Flux: Solving Exascale Workflow and Resource Challenges

Plus - How Open-Source Drives Our Project Design

Dong H. Ahn, Ned Bass, Albert Chu, Frank Di Natale, Jim Garlick, Mark Grondona, **Stephen Herbein**, Joseph Koning, Chris Moussa, Daniel Milroy, Tapasya Patki, Thomas R. W. Scogland, Becky Springmeyer

LLNL-PRES-818397

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

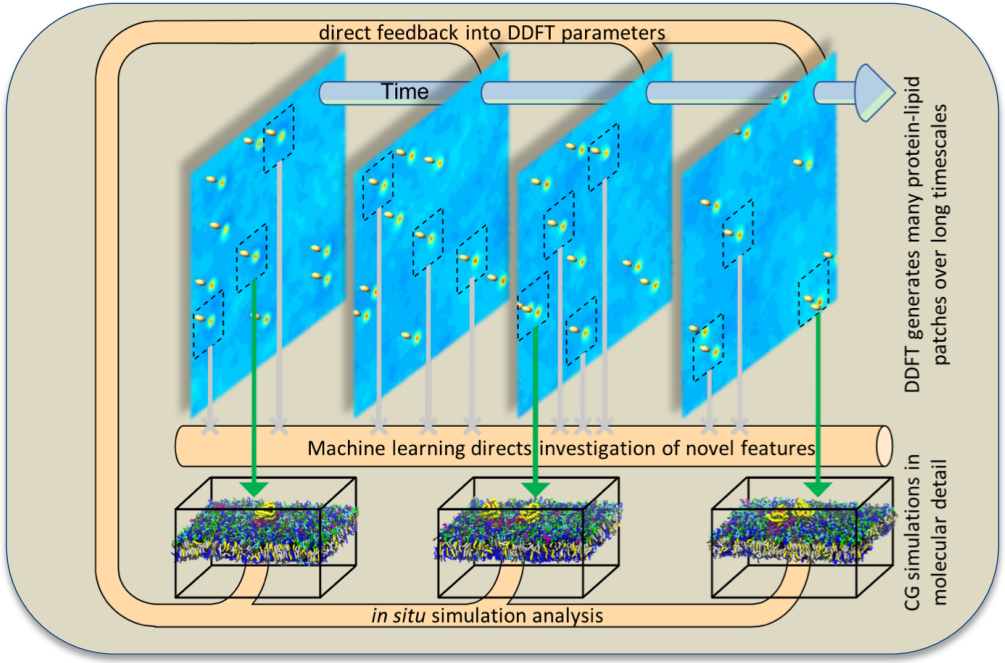
Lawrence Livermore National Laboratory

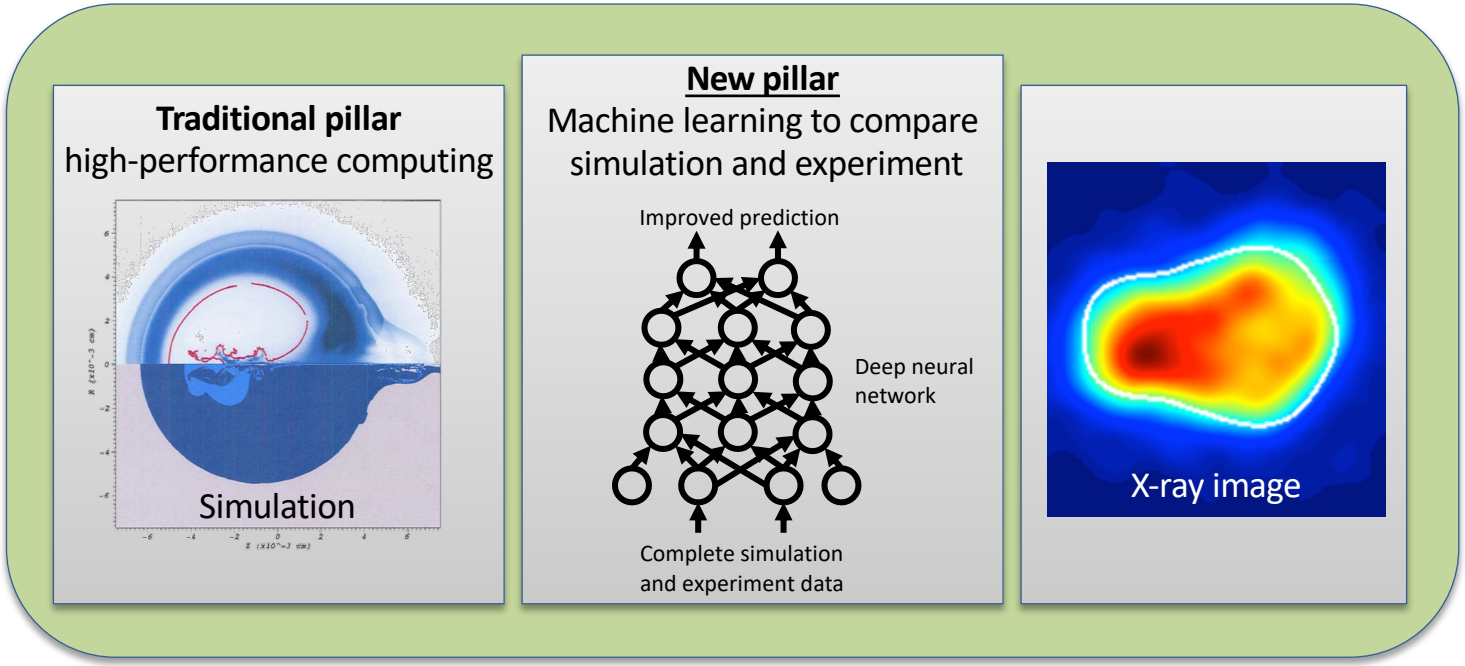
Demo

Talk Outline

- Overview of Flux
  - Examples of the APIs relevant to workflows
- Ongoing research within Flux
- Flux as an Open-Source Project

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

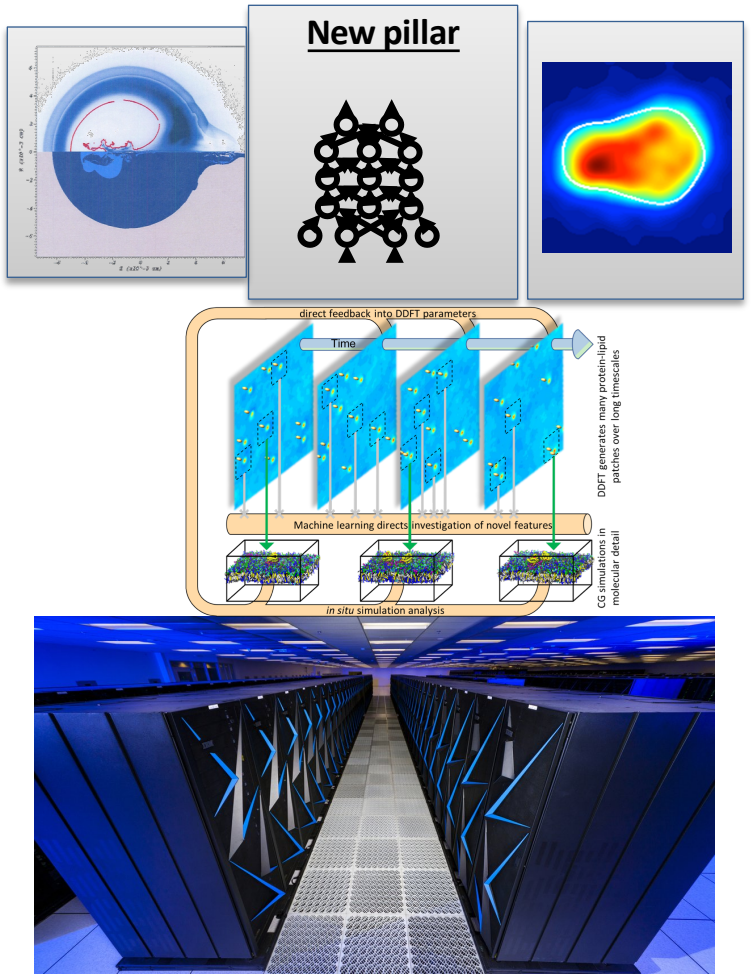




- MuMMI– co-schedule many elements and ML continuously schedules, de-schedules and executes MD jobs.
- In-situ analytics modules
- ~7,500 jobs simultaneously running

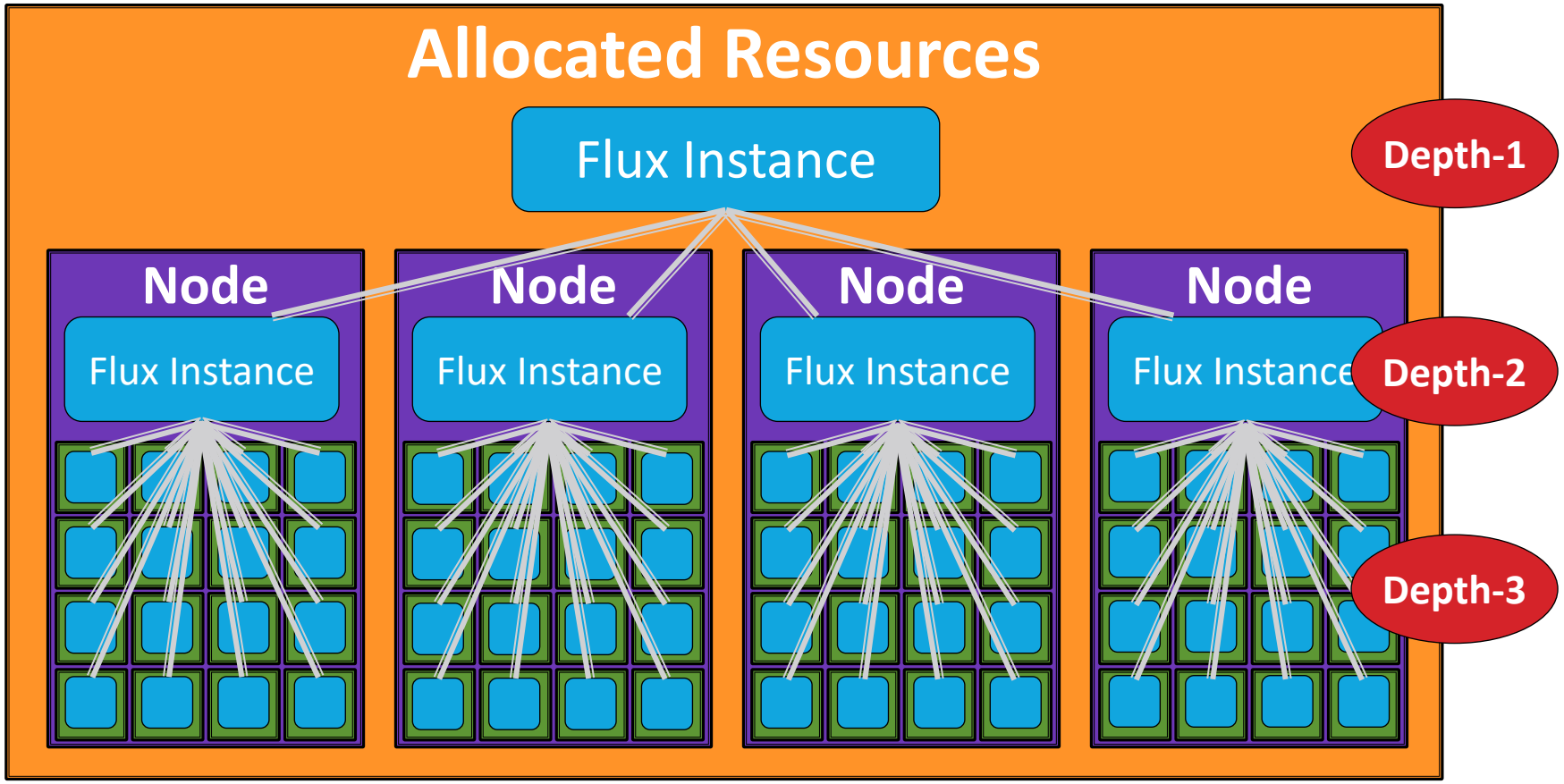
- Machine Learning Strategic Initiative (MLSI) – 1 billion short-running jobs!
- Similar needs for co-scheduling heterogenous components

Key challenges in emerging workflow scheduling include…



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

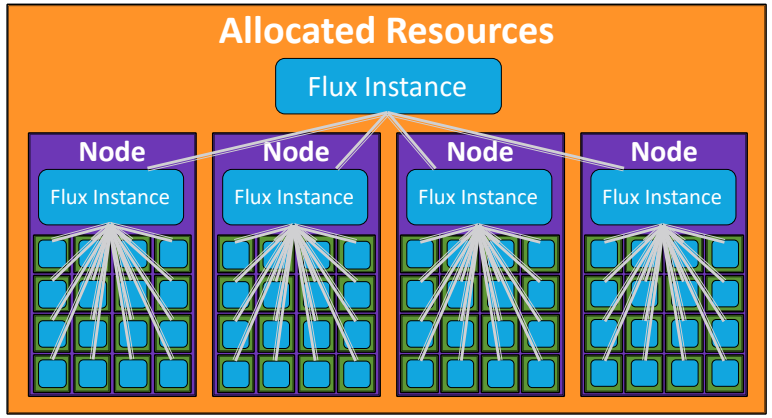
Flux provides a new scheduling model to meet these challenges.



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

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

Scheduling Model



Techniques

User-Level Scheduling

Scheduler Parallelism

Rich API set

Nested Launching

Challenges

Co-scheduling challenge

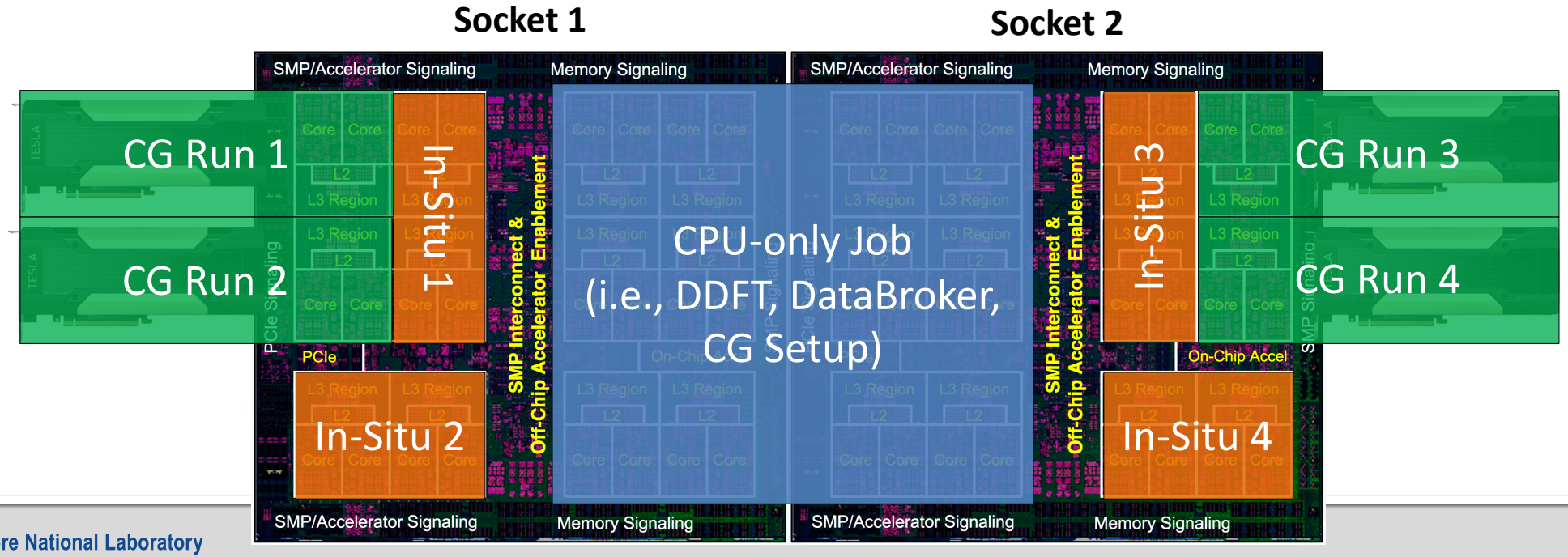
Job throughput challenge

Job coordination challenge

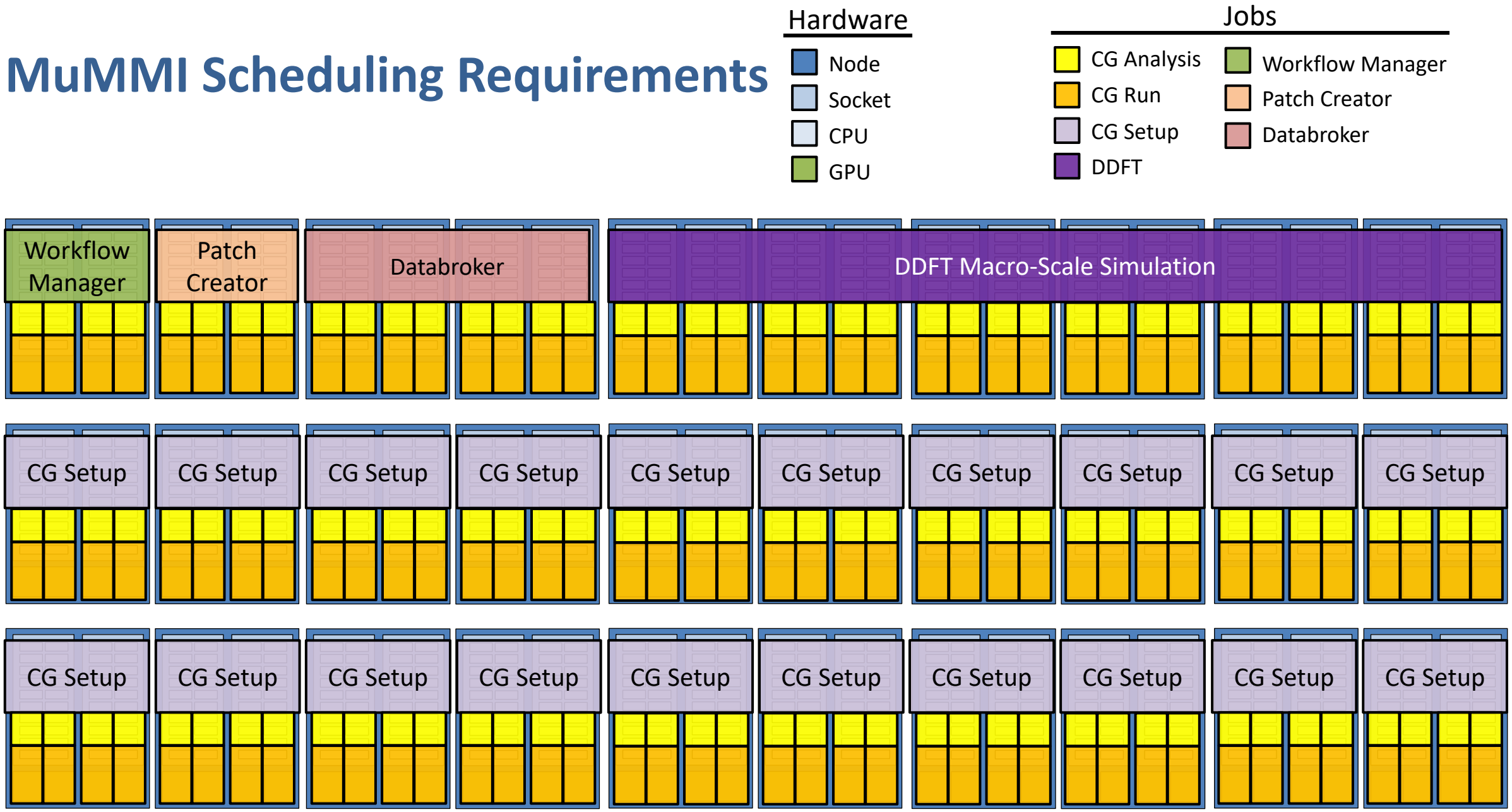
Portability challenge

User-level scheduling solves the co-scheduling challenge.

- Typical HPC schedulers provide batch jobs + job steps
  - Complex job-step launchers != full-featured scheduler
- Flux enables system- and user-level scheduling under a common infrastructure.
  - Gives users access to a full-featured scheduler within their allocation
  - Gives users the freedom to adapt their scheduler instance to their needs.

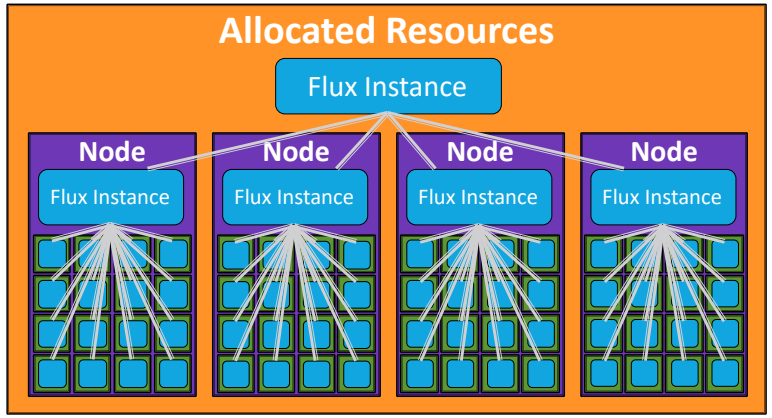


MuMMI Scheduling Requirements



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

Scheduling Model



Techniques

User-Level Scheduling

Scheduler Parallelism

Rich API set

Nested Launching

Challenges

Co-scheduling challenge

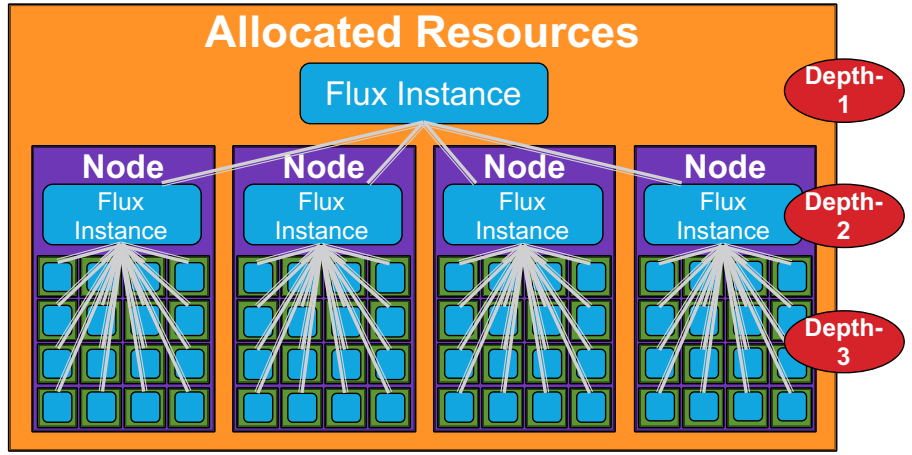
Job throughput challenge

Job coordination challenge

Portability challenge

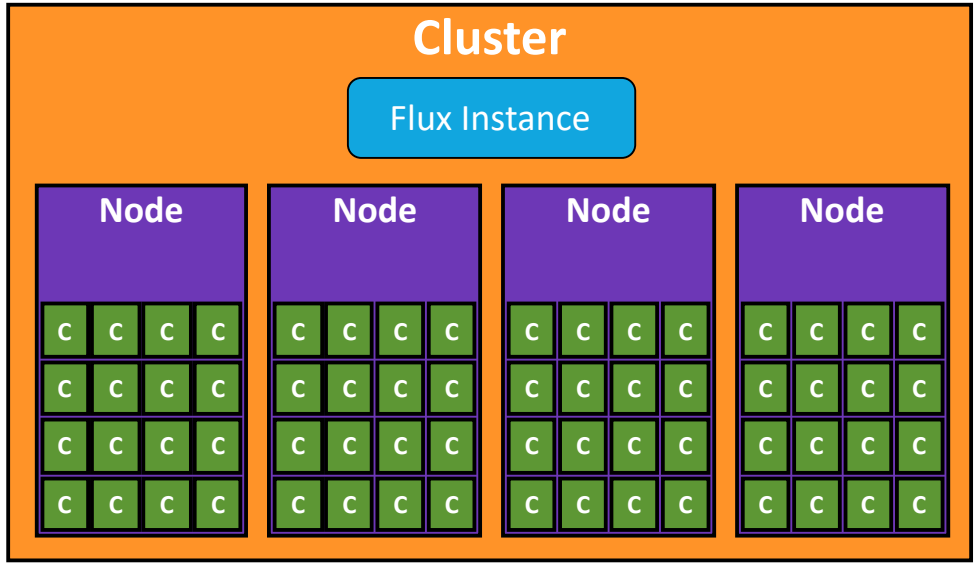
Scheduler parallelism solves the throughput challenge.

- 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.



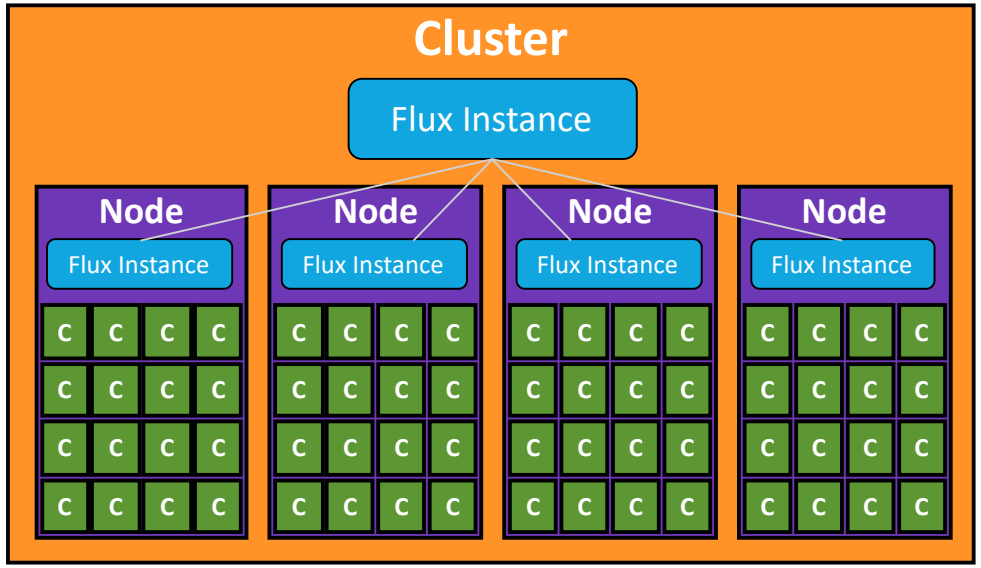
Flux API Example: Running Millions of Jobs

- Single Flux Instance
  - `flux start my_workflow.py`



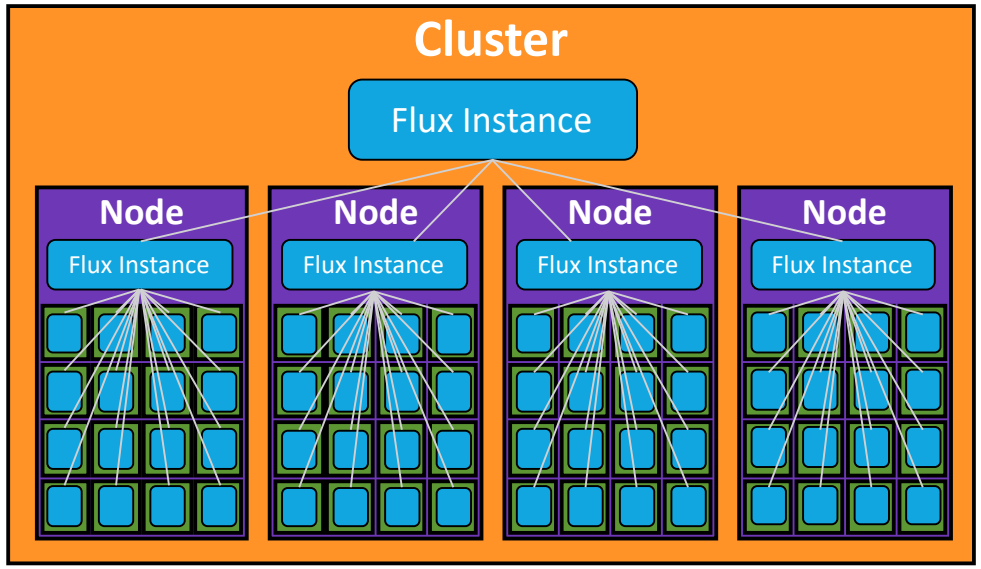
Flux API Example: Running Millions of Jobs

- 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.py $x
done
```



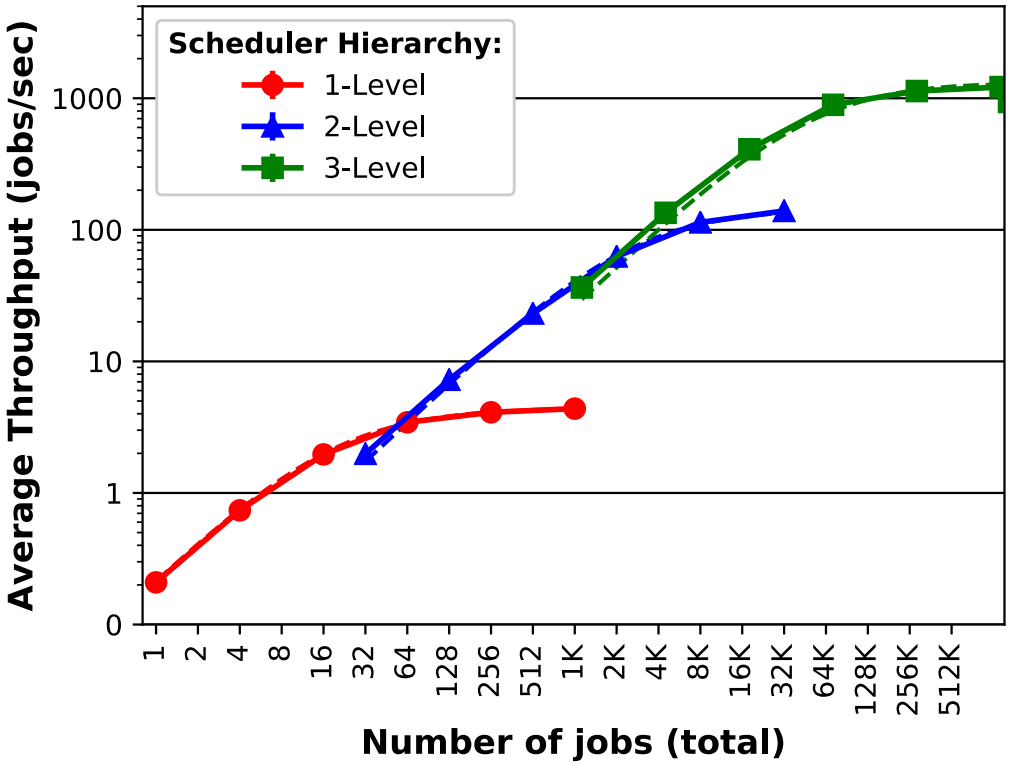
Flux API Example: Running Millions of Jobs

- 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.py $x
  done
```
- Flux Hierarchy
  -
```
flux-tree –T ${num_nodes}  \
        -J $num_jobs -- flux submit my_job.py
```
  -
```
flux-tree \
        –T ${num_nodes}x${cores_per_node} \
        -J $num_jobs – flux submit my_job.py
```



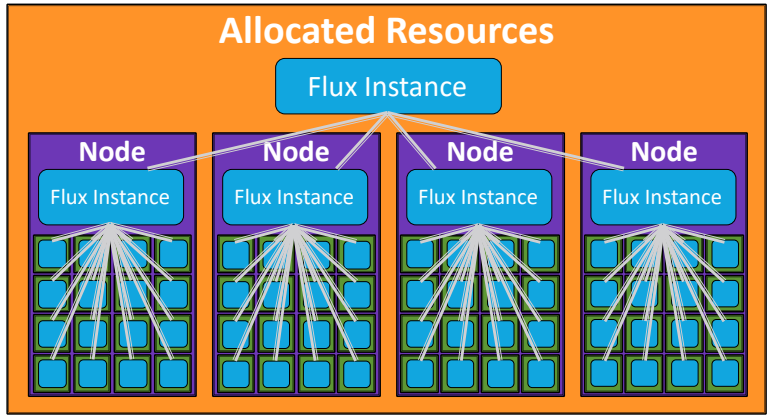
Flux API Example: Running Millions of Jobs

- 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.py $x
  done
```
- Flux Hierarchy
  -
```
flux-tree –T ${num_nodes}  \
        -J $num_jobs -- flux submit my_job.py
```
  -
```
flux-tree \
        –T ${num_nodes}x${cores_per_node} \
        -J $num_jobs – flux submit my_job.py
```



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

Scheduling Model



Techniques

User-Level Scheduling

Scheduler Parallelism

Rich API set

Nested Launching

Challenges

Co-scheduling challenge

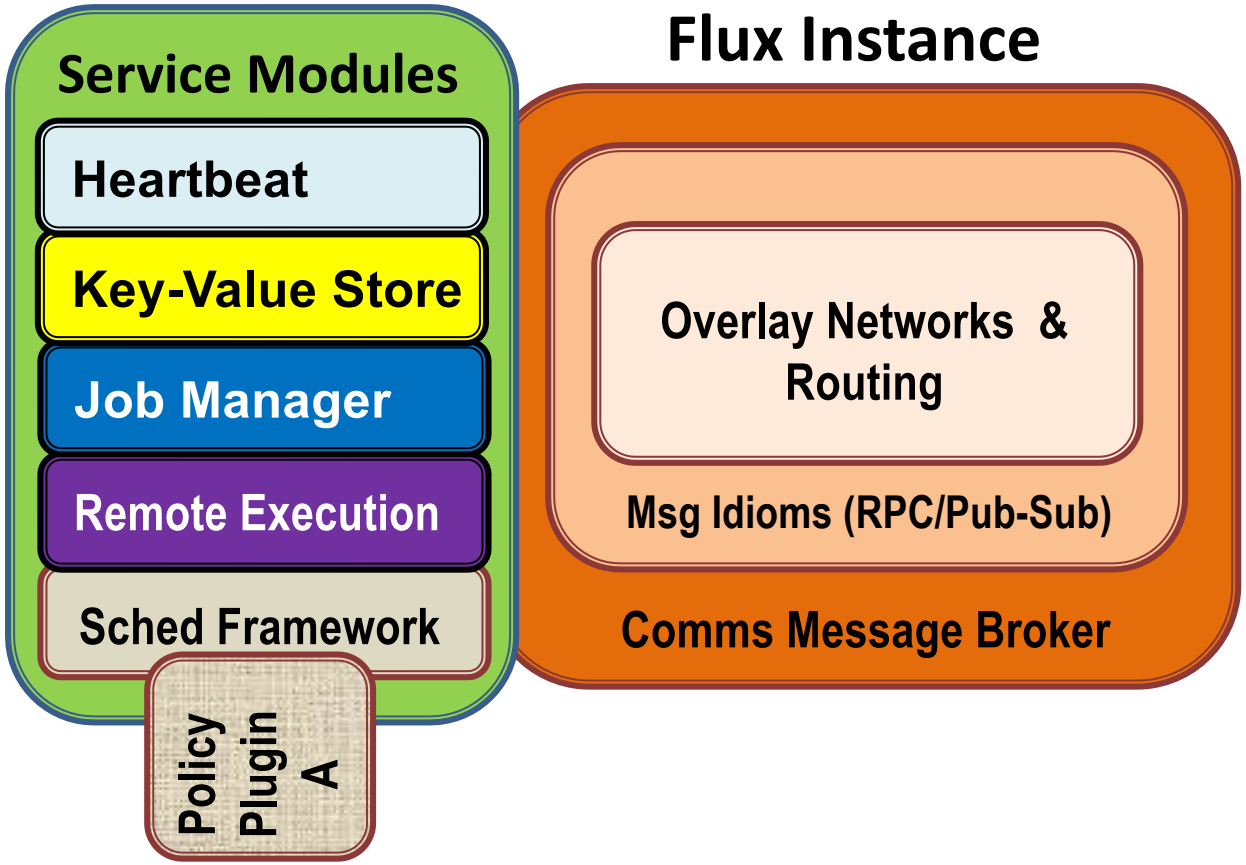
Job throughput challenge

Job coordination challenge

Portability challenge

A rich API set enables easy job coordination and communication.

- Co-scheduled jobs often require close coordination and communication with each other and the scheduler
  - Traditional CLI-based approach is too slow and cumbersome.
  - Ad hoc approaches (e.g., many empty files) can lead to many side-effects.
- High-level services
  - Communication overlay: pub/sub, RPC
  - Job status monitoring API
  - Key-value store (KVS) API

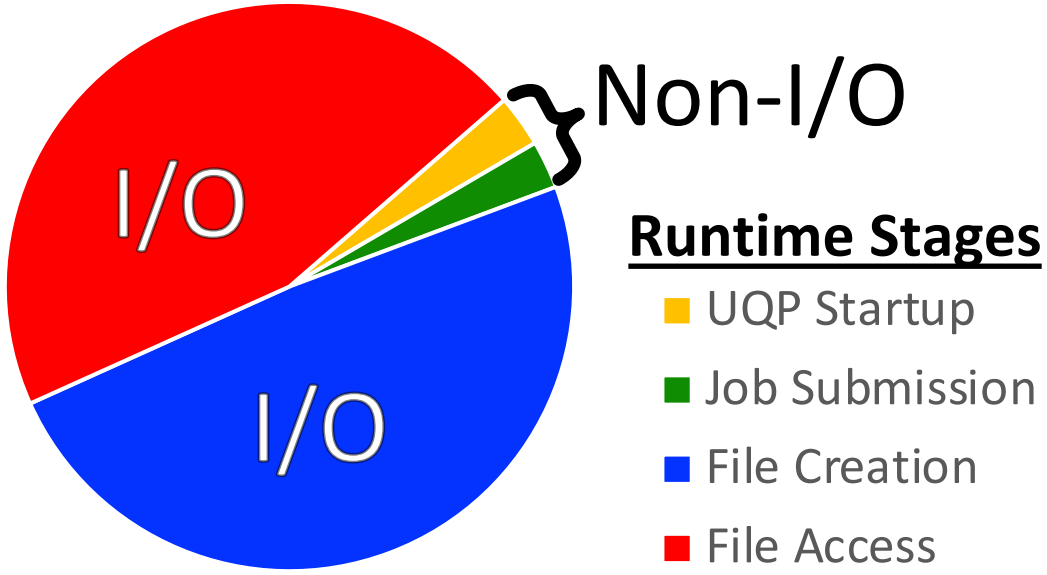


Lawrence Livermore National Laboratory
LLNL-PRES-818397
National Nuclear Security Administration

Flux API Example: 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
```



Flux API Example: Tracking Job Status

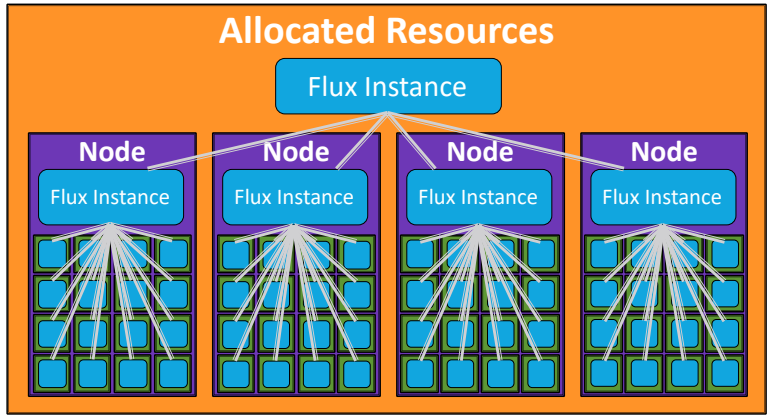
- CLI: slow, non-programmatic, inconvenient to parse
  - `watch squeue`
  - `watch flux job list`
- Tracking via the filesystem
  - `date > $JOBID.start; srun myApp; date > $JOBID.stop`
- Job tracking via Flux:

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

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

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

Scheduling Model



Techniques

User-Level Scheduling

Scheduler Parallelism

Rich API set

Nested Launching

Challenges

Co-scheduling challenge

Job throughput challenge

Job coordination challenge

Portability challenge

Nested launch facilitates high portability.

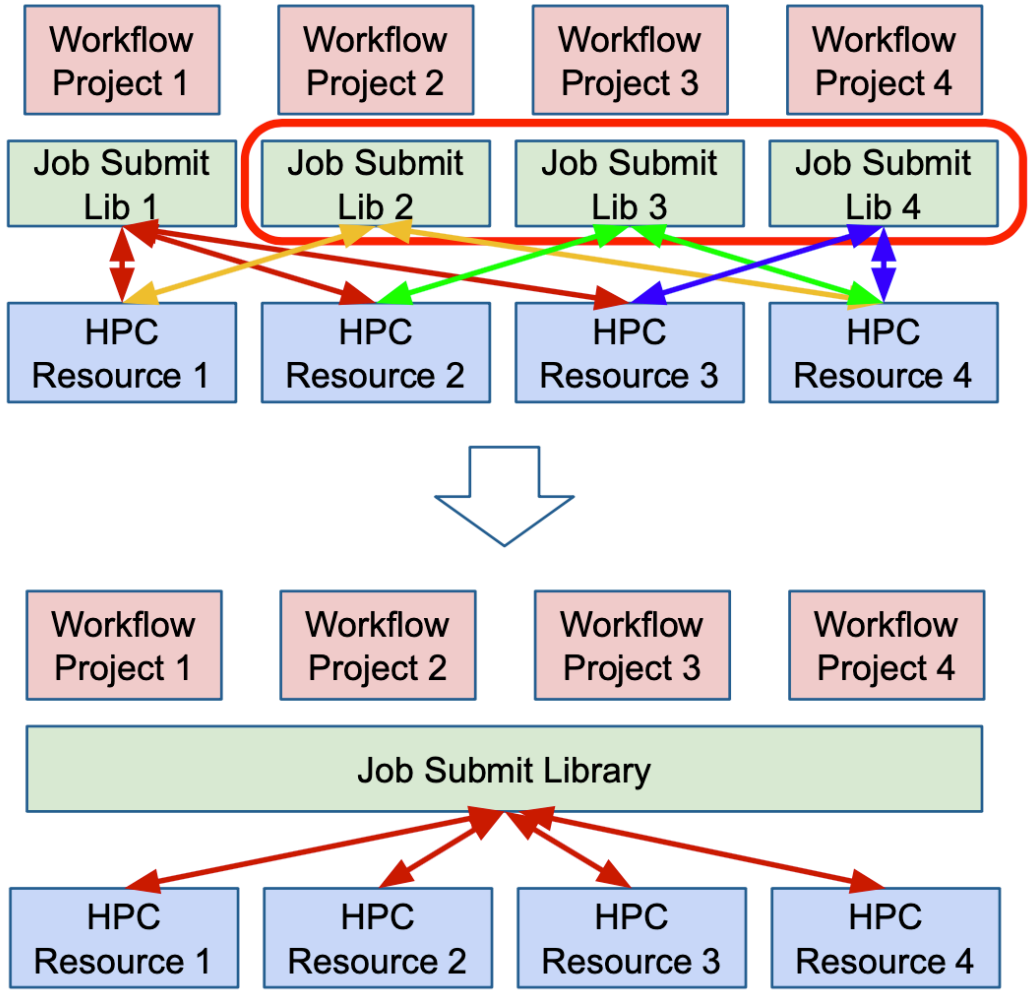
- Flux can run anywhere that MPI can run, (via PMI – Process Management Interface)
  - Inside a resource allocation from: itself (hierarchical Flux), Slurm, Moab, PBS, LSF, etc
  - `flux start` **OR** `srun flux start`
- Flux can run anywhere that supports TCP and you have the IP addresses
  - FLUX_CONF_DIR=/etc/flux `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" } ]
```

- Already installed on many DOE systems
  - `spack install flux-sched` for everywhere else

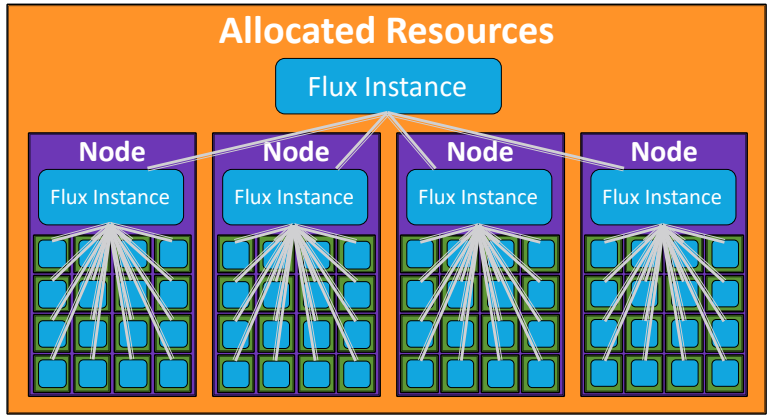
ExaWorks

- Our survey of HPC applications shows
  - a plethora of job submission libraries, interfaces, …
  - overheads associated with supporting many schedulers and systems
- J/PSI is a Job management API for managing jobs on HPC systems
  - Lightweight, user space deployment
  - Minimally prescriptive/interface simplicity
  - Language independent
  - Async and bulk operations where possible
- Call to help with the J/PSI specification, prototype Python binding, and integration with community workflow systems
  - http://exaworks.org/job-api-spec/specification.html
  - https://github.com/ExaWorks/jpsi-python



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

Scheduling Model



Techniques

User-Level Scheduling

Scheduler Parallelism

Rich API set

Nested Launching

Challenges

Co-scheduling challenge

Job throughput challenge

Job coordination challenge

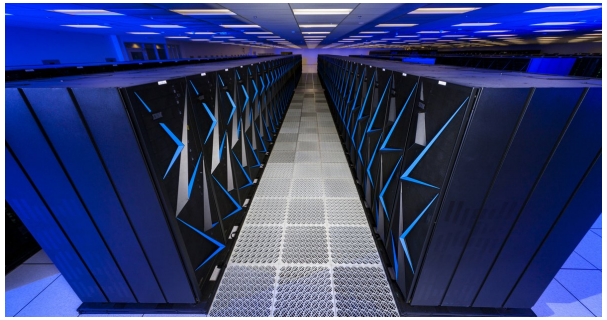
Portability challenge

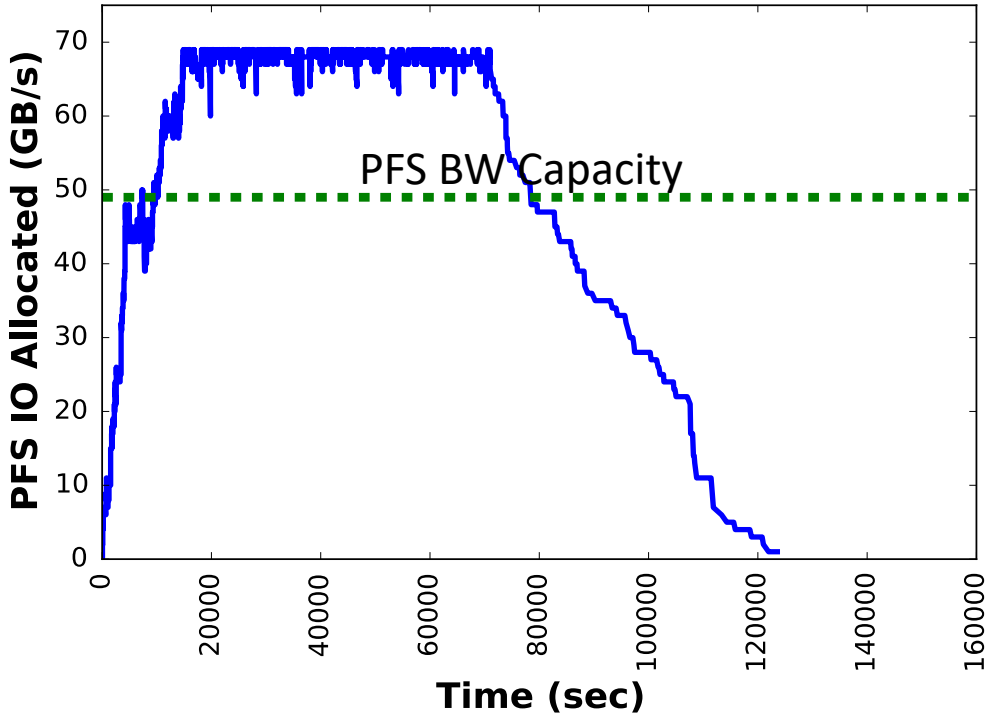
Talk Outline

- Overview of Flux
  - Examples of the APIs relevant to workflows
- Ongoing research within Flux
- Flux as an Open-Source Project

The changes in resource types are equally challenging.

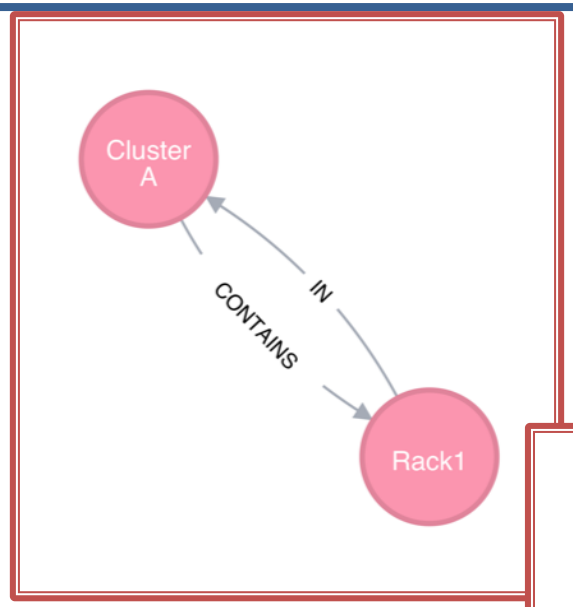
- Problems are not just confined to the workload/workflow challenge.
- Resource types and their relationships are also becoming increasingly complex.
- Much beyond compute nodes and cores…
  - GPGPUs
  - Burst buffers
  - I/O and network bandwidth
  - Network locality
  - Power



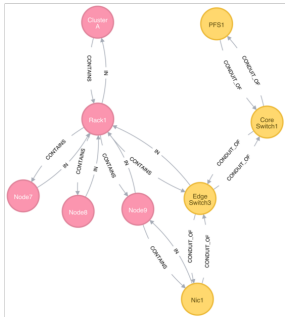


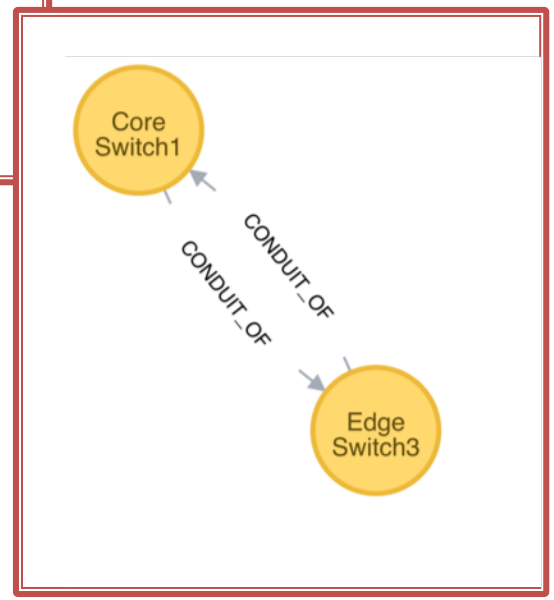
Flux uses a graph-based resource data model to represent schedulable resources and their relationships.

- 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

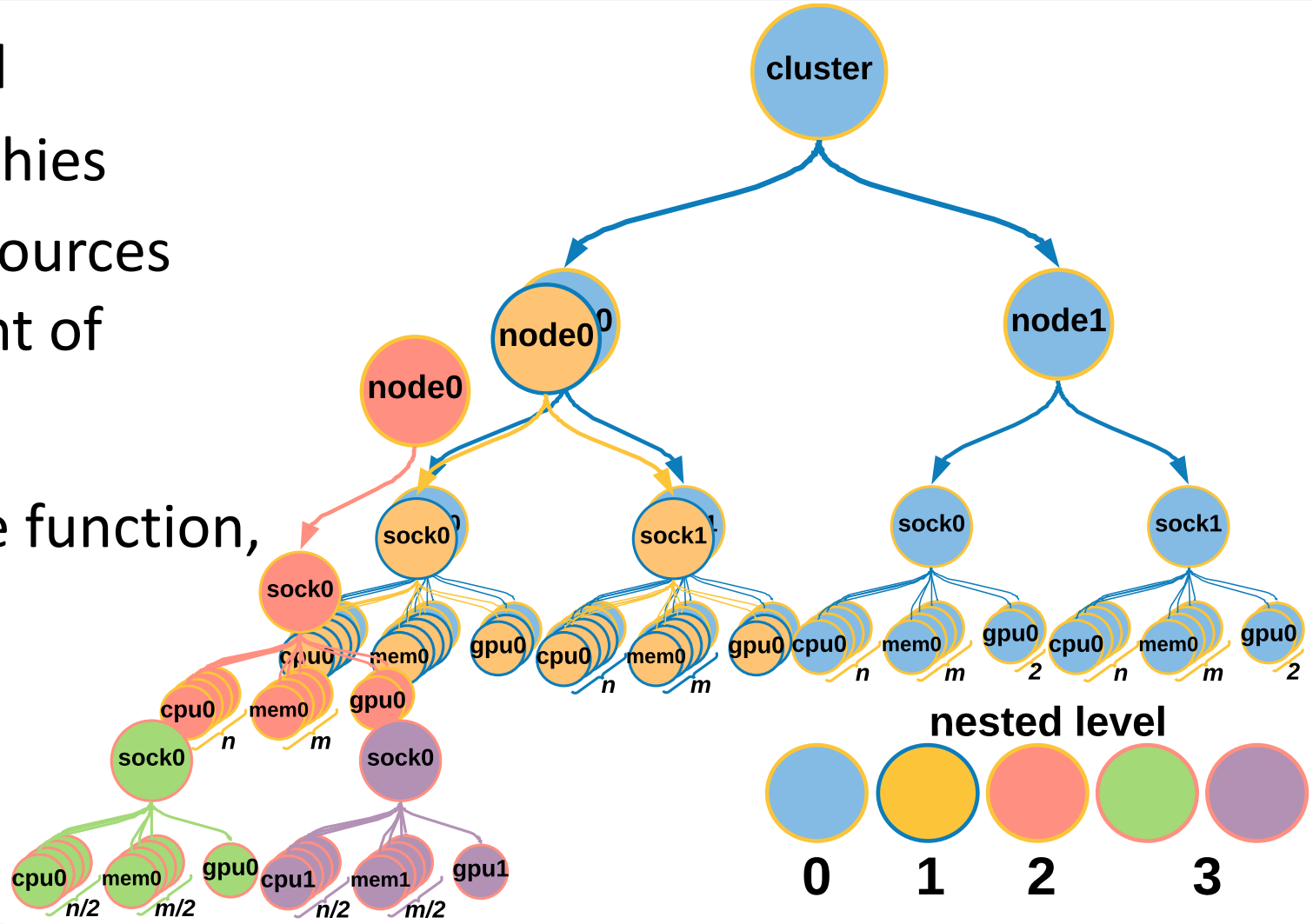




Network connectivity subsystem

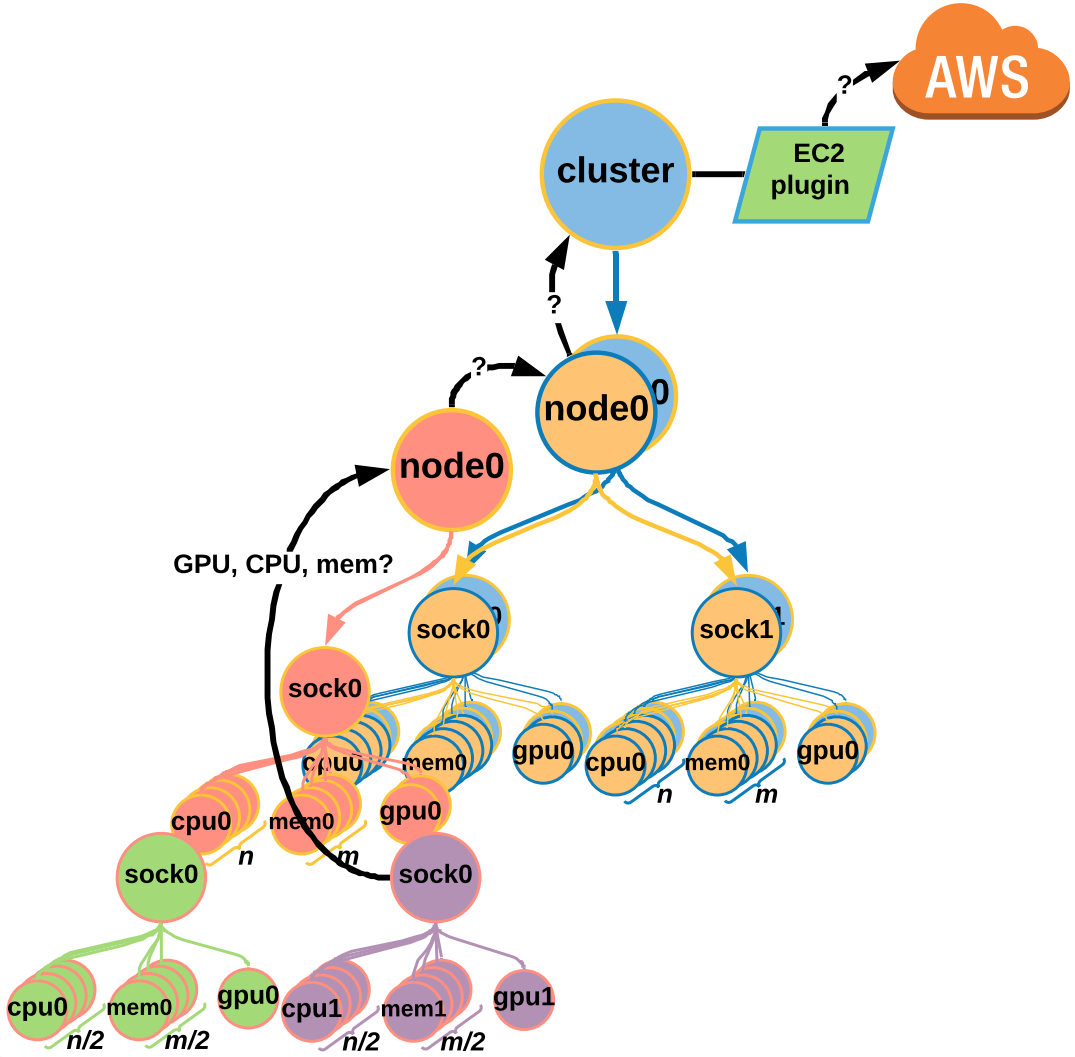
Enabling elasticity in Flux-framework via nested schedulers and directed graphs

- Fluxion directed graph model
  - —naturally expresses hierarchies
  - —child instance requests resources from parent, notifies parent of relinquished resources
- Graph mutation is a recursive function, pairwise operation



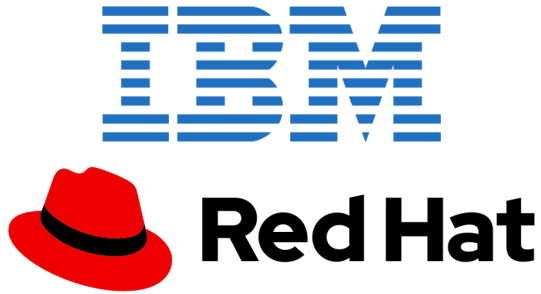
Generalized Multi-Level Scheduling works for bursting: cloud is just another level

- traditional schedulers not designed for resource dynamism
  - often hard to add new relationships without scheduler modification/restart
- elastic resource addition/removal are graph operations



Building industry collaborations for HPC+cloud

- Explore best ways to express converged resources
- Enable Fluxion to schedule pod binding in OpenShift
- Develop tenancy model for HPC+cloud
- Flux in the cloud, bursting to

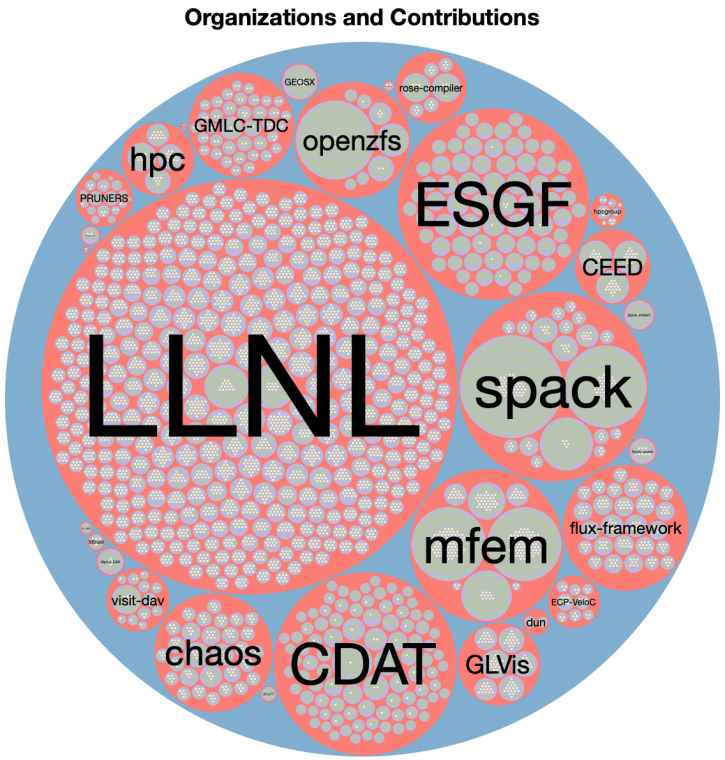


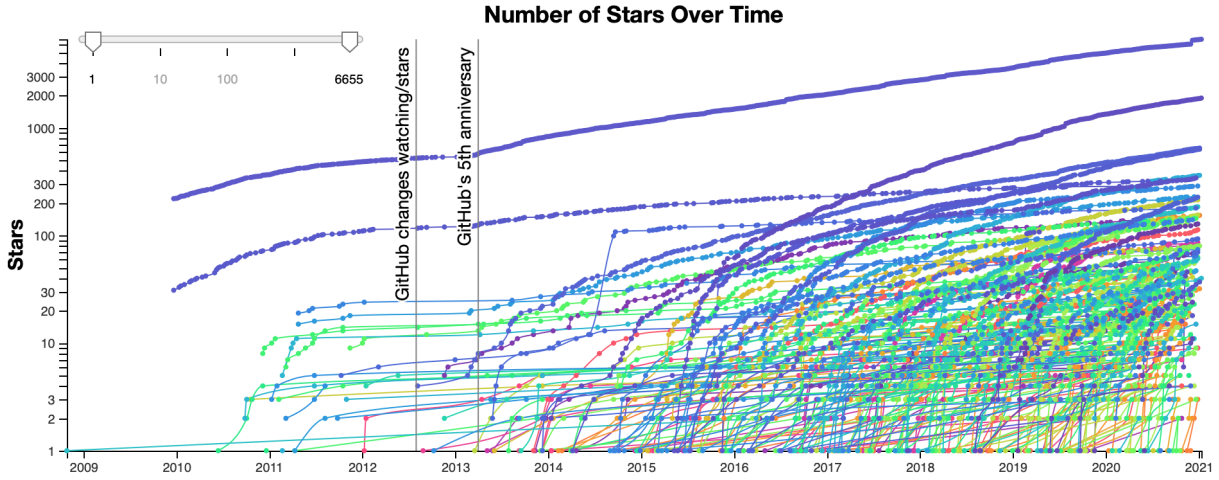
Talk Outline

- Overview of Flux
  - Examples of the APIs relevant to workflows
- Ongoing research within Flux
- Flux as an Open-Source Project

Open-Source at LLNL

- For decades, LLNL has made software developed for programmatic work publicly available as open source.
- US Federal Source Code Policy mandates that at least 20% of code developed by or for government institutions be made open source
- Today, nearly 700 open-source software packages are developed at LLNL
  - ZFS, Spack, ZFP, mfem, raja, HPSS, Slurm, …





Sources: https://software.llnl.gov
https://computing.llnl.gov/sites/default/files/public/COMP_Poster_OSS.pdf

Flux's Use of the Collective Code Construction Contract (C4.1)

- Based on Zeromq's C4.1: https://rfc.zeromq.org/spec/22/
- Goal: provide a collaboration model that is scalable, open/diverse, and fast moving
- Formal Design:
  - Use git, host on a git "platform", use the platform issue tracker and code review/merge system
  - No one pushes to main fork, everything done in personal forks, all changes get reviewed via PRs
  - Use share-alike license (e.g., [L]GPLv3, MPLv2). No copyright assignment process.
  - Stable releases get their own repo
  - All public contracts SHOULD be documented in an RFC
- Informal Design:
  - Major modular components are encouraged to be developed in their own repository

Use git, host on a git “platform”, use the platform issue tracker and code review/merge system

- Pros:
  - All of our work is out in the open and easily accessible (“just check my GitHub profile”)
  - We benefit from GitHub’s network effects (e.g, easy to add contributors) and tooling (e.g., GH Actions, bots, static analysis, RTD, etc)
- Cons:
  - Using GitHub, especially their integrated CI/Actions, provides a fair amount of lock-in
- Other Lessons Learned:
  - GitHub Discussions are a much better place for mailing list style discussions than GH Issues
  - For any SAAS (e.g. Travis CI, LGTM, Mergify) that you use, be ready to either pay or switch if the company behind the tool turns off the faucet to open-source projects

No one pushes to main fork, everything done in personal forks, all changes get reviewed via PRs

- Pros:
  - No patch or change is special. No contributor is “blessed”. Every PR being reviewed encourages consistent quality
  - Reliance on forks means main repos stays clean and free of stale topic branches
- Cons:
  - Discovery of ongoing work can be difficult – distributed across contributors’ forks
  - New contributors are sometimes unfamiliar with using multiple forks/remotes
- Other Lessons Learned:
  - Use WIP PRs to make ongoing work more visible
  - Whatever Git workflow your project uses, document it well

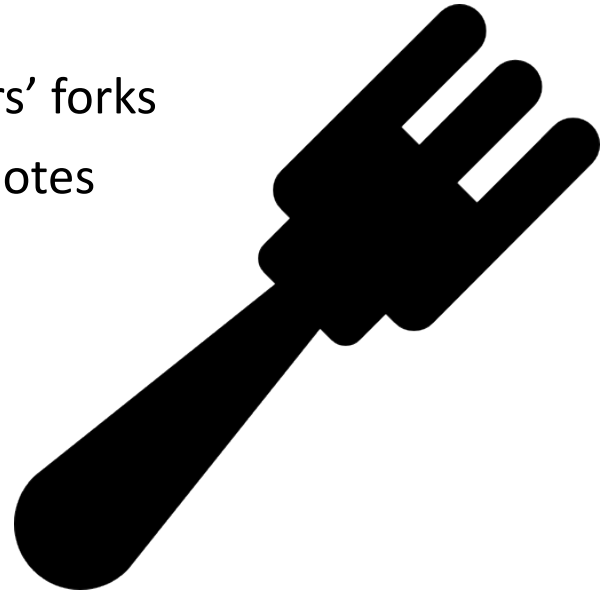


image: Flaticon.com

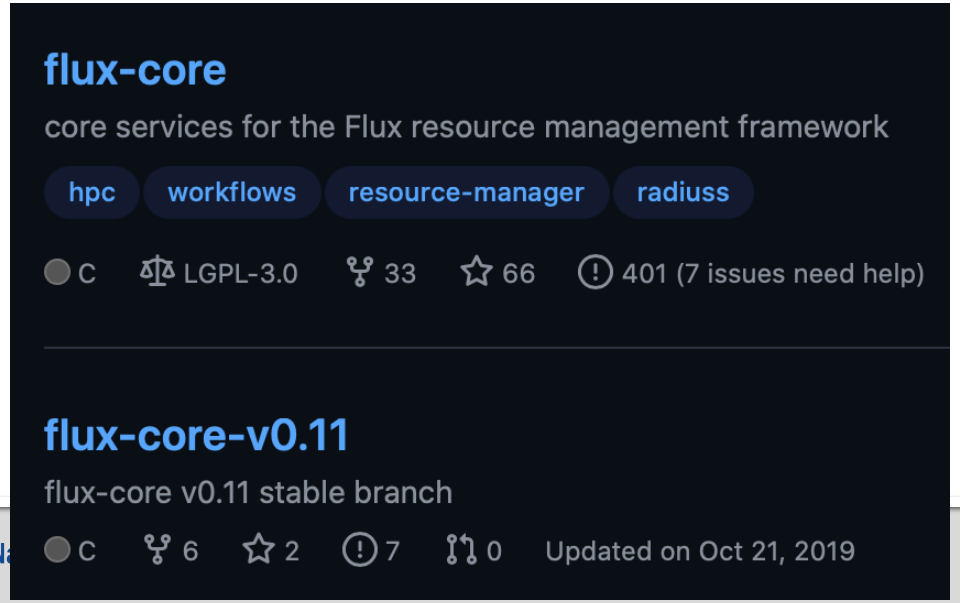
Use share-alike license (e.g., [L]GPLv3, MPLv2). No copyright assignment process.

- Pros:
  - Share-alike ensures that modifications to the project remain open-source
  - No copyright assignment/contributor agreement makes life easier for new contributors
    - C4 also mentions that it prevents hostile takeovers (no direct experience with this)
- Cons:
  - May prevent private companies from incorporating your software into their products [1]
    - Note: it shouldn't prevent them from contributing directly though
- Other Lessons Learned:
  - Mixing GPL and LGPL in the same repo is difficult. GPL dependencies can easily slip into LGPL libraries. We went fully LGPL at the repo level to avoid this issue.

[1] https://opensource.google/docs/thirdparty/licenses/#LinkingRequirements

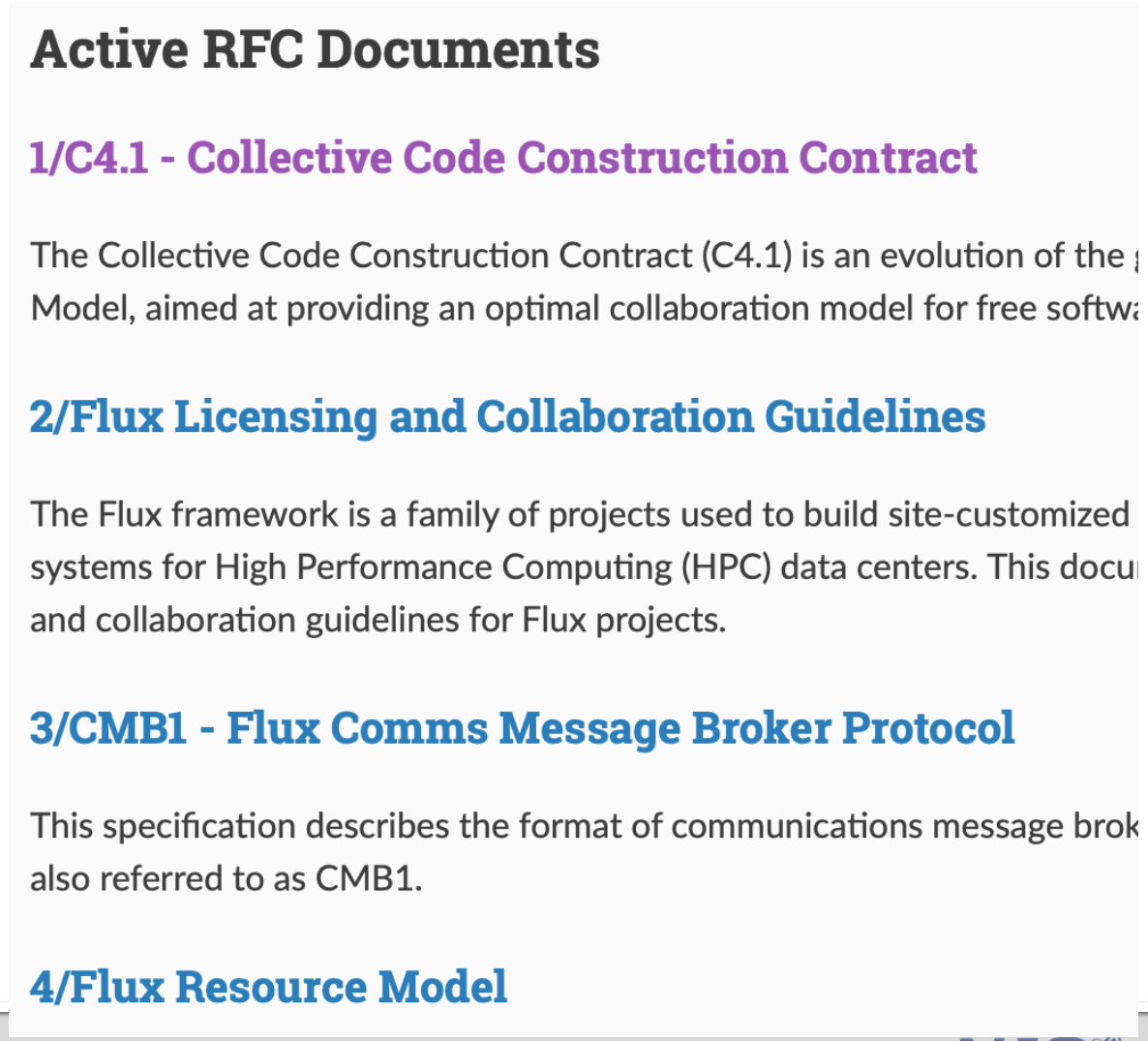
Stable releases get their own repo

- Pros:
  - Releases get their own separate issue tracker and area of backport Pull Requests
- Cons:
  - Higher overhead for creating a X.1 minor release or X.Y.1 patch release



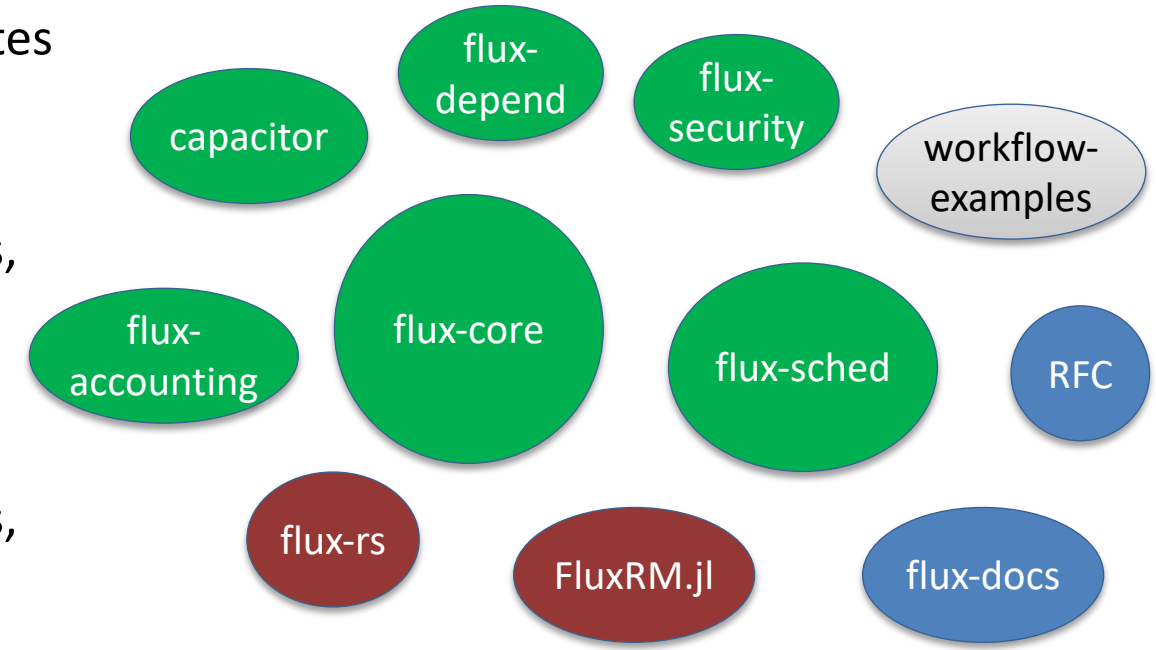
All public contracts SHOULD be documented in an RFC

- Pros:
  - Enables easy modularity due to rigorous interface/contract documentation
  - Encourages well thought-out design before implementation



Major modular components are encouraged to be developed in their own repository

- Pros:
  - Modular components can move at independent rates
    - Experimental components can “move fast and break things” while stable components move more slowly
  - Each component’s repo can have its own processes, languages, dependencies, and license
- Cons:
  - Each component’s repo can have its own processes, languages, dependencies, and license
  - Higher overhead to clone, build, and install the full Flux suite



Join the Flux Community!

- Flux welcomes all contributors for bug fixes, code improvements, new features, simplifications, documentation, and more.
- Contributing Guide: https://github.com/flux-framework/flux-core/discussions
- GitHub Discussions available for questions, ideas, and general discussion: https://github.com/flux-framework/flux-core/discussions

github.com/flux-framework

@flux-framework