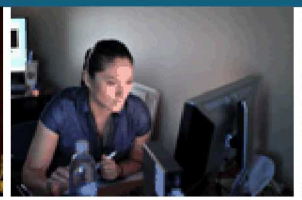
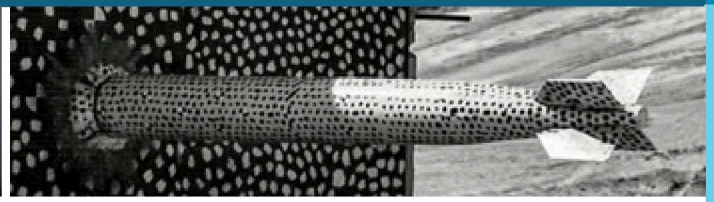
SAND2020-1164PE

Manage Internal Queue w/ Flux for Common Workflows





Slides & Content Generated by

Anthony Agelastos, Gary Lawson, Dong Ahn, Mark Grondona, Stephen Herbein

Slides Presented by

LLNL Flux Team

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Acknowledgments

This presentation would not have been possible without the following direct contributions

LLNL
- Dong Ahn
- Jim Garlick
- Mark Grondona
- Stephen Herbein
- Daniel Milroy
- Tapasya Patki
- Francesco Di Natale

SNL
- Anthony Agelastos
- Gary Lawson

… with motivation/need exhibited from the following indirect contributions

SNL
- Micheal Glass (ASC IC/ATDM)
- Brian Adams (ASC Dakota)
- Paul Wolfenbarger (ASC CompSim DevOps)
- Richard Drake (ASC CompSim DevOps)
- Steve Monk (FOUS)
- Ann Gentile (FOUS)

Executive Summary

There are many workflows that require launching and processing jobs in bulk.

- Testing, Uncertainty Quantification (UQ), Verification and Validation (V&V), optimization, parameter study, and Design of Experiments (DoE) are typical activities that require bulk processing.

The activities above are lumped into 2 the following 2 categories.

Unit & Regression Testing

Development teams have tens of thousands of test cases that, ideally, would be run for every commit (many times per day) using low-to-medium scale resources.

Ensemble Processing

Analysts may need to have thousands-to-millions of fast test cases for a study using low-to-large resources.

These cases, individually, are fast but their quantity exceeds what is reasonable for submitting to traditional HPC queues which are optimized for lower numbers of longer-running jobs.

Executive Summary (contd.)

Purpose

- This presentation provides a demonstration of using Flux to address these use cases.

Method

- A simple test case was created for the demonstration.
- A modern Flux installation was created (the version bundled with TOSS is stale).
- Caveats, experiences, results, and methods for extending to more complicated problems are disseminated.
- Discuss observed barrier to entry from someone _not_ on the development team.
- The example is simple to help prospective users understand Flux's small barrier to entry for evaluation.

Results

- Flux installation was robust and easy.
- Getting Flux working as an internal queue for problems is very fast and reliable.
- Information supporting both activity categories is presented.

Installing Flux

Experiences & Notes

- Instructions for "developer" and Spack builds were followed and resulted in success *on the 1st try*.
  - Intel compiler versions 16, 19 & GNU compiler versions 4.8, 8.2 were tried and all resulted in a build of Flux on TOSS.
- Certain OpenMPI versions contain bugs and cause cause issues with Flux.
  - Bug: https://github.com/open-mpi/ompi/issues/6730
  - PR: https://github.com/open-mpi/ompi/pull/6764
  - Currently, avoid most versions >= 3.1.0 (issue confirmed on 3.1.0, 3.1.3, 4.0.0, 4.0.1)
  - Test case was performed with OpenMPI 1.10.
- Test suite worked as expected and displayed passing/failing tests.

Compiling and installing Flux was easy & reliable on TOSS/RHEL.

Unit & Regression Test Case: Description

`mpi_greet_after_sleep` is a simple MPI application that does the following:

- `MPI_Barrier()` (syncs ranks after initialization)
- `sleep(20)` (sleeps for 20 sec.)
- `get_time_of_day()` (gets current date & time)
- `MPI_Barrier()` (syncs ranks)
- rank 0 process spits out location and timing info

```
$ srun -N 2 -n 72 ./mpi_greet_after_sleep
Hello WORLD! I am node ama42 with 72 ranks
Started at: Mon Jan 27 22:13:55 2020
Finished at: Mon Jan 27 22:14:15 2020
```

Simple application is used to create example test case

Unit & Regression Test Case: Using with Flux CLI

Get an allocation on 4 nodes

- `salloc -N 4 --time=4:00:00`

Tell Flux you want backfill

- `export FLUX_QMANAGER_OPTIONS="queue-policy=easy"`

Start Flux upon the 4 nodes with 1 Flux instance per node

- `srun --pty --mpi=none -N 4 -n 4 flux start -o,-S,log-filename=mpi_greet_after_sleep.log`

Submit a case

- `flux mini submit -N 4 -n 144 -t 30 ./mpi_greet_after_sleep >>submit_id 2>&1`

Get job output if desired

- `flux job attach \`cat submit_id\``

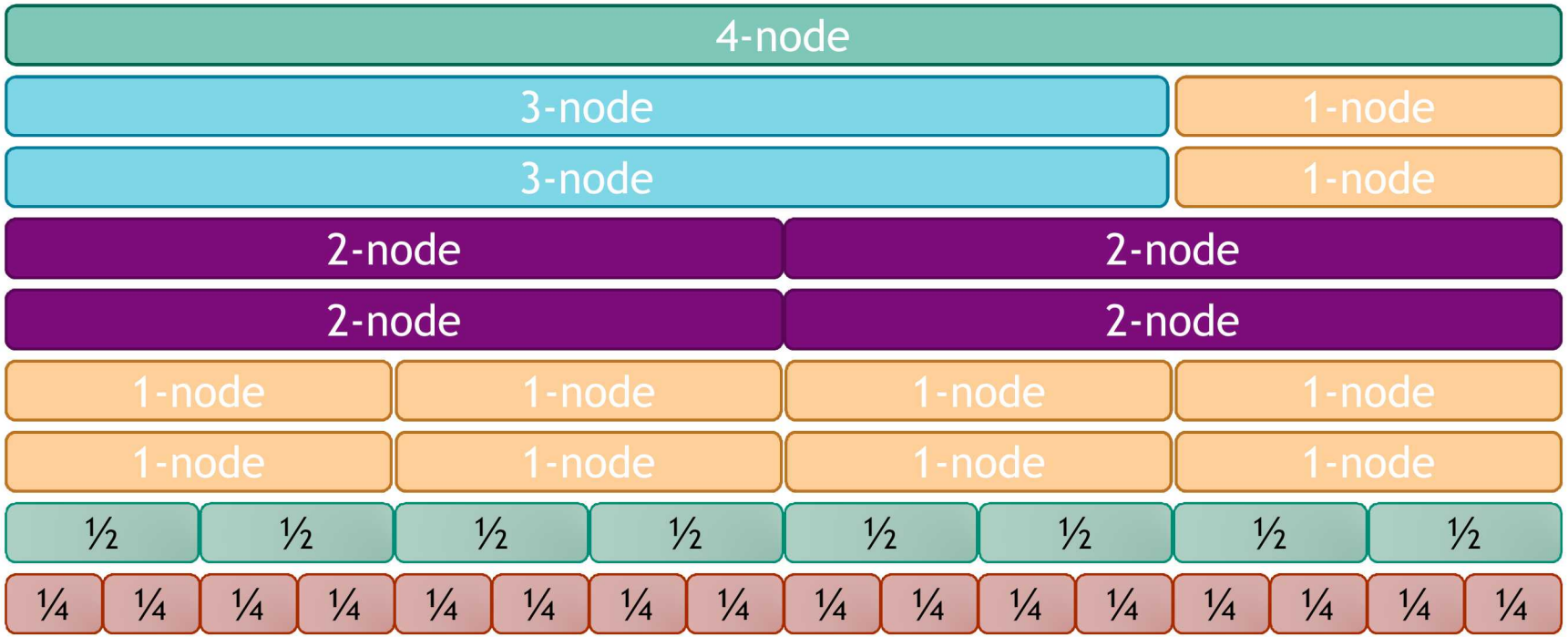
This is a high-level overview of the ***entire*** process! There are not many steps!

Unit & Regression Test Case: Workload

The workload contains the following 41 jobs:

- 1x 4-node job
- 2x 3-node jobs
- 4x 2-node jobs
- 10x 1-node jobs
- 8x ½-node jobs
- 16x ¼-node jobs

The workload was designed to take ≅3:00 if the entire 4-node allocation was wholly utilized.

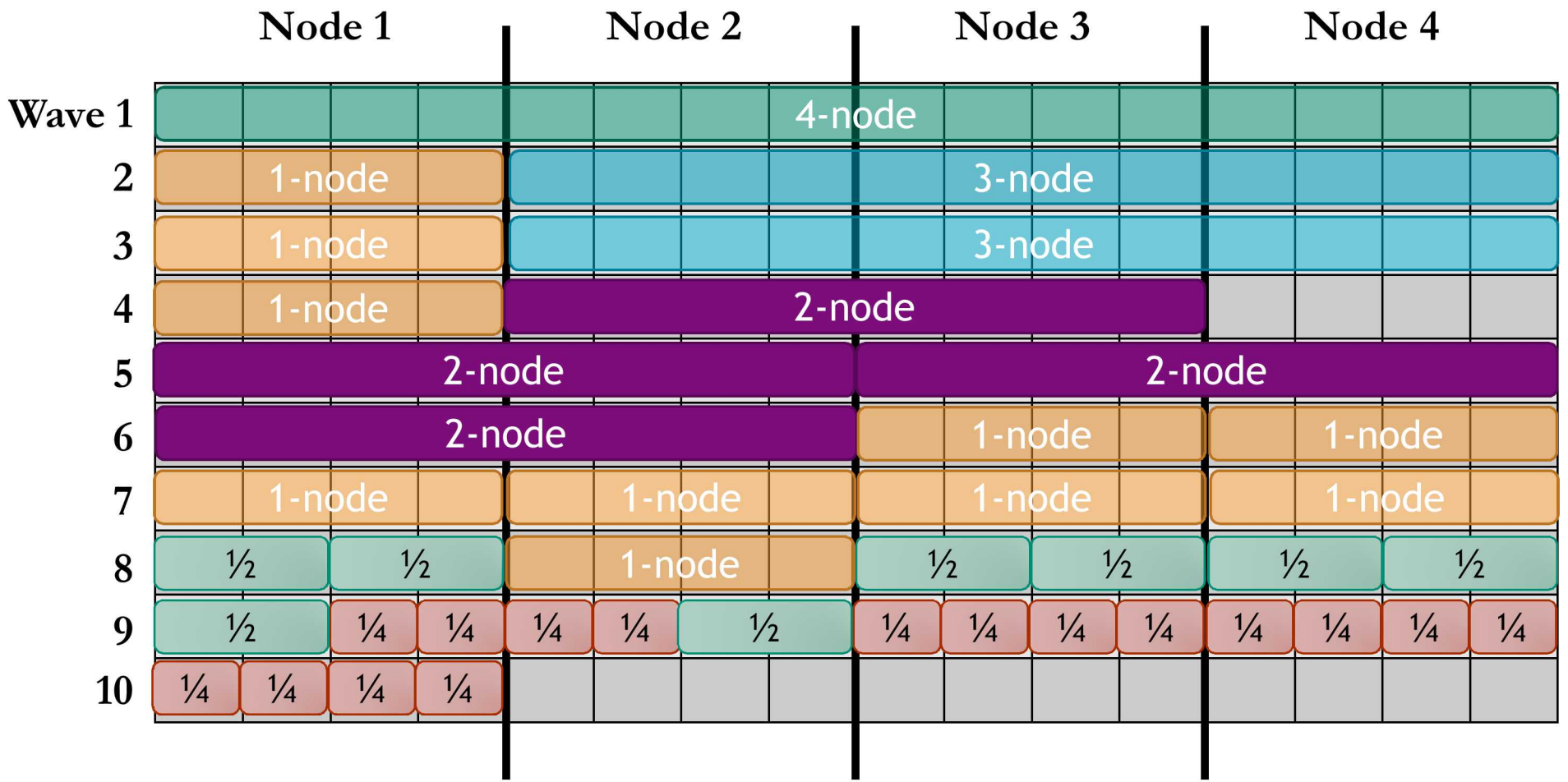


Workload contains jobs that only require a fraction of a node's resources up to 4 nodes.

Unit & Regression Test Case: Ordered Submission

Submission Order (*# nodes*)

1.	4	22.	½
2.	3	23.	½
3.	3	24.	½
4.	2	25.	½
5.	2	26.	¼
6.	2	27.	¼
7.	2	28.	¼
8.	1	29.	¼
9.	1	30.	¼
10.	1	31.	¼
11.	1	32.	¼
12.	1	33.	¼
13.	1	34.	¼
14.	1	35.	¼
15.	1	36.	¼
16.	1	37.	¼
17.	1	38.	¼
18.	½	39.	¼
19.	½	40.	¼
20.	½	41.	¼
21.	½		

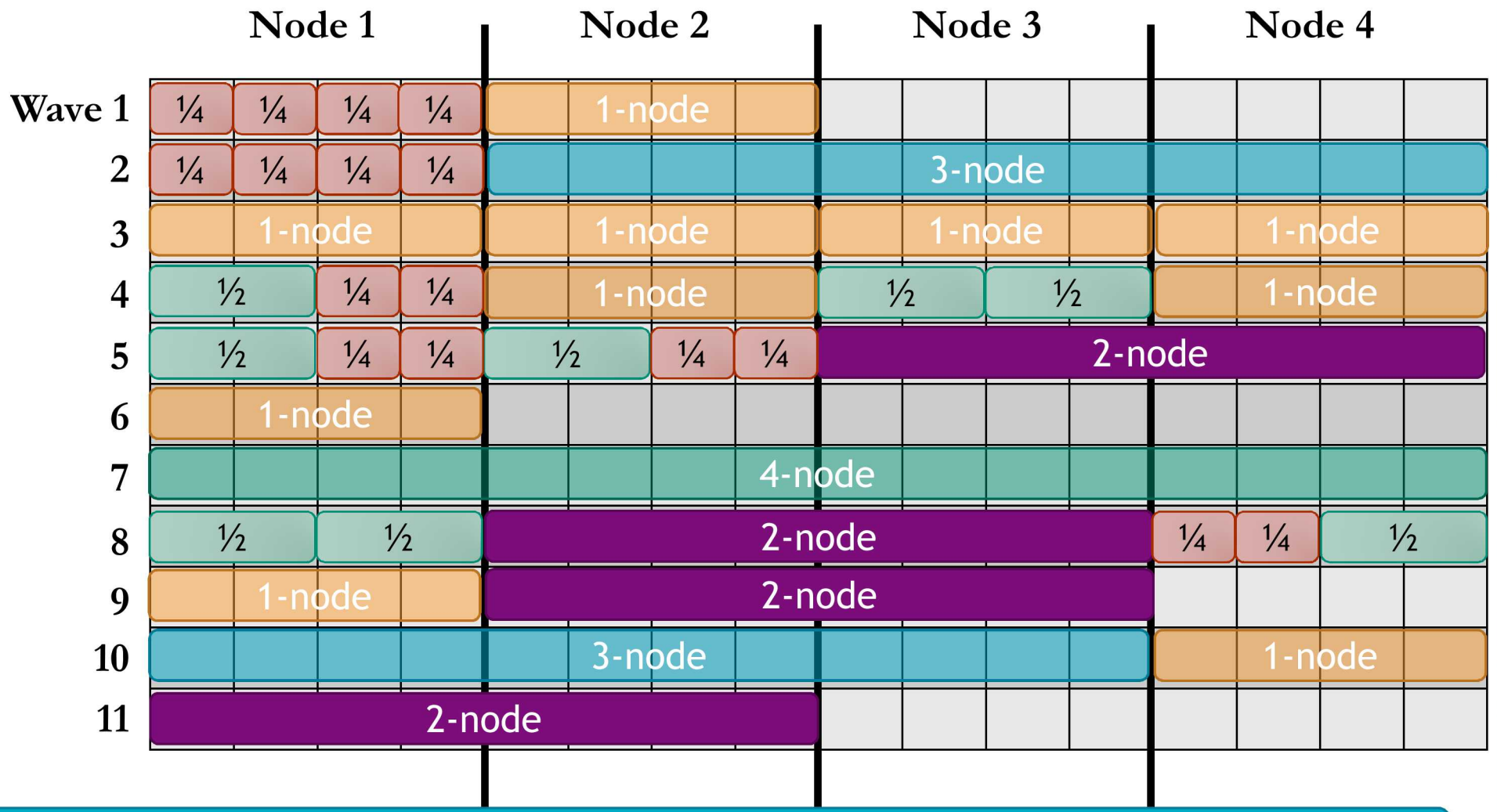


The packing was nearly optimal; runtime variations and inconsistent start times are cause for extra wave.

Unit & Regression Test Case: Random Submission

Submission Order (*# nodes*)

1.	¼	22.	½
2.	1	23.	2
3.	¼	24.	¼
4.	¼	25.	½
5.	¼	26.	1
6.	3	27.	4
7.	¼	28.	¼
8.	1	29.	¼
9.	¼	30.	¼
10.	¼	31.	2
11.	1	32.	½
12.	1	33.	½
13.	1	34.	¼
14.	¼	35.	2
15.	½	36.	¼
16.	¼	37.	1
17.	1	38.	3
18.	½	39.	1
19.	1	40.	½
20.	¼	41.	2
21.	½		



Packing was less optimal due to queue depth optimization + job start/stop not synced.

Unit & Regression Test Case: Random Submission (contd.)

Extra Specialization

- Can specify queue depth optimization parameter via:
  `FLUX_QMANAGER_OPTIONS="queue-policy=easy queue-params=queue-depth=<#>"`
- Queue depth = 2 had 12 waves
- Queue depth = 20 had 11 waves
- Queue depth = 41 had 10 waves
- Queue depth = 100 had 10 waves
- More detailed discussion regarding this at https://github.com/flux-framework/flux-sched/issues/572

Simple tuning of the queue depth reduced the amount of waves down to 10.

Enhancing the Test Case: Submitting Scripts

OLD: Submit a case (directly calling application `mpi_greet_after_sleep`)

- `flux mini submit -N 4 -n 144 -t 30 ./mpi_greet_after_sleep`

NEW: Submit a case (calling script `internal_script.sh`)

- `flux mini submit -N 4 -n 144 flux start ./internal_script.sh`

NEW: Within script (`internal_script.sh`) execute parallel application with this

- `flux mini run -N 4 -n 144 -t 30 ./mpi_greet_after_sleep`

Scripts can be run as well and each script can run different sized applications that Flux will manage.

Limitations with Large Ensemble Processing

Flux Interface

- Flux supports a command line interface (which was leveraged for the technology demonstration herein).
- Flux also provides a Python API and other methods for ingesting jobs.
  - Python API is the preferred method for interacting with Flux for medium-to-complicated workflows.

Issue with Submitting Large Ensembles

- Methods described herein have a "slow" ingest rate of 4-8 job/s. This would require almost 2 days for 1 million jobs, which may not work for such a large ensemble. ☹
- Thankfully, they have bulk loaders and methods directly accessible with their Python API to drastically increase this to over 600 job/s (needing about 30 minutes to submit a full stack of 1 million jobs). ☺
  - Refer to discussions within https://github.com/flux-framework/flux-sched/issues/573 for more details.

Flux has a diverse set of accessor methods that can tailor to many workflows.

Conclusions & Future Work

Conclusions

- Flux was easy to install, run, and extend to suit the simple example herein.
- Flux is very easy to use for managing internal queues and provides good behaviors right out of the box with minimal tinkering.
- Flux is the most comprehensive solution tested to date for dynamic workflows consisting of heterogeneous jobs.

Future Work

- SNL Dakota team is crafting a dynamic, heterogeneous optimization case that involves real simulation codes being driven by Dakota. Flux will be put in the loop to manage the resource allocations as a more expanded example case to share with the broader Dakota user base.
- Data herein will be disseminated to various teams whose testing strategies can be enhanced with Flux.