

HPC User Environment 2

Yuwu Chen
HPC User Services
LSU HPC & LONI
sys-help@loni.org

Louisiana State University
Baton Rouge
June 26, 2019

Outline

- **Review HPC User Environment 1 topics**
 - Cluster architecture
 - Connect to clusters
 - Software management using ~~softenv~~ and module
- **Things to be covered in this training**
 - Job management
 - Job queue basics
 - Interactive vs Batch jobs
 - Submit and monitor your jobs
 - Understanding Job scheduling
 - Job priority
 - Backfill
 - Compiling and analyze codes on cluster
 - Serial program
 - Parallel program

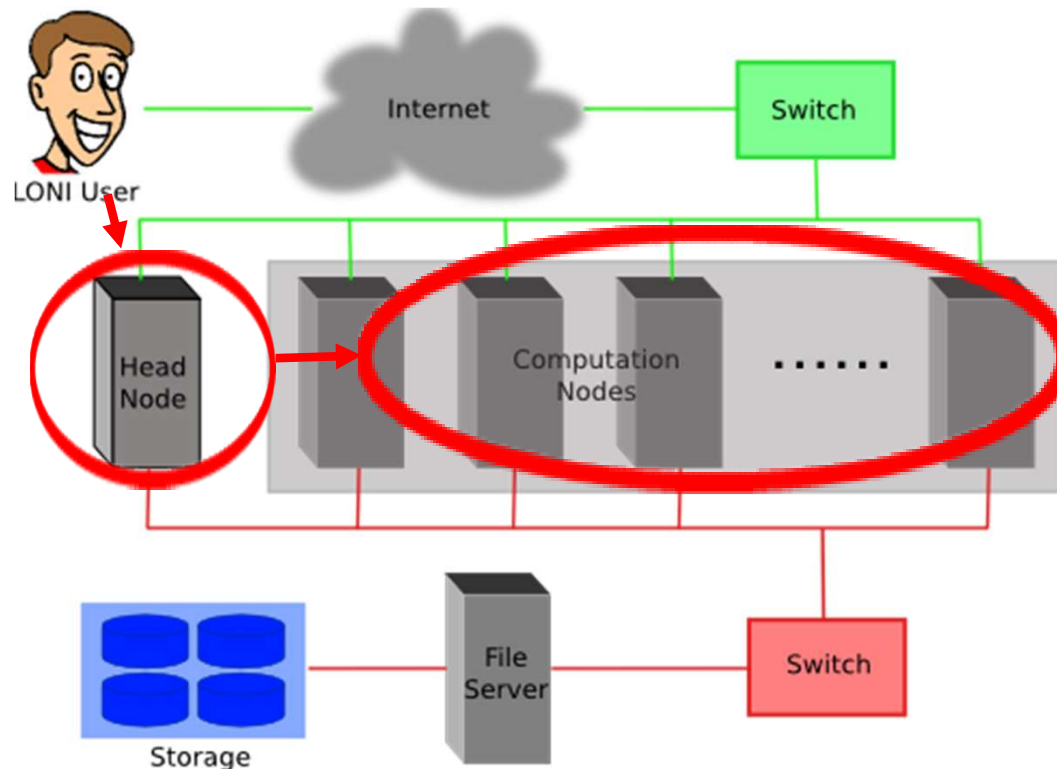


HPC User Environment 2

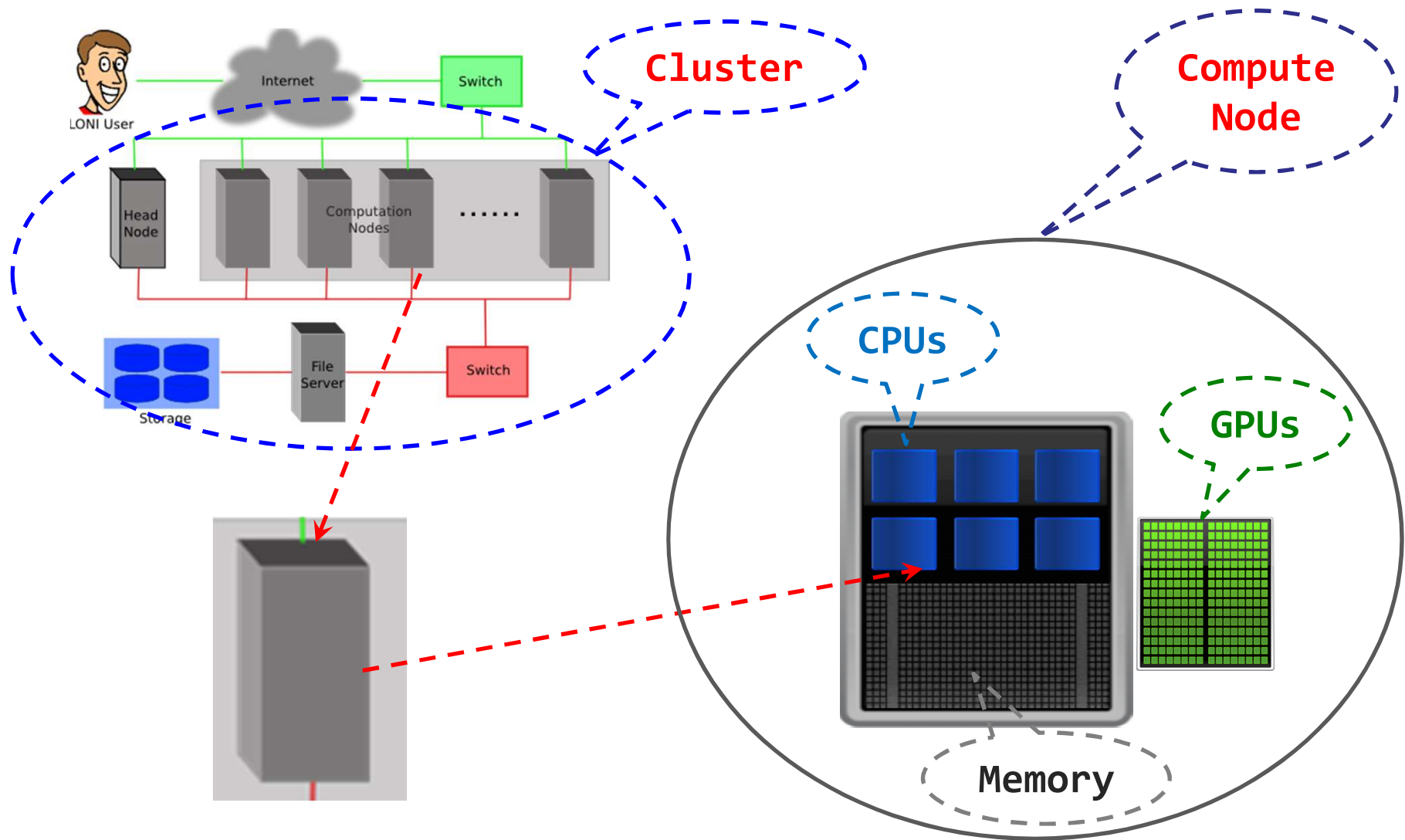
Review of HPC User Environment 1

Cluster Environment

- Multiple compute nodes
- Multiple users
- Each user may have multiple jobs running simultaneously
- Multiple users may share the same node

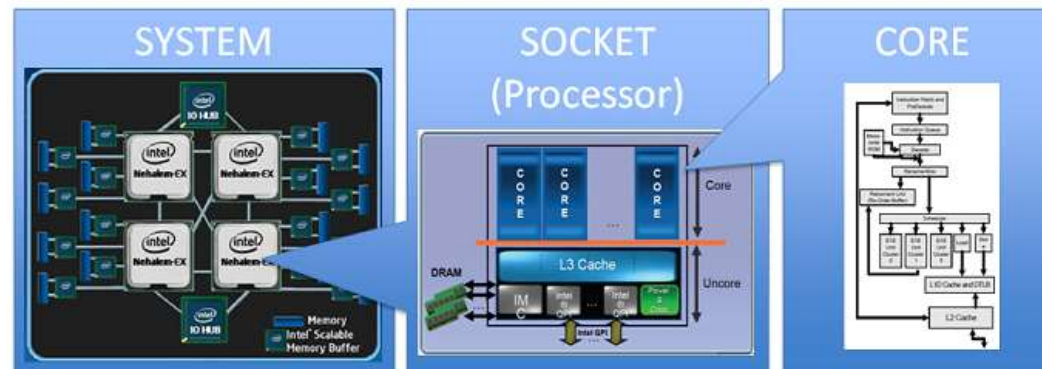


Conceptual Relationship



Cluster Nomenclature

Term	Definition
Cluster	The top-level organizational unit of an HPC cluster, comprising a set of nodes, a queue, and jobs.
Node	A single, named host machine in the cluster.
Core	The basic computation unit of the CPU. For example, a quad-core processor is considered 4 cores.
Job	A user's request to use a certain amount of resources for a certain amount of time on cluster for his work.



Accessing cluster using ssh (Secure Shell)

- **On Unix and Mac**
 - use ssh on a terminal to connect
- **Windows box (ssh client):**
 - MobaXterm (<http://mobaxterm.mobatek.net/>)
 - Putty, Cygwin
(<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>)
- `ssh username@mike.hpc.lsu.edu`
- **Host name**
 - LONI: <cluster_name>.loni.org
 - <cluster_name> can be:
 - `qb.loni.org`
 - LSU HPC: <cluster_name>.hpc.lsu.edu
 - <cluster_name> can be:
 - `mike.hpc.lsu.edu`
 - `smic.hpc.lsu.edu`
 - `philip.hpc.lsu.edu`

Software Management with Environment Modules

- To list all available or part of packages is: `module av`
- To see what packages are currently loaded into a user's environment, the command is: `module list`
- The command for loading a package into a user's environment is: `module load <package name>`. If a specific version of a package is desired, the command can be expanded to: `module load <package name>/<package version>`.
- On HPC and LONI clusters, Modules can be loaded automatically on login by adding the appropriate module load commands to a user's `~/.bashrc` or `~/.modules` (recommended) file

HPC User Environment 2

Review Questions for Section 1

Access to cluster

➤ **How do I connect to HPC/LONI cluster?**

- a) By logging onto HPC webpage at www.hpc.lsu.edu
- b) Using an ssh (secure shell) client such as MobaXterm/Putty ☺
- c) Go to the machine room in ISB in downtown Baton Rouge and connect my laptop to the nodes using a cable

➤ **Login onto Supermike2 or Queenbee2**

`ssh username@mike.hpc.lsu.edu`

`ssh username@qb.loni.org`

Software Management

➤ **How do we manage the software installed on HPC/LONI clusters?**

- a) Using softenv/modules ☺
- b) Using a drop down menu on the www.hpc.lsu.edu webpage

➤ **Check your software environment**

```
[ychen64@mike5 ~]$ module list
```

Currently Loaded Modulefiles:

1) intel/18.0.0 2) INTEL/18.0.0 3) mvapich2/2.2/INTEL-18.0.0

```
[ychen64@qb4 ~]$ module list
```

Currently Loaded Modulefiles:

1) intel/14.0.2 2) INTEL/14.0.2 3) mvapich2/2.0/INTEL-14.0.2

Allocation

- **To run job on the cluster, you must**
 - a) Send your credit card information to the HPC staff
 - b) Make sure your advisor has enough funding for you
 - c) Have an activate allocation 😊
 - d) All of the above

- **List active allocation balance:** [balance](#)

Outline

- **Review HPC User Environment 1 topics**
 - Cluster architecture
 - Connect to clusters
 - Software management using softenv and module
- **Things to be covered in this training**
 - Job management
 - Job queue basics
 - Interactive vs Batch jobs
 - Submit and monitor your jobs
 - Understanding Job scheduling
 - Job priority
 - Backfill
 - Compiling and analyze codes on cluster
 - Serial program
 - Parallel program

HPC User Environment 2

Job Queue Basics

Job submission basics

1. Find appropriate queue

- Understand the queuing system and your requirements and
- Queue Querying

2. Submit job

3. Monitor jobs during execution

Job Queues

- **Nodes are organized into queues. Nodes can be shared.**
- **Each job queue differs in**
 - Number of available nodes
 - Max run time
 - Max running jobs per user
 - Nodes may have special characteristics: GPU/Xeon Phi's, Large memory, etc.
- **Jobs need to specify resource requirements**
 - Nodes, time, queue
- **It is called a queue for a reason, but jobs don't run on a "First Come First Served" policy.**
 - This will be detailed in later slides

Queue Characteristics – LONI clusters

Machine	Queue	Max Runtime	ppn	Max running jobs	Max nodes per job	Use
QB2	workq	3 days	20	64	128	Unpreemptable
	checkpt		20		256	Preemptable
	bigmem		48		1	Big memory
	single	7 days	1,2,4,6,8		1	Single node jobs

Queue Characteristics – LSU Linux clusters

Machine	Queue	Max Runtime	ppn	Max running jobs	Max nodes per job	Use
SuperMike II	workq	3 days	16	44	128	Unpreemptable
	checkpt		16		128	Preemptable
	bigmem	7 days	16		1	Big memory
	gpu	3 days	16		16	Job using GPU
	single	3 days	1,2,4,6,8		1	Single node jobs
SuperMIC	workq	3 days	20	34	128	Unpreemptable
	checkpt		20		200	Preemptable
	hybrid		20		20	Job using GPU
	single		1,2,4,6,8		1	Single node jobs

Queue Characteristics

- “qstat -q” will give you more info on the queues

```
[fchen14@mike2 ~]$ qstat -q
```

```
server: mike3
```

Queue	Memory	CPU Time	Walltime	Node	Run	Que	Lm	State
workq	--	--	72:00:00	128	31	6	--	E R
mwfa	--	--	72:00:00	8	3	0	--	E R
bigmem	--	--	48:00:00	1	0	0	--	E R
lasigma	--	--	72:00:00	28	28	7	--	E R
bigmemtb	--	--	48:00:00	1	0	0	--	E R
priority	--	--	168:00:0	128	0	0	--	E R
single	--	--	72:00:00	1	62	0	--	E R
gpu	--	--	24:00:00	16	1	0	--	E R
preempt	--	--	72:00:00	--	0	0	--	E R
checkpt	--	--	72:00:00	128	31	137	--	E R
admin	--	--	24:00:00	--	0	0	--	E R
scalemp	--	--	24:00:00	1	0	0	--	E R
					156	150		

- For a more detailed description use mdiag

Queue Querying – Linux Clusters

- Displays information about active, eligible, blocked, and/or recently completed jobs: `showq` command

`$ showq`

active jobs-----

JOBID	USERNAME	STATE	PROCS	REMAINING	STARTTIME
236875	ebeigi3	Running	16	1:44:29	Mon Sep 15 20:00:22
236934	mwu3	Running	16	00:03:27	Mon Sep 15 19:04:20

...

eligible jobs-----

JOBID	USERNAME	STATE	PROCS	WCLIMIT	QUEUE TIME
236795	dmarce1	Idle	1456	00:15:00	Mon Sep 15 16:38:45
236753	rsmith	Idle	2000	4:00:00	Mon Sep 15 14:44:52
236862	dlamas1	Idle	576	2:00:00	Mon Sep 15 17:28:57

...

121 eligible jobs

blocked jobs-----

JOBID	USERNAME	STATE	PROCS	WCLIMIT	QUEUE TIME
232741	myagho1	Idle	2000	1:00:00:00	Mon Sep 8 07:22:12
235545	tanping	Idle	1	2:21:10:00	Fri Sep 12 16:50:49
235546	tanping	Idle	1	2:21:10:00	Fri Sep 12 16:50:50

...

Queue Querying – Free nodes

➤ Query free nodes: qfree command

\$ qfree

PBS total nodes: 506, free: 215, busy: 290 *33, down: 1, use: 57%

PBS workq nodes: 476, free: 190, busy: 162, queued: 163

PBS checkpt nodes: 476, free: 190, busy: 124, queued: 284

PBS single nodes: 18, free: 15 *258, busy: 13, queued: 0

PBS k40 nodes: 4, free: 3, busy: 1, queued: 0

(Highest priority job 660266 on queue checkpt will start in 2:27:00)

...

HPC User Environment 2

Submit and Monitor Your Jobs

Two Job Types

➤ Interactive job

- Set up an interactive environment on compute nodes for users
 - Advantage: can run programs interactively
 - Disadvantage: must be present when the job starts
- Purpose: testing and debugging, compiling
 - **DO NOT RUN ON THE HEAD NODE!!!**
 - Try not to run interactive jobs with large core count, which is a waste of resources

➤ Batch job

- Executed without user intervention using a job script
 - Advantage: the system takes care of everything
 - Disadvantage: can only execute one sequence of commands which cannot be changed after submission
- Purpose: production run

Submitting Jobs on Linux Clusters

➤ Interactive job example:

```
qsub -I -X -V \  
    -l walltime=<hh:mm:ss>,nodes=<num_nodes>:ppn=<num_cores> \  
    -A <Allocation> \  
    -q <queue name>
```

**DO NOT directly ssh to compute nodes,
unless the nodes are assigned to you by the job scheduler.**

- Add **-X** to enable X11 forwarding

➤ Batch Job example:

```
qsub qsub.submit
```


Check Your Available Allocations

```
[fchen14@mike2 ~]$ balance
```

```
===== Allocation information for fchen14 =====
  Proj. Name|          Alloc|  Balance| Deposited|    %Used| Days Left|          End
-----
hpc_hpcadmin3|hpc_hpcadmin3 on @mike2|282854.91| 350000.00|    19.18|          16|2017-06-30
hpc_trn17mike2|hpc_trn17mike2 on @mike2| 20305.62| 25000.00|    18.78|          291|2018-04-01
```

Note: Balance and Deposit are measured in CPU-hours

```
[fchen14@mike2 ~]$ showquota
```

Hard disk quotas for user fchen14 (uid 32584):

Filesystem	MB used	quota	files	fquota
/homem	4518	5000	94354	0
/work	424228	0	286002	4000000
/project	65346	100000	1119432	4000000

CPU Allocation SUs remaining:

```
hpc_hpcadmin3: 282854.91
hpc_trn17mike2: 20305.62
```

Submit An Interactive Job on SuperMike2

```
[fchen14@mike1 ~]$ qsub -I -X -l nodes=1:ppn=16,walltime=2:00:00 -q workq -A hpc_trn17mike2
qsub: waiting for job 675733.mike3 to start
qsub: job 675733.mike3 ready
-----
Running PBS prologue script
...
Job ID:      675733.mike3
Username:    fchen14
Group:       Admins
Date:        13-Jun-2017 15:34
Node:        mike044 (62703)
-----
PBS has allocated the following nodes:
mike044
A total of 16 processors on 1 nodes allocated
-----
...
Concluding PBS prologue script - 13-Jun-2017 15:34:19
-----
[fchen14@mike044 ~]$
```

Annotations for the command line:

- Enable X11 forwarding (GUI)**: points to `-X`
- Interactive job**: points to `-I`
- 1 node**: points to `nodes=1`
- 16 cores per node**: points to `ppn=16`
- 2 hour walltime**: points to `walltime=2:00:00`
- submit to workq**: points to `-q workq`
- Allocation name**: points to `-A hpc_trn17mike2`

Submit An Interactive Job on SuperMike2

```
[fchen14@mike1 ~]$ qsub -I -X -l nodes=1:ppn=16,walltime=2:00:00 -q workq -A hpc_trn17mike2
qsub: waiting for job 675733.mike3 to start
qsub: job 675733.mike3 ready
-----
Running PBS prologue script
...
Job ID:      675733.mike3
Username:    fchen14
Group:       Admins
Date:        13-Jun-2017 15:34
Node:        mike044 (62703)
-----
PBS has allocated the following nodes:
mike044
A total of 16 processors on 1 nodes allocated
-----
...
Concluding PBS prologue script - 13-Jun-2017 15:34:19
-----
[fchen14@mike044 ~]$
```

❖ Please pay close attention to the digit change.

Submit An Interactive Job on QB2

```
[ychen64@qb2 ~]$ qsub -I -X -l nodes=1:ppn=20,walltime=01:30:00, -q workq -A loni_train_2018
```

```
qsub: waiting for job 505851.qb3 to start
```

```
qsub: job 505851.qb3 ready
```

```
-----  
Running PBS prologue script  
-----
```

```
User and Job Data:  
-----
```

```
Job ID:      505851.qb3
```

```
Username:    ychen64
```

```
Group:       loniadmin
```

```
Date:        13-Jun-2018 01:27
```

```
Node:        qb061 (4497)  
-----
```

```
PBS has allocated the following nodes:  
...
```

```
-----  
Concluding PBS prologue script - 13-Jun-2018 01:27:39  
-----
```

```
[ychen64@qb061 ~]$
```

20 cores
per node

Allocation
name

Exercise

- **Start an interactive job session for 1 hr 30 min**
 - Find out your allocation name if you don't remember
 - Decide how many node and which queue to use
 - Use “qsub -I”, including all necessary options
 - **Once job started, verify that you are NOT on the head node**
- **Computing an approximate value for PI**
 - cd to your work directory


```
$ cd /work/$USER
```
 - Download the tarball from HPC website to the home directory


```
$ wget http://www.hpc.lsu.edu/training/weekly-materials/2018-Summer/pi.tar.gz
```
 - Untar it


```
$ tar -xvzf pi.tar.gz
```
 - cd to the directory “pi”


```
$ cd pi
```
 - If not using the default compiler, re-compile the source code
 - Execute serialpi.out or mpi_pi.out along with or without a number argument


```
$ serialpi.out #serial version, if no argument given, default value 1000000000
```

```
$ mpirun -np 20 ./mpi_pi.out 100000000000 #mpi version, default 1000000000000
```

During the break...

- **Finish the exercise run.**
- **If you are not familiar with the Linux commands used in the exercise, review the Linux commands cheat sheet in the next slide.**

Cheat sheet of Commands in Linux

- history
- mkdir (name of file) -> makes a folder
- ls -> list
 - a list all files including hidden
 - l shows files with a long listing format
- cd -> change directory
- pwd -> shows location
- cp -> copy
- rm -> Remove files (careful)
- Up arrow (↑) -> moves back in history
- Tab -> fills in unique file name
- Tab Tab -> press tab twice, shows all available file names

Submit An Batch Job

➤ **Batch Job example:**

```
[ychen64@qb2 pi]$ qsub qsub.submit
```

➤ **Batch job cannot be submitted when you are on the compute node**

```
[ychen64@qb023 pi]$ qsub qsub.submit
```

```
qsub: Bad UID for job execution MSG=ruserok failed validating ychen64/ychen64  
from qb023
```

➤ **Carefully prepare the PBS job script**

- examples in the next few slides

PBS Job Script – Serial Job

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

```
#PBS -l nodes=1:ppn=1      # Number of nodes and processor
#PBS -l walltime=24:00:00  # Maximum wall time
#PBS -N myjob              # Job name
#PBS -o <file name>        # File name for standard output
#PBS -e <file name>        # File name for standard error
#PBS -q single             # The queue for serial jobs
#PBS -A <lioni_allocation> # Allocation name
#PBS -m e                  # Send mail when job ends
#PBS -M <email address>    # Send mail to this address
```

Tells the job scheduler how much resource you need.

```
<shell commands>
<path_to_executable> <options>
<shell commands>
```

How will you use the resources?

- Note: don't let your <path_to_executable> <options> be the EOF
 - EOF can be <shell commands>, comments or a blank line.

PBS Job Script – Parallel Job

```
#!/bin/bash
#PBS -l nodes=2:ppn=16      #Number of nodes and processors per node
#PBS -l walltime=24:00:00   #Maximum wall time
#PBS -N myjob               #Job name
#PBS -o <file name>         #File name for standard output
#PBS -e <file name>         #File name for standard error
#PBS -q checkpt             #Queue name
#PBS -A <allocation_if_needed> #Allocation name
#PBS -m e                   #Send mail when job ends
#PBS -M <email address>     #Send mail to this address

<shell commands>
mpirun -machinefile $PBS_NODEFILE -np 32 <path_to_executable> <options>
<shell commands>
```

Tells the scheduler how much resource you need.

How will you use the resources?

- Note: don't let your <path_to_executable> <options> be the EOF
 - EOF can be <shell commands>, comments or a blank line.

True or False?

- I have the below job script on QB2, since I used nodes=2:ppn=20, my script will run in parallel using 2 nodes with 40 cores.
 - a) True
 - b) False

```
#!/bin/bash
#PBS -l nodes=2:ppn=20
#PBS -l walltime=24:00:00
#PBS -N myjob
#PBS -j oe
#PBS -q ckcpt
#PBS -A my_allocation
```

```
./my_executable.out
```

Job Monitoring - Linux Clusters

➤ **Check details on your job using `qstat`**

`$ qstat -n -u $USER` : For quick look at nodes assigned to you

`$ qstat -f jobid` : For details on your job

`$ qdel jobid` : To delete job

➤ **Check approximate start time using `showstart`**

`$ showstart jobid`

➤ **Check details of your job using `checkjob`**

`$ checkjob jobid`

➤ **Check health of your job using `qshow`**

`$ qshow jobid`

❖ **Please pay close attention to the load and the memory consumed by your job!**

Using the “top” command

- The **top** program provides a dynamic real-time view of a running system.

```
top - 19:39:56 up 89 days,  4:13,  1 user,  load average: 0.63, 0.18, 0.06
Tasks: 489 total,   2 running, 487 sleeping,   0 stopped,   0 zombie
Cpu(s):  6.3%us,   0.0%sy,   0.0%ni, 93.7%id,   0.0%wa,   0.0%hi,   0.0%si,   0.0%st
Mem:  65909356k total,  3389616k used, 62519740k free,   151460k buffers
Swap: 207618040k total,    5608k used, 207612432k free,   947716k cached
```

PID	USER	PR	NI	VRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
39595	fchen14	20	0	266m	257m	592	R	99.9	0.4	0:06.94	a.out
39589	fchen14	20	0	17376	1612	980	R	0.5	0.0	0:00.05	top
38479	fchen14	20	0	108m	2156	1348	S	0.0	0.0	0:00.03	bash
39253	fchen14	20	0	103m	1340	1076	S	0.0	0.0	0:00.00	236297.mike3.SC
39254	fchen14	20	0	103m	1324	1060	S	0.0	0.0	0:00.00	bm_laplace.sh
39264	fchen14	20	0	99836	1908	992	S	0.0	0.0	0:00.00	sshd
39265	fchen14	20	0	108m	3056	1496	S	0.0	0.0	0:00.03	bash

Using the “free” command

- The **free** displays the total amount of free and used physical and swap memory in the system

\$ free -h

	total	used	free	shared	buffers	cached
Mem:	62G	3.1G	59G	177M	31M	1.3G
-/+ buffers/cache:		1.7G	61G			
Swap:	127G	0B	127G			

PBS Environmental Variables

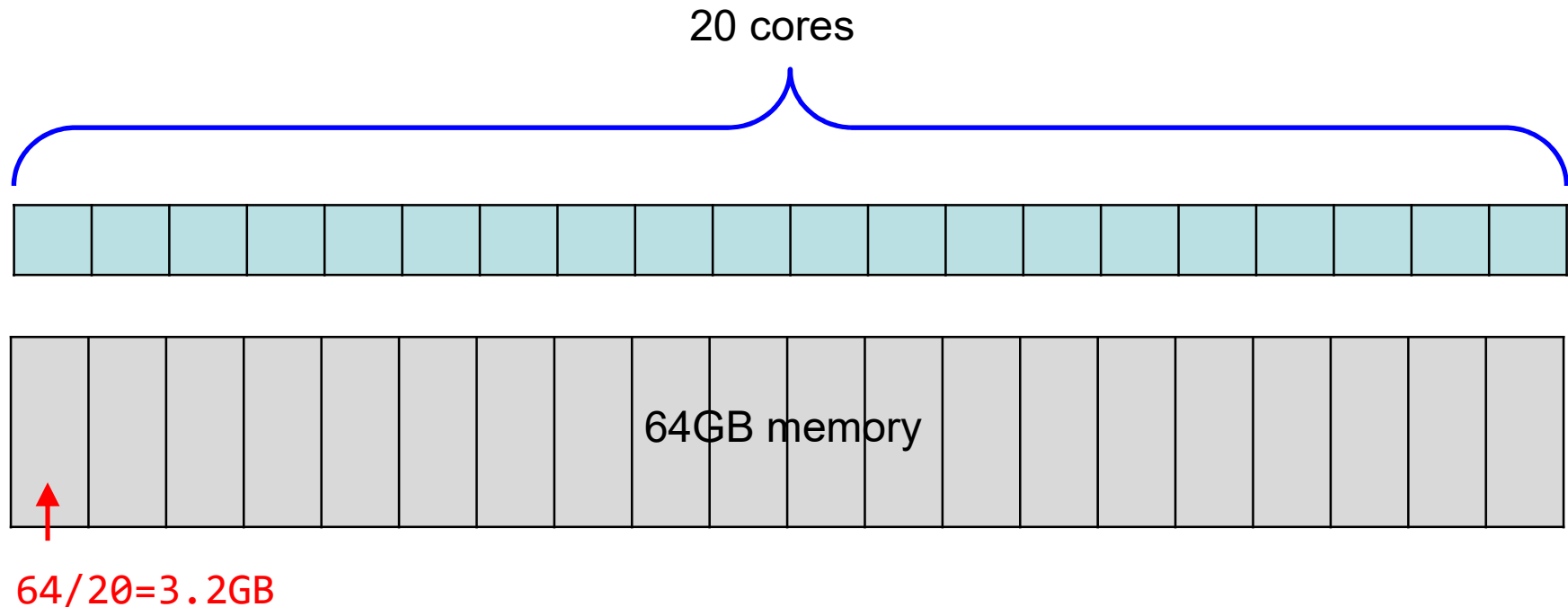
```
[fchen14@mike315 ~]$ echo $PBS_
```

\$PBS_ENVIRONMENT	\$PBS_MOMPORT	\$PBS_NUM_PPN	\$PBS_O_MAIL
\$PBS_QUEUE	\$PBS_WALLTIME	\$PBS_GPUFILE	\$PBS_NODEFILE
\$PBS_O_HOME	\$PBS_O_PATH	\$PBS_SERVER	\$PBS_JOBCOOKIE
\$PBS_NODENUM	\$PBS_O_HOST	\$PBS_O_QUEUE	\$PBS_TASKNUM
\$PBS_JOBID	\$PBS_NP	\$PBS_O_LANG	\$PBS_O_SHELL
\$PBS_VERSION	\$PBS_JOBNAME	\$PBS_NUM_NODES	\$PBS_O_LOGNAME
\$PBS_O_WORKDIR	\$PBS_VNODENUM		

Pay attention to single queue usage

- **Single queue** - Used for jobs that will only execute on a single node, i.e. **nodes=1:ppn=1/2/4/6/8**.
- **Jobs in the single queue should not use:**
 - More than 2GB memory per core for Eric, Philip and SuperMike2 (32G/16).
 - More than 3.2GB memory per core for QB2 (64G/20).
- **If applications require more memory, scale the number of cores (ppn) to the amount of memory required: i.e. max memory available for jobs in single queue is 8GB for ppn=4 on SuperMikell.**
- **Typical type of warning:**
 - E124 - **Exceeded memory allocation**. This Job XXXX appears to be using more memory (GB) than allocated (9 > 3).
 - E123 - **Exceeded ppn/core allocation**. This Job XXXX appears to be using more cores than allocated (6 > 1). Please allocate the number of cores that the job will use, (ppn=6). This Job has 1 core(s) allocated (ppn=1).

Core and Memory in Single queue



Question:

On QB2, if my job needs 7GB memory, what ppn value should I use?

On SuperMike2, if my job needs 7GB memory, what ppn value should I use?

More things to be noticed

- The purpose of bigmem queue on QB-2 is for jobs costing big (larger than 64 GB) memory not for jobs using more number of cores.
- GPU is available to workq or checkpt queues on QB-2.
- Xeon Phi is available to workq or checkpt queues on SuperMIC.
- Users are encouraged to use accelerators (GPU/Xeon Phi) whenever possible. Application for allocation involving with usage of accelerators will be easier to be approved.

Job Submission Quiz

- **How to suspend your account? (cont'd)**
 - Use more memory than allowed. (e.g. use 5GB memory on SuperMike2 with ppn=1)
 - Seriously underutilize node resources (e.g. allocate 32 nodes but just use 1 core)
 - Submit job to the big memory queue but use only few MB of memory

- **How to monitor core and memory usage?**

Outline

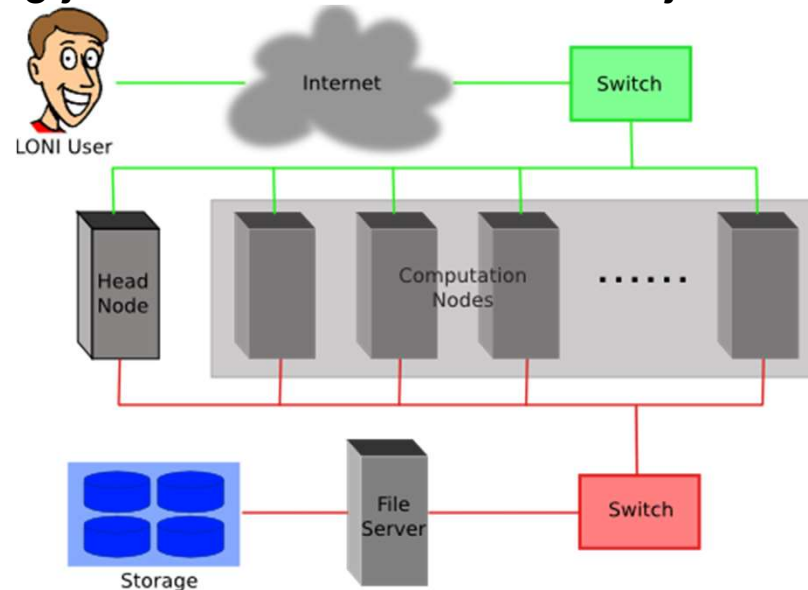
- **Review HPC User Environment 1 topics**
 - Cluster architecture
 - Connect to clusters
 - Software management using softenv and module
- **Things to be covered in this training**
 - Job management
 - Job queue basics
 - Interactive vs Batch jobs
 - Submit and monitor your jobs
 - Understanding Job scheduling
 - Job priority
 - Backfill
 - Compiling and analyze codes on cluster
 - Serial program
 - Parallel program

HPC User Environment 2

Job Scheduling Basics

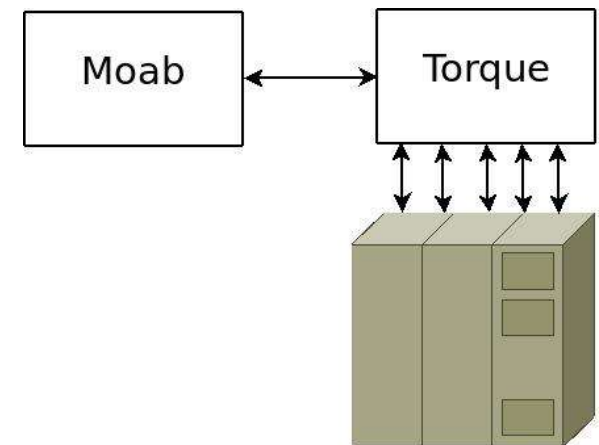
Back to Cluster Architecture

- **As a user, you interact with the scheduler and/or resource manager whenever you submit a job, or query on the status of your jobs or the whole cluster, or seek to manage your jobs.**
- **Resource managers give access to compute resource**
 - Takes in a resource request (job) on login node
 - Finds appropriate resource and assigns you a priority number
 - Positions your job in a queue based on the priority assigned.
 - Starts running jobs until it cannot run more jobs with what is available.



Job Scheduler

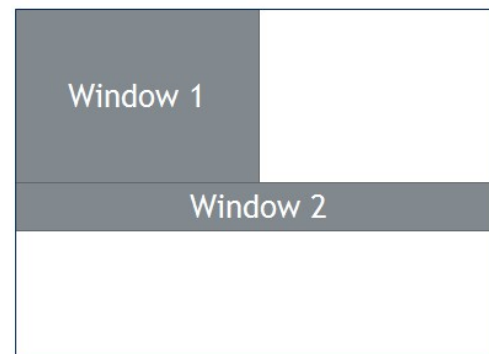
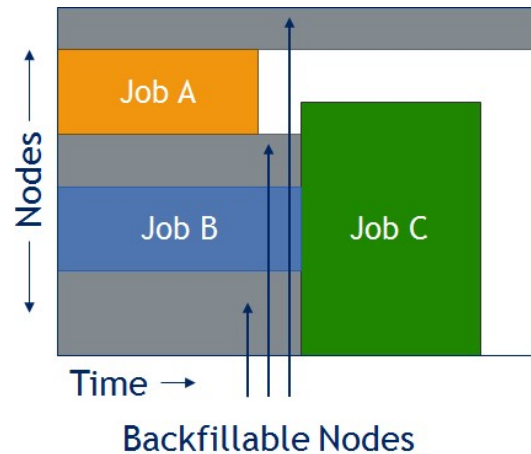
- **HPC & LONI Linux clusters use TORQUE, an open source version of the **Portable Batch System (PBS)** together with the MOAB Scheduler, to manage user jobs.**
- **Resource Manager - **Torque****
 - Manages a queue of jobs for a cluster of resources
 - Launches job to a simple FIFO job queue
- **Workload Manager - **Moab****
 - A scheduler that integrates with one or more Resource Managers to schedule jobs across domains of resources (servers, storage, applications)
 - Prioritizes jobs
 - Provides status of running and queued jobs, etc.
- **The batch queuing system determines**
 - The order jobs are executed
 - On which node(s) jobs are executed



Job management philosophy

➤ Working Philosophy

- Prioritize workload into a queue for jobs
- **Backfill** idle nodes to maximize utilization
 - Will be detailed later...



Job Priorities

- **Jobs with a higher job priority are scheduled ahead of jobs with a lower priority.**
- **Job priorities have contributions from the following:**
 - credential priority
 - fairshare priority
 - service priority
 - resource priority
- **Priority determination for each queued job, use**
 - `mdiag -p:`

```
$ mdiag -p
```

```
diagnosing job priority information (partition: ALL)
```

Job	PRIORITY*	Cred(User:Class)	FS(User: WCA)	Serv(QTime:XFctr)	Res(Proc)
Weights	-----	100(10: 10)	100(10: 50)	2(2: 20)	30(10)
236172	246376	40.6(100.0: 0.0)	8.6(19.6: 0.3)	4.0(1480.: 99.7)	46.8(2048.)
235440	242365	41.3(100.0: 0.0)	4.6(8.2: 0.6)	6.6(3959.: 6.5)	47.5(512.0)
235441	242365	41.3(100.0: 0.0)	4.6(8.2: 0.6)	6.6(3959.: 6.5)	47.5(512.0)
235442	242361	41.3(100.0: 0.0)	4.6(8.2: 0.6)	6.6(3958.: 6.5)	47.5(512.0)
236396	241821	41.4(100.0: 0.0)	8.8(19.6: 0.3)	2.2(664.0: 67.4)	47.6(1456.)

Priority components

- **Credential priority** = credweight * (userweight * job.user.priority)
= 100 * (10 * 100) = 100000

It is a **constant** for all users.

- **Fairshare priority** = fsweight * min (fscap, (fsuserweight * DeltaUserFSUsage))
= 100 * (10 * **DeltaUserFSUsage**)

If you have not submitted jobs in the past 7 days, **DeltaUserFSUsage** = 20000

- **Service priority** = serviceweight * (queuetimeweight * QUEUETIME +
xfactorweight * XFACTOR)
= 2 * (2 * **QUEUETIME** + 20 * XFACTOR),

where XFACTOR = 1 + **QUEUETIME / WALLTIMELIMIT**.

- **Resource priority** = resweight * min (rescap, (procweight *
TotalProcessorsRequested)
= 30 * min (3840, (10 * **TotalProcessorsRequested**))

- See <http://www.hpc.lsu.edu/docs/pbs.php> , click “Job priority”.

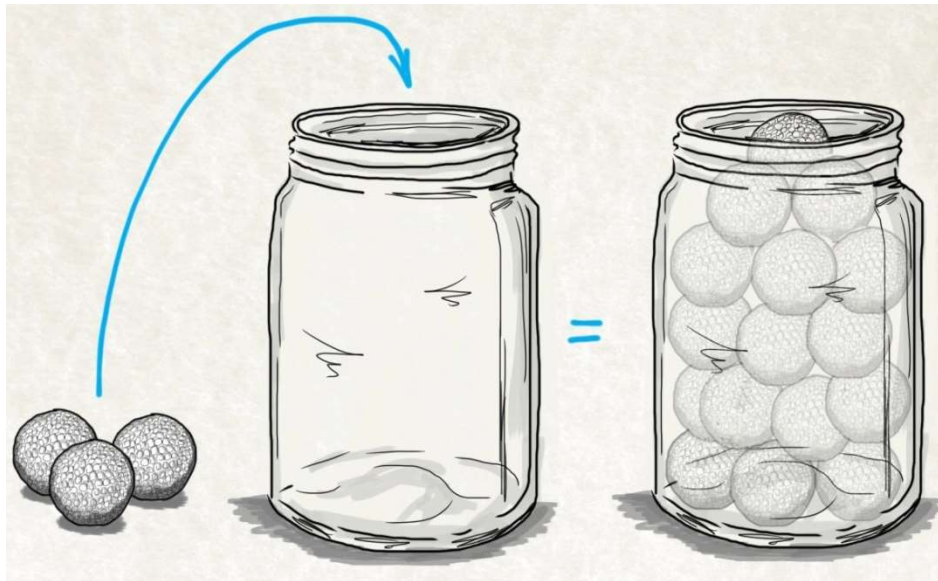
How to get higher priority?

- Do not submit too many jobs within one week.
- Submit your job early to accumulate the queue time.
- **More on resource priority:**
 - Request more compute nodes.
 - Request a smaller walltime limit.
 - see next few slides...

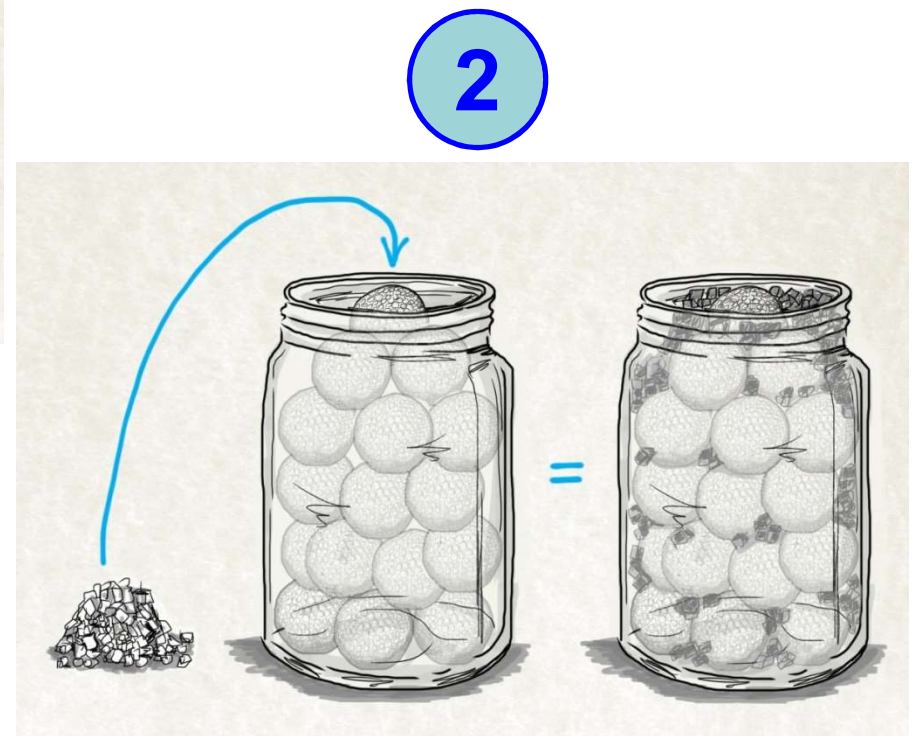


How to maximize the usage of a cluster?

- Fill in high-priority (large) jobs
- Backfill low-priority (small) jobs



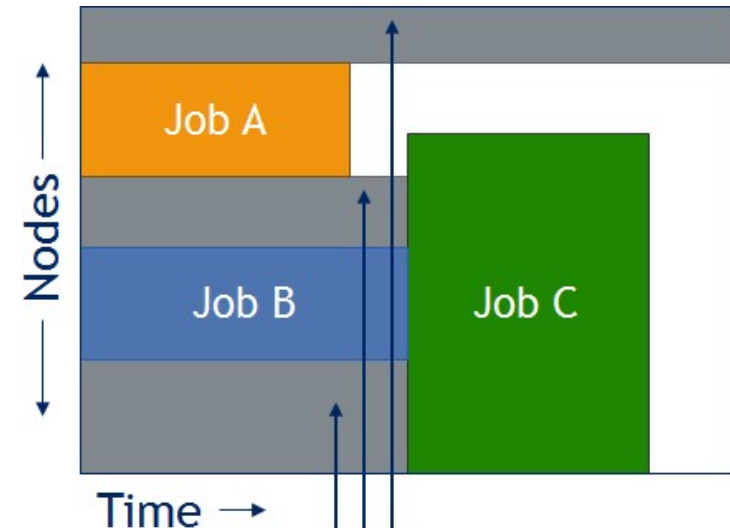
1



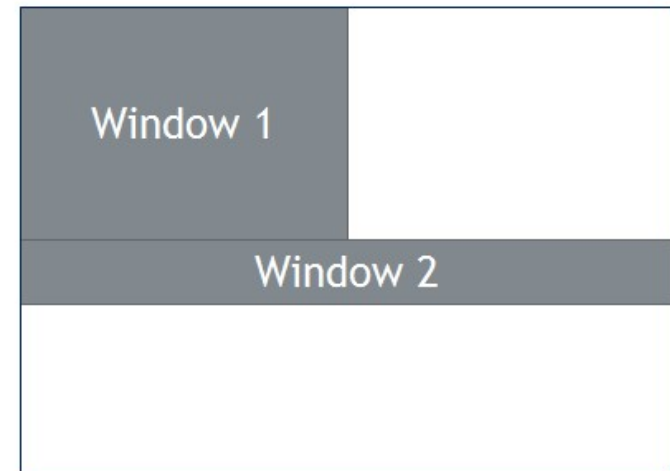
2

An Overview of Backfilling (1)

- **Backfill** is a scheduling optimization that allows a scheduler to make better use of available resources by running jobs out of order.
- **Enabling backfill** allows the scheduler to start other, lower-priority jobs so long as they do not delay the highest priority job.
- **If the FIRSTFIT algorithm is applied, the following steps are taken:**
 - The list of feasible backfill jobs is filtered, selecting only those that will actually fit in the current backfill window.
 - The first job is started.
 - While backfill jobs and idle resources remain, repeat step 1.



Backfillable Nodes



An Overview of Backfilling (2)

- Although by default the start time of the highest priority job is protected by a reservation, there is nothing to prevent the third priority job from starting early and possibly delaying the start of the second priority job.
- Command to show current backfill windows:
 - `showbf`
 - Shows what resources are available for immediate use.
 - This command can be used by any user to find out how many processors are available for immediate use on the system. It is anticipated that users will use this information to submit jobs that meet these criteria and thus obtain quick job turnaround times.
 - Example:

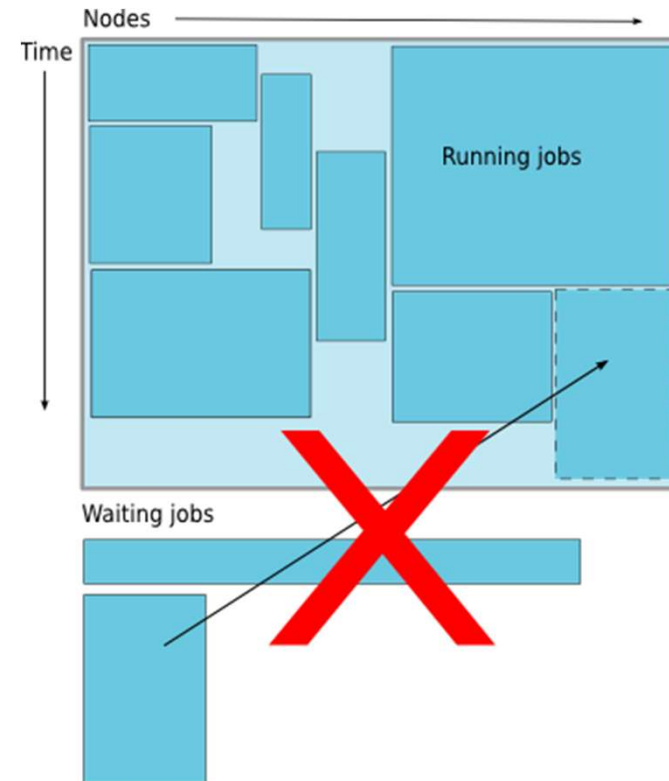
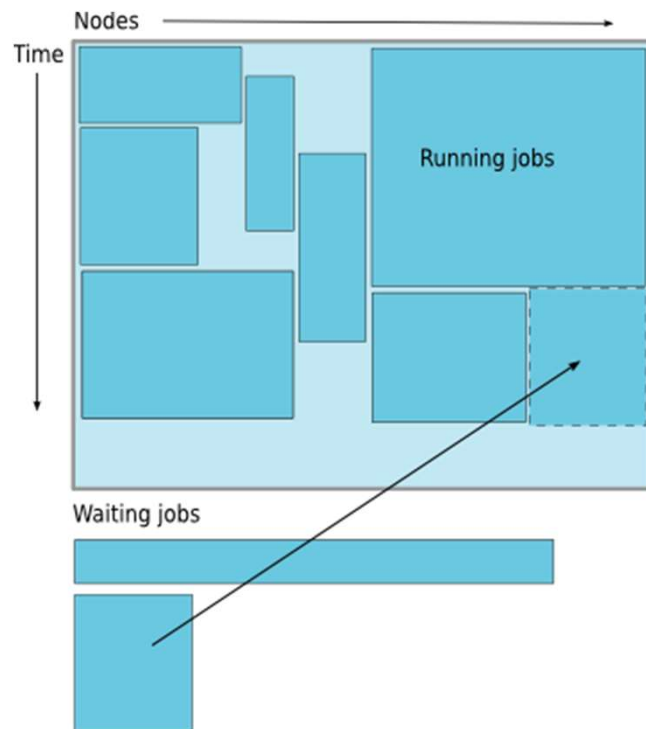
```
[fchen14@eric2 ~]$ showbf -c workq
```

Partition	Tasks	Nodes	Duration	StartOffset	StartDate
-----	-----	-----	-----	-----	-----
ALL	40	5	18:50:35	00:00:00	11:16:49_09/04
ALL	8	1	INFINITY	00:00:00	11:16:49_09/04

How Much Time Should I Ask for?

➤ It should be

- Long enough for your job to complete
- As short as possible to increase the chance of backfilling



Frequently Asked Questions

- **I submitted job A before job B. Why job B started earlier than job A?**
- **There are free nodes available, why my job is still waiting and not running?**
- **Why my job is not get accelerated when running on cluster?**
 - Is your job utilizing the parallel resource on the cluster?
 - Does you job have lots of I/O tasks?

Take-home message

- **All words in red font are very important, CAPITALIZED ARE EVEN MORE IMPORTANT!**
- **Find appropriate queue**
 - Nodes are organized into queues. Nodes can be shared.
 - Nodes may have special characteristics: GPU, Large memory, etc
- **Submit job**
 - interactively, use “-I” option in the qsub command
 - in batch with a carefully-prepared PBS script
- **How to monitor core and memory usage**
 - on the headnode: qstat, qshow etc
 - on the compute node: top command
- **Job schedule basics**
 - The batch queuing system determines 1) the order jobs are executed
2) on which node(s) jobs are executed
 - Jobs with a higher job priority are scheduled ahead of jobs with a lower priority.
 - How to get higher priority

Future Trainings

- **Next week training: Basic Shell Scripting**
 - Wednesdays 9:00am, July 10, Frey Computing Service Center 307
- **Programming/Parallel Programming workshops**
 - Usually in summer
- **Keep an eye on our webpage: www.hpc.lsu.edu**

HPC@LSU User Services

- **Hardware resources**
 - Currently manages 4 clusters
- **Software stack**
 - Communication software
 - Programming support: compilers and libraries
 - Application software
- **Contact user services**
 - Email Help Ticket: sys-help@loni.org
 - Telephone Help Desk: +1 (225) 578-0900

HPC User Environment 2

Compile and Analyze Codes on Cluster

Compilers

➤ Serial compilers

Language	Linux cluster		
	Intel	PGI	GNU
Fortran	ifort	pgf77, pgf90	gfortran
C	icc	pgcc	gcc
C++	icpc	pgCC	g++

➤ Parallel compilers

Language	Linux clusters
Fortran	mpif77, mpif90
C	mpicc
C++	mpiCC

Example compiling serial code

- **icc serialpi.c**
- **gfortran test_hello2.f90**

- **List symbols for executables:**

`nm - list symbols from object files`

- **Example:**

```
[fchen14@mike2 hello]$ nm ./a.out | grep intel
000000000060eb60 B __intel_cpu_indicator
```

```
[fchen14@mike2 hello]$ nm ./a.out | grep gfortran
U _gfortran_set_args@@GFORTRAN_1.0
```

CPU time vs Elapsed time

➤ **CPU time (or process time):**

- The amount of time for which a central processing unit (CPU) was used for processing instructions of a computer program or operating system, as opposed to, for example, waiting for input/output (I/O) operations or entering low-power (idle) mode.

➤ **Elapsed real time (or simply real time, or wall clock time)**

- The time taken from the start of a computer program until the end as measured by an ordinary clock. Elapsed real time includes I/O time and all other types of waits incurred by the program.

➤ **If a program uses parallel processing, total CPU time for that program would be more than its elapsed real time.**

- $(\text{Total CPU time}) / (\text{Number of CPUs})$ would be same as elapsed real time if work load is evenly distributed on each CPU and no wait is involved for I/O or other resources.

Compiling and Analyzing C serial program

```
#include <stdio.h>
#include <time.h>
int main(char *argc, char **argv) {
    double s=0.0;
    // fundamental arithmetic type representing clock tick counts.
    clock_t start, end;
    int i;
    start = clock();
    for (i=0;i<1000000000;i++)
        s+=i*2.0; // doing some floating point operations
    end = clock();
    double time_elapsed_in_seconds = (end - start)/(double)CLOCKS_PER_SEC;
    printf("cputime_in_sec: %e\n", time_elapsed_in_seconds);
    start = clock();
    system ("sleep 5"); // just sleep, does this accumulate CPU time?
    end = clock();
    time_elapsed_in_seconds = (end - start)/(double)CLOCKS_PER_SEC;
    printf("cputime_in_sec: %e\n", time_elapsed_in_seconds);
    return 0;
}
```


Watch the actual cpu time using “time”

```
[fchen14@mike429 serial]$ gcc hello_cpu_elapsed.c
```

```
[fchen14@mike429 serial]$ time ./a.out
```

```
cputime_in_sec: 2.740000e+00
```

```
cputime_in_sec: 0.000000e+00
```

```
real    0m7.782s
```

```
user    0m2.750s
```

```
sys     0m0.005s
```

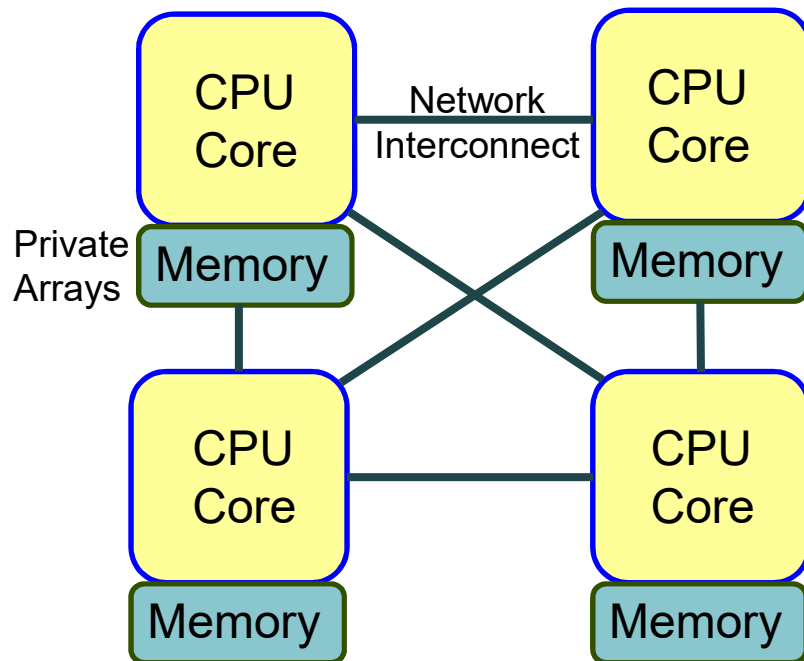
Some additional info about “time”

- **Use the Linux command `time` to evaluate the actual time usage**
 - time a simple command or give resource usage
- **Real refers to actual elapsed time (wall clock time)**
 - Time from start to finish of the call. This is all elapsed time including time used by other processes and time the process spends blocked (for example if it is waiting for I/O to complete).
- **User and Sys refer to CPU time used only by the process.**
 - User is the amount of CPU time spent in user-mode code (*outside the kernel*) within the process.
 - Sys is the amount of CPU time spent *in the kernel* within the process.
- **Purpose of this example:**
 - $\text{real} < \text{user}$: The process is CPU bound and takes advantage of parallel execution on multiple cores/CPU's.
 - $\text{real} \approx \text{user}$: The process is CPU bound and takes no advantage of parallel execution.
 - $\text{real} > \text{user}$: The process is I/O bound. Execution on multiple cores would be of little to no advantage.

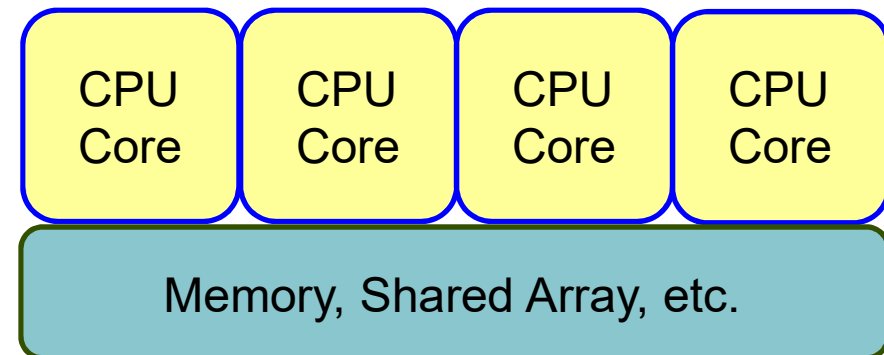
Two parallel schemes

- **Shared Memory system**
 - A single multicore compute node
 - Open Multi-processing (OpenMP)
- **Distributed Memory system**
 - Multiple compute nodes
 - Message Passing Interface (MPI)

MPI: Distributed Memory System



OpenMP: Shared Memory System



Typically less memory overhead/duplication.
Communication often implicit, through cache
coherency and runtime.

Example compiling threaded OpenMP code

- **Compiling OpenMP code often requires the openmp compiler flags, it varies with different compiler**
- **Environment Variable OMP_NUM_THREADS sets the number of threads**
- **Examples:**

```
[fchen14@mike2 src]$ gcc -fopenmp hello_openmp.c
```

```
[fchen14@mike2 src]$ ifort -openmp hello_openmp.f90
```

Compiler	Compiler Options	Default behavior for # of threads (OMP_NUM_THREADS not set)
GNU (gcc, g++, gfortran)	-fopenmp	as many threads as available cores
Intel (icc ifort)	-openmp	as many threads as available cores
Portland Group (pgcc,pgCC,pgf77,pgf90)	-mp	one thread

Sample OpenMP C code

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]) {
    int nthreads, tid;
    /* Fork a team of threads with their own copies of variables */
#pragma omp parallel private(nthreads, tid)
    {
        /* Obtain thread number */
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);
        /* Only master thread does this */
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }
    } /* All threads join master thread and disband */
}
```

Sample OpenMP Fortran code

```

program hello

integer nthreads,tid,omp_get_num_threads,omp_get_thread_num
! fork a team of threads giving them their own copies of variables
!$omp parallel private(nthreads, tid)
! obtain thread number
tid = omp_get_thread_num()
print *, 'hello world from thread = ', tid
! only master thread does this
if (tid .eq. 0) then
    nthreads = omp_get_num_threads()
    print *, 'number of threads = ', nthreads
end if
! all threads join master thread and disband
!$omp end parallel
end
    
```

Analyzing a parallel (OpenMP) program

- What will be the CPU time and elapsed time for the following code segment:

See (on SuperMike II):

/home/fchen14/userenv/src/openmp/hello_openmp_cpu_elapse.c

// fundamental arithmetic type representing clock tick counts.

clock_t start, end;

struct timeval r_start, r_end;

int i;

gettimeofday(&r_start, NULL);

start = clock();

#pragma omp parallel for // spawn the openmp threads

for (i=0;i<N;i++) a = i*2.0; // doing some floating point operations

end = clock();

gettimeofday(&r_end, NULL);

double cputime_elapsed_in_seconds = (end - start)/(double)CLOCKS_PER_SEC;

double realtime_elapsed_in_seconds = ((r_end.tv_sec * 1000000 + r_end.tv_usec) - (r_start.tv_sec * 1000000 + r_start.tv_usec))/1000000.0;

Available MPI libraries on LONI & HPC

	Name	MPI Library				Default serial compiler
Cluster Resource		Mvapich	Mvapich2	OpenMPI	MPICH	
LONI	QB2	X	2.0	1.8.1	3.0.3	Intel 14.0.2
LSU	SuperMikell	X	1.9, 2.0.1	1.6.2 1.6.3 1.6.5	3.0.2	Intel 13.0.0
	Philip	X	X	1.4.3, 1.6.1	1.2.7, 1.3.2, 1.4.1	Intel 11.1
	SuperMIC	X	2.0	1.8.1	3.0.3 3.1.1	Intel 14.0.2

MPI Compilers (1)

Language	Linux clusters	AIX clusters
Fortran	mpif77, mpif90	mpxlf, mpxlf90
C	mpicc	mpcc
C++	mpiCC	mpCC

mpif90 hello.f90

mpicc hello.c

mpicxx hello.cpp

MPI Compilers (2)

- **These MPI compilers are actually wrappers**
 - They still use the compilers we've seen on the previous slide
 - Intel, PGI or GNU
 - They take care of everything we need to build MPI codes
 - Head files, libraries etc.
 - What they actually do can be reveal by the `-show` option
- **It's extremely important that you compile and run your code with the same version of MPI!**
 - Use the default version if possible

Compiling a MPI C program

➤ Compiling Hello world in C version:

– mpicc hello_mpi.c

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
    int name_len, world_size, world_rank;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    //Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Get the number and rank of processes
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    // Get the name of the processor
    MPI_Get_processor_name(processor_name, &name_len);
    // Print off a hello world message
    printf("Iam from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);
    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Compiling a MPI Fortran program

➤ Compiling Hello world in Fortran:

– mpif90 hellp_mpi.f90

```
program hello_mpi
  include 'mpif.h'
  !use mpi
  character 10 name
  ! Initialize the MPI library:
  call MPI_Init(ierr)
  ! Get size and rank
  call MPI_Comm_Size(MPI_COMM_WORLD, numtasks, ierr)
  call MPI_Comm_Rank(MPI_COMM_WORLD, rank, ierr)
  ! print date
  if (nrank == 0) then
    write( , )'System date'
    call system('date')
  endif
  call MPI_Barrier(MPI_COMM_WORLD, ierr)
  ! print rank
  call MPI_Get_Processor_Name(name, len, ierr)
  write( , )"I am ", nrank, "of", numtasks, "on ", name
  ! Tell the MPI library to release all resources it is using:
  call MPI_Finalize(ierr)
end program hello_mpi
```

Notes for compiling a MPI program (1)

- **Always verify what compiler/library is being used:**

```
$ mpicc -show
icc -I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/include -
L/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib -lmpi -ldl -lm -
Wl,--export-dynamic -lrt -lnsl -libverbs -libumad -lpthread -lutil
```

```
$ mpif90 -show
ifort -I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/include -
I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib -
L/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib -lmpi_f90 -
lmpi_f77 -lmpi -ldl -lm -Wl,--export-dynamic -lrt -lnsl -libverbs -
libumad -lpthread -lutil
```

Notes for compiling a MPI program (2)

➤ **Always verify what library is being used: Before and after:**

```
$ ldd a.out #ldd - print shared library dependencies
```

```
linux-vdso.so.1 => (0x00007fff907ff000)
```

```
libmpi_f90.so.1 => /usr/local/packages/openmpi/1.6.2/Intel-  
13.0.0/lib/libmpi_f90.so.1 (0x00002b9ae577e000)
```

```
libmpi_f77.so.1 => /usr/local/packages/openmpi/1.6.2/Intel-  
13.0.0/lib/libmpi_f77.so.1 (0x00002b9ae5982000)
```

```
libmpi.so.1 => /usr/local/packages/openmpi/1.6.2/Intel-  
13.0.0/lib/libmpi.so.1 (0x00002b9ae5bb9000)
```

...

```
libpthread.so.0 => /lib64/libpthread.so.0 (0x0000003b21800000)
```

...

```
libifport.so.5 =>  
/usr/local/compilers/Intel/composer_xe_2013.0.079/compiler/lib/intel64/l  
ibifport.so.5 (0x00002b9ae61ee000)
```

```
libifcore.so.5 =>  
/usr/local/compilers/Intel/composer_xe_2013.0.079/compiler/lib/intel64/l  
ibifcore.so.5 (0x00002b9ae641d000)
```

Running and Analyzing MPI program

- **Make sure you are running your jobs on the correct nodes**
- **Important if you want to run less processes than ppn**
- **Understand the usage of \$PBS_NODEFILE**

```
[fchen14@mike2 ~]$ qsub -I -X -l nodes=2:ppn=16 -l walltime=01:00:00 -q gpu
...
[fchen14@mike429 ~]$ echo $PBS_NODEFILE
/var/spool/torque/aux//236660.mike3
[fchen14@mike429 ~]$ cat $PBS_NODEFILE
mike429
...
mike429
mike430
...
mike430
# 16 repeats of mike429
# 16 repeats of mike430
[fchen14@mike429 hybrid]$ cat $PBS_NODEFILE | uniq > hosts
[fchen14@mike429 hybrid]$ cat hosts
mike429
mike430
```

Running and Analyzing MPI program

```
[fchen14@mike315 mpi]$ mpicc hello_mpi.c
[fchen14@mike315 mpi]$ mpirun -np 32 -hostfile $PBSNODEFILE ./a.out
Iam from processor mike315, rank 1 out of 32 processors
Iam from processor mike315, rank 6 out of 32 processors
Iam from processor mike315, rank 9 out of 32 processors
Iam from processor mike315, rank 12 out of 32 processors
Iam from processor mike315, rank 0 out of 32 processors
Iam from processor mike315, rank 2 out of 32 processors
Iam from processor mike315, rank 3 out of 32 processors
Iam from processor mike315, rank 7 out of 32 processors
Iam from processor mike315, rank 10 out of 32 processors
Iam from processor mike315, rank 5 out of 32 processors
Iam from processor mike315, rank 13 out of 32 processors
Iam from processor mike315, rank 4 out of 32 processors
Iam from processor mike315, rank 8 out of 32 processors
Iam from processor mike334, rank 17 out of 32 processors
Iam from processor mike315, rank 11 out of 32 processors
Iam from processor mike315, rank 14 out of 32 processors
Iam from processor mike315, rank 15 out of 32 processors
Iam from processor mike334, rank 18 out of 32 processors
```


Compiling hybrid (MPI+OpenMP) program

- **See** /home/fchen14/userenv/src/hybrid/hello_hybrid.c **for complete source**
- **Use command:**
 - \$ mpicc -openmp hello_hybrid.c

```
#pragma omp parallel default(shared) private(itd, np)
{
    gtd = omp_get_num_threads(); //get total num of threads in a process
    itd = omp_get_thread_num(); // get thread id
    gid = nrank*gtd + itd;        // global id
    printf("Gid %d from thd %d out of %d from process %d out of %d on %s\n",
           gid, itd, gtd, nrank, numprocs, processor_name);
    if (nrank==0 && itd==0)
    {
        // system("pstree -ap -u $USER");
        system("for f in `cat $PBS_NODEFILE|uniq`; do ssh $f pstree -ap -u
$USER; done;");
        system("sleep 10");
    }
}
```

Analyzing a hybrid program

```
[fchen14@mike315 hybrid]$ export OMP_NUM_THREADS=4
[fchen14@mike315 hybrid]$ mpirun -np 2 -x OMP_NUM_THREADS ./a.out
Gid 0 from thread 0 out of 4 from process 0 out of 2 on mike315
Gid 2 from thread 2 out of 4 from process 0 out of 2 on mike315
Gid 1 from thread 1 out of 4 from process 0 out of 2 on mike315
Gid 3 from thread 3 out of 4 from process 0 out of 2 on mike315
Gid 4 from thread 0 out of 4 from process 1 out of 2 on mike315
Gid 6 from thread 2 out of 4 from process 1 out of 2 on mike315
Gid 7 from thread 3 out of 4 from process 1 out of 2 on mike315
Gid 5 from thread 1 out of 4 from process 1 out of 2 on mike315
bash,108067
|-mpirun,110651 -np 2 -x OMP_NUM_THREADS ./a.out
|   |-a.out,110652
|   |   |-sh,110666 -c ...
|   |   |   `--ssh,110670 mike315 pstree -ap -u fchen14
|   |   |-{a.out},110654
|   |   |-{a.out},110656
|   |   |-{a.out},110662
|   |   |-{a.out},110663
|   |   |-{a.out},110664
|   |   `--{a.out},110665
|
```

Exercise

- **Submit a small job to run “sleep 180” and “print PBS variables”**
 - Create a script to submit a 5 min job and print from within the job script PBS variables \$PBS_NODEFILE, \$PBS_WORKDIR. Also run “sleep 180” to give you a few minutes to verify status.
 - Once the job is running, find out the Mother Superior node and other slave nodes assigned to your job using qstat.
 - Log into MS node and verify that your job is running and find your temporary output file
 - Modify your script to print hello from each of your assigned nodes
- **Run a shell script using mpirun to print process id of shell**

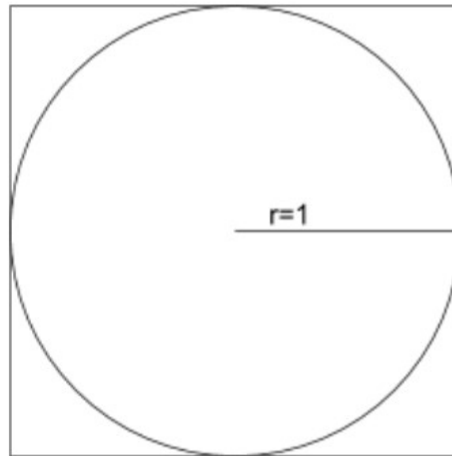
HPC@LSU User Services

- **Hardware resources**
 - Currently manages 4 clusters
- **Software stack**
 - Communication software
 - Programming support: compilers and libraries
 - Application software
- **Contact user services**
 - Email Help Ticket: sys-help@loni.org
 - Telephone Help Desk: +1 (225) 578-0900

Appendix

Computing an approximate value for PI

- The executables in this training calculate the value for PI based on the math which is actually quite simple: Imagine a square dartboard with circle inscribed within it such that the diameter of the circle is the length of a side of the square.



- We can observe that the ratio of the area of the circle to the area of the square is equal to some constant, $\pi/4$ (since the square's area is $2*2 = 4$ and $\text{area_circle} = \pi*r^2 = \pi$). If we randomly place many points (darts) inside the square, we can count how many are also inside the circle (satisfy $x^2+y^2 \leq 1$) vs the total number of points and compute an estimate for the value of π . (Problem description is from Jared Baker, UW; Ben Matthews, NCAR)