Balsam Workflows

balsam.readthedocs.io

Misha Salim
ALCF Datascience team
msalim@anl.gov

Data-centric computing at the ALCF

*"ALCF's mission is now to officially support not just traditional simulation-based HPC, but to support the three pillars of Simulation, **Data**, and **Learning**"*

https://www.alcf.anl.gov/early-science-program

Theta Ensemble Jobs

MOM (Broadwell) node

```
#!/bin/bash

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

Compute (KNL) Nodes

- nid00001
- nid00002
- nid00003
- nid00004
- nid00005

Theta Ensemble Jobs

MOM (Broadwell) node

```
#!/bin/bash

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

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

aprun

Compute (KNL) Nodes

- nid00001
- nid00002
- nid00003
- nid00004
- nid00005

Theta Ensemble Jobs

MOM (Broadwell) node

```
#!/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

aprun

Compute (KNL) Nodes

nid00001

nid00002

nid00003

nid00004

nid00005

Theta Ensemble Jobs

MOM (Broadwell) node

```
#!/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
```

aprun

aprun

aprun

Compute (KNL) Nodes

nid00001

nid00002

nid00003

nid00004

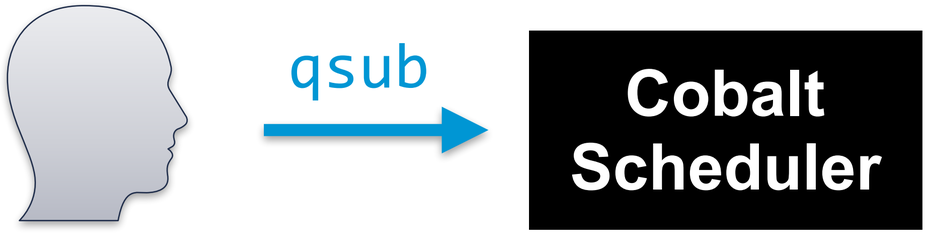
nid00005

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`



- 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
```

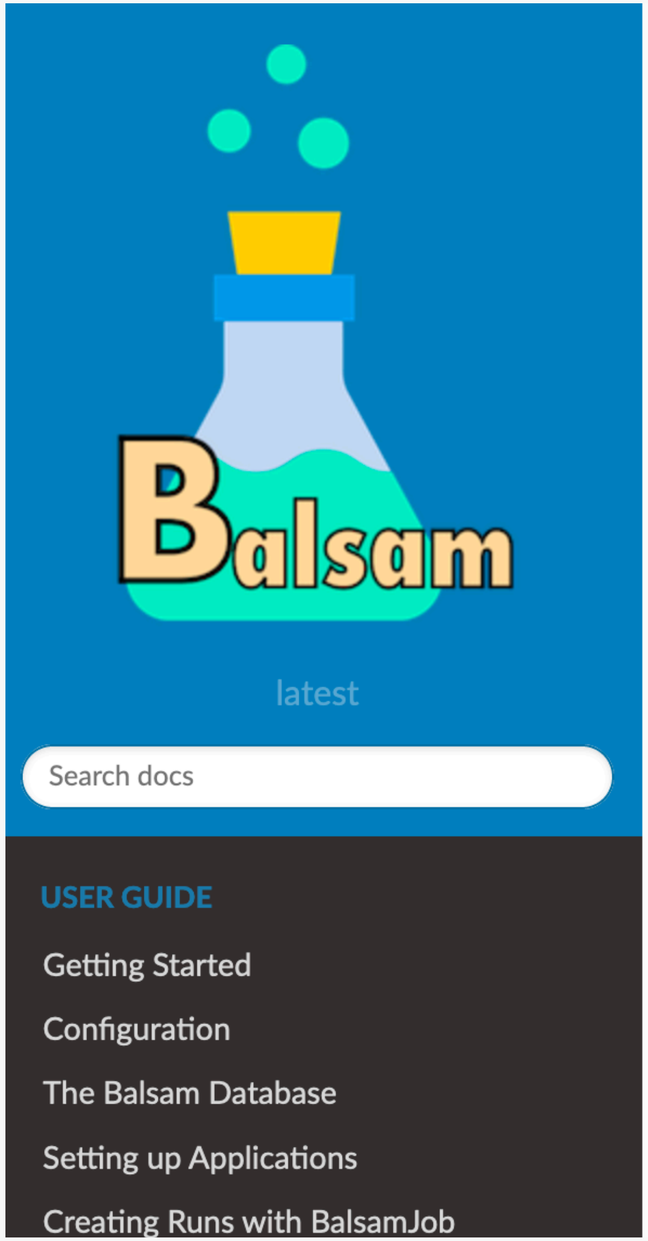
You either build workflow tools or adopt existing ones

Balsam

Automated 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 jobs
  - Runs **unmodified** user applications or Singularity containers
- Workflow status and project statistics available at-a-glance

In production @ ALCF



Docs » Balsam: HPC Workflow

```
module load balsam
```

Balsam: HPC Workflows & Edge Service

Balsam allows you to manage a database of workflows with simple command-line interfaces and a Python API. As you populate the database, the **Balsam service** can automatically reserve bundles of tasks and batch-schedule them for parallel execution. Inside each batch job, the **Balsam launcher** monitors available resources, launches applications, and sends status updates back to the database.

Balsam is designed to minimize user “buy-in” and cognitive overhead. You don’t have to learn an API or write any glue code to acheive throughput with existing applications. In fact, it’s arguably faster and easier to run a simple app several times using Balsam than by writing an ensemble job script:

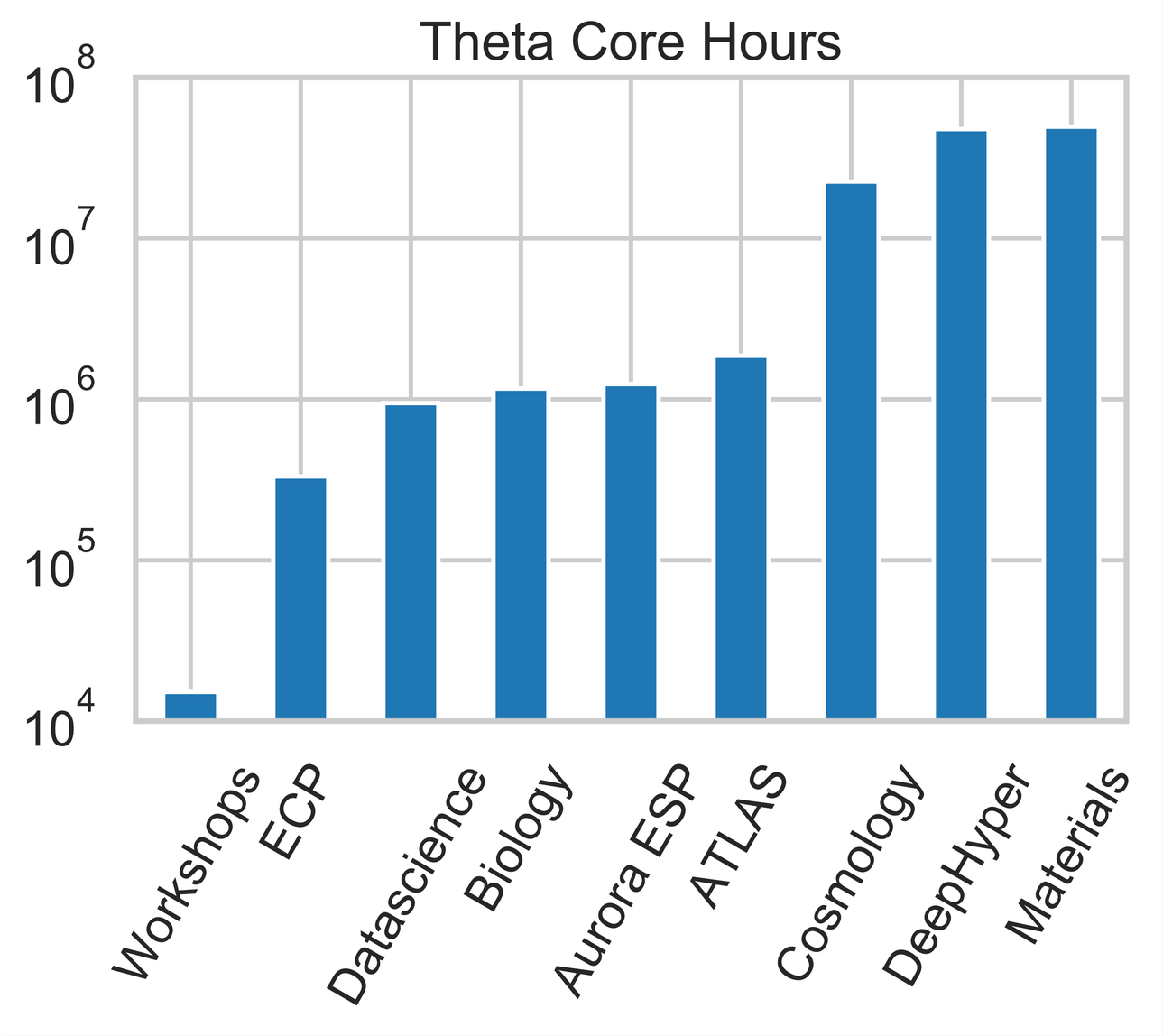
```
$ balsam app --name SayHello --executable "echo hello,"
$ for i in {1..10}
> do
>  balsam job --name hi$i --workflow test --application SayHello --args '
```

Argonne NATIONAL LABORATORY

Tracking Balsam Usage

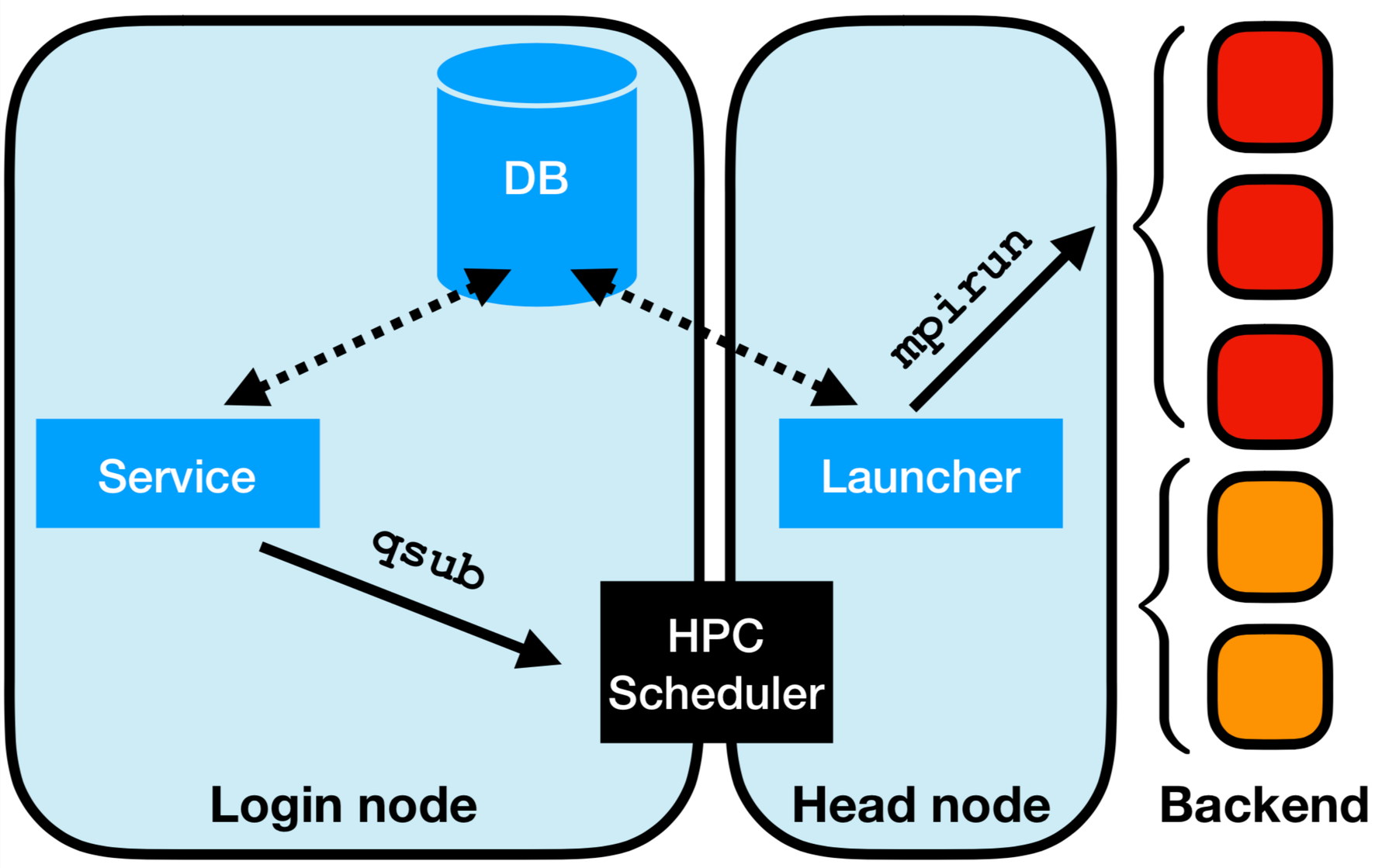
125M Theta core-hours since September 2018

- 48 users
- 28 projects
- Top usage categories:
  - Materials Science (39%)
  - DeepHyper (38%)
  - Cosmology (18%)



Data mined by Taylor Childers

A quick look at Balsam components



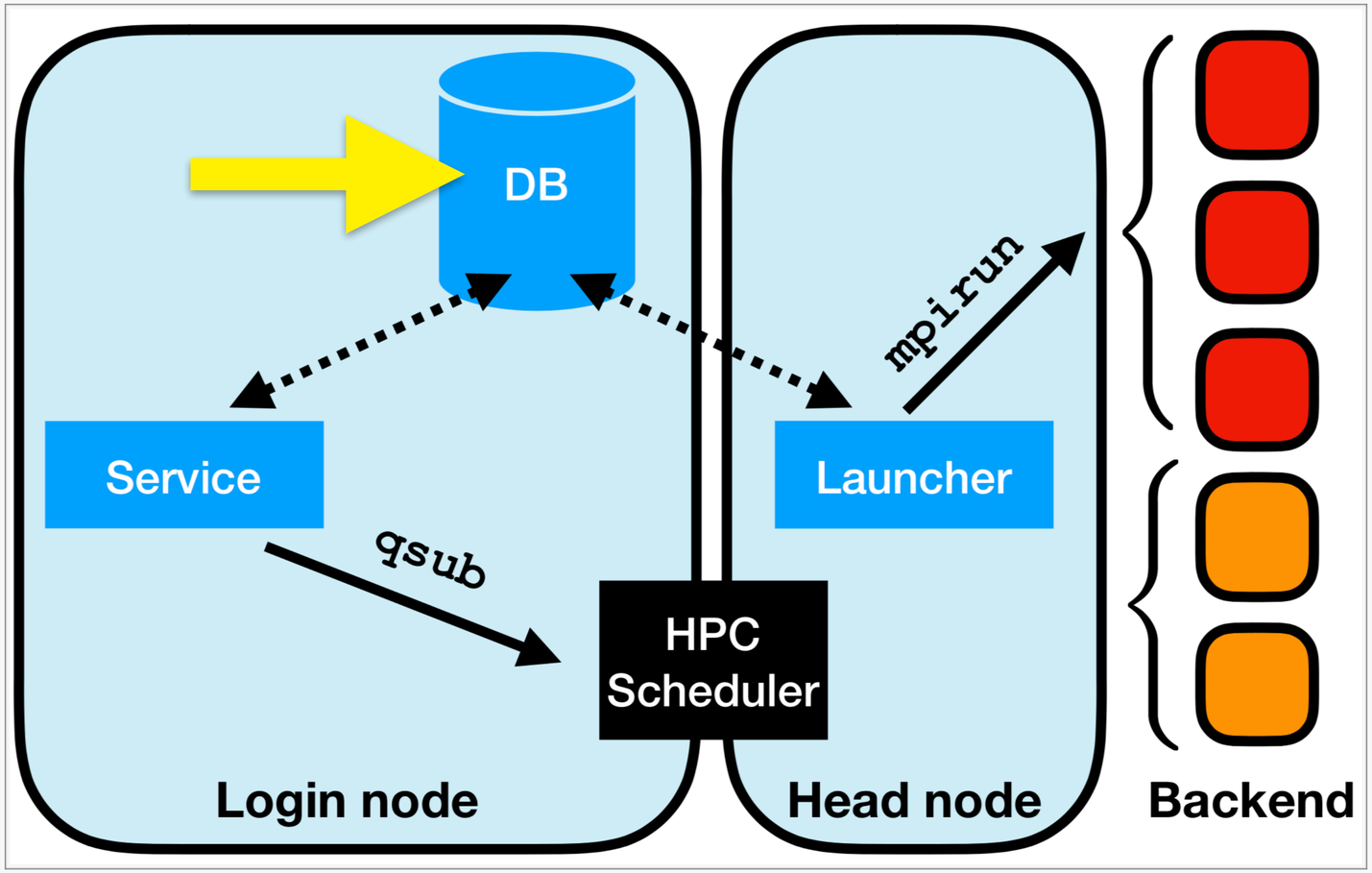
Database

BalsamJob table:

<u>one row per task</u>

One line setup:

```
balsam init ~/myproject
```

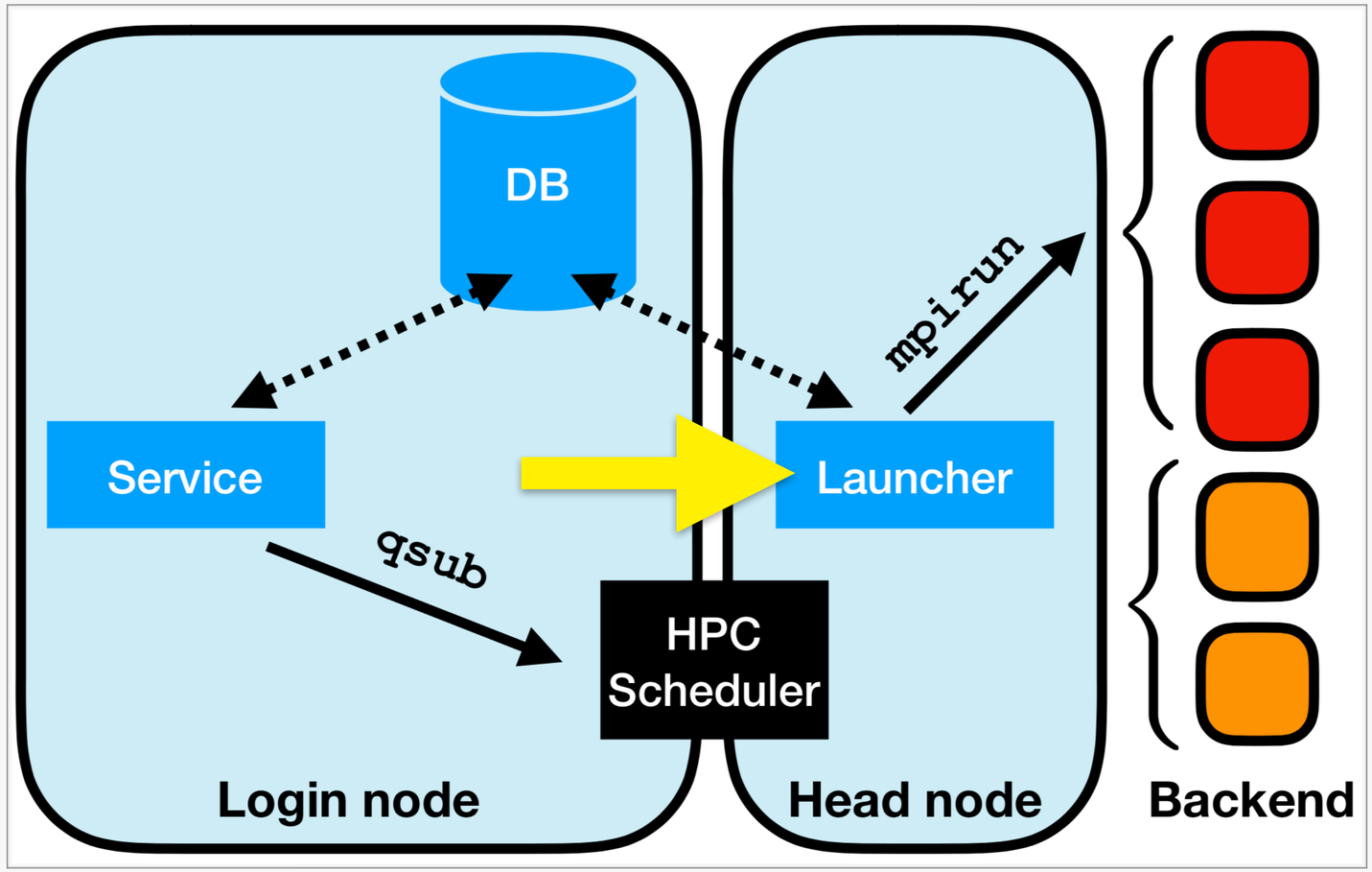


Launcher

Dynamic task pull and execution

MPI job mode for conventional app launch

serial job mode to pack many tasks under one mpirun



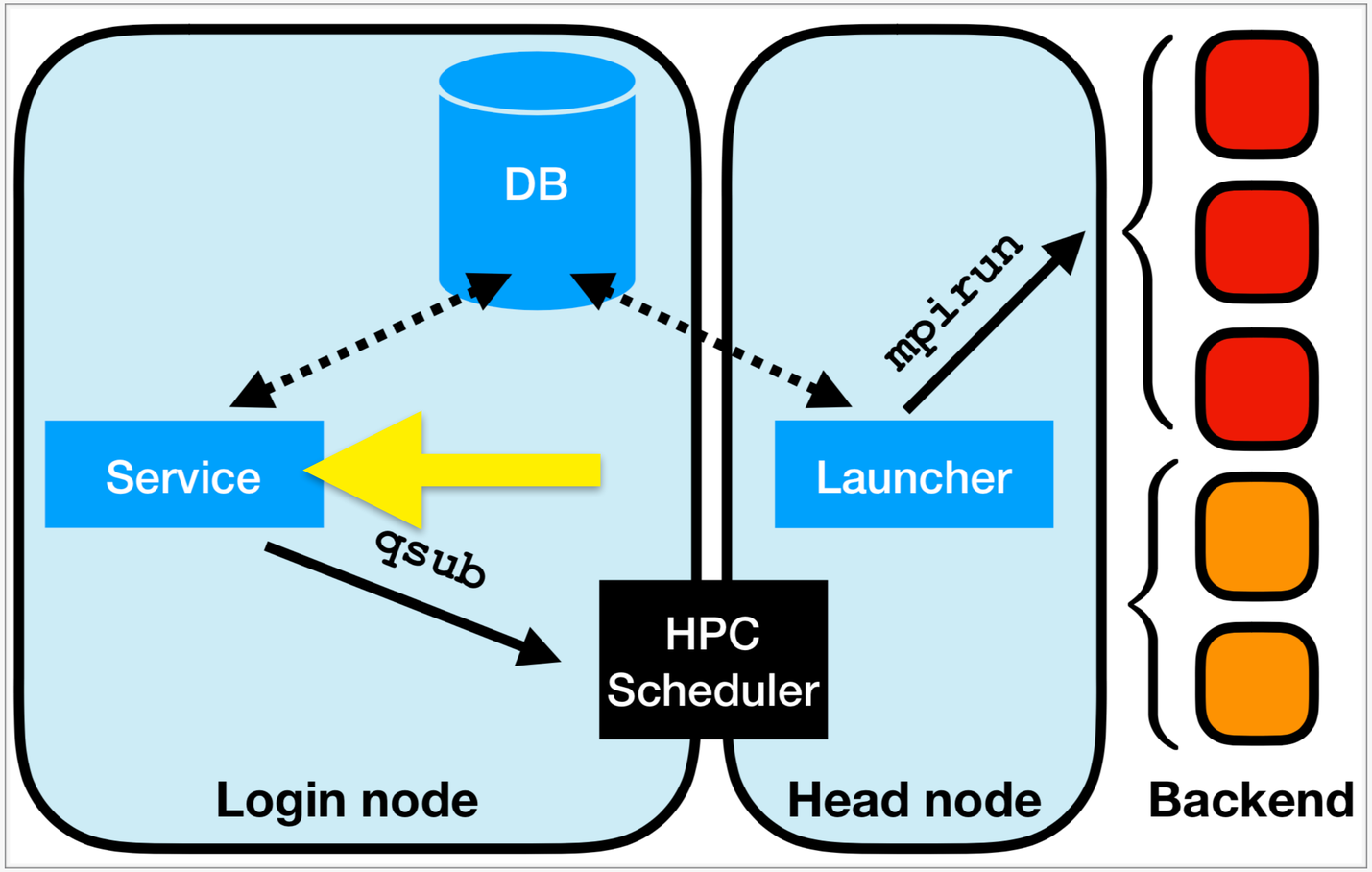
Service

Submission interface

```
balsam submit-launch
```

Auto queue submission

```
balsam service
```

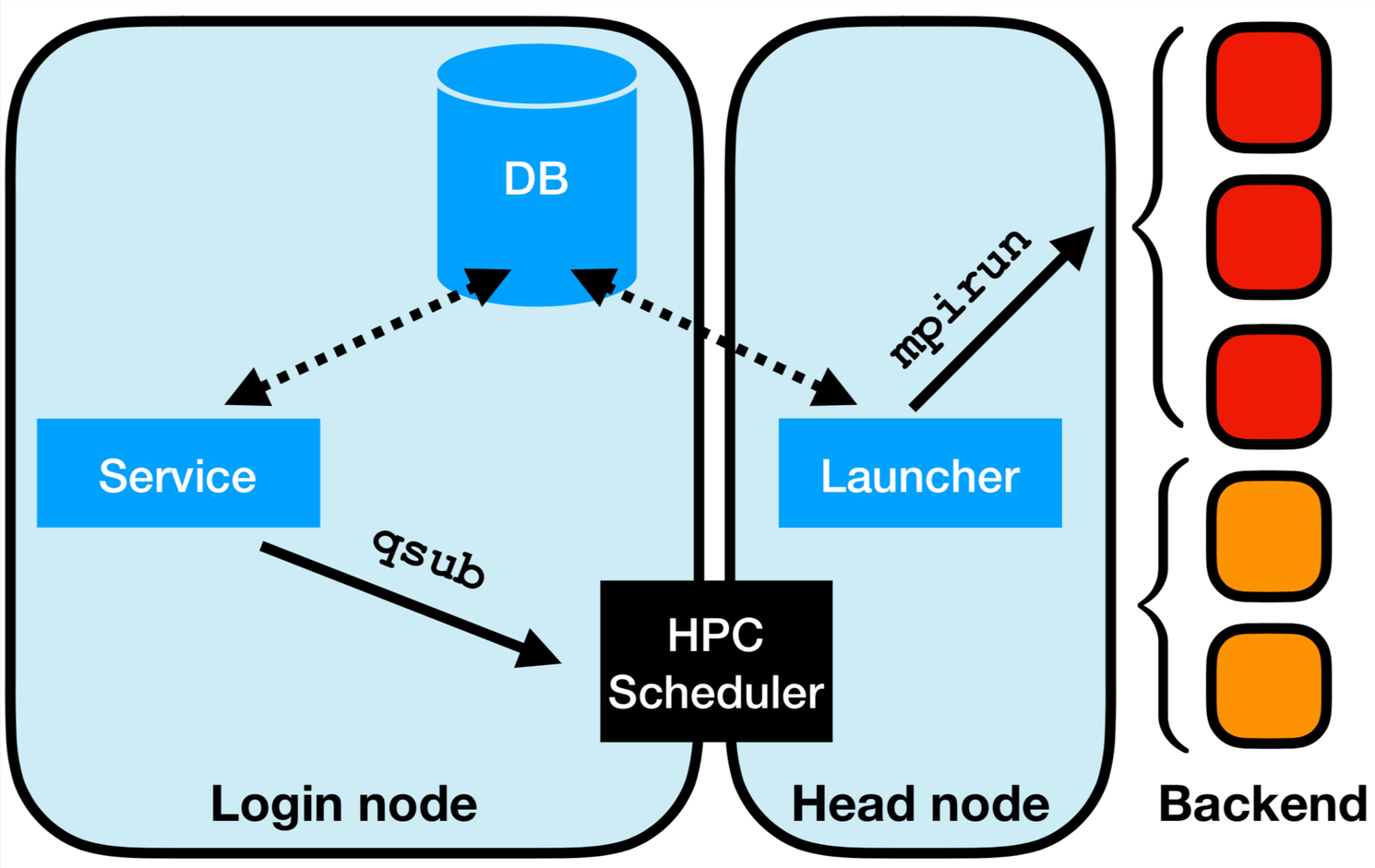


Balsam Usage Themes

- Balsam usage so far can be classified under one of 3 themes:
    1. A user populates their database with a campaign of tasks
    2. A "manager" process dynamically adds tasks during a running job
    3. Tasks are added across the network; Balsam triggers data stage-in
- The same Balsam components are used in every case
- The themes differ in *where* the **BalsamJob** object is created

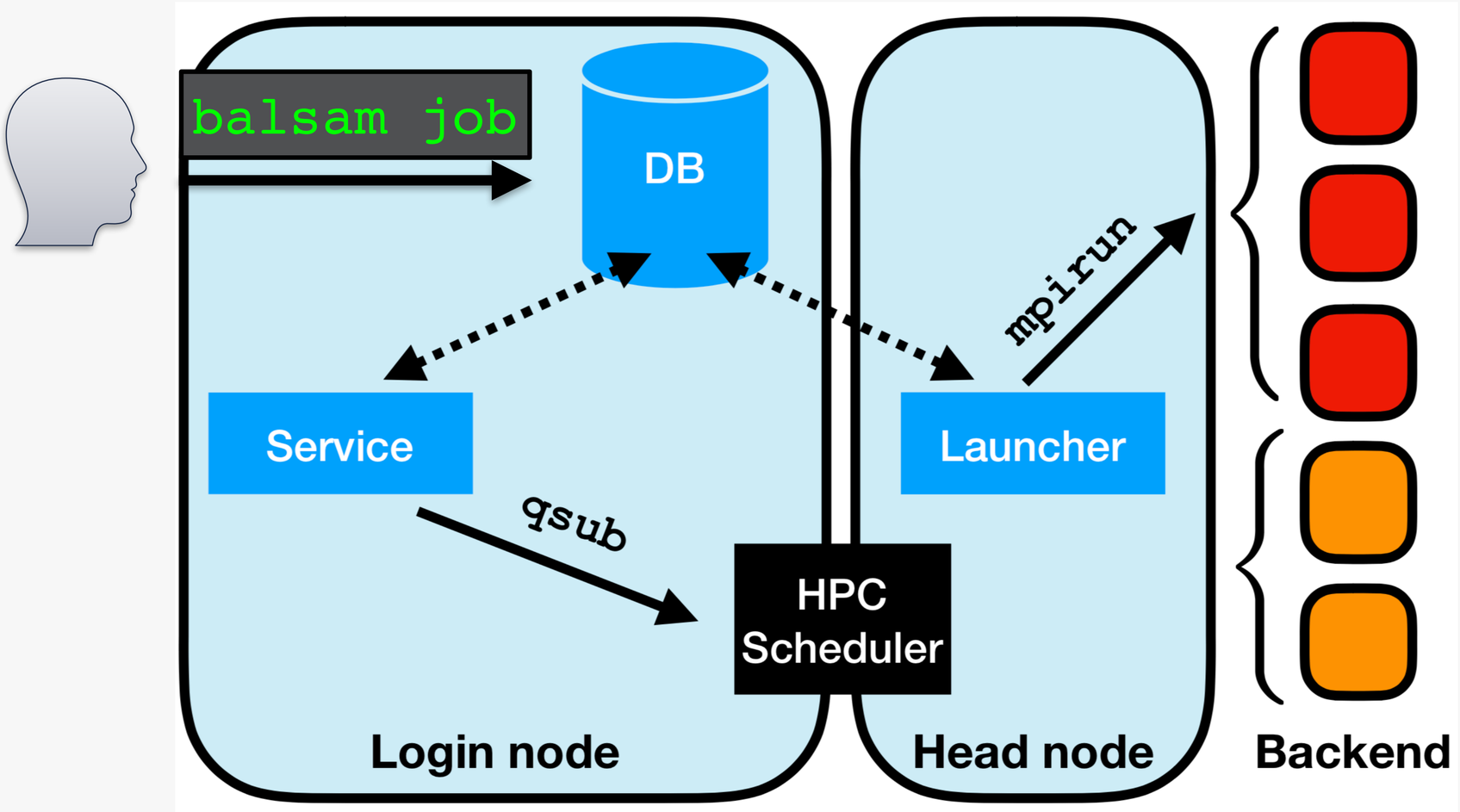
Theme 1: managing a job campaign locally

Submit application runs from login node



Theme 1: managing a job campaign locally

Submit application runs from login node



Chemistry & Materials Science Applications

NWCHEM
HIGH-PERFORMANCE COMPUTATIONAL CHEMISTRY SOFTWARE

Typical Workflow

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',
    )
```

Typical Workflow

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()
```

Typical Workflow

Check stderr of failed jobs at-a-glance

```
[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")
```

Typical Workflow

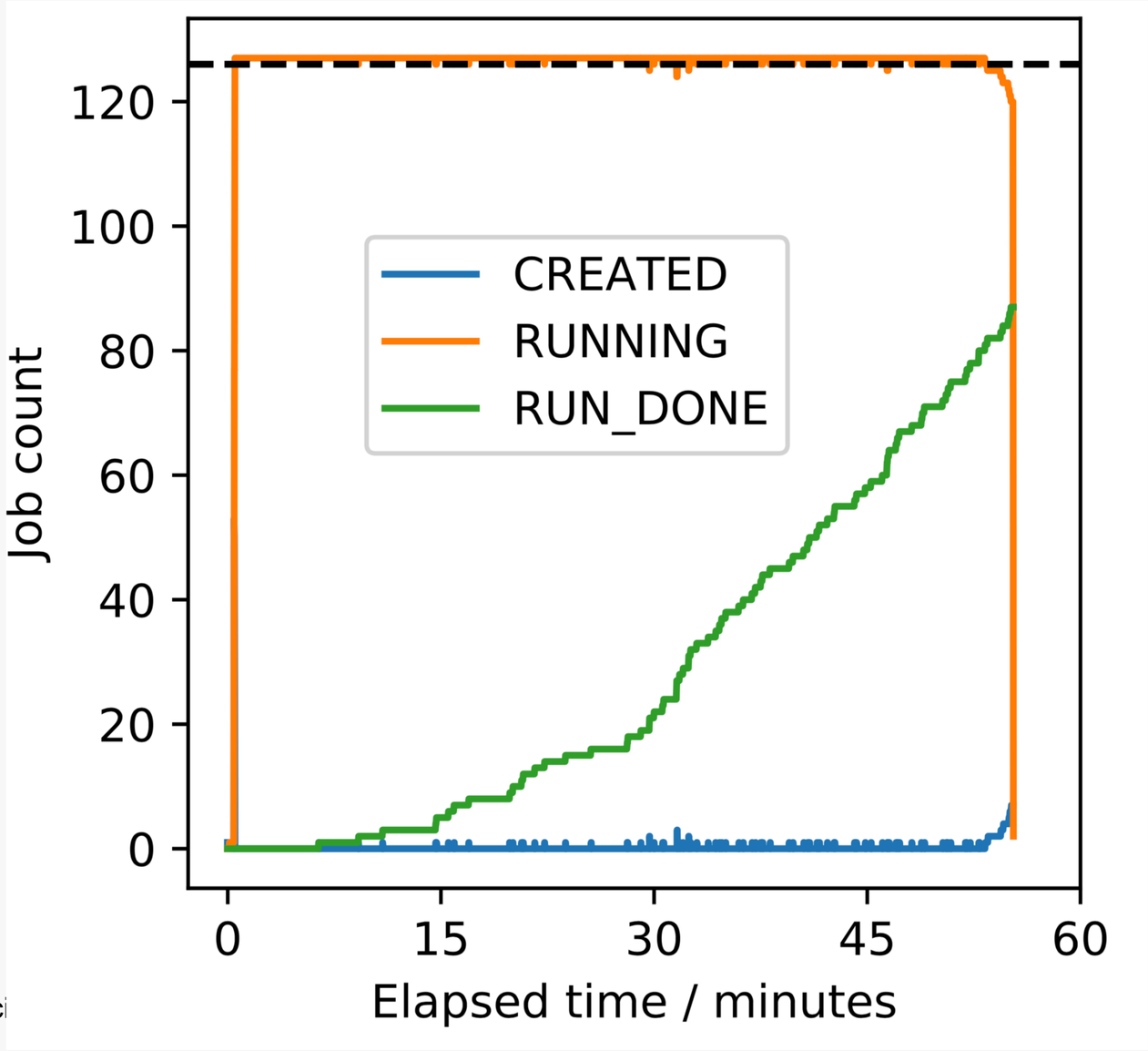
Gather working directories of timed-out jobs

```
from balsam.launcher.dag import BalsamJob

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

Typical Workflow

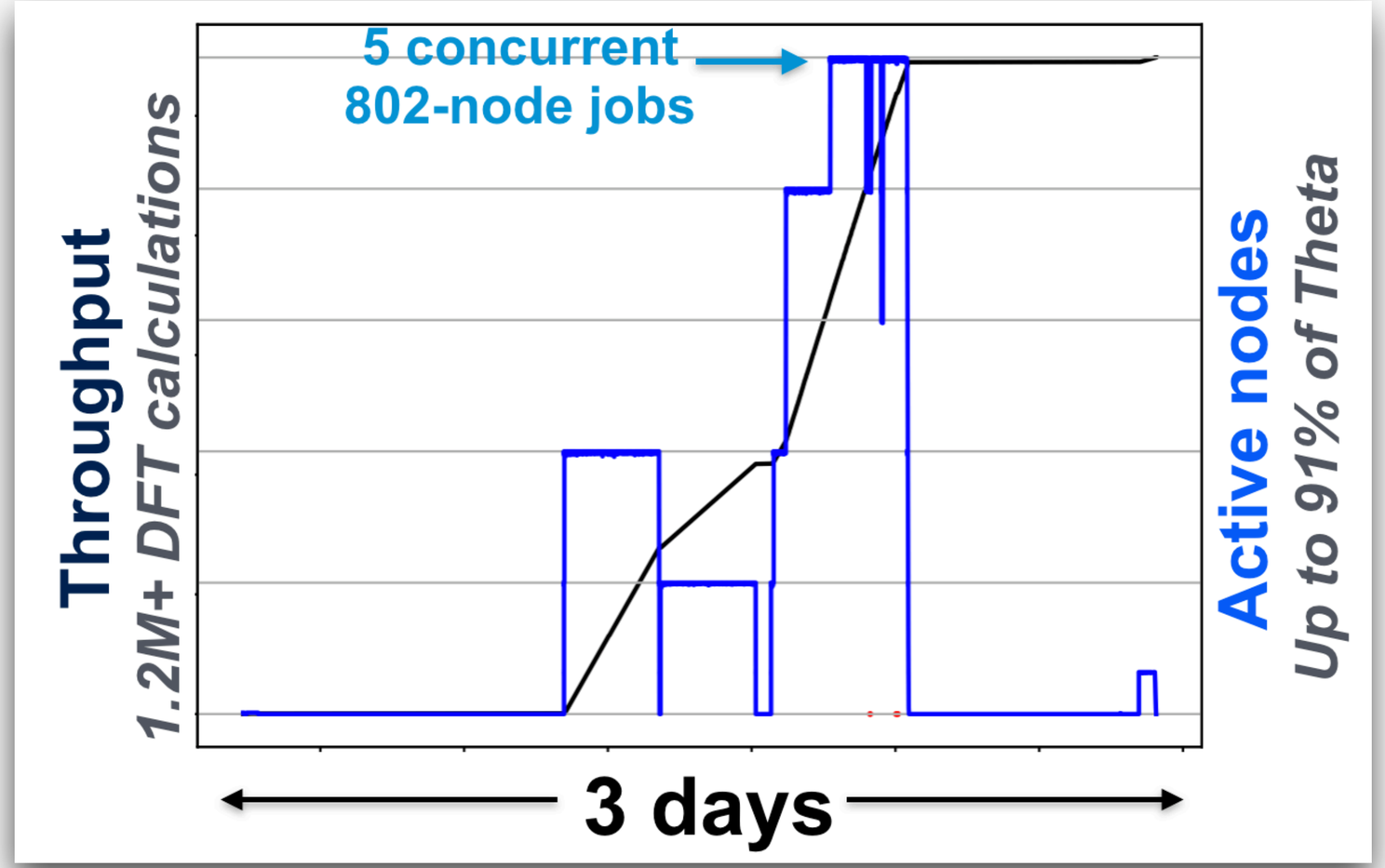
Convenience functions for visualizing throughput & utilization



Molecular Crystals ADSP

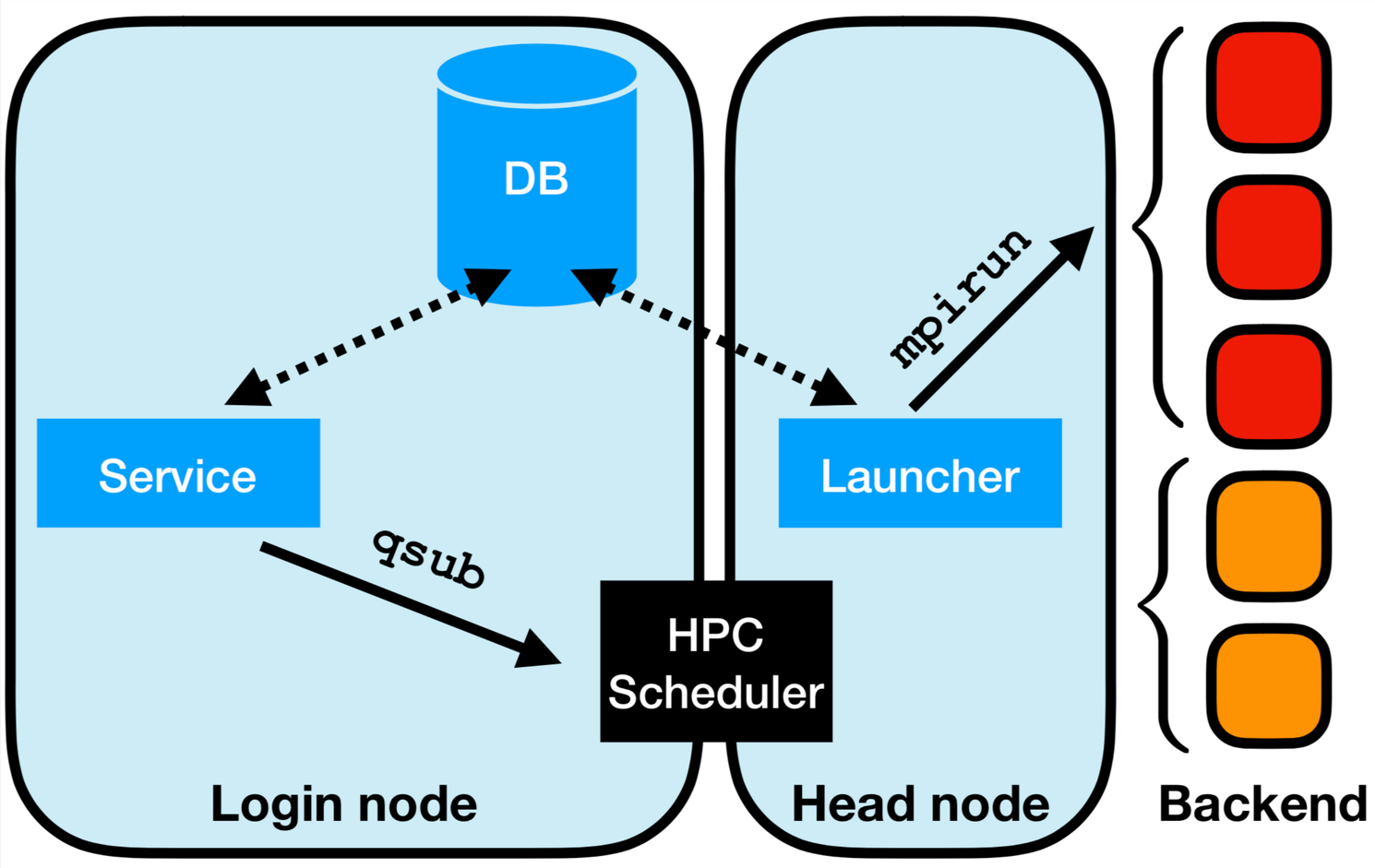
Cataloging free-energies of crystalline polymorphs (PI: Alexandre Tkatchenko)

- 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



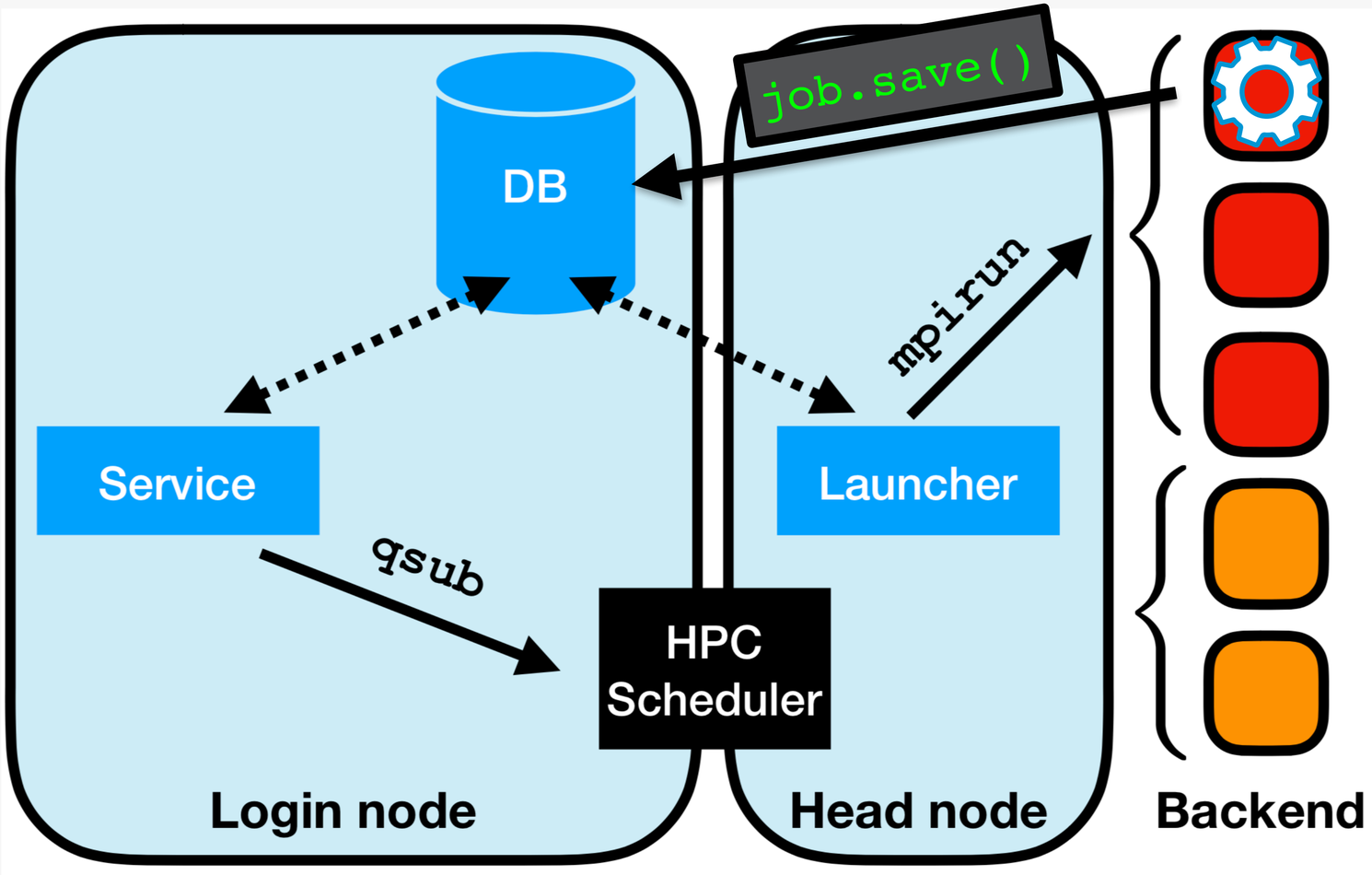
Theme 2: Dynamic Job Launch

Manager generates new runs from compute node



Theme 2: Dynamic Job Launch

Manager generates new runs from compute node



Theme 2: Dynamic Job Launch

Manager generates new runs from compute node

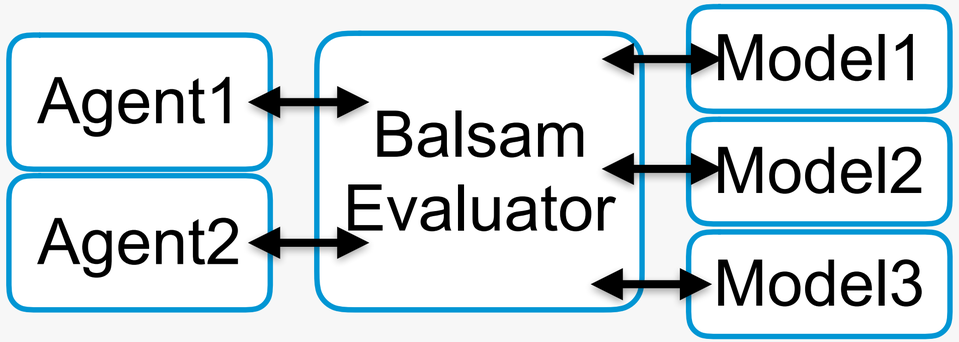
Hyperparameter Optimization and Neural Architecture Search

Framework for Generator/simulator-type ensemble jobs

DeepHyper

Scaling Hyperparameter Optimization and Neural Architecture Search

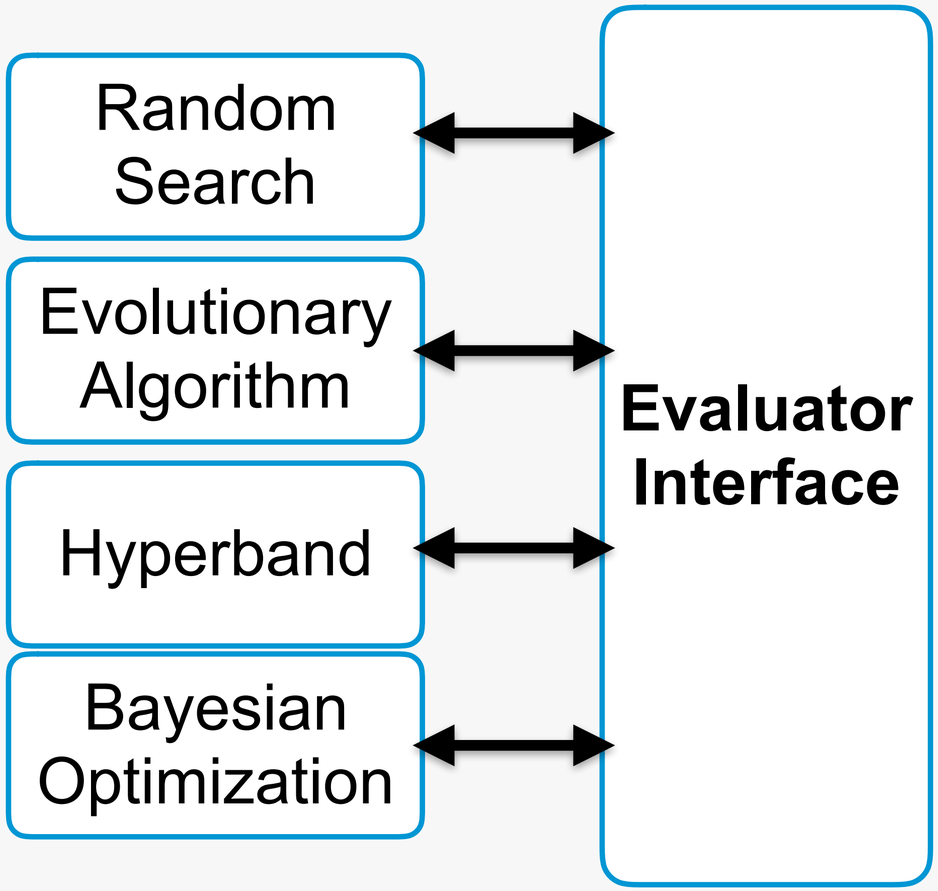
- A framework for coupling black-box optimizers to a controller managing parallel evaluations of an expensive function



https://deephyper.readthedocs.io/en/latest/

DeepHyper

Model evaluation API decouples search from model execution



3-function interface for async model evaluation

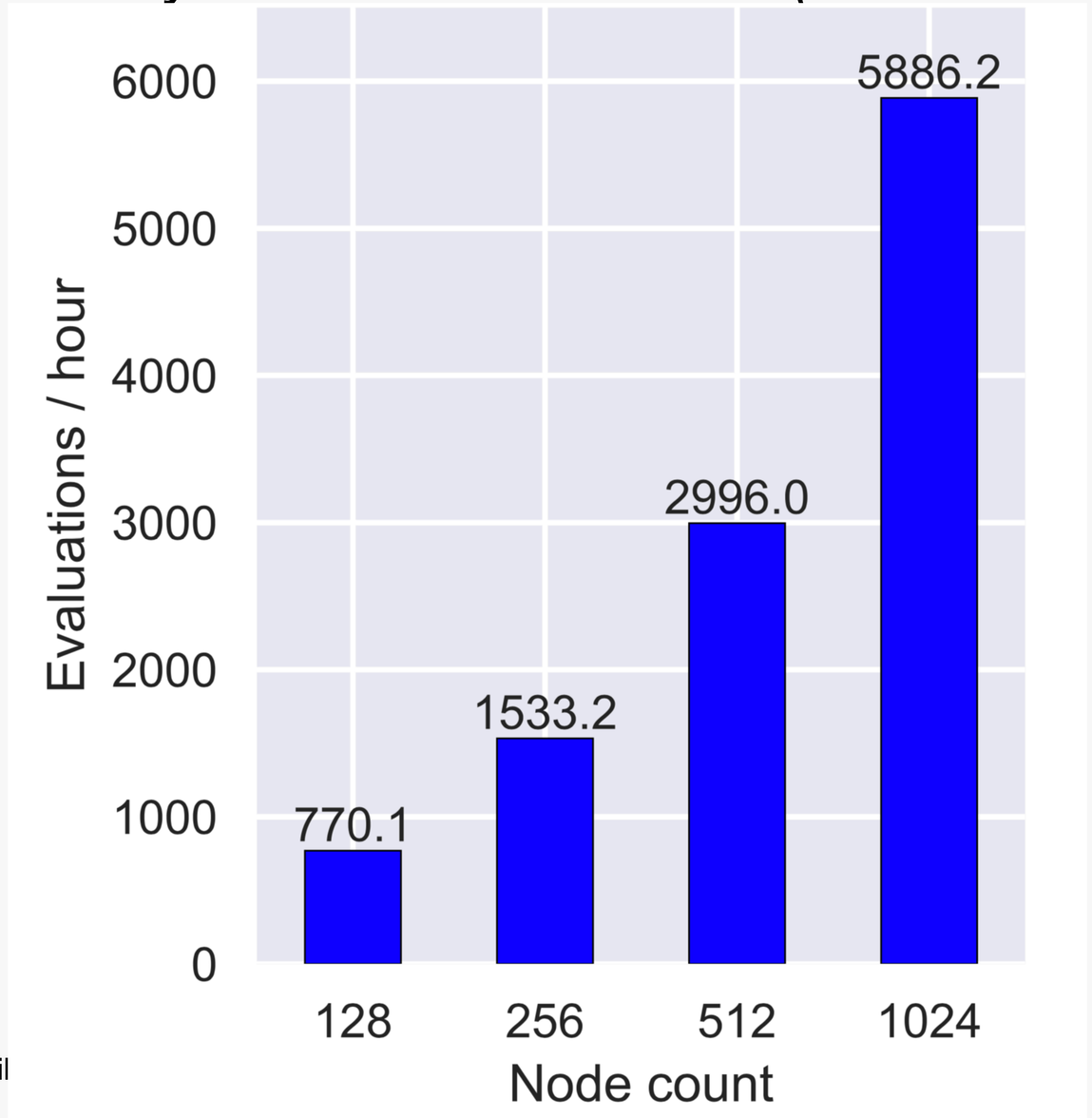
```
add_eval_batch(XX)
```

```
get_finished_evals()
```

```
await_evals(XX)
```

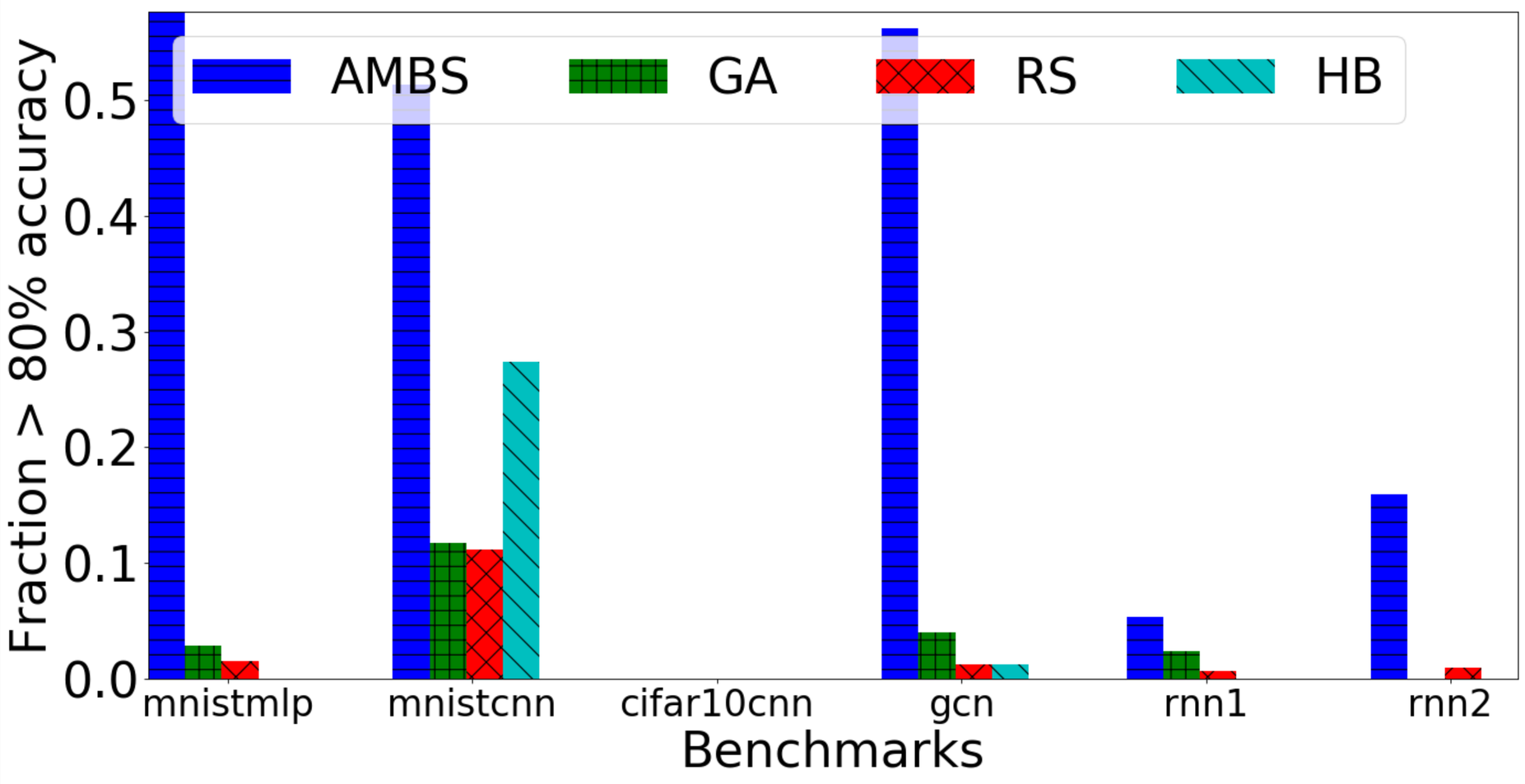
Scaling DeepHyper with Balsam

96% weak scaling efficiency to 1024 Theta nodes (1 model/KNL)



Hyperparameter optimization studies

At 128 nodes, Bayesian search (AMBS) outperforms genetic algorithm, random search, hyperband on selected benchmarks



Theta/Cooley Walkthrough

/projects/SDL_Workshop/training/Balsam

On Theta:

```
$ module load balsam
```

On Cooley:

```
$ source /soft/datascience/balsam/setup.sh
```

Sets PATH to include:
PostgreSQL binaries
Python 3.6 virtualenv's bin/ (with Balsam)

Command line interface

```
$   balsam

usage: balsam [-h]

{app,job,dep,ls,modify,rm,killjob,mkchild,launcher,submit-
launch,init,service,make_dummies,which,log,server}

Balsam 0.3.5

Command line interface:

    app                     add a new application definition
    job                     add a new Balsam job
    dep                     add a dependency between two existing jobs
    ls                      list jobs, applications, or jobs-by-workflow
```

Start a new Balsam DB

Use `balsam init` to create a new database directory

```
balsam init ~/myproject
```

```
**********************************************************************
  Successfully created Balsam DB at: /home/user/myProject
  Use `source balsamactivate myProject` to begin working.
**********************************************************************
```

Start a new Balsam DB

`source balsamactivate <db-name>`

starts server (if not running), sets environment

```
$  . balsamactivate myProject
Launching Balsam DB server
waiting for server to start.... done
server started
[BalsamDB: myProject] $
```

Hello World

`balsam app :`

Register new applications with Balsam

```
[BalsamDB: myProject] $ balsam app --name say-hello \
--executable "echo Hello, "

Application 1:
--------------
name:                                   say-hello
description:
executable:                             echo Hello,

Added app to database
```

Hello World

`balsam job` :

Add a new task

```
[BalsamDB: myProject] $  balsam job --name test1 --workflow test \
   --app say-hello --args "World 1!"
```

```
[BalsamDB: myProject] $ for i in {2..10}
> do
> balsam job --name test$i --workflow test \
  --app say-hello --args "World $i!" --yes
> done
```

```
BalsamJob 0796bd50-adbc-424b-bd26-476f2c00275b
-----------------------------------------------
workflow:                       test
name:                           test1
description:
parents:                        []
input_files:                    *
num_nodes:                      1
ranks_per_node:                 1
environ_vars:
application:                    say-hello
args:                           World 1!
auto_timeout_retry:             True
  *** Executed command:         echo Hello, World 1!
  *** Working directory:        ~/myProject/data/test/test1_0796bd50

Confirm adding job to DB [y/n]: y
```

Confirmation shows task details and adjustable fields

BalsamJob fields

Name

Workflow Tag

Description

Working directory

JSON data

Working directory default:

`{workflow}/{jobname}_{jobid}`

Or, specify another workdir

Store & Query JSON data with Python API

BalsamJob fields

Parents

Stage-out URL

Stage-in URL

Input file patterns

Move data in & out

Make parent job outputs visible to children

BalsamJob fields

Executable

Environment variables

Command line arguments

Pre/post-process scripts

 # MPI ranks

ranks / node

Control how / on what resources application is launched

Run pre/post-processing stages for a particular app

Hello World

`balsam ls` :
View tasks in database

```
[BalsamDB: myProject] $  balsam ls

                                    job_id |    name | workflow | application |    state
-----------------------------------------------------------------------------------------
0796bd50-adbc-424b-bd26-476f2c00275b | test1   | test     | say-hello   | CREATED
421e6df8-4984-423f-b44f-c58c6e2e8307 | test2   | test     | say-hello   | CREATED
d62b194a-e20b-4111-a407-3669fb4c89e6 | test3   | test     | say-hello   | CREATED
e73325df-3104-4e7f-aa1e-9ba61840ecc6 | test4   | test     | say-hello   | CREATED
79becbd6-8ab9-4012-9828-6dc98157eb5c | test5   | test     | say-hello   | CREATED
c7ed41fd-6aa4-4a29-957e-bf91cfef3453 | test6   | test     | say-hello   | CREATED
dda7cdd3-0098-4fe7-9827-ca78a73d7be9 | test7   | test     | say-hello   | CREATED
2e6c4914-a1a7-4ea1-be5e-994fc6ca7830 | test8   | test     | say-hello   | CREATED
4a12cb47-cbe1-4f83-97d2-ee97eefc9343 | test9   | test     | say-hello   | CREATED
f4d77a25-1d48-4fc9-be2b-cc0e1af61473 | test10  | test     | say-hello   | CREATED
```

Hello World

`balsam submit-launch` :

Shortcut for Cobalt job submission (template in ~/.balsam)

```
[BalsamDB: myProject] $  balsam submit-launch -n 2 -t 5 \
   -q debug-cache-quad -A datascience --job-mode serial

Submit OK: Qlaunch {
    'command': '~/myProject/qsubmit/qlaunch1.sh',
    'id': 1,
    'job_mode': 'serial',
    'nodes': 2,
    'project': 'datascience',
    'queue': 'debug-cache-quad',
    'scheduler_id': 333718,
    'state': 'submitted',
    'wall_minutes': 5,
    'wf_filter': ''}
```

Customizable templated script

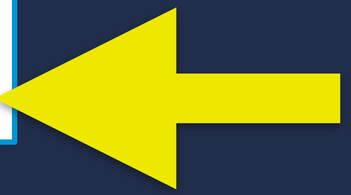
The Launcher job template

~/.balsam/settings.json

```
"NUM_TRANSITION_THREADS": 5,
"MAX_CONCURRENT_MPIRUNS": 1000,

"LOG_HANDLER_LEVEL": "INFO",
"LOG_BACKUP_COUNT": 5,
"LOG_FILE_SIZE_LIMIT": 104857600,

"QUEUE_POLICY": "theta_policy.ini",
"JOB_TEMPLATE": "job-templates/theta.cobaltscheduler.tmpl"
```



The Launcher job template

```
#!/bin/bash -x
#COBALT -A {{ project }}
#COBALT -n {{ nodes }}
#COBALT -q {{ queue }}
#COBALT -t {{ time_minutes }}
#COBALT --attrs ssds=required:ssd_size=128

export PATH={{ balsam_bin }}:{{ pg_bin }}:$PATH

        Insert pre-run commands here
      aprun -n $COBALT_JOBSIZE -N 1 cp ...

source balsamactivate {{ balsam_db_path }}

balsam launcher --{{ wf_filter }} --job-mode={{ job_mode }} \
    --time-limit-minutes={{ time_minutes }}
```

Hello World

If successful, jobs eventually marked JOB_FINISHED

```
[BalsamDB: myProject] $  balsam ls

                                 job_id |   name | workflow | application |        state
-------------------------------------------------------------------------------------------
dda7cdd3-0098-4fe7-9827-ca78a73d7be9 | test7  | test     | say-hello   | JOB_FINISHED
f4d77a25-1d48-4fc9-be2b-cc0e1af61473 | test10 | test     | say-hello   | JOB_FINISHED
79becbd6-8ab9-4012-9828-6dc98157eb5c | test5  | test     | say-hello   | JOB_FINISHED
c7ed41fd-6aa4-4a29-957e-bf91cfef3453 | test6  | test     | say-hello   | JOB_FINISHED
e73325df-3104-4e7f-aa1e-9ba61840ecc6 | test4  | test     | say-hello   | JOB_FINISHED
4a12cb47-cbe1-4f83-97d2-ee97eefc9343 | test9  | test     | say-hello   | JOB_FINISHED
421e6df8-4984-423f-b44f-c58c6e2e8307 | test2  | test     | say-hello   | JOB_FINISHED
2e6c4914-a1a7-4ea1-be5e-994fc6ca7830 | test8  | test     | say-hello   | JOB_FINISHED
0796bd50-adbc-424b-bd26-476f2c00275b | test1  | test     | say-hello   | JOB_FINISHED
d62b194a-e20b-4111-a407-3669fb4c89e6 | test3  | test     | say-hello   | JOB_FINISHED
```

Where did the output go?

Job working directories are created as:

`data/<workflow>/<name>_<id>`

```
[BalsamDB: myProject] $ ~/myProject/data/test>   ls

test1_0796bd50   test2_421e6df8  test4_e73325df  test6_c7ed41fd  test8_2e6c4914
test10_f4d77a25  test3_d62b194a  test5_79becbd6  test7_dda7cdd3  test9_4a12cb47
```

Modifying Tasks

Modify BalsamJob state from command line:

```
BalsamDB: myProject] $  balsam modify dda7 state RESTART_READY
job state changed to:  RESTART_READY
```

Or with more flexible Python API:

```
[BalsamDB: myProject] $  python

>>> from balsam.launcher.dag import BalsamJob
>>> BalsamJob.objects.filter(num_nodes__lte=100,
state="JOB_FINISHED").update(state="RESTART_READY")
```

Populating DB with Python

```
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',
    )
```

Populating DB with Python

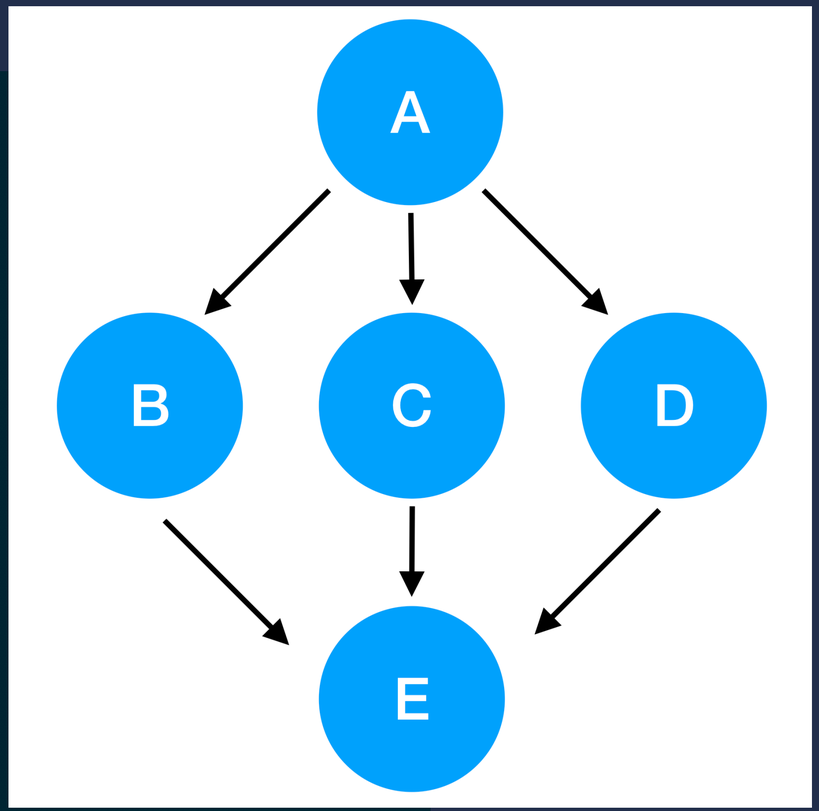
```
for (dirpath, dirnames, filenames) in os.walk(top):
    xyz_files = [f for f in filenames if f.endswith('.xyz')]
    for f in xyz_files:
        name, _ = os.path.splitext(f)
        workflow = os.path.basename(dirpath)
        xyz_path = os.path.join(dirpath, f)
        job = prep_job(name, workflow,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

Defining Dependencies

```
from balsam.launcher.dag import (
        add_job, add_dependency)

A = add_job(name="A", application="generate")
B,C,D = [
        add_job(
            name=name,
            application="simulate",
            input_files=name+".inp"
        )
        for name in "BCD"
        ]
E = add_job(name="E", application="reduce", input_files="*.out")

for job in B,C,D:
    add_dependency(A, job)
    add_dependency(job,E)
```



Questions?

/projects/SDL_Workshop/training/Balsam

balsam.readthedocs.io

msalim@anl.gov

Workflow-as-code approaches

http://swift-lang.org/Swift-T/index.php

http://parsl-project.org/

- Swift language or Python library for expressing workflows
- Transparent data-flow & concurrency
- Swift/T implementation: compile & run with MPI
- Parsl: numerous execution backends

Parallelize-a-script mindset

contrasts with Balsam's

Submit-many-tasks mindset

No central task database

Less tailored to long campaigns