Agenda

Time	Topic	Presenters
01:00PM – 01:30PM	Using Flux to overcome scheduling challenges of exascale workflows	Dong H. Ahn
01:30PM – 01:45PM	Fully Hierarchical Scheduling and Job Interfaces for Large Ensembles of Uncertainty Quantification (UQ) Tasks	James Corbett
01:45PM – 2:00PM	Flux's job and resource model	Stephen Herbein
2:00PM – 2:15PM	Play the full game!	Dan Milroy
2:15PM – 3:00PM	Example/Use-case driven hands-on session	Herbein/Tapasya
03:00PM – 3:30PM	Bring Your Own Cluster (BYOC): Testing Flux on your clusters	All

Note: All tutorial slides and demo content are available on Github: https://github.com/flux-framework/Tutorials/tree/master/2021-ECP

Using Flux to overcome scheduling challenges of exascale workflows

Flux Tutorial at ECP Annual Meeting, April 14, 2020

Dong H. Ahn

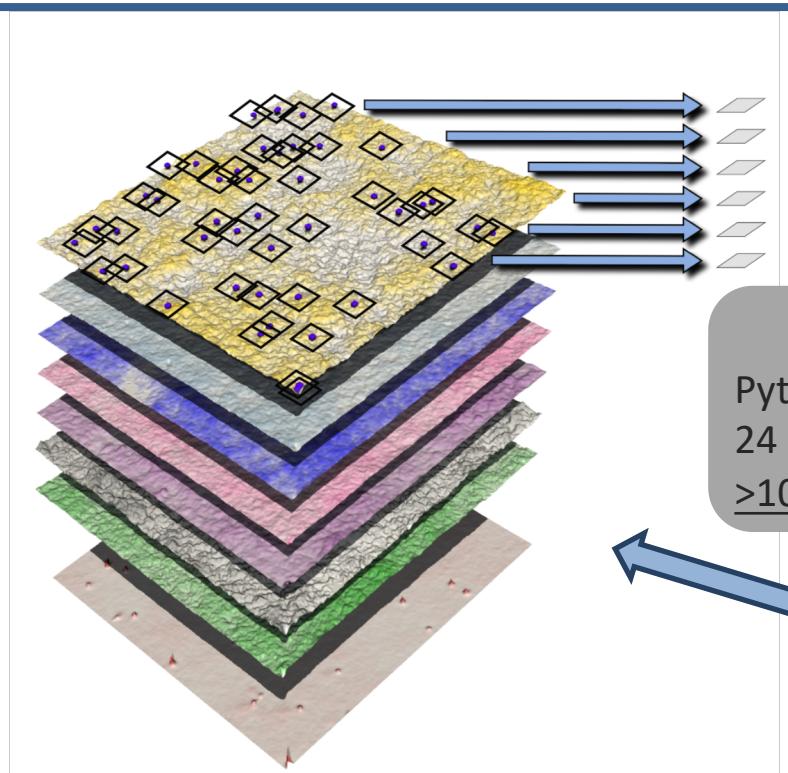
Acknowledgment: Ned Bass, Subhendu Behera, Albert Chu, Kyle Chard, Brandon Cook, James Corbett, Ryan Day, Franc Di Natalie, David Domyancic, Phil Eckert, Stephen Herbein, Jim Garlick, Elsa Gonsiorowski, Mark Grondona, Helgi Ingolfsson, Shantenu Jha, Zvonko Kaiser, Dan Laney, Carlos Eduardo Gutierrez, Edgar Leon, Don Lipari, Daniel Milroy, Joseph Koning, Claudia Misale, Chris Moussa, Frank Muller, Tapasya Patki, Yoonho Park, Luc Peterson, Barry Rountree, Thomas R. W. Scogland, Becky Springmeyer, Michela Taufer, Jae-Seung Yeom, Veronica Vergara and Xiaohua Zhang

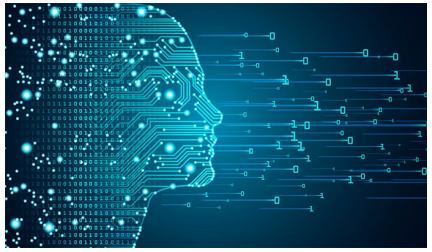
LLNL-PRES-821226

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

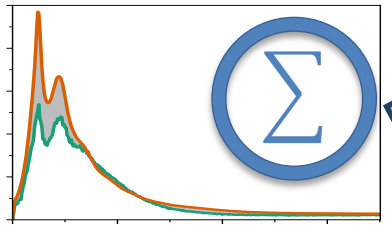
Lawrence Livermore National Laboratory

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

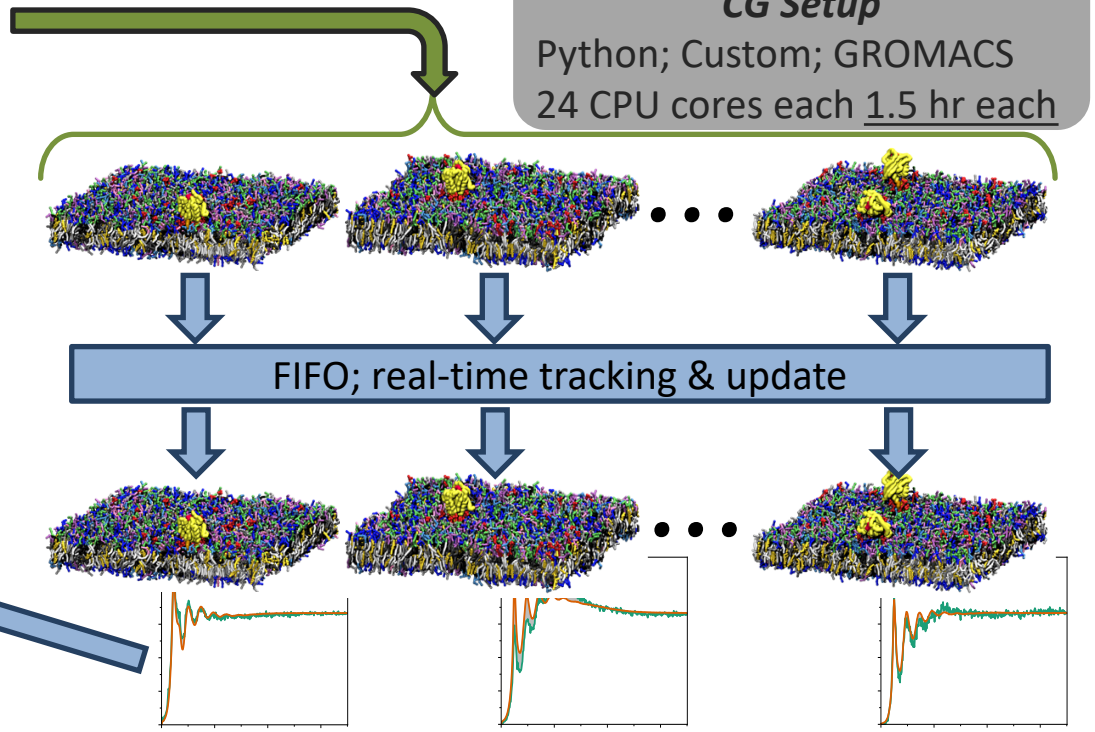




ML-based Selection
Python; ML frameworks; FAISS
24 CPU cores
>1000 decisions per minute



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



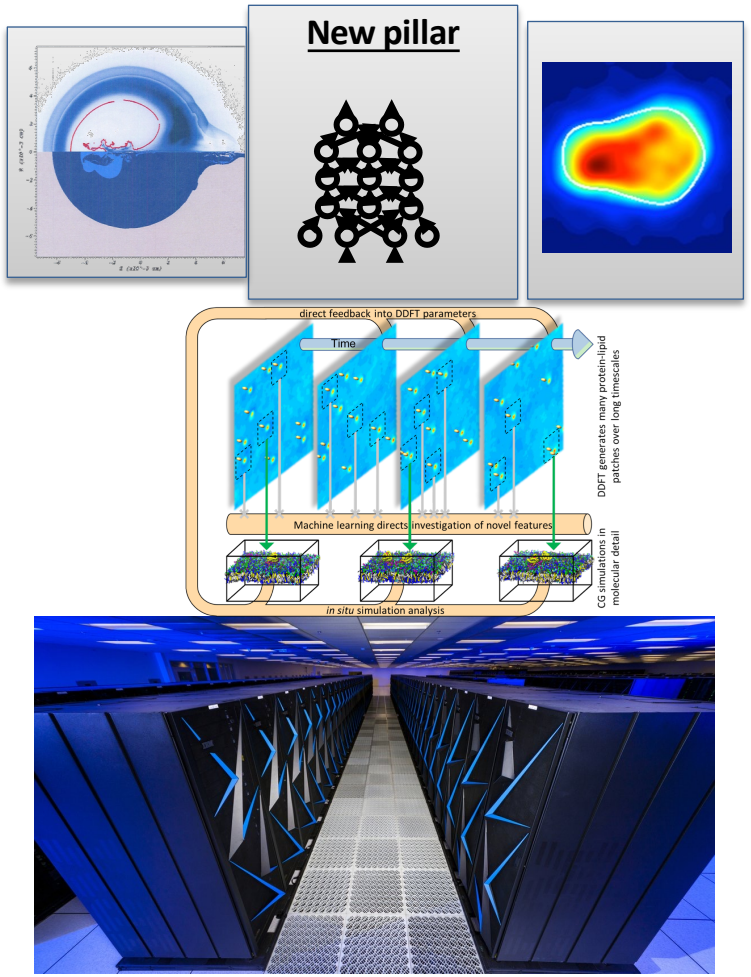
Single Macro Simulation
C++ (with MPI); MOOSE; ddcMD
24 CPU cores/node; 2400 MPI tasks
242 GB per day

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

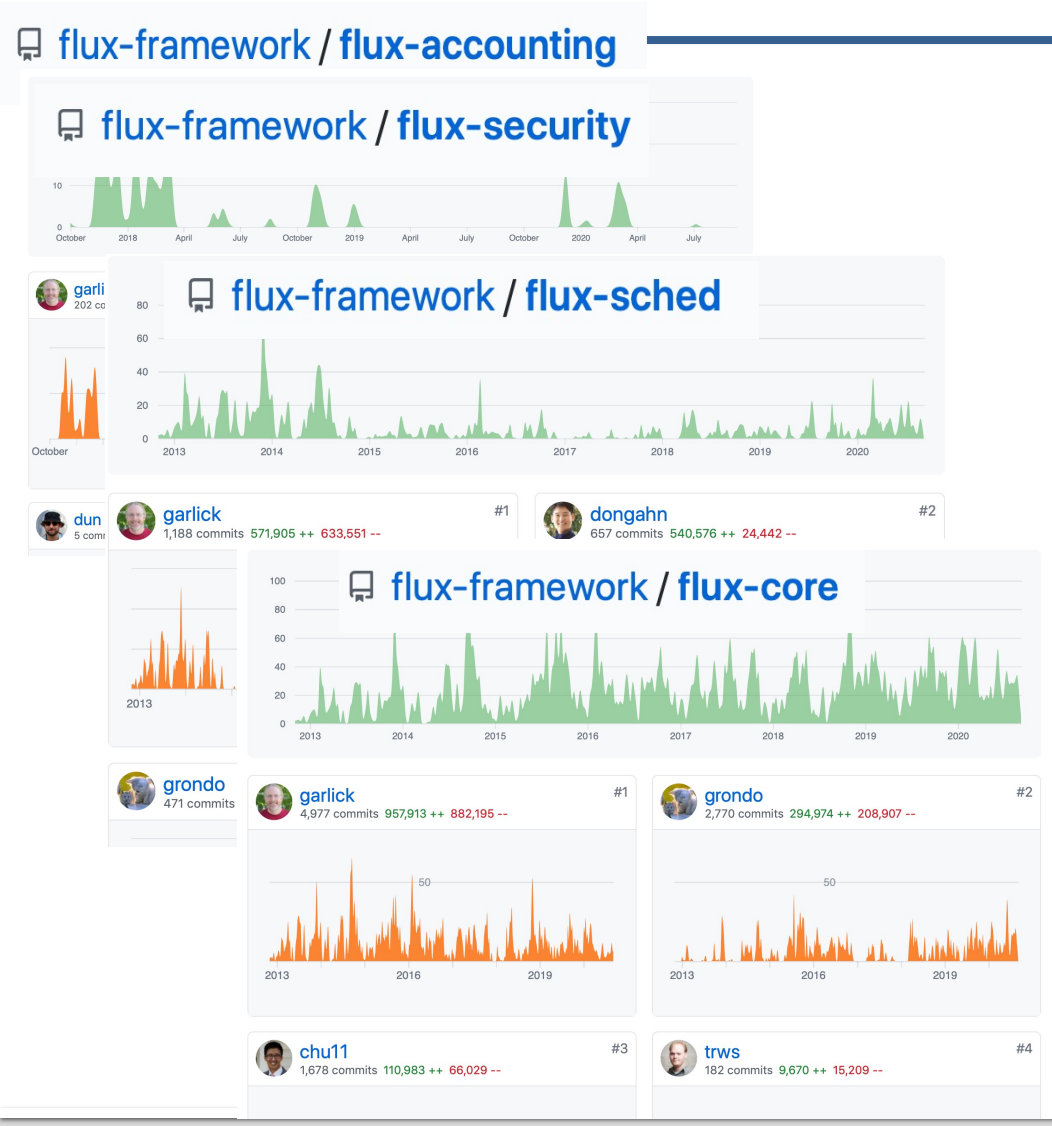
In situ CG Analysis
Python; Custom
3 CPU cores each
>2K frames per day

Key challenges in emerging workflow scheduling include…



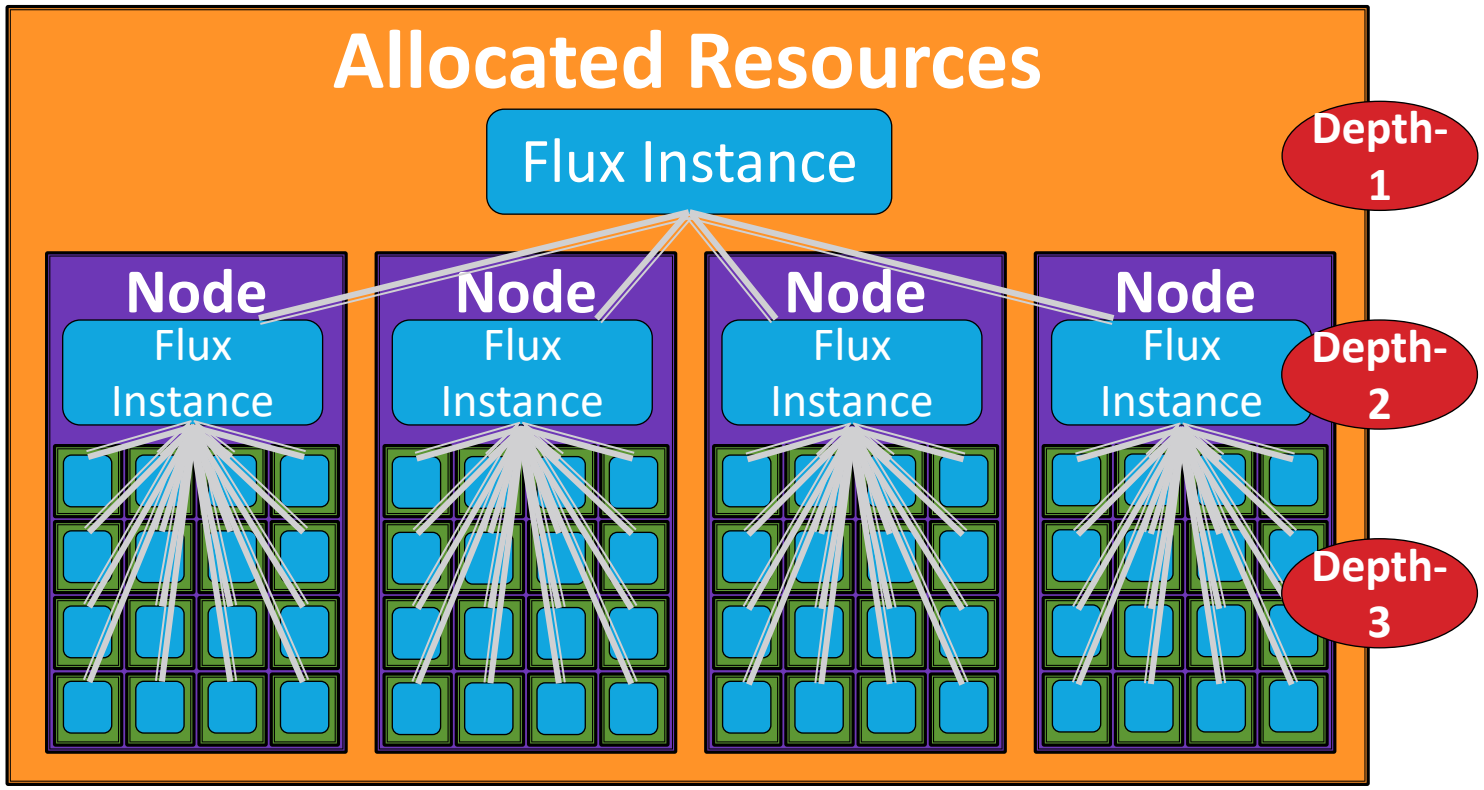
- Co-scheduling challenge
- Job throughput challenge
- Job communication/coordination challenge
- Portability challenge

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

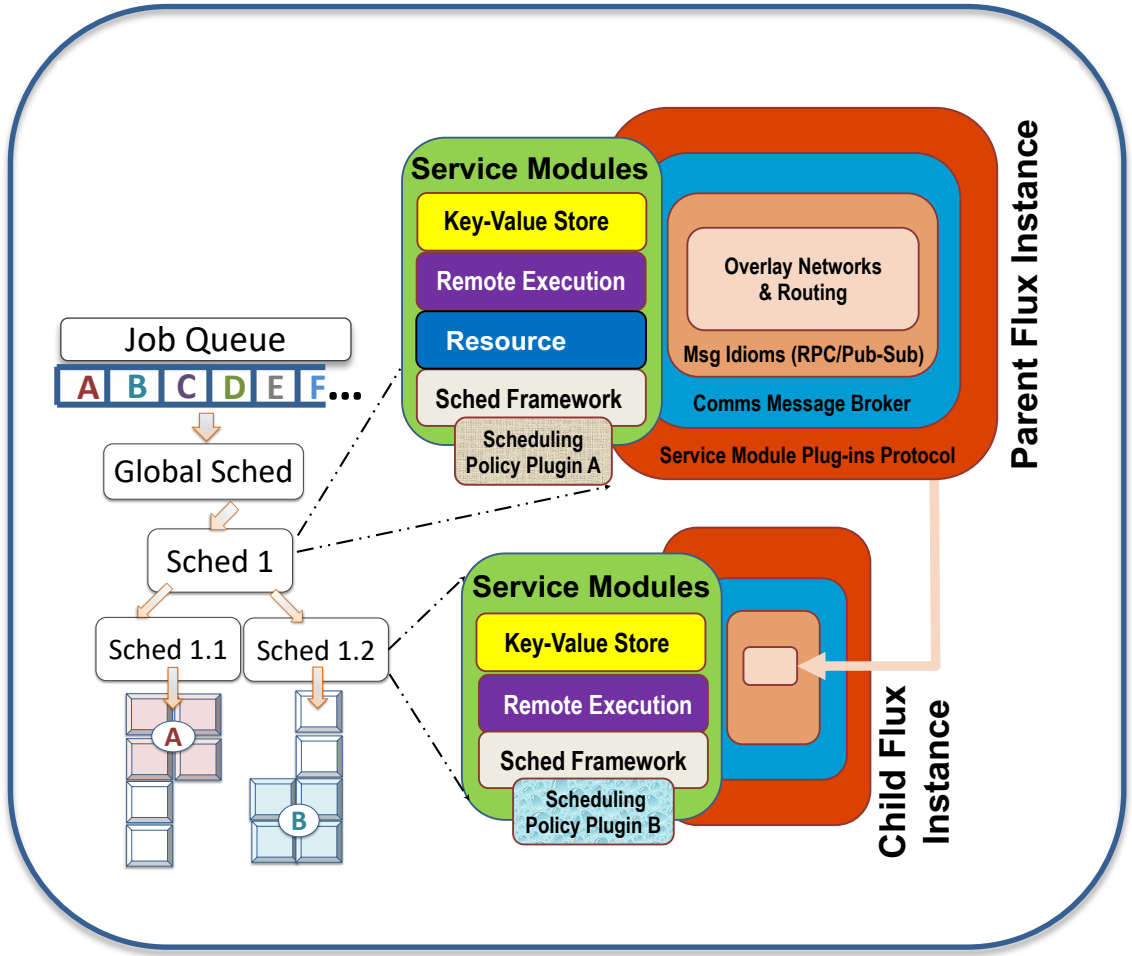
Flux offers a new scheduling model to meet modern-day workflow challenges.



- 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 "fully hierarchical model" is designed specifically to meet modern-day workflow needs.

Flux is architected to embody our fully hierarchical scheduling model seamlessly.



Techniques

Scheduler Specialization

Scheduler Parallelism

Rich API set

Consistent API set

Challenges

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

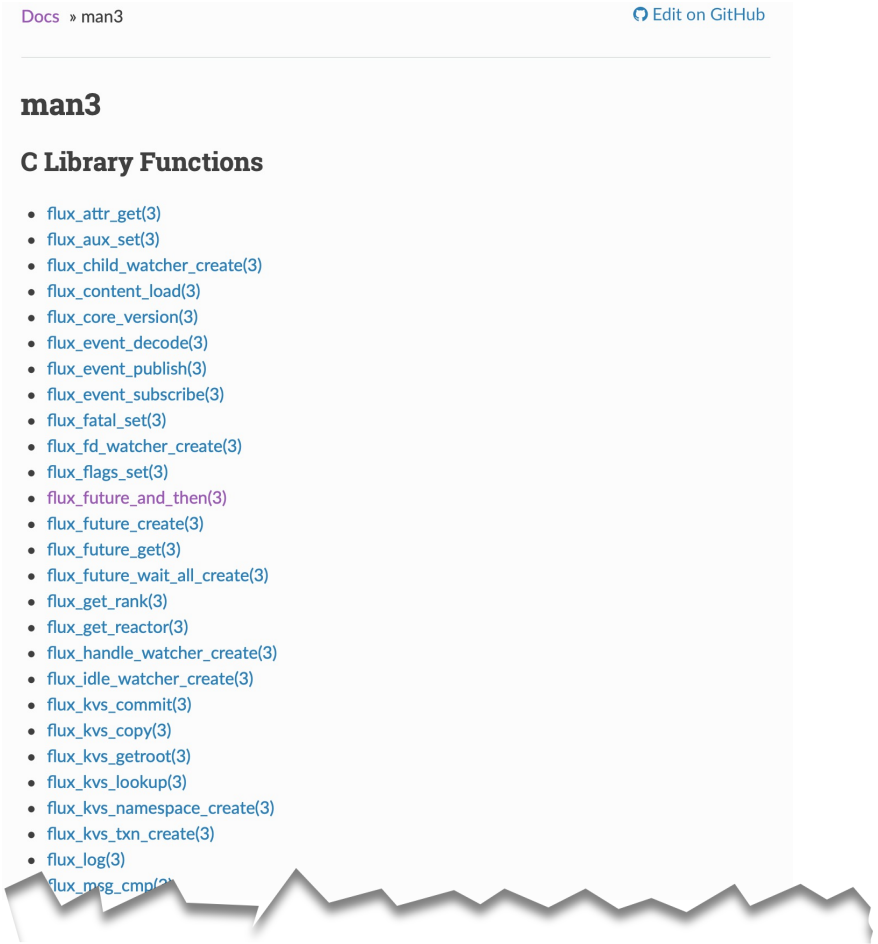
Scheduler specialization solves the co-scheduling challenge.

- 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
```

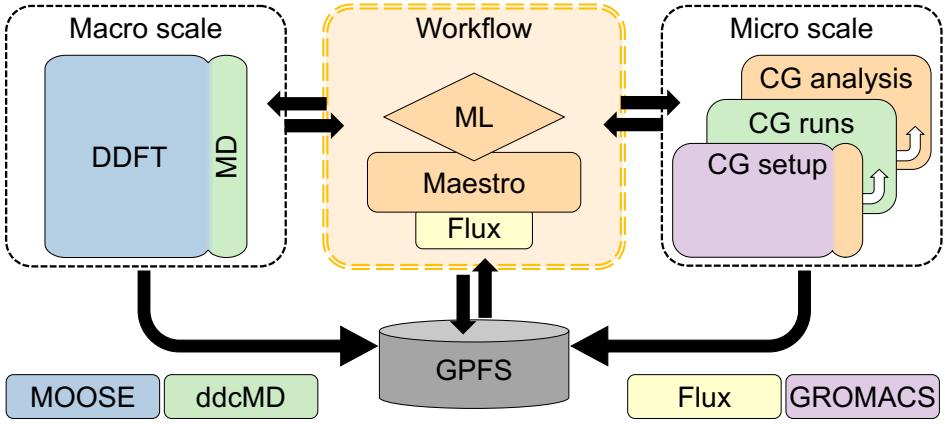
A rich set of well-defined APIs enables easy job coordination and communication.

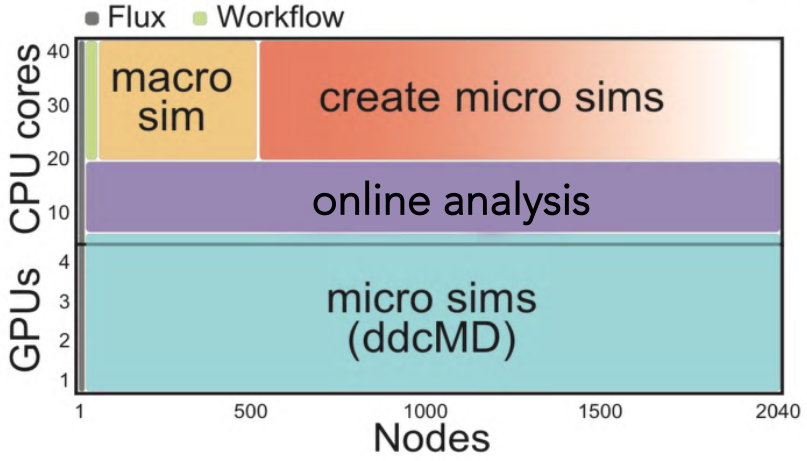
- Jobs in ensemble-based simulations often require close coordination and communication with the scheduler as well as among them.
  - Traditional CLI-based approach can be slow and cumbersome.
  - Ad hoc approaches (e.g., many empty files) can lead to many side-effects.
- Flux provides well-known communication 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)
- Flux's APIs are consistent across different platforms

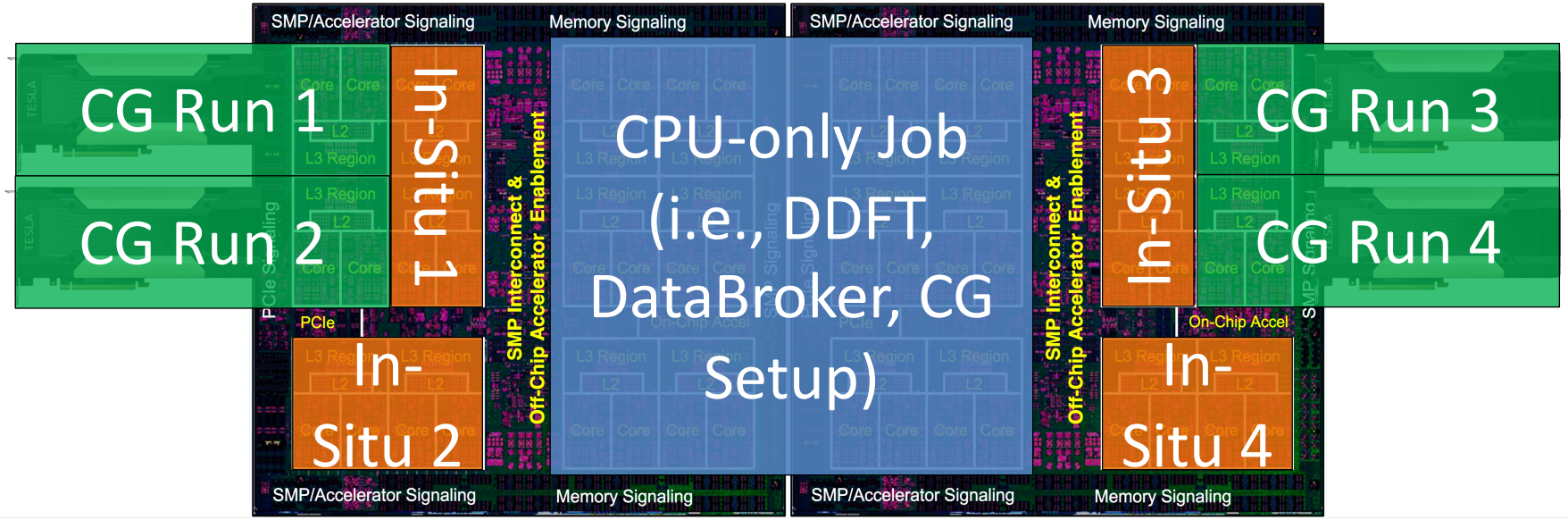


https://flux-framework.readthedocs.io/projects/flux-core/en/latest/index.html

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

MuMMI has continuously adopted new features of Flux

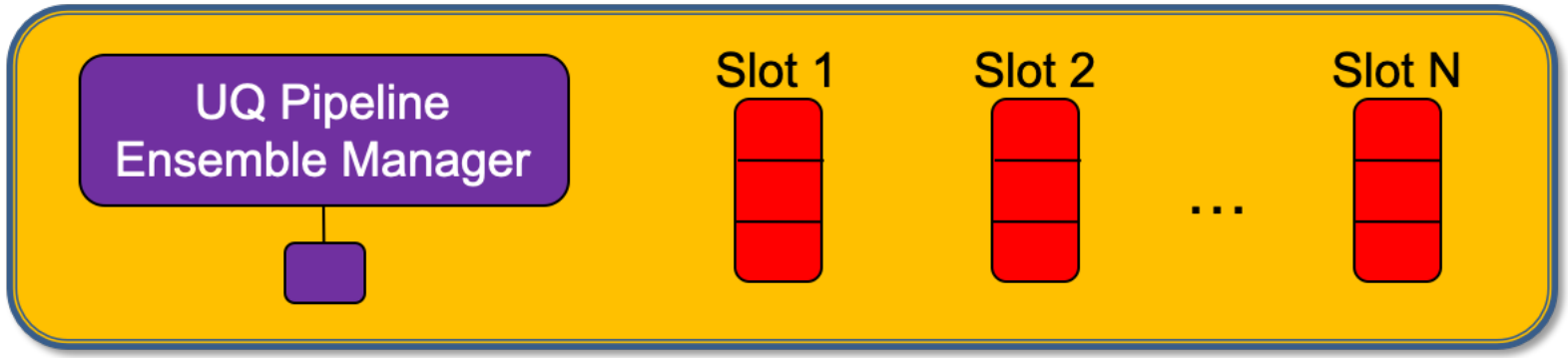
- Flux has expanded its resource management and scheduling feature set
  - New remote execution service
  - Fluxion, a graph-based scheduler
  - Python APIs and our command line interface set
- The MuMMI team was one of our co-design partners
  - Get and incorporate the feedback on our new functionalities and APIs set
  - Much of this churn has been hidden under Maestro
- Maestro continuously updated its Flux Adapter
  - https://github.com/LLNL/maestrowf/releases/tag/v1.1.7
  - https://github.com/LLNL/maestrowf/releases/tag/v1.1.6
- Major challenges arose when MuMMI was ported ORNL Summit
  - Their local computing environment is significantly different!
  - Worked with OCLF support staff to test and install Flux
  - Updated our documentation (https://flux-framework.readthedocs.io/en/latest/coral.html)

The MuMMI work won the SC19 best paper award!

UQPipeline: Seeding an ensemble study has faced well-known problems with no easy solution

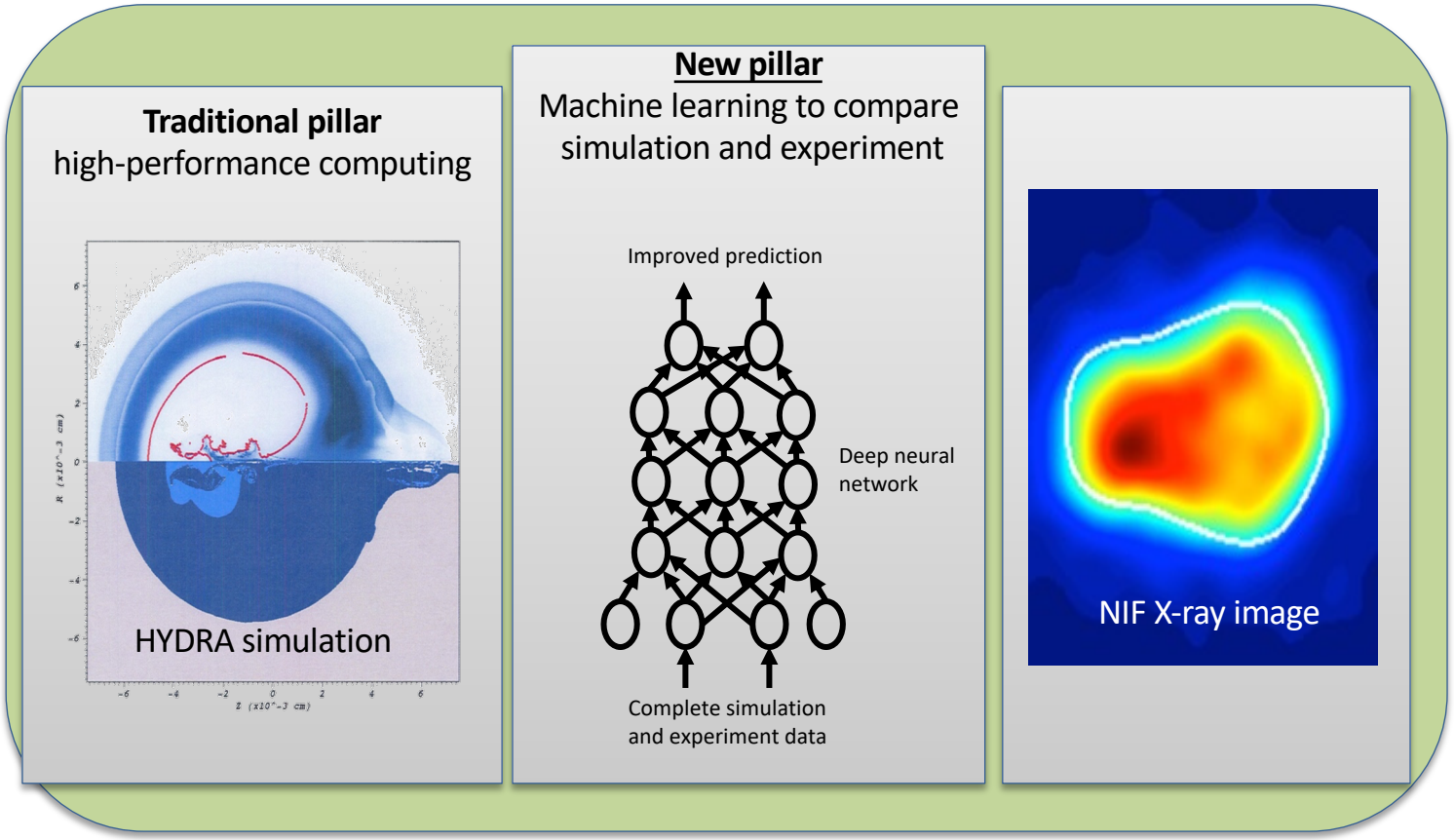
- Using scientist's batch script to seed a large ensemble would be ideal, but…
  - Impose limits on the number of concurrently running job
  - Getting a resource allocation consisting of N nodes is often much quicker than getting N resource allocations
- Tools like UQPipeline (UQP) can provide requisite throughput, but…
  - Arduous effort to convert their (often complex) batch scripts into the tool's interface
  - Must do this for N times for N different tools

```
#!/bin/sh
#
# A physicist's cannonical batch submission script

<Complex pre-processing logic begins>
...

flux mini run -n ${ntasks} -N ${nnodes} /usr/apps/bin/app1 input.py
flux mini run -n ${ntasks} -N ${nnodes} /usr/apps/bin/app1 input2.py

...
<Complex post processing logic ends>
```



James Corbett from the Themis team will discuss
our scheduling-unification approach and lessons-learned at 1:30PM

Flux underpins new AI methods to combine simulations with experiments.



- Machine Learning Strategic Initiative (MLSI): Initial target was to run 1 billion short-running jobs
- Similar needs as MuMMI for co-scheduling heterogenous components
- Developed Merlin which uses Flux to address throughput and portability
  - Merlin used by other workflows: Pf3d and COVID-19 modeling simulation ensemble

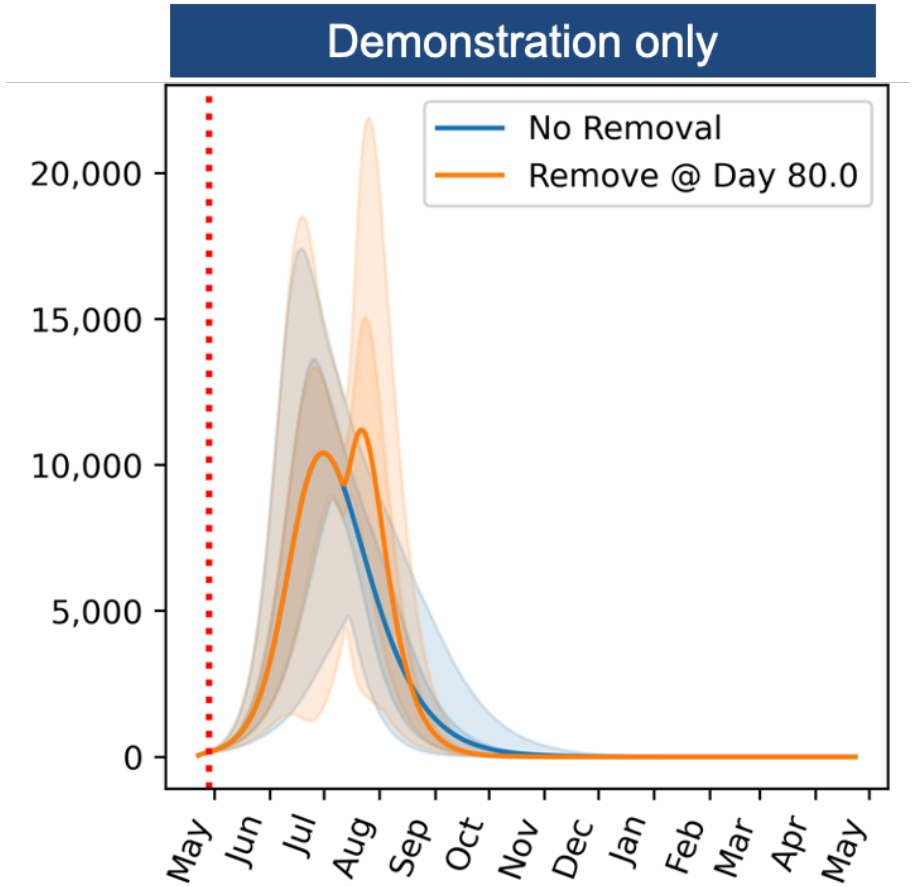
ELSEVIER
Future Generation Computer Systems
Volume 110, September 2020, Pages 202-213

Flux: Overcoming scheduling challenges for exascale workflows

Dong H. Ahn [a], Ned Bass [a], Albert Chu [a], Jim Garlick [a], Mark Grondona [a], Stephen Herbein [a], Helgi I. Ingólfsson [a], Joseph Koning [a], Tapasya Patki [a], Thomas R.W. Scogland [a], Becky Springmeyer [a], Michela Taufer [b]

Check out our new FGCS journal paper detailing our techniques and results! (https://www.sciencedirect.com/science/article/abs/pii/S0167739X19317169)

Flux has been providing workflow aid for COVID-19 scenario modeling workflow



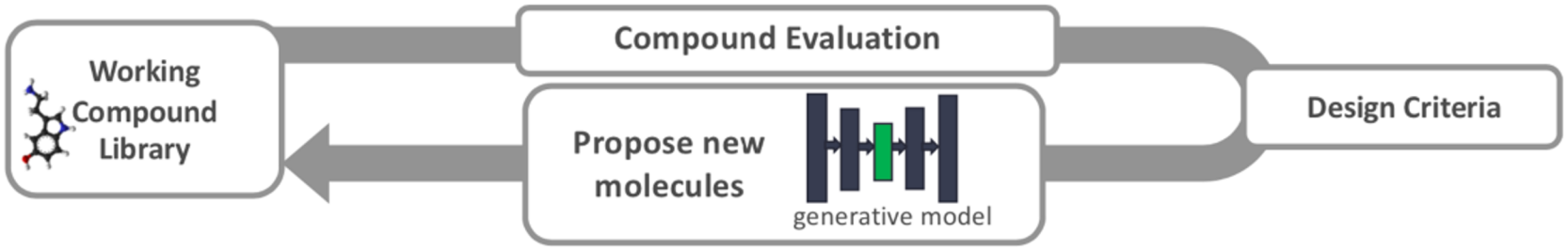
- Collaboration among LLNL, LANL and NERSC
- A large ensemble of EPICAST simulations are employed to model COVID-19 spread patterns
- Inform agencies like FEMA of the prediction
- If school is open only 2 days a week how differently will COVID-19 spread?
- Port Flux to ORNL Summit and NERSC CORI
- *"With flux, we can model one scenario with UQ for the entire country in ~5 minutes on a few [lassen] nodes: near real-time feedback."* – Luc Peterson

A Demo COVID-19 Scenario Modeling (Credited to Luc Peterson)

Flux has been providing workflow aid for fast ML-based COVID-19 drug molecule design workflow

- A COVID-19 drug molecule design multi-disciplinary team formed
  - Develop a new highly scalable end-to-end drug design workflow
  - Expediently produce potential COVID-19 drug molecules for the next stages of drug discovery
  - This team brought together multiple LLNL experts from multiple directorates including PLS, GS, CASC and LC
- LBANN (Livermore Big Artificial Neural Network Toolkit) researchers
  - Focus on developing a scalable training technique
- ATOM (Accelerating Therapeutics for Opportunities in Medicine) researchers
  - Focus on coupling this high-quality ML model with its generative molecular design (GMD) pipeline
  - Increase the probability to generate new COVID-19 drug molecules with desired properties (e.g., diversity)
- ConveyorLC researchers focus on coupling the above with their HPC simulations
  - Allow for evaluating the docking properties of the newly generated drug candidate molecules
  - Search for an appropriate ligand that fits both energetically and geometrically the target protein's binding site
- Workflow infrastructure researchers devise workflow techniques
  - Ensure the scalability of this newly envisioned coupled workflow

Flux provides a scalable foundation for fast drug discovery workflow, Gordon Bell Covid-19 finalist

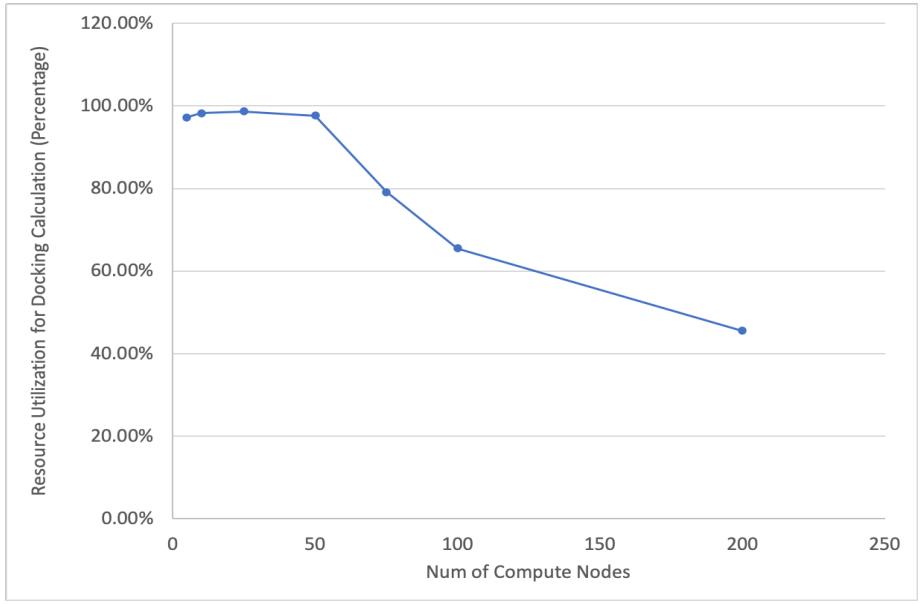


Generative molecule antiviral design loop overview.

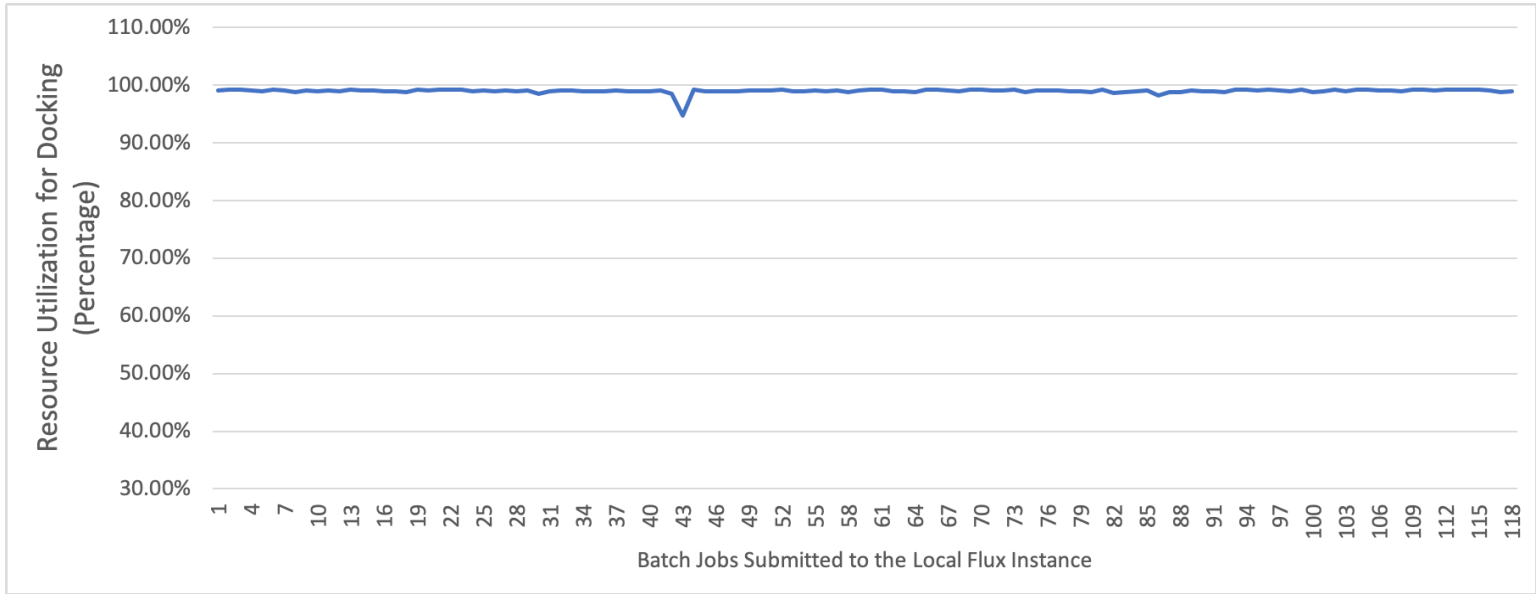
- Scalable machine learning techniques to Improve the quality and reduced the time to produce machine-learned models for use in small molecule antiviral design.
- Flux is the scalable backbone of the overall generative drug design pipeline
  - Compounds are evaluated to optimize for properties, e.g., minimal toxicity.
  - After each iteration of compound evaluation, compounds in the latent space of the generative model are perturbed to move closer to meeting the design objectives.
  - The molecules are decoded from the latent space to generate new molecules for evaluation in the next iteration.
  - As docking and MD simulations are also coupled into this pipeline, the multi-disciplinary design team approached Flux to couple them in a scalable and portable way

- https://www.exascaleproject.org/workflow-technologies-impact-sc20-gordon-bell-covid-19-award-winner-and-two-of-the-three-finalists/

Flux solves two most critical workflow problems in this COVID19 drug molecule design workflow.



Resource Utilization of ConveyorLC

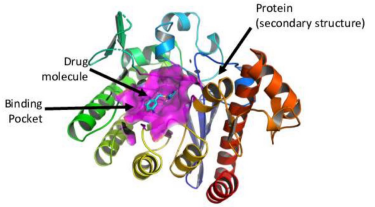


Resource Utilization at 2,950 Quartz Nodes with Flux

```
#!/bin/bash
#MSUB -A conveyorLC
#MSUB -l nodes=250
#MSUB -l walltime=16:00:00
#MSUB -l partition=quartz
#MSUB -m be
#MSUB -N flux_job

srun -N ${SLURM_NNODES} flux start ./dock_flux.job
```

```
#!/bin/bash

cDir=`pwd`

for loop in $(seq 1 10) ; do
    cd $cDir/$loop
    flux mini batch -t 16h --nslots=25 -c 36 -N 25 ./CDT3Docking.job
done

flux queue drain
flux jobs -a
```

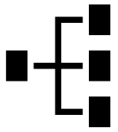
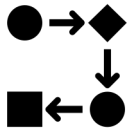
```
#!/bin/bash

let nnodes=$FLUX_JOB_NNODES
let procs=$FLUX_JOB_NNODES*9

flux mini run -N $nnodes -n $procs -c4  CDT3Docking ...
```



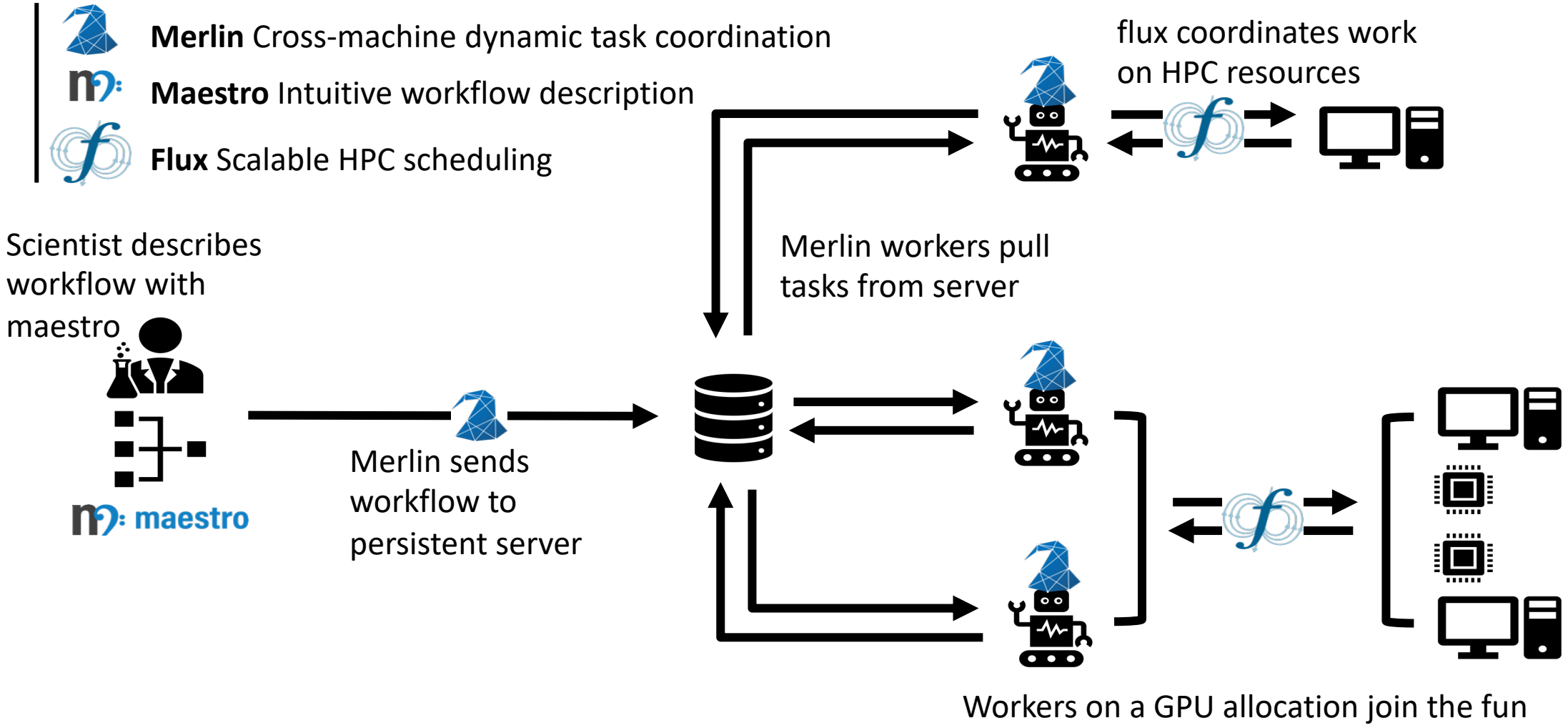
Flux is uniquely enabling many modern scientific workflows including COVID-19 problems.

- **MuMMI workflow (Cancer research)**
  - Has run over several 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 (UQ and V&V)**
  - Refactored its ensemble manager layer into a standard-alone component by building on Flux
  - Produced Themis
- **ECP ExaAM (Advanced additive manufacturing)**
  - 4x throughput improvements with 5-line change
- **MLSI Merlin workflow (Large ML/AI)**
  - Addressed throughput and portability needs
  - Demonstrated 100 million short running jobs
- **COVID-19 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
- **COVID-19 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
  - SC20 COVID-19 Gordon Bell Award finalist:

How can workflow tools cooperate?

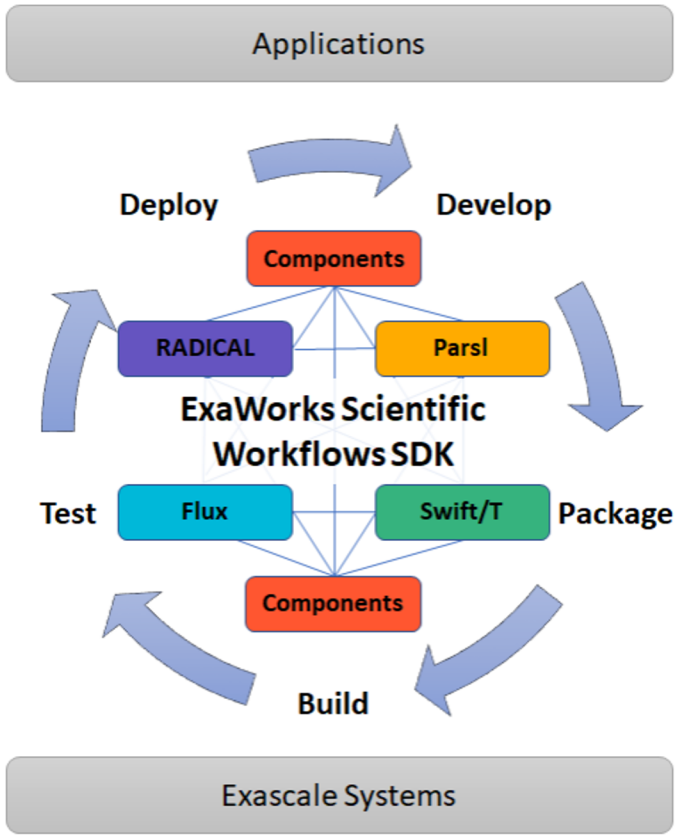
Tools		UQPipeline	sina	conduit	codes
Description		UQPipeline	maestro		
Orchestration		UQPipeline	maestro	merlin	
Resource Allocation		lsf	flux	slurm	

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



Flux is one of the founding member of ECP ExaWorks to define APIs/methodology to create a plug-N-play component-based ecosystem.

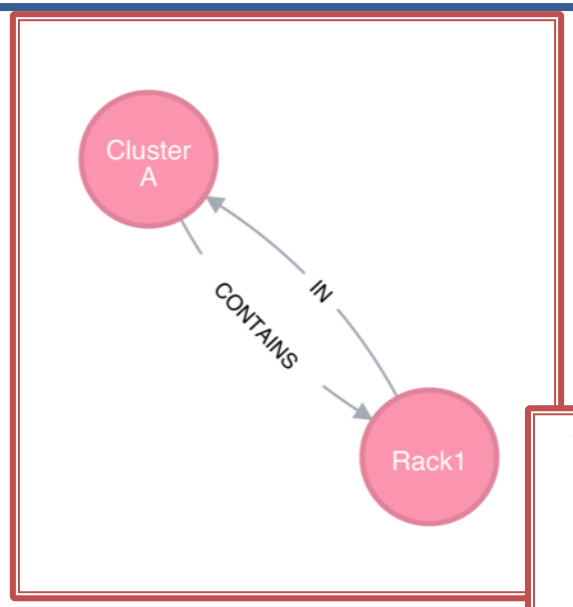
- ExaWorks: Provide a production-grade Software Development Kit (SDK) for exascale workflows
- Focus is on defining this SDK and common API's with the workflows community
- Implemented at progressively integrated levels
  - Level 0: Technologies can be packaged together
  - Level 1: Break down vertical silos and leverage capabilities via component interfaces or pairwise integrations
  - Level 2: Community developed and supported component APIs
- Approach:
  - Community policy to ensure minimum standard software quality practices for reliable deployment and use (modeled on E4S)
  - Integrate community workflows libraries/tools in the SDK and develop pair-wise integrations
  - Engage an open community-based design process
  - Ensure scalability of components on Exascale Systems



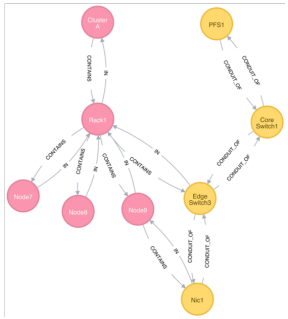
Check out our breakout slides on "*The ExaWorks Workflows SDK Project*" presented yesterday at 1PM – 2:30PM! (https://whova.com/portal/webapp/ecpan_202104/Agenda/1510949)

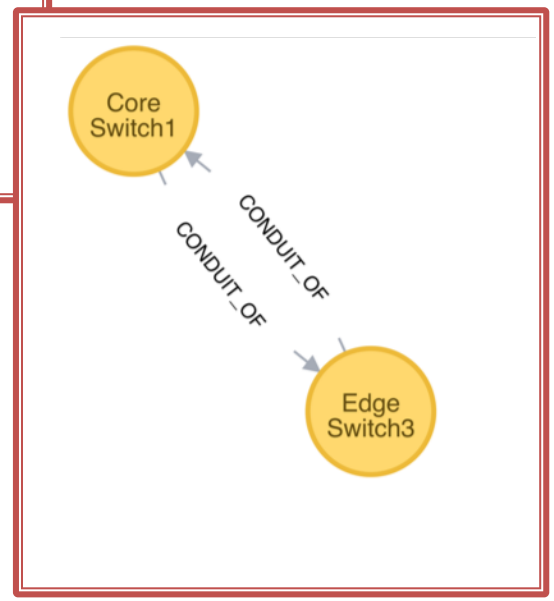
Fluxion uses a graph-based resource data model to represent schedulable resources and their relationships.

- 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

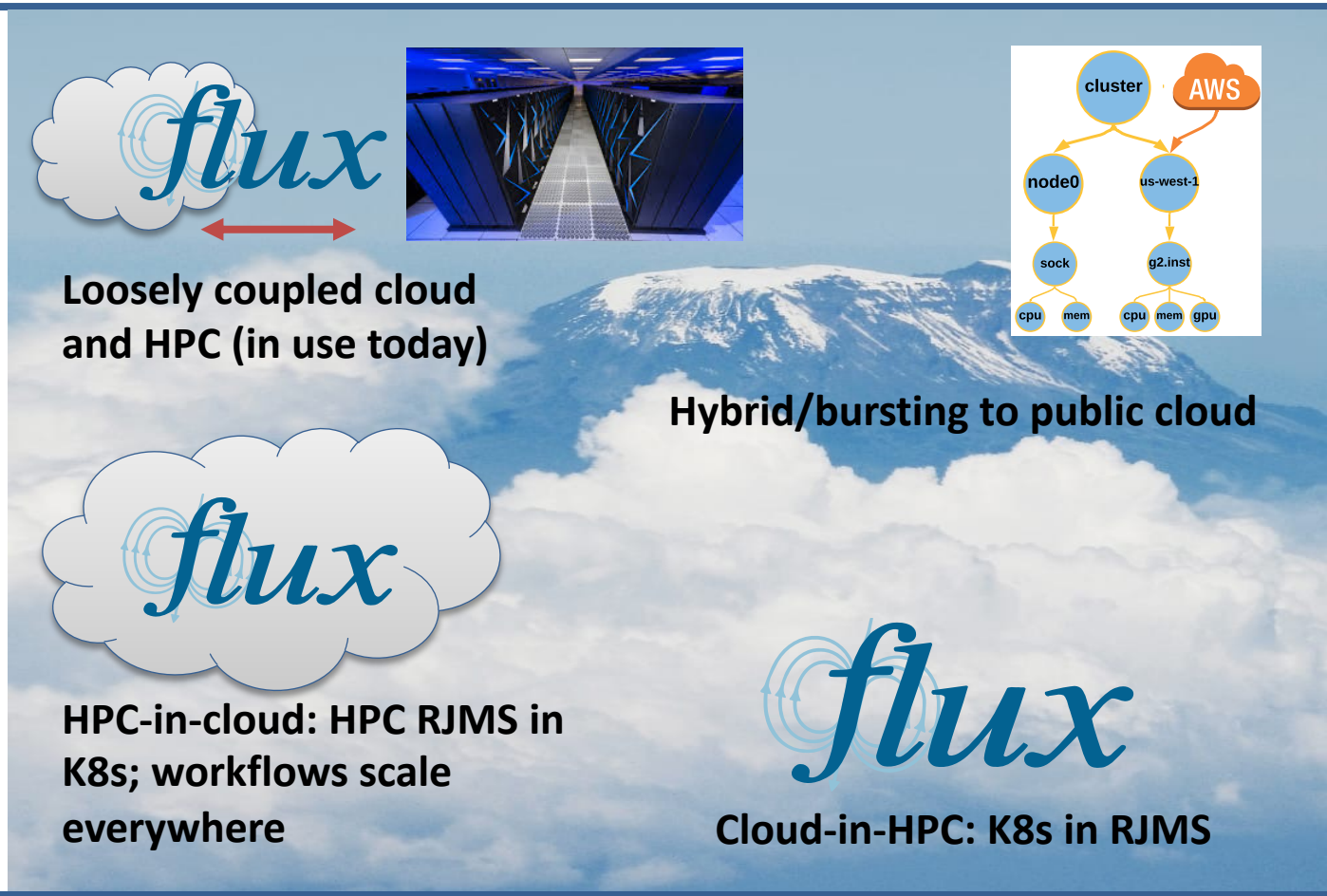




Network connectivity subsystem

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
- Check out our SC20 state-of-the-practice talk video: "Current and Future Converged Cloud-HPC Workflows at LLNL"



Check out our breakout slides on "*Converging on a Unified Scheduling and Management Framework for HPC and Cloud Tasks*" presented this morning at 10AM – 11:30AM (https://whova.com/portal/webapp/ecpan_202104/Agenda/1511029)

Fully Hierarchical Scheduling and Job Interfaces for Large Ensembles of Uncertainty Quantification (UQ) Tasks

Flux Tutorial at ECP Annual Meeting, April 14, 2020

James Corbett

Acknowledgment: Ned Bass, Subhendu Behera, Albert Chu, Kyle Chard, Brandon Cook, James Corbett, Ryan Day, Franc Di Natalie, David Domyancic, Phil Eckert, Stephen Herbein, Jim Garlick, Elsa Gonsiorowski, Mark Grondona, Helgi Ingolfsson, Shantenu Jha, Zvonko Kaiser, Dan Laney, Carlos Eduardo Gutierrez, Edgar Leon, Don Lipari, Daniel Milroy, Joseph Koning, Claudia Misale, Chris Moussa, Frank Muller, Tapasya Patki, Yoonho Park, Luc Peterson, Barry Rountree, Thomas R. W. Scogland, Becky Springmeyer, Michela Taufer, Jae-Seung Yeom, Veronica Vergara and Xiaohua Zhang

LLNL-PRES-821226

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

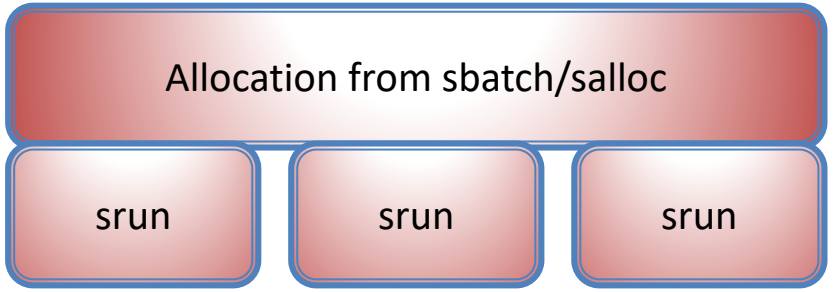
An Old Problem: Running Batch Jobs in Parallel

- Traditional HPC resource managers are great for small numbers of large jobs.
- Want to run a geophysics simulation across 500 nodes? Easy:
  - Write a batch script and submit it.
- Want to run 500 geophysics simulations across 500 nodes? Not so easy. Could submit 500 batch jobs, but:
  - Some resource managers impose limits on the number of concurrently running jobs.
  - Some HPC centers, like ANL and ORNL, prioritize large jobs.
  - Difficult to keep track of so many jobs.

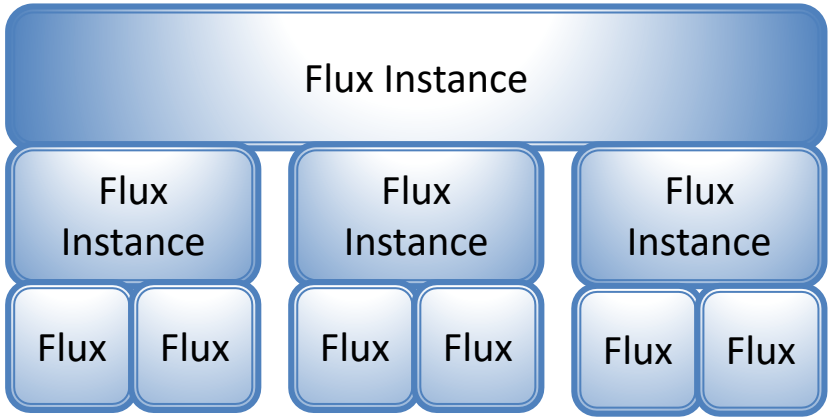
```
#!/bin/bash
#
# A user's batch-submission script

< begin complex pre-processing logic >
...

echo "starting run 1"

flux mini run -n72 -g1 /usr/global/geosx -i input_1

echo "starting run 2"

flux mini run -n36 -c2 -g1 /usr/global/geosx -i input_2

echo "starting post-processing"

< begin complex post-processing logic >
...
```

Flux's Hierarchical Scheduling: A Natural Solution

- When users want to submit large numbers of jobs, the usual solution is to use a workflow manager.
- A good solution, but has significant start-up cost. User must:
  - Learn how to use the workflow manager.
  - Translate their batch script logic into the language of the workflow manager.
- Flux's hierarchical scheduling features provide a natural solution.
  - Request one 500-node allocation and painlessly, dynamically partition it into 1-node allocations.
- Most resource managers cannot subdivide resource allocations.
  - For instance, Slurm has two levels: **sbatch** gets you an allocation, and **srun** inside an allocation launches a parallel task.
  - Calling **sbatch** from inside an allocation gets you an entirely separate allocation.



The Slurm hierarchy



One possible Flux hierarchy

Hierarchical Scheduling + Workflow Managers

- With Flux's hierarchical scheduling, users don't need to make any changes to their batch scripts.
- For more advanced use-cases, users can combine workflow managers and hierarchical scheduling.
  - LLNL's Themis and Merlin already build in support for hierarchical scheduling
- LLNL's Themis uses Flux's hierarchical scheduling for two purposes:
  - Provide a simple, streamlined interface to running batch jobs in parallel
  - Maximize scaling and performance by multiplying the number of available schedulers
- Though new, the Themis-Flux integration has already made an impact on multiple LLNL research projects.

The Four Interfaces to Flux

```python
import flux
from flux.job import JobspecV1, submit, wait

handle = flux.Flux()

compute_jobreq = JobspecV1.from_command(
    command=["./compute.py", "120"], num_tasks=4, num_nodes=2, cores_per_task=2
)
compute_jobreq.cwd = os.getcwd()
compute_jobreq.environment = dict(os.environ)

jobid = submit(handle, compute_jobreq, waitable=True)
print(jobid)
print(wait(handle, jobid))
```

The Python Library

```c
#include <flux/core.h>

int ev_flux_init (struct ev_flux *w, ev_flux_f cb,
                  flux_t *h, int events)
{
    w->cb = cb;
    w->h = h;
    w->events = events;
    if ((w->pollfd = flux_pollfd (h)) < 0)
        return -1;

    ev_prepare_init (&w->prepare_w, prepare_cb);
    ev_check_init (&w->check_w, check_cb);
    ev_idle_init (&w->idle_w, NULL);
    ev_io_init (&w->io_w, NULL, w->pollfd, EV_READ);

    return 0;
}

void ev_flux_start (struct ev_loop *loop, struct ev_flux *w)
{
    ev_prepare_start (loop, &w->prepare_w);
    ev_check_start (loop, &w->check_w);
}
```

The C Library

```lua
local flux = require 'flux'
local f = flux.new ()
local amount = tonumber (arg[1]) or 120

local function sleep (n)
    os.execute ("sleep " .. n)
end

if #arg ~= 1 then
   print ("Usage: io-forward.lua seconds")
   print ("    Forward I/O requests for seconds")
   os.exit (1)
end

local rc, err = f:sendevent ({ data = "please proceed" }, "app.iof.go")
if not rc then error (err) end
print ("Sent a go event")

print ("Will forward IO requests for " .. amount .. " seconds")
sleep (amount)
```

The Lua Library

```
#!/bin/bash

flux mini batch -N4 -n144 script.sh

JOBID_1=$(flux mini submit -n72 -g1 /usr/global/geosx -i input_1)

JOBID_2=$(flux mini submit -n36 -c2 -g1 /usr/global/geosx -i input_2)

flux job attach $JOBID_1

flux job drain
```

The Command-Line Interface

Flux's Python Library

- Like the underlying C library, Flux's Python library follows a callback-heavy event loop model.
- For instance, to submit a job and get the job ID, the basics steps are:
  1. Call Flux's submit function with a job description and get a future representing the ID of the job.
  2. Attach a callback to the future.
  3. Enter Flux's event loop, and the callback will be triggered when the ID is ready.
- The event loop is not thread-safe.
  - Many Flux objects (like futures) cannot be shared across threads.
- The model can be a burden on workflow managers that want to use a different model, or that need to support other resource managers.

FluxExecutor: Hiding the Event Loop

- The **FluxExecutor** extension to Flux's Python library provides a simple multithreading interface that completely hides Flux's event loop.
  - Almost perfectly compatible with Python's concurrent.futures standard library.
- Submit a job to **FluxExecutor**, and you get a future representing the completion of the job and all its state changes.
  - **FluxExecutor** still deals in futures and callbacks, but futures are fulfilled (and callbacks are invoked) by other threads.
  - Calling threads (i.e. user threads) never need to block on *anything.*

```python
import concurrent.futures

from flux.job import FluxExecutor


def event_callback(future, event):
    """Log the event update for the job."""
    log(f"Job {future.jobid()} triggered event {event.name!r}")


def submit_jobs(jobspecs):
    """Submit an iterable of jobs and return their results."""
    with FluxExecutor() as executor:
        futures = []
        for spec in jobspecs:
            # submit the jobspec and get a future
            fut = executor.submit(spec)
            # add event callbacks to the future
            for event in executor.EVENTS:
                fut.add_event_callback(event, event_callback)
            # append the latest future to the list
            futures.append(fut)
        # wait for all the futures to complete
        return concurrent.futures.wait(futures)
```

Eleven lines of code to submit a collection of jobs, log their event updates, and collect their results.

Coming Soon: Flux-Asyncio Integration

- Hides Flux's event loop behind **async/await** syntax.
- Users see no futures or callbacks.
- Flux's event loop runs automatically when there is work to do, without burdening the Asyncio event loop unnecessarily.

```python
import asyncio

import flux
from flux.job import JobspecV1
import flux.asyncio.job


async def submit_wait(jobspec):
    """Submit a single job and monitor it for event updates."""
    jobid = await flux.asyncio.job.submit(jobspec, waitable=True)
    async for event in flux.asyncio.job.event_watch(jobid):
        await log(f"Job {jobid} triggered event {event.name!r}")
    return await flux.asyncio.job.wait(jobid)


async def submit_jobs(jobspec_iter):
    """Submit an iterable of jobs to Flux and return their results."""
    flux.asyncio.start_flux_loop(flux.Flux())
    return await asyncio.gather(*(submit_wait(jobspec) for jobspec in jobspec_iter))
```

Ten lines of code to submit a collection of jobs, log their event updates, and collect their results.

References

- Python's concurrent.futures library: https://docs.python.org/3/library/concurrent.futures.html
- Python's asyncio library: https://docs.python.org/3/library/asyncio.html
- Flux's Python documentation: https://flux-framework.readthedocs.io/projects/flux-core/en/latest/python/modules.html
- Example usage of Flux's Python bindings: https://flux-framework.readthedocs.io/projects/flux-workflow-examples/en/latest/index.html

Flux's Resource & Job Model

Flux Tutorial at ECP Annual Meeting, April 14, 2020

Stephen Herbein

Acknowledgment: Dong H. Ahn, Ned Bass, Subhendu Behera, Albert Chu, Kyle Chard, Brandon Cook, James Corbett, Ryan Day, Franc Di Natalie, David Domyancic, Phil Eckert, Stephen Herbein, Jim Garlick, Elsa Gonsiorowski, Mark Grondona, Helgi Ingolfsson, Shantenu Jha, Zvonko Kaiser, Dan Laney, Carlos Eduardo Gutierrez, Edgar Leon, Don Lipari, Daniel Milroy, Joseph Koning, Claudia Misale, Chris Moussa, Frank Muller, Tapasya Patki, Yoonho Park, Luc Peterson, Barry Rountree, Thomas R. W. Scogland, Becky Springmeyer, Michela Taufer, Jae-Seung Yeom, Veronica Vergara and Xiaohua Zhang

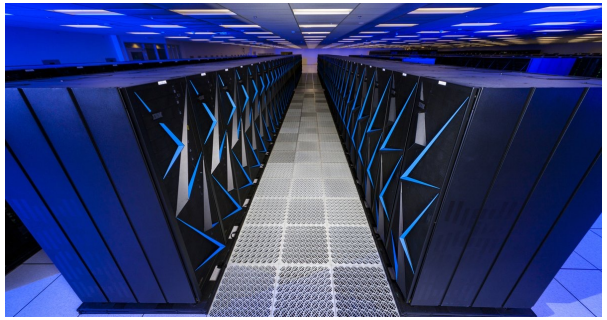
LLNL-PRES-821226

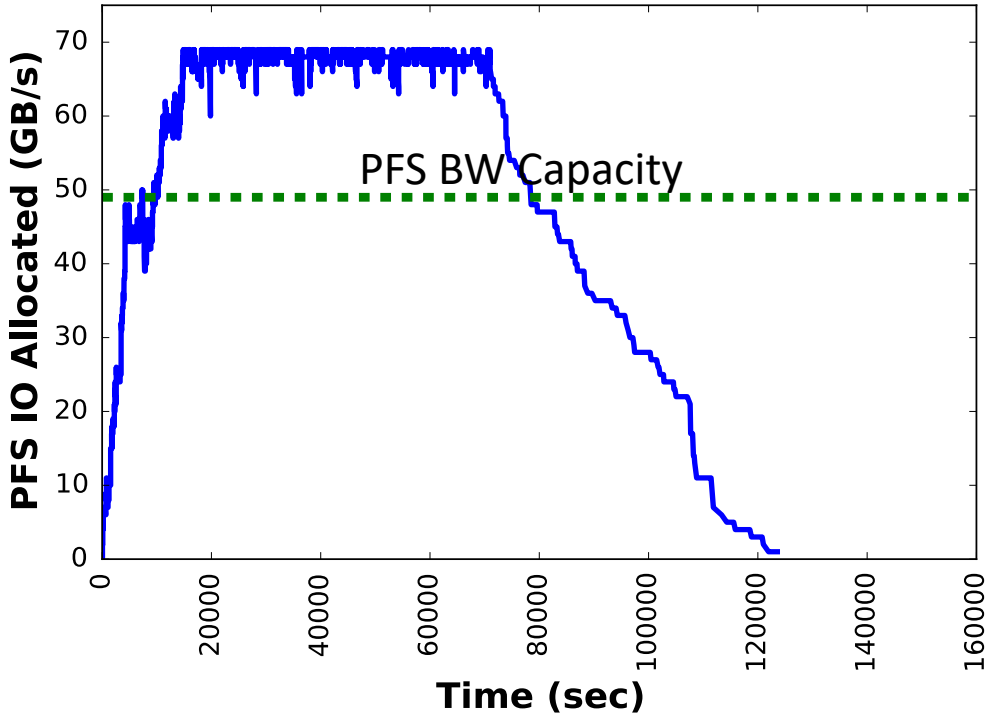
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

The changes in resource types are equally challenging.

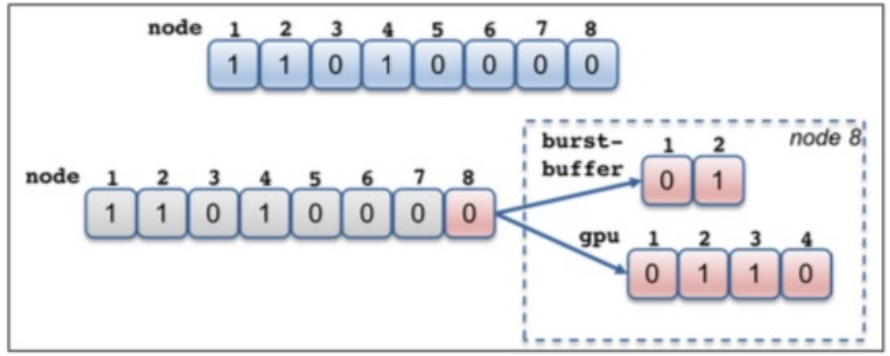
- Problems are not just confined to the workload/workflow challenge.
- Resource types and their relationships are also becoming increasingly complex.
- Much beyond compute nodes and cores…
  - GPGPUs
  - Burst buffers
  - I/O and network bandwidth
  - Network locality
  - Power





The traditional resource data models are largely ineffective to cope with the resource challenge.

- Designed when the systems are much simpler
  - Node-centric models
  - SLURM: bitmaps to represent a set of compute nodes
  - PBSPro: a linked-list of nodes

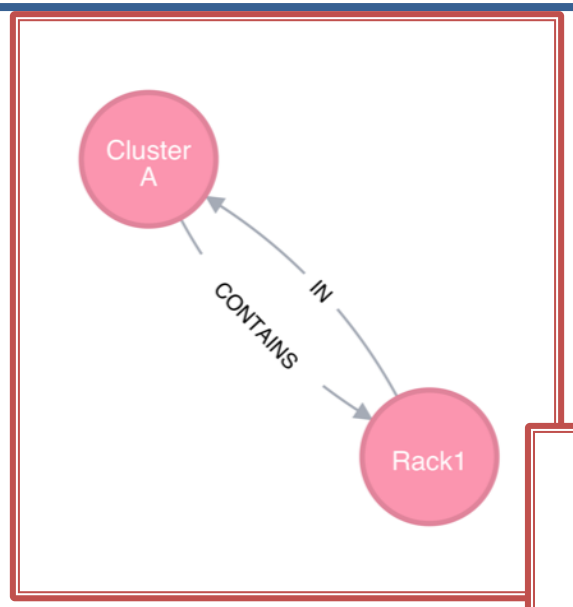


- HPC has become far more complex
  - Evolutionary approach to cope with the increased complexity
  - E.g., add auxiliary data structures on top of the node-centric data model

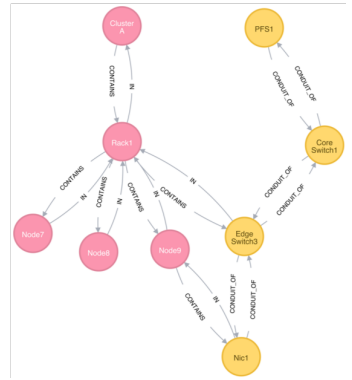
- Can be quickly unwieldy
  - Every new resource type requires new a user-defined type
  - A new relationship requires a complex set of pointers cross-referencing different types.

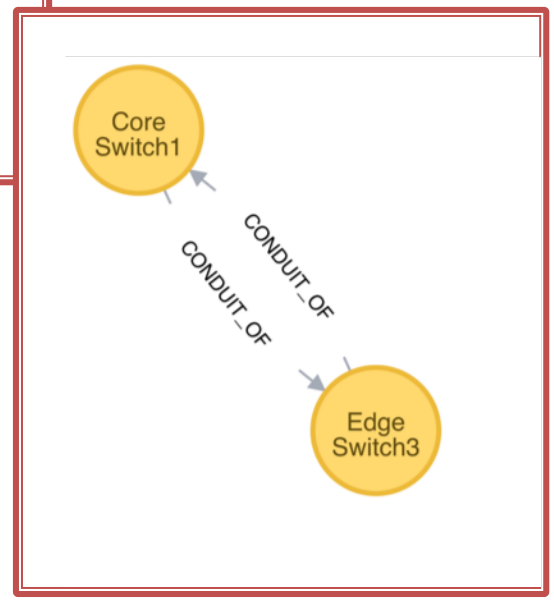
Flux uses a graph-based resource data model to represent schedulable resources and their relationships.

- 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

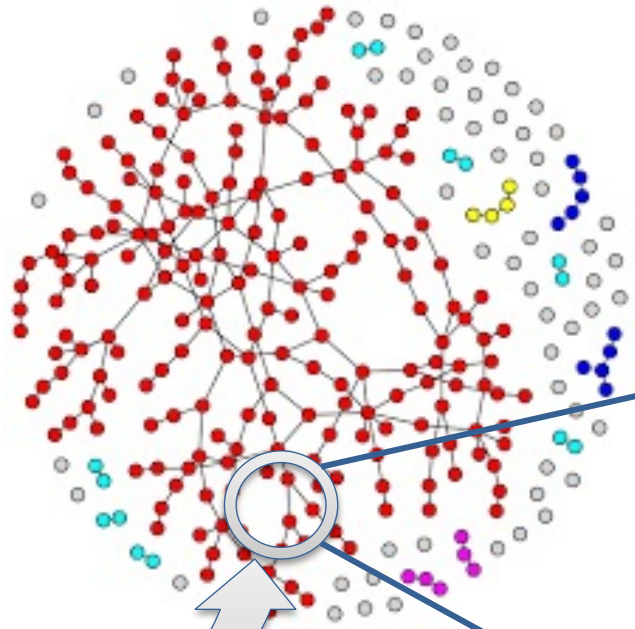
Flux's graph-oriented canonical job-spec allows for a highly expressive resource requests specification.

- 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

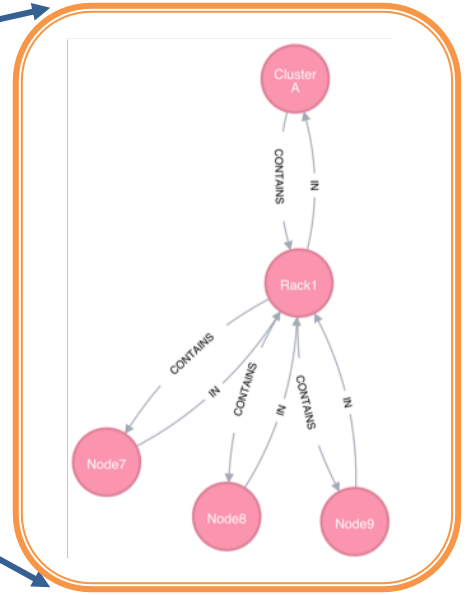
```
version: 1
resources:
  - type: cluster
    count: 1
    with:
      - type: rack
        count: 2
        with:
          - type: slot
            label: myslot
            count: 3
            with:
              - type: node
                count: 1
                with:
                  - type: socket
                    count: 2
                    with:
                      - type: core
                        count: 18

# a comment
attributes:
  system:
    duration: 3600
tasks:
  - command: app
    slot: myslot
    count:
      per_slot: 1
```

Flux maps our complex scheduling problems into graph matching problems.

```yaml
version: 1
resources:
  - type: cluster
    count: 1
    with:
      - type: rack
        count: 2
        with:
          - type: slot
            label: myslot
            count: 3
            with:
              - type: node
                count: 1
                with:
                  - type: socket
                    count: 2
                    with:
                      - type: core
                        count: 18

# a comment
attributes:
  system:
    duration: 3600
tasks:
  - command: app
    slot: myslot
    count:
      per_slot: 1
```

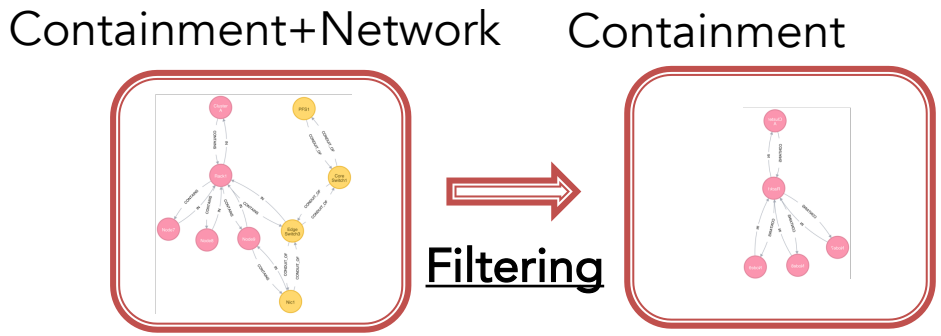


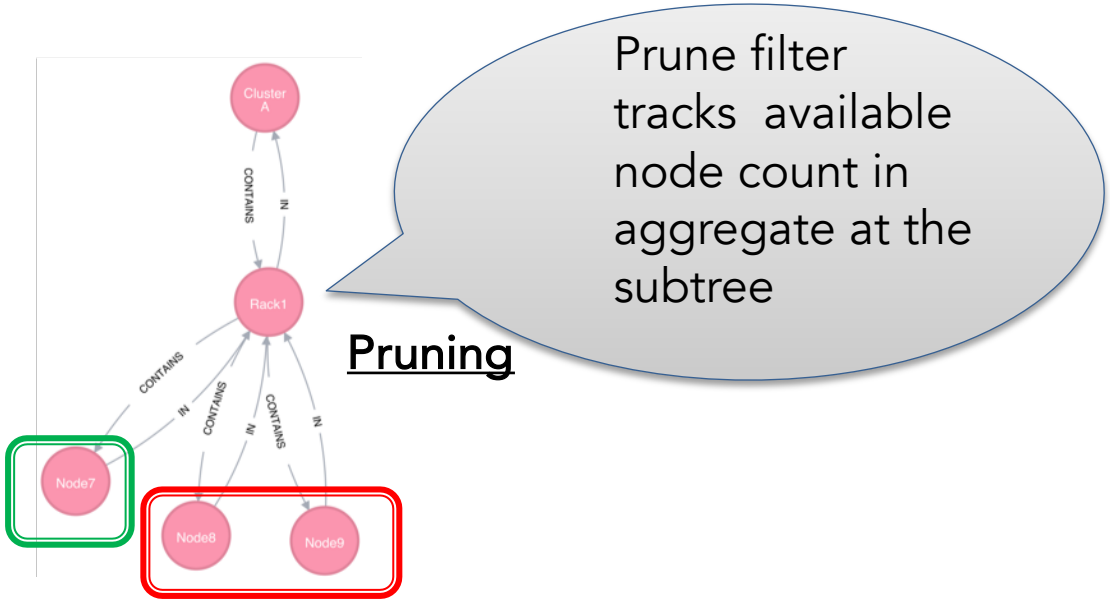
Traverse, match and score



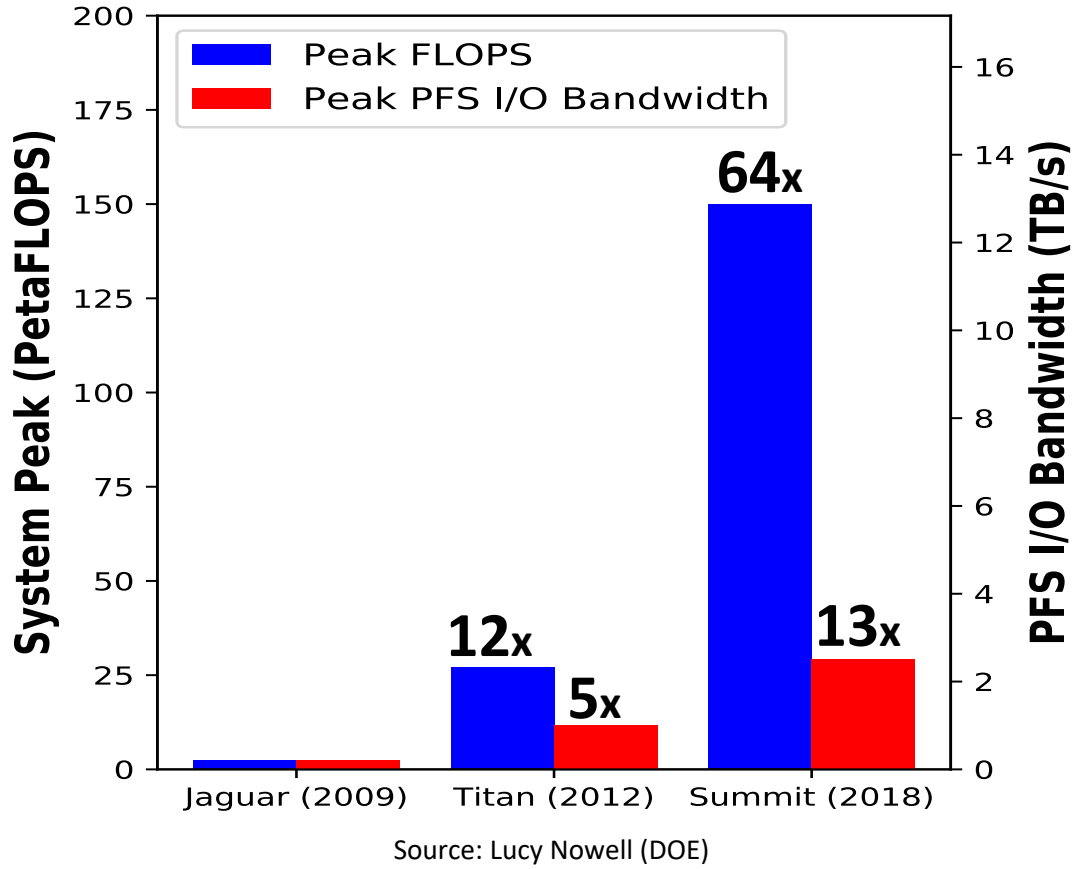
We use graph filtering and pruned searching to manage the graph complexity and optimize our graph search.

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

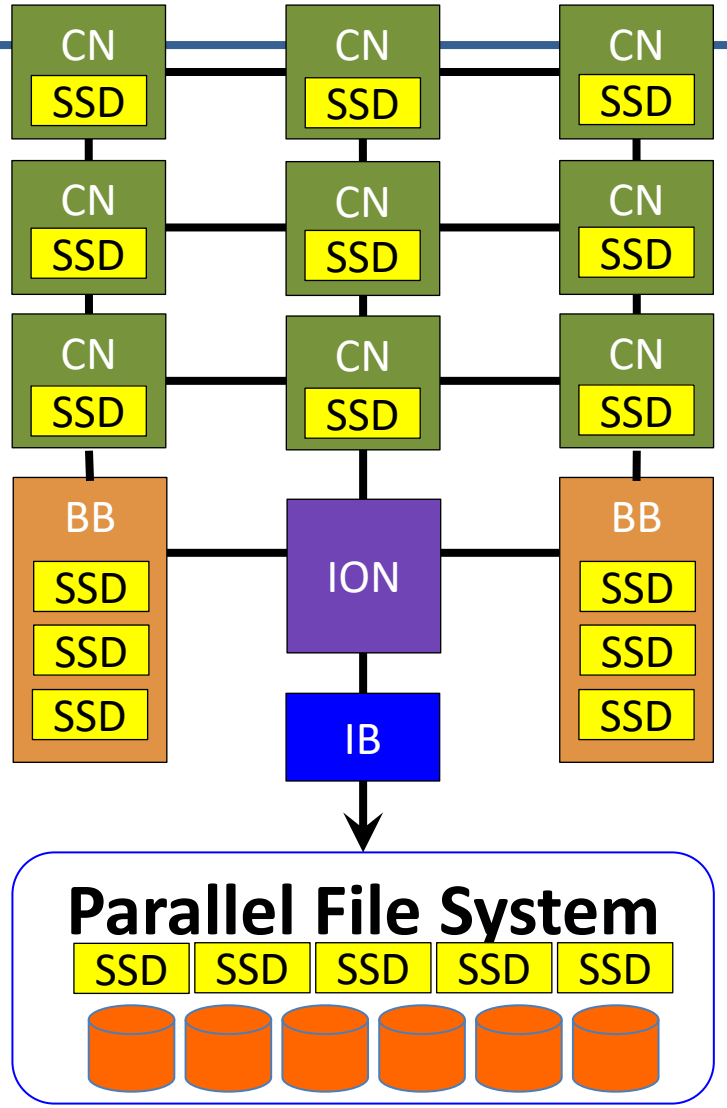




Tiered Storage in HPC

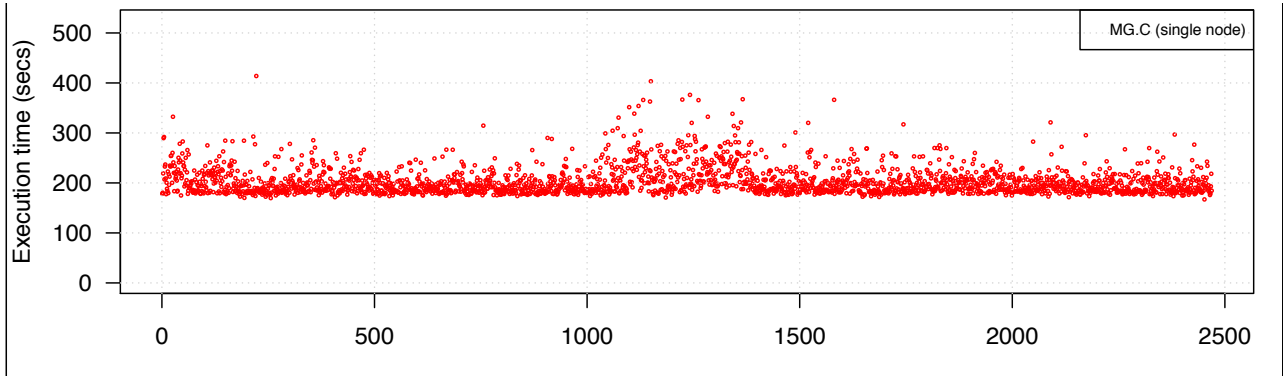


Source: Lucy Nowell (DOE)

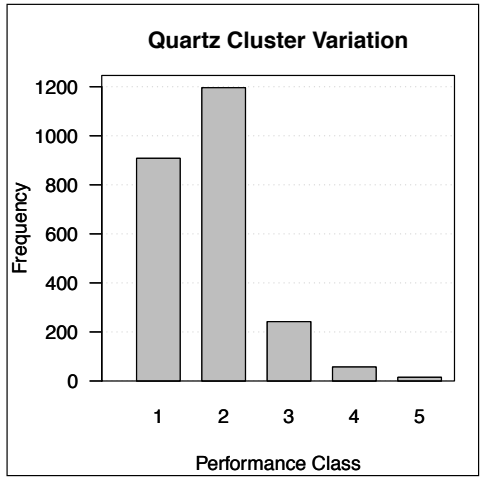


We can use the Flux graph scheduler to allocate these new storage tiers with 0 code changes

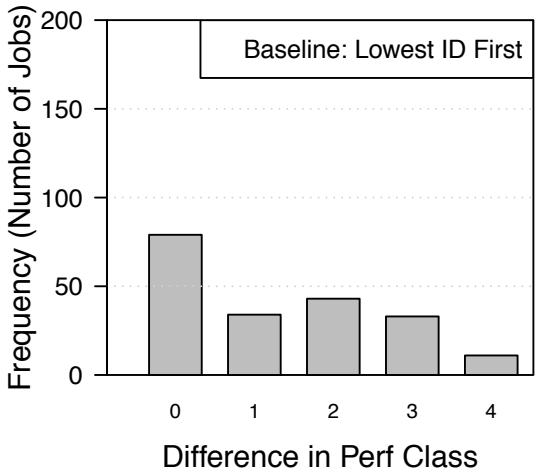
Performance Variation-Aware Scheduling

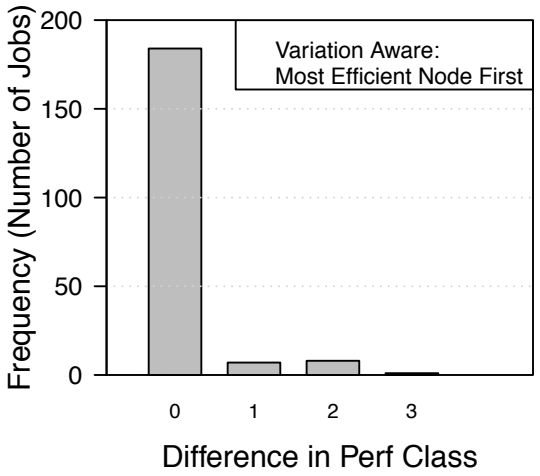


- 2.47x difference between the slowest/fastest node for MG
- 1.91x difference for LULESH.
- Idea: capture deterministic variation by statically grouping nodes into performance classes



- Implement a variation-aware match policy plugin with Flux's graph scheduler
- Goal: minimize the difference in perf classes within a job

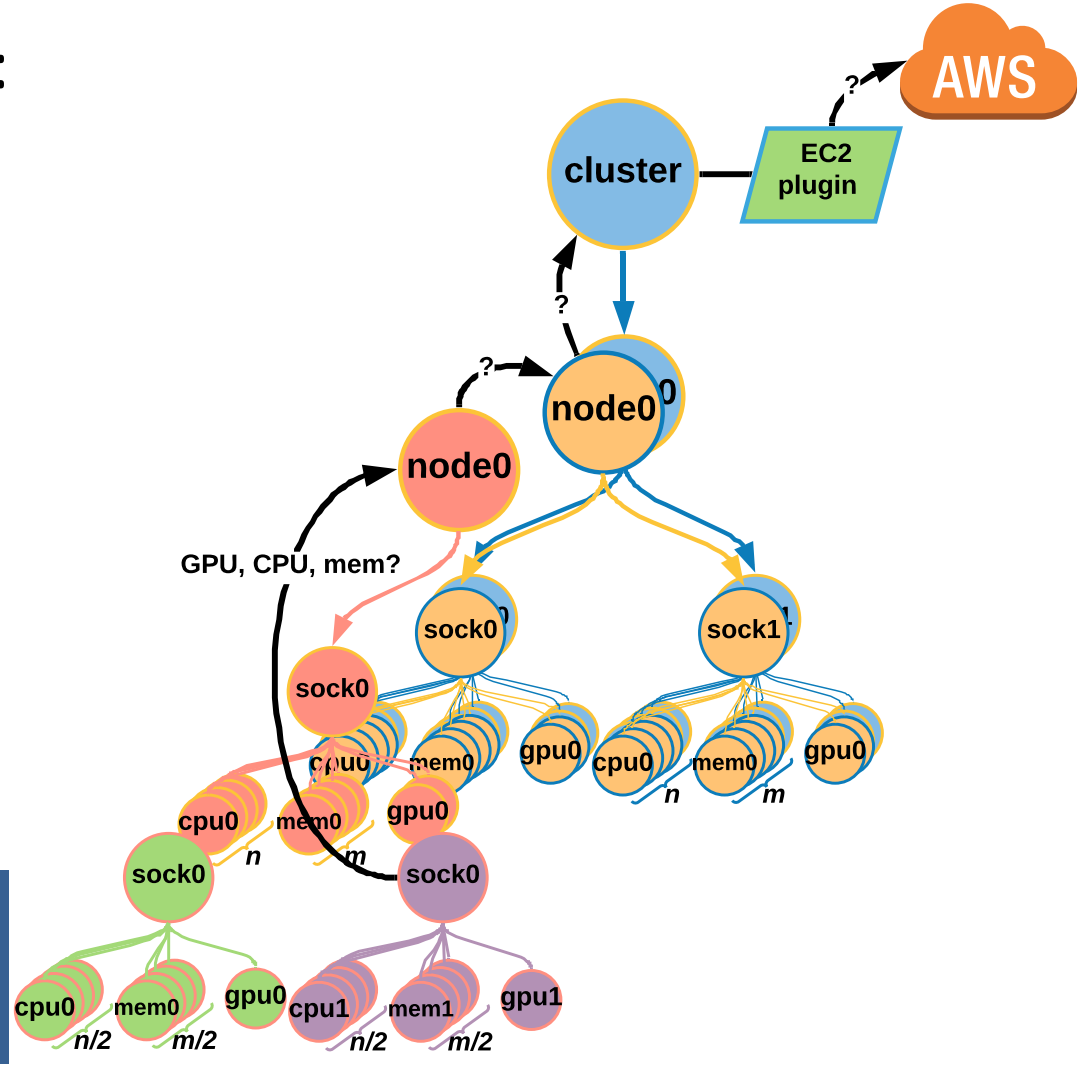




Flux's graph-based resource model easily and effectively enables this variation-aware scheduler optimization

Cloud Bursting

- Hierarchical 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



Check out our breakout slides on "*Converging on a Unified Scheduling and Management Framework for HPC and Cloud Tasks*" (https://whova.com/portal/webapp/ecpan_202104/Agenda/1511029)

Play The Full Game!

Flux Tutorial at ECP Annual Meeting, April 14, 2020

Dan Milroy

Acknowledgment: Dong H. Ahn, Ned Bass, Subhendu Behera, Albert Chu, Kyle Chard, Brandon Cook, James Corbett, Ryan Day, Franc Di Natalie, David Domyancic, Phil Eckert, Stephen Herbein, Jim Garlick, Elsa Gonsiorowski, Mark Grondona, Helgi Ingolfsson, Shantenu Jha, Zvonko Kaiser, Dan Laney, Carlos Eduardo Gutierrez, Edgar Leon, Don Lipari, Daniel Milroy, Joseph Koning, Claudia Misale, Chris Moussa, Frank Muller, Tapasya Patki, Yoonho Park, Luc Peterson, Barry Rountree, Thomas R. W. Scogland, Becky Springmeyer, Michela Taufer, Jae-Seung Yeom, Veronica Vergara and Xiaohua Zhang

LLNL-PRES-821226

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

Hand—on Session

Flux Tutorial at ECP Annual Meeting, April 14, 2020

Stephen Herbein & Tapasya Patki

Acknowledgment: Dong H. Ahn, Ned Bass, Subhendu Behera, Albert Chu, Kyle Chard, Brandon Cook, James Corbett, Ryan Day, Franc Di Natalie, David Domyancic, Phil Eckert, Stephen Herbein, Jim Garlick, Elsa Gonsiorowski, Mark Grondona, Helgi Ingolfsson, Shantenu Jha, Zvonko Kaiser, Dan Laney, Carlos Eduardo Gutierrez, Edgar Leon, Don Lipari, Daniel Milroy, Joseph Koning, Claudia Misale, Chris Moussa, Frank Muller, Tapasya Patki, Yoonho Park, Luc Peterson, Barry Rountree, Thomas R. W. Scogland, Becky Springmeyer, Michela Taufer, Jae-Seung Yeom, Veronica Vergara and Xiaohua Zhang

LLNL-PRES-821226

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

Accessing the Hands-On Demo

- Run locally with Docker:
  - docker run –ti –p 8888:8888 fluxrm/ecp-tutorial:2021
  - This image will be available in perpetuity
- Using our EC2 instances
  - ssh ecpuser@ecp-X.herbein.dev
  - Ping the chat for your value of X
  - Password (for ssh and jupyter notebook): 2021fluxecp
  - These instances will disappear shortly after the tutorial ends
- In the next section, we will help you get things running on your own cluster!

Bring Your Own Cluster (BYOC): Testing Flux on your clusters

Flux Tutorial at ECP Annual Meeting, April 14, 2020

All

Acknowledgment: Dong H. Ahn, Ned Bass, Subhendu Behera, Albert Chu, Kyle Chard, Brandon Cook, James Corbett, Ryan Day, Franc Di Natalie, David Domyancic, Phil Eckert, Stephen Herbein, Jim Garlick, Elsa Gonsiorowski, Mark Grondona, Helgi Ingolfsson, Shantenu Jha, Zvonko Kaiser, Dan Laney, Carlos Eduardo Gutierrez, Edgar Leon, Don Lipari, Daniel Milroy, Joseph Koning, Claudia Misale, Chris Moussa, Frank Muller, Tapasya Patki, Yoonho Park, Luc Peterson, Barry Rountree, Thomas R. W. Scogland, Becky Springmeyer, Michela Taufer, Jae-Seung Yeom, Veronica Vergara and Xiaohua Zhang

LLNL-PRES-821226

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