

**COMPLETE
SUCCESSFULLY**

<code>sbatch job.sh</code>	Submit a job script
<code>squeue -u \$USER</code>	View your running/queued jobs
<code>squeue -j <jobid></code>	View details of a specific job
<code>scancel <jobid></code>	Cancel a job
<code>scancel -u \$USER</code>	Cancel all your jobs
<code>salloc -A <account></code>	Allocate interactive resources
<code>srun ./my_program</code>	Run a command in allocated resources
<code>scontrol show job <jobid></code>	Show detailed job information
<code>seff <jobid></code>	View job efficiency report
<code>sacct -u \$USER</code>	View completed jobs
<code>sacct -j <jobid> --format=JobID,State,Elapsed,MaxRSS</code>	Detailed accounting for a job

HPC User Environment 2

Oleg N. Starovoytov

HPC User Services

LSU HPC / LONI

sys-help@loni.org

Louisiana State University, Baton Rouge

July 15th, 2026

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

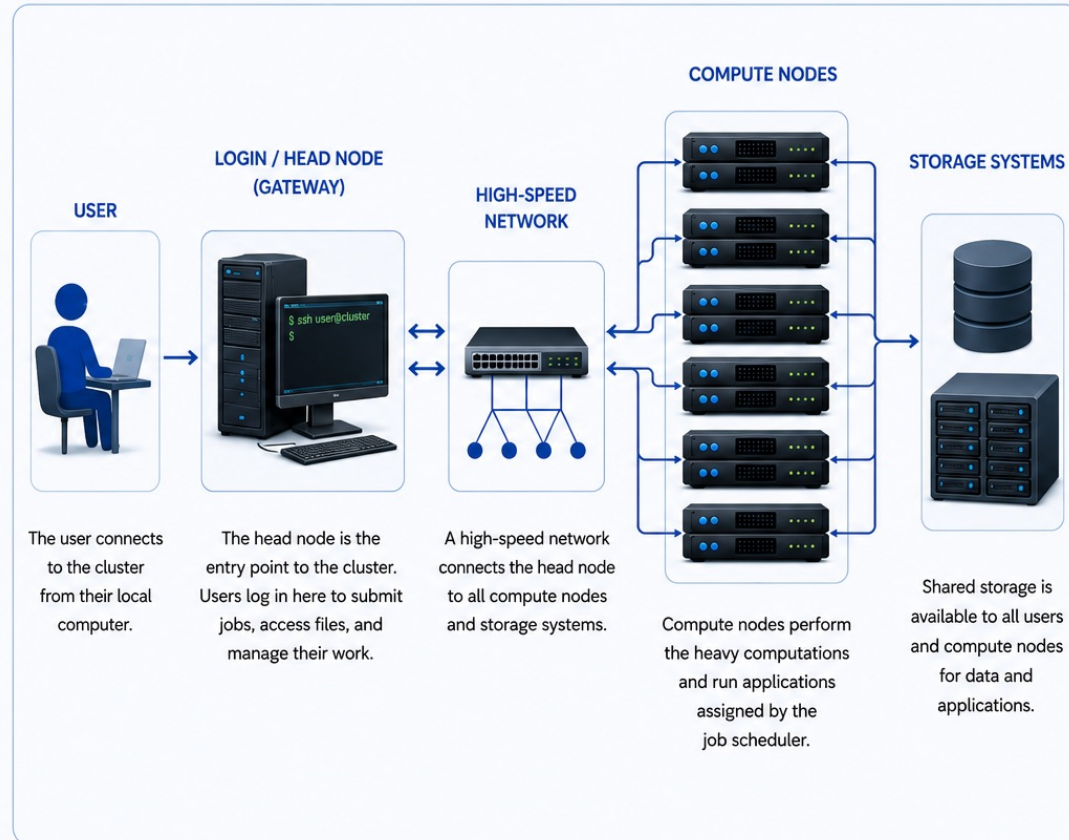
WHAT IS A CLUSTER?

A cluster is a **collection of interconnected computers (nodes)** that work together as a **single system**.

It combines the resources of multiple machines to solve large problems, run applications faster, and handle many tasks efficiently.

★ KEY FEATURES

- ✓ Multiple nodes working together
- ✓ Shared storage and resources
- ✓ Managed by a job scheduler (e.g., Slurm)
- ✓ Designed for high performance and scalability
- ✓ Used for research, simulations, data analysis, AI/ML, and more



What is a cluster?

 **High Performance Computing | Louisiana State University**

Home <https://www.hpc.lsu.edu/>**Users**

LONI HPC <https://hpc.loni.org/>

LONI High Performance Computing promotes scientific computing and technology across all disciplines, enabling education, research and discovery through the use of emerging, advanced technologies. LONI HPC provides the infrastructure and support necessary to facilitate heroic research efforts, utilizing cutting-edge technology to push the limits of scientific discovery.

- Quick Links
 - [Login to: LONI HPC User Profile](#)
 - [Request Compute Allocations](#)
 - [Request Storage Allocations](#)
- About LONI HPC
 - [About Us](#)
 - [Contact Us](#)
 - [Staff](#)
- LONI HPC Users
 - [Accounts & Allocations](#)
 - [Help](#)
 - [Password Security](#)
 - [Policy](#)
 - [Submit Ticket](#)
 - [User Information](#)

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

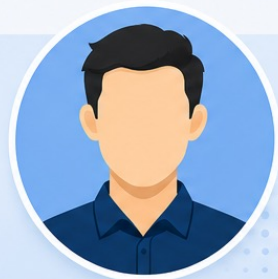
- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

1 ACCOUNT



Account

An **individual** who is authorized to access a computing cluster based on their affiliation, credentials, and permissions.



Key Points

- ✓ Represents a user identity on the system
- ✓ Managed by system administrators
- ✓ Required to log in and submit jobs

2 ALLOCATION



Allocation

A share of computing **resources** (like CPU cores or GPU hours) granted to an account for a specific period.



Key Points

- ✓ Determines the amount of computing resources you can use
- ✓ Has limits (cores, GPUs, storage, time)
- ✓ Typically associated with a project or group

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

A job is a **request to execute an application** on one or more compute nodes using resources allocated by the job scheduler.

The **job scheduler** (e.g., Slurm) allocates the requested resources (CPU cores, memory, GPUs, time) and runs the job on the appropriate compute nodes.

TYPES OF JOBS

1. BATCH JOBS (sbatch)

Submitted to the scheduler and executed without user interaction.

- Standard Batch Jobs**
Single production job.
- Job Arrays**
Many similar jobs submitted together.
- Dependent Jobs**
Jobs that start after another job completes.
- Long-running Jobs**
Batch jobs with extended wall times.
- GPU Batch Jobs**
Batch jobs requesting GPU resources.

2. INTERACTIVE JOBS (salloc)

Run interactively on allocated compute nodes for debugging, testing, compiling, and visualization.

- CPU Interactive Jobs**
Interactive jobs using CPU resources.
- GPU Interactive Jobs**
Interactive jobs using GPU resources.

JOB COMPONENTS

Each job you submit to the cluster consists of the following components:

1. JOB SCRIPT



A script (e.g., Bash) that contains scheduler directives (resource requests) and the commands to run your application.

2. RESOURCES REQUESTED



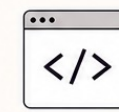
The resources you request such as CPU cores, memory, GPUs, time limit, and other constraints.

3. INPUT DATA



Input files and data needed by your application, stored in the shared filesystem.

4. APPLICATION / PROGRAM



The executable or program you want to run (e.g., VASP, Gaussian, GROMACS, Python script, etc.).



JOB

5. OUTPUT DATA



Results, logs, and output files generated by your application, saved to the shared filesystem.

6. ENVIRONMENT



Software modules, environment variables, and runtime settings required to run the application correctly.

7. JOB SCHEDULER



The scheduler (e.g., Slurm) manages the job queue, allocates resources, and executes the job.

JOB EXECUTION FLOW

1. SUBMIT JOB



You submit a job script (sbatch or salloc).

2. SCHEDULER QUEUE



The job is placed in the scheduler queue.

3. RESOURCE ALLOCATION



The scheduler allocates the requested resources (CPUs, memory, GPUs, time) to the job.

4. JOB EXECUTION



Your application runs on the allocated compute node(s).

5. OUTPUT GENERATED



Output files and logs are written to the shared filesystem.

6. JOB COMPLETED



The job finishes and resources are released back to the cluster.

RESOURCES TO REQUEST FOR YOUR JOB

When you submit a job to the cluster, you must request the resources your job needs.
Requesting the right resources helps your job run successfully and allows fair use of the cluster.

RESOURCE	WHAT IT MEANS	WHY IT MATTERS	EXAMPLE (SLURM)	TIPS
 1. CPU CORES Number of CPU cores (or tasks) your job will use.	More cores allow your program to run tasks in parallel.	Insufficient cores may slow down your job. Too many may waste cluster resources.	<pre>#SBATCH --cpus-per-task=16 or #SBATCH --ntasks=4 #SBATCH --cpus-per-task=4</pre>	 Check your application scalability. Use only the cores you need.
 2. MEMORY (RAM) Amount of memory required per node.	Memory is used to store data and perform calculations.	Not enough memory can cause slowdowns or job failure (e.g., out of memory).	<pre>#SBATCH --mem=64G or #SBATCH --mem-per-cpu=4G</pre>	 This option is not supported on LSU and LONI HPC clusters.
 3. GPUS Number of GPUs (if your job uses GPUs).	GPUs accelerate certain computations (e.g., AI/ML, molecular dynamics).	GPUs are limited resources. Request only if your application can use them.	<pre>#SBATCH --gres=gpu:2 (requests 2 GPUs)</pre>	 Confirm your code can use GPUs and how many it supports.
 4. WALL TIME Maximum time your job is allowed to run.	Time limit for the job from start to finish.	If your job exceeds the time limit, it will be terminated.	<pre>#SBATCH --time=DD-HH:MM:SS (DD=days, HH=hours, MM=minutes, SS=seconds)</pre>	 Provide realistic estimates with some buffer.
 5. STORAGE Disk space for input, output, and temporary files.	Includes space in your home, project (allocation), or scratch storage.	Not enough space can cause job failure or prevent writing output.	(No direct #SBATCH option) Ensure sufficient space in your allocation or scratch.	 Use scratch storage for large or temporary data when possible.
 6. NODES Number of compute nodes required.	Some jobs need multiple nodes (e.g., MPI jobs with many tasks).	Multi-node jobs require more resources and network communication.	<pre>#SBATCH --nodes=2 (requests 2 nodes)</pre>	 Check your application documentation for node requirements.

Problem:

You are given a **startup allocation of 150,000 Service Units (SUs)** to use on an HPC cluster. Each compute node contains **64 cores**. Service Units are charged based on the **number of cores used multiplied by the number of hours the job runs**. (CORES x HOURS)

You plan to run a job that uses **two nodes (128 cores)** and is expected to run for **10 hours**.

Your task is to **calculate how many Service Units this job will consume** and then **determine how many Service Units will remain** in your allocation after the job finishes.

$$1 \text{ SU Unit} = 1 \text{ CORE} / \text{hour}$$

Problem:

You are given a **startup allocation of 150,000 Service Units (SUs)** to use on an HPC cluster. Each compute node contains **64 cores**. Service Units are charged based on the **number of cores used multiplied by the number of hours the job runs**. (CORES x HOURS)

You plan to run a job that uses **two nodes (128 cores)** and is expected to run for **10 hours**.

Your task is to **calculate how many Service Units this job will consume** and then **determine how many Service Units will remain** in your allocation after the job finishes.



1 SU Unit = 1 CORE / hour

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

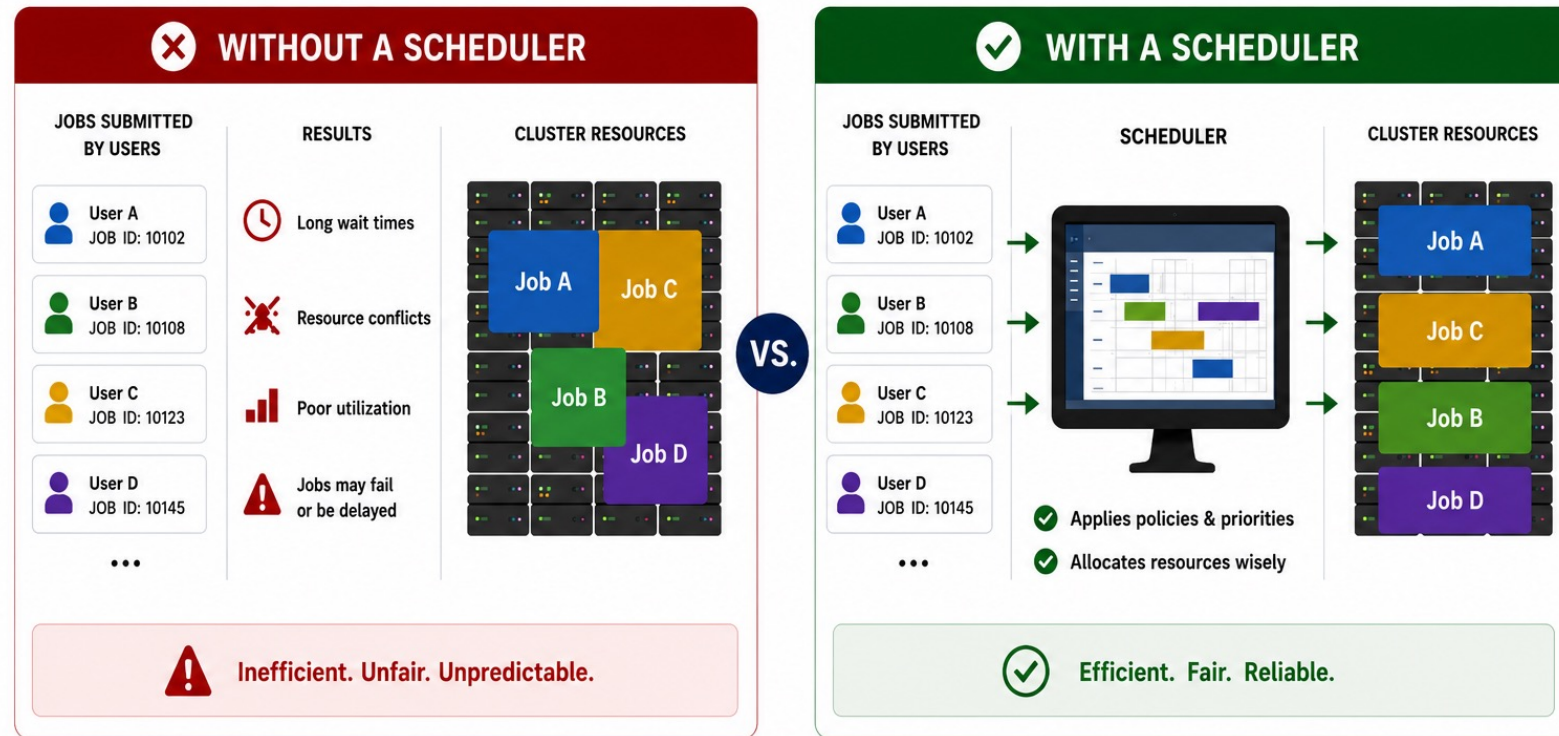
7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

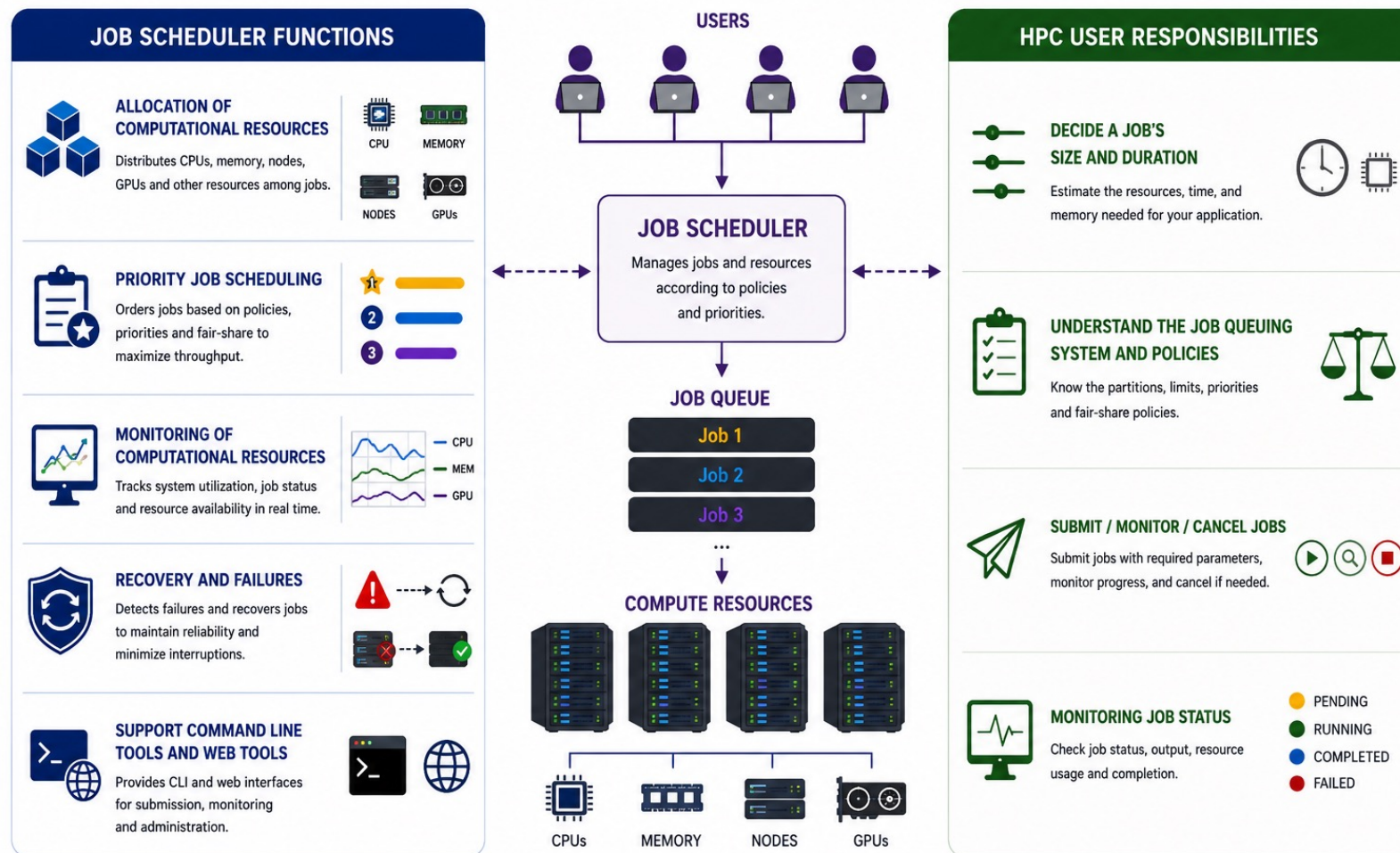
8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

Why a scheduler is needed



Why a scheduler is needed



1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

SLURM is Installed on LSU and LONI HPC Systems

A unified workload manager for reliable and efficient computing



SLURM (Simple Linux Utility for Resource Management) schedules and manages jobs, resources and users across our high-performance computing clusters.



Key Benefits

- ✓ Efficient resource utilization
- ✓ Fair share scheduling for all users
- ✓ Scalable for large and small workloads
- ✓ Reliable and widely used in the HPC community

SLURM is Installed On

SuperMike III



Powerful HPC system for advanced research and high-throughput computing

 mike.hpc.lsu.edu

SuperMic



High-performance computing cluster for the LSU research community

 smic.hpc.lsu.edu

Deep Bayou



State-of-the-art AI & HPC supercomputer for research and discovery

 db.hpc.lsu.edu

QB4 (QBD)



Scalable HPC resources for demanding research and data-intensive workloads

 qbd.loni.org

QB3 (QBC)



High-performance computing cluster for the Louisiana research and education community

 qbc.loni.org



<https://slurm.schedmd.com/documentation.html>

Quick Start User Guide

Overview §

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. Slurm requires no kernel modifications for its operation and is relatively self-contained. As a cluster workload manager, Slurm has three key functions. First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing a queue of pending work.

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

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

Guidelines for Requesting HPC Resources

Request sufficient resources



Large enough to meet computational needs.



Avoid overestimation



Small enough to optimize resource usage and access.



SLURM CLOCK



DD - HH : MM : SS

HPC Queue Information

Partition Name	Max Walltime (hours)	Max Nodes (per user)	Allowed Cores per Node
single	 168		 1 to 64 Cores
workq	 72		 64 Cores
checkpt	 72		 64 Cores
gpu2	 72	 Up to 4	 32/64 Cores
gpu4	 72	 Up to 4	 16/32/48/64 Cores
bigmem	 72	 Up to 4	 64 Cores

Questions	Answers
What if my command is still running when the wall time runs out?	The job is <u>terminated</u>. Any running process will be killed.
What if all my commands in the job finished before the wall time runs out?	The job will <u>exit</u> successfully when all commands finish.
If my job exits before requested wall time, how many SUs will I be charged?	You will be charged based on your <u>actual time used</u> (if it is less than the requested amount).
In that case, why don't I just request maximum wall time every time?	The queuing time might be long.



HPC Maximum Walltime

How long should you choose for your job?



Large enough

- To successfully complete your job



Small enough

- To ensure quick turnaround



Small enough

- Not to waste resources for other users

Questions	Answers
My code runs slow. Can I request more nodes / cores to make it faster?	- Not quite! Your code most likely is NOT using multiple nodes / cores, if: - You do not know if it is using multiple nodes / cores - You did not tell it to use multiple nodes / cores - You are not familiar with names like “MPI” / “OpenMP” Underutilization is THE most common warning received on our clusters
How many nodes / cores should I request?	- In short: We can’t answer that - Each code / job is different. You must test to determine

SuperMike-III Partitions

****Note:** Starting on July 1, 2023, jobs running in the gpu queue on Superl

Queue Name	Max Walltime	Max Nodes (per user)	Allowed Cores per Node
workq	72	84	64
checkpt	72	84	64
single	168	84	1 to 64
gpu	72	4	16/32/48/64
bigmem	72	4	64

<https://www.hpc.lsu.edu/resources/hpc/#available-partitions>

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

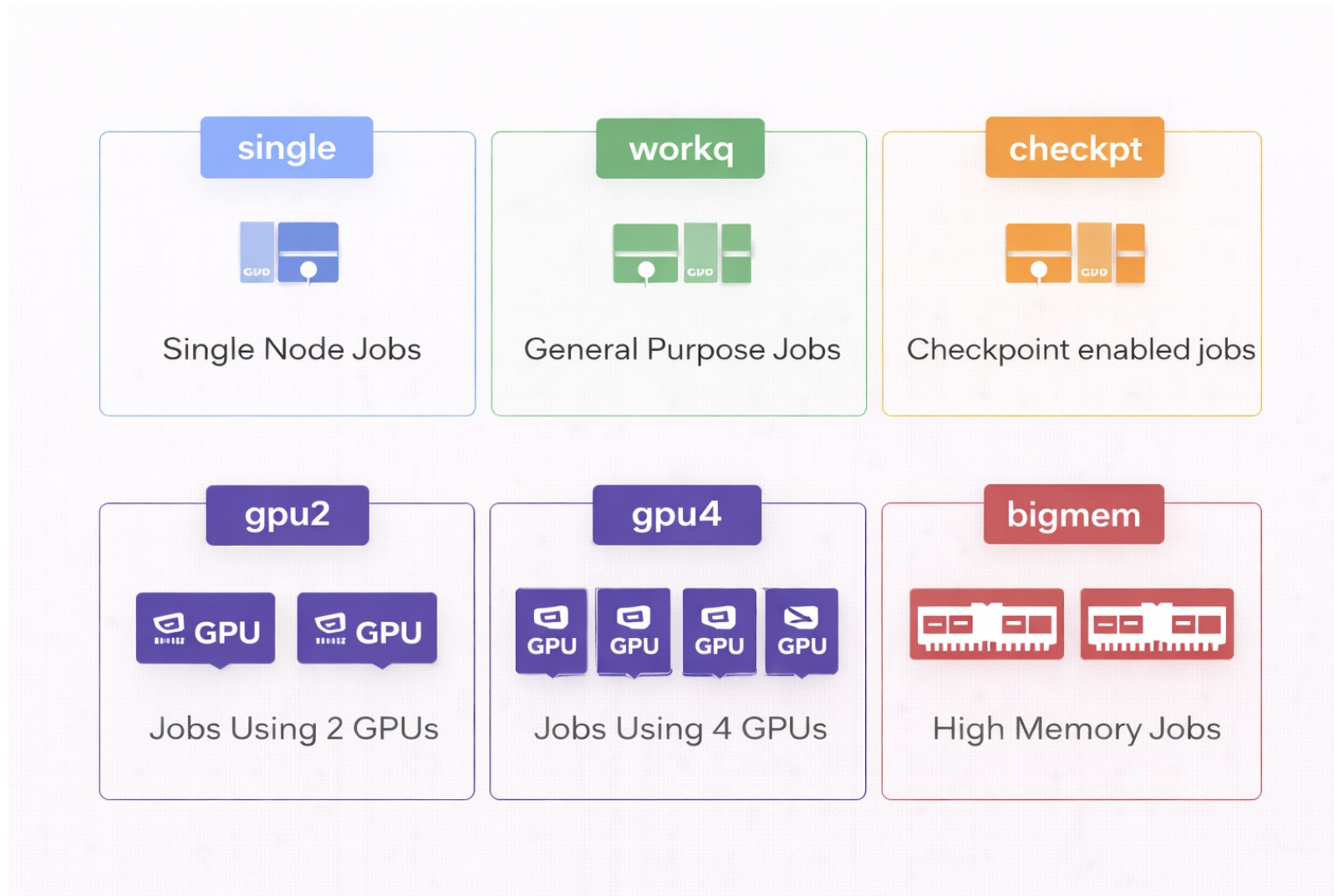
7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

Partition - a scheduler-defined group of compute nodes.



single

Purpose		• Only need a portion of one node
Names		• All clusters: single
Resource availability	Nodes	• Portion of one node: 1/2/4/8/16/32/64
	Cores	• 1 ~ all cores
	Memory	• A portion, proportional to the number of requested cores
Max duration		• 168 hours (7 days)

[QB-4]

- **Total:** 64 cores, 256 GB memory
→ 4 GB / core
- **Request:** 10 cores
→ 40 GB memory

checkpt / workq

Purpose		- General purposes - Most likely your default queue - Difference: non-preemptible (workq) vs. preemptible (checkpt)
Name		All clusters: workq / checkpt
Resource availability	Nodes	Entire node(s) - Up to a maximum
	Cores	All cores on the node(s)
	Memory	All memory on the node(s)
Max duration		72 hours (3 days)

bigmem

Purpose		• Need large memory (larger than regular computing nodes have)
Names		• All clusters: bigmem
Resource availability	Nodes	• Entire node(s)
	Cores	• All cores on the node
	Memory	• All memory on the node
Max duration		• 72 hours (3 days)

gpuX

gpuX :
X = [Number of GPUs on one node]

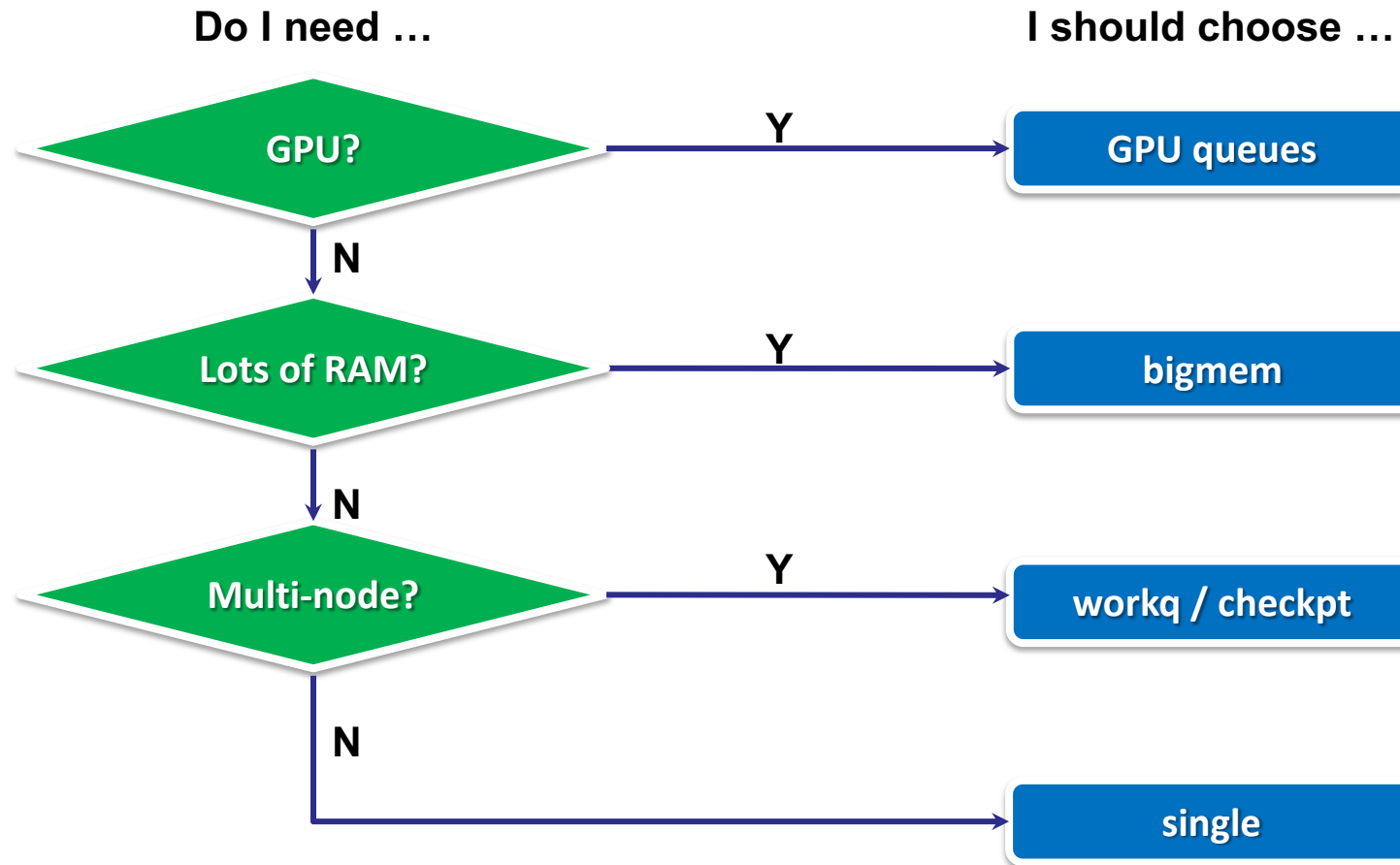
Purpose		• Need GPU	
Name		• QB-3: gpu2	• SMIC: gpu2 • Deep Bayou: gpu2, gpu4 • SuperMike 3: gpu4 • QB-4: gpu2, gpu4
Resource availability	Nodes	• Entire node(s)	• Portion or entire node(s)
	Cores	• All cores on the node(s)	• Portion or all on the node(s)
	Memory	• All memory on the node(s)	• Portion or all on the node(s)
	GPU	• All GPUs on the node(s)	• 1 ~ all GPU on the node(s)
Max duration		• 72 hours (3 days)	

[QBD / gpu4]

- Total: 64 cores, 4 GPUs
→ 16 cores / GPU
- Request: 3 GPUs
→ 48 cores

LSU HPC Cluster	Partition	Cores per node (ppn)	Max running jobs	Max nodes per user
SuperMIC (smic.hpc.lsu.edu)	workq	20	45 (global)	86
	checkpt			
	single	1 ~ 20		
	gpu2	18,36		2
	bigmem	28		3
DeepBayou (db1.hpc.lsu.edu)	gpu2	24,48	-	8
	gpu4	12,24,36,48		2
SuperMike-III (mike.hpc.lsu.edu)	workq	64	32 (global)	84
	checkpt			
	single	1 ~ 64		
	gpu4	16,32,48,64		4
	bigmem	64		4

LONI HPC Clusters	Partitions	Cores per node (n)	Max running jobs	Max nodes per user
QBC (QB3) (qbc.loni.org)	workq	48	32 (<i>global</i>)	48
	checkpt			
	single	1 ~ 48		
	gpu2	48		4
	bigmem	48		2
QBD (QB4) (qbd.loni.org)	workq	64	32 (<i>global</i>)	84
	checkpt			
	single	1 ~ 64		
	gpu2	32,64		4
	gpu4	16,32,48,64		4
	bigmem	64		5



Test

My job	Partition choice? (include number of nodes / cores)
• SMIC • MPI code, needs 100 CPU cores - Hint: SMIC has 20 cores / node	workq / checkpoint (5 nodes, 20 cores per node)
• SuperMike 3 • Uses 3 GPUs to train a neural network - Hint: SuperMike 3 has 64 cores / node, 4 GPUs / node → 16 cores / GPU	gpu4 (1 node, 48 cores per node)
• QB-3 • Single-core serial code • Needs to store and process 30 GB data in RAM - Hint: QB-3 has 192 GB RAM / node, 4 GB RAM / core	single (1 node, 8 cores per node)

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login In to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

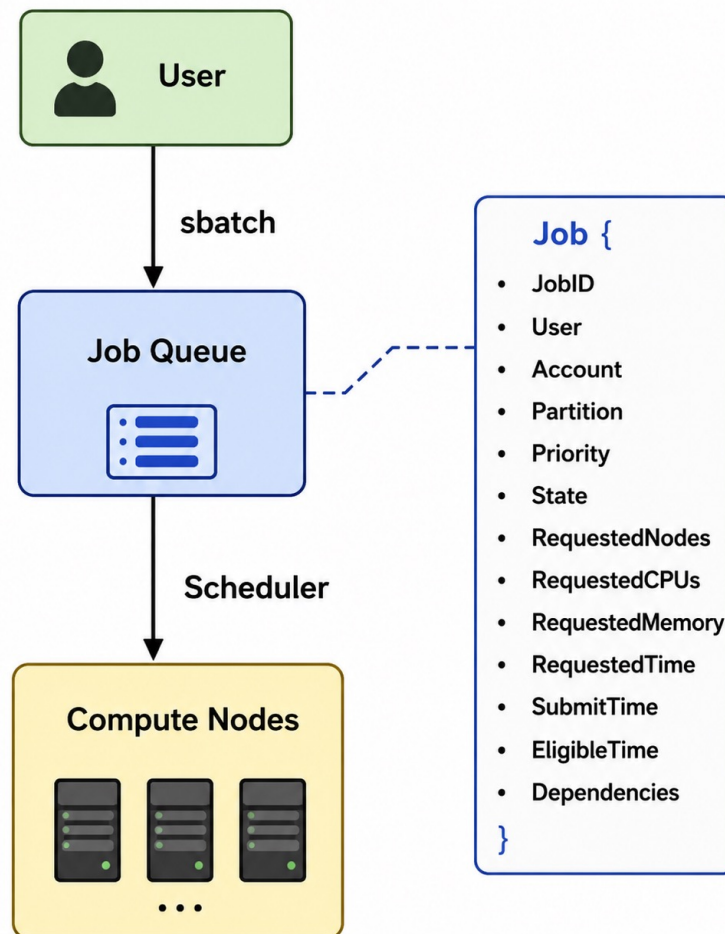
7. Monitoring Jobs

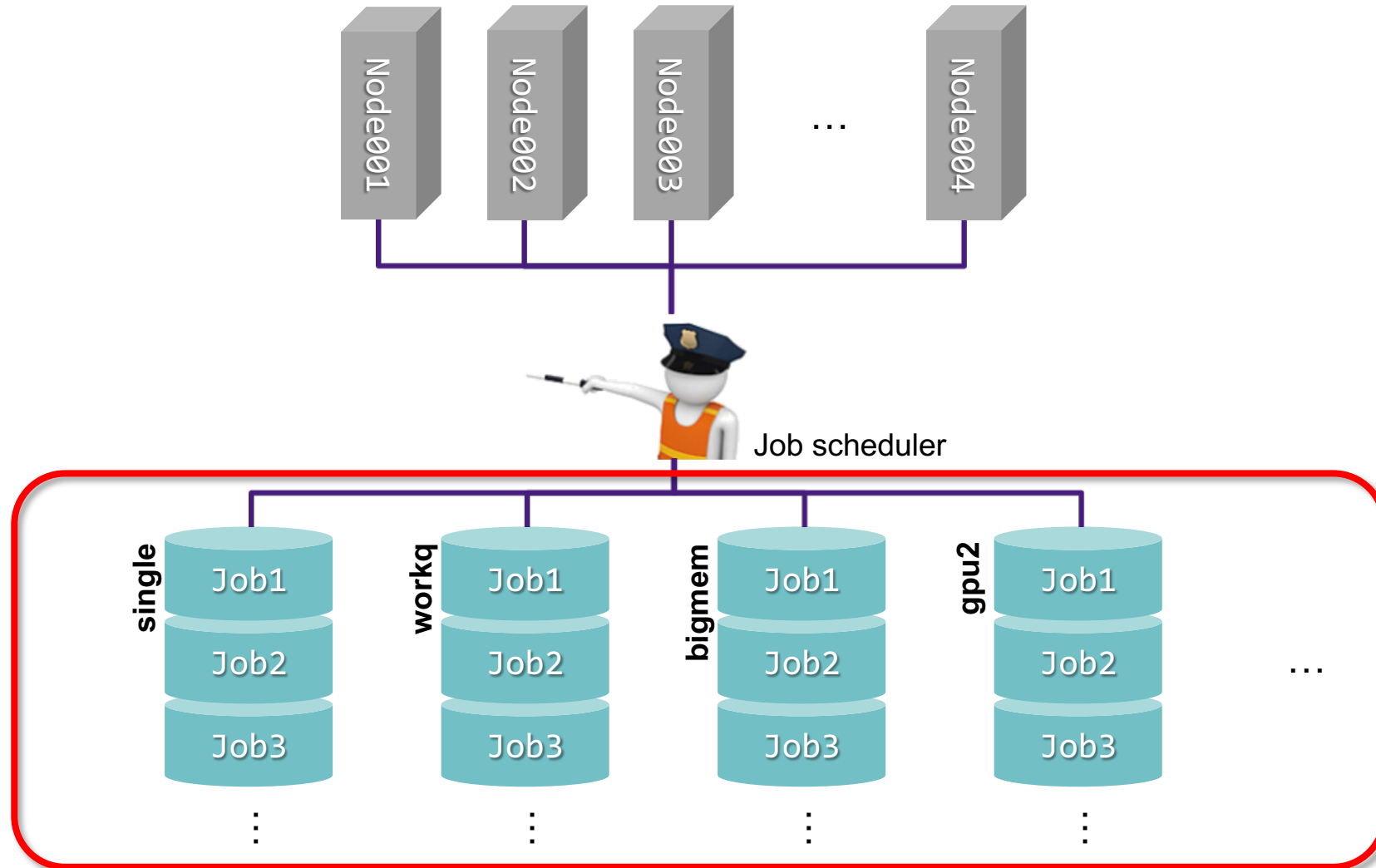
- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

Queue - a scheduler-defined waiting list of jobs with shared policies





Job Queue on HPC



1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login In to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

```
[username@hostname ~]$ sinfo
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
single*	up	7-00:00:00	1	drain*	qbd422
single*	up	7-00:00:00	1	down*	qbd041
single*	up	7-00:00:00	163	alloc	qbd[056-080,088-091,191-299,431-455]
single*	up	7-00:00:00	315	idle	qbd[001-040,042-055,081-087,092-190,300-421,423-430,456-480]
checkpt	up	3-00:00:00	1	drain*	qbd422
checkpt	up	3-00:00:00	1	down*	qbd041
checkpt	up	3-00:00:00	163	alloc	qbd[056-080,088-091,191-299,431-455]
checkpt	up	3-00:00:00	315	idle	qbd[001-040,042-055,081-087,092-190,300-421,423-430,456-480]
workq	up	3-00:00:00	1	drain*	qbd422
workq	up	3-00:00:00	1	down*	qbd041
workq	up	3-00:00:00	163	alloc	qbd[056-080,088-091,191-299,431-455]
workq	up	3-00:00:00	315	idle	qbd[001-040,042-055,081-087,092-190,300-421,423-430,456-480]
bigmem	up	3-00:00:00	1	alloc	qbd482
bigmem	up	3-00:00:00	4	idle	qbd[481,483-485]
gpu2	up	3-00:00:00	52	idle	qbd[486-511,517-542]
gpu4	up	3-00:00:00	3	mix	qbd[543-545]
gpu4	up	3-00:00:00	6	idle	qbd[512-516,546]
gpu-small	up	3-00:00:00	1	idle	qbd547

```
[username@host ~]$ sinfo --states=idle
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
single*	up	7-00:00:00	105	idle	mike[002-003,005,...,155-158]
ckpt	up	3-00:00:00	105	idle	mike[002-003,005,...,155-158]
workq	up	3-00:00:00	105	idle	mike[002-003,005,...,155-158]
bigmem	up	3-00:00:00	1	idle	mike175
gpu	up	3-00:00:00	2	idle	mike[182-183]
gpu4	up	3-00:00:00	2	idle	mike[182-183]

```
[username@hostname ~]$ sinfo -s
```

PARTITION	AVAIL	TIMELIMIT	NODES(A/I/O/T)	NODELIST
single*	up	7-00:00:00	408/7/65/480	qbd[001-480]
ckpt	up	3-00:00:00	408/7/65/480	qbd[001-480]
workq	up	3-00:00:00	408/7/65/480	qbd[001-480]
bigmem	up	3-00:00:00	5/0/0/5	qbd[481-485]
gpu2	up	3-00:00:00	22/30/0/52	qbd[486-511,517-542]
gpu4	up	3-00:00:00	1/9/0/10	qbd[512-516,543-547]

```
[username@hostname ~]$ sinfo -0 partitionname
```

```
PARTITION
```

```
single
```

```
ckpt
```

```
workq
```

```
bigmem
```

```
gpu2
```

```
gpu4
```


1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

```
[username@hostname ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME_LIMIT	TIME	CPUS	NODES	NODELIST(REASON)
525028	bigmem	ATAC_cell_type_heatmygong	mygong	R	3-00:00:00	9:42:02	64	1	qbd481
523481	bigmem	P150GPa	jhinz	R	3-00:00:00	1-03:13:07	128	2	qbd[484-485]
525191	bigmem	AFM	agiri	R	2-00:00:00	8:03:49	128	2	qbd[482-483]
526416	checkpt	batch_cpu_newqb4.sh	sid	PD	3-00:00:00	0:00	1024	16	(Resources)
525418	checkpt	ScAlN-MC-30	hsh012	R	3-00:00:00	7:29:15	256	4	qbd[286-289]
525199	checkpt	ScAlN-MC-50	hsh012	R	3-00:00:00	7:47:08	256	4	qbd[294-297]
525536	checkpt	ScAlN-MC-10	hsh012	R	3-00:00:00	6:18:29	256	4	qbd[350-353]
525523	checkpt	welding	collin	R	3-00:00:00	6:43:48	64	1	qbd053
524943	checkpt	welding	collin	R	3-00:00:00	10:29:00	64	1	qbd020
524931	checkpt	welding	collin	R	3-00:00:00	10:40:08	64	1	qbd015
524926	checkpt	welding	collin	R	3-00:00:00	10:43:11	64	1	qbd003
525882	checkpt	ScAlN-MC-20	hsh012	R	3-00:00:00	5:34:35	256	4	qbd[447-450]

```
[username@hostname ~]$ squeue -l
```

JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMIT	NODES	NODELIST(REASON)
50878	bigmem	submit-p	username1	RUNNING	2-06:28:02	3-00:00:00	1	qbd482
51761	checkpt	CuFe_Fit	username2	RUNNING	1:29	1-00:00:00	1	qbd081
51257	checkpt	GOM1km_e	username3	RUNNING	1-17:35:43	3-00:00:00	25	qbd[275-299]
51254	checkpt	GOM1km_e	username3	RUNNING	1-17:43:59	3-00:00:00	25	qbd[431-455]
51252	checkpt	GOM1km_e	username3	RUNNING	1-17:58:50	3-00:00:00	25	qbd[056-080]
50872	checkpt	Asph50_S	username4	RUNNING	2-07:16:01	3-00:00:00	4	qbd[088-091]
51756	gpu2	interact	username5	RUNNING	1:07:05	12:00:00	1	qbd486
51740	gpu4	submit-1	username6	RUNNING	2:38:09	3-00:00:00	1	qbd545
51731	gpu4	submit-1	username6	RUNNING	2:55:23	3-00:00:00	1	qbd544
51728	gpu4	submit-1	username6	RUNNING	3:14:49	3-00:00:00	1	qbd543
49034	workq	dynAB+cp	username7	PENDING	0:00	10:00	200	(MaxNodePerAccount)
50873	workq	qbd.sh	username8	RUNNING	2-07:06:42	3-00:00:00	84	qbd[191-274]

```
[username@hostname ~]$ squeue -u username6
```

JOBID	PARTITION	NAME	USER	ST	TIME_LIMIT	TIME	CPUS	NODES	NODELIST(REASON)
51740	gpu4	submit-1-4.sh	username6	R	3-00:00:00	2:40:00	16	1	qbd545
51731	gpu4	submit-1-4.sh	username6	R	3-00:00:00	2:57:14	16	1	qbd544

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

```
[username@hostname ~]$ ssh -X -Y username@hostname
```

```
(username@hostname) Password:
```

```
(username@hostname) TOTP Authenticator Token:
```

How to Login to the clusters

Interactive ssh

Only `ssh` access is allowed for interactive access. One would issue a command similar to the following:

```
# LSU HPC
ssh -X -Y username@mike.hpc.lsu.edu

# LONI
ssh -X -Y username@qbc.loni.org
```

The user would then be prompted for password. The `-X -Y` flags allow for `X11` forwarding to be set up automatically. [How to establish a remote X11 session](#)

For a Windows client, please look at the [MobaXterm](#) utility.

<https://www.hpc.lsu.edu/documentation/frequently-asked-questions/login/>

Links

User Profile Login: [LSU HPC](#) | [LONI HPC](#)

Open OnDemand Login: [SuperMike-III](#) | [QB3](#) | [QB4](#)

Quick Links: <https://www.hpc.lsu.edu/links/>

Article	Organization	
User Account Request	LONI	LSU
User Allocation Request(*)(\$)	LONI	LSU
User Password Reset(*)	LONI	LSU
User Profile Settings(*)	LONI	LSU
Storage Allocation Request(*)	LONI	LSU

Two-Factor Authentication

Also known as multi-factor authentication.

[View the setup and usage guide](#)

<https://www.hpc.lsu.edu/documentation/mfa/>

QB-4 at LONI (Open for general use)

02-May-2024

QB-4 is a 4.3 PetaFlop peak performance cluster with 35,008 CPU cores and 144 NVIDIA A100 GPUs, comprised of 547 compute nodes connected by 200 Gbps Infiniband fabric.

Node Specs:

compute CPUs=64 SocketsPerBoard=2 CoresPerSocket=32 ThreadsPerCore=1 RealMemory=256000 DefMemPerCPU=3920 MB
bigmem CPUs=64 SocketsPerBoard=2 CoresPerSocket=32 ThreadsPerCore=1 RealMemory=2063000 DefMemPerCPU=32100 MB
gpu2 Gres=gpu:2 CPUs=64 SocketsPerBoard=2 CoresPerSocket=32 ThreadsPerCore=1 RealMemory=514000 DefMemPerGPU=254000 MB
gpu4 Gres=gpu:4 CPUs=64 SocketsPerBoard=2 CoresPerSocket=32 ThreadsPerCore=1 RealMemory=514000 DefMemPerGPU=127000 MB

Quotas for the /home volume are enabled at 10 GB. Please do not use the /home volume for batch job I/O, use the /work volume instead. Please limit the number of files per directory to 10,000. No disk GB quotas are in effect for the /work volume, but files are purged that are older than 60 days. There are quotas on file counts. Manage your files, but it is against our policy to blatantly circumvent the purge procedure. Misconfigured jobs that are wasting CPU resources are subject to deletion.

Allocations are required. Use -A with sbatch/srun/qsub to specify your allocation.

NOTE: The same compute allocation can be used on LONI QB-2, QB-3, and QB-4
NOTE: /project and /work are shared between QB-3 and QB-4

If you have any questions or need assistance migrating your code, please contact sys-help@loni.org

----- RECENT UPDATES -----

For Two-Factor Authentication help see:

<https://www.hpc.lsu.edu/documentation/mfa/>

06Jan2026 Upgrades

- * GPU Drivers now support CUDA 13.1
- * Added hardware information to top of message of the day.
- * Upgraded Slurm scheduler version
- * Enabled rate limits to prevent abuse of client commands when accidentally looped.
Do not run client commands like srun, sinfo, squeue, etc in a loop.

```
[username@hostname ~]$ scontrol show partitions
```

```
PartitionName=workq
```

```
AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
AllocNodes=ALL Default=NO QoS=normal
DefaultTime=12:00:00 DisableRootJobs=YES ExclusiveUser=NO ExclusiveTopo=NO GraceTime=0 Hidden=NO
MaxNodes=UNLIMITED MaxTime=3-00:00:00 MinNodes=0 LLN=NO MaxCPUsPerNode=64 MaxCPUsPerSocket=UNLIMITED
Nodes=qbd[001-480]
PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=EXCLUSIVE
OverTimeLimit=NONE PreemptMode=OFF
State=UP TotalCPUs=30720 TotalNodes=480 SelectTypeParameters=NONE
JobDefaults=(null)
DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
TRES=cpu=30720,mem=120000G,node=480,billing=30720
TRESBillingWeights=CPU=1
```

```
PartitionName=bigmem
```

```
AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
AllocNodes=ALL Default=NO QoS=normal
DefaultTime=12:00:00 DisableRootJobs=YES ExclusiveUser=NO ExclusiveTopo=NO GraceTime=0 Hidden=NO
MaxNodes=UNLIMITED MaxTime=3-00:00:00 MinNodes=0 LLN=NO MaxCPUsPerNode=64 MaxCPUsPerSocket=UNLIMITED
Nodes=qbd[481-485]
PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=EXCLUSIVE
OverTimeLimit=NONE PreemptMode=OFF
State=UP TotalCPUs=320 TotalNodes=5 SelectTypeParameters=NONE
JobDefaults=(null)
DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
TRES=cpu=320,mem=10315000M,node=5,billing=320
TRESBillingWeights=CPU=1
```

LSU

High Performance Computing | Louisiana State University

☀

Q

Search

⌕K

Home

About

Documentation

Links

Resources

Training

Users

Accounts and Allocations

Help

Password security

Policy

>

Home > Users

Help

Contact Us

Getting Started

Having Trouble?

Training & Workshops

Our mission is to provide superlative service to all HPC users within the resources and time available. The broad areas of assistance that we can provide are described below. In all cases, the best ways to get in touch with us are described on the [contacts page](#).

Getting Started

To get yourself started with using HPC resources, please visit the [Documentation](#) page where you can find getting started guides for all supported clusters.

Having Trouble?

If you have trouble using either LSU or LONI HPC resource, please send an email to sys-help@hpc.lsu.edu. Please make sure that it includes the name of the cluster you are using, a description of the issue, as well output and/or error messages if there is any.

Training & Workshops

HPC @ LSU offers [workshops and training courses](#) on Linux/Unix and parallel programming for HPC systems. If there is sufficient demand, special tutorial topics can also be developed, assuming sufficient expertise can be located.

On this page

Contact Us

Getting Started

Having Trouble?

Training & Workshops

User Meetings

Personal Consultation

LSU

INFORMATION
TECHNOLOGY
SERVICES



It is time for a break.
See you in 5 minutes!

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

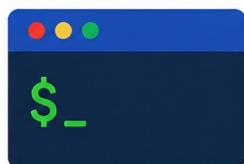
7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

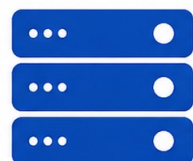
8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

Interactive Jobs



Terminal



Compute node



Can

- Run commands in real time
- Interact with the job while running
- Compile, test, and debug
- Monitor program output



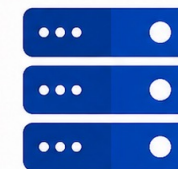
Cannot

- Continue after you disconnect
- Be used for unattended jobs
- Run for long production jobs

Batch Jobs



job_script.sh



Compute node



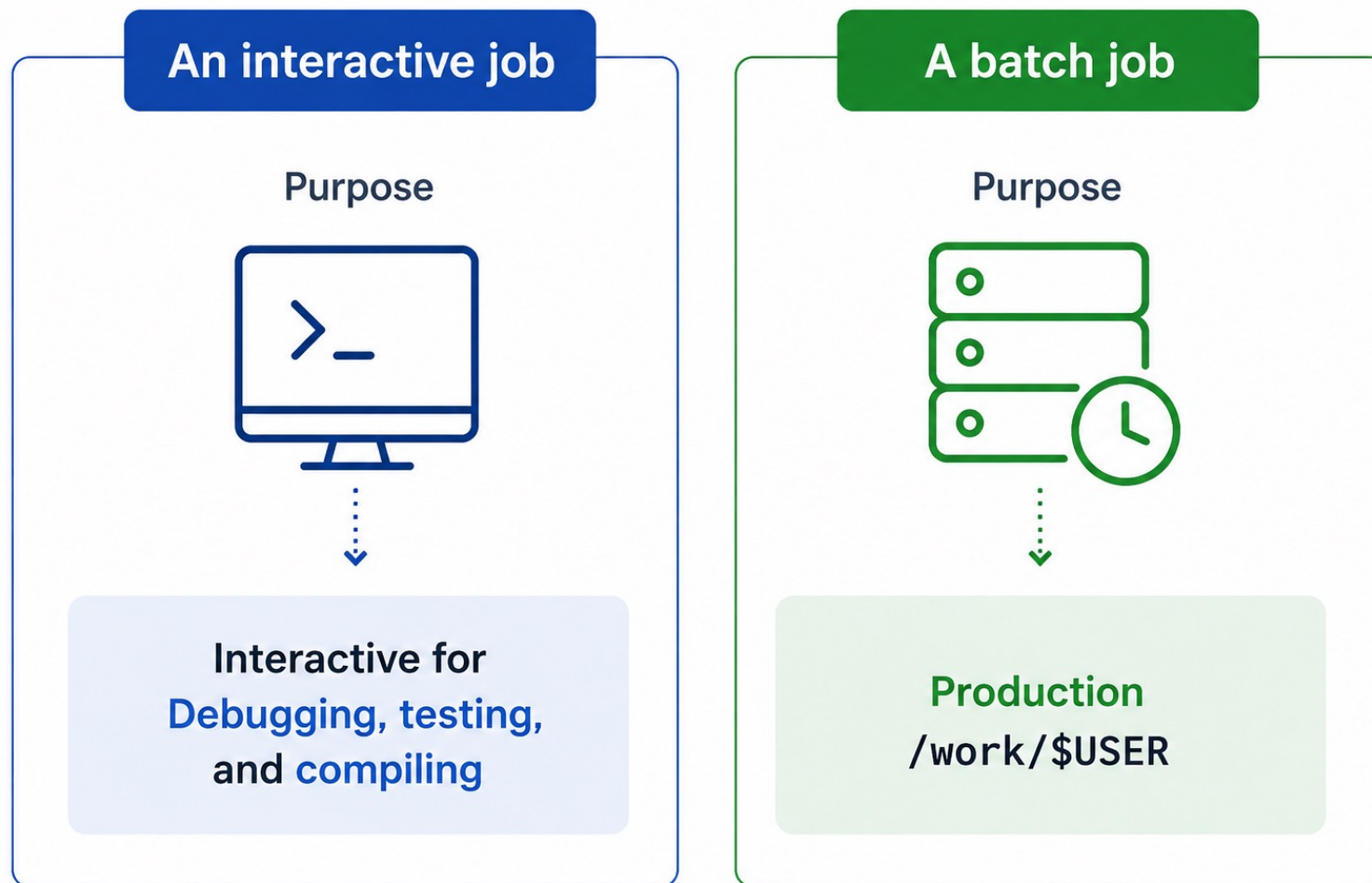
Can

- Submit to server and leave
- Runs by itself until finished or error
- Receive email notifications
- Best for long production runs



Cannot

- Interact with the job while running
- Type commands in the job
- Modify the job after submission



1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

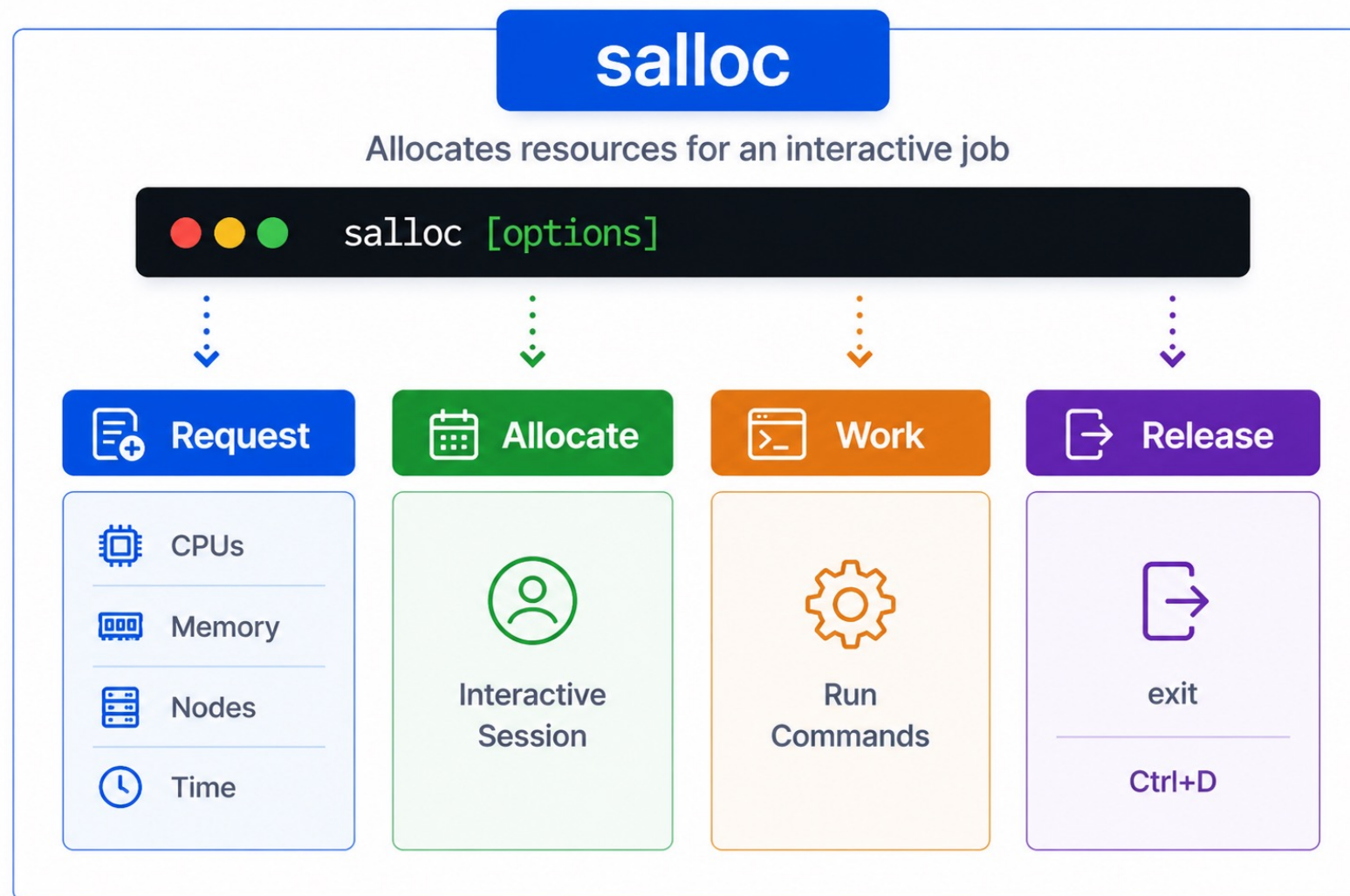
- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

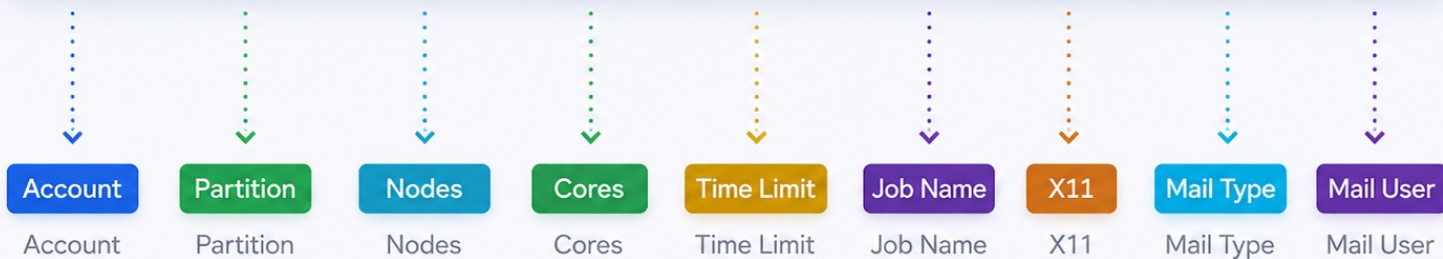


salloc

Allocates resources for an interactive job

```
●●● salloc [options] --x11 --mail-type=ALL --mail-user=user@mail.address
```

```
salloc -A allocation_name -p partition_name -N number_of_nodes -n number_of_cores  
-t DD-HH:MM:SS -J job_name --x11 --mail-type=ALL --mail-user=user@mail.address
```



Interactive Jobs

https://slurm.schedmd.com/faq.html#sbatch_srun

<https://slurm.schedmd.com/faq.html#prompt>

To start an interactive job, use the `salloc` command similar to the example below:

```
salloc -t 1:00:00 -n8 -N1 -A your_allocation_name -p single
```



Similar to the batch job script, the `-n` denotes 8 tasks (cores), the `-N` denotes 1 compute node. The complete form of the above command can be:

```
salloc --time=1:00:00 --ntasks=8 --nodes=1 --account=your_allocation_name --partition=sing
```

Note:

If an interactive job session is submitted to a partition other than single, the `-n` or `--ntasks` flag will be ignored and one or more entire nodes will be allocated to the job.

<https://www.hpc.lsu.edu/documentation/job-submission/quick-start/>

```
[username@host ~]$ salloc -A allocation_name -p single -N 1 -n 1 --time=01:00:00
```

```
salloc: Pending job allocation 51814  
salloc: lua: Submitted job 51814  
salloc: job 51814 queued and waiting for resources  
salloc: job 51814 has been allocated resources  
salloc: Granted job allocation 51814  
salloc: Waiting for resource configuration  
salloc: Nodes qbd085 are ready for job
```

```
[username@host ~]$ salloc -A allocation_name -p workq -N 1 -n64 --time=01:00:00
```

```
salloc: Pending job allocation 51814  
salloc: lua: Submitted job 51814  
salloc: job 51814 queued and waiting for resources  
salloc: job 51814 has been allocated resources  
salloc: Granted job allocation 51814  
salloc: Waiting for resource configuration  
salloc: Nodes qbd085 are ready for job
```

```
[username@qbd1 ~]$ salloc -A allocation_name -p workq -N1 -n 64 --time=12:00:00
```

```
[username@qbd085 ~]$
```



Successfully started: on a computing node (**3-digit** number)


```
[username@qbd1 Tests]$ salloc -A allocation_name -p workq -N1 -n64 -t 12:00:00
```

```
salloc: Pending job allocation 51860  
salloc: lua: Submitted job 51860  
salloc: job 51860 queued and waiting for resources  
salloc: job 51860 has been allocated resources  
salloc: Granted job allocation 51860  
salloc: Nodes qbd083 are ready for job  
[username@qbd083 Tests]$
```



Job starts in **where the job was submitted**

```
[username@qbd1 Tests]$ salloc -A allocation_name -p workq -N1 -n64 -t 12:00:00
```

```
salloc: Pending job allocation 51860  
salloc: lua: Submitted job 51860  
salloc: job 51860 queued and waiting for resources  
salloc: job 51860 has been allocated resources  
salloc: Granted job allocation 51860  
salloc: Nodes qbd083 are ready for job  
[username@qbd083 Tests]$
```



Once a job starts, **type and run commands** as you normally do.

Running Jobs on an Interactive Node

Use your allocated interactive session to run applications



Serial (Single-thread)

Run commands as you normally do



`<Executable> [options]`

Executable

Your program or command

Options

Command-line options
and arguments



Parallel (MPI)

Method 1

(Recommended)



`srun -N[...] -n[...] -c[...] <mpi_executable> [options]`

-N

Number of
nodes

-n

Number of
tasks
(MPI ranks)

-c

CPUs per
task

MPI Executable

Your MPI
program

Options

Command-line options
and arguments

Method 2



```
1 module load <desired MPI>
2 export OMP_NUM_THREADS=[...]
3 mpirun -np [...] <mpi_executable> [options]
```

Load MPI

Load the MPI module
you want to use

Set Threads

Set number of OpenMP
threads per task

Run with MPI

Launch MPI program
with the specified ranks

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

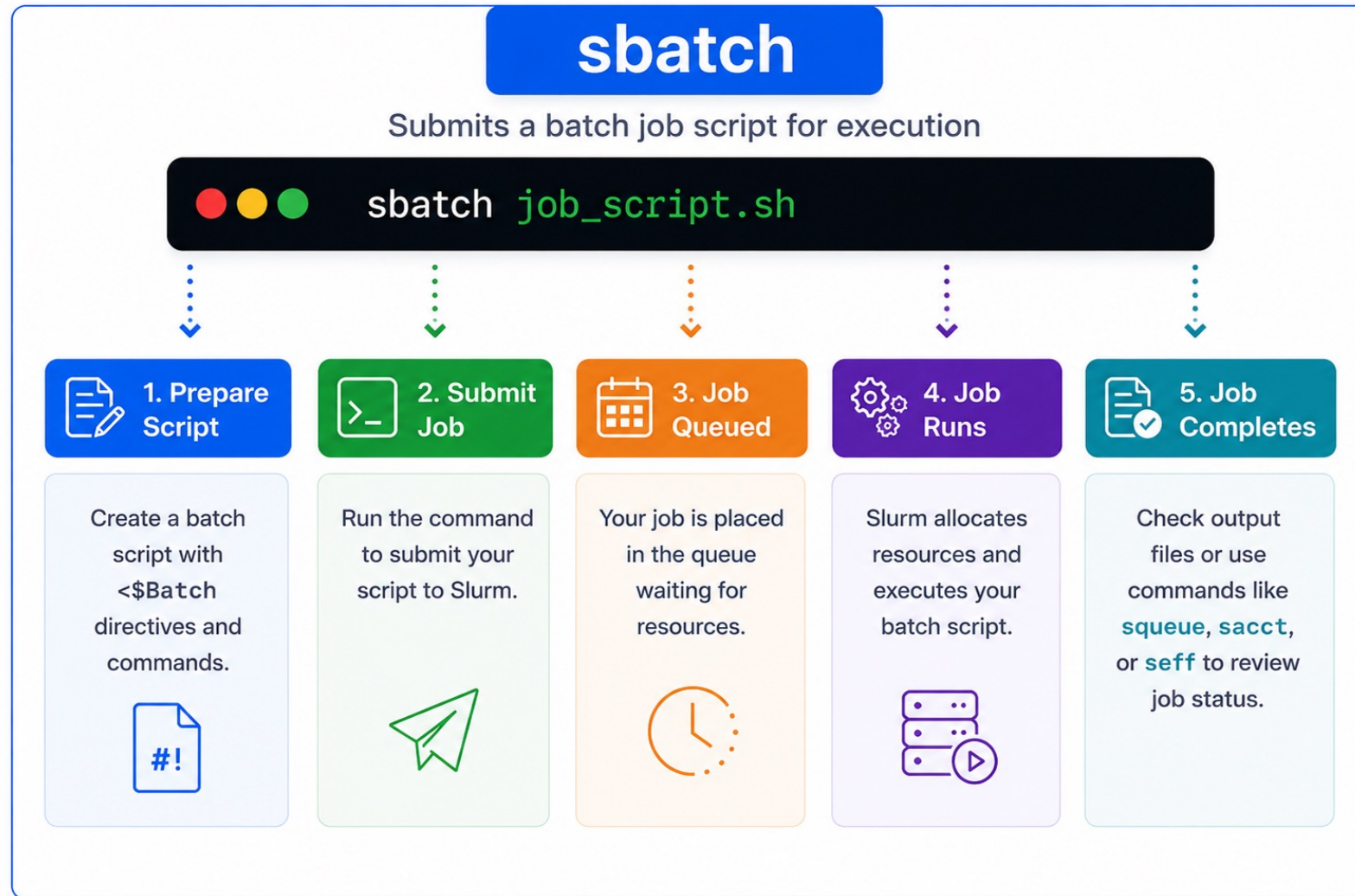
- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**



Batch Script Structure

A batch script has two main parts: header (parameters) and body (commands)

1. HEADER – Parameters (Directives)

Tells the scheduler what resources and settings are needed

```
#!/bin/bash
#SBATCH -A allocation_name      # Allocation/Project account
#SBATCH -p partition_name      # Partition to use
#SBATCH -N number_of_nodes     # Number of nodes
#SBATCH -n number_of_tasks     # Total number of tasks (MPI processes)
#SBATCH -c number_of_CPU_cores_per_task # CPU cores per task
#SBATCH -t DD-HH:MM:SS         # Time limit (DD-HH:MM:SS format)
#SBATCH -J job_name            # Name of the job
#SBATCH -o standard.out        # Standard output file
#SBATCH -e standard.err        # Standard error file
#SBATCH --mail-user=user@mail.address # Email for notifications
#SBATCH --mail-type=ALL        # Mail events (BEGIN, END, FAIL, ALL)
```



Header Specifies:

- Account (project allocation)
- Partition
- Number of nodes
- Number of tasks (MPI processes)
- CPU cores per task
- Time limit
- Job name
- Output file (standard output)
- Error file (standard error)
- Email address for notifications
- Email type (ALL, BEGIN, END, FAIL)

2. BODY – Commands

Commands to run on the allocated resources

```
# Load required modules
module purge
module load gcc/12.2.0
module load openmpi/4.1.5

# Set environment variables
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK # Match OpenMP threads to CPUs per task

# Change to working directory (optional)
cd $SLURM_SUBMIT_DIR

# Run your application
srun ./my_program.x -in input.dat -out output.dat
```



Body Contains:

- Load modules and software
- Set environment variables
- Change directories
- Run your applications
- Any other commands needed

Example Batch Script (job_script.sh)

```
##### 1. HEADER (PARAMETERS/DIRECTIVES) #####
#!/bin/bash

#SBATCH -A allocation_name      # Allocation/Project account
#SBATCH -p partition_name      # Partition to use
#SBATCH -N number_of_nodes     # Number of nodes
#SBATCH -n number_of_tasks     # Total number of tasks (MPI processes)
#SBATCH -c number_of_CPU_cores_per_task # CPU cores per task
#SBATCH -t DD-HH:MM:SS         # Time limit (DD-HH:MM:SS) format
#SBATCH -J job_name            # Name of the job
#SBATCH -o standard.out        # Standard output file
#SBATCH -e standard.err        # Standard error file
#SBATCH --mail-user=user@mail.address # Email for notifications
#SBATCH --mail-type=ALL        # Mail events (BEGIN, END, FAIL, ALL)

##### 2. BODY (COMMANDS) #####

# Load required modules
module purge
module load gcc/12.2.0
module load openmpi/4.1.5

# Set environment variables
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK # Match OpenMP threads to CPUs per task

# Print job and system information
echo "Job Name      : $SLURM_JOB_NAME"
echo "Job ID       : $SLURM_JOBID"
echo "Nodes        : $SLURM_JOB_NUM_NODES"
echo "Tasks         : $SLURM_NTASKS"
echo "CPUs/task      : $SLURM_CPUS_PER_TASK"
echo "Time Limit    : $SLURM_TIMELIMIT"
echo "Working Dir   : $SLURM_SUBMIT_DIR"
echo "-----"

# Change to working directory (optional)
cd $SLURM_SUBMIT_DIR

# Run your application
srun ./my_program.x -in input.dat -out output.dat

# End of batch script
echo "Job completed."
```

Example Batch Script (job_script.sh)

```
##### 1. HEADER (PARAMETERS/DIRECTIVES) #####
#!/bin/bash

#SBATCH --account=allocation_name      # Allocation/Project account
#SBATCH --partition=partition_name     # Partition to use
#SBATCH --nodes=number_of_nodes       # Number of nodes
#SBATCH --ntasks=number_of_tasks      # Total number of tasks (MPI processes)
#SBATCH --cpu-per-task=number_of_CPU_cores_per_task # CPU cores per task
#SBATCH --time=DD-HH:MM:SS            # Time limit (DD-HH:MM:SS) format
#SBATCH --job-name=job_name           # Name of the job
#SBATCH --output=standard.out          # Standard output file
#SBATCH --error=standard.err           # Standard error file
#SBATCH --mail-user=user@mail.address  # Email for notifications
#SBATCH --mail-type=ALL                # Mail events (BEGIN, END, FAIL, ALL)

##### 2. BODY (COMMANDS) #####

# Load required modules
module purge
module load gcc/12.2.0
module load openmpi/4.1.5

# Set environment variables
export OMP_NUM_THREADS=number_of_CPU_cores_per_task # Match OpenMP threads to CPUs per task

# Print job and system information
echo "Job Name      : $SLURM_JOB_NAME"
echo "Job ID       : $SLURM_JOBID"
echo "Nodes        : $SLURM_JOB_NUM_NODES"
echo "Tasks         : $SLURM_NTASKS"
echo "CPUs/task      : $SLURM_CPUS_PER_TASK"
echo "Time Limit    : $SLURM_TIMELIMIT"
echo "Working Dir   : $SLURM_SUBMIT_DIR"
echo "-----"

# Change to working directory (optional)
cd $SLURM_SUBMIT_DIR

# Run your application
srun ./my_program.x -in input.dat -out output.dat

# End of batch script
echo "Job completed."
```


Key SLURM Environment Variables

Automatically set by Slurm for your job

```
echo $SLURM_JOBID $SLURM_SUBMIT_DIR $SLURM_JOB_NODELIST $SLURM_NNODES $SLURM_NTASKS
```

\$SLURM_JOBID



Unique ID of your job.

Example: 1234567

Use it to reference your job in commands and files.

\$SLURM_SUBMIT_DIR



Directory where you submitted the job.

Example: /home/user/project

Useful for accessing input files and paths.

\$SLURM_JOB_NODELIST



List of nodes allocated to your job.

Example: node[01-02,05]

Shows the specific nodes you are running on.

\$SLURM_NNODES



Total number of nodes allocated to your job.

Example: 2

Use in scripts to adapt to the number of nodes.

\$SLURM_NTASKS



Total number of tasks (MPI processes) allocated.

Example: 64

Use to configure MPI applications and parallel runs.

Submit a Batch Job

To create a batch Slurm script, use your favorite editor (e.g. vi or emacs) to create a text file containing both Slurm instructions and commands to run your job. All Slurm directives (special instructions) are prefaced by `#SBATCH`. The following is an example of a Slurm batch job script:

```
#!/bin/bash
#SBATCH -N 1           # request one node
#SBATCH -t 2:00:00     # request two hours
#SBATCH -p single      # in single partition (queue)
#SBATCH -A your_allocation_name
#SBATCH -o slurm-%j.out-%N # optional, name of the stdout, using the job number (%j) and t
#SBATCH -e slurm-%j.err-%N # optional, name of the stderr, using job and hostname values
# below are job commands
date

# Set some handy environment variables.

export HOME_DIR=/home/$USER/myjob
export WORK_DIR=/work/$USER/myjob

# Make sure the WORK_DIR exists:
mkdir -p $WORK_DIR
# Copy files, jump to WORK_DIR, and execute a program called "mydemo"
cp $HOME_DIR/mydemo $WORK_DIR
cd $WORK_DIR
./mydemo
# Mark the time it finishes.
date
# exit the job
exit 0
```

<https://www.hpc.lsu.edu/documentation/job-submission/quick-start/#submit-a-batch-job>

Example: Reserving a GPU Node

This script reserves 1 GPU node with 2 GPUs and runs LAMMPS using GPU acceleration.

1. HEADER – Parameters (Directives)

Tells the scheduler what resources and settings are needed

```
1 #!/bin/bash
2 #SBATCH -p gpu4                # Use partition 'gpu4'
3 #SBATCH --gres=gpu:2           # Request 2 GPUs
4 #SBATCH -N 1                   # 1 node
5 #SBATCH -n 32                  # Total number of MPI tasks
6 #SBATCH -c 1                   # One CPU core per task
7 #SBATCH -t HH:MM:SS            # Time limit (e.g., 02:00:00)
8 #SBATCH -A allocation_name     # Allocation / project account
9 #SBATCH -J job_name            # Job name
10 #SBATCH -o standard-output.%j.out # Standard output file
11 #SBATCH -e standard-error.%j.err  # Standard error file
12 #SBATCH --mail-user=user@email.address # Email for notifications
13 #SBATCH --mail-type=ALL         # Mail on all events
```



Header Specifies:

- Partition: gpu4
- GPUs: 2
- Nodes: 1
- Tasks: 32 MPI tasks
- CPUs per task: 1
- Time limit: as specified
- Account: allocation_name
- Job name: job_name
- Output: standard-output.%j.out
- Error: standard-error.%j.err
- Email: user@email.address (ALL events)

2. BODY – Commands

Commands to run on the allocated resources

```
1 module purge
2 module load lammmps/29Aug2024/intel-2021.5.0-intel-mpi-2021.5.1
3
4 export OMP_NUM_THREADS=1      # One core per task
5
6 echo "Current working directory: $SLURM_SUBMIT_DIR"
7 cd $SLURM_SUBMIT_DIR
8 time srun -N 1 -n 32 lmp_mpi -sf gpu -pk gpu 2 neigh yes newton off -in lj.in > lj.out
```



Body Contains:

- Load required modules (LAMMPS with Intel MPI)
- Set OpenMP threads = 1 (one core per task)
- Print working directory
- Change to submission directory
- Run LAMMPS with GPU acceleration (2 GPUs, 32 MPI tasks)

```
#!/bin/bash

#SBATCH -p gpu4           # Use partition 'gpu4'
#SBATCH --gres=gpu:2      # Request 2 GPUs
#SBATCH -N 1              # 1 node
#SBATCH -n 32             # Total number of MPI tasks
#SBATCH -c 1              # One CPU core per task
#SBATCH -t HH:MM:SS       # Time limit (e.g., 02:00:00)
#SBATCH -A allocation_name # Allocation / project account
#SBATCH -J job_name       # Job name
#SBATCH -o standard.out   # Standard output file
#SBATCH -e standard.err   # Standard error file
#SBATCH --mail-user=user@email.address # Email for notifications
#SBATCH --mail-type=ALL    # Mail on all events (BEGIN, END, FAIL, ALL)

module purge
module load lammps/29Aug2024/intel-2021.5.0-cuda-12.2.1-intel-mpi-2021.5.1

export OMP_NUM_THREADS=1      # One core per task

echo "Current working directory: $SLURM_SUBMIT_DIR"
cd $SLURM_SUBMIT_DIR

time srun -N 1 -n 32 lmp_mpi -sf gpu -pk gpu 2 neigh yes newton off -in lj.in > lj.out
```

Submit a Batch File in the Cluster

Prepare your batch script and submit it to the scheduler

```
#!/bin/bash
#SBATCH --account=allocation_name
#SBATCH --job-name=job_name
#SBATCH --nodes=1
#SBATCH --ntasks=32
#SBATCH --time=01:00:00
#SBATCH --partition=single
#SBATCH --output=job.out
#SBATCH --error=job.err
...
# Your job script commands
```



Batch File

(parameters, job scripts)

A text file containing SBATCH directives and the commands to run your job.

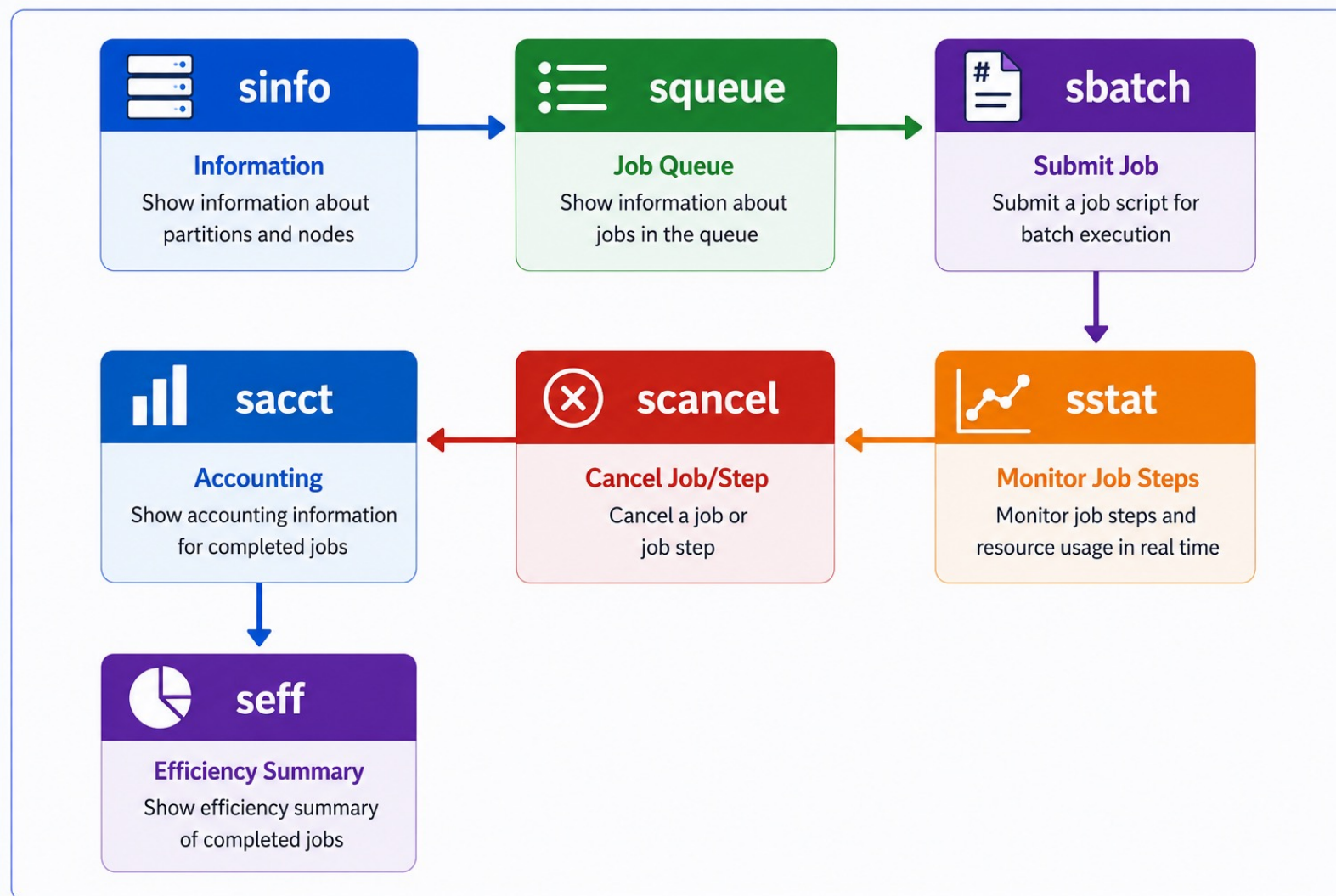


```
$ sbatch job_script.sh
```



Submission Command

Use the **sbatch** command to submit your batch file to the scheduler.



1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

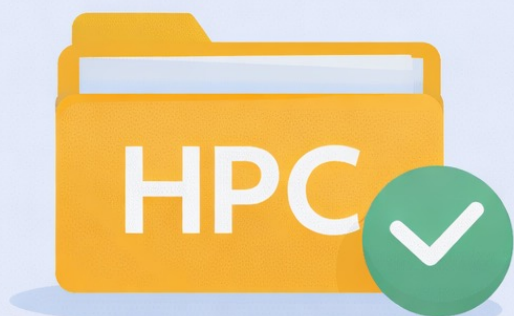
7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

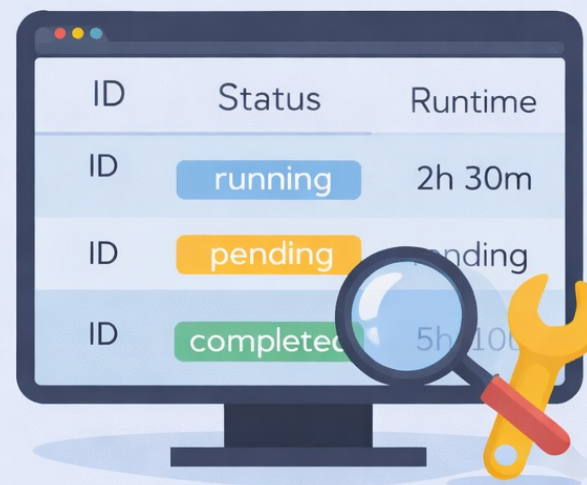
8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

Running Jobs on HPC **≠** “Submit and done”



“Submit and done”



Monitoring &
Managing Jobs

Monitoring and managing jobs are part of the work

MONITORING ON AN HPC CLUSTER (SLURM-BASED)

On an HPC cluster (especially Slurm-based systems), there are several levels of monitoring:

USEFUL COMMANDS (on a compute node)

top	Real-time CPU and memory usage
free -h	Memory usage summary
df -h	Disk usage summary
uptime	System load and uptime

top (CPU)

```
top - 12:34:56 up 5 days, 2:15, 64 users, load average: 4.65, 4.32, 3.91
Tasks: 352 total, 4 running, 348 sleeping, 0 stopped, 0 zombie
%Cpu(s): 82.1 us, 10.5 sy, 0.0 ni, 7.1 id, 0.0 wa, 0.0 hi, 0.3 si, 0.0 st
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
12345	user	20	0	8.5g	2.1g	30m	R	98.7	2.7	12:34.56	vasp_std
12346	user	20	0	8.5g	2.0g	30m	R	97.9	2.6	12:34.40	vasp_std
12347	user	20	0	8.4g	2.0g	30m	R	97.5	2.6	12:33.81	vasp_std
23456	user	20	0	1.2g	108m	20m	S	1.3	0.1	0:10.21	python

free -h (Memory)

```
free -h
```

	total	used	free	shared	buff/cache	available
Mem:	251G	218G	12G	2.0G	29G	30G
Swap:	0B	0B	0B			

df -h (Disk)

```
df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3       1.8T  1.2T  556G   69% /-crash
/dev/sda2       180G   45G   50G   48% /
/dev/sda1       2.0T  1.1T  900G   56% /work
```


USEFUL COMMANDS (from login node)

ssat -j <jobid>	Real-time job statistics
sacct -j <jobid>	Job accounting (after completion)
seff <jobid>	Job efficiency summary
scontrol show job <jobid>	Detailed job information
squeue -u \$USER	Your jobs in the queue

sstat -j 123456 (real-time)

JobID	Elapsed	AveCPU	MaxRSS	MaxVMSize	NTasks	NCPUS
123456	01:10:23	248.2	95.4G	120.1G	32	64

sacct -j 123456 (after completion)

JobID	JobName	State	Elapsed	TotalCPU	MaxRSS	AveCPU
123456	vasp_job	COMPLETED	01:12:05	02:18:40	96.1G	92.3%

seff 123456 (job efficiency)

```
Job ID: 123456
Cluster: supercluster
User/Group: user/group
State: COMPLETED (exit code 0)
Cores: 64
CPU Utilized: 02:18:40
CPU Efficiency: 92.31% of 02:30:00 core-walltime
Job Wall-clock time: 01:12:05
Memory Utilized: 96.1 GB
Memory Efficiency: 80.08% of 120.0 GB
```

USEFUL COMMANDS (on GPU node)

nvidia-smi	GPU status summary
nvidia-smi -l 1	Update every 1 second
nvidia-smi topo -m	GPU topology
nvidia-smi pmon -c 1	GPU process monitoring

EXAMPLE OUTPUT (nvidia-smi)

Thu Jul 17 12:34:56 2025

NVIDIA-SMI 550.54.14 Driver Version: 550.54.14 CUDA Version: 12.4							Processes:						
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC		GPU	GI	CI	PID	Type	Process name	GPU Memory
Fan	Temp	Perf	Par:Usage/Cap	Memory-Usage	GPU-Util	Compute M.		ID	ID				Usage
						MIG M.							
0	NVIDIA	A100-S4M-40GB	On	00000000:65:00:00	Off	0	0	N/A	N/A	12345	C	/usr/bin/vasp_std	25.1GiB
N/A	72C	P0	285W / 400W	28.3GiB / 40.0GiB	92%	Default	0	N/A	N/A	12346	C	/usr/bin/vasp_std	2.8GiB
							0	N/A	N/A	23456	C	python	0.3GiB

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

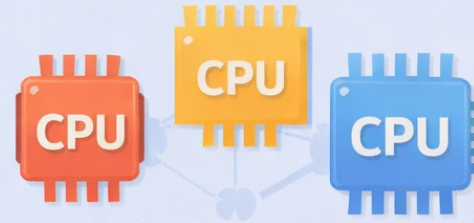
8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

Cores Utilization



A job requesting n cores

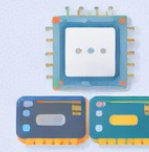


A job utilizing n cores



Goal

- Use the allocated resources (CPU cores, RAM, time, ...) as fully and efficiently as possible
- No serious underutilizing
- No serious overutilizing



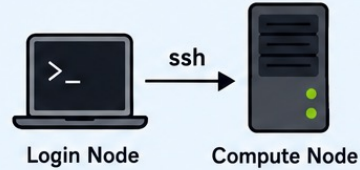
Things to check

- CPU / GPU load
- Memory usage

top – Real-time View of System Performance and Running Processes

How to use

- 1 Submit a job to a compute node.
- 2 SSH to the compute node while your job is running
(SSH access is only available when you have an active job on that node).
- 3 Run: **top**



What it shows

- ✓ CPU and memory usage
- ✓ System load averages
- ✓ Uptime
- ✓ List of active processes
- ✓ Helps identify processes using the most CPU or memory

```
top - 11:42:35 up 2:15,  2 users,  load average: 1.25, 0.98, 0.76
Tasks: 156 total,  2 running, 154 sleeping,  0 stopped,  0 zombie
%Cpu(s): 45.3 us, 12.1 sy,  0.0 ni, 42.3 id,  0.1 wa,  0.1 hi,  0.1 si,  0.0 st
MiB Mem : 128808.6 total, 24536.7 free,  51234.1 used,  52837.8 buff/cache
MiB Swap:  8192.0 total,  8192.0 free,    0.0 used.  70123.7 avail Mem

  PID USER  PR  NI  VIRT  RES  SHR  S  %CPU  %MEM  TIME+  COMMAND
12345 user1  20   0 3923456 512344 23456 R  85.3   0.4   1:23.45 my_program
12346 user1  20   0 1623456 123456 12345 R  45.1   0.1   0:45.12 python
12347 user1  20   0  823456  65432 12345 S  10.2   0.1   0:12.33 worker
12348 user1  20   0 2245678 234567 23456 S   5.3   0.2   0:22.11 Rscript
12349 user1  20   0  512356  45321  6789 S   1.3   0.0   0:03.21 bash
12350 user1  20   0 305432  23456  5678 S   0.3   0.0   0:01.23 sshd
```

System load averages
(1, 5, 15 min)

CPU usage
(user, system, idle, etc.)

Memory usage
(total, used, free, cache)

Active processes
(sorted by CPU usage and memory)

 **Tip:** Press **q** to quit **top**.

- ✓ Displays a dynamic, real-time view of a compute node.
- ✓ SSH access only when you have an active job on that node.

- ✓ Must be run on a compute node.

```
top - 02:23:58 up 278 days, 19:17, 2 users, load average: 63.63, 39.81, 17.49
Tasks: 981 total, 65 running, 916 sleeping, 0 stopped, 0 zombie
%Cpu(s): 90.2 us, 9.2 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.5 hi, 0.0 si, 0.0 st
MiB Mem : 257004.8 total, 211261.0 free, 41926.9 used, 3816.9 buff/cache
MiB Swap: 16641.0 total, 16580.7 free, 60.2 used. 212737.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2701318	jasonli3	20	0	595668	582356	2568	R	100.0	0.2	4:08.94	TDSE_np3_e0
2701342	jasonli3	20	0	595668	581944	2616	R	100.0	0.2	4:08.90	TDSE_np3_e0
2701249	jasonli3	20	0	595668	581792	2464	R	99.7	0.2	4:08.97	TDSE_np3_e0
2701252	jasonli3	20	0	595668	514684	2520	R	99.7	0.2	4:09.00	TDSE_np3_e0
2701261	jasonli3	20	0	595668	393828	2616	R	99.7	0.1	4:08.97	TDSE_np3_e0
2701264	jasonli3	20	0	595668	581856	2532	R	99.7	0.2	4:08.92	TDSE_np3_e0
2701270	jasonli3	20	0	595668	582480	2432	R	99.7	0.2	4:08.95	TDSE_np3_e0
2701273	jasonli3	20	0	595668	581776	2448	R	99.7	0.2	4:08.81	TDSE_np3_e0
2701276	jasonli3	20	0	595668	582160	2568	R	99.7	0.2	4:08.98	TDSE_np3_e0
2701279	jasonli3	20	0	595668	222064	2644	R	99.7	0.1	4:08.98	TDSE_np3_e0

What to look at ...	Normal behavior ...	You should be concerned if ...
---------------------	---------------------	--------------------------------

```
top - 02:23:58 up 278 days, 19:17, 2 users, load average: 63.63, 39.81, 17.49
Tasks: 981 total, 65 running, 916 sleeping, 0 stopped, 0 zombie
%Cpu(s): 90.2 us, 9.2 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.5 hi, 0.0 si, 0.0 st
MiB Mem : 257004.8 total, 211261.0 free, 41926.9 used, 3816.9 buff/cache
MiB Swap: 16641.0 total, 16580.7 free, 60.2 used. 212737.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2701318	jasonli3	20	0	595668	582356	2568	R	100.0	0.2	4:08.94	TDSE_np3_e0
2701342	jasonli3	20	0	595668	581944	2616	R	100.0	0.2	4:08.90	TDSE_np3_e0
2701249	jasonli3	20	0	595668	581792	2464	R	99.7	0.2	4:08.97	TDSE_np3_e0
2701252	jasonli3	20	0	595668	514684	2520	R	99.7	0.2	4:09.00	TDSE_np3_e0
2701261	jasonli3	20	0	595668	393828	2616	R	99.7	0.1	4:08.97	TDSE_np3_e0
2701264	jasonli3	20	0	595668	581856	2532	R	99.7	0.2	4:08.92	TDSE_np3_e0
2701270	jasonli3	20	0	595668	582480	2432	R	99.7	0.2	4:08.95	TDSE_np3_e0
2701273	jasonli3	20	0	595668	581776	2448	R	99.7	0.2	4:08.81	TDSE_np3_e0
2701276	jasonli3	20	0	595668	582160	2568	R	99.7	0.2	4:08.98	TDSE_np3_e0
2701279	jasonli3	20	0	595668	222064	2644	R	99.7	0.1	4:08.98	TDSE_np3_e0

What to look at ...	Normal behavior ...	You should be concerned if ...
Load average	Close to allocated number of cores on the node	Consistently too low or too high

```
top - 02:23:58 up 278 days, 19:17, 2 users, load average: 63.63, 39.81, 17.49
Tasks: 981 total, 65 running, 916 sleeping, 0 stopped, 0 zombie
%Cpu(s): 90.2 us, 9.2 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.5 hi, 0.0 si, 0.0 st
MiB Mem : 257004.8 total, 211261.0 free, 41926.9 used, 3816.9 buff/cache
MiB Swap: 16641.0 total, 16580.7 free, 60.2 used. 212737.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2701318	jasonli3	20	0	595668	582356	2568	R	100.0	0.2	4:08.94	TDSE_np3_e0
2701342	jasonli3	20	0	595668	581944	2616	R	100.0	0.2	4:08.90	TDSE_np3_e0
2701249	jasonli3	20	0	595668	581792	2464	R	99.7	0.2	4:08.97	TDSE_np3_e0
2701252	jasonli3	20	0	595668	514684	2520	R	99.7	0.2	4:09.00	TDSE_np3_e0
2701261	jasonli3	20	0	595668	393828	2616	R	99.7	0.1	4:08.97	TDSE_np3_e0
2701264	jasonli3	20	0	595668	581856	2532	R	99.7	0.2	4:08.92	TDSE_np3_e0
2701270	jasonli3	20	0	595668	582480	2432	R	99.7	0.2	4:08.95	TDSE_np3_e0
2701273	jasonli3	20	0	595668	581776	2448	R	99.7	0.2	4:08.81	TDSE_np3_e0
2701276	jasonli3	20	0	595668	582160	2568	R	99.7	0.2	4:08.98	TDSE_np3_e0
2701279	jasonli3	20	0	595668	222064	2644	R	99.7	0.1	4:08.98	TDSE_np3_e0

What to look at ...	Normal behavior ...	You should be concerned if ...
Load average	Close to allocated number of cores on the node	Consistently too low or too high
Memory usage (not virtual memory)	Does not exceed total allocated memory	Exceeds total allocated memory

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

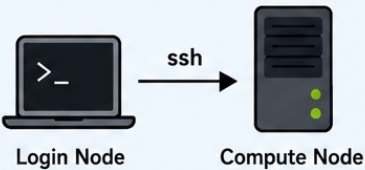
8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

free – Check Memory (RAM) and Swap Usage

What it does

The **free** command displays the system's memory usage, including total, used, free, shared, cached, and available RAM, as well as swap memory.



Why use it?

- ✓ Shows RAM and swap memory usage.
- ✓ Displays used and available memory.
- ✓ Helps monitor system memory.



Usage: `free -h` (Human-readable format)

```
[user@node ~]$ free -h
```

Example Output

	total	used	free	shared	buff/cache	available
Memory (RAM)	251Gi	123Gi	24Gi	2.1Gi	103Gi	110Gi
Swap Memory	16Gi	1.2Gi	14.8Gi	-	-	15Gi

Total installed RAM

Currently used RAM

Currently free RAM

Memory used by tmpfs/shm

Used for buffers and cache

Available RAM for new applications

Total swap memory

Used swap memory

Free swap memory

Available swap memory

Example interpretation

From the example above:

- Total RAM: 251 GiB
- Used RAM: 123 GiB
- Free RAM: 24 GiB
- Available RAM: 110 GiB
- Total Swap: 16 GiB
- Used Swap: 1.2 GiB
- Available Swap: 15 GiB



Good to know

- If available memory is low, the system may slow down.
- Swap is used when RAM is full.



Tip: Use `free -h` for human-readable output (KB, MB, GB).

Press **q** to quit the output.

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login In to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

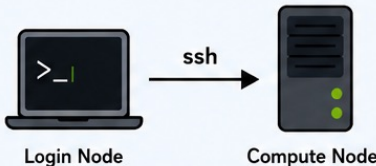
- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

qshow – Show Detailed Usage of a Running PBS Job

What it does

The **qshow** command shows detailed usage information for a running PBS job on each node, including:

- Load averages, CPU usage, number of processes running, and per-process memory usage.
- A summary line at the end with overall job statistics.



Why use it?

- ✓ Shows detailed resource usage of a running PBS job.
- ✓ Helps identify which processes are using the most CPU and memory.
- ✓ Useful for performance monitoring and troubleshooting.



Usage: `qshow <job_id>` Example: `qshow 916643`

```
[user@login ~]$ qshow 916643
PBS job: 916643, nodes: 1

Hostname    Days    Load    CPU    U#    (User:Process:VirtualMemory:Memory:Hours)
qbd163      66     66.48   2397   68   user:vasp_std:4272M:826M:57.5 user:vasp_std:4271M:817M:57.5 ...

PBS_job=916643 user=user allocation=loni_mat_bio24 queue=workq total_load=66.48
cpu_hours=3703.35 wall_hours=58.93 unused_nodes=0 total_nodes=1 ppn=64
avg_load=66.48 avg_cpu=2397% avg_mem=52906mb avg_vmem=273434mb
top_proc=user:vasp_std:qbd163:4271M:817M:57.5hr:40% node_processes=68
```

Job and node info
PBS job ID and number of nodes allocated.

Per-node summary
Hostname, days running, load average, CPU usage, and number of processes (U#).

Process details
User:Process:VirtualMemory:Memory:Hours for each running process on the node.

Job summary
Overall job statistics across all nodes.

Column / Field Explanation		U#	Number of user processes running on the node.
Hostname	Name of the compute node.	User	User who owns the process.
Days	Number of days the job has been running.	Process	Name of the process.
Load	1-minute load average on the node.	VirtualMemory	Virtual memory used by the process (in MB).
CPU	Total CPU usage (%) on the node.	Memory	Physical memory (RSS) used by the process (in MB).
		Hours	CPU hours consumed by the process.

Good to know

- High load (> 100 cores worth) or high CPU% may indicate heavy CPU usage.
- Check the top_proc field to find the process using the most memory.
- Useful for identifying memory-hungry or long-running processes.

★ **Tip:** Use `qshow <job_id>` while the job is running to monitor real-time resource usage on all nodes.

Press **q** to quit.


```
(base) [jasonli3@mike4 ~]$ qshow 38581
PBS job: 38581, nodes: 1
Hostname Days Load CPU U# (User:Process:VirtualMemory:Memory:Hours)
mike145    278 64.12 6033 68 yxan:tmp_mik+:524M:104M:13.5 yxan:tmp_mik+:524M:104M:13.5 yxan:tmp_mik+:533M:107M:13.5 yxan:tmp_mik+:748M:128M:13.5
yxan:tmp_mik+:738M:124M:13.5 yxan:tmp_mik+:520M:104M:13.5 yxan:tmp_mik+:587M:109M:13.5 yxan:tmp_mik+:743M:128M:13.5 yxan:tmp_mik+:696M:118M:13.5
yxan:tmp_mik+:528M:101M:13.5 yxan:tmp_mik+:578M:108M:13.5 yxan:tmp_mik+:528M:105M:13.5 yxan:tmp_mik+:528M:106M:13.5 yxan:tmp_mik+:520M:105M:13.5
yxan:tmp_mik+:561M:106M:13.5 yxan:tmp_mik+:583M:109M:13.5 yxan:tmp_mik+:520M:103M:13.5 yxan:tmp_mik+:524M:103M:13.5 yxan:tmp_mik+:738M:125M:13.5
yxan:tmp_mik+:709M:119M:13.5 yxan:tmp_mik+:524M:103M:13.5 yxan:tmp_mik+:574M:107M:13.5 yxan:tmp_mik+:697M:121M:13.5 yxan:tmp_mik+:658M:115M:13.5
yxan:tmp_mik+:528M:102M:13.5 yxan:tmp_mik+:557M:108M:13.5 yxan:tmp_mik+:524M:105M:13.5 yxan:tmp_mik+:524M:105M:13.5 yxan:tmp_mik+:515M:102M:13.5
yxan:tmp_mik+:520M:104M:13.5 yxan:tmp_mik+:567M:108M:13.5 yxan:tmp_mik+:566M:108M:13.5 yxan:tmp_mik+:519M:103M:13.5 yxan:tmp_mik+:536M:105M:13.5
yxan:tmp_mik+:519M:104M:13.5 yxan:tmp_mik+:528M:103M:13.5 yxan:tmp_mik+:519M:103M:13.5 yxan:tmp_mik+:524M:104M:13.5 yxan:tmp_mik+:524M:104M:13.5
yxan:tmp_mik+:528M:104M:13.5 yxan:tmp_mik+:516M:101M:13.5 yxan:tmp_mik+:515M:101M:13.5 yxan:tmp_mik+:515M:104M:13.5 yxan:tmp_mik+:520M:101M:13.5
yxan:tmp_mik+:524M:103M:13.5 yxan:tmp_mik+:520M:101M:13.5 yxan:tmp_mik+:515M:103M:13.5 yxan:tmp_mik+:516M:102M:13.5 yxan:tmp_mik+:587M:110M:13.5
yxan:tmp_mik+:558M:108M:13.5 yxan:tmp_mik+:524M:102M:13.5 yxan:tmp_mik+:537M:103M:13.5 yxan:tmp_mik+:572M:109M:13.5 yxan:tmp_mik+:549M:104M:13.5
yxan:tmp_mik+:519M:103M:13.5 yxan:tmp_mik+:528M:104M:13.5 yxan:tmp_mik+:520M:104M:13.5 yxan:tmp_mik+:515M:103M:13.5 yxan:tmp_mik+:515M:103M:13.5
yxan:tmp_mik+:520M:105M:13.5 yxan:tmp_mik+:528M:105M:13.5 yxan:tmp_mik+:515M:103M:13.5 yxan:tmp_mik+:515M:104M:13.5 yxan:tmp_mik+:515M:104M:13.5
yxan:slurm_s+:12M:3M yxan:srun:324M:8M yxan:srun:53M:1M
PBS_job=38581 user=yxan allocation=hpc_lipidhpre queue=checkpoint total_load=64.12 cpu_hours=866.08 wall_hours=13.21 unused_nodes=0 total_nodes=1 pp
n=64 avg_load=64.12 avg_cpu=6033% avg_mem=6852mb avg_vmem=36176mb top_proc=yxan:tmp_mik+:mike145:524M:104M:13.5hr:100% toppm=yxan:tmp_mikeCpu:mik
e145:730M:125M node_processes=68
```

What to look at ...

Normal behavior ...

You should be concerned if ...

```
(base) [jasonli3@mike4 ~]$ qshow 38581
PBS job: 38581, nodes: 1
Hostname Days Load CPU U# (User:Process:VirtualMemory:Memory:Hours)
mike145    278 64.12 6033 68 yxan:tmp_mik+:524M:104M:13.5 yxan:tmp_mik+:524M:104M:13.5 yxan:tmp_mik+:533M:107M:13.5 yxan:tmp_mik+:748M:128M:13.5
yxan:tmp_mik+:738M:124M:13.5 yxan:tmp_mik+:520M:104M:13.5 yxan:tmp_mik+:587M:109M:13.5 yxan:tmp_mik+:743M:128M:13.5 yxan:tmp_mik+:696M:118M:13.5
yxan:tmp_mik+:528M:101M:13.5 yxan:tmp_mik+:578M:108M:13.5 yxan:tmp_mik+:528M:105M:13.5 yxan:tmp_mik+:528M:106M:13.5 yxan:tmp_mik+:520M:105M:13.5
yxan:tmp_mik+:561M:106M:13.5 yxan:tmp_mik+:583M:109M:13.5 yxan:tmp_mik+:520M:103M:13.5 yxan:tmp_mik+:524M:103M:13.5 yxan:tmp_mik+:738M:125M:13.5
yxan:tmp_mik+:709M:119M:13.5 yxan:tmp_mik+:524M:103M:13.5 yxan:tmp_mik+:574M:107M:13.5 yxan:tmp_mik+:697M:121M:13.5 yxan:tmp_mik+:658M:115M:13.5
yxan:tmp_mik+:528M:102M:13.5 yxan:tmp_mik+:557M:108M:13.5 yxan:tmp_mik+:524M:105M:13.5 yxan:tmp_mik+:524M:105M:13.5 yxan:tmp_mik+:515M:102M:13.5
yxan:tmp_mik+:520M:104M:13.5 yxan:tmp_mik+:567M:108M:13.5 yxan:tmp_mik+:566M:108M:13.5 yxan:tmp_mik+:519M:103M:13.5 yxan:tmp_mik+:536M:105M:13.5
yxan:tmp_mik+:519M:104M:13.5 yxan:tmp_mik+:528M:103M:13.5 yxan:tmp_mik+:519M:103M:13.5 yxan:tmp_mik+:524M:104M:13.5 yxan:tmp_mik+:524M:104M:13.5
yxan:tmp_mik+:528M:104M:13.5 yxan:tmp_mik+:516M:101M:13.5 yxan:tmp_mik+:515M:101M:13.5 yxan:tmp_mik+:515M:104M:13.5 yxan:tmp_mik+:520M:101M:13.5
yxan:tmp_mik+:524M:103M:13.5 yxan:tmp_mik+:520M:101M:13.5 yxan:tmp_mik+:515M:103M:13.5 yxan:tmp_mik+:516M:102M:13.5 yxan:tmp_mik+:587M:110M:13.5
yxan:tmp_mik+:558M:108M:13.5 yxan:tmp_mik+:524M:102M:13.5 yxan:tmp_mik+:537M:103M:13.5 yxan:tmp_mik+:572M:109M:13.5 yxan:tmp_mik+:549M:104M:13.5
yxan:tmp_mik+:519M:103M:13.5 yxan:tmp_mik+:528M:104M:13.5 yxan:tmp_mik+:520M:104M:13.5 yxan:tmp_mik+:515M:103M:13.5 yxan:tmp_mik+:515M:103M:13.5
yxan:tmp_mik+:520M:105M:13.5 yxan:tmp_mik+:528M:105M:13.5 yxan:tmp_mik+:515M:103M:13.5 yxan:tmp_mik+:515M:104M:13.5 yxan:tmp_mik+:515M:104M:13.5
yxan:slurm_s+:12M:3M yxan:srun:324M:8M yxan:srun:53M:1M
PBS job: 38581 user=yxan allocation=hpc_lipidhpre queue=checkpoint total_load=64.12 cpu_hours=866.08 wall_hours=13.21 unused_nodes=0 total_nodes=1 pp
n=64 avg_load=64.12 avg_cpu=6033% avg_mem=6852mb avg_vmem=36176mb top_proc=yxan:tmp_mik+:mike145:524M:104M:13.5hr:100% toppm=yxan:tmp_mikeCpu:mik
e145:738M:125M_node_processes=68
```

What to look at ...	Normal behavior ...	You should be concerned if ...
avg_load	Close to allocated number of cores on the node	Consistently too low or too high


```
(base) [jasonli3@mike4 ~]$ qshow 38581
PBS job: 38581, nodes: 1
Hostname Days Load CPU U# (User:Process:VirtualMemory:Memory:Hours)
mike145    278 64.12 6033 68 yxan:tmp_mik+:524M:104M:13.5 yxan:tmp_mik+:524M:104M:13.5 yxan:tmp_mik+:533M:107M:13.5 yxan:tmp_mik+:748M:128M:13.5
yxan:tmp_mik+:738M:124M:13.5 yxan:tmp_mik+:520M:104M:13.5 yxan:tmp_mik+:587M:109M:13.5 yxan:tmp_mik+:743M:128M:13.5 yxan:tmp_mik+:696M:118M:13.5
yxan:tmp_mik+:528M:101M:13.5 yxan:tmp_mik+:578M:108M:13.5 yxan:tmp_mik+:528M:105M:13.5 yxan:tmp_mik+:528M:106M:13.5 yxan:tmp_mik+:520M:105M:13.5
yxan:tmp_mik+:561M:106M:13.5 yxan:tmp_mik+:583M:109M:13.5 yxan:tmp_mik+:520M:103M:13.5 yxan:tmp_mik+:524M:103M:13.5 yxan:tmp_mik+:738M:125M:13.5
yxan:tmp_mik+:709M:119M:13.5 yxan:tmp_mik+:524M:103M:13.5 yxan:tmp_mik+:574M:107M:13.5 yxan:tmp_mik+:697M:121M:13.5 yxan:tmp_mik+:658M:115M:13.5
yxan:tmp_mik+:528M:102M:13.5 yxan:tmp_mik+:557M:108M:13.5 yxan:tmp_mik+:524M:105M:13.5 yxan:tmp_mik+:524M:105M:13.5 yxan:tmp_mik+:515M:102M:13.5
yxan:tmp_mik+:520M:104M:13.5 yxan:tmp_mik+:567M:108M:13.5 yxan:tmp_mik+:566M:108M:13.5 yxan:tmp_mik+:519M:103M:13.5 yxan:tmp_mik+:536M:105M:13.5
yxan:tmp_mik+:519M:104M:13.5 yxan:tmp_mik+:528M:103M:13.5 yxan:tmp_mik+:519M:103M:13.5 yxan:tmp_mik+:524M:104M:13.5 yxan:tmp_mik+:524M:104M:13.5
yxan:tmp_mik+:528M:104M:13.5 yxan:tmp_mik+:516M:101M:13.5 yxan:tmp_mik+:515M:101M:13.5 yxan:tmp_mik+:515M:104M:13.5 yxan:tmp_mik+:520M:101M:13.5
yxan:tmp_mik+:524M:103M:13.5 yxan:tmp_mik+:520M:101M:13.5 yxan:tmp_mik+:515M:103M:13.5 yxan:tmp_mik+:516M:102M:13.5 yxan:tmp_mik+:587M:110M:13.5
yxan:tmp_mik+:558M:108M:13.5 yxan:tmp_mik+:524M:102M:13.5 yxan:tmp_mik+:537M:103M:13.5 yxan:tmp_mik+:572M:109M:13.5 yxan:tmp_mik+:549M:104M:13.5
yxan:tmp_mik+:519M:103M:13.5 yxan:tmp_mik+:528M:104M:13.5 yxan:tmp_mik+:520M:104M:13.5 yxan:tmp_mik+:515M:103M:13.5 yxan:tmp_mik+:515M:103M:13.5
yxan:tmp_mik+:520M:105M:13.5 yxan:tmp_mik+:528M:105M:13.5 yxan:tmp_mik+:515M:103M:13.5 yxan:tmp_mik+:515M:104M:13.5 yxan:tmp_mik+:515M:104M:13.5
yxan:slurm_s+:12M:3M yxan:srun:324M:8M yxan:srun:53M:1M
PBS_job=38581 user=yxan allocation=64.12 top_cpus=68 queue=checkpoint total_load=64.12 cpu_hours=866.08 wall_hours=13.21 unused_nodes=0 total_nodes=1 pp
n=64 avg_load=64.12 avg_cpu=603% avg_mem=6852mb avg_vmem=36176mb top_proc=yxan:tmp_mik+:mike145:524M:104M:13.5hr:100% toppm=yxan:tmp_mikeCpu:mik
e145:730M:125M node_processes=68
```

What to look at ...	Normal behavior ...	You should be concerned if ...
avg_load	Close to allocated number of cores on the node	Consistently too low or too high
ave_mem	Does not exceed total allocated memory	Exceeds total allocated memory

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

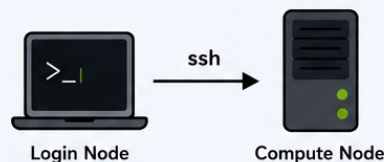
- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

scontrol – Show Detailed Information About Jobs, Partitions, and More

What it does

The **scontrol** command provides detailed information about the Slurm workload manager, including:

- Jobs (status, resources, time, nodes, etc.)
- Nodes (state, CPUs, memory, features, etc.)
- Partitions, reservations, accounting, and more



Why use it?

- ✓ Provides in-depth information about Slurm jobs, nodes, partitions, and the system.
- ✓ Useful for troubleshooting, debugging jobs, and checking resource allocation.
- ✓ Helps understand the state and configuration of the cluster.



Usage:

`scontrol <option>`

Examples:

`scontrol show job 916643`

`scontrol show node qbd163`

`scontrol show partition`

```
[user@login ~]$ scontrol show job 916643
```

```
JobId=916643 JobName=vasp_std UserId=user(34042) GroupId=users(100)
State=RUNNING Partition=workq AllocNode:Sid=qbd163:248848
RunTime=58:56:34 TimeLimit=72:00:00
SubmitTime=2025-07-12T10:15:20 EligibleTime=2025-07-12T10:15:20
StartTime=2025-07-12T10:15:22 EndTime=2025-07-15T10:15:22 Deadline=N/A
Nodes=qbd163 NumNodes=1 CPUs/Node=64 NumCPUs=64
ReqMem=128G AllocMem=128G NodeList=qbd163
Reason=None
Command=/work/user/vasp/run.sh
WorkDir=/work/user/vasp
StdErr=/work/user/vasp/slurm=916643.err
StdOut=/work/user/vasp/slurm=916643.out
```

Job identity

Job ID, name, owner, and group.

Job state and partition

Current state and partition where the job is running.

Time information

Runtime, time limit, start, end, and submission times.

Resource allocation

Nodes, CPUs, and memory allocated to the job.

Job reason

Reason for current job state (if not running).

Paths and I/O files

Command, working directory, and standard output/error files.

Other useful examples

`scontrol show job <job_id>`
Show detailed information about a specific job.

`scontrol show node <node>`
Show detailed information about a compute node.

`scontrol show partition`
List all partitions and their properties.

`scontrol show config`
Show Slurm configuration parameters.

`scontrol show reservation`
List all reservations.

Common Job Fields (`scontrol show job`)

JobId	Unique job identifier.
JobName	Name of the job.
UserId	User who submitted the job.
State	Current job state (e.g., RUNNING, PENDING).
Partition	Partition where the job is running.
RunTime	Time the job has been running.
TimeLimit	Maximum allowed time for the job.

Nodes / NodeList	Nodes allocated to the job.
CPUs/Node / NumCPUs	CPUs per node and total CPUs allocated.
ReqMem / AllocMem	Requested and allocated memory.
Reason	Reason for the job state (if not running).
WorkDir	Job's working directory.
StdOut / StdErr	Files for standard output and error.



Good to know

- Use **scontrol** for detailed, low-level information.
- Very helpful when debugging jobs or understanding resource usage.
- Combine with other commands like **squeue**, **sacct**, and **seff** for complete monitoring.



Tip: Use `scontrol show job <job_id>` to get full details about a job. Replace `<job_id>` with your actual job ID.

Press **q** to quit the output.

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

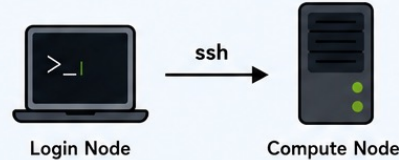
- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

sstat – Monitor Real-time Job and Step Resource Usage

What it does

The **sstat** command displays real-time resource usage for running jobs and steps in Slurm, including:

- CPU usage, memory usage, I/O, and more.
- Updated at regular intervals.
- Helpful to monitor performance and diagnose issues.



Why use it?

- ✓ Shows real-time resource usage for running jobs and steps.
- ✓ Helps identify CPU, memory, or I/O bottlenecks.
- ✓ Useful for performance monitoring and job optimization.
- ✓ Can be run repeatedly to watch usage over time.



Usage:

`sstat [options]`

Examples:

`sstat`

`sstat -j <job_id>`

`sstat -j <job_id>.batch`

`sstat -j <job_id>.extern`

```

[user@loginin ~]$ sstat -j 916643.batch
Job ID: 916643.batch
Cluster: qb4
User: user
Start Time: 07/12/2025 10:15:20
Elapsed Time: 00:42:16
    
```

JobID	StepID	Partition	AveCPU	MaxRSS	AveRSS	AveVMSize	AveDiskRead	AveDiskWrite	AvePagesIn	AvePagesOut	AveSwap	AveTmpDist
916643	batch	workq	2387.45	84.7G	72.3G	210.2G	12.5M	45.2M	1.2M	0.8M	0	1.1M
916643	extern	workq	15.32	1.3G	1.1G	2.4G	0	0	2.1K	1.8K	0	0
916643	0	workq	2387.45	84.7G	72.3G	210.2G	12.5M	45.2M	1.2M	0.8M	0	1.1M

Summary for JobID 916643:

CPU Efficiency: 89.4% Memory Efficiency: 78.6% Disk Read: 12.5M Disk Write: 45.2M

Job information

Basic details about the job and the reporting period.

Column headers

Each column shows a different resource metric.

Step / process usage

Real-time resource usage for each job step.

Summary

Overall efficiency and I/O summary for the job.

Common Column / Field Explanation

JobID	Job or step identifier.
StepID	Step name (batch, extern, or step ID).
Partition	Partition where the job is running.
AveCPU	Average CPU usage (in %).
MaxRSS	Maximum resident set size (physical memory).
AveRSS	Average resident set size.

AveVMSize	Average virtual memory size.
AveDiskRead	Average disk read rate.
AveDiskWrite	Average disk write rate.
AvePagesIn	Average pages paged in.
AvePagesOut	Average pages paged out.
AveSwap	Average swap space used.
AveTmpDisk	Average temporary disk space used.



Good to know

- Run **sstat** repeatedly to watch resource usage change over time:


```
watch -n 5 sstat -j <job_id>.batch
```
- High AveCPU close to 100% means all CPUs are busy.
- High MaxRSS or AveRSS indicates memory-intensive processes.
- Use with other tools (top, free, iostat, nvidia-smi) for deeper analysis.



Tip: Use `sstat -j <job_id>.batch` to monitor the main job step. Press **Ctrl+C** to stop watching.

Press **q** to quit the output.

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

107

```
(base) [jasonli3@qbc193 ~]$ nvidia-smi
Wed Feb  1 02:38:32 2023
```

NVIDIA-SMI 510.47.03 Driver Version: 510.47.03 CUDA Version: 11.6							
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr. ECC	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	MIG M.
0	Tesla V100-PCIE...	On	00000000:3B:00.0	Off		Off	
N/A	36C	P0	54W / 250W	4155MiB / 32768MiB	72%	Default	N/A
1	Tesla V100-PCIE...	On	00000000:AF:00.0	Off		Off	
N/A	36C	P0	52W / 250W	4155MiB / 32768MiB	78%	Default	N/A

Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory	
	ID	ID				Usage	
0	N/A	N/A	259491	C	... che/TeraChem/bin/terachem	4147MiB	
1	N/A	N/A	259491	C	... che/TeraChem/bin/terachem	4147MiB	

What to look at ...	Normal behavior ...	You should be concerned if ...
---------------------	---------------------	--------------------------------

```
(base) [jasonli3@qbc193 ~]$ nvidia-smi
Wed Feb  1 02:38:32 2023
```

NVIDIA-SMI 510.47.03 Driver Version: 510.47.03 CUDA Version: 11.6							
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Incorr. ECC	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	MIG M.
0	Tesla V100-PCIE...	On	00000000:3B:00.0	Off	72%	Off	
N/A	36C	P0	54W / 250W	4155MiB / 32768MiB		Default	N/A
1	Tesla V100-PCIE...	On	00000000:AF:00.0	Off	78%	Off	
N/A	36C	P0	52W / 250W	4155MiB / 32768MiB		Default	N/A

Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory	
	ID	ID				Usage	
0	N/A	N/A	259491	C	... che/TeraChem/bin/terachem	4147MiB	
1	N/A	N/A	259491	C	... che/TeraChem/bin/terachem	4147MiB	

What to look at ...	Normal behavior ...	You should be concerned if ...
GPU usage	Close to 100%	Consistently too low

```
(base) [jasonli3@qbc193 ~]$ nvidia-smi
Wed Feb  1 02:38:32 2023

+-----+
| NVIDIA-SMI 510.47.03      Driver Version: 510.47.03      CUDA Version: 11.6     |
+-----+-----+
| GPU  Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
|  0   Tesla V100-PCIE...    0n     | 00000000:3B:00.0 Off  |  4155MiB / 32768MiB |    72%      Default  |
| N/A   36C    P0      54W / 250W    |                   |           N/A         |
+-----+-----+
|  1   Tesla V100-PCIE...    0n     | 00000000:AF:00.0 Off  |  4155MiB / 32768MiB |    78%      Default  |
| N/A   36C    P0      52W / 250W    |                   |           N/A         |
+-----+-----+

Processes:
+-----+
| GPU  GI   CI       PID   Type   Process name                      GPU Memory |
|      ID   ID                     |              Usage |
+-----+-----+
|  0   N/A  N/A     259491   C     ... che/TeraChem/bin/terachem     4147MiB |
|  1   N/A  N/A     259491   C     ... che/TeraChem/bin/terachem     4147MiB |
+-----+-----+
```

What to look at ...	Normal behavior ...	You should be concerned if ...
GPU usage	Close to 100%	Consistently too low
Memory usage (not virtual memory)	Not used up	Used up

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

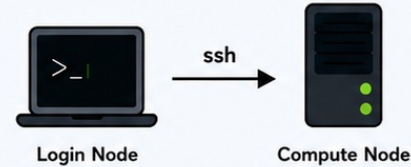
8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

scancel – Cancel (Terminate) Jobs or Job Steps in Slurm

What it does

The `scancel` command is used to cancel (terminate) running, pending, or held jobs or job steps in Slurm. It allows users to stop jobs they no longer need, free resources, or terminate problematic jobs.



Why use it?

- ✓ Cancel jobs that are no longer needed.
- ✓ Free up resources for yourself or other users.
- ✓ Stop jobs that are stuck, running too long, or using too many resources.
- ✓ Cancel specific job steps within a job.



Usage: `scancel [options] <job_id>`

Examples: `scancel 916643` | `scancel -u user` | `scancel 916643.1`

```
[user@login ~]$ scancel 916643
Canceled job 916643
```

Cancel a specific job

`scancel 916643` cancels job with ID 916643. You will see a confirmation message.

```
[user@login ~]$ squeue -u user
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
916642	workq	vasp_std	user	R	02:10:15	1	qbd162
916644	workq	qe_relax	user	PD	00:00	1	(Priority)

Verify the job is canceled

Use `squeue` to check that the job is no longer in the queue.

Cancel job steps

To cancel a specific job step:
`scancel <job_id>.<step_id>`
 Example: `scancel 916643.1`

Common Options

`-u, --user <user>` Cancel jobs for a specific user.
`-n, --nodelist <node>` Cancel jobs running on a specific node.
`-p, --partition <part>` Cancel jobs in a specific partition.
`-t, --state <state>` Cancel jobs in given state (PD,R,CG, etc.).
`-f, --force` Force cancel (use with caution).
`-h, --help` Show help message.

Common States (for -t option)

PD Pending
R Running
CG Completing
CD Completed
F Failed
CA Cancelled



Good to know

- You can cancel your own jobs or, if permitted, jobs of other users.
- Canceling a job frees allocated resources immediately.
- Be careful with `scancel -f`; it kills jobs without asking.
- Use `scancel` for whole jobs or `scancel <job_id>.<step_id>` to cancel a specific step.

Examples

`scancel 916643` # Cancel job 916643
`scancel -u user` # Cancel all jobs of user
`scancel -n qbd163` # Cancel jobs on node qbd163
`scancel -p workq -t PD` # Cancel pending jobs in workq
`scancel 916643.1` # Cancel step 1 of job 916643



Tip: Always verify before canceling: `squeue -j <job_id>`

Press `q` to quit the output.

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login In to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

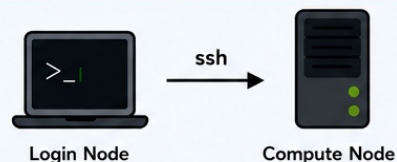
- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

sacct – Report Accounting Information for Jobs and Job Steps

What it does

The `sacct` command reports accounting information about jobs and job steps that have finished (completed, failed, cancelled, or timed out).

It provides details such as runtime, resources used, CPU efficiency, memory usage, exit status, and more.



Why use it?

- ✓ Check the resource usage of completed jobs.
- ✓ Understand job performance and efficiency.
- ✓ Review job history and accounting data.
- ✓ Useful for billing/allocation and troubleshooting.



Usage:

`sacct [options]`

Examples:

`sacct -j 916643`

`sacct -u user`

`sacct --starttime 2025-07-11`

`sacct -X -j 916643`

```
[user@login ~]$ sacct -j 916643
```

JobID	JobName	User	Account	Partition	State	ExitCode	AllocCPUS (hh:mm:ss)	Elapsed (hh:mm:ss)	TotalCPU (hh:mm:ss)	ReqMem (Max)	MaxRSS (Max)	AveCPU (%)	AveDiskRead (MB/sec)	AveDiskWrite (MB/sec)
916643	vasp_std	user	loni_mat_bio24	workq	COMPLETED	0:0	64	58:56:34	3703:21:12	128G	82.6G	89.4	12.5	45.2
-916643.batch	batch	user	loni_mat_bio24	workq	COMPLETED	0:0	1	58:56:34	58:56:34	128G	0.1G	1.7	0.0	0.0
-916643.0	vasp_std	user	loni_mat_bio24	workq	COMPLETED	0:0	64	58:56:34	3703:21:12	128G	82.6G	89.4	12.5	45.2
916640	relax	user	loni_mat_bio24	workq	FAILED	1:0	32	02:13:07	41:22:18	64G	63.1G	52.3	8.1	21.4
-916640.batch	batch	user	loni_mat_bio24	workq	FAILED	1:0	1	02:13:07	02:13:07	64G	0.1G	1.3	0.0	0.0
-916640.0	relax	user	loni_mat_bio24	workq	FAILED	1:0	32	02:13:07	41:22:18	64G	63.1G	52.3	8.1	21.4

Job information

Shows the job ID, name, user, account, partition, state, exit code, and allocated CPUs.

Time and CPU usage

Elapsed: Wall clock time
TotalCPU: Total CPU time used (across all CPUs)

Resource usage

ReqMem (Max): Memory requested
MaxRSS (Max): Maximum resident memory used

Efficiency and I/O

AveCPU (%): Average CPU efficiency
AveDiskRead: Average disk read rate
AveDiskWrite: Average disk write rate

Common Options

`-j, --jobs <jobid>` Report for a specific job ID.
`-u, --user <user>` Report for a specific user.
`--starttime <time>` Start time (e.g., 2025-07-11).
`--endtime <time>` End time.
`-X` Skip steps and show only job records.
`--format <list>` Specify output columns.
`--long` Long (detailed) output format.

Common States

COMPLETED Job finished successfully.
FAILED Job failed.
CANCELLED Job was cancelled.
TIMEOUT Job exceeded time limit.
NODE_FAIL Node failure.
OUT_OF_MEMORY Job was killed due to OOM.



Good to know

- `sacct` shows information only for completed jobs/steps. It does not show running jobs (use `squeue` for that).
- CPU efficiency (AveCPU) close to 100% is ideal.
- High MaxRSS near ReqMem means memory was fully used.
- Use `-X` to see only job-level summary without steps.

Examples

`sacct -j 916643` # Report for job 916643
`sacct -u user` # Report for user
`sacct --starttime 2025-07-11` # Jobs since July 11, 2025
`sacct -X -j 916643` # Job summary only (no steps)
`sacct --format=JobID,State,Elapsed,MaxRSS -j 916643` # Custom output format



Tip: Use `sacct` after your job has finished to analyze performance and resource usage.

Press `q` to quit the output.

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

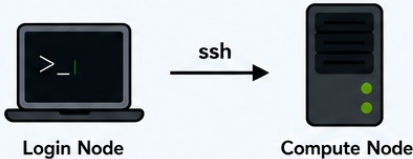
- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

seff - Slurm Efficiency Report for Jobs

What it does

The `seff` command provides a quick summary of efficiency and resource usage for a completed job in Slurm.

It calculates CPU efficiency, memory efficiency, and other key metrics to evaluate how well resources were utilized.



Why use it?

- ✓ Quickly assess how efficiently your job used allocated resources.
- ✓ Identify underutilized CPU or memory.
- ✓ Helpful for optimizing job requests and performance.
- ✓ Great for reporting and comparing job efficiency.



Usage:

`seff <job_id>`

Examples:

`seff 916643`

`seff -j 916643`

`seff --jobs 916643`

```
[user@login ~]$ seff 916643
Job ID: 916643
Cluster: qb4
User/Group: user/users(34042)
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 64

CPU Utilized: 3-11:23:45
CPU Efficiency: 89.41% of 3-00:00:00 core-walltime
Job Wall-clock time: 58:56:34
Memory Utilized: 82.60 GB
Memory Efficiency: 68.83% of 120.00 GB
Swap Utilized: 0.00 MB
```

Job information

Basic details about the job.

CPU usage

Total CPU time used by all cores.
CPU Efficiency = (CPU Utilized) / (Allocated CPU time)
Shows how well the CPUs were used.

Time information

Total elapsed (wall-clock) time of the job.

Memory usage

Highest memory used.
Memory Efficiency = (Memory Utilized) / (Requested Memory)
Shows how efficiently memory was used.

Swap usage

Amount of swap space used (if any).

Key Metrics Explained

CPU Efficiency

Percentage of allocated CPU time that was effectively used.

Memory Efficiency

Percentage of requested memory that was actually used.

CPU Utilized

Sum of CPU time used across all cores (reported in D-HH:MM:SS).

Job Wall-clock time

Actual elapsed time from start to finish.

Memory Utilized

Maximum resident memory used by the job.

Common Options

- `-j, --jobs <jobid>` Show efficiency for a specific job.
- `--format <list>` Specify output format (default is standard).
- `--allnodes` Show per-node efficiency (if multi-node job).
- `-h, --help` Show help message.

Efficiency Guidelines

CPU Efficiency	Memory Efficiency	Interpretation
> 80%	> 80%	Excellent – resources well utilized.
50% – 80%	50% – 80%	Good – some room for optimization.
< 50%	< 50%	Low – consider reducing requested resources.



Good to know

- `seff` works only for completed jobs.
- High CPU efficiency close to 100% is ideal.
- Low memory efficiency may mean you requested more memory than needed.
- Use `seff` regularly to tune your resource requests.

Example

```
[user@login ~]$ seff 916643
...
CPU Efficiency: 89.41% of 3-00:00:00 core-walltime
Memory Efficiency: 68.83% of 120.00 GB
Job Wall-clock time: 58:56:34
...
```



Tip: Use `seff` after your job has finished to evaluate efficiency and improve future job performance.

Press `q` to quit the output.

Common issues	What would happen
Exceeded memory allocation (e.g., using more memory than allocated w/ single queue)	Terminated. Receive email notice.
Exceeded ppn/core allocation (e.g., using more cores than allocated w/ single queue)	Terminated. Receive email notice.
Seriously underutilize node CPU cores / unused nodes (e.g., Requested multiple nodes but only runs on one node)	Receive email warning. (* Killed if completely idle for a long time)
Submitting to bigmem but only using little memory	Receive email warning.
Running intensive calculation on head nodes	Terminated. Receive email notice.
Submitting too many (i.e., hundreds of) single-thread jobs	Poor parallelization and bad for server. We may reach out to you to help. (Better yet, reach out to us first)

1. Basic Concepts

- a) What is the cluster?
- b) Login vs. compute node
- c) Account and allocation

2. Jobs and the Job Scheduler

- a) What is a job?
- b) Why a scheduler is needed
- c) Slurm job scheduler

3. Requesting Resources

- a) Resource allocation
- b) Partitions
- c) Queues

4. Before Submitting a Job

- a) Checking available resources
 - **sinfo**
- b) Checking the queue
 - **squeue**

5. Login to the Cluster

6. Running Jobs on the Clusters

- a) Interactive jobs
 - **salloc**
- b) Batch jobs
 - **sbatch**

7. Monitoring Jobs

- a) Node-level monitoring
 - **top**
 - **free**
- b) Job-level monitoring
 - **qshow**
 - **scontrol**
 - **sstat**
 - **nvidia-smi**
 - **scancel**

8. After the Jobs Complete

- a) Job health analysis
 - **sacct**
 - **seff**
- b) Job history
 - **sacct**

sacct – View All Your Jobs (All History)



Purpose:

Display all jobs you have ever run in the Slurm accounting database with resource usage in Gigabytes.



Login Node



Cluster
(Slurm Controllers)

Syntax

```
sacct [options] [filters] [--starttime=YYYY-MM-DD] [--endtime=YYYY-MM-DD] [--units=G]
--format=fields
Retrieve accounting information from the Slurm database.
```

Command

```
sacct -X -u user --starttime=YYYY-MM-DD --units=G
```

-X

Show only top-level jobs (exclude .batch, .extern, .0 steps)

-u user

Your user account

--starttime=YYYY-MM-DD

Show all jobs ever recorded (YYYY = year, MM = month, DD = day)

--units=G

Display memory in Gigabytes

Example Output (truncated)

JobID	JobName	Partition	State	ExitCode	Elapsed	AllocCPUS	ReqMem (G)	MaxRSS (G)	NodeList
689123	vasp_relax	workq	COMPLETED	0:0	02:14:33	64	128.0	82.3	qbd101
689456	orca_freq	workq	COMPLETED	0:0	01:08:12	32	64.0	41.7	qbd115
689789	qe_scf	single	FAILED	1:0	00:12:48	16	32.0	28.5	qbd022
690102	md_gmx	workq	COMPLETED	0:0	10:32:55	128	256.0	173.6	qbd[050-051]
690555	lammps_run	workq	TIMEOUT	0:0	3-00:00:05	64	128.0	---	qbd273
691001	test_job	workq	CANCELLED	0:0	00:00:10	0	64.0	---	None assigned

★ Useful Tips

- Use this command to audit your complete job history.
- You can add `--helpformat` to see all available fields.
- Large history queries may take more time.

⚙️ Related Commands

```
sacct -u $USER --starttime=YYYY-MM-DD
# Jobs from YYYY-MM-DD to now
sacct -j <jobid> --units=G
# Details for a specific job
sacct -j <jobid> --helpformat
# See all available fields
```

Commonly Used --format Fields

• JobID	• Elapsed	• ReqMem	• NodeList
• JobName	• ElapsedRaw	• MaxRSS	• NNodes
• Account	• Start	• AveRSS	• AllocNodes
• Partition	• End	• MaxVMSize	• User
• State	• Submit	• AveVMSize	• WorkDir
• ExitCode	• AllocCPUS	• AveCPU	• QOS
• DerivedExitCode	• ReqCPUS	• CPUTime	• TRESAlloc

Example with Custom Format

```
sacct -X -u user --starttime=YYYY-MM-DD --units=G \
--format=JobID,JobName,State,Elapsed,AllocCPUS,ReqMem,MaxRSS,NodeList
```

JobID	JobName	State	Elapsed	AllocCPUS	ReqMem (G)	MaxRSS (G)	NodeList
689123	vasp_relax	COMPLETED	02:14:33	64	128.0	82.3	qbd101
689789	qe_scf	FAILED	00:12:48	16	32.0	28.5	qbd022
...							

See All Available Fields

To see all format options:

```
sacct --helpformat
```

This prints all available fields that can be used with `--format=`.



Note: Use `-X` to show only the job level (not individual steps like .batch, .extern, .0). Use `--units=G` to display memory in Gigabytes.

sreport – Slurm Reporting Commands



Purpose:

Generate summary reports from the Slurm accounting database.
These reports help you analyze resource usage by users, accounts, and job sizes.



Login Node

ssh



Cluster
(Slurm Controllers)

Syntax

```
sreport [options] job SizesByAccount [parameters]
```

Run the job report "SizesByAccount" with optional time unit and additional parameters.

Common sreport Commands

#	Command	Description	Notes
1	<code>sreport user TopUsage</code>	Show top users by resource usage (CPU time by default).	Useful to identify the most resource-consuming users.
2	<code>sreport job SizesByAccount</code>	Show CPU time usage (by default in Minutes) by account, grouped by job size (number of CPUs).	No date range → reports all available accounting data.
3	<code>sreport job SizesByAccount Start=YYYY-MM-DD End=YYYY-MM-DD</code>	Show CPU time usage by account for jobs run in the specified date range.	Time format: YYYY-MM-DD (e.g., <code>Start=2026-06-01</code>)
4	<code>sreport job SizesByAccount Start=YYYY-MM-DD End=YYYY-MM-DD PrintJobCount Format=Cluster,Account%30</code>	Show NUMBER OF JOBS (not time) by account and job size in the given date range.	<code>PrintJobCount</code> replaces time values with job counts. <code>Format=Cluster,Account%30</code> shows full account names.

1. Top Users by Usage

```
sreport user TopUsage
```

User	Used (CPU Hours)	% of Total
user1	12,543.2	18.92%
user2	9,823.1	14.83%
user3	6,789.0	10.24%
user4	5,577.3	8.41%
user5	4,211.8	6.35%
...		

2. Resource Usage by Account (CPU Time)

```
sreport job SizesByAccount Start=YYYY-MM-DD End=YYYY-MM-DD
```

Cluster	Account	0-49 CPUs	50-249 CPUs	250-499 CPUs	500-999 CPUs	>= 1000 CPUs	Total (CPU Hours)	% of Cluster
clusterA	project_alpha	1,245.2	5,678.1	0.0	0.0	0.0	6,923.3	15.63%
clusterA	project_beta	2,345.6	56,789.4	0.0	0.0	0.0	59,135.0	46.18%
clusterA	research_gamma	1,234.0	42,345.7	0.0	0.0	0.0	43,579.7	38.19%
...								

★ Useful Tips

- CPU time is reported in CPU hours by default.
- Use `-t Minutes`, `-t Hours`, or `-t Percent` to change the time unit.
- Use `Format=Cluster,Account%30` to show full account names.
- Use `PrintJobCount` to see the number of jobs instead of time.
- Use `-P` for a parsable (pipe-delimited) output.

3. Number of Jobs by Account and Job Size

```
sreport job SizesByAccount Start=YYYY-MM-DD End=YYYY-MM-DD PrintJobCount Format=Cluster,Account%30
```

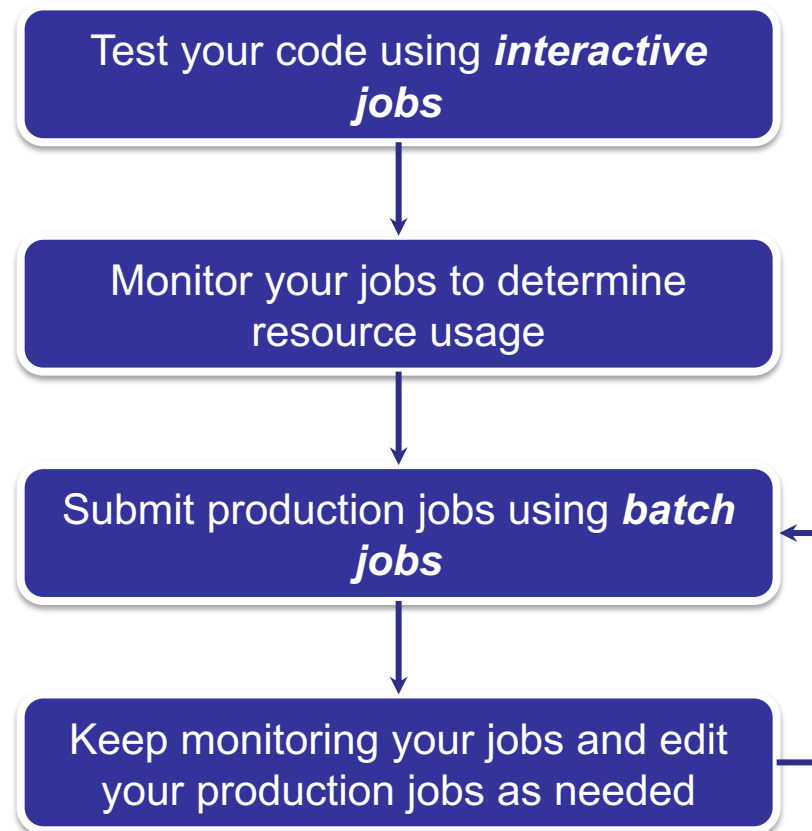
Cluster	Account	0-49 CPUs	50-249 CPUs	250-499 CPUs	500-999 CPUs	>= 1000 CPUs	Total Jobs	% of Cluster
clusterA	project_alpha	128	1,203	0	0	0	1,331	16.02%
clusterA	project_beta	256	2,845	0	0	0	3,101	45.66%
clusterA	research_gamma	187	2,114	0	0	0	2,301	38.32%
...								

⚙ Common Options

Option	Description
<code>Start=YYYY-MM-DD</code>	Start date (inclusive)
<code>End=YYYY-MM-DD</code>	End date (inclusive)
<code>-t Minutes Hours Percent</code>	Set time unit
<code>-P</code>	Parsable (pipe-delimited) output
<code>Format=<fields></code>	Select and format output columns
<code>PrintJobCount</code>	Show number of jobs instead of time

Note: Dates use the format: YYYY-MM-DD (e.g., 2026-06-01 for June 1, 2026).

If End date is in the future, the report will include data only up to the last available accounting period.



Thank you!



Job Information

- **scontrol show job <jobid>**
Detailed information about a job.
- **scontrol show node <nodename>**
Show node status and resources.
- **scontrol show partition**
List available partitions.
- **sinfo -N -l**
Show nodes and partitions.
- **sprio -u \$USER**
Show job priority (backfill priority).

```
$ scontrol show job 123456
JobId=123456 JobName=test
UserId=user(1001) GroupId=user(100)
Partition=workq State=RUNNING
Nodes=qbd101 NumCPUs=32
StartTime=2024-05-20T10:15:30
TimeLimit=02:00:00 TimeUsed=00:35:12
...
```

```
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE
single    up      168:00:00   20   idle
workq     up       72:00:00    60   mix
gpu       up       72:00:00    10   idle
```



Job Submission

- **sbatch job.sh**
Submit a batch job script.
- **sbatch -A <account> -p <part> \ -N 1 -n 32 job.sh**
Submit a batch job with options.
Examples: -A, -p, -N, -n, --time
- **srun ./a.out**
Run an interactive command.
- **salloc -N 1 -n 32 -t 02:00:00**
Create an interactive allocation.
Examples: -N, -n, -t, -p

```
$ sbatch job.sh
Submitted batch job 123456
```

```
$ salloc -N 1 -n 32 -t 02:00:00
salloc: Granted job allocation 123457
salloc: Nodes: qbd101
salloc: CPUs: 32
[user@qbd101 ~]$
```



Job Monitoring (Running)

- **squeue -u \$USER**
Show your running and pending jobs.

```
$ squeue -u $USER
JOBID PARTITION ST  TIME  NODES NODELIST(REASON)
123456 workq    R   00:35:12   1   qbd101
123457 workq    PD  00:00:00   1   (Priority)
```

- **watch -n 10 squeue -u \$USER**
Refresh job queue every 10 sec.

- **sstat -j <jobid>**
Real-time statistics for a running job.

```
$ sstat -j 123456 --format=JobID,MaxRSS,\
AveCPU,Elapsed
JobID  MaxRSS  AveCPU  Elapsed
123456  8.12G   95.3    00:35:12
```

- **seff <jobid>**
Job efficiency summary.

```
$ seff 123456
Job ID: 123456
CPU Efficiency: 92.1%
Memory Efficiency: 78.6%
```



Job Management

- **scancel <jobid>**
Cancel a specific job.
- **scancel -u \$USER**
Cancel all your jobs.
- **scancel -t <sec> <jobid>**
Cancel a job after <sec> seconds.
Example: scancel -t 60 123456
- **sprio -u \$USER**
Show job priority (backfill priority).

```
$ scancel 123456
$ scancel -u $USER
$ scancel -t 60 123456
```



Job History

- **sacct -u \$USER**
Show your job history.

```
$ sacct -u $USER
JobID  State  Elapsed  MaxRSS
123456 COMPLETED 01:15:32 12.4G
123455 FAILED    00:05:12 1.2G
123454 CANCELLED 00:02:01 0
```

- **sacct -j <jobid> --format=...**
Detailed information for a job.
Example: sacct -j 123456 -o JobID,State,Elapsed,MaxRSS

- **sacctmgr show assoc user=\$USER**
Show account/association usage.

- **sreport job account=\$USER**
Compute account usage report.

```
$ sreport job account=$USER
Cluster: cluster
User: user
Account: account1
Used CPUs: 123,456
Used CPUHours: 45,678.90
Used Jobs: 234
```

```
salloc -A loni_loniadmin1 -p single --nodelist=qbd042 -t 1-00:00:00
```

```
salloc -A loni_loniadmin1 -p single --nodelist=qbd042 -N1 -n64 -t 1-00:00:00
```

```
salloc -A loni_loniadmin1 -p workq --nodelist=qbd[046-047] -N2 -n128 -t 1-00:00:00
```


Batch Script Structure

A batch script has two main parts: header (parameters) and body (commands)

1. HEADER – Parameters (Directives)

Tells the scheduler what resources and settings are needed

```
#!/bin/bash
#SBATCH --account=allocation_name      # Allocation/Project account
#SBATCH --partition=partition_name     # Partition to use
#SBATCH --nodes=number_of_nodes       # Number of nodes
#SBATCH --ntasks=number_of_tasks      # Total number of tasks (MPI processes)
#SBATCH --cpus-per-task=number_of_CPU_cores_per_task # CPU cores per task
#SBATCH --time=DD-HH:MM:SS            # Time limit (DD-HH:MM:SS) format)
#SBATCH --job-name=job_name           # Name of the job
#SBATCH --output=standard.out         # Standard output file
#SBATCH --error=standard.err          # Standard error file
#SBATCH --mail-user=user@mail.address # Email for notifications
#SBATCH --mail-type=ALL                # Mail events (BEGIN, END, FAIL, ALL)
```



Header Specifies:

- Account (project allocation)
- Partition
- Number of nodes
- Number of tasks (MPI processes)
- CPU cores per task
- Time limit
- Job name
- Output file (standard output)
- Error file (standard error)
- Email address for notifications
- Email type (ALL, BEGIN, END, FAIL)

2. BODY – Commands

Commands to run on the allocated resources

```
# Load required modules
module load gcc/12.2.0
module load openmpi/4.1.5

# Set environment variables
export OMP_NUM_THREADS=number_of_CPU_cores_per_task # Match OpenMP threads to CPUs per task

# Run the application
srun ./my_program.x -in input.dat -out output.dat
```



Body Contains:

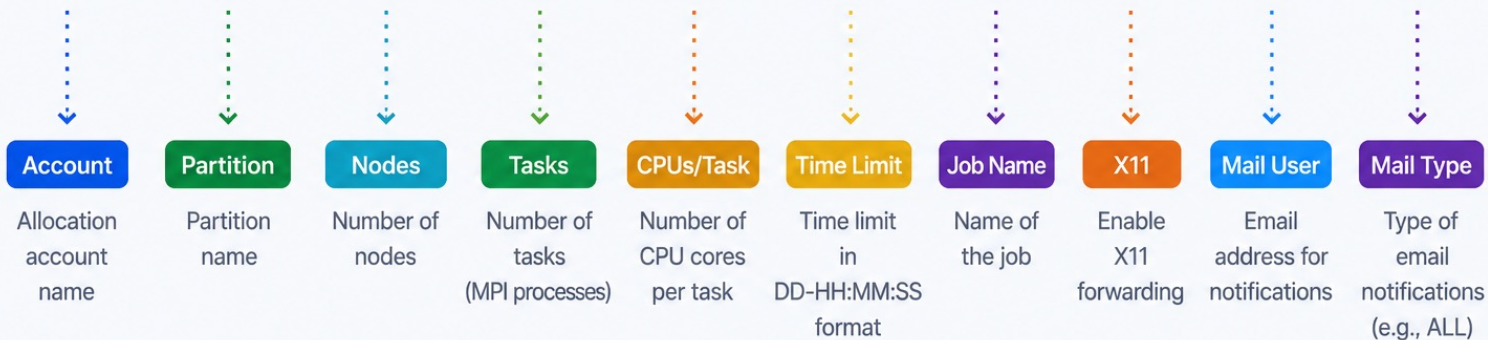
- Load modules and software
- Set environment variables
- Change directories
- Run your applications
- Any other commands needed

salloc

Allocates resources for an interactive job

```
●●● salloc --account=allocation_name --partition=partition_name --nodes=number_of_nodes
      --ntasks=number_of_tasks --cpus-per-task=number_of_CPU_cores_per_task --time=DD-HH:MM:SS
      --job-name=job_name --x11 --mail-user=user@mail.address --mail-type=ALL
```

```
salloc --account=allocation_name --partition=partition_name --nodes=number_of_nodes
      --ntasks=number_of_tasks --cpus-per-task=number_of_CPU_cores_per_task --time=DD-HH:MM:SS
      --job-name=job_name --x11 --mail-user=user@mail.address --mail-type=ALL
```

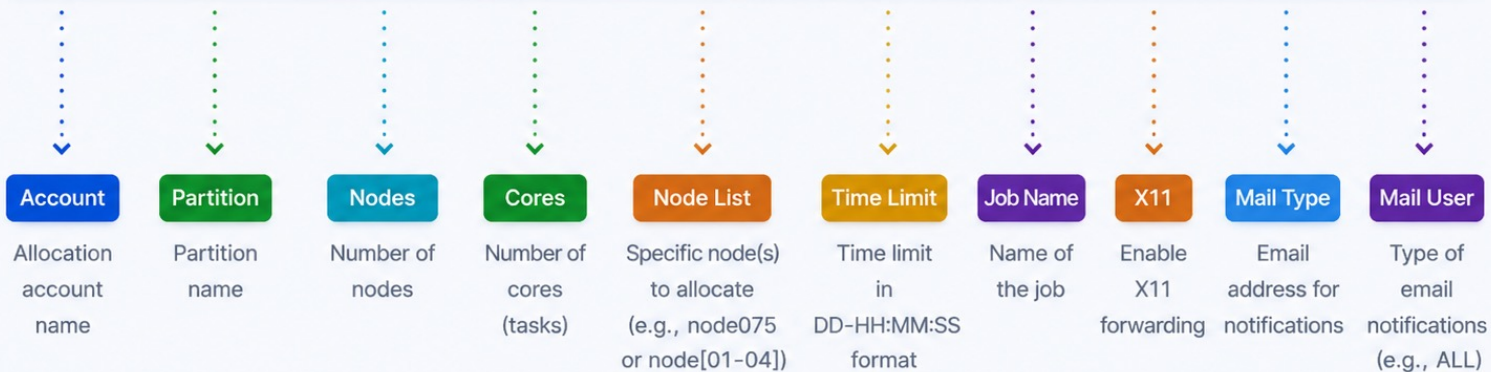


salloc

Allocates resources for an interactive job

```
●●● salloc [options] --nodelist=node_list --x11 --mail-type=ALL --mail-user=user@mail.address
```

```
salloc -A allocation_name -p partition_name -N number_of_nodes -n number_of_cores  
--nodelist=node_list -t DD-HH:MM:SS -J job_name --x11 --mail-type=ALL --mail-user=user@mail.address
```

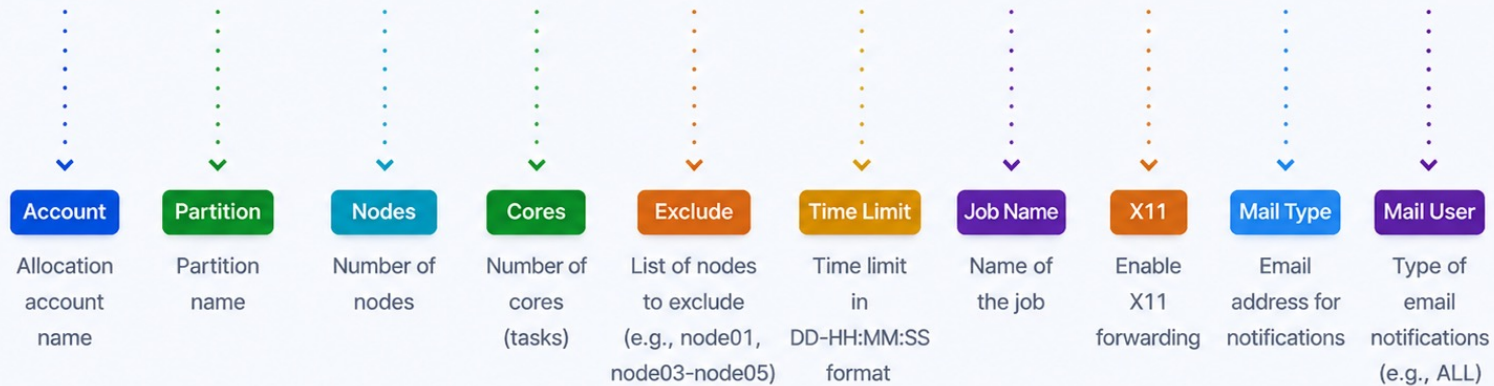


salloc

Allocates resources for an interactive job

```
●●● salloc [options] --exclude=list_of_nodes --x11 --mail-type=ALL --mail-user=user@mail.address
```

```
salloc -A allocation_name -p partition_name -N number_of_nodes -n number_of_cores  
--exclude=list_of_nodes -t DD-HH:MM:SS -J job_name --x11 --mail-type=ALL --mail-user=user@mail.address
```



Short Options

-A	account
-p	partition
-N	nodes
-n	tasks
-c	CPUs/task
-t	time
-J	job name

=

Long Options

--account	account
--partition	partition
--nodes	nodes
--ntasks	tasks
--cpus-per-task	CPUs/task
--time	time
--job-name	job name

=

=

=

=

=

=

Running LAMMPS jobs using 1 GPU

```
#!/bin/bash
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --nodes=1
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --time=D-HH:MM:SS
#SBATCH --account=allocation-name
#SBATCH --job-name=test
#SBATCH --output=lammps.%j.out
#SBATCH --error=lammps.%j.err
#SBATCH --mail-user=your@email.address
#SBATCH --mail-type=BEGIN,END

module purge
module load lammps/02Aug2023/intel-2021.5.0-cuda-11.6.0-intel-mpi-2021.5.1

echo $SLURM_JOBID
echo $SLURM_NTASKS
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

echo $SLURM_SUBMIT_DIR
cd $SLURM_SUBMIT_DIR

time srun -N1 -n16 lmp_mpi -sf gpu -pk gpu 1 neigh yes newton
off -in lj.in > lj.out
```

Running LAMMPS jobs using 2 GPUs

```
#!/bin/bash
#SBATCH -p gpu
#SBATCH --gres=gpu:2
#SBATCH -N 1
#SBATCH -n 32
#SBATCH -c 1
#SBATCH -t D-HH:MM:SS
#SBATCH -A hpc_allocation
#SBATCH -J test
#SBATCH -o lammps.%j.out
#SBATCH -e lammps.%j.err
#SBATCH --mail-user=your@email.address
#SBATCH --mail-type=BEGIN,END

module purge
module load lammps/02Aug2023/intel-2021.5.0-cuda-11.6.0-intel-mpi-2021.5.1

echo $SLURM_JOBID
echo $SLURM_NTASKS
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

echo $SLURM_SUBMIT_DIR
cd $SLURM_SUBMIT_DIR

time srun -N1 -n32 lmp_mpi -sf gpu -pk gpu 2 neigh yes newton
off -in lj.in > lj.out
```


SLURM_JOB_ID

- This ID can be used to track the job status and retrieve logs

SLURM_JOB_NAME

- The name of the job as specified by the user in command or script.

SLURM_NTASKS

- **The total number of tasks (processes) allocated for the job.**

SLURM_CPUS_PER_TASK

- **The number of CPUs allocated per task.**

SLURM_NODES

- **A list of nodes allocated for the job.**

SLURM_NODELIST

- A comma-separated list of nodes that are allocated for the job.

SLURM_SUBMIT_DIR

- **The directory from which the job was submitted.**

SLURM_ARRAY_JOB_ID

- The job ID of the array job if the job is part of a job array.

SLURM_ARRAY_TASK_ID

- The ID of the specific task in an array job.

SLURM_JOB_NODELIST

- List of nodes allocated for the job, specifically formatted for easier parsing.

SLURM_JOB_DEPENDENCY.

- Indicating which jobs must complete before this job can start.

SLURM_MEM_PER_CPU

- The amount of memory allocated per CPU core for the job.

command > output.log 2>&1

In Unix-like operating systems, processes have three standard file descriptors:

- 1. Standard Input (stdin): File descriptor 0 (used for input).
- 2. Standard Output (stdout): **File descriptor 1** (used for normal output).
- 3. Standard Error (stderr): **File descriptor 2** (used for error messages).