Flux: Overcoming Scheduling Challenges for Exascale Workflows

Seminar at NERSC, Sep 11, 2020

Dong H. Ahn and Stephen Herbein

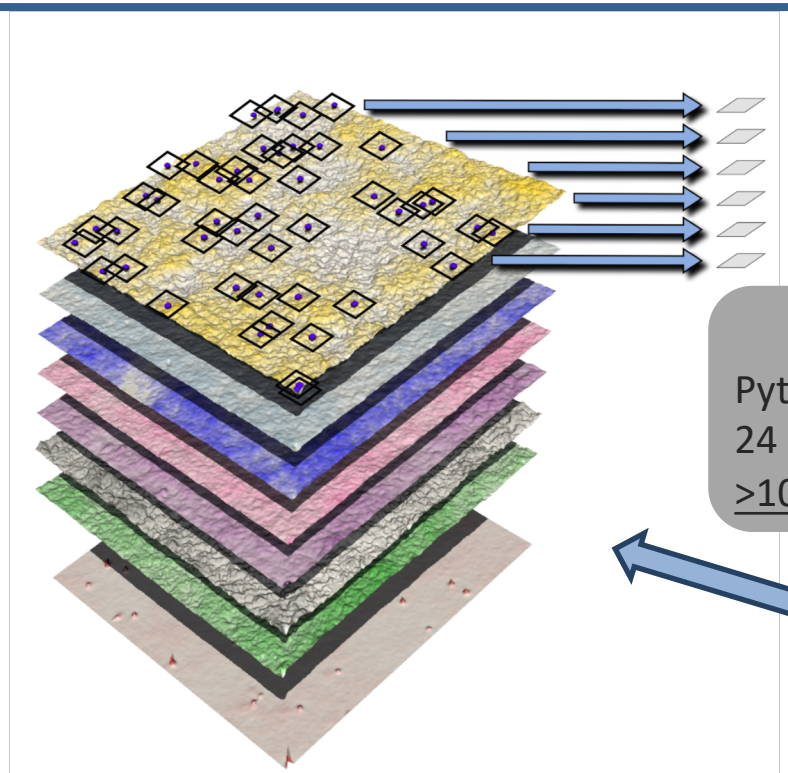
Acknowledgment: Ned Bass, Albert Chu, Kyle Chard, James Corbett, Ryan Day, Franc Di Natalie, David Domyancic, Phil Eckert, Jim Garlick, Elsa Gonsiorowski, Mark Grondona, Helgi Ingolfsson, Shantenu Jha, Zvonko Kaiser, Dan Laney, Don Lipari, Joseph Koning, Dan Milroy, Claudia Misale, Chris Moussa, Tapasya Patki, Luc Peterson, Thomas R. W. Scogland, Becky Springmeyer, Michela Taufer, and Xiaohua Zhang

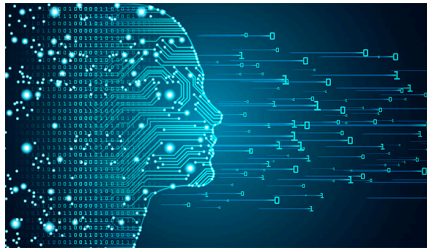
LLNL-PRES- 814589

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

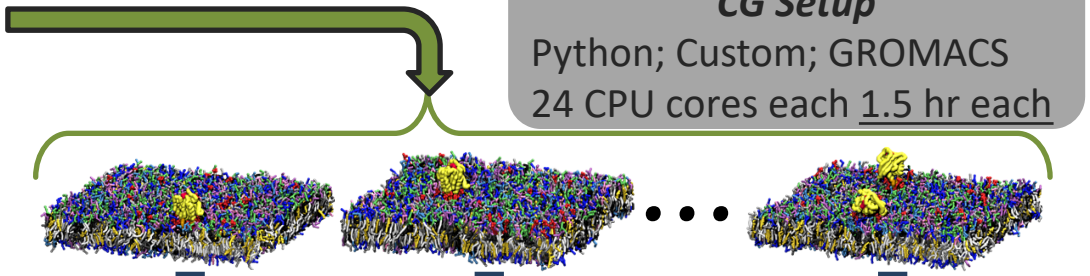
Sierra pre-exascale system is a wakeup call (MuMMI).



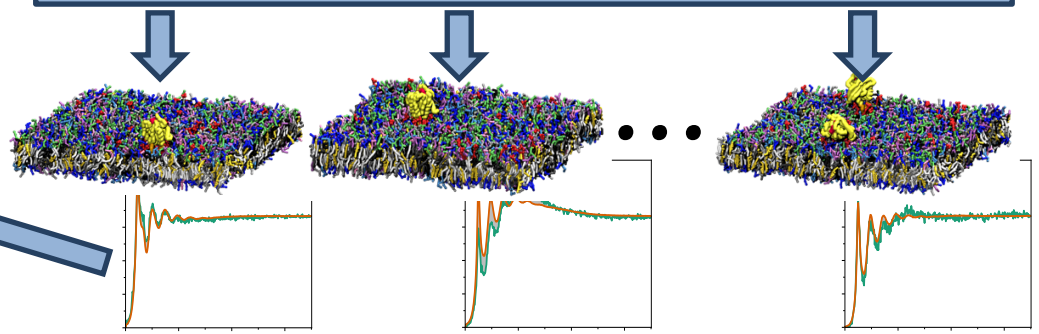


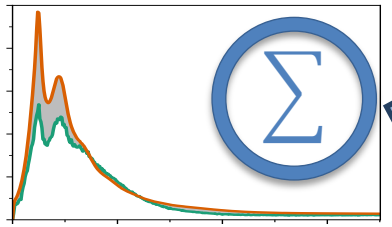
ML-based Selection
Python; ML frameworks; FAISS
24 CPU cores
>1000 decisions per minute

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



FIFO; real-time tracking & update





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

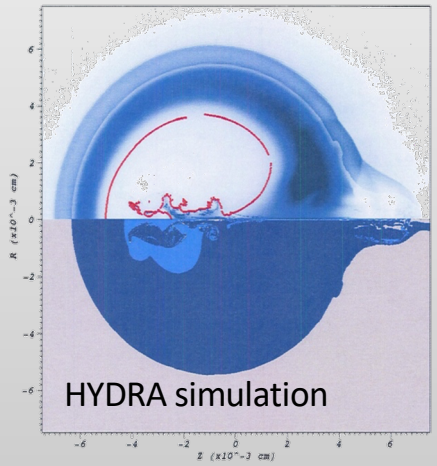
Analysis Aggreg. & Feedback
Python; Custom
24 CPU cores
120,000 reads per cycle

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

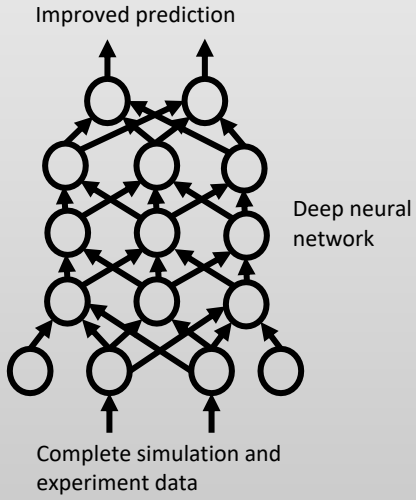
In situ CG Analysis
Python; Custom
3 CPU cores each
>2K frames per day

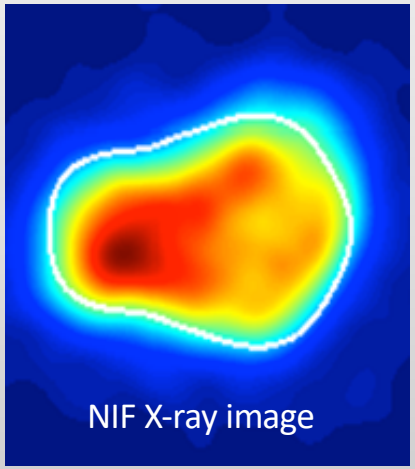
1 billion short-running jobs (MLSI)!

Traditional pillar
high-performance computing



New pillar
Machine learning to compare simulation and experiment

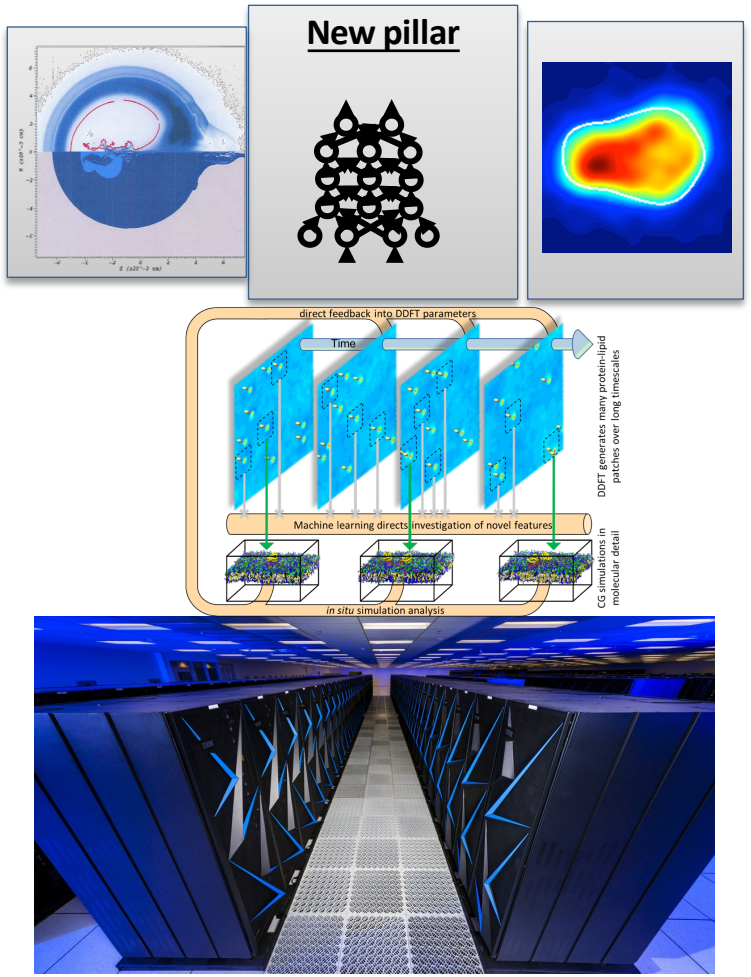




- Machine Learning Strategic Initiative (MLSI) – Initial target was to run 1 billion short-running jobs
- Similar needs for co-scheduling heterogenous components

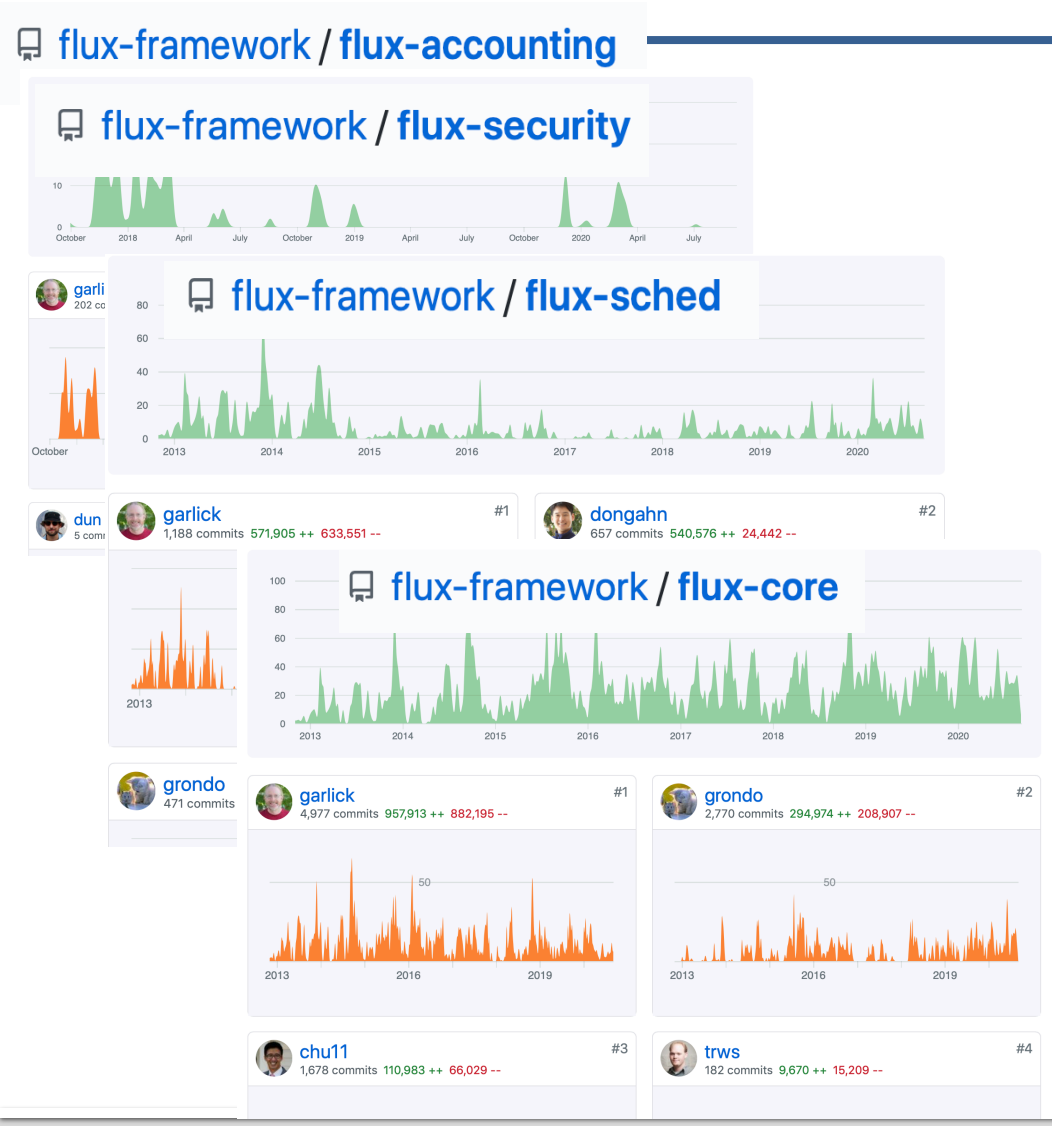
Machine learning, In-situ data analytics, and ensemble approaches demanded by UQ, V&V and testing are driving workflow changes.

Key challenges in emerging workflow scheduling include…



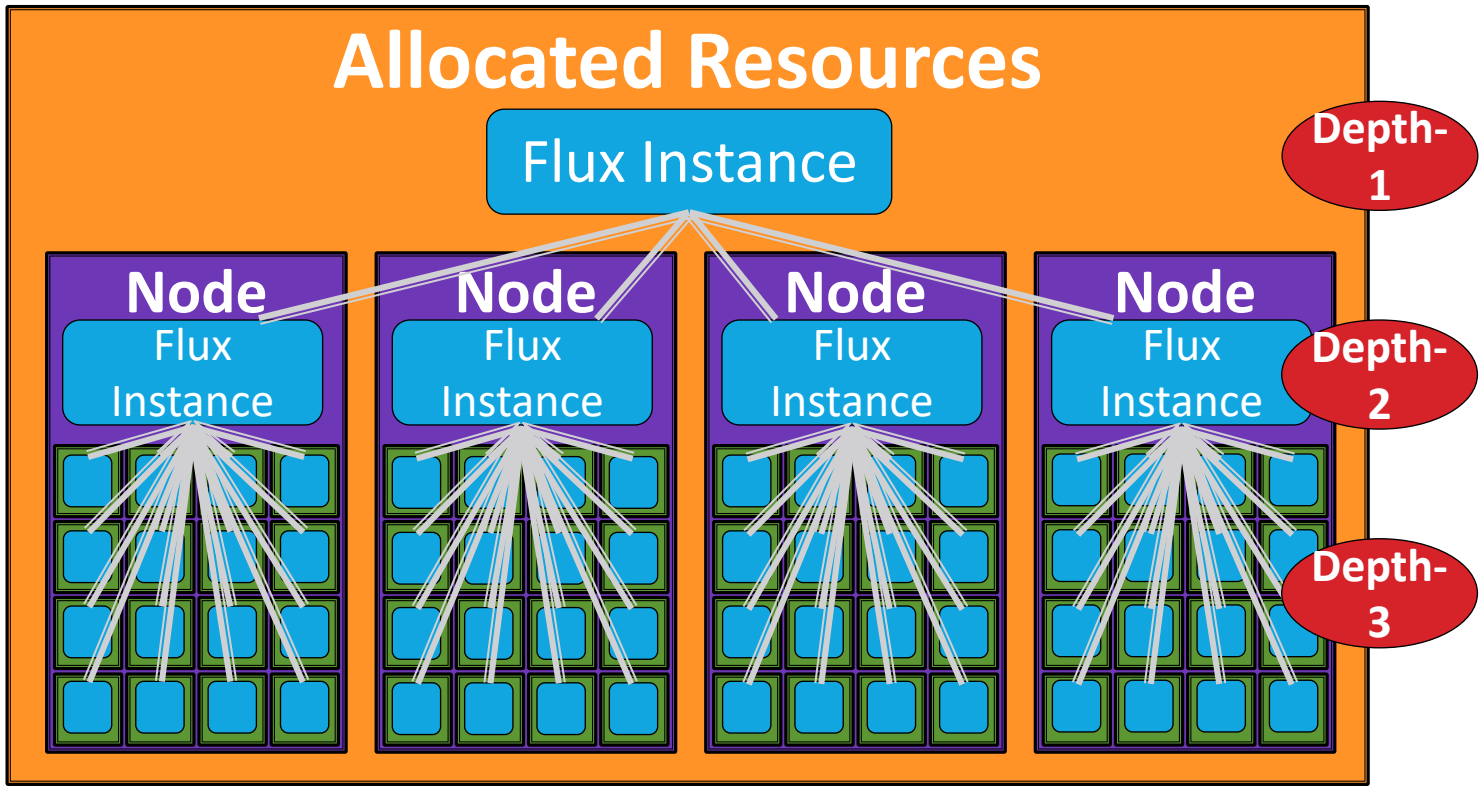
- Co-scheduling challenge
- Job throughput challenge
- Job communication/coordination challenge
- Portability challenge

flux at a glance



- Open-source project in active development at flux-framework GitHub organization
  - Composed of multiple projects: flux-core, -sched, -security, -accounting etc
  - Over 15 contributors including some of the principal engineers behind SLURM
- Single-user and multi-user (a.k.a. system instance) modes
  - Single-user mode has been used in production for 3 years
  - Multi-user mode is having its debut on LLNL's Linux clusters
- Plan of record for LLNL's El Capitan exascale system

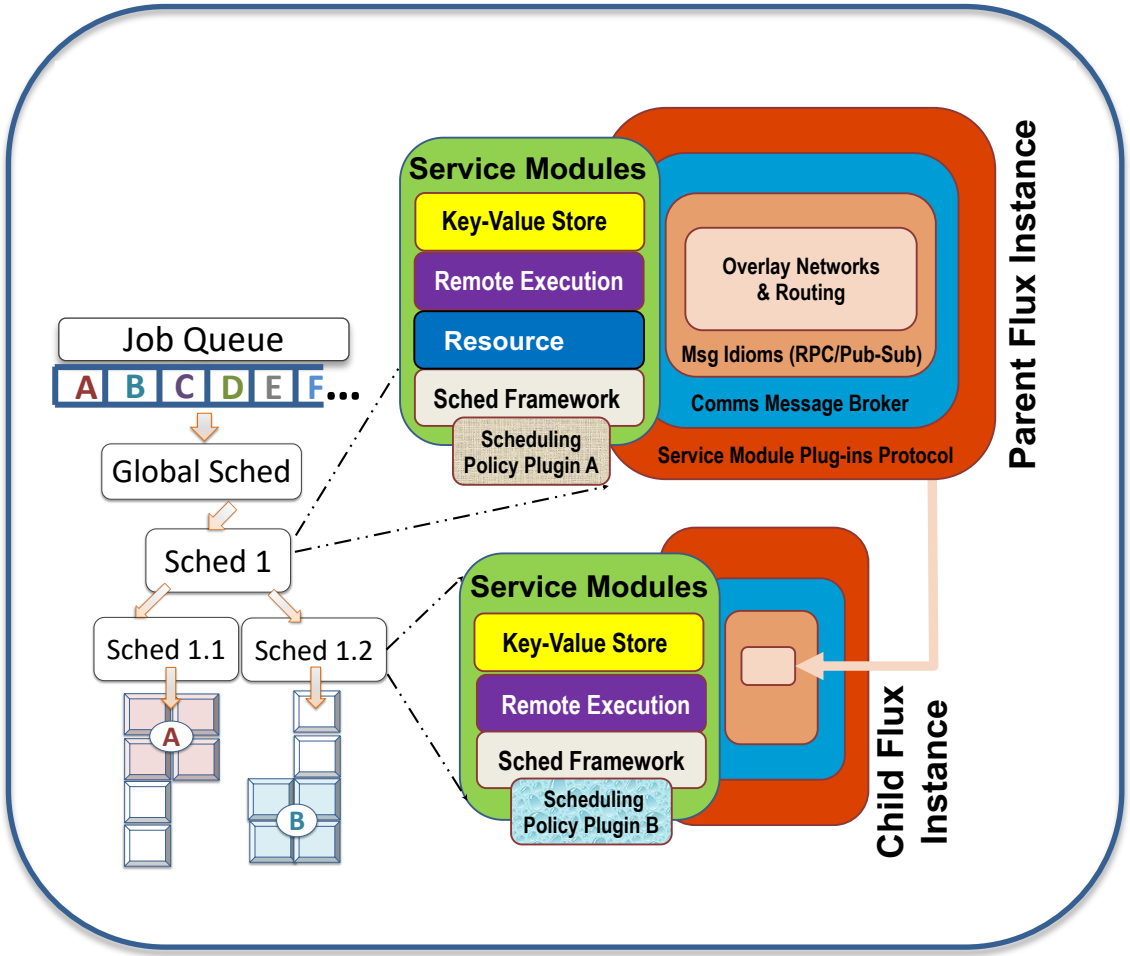
Flux offers a new scheduling model to meet modern-day workflow challenges.



- Native support for seamless nesting
  - Users don't need to wait for the facility to deploy Flux as the system resource manager and scheduler to benefit from it!
- Your personalized Flux instances provide ample opportunities for you to tailor Flux to your unique resource and job management needs.
- Rich and well-defined interfaces facilitate easy software composition.

Our "Generalized Multi-level Model" is designed specifically to meet modern-day workflow needs.

Flux is architected to seamlessly provide our generalized multi-level scheduling model.



Flux framework

Techniques

Scheduler Specialization

Scheduler Parallelism

Rich API set

Consistent API set

Challenges

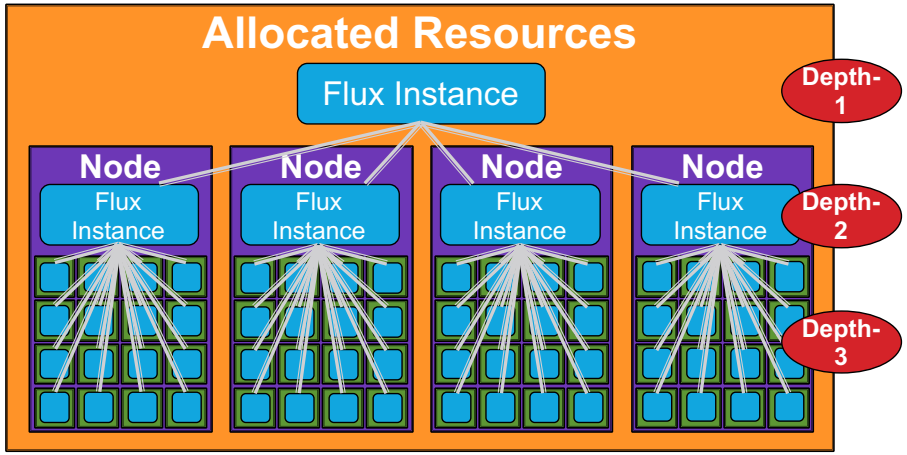
- Co-scheduling challenge
- Job throughput challenge
- Job communication/coordination challenge
- Portability challenge

Scheduler specialization solves the co-scheduling challenge.

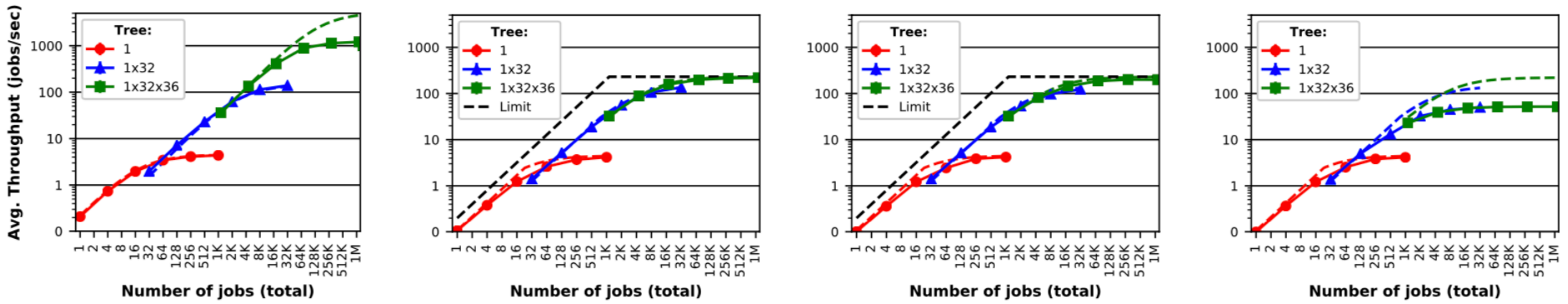
- Traditional approach
  - A single site-wide policy being enforced for all jobs
  - No support for user-level scheduling with distinct policies
- Flux enables both system- and user-level scheduling ***under the same common interfaces***.
- Give users freedom to adapt their scheduler instance(s)
  - Instance owners can choose predefined policies different from system-level policies.
    - Queuing policies (e.g., FCFS or backfilling-capable policies) and other parameters (queue depth, reservation depth etc) to allow for trading-off between performance and effectiveness
    - Resource match polices
    - Policies to control the flux instance hierarchy topology
  - Create their own policy plug-in

```
$ salloc -N4 -ppdebug
salloc: Granted job allocation 5620626
$ export FLUXION_QMANAGER_OPTIONS=queue-policy=easy
$ srun -N ${SLURM_NNODES} 4 -n ${SLURM_NNODES} --pty --mpi=none flux start
```

Scheduler parallelism solves the throughput challenge.



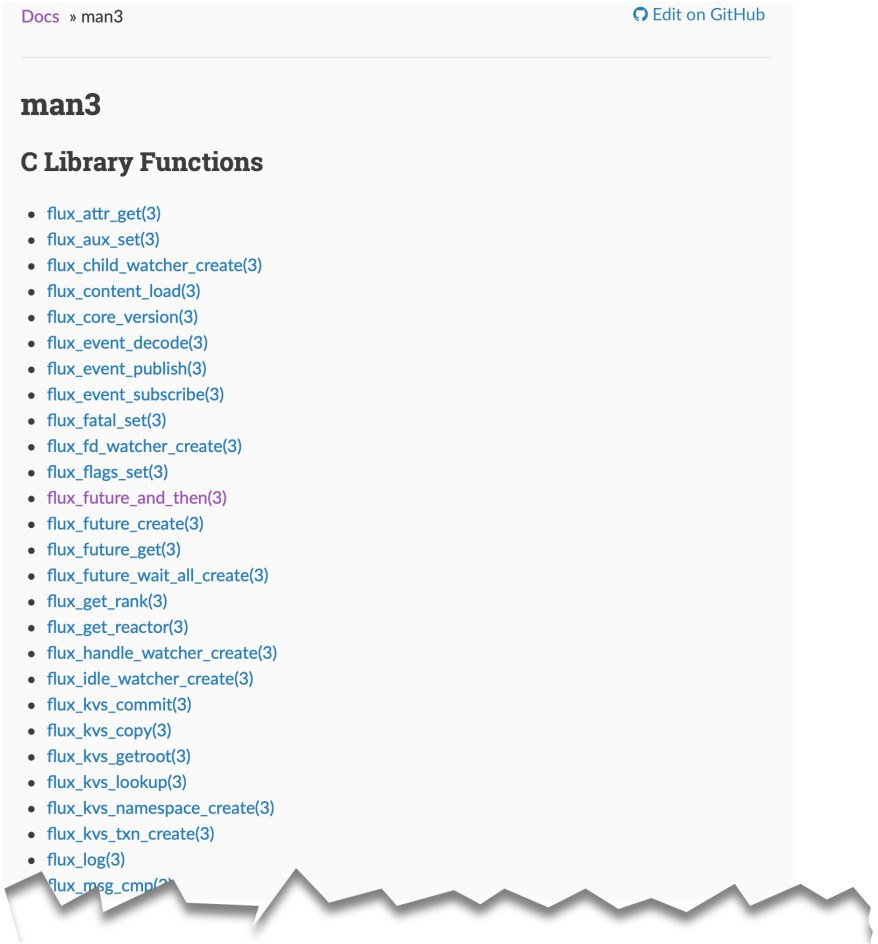
- The centralized model is fundamentally limited.
- Hierarchical design facilitates scheduler parallelism.
- Deepening the scheduler hierarchy allows for higher levels of scheduler parallelism



(a) Sleep 0 (b) Sleep 5 (c) Firestarter (d) Stream

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-reply, and send-recv patterns
- High-level services
  - Key-value store (KVS) API
  - Job API (submit, wait, state change notification, etc)

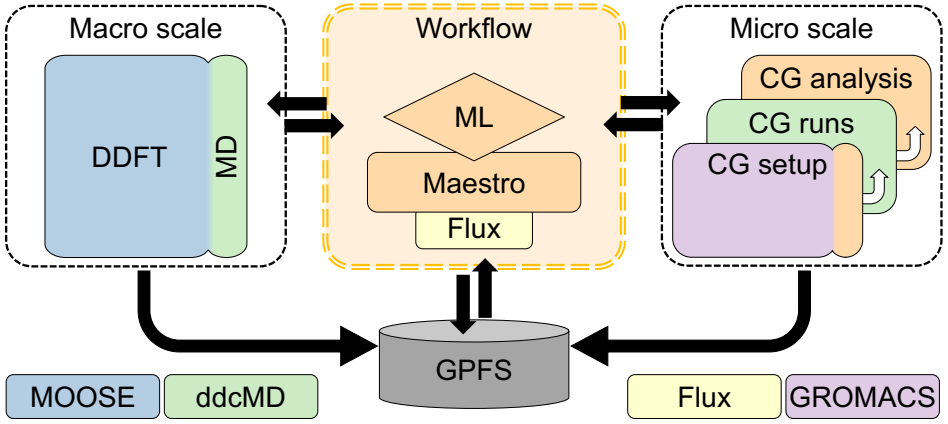


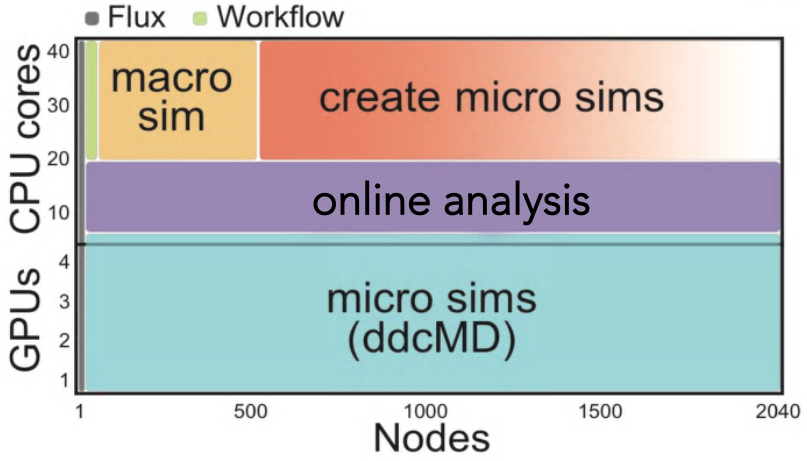
https://flux-framework.readthedocs.io/projects/flux-core/en/latest/index.html

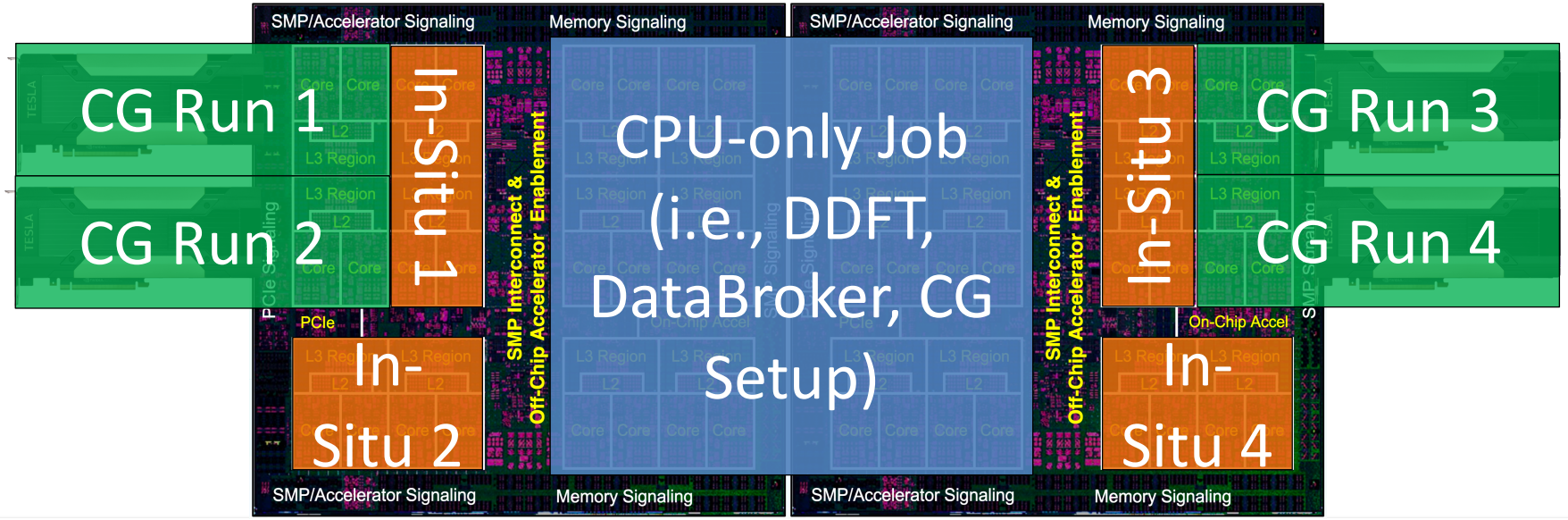
A consistent API set facilitates high portability.

- 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)

MuMMI: Scheduler specialization addresses co-scheduling and throughput challenges on Sierra.







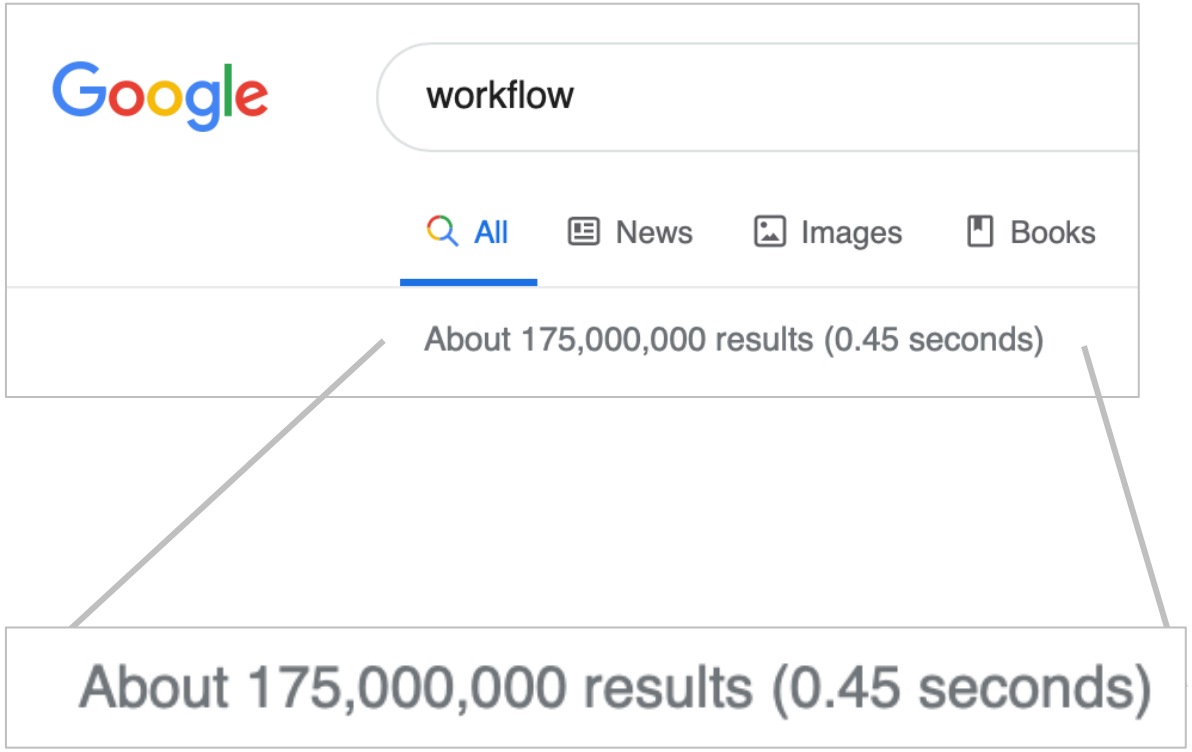
- Integrate Flux into Maestro workflow manager to *enqueue* and *dequeue* jobs
  - ML dynamically determines "most interesting" jobs
- Specialize the policy to be non-node-exclusive scheduling for co-scheduling
- At least 5 different logically separate jobs, each with CPU and/or GPU requirements on every node
- Handle the volume of jobs using two-level hierarchy

Flux is uniquely enabling many modern scientific workflows including COVID19 problems.

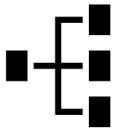
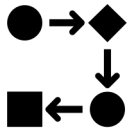
- **MuMMI workflow**
  - Successively ran last few years
  - Won the SC19 best paper award for its approach that enable a new genre of cancer research
  - Recently updated their workflow on the latest Flux to gear up for runs on ORNL Summit and others
- **LLNL's UQP workflow**
  - Refactored its ensemble manager layer into a standard-alone component by building on Flux
  - Produced Themis "component" targeting UQ, V&V, parameter and sensitivity study workflows
  - Demonstrated the scheduling unification provides significant benefits (performance and complexity)
- **MLSI Merlin workflow**
  - Addressed throughput and portability needs
  - Demonstrated 100 million short running jobs
- **COVID19 scenario modeling workflow**
  - Support what-if scenarios on COVID19 spread
  - "With flux we can model one scenario w/ UQ for the entire country in ~5 minutes on a few [lassen] nodes: near real-time feedback"
  - *Portably* ran LLNL, ORNL and NERSC machines
- **COVID19 drug design end-to-end workflow**
  - Combine ATOM (ML-based drug molecule design framework) with large docking simulations (ConveyorLC) to expediently produce COVID 19 drug compounds for further clinical testing.
  - At 256 nodes, Flux improved resource utilization of ConveyorLC by a factor of 2
  - Making it easy to couple ATOM with ConveyorLC
  - Putting together a COVID19 Gordon Bell submission for SC20

Use the right tool for the job

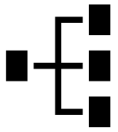
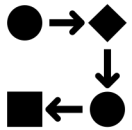
"If that saw doesn't work, try this hammer"



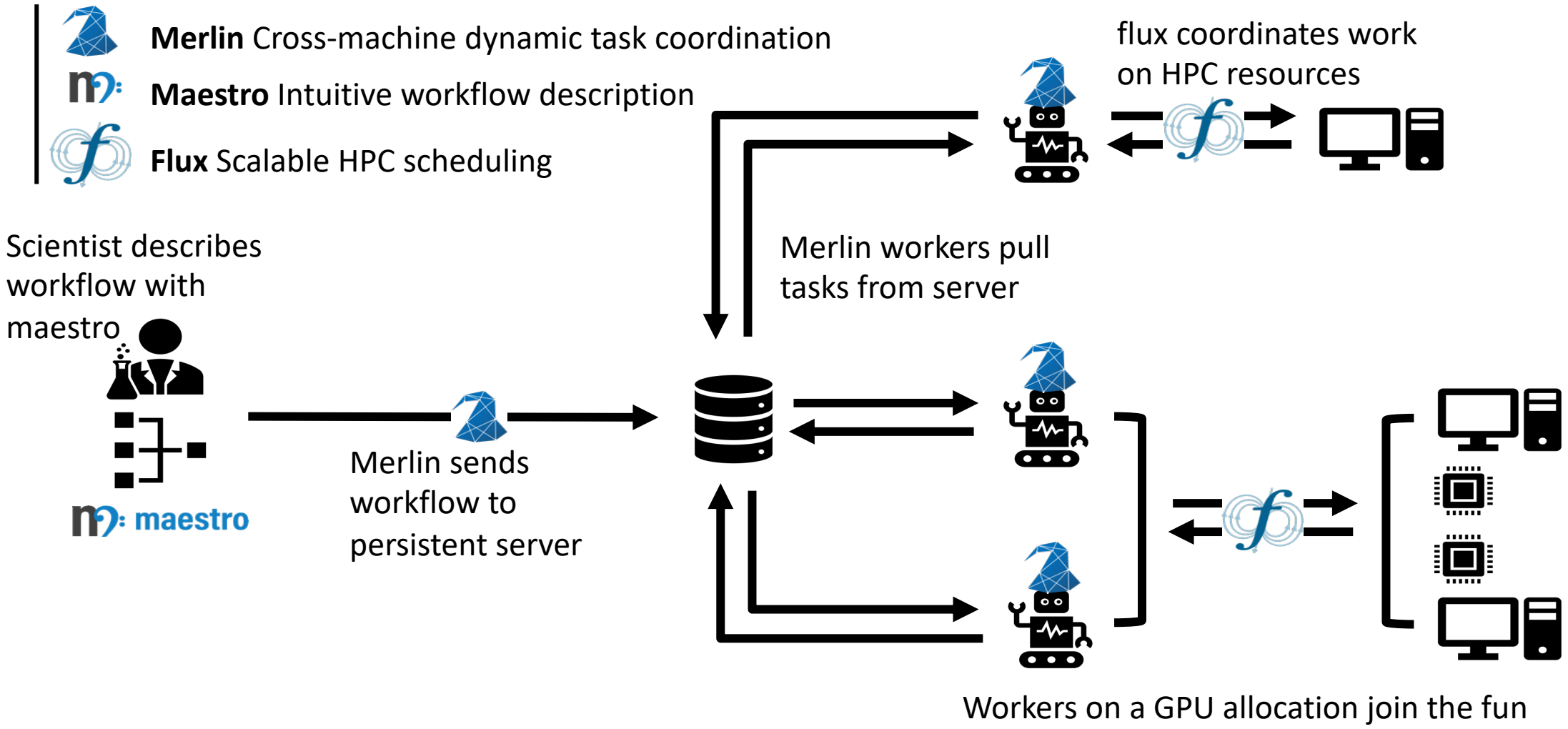
How can LLNL's tools cooperate?

Components		UQPipeline	sina	conduit	codes
Description		UQPipeline	maestro	
Orchestration		UQPipeline	maestro	merlin
Resource Allocation		lsf	flux	slurm

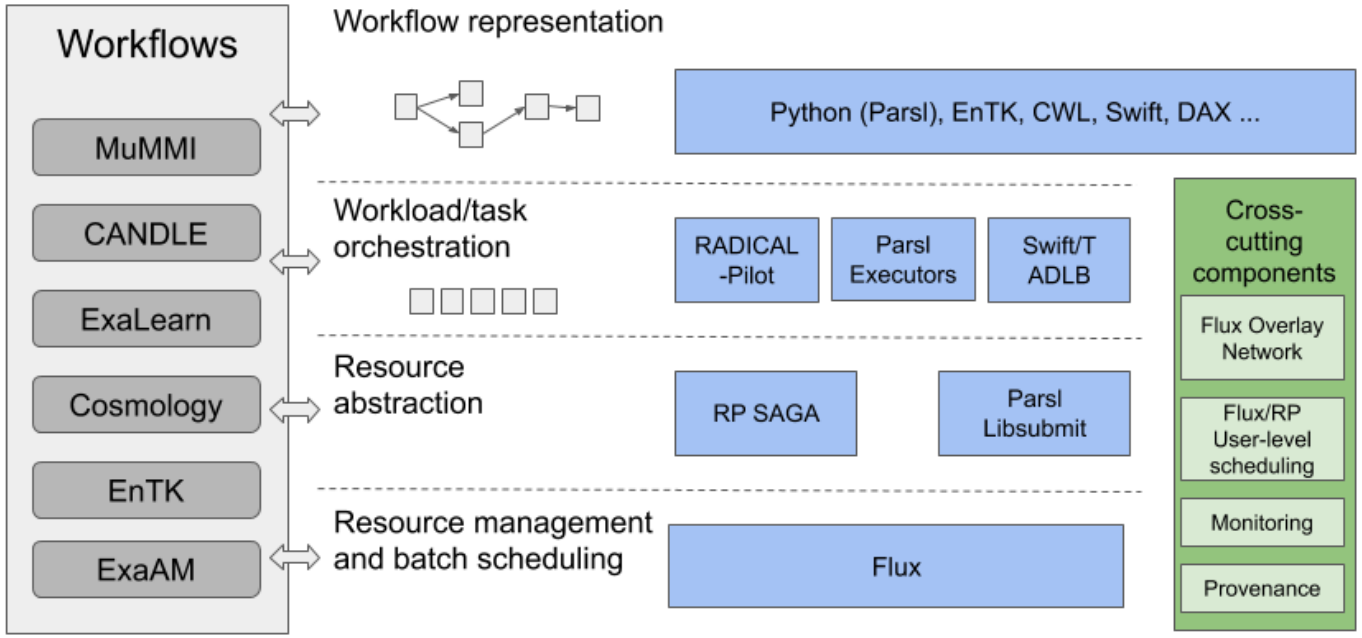
Ideally, we want to be able to pick and compose functionalities from multiple tools.

Component		UQPipeline	sina	conduit	codes
Description		UQPipeline	maestro		
Orchestration		UQPipeline	maestro	merlin	
Resource Allocation		lsf	flux	slurm	

LLNL teams have been collaborating with a vision to create a "plug and play" workflow ecosystem.



Flux is a part of ExaWorks to define APIs and methodology to create a plug-and-play component-based workflow ecosystem.

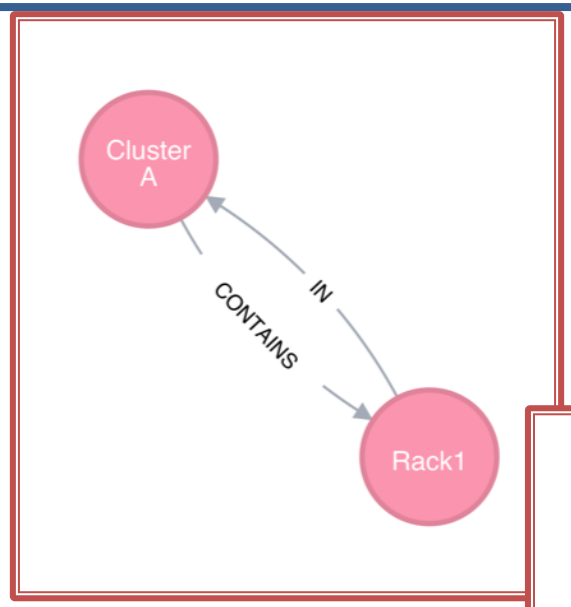


ECP ExaWorks project

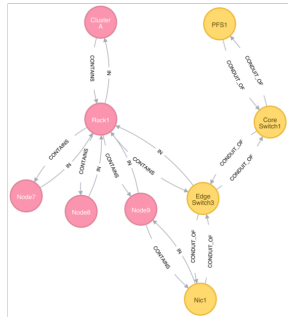
- New ECP project that started last month
- Three representative workflow and resource management software systems.
  - Flux (LLNL), Parsl (ANL) and RadicalPilot (BNL)
- Co-design requisite interfaces (vertical and horizontal) while engaging with a wider range of HPC community.
- Harden our reference implementation (i.e., ExaWorks toolbox) written on top of these interfaces and deploy it on exascale systems.

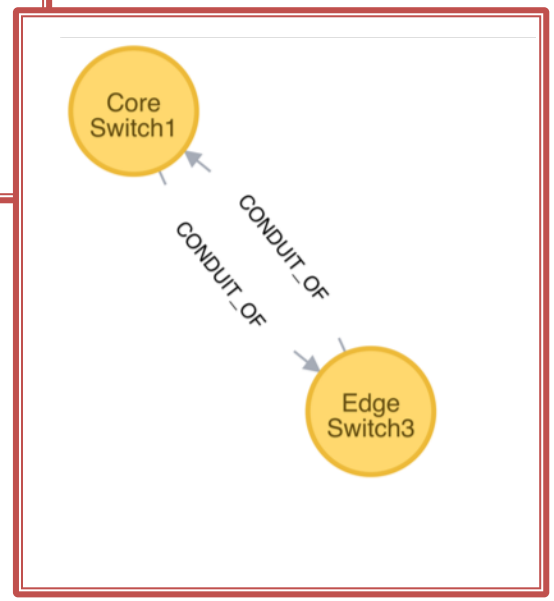
Fluxion uses a graph-based resource data model to represent schedulable resources and their relationships.

- The traditional resource data models are largely ineffective to cope with the resource challenge
  - Designed when the systems are much simpler
  - Node-centric models
- A graph consists of a set of vertices and edges
- Highly composable to support a graph with arbitrary complexity
- The scheduler remains to be a highly generic graph code.



Containment subsystem

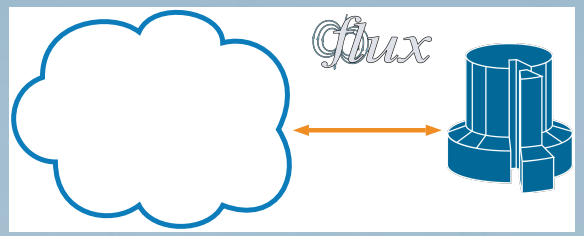




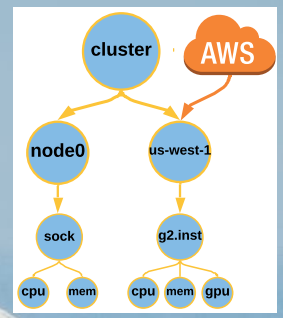
Network connectivity subsystem

Fluxion has proven effective in scheduling diverse resources such as exascale multi-tiered storage and cloud resources

- Co-designing of multi-tiered storage with HPE for El Cap is under way
- Starting a 3-way partnership with RedHat OpenShift and IBM T.J. Watson teams
  - Use Fluxion as the driver to define Kubernete's standard scheduler interface
  - IBM already developed KubeFlux prototype
- Position Flux for all four tenets of HPC+Cloud convergence
- Dan Milroy will present this topic at his SC20 state-of-the-practice talk (11/17/2020)



Loosely coupled cloud and HPC (in use today)



Hybrid/bursting to public cloud

HPC-in-cloud: HPC RJMS in K8s; workflows scale everywhere

Cloud-in-HPC: K8s in RJMS

Flux is readily available on Cori!

- Loading Modules
  - `module use /global/common/software/flux/modulefiles`
  - `module load czmq flux`
- Launching Flux Session
  - `salloc --nodes=2 --time=60 --qos=interactive`
  - `srun --pty -N2 -n2 flux start`
  - Note: `--pty` and `--qos=interactive` only necessary for interactive sessions
- Resources
  - Docs: https://flux-framework.readthedocs.io/en/latest
  - “mailing list” : https://github.com/flux-framework/flux-core/discussions