Getting Started with Batch Job Scheduling: Slurm Edition

COMPLECS Team

https://bit.ly/COMPLECS
https://github.com/sdsc-complecs

SDSC
SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

About COMPLECS

COMPLECS (COMPrehensive Learning for end-users to Effectively utilize CyberinfraStructure) is a new SDSC program where training will cover ***non-programming*** skills needed to effectively use supercomputers. Topics include parallel computing concepts, Linux tools and bash scripting, security, batch and interactive computing, how to get help, and data management.

COMPLECS is supported by NSF CISE/OAC-2320934

COMPLECS: Batch Computing Series

Working with the Linux Scheduler

How to interact with the Linux scheduler and run your research workloads on your personal computer, a shared workstation, or even a high-performance computing system.

Getting Started with Batch Job Scheduling: Slurm Edition

How to schedule your batch jobs on high-performance computing systems using the Slurm Workload Manager.

High-Throughput and Many-Task Computing Workflows: Slurm Edition

How to build and run your high-throughput and many-task computing workflows on high-performance computing systems using the Slurm Workload Manager.

Getting Started with Batch Job Scheduling: Slurm Edition

Learning Goals:

- What is a batch job?
- What is Slurm?
- How to start interacting with Slurm
- How to write, submit and run your first batch job on a high-performance computing system

Prerequisite Knowledge:

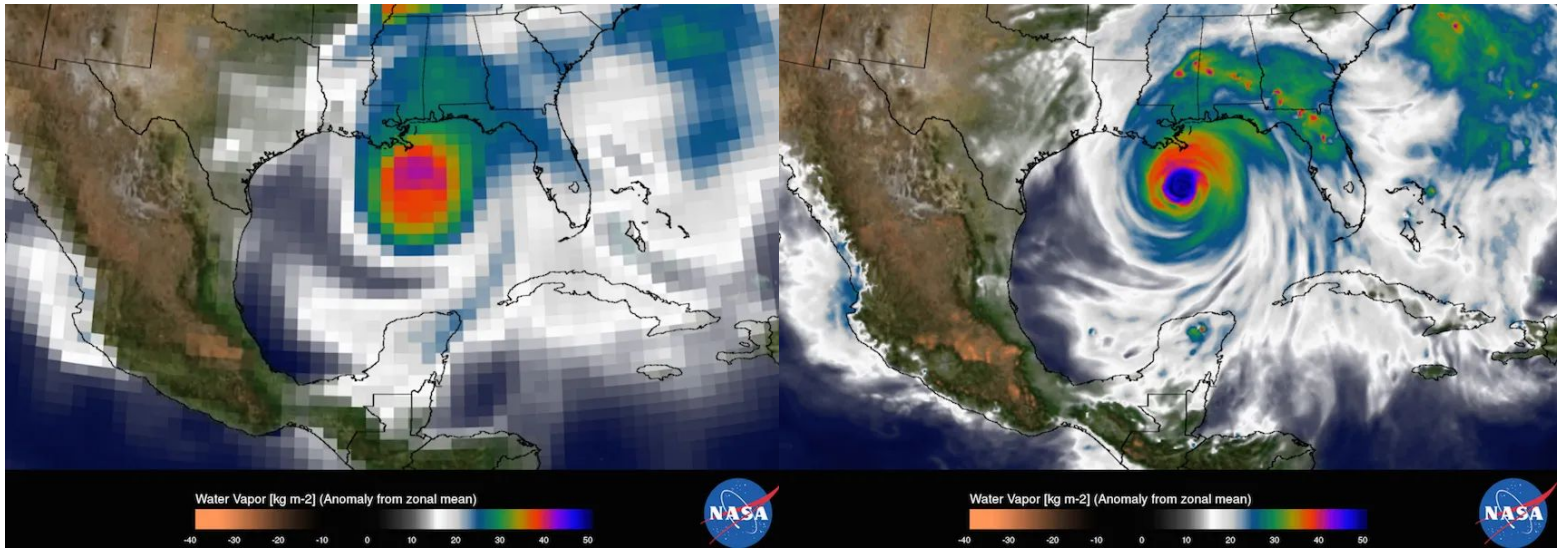
- Parallel Computing Concepts
- Intermediate Linux and Shell Scripting

Table of Contents

High-Performance Computing (HPC)

High-performance computing (HPC) uses supercomputers and computer clusters to solve advanced computation problems.

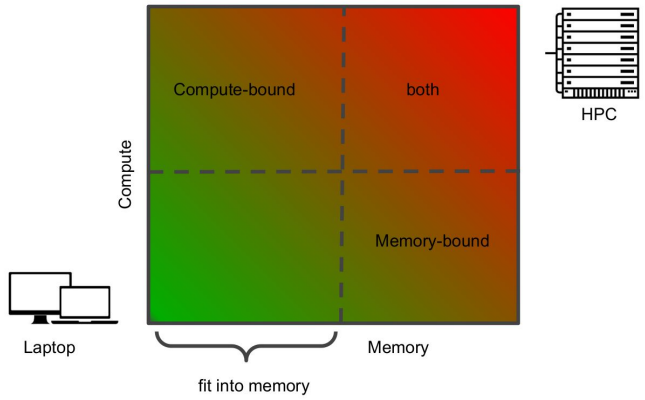
https://svs.gsfc.nasa.gov/11925

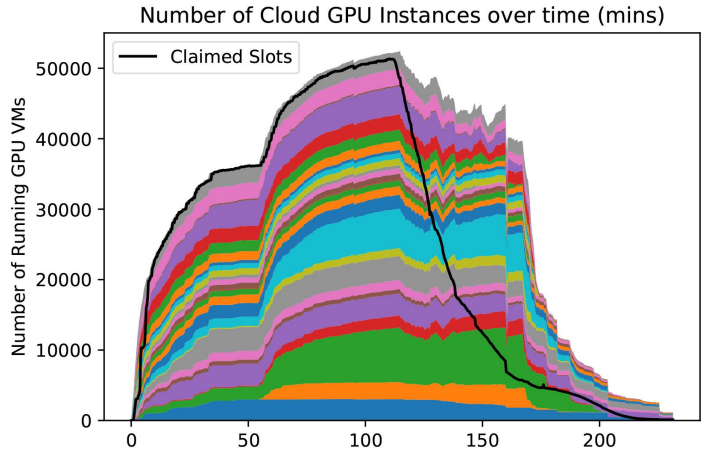


https://www.youtube.com/watch?v=p-3aB9hJ8Hc&t=11s

Hurricane Katrina @ 50-km (2005) resolution

Hurricane Katrina @ 6.25-km (2015) resolution



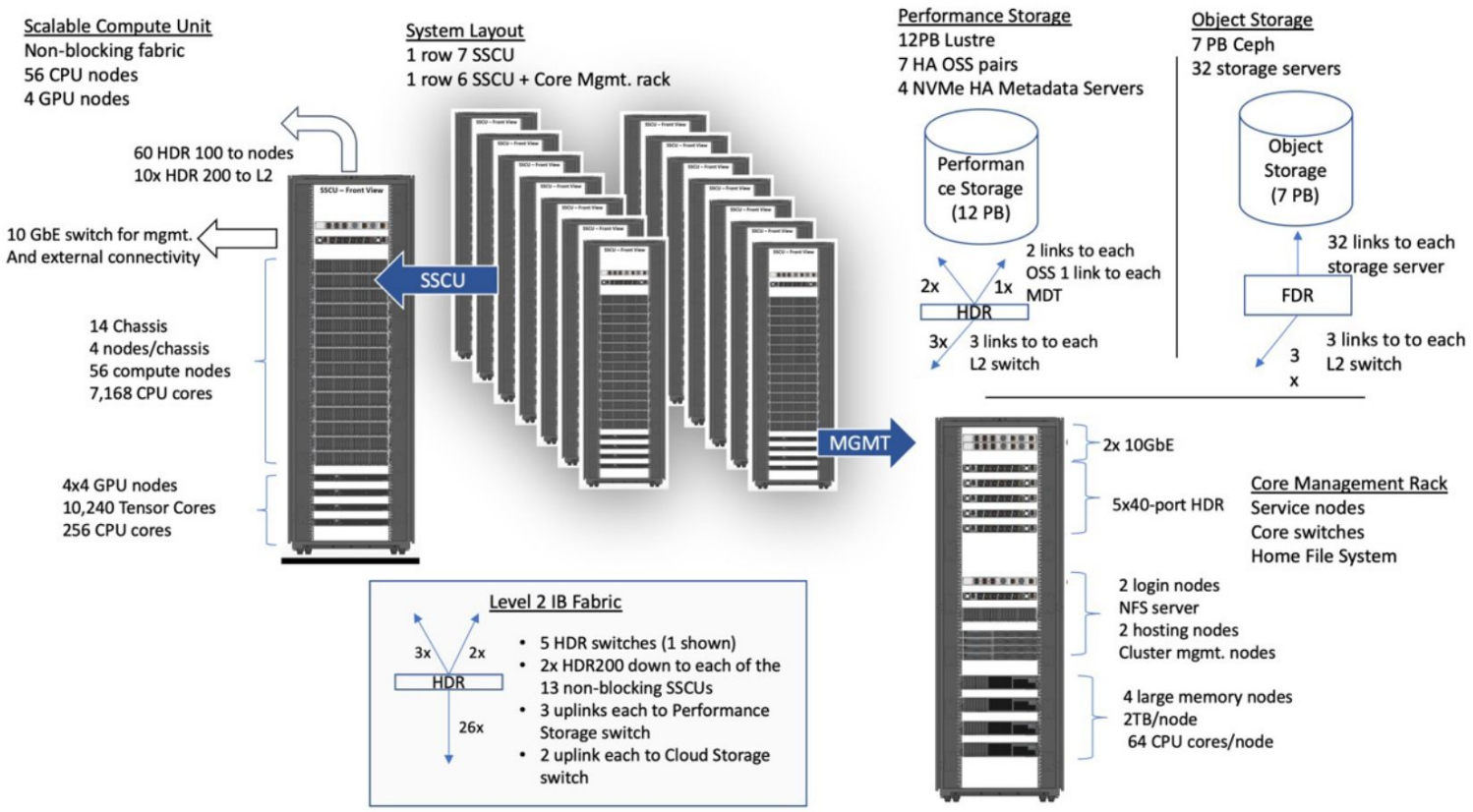


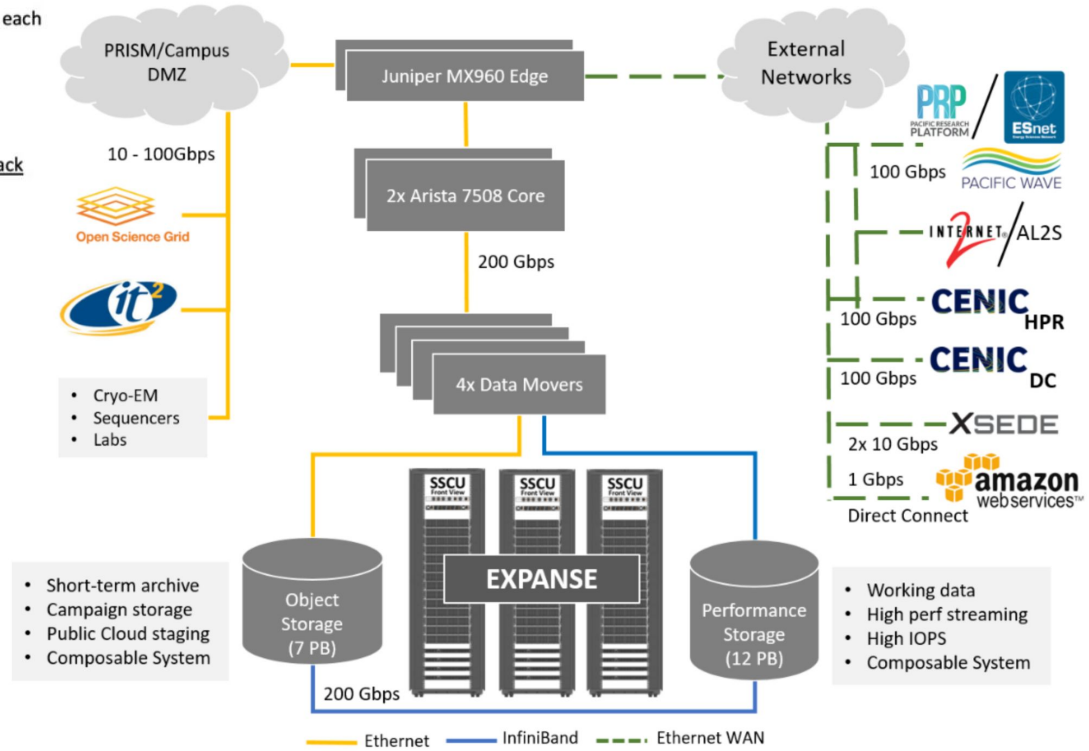
IceCube Neutrino Observatory, SDSC, WIPAC: GPU Cloud Burst Experiment

Key advantages of leveraging HPC for your research:

- **Speed** - Solve a problem more quickly
- **Scale** - Solve a larger, more complex problem
- **Throughput** - Solve many (simple) problems more quickly

HPC System Architecture: Expanse @ SDSC





https://access-ci.org

https://expanse.sdsc.edu

https://www.youtube.com/watch?v=uNZyg6X_t3s

HPC Compute Node: Dell C4140

Figure 1. Front view of the system

1. Control panel

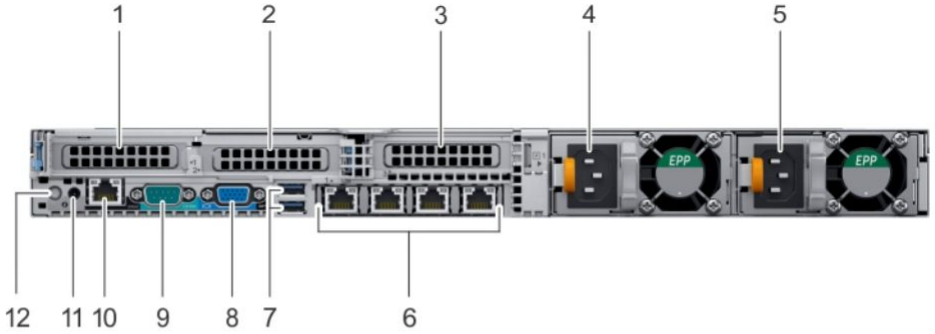


Figure 2. Rear view of the system

1. PCIe expansion card slot 1
2. PCIe expansion card slot 2

   NOTE: This slot is dedicated for BOSS c
3. PCIe expansion card slot 3
4. Power supply unit (PSU 1)
5. Power supply unit (PSU 2)
6. Ethernet connectors (4)
7. USB connectors (2)
8. Video connector
9. Serial connector
10. iDRAC Enterprise port
11. NMI button
12. System identification button

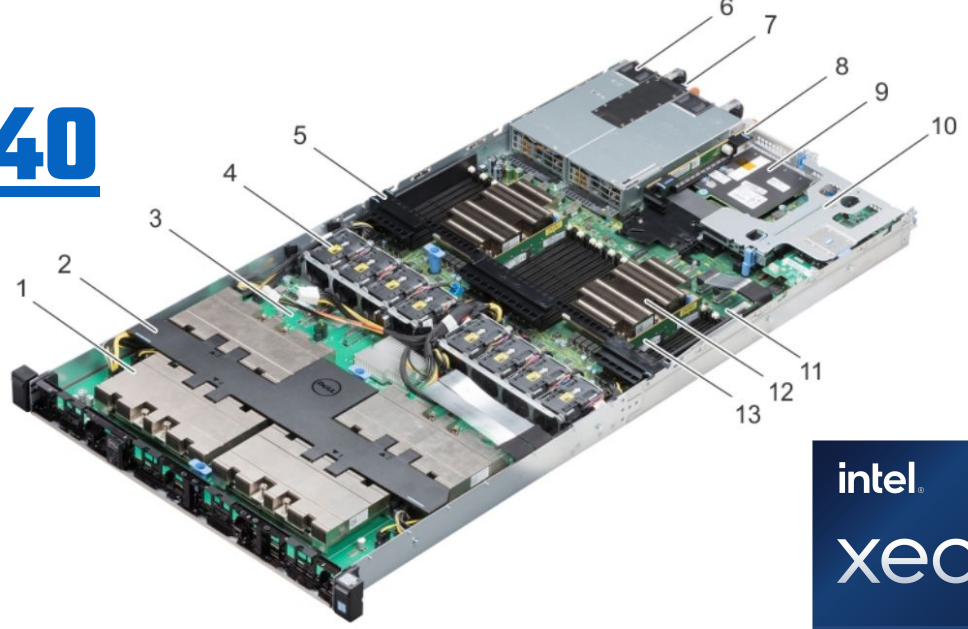
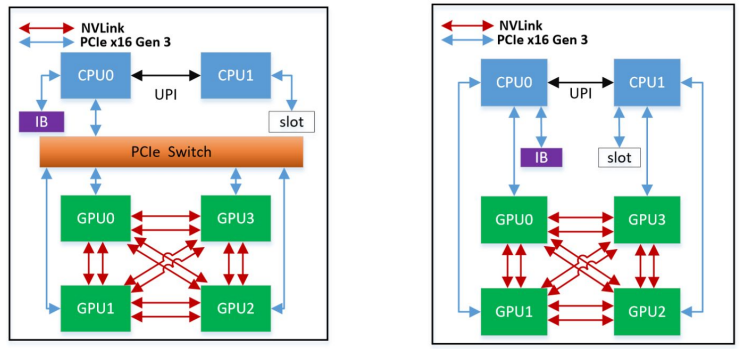


Figure 4. Configuration K

1. NVLink Heat sink and processor (4)
2. NVLink air shroud
3. NVLink board
4. Cooling fan (8)
5. Air shroud
6. PSU (2)
7. Information tag
8. Riser 2A (Low-profile PCIe expansion card: Slot 3)
9. Network daughter card (NDC)
10. Riser 1A (Low-profile PCIe expansion cards: Slots 1 and 2)
11. System board
12. Processor and heat sink (2)
13. DIMMs (24)

Intel Xeon Gold 6248

NVIDIA V100 32GB SMX2



(a) Configuration K (b) Configuration M

Figure 1: The comparison between configuration K and configuration M in C4140 server

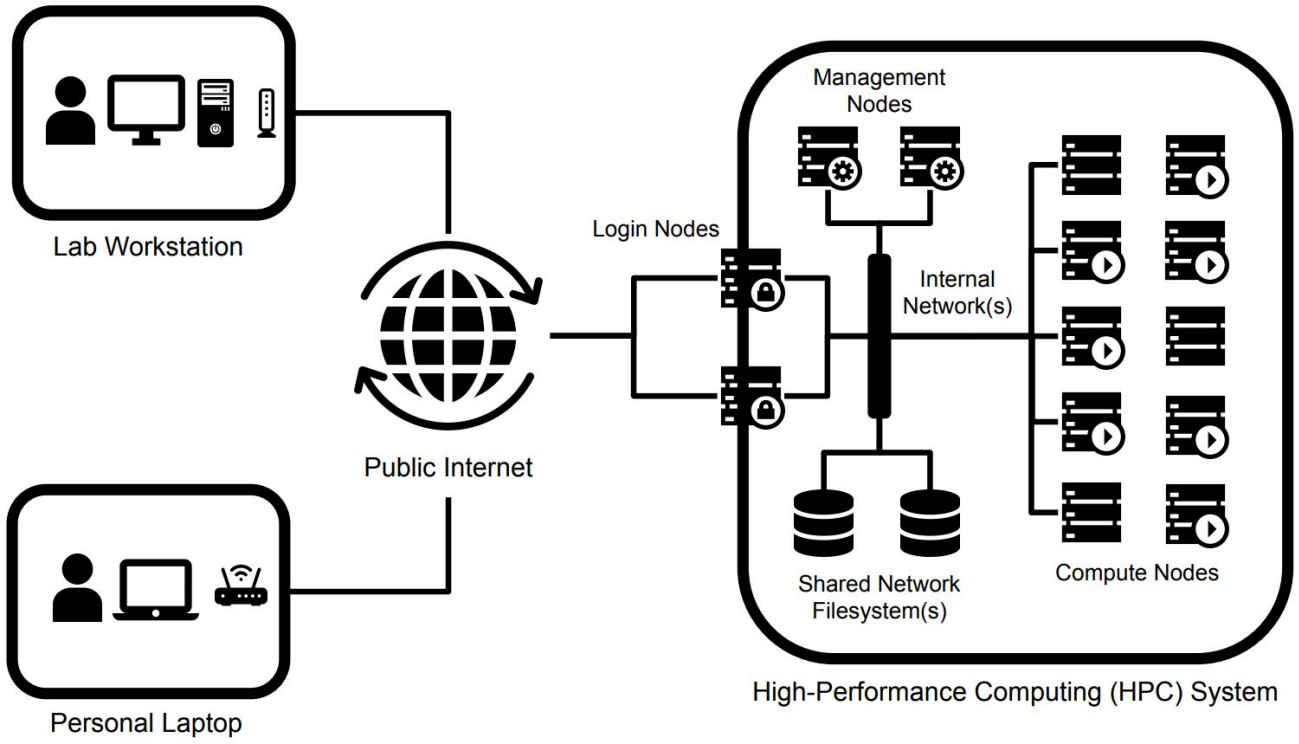
HPC System Architecture: Conceptual Model

Login node(s): Provide remote access to an HPC system; use only for simple tasks such as editing files, limited data transfers to and from the system, and batch job submission

Compute nodes: Run computational workloads: simulations, data analysis and visualization

Internal Network(s): Provide high-bandwidth, low-latency communication between computes nodes ; access to shared (parallel) filesystems; system management

Shared Network Filesystem(s): Provide input/output (I/O) access to data storage systems from any compute node



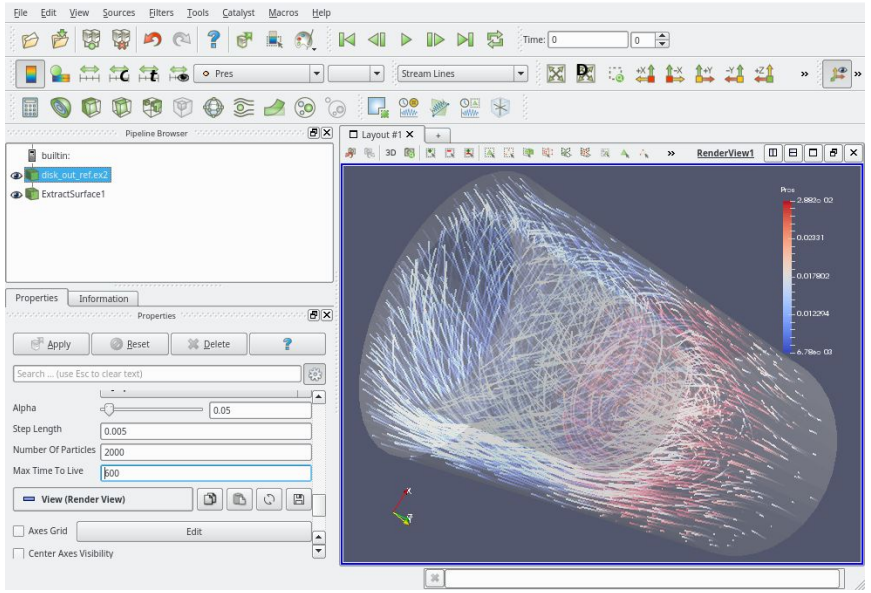
Management node(s): Run core system services such as cluster management software, system monitoring software, *batch job scheduler*, etc

Table of Contents

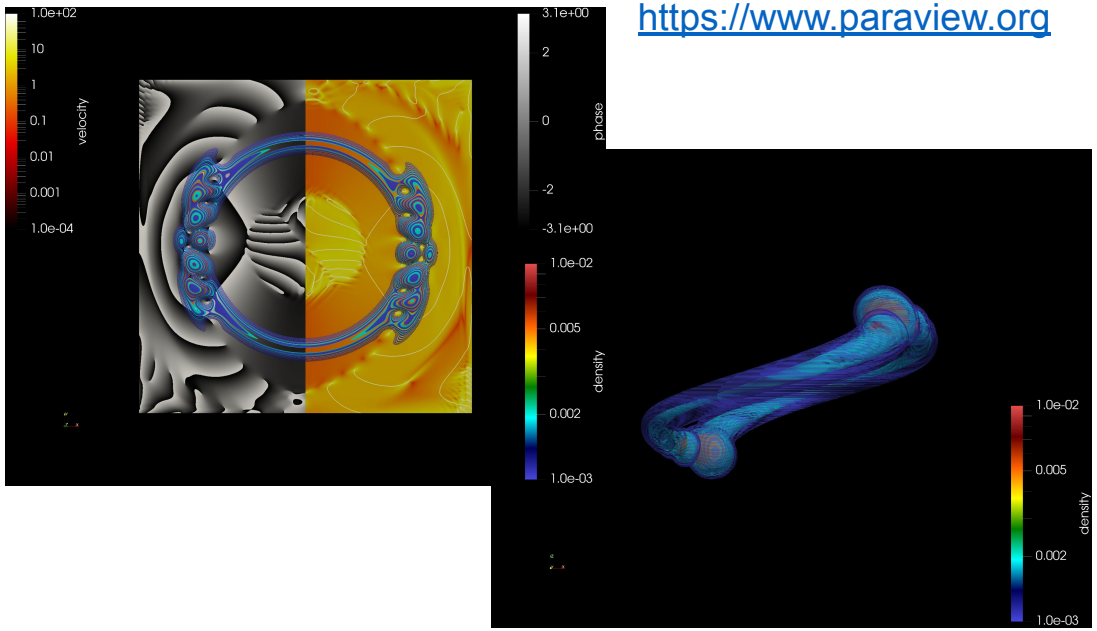
Interactive vs. Batch Computing

Interactive computing is the use of computer programs that accept input from the user as they execute. Examples in HPC include the use of software applications for code development and performance optimization as well as real-time data exploration, analysis and visualization.

Batch processesing is the execution of a series of computer programs, known as jobs, on a system without user interaction. Batch computing systems are widely used to automate high-volume, repetitive tasks such as data processing and to allow the shared use of advanced compute resources like an HPC system among many users.



https://www.paraview.org

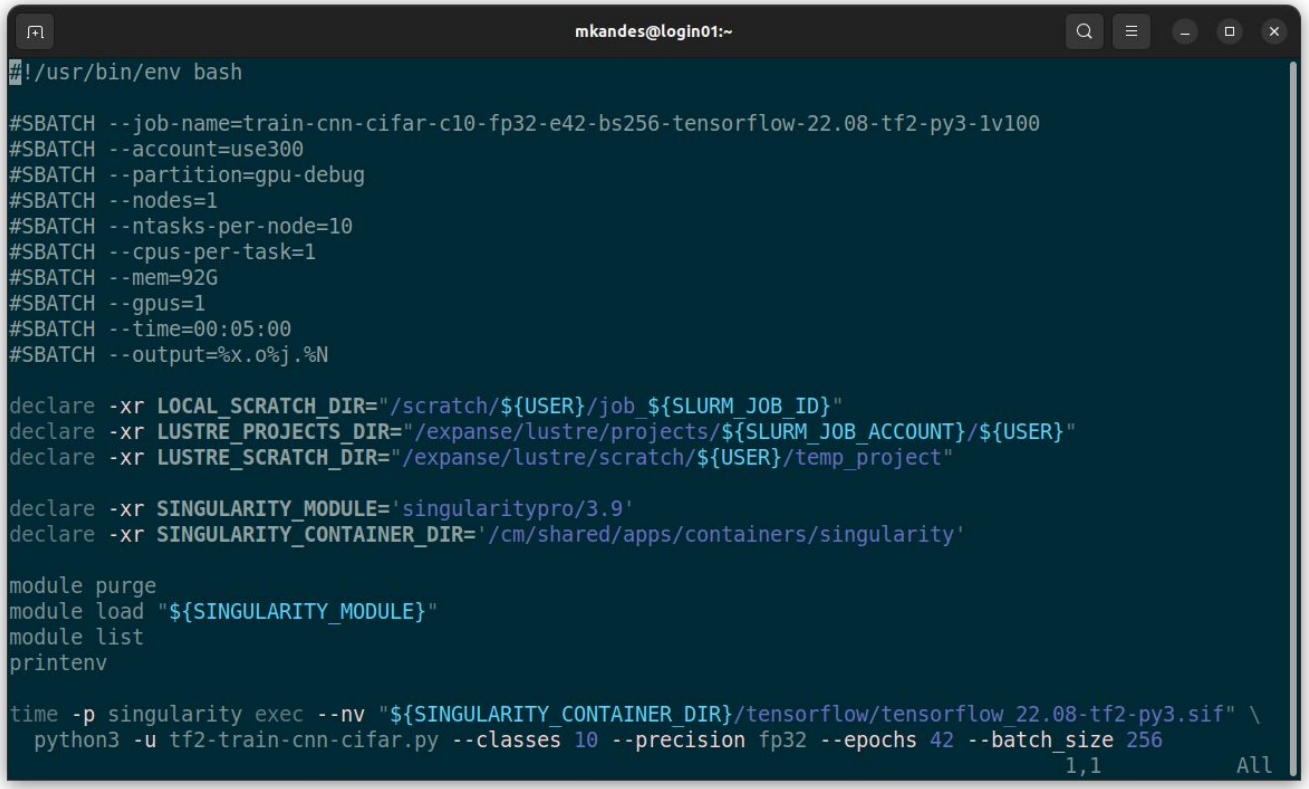


What is a Batch Job?

A **batch job** is a pre-*scripted* sets of commands that are executed as processes, which may or may not be multi-threaded, on a certain type or set of dedicated compute resources within a computing system for a given amount of time without user interaction.

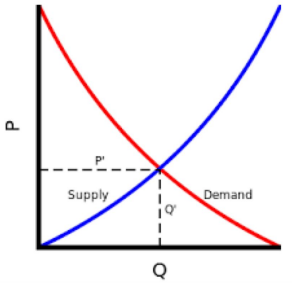
On HPC systems, batch jobs may run large-scale simulations, perform complex data analysis, or any other types of demanding computational tasks.

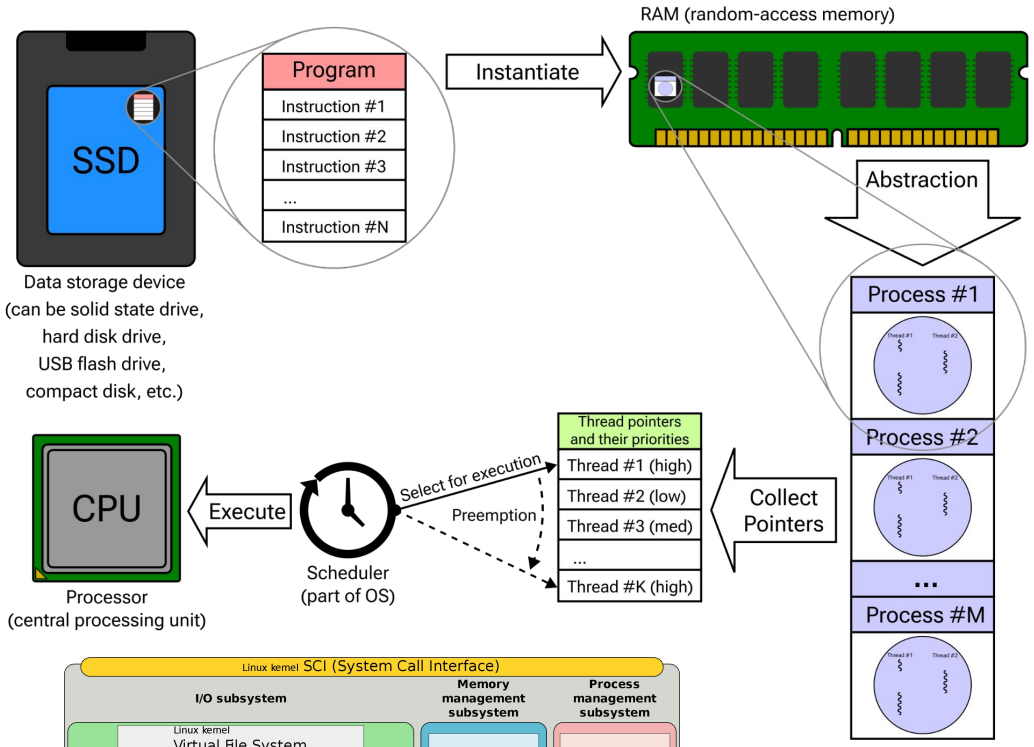
A batch job is specified as a standard shell script that is then submitted to a queue of other batch jobs managed by a scheduler, which assigns the job to available resources and ensures that the job runs to completion.

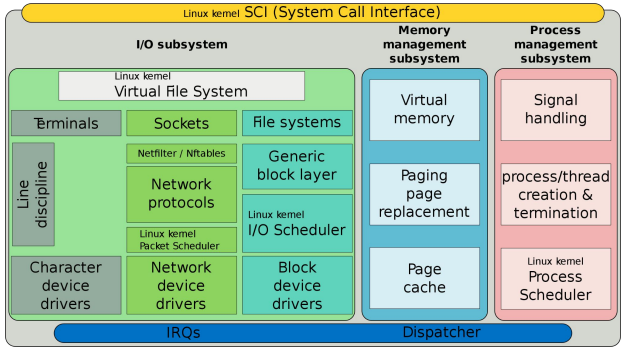


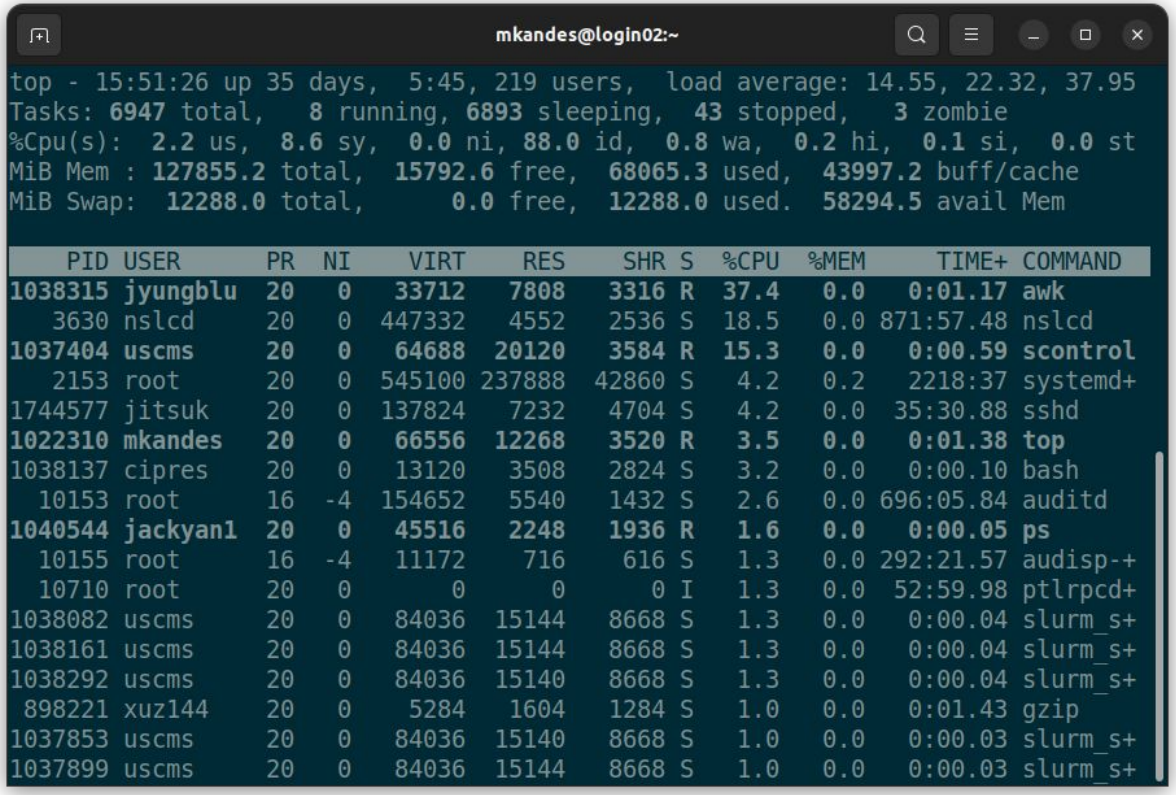
Scheduling

Scheduling is the action of assigning resources to perform tasks.





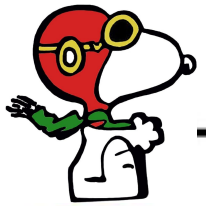




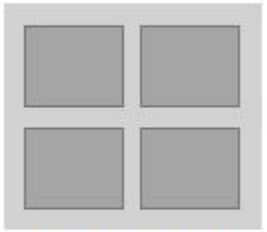
top and htop are useful interactive tools for inspecting and monitoring running processes on Unix-like operating systems.

What is a Batch Job Scheduler?

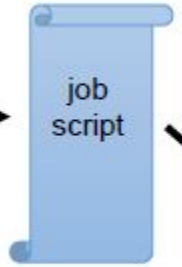
A **batch job scheduler** is software that controls and tracks *where* and *when* batch jobs submitted to a computing system will eventually run on its compute resources.

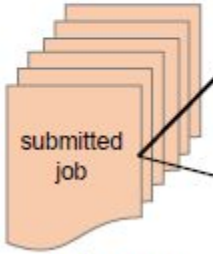


user

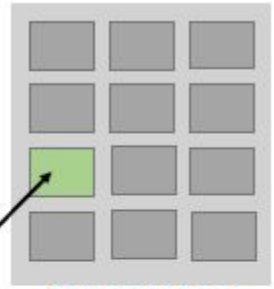


login / front-end nodes

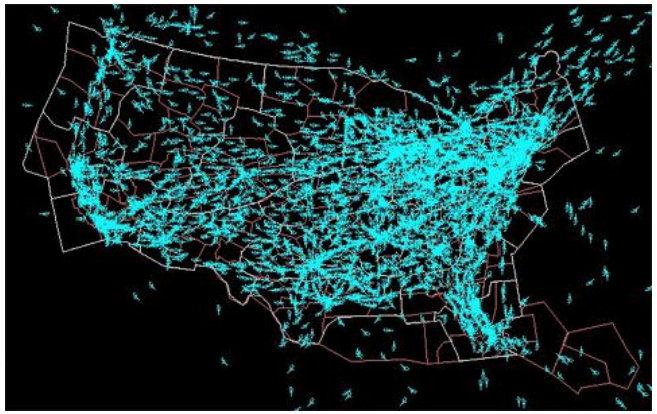




job queue

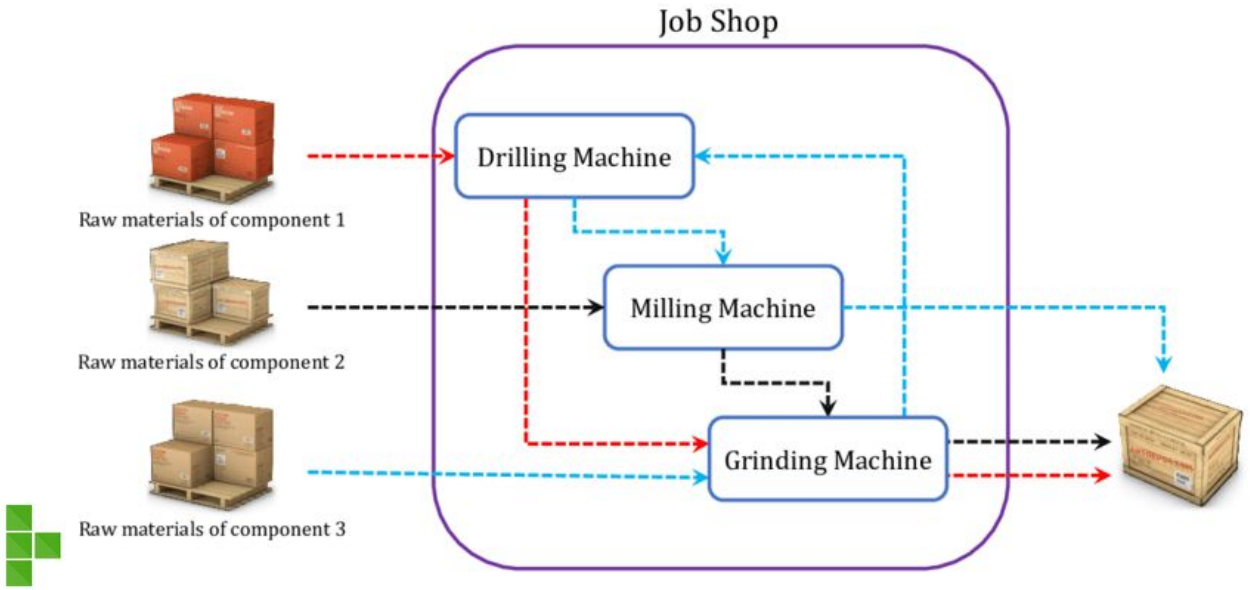


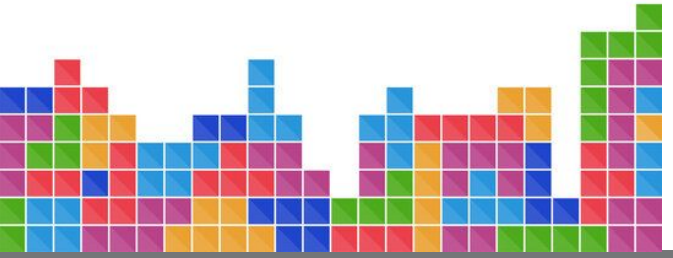
batch system



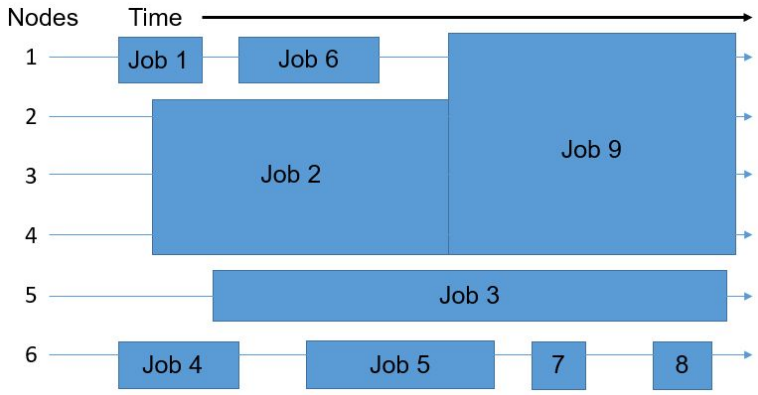
Job-Shop Scheduling Problem

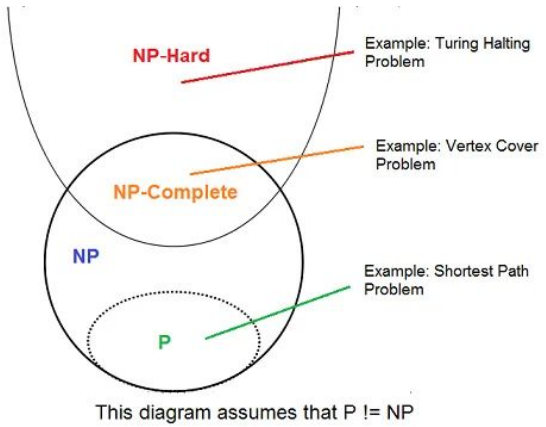
In the general job-shop scheduling problem, given *N* jobs of varying processing times, which need to be scheduled on *M* machines, each of which may have different processing powers, minimize the makespan – the total length of the schedule to complete all *N* jobs.

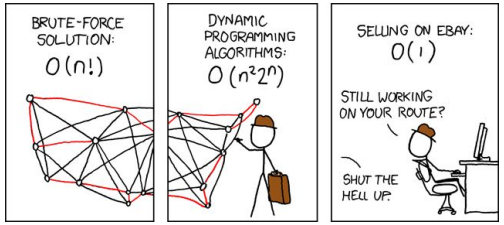




"multidimensional tetris"







Batch Job Scheduling in Practice

In practice, real-world batch job schedulers typically have three primary aims when they are scheduling your jobs on the compute resources they help manage:

- To minimize the time between the job submission and job completion: no user job should stay pending in the queue for extensive periods of time
- To optimize compute resource utilization: no compute resource should be idle for extensive periods of time
- To maximize job throughput: process as many jobs as soon as possible

In most cases, the batch job schedulers on the HPC systems you will encounter typically also employ some form of what is known as a fairshare scheduling algorithm to match user jobs to compute nodes over time.

** All interactive jobs you run on an HPC system are usually treated the same as batch jobs.*

Table of Contents

Slurm Workload Manager

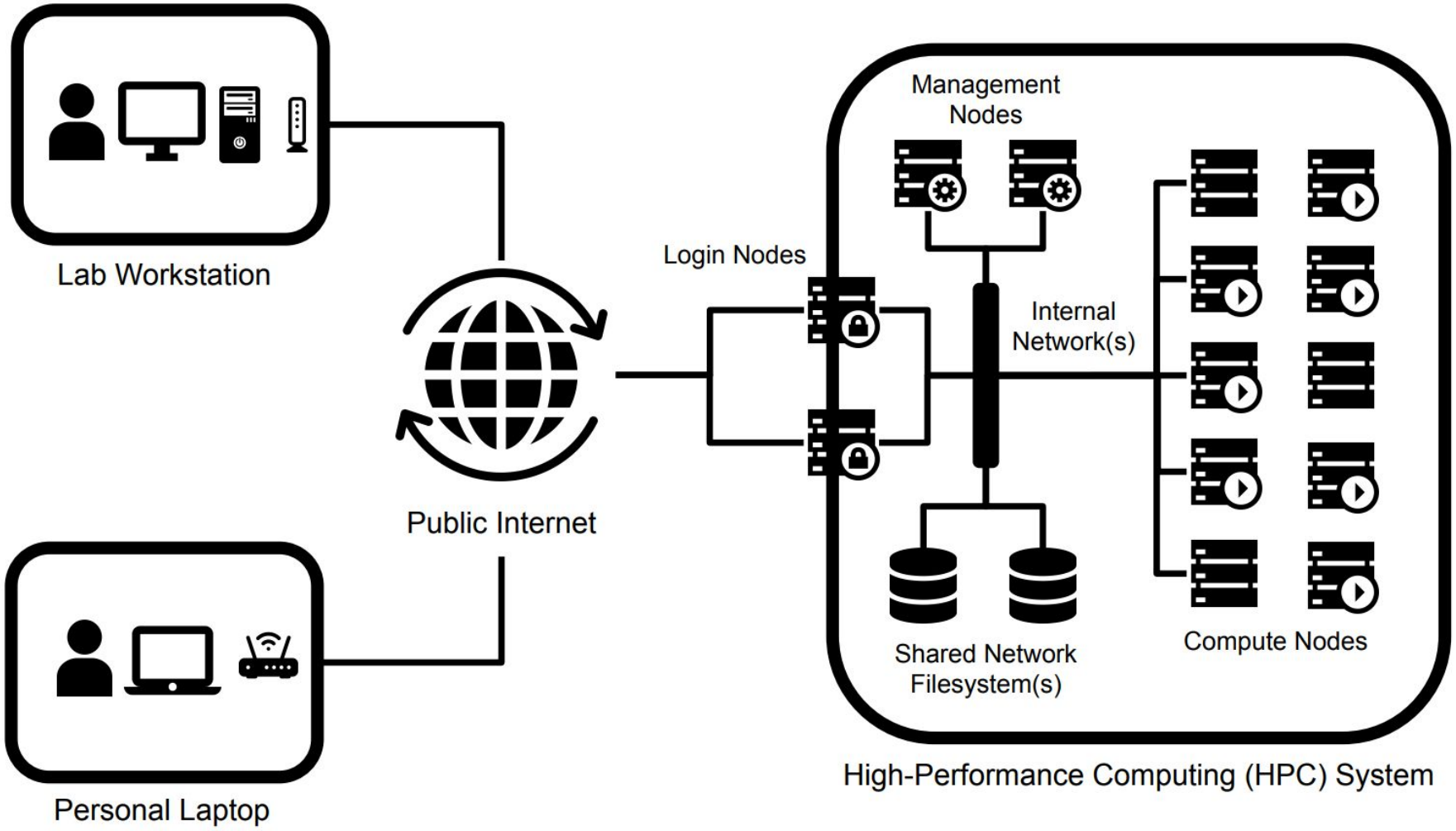
The Slurm Workload Manager is a free and open-source distributed batch job scheduler and resource manager for Unix-like operating systems used by the majority of the world's supercomputers and computer clusters today.

Slurm provides three key functions:

- the allocation of compute nodes for some period of time to process user jobs;
- a framework for starting, executing, and monitoring the progress of user jobs across a set of allocated compute nodes; and
- arbitrating the demand for compute nodes by managing a queue of pending user jobs

For more information, see the Slurm documentation: https://slurm.schedmd.com

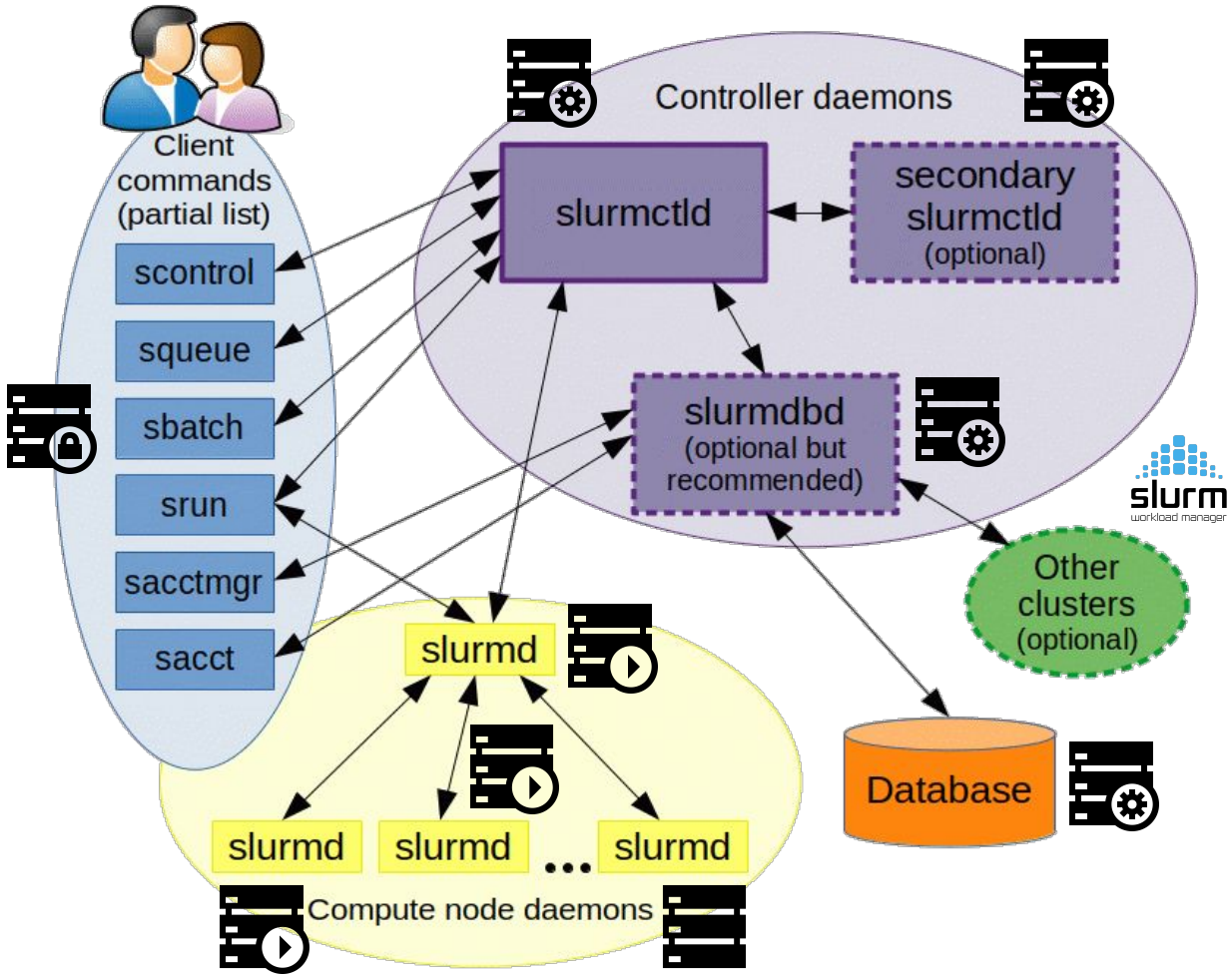
HPC System Architecture: Conceptual Model



Slurm Architecture

All Slurm schedulers are comprised of the three following types of service daemons:

- **`slurmd`** is the compute node daemon; it monitors all jobs running on the compute node, accepts jobs, launches jobs, and stops running jobs
- **`slurmctld`** is the central management daemon; it monitors all other Slurm daemons and resources, accepts jobs, and allocates resources to the jobs.
- **`slurmdbd`** is the accounting database daemon; it provides an interface for archiving accounting records to an external database



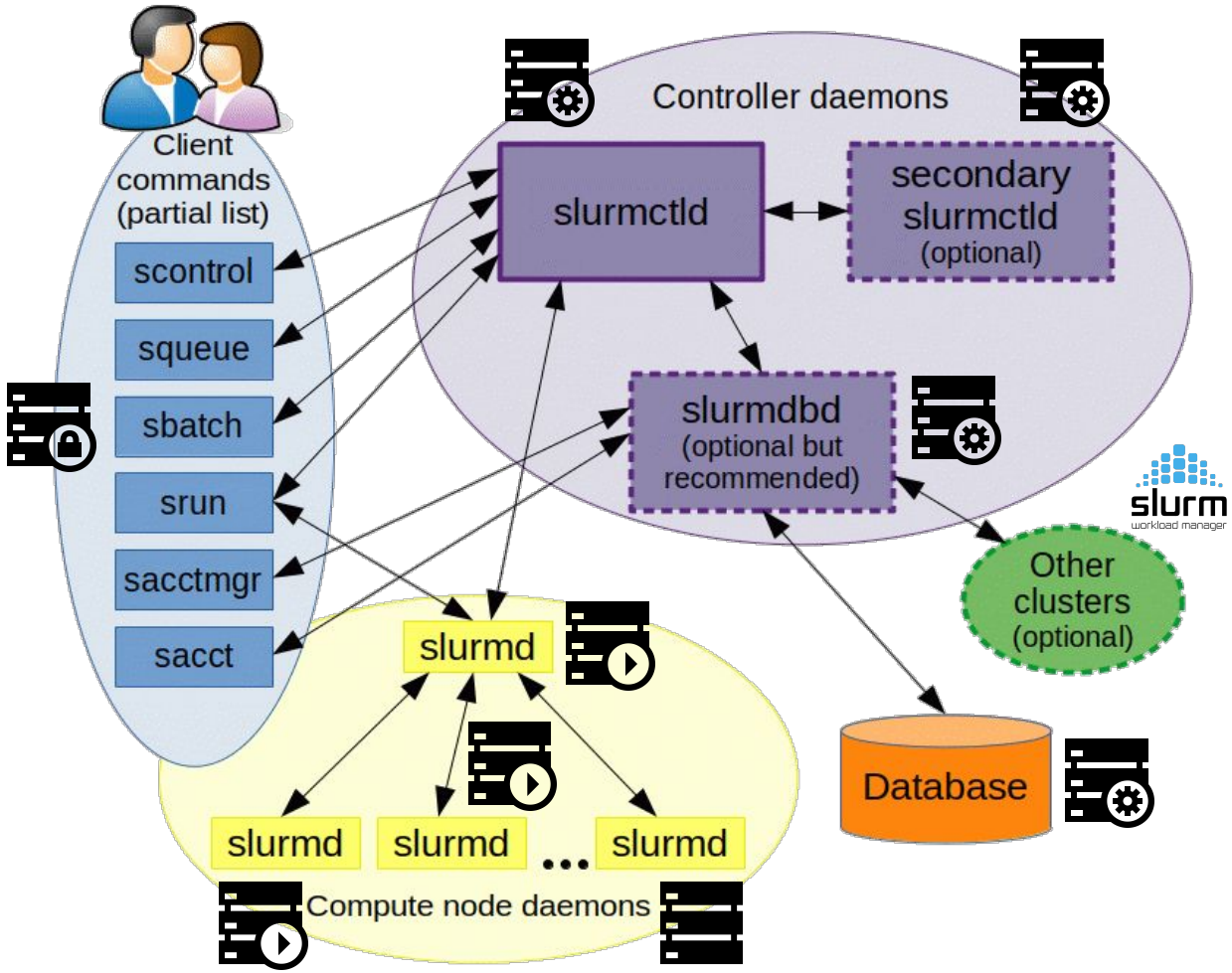
https://slurm.schedmd.com/quickstart.html

Slurm User Commands

The essential user commands you'll want to learn how to use when first getting started with Slurm are the following ones:

- `squeue`
- `sinfo`
- `sbatch`
- `scancel`
- `sacct`

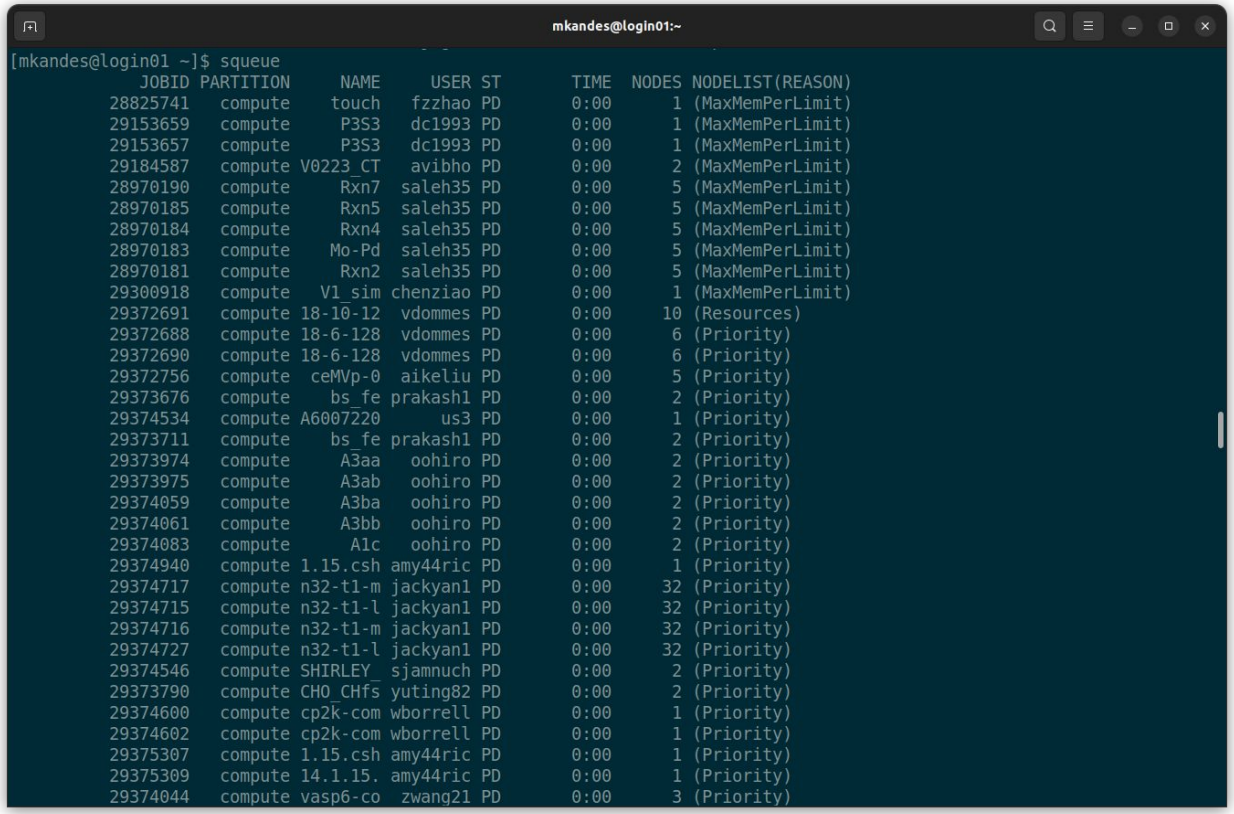
In general, you will only run these commands interactively on an HPC system's login nodes.



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

squeue

The **squeue** command is used to query and view information about the user jobs submitted to the Slurm scheduling queue.



In general, you should only need to check the status of the jobs you've submitted to Slurm.

Recommendation: Create an alias in your shell's configuration file to only check the status of your jobs.

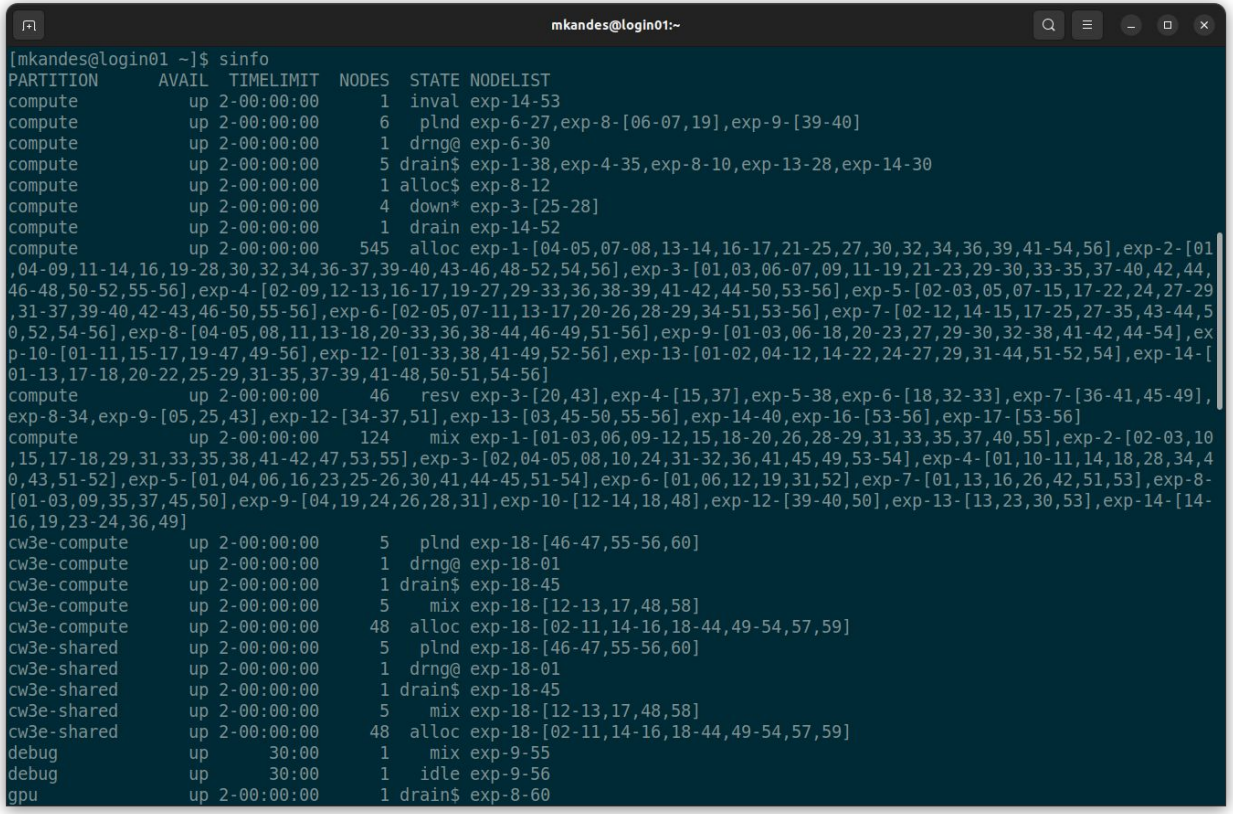
```
alias squeue="squeue -u ${USER}"
```

Do ***NOT*** run the `squeue` command incessantly, either on purpose or on accident.

slurm
workload manager

sinfo

The **`sinfo`** command is used to query and view information about the compute nodes and partitions managed by Slurm.



Slurm partitions are used to group nodes into logical, sometimes overlapping sets of nodes.

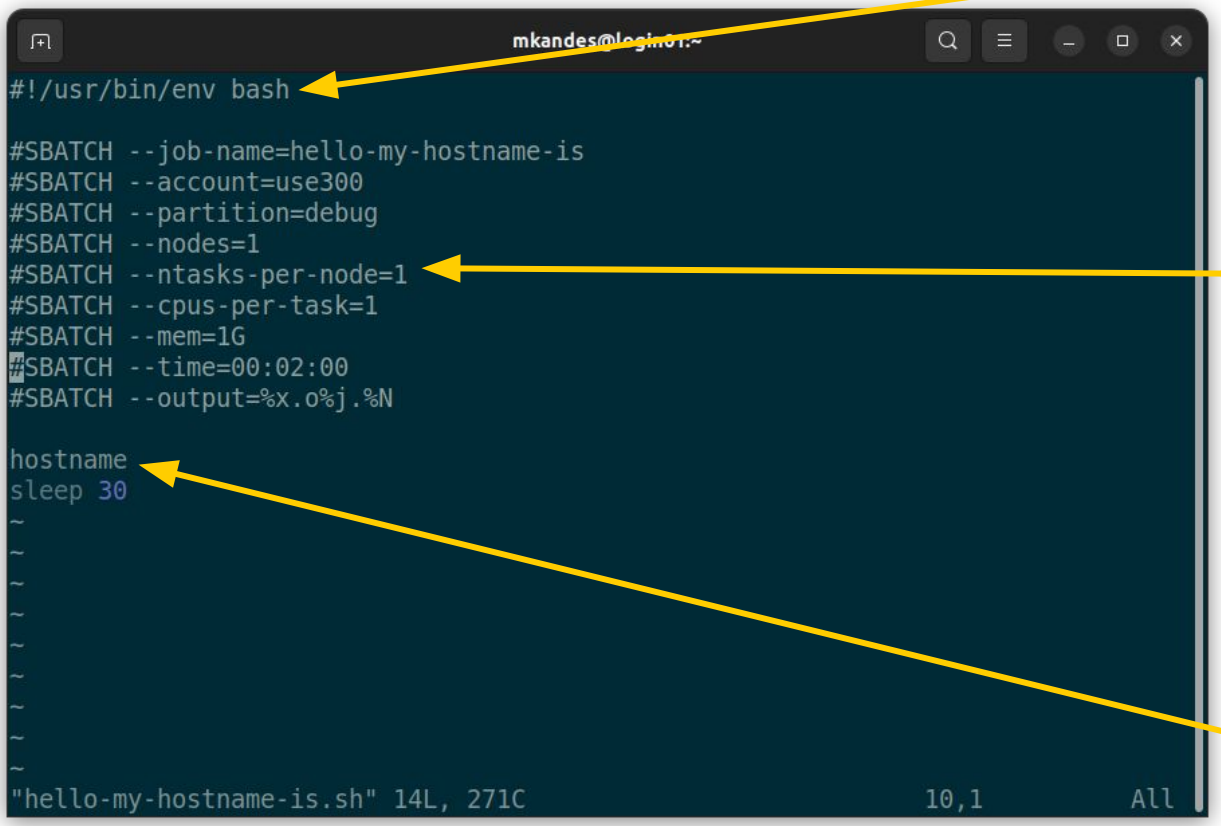
Each partition may have a unique set of constraints and features such as job size and runtime limits, types of resources available, user-based access permissions, etc.

You should always select the most appropriate partition for any job.

https://www.sdsc.edu/support/user_guides/expanse.html#running

Table of Contents

Exercise 1: Hello, my hostname is



shebang (#!) character sequence is used to specify an interpreter that will execute the commands present in a plain text file.

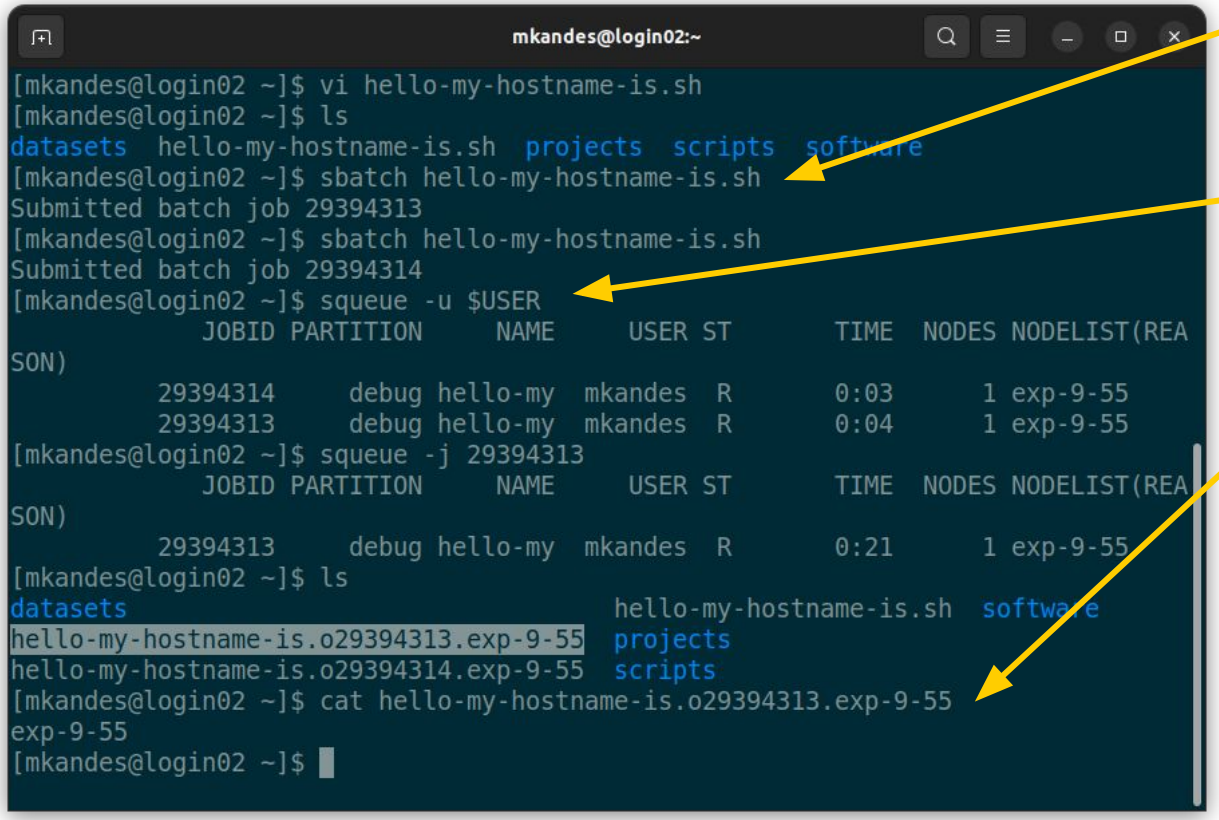
#SBATCH directives are used to specify the computational resource requirements for your job and various other attributes of it. e.g., name of the job and output files, which account to charge the job to, etc.

All executable commands to be run must follow the list of #SBATCH directives specified in any job script.

Exercise materials are available at
https://github.com/sdsc-complecs/batch-computing

sbatch

The **sbatch** command is used to submit a batch job script to Slurm.



- Upon submission, every batch job is assigned a `SLURM_JOB_ID` number.
- After submission, you can monitor the status of your jobs with the squeue command.
- While a job runs, it will write standard output (and error) data to the file(s) named in your #SBATCH directives.

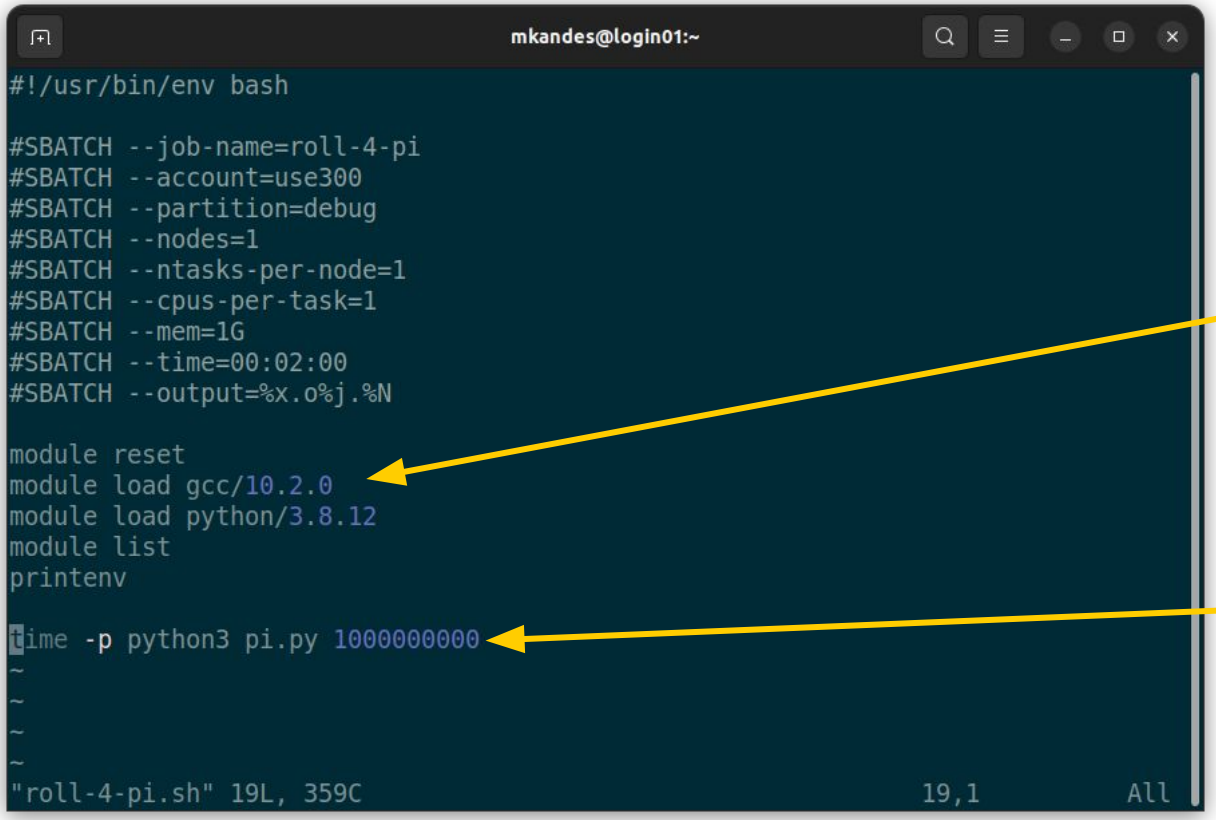
```
#SBATCH --output=%x.o%j.%N
```

`%x` = `SLURM_JOB_NAME`

`%j` = `SLURM_JOB_ID`

`%N` = `hostname -s`

Exercise 2: Roll 4 Pi



Exercise materials are available at
https://github.com/sdsc-complecs/batch-computing

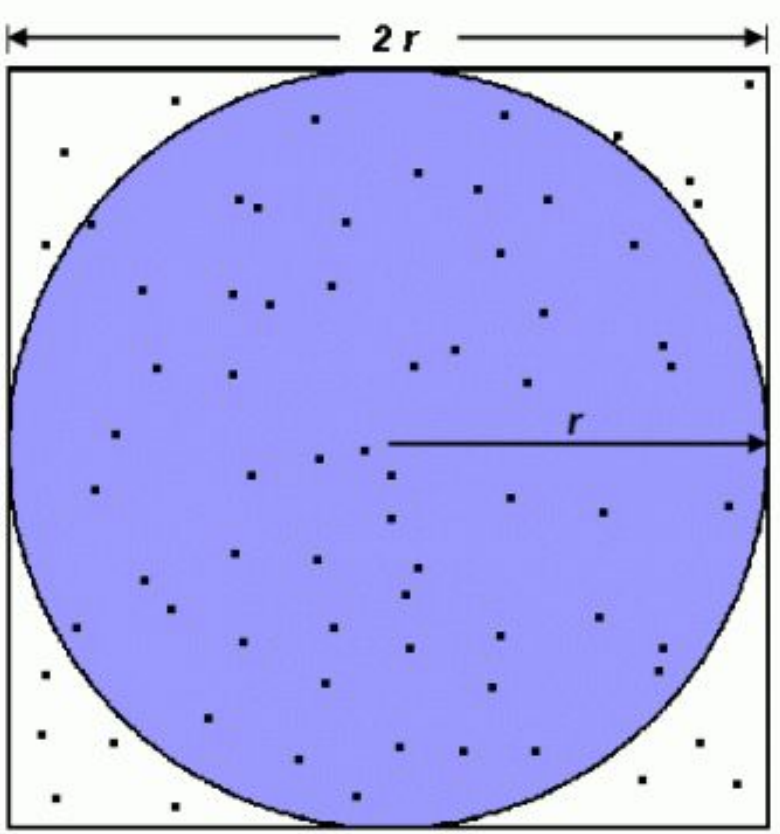
In most cases, you will want to follow the #SBATCH directives in your job script with a set of commands that configure the software environment in which the primary executable will run.

On many HPC systems, you will configure the software environment using pre-compiled and installed software modules.

Here, we estimate the value of Pi using a Monte Carlo method implemented in Python. The prepended **`time`** command measures the runtime of the estimation.

sbatch and roll (100M samples)

```
mkandes@login02:~
[mkandes@login02 ~]$ ls
datasets                                hello-my-hostname-is.sh  roll-4-pi.sh
hello-my-hostname-is.o29394313.exp-9-55  pi.py                    scripts
hello-my-hostname-is.o29394314.exp-9-55  projects                 software
[mkandes@login02 ~]$ sinfo -p debug
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up      30:00      1   plnd exp-9-56
debug        up      30:00      1    mix exp-9-55
[mkandes@login02 ~]$ sbatch roll-4-pi.sh
Submitted batch job 29402750
[mkandes@login02 ~]$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REA
SON)
          29402750     debug roll-4-p  mkandes  R       0:22      1 exp-9-55
[mkandes@login02 ~]$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REA
SON)
[mkandes@login02 ~]$ tail -n 4 roll-4-pi.o29402750.exp-9-55
3.1413662714136628
real 62.54
user 62.36
sys 0.01
[mkandes@login02 ~]$
```



$$A_S = (2r)^2 = 4r^2$$

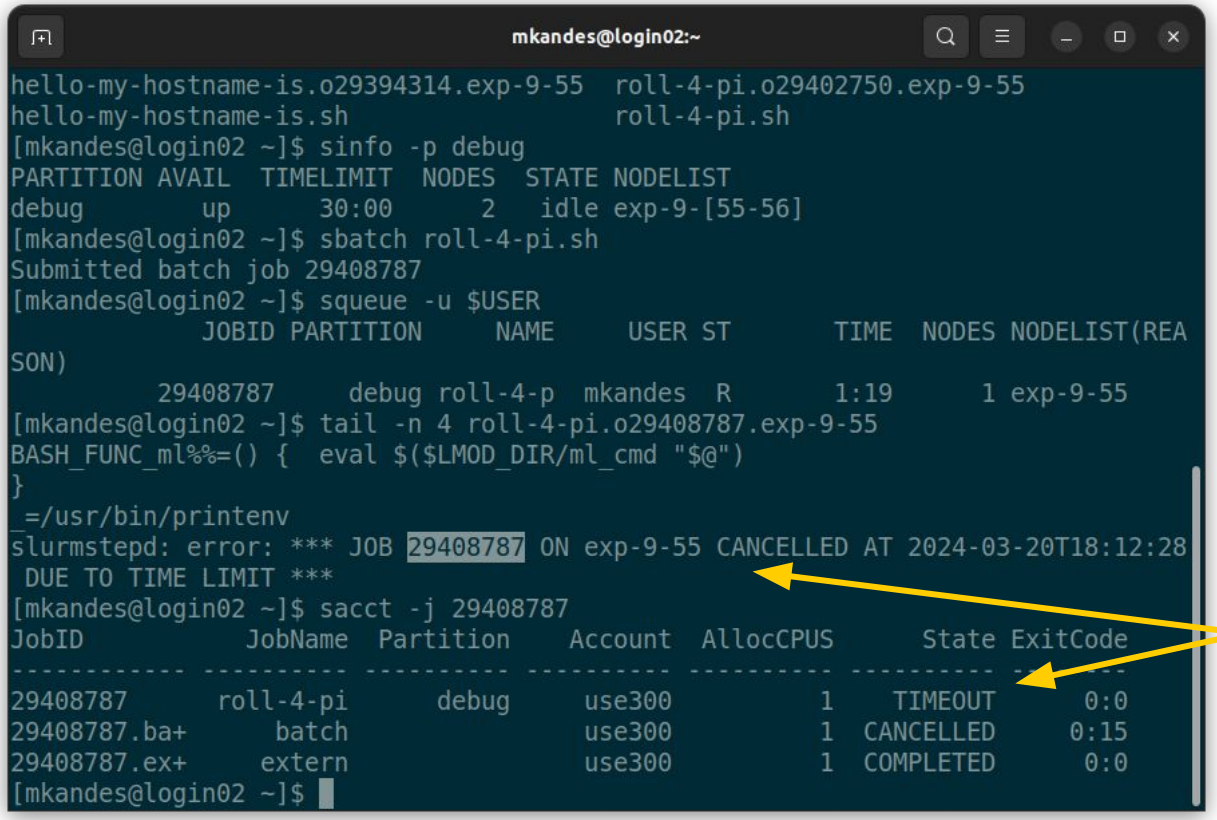
$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

What do you expect will happen if we increase the number of samples to one billion and resubmit the job??

sacct

The **`sacct`** command displays job data from the Slurm accounting database.



What can you do when you submit a job you later realize has a mistake and/or might fail?

Use to look up information about completed and/or failed jobs when they are no longer visible in `squeue`.

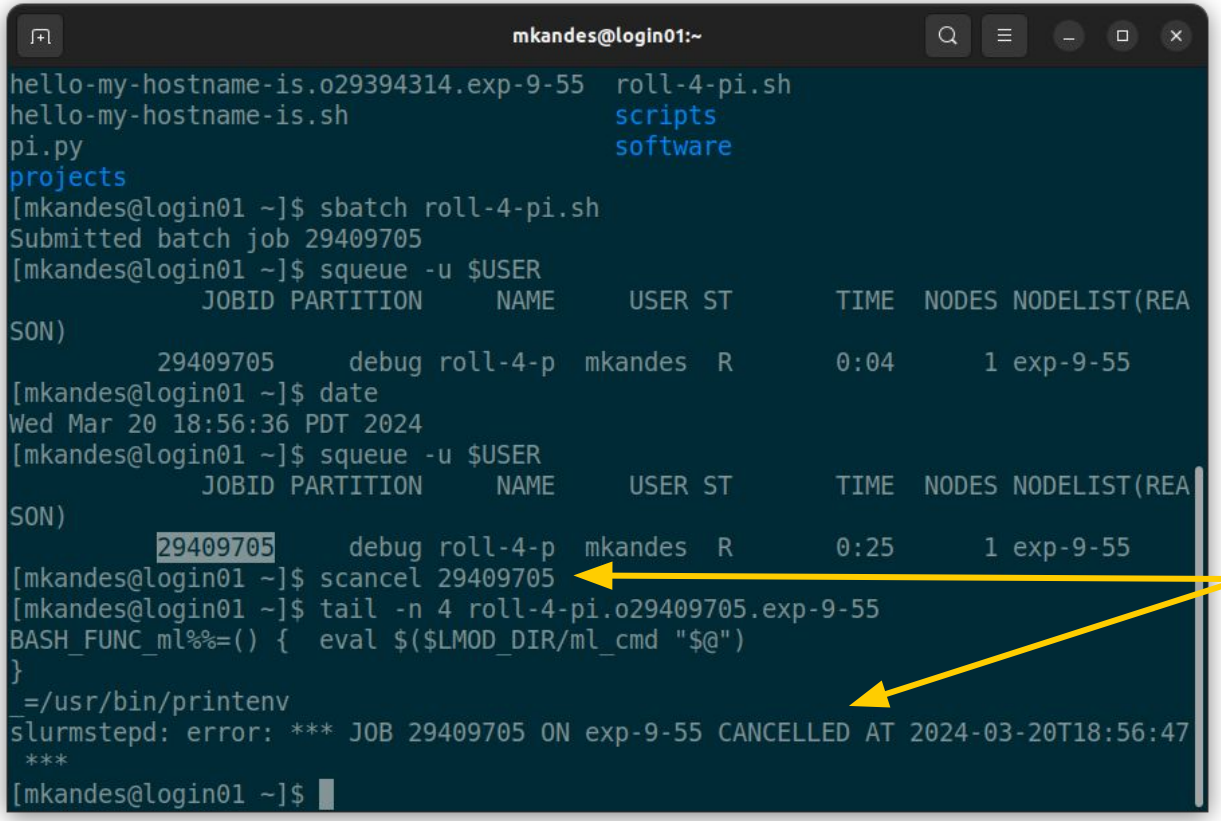
Quickly check the final state of a job and answer these questions: Did the job run and exit normally? If not, does Slurm know why it failed?

One of the most common failure modes for any batch job on an HPC system is running out of time.

Do ***NOT*** run extensive `sacct` queries without good reason.

scancel

The **scancel** command is used to signal or cancel jobs, job arrays or job steps.



Use whenever you want to remove a pending job from the squeue or stop a running job.

Table of Contents

Parallel Computing Myth #2: Scheduler Variant

Simply requesting more compute resources from your batch job scheduler will automatically reduce the time to solution for your problem.

WRONG!

The truth is your application must be designed to take advantage of the additional compute resources you've requested. If it is designed to do so, then you generally must still configure your batch job to correctly utilize those resources.

Prerequisite Knowledge:

- Parallel Computing Concepts

Exercise 3: Build Job, Run Job

If you need to compile and install software on an HPC system, then you should consider writing a batch job to build and install the software instead of proceeding to do so interactively from the command-line.

There are a number of advantages of using this approach:

- the build job script will document how to build and install the software
- the build and install logs will be available in the job's standard output (and/or error) files
- the software may need to be compiled on a specific compute node architecture
- the software environment created within the build job to compile the software will need to be replicated in the job scripts that eventually run the software anyway

Exercise materials are available at
https://github.com/sdsc-complecs/batch-computing

Parallel Pi with OpenMP

Parallel computing jobs can take advantage of more CPU resources

mkandes@login01:~

```
#!/usr/bin/env bash

#SBATCH --job-name=build-pi-omp
#SBATCH --account=use300
#SBATCH --partition=debug
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=1G
#SBATCH --time=00:02:00
#SBATCH --output=%x.o%j.%N

module reset
module load gcc/10.2.0
module list
printenv

make
~
~
~
~
~
"build-pi-omp.sh" 18L, 308C                                   18,1          All
```

Build Job

mkandes@login01:~

```
#!/usr/bin/env bash

#SBATCH --job-name=run-pi-omp
#SBATCH --account=use300
#SBATCH --partition=debug
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=1G
#SBATCH --time=00:02:00
#SBATCH --output=%x.o%j.%N

module reset
module load gcc/10.2.0
module list
export OMP_NUM_THREADS="${SLURM_CPUS_PER_TASK}"
printenv

time -p ./pi_omp.x -s 10000000000
~
~
~
~
"run-pi-omp.sh" 19L, 383C                                     8,1           All
```

Run Job

Use Slurm environment variables!

Exercise 4: GPU-Accelerated Workloads

mkandes@login02:~

```
#!/usr/bin/env bash

#SBATCH --job-name=train-cnn-cifar-c10-fp32-e42-bs256-tensorflow-22.08-tf2-py3-1v100
#SBATCH --account=use300
#SBATCH --partition=gpu-debug
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=10
#SBATCH --cpus-per-task=1
#SBATCH --mem=92G
#SBATCH --gpus=1
#SBATCH --time=00:05:00
#SBATCH --output=%x.o%j.%N

declare -xr LOCAL_SCRATCH_DIR="/scratch/${USER}/job_${SLURM_JOB_ID}"
declare -xr LUSTRE_PROJECTS_DIR="/expanse/lustre/projects/${SLURM_JOB_ACCOUNT}/${USER}"
declare -xr LUSTRE_SCRATCH_DIR="/expanse/lustre/scratch/${USER}/temp_project"
declare -xr SINGULARITY_CONTAINER_DIR='/cm/shared/apps/containers/singularity'

declare -xr SINGULARITY_MODULE='singularitypro/3.9'

module purge
module load "${SINGULARITY_MODULE}"
module list
export KERAS_HOME="${LOCAL_SCRATCH_DIR}"
printenv

time -p singularity exec --bind "${KERAS_HOME}:/tmp" --nv "${SINGULARITY_CONTAINER_DIR}/tensorflow/tensorflow_22.08-tf2-py3.sif" \
  python3 -u tf2-train-cnn-cifar.py --classes 10 --precision fp32 --epochs 42 --batch_size 256
                                                                                    1,1           All
```

GPUs are now a first-class compute resource in Slurm

Define your own environment variables as needed!

Your job's software environment can be a mix of standard HPC software modules and alternative software package management methods. e.g., Singularity containers, conda environments

Exercise materials are available at
https://github.com/sdsc-complecs/batch-computing

Exercise 5: Multinode MPI Jobs

You can request multiple compute nodes for your jobs. But only do so if the software you're running supports distributed memory parallelism. e.g., MPI

mkandes@login01:~

```
#!/usr/bin/env bash

#SBATCH --job-name=gromacs-2020.4-2ufeq67-aocc-3.2.0-io3s466-openmpi-4.1.3-xigazqd-water-cut1.0_GMX50_bare-3072-4-node-128-mpi-1-omp
#SBATCH --account=use300
#SBATCH --partition=compute
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=1
#SBATCH --mem=243G
#SBATCH --time=00:05:00
#SBATCH --output=%x.o%j.%N

declare -xr COMPILER_NAME='aocc'
declare -xr COMPILER_VERSION='3.2.0'
declare -xr COMPILER_VARIANT='io3s466'
declare -xr COMPILER_MODULE="${COMPILER_NAME}/${C

declare -xr MPI_NAME='openmpi'
declare -xr MPI_VERSION='4.1.3'
declare -xr MPI_VARIANT='xigazqd'
declare -xr MPI_MODULE="${MPI_NAME}/${MPI_VERSION

declare -xr APPLICATION_NAME='gromacs'
```

mkandes@login01:~

```
  if [[ ! -f "${SLURM_SUBMIT_DIR}/benchmarks/${GROMACS_BENCHMARK_TARBALL}" ]]; then
    wget "${GROMACS_ROOT_URL}/benchmarks/${GROMACS_BENCHMARK_TARBALL}"
  fi
  tar -xzvf "${GROMACS_BENCHMARK_TARBALL}"
fi

cd "${SLURM_SUBMIT_DIR}"

time -p mpirun -n 1 gmx_mpi grompp \
  -f "${GROMACS_BENCHMARK_DATA_DIR}/pme.mdp" \
  -c "${GROMACS_BENCHMARK_DATA_DIR}/conf.gro" \
  -p "${GROMACS_BENCHMARK_DATA_DIR}/topol.top" \
  -po "mdout.${SLURM_JOB_NAME}.${SLURM_JOB_ID}.mdp" \
  -o "topol.${SLURM_JOB_NAME}.${SLURM_JOB_ID}.tpr"

time -p mpirun -n "${SLURM_NTASKS}" --bind-to core --map-by ppr:1:core:PE=1 \
  --display-allocation --display-map --report-bindings --verbose gmx_mpi mdrun \
  -nb cpu -pme cpu -bonded cpu -pin on -resethway -noconfout -nsteps 10000 \
  -s "topol.${SLURM_JOB_NAME}.${SLURM_JOB_ID}.tpr" \
  -cpo "state.${SLURM_JOB_NAME}.${SLURM_JOB_ID}.cpt" \
  -e "ener.${SLURM_JOB_NAME}.${SLURM_JOB_ID}.edr" \
  -g "md.${SLURM_JOB_NAME}.${SLURM_JOB_ID}.log" \
  -v
                                                                77,3          Bot
```

Exercise materials are available at
https://github.com/sdsc-complecs/batch-computing

But again, you will still likely need to configure the call to the application to use the requested resources correctly.

Some final notes

- When you get started on a new HPC system, or begin working with a new piece of software, you may not know computational requirements for the types of workloads you need to run. Experiment! Start small on number of CPUs/GPUs, amount of memory, then scale up your job resource requests as needed. Start long on runtime, then dial it back.
- Be aware of how your HPC system charges for the resources requested from the scheduler. Some charge by node-hour, some by CPU-core-hour, some by GPU-hour, some by memory, or even a combination of all of the above. In general, you are only charged for the resources consumed, not those requested.
- Read your HPC system's user guide, the Slurm documentation, and/or the documentation to the specific software you're attempting to run.
- If you have any questions, please don't hesitate to contact your HPC user services support team. They are there to help you understand how to utilize these specialized resources effectively for your research.

Scheduler Rosetta Stone

Common User Commands					
	PBS/Torque	**Slurm**	**LSF**	**SGE**	**LoadLeveler**
Job submission	qsub [script_file]	sbatch [script_file]	bsub [script_file]	qsub [script_file]	llsubmit [script_file]
Job deletion	qdel [job_id]	scancel [job_id]	bkill [job_id]	qdel [job_id]	llcancel [job_id]
Job status	qstat [job_id]	squeue [job_id]	bjobs	qstat [-j job_id]	llq -u [username]
Job status (by user)	qstat -u [user_name]	squeue -u [user_name]	bjobs	qstat [-j job_id]	llq -u [user_name]
Job hold	qhold [job_id]	scontrol hold [job_id]	bstop [job_id]	qhold [job_id]	llhold -r [job_id]
Job release	qrls [job_id]	scontrol release [job_id]	bresume [job_id]	qrls [job_id]	llhold -r [job_id]
Queue list	qstat -Q	squeue	bqueues	qconf -sql	llclass
GUI	xpbsmon	sview	xlsf/xlsbatch	N/A	xload
Environment Variables					
	PBS/Torque	**Slurm**	**LSF**	**SGE**	**LoadLeveler**
Job ID	$PBS_JOBID	$SLURM_JOBID	$LSB_JOBID	$JOB_ID	$LOAD_STEP_ID
Working Directory	$PBS_O_WORKDIR	Use "pwd" command	Use "pwd" command	$SGE_O_WORKDIR	$LOADL_STEP_INITDIR
Node List	$PBS_NODEFILE	$SLURM_JOB_NODELIST	$LSB_HOSTS/LSB_MCPU_HOST	$PE_HOSTFILE	$LOADL_PROCESSOR_LIST
Job Specification					
	PBS/Torque	**Slurm**	**LSF**	**SGE**	**LoadLeveler**
Script directive	#PBS	#SBATCH	#BSUB	#$	#@
Queue	-q [queue]	-p [queue]	-q [queue]	-q [queue]	class=[queue]
Node Count	-l nodes=[count]	-N [min[-max]]	-n [count]	N/A	node=[count]
CPU Count	-l ppn=[count]	-n [count]	-n [count]	-pe ompi [#]	tasks_per_node=[count]
Wall Clock Limit	-l walltime=[hh:mm:ss]	-t [min] OR -t [days-hh:mm:ss]	-W	-l time=[hh:mm:ss]	wall_clock_limit=[hh:mm:ss]
Standard Output FIle	-o [file_name]	-o [file_name]	-o [file_name]	-o [path]	output=[file]
Standard Error File	-e [file_name]	e [file_name]	-e [file_name]	-e [path]	error=[file]
Combine stdout/err	-j oe (both to stdout) OR -j eo (both to stderr)	(use -o without -e)	(use -o without -e)		
Copy Environment	-V	--export=[ALL | NONE | variables]	?	-V	environment=COPY_ALL
Event Notification	-m abe	--mail-type=[events]	-B or -N	-m abe	notification=start|error|complete|never|always
Email Address	-M [address]	--mail-user=[address]	-u [address]	-M [address]	notify_user=[address]
Job Name	-N [name]	--job-name=[name]	-J [name]	-N [name]	job_name=[name]
Job Restart	-r [y|n]	--requeue OR --no-requeue (NOTE: configurable default)	-r	-r [yes|no]	restart=[yes|no]
Job Type	N/A	N/A	N/A	N/A	job_type=[type]
Working Directory	N/A	--workdir=[dir_name]	(submission directory)	-wd [directory]	initialdir=[directory]
Resource Sharing	-l naccesspolicy=singlejob	--exclusive OR--shared	-x	N/A	node_usage=not_shared
Memory Size	l mem=[MB]	--mem=[MB] OR --mem-per-cpu=[MB]	-M [MB]	-mem [GB]	requirements=(Memory >= [MB])
Project to charge	-W group_list=<project>	--acount=[project]	-P <project>		
Resource Requirements			-R		
Job dependency		--depend=[job_id:state]	-w [done | exit | finish]		
Job project		--wckey=[name]	-P [name]		
Job host preference		--nodelist=[nodes] AND/OR --exclude= [nodes]	-m [nodes]		

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

Additional Information

- Slurm Quick Start User Guide
- HPCWIKI: Scheduling Basics
- Introduction to HPC
- HPC in a Day
- Google's Shell Style Guide

Table of Contents

Questions & Answers