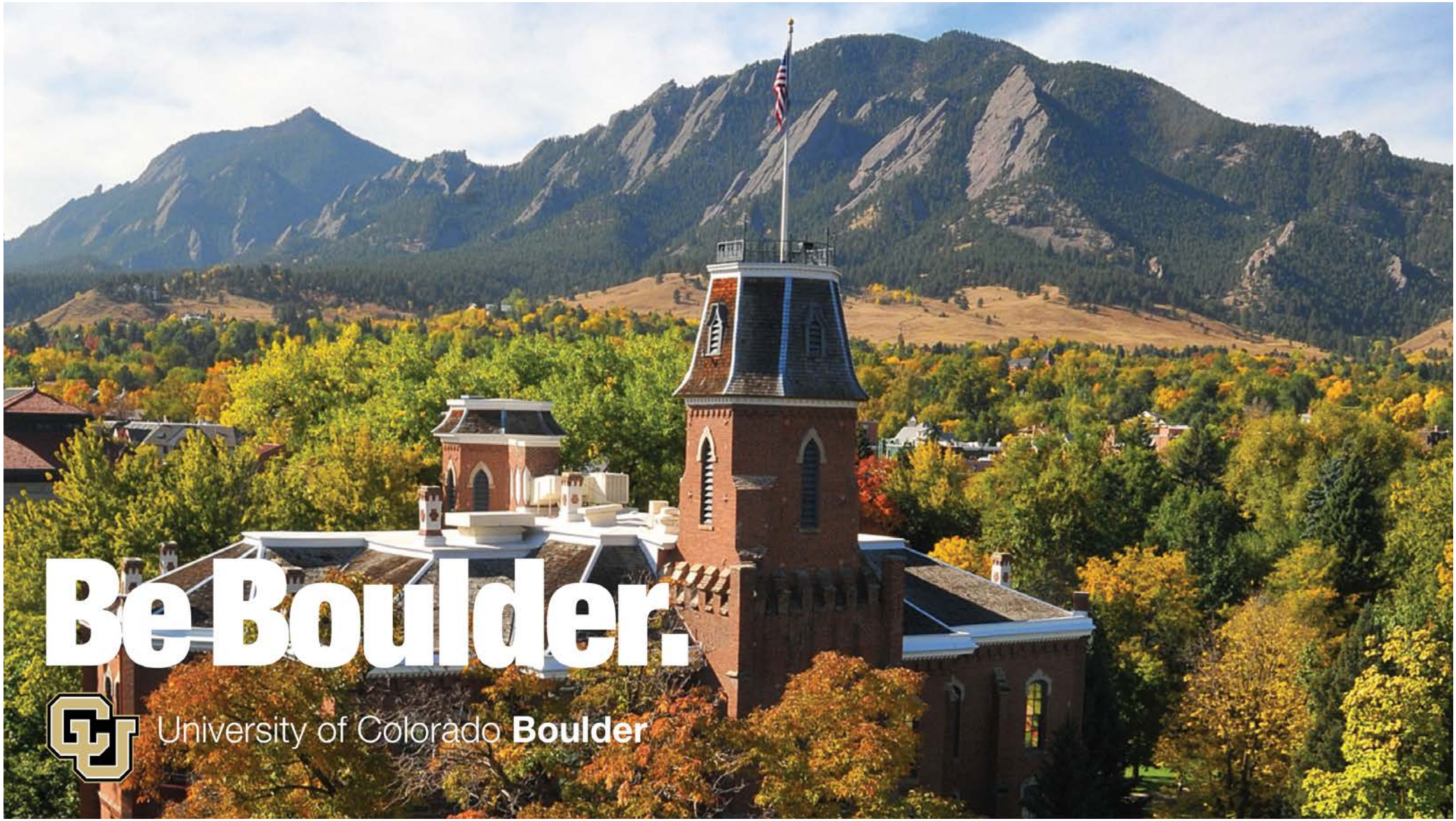

Be Boulder.
University of Colorado Boulder

Outline

- Speedup
- Strong scaling
- Week scaling
- Review – jobs on Yeti

Supercomputer Details

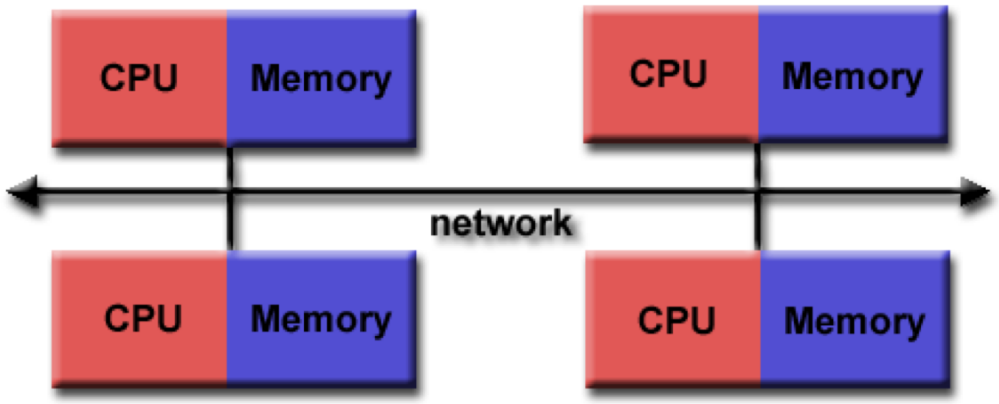
Distributed Memory Parallel Processing – Thought Experiment

- Jigsaw puzzle analogy**
  - Now we have two tables with one person at each table doing the puzzle
  - We split the puzzle equally between tables
  - Each person works completely independently
  - But to communicate costs more
    - How do you work out connecting the puzzle?
  - Can you really divide up the puzzle evenly?

**from Henry Neeman, OSCER, “Supercomputing in Plain English”

Distributed - memory Model

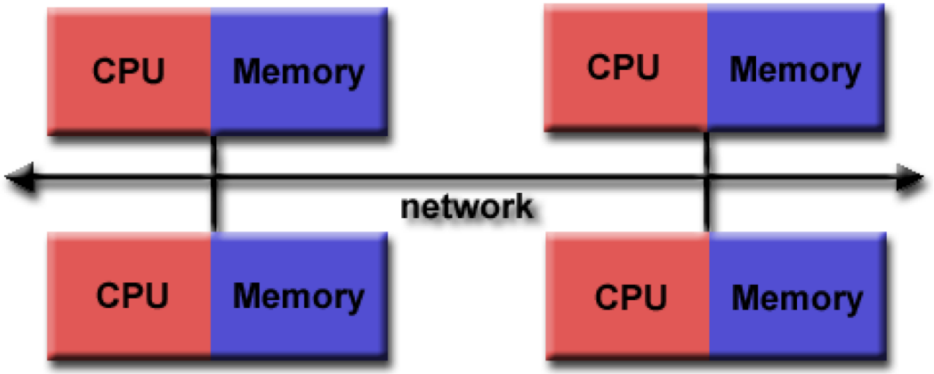
- Distributed memory requires a communication network to connect memory
- Processors have own memory and don't map globally



Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

Distributed-memory Model

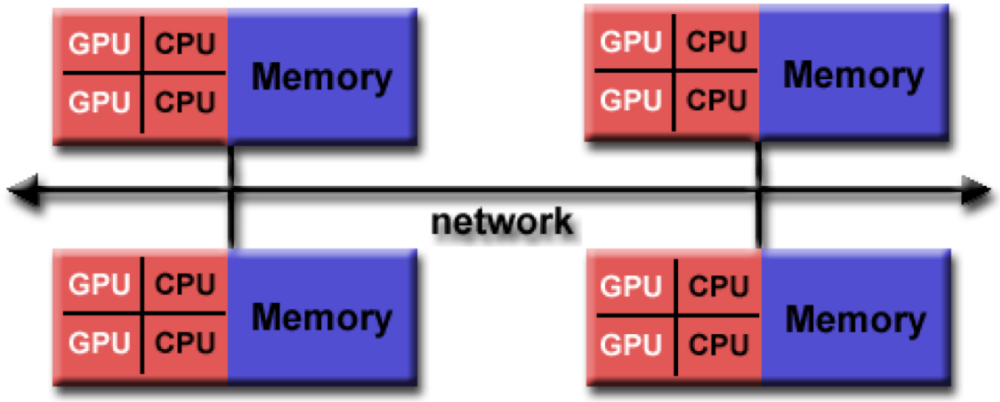
- Programmers explicitly define how processors access other processor's memory
- Advantage: scalable memory
- Disadvantage: need to know parallel programming!



Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

Distributed-Shared Memory

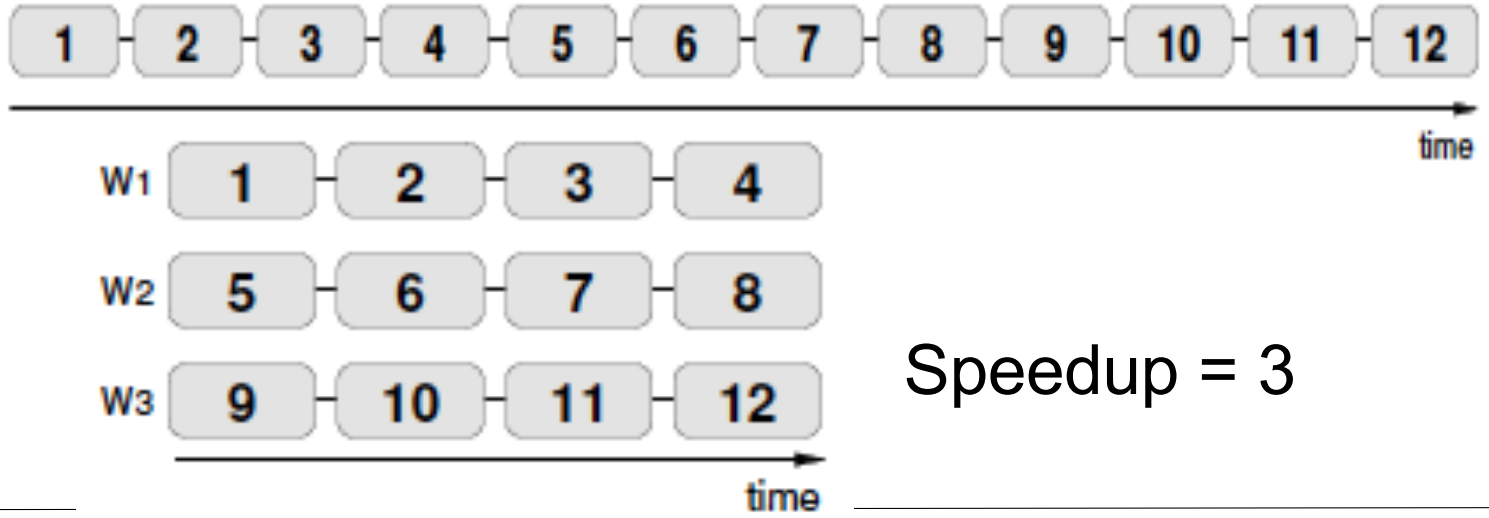
- Most large and fast computers now
- Shared memory machines connected to other shared memory machines



Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

Speedup Formula

$$\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$



Speedup = 3

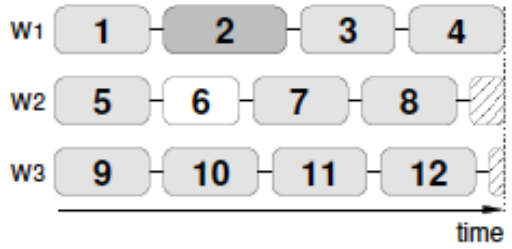
Execution Time Components

- Inherently sequential computations: $\mathrm{s(n)}$
- Potentially parallel computations: $\mathrm{p(n)}$
- Communication operations: $\mathrm{c(n,\ p)}$

- Speedup expression: $S \leq \frac{s+p}{s(n)+p/N+c}$

Parallel Overhead

- Overhead because of
  - Startup time
  - Synchronizations
  - Communication
  - Overhead by libraries, compilers
  - Termination time
- Other barriers to perfect speedup
  - Not perfectly load balanced



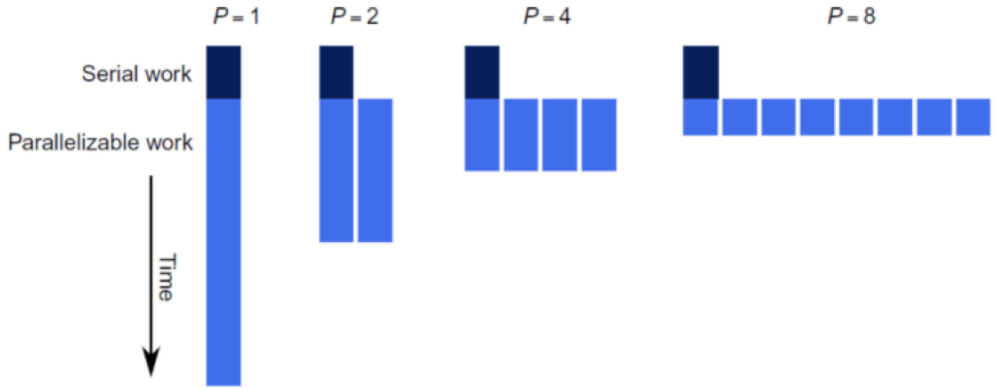
Efficiency

$$\text{Efficiency} = \frac{\text{Sequential execution time}}{\text{Processors} \times \text{Parallel execution time}}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Processors}}$$

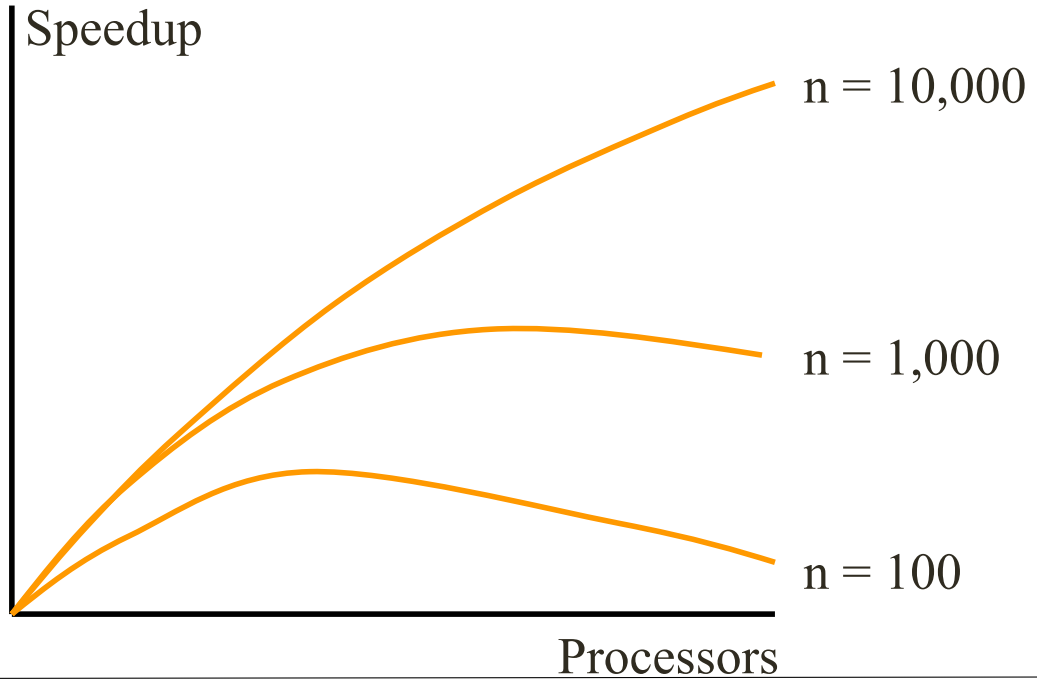
Strong Scaling

- Keep problem size the same
- Increase the number of processors



http://www.drdobbs.com/parallel/amdahls-law-vs-gustafson-barsis-law/240162980?pgno=2

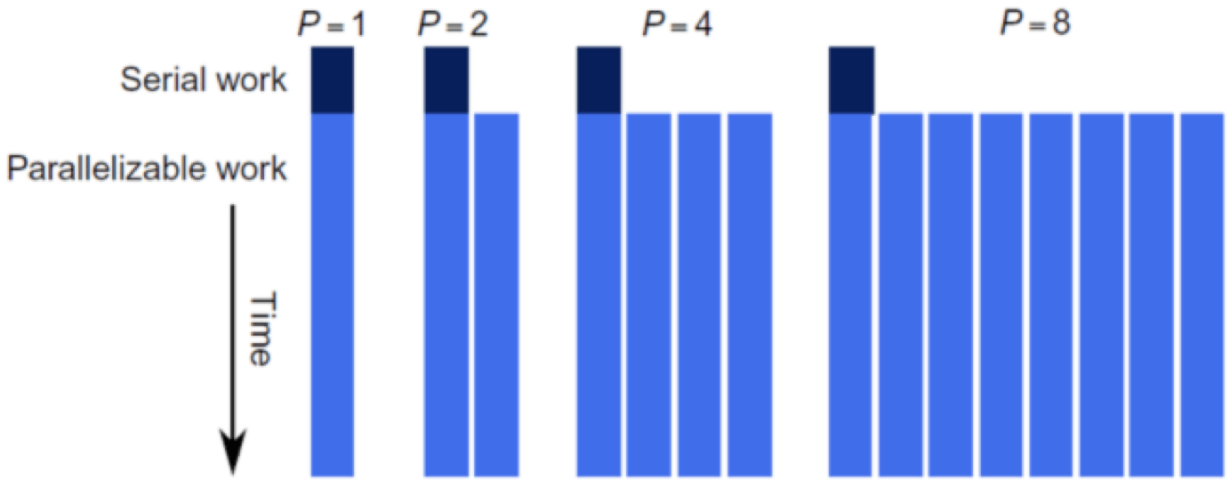
Effect of Problem Size



Another Perspective

- We often use faster computers to solve larger problem instances
- Let's treat time as a constant and allow problem size to increase with number of processors
- "…speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size" – John Gustafson

Weak Scaling



http://www.drdobbs.com/parallel/amdahls-law-vs-gustafson-barsis-law/240162980?pgno=2

Summary

- Speedup
- Strong Scaling
- Weak Scaling

Jobs

What is Job Scheduling

- Supercomputers usually consist of many nodes
- Users submit jobs that may run on one or multiple nodes
- Sometimes these jobs are very large; sometimes there are many small jobs
- Need software that will distribute the jobs appropriately
  - Make sure the job requirements are met
    - Reserve nodes until enough are available to run a job
    - Account for offline nodes
- Also need software to manage the resources
- Integrated with scheduler

Job Scheduling

- On a supercomputer, jobs are scheduled rather than just run instantly at the command line
  - People “buy” time to use the resources (allocation)
  - Shared system
  - Request the amount of resources needed and for how long
  - Jobs are put in a queue until resources are available
  - Once the job is run they are “charged” for the time they used

Job Scheduling - Priority

- What jobs receive priority?
  - Can depend on the center
  - Can arrange for certain people who “pay more” receive priority
  - Generally though based on job size and time of entry
- Might have different queues based on different job needs
- Can receive priority on a job by creating a reservation

Wall Times

- The maximum amount of time your job will be allowed to run
- How do I know how much time that will be?
- What happens if I select too much time?
- What happens if I select too little time?

Job Schedulers - Slurm

- Jobs on supercomputers are managed and run by different software
- Simple Linux Utility for Resource Management (Slurm)
  - Open source software package
- Slurm is a resource manager
  - Keeps track of what nodes are busy/available, and what jobs are queued or running
- Slurm is a scheduler
  - Tells the resource manager when to run which job on the available resources

Running Jobs

- What is a “job”?
- Interactive jobs
  - Work interactively at the command line of a compute node
- Batch jobs
  - Submit job that will be executed when resources are available
  - Create a text file containing information about the job
  - Submit the job file to a queue
- Load the Slurm module!

Useful Slurm Commands

- **sbatch**: submit a batch script to slurm
  - Standard input (keyboard)
  - File name
    - Options preceded with #SBATCH
- sbatch exits immediately after receiving a slurm job ID
- By default, standard output and errors go to file named slurm-%j.out (job allocation number)
- Slurm runs a single copy of the script on the first node in the set of allocated nodes

http://slurm.schedmd.com/sbatch.html

http://slurm.schedmd.com/sbatch.html

SBATCH Options

- In batch script put:
  #SBATCH <options> OR sbatch <options>

- **Account:** **`-A <account_name>`**
- Checkpoints: `--checkpoint=<interval>`
- Sending emails: `--mail-type=<type>`
- Email address: `--mail-user=<user>`
- Number of nodes: `-N <nodes>`
- Reservation: `--reservation=<name>`
- **Wall time:** **`-t <wall time>`**
- Job Name: `-J <jobname>` or `--job-name=<jobname>`
- **Partition:** **`-p <partition_name>`**

Queues

- There are several ways to define a “queue”
- Clusters may have different queues set up to run different types of jobs
  - Certain queues might exist on certain clusters/resources
  - Other queues might be limited by maximum wall time
- Slurm can use a “quality of service” for each queue
  - aka “QOS”
- Also can use a “partition” (or set of nodes) that corresponds to a queue

Partitions

- UV: SGI UV2000 shared memory, cache coherent
  - 256 cores, 4TB memory (16GB/core)
  - Can see all processors and all 4TB memory from a single operating system

- Normal: Cray, distributed memory cluster
  - 1200 cores, 7.68TB RAM (128GB/core)
  - 60 compute nodes, each with 20 cpu cores

Software

- Common software is available to everyone on the systems
- To find out what software is available, you can type `module avail`
- To set up your environment to use a software package, type `module load <package>/<version>`
- Can install your own software
  - But you are responsible for support
  - We can assist

Login to Yeti

- Step1. Log in to Yeti.

```
Laptop ~$  ssh name@yeti.cr.usgs.gov
```

- Step 2. Clone the git repository

```
yeti-login01 ~$ git clone  \
 https://github.com/ResearchComputing/USGS_2018_02
```

3a. Start an interactive compute job on UV partition

```
yeti-login01 ~$ sinteractive -A training    \
                -p UV                       \
                -t 01:00:00  -n 1           \
                --cpus-per-task=4           \
                --reservation=training_UV       \
                --gres=gpu:tesla:[1-6]
```

3b. Start a interactive compute job on normal partition

```
yeti-login01 ~$ salloc -A training          \
                -p  normal                  \
                -t  01:00:00  -n  1         \
                --cpus-per-task=4           \
                --reservation=training_normal
```