Job scripts and how to reserve resources in Prince
## Running jobs on the Prince Cluster

### Accessing the Prince Cluster
- From Windows workstation
- From Mac workstation

### Software and Environment Module

### Job script and resource request
- Introduction to job scheduling
- Submitting jobs with `sbatch`
- Requesting resources
- Using computing nodes interactively

### Monitoring batch jobs
- Monitoring batch jobs - `squeue`
**Introduction to Job Scheduling**

A CPU can only do one thing at a time. When you run two programs at a time on your single-CPU workstation, the operating system uses "timeslicing" to give the illusion of running both at once: it runs one of the programs for a moment, then pauses that program to run the other one for a moment, and so on. If you are reading and editing documents, as most workstations are used for, this works fine - most of the time the computer is waiting for you to hit the next key anyway.

Running a simulation, however, uses the CPU heavily. When timeslicing between two simulations on a single-CPU workstation, you can expect each simulation to take roughly twice as long as if it had dedicated use of the CPU. This is obviously counter to the goals of high performance computing, so HPC clusters use batch job scheduling to ensure that each job has dedicated access to the resources it needs.

The principle behind batch job scheduling is:

- Each simulation is prepared (by the user) as a job, ( batch job) which is a script that sets up and runs the simulation without interactive input from the user.
- Each job needs a certain set of resources for a certain amount of time. The user knows these needs in advance and can specify them in the job script. "Resources" includes a number of CPUs and an amount of memory. Some jobs might also need a specific type of CPU or a specific software license.
- The scheduler plans when and on which compute nodes to run each job.

The diagram below illustrates how jobs can be allocated to certain parts of the cluster at certain times. When a new job is submitted, the scheduler looks for a place and time to run the job, always aiming to start the job as soon as possible and to make the most efficient use of the full resources of the cluster:
On the NYU Prince cluster, SLURM manage the running and scheduling of jobs. As a user you will interact with SLURM, which accepts and runs job scripts and manages and monitors the cluster's compute resources, and also does the heavy thinking: the planning of which job should be run where and when.

### Partitions, Job Queuing, and QoS

Not all jobs can be run at once - the cluster is finite! - so when jobs are submitted they are placed into a queuing list. When a “space” becomes available in the schedule Slurm looks down the queue for the first job that will fit into the space.

Jobs are not necessarily placed at the end of the queue - Slurm uses the priority to determine where in the queue a job should be placed.

QoS is a set of limits recorded in Slurm database which will be applied to jobs. The limits could be defined on resources such as number of CPU cores, GPU cards, and also run time duration etc. A QoS can have an associated scheduling priority. In NYU HPC we configure QoS for different types of jobs, and has resource limits and priorities set on jobs accordingly. If you do not specify a QoS to submit to, Slurm will use the resources requested to select a QoS for you. Frequently this is the best option, however in some circumstances you are better off explicitly specifying one.

You can see the list of QoS with the command "sacctmgr list qos", and you can list a specific QoS in a controlled format with e.g. "sacctmgr list qos format=name,maxwall,maxtresperuser%40 where name=qosname".

**In almost all cases, do not specify a QoS** - the system will work out where to best place your job according to the resources requested.

---

Jobs needing fewer resources are easier to schedule - in the diagram above, a job requiring just 1 CPU for 1 hour could be inserted into the gap on Node 1 CPU 4. Smaller jobs are also more likely to receive priority when being scheduled. Therefore **avoid requesting vastly more CPUs, memory or walltime than you actually need**. Note that a small overestimate, such as 10%-20%, is wise, lest your job run out of time and be killed before it finishes, but requesting several times what you need will result in longer queueing time for your job and less efficient system utilization for everybody.