OAK RIDGE National Laboratory

Slurm on Frontier

Nick Hagerty

HPC Engineer

System Acceptance & User Environment Group

Oak Ridge Leadership Computing Facility (OLCF)

Oak Ridge National Laboratory (ORNL)

August 23, 2023

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

U.S. DEPARTMENT OF ENERGY

Slurm on Frontier and Crusher

Slurm provides 3 ways of submitting and launching jobs on Frontier/Crusher 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>`

**You can also do `sbatch --wrap '<executable commands>'` to run a command without a job 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

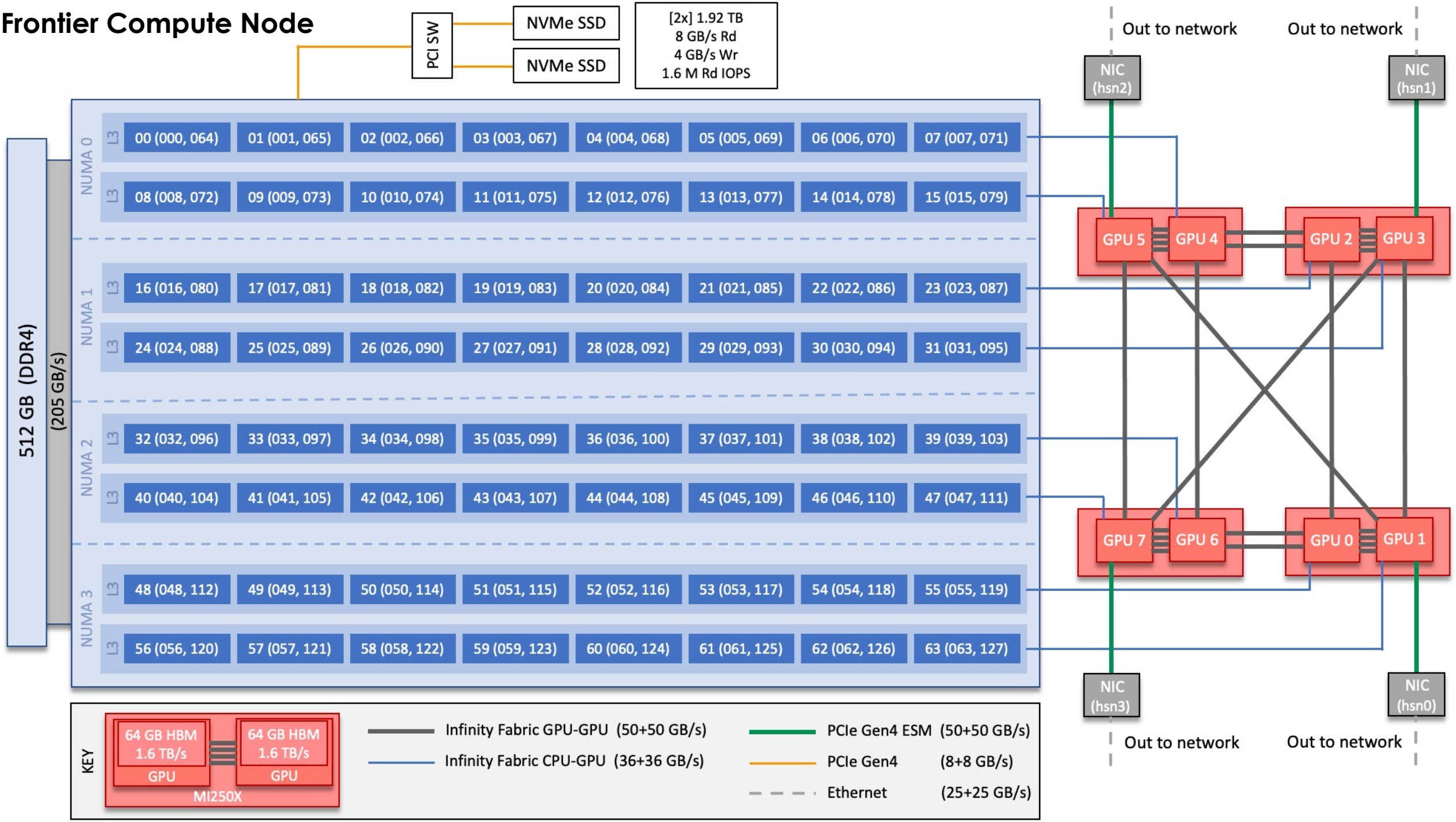
`-A <project_id>`	Project ID to charge
`-J <job_name>`	Name of job
`-p <partition>`	Partition / batch queue
`-t <time>`	Wall clock time <HH:MM:SS> (or you can just give minutes)
`-N <number_of_nodes>`	Number of compute nodes
`-o <file_name>`	Standard output file name
`-e <file_name>`	Standard error file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

Common Slurm Submission Options:

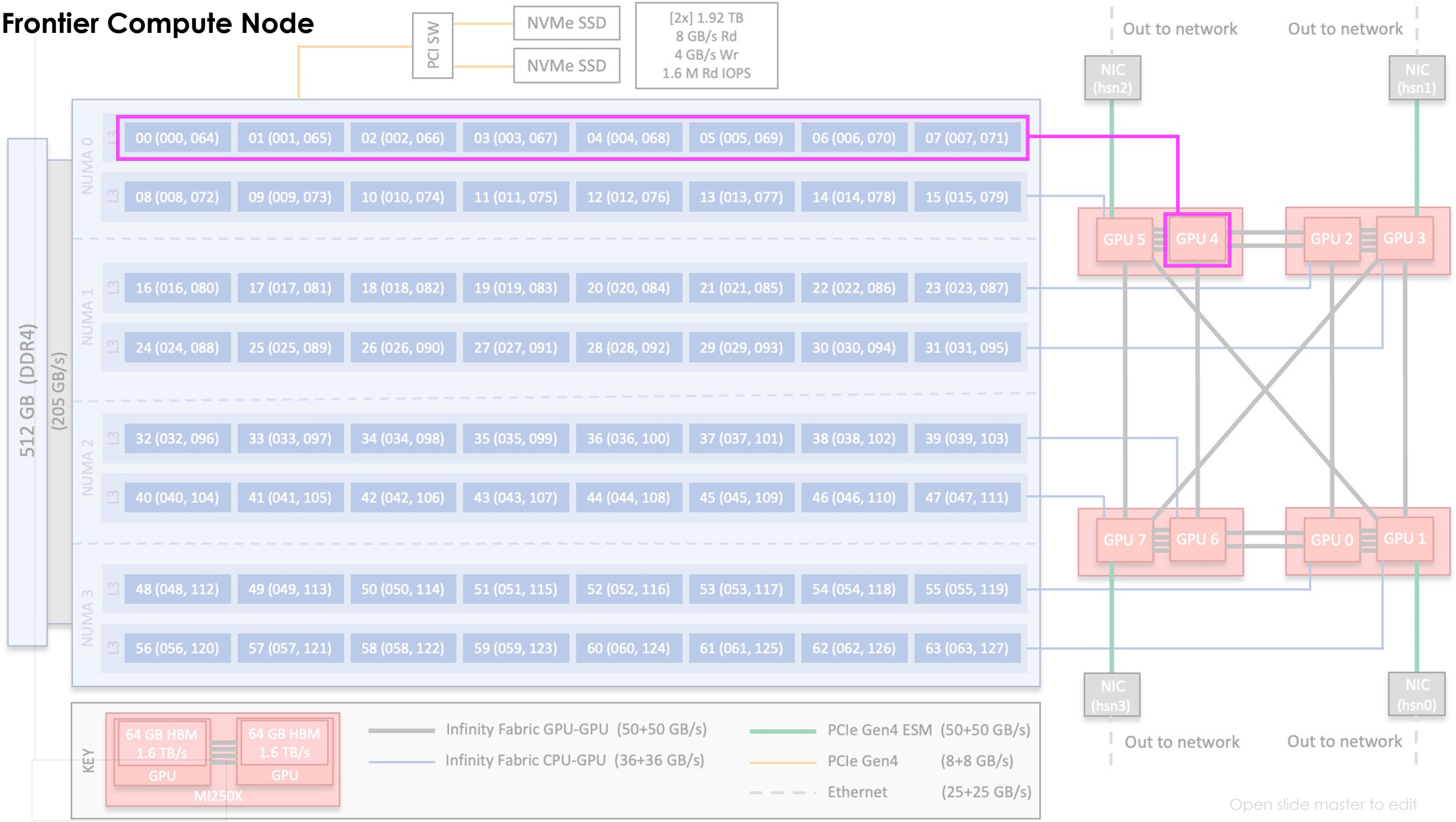
see individual `man` pages for more info

`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 Slurm database.
`scancel`	Used to signal or cancel jobs or job steps.
`scontrol`	Used to view or modify active job configuration.

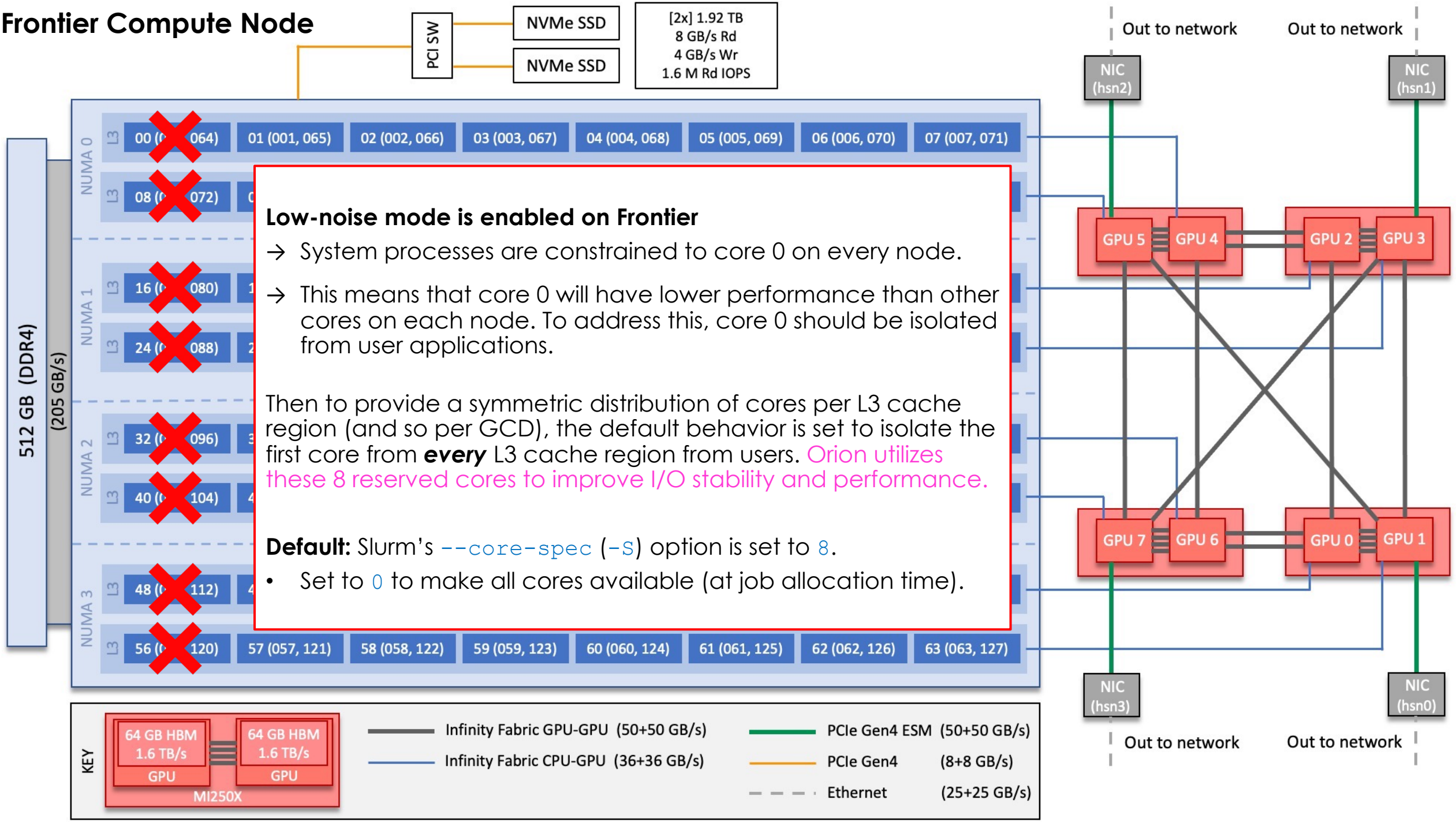
Frontier Compute Node



Frontier Compute Node



Frontier Compute Node



Low-noise mode is enabled on Frontier

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

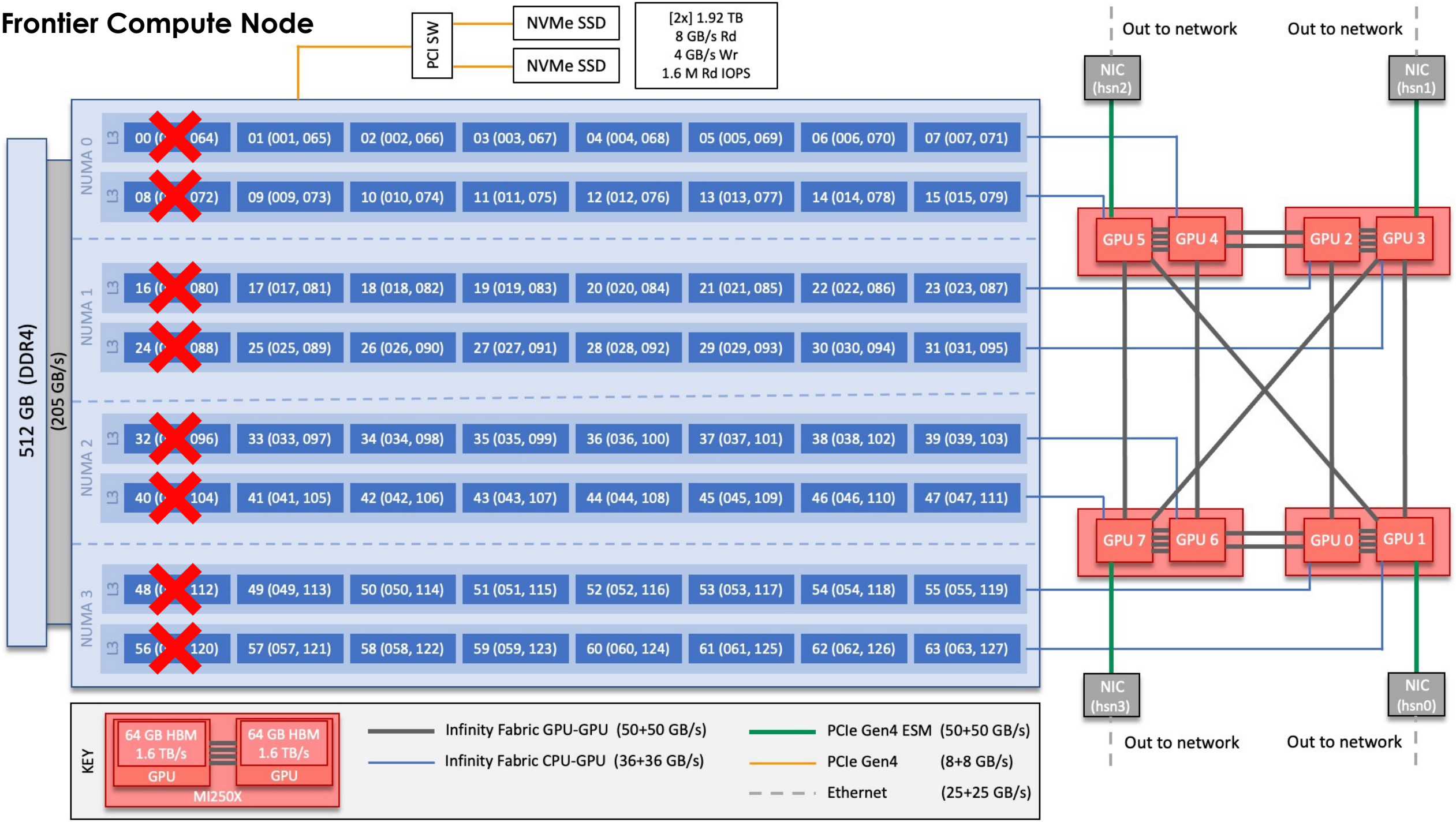
→ This means that core 0 will have lower performance than other cores on each node. To address this, core 0 should be isolated from user applications.

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. Orion utilizes these 8 reserved cores to improve I/O stability and performance.

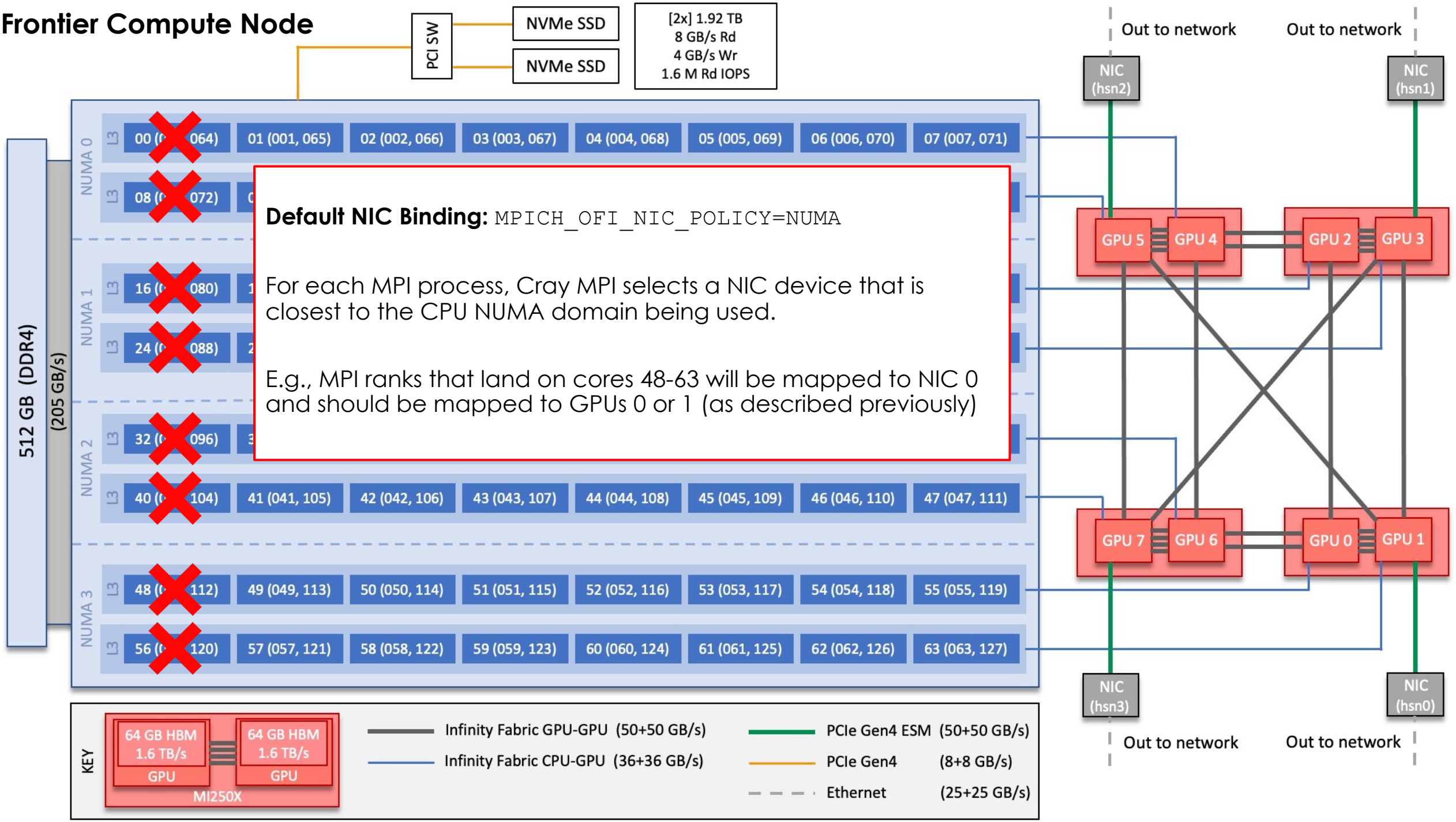
Default: Slurm's `--core-spec` (`-S`) option is set to `8`.

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

Frontier Compute Node



Frontier Compute Node



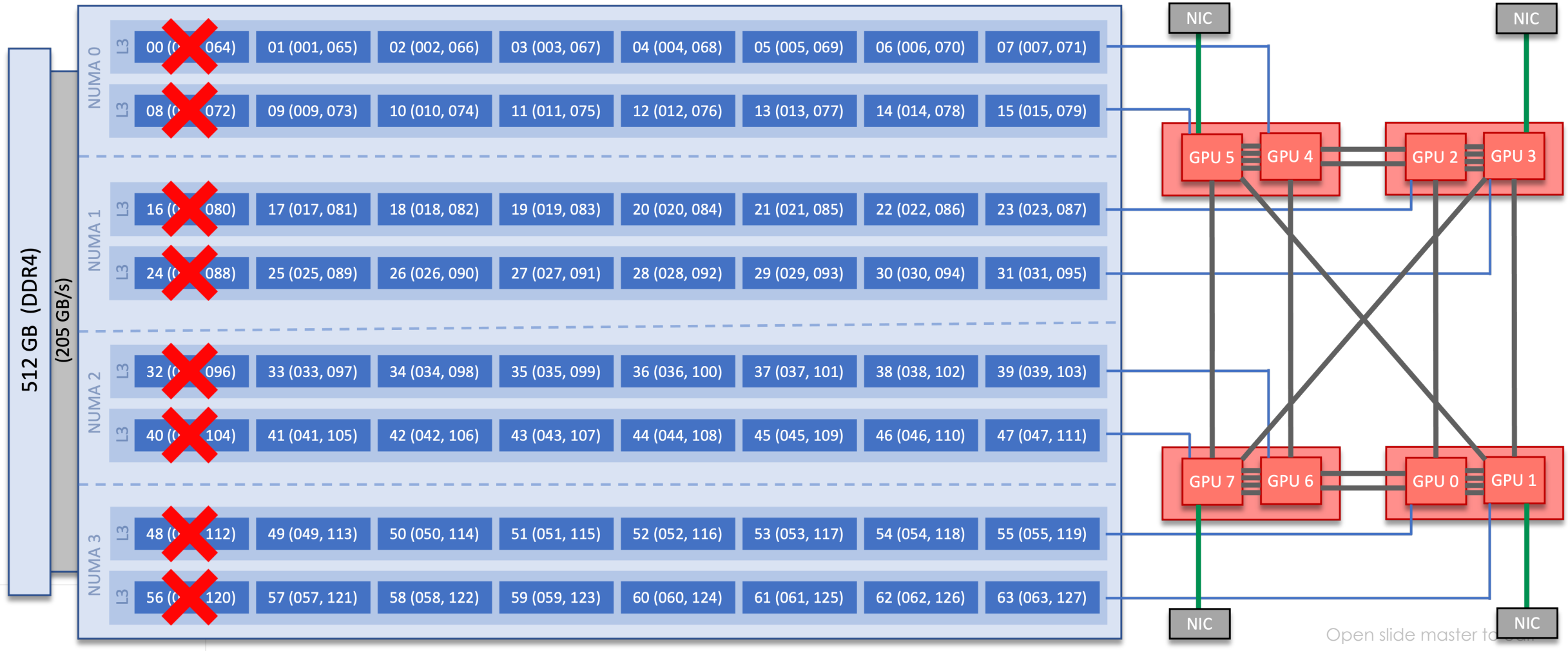
OAK RIDGE National Laboratory

Examples

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

Clone the repo and compile:

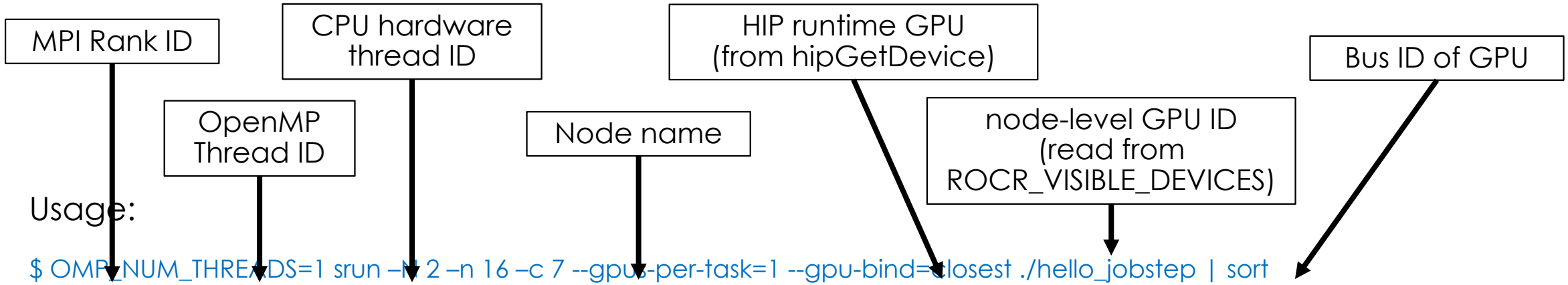
```
$ git clone https://code.ornl.gov/olcf/hello_jobstep.git
$ cd hello_jobstep
$ module load craype-accel-amd-gfx90a rocm
$ make
```

Usage:

```
$ OMP_NUM_THREADS=1 srun –N 2 –n 16 –c 7 --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
```

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

Used to test process, thread, and GPU binding



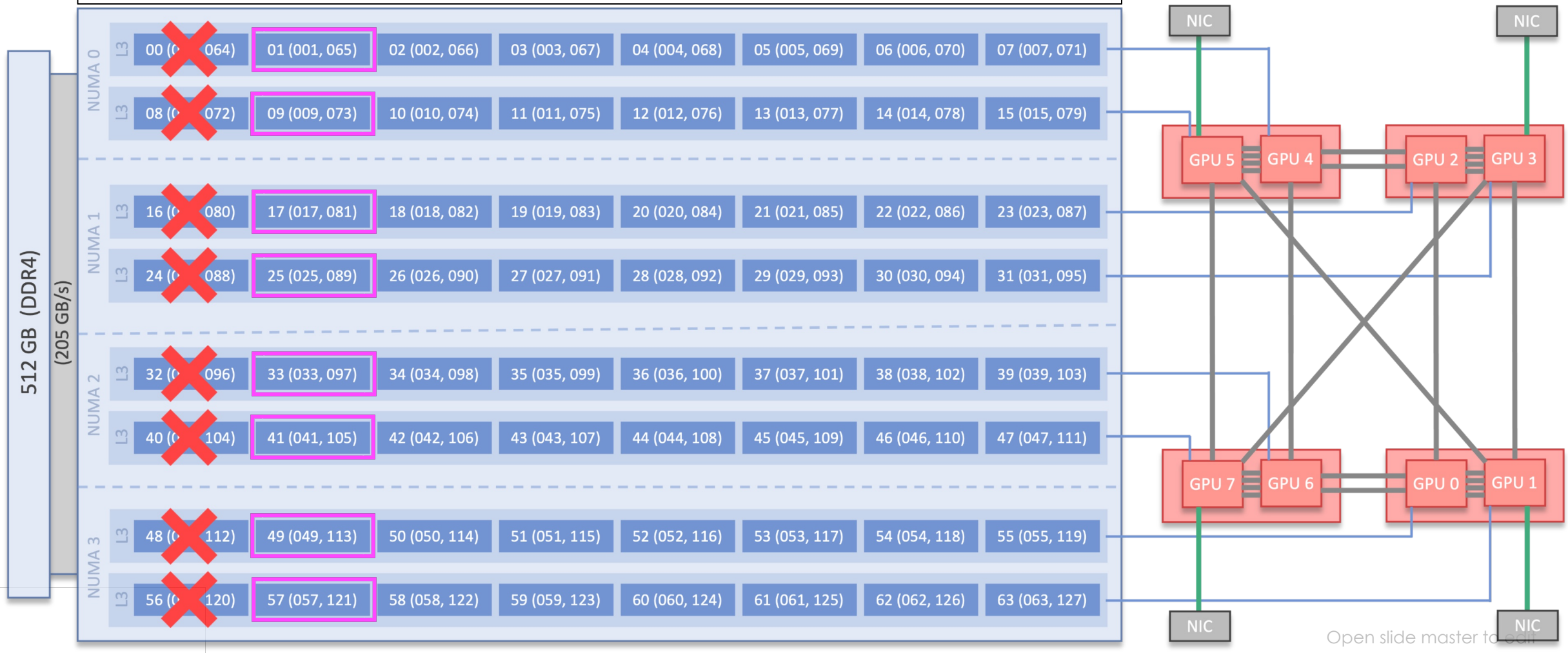
Usage:

```
$ OMP_NUM_THREADS=1 srun –N 2 –n 16 –c 7 --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
```

```
$ 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
```

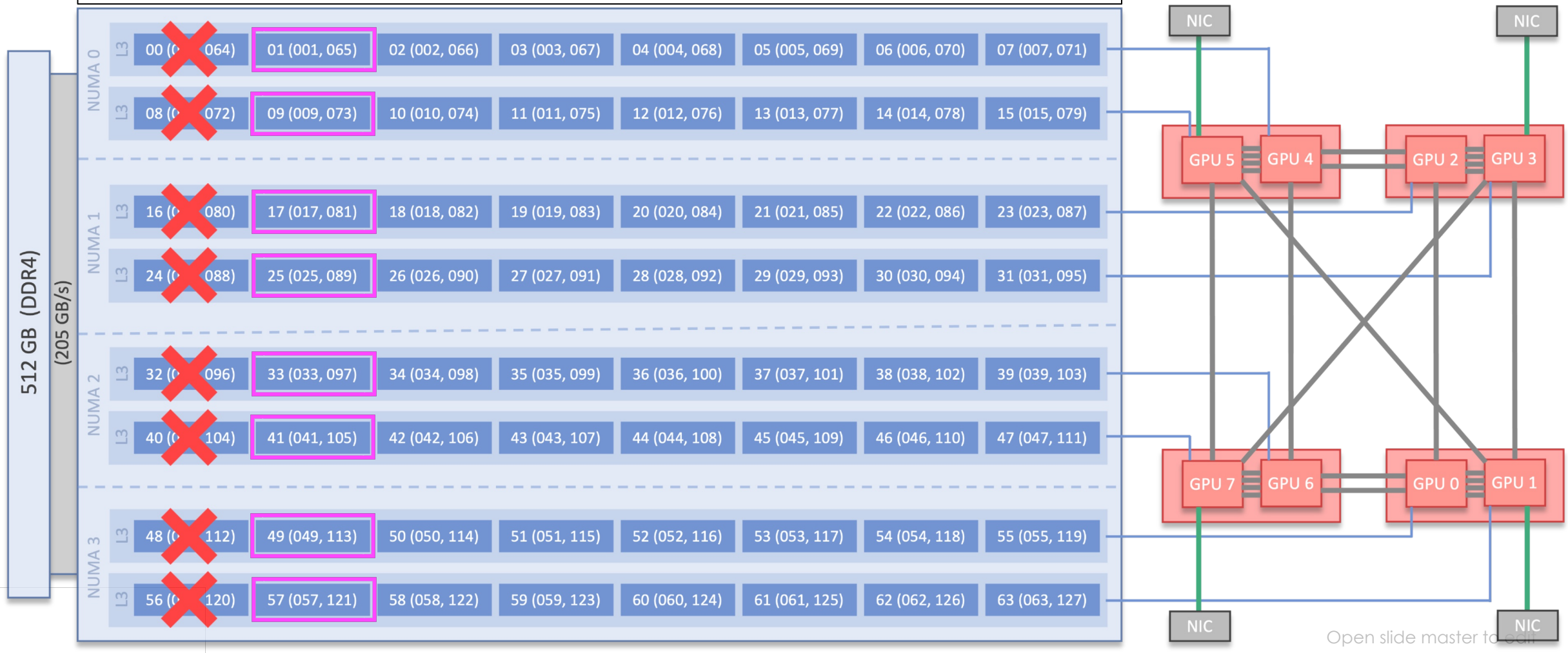
Incorrect mapping!

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
```

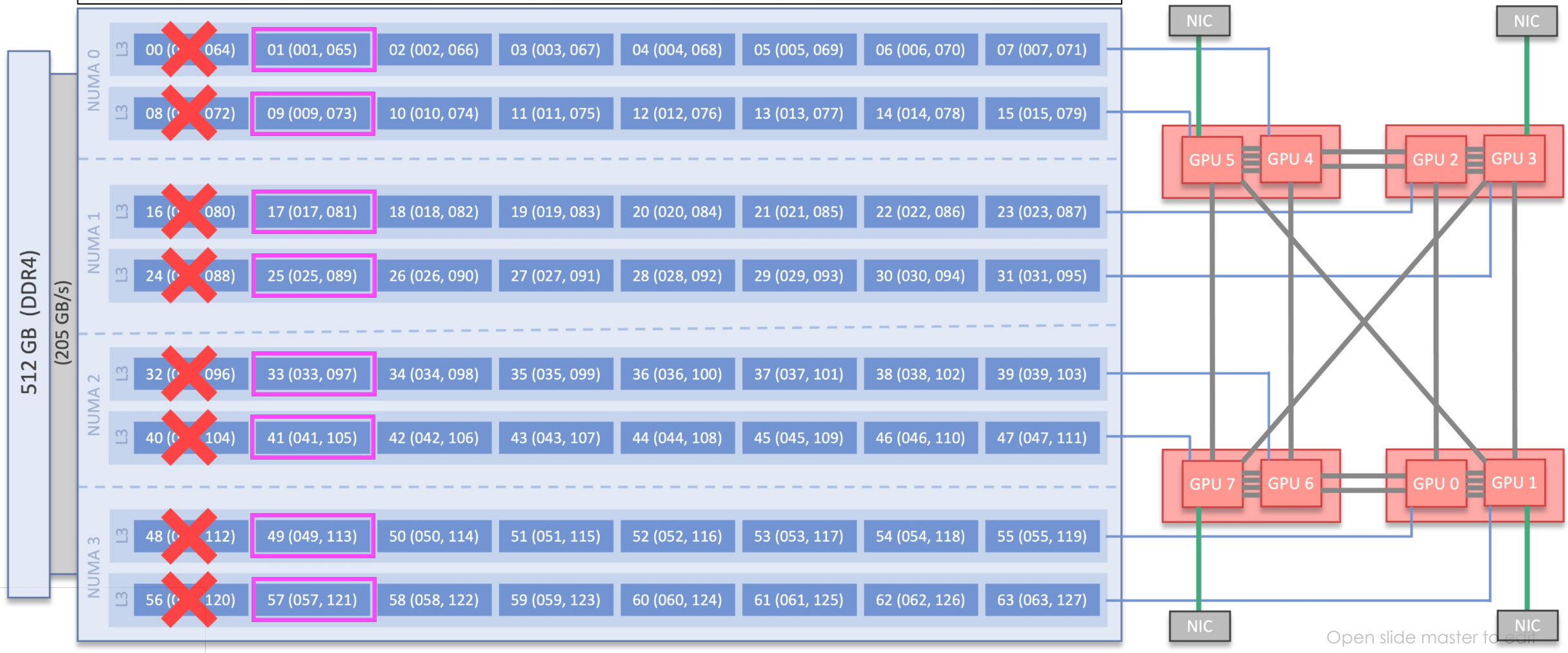
8 tasks per node, each with 7 CPU cores and 1 GPU



```
$ 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!

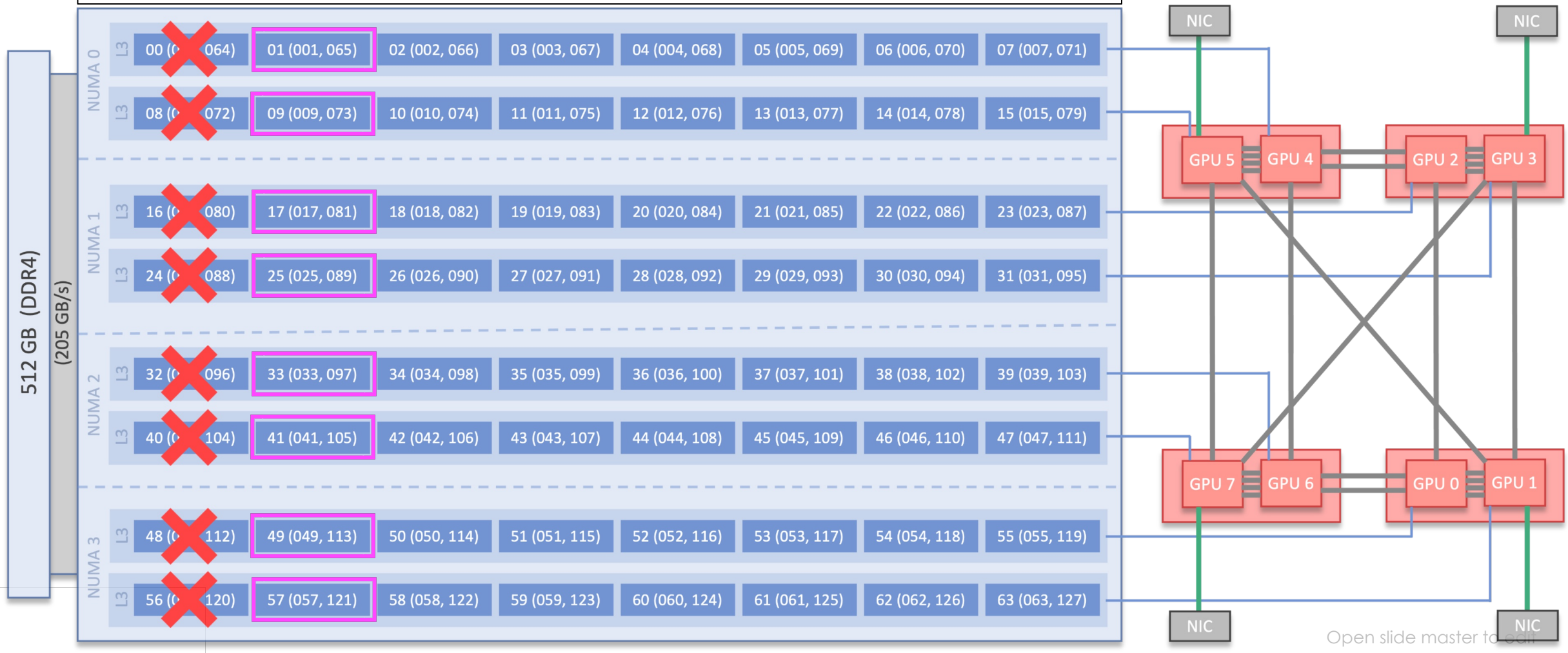
8 tasks per node, each with 7 CPU cores and 1 GPU



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
```

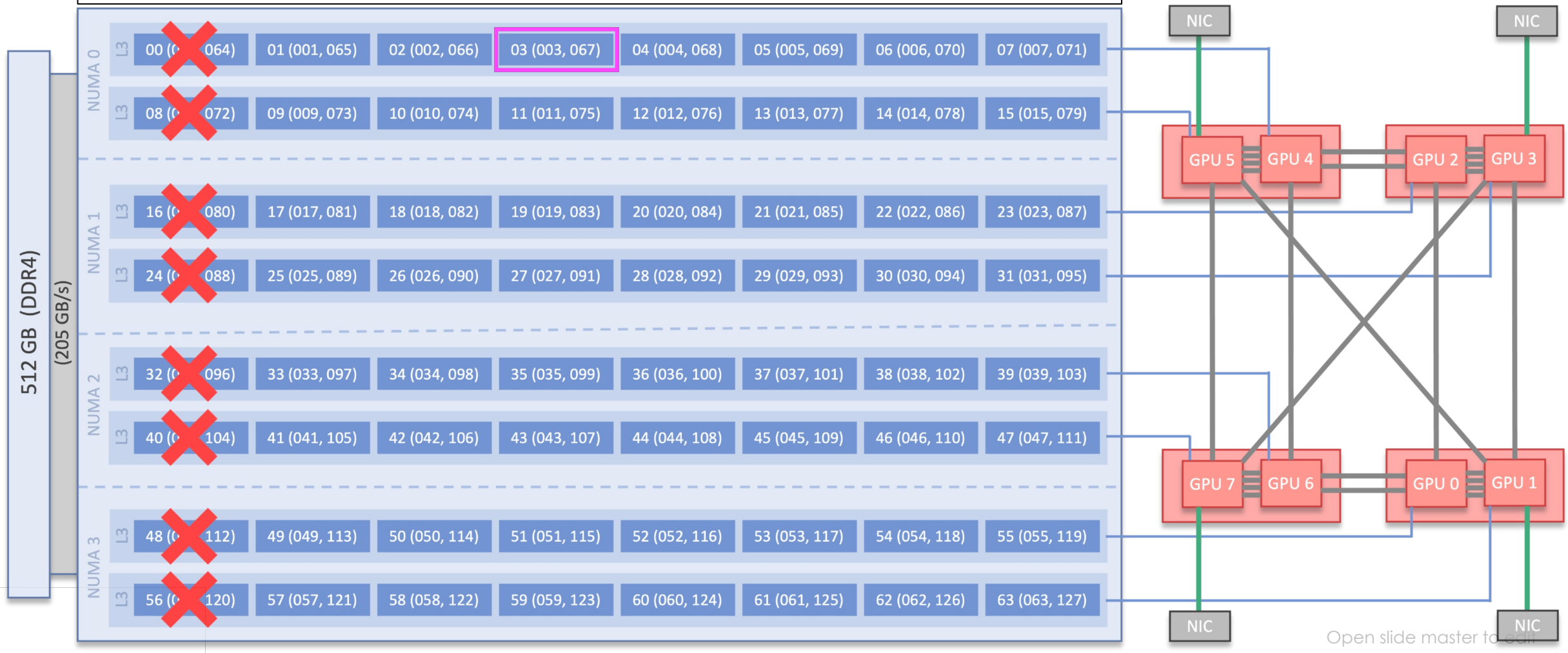
✗ Incorrect mapping!

```
--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.
```

ʊ_ʊ

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



Open slide master to edit

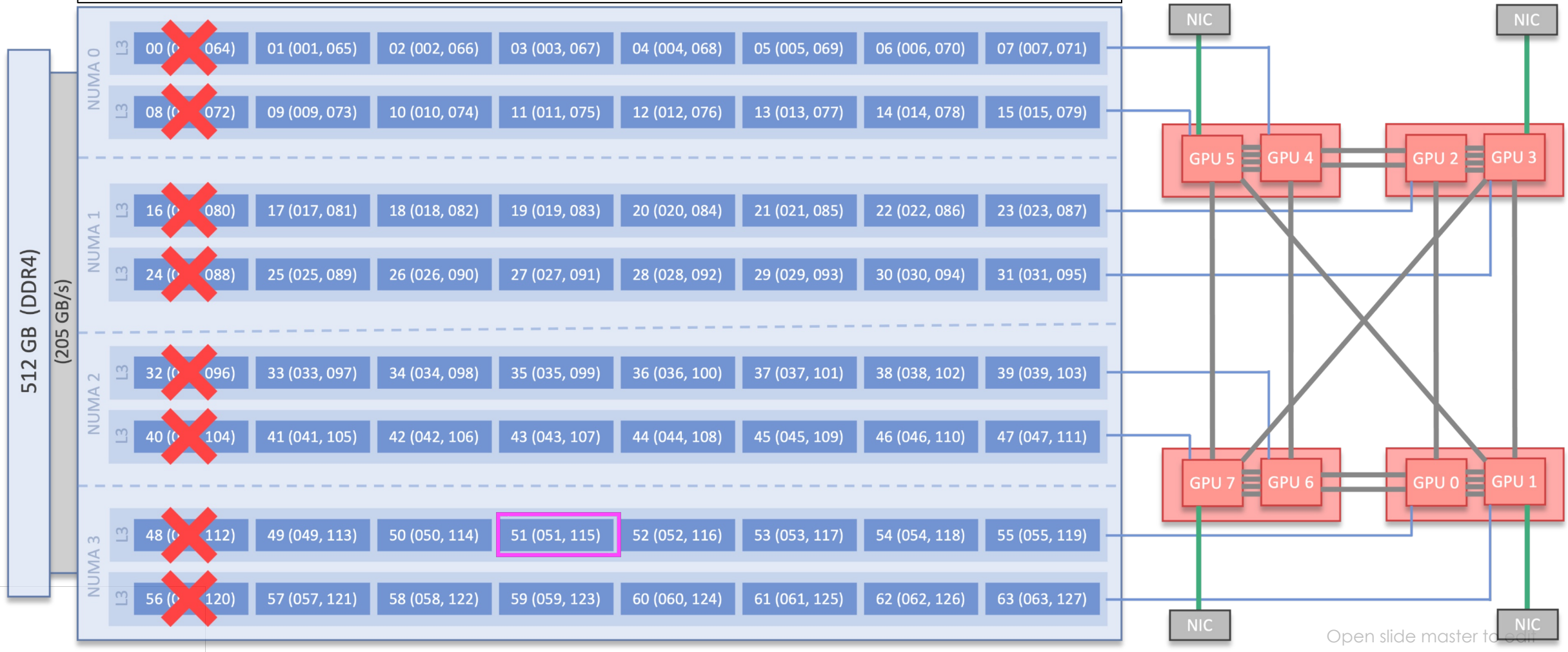
```
$ OMP_NUM_THREADS=1 srun -N1 -n1 -c7 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 051 - Node frontier08303 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
```

```
--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
```

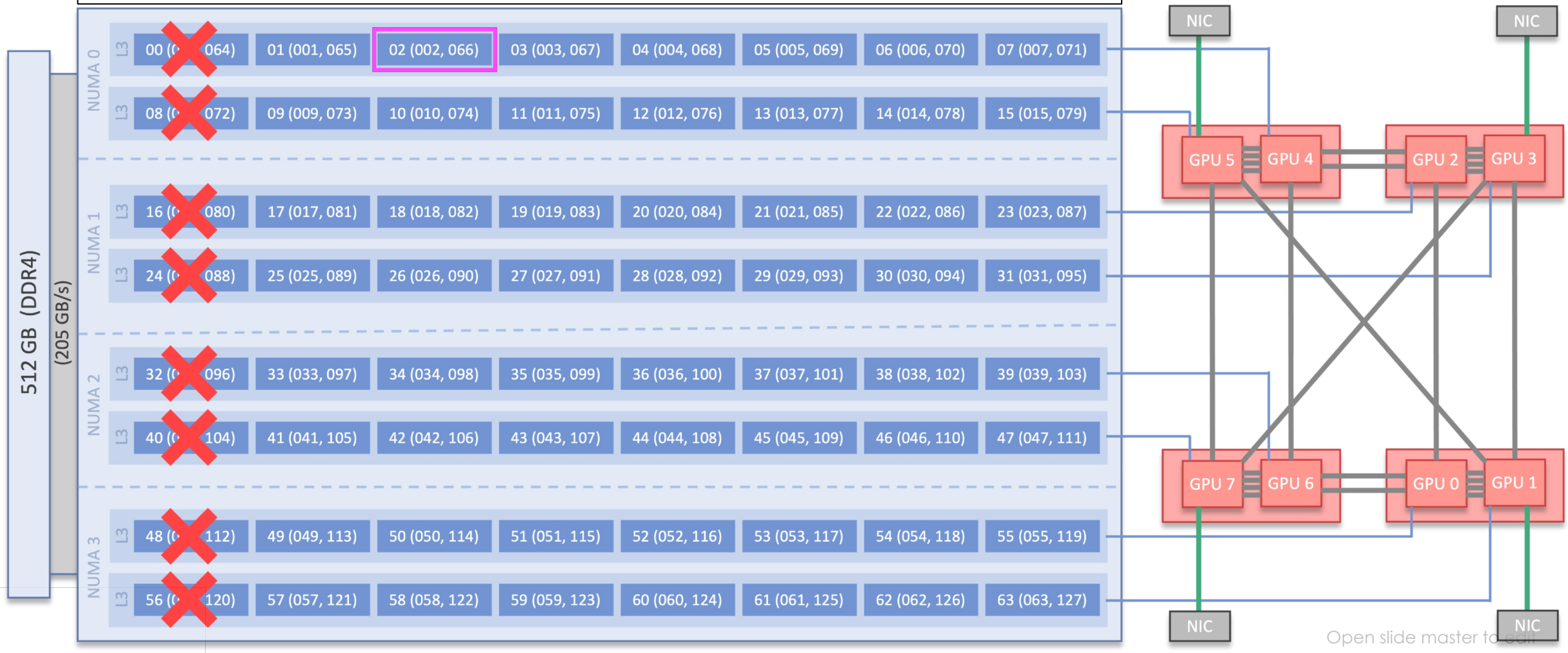
✗ Incorrect mapping!

```
--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.
```

ʊ_ʊ

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



Open slide master to edit

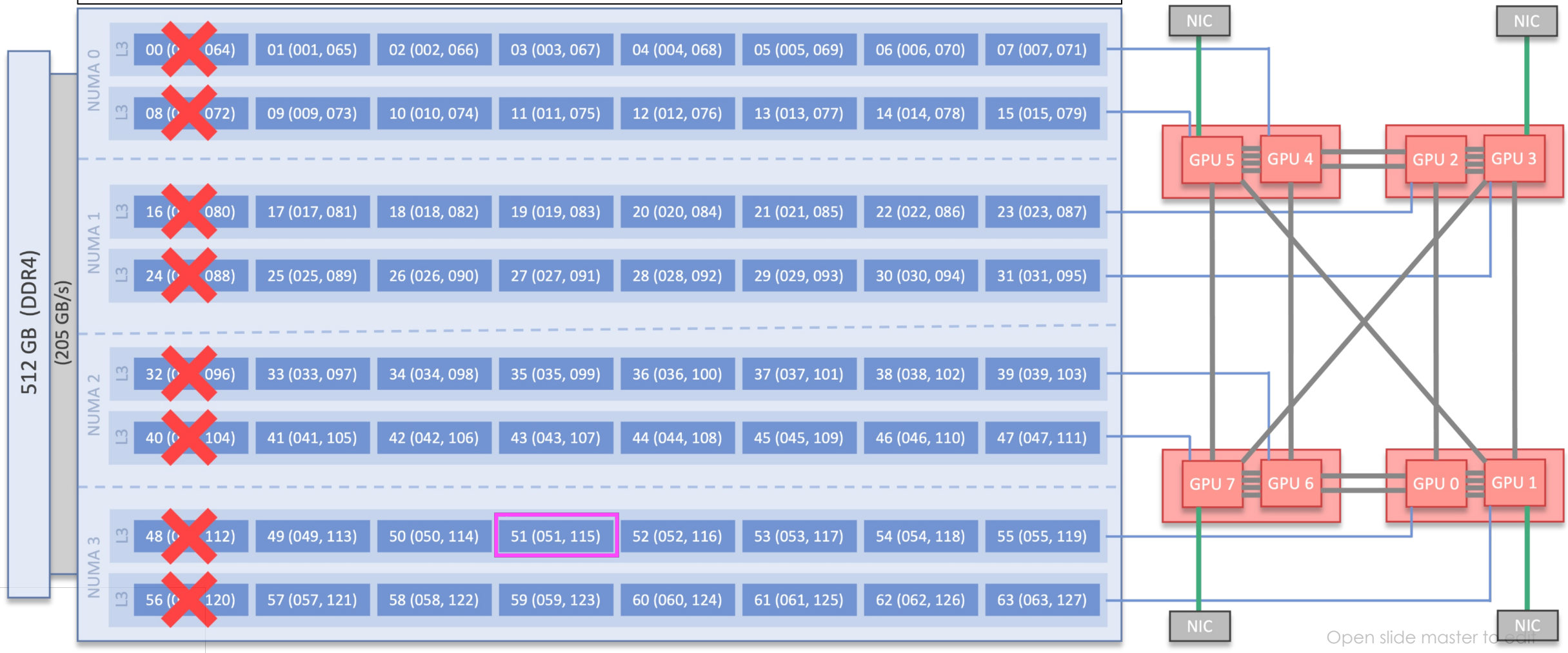
```
$ OMP_NUM_THREADS=1 srun -N2 -n2 -c7 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep | sort
MPI 000 - OMP 000 - HWT 051 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 001 - OMP 000 - HWT 049 - Node frontier10228 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
```

```
--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 Crusher), 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 -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

??

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
```

Due to core isolation:

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

This can be resolved by either:

1. Setting core isolation to `-S16` so there are only 6 cores available per L3
2. Removing the core isolation and using 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 -A<project> -N 1 -t 10 -S 16
...
$ 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 frontier01738 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 005 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 002 - OMP 000 - HWT 010 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 003 - OMP 000 - HWT 013 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 004 - OMP 000 - HWT 018 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 005 - OMP 000 - HWT 021 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 006 - OMP 000 - HWT 026 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 007 - OMP 000 - HWT 029 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 008 - OMP 000 - HWT 034 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 009 - OMP 000 - HWT 037 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 010 - OMP 000 - HWT 042 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 011 - OMP 000 - HWT 045 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
MPI 012 - OMP 000 - HWT 052 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 013 - OMP 000 - HWT 053 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 014 - OMP 000 - HWT 058 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 015 - OMP 000 - HWT 061 - Node frontier01738 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6

$ 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: frontier01738 Cpus_allowed_list: 2-4 GPUS: 4
 1: frontier01738 Cpus_allowed_list: 5-7 GPUS: 4
 2: frontier01738 Cpus_allowed_list: 10-12 GPUS: 5
 3: frontier01738 Cpus_allowed_list: 13-15 GPUS: 5
 4: frontier01738 Cpus_allowed_list: 18-20 GPUS: 2
 5: frontier01738 Cpus_allowed_list: 21-23 GPUS: 2
 6: frontier01738 Cpus_allowed_list: 26-28 GPUS: 3
 7: frontier01738 Cpus_allowed_list: 29-31 GPUS: 3
 8: frontier01738 Cpus_allowed_list: 34-36 GPUS: 6
 9: frontier01738 Cpus_allowed_list: 37-39 GPUS: 6
10: frontier01738 Cpus_allowed_list: 42-44 GPUS: 7
11: frontier01738 Cpus_allowed_list: 45-47 GPUS: 7
12: frontier01738 Cpus_allowed_list: 50-52 GPUS: 0
13: frontier01738 Cpus_allowed_list: 53-55 GPUS: 0
14: frontier01738 Cpus_allowed_list: 58-60 GPUS: 1
15: frontier01738 Cpus_allowed_list: 61-63 GPUS: 1
```

Make only needed cores available

Using all CPU cores on a Crusher/Frontier node

NOT RECOMMENDED. Why?

- All system processes are still bound to core 0, so core 0 will provide lower performance
- Orion is configured to use the 1st core from each L3 cache to improve I/O performance

```
 -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
```

← `-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
...
```

Remember:

- core 0 will have lower performance than other cores
- Orion performance may be reduced

Multiple job steps on a single node

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

for idx in {1..8}; do
    # Requires cray-mpich version > 8.1.18
    OMP_NUM_THREADS=1 srun --exact -u -N1 -w <node_name> -n1 -c7 --gpus-per-task=1 --gpu-bind=closest ./hello_jobstep &
    usleep 50000
done
wait
```

```
$ ./multiple_jobsteps.sh
Tue 14 Feb 2023 12:31:59 PM EST
MPI 000 - OMP 000 - HWT 051 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1

Tue 14 Feb 2023 12:32:00 PM EST
MPI 000 - OMP 000 - HWT 059 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6

Tue 14 Feb 2023 12:32:01 PM EST
MPI 000 - OMP 000 - HWT 019 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9

Tue 14 Feb 2023 12:32:02 PM EST
MPI 000 - OMP 000 - HWT 027 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce

Tue 14 Feb 2023 12:32:03 PM EST
MPI 000 - OMP 000 - HWT 003 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1

Tue 14 Feb 2023 12:32:04 PM EST
MPI 000 - OMP 000 - HWT 011 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6

Tue 14 Feb 2023 12:32:05 PM EST
MPI 000 - OMP 000 - HWT 035 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9

Tue 14 Feb 2023 12:32:06 PM EST
MPI 000 - OMP 000 - HWT 042 - Node frontier10227 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID de
```

Added a `sleep(20)` to the bottom of `hello_jobstep` so it didn't return so quickly.

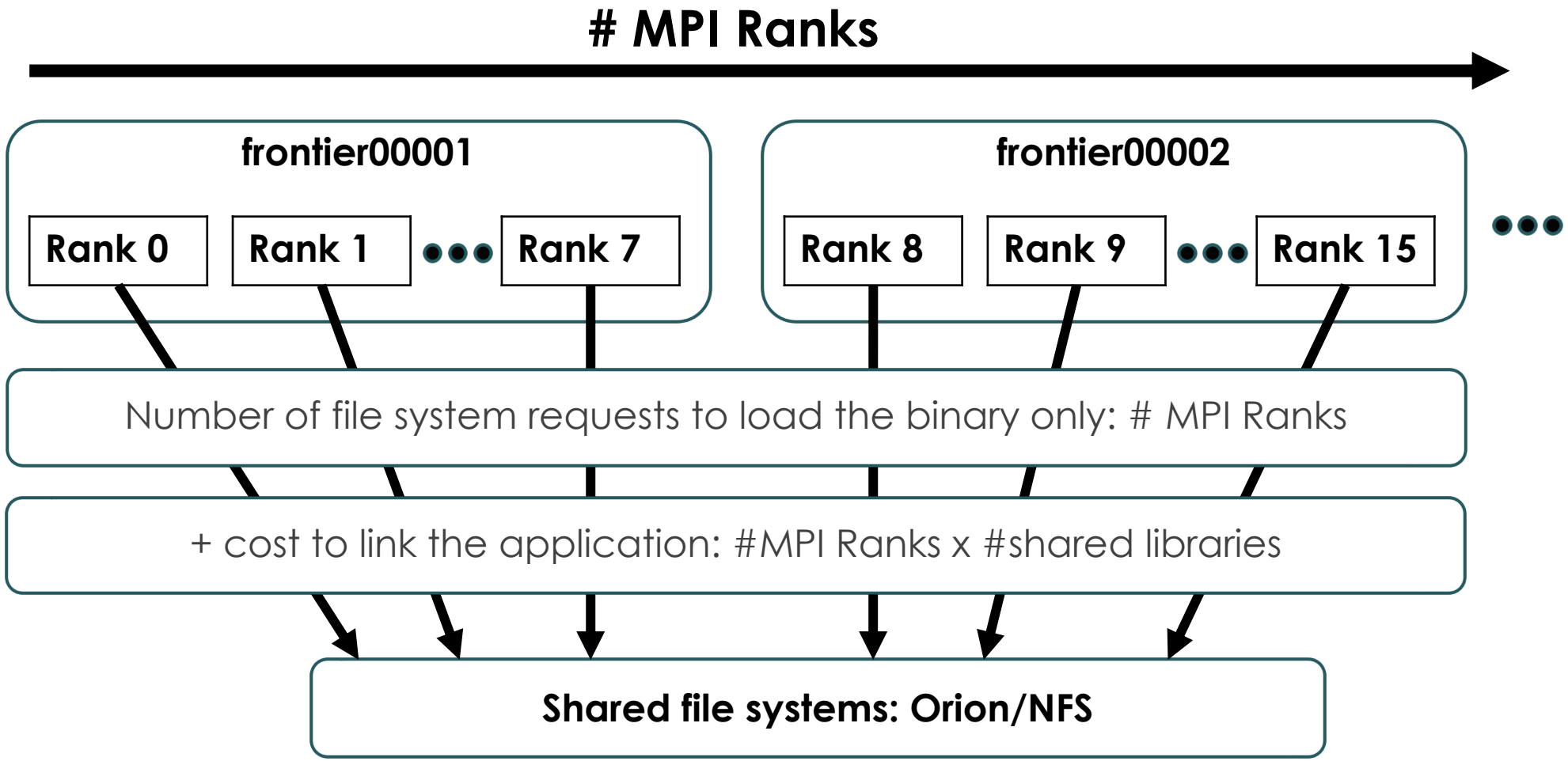
For concurrent multi-node steps, same process, but can only run 3 concurrent steps

OAK RIDGE National Laboratory

Using sbcast to improve job initialization at scale

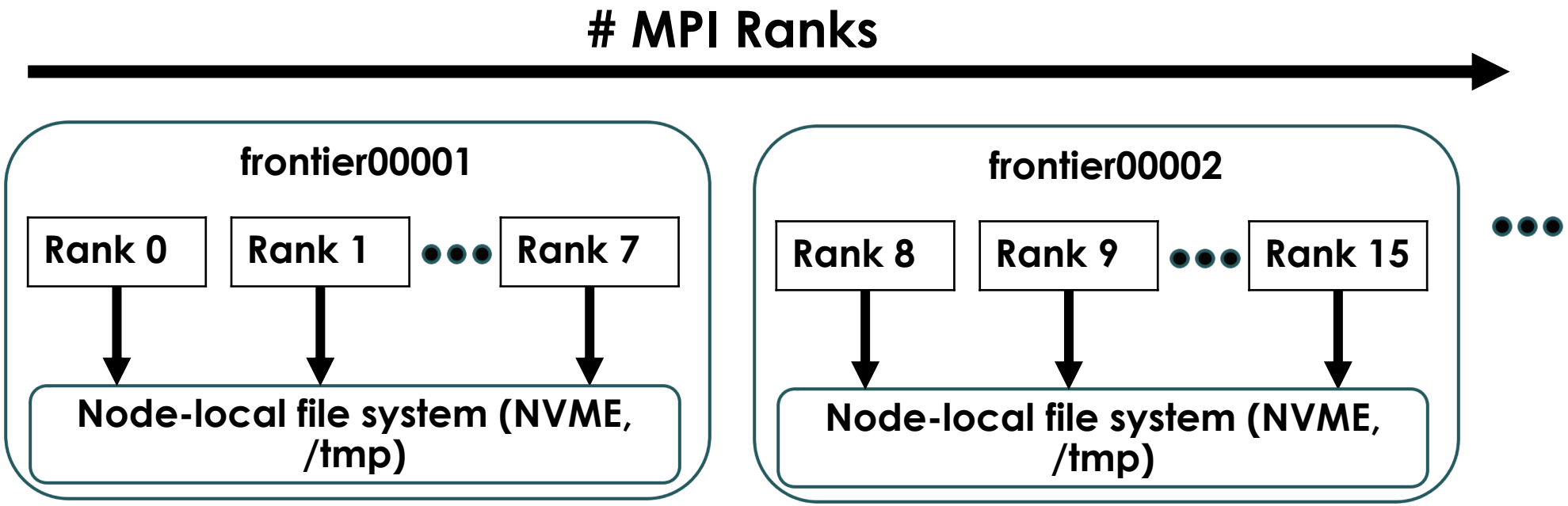
Running at scale

By default, every single rank reads from the file system independently:



Running at scale

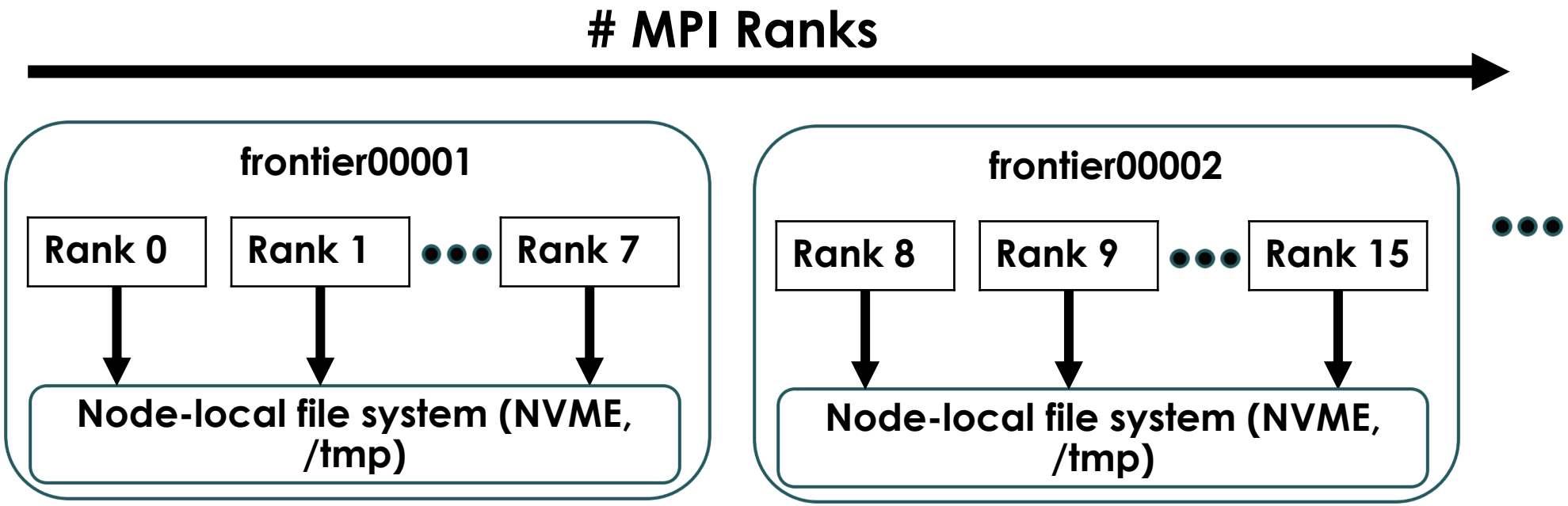
But, if we can pre-stage binary & libraries in node-local storage:



Shared file systems: Orion/NFS

Running at scale

But, if we can pre-stage binary & libraries in node-local storage:



Benefits*:

- Decrease in application launch & link times
- Reduced load & dependence on shared file systems
- Consistent/predictable launch-time behavior
- Reduced occurrence of timeouts at launch time
- Early detection of failed run

Running at scale - sbcast

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 [ ! "$?" == 0 ]; then echo "SBCAST failed"; exit 1; fi`
6	
7	`# This does not work if you use RPATH's`
8	`export LD_LIBRARY_PATH="/mnt/bb/$USER/${exe}_libs"`
9	
10	`# libfabric dlopen's several libraries. Add those paths to end of LD_LIBRARY_PATH:`
11	`export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:$(pkg-config --variable=libdir libfabric)"`
12	
13	`# Patch: libfabric dlopen's `libhsa-runtime64.so`, but only `libhsa-runtime64.so.1` is sbcast`
14	`# Add sym-link on each node for `libhsa-runtime64.so``
15	`srun -N ${SLURM_NNODES} -n ${SLURM_NNODES} --ntasks-per-node=1 --label \`
16	`    -D /mnt/bb/$USER/${exe}_libs`
17	`    if [ -f libhsa-runtime64.so.1 ]; then ln -s libhsa-runtime64.so.1 libhsa-runtime64.so; fi`

Running at scale - sbcast

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`

Running at scale - sbcast

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

Running at scale - sbcast

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

Running at scale - sbcast

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`

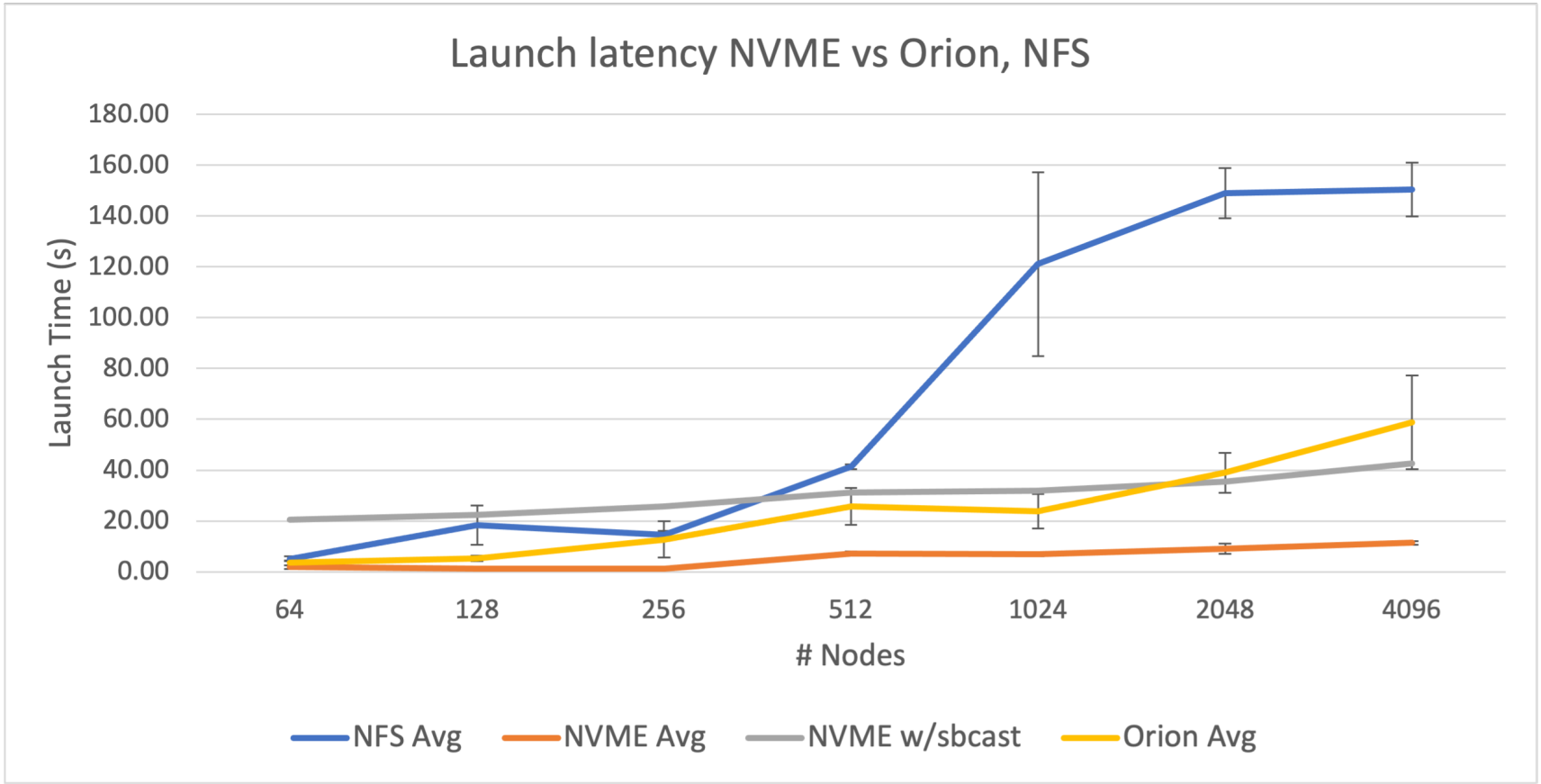
sbcast cannot detect `dlopen` calls, so a few patches are required:

Running at scale - sbcast

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, RUNPATH paths will still be used**
  - Spack often adds library paths to RPATH
  - make and cmake *sometimes* do this, check your binary to confirm (see: `readelf`)
- 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. Include any CrayPE and ROCm libraries that your application links (ie, rocBLAS).**

The benefit of sbcast



- Reduced occurrence of timeouts at launch time
- Early detection of failed initialization

- Consistent/predictable launch behavior
- Lower launch times
- Reduced reliance on Orion/NFS latency

OAK RIDGE National Laboratory

Job Accounting

Slurm Tips – job analysis slide 1/3

2 available sources of information: scontrol and sacct

- scontrol shows information about a pending, running, or recently completed (within 10 minutes) job
- sacct queries the Slurm database to show detailed information about each step in a job

```
> scontrol show job 1413140
JobId=1413140 JobName=lammps_cpu_nodecheck
   UserId=efaccept(15549) GroupId=efaccept(27243) MCS_label=N/A
   Priority=3447 Nice=0 Account=stf016_frontier QOS=normal
   JobState=PENDING Reason=Priority Dependency=(null)
   Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
…
```

```
> sacct -j 1234567
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
1234567      orion_85_+      batch     stf002        126  COMPLETED      0:0
1234567.bat+      batch                stf002         63  COMPLETED      0:0
1234567.ext+     extern                stf002        126  COMPLETED      0:0
1234567.0           lcio                stf002         63  CANCELLED     0:15
```

NOTE: sacct data is not guaranteed to be accurate until a job completes. Running sacct during a job is not recommended.

Slurm Tips – job analysis slide 2/3

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

- The default time window depends on Slurm configuration (see `man sacct`)

```
$ sacct --user=tpapathe -S 2022-05-05T00:01 -E 2022-05-05T23:59 -X

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 – job analysis slide 3/3

sacct --env-vars shows … all environment variables set at the start of a specific Slurm job.
sacct --batch-script … shows the batch script run by a specific Slurm job

Slurm provides 100+ fields in the sacct command

```
$ sacct --helpformat
Account              AdminComment         AllocCPUS            AllocNodes
AllocTRES            AssocID              AveCPU               AveCPUFreq
AveDiskRead          AveDiskWrite         AvePages             AveRSS
AveVMSize            BlockID              Cluster              Comment
Constraints          ConsumedEnergy       ConsumedEnergyRaw    Container
CPUTime              CPUTimeRAW           DBIndex              DerivedExitCode
…
$ sacct -j 123456 -o jobid,jobname,account,allocnodes
JobID           JobName    Account AllocNodes
------------ ---------- ---------- ----------
1234567      orion_85_+     stf002          1
1234567.bat+      batch     stf002          1
1234567.ext+     extern     stf002          1
1234567.0          lcio     stf002          1
```

OAK RIDGE National Laboratory

Getting Help

Helpful debugging flags

Useful Slurm flags:

- `-u` flag gives unbuffered output
- `-l` flag prepends the task ID to lines of stdout

Shell commands to run immediately before a job step:

`ldd ${exe}  &> ldd.log` – verifies that your program selects the intended libraries

`module -t list  &> modules.log` – prints list of loaded modules

`printenv  &> env.log` – prints all environment variables

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

Questions?

HAGERTYNL@ornl.gov

OAK RIDGE National Laboratory