OAK RIDGE National Laboratory

Slurm on Frontier

Tom Papatheodore

HPC Engineer
System Acceptance & User Environment Group
Oak Ridge Leadership Computing Facility (OLCF)
Oak Ridge National Laboratory (ORNL)

May 18, 2023

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

Using Slurm on Frontier

Slurm provides 3 ways of submitting and launching jobs on Frontier compute nodes: **batch scripts, interactive, and single-command**.

> **NOTE:** Jobs launch on the allocated compute nodes, with the first compute node in the allocation serving as head-node.

Slurm commands associated with the 3 methods:

`sbatch`	Used to submit a batch script to allocate a Slurm job allocation. The script contains options preceded with `#SBATCH`.
`salloc`	Used to allocate an interactive Slurm job allocation, where one or more job steps (i.e., `srun` commands) can then be launched on the allocated resources (i.e., nodes).
`srun`	Used to run a parallel job (job step) on the resources allocated with `sbatch` or `salloc`, or used to create a resource allocation and run a job step in a single command.

Batch Scripts

A **batch script** can be used to submit a job to run on the compute nodes at a later time. In this case, stdout and stderr will be written to a file(s) that can be opened after the job completes.

Here is an example of a simple batch script:

Line	Actual batch script	Description
1	`#!/bin/bash`	[optional] shell interpreter line
2	`#SBATCH -A <project_id>`	OLCF project to charge
3	`#SBATCH -J <job_name>`	Job name
4	`#SBATCH -o %x-%j.out`	stdout file name (`%x` is job name, `%j` is job id)
5	`#SBATCH -t 00:05:00`	Walltime requested (`HH:MM:SS`)
6	`#SBATCH -p <partition>`	Batch queue
7	`#SBATCH -N 2`	Number of computed nodes requested
8		Blank line
9	`srun -N2 -n4 --ntasks-per-node=2 ./a.out`	srun command to launch parallel job

The script would then be submitted to the queue using `sbatch <slurm_script>`

Interactive Jobs

An **interactive job** allows multiple job steps (i.e., multiple `srun` commands) to be launched (interactively) on compute nodes allocated with the `salloc` command.

Here is an example of such a job:

```
$ salloc -A <project_id> -J <job_name> -t 00:05:00 -p <partition> -N 2
salloc: Granted job allocation 4258
salloc: Waiting for resource configuration
salloc: Nodes frontier[10469-10470] are ready for job

$ srun -n 4 --ntasks-per-node=2 ./a.out
<output printed to terminal>

$ srun -n 2 --ntasks-per-node=1 ./a.out
<output printed to terminal>
```

Here, `salloc` is used to request 2 compute nodes for 5 minutes, and once the compute nodes become available, job steps can be launched on them using `srun`.

Single Command (non-interactive)

A **single command** job allows job allocation and a job step to be run in the same `srun` command.

Here is an example of such a job:

```
$ srun -A <project_id> -t 00:05:00 -p <partition> -N 2 -n 4 --ntasks-per-node=2 ./a.out
<output printed to terminal>
```

Here, `srun` is used to request 2 compute nodes for 5 minutes and launch a job step with 4 tasks (2 on each node) – all in the same single command.

Common Slurm Options and Commands

Common Slurm Submission Options:

see `man sbatch` for more info

`-A <project_id>`	Project ID to charge
`-J <job_name>`	Name of job
`-p <partition>`	Partition / batch queue
`-t <time>`	Wall clock time <HH:MM:SS> (or you can just give minutes)
`-N <number_of_nodes>`	Number of compute nodes
`-o <file_name>`	Standard output file name
`-e <file_name>`	Standard error file name
`--threads-per-core=<threads>`	Number of active hardware threads per core [1 (default) or 2]

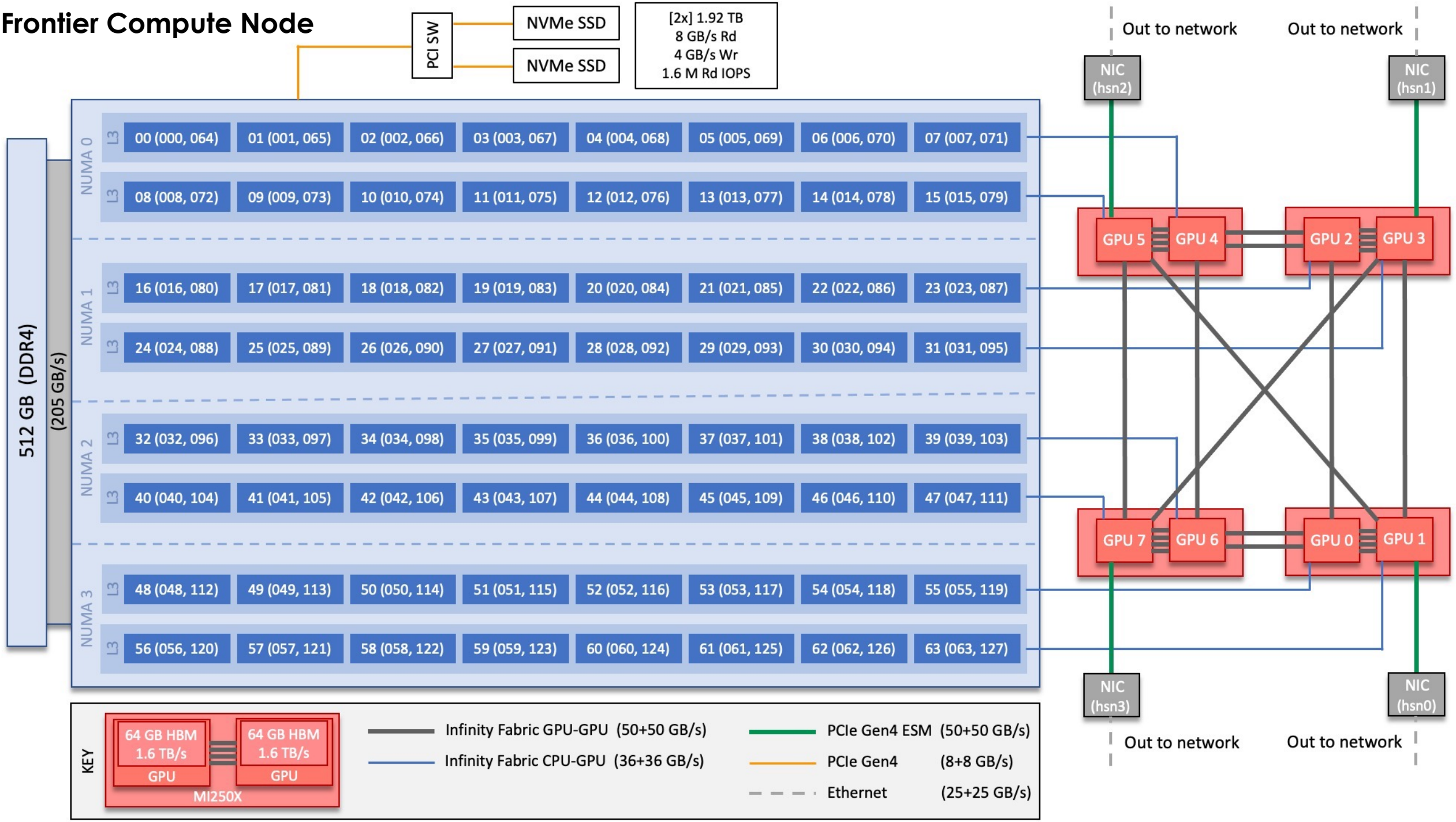
Common Slurm Submission Options:

see individual `man` pages for more info

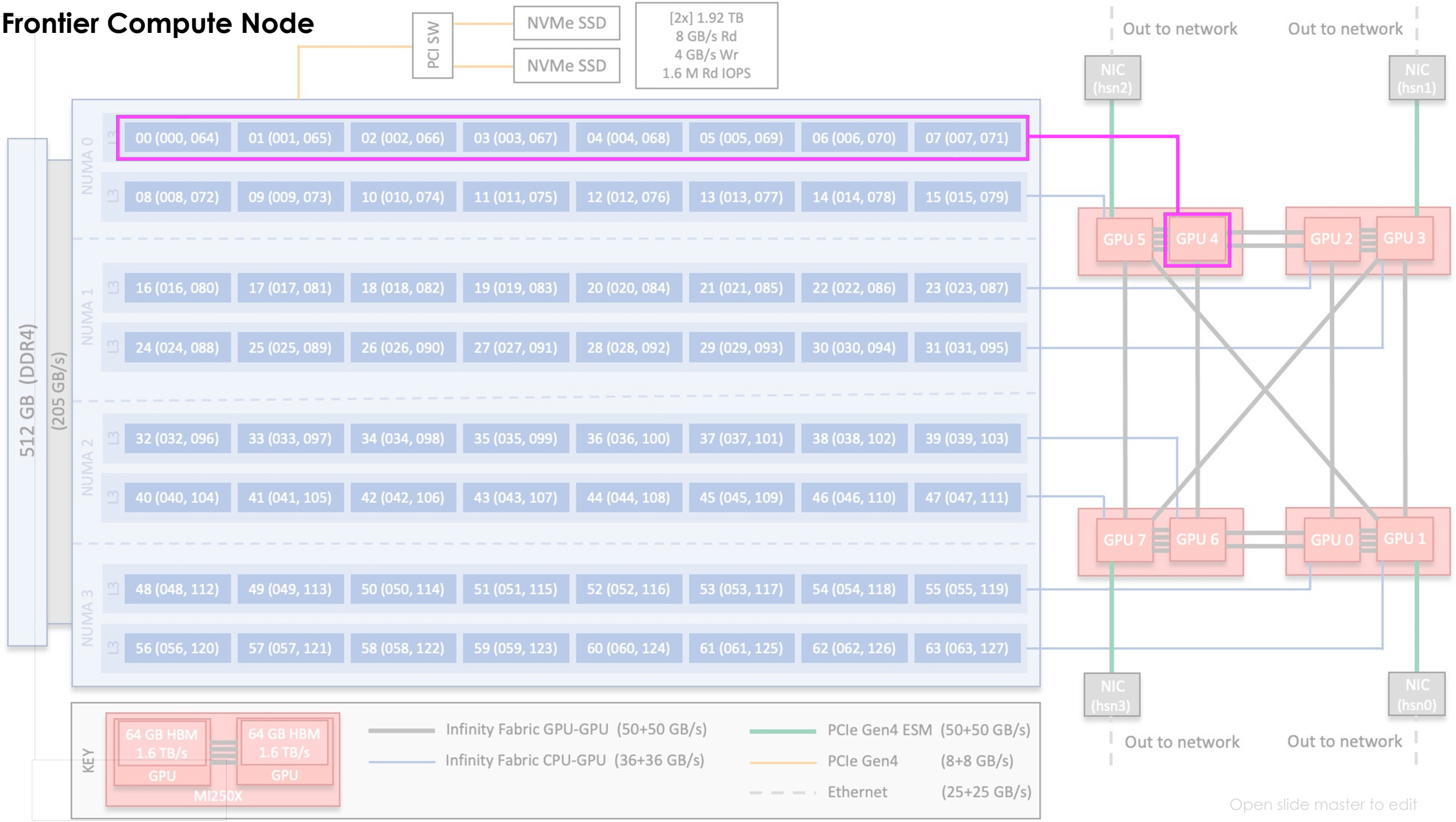
`sinfo`	Used to view partition and node information.
`squeue`	Used to view job and job step information for jobs in the scheduling queue.
`sacct`	Used to view accounting data for jobs and job steps in the accounting log (in queue or recently completed).
`scancel`	Used to signal or cancel jobs or job steps.
`scontrol`	Used to view or modify job configuration.

Also use `jobstat` for queue information

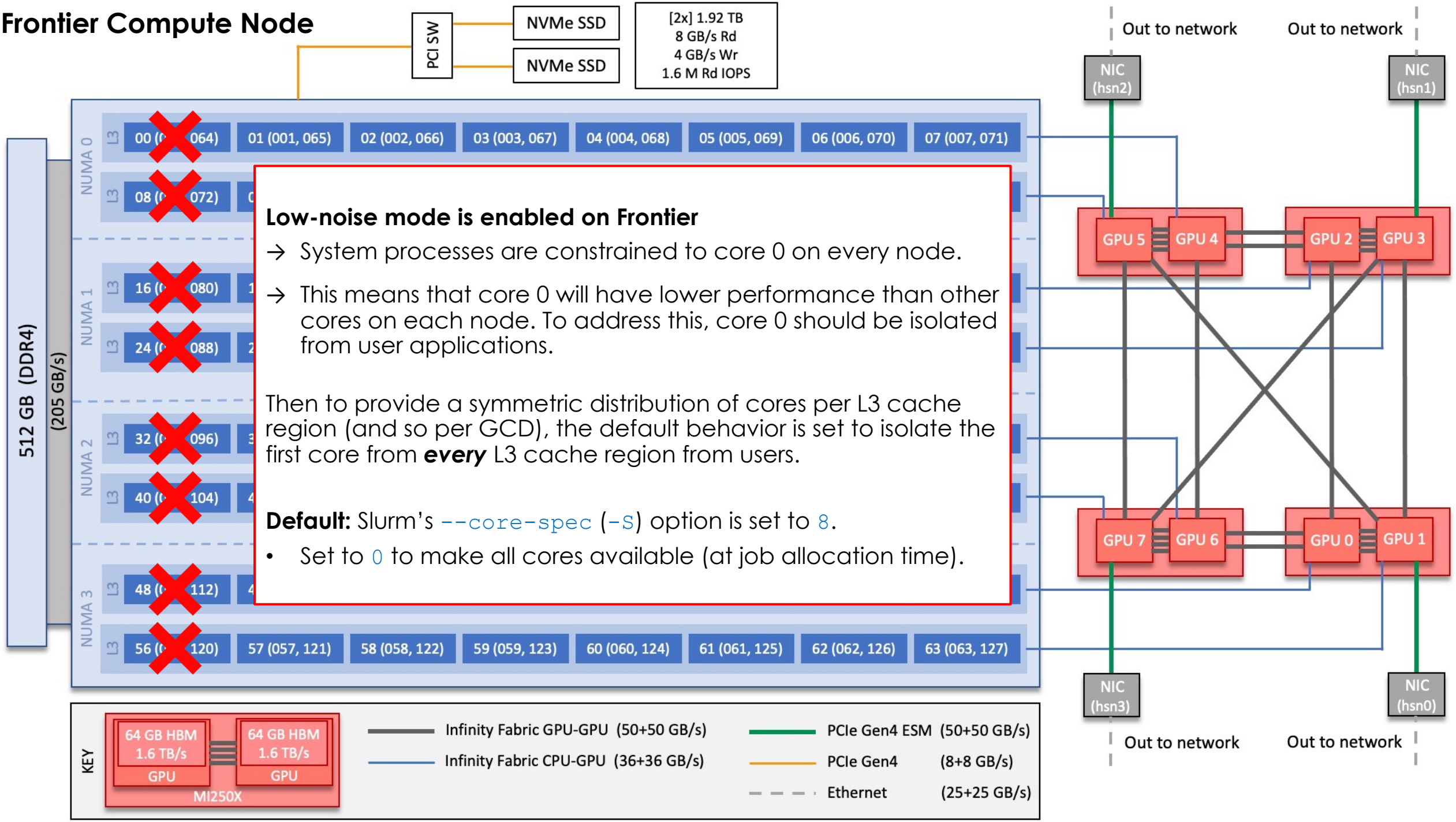
Frontier Compute Node



Frontier Compute Node



Frontier Compute Node



Low-noise mode is enabled on Frontier

→ System processes are constrained to core 0 on every node.

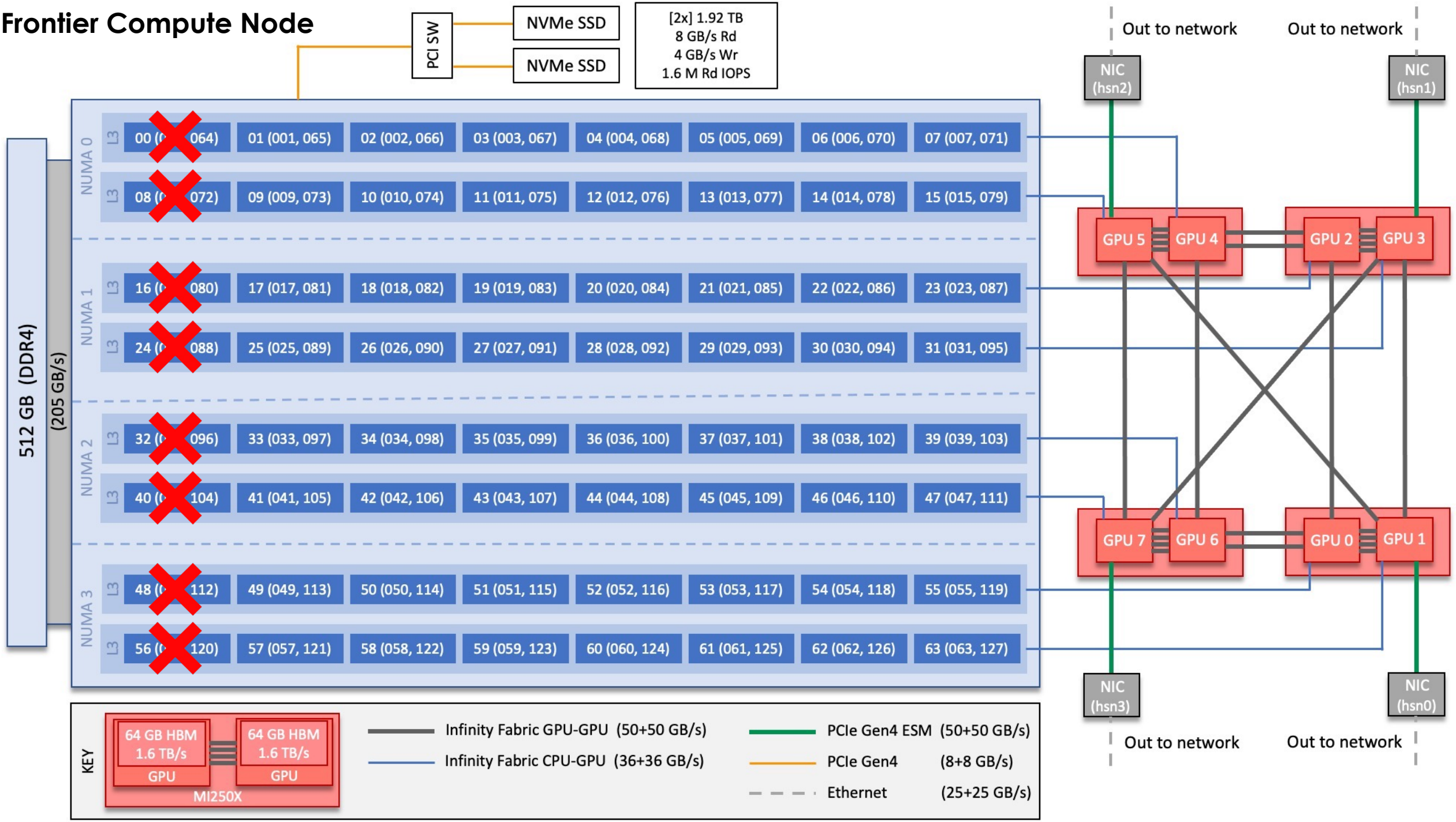
→ This means that core 0 will have lower performance than other cores on each node. To address this, core 0 should be isolated from user applications.

Then to provide a symmetric distribution of cores per L3 cache region (and so per GCD), the default behavior is set to isolate the first core from ***every*** L3 cache region from users.

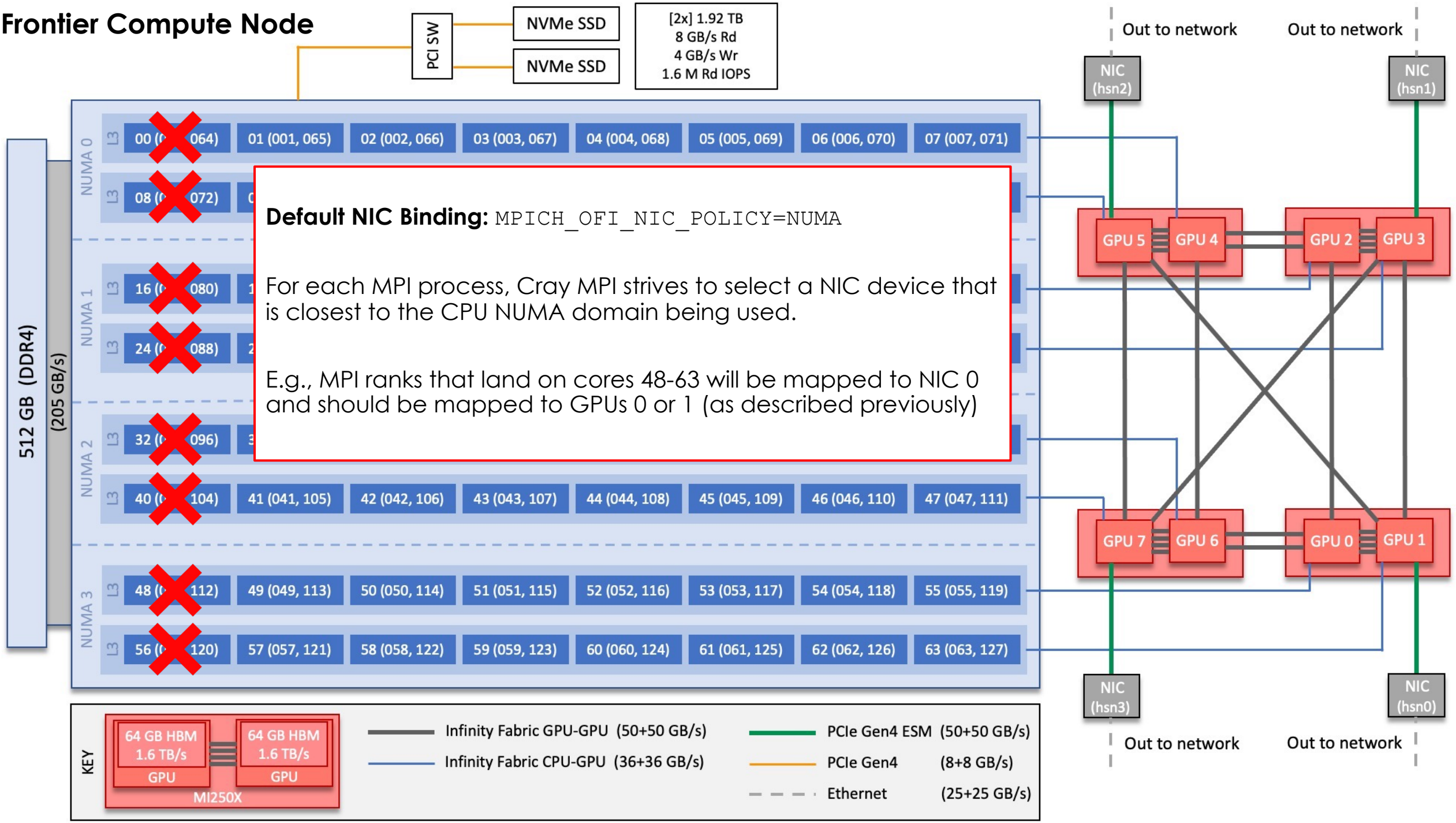
Default: Slurm's `--core-spec` (`-S`) option is set to `8`.

- Set to `0` to make all cores available (at job allocation time).

Frontier Compute Node



Frontier Compute Node



Default NIC Binding: `MPICH_OFI_NIC_POLICY=NUMA`

For each MPI process, Cray MPI strives to select a NIC device that is closest to the CPU NUMA domain being used.

E.g., MPI ranks that land on cores 48-63 will be mapped to NIC 0 and should be mapped to GPUs 0 or 1 (as described previously)

Examples

If you want to run any of your own job steps along the way, we have an 18-node reservation during this tutorial you can use.

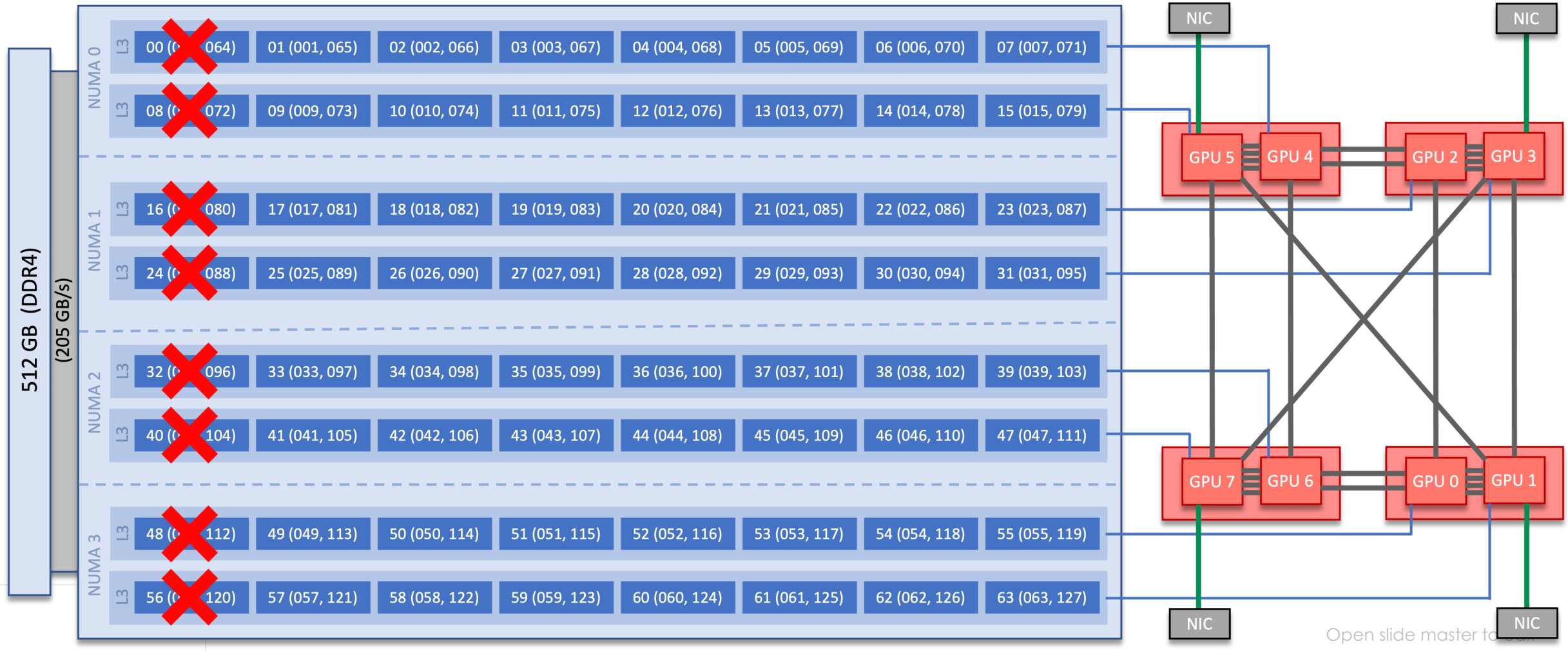
To access it, simply add the following to your job request at job-allocation time:

```
--reservation=slurm-tutorial
```

Slurm GPU Bindings (Recall: MPI ranks should target the GPU associated with their L3 cache region)

Always check process/thread/GPU bindings: https://code.ornl.gov/olcf/hello_jobstep

- **`GPU_ID`** – node-level (or global) GPU ID read from `ROCR_VISIBLE_DEVICES`.
- **`RT_GPU_ID`** – HIP runtime GPU ID (as reported from `hipGetDevice`).
- **`Bus_ID`** – physical bus ID associated with the GPUs. Comparing bus IDs shows that different GPUs are being used.



hello_jobstep (https://code.ornl.gov/olcf/hello_jobstep)

Used to test process, thread, and GPU binding

NOTE: a *job step* is simply when you launch a program on a compute node using `srun`.

Clone the repo and compile:

```
$ git clone https://code.ornl.gov/olcf/hello_jobstep.git
$ cd hello_jobstep
$ module load craype-accel-amd-gfx90a rocm
$ make
```

Usage:

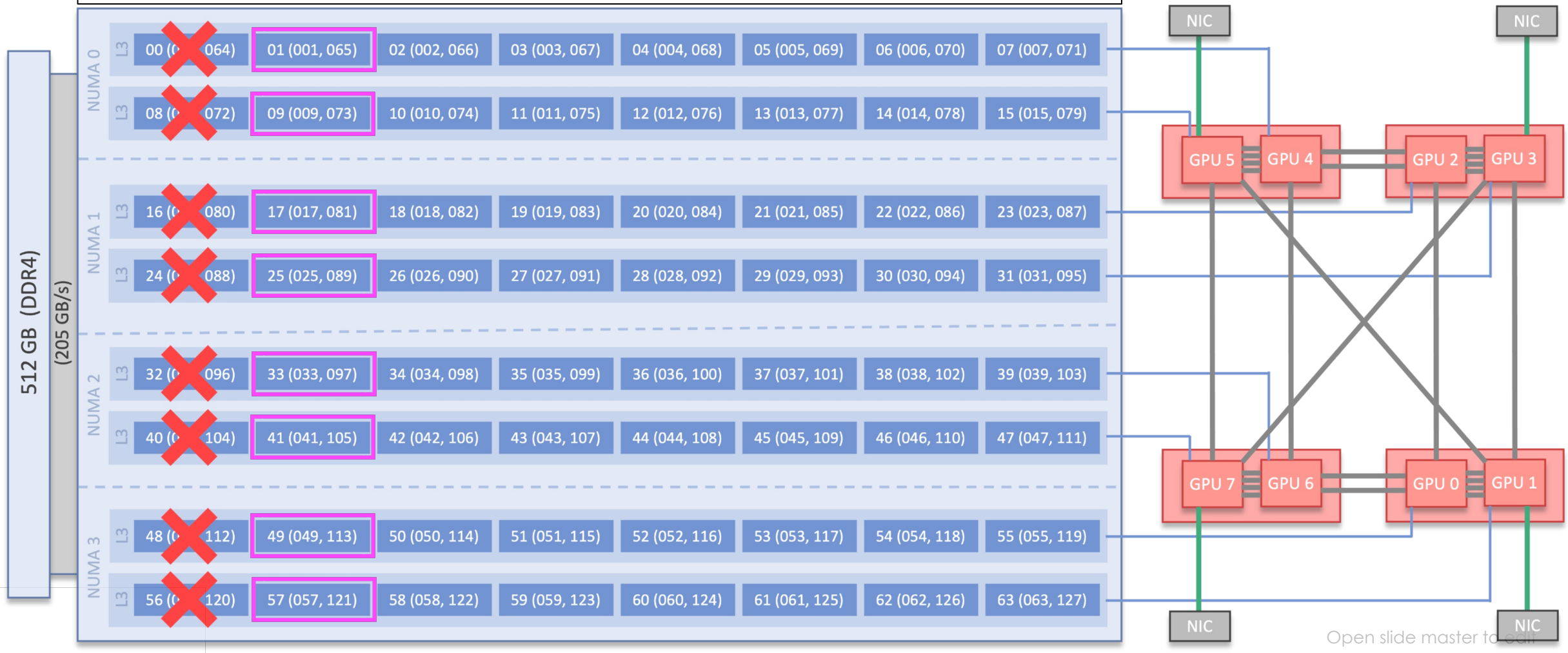
```
$ OMP_NUM_THREADS=1 srun -N2 -n16 -c7 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 009 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 017 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 041 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 006 - OMP 000 - HWT 049 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 057 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 008 - OMP 000 - HWT 001 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 009 - OMP 000 - HWT 009 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 010 - OMP 000 - HWT 017 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 011 - OMP 000 - HWT 025 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 012 - OMP 000 - HWT 033 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 013 - OMP 000 - HWT 041 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 014 - OMP 000 - HWT 049 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 015 - OMP 000 - HWT 057 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

- **MPI** – MPI rank ID
- **OMP** – OpenMP thread ID
- **HWT** – CPU hardware thread ID
- **Node** – Frontier node ID
- **RT_GPU_ID** – HIP runtime GPU ID (as reported from `hipGetDevice`)
- **GPU_ID** – node-level (or global) GPU ID read from `ROCR_VISIBLE_DEVICES`
- **Bus_ID** – Bus ID of GPU

```
$ OMP_NUM_THREADS=1 srun -N1 -n8 -c7 --gpus-per-node=8 ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier08303 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 001 - OMP 000 - HWT 009 - Node frontier08303 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 002 - OMP 000 - HWT 017 - Node frontier08303 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 003 - OMP 000 - HWT 025 - Node frontier08303 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 004 - OMP 000 - HWT 033 - Node frontier08303 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 005 - OMP 000 - HWT 041 - Node frontier08303 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 006 - OMP 000 - HWT 049 - Node frontier08303 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 007 - OMP 000 - HWT 057 - Node frontier08303 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
```

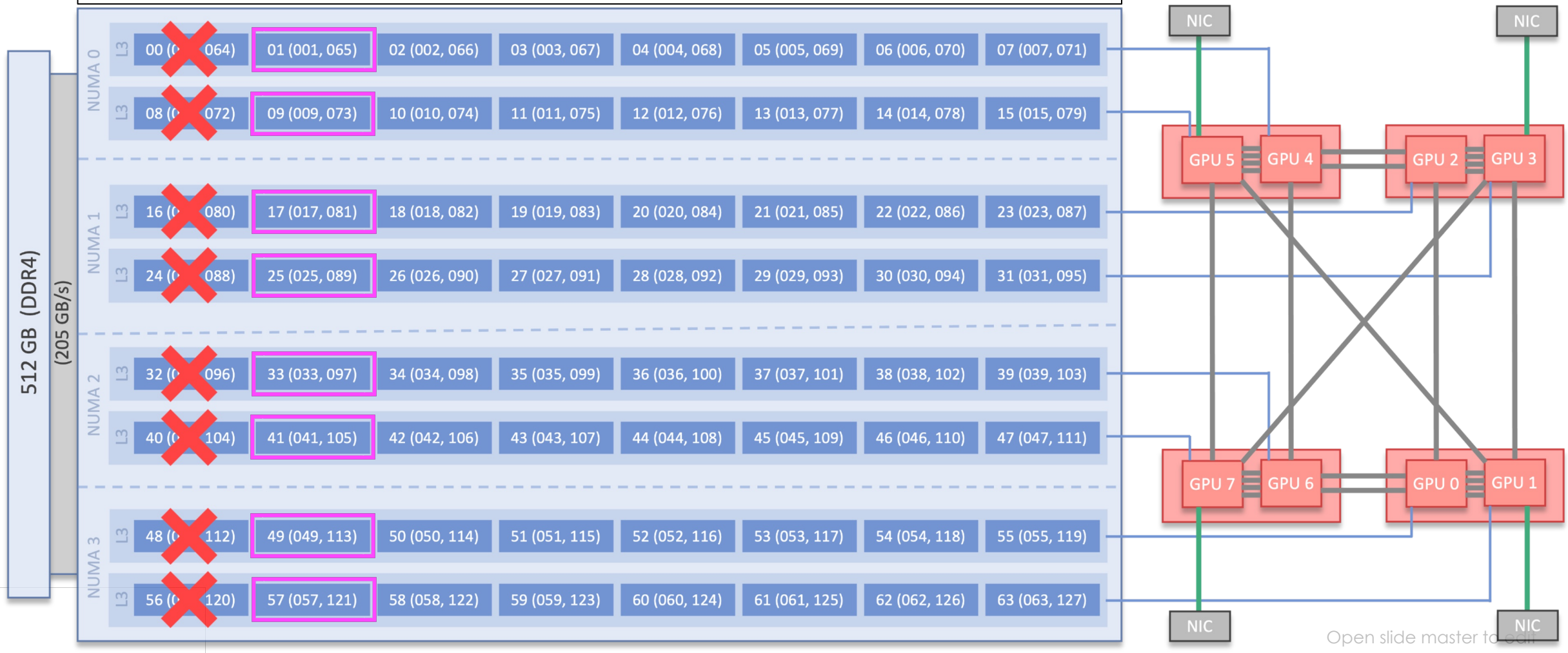
Incorrect mapping!

8 tasks per node, each with 7 CPU cores and 1 GPU



```
$ OMP_NUM_THREADS=1 srun -N1 -n8 -c7 --gpus-per-node=8 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 009 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 017 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 041 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 006 - OMP 000 - HWT 049 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 057 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

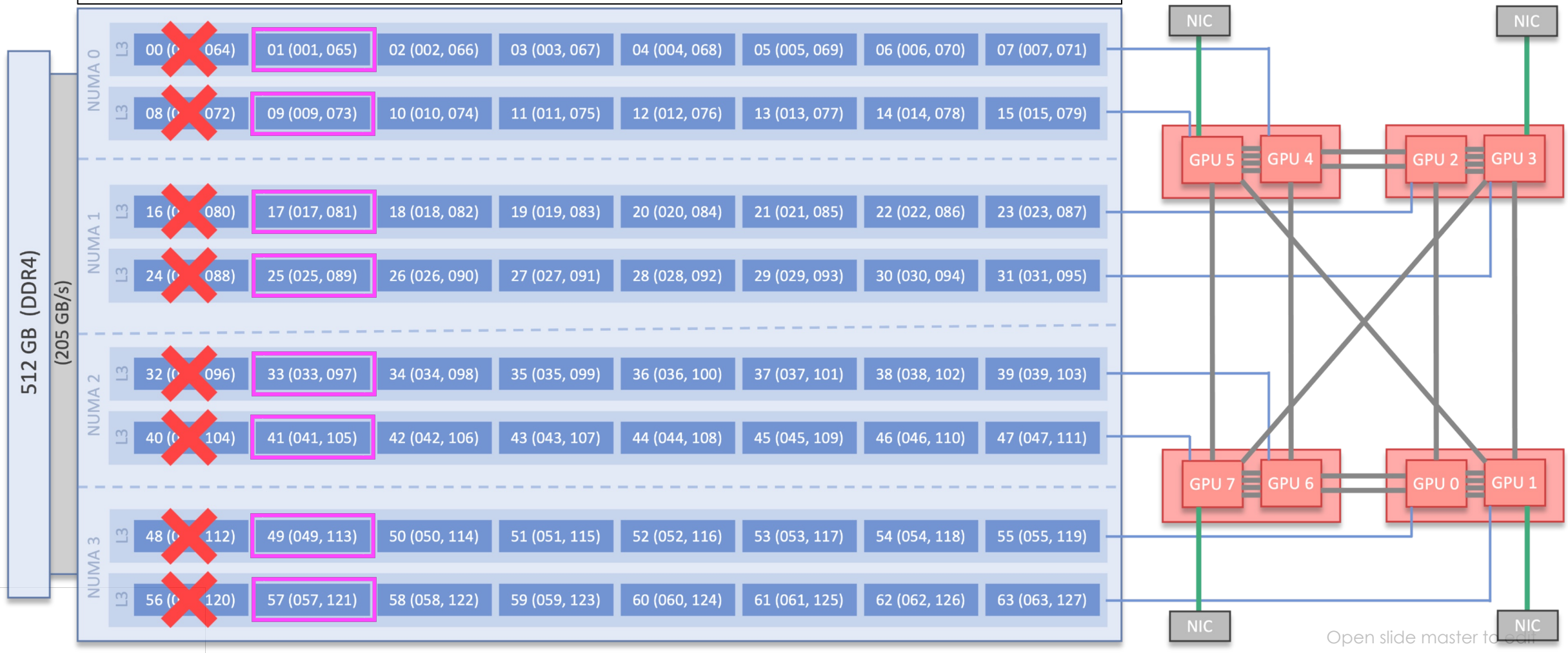
8 tasks per node, each with 7 CPU cores and 1 GPU



```
$ OMP_NUM_THREADS=1 srun -N1 -n8 -c7 --gpus-per-task=1 ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 001 - OMP 000 - HWT 009 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 002 - OMP 000 - HWT 017 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 005 - OMP 000 - HWT 041 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 006 - OMP 000 - HWT 049 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 007 - OMP 000 - HWT 057 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
```

Incorrect mapping!

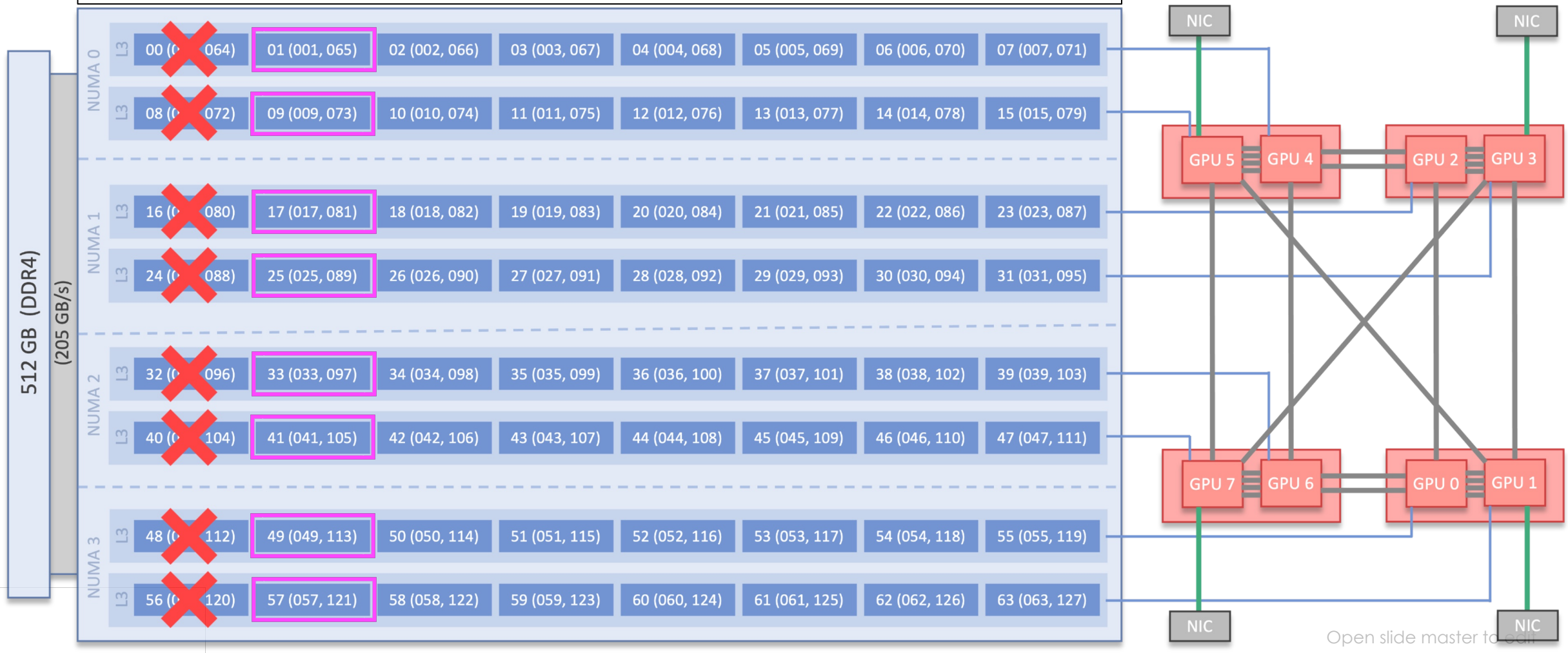
8 tasks per node, each with 7 CPU cores and 1 GPU



Open slide master to edit

```
$ OMP_NUM_THREADS=1 srun -N1 -n8 -c7 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 009 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 017 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 041 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 006 - OMP 000 - HWT 049 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 057 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

8 tasks per node, each with 7 CPU cores and 1 GPU

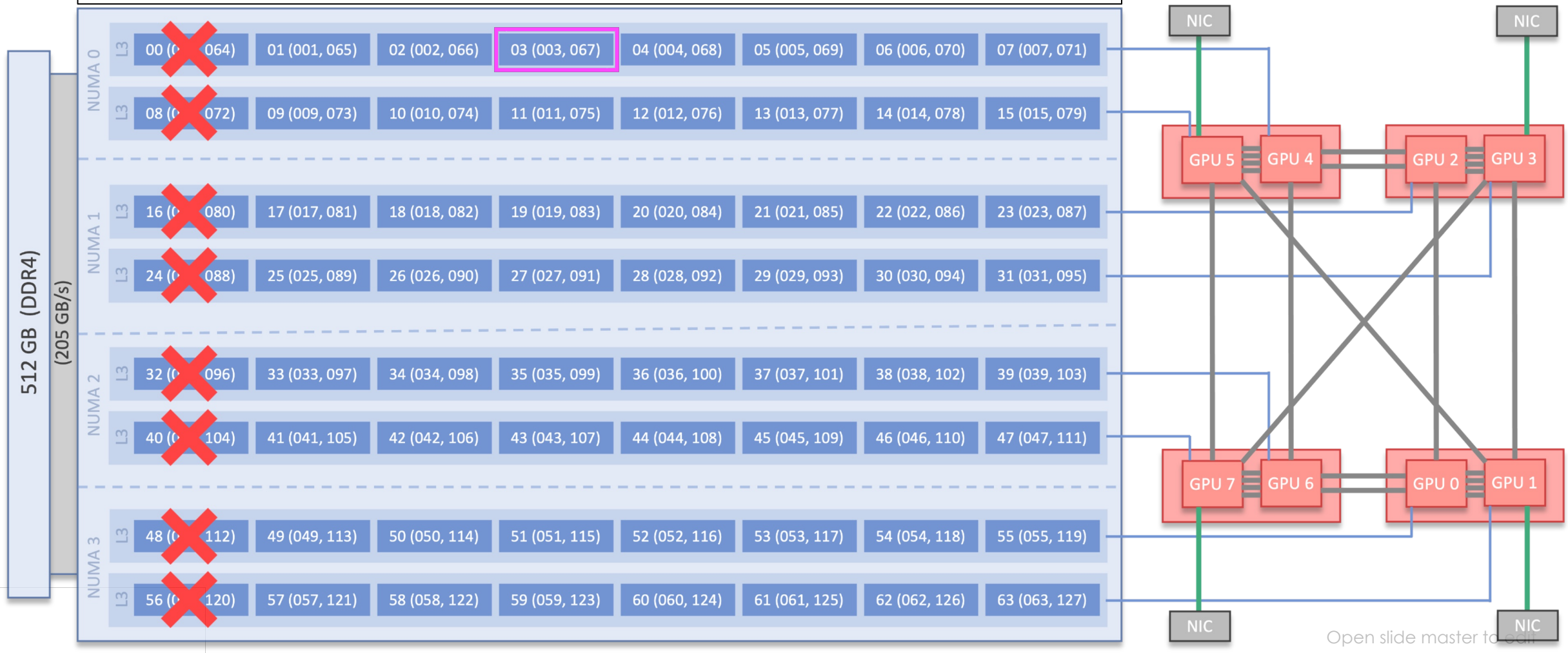


```
$ OMP_NUM_THREADS=1 srun -N1 -n1 -c7 --gpus-per-node=8 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 003 - Node frontier08303 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
```

Incorrect mapping!

```
--gpu-bind=[verbose,]<type>

Bind tasks to specific GPUs.  By default every spawned task can access every GPU allocated to the step.  If "verbose," is
specified before  <type>.  then print out GPU binding debug information to the stderr of the tasks. GPU binding is ignored if
there is only one task. ʘ_ʘ
```

1 task per node, with 7 CPU cores and 1 GPU



Open slide master to edit

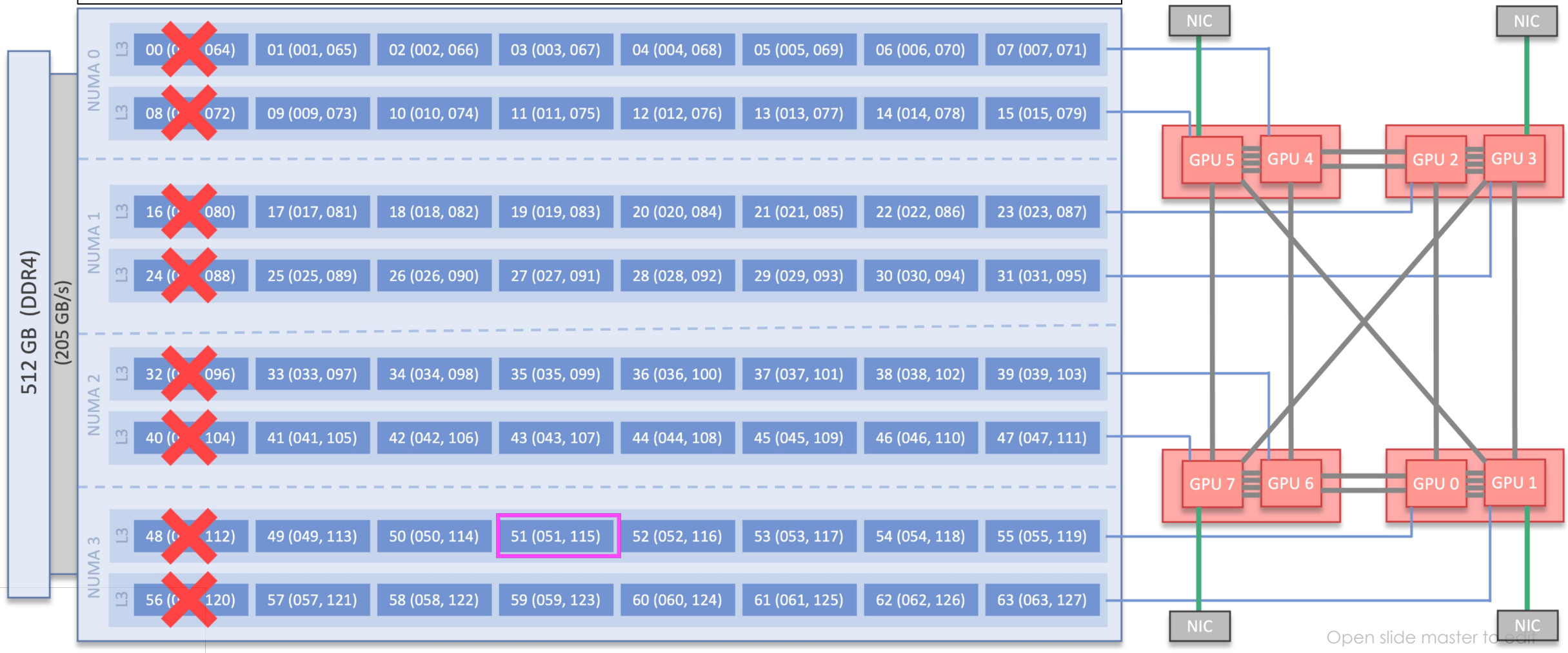
```
$ OMP_NUM_THREADS=1 srun -N1 -n1 -c7 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 051 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
```

✅

```
--gpu-bind=[verbose,]<type>

Bind tasks to specific GPUs.  By default every spawned task can access every GPU allocated to the step.  If "verbose," is
specified before  <type>.  then print out GPU binding debug information to the stderr of the tasks. GPU binding is ignored if
there is only one task.
```

ʊ_ʊ

1 task per node, with 7 CPU cores and 1 GPU

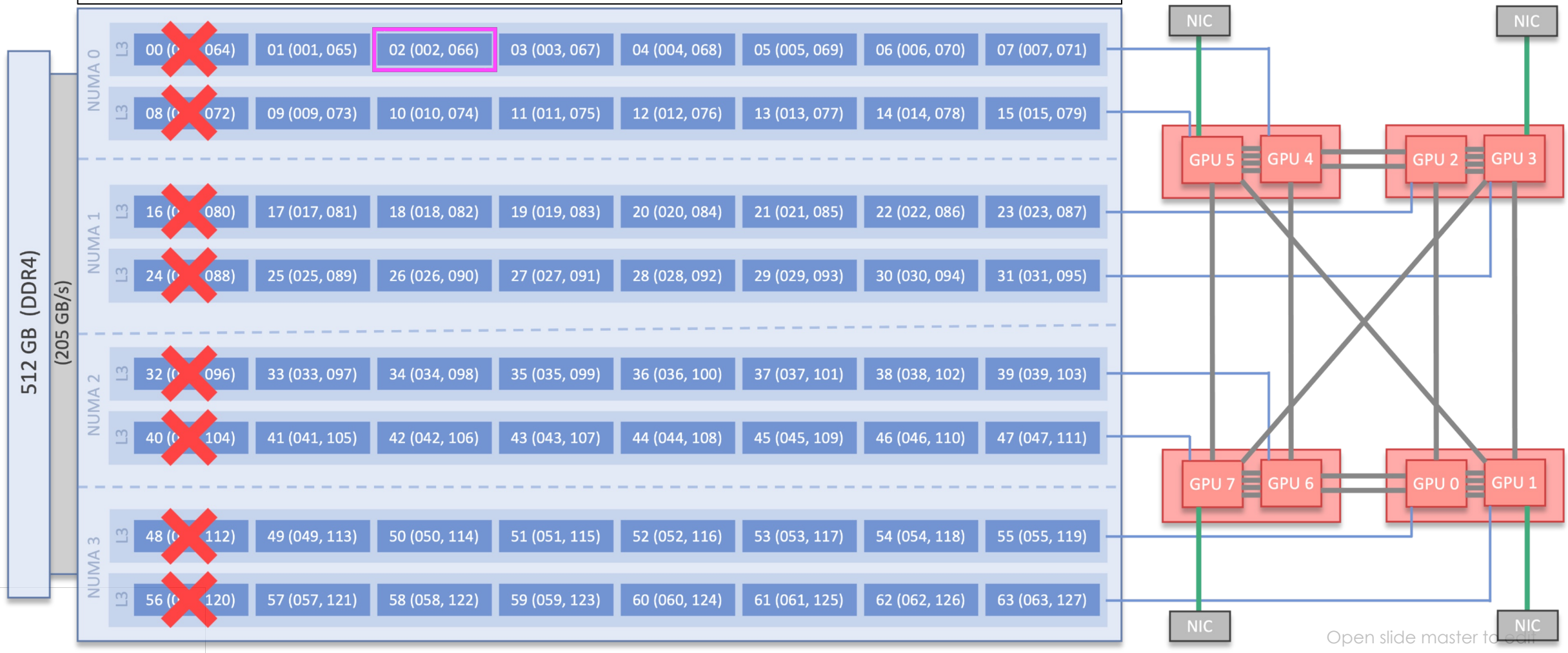


```
$ OMP_NUM_THREADS=1 srun -N2 -n2 -c7 --gpus-per-node=8 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 002 - Node frontier10227 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 001 - OMP 000 - HWT 002 - Node frontier10228 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
```

Incorrect mapping!

```
--gpu-bind=[verbose,]<type>

Bind tasks to specific GPUs.  By default every spawned task can access every GPU allocated to the step.  If "verbose," is
specified before  <type>.  then print out GPU binding debug information to the stderr of the tasks. GPU binding is ignored if
there is only one task. ʘ_ʘ
```

1 task per node, with 7 CPU cores and 1 GPU (on 2 nodes)



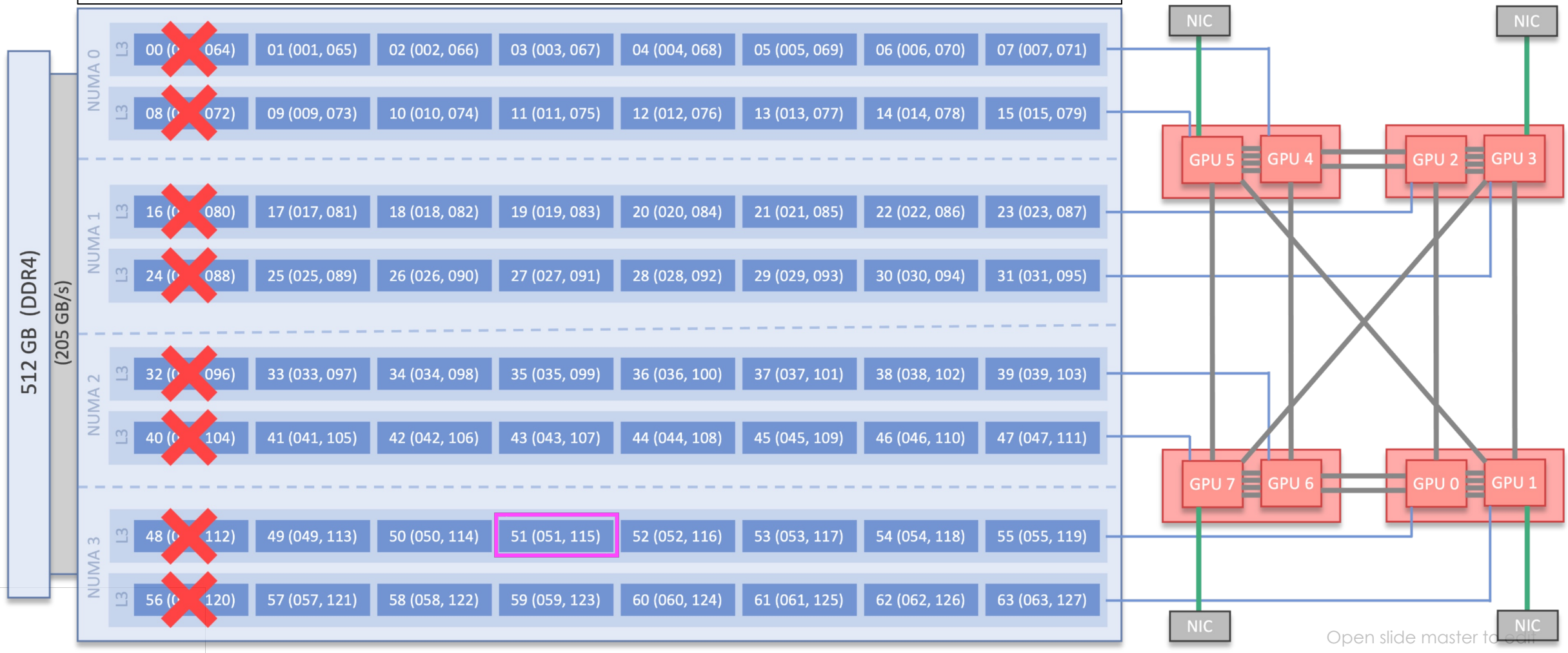
Open slide master to edit

```
$ OMP_NUM_THREADS=1 srun -N2 -n2 -c7 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 051 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 001 - OMP 000 - HWT 049 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
```

```
--gpu-bind=[verbose,]<type>

Bind tasks to specific GPUs.  By default every spawned task can access every GPU allocated to the step.  If "verbose," is
specified before  <type>,  then print out GPU binding debug information to the stderr of the tasks. GPU binding is ignored if
there is only one task.
```

✅ ʊ_ʊ

1 task per node, with 7 CPU cores and 1 GPU (on 2 nodes)



Distribution of tasks to node, sockets, and CPUs

--distribution=<value>[:<value>][:<value>][,{Pack|NoPack}]	Specifies the distribution of MPI ranks across compute nodes, sockets (L3 regions on Frontier), and cores, respectively. The default values are block:cyclic:cyclic * Can be used to refer to the default value

-m, --distribution=
{*|block|cyclic|arbitrary|plane=<size>}[:{*|block|cyclic|fcyclic}[:{*|block|cyclic|fcyclic}]][,{Pack|NoPack}]

This option controls the distribution of tasks to the nodes on which resources have been allocated, and the distribution of those resources to tasks for binding (task affinity).

The **first distribution method** (before the first ":") controls the distribution of tasks to nodes.

[default method for distributing tasks to nodes (block)]

The **second distribution method** (after the first ":") controls the distribution of allocated CPUs across sockets for binding to tasks.

[default method for distributing CPUs across sockets (cyclic)]

first distribution method (before first ":") controls **the distribution of tasks to nodes.**

[default method for distributing tasks to nodes (block)]

```
$ OMP_NUM_THREADS=1 srun -N2 -n16 -c7 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 009 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 017 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 041 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 006 - OMP 000 - HWT 049 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 057 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 008 - OMP 000 - HWT 001 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 009 - OMP 000 - HWT 009 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 010 - OMP 000 - HWT 017 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 011 - OMP 000 - HWT 025 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 012 - OMP 000 - HWT 033 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 013 - OMP 000 - HWT 041 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 014 - OMP 000 - HWT 049 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 015 - OMP 000 - HWT 057 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

block distribution (default)

```
$ OMP_NUM_THREADS=1 srun -N2 -n16 -c7 --gpus-per-task=1 --gpu-bind=closest -m cyclic ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 001 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 002 - OMP 000 - HWT 009 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 003 - OMP 000 - HWT 009 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 004 - OMP 000 - HWT 017 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 005 - OMP 000 - HWT 017 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 006 - OMP 000 - HWT 025 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 007 - OMP 000 - HWT 025 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 008 - OMP 000 - HWT 033 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 009 - OMP 000 - HWT 033 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 010 - OMP 000 - HWT 041 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 011 - OMP 000 - HWT 041 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 012 - OMP 000 - HWT 049 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 013 - OMP 000 - HWT 049 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 014 - OMP 000 - HWT 057 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 015 - OMP 000 - HWT 057 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

cyclic distribution

second distribution method (after first ":") controls **the distribution of allocated CPUs across sockets for binding to tasks.**

[default method for distributing CPUs across sockets (cyclic)]

```
$ OMP_NUM_THREADS=1 srun -N1 -n56 -c1 --gpu-bind=closest --ntasks-per-gpu=7 ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 009 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 017 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 041 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 006 - OMP 000 - HWT 049 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 057 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 008 - OMP 000 - HWT 002 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 009 - OMP 000 - HWT 010 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 010 - OMP 000 - HWT 018 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 011 - OMP 000 - HWT 026 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 012 - OMP 000 - HWT 034 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 013 - OMP 000 - HWT 042 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 014 - OMP 000 - HWT 050 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 015 - OMP 000 - HWT 058 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
...                                                                  ...
```

cyclic distribution (default)

```
$ OMP_NUM_THREADS=1 srun -N1 -n56 -c1 --gpu-bind=closest --ntasks-per-gpu=7 -m block:block ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 002 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 002 - OMP 000 - HWT 003 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 003 - OMP 000 - HWT 004 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 004 - OMP 000 - HWT 005 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 005 - OMP 000 - HWT 006 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 006 - OMP 000 - HWT 007 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 007 - OMP 000 - HWT 009 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 008 - OMP 000 - HWT 010 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 009 - OMP 000 - HWT 011 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 010 - OMP 000 - HWT 012 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 011 - OMP 000 - HWT 013 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 012 - OMP 000 - HWT 014 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 013 - OMP 000 - HWT 015 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 014 - OMP 000 - HWT 017 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 015 - OMP 000 - HWT 018 - Node frontier06352 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
...                                                                  ...
```

block distribution

second distribution method (after first ":") controls **the distribution of allocated CPUs across sockets for binding to tasks.**

[default method for distributing CPUs across sockets (cyclic)]

```
$ OMP_NUM_THREADS=1 srun -N1 -n16 -c3 --gpu-bind=closest --ntasks-per-gpu=2 ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 009 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 017 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 041 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 006 - OMP 000 - HWT 049 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 057 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 008 - OMP 000 - HWT 004 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 009 - OMP 000 - HWT 012 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 010 - OMP 000 - HWT 020 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 011 - OMP 000 - HWT 028 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 012 - OMP 000 - HWT 036 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 013 - OMP 000 - HWT 044 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 014 - OMP 000 - HWT 052 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 015 - OMP 000 - HWT 060 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

cyclic distribution (default)

```
$ OMP_NUM_THREADS=1 srun -N1 -n16 -c3 --gpu-bind=closest --ntasks-per-gpu=2 -m block:block ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 004 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 002 - OMP 000 - HWT 007 - Node frontier10227 - RT_GPU_ID 0,1 - GPU_ID 4,5 - Bus_ID d1,d6
MPI 003 - OMP 000 - HWT 011 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 004 - OMP 000 - HWT 017 - Node frontier10227 - RT_GPU_ID 0,1 - GPU_ID 2,5 - Bus_ID c9,d6
MPI 005 - OMP 000 - HWT 018 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 006 - OMP 000 - HWT 021 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 007 - OMP 000 - HWT 025 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 008 - OMP 000 - HWT 028 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 009 - OMP 000 - HWT 033 - Node frontier10227 - RT_GPU_ID 0,1 - GPU_ID 3,6 - Bus_ID ce,d9
MPI 010 - OMP 000 - HWT 035 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 011 - OMP 000 - HWT 041 - Node frontier10227 - RT_GPU_ID 0,1 - GPU_ID 6,7 - Bus_ID d9,de
MPI 012 - OMP 000 - HWT 042 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 013 - OMP 000 - HWT 045 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 014 - OMP 000 - HWT 049 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 015 - OMP 000 - HWT 052 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
```

block distribution

??

A brief aside…

Another way to check for resources available to each process

Prepend stdout lines with process ID

```
$ srun -l <srun-options> /bin/bash -c 'echo $(hostname) $(grep Cpus_allowed_list /proc/self/status) GPUS: $ROCR_VISIBLE_DEVICES' | sort
```

```
$ srun -l -N1 -n8 -c7 --gpus-per-task=1 --gpu-bind=closest /bin/bash -c 'echo $(hostname) $(grep Cpus_allowed_list /proc/self/status) GPUS: $ROCR_VISIBLE_DEVICES' | sort

0: crusher109 Cpus_allowed_list: 1-7 GPUS: 4
1: crusher109 Cpus_allowed_list: 9-15 GPUS: 5
2: crusher109 Cpus_allowed_list: 17-23 GPUS: 2
3: crusher109 Cpus_allowed_list: 25-31 GPUS: 3
4: crusher109 Cpus_allowed_list: 33-39 GPUS: 6
5: crusher109 Cpus_allowed_list: 41-47 GPUS: 7
6: crusher109 Cpus_allowed_list: 49-55 GPUS: 0
7: crusher109 Cpus_allowed_list: 57-63 GPUS: 1
```

Tells you which CPU cores and GPUs are available to each process, but not where they actually ran.

second distribution method (after first ":") controls **the distribution of allocated CPUs across sockets for binding to tasks.**

[default method for distributing CPUs across sockets (cyclic)]

```
$ OMP_NUM_THREADS=1 srun -N1 -n16 -c3 --gpu-bind=closest --ntasks-per-gpu=2 -m block:block ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 001 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 004 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 002 - OMP 000 - HWT 007 - Node frontier10227 - RT_GPU_ID 0,1 - GPU_ID 4,5 - Bus_ID d1,d6
MPI 003 - OMP 000 - HWT 011 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 004 - OMP 000 - HWT 017 - Node frontier10227 - RT_GPU_ID 0,1 - GPU_ID 2,5 - Bus_ID c9,d6
MPI 005 - OMP 000 - HWT 018 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 006 - OMP 000 - HWT 021 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 007 - OMP 000 - HWT 025 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 008 - OMP 000 - HWT 028 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 009 - OMP 000 - HWT 033 - Node frontier10227 - RT_GPU_ID 0,1 - GPU_ID 3,6 - Bus_ID ce,d9
MPI 010 - OMP 000 - HWT 035 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 011 - OMP 000 - HWT 041 - Node frontier10227 - RT_GPU_ID 0,1 - GPU_ID 6,7 - Bus_ID d9,de
MPI 012 - OMP 000 - HWT 042 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 013 - OMP 000 - HWT 045 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 014 - OMP 000 - HWT 049 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 015 - OMP 000 - HWT 052 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
```

```
$ OMP_NUM_THREADS=1 srun -l -N1 -n16 -c3 --gpu-bind=closest --ntasks-per-gpu=2 -m block:block /bin/bash -c 'echo $(hostname) $(grep
Cpus_allowed_list /proc/self/status) GPUS: $ROCR_VISIBLE_DEVICES' | sort -g
 0: frontier10227 Cpus_allowed_list: 1-3 GPUS: 4
 1: frontier10227 Cpus_allowed_list: 4-6 GPUS: 4
 2: frontier10227 Cpus_allowed_list: 7,9-10 GPUS: 4,5
 3: frontier10227 Cpus_allowed_list: 11-13 GPUS: 5
 4: frontier10227 Cpus_allowed_list: 14-15,17 GPUS: 2,5
 5: frontier10227 Cpus_allowed_list: 18-20 GPUS: 2
 6: frontier10227 Cpus_allowed_list: 21-23 GPUS: 2
 7: frontier10227 Cpus_allowed_list: 25-27 GPUS: 3
 8: frontier10227 Cpus_allowed_list: 28-30 GPUS: 3
 9: frontier10227 Cpus_allowed_list: 31,33-34 GPUS: 3,6
10: frontier10227 Cpus_allowed_list: 35-37 GPUS: 6
11: frontier10227 Cpus_allowed_list: 38-39,41 GPUS: 6,7
12: frontier10227 Cpus_allowed_list: 42-44 GPUS: 7
13: frontier10227 Cpus_allowed_list: 45-47 GPUS: 7
14: frontier10227 Cpus_allowed_list: 49-51 GPUS: 0
15: frontier10227 Cpus_allowed_list: 52-54 GPUS: 0
```

Caused by block distribution across L3s when not using all cores:

2 tasks (each with 3 cores) can fit in an L3, but there is still 1 more core from that L3 left over that will be given to the next task.

This can be resolved in different ways…

- Set core isolation to `-S16` so there are only 6 cores available per L3
- Remove core isolation (`-S0`) and use 4 threads per task.

second distribution method (after first ":") controls **the distribution of allocated CPUs across sockets for binding to tasks.**

[default method for distributing CPUs across sockets (cyclic)]

```
$ salloc -Astf016_frontier -N2 -t10 -S16
...
```

Make only needed cores available

```
$ OMP_NUM_THREADS=1 srun -N1 -n16 -c3 --gpu-bind=closest --ntasks-per-gpu=2 -m block:block ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 002 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 005 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 002 - OMP 000 - HWT 010 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 003 - OMP 000 - HWT 013 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 004 - OMP 000 - HWT 018 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 005 - OMP 000 - HWT 021 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 006 - OMP 000 - HWT 026 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 007 - OMP 000 - HWT 029 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 008 - OMP 000 - HWT 034 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 009 - OMP 000 - HWT 037 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 010 - OMP 000 - HWT 042 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 011 - OMP 000 - HWT 045 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 012 - OMP 000 - HWT 050 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 013 - OMP 000 - HWT 053 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 014 - OMP 000 - HWT 058 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 015 - OMP 000 - HWT 061 - Node frontier07742 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

Using `hello_jobstep`

```
$ OMP_NUM_THREADS=1 srun -l -N1 -n16 -c3 --gpu-bind=closest --ntasks-per-gpu=2 -m block:block /bin/bash -c 'echo $(hostname) $(grep
Cpus_allowed_list /proc/self/status) GPUS: $ROCR_VISIBLE_DEVICES' | sort -g
 0: frontier07742 Cpus_allowed_list: 2-4 GPUS: 4
 1: frontier07742 Cpus_allowed_list: 5-7 GPUS: 4
 2: frontier07742 Cpus_allowed_list: 10-12 GPUS: 5
 3: frontier07742 Cpus_allowed_list: 13-15 GPUS: 5
 4: frontier07742 Cpus_allowed_list: 18-20 GPUS: 2
 5: frontier07742 Cpus_allowed_list: 21-23 GPUS: 2
 6: frontier07742 Cpus_allowed_list: 26-28 GPUS: 3
 7: frontier07742 Cpus_allowed_list: 29-31 GPUS: 3
 8: frontier07742 Cpus_allowed_list: 34-36 GPUS: 6
 9: frontier07742 Cpus_allowed_list: 37-39 GPUS: 6
10: frontier07742 Cpus_allowed_list: 42-44 GPUS: 7
11: frontier07742 Cpus_allowed_list: 45-47 GPUS: 7
12: frontier07742 Cpus_allowed_list: 50-52 GPUS: 0
13: frontier07742 Cpus_allowed_list: 53-55 GPUS: 0
14: frontier07742 Cpus_allowed_list: 58-60 GPUS: 1
15: frontier07742 Cpus_allowed_list: 61-63 GPUS: 1
```

Using bash command

Using all CPU cores on a Crusher/Frontier node

```
 -S, --core-spec=<num>

Count  of  specialized  cores per node reserved by the job for system operations and not used by the application.
```

```
$ salloc -Astf016_frontier -N1 -t120 -S0
salloc: Pending job allocation 1273083
salloc: job 1273083 queued and waiting for resources
salloc: job 1273083 has been allocated resources
salloc: Granted job allocation 1273083
salloc: Waiting for resource configuration
salloc: Nodes frontier09076 are ready for job
```

`-S` flag is used at *job allocation time*

```
$ OMP_NUM_THREADS=8 srun -N1 -n8 -c8 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 000 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 000 - OMP 001 - HWT 001 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 000 - OMP 002 - HWT 002 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 000 - OMP 003 - HWT 003 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 000 - OMP 004 - HWT 004 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 000 - OMP 005 - HWT 005 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 000 - OMP 006 - HWT 006 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 000 - OMP 007 - HWT 007 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 008 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 001 - OMP 001 - HWT 009 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 001 - OMP 002 - HWT 010 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 001 - OMP 003 - HWT 011 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 001 - OMP 004 - HWT 012 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 001 - OMP 005 - HWT 013 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 001 - OMP 006 - HWT 014 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 001 - OMP 007 - HWT 015 - Node frontier09076 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
...
...
```

But remember that core 0 will have lower performance than other cores on each node.

Multiple job steps on a single node

Run 8 concurrent job steps on a node

This will only work correctly for 1-6 cores when –S8 is set.

Not a problem for –S0

```
$ cat multiple_jobsteps.sh
#!/bin/bash

for idx in {1..8};

    do
        date

        srun -u --gpus-per-task=1 --gpu-bind=closest -N1 -n1 -c6 \
             /bin/bash -c 'echo $(hostname) $(grep Cpus_allowed_list /proc/self/status) GPUS: $ROCR_VISIBLE_DEVICES; sleep 10' &

        sleep 1
    done

wait
```

Slurm needs a small pause between job step submissions.

To "pretend" the job step is doing something.

```
$ ./multiple_jobsteps.sh
Mon 15 May 2023 01:38:19 PM EDT
frontier01357 Cpus_allowed_list: 49-54 GPUS: 0

Mon 15 May 2023 01:38:20 PM EDT
frontier01357 Cpus_allowed_list: 57-62 GPUS: 1

Mon 15 May 2023 01:38:21 PM EDT
frontier01357 Cpus_allowed_list: 17-22 GPUS: 2

Mon 15 May 2023 01:38:22 PM EDT
frontier01357 Cpus_allowed_list: 25-30 GPUS: 3

Mon 15 May 2023 01:38:23 PM EDT
frontier01357 Cpus_allowed_list: 1-6 GPUS: 4

Mon 15 May 2023 01:38:24 PM EDT
frontier01357 Cpus_allowed_list: 9-14 GPUS: 5

Mon 15 May 2023 01:38:25 PM EDT
frontier01357 Cpus_allowed_list: 33-38 GPUS: 6

Mon 15 May 2023 01:38:26 PM EDT
frontier01357 Cpus_allowed_list: 41-46 GPUS: 7
```

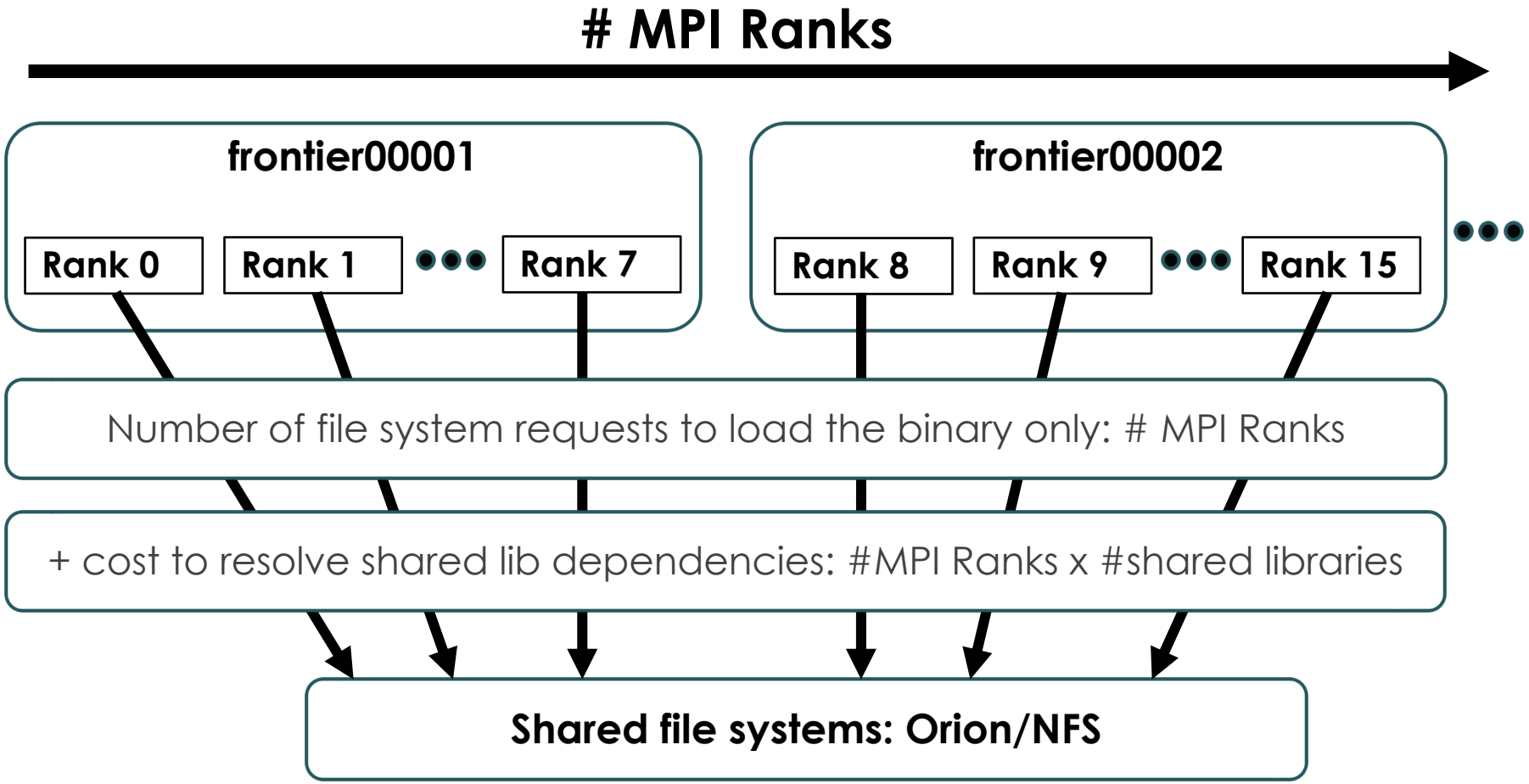
Using `sbcast` to improve job initialization at scale.

Thank you to Nick Hagerty for putting the slides in this section together!

https://docs.olcf.ornl.gov/systems/frontier_user_guide.html#tips-for-launching-at-scale

Cost of Binary and Library Access when Running at Scale

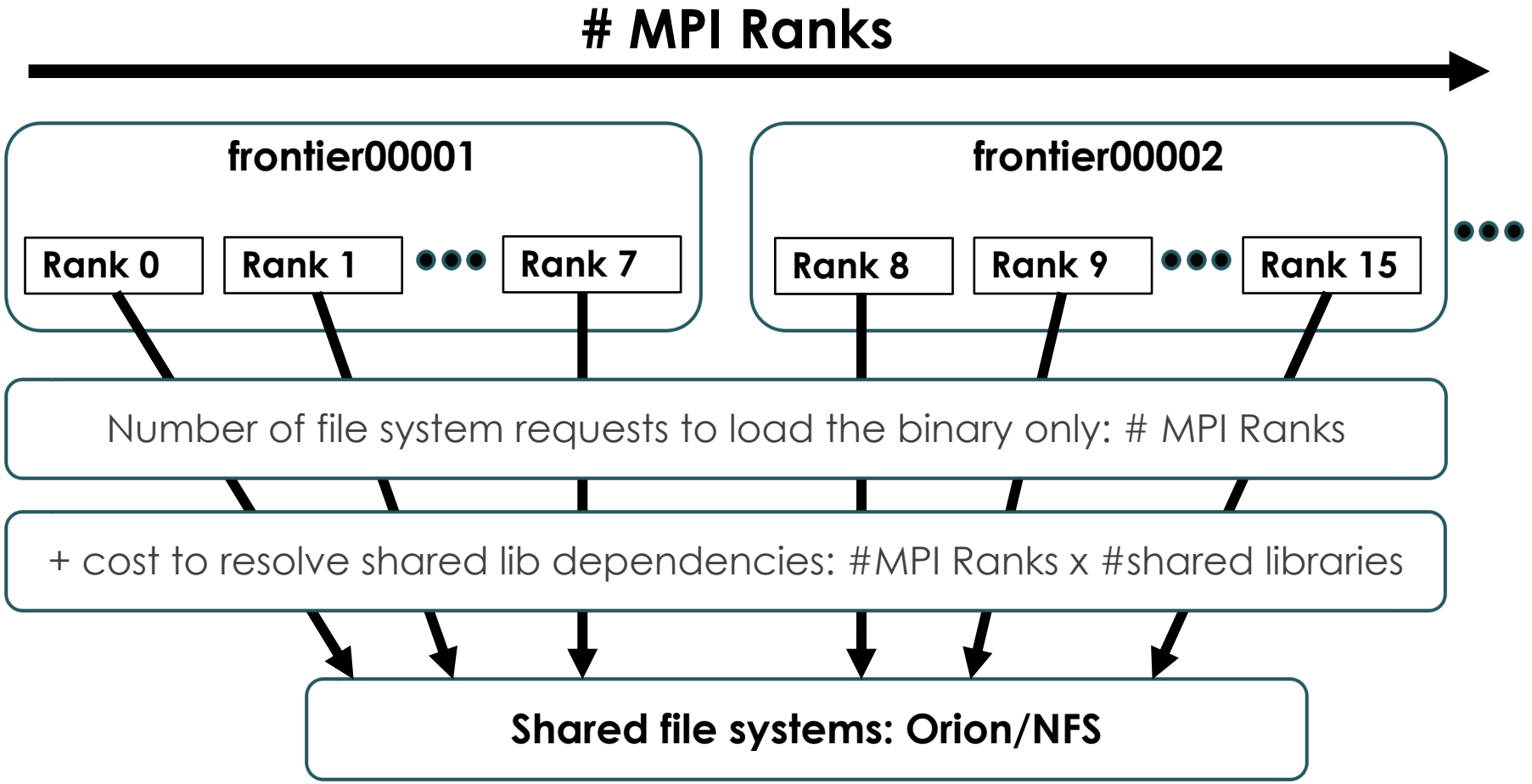
By default, each rank reads from the file system independently:



# of File System Requests = # MPI Ranks + (# MPI Ranks * # Shared Libs)

Cost of Binary and Library Access when Running at Scale

By default, each rank reads from the file system independently:



For large jobs, can lead to:

- Poor (longer) launch times
- Inconsistent launch behavior
- Increased occurrences of timeouts at launch time
  - → Wasted node-hours
  - → Wasted time waiting in queue

# of File System Requests = # MPI Ranks + (# MPI Ranks * # Shared Libs)

Cost of Binary and Library Access when Running at Scale

(Dealing with timeouts during initialization)

From `man pmi`

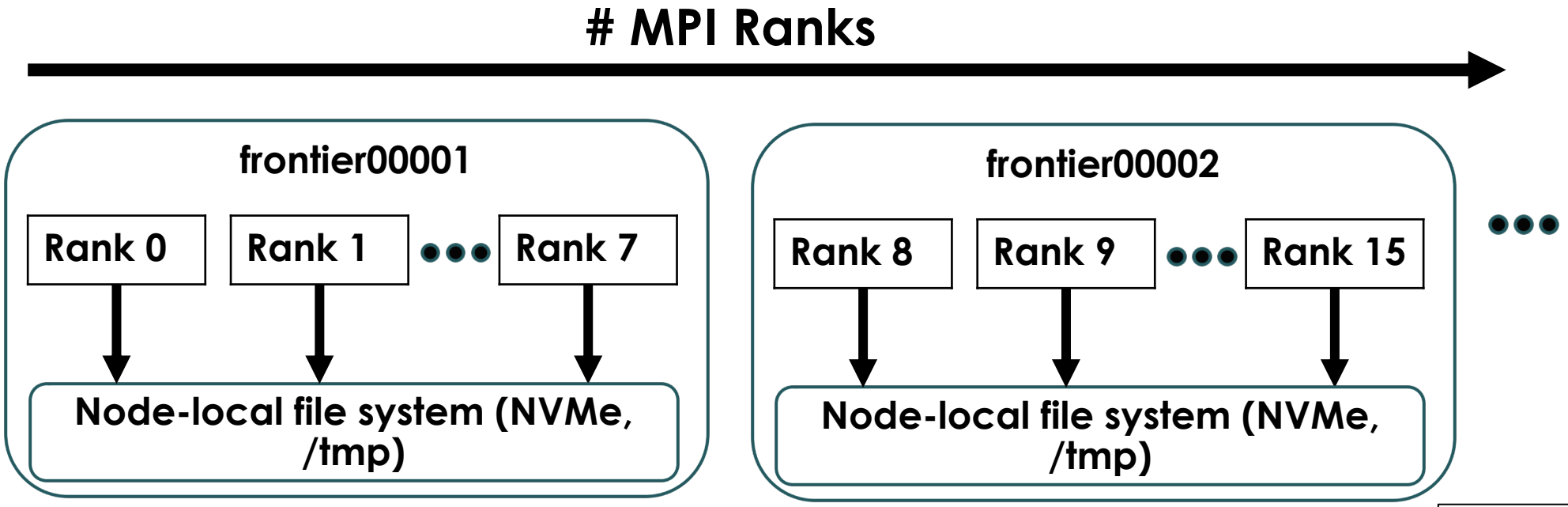
```
PMI_MMAP_SYNC_WAIT_TIME
    This environment variable sets the length of seconds PMI waits before returning a timeout error
    during process launch.

    By default, PMI expects all requested processes to be launched on any given node within 180 seconds
    of each other.

    If the time span is longer than the value specified by PMI_MMAP_SYNC_WAIT_TIME, PMI assumes an error
    has occurred, and a timeout error is returned.

    Note that each node has its own timer and does not depend on total ranks in the job.

    Default: 180
```

You can increase the value of this environment variable, but if you "set it and forget it" in your batch script, **it could potentially lead to wasted node-hours.**

E.g., say you set it to 15 minutes. Then one day the FS is unhappy and that limit is reached in your 4096-node (or larger job). The job times out and you've wasted 1024 node-hours (~5X the cost if running with the default of 3-minute timeouts).

Cost of Binary and Library Access when Running at Scale

Instead, we can pre-stage binary & libraries in node-local storage using `sbcast`:



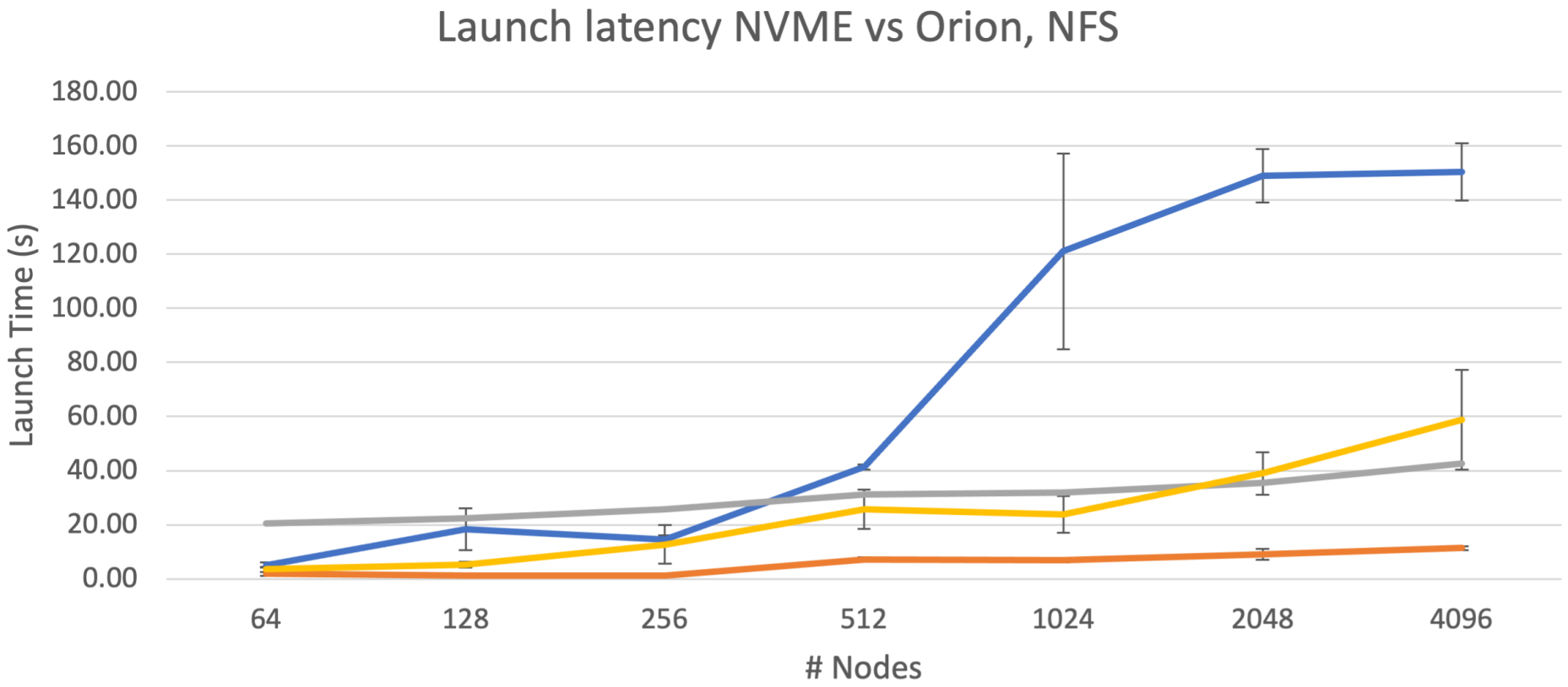
Shared file systems: Orion/NFS

- Improve launch times
- More consistent launch behavior
- Reduce occurrences of launch timeouts
- Save node-hours

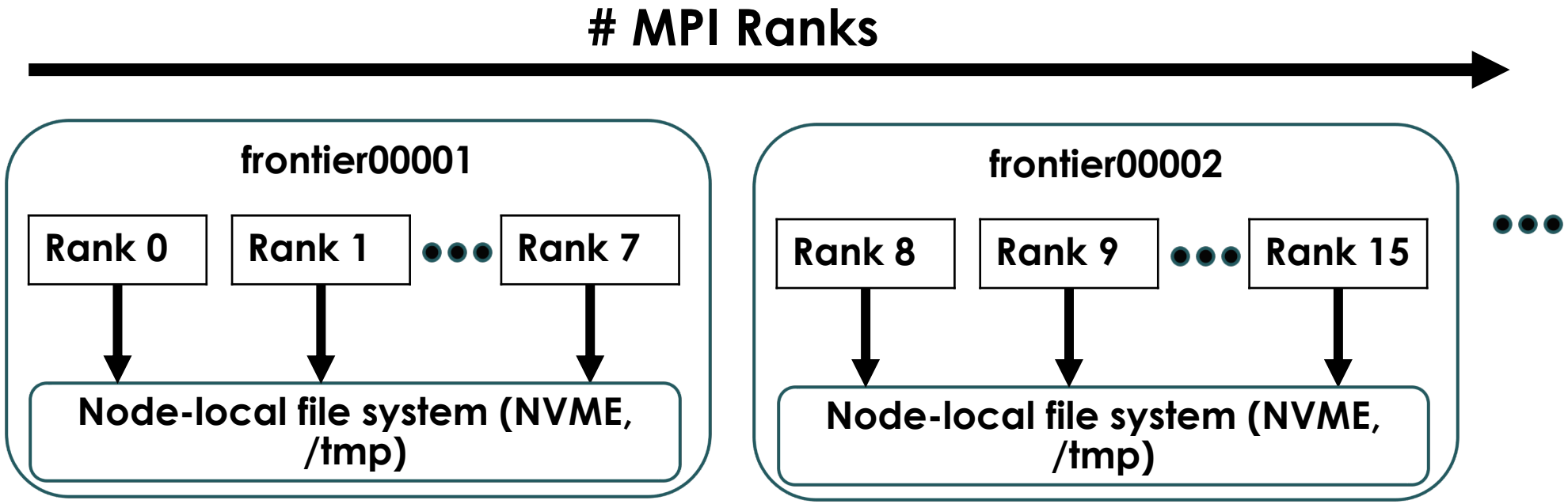
Cost of Binary and Library Access when Running at Scale

Application: DFTFE (6 libraries accessed from NFS, Orion, or NVMe)

Each library in its own directory and had its own entry in `LD_LIBRARY_PATH`



Cost of Binary and Library Access when Running at Scale



Benefits to using `sbcast`:

- Improves application launch/initialization times
- Gives more consistent/predictable launch-time behavior
- Reduces occurrence of timeouts at launch time
- Allows early detection of failed job initialization (minimize wasted node-hours)
- Reduces load and dependence on shared file systems

Using `sbcast` to improve job initialization

How to use `sbcast` to pre-stage binary & libraries:

Line	Bash code in Job Script
1	`# Must use ``-C nvme`` in SBATCH headers to use NVME drive`
2	`sbcast --send-libs --exclude=NONE -pf ${exe} /mnt/bb/$USER`
3	
4	`# if SBCAST fails, abort`
5	`if [ ! "$?" == 0 ]; then echo "SBCAST failed"; exit 1; fi`
6	
7	`export LD_LIBRARY_PATH="/mnt/bb/$USER/${exe}_libs"`
8	
9	`# libfabric dlopen's several libraries. Add those paths to end of LD_LIBRARY_PATH:`
10	`export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:$(pkg-config --variable=libdir libfabric)"`
11	
12	`# Patch: libfabric dlopen's `libhsa-runtime64.so`, but only `libhsa-runtime64.so.1` is sbcast`
13	`# Add sym-link on each node for `libhsa-runtime64.so``
14	`srun -N ${SLURM_NNODES} -n ${SLURM_NNODES} --ntasks-per-node=1 --label \`
15	`    -D /mnt/bb/$USER/${exe}_libs`
16	`    if [ -f libhsa-runtime64.so.1 ]; then ln -s libhsa-runtime64.so.1 libhsa-runtime64.so; fi`

Using `sbcast` to improve job initialization

How to use `sbcast` to pre-stage binary & libraries:

Broadcast files from the first node in the allocation to all nodes

Line	Bash code in Job Script
1	`# Must use ``-C nvme`` in SBATCH headers to use NVME drive`
2	`sbcast --send-libs --exclude=NONE -pf ${exe} /mnt/bb/$USER`
3	
4	`# if SBCAST fails, abort`
5	`if [ ! "$?" == 0 ]; then echo "SBCAST failed"; exit 1; fi`
6	
7	`export LD_LIBRARY_PATH="/mnt/bb/$USER/${exe}_libs"`
8	
9	`# libfabric dlopen's several libraries. Add those paths to end of LD_LIBRARY_PATH:`
10	`export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:$(pkg-config --variable=libdir libfabric)"`
11	
12	`# Patch: libfabric dlopen's `libhsa-runtime64.so`, but only `libhsa-runtime64.so.1` is sbcast`
13	`# Add sym-link on each node for `libhsa-runtime64.so``
14	`srun -N ${SLURM_NNODES} -n ${SLURM_NNODES} --ntasks-per-node=1 --label \`
15	`    -D /mnt/bb/$USER/${exe}_libs`
16	`    if [ -f libhsa-runtime64.so.1 ]; then ln -s libhsa-runtime64.so.1 libhsa-runtime64.so; fi`

Using `sbcast` to improve job initialization

How to use `sbcast` to pre-stage binary & libraries:

Line	Bash code in Job Script
1	`# Must use ``-C nvme`` in SBATCH headers to`
2	`sbcast --send-libs --exclude=NONE -pf ${exe`
3	
4	`# if SBCAST fails, abort`
5	`if [ ! "$?" == 0 ]; then echo "SBCAST failed"; exit 1; fi`
6	
7	`export LD_LIBRARY_PATH="/mnt/bb/$USER/${exe}_libs"`
8	
9	`# libfabric dlopen's several libraries. Add those paths to end of LD_LIBRARY_PATH:`
10	`export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:$(pkg-config --variable=libdir libfabric)"`
11	
12	`# Patch: libfabric dlopen's `libhsa-runtime64.so`, but only `libhsa-runtime64.so.1` is sbcast`
13	`# Add sym-link on each node for `libhsa-runtime64.so``
14	`srun -N ${SLURM_NNODES} -n ${SLURM_NNODES} --ntasks-per-node=1 --label \`
15	`    -D /mnt/bb/$USER/${exe}_libs`
16	`    if [ -f libhsa-runtime64.so.1 ]; then ln -s libhsa-runtime64.so.1 libhsa-runtime64.so; fi`

SBCAST sends files chunk-by-chunk, so a failed SBCAST may leave partial libraries on some nodes

Using `sbcast` to improve job initialization

How to use `sbcast` to pre-stage binary & libraries:

Line	Bash code in Job Script
1	`# Must use ``-C nvme`` in SBATCH headers to use NVME drive`
2	`sbcast --send-libs --exclude=NONE -pf ${exe`
3	
4	`# if SBCAST fails, abort`
5	`if [ ! "$?" == 0 ]; then echo "SBCAST faile`
6	
7	`export LD_LIBRARY_PATH="/mnt/bb/$USER/${exe}_libs"`
8	
9	`# libfabric dlopen's several libraries. Add those paths to end of LD_LIBRARY_PATH:`
10	`export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:$(pkg-config --variable=libdir libfabric)"`
11	
12	`# Patch: libfabric dlopen's `libhsa-runtime64.so`, but only `libhsa-runtime64.so.1` is sbcast`
13	`# Add sym-link on each node for `libhsa-runtime64.so``
14	`srun -N ${SLURM_NNODES} -n ${SLURM_NNODES} --ntasks-per-node=1 --label \`
15	`    -D /mnt/bb/$USER/${exe}_libs`
16	`    if [ -f libhsa-runtime64.so.1 ]; then ln -s libhsa-runtime64.so.1 libhsa-runtime64.so; fi`

All libraries are placed in an `${exe}_libs` directory in the directory the executable is in

Using `sbcast` to improve job initialization

How to use `sbcast` to pre-stage binary & libraries:

Line	Bash code in Job Script
1	`# Must use ``-C nvme`` in SBATCH headers to use NVME drive`
2	`sbcast --send-libs --exclude=NONE -pf ${exe} /mnt/bb/$USER`
3	
4	`# if SBCAST fails, abort`
5	`if [ ! "$?" == 0 ]; then echo "SBCAST faile`
6	
7	`export LD_LIBRARY_PATH="/mnt/bb/$USER/${exe`
8	
9	`# libfabric dlopen's several libraries. Add those paths to end of LD_LIBRARY_PATH:`
10	`export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:$(pkg-config --variable=libdir libfabric)"`
11	
12	`# Patch: libfabric dlopen's `libhsa-runtime64.so`, but only `libhsa-runtime64.so.1` is sbcast`
13	`# Add sym-link on each node for `libhsa-runtime64.so``
14	`srun -N ${SLURM_NNODES} -n ${SLURM_NNODES} --ntasks-per-node=1 --label \`
15	`    -D /mnt/bb/$USER/${exe}_libs`
16	`    if [ -f libhsa-runtime64.so.1 ]; then ln -s libhsa-runtime64.so.1 libhsa-runtime64.so; fi`

sbcast cannot detect `dlopen` calls, so a few patches are required:

Using `sbcast` to improve job initialization

Caveats:

- As mentioned, `dlopen` calls are not detected by `sbcast`. This requires additional effort from the user to place `dlopen`'d libraries correctly
- Any RPATH paths will still be used
  - Spack often adds library paths to RPATH
  - make and cmake *sometimes* do this, check your build to confirm
- libfabric is not the only libraries that require patches
  - ROCBLAS library requires `ROCBLAS_TENSILE_LIBPATH` to be set
  - **Please submit a ticket to OLCF Help Desk to receive the most updated list of patches required for your specific application**

Questions?

papatheodore@ornl.gov

OAK RIDGE National Laboratory

OAK RIDGE National Laboratory

Bonus Slides

Slurm Tips – Flags and Completed Jobs

- `-u` flag gives unbuffered output, which can be helpful when debugging.
- `-l` flag prepends the task ID to lines of stdout.

To show completed jobs in a specific time period, specify a start (`-S`) and end (`-E`) time

- The default time window depends on other options (see `man sacct`)

```
$ sacct --user=tpapathe -S 2022-05-05T00:01 -E 2022-05-05T23:59 | awk '$0 !~ /\./'

JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
107814       interacti+      batch     stf016        128  COMPLETED      0:0
107836       interacti+      batch     stf016        128  COMPLETED      0:0
107849       hello_job+      batch     stf016        128  COMPLETED      0:0
107858       hello_job+      batch     stf016        256  COMPLETED      0:0
107866       hello_job+      batch     stf016        256  COMPLETED      0:0
107867       hello_job+      batch     stf016        256 CANCELLED+      0:0
107868       hello_job+      batch     stf016        256  COMPLETED      0:0
107965       MatrixTra+      batch     stf016        128  COMPLETED      0:0
107966          rocprof      batch     stf016        128     FAILED      6:0
107969          rocprof      batch     stf016        128     FAILED      6:0
```

Then you can drill into more details about each job with the `-j` flag and customized output.

Slurm Tips – Capture Job Information

```
$ cat submit.sh
#!/bin/bash

#SBATCH -A stf016
#SBATCH -J job-name
#SBATCH -o %x-%j.out
#SBATCH -t 5
#SBATCH -p batch
#SBATCH -N 1

scontrol show job ${SLURM_JOBID}

srun -n1 ./p2p/p2p --correct

sacct -j ${SLURM_JOBID} -o jobid%20,Start%20,elapsed%20
```

```
$ cat job-name-108121.out

JobId=108121 JobName=job-name
   UserId=tpapathe(5987) GroupId=tpapathe(8654) MCS_label=N/A
   Priority=200 Nice=0 Account=stf016 QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=00:05:00 TimeMin=N/A
   SubmitTime=2022-05-05T17:34:51 EligibleTime=2022-05-05T17:34:51
   AccrueTime=2022-05-05T17:34:51
   StartTime=2022-05-05T17:35:03 EndTime=2022-05-05T17:40:03 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2022-05-05T17:35:03 Scheduler=Backfill
   Partition=batch AllocNode:Sid=login2:106343
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=crusher048
   BatchHost=crusher048
   NumNodes=1 NumCPUs=128 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:1
   TRES=cpu=128,node=1,billing=128
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
   Command=/autofs/nccs-svm1_home1/tpapathe/capture-job-info/submit.sh
   WorkDir=/autofs/nccs-svm1_home1/tpapathe/capture-job-info
   StdErr=/autofs/nccs-svm1_home1/tpapathe/capture-job-info/job-name-108121.out
   StdIn=/dev/null
   StdOut=/autofs/nccs-svm1_home1/tpapathe/capture-job-info/job-name-108121.out


...<APPLICATION OUTPUT>...


 JobID                Start              Elapsed
-------------------- ------------------- ----------
               108121 2022-05-05T17:35:03   00:02:34
         108121.batch 2022-05-05T17:35:03   00:02:34
        108121.extern 2022-05-05T17:35:03   00:02:34
            108121.0 2022-05-05T17:35:04   00:02:32
```

Also add…

`module -t list`
(lists currently-loaded environment modules)

`ldd ./<your-executable>`
(prints shared object dependencies)

Submitting Helpful User Support Tickets

Submit support tickets to help@olcf.ornl.gov

Typical questions we ask users for:

- Job ID
- List of modules loaded when compiling and running
- Batch script used to launch the job
- Job stdout and stderr
- Output from `ldd` run on executable

Notice these are all things that are included from the previous slide 😉

Other helpful things to include in your tickets

- Full errors that are generated
- Has the program run successfully before?
  - If so, did you change anything since then?
- Programming models used in your code