



## Welcome to the RADIUSS AWS Tutorial Series!

Go to:

<https://software.llnl.gov/radiuss/event/2023/07/11/radiuss-on-aws/>

to learn more about our other tutorials and documentation!

Date	Time (Pacific)	Project
August 3, 2023	9:00a.m.–11:00a.m.	 Build, link, and test large-scale applications with <b>BLT</b>
August 8–9 2023	8:00a.m.–11:30a.m. both days	 Learn to install your software quickly with <b>Spack</b>
August 10, 2023	9:00a.m.–11:00a.m.	 Use <b>MFEM</b> for scalable finite element discretization application development
August 14, 2023	9:00a.m.–12:00p.m.	 Integrate performance profiling capabilities into your applications with <b>Caliper</b>
		 Analyze hierarchical performance data with <b>Hatchet</b>
		 Optimize application performance on supercomputers with <b>Thicket</b>
August 17, 2023	9:00a.m.–11:00a.m.	 Use <b>RAJA</b> to run and port codes quickly across NVIDIA, AMD, and Intel GPUs
		 Discover, provision, and manage HPC memory with <b>Umpire</b>
August 22, 2023	9:00a.m.–11:00a.m.	 Visualize and analyze your simulations in situ with <b>Ascent</b>
August 24, 2023	9:00a.m.–11:00a.m.	 Leverage robust, flexible software components for scientific applications with <b>Axiom</b>
August 29, 2023	9:00a.m.–11:00a.m.	 Analyze runs of your code with <b>WEAVE</b> <small>Open Source Lawrence Livermore National Laboratory</small>
August 31, 2023	9:00a.m.–11:00a.m.	 Learn to run thousands of jobs in a workflow with <b>Flux</b>

# The Flux Framework Tutorial

RADIUSS AWS Tutorial Series  
August 31, 2023

Al Chu, James Corbett, Ryan Day, Jim Garlick,  
Giorgis Georgakoudis, **Mark Grondona**,  
**Dan Milroy**, Zeke Morton, Chris Moussa,  
Tapasya Patki, Barry Rountree, Abhik Sarkar,  
Tom Scogland, **Vanessa Sochat**, Becky  
Springmeyer, **Jae-Seung Yeom**



# Pre-exascale scientific workflows strain the capabilities of traditional HPC resource managers and schedulers.

Co-scheduling:

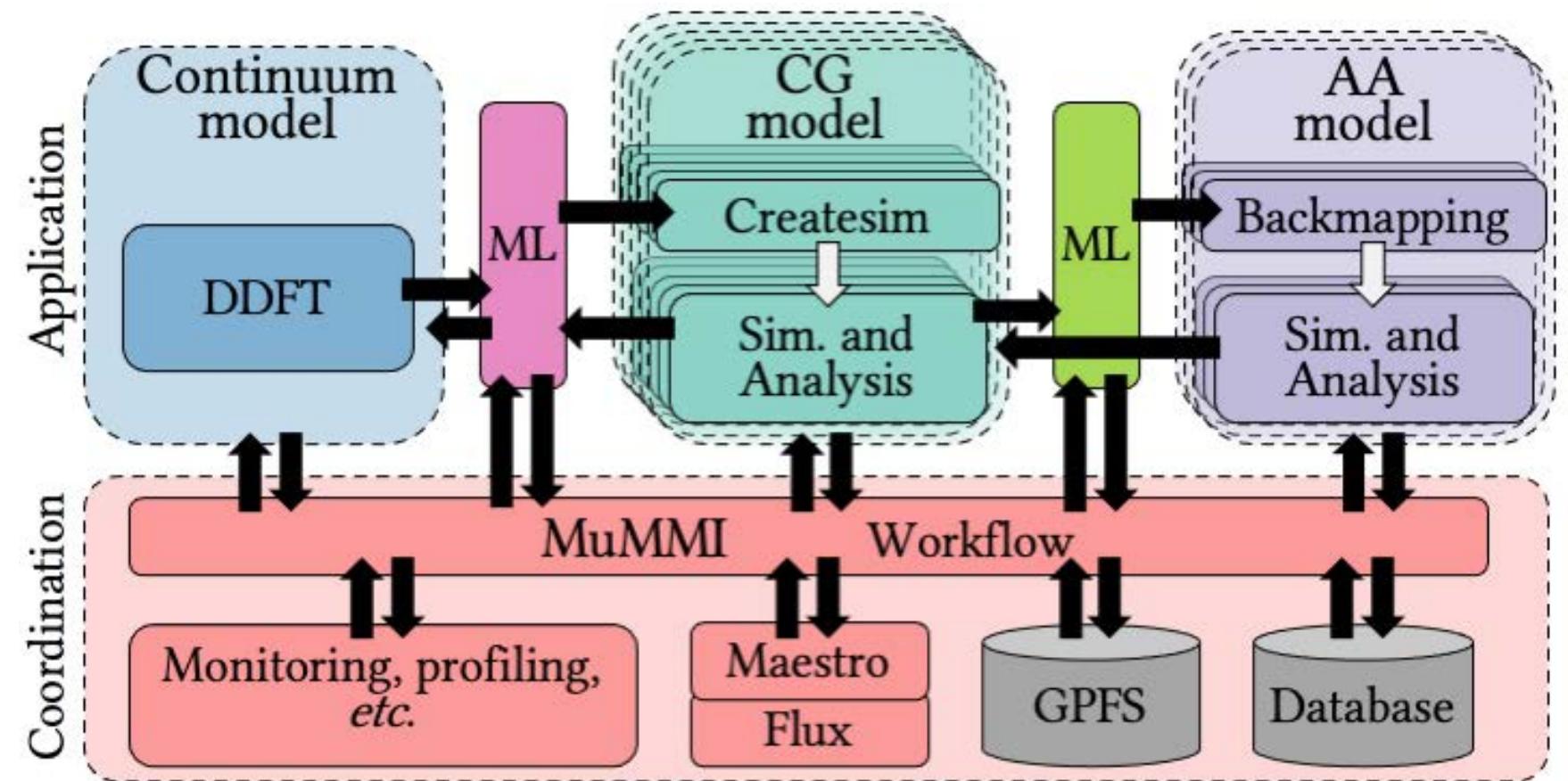
CG, analysis bound to cores  
nearest PCIe buses

Job comms/coordination:

36,000 concurrent tasks;  
176,000 cores, 16,000 GPUs

Portability:

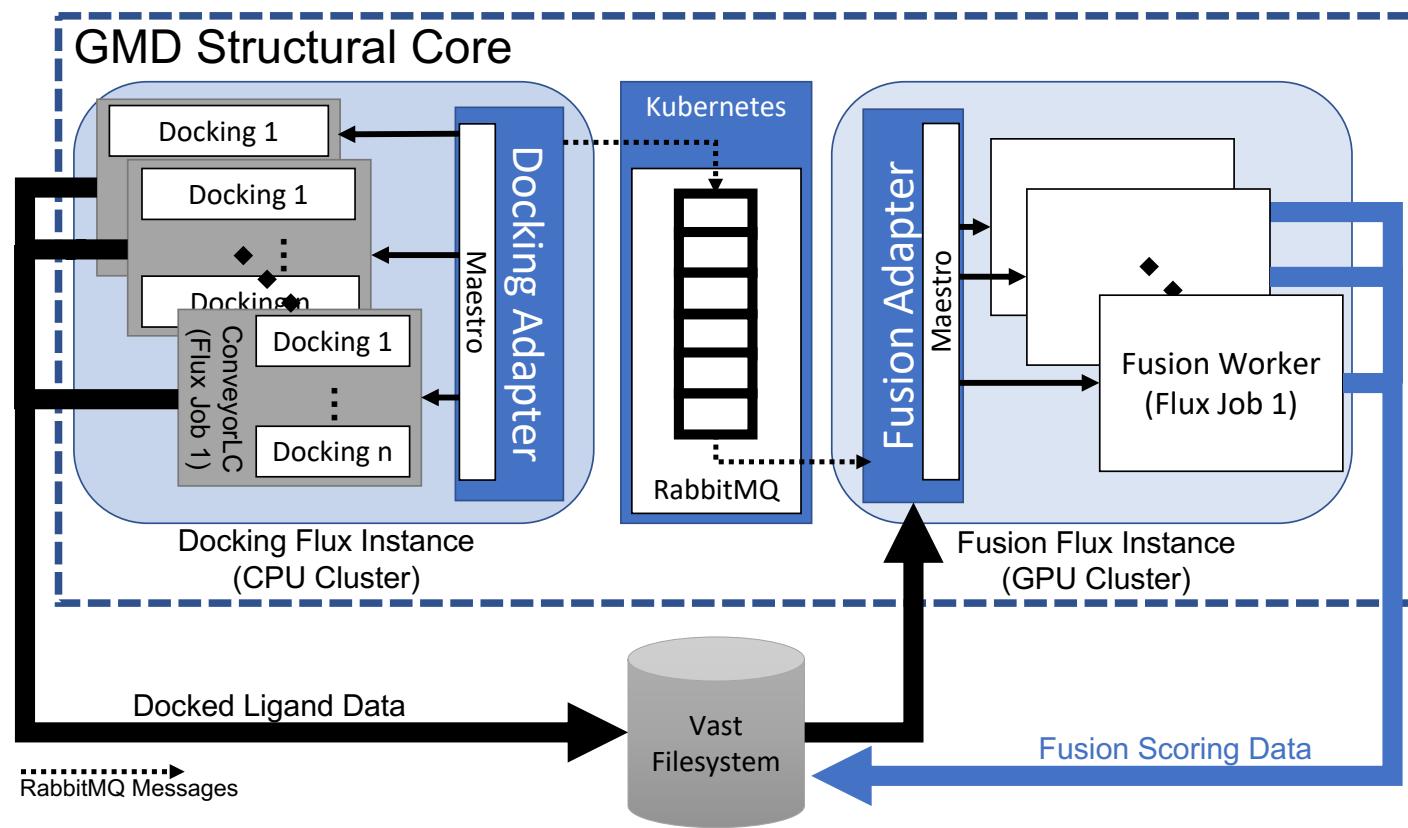
adapt tasks to different  
Schedulers/managers



MuMMI: SC'19 best paper, SC'21 paper

MPI-based simulation with in-situ analysis plus AI/ML

# Next-generation, cross-cluster scientific workflows are demanding portability and cloud integration.



AHA MoleS: eScience'22 best paper

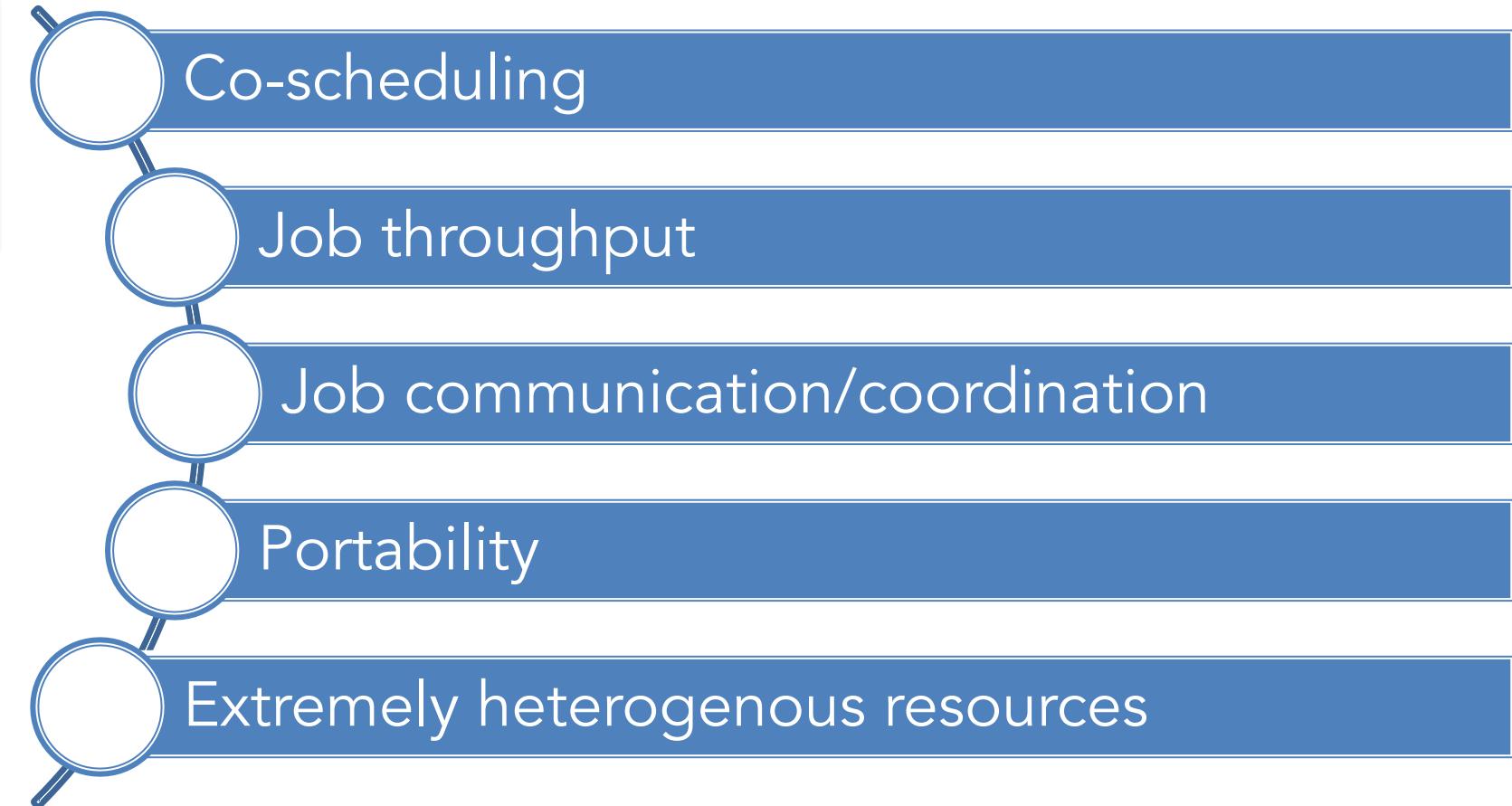
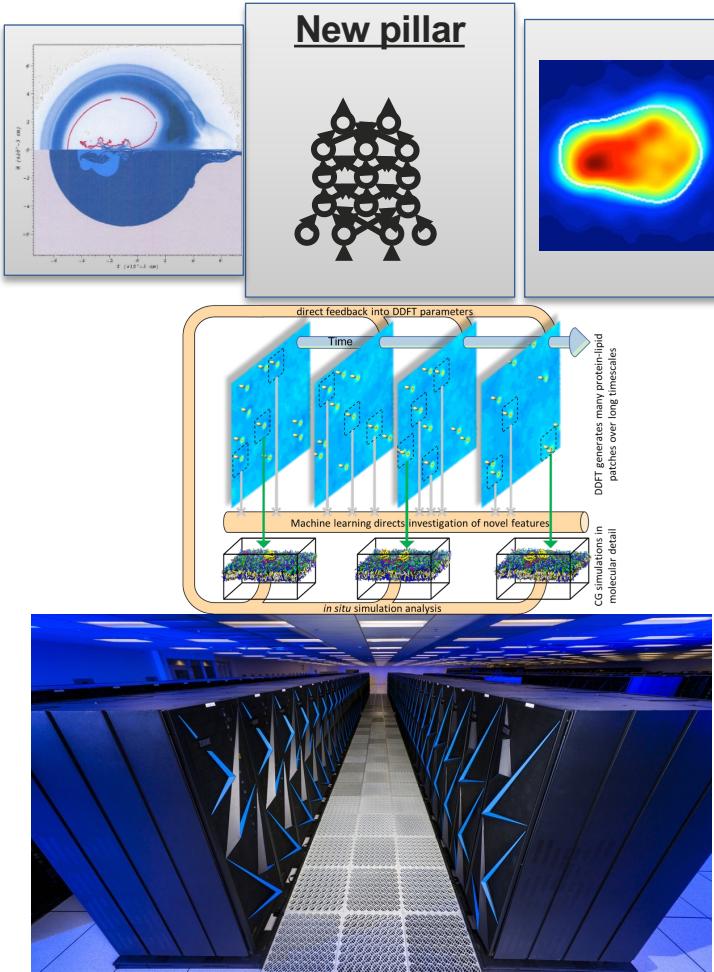
Complex workflows integrating cloud technologies at LLNL and beyond

- Scalable message broker couples MPI-based tasks (AHA MoleS)
- HPC simulation with AI/ML surrogates, orchestrated data (AMS)
- Many other examples

2020 lab survey found that 73% of LLNL workflows interested in cloud integration

MPI-based simulation with analysis, AI/ML, containerized components

# Trends towards complex workflows, extreme resource heterogeneity, and converged computing render traditional workload managers increasingly ineffective.



# Flux solves key technical problems that emerge from these trends.

flux-framework / flux-accounting

flux-framework / flux-security



garli 202 commits

flux-framework / flux-sched



dun 5 commits

garlick 1,188 commits 571,905 ++ 633,551 --

#1 dongahn 657 commits 540,576 ++ 24,442 --

#2

flux-framework / flux-core



grondo 471 commits

garlick 4,977 commits 957,913 ++ 882 --

flux-framework / flux-k8s



chu11

1,678 commits 110,983 ++ 66,00 --

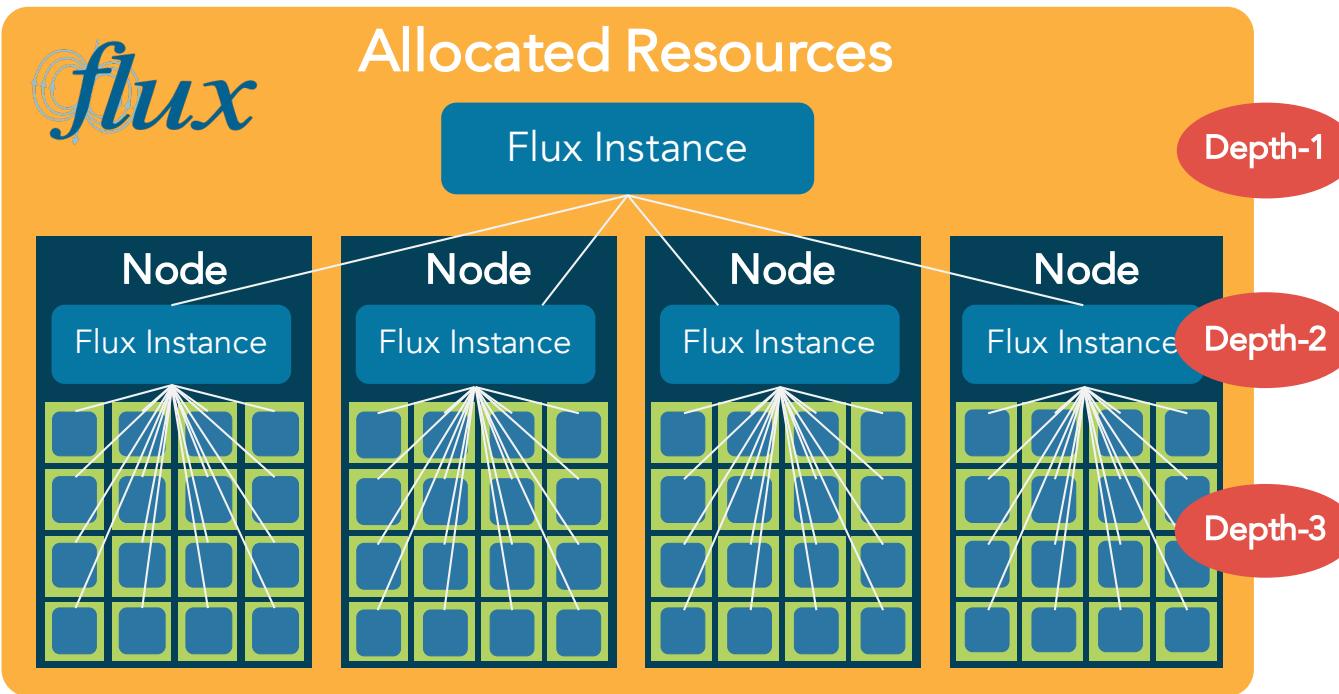
#1 milroy 20 commits 2,074 ++ 1,908 --

#2 cmisale 18 commits 4,125 ++ 2,297 --

- Open-source project in active development at flux-framework GitHub organization
  - Multiple projects: flux-core, -sched, -security, -accounting, -k8s etc.
  - Over 15 contributors including some principal engineers behind Slurm
- Single-user and System instance modes
  - Single-user mode in production for about 4 years
  - Multi-user mode debuting on LLNL Linux clusters
- Plan of record for **LLNL El Capitan** exascale system



# Flux hierarchical management and graph-based scheduling address exascale and converged computing challenges.



“Fractal scheduling” mitigates centralized scheduler bottleneck

- handles high throughput
- job steps needn’t hit central scheduler

Modular, hierarchical design

- Hierarchical resource management and scheduling (separate modules)
- Sub-manager with specialized scheduler
- Schedules cloud resources

Manages resources nearly anywhere

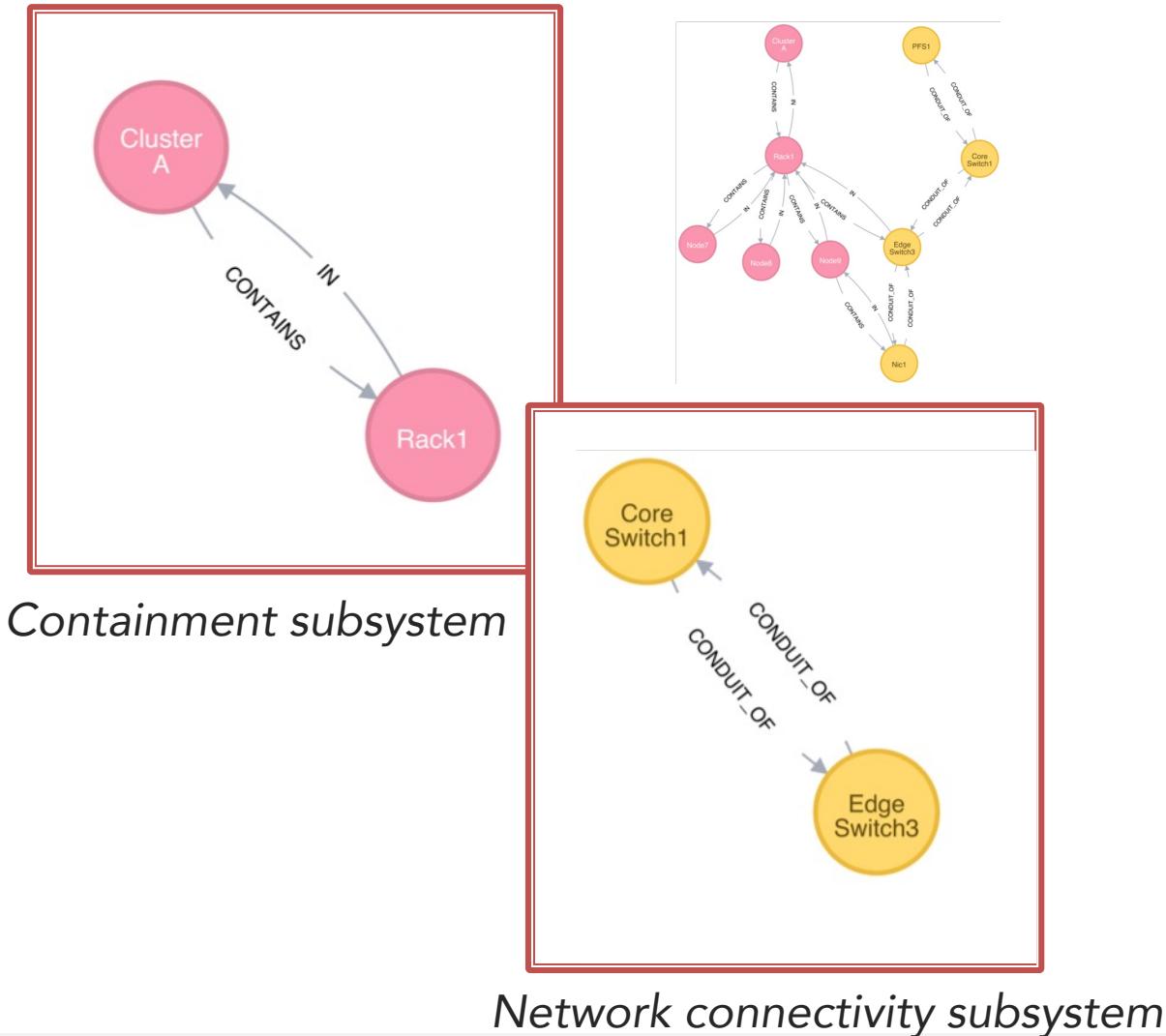
- Bare metal resources, virtual machines in the cloud, HPC resources in another workload manager, pods in Kubernetes
- Workflows only need to program to Flux
- Directed graph resource model expresses complex, dynamic resources

Rich, well-defined interfaces

- Facilitate communications and coordination among tasks within a workflow
- CLI, Python, C, C++, Rust, **Go (in progress)**, etc.

# Flux pioneers and uses graph-based scheduling to manage complex combinations of extremely heterogenous resources.

- Traditional resource data models are largely ineffective for resource heterogeneity
  - Designed with node-centric models when systems were simpler
- Elevate resource relationships (edges) to an equal footing with resources (vertices)
- Complex scheduling can be expressed without changing the scheduler code
- Rich and well-defined C and C++ API (Golang soon) for graph allocation



# Flux's graph-oriented jobspec allows for highly expressive resource requests.

- 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 is running on LLNL production clusters in preparation for the deployment of El Capitan.



Rolling out on production systems, addressing user groups one at a time. El Capitan will be a leap.

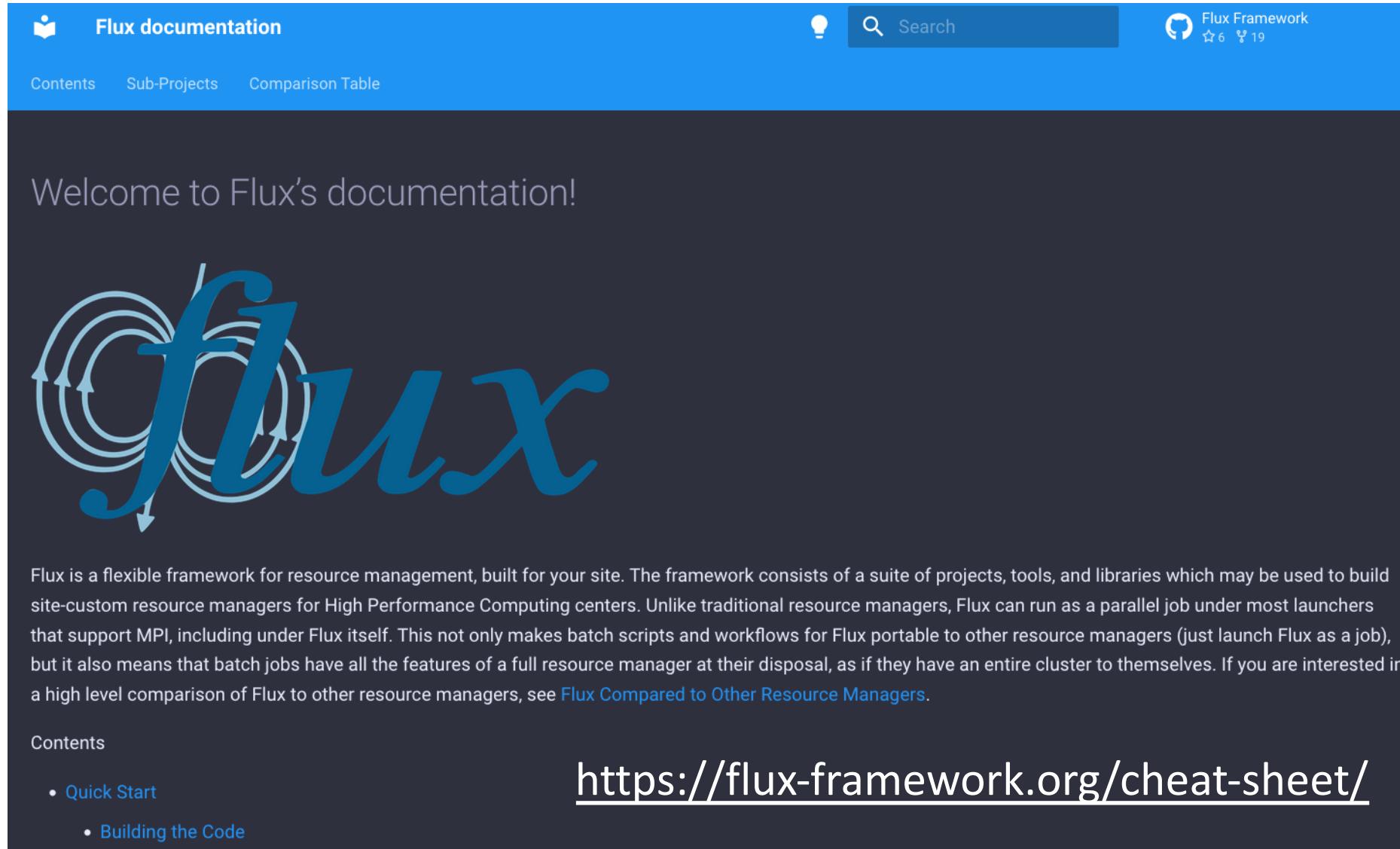
Smaller clusters for user feedback (three are in the top 200 of the Top500)

- Tioga, Corona (pictured here), RZVernal, Tenaya
- Hetchy, Fluke, Elmerfudd

# Updates on capabilities identified last year: Flux core team has made excellent progress leading up to production.

- ✓ Support multiple queues/partitions
- ✓ Support limits and defaults, boosted access/priority
- ✓ Propagate time limits through job hierarchy
- ✓ Replacement for batch #directives
- ✓ CORAL2: rabbit integration, common tools interface
- ✓ Start/stop Flux without losing running jobs
- ✓ Resolve “second order” resiliency issues
  - Solved issues that surfaced during crashes, others will manifest at scale
- ✓ Node-hosted prolog/epilog
  - Still need to launch pro/epilog from rank 0 host
  - Partial release of resources

# Flux is advancing rapidly. (Evidence: dark mode)



The screenshot shows the Flux documentation homepage in dark mode. The header features a blue navigation bar with the "Flux documentation" logo, a search bar, and a "Flux Framework" section showing 6 stars and 19 forks. Below the header, a large "Welcome to Flux's documentation!" message is displayed, followed by the Flux logo. A detailed description of Flux as a flexible framework for resource management is provided, along with a link to compare it with other resource managers. The footer contains a "Contents" section with links to "Quick Start" and "Building the Code".

Flux documentation

Search

Flux Framework

Contents Sub-Projects Comparison Table

Welcome to Flux's documentation!

# flux

Flux is a flexible framework for resource management, built for your site. The framework consists of a suite of projects, tools, and libraries which may be used to build site-custom resource managers for High Performance Computing centers. Unlike traditional resource managers, Flux can run as a parallel job under most launchers that support MPI, including under Flux itself. This not only makes batch scripts and workflows for Flux portable to other resource managers (just launch Flux as a job), but it also means that batch jobs have all the features of a full resource manager at their disposal, as if they have an entire cluster to themselves. If you are interested in a high level comparison of Flux to other resource managers, see [Flux Compared to Other Resource Managers](#).

Contents

- [Quick Start](#)
- [Building the Code](#)

<https://flux-framework.org/cheat-sheet/>

# The Flux team is rapidly filling in user-identified gaps.

- Deprecate `flux mini`, now `flux run`, `flux submit`, etc.
- Introduce `#flux`: batch directives ([flux-batch\(1\) man page](#))
- Groundwork for executing jobs under systemd
  - eventually support Flux restart without running job loss
- `flux filemap` and job shell `stage-in` plugin for file broadcast
- Support `-o cpu-affinity=map`: option explicitly map cores to tasks
- RFC 34 Flux Task Map: a compact mapping between job task ranks and nodeids
- Only prepend the path to [`flux\(1\)`](#) if necessary (avoid prepending `/usr/bin` to PATH)
- Unbuffered I/O option for `flux run` and ability to direct stdin to subset of tasks
- Support rank, host constraints in `--requires=` submission option



# And providing new capabilities.

## Side install Python bindings via flux-python

- if Python not ABI compatible, `pip install` Flux bindings
  - must match Flux version of bindings and core

## Send signal to a job before it expires

- used to checkpoint before walltime

## --add-file - attach small files to job submission

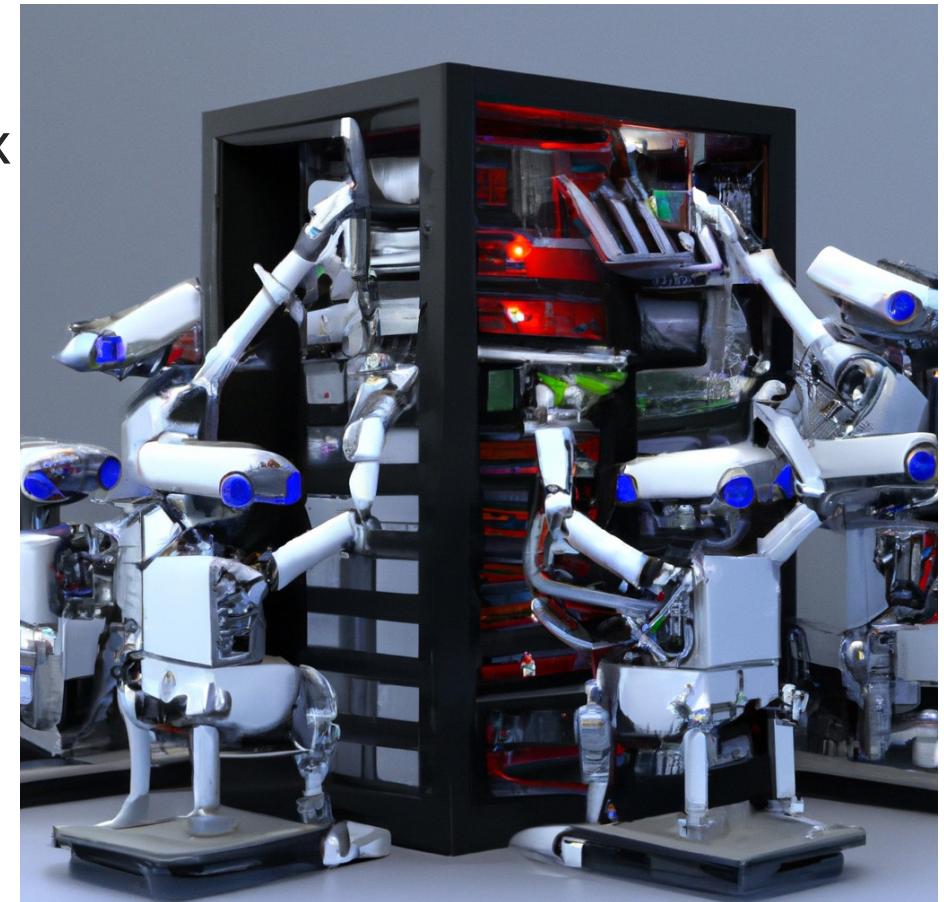
- can be signed and saved to compute nodes

## Flux-accounting runs as a service

- run anywhere on cluster rather than mgt node

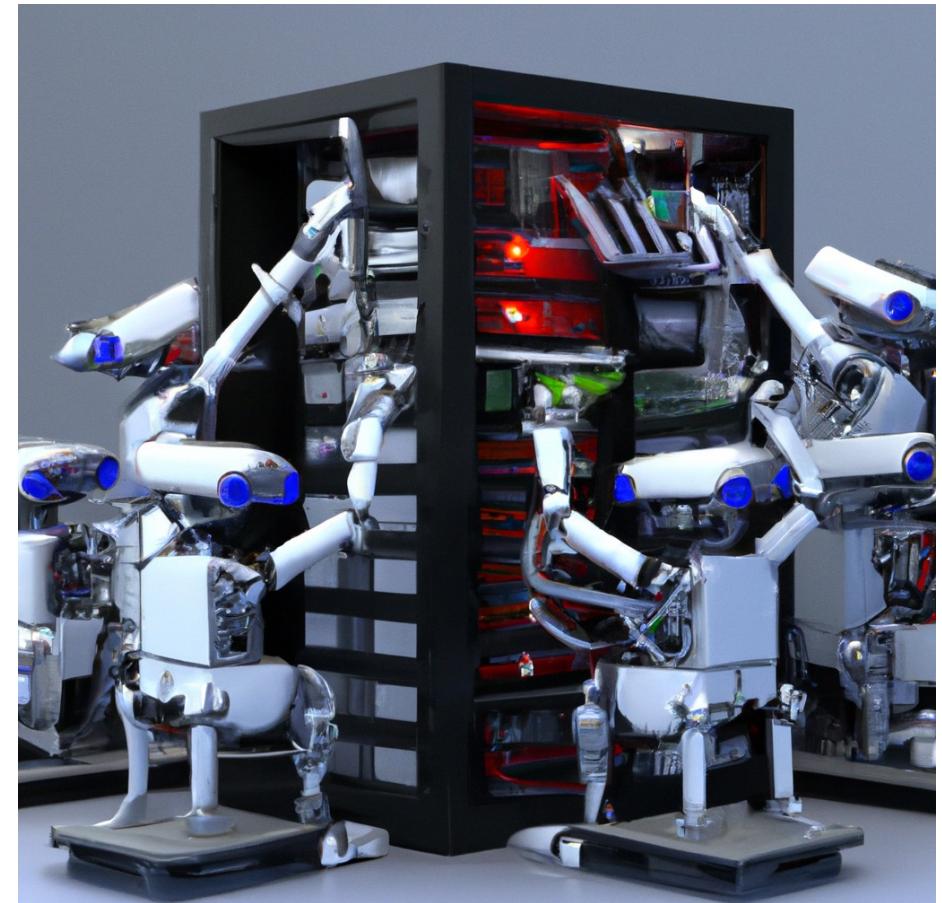
## Plugin query callback

- with multi-factor priority plugin, query returns user and bank's job counts
- explains limit enforcement



## And providing new capabilities (cont.).

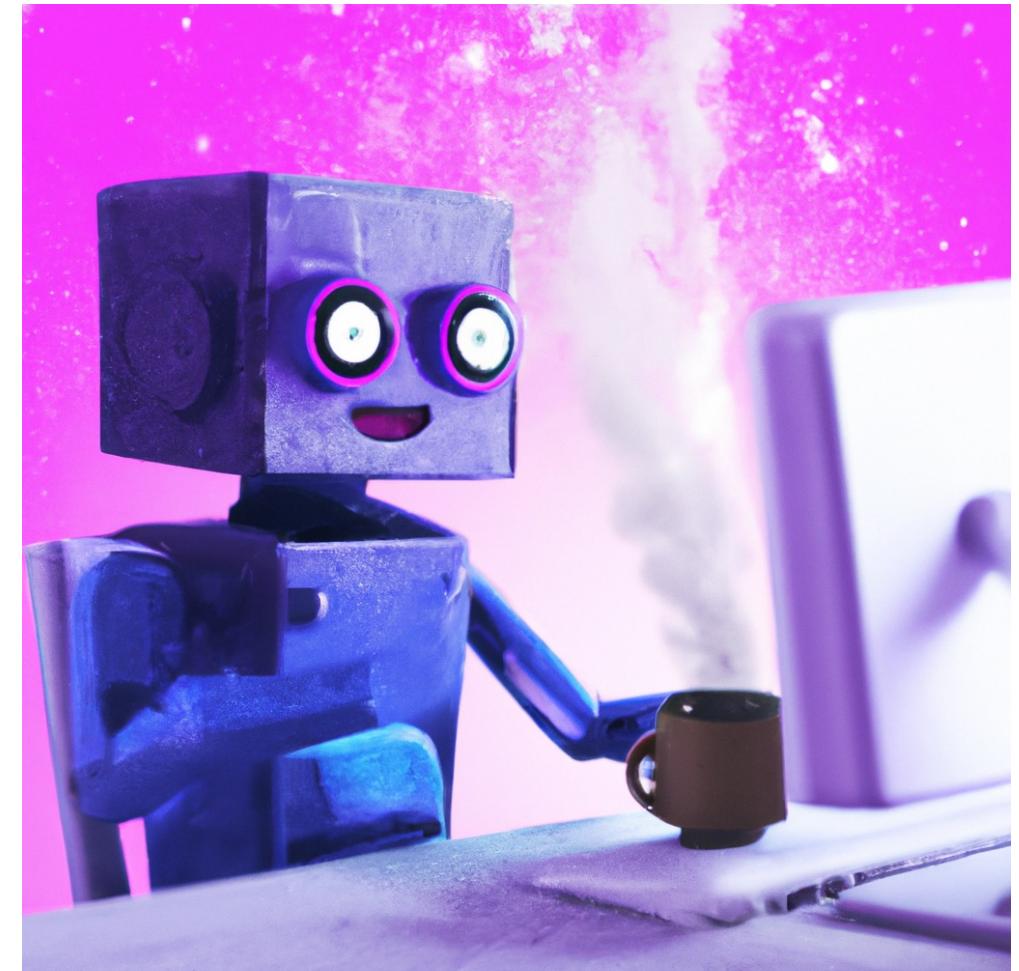
- new command **flux-watch** to watch output for a single or multiple flux jobs
- Python API for monitoring job output
- allow memory limits on jobs to be set for system instance jobs
- easy configuration of batch job instances with **--conf** option to **flux-batch** and **flux-alloc**



# The Flux team has developed utilities to improve user experience.

## New utilities

- **flux job last** - list last submitted jobs for the current user
  - takes **slice** option
- **flux job timeleft** - get remaining time
- **flux pgrep** and **flux pkill** - query/cancel jobs by name, id range, etc.
- **flux job taskmap** - query taskmap for jobs
  - rank-to-host mapping

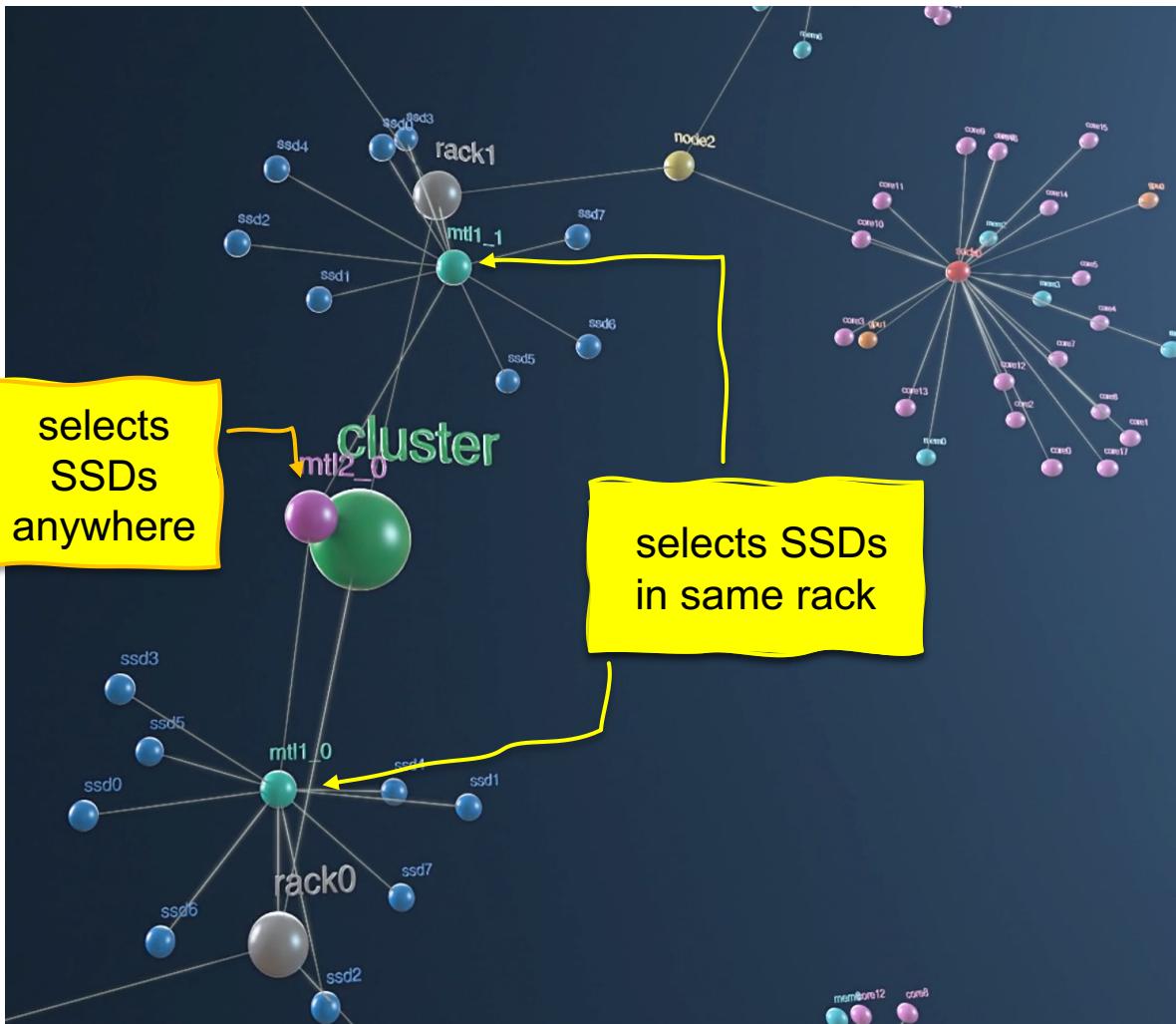


# The rabbits of El Cap present a fearsome scheduling problem.

- Multi-tiered storage features “rabbit” nodes
  - 18 SSDs, direct PCIe to chassis compute nodes
- Dynamically configured as node-local storage or job-global
  - node-local via PCIe, global via network
  - single rabbit can serve both at once
  - scheduler must handle both
- Can be allocated independent of jobs
- Too difficult for traditional schedulers



# Fluxion's directed-graph approach addresses the rabbit challenge and facilitates cloud integration.



- Fluxion schedules rack-local and global storage with no code change (caveats)
  - inefficient for scheduling same resource type multiple times
  - affects jobs requesting multiple rabbit allocations
  - known issue to be fixed before El Cap deployment
- Directed-graph representation enables scheduling, managing dynamic, cloud resources

# The what and why of movement to the cloud; HPC doesn't want to be left behind.



## The cloud is an environment (public, private) that supports:

- Portability, reproducibility (e.g., containerization)
- Resiliency, efficiency (e.g., resource dynamism, elasticity, declarative management)
- Reduced complexity via automation (autoscaling, elasticity, declarative management)

## Companies rent this environment; hugely profitable

- projected to \$920B by 2025, 20% CAGR (20-25)<sup>1</sup> vs HPC: \$40B by 2025, 20-25 CAGR 8%<sup>2</sup>

**CACM:** New economic cycle of computing leads to greater hardware specialization. Areas more distinct and provide fewer benefits to others. **Areas that get left behind**<sup>3</sup>:

- See little performance benefit
- Market too small to justify upfront costs
- Cannot coordinate demand (cloud)

**Focuses on hardware, but software development is crucial, closely connected**

<sup>1</sup>Gartner 2022, <sup>2</sup>Hyperion 2021, <sup>3</sup>*The Decline of Computers as a General Purpose Technology*, CACM March 2021

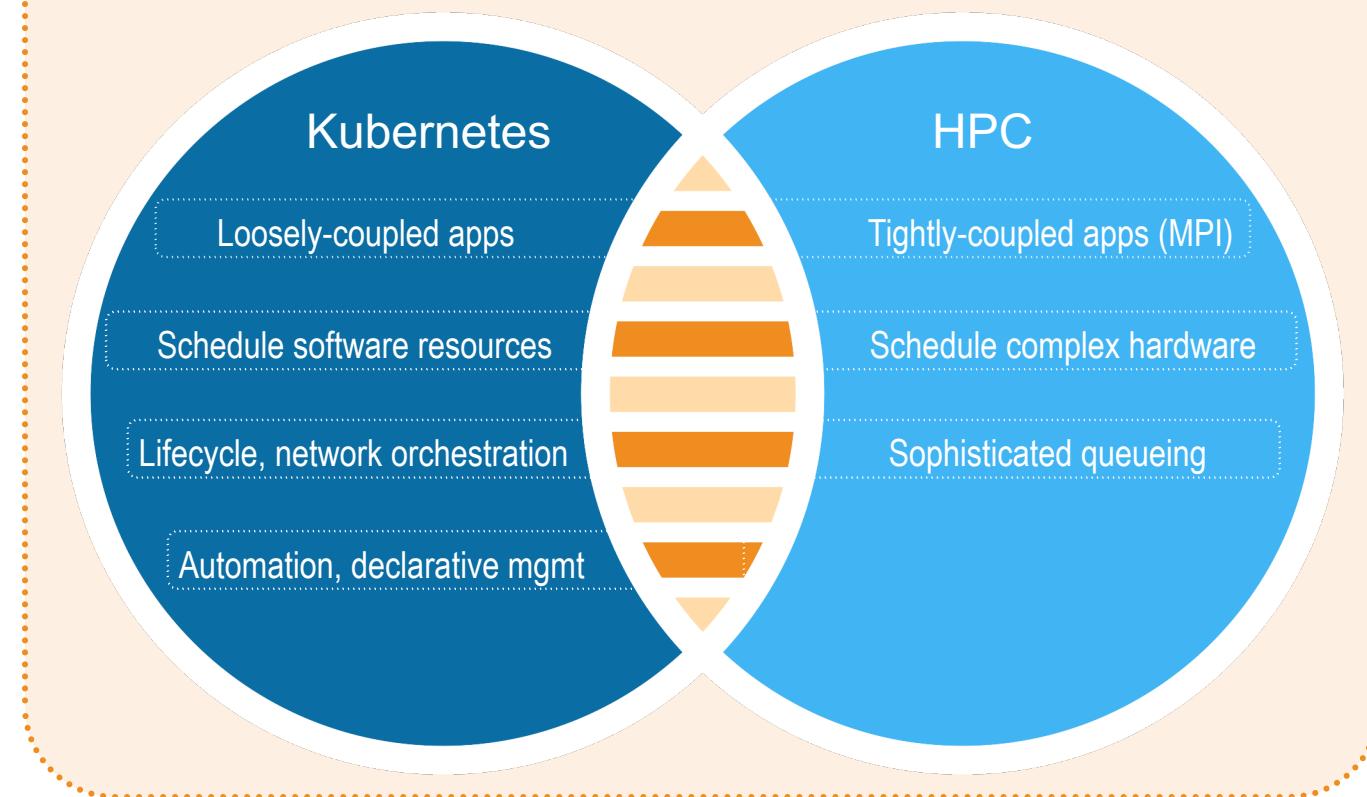
# A key to converged computing is combining HPC scheduling and resource management with Kubernetes.

## Kubernetes (K8s):

- cloud “OS” with 75K contributors (one of the fastest growing OSS projects ever)
- designed for loosely coupled apps
- not focused on performance (scheduling limitations, throughput...)

## Converged Computing LDRD:

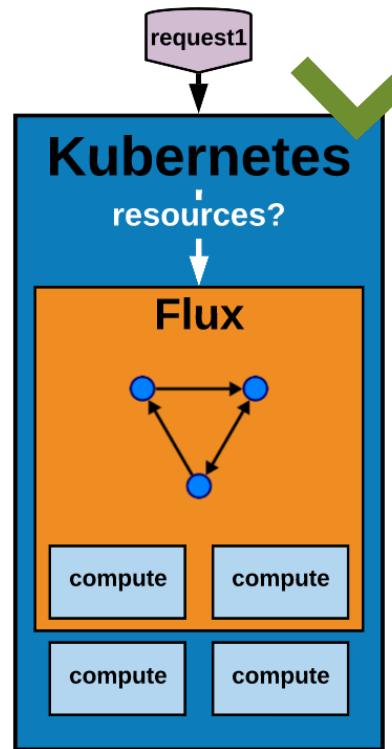
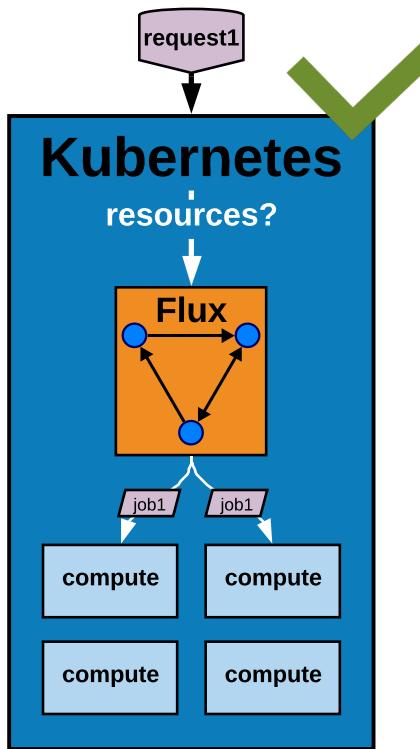
how to create a converged environment composed of the best of both worlds?



## HPC:

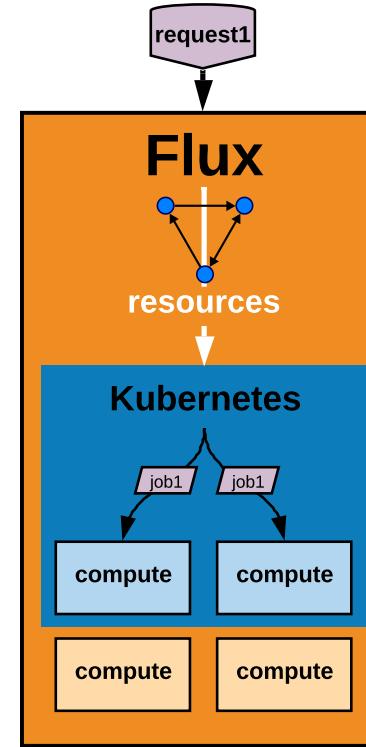
- performance is in the name
- very difficult to manage modern workflows
- not designed for dynamism/elasticity

# LLNL converged computing project advances convergence with representative, Flux-based models.

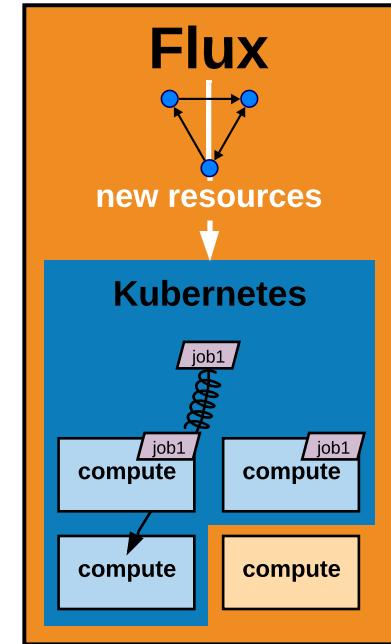


L1: APIs, resource representations, scheduling.  
Fluence enables portable converged workflows

Portability is a component of performance!



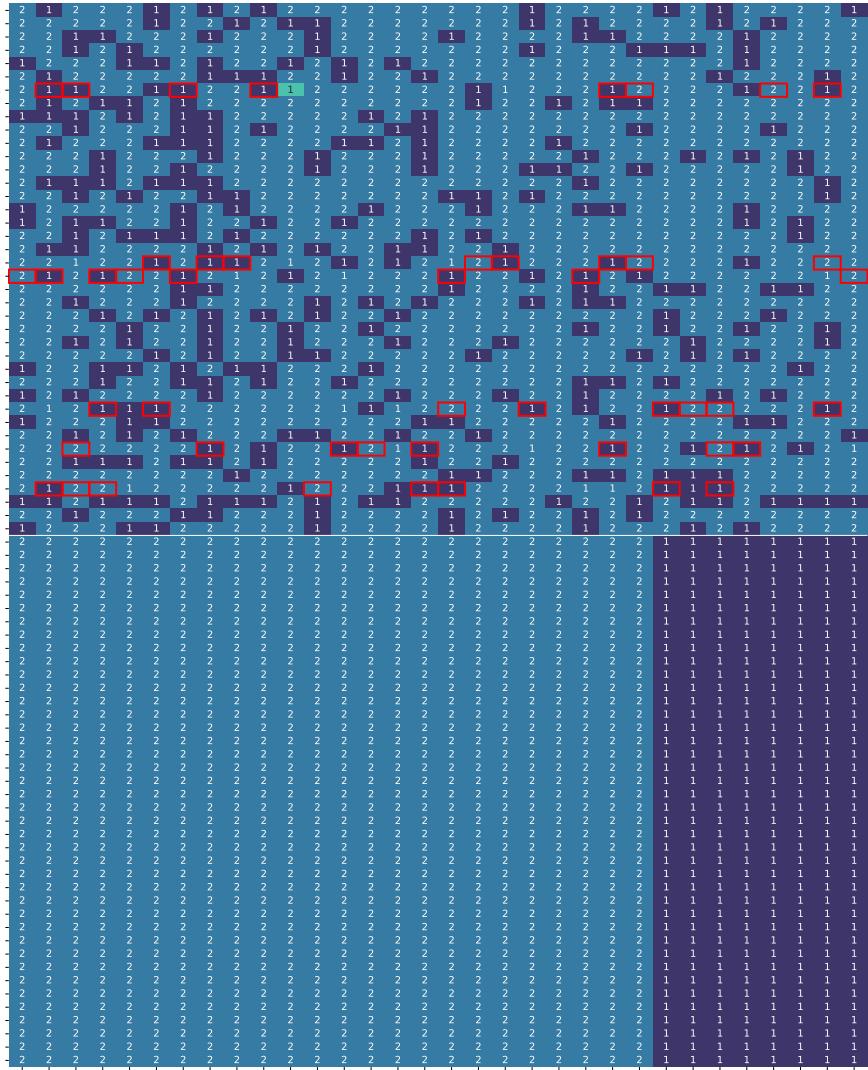
L3: Cloud nesting model reduces software complexity, increases automation, performance



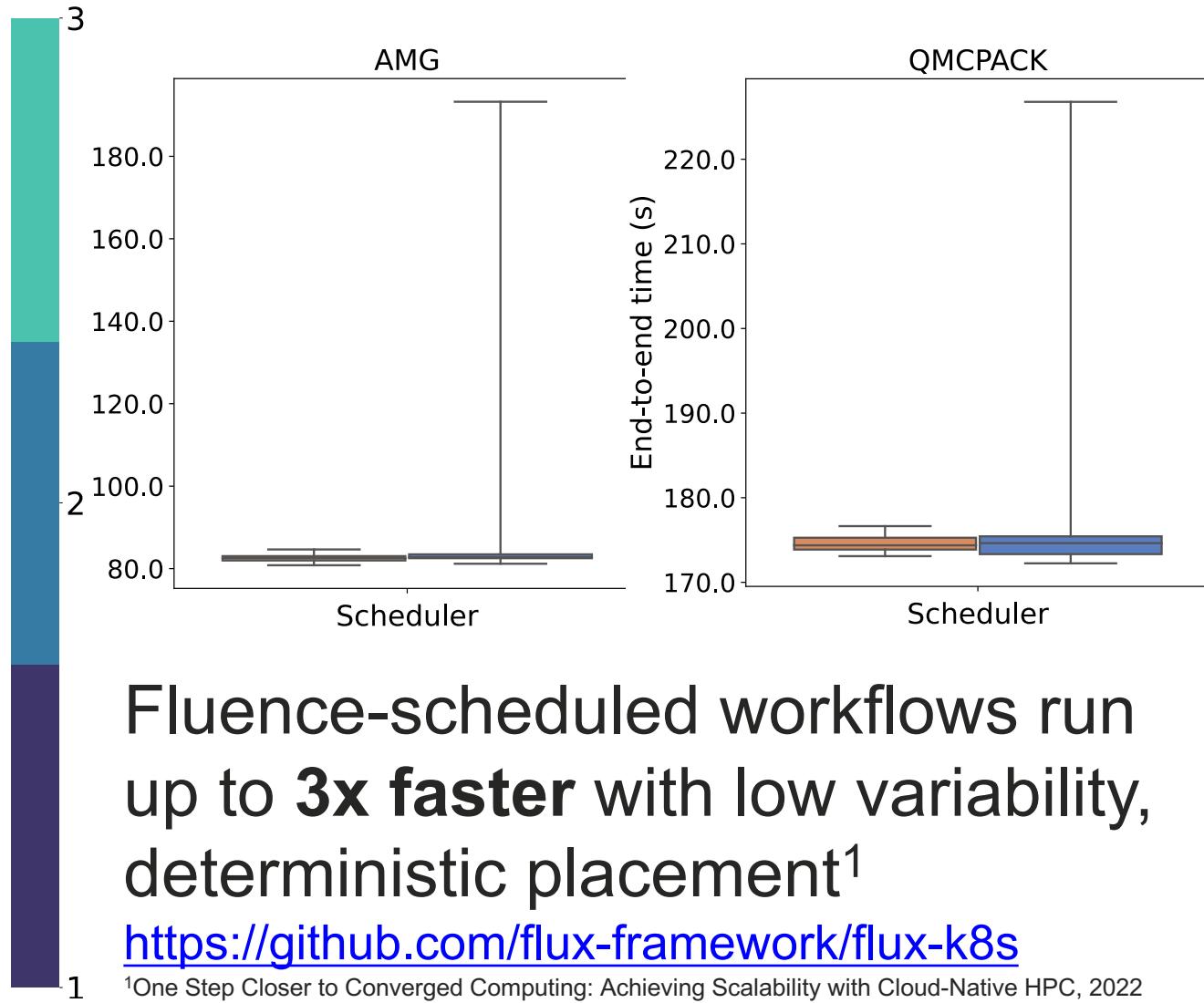
L4: Elastic cloud nesting model enables autoscaling and dynamism

# The Fluence project improves performance and reduces workload variability.

kube-scheduler



Fluence

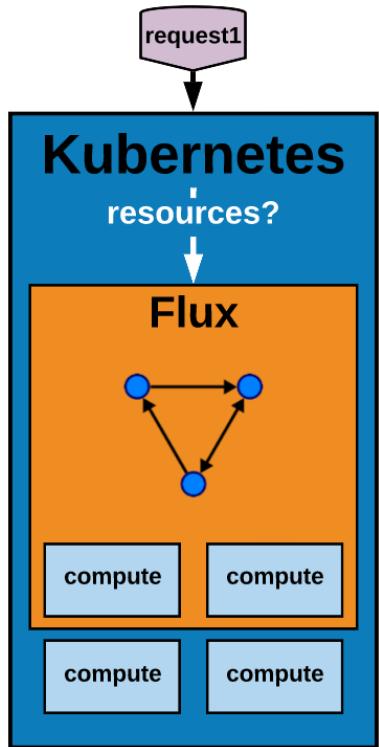


Fluence-scheduled workflows run up to **3x faster** with low variability, deterministic placement<sup>1</sup>

<https://github.com/flux-framework/flux-k8s>

<sup>1</sup>One Step Closer to Converged Computing: Achieving Scalability with Cloud-Native HPC, 2022

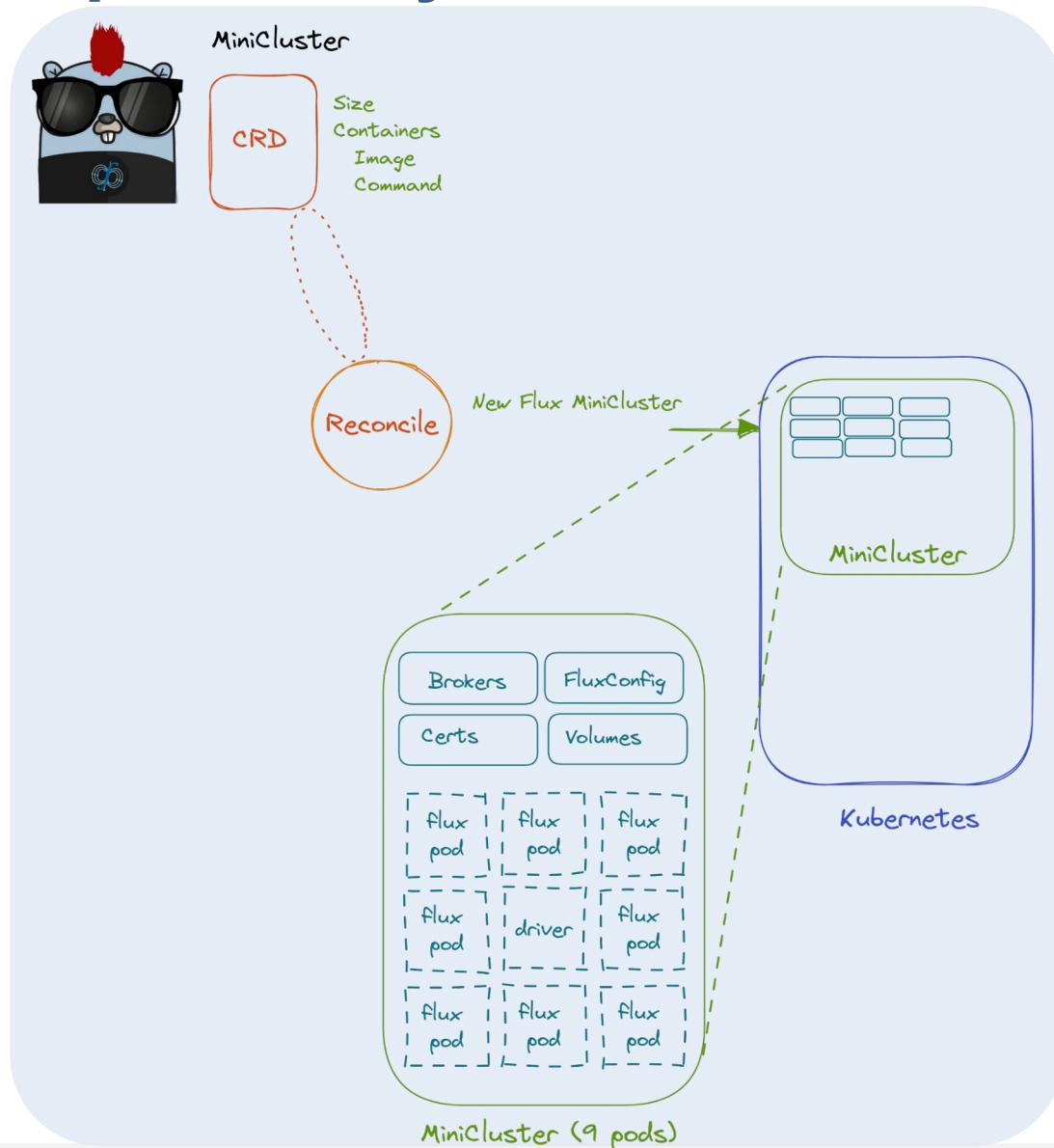
# The project has enabled workflow portability via the Flux Operator.



## Flux Operator

- Bootstrap Flux in K8s; “fractal scheduling” of pods
- Scalable MPI bootstrap
- Ported 14 HPC benchmarks, tools, and full workflow
- JobSet API
- Autoscaling in progress
- RESTful interface in progress

<https://github.com/flux-framework/flux-operator>



# We formed industry and academic collaborations to tackle converged computing, contribute to community, and advocate for HPC.



**IBM T.J. Watson Research Center:**  
Expertise in K8s Scheduling



**UTK:**  
Science workflows,  
converged computing

**Red Hat:**  
Developers of OpenShift K8s platform

**LLNL:**  
Dan Milroy  
Giorgis Georgakoudis  
Md Rajib Hossen  
Zeke Morton  
Tapasya Patki  
Abhik Sarkar  
Vanessa Sochat  
Jae-Seung Yeom

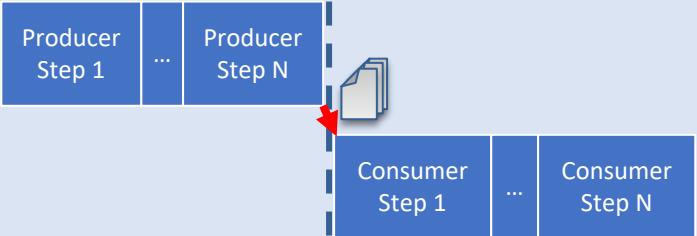
**AWS:**  
Largest cloud platform, deep HPC expertise

## Formalizing collaboration with Google!

# Enabling High Throughput Data Movement with the Dynamic and Asynchronous Data Streamliner (DYAD)

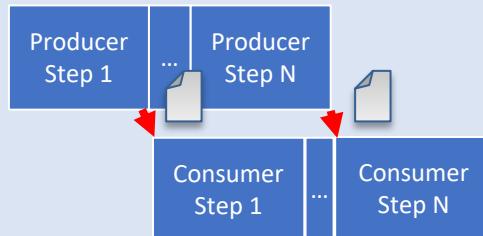
## Existing Approaches for Data Movement

### Sequential Approach



Strengths: easy-to-use, portable across storage solutions  
Weaknesses: slow, dependent on parallel file system

### In Situ/In Transit Approach



Strengths: fast, dependent on local data staging  
Weaknesses: hard-to-use, not portable across storage solutions

## Advanced Data Movement with DYAD: Taking the Best of Both Existing Approaches

### Easy to configure and run

(1) Launch the DYAD Service  
`$ flux exec -r all \flux module load dyad.so <producer directory>`

### (2) Configure DYAD for Apps

```
export DYAD_KVS_NAMESPACE=...
export DYAD_PATH_PRODUCER=...
export DYAD_PATH_CONSUMER=...
```

### (3) Load the DYAD Wrapper (when not using C++ API)

```
export \
LD_PRELOAD="dyad_wrapper.so"
```

### (4) Run application as usual

### Simple to integrate

C (and soon Python) Code  
No Change Required!

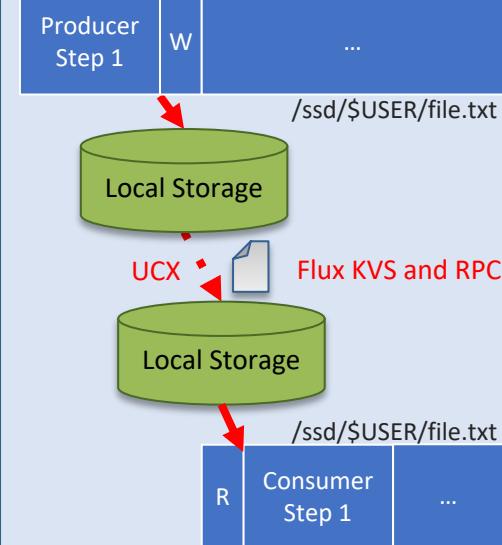
C++ Code  
`#include <dyad_stream_api.hpp>
using namespace std;`

```
void write_file(char* fname,
                void* buf,
                size_t buflen)
```

```
{
    dyad::ofstream_dyad ofs;
    ofs.open(fname, ios::binary);
    ofstream& real_ofs =
        ofs.get_stream();
    real_ofs.write(buf, buflen);
    ofs.close();
```

Interested in other languages?  
Reach out!

### Transparent to applications



- Application sees read/write to local storage
- DYAD moves data as needed between producer and consumer

### Targeted Use Cases

Scientific Computing Workflows (e.g., MuMMI, AHA Moles), Data Parallel Deep Learning (e.g., DLIO Benchmark)  
Interested in using DYAD? Reach out!

### Project Team

LLNL: Jae-Seung Yeom, Hari Devarajan  
University of Tennessee: Ian Lumsden, Jakob Luettgau, Jack Marquez, Michela Taufer

Check out our eScience '22 short paper:

<https://doi.org/10.1109/eScience55777.2022.00068>



# Accessing the hands-on tutorial

- Using our EKS cluster at <https://tutorial.flux-framework.org>
  - Choose a unique username (if not, you'll be sharing a pod)
  - Password: XXXX
  - Navigate to **flux-tutorial/notebook/** and double-click **flux.ipynb** to start
    - To execute a cell in JupyterLab: **Shift+Enter**
  - It's a shared instance- please don't run computationally demanding tasks in your pod
  - The cluster will disappear shortly after the tutorial
- Running locally with Docker:
  - git clone <https://github.com/flux-framework/Tutorials.git>; README in the 2023-RADIUSS-AWS JupyterNotebook directory

Thank you!  
Questions?

<https://github.com/flux-framework/Tutorials.git>