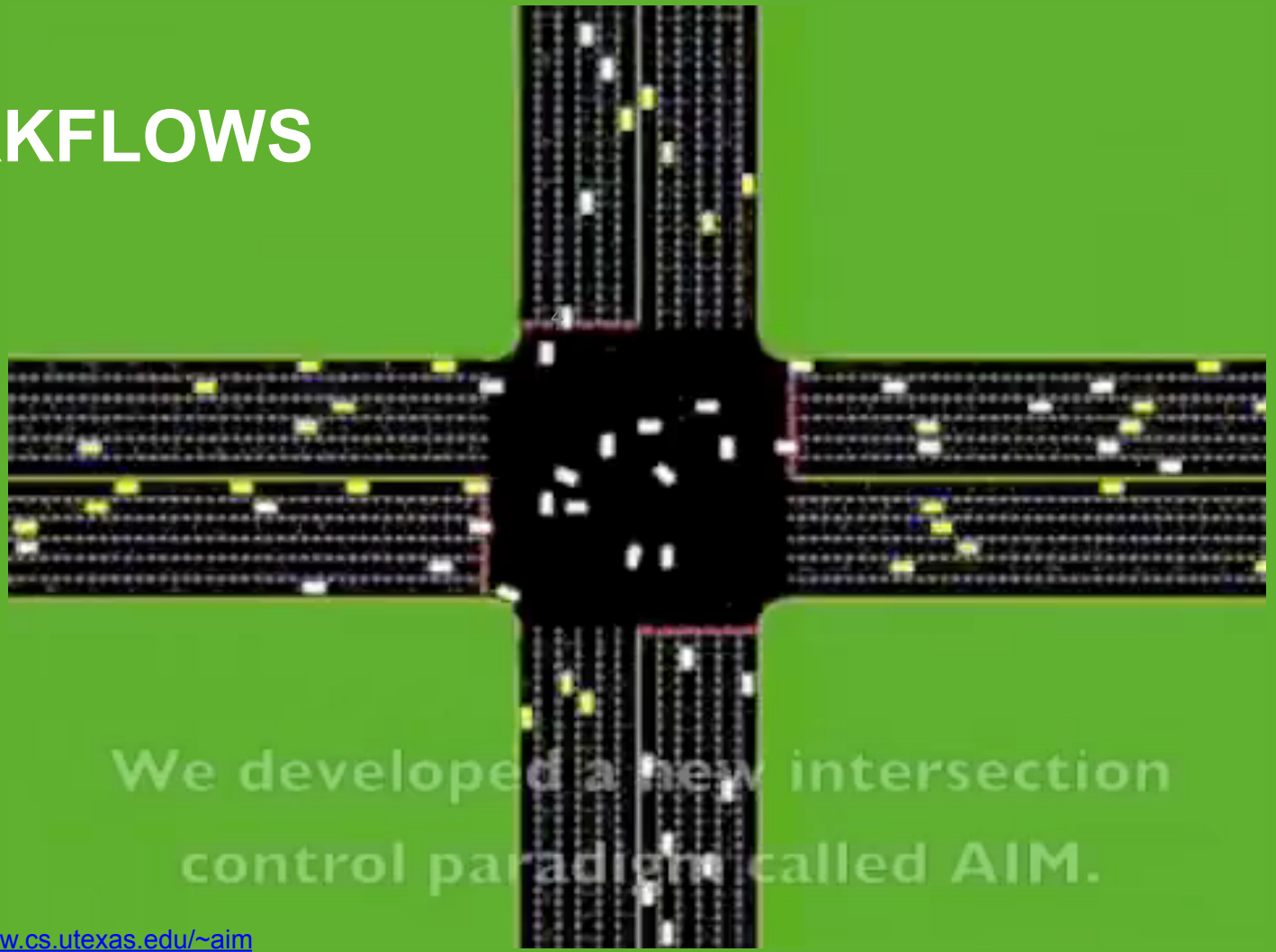
EXECUTING WORKFLOWS

Thomas Uram, Misha Salim, Taylor Childers, and the Data Science group
Argonne Leadership Computing Facility

WHY SHOULD YOU CARE ABOUT WORKFLOWS?

- You came to the workshop to learn to use large-scale parallel systems for simulation, data, and learning:
  - architectures
  - programming models
  - communication
  - solvers
  - visualization
  - profiling
  - optimizing for node-level performance
  - scaling parallel applications to tens of thousands of nodes
- Once you've written your application and achieved near-peak node-level performance and scalability, what comes next?

WORKFLOWS



Source: http://www.cs.utexas.edu/~aim

ALLOCATION EXAMPLES

- Small number of large jobs running on Mira over course of the year followed by some analysis
  - 40 jobs x 16K nodes * 16 cores/node * 12 hours ~= 100M core-hours
  - ~Reasonable to manage manually
- Large number of simulation+analysis jobs running across multiple facilities, requires coordination of jobs submitted to multiple schedulers, large/long data transfers, and interaction with project storage and archival storage
  - Programmatic management would be a clear benefit

Workflows can save you!
…but you might have to code them yourself!

WHY USE WORKFLOWS?

- Automate job submission
- Simplify computational campaigns
- Increase concurrency by disentangling data dependencies
- Robustness: Improve error handling and recovery (retries)
- Coscheduling of multiple resources
- Systematize data management
- Provenance/Metadata tracking
  - Validation
  - Reuse

WHAT IS A WORKFLOW?

- It depends who you ask
- Basically a collection of jobs to be run
  - Could be a sequence of individual jobs
  - Could be a sequence of varying numbers of jobs
- Available means of describing and running workflows
  - Script jobs
  - Job dependencies
  - Ensemble jobs
  - In situ
  - Custom workfows
  - Workflow software

WORKFLOW VIA SCRIPT JOB

- You can submit a job that executes your application
  - qsub -q cache-quad -n 512 -t 60 -A yourproject application.exe
- Alternatively, you can submit a script job that executes multiple applications sequentially
  - qsub -q cache-quad -n 512 -t 60 -A yourproject script.sh

script.sh

```
aprun -n 512 -N 1 application.exe
aprun -n 512 -N 1 application.exe
```

- With this approach, you wait in the queue one time to run your application multiple times
- However, small long jobs tend to stay in the queue longer than small short jobs
  - It may be better to submit individual jobs with dependencies

WORKFLOW VIA COBALT JOB DEPENDENCIES

- A simple way to achieve a linear workflow is simply to set dependencies between your jobs
  - qsub -q cache-quad -n 512 -t 60 -A yourproject a.out

    Job 12345 submitted
  - qsub -q cache-quad -n 512 -t 60 -A yourproject --dependencies 12345 a.out
  - qsub -q cache-quad -n 512 -t 60 -A yourproject --dependencies 12345:12346 a.out

- Is there an advantage to setting job dependencies?
  - Dependent jobs accumulate score more quickly
- How many jobs can be submitted at once?

```
qstat -Q
Name         Users    Groups   MinTime   MaxTime    MaxRunning  MaxQueued  MaxUserNodes  MaxNodeHours
cache-quad   None     None     None      06:00:00   10          20         None          None
flat-quad    None     None     None      06:00:00   10          20         None          None
```

  - On Theta, max queued is set to 20

multiple independent short jobs

WORKFLOW VIA ENSEMBLE JOBS: THETA

```
for x in {0..100}
do
   # background one aprun call per job
   aprun -n 1 -N 1 myapp.py --args $x &
   sleep 1
done
wait
```

Note:

- There is a system limitation of 1000 simultaneous apruns per Cobalt script job
- You should include a sleep call between aprun invocations

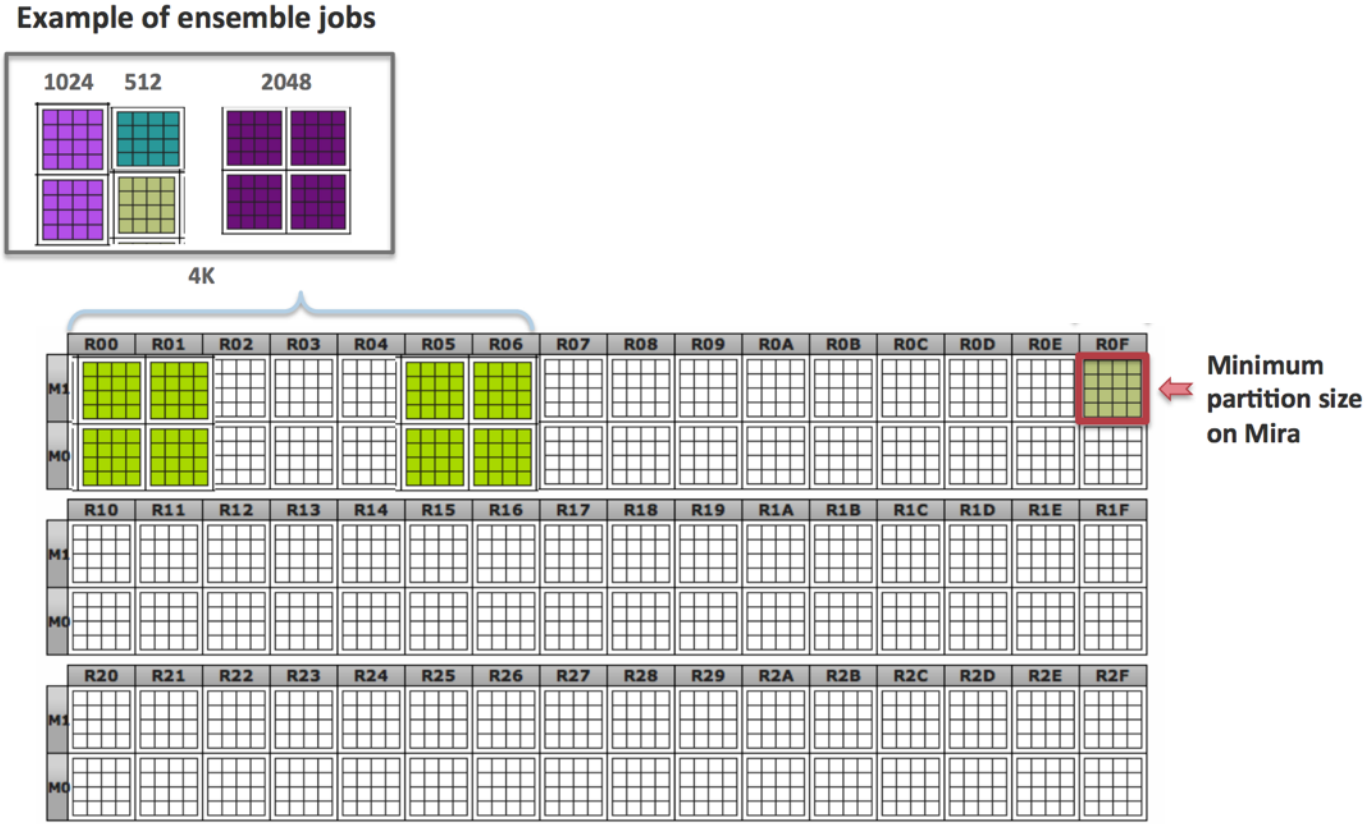
For more details of running ensemble jobs on Theta, see Paul Rich's slides
https://www.alcf.anl.gov/files/rich-ensemble-jobs-2017_1.pdf

WORKFLOW VIA ENSEMBLE JOBS: MIRA

http://trac.mcs.anl.gov/projects/cobalt/wiki/BGQUserComputeBlockControl

```
#!/bin/bash
BLOCKS=`get-bootable-blocks --size 512
$COBALT_PARTNAME`

for BLOCK in $BLOCKS
do
    boot-block --block $BLOCK &
done
wait

for BLOCK in $BLOCKS
do
    runjob --block $BLOCK : ./my_binary &
done
wait

for BLOCK in $BLOCKS
do
    boot-block --block $BLOCK --free &
done
wait
```

WORKFLOW VIA ENSEMBLE JOBS: MIRA



WORKFLOW ACROSS ALCF SYSTEMS

- How can I set job dependencies across multiple resources (e.g. Mira and Cooley)?
  - ALCF does not provide a means of doing this currently†
  - However, Cetus and Cooley mount the Mira filesystems. Analysis jobs run on Cooley without needing to transfer the data.

SOFTWARE FOR MANAGING WORKFLOWS

This manual effort can be ameliorated by scripting your job submission so that new jobs are automatically submitted as you drop below the max_queued limit. Multiple toolkits are available for this purpose.

Balsam (https://www.alcf.anl.gov/balsam)
Balsam is an ALCF project for managing jobs and workflows on our systems. Balsam allows users to define campaigns of many jobs with interdependencies and control how they execute on the systems, either by submitting them to the job scheduler, or running them within a single job for higher throughput.

Swift (http://swift-lang.org)
Swift provides a C-like language for expressing workflows that consist of command-line invocations, and an engine for managing their execution. For a recent Python implementation of Swift concepts, also see Parsl (http://parsl-project.org).

SOFTWARE FOR MANAGING WORKFLOWS

Fireworks (https://pythonhosted.org/FireWorks)

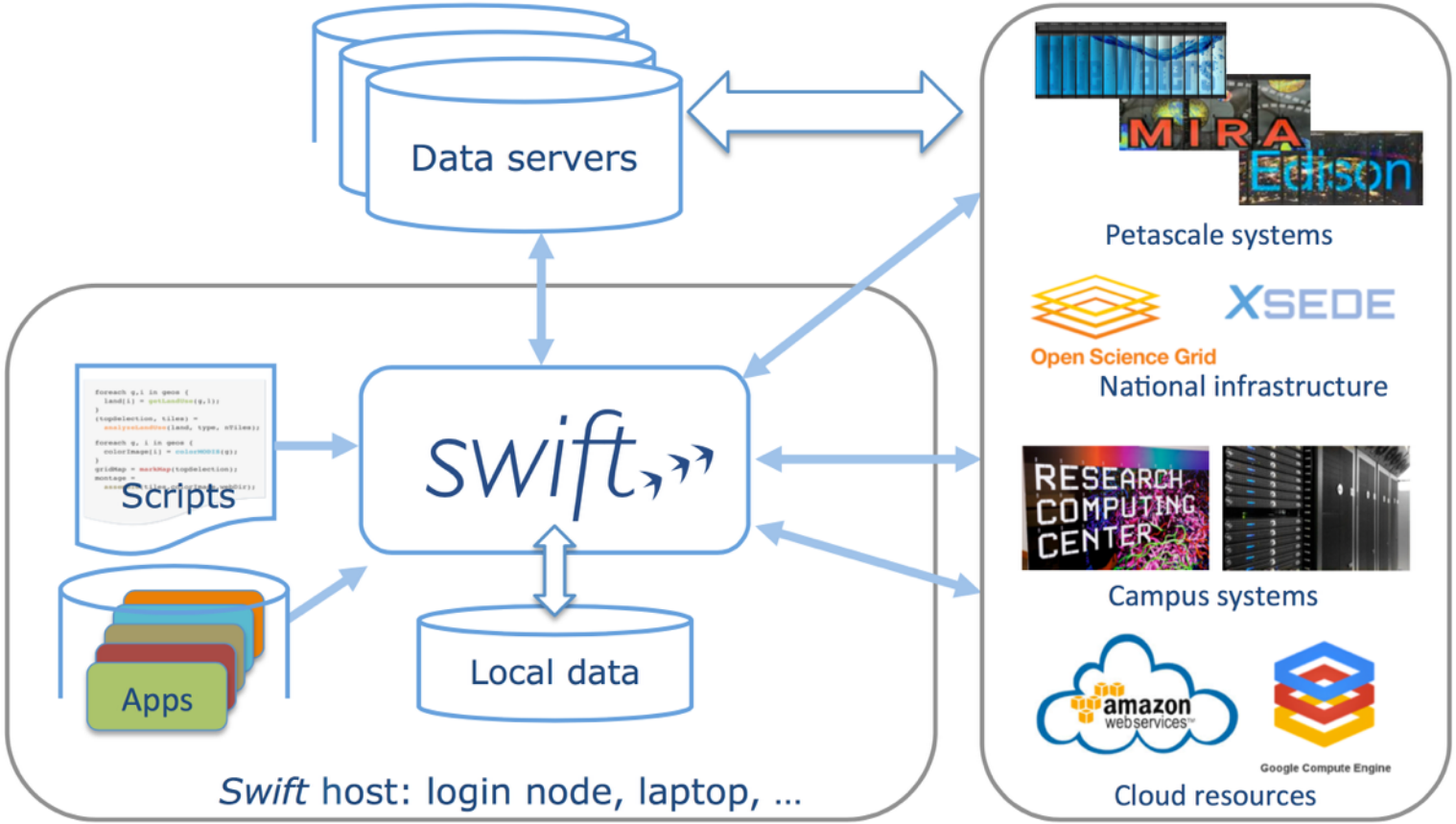
FireWorks is a free, open-source code for defining, managing, and executing workflows. Complex workflows can be defined using Python, JSON, or YAML, are stored using MongoDB, and can be monitored through a built-in web interface. Workflow execution can be automated over arbitrary computing resources, including those that have a queueing system. FireWorks has been used to run millions of workflows encompassing tens of millions of CPU-hours across diverse application areas and in long-term production projects over the span of multiple years.

Pegasus/Condor (https://pegasus.isi.edu)

Pegasus bridges the scientific domain and the execution environment by automatically mapping high-level workflow descriptions onto distributed resources. It automatically locates the necessary input data and computational resources necessary for workflow execution. Pegasus enables scientists to construct workflows in abstract terms without worrying about the details of the underlying execution environment or the particulars of the low-level specifications required by the middleware

Many others…

Swift enables execution of simulation campaigns across multiple HPC and cloud resources

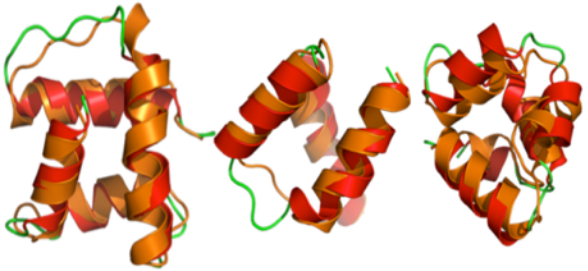


See https://www.alcf.anl.gov/swift

When do you need HPC workflow?

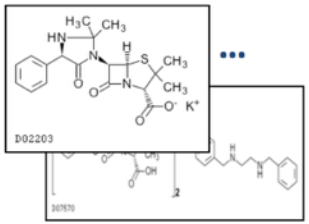
Example application: protein-ligand docking for drug screening

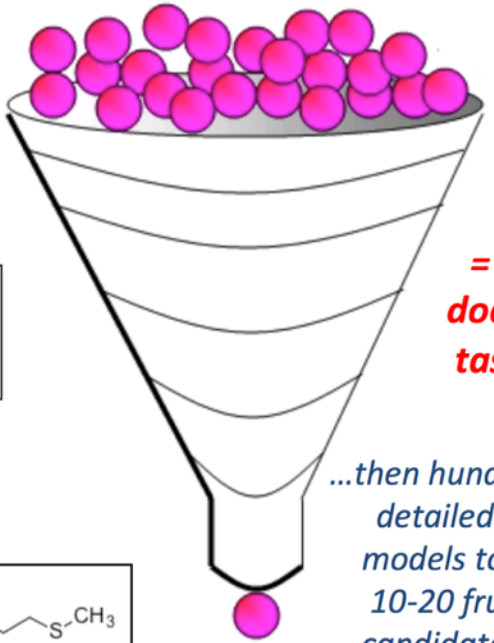
O(10) proteins implicated in a disease



X

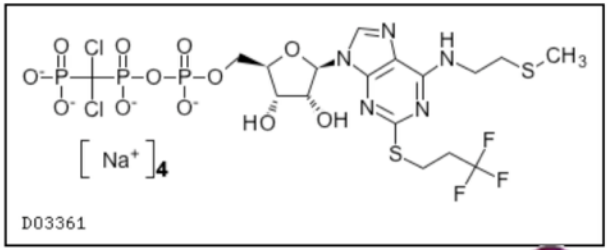
O(100K) drug candidates

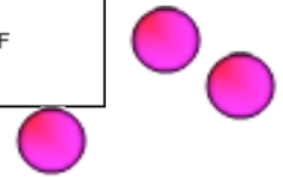




= 1M docking tasks…

…then hundreds of detailed MD models to find 10-20 fruitful candidates for wetlab & APS crystallography





Expressing this many task workflow in Swift

For protein docking workflow:

```
foreach p, i in proteins {
   foreach c, j in ligands {
      (structure[i,j], log[i,j]) =
       dock(p, c, minRad, maxRad);
}
scatter_plot = analyze(structure)
```

To run:

```
swift -site cooley,blues dock.swift
```

BALSAM

We are developing Balsam to manage workflows on systems at ALCF (and elsewhere). What is Balsam?

- A workflow system for managing large campaigns of interdependent jobs (unlimited queue depth)
- A service for managing flow of jobs into scheduler (automated job submission)
- Flexible launcher for high throughput execution by side-stepping the queue (user control)
- Built on Python and Django
  - Powerful Django object-relational mapper (ORM)
  - Support for many databases (sqlite, MySQL, Postgres)
- Handles data transfers for stage-in and stage-out
- Modular interface to job sources, schedulers, and transfer utilities

BALSAM INSTALLATION ON THETA

```
export PATH=$PATH:$HOME/bin:/opt/intel/python/2017.0.035/intelpython35/bin
conda config --add channels intel
conda create --name balsam_env intelpython3_full python=3
source activate balsam_env
cp /opt/cray/pe/mpt/7.6.0/gni/mpich-intel-abi/16.0/lib/libmpi* ~/.conda/envs/
balsam_env/lib/
export LD_LIBRARY_PATH=~/.conda/envs/balsam_env/lib:$LD_LIBRARY_PATH

git clone git@xgitlab.cels.anl.gov:datascience/balsam.git
cd balsam
pip install -e .
```

BALSAM INSTALLATION ON THETA

```
export PATH=$PATH:$HOME/bin:/opt/intel/python/2017.0.035/intelpython35/bin
conda config --add channe     ntel
conda create --name balsa      in      thon3_full python=3
source activate balsam_env
cp /opt/cray/pe/mpt/7.6.0/g      -intel-abi/16.0/lib/libmpi* ~/.conda/envs/
balsam_env/lib/
export LD_LIBRARY_PATH=      vs/balsam_env/lib:$LD_LIBRARY_PATH

git clone git@xgitlab      s.anl.go      datascience/balsam.git
cd balsam
pip install -e .
```

```
module load miniconda-3.6  # or any Python3.6
module load balsam
```

BALSAM USAGE

Add an application to the database

```
balsam app --name myapp --desc “my application" --exec myapp.py
```

Add a job to the database (options mirror qsub/aprun options); unlimited depth

```
balsam job --name myjob --workflow balsam_tutorial --application myapp \
           --wall-min 60 --num-nodes 8 --ranks-per-node 1
```

Launch the Balsam service to automatically flow jobs from the database to the scheduler (job submission will honor user-defined limits)

```
balsam service
```

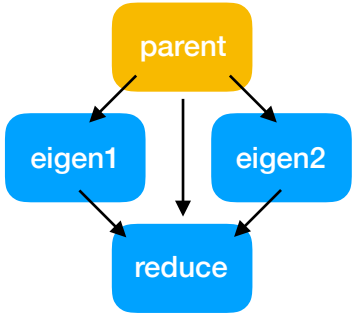
Submit Balsam launcher jobs manually to run jobs from the database continuously in the context of a single Cobalt job (e.g. ensembles)

```
qsub -n 128  balsam launcher --consume-all --max-ranks 4
qsub -n 4096 balsam launcher --consume-all --max-ranks 4
```

Submit jobs from within jobs, via the post-process script…

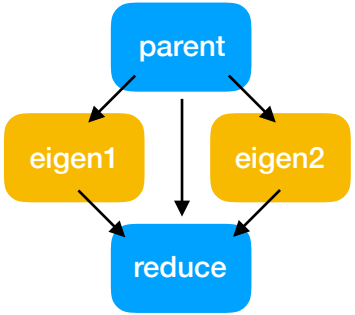
BALSAM COMMAND LINE INTERFACE (CLI)

```
$ balsam ls
                                  job_id |  name |        workflow | application |   state
-------------------------------------------------------------------------------------------
43e4d1be-fc30-4618-91e0-65b100bc18a8 | myjob | balsam_tutorial | myapp       | CREATED
77d5e32e-8595-4162-ac21-1b86835a6cd0 | myjob | balsam_tutorial | myapp       | CREATED
ebcdc382-d548-4d1d-8c6a-53f3ac59d64e | myjob | balsam_tutorial | myapp       | CREATED
e705d917-b196-4cf4-9b6f-c2ea453e14ae | myjob | balsam_tutorial | myapp       | CREATED
35a570d9-2120-42a8-9984-387285827606 | myjob | balsam_tutorial | myapp       | CREATED
50d8bea1-977a-4028-84ac-022686c7949b | myjob | balsam_tutorial | myapp       | CREATED
eb613637-efc2-49c0-9105-bac8cd48fdd8 | myjob | balsam_tutorial | myapp       | CREATED

$ balsam rm jobs --id 43e4d1be-fc30-4618-91e0-65b100bc18a8

$ balsam --help
   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
    modify              alter job or application
    rm                  remove jobs or applications from the database
    qsub                add a one-line bash command or script job
    killjob             Kill a job without removing it from the DB
    mkchild             Create a child job of a specified job
    launcher            Start an instance of the balsam launcher
    dbserver            Start/stop database server process
    init                Create new balsam DB
    service             Start an instance of the balsam metascheduler service
```



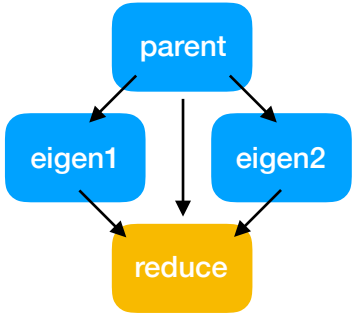
parent.py¶

```
'''Generate random matrices'''
import random
import numpy as np

num_outputs = random.randint(2,6)

for i in range(num_outputs):

    out_path = f"output{i}.npy"
    dim = random.randint(10,100)
    data = np.random.random((dim, dim))
    data = 0.5*(data + data.T)
    np.save(out_path, data)
```



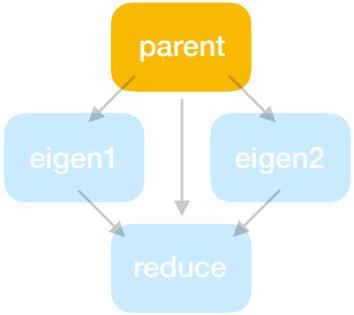
eigen.py¶

```
'''Get eigenvalues'''
import sys
import numpy as np

mat_file = sys.argv[1]
matrix = np.load(mat_file)
eigvals = np.linalg.eigvalsh(matrix)
np.save("eigvals", eigvals)
```



reduce.py¶

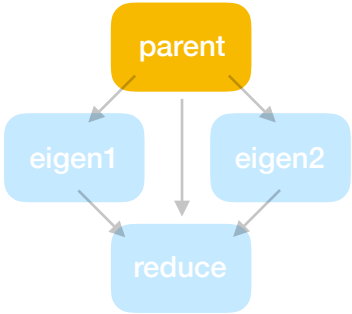
```
'''Read and sort many eigenvalues'''
import glob
import numpy as np

eig_files = glob.glob("eigvals*")
eigs = [np.load(f) for f in eig_files]
eigs = np.concatenate(eigs)
eigs.sort()

lo5 = '\n'.join(str(x) for x in eigs[:5])
hi5 = '\n'.join(str(x) for x in eigs[-5:])

with open("results.dat", 'w') as fp:
    fp.write("Lowest 5:\n")
    fp.write(lo5)
    fp.write("\nHighest 5:\n")
    fp.write(hi5)
```



parent.py¶

```
'''Generate random matrices'''
import random
import numpy as np

num_outputs = random.randint(2,6)

for i in range(num_outputs):

    out_path = f"output{i}.npy"
    dim = random.randint(10,100)
    data = np.random.random((dim, dim))
    data = 0.5*(data + data.T)
    np.save(out_path, data)
```

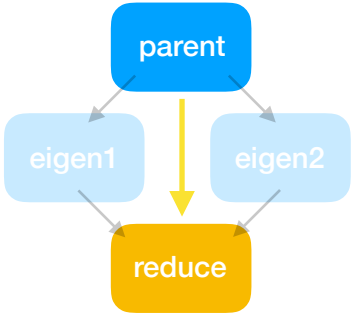
- **Manipulate job DB and create task graphs dynamically**
- **Balsam Python API can be invoked in post-execution scripts which are specified for each application**



parent-post.py¶

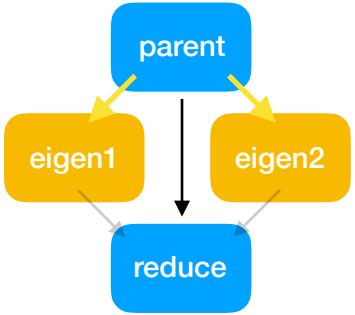
```
'''Post-processing: dynamically create jobs from output files'''
import glob
import balsam.launcher.dag as dag

reduce_job = dag.spawn_child(name = "reduce", application="reduce")

out_files = glob.glob("output*.npy")

for i, fname in enumerate(out_files):
    eig_job = dag.spawn_child(name = f"eigen{i}", application = "eigen",
                              input_files = fname, application_args = fname)

    dag.add_dependency(parent=eig_job, child=reduce_job)
```

Inherit job-specific environment



parent-post.py¶

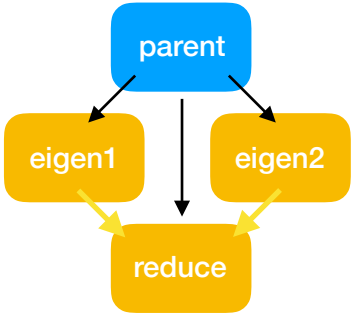
```
'''Post-processing: dynamically create jobs from output files'''
import glob
import balsam.launcher.dag as dag

reduce_job = dag.spawn_child(name = "reduce", application="reduce")

out_files = glob.glob("output*.npy")

for i, fname in enumerate(out_files):
    eig_job = dag.spawn_child(name = f"eigen{i}", application = "eigen",
                              input_files = fname, application_args = fname)

    dag.add_dependency(parent=eig_job, child=reduce_job)
```



specify files to symlink & and command line args

parent-post.py¶

```
'''Post-processing: dynamically create jobs from output files'''
import glob
import balsam.launcher.dag as dag

reduce_job = dag.spawn_child(name = "reduce", application="reduce")

out_files = glob.glob("output*.npy")

for i, fname in enumerate(out_files):
    eig_job = dag.spawn_child(name = f"eigen{i}", application = "eigen",
                              input_files = fname, application_args = fname)

    dag.add_dependency(parent=eig_job, child=reduce_job)
```



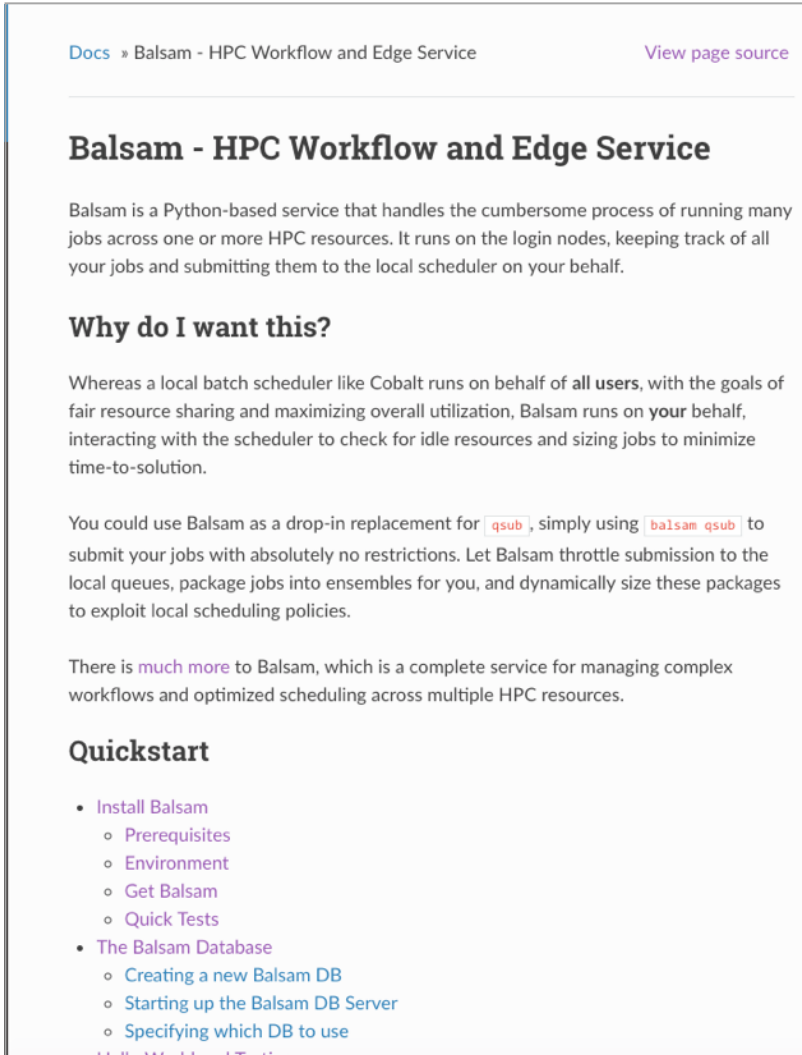
by default, child makes symlinks to all parent files

parent-post.py¶

```
'''Post-processing: dynamically create jobs from output files'''
import glob
import balsam.launcher.dag as dag

reduce_job = dag.spawn_child(name = "reduce", application="reduce")

out_files = glob.glob("output*.npy")

for i, fname in enumerate(out_files):
    eig_job = dag.spawn_child(name = f"eigen{i}", application = "eigen",
                              input_files = fname, application_args = fname)

    dag.add_dependency(parent=eig_job, child=reduce_job)
```

MORE INFO

- Code available at xgitlab.cels.anl.gov/datascience/balsam
- Documentation
  - Generated docs exist in repo under balsam/docs
  - Online documentation at balsam.alcf.anl.gov



Docs » Balsam - HPC Workflow and Edge Service View page source

Balsam - HPC Workflow and Edge Service

Balsam is a Python-based service that handles the cumbersome process of running many jobs across one or more HPC resources. It runs on the login nodes, keeping track of all your jobs and submitting them to the local scheduler on your behalf.

Why do I want this?

Whereas a local batch scheduler like Cobalt runs on behalf of **all users**, with the goals of fair resource sharing and maximizing overall utilization, Balsam runs on **your** behalf, interacting with the scheduler to check for idle resources and sizing jobs to minimize time-to-solution.

You could use Balsam as a drop-in replacement for `qsub`, simply using `balsam qsub` to submit your jobs with absolutely no restrictions. Let Balsam throttle submission to the local queues, package jobs into ensembles for you, and dynamically size these packages to exploit local scheduling policies.

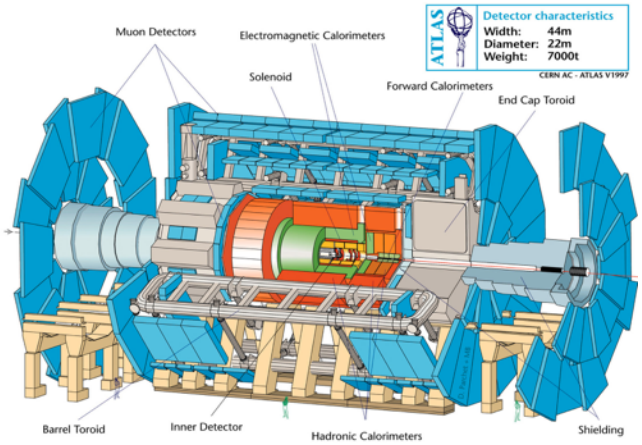
There is much more to Balsam, which is a complete service for managing complex workflows and optimized scheduling across multiple HPC resources.

Quickstart

- Install Balsam
  - Prerequisites
  - Environment
  - Get Balsam
  - Quick Tests
- The Balsam Database
  - Creating a new Balsam DB
  - Starting up the Balsam DB Server
  - Specifying which DB to use

HIGH ENERGY PHYSICS EXAMPLE

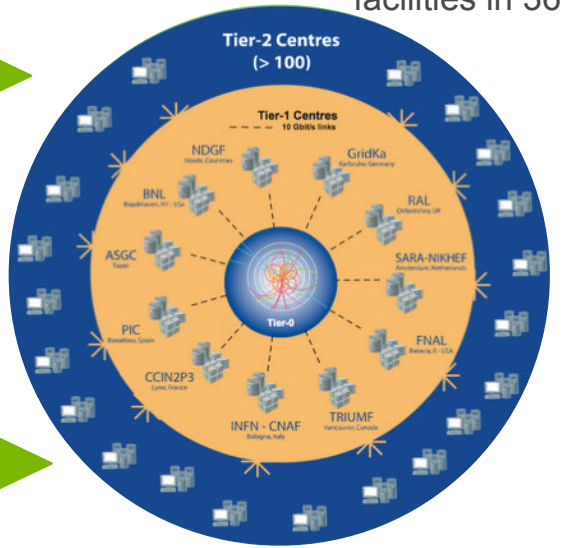
ATLAS detector at LHC



Experimental data (>25PB/year)

LHC Computing Grid (analysis)

>170 computing facilities in 36 countries



ALCF

Simulation data

Argonne NATIONAL LABORATORY

HIGH ENERGY PHYSICS EXAMPLE

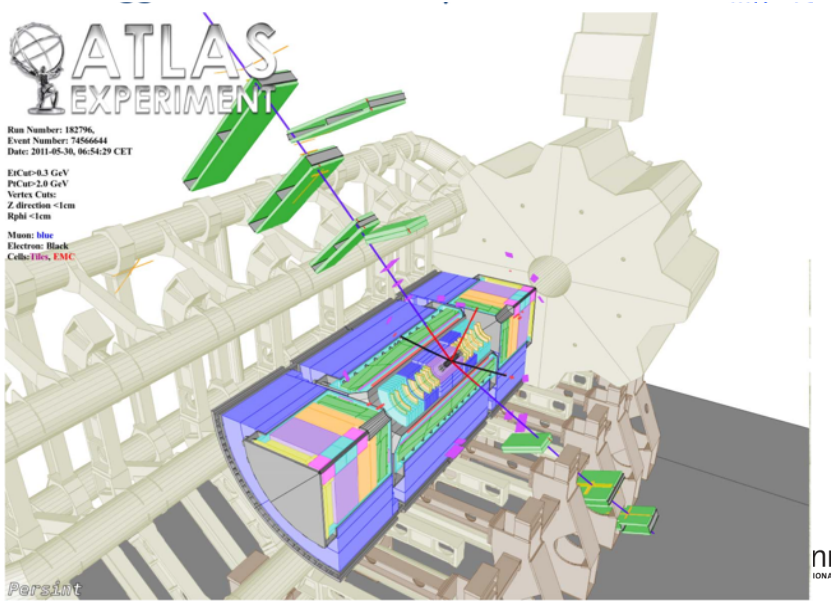
- The ATLAS experiment uses more than 2 billion compute hours per year
- This consists of integration, event generation, showering, simulation, reconstruction, and analysis
- This (very loosely coupled) workflow propagates step-by-step through the ATLAS production system based on requests from users.
- Simulation with Geant4 accounts for 60% of ATLAS's computing. We wanted to leverage Mira to offload some of this computing, and chose to target event generation (not simulation) as a first step. This task required coordination of multiple workflow-like steps:
  - fetching job descriptions from ATLAS (manual process)
  - serial phase-space grid integration on local cluster
  - transfer of data from local cluster to ALCF
  - large-scale event generation on Mira, based on incoming integration grids
  - transfer of data from ALCF to local cluster
  - post-processing of output data to exportable format
  - transfer of data from local cluster to WLCG

EXAMPLE: HEP EVENT GENERATION

Monte Carlo-based generation of particle collision events such as occur in the ATLAS detector at the Large Hadron Collider, using Alpgen.

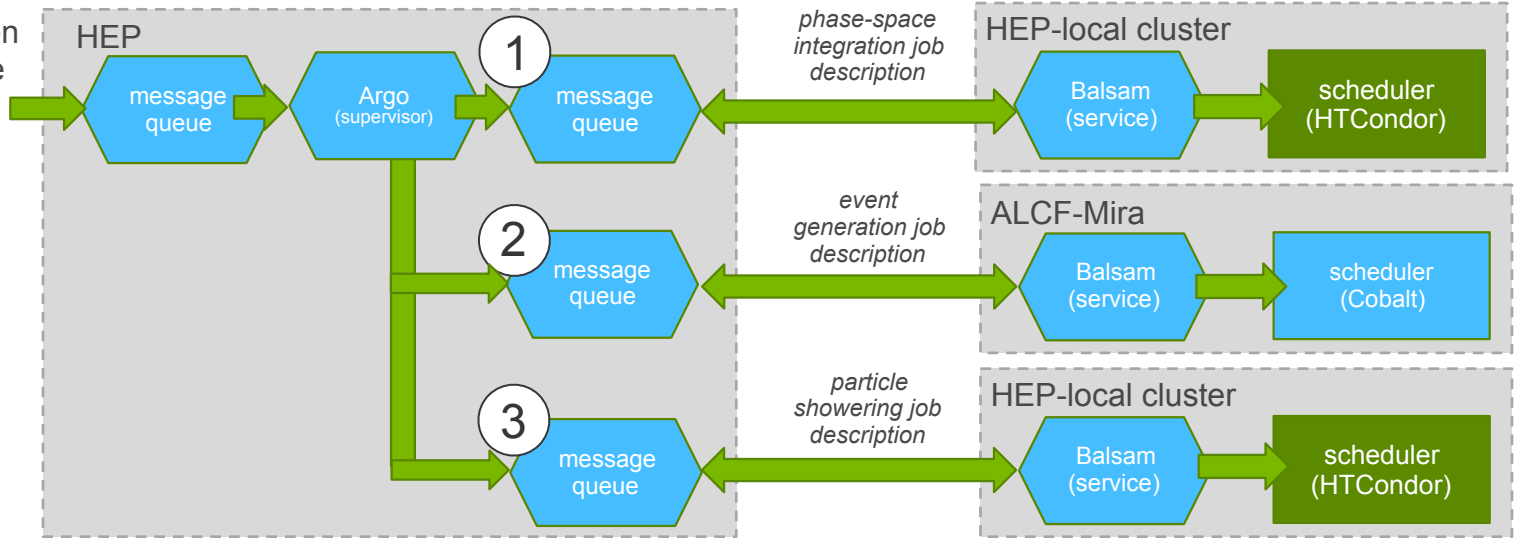
Consists of three stages:

1. Generation of phase-space integration grid
2. Generation of weighted events from integration grid
3. Statistical unweighting of events



MULTI-RESOURCE EVENT GENERATION WORKFLOW

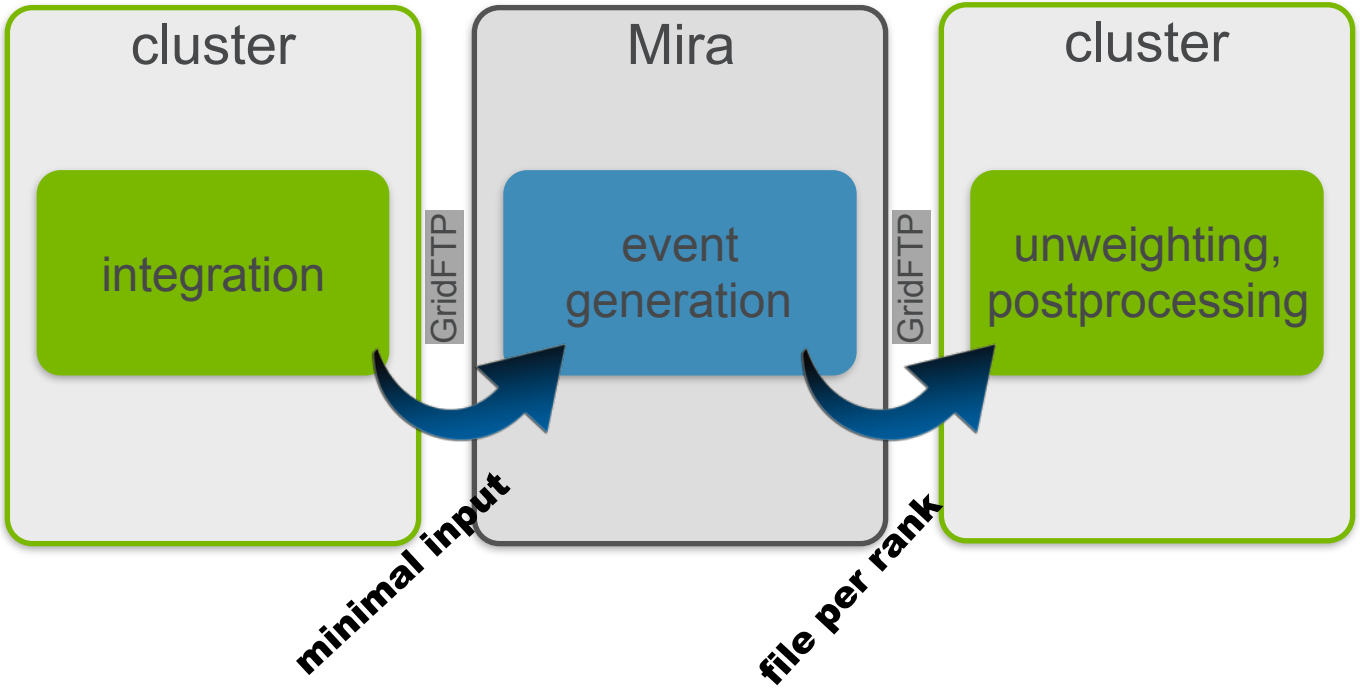
Job description messages are placed on the incoming message queue*



Final events are transferred manually to ATLAS; this could be automated in future

† et voila: Balsam is enabling workflows across ALCF systems

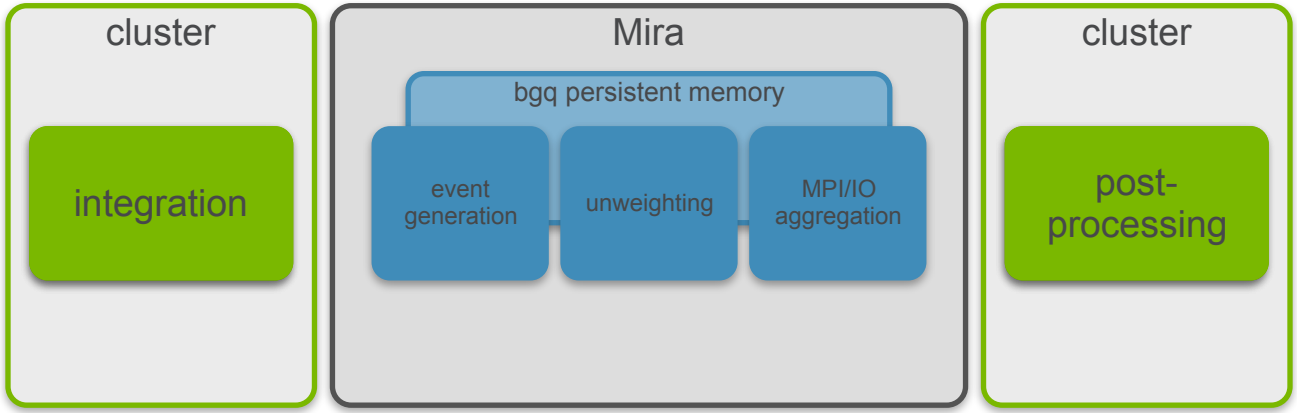
EVENT GENERATION ON MIRA



ALPGEN, a generator for hard multiparton processes in hadronic collisions, M.L. Mangano, M. Moretti, F. Piccinini, R. Pittau, A. Polosa, JHEP 0307:001,2003

EVENT GENERATION ON MIRA

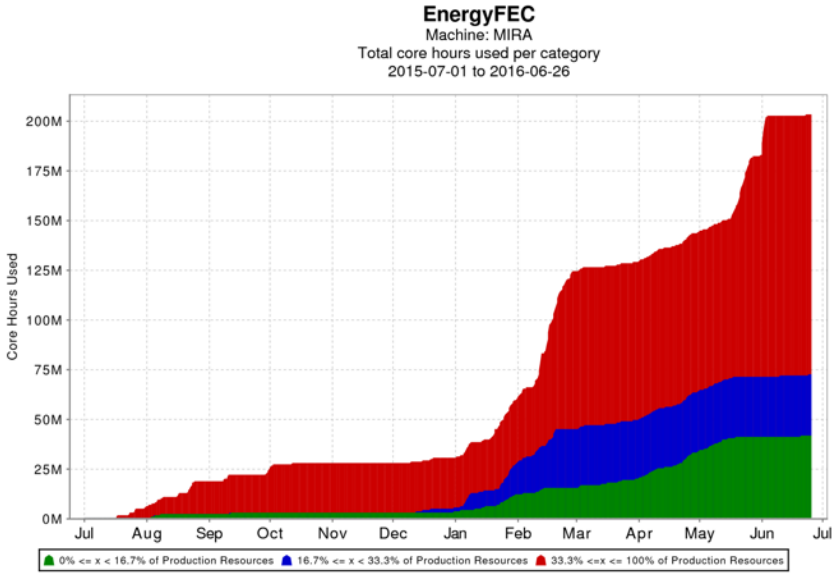
- Combine multiple application invocations in single script job
- Use persistent memory for exchanging data between invocations
- Aggregate data from persistent memory to filesystem

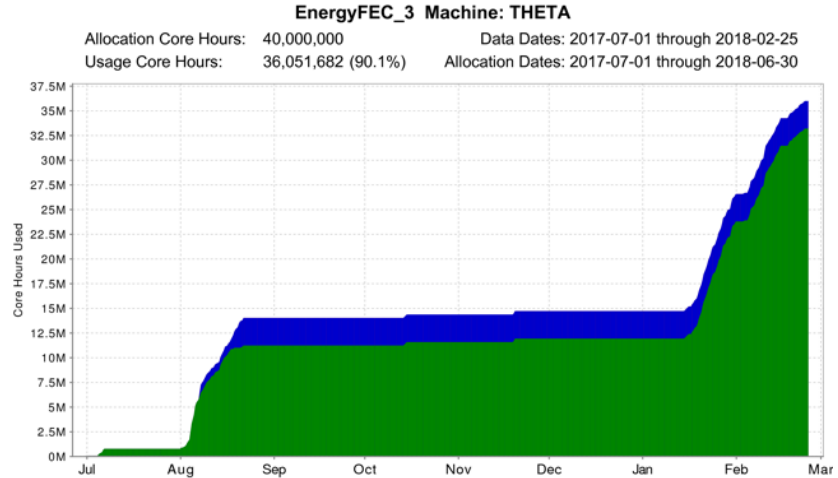


For details of container-based workflows for HEP work on Theta, see Taylor Childers' talk on Thursday

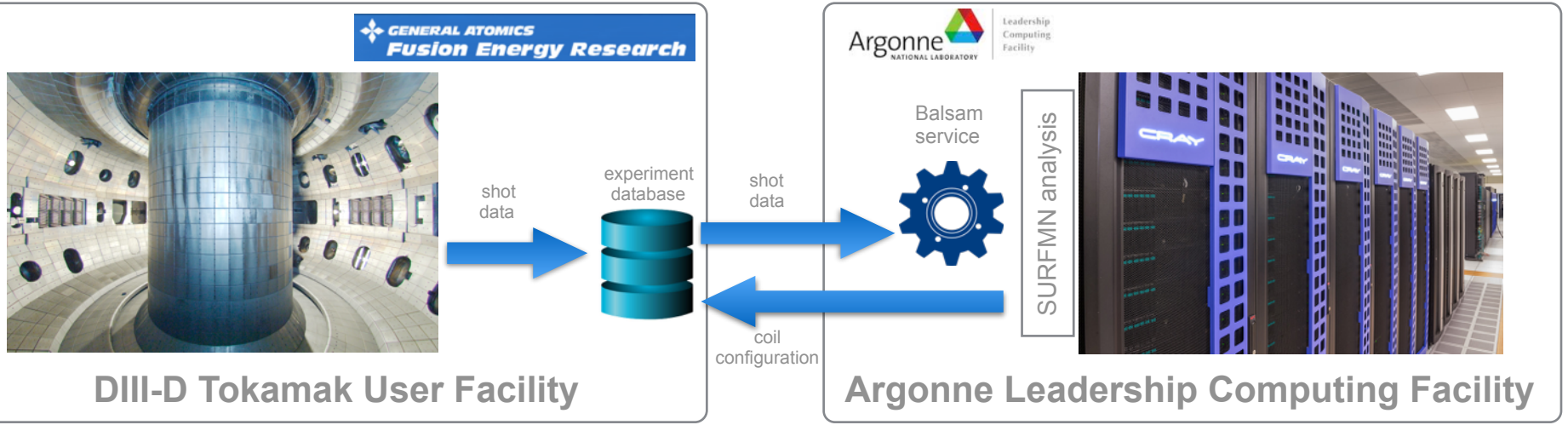
RESULTS

- **100M+ hours used for ATLAS event generation through Balsam**
- Unprecedented rates of event generation (largest event generation jobs on Mira requested >1T events)
- More complex/rare events than could be produced on the Grid
- Mira has become the primary site for ATLAS event generation
- 100 million compute hours have been offloaded from the WLCG to Mira, freeing this time to be used for other purposes
- Events that would have taken years to produce on the Grid were generated within a couple months, accelerating the simulation and analysis pipeline and therefore publications

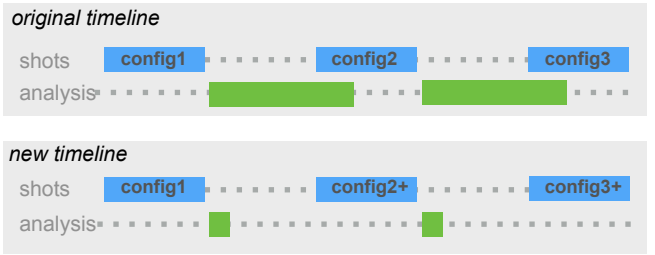




AUTOMATIC BETWEEN-SHOT ANALYSIS OF FUSION EXPERIMENT DATA



- Scientists configure experimental "shots" every 15 minutes
  - A shot is an attempt to magnetically confine high temperature plasma
  - The timing/current of magnetic coils are configured to control the plasma during a disruption to avoid damage to the containing vessel (applicable to DIII-D and future reactors)
  - Analyses indicate how to optimize coil configuration for confinement
- Each shot triggers an automatic, real-time analysis job at ALCF
- GA scientists integrate analysis results into configuration for next shot
- Analysis at ALCF enables more complex analyses (128^2 FFT vs 32^2) to be completed faster, improving the accuracy of results and allowing analyses to inform every shot instead of every other



Faster analysis time allows analysis results to be integrated into magnet configuration for subsequent shots. Higher resolution analyses improve configuration accuracy.

SUMMARY

- Workflows are inevitable in computational science
- Small-scale workflow management should be easy and is normally managed using scripting
- Large-scale workflow management can benefit from a workflow system
  - In the transition from small to large workflows, you might build your own workflow system
- The Data Science group at ALCF can help scale your workflows on our systems