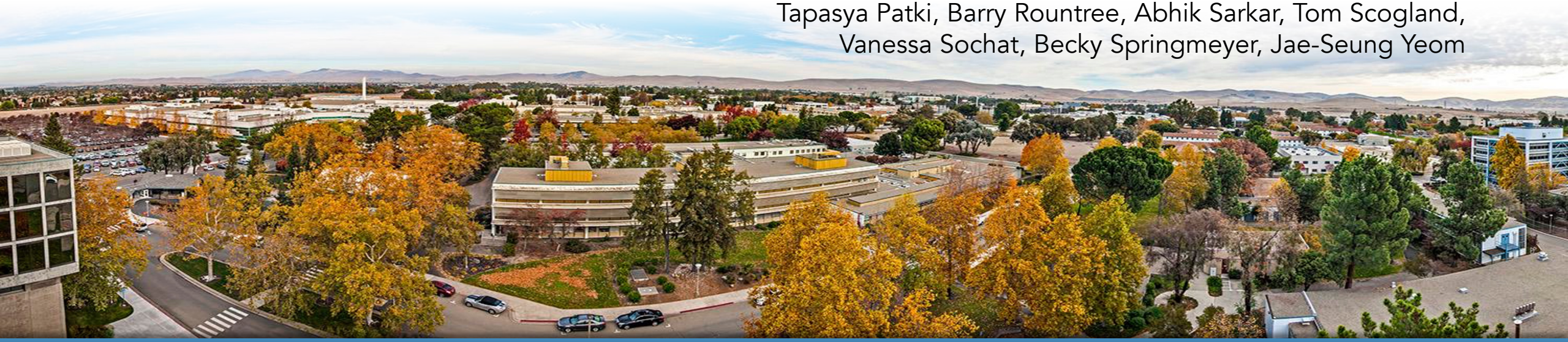
The Flux framework: overcoming scheduling and management challenges of exascale workflows

RADIUSS Tutorial on AWS
August 17, 2022

Dan Milroy

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

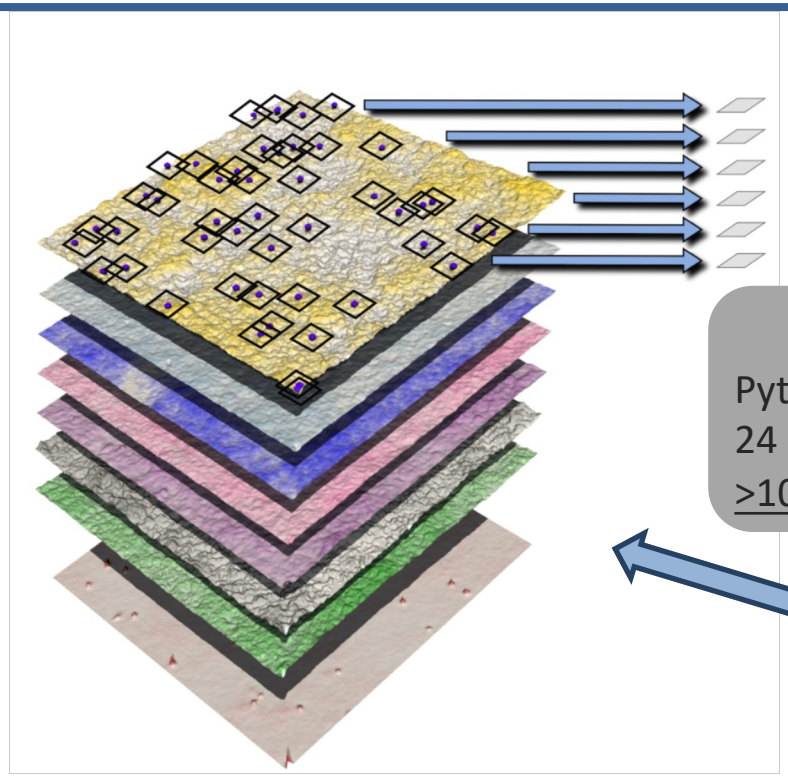


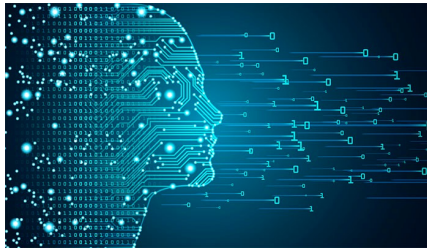
LLNL-PRES-1059443

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

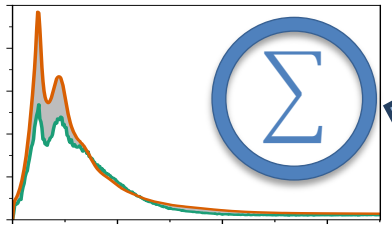
Lawrence Livermore National Laboratory

Sierra pre-exascale system is a wakeup call (MuMMI).

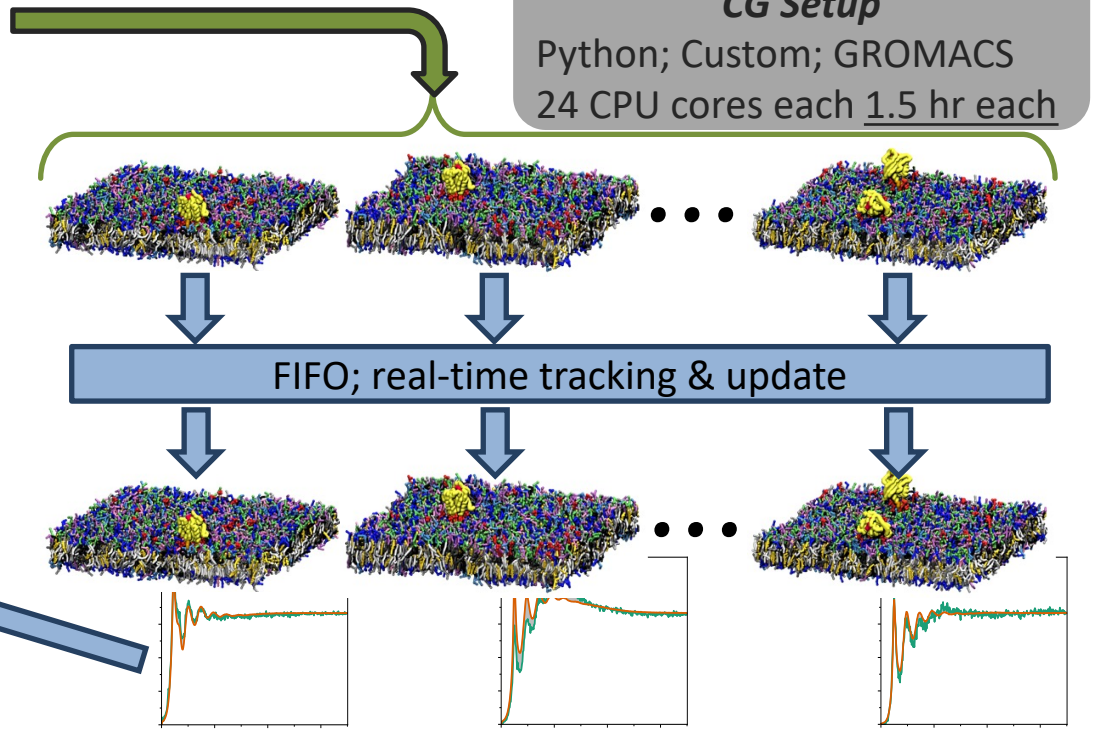




ML-based Selection
Python; ML frameworks; FAISS
24 CPU cores
>1000 decisions per minute



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



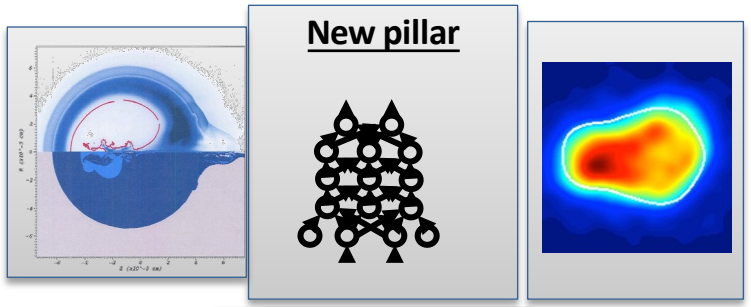
Single Macro Simulation
C++ (with MPI); MOOSE; ddcMD
24 CPU cores/node; 2400 MPI tasks
242 GB per day

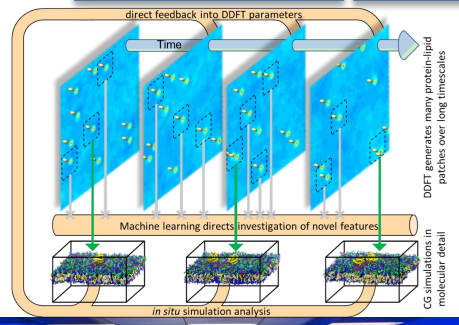
Analysis Aggreg. & Feedback
Python; Custom
24 CPU cores
120,000 reads per cycle

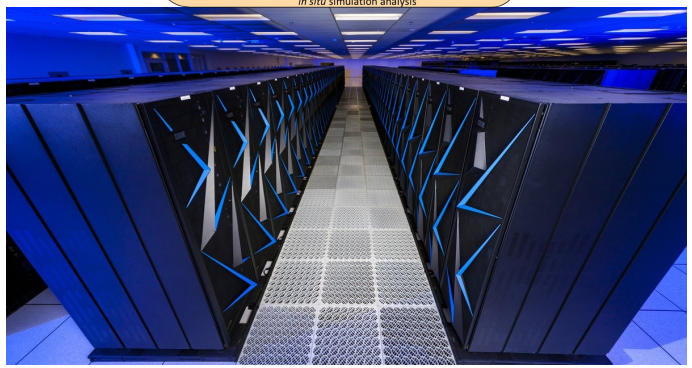
CG Simulation
C++ (with CUDA); ddcMD
1 GPU + 1 CPU core each
1.04 μs and ~6.5 GB per day

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

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

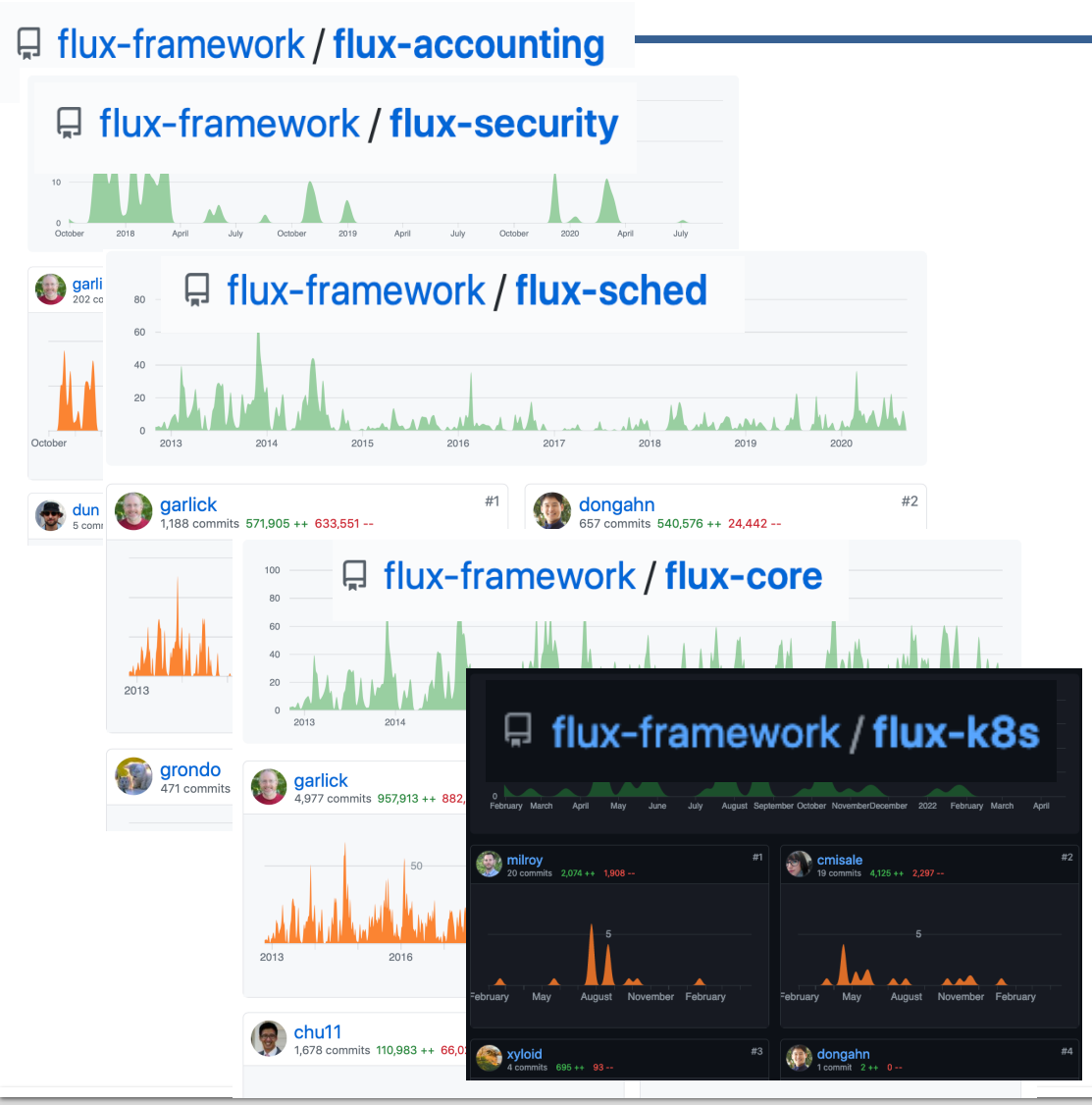






- Co-scheduling
- Job throughput
- Job communication/coordination
- Portability
- Extremely heterogenous resources

Flux solves key technical problems that emerge from these trends.

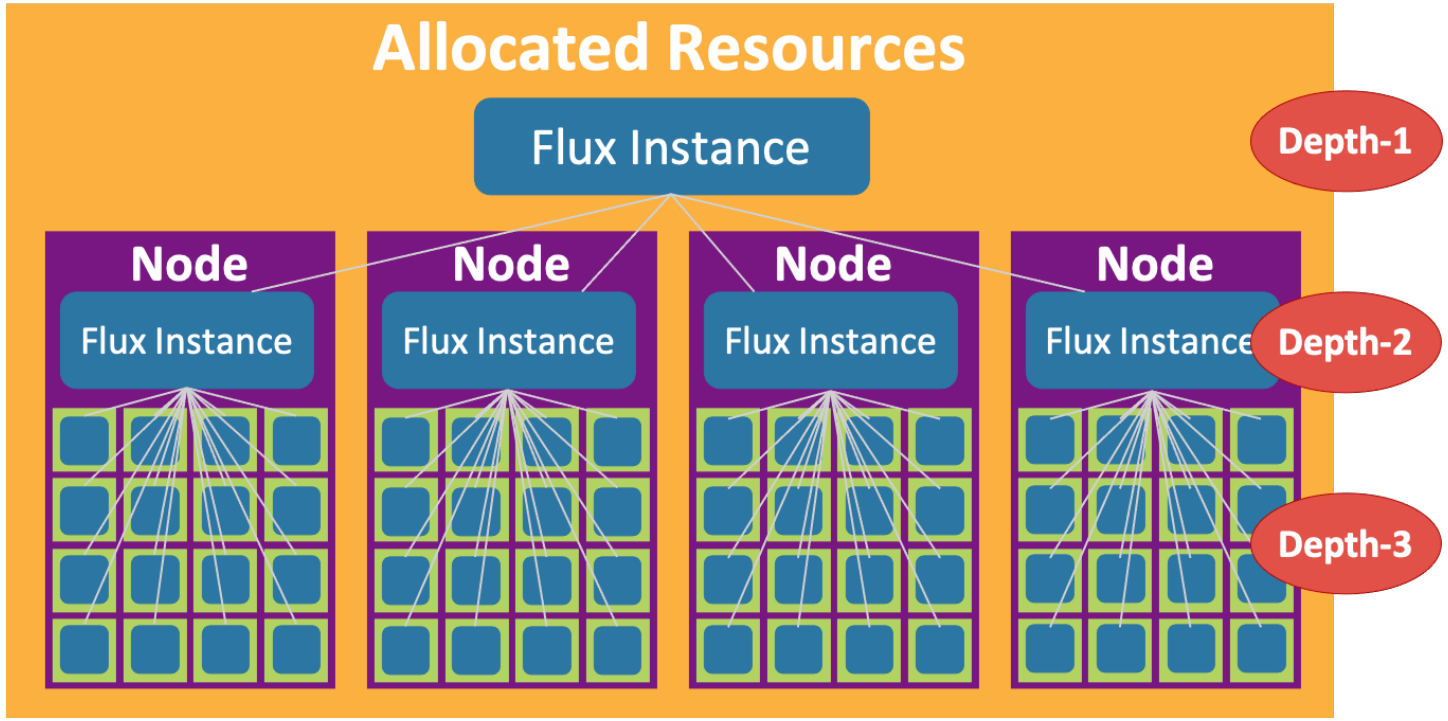


- Open-source project in active development at flux-framework GitHub organization
  - Multiple projects: flux-core, -sched, -security, -accounting, -k8s etc.
  - Over 15 contributors including some principal engineers behind Slurm
- Single-user and System instance modes
  - Single-user mode in production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

flux

2021 R&D 100 WINNER

Flux's fully hierarchical resource management solves primary deficiencies of the conventional approach.

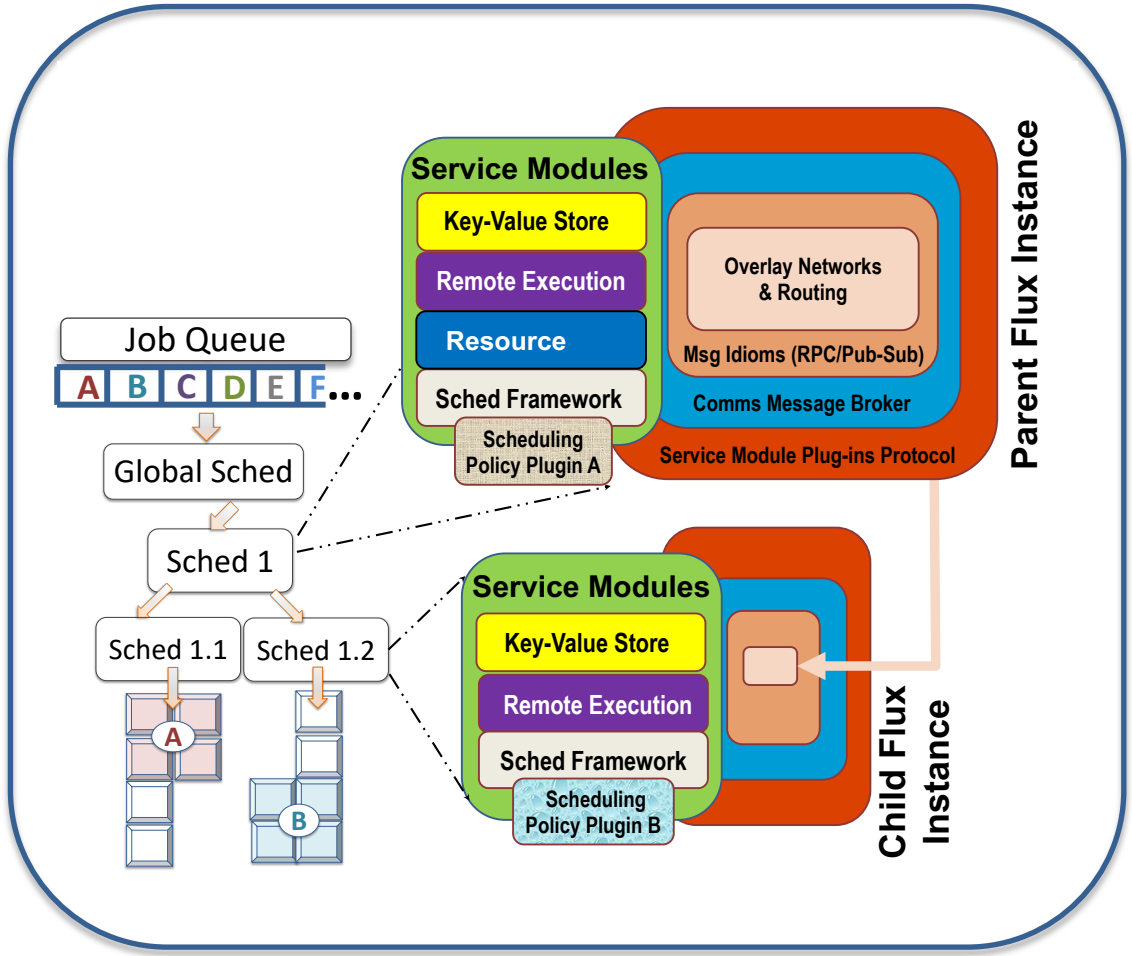


"Fractal scheduling" mitigates centralized scheduler bottleneck

- handles high throughput
- job steps needn't hit central sched

- Full workflow-enablement support
  - Via hierarchical resource subdivision
  - Sub-resource manager per subdivision with service specialization
  - E.g., at LLNL: MuMMI, AHA MoleS, UQP
- Capable of managing resources from almost anywhere
  - Bare metal resources, virtual machines in the cloud, HPC resources allocated by another workload manager
  - Workflows only need to program to Flux
- Provide rich and well-defined interfaces
  - Facilitate communications and coordination among tasks within a workflow
  - CLI, Python, C

Flux is architected to embody our fully hierarchical scheduling model.



Flux framework

Techniques

Scheduler Specialization

Scheduler Parallelism

Rich API set

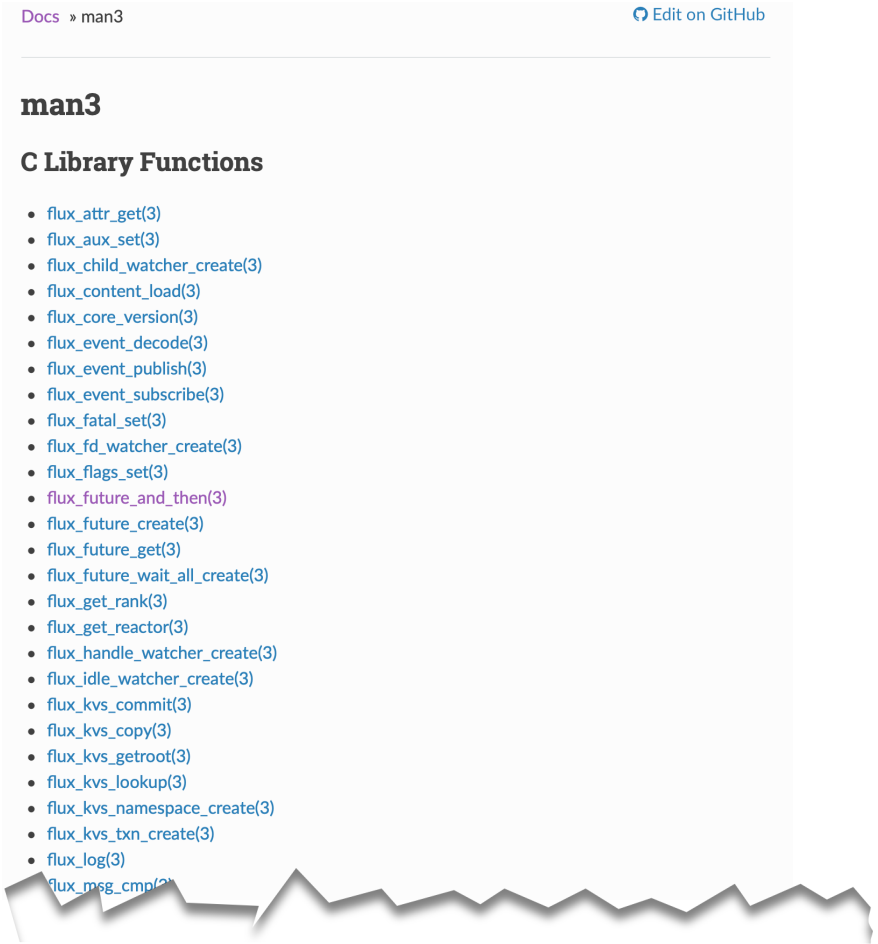
Consistent API set

Challenges

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

A rich set of well-defined APIs enables easy job coordination and communication.

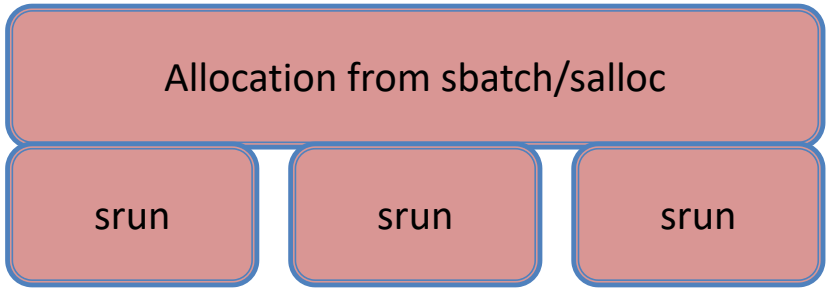
- Jobs in ensemble-based simulations often require close coordination and communication with the scheduler as well as among them.
  - Traditional CLI-based approach can be slow and cumbersome.
  - Ad hoc approaches (e.g., many empty files) can lead to many side effects.
- Flux provides well-known communication primitives.
  - Pub/sub, request-response, and send-recv patterns
- High-level services
  - Key-value store (KVS) API
  - Job API (submit, wait, state change notification, etc)
- Flux's APIs are consistent across different platforms



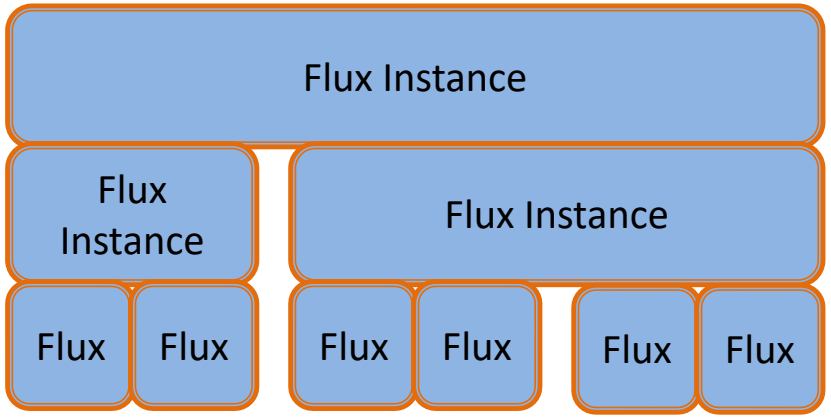
https://flux-framework.readthedocs.io/projects/flux-core/en/latest/index.html

Flux's hierarchical scheduling makes managing and scheduling complex allocations easier.

- Significant start-up cost for running on new resource manager (portability)
  - Learn to use the manager
  - Translate batch script logic
- Traditional resource managers can't subdivide allocations (hierarchy)
  - Slurm has two levels: **`sbatch`** creates allocation, **`srun`** launches parallel task
  - **`sbatch`** from inside an allocation starts a separate allocation
- Flux's hierarchical scheduling provides a natural solution
  - Request one large allocation and partition schedulable resource subgroups



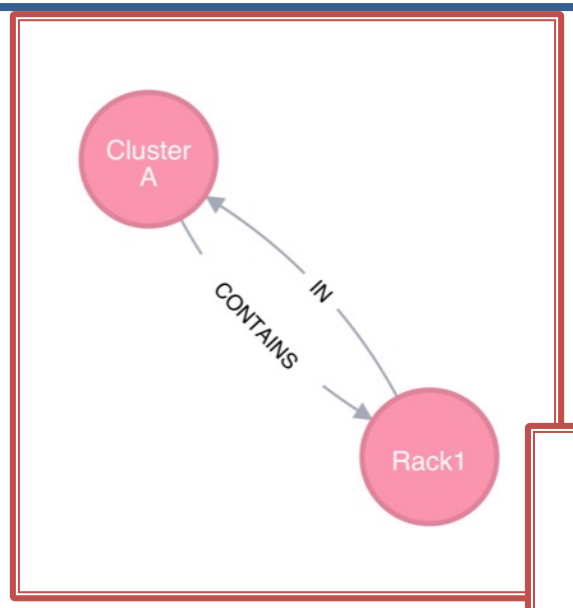
The Slurm hierarchy



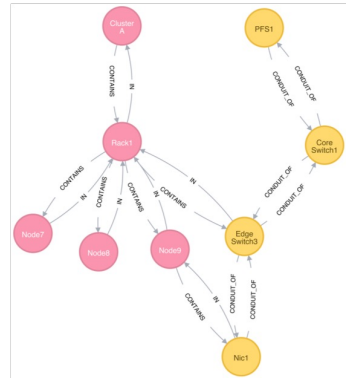
One possible Flux hierarchy

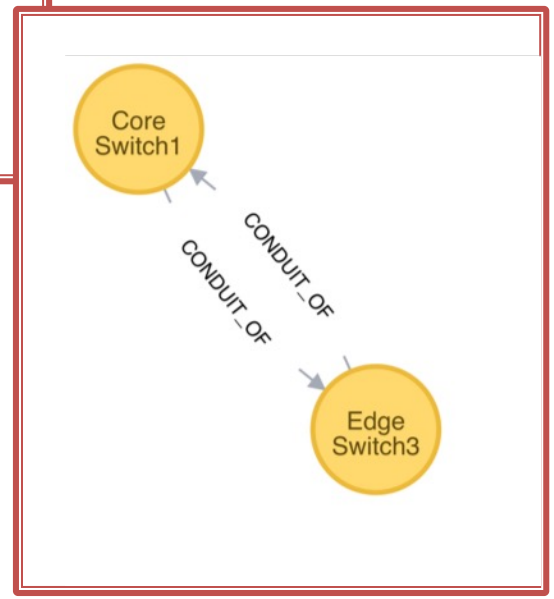
Flux pioneers and uses graph-based scheduling to manage complex combinations of extremely heterogenous resources.

- Traditional resource data models are largely ineffective for resource heterogeneity
  - designed when systems were simpler
  - node-centric models
- 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 for graph traversal and allocation



Containment subsystem





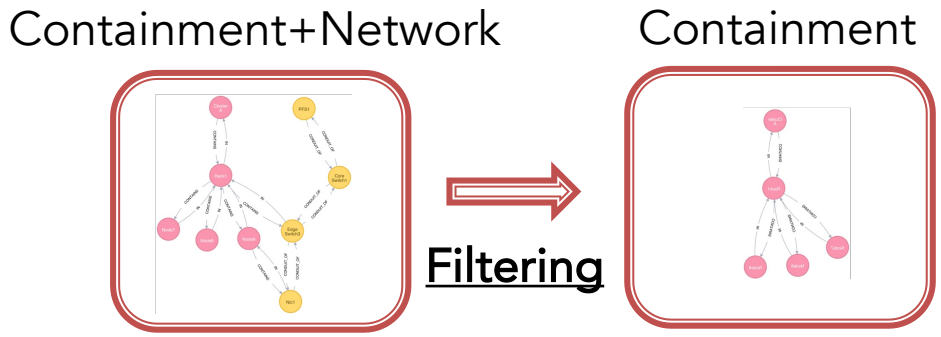
Network connectivity subsystem

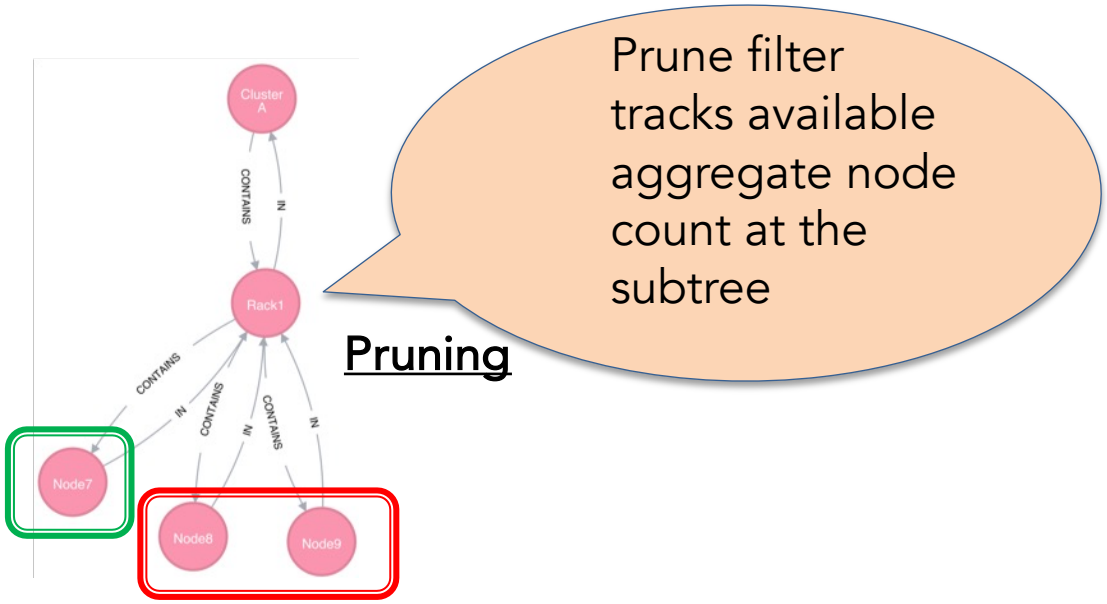
We use graph filtering and pruned searching to manage the graph complexity and optimize our graph search.

- The total graph can be quite complex
  - Two techniques to manage the graph complexity and scalability

1. 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

2. 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



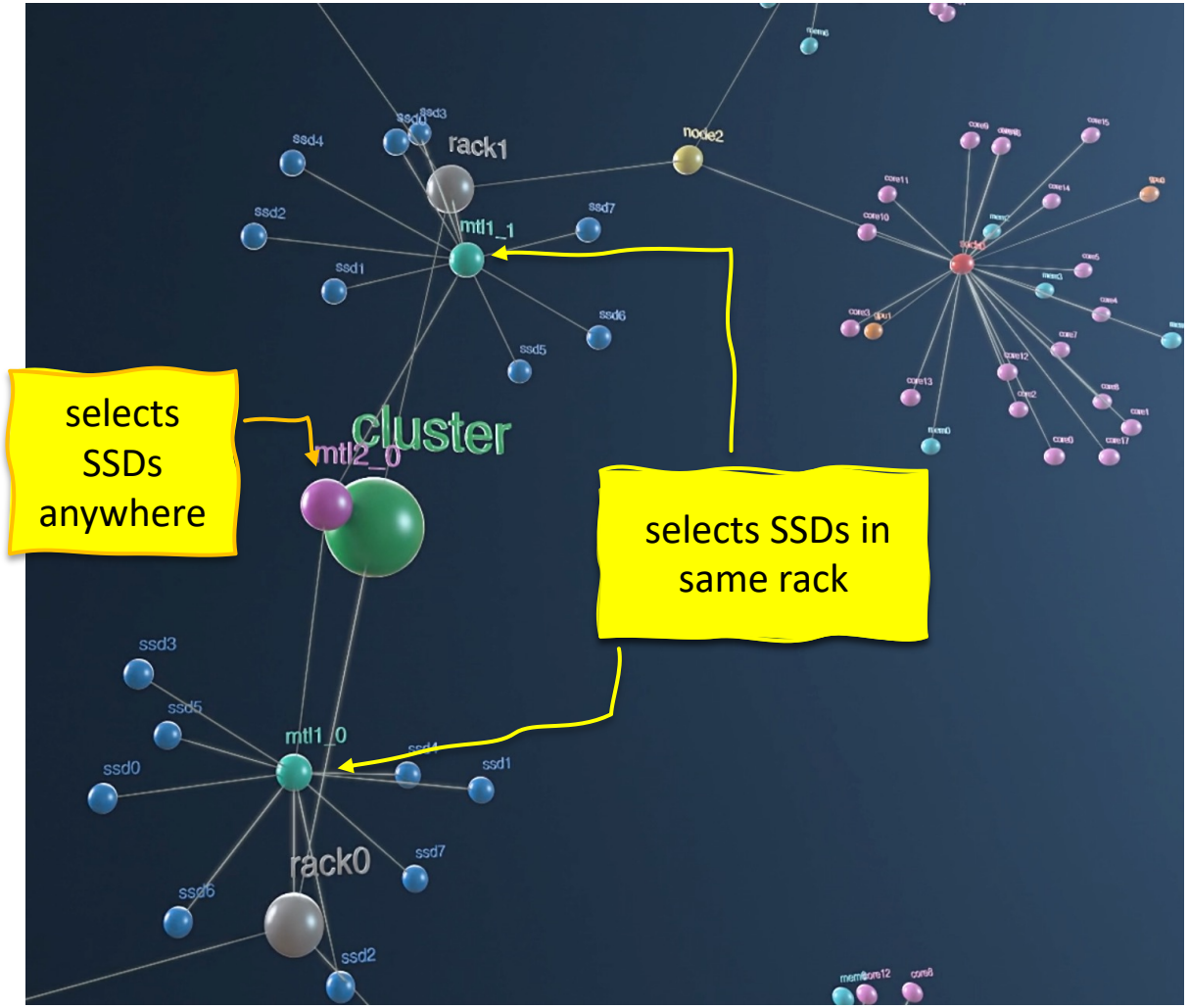


Flux's graph-oriented canonical jobspec allows for a highly expressive resource request specification.

- Graph-oriented resource requirements
  - Express the resource requirements of a program to the scheduler
  - Express program attributes such as arguments, runtime, and task layout to the execution service
- cluster→ rack[2]→slot[3]→node[1]→socket[2]→core[18]
- ***slot*** is the only non-physical resource type
  - Represent a schedulable place where program processes will be spawned and contained
- Tasks section references slot and defines command

```
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
```

Fluxion graph approach solves the critical scheduling need for El Capitan's Rabbit multi-tired storage.

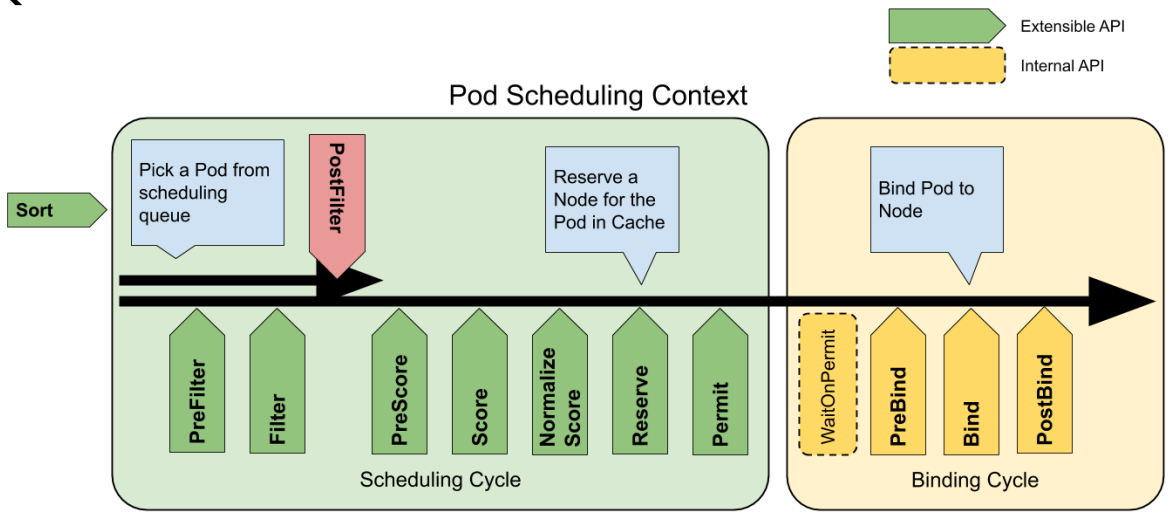


- Schedule SSDs in each rack
  - mount as node-local storage to compute in same rack
  - used to build ephemeral Lustre
  - deemed too difficult for traditional schedulers
- Easily enabled by Fluxion; no code change
- End-to-end job state transition
  - DataWarp/Rabbit mock service containers by HPE
  - Uses Kubernetes CRD

Team formed industry collaborations to tackle converged computing challenges and contribute to community.

We bring complementary backgrounds in HPC, cloud, and performance-oriented orchestration

- LLNL LDRD team
- **IBM T.J. Watson Research Center**: T. Elengikal, M. Drocco, C. Misale, Y. Park
- **Red Hat**: E. Arango Gutierrez
- Newly added **AWS**: E. Bollig, M. Hugues, H. Poxon, L. Wofford
- **Integrate Fluxion into Kubernetes via** KubeFlux



Accessing the hands-on tutorial

- Using our EC2 instance at **https://tutorial.flux-framework.org**
  - Choose a unique username (if not, you'll be sharing with someone else)
  - Password: **<password>**
  - Double-click **radiuss-aws-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 container
  - The instance will disappear shortly after the tutorial
- Running locally with Docker:
  - **git clone https://github.com/flux-framework/Tutorials.git**; README in the 2022-RADIUSS-AWS directory