Introduction to High-Performance Computing

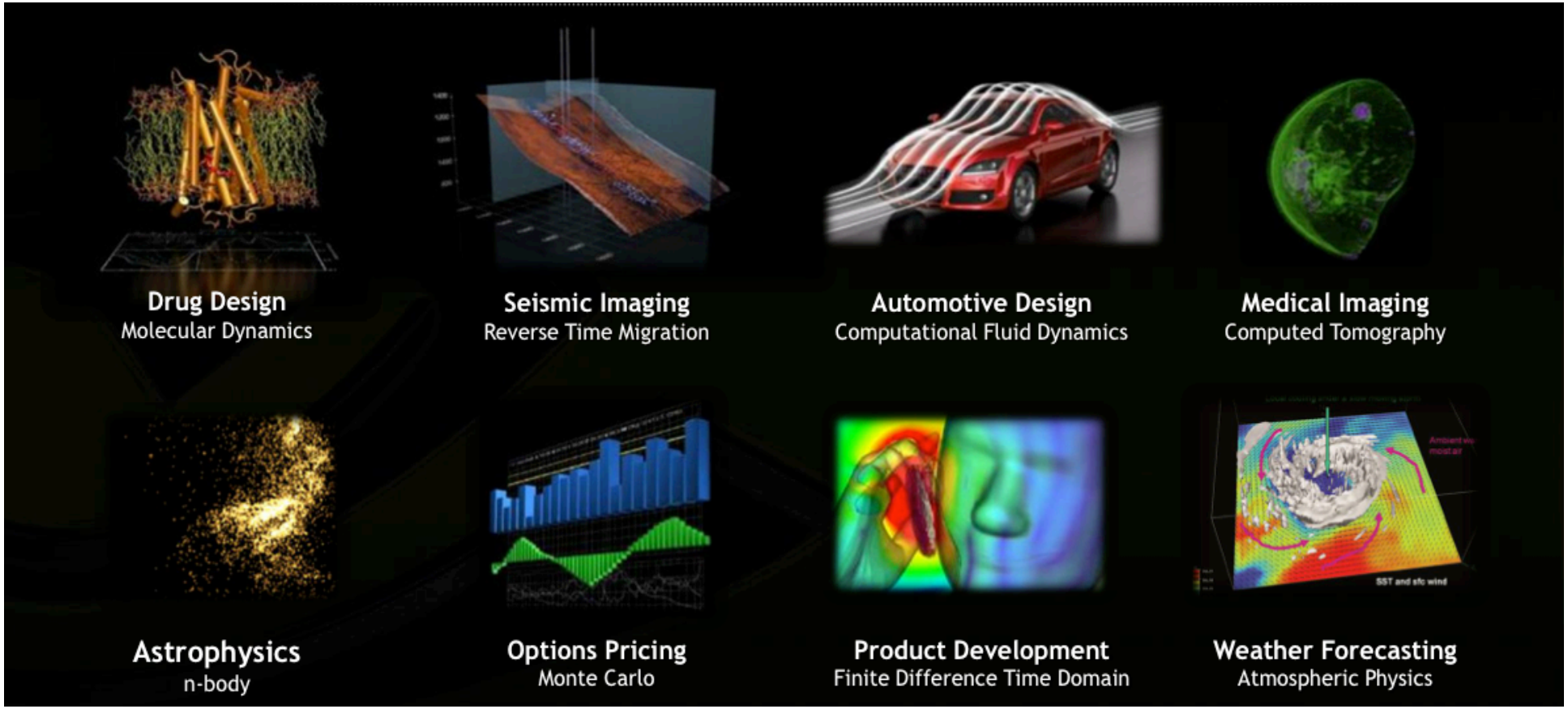
Lecture's objectives

- Explain different types of "big problem"
- Compare and contrast different types of parallel computer architecture
- Identify the difference between node, socket, and core

Scientific discovery

- Historically, people said science stands on two legs:
  - Experiment
  - Theory
- Then, science got the third leg:
  - Computing
- Now, science has the fourth leg:
  - Big data

Computing: the 3rd pillar



Computing and *Big* problems

- Computing-based science discovery deals with Big problems

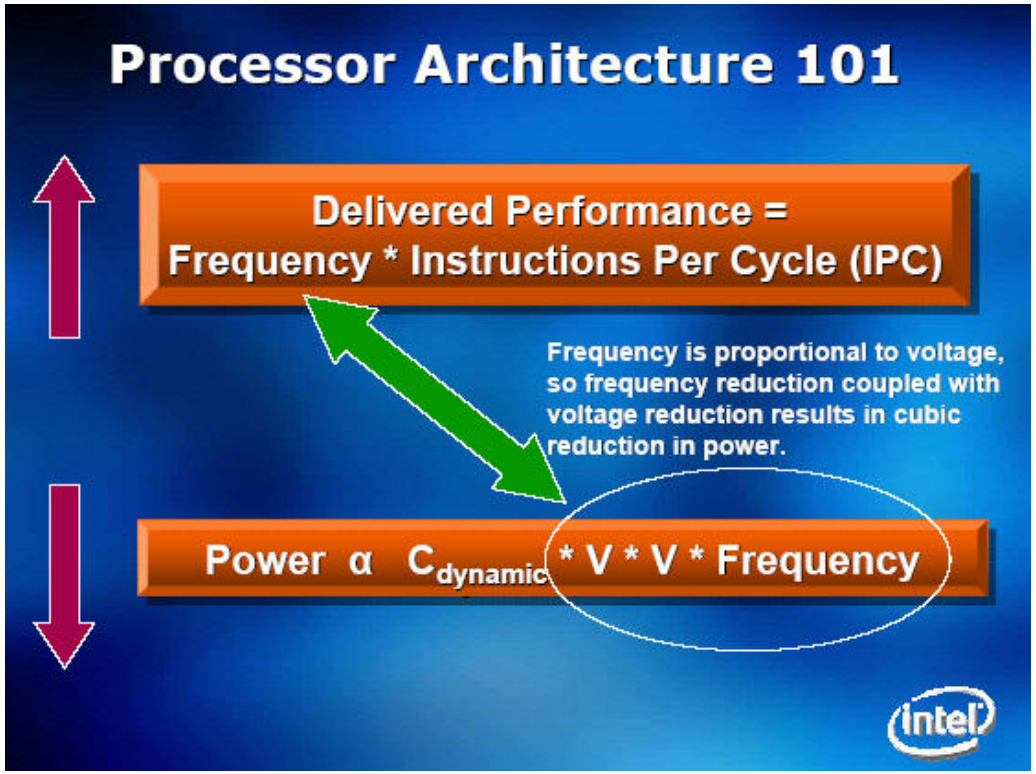
- For example:
  - It will take many days to perform a very short Molecular Dynamic (MD) simulation of 20,000 atoms
  - Whole genomic analysis of 100 human genomes using currently available desktop computers could take 30+ years

Types of *Big* problem

- Compute Intensive
  - A single problem requires a large amount of computation
- Memory Intensive
  - A single problem requires a large amount of memory
- Data Intensive
  - A single problem operates on a large amount of data
- High Throughput
  - Many copies of the same problem to be executed in parallel

Big problems require *Big* computers

- Faster CPUs
  - Processor clock speeds have flattened out
  - The fastest commercially available chip is about 5.0 GHz
  - Clock speed is limited by power consumption, heat dissipation, current leakage



Big problems require *Big* computers

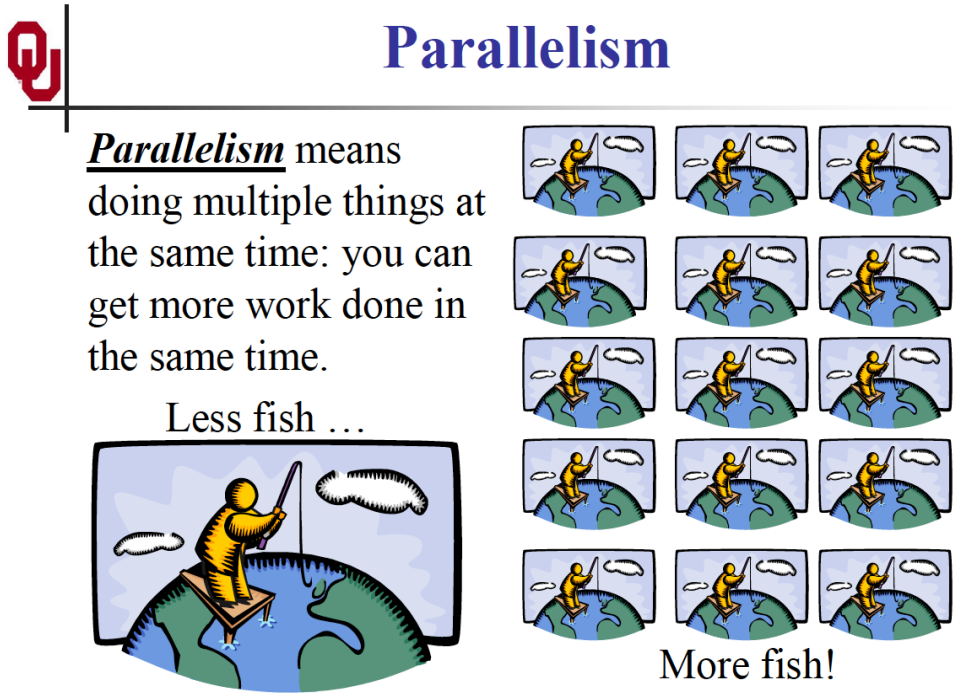
- Many CPUs
- Lots of memory
  - Limited by how much memory a CPU can support
- Large amount of storage space

Parallelism, the path toward future performance gains

  - Trend is toward multi-core and many core

What is HPC?

- High Performance Computing is the “practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop/laptop computer”[1]



[1]http://insidehpc.com/hpc-basic-training/what-is-hpc/
Picture from “supercomputing in plain English”, http://www.oscer.ou.edu/education.php

Parallel Computer Architectures

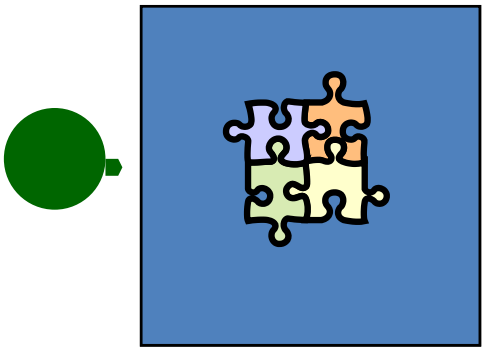
The Jigsaw Puzzle

- Say you have a jigsaw puzzle with 1000 pieces
- How can you put it together as fast as possible?

Adapted from “supercomputing in plain English”, http://www.oscer.ou.edu/education.php

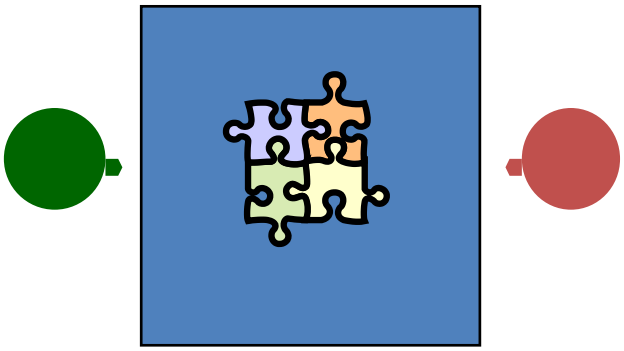
The Jigsaw Puzzle

- Serial Computing
  - You sit down at the table by yourself and put together all 1000 pieces, one after the next
  - It takes you 1 hour to assemble the puzzle
  - Well done!
  - But can we do better?



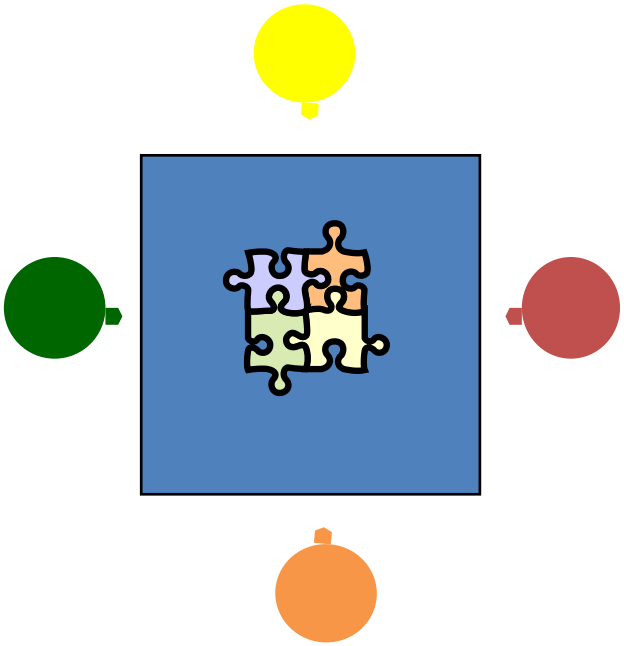
The Jigsaw Puzzle

- Shared Memory Parallelism
  - If your friend sits across from you, then he can work on his half of the puzzle and you can work on yours
  - Once in a while, you'll both reach into the pile for the same piece (you will **contend** for the same resource) which causes a little slowdown
  - From time to time, you'll have to work together (**communicate**) at the interface of your halves
  - If all goes well, you'll get a 2x speedup
  - But we have communication and contention overhead so it will probably take more like 35 minutes instead of 1 hour



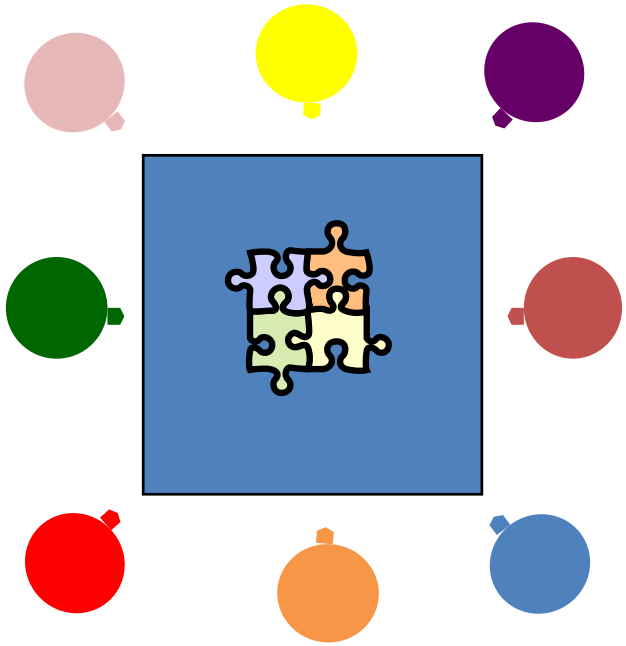
The Jigsaw Puzzle

- More Shared Memory Parallelism?
  - Let's say you add two more friends
  - Each person works on their quadrant of the puzzle, we should get a 4x speedup, right?
  - But, there will be a lot more contention for pieces and a lot more communication
  - Instead of putting together puzzle in ¼ hours or 15 minutes, you'll probably be closer to 20 minutes



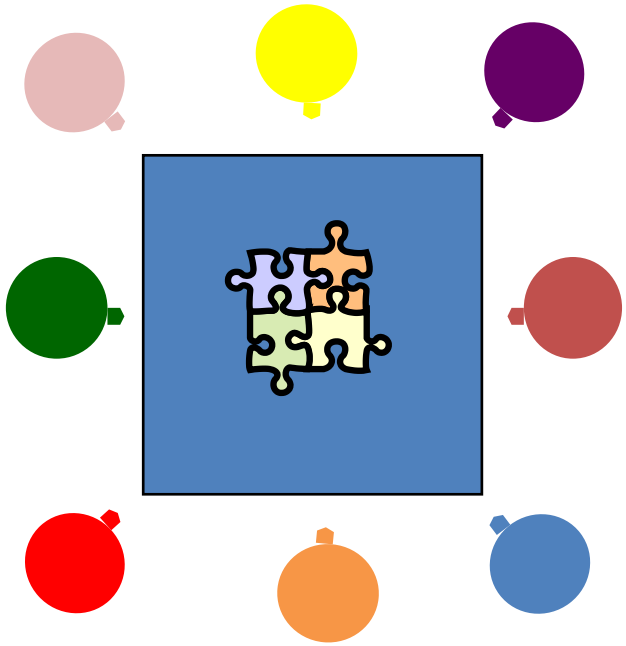
The Jigsaw Puzzle

- Diminishing returns?
  - Let's say you add 4 more friends
  - MUCH more contention for pieces and a TON more communication
  - If you actually manage to get a 8x speedup it would be very impressive



The Jigsaw Puzzle

- But...
  - Let's assume the 8 of you are extremely good at working together
  - Maybe you can get an almost 8x speedup
  - But we want to go faster yet
  - Problem! There's only enough space for 8 people at the table!

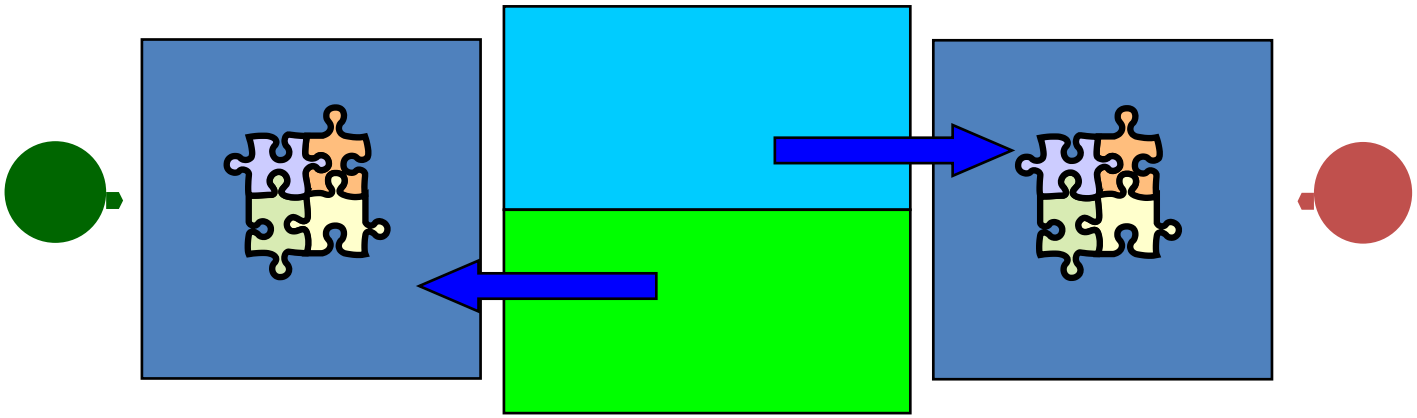


The Jigsaw Puzzle

- Distributed Parallelism
  - You sit at table 1 and your friend sits at table 2
  - PRO: Plenty of elbow room
  - PRO: You can work without contention for resources
  - CON: Communication is much more difficult. You need to carry pieces from one table to the other to assemble them
  - Other problems?

The Jigsaw Puzzle

- Load balancing and Domain Decomposition
  - Say the puzzle is half blue sky and half green grass
  - Put all the blue pieces on one table and all the green pieces on the other
  - Pieces/table are roughly equal so each half should be assembled in roughly equal amount of time



A word about load balancing

- **Q:** How long does it take to finish 100 parallel calculations that all start at the same time?

A word about load balancing

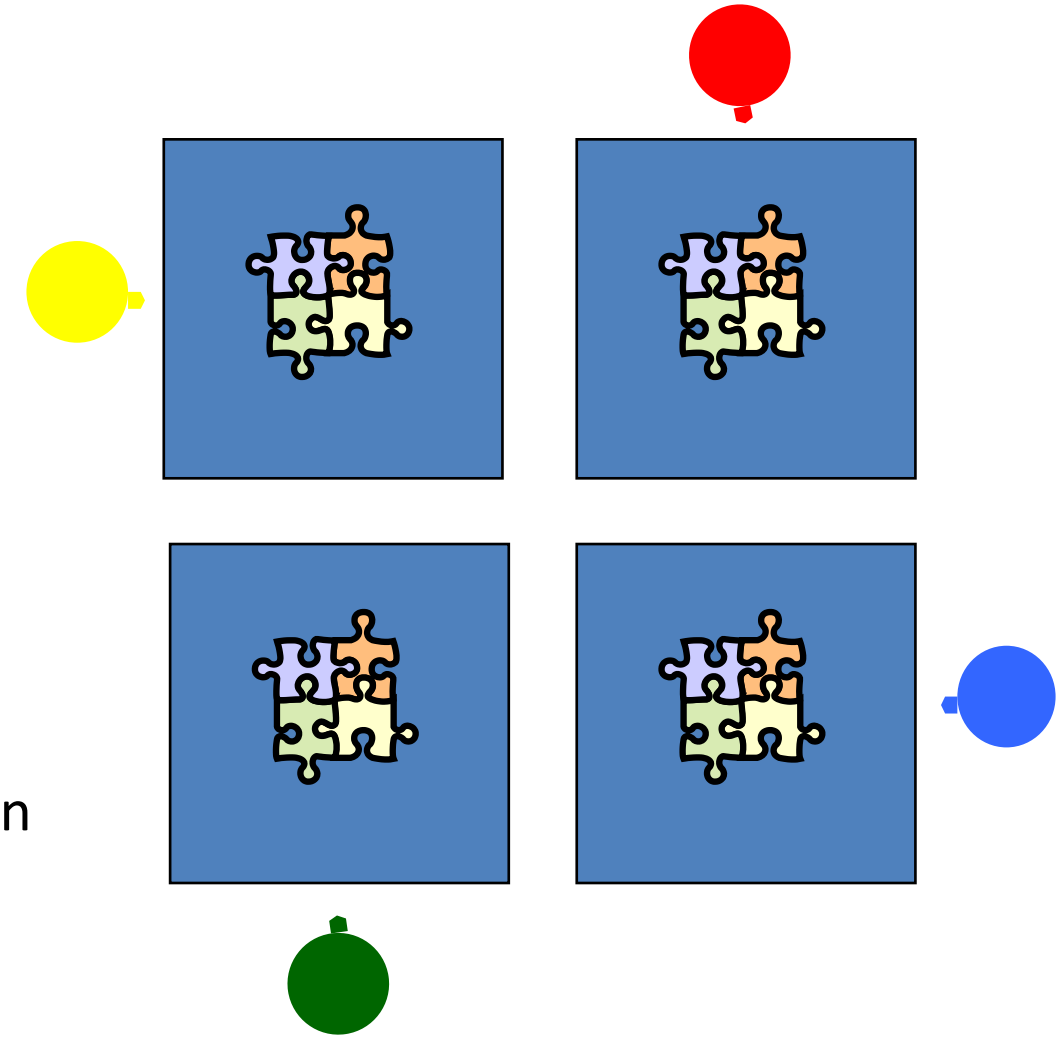
- **Q:** How long does it take to finish 100 parallel calculations that all start at the same time?

- **A:** However long the slowest of the 100 calculations takes

If one calculation is significantly slower than all the rest, parallelism will not help you

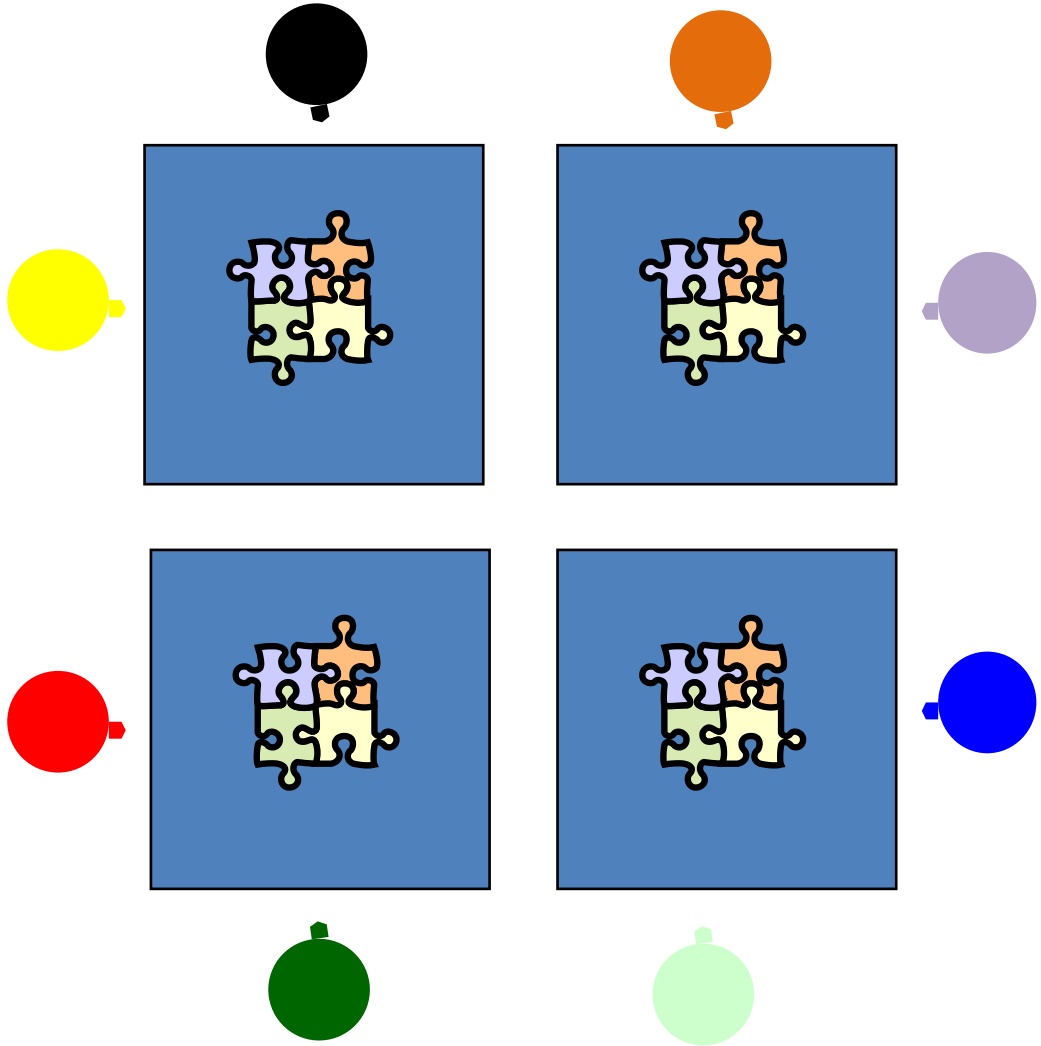
The Jigsaw Puzzle

- More Distributed Parallelism?
  - It's very easy to keep adding more tables
  - But…
    - Communication
    - Load balancing
    - Domain decomposition



The Jigsaw Puzzle

- Hybrid Parallelism
  - Combine distributed and shared models
  - What are the problems?
  - What about advantages?



Some definitions

- **Core**: smallest computation unit that can run a program (used to be called a processor, still is, also called a CPU — Central Processing Unit)

- **Socket**: a computational unit, packaged as one and usually made of a single chip often called processor. Modern sockets carry many cores (2, 4 on most laptops, 8 to 16 on most servers)

- **Node**: a stand-alone computer system that contains one or more sockets, memory, storage, etc. connected to other nodes via a fast network interconnect

Question

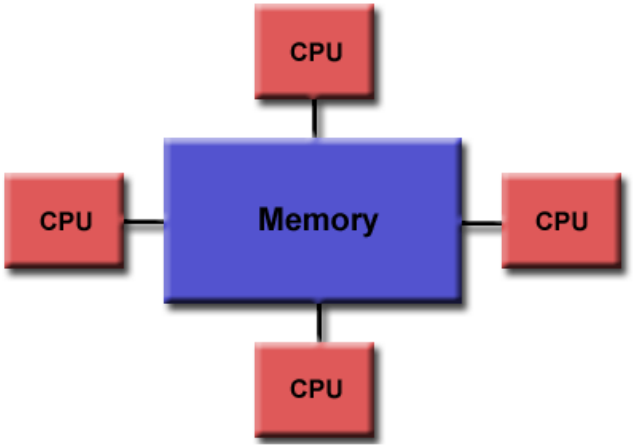
- Find out how many cores, sockets, how much main memory (aka RAM), and hard drive your laptop has.

A super computer is...

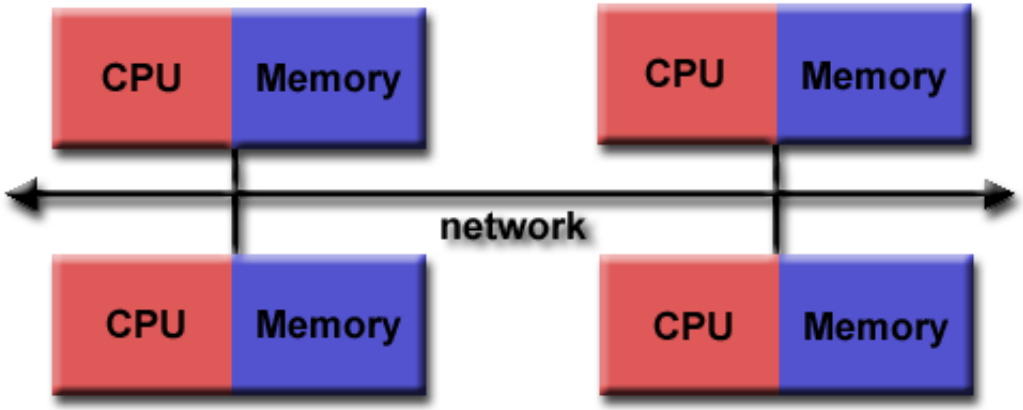
- A collection of small computers, called **nodes**, hooked together by an **interconnection network** (or interconnect for short)
- Software that allows nodes to communicate with one another
- All of these nodes work together as if they are one big computer … a supercomputer!

Parallel computer architectures

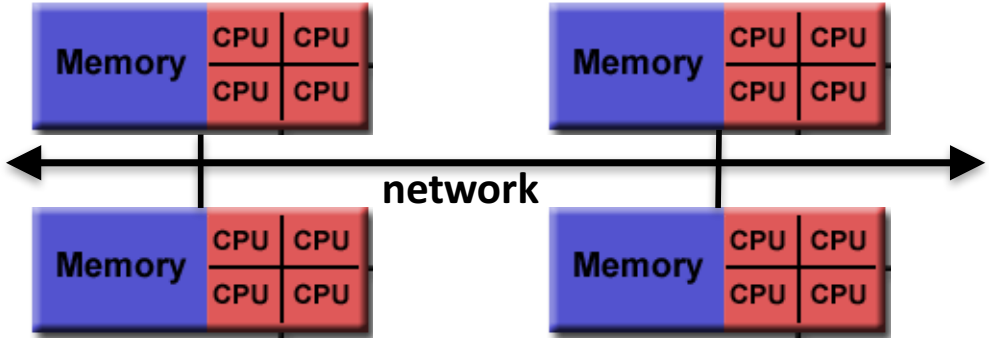
Shared Memory



Distributed Memory



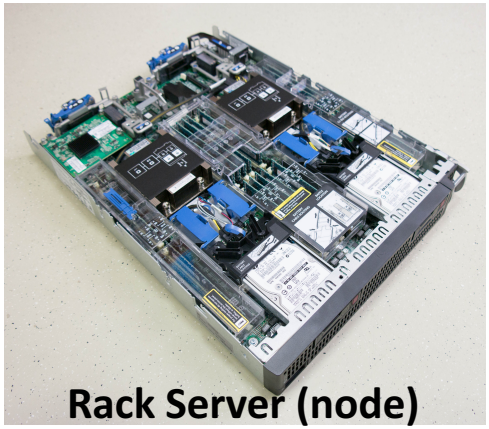
Hybrid



Images from https://computing.llnl.gov/tutorials/parallel_comp/

What is HPC?

From node to cluster



Rack Server (node)

Cabinet

Cluster

Data Center

Fastest super computers

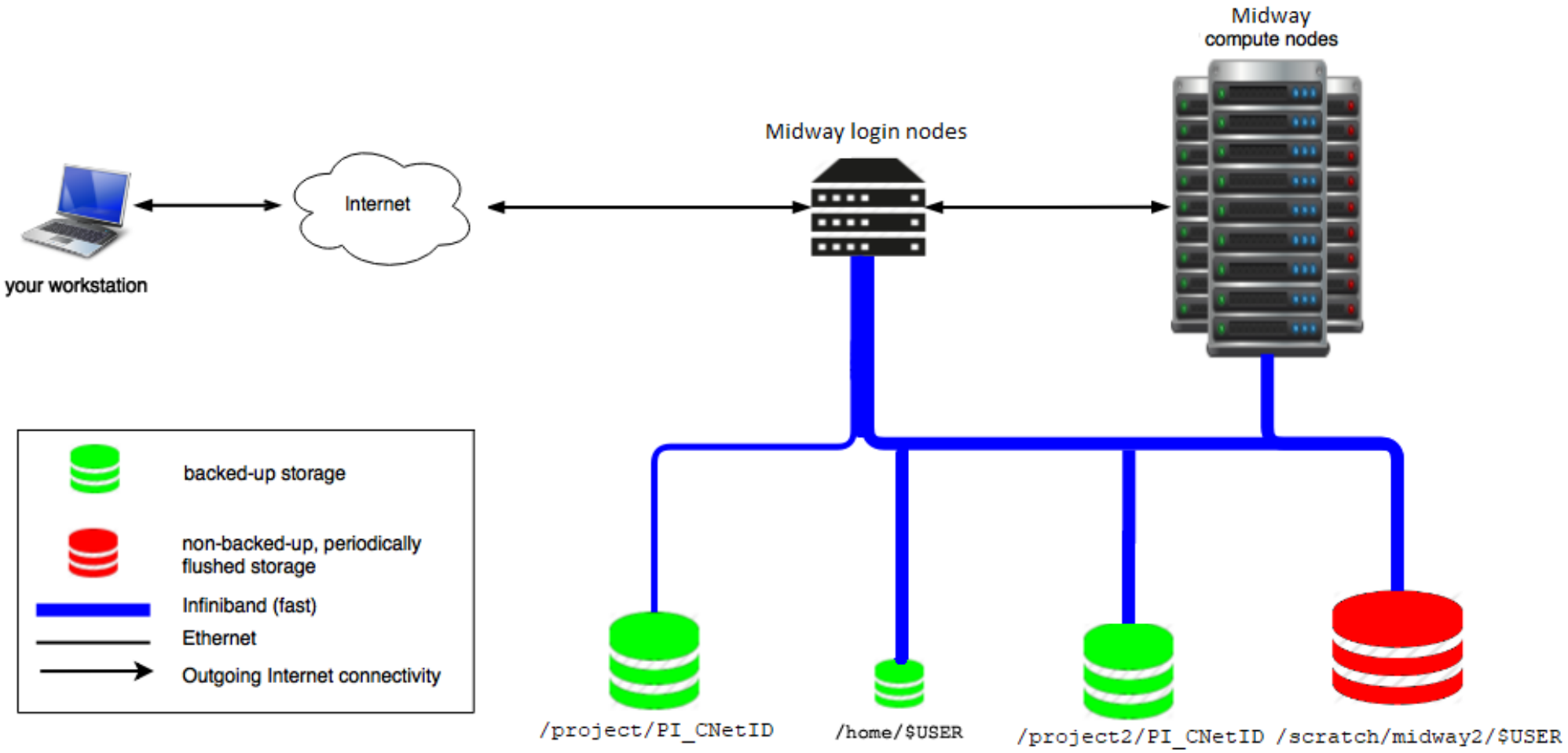
1. Summit - Oak Ridge National Laboratory
   - Cores: 2,414,592 Pflop/sec: 148.60 Power: 10.096 MW
2. Sierra Lawrence Livermore National Lab
   - Cores: 1,572,480 Pflop/sec: 94.64 Power: 7.438 MW
3. Sunway TaihuLight National Supercomputing Center in Wuxi, China
   - Cores: 10,649,600 Pflop/sec: 93.01 Power: 15.371 MW

1 Pflop = 1,000,000,000,000,000 floating point operations

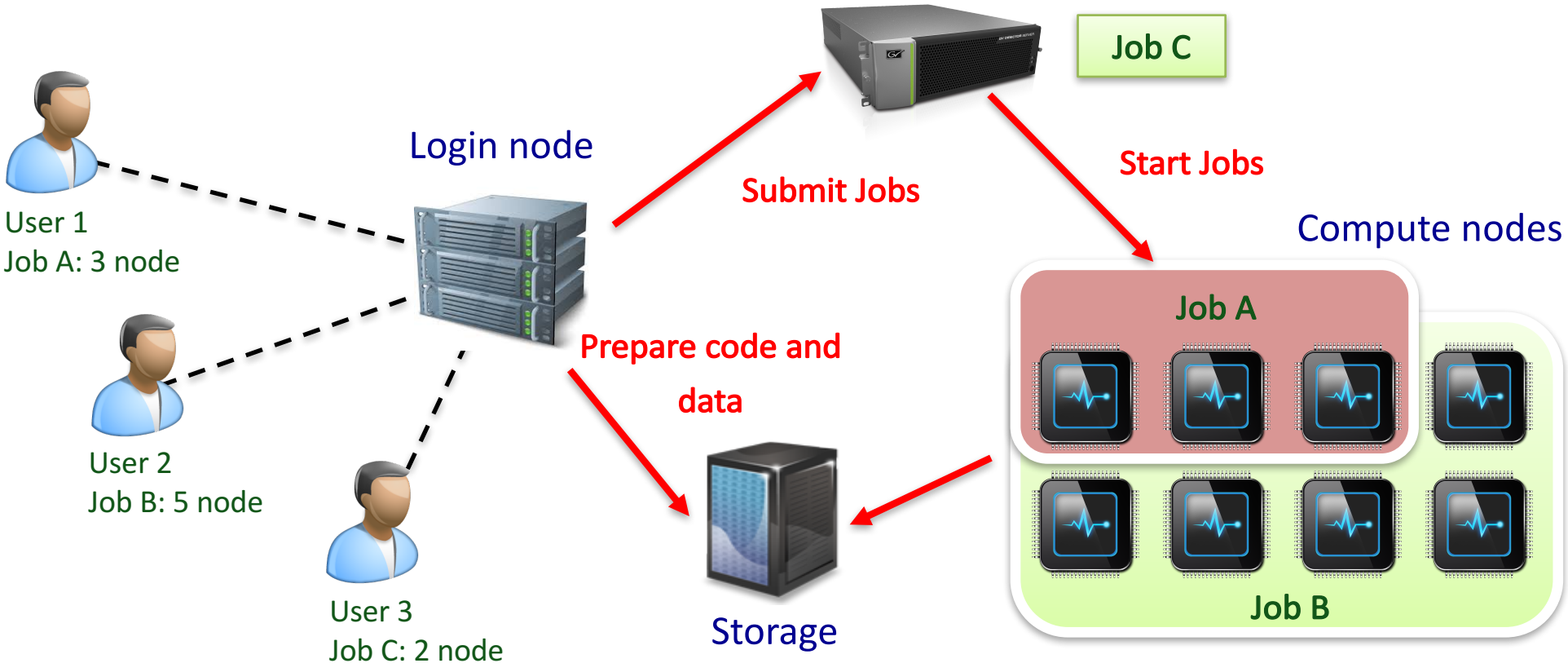
The RCC's Midway system

- Compute
  - Login nodes
  - Compute nodes
    - 300+ Tightly coupled nodes
    - 4 Large memory nodes
    - 6 GPU nodes
- Storage
  - home, project, and scratch spaces
- Software
  - Commercial as well as open source

Schematic of a typical cluster



Job scheduling on Midway



Takeaways

- Computing is the third pillar of scientific discovery
- Computing-based scientific discovery deals with Big problems
- A parallel computer (aka supercomputer) is a collection of small computers working together
- Shared memory, distributed memory, and hybrid are types of parallel computer architectures

References

- The following resources were used in preparation of this presentation:
  - D. Eadline, High Performance Computing for Dummies, Second AMD Special Edition, Wiley Publishing Inc., 2011.
  - S. Rankin, An Introduction to High Performance Computing, https://www.hpc.cam.ac.uk/files/introduction_to_hpc-nov2018-handout.pdf
  - S. Lantz, Effective Use of High Performance Computing, http://www.cac.cornell.edu/slantz/CIS4205/, 2009.
  - G. Hager and G. Wellein, Introduction to High Performance Computing for Scientists and Engineers, CRC Press, 2010.

Allocation Policy

- Allocations of projects accounts provided throughout the year.
- All Principal Investigators (PIs) Automatically receive a startup account.
- Larger allocations (Research II) to be reviewed.
- Allocation to expire on September 30th of each year.
- To check account balance:
  - Accounts balance: accounts balance
  - Accounts usage: accounts usage

https://rcc.uchicago.edu/accounts-allocations/request-allocation

Midway User Guide

- Best first place to check if have questions about using the resource.

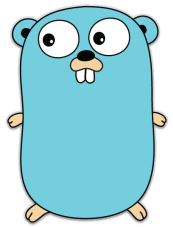
  https://rcc.uchicago.edu/docs/using-midway/index.html

- If still have questions after consulting the User Guide, write to help@rcc.uchicago.edu

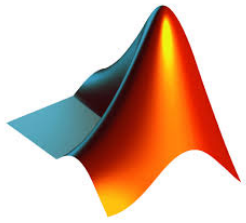
Software on Midway2

- **System Tools and Math Libraries**
  - Compilers: Intel, Portland, GCC
  - Debuggers: Alinea DDT, Intel Roofline
  - Python, Perl, Ruby interpreters
  - MPI, MKL, FFTW, GSL, CUDA, etc.



- **Commercial software**
  - MATLAB, STATA, AMIRA, Gaussian, etc.


- **Domain Specific Applications**
  - chemistry
  - biology
  - Physics
  - etc.

- Use **module system** to manage user's software environment.

Up to date list of software modules installed on midway:

https://rcc.uchicago.edu/docs/software/modulelist.html

Module System

HPC centers typically use a software module system to manage the software packages that loaded into your environment.

This is useful in that you don't have to install the software your self and can selectively choose which software packages are accessible to you so that you can possibly avoid software conflicts.

Useful Module Commands

```
[johnnyb@midway1]$ module help       # information about using module

[johnnyb@midway1]$ module list       # list your currently loaded modules

[johnnyb@midway1]$ module avail      # list all avail software packages

[johnnyb@midway1]$ module load  <package>  # load <package> into your env

[johnnyb@midway1]$ module unload <package> # unload <package> from env
```

Using Midway Module System

- To use a particular software package load the package/version name. The default version will load if the version is not specified

```
[johnnyb@midway1]$ module load python

[johnnyb@midway1]$ module list
```

- The module load command appends to your $PATH and $LD_LIBRARY_PATH the locations of the compilers and their accompanying libraries.

```
[johnnyb@midway1]$ echo $PATH; echo $LD_LIBRARY_PATH
```

- Notice that not just the python module has been loaded. Any other software that the module you load is dependent upon will also be loaded. Keep this in mind as this can potentially lead to conflicts with other software you may load/use. List info about the module package:

```
[johnnyb@midway1]$ module show python
```

Python Modules on Midway

There is a plethora of python modules on midway. How then to choose which to use?

```
[johnnyb@midway1]$ module avail python

------------------------ /software/modulefiles ------------------------
python/2.7                              python/2.7-2015q1
python/3.3                              python/2.7-2015q2(default)
python/3.4-2015q1                       python/2.7.12+gcc-4.7
python/3.5.2+gcc-4.8                    python/2.7.12+gcc-6.1
python/3.5.2+intel-16.0                 python/2.7.12+intel-16.0
--------------------------- /etc/modulefiles ------------------------
```

```
[johnnyb@midway1]$ module avail Anaconda2
   Anaconda2/4.1.1(default)

[johnnyb@midway1]$ module avail Anaconda3
Anaconda3/4.1.1(default)
```

Python Modules on Midway

There is a plethora of python modules on midway. How then to choose which to use?

- There are two main different python versions (python2.7 and python3.x) which are indicated in the module version.

- Some installs are built with different system compilers (gcc, intel) This is important if coupling your python code with other compiled software (want to have homogeneity in compilers/libs).

- Not all python modules have the same packages. You can check this with pip or conda (Anaconda distribution only).

- Recommendation is to use the latest Anaconda3 module.

Running Jobs on Midway

- Interactive jobs

  Interactively login to a node to run directly from the command line:

```
[johnnyb@midway2]$ sinteractive --partition=edu –time=1:00:00
```

- Submitting jobs to Slurm scheduler (sbatch jobs)

  To submit a job:

  sbatch submission_script

```
[johnnyb@midway2]$ sbatch job.sbatch
```

Example sbatch script

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --output=test.out
#SBATCH --error=test.err
#SBATCH --partition=edu
#SBATCH --nodes=1
#SBATCH --account=env_bootcamp
#SBATCH --ntasks-per-node=4
#SBATCH --time=1:00:00

module load Anaconda3/2019.03

python mycode.py
```

Running a python code from CLI

- You should first start an interactive session on a compute node:

```
[johnnyb@midway2]$ sinteractive –partition=edu –time=1:00:00
```

- Load the python module you wish to use and run your python code from the CLI

```
[johnnyb@midway2]$ module load Anaconda3/2018.12

[johnnyb@midway2]$ python example0.py
```

- You can access the same compute node we have an interactive session started on by ssh'ing from midway-login to the compute node hostname. Open separate terminal and do so.

Inspecting Your Submitted Job

- The user can directly login to the node that their job is running on and observe the progress of the job.

```
squeue –u <userid>
```

Will return job ids of jobs you have running or pending.
Get a running jobs list of nodes it is using:

```
squeue -O nodelist -j <job_id>
```

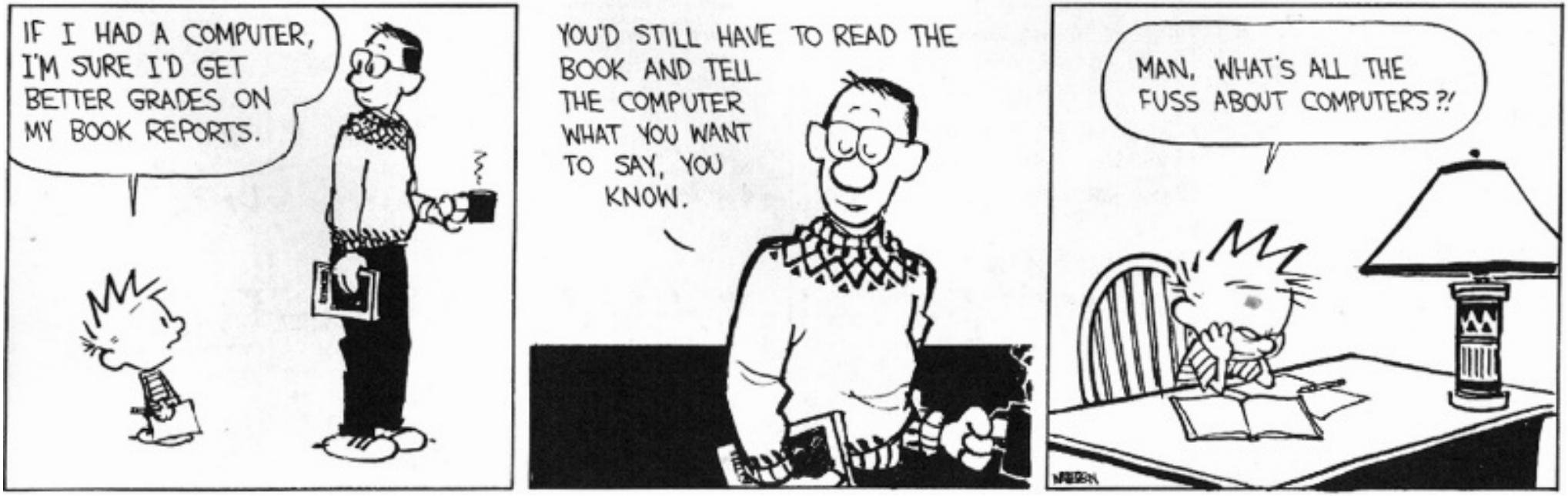
Login directly to that node and run commands `top` and `free –g`

If the job is not running check its priority:

```
squeue -O prioritylong -j <job_id>
```

```
squeue -p sandyb –O jobid,numnodes,reason,timelimit,prioritylong
```

RCC Help



- Email: help@rcc.uchicago.edu
- Web: rcc.uchicago.edu
- Phone: 773-795-2667
- Walk-in: Regenstein Library, suite 216