Using Computer Clusters with Python

BIOS 274: Introductory Python Programming for Genomics
Veronica Behrens
12/9/2019

What is a computer cluster?

A group of powerful computers (and other resources like software and storage) that act like a single system

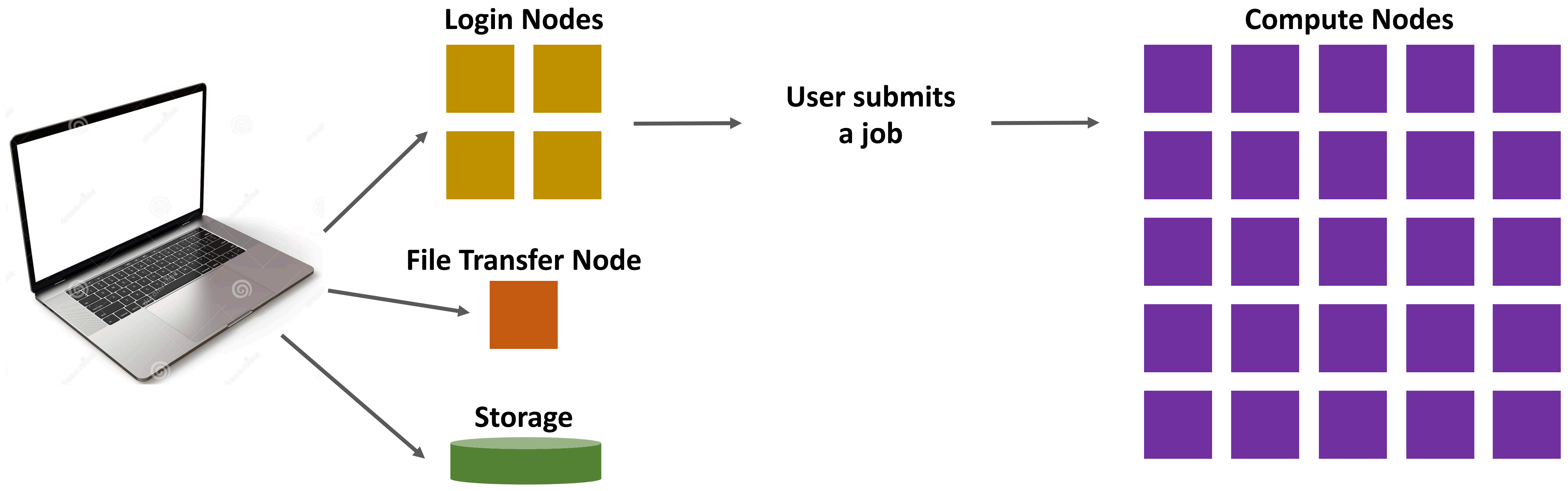
Why use a computer cluster?

- For many genomics “big data” analyses, your local laptop/computer is not powerful enough!

- On the cluster:
    - Useful software is already installed
    - There’s lots of storage
    - There’s lots of RAM
    - You can parallelize jobs

How are computer clusters organized?

- **Login/Head nodes**: For copying data, editing scripts, short test runs
- **Compute nodes**: For running jobs (scripts)
- **Storage**



Computer clusters at Stanford

Available to everyone at Stanford for free:

- FarmShare (rice)

Must be added as an authorized user by PI:

- SCG4
- Sherlock

Logging in to the FarmShare cluster

```
ssh SUNetID@rice.stanford.edu
```

- password
- DUO authentication

Transferring files to and from the cluster

- **Secure File Transfer Protocol**
  - `sftp`
  - `lpwd, lcd, lls` to navigate local computer (your laptop)
  - `pwd, cd, ls` to navigate the remote computer (the cluster / rice)
  - `put FILENAME` to transfer to cluster from local computer
  - `get FILENAME` to transfer from cluster to local computer

- **Secure Copy**
  - `scp FILE_ON_LAPTOP SUNetID@rice.stanford.edu:DIRECTORY_ON_CLUSTER`
  - `scp SUNetID@rice.stanford.edu:FILE_ON_CLUSTER DIRECTORY_ON_LAPTOP`

- **User-Friendly Graphical User Interfaces (GUIs)**
  - Fetch (Mac): https://fetchsoftworks.com/fetch/
  - CyberFX (Windows): https://uit.stanford.edu/software/scrt_sfx

Accessing software on the cluster

`module avail`	List all available modules
`module load MODULE_NAME`	Load a module for use
`module unload MODULE_NAME`	Unload a module
`module list`	List all currently loaded modules
`module spider MODULE_NAME`	Detailed information about a particular module

Check out all the software in /usr/bin, too!

Some useful software

• **bedtools**	
• **samtools**	
• bamtools	
• vcftools	
• **bcftools**	
• sratoolkit	
• picard-tools	manipulating various sequencing data files
• ucsc_tools	manipulating various sequencing data files
• **blastn**	BLAST on nucleotides
• muscle	multiple sequence alignment
• **mafft**	multiple sequence alignment

bold indicates the tool is available on rice

Some useful software

- Dealing with FASTQ files and mapping

`bwa`	mapping (DNA-seq)
`bowtie2`	mapping (DNA-seq)
`STAR`	mapping (RNA-seq)
`salmon`	mapping (RNA-seq)
`fastqc`	stats about fastq file
`trim_galore`	quality filter fastq files
`trimmomatic`	quality filter fastq files
`cutadapt`	quality filter fastq files

bold indicates the tool is available on rice

Submitting a job on rice using the SLURM job scheduler

1. Save the following text in a file called `testJob.sh`

```
#!/bin/bash
#SBATCH --job-name=testJob                     # Job name
#SBATCH --mail-type=END,FAIL                   # Mail events (NONE, BEGIN, END, FAIL, ALL)
#SBATCH --mail-user=SUNetID@stanford.edu       # Where to send notification emails
#SBATCH --ntasks=1                             # Number of CPUs to use
#SBATCH --mem-per-cpu=1GB                      # Job memory request
#SBATCH --time=00:05:00                        # Time limit in hrs:min:sec
#SBATCH --output=testJob.log                   # Output file for job

##### YOUR COMMANDS TO RUN HERE #####
echo 'Running test job!' > testJob.txt
python3 test.py
```

2. Submit the script as job: `sbatch testJob.sh`

Other useful SLURM commands:

- `squeue -u SUNetID` Check the status of your jobs
- `scancel JOB_ID` Delete a job

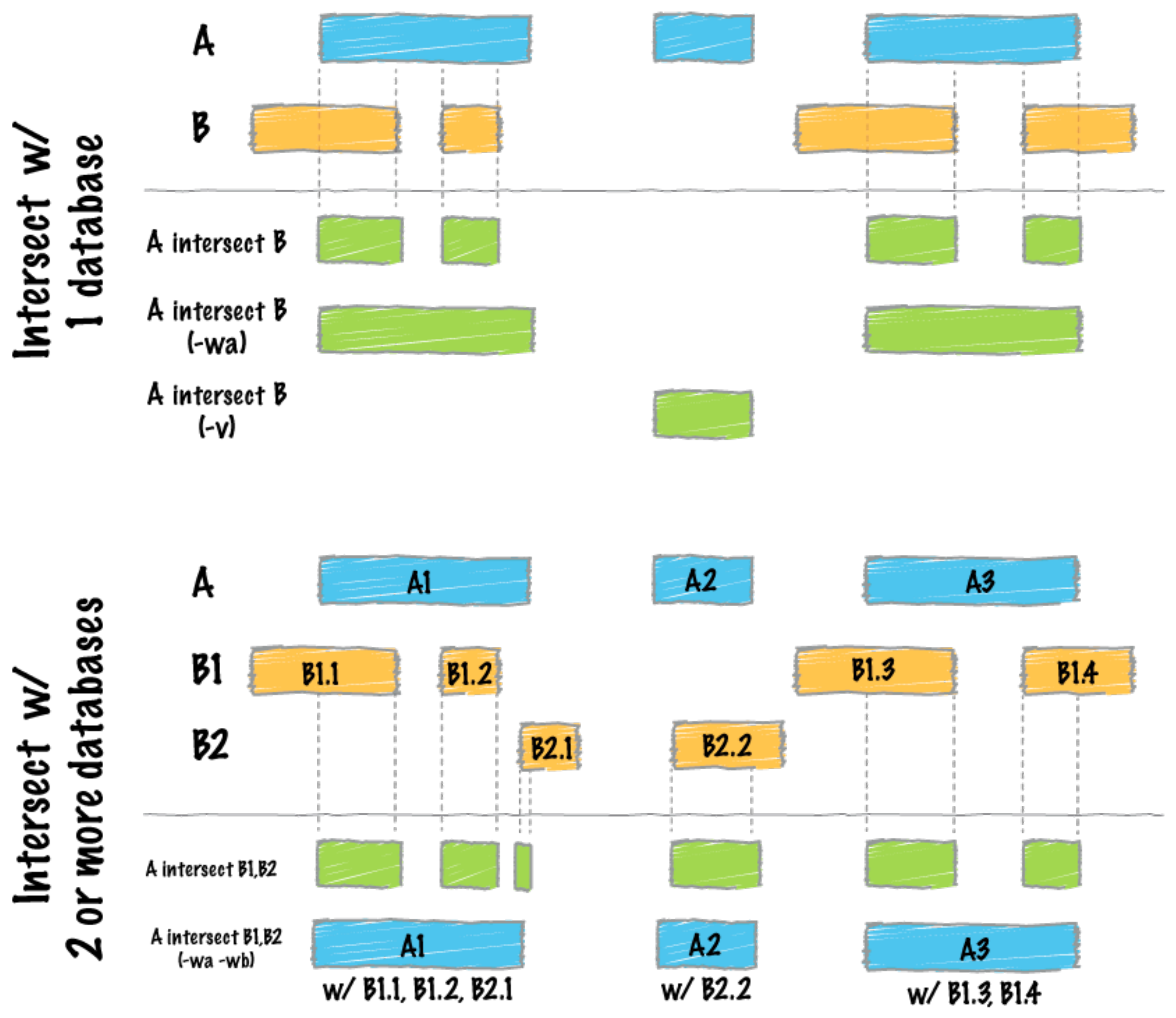
Login to rice, make directory and symbolic links

```
1. ssh SUNetID@rice.stanford.edu

2. mkdir Day9

3. cd Day9

4. ln -s /afs/ir.stanford.edu/class/gene211/misc/BIOS274/Day9/blastQuery.fa

5. ln -s /afs/ir.stanford.edu/class/gene211/misc/BIOS274/Day9/hg38.chrom.sizes

6. ln -s /afs/ir.stanford.edu/class/gene211/misc/BIOS274/Day9/hg38_exons.bed

7. ln -s /afs/ir.stanford.edu/class/gene211/misc/BIOS274/Day9/hg38_genes.bed

8. ln -s /afs/ir.stanford.edu/class/gene211/misc/BIOS274/Day9/hg38.fa.gz

9. ln -s /afs/ir.stanford.edu/class/gene211/misc/BIOS274/Day9/test.bam
```

Run blast from the command line

```
# Make a blast database from the hg38 genome
# After running this command, we will also have hg38.nhr, hg38.nin, and hg38.nsq files
zcat hg38.fa.gz | makeblastdb -dbtype nucl -out hg38 -title hg38

# Run blast
blastn -db hg38 -query blastQuery.fa -out blastQuery_results.txt

# How can we make the output more useful?
blastn -db hg38 -query blastQuery.fa -out blastQuery_results.tsv \
-outfmt '6 sseqid sstart send qseqid pident’

# Write a python script called formatResults.py that removes ‘chr’ from the beginning of each chromosome name,
ensures the start <= end, and condenses the final 2 columns into a single |-delimited column
python3 formatResults.py blastQuery_results.tsv > blastQuery_results.bed

# Which queries are in exons?
bedtools intersect -wa -a blastQuery_results.bed -b hg38_exons.bed \
> blastQuery_results_inExons.bed
```

bedtools intersect



https://bedtools.readthedocs.io/en/latest/content/tools/intersect.html

Download ChIP-seq data

```
# Download:  ZFP91 ChIP-seq on human K562
wget https://www.encodeproject.org/files/ENCFF150ZBH/@@download/ENCFF150ZBH.bed.gz
gunzip ENCFF150ZBH.bed.gz
mv ENCFF150ZBH.bed ZFP91_K562.bed

# Download:  EGR1 ChIP-seq on human K562
wget https://www.encodeproject.org/files/ENCFF175VSS/@@download/ENCFF175VSS.bed.gz
gunzip ENCFF175VSS.bed.gz
mv ENCFF175VSS.bed EGR1_K562.bed

# Download:  FOXA1 ChIP-seq on human K562
wget https://www.encodeproject.org/files/ENCFF765NAN/@@download/ENCFF765NAN.bed.gz
gunzip ENCFF765NAN.bed.gz
mv ENCFF765NAN.bed FOXA1_K562.bed

# Download:  K562 DNase-seq
wget https://www.encodeproject.org/files/ENCFF821KDJ/@@download/ENCFF821KDJ.bed.gz
gunzip ENCFF821KDJ.bed.gz
mv ENCFF821KDJ.bed DNase_K562.bed
```

Find the most interesting regions

```
# Find regions that overlap an EGR1, a FOXA1, a ZFP91, and a DNase peak.
bedtools intersect -a EGR1_K562.bed -b FOXA1_K562.bed | \
bedtools intersect -a stdin -b ZFP91_K562.bed | \
bedtools intersect -a stdin -b DNase_K562.bed > \
EGR1_FOXA1_ZFP91_DNase_K562.bed

# Make the format more useful.
python3 formatResults.py EGR1_FOXA1_ZFP91_DNase_K562.bed \
> EGR1_FOXA1_ZFP91_DNase_K562_temp.bed

mv EGR1_FOXA1_ZFP91_DNase_K562_temp.bed EGR1_FOXA1_ZFP91_DNase_K562.bed

# Which gene is closest to each of these regions?
bedtools sort -faidx hg38.chrom.sizes -i EGR1_FOXA1_ZFP91_DNase_K562.bed \
> EGR1_FOXA1_ZFP91_DNase_K562_sorted.bed

mv EGR1_FOXA1_ZFP91_DNase_K562_sorted.bed EGR1_FOXA1_ZFP91_DNase_K562.bed \
bedtools closest -a EGR1_FOXA1_ZFP91_DNase_K562.bed -b hg38_genes.bed \
-g hg38.chrom.sizes

# Use cut to generate a file with only the most important columns.
cut -f1,2,3,8,9 EGR1_FOXA1_ZFP91_DNase_K562_withGenes.bed \
> EGR1_FOXA1_ZFP91_DNase_K562_withGenes_simple.bed
```

Let’s work with a BAM file!

```
# Try to look at test.bam.  What happens?  Can samtools help?
samtools view test.bam

# Let’s look at some informative stats about test.bam
samtools flagstat test.bam

# How many reads have a MAPQ score above 30?
samtools view -q 30 test.bam | wc -l

# What is the read depth in these three regions?
chr1:10311-10472770
chr1:16212-16230346
chr1:104944-104970612
samtools index test.sam
samtools bedcov roi.bed test.sam
```