Flux: Using Next-Generation Resource Management and Scheduling Infrastructure for Exascale Workflows

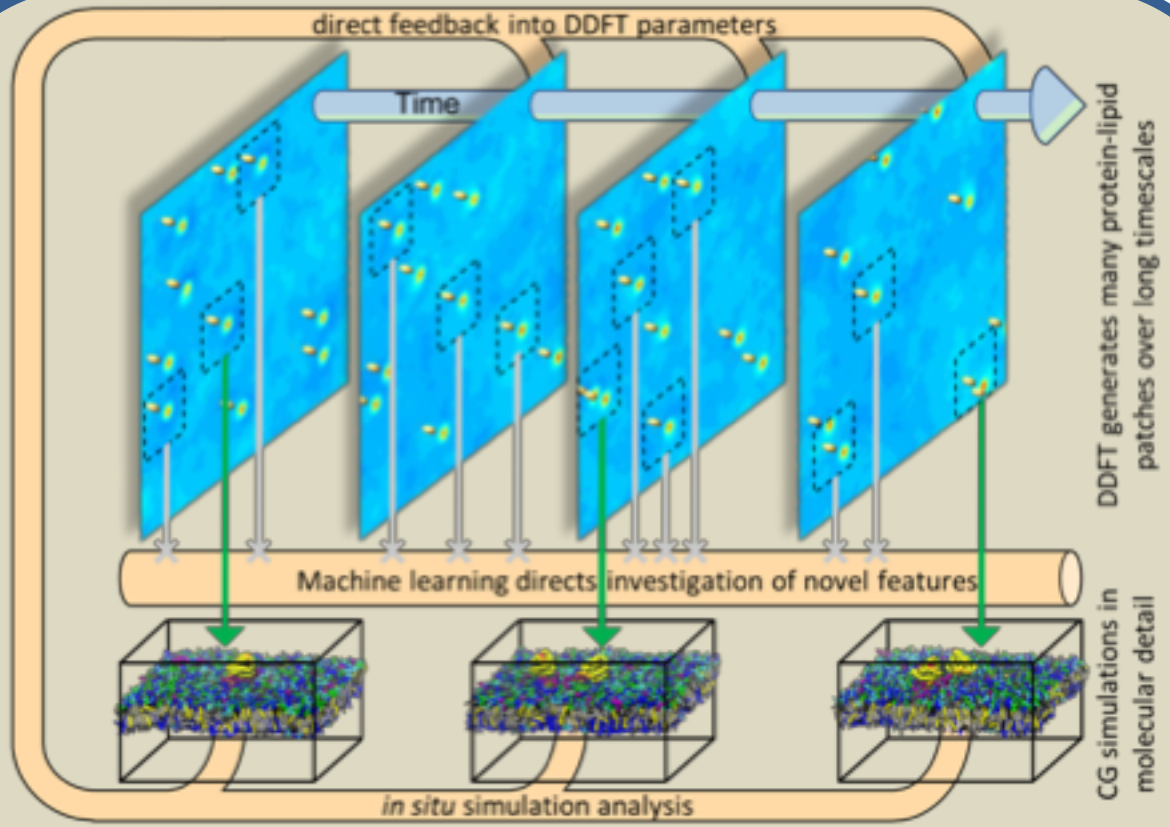
University of Tennessee, Knoxville - March 8th, 2019

Dong H. Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, **Stephen Herbein**, Joseph Koning, Tapasya Patki, Thomas R. W. Scogland, Becky Springmeyer, and Michela Taufer

LLNL-PRES-768738
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

Workflows on high-end HPC systems are undergoing significant changes

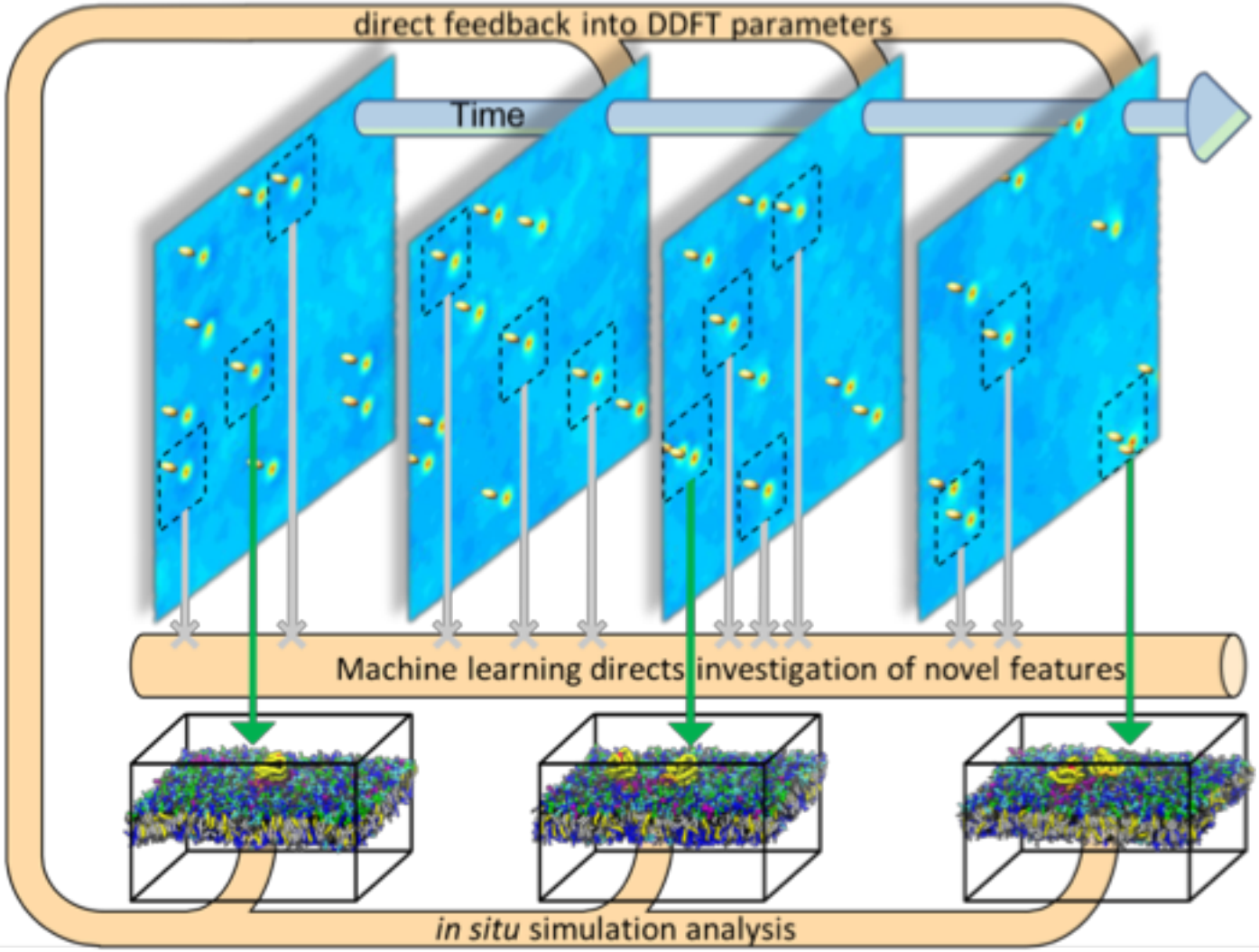


Cancer Moonshot Pilot2:
many co-scheduled elements

Automated Testing System (ATS)

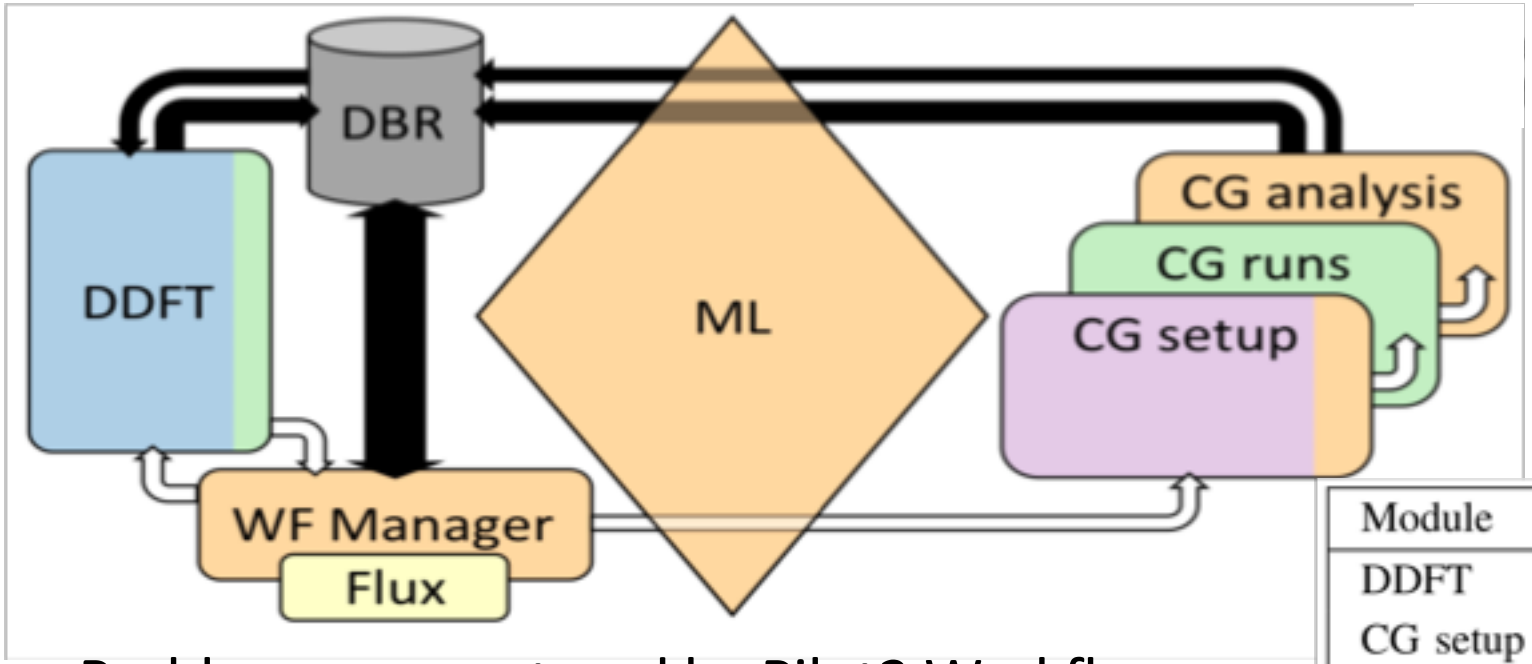
Uncertainty Quantification Pipeline (UQP)

Pilot2 Workflow Overview



- Cellular-scale continuum model simulates large spatial and temporal scales
- Machine learning model selects novel patches to simulate in molecular detail
- Microscale results fed back into macroscale

Pilot2 Scheduling Requirements



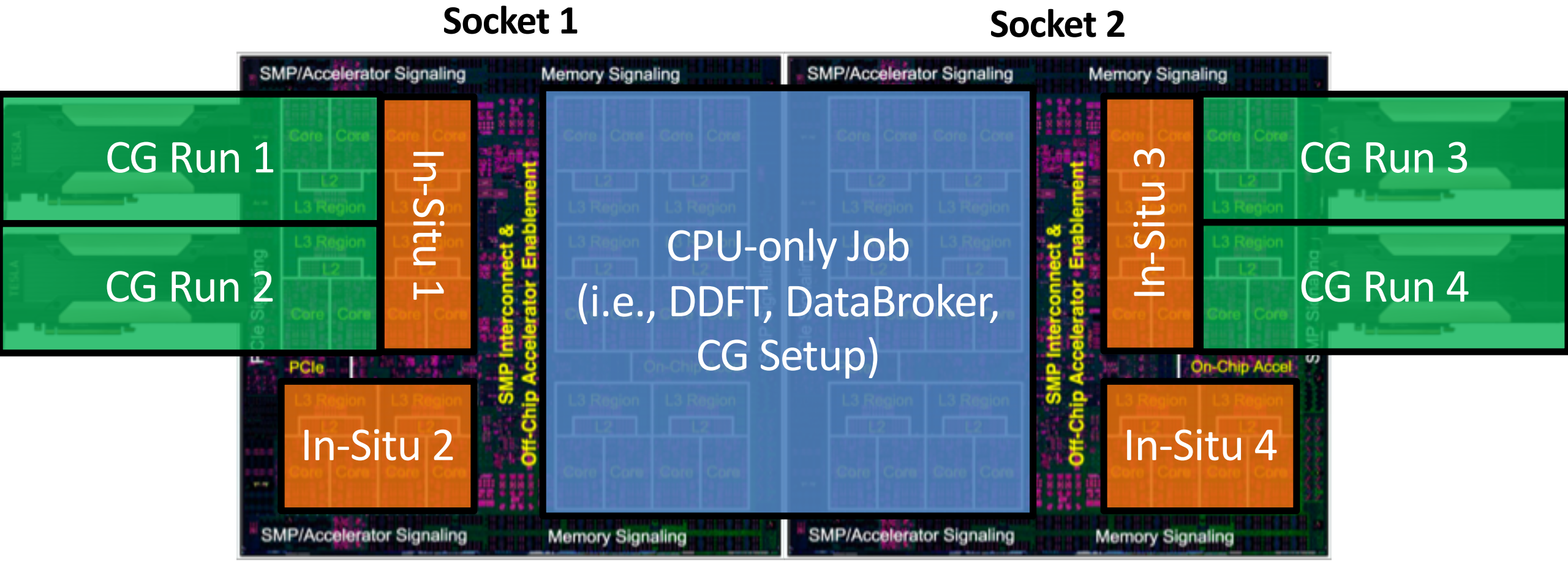
- Many co-scheduled components, some are co-located
- Each with their own unique resource requirements

Module	# Jobs	# Nodes	# CPU cores	# GPUs
DDFT	1	1000	1000×24	0
CG setup	2528	2528	2528×24	0
CG runs	3540×4	3540	3540×8	3540×4
CG analysis	3540×4	3540	3540×12	0
DBR	1	10	10×24	0
PatchCreator	1	1	1×24	0
WFManager	1	1	1×24	0
Total	30852	3540	3540×44	3540×4

Problems encountered by Pilot2 Workflow:

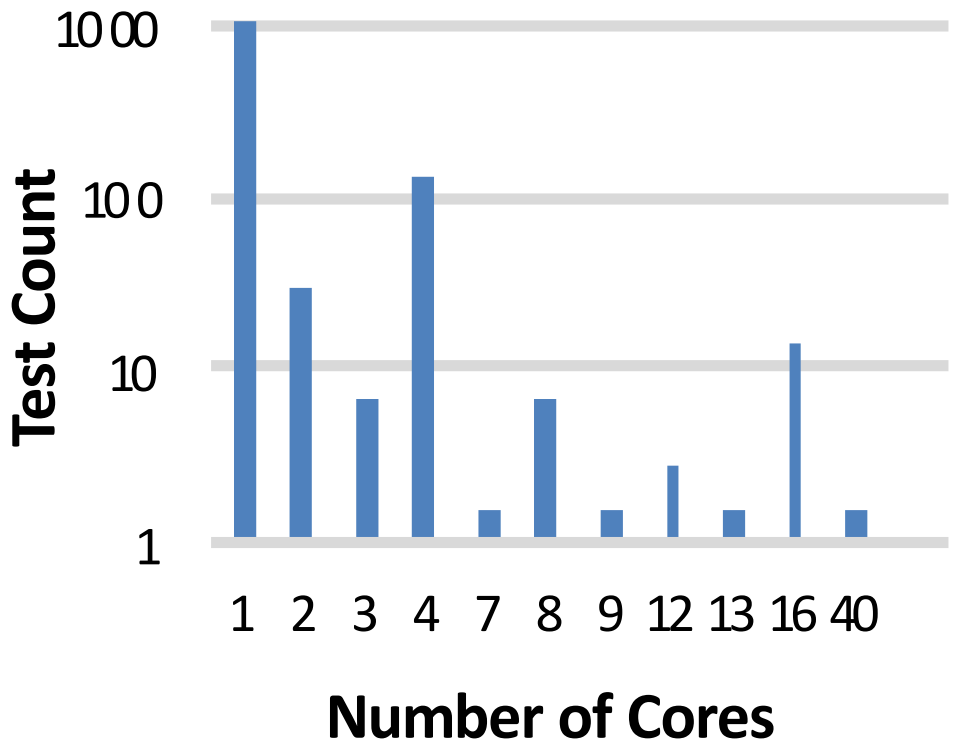
- Co-locating jobs on the same node + binding to the optimal cores and GPUs
- Saturating the system with 30,000 simultaneous jobs
- Communication between the loosely-coupled apps
- Transitioning to different systems (with different scheduler)

Pilot2 Scheduling Requirements – Scheduling on a Single Node



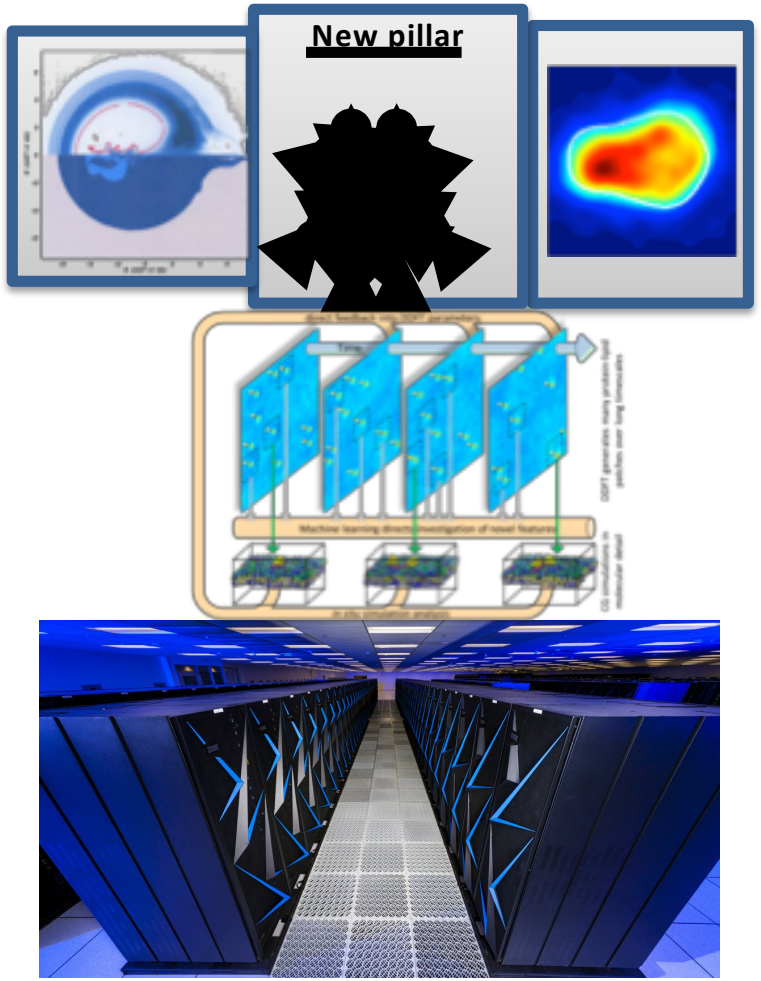
Other “Workhorse” Workflows

- Automated Testing System (ATS)
  - Regression and performance testing of MPI application
  - Run the same application at varying scales, analyze output for correctness
  - Problems encountered: Co-locating applications, job throughput, and portability
- Uncertainty Quantification Pipeline (UQP)
  - Identify and characterize source of uncertainty
  - Require running the same application many times with varying inputs
  - Problems encountered: job throughput and portability



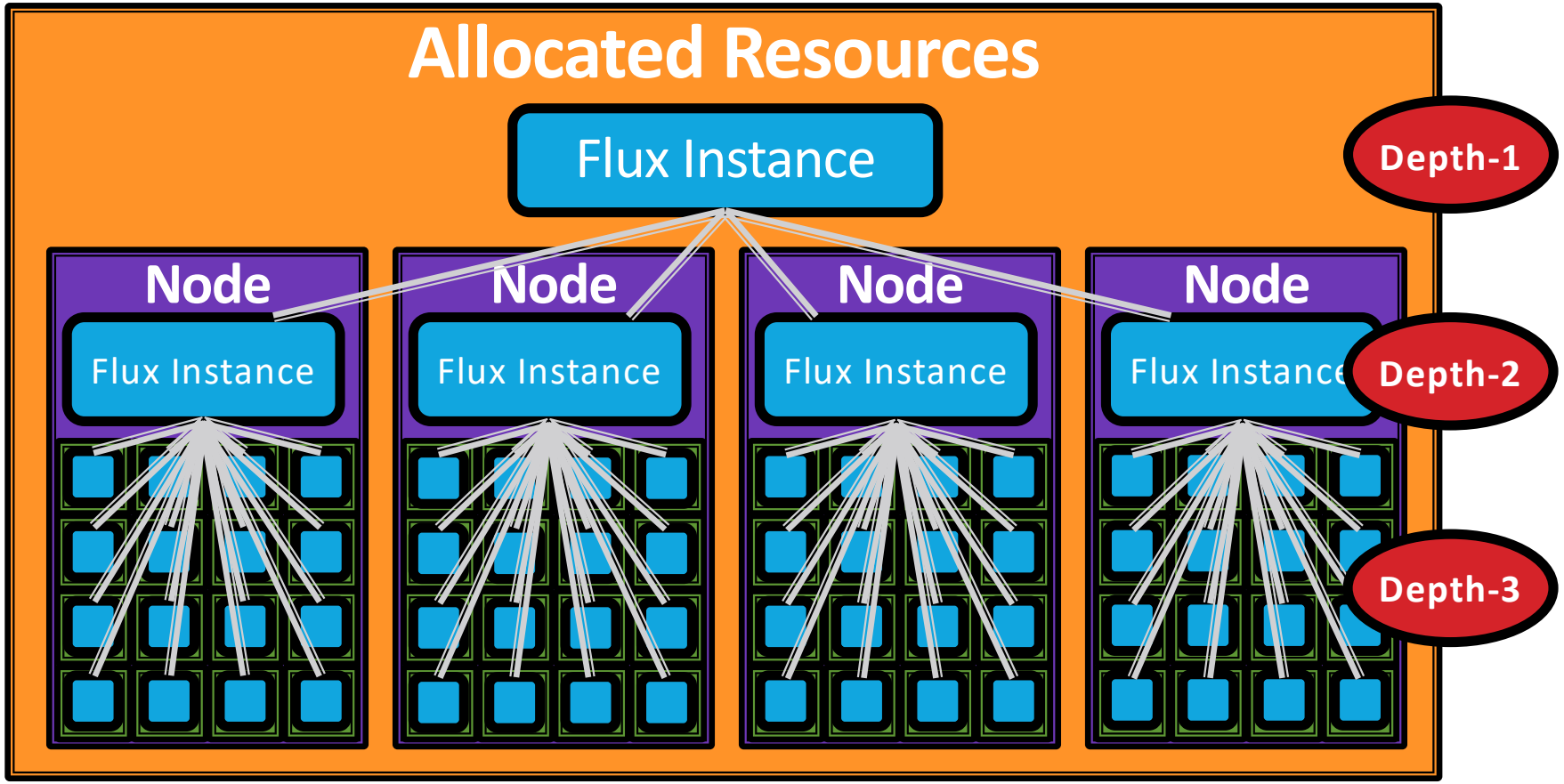
ATS workflow’s diverse resource requirements

Key challenges in emerging workflow scheduling include…



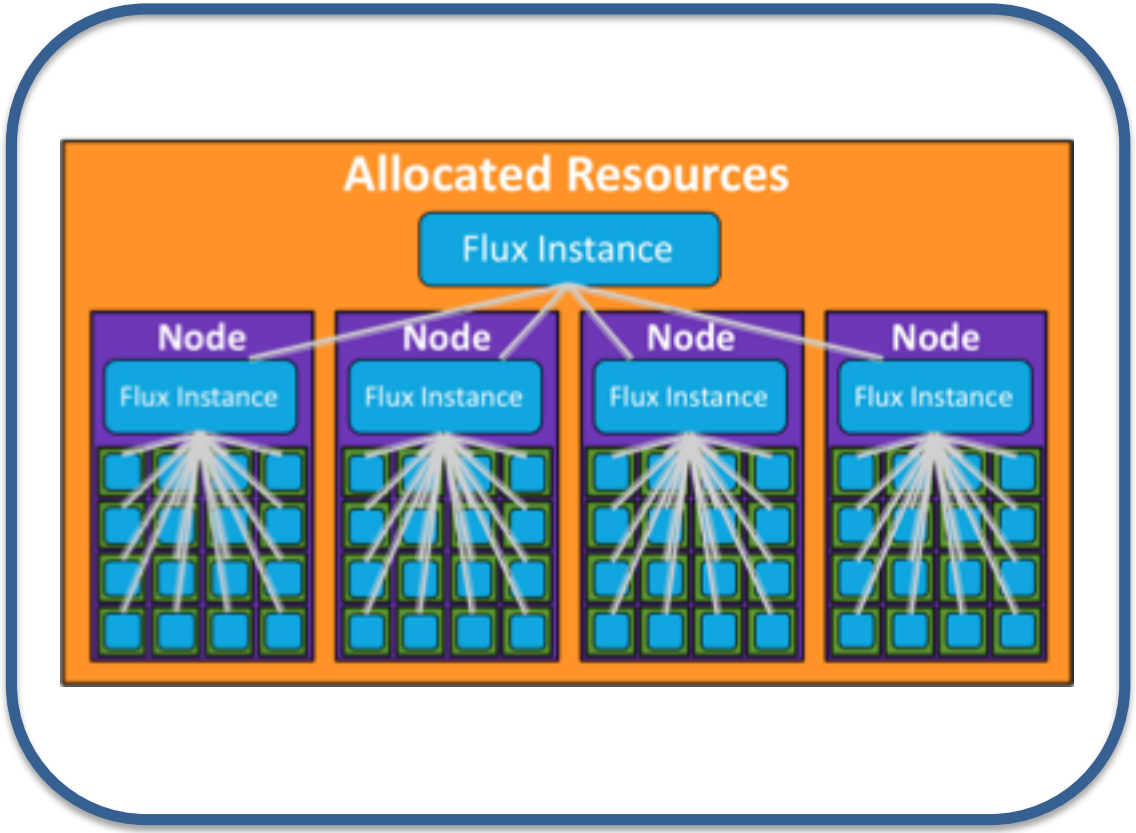
- Job throughput challenge
- Co-scheduling challenge
- Job communication/coordination challenge
- Portability challenge

Flux provides a new scheduling model to meet these challenges



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

Flux is specifically designed to embody our fully hierarchical scheduling model



Techniques

Hierarchical Nesting

Scheduler Specialization

Rich API set

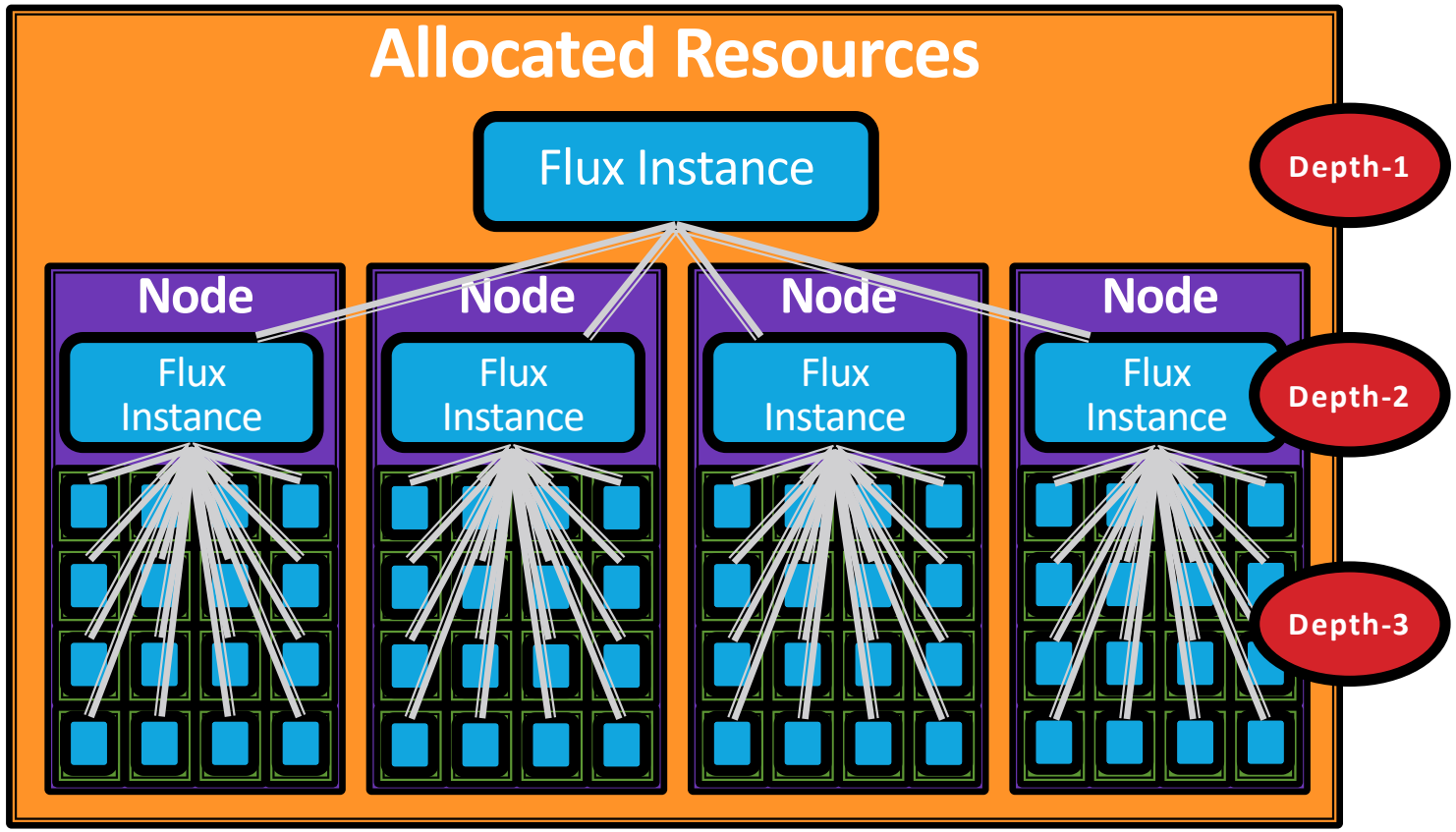
Consistent API set

Challenges

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

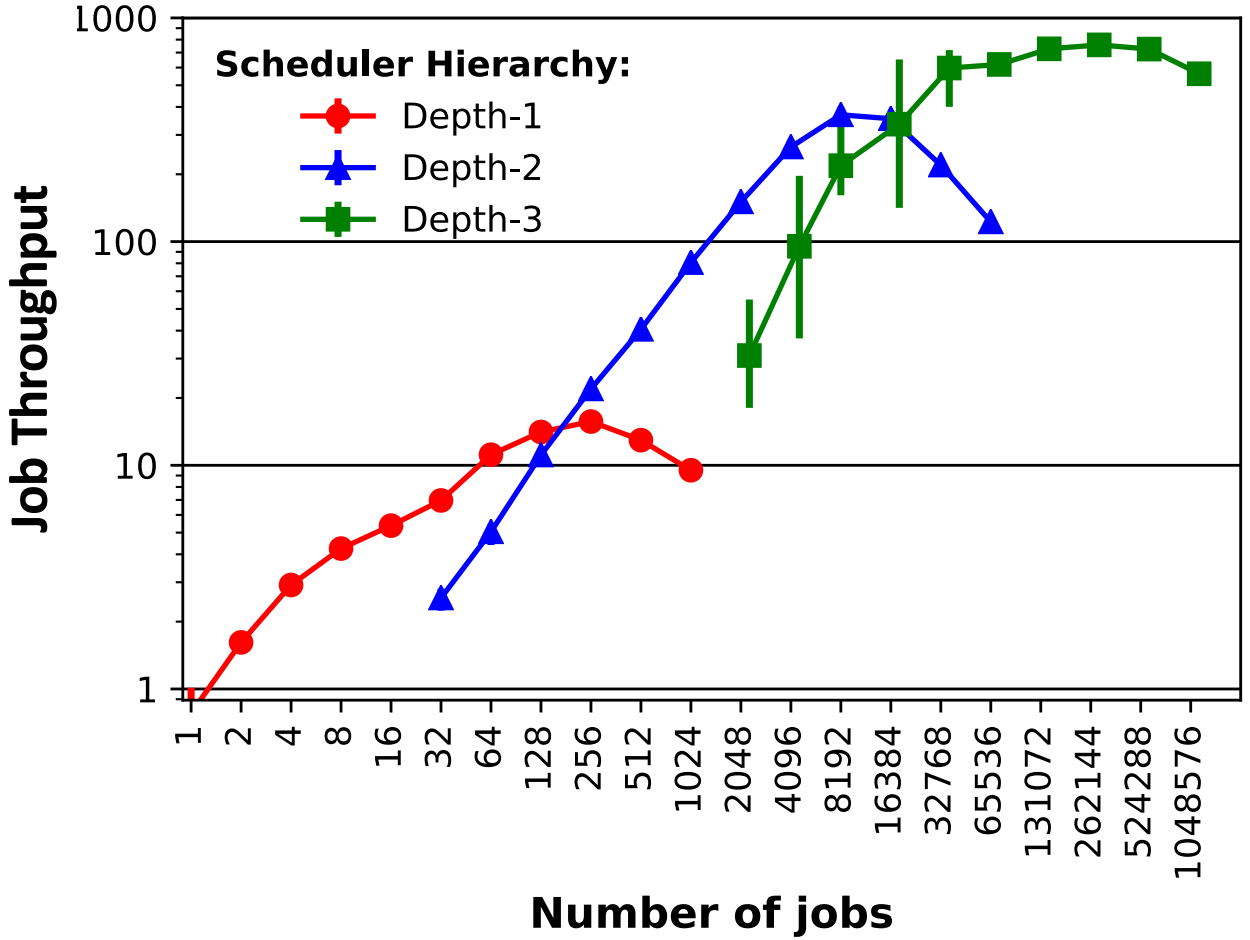
Hierarchical nesting solves the throughput challenge

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

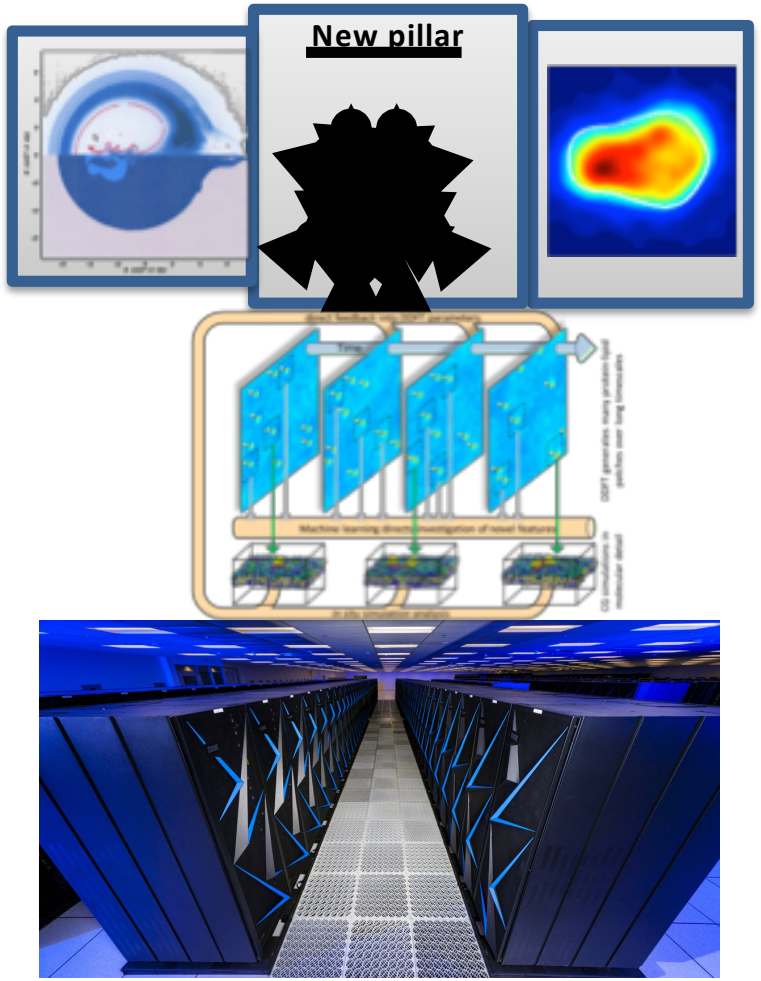


Hierarchical nesting solves the throughput challenge

- UQ Pipeline Workflow
- 1,152 cores
- Variable numbers of jobs
- Implementation used in our evaluation:
  - Submit each job in the ensemble individually to the root
  - The jobs are distributed automatically across the hierarchy



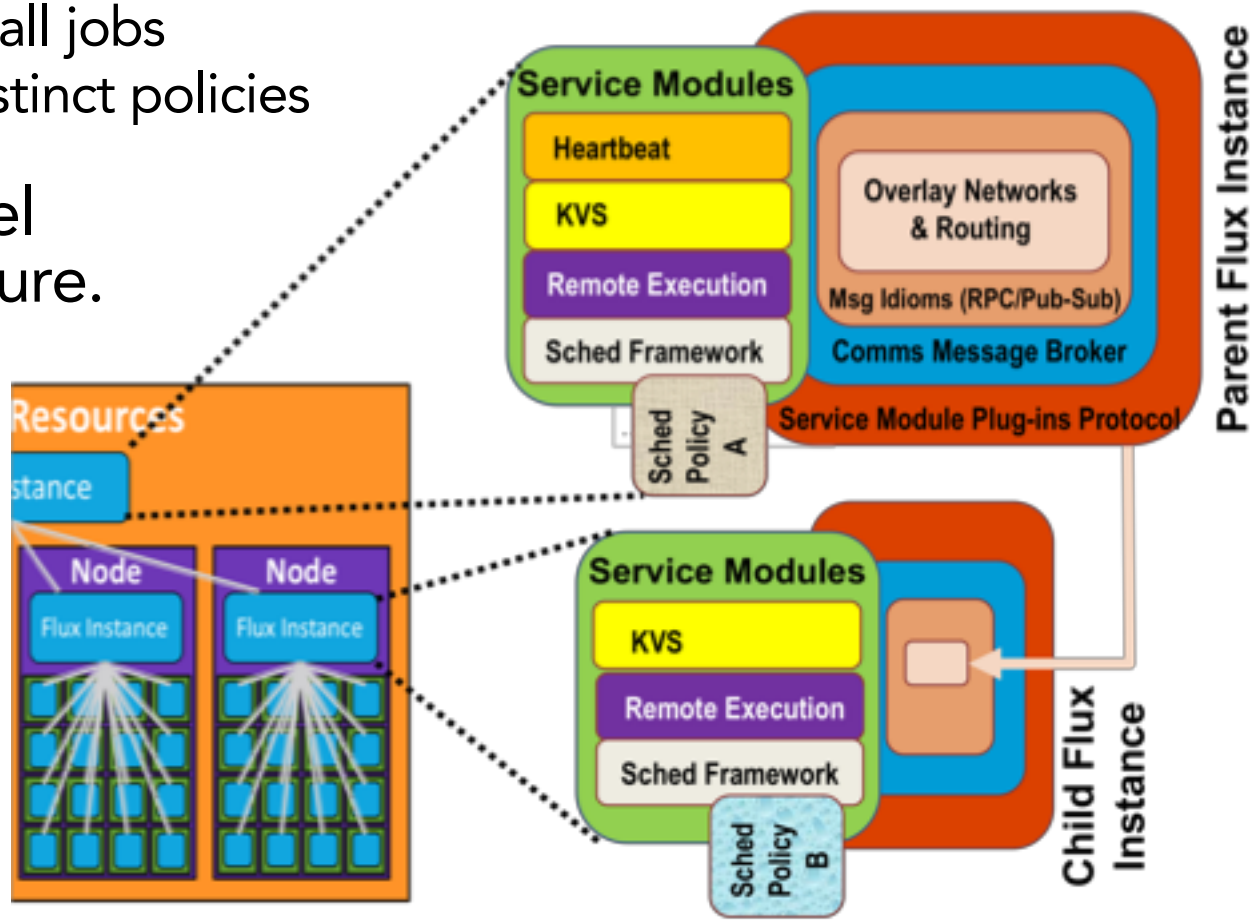
Key challenges in emerging workflow scheduling include…



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

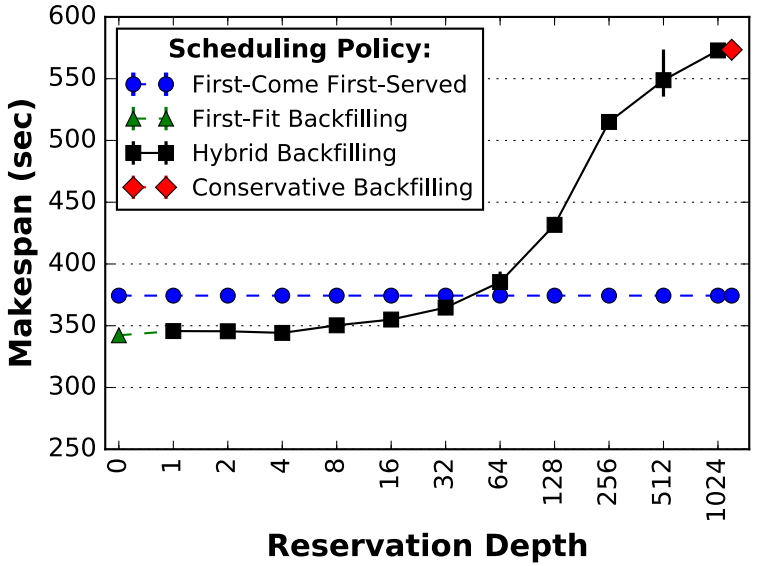
Scheduler specialization solves the co-scheduling challenge

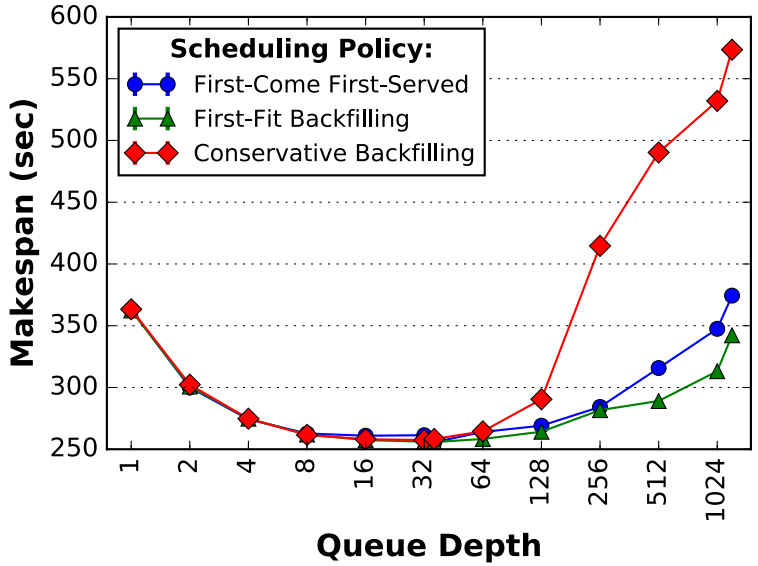
- Traditional approach
  - A single site-wide policy being enforced for all jobs
  - No support for user-level scheduling with distinct policies
- Flux enables both system- and user-level scheduling under a common infrastructure.
- Give users the freedom to adapt their scheduler instance to their needs.
  - Instance owners can customize the scheduling policy
    - Enable/disable backfilling
    - Tune scheduling parameters
  - Create their own policy plug-in
    - Variation-aware scheduling

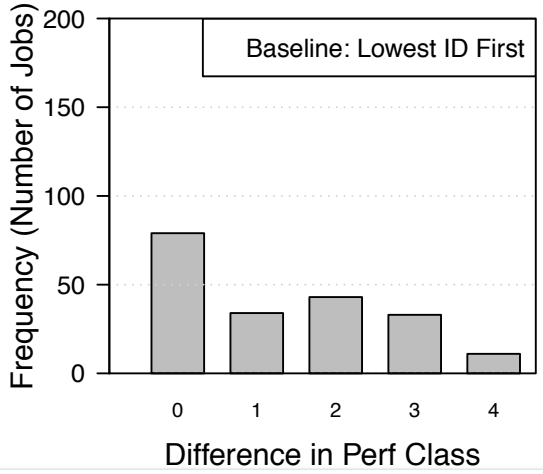


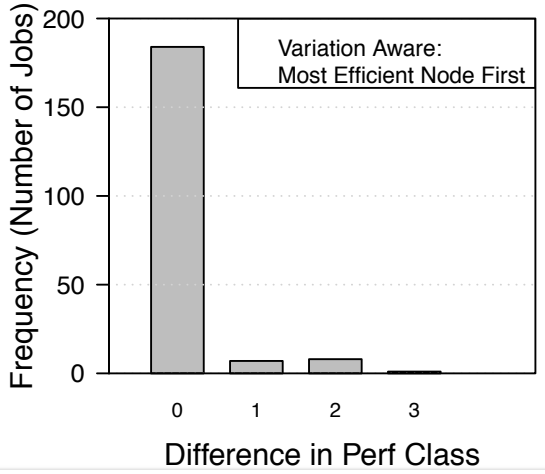
Scheduler specialization solves the co-scheduling challenge and helps the throughput challenge

- ATS workflow
  - Run under FCFS and Backfill policies
  - Vary the *reservation depth* and *queue depth* scheduling parameters
  - 2.2x end-to-end speedup over system default settings
- Traces from LLNL's Quartz
  - Bin nodes based on performance
    - <10%, 10-25%, 26-40%, 41-60%, >60%
  - Run under baseline & variation-aware policies
  - Measure the difference in bins (slowest bin – fastest bin) per job allocation
  - 2.4x reduction in rank-to-rank variation

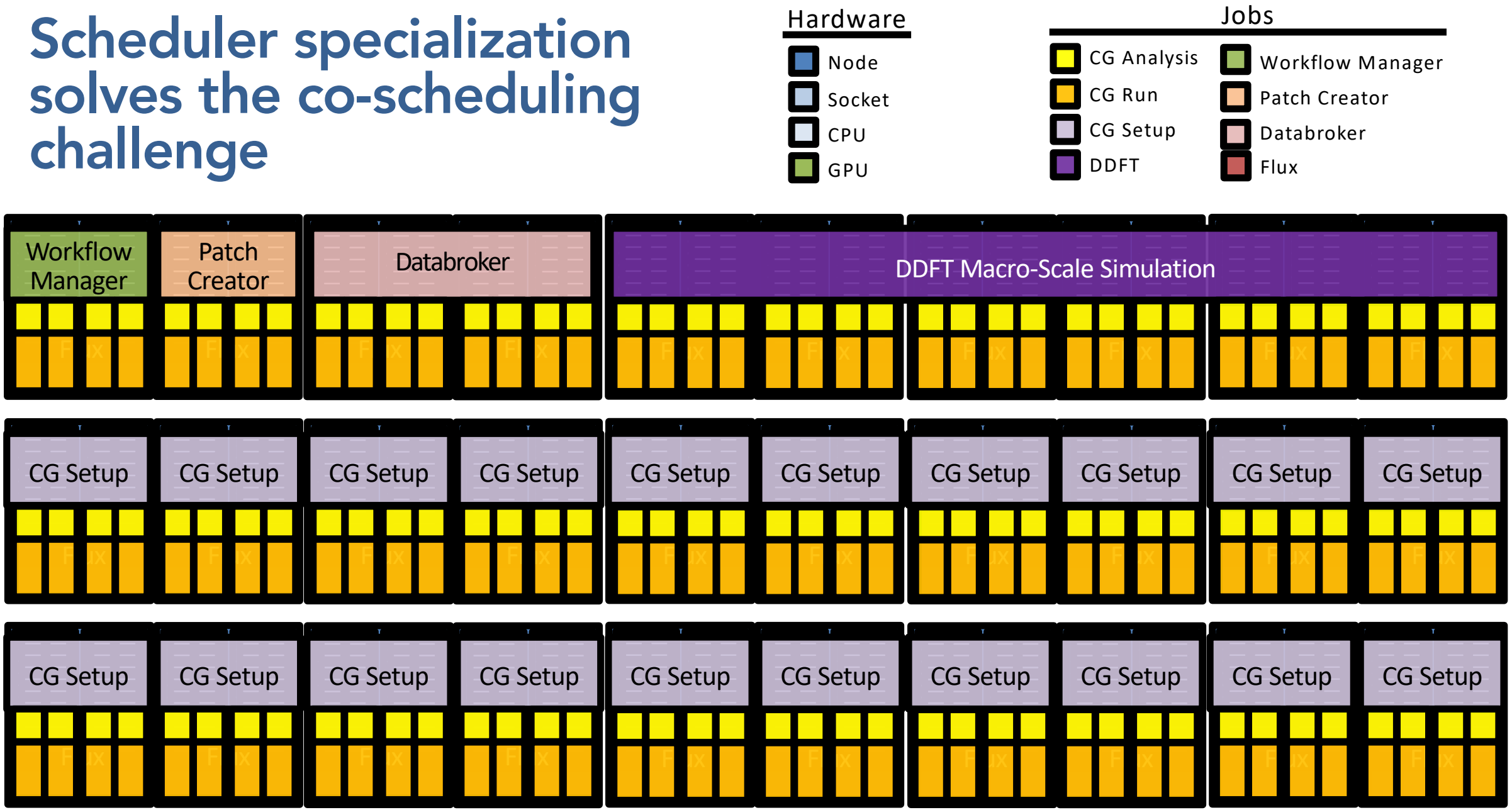




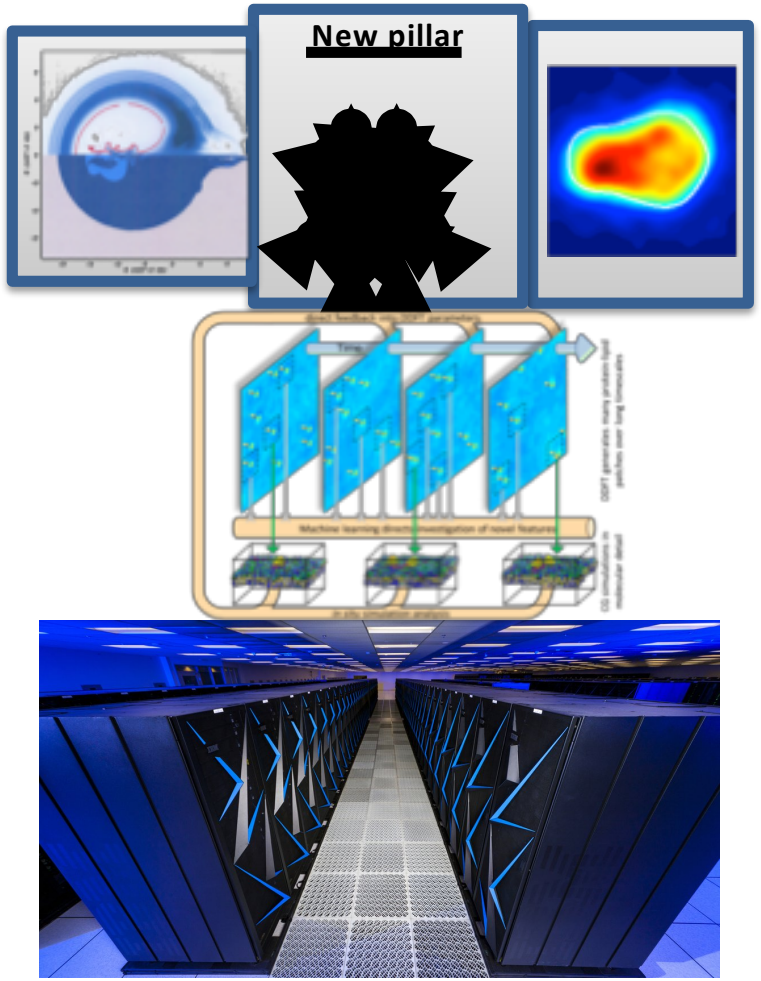




Scheduler specialization solves the co-scheduling challenge



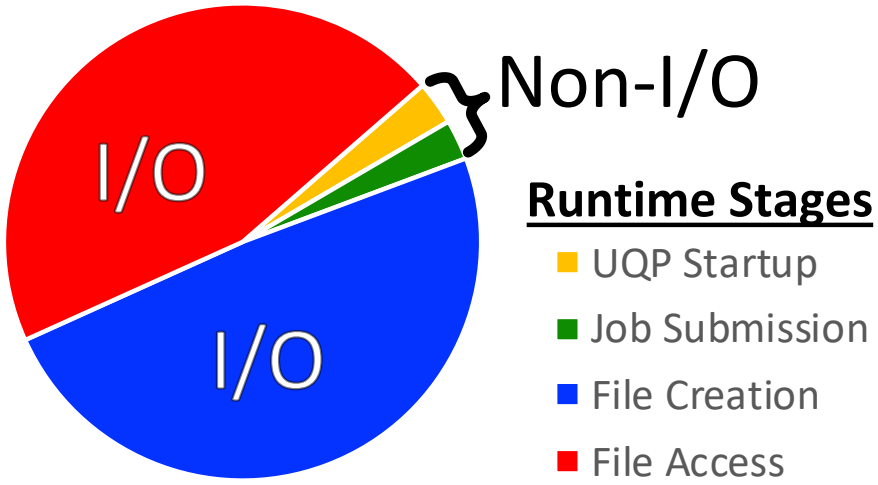
Key challenges in emerging workflow scheduling include…



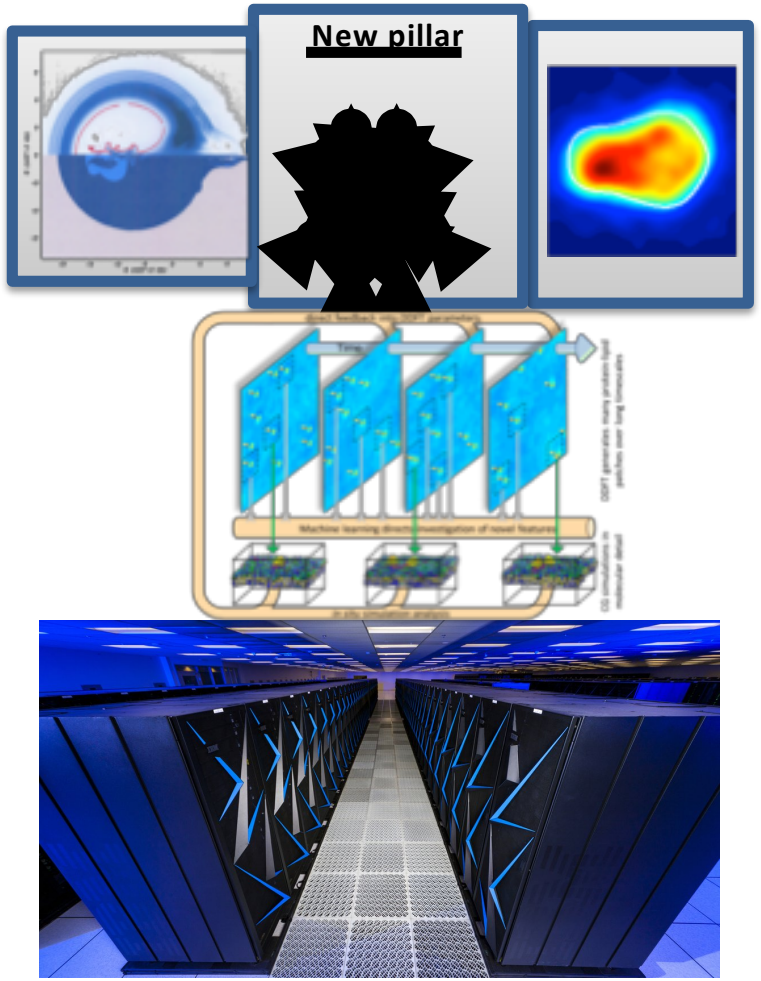
- ✓ Job throughput challenge
- ✓ Co-scheduling challenge
- Job communication/coordination challenge
- Portability challenge

A rich API set 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 is too 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 status/control (JSC) API
  - KZ stdout/stderr stream API



Key challenges in emerging workflow scheduling include…



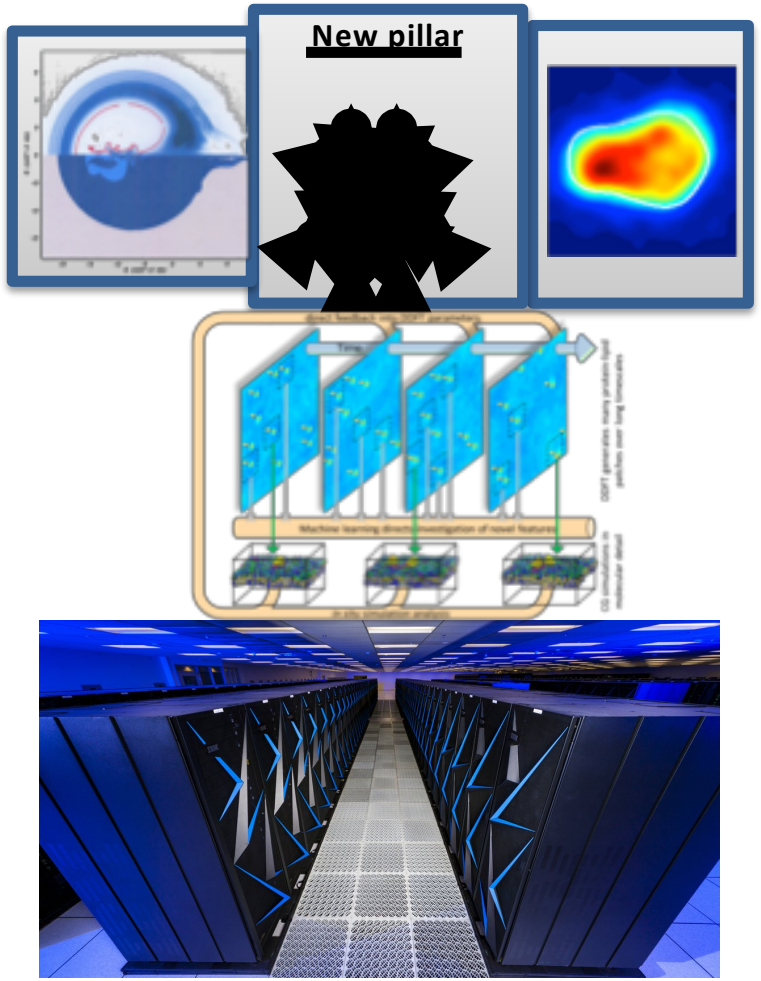
- Job throughput challenge
- Co-scheduling challenge
- Job communication/coordination challenge
- Portability challenge

A consistent API set facilitates high portability

- Flux's APIs are consistent across different platforms
- Flux can run anywhere that MPI can run (via PMI – Process Management Interface)
  - Inside a resource allocation from: itself (hierarchical Flux), Slurm, Moab, PBS, LSF, etc
  - `flux start` **OR** `srun flux start`
- Flux can run anywhere that supports TCP and you have the IP addresses
  - `flux broker -Sboot.method=config -Sboot.config_file=boot.conf`
  - `boot.conf:`

```
session-id = "mycluster"
tbon-endpoints = [
  "tcp://192.168.1.1:8020",
  "tcp://192.168.1.2:8020",
  "tcp://192.168.1.3:8020"]
```

Key challenges in emerging workflow scheduling include…



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

Flux significantly enables emerging workflows on high-end HPC systems

- Ensemble-, coupling-based workflows are increasingly becoming a "norm" on high-end HPC systems and traditional approaches are hard-pressed.
- Four key challenges have been identified.
- Flux provides a new scheduling model and an implementation and API set that embody this model.
- Our case studies and performance evaluations suggest that our model can significantly address all of the challenges.
- Flux is powering up the production runs of the major science runs on LLNL's Sierra, pre-exascale system.

Current Milestones

- Research
  - Building a model to predict the best hierarchy for a given workflow and resource allocation
  - Integration with Kubernetes and other cloud schedulers
  - Proposing a new model for workflow exception handling: Multi-level Chained Exception Model
- Production
  - Standardizing interfaces between the resource manager and other entities: users, parallel runtimes, debuggers/tools, administrators, etc.
  - Deploying to LLNL users on a new “Big Data” cluster
  - Developing a flux-bootstrap command to easily launch under other system schedulers using Flux’s job specification language

Resources

- flux-core:https://github.com/flux-framework/flux-core
- flux-sched: https://github.com/flux-framework/flux-sched
- Fully hierarchical scheduler: https://github.com/flux-framework/flux-hierarchy
- Workflow examples: https://github.com/flux-framework/flux-workflow-examples
- Quick guide: https://github.com/flux-framework/flux-framework.github.io
- Mailing list: flux-discuss@listserv.llnl.gov