PINNING WITH PSSLURM

Installation of new default 12th May 2020

MAY 2020 I SC SUPPORT

TERMINOLOGY

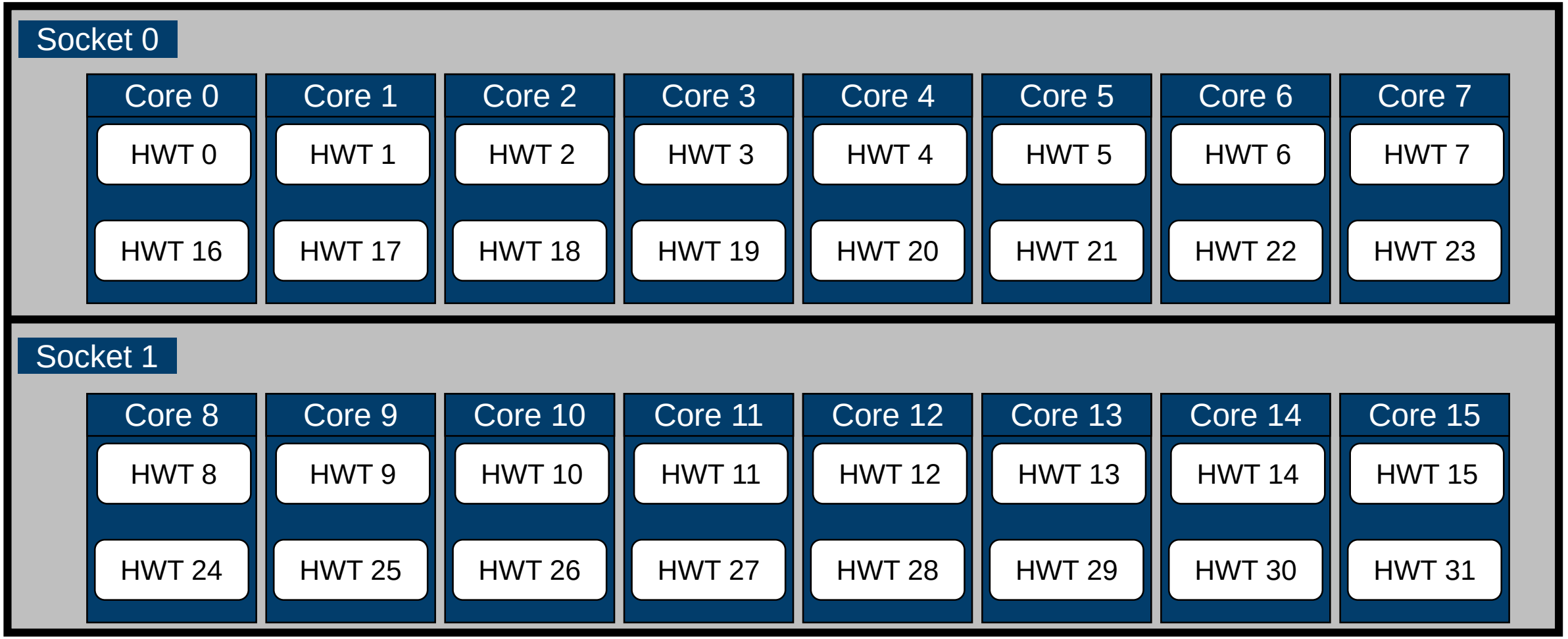
- **Thread**
  - One CPU thread
- **Task**
  - Part of a job consisting of a number of requested CPU threads (specified by -c, --cpus-per-task). Usually this is a process
- **Core**
  - Consists of a number of CPU threads sitting on the same physical CPU core and thus sharing caches (traditional name of the second memory locality domain)
- **Socket**
  - Consists of a number of CPU threads with the same memory locality (traditional name of the top most memory locality domain)

USED HARDWARE

For the purpose of demonstration
our internal test system JUROPA-3 has been used.
The main difference to JUWELS, JURECA and JUSUF is
the amount of cores per socket.
The rules for the processor affinity are the same on all four systems.

JUROPA-3 NODE

2 Intel Xeon E5-2650 (Sandy Bridge-EP) eight-core processors (SMT)



SLURM OPTIONS

Overview

The options to **srun** meant here are:

- **--cpu-bind**=
  [{quiet,verbose},none|rank|map_cpu:<list>|mask_cpu:<list>|rank_ldom|map_ldom:<list>|
  mask_ldom:<list>|sockets|cores|threads|ldoms|boards]

- **--distribution/-m**=*|block|cyclic|arbitrary|plane=<options>
  [:*|block|cyclic|fcyclic[:*|block|cyclic|fcyclic]][,Pack|NoPack]

- **--hint**=nomultithread

SLURM OPTIONS

--cpu-bind

- **General syntax:**
  - --cpu-bind=[{quiet,verbose},]type
- **Report binding before task runs:**
  - [{quiet,verbose}]
  - Default: quiet
- **Type:**
  - Explicit**:**
    - map_cpu:<list> | mask_cpu:<list> | map_ldom:<list> | mask_ldom:<list>
  - Implicit**:**
    - none | rank | rank_ldom | sockets | cores | threads | ldoms | boards

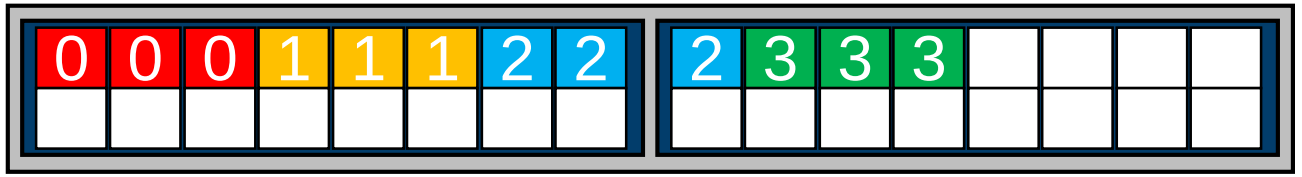
Hint: task n+1 will be placed depending on the last thread of task n

SLURM OPTIONS

--cpu-bind

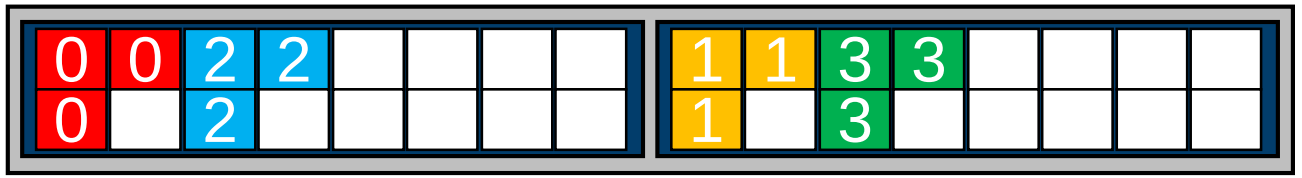
- 1st example:
  - **rank**: Each task is pinned to as many threads as it requests, just filling cores round-robin. Spread the threads and tasks to as many cores as possible. This type is not influenced by the second and third part of the *--distribution* option. (default until 12th May 2020)

    --ntasks-per-node=4
    --cpus-per-task=3
    --cpu-bind=rank



  - **threads**: Each task is pinned to as many threads as it requests. Which threads each process gets is controlled by the *--distribution* option.

    --ntasks-per-node=4
    --cpus-per-task=3
    --cpu-bind=threads
    --distribution=:*:*cyclic



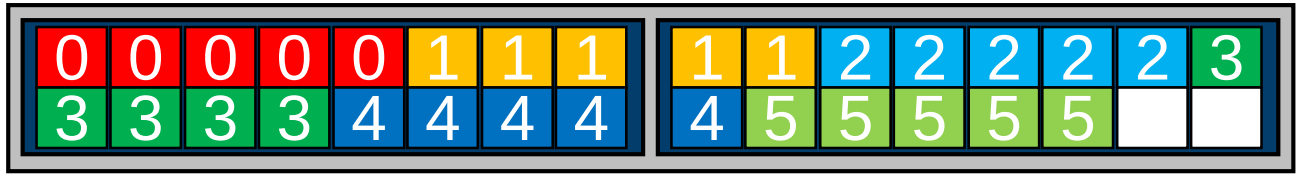
Hint: number = task id

SLURM OPTIONS

--cpu-bind

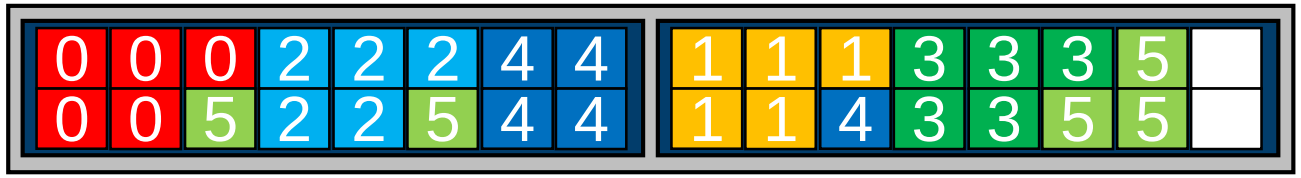
- 2nd example “full node”:
  - **rank**: (default until 12th May 2020)

```
--ntasks-per-node=6
--cpus-per-task=5
--cpu-bind=rank
```



  - **threads**:

```
--ntasks-per-node=6
--cpus-per-task=5
--cpu-bind=threads
--distribution=*:*:cyclic
```



Hint: number = task id

SLURM OPTIONS

--distribution/-m

Has up to four parts separated by colon and comma:

1. ***|block|cyclic|arbitrary|plane=<options>***
   - controls the distribution of tasks to nodes (Default: block)
2. ***[:*|block|cyclic|fcyclic***
   - controls the distribution of tasks over sockets inside one node; done by *psslurm*
   - Default: cyclic
3. ***[:*|block|cyclic|fcyclic]]***
   - controls the distribution of tasks over cores inside one node; done by *psslurm*
   - Default: inherited from the second part
4. ***[,Pack|NoPack]***
   - optional control for task distribution over nodes

OPEN ISSUE

Hint: The following slides are based on the proposed default: *--cpu-bind=threads*

SLURM OPTIONS

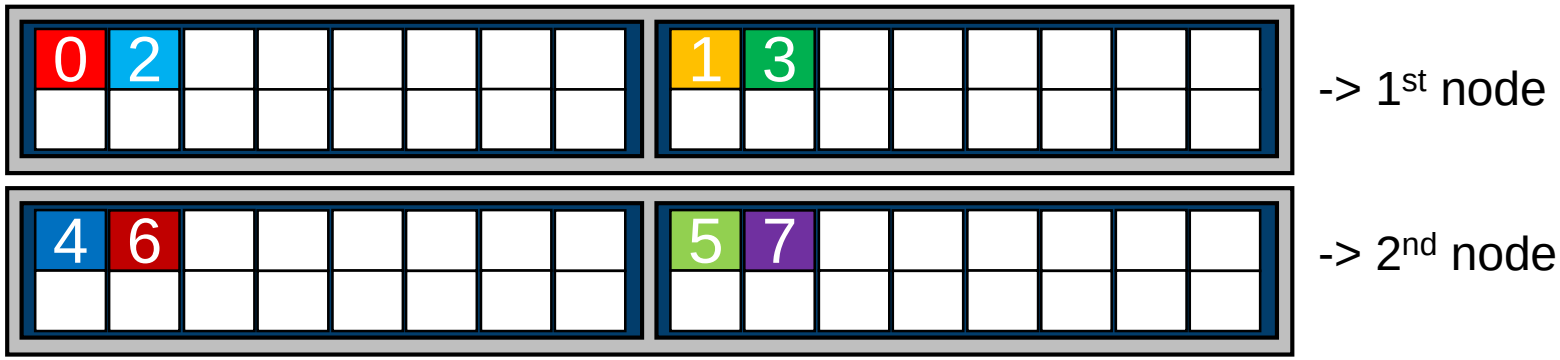
--distribution/-m = *|*block*|*cyclic*|*arbitrary*|*plane=<options>[:*:*][,Pack|NoPack]*

1. (& 4. part) (nodes):

- ***:** The default is *block*
- **block**: Distribute tasks to a node such that consecutive tasks share a node
  - 1st example:

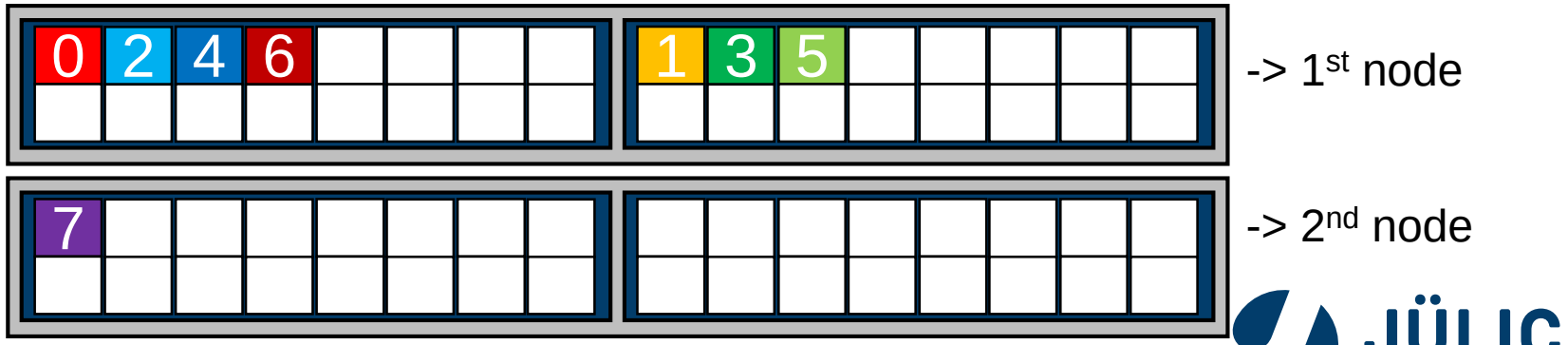
    --nodes=2
    --ntasks=8
    --cpus-per-task=1
    --cpu-bind=threads

    [--distribution=*:*:*,NoPack]



  - 2nd example:

    --nodes=2
    --ntasks=8
    --cpus-per-task=1
    --cpu-bind=threads
    --distribution=*:*:*,Pack



Hint: number = task id

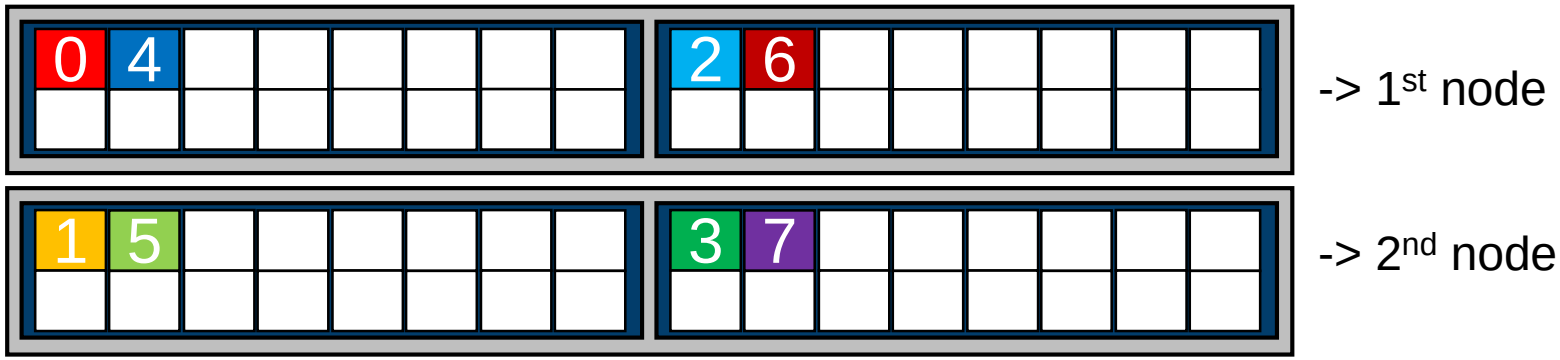
SLURM OPTIONS

--distribution/-m = **|block|cyclic|arbitrary|plane=<options>[:*:*,*]*

1. part (nodes):

- **cyclic**: Distribute tasks to a node such that consecutive tasks are distributed over consecutive nodes (in a round-robin fashion)
  - example:

    --nodes=2
    --ntasks=8
    --cpus-per-task=1
    --cpu-bind=threads
    --distribution=cyclic



- **arbitrary**: see https://slurm.schedmd.com/srun.html
- **plane=<options>**: see https://slurm.schedmd.com/dist_plane.html

Hint: number = task id

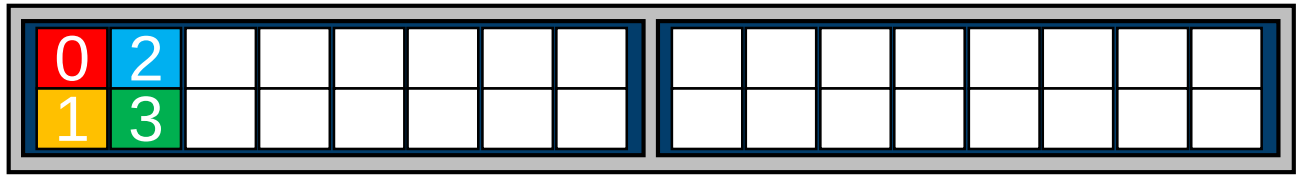
SLURM OPTIONS

--distribution/-m = **:[:*|block|cyclic|fcyclic[:*]],**

2. part (sockets):

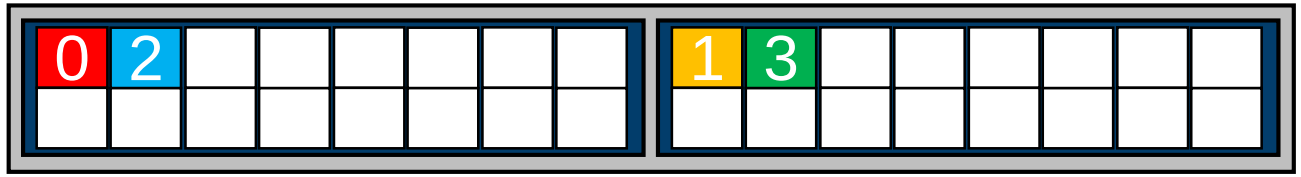
- ***:** The default is to cycle over sockets, see cyclic
- **block**: Each socket is first filled with tasks before the next socket will be used.

```
--ntasks-per-node=4
--cpus-per-task=1
--cpu-bind=threads
--distribution=*:block:block
```



- **cyclic:** Each task will be assigned to the next socket(s) in a round-robin fashion.

```
--ntasks-per-node=4
--cpus-per-task=1
--cpu-bind=threads
--distribution=*:*
```



- **fcyclic:** Each thread inside a task will be assigned to the next socket in a round-robin fashion, spreading the task itself most possibly over all sockets. *Fcyclic* implies *cyclic*.

Hint: number = task id

SLURM OPTIONS

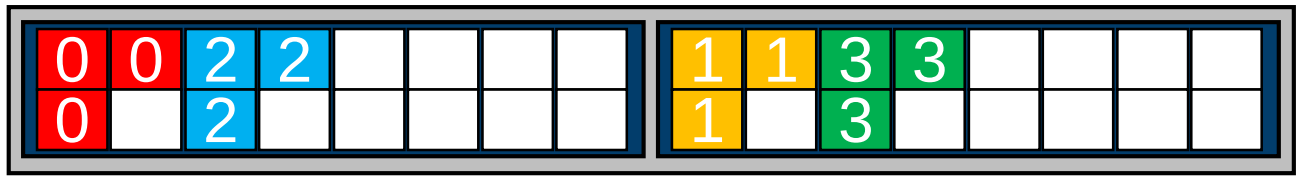
--distribution/-m = **:*[:*|block|cyclic|fcyclic]],**

3. part (cores):

- ***:** *fcyclic*
- **cyclic**: Each task will be assigned to the next core(s) in a round-robin fashion. The threads of a task will fill the cores.
  - 1st example:

    --ntasks-per-node=4
    --cpus-per-task=3
    --cpu-bind=threads

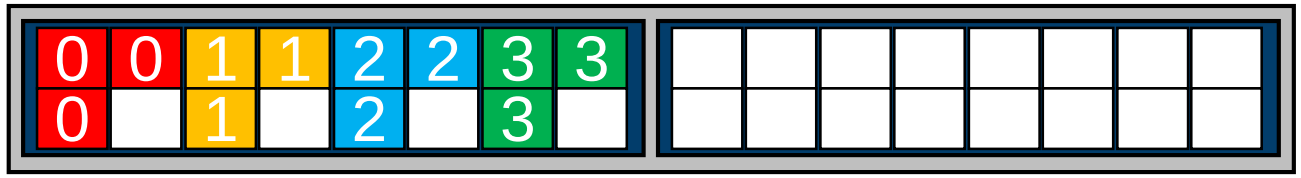
    --distribution=*:*:cyclic



  - 2nd example:

    --ntasks-per-node=4
    --cpus-per-task=3
    --cpu-bind=threads

    --distribution=*:block:cyclic



Hint: number = task id

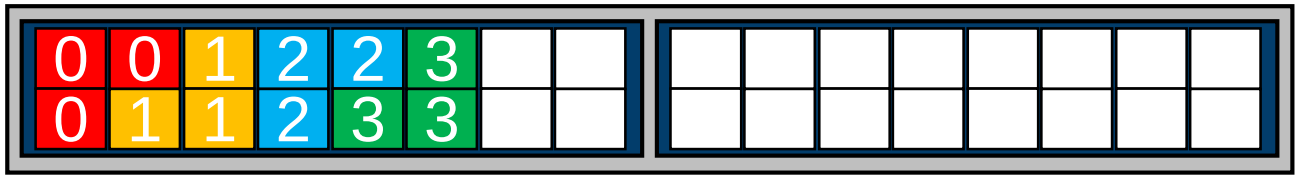
SLURM OPTIONS

--distribution/-m = **:*[:*|block|cyclic|fcyclic]],**

3. part (cores):

- **block**: Each core is first filled with tasks before the next core will be used.
  - 1st example:

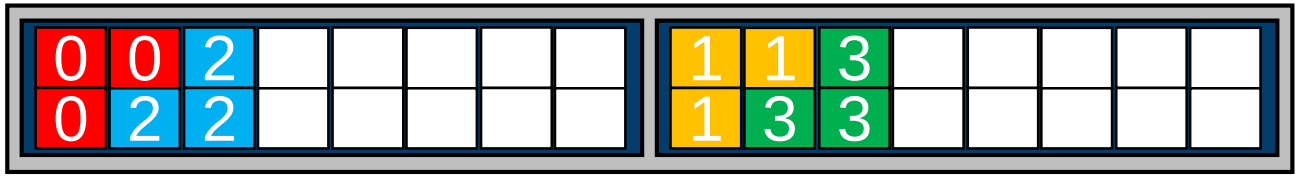
    --ntasks-per-node=4
    --cpus-per-task=3
    --cpu-bind=threads
    --distribution=*:block:cyclic



  - 2nd example:

    --ntasks-per-node=4
    --cpus-per-task=3
    --cpu-bind=threads
    --distribution=*:*:block

Hint: same as *--cpu-bind=rank_ldom*



Hint: number = task id

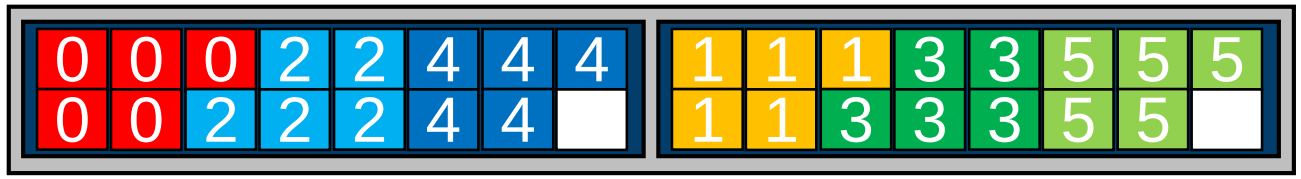
SLURM OPTIONS

--cpu-bind

- 3rd example “full node”**:**
  - **block**:

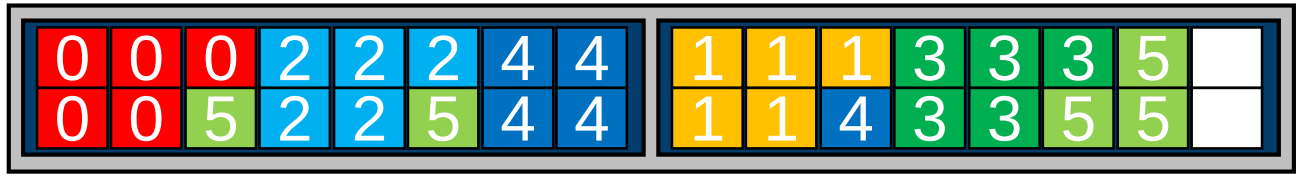
    --ntasks-per-node=6
    --cpus-per-task=5
    --cpu-bind=threads
    --distribution=*:*:block

    Hint: same as *--cpu-bind=rank_ldom*



  - **cyclic** (repetetion):

    --ntasks-per-node=6
    --cpus-per-task=5
    --cpu-bind=threads
    --distribution=*:*:cyclic



Hint: number = task id

SLURM OPTIONS

--distribution/-m = **:*[:*|block|cyclic|fcyclic]],**

hint: --cpu-bind=threads

3. part (cores):

- **fcyclic:** Each thread inside a task will be assigned to the next core in a round-robin fashion, spreading the task itself most possibly over all cores. *Fcyclic* implies *cyclic*.
  - 1st example:

    --ntasks-per-node=4
    --cpus-per-task=3

    --distribution=*:fcyclic

  - 2nd example:

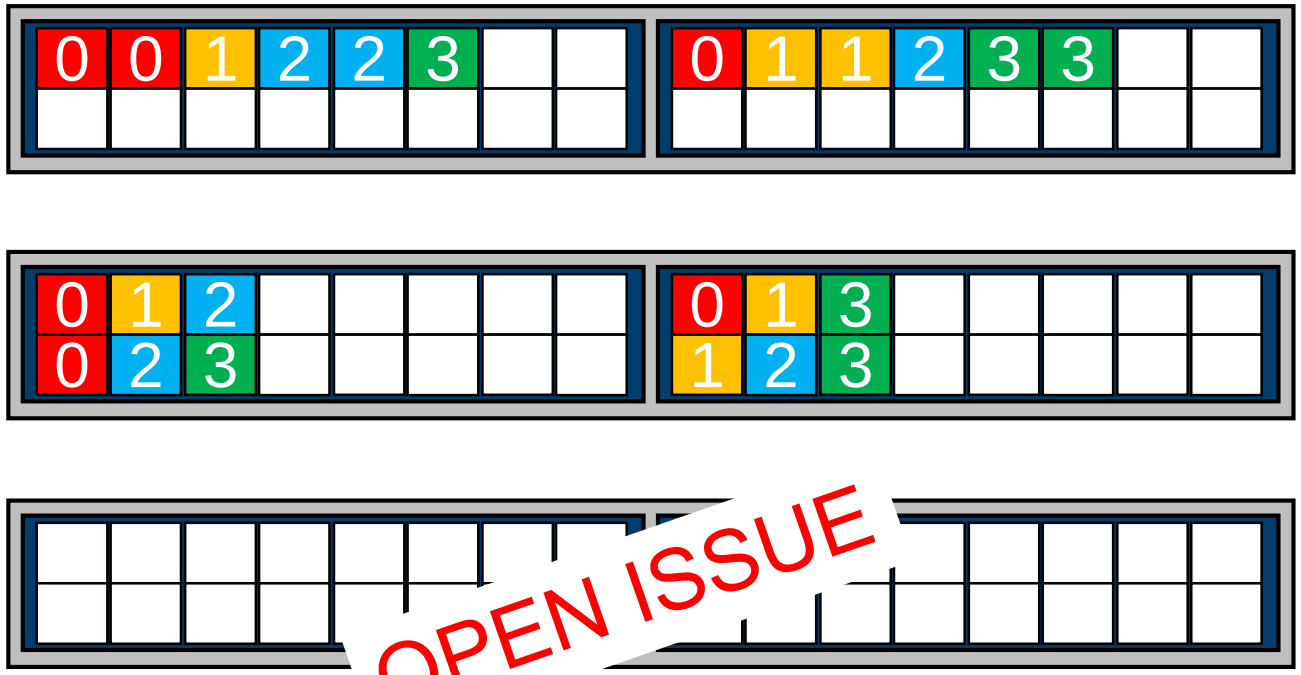
    --ntasks-per-node=4
    --cpus-per-task=3

    --distribution=*:fcyclic:block

  - 3rd example:

    --ntasks-per-node=4
    --cpus-per-task=3

    --distribution=*:fcyclic:cyclic



Hint: number = task id

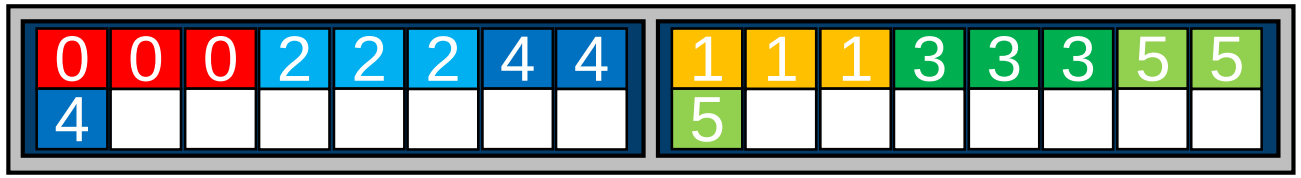
SLURM OPTIONS

--distribution/-m = **:*[:*|block|cyclic|fcyclic]],**

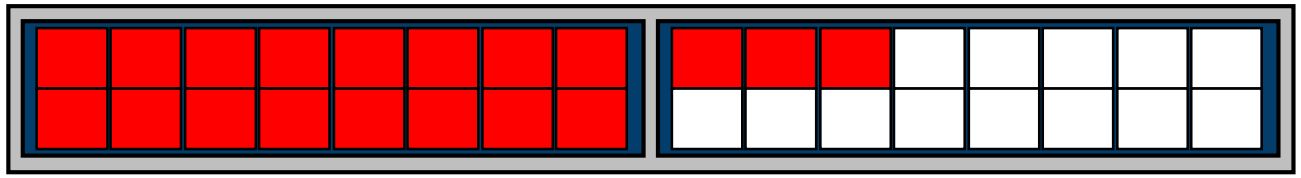
hint: --cpu-bind=threads

- 4th example (fcyclic):

  --ntasks-per-node=6
  --cpus-per-task=3
  --distribution=*:*:fcyclic

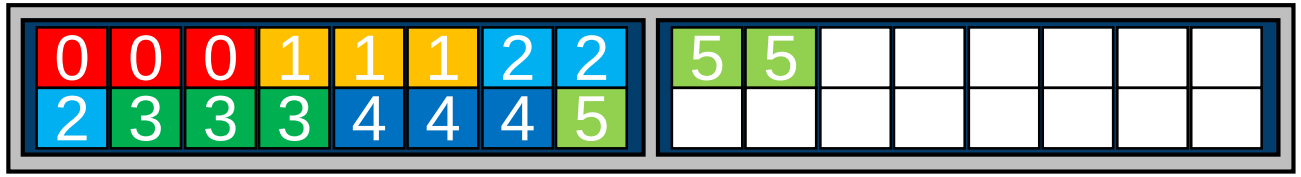


  --ntasks-per-node=1
  --cpus-per-task=19
  --distribution=*:*:fcyclic

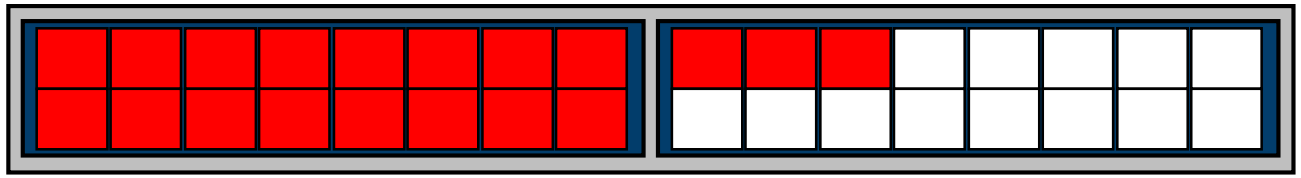


- 5th example (fcyclic):

  --ntasks-per-node=6
  --cpus-per-task=3
  --distribution=*:block:fcyclic



  --ntasks-per-node=1
  --cpus-per-task=19
  --distribution=*:block:fcyclic



Hint: number = task id

DIFFERENCES TO VANILLA SLURM (19.05)

- Auto binding is not supported in psslurm
- *--cpu-bind=boards* is not supported
- The option *--cpu-bind=rank* is implemented differently in psslurm, since it is redundant and makes no sense without auto-pin. Slurm completely ignores the *--cpus-per-task* option here, psslurm does not.
- psslurm does NOT YET differentiate *ldoms* from *sockets*. This keywords are currently used equivalent.
- psslurm does not consider the values given to the options *--ntasks-per-core and --ntasks-per-socket*. (As far as we could observe, Slurm does neither, even though it is described otherwise in the srun manpage.)
- The hints *compute_bound* and *memory_bound* are currently not supported.
- psslurm follows what is described at the srun manpage. (In many cases, Slurm does not.)

THREAD AFFINITY INTERFACE - INTEL

KMP_AFFINITY=[<*modifier*>,...]<*type*>[,<*permute*>][,<*offset*>]

- modifier:
  - **verbose**: Prints messages concerning the supported affinity
  - …
- type:
  - **compact:** OpenMP thread <*n*>+1 is assigned to a free thread context as close as possible to the thread context where the <*n*> OpenMP thread was placed.
  - **scatter:** distributes the threads as evenly as possible across the entire system
  - **…**

https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference-thread-affinity-interface-linux-and-windows

Alternative examples:

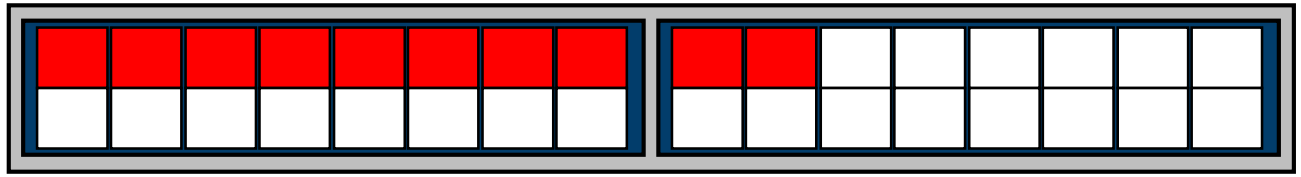
- GOMP_AFFINITY
- *sched_setaffinity* or *kmp_set_affinity* API

INTEL - THREAD AFFINITY INTERFACE

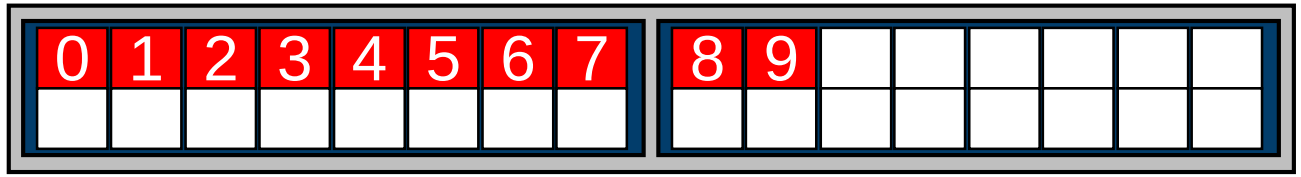
KMP_AFFINITY=verbose, [compact | scatter]

- 1st example:

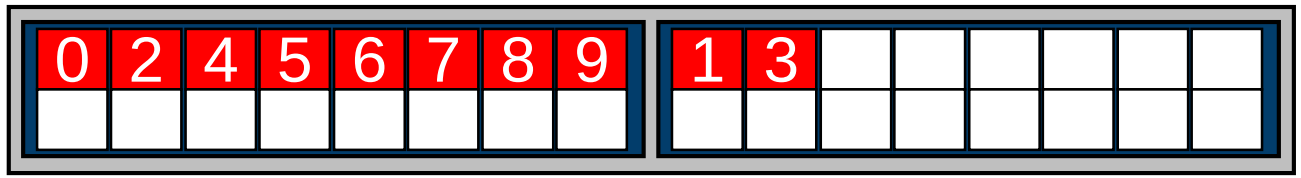
  --ntasks-per-node=1
  --cpus-per-task=10
  --cpu-bind=**rank**
  OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}



- KMP_AFFINITY=verbose,**compact**



- KMP_AFFINITY=verbose,**scatter**



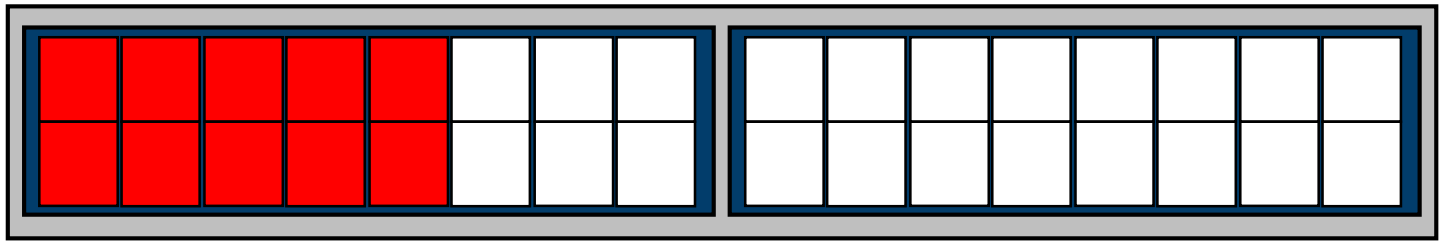
Hint: number = thread id

INTEL - THREAD AFFINITY INTERFACE

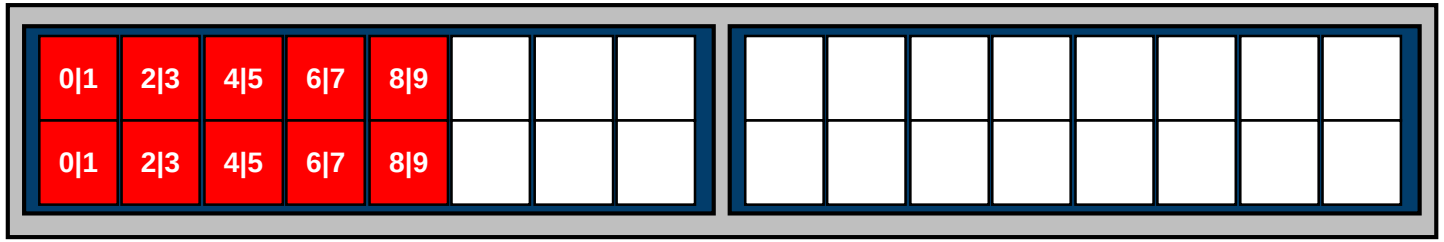
KMP_AFFINITY=verbose, [compact | scatter]

- 2nd example:

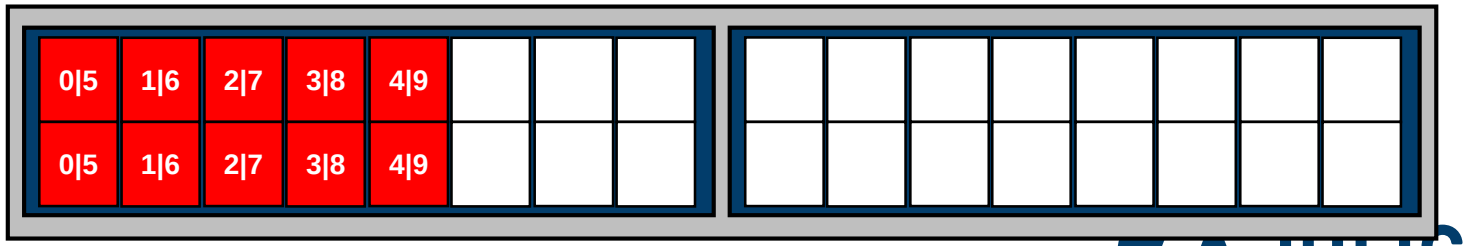
  --ntasks-per-node=1
  --cpus-per-task=10
  --cpu-bind=**threads**
  --distribution=*:*:**cyclic**
  OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}



- KMP_AFFINITY=verbose,**compact**



- KMP_AFFINITY=verbose,**scatter**



Hint: number = thread id

INTEL - THREAD AFFINITY INTERFACE

KMP_AFFINITY=verbose, [compact | scatter]

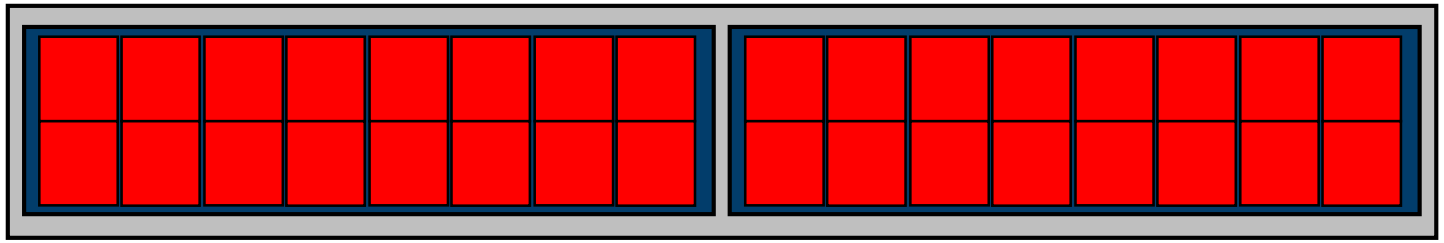
- 3rd example:

  --ntasks-per-node=1

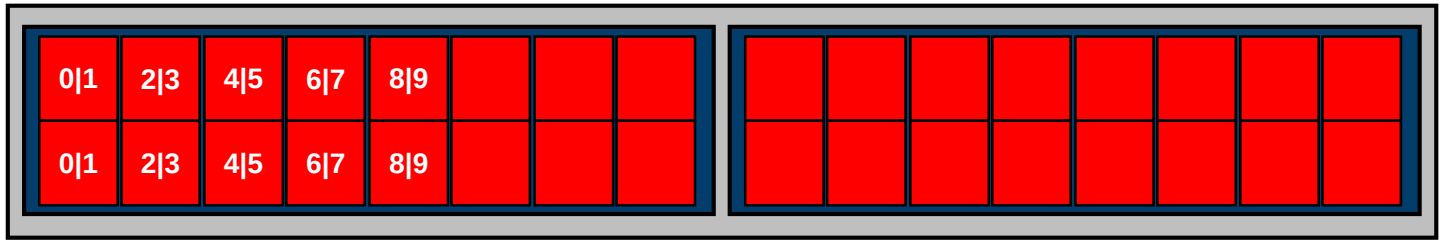
  --cpus-per-task=10

  --cpu-bind=**none**

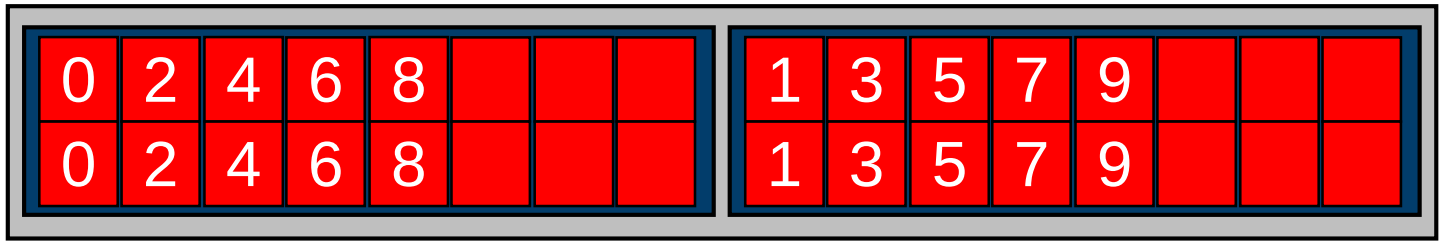
  OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}



- KMP_AFFINITY=verbose,**compact**



- KMP_AFFINITY=verbose,**scatter**

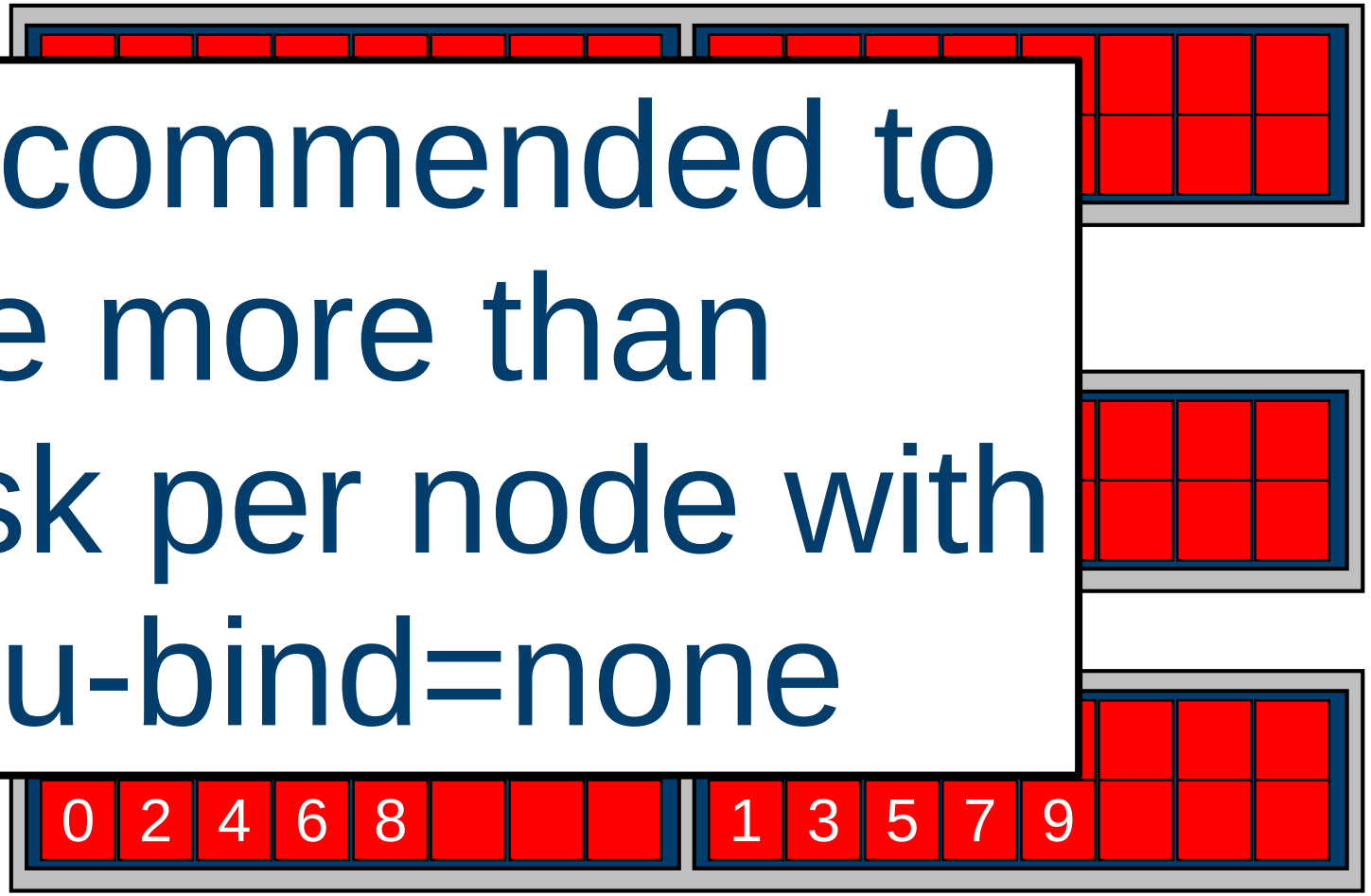


Hint: number = thread id

INTEL - THREAD AFFINITY INTERFACE

KMP_AFFINITY=verbose, [compact | scatter]

- 3rd example:
  - --ntasks-pe
  - --cpus-per-t
  - --cpu-bind=
  - OMP_NUM
- KMP_AFFINI
- KMP_AFFINI



Not recommended to use more than one task per node with --cpu-bind=none

Hint: number = thread id

NEW TOOL - PSSLURMGETPIN

Get pinning within a node

Usage:

psslurmgetpin <sockets> <coresPerSocket> <threadsPerCore> <options> : <srunOptions>

Options:

--help	Print this help
-v, --verbose	Be verbose (twice for debugging)
-h, --human-readable	Print 0/1-blocks instead of hex masks

Supported <*srunOptions*> (see srun manpage):

-N 1

-n <tasks>

-c <threadsPerTask>

--cpu_bind=<cpuBindType>

-m <distribution>, --distribution=<distribution>

NEW TOOL - PSSLURMGETPIN

1st example (JURECA):

$ psslurmgetpin 2 12 2 -h : -N 1 -n 6 -c 8 --cpu-bind=rank

```
0:
  111111110000 000000000000
  000000000000 000000000000
1:
  000000001111 111100000000
  000000000000 000000000000
2:
  000000000000 000011111111
  000000000000 000000000000
3:
  000000000000 000000000000
  111111110000 000000000000
4:
  000000000000 000000000000
  000000001111 111100000000
5:
  000000000000 000000000000
  000000000000 000011111111
```

NEW TOOL - PSSLURMGETPIN

2nd example (JURECA):

$ psslurmgetpin 2 12 2 -h : -N 1 -n 6 -c 8 --cpu-bind=mask_cpu:
0xF00000F,0xF00000F0,0xF00000F00,0xF00000F000,0xF00000F0000,0xF00000F00000

0:
111100000000 000000000000
111100000000 000000000000

1:
000011110000 000000000000
000011110000 000000000000

2:
000000001111 000000000000
000000001111 000000000000

3:
000000000000 111100000000
000000000000 111100000000

4:
000000000000 000011110000
000000000000 000011110000

5:
000000000000 000000001111
000000000000 000000001111

PORTABLE HARDWARE LOCALITY

hwloc

- What can it do?
  - Queries and visualisations of node topology
  - Generation and conversion of bitmasks from/for other tools
  - Distribution of tasks across hardware
  - Binding tasks to system locations
  - ….

For more information:

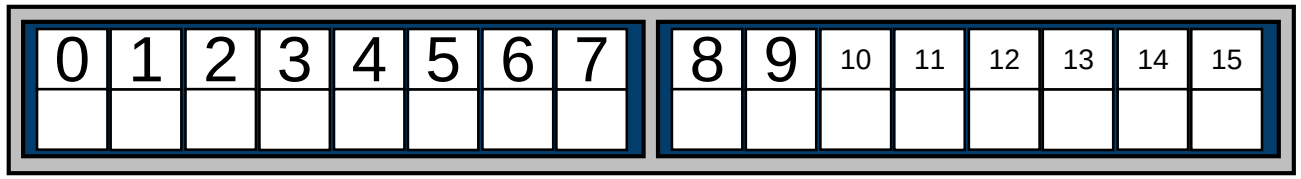
- https://www.open-mpi.org/projects/hwloc/

FURTHER EXAMPLES

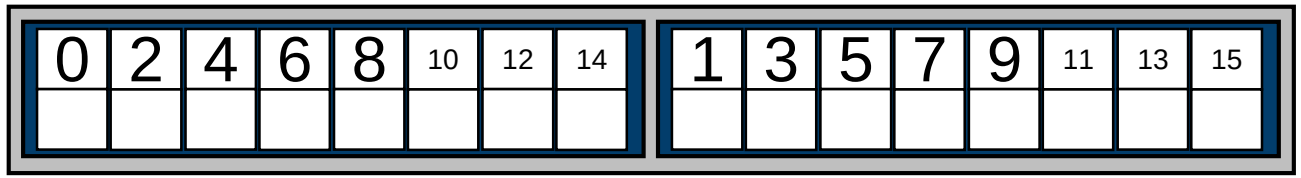
COMPARISON – RANK VS THREADS (PURE MPI)

- 1st example:

  --ntasks-per-node=16
  --cpus-per-task=1
  --cpu-bind=**rank**

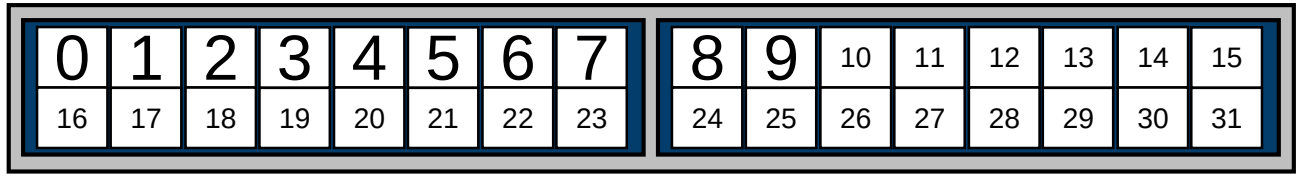


  --cpu-bind=**threads**
  [--distribution=*:*:*] or
  [--distribution=*:*:cyclic]

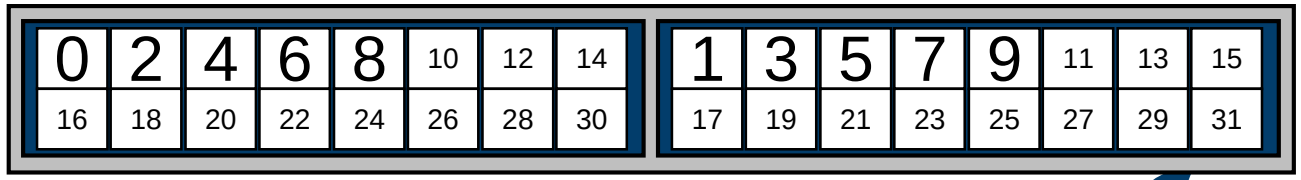


- 2nd example:

  --ntasks-per-node=32
  --cpus-per-task=1
  --cpu-bind=**rank**



  --cpu-bind=**threads**
  [--distribution=*:*:*] or
  [--distribution=*:*:cyclic]

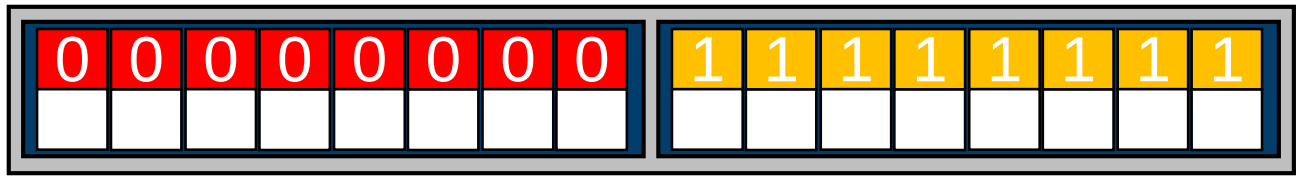


Hint: number = task id

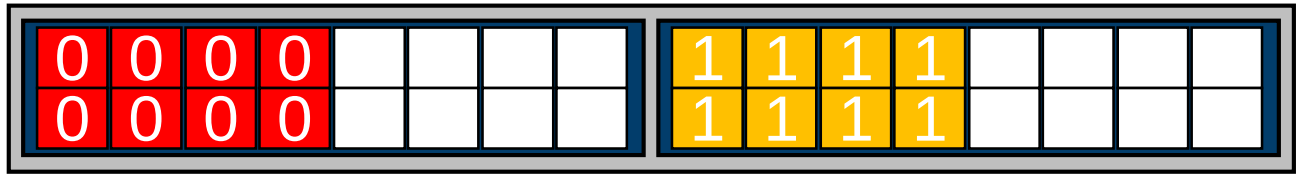
COMPARISON – RANK VS THREADS (MPI+OMP)

- 3rd example:

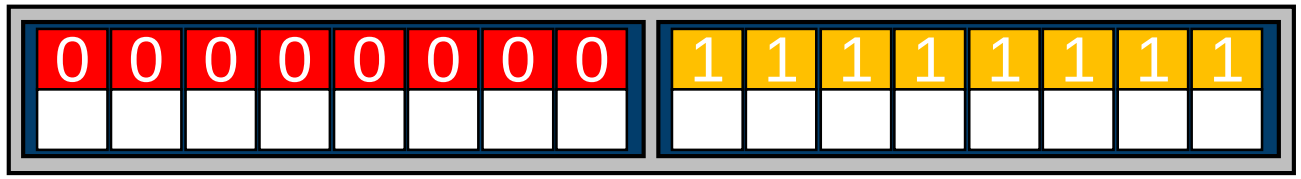
  --ntasks-per-node=2
  --cpus-per-task=8
  --cpu-bind=**rank**



  --cpu-bind=**threads**
  --distribution=*:*:cyclic



  --cpu-bind=**threads**
  [--distribution=*:*:*]

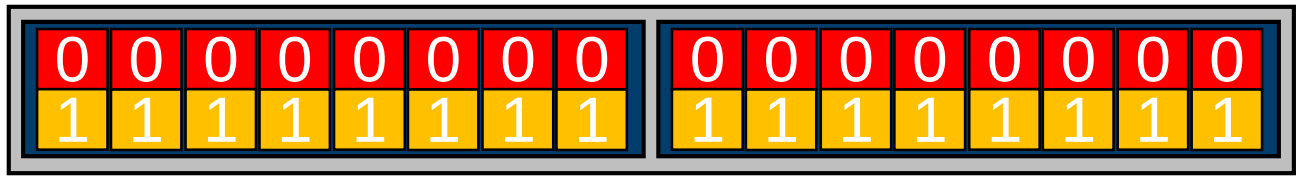


Hint: number = task id

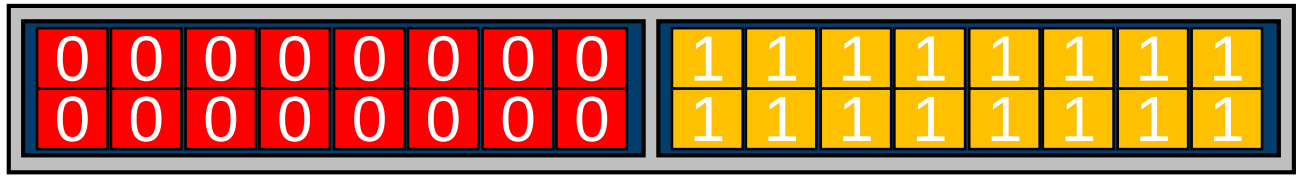
COMPARISON – RANK VS THREADS (MPI+OMP)

- 4th example:

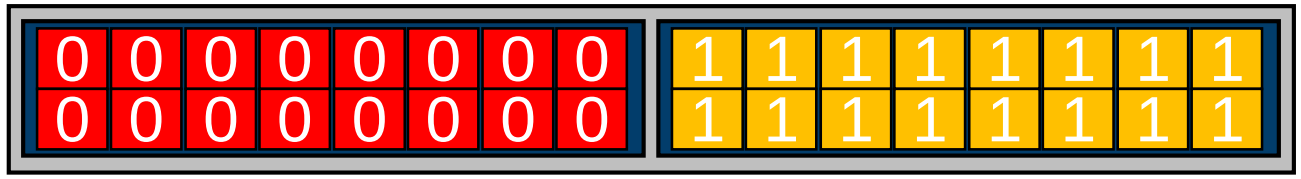
  --ntasks-per-node=2
  --cpus-per-task=16
  --cpu-bind=**rank**



  --cpu-bind=**threads**
  --distribution=*:*:cyclic



  --cpu-bind=**threads**
  [--distribution=*:*:*]

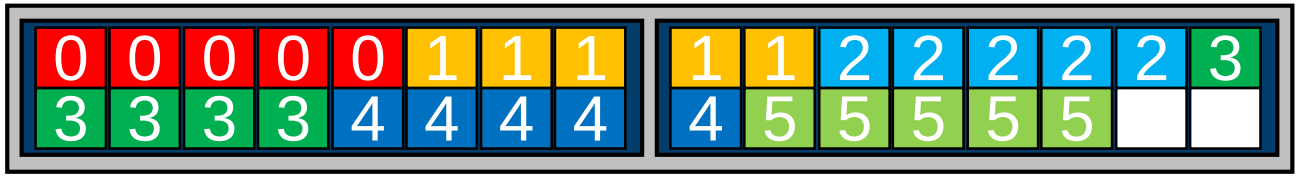


Hint: number = task id

COMPARISON – RANK VS THREADS (MPI+OMP)

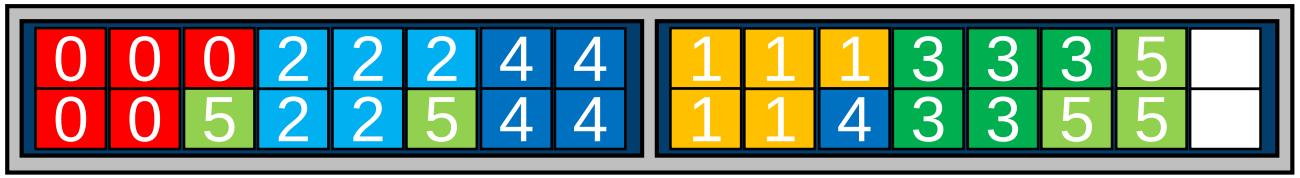
- 5th example:
  - **rank**: (current default)

    --ntasks-per-node=6
    --cpus-per-task=5
    [--cpu-bind=rank]

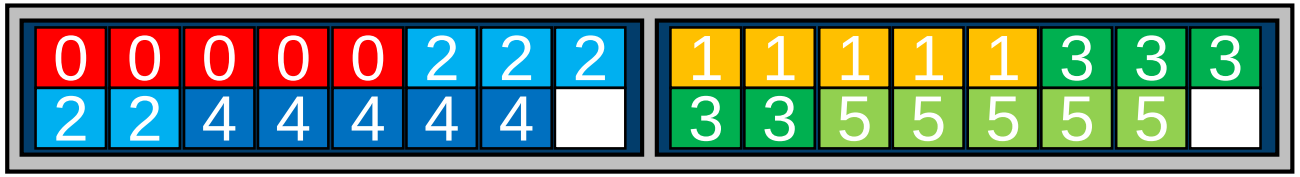


  - **threads**: (not supported in current version)

    --ntasks-per-node=6
    --cpus-per-task=5
    --cpu-bind=threads
    --distribution=*:*:cyclic



    --ntasks-per-node=6
    --cpus-per-task=5
    --cpu-bind=threads
    [--distribution=*:*:*]



Hint: number = task id

DECIDE FOR BINDING STRATEGY

- Selecting the „right“ binding strategy depends not only on the topology, but also on the characteristics of your application.
  - ➢ **Putting threads far apart**, i.e., on different sockets
    - \+ May improve the aggregated memory bandwidth available to your application
    - \+ May improve the combined cache size available to your application
    - – May decrease performance of synchronization constructs
  - ➢ **Putting threads close together**, i.e., on two adjacent cores that possibly share some caches
    - \+ May improve performance of synchronization constructs
    - – May decrease the available memory bandwidth and cache size
- If you are unsure, just try a few options and then select the best one.

Source: (Christian Terboven – OpenMPCon & IWOMP 2017)
https://openmpcon.org/wp-content/uploads/openmpcon2017/Tutorial2-Advanced_OpenMP.pdf