Dark Energy Spectroscopic Instrument (DESI)

Making a 3D Map of the Universe at NERSC

Stephen Bailey
LBNL Physics

- What we do at NERSC
- Successes
- Challenges

NUG SIG
Experimental Facilities
2020-06-03

DARK ENERGY SPECTROSCOPIC INSTRUMENT

U.S. Department of Energy Office of Science

We're making a 3D Map of the Universe

Start with 2D map

- Take pictures of the night sky
- Measure locations of blobs –> x,y
- Combine 10s of millions of images from:
  - 5 ground-based telescopes
  - 3 satellites
  - 6 funding agencies on 4 continents
  - 4 data portals (some with single image http as the only data access option)
- Bring them all together at NERSC and do a joint fit across all datasets
- Share the results with everyone
  - http://legacysurvey.org
  - http://legacysurvey.org/viewer (a Spin service at NERSC)
  - /global/cfs/cdirs/cosmo/data/legacysurvey/dr8
  - ~900 TB of inputs + outputs

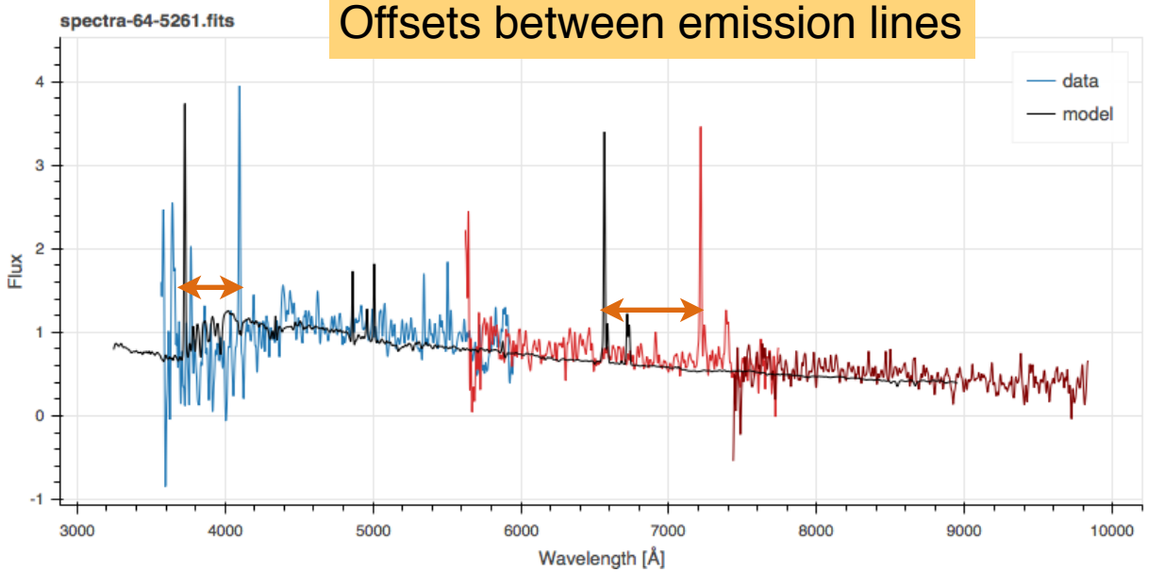
The past 5 years.
One big run left
to go (~20M hours)

DESI: getting the 3rd dimension

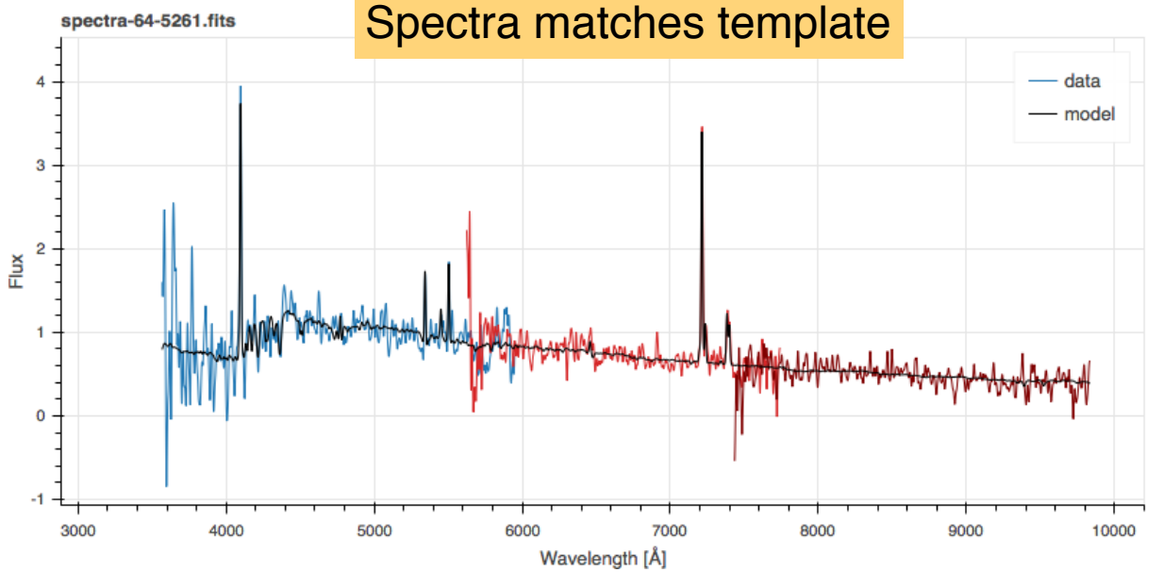
(and statistically the 4th, 5th, and 6th dimensions, not covered here)

- Select a subset of objects
- Measure their spectra (photons vs. wavelength)
- Fit scale factor in wavelength compared to reference templates –> “redshift”

Template at redshift=0
Offsets between emission lines



Template at redshift=0.1
Spectra matches template



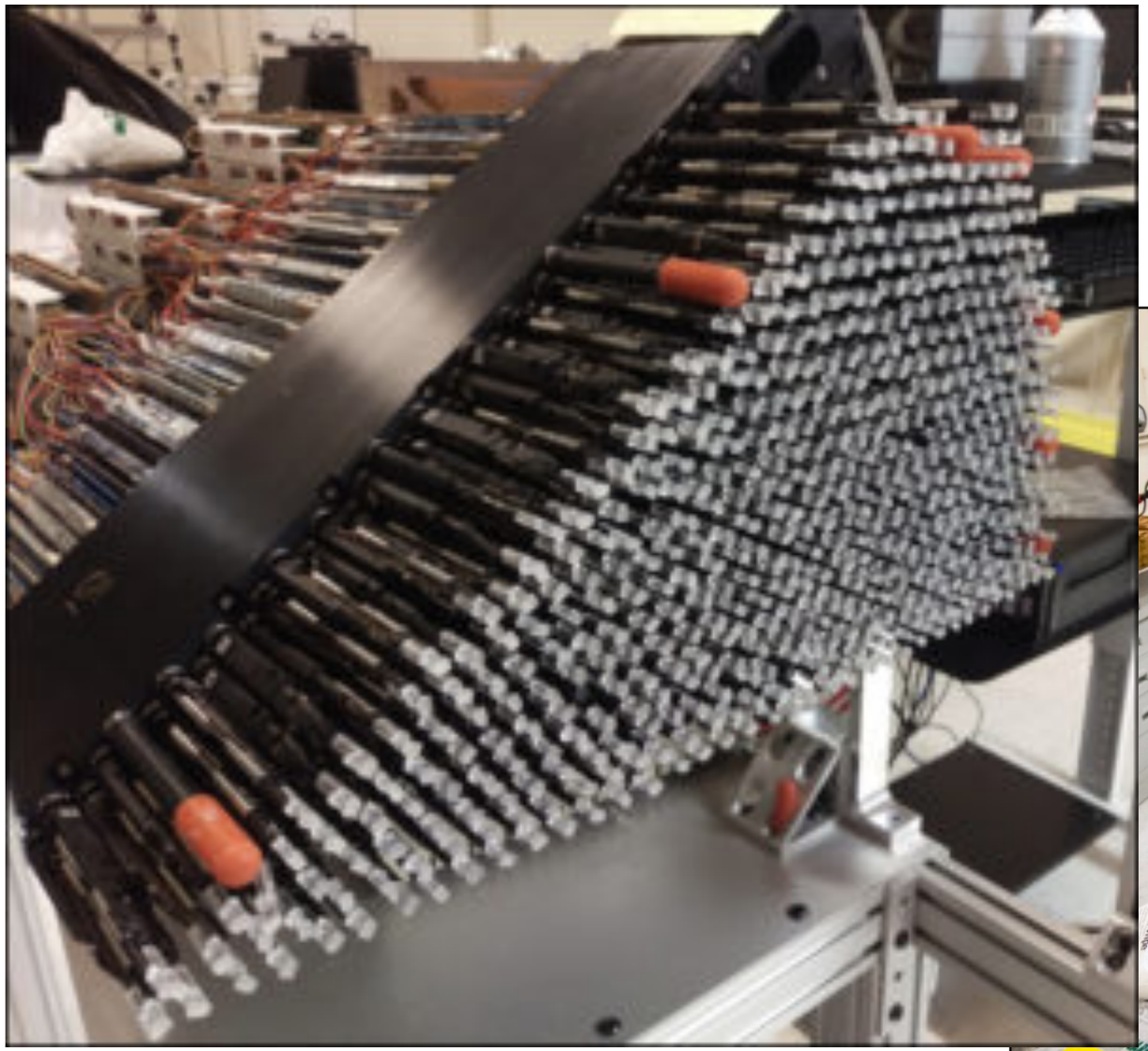
The next 5 years
(once we tackle COVID19 and can reopen)

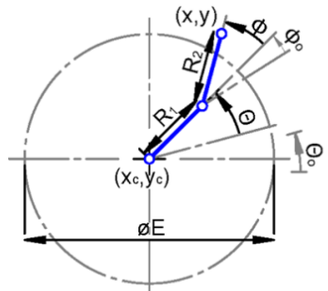
DESI Robotic Positioners

5000 robots + 10 spectrographs + 250 km of fiber optic cable

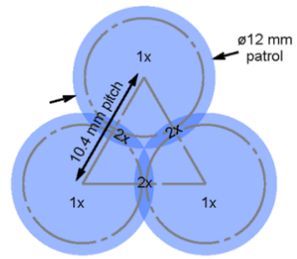
= measure 5000 redshifts every ~15 minutes

–> data 30x spectro CCD images, ~700 video frames (~3Mpix each) “guiding” the telescope

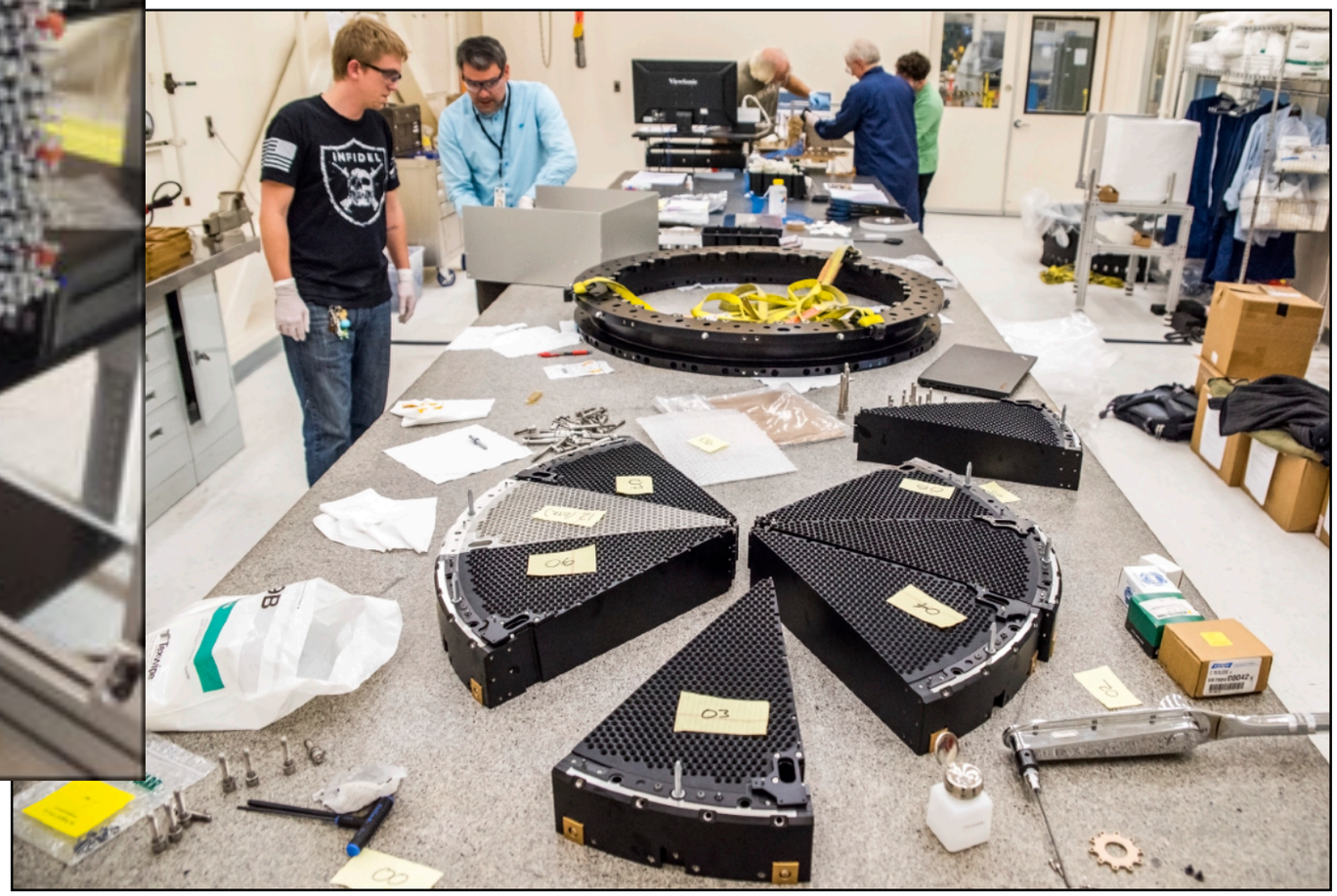




KINEMATICS / COORDINATES



EXACT COVERAGE
R1 = R2 = 3 mm



DESI Data Processing Basics

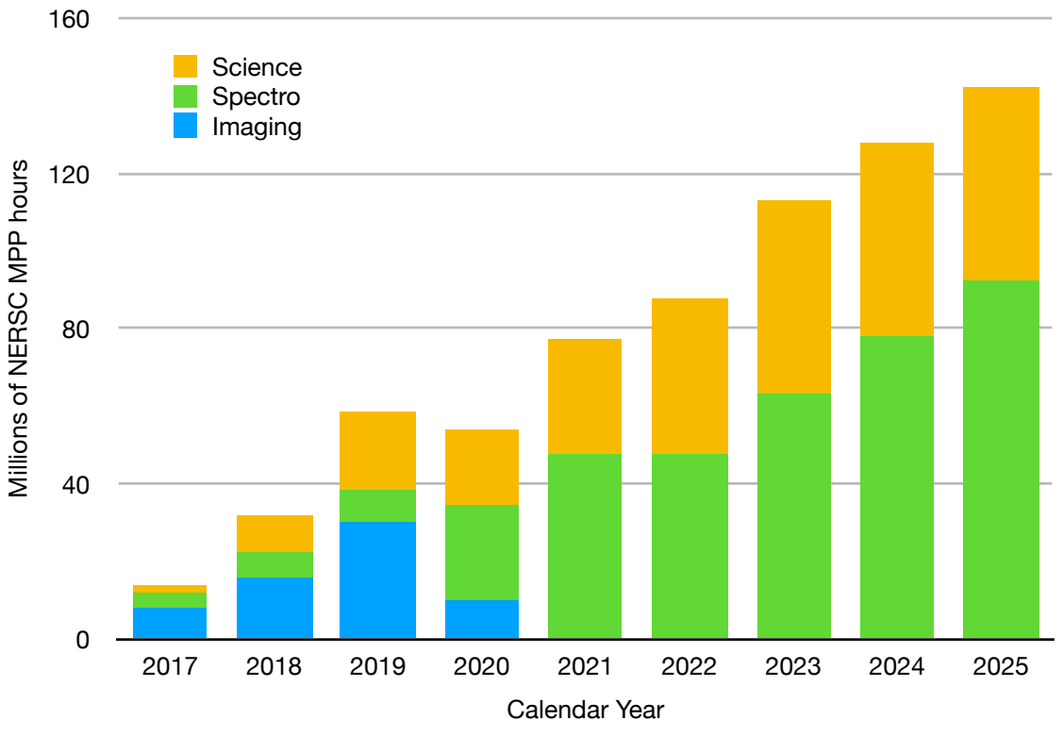
Nightly: long running processes on workflow node

- Every 10 minutes rsync new data from Kitt Peak, AZ –> NERSC
- New data –> launch jobs to realtime queue (10 nodes)
- Results ready by breakfast for analysis during day,
  to inform the following night's observing plan
- ~60 GB/night input —> ~375 GB/night output
  - spectra + redshifts of 50k-100k galaxies, quasars, stars
- Repeat for 5 years to build 3D map of ~50M objects
- Mostly python, but designed for both laptops and HPC from the beginning

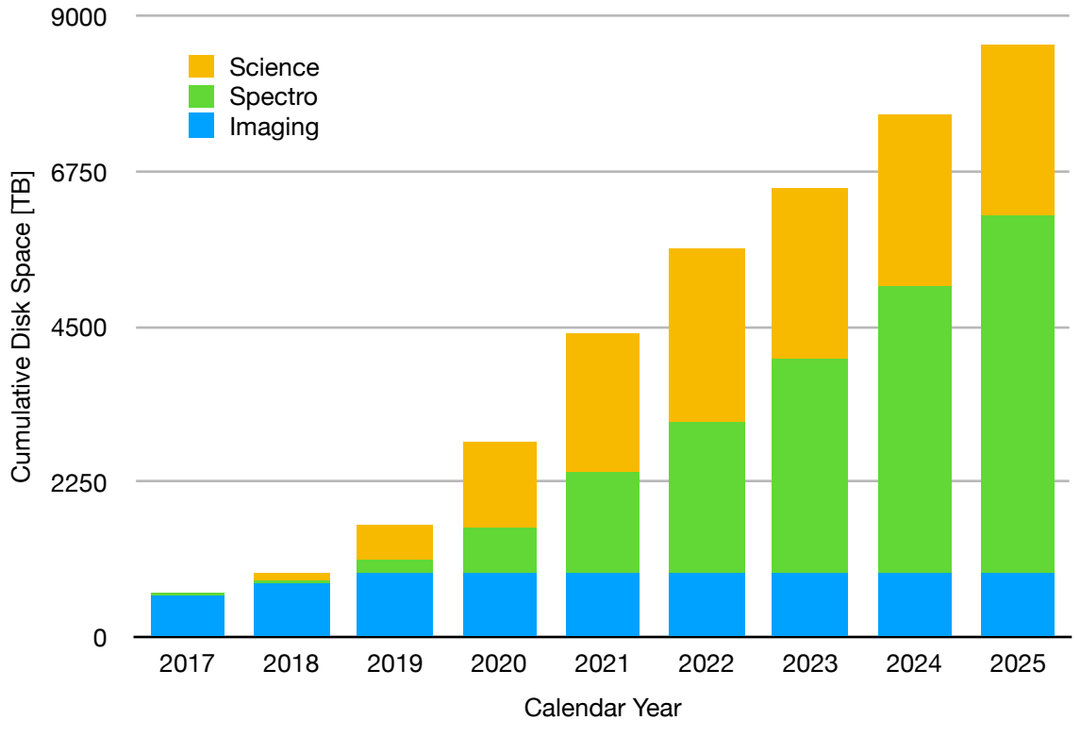
Monthly / yearly

- Reprocessing runs with latest tagged code, starting from raw data
- Same code as nightly processing, but very different scaling needs
- **This is the primary reason for DESI @ NERSC**
  - Also: one stop shopping for daily processing, big reruns, final science analyses

CPU & Disk Projections



50M - 150M
MPP hours/year



Growing to ~9 PB
on disk

DESI uses the full NERSC ecosystem

Computing

- Realtime for nightly, big iron for reprocessing

I/O

- CFS, scratch, HPSS
- Globus, rsync, portal.nersc.gov + spin container with nginx

Workflow

- Workflow nodes, databases

Analysis

- Jupyter
- Interactive & debug queues

QA monitoring

- Spin, cron jobs

We rely upon much more than just raw FLOPS and I/O bandwidth. Corollary: outages of any of these services negatively impact us

Success: testing @ NERSC

Open source on github + travis continuous integration testing

- Branches, pull requests, code review
- Travis CI automatically runs unit tests for all pull requests

Nightly cronjob script at NERSC

- “git pull” for all our repos
- Rerun unit tests at NERSC
  - does it work *as installed at NERSC*, not just on Travis servers?
- Run integration tests combining repos
  - uses larger datasets and longer runtime than viable for Travis
  - nightly email with success or problems
- Not fancy, but easy to maintain and very useful to catch problems early

Quarterly for software releases

- larger integration test: jupyter + interactive queue –> “reference runs”

[WIP] Success: resilient workflow

Hard: make jobs that don't fail

- Hard because often the failures are beyond your control
  (I/O hangs, bad nodes, srun failures, DB connection failures)
- Reasons vary with time, but 1-2% failures is common

Better: make it easy & efficient to recover

- Efficient for human to know what went wrong and want to do
  - or even fully automated retries
- Same launch commands and jobs adapt to redoing only what is needed
  - it shouldn't require a lot of handwork to re-submit the 1% of input files that failed

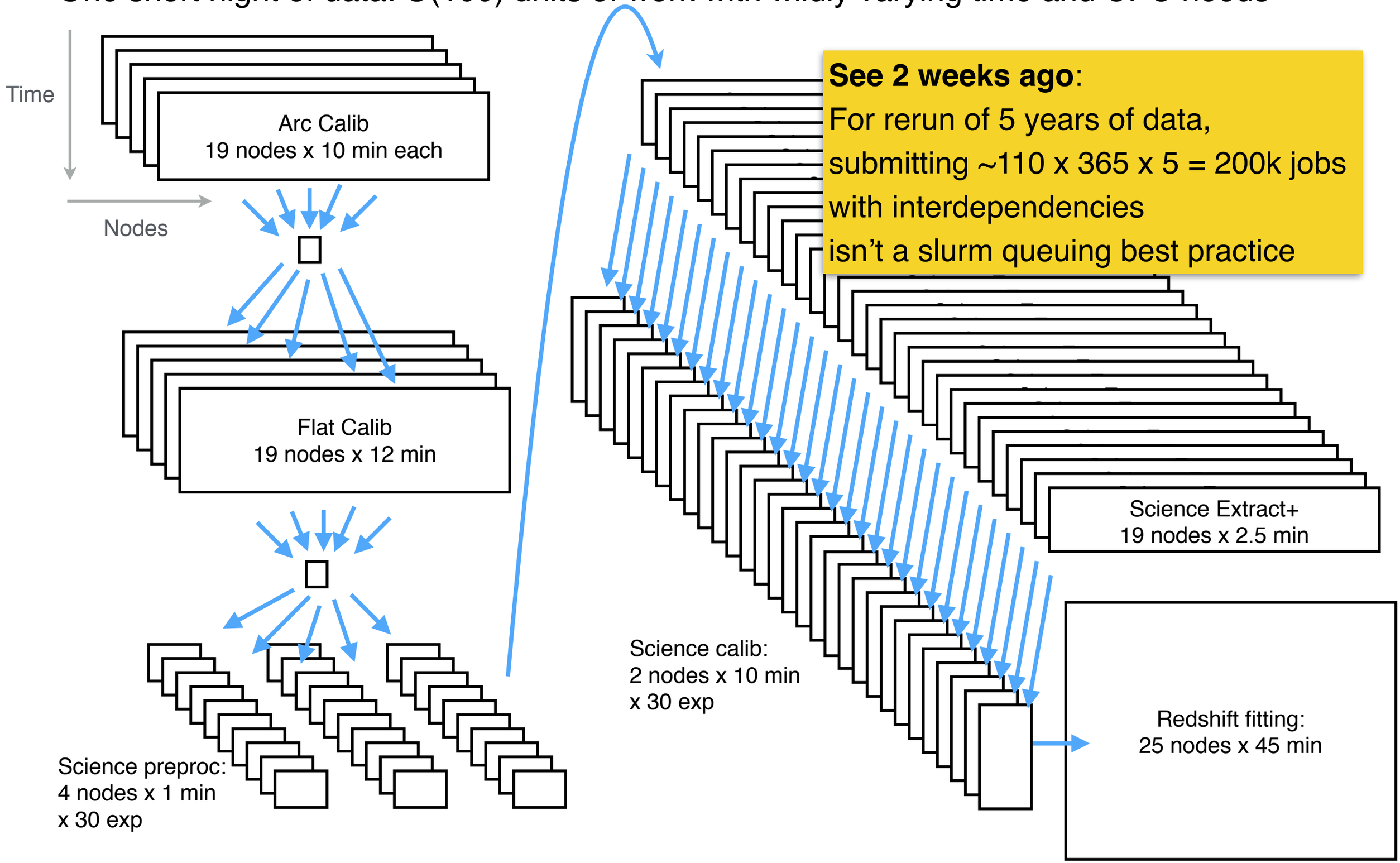
Ideas aren't new

- TCP packet retries
- MapReduce / Hadoop
- Checkpoint restart on steroids

Tools like taskfarmer and GNU parallel are only part of the solution; also need tools for how to efficiently recover when things go wrong (because in big runs, *something* going wrong is the norm, not the exception).

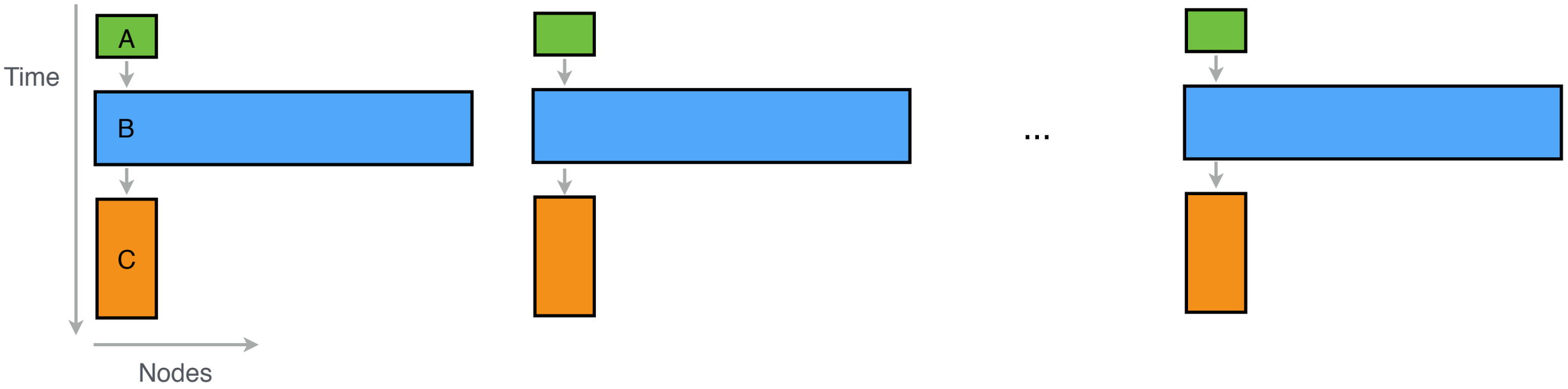
Challenge: queuing complex dependencies

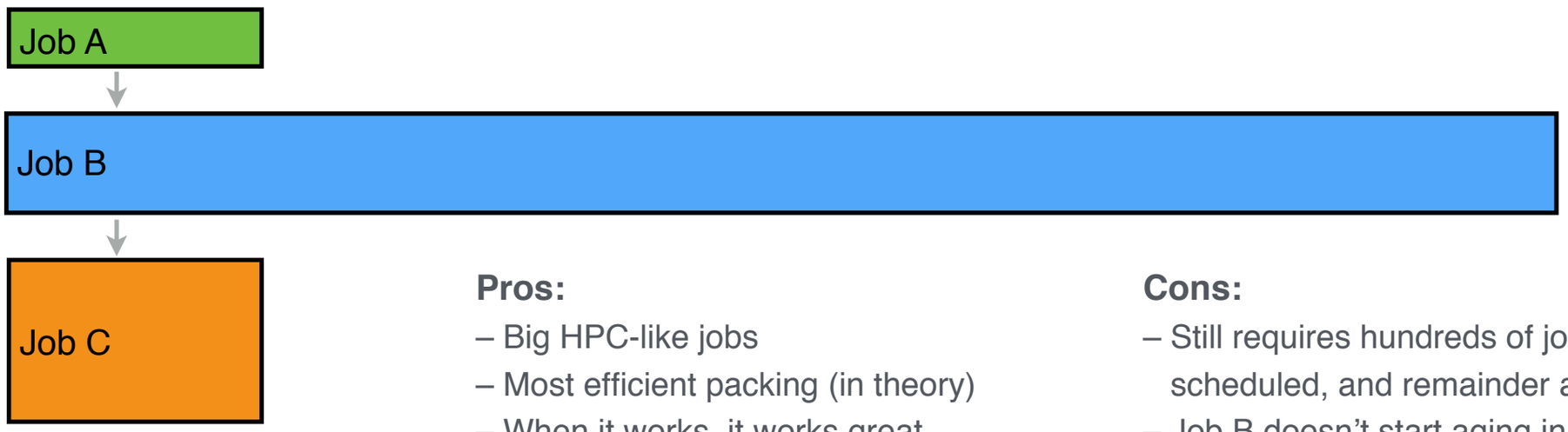
One short night of data: O(100) units of work with wildly varying time and CPU needs



See 2 weeks ago:
For rerun of 5 years of data,
submitting ~110 x 365 x 5 = 200k jobs
with interdependencies
isn't a slurm queuing best practice

Attempt 1: Bundle each step x ~1 week of data





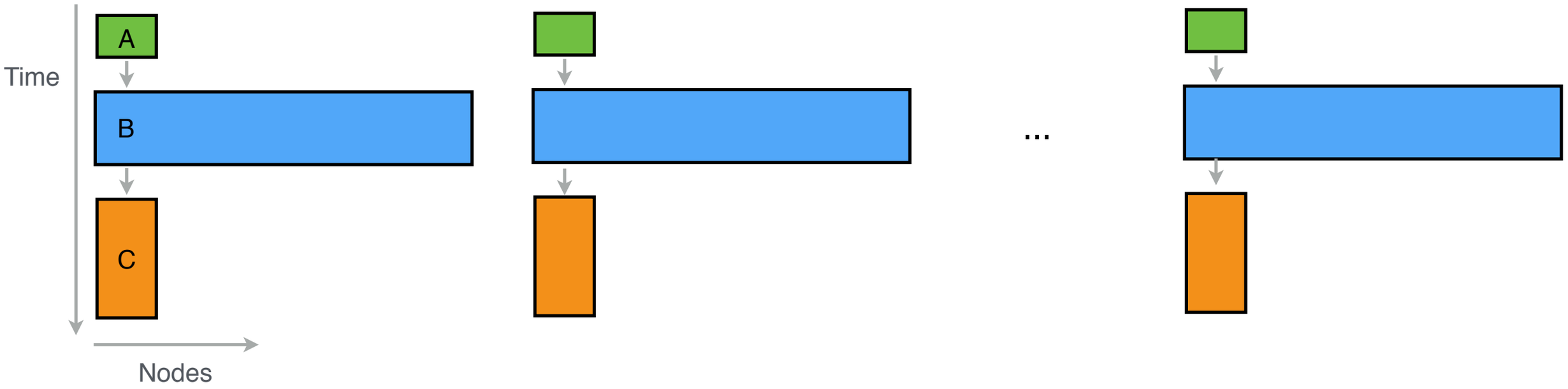
Pros:

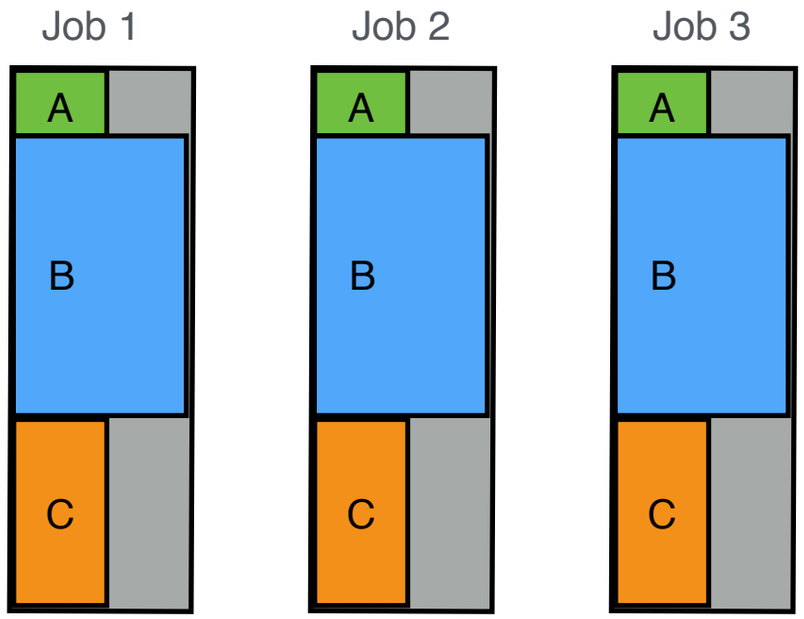
– Big HPC-like jobs
– Most efficient packing (in theory)
– When it works, it works great

Cons:

– Still requires hundreds of jobs, only 2 of which are priority scheduled, and remainder are bigger than ideal for backfill
– Job B doesn't start aging in queue until A finishes
– Couples otherwise completely independent data processing (one rank can take down all ranks)
– I/O hang of one rank wastes time of thousands of others waiting for job to timeout
– Recovery takes a long time to get through queue
– Startup hammering disk, DB

Attempt 2: 1 exposure = 1 job, accept inefficiencies





Pros:

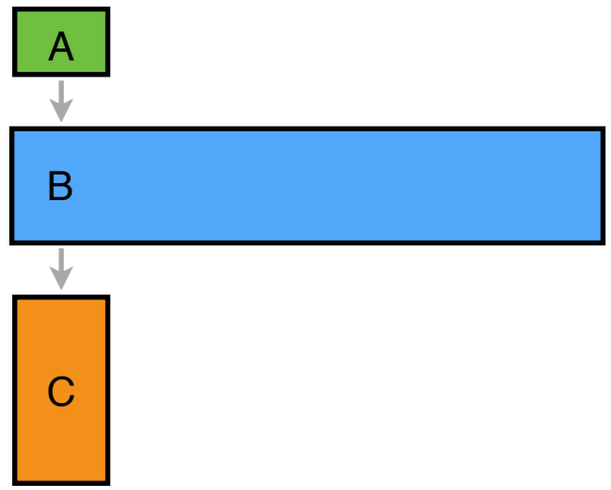
– Faster end-to-end for subset of data
– Decouples independent data
– Matches realtime job packing method
– I find this structure easier to work with, but that is somewhat a matter of taste

Cons:

– Many more jobs (~60k), won't scale to 5 years of data without job launch throttling (like Fireworks launcher)
– Wasted cores within a job during certain steps
– We've started thinking of ways that we would like to couple different exposures for algorithmic reasons

Dreaming big: what would help

"This then that" scheduling



- I know I want to do A then B then C, and I know their sizes ahead of time
- Scheduler Tetris with non-rectangles

"Big bag of equivalent jobs" scheduling

- If I have 100k identically sized jobs to run, slurm shouldn't have to loop over them individually to pick the best one if it only has one slot to fill
  - cases where total work is >> 1 job + taskfarmer / GNUparallel

Dynamically sized jobs

- Job with steps A+B+C could release a lot of nodes after finishing B

I know these are hard to implement

- Do other Experimental Facilities projects face workflows where different steps have very different parallelism needs, making them hard to pack into jobs?

Challenge 2: data shuffling within NERSC

"Compute on $SCRATCH, share on CFS" model doesn't work well for us

- Example reasons
  - Until recently, lack of Globus for collaboration accounts
  - scratch quotas << campaign data volumes
  - scratch quota per user, CFS quota per repo
  - campaign durations >> scratch purge cycles
  - Scratch metadata performance fluctuations outweigh theoretical bandwidth benefits for some of our workflows
- All of these have workarounds, but in practice this has led to lots of human effort and lots of human errors

Feels like an area that could use better / shared tools

- Seamlessly flowing data between HPSS, scratch, CFS, burst buffer, /tmp
- Do other projects share this pain point? Or have transferable solutions?

Summary

DESI is making a 3D map of the universe
using NERSC as the primary computing center

- Yearly reprocessing drives the need for HPC
- Also benefit from one stop shopping for data processing + science analyses

Successes

- Testing @ NERSC
- [WIP] resilient workflows

Challenges

- Queueing N>>1 algorithmic steps with very different parallelism needs
- Coordinating dataflow within NERSC (HPSS, scratch, CVS, BB, /tmp)

Thank you

DARK ENERGY SPECTROSCOPIC INSTRUMENT

U.S. Department of Energy Office of Science