Introduction to High-Performance Computing (HPC)

Computer components

CPU : Central Processing Unit

cores : individual processing units within a CPU

Storage : Disk drives

HDD : Hard Disk Drive

SSD : Solid State Drive

Memory : small amount of volatile or temporary information storage

Computer components (my Macbook Pro)

Model Name: MacBook Pro

Processor Name: Intel Core i7

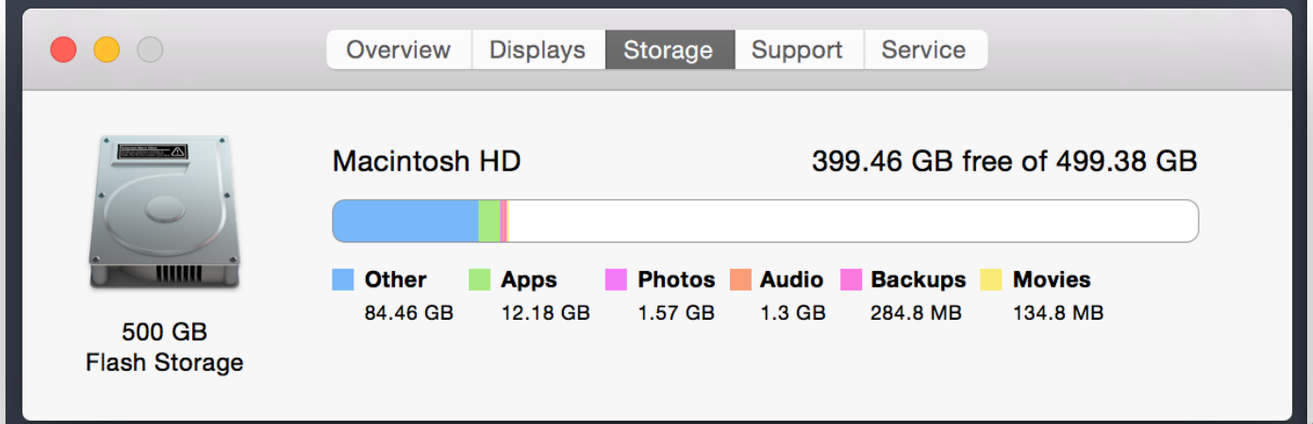
Processor Speed: 2.5 GHz

Number of Processors: 1

Total Number of Cores: 4

Memory: 16 GB

Computer components (my Macbook Pro)

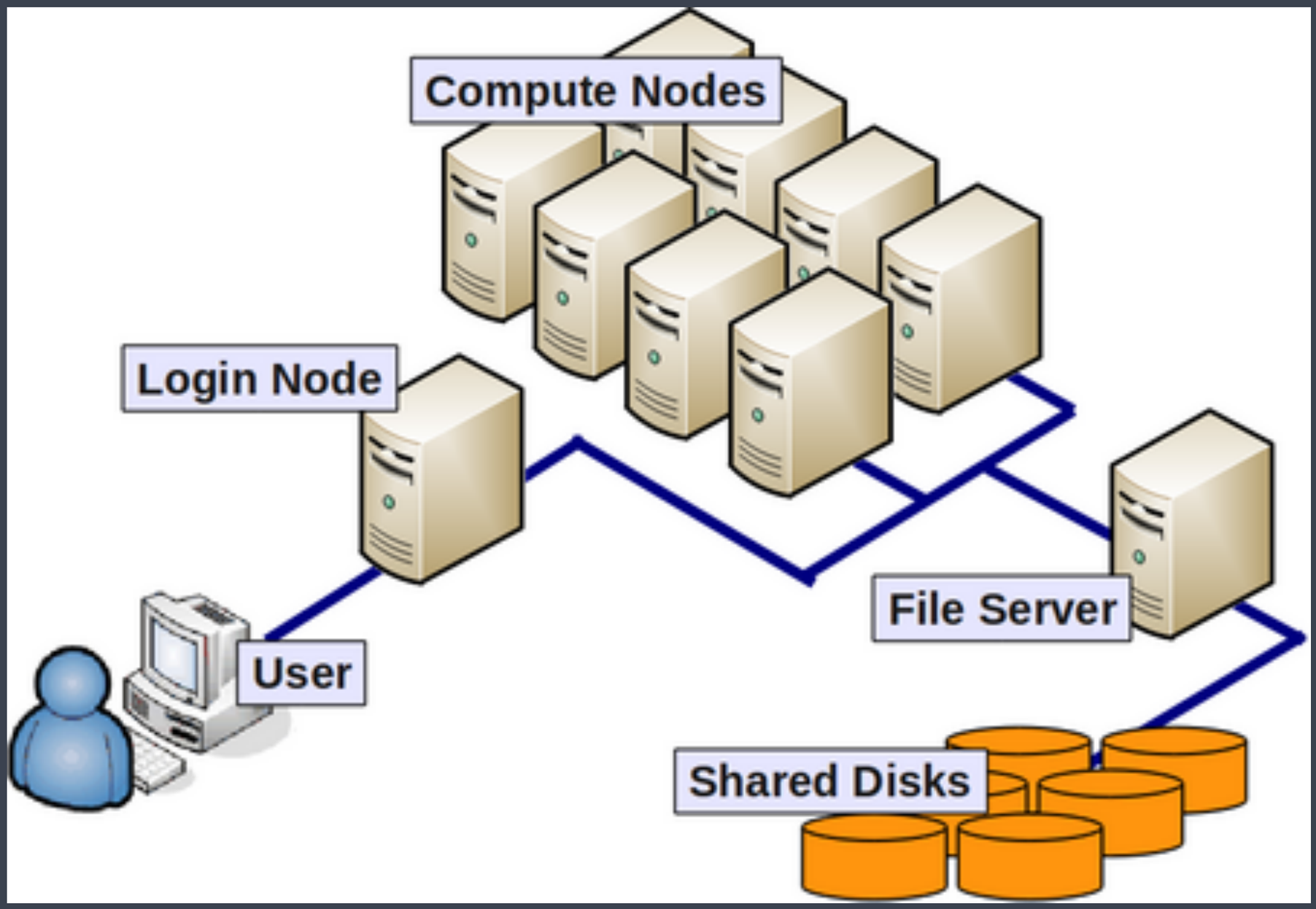


HPC cluster

“High Performance Computing most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business.”

http://insidehpc.com/hpc-basic-training/what-is-hpc/

HPC cluster



Need for HPC cluster?

- Resources: Your computer does not have the resources to run the desired NGS analysis
- Speed: You want to produce results faster than your computer can
- Software: You need software that is unavailable or unusable on your computer

HPC cluster components

Nodes: Individual computers in the cluster

Cores: individual processing units available within each CPU of each Node

e.g. a “Node” with eight “quad”-core CPUs = 32 cores for that node.

Shared disk: storage that can be shared (and accessed) by all nodes

HPC cluster

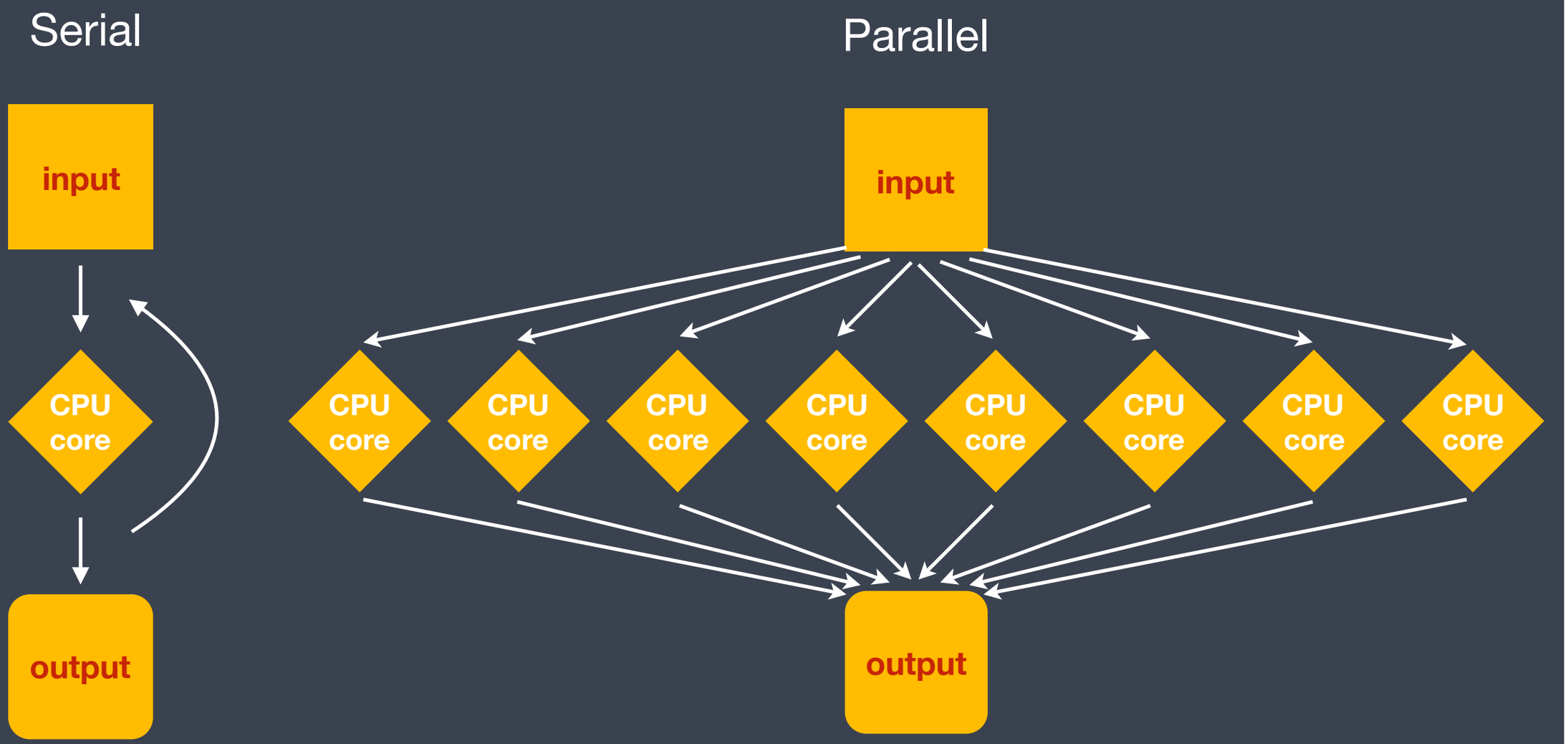
- multi-user, shared resource
- lots of nodes = lots of processing capacity + lots of memory

Parallel Computing

“Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel”)."

http://en.wikipedia.org/wiki/Parallel_computing

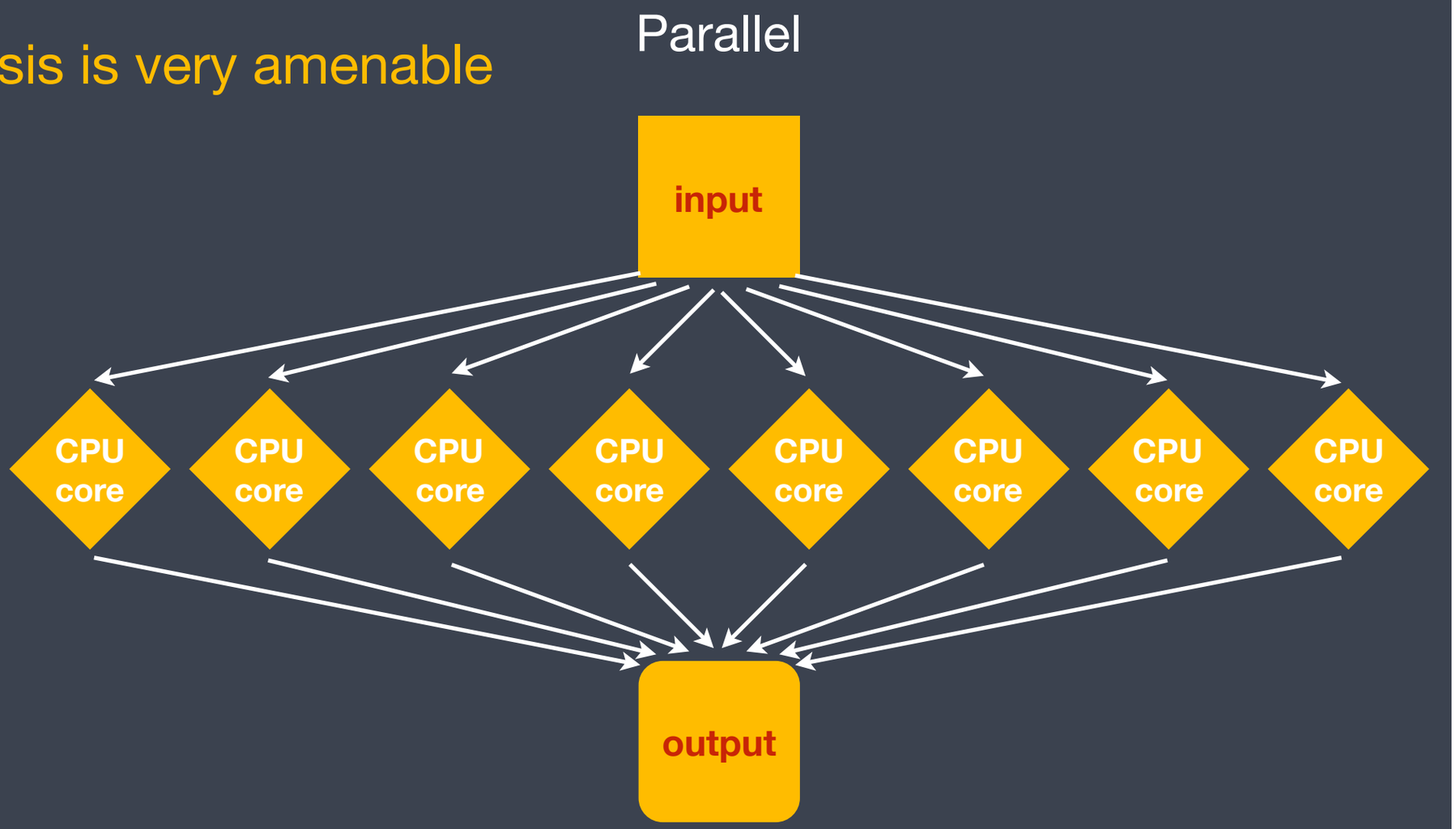
Parallel Computing



Faster and more efficient

Parallel Computing

NGS data analysis is very amenable to this strategy.



Faster and more efficient

HPC cluster

- multi-user, shared resource
- lots of nodes = lots of processing capacity + lots of memory
- a system like this requires constant maintenance and upkeep.

What is Orchestra?

Please see our Orchestra status page for known issues.

The **Orchestra** platform provides UNIX-based high performance computing, web hosting, and database hosting services at Harvard Medical School.

Orchestra and its associated services are managed by the **Research Computing Group**, part of the HMS Information Technology Department.

Please submit a request via the RC web site for help with Orchestra or feedback. You can also subscribe to the mailing list for Orchestra users.

Wiki page: https://wiki.med.harvard.edu/Orchestra

Introduction to
High Performance Computing and Orchestra

HMS Research Computing
Spring 2016

What is Orchestra?

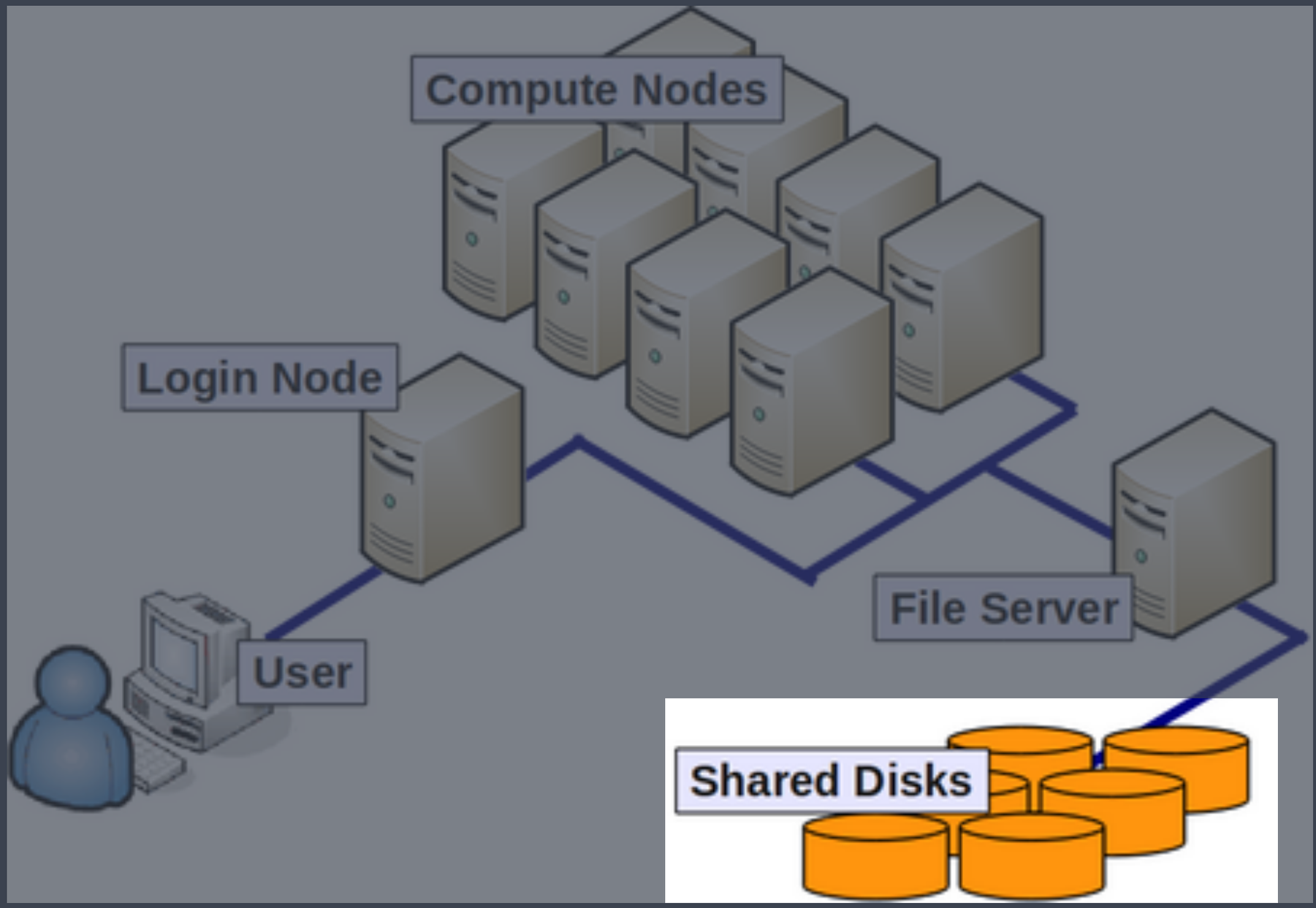
- Tech specs:
  - Over 500 compute nodes
  - Over 6,500 cores
  - 10GigE interconnection
  - Over 40TB RAM
- CentOS Linux
- LSF scheduler
- Total 25PB storage

Orchestra Compute Nodes

NODE CLASS	CPU TYPE	Number of Nodes	Cores x Node	MEMORY x Node (~GB)	Total Cores
Flute	Opteron2216	25	4	32 (vary)	100
Bassoon	XeonE5345	43	8	32 (vary)	344
Bassoon	XeonE5430	12	8	32	96
Sax	XeonE74807	2	16	1024	32
Clarinet	XeonE5530	21	8	vary	168
Clarinet	XeonL5640	294	12	96 (vary)	3528
Fife	XeonE52680	16	20	190	320
Piccolo	XeonE5266v2	40	20	128	800
Ocarina	Opteron6376	8	64	505	512
Ottavino	XeonE52697	16	28	252	448

Filesystems and storage

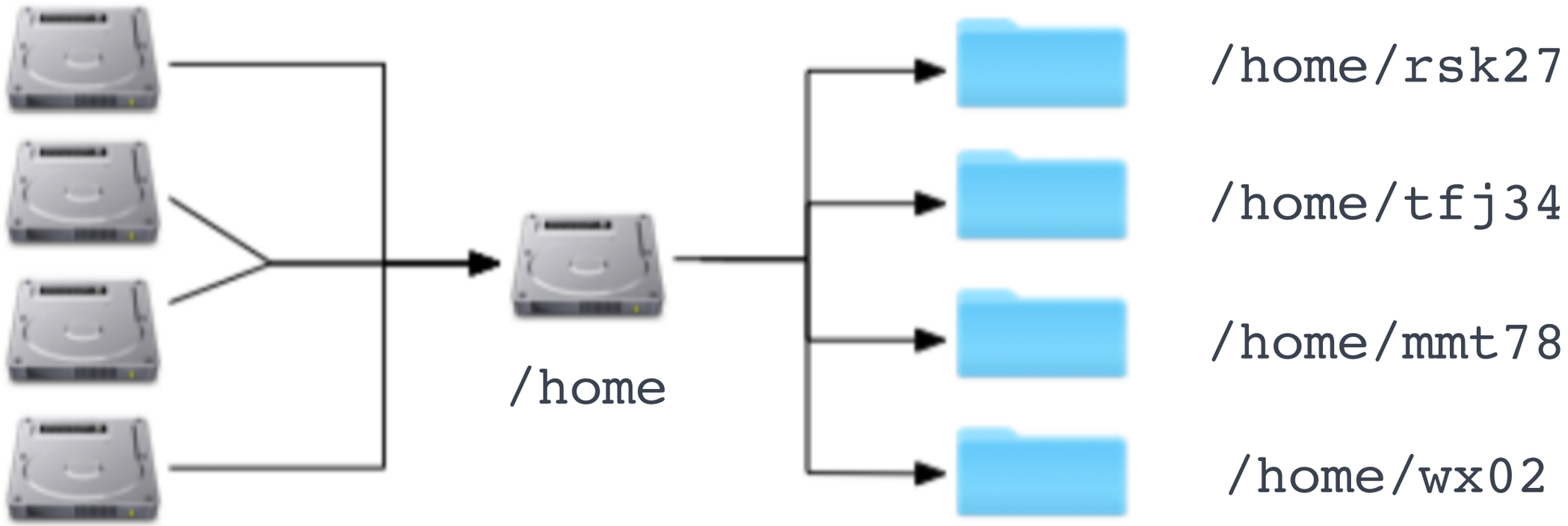
Filesystems and storage



Filesystems and storage

- Storage on HPC systems is organized differently than on your personal machine
- Physical disks are bundled together into a virtual volume; this volume may represent a single filesystem, or may be divided up, or partitioned, into multiple filesystems
- Filesystems are accessed over the internal network

Filesystems and storage



Storage on Orchestra

/home

- /home/user_id
- quota: 100GB per user
- Backup: 14-day snapshots + 60 day weekly snapshots

/groups, /n/data1, /n/data2

- /groups/your_lab/your_dir
- quota: expandable
- Backup: 14-day snapshots + 60-day weekly snapshots
- Run jobs here

Checking storage usage

- To check your storage available:

```
$ quota
```

- Home directory (`/home/username`): you get 100GB.

- Group directories: space varies, can increase.

```
/groups/groupname

/n/data1

/n/data2
```

More storage coming soon!

DDN ExaScalar

- High-performance parallel file system: Lustre
- More than 1PB of disk space
- No quota, no backup, data will be purged
- Ideal for large-scale job running

DDN GridScalar

- High-performance parallel file system: GPFS
- More than 2PB of space
- Quota policy, for long-term storage
- Ideal for large-scale job running
- Available for large-data producing labs

Using Orchestra!

1. Logging in to remote machines (securely)

- When logging in we used the “ssh” command,
  ssh stands for Secure SHell

- **ssh** is a protocol for data transfer that is secure, i.e the data is encrypted as it travels between your computer and the cluster (remote computer)

- Commonly used commands that use the **ssh** protocol for data transfer are, **scp** and **sftp**

2. Using & installing software

Using Software Environment Modules

- Most “software” on Orchestra is installed as an environment module.
- An environment module makes the necessary modifications in $PATH to make sure the program runs without any issues.
- Allows for clean, easy loading, including most dependencies, and switching versions.

Using Software Environment Modules

- Most “software” on Orchestra is installed as an environment module.
- An environment module makes the necessary modifications in $PATH to make sure the program runs without any issues.
- Allows for clean, easy loading, including most dependencies, and switching versions.

```
$ module avail
$ module avail seq/
$ module avail stats/
$ module avail seq/bowtie/
```

What software environment modules are available on Orchestra?

Loading/Unloading Modules

- Loading modules

```
$ module load seq/bowtie/2.0.6
```

- Which module version is loaded (if at all)?

```
$ which bowtie
```

- See all modules that have been loaded

```
$ module list
```

- Unloading modules

```
$ module unload seq/bowtie/2.0.6
```

- Dump all modules

```
$ module purge
```

3. The Job Scheduler, LSF

LSF: Fair Sharing

- **L**oad **S**haring **F**acility: distributes jobs across Orchestra fairly
- Ensures that no single user or core monopolizes Orchestra
- Users are assigned dynamic priorities
- Queues also have priorities
- Submitting lots of jobs reduces your fairshare priority
- Even if many jobs are pending, your jobs will start quickly provided you have not submitted many jobs

3a. Running and submitting jobs with the LSF scheduler

Submitting Jobs

- In an “interactive session”, programs can be called directly.

```
mfk8@clarinet001:~$ bowtie –p 4 hg19 file1_1.fq file1_2.fq
```

- From the login shell (and also interactive or any compute nodes), a program is submitted to Orchestra via a job (bsub).

```
mfk8@clarinet001:~$ bsub < mybowtiejob.lsf
```

- Orchestra will notify you when the job is done, or if there is an error.

3b. Choosing the proper resources for your job with the appropriate **bsub** options

The “bsub”

```
mfk8@clarinet001:~$ bsub –q queue –W hr:min job
```

- Necessary to specify:
  - -q (queue)
  - -W (runtime in hr:min)

Runtime Limit

- -W in hours:minutes
- Runtimes are subject to the maximum time permitted per queue (see table)
- If your job exceeds your runtime, your job will be killed ☹
- Running many jobs that finish quickly (less than a few minutes) is suboptimal and may result in job suspension, contact RC to learn how to batch jobs

CPU Limit

- Amount of seconds the cluster works on your job (calculated by LSF)
- Ncores * Runlimit (-n * -W)
- Common error:

  bsub –q short –W 8:00 tophat –p 8

  tophat asks for 8 cores (-p 8) but only 1 core requested (no –n), job killed in 1 hour (8/8)

Reserving Memory

- Most nodes have 90GB memory available over all cores, some have more
- Make a resource request with

  -R “rusage[mem=8000]” (memory requested in MB)

- Memory multiplies by cores requested, so

  -n 4 –R “rusage[mem=16000]” reserves 64GB memory

- Asking for more memory may cause jobs to pend longer
- TERM_MEMLIMIT errors indicate that not enough memory was reserved

Multithreading

- A single CPU can execute multiple processes (threads) concurrently
- -n indicates how many cores are requested
- Jobs that are overefficient (use more cores than reserved) jeopardize the health of a node
- Reserve the same amount of cores in your job and your bsub!

MPI

- Efficient way to run parallel jobs
- Dedicated queue with same-type compute nodes for optimal performance (800 cores)
- Matlab, Python, Java, R, C++, Fortran have MPI options
- Very commonly used by structural biology software
- Not as commonly used by software dealing with NGS data
- Orchestra implementation: openMPI-1.8.6

bsub options

```
-n 4                (number of cores requested)
-R “rusage[mem=8000]”    (memory requested in MB)
-J jobname
-a openmpi          (type of mpi run)
-Is                 (interactive shell)
-e errfile          (send  errors to file)
-o outfile          (send screen output to file)
-N                  (notify when job completes)
```

3c. Job Queues, choosing the computer nodes that suit your needs the best

Shared Queues

- ***mpi*** queue if you have an MPI parallel job
- ***priority*** queue if you have just one or two jobs to run
- ***mcore*** queue if you have multi-core jobs to run.
- ***short*** queue if your jobs will take less than 12 hours to run.
- Else: ***long*** queue.

Shared Queues Breakdown

Queue Name	Priority	Max Cores	Max Runtime
interactive	*12*	*12*	*12 hours*
priority	*14*	*12*	*1 month*
mcore	*10*	*12*	*1 month*
mpi	*12*	*512*	*1 month*
parallel	*12*	*512*	*1 month*
short	*8*	*12*	*12 hours*
long	*6*	*12*	*1 month*
mini	*5*	*1*	*10 minutes*

3d. Job submission scripts

Shell Scripts: The Basics

```
#! /bin/sh              #always at the top

#commands with options
tophat -p 4 -o ./mytophatdir1 hg19 file1_1.fastq file1_2.fastq
tophat -p 4 -o ./mytophatdir2 hg19 file2_1.fastq file2_2.fastq
```

```
#save as myshellscript.sh
```

Calling a Shell Script

```
mfk8@balcony:~$ bsub -q mcore -W 12:00 -n 4 -e errfile.txt
-o outfile.txt -N sh myshellscript.sh
```

Creating a Complete Shell Script

```
#! /bin/sh
#BSUB -q mcore
#BSUB -W 12:00
#BSUB -o outfile.txt
#BSUB -e errfile.txt
#BSUB –N
#BSUB -n 4
 #BSUB –R “rusage[mem=12000]”

 tophat -p 4 -o ./mytophatdir1 hg19 file1_1.fastq file1_2.fastq

#save as myshellscript2.lsf
```

Calling a Complete Shell Script

```
mfk8@balcony:~$ bsub < myshellscript2.lsf
```

3e. Managing jobs and getting information about submitted/running jobs

Monitoring Jobs

- List info about jobs/their status:

```
mfk8$loge:~$ bjobs
```

-r (running jobs)

-p (pending jobs)

-l (command entered, long form)

- List historical job information

```
mfk8$loge:~$ bhist jobid
```

Job Suspensions

- Jobs in the “long” queue can be suspended by LSF temporarily to allow jobs from the “short” queue (less than 12h) to run.
- If a job is suspended for >50% of the –W runlimit cumulatively, it will be automatically moved to the “no_suspend” queue where it can’t be suspended anymore.

Terminating Jobs

- Terminate a job (jobid given at submission)

```
mfk8$loge:~$ bkill jobid
```

- Terminate all jobs (be careful!)

```
mfk8$loge:~$ bkill 0
```

4. Additional tips and information for effectively using Orchestra

Snapshots

- Snapshots are retained for up to 60 days: recover data

```
mfk8@clarinet001:~$ cd .snapshot
mfk8@clarinet001:~$ ls
   Orchestra_home_daily_2015-10-02-02-00
   Orchestra_home_daily_2015-10-01-02-00
mfk8@clarinet001:~$ cd Orchestra_home_daily_2015-10-02-02-00
mfk8@clarinet001:~$ cp MyRetrievedFile ~
```

Compiling your own software

- Users can compile software in their /home or /groups directories, where they have permission
- Binaries just require “unzipping” (ie tar –zxvf .tar.gz)
- Common compiling libraries are found as modules:

  dev/compiler/gcc-4.8.5

  dev/boost/1.57.0

  dev/openblas/0.2.14

  dev/lapack

- Common libraries are found in “utils”

Installing Software: Binary Example

```
mfk8@loge:~$ bsub –Is –q interactive bash
mfk8@clarinet001:~$  wget http://path/to/binary/
  mysoftware.tar.gz
mfk8@clarinet001:~$ tar –zxvf mysoftware.tar.gz
mfk8@clarinet001:~$ ls mysoftware/bin
```

Installing Software: Source

```
mfk8@loge:~$ bsub –Is –q interactive bash
mfk8@clarinet001:~$ module load dev/compiler/gcc-4.8.5
mfk8@clarinet001:~$  wget http://path/to/source/mysoftware.tar.gz
mfk8@clarinet001:~$ tar –zxvf mysoftware.tar.gz
mfk8@clarinet001:~$ cd mysoftware
```

```
mfk8@clarinet001:~$ less README
mfk8@clarinet001:~$ ./configure ––prefix=/home/mfk8/software
mfk8@clarinet001:~$ make
mfk8@clarinet001:~$ make install
```

Python, R, Perl

- Users manage their own package libraries: get the version you want, when you want it!
- Python: virtual environment allows pip installs

  https://wiki.med.harvard.edu/Orchestra/PersonalPythonPackages
- R: set up personal R library for cran, bioconductor

  https://wiki.med.harvard.edu/Orchestra/PersonalRPackages
- Perl: local::lib allows cpan, cpanm

  https://wiki.med.harvard.edu/Orchestra/PersonalPerlPackages
- Shebangs: #!/usr/bin/env python/Rscript/perl

For more direction:

- https://wiki.med.harvard.edu/Orchestra/NewUserGuide
- http://rc.hms.harvard.edu
- rchelp@hms.harvard.edu
- Office Hours: Wednesdays **1-3p Gordon Hall 500**