Programming Environment

Cornell Center for Advanced Computing

January 14, 2015

Thanks to Dan Stanzione, Bill Barth, Lars Koesterke, Kent Milfeld, Doug James, and Robert McLay for their materials developed at TACC and XSEDE that were incorporated into this talk.

1. Accessing Stampede
2. Login Environment
3. Stampede Overview
4. Software
5. Compiling
6. Timing
7. Editing Files
8. Batch Job Submission: SLURM
9. Play Nice
10. Help

1. Accessing Stampede

Before you Start

- Get an XSEDE Portal Account : https://portal.xsede.org/
- Get an Allocation (computing hours)
  - PI must request allocation through appropriate portal
  - PI may use portal to assign active users to an allocation
- Note your allocation's "project name" (account code)
- Activate your account on TACC resources
  - Involves email handshake(s) with TACC user services
  - May take a few business days
  - Note that your TACC credentials (think ssh) may differ from XSEDE
  - To activate, log into TACC portal, activation is immediate.
- Reset password on TACC portal. Takes 30 minutes to propagate.

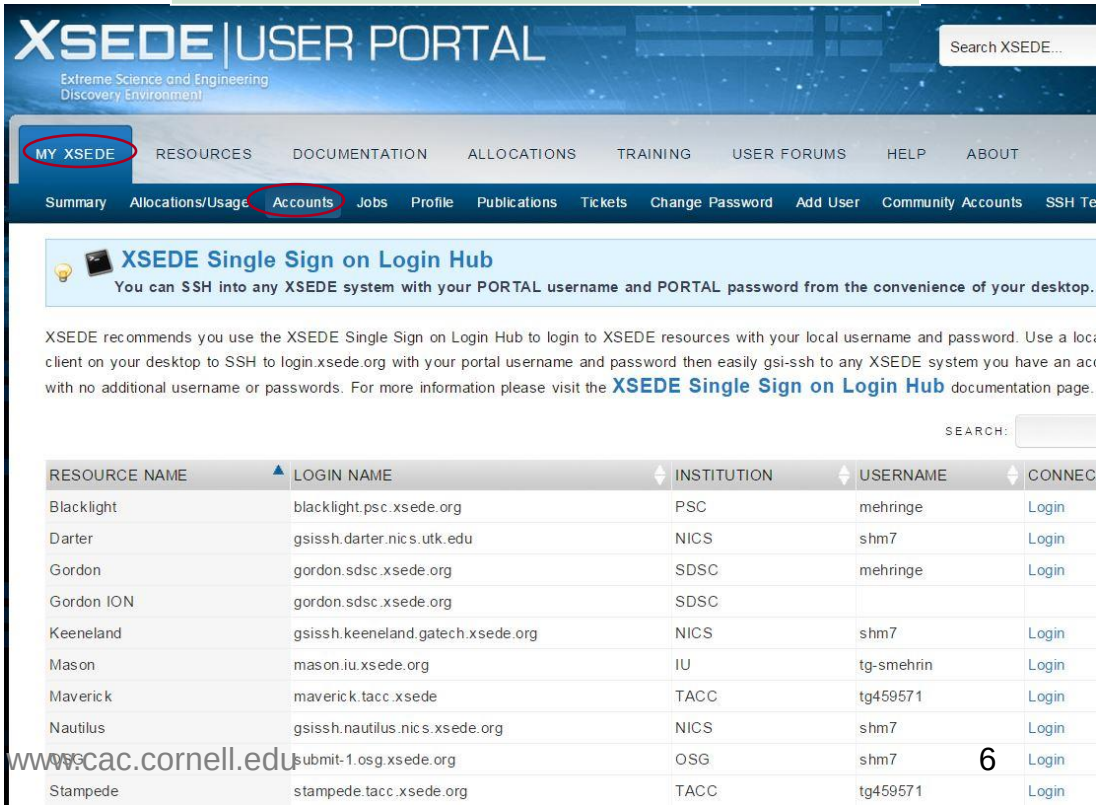
Logging into XSEDE Resources:

- Command Line (Unix/Linux, or Mac Terminal window) – ssh
- SSH / telnet client – e.g. Putty or Secure Shell Client
- Single Sign On (SSO) from the XSEDE User Portal

XUP: View Accounts

- Go to the XSEDE User Portal (XUP): portal.xsede.org
- Log in
- Go to 'My XSEDE' tab
- Choose 'Accounts' on the nav bar
- See which resources you have access to
- Note: you may not have the same username on all resources

View accounts using the XUP



Login with SSH:

- SSH Secure Shell - client for Windows
- You will be connected to login#.stampede.tacc.utexas.edu
- **Do not** overwrite ~/.ssh/authorized_keys
- **Do** add stampede to the list of known hosts, if prompted

If you're using your linux or Mac machine:
Using the SSH, login to stampede.tacc.utexas.edu:
$ ssh tg459xxx@stampede.tacc.utexas.edu

-or-

If you're using one of the lecture room's machines:
Using PuTTY, login to stampede.tacc.utexas.edu:
Click on  then enter ***putty*** into the search box
Use Host Name: stampede.tacc.utexas.edu

Single Sign On (SSO) Login Hub

- **login.xsede.org** - a single point-of-entry to the XSEDE cyberinfrastructure.
- Log in to the hub with your XUP username and password
  - A 12 hour proxy certificate is automatically generated
- gsissh to any XSEDE compute resource
- Pros:
  - No need for a resource-specific username and password.
  - Easy way to switch between XSEDE resources

https://portal.xsede.org/web/xup/single-sign-on-hub

2. Login Environment

Account Info

Note your account number in the splash screen.

```
---------------------- Project balances for user tg459571 ----------------------
| Name           Avail SUs     Expires    |                                     |
| TG-TRA120006       49998     2015-12-02 |                                     |
------------------------ Disk quotas for user tg459571 -------------------------
| Disk         Usage (GB)     Limit    %Used   File Usage       Limit   %Used |
| /home1              0.0       5.0     0.07          125      150000    0.08 |
| /work               0.0    1024.0     0.00           11    30000000    0.00 |
-------------------------------------------------------------------------------
```

Get the Lab Files

- TAR = Tape ARchive. Just concatenates files.
- tar <switches> <files>
  - z = compress or decompress
  - x = extract
  - c = create
  - v = verbose
  - t = list files
  - f = next argument is the file to read or write
- ~*username* is the home directory of that user
- For example, to create a tar: *tar cvf myfiles.tar dir1 dir2 README*

Get the lab files:
$ tar xvf ~tg459572/LABS/envi.tar

Change directory to the envi directory:
$ cd envi
List the lab files:
$ ls -R

Stampede's Operating System: Linux

$ pwd	(Print the current directory)
$ ls -la	(List the content of the current directory)
$ cd $HOME	(Change the working directory to home directory)
$ cat .profile	(Print the file *.profile* to the screen)
$ mkdir testdir	(Create the directory, *testdir*)
$ touch test.txt	(touch renews a file's timestamp, but here is used to create an empty file)
$ mv test.txt testdir	(Move text.txt into the directory testdir)
$ rm -r testdir	(Delete the directory and all subdirectories)
$ man ls	(Show the manual page for ls, **q** to quit)
$ env	(Show all environment/global variables)
$ export newgreeting="Hello World"	(Set an environment variable)
$ echo $newgreeting	(Print the variable *newgreeting*)

Shells and Startup Scripts on Stampede

Shells:

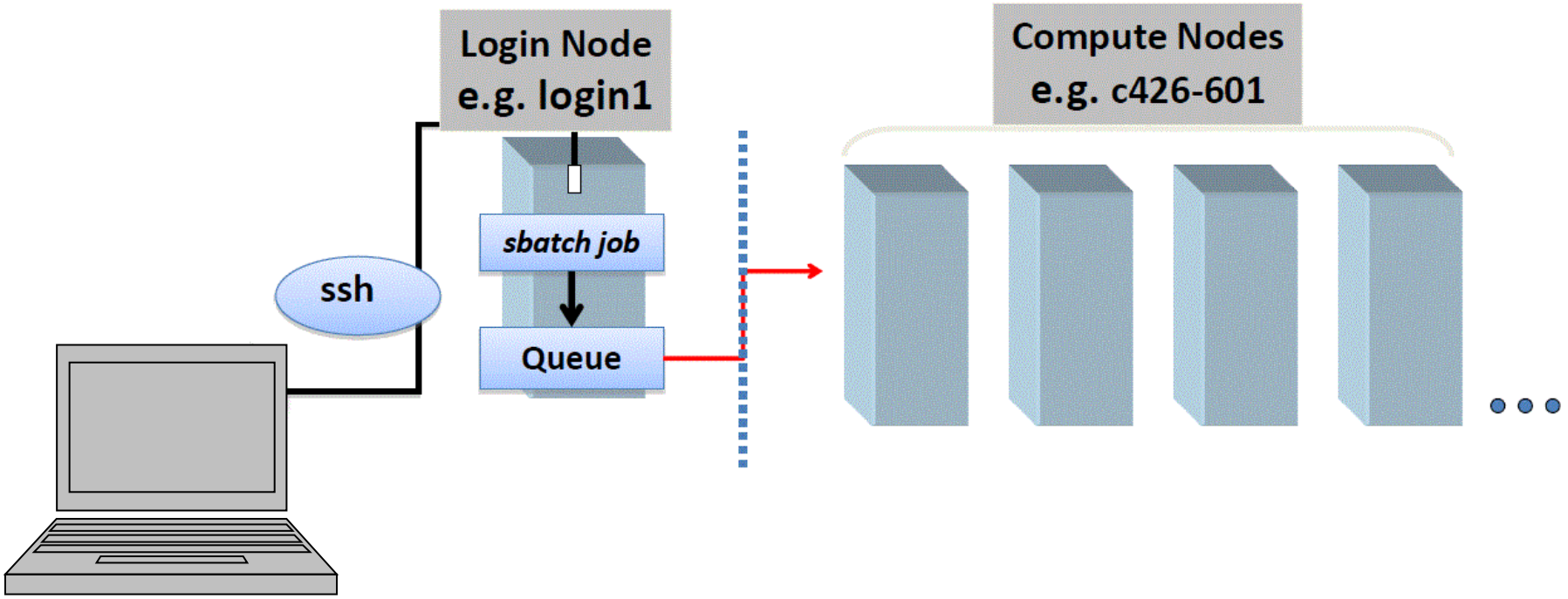
- bash is the default shell on Stampede
- TACC supports most major shells, e.g. csh, tcsh, zsh …
- To change your default shell, submit a ticket (chsh won't work)

Startup Scripts:

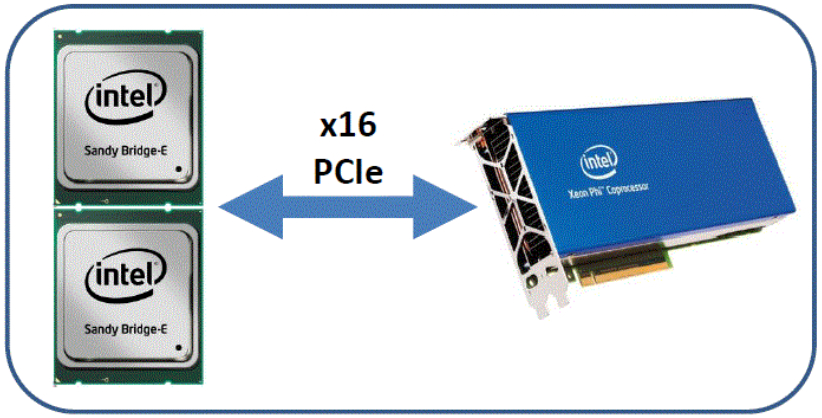
- When you log in, system-level startup files execute to allow administrators to enhance and customize the environment
- Enhance your shell environment, not your account
- Don't use "echo" in startup scripts, will break other tools
- TACC staff recommends that Bash shell users use ~/.profile rather than .bash_profile or .bash_login.
  Bash Users' Startup Files: Quick Start Guide

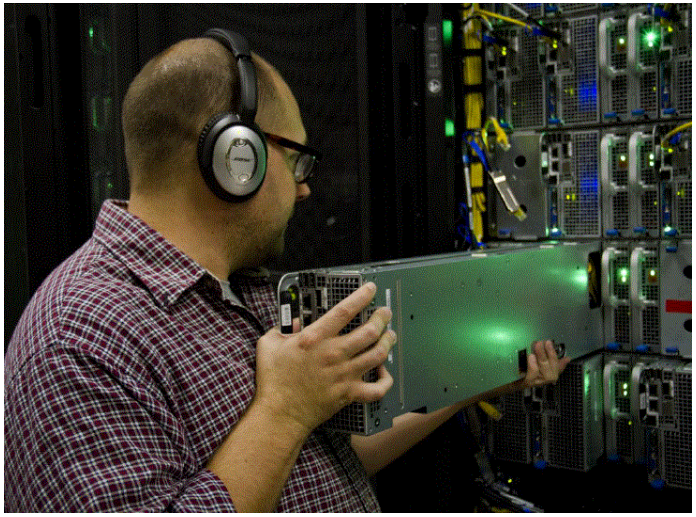
3. Stampede Overview

The Generic Environment



Typical Stampede Node (= blade)





16 cores
32G RAM

Two Xeon E5
8-core processors

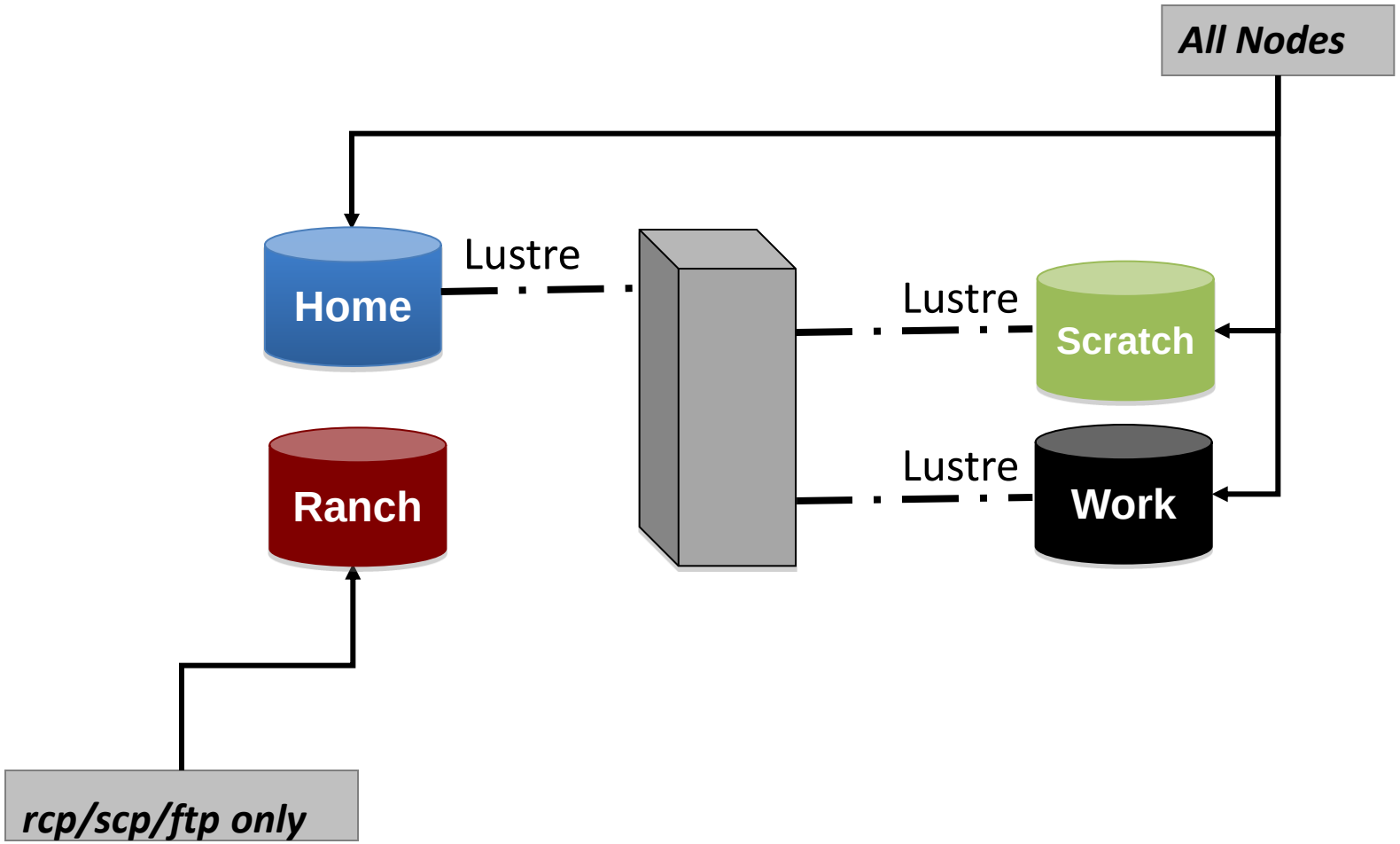
61 lightweight cores
8G RAM

Xeon Phi Coprocessor
Each core has 4 hardware threads

MIC runs lightweight
Linux-like OS (BusyBox)

System	Stampede	Memory
Nodes	~6400 (in 160 racks) ➔ 96,000+ total cores	
Typical Node	16 cores: 2 cpus/node x 8 cores/cpu	32GB RAM
	61 cores on MIC coprocessor	8G RAM
Special Nodes	16 large memory nodes (32 Xeon cores)	1TB/node RAM
	128 GPU nodes (w/ NVIDIA Kepler 2 & MIC)	2GB/core
	Login nodes (don't have MIC)	
CPUs	Intel Sandy Bridge Intel Xeon Phi coprocessor	
Interconnect	56Gb FDR IB	
Disk	14PB Lustre (IB)	

Available File Systems



File System

Environment Variable	Purpose	User Access Limits	Lifetime
$HOME	Source code	5 GB	Backups
$WORK	Large file storage	1.0 TB	No backup
$SCRATCH	Large files needed by compute jobs	~8.5PB total	Purged after 10 days
/tmp	Local disk on batch job node	~80 GB / node	Purged after job ends
${ARCHIVER}:$ARCHIVE	Archival tape	Essentially unlimited	Project

Sharing Files with your Research Group

- All accounts have a default group when the account is created
- All usernames sharing an allocation should be in a common group (If they are not, submit a ticket)

$ groups <username>	Display groups that username belongs to
$ groups	Display groups that you belong to
$ id –g –n	Display your default group
$ id <username>	Display username, default group, all groups, for that user
$ touch test.txt	Create a file
$ ls -la	Display your files, including group information. Note that the file you just created has your default group ownership
$ chgrp -v G-803077 test.txt	Change the group ownership of a file to a different group (verbose output)
$ chmod 644 test.txt	Modify permissions so everyone in that group has access

Sharing Files with your Research Group

Want to share files with your colleagues, but you have different default groups? You can

a) submit a ticket to get your default group changed, or

b) create a common folder with the proper settings:

$ mkdir /scratch/01871/apb18/test	Create a directory for everybody to share
$ chmod g+rwx /scratch/01871/apb18/test	Set permissions allow group read/write/execute (also make sure the parent dir's permissions aren't too restrictive).
$ chgrp G-803077 test	Change its group via 'chgrp' to the common group
$ chmod g+s test	Set the setgid bit, so that everything created underneath it will inherit its group.
$ umask 177	Everyone in the group should use an appropriate umask such as 002 or 117, so that files they create are actually group readable and writable. (Put this into a login script!)

File System

$ lfs quota -u <*username*> $HOME	see quota limits & usage
$ lfs quota -u <*username*> $WORK	
$ lfs quota -u <*username*> $SCRATCH	
$ cd	change directory to $HOME
$ pwd	
$ cdw	change directory to $WORK
$ pwd	
$ cds	change directory to $SCRATCH
$ pwd	
$ du -sh	see how much space is available in the current user-owned directory
$ df -k .	see the amount of disk space used in a file system, "." meaning in the current directory

Note: File systems are visible from MIC, but use full explicit paths

4. Software

Software

Use the module utility on Stampede to provide a consistent, uniform method to access software

- Loads specific versions of libraries/executables
- Manages dependencies between multiple compilers and software stacks
- Works in your batch file, Makefile, and scripts, but not on MICs
- Affects $PATH, $MANPATH, $LIBPATH
- Order matters! First choose compiler, then application software.

Software available on Stampede

Software search available on XSEDE

Lmod is TACC's Module System

Setting your Default Software Environment

Set and save your personal default module environment:

```
$ module reset                  # return to the default environment
$ module load ddt
$ module load fftw3
$ module save                   # will load at login or restore
```

Create a named collection of modules for reliability and repeatability:

```
$ module save chemtools
        …
$ module restore chemtools
```

Module

This utility is used to set up your PATH and other environment variables:

$ module help	{lists options}
$ module avail	{lists available modules}
$ module list	{lists loaded modules}
$ module load boost	{add a module}
$ module unload boost	{remove a module}
$ module help <module_name>	{module-specific help}
$ module spider	{lists all modules}
$ module spider petsc	{list all versions of petsc}
$ module swap mvapich2 impi	{unload module A, load module B}
$ module reset	{return to system defaults}

5. Compiling

Compiling Serial Code

- The default compilers on Stampede are Intel C++ and Fortran
  - These are the only compilers that support the Phi coprocessors
- Compilers are available on login and compute nodes
  - But not on MIC coprocessors; compile from a Sandy Bridge host
- Use **man** or **–help** option, e.g. **man icc**.

Compiler	Language	File Extension	Example
icc	C	.c	icc *compiler_options* prog.c
icpc	C++	.C, .cc, .cpp, .cxx	icpc *compiler_options* prog.cpp
ifort	F77	.f, .for, .ftn	ifort *compiler_options* prog.f
ifort	F90	.f90, .fpp	ifort *compiler_options* prog.f90

- Use the **module** command to list modules & versions & to change the default compiler.
- Use MKL for math libraries; it has MIC support
- Also available: several versions of gcc suite and some specialized compilers, e.g. cuda support (nvcc): **module load cuda**

Compiler Options

- Use compiler options to achieve optimal performance.
- To obtain best results:
  - Select the appropriate optimization level
  - Target the architecture of the computer (CPU, cache, memory system)
  - Allow for interprocedural analysis (inlining, etc.)
- No single answer for all cases; test different combinations.

Optimization Level	Description
-O0	Fast compilation, full debugging support. Automatically enabled if using -g.
-O1 -O2	Low to moderate optimization, partial debugging support:
-O3	Aggressive optimization - compile time/space intensive and/or marginal effectiveness; may change code semantics and results (sometimes even breaks code!)

See the User Guide for additional compiler options.

Makefiles

$ cd $HOME/envi/using_makefiles

$ cat Makefile	Read over the Makefile
$ make	Compile the program, generate a.out
$ make	Reports “up to date”, i.e. not recompiled
$ touch suba.f	Simulate changing a file
$ make	suba.f (and only suba.f) is recompiled

6. Timing

Timers

- Time your code to see how long your program runs and estimate if it's having gross difficulties. Gauge effectiveness of code and software changes.
- Wall-clock time in a dedicated environment is most accurate
- **/usr/bin/time -p** is preferred over the shell's time command (-p specifies traditional precision output in seconds)

```
$ cd $HOME/envi/intro
$ make
g++ hello.c -o hello
$ /usr/bin/time -p ./hello
Hello world!
real 0.01
user 0.00
sys 0.01
$
```

You can also time specific sections of your code by inserting timer calls before and after important sections.

Profilers: gprof (GNU profiler)

- gprof reports a basic profile of time spent in each subroutine
- Find the most time-consuming routines, the hotspots
- As with all profiling tools, the code must be instrumented to collect the timing data and then executed to create a raw-date report file.
- Read the data file into an ASCII report or a graphic display.
- Instrument the code by recompiling using the -pg option (Intel)
- More detail can be found in the Profiling and Debugging Virtual Workshop module.

```
$ cd $HOME/envi/precision
$ ifort -pg precision.f90        instrument code with -pg
$ a.out                          produce gmon.out trace file
$ gprof                          reads gmon.out (default args: a.out gmon.out)
                                 report sent to STDOUT
```

7. Editing Files

vi (short for “visual”)

- “vi filename” will open it or create it if it doesn’t exist.
- Command mode: keystrokes are commands
- Input mode: keystrokes are text you are adding to the file
- Last line mode: start with : end with <return>
- Examples:
  - i Insert characters before current position (use ESC to exit)
  - dd Delete current line
  - R Overwrite existing text (until ESC)
  - u Undo last operation
  - :wq Writes a file to disk and exit editor
  - :q! Quit without saving

http://www.tuxfiles.org/linuxhelp/vimcheat.html

nano

- The commands for all operations are preceded by the Control key:
  - ^G Get Help
  - ^O WriteOut
  - ^X Exit
  - ….
- If you have modified the file and try to exit (^X) without writing those changes (^O) you will be warned.
- Makes text editing simple, but it has less powerful options than vi (search with regular expressions, etc..)

emacs

- emacs is actually a lisp interpreter with extensions to use it as a text editor
- Can perform the same operations as in vi
- Uses Control or ESC followed by keystroke combinations to execute commands
- “Hard to learn, easy to use”

http://emacswiki.org/emacs/ReferenceCards

Use Your Computer's Editor

Copying the file to your computer might be quicker than learning a new editor. Use a simple file transfer client, e.g. FileZilla, or in the lab:

Click on  then enter ***Secure File Transfer Client*** into the search box

- Start Secure File Transfer Client
- Use Quick Connect, specify hostname stampede.tacc.utexas.edu
- In the left pane, navigate to the desktop.
- Drag files between panes to copy.

Warnings

- Beware line ending differences
- Do not use MS Word (e.g.) to create scripts or programs
- Do not copy/paste from PDF files
- Linux filenames are case-sensitive
- Blanks in filenames can be problematic
- Use Wordpad rather than Notepad if necessary
- dos2unix utility can convert text files with DOS or MAC line breaks to Unix line breaks

8. Batch Job Submission: SLURM

Getting to the Compute Nodes

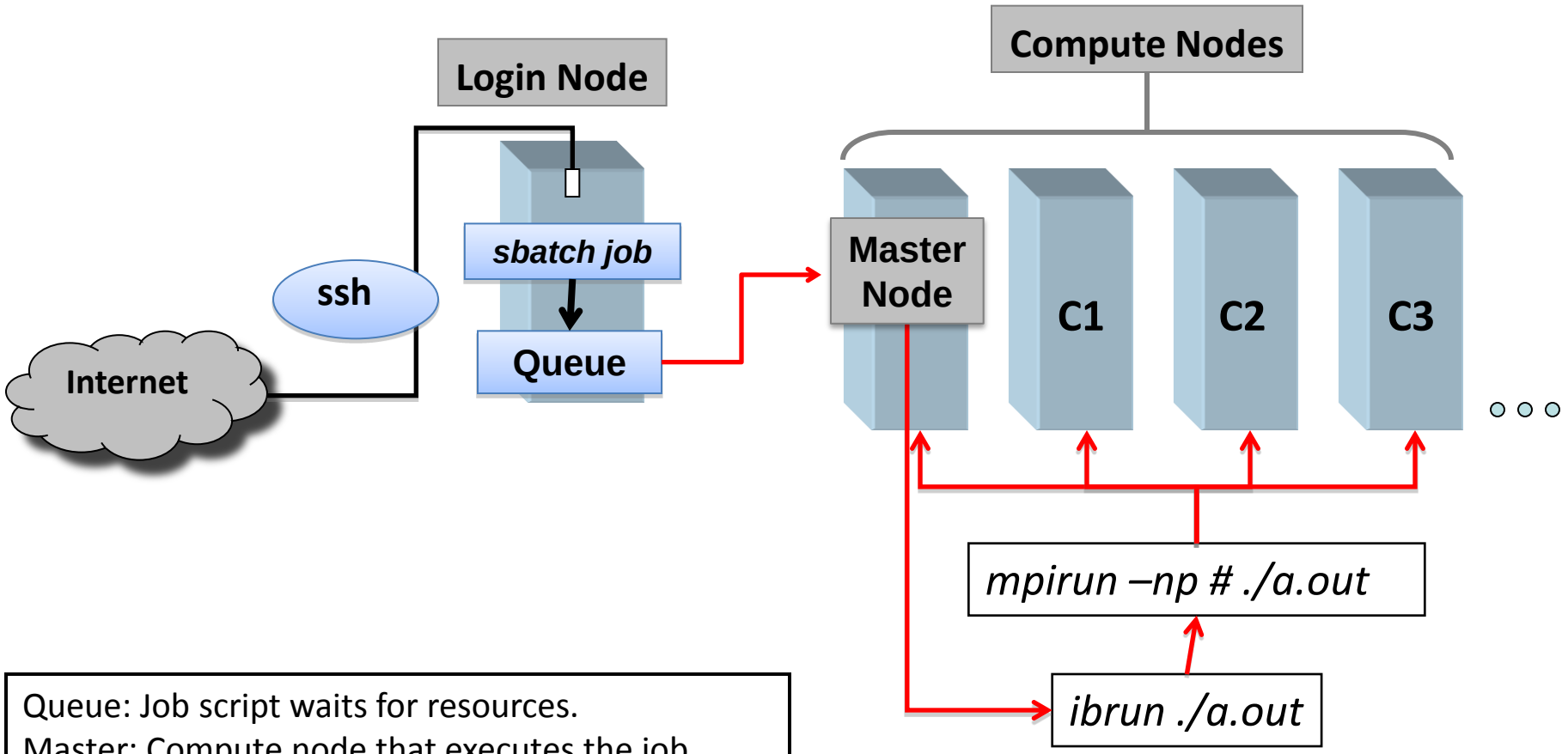
Four ways to get to the back end (compute nodes):

- SLURM batch job: **sbatch <batchfilename>**
- SLURM interactive session: **srun <flags>**
- Run special app that connects to back end: e.g. **ddt**
- ssh to node on which you already have a job running
  - once on compute node, **ssh mic0** gets you to its mic

If you don't use sbatch, srun, or equivalent, you're running on the front end (login nodes) – don't do this!

- Don't launch exe (e.g. **./a.out**) on the command line
- One of the easiest ways to get your account suspended

Batch Submission Process



Queue: Job script waits for resources.
Master: Compute node that executes the job script, launches all MPI processes.

Stampede Batch Environment Queues

Queue Name	Max Runtime	Max Nodes/Procs	SU Charge Rate	Purpose
normal	48 hrs	256 / 4K	1	normal production
development	2 hrs	16 / 256	1	development nodes
largemem	48 hrs	4 / 128	2	large memory 32 cores/node
serial	12 hrs	1 / 16	1	serial/shared_memory
large	24 hrs	1024 / 16K	1	large core counts **
normal-mic	24 hrs	256 / 4k	1	production MIC nodes
gpu	24 hrs	32 / 512	1	GPU nodes
gpudev	4 hrs	4 / 64	1	GPU development nodes
vis	8 hrs	32 / 512	1	GPU nodes + VNC service

http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide#running-slurm-queue

Batch on Stampede: Select SLURM Commands

- **showq** - view summary of jobs in the batch system (not SLURM)
  showq | more
  showq –u <userid>
- **sacct** - report job or job step accounting information.
- **salloc** - allocate resources for a job in real time.
- **sbatch** - submit a job script for later execution.
  sbatch <batchfilename>
- **sbcast** - transfer a file from local disk to local disk on the job nodes.
- **scancel** - cancel a pending or running job or job step.
  scancel <jobid>
- **sinfo** - lists the availability and status of queues
  sinfo -o "%20P %5a %.10l %16F"
- **squeue** - reports the state of jobs or job steps.
  squeue | more
  squeue -u <userid>
- **srun** - submit an interactive job (this example: 1-node 16 core)
  srun --pty -n 16 -t 00:30:00 -p development -A 20130418HPC /bin/bash -l
- **ibrun** – run an MPI program (put this command in your batch script for MPI jobs)

Man pages exist for all SLURM daemons, commands, and API functions. The command option **--help** also provides a brief summary of options. Note that the command options are all case insensitive.

squeue Options, Output, and Job State Codes

-i <interval>	Repeatedly report at intervals (in seconds).
-j <job_list>	Displays information for specified job(s)
-p <part_list>	Displays information for specified partitions (queues).
-t <state_list>	Shows jobs in the specified state(s)

JOBID	job id assigned to the job
USER	user that owns the job
STATE	current job status.

PD	Pending
R	Running
S	Suspended
CA	Configuring
CG	Completing
CD	Completed
CF	Cancelled
F	Failed
TO	Timeout
PR	Preempted
NF	Node_fail

Batch Job Script Example: MPI

```
#!/bin/bash                          # Don't miss this line!

#-------------------------------------------------
# Generic SLURM script -- MPI
#-------------------------------------------------

#SBATCH -J myjob                     # Job name
#SBATCH -o myjob.%j.out              # stdout; %j expands to jobid
#SBATCH -e myjob.%j.err              # stderr; skip to combine stdout and stderr
#SBATCH -p development               # queue
#SBATCH -N 2                         # Number of nodes, not cores (16 cores/node)
#SBATCH -n 32                        # Total number of MPI tasks (if omitted, n=N)
#SBATCH -t 00:30:00                  # max time

#SBATCH --mail-user=myemail@myuniv.edu
#SBATCH --mail-type=ALL

#SBATCH -A TG-TRA120006              # necessary if you have multiple project accounts

module load fftw3                    # You can also load modules before launching job
module list

ibrun ./main.exe                     # Use ibrun for MPI codes. Don't use mpirun or srun.
```

Batch Job Script Example: Serial

```
#!/bin/bash                          # Don't miss this line!

#----------------------------------------------------
# Generic SLURM script
#----------------------------------------------------

#SBATCH -J myjob                     # Job name
#SBATCH -o myjob.%j.out              # stdout; %j expands to jobid
#SBATCH -e myjob.%j.err              # stderr; skip to combine stdout and stderr
#SBATCH -p serial                    # queue
#SBATCH -N 1 -n 1                    # one node and one task
#SBATCH -t 00:30:00                  # max time

#SBATCH --mail-user=myemail@myuniv.edu
#SBATCH --mail-type=ALL

#SBATCH -A TG-01234                  # necessary if you have multiple project accounts

module load fftw3                    # You can also load modules before launching job
module list

./main.exe
```

Batch on Stampede: SLURM Commands

1. Use **sinfo -o "%20P %5a %.10l %16F"** to list queues, nodes, and system state
2. Issue **showq** to show all queued jobs
3. Issue srun to run simple commands (e.g. an interactive shell) (ctrl-D to exit)
   $ srun --pty -A TG-TRA120006 -p serial -t 00:10:00 -n 1 -N 1 /bin/bash -l
4. Issue **cat** to take one last look at the batch script
   $ cd $HOME/envi/batch
   $ cat job

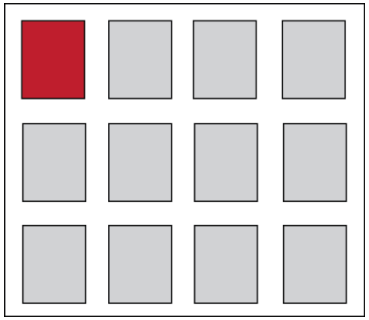
```
#!/bin/bash
#SBATCH -J myMPI                # Job name
#SBATCH -o myjob.%j.out         # stdout file (%j expands to jobId)
#SBATCH -p development          # Queue name
#SBATCH -N 2                    # Total number of nodes requested (16 cores/node)
#SBATCH -n 32                   # Total number of mpi tasks requested
#SBATCH -t 01:30:00             # Run time (hh:mm:ss) - 1.5 hours
#SBATCH –A TG-TRA120006         # Specify account
ibrun ./a.out
```

5. Compile: **mpicc -O3 mpihello.c** –OR- **mpif90 -O3 mpihello.f90**
6. Issue **sbatch** to submit a batch script : **sbatch job**
7. Issue **squeue -u <your username>** to see the job status
8. Run **scancel <jobid>** to cancel the job, or **cat myjob.###.out to** view your output

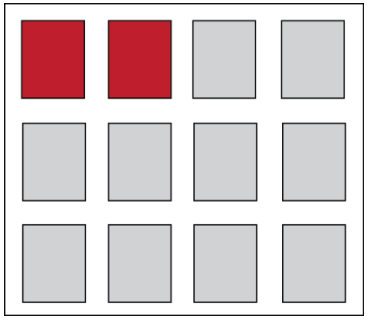
Resource Allocation on SLURM

- ***-N*** **– Number of node requested**
- ***-n*** **– Number of tasks to run**

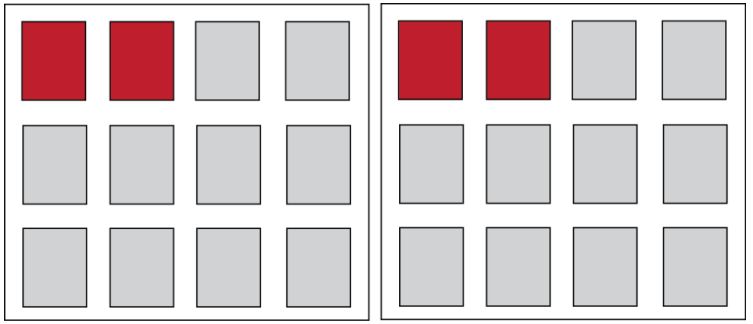
Serial Job
#SBATCH -N 1
#SBATCH -n 1



2 Tasks
#SBATCH -N 1
#SBATCH -n 2



4 Tasks Parallel
#SBATCH -N 2
#SBATCH -n 4



9. Play Nice

- Remember you are sharing resources (file systems, bandwidth)
- Do not run parallel programs on the Login nodes; either it won't run or you'll annoy the sys admins (who may lock your account). Submit a batch job instead.
- Login nodes are for compiling, file management, managing batch jobs, modest post-processing. They are not for heavy computation of any kind.
- I/O should be infrequent, local, and parallel if necessary.

10. Help

Questions?

- CAC help@cac.cornell.edu

- portal.xsede.org -> **Help** (in the navigation bar)
- portal.xsede.org -> My XSEDE -> **Tickets**
- portal.xsede.org -> Documentation -> **Knowledge Base**

- User Guide(s), Usage Policies, etc. and associated links:
  http://www.tacc.utexas.edu/user-services

- Try **man <*command*>** or **man -k <*command*>** or **<*command*> -h** or **<*command*> -help**

Appendix

Precision

The precision program computes and prints *sin(π)*.
The *π* constant uses “E” (double precision) format in one case and “D” (single) in the other.

```
$ cd $HOME/envi/precision
$ cat precision.f90
$ module load intel
$ ifort precision.f90
$ ./a.out
```