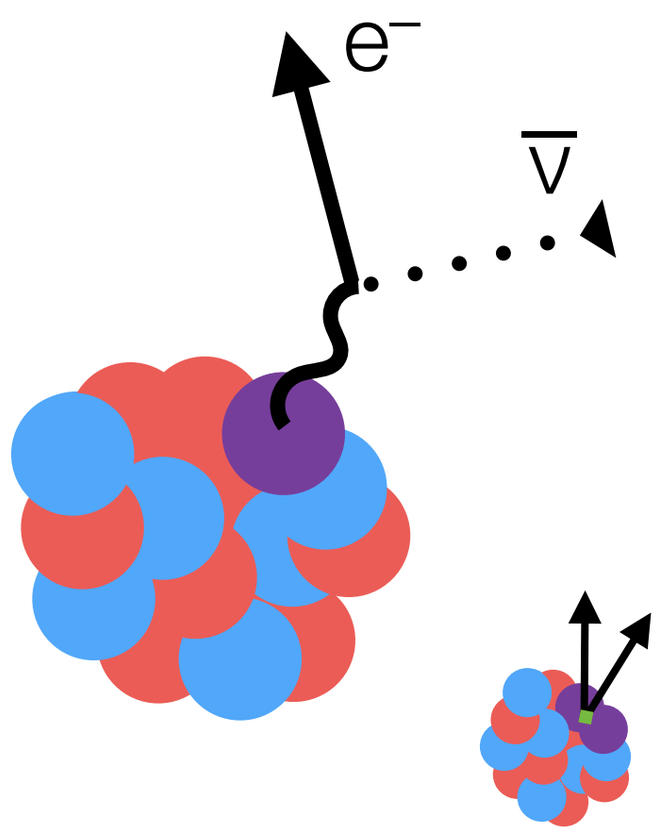


Job Management and Task Bundling

Evan Berkowitz
Institut für Kernphysik
Institute for Advanced Simulation
Forschungszentrum Jülich

22 June 2017
LATTICE 2017
Grenada, Spain

JÜLICH FORSCHUNGSZENTRUM

CalLat


Berkeley, LBL Ken McElvain, André Walker-Loud

FZJ EB

Oak Ridge Gustav Jansen

`METAQ`

arXiv:1702.06122

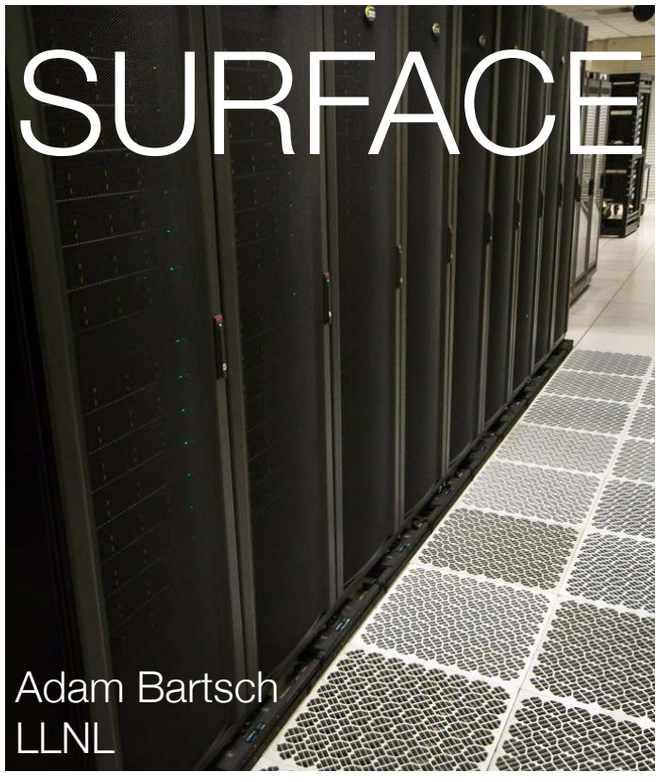
https://github.com/evanberkowitz/metaq

`mpi_jm`

[in preparation]


TITAN
OLCF


Edison
NERSC


SURFACE
Adam Bartsch
LLNL

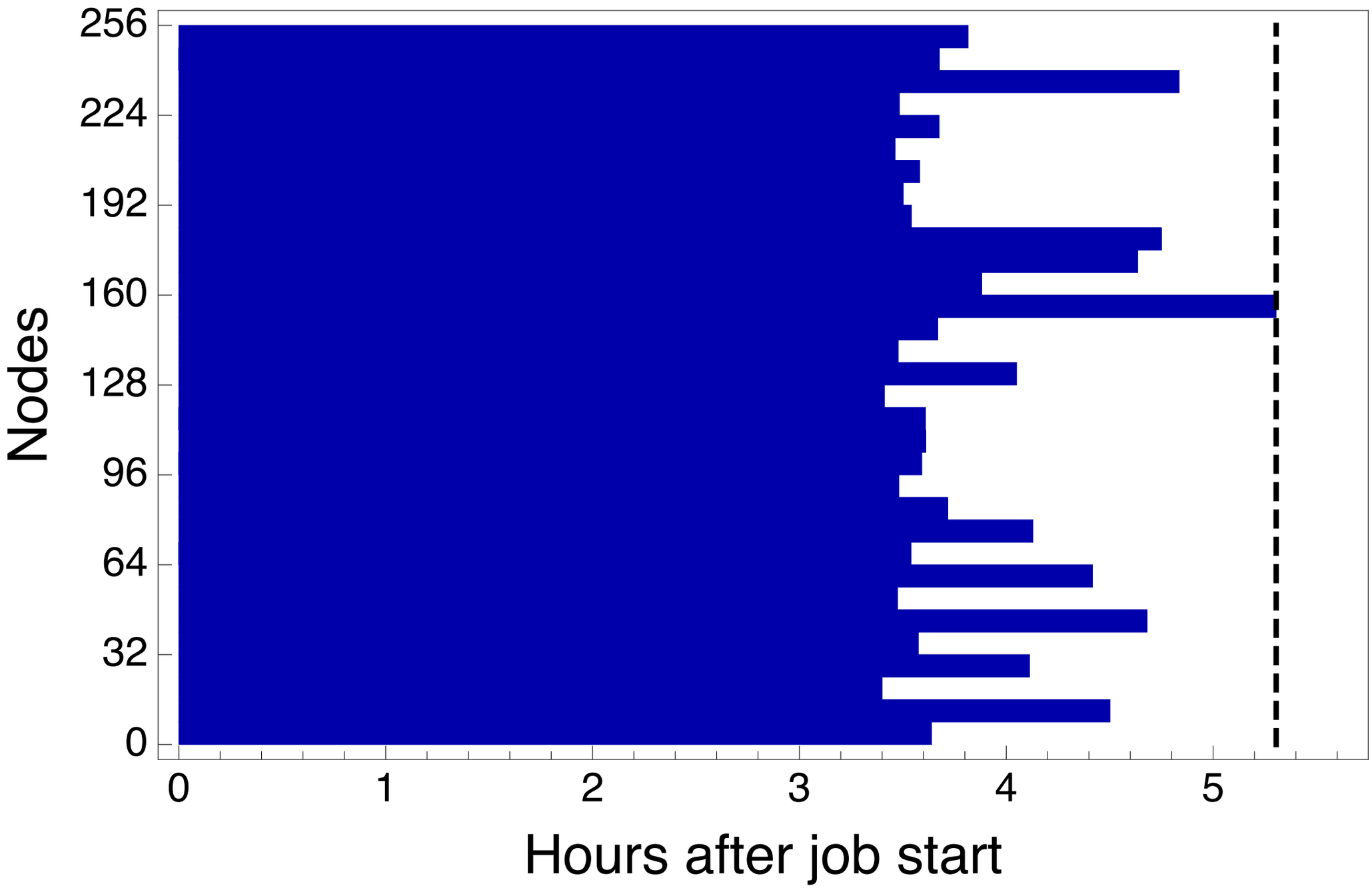
Leadership Computing for QCD

- NERSC: 40% discount if jobs are larger than 683 nodes
- Titan: increased priority for jobs larger than 3750 nodes

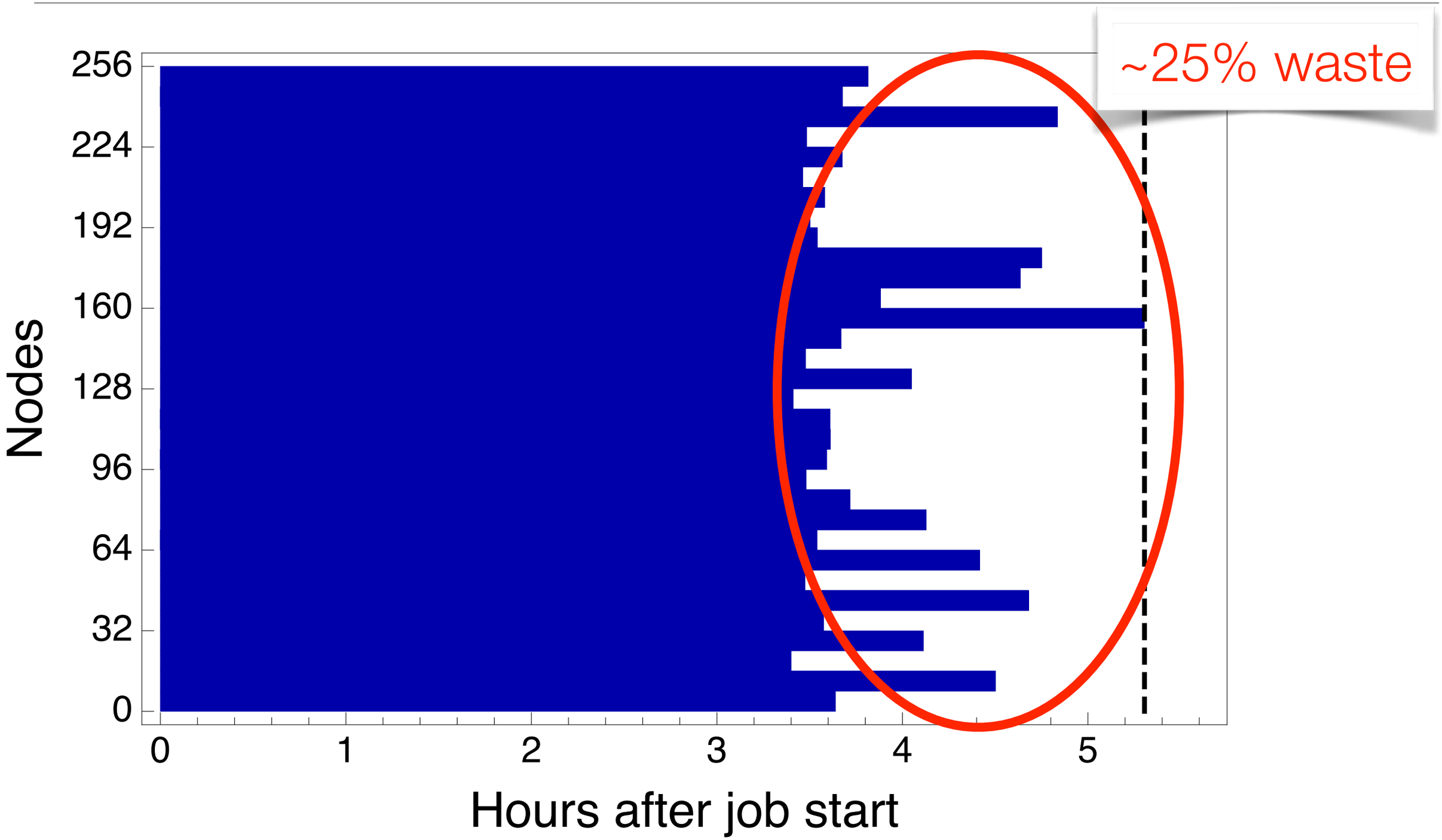
g_A (see Chia Cheng Chang's plenary)

- 1000s of propagators: 32 nodes each — GPUs
- 1000s of FH propagators: 32 nodes each — GPUs
- 1000s of contractions: 8 nodes each — CPUs

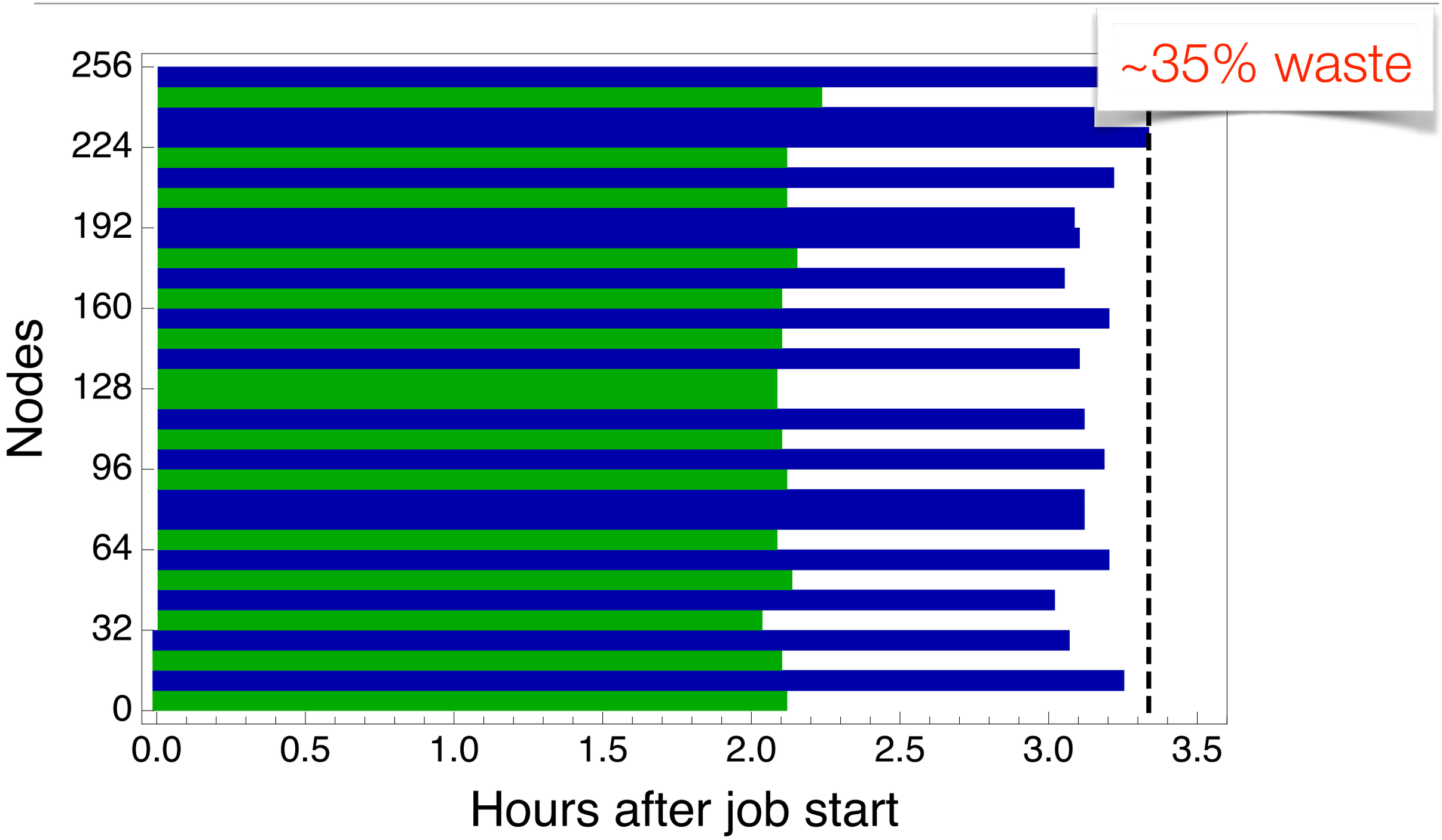
Naïve bundling



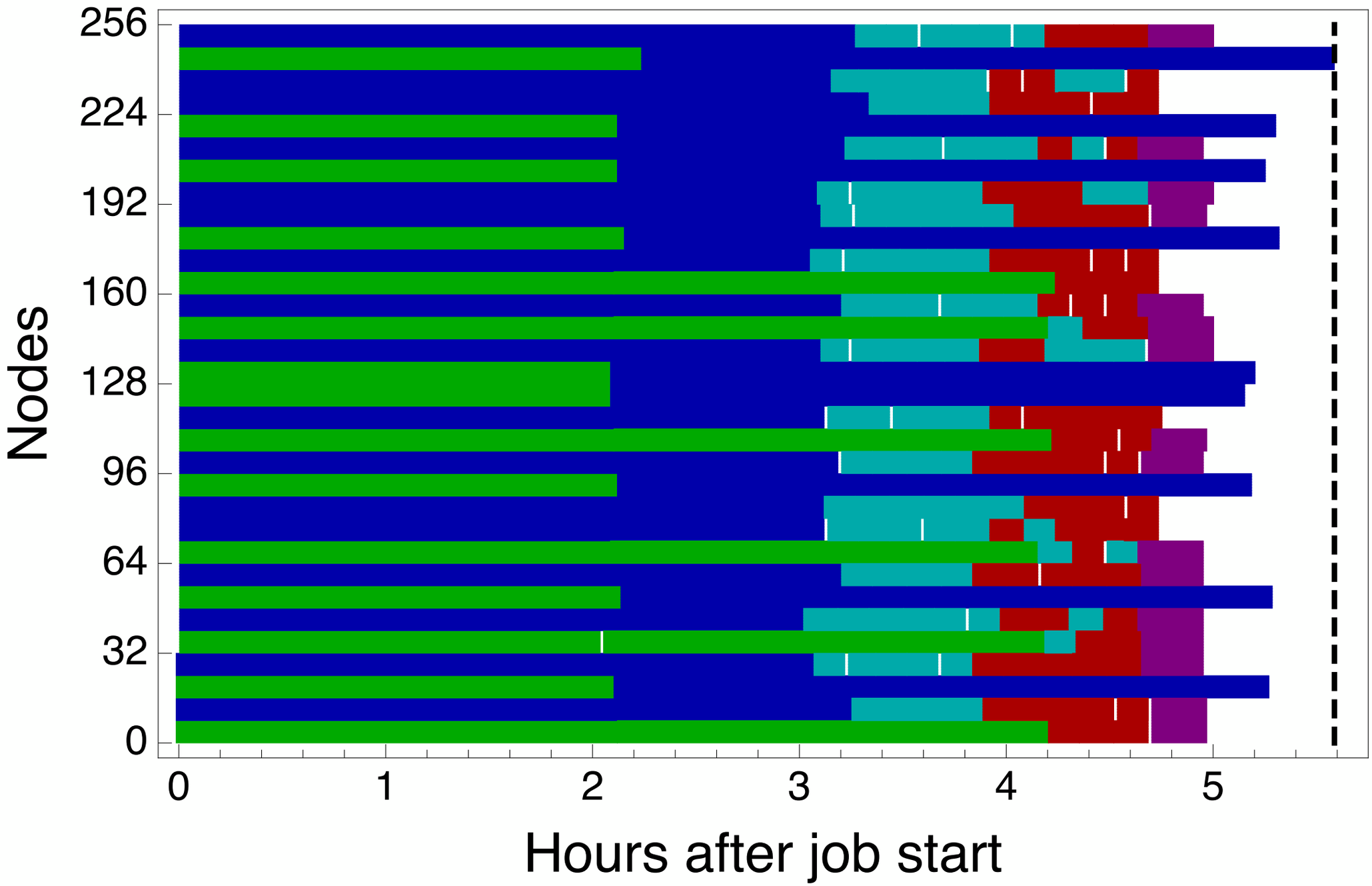
Naïve bundling



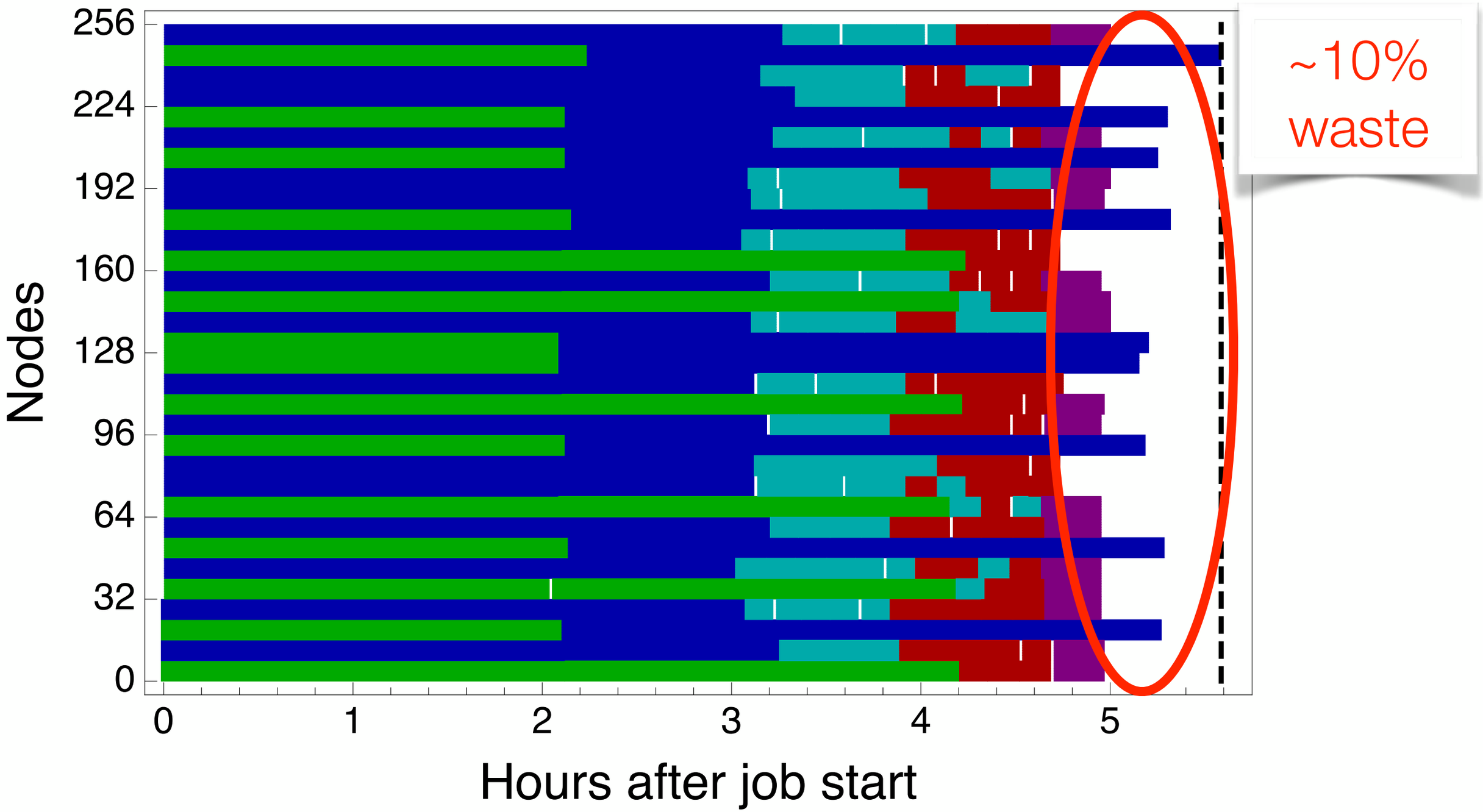
Naïve bundling: heterogeneous mix



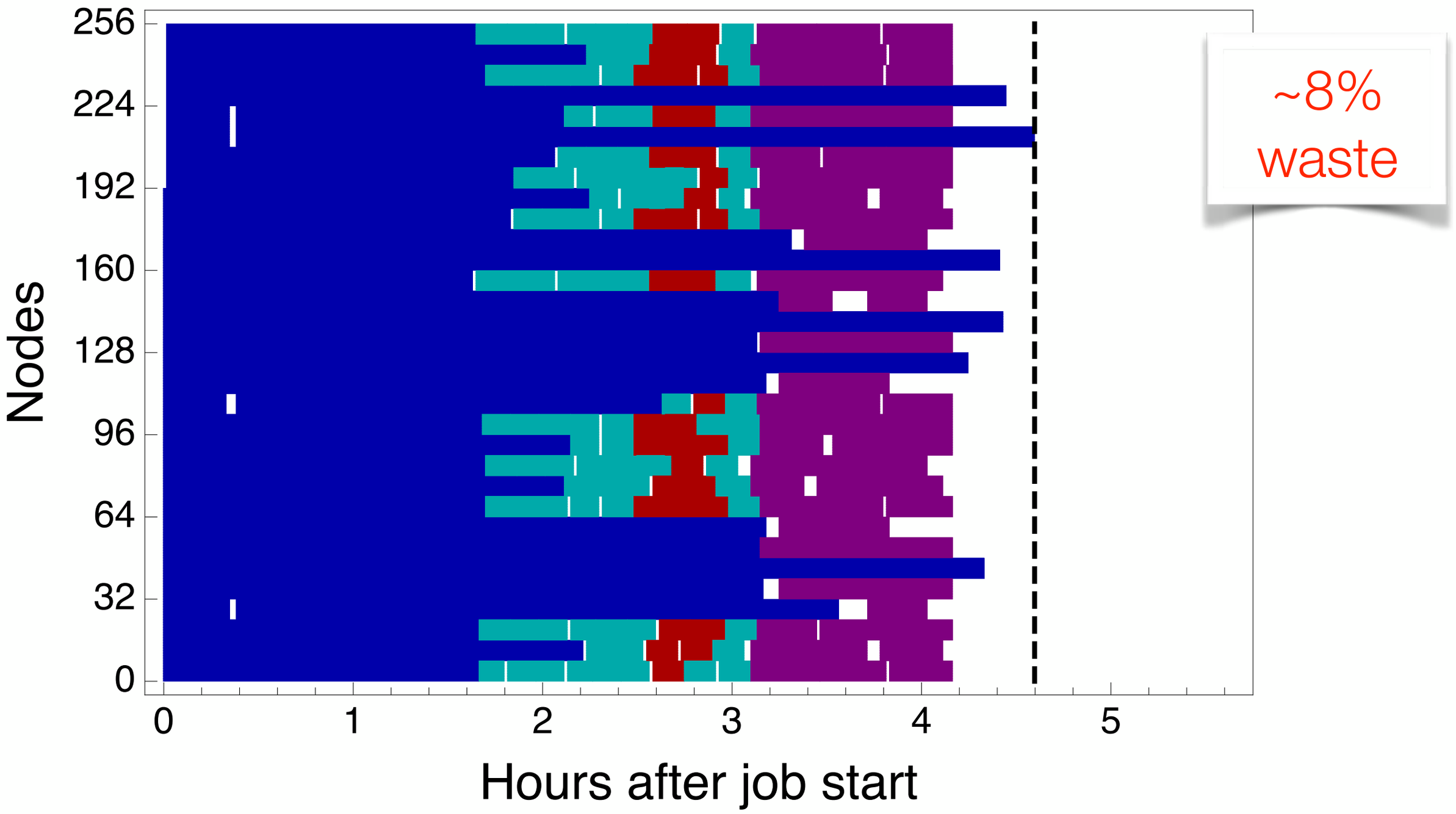
Back Filling



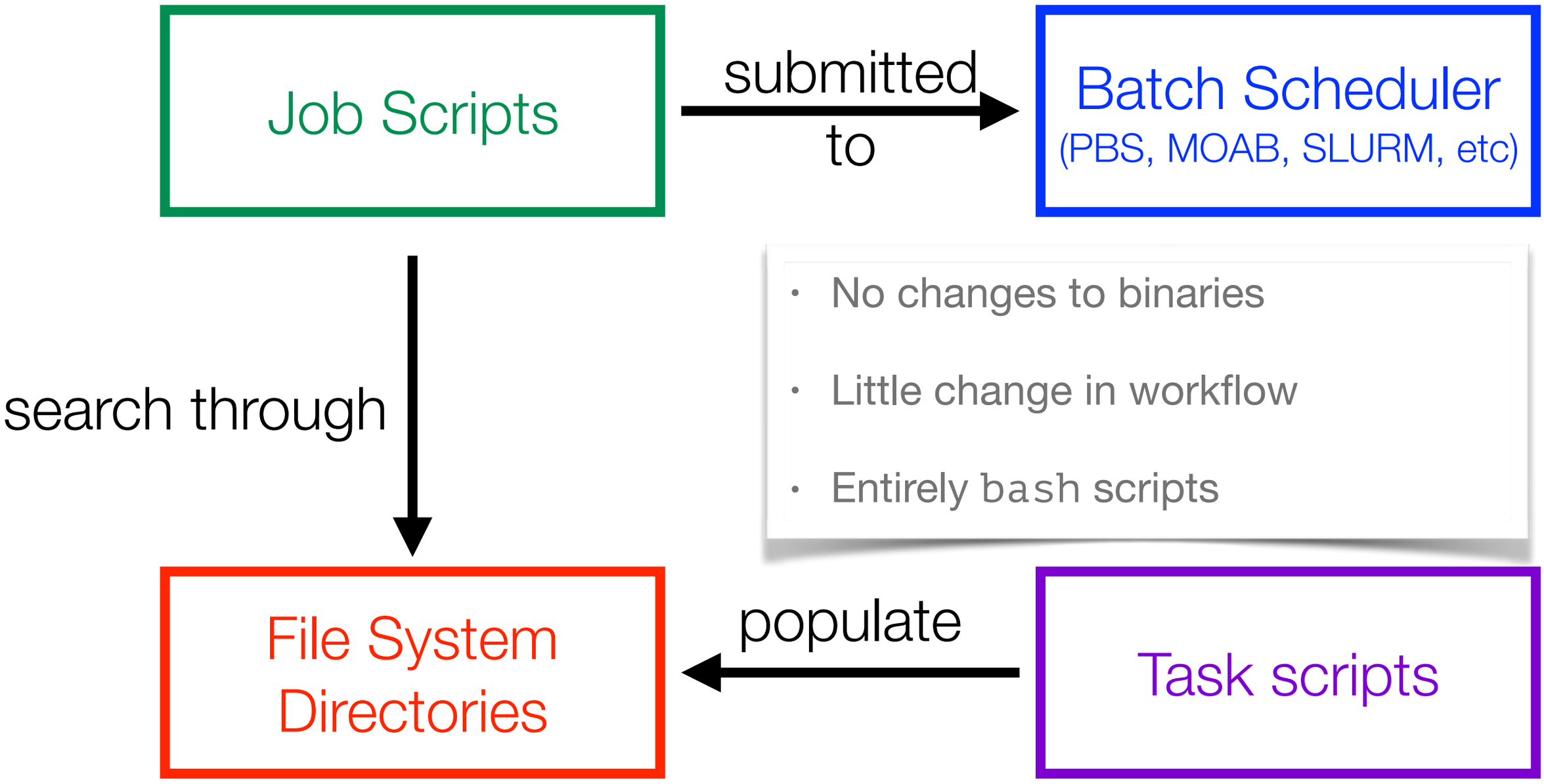
Back Filling



Back Filling



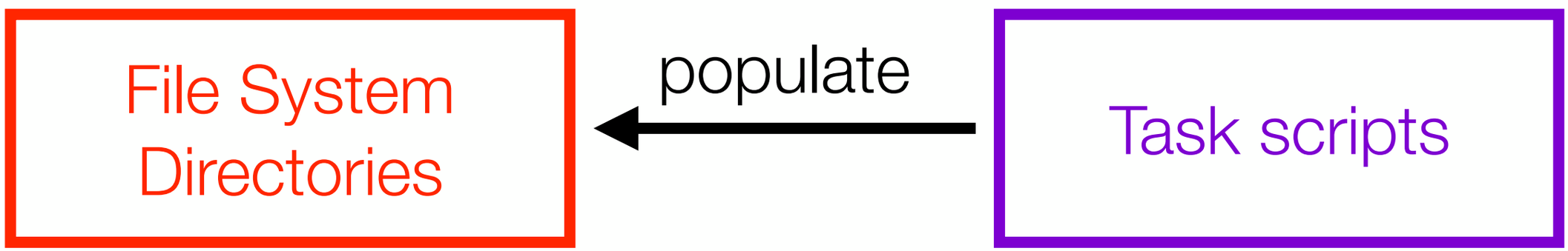
METAQ


Job Scripts
submitted to
Batch Scheduler
(PBS, MOAB, SLURM, etc)
search through
File System
Directories
populate
Task scripts
No changes to binaries
Little change in workflow
Entirely bash scripts

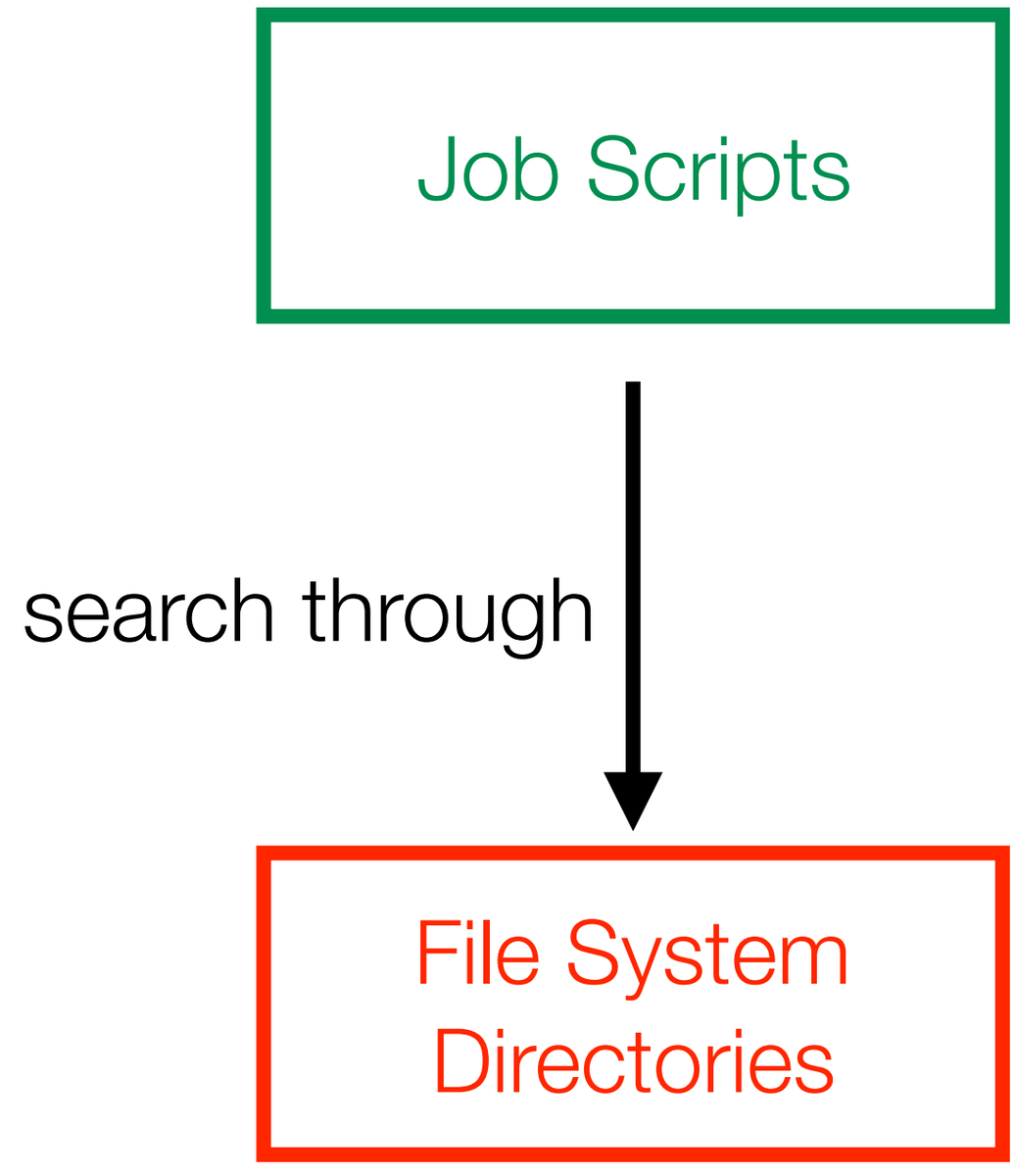
METAQ Task Scripts

```
#!/bin/bash
#METAQ NODES 2
#METAQ MIN_WC_TIME 5:00
#METAQ PROJECT metaq.example.1
#METAQ LOG /path/to/output.log

# do setup:
load modules
set environment variables
[etc]

echo "working hard..."
[call srun, aprun, etc. using 2 nodes for ~5 minutes in this example]
echo "finished!"
```

- Describe computational steps
- Look like familiar batch-scheduler scripts
- Any collaborator may contribute
- Tasks can be completely unrelated



METAQ Job Scripts



- Batch scheduler flags
- Describe allocation
- Any collaborator may submit
- Build batch priority independent of work
- Prioritize directories to consider
- Extremely uniform, simple to write

METAQ Drawbacks

- Blindly trust the user
- GPU / CPU overcommitting dependent on administrative policy
- No `msub`, `qsub`, `sbatch` equivalent
- Walking over task scripts can be slow and disk intensive
- Runs on head nodes, can stress common resources

METAQ Drawbacks

- Blindly trust the user
- GPU / CPU overcommitting dependent on administrative policy
- No `msub`, `qsub`, `sbatch` equivalent
- Walking over task scripts can be slow and disk intensive
- Runs on head nodes, can stress common resources

Warning: On Titan, each batch job is limited to 200 simultaneous processes. Attempting to open more simultaneous processes than the limit will result in `No space left on device` errors.

https://www.olcf.ornl.gov/support/system-user-guides/titan-user-guide/

`mpi_jm`

- C++ based
- Relies on `MPI_comm_spawn`
- Job management is done on compute nodes
- Specify GPUs, CPUs, memory
- Python front-end with pre- and post- actions

in OpenMPI today
promised by IBM in Spectrum MPI!

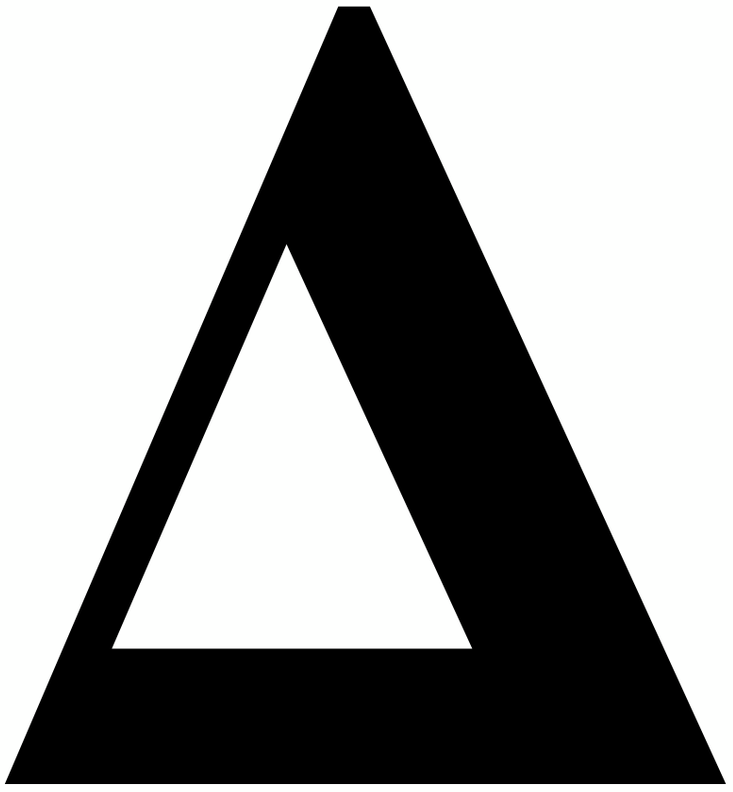
neighborly

policy independent

high-level, lots of libraries

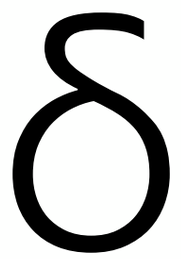
`mpi_jm` Required Application Changes

- One file to include, one library to link against.
- Two handshakes (after MPI `init` and `finalize`)
    - Disconnect for MPI safety and status call
- Binary works with or without `mpi_jm` management
    - Crucial for easy debugging
- Already integrated into QMP (`mpi_jm` branch)
  github.com/callat-qcd/qmp will be merged via pull request when **`mpi_jm`** is made public
    - Configure with `--with-mpi-jm=/path/to/mpi_jm`

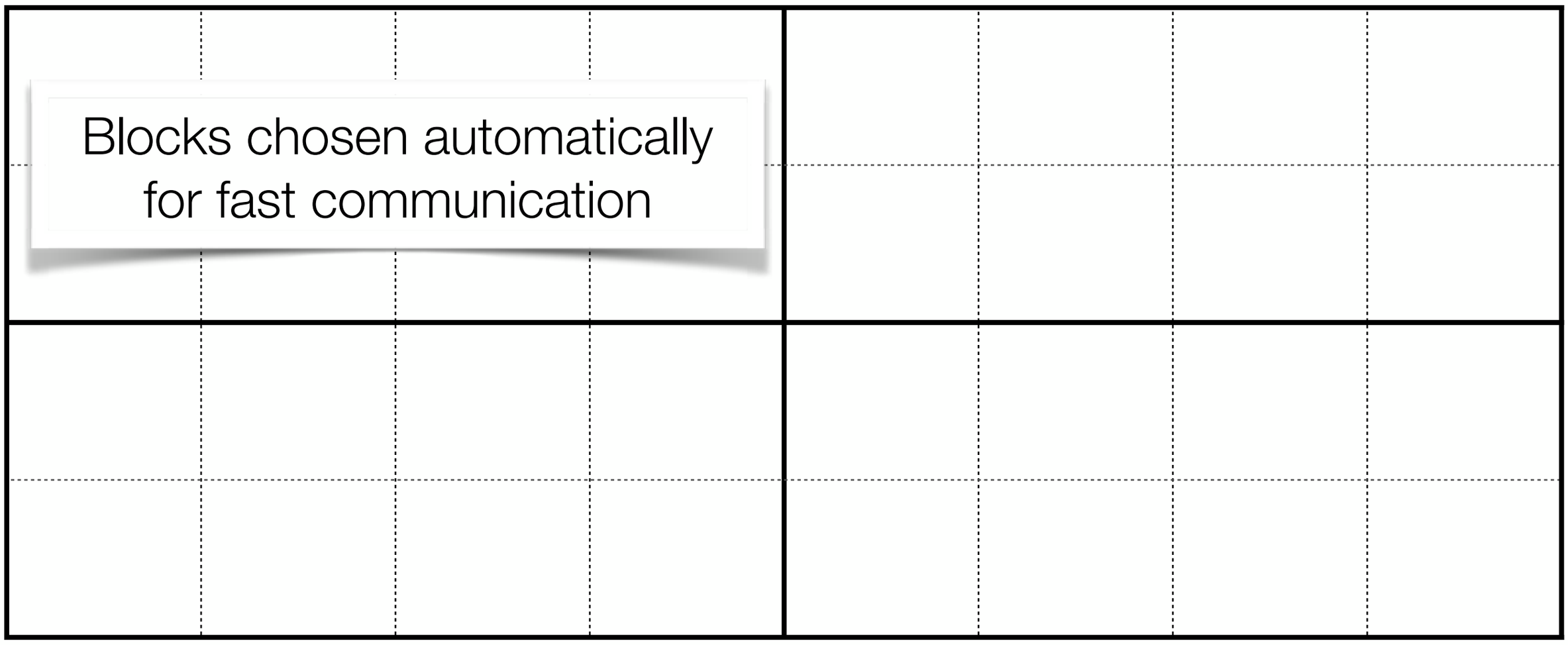


`mpi_jm` Required Application Changes

- One file to include, one library to link against.
- Two handshakes (after MPI `init` and `finalize`)
    - Disconnect for MPI safety and status call
- Binary works with or without `mpi_jm` management
    - Crucial for easy debugging
- Already integrated into QMP (`mpi_jm` branch)
  github.com/callat-qcd/qmp will be merged via pull request when **`mpi_jm`** is made public
    - Configure with `--with-mpi-jm=/path/to/mpi_jm`



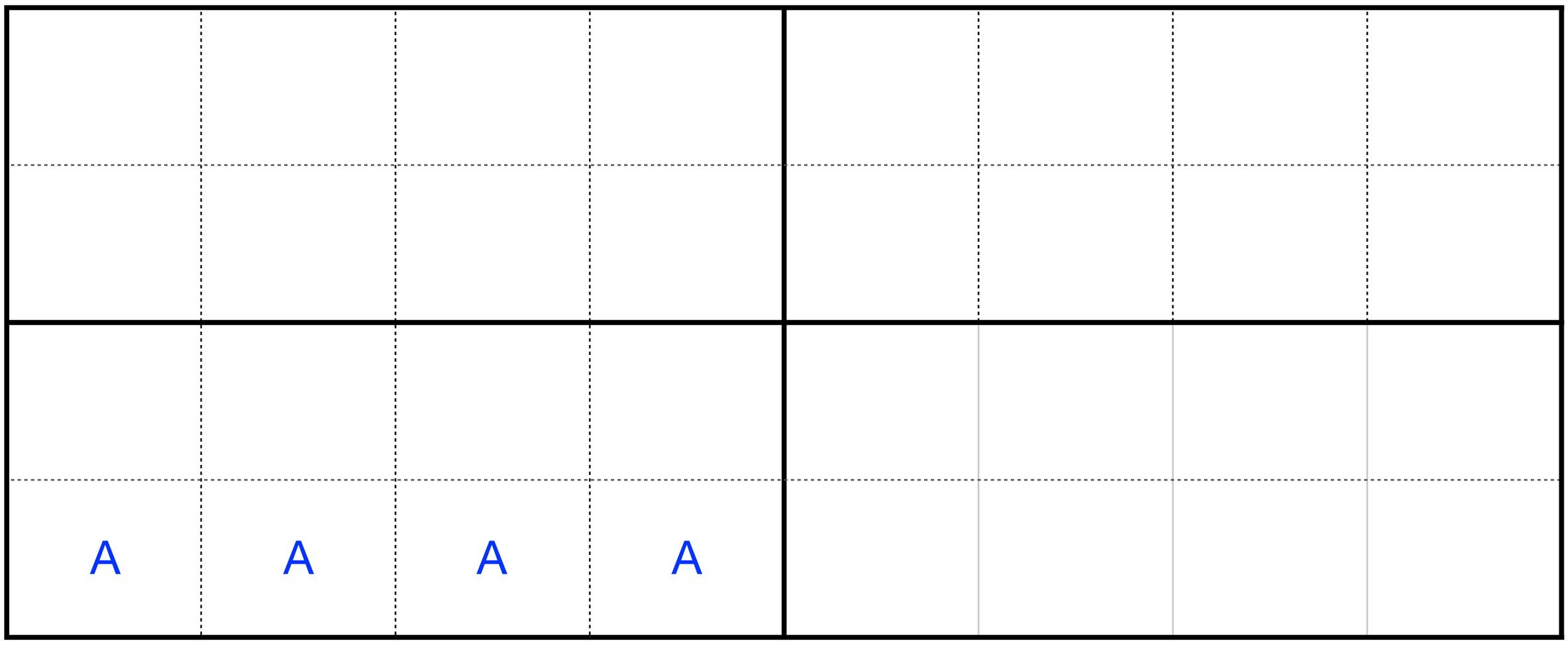
`mpi_jm` Block = 8 × Node with 16 CPUS + 1 GPU



32 node job

A: 4 × 16 CPU
B: 8 × 1 CPU+GPU
C: 8 × 14 CPU

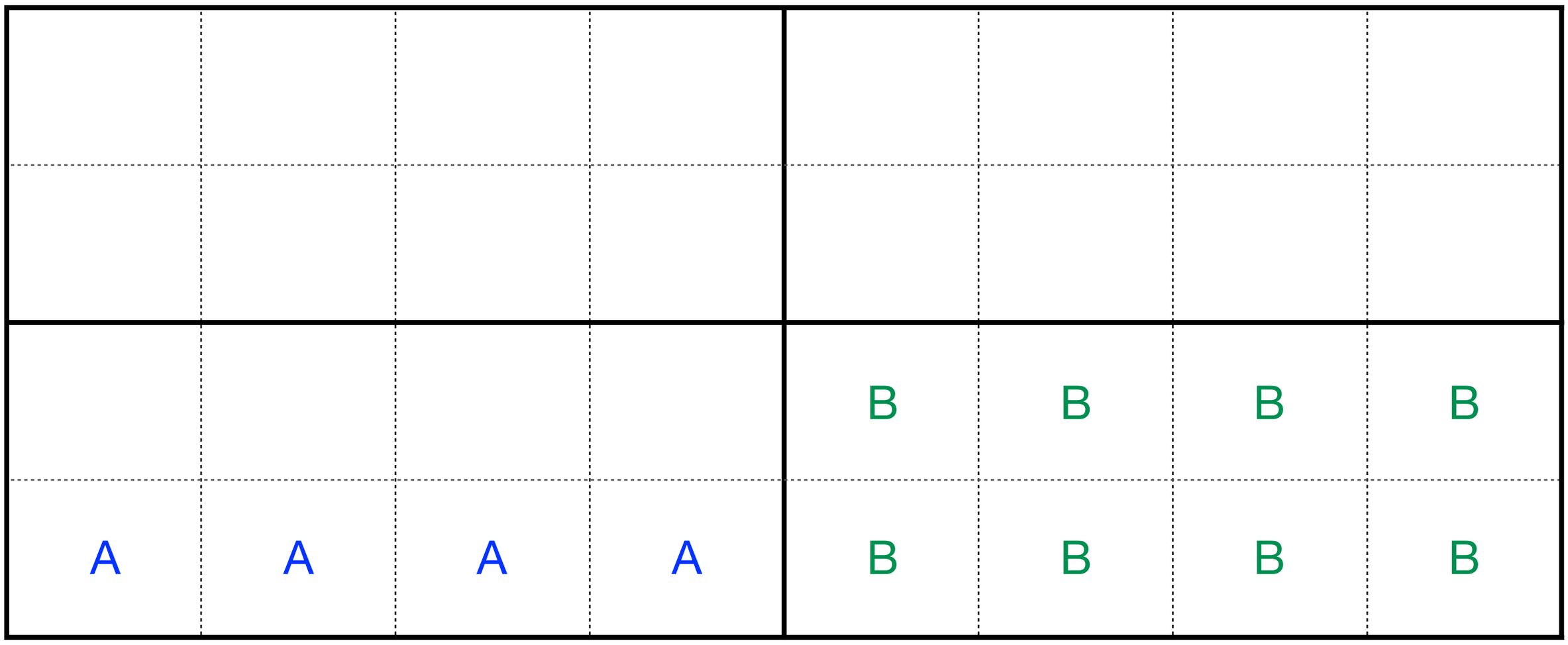
`mpi_jm` Block = 8 × Node with 16 CPUS + 1 GPU



32 node job

A: 4 × 16 CPU
B: 8 × 1 CPU+GPU
C: 8 × 14 CPU

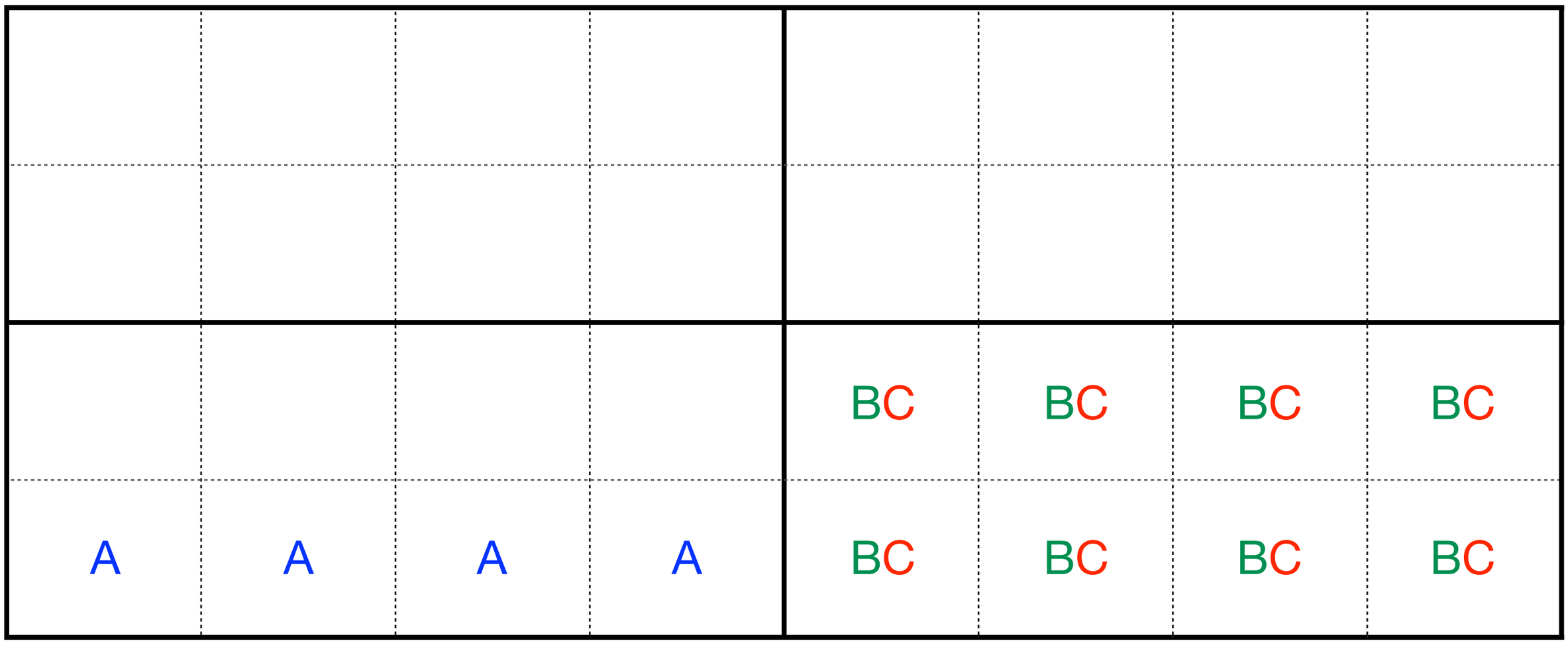
`mpi_jm` Block = 8 × Node with 16 CPUS + 1 GPU



32 node job

A: 4 × 16 CPU
B: 8 × 1 CPU+GPU
C: 8 × 14 CPU

`mpi_jm` Block = 8 × Node with 16 CPUS + 1 GPU

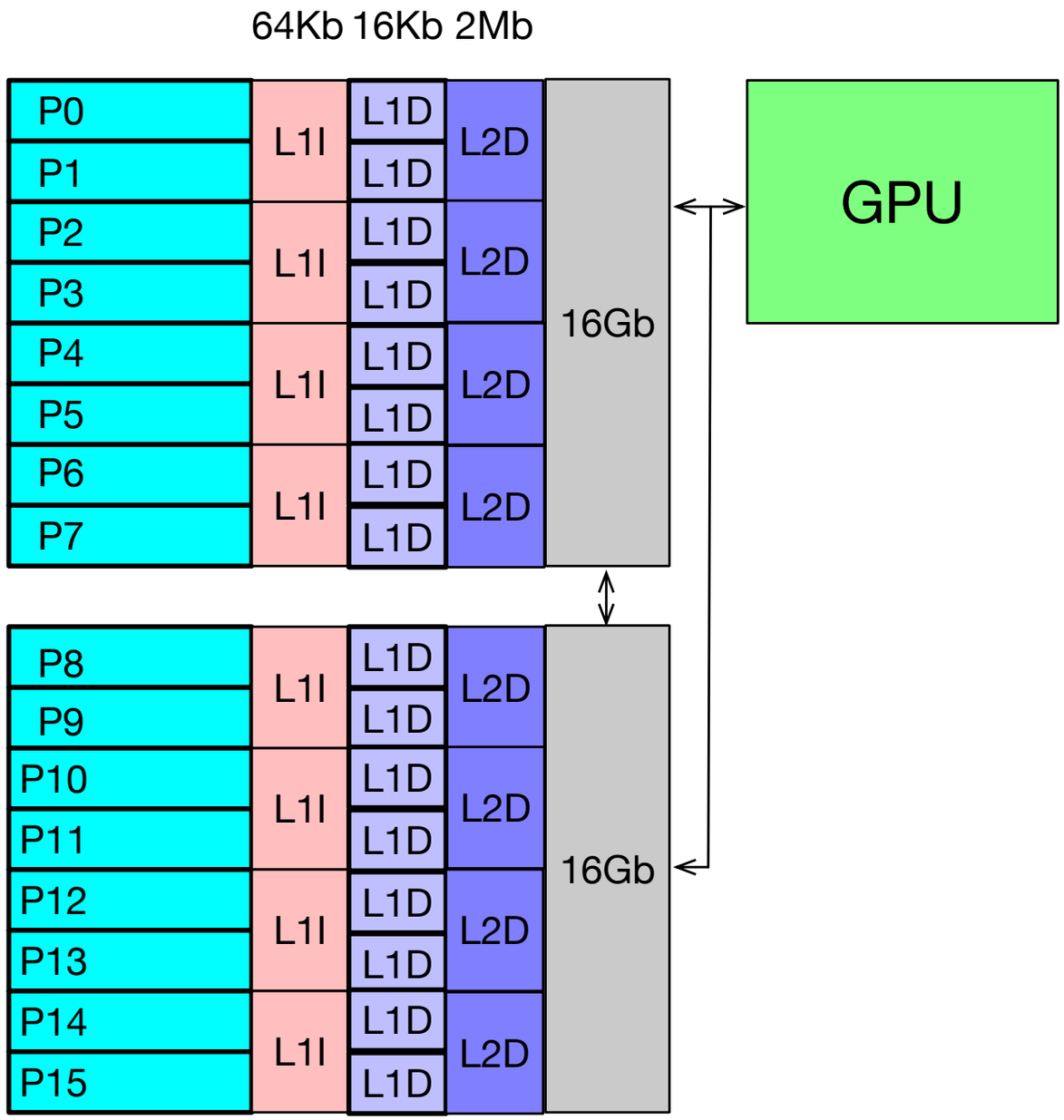


32 node job

A: 4 × 16 CPU
B: 8 × 1 CPU+GPU
C: 8 × 14 CPU

Simultaneous CPU + GPU tasks

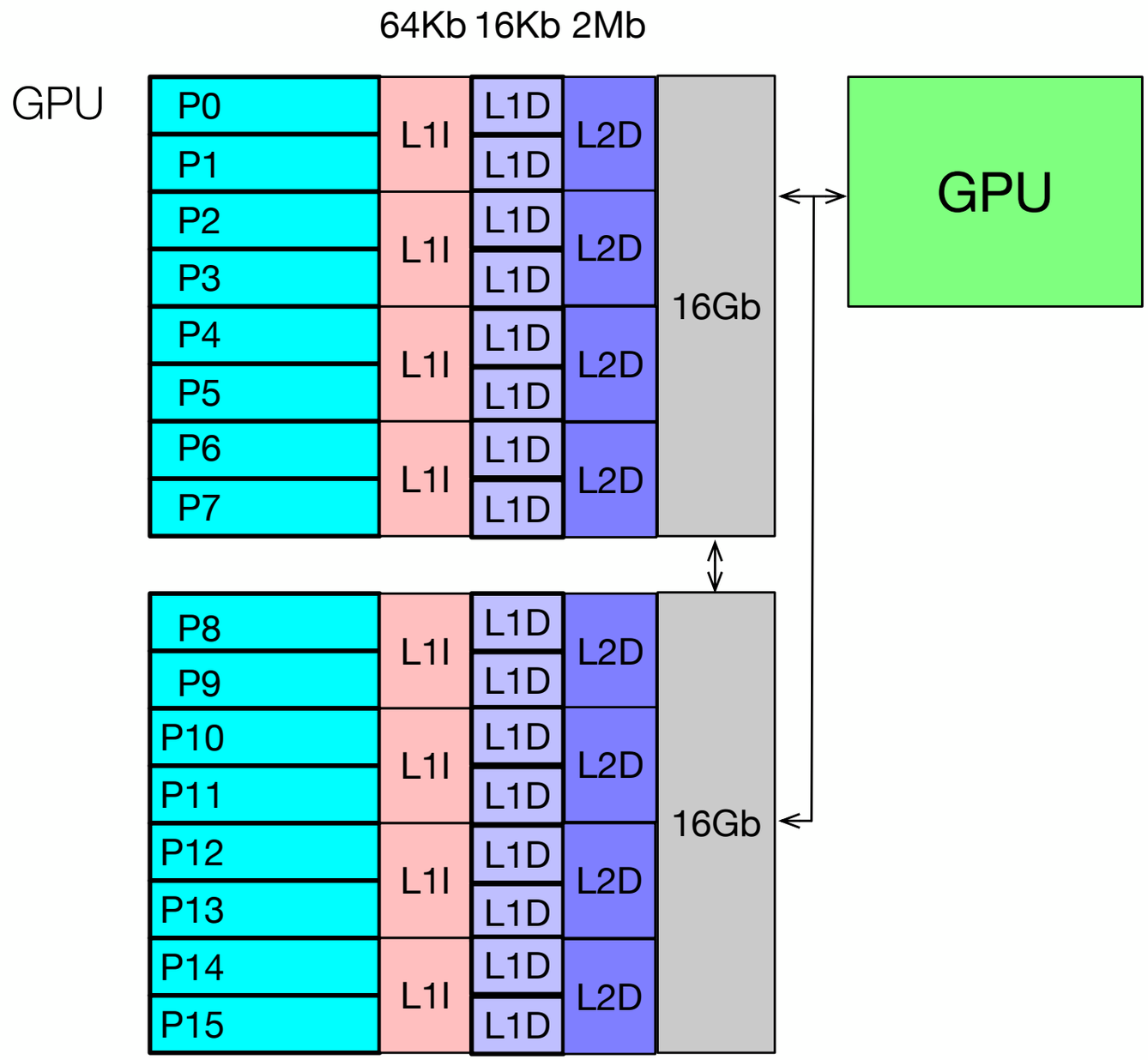
- GPU task needs one core. Place in P0
- Block P1 to protect caches etc.
- CPU ranks in numa0 will compete for bandwidth.
  - Tune #threads for max performance of both tasks.



slide courtesy of Ken McElvain

Simultaneous CPU + GPU tasks

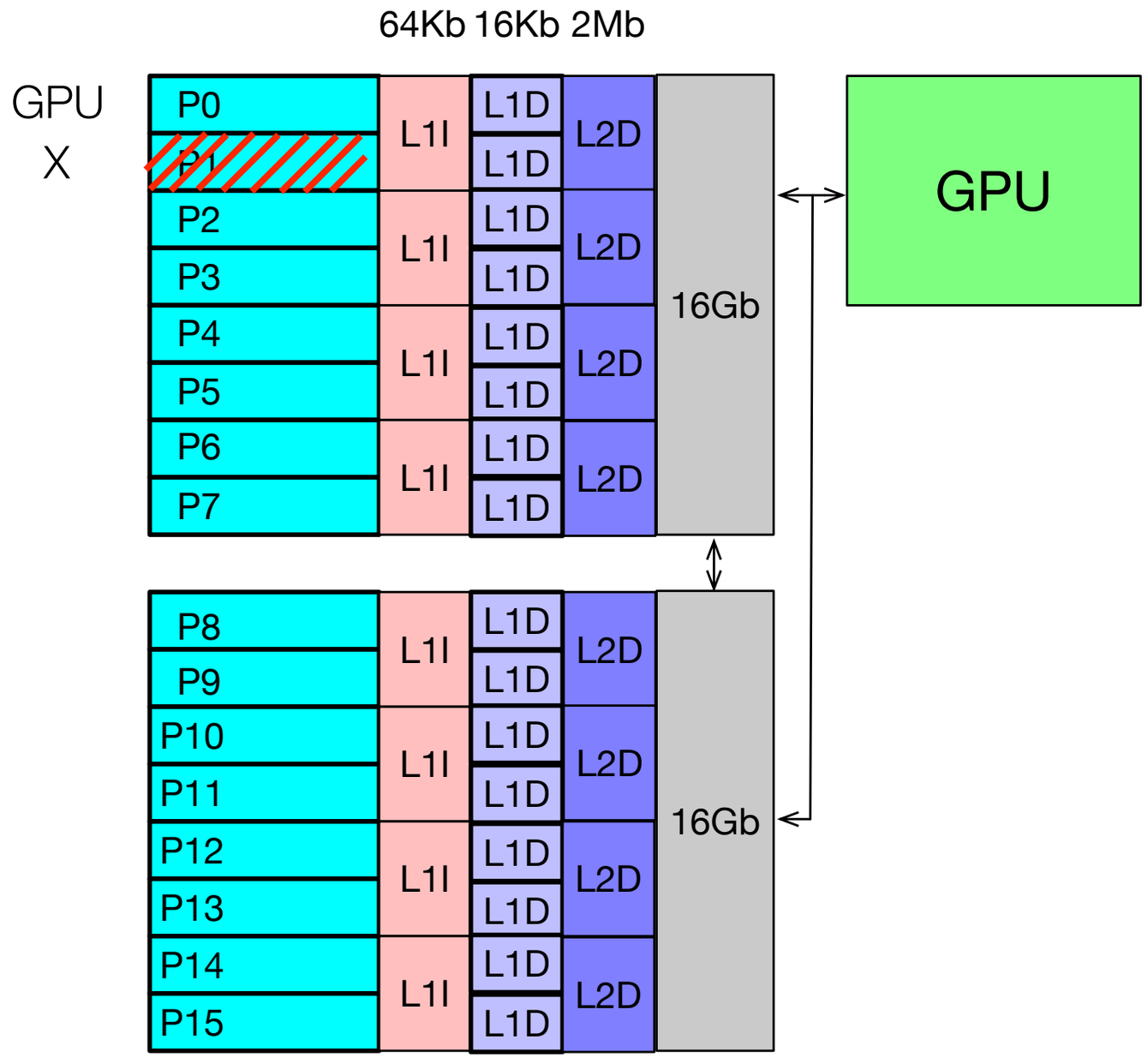
- GPU task needs one core. Place in P0
- Block P1 to protect caches etc.
- CPU ranks in numa0 will compete for bandwidth.
  - Tune #threads for max performance of both tasks.



slide courtesy of Ken McElvain

Simultaneous CPU + GPU tasks

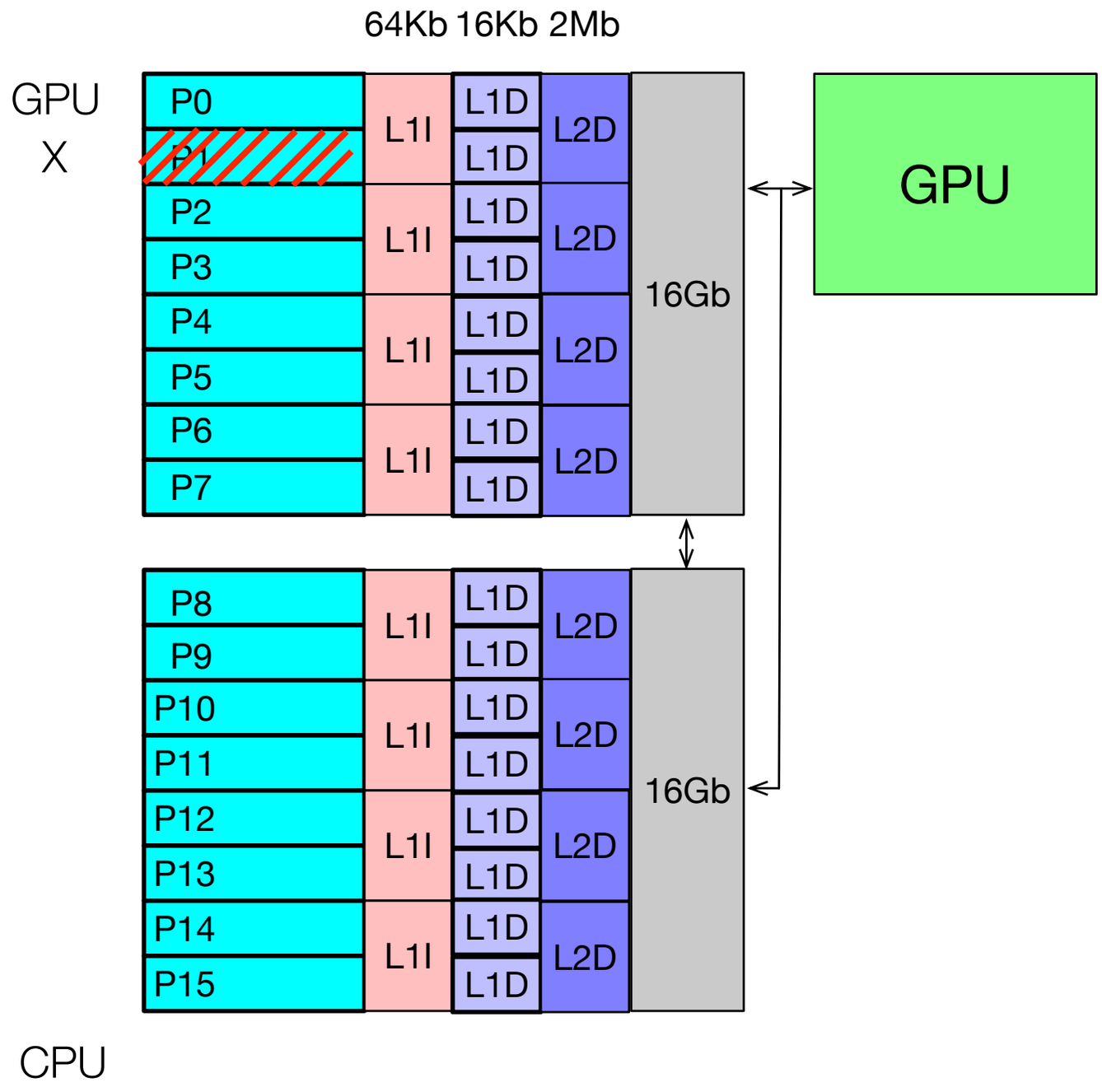
- GPU task needs one core. Place in P0
- Block P1 to protect caches etc.
- CPU ranks in numa0 will compete for bandwidth.
  - Tune #threads for max performance of both tasks.



slide courtesy of Ken McElvain

Simultaneous CPU + GPU tasks

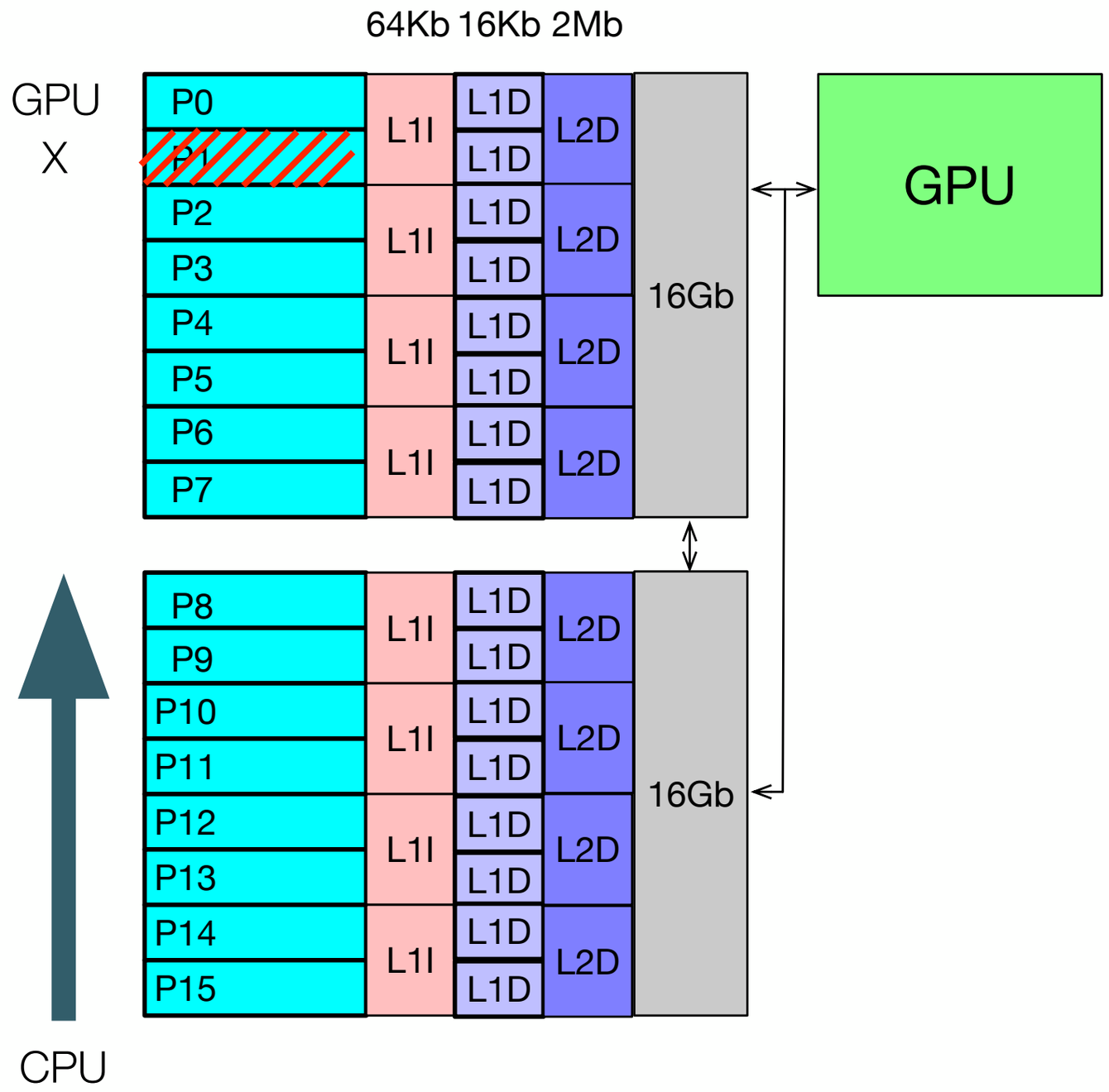
- GPU task needs one core. Place in P0
- Block P1 to protect caches etc.
- CPU ranks in numa0 will compete for bandwidth.
  - Tune #threads for max performance of both tasks.



slide courtesy of Ken McElvain

Simultaneous CPU + GPU tasks

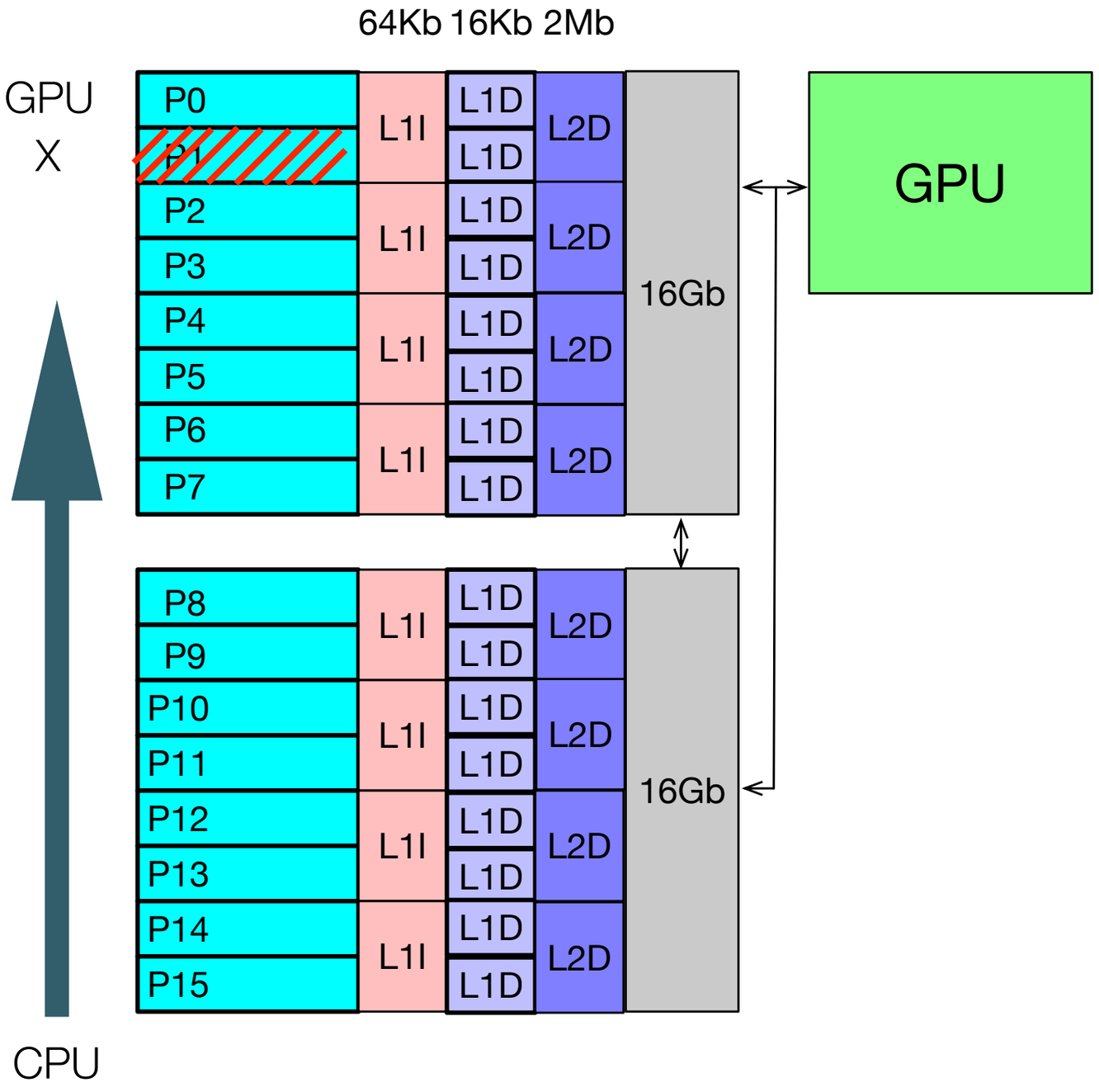
- GPU task needs one core. Place in P0
- Block P1 to protect caches etc.
- CPU ranks in numa0 will compete for bandwidth.
  - Tune #threads for max performance of both tasks.



slide courtesy of Ken McElvain

Simultaneous CPU + GPU tasks

- GPU task needs one core. Place in P0
- Block P1 to protect caches etc.
- CPU ranks in numa0 will compete for bandwidth.
  - Tune #threads for max performance of both tasks.



slide courtesy of Ken McElvain

Python Workload Generation

- Tasks are YAML python dictionaries.
- Class parses task and knows multiple valid configurations.
- The task queue selects one configuration to match available resources.

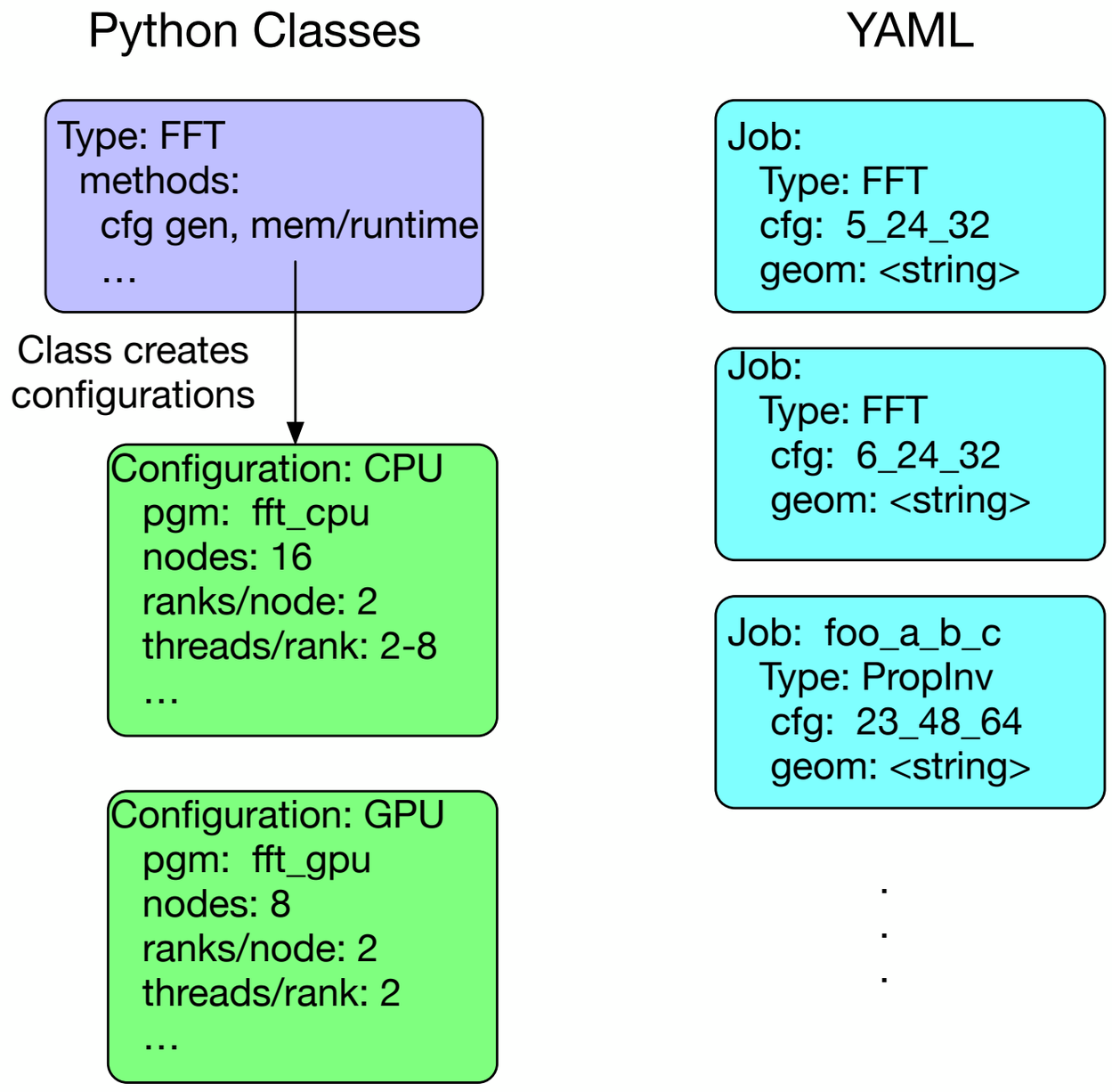


figure courtesy of Ken McElvain

`mpi_jm` Benefits

- Take advantage of policy incentives by bundling tasks together
- Backfill small tasks, reduce waste
- Use CPUs + GPUs independently
- Simplified workflow + collaboration

- Useable across grids (if MPI works)
  eg. Big PanDA https://www.bnl.gov/newsroom/news.php?a=25796
- Useful for next-gen machines
  `MPI_comm_spawn` promised in Spectrum MPI
- Modest application changes
- Python interface could be mated to a database

δ