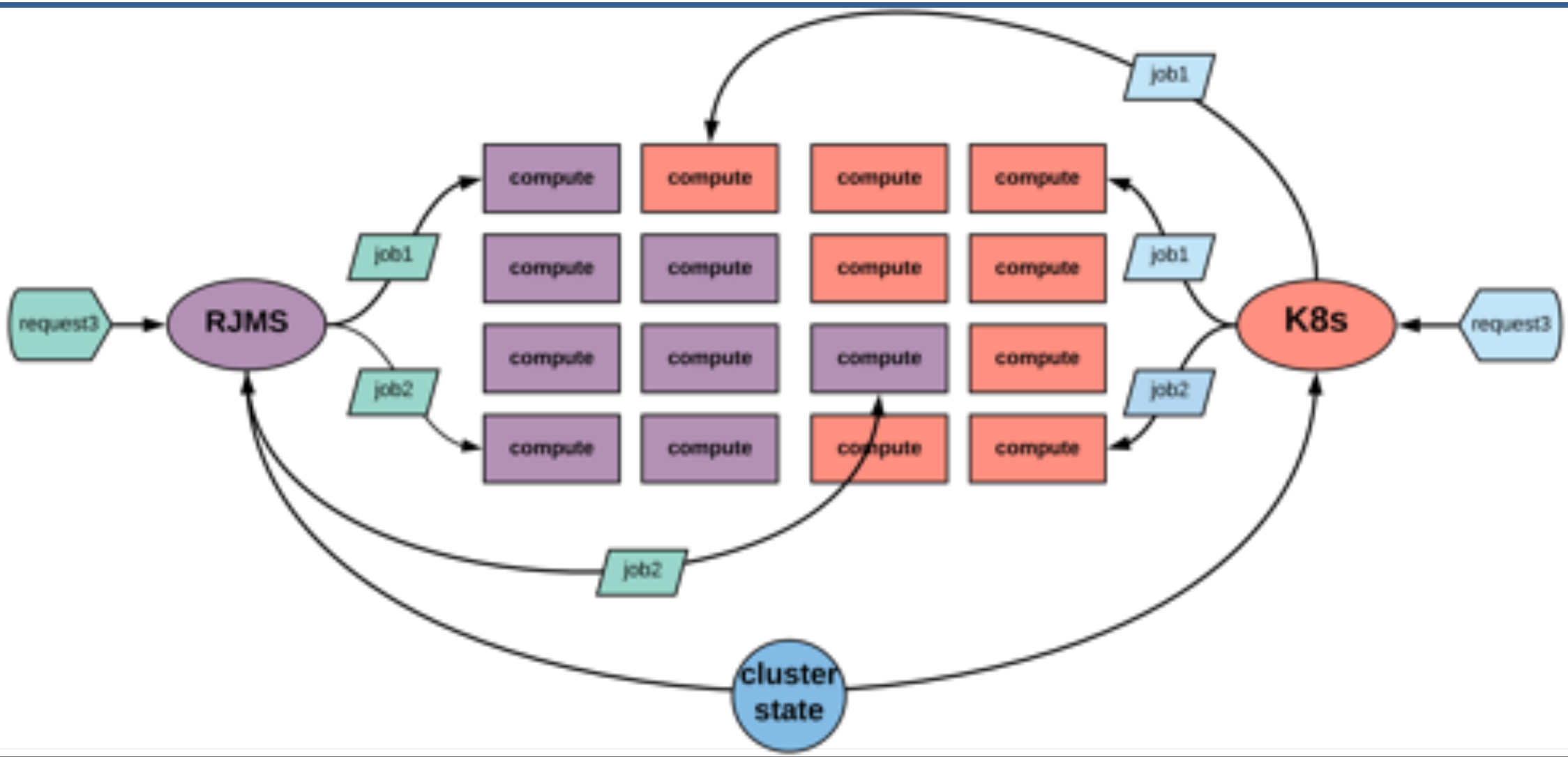
Converged k8s and HPC: resource broker



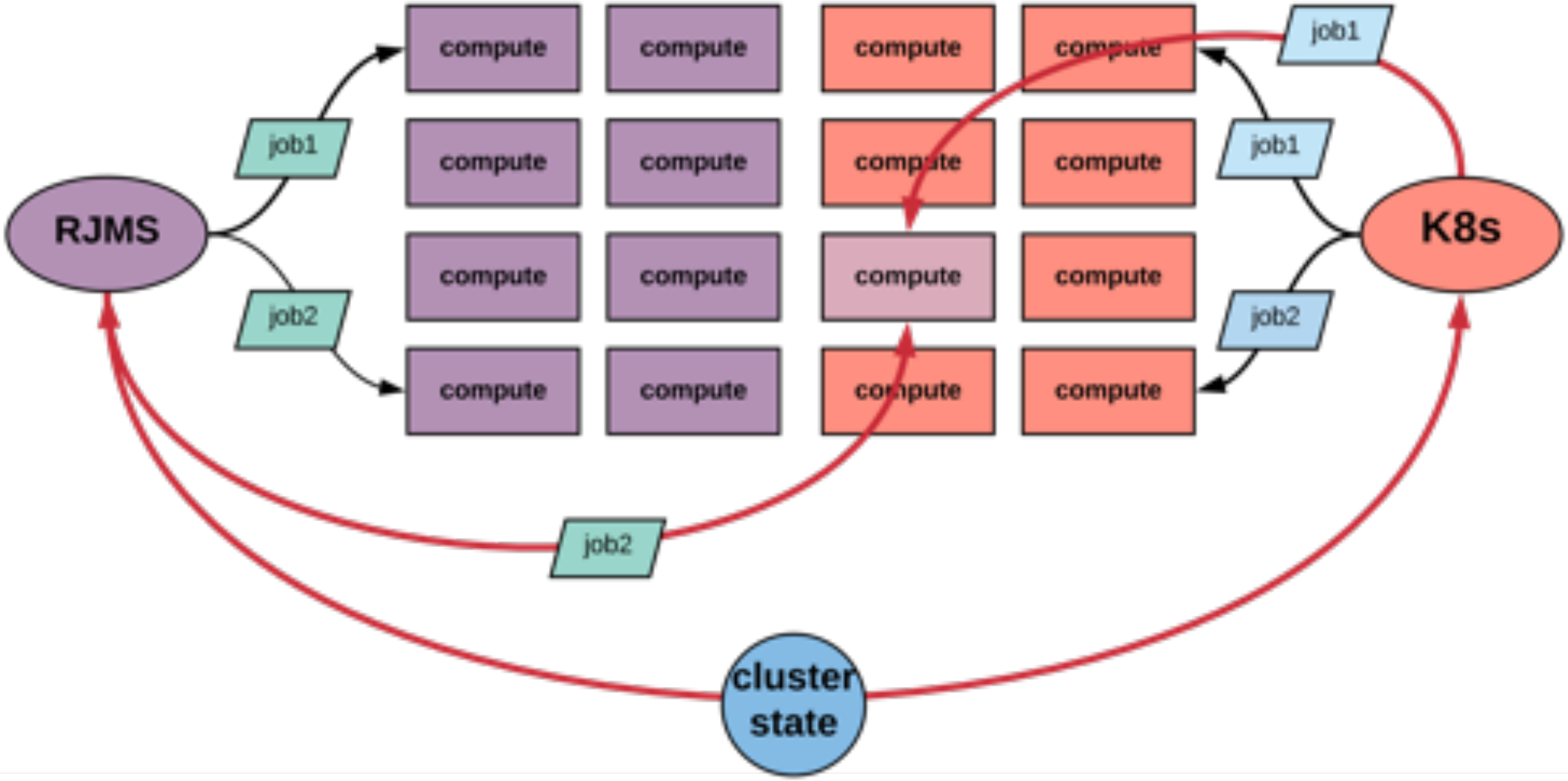
Resource broker challenge: synchronization



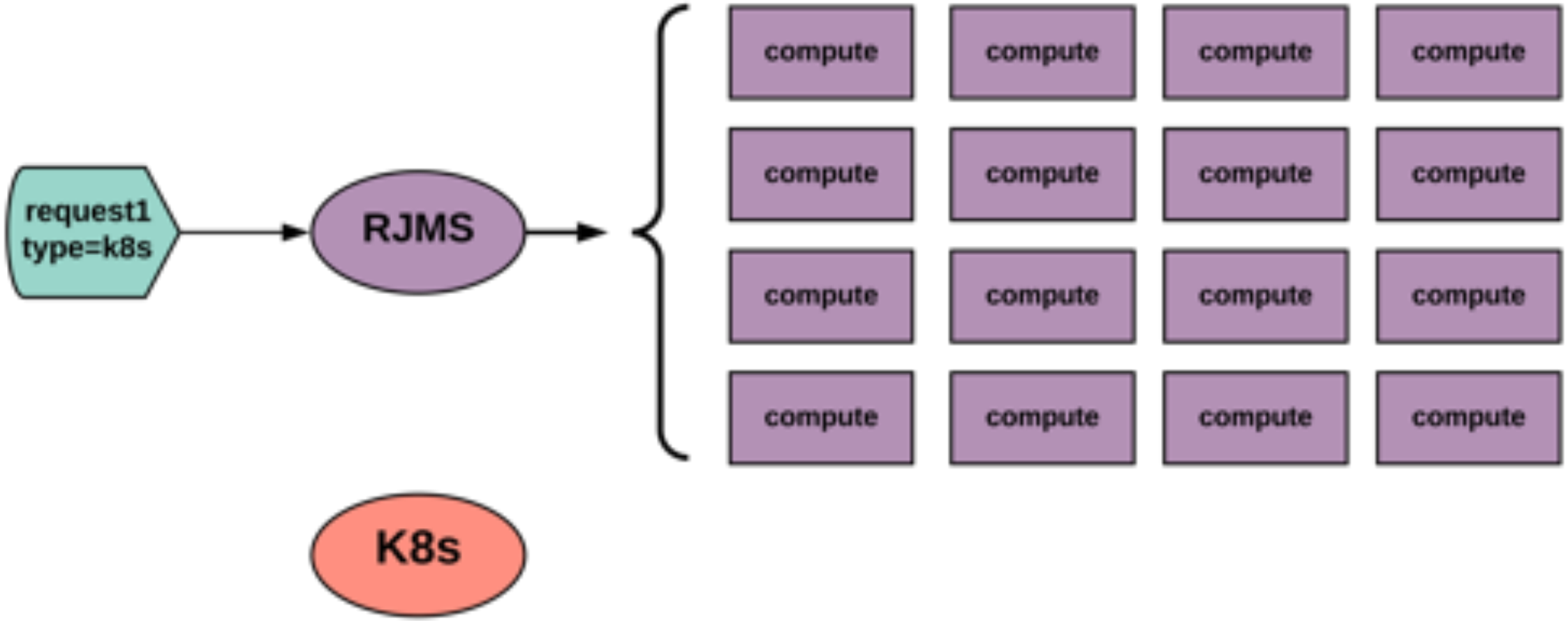
Converged k8s and HPC: shared state



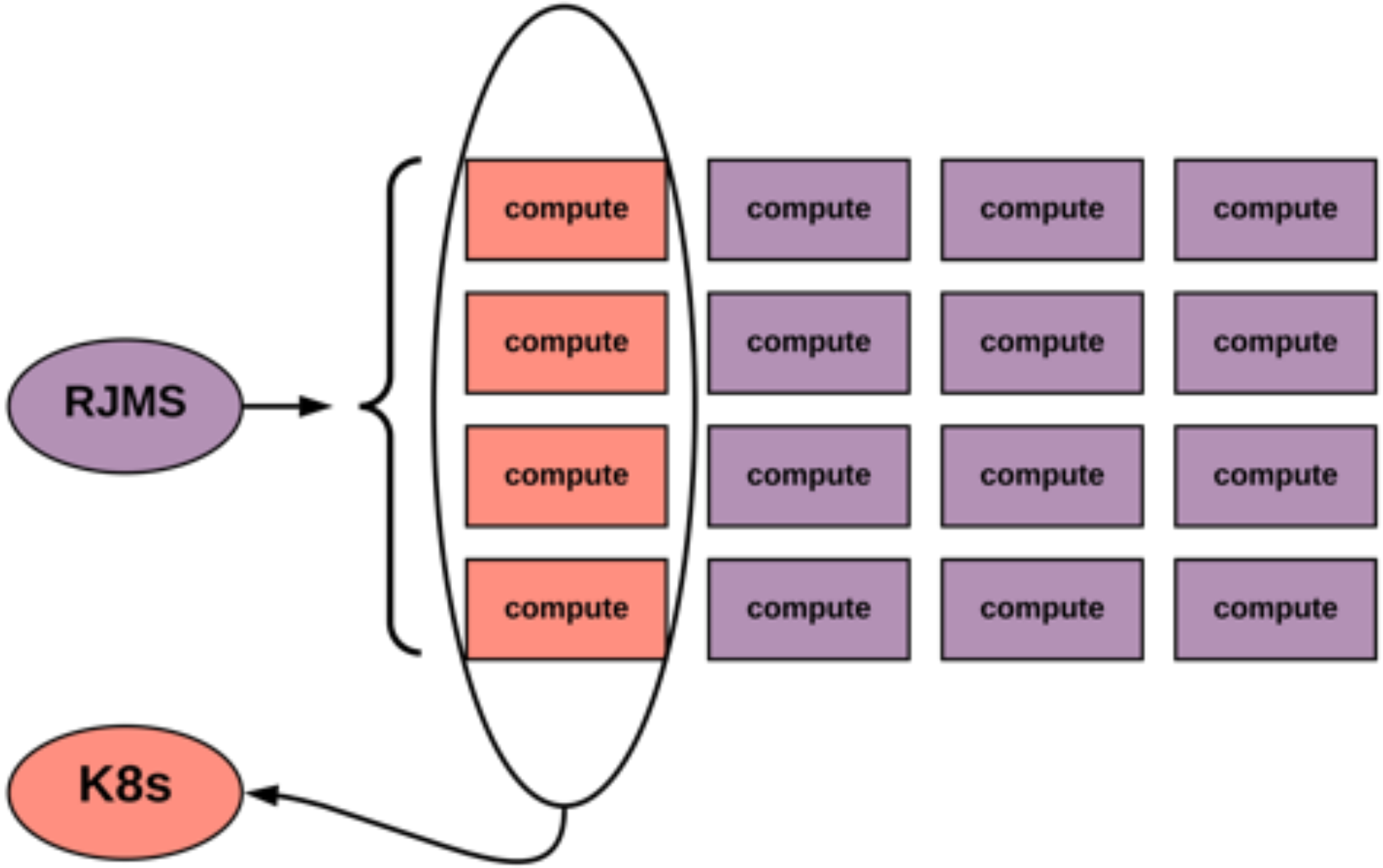
Shared state challenge: conflict



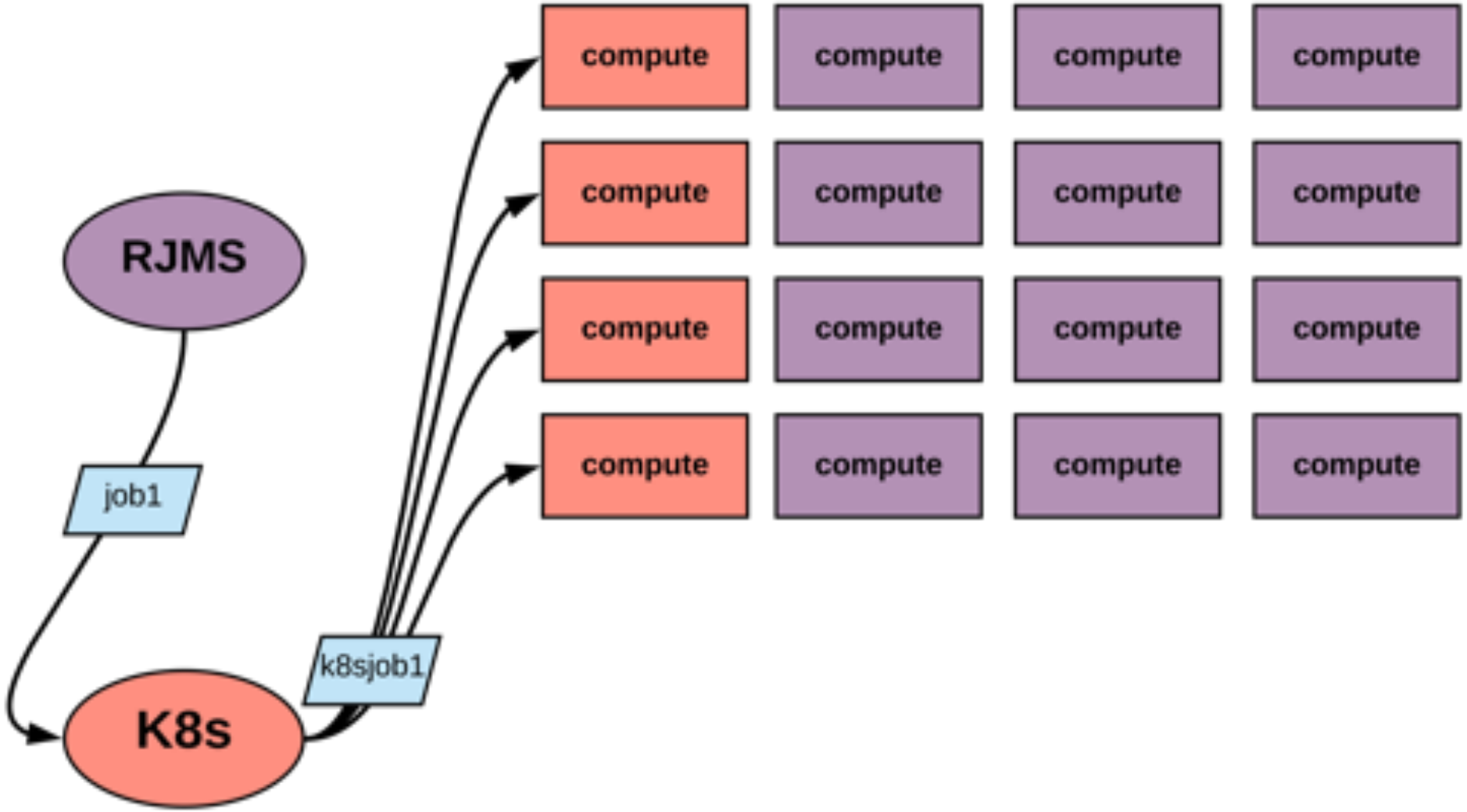
Converged k8s and HPC: subordinate manager



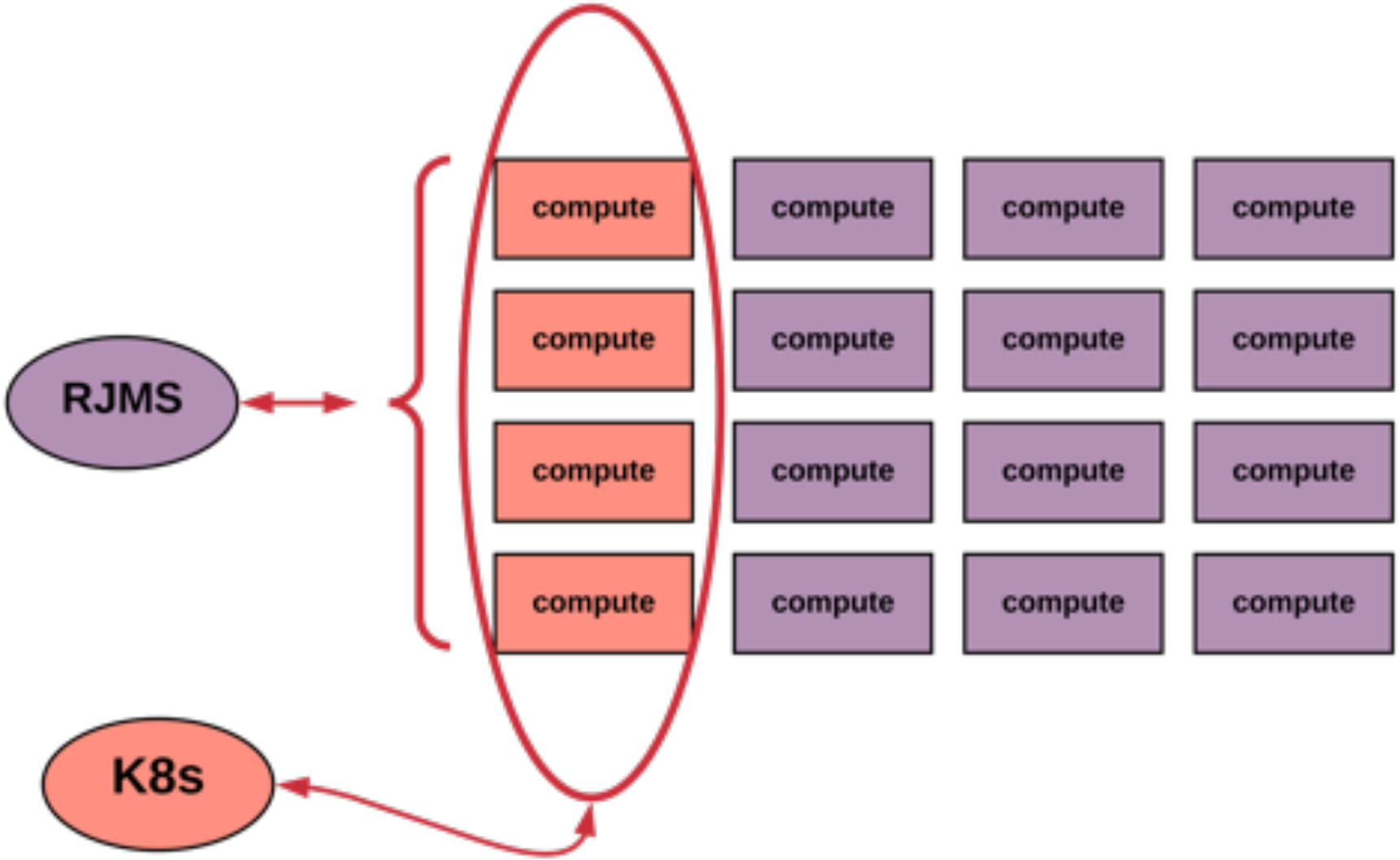
Converged k8s and HPC: subordinate manager



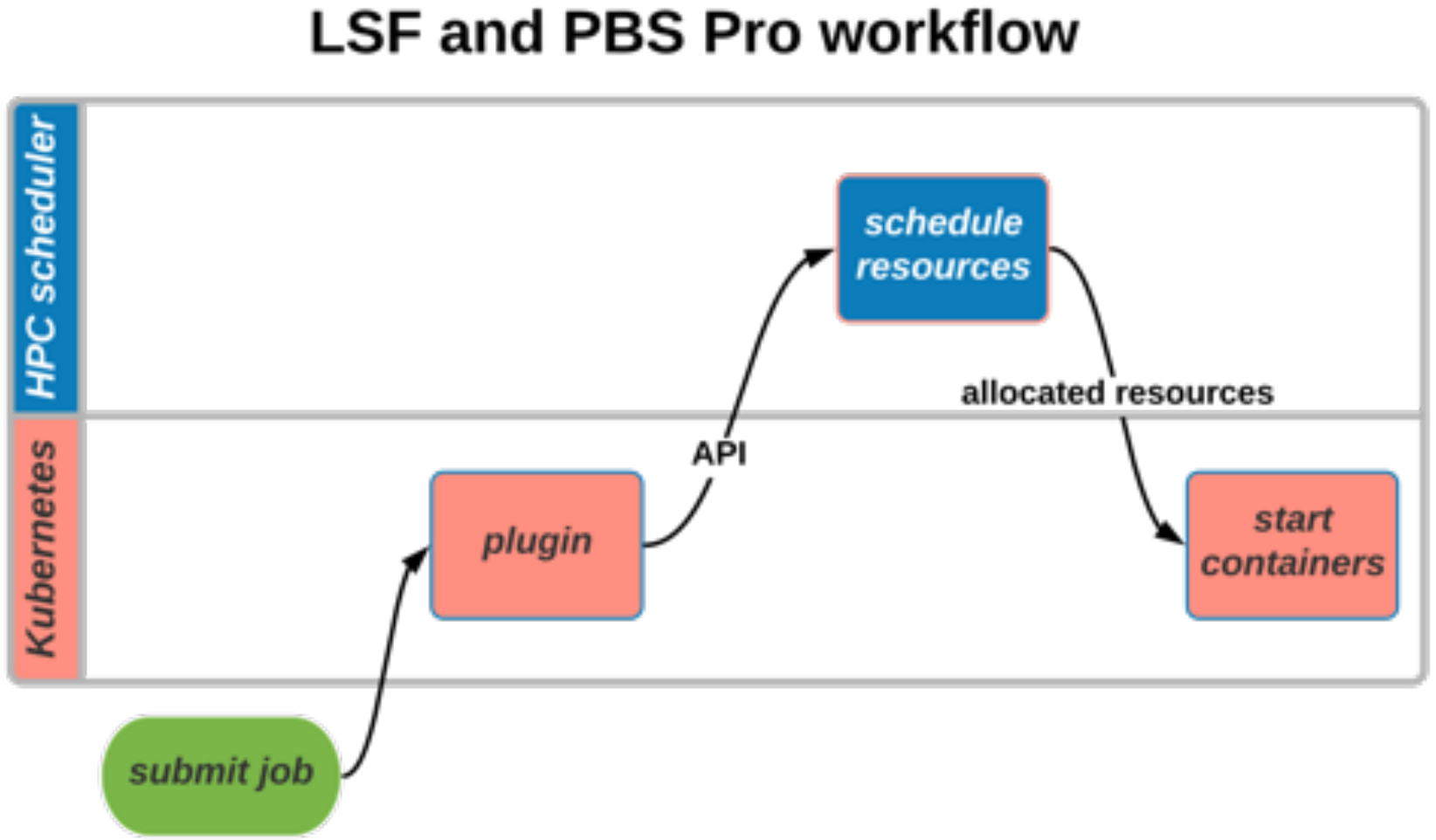
Converged k8s and HPC: subordinate manager



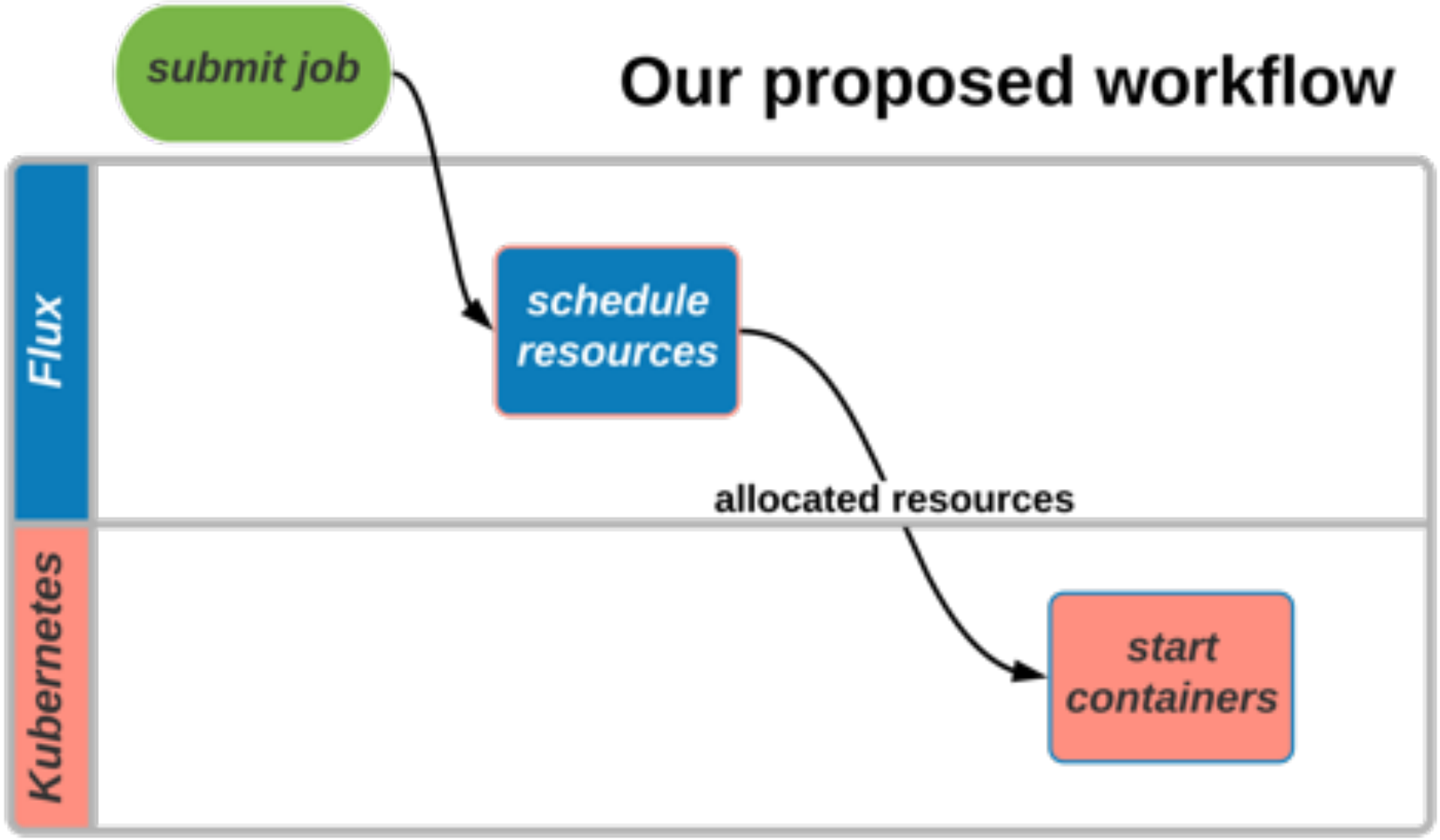
Subordinate manager challenge: resource management



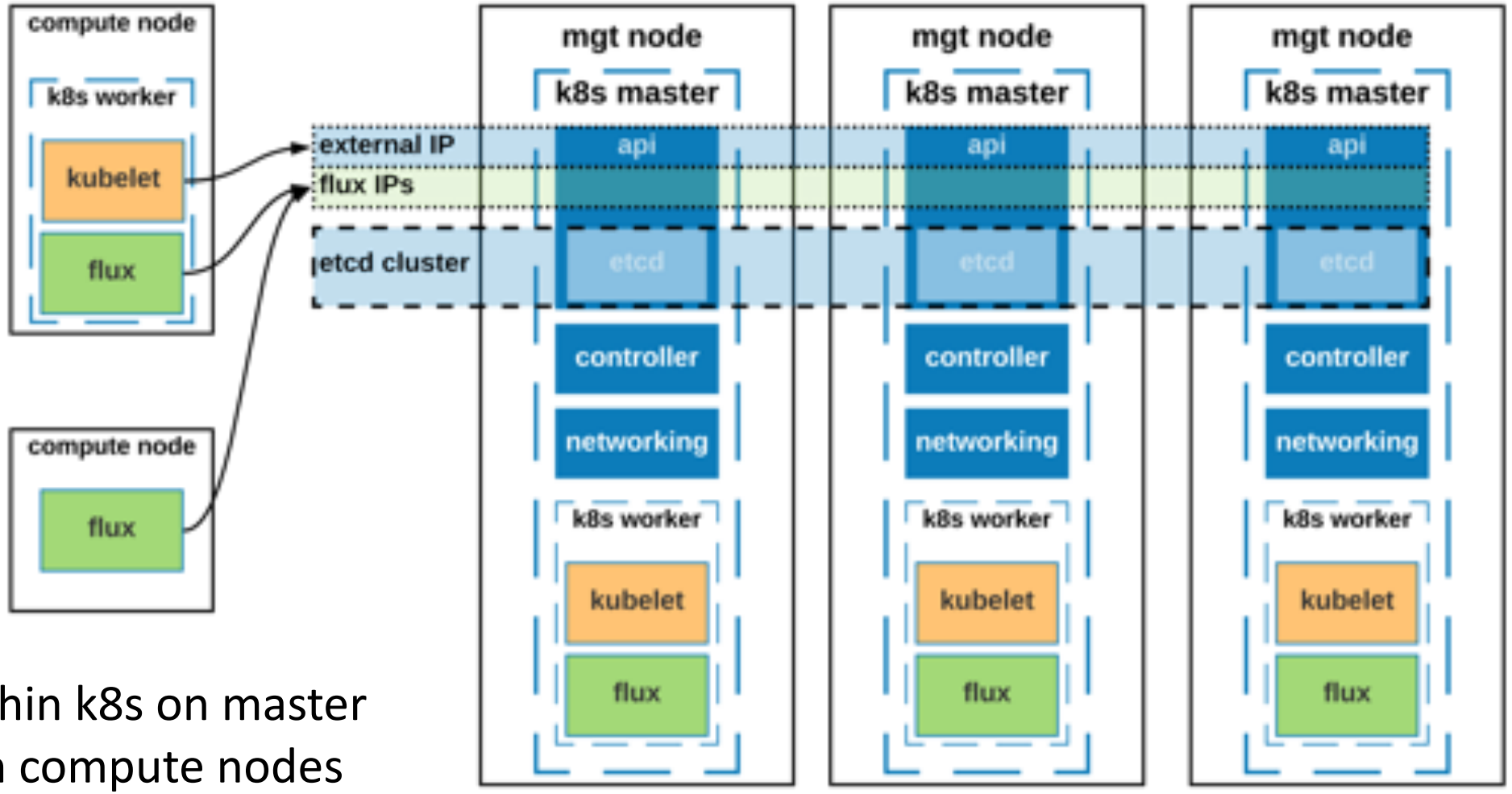
Current investigations in converged k8s and HPC



Current investigations in converged k8s and HPC



Current investigations in converged k8s and HPC

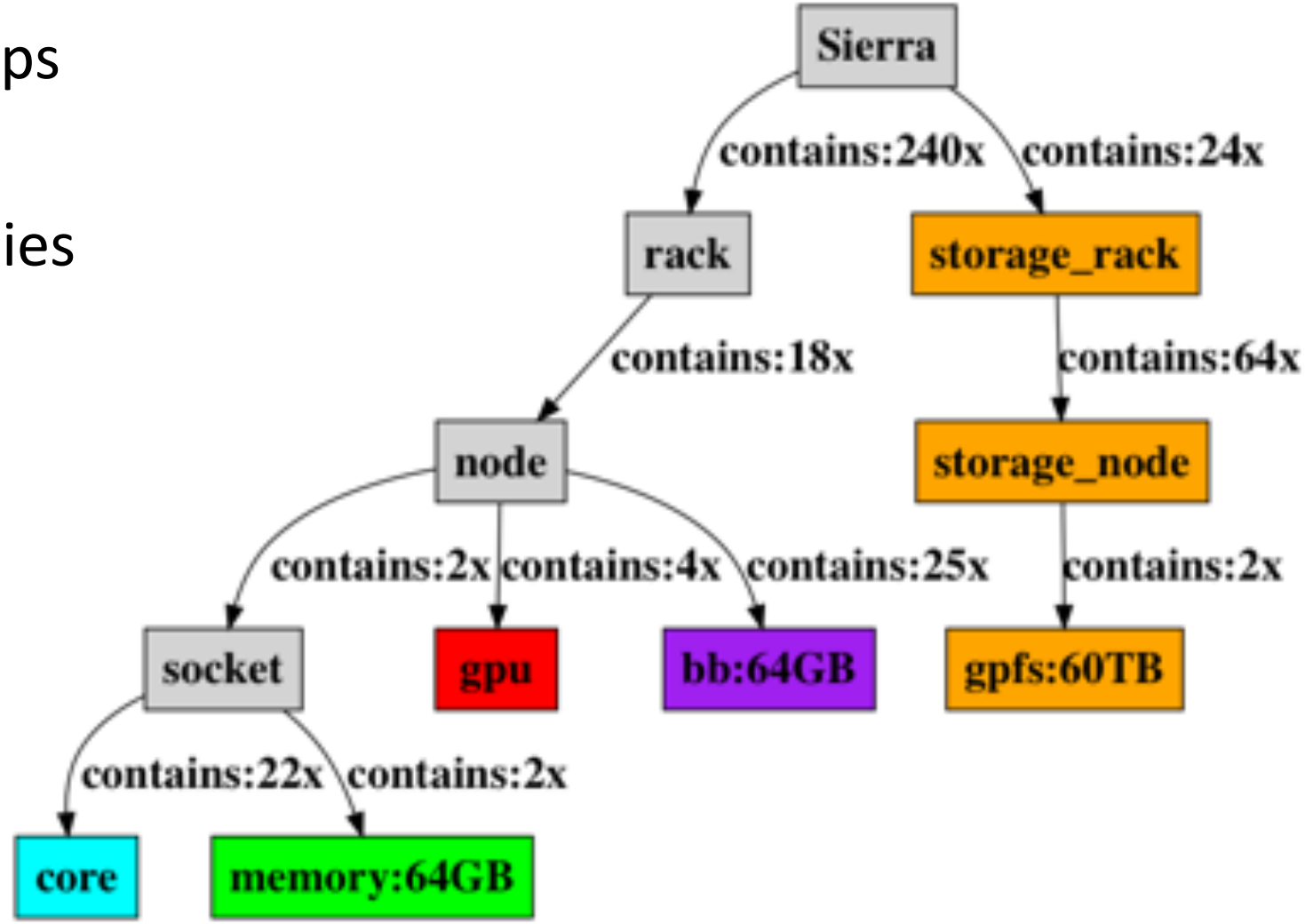


Run top-level Flux within k8s on master nodes, flux brokers on compute nodes

Flux in the cloud

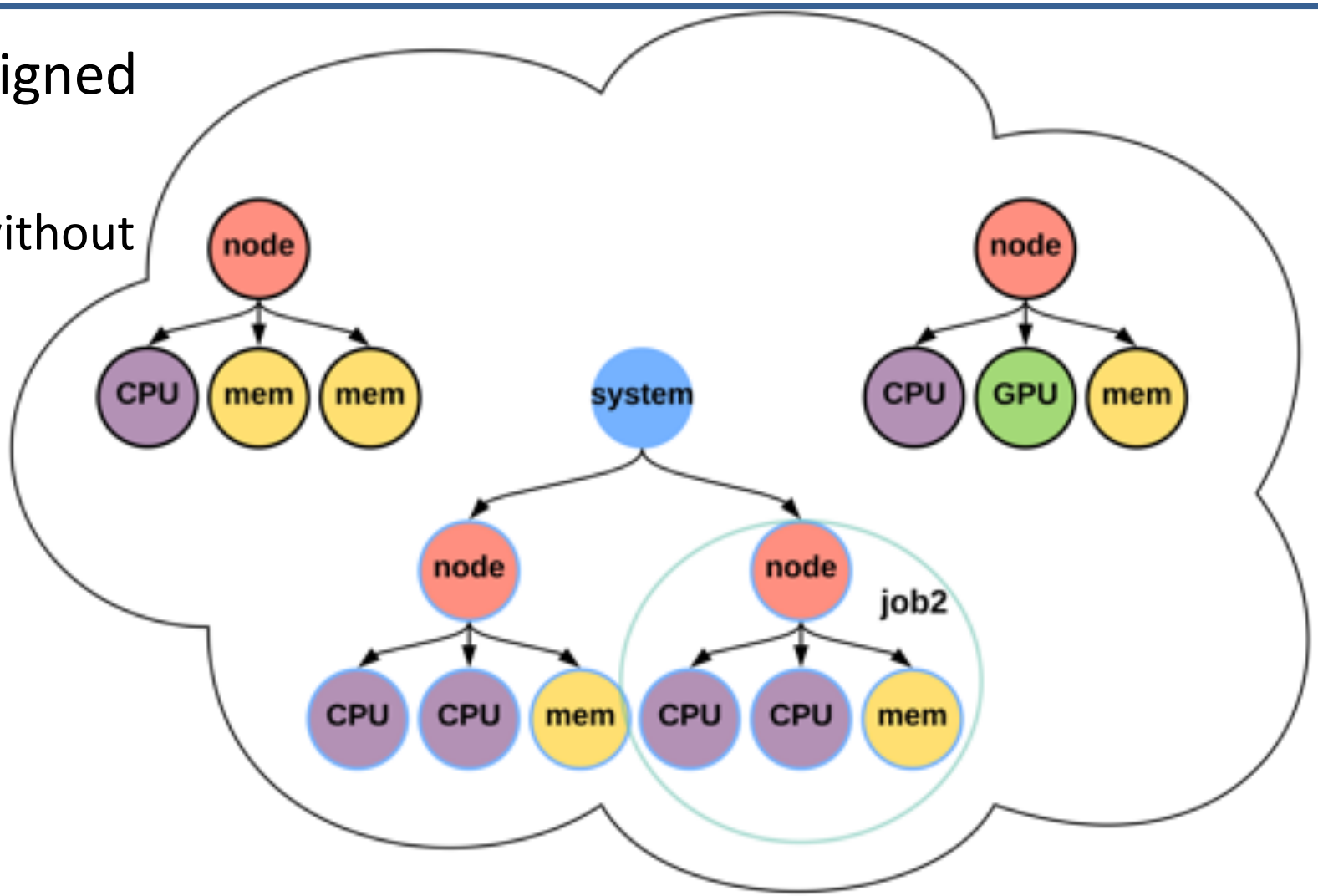
Flux graph-based resource model is powerful

- naturally expresses relationships
  - —can be dynamic
  - —naturally expresses hierarchies
- facilitates elasticity
- opportunities for scheduling research



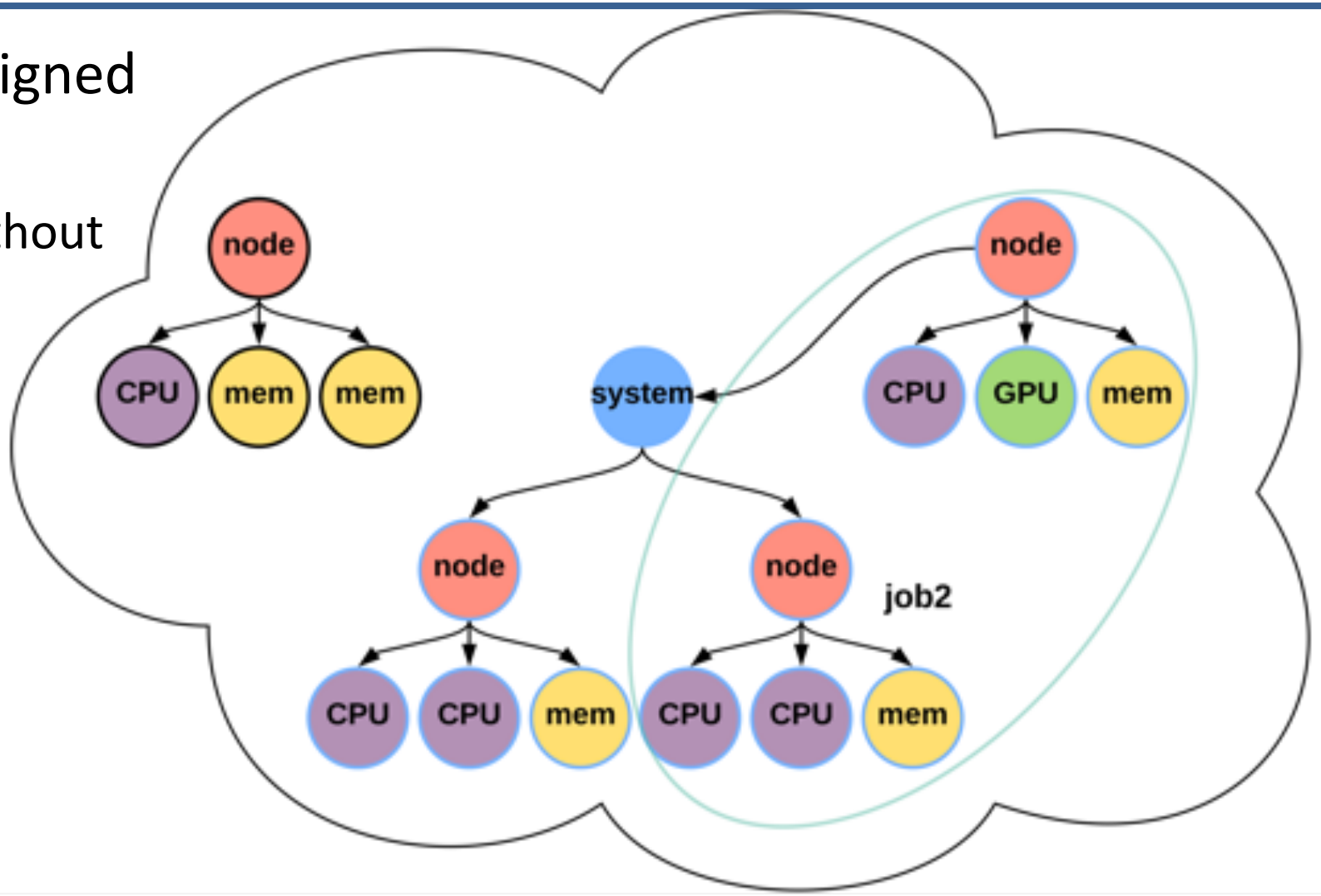
Flux facilitates elasticity

- traditional schedulers not designed for dynamic resources
  - hard to add new relationships without scheduler modification



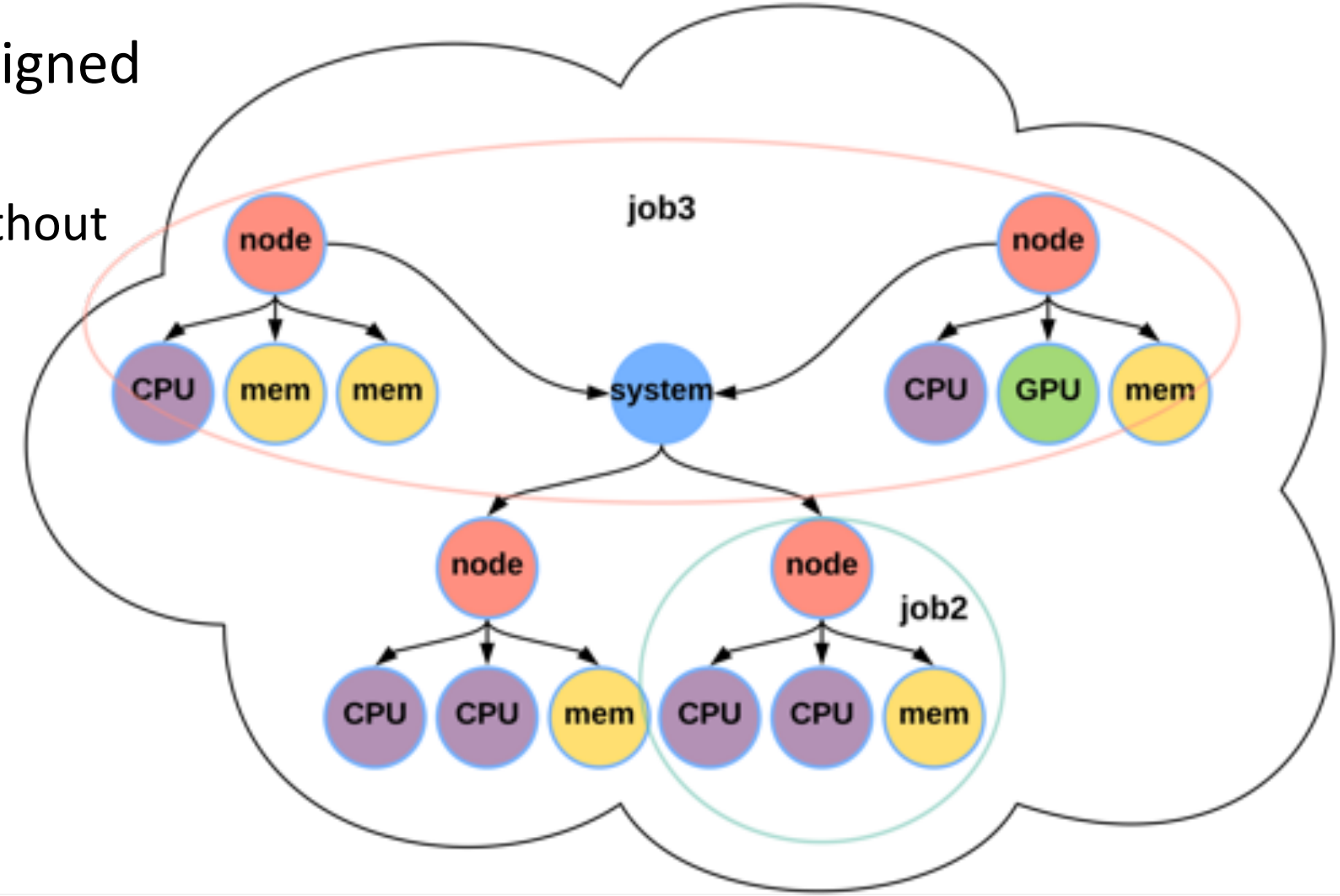
Flux facilitates elasticity

- traditional schedulers not designed for dynamic resources
  - hard to change relationships without scheduler modification



Flux facilitates elasticity

- traditional schedulers not designed for dynamic resources
  - hard to change relationships without scheduler modification



Summary

- Containerized workflows are increasingly common in HPC
- Flux facilitates converged k8s-HPC: we're taking the subordinate manager approach
- HA Flux-k8s for Exascale at LLNL
- We're putting Flux in the cloud!

Audience feedback

Questions?

Feature requests/workflow support?


Lawrence Livermore
National Laboratory

Fluxion: A Scalable Graph-Based Resource Model to Multifaceted HPC Scheduling Challenges

Flux Tutorial at ECP Annual Meeting, Feb 6, 2020

Dong H. Ahn, Tapasya Patki, Stephen Herbein, Daniel Milroy, Bruce D'Amora (IBM), Claudia Misale (IBM)

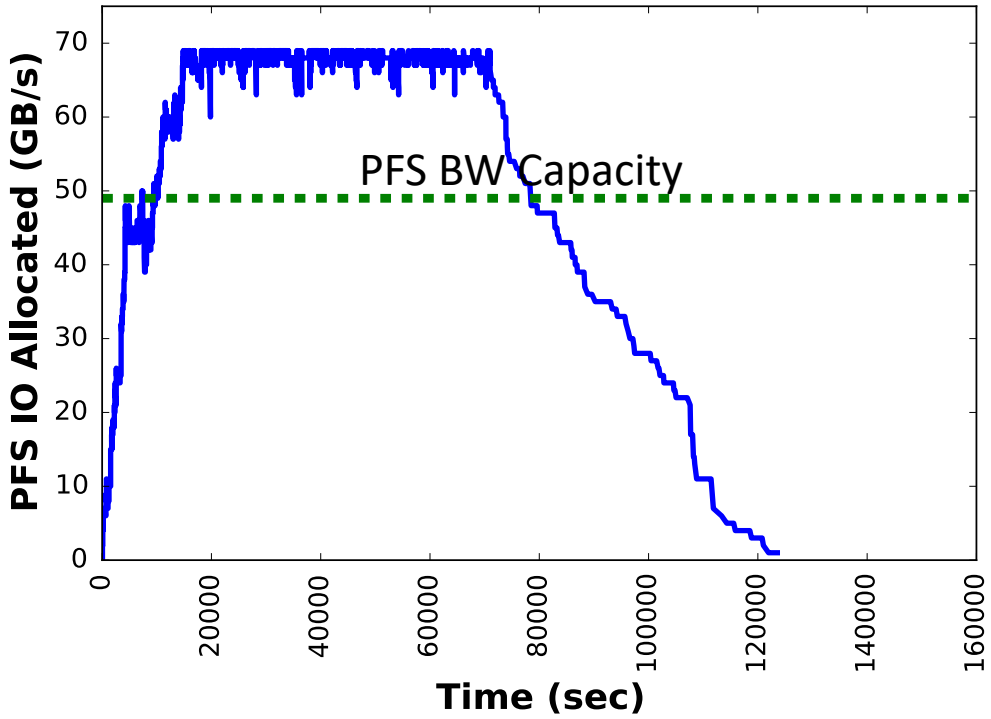
LLNL-PRES- 804364

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

Lawrence Livermore National Laboratory

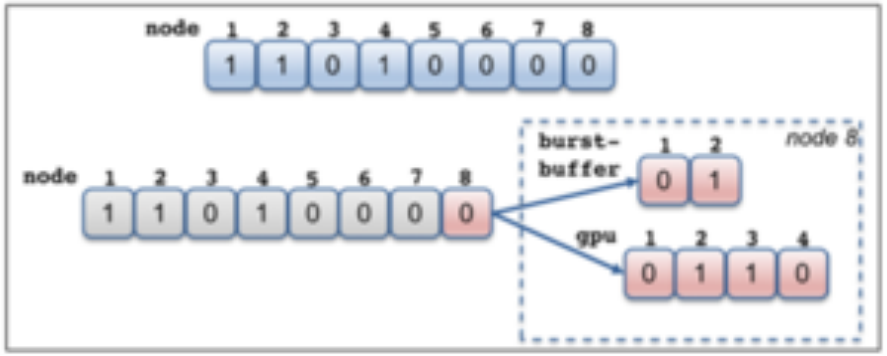
The changes in resource types are equally challenging.

- Problems are not just confined to the workload/workflow challenge.
- Resource types and their relationships are also becoming increasingly complex.
- Much beyond compute nodes and cores…
  - GPGPUs
  - Burst buffers
  - I/O and network bandwidth
  - Network locality
  - Power



The traditional resource data models are largely ineffective to cope with the resource challenge.

- Designed when the systems are much simpler
  - Node-centric models
  - SLURM: bitmaps to represent a set of compute nodes
  - PBSPro: a linked-list of nodes

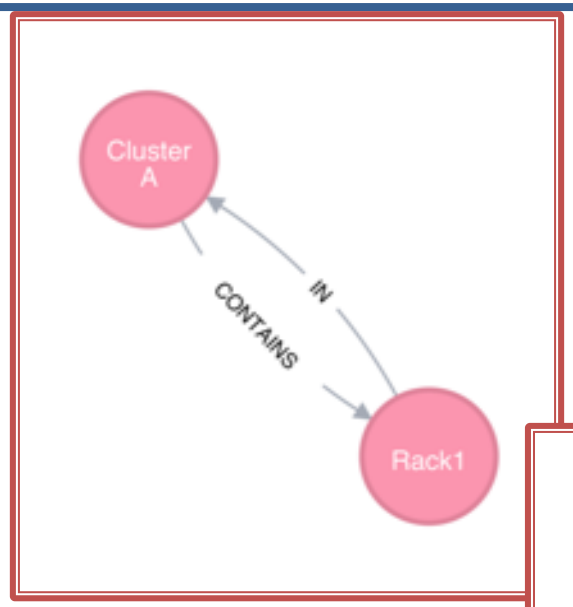


- HPC has become far more complex
  - Evolutionary approach to cope with the increased complexity
  - E.g., add auxiliary data structures on top of the node-centric data model

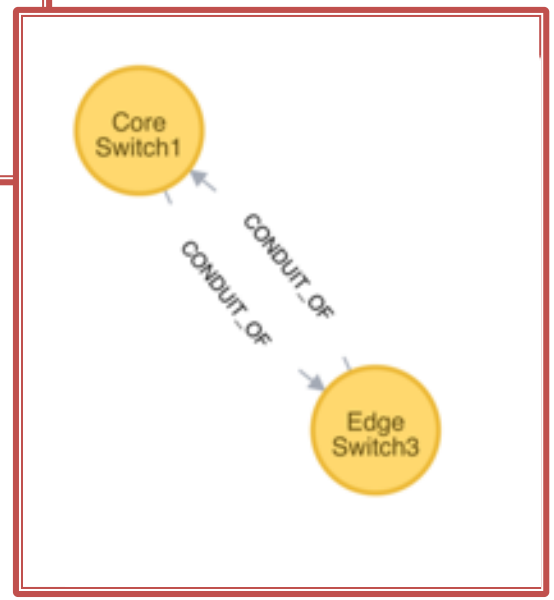
- Can be quickly unwieldy
  - Every new resource type requires new a user-defined type
  - A new relationship requires a complex set of pointers cross-referencing different types.

Flux uses a graph-based resource data model to represent schedulable resources and their relationships.

- A graph consists of a set of vertices and edges
  - Vertex: a resource
  - Edge: a relationship between two resources
- Highly composable to support a graph with arbitrary complexity
- The scheduler remains to be a highly generic graph code.



Containment subsystem



Network connectivity subsystem

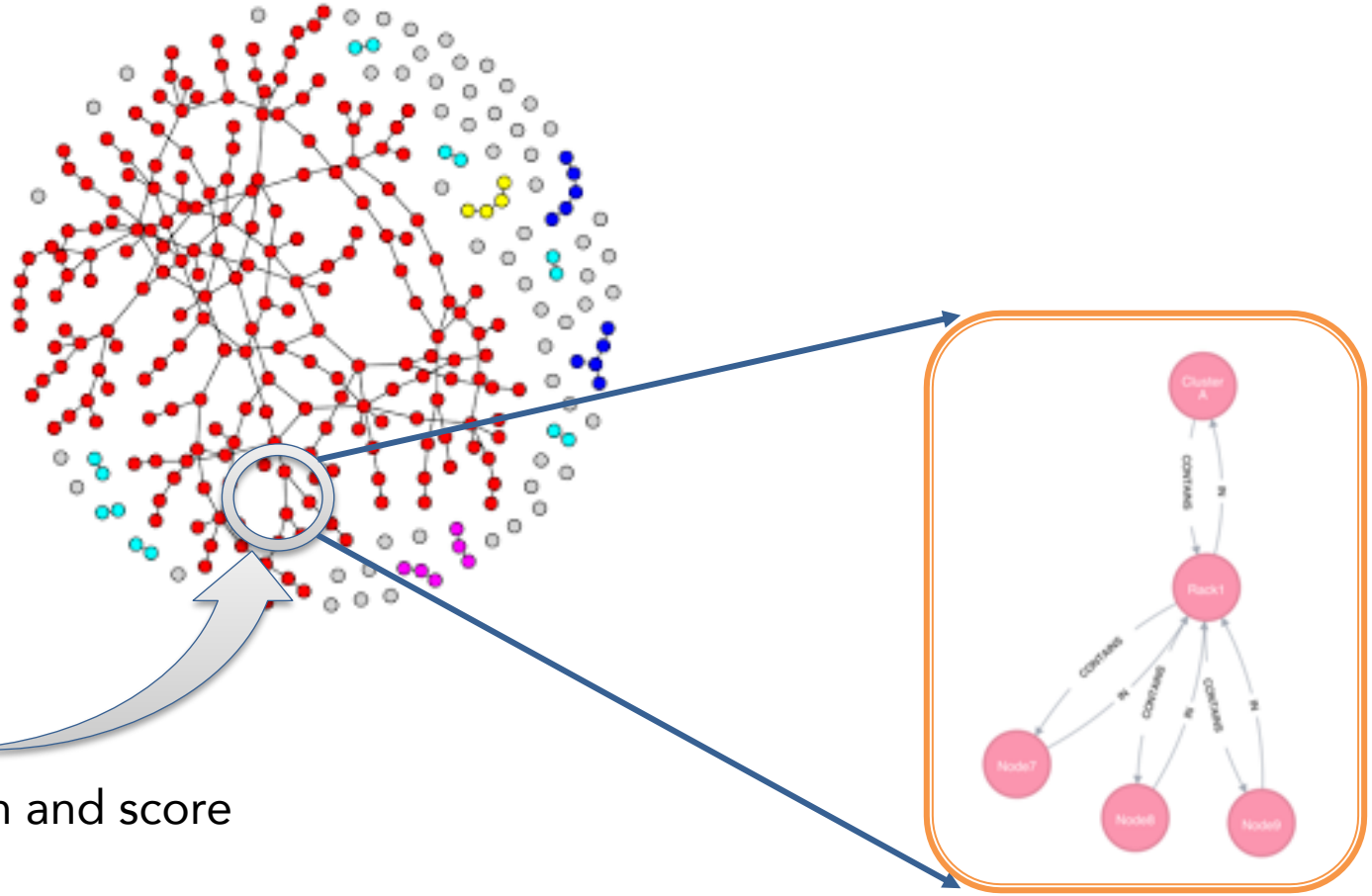
Flux's graph-oriented canonical job-spec allows for a highly expressive resource requests specification.

- Graph-oriented resource requirements
  - Express the resource requirements of a program to the scheduler
  - Express program attributes such as arguments, run time, and task layout, to be considered by the execution service
- cluster->racks[2]->slot[3]->node[1]->sockets[2]->core[18]
- ***slot*** is the only non-physical resource type
  - Represent a schedulable place where program process or processes will be spawned and contained
- Referenced from the tasks section

```
1  version: 1
2  resources:
3    - type: cluster
4      count: 1
5      with:
6        - type: rack
7          count: 2
8          with:
9            - type: slot
10             label: myslot
11             count: 3
12             with:
13               - type: node
14                 count: 1
15                 with:
16                   - type: socket
17                     count: 2
18                     with:
19                       - type: core
20                         count: 18
21
22 # a comment
23 attributes:
24   system:
25     duration: 3600
26 tasks:
27   - command: app
28     slot: myslot
29     count:
30       per_slot: 1
```

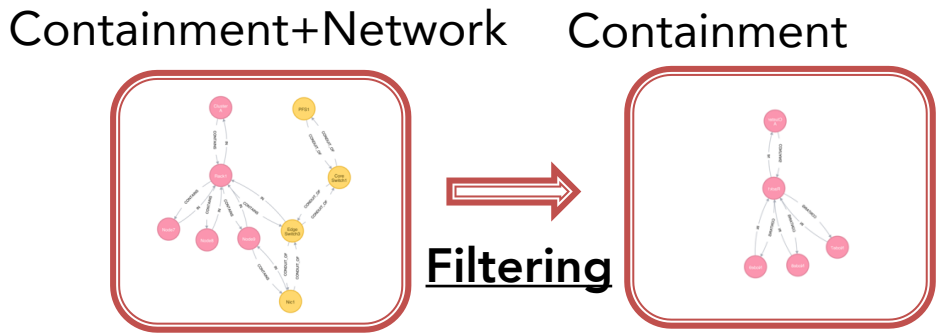
Flux maps our complex scheduling problems into graph matching problems.

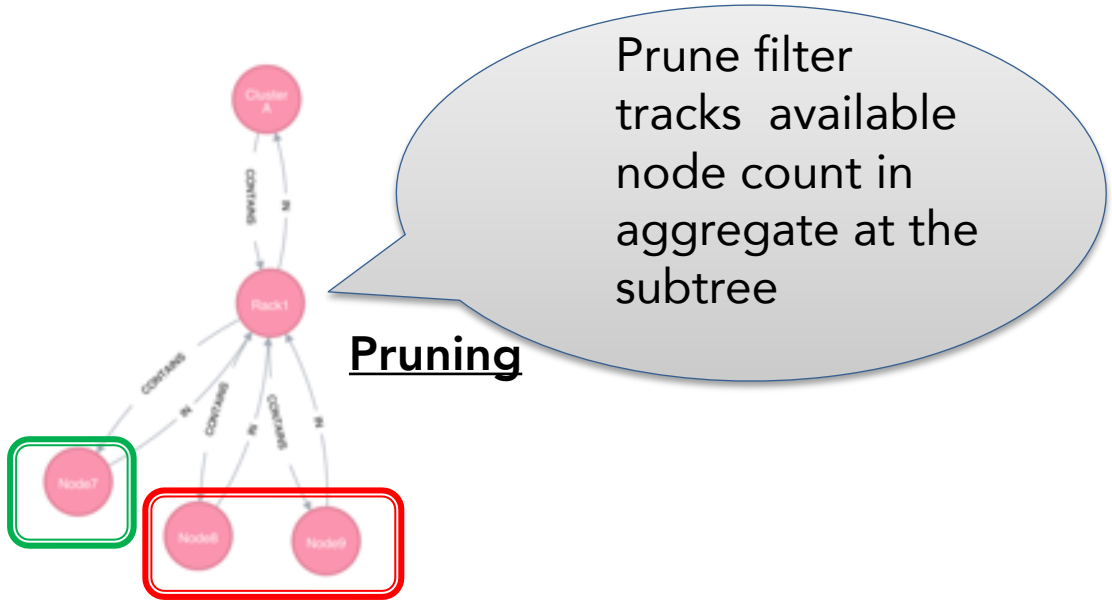
```yaml
version: 1
resources:
  - type: cluster
    count: 1
    with:
      - type: rack
        count: 2
        with:
          - type: slot
            label: myslot
            count: 3
            with:
              - type: node
                count: 1
                with:
                  - type: socket
                    count: 2
                    with:
                      - type: core
                        count: 18

# a comment
attributes:
  system:
    duration: 3600
tasks:
  - command: app
    slot: myslot
    count:
      per_slot: 1
```



We use graph filtering and pruned searching to manage the graph complexity and optimize our graph search.

- The total graph can be quite complex
  - Two techniques to manage the graph complexity and scalability
- Filtering reduces graph complexity
  - The graph model needs to support schedulers with different complexity
  - Provide a mechanism by which to filter the graph based on what subsystems to use
- Pruned search increases scalability
  - Fast RB tree-based planner is used to implement a pruning filter per each vertex.
  - Pruning filter keeps track of summary information (e.g., aggregates) about subtree resources.
  - Scheduler-driven pruning filter update





Flux Support for Tiered Storage Systems

Anthony Agelastos, Dong H. Ahn, Frank Di Natale,
Stephen Herbein, Dan Milroy, Tapasya Patki

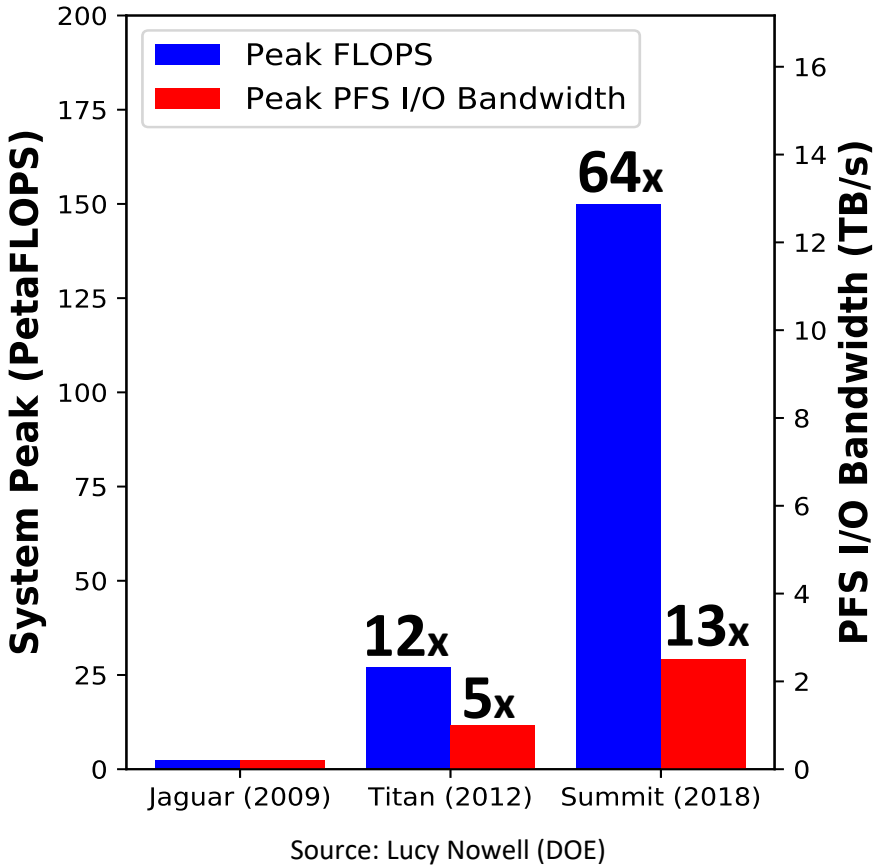
February 2020

LLNL-PRES- 804364

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

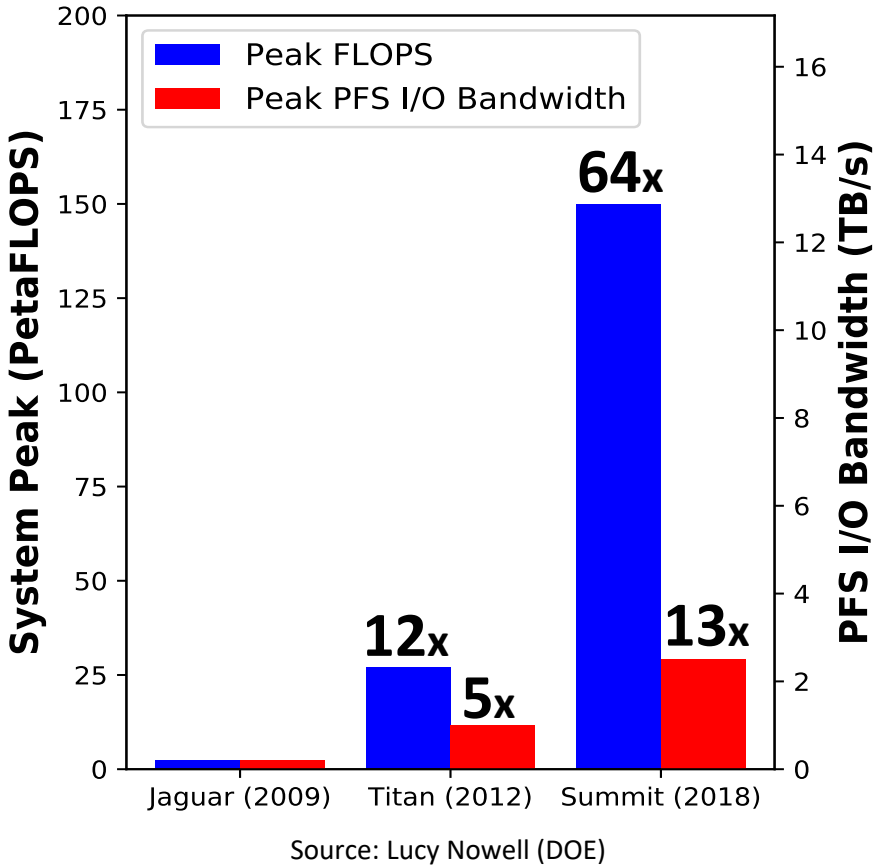
Lawrence Livermore National Laboratory

Tiered Storage in HPC

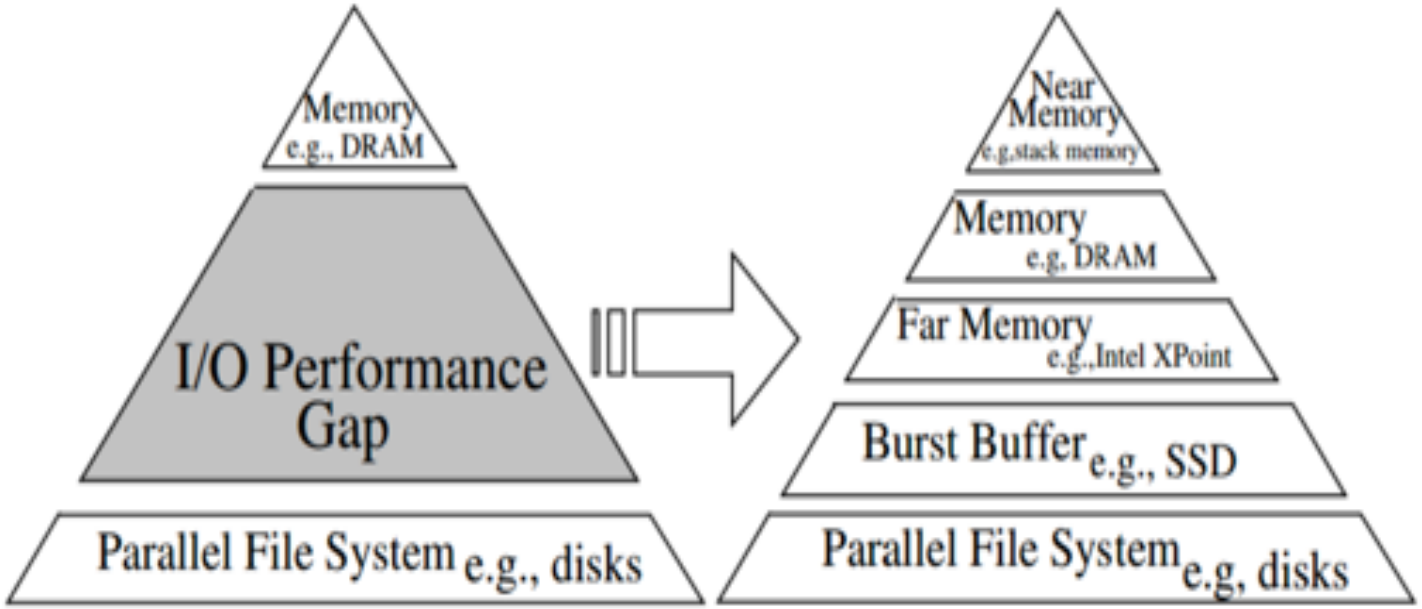


Source: Lucy Nowell (DOE)

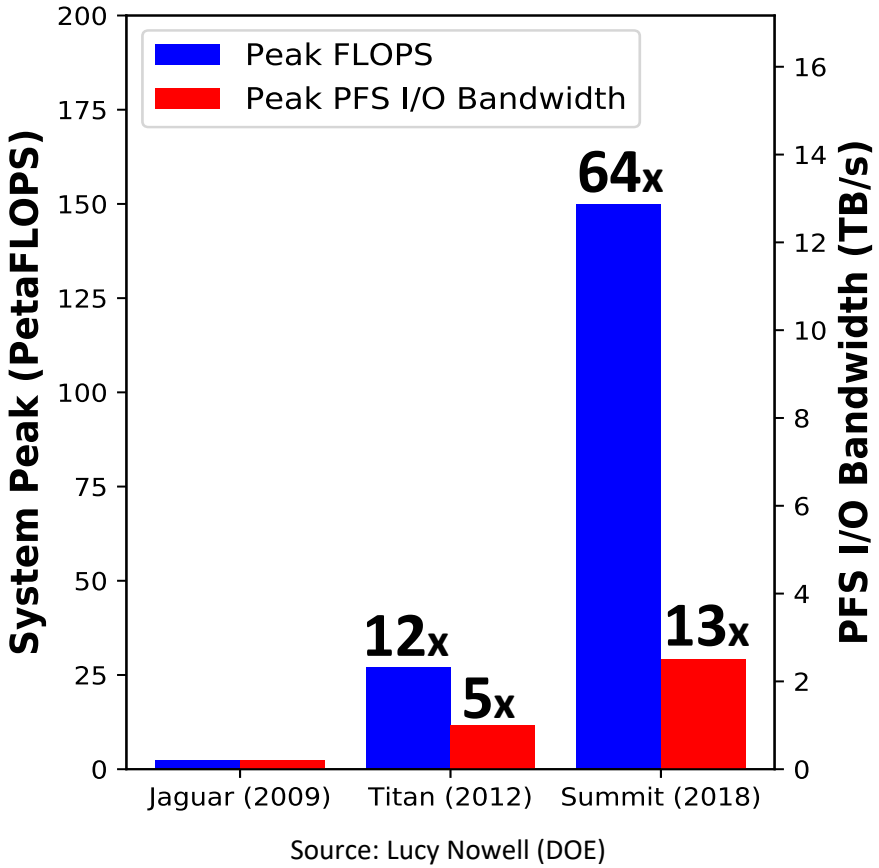
Tiered Storage in HPC



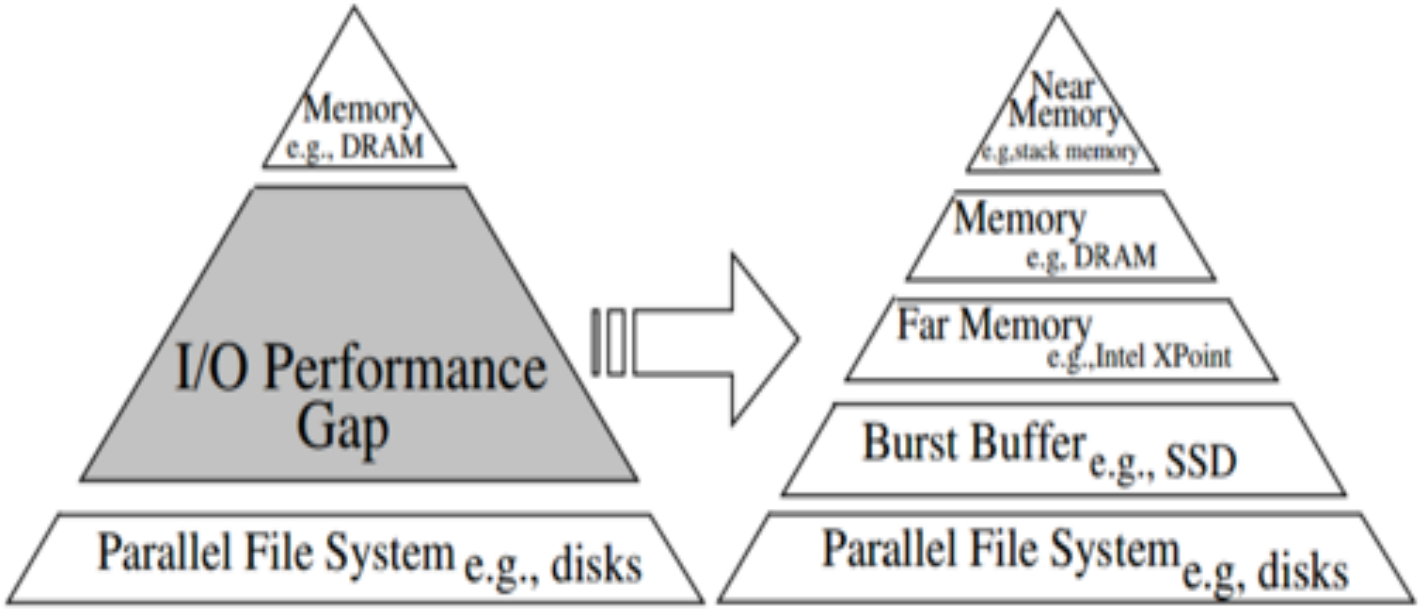
Source: Lucy Nowell (DOE)



Tiered Storage in HPC

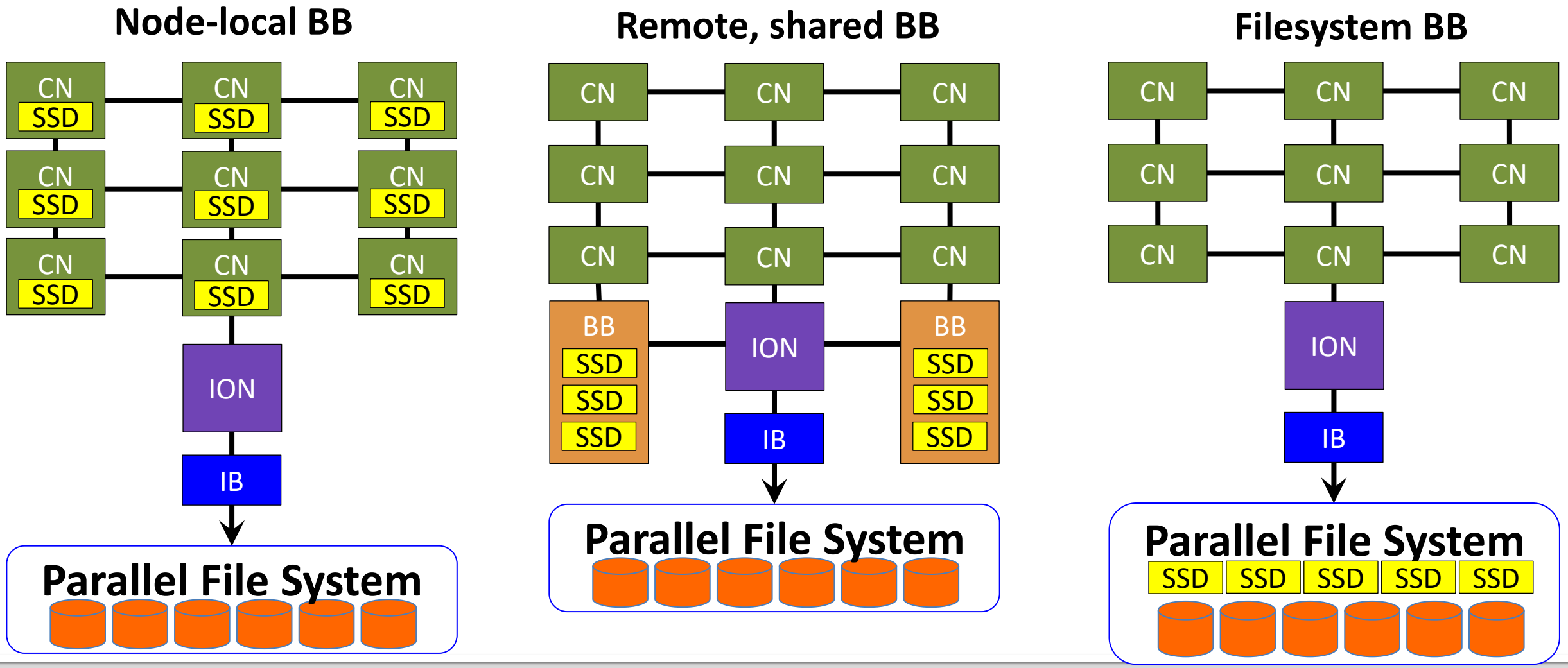


Source: Lucy Nowell (DOE)



How can we use the Flux graph-based scheduler to allocate these new storage tiers?

Burst Buffer Architectures

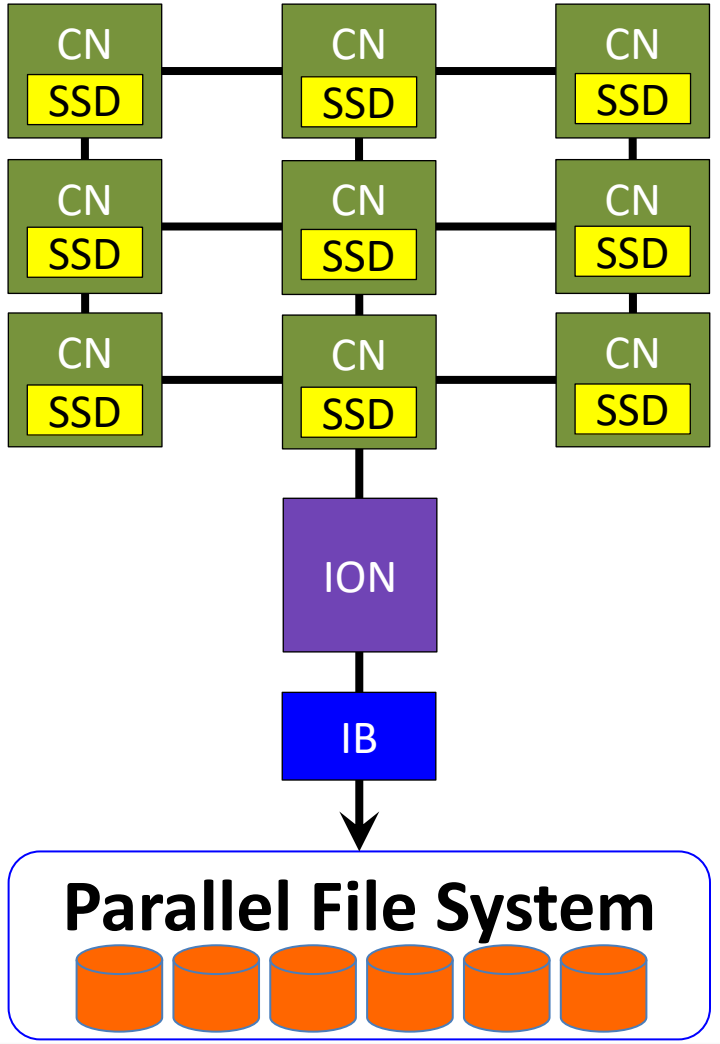


Burst Buffers Functionality “Classes”

- Cache
  - acts like an extended OS page cache for a given job or parallel filesystem
  - all data movement between the BB and the PFS occurs implicitly
- Scratch
  - functions like a /tmp file system for the given node, job, or workflow
- Staging
  - all data movement occurs explicitly for a given job
  - data movement can occur before and after the job is the running

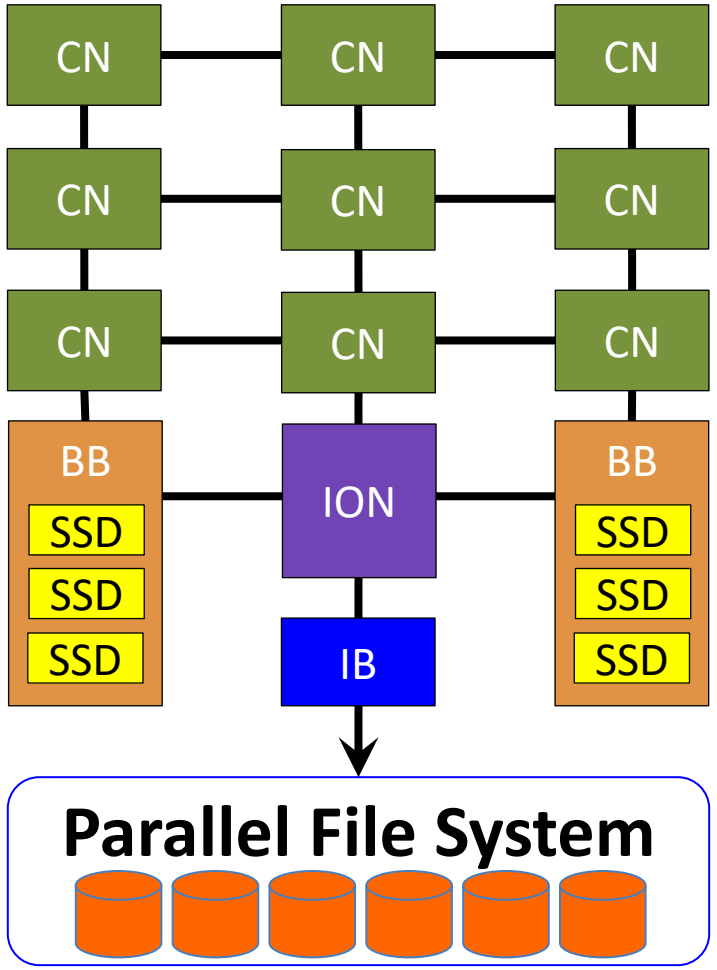
Node-local Burst Buffer Architecture

- Real-World Implementations
  - IBM's Sierra (LLNL) and Summit (ORNL)
- Use-cases
  - Scratch
    - Node-level
    - Job-level via software like SCR, Intel's CPPR, or IBM's BSCFS
  - Staging
- Other notes
  - Application BB bandwidth and space scales linearly with the number of nodes (i.e., cannot get lots of BB with a single node)
  - Staging can cause node "lock-in" with a backfilling scheduler



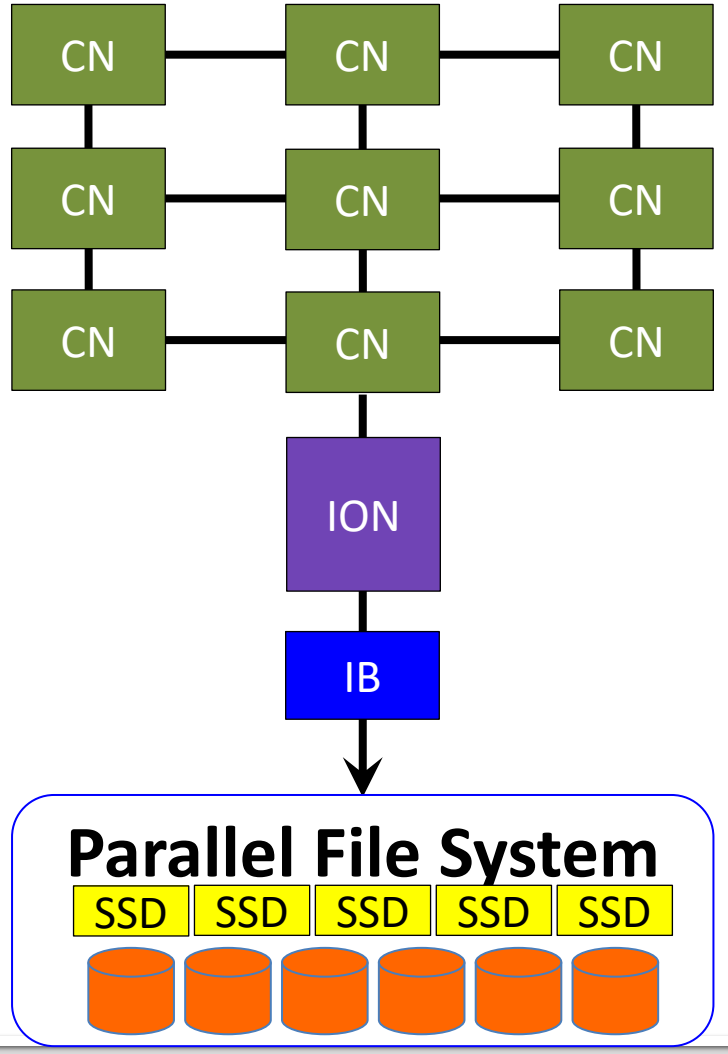
Remote, Shared Burst Buffer Architecture

- Real-World Implementations
  - Cray DataWarp
- Use-cases
  - Cache
    - Job-level
  - Scratch
    - Workflow or job-level
  - Staging
- Other notes
  - BB space and bandwidth can scale independently of number of nodes



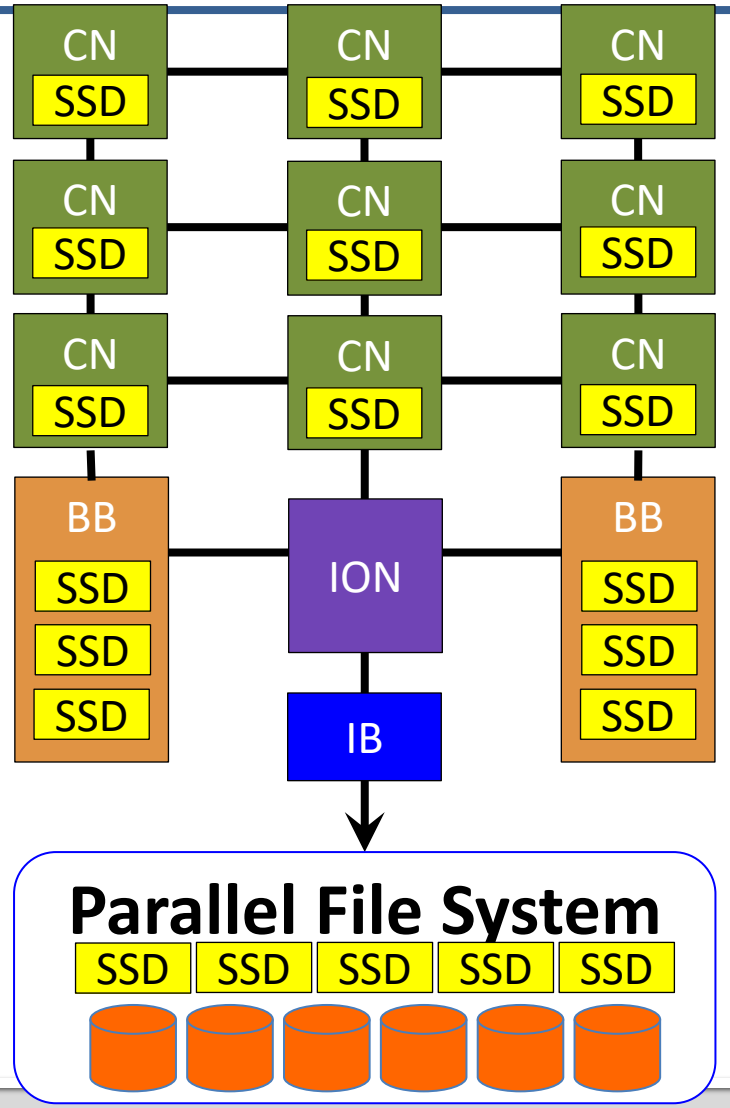
Filesystem Burst Buffer Architecture

- Real-World Implementations
  - DDN Infinite Memory Engine
    - Note: exact architecture details are sparse
- Use-cases
  - Cache
    - Parallel filesystem level
  - Staging
- Other notes
  - Even after staging, data is still far away from the nodes, so network congestion can be an issue



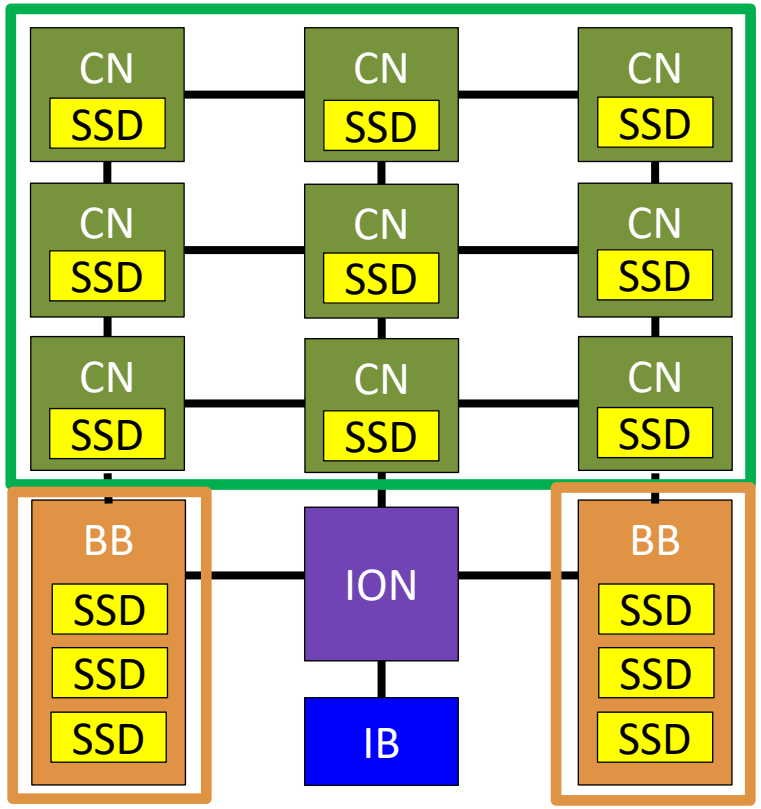
Hybrid Burst Buffer Architecture

- Real-World Implementations
  - None
- Use-cases
  - Cache
    - Job or parallel filesystem level
  - Scratch
    - Workflow, job, or node-level
  - Staging
- Other notes
  - The remote, shared BB could be composed of node-local NVMe devices accessed via RDMA.



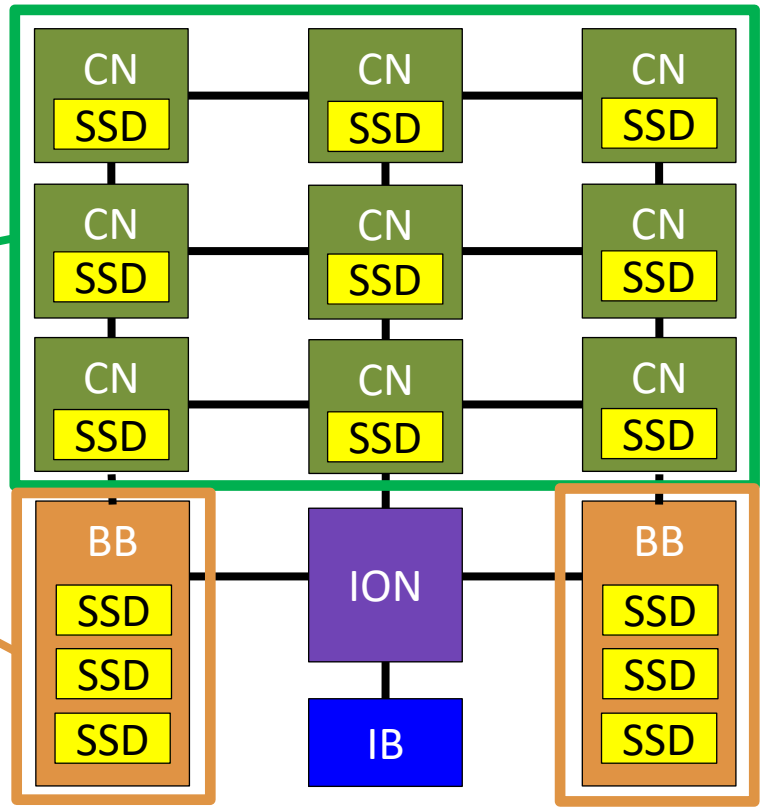
Example Tiered Storage Request in Existing Interfaces

- Application/Workflow Interfaces
  - Job launch CLI
    - `bsub -stage "storage=2:in=stage-in.sh:out=stage-out.sh"`
  - Job script directives
    - `#DW jobdw capacity=4TB type=scratch granularity=per-node`
    - `#DW jobdw capacity=10TB type=cache`
    - `#DW stage_in type=type source=spath destination=dpath`
- Cons
  - Vendor-specific directives in your job-scripts



Example Tiered Storage Request in Existing Interfaces

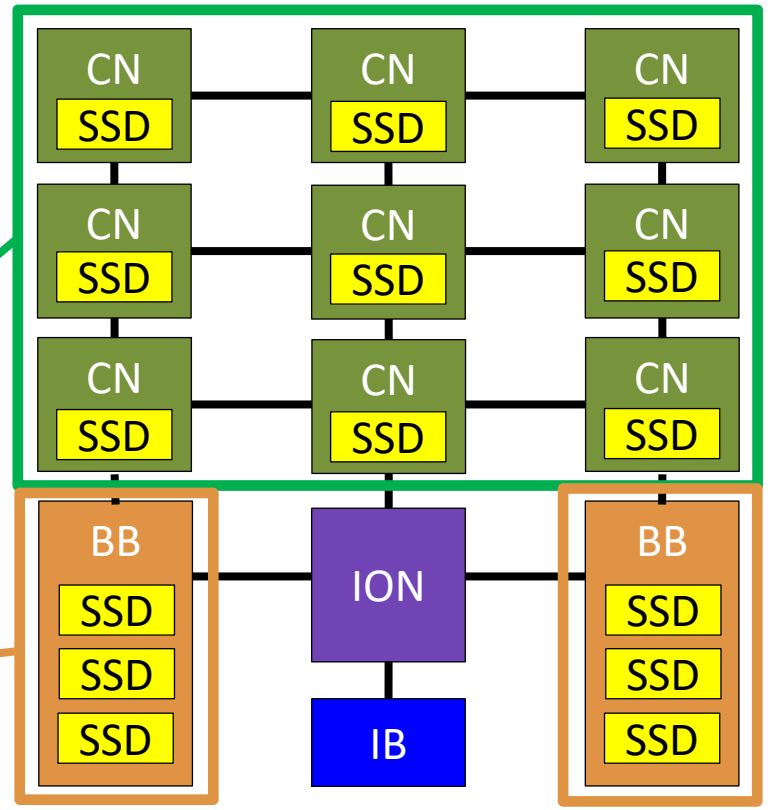
- Application/Workflow Interfaces
  - Job launch CLI
    - `bsub -stage "storage=2:in=stage-in.sh:out=stage-out.sh"`
  - Job script directives
    - `#DW jobdw capacity=4TB type=scratch granularity=per-node`
    - `#DW jobdw capacity=10TB type=cache`
    - `#DW stage_in type=type source=spath destination=dpath`
- Cons
  - Vendor-specific directives in your job-scripts



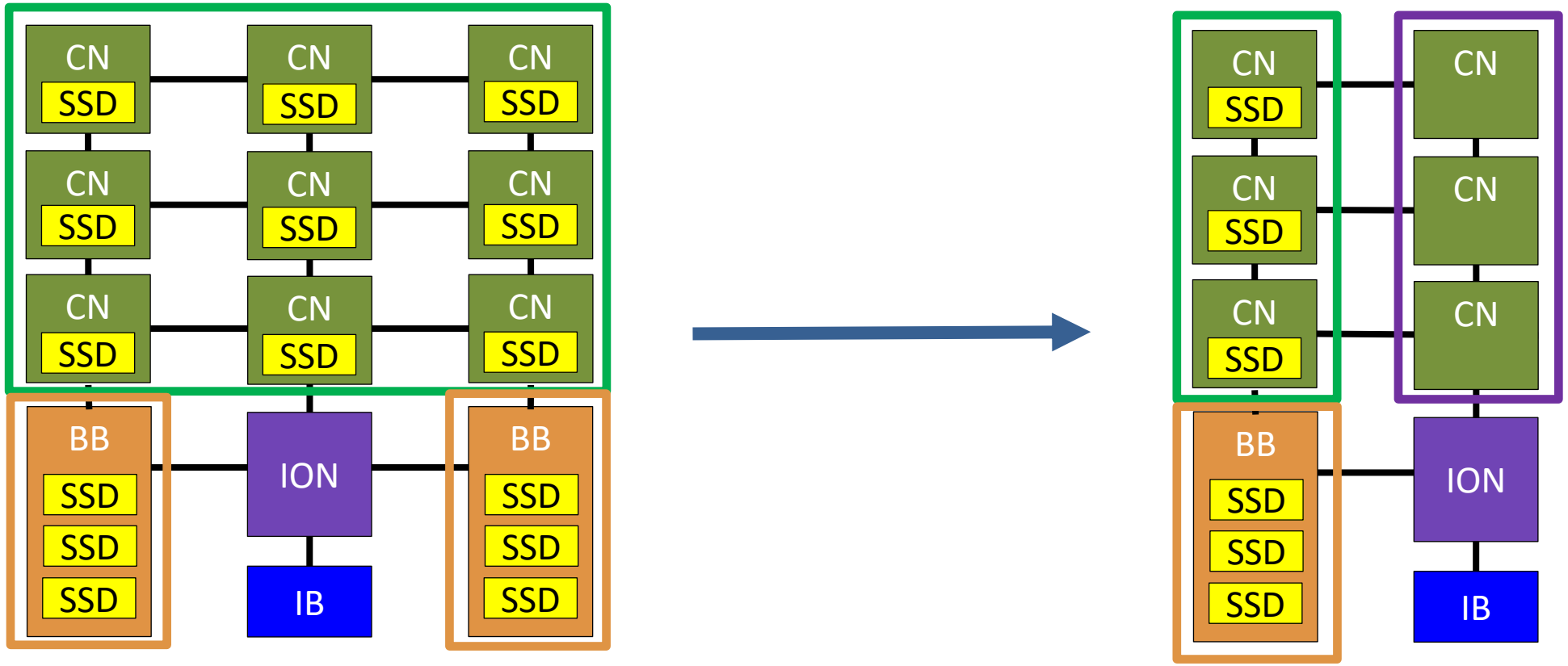
Example Tiered Storage Request in Flux Jobspec

```
resources:
  - type: node
    count: 9
    with:
      - type: slot
        count: 1
        label: default
        with:
          - type: core
            count: 2
      - type: storage
        count: 1
        unit: terabytes
        label: node-local-scratch
  - type: storage
    count: 4
    unit: terabytes
    label: PFS-cache
```

```
attributes:
  storage:
    - label: node-local-scratch
      mode: scratch
      granularity: per-node
      stage-in:
        list: /path/to/stage-in-listing
    - label: PFS-cache
      data-layout: striped
      mode: cache
      stage-in:
        directory: /path/to/PFS
```



Co-Scheduling Request

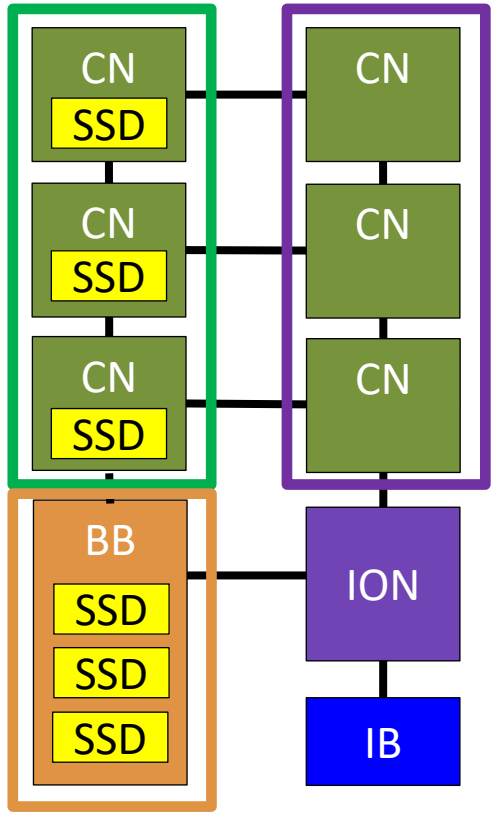


Existing Interfaces

- Application/W… interfaces
  - Jo… CLI

```
bsub -st… "storage=2:in=stage-in.sh:out=sta…-
out.sh"
```

  - Job script directives
    - `#DW jobdw capacity=4TB ty… …cratch`
    - `#DW jobdw capacity=10TB type=ca…`
    - `…W stage_in type=type source=spath`
      `desti… …=dpath`

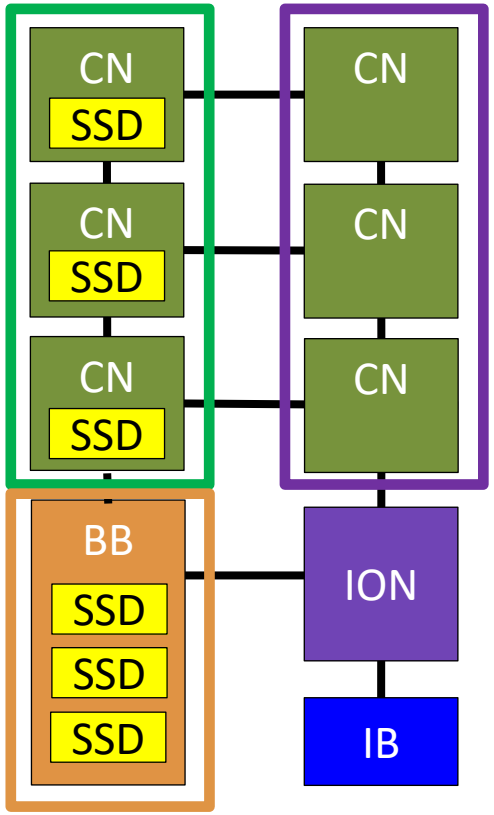


- Job script directives operate at the granularity of the entire job, not job-steps

Example Tiered Storage Request in Flux Jobspec

```
resources:
  - type: node
    count: 3
    with:
      - type: slot
        count: 1
        label: local-ssd-slot
        with:
          - type: core
            count: 2
      - type: storage
        count: 1
        unit: terabytes
        label: node-local-scratch
  - type: node
    count: 3
    with:
      - type: slot
        count: 1
        label: no-ssd-slot
        with:
          - type: core
            count: 2
  - type: storage
    count: 4
    unit: terabytes
    label: PFS-cache
```

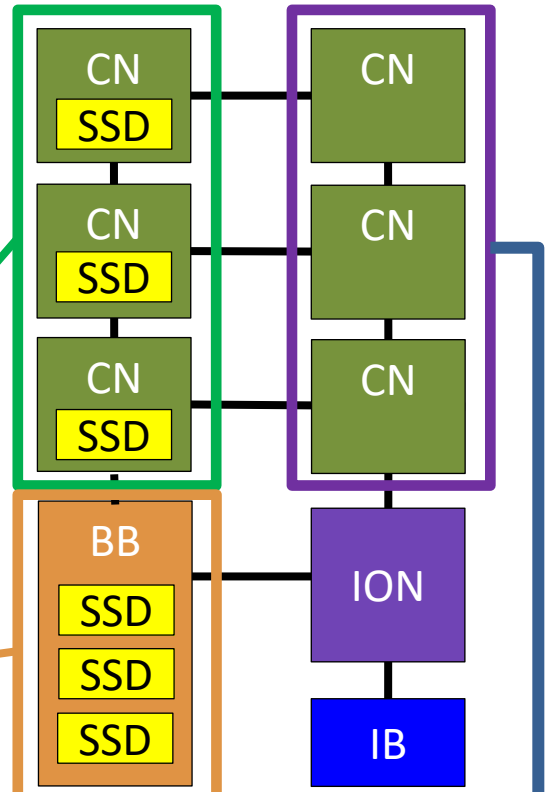
```
attributes:
  storage:
    - label: node-local-scratch
      mode: scratch
      granularity: per-node
      stage-in:
         list: /path/to/stage-in-listing
    - label: PFS-cache
      data-layout: striped
      mode: cache
      stage-in:
        directory: /path/to/PFS
```



Example Tiered Storage Request in Flux Jobspec

```
resources:
- type: node
  count: 3
  with:
    - type: slot
      count: 1
      label: local-ssd-slot
      with:
        - type: core
          count: 2
    - type: storage
      count: 1
      unit: terabytes
      label: node-local-scratch
- type: node
  count: 3
  with:
    - type: slot
      count: 1
      label: no-ssd-slot
      with:
        - type: core
          count: 2
- type: storage
  count: 4
  unit: terabytes
  label: PFS-cache
```

```
attributes:
  storage:
    - label: node-local-scratch
      mode: scratch
      granularity: per-node
      stage-in:
        list: /path/to/stage-in-listing
    - label: PFS-cache
      data-layout: striped
      mode: cache
      stage-in:
        directory: /path/to/PFS
```



Roadmap for Tiered Storage Support in Flux

- Extend **jobspec** to include stage-in and stage-out attributes
  - https://github.com/flux-framework/rfc/issues/204
- Extend **resource** to support multiple top-level resource requests
- Extend **exec** to support multiple slots
- Extend the **IMP** to have a two-phase prologue and support vendor BB APIs
  - launch a prologue across all the nodes and ensure they all succeed before continuing
- Using the **resource** module and **simulator,** demonstrate representations of capacity and locality for relevant BB architectures