2026 OFA Webinar Series

Wednesday, February 25 at 10:00am PT

THE FLUX FRAMEWORK: MODERN, HIERARCHICAL RESOURCE MANAGEMENT FOR MANAGING HPC WORKLOADS

Nathan Hanford, Computer Scientist, and William Hobbs, Software Developer

Livermore Computing Division, Lawrence Livermore National Laboratory

Lawrence Livermore National Laboratory

MEET THE PRESENTERS: Nathan Hanford and William Hobbs, Lawrence Livermore National Laboratory

Nathan Hanford, Computer Scientist, Livermore Computing Division, Lawrence Livermore National Laboratory

Nathan Hanford is a Computer Scientist in the Livermore Computing Division at Lawrence Livermore National Laboratory. His research is currently focused on application and development environment portability for parallel software applications at the application binary interface (ABI). His operational work is focused on development environment design and verification, message passing interface (MPI) support and development, and system-wide accelerator-aware interconnect benchmarking for codesign, system acceptance, and strategic decision-making support.

Nathan came from a high-performance networking background. While earning his PhD at University of California Davis, he was a perennial summer student with ESnet at Lawrence Berkeley Laboratory. During this time, he focused on end-system optimizations and congestion avoidance for high-speed, long-distance networking.

William Hobbs, Software Developer, Livermore Computing Division, Lawrence Livermore National Laboratory

William Hobbs is a software developer in the Livermore Computing Division of Lawrence Livermore National Laboratory. Hobbs works on the Flux Framework: a modern, hierarchical resource management framework for managing HPC workloads at scales from containers to the world's fastest supercomputers. He maintains and develops Flux plugins, the Fluxion scheduler, and I/O plugins for LC clusters. His previous experience is with software development and data analysis for bioinformatics and manufacturing teams. Hobbs has worked with the Flux Framework team since 2021 and joined the Laboratory in 2023.

Hobbs was born and raised in South Carolina, lives in Oakland, California and holds a B.S. in Computer Science from the University of South Carolina.

Flux for OpenFabrics Designers & Developers

An overview of resource managers used for HPC systems at LLNL + the unique development capabilities of Flux

Nathan Hanford and William Hobbs

Lawrence Livermore National Laboratory

OpenFabrics Alliance Webinar Series 2/25/2026

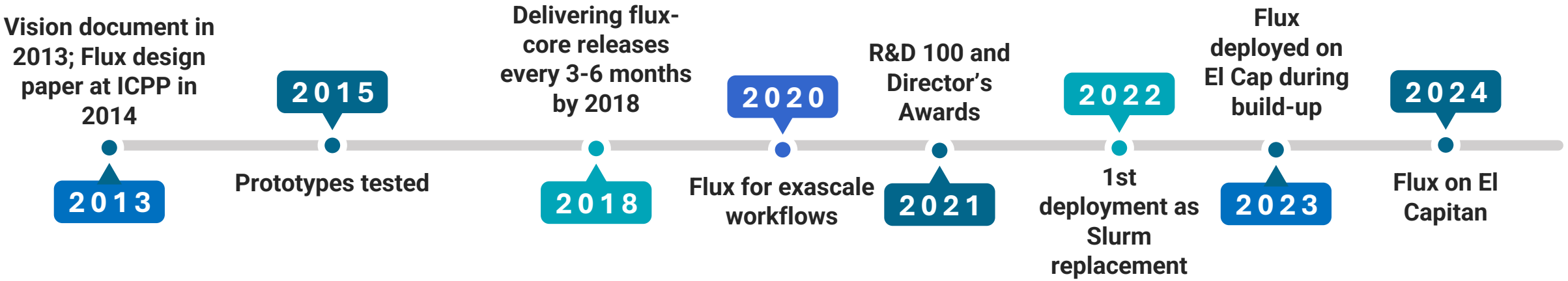
Prepared by LLNL under Contract DE-AC52-07NA27344.

LLNL-PRES-2016005

The Resource Manager is a user's entry point to HPC resources

- **Resource managers** provide scheduling, job launch, resource specifications, the PMI service for MPI jobs, accounting/bank information, and limits
- Within the Flux Framework, we split these responsibilities into multiple components:
  - **flux-core** for the underlying messaging and basic job launch
  - **flux-sched** for advanced, graph-based, and arbitrary scheduling
  - **flux-security** for multi-user instances
  - **flux-accounting** for user limits, bank, and accounting information
  - **flux-pmix** for providing the PMI services required to support OpenMPI v5+

A brief bit of Flux history…



Flux Strategy – *Build framework for user-level as well as system-level RMs with hierarchical scheduling*

- Flux addresses issues with classical RMs
- Portable and efficient
- Collaboration-rich environment
- Expanding to clouds
- Used for workflows at NERSC, ORNL, and some European sites

Next-Gen Workflow Enablement – *Flux teams with scientists and CS researchers on workflows*

- MuMMi, SC19 Best Paper-winning workflow for studying RAS interactions with cancer
- AHA MoleS, eScience'22 Best Paper and American Heart Association drug discover workflow
- Converged Computing LDRD; FRACTALE Strategic Initiative

Flux *is* the system scheduler for El Capitan

and, a number of other systems on the Top500:

1.	El Capitan	*(LLNL)*
12.	Tuolumne	*(LLNL)*
33.	El Dorado	*(SNL)*
72.	rzAdams	*(LLNL)*
288.	Tioga	*(LLNL)*
366.	Tenaya	*(LLNL)*

Flux grew enormously to fill acceptance needs

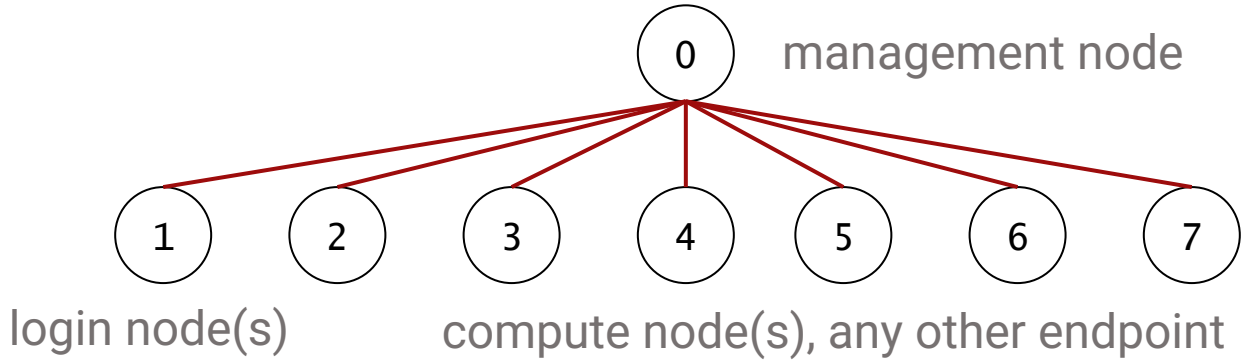
- ✓Flux was chosen as the scheduler for El Capitan in part due to support for arbitrary resource types (i.e. rabbits)
- ✓Scaled from 200 nodes to 11,520 nodes (~1,000,000 ranks)
- ✓Spun up new framework project (flux-coral2) to create infrastructure for Cray PMI and Rabbit support
- ✓Serious engagement with system administrators and HPE partners
- ✓Enormous growth in user base

Corona cluster, previously largest Flux system-instance

The Flux Framework is a *suite* of projects that implement many innovations from LLNL expertise

- **Flux** is a job manager, launcher, and scheduler underpinned by a distributed key-value store and overlay network
- Flux can be run completely in user-space and without elevated privilege
- Flux separates concerns through *broker modules* and *plugins* that allow a scheduler and hardware-specific code to remain out of the flux-core tree
- Flux's ability to be run in user-space allows for *hierarchical launch* of jobs, meaning that batch jobs can be scheduled, manipulated, and further subdivided ad infinitum

Flux brokers create a distributed overlay network on the cluster and enable modules at each level

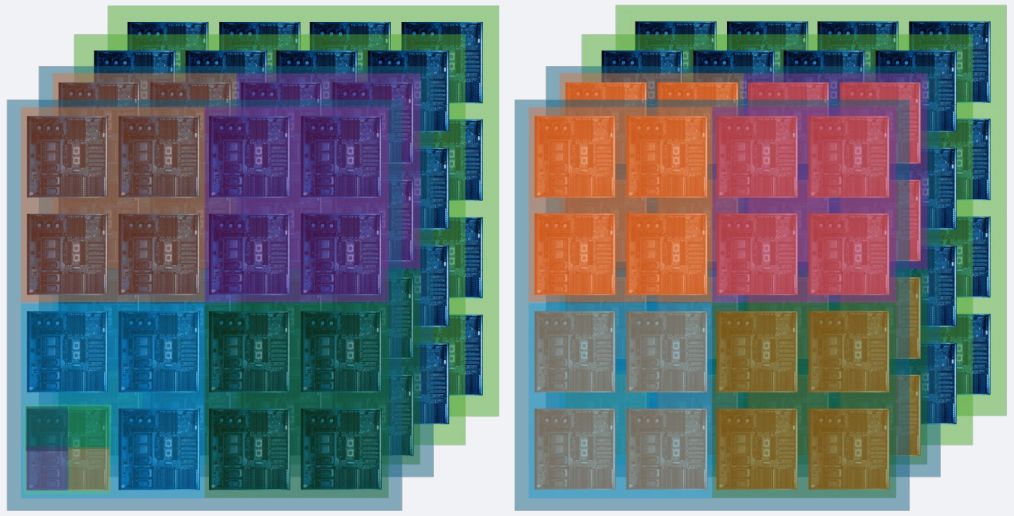



```
$ flux module list
Module                   Idle  S Sendq Recvq Service
resource                   27  R     0     0
connector-local             0  R     0     0
heartbeat                   1  R     0     0
groups                     27  R     0     0
job-manager                28  R     0     0
sched-fluxion-feasibility  27  R     0     0 feasibility
content                    29  R     0     0
job-exec                   29  R     0     0
job-ingest                 29  R     0     0
job-list                   29  R     0     0
sched-fluxion-qmanager     28  R     0     0 sched
sched-fluxion-resource     27  R     0     0
job-info                   29  R     0     0
content-sqlite             29  R     0     0 content-backing
kvs                        29  R     0     0
kvs-watch                  29  R     0     0
cron                       29  R     0     0
$
```

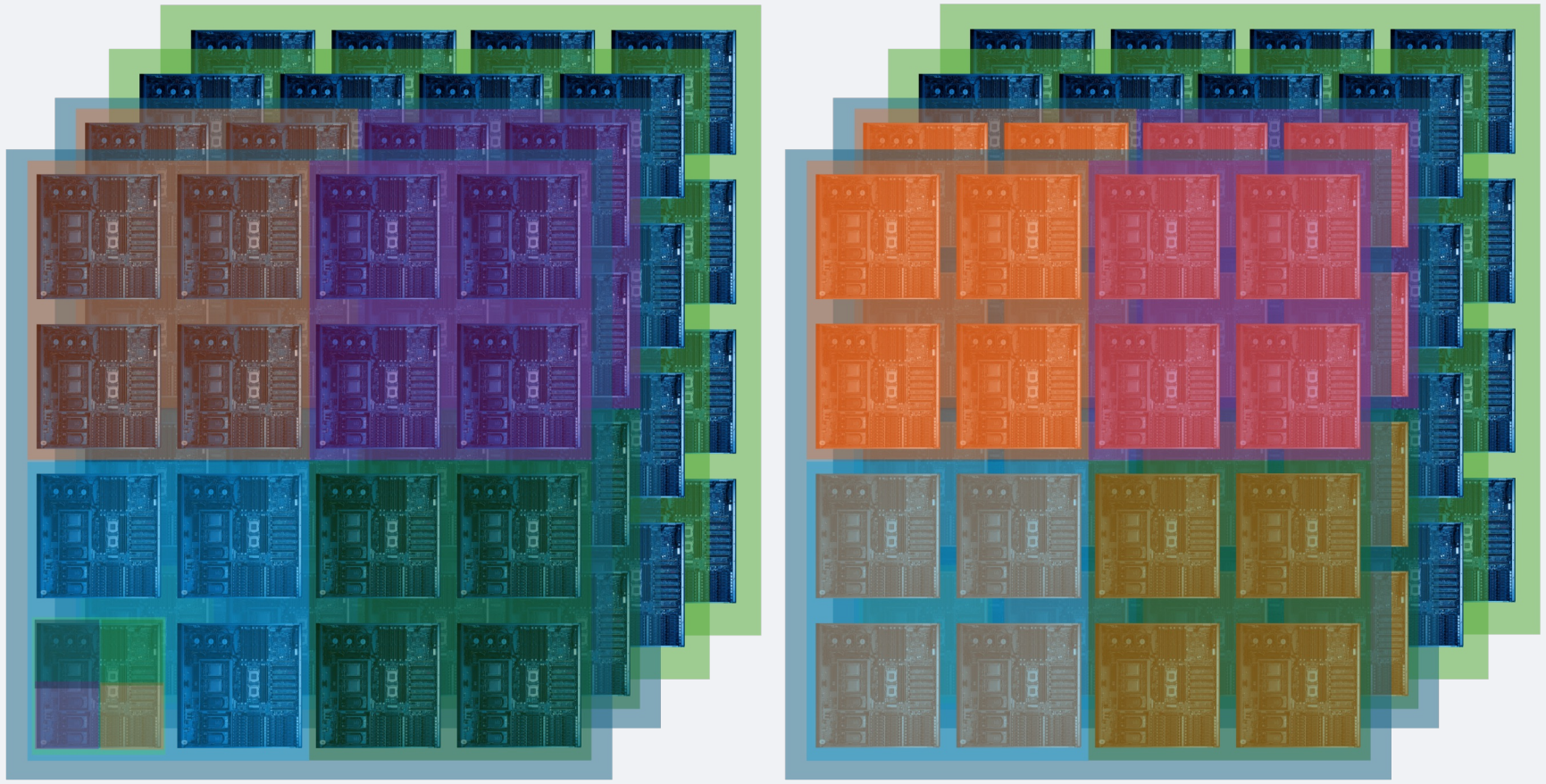
Modules loaded on an example rank 0

- At rank 0, many critical services are loaded, such as the scheduler
- Many modules run at all ranks, such as the key-value store (KVS)
- Modules receive and respond to messages sent over TCP/IP via ZeroMQ sockets

Flux level 0 (system)

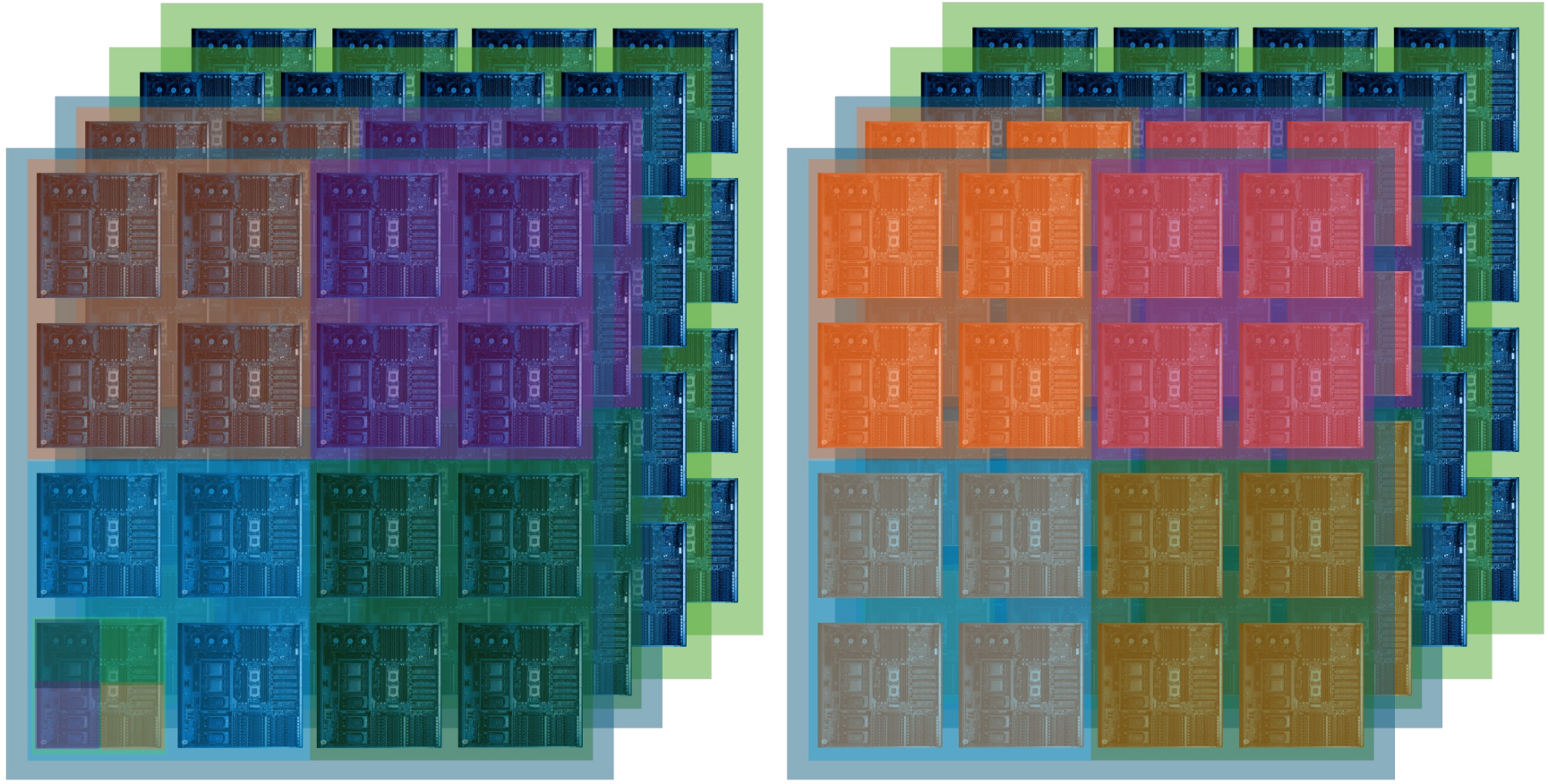


Flux level 0 (system)



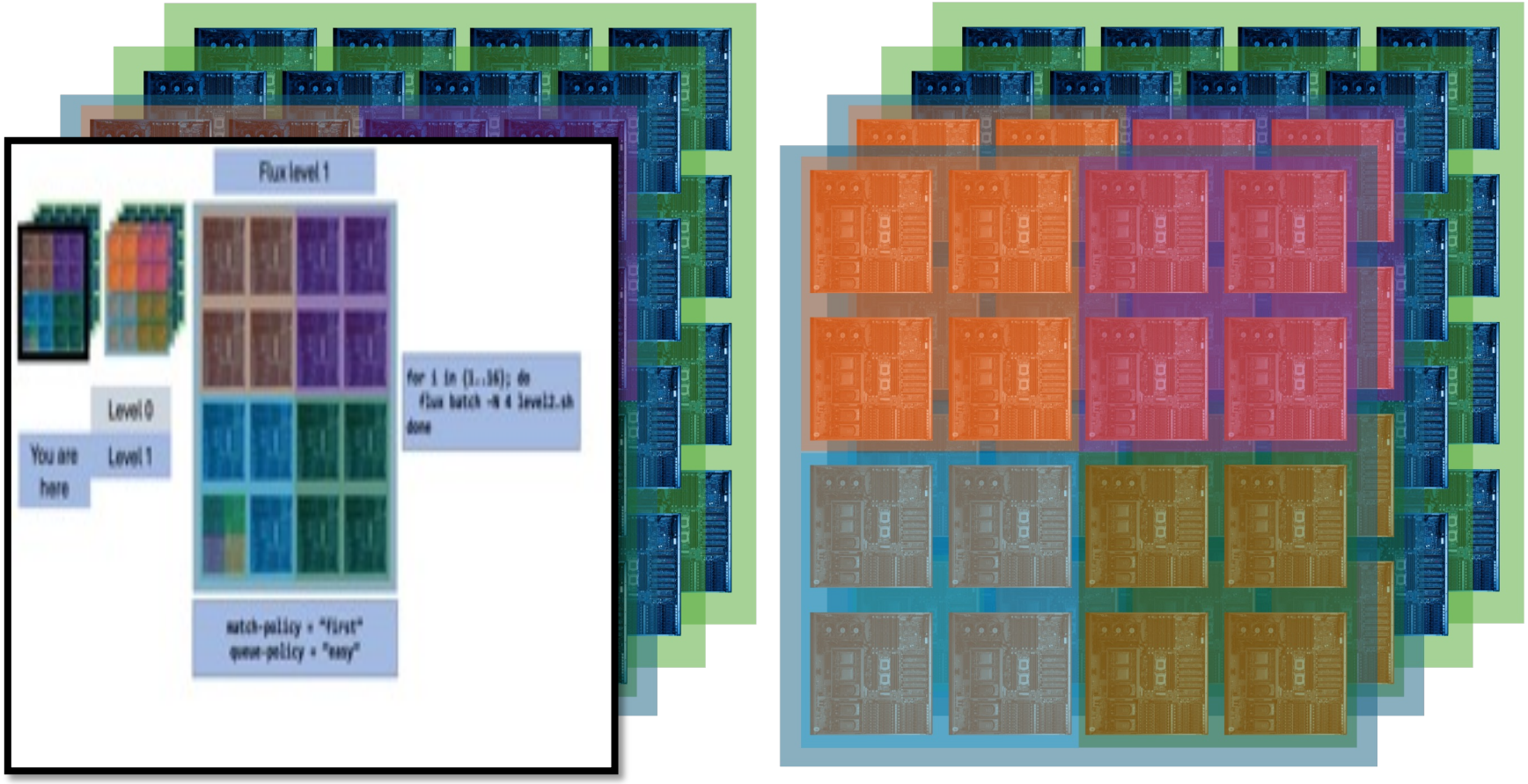
```
match-policy = “first”
queue-policy = “easy"
```

Flux level 1



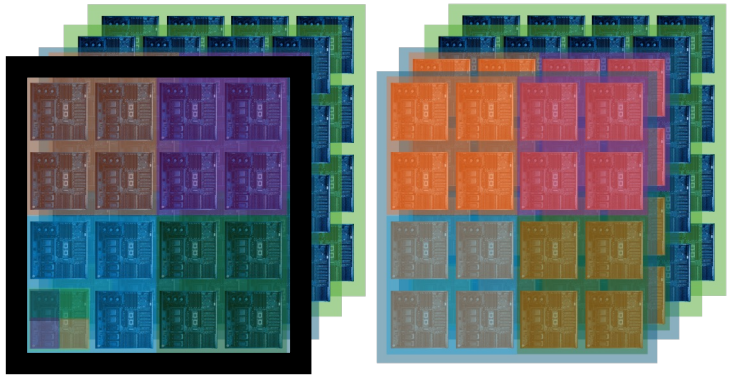
```
flux batch -N 64 -x level1.sh
match-policy = “first”
queue-policy = “easy"
```

Flux level 1




```
flux batch -N 64 -x level1.sh
match-policy = “first”
queue-policy = “easy"
```

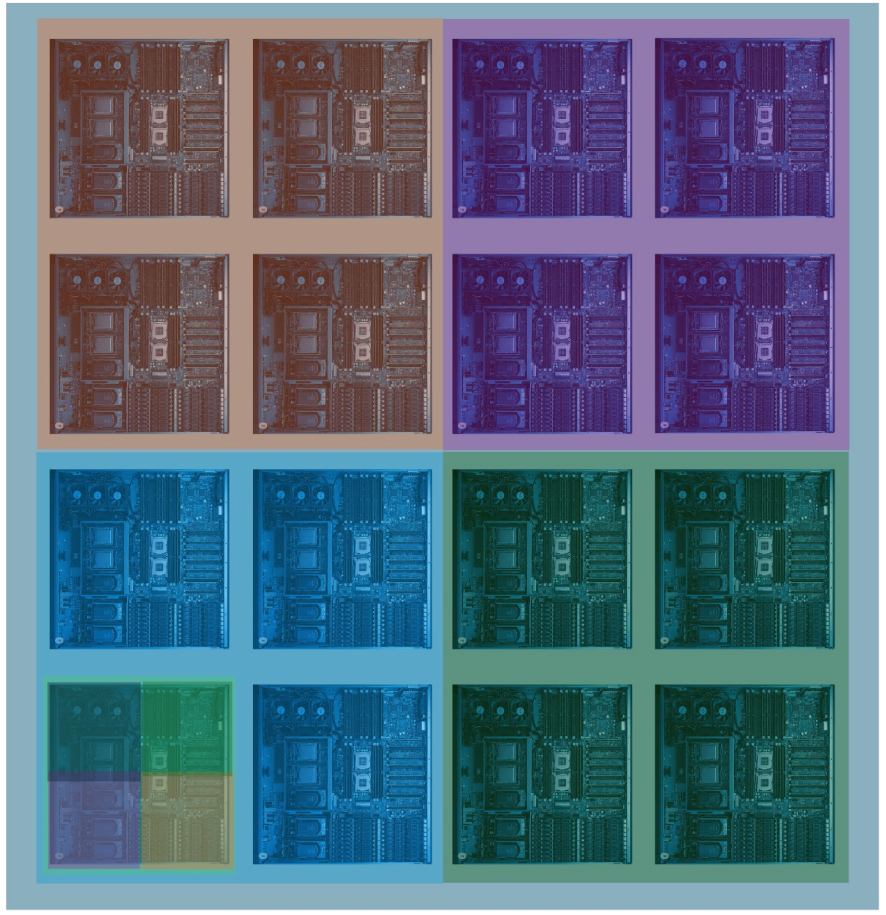
Flux level 1



Level 0

You are here

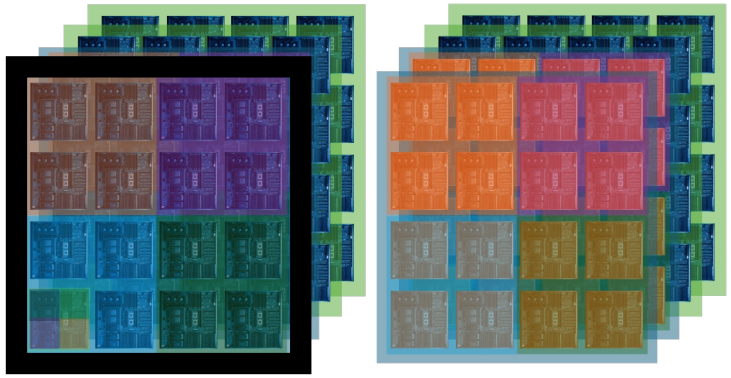
Level 1



```
for i in {1..16}; do
  flux batch -N 4 level2.sh
done
```

```
match-policy = “first”
queue-policy = “easy”
```

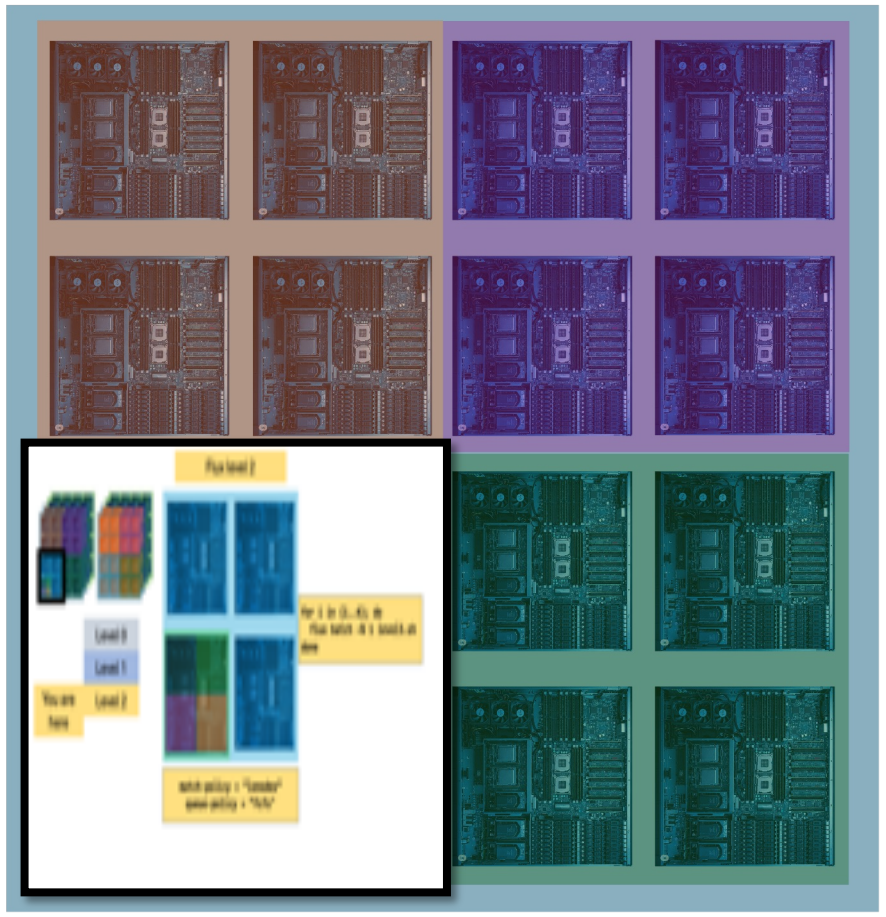
Flux level 1



Level 0

You are here

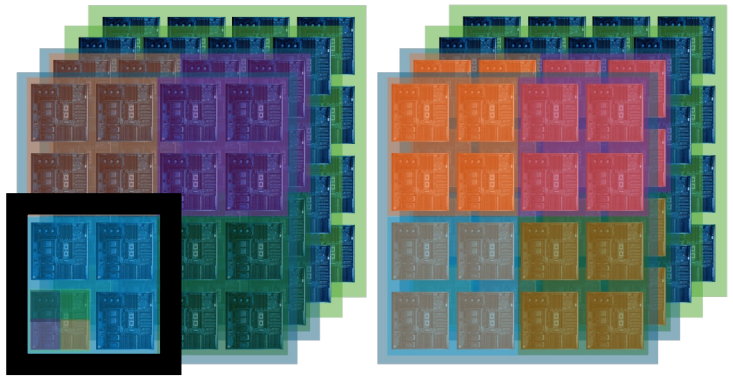
Level 1



```
for i in {1..16}; do
  flux batch -N 4 level2.sh
done
```

```
match-policy = “first”
queue-policy = “easy”
```

Flux level 2

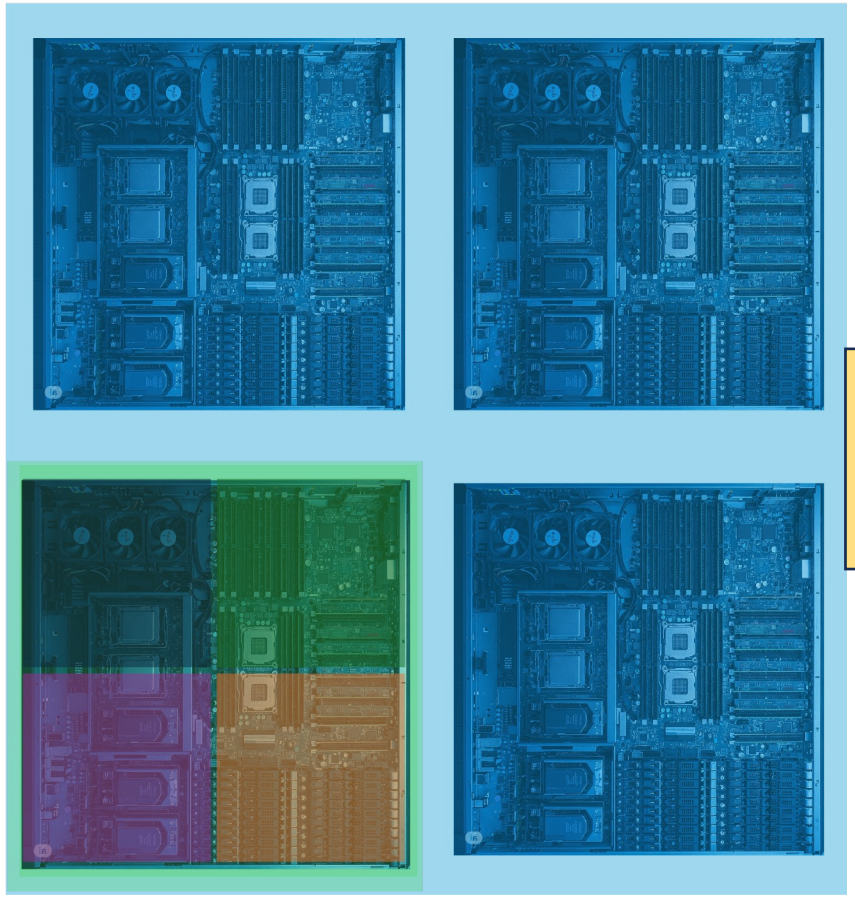


Level 0

Level 1

You are here

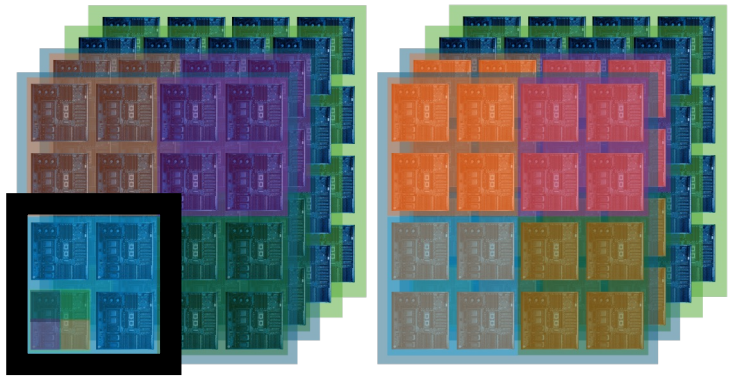
Level 2



```
for i in {1..4}; do
  flux batch -N 1 level3.sh
done
```

```
match-policy = “lonodex”
queue-policy = “fcfs"
```

Flux level 2

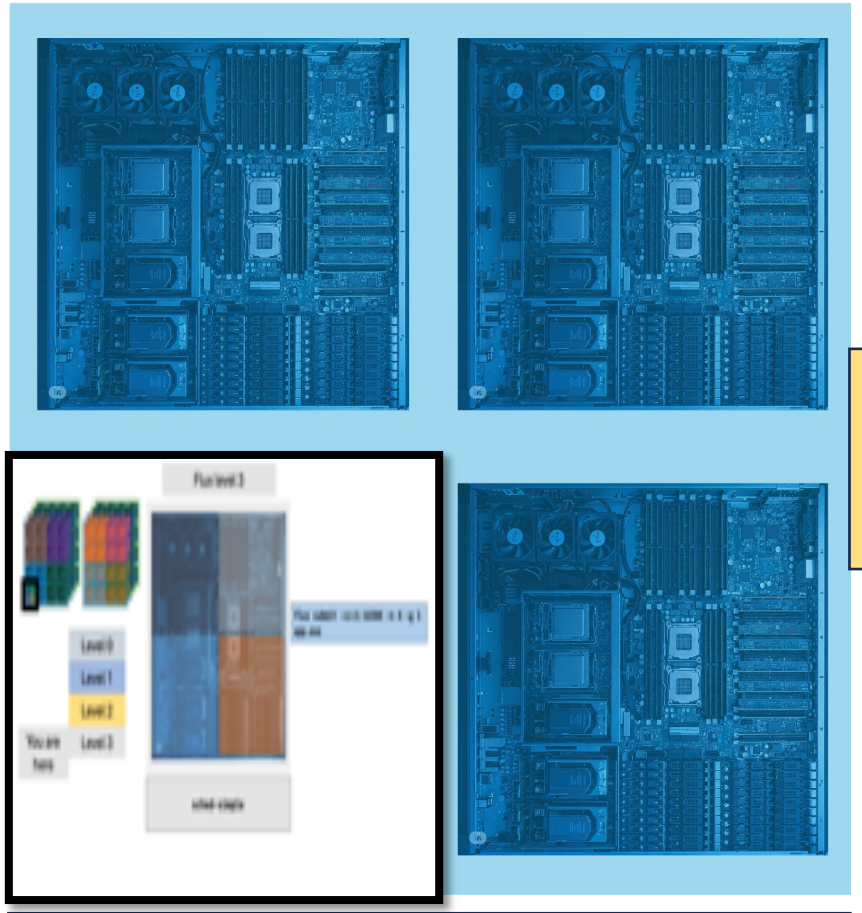


Level 0

Level 1

Level 2

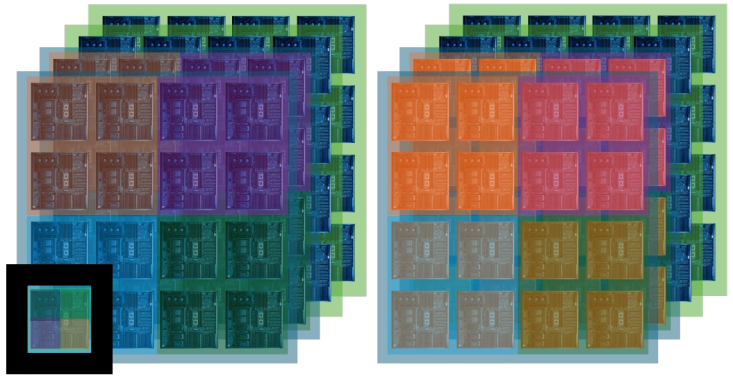
You are here



```
for i in {1..4}; do
  flux batch -N 1 level3.sh
done
```

```
match-policy = “lonodex”
queue-policy = “fcfs"
```

Flux level 3



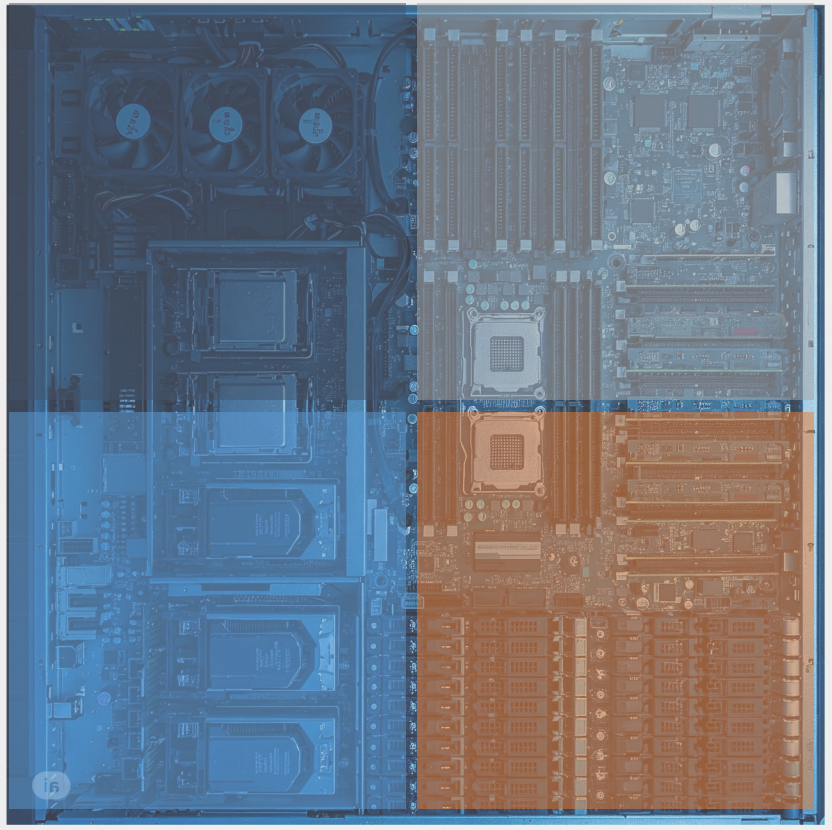
Level 0

Level 1

Level 2

Level 3

You are here



```
flux submit -cc=1-16384 -n 4 -g 1 app.exe
```

```
sched-simple
```

Some “above all” Flux design principles

- Preserve **separation of concerns** in our recursive model
  - Calls “up” to the system instance from a user instance are not recommended
- **Portable layer** across Linux distributions (and even starting to support macOS) that can be run on *anything*
- Hardware-specific and fabric-specific code is implemented as **plugins** in separate projects
- **Principle of least privilege** minimizes code that requires elevated privilege into a separate project, and elevated privilege plugins are supported
  - All code requiring elevated privilege is kept in a separate repository, flux-security

Flux can run just about anywhere

- Under another resource manager
- Within a docker container
  - On DockerHub, official container images are provided
  - **fluxrm/flux-core** provides a container with a pre-built latest Flux binary
  - **fluxrm/flux-sched** provides everything in the flux-core container, along with the flux-sched (“fluxion”) scheduler
  - **fluxrm/testenv** provides a container with all the required packages to build Flux from source
  - **Progress toward providing more containers specifically for fabrics/plugin developers** (built on the base of the ones above)
- Locally, on Linux laptops/desktops (with progress toward macOS)

For developers: SPANK plugin & Flux plugin capabilities

	SPANK plugins	Flux CLI plugins (and connected interfaces)
Base language	C/Lua wrappers	Python
Can be loaded/customized to an individual user's terminal	No	Yes
Can be developed to activate additional functionality in the job	Yes	Yes
Ability to parse user input	Limited to string parsing supported by C/Lua	Richer interface allows for JSON to be passed with more options for customization
Associated functionality can activate root-level features configured by system admins	Yes	Yes, and Flux shell plugins allow more specifically scoped functionality as well

Takeaways from the hierarchical model

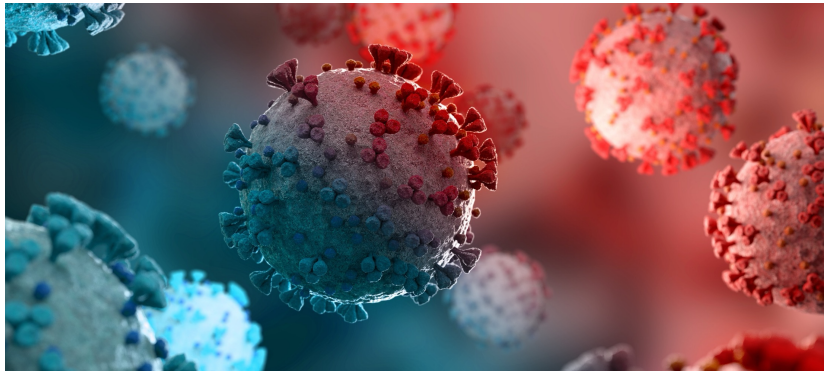
- Infinitely recursive ability to create sub-instances mean that *job steps* are just *jobs*, but do not hit the system-instance
  - This allows users to create more “job steps” than they would otherwise be able or advised to
- User sub-instances have no access to services that require elevated privilege (i.e. a sub-instance cannot register an elevated privilege prolog)

How a ‘typical’ user runs jobs on our clusters

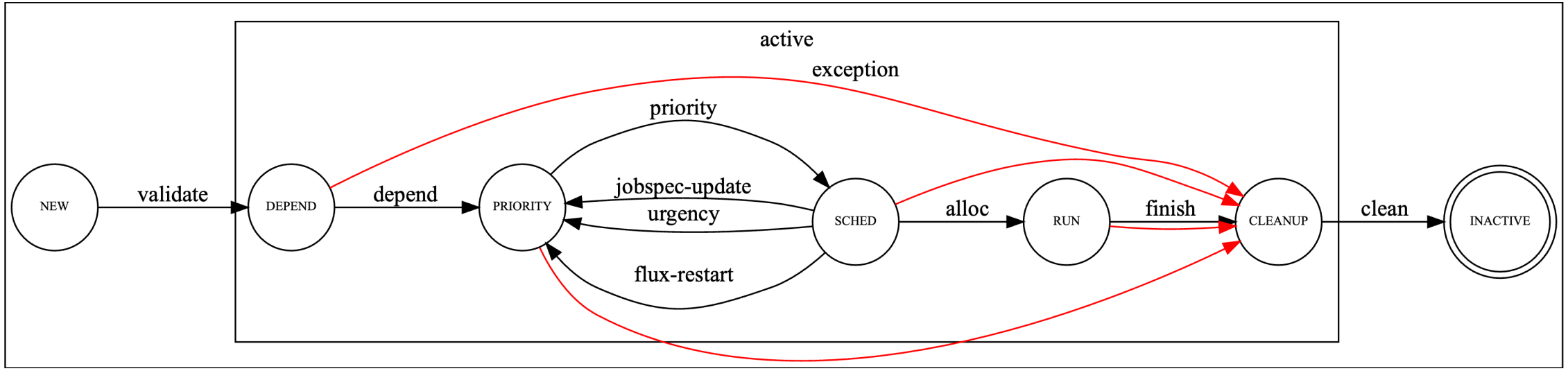
```
$ flux alloc -N 10 --requires=hosts:fluke[62-71] --conf=queues.toml -t 8h
```

```
#!/bin/sh
#FLUX: -N4                  # Request four nodes
#FLUX: --queue=batch        # Submit to the batch queue
#FLUX: --job-name=app001    # Set an explicit job name
flux run -N4 app
```

```
import flux
import flux.job

def submit_cb(fut, flux_handle):
        # when this callback fires, the jobid will be ready
        jobid = fut.get_id()
        # Create a future representing the result of the job
        result_fut = flux.job.result_async(flux_handle, jobid)
        # attach a callback to fire when the job finishes
        result_fut.then(result_cb)

def result_cb(fut):
        job = fut.get_info()
        result = job.result.lower()
        print(f"{job.id}: {result} with returncode {job.returncode}")

flux_handle = flux.Flux()
jobspec = flux.job.JobspecV1.from_command(["/bin/true"])
for _ in range(5):
        # submit 5 futures and attach callbacks to each one
        submit_future = flux.job.submit_async(flux_handle, jobspec)
        submit_future.then(submit_cb, flux_handle)
# enter the flux event loop (the 'reactor') to trigger the callbacks
# once the futures complete
flux_handle.reactor_run()
```



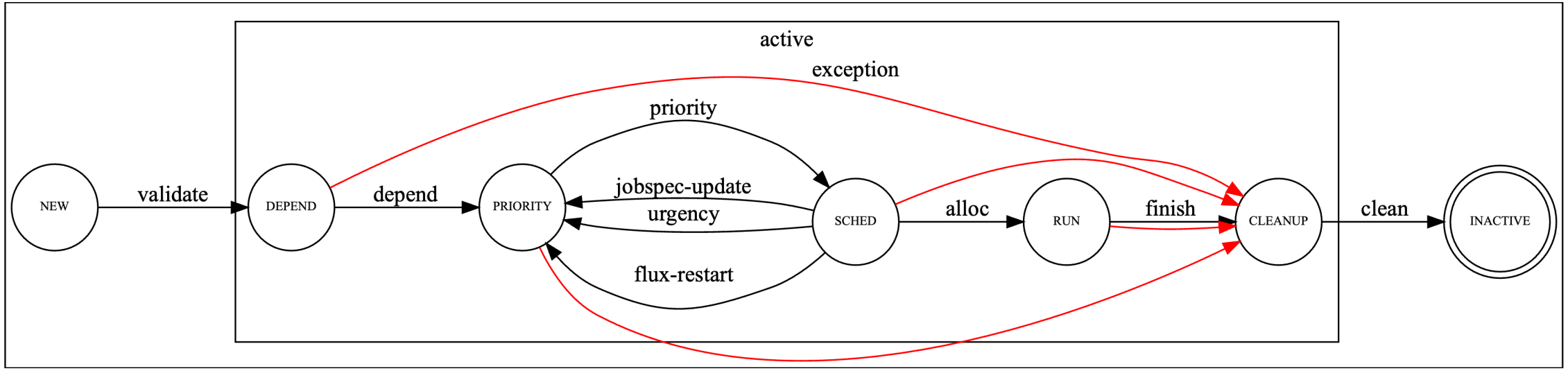
A state machine is embedded in the ”arrow”



Flux RFC 21 / Job States and Events

- **New** jobs get validated, evaluated for dependencies, and a priority is calculated
- At each state in this state diagram *callbacks* can be registered in the job manager

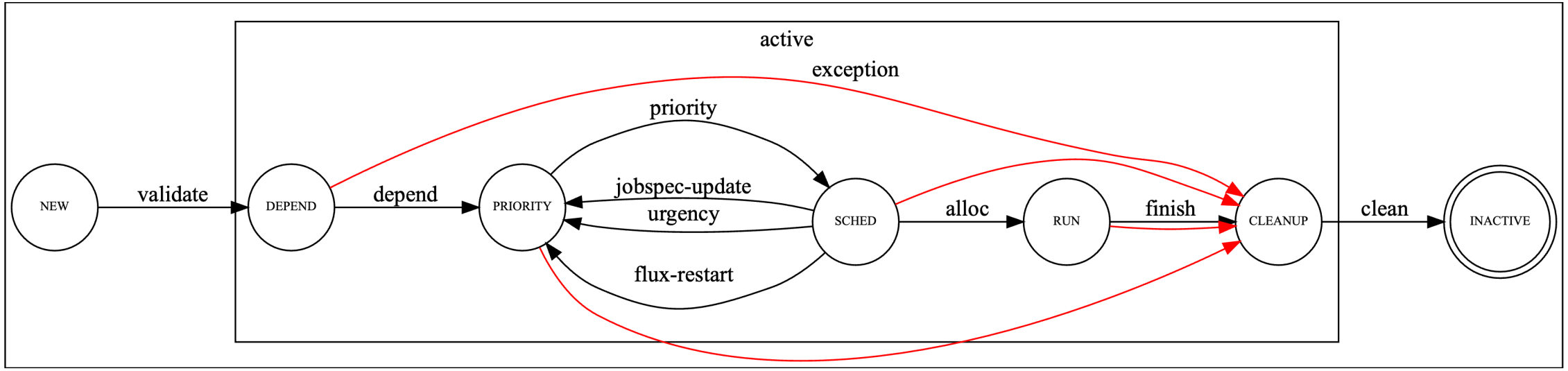
A state machine is embedded in the ”arrow”



Flux RFC 21 / Job States and Events

- Inside the box, the job is considered ”active” and a log of all events is kept
- Both jobs at the system-instance level (“batch jobs”) and jobs within a sub-instance (“job steps”) go through this state machine

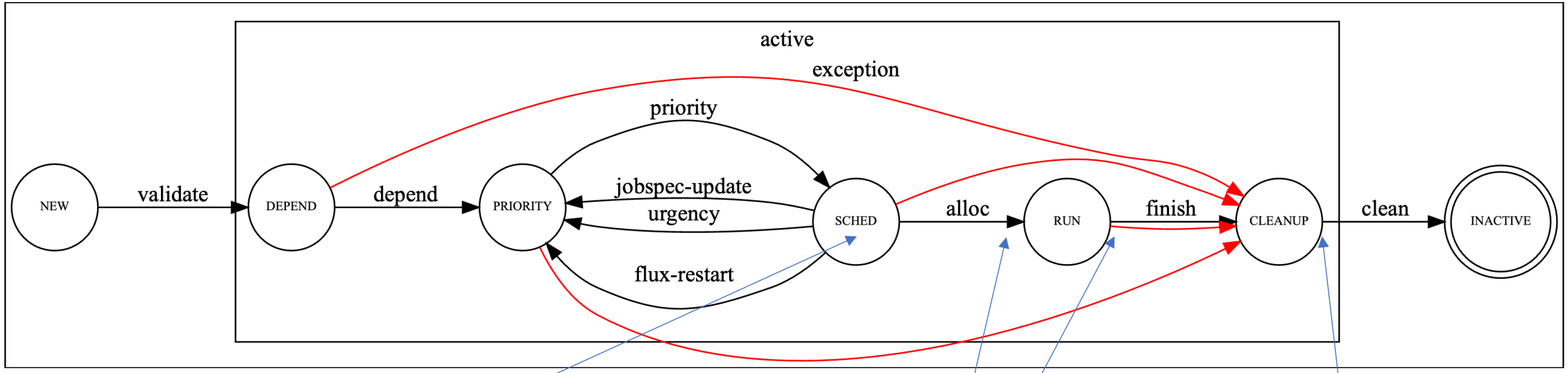
A state machine is embedded in the ”arrow”



Flux RFC 21 / Job States and Events

- In addition to callbacks, before and after a job RUNs, an elevated privilege “prolog” and “epilog” runs on each node in the allocation for system services

Example use for plugins: Cray VNI support



Flux RFC 21 / Job States and Events

VNI ”tag” issued (round robin 64k integer) through a “jobtap plugin” (event callback)

VNI ”tag” released back into Flux-managed pool of tags

CXI service started via prolog / shut down via epilog

Getting Flux

- The Flux project has containers on DockerHub.
- Flux can be installed from SRPMs.
- Flux can also be installed via Spack or from source.

Using Flux: Commands and Interfaces

- Start a broker with `flux start`.
- Run a job with `flux run`.
- Run a job from a batch script with `flux batch`
  - Directives look like `#flux: --nodes 4`
- See your running jobs with `flux jobs`

Container Demo

- `docker run -it wihobbs/flux-demo`

- This container will pull its image from https://hub.docker.com/r/wihobbs/flux-demo

- Container image will stay available after this talk, but “frozen:” use the official containers provided under **fluxrm** for guaranteed up-to-date containers

Flux Maturity & Future Work

El Capitan proved that a Flux instance of nearly any size is possible and stable, but we expect progress to be made on several important features in 2026:

- **Rolling software upgrades:** Current upgrades to the Flux system instance require downtime and a full restart of the instance, losing running jobs.
- **Restart with running jobs:** restarting any Flux instance currently kills running jobs.
- **Reservation support:** support for deferring a job/reservation start time into the future, which is currently supported with admin intervention, scripting and queues.
- **Package availability:** Source RPMs are published with each tagged release, although `dnf install flux-core` is not currently available. Collaboration is underway with Fedora/Red Hat for inclusion in their Extensions Repository.

Upcoming Flux Events & Contact Information

- Open GitHub issues for questions or bugs
  - flux-framework/flux-core

- Flux Framework Project Meeting at HPSF Conference 2026
- flux-discuss@lists.llnl.gov
- Slack invitation by request

Join us for the first-ever Flux Community Meeting!

We're excited to announce that at the HPSF Conference in Chicago on **March 17th from 1:45-4:30 PM,** we'll have an afternoon filled with insight from Flux developers and admins:

- *Supporting Slurm Users on Flux Clusters,* presented by Ryan Day
- *I Like to Kube It, Kube It: Move your AI/ML workloads with Flux,* presented by Vanessa Sochat
- *An Introduction to Plugins in the Flux Framework,* presented by Hobbs
- *The Evolution of Job Priority and Administration on Systems Running Flux,* presented by Christopher Moussa
- *Resource-Aware Benchmarking and Performance Tuning of Deep Learning I/O Pipelines using Flux,* presented by Hari Harihan Devarajan

2026 OFA Webinar Series

THANK YOU

Nathan Hanford and William Hobbs

Lawrence Livermore National Laboratory

Documentation of Interest

- Manual Pages for flux-core
- Flux RFC 13/Simple Process Management Interface (PMI) v1
- Slingshot CXI Service/VNI Implementation in Flux
- Flux Administrator's Guide
  - Specifically software components and security
- Flux KVS Internals