Preparing, submitting and managing jobs with



Damien François — damien.francois@uclouvain.be

October 2026







Until now

Until now, you learned how to:

- access the cluster ✔
- copy data to/from the cluster ✔
- choose and activate software ✔
- run software in the command-line prompt ✔
- create and write text files ✔

Next: actually run software on the cluster ❓

tl;dr

- **Don't** run software on the login node
- **Do** submit a job to the resource manager/job scheduler

What is a job?

Dictionary

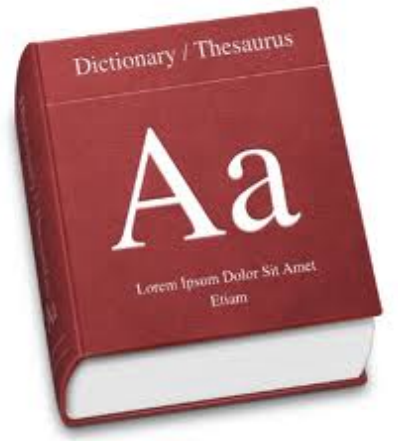
job [1] |jäb|

noun

1 a paid position of regular employment *: jobs are created in the private sector, not in Washington | a part-time job.*

2 a task or piece of work, esp. one that is paid *: she wants to be left alone to get on with the job | you **did a good job of** explaining.*

- a responsibility or duty *: it's our job to find things out.*
- [in sing.] informal a difficult task *: we thought you'd have a job getting there.*
- [with adj.] informal a procedure to improve the appearance of something, esp. an operation involving plastic surgery *: she's had a nose job | someone had done a skillful paint job.*
- [with adj.] informal a thing of a specified nature *: the car was a blue malevolent-looking job.*
- informal a crime, esp. a robbery *: a series of daring bank jobs.*
- Computing an operation or group of operations treated as a single and distinct unit.



What is a resource manager/scheduler ?

Job scheduler

From Wikipedia, the free encyclopedia

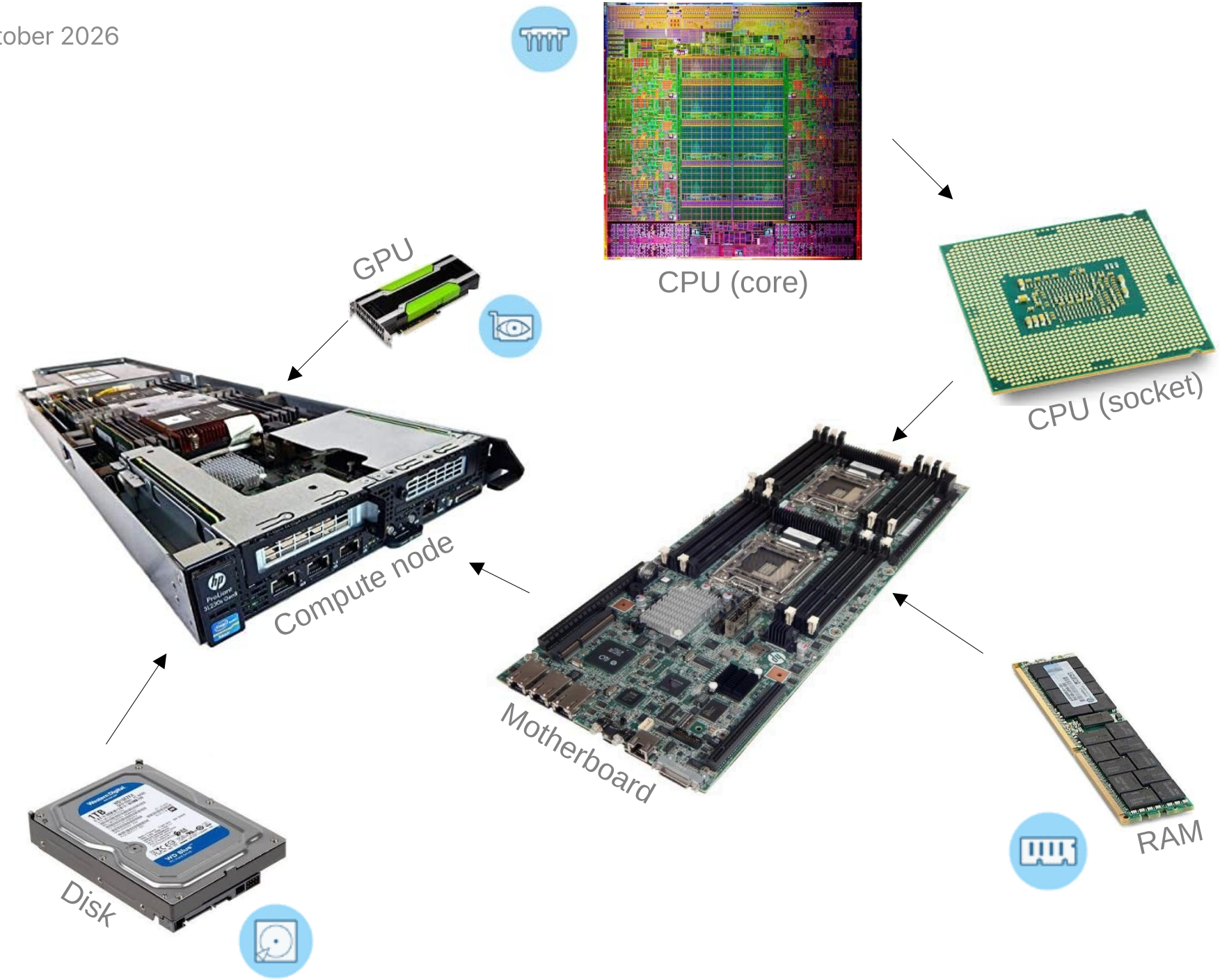
A **job scheduler** is a computer application for controlling unattended background program execution of jobs.[1] This is commonly called **batch scheduling**, as execution of non-interactive jobs is often called batch processing, though traditional *job* and *batch* are distinguished and contrasted; see that page for details. Other synonyms include **batch system**, **distributed resource management system** (**DRMS**), **distributed resource manager** (**DRM**), and, commonly today, **workload automation** (**WLA**). The data structure of jobs to run is known as the job queue.



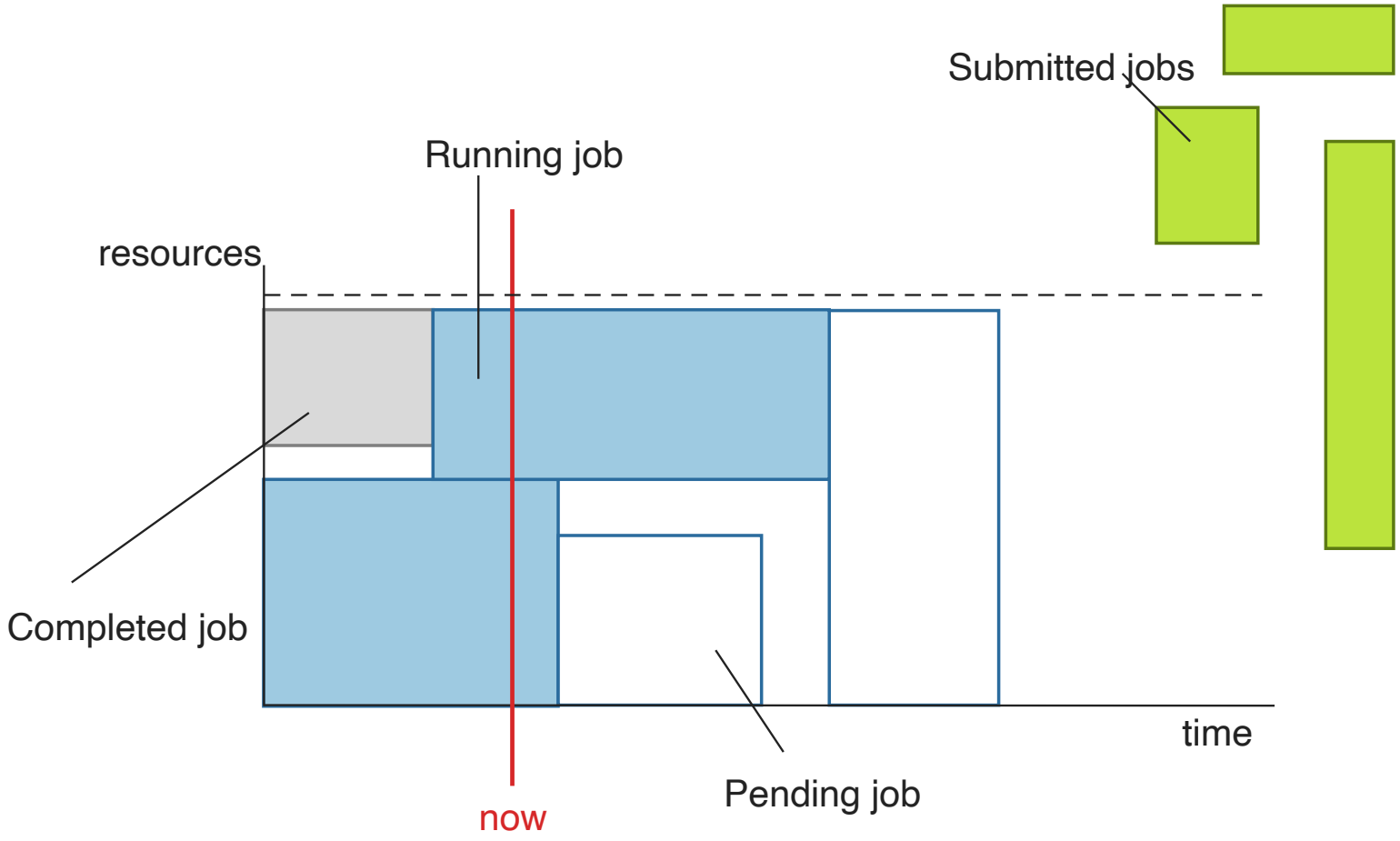
Resource management (computing)

From Wikipedia, the free encyclopedia

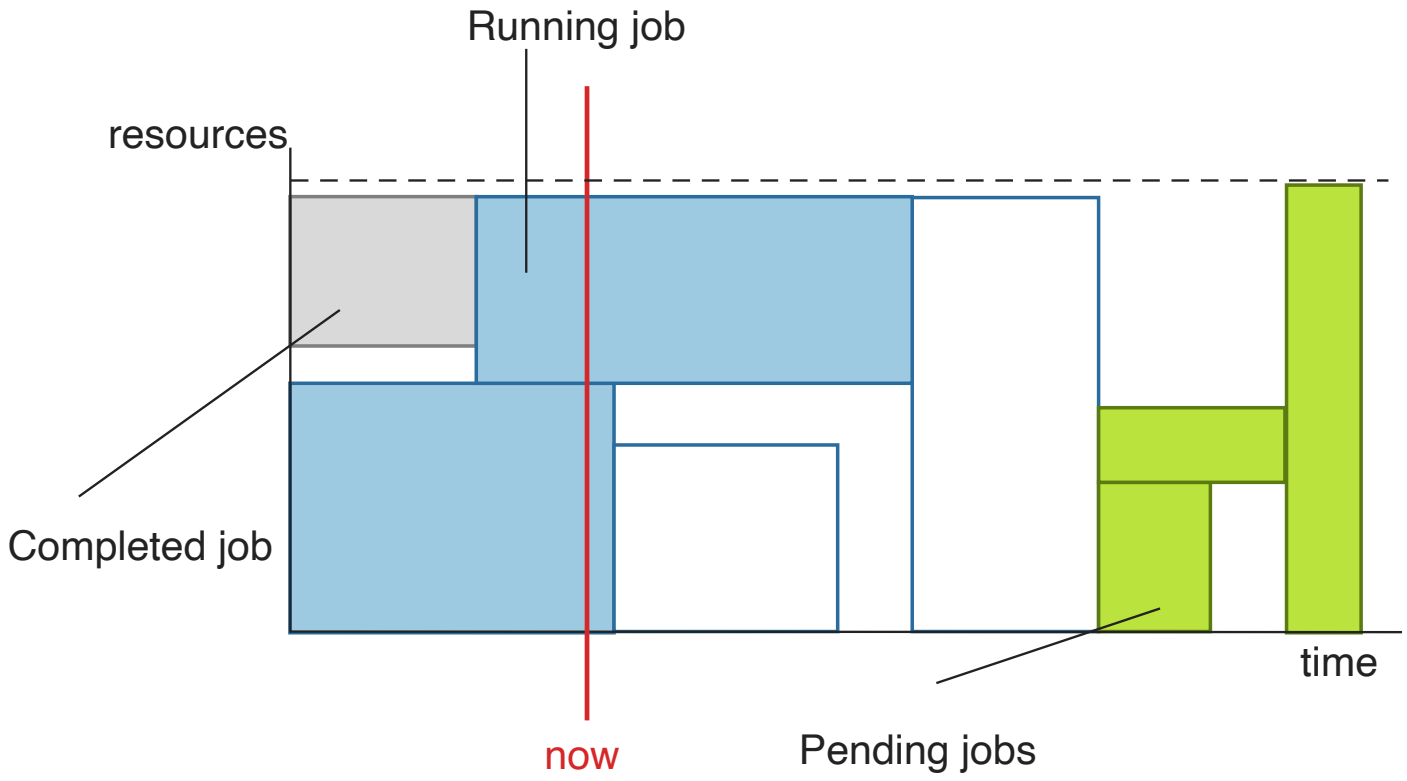
In computer programming, **resource management** refers to techniques for managing resources (components with limited availability).


CPU (core)
CPU (socket)
GPU
Compute node
Motherboard
RAM
Disk

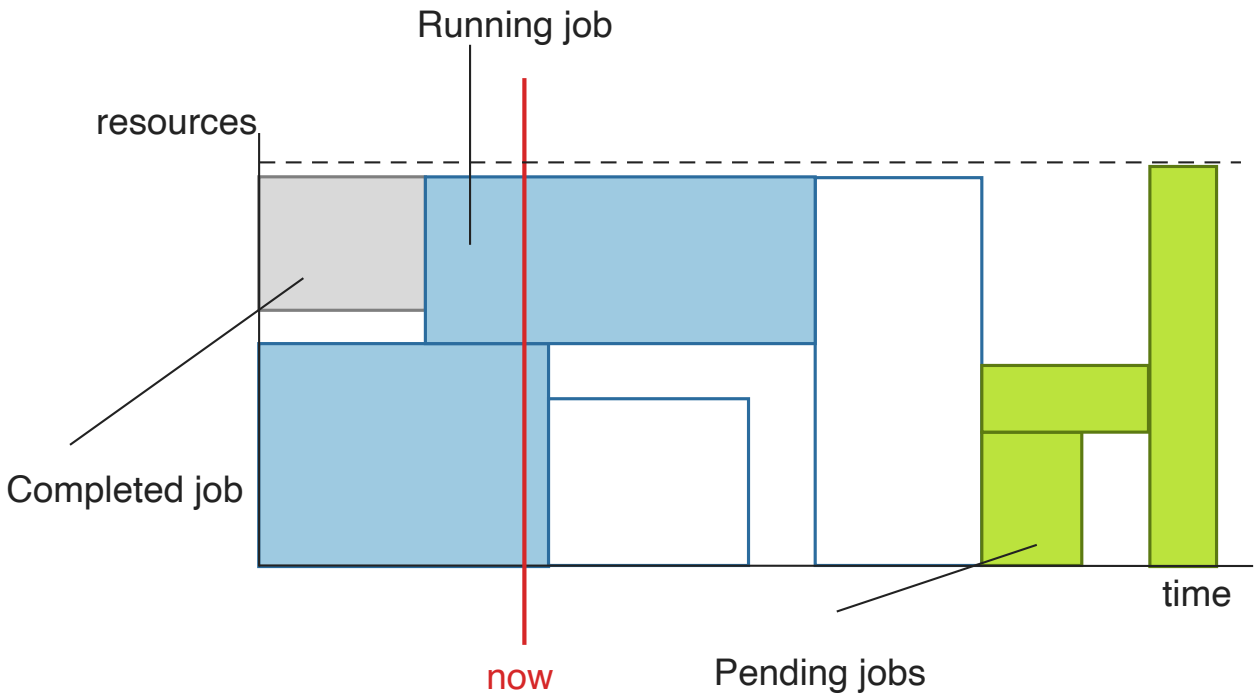
Scheduling


Submitted jobs
Running job
resources
Completed job
time
Pending job
now

Scheduling



Scheduling



Scheduling: exclusive allocation of resources to jobs over time

Scheduling is a game of Tetris



Slurm

- Free and free
- Mature (exists since ~2003)
- Very active community
- Many success stories
- Widely used
- Also an intergalactic soft drink



Topics:

	How to create a job
	How to choose resources
	Understand priorities
	Typical workloads
	Interactive sessions
	Debugging

Part 1 — Create, monitor and control jobs

You will learn how to:

- create and submit a job
- monitor and inspect jobs
- control (your own) jobs



Make up your mind …

	What	Example
Job parameters	resources you need for those operations	1 core, 2 GB RAM for 1 hour
Job steps	operations you need to perform	launch program `myprog`

… then write a submission script…

```
#!/bin/bash
# Submission script for demonstrating
# slurm usage.

# Job parameters
#SBATCH --job-name=demo
#SBATCH --output=res.txt
# Needed resources
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=2000
#SBATCH --time=1:00:00

# Operations
echo "Job start at $(date)"
# Job steps
srun ~/bin/myprog < mydata1

echo "Job end at $(date)"
```

- `#!/bin/bash` : it is a shell script (Bash)
- `# ...` : regular Bash comment
- `#SBATCH` : Bash sees these as comments, Slurm takes them as parameters
- `srun` : job step creation
- the rest: regular Bash commands

… then write a submission script…

```
#!/bin/bash
# Submission script for demonstrating
# slurm usage.

# Job parameters
#SBATCH --job-name=demo
#SBATCH --output=res.txt
# Needed resources
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=2000
#SBATCH --time=1:00:00

# Operations
echo "Job start at $(date)"
# Job steps
srun ~/bin/myprog < mydata1

echo "Job end at $(date)"
```

- `#!/bin/bash` : it is a shell script (Bash)
- `# ...` : regular Bash comment
- `#SBATCH` : Bash sees these as comments, Slurm takes them as parameters
- `srun` : job step creation
- the rest: regular Bash commands

No Bash variables allowed in parameters:

`#SBATCH --output=$HOME/res.txt` will not work.

… then write a submission script…

```
#!/bin/bash
# Submission script for demonstrating
# slurm usage.

# Job parameters
#SBATCH --job-name=demo
#SBATCH --output=res.txt
# Needed resources
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=2000
#SBATCH --time=1:00:00

# Operations
echo "Job start at $(date)"
# Job steps
srun ~/bin/myprog < mydata1

echo "Job end at $(date)"
```

- `#!/bin/bash` : it is a shell script (Bash)
- `# ...` : regular Bash comment
- `#SBATCH` : Bash sees these as comments, Slurm takes them as parameters
- `srun` : job step creation
- the rest: regular Bash commands

No command allowed between parameters:

```
#SBATCH --job-name=demo
echo Hello
#SBATCH --output=res.txt
```

will not work

… then write a submission script…

```
#!/bin/bash
# Submission script for demonstrating
# slurm usage.

# Job parameters
#SBATCH --job-name=demo
#SBATCH --output=res.txt
# Needed resources
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=2000
#SBATCH --time=1:00:00

# Operations
echo "Job start at $(date)"
# Job steps
srun ~/bin/myprog < mydata1

echo "Job end at $(date)"
```

- `#!/bin/bash` : it is a shell script (Bash)
- `# ...` : regular Bash comment
- `#SBATCH` : Bash sees these as comments, Slurm takes them as parameters
- `srun` : job step creation
- the rest: regular Bash commands

`srun` commands run on all nodes of the allocation and are monitored specifically; **non-`srun` commands** run on the first node of the allocation and are not monitored

… and submit it with `sbatch`

```
$ sbatch --partition=batch submit.sh
Submitted batch job 97920
```

`--partition` on the command line is one more job parameter.

Slurm gives me the *job ID*, used later on to uniquely identify the job.

Job parameters can be specified by (from lowest to highest precedence):

1. `#SBATCH --...` directives in the submission script
2. environment variables `$SBATCH_...`
3. parameters on the `sbatch --...` command line

Most of the parameters have default values and can be omitted but some are important.

Monitor jobs with the `squeue` command

```
SQUEUE(1)                 Slurm components

NAME
       squeue - view information about jobs located in the SLURM
       scheduling queue.

SYNOPSIS
       squeue [OPTIONS...]

DESCRIPTION
       squeue is used to view job and job step information for jobs
       managed by SLURM.
```

Monitor jobs with the `squeue` command

```
$ squeue
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
12324     batch     demo      dfr  R   11:10:02      4 node[001-004]
12325     batch     demo      dfr PD      00:00      2 (Resources)
12329     batch   prod_1      bvr PD      00:00      1 (Priority)
12422     debug   test_2      bvr  R      04:01      1 node005
```

- `JOBID` job ID assigned by Slurm
- `PARTITION` set of nodes targeted
- `NAME` set with `--job-name`
- `USER` owner of the job
- `ST` state: R (running), PD (pending), …
- `TIME` time the job has been running
- `NODES` number of nodes
- `NODELIST` nodes assigned: `node[001-004]`
- `(REASON)` why pending: Resources, Priority, …

Monitor jobs with the `squeue` command

```
$ squeue --me
       JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
       12324     batch     demo      dfr  R   11:10:02      4 node[001-004]
       12325     batch     demo      dfr PD      00:00      2 (Resources)

$ squeue --me --start
       JOBID PARTITION     NAME     USER ST       START_TIME NODES SCHEDNODES    NODELIST(REASON)
       12325     batch     demo      dfr PD 2025-02-12T09:12     2 node[001-002] (resources)

$ squeue --partition=debug
       JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
       12422     debug   test_2      bvr  R      04:01      1 node005

$ squeue --Format=jobid,partition,timeused,timelimit --partition=debug
       JOBID               PARTITION           TIME                TIME_LIMIT
       12422               debug               04:01               20:00
```

Get all information about a job with `scontrol show`

```
$ scontrol show job 12324
JobId=12324 JobName=demo
   UserId=dfr(3000003) GroupId=dfr(3000003) MCS_label=N/A Priority=6936634 Nice=0
   Account=ceci QOS=normal JobState=RUNNING Reason=None Dependency=(null) Requeue=0
   Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0 RunTime=00:00:00 TimeLimit=14:00:00
   TimeMin=N/A SubmitTime=2021-10-06T16:07:57 EligibleTime=2021-10-06T16:07:57
   AccrueTime=2021-10-06T16:07:57 StartTime=2021-10-07T17:42:35 EndTime=2021-10-07T21:42:35
   Deadline=N/A SuspendTime=None SecsPreSuspend=0 LastSchedEval=2021-10-06T16:08:38
   Partition=batch AllocNode:Sid=lm3-w078:184117 ReqNodeList=(null) ExcNodeList=(null)
   NodeList=(null) FedOrigin=cluster1 FedViableSiblings=lemaitre3 FedActiveSiblings=cluster1
   NumNodes=4 NumCPUs=4 NumTasks=1 CPUs/Task=4 ReqB:S:C:T=0:0:*:*
   TRES=cpu=4,mem=2400M,node=1,billing=4 Socks/Node=* NtasksPerN:B:S:C=1:0:*:* CoreSpec=*
   MinCPUsNode=4 MinMemoryCPU=600M MinTmpDiskNode=0 Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/users/d/f/dfr/test.sh WorkDir=/home/users/d/f/dfr/test.sh
   StdErr=home/users/d/f/dfr/res.txt StdIn=/dev/null StdOut=home/users/d/f/dfr/res.txt
   Power= MailUser=damien.francois@uclouvain.be MailType=FAIL
```

Cancel jobs with … `scancel`

```
Usage: scancel [-A account] [--batch] [--full] [--interactive] [-n job_name]
               [-p partition] [-Q] [-q qos] [-R reservation][-s signal | integer]
               [-t PENDING | RUNNING | SUSPENDED] [--usage] [-u user_name]
               [--hurry] [-V] [-v] [-w hosts...] [--wckey=wckey]
               [job_id[_array_id][.step_id]]
```

```
$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             12324     batch     demo      dfr  R   11:10:02      4 node[001-004]
             12325     batch     demo      dfr PD      00:00      2 (Resources)

$ scancel 12324

$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             12325     batch     demo      dfr PD      00:00      2 (Resources)
```

Modify jobs with `scontrol update`

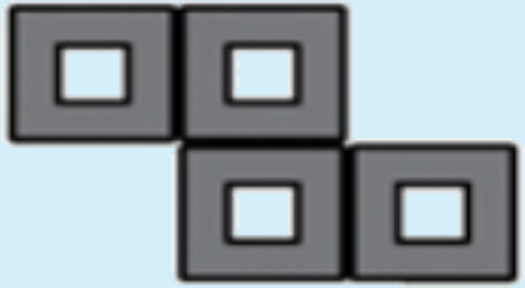
```
$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             12324     batch     demo      dfr  R   11:10:02      4 node[001-004]
             12325     batch     demo      dfr PD      00:00      2 (Resources)

$ scontrol update jobid=12325 numnodes=3

$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             12324     batch     demo      dfr  R   11:10:02      4 node[001-004]
             12325     batch     demo      dfr PD      00:00      3 (Resources)
```

Most parameters can only be changed for PENDING jobs.

Part 2 — Discover, target and size resources



You will learn how to:

- discover cluster features (resources)
- target specific features and tune your jobs
- choose suitable resource values
- get job actual resource usage

Use `sinfo` to find out about the nodes and the partitions

```
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*       up 2-00:00:00      2   idle node[001-002]
batch*       up 2-00:00:00      1  alloc node003
batch*       up 2-00:00:00      1    mix node004
debug        up      06:00      1   idle node005
```

- `PARTITION` partition name (`*` = default)
- `AVAIL` partition state: up, down, …
- `TIMELIMIT` maximum run time of a job
- `NODES` number of nodes in that state
- `STATE` node state: idle, alloc, mix, …
- `NODELIST` nodes in that state

Use `sinfo` to find out about the nodes and the partitions

```
$ sinfo --format "%5D %9P  %25f  %.5c  %.8m %G"
NODES PARTITION  AVAIL_FEATURES               CPUS    MEMORY GRES
4     batch*     amd,rome,7542,zenver2         64    257790 gpu:TeslaA100:2
1     debug      amd,rome,7542,zenver2         64   1031900 (null)
```

- `NODES` number of identical nodes
- `PARTITION` partition of the nodes
- `AVAIL_FEATURES` features set by the admins
- `CPUS` compute units (e.g. cores) per node
- `MEMORY` RAM per node, in MB
- `GRES` generic resources, e.g. GPUs

Use `sacctmgr` and `scontrol` to find out about QOSes and licenses

```
$ sacctmgr list qos
      Name   Priority  GraceTime    Preempt   PreemptExemptTime PreemptMode
---------- ---------- ---------- ---------- ------------------- -----------
    normal          0   00:00:00                                     cluster
  priority      10000   00:00:00                                     cluster

$ scontrol show licenses
LicenseName=abaqus@ucl
    Total=48 Used=0 Free=48 Remote=yes
```

- **QOS** (Quality of Service): used by sysadmins to organise and prioritise jobs
- **License**: used to distribute software licenses to jobs, often also other cluster-wide resources

Target resources with `#SBATCH` parameters

You want	You ask
To choose a specific feature (e.g. a processor type or a network type)	`--constraint`
To use a generic resource (e.g. a GPU)	`--gres` (or `--gpus`)
To access a specific licensed software	`--licenses`
To choose a partition	`--partition`
To use a specific QOS	`--qos`
To choose the CPU distribution on nodes	`--nodes`, `--ntasks-per-node`, `--cpus-per-task`

Tune your jobs with `#SBATCH` parameters

You want	You ask
To set a job name	`--job-name`
To attach a comment to the job	`--comment="Some comment"`
To get emails	`--mail-type=BEGIN|END|FAIL|ALL|TIME_LIMIT_90` `--mail-user=my@mail.com`
To set the name of the output file	`--output=result-%j.txt --error=error-%j.txt`
To enquire when it would start	`--test-only`
To specify an ordering	`--dependency=after(ok|notok|any):jobids` `--dependency=singleton`

A word about limits

- **Natural limits**: the hardware specifications
- **Admin-defined limits**: to ensure fair access for everyone, e.g. maximum job time

Limits can be set on:

- the number of running or submitted jobs
- the size of a job
- the duration of a job
- the CPU usage of all jobs of a user
- the cluster usage of an account
- …



View limits with `sacctmgr`

Limits set	Command
globally for all users	`sacctmgr show cluster`
globally for a specific user	`sacctmgr list user $USER withassoc`
at the QOS level	`sacctmgr list qos`
at the account (project) level	`sacctmgr list account MyAccount withassoc where user=$USER`
on partitions	`scontrol show partitions`

View limits with `sacctmgr`

NIC6:

```
$ sacctmgr list qos format=Name,GrpTRES,GrpJobs,GrpSubmit,MaxTRES,MaxTRESPerUser,MaxJobsPU
      Name       GrpTRES GrpJobs GrpSubmit       MaxTRES     MaxTRESPU MaxJobsPU
---------- ------------- ------- --------- ------------- ------------- ---------
    normal                                       cpu=384       200
```

Lemaitre4:

```
$ sacctmgr list qos format=Name,GrpTRES,GrpJobs,GrpSubmit,MaxTRES,MaxTRESPerUser,MaxJobsPU
      Name       GrpTRES GrpJobs GrpSubmit       MaxTRES     MaxTRESPU MaxJobsPU
---------- ------------- ------- --------- ------------- ------------- ---------
    normal                                      cpu=4096
```

https://support.ceci-hpc.be/doc/SubmittingJobs/SlurmLimits/

Know why your job is pending with `squeue -l`

```
$ squeue --me -l
Wed Aug 24 11:00:30 2022
CLUSTER: lemaitre3
             JOBID PARTITION     USER    STATE       TIME TIME_LIMI  NODES NODELIST(REASON)
          70786661     batch      dfr  PENDING       0:00      6:00     50 (Resources)
          70786672     batch      dfr  PENDING       0:00      6:00     50 (Priority)
          70786664     batch      dfr  PENDING       0:00      6:00      1 (BeginTime)
          70786673     batch      dfr  PENDING       0:00      6:00      1 (ReqNodeNotAvail)
          70786670     batch      dfr  PENDING       0:00      6:00      1 (Dependency)
          70786657     batch      dfr  PENDING       0:00      6:00      1 (JobHeldUser)
          70786658     debug      dfr  PENDING       0:00      6:00      5 (PartitionNodeLimit)
```

Look at the `NODELIST(REASON)` column.

Reason codes: https://slurm.schedmd.com/squeue.html#SECTION_JOB-REASON-CODES

HANDS-ON

Play Gameshell, Slurm edition

1. SSH to Lemaitre4
2. Run `module load gameshell/slurm`



A word about resource requests

There is no magic solution to finding the optimal resource request for a given job.

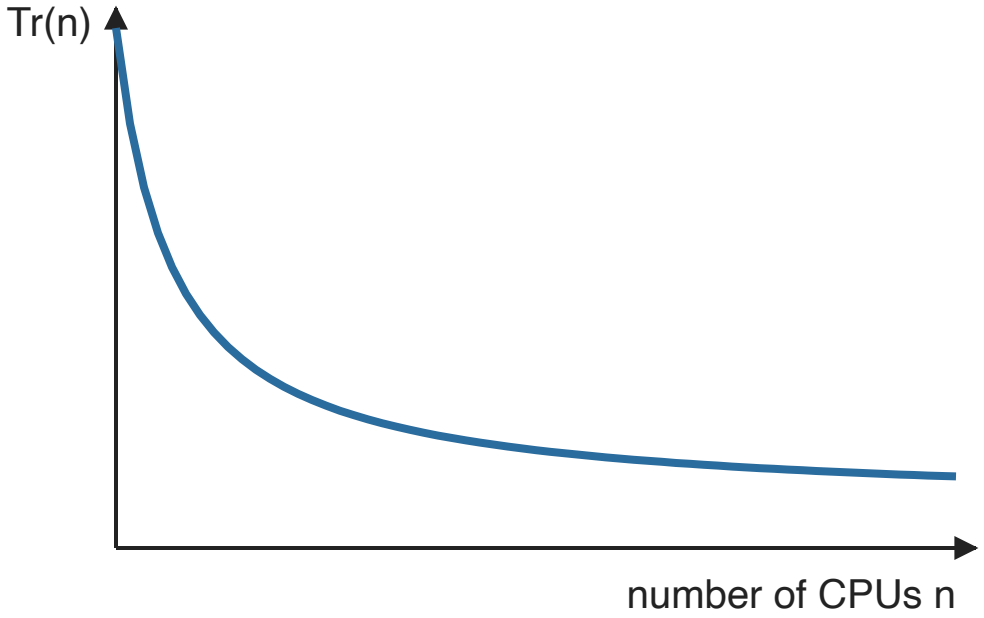
- Too much → idle resources → waste of resources
- Too few → job killed → waste of resources

Let t be the requested time, m the requested memory, n the requested number of CPUs, and ϵ the risk for your job to be killed due to limit trespassing. The problem is:

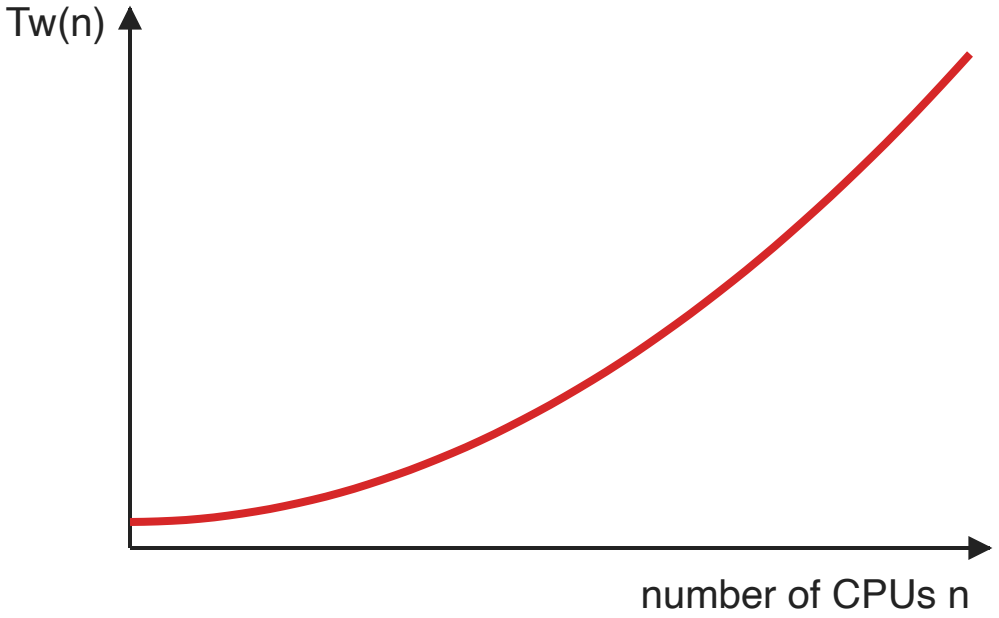
$$\min_{t,m,n} \; T_w(t,m,n) + T_r(n) \quad \text{subject to} \quad P\big(T_r(n) > t\big) < \epsilon, \quad P\big(M_r(n) > m\big) < \epsilon$$

with $T_w(t,m,n)$ the job waiting time in the queue, $T_r(n)$ the job running time, and $M_r(n)$ the job memory usage.

More CPUs: shorter run time, longer wait time



More CPUs → less running time



More CPUs → more waiting time

Practical approach

Run a sized-down problem on your laptop or the frontend and observe memory usage and CPU usage for several values of the number of CPUs with the `top` command.

```
top - 14:13:10 up 57 days,  5:06, 14 users,  load average: 1.56, 1.34, 1.35
Tasks: 557 total,   2 running, 555 sleeping,   0 stopped,   0 zombie
Cpu(s):  9.0%us,  6.3%sy,  0.0%ni, 84.4%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:  65957916k total, 63904772k used,  2053144k free,   306688k buffers
Swap: 33554428k total,  1919120k used, 31635308k free, 21674972k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
29436 user1     20   0  662m 137m 8468 R 100.0  0.2   2975:39 casm-learn
 2908 root      20   0 6657m  19m 1932 S 83.9  0.0   2478:14 beegfs-meta/Mai
65485 user2     20   0 14100 1544  920 S  2.0  0.0   1:32.05 htop
```

Practical approach

You can also use `/usr/bin/time -v` (full path, not the shell built-in `time`). Look at *User time*, *Percent of CPU this job got* and *Maximum resident set size*.

```
$ /usr/bin/time --verbose timeout 5s yes > /dev/null
Command exited with non-zero status 124
        Command being timed: "timeout 5s yes"
        User time (seconds): 4.92
        System time (seconds): 0.06
        Percent of CPU this job got: 99%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:05.00
        [...]
        Maximum resident set size (kbytes): 776
        [...]
        Exit status: 124
```

Pragmatic approach

1. Use guesstimates for the first job
2. Then analyse the accounting information
3. Extrapolate for the next jobs

Use the `sstat` command for running steps

```
sstat(1)                    Slurm Commands

NAME
       sstat - Display various status information of a running job/step.

SYNOPSIS
       sstat [OPTIONS...]

DESCRIPTION
       Status information for running jobs invoked with Slurm.

       The sstat command displays job status information for your
       analysis. The sstat command displays information pertaining to
       CPU, Task, Node, Resident Set Size (RSS) and Virtual Memory (VM).
       You can tailor the output with the use of the --fields= option
       to specify the fields to be shown.
```

Recall: steps are started with `srun`

Use the `sacct` command for completed jobs

```
SACCT(1)                    Slurm components

NAME
       sacct - displays accounting data for all jobs and job steps in
       the SLURM job accounting log or SLURM database

SYNOPSIS
       sacct [OPTIONS...]

DESCRIPTION
       Accounting information for jobs invoked with SLURM are either
       logged in the job accounting log file or saved to the SLURM
       database.
```

Use the `sacct` command for completed jobs

```
$ sacct --format Jobid,ReqMem,MaxRSS,TimeLimit,Elapsed,AllocCPUS,CPUTime,TotalCPU
       JobID     ReqMem     MaxRSS  Timelimit    Elapsed  AllocCPUS   CPUTime  TotalCPU
------------ ---------- ---------- ---------- ---------- ---------- --------- ---------
   12329             1Gc              00:05:00   00:03:22          2  00:06:44 06:20.781
   12329.ba+         1Gc     13820K               00:03:22          2  00:06:44 06:20.780
   12329.ex+         1Gc      1044K               00:03:22          2  00:06:44 06:20.780
   12329.0           1Gc      1044K               00:00:00          2  00:06:44 00:00.001
   12329.1           1Gc      1044K               00:03:21          2  00:06:44 06:20.780
```

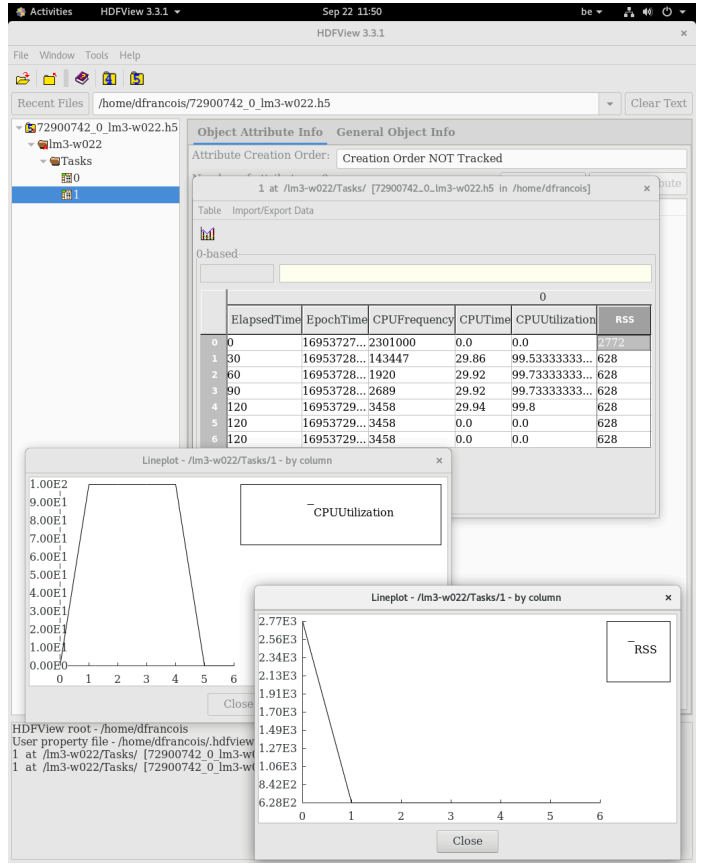
- `JobID` job ID . step ID
- `ReqMem` requested memory (Gc: GB per core)
- `MaxRSS` memory actually used (RSS)
- `Timelimit` time requested with `--time`
- `Elapsed` time actually used
- `AllocCPUs` number of CPUs allocated
- `CPUTime` CPU time allocated (Elapsed × CPUs)
- `TotalCPU` CPU time actually consumed

Use `--profile` for detailed information

- Time series for CPU usage, memory, etc.
- Might not be available on all clusters
- Self-service alternative: `sps`, which creates `.csv` files —> https://github.com/OxfordCBRG/sps

```
$ scontrol show config | grep AcctGatherProfileType
AcctGatherProfileType   = acct_gather_profile/hdf5
$ scontrol show config | grep ProfileHDF5Dir
ProfileHDF5Dir          = /scratch/acct_gather
$ sbatch --time 5:00 -n2 --wrap "srun stress -c 2 -t 120" --profile=all
Submitted batch job 72900743 on cluster lemaitre3
```

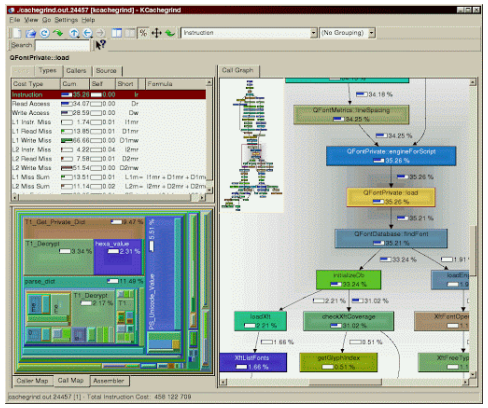
```
$ pwd
/scratch/acct_gather/dfr
$ ls
72900724_0_lm3-w022.h5      72900742_0_lm3-w022.h5      72900743_0_lm3-w026.h5
72900724_batch_lm3-w022.h5  72900742_batch_lm3-w022.h5  72900743_batch_lm3-w026.h5
```



Best approach

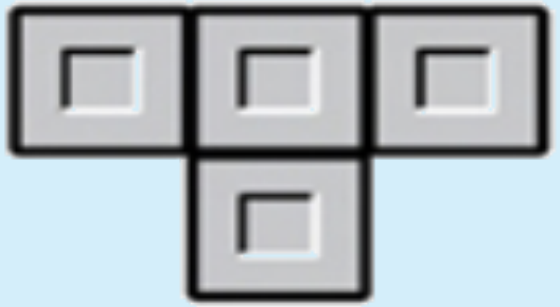
Use profiling tools…





`perf`, Valgrind/Callgrind (with KCachegrind), `gprof`, HPCToolkit, Score-P, TAU

Part 3 — Priorities, fairshare and scheduling



You will learn how to:

- understand priorities, fairshare, and scheduling

Priority is a weighted sum of job and account factors

```
Job_priority =
    (PriorityWeightAge) * (age_factor) +
    (PriorityWeightFairshare) * (fair-share_factor) +
    (PriorityWeightJobSize) * (job_size_factor) +
    (PriorityWeightPartition) * (partition_factor) +
    (PriorityWeightQOS) * (QOS_factor) +
    SUM(TRES_weight_cpu * TRES_factor_cpu,
        TRES_weight_<type> * TRES_factor_<type>,
        ...)
```

https://slurm.schedmd.com/priority_multifactor.html

Use `sprio` to get the details

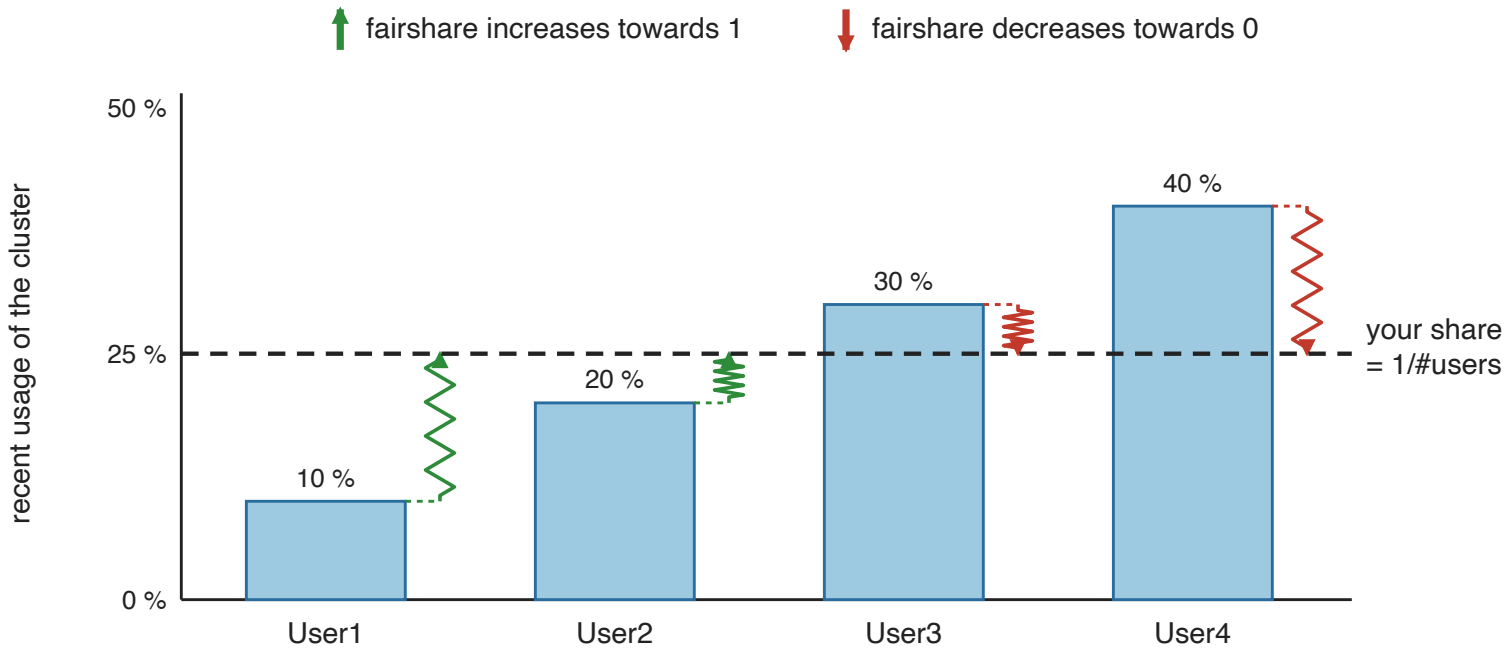
```
SPRIO(1)                    SLURM commands

NAME
       sprio - view the factors that comprise a job's scheduling priority

SYNOPSIS
       sprio [OPTIONS...]

DESCRIPTION
       sprio is used to view the components of a job's scheduling
       priority when the multi-factor priority plugin is installed.
       sprio is a read-only utility that extracts information from the
       multi-factor priority plugin. By default, sprio returns
       information for all pending jobs. Options exist to display
       specific jobs by job ID and user...
```

The fairshare factor helps everyone get access to resources



Past usage fades over time: usage two months ago counts less than usage two days ago.

The Slurm fairshare formula

The Slurm Fairshare Formula

- The Slurm Fairshare formula has been designed to provide fair scheduling to users based on the allocation and usage of **every** account. Now, the usage term is **effective** usage:

$F = 2^{(-U_E/S)}$ (Effective Usage Formula)

$U_E = U_{Achild} + ((U_{Eparent} - U_{Achild}) * S_{child}/S_{all_siblings})$

Where:

U_E is the effective usage of the child user or child account

U_{Achild} is the actual usage of the child user or child account

UE_{parent} is the effective usage of the parent account

S_{child} is the shares allocated to the child user or child account

$S_{all_siblings}$ is the shares allocated to all the children of the parent account

Copyright 2019 SchedMD LLC

www.schedmd.com

SLUG Sep 17-18, 2019

The fairshare decay factor

Fairshare-Decay Factor

- Most workload spans multiple time periods. Slurm's fairshare priority calculation places more importance on the most recent resource usage and less importance on usage from way back
- The metric used is based on a half-life formula that favors most recent usage statistics, based on a **decay** factor (D):

$$U_H = U_{current_period} + (D * U_{last_period}) + (D * D * U_{period-2}) + \ldots$$

Where:

U_H	is the historical usage subject to the half-life decay
$U_{current_period}$	is the usage charged over the current measurement period
U_{last_period}	is the usage charged over the last measurement period
$U_{period-2}$	is the usage charged over the second last measurement period
D	is a decay factor between zero and one that delivers the half-life decay based off the PriorityDecayHalfLife setting in the slurm.conf file

Get your current share with `sshare`

```
SSHARE(1)                       SLURM Commands

NAME
       sshare - Tool for listing the shares of associations to a cluster.

SYNOPSIS
       sshare [OPTIONS...]

DESCRIPTION
       sshare is used to view SLURM share information. This command is
       only viable when running with the priority/multifactor plugin.
       The sshare information is derived from a database with the
       interface being provided by slurmdbd (SLURM Database daemon)
       which is read in from the slurmctld and used to process the
       shares available to a ...
```

Get your current share with `sshare`

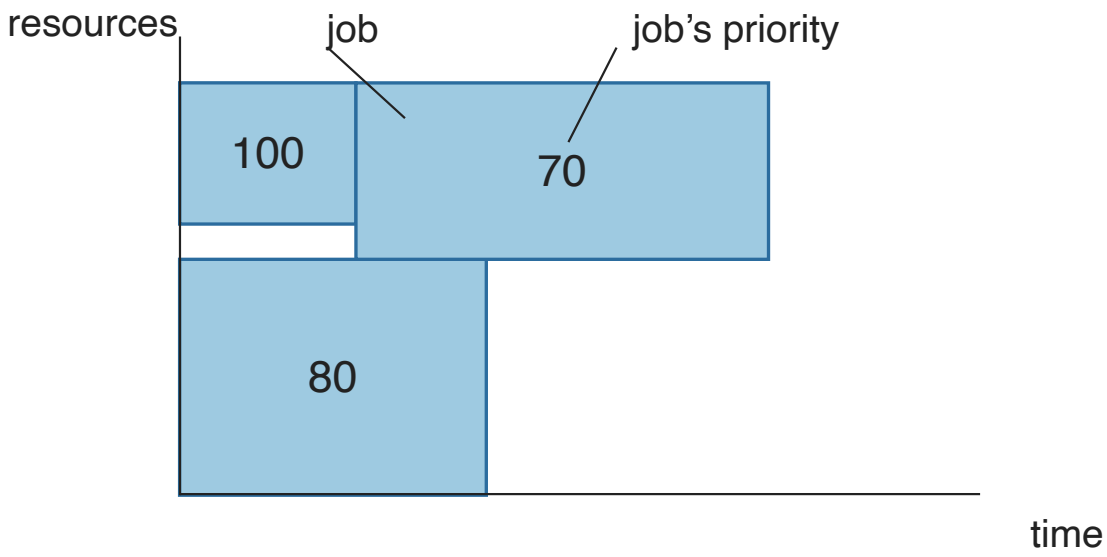
```
$ sshare -l
Account                    User  RawShares  NormShares    RawUsage   NormUsage  EffectvUsage  FairShare
-------------------- ---------- ---------- ----------- ----------- ----------- ------------- ----------
root                                          1.000000   823547414                 1.000000   0.870551
 ceci                                1000000    0.999998   823547414    1.000000      1.000000   0.870550
  ceci                    User1           1    0.000248      672111    0.000816      0.001064   0.551422
```

Fairshare of User1, step by step

- Normalised share for CÉCI: `1000000 / (1000000 + 1 + 1) = 0.999998`
- Normalised share for a CÉCI user: `0.999998 × 1/4037 = 0.0002477081992`
- RawUsage User1 = `672111`
- NormalisedUsage User1 = `672111 / 823547414 = 0.0008161169455`
- EffectiveUsage User1 = `0.0008161169455 + (1 − 0.0008161169455) × 0.0002477081992 / 0.999998 = 0.001063623481`
- FairShare User1 = `2^(-0.001063623481 / 0.0002477081992 / 5) = 0.5514219814`

Resources are “reserved” for top job but small jobs can be “backfilled”

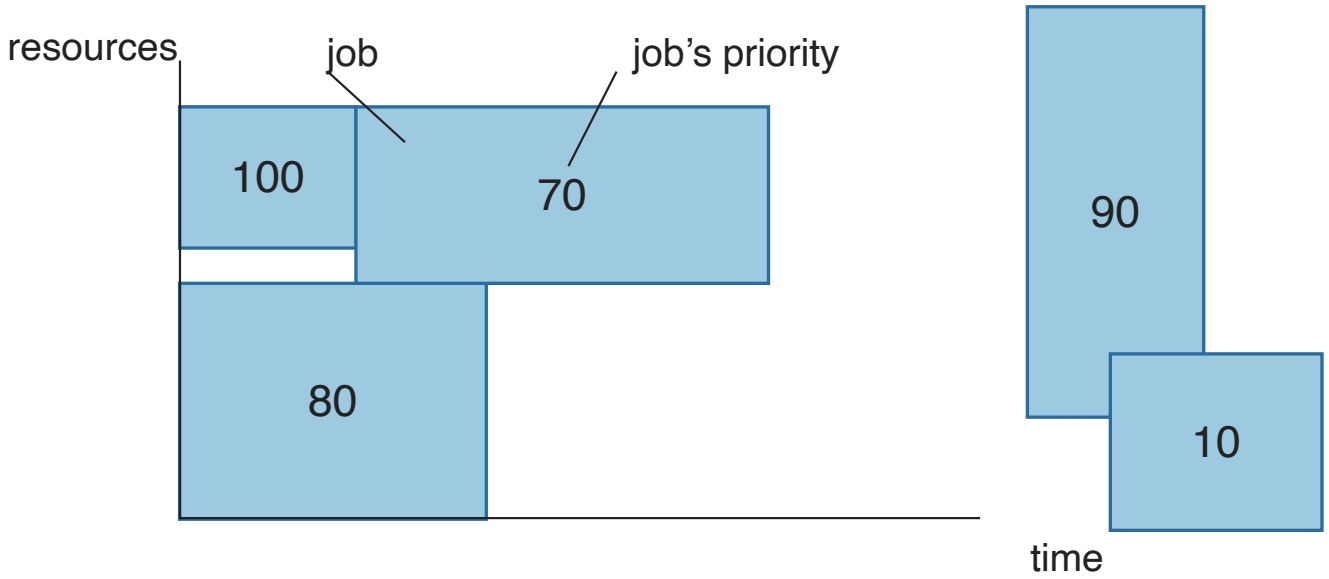
A job with a lower priority can start before a job with a higher priority if it does not delay that job's start time.



A job consumes resources for a specific duration

Resources are “reserved” for top job but small jobs can be “backfilled”

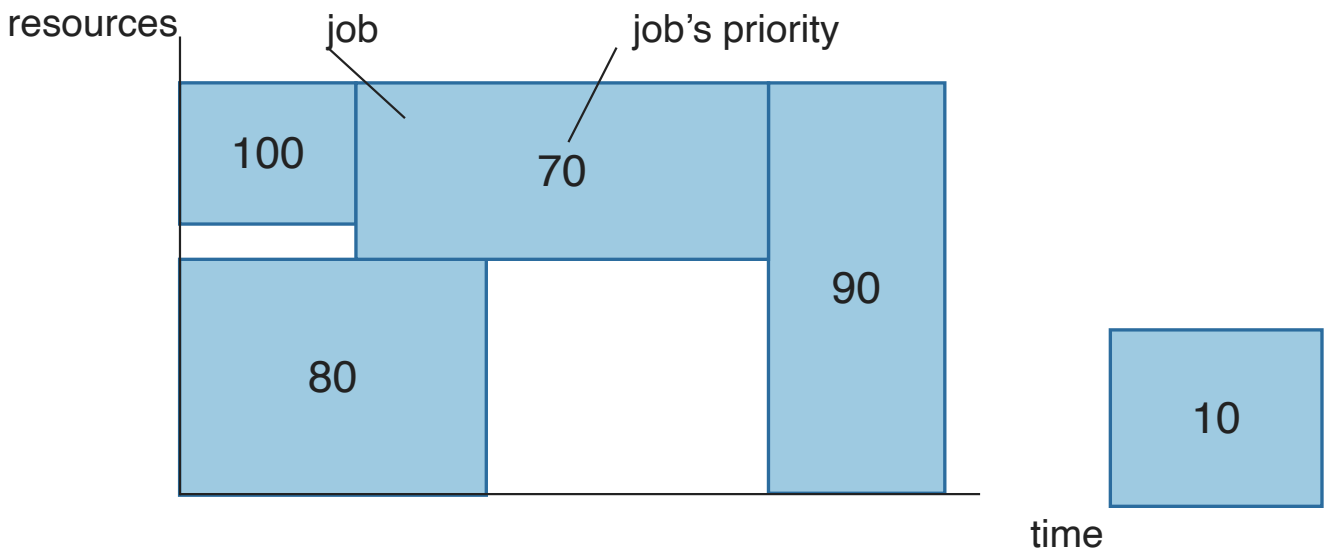
A job with a lower priority can start before a job with a higher priority if it does not delay that job's start time.



Two more jobs to schedule

Resources are “reserved” for top job but small jobs can be “backfilled”

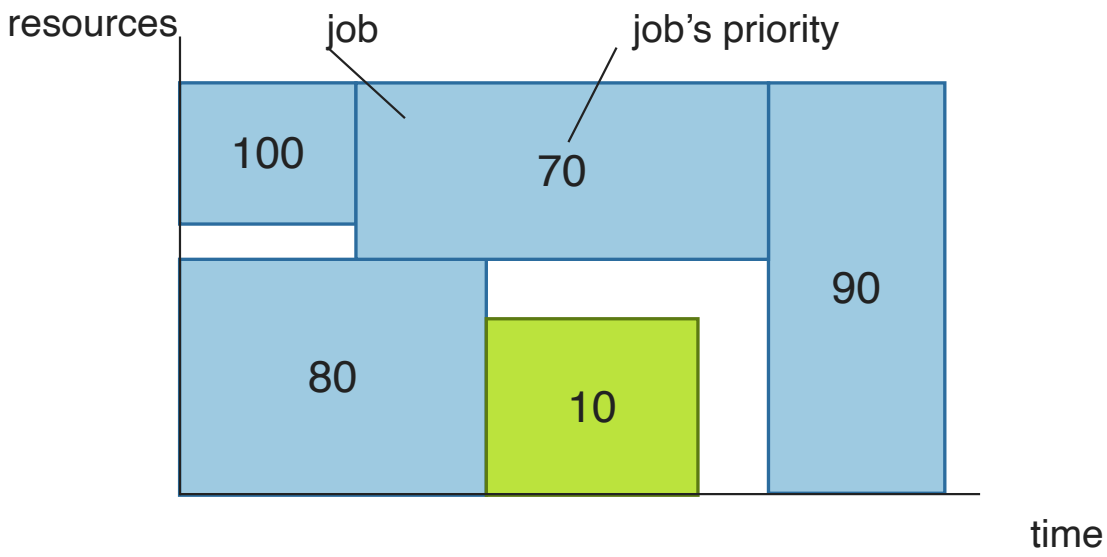
A job with a lower priority can start before a job with a higher priority if it does not delay that job's start time.



Job 90 waits for job 70 to finish: it needs its resources

Resources are “reserved” for top job but small jobs can be “backfilled”

A job with a lower priority can start before a job with a higher priority if it does not delay that job's start time.



Job 10 is short and small: it starts before the higher-priority job 90

Part 4 — Submission scripts for typical workloads



You will learn how to write submission scripts for:

- Multi-node SPMD programs (e.g. MPI)
- Single-node shared-memory programs (e.g. OpenMP)
- Master/slave programs
- Embarrassingly parallel workloads
- Accelerators (GPUs)

Jobs, steps, tasks and CPUs

A parallel job runs one or more **steps**, one per `srun` call; each step starts several **tasks** (processes) that run in parallel.

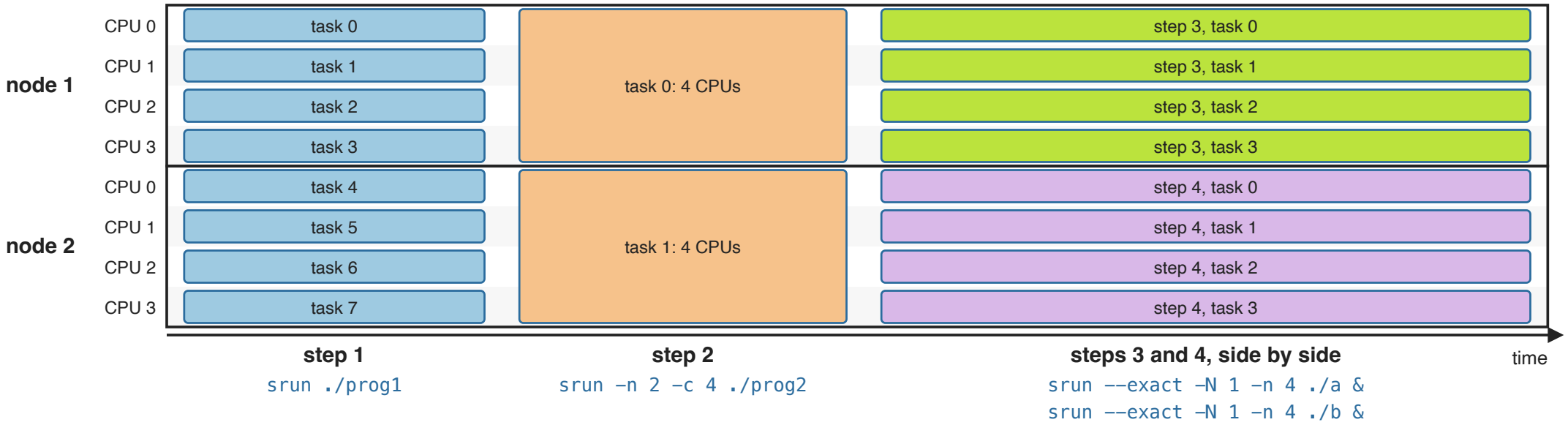
Option	Controls
`--ntasks`	number of tasks (processes)
`--cpus-per-task`	CPUs (threads) per task
`--nodes`	nodes the tasks are spread over

`srun` inherits these values from the job:

```
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=4
srun ./prog        # 2 processes × 4 threads = 8 CPUs
```

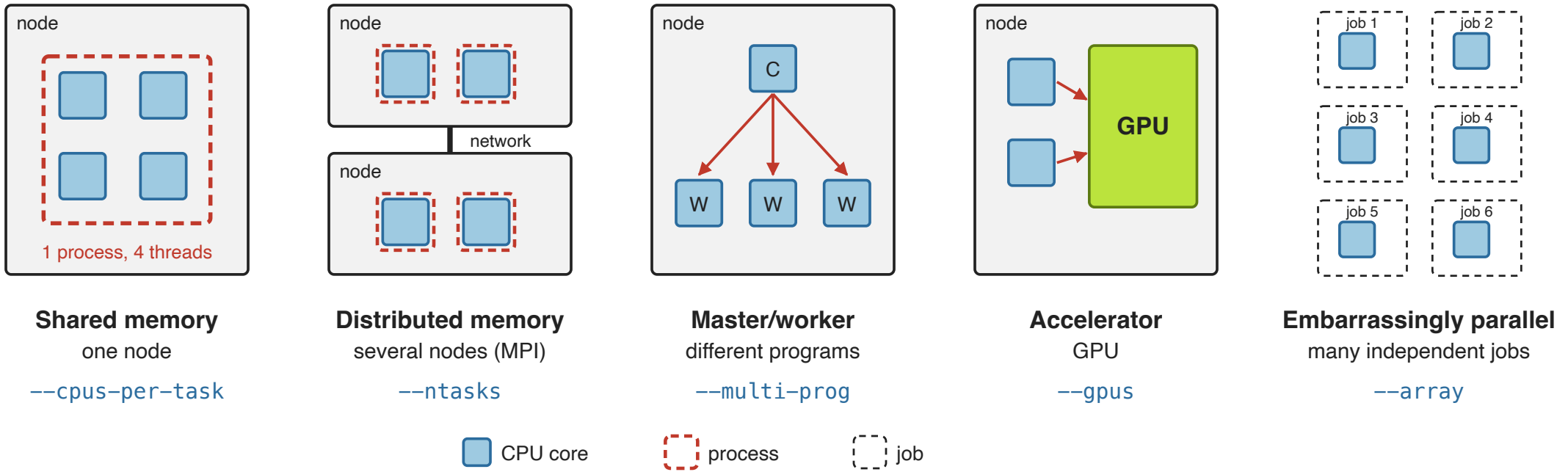
Steps run one after the other, or side by side

Job: `--nodes=2 --ntasks-per-node=4` (8 CPUs)



A task can use several CPUs but never spans two nodes. Run steps side by side with `&` and `wait`; `--exact` gives each step only the CPUs it asks for.

Clusters work best with parallel jobs



Shared memory
one node
`--cpus-per-task`

Distributed memory
several nodes (MPI)
`--ntasks`

Master/worker
different programs
`--multi-prog`

Accelerator
GPU
`--gpus`

Embarrassingly parallel
many independent jobs
`--array`

Which type you can use depends on the software: check its documentation.

Which option for which type of job

Job type	Slurm option	Slide
Shared memory, one node	`--cpus-per-task`	64
Distributed memory (MPI)	`--ntasks`	67
Master/worker	`--ntasks` + `--multi-prog`	70
Accelerators (GPU)	`--gpus` / `--gres`	72
Embarrassingly parallel	`--array`	75

Copy the example scripts to try them:

```
$ cp -r /CECI/proj/training/slurm ~/slurm-examples
```

Single-node job: Specify a number of `cpus`

You want	You ask
N CPUs to launch N threads or processes on the same node (= single task)	`--cpus-per-task=N`

`submit-omp.sh`

```
#! /bin/bash
#SBATCH --cpus-per-task=3
module load GCC
gcc -fopenmp /CECI/proj/training/slurm/omp_hello_world.c -o omp_hello_world
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./omp_hello_world
```

Software that uses several CPUs on one node

- Python: NumPy/SciPy (multithreaded BLAS), `multiprocessing`, `joblib`
- R: `parallel`, `doParallel`, `data.table`
- MATLAB (built-in multithreading, local `parpool`)
- Julia (`julia -t N`)
- BLAST+, BWA, Bowtie2, STAR, `samtools -@`
- Gaussian (`%NProcShared`)
- GROMACS (thread-MPI build)

Check the software documentation for its "threads" or "cores" option, and set it to `$SLURM_CPUS_PER_TASK`.

Examples only; the list is not exhaustive.

Full node(s) job

You want	You ask
All the CPUs on the node and all the memory	`--nodes=1` `--exclusive` `--mem=0`

`submit-omp.sh`

```
#! /bin/bash
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --mem=0
module load GCC
gcc -fopenmp /CECI/proj/training/slurm/omp_hello_world.c -o omp_hello_world
export OMP_NUM_THREADS=$SLURM_CPUS_ON_NODE
./omp_hello_world
```

Multi-node job: Specify a number of “tasks”

You want	You ask
N CPUs, to launch N MPI processes	`--ntasks=N`

`submit-mpi.sh`

```
#! /bin/bash
#SBATCH --ntasks=4
module load OpenMPI
mpicc /CECI/proj/training/slurm/mpi_hello_world.c -o mpi_hello_world
#mpirun ./mpi_hello_world
srun ./mpi_hello_world
```

Software that runs on several nodes with MPI

- ABINIT (developed at UCLouvain)
- Quantum ESPRESSO, VASP, CP2K, SIESTA
- LAMMPS, GROMACS (MPI build), NAMD
- OpenFOAM, Code_Saturne
- WRF
- Python with `mpi4py`

Some programs start MPI themselves:

- ORCA runs `mpirun` itself: start it with its full path, without `srun`
- R with Rmpi spawns its own workers (`mpi.spawn.Rslaves()`): start a single R process

Examples only; the list is not exhaustive.

Specify a number of “tasks” and a number of “nodes”

You want	You ask
N CPUs spread across distinct nodes	`--ntasks=N --nodes=N` or `--ntasks=N --ntasks-per-node=1`
N CPUs spread across distinct nodes and nobody else around	`--nodes=N --exclusive`
N CPUs spread across N/2 nodes	`--ntasks=N --ntasks-per-node=2`
N CPUs on the same node	`--ntasks=N --ntasks-per-node=N` or `--ntasks=N --nodes=1`
Between 8 and 16 CPUs	`--nodes=4-8 --ntasks-per-node=2`

Use `srun --multi-prog`

You want	You ask
N CPUs to launch N processes	`--ntasks=N`

`submit-masterslave.sh`

```
#! /usr/bin/env bash
#SBATCH --ntasks=3
cp /CECI/proj/training/slurm/coordinator.sh .
cp /CECI/proj/training/slurm/worker.sh .
cp /CECI/proj/training/slurm/multi.conf .
srun --multi-prog multi.conf
```

`multi.conf`

```
0       ./coordinator.sh
1-2     ./worker.sh
```

Software that uses several different programs in one step

- Coupled models: EC-Earth (IFS atmosphere + NEMO ocean, coupled through OASIS)
- NEMO with XIOS I/O servers
- Your own coordinator/worker scripts

Dask, Ray and Spark use the same coordinator/worker idea, but are usually started with separate `srun` calls or helpers such as `dask-jobqueue`.

Examples only; the list is not exhaustive.

Request a GPU with `--gres` or `--gpus`

You want	You ask
N GPUs	`--gpus=N`
N GPUs per node	`--gres=gpu:N`
1 specific GPU (e.g. TeslaV100)	`--gpus=TeslaV100:1` or `--gres=gpu:TeslaV100:1`

`submit.sh`

```
#! /bin/bash
#SBATCH --cpus-per-task=3
#SBATCH --mem-per-cpu=1g
#SBATCH --gres=gpu:2
module load CUDA # or cuda on some clusters
nvidia-smi
```

Software that uses GPUs

- PyTorch, TensorFlow/Keras, JAX
- AlphaFold, ColabFold
- GROMACS, AMBER (`pmemd.cuda`), NAMD, LAMMPS (GPU/Kokkos)
- Quantum ESPRESSO and VASP GPU builds
- RELION, cryoSPARC
- vLLM, llama.cpp

Examples only; the list is not exhaustive.

Hybrid jobs

with for instance MPI and OpenMP

`submit.sh`

```
#! /bin/bash
#
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=3
module load OpenMPI
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun ./hello_world_mpi
```

Create job arrays with `--array`

You want	You ask
N CPUs to launch N completely independent jobs	`--array=1-N`

`submit-array.sh`

```
#! /usr/bin/env bash
#SBATCH --array=1-4
[ ! -f ./array_hello.sh ] && \
     cp /CECI/proj/training/slurm/array_hello.sh .
./array_hello.sh $SLURM_ARRAY_TASK_ID
```

The script generates 4 jobs. Each job "sees" a different value for `$SLURM_ARRAY_TASK_ID`.

```
CLUSTER: lemaitre3
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    72772281_[1-4]     batch array_he      dfr PD       0:00      1 (Priority)
```

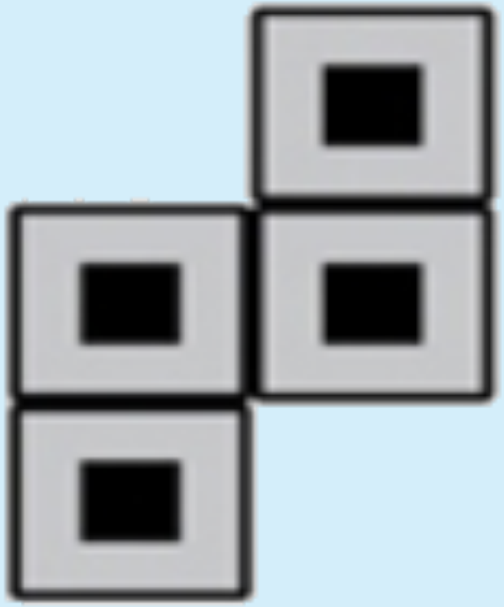
Set job dependencies with `--dependency`

You want	You ask
Job B to start after job A	`--dependency=afterok:<JOBID of A>`

Using `--dependency=afterok:12345`, the submitted job only starts after job 12345 has completed successfully. Dependent jobs wait for the dependee.

```
$ sbatch /CECI/proj/training/slurm/job-dependee.sh
Submitted batch job 72772285 on cluster lemaitre3
$ sbatch --dependency=afterok:72772285 /CECI/proj/training/slurm/job-dependent.sh
Submitted batch job 72772286 on cluster lemaitre3
$ squeue  --me
CLUSTER: lemaitre3
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          72772286     batch dependen      dfr PD       0:00      1 (Dependency)
          72772285     batch dependee      dfr PD       0:00      1 (Priority)
```

Part 5 — Interactive sessions



You will learn how to:

- create an interactive Bash session
- launch JupyterLab or RStudio

Use `salloc` to test multi-node setups

e.g. `salloc --ntasks=4 --nodes=2`

```
salloc(1)                  SLURM Commands

NAME
       salloc - Obtain a SLURM job allocation (a set of nodes), execute
       a command, and then release the allocation when the command is
       finished.

SYNOPSIS
       salloc [options] [<command> [command args]]

DESCRIPTION
       salloc is used to allocate a SLURM job allocation, which is a
       set of resources (nodes), possibly with ...
```

Use `salloc` to test multi-node setups

```
$ salloc --partition debug --ntasks 2 --nodes 2
salloc: Pending job allocation 70299307
salloc: job 70299307 queued and waiting for resources
salloc: job 70299307 has been allocated resources
salloc: Granted job allocation 70299307
salloc: Waiting for resource configuration
salloc: Nodes lm3-w[091-092] are ready for job
[... login message ...]
$ ml OpenMPI
$ mpirun mpi_hello_world
Hello world from processor lm3-w091.cluster, rank 0 out of 2 processors
Hello world from processor lm3-w092.cluster, rank 1 out of 2 processors
$ exit
exit
salloc: Relinquishing job allocation 70299307
salloc: Job allocation 70299307 has been revoked.
```

Use `salloc` for a shell on a compute node

New canonical way of getting a shell, if the configuration has

`LaunchParameters=use_interactive_step`

```
[dfr@lm4-f001 ~]$ salloc -t 10:00
salloc: Granted job allocation 2082958
salloc: Waiting for resource configuration
salloc: Nodes lm4-w010 are ready for job
[dfr@lm4-w010 ~]$ hostname
lm4-w010
[dfr@lm4-w010 ~]$ exit
```

Use `srun` for a shell on a compute node

If `salloc` does not give you one:

```
srun --pty bash -l
```

```
dfr@nic5-login1 ~ $ srun --pty bash -l
dfr@nic5-w034 ~ $ hostname
nic5-w034
dfr@nic5-w034 ~ $
```

Start and connect a service on a compute node in 4 steps

1. On the login node, get a shell on a compute node with `srun --pty bash -l`
2. On the compute node, start the service and note its URL
3. On your computer, open an SSH tunnel (`ssh -D` or `ssh -L` or `sshuttle`)
4. In your browser, (configure a proxy if `ssh -D`), paste the URL from step 2.

Example: JupyterLab

1. Use `srun` for shell on compute node

```
dfr@nic5-login1 ~ $ srun --pty -c 4 bash -l
srun: job 1824417 queued and waiting for resources
srun: job 1824417 has been allocated resources
```

2. Load module and start service

Use the `--ip` option to get the right URL, then copy the URL with the token

```
dfr@nic5-w022 ~ $ ml releases/2020b JupyterLab
dfr@nic5-w022 ~ $ jupyter-lab --ip=$(hostname -i)
[I 09:49:01.611 LabApp] Serving notebooks from local directory: /home/users/d/f/dfr
[I 09:49:01.611 LabApp] Jupyter Notebook 6.1.4 is running at:
[I 09:49:01.612 LabApp] http://10.252.2.55:8888/?token=c8515c45ec8066710aa9ba1c2d15b897a27514157780a3cf
[I 09:49:01.612 LabApp]  or http://127.0.0.1:8888/?token=c8515c45ec8066710aa9ba1c2d15b897a27514157780a3cf
[I 09:49:01.612 LabApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 09:49:01.617 LabApp] No web browser found: could not locate runnable browser.
[C 09:49:01.618 LabApp]

    To access the notebook, open this file in a browser:
        file:///home/users/d/f/dfr/.local/share/jupyter/runtime/nbserver-2312329-open.html
    Or copy and paste one of these URLs:
        http://10.252.2.55:8888/?token=c8515c45ec8066710aa9ba1c2d15b897a27514157780a3cf
     or http://127.0.0.1:8888/?token=c8515c45ec8066710aa9ba1c2d15b897a27514157780a3cf
```

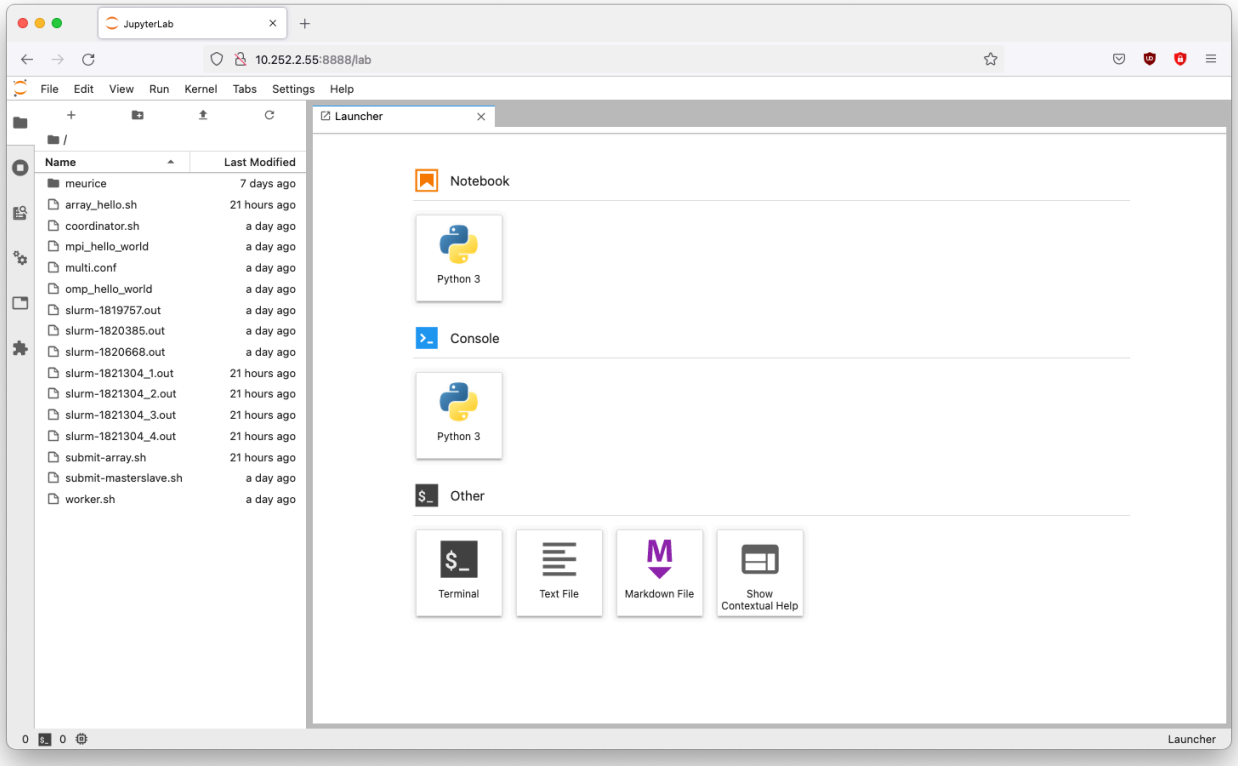
3. Create SSH tunnel

Run `ssh -L` in a new terminal and leave it open. Use the `10....` IP address shown in the output of Step 2.

```
$ ssh -L 8888:10.252.2.55:8888 nic5
[... login message ...]
dfr@nic5-login1 ~ $
```

4. Connect to URL

Paste the `127.0.0.1` URL you got in step 2 in the address bar: `http://127.0.0.1:8888/?token=…`



Example: RStudio

0. Install helper script

https://raw.githubusercontent.com/nickjer/singularity-rstudio/master/rstudio_auth.sh

`chmod +x rstudio_auth.sh`

```
$ wget https://raw.githubusercontent.com/nickjer/singularity-rstudio/master/rstudio_auth.sh
[...]
HTTP request sent, awaiting response... 200 OK
Length: 569 [text/plain]
Saving to: 'rstudio_auth.sh'

2021-11-04 10:16:08 (16.7 MB/s) - 'rstudio_auth.sh' saved [569/569]
```

1. Use `srun` for shell on compute node

```
$ srun --partition debug --pty bash -l
srun: job 70299666 queued and waiting for resources
srun: job 70299666 has been allocated resources
```

2. Load module and start service

1. Get the node's IP address with `hostname -i`
2. Choose a password and a port
3. Run the server

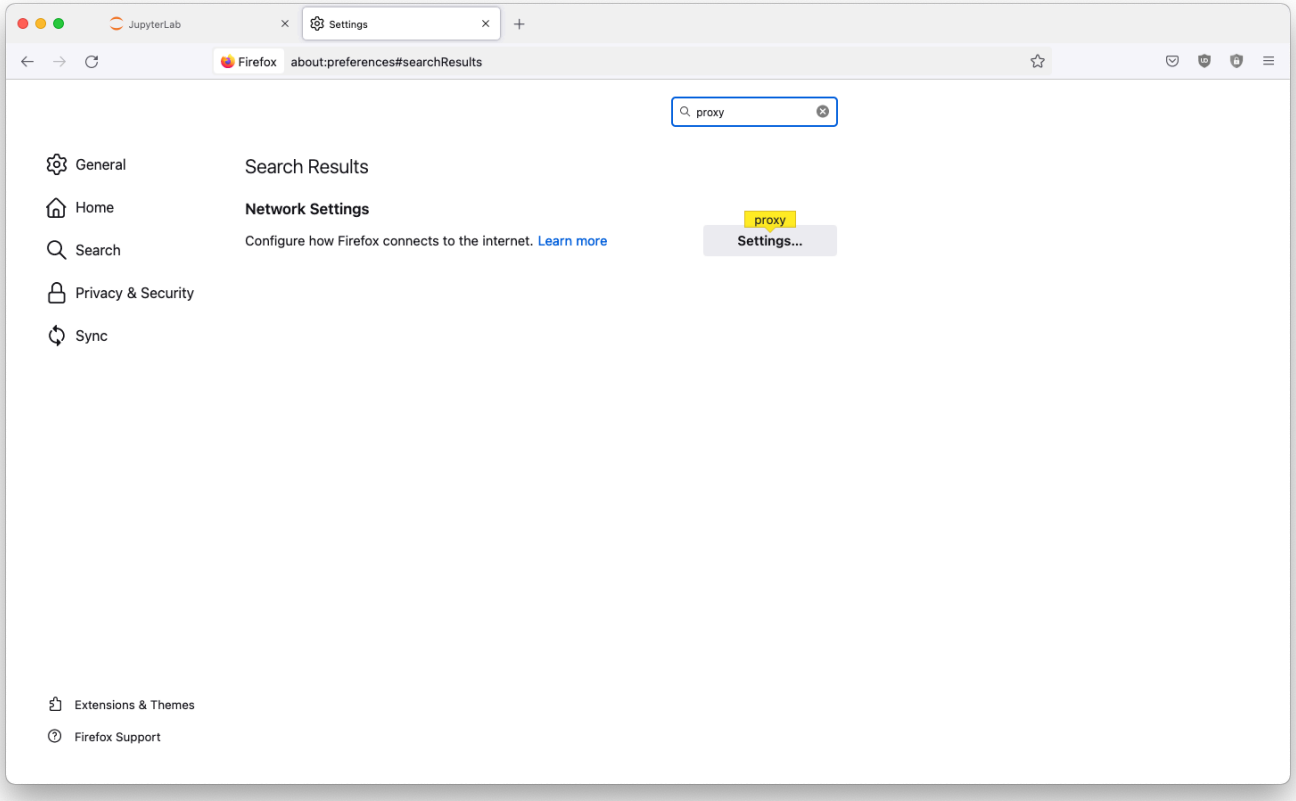
```
$ ml releases/2019b RStudio-Server/1.2.5042-foss-2019b-Java-11
$ export IP=$(hostname -i)
$ export PORT=8787
$ export RSTUDIO_PASSWORD="<choose-a-password>"
$ echo "http://$IP:$PORT"
http://10.7.1.94:8787
$ rserver --server-daemonize=0 --www-port $PORT \
    --rsession-which-r=$(which R) --auth-none 0 \
    --auth-pam-helper $PWD/rstudio_auth.sh
```

3. Create SSH tunnel (SOCKS proxy)

Run `ssh -D` in a new terminal and leave it open

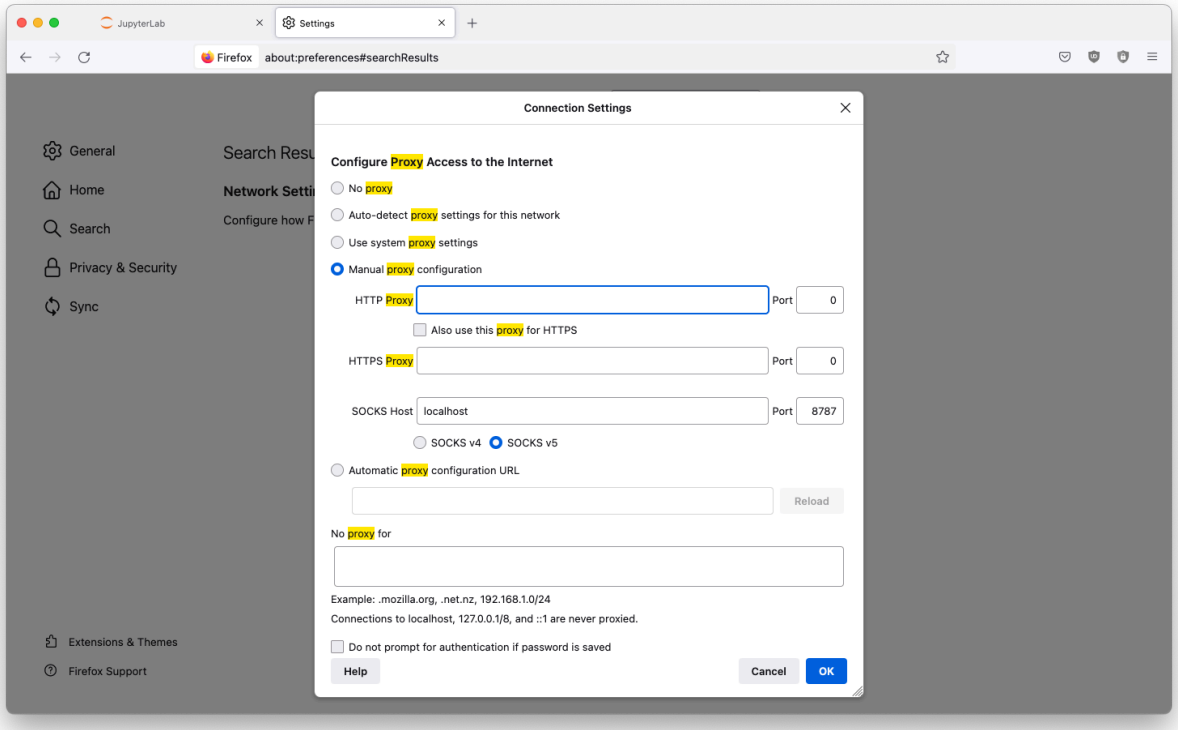
```
$ ssh -D 1080 nic5
[... login message ...]
dfr@nic5-login1 ~ $
```

4. Configure browser



4. Configure browser

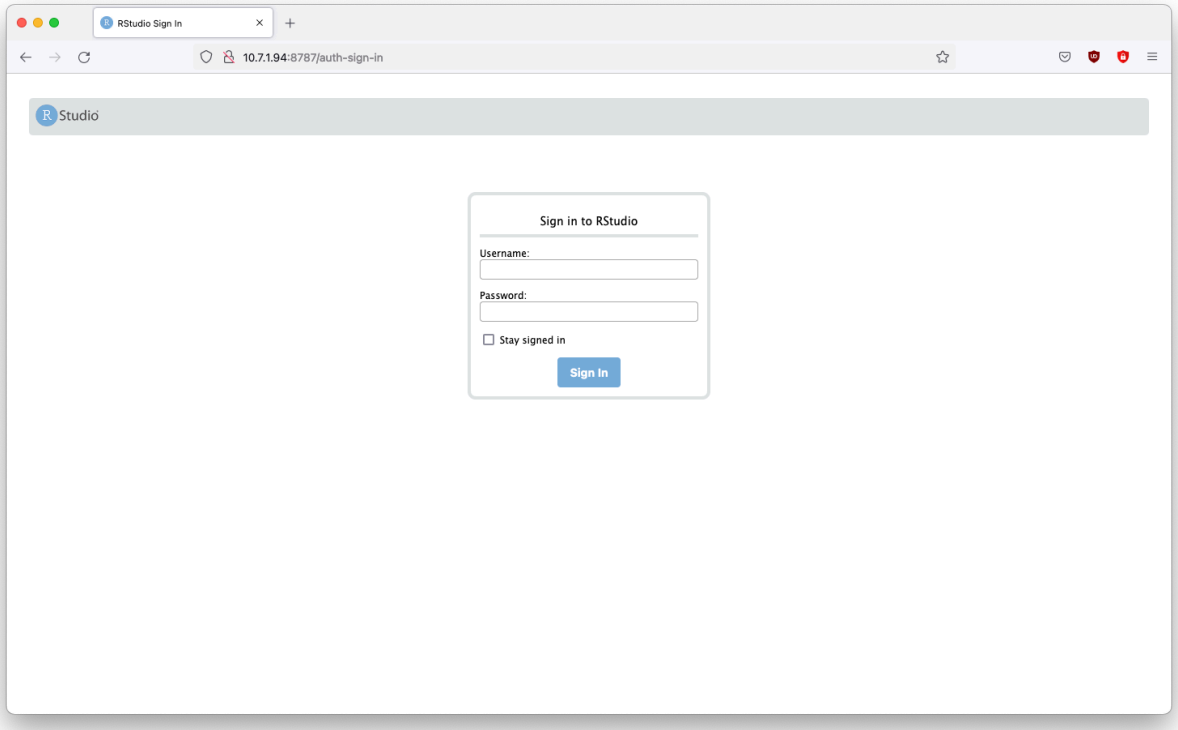
SOCKS Host: `localhost` ; Port: the same port you chose in step 3 (1080), SOCKS v5.



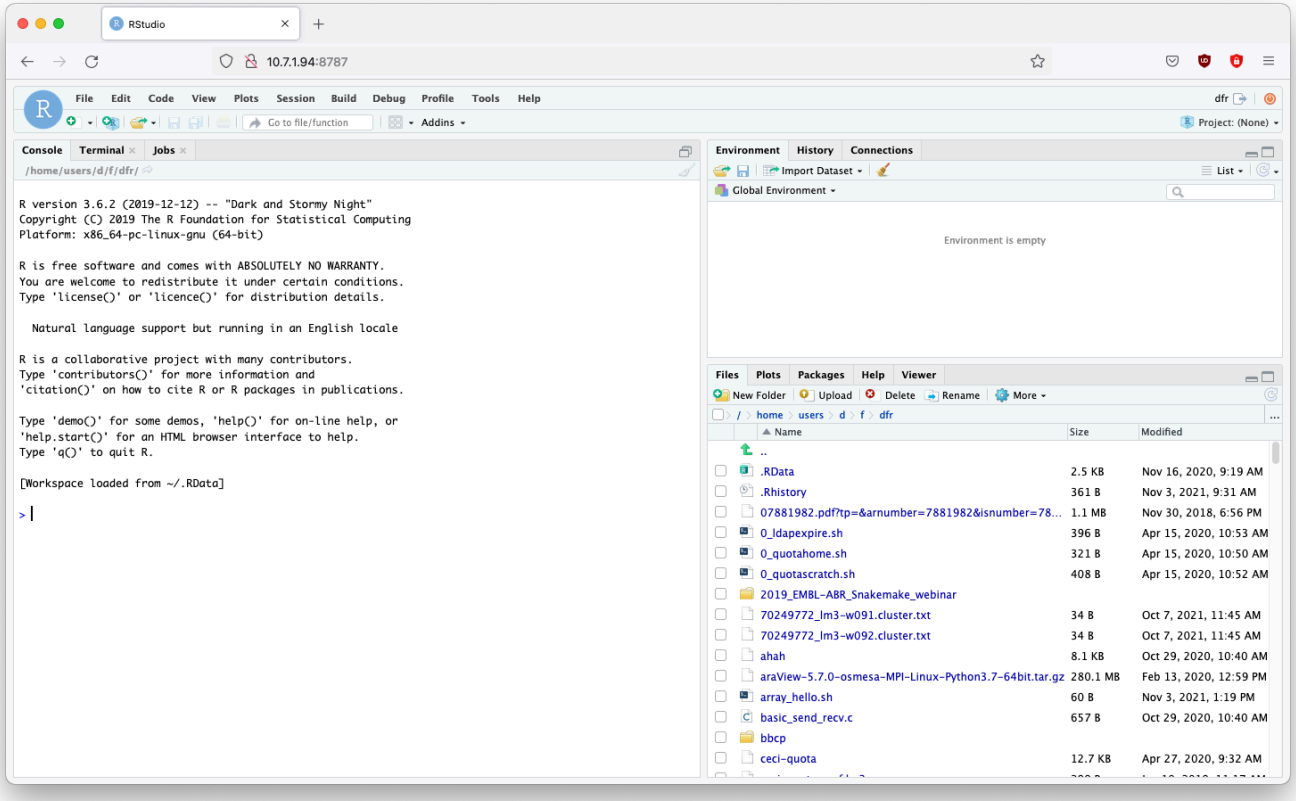
Note: This will prevent you from browsing the web: consider creating a specific profile

5. Connect to URL built at Step 2

Enter your CÉCI login and the password you chose in step 2.



5. Connect to URL built at Step 2



Example: VNC

0. Setup VNC password

```
[dfr@mb-icg101 ~]$ vncpasswd
Password:
Verify:
Would you like to enter a view-only password (y/n)? n
A view-only password is not used
```

1. Use `srun` for shell on compute node

Compute node name : virtual display number

```
[dfr@mbackf2 ~]$ srun --pty -p gpu --gres=gpu:1 bash -l
[dfr@mb-icg101 ~]$ which vncserver
/usr/bin/vncserver
[dfr@mb-icg101 ~]$ vncserver

WARNING: vncserver has been replaced by a systemd unit and is now considered deprecated and removed in upstream.
Please read /usr/share/doc/tigervnc/HOWTO.md for more information.

New 'mb-icg101.cism.ucl.ac.be:1 (dfr)' desktop is mb-icg101.cism.ucl.ac.be:1

Starting applications specified in /home/ucl/pan/dfr/.vnc/xstartup
Log file is /home/ucl/pan/dfr/.vnc/mb-icg101.cism.ucl.ac.be:1.log
```

2. Start the program within the display

`sviz` on Hercules helps the process

```
[dfr@mb-icg101 ~]$ export DISPLAY=:1
[dfr@mb-icg101 ~]$ gnome-text-editor
```

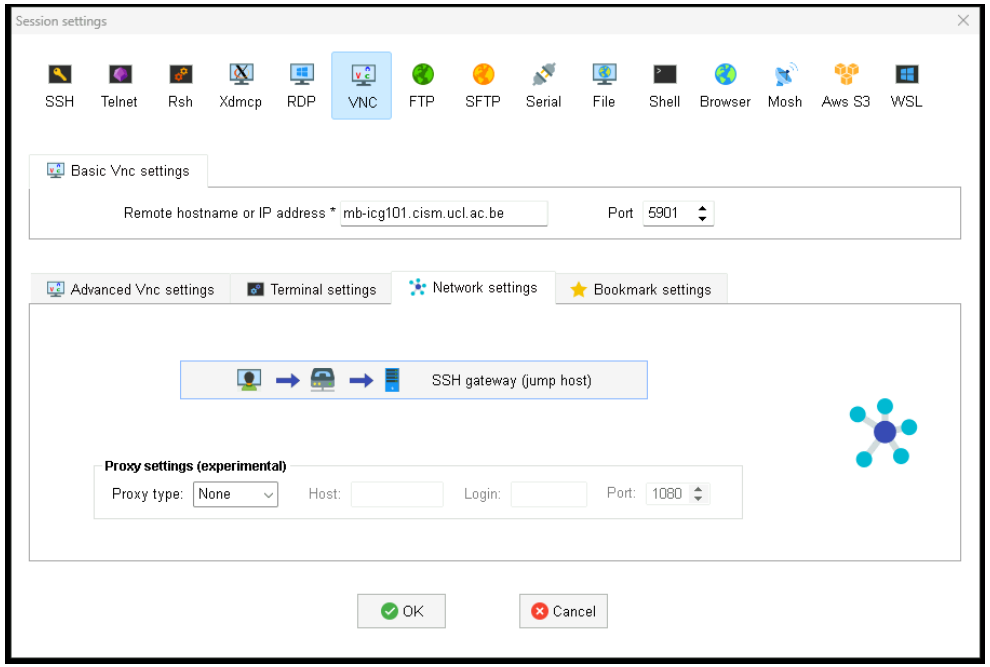
3. Configure MobaXterm

Click on the Session icon 

, then in the VNC icon 

and select **Network settings** tab  Network settings

Port = 5900 + virtual display number

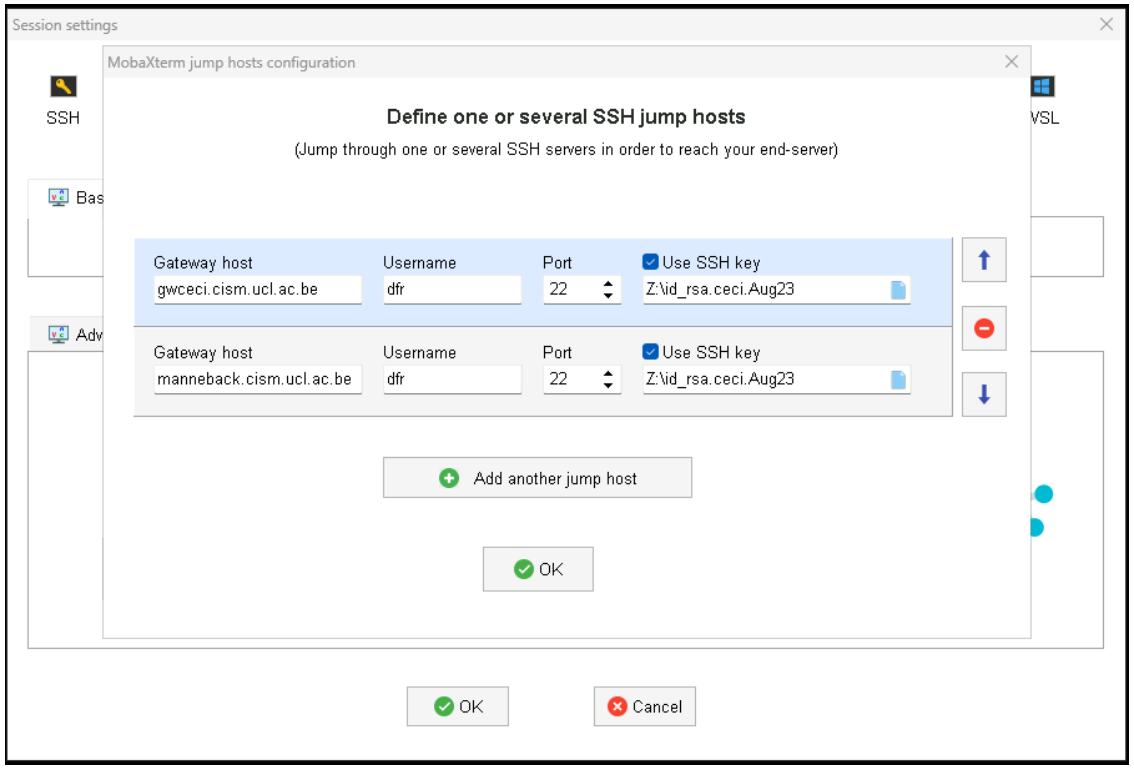


3. Configure MobaXterm

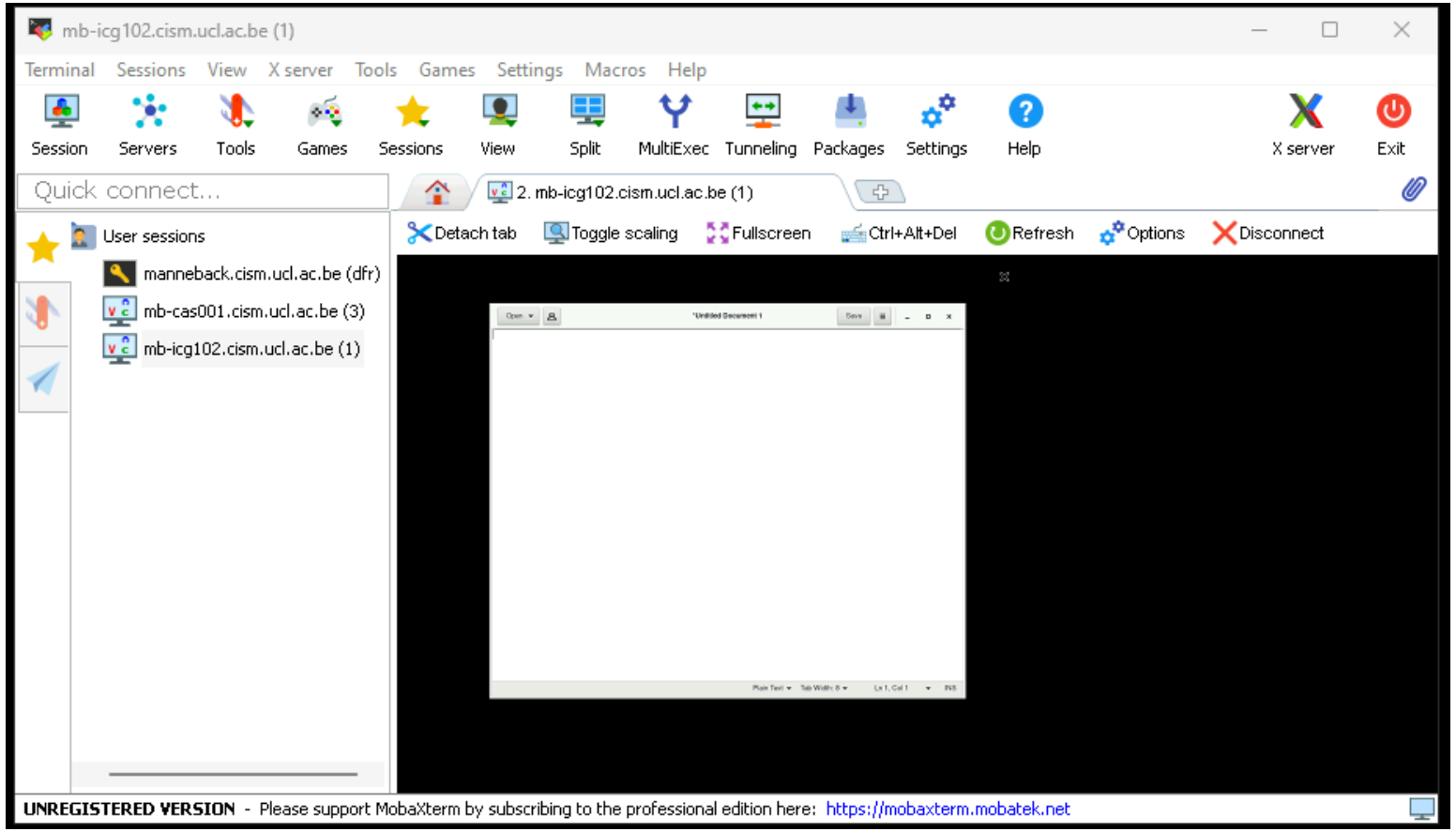
Click on the Session icon 

, then in the VNC icon 

and select **Network settings** tab 



4. Start the session



Example: VS Code

1. Start an SSH server in a job

This way is convoluted but preserves the whole Slurm job environment.

- Your public key must be in `~/.ssh/authorized_keys` on the cluster

```
$ module load Python
$ srun --job-name=tunnel --pty -t 10:00 /usr/sbin/sshd -D -p 2222 -f /dev/null -h $HOME/.ssh/id_ecdsa
```

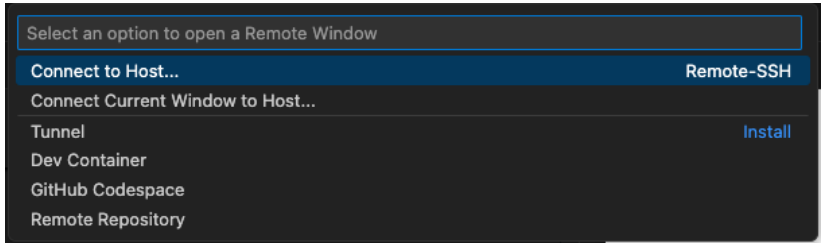
2. Configure SSH on your computer

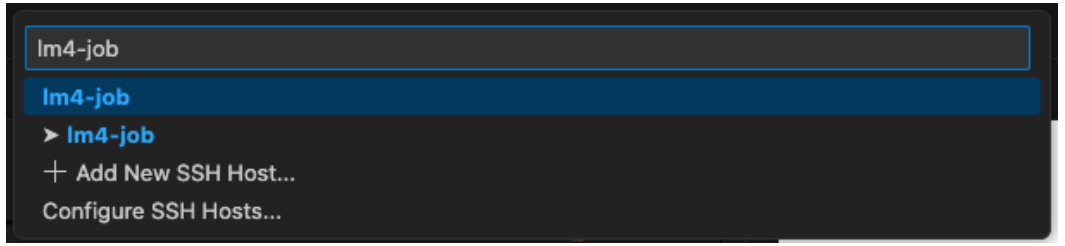
Add to `~/.ssh/config` on your computer (example for Lemaitre4):

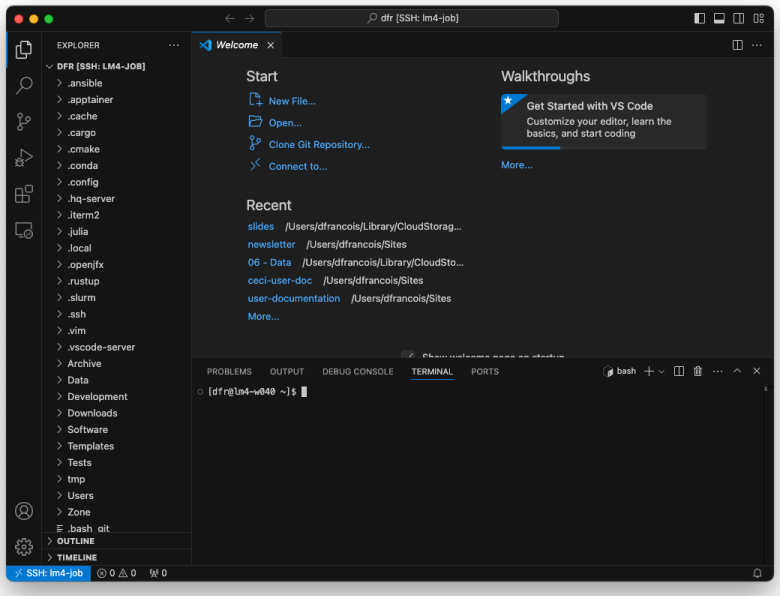
```
Host lm4-job
    user dfr
    ProxyCommand ssh lm4 "nc \$(squeue --me --name=tunnel --states=R -h -O NodeList) 2222"
    StrictHostKeyChecking no
```

- Beware of usernames: use your cluster username in `user`
- Use an SSH agent
- https://github.com/microsoft/vscode-remote-release/issues/1722

3. Connect with VS Code







⚠️ Interactive jobs often waste resources

- The job starts when Slurm decides, not when you are ready: the CPUs wait until you notice.
- Interactive work is iterative: think, run a command, read the output, think again… The CPUs sit idle most of the time.
- Sessions stay open during meetings, lunch or the night, until the time limit.
- Idle CPUs, GPUs and memory are blocked for others, and still count in your fairshare.

Explore on a small test, then turn the work into a batch job:

- Jupyter notebook: `jupyter nbconvert --to script your_notebook.ipynb`
- R: `R CMD BATCH script.R` or `Rscript script.R`
- MATLAB: use the MATLAB Compiler; the runtime is often installed: `ml spider MCR`

Keep interactive sessions short and small, and end them (`exit`, `scancel`) as soon as you are done.

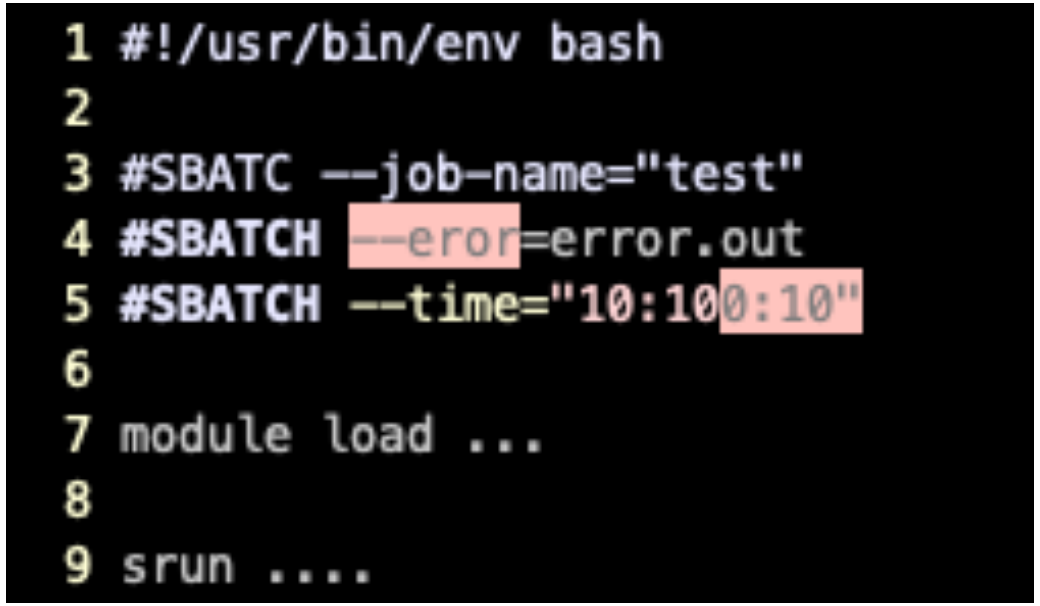
Debugging



Check Slurm's interpretation

```
$ sbatch --verbose --test-only submit.sh
sbatch: defined options
sbatch: -------------------- --------------------
sbatch: clusters             : lemaitre3
sbatch: mem                  : 4G
sbatch: ntasks               : 1
sbatch: test-only            : set
sbatch: time                 : 06:00:00
sbatch: verbose              : 1
sbatch: -------------------- --------------------
sbatch: end of defined options
[…]
sbatch: Job 72819422 to start at 2023-09-14T21:52:05 using 1 processors
on nodes lm3-w071 in partition batch
```

Use Vim syntax highlighting



https://github.com/damienfrancois/slurm-helper#vim-syntax-file

Good practice: “strict mode”

```
#!/bin/bash
set -euo pipefail
IFS=$'\n\t'
# -e immediately exit if any command [1] has a non-zero exit status.
# -o pipefail as above for commands in pipe sequences
# -u a reference to an undefined variable is an error (prevent typos)
# IFS define word splitting characters (prevent common errors with spaces in filenames)
```

Make sure the script stops when any command returns an error, and make the job status "FAILED" rather than "COMPLETED".

http://redsymbol.net/articles/unofficial-bash-strict-mode/

Good practice: “strict mode”

```
#!/bin/bash
#SBATCH ...
set -euo pipefail
IFS=$'\n\t'
cd $LOCALSRATCH
rm -rf *
[...]
```

Disaster prevented.

http://redsymbol.net/articles/unofficial-bash-strict-mode/

Good practice: exit codes

```
#!/bin/bash
#SBATCH ...
#SBATCH ...
cp /globalscratch/my/big/file /localscratch || exit 11 # Exit with code 11 if cp fails
[...]
```

```
$ sacct
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
123454321           bash      batch       ceci          2     FAILED     11:0
```

Some exit codes are "reserved" by Bash or Slurm. It is safe to use the ranges 10-39 and 65-120.

Exit codes: what Slurm records

- The job's exit code is the exit code of the submission script (`salloc`: of the last command). Each `srun` step is recorded too.
- `0` means success, anything else an error.
- Bash does not stop when a command fails, and returns the code of the **last** command: a final `echo Job finished` gives `0` even if everything failed. Use `set -e`.
- `srun` returns the highest exit code of its tasks.
- `sacct` shows `ExitCode` as `code:signal`:
  - `0:0` success
  - `2:0` the script, or its last command, returned `2`
  - `0:4` killed by signal 4 (illegal instruction): **not** a success

Common exit codes

Code	Meaning
`1`	General error, e.g. `$((10/0))`
`2`	Syntax error, bad option to a builtin, or executable not found
`126`	Command found but not executable
`127`	Command not found
`128+N`	Killed by signal `N`, e.g. `137` = SIGKILL, `139` = SIGSEGV
`0:125`	Out of memory (state `OUT_OF_MEMORY`); `srun` itself returns `253` = 128 + 125
`0:53`	Slurm could not create the `--output` or `--error` file

The final exit code could actually be set by Slurm, Bash, or your own program.

Good practice: trapping errors

```
#!/bin/bash
set -euo pipefail
trap 'echo "error: Script $0 failed line ${LINENO}: \"$BASH_COMMAND\": see above"' ERR
ls /nonexistent
```

Print script name, line, and command that failed. Useful when scripts call other scripts.

```
$ ./test.sh
ls: cannot access /nonexistent: No such file or directory
error: Script ./test.sh failed line 3: "ls /nonexistent": with error message above
```

Good practice: errors to `stderr`

```
#!/bin/bash
#SBATCH --output=out.txt
#SBATCH --error=err.txt
echo "some comment"
echo >&2 "some error"
```

Normal content is written to `out.txt`, errors are written to `err.txt`.

```
$ cat out.txt
some comment
$ cat err.txt
some error
```

Write errors to stderr so they are filtered to the `--error` file.

Bash verbosity

```
#!/bin/bash -vx
A=(/usr/l*)
ls -d ${A[@]}
```

Enable "trace" mode with `-vx` to see details of script execution.

```
$ ./test.sh
A=(/usr/l*)
+ A=(/usr/l*)
ls -d ${A[@]}
+ ls -d /usr/lib /usr/libexec /usr/local
/usr/lib /usr/libexec /usr/local
```

Dry run (syntax check)

```
#!/bin/bash -vn
A=(/usr/l*)
ls -d ${A[@]}
```

Check syntax with `-vn`

```
$ ./test.sh
A=(/usr/l*)
ls -d ${A[@]}
```

Or use ShellCheck: https://www.shellcheck.net/ – https://github.com/koalaman/shellcheck

Line by line execution

```
$ cat test.sh
#!/bin/bash -v
trap read debug
A=(/usr/l*)
ls -d ${A[@]}
$ ./test.sh
A=(/usr/l*)
read
<ENTER>
ls -d ${A[@]}
read
<ENTER>
/usr/lib        /usr/libexec    /usr/local
```

Changing scripts "in flight"

Script	Read by	Editing it after `sbatch`
Submission script	Slurm, copied at submission	has no effect
Scripts it calls	Bash, when the job script reaches that line	changes what the job runs, unless the job already reached that line

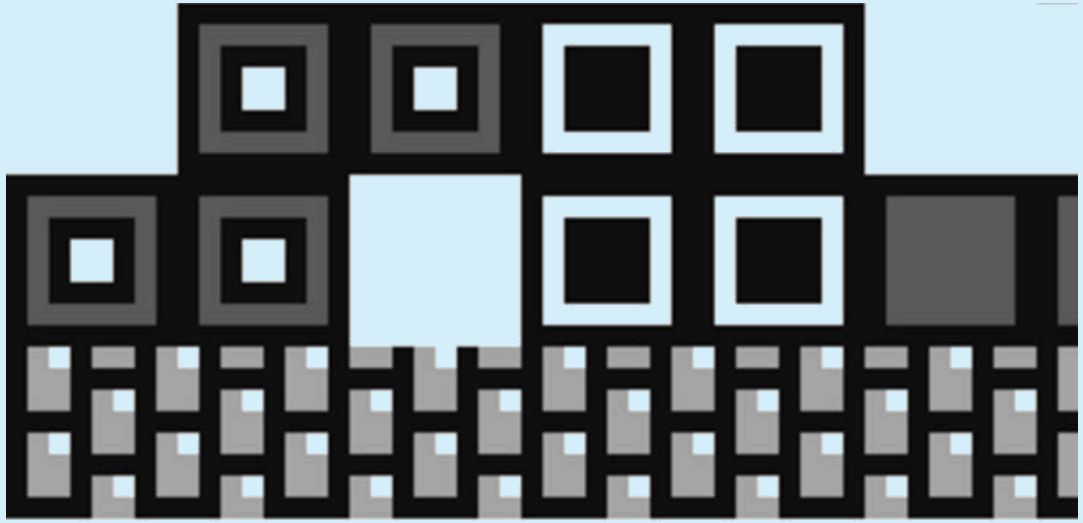
```
#!/bin/bash
#SBATCH --time=1:00:00
rsync some_data somewhere
./run_analysis.sh      # not loaded until the job reaches this line
```

Note that lines appended to a running Bash script will still run, while other edits break it.

Do not rely on this behaviour.

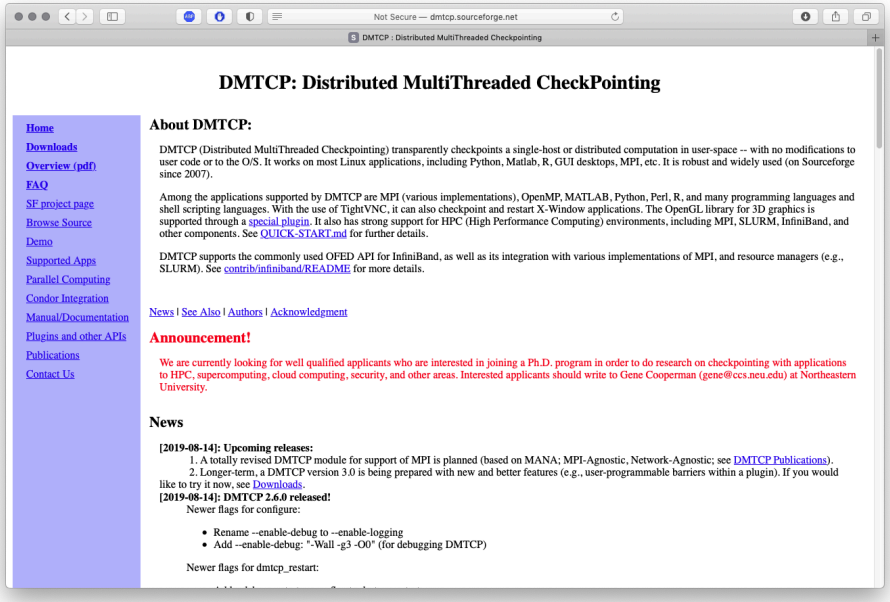
Final words...

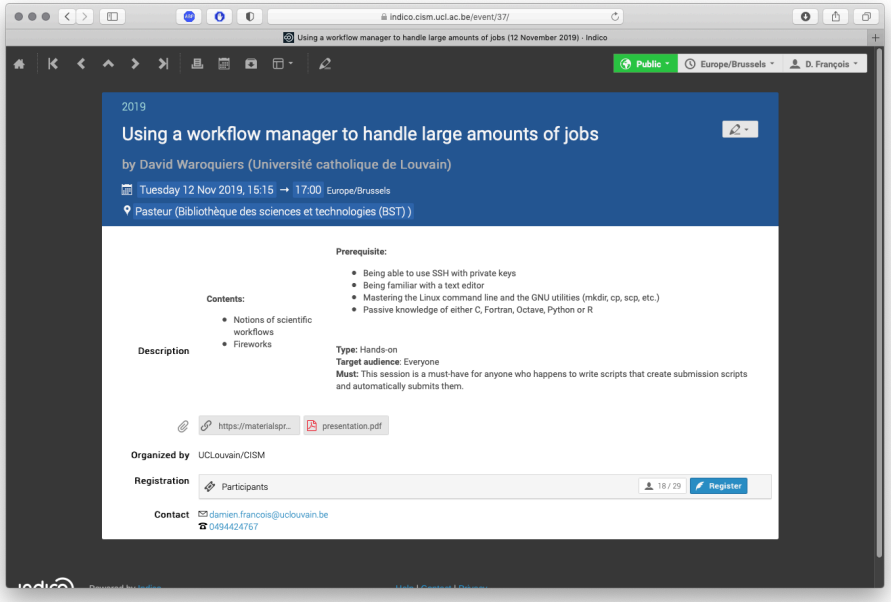
- Write and submit submission scripts
- Explore the clusters
- Read the man pages of Slurm commands
- Use the resources you request
- Beware of limits
- Build workflows
- Submit jobs!
- Look out for next sessions ...



Checkpointing

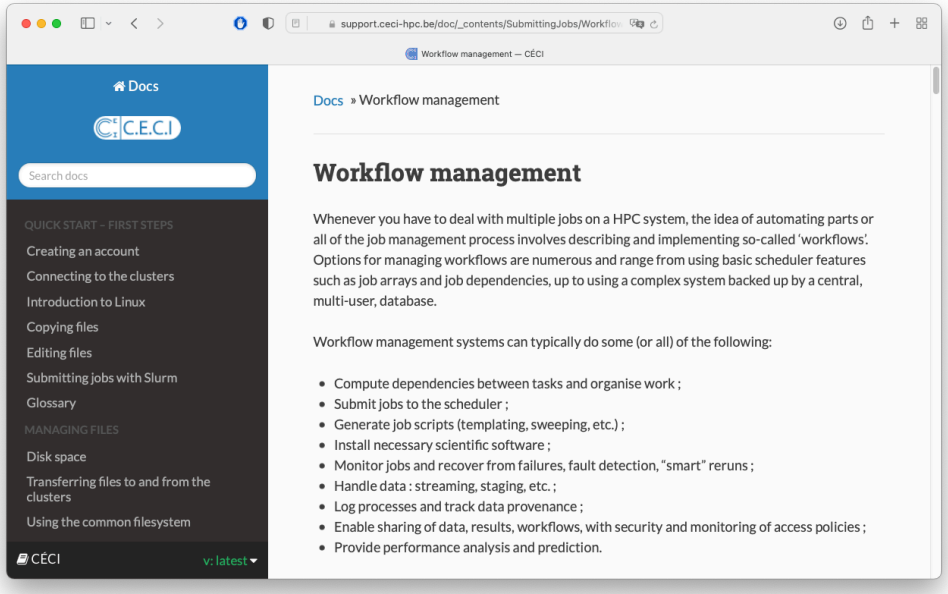
when your jobs are toooooo looooooong compared with the cluster maximum walltimes





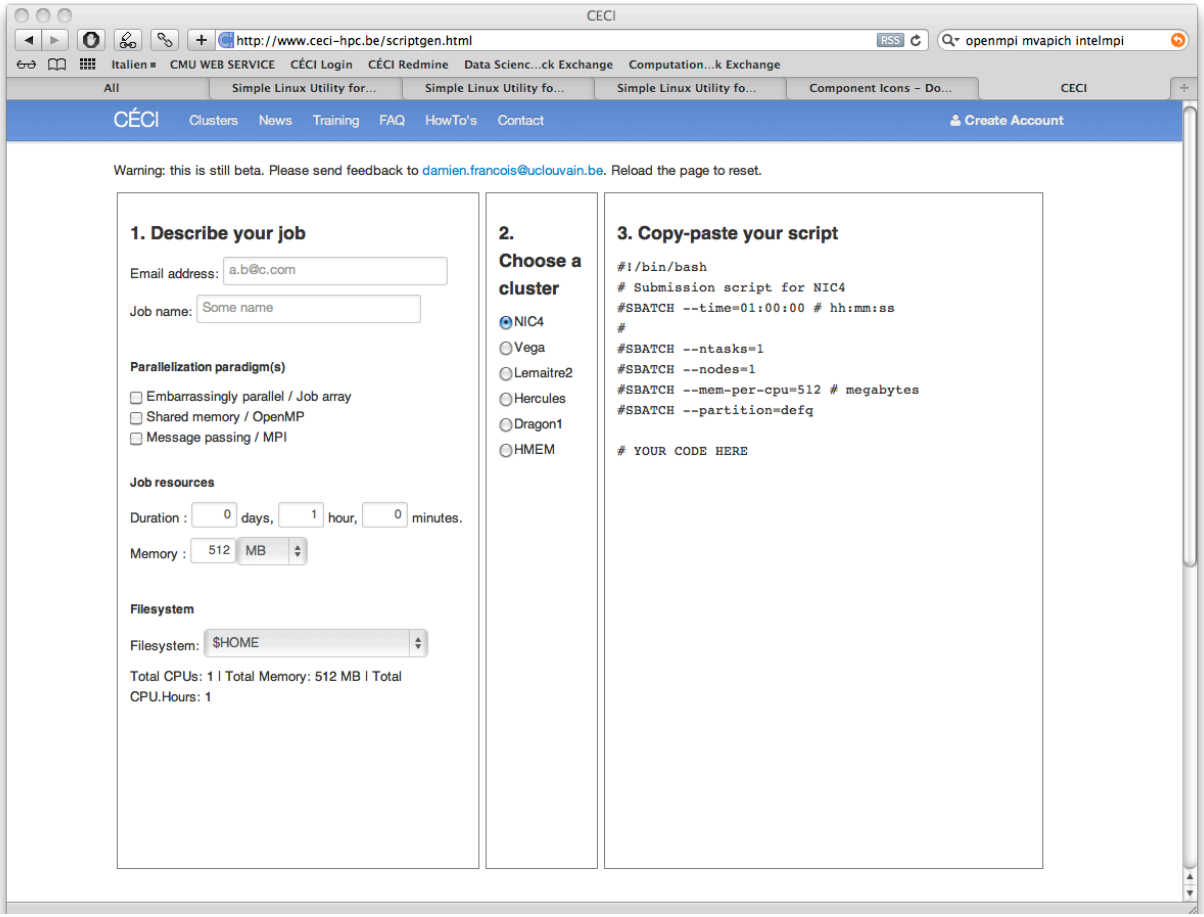
Workflow management systems

when your job dependencies and parameters are too complex to handle by hand



https://support.ceci-hpc.be/doc/_contents/SubmittingJobs/WorkflowManagement.html

CÉCI submission script generator



HANDS-ON

Final exercise



Write the submission script for your use case

- Which program will you use? What module(s) to load?
- What type of parallelism? Is the program able to use GPUs? multiple nodes? multiple cores? How many at the same time?
- What data will the job consume or produce? Where is the input data located? Where will the output data be located? How much disk does the job need?
- How much memory does the job need? For how long should the job run?
- What should the output file be named? Do you want email notifications? Do you want to refer to the job by some name rather than ID?
- Which cluster is the most appropriate? Which partition should you target? Are there specific hardware types you want to avoid? What are the limits in place?

Or submit an interactive job and connect with a tunnel.

Typical skeleton

Resources · Targets · Parameters · Environment · Data in · Compute · Data out

```
#!/bin/bash

# Requested resources
#SBATCH --ntasks=
#SBATCH --cpus-per-task=
#SBATCH --mem-per-cpu=

# Job duration
#SBATCH --time=

# GPUs?
#SBATCH --gres=gpu:

# Partition, QOS, License, Account?
#SBATCH --partition=
#SBATCH --qos=
#SBATCH --licenses=
#SBATCH --account=
```

```
# Job parameters
#SBATCH --output=
#SBATCH --mail-type=
#SBATCH --mail-user=
#SBATCH --job-name=

### Setup the environment
module load ...
export ...

### Prepare data
mkdir -p ...

### Compute
srun ...

### Cleanup results
cp -r ...
rm -r ...
```