"This is an advanced course but we get mixed bag: students that have 5+ years of MPI programming on supercomputers, to students that have only done Java on Windows."

- Some background on supercomputing, high performance computing, parallel computing, scientific computing (there is overlap, but they're not the same thing).
- An introduction to Edward and Spartan, University of Melbourne's HPC system and new HPC/Cloud hybrid system
- Logging in, help, and environment modules.
- Job submission with TORQUE/MOAB and SLURM; simple submissions, multicore, job arrays, job dependencies, interactive jobs.
- Parallel programming with shared memory and threads (OpenMP) and distributed memory and message passing (OpenMPI)
'Supercomputer’ arbitrary term with no specific definition. In general use it means any single computer system (itself a contested term) that has exceptional processing power for its time. A well-adopted metric is the number of floating-point operations per second (FLOPS) such a system can carry out.

Supercomputers, like any other computing system, have improved significantly over time. The Top500 list is based on FLOPS using LINPACK - HPC Challenge is a broader, more interesting metric. The current number #1 system is Tianhe-2, a supercomputer developed by China’s National Super Computer Center.

1993: 124.50 GFLOPS
1994: 170.40 GFLOPS
1996: 368.20 GFLOPS
1997: 1.338 TFLOPS
1999: 2.3796 TFLOPS
2000: 7.226 TFLOPS
2004: 70.72 TFLOPS
2005: 280.6 TFLOPS
2007: 478.2 TFLOPS
2008: 1.105 PFLOPS
2009: 1.759 PFLOPS
2010: 2.566 PFLOPS
2011: 10.51 PFLOPS
2012: 17.59 PFLOPS
2013: 33.86 PFLOPS
2014: 33.86 PFLOPS
2015: 33.86 PFLOPS (54.90 PFLOP peak)
High-performance computing (HPC) is any computer system whose architecture allows for above average performance. A system that is one of the most powerful in the world, but is poorly designed, could be a "supercomputer".

Clustered computing is when two or more computers serve a single resource. This improves performance and provides redundancy in case of failure system. To describe simply, there are a collection of smaller computers strapped together with a high-speed local network (e.g., Myrinet, InfiniBand, 10 Gigabit Ethernet), although a low-speed network system could certainly be used.

Horse and cart as a computer system and the load as the computing tasks. Efficient arrangement, bigger horse and cart, or a teamster?

The clustered HPC is the most efficient, economical, and scalable method, and for that reason it dominates supercomputing today.
Parallel and Scientific Programming

With a cluster architecture, applications can be more easily parallelised across them. Parallel computing refers to the submission of jobs or processes over multiple processors and by splitting up the data or tasks between them (random number generation as data parallel, driving a vehicle as task parallel). Imagine trying to simulate massive parallel systems (e.g., the brain, the weather etc) in serial!

Scientific computing is the software applications used by the scientific community to aid research. Does not necessarily equate with high performance computing, or the use of clusters - it is whatever scientists use and do. This skills gap is a major problem and must be addressed because as the volume, velocity, and variety of datasets increases then researchers will need to be able to process this data.

Computational capacity does have a priority (the system must exist prior to use), in order for that capacity to realised in terms of usage a skill-set competence must also exist. The the core issue is that high performance compute clusters is just speed and power but also usage, productivity, correctness, and reproducibility. (image from Lawrence Livermore National Laboratory).
Scientific Computing

The importance of supercomputing cannot be measured simply by their processing capacity, but rather what is done with that capacity. Essentially however, any hard number-crunching or processor intensive visualisation requires a supercomputer for both speed and effectiveness. Supercomputers are typically (but not exclusively) used for scientific computing. Some applications have included weather forecasting, aerodynamic design, fluid mechanics, radiation modelling, molecular dynamics, CGI rendering for popular movies.

Supercomputers are increasingly important in a world of large datasets (e.g., Square Kilometre Array).

A local example, researchers from Monash University, the Peter MacCallum Cancer Institute in Melbourne, the Birkbeck College in London, and VPAC in 2010 unravelled the structure the protein perforin to determine how pathogenic cells (c.f., http://www.nature.com/nature/journal/v468/n7322/full/nature09518.html).
HPC Cluster Design
In November 2015 of the Top 500 Supercomputers worldwide, every single machine used a "UNIX-like" operating system; 98.8% used Linux, 1.2% used AIX.

The command-line interface provides a great deal more power and is very resource efficient.

GNU/Linux scales and does so with stability and efficiency. Critical software such as the Message Parsing Interface (MPI) and nearly all scientific programs are designed to work with GNU/Linux.

The operating system and many applications are provided as "free and open source", which means that not only are there are some financial savings, were also much better placed to improve, optimize and maintain specific programs.

Free or open source software (not always the same thing) can be compiled from source for the specific hardware and operating system configuration, and can be optimised according to compiler flags. There is necessary where every clock cycle is important.
Flynn's Taxonomy and Multicore Systems

It is possible to illustrate the degree of parallelisation by using Flynn's Taxonomy of Computer Systems (1966), where each process is considered as the execution of a pool of instructions (instruction stream) on a pool of data (data stream).

Over time computing systems have moved towards multi-processor, multi-core, and often multi-threaded and multi-node systems.

The engineering imperative to these systems comes down to heat. From the mid-2000s clock speed on CPUs have largely stalled.

Some trends include massive multicore systems (e.g., The Angstrom Project, the Tile CPU with 1000 cores) and massive network connectivity and shared resources (e.g., Plan9 Operating System).

(Image from Dr. Mark Meyer, Canisius College)
Parallel programming and multicore systems should mean better performance. This can be expressed a ratio called speedup

\[ \text{Speedup} (p) = \frac{\text{Time (serial)}}{\text{Time (parallel)}} \]

Correctness in parallelisation requires synchronisation (locking). Synchronisation and atomic operations causes loss of performance, communication latency.

Amdahl's law, establishes the maximum improvement to a system when only part of the system has been improved. Gustafson and Barsis noted that Amdahl's Law assumed a computation problem of fixed data set size.
Edward - UniMelb Cluster - System

Installed and operational since 2011. Named after King of Wessex (prior cluster was Alfred)
Hardware: 48 main compute nodes  2x AMD Opteron 6128 (16 cores per node), plus water partion (3 * 40 core nodes), pophealth (4 * 64 core nodes), gpu (3 * 8 core, plus GPUs), ashley (10 * 64 core). 1TB main localdisk, 32 GB memory.
2 Storage Nodes. IBM x3650 (for user and project data, 45TB each)
Interconnect and Filesystem: 10GbE with XFS, NFS
Resource Manager and Job Scheduler: Torque, Moab

As of February 9, 2015: 816 active users: (including 25 from Victoria University) and 333 projects.
From January 1st 2015 to December 31st 2015: 4,930,340 compute hours from 375,458 jobs;
74.84% utilisation (and increasing).

Major projects in the past year; physarum optimisation, fluid flow with deforming interfaces in microchannels, bushfire dynamics (three different projects), path analysis for genomic datasets, efficiency in luminescent solar concentrators ....

Users have not had been engaged in an extensive HPC training program; approximately only 38 student/days training from 2012-2014. Compares strangely to other extensive ResBaz activities. As a result, single-core and low memory jobs dominate; 76.35% of jobs from Feb 9 2015 to Feb 9 2016 were single core, and 96.83% used 1-4GB of memory.
A detailed review was conducted last year looking at the infrastructure of the Melbourne Research Cloud, High Performance Computing, and Research Data Storage Services. University desired a 'more unified experience to access compute services'

Recommended solution, based on technology and usage, is to make use of existing NeCTAR Research cloud with an expansion of general cloud compute provisioning and use of a smaller "true HPC" system on bare metal nodes.

The HPC will be called "Spartan" (not Æthelstan or Ælfwæord!).

The 'bare metal' HPC component really will be laconic. Think of Sparta's citenzship structure - the few Spartiate citizens are bare metal HPC, the more numerous Perioeci free inhabitants are vHPC nodes, and the many Helot slaves are elastic compute nodes. "Real" HPC is a mere c200 cores, 16 GB per core. 2 socket Intel E5-2643 v3 CPU with 6-core per socket, 192GB memory, 2x 1.2TB SAS drives, 2x 40Gbe network
Slurm will be implemented for job management and integrated with Openstack with Cloudbursting ("elastic computing"). Burst capability into the Research Cloud when resources are not available in the HPC cluster (vHPC) (1:1 subscription of CPU for vHPC)

Single authentication with Openstack and HPC Job Management
4000 cores with 4+GB per virtual core for the Research Cloud Compute environment.
Server hardware is 2 socket Intel E5-2699 v3 CPU with 18-core per socket 384 GB memory.

Initial live tests being conducted by 250 students in March entirely on vHPC system (that's you!).

Widespread existing performance metrics suggest that cloud computing performs poorly compared to HPC in MPI and distributed computing but is comparable with single-core or multi-core shared memory nodes with minimal overhead.
Moving Towards A New System
To log on to a HPC system, you will need a user account and password and a Secure Shell (ssh) client. Most HPC cluster administrators do not allow connections with protocols such as Telnet, FTP or RSH as they insecurely send passwords in plain-text over the network, which is easily captured by packet analyser tools (e.g., Wireshark). Linux distributions almost always include SSH as part of the default installation as does Mac OS 10.x, although you may also wish to use the Fugu SSH client. For MS-Windows users, the free PuTTY client is recommended. To transfer files use scp, WinSCP, Filezilla, or rsync.

Logins to Edward are based on POSIX identity for the Edward system

```
ssh your-username@edward.hpc.unimelb.edu.au or
```

Logins to Spartan are based on University Active Directory credentials.

```
ssh your-unimelb-email@spartan.hpc.unimelb.edu.au OR
ssh 'your-unimelb-email'@spartan.hpc.unimelb.edu.au OR
ssh UNIMELB\yourUniID@spartan.hpc.unimelb.edu
```

If a user has problems with submitting a job, needs a new application or extension to an existing application installed, if their submissions are generated unexpected errors etc., an email can be sent to: hpc-support@unimelb.edu.au
Assumption here is that everyone has had exposure to the Linux command line. If not, you'd better get some! At least learn the twenty or so basic environment commands to navigate the environment, manipulate files, manage processes. Plenty of good online material available (e.g., "Supercomputing with Linux", https://github.com/VPAC/superlinux)

Environment modules provide for the dynamic modification of the user's environment (e.g., paths) via module files. Each module contains the necessary configuration information for the user's session to operate according to the modules loaded, such as the location of the application's executables, its manual path, the library path, and so forth.

Modulefiles also have the advantages of being shared with many users on a system and easily allowing multiple installations of the same application but with different versions and compilation options. Sometimes users want the latest and greatest of a particular version of an application for the feature-set they offer. In other cases, such as someone who is participating in a research project, a consistent version of an application is desired. Having multiple version of applications available on a system is essential in research computing.
Some basic module commands include the following:

module help
The command module help, by itself, provides a list of the switches, subcommands, and
subcommand arguments that are available through the environment modules package.

module avail
This option lists all the modules which are available to be loaded.

module whatis <modulefile>
This option provides a description of the module listed.

module display <modulefile>
Use this command to see exactly what a given modulefile will do to your environment, such as
what will be added to the PATH, MANPATH, etc. environment variables.
**module load** `<modulefile>`
This adds one or more modulefiles to the user's current environment (some modulefiles load other modulefiles).

**module unload** `<modulefile>`
This removes any listed modules from the user's current environment.

**module switch** `<modulefile1>  <modulefile2>`
This unloads one modulefile (modulefile1) and loads another (modulefile2).

**module purge**
This removes all modules from the user's environment.

In the lmod system as used on Spartan there is also “module spider” which will search for all possible modules and not just those in the existing module path.

(Image from NASA, Apollo 9 “spider module”)
The Portable Batch System (or simply PBS) is a utility software that performs job scheduling by assigning unattended background tasks expressed as batch jobs among the available resources. The scheduler provides for parameterisation of computer resources, an automatic submission of execution tasks, and a notification system for incidents.

The original Portable Batch System was developed by MRJ Technology Solutions under contract to NASA in the early 1990s. In 1998 the original version of PBS was released as an open-source product as OpenPBS. This was forked by Adaptive Computing (formally, Cluster Resources) who developed TORQUE (Terascale Open-source Resource and QUEue Manager). Many of the original engineering team and what commercial property of exists from the original product is now part of Altair Engineering who have their own version, PBSPro. In addition to this the popular job scheduler SLURM (Simple Linux Utility for Resource Management) also uses batch script where are very similar in intent and style to PBS scripts.

Edward uses TORQUE. Spartan uses SLURM. A job script written on one needs to be translated to another (handy script available pbs2slurm https://github.com/bjpop/pbs2slurm)

In addition to this variety of implementations of PBS different institutions may also make further elaborations and specifications to their submission filters (e.g., site-specific queues, user projects for accounting).
Submitting and Running Jobs

Jobs and subjobs to run

[Application]

Job queue

Job scheduler

Collecting results
Submitting and Running Jobs

(Image from the otherwise dry IBM 'Red Book' on Queue Management)

Submitting and running jobs is a relatively straight-forward process consisting of:

1) Setup and launch
2) Job Control, Monitor results
3) Retrieve results and analyse.

Don't run jobs on the login node! Instead, using the queuing system to submit jobs.

1. Setup and launch consists of writing a short script that initially makes resource requests (walltime, processors, memory, queues) and then commands (loading modules, changing directories, running executables against datasets etc), and optionally checking queueing system.

Core command for checking queue in TORQUE: `showq`
Core command for checking queue in SLURM: `squeue`
Core command for job submission in TORQUE: `qsub [jobscript]`
Core command for job submission in SLURM: `sbatch [jobscript]`
Submitting and Running Jobs

2. Check job status (by ID or user), cancel job.

   Core command for checking job in TORQUE: `qstat [jobid]`
   Core command for checking job in Moab: `checkjob [jobid]`
   Core command for checking job in Slurm: `squeue -j [jobid]`
   Detailed command in Slurm: `scontrol show job [jobid]`
   Core command for deleting job in TORQUE: `qdel [jobid]`
   Core command for deleting job in Slurm: `scancel [jobid]`

3. Both TORQUE and Slurm provide error and output files (combined into one by default in SLURM). They may also have files for post-job processing. Graphic visualisation is best done on the desktop.
The script first invokes a shell environment, followed by the partition the job will run on (the default is 'compute' for TORQUE/Edward and 'cloud' for SLURM/Spartan). The next four lines are resource requests, specifically for one compute node, one task, and cpu core per task. After these requests are allocated, the script loads a module and then runs the executable against the dataset specified. Note that in TORQUE you need to specify a change to the working directory, where you launched your job from; this is not necessary in SLURM. SLURM also automatically exports your environment variables when you launch your job.
Modifying resource allocation requests can improve job efficiency. For TORQUE/Edward use the same script as previously provided but change the resource request as follows:

```
#PBS -l nodes=2:ppn=2
```

For example shared-memory multithreaded jobs on SLURM/Spartan (e.g., OpenMP), modify the `--cpus-per-task` to a maximum of 16, which is the maximum number of cores on a single instance.

```
#SBATCH --cpus-per-task=16
```

For distributed-memory multicore job using message passing, the multinode partition has to be invoked and the resource requests altered e.g.,

```
#!/bin/bash
#SBATCH -p cloud
#SBATCH --nodes=2
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=1
module load my-app-compiler/version
srun my-mpi-app
```

Note that there is only 1 CPU per task, which is typical with code written with message passing. Multinode jobs on SLURM/Spartan may be slower if they have a lot of interprocess communication.
Alternative Job Submissions

Alternative job submissions include specifying batch arrays, batch dependencies, and interactive sessions.

In the first case, the same batch script, and therefore the same resource requests, is used multiple times. A typical example is to apply the same task across multiple datasets. The following example submits 10 batch jobs with myapp running against datasets dataset1.csv, dataset2.csv, ... dataset10.csv

```
#PBS -t 1-10
myapp ${PBS_ARRAYID}.csv
```

```
#SBATCH --array=1-10
myapp ${SLURM_ARRAY_TASK_ID}.csv
```

In the second case a dependency condition is established on which the launching of a batch script depends, creating a conditional pipeline. The dependency directives consist of `after`, `afterok`, `afternotok`, `before`, `beforeok`, `beforenotok`. A typical use case is where the output of one job is required as the input of the next job.

```
#PBS -W x=depend:afterok:myfirstjob

#SBATCH --dependency=afterok:myfirstjobid mysecondjob
```
In the third case TORQUE or SLURM, based on the resource requests made on the command line, puts the user on to a compute node. This is typically done if the user wants to run a large script (and shouldn't do it on the login node), or wants to test or debug a job. The following command would launch one node with two processors for ten minutes.

```
[lev@edward ~]$ qsub -l nodes=1:ppn=2,walltime=0:10:0 -I
qsub: waiting for job 2132478.edward-m to start
qsub: job 2132478.edward-m ready

[lev@edward042 ~]$ 

[llafayette@unimelb.edu.au@spartan interact]$
```

```
sinteractive --nodes=1 --ntasks-per-node=2 --time=0:10:0
srun: job 164 queued and waiting for resources
srun: job 164 has been allocated resources
```

```
[llafayette@unimelb.edu.au@spartan001 interact]$
```
## PBS, SLURM Comparison

<table>
<thead>
<tr>
<th>User Commands</th>
<th>PBS/Torque</th>
<th>SLURM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job submission</td>
<td><code>qsub [script_file]</code></td>
<td><code>sbatch [script_file]</code></td>
</tr>
<tr>
<td>Job submission</td>
<td><code>qdel [job_id]</code></td>
<td><code>scancel [job_id]</code></td>
</tr>
<tr>
<td>Job status (by job)</td>
<td><code>qstat [job_id]</code></td>
<td><code>squeue [job_id]</code></td>
</tr>
<tr>
<td>Job status (by user)</td>
<td><code>qstat -u [user_name]</code></td>
<td><code>squeue -u [user_name]</code></td>
</tr>
<tr>
<td>Node list</td>
<td><code>pbsnodes -a</code></td>
<td><code>sinfo -N</code></td>
</tr>
<tr>
<td>Queue list</td>
<td><code>qstat -Q</code></td>
<td><code>squeue</code></td>
</tr>
<tr>
<td>Cluster status</td>
<td><code>showq, qstatus -a</code></td>
<td><code>squeue -p [partition]</code></td>
</tr>
</tbody>
</table>

### Environment

<table>
<thead>
<tr>
<th></th>
<th>$PBS_JOBID</th>
<th>$SLURM_JOBID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job ID</td>
<td>$PBS_O_WORKDIR</td>
<td>$SLURM_SUBMIT_DIR</td>
</tr>
<tr>
<td>Submit Directory</td>
<td>$PBS_O_HOST</td>
<td>$SLURM_SUBMIT_HOST</td>
</tr>
<tr>
<td>Submit Host</td>
<td>$PBS_NODEFILE</td>
<td>$SLURM_JOB_NODELIST</td>
</tr>
<tr>
<td>Node List</td>
<td>$PBS_ARRAYID</td>
<td>$SLURM_ARRAY_TASK_ID</td>
</tr>
<tr>
<td>Job Specification</td>
<td>PBS</td>
<td>SLURM</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Script directive</td>
<td>#PBS</td>
<td>#SBATCH</td>
</tr>
<tr>
<td>Queue</td>
<td>-q [queue]</td>
<td>-p [queue]</td>
</tr>
<tr>
<td>Job Name</td>
<td>-N [name]</td>
<td>--job-name=[name]</td>
</tr>
<tr>
<td>Nodes</td>
<td>-l nodes=[count]</td>
<td>-N [min[-max]]</td>
</tr>
<tr>
<td>CPU Count</td>
<td>-l ppn=[count]</td>
<td>-n [count]</td>
</tr>
<tr>
<td>Wall Clock Limit</td>
<td>-l walltime=[hh:mm:ss]</td>
<td>-t [days-hh:mm:ss]</td>
</tr>
<tr>
<td>Event Address</td>
<td>-M [address]</td>
<td>--mail-user=[address]</td>
</tr>
<tr>
<td>Event Notification</td>
<td>-m abe</td>
<td>--mail-type=[events]</td>
</tr>
<tr>
<td>Memory Size</td>
<td>-l mem=[MB]</td>
<td>--mem=[mem][M</td>
</tr>
<tr>
<td>Proc Memory Size</td>
<td>-l pmem=[MB]</td>
<td>--mem-per-cpu=[mem][M</td>
</tr>
</tbody>
</table>
One form of parallel programming is multithreading, whereby a master thread forks a number of sub-threads and divides tasks between them. The threads will then run concurrently and are then joined at a subsequent point to resume normal serial application.

One implementation of multithreading is OpenMP (Open Multi-Processing). It is an Application Program Interface that includes directives for multi-threaded, shared memory parallel programming. The directives are included in the C or Fortran source code and in a system where OpenMP is not implemented, they would be interpreted as comments.

There is no doubt that OpenMP is an easier form of parallel programming, however it is limited to a single system unit (no distributed memory) and is thread-based rather than using message passing.

(Image from Wikipedia)

Examples in C and Fortran follow.
```c
#include <stdio.h>
#include "omp.h"
int main(void)
{
    int id;

    #pragma omp parallel num_threads(4) private(id)
    {
        int id =omp_get_thread_num();
        printf("Hello world \%d\n", id);
    }
    return 0;
}

program hello2omp
    include "omp_lib.h"
    integer :: id
    !$omp parallel num_threads(17) private(id)
        id = omp_get_thread_num()
        print *, "Hello world", id
    !$omp end parallel
end program hello2omp
```
Moving from shared memory to parallel programming involves a conceptual change from multi-threaded programming to a message passing paradigm. In this case, MPI (Message Passing Interface) is one of the most well popular standards and is used here, along with a popular implementation as OpenMPI.

The core principle is that many processors should be able cooperate to solve a problem by passing messages to each through a common communications network.

The flexible architecture does overcome serial bottlenecks, but it also does require explicit programmer effort (the "questing beast" of automatic parallelisation remains somewhat elusive).

The programmer is responsible for identifying opportunities for parallelism and implementing algorithms for parallelisation using MPI.

Again simple hello-world examples are provided for C and Fortran with the associated TORQUE and SLURM job submission scripts..
```c
#include <stdio.h>
#include "mpi.h"

int main( argc, argv )
int  argc;
char **argv;
{
  int rank, size;
  MPI_Init( &argc, &argv );
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  printf( "Hello world from process %d of %d\n", rank, size );
  MPI_Finalize();
  return 0;
}
```
Distributed Memory Parallel Programming

! Fortran MPI Hello World
program hello
include 'mpif.h'
integer rank, size, ierror, tag, status(MPI_STATUS_SIZE)
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
print*, 'node', rank, ': Hello world'
call MPI_FINALIZE(ierr)
end
Distributed Memory Parallel Programming

#!/bin/bash
#PBS -N HelloWorld
#PBS -l nodes=2:ppn=16
#PBS -l walltime=00:10:00
cd $PBS_O_WORKDIR
module load openmpi-gcc
mpiexec ./mpi-helloworld

#!/bin/bash
#SBATCH --nodes-2
#SBATCH --ntasks=16
module load OpenMPI
srun ./mpi-helloworld
THANKS FOR WATCHING
& LISTENING PATIENTLY