Mixing diverse workloads within a single job: a heterogeneous scheduling primer

Sergey Mashchenko
(SHARCNET)
syam@sharcnet.ca

April 8, 2026

Outline

- Introduction
- Heterogeneous scheduling in SLURM
- Use cases
  - MPI codes with unbalanced memory consumption
  - Codes with GPU and CPU components
  - Multi-model / multi-physics simulations
- Demo
- Conclusions

Introduction

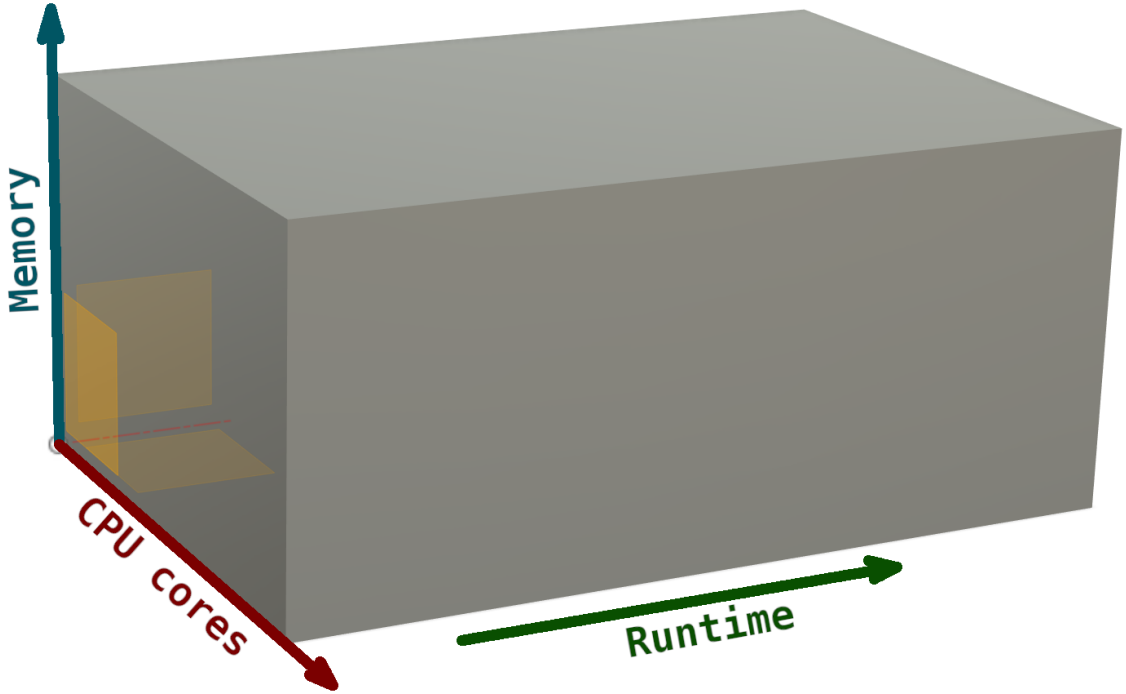
Traditional (homogeneous) scheduling

- National clusters are shared between thousands of researchers.
- Most of cluster resources are consumed via jobs submitted to our scheduler, SLURM.
- The bulk of the resources are the basic compute nodes (typically 192 cpu cores and 768GB of RAM per node), but most clusters also have some specialized nodes:
  - GPU nodes (often more than one flavour)
  - Large memory nodes

(cont.)

- Almost all jobs on our clusters use the homogeneous model of scheduling:
  - Jobs using more than one cpu core (or GPU) assume that the resources are identical for each process or thread, e.g.
    - MPI code where each MPI rank uses one cpu core and a fixed amount of memory
    - MPI / CUDA code where each rank uses one CPU core and one GPU (usually via the GPU sharing MPS mechanism), and again a fixed amount of memory.
    - MPI / OpenMP code where each rank uses a fixed number of CPU cores (one thread per core), and a fixed amount of shared memory.

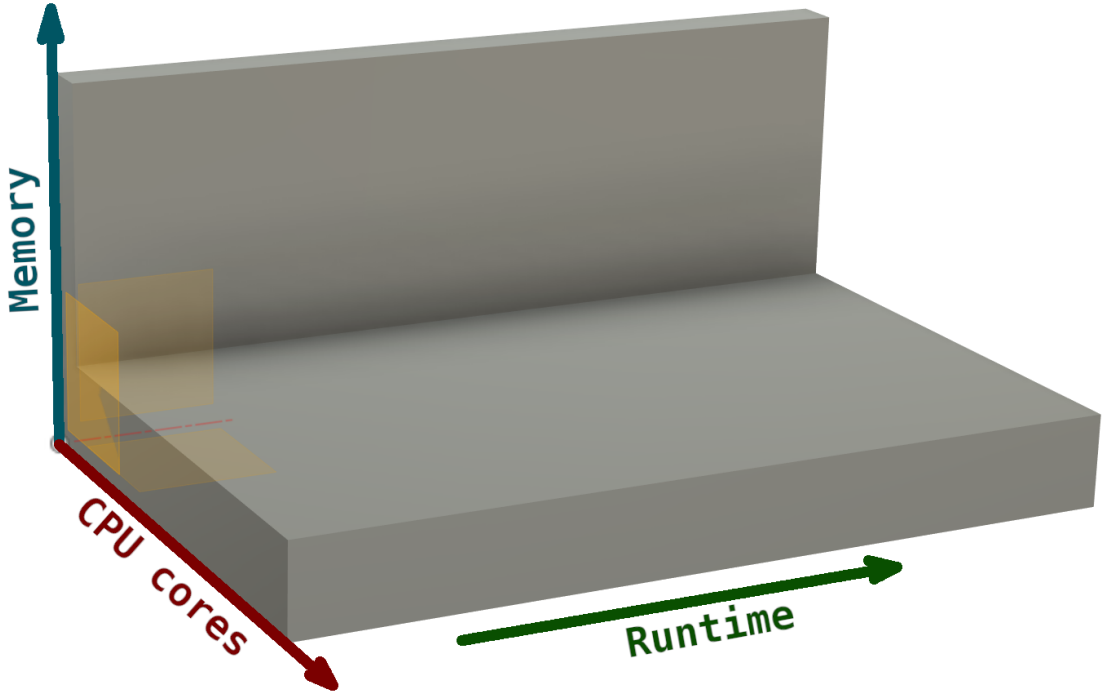
Good job shape



(cont.)

- In most cases, this works fine.
- But in some cases, the code has parts with distinctly different resources requirements, e.g.:
  - MPI code where rank 0 needs much more memory (and/or cpu cores) than the rest of the ranks, or
  - One code component primarily needs GPUs, while the other part needs lots of CPU cores.
- That can result in a significant waste of resources, and occasionally in jobs which cannot run on our clusters.

Bad job shape



Example

- In an MPI code, rank 0 is often “the administrator”, and as such deals with a very different workload than the rest of the ranks.
- E.g. it may need more memory (when reading from / writing to the file system).
- As an example, lets assume the rank 0 needs 16 GB of RAM, with the remaining 191 ranks requiring 4 GB of RAM per rank.

(cont.)

- How to schedule such a job?
- The most straightforward (and wasteful) way: request the same (maximum) amount of memory for each rank:

```
#SBATCH --mem-per-cpu=16G
#SBATCH --ntasks=192
```

- The above job will waste ~2.3 TB (66%) of RAM, and almost 600 CPU cores – not acceptable.

(cont.)

- If there is flexibility on the number of MPI ranks, we could try to fit the whole job into a single base node by reducing ntasks:

```
#SBATCH --mem=0
#SBATCH --nodes=1
#SBATCH --ntasks=189
```

- Alternatively, if the memory per rank is slightly less than 4 GB, we could try to fit all 192 ranks into a single base node:

```
#SBATCH --mem=0
#SBATCH --nodes=1
#SBATCH --ntasks=192
```

(cont.)

- If neither assumptions work, you can still use the “--mem=0” trick, but with 2 basic nodes:

```
#SBATCH --mem=0
#SBATCH --nodes=2
#SBATCH --ntasks=192
#SBATCH --ntasks-per-node=96
```

- This approach is still very wasteful, though not as bad as our first (naive) solution.
  - We are wasting ~50% of the CPU cores and memory

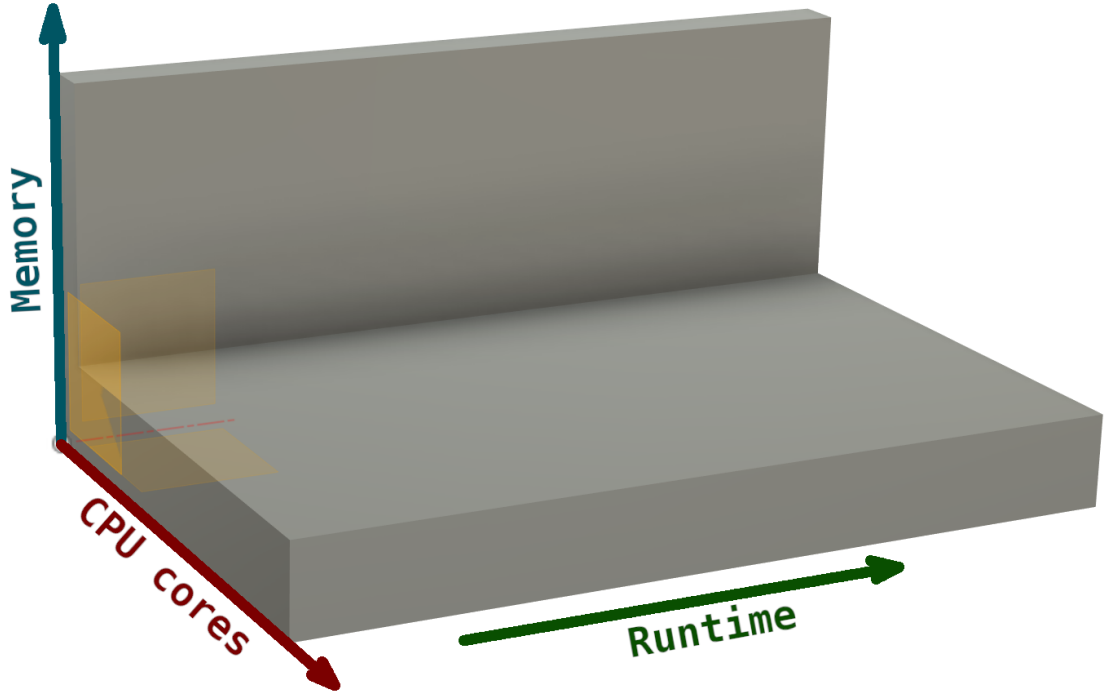
More examples

- No tricks will help in the following cases:
  - If rank 0 needs a GPU, while the rest don’t, such a job can end up wasting multiple GPUs (because it can only run on GPU nodes).
  - If rank 0 needs multiple threads (multiple CPU cores), and the rest just need one CPU core per rank, we need to request multiple CPU cores per rank for all the ranks (using the `--cpus-per-task` argument), which will result in wasting lots of CPU cores and memory.

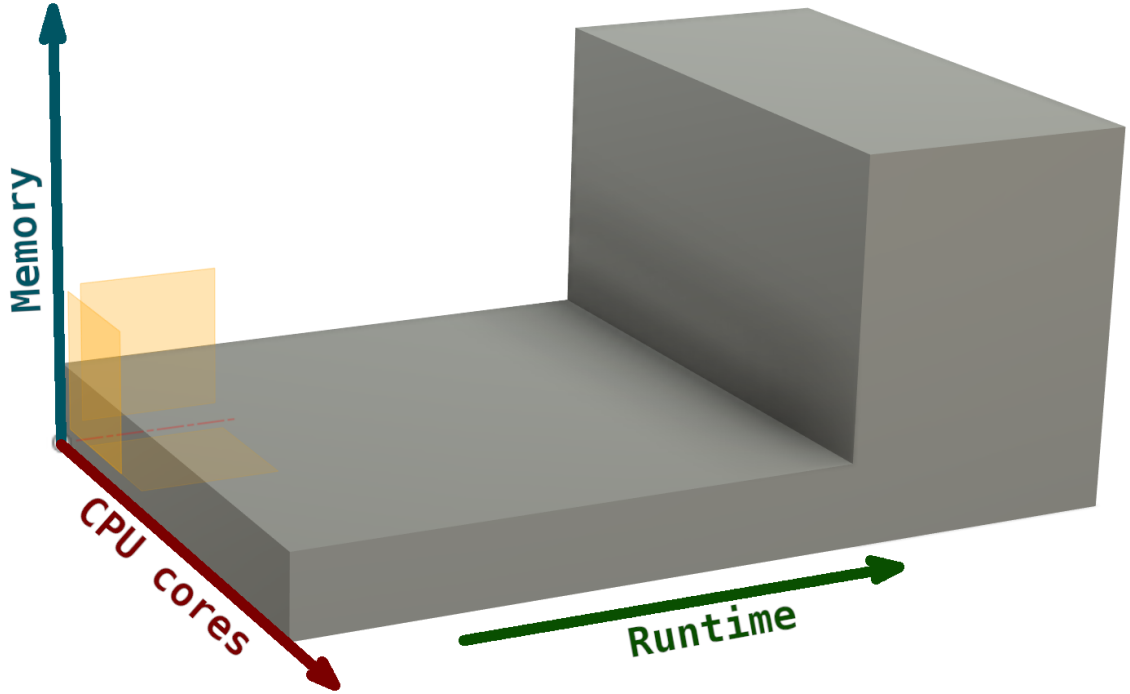
Limits to homogeneous scheduling

- More generally, the traditional (homogeneous) model of scheduling becomes very wasteful and hard or even impossible to schedule if the computational task consists of two or more distinct **concurrent** components, with each component having significantly different resource requirements.
- The solution: **heterogeneous scheduling**.
  - The main idea: a job can have 2 or more concurrent components, each with its own set of resource requirements.

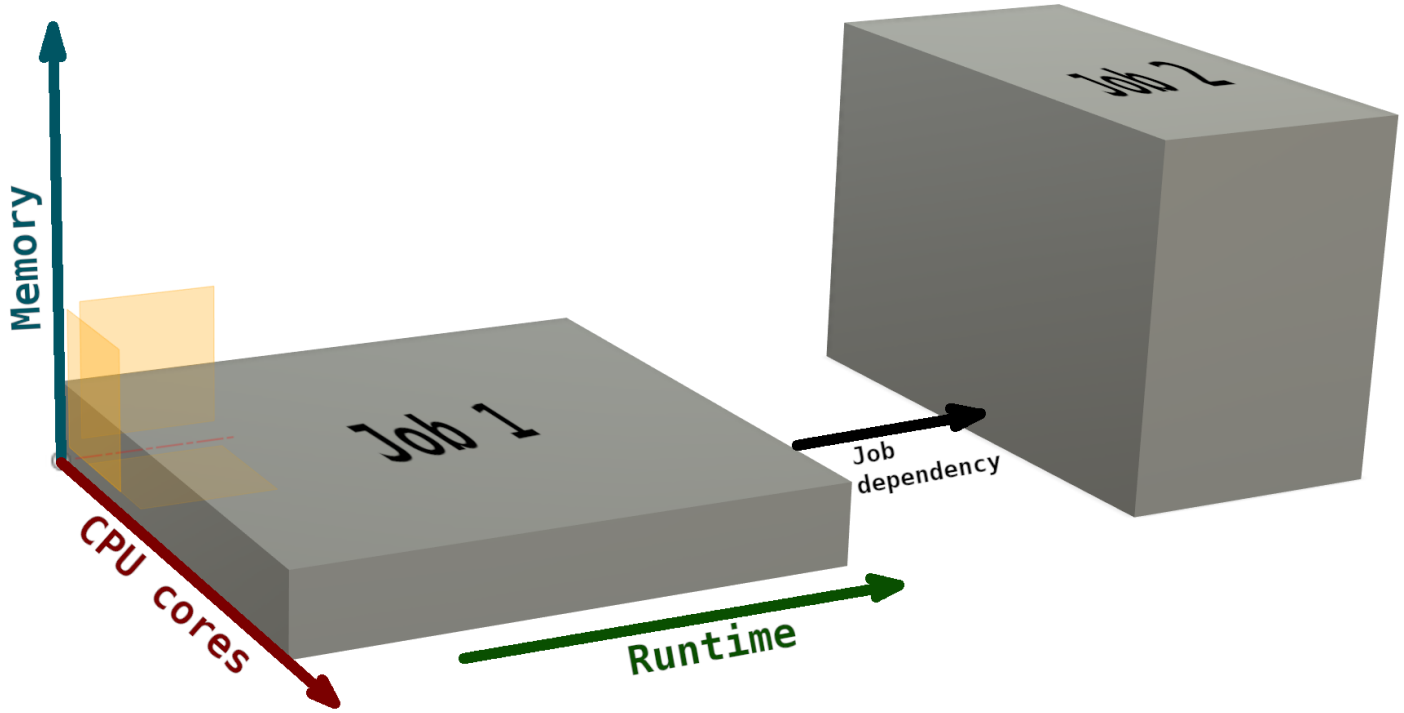
Problems heterogeneous scheduling can solve



Problems heterogeneous scheduling **cannot** solve



For the latter, we need job dependencies



Dependency or heterogeneous?

- When the job shape **changes with time** quantitatively (a dimension) or qualitatively (add / remove a special resource like GPUs or large memory nodes): use multiple jobs plus job dependencies.
- When the job has a complex shape (e.g. one part needs a GPU, the rest - don’t) which **doesn’t change with time**: use heterogeneous scheduling.
- In the most general case, for jobs with a complex shape which changes with time: use a combination of the above.

Heterogeneous scheduling in SLURM

SLURM: heterogeneous scheduling basics

- **SLURM** (Simple Linux Utility for Resource Management) is an open-source, highly scalable workload manager and job scheduler for Linux clusters and supercomputers.
- SLURM is used on all national systems.
- SLURM version 17.11 and later supports the ability to submit and manage heterogeneous jobs, in which each component has virtually all job options available.
  - Version 17.11 was released in 2017.
  - The heterogeneous jobs feature wasn’t widely advertised or tested on our systems.

Command line approach

- It is possible to specify the different components of the job on command line using `sbatch` or `salloc` – they just need to be separated by `:` (column):

```
$ cat my.bash
#!/bin/bash
srun : run.app

$ sbatch --cpus-per-task=4 --mem-per-cpu=16g --ntasks=1 : \
         --cpus-per-task=2 --mem-per-cpu=1g  --ntasks=8 my.bash
```

Separator

hetjob

- But it is more convenient to describe the components inside the jobscript using the special SBATCH keyword `hetjob`:

```
$ cat new.bash
#!/bin/bash
#SBATCH --cpus-per-task=4 --mem-per-cpu=16g --ntasks=1
#SBATCH hetjob
#SBATCH --cpus-per-task=2 --mem-per-cpu=1g  --ntasks=8

srun : run.app

$ sbatch new.bash
```

Don't put `hetjob` before the first component

Each `hetjob` starts a new component

sbatch arguments

- Many `sbatch` arguments are inherited by subsequent components (so one only needs to provide them for the first component). For example:

```
--account;
--dependency;
--job-name;
--mem;
--output;
--reservation;
--time.
```

- Some arguments should **only** be provided for the first component (e.g. `--output`).
- The rest should be explicitly provided for each component (otherwise they will get the default value), e.g.

```
--ntasks;
--nodes;
--cpus-per-task;
--mem-per-cpu.
```

After the job submission

- Commands like `squeue` and `sacct` by default will show the heterogeneous jobs in the following format:

```
Root_ID+Component_ID
```

- E.g.: 573452+0, 573452+1 etc.
- To cancel all the components: `scancel Root_ID` .
- To cancel one specific component: `scancel Root_ID+Component_ID` (only works when the jobs is already running).

srun: single MPI_COMM_WORLD

- Inside a heterogeneous job script, launch individual applications using `srun` (don't use `mpirun` – it will not work).
- Use the `:` character to separate the components of the job:

```
$ cat single.bash
#!/bin/bash
#SBATCH -N1 --mem=256G
#SBATCH hetjob
#SBATCH -N16 -n3072 --ntasks-per-core=1

srun server : client

$ sbatch single.bash
```

(cont.)

- Given the following two job components,

```
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=10G
#SBATCH hetjob
#SBATCH --ntasks=3
#SBATCH --mem-per-cpu=4G
```

the following commands are all equivalent:

```
srun ./mpi_hetero : ./mpi_hetero
srun  : ./mpi_hetero
srun --het-group=0,1 ./mpi_hetero
srun --het-group=0 : --het-group=1 ./mpi_hetero
```

srun: distinct MPI_COMM_WORLD

- Use `--het-group` to tell srun which job component(s) it should apply to. E.g. “`--het-group=0`”, “`--het-group=0,1`”, “`--het-group=0-2,4`”.
- Use more than one `srun` (with corresponding `--het-group` flags) to have distinct `MPI_COMM_WORLD` communicators (one per `srun`):

```
$ cat two.bash
#!/bin/bash
#SBATCH -N1 --mem=256G
#SBATCH hetjob
#SBATCH -N16 -n3072 --ntasks-per-core=1

srun --het-group=0 server &
srun --het-group=1 client &
wait

$ sbatch two.bash
```

A summary

- If your job components are a part of **a single MPI framework**: use a single `srun` command, with the components separator `:` and/or `--het-group` switch(es) as needed.
- **Otherwise**, use separate `srun` commands with corresponding `--het-group` switches for launching separate job components.
  - Make sure to add `&` at the end of each `srun` command, and use `wait` at the end of the job script.

Use cases

Case #1: memory-hungry rank 0

- In MPI programs, rank 0 is typically utilized for administrative tasks, often including:
  - Reading initial conditions or checkpointing files.
  - Performing domain decomposition of the input data, scattering the data across the rest of the ranks.
  - The opposite operation: gathering the parts of the data from the rest of the ranks, assembling it into a single data structure, then writing it to the file system (output or checkpointing files).
- This may result in an unbalanced memory utilization across the ranks, with the rank 0 requiring significantly more memory than the rest.

(cont.)

- Solutions:
  - If possible, use the "`--mem=0`" trick within a normal (homogeneous) job.
  - If that results in too much resources wasted, or is impossible to satisfy on our systems: use a two-component heterogeneous job with a single `srun`:

```
#!/bin/bash
#SBATCH -n1 --mem-per-cpu=256G
#SBATCH hetjob
#SBATCH -N2 -n384 --mem-per-cpu=4G

srun : mpi_code
```

Case #2: threaded rank 0

- Sometimes the rank 0 needs more compute resources than the rest of the ranks, to properly do it’s administrative duties.
  - E.g. this can happen if the MPI code utilizes a dynamic workload balancing scheme, which will increase the load on the rank 0 as the total number of ranks increases.
- One approach is to make rank 0 multi-threaded (e.g. using OpenMP), with each thread using a separate CPU core.

(cont.)

- Solution:
  - Use a two-component heterogeneous job with a single srun:

```
#!/bin/bash
#SBATCH -n1 --mem-per-cpu=16G --cpus-per-task=4
#SBATCH hetjob
#SBATCH -n384 --mem-per-cpu=4G --cpus-per-task=1

srun --cpus-per-task=4 : --cpus-per-task=1 ./mpi_code
```

Case #3: rank 0 doesn’t need a GPU

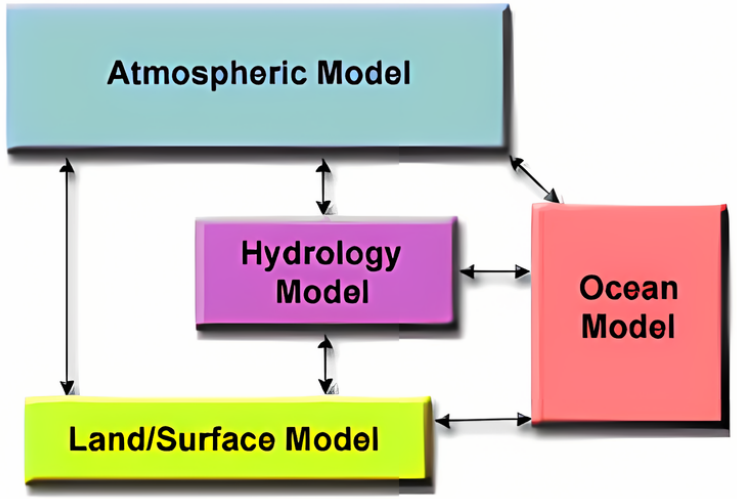
- In MPI/CUDA applications, the administrative rank 0 may not need a GPU, while the rest of the ranks may need a GPU each.
- Using the usual (homogeneous) job scheduling will result in wasting one GPU.
- Instead, use heterogeneous scheduling:

```
#!/bin/bash
#SBATCH -n1 --mem-per-cpu=4G
#SBATCH hetjob
#SBATCH --mem-per-cpu=4G --ntasks-per-gpu=1 --gres=gpu:a100:8

srun : mpi_code
```

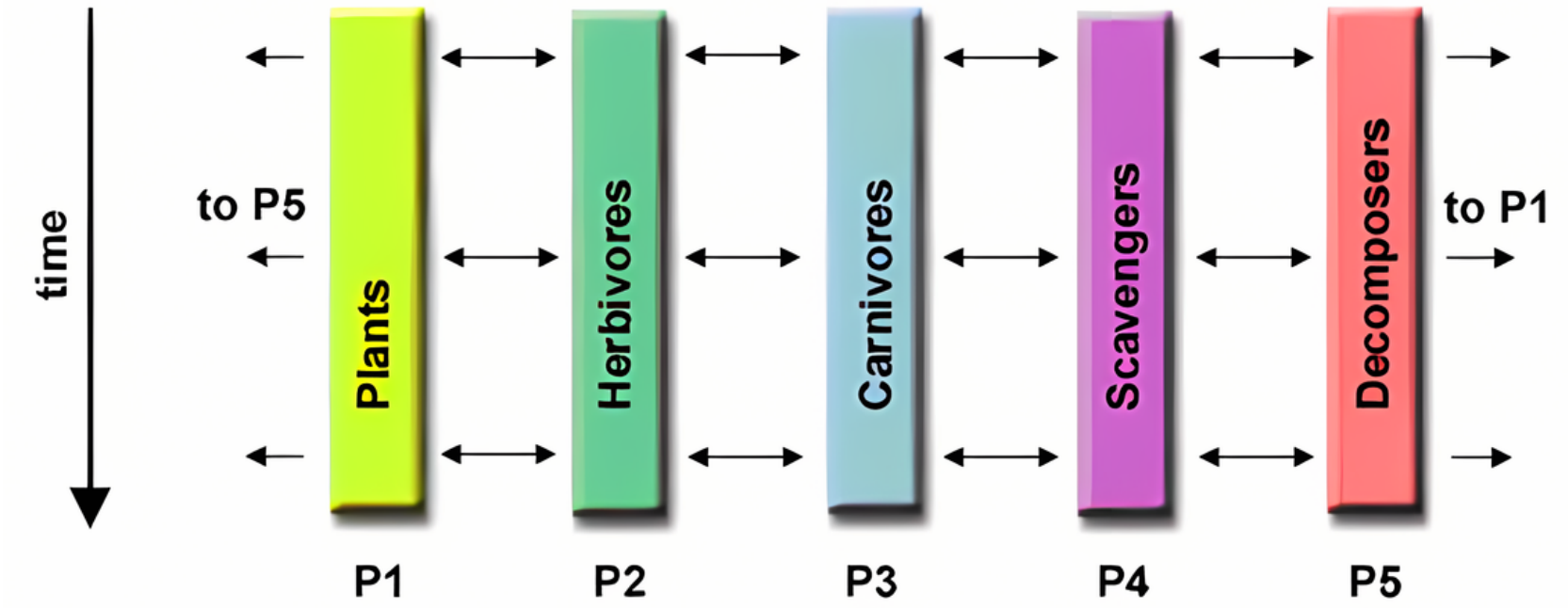
Case #4: a multi-physics simulation

- Sometimes a simulation consists of multiple models, often developed by different research groups, with very different compute requirements, running concurrently within a single job.
- Example: climate modeling



(cont.)

- Another example: ecosystem modeling

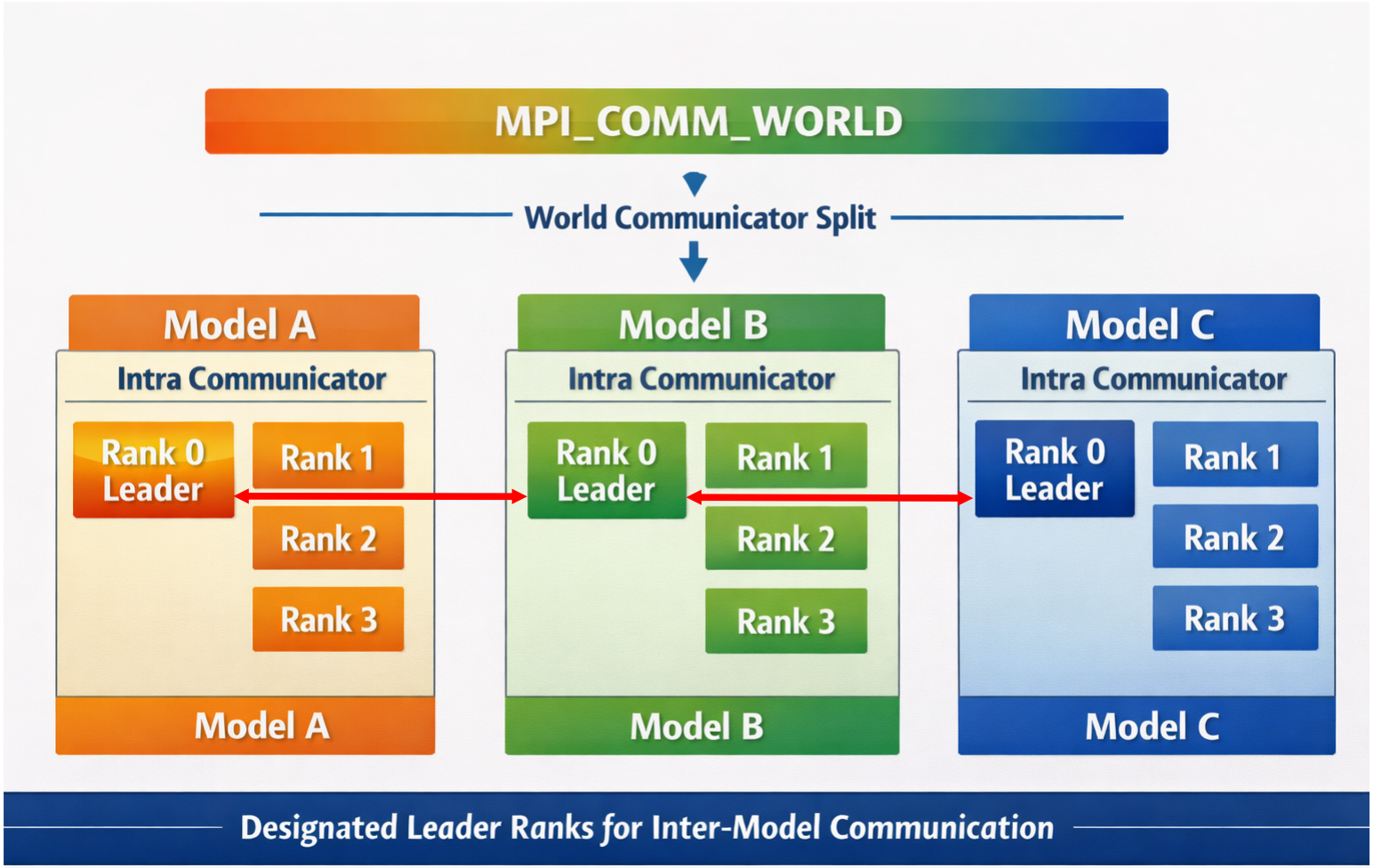


(cont.)

- Some models may need a large memory multi-threaded environment, others will need GPUs, or a basic MPI setup.
- Such multi-model packages typically use separate MPI communicators for each model, and for inter-model communications.
- Heterogeneous job may look like this:

```
#!/bin/bash
#SBATCH -n192 --mem-per-cpu=4G
#SBATCH hetjob
#SBATCH --mem-per-cpu=4G --ntasks-per-gpu=1 --gres=gpu:a100:8

srun --het-group=0 ./model_a : --het-group=1 ./model_b
```


MPI_COMM_WORLD
World Communicator Split
Model A
Intra Communicator
Rank 0 Leader
Rank 1
Rank 2
Rank 3
Model A
Model B
Intra Communicator
Rank 0 Leader
Rank 1
Rank 2
Rank 3
Model B
Model C
Intra Communicator
Rank 0 Leader
Rank 1
Rank 2
Rank 3
Model C
Designated Leader Ranks for Inter-Model Communication

Demo

Code examples

- Login to narval, then copy the examples:

```
$ ssh login_name@narval.alliancecan.ca
$ cp -r ~syam/Het-webinar .
```

- **Example #1:** `mpi_hetero`
  - Memory-hungry rank 0
- **Example #2:** `mpi_omp_hetero`
  - Multi-threaded rank 0

(cont.)

- **Example #3:** `cpu_gpu_mpi`
  - Rank 0 doesn’t use GPU, rank 1 uses GPU
- **Example #4:** `multi_model`
  - Inter-communicator is used for two MPI models to communicate with each other.

Conclusions

- Heterogeneous job scheduling is used when there are two or more concurrent job components with different resource requirements.
  - Using regular scheduling for such cases will often result in a significant waste of resources, and sometimes in jobs which cannot run on our clusters.
- Our scheduler SLURM provides a simple mechanism for enabling heterogeneous job scheduling.

Further reading

- https://slurm.schedmd.com/heterogeneous_jobs.html
- https://docs.alliancecan.ca/wiki/Advanced_Job_Submission

The end