UL HPC School 2017

PS5: Advanced Scheduling with SLURM and OAR on UL HPC clusters

UL High Performance Computing (HPC) Team

V. Plugaru

University of Luxembourg (UL), Luxembourg

http://hpc.uni.lu

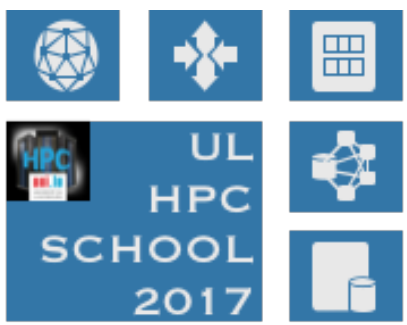
Latest versions available on Github:

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

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

PS5 tutorial sources:

https://github.com/ULHPC/tutorials/tree/devel/advanced/advanced_scheduling



Summary

Main Objectives of this Session

- **Design and usage of SLURM**
  - ↪ cluster workload manager of the UL HPC `iris` cluster
  - ↪ . . . and future HPC systems

The tutorial will show you:

- the way SLURM was **configured**, **accounting** and **permissions**
- **common** and **advanced** SLURM tools and commands
  - ↪ `srun`, `sbatch`, `squeue` etc.
  - ↪ job specification
  - ↪ SLURM job types
  - ↪ comparison of SLURM (`iris`) and OAR (`gaia` & `chaos`)
- SLURM generic **launchers** you can use for your own jobs

Documentation & comparison to OAR

`https://hpc.uni.lu/users/docs/scheduler.html`

Summary

SLURM - core concepts

- SLURM manages user jobs with 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 (logical) CPU
    - ✓ **(wall)time** needed for the user's 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
- **Example**: run an interactive job Alias: `si [...]`

```
(access)$ srun -p interactive --qos qos-interactive --pty bash
(node)$ echo $SLURM_JOBID
2058
```

Simple interactive job running under SLURM

SLURM - job example (I)

```
$ scontrol show job 2058
JobId=2058 JobName=bash
   UserId=vplugaru(5143) GroupId=clusterusers(666) MCS_label=N/A
    Priority =100 Nice=0 Account=ulhpc QOS=qos-interactive
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
   RunTime=00:00:08 TimeLimit=00:05:00 TimeMin=N/A
   SubmitTime=2017-06-09T16:49:42 EligibleTime=2017-06-09T16:49:42
   StartTime=2017-06-09T16:49:42 EndTime=2017-06-09T16:54:42 Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
    Partition =interactive  AllocNode:Sid=access2:163067
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=iris-081
   BatchHost=iris-081
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,mem=4G,node=1
   Socks/Node=* NtasksPerN:B:S:C=1:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryCPU=4G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   Gres=(null) Reservation=(null)
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=bash
   WorkDir=/mnt/irisgpfs/users/vplugaru
   Power=
```

Simple interactive job running under SLURM

SLURM - job example (II)

- Many metrics available during and after job execution
  - ↪ including energy (J) – but with caveats
  - ↪ job **steps** counted individually
  - ↪ enabling advanced application debugging and optimization
- Job information available in easily parseable format (add -p/-P)

```
$ sacct -j 2058 --format=account,user,jobid,jobname,partition,state
   Account       User         JobID    JobName  Partition        State
     ulhpc   vplugaru          2058       bash  interacti +  COMPLETED

$ sacct -j 2058 --format=elapsed,elapsedraw,start,end
   Elapsed ElapsedRaw                Start                  End
  00:02:56         176 2017-06-09T16:49:42 2017-06-09T16:52:38

$ sacct -j 2058 --format=maxrss,maxvmsize,consumedenergy,consumedenergyraw,nnodes,ncpus,nodelist
     MaxRSS MaxVMSize ConsumedEnergy ConsumedEnergyRaw NNodes NCPUS      NodeList
          0    299660K         17.89K       17885.000000      1     1         iris-081
```

Job metrics after execution ended

SLURM - design for iris (I)

Partition	# Nodes	Default time	Max time	Max nodes/user
batch*	88 (82%)	0-2:0:0	5-0:0:0	unlimited
interactive	10 (9%)	0-1:0:0	0-4:0:0	2
long	10 (9%)	0-2:0:0	30-0:0:0	2

SLURM - design for iris (I)

Partition	# Nodes	Default time	Max time	Max nodes/user
batch*	88 (82%)	0-2:0:0	5-0:0:0	unlimited
interactive	10 (9%)	0-1:0:0	0-4:0:0	2
long	10 (9%)	0-2:0:0	30-0:0:0	2

QoS	User group	Max cores	Max jobs/user
qos-besteffort	**ALL**	**no limit**	
qos-batch	**ALL**	**1064**	100
qos-interactive	**ALL**	**224**	10
qos-long	**ALL**	**224**	10
qos-batch-001	private	1400	100
qos-interactive-001	private	56	10
qos-long-001	private	56	10

SLURM - design for iris (II)

- **Default partition**: **batch**, meant to receive most user jobs
  - ↪ we hope to see majority of user jobs being able to scale
- All partitions have a correspondingly named **QOS**
  - ↪ granting resource access (**long** – **qos-long**)
  - ↪ any job is tied to one QOS (user specified or inferred)
  - ↪ automation in place to select QOS based on partition

SLURM - design for iris (II)

- **Default partition**: **batch**, meant to receive most user jobs
  - ↪ we hope to see majority of user jobs being able to scale
- All partitions have a correspondingly named **QOS**
  - ↪ granting resource access (**long** – **qos-long**)
  - ↪ any job is tied to one QOS (user specified or inferred)
  - ↪ automation in place to select QOS based on partition

- Preemptible **besteffort** QOS available for `batch` and `interactive` partitions (but not for `long`)
  - ↪ meant to ensure maximum resource utilization
  - ↪ should be used together with checkpointable software
- QOSs specific to particular group accounts exist (discussed later)
  - ↪ granting additional accesses to platform contribuitors

SLURM - design for iris (III)

- **Backfill** scheduling for efficiency
  - ↪ **multifactor job priority** (size, age, fairshare, QOS, . . .)
  - ↪ currently weights set for: job age, partition and fair-share
  - ↪ other factors/decay to be tuned **after observation** period
    - ✓ with more user jobs in the queues
- Resource selection: **consumable resources**
  - ↪ **cores and memory** as consumable (per-core scheduling)
  - ↪ block distribution for cores (best-fit algorithm)
  - ↪ default memory/core: 4GB (4.1GB maximum, rest is for OS)

SLURM - design for iris (III)

- **Backfill** scheduling for efficiency
  - ↪ **multifactor job priority** (size, age, fairshare, QOS, . . .)
  - ↪ currently weights set for: job age, partition and fair-share
  - ↪ other factors/decay to be tuned **after observation** period
    - ✓ with more user jobs in the queues
- Resource selection: **consumable resources**
  - ↪ **cores and memory** as consumable (per-core scheduling)
  - ↪ block distribution for cores (best-fit algorithm)
  - ↪ default memory/core: 4GB (4.1GB maximum, rest is for OS)
- Reliable user process tracking with **cgroups**
  - ↪ cpusets used to constrain cores and RAM (no swap allowed)
  - ↪ task affinity used to bind tasks to cores (hwloc based)
- Hierarchical tree topology defined (for the network)
  - ↪ for optimized job resource allocation

SLURM - design for iris (III)

- **Backfill** scheduling for efficiency
  - ↪ **multifactor job priority** (size, age, fa)
  - ↪ currently weights set for: job age share
  - ↪ other factors/decay to be tuned n period
    - ✓ with more user jobs in th
- Resource selection: **consum**
  - ↪ **cores and memory** as -core scheduling)
  - ↪ block distribution f lgorithm)
  - ↪ default memory (maximum, rest is for OS)
- Reliable user p with **cgroups**
  - ↪ cpusets cores and RAM (no swap allowed)
  - ↪ task nd tasks to cores (hwloc based)
- Hierar ogy defined (for the network)
  - ↪ for b resource allocation

Help will be needed on your part to optimize your job parameters!

A note on job priority

```
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>,
        ...)
```

- TRES – Trackable RESources
  - ↪ CPU, Energy, Memory and Node tracked by default All details at slurm.schedmd.com/priority_multifactor.html
- The corresponding weights and reset periods we **need to tune**
  - ↪ we require (your!) real application usage to optimize them

SLURM - design for iris (IV)

Some details on job permissions...

- Partition limits + association-based rule enforcement
  - ↪ association settings in SLURM's accounting database
- **QOS** limits imposed, e.g. you will see (`QOSGrpCpuLimit`)
- Only users with existing **associations** able to run jobs

SLURM - design for iris (IV)

Some details on job permissions...

- Partition limits + association-based rule enforcement
  - ↪ association settings in SLURM's accounting database
- **QOS** limits imposed, e.g. you will see (`QOSGrpCpuLimit`)
- Only users with existing **associations** able to run jobs

- **Best-effort** jobs possible through preemptible QOS: **qos-besteffort**
  - ↪ of lower priority and preemptible by all other QOS
  - ↪ preemption mode is **requeue**, requeueing enabled by default

SLURM - design for iris (IV)

Some details on job permissions...

- Partition limits + association-based rule enforcement
  - ↪ association settings in SLURM's accounting database
- **QOS** limits imposed, e.g. you will see (`QOSGrpCpuLimit`)
- Only users with existing **associations** able to run jobs

- **Best-effort** jobs possible through preemptible QOS: **qos-besteffort**
  - ↪ of lower priority and preemptible by all other QOS
  - ↪ preemption mode is **requeue**, requeueing enabled by default
- **On metrics**: Accounting & profiling data for jobs sampled every 30s
  - ↪ tracked: cpu, mem, energy
  - ↪ energy data retrieved through the RAPL mechanism
    - ✓ **caveat**: for energy not all hw. that may consume power is monitored with RAPL (CPUs, GPUs and DRAM are included)

SLURM - design for iris (V)

- **On tightly coupled parallel jobs (MPI)**
  - ↪ Process Management Interface (PMI 2) highly recommended
  - ↪ PMI2 used for better scalability and performance
    - ✓ faster application launches
    - ✓ tight integration w. SLURM's job steps mechanism (& metrics)
    - ✓ we are also testing PMIx (PMI Exascale) support

SLURM - design for iris (V)

- **On tightly coupled parallel jobs (MPI)**
  - ↪ Process Management Interface (PMI 2) highly recommended
  - ↪ PMI2 used for better scalability and performance
    - ✓ faster application launches
    - ✓ tight integration w. SLURM's job steps mechanism (& metrics)
    - ✓ we are also testing PMIx (PMI Exascale) support
  - ↪ PMI2 enabled in default software set for IntelMPI and OpenMPI
    - ✓ requires minimal adaptation in your workflows
    - ✓ replace **mpirun** with SLURM's **srun** (at minimum)
    - ✓ if you compile/install your own MPI you'll need to configure it
  - ↪ **Example**: `https://hpc.uni.lu/users/docs/slurm_launchers.html`

SLURM - design for iris (V)

- **On tightly coupled parallel jobs (MPI)**
  - ↪ Process Management Interface (PMI 2) highly recommended
  - ↪ PMI2 used for better scalability and performance
    - ✓ faster application launches
    - ✓ tight integration w. SLURM's job steps mechanism (& metrics)
    - ✓ we are also testing PMIx (PMI Exascale) support
  - ↪ PMI2 enabled in default software set for IntelMPI and OpenMPI
    - ✓ requires minimal adaptation in your workflows
    - ✓ replace **mpirun** with SLURM's **srun** (at minimum)
    - ✓ if you compile/install your own MPI you'll need to configure it
  - ↪ **Example**: `https://hpc.uni.lu/users/docs/slurm_launchers.html`
- **SSH-based connections** between computing nodes still **possible**
  - ↪ other MPI implementations can still use `ssh` as launcher
    - ✓ but **really shouldn't need to**, PMI2 support is everywhere
  - ↪ user jobs are tracked, no job == no access to node

SLURM - design for iris (VI)

ULHPC customizations through plugins

- Job submission rule / filter
  - ↪ for now: QOS initialization (if needed)
  - ↪ more rules to come (group credits, node checks, etc.)
- Per-job temporary directories creation & cleanup
  - ↪ better security and privacy, using kernel namespaces and binding
  - ↪ `/tmp` & `/var/tmp` are `/tmp/$jobid.$rstcnt/[tmp,var_tmp]`
  - ↪ transparent for apps. ran through `srun`
  - ↪ apps. ran with ssh cannot be attached, will see base /tmp!
- X11 forwarding (GUI applications)
  - ↪ enabled with `--x11` parameter to `srun/salloc`
  - ↪ currently being rewritten to play nice with per-job tmpdir
    - ✓ workaround: create job and `ssh -X` to head node (need to propagate job environment)

SLURM - design for iris (VII)

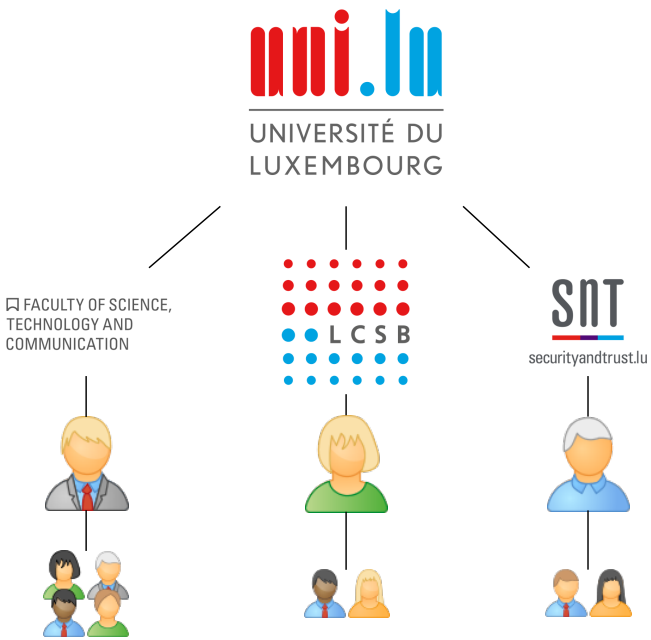
Software licenses in SLURM

- Allinea Forge and Performance Reports for now
  - ↪ static allocation in SLURM configuration
  - ↪ dynamic checks for FlexNet / RLM based apps. coming later
- Number and utilization state can be checked with:
  - ↪ `scontrol show licenses`
- Use not enforced, **honor system** applied
  - ↪ `srun [...] -L $licname:$licnumber`

```
$> srun -N 1 -n 28 -p interactive -L forge:28 --pty bash -i
```

SLURM - bank (group) accounts

- Hierarchical **bank (group) accounts**
- UL as root account, then underneath accounts for the 3 Faculties and 3 ICs
- All Prof., Group leaders and above have **bank accounts**, linked to a Faculty or IC
  - ↪ with their own name: **Name.Surname**
- All **user accounts** linked to a bank account
  - ↪ including Profs.'s own user
- Iris accounting DB contains over
  - ↪ 75 group accounts from all Faculties/ICs
  - ↪ comprising 477 users



Allows better usage tracking and reporting than was possible before.

SLURM - brief commands overview

- **squeue**: view queued jobs
- **sinfo**: view partition and node info.
- **sbatch**: submit job for batch (scripted) execution
- **srun**: submit interactive job, run (parallel) job step
- **scancel**: cancel queued jobs

SLURM - brief commands overview

- **squeue**: view queued jobs
- **sinfo**: view partition and node info.
- **sbatch**: submit job for batch (scripted) execution
- **srun**: submit interactive job, run (parallel) job step
- **scancel**: cancel queued jobs

- **scontrol**: detailed control and info. on jobs, queues, partitions
- **sstat**: view system-level utilization (memory, I/O, energy)
  - ↪ for running jobs / job steps
- **sacct**: view system-level utilization
  - ↪ for completed jobs / job steps (accounting DB)
- **sacctmgr**: view and manage SLURM accounting data

SLURM - brief commands overview

- **squeue**: view queued jobs
- **sinfo**: view partition and node info.
- **sbatch**: submit job for batch (scripted) execution
- **srun**: submit interactive job, run (parallel) job step
- **scancel**: cancel queued jobs

- **scontrol**: detailed control and info. on jobs, queues, partitions
- **sstat**: view system-level utilization (memory, I/O, energy)
  - ↪ for running jobs / job steps
- **sacct**: view system-level utilization
  - ↪ for completed jobs / job steps (accounting DB)
- **sacctmgr**: view and manage SLURM accounting data

- **sprio**: view job priority factors
- **sshare**: view accounting share info. (usage, fair-share, etc.)

SLURM - basic commands

Action	SLURM command
Submit passive/batch job	`sbatch $script`
Start interactive job	`srun --pty bash -i`
Queue status	`squeue`
User job status	`squeue -u $user`
Specific job status (detailed)	`scontrol show job $jobid`
Job metrics (detailed)	`sstat --job $jobid -l`
Job accounting status (detailed)	`sacct --job $jobid -l`
Delete (running/waiting) job	`scancel $jobid`
Hold job	`scontrol hold $jobid`
Resume held job	`scontrol release $jobid`
Node list and their properties	`scontrol show nodes`
Partition list, status and limits	`sinfo`

QOS deduced if not specified, partition needs to be set if not "batch"

SLURM - basic options for sbatch/srun

Action	sbatch/srun option
Request `$n` distributed nodes	`-N $n`
Request `$m` memory per node	`--mem=$mGB`
Request `$mc` memory per core (logical cpu)	`--mem-per-cpu=$mcGB`
Request job walltime	`--time=d-hh:mm:ss`
Request `$tn` tasks per node	`--ntasks-per-node=$tn`
Request `$ct` cores per task (multithreading)	`-c $ct`
Request `$nt` total # of tasks	`-n $nt`
Request to start job at specific `$time`	`--begin $time`
Specify job name as `$name`	`-J $name`
Specify job partition	`-p $partition`
Specify QOS	`--qos $qos`
Specify account	`-A $account`
Specify email address	`--mail-user=$email`
Request email on event	`--mail-type=all[,begin,end,fail]`
Use the above actions in a batch script	`#SBATCH $option`

SLURM - basic options for sbatch/srun

Action	sbatch/srun option
Request $n distributed nodes	`-N $n`
Request $m memory per node	`--mem=$mGB`
Request $mc memory per core (logical cpu)	`--mem-per-cpu=$mcGB`
Request job walltime	`--time=d-hh:mm:ss`
Request $tn tasks per node	`--ntasks-per-node=$tn`
Request $ct cores per task (multithreading)	`-c $ct`
Request $nt total # of tasks	`-n $nt`
Request to start job at specific $time	`--begin $time`
Specify job name as $name	`-J $name`
Specify job partition	`-p $partition`
Specify QOS	`--qos $qos`
Specify account	`-A $account`
Specify email address	`--mail-user=$email`
Request email on event	`--mail-type=all[,begin,end,fail]`
Use the above actions in a batch script	`#SBATCH $option`

- Diff. between **-N**, **-c**, **-n**, **--ntasks-per-node**, **--ntasks-per-core** ?
- Normally you'd specify **-N** and **--ntasks-per-node**
  - ↪ fix the latter to 1 and add **-c** for MPI+OpenMP jobs
- If your application is scalable, just **-n** might be enough
  - ↪ iris is homogeneous (for now)

SLURM - more options for sbatch/srun

Start job when... (dependencies)	sbatch/srun option
these other jobs have started	`-d after:$jobid1:$jobid2`
these other jobs have ended	`-d afterany:$jobid1:$jobid2`
these other jobs have ended with no errors	`-d afterok:$jobid1:$jobid2`
these other jobs have ended with errors	`-d afternok:$jobid1:$jobid2`
all other jobs with the same name have ended	`-d singleton`

Job dependencies and especially "singleton" can be very useful!

SLURM - more options for sbatch/srun

Start job when... (dependencies)	sbatch/srun option
these other jobs have started	`-d after:$jobid1:$jobid2`
these other jobs have ended	`-d afterany:$jobid1:$jobid2`
these other jobs have ended with no errors	`-d afterok:$jobid1:$jobid2`
these other jobs have ended with errors	`-d afternok:$jobid1:$jobid2`
all other jobs with the same name have ended	`-d singleton`

Job dependencies and especially "singleton" can be very useful!

Allocate job at... (specified time)	sbatch/srun option
exact time today	`--begin=16:00`
tomorrow	`--begin=tomorrow`
specific time relative to now	`--begin=now+2hours`
given date and time	`--begin=2017-06-23T07:30:00`

Jobs run like this will wait as PD – Pending with "(BeginTime)" reason

SLURM - more options for sbatch/srun

Start job when... (dependencies)	sbatch/srun option
these other jobs have started	`-d after:$jobid1:$jobid2`
these other jobs have ended	`-d afterany:$jobid1:$jobid2`
these other jobs have ended with no errors	`-d afterok:$jobid1:$jobid2`
these other jobs have ended with errors	`-d afternok:$jobid1:$jobid2`
all other jobs with the same name have ended	`-d singleton`

Job dependencies and especially "singleton" can be very useful!

Allocate job at... (specified time)	sbatch/srun option
exact time today	`--begin=16:00`
tomorrow	`--begin=tomorrow`
specific time relative to now	`--begin=now+2hours`
given date and time	`--begin=2017-06-23T07:30:00`

Jobs run like this will wait as PD – Pending with "(BeginTime)" reason

Other scheduling request	sbatch/srun option
Ask for minimum/maximum # of nodes	`-N minnodes-maxnodes`
Ask for minimum run time (start job faster)	`--time-min=d-hh:mm:ss`
Ask to remove job if deadline can't be met	`--deadline=YYYY-MM-DD[THH:MM[:SS]]`
Run job within pre-created (admin) reservation	`--reservation=$reservationname`
Allocate resources as specified job	`--jobid=$jobid`

Can use --jobid to connect to running job (different than sattach!)

SLURM - environment variables

- 53 input env. vars. can be used to define job parameters
  - ↪ almost all have a command line equivallent
- up to 59 output env. vars. available within job environment
  - ↪ some common ones:

Description	Environment variable
Job ID	`$SLURM_JOBID`
Job name	`$SLURM_JOB_NAME`
Name of account under which job runs	`$SLURM_JOB_ACCOUNT`
Name of partition job is running in	`$SLURM_JOB_PARTITION`
Name of QOS the job is running with	`$SLURM_JOB_QOS`
Name of job's advance reservation	`$SLURM_JOB_RESERVATION`
Job submission directory	`$SLURM_SUBMIT_DIR`
Number of nodes assigned to the job	`$SLURM_NNODES`
Name of nodes assigned to the job	`$SLURM_JOB_NODELIST`
Number of tasks for the job	`$SLURM_NTASKS` or `$SLURM_NPROCS`
Number of cores for the job on current node	`$SLURM_JOB_CPUS_PER_NODE`
Memory allocated to the job per node	`$SLURM_MEM_PER_NODE`
Memory allocated per core	`$SLURM_MEM_PER_CPU`
Task count within a job array	`$SLURM_ARRAY_TASK_COUNT`
Task ID assigned within a job array	`$SLURM_ARRAY_TASK_ID`

Outputting these variables to the job log is essential for bookkeeping!

Usage examples (I)

> Interactive jobs

```
srun -p interactive --qos qos-interactive --time=0:30 -N2 --ntasks-per-node=4 --pty bash -i
srun -p interactive --qos qos-interactive --pty --x11 bash -i
srun -p interactive --qos qos-besteffort --pty bash -i
```

Usage examples (I)

> Interactive jobs

```
srun -p interactive --qos qos-interactive --time=0:30 -N2 --ntasks-per-node=4 --pty bash -i
srun -p interactive --qos qos-interactive --pty --x11 bash -i
srun -p interactive --qos qos-besteffort --pty bash -i
```

> Batch jobs

```
sbatch job.sh
sbatch -N 2 job.sh
sbatch -p batch --qos qos-batch job.sh
sbatch -p long --qos qos-long job.sh
sbatch --begin=2017-06-23T07:30:00 job.sh
sbatch -p batch --qos qos-besteffort job.sh
```

Usage examples (I)

> Interactive jobs

```
srun -p interactive --qos qos-interactive --time=0:30 -N2 --ntasks-per-node=4 --pty bash -i
srun -p interactive --qos qos-interactive --pty --x11 bash -i
srun -p interactive --qos qos-besteffort --pty bash -i
```

> Batch jobs

```
sbatch job.sh
sbatch -N 2 job.sh
sbatch -p batch --qos qos-batch job.sh
sbatch -p long --qos qos-long job.sh
sbatch --begin=2017-06-23T07:30:00 job.sh
sbatch -p batch --qos qos-besteffort job.sh
```

Status and details for partitions, nodes, reservations

```
squeue / squeue -l / squeue -la / squeue -l -p batch / squeue -t PD
scontrol show nodes / scontrol show nodes $nodename
sinfo / sinfo -s / sinfo -N
sinfo -T
```

Usage examples (II)

Collecting job information, priority, expected start time

```
scontrol show job $jobid # this is only available while job is in the queue + 5 minutes
sprio -l
squeue --start -u $USER
```

Usage examples (II)

Collecting job information, priority, expected start time

```
scontrol show job $jobid # this is only available while job is in the queue + 5 minutes
sprio -l
squeue --start -u $USER
```

Running job metrics – sstat tool

```
sstat -j $jobid / sstat -j $jobid -l
sstat -j $jobid1 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize
sstat -p -j $jobid1,$jobid2 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize
```

Usage examples (II)

Collecting job information, priority, expected start time

```
scontrol show job $jobid # this is only available while job is in the queue + 5 minutes
sprio -l
squeue --start -u $USER
```

Running job metrics – sstat tool

```
sstat -j $jobid / sstat -j $jobid -l
sstat -j $jobid1 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize
sstat -p -j $jobid1,$jobid2 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize
```

Completed job metrics – sacct tool

```
sacct -j $jobid / sacct -j $jobid -l
sacct -p -j $jobid --format=account,user,jobid,jobname,partition,state,elapsed,elapsedraw,
\ start,end,maxrss,maxvmsize,consumedenergy,consumedenergyraw,nnodes,ncpus,nodelist
sacct --starttime 2017-06-12 -u $USER
```

Usage examples (III)

Controlling queued and running jobs

```
scontrol hold $jobid
scontrol release $jobid
scontrol suspend $jobid
scontrol resume $jobid
scancel $jobid
scancel -n $jobname
scancel -u $USER
scancel -u $USER -p batch
scontrol requeue $jobid
```

Usage examples (III)

Controlling queued and running jobs

```
scontrol hold $jobid
scontrol release $jobid
scontrol suspend $jobid
scontrol resume $jobid
scancel $jobid
scancel -n $jobname
scancel -u $USER
scancel -u $USER -p batch
scontrol requeue $jobid
```

Checking accounting links and QOS available for you

```
sacctmgr show user $USER format=user%20s,defaultaccount%30s  sacctmgr list association where
users=$USER format=account%30s,user%20s,qos%120s
```

Usage examples (III)

Controlling queued and running jobs

```
scontrol hold $jobid
scontrol release $jobid
scontrol suspend $jobid
scontrol resume $jobid
scancel $jobid
scancel -n $jobname
scancel -u $USER
scancel -u $USER -p batch
scontrol requeue $jobid
```

Checking accounting links and QOS available for you

```
sacctmgr show user $USER format=user%20s,defaultaccount%30s  sacctmgr list association where
users=$USER format=account%30s,user%20s,qos%120s
```

Checking accounting share info - usage, fair-share, etc.

```
sshare -U
sshare -A $accountname
sshare -A $(sacctmgr -n show user $USER format=defaultaccount%30s)
sshare -a
```

Job launchers - basic (I)

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=0-00:05:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "Hello from the batch queue on node ${SLURM_NODELIST}"
# Your more useful application can be started below!
```

Submit it with: `sbatch launcher.sh`

Job launchers - basic (II)

```
#!/bin/bash -l
#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH --time=0-03:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
```

Job launchers - basic (III)

```
#!/bin/bash -l
#SBATCH -J MyTestJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH --time=0-03:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
```

Job launchers - requesting memory

```
#!/bin/bash -l
#SBATCH -J MyLargeMemorySequentialJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=64GB
#SBATCH --time=1-00:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
```

Use "mem" to request (more) memory per node for low #core jobs

Job launchers - long jobs

```
#!/bin/bash -l
#SBATCH -J MyLongJob
#SBATCH --mail-type=all
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=3-00:00:00
#SBATCH -p long
#SBATCH --qos=qos-long

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
```

Longer walltime now possible but you should not (!) rely on it.
Always prefer batch and requeue-able jobs.

Job launchers - besteffort

```
#!/bin/bash -l
#SBATCH -J MyRerunnableJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH --time=0-12:00:00
#SBATCH -p batch
#SBATCH --qos=qos-besteffort
#SBATCH --requeue

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
```

Many scientific applications support internal state saving and restart!
We will also discuss system-level checkpoint-restart with DMTCP.

Job launchers - threaded parallel

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
/path/to/your/threaded.app
```

By threaded we mean pthreads/OpenMP shared-memory applications.

Job launchers - MATLAB

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load base/MATLAB
matlab -nodisplay -nosplash < /path/to/infile > /path/to/outfile
```

MATLAB spawns processes, limited for now to single node execution.
We are still waiting for Distributed Computing Server availability.

Job launchers - MATLAB

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load base/MATLAB
matlab -nodisplay -nosplash < /path/to/infile > /path/to/outfile
```

MATLAB spawns processes, limited for now to single node execution.
We are still waiting for Distributed Computing Server availability.

A note on parallel jobs

Currently the iris cluster is homogeneous.
Its core networking is a non-blocking fat-tree.

- For now simply requesting a number of tasks (with 1 core/task) **should be performant**
- Different MPI implementations will however **behave differently**
  - ↪ very recent/latest versions available on **iris** for IntelMPI, OpenMPI, MVAPICH2
  - ↪ we ask that you let us know any perceived benefit for your applications when using one or the other
- We can make available optimized MPI-layer parameters obtained during our tuning runs
  - ↪ and hope they will improve even more your time to solution

Job launchers - IntelMPI

```
#!/bin/bash -l
#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
srun -n $SLURM_NTASKS /path/to/your/intel-toolchain-compiled-app
```

IntelMPI is configured to use PMI2 for process management (optimal).
Bare mpirun will not work for now.

Job launchers - OpenMPI

```
#!/bin/bash -l
#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/foss
srun -n $SLURM_NTASKS /path/to/your/foss-toolchain-compiled-app
```

OpenMPI also uses PMI2 (again, optimal).
Bare mpirun does work but is not recommended.

You can easily generate a hostfile from within a SLURM job with:
`srun hostname | sort -n > hostfile`

Job launchers - MPI+OpenMP

```
#!/bin/bash -l
#SBATCH -N 10
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
srun -n $SLURM_NTASKS /path/to/your/parallel-hybrid-app
```

Compile and use your applications in hybrid MPI+OpenMP mode when you can for better (best?) possible performance.

Summary

Notes on OAR

- OAR will remain the workload manager of Gaia and Chaos
  - ↪ celebrating **4250995** jobs on Gaia! (2017-11-07)
  - ↪ celebrating **1615659** jobs on Chaos! (2017-11-07)

- Many of its features are common to other workload managers, incl. SLURM
  - ↪ some things are exactly the same
  - ↪ but some things work in a different way
  - ↪ . . . and some have no equivallent or are widely different
- An adjustment period for you is needed if you've only used OAR
  - ↪ next slides show a brief transition guide

OAR/SLURM - commands guide

Command	OAR (gaia/chaos)	SLURM (iris)
Submit passive/batch job	`oarsub -S $script`	`sbatch $script`
Start interactive job	`oarsub -I`	`srun -p interactive --qos qos-interactive --pty bash -i`
Queue status	`oarstat`	`squeue`
User job status	`oarstat -u $user`	`squeue -u $user`
Specific job status (detailed)	`oarstat -f -j $jobid`	`scontrol show job $jobid`
Delete (running/waiting) job	`oardel $jobid`	`scancel $jobid`
Hold job	`oarhold $jobid`	`scontrol hold $jobid`
Resume held job	`oarresume $jobid`	`scontrol release $jobid`
Node list and properties	`oarnodes`	`scontrol show nodes`

Similar yet different?
Many specifics will actually come from the way Iris is set up.

OAR/SLURM - job specifications

Specification	OAR	SLURM
Script directive	`#OAR`	`#SBATCH`
Nodes request	`-l nodes=$count`	`-N $min-$max`
Cores request	`-l core=$count`	`-n $count`
Cores-per-node request	`-l nodes=$ncount/core=$ccount`	`-N $ncount --ntasks-per-node=$ccount`
Walltime request	`-l [...],walltime=hh:mm:ss`	`-t $min OR -t $days-hh:mm:ss`
Job array	`--array $count`	`--array $specification`
Job name	`-n $name`	`-J $name`
Job dependency	`-a $jobid`	`-d $specification`
Property request	`-p "$property=$value"`	`-C $specification`

Job specifications will need most adjustment on your side
... but thankfully Iris has a homogeneous configuration.
Running things in an optimal way will be much easier.

OAR/SLURM - env. vars.

Environment variable	OAR	SLURM
Job ID	`$OAR_JOB_ID`	`$SLURM_JOB_ID`
Resource list	`$OAR_NODEFILE`	`$SLURM_NODELIST` #List not file! See below.
Job name	`$OAR_JOB_NAME`	`$SLURM_JOB_NAME`
Submitting user name	`$OAR_USER`	`$SLURM_JOB_USER`
Task ID within job array	`$OAR_ARRAY_INDEX`	`$SLURM_ARRAY_TASK_ID`
Working directory at submission	`$OAR_WORKING_DIRECTORY`	`$SLURM_SUBMIT_DIR`

Check available variables: `env | egrep "OAR|SLURM"`
Generate hostfile: `srun hostname | sort -n > hostfile`

Summary

Conclusion and Practical Session start

We've discussed

- The design of SLURM for the **iris** cluster
- The permissions system in use through group accounts and QOS
- Main SLURM tools and how to use them
- Job types possible with SLURM on **iris**
- SLURM job launchers for sequential and parallel applications
- Transitioning from OAR to SLURM

And now..

Short DEMO time!

Conclusion and Practical Session start

We've discussed

- The design of SLURM for the **iris** cluster
- The permissions system in use through group accounts and QOS
- Main SLURM tools and how to use them
- Job types possible with SLURM on **iris**
- SLURM job launchers for sequential and parallel applications
- Transitioning from OAR to SLURM

And now..

Short DEMO time!

Your Turn!

Questions?

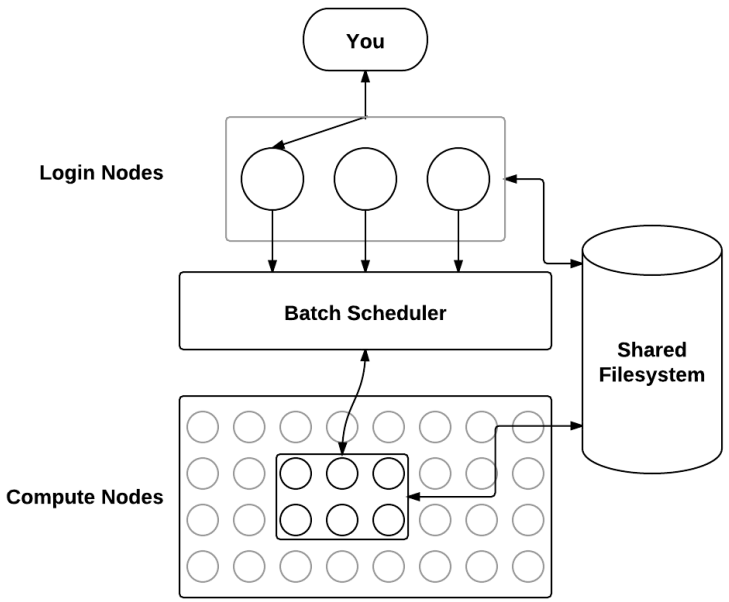
http://hpc.uni.lu

High Performance Computing @ UL

Prof. Pascal Bouvry
Dr. Sebastien Varrette & the UL HPC Team
(V. Plugaru, S. Peter, H. Cartiaux & C. Parisot)
University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu

Resource and Job Management Systems

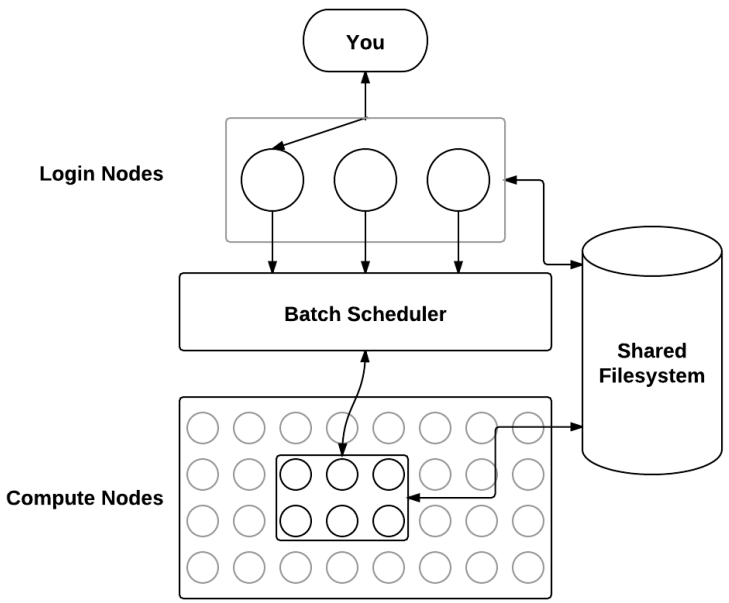
- **Resource and Job Management System (RJMS)**
  - ↪ "Glue" for a parallel computer to execute parallel jobs
  - ↪ **Goal**: satisfy users' demands for computation
    - ✓ assign resources to user jobs with an efficient manner



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



RJMS Layers

- Resource Allocation involves three principal abstraction layers:
  - ↪ **Job Management**:
    - ✓ declaration of a job & demand of resources and job characteristics,
  - ↪ **Scheduling**: matching of the jobs upon the resources,
  - ↪ **Resource Management**:
    - ✓ launching and placement of job instances. . .
    - ✓ . . . along with the job's control of execution

- **When there is more work than resources**
  - ↪ the job scheduler manages queue(s) of work
    - ✓ supports complex scheduling algorithms
  - ↪ Supports resource limits (by queue, user, group, etc.)

RJMS Detailed Components

- **Resource Management**
  - ↪ Resource Treatment (hierarchy, partitions,..)
  - ↪ Job Launcing, Propagation, Execution control
  - ↪ Task Placement (topology,binding,..)
  - ↪ **Advanced Features**:
    - ✓ High Availability, Energy Efficiency, Topology aware placement

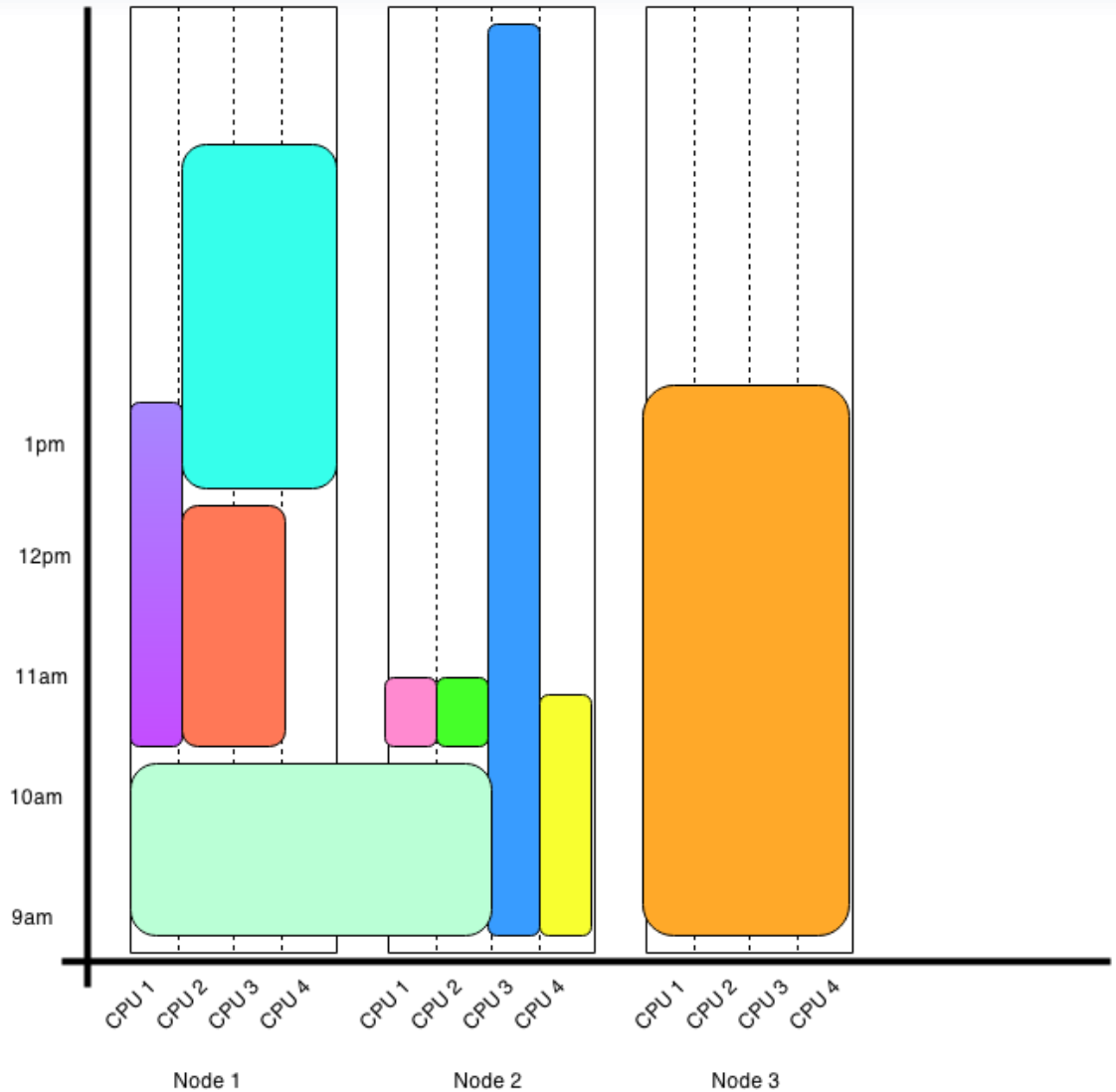
RJMS Detailed Components

- **Resource Management**
  - ↪ Resource Treatment (hierarchy, partitions,..)
  - ↪ Job Launcing, Propagation, Execution control
  - ↪ Task Placement (topology,binding,..)
  - ↪ **Advanced Features**:
    - ✓ High Availability, Energy Efficiency, Topology aware placement
- **Job Management**
  - ↪ Job declaration and control (signaling, reprioritizing,. . .)
  - ↪ Monitoring (reporting, visualization,..)
  - ↪ **Advanced Features**:
    - ✓ Authentication (limitations, security,..)
    - ✓ QOS (checkpoint, suspend, accounting,. . .)
    - ✓ Interfacing (MPI libraries, debuggers, APIs,..)

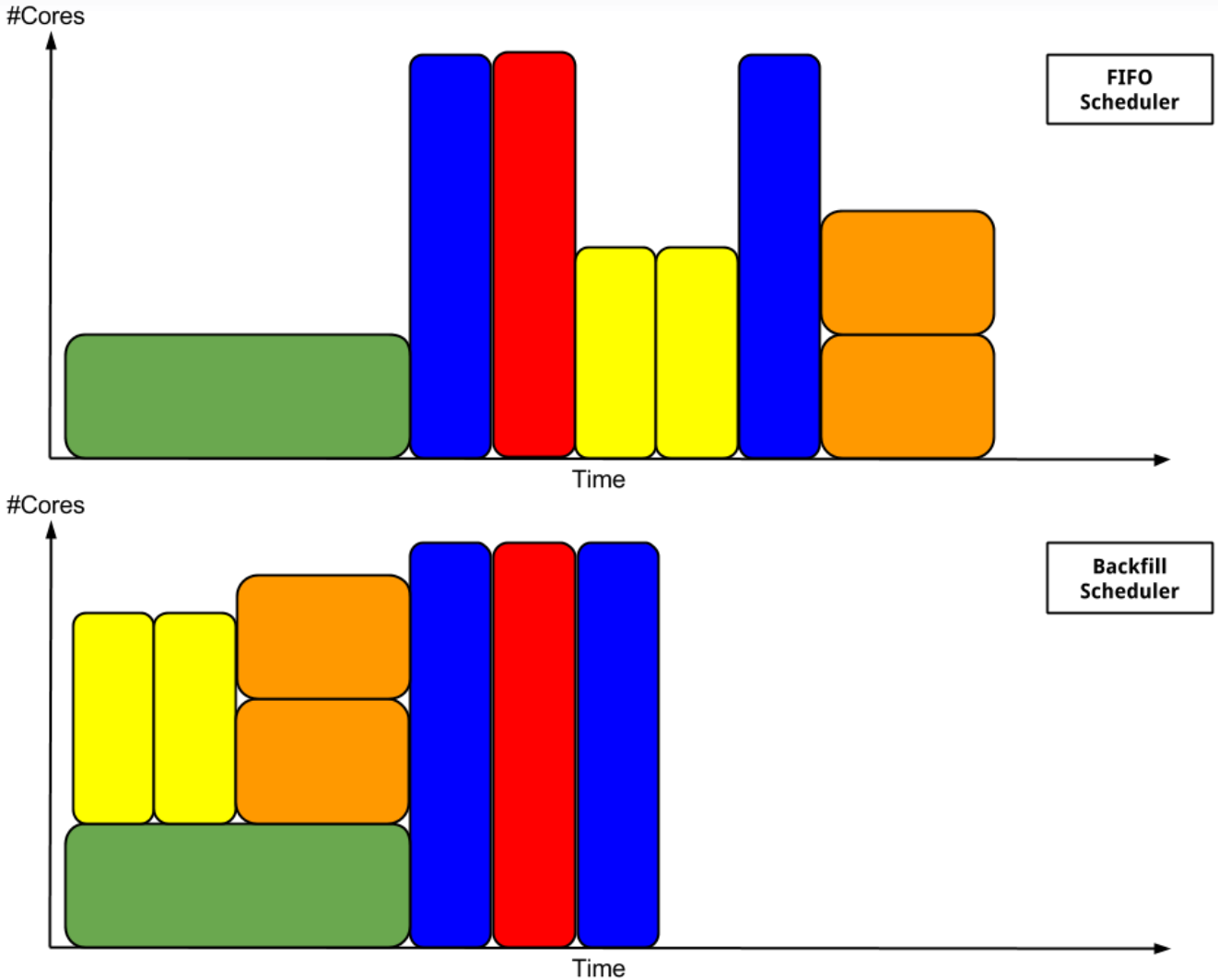
RJMS Detailed Components

- **Resource Management**
  - ↪ Resource Treatment (hierarchy, partitions,..)
  - ↪ Job Launcing, Propagation, Execution control
  - ↪ Task Placement (topology,binding,..)
  - ↪ **Advanced Features**:
    - ✓ High Availability, Energy Efficiency, Topology aware placement
- **Job Management**
  - ↪ Job declaration and control (signaling, reprioritizing,. . .)
  - ↪ Monitoring (reporting, visualization,..)
  - ↪ **Advanced Features**:
    - ✓ Authentication (limitations, security,..)
    - ✓ QOS (checkpoint, suspend, accounting,. . .)
    - ✓ Interfacing (MPI libraries, debuggers, APIs,..)
- **Scheduling**
  - ↪ Queues Management (priorities,multiple,..)
  - ↪ Advanced Reservation

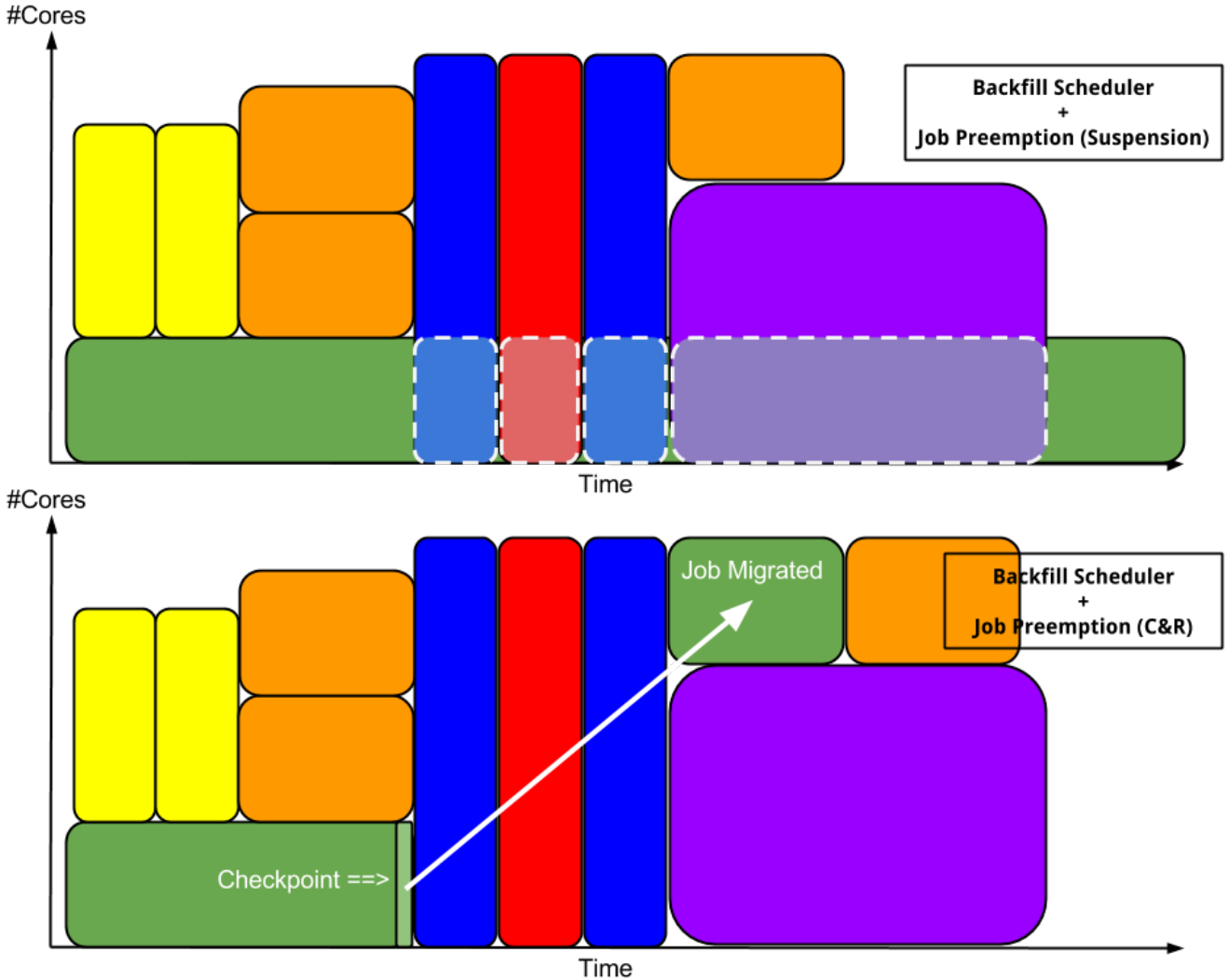
Job Scheduling



Job Scheduling (backfilling)



Job Scheduling (suspension & requeue)



Main Job Schedulers

Name	Company	Version*
SLURM	SchedMD	17.02.8
LSF	IBM	10.1
OpenLava	LSF Fork	2.2
MOAB/Torque	Adaptative Computing	6.1
PBS	Altair	13.0
OAR (PBS Fork)	LIG	2.5.7
Oracle Grid Engine (formely SGE)	Oracle	

*: *As of Oct. 2017*

Main Job Schedulers

Name	Company	Version*
SLURM	SchedMD	17.02.8
LSF	IBM	10.1
OpenLava	LSF Fork	2.2
MOAB/Torque	Adaptative Computing	6.1
PBS	Altair	13.0
OAR (PBS Fork)	LIG	2.5.7
Oracle Grid Engine (formely SGE)	Oracle	

*: *As of Oct. 2017*

UL HPC resource manager: OAR

The OAR Batch Scheduler

http://oar.imag.fr

- Versatile resource and task manager
  - ↪ schedule **jobs** for users on the cluster **resource**
  - ↪ OAR resource = a node or part of it (CPU/core)
  - ↪ OAR job = execution time (**walltime**) on a set of resources

UL HPC resource manager: OAR

The OAR Batch Scheduler

http://oar.imag.fr

- Versatile resource and task manager
  - ↪ schedule **jobs** for users on the cluster **resource**
  - ↪ OAR resource = a node or part of it (CPU/core)
  - ↪ OAR job = execution time (**walltime**) on a set of resources

OAR main features includes:

- **interactive vs. passive (aka. batch) jobs**
- **best effort jobs**: use more resource, accept their release any time
- **deploy jobs** (Grid5000 only): deploy a customized OS environment
  - ↪ ... and have full (root) access to the resources
- **powerful resource filtering/matching**

Main OAR commands

oarsub	submit/reserve a job	(by default: **1 core for 2 hours**)
oardel	delete a submitted job	
oarnodes	shows the resources states	
oarstat	shows information about running or planned jobs	

	Submission
interactive	`oarsub [options] -I`
passive	`oarsub [options] scriptName`

- Each created job receive an identifier `JobID`
  - ↪ Default passive job log files: `OAR.JobID.std{out,err}`
- You can make a reservation with `-r "YYYY-MM-DD HH:MM:SS"`

Main OAR commands

oarsub	submit/reserve a job	(by default: **1 core for 2 hours**)
oardel	delete a submitted job	
oarnodes	shows the resources states	
oarstat	shows information about running or planned jobs	

	Submission
interactive	`oarsub [options] -I`
passive	`oarsub [options] scriptName`

- Each created job receive an identifier `JobID`
  - ↪ Default passive job log files: `OAR.JobID.std{out,err}`
- You can make a reservation with `-r "YYYY-MM-DD HH:MM:SS"`

Direct access to nodes by `ssh` is forbidden: use `oarsh` instead

OAR job environment variables

Once a job is created, some environments variables are defined:

Variable	Description
`$OAR_NODEFILE`	Filename which lists all reserved nodes for this job
`$OAR_JOB_ID`	OAR job identifier
`$OAR_RESOURCE_PROPERTIES_FILE`	Filename which lists all resources and their properties
`$OAR_JOB_NAME`	Name of the job given by the "-n" option of oarsub
`$OAR_PROJECT_NAME`	Job project name

Useful for MPI jobs for instance:

```
$> mpirun -machinefile $OAR_NODEFILE /path/to/myprog
```

... Or to collect how many cores are reserved per node:

```
$> cat $OAR_NODEFILE | uniq -c
```

OAR job types

Job Type	Max Walltime (hour)	Max #active_jobs	Max #active_jobs_per_user
interactive	12:00:00	10000	5
default	120:00:00	30000	10
besteffort	9000:00:00	10000	1000

cf `/etc/oar/admission_rules/*.conf`

- **interactive**: useful to test / prepare an experiment
  - ↪ you get a shell on the first reserved resource
- **best-effort vs. default**: nearly unlimited constraints **YET**
  - ↪ a `besteffort` job can be killed as soon as a default job as no other place to go
  - ↪ enforce checkpointing (and/or idempotent) strategy

Characterizing OAR resources

Specifying wanted resources in a hierarchical manner

- Use the `-l` option of `oarsub`. Main constraints:

`enclosure=N`	number of enclosure
`nodes=N`	number of nodes
`core=N`	number of cores
`walltime=hh:mm:ss`	job's max duration

Specifying OAR resource properties

- Use the `-p` option of `oarsub`:

Syntax: `-p "property='value'"`

`gpu='{YES,NO}'`	has (or not) a GPU card
`host='fqdn'`	full hostname of the resource
`network_address='hostname'`	Short hostname of the resource
(Chaos only) `nodeclass='{k,b,h,d,r}'`	Class of node

OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15: Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15: Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

- 4 cores on a GPU node for 8 hours Total: 4 cores

```
(frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
```

OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15: Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

- 4 cores on a GPU node for 8 hours Total: 4 cores

```
(frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
```

- 2 nodes among the `h-cluster1-*` nodes (Chaos only) Total: 24 cores

```
(frontend)$> oarsub -I -l nodes=2 -p "nodeclass='h'"
```

OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15: Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

- 4 cores on a GPU node for 8 hours Total: 4 cores

```
(frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
```

- 2 nodes among the `h-cluster1-*` nodes (Chaos only) Total: 24 cores

```
(frontend)$> oarsub -I -l nodes=2 -p "nodeclass='h'"
```

- 4 cores on 2 GPU nodes + 20 cores on other nodes Total: 28 cores

```
$> oarsub -I -l "{gpu='YES'}/nodes=2/core=4+{gpu='NO'}/core=20"
```

OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15: Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

- 4 cores on a GPU node for 8 hours Total: 4 cores

```
(frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
```

- 2 nodes among the `h-cluster1-*` nodes (Chaos only) Total: 24 cores

```
(frontend)$> oarsub -I -l nodes=2 -p "nodeclass='h'"
```

- 4 cores on 2 GPU nodes + 20 cores on other nodes Total: 28 cores

```
$> oarsub -I -l "{gpu='YES'}/nodes=2/core=4+{gpu='NO'}/core=20"
```

- A full big SMP node Total: 160 cores on gaia-74

```
$> oarsub -t bigsmp -I l node=1
```

Some other useful features of OAR

Connect to a running job

```
(frontend)$> oarsub -C JobID
```

Cancel a job

```
(frontend)$> oardel JobID
```

Status of a jobs

```
(frontend)$> oarstat -state -j JobID
```

View the job

```
(frontend)$> oarstat
(frontend)$> oarstat -f -j JobID
```

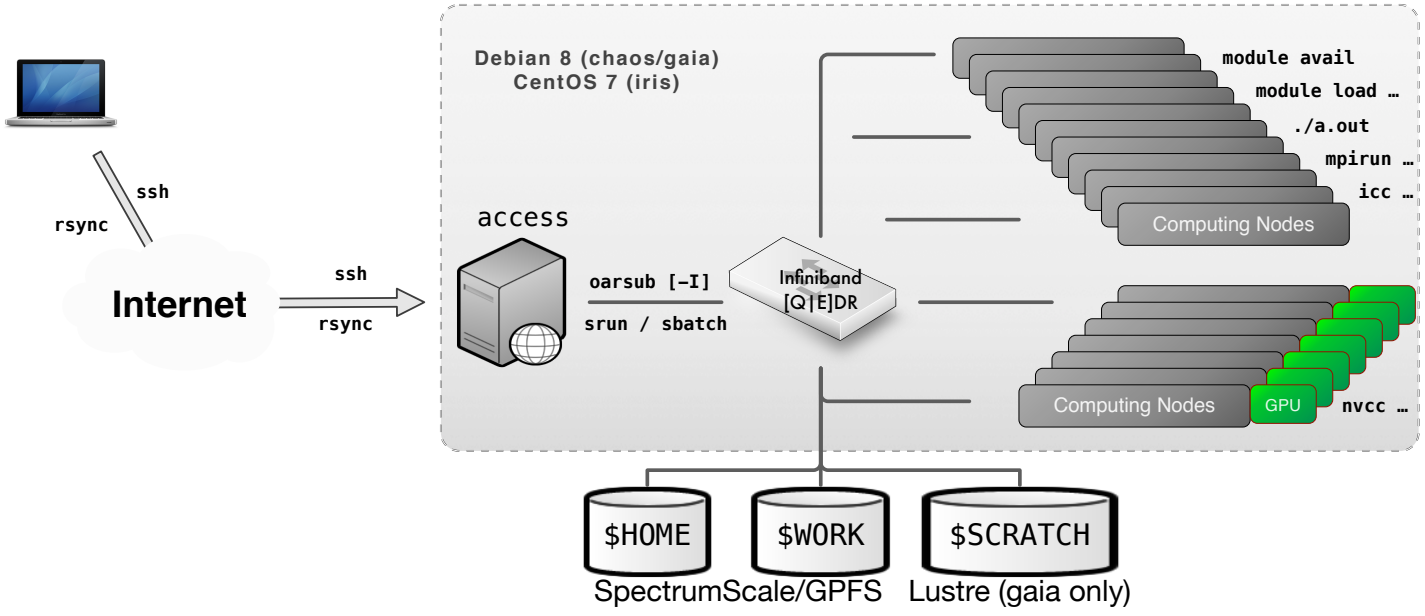
Get info on the nodes

```
(frontend)$> oarnodes
(frontend)$> oarnodes -l
(frontend)$> oarnodes -s
```

Run a best-effort job

```
(frontend)$> oarsub -t besteffort ...
```

OAR Practical session



Demo Time

- `gaia` or `chaos` UL cluster access
- Interactive / Passive job submission

Designing efficient OAR job launchers

Resources/Example

https://github.com/ULHPC/launcher-scripts

- UL HPC grant access to **parallel computing** resources
  - ↪ ideally: OpenMP/MPI/CUDA/OpenCL jobs
  - ↪ if serial jobs/tasks: run them efficiently
- Avoid to submit purely serial jobs to the OAR queue a
  - ↪ waste the computational power (11 out of 12 cores on `gaia`).
  - ↪ use whole nodes by running at least 12 serial runs at once
- **Key**: understand difference between **Task** and **OAR job**

Designing efficient OAR job launchers

Resources/Example

https://github.com/ULHPC/launcher-scripts

- UL HPC grant access to **parallel computing** resources
  - ↪ ideally: OpenMP/MPI/CUDA/OpenCL jobs
  - ↪ if serial jobs/tasks: run them efficiently
- Avoid to submit purely serial jobs to the OAR queue a
  - ↪ waste the computational power (11 out of 12 cores on `gaia`).
  - ↪ use whole nodes by running at least 12 serial runs at once
- **Key**: understand difference between **Task** and **OAR job**

For more information...

- Incoming Practical Session
  - ↪ HPC workflow with sequential jobs (C,python,java etc.)

OAR Simple Example of usage

```
# Simple interactive job
(access)$> oarsub -I
(node)$> echo $OAR_JOBID
4239985
(node)$> echo $OAR_NODEFILE
/var/lib/oar//4239985
(node)$> cat $OAR_NODEFILE | wc -l
8
(node)$> cat $OAR_NODEFILE
moonshot1-39
moonshot1-39
moonshot1-39
moonshot1-39
moonshot1-40
moonshot1-40
moonshot1-40
moonshot1-40
```

View existing job

```
# View YOUR jobs (remove -u to view all)
(access)$> oarstat -u
Job id     Name    User       Submission Date     S Queue
---------- ------- ---------- ------------------- - ----------
4239985            svarrette  2017-10-23 12:33:41 R default
```

View Detailed info on jobs

```
(access)$> oarstat -f -j 4239985
Job_Id: 4239985
    [...]
    state = Running
    wanted_resources = -l "{type = 'default'}/ibpool=1/host=2,walltime=2:0:0
    types = interactive, inner=4236343, moonshot
    assigned_resources = 3309+3310+3311+3312+3313+3314+3315+3316
    assigned_hostnames = moonshot1-39+moonshot1-40
    queue = default
    launchingDirectory = /home/users/svarrette
    stdout_file = OAR.4239985.stdout
    stderr_file = OAR.4239985.stderr
    jobType = INTERACTIVE
    properties = (((bigmem='NO' AND bigsmp='NO') AND dedicated='NO') AND os=
    walltime = 2:0:0
    initial_request = oarsub -I -l nodes=2 -t moonshot -t inner=4236343
    message = R=8,W=2:0:0,J=I,T=inner|interactive|moonshot (Karma=1.341)
```

Access to an existing job: Attempt 1

```
# Get your job ID...
(access)$> oarstat -u
```

- **Attempt 1**: Get assigned resources and ...

Access to an existing job: Attempt 1

```
# Get your job ID...
(access)$> oarstat -u
```

- **Attempt 1**: Get assigned resources and ... ssh to it!

```
# Collect the assigned resources
(access)$>  oarstat -f -j 4239985 | grep hostname
   assigned_hostnames = moonshot1-39+moonshot1-40

(access)$> ssh moonshot1-39
[...]
================================================================
 /!\ WARNING:   Direct login by ssh is forbidden.

 Use oarsub(1) to reserve nodes, and oarsh(1) to connect to your
 reserved nodes, typically by:
        OAR_JOB_ID=<jobid> oarsh <nodename>
================================================================
```

Access to an existing job

- **Using `oarsh`:**

```
# Get your job ID...
(access)$> oarstat -u
# ... get the hostname of the nodes allocated ...
(access)$> oarstat -f -j 4239985 | grep hostname
# ... and connect to it with oarsh
(access)$> OAR_JOB_ID=4239985 oarsh moonshot1-39
```

Access to an existing job

- **Using `oarsh`**:

```
# Get your job ID...
(access)$> oarstat -u
# ... get the hostname of the nodes allocated ...
(access)$> oarstat -f -j 4239985 | grep hostname
# ... and connect to it with oarsh
(access)$> OAR_JOB_ID=4239985 oarsh moonshot1-39
```

- (better) **Using `oarsub -C`**:

```
# Get your job ID...
(access)$> oarstat -u
# ... and connect to the FIRST node of the reservation
(access)$> oarsub -C 4239985
```

MPI jobs

- Intel MPI

```
(node)$> module load toolchain/intel
# ONLY on moonshot node have no IB card: export I_MPI_FABRICS=tcp
(node)$> mpirun -hostfile $OAR_NODEFILE /path/to/mpiprog
```

- OpenMPI:

```
(node)$> module load mpi/OpenMPI
(node)$> mpirun -hostfile $OAR_NODEFILE -x PATH -x LD_LIBRARY_PATH \
         /path/to/mpiprog
```

- For more details: See MPI sessions

  https://github.com/ULHPC/launcher-scripts/blob/devel/bash/MPI/mpi_launcher.sh

OAR Launcher Scripts

```
$> oarsub -S <scriptname>          # -S: interpret #OAR comments
```

- Our launcher scripts on Github: https://github.com/ULHPC/launcher-scripts
  ↪ see in particular our generic launcher compliant w. OAR & SLURM
- **Example**:

```
#! /bin/bash
#OAR -l nodes=2/core=1,walltime=1
#OAR -n MyNamedJob
# Prepare UL HPC modules
if [ -f  /etc/profile ]; then
.  /etc/profile
fi

module load toolchain/intel
/path/to/prog <ARGS>
```

Slurm Workload Manager

- **S**imple **L**inux **U**tility for **R**esource **M**anagement
  - ↪ Development started in 2002
    - ✓ initially as a simple resource manager for Linux clusters
  - ↪ Has evolved into a capable job scheduler through use of opt. plugins
  - ↪ About 500,000 lines of C code today.
  - ↪ Supports AIX, Linux, Solaris, other Unix variants
- Used on many of the world's largest computers

Slurm Workload Manager

- **S**imple **L**inux **U**tility for **R**esource **M**anagement
  - ↪ Development started in 2002
    - ✓ initially as a simple resource manager for Linux clusters
  - ↪ Has evolved into a capable job scheduler through use of opt. plugins
  - ↪ About 500,000 lines of C code today.
  - ↪ Supports AIX, Linux, Solaris, other Unix variants
- Used on many of the world's largest computers

- **Now deployed on new UL HPC clusters**
  - ↪ starting `iris` cluster (2017)

Slurm Design Goals

- **Small and simple**
- **Highly scalable** and **Fast**
  - ↪ managing 1.6 million core IBM BlueGene/Q,
  - ↪ tested to 33 million cores using emulation
  - ↪ throughput: up to 600 jobs p.s. & 1000 job submissions p.s.
- **Modular**:
  - ↪ plugins to support $\neq$ scheduling policies, MPI librairies...
- **Secure** and **Fault-tolerant**
  - ↪ highly tolerant of system failures
- **Power Management** and detailed monitoring

- **Open source**: GPL v2, active world-wide development
- Portable: written in C with a GNU autoconf configuration engine

Slurm Docs and Resources

https://slurm.schedmd.com/

- User and Admins latest documentation:
  - ↪ http://slurm.schedmd.com/documentation.html
- Detailed man pages for commands and configuration files
  - ↪ http://slurm.schedmd.com/man_index.html
- All SLURM related publications and presentations:
  - ↪ http://slurm.schedmd.com/publications.html

ULHPC Documentation & comparison to OAR

https://hpc.uni.lu/users/docs/scheduler.html

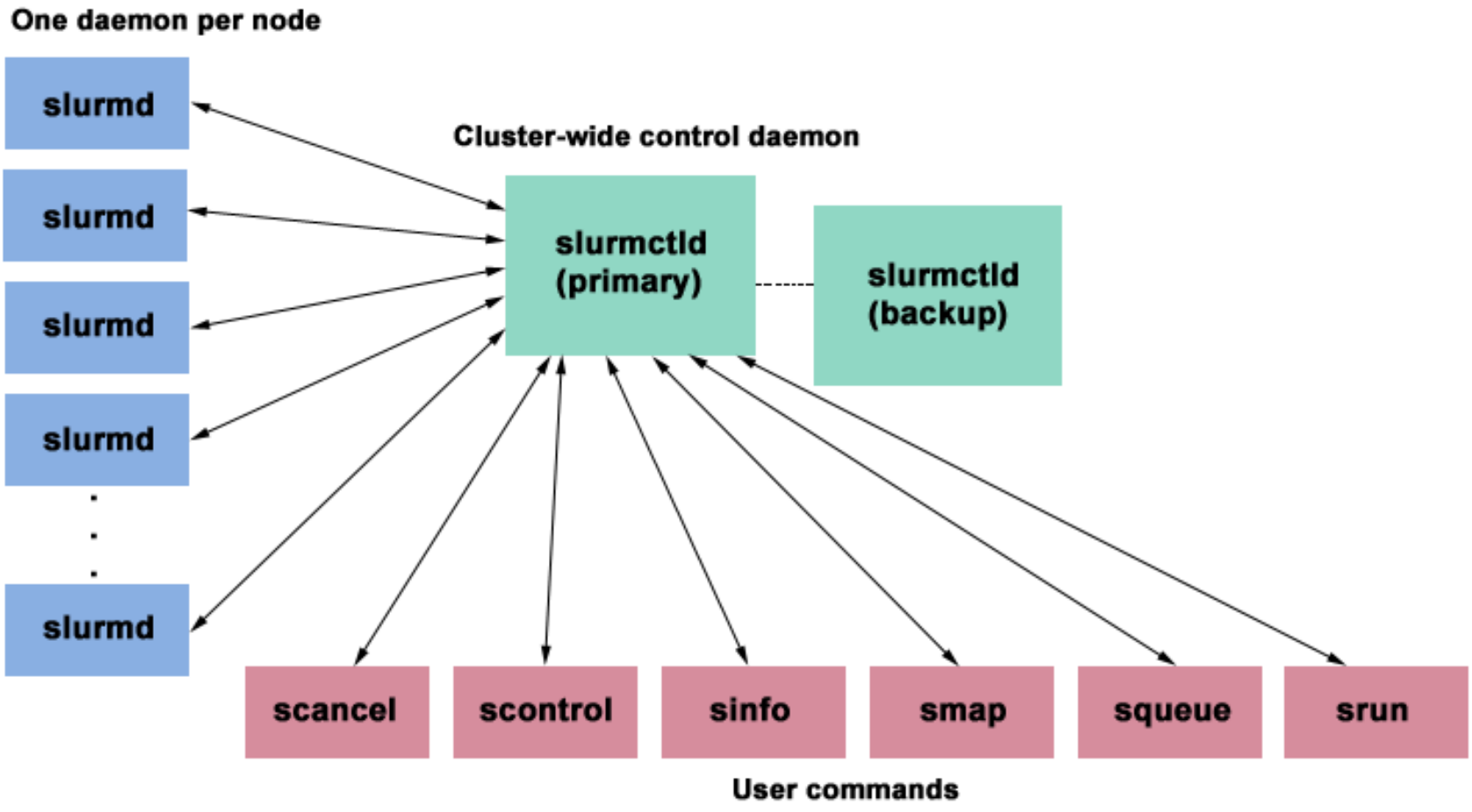
Other Resources

- **Puppet module** `ULHPC/slurm`
  - ↪ Developed by the UL HPC Team – see `https://forge.puppet.com/ULHPC/`
  - ↪ Used in production on `iris` cluster
  - ↪ see also Slurm Control Repo Example

Puppet Class	Description
`slurm`	The main slurm class, piloting all aspects of the configuration
`slurm::slurmdbd`	Specialized class for Slurmdbd, the Slurm Database Daemon.
`slurm::slurmctld`	Specialized class for Slurmctld, the central management daemon of Slurm.
`slurm::slurmd`	Specialized class for Slurmd, the compute node daemon for Slurm.
`slurm::munge`	Manages MUNGE, an authentication service for creating and validating credentials.
`slurm::pam`	Handle PAM aspects for SLURM (Memlock for MPI etc.)

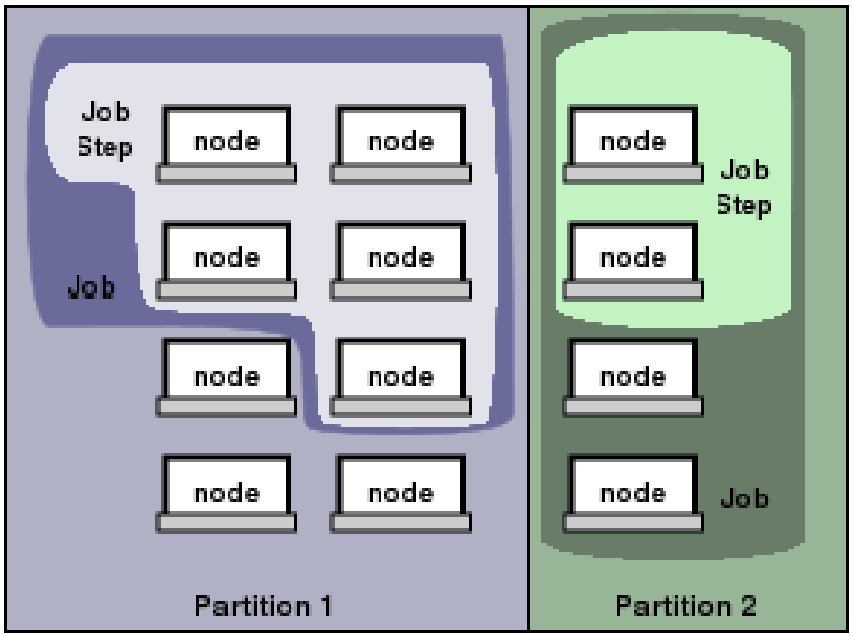
Puppet Defines	Description
`slurm::download`	takes care of downloading the SLURM sources for a given version passed as resource name
`slurm::build`	building Slurm sources into packages (_i.e. RPMs for the moment)
`slurm::install::packages`	installs the Slurm packages, typically built from `slurm::build`
`slurm::acct::*`	adding (or removing) accounting resources to the slurm database

SLURM Architecture

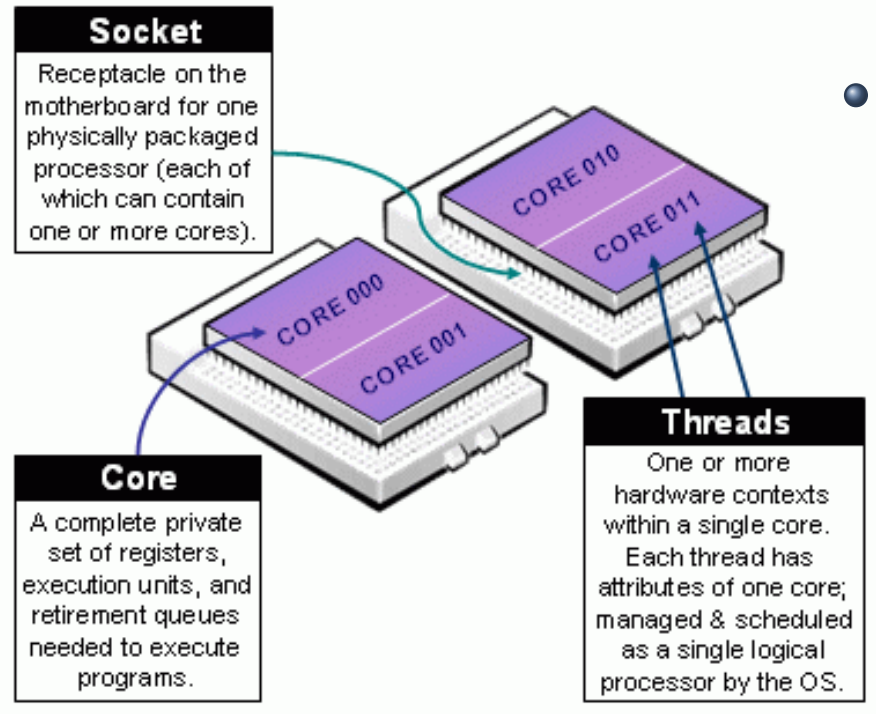


SLURM Entities

Entity	Description
Computing node	Computer used for the execution of programs
Partition	Group of nodes into logical sets
Job	Allocation of resources assigned to a user for some time
Job Step	Sets of (possible parallel) tasks with a job

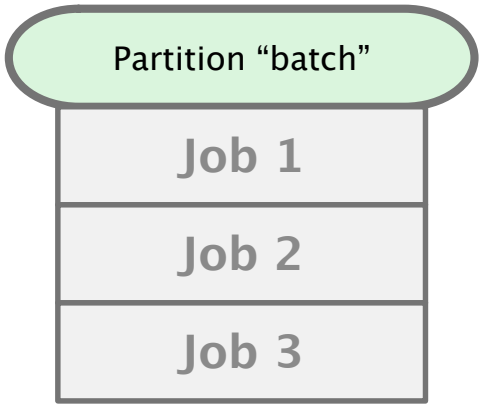


SLURM Multi-Core/Thread Support



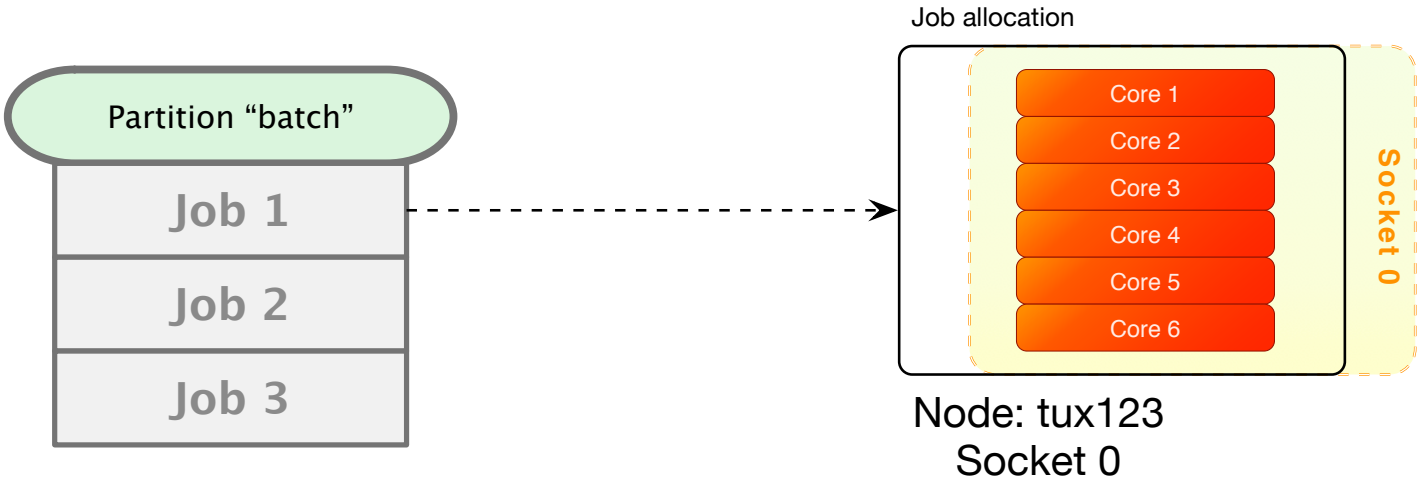
- **Nodes** hierarchy
  - ↪ NUMA [base]board
    - ✓ Socket/Core/Thread
  - ↪ Memory
  - ↪ Generic Resources GRES (e.g. GPUs)

SLURM Entities example



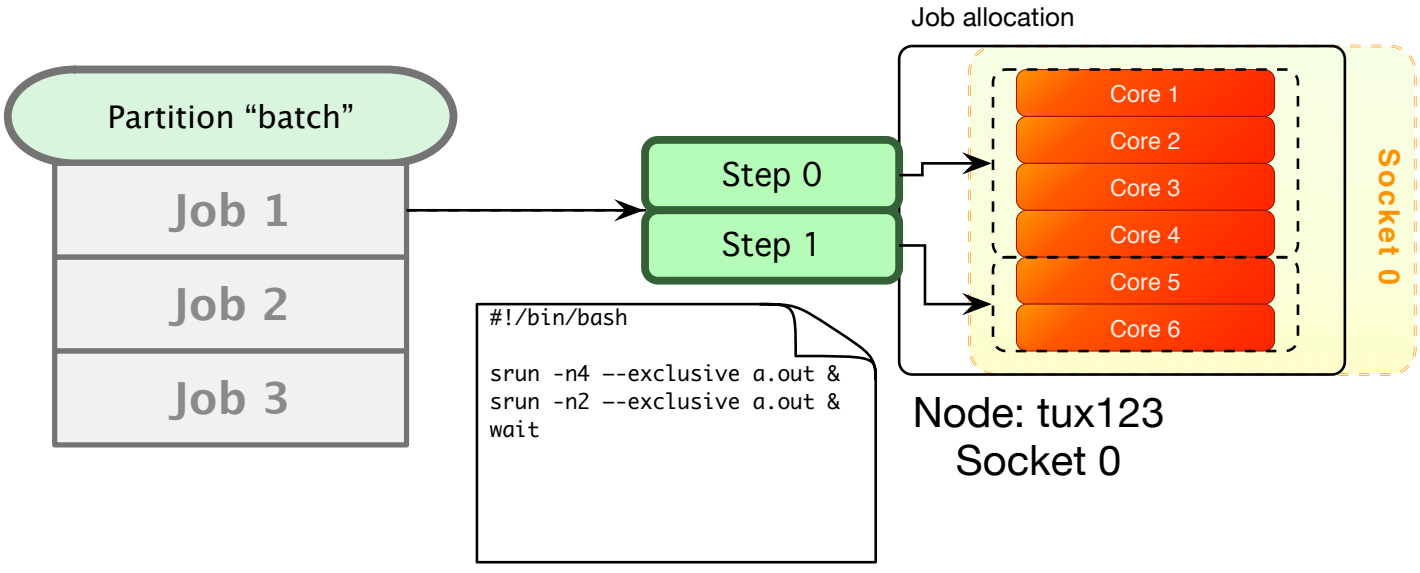
- Users submit jobs to a partition (queue)

SLURM Entities example



- Jobs are allocated resources

SLURM Entities example



- Jobs spawn steps, which are allocated resources from within the job's allocation

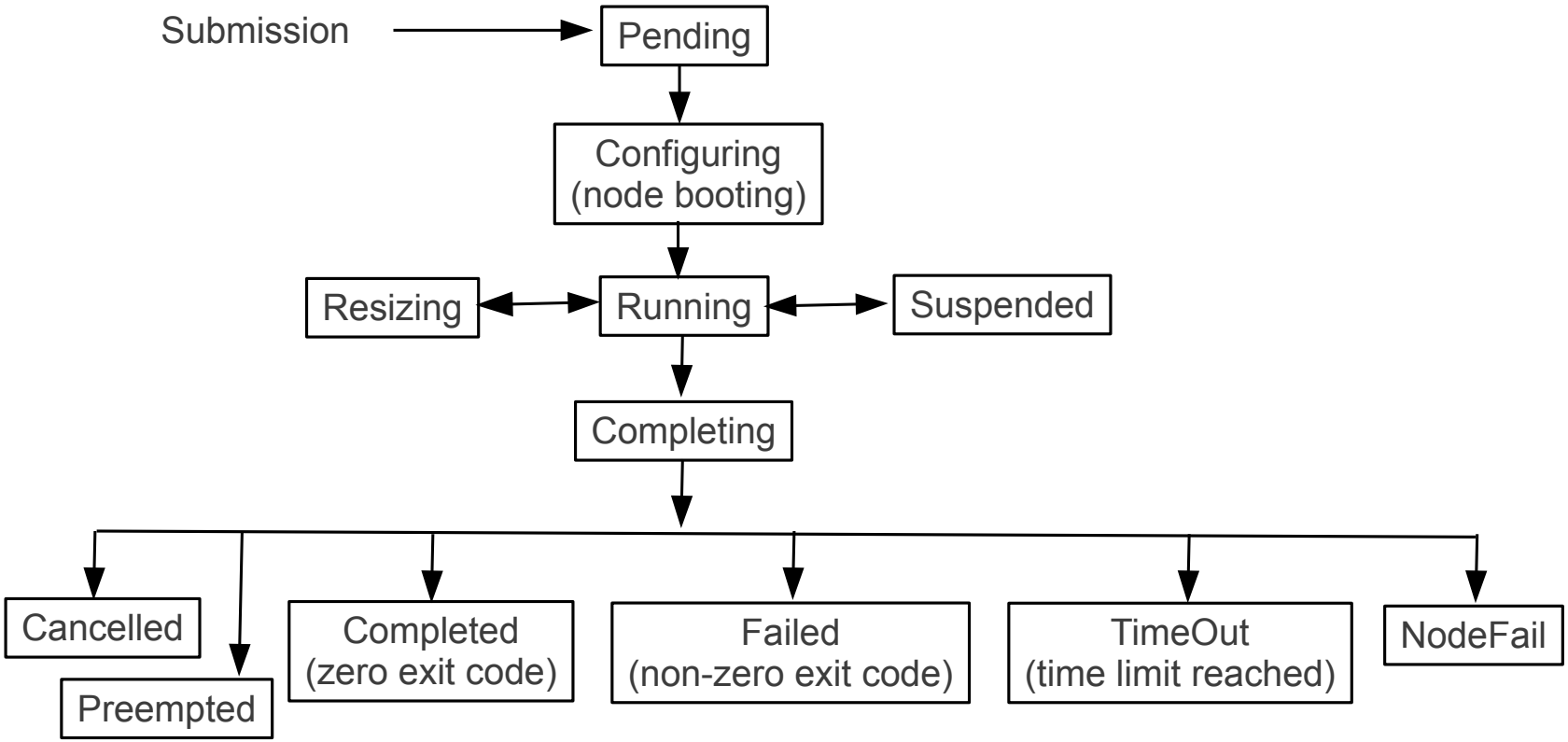
Node State Information

- NUMA boards, Sockets, Cores, Threads
- CPUs
  - ↪ can treat each core or each thread as a CPU for scheduling purposes
- Memory size
- Temporary disk space
- Features (arbitrary string, e.g. OS version)
- Weight (scheduling priority,. . .)
  - ↪ can favor least capable node that satisfies job requirement
- Boot time
- CPU Load
- State (e.g. drain, down, etc.)
- Reason, time and user ID
  - ↪ e.g. “Bad PDU [operator@12:40:10T12/20/2011]”

Queues/Partitions State Information

- Associated with specific set of nodes
  - ↪ Nodes can be in more than one partition (not the case in `iris`)
- Job size and time limits
- Access control list (by Linux group) / QoS
- Preemption rules
- State information (e.g. drain)
- Over-subscription and gang scheduling rules

Job State


Submission
Pending
Configuring
(node booting)
Resizing
Running
Suspended
Completing
Cancelled
Preempted
Completed
(zero exit code)
Failed
(non-zero exit code)
TimeOut
(time limit reached)
NodeFail

Slurm Daemons

- `slurmctld`: Central controller (typically one per cluster)
  - ↪ Optional backup with automatic fail over
  - ↪ Monitors state of resources
  - ↪ Manages job queues and Allocates resources
- `slurmd`: Compute node daemon
  - ↪ typically one per compute node, one or more on front-end nodes
  - ↪ Launches and manages tasks
  - ↪ Small and very light-weight (low memory and CPU use)

- **Common configuration file**: `/etc/slurm/slurm.conf`
  - ↪ Other interesting files: `/etc/slurm/{topology,gres}.conf`

- `slurmdbd`: database daemon (typically one per site)
  - ↪ Collects accounting information
  - ↪ Uploads configuration information (limits, fair-share, etc.)

Slurm in `iris` cluster

- Predefined **Queues/Partitions**:
  - ↪ `batch` (Default) — **Max**: 30 nodes, 5 days walltime
  - ↪ `interactive` — **Max**: 2 nodes, 4h walltime, 10 jobs
  - ↪ `long` — **Max**: 2 nodes, 30 days walltime, 10 jobs
- Corresponding Quality of Service (QOS)
- Possibility to run `besteffort` jobs via the `qos-besteffort` QOS
- Accounts associated to supervisor (multiple associations possible)
- Proper group/user accounting

Slurm Job Management

- 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 user's 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

By default...

- users submit jobs to a particular partition, **and** under a particular account (pre-set per user).

Slurm Commands: General Info

- Man pages available for all commands, daemons and config. files
  - ↪ `--help` option prints brief description of all options
  - ↪ `--usage` option prints a list of the options
  - ↪ `-v | -vv | -vvv`: verbose output
- Commands can be run on any node in the cluster
- Any failure results in a non-zero exit code
- APIs make new tool development easy
  - ↪ Man pages available for all APIs
- Almost all options have two formats
  - ↪ A single letter option (e.g. `-p batch` for partition 'batch')
  - ↪ A verbose option (e.g. `--partition=batch`)
- Time formats: `DD-HH:MM::SS`

User Commands: Job/step Allocation

- `sbatch`: Submit script for later execution (batch mode)
  - ↪ allocate resources (nodes, tasks, partition, etc.)
  - ↪ Launch a script containing `sruns` for series of steps on them.
- `salloc`: Create job allocation & start a shell to use it
  - ↪ allocate resources (nodes, tasks, partition, etc.),
  - ↪ either run a command or start a shell.
  - ↪ Request launch `srun` from shell. (interactive commands within one allocation)
- `srun`: Create a job allocation (if needed) and launch a job step (typically an MPI job)
  - ↪ allocate resources (number of nodes, tasks, partition, constraints, etc.)
  - ↪ launch a job that will execute on them.
- `sattach`: attach to running job for debuggers

User & Admin Commands: System Information

- `sinfo`: Report system status (nodes, partitions etc.)
- `squeue`: display jobs[steps] and their state
- `scancel`: cancel a job or set of jobs.
- `scontrol`: view and/or update system, nodes, job, step, partition or reservation status
- `sstat`: show status of running jobs.
- `sacct`: display accounting information on jobs.
- `sprio`: show factors that comprise a jobs scheduling priority
- `smap`: graphically show information on jobs, nodes, partitions
  - ↪ not available on `iris`

Slurm Admin Commands

- `sacctmgr`: setup accounts, specify limitations on users and groups.
- `sshare`: view sharing information from multifactor plugin.
- `sreport`: display information from accounting database on jobs, users, clusters.
- `sview`: graphical view of cluster. Display and change characteristics of jobs, nodes, partitions.
  - ↪ not yet available on `iris` cluster
- `strigger`: show, set, clear event triggers. Events are usually system events such as an equipement failure.

Slurm vs. OAR Main Commands

Action	SLURM command	OAR Command
Submit passive/batch job	`sbatch [...] $script`	`oarsub [...] $script`
Start interactive job	`srun [...] --pty bash`	`oarsub -I [...]`
Queue status	`squeue`	`oarstat`
User job status	`squeue -u $user`	`oarstat -u $user`
Specific job status (detailed)	`scontrol show job $jobid`	`oarstat -f -j $jobid`
Job accounting status (detailed)	`sacct --job $jobid -l`	
Delete (running/waiting) job	`scancel $jobid`	`oardel $jobid`
Hold job	`scontrol hold $jobid`	`oarhold $jobid`
Resume held job	`scontrol release $jobid`	`oarresume $jobid`
Node list and their properties	`scontrol show nodes`	`oarnodes`

Job Specifications

Specification	SLURM	OAR
Script directive	`#SBATCH`	`#OAR`
<n> **Nodes** request	`-N <n>`	`-l nodes=<n>`
<n> **Cores/Tasks** request	`-n <n>`	`-l core=<n>`
<c> Cores-per-node request	`--ntasks-per-node=<c>`	`-l nodes=<n>/core=<c>`
<c> Cores-per-task request (multithreading)	`-c=<c>`	
<m>**GB memory per node** request	`--mem=<m>GB`	
Walltime request	`-t <mm>/<days-hh[:mm:ss]>`	`-l walltime=hh[:mm:ss]`
Job array	`--array <specification>`	`--array <count>`
Job name	`-J <name>`	`-n <name>`
Job dependency	`-d <specification>`	`-a <jobid>`
Property request	`-C <specification>`	`-p "<property>=<value>"`
Specify job partition/queue	`-p <partition>`	`-t <queue>`
Specify job qos	`--qos <qos>`	
Specify account	`-A <account>`	
Specify email address	`--mail-user=<email>`	`--notify "mail:<email>"`

Typical Workflow

```
# Run an interactive job     -- make an alias 'si [...]'
$> srun -p interactive --qos qos-interactive --pty bash
# Ex: interactive job for 30 minutes, with 2 nodes/4 tasks per node
$> si --time=0:30:0 -N 2 --ntasks-per-node=4
# Run a [passive] batch job -- make an alias 'sb [...]'
$> sbatch -p batch     --qos qos-batch       /path/to/launcher.sh
# Will create (by default) slurm-<jobid>.out file
```

Environment variable	SLURM	OAR
Job ID	$SLURM_JOB_ID	$OAR_JOB_ID
Resource list	$SLURM_NODELIST **#List not file!**	$OAR_NODEFILE
Job name	$SLURM_JOB_NAME	$OAR_JOB_NAME
Submitting user name	$SLURM_JOB_USER	$OAR_USER
Task ID within job array	$SLURM_ARRAY_TASK_ID	$OAR_ARRAY_INDEX
Working directory at submission	$SLURM_SUBMIT_DIR	$OAR_WORKING_DIRECTORY
Number of nodes assigned to the job	$SLURM_NNODES	
Number of tasks of the job	$SLURM_NTASKS	$(wc -l ${OAR_NODEFILE})

- **Note**: create the equivalent of $OAR_NODEFILE in Slurm:
  - ↪ `srun hostname | sort -n > hostfile`

Available Node partitions

- Slurm Command Option `-p, --partition=<partition>`
  - ↪ **Ex**: `{srun,sbatch} -p batch [...]`
- Date format: `-t <minutes>` or `-t <D>-<H>:<M>:<S>`

Partition	#Nodes	Default time	Max time	Max nodes/user
`batch`	80%	0-2:0:0 [2h]	5-0:0:0 [5d]	unlimited
`interactive`	10%	0-1:0:0 [1h]	0-4:0:0 [4h]	2
`long`	10%	0-2:0:0 [2h]	30-0:0:0 [30d]	2

Quality of Service (QOS)

- Slurm Command Option `--qos=<qos>`
- There is **no default** QOS (due to the selected scheduling model)
  - ↪ you **MUST** provide upon **any job submission**
  - ↪ a default qos is guessed from the partition **i.e.** `qos-<partition>`

QoS	User group	Max cores	Max jobs/user	Description
qos-besteffort	**ALL**	**no limit**		Preemptible jobs, requeued on preemption
qos-batch	**ALL**	**1064**	100	Normal usage of the `batch` partition
qos-interactive	**ALL**	**224**	10	Normal usage of the `interactive` partition
qos-long	**ALL**	**224**	10	Normal usage of the `long` partiton
qos-batch-###	rsvd	rsvd	100	Reserved usage of the `batch` partition
qos-interactive-###	rsvd	rsvd	10	Reserved usage of the `interactive` partition
qos-long-###	rsvd	rsvd	10	Reserved usage of the `long` partiton

Accounts

- Every user job runs under a group account
  - ↪ granting access to specific QOS levels.

Account	Parent Account
UL	
FSTC	UL
FDEF	UL
FLSHASE	UL
LCSB	UL
SNT	UL
Professor $X	*FACULTY* /IC
Group head $G	*FACULTY* /IC
Researcher $R	Professor $X
Researcher $R	Group head $G
Student $S	Professor $X
Student $S	Group head $G
External collaborator $E	Professor $X
External collaborator $E	Group head $G

```
$> sacctmgr list associations where users=$USER \
          format=Account%30s,User,Partition,QOS
```

Other Features

- **Checkpoint / Restart**
  - ↪ Based on DMTCP: Distributed MultiThreaded CheckPointing
  - ↪ see the official DMTCP launchers
  - ↪ ULHPC example
- Many metrics can be extracted from user jobs
  - ↪ with SLURM's own tools (sacct/sstat)
  - ↪ within the jobs with e.g. PAPI
  - ↪ easy to bind executions with Allinea Performance Report
- Advanced admission rules
  - ↪ to simplify CLI
- Container Shifter / Singularity
  - ↪ Work in progress, not yet available on `iris`

Simple Example of usage

```
# Simple interactive job
(access)$> srun -p interactive [--qos qos-interactive] --pty bash
(node)$> echo $SLURM_NTASKS
1
(node)$> echo $SLURM_JOBID
59900
```

Simple Example of usage

```
# Simple interactive job
(access)$> srun -p interactive [--qos qos-interactive] --pty bash
(node)$> echo $SLURM_NTASKS
1
(node)$> echo $SLURM_JOBID
59900
```

```
$> squeue -u $USER -l    # OR 'sq'
```

Simple Example of usage

```
# Simple interactive job
(access)$> srun -p interactive [--qos qos-interactive] --pty bash
(node)$> echo $SLURM_NTASKS
1
(node)$> echo $SLURM_JOBID
59900
```

```
$> squeue -u $USER -l    # OR 'sq'
```

- Many metrics during (`scontrol`)/after job execution (`sacct`)
  - ↪ including energy (J) – but with caveats
  - ↪ job **steps** counted individually
  - ↪ enabling advanced application debugging and optimization
- Job information available in easily parseable format (add -p/-P)

Live Job Statistics

```
$> scontrol show job 59900
JobId=59900 JobName=bash
   UserId=<login>(<uid>) GroupId=clusterusers(666) MCS_label=N/A
   Priority=6627 Nice=0 Account=ulhpc QOS=qos-interactive
   JobState=RUNNING Reason=None Dependency=(null)
   RunTime=00:04:19 TimeLimit=01:00:00 TimeMin=N/A
   SubmitTime=2017-10-22T23:07:02 EligibleTime=2017-10-22T23:07:02
   StartTime=2017-10-22T23:07:02 EndTime=2017-10-23T00:07:02 Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=interactive AllocNode:Sid=access1:72734
   [...]
   NodeList=iris-002
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,mem=4G,node=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryCPU=4G MinTmpDiskNode=0
   [...]
   Command=bash
   WorkDir=/mnt/irisgpfs/users/<login>
```

Node/Job Statistics

```
$> sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
interactive    up    4:00:00     10   idle iris-[001-010]
long           up 30-00:00:0      2   resv iris-[019-020]
long           up 30-00:00:0      8   idle iris-[011-018]
batch*         up 5-00:00:00      5    mix iris-[055,060-062,101]
batch*         up 5-00:00:00     13  alloc iris-[053-054,056-059,102-108]
batch*         up 5-00:00:00     70   idle iris-[021-052,063-100]
```

Node/Job Statistics

```
$> sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
interactive    up    4:00:00     10   idle iris-[001-010]
long           up 30-00:00:0      2   resv iris-[019-020]
long           up 30-00:00:0      8   idle iris-[011-018]
batch*         up 5-00:00:00      5    mix iris-[055,060-062,101]
batch*         up 5-00:00:00     13  alloc iris-[053-054,056-059,102-108]
batch*         up 5-00:00:00     70   idle iris-[021-052,063-100]
```

```
$> sacct --format=account,user,jobid,jobname,partition,state -j <JOBID>

$> sacct --format=elapsed,elapsedraw,start,end -j <JOBID>

$> sacct -j <JOBID> --format=maxrss,maxvmsize,consumedenergy,\
                consumedenergyraw,nodelist
```

Playing with hostname and task ID label

```
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

```
# -n: #tasks
$> srun -n 4 -l hostname
1: iris-055
2: iris-055
3: iris-055
0: iris-055
```

Playing with hostname and task ID label

```
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

```
# -n: #tasks
$> srun -n 4 -l hostname
1: iris-055
2: iris-055
3: iris-055
0: iris-055
```

```
# -N: #nodes
$> srun -N 4 -l hostname
3: iris-058
2: iris-057
1: iris-056
0: iris-055
```

Playing with hostname and task ID label

```
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

```
# -n: #tasks
$> srun -n 4 -l hostname
1: iris-055
2: iris-055
3: iris-055
0: iris-055
```

```
# -N: #nodes
$> srun -N 4 -l hostname
3: iris-058
2: iris-057
1: iris-056
0: iris-055
```

```
# -c: #cpus/task ~#thread/task
$> srun -c 4 -l hostname
0: iris-055
```

Playing with hostname and task ID label

```
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

```
# -n: #tasks
$> srun -n 4 -l hostname
1: iris-055
2: iris-055
3: iris-055
0: iris-055
```

```
# -N: #nodes
$> srun -N 4 -l hostname
3: iris-058
2: iris-057
1: iris-056
0: iris-055
```

```
# -c: #cpus/task ~#thread/task
$> srun -c 4 -l hostname
0: iris-055
```

```
$> srun -N 2 -n 4 -l hostname
3: iris-056
0: iris-055
1: iris-055
2: iris-055
$> srun -N 2 --ntasks-per-node 2 -l hostname
3: iris-056
2: iris-056
1: iris-055
0: iris-055
```

Job submission with salloc

```
$> salloc [-N #node] [-n #task] [--ntasks-per-node #n]
```

```
$> salloc -N 4
salloc: Granted job allocation 59955
salloc: Waiting for resource configuration
salloc: Nodes iris-[055,060-062] are ready for job
$> env | grep SLURM
$> hostname
access1.iris-cluster.uni.lux
$> srun -l hostname
0: iris-055
2: iris-061
1: iris-060
3: iris-062
```

Reservations and scontrol features

```
$> scontrol show job <JOBID>                    # Job info
```

Reservations and scontrol features

```
$> scontrol show job <JOBID>                    # Job info
```

```
$> scontrol show {partition,topology}
```

Reservations and scontrol features

```
$> scontrol show job <JOBID>                    # Job info
```

```
$> scontrol show {partition,topology}
```

```
$> scontrol show reservations      # Show existing reservations
```

```
$> scontrol create reservation
ReservationName=<name>
accounts=<account_list>                          licenses=<license>
corecnt=<num>                                    nodecnt=<count>
duration=[days-]hours:minutes:seconds            nodes=<node_list>
endtime=yyyy-mm-dd[thh:mm[:ss]]                  partitionname=<partition(s)>
features=<feature_list>                          starttime=yyyy-mm-dd[thh:mm[:s
flags=maint,overlap,ignore_jobs,daily,weekly     users=<user_list>
```

Basic Slurm Launcher Examples

Documentation

https://hpc.uni.lu/users/docs/slurm_launchers.html

See also **PS1**, **PS2** and **PS3**

```
#!/bin/bash -l
# Request one core for 5 minutes in the batch queue

#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=0-00:05:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

[...]
```

Basic Slurm Launcher Examples (cont.)

```
#!/bin/bash -l
# Request two cores on each of two nodes for 3 hours

#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH --time=0-03:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"

[...]
```

Basic Slurm Launcher Examples (cont.)

```
#!/bin/bash -l
# Request one core and half the memory available on an iris cluster
# node for one day
#
#SBATCH -J MyLargeMemorySequentialJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=64GB
#SBATCH --time=1-00:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
```

pthreads/OpenMP Slurm Launcher

```
#!/bin/bash -l
# Single node, threaded (pthreads/OpenMP) application launcher,
# using all 28 cores of an iris cluster node:

#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
/path/to/your/threaded.app
```

MATLAB Slurm Launcher

```
#!/bin/bash -l
# Single node, multi-core parallel application (MATLAB, Python, R...)
# launcher, using all 28 cores of an iris cluster node:

#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load base/MATLAB
matlab -nodisplay -nosplash < /path/to/inputfile > /path/to/outputfile
```

Intel MPI Slurm Launchers

- Official SLURM guide for Intel MPI

```
#!/bin/bash -l
# Multi-node parallel application IntelMPI launcher,
# using 128 distributed cores:

#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun -n $SLURM_NTASKS /path/to/your/intel-toolchain-compiled-application
```

OpenMPI Slurm Launchers

- Official SLURM guide for Open MPI

```
#!/bin/bash -l
# Multi-node parallel application openMPI launcher,
# using 128 distributed cores:

#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/foss
mpirun -n $SLURM_NTASKS /path/to/your/foss-toolchain-compiled-application
```

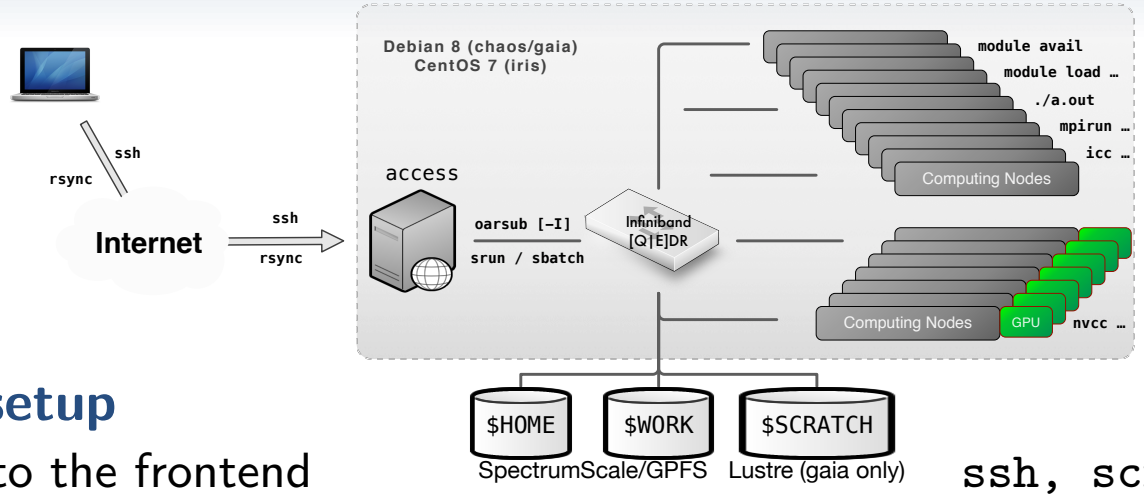
Hybrid IntelMPI+OpenMP Launcher

```
#!/bin/bash -l
# Multi-node hybrid application IntelMPI+OpenMP launcher,
# using 28 threads per node on 10 nodes (280 cores):

#SBATCH -N 10
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun -n $SLURM_NTASKS /path/to/your/parallel-hybrid-app
```

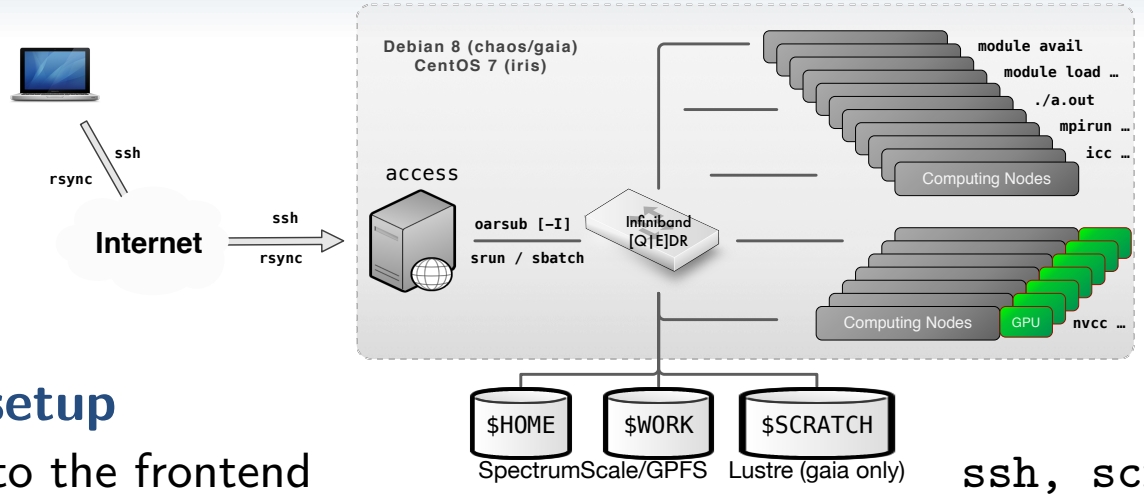
Typical Workflow on UL HPC resources



- **Preliminary setup**
  1. Connect to the frontend — `ssh, screen`
  2. Synchronize you code — `scp/rsync/svn/git`
  3. Reserve a few interactive resources — `oarsub -I [...]`

     or, on iris: `srun -p interactive [...]`
     - ✓ *(eventually)* build your program — `gcc/icc/mpicc/nvcc..`
     - ✓ Test on small size problem — `mpirun/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 — `oarsub -I [...]`

     or, on iris: `srun -p interactive [...]`
     - ✓ *(eventually)* build your program — `gcc/icc/mpicc/nvcc..`
     - ✓ Test on small size problem — `mpirun/srun/python/sh...`
     - ✓ Prepare a launcher script — `<launcher>.{sh|py}`
- **Real Experiment**
  1. Reserve passive resources — `oarsub [...] <launcher>`

     or, on iris: `sbatch -p {batch|long} [...] <launcher>`
  2. Grab the results — `scp/rsync/svn/git ...`