

# Introduction to HPC and SLURM

MONASH DATA FLUENCY

## **Instructors :**

- Simon Michnowicz
- Jay van Schyndel

## **Helpers:**

- Trung Nguyen
- Luhan Cheng
- Naveen Kaushik
- Lachlan O'Neill
- Tyrone Chen
- Tarun Bonu



## Course Objectives

- Understand basic parallel computing concepts and workflows
- Understand the high-level architecture of a supercomputer
- Introductory Unix
- Use a basic workflow to submit a job and monitor it
- Understand a resource request and know what to expect



# Course Materials

## **ETHERPAD**

<https://biotraining.erc.monash.edu/etherpad/p/introtohpc1807>

## **COURSE MATERIAL**

<https://gintan.github.io/intro-to-hpc/>



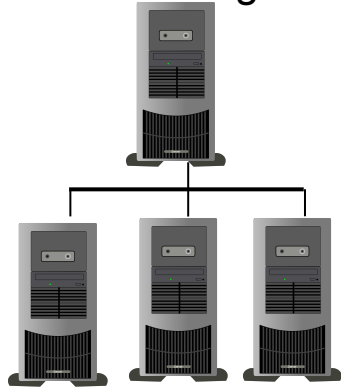
# CHAPTER 1: WHY USE A CLUSTER?



# HPC CLUSTERS



Slurm-login



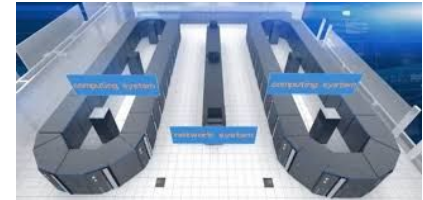
slurm0 slurm1 slurm2



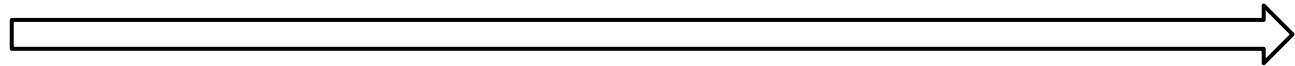
Pawsey  
(Perth)



NCI  
(Canberra)



Sunway  
(China)

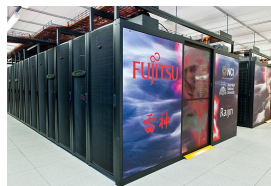


*Our cluster for today!*

And getting bigger all the time

# e-Research Workflows

- **Collect**
  - Massive amounts of data are generated by modern instruments
- **Stage**
  - Data has to be stored
- **Process**
  - Single Large Job?
  - Large parallel jobs?
  - Multiple small jobs?
- **Visualise**
  - Examine values, generating images, interact
- **Archive**
  - Long term storage, sharing



# You need a cluster when your research needs..

- Speed**

- more CPU cores, often with higher performance specs, i.e. memory, disk speed, network speeds

- Volume**

- Terabyte and Petabyte disk storage ( Spinning Disk, Solid State Disk, tape)

- Efficiency**

- Many HPC systems operate a pool of resources, running most of the time

- Cost**

- Often free to researchers due to competitive grants (i.e. NCMAS).

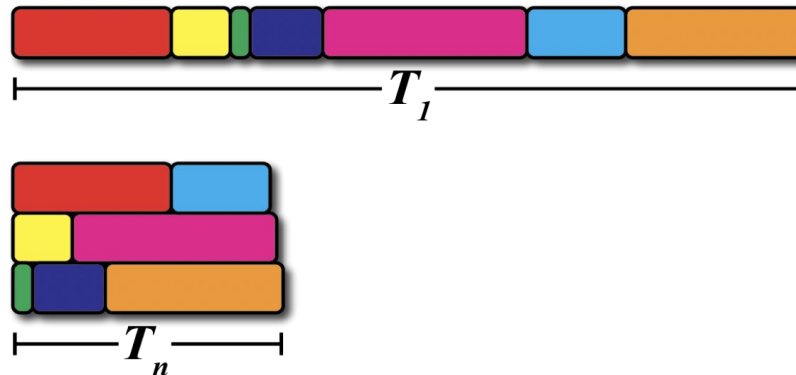
- Convenience**

- Professionally managed with advanced user support

## Parallelism in Workflows

Exploiting parallelism in a workflow allows us to

- get results faster, and
- break up big problems into manageable sizes.
- A modern supercomputer is not a fast processor. It is many processors working together in parallel







## Workflow Example – Cake Baking

It has a sequence of tasks that follow a recipe. Just like a computer program!

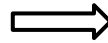
- Some tasks are independent
- Can be done in any order.
- Can be done in parallel.
- Some tasks have prerequisites.
  - Must be done sequentially.

## Baking a Cake - Staging

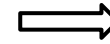
- Staging” the ingredients improves access time.
- Ingredients are “prerequisites”.
- Each ingredient does not depend on others, so can be moved in parallel, or at different times.



Supermarket



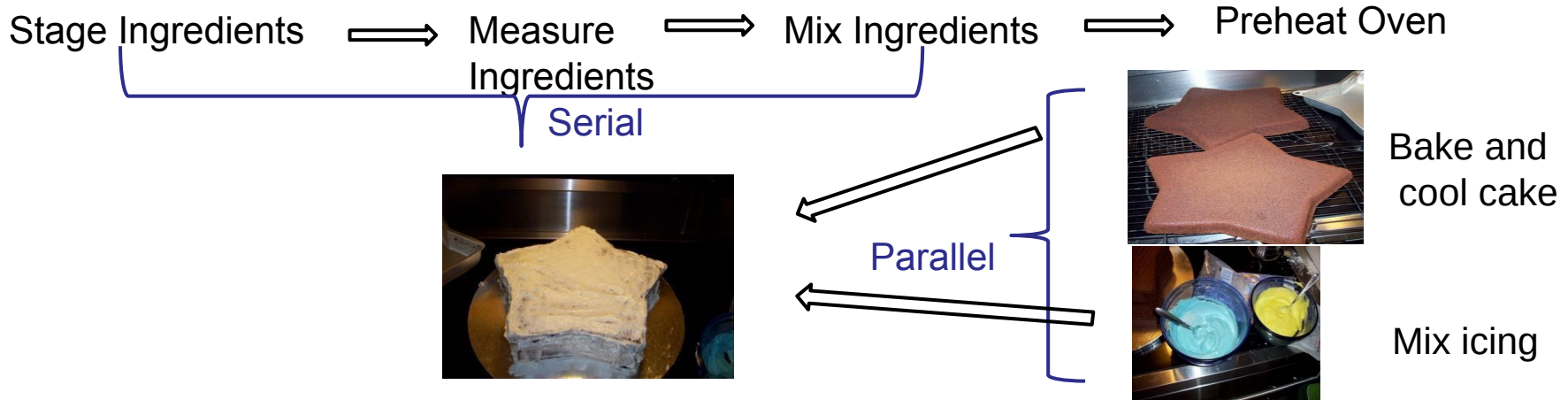
Pantry



Bench



## Baking a Cake - Process



## Levels of Parallelism

### **Coarse-grained parallelism (high level)**

- Different people baking cakes in their own kitchens.
- Preheating oven while mixing ingredients.
- Greater autonomy, can scale to large problems and many helpers.

### **Fine-grained parallelism (low level)**

- Spooning mixture into cupcake tray.
- Measuring ingredients.
- Higher coordination requirement. Difficult to get many people to help on a single cupcake tray



## How many helpers?

### **What is *your* goal?**

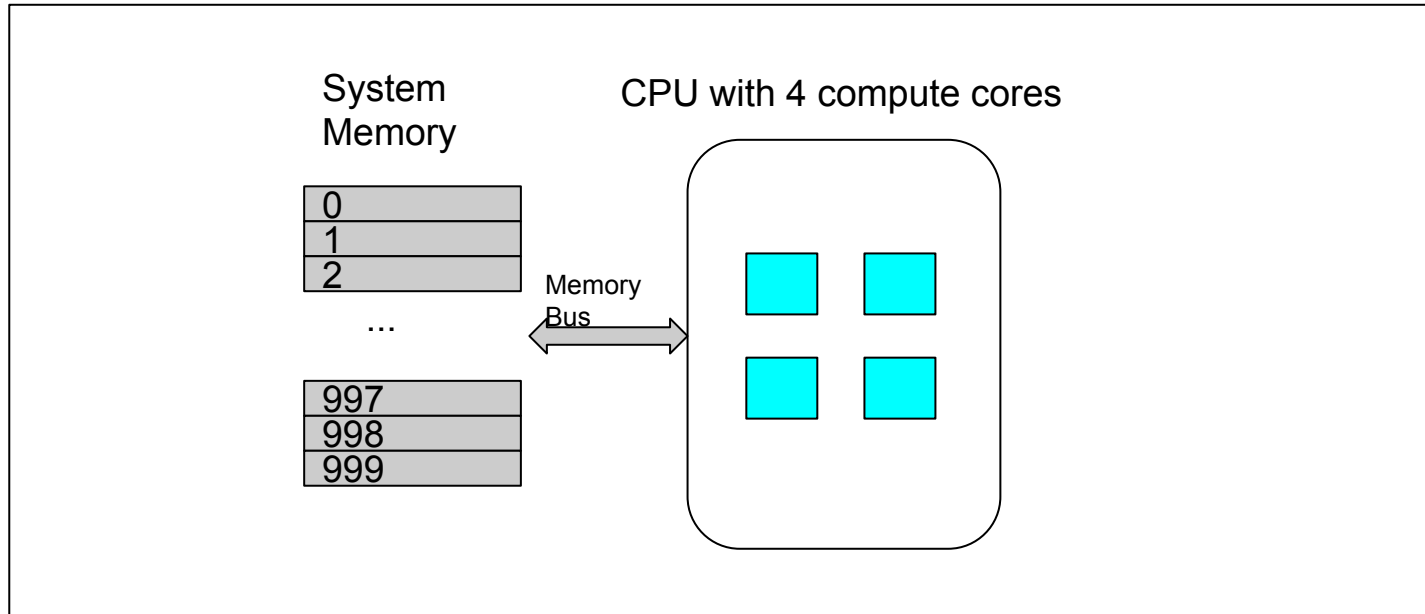
– high throughput, to do one job fast, or solve a grand-challenge problem?

### **High Throughput:**

- For many cakes, get many people to bake independently in their own kitchens – minimal coordination.
- Turn it into a production line. Use specialists and teams in some parts.
- Doing one job fast:
- Experience as well as trial and error will find the optimal number of helpers.

# A simple example

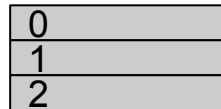
Add 1 to each element of an array of size 1000. We have previously put values 0..999 into the array



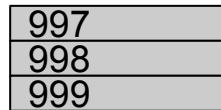
# Serial example

We have to process each element one at a time

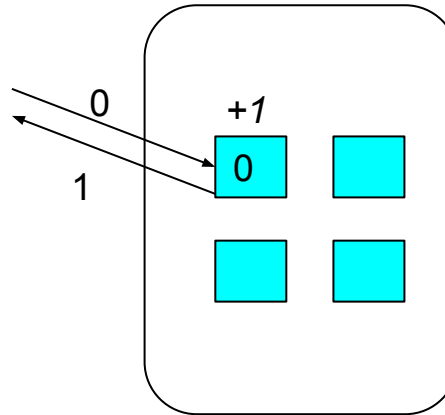
System  
Memory



...



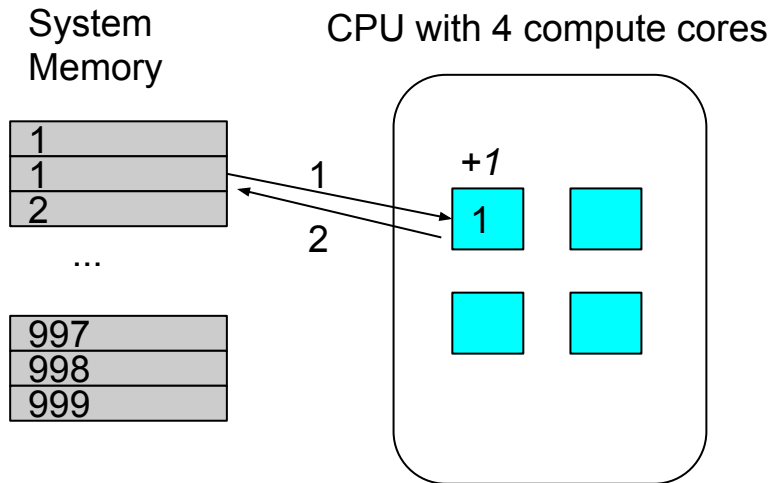
CPU with 4 compute cores



- Move first value to cpu
- Increment value
- Write result back to memory

# Serial example

Afterwards...Then repeat for the 999 other values





# Parallel example

Start using all the available cores

System  
Memory

0
1
2
3

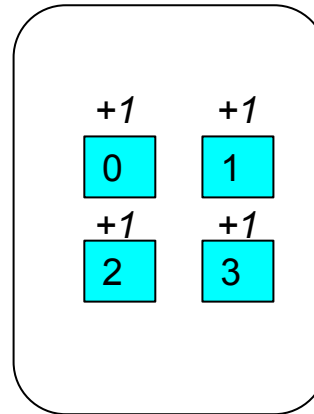
...

997
998
999

0,1,2,3

1,2,3,4

CPU with 4 compute cores



- Move 4 values to cpu
- Increment values
- Write results back to memory



## Parallel example

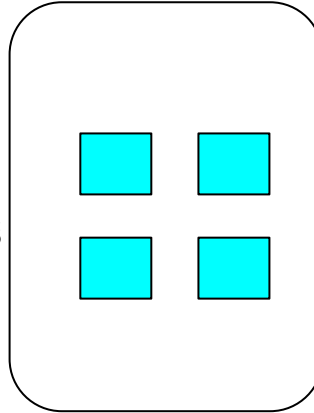
We should be getting a ~4x speed improvement..

System  
Memory

1
2
3
4
...
997
998
999

Processing  
← Next 4 values

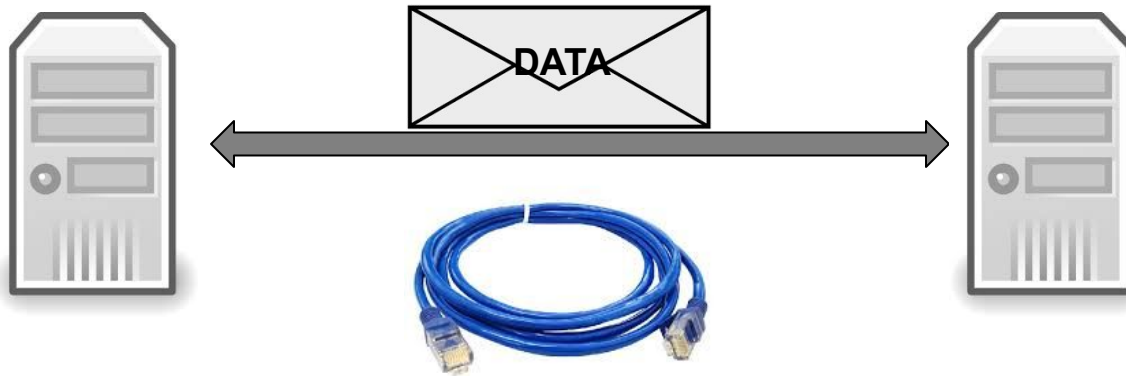
CPU with 4 compute cores



- Move values to cpu
- Increment values
- Write results back to memory

## Parallel example -with multiple computers

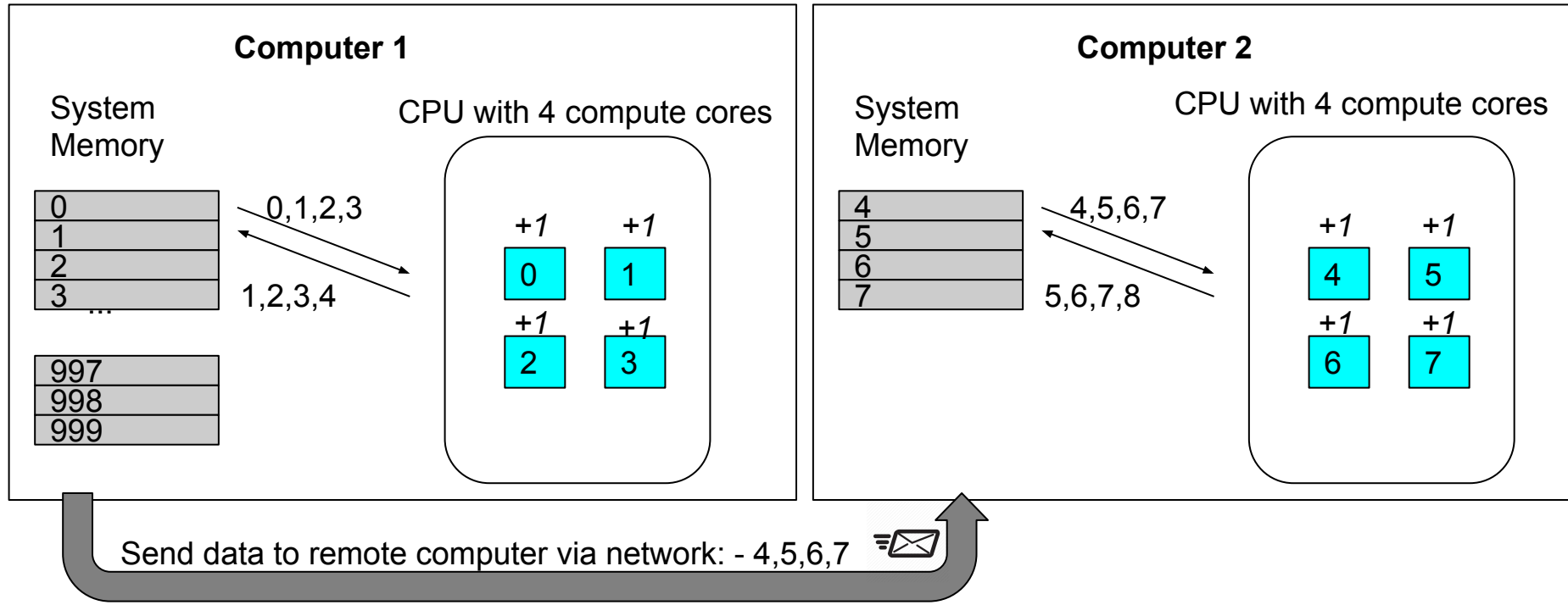
If we have multiple computers, can we get them to help as well?  
Yes, but we have to send data via a message (over a communications channel)





# Parallel example - multiple servers

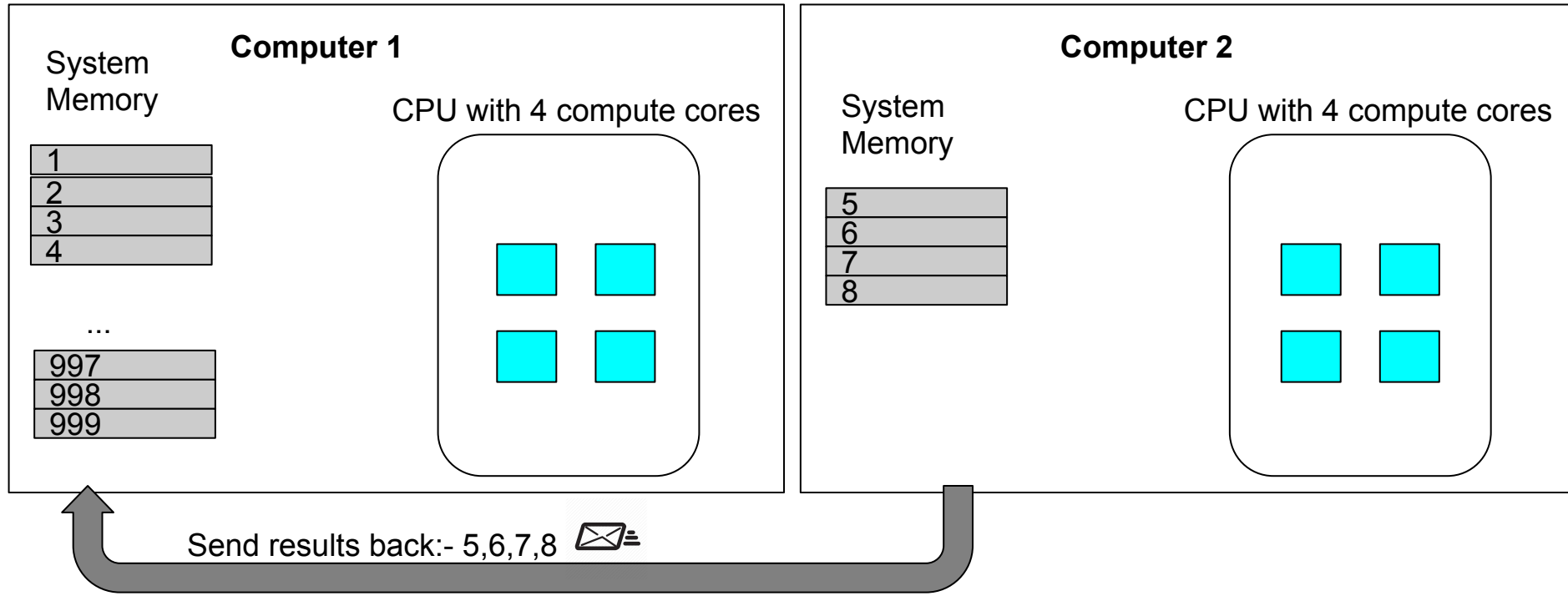
Send some of the data to another computer for processing





## Parallel example - multiple servers

The results have to be sent back. The arrival will not be synched with whatever Computer1 might be doing at the time.





# Message passing lets us scale up the number of computers we can use

But the time to send data and receive results can make a big difference to our program performance. There are many questions to ask:

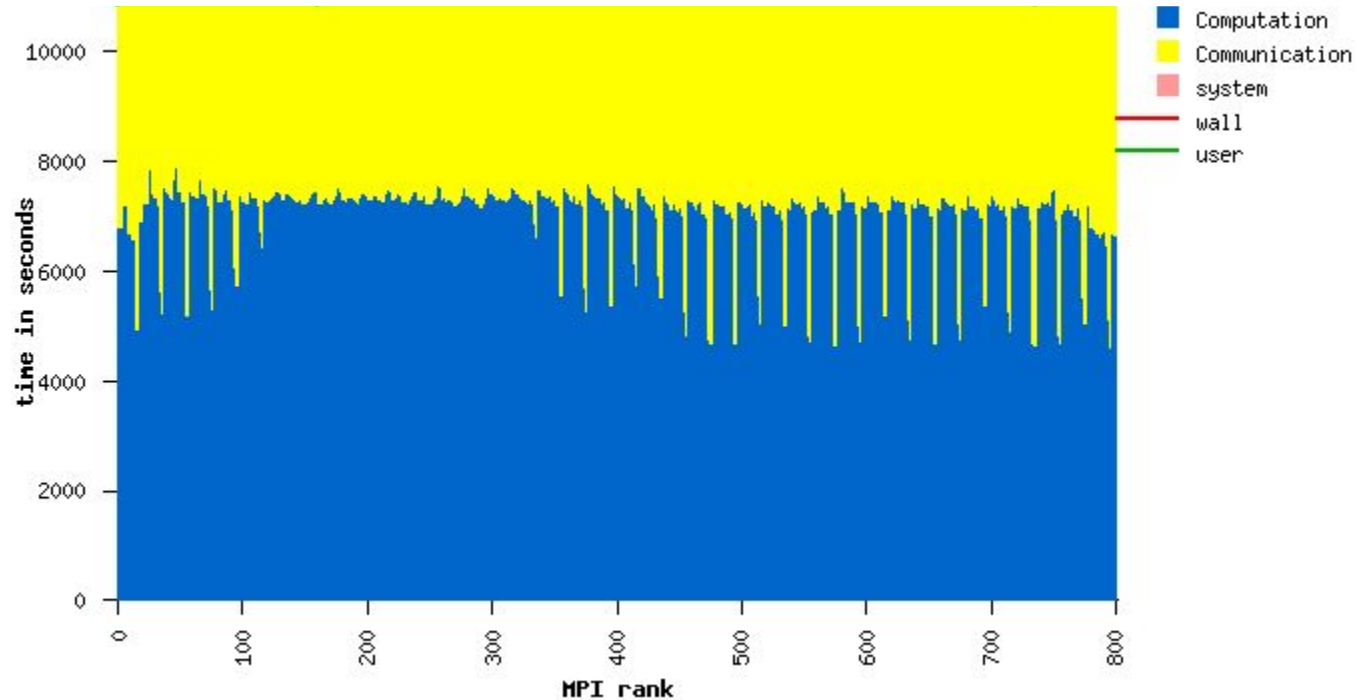
- How much data do we send in a message?
  - In our example, if we had 10 computers it might make sense to send 100 values
  - But if your program is using all the memory in the computer, then there may not be enough space for buffers to store the message
- There has to be a way for the program to synchronise all the results coming back.
- Message passing libraries such as **Message Passing Interface (MPI)** will do these tasks for you
- 

A slow link or a poor parallel algorithm can make the computers wait for data to process. This can be measured with diagnostic tools..



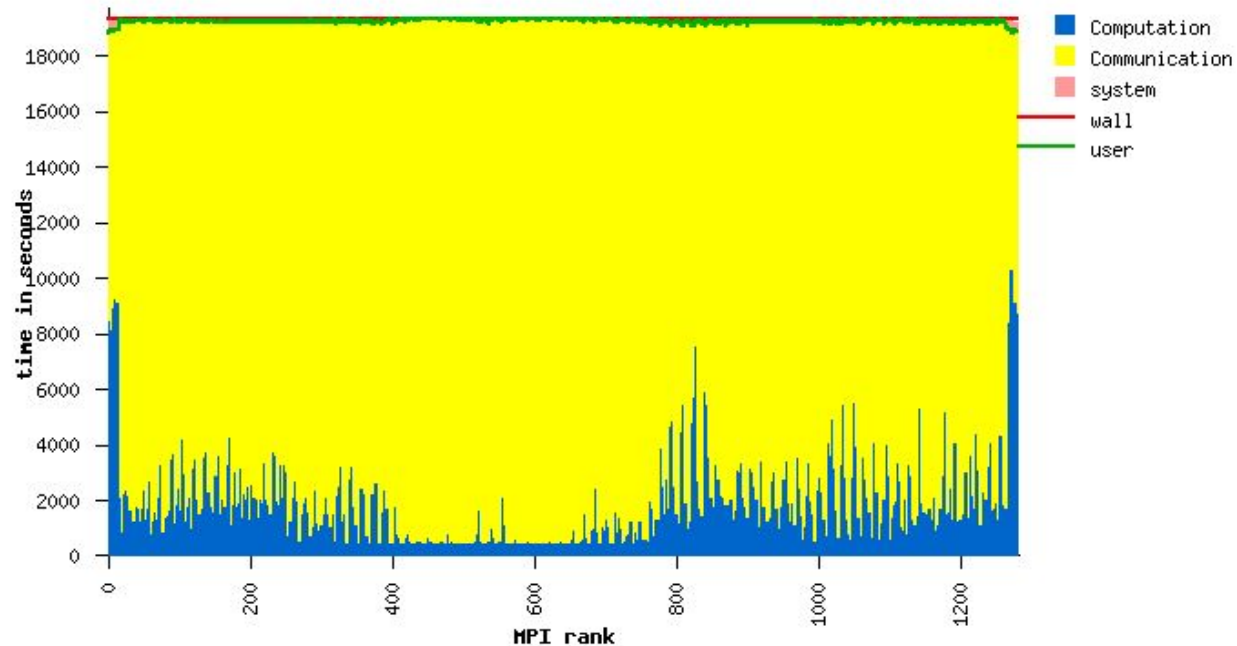
## A *good* parallel program.

- Lots of time spend in computation
- Not that much time spend in communication



## A *poor* parallel program.

- Not much time spent in computation
- Too much time spent in communication





## When to use supercomputing

Workflows need supercomputing when the resources of a single laptop or workstation are not sufficient:

Workflows need supercomputing when the resources of a single laptop or workstation are not sufficient when:

- The program takes too long to process
- There is not sufficient memory for the program
- The dataset is too large to fit on the computer

**If you are unsure whether moving to a supercomputer will help please email:**

**[mcc-help@monash.edu](mailto:mcc-help@monash.edu)**

# What to expect in a HPC System

## Login Node(s)

Users login into a computer and use it to prepare data and processing jobs

- e.g. **slurm-login** for today's course

## Data Transfer Node(s)

Users use this to transfer large data files (so as to not interfere with those on the Login Node)

- We do not have a DTN on our cluster

## Data Storage

Most have a large parallel file system (e.g.CEPH, Lustre), often with tape archive.

- We have CEPH Storage Volume on **/mnt/nfs**

## Batch Scheduler

To ensure a fair usage of compute resources, a scheduler will run your jobs

- SLURM

## Software

HPC systems have pre-installed software to use. Don't reinvent the wheel!

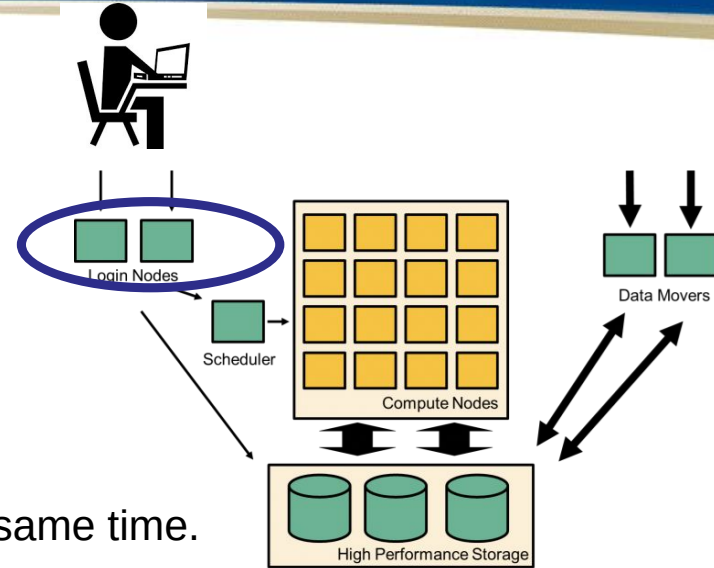
- Environment modules

## Data

Often large domain-specific data sets are stored on HPC systems

- e.g. bioinformatics databases

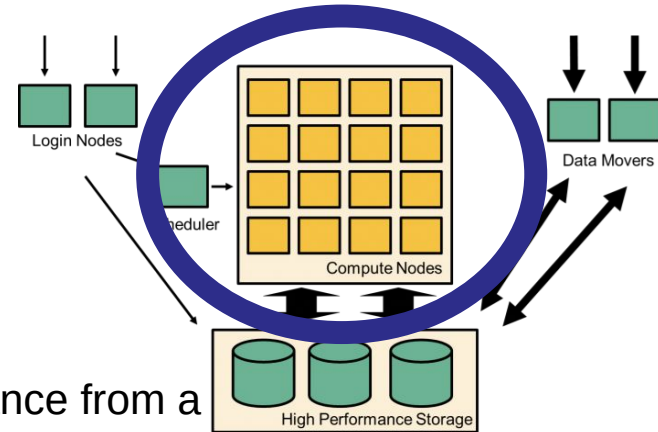
# Login Nodes



- Remote access to the supercomputer
- Where users should manage workflows
- Many people (~100) can share a login node at the same time.
- **Do not run your programs on the login nodes!**
- Use the login nodes to submit jobs to the queue to be executed on the compute nodes
- Login nodes *can* have different hardware to compute nodes.
  - Some build tests may fail if you try to compile on login nodes.

## Compute Nodes

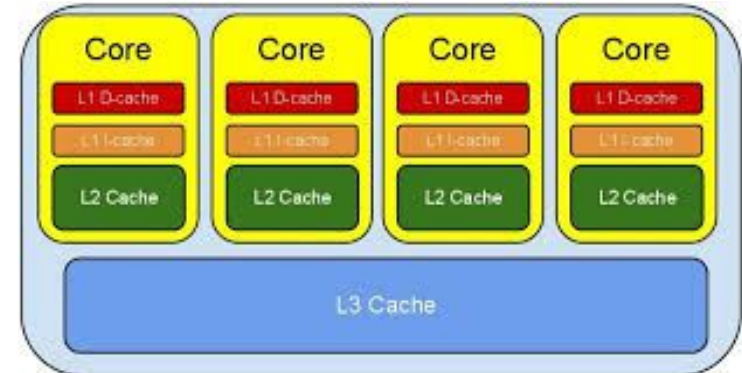
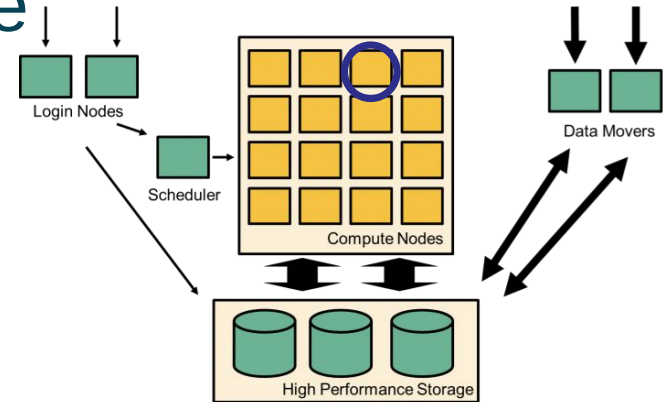
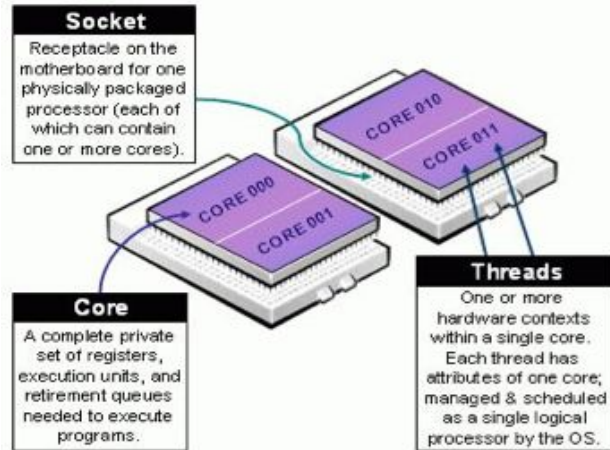
- Programs should run on the compute nodes
- Access is provided via the scheduler
- Compute nodes have a fast interconnect that allows them to communicate with each other
- Jobs can span multiple compute nodes
- Individual nodes are not that different in performance from a workstation
- Parallelism across compute nodes is how significant performance improvements are achieved.
- Nodes typically have access to the shared 'global' filesystem



## Inside a Compute Node

Each compute node has one or more CPUs:

- Each CPU has multiple cores
- Each CPU has memory attached to it
- Each node has an external network connection
- Some systems have accelerators (e.g. GPUs)



# File Systems

- HPC systems typically have many filesystems
- Each file systems typically has very different properties
  - Users will have quotas on their file systems
    - Amount of data they can have (~GB)
    - The number of files they can have (inodes)
  - Some file systems are:
    - Local only to the compute node (e.g. /tmp)
    - Global across all nodes (e.g. /mnt/home)
- Some file systems are backed up to tape, some not
- Some are high-performance parallel file systems, others not



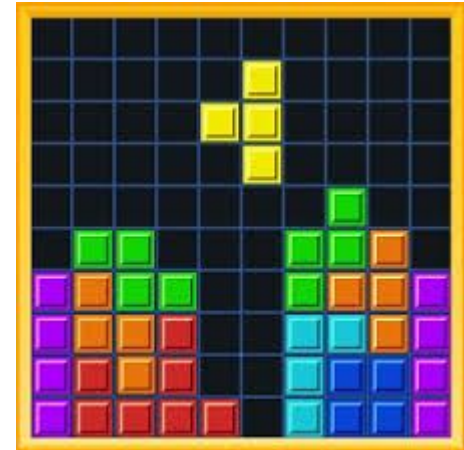
*Please consult the documentation when using a HPC system*



# Schedulers

All HPC systems have a queuing system that examines your job requirements and finds available hardware, where it is placed in a queue until it is scheduled to run on a compute node.

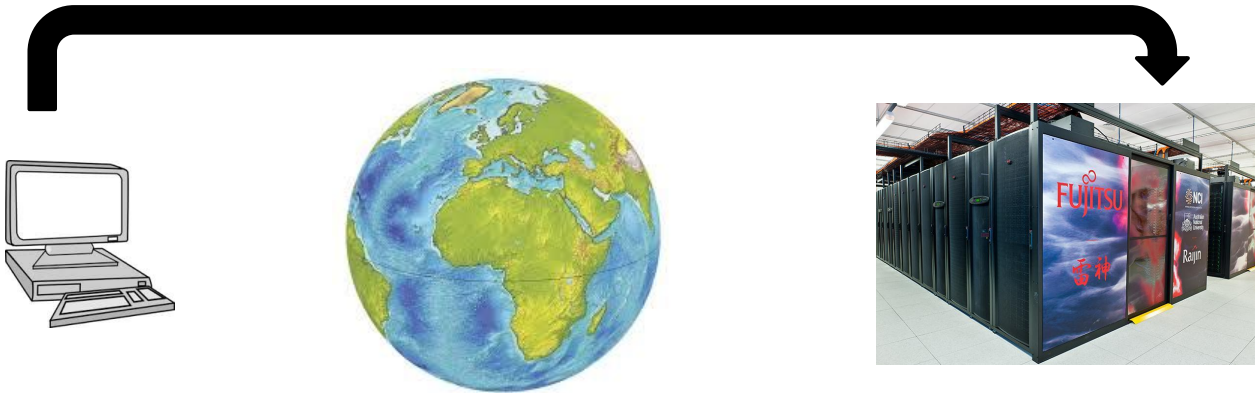
More will be covered later, but behind the scenes, schedulers are often considered to be like a game of Tetris...







## CHAPTER 2: Connecting to the cluster







# Secure Shell (ssh)

Logging into servers requires a **ssh** client on your local computer

Common terminal programs:

- **Windows**

- use putty (download <https://www.putty.org/>)
- Or MobaXterm (download <https://mobaxterm.mobatek.net/>)
- Or Cygwin if you are an advanced user


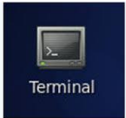

- **Linux**, use xterm (preinstalled)

- **OS X**, use Terminal (preinstalled) or iTerm2 (download)



# Secure Shell (ssh)

Logging into servers requires a **ssh** client on your local computer

OS		TOOL	Command
Mac		ssh is already in your terminal window	ssh username@43.240.99.84
Unix		ssh is already in your terminal window	ssh username@43.240.99.84
Windows		If you have a Cygwin terminal	ssh username@43.240.99.84



*Example.* **ssh simon@43.240.99.84**

The authenticity of host 'localhost (127.0.0.1)' can't be established.

ECDSA key fingerprint is SHA256:CCVXbbNqjWEv9bvtcnzNT3O2n3ii9Y5rhg0GvqOXXiM.

ECDSA key fingerprint is MD5:4b:84:40:45:bd:05:27:cf:c3:33:99:58:96:13:d2:d0.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.

Password:

Last login: Wed Aug 1 05:43:30 2018

*The very first time you log in, you will get a ssh-key exchange message, and you will be asked to accept the key. Hit. 'Yes'. This won't happen again.*

*You are prompted for password. You will **not** see the password when you type. This is a security measure*

---

Nectar CentOS 7 (Core)

Image details and information is available at

<https://support.ehelp.edu.au/support/solutions/articles/6000106269-image-catalog>

---

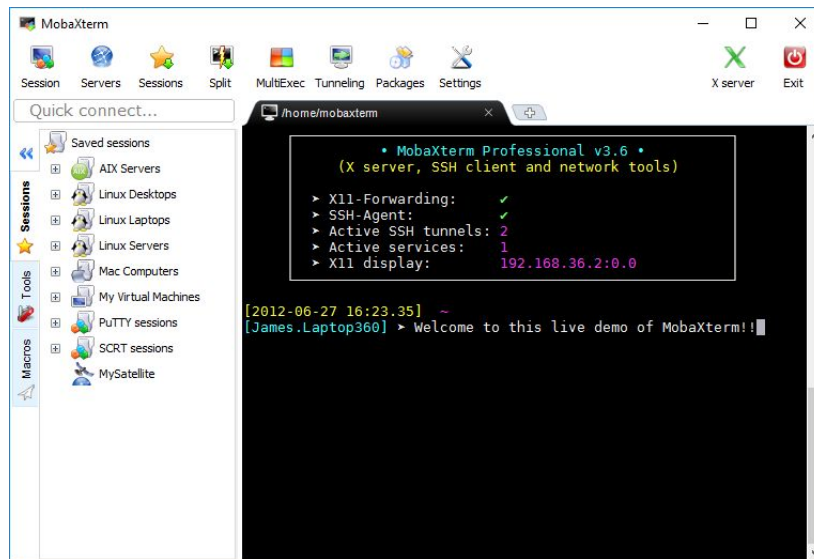
[simon@slurm-login ~]\$

# Using MobaXterm (Windows Only)

MobaXterm is a Windows GUI often found on computers. Besides a ssh terminal, it has a X-client so you can run graphics programs on the Unix Server

If you are on a windows box, and do not have MobaXterm, please download it now if you want to use it.

<https://mobaxterm.mobatek.net/>





# Using PUTTY (Windows Only)

PUTTY is a Windows GUI often found on computers.

If you are on a windows box, and do not have putty, please download it now if you want to use it.

<http://www.putty.org/>



Advanced Users:

If you want a Unix experience on windows, then install Cygwin

<https://www.cygwin.com/>

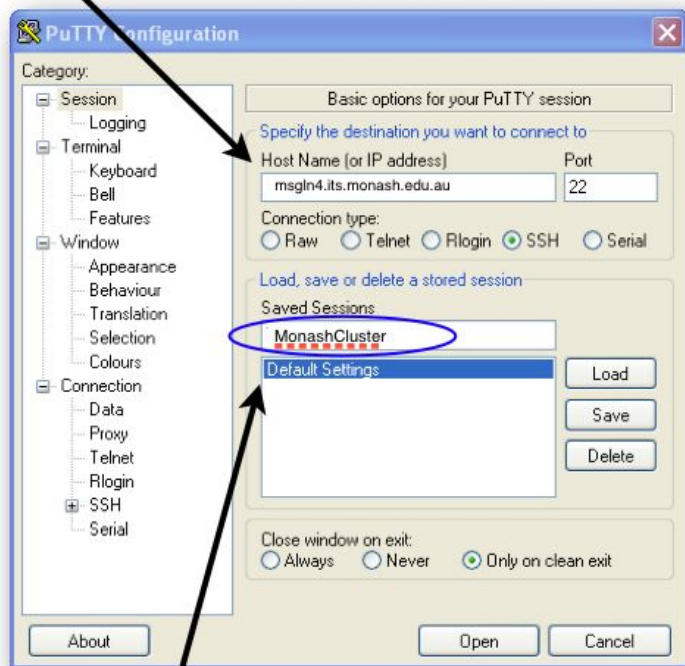
(We will use putty today as it is a lot easier to install and configure)



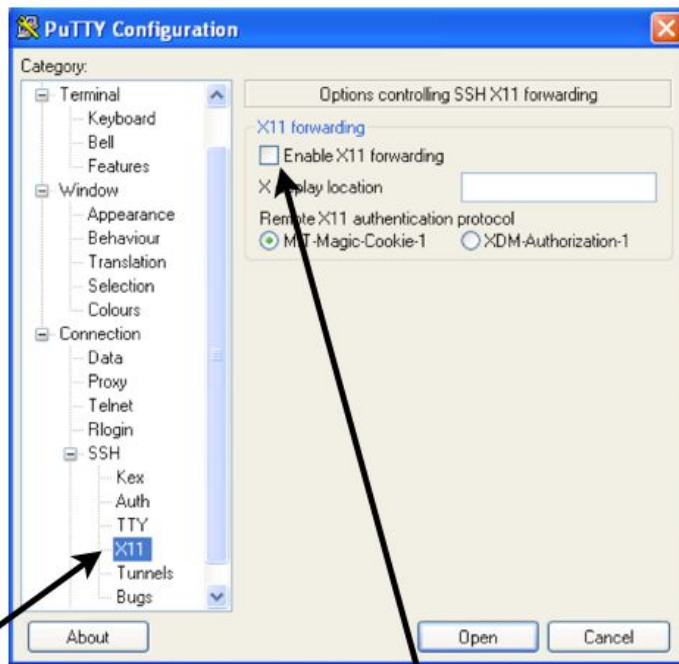
# Using PUTTY (Windows Only)

PUTTY is a Windows GUI often found on computers. For the course set 'Host Name' to **43.240.99.84**

step 1: enter hostname



step 2: type descriptive name



step 3: expand the SSH and click X11

step 4: check this box to enable X11 forwarding

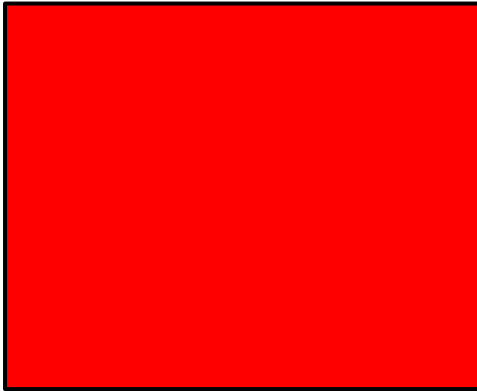


## Account Security

- SSH uses fingerprints to identify computers ... so you don't give your password to someone pretending to be the remote host
- Do not share your account with others, this violates the conditions of use (have the project leader add them to the project)
- Please do **not** provide your password in help desk tickets, nobody ever asks for your password via email as it is not secure



For all exercises use the sticky notes.



Need help



All Good



## First Challenge

1. Open the course material in a web page
2. Login to our slurm login node **43.240.99.84**
  1. You will needs the username and password provided to you
3. Then let us know when you are finished...

Need help



All Good





## Chapter 3: Scripts, variables





## Unix Scripts

Shell commands can be placed in text files and executed. e.g. **file.sh**

-the files should have execute permission on them,

i.e. **chmod gou+x file.sh**

-the files are plain text files (and are not compiled)

The first line of the file is used to specify the shell being used.

***#!/path/to/shell***

e.g.

***#!/usr/bin/bash***

Or

***#!/bin/bash***

This will be machine dependent –where did they put the bash executable?

'#' normally indicates a comment line.

Then put whatever commands you want to execute.

Then you can run the command as if it were an executable.

***./file.sh***

OR run it with the correct shell command

***sh file.sh***



## Unix Scripts

```
$ nano mycommands.sh  
$ chmod gou+x  
mycommands.sh  
$ ./mycommands.sh
```

mycommands.sh

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

What do you expect to see?



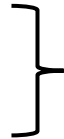
## Editors - revision.

UNIX has several in-built editors to create text files.

- **vi** - screen oriented text editor. As old as Unix..
- **emacs** – also created decades ago. Part of GNU project.
- **nano** – simple micro editor
- Plus many more....

*Nano is the easiest editor to use in a beginner's course...*

nano myfile.txt  
vi myfile.txt  
emacs myfile.txt



*Either command creates a new file 'myfile.txt' if it does not exist, or edits an existing file.*

# NANO





## Shell Variables

**X=1**



# VARIABLES

Shells are programs like any other, and like all programs, they have variables that control their behaviour. You can view the shell variables you have by typing

**env**

or

**printenv**

```
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="3" [1]="2" [2]="25" [3]="1" [4]="release" [5]="x86_64-redhat-linux-gnu")
BASH_VERSION='3.2.25(1)-release'
COLORS=/etc/DIR_COLORS
COLUMNS=172
CONDOR_CONFIG=/opt/vdt/condor/etc/condor_config
CVS_RSH=ssh
...
...
etc
```





**COLUMNS=172**

Convention  
makes variable  
names upper  
case

Variable values are always  
strings, and have to be  
converted to other types if  
necessary.

**MANPATH=:/usr/share/man:/opt/n1ge62/man**

When variables contains lists of values, the  
*convention* uses the : character as the delimiter



**The PATH variable controls where the shell looks for executables.**

*PATH=/usr/lib64/qt-3.3/bin:/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/opt/n1ge62/bin/lx24-amd64:/opt/dell/srvadmin/bin:/nfs/home/hpcmerc/vlad/bin:/usr/bin:/usr/ucb:/etc:/opt/vdt/condor/bin:/opt/vdt/jdk1.5/bin:.*

**./myprog.exe**      ← This runs myprog.exe in the current directory

**/bin/myprog.exe**      ← This runs myprog.exe found in /bin

**myprog.exe**      ← Which program is executed here?

**The shell separates the paths found in the PATH variable and searches each one in order for the program to execute.**



*/usr/lib64/qt-3.3/bin*  
*/usr/kerberos/bin*  
*/usr/local/bin:/bin*  
*/usr/bin*  
*/opt/n1ge62/bin/lx24-amd64*  
*/opt/dell/srvadmin/bin*  
*/nfs/home/hpcmerc/vlad/bin*  
*/usr/bin*  
*/usr/ucb*  
*/etc*  
*/opt/vdt/condor/bin*  
*/opt/vdt/jdk1.5/bin*  
*.*

Order of  
search

Current directory '.' last to be searched,  
but only because of its position in PATH.  
Some systems do not put '.' in the PATH  
at all. Can you think why?



## Printing Variables

The 'echo' command can be used to print variables. A \$ is needed to discriminate between text and variables.

**echo HOME**

HOME

**echo \$HOME**

/nfs/home/hpcmerc/vlad

You can create a variable by assigning to it

**PIN=1234**

**echo \$PIN**

1234

**PIN=4321**

**echo \$PIN**

4321



To assign a variable in a shell so that it is copied into child processes that it creates, you use the **export** command.

```
PIN=1234  
export PIN  
bash  
echo $PIN  
1234
```

## (10 min) Exercise.

- Write a script file that executes the following commands
- **date**
  - Date is a unix shell command that prints the time and date
- **echo 'This script is running on:'**
  - to print the text
- **hostname**
  - Hostname is a unix shell command that prints the name of the computer you are on
- **sleep 30**
  - **sleep N** sleeps for N seconds
- **echo 'Completed by' \$USER**
  - to print the text follow by an environment variable for username

Need help



All Good



Execute your script from the shell command line and examine the output

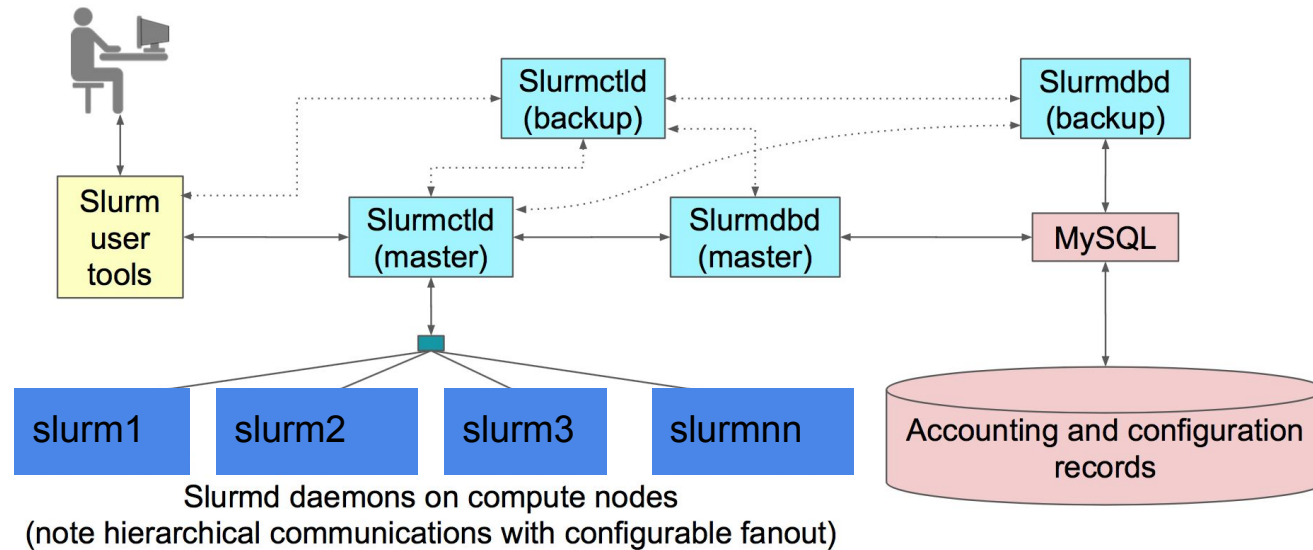


At the end of this exercise, you should be able to put several shell commands into a file and run them



## CHAPTER 4 : Working on a cluster

SLURM as a job scheduler and resource manager.







## **SLURM user commands:**

`sbatch`

- Submit a batch script to Slurm

`salloc`

- Obtain a Slurm job allocation (a set of nodes), execute a command, and then release the allocation when the command is finished

`srun`

- run parallel jobs



Exercise (5 min):

Now that you've learned how to create a script, we will submit the script to the job scheduler.

Content of firstjob.sh:

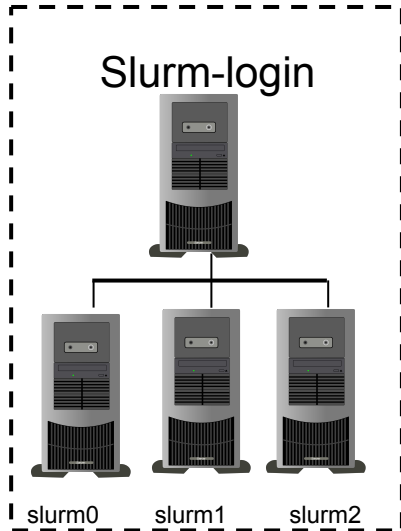
```
#!/bin/bash
date
echo 'This script is running on:'
hostname
sleep 120
echo 'Completed by' $USER
```

To submit:

```
sbatch firstjob.sh
```

Do you get a job ID?

The job ID is for record purposes and it's very useful when you are troubleshooting your job.



*Our cluster for today!*

While you are waiting for your job to run, investigate what our cluster looks like. You should be able to login into our nodes and investigate them. (Or you can ask SLURM to advise you of some info with **scontrol show node <nodename>** )

Node	No of cpus (lscpu)	Total Amount of memory (free -m)	Main disks systems (df -k)
slurm-login			
slurm0			
slurm1			
slurm2			

Need help



All Good





Did you find?

Node	No of cpus (lscpu)	Total Amount of memory (free -m)	Main disks systems
slurm-login	2	5806	/mnt/nfs, /mnt
slurm0	8	32012	/mnt/nfs, /mnt
slurm1	4	11854	/mnt/nfs, /mnt
slurm2	4	11854	/mnt/nfs, /mnt



## Useful commands to check your job on the scheduler:

`scontrol show job {jobID}`

- When you obtain your job ID, you can check the information about your job

`squeue`

- view information about jobs located in the Slurm scheduling queue

`squeue -u {username}`

- view information about jobs located in the queue for one or more users

`scancel {jobID}`

- Used to signal jobs or job steps that are under the control of Slurm



`squeue -u {username}`

```
[user1@slurm-login ~]$ squeue -u user1
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	ODELIST(Reason)
47	all	firstjob	user1	R	0:03	1	slurm1

`squeue`

```
[user1@slurm-login ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	ODELIST(Reason)
48	all	firstjob	user1	R	0:01	1	slurm1



```
[user1@slurm-login ~]$ scontrol show job 47
JobId=47 JobName=firstjob.sh
  UserId=user1(1001) GroupId=user1(1001) MCS_label=N/A
  Priority=20182 Nice=0 Account=slurmclass QOS=normal
  JobState=RUNNING Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  RunTime=00:00:16 TimeLimit=02:00:00 TimeMin=N/A
  SubmitTime=2018-08-02T22:16:51 EligibleTime=2018-08-02T22:16:51
  StartTime=2018-08-02T22:16:51 EndTime=2018-08-03T00:16:51 Deadline=N/A
  PreemptTime=None SuspendTime=None SecsPreSuspend=0
  LastSchedEval=2018-08-02T22:16:51
  Partition=all AllocNode:Sid=slurm-login:7279
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=slurm1
  BatchHost=slurm1
  NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
  TRES=cpu=1,mem=4G,node=1,billing=1
  Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
  MinCPUsNode=1 MinMemoryCPU=4G MinTmpDiskNode=0
  Features=(null) DelayBoot=00:00:00
  Gres=(null) Reservation=(null)
  OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
  Command=/mnt/nfs/home/user1/firstjob.sh
  WorkDir=/mnt/nfs/home/user1
  StdErr=/mnt/nfs/home/user1/slurm-47.out
  StdIn=/dev/null
  StdOut=/mnt/nfs/home/user1/slurm-47.out
  Power=
```



More useful commands to check the information about a compute node:

`sinfo`

- To see all the partition in the queue

`sinfo -p {partition name}`

- if you just want to look at a specific partition, you can parse in -p follow by the name of the partition

`sinfo -NI`

- to see all the nodes in the job scheduler

`scontrol show node {node name}`

- to see a specific node in the queue





```
[user1@slurm-login ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
all*      up 7-00:00:00      3   idle slurm[0-2]
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$ sinfo -p all
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
all*      up 7-00:00:00      3   idle slurm[0-2]
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$ |
```



```
[user1@slurm-login ~]$ sinfo -Nl
Thu Aug  2 22:28:46 2018

```

NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK	WEIGHT	AVAIL_FE	REASON
slurm0	1	all*	idle	8	8:1:1	31000	0	1	(null)	none
slurm1	1	all*	idle	4	4:1:1	11000	0	1	(null)	none
slurm2	1	all*	idle	4	4:1:1	11000	0	1	(null)	none

```

[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$ scontrol show node slurm0
NodeName=slurm0 Arch=x86_64 CoresPerSocket=1
  CPUAlloc=0 CPUErr=0 CPUTot=8 CPULoad=0.02
  AvailableFeatures=(null)
  ActiveFeatures=(null)
  Gres=(null)
  NodeAddr=slurm0 NodeHostName=slurm0 Version=17.11
  OS=Linux 3.10.0-693.17.1.el7.x86_64 #1 SMP Thu Jan 25 20:13:58 UTC 2018
  RealMemory=31000 AllocMem=0 FreeMem=29215 Sockets=8 Boards=1
  State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
  Partitions=all
  BootTime=2018-07-26T11:51:18 SlurmdStartTime=2018-08-01T16:30:13
  CfgTRES=cpu=8,mem=31000M,billing=8
  AllocTRES=
  CapWatts=n/a
  CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
  ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s

```



## **Demonstration**

Now use the commands to check the status of a compute node

How many compute node do we have in this cluster?

How many cpu is available for each of the compute node?

How many partition do we have in this cluster?

What is the time limit for each of this partition?

Commands:

```
sinfo
```

```
sinfo -p comp
```

```
sinfo -NI
```

```
scontrol show node slurm1
```



## What is a partition?

A partition is a pool of resources, it usually consists of one or more nodes that have the same constraints.

For example:

A partition can have a maximum time limit of 7 days and the default memory per cpu is 4096MB

```
PartitionName=all Default=yes Nodes=slurm[0-2] State=Up MaxTime=7-0  
DefaultTime=02:00:00 DefMemPerCPU=4096 AllowQOS=ALL
```



## Customise your job script?

You can tell the job scheduler how much of the resources do you need to run your job?

- Time limit
- Memory
- Account
- Number of tasks
- Number of nodes

<code>--ntasks</code> or <code>-n</code>	Number of tasks and by default the number of CPUs
<code>--mem</code>	Total memory size
<code>--time</code> or <code>-t</code>	Wall time limit
<code>--input</code> or <code>-i</code>	File used for standard input (default is <code>/dev/null</code> )
<code>--output</code> or <code>-o</code>	File used for standard output
<code>--error</code> or <code>-e</code>	standard error (default is to combine with <code>stdout</code> )





An example of a complex job script:

```
#!/bin/bash
#SBATCH --job-name=desktop
# --exclusive allows the job to consume all resources on the node regardless of how many cpus/gpus/memory/etc
#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:K80:2
#SBATCH --partition=m3c
#SBATCH --odelist=m3c001
# SBATCH --reservation=Maint-09Jan18

env | grep SLURM
echo " Starting MASSIVE desktop..."
echo " Setting up the system environment..."
# This is required for tcsh desktops to work
# source /etc/profile.d/modules.sh
module purge
```



Exercise: (15 min)

Write a job script to assign 1GB to the job?

Also, add in the time limit as well, set it to 1 mins.

Submit the job and examine the job information.

[Hints] Combine the following with your previous script:

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

```
#SBATCH --mem=1024
```

```
#SBATCH --time=00:01:00
```

Need help



All Good



## Running a parallel job

srun - Run a parallel job on cluster managed by Slurm. If necessary, srun will first create a resource allocation in which to run the parallel job.

```
srun --ntasks=2 --label hostname
```

### Not sure what an option does?

The Unix Manual system can help  
On the shell window, use

**man srun**

To find out what --label does

In this example, we request SLURM to allocate two CPU and precede each standard output or error message with its rank number

The output:

```
0: slurm1  
1: slurm1
```





Running a parallel job across two nodes:

```
srun --ntasks=2 -N 2 --label hostname
```

and the output is:

```
1: slurm1  
0: slurm0
```

From srun manual page

**-N, --nodes=<minnodes[-maxnodes]>**



## Exercise: run a job on more than one core/node

To demonstrate the performance differences running a job with one core vs two cores.

```
$ mkdir mpi  
$ cd mpi  
$ cp -r /usr/local/exercises/mpi/* .
```

- examine run.sh and see what it does
- modify slurm.slm to request two CPU
- submit job with sbatch
- examine the output

### [Hints]

```
#!/bin/sh  
#SBATCH --job-name=mpitest  
#SBATCH --time=00:05:00  
#SBATCH --ntasks=2  
#SBATCH --cpus-per-task=1  
  
#SBATCH --partition=all  
  
./run.sh
```

Need help



All Good





## CHAPTER 5 : Accessing Software

# User Environment

HPC systems can support a large and complex range of software

- Operating System (Ubuntu, CentOS)
- Compilers (e.g. Intel, GCC, Cray, PGI)
- Debuggers and Profilers (e.g. MAP , DDT)
- Performance mathematical libraries (e.g. Intel's MKL, Lapack, Petsc)
- Parallel programming libraries (e.g. MPI)
- File format libraries for parallel IO (e.g. HDF5)

Project groups may be expected to manage the installation of their own software stack.

**All of the above software is not immediately available as soon as you log in.**



# Environment Modules

To prevent conflicts between software names and versions, applications and libraries are not installed in the standard directory locations.

Modules modify the environment to easily locate software, libraries, documentation, or particular versions of the software

```
module load openmpi
```



# Module Commands

Command	Description
<code>module avail</code>	Show available modules
<code>module list</code>	List loaded modules
<code>module load <i>modulename</i></code>	Load a module into the current environment
<code>module rm unload <i>modulename</i></code>	Unload a module from the environment
<code>module purge</code>	Unload all loaded modules
<code>module swap <i>module1 module2</i></code>	Swap a loaded module with another
<code>module show <i>modulename</i></code>	Give help for a particular module
<code>module help</code>	Show module specific help

## How it works...

Unix shells (Bash, C-shell, etc) use system environment variables to define important values

- By convention, these variables use upper-case letters
- The shell command **printenv** prints all the variables
- To look at a particular one, we use 'echo' and the variable name
  - i.e. to look at the **PATH** variable we type

echo \$PATH

*Note the \$ is used to tell shell that we are looking for a variable called PATH*

Another important variable is **LD\_LIBRARY\_PATH** that lists the locations of shared libraries.

Modules work by modifying these variables in a very easy to use manner, independent of the shell you are using.



**PATH** is used by the shell to search where executables might be. The shell command **which** is used to find the path of an executable.

For example, lets look for where our Python executable may be

```
which python  
/usr/bin/python
```

And we can ask Python which version it is

```
python --version  
Python 2.7.5
```

Suppose we wanted the latest version of Python ?





## module avail

So before we install the latest version, or request the Help Desk to do it, let's check the modules

```
module avail python
-----/usr/local/Modules/modulefiles -----
python/2.7.12
```

So let's use it!

```
module load python
```

Let's check we have the right version

```
which python
/usr/local/python/2.7.12/bin/python
python --version
Python 2.7.12
```

## Removing Modules

Suppose we wanted to go back to using the system python. This can be done with a number of commands

```
module rm python
```

```
module delete python
```

```
module purge #remove ALL modules, not just python
```

Let's check we have gone back to our original version

```
which python
```

```
/usr/bin/python
```

```
python --version
```

```
module list
```



## module Versions

It's possible to have lots of different versions of the same software

```
module avail python
```

```
-----/usr/local/Modules/modulefiles
```

```
python/2.7.12-gcc
```

```
python/2.7.8-gcc(default)
```

```
python/2.7-intel
```

```
python/3.4.3-gcc
```

```
python/3.5.1-gcc
```

Environment module lets you specify the version, or swap between them

**module load python** #loads the default, same as *module load python/2.7.8-gcc*

**which python**

```
/usr/local/python/2.7.8-gcc/bin/python
```

**module swap python/2.7-intel** #unloads the existing Python, then loads the version

#compiled with the intel compiler

**which python**

```
/usr/local/python/2.7-intel/bin/python
```

## Exercise 6

Build on previous work, write a script (say **interrogate.sh**) to interrogates a node for information

- Print the hostname (*hostname*)
- Print the time of day (*date*)
- Look for an OS system python
  - What is its path?
  - What version is it?
- Load Python via modules
  - What is its path?
  - What version is it?
- Print a farewell message and time

All Good

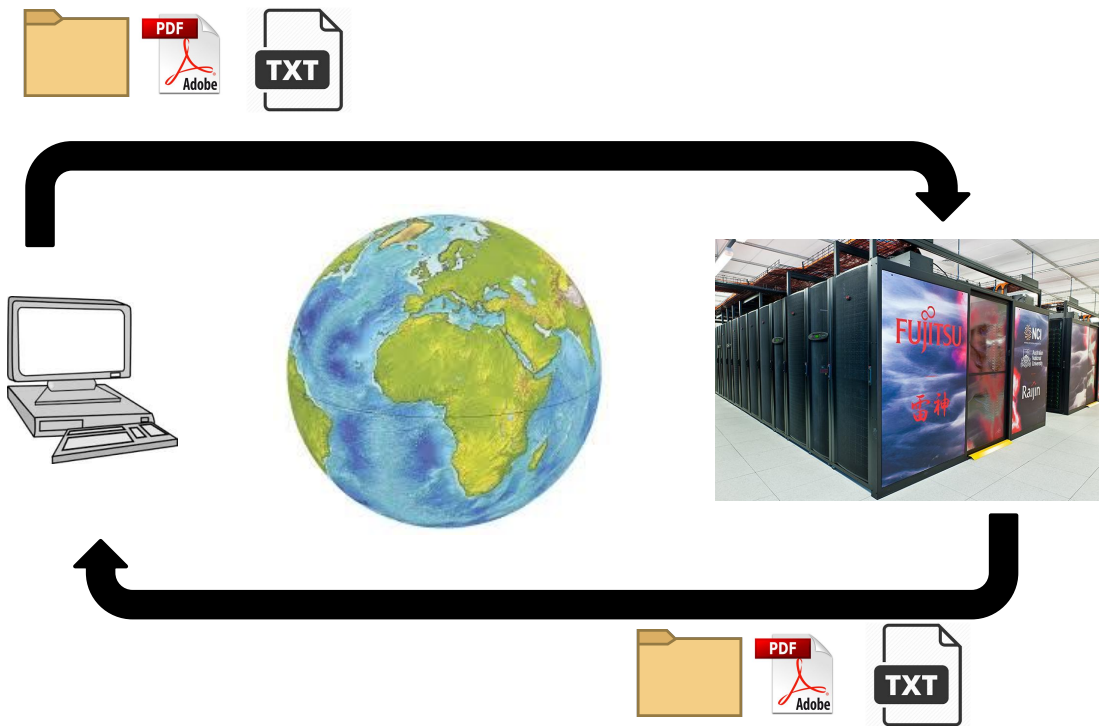


Need help



- Once you have run this on the login node, write a slurm script to run **interrogate.sh** on a single node
- If you have time, modify the slurm script to run on 2 nodes.
  - Remember to use **srun** when invoking **interrogate.sh**
  - Is the output of the text what you expect?

## CHAPTER 6 : Transferring software





## scp – Copying files between computers

You can copy files *between* computers with *scp* – ‘secure cp’  
*scp* has similar syntax to *cp*. You specify the username and machine name of the remote host in the command line

**scp <from> <to>**

where <from> or <to> can also specify a **username@computername:**

e.g.

**scp \*.db username@computer:/home/apps/dbfiles**

Note the ‘:’

The wild card \*.db matches any filename whose ending is the string ‘.db’



## scp – Copying files between computers

EXAMPLES (Using a computer called **raijin.nci.org.au** with a user called **vader**):

### *COPYING TWO FILES TO RAIJIN*

```
scp mfile.c myfile.h vader@raijin.nci.org.au:
```

### *COPYING FROM RAIJIN WITH A WILDCARD*

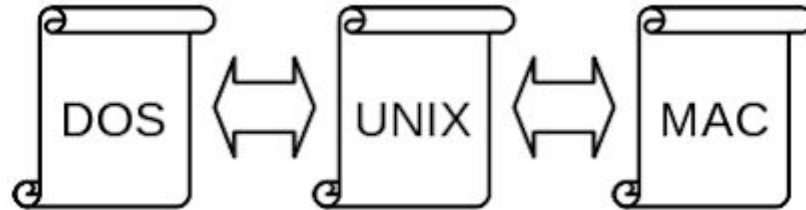
```
scp vader@raijin.nci.org.au:mydata/*.dat /home/user
```

Other interesting options ('man cp')

**-i** <identity\_file> use a predefined crypto-key file, removing the need to use a password. Use with caution!

**-r** copy directories recursively

## dos2unix and unix2dos



In Windows (DOS), text files have every line terminated with two characters –CARRIAGE RETURN and LINEFEED =>

*Using the C computer language syntax, this maps to `\r\n`*

In Unix and Mac, text files have every line terminated with `\n` only

When viewing a DOS file in Unix, you may see a **^M** at the end of every line.  
Or worst, the character may appear invisible and cause errors later !

Two tools exist to convert text files between the two Operating systems

**dos2unix <filename>**

**unix2dos <filename>**



# Making files easier to move

## COMPRESSING FILES

Lossless compression of files is useful:

- When transmitting files across the internet
- Saving space in your file system (NCI have quotas)



There are many tools to compress files. One of the most common is **gzip**.

To compress a file:

**gzip myfile.txt**

Produces a file called myfile.txt.gz (and removes myfile.txt)

To get the file back:

**gunzip myfile.txt.gz**

Produces a file called myfile.txt (and removes myfile.txt.gz)

# Creating and extracting archives

## Archiving files

It is often useful to combine a large number of files into one big file

- When transmitting files across the internet
- Quicker access on tape archive systems
- Your file system may have limits on the number of files you can have (N have quotas on this – the inode quota)



There are many tools to archive files. One of the most common is **tar**.

To create an archive:

**tar -cf archive.file.name.tar <list of files>**

Produces a file called **archive.file.name.tar**

To extract the archive

**tar -xf archive.file.name.tar**

Extracts all the files in the archive

Once an archive is created, it is a good idea to compress it as well.

This can be done automatically with tar by adding the **-z** flag. i.e. **tar -cfz ...**



## Exercise

You want to copy some data from another machine to your home directory. You know the file is called **whatami.tar.gz** and it lives in **/tmp** of a machine called **slurm0**. Use **scp** to fetch the file to your home on the login node. Don't forget the "." at the end of the line.

All Good



```
scp slurm0:/tmp/whatami.tar.gz .
```

Need help



Check that your the scp worked and you have a file called **whatami.tar.gz** in your directory. Now lets see what is in the archive..

```
tar -xf whatami.tar.gz
```

Use **ls** and **cat** to see what was produced by the file.

Note:

The **/tmp** file system of every Unix server is a disk local only to that node. I.e. the **/tmp** of every machine will be different.



## CHAPTER 7 : Using resources effectively



Although we covered requesting resources from the scheduler earlier, how do we know how much and what type of resources we will need in the first place?

Answer: we don't. Not until we've tried it ourselves at least once. We'll need to benchmark our job and experiment with it before we know how much it needs in the way of resources.

```
sacct -u {userID}
```

```
sacct -j {jobID} -l
```

- **Hostname** - Where did your job run?
- **MaxRSS** - What was the maximum amount of memory used?
- **Elapsed** - How long did the job take?
- **State** - What is the job currently doing/what happened to it?
- **MaxDiskRead** - Amount of data read from disk.
- **MaxDiskWrite** - Amount of data written to disk.



```
[user2@slurm-login ~]$ sacct -u user2
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
54	hostname	all	<a href="#">slurmclass</a>	2	COMPLETED	0:0
55	hostname	all	slurmclass	2	COMPLETED	0:0
56	run.sh	all	slurmclass	1	COMPLETED	0:0
56.batch	batch		slurmclass	1	COMPLETED	0:0
57	mpitest	all	slurmclass	2	COMPLETED	0:0
57.batch	batch		slurmclass	2	COMPLETED	0:0
58	mpitest	all	slurmclass	1	COMPLETED	0:0
58.batch	batch		slurmclass	1	COMPLETED	0:0
59	mpitest	all	slurmclass	3	COMPLETED	0:0
59.batch	batch		slurmclass	3	COMPLETED	0:0





```
[user2@slurm-login ~]$ sacct -j 59
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
59	mpitest	all	slurmclass	3	COMPLETED	0:0
59.batch	batch		slurmclass	3	COMPLETED	0:0

- The state showing that the job is completed successfully.
- The partition that was used in `all`
- The account that was used is `slurmclass`



If you would like to see more information about a job, you can always customise the format for your command

```
sacct -j 59 --format=User,JobID,Jobname,partition,state,time,start,end,elapsed,MaxRss,ncpus,nodelist
```

```
[user2@slurm-login ~]$ sacct -j 59 --format=User,JobID,Jobname,partition,state,time,start,end,elapsed,MaxRss,ncpus,nodelist
```

User	JobID	JobName	Partition	State	Timelimit	Start	End	Elapsed	MaxRSS	NCPUS	NodeList
user2	59	mpitest	all	COMPLETED	05:00:00	2018-08-03T00:35:23	2018-08-03T00:36:19	00:00:56		3	slurm0
	59.batch	batch		COMPLETED		2018-08-03T00:35:23	2018-08-03T00:36:19	00:00:56	32192K	3	slurm0





## **Why is my job not running?**

Use “`queue --start`” command

Jobs will be listed in order expected start time

Depending upon configuration, not all jobs may have a start time set

Times are only estimates and subject to change



```
01:15:37 m3-login1:~ ctan$ squeue --start
```

JOBID	PARTITION	NAME	USER	ST	START_TIME	NODES	SCHEDNODES	ODELIST(REASON)
2796646	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2796655	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2797577	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2797637	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2798766	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2798776	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2802511	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2802520	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2802526	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2802988	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2802994	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
2809141	comp	GPR_tens	yc1	PD	N/A	1	(null)	(DependencyNeverSatisfied)
3133116	m3i	nt	miguelg	PD	N/A	1	(null)	(launch failed requested held)
3297560	m3g	Refine3D	joshuamh	PD	N/A	1	(null)	(QOSMaxGRESPerUser)
3157584_[464-500]	comp	nCellsDe	mcfadyen	PD	N/A	1	(null)	(QOSMaxCpuPerUserLimit)
3312396	comp	hiplexpi	jste0021	PD	N/A	1	(null)	(Priority)
3312397	comp	hiplexpi	jste0021	PD	N/A	1	(null)	(Priority)
3312365	comp	hiplexpi	jste0021	PD	2018-08-03T01:35:12	1	m3i009	(Resources)
3312385	comp	hiplexpi	jste0021	PD	2018-08-03T01:35:12	1	m3i029	(Priority)
3312393	comp	hiplexpi	jste0021	PD	2018-08-03T01:35:13	1	m3i033	(Priority)
3312394	comp	hiplexpi	jste0021	PD	2018-08-03T01:35:16	1	m3i036	(Priority)
3312395	comp	hiplexpi	jste0021	PD	2018-08-03T01:35:16	1	m3h005	(Priority)
3273020	m3a	hsa_chn2	mziemann	PD	2018-08-03T03:21:01	1	m3a006	(Resources)
3284320	m3a	hsa_chn2	mziemann	PD	2018-08-03T03:21:01	1	m3a007	(Resources)
3130659	comp	oxalipia	sgwe0001	PD	2018-08-03T04:05:56	1	(null)	(Priority)
3221688	m3i	aFe5Dv3A	eyk	PD	2018-08-03T07:10:34	4	(null)	(QOSMaxCpuPerUserLimit)
3223197	m3i	aFe5Dv2B	eyk	PD	2018-08-03T07:10:34	4	(null)	(QOSMaxCpuPerUserLimit)
3073122	comp	238-cpcm	sgwe0001	PD	2018-08-03T07:16:35	1	m3c005	(Priority)
3073134	comp	238-smd-	sgwe0001	PD	2018-08-03T07:16:35	1	(null)	(Priority)
3073164	comp	238-gas-	sgwe0001	PD	2018-08-03T07:16:35	1	(null)	(Priority)
3073192	comp	248-cpcm	sgwe0001	PD	2018-08-03T07:16:35	1	(null)	(Priority)
3073193	comp	248-smd-	sgwe0001	PD	2018-08-03T07:16:35	1	(null)	(Priority)
3073195	comp	248-gas-	sgwe0001	PD	2018-08-03T07:16:35	1	(null)	(Priority)



## Be kind to the login node

- The login node is very busy managing lots and lots of jobs! It doesn't have any extra space to run computational work. Don't run jobs on the login node, you can use the interactive session to test out your script.
- Remember, the login node is to be shared with other users.
- If someone is being inappropriate and using the login node to run all of their stuff, message an administrator to take a look at things and deal with them.

## Test before scaling

- Before submitting a large run of jobs, submit one as a test first to make sure everything works.

## Understanding the filesystem

- always know where to write your data, in the production system, there's usually some space that being backed up and some space might be allocated for scratch
- make sure you check your quota for each of this filesystem



## Save time

- Compress files before transferring to save file transfer times with large datasets.
- The less resources you ask for, the faster your jobs will find a slot in which to run. Lots of small jobs generally beat a couple big jobs.

## Software tips

- You can generally install software yourself, but if you want a shared installation of some kind, it might be a good idea to message an administrator.
- Always use the default compilers if possible. Newer compilers are great, but older stuff generally has less compatibility issues.



# HPC Clusters in Monash University

## MASSIVE M3

A Computer for Next-Generation Data Science

1,700 Intel Haswell CPU-cores

2,000 Intel Skylake CPU-cores

48 NVIDIA Tesla K80 GPU

20 NVIDIA Tesla P100 GPU

60 NVIDIA Tesla V100 GPU

coprocessors for data processing and high end visualisation

A 3 petabyte Lustre parallel file system

100 Gb/s Ethernet Mellanox Spectrum  
Supplied by Dell, Mellanox and NVIDIA

STRUDEL ([desktop.massive.org.au](http://desktop.massive.org.au))





## Monarch

### Campus Cluster

**Provide Monash researchers with a local capability that focuses on engagement, education and community.**

### Investment

A co-investor model

Examples include Computational Chemistry, Astro and Fluid Dynamics

1/3rd of MonARCH is co-purchased

### Integrated into undergraduate study

#### CHM3911 Advanced Physical Chemistry

80 students across 3 practical sessions

Gaussian and GaussView for calculations

Students taught how to use a HPC system to perform their calculations



## WRAPPING UP

Do you have the tools you need to do your Research?

Would you want to know more?

- Email [mcc-help@monash.edu](mailto:mcc-help@monash.edu)
- Data Fluency Slack Channel
- Bioinformatic Drop-in (every Friday)
- <https://docs.monarch.erc.monash.edu.au>
- <https://docs.massive.org.au>
- HPC web sites (i.e. [www.nci.org.au](http://www.nci.org.au) or <https://pawsey.org.au/>)
- Google

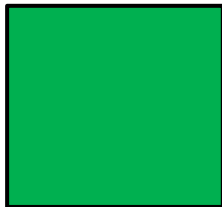






## **Your Feedback please!**

ONE THING YOU LIKED



ONE THING TO IMPROVE

