Hierarchical Resources (HRES) and topology/block

Ben Glines
Slurm Support and Development Engineer

slurm | SCHEDMD

- Do you have jobs with very strict topology requirements?
- Do you want to make sure jobs are getting nodes within the same NVLink domain?
- Do you have any scarce resources (e.g. network, storage, NVIDIA SHARP trees, etc.) tied to groups of nodes that you want Slurm to “limit”?
- Do you just want to learn more about Slurm?

HRES and topology/block could make your wildest cluster configuration dreams come true!

Overview

- Basic overview of Slurm node selection
- Limitations of topology/tree plugin
- HRES examples and configuration
- topology/block examples and configuration

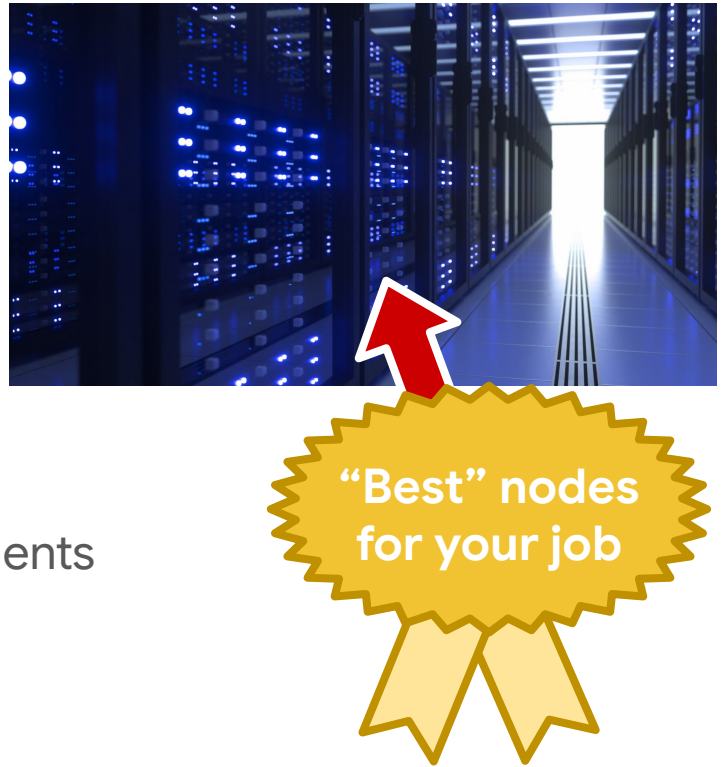
Overview

- Basic overview of Slurm node selection
- Limitations of topology/tree plugin
- HRES examples and configuration
- topology/block examples and configuration

Slurm Node Selection

When a job is ready for allocation:

- slurmctld builds a list of nodes that *could* fulfill job's requirements
- slurmctld determines if there are available nodes for the job, and selects the "best" nodes.



Hierarchical resources (HRES) and the **topology/block** plugin provide strict requirements for what nodes will be selected for a job.

Overview

- Basic overview of Slurm node selection
- Limitations of topology/tree plugin
- HRES examples and configuration
- topology/block examples and configuration

topology/tree

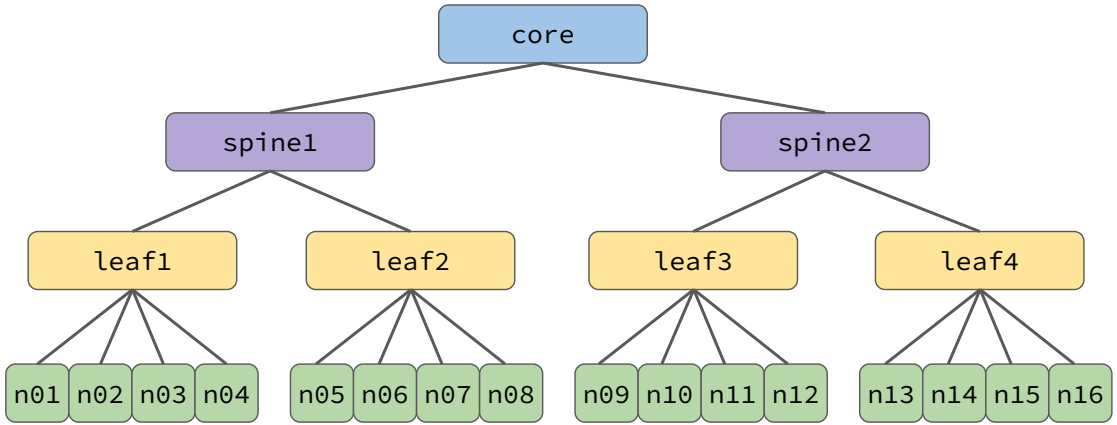
Find “best fit” nodes for a job

- Try to minimize leaf switches
- Try to minimize hops
- Enforce `--switches`

```
SwitchName=core Switches=spine[1-2]
SwitchName=spine1 Switches=leaf[1-2]
SwitchName=spine2 Switches=leaf[3-4]
SwitchName=leaf1 Nodes=n[01-04]
SwitchName=leaf2 Nodes=n[05-08]
SwitchName=leaf3 Nodes=n[09-12]
SwitchName=leaf4 Nodes=n[13-16]
```

Limitations:

- Topology optimization is not paramount
- No arbitrary resource limits
- No "exclusive" switch access



Overview

- Basic overview of Slurm node selection
- Limitations of topology/tree plugin
- HRES examples and configuration
- topology/block examples and configuration

HRES Overview

Hierarchical resources allow limiting resources tied to groups of nodes

- **HRES** are intentionally generic, allowing configuration for your needs:
  - Scarce network/storage resources
  - NVIDIA SHARP trees
  - Power capping on different levels

- Resources configured independently of each other and of any topology plugin
- **No enforcement on actual resource usage**

HRES Configuration via resources.yaml

```
- resource: power
  mode: MODE_3
  layers:
    - nodes:
        - "node[01-08]"
      count: 60
    - nodes:
        - "node[09-16]"
      count: 60
    . . .
```

HRES Planning Modes

MODE_1

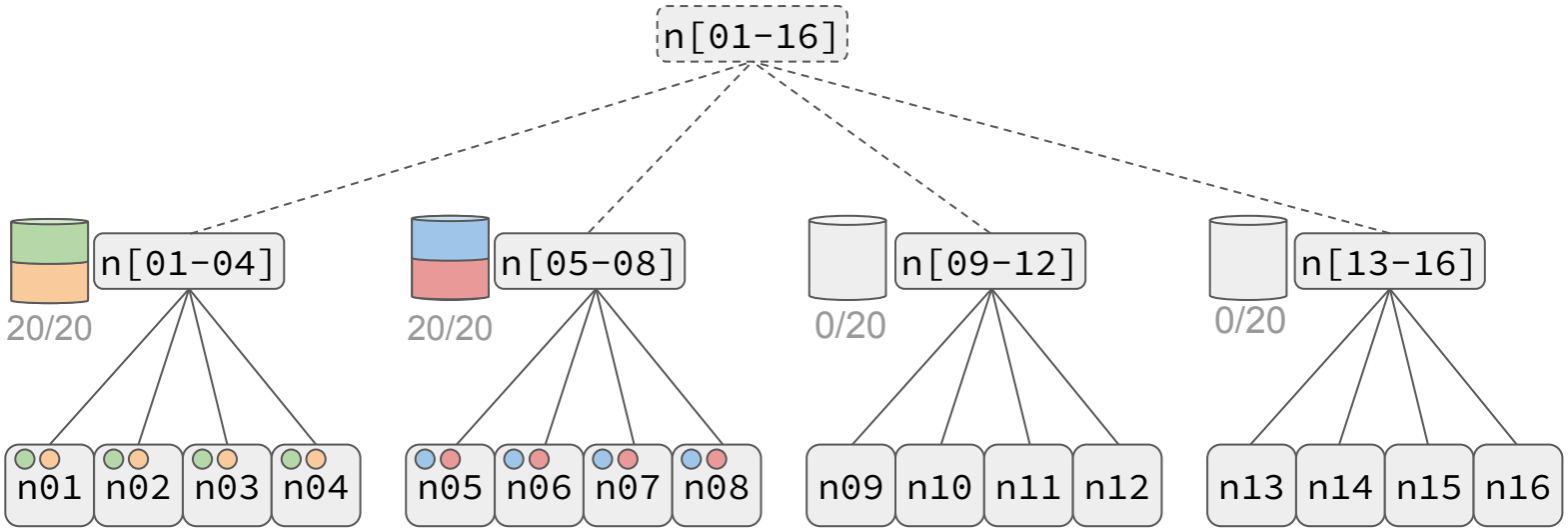
Resources only required on one layer

- Only a single level may be needed, multiple levels may be defined for flexibility
- “`count: 0`” will have no impact on scheduling
- “`count: -1`” will always allow a job requesting that resource to succeed

HRES Mode 1

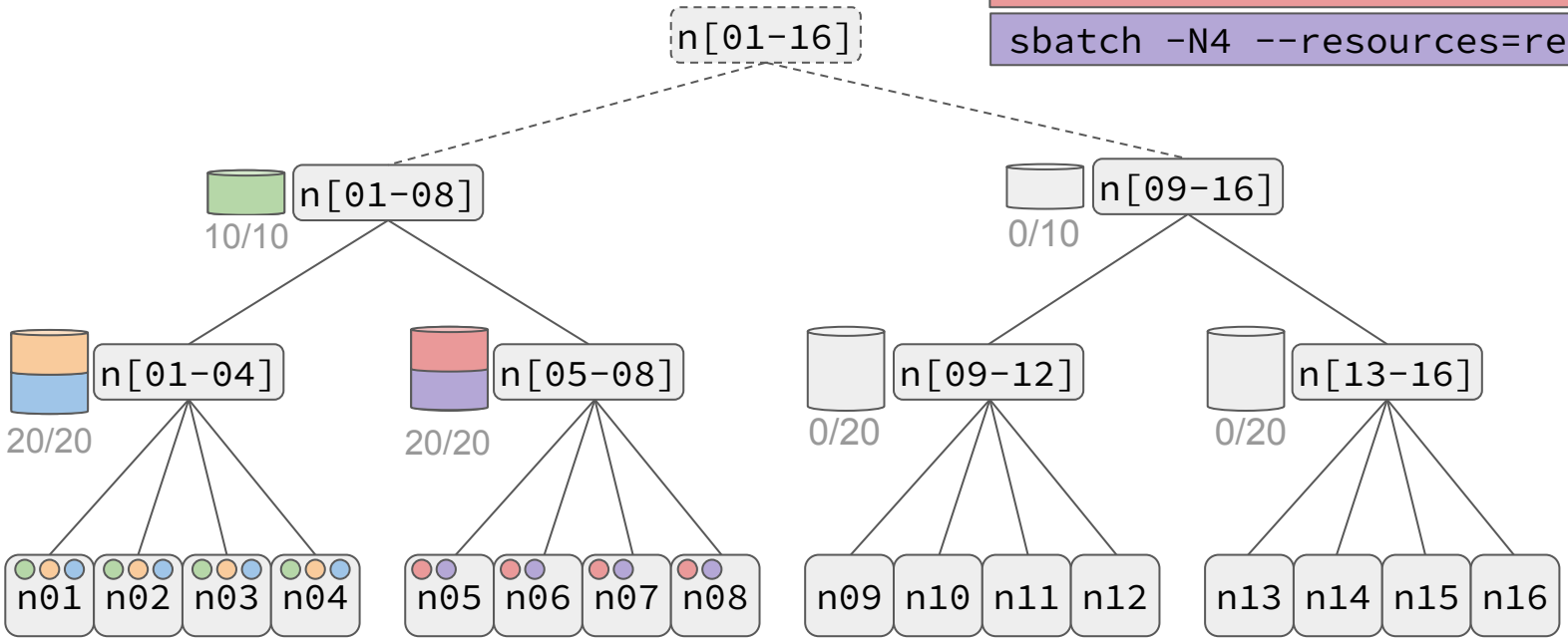
Resources only required on one layer

```
sbatch -N4 --resources=res_1:10
sbatch -N4 --resources=res_1:10
sbatch -N4 --resources=res_1:10
sbatch -N4 --resources=res_1:10
```



HRES Mode 1 (multiple levels)

```
sbatch -N4 --resources=res_1:10
sbatch -N4 --resources=res_1:10
sbatch -N4 --resources=res_1:10
sbatch -N4 --resources=res_1:10
sbatch -N4 --resources=res_1:10
```



HRES Planning Modes

MODE_2

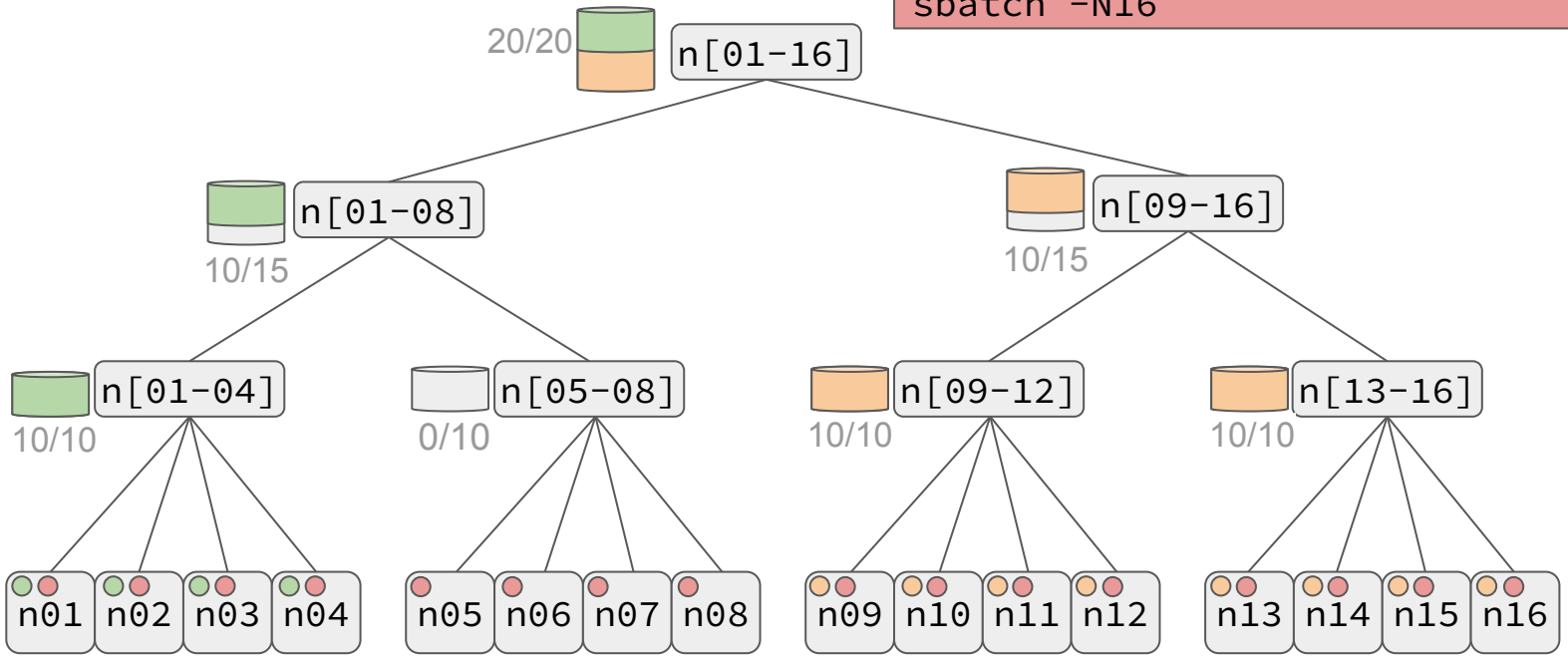
Resources required on all layers.

- Same number of resources is required in each layer. E.g. if a job requests 10 of a resource, then each layer (group of nodes) needs to have 10 resources available
- “`count: 0`” marks that resource unavailable to all nodes in that layer
- “`count: -1`” will have no impact on scheduling

HRES Mode 2

Resources required by each layer

```
sbatch -w n[01-04] --resources=res_2:10
sbatch -w n[09-16] --resources=res_2:10
sbatch -w n[05-08] --resources=res_2:10
sbatch -N16
```



HRES Planning Modes

MODE_3

Resources are allocated per-node (not per-layer) and consumed in all encompassing higher layers

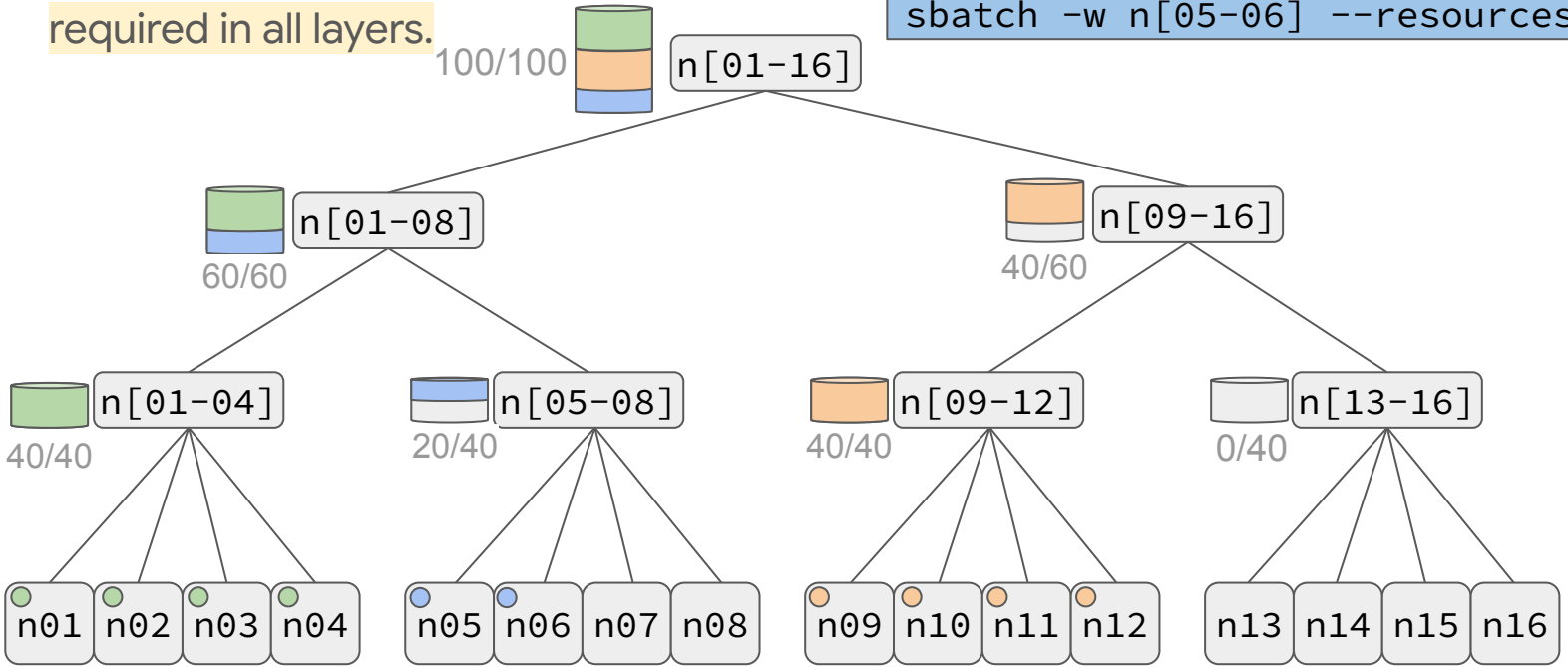
- The driving use case for this model is power capping - where, e.g., adequate power needs to be allocated at the chassis level, then rack level summing multiple chassis, then datacenter row, and finally at the cluster-wide level.

NOTE: Layers defined under Mode 3 must be able to be represented as a rooted uniform depth tree.

HRES Mode 3

Resources allocated per-node and required in all layers.

```
sbatch -w n[01-04] --resources=power:10
sbatch -w n[09-12] --resources=power:10
sbatch -w n[13-16] --resources=power:10
sbatch -w n[05-06] --resources=power:10
```



HRES resources.yaml Convenience Options

- `base` - Subtracted from layer's count

```
- nodes:
  - "node[01-32]"
    count: 130000
    base:
      - name: network
        value: 10000
      - name: storage
        value: 5000
```

- `variables` - Predefined counts of resources used at job submission
  - e.g. `sbatch -N4 --resources=full_node == sbatch -N4 --resources=1000`

```
- resource: power
  mode: MODE_3
  variables:
    - name: full_node
      value: 1000
    - name: full_gpu_node
      value: 2000
```

HRES Limitations

- The resource counts cannot be dynamically changed
- Dynamic nodes are not supported
- Resource names defined through the hierarchical resources configuration must not conflict with any cluster licenses

Overview

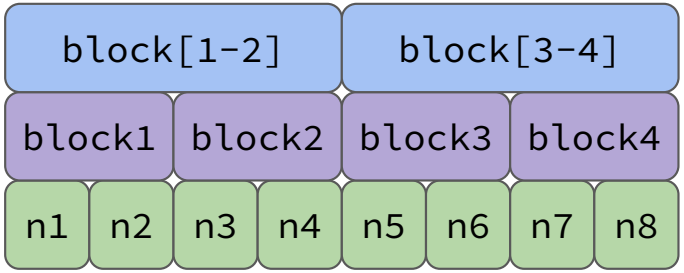
- Basic overview of Slurm node selection
- Limitations of topology/tree plugin
- HRES examples and configuration
- topology/block examples and configuration

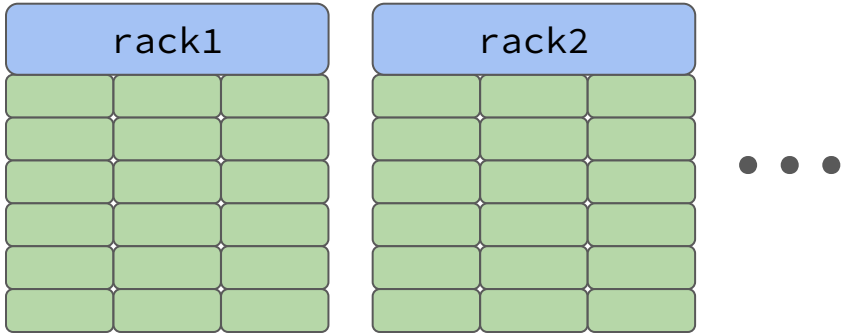
topology/block

Strictly enforce node placement based on hierarchical block structure

- e.g. NVLink domains
- Fragmentation is the enemy
- Exchange “cluster utilization” for better job performance

Several job submission options for flexibility/extra enforcement





From SLUG 24: “NVIDIA - Gaining more control over node scheduling with the Topology/Block Plugin”

https://slurm.schedmd.com/SLUG24/NVIDIA-Craig_Tierney.pdf

(can be found in https://www.schedmd.com/publications/)

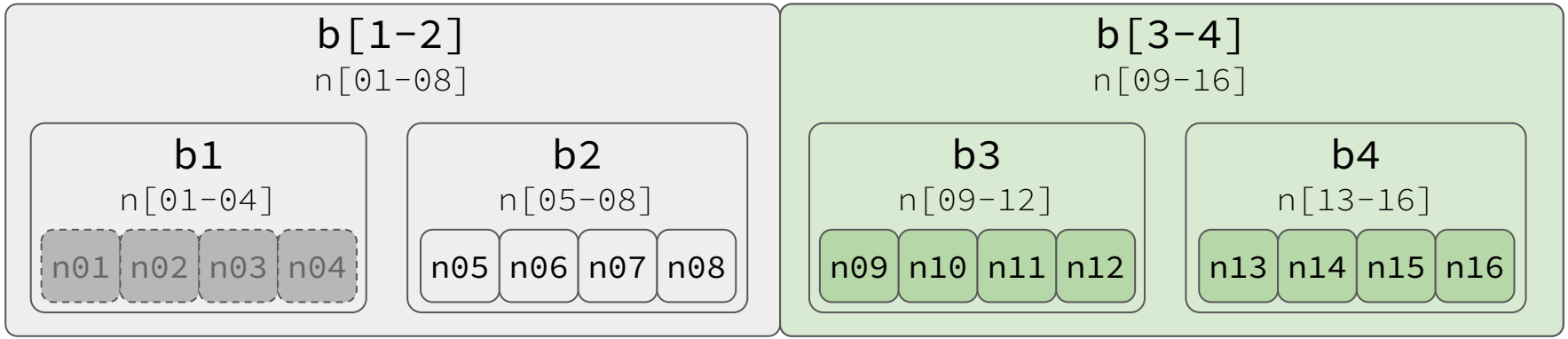
topology.yaml for topology/block plugin

```
- topology: topo1
  cluster_default: true
  block:
    block_sizes:
      - 4
      - 8
    blocks:
      - block: b1
        nodes: node[01-04]
      - block: b2
        nodes: node[05-08]
      - block: b3
        nodes: node[09-12]
      - block: b4
        nodes: node[13-16]
```

topology/block

- Job is placed on b[3-4] since b[1-2] does not have 8 available nodes.

```
n[01-04,n13] are busy
sbatch -N8
n13 becomes available
```



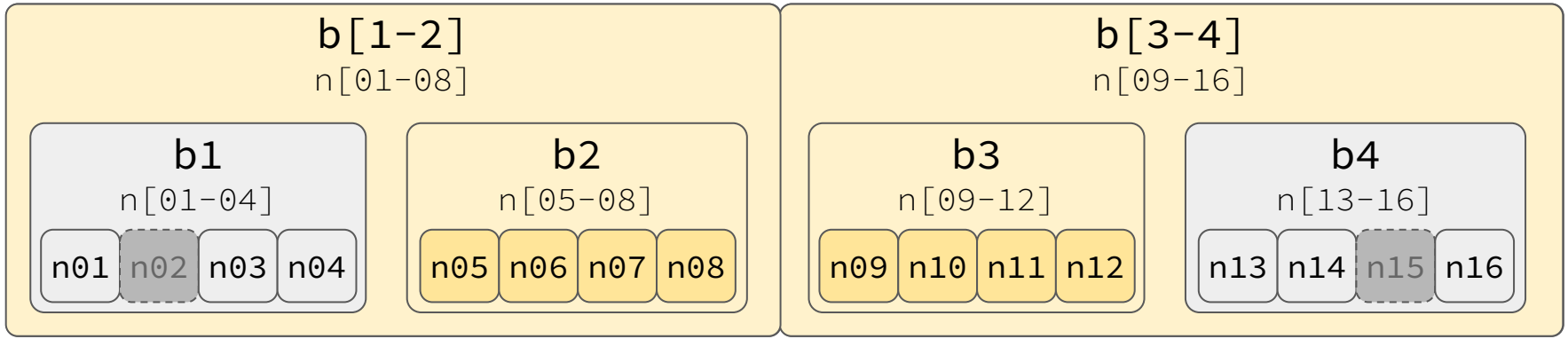
topology/block

`--segment=<segment_size>`

- Number of nodes in each "segment"

```
n[02,15] are busy
sbatch -N8
sbatch -N8 --segment=4
```

- The first job is pending because no single higher-order block has 8 available nodes.
- The second job is placed on b2 and b3 (separate higher-order blocks) due to the flexibility provided by `--segment`.



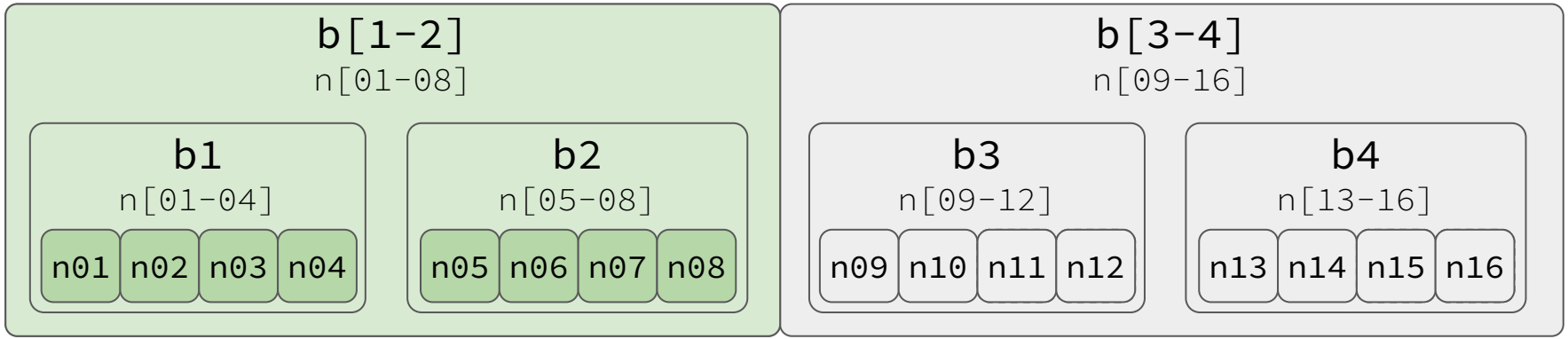
topology/block

`--consolidate-segments`

- Ensure all segments are consolidated into one higher-level aggregated block.
- Enforce with job_submit

```
n[03-04,07-08,11-12,15-16] are busy
sbatch -N8 --segment=2 --consolidate-segments
n[03-04,07-08] become available
n[11-12,15-16] become available
```

- `--consolidate-segments` may delay job start, but ultimately can result in less fragmentation.



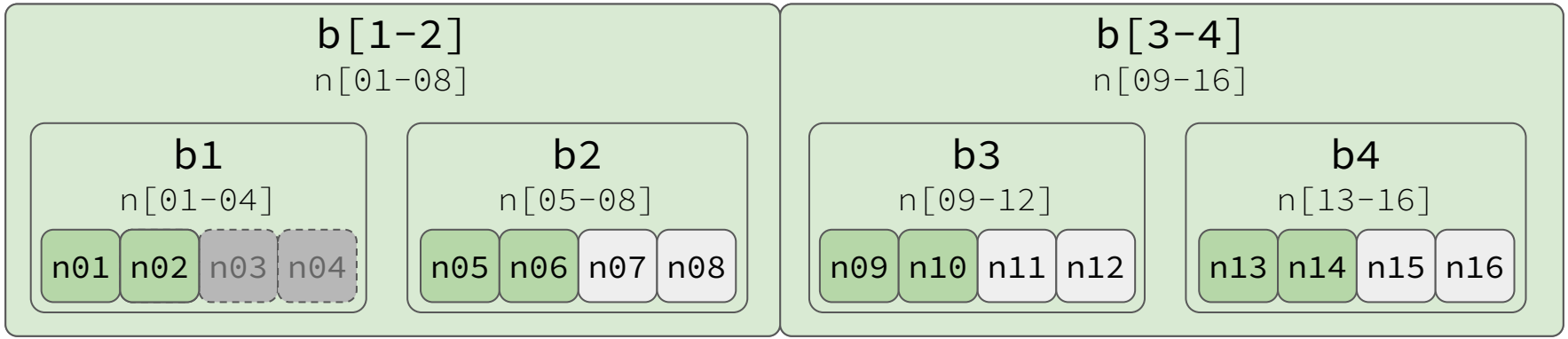
topology/block

`--spread-segments`

- Prevents nodes within the same base block from being allocated to separate segments within the same block.

```
n[02-04] are busy
sbatch -N8 --segment=2 --spread-segments
n02 becomes available
```

- `--spread-segments` ensures that each segment is on a separate block in a case which normally segments would be sharing blocks.



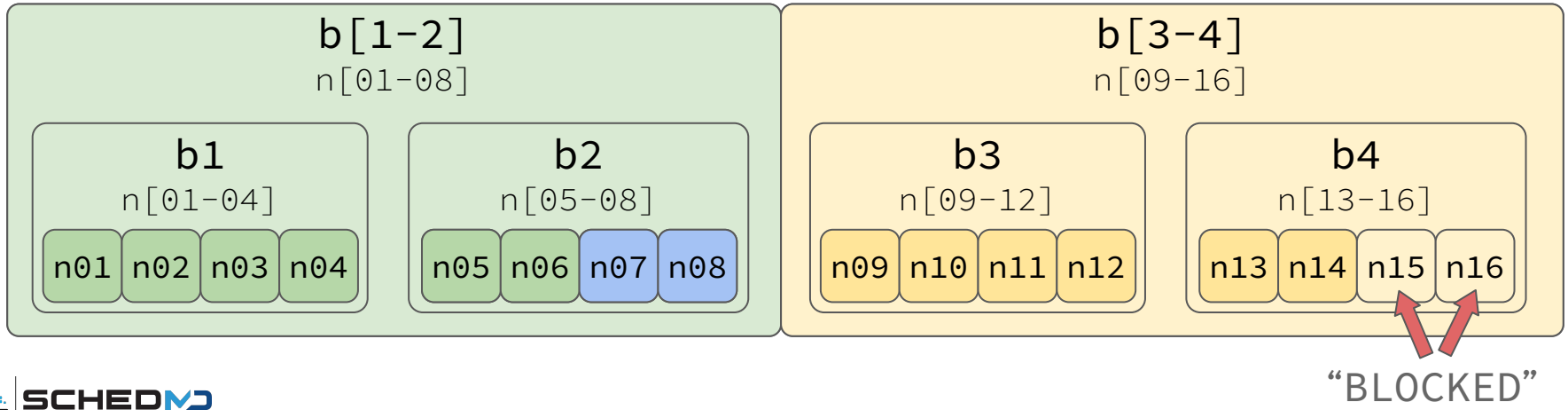
topology/block

`--exclusive=topo`

- Prevents other jobs from running on any block that is allocated to this job. Unused nodes are marked as "BLOCKED"

```
sbatch -N6
sbatch -N2
sbatch -N6 --exclusive=topo
sbatch -N2
```

- The job with `--exclusive=topo` prevents other jobs from running on the same block.



Questions?


SCHEDMD
The Slurm Company