OAK RIDGE National Laboratory

jsrun Basics

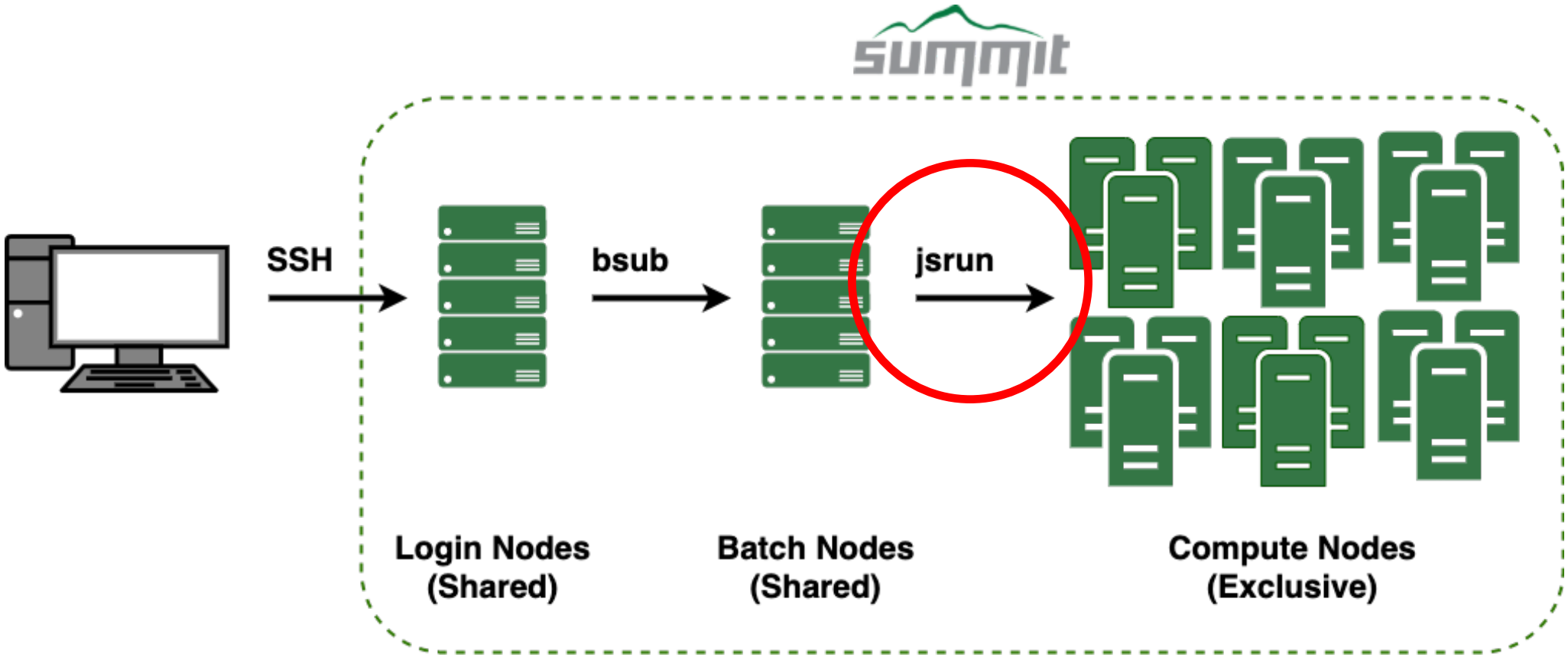
Jack Morrison

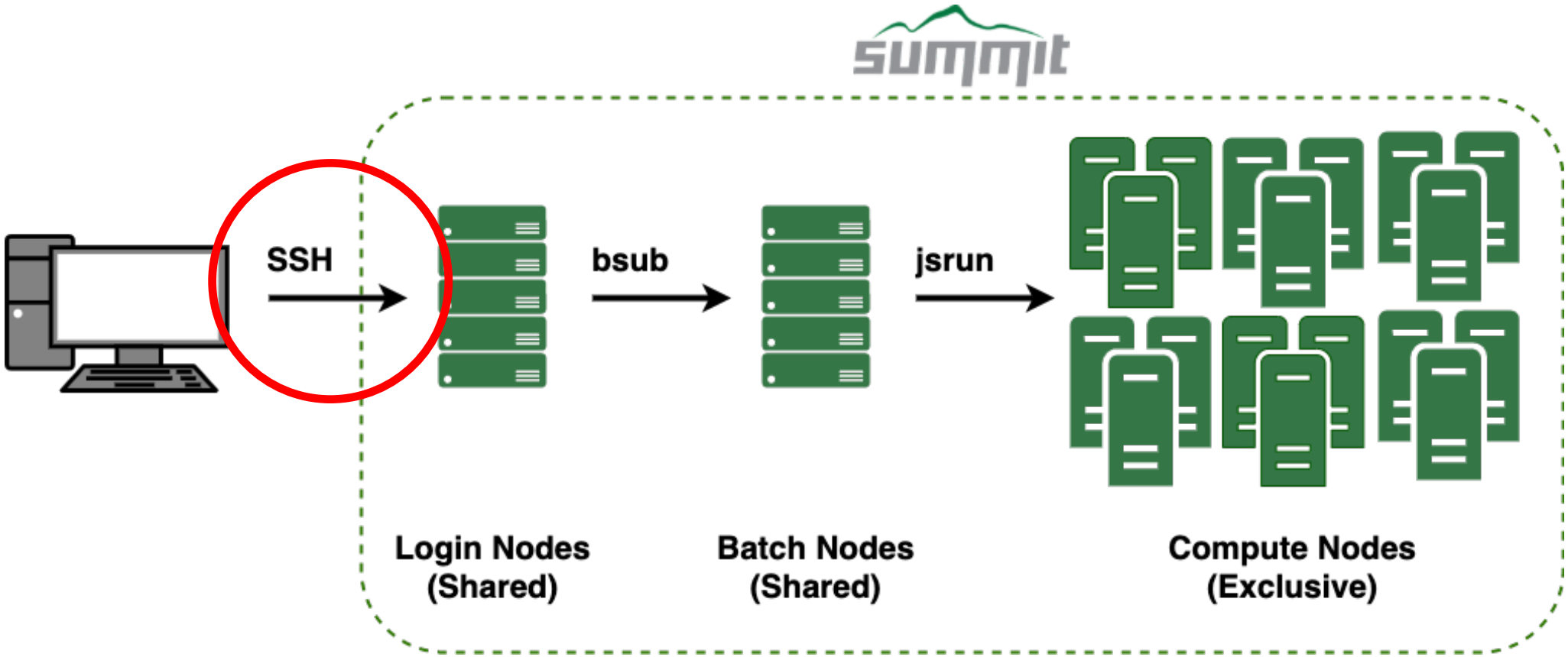
User Assistance and Outreach Group
Oak Ridge Leadership Computing Facility

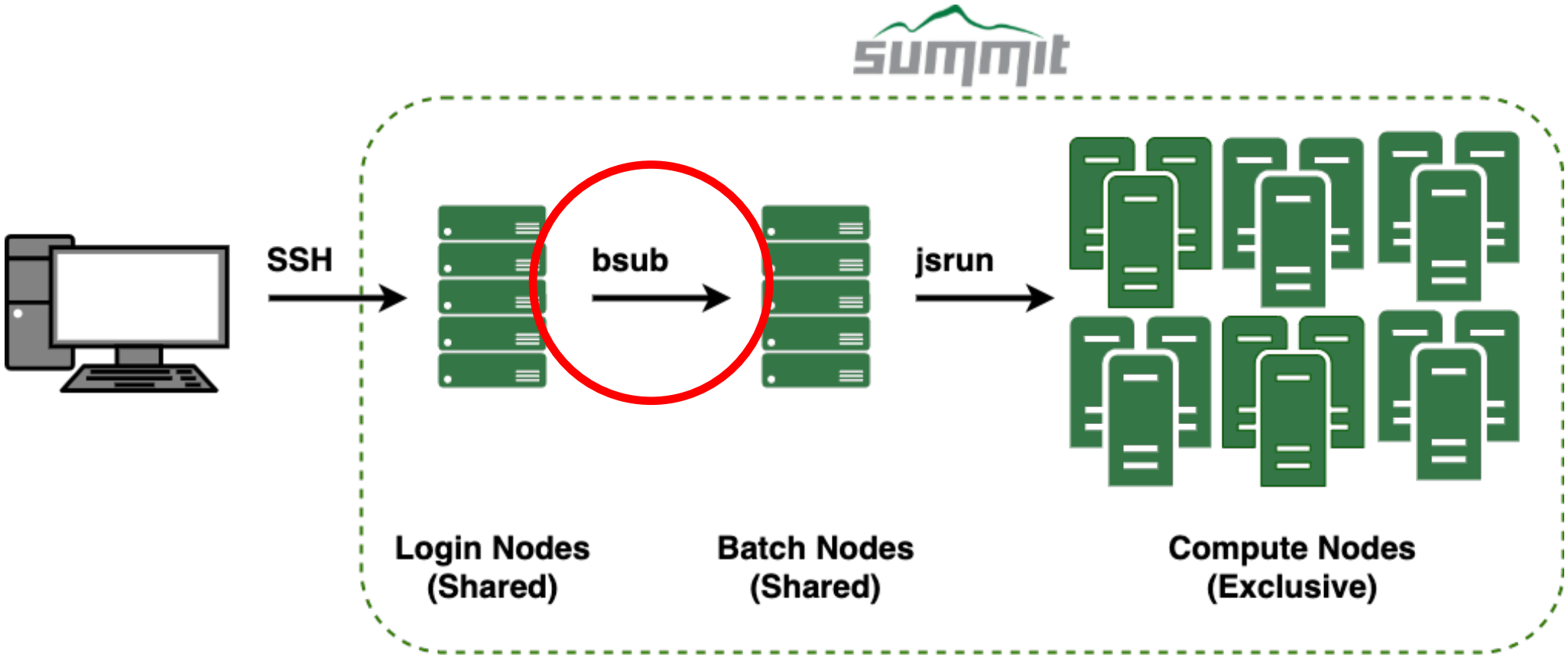
February 18, 2020

U.S. DEPARTMENT OF ENERGY


summit
IBM
EXIT

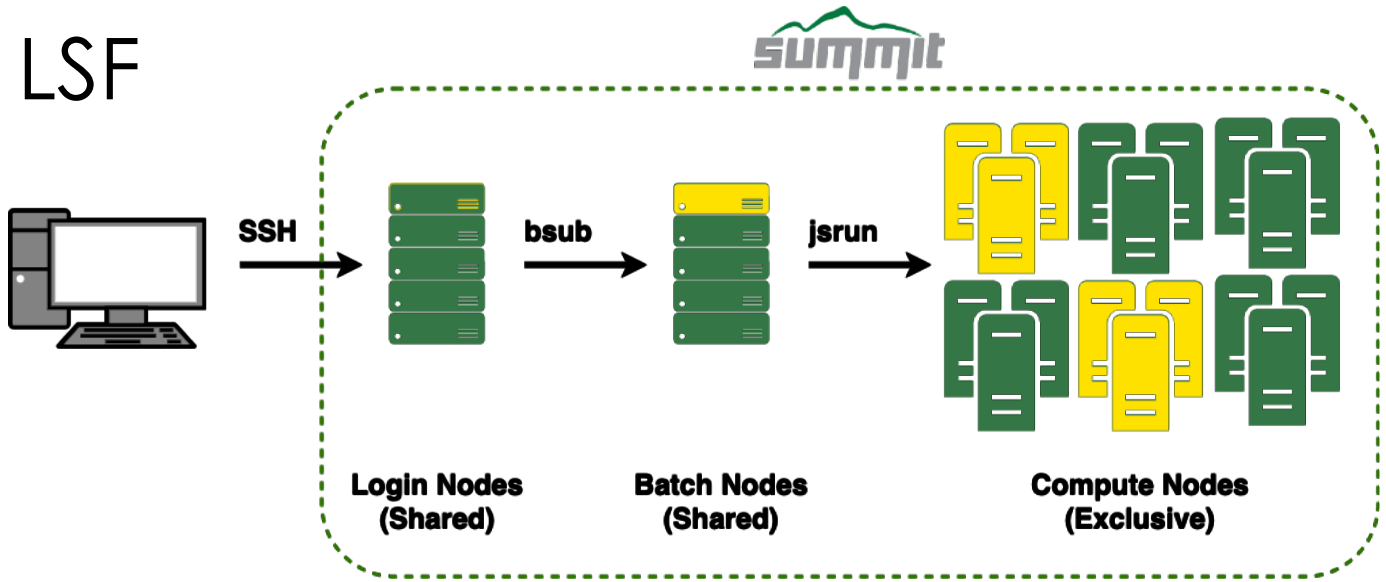

summit
SSH
bsub
jsrun
Login Nodes
(Shared)
Batch Nodes
(Shared)
Compute Nodes
(Exclusive)


summit
SSH
bsub
jsrun
Login Nodes
(Shared)
Batch Nodes
(Shared)
Compute Nodes
(Exclusive)


summit
SSH
bsub
jsrun
Login Nodes
(Shared)
Batch Nodes
(Shared)
Compute Nodes
(Exclusive)

bsub – Submit a job to LSF

- bsub allocates 1 batch node + # of requested compute nodes



- bsub -alloc_flags defines allocation-wide configurations
  - Applied to every compute node
    - CPU Simultaneous Multithreading Level (smt)
    - GPU Multi-Process Service (gpumps)
    - Burst Buffer (nvme)
    - Others (spectral, maximizegpfs, ...)
  - Multiple options require quoting, space separated
    - #BSUB -alloc_flags "gpumps smt1 nvme"

bsub (non-interactive)

myScript.lsf

```
#!/bin/bash

#BSUB -P ABC123
#BSUB -J myLSFjob
#BSUB -o out.%J
#BSUB -e err.%J
#BSUB -W 30
#BSUB -nnodes 1
#BSUB -alloc_flags nvme

hostname
jsrun -n1 hostname
hostname
```

```
batch2
h23n01
batch2
```

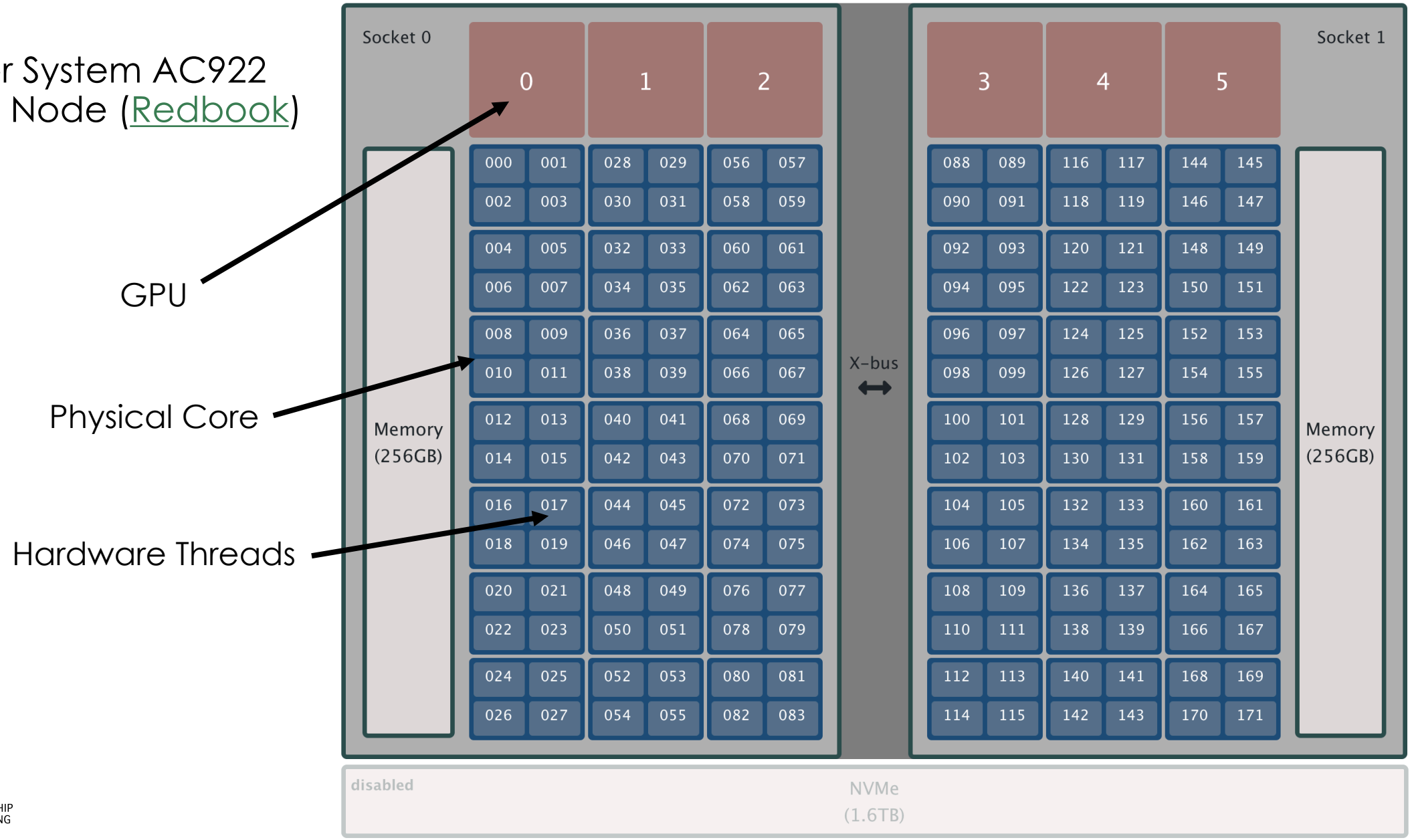
```
morrison@login1.summit> bsub myScript.lsf
Job <12345> is submitted to default queue <batch>.
morrison@login1.summit>
```

bsub (interactive)

```
morrison@login1.summit> bsub -P ABC123 -J myLSFjob -W30 -nnodes 1 -alloc_flags nvme -Is $SHELL
Job <12345> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on batch2>>

morrison@batch2.summit> hostname
batch2

morrison@batch2.summit> jsrun -n1 hostname
h23n01

morrison@batch2.summit> hostname
batch2
```

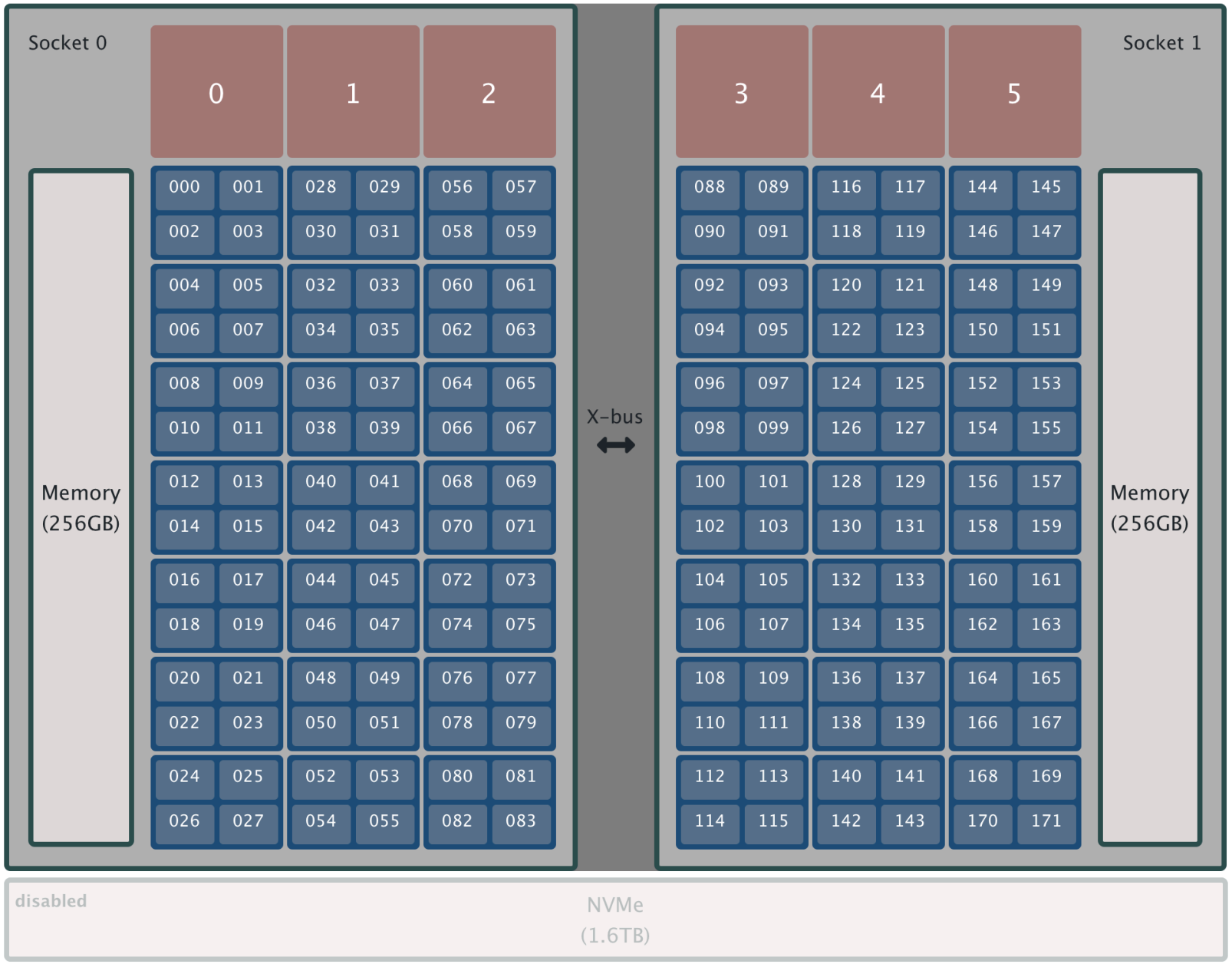
Summit Compute Node

- IBM Power System AC922 Compute Node (Redbook)



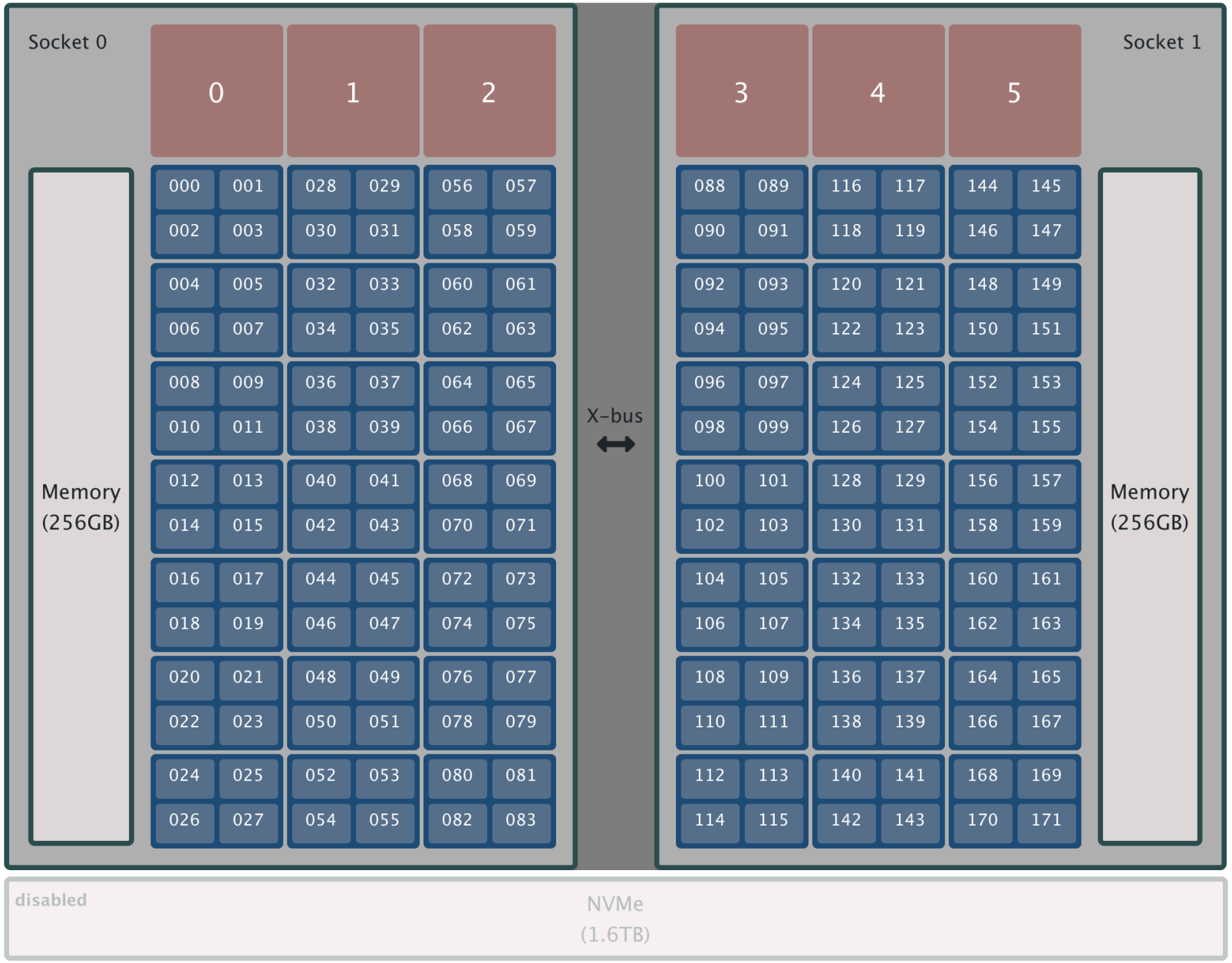
Summit Compute Node

- IBM Power System AC922 Compute Node (Redbook)
- 2 Sockets
  - 3 NVIDIA V100 GPUs
  - 21 usable cores
    - {1, 2, 4}-way Multithreading (SMT)
  - 256 GB DDR4 RAM



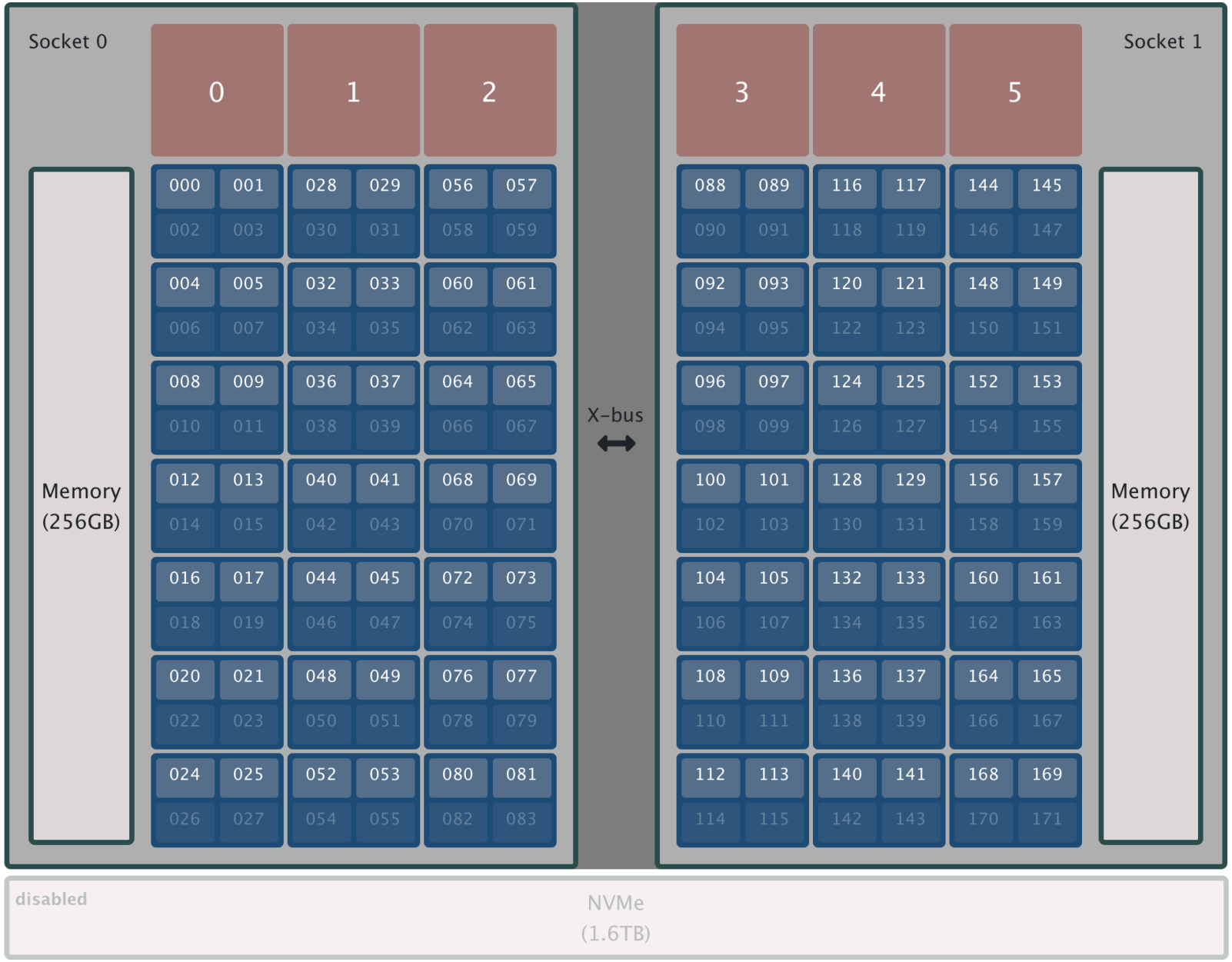
Summit Compute Node

- IBM Power System AC922 Compute Node (Redbook)
- 2 Sockets
  - 3 NVIDIA V100 GPUs
  - 21 usable cores
    - {1, 2, 4}-way Multithreading (SMT)
  - 256 GB DDR4 RAM



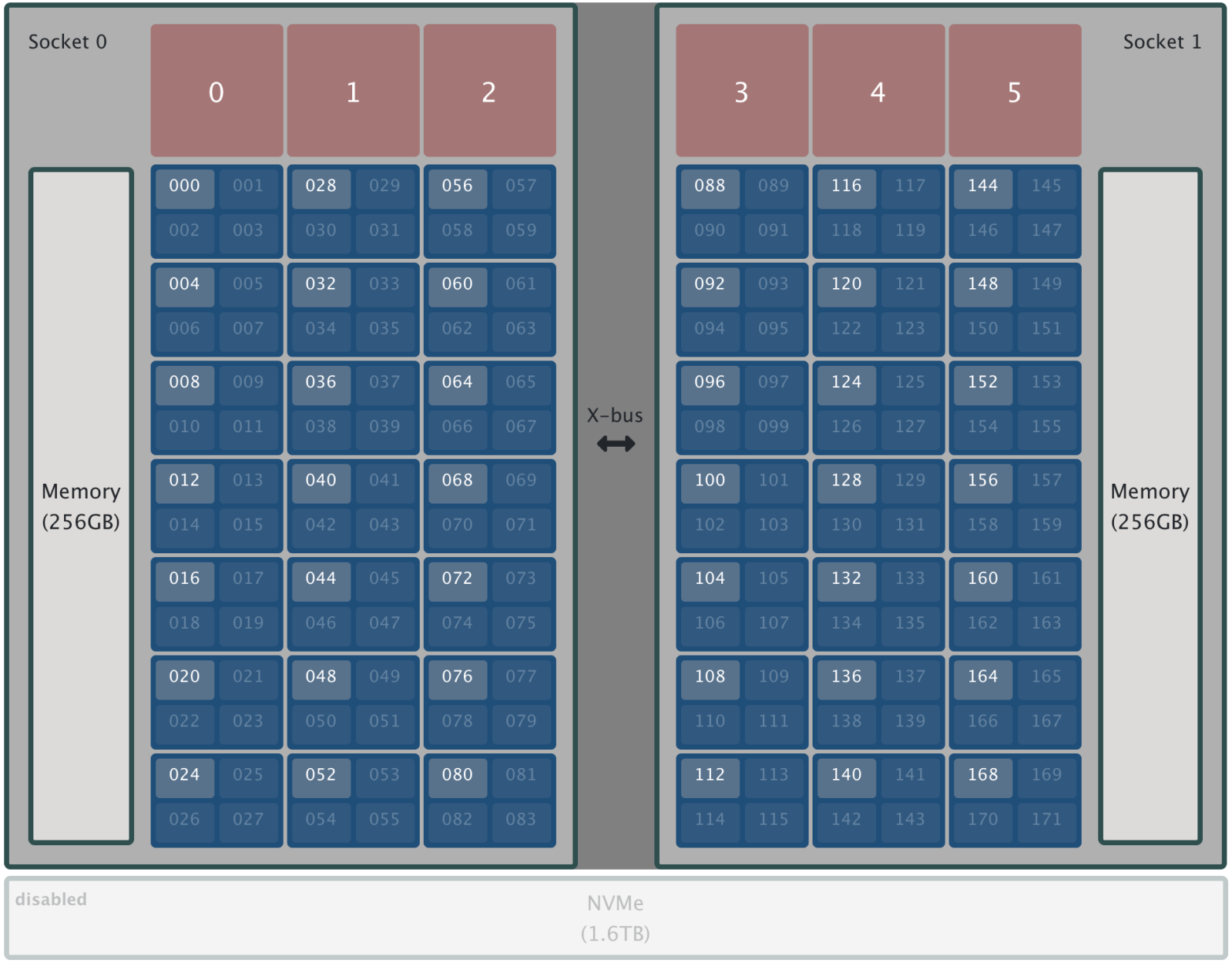
Summit Compute Node

- IBM Power System AC922 Compute Node (Redbook)
- 2 Sockets
  - 3 NVIDIA V100 GPUs
  - 21 usable cores
    - {1, 2, 4}-way Multithreading (SMT)
  - 256 GB DDR4 RAM



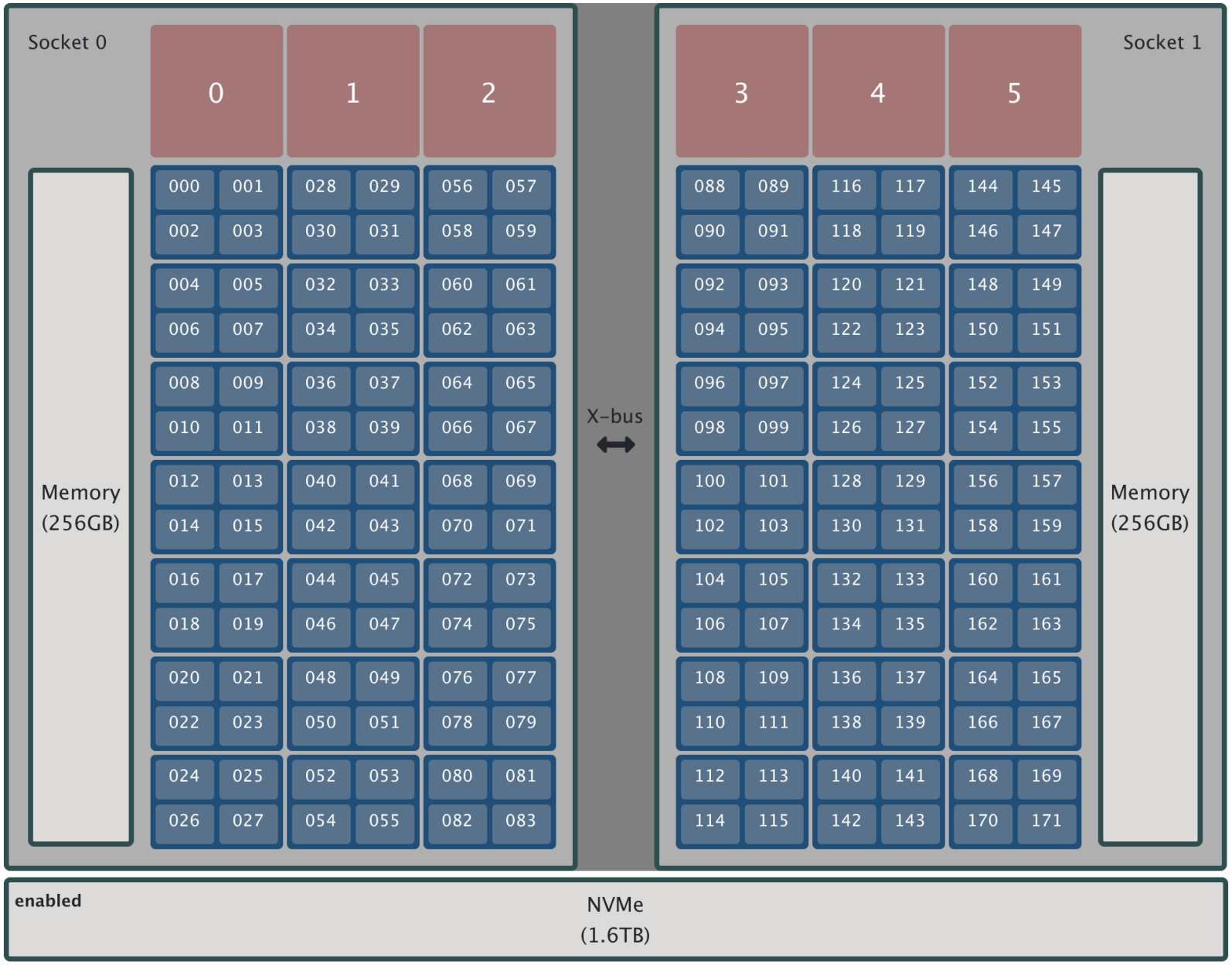
Summit Compute Node

- IBM Power System AC922 Compute Node (Redbook)
- 2 Sockets
  - 3 NVIDIA V100 GPUs
  - 21 usable cores
    - {1, 2, 4}-way Multithreading (SMT)
  - 256 GB DDR4 RAM



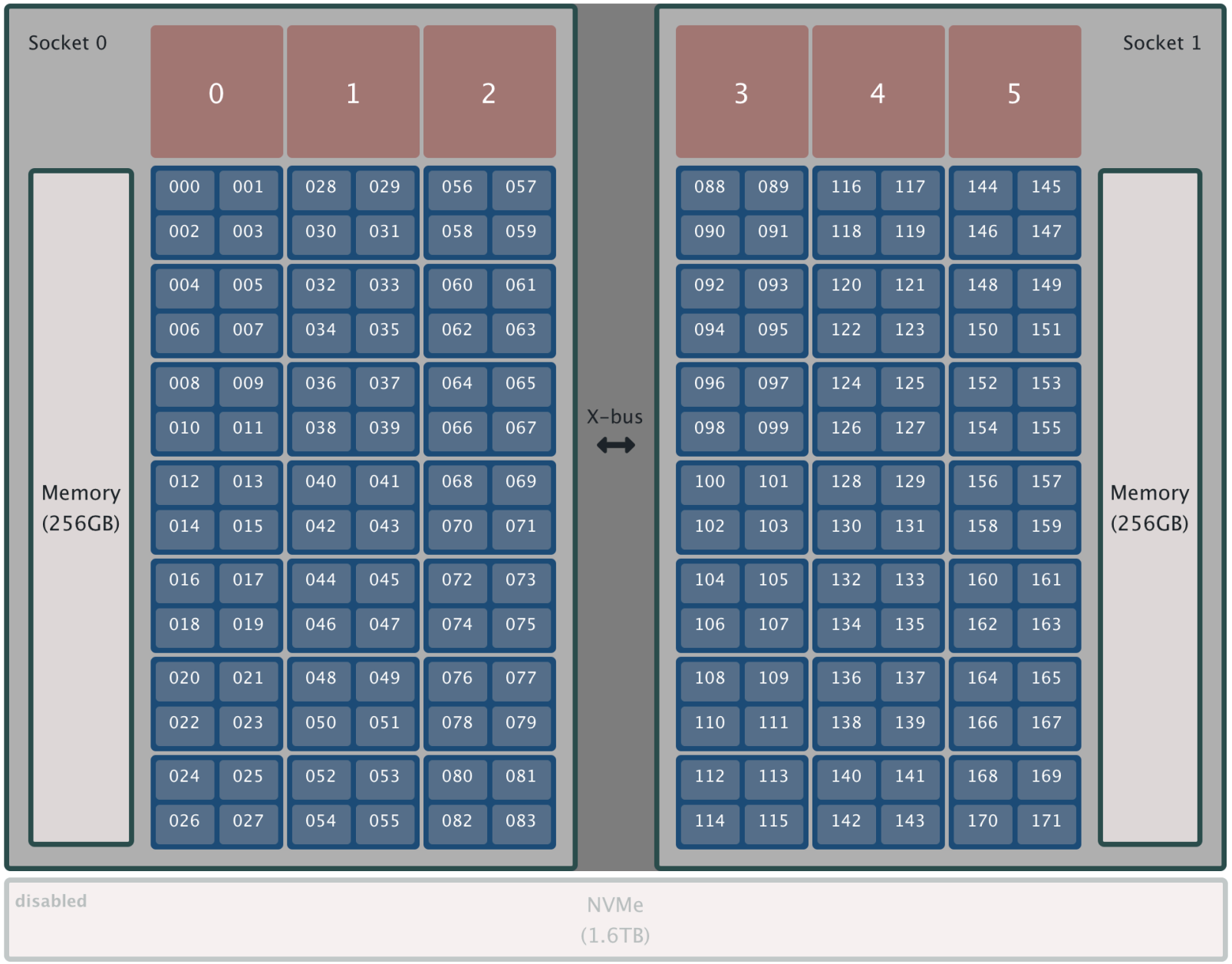
Summit Compute Node

- IBM Power System AC922 Compute Node (Redbook)
- 2 Sockets
  - 3 NVIDIA V100 GPUs
  - 21 usable cores
    - {1, 2, 4}-way Multithreading (SMT)
  - 256 GB DDR4 RAM
- 1.6TB NVMe (Burst Buffer)



Summit Compute Node

- IBM Power System AC922 Compute Node (Redbook)
- 2 Sockets
  - 3 NVIDIA V100 GPUs
  - 21 usable cores
    - {1, 2, 4}-way Multithreading (SMT)
  - 256 GB DDR4 RAM
- 1.6TB NVMe (Burst Buffer)
- Exclusive access during job
- Service Core Isolation
- X-bus (64GB/s)



jsrun format

```
jsrun [-n #resource sets] [tasks, threads, and GPUs in each resource set] program [program args]
```

- Resource Sets (RS)
  - They're just cgroups!
  - Used to shape the compute node to your application by grouping
    - Physical cores
    - GPUs
    - RAM*
  - Must contain 1 or more physical cores and 0 or more GPUs
  - Allow for multiple job steps on a single node (Adv.)
  - Highly flexible with Explicit Resource Files (Adv.)
- Not as scary as they seem

`jsrun` format

```
jsrun [-n #resource sets] [tasks, threads, and GPUs in each resource set] program [program args]
```

- Some limitations
  - A resource set may span sockets, but *cannot* span nodes
    - Creating resource sets within sockets can avoid cross-socket communication
  - Memory access requires a physical core or GPU on its host socket
    - Again, X-bus may be a bottleneck
  - Resource Sets are homogeneous by default
    - Heterogeneous resource sets possible with ERF (Adv.)

Designing a Resource Set

```
jsrun [-n #resource sets] [tasks, threads, and GPUs in each resource set] program [program args]
```

1. Understand how the code expects the node to appear
   - How many tasks/threads per GPU?
   - Does each task expect to see a single GPU? Do multiple tasks expect to share a GPU (gpumps)? Is the code written to internally manage task to GPU workloads based on the number of available cores and GPUs?
2. Create Resource Sets containing the needed GPU to task binding
   - Describe a resource set that meets the requirements of #1. If the code is written for one GPU per task, consider a resource set with just one GPU.
3. Decide on the number of Resource Sets needed
   - After understanding task, thread, and GPU requirements, scale the number of Resource Sets (and number of nodes) as needed.

Designing a Resource Set – Basic Options

```
jsrun [-n #resource sets] [tasks, threads, and GPUs in each resource set] program [program args]
```

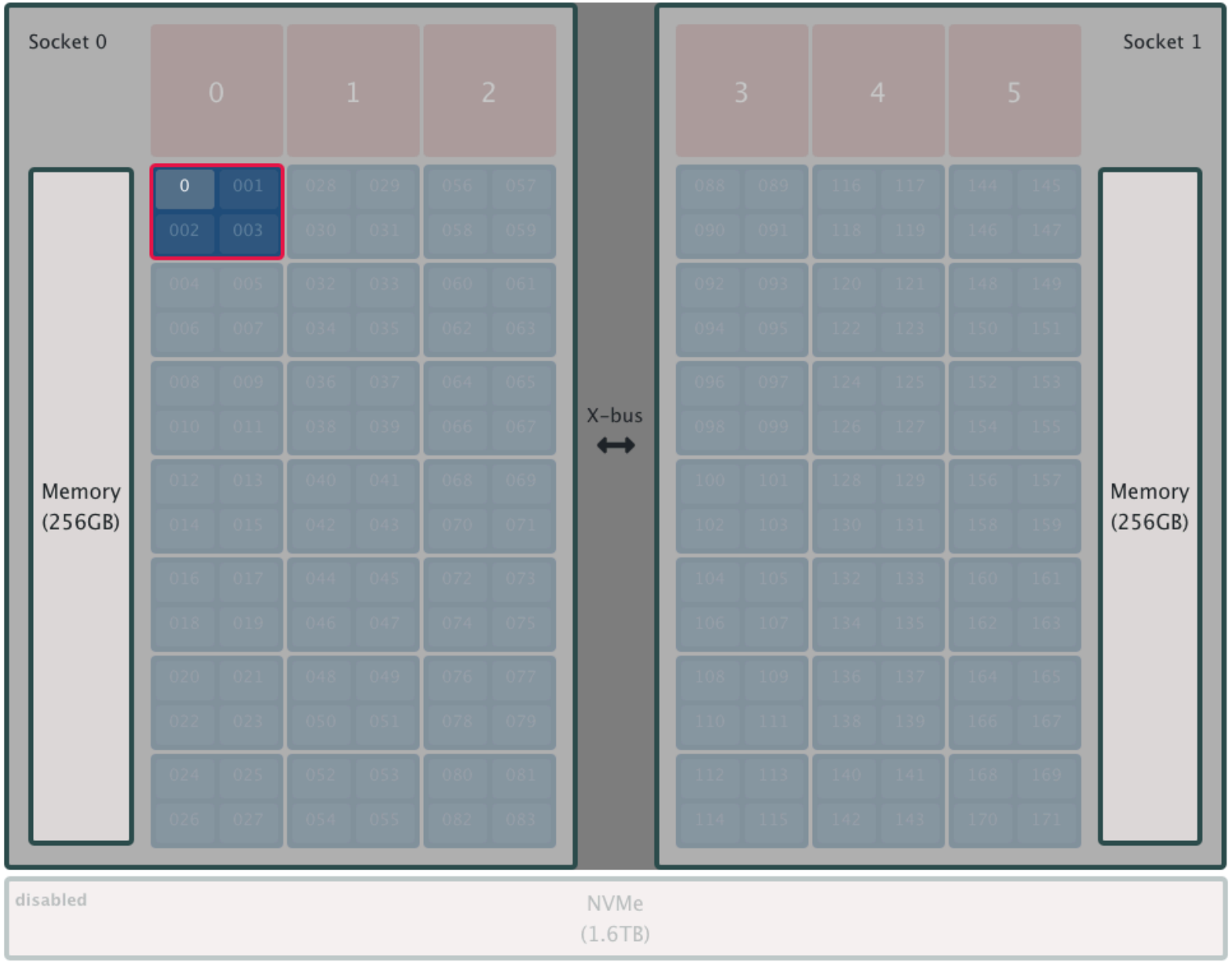
jsrun Flags		Description	Default Value
Long	Short		
`--nrs`	`-n`	Number of RS	All available physical cores
`--tasks_per_rs`	`-a`	Number of MPI tasks (ranks) per RS	N/A (total set instead [`-p`])
`--cpu_per_rs`	`-c`	Number of CPUs (physical cores) per RS	`1`
`--gpu_per_rs`	`-g`	Number of GPUs per RS	`0`
`--bind`	`-b`	Number of physical cores allocated per task	`packed:1`
`--rs_per_host`	`-r`	Number of RS per host (node)	N/A
`--latency_priority`	`-l`	Controls layout priorities	`gpu-cpu,cpu-mem,cpu-cpu`
`--launch_distribution`	`-d`	Order of tasks started on multiple RS	`packed`

See `man jsrun` for full list of options

jsrun examples

```
jsrun -n1
```

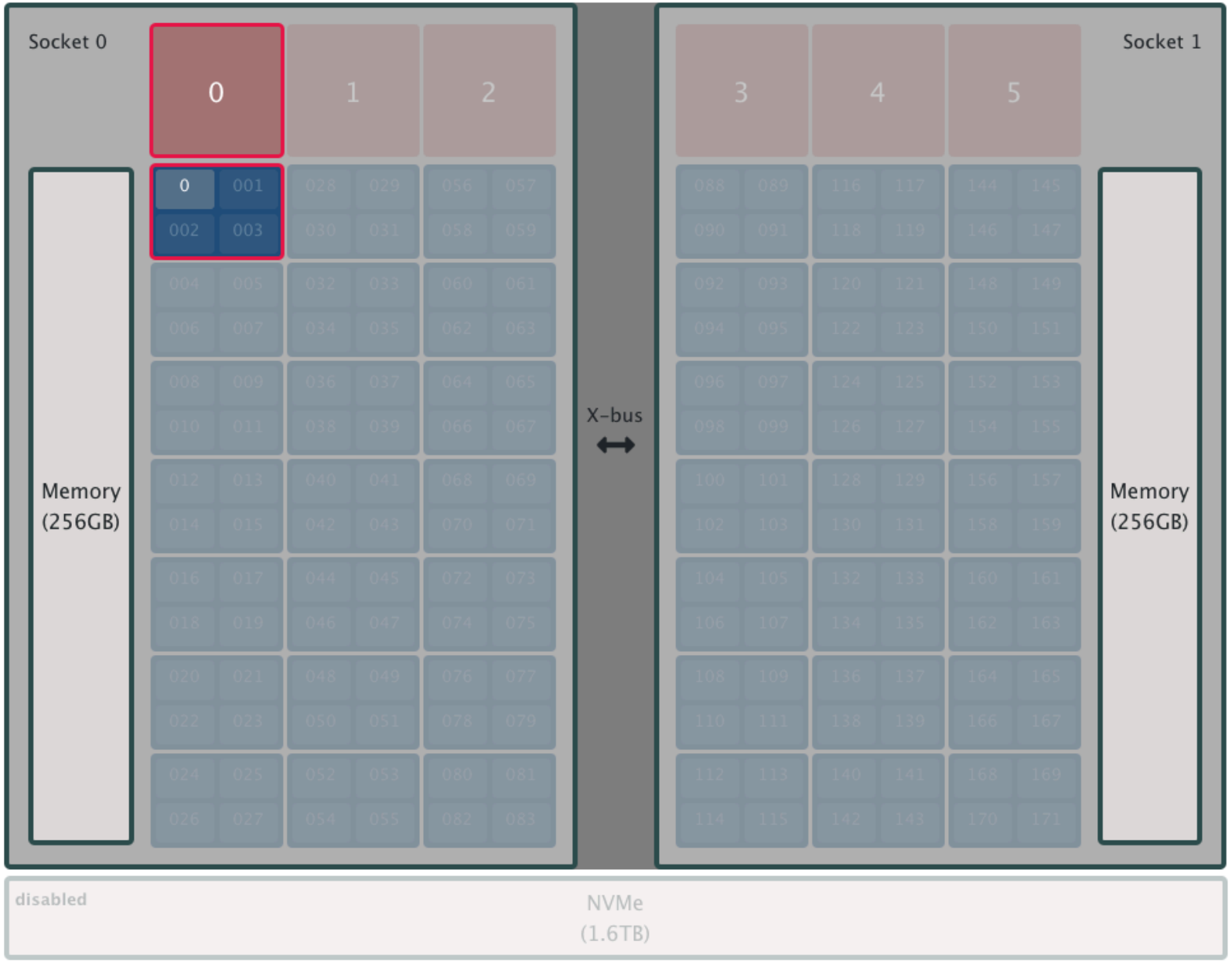
- What do you get by default?
  - Almost nothing
- Building a resource set can be iterative



jsrun examples

```
jsrun -n1 -c1 -g1 -a1
```

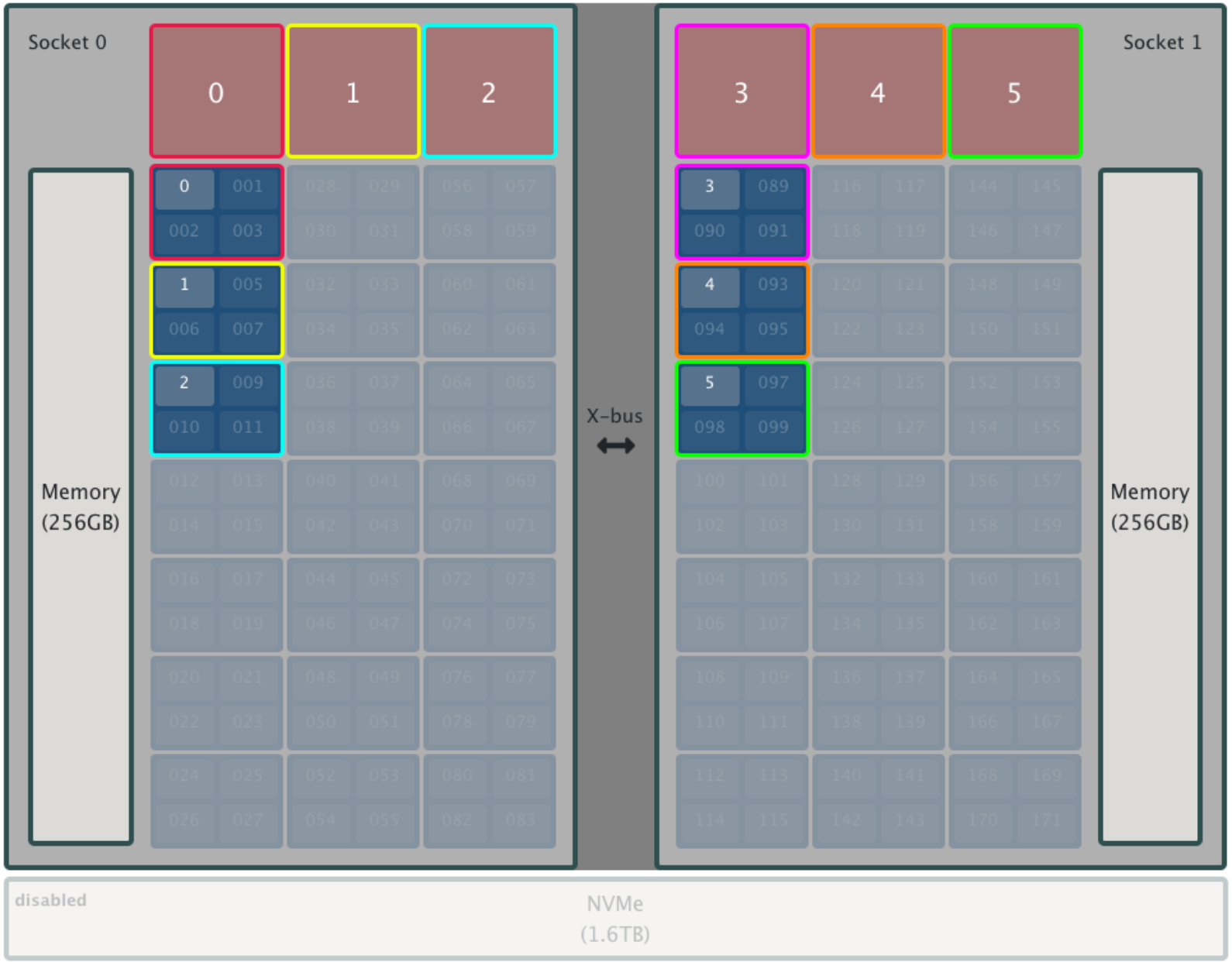
- Explicitly requesting resources is to your benefit.
  - Don't rely on defaults.
- How many of these resource sets can we fit on a single node?



jsrun examples

```
jsrun -n6 -c1 -g1 -a1
```

- What about utilizing all available physical cores?



`jsrun` examples

```
jsrun -n6 -c7 -g1 -a1
```

- How about multiple ranks per GPU?

jsrun examples

```
jsrun -n6 -c7 -g1 -a7 -dpacked
```

- Change the order of tasks started across multiple resource sets

jsrun examples

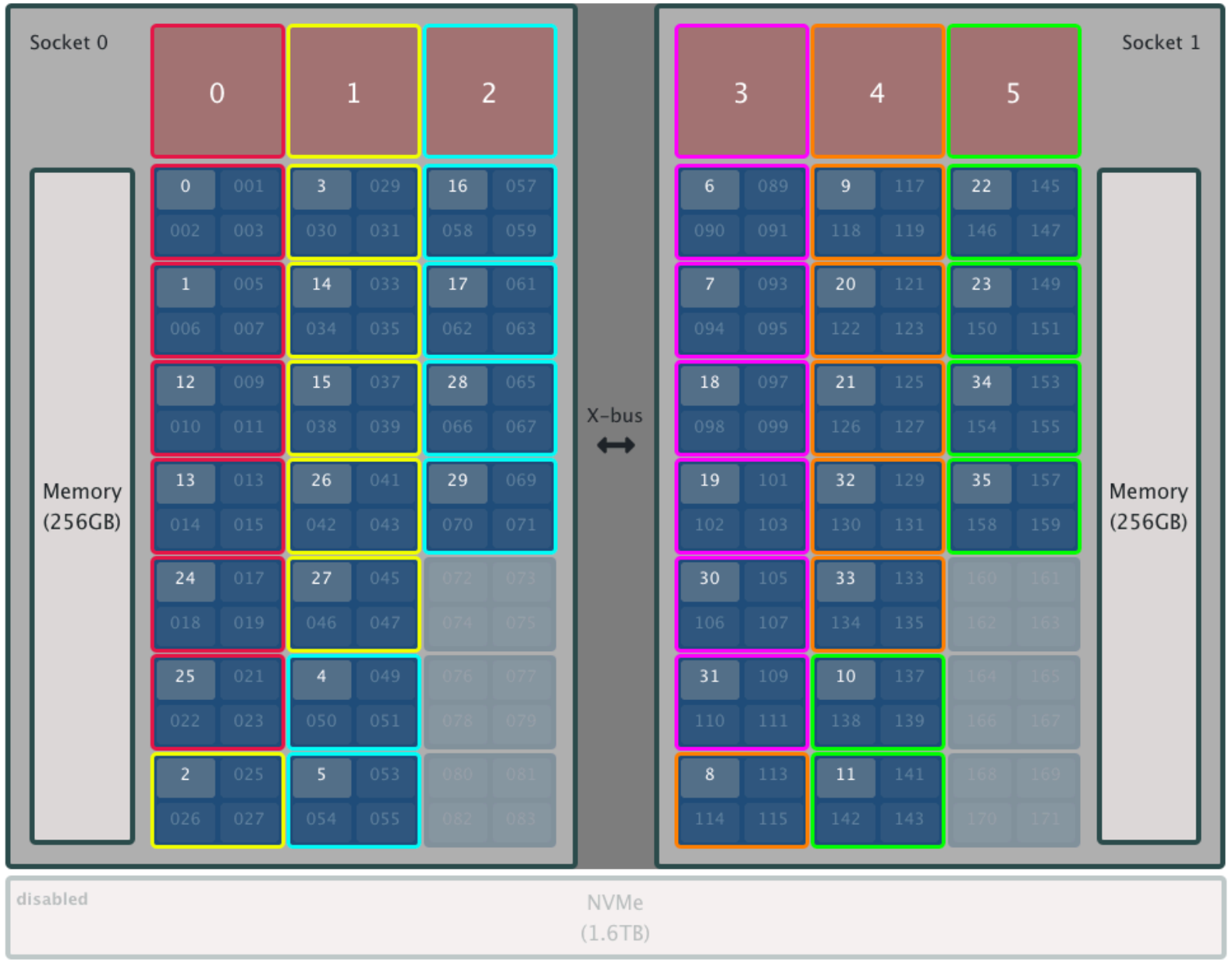
```
jsrun -n6 -c7 -g1 -a7 -dcyclic
```

- Change the order of tasks started across multiple resource sets

jsrun examples

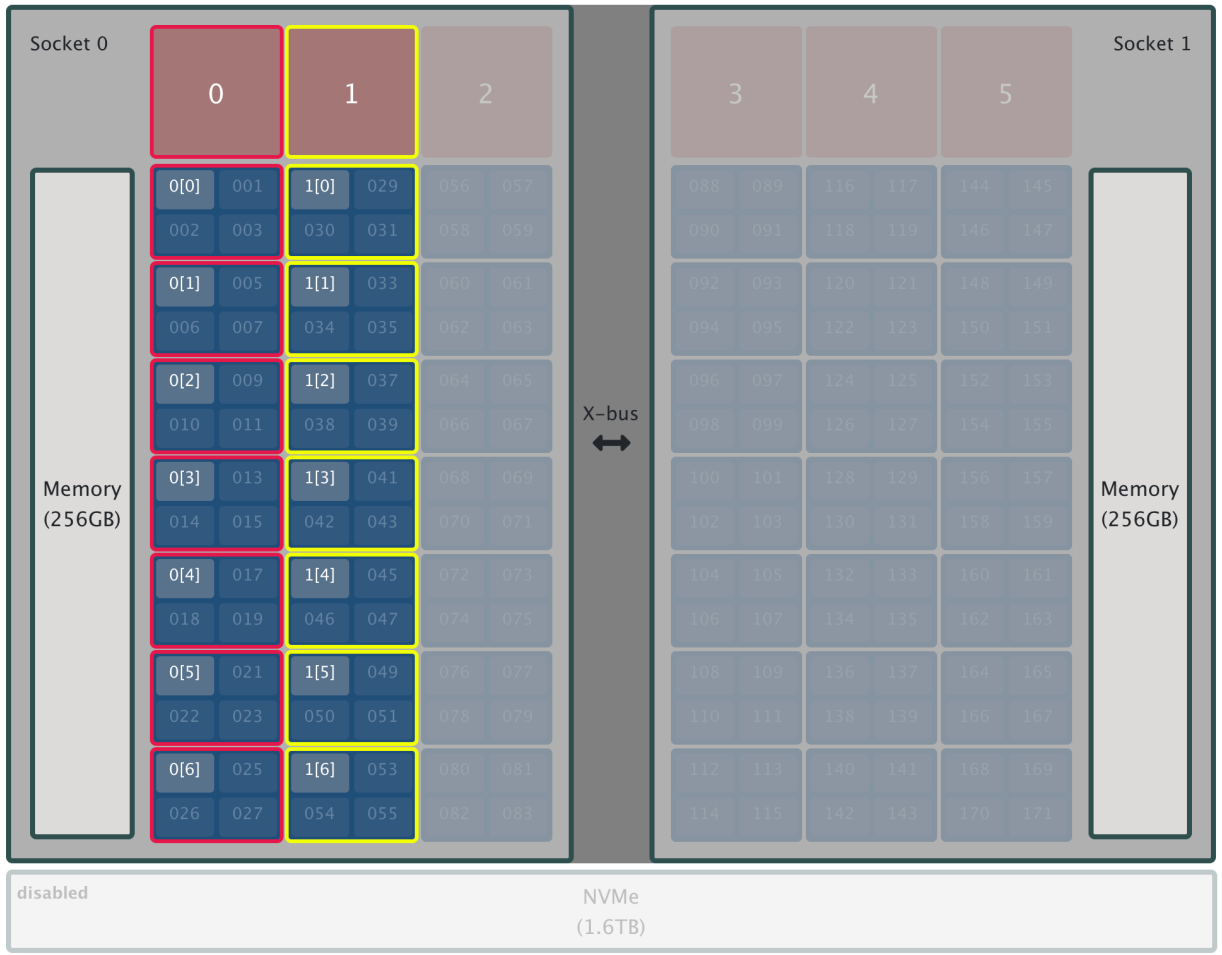
```
jsrun -n6 -c6 -g1 -a6 -dplane:2
```

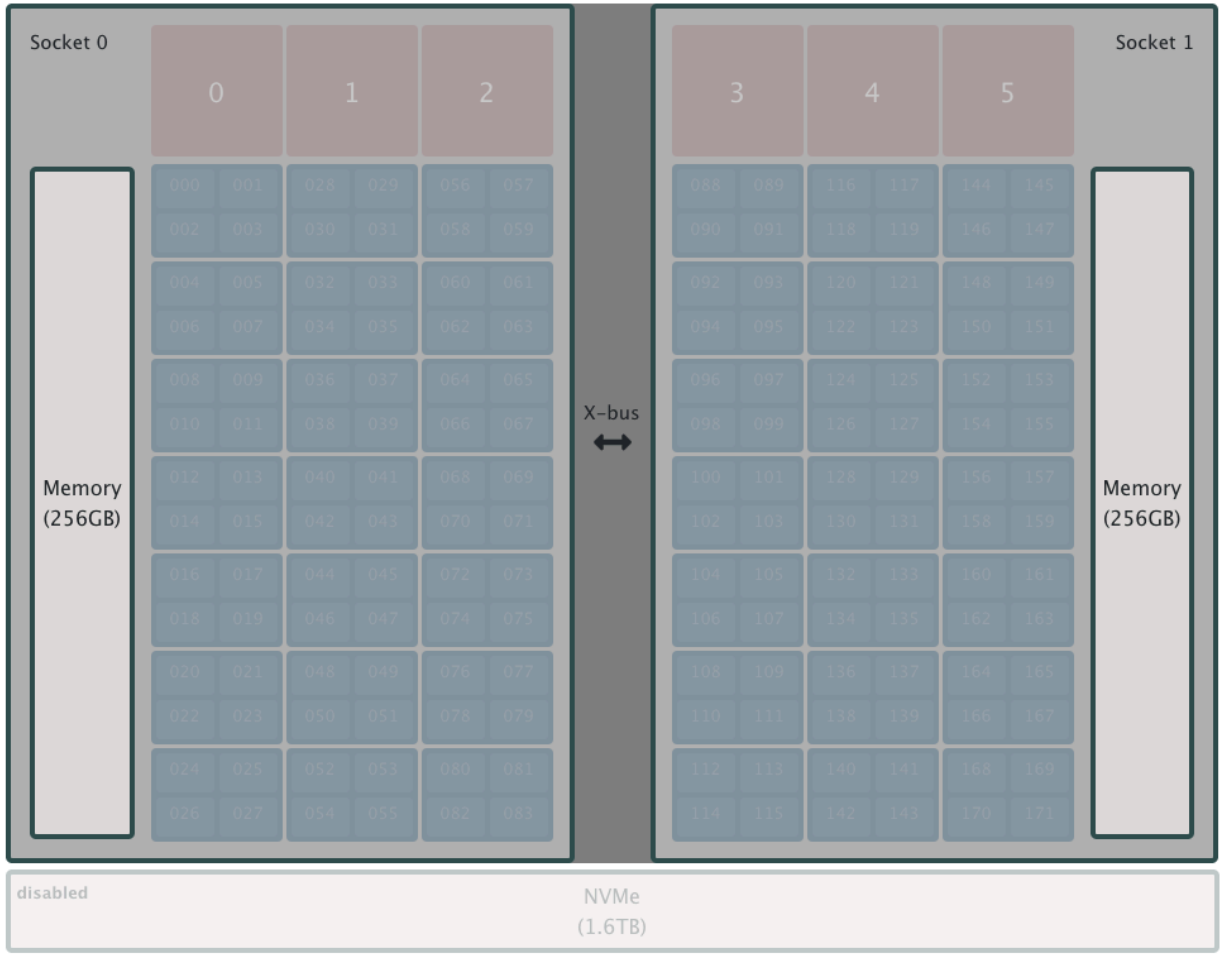
- -c# must be evenly divisible by launch distribution value -d plane:#



jsrun examples

```
jsrun -n2 -a1 -EOMP_NUM_THREADS=7 -brs -g1 -c7
```





jsrun examples

```
jsrun -n2 -r1 -a1 -EOMP_NUM_THREADS=7 -brs -g1 -c7
```

Designing a Resource Set – Basic Options

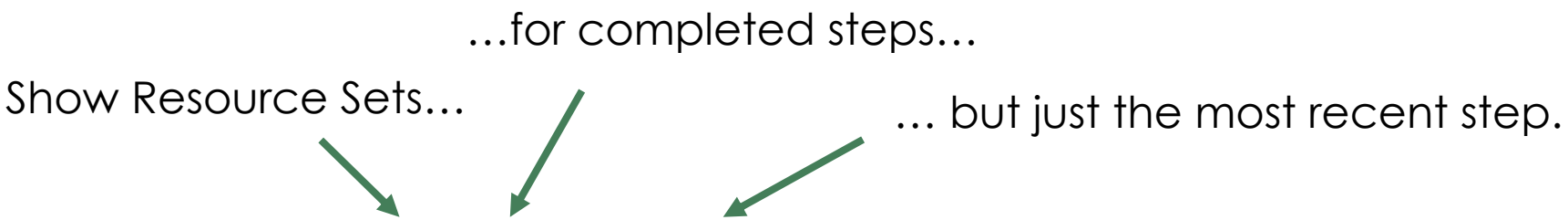
```
jsrun [-n #resource sets] [tasks, threads, and GPUs in each resource set] program [program args]
```

jsrun Flags		Description	Default Value
Long	Short		
`--nrs`	`-n`	Number of RS	All available physical cores
`--tasks_per_rs`	`-a`	Number of MPI tasks (ranks) per RS	N/A (total set instead [`-p`])
`--cpu_per_rs`	`-c`	Number of CPUs (physical cores) per RS	`1`
`--gpu_per_rs`	`-g`	Number of GPUs per RS	`0`
`--bind`	`-b`	Number of physical cores allocated per task	`packed:1`
`--rs_per_host`	`-r`	Number of RS per host (node)	N/A
`--latency_priority`	`-l`	Controls layout priorities	`gpu-cpu,cpu-mem,cpu-cpu`
`--launch_distribution`	`-d`	Order of tasks started on multiple RS	`packed`

See `man jsrun` for full list of options

`jsrun` Tools and Useful Commands

- `jslist` – list job steps are are queued, running, or completed.




```
$ jslist -R -d --last 1
     parent                    cpus       gpus       exit
   ID    ID         nrs     per RS     per RS     status           status
=============================================================================
    1     0           2         21          3          0        Complete
-----------------------------------------------------------------------------
RS 0 HOST h27n04:
       SOCKET 0:     cpus: 0-20 gpus: 0 1 2 mem: 1000
RS 1 HOST h27n04:
       SOCKET 1:     cpus: 21-41 gpus: 3 4 5 mem: 1000
-----------------------------------------------------------------------------
```

jsrun Tools and Useful Commands

- js_task_info – built-in test binary for jsrun
  - Available in default $PATH

```
$ jsrun -n1 -a1 -c1 -g1 js_task_info
Task 0 ( 0/1, 0/1 ) is bound to cpu[s] 0-3 on host h27n04 with
OMP_NUM_THREADS=4 and with OMP_PLACES={0:4} and CUDA_VISIBLE_DEVICES=0
```

- CUDA-Aware MPI – Use GPU buffers directly in MPI calls

```
$ jsrun --smpiargs="-gpu" ...
```

jsrun Tools and Useful Commands

- Hello_jsrun – "Hello World" made for jsrun
  - Quick layout iteration from the command line

```
$ git clone https://code.ornl.gov/t4p/Hello_jsrun.git
$ cd Hello_jsrun/
$ module load cuda
$ make
$ jsrun -n1 ./hello_jsrun

------ MPI Ranks: 1, OpenMP Threads: 4, GPUs per Resource Set: 0 ------
MPI Rank 000 of 001 on HWThread 002 of Node h27n04, OMP_threadID 0 of 4
MPI Rank 000 of 001 on HWThread 003 of Node h27n04, OMP_threadID 1 of 4
MPI Rank 000 of 001 on HWThread 001 of Node h27n04, OMP_threadID 3 of 4
MPI Rank 000 of 001 on HWThread 000 of Node h27n04, OMP_threadID 2 of 4
```

`jsrun` Tools and Useful Commands

(new!)

- Job Step Viewer - https://jobstepviewer.olcf.ornl.gov/
  - Generate a graphical view of an application's runtime layout on Summit.
  - Used to create all resource set images in this presentation

```
$ module load job-step-viewer
$ jsrun -n6 -g1 -c7 -a1
https://jobstepviewer.olcf.ornl.gov/summit/926674-5
```

1. Load the `job-step-viewer` module.
2. Test out a `jsrun` line by itself or run an executable as normal.
3. Visit the provided URL.

Questions?

OAK RIDGE National Laboratory

jsrun Basics

Jack Morrison

User Assistance and Outreach Group
Oak Ridge Leadership Computing Facility

February 18, 2020

U.S. DEPARTMENT OF ENERGY