High Performance
Computing &
Big Data Services

hpc.uni.lu

hpc@uni.lu

@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

Uni.lu HPC School 2020

PS2: Getting Started 2.0: SLURM, performance engineering and basic launchers

Uni.lu High Performance Computing (HPC) Team

H. Cartiaux

University of Luxembourg (UL), Luxembourg

http://hpc.uni.lu

Latest versions available on Github:

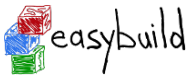
UL HPC tutorials: https://github.com/ULHPC/tutorials

UL HPC School: http://hpc.uni.lu/hpc-school/

PS2 tutorial sources: ulhpc-tutorials.rtfd.io/en/latest/beginners

Summary

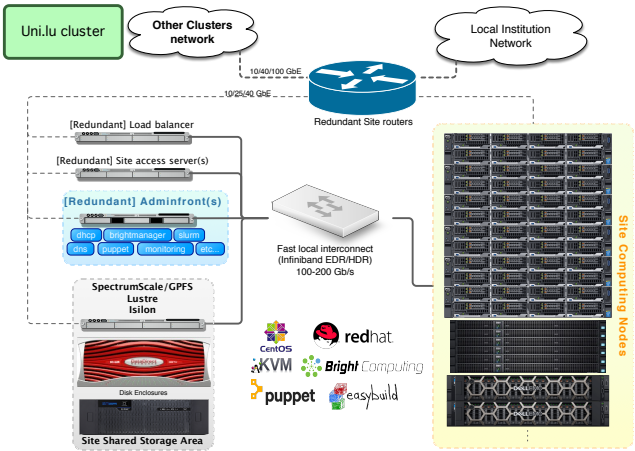
Main Objectives of this Session



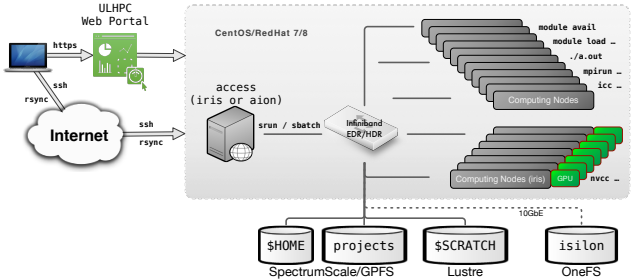
- Review ULHPC environment
- **Design and usage of SLURM**
  - ↪ recent changes in configuration
  - ↪ **common** and **advanced** SLURM tools and commands
    - ✓ `srun`, `sbatch`, `squeue` etc.
    - ✓ job specification
- Understand User Software environment
  - ↪ LMod modules, generated with Easybuild
- SLURM generic **launchers** you can use for your own jobs

Summary

UL HPC Supercomputers: General Architecture



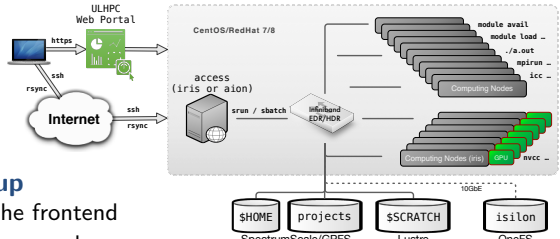
Compute Nodes / Storage Environment



- **Storage usage**: `df -ulhpc [-i]`
  - ↪ `$HOME`: regular backup policy
  - ↪ `$SCRATCH` **NO** backup & purged
    - ✓ 60 days retention policy
  - ↪ Project quotas attached to group
    - ✓ **not** *(default)* `clusterusers` group
    - ✓ Commands writing in project dir: `sg <group> -c "<command>"`
- **LMod/Environment modules**
  - ↪ **Not** on access, **only** on compute nodes

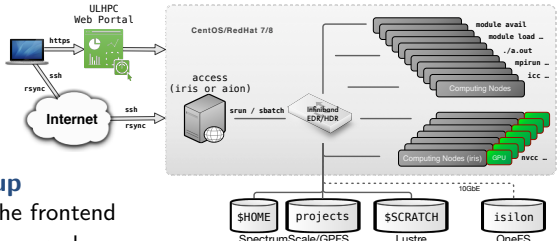
Directory	FileSystem	Max size	Max #files	Backup
`$HOME` (`iris`)	GPFS	500 GB	1.000.000	YES
`$SCRATCH`	Lustre	10 TB	1.000.000	**NO**
Project	GPFS	*per request*		PARTIALLY (`/backup` subdir)
Project	OneFS	*per request*		PARTIALLY

Typical Workflow on UL HPC resources



- **Preliminary setup**
  1. Connect to the frontend — `ssh, screen`
  2. Synchronize you code — `scp/rsync/svn/git`
  3. Reserve a few interactive resources — `srun -p interactive [...]`
     - ✓ *(eventually)* build your program — `gcc/icc/mpicc/nvcc..`
     - ✓ Test on small size problem — `srun/python/sh...`
     - ✓ Prepare a launcher script — `<launcher>.{sh|py}`

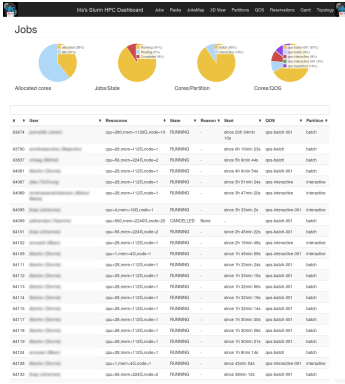
Typical Workflow on UL HPC resources

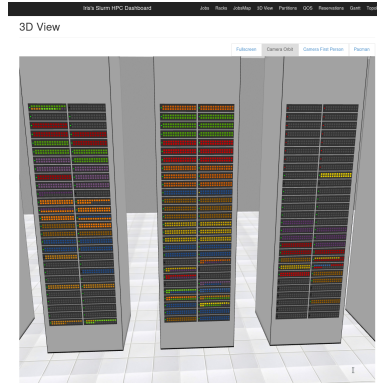


- **Preliminary setup**
  1. Connect to the frontend — `ssh, screen`
  2. Synchronize you code — `scp/rsync/svn/git`
  3. Reserve a few interactive resources — `srun -p interactive [...]`
     - ✓ *(eventually)* build your program — `gcc/icc/mpicc/nvcc..`
     - ✓ Test on small size problem — `srun/python/sh...`
     - ✓ Prepare a launcher script — `<launcher>.{sh|py}`
- **Real Experiment**
  1. Reserve passive resources — `sbatch [...] <launcher>`
  2. Grab the results — `scp/rsync/svn/git ..`

Slurm Web

(internal network only)





Ganglia

https://hpc.uni.lu/iris/ganglia/

- **WIP**: Prometheus / Graphana
  ↪ together with `aion`

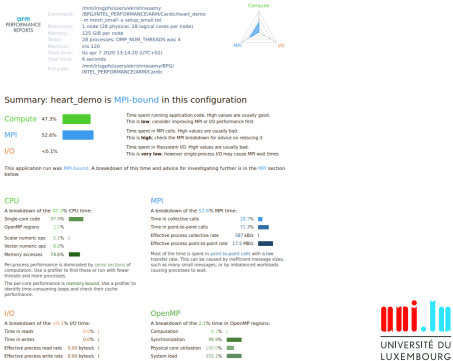
Interactive Monitoring/debugging

```
$> sq
# Connect to your job (first allocated node)
$> sjoin <JOBID>
$> htop    # view of all processes
           # F5: tree view
           # u <name>: filter by process of <name>
           # q: quit
```

Interactive Monitoring/debugging

```
$> sq
# Connect to your job (first allocated node)
$> sjoin <JOBID>
$> htop   # view of all processes
          # F5: tree view
          # u <name>: filter by process of <name>
          # q: quit
```

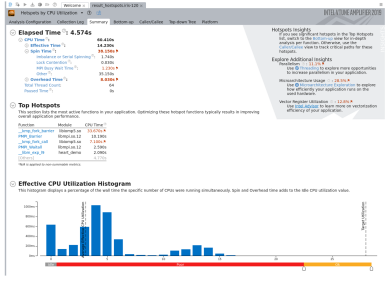
- Arm Forge suit
  - ↪ `ddt`, `map`, `perf-report`



Interactive Monitoring/debugging

```
$> sq
# Connect to your job (first allocated node)
$> sjoin <JOBID>
$> htop    # view of all processes
           # F5: tree view
           # u <name>: filter by process of <name>
           # q: quit
```

- Arm Forge suit
  - ↪ `ddt`, `map`, `perf-report`
- Intel VTune/Advisor/Inspector



Discovering and visualizing UL HPC environment

Your Turn!

Hands-on Discovering your HPC environment

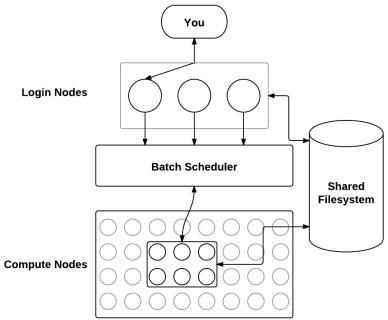
▶ url ◀ | github | src

- **Connect to the cluster frontend**
- Navigate between `$HOME`, `$SCRATCH`
- Discovering storage usage
- Live status tools (Slurm Web, Ganglia...)

Summary

Resource and Job Management Systems

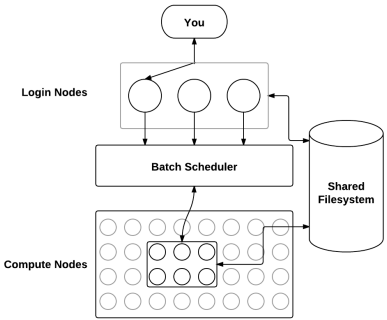
- **Resource and Job Management System (RJMS)**
  - ↪ *Glue* for a parallel computer to execute parallel jobs
  - ↪ **Goal**: satisfy users demands for computation
    - ✓ assign resources to user jobs with an efficient manner



Resource and Job Management Systems

- **Resource and Job Management System (RJMS)**
  - ↪ *Glue* for a parallel computer to execute parallel jobs
  - ↪ **Goal**: satisfy users demands for computation
    - ✓ assign resources to user jobs with an efficient manner

- **HPC Resources**:
  - ↪ Nodes (typically a unique IP address)
    - ✓ Sockets / Cores / Hyperthreads
    - ✓ Memory
    - ✓ Interconnect/switch resources
  - ↪ Generic resources (e.g. GPUs)
  - ↪ Licenses
- **Strategic Position**
  - ↪ Direct/constant knowledge of resources
  - ↪ Launch and otherwise manage jobs

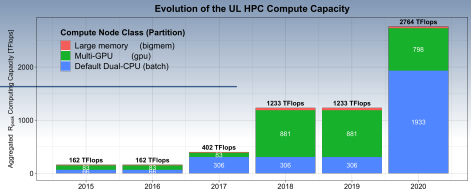


Slurm on ULHPC clusters

- ULHPC uses Slurm for cluster/resource management and job scheduling
  - ↪ Simple Linux Utility for Resource Management `https://slurm.schedmd.com/`
  - ↪ Handles submission, scheduling, execution, and monitoring of **jobs**
  - ↪ official documentation, tutorial, FAQ

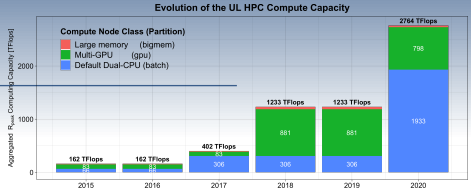
- **User jobs** have the following key characteristics:
  - ↪ set of requested resources:
    - ✓ number of computing resources: **nodes** (including all their CPUs and cores) or **CPUs** (including all their cores) or **cores**
    - ✓ amount of **memory**: either per node or per CPU
    - ✓ **(wall)time** needed for the users tasks to complete their work
  - ↪ a requested node **partition** (job queue)
  - ↪ a requested **quality of service** (QoS) level which grants users specific accesses
  - ↪ a requested **account** for accounting purposes

Slurm on ULHPC clusters



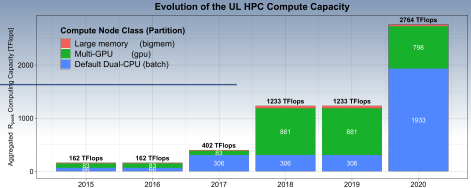
- Predefined **Queues/Partitions** depending on node type
  - ↪ `batch` (Default Dual-CPU nodes) — **Max**: 64 nodes, 2 days walltime
  - ↪ `gpu` (GPU nodes nodes) — **Max**: 4 nodes, 2 days walltime
  - ↪ `bigmem` (Large-Memory nodes) — **Max**: 1 node, 2 days walltime
  - ↪ In addition: `interactive` (for quicks tests) — **Max**: 2 nodes, 2h walltime
    - ✓ for code development, testing, and debugging

Slurm on ULHPC clusters



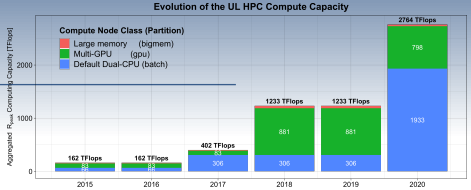
- Predefined **Queues/Partitions** depending on node type
  - ↪ `batch` (Default Dual-CPU nodes) — **Max**: 64 nodes, 2 days walltime
  - ↪ `gpu` (GPU nodes nodes) — **Max**: 4 nodes, 2 days walltime
  - ↪ `bigmem` (Large-Memory nodes) — **Max**: 1 node, 2 days walltime
  - ↪ In addition: `interactive` (for quicks tests) — **Max**: 2 nodes, 2h walltime
    - ✓ for code development, testing, and debugging
- Queue Policy: **cross-partition QOS**, mainly tied to **priority level** (`low` → `urgent`)
  - ↪ `long` QOS with extended Max walltime (`MaxWall`) set to **14 days**
  - ↪ special **preemptible QOS** for best-effort jobs: `besteffort`.

Slurm on ULHPC clusters



- Predefined **Queues/Partitions** depending on node type
  - ↪ `batch` (Default Dual-CPU nodes) — **Max**: 64 nodes, 2 days walltime
  - ↪ `gpu` (GPU nodes nodes) — **Max**: 4 nodes, 2 days walltime
  - ↪ `bigmem` (Large-Memory nodes) — **Max**: 1 node, 2 days walltime
  - ↪ In addition: `interactive` (for quicks tests) — **Max**: 2 nodes, 2h walltime
    - ✓ for code development, testing, and debugging
- Queue Policy: **cross-partition QOS**, mainly tied to **priority level** (`low` → `urgent`)
  - ↪ `long` QOS with extended Max walltime (`MaxWall`) set to **14 days**
  - ↪ special **preemptible QOS** for best-effort jobs: `besteffort`.
- Accounts associated to supervisor (multiple associations possible)
  - ↪ Proper group/user accounting

Slurm on ULHPC clusters



- Predefined **Queues/Partitions** depending on node type
  - ↪ `batch` (Default Dual-CPU nodes) — **Max**: 64 nodes, 2 days walltime
  - ↪ `gpu` (GPU nodes nodes) — **Max**: 4 nodes, 2 days walltime
  - ↪ `bigmem` (Large-Memory nodes) — **Max**: 1 node, 2 days walltime
  - ↪ In addition: `interactive` (for quicks tests) — **Max**: 2 nodes, 2h walltime
    - ✓ for code development, testing, and debugging
- Queue Policy: **cross-partition QOS**, mainly tied to **priority level** (`low` → `urgent`)
  - ↪ `long` QOS with extended Max walltime (`MaxWall`) set to **14 days**
  - ↪ special **preemptible QOS** for best-effort jobs: `besteffort`.
- Accounts associated to supervisor (multiple associations possible)
  - ↪ Proper group/user accounting
- Slurm Federation configuration between `iris` and `aion`
  - ↪ ensures global policy (coherent job ID, global scheduling, etc.) within ULHPC systems
  - ↪ easily submit jobs from one cluster to another `-M, --cluster aion|iris`

Main Slurm Commands: Submit Jobs

```
$> sbatch -p <partition> [--qos <qos>] [-A <account>] [...]  <path/to/launcher.sh>
```

Submitting Jobs

- `sbatch`: Submit batch **launcher script** for later execution **batch/passive mode**
  - ↪ allocate resources (nodes, tasks, partition, etc.)
  - ↪ runs a single **copy** of the batch script on the **first** allocated node

Main Slurm Commands: Submit Jobs

```
$> srun -p <partition> [--qos <qos>] [-A <account>] [...]  --pty bash
```

Submitting Jobs

- `sbatch`: Submit batch **launcher script** for later execution **batch/passive mode**
  - ↪ allocate resources (nodes, tasks, partition, etc.)
  - ↪ runs a single **copy** of the batch script on the **first** allocated node
- `srun`: initiate parallel **job steps within a job OR start an interactive job**
  - ↪ allocate resources (number of nodes, tasks, partition, constraints, etc.)
  - ↪ launch a job that will execute on them.

Main Slurm Commands: Submit Jobs

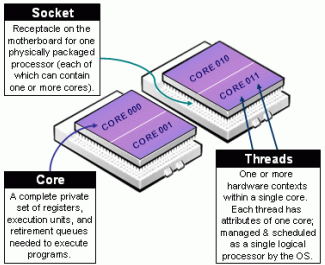
```
$> salloc -p <partition> [--qos <qos>] [-A <account>] [...]  <command>
```

Submitting Jobs

- `sbatch`: Submit batch **launcher script** for later execution **batch/passive mode**
  - ↪ allocate resources (nodes, tasks, partition, etc.)
  - ↪ runs a single **copy** of the batch script on the **first** allocated node
- `srun`: initiate parallel **job steps within a job OR start an interactive job**
  - ↪ allocate resources (number of nodes, tasks, partition, constraints, etc.)
  - ↪ launch a job that will execute on them.
- `salloc`: : request interactive jobs/allocations
  - ↪ allocate resources (nodes, tasks, partition, etc.), either run a command or start a shell.

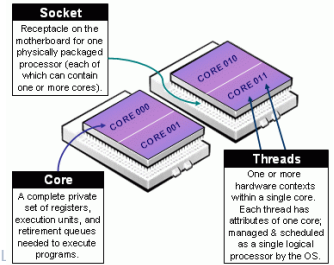
Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** `--ntasks-per-node <n>`



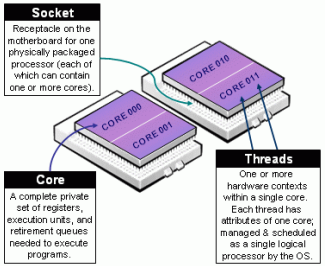
Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node — `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** — `--ntasks-per-node <n>`
  - ↪ Slurm Socket = Physical Socket/**CPU**/Processor — `--ntasks-per-socket <n>`



Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** `--ntasks-per-node <n>`
  - ↪ Slurm Socket = Physical Socket/**CPU**/Processor `--ntasks-per-socket <n>`
  - ↪ **Slurm CPU** = Physical **Core** `-c <#threads>`
    - ✓ Hyper-Threading (HT) Technology is **disabled** on all the compute nodes
    - ✓ #cores = #threads `-c <N>` ⟶ `OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}`
    - ✓ Total number of tasks `${SLURM_NTASKS}` ⟶ `srun -n ${SLURM_NTASKS} [...]`



Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** `--ntasks-per-node <n>`
  - ↪ Slurm Socket = Physical Socket/**CPU**/Processor `--ntasks-per-socket <n>`
  - ↪ **Slurm CPU** = Physical **Core** `-c <#threads>`
    - ✓ Hyper-Threading (HT) Technology is **disabled** on all the compute nodes
    - ✓ #cores = #threads `-c <N>` ⟶ `OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}`
    - ✓ Total number of tasks: `${SLURM_NTASKS}` ⟶ `srun -n ${SLURM_NTASKS} [...]`

- **Important: Always try to align resource specs with physical characteristics**
  - ↪ **Ex: 64 cores per socket and 2 sockets (physical CPUs) per `aion` node**
  - ↪ `[-N <N>] --ntasks-per-node <2n> --ntasks-per-socket <n> -c <thread>`
    - ✓ **Total**: `<N>`×2×`<n>` tasks, each on `<thread>` threads
    - ✓ **Ensure** `<n>`×`<thread>`=64 (#cores) **in this case** (target 14 on iris)
    - ✓ Ex: `-N 2 --ntasks-per-node 32 --ntasks-per-socket 16 -c 4` (**Total**: 64 tasks)

Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** `--ntasks-per-node <n>`
  - ↪ Slurm Socket = Physical Socket/**CPU**/Processor `--ntasks-per-socket <n>`
  - ↪ **Slurm CPU** = Physical **Core** `-c <#threads>`
    - ✓ Hyper-Threading (HT) Technology is **disabled** on all the compute nodes
    - ✓ #cores = #threads `-c <N>` ⟶ `OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}`
    - ✓ Total number of tasks: `${SLURM_NTASKS}` ⟶ `srun -n ${SLURM_NTASKS} [...]`

Hostname	Node type	#Nodes	#Socket	#Cores	RAM	Features
aion-[0001-0318]	Regular	318	2	128	256 GB	batch,epyc
iris-[001-108]	Regular	108	2	28	128 GB	batch,broadwell
iris-[109-168]	Regular	60	2	28	128 GB	batch,skylake
iris-[169-186]	Multi-GPU	18	2	28	768 GB	gpu,skylake,volta
iris-[191-196]	Multi-GPU	6	2	28	768 GB	gpu,skylake,volta32
iris-[187-190]	Large Memory	4	4	112	3072 GB	bigmem,skylake

- List available features: `sfeatures`

```
# OR: sinfo -o '%20N %.6D %.6c %15F %12P %f'
```

Main Slurm Commands: Submit Jobs options

```
$> {sbatch | srun | salloc} [...]
```

Command-line option	Description	Example
-N <N>	<N> **Nodes** request	-N 2
--ntasks-per-socket=<n>	<n> Tasks-per-socket request	--ntasks-per-socket=14
--ntasks-per-node=<n>	<n> Tasks-per-node request	--ntasks-per-socket=28
-c=<c>	<c> Cores-per-task request (multithreading)	-c 1
--mem=<m>GB	<m>**GB memory per node** request	--mem 0
-t [DD-]HH[:MM:SS]>	**Walltime** request	-t 4:00:00
-G <gpu>	<gpu> GPU(s) request	-G 4
-C <feature>	Feature request (Ex: broadwell,skylake,...)	-C skylake
-p <partition>	Specify job partition/queue	
--qos <qos>	Specify job qos	
-A <account>	Specify account	
-J <name>	Job name	-J MyApp
-d <specification>	Job dependency	-d singleton
--mail-user=<email>	Specify email address	
--mail-type=<type>	Notify user by email when certain event types occur.	--mail-type=END,FAIL

Main Slurm Commands: Collect Information

- Partition (queue) and node status
  ↪ eventually filter on specific job state (**R**:running /**PD**:pending / **F**:failed / **PR**:preempted)

```
$> squeue [-u <user>] [-p <partition>] [--qos <qos>] [-t R|PD|F|PR]
```

Main Slurm Commands: Collect Information

- Partition (queue) and node status
  ↪ eventually filter on specific job state (**R**:running /**PD**:pending / **F**:failed / **PR**:preempted)

```
$> squeue [-u <user>] [-p <partition>] [--qos <qos>] [-t R|PD|F|PR]
```

- Show partition status, summarized status (`-s`), problematic nodes (`-R`), reservations (`-T`)

```
$> sinfo [-p <partition>] {-s | -R | -T |...}
```

Main Slurm Commands: Collect Information

- Partition (queue) and node status
  ↪ eventually filter on specific job state (**R**:running /**PD**:pending / **F**:failed / **PR**:preempted)

```
$> squeue [-u <user>] [-p <partition>] [--qos <qos>] [-t R|PD|F|PR]
```

- Show partition status, summarized status (`-s`), problematic nodes (`-R`), reservations (`-T`)

```
$> sinfo [-p <partition>] {-s | -R | -T |...}
```

- View job, partition, nodes, reservation status

```
$> scontrol show { job <jobid> | partition [<part>] | nodes <node>| reservation...}
```

Main Slurm Commands: Collect Information

Command	Description
`sinfo`	Report system status (nodes, partitions etc.)
`squeue [-u $(whoami)]`	display jobs[steps] and their state
`seff <jobid>`	get efficiency metrics of past job
`scancel <jobid>`	cancel a job or set of jobs.
`scontrol show [...]`	view and/or update system, nodes, job, step, partition or reservation status
`sstat`	show status of running jobs.
`sacct [-X] -j <jobid> [...]`	display accounting information on jobs.
`sprio`	show factors that comprise a jobs scheduling priority
`smap`	graphically show information on jobs, nodes, partitions

```
### Get statistics on past job
slist <jobid>
# sacct [-X] -j <jobid> --format User,JobID,Jobname%30,partition,state,time,elapsed,MaxRss,\
#                                MaxVMSize,nnodes,ncpus,nodelist,AveCPU,ConsumedEnergyRaw
# seff <jobid>
```

ULHPC Slurm Partitions 2.0

`-p, -partition=<partition>`

```
$> {srun|sbatch|salloc|sinfo|squeue...} -p <partition> [...]
```

AION Partition	Type	#Node	PriorityTier	DefaultTime	MaxTime	MaxNodes
`interactive`	floating	318	100	30min	2h	2
`batch`		318	1	2h	48h	64

IRIS Partition	Type	#Node	PriorityTier	DefaultTime	MaxTime	MaxNodes
`interactive`	floating	196	100	30min	2h	2
`batch`		168	1	2h	48h	64
`gpu`		24	1	2h	48h	4
`bigmem`		4	1	2h	48h	1

ULHPC Slurm QOS 2.0

`--qos=<qos>`

```
$> {srun|sbatch|salloc|sinfo|squeue...} [-p <partition>] --qos <qos> [...]
```

QOS	Partition	Allowed [L1] Account	Prio	GrpTRES	MaxTresPJ	MaxJobPU	Flags
besteffort	*	**ALL**	1			100	NoReserve
low	*	**ALL** (default for CRP/externals)	10			2	DenyOnLimit
normal	*	**Default** (UL,Projects,...)	100			50	DenyOnLimit
long	*	UL,Projects,etc.	100	node=6	node=2	1	DenyOnLimit,PartitionTimeLimit
debug	interactive	**ALL**	150	node=8		2	DenyOnLimit
high	*	(**restricted**) UL,Projects,Industry	200			10	DenyOnLimit
urgent	*	(**restricted**) UL,Projects,Industry	1000			100 ?	DenyOnLimit

- **Cross-partition QOS**, mainly tied to **priority level** (low → urgent)
  - ↪ Simpler names than before (i.e. no more qos- prefix)
  - ↪ special **preemptible QOS** for best-effort jobs: `besteffort`

Usage Example: Interactive Jobs

`si,si-[gpu|bigmem]`

```
# Simple interactive job - match feature name with target partition
srun -p interactive --qos debug -C {batch,gpu,bigmem} [...] --pty bash
# OR, **better**
si[-gpu|-bigmem] [...]
```

- *Floating* `interactive` partition allow to access all nodes
  - ↪ no more dedicated resources: optimize the cluster for passive jobs
  - ↪ no guarantee if partition is full **YET** backfilling and priority ensure first served

Node Type	Slurm command	Helper script
regular	`srun -p interactive --qos debug -C batch [-C {broadwell,skylake}] [...] --pty bash`	`si [...]`
gpu	`srun -p interactive --qos debug -C gpu [-C volta[32]] -G 1 [...] --pty bash`	`si-gpu [...]`
bigmem	`srun -p interactive --qos debug -C bigmem [...] --pty bash`	`si-bigmem [...]`

Usage Example: Interactive Jobs

si,si-[gpu|bigmem]

```
# Simple interactive job - match feature name with target partition
srun -p interactive --qos debug -C {batch,gpu,bigmem} [...] --pty bash
# OR, **better**
si[-gpu|-bigmem] [...]
```

```
# Simple interactive job on batch partition
(access)$> si
# srun -p interactive --qos debug -C batch  --mem-per-cpu 4096 --pty bash
(node)$> echo $SLURM_NTASKS
1
(node)$> echo $SLURM_JOBID
2166371
$> sq
# squeue -u <login>
```

Usage Example: Regular Jobs

`sbatch [...]`

- Restricted to **Max 2 days** / Max 64 nodes — default QOS induced by the `job_submit.lua` plugin
  - ↪ enforce precision of project/training account (`-A <account>`) **when appropriate** *i.e.* for
    - ✓ project name `<project>`
    - ✓ lecture/course name: `<lecture>`
  - ↪ **do NOT reserve `gpu` nodes (`-p gpu`) WITHOUT `-G [1-4]`**
    - ✓ Iris compute nodes in the gpu partition have 4xVolta V100 GPUs and 768GB RAM

Node Type	Slurm command
regular	`sbatch [-A <project>] -p batch [--qos {high,urgent}] [-C {broadwell,skylake}] [...]`
gpu	`sbatch [-A <project>] -p gpu -G <N> [--qos {high,urgent}] [-C volta[32]] [...]`
bigmem	`sbatch [-A <project>] -p bigmem [--qos {high,urgent}] [...]`

Usage Example: Long Jobs

`--qos long`

```
# Select target partition to bypass default walltime restrictions
sbatch -p {batch | gpu | bigmem} --qos long [...]
```

- Extended Max walltime (`MaxWall`) set to **14 days** — EuroHPC/PRACE Recommendations
  - ↪ **yet** restricted usage: Max 6 nodes, Max 2 nodes per Job, Max **4** Jobs per User

Node Type	Slurm command
regular	`sbatch [-A <project>] -p batch --qos long [-C {broadwell,skylake}] [...]`
gpu	`sbatch [-A <project>] -p gpu --qos long [-C volta[32]] -G 1 [...]`
bigmem	`sbatch [-A <project>] -p bigmem --qos long [...]`

Usage Example: Best-effort Jobs

`--qos besteffort`

- Special **preemptible QOS** for **best-effort jobs**
  - ↪ *lowest* priority
  - ↪ **deleted if another non-besteffort job wants resources** where they are running.
  - ↪ useful to maximize use of the cluster for short-term jobs
    - ✓ several limits inherent to *regular* QOS are **bypassed** in besteffort
  - ↪ many scientific applications support internal state saving and restart **Checkpoint-restart**
    - ✓ System-level CR possible with DMTCP: Distributed MultiThreaded CheckPointing
    - ✓ see the official DMTCP launchers

```
sbatch -p {batch | gpu | bigmem} --qos besteffort [...]
```

Live Job Statistics

```
$> scontrol show job 2166371
JobId=2166371 JobName=bash
   UserId=<login>(<uid>) GroupId=clusterusers(666) MCS_label=N/A
   Priority=12741 Nice=0 Account=ulhpc QOS=debug JobState=RUNNING Reason=None
   [...]
   SubmitTime=2020-12-07T22:08:25 EligibleTime=2020-12-07T22:08:25
   StartTime=2020-12-07T22:08:25 EndTime=2020-12-07T22:38:25
   [...]
   WorkDir=/mnt/irisgpfs/users/<login>
```

Node/Job Statistics

```
$> sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
interactive    up    4:00:00    196   idle iris-[001-196]
batch*         up 2-00:00:00      5    mix [...]
batch*         up 2-00:00:00    127  alloc [...]
batch*         up 2-00:00:00     36   idle [...]
gpu            up 2-00:00:00      4   resv iris-[186,191-193]
gpu            up 2-00:00:00     19  alloc [...]
gpu            up 2-00:00:00      1   idle iris-185
bigmem         up 2-00:00:00      4    mix iris-[187-190]
```

Node/Job Statistics

```
$> sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
interactive    up    4:00:00    196   idle iris-[001-196]
batch*         up 2-00:00:00      5    mix [...]
batch*         up 2-00:00:00    127  alloc [...]
batch*         up 2-00:00:00     36   idle [...]
gpu            up 2-00:00:00      4   resv iris-[186,191-193]
gpu            up 2-00:00:00     19  alloc [...]
gpu            up 2-00:00:00      1   idle iris-185
bigmem         up 2-00:00:00      4    mix iris-[187-190]
```

```
$> slist <JOBID>
# sacct -j <JOBID> --format User,JobID,Jobname%30,partition,state,time,elapsed,\
#              MaxRss,MaxVMSize,nnodes,ncpus,nodelist,AveCPU,ConsumedEnergyRaw
# seff <JOBID>
```

Basic Slurm Launcher Examples

Prefer `--ntasks-per-node` over `-n` !

```
#!/bin/bash -l                # <--- DO NOT FORGET '-l'
### Request a single task using one core on one node for 5 minutes in the batch queue
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 1
#SBATCH --time=0-00:05:00
#SBATCH -p batch

print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
# Safeguard for NOT running this launcher on access/login nodes
module purge || print_error_and_exit "No 'module' command"
# List modules required for execution of the task
module load <...>
# [...]
```

Basic Slurm Launcher Examples

Prefer `--ntasks-per-node` over `-n` !

```
#!/bin/bash -l
### Request 6 single core tasks equally spread across two nodes for 3 hours
#SBATCH -N 2
#SBATCH --ntasks-per-node=3
#SBATCH -c 1
#SBATCH --time=0-03:00:00
#SBATCH -p batch
print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
module load <...>
# [...]
```

```
# Total: 6 tasks within 2 nodes
sbatch <launcher>
```

```
# Total: 9 tasks within 3 nodes
sbatch -N 3 <launcher>
```

Basic Slurm Launcher Examples

Prefer `--ntasks-per-node` over `-n` !

```
#!/bin/bash -l
### Request as many tasks as cores available on a single node for 3 hours
#SBATCH -N 1
#SBATCH --ntasks-per-node=28  # On iris; for aion, use --ntasks-per-node=128
#SBATCH -c 1
#SBATCH --time=0-03:00:00
#SBATCH -p batch
print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
module load <...>
# [...]
```

```
# Total (iris): 28 tasks within 1 node
sbatch <launcher>
```

```
# Total (iris): 56 tasks within 2 nodes
sbatch -N 2 <launcher>
```

Basic Slurm Launcher Examples (Large memory)

```
#!/bin/bash -l
### Request one sequential task requiring half the memory of a regular iris node for 1 day
#SBATCH -J MyLargeMemorySequentialJob       # Job name
#SBATCH --mail-user=Your.Email@Address.lu   # mail me ...
#SBATCH --mail-type=end,fail                # ... upon end or failure
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 1
#SBATCH --mem=64GB          # if above 112GB: consider bigmem partition (USE WITH CAUTION)
#SBATCH --time=1-00:00:00
#SBATCH -p batch            # if above 112GB: consider bigmem partition (USE WITH CAUTION)
print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
module load <...>
# [...]
```

Basic Slurm Launcher Examples (GPU)

```
#!/bin/bash -l
### Request one GPU tasks for 4 hours - dedicate 1/4 of available cores for its management
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 7
#SBATCH -G 1
#SBATCH --time=00:04:00
#SBATCH -p gpu
print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
# You probably want to use more recent gpu/CUDA-optimized software builds
module use /opt/apps/resif/iris/2019b/gpu/modules
module load <...>
# This should report a single GPU (over 4 available per gpu node)
nvidia-smi
# [...]
```

Slurm [Generic] Launchers 2.0

```
#!/bin/bash -l              # <--- DO NOT FORGET '-l'
###SBATCH --job-name=<name>
###SBATCH --dependency singleton
###SBATCH -A <account>
#SBATCH --time=0-01:00:00  # 1 hour
#SBATCH --partition=batch  # If gpu: set '-G <gpus>'
#SBATCH -N 1               # Number of nodes
#SBATCH --ntasks-per-node=2
#SBATCH -c 1               # multithreading per task
#SBATCH -o %x-%j.out       # <jobname>-<jobid>.out
print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
# Load ULHPC modules
[ -f  /etc/profile ] && source /etc/profile
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}
module purge || print_error_and_exit "No 'module' command"
module load <...>
srun [-n $SLURM_NTASKS] [...]
```

- **Best-Practices**
  - ↪ use `/bin/bash -l` on top
  - ↪ set **reasonable time limits**
  - ↪ set (short) job name
  - ↪ specify account
  - ↪ `--exclusive` allocation?
  - ↪ **Avoid** Job arrays
  - ↪ consider singleton pipelining
    - ✓ job dep. made easy
  - ↪ GPU jobs (gpu partition)
    - ✓ Set #GPUs with `-G <n>`
  - ↪ Use `$SCRATCH` for large/temporary storage
  - ↪ consider night jobs
    - ✓ `--begin=20:00`

Summary

Software/Modules Management

https://hpc.uni.lu/users/software/

- Based on Environment Modules / LMod
  - ↪ convenient way to dynamically change the users environment $PATH
  - ↪ permits to easily load software through `module` command
- Currently on UL HPC: > **230 software packages**, in *multiple* versions, within **18 categ.**
  - ↪ reworked software set now deployed everywhere
    - ✓ RESIF v3.0, allowing [real] semantic versioning of released (arch-based) builds
  - ↪ hierarchical organization **Ex**: `toolchain/{foss,intel}`

```
$> module avail                                    # List available modules
```

```
$> module spider <pattern>          # Search for <pattern> within available modules
```

```
$> module load <category>/<software>[/<version>]
```

Software/Modules Management

- Key `module` variable: `$MODULEPATH` / where to look for modules.
  - ↪ **default** `iris`: `/opt/apps/resif/iris/<version>/{broadwell,skylake,gpu}/modules/all`
  - ↪ **default** `aion`: `/opt/apps/resif/aion/<version>/{epyc}/modules/all`
    - ✓ altered/prefix new path with `module use <path>`. **Ex** (to use **local** modules):

```
export EASYBUILD_PREFIX=$HOME/.local/easybuild
export LOCAL_MODULES=$EASYBUILD_PREFIX/modules/all
module use $LOCAL_MODULES
```

Software/Modules Management

- Key `module` variable: `$MODULEPATH` / where to look for modules.
  - ↪ **default** `iris`: `/opt/apps/resif/iris/<version>/{broadwell,skylake,gpu}/modules/all`
  - ↪ **default** `aion`: `/opt/apps/resif/aion/<version>/{epyc}/modules/all`
    - ✓ altered/prefix new path with `module use <path>`. **Ex** (to use **local** modules):

```
export EASYBUILD_PREFIX=$HOME/.local/easybuild
export LOCAL_MODULES=$EASYBUILD_PREFIX/modules/all
module use $LOCAL_MODULES
```

Command	Description
`module avail`	Lists all the modules which are available to be loaded
`module spider <pattern>`	Search for among available modules **(Lmod only)**
`module load <mod1> [mod2...]`	Load a module
`module unload <module>`	Unload a module
`module list`	List loaded modules
`module purge`	Unload all modules (purge)
`module use <path>`	Prepend the directory to the MODULEPATH environment variable
`module unuse <path>`	Remove the directory from the MODULEPATH environment variable

ULHPC Toolchains and Software Set Versioning

- **Yearly** release based on Easybuid release of toolchains
  - ↪ see Component versions (**fixed per release**) in the foss and intel toolchains
    - ✓ count 6 months of validation/import *after* EB release before ULHPC release

Name	**Type**	2019[a] (prod/old)	**2019b** (devel)	**2020a** (next)
`GCCCore`	compiler	8.2.0	8.3.0	9.3.0
`foss`	toolchain	2019a	2019b	2020a
`intel`	toolchain	2019a	2019b	2020a
Python		3.7.2 (and 2.7.15)	3.7.4 (and 2.7.16)	3.8.2

```
# (new) 2019b software set - iris cluster
unset MODULEPATH
module use /opt/apps/resif/iris/2019b/broadwell/modules/all
# OR (when appropriate) skylake/GPU-specialized builds **ONLY** on skylake/GPU nodes
module use /opt/apps/resif/iris/2019b/skylake/modules/all
module use /opt/apps/resif/iris/2019b/gpu/modules/all
```

ULHPC Toolchains and Software Set Versioning

- **Yearly** release based on Easybuid release of toolchains
  - ↪ see Component versions (**fixed per release**) in the foss and intel toolchains
    - ✓ count 6 months of validation/import *after* EB release before ULHPC release

Name	**Type**	2019[a] (prod/old)	**2019b** (devel)	**2020a** (next)
GCCCore	compiler	8.2.0	8.3.0	9.3.0
foss	toolchain	2019a	2019b	2020a
intel	toolchain	2019a	2019b	2020a
Python		3.7.2 (and 2.7.15)	3.7.4 (and 2.7.16)	3.8.2

```
# INCOMING in 2021: (new) 2020a software set - aion cluster
module use /opt/apps/resif/aion/2020a/epyc/modules/all
# iris cluster
module use /opt/apps/resif/iris/2020a/{broadwell,skylake,gpu}/modules/all
```

Interaction with Slurm and the software environment

Your Turn!

Hands-on Slurm

▶ url ◀ | github | src

- Run interactive job `si`
- Run passive jobs `sbatch`
- Write basics launcher scripts

Hands-on Slurm

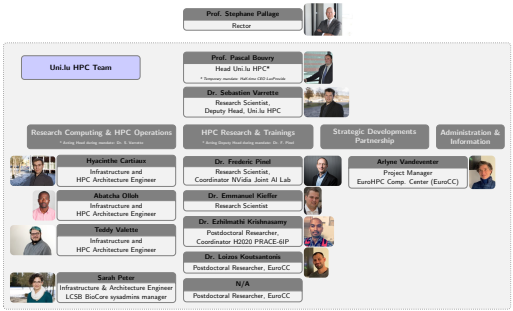
▶ url ◀ | github | src

- play with `module` command
- Use the `MODULEPATH` variable in order to change the software set
- Compile and run examples in at least two different languages

Thank you for your attention...

Questions?

High Performance Computing @ Uni.lu



University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu

https://hpc.uni.lu