

# Parallel computing infrastructure

Simon Scheidegger  
simon.scheidegger@gmail.com

July 16<sup>th</sup> 2019

Open Source Macroeconomics Laboratory Boot Camp – BFI/UChicago

# Outline

- **Make first steps on a Linux Cluster**

Login via ssh, remotely, short overview of basic unix commands like cd, pwd, cp, scp,...

- **Submit jobs to the queue**

Slurm

- **Get lecture notes**

Clone a git repository

# <https://rcc.uchicago.edu>

For this course, we use the Uchicago's **Midway compute cluster**.

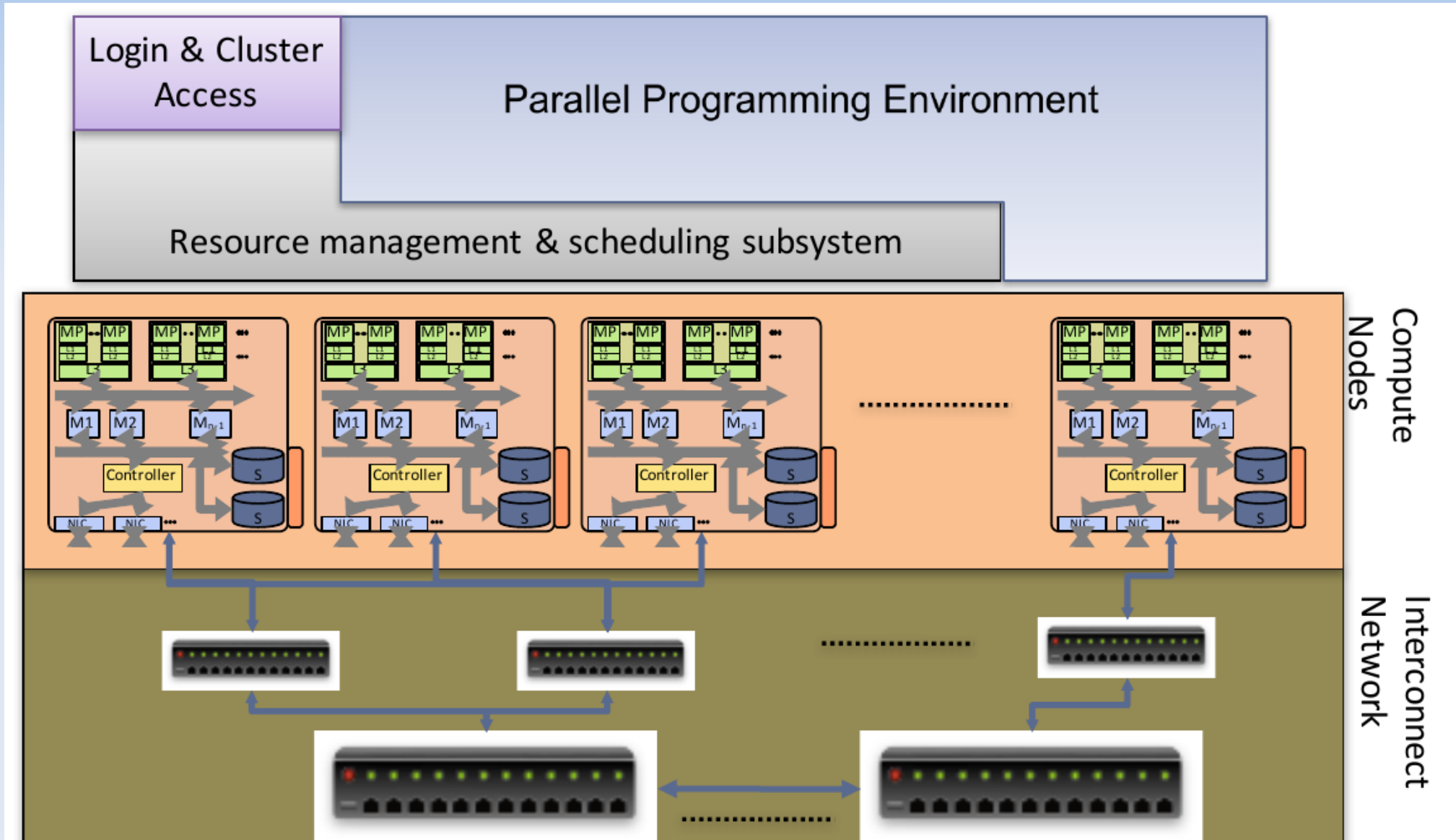
It is managed by the **R**esearch **C**omputing **C**entre

→ Its setup is very similar to any other top system

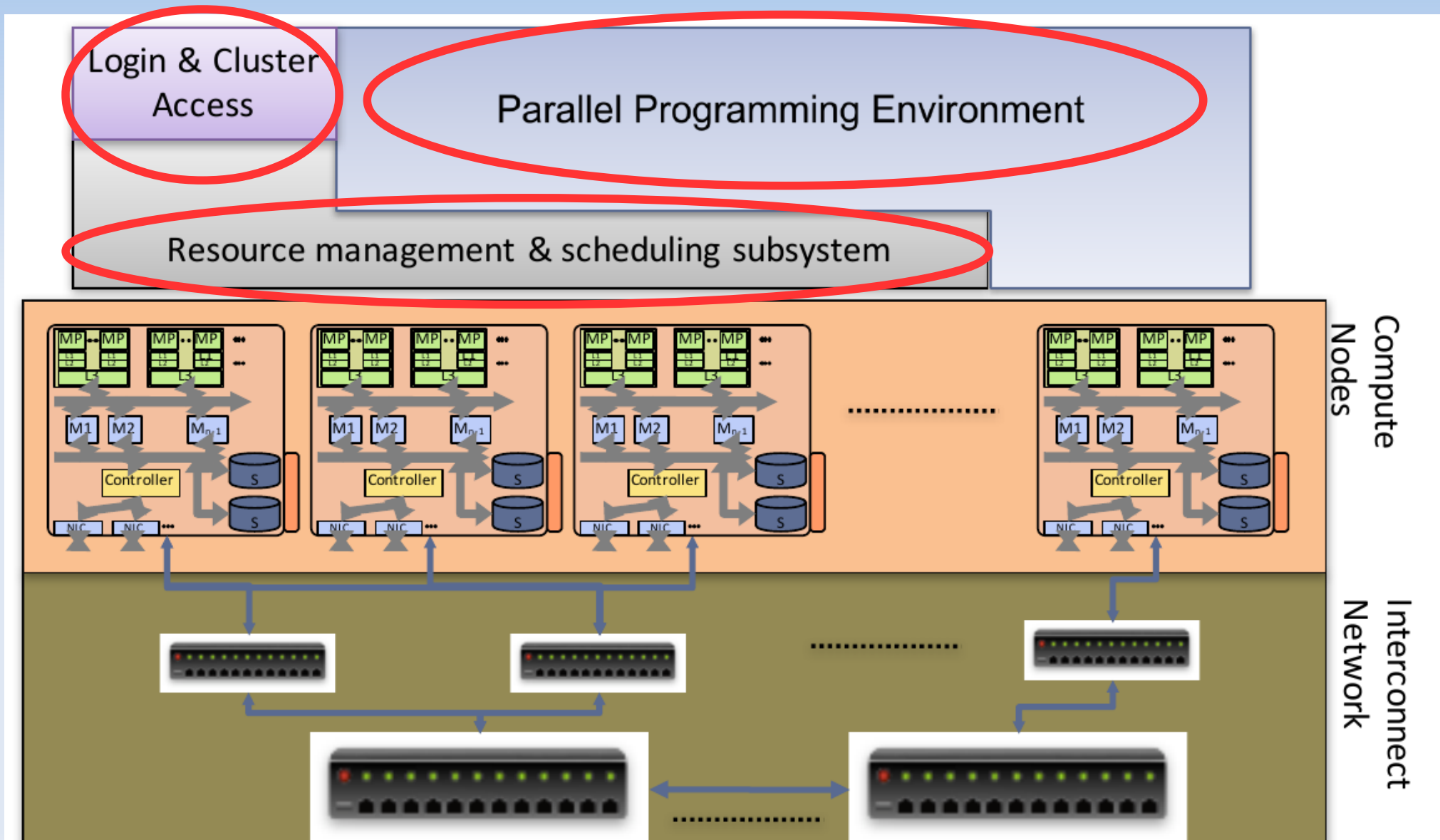
For the RCC Manual see the documentation site at

<http://docs.rcc.uchicago.edu>

# An abstract compute cluster



# An abstract compute cluster



# A real compute cluster



# Login for participants

- If you don't have an account on Midway request one (infrastructure for this course)  
<https://rcc.uchicago.edu/docs/using-midway/index.html>
- For MS-Windows users: Download and install Putty  
→ <http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe>  
→ Download and install Winscp  
→ <http://winscp.net/download/winscp576setup.exe>

# Basic Linux commands (1)

Command	Description
<code>pwd</code>	Print name of current/working directory
<code>cd [Directory]</code>	Change directory (no directory → change to home)
<code>ls [Directory]</code>	List directory contents (no directory → list current)
<code>cat FILE</code>	Concatenate files and print on the standard output
<code>mkdir DIRECTORY</code>	Make directories
<code>mkdir -p DIRECTORY</code>	Make directories, make parent directories as needed
<code>cp SOURCE... DIRECTORY</code>	Copy files and directories
<code>cp -r SOURCE... DIRECTORY</code>	Copy files and directories, copy directories recursively
<code>mv SOURCE... DIRECTORY</code>	Move (rename) files
<code>man COMMAND</code>	An interface to the on-line reference manuals



# Basic Linux commands (2)

Command	Description
<code>ssh -X foo@host.com</code>	OpenSSH SSH client (remote login program), access to host.com with user foo
<code>scp foo@host.com:/home/bar ./</code>	Secure copy (remote file copy program), copy file bar from /home on host.com to directory
<code>scp bar foo@host.com:/home/</code>	Secure copy (remote file copy program), copy file bar from the local host to /home on host.com
<code>git clone git@github.com:whatever folder-name</code>	The stupid content tracker, Clone a repository (whatever) into a new directory (folder-name).
<code>git checkout</code>	Checkout a branch or paths to the working tree.

# Other clusters – Step-by-Step

- First login, change password and get lecture notes (MS-Windows: Putty, Linux/MacOS: Terminal)

```
> ssh -X USERNAME@midway1.rcc.uchicago.edu
> passwd #Change password for USERNAME.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
Password changed
> git clone ***lecture-folder*** #clone lecture
> cd ***lecture_folder*** #go into folder
> ls # list content of folder
```

# Step-by-Step (2)

## → Perform some basic operations on the cluster

```
> ssh -X USERNAME@hpc.alphacruncher.net
> pwd
/home/USERNAME
> mkdir -p firstFolder/secondFolder
> ls
FirstFolder
> ls firstFolder
secondFolder
> cd firstFolder
> pwd
/home/USERNAME/firstFolder
> ls
secondFolder
> exit
```

# Step-by-Step (3)

- How to copy folders and files to your PC?
- MS-Windows, start WinSCP
  - Host-Name: `midway1.rcc.uchicago.edu`
  - User: `USERNAME`
- Linux/MacOS, replace `/YOUR-LOCAL-PATH/`
  - with `/home/LOCAL-LOGIN-NAME/` for linux
  - with `/Users/LOCAL-LOGIN-NAME/` for MacOS

```
>scp -r USERNAME@midway1.rcc.uchicago.edu:/home/simonsch/YOUR-LOCAL-PATH/ .
```

# Step-by-Step (4)

- Copy folders and files from your notebook  
create a file named firstFile in firstFolder
  - MS-Windows: use WinSCP to copy the directory back
  - Linux/MacOS

```
>scp -r /YOUR-LOCAL_PATH/firstFolder/FILENAME USERNAME@midway1.rcc.uchicago.edu:
```

- Check that file is there by

```
>ssh -X midway1.rcc.uchicago.edu  
> ls  
FILENAME  
>cat FILENAME #shows content of file
```

# Environment setup

Supporting diverse user community requires supporting diverse tool sets (different vendors, versions of compilers, debuggers, libraries, apps, etc)

User environments are customized via modules system (or softenv)

```
> module avail #shows list of available modules  
> module list  #shows list of modules loaded by user  
> module load module_name #load a module e.g. compiler  
> module unload module_name #unload a module
```

# Example – environment setup

```
> vi ~/.bashrc    #here you can setup/store your profile  
module load openmpi  #always load this lib upon login
```

# Using an editor on a cluster

Compute clusters like Midway's infrastructure have a variety of simple text editors available.

→ vi, vim

```
>vi helloworld.cpp
#include <iostream>

int main()
{
    std::cout << "Hello World!" << std::endl;

    return 0;
}
```



# More low bandwidth editors

Depending on network and preference, you may want to use an editor without a graphical user interface; common options:

- vi/vim
- emacs
- nano

emacs:

Undo: C-\_

Find/create file: C-x C-f

Save file: C-x C-s

Exit Emacs: C-x C-c

Quit: C-g

Two modes – insertion and command mode  
Insertion mode begins upon an insertion  
[ESC] returns to command mode

Command mode options:

:w save

:wq save and exit

:q exit as long as there are no changes

:q! exit without saving

Insertion:

i (insert before cursor)

a (append)

Deletion: x

Motion: h (left) k (up)

j (down) l (right)

# Compiling & running code interactively

→ go to **OSE2019/training/sample\_codes** → **cd OSE2019/training/sample\_codes**

If your program is only in one file (a hello-world program, or any simple code that doesn't require external libraries), the compilation is straightforward:

```
> gfortran helloworld.f90 -o helloworld.exe #Fortran
```

```
> g++ helloworld.cpp -o helloworld.exe #C++
```

Once you produced the executable, you can run it (serial code) by

```
> ./helloworld.exe
```

```
> hello
```

Example: ...

*\*later on, we will also will link in libraries as well as use optimization flags.*

# Compiling Code with a makefile

In case your program consists of many routines (files), compiling by hand gets very cumbersome

```
> g++ -o abc abc.cpp a.cpp b.cpp c.cpp
```

→ **A makefile is just a set of rules to determine which pieces of a large program need to be recompiled, and issues commands to recompile them**

→ For large programs, it's usually convenient to keep each program unit in a separate file. Keeping all program units in a single file is impractical because a change to a single subroutine requires recompilation of the entire program, which can be time consuming.

→ When changes are made to some of the source files, only the updated files need to be recompiled, although all relevant files must be linked to create the new executable.

# Compiling Code with a makefile (2)

Basic makefile structure: a list of rules with the following format:

**target ... : prerequisites ...**  
**<TAB> construction-commands**

A “**target**” is usually the name of a file that is generated by the program (e.g, executable or object files). It can also be the name of an action to carry out, like “clean”.

A “prerequisite” is a file that is used as input to create the target.

```
# makefile : makes the ABC program

abc : a.o b.o c.o ### by typing „make“, the makefile generates an executable denotes as „abc“

g++ -o abc a.o b.o c.o

a.o : a.cpp

    g++ -c a.cpp

b.o : b.cpp

    g++ -c b.cpp

c.o : c.cpp

    g++ -c c.cpp

clean : ### by typing „make clean“, the executable, the *.mod as well as the *.o files are deleted

    rm *.mod *.o abc
```

# Compiling Code with a makefile (3)

- By default, the first target listed in the file (the executable `abc`) is the one that will be created when the `make` command is issued.
- Since `abc` depends on the files `a.o`, `b.o` and `c.o`, all of the `.o` files must exist and be up-to-date. `make` will take care of checking for them and recreating them if necessary. Let's give it a try!
- Makefiles can include **comments** delimited by hash marks (`#`).
- A **backslash** (`\`) can be used at the end of the line to continue a command to the **next physical** line.
- The `make` utility **compares** the modification time of the target file with the modification times of the prerequisite files.
- Any prerequisite file that has a more recent modification time than its target file forces the target file to be recreated.

→ A lot more can be done with makefiles (beyond the scope of this lecture)

# Slurm Workload Manager

<http://slurm.schedmd.com/>

Simple Linux Utility for Resource Management (SLURM).

Open-source workload manager designed for Linux clusters of all sizes.

Provides three key functions:

- 1) It **allocates** exclusive and/or non-exclusive access to resources (computer nodes) to users for some duration of time so they can perform work.
- 2) It provides a **framework for starting, executing, and monitoring work** (typically a parallel job) on a set of allocated nodes.
- 3) It arbitrates contention for resources by managing a queue of pending work.

```
> sbatch submit_helloworld.sh (submit job)
> squeue -u NAME (status of job)
> scancel JOBID (cancel job)
```

# Run an executable on MIDWAY

<https://rcc.uchicago.edu/docs/running-jobs/index.html#running-jobs>

```
#!/bin/bash
```

```
# a sample job submission script to submit an MPI job to the sandyb partition on Midway1
```

```
# set the job name to hello-world
```

```
#SBATCH --job-name=hello-world
```

```
# send output to hello-world.out
```

```
#SBATCH --output=hello-world.out
```

```
# receive an email when job starts, ends, and fails
```

```
#SBATCH --mail-type=BEGIN,END,DAIL
```

```
# this job requests 1 core. Cores can be selected from various nodes.
```

```
#SBATCH --ntasks=1
```

```
# there are many partitions on Midway1 and it is important to specify which
```

```
# partition you want to run your job on. Not having the following option, the
```

```
# sandyb partition on Midway1 will be selected as the default partition
```

```
#SBATCH --partition=sandyb
```

```
# Run the process with mpirun. Notice -n is not required. mpirun will
```

```
# automatically figure out how many processes to run from the slurm options
```

```
./helloworld.exe
```

→ Try NOW on MIDWAY

> **cd /OSE2019/training/sample\_codes**

> **make -f makefile\_cpp**

> **qsub submit\_midway.sh**

→What is the output? Play with it a bit.

# Nodes available on Midway

<https://rc.uchicago.edu/docs/using-midway/index.html#using-midway>

## Types of Compute Nodes

The Midway compute cluster is made up of compute nodes with a variety architectures and configurations. A **partition** is a collection of compute nodes that all have the same, or similar, architecture and configuration. Currently, Midway has the following partitions:

Cluster	Partition	Compute cores (CPUs)	Memory	Other configuration details
midway	broadwl	28 x Intel E5-2680v4 2.4GHz	64 GB	EDR and FDR Infiniband interconnect
	broadwl-ic	28 x Intel E5-2680v4 @ 2.4 GHz	64 GB	10G Ethernet interconnect
	bigmem2	28 x Intel E5-2680v4 @ 2.4 GHz	512 GB	FDR Infiniband interconnect
	gpu2	28 x Intel E5-2680v4 @ 2.4 GHz	64 GB	4 x Nvidia K80 GPU

You can also retrieve a summary of the partitions on Midway using the **sinfo** command:

```
$ rcchelp sinfo shared
```

In the **rcchelp sinfo shared** summary, the "NODES" column gives the total number of nodes in each partition. This summary also lists partitions that are reserved for use by certain labs.



# Clone a git repository

```
>ssh -X USERNAME@midway1.rcc.uchicago.edu
```

```
> git clone .... # clone the git repository
```

```
> cd .. # go into the repository
```

```
> ls ... # check that all is there
```