Balsam Workflows

balsam.readthedocs.io

Misha Salim
Argonne Leadership Computing Facility
msalim@anl.gov

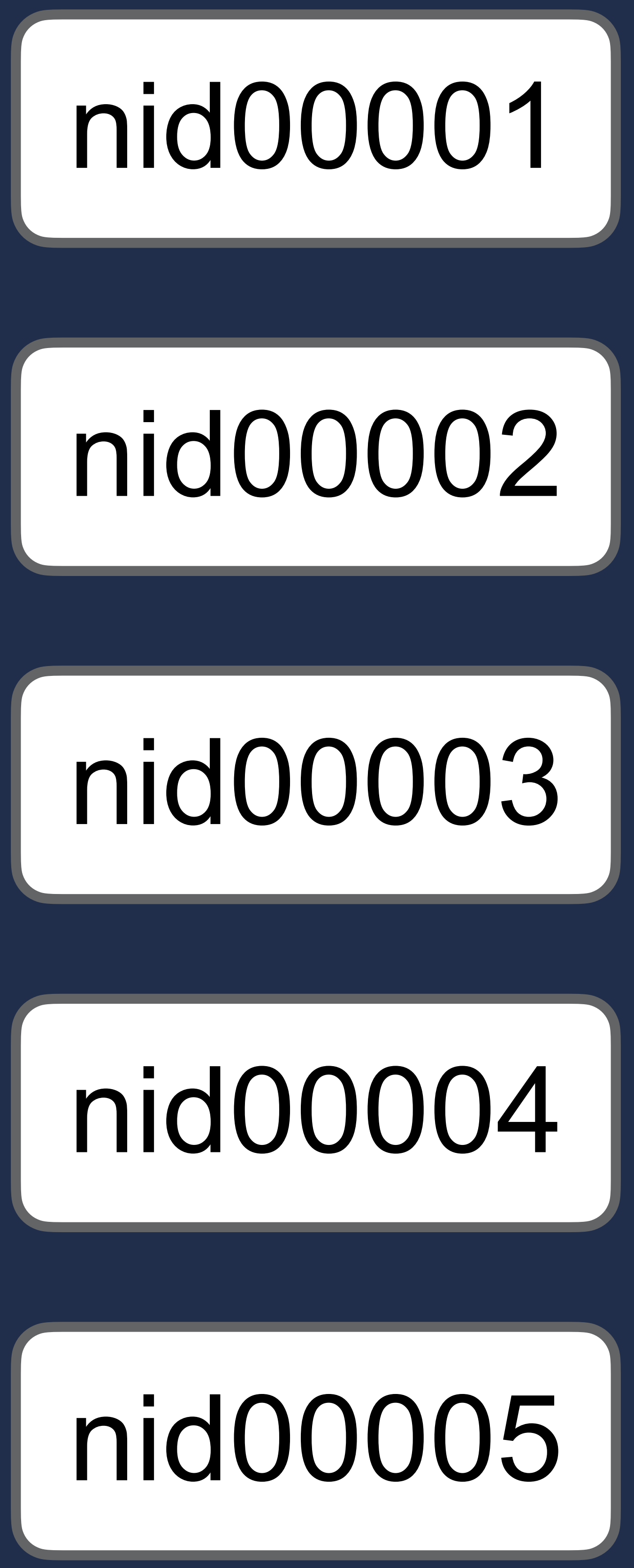
Ensemble Jobs at Argonne LCF

```
#!/bin/bash

myApp="/path/to/app --input="
```

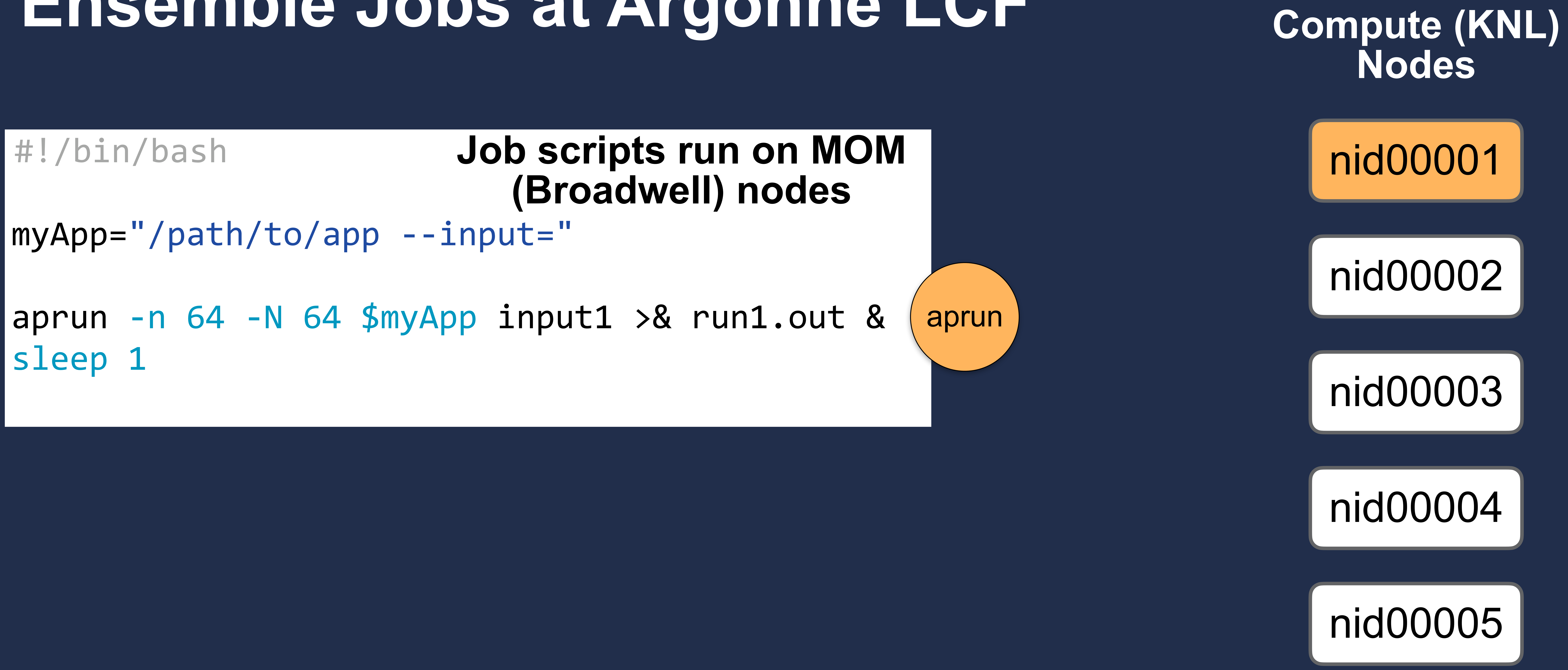
Job scripts run on MOM (Broadwell) nodes

Compute (KNL) Nodes



alcf.anl.gov/user-guides/running-jobs-xc40#bundling-multiple-runs-into-a-script-job

Ensemble Jobs at Argonne LCF



```
#!/bin/bash

myApp="/path/to/app --input="

aprun -n 64 -N 64 $myApp input1 >& run1.out &
sleep 1
```

Job scripts run on MOM (Broadwell) nodes

aprun

Compute (KNL) Nodes

nid00001

nid00002

nid00003

nid00004

nid00005

alcf.anl.gov/user-guides/running-jobs-xc40#bundling-multiple-runs-into-a-script-job

Ensemble Jobs at Argonne LCF

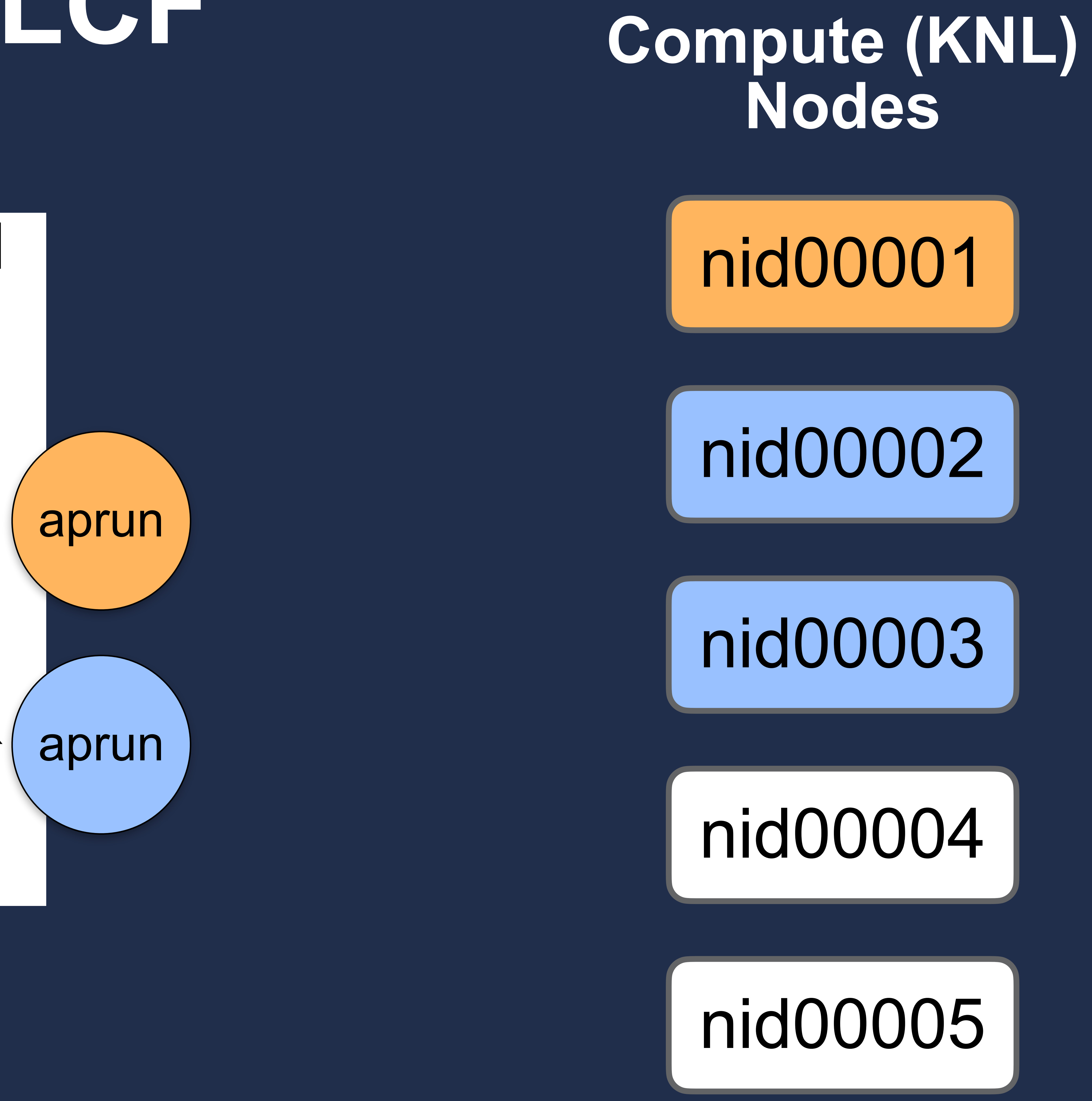
Job scripts run on MOM (Broadwell) nodes

```
#!/bin/bash

myApp="/path/to/app --input="

aprun -n 64 -N 64 $myApp input1 >& run1.out &
sleep 1

aprun -n 128 -N 64 $myApp input2 >& run2.out &
sleep 1
```



alcf.anl.gov/user-guides/running-jobs-xc40#bundling-multiple-runs-into-a-script-job

Ensemble Jobs at Argonne LCF

Job scripts run on MOM (Broadwell) nodes

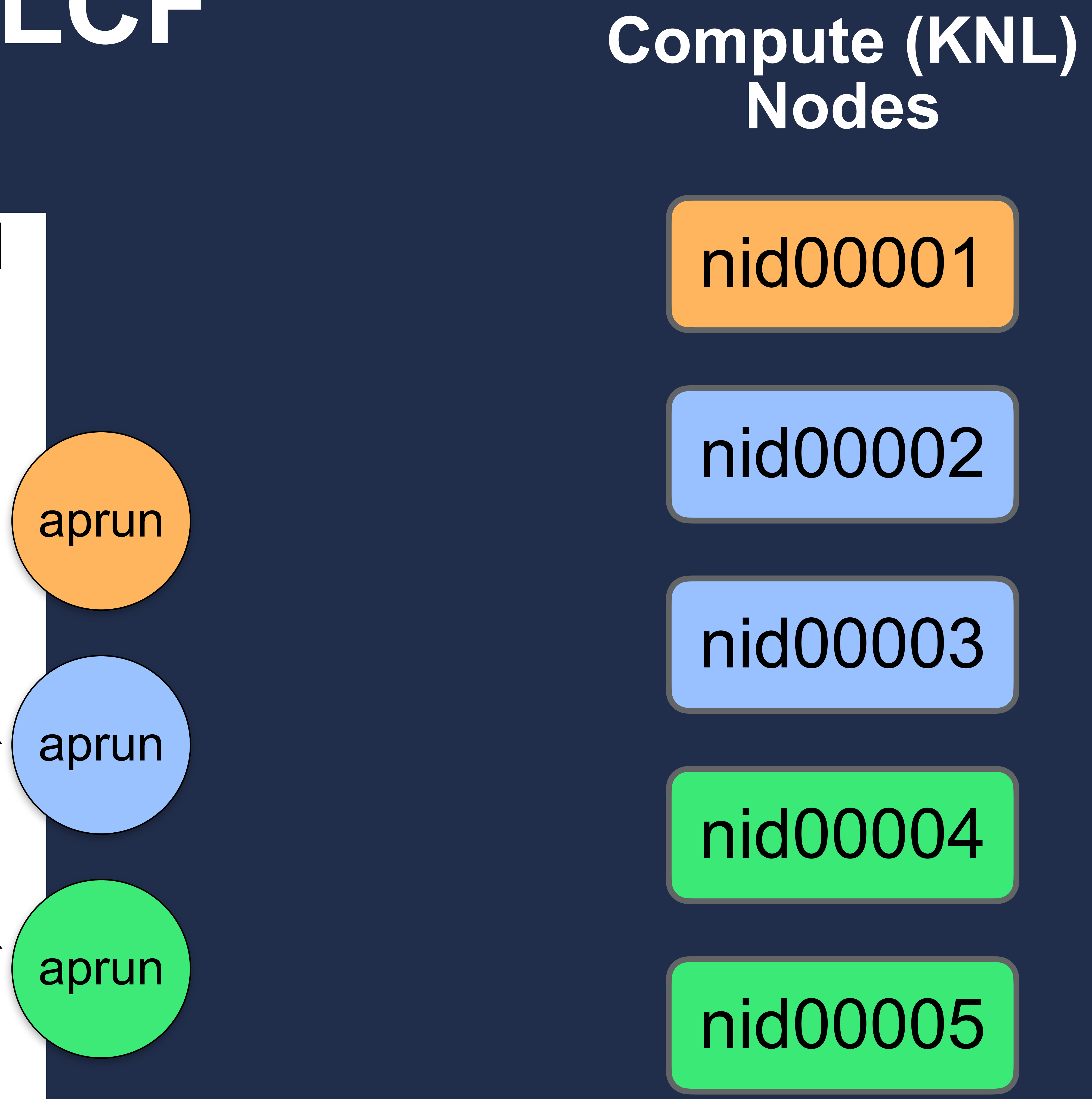
```
#!/bin/bash

myApp="/path/to/app --input="

aprun -n 64 -N 64 $myApp input1 >& run1.out &
sleep 1

aprun -n 128 -N 64 $myApp input2 >& run2.out &
sleep 1

aprun -n 128 -N 64 $myApp input3 >& run3.out &
wait
```

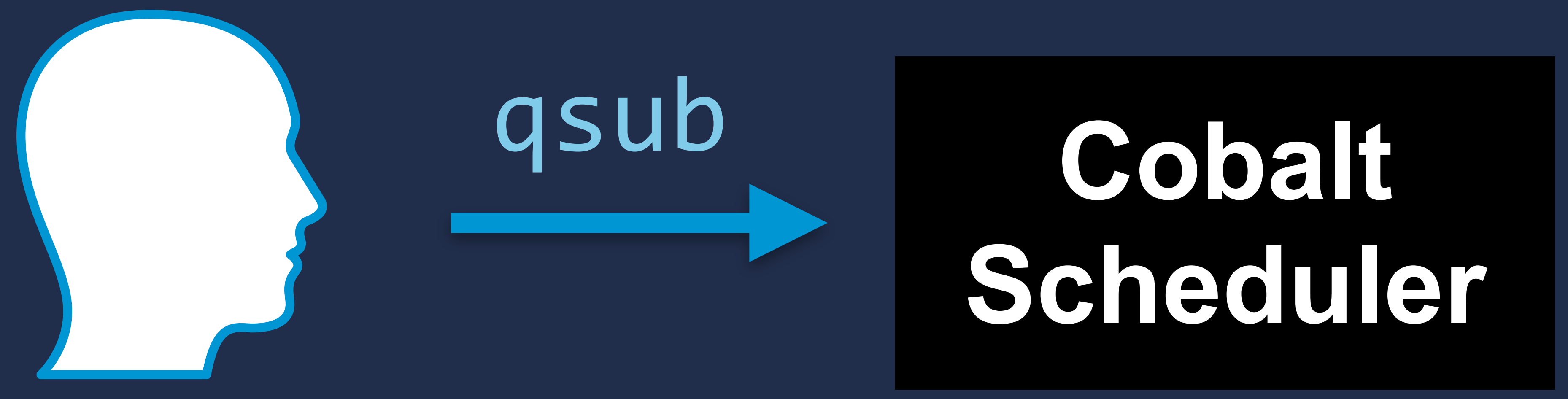


alcf.anl.gov/user-guides/running-jobs-xc40#bundling-multiple-runs-into-a-script-job

What do we mean by workflow?

Sometimes a few scripts is enough

```
(100 runs) (1024 nodes) (12 hours) = 1.23 M node-hours
```



qsub

Cobalt Scheduler

- Queue up to 20 script jobs
- Keep organized directory layout
- Compose shell commands with bash or Python scripting

What do we mean by workflow?

Sometimes a few scripts is enough

`(100 runs) (1024 nodes) (12 hours) = 1.23 M node-hours`

Large ensembles: start building more complex workflows

`(9600 runs) (128 node) (1 hour) = 1.23 M node-hours`

- Run jobs concurrently ***and*** one-after-another?
- Track which tasks are left to run?
- Handle timed-out runs?

What do we mean by workflow?

Sometimes a few scripts is enough

`(100 runs) (1024 nodes) (12 hours) = 1.23 M node-hours`

Large ensembles: start building more complex workflows

`(9600 runs) (128 node) (1 hour) = 1.23 M node-hours`

Human effort scales unfavorably with # of runs

`(12,288,000 runs) (1 node) (6 minutes) = 1.23 M node-hours`

What do we mean by workflow?

Max 20 queued jobs

Lacking job packing / MPMD execution

Cumbersome error & timeout handling

Human effort scales unfavorably with # of runs

(12,288,000 runs) (1 node) (6 minutes) = 1.23 M node-hours

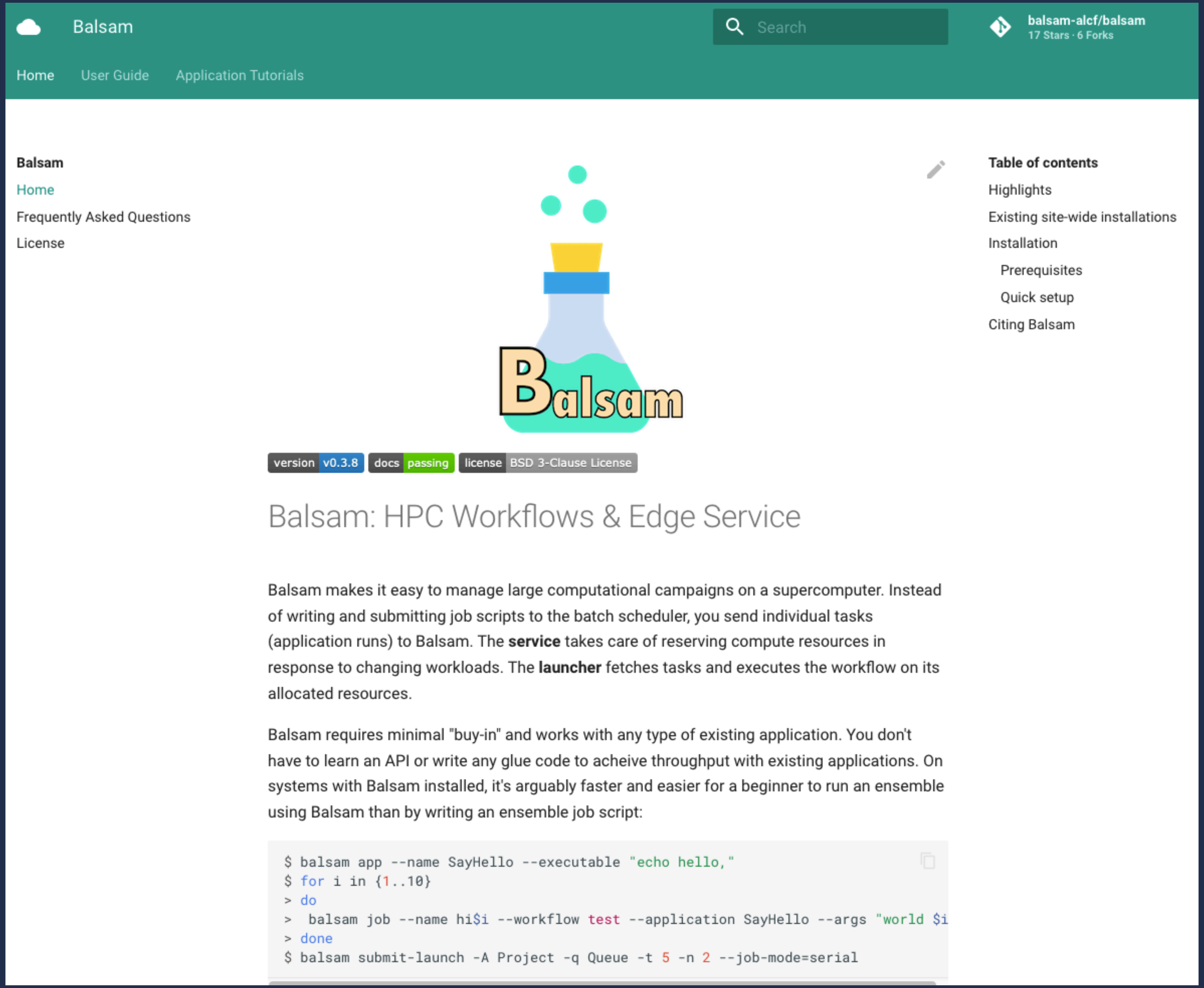
You either build workflow tools or adopt existing ones

Balsam

Workflows, scheduling, and execution for HPC

- Submit unlimited application runs to a private task database
- **Service** component automates queue submission
- **Launcher** component pulls tasks for load-balanced execution
  - Resilient to task-level faults
  - Automatic retry or custom handling of timed-out, failed jobs
  - Runs **unmodified** user applications or Singularity containers
- Workflow status and project statistics available at-a-glance

Release in production @ ALCF



Balsam: HPC Workflows & Edge Service

Balsam makes it easy to manage large computational campaigns on a supercomputer. Instead of writing and submitting job scripts to the batch scheduler, you send individual tasks (application runs) to Balsam. The **service** takes care of reserving compute resources in response to changing workloads. The **launcher** fetches tasks and executes the workflow on its allocated resources.

Balsam requires minimal "buy-in" and works with any type of existing application. You don't have to learn an API or write any glue code to acheive throughput with existing applications. On systems with Balsam installed, it's arguably faster and easier for a beginner to run an ensemble using Balsam than by writing an ensemble job script:

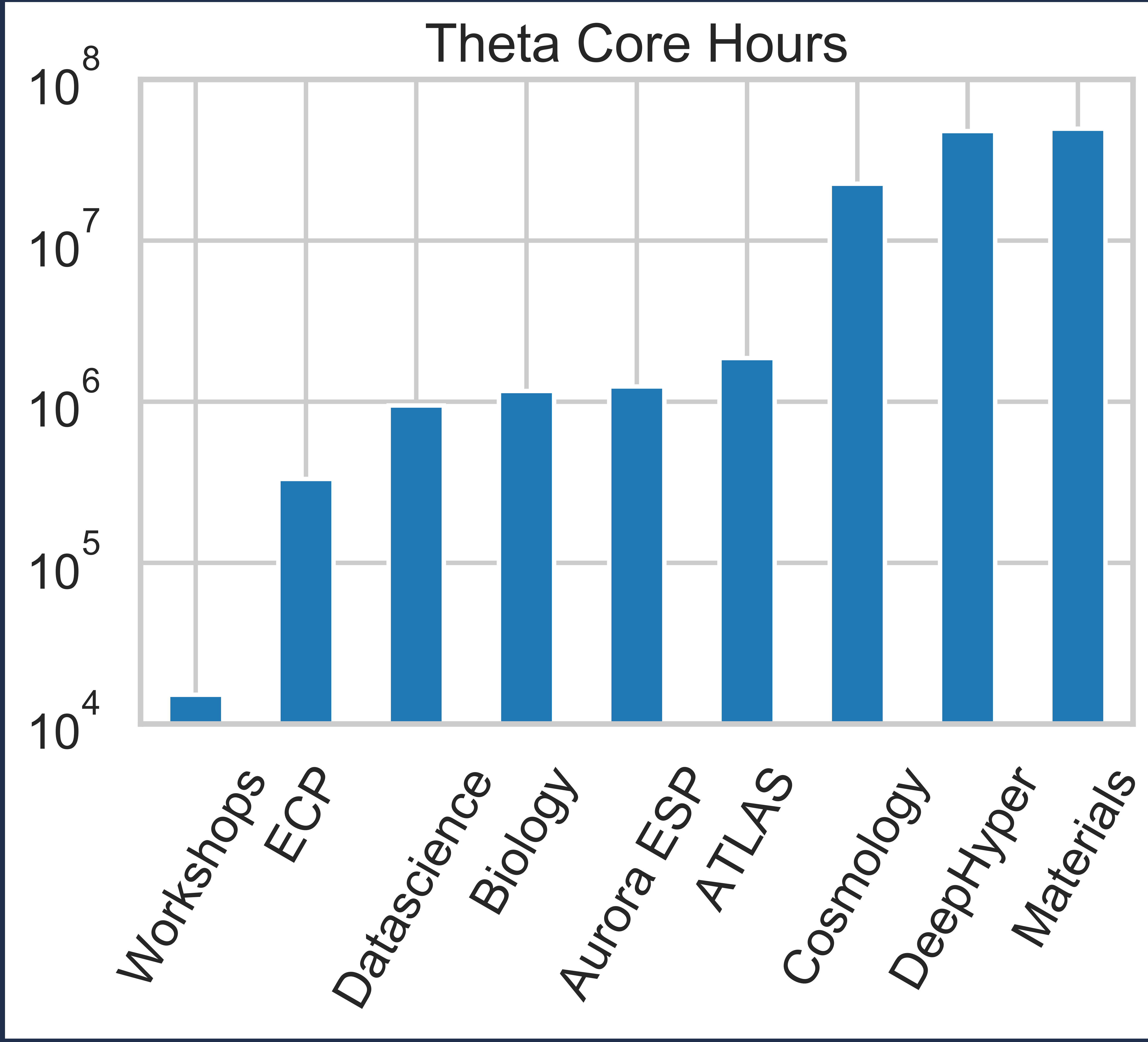
```
$ balsam app --name SayHello --executable "echo hello,"
$ for i in {1..10}
> do
>  balsam job --name hi$i --workflow test --application SayHello --args "world $i
> done
$ balsam submit-launch -A Project -q Queue -t 5 -n 2 --job-mode=serial
```

balsam.readthedocs.io

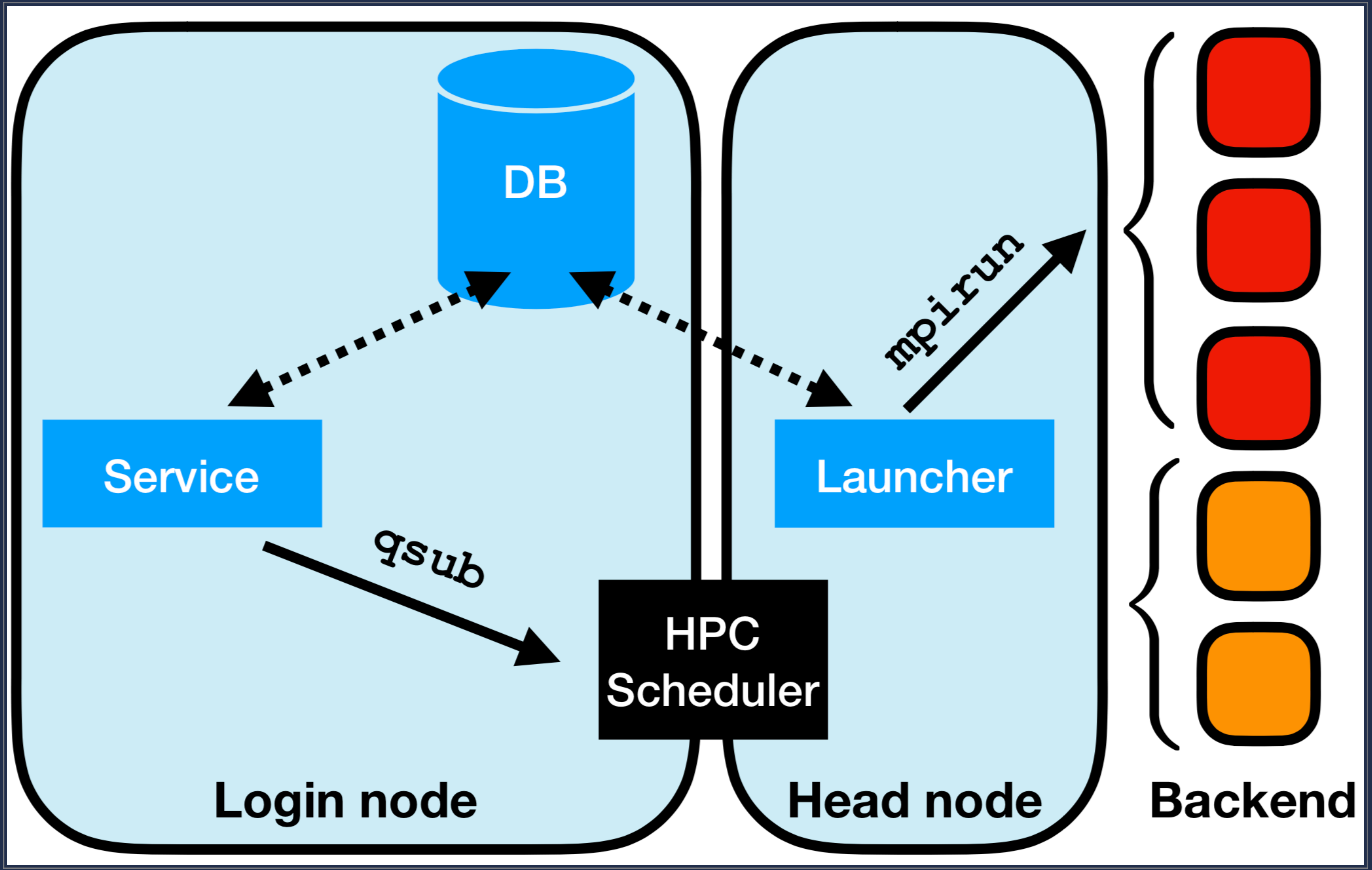
Tracking Balsam Usage

(September 2018 -- 2019)

- 125M Theta core-hours
- 48 users
- 28 projects
- Top usage categories:
  - Materials Science (39%)
  - DeepHyper (38%)
  - Cosmology (18%)



A quick look at Balsam components



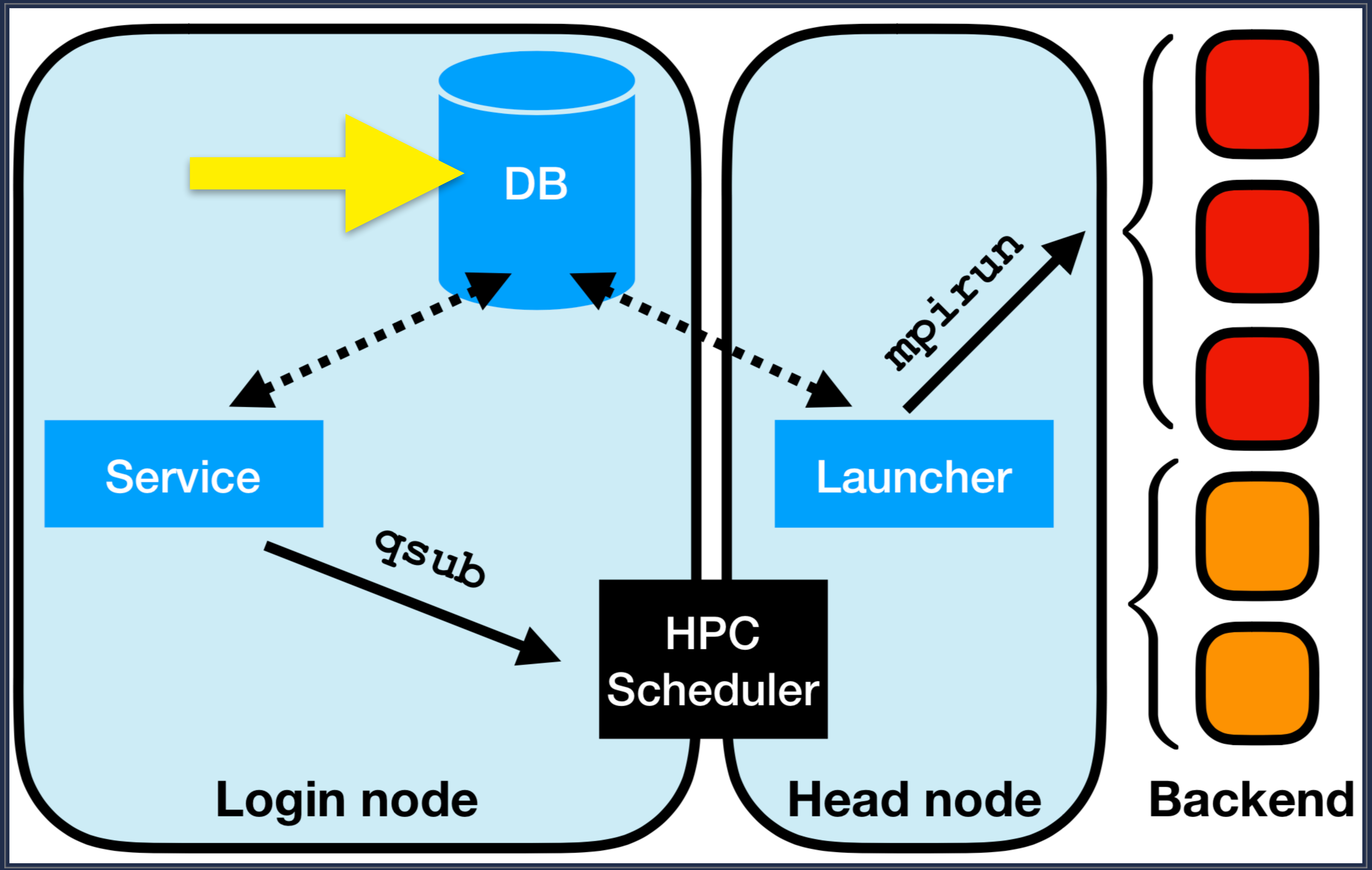
Database

BalsamJob table:

one row per task

One line setup:

```
balsam init myproject
```

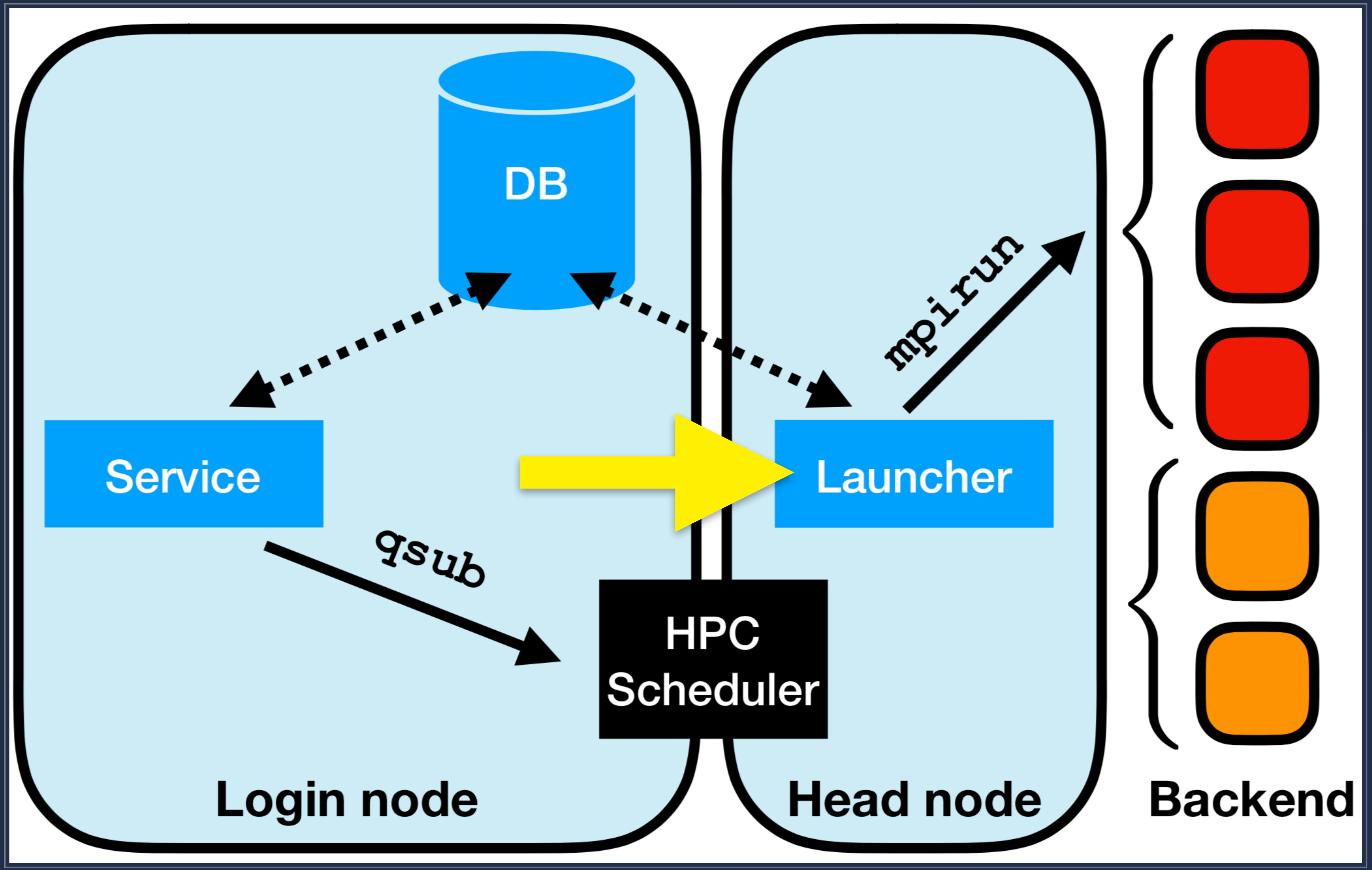


Launcher

Dynamic task pull and execution

MPI job mode for conventional app launch
(1 aprun per task)

Serial job mode to pack many tasks per node
(1 aprun: `mpi4py` runtime)



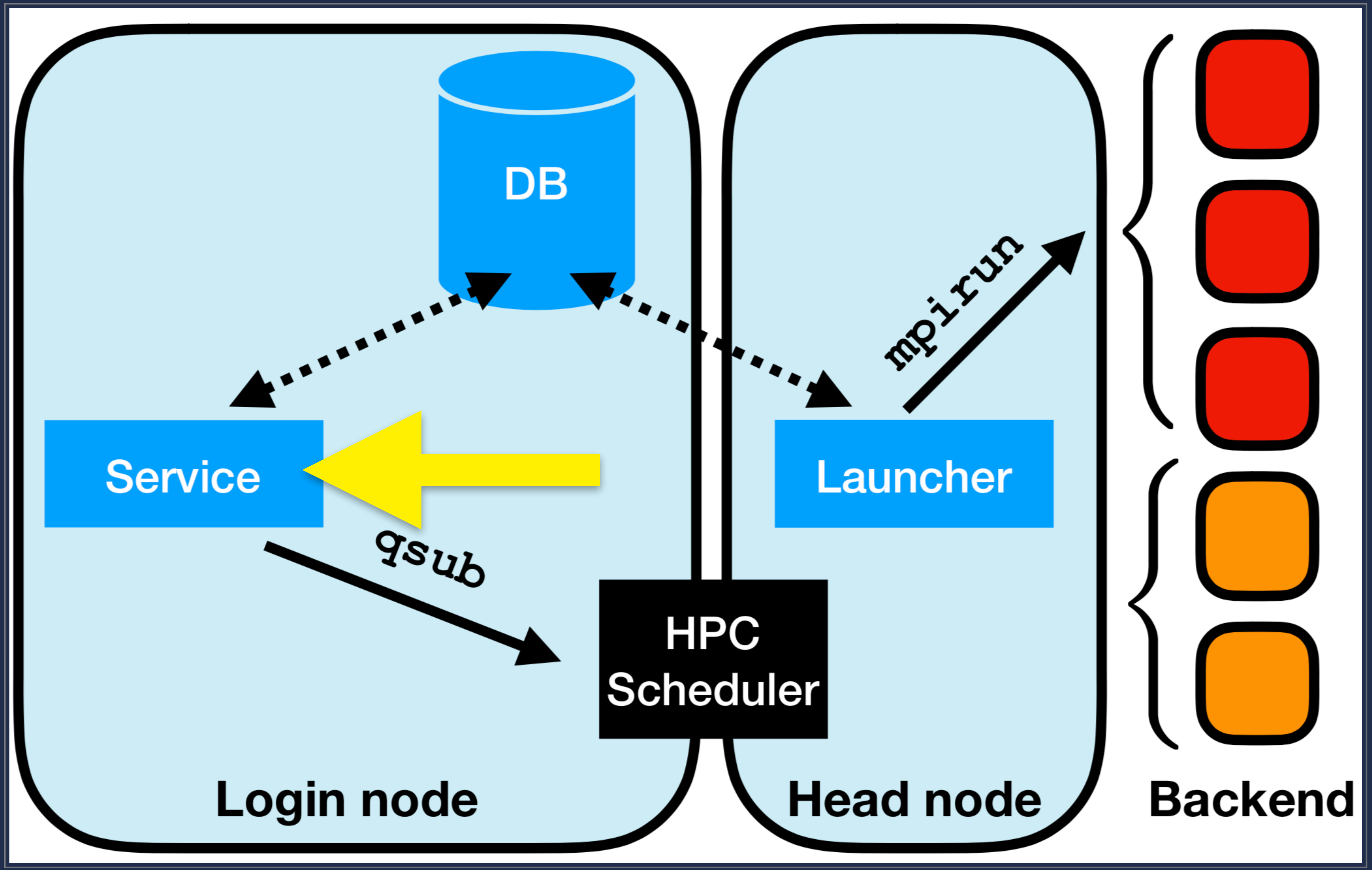
Service

Submission interface

```
balsam submit-launch
```

Auto queue submission

```
balsam service
```



Complementary job modes needed to work around various limitations on ALCF Theta platform

	MPI job mode (1 aprun per task)	Serial job mode (1 fork/exec per task)
Why?	No support for MPI_Comm_spawn Cannot use alternative MPI launchers or launch jobs from compute nodes	Theta aprun does not permit multiple apps per node: this is often wasteful in data-intensive workflows
Abilities	Can run any kind of application; good isolation between apps	1 MPI rank per node "packs" multiple tasks (BalsamJob `node_packing_count`) and manages CPU affinity via subprocess/ psutil Task prefetch and bulk DB updates: high efficiency even at 2k nodes
Limitations	Max ~980 concurrent aprun on Theta No support for multi-apps per node Overhead: 10ms sleep between launches	Cannot run apps that invoke MPI_Init, even if intended to run on single node

A Typical Workflow

Populate database with runs, then track progress

A Typical Workflow

1. Populate database from script

```
def prep_job(name, workflow, xyz_path):
    return BalsamJob(
        name = name,
        workflow = workflow,
        stage_in_url = xyz_path,
        application = 'fhi-aims',
        ranks_per_node = 64,
        threads_per_rank = 1,
        cpu_affinity = 'depth',
    )
```

A Typical Workflow

1. *Populate database from script*

```
for (dirpath, dirnames, filenames) in os.walk(top):
    xyz_files = [f for f in filenames if f.endswith('.xyz')]
    for f in xyz_files:
        name, _ = os.path.splitext(f)
        workflow = os.path.basename(dirpath)
        xyz_path = os.path.join(dirpath, f)
        job = prep_job(name, workflow, xyz_path)
        job.save()
```

A Typical Workflow

2. Request compute nodes

`balsam submit-launch :`

Shortcut for Cobalt job submission

```
[BalsamDB: myProject] $  balsam submit-launch -n 2 -t 10 \
   -q debug-cache-quad -A MyAllocation --job-mode mpi
```

A Typical Workflow

3. Track status of ongoing jobs

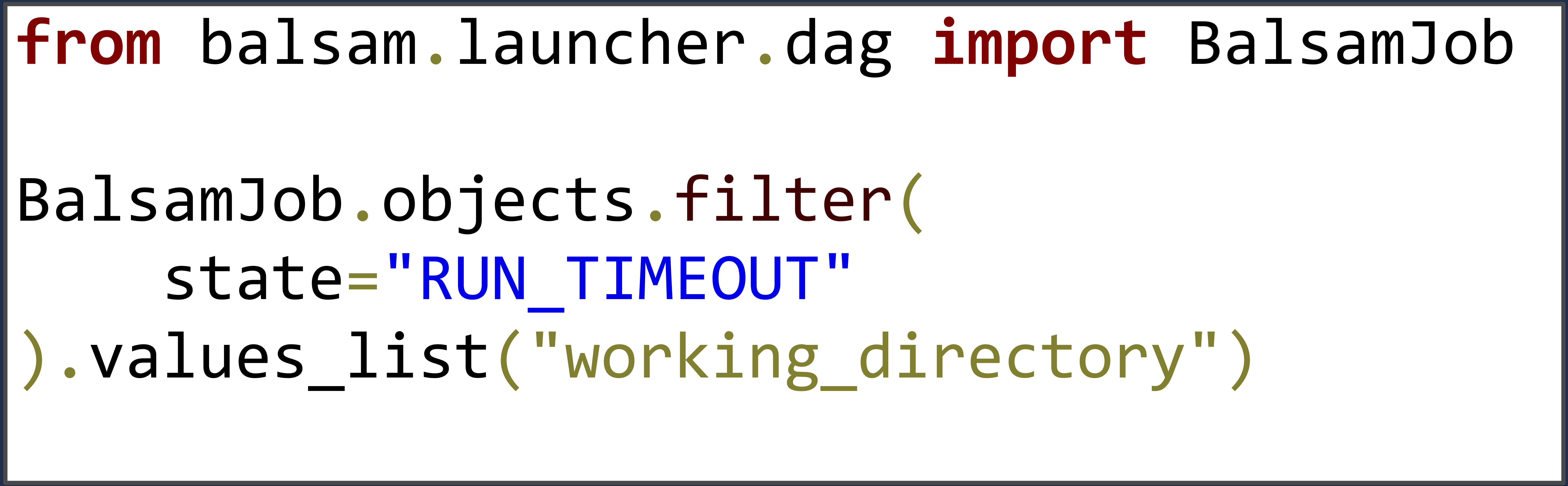
```
[BalsamDB: test-db] $ balsam ls --state FAILED  --history

Job testfail [fab575a3-01db-41b5-b70d-c396c17ef10d]
---------------------------------------------------


[10-03-2018 19:34:38.379895 CREATED]
[10-03-2018 19:38:24.490910 PREPROCESSED]
[10-03-2018 19:38:24.701099 RUNNING]
[10-03-2018 19:38:30.618931 RUN_ERROR]
Traceback (most recent call last):
   Hello from rank 2
   Hello from rank 1
      File "/gpfs/mira-home/msalim/test-db/fail.py", line 5
           raise RuntimeError("simulated error")
```

A Typical Workflow

Use Python API for more flexible queries

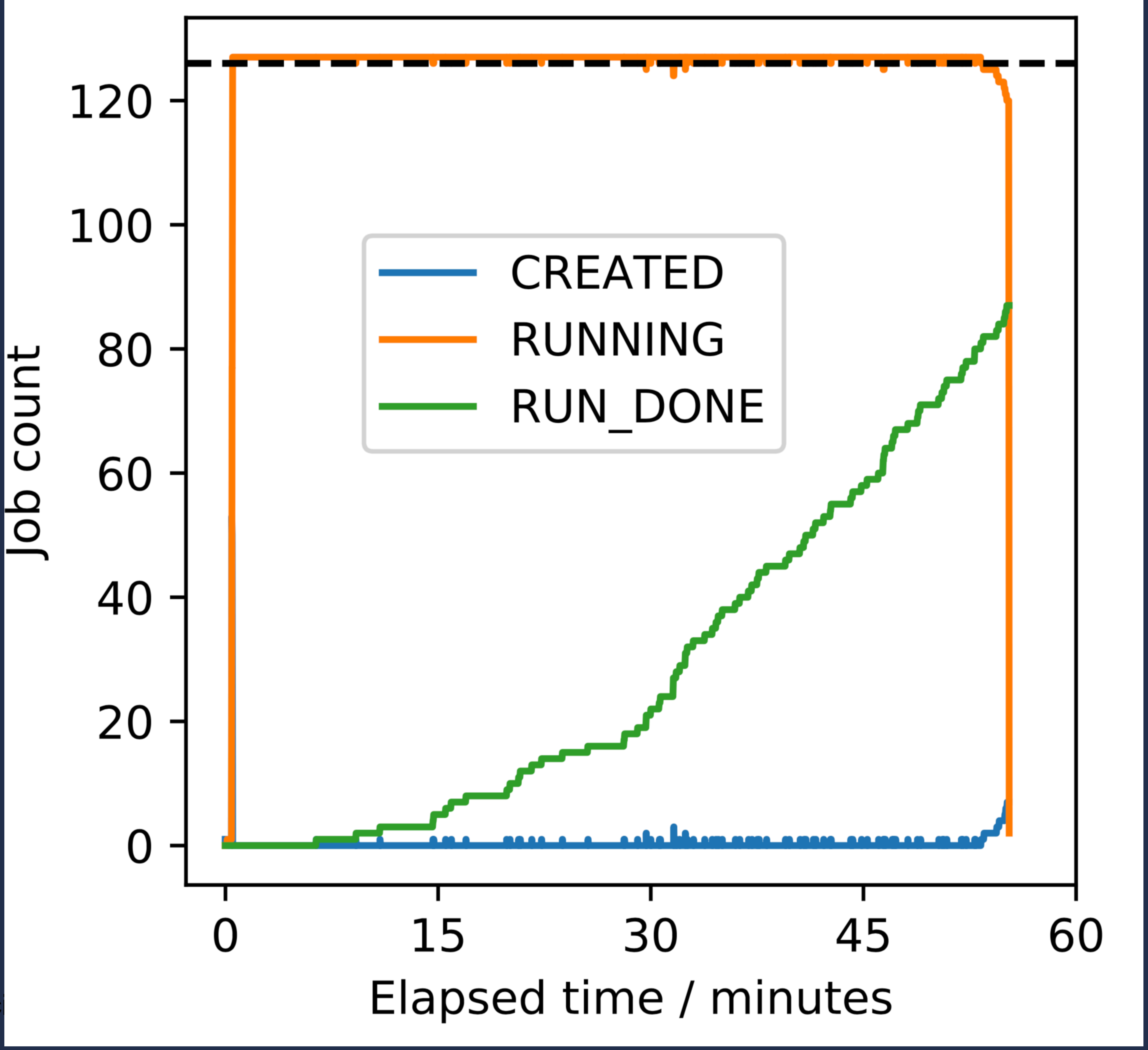


```python
from balsam.launcher.dag import BalsamJob

BalsamJob.objects.filter(
    state="RUN_TIMEOUT"
).values_list("working_directory")
```

A Typical Workflow

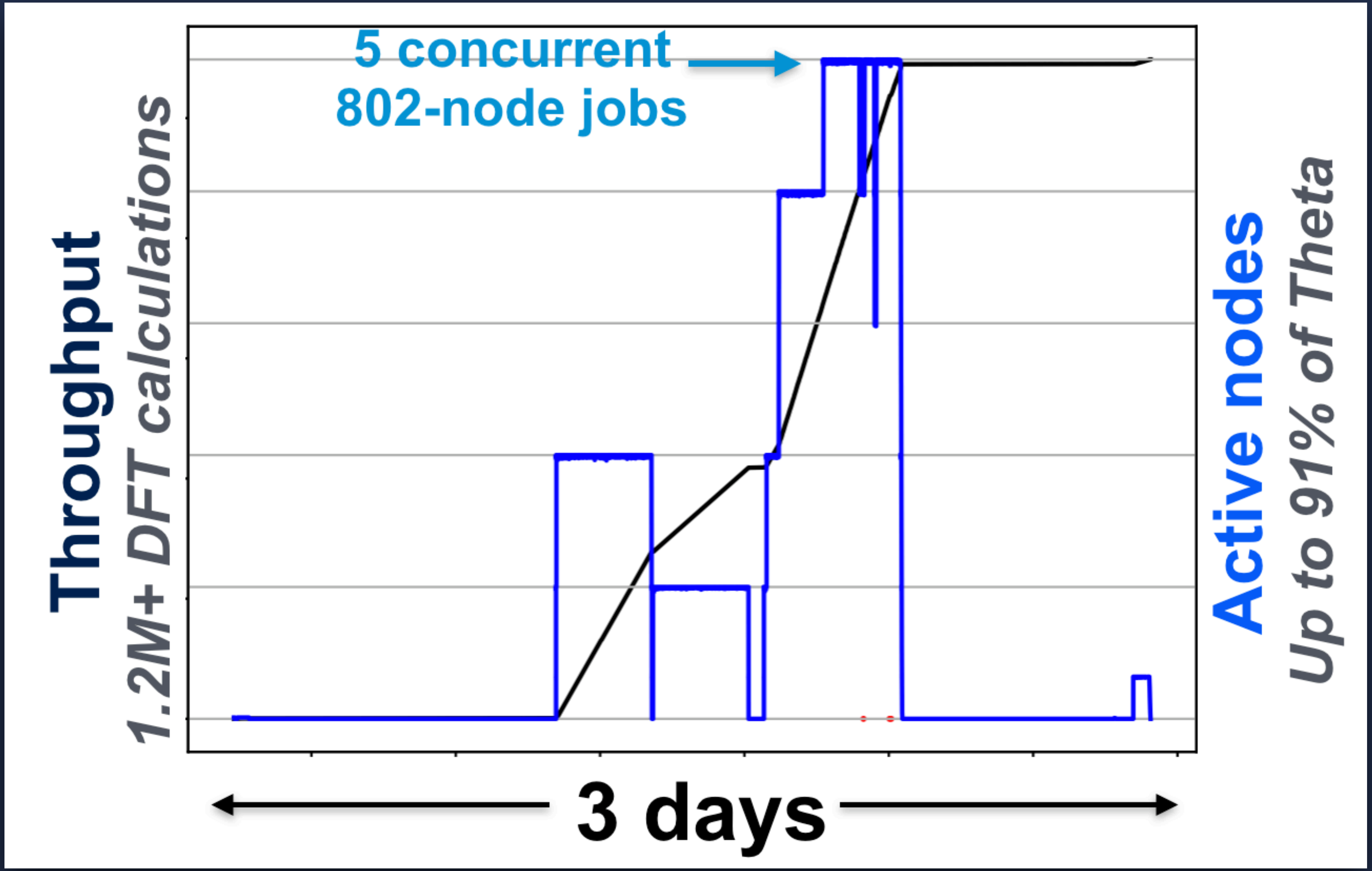
Convenience functions for visualizing throughput & utilization



Molecular Crystals ADSP

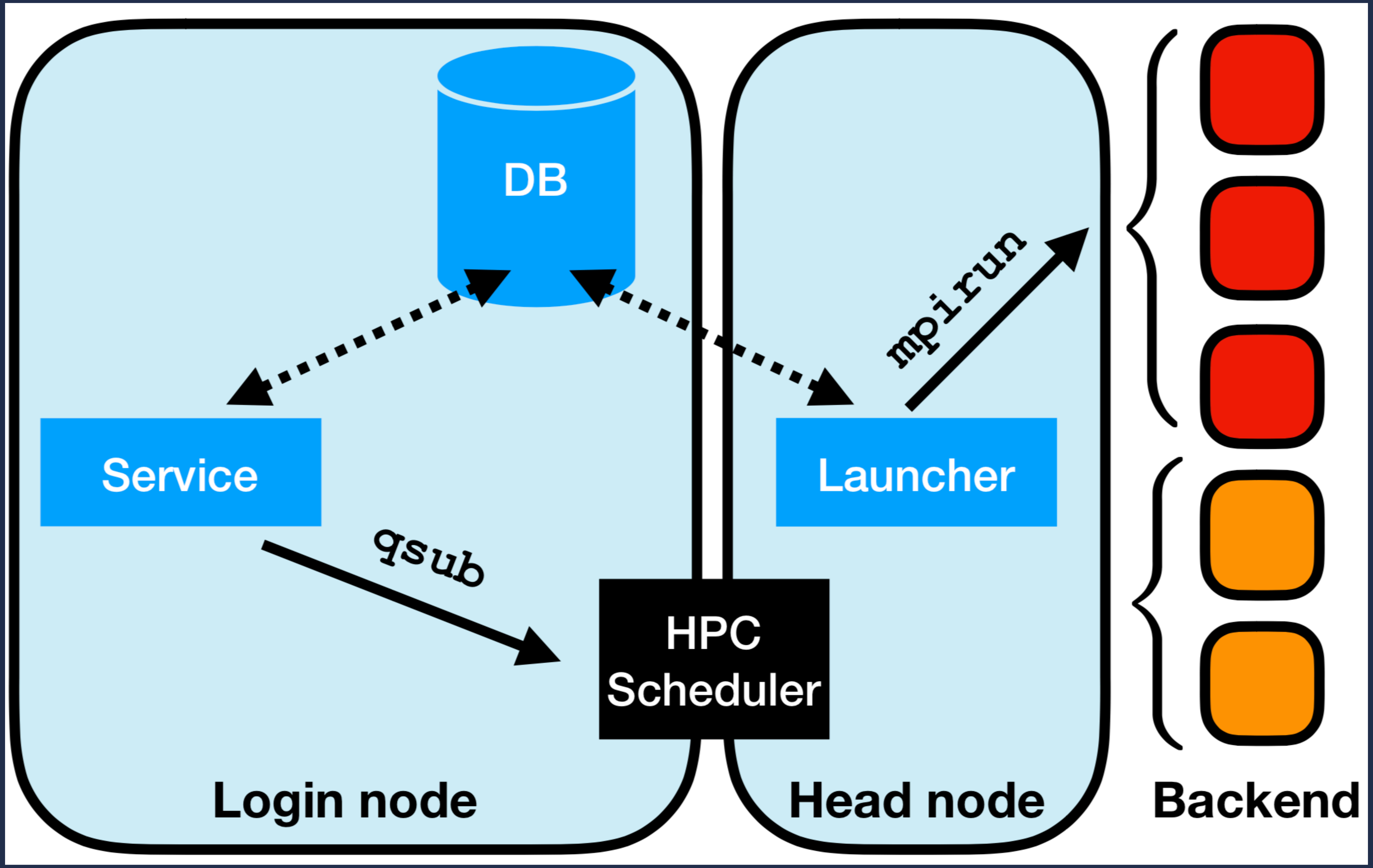
Cataloging free-energies of crystalline polymorphs (PI: Alexandre Tkatchenko)

- 22.9M core hours of DFT with FHI-AIMS
- Scaled to 91% of Theta, 1.2M+ tasks
- Up to 5 simultaneous Cobalt jobs running tasks from DB



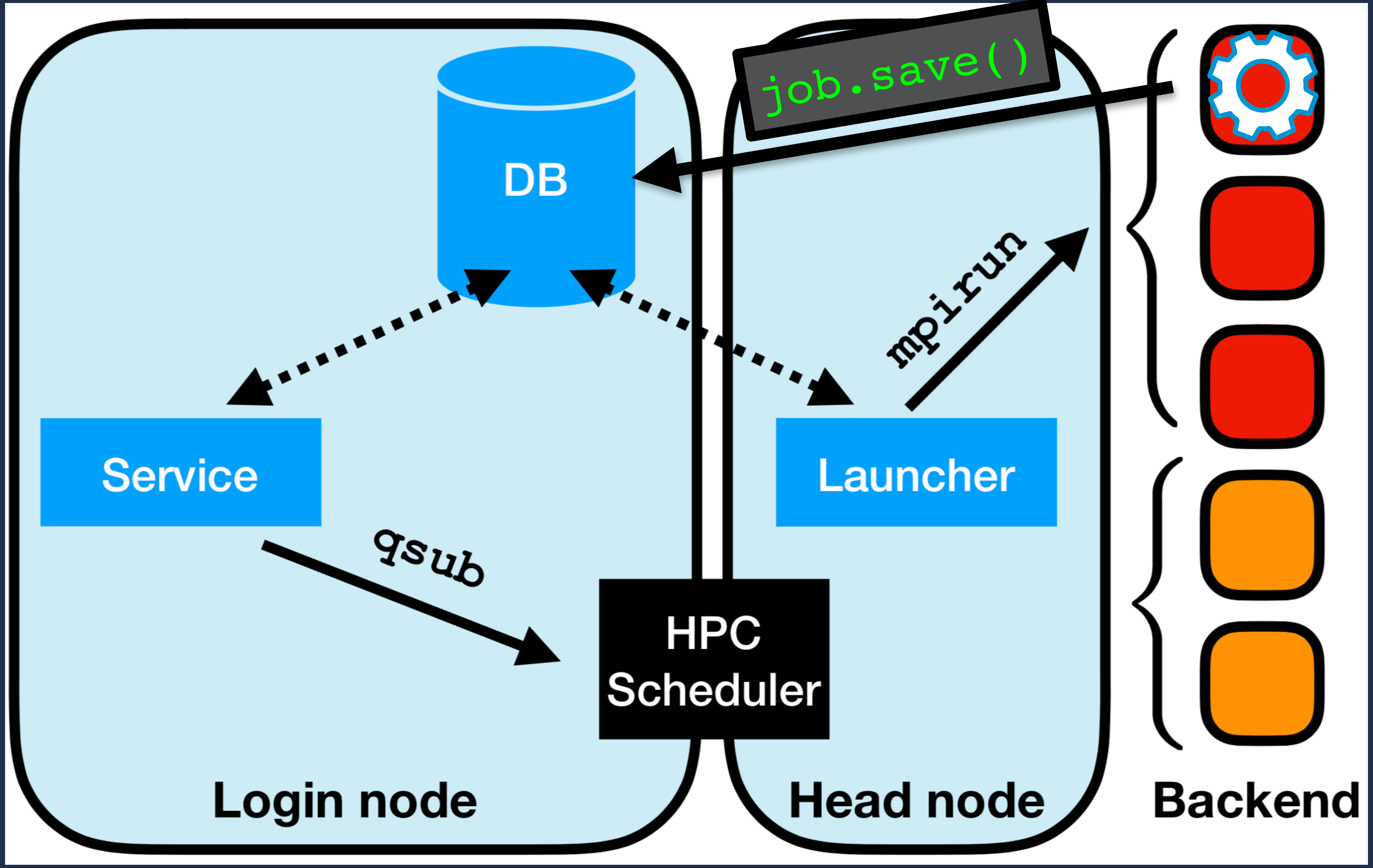
Dynamic Job Launch

Write applications that dynamically generate new runs from compute nodes



Dynamic Job Launch

Write applications that dynamically generate new runs from compute nodes



Dynamic Job Launch

Frameworks using Balsam for dispatching runs

DeepHyper

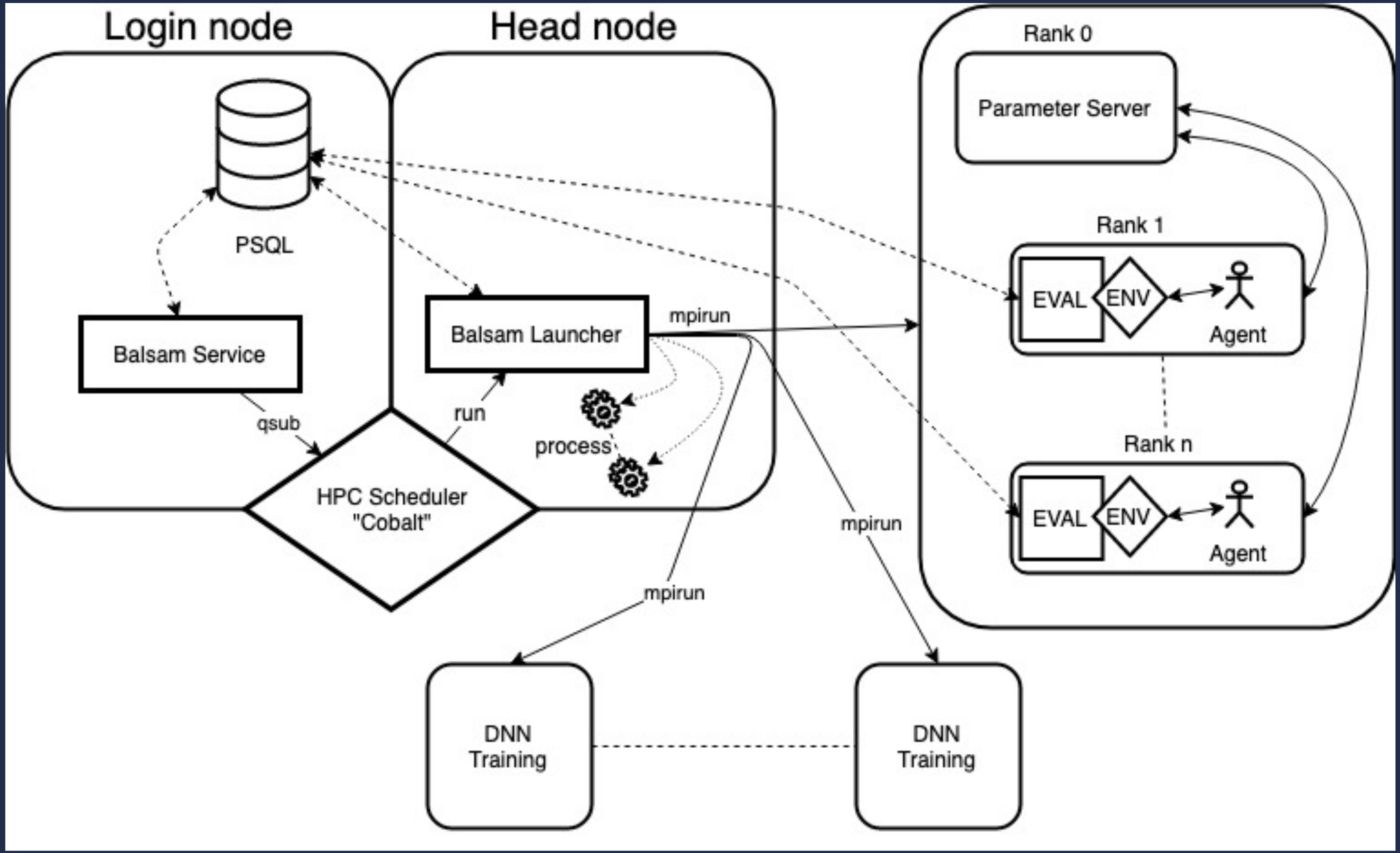
Hyperparameter Optimization and Neural Architecture Search

libEnsemble

Framework for Generator/simulator-type ensemble jobs

DeepHyper

Scalable reinforcement learning-based neural architecture search



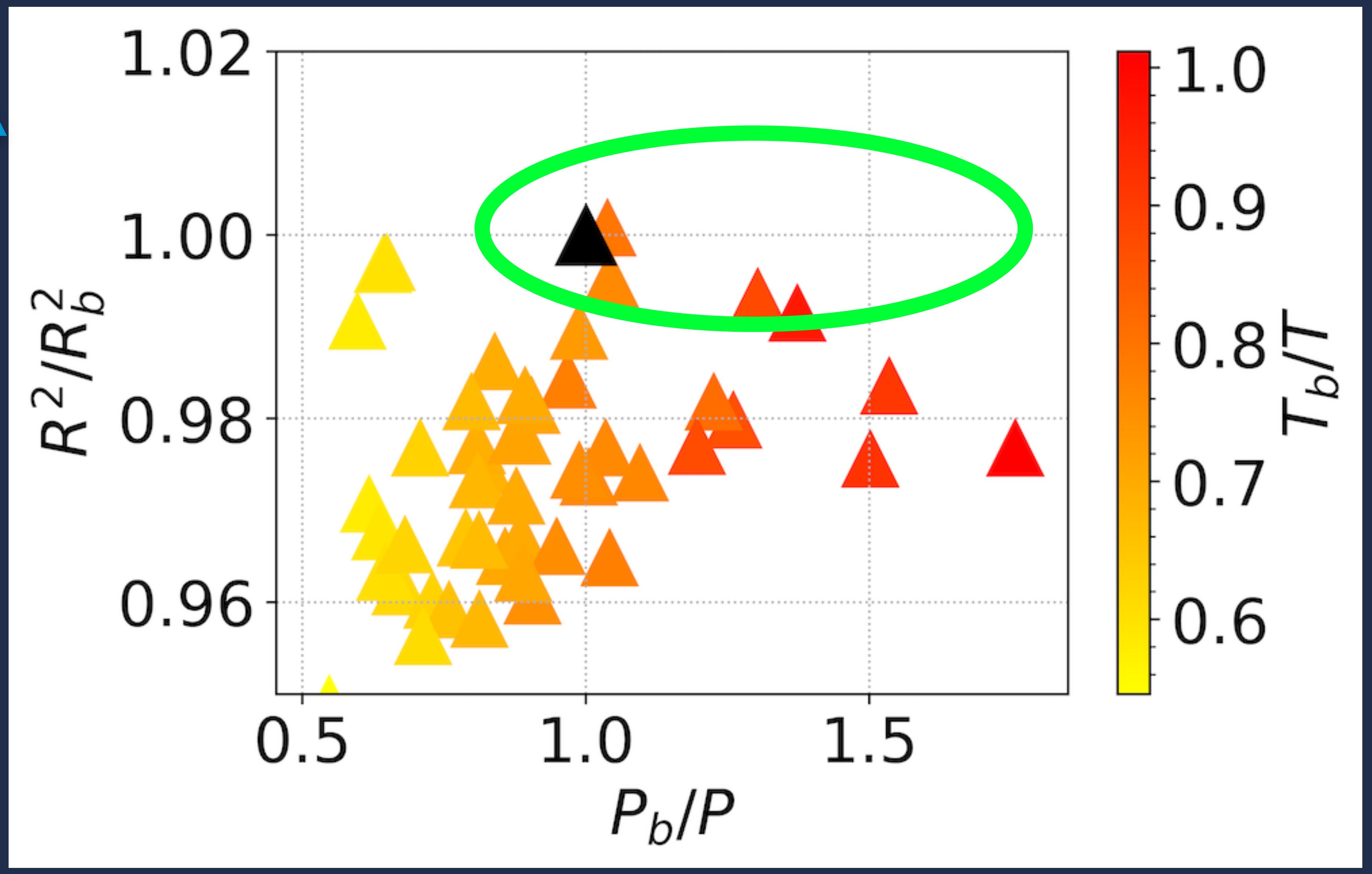
- Distributed RL with asynchronous advantage actor-critic (A3C) scheme
- RL agents send async. gradient updates to parameter server
- multiple workers (concurrent DNN model evaluations) per agent launched via Balsam serial job mode

DeepHyper

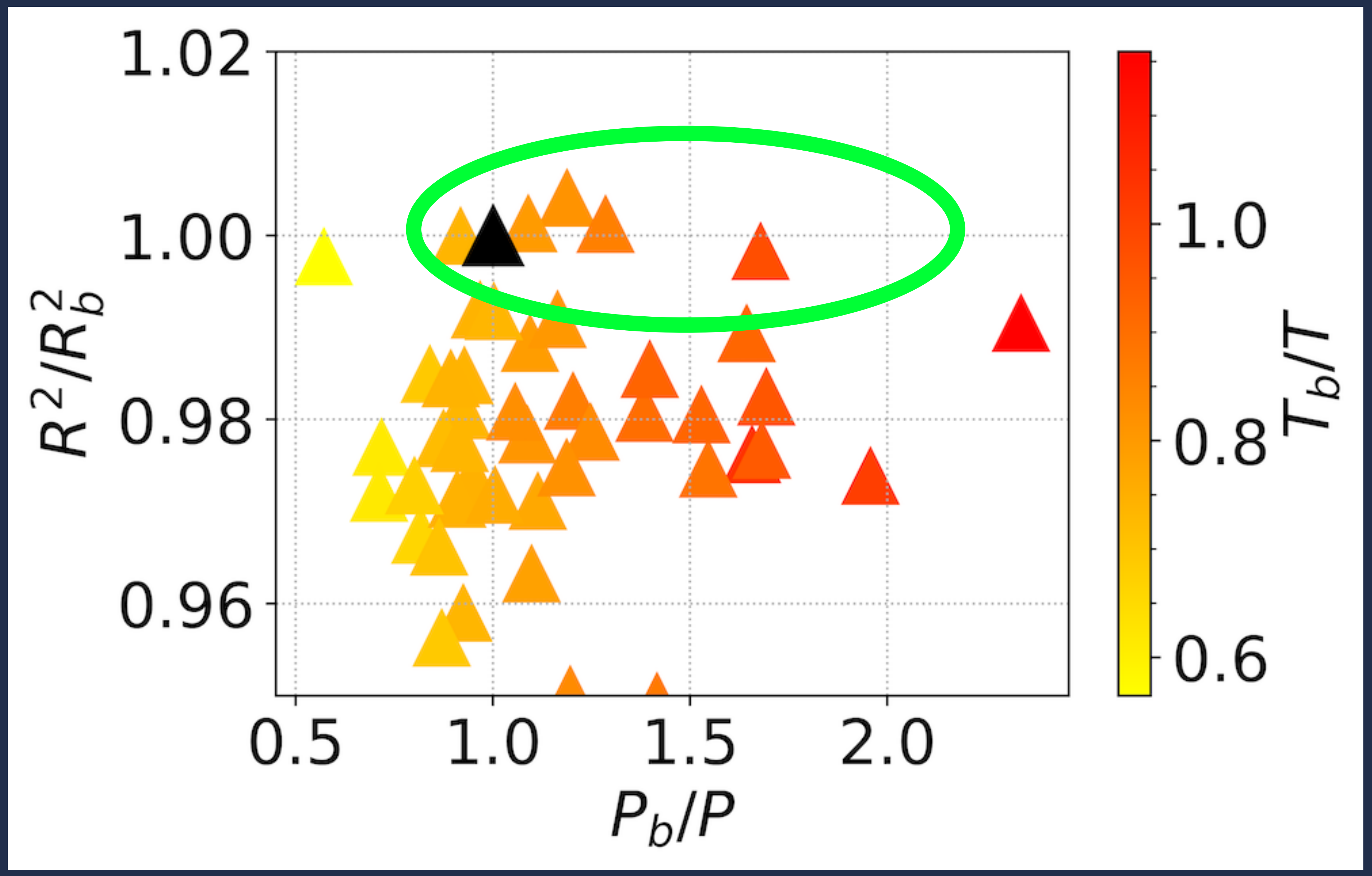
CANDLE Combo Benchmark: top 50 R^2 architectures selected for "post-training"

Fewer Parameters

Higher accuracy



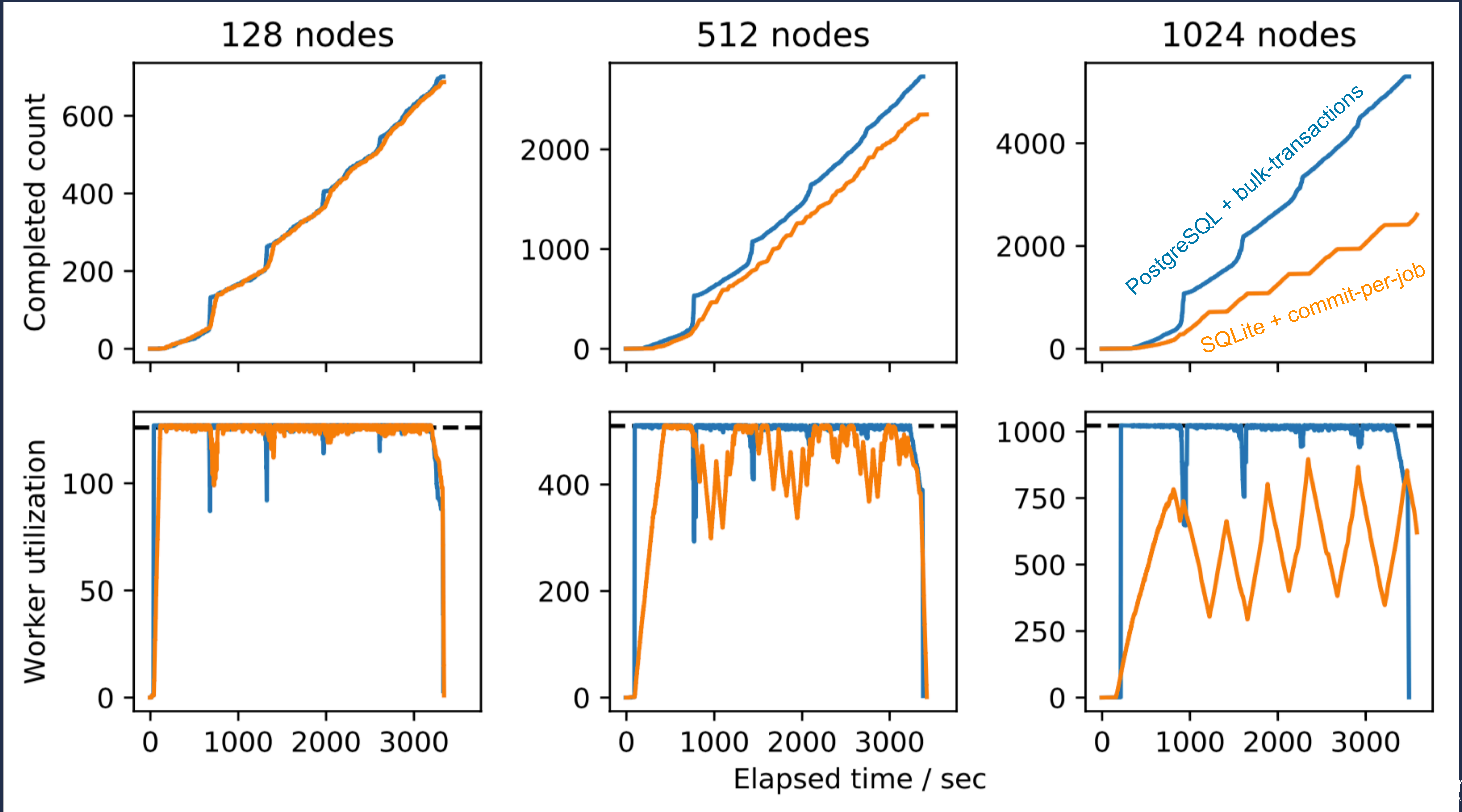
506 Theta KNL nodes
42 agents * 11 workers/agent



1022 Theta KNL nodes
85 agents * 11 workers/agent

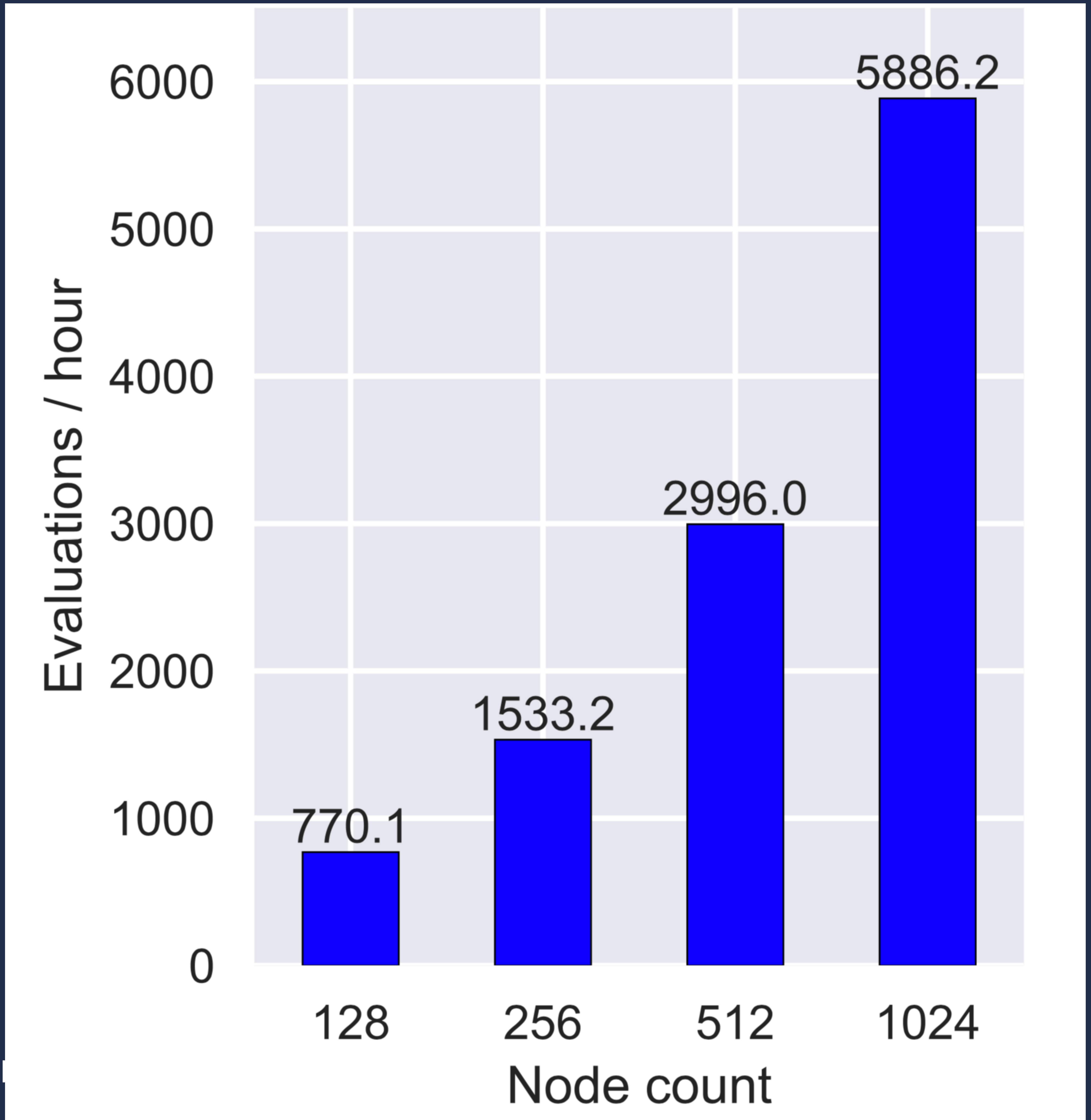
DeepHyper random search with Balsam serial job mode

Optimized database access patterns, task prefetch



DeepHyper random search with Balsam serial job mode

96% weak scaling efficiency from 128 to 1024 KNL nodes

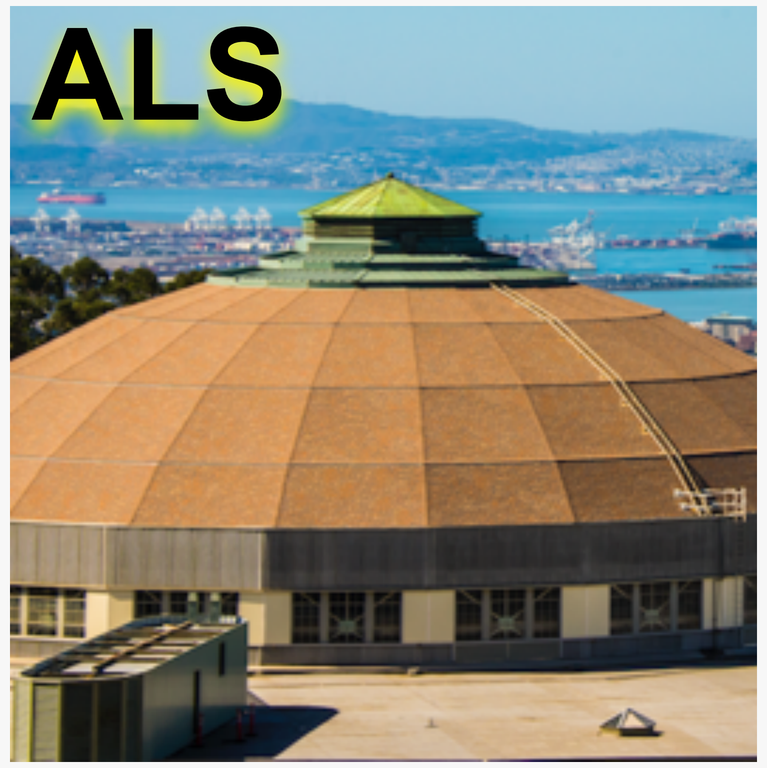


Toward production real-time analysis workloads

- Worked with national light sources to simulate real-time XPCS analysis scenario on Theta

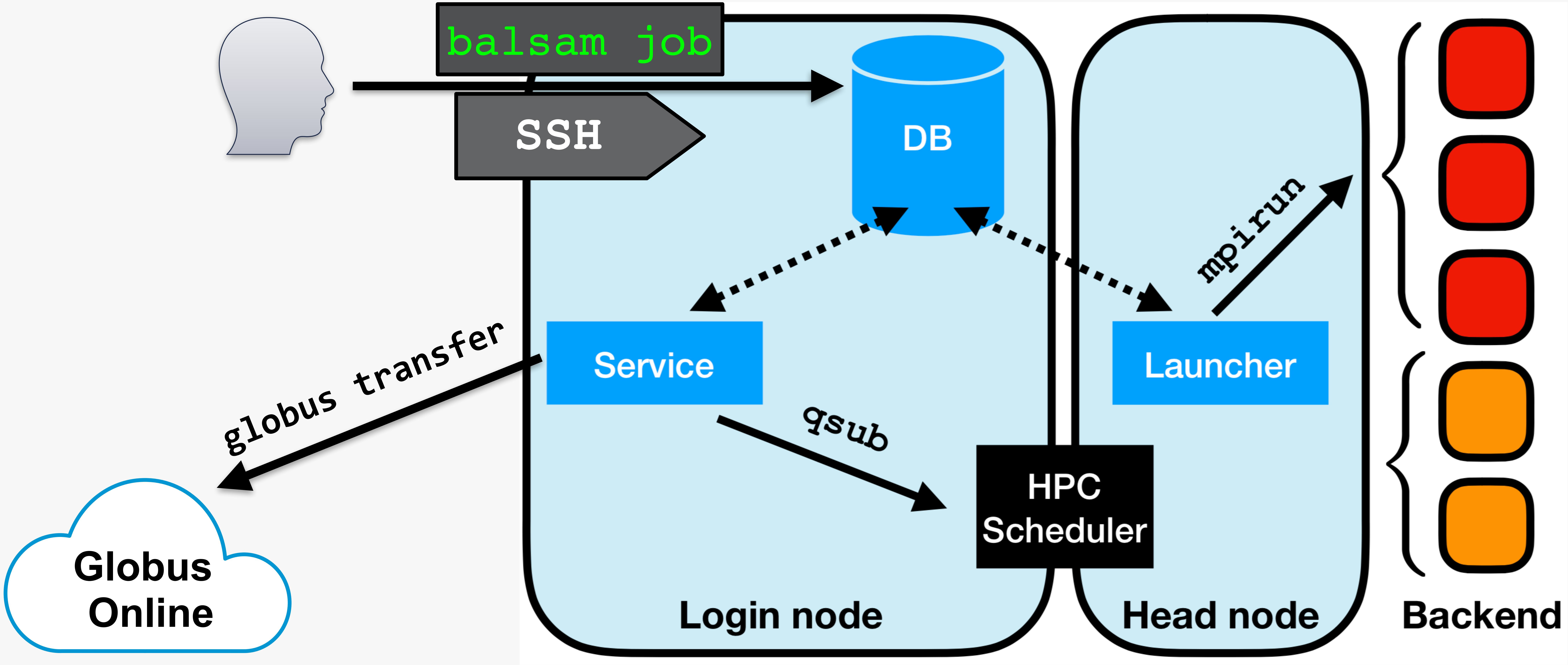
- ALCF piloting special **backfill queues** that uphold large-job mandate while allowing smaller jobs to "fill gaps" between production runs

- Balsam service elastically scales job submissions to task backlog



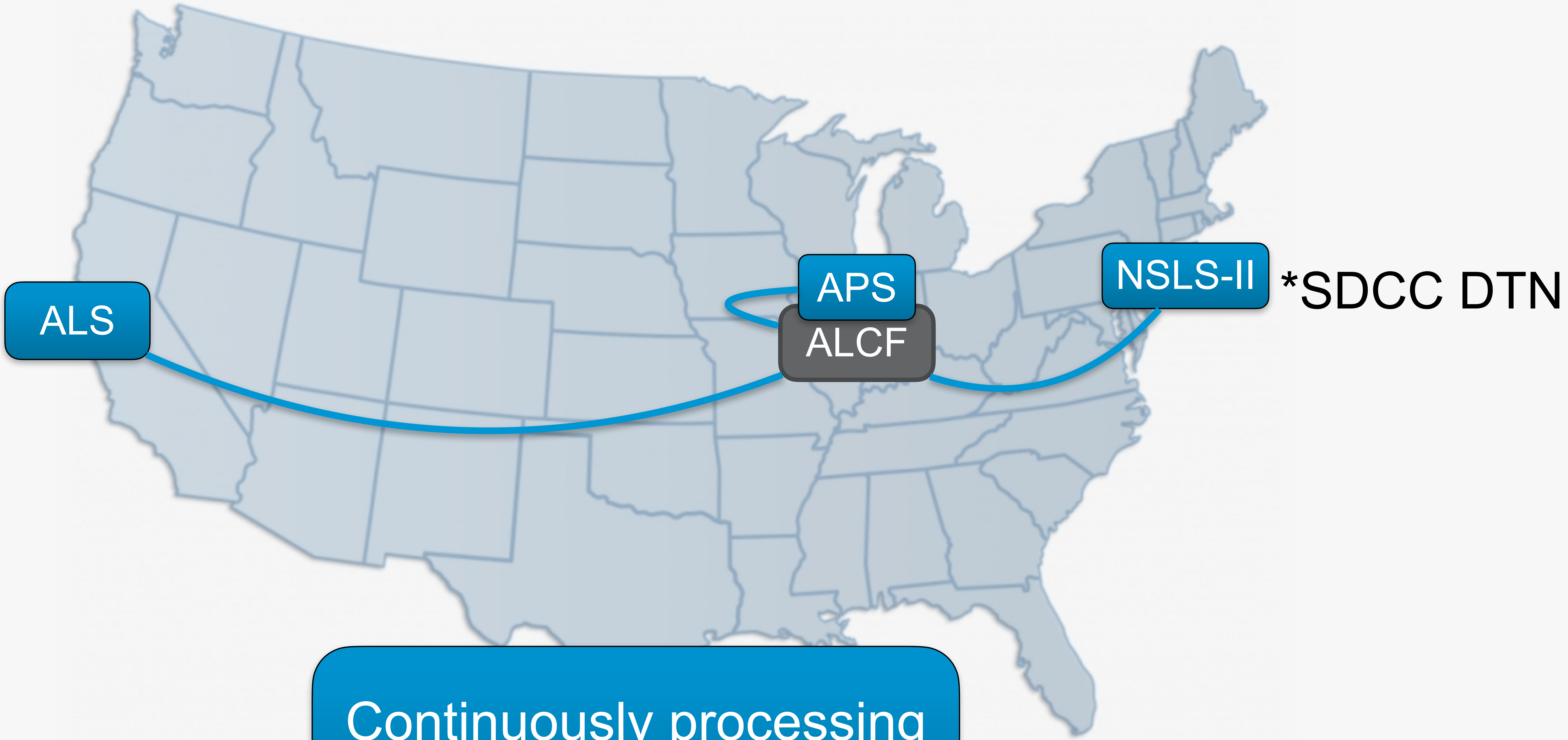
Remote Job Submission

New runs submitted from external client



Multi-source data analysis

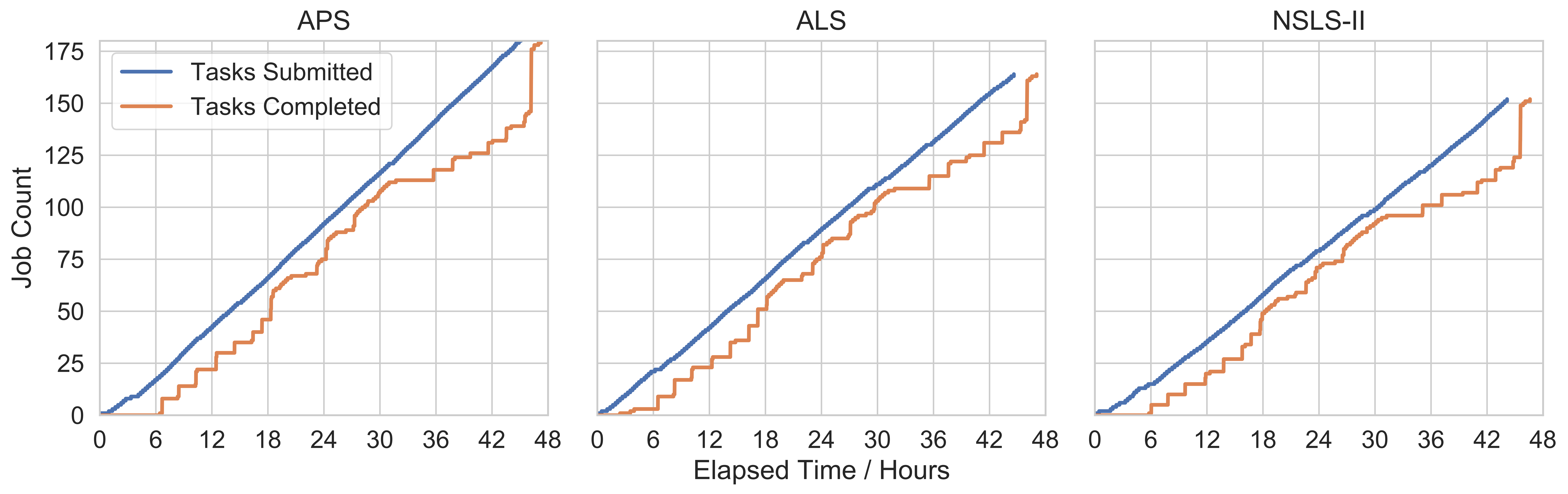
48 hours continuous XPCS data transfer & analysis between Theta and 3 science facilities



Continuously processing the same 40 GB dataset

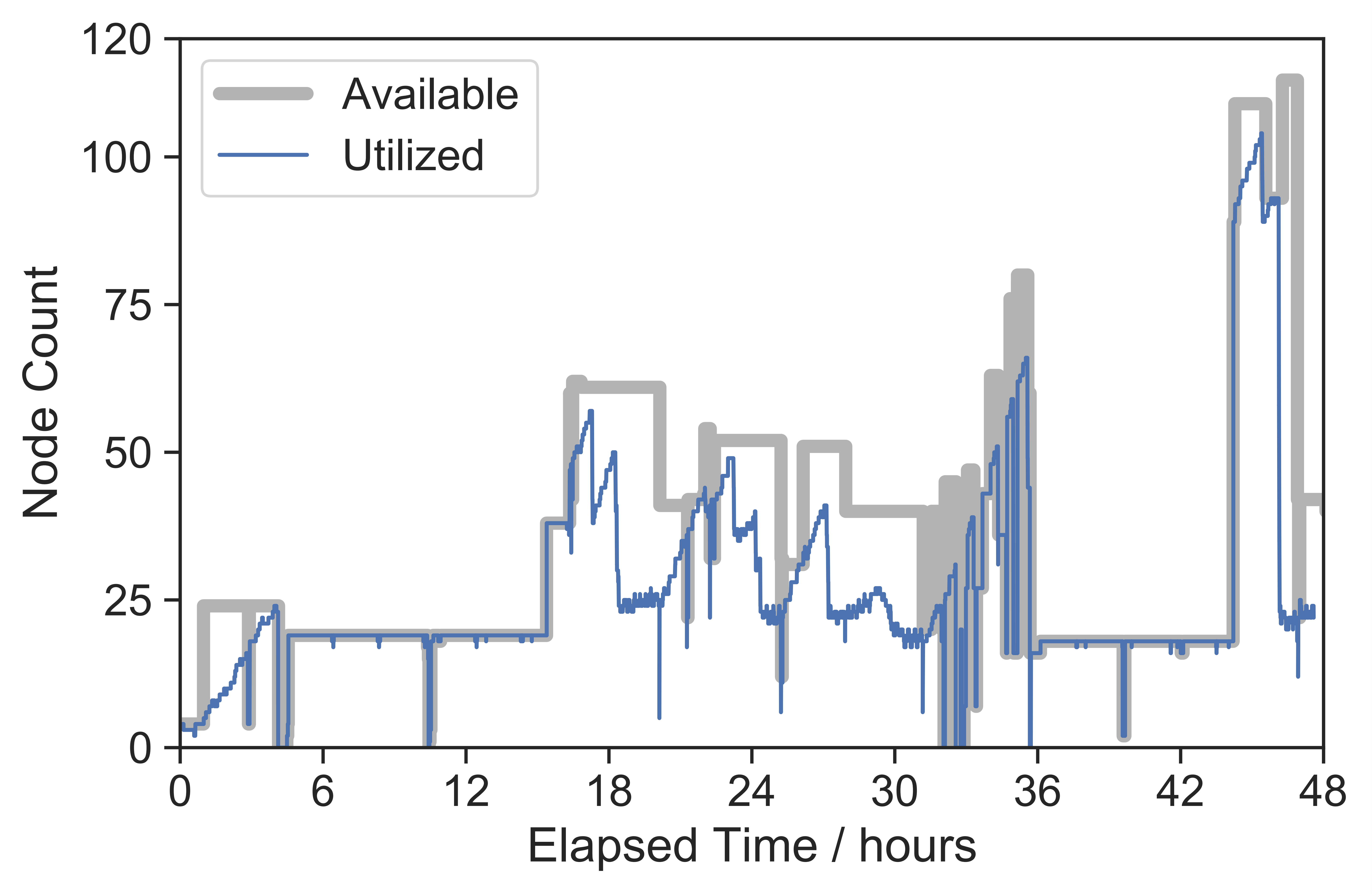
Multi-source analysis throughput

Averaged 3.5 tasks per-source, per-hour. All tasks executing concurrently through Balsam

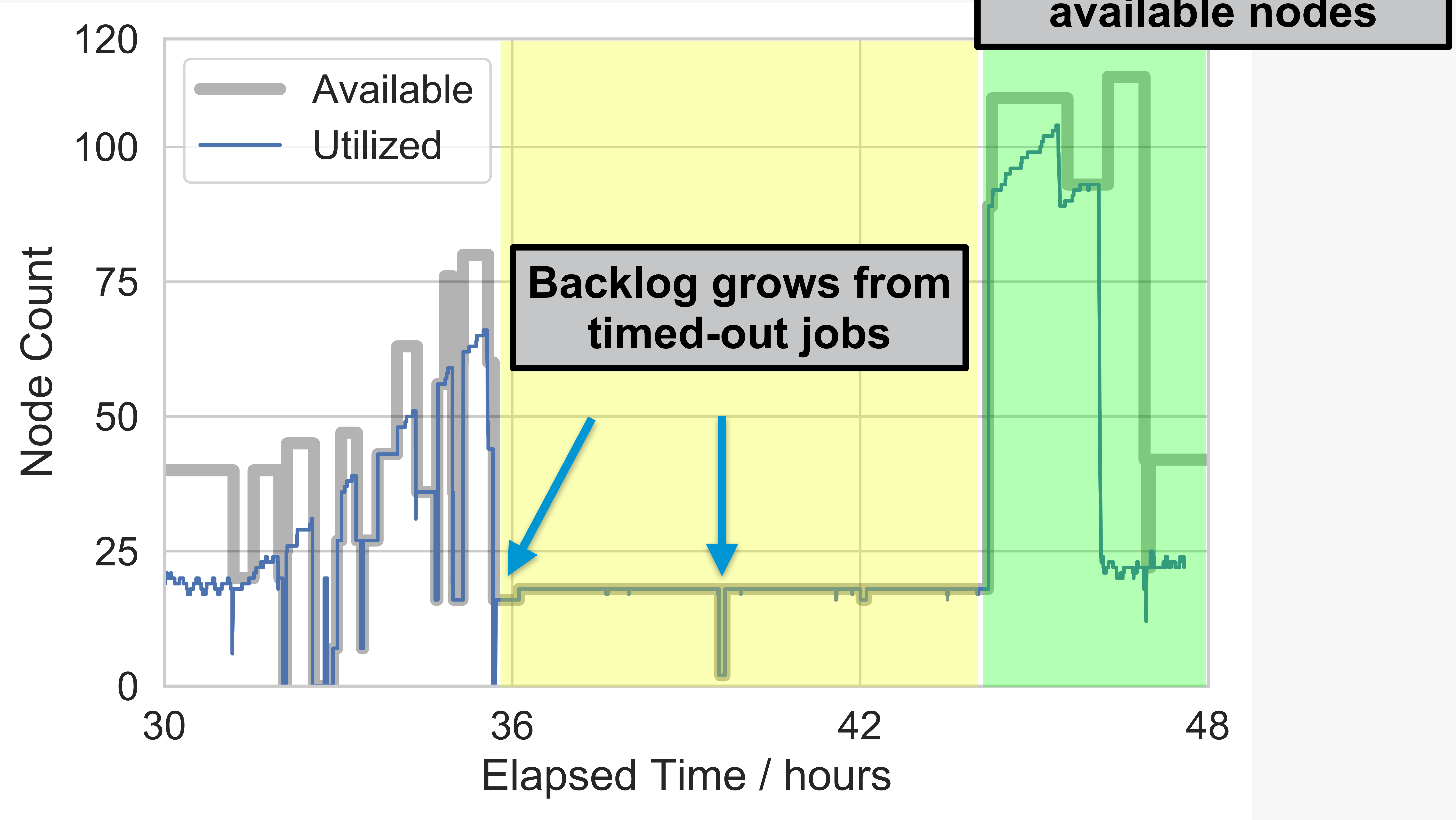


Auto-scaling by demand & availability

Execution spanned 46 jobs (resource requests) via backfill queue targeting idle nodes



Auto-scaling by demand & availability



Backlog surges onto available nodes

Backlog grows from timed-out jobs

Questions?