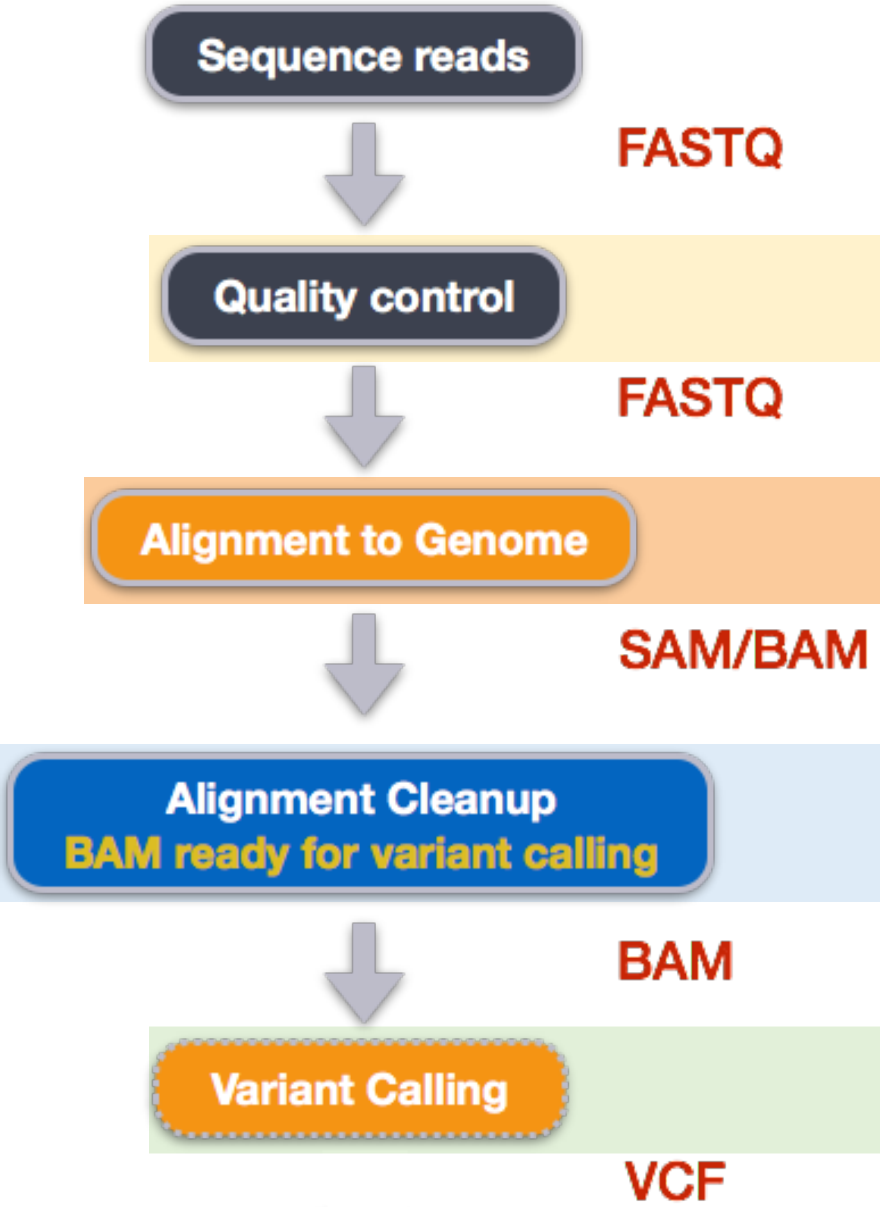
So far:

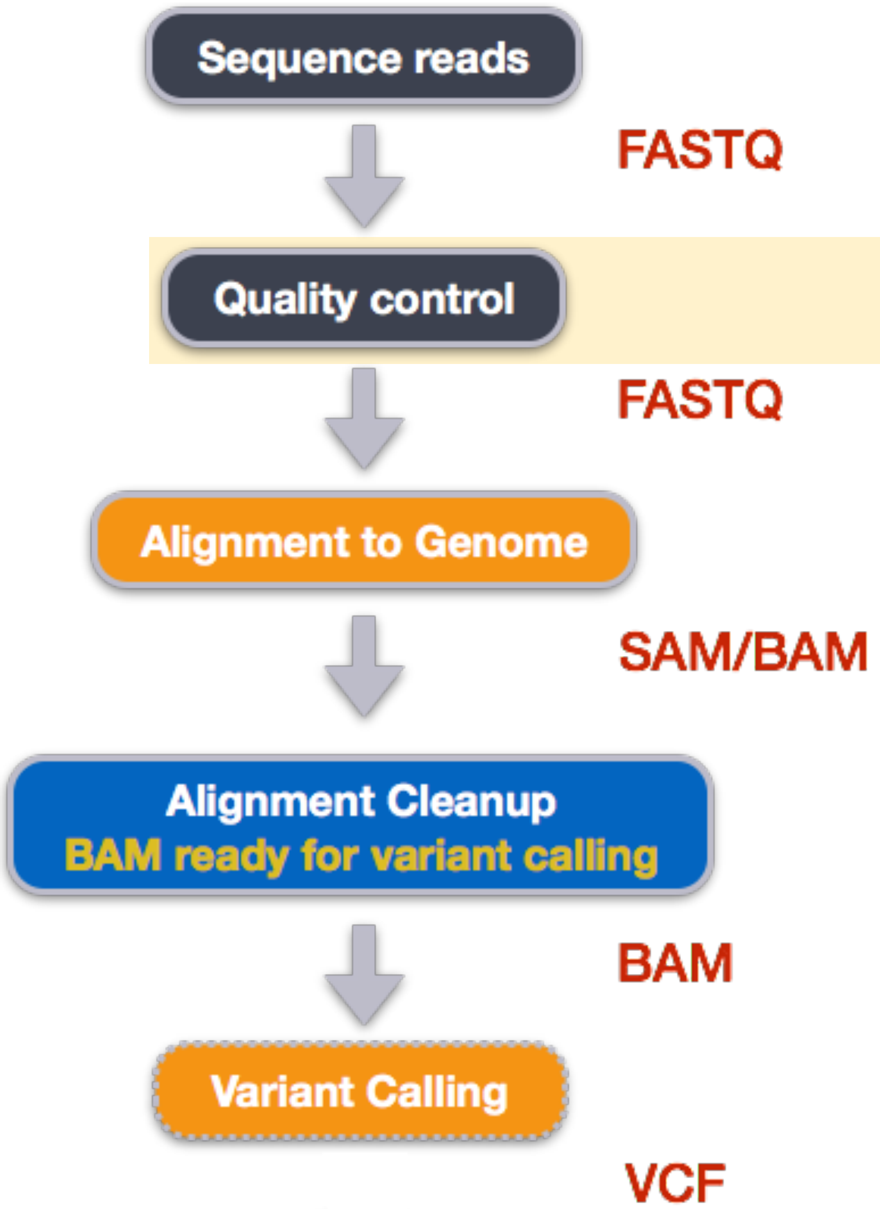
- Project Organization
- Background and metadata
- Assessing sequencing read quality
- Trimming and filtering reads based on read quality
- Variant calling workflow
- Automating a variant calling workflow
- Cloud computing

Building our Pipeline:



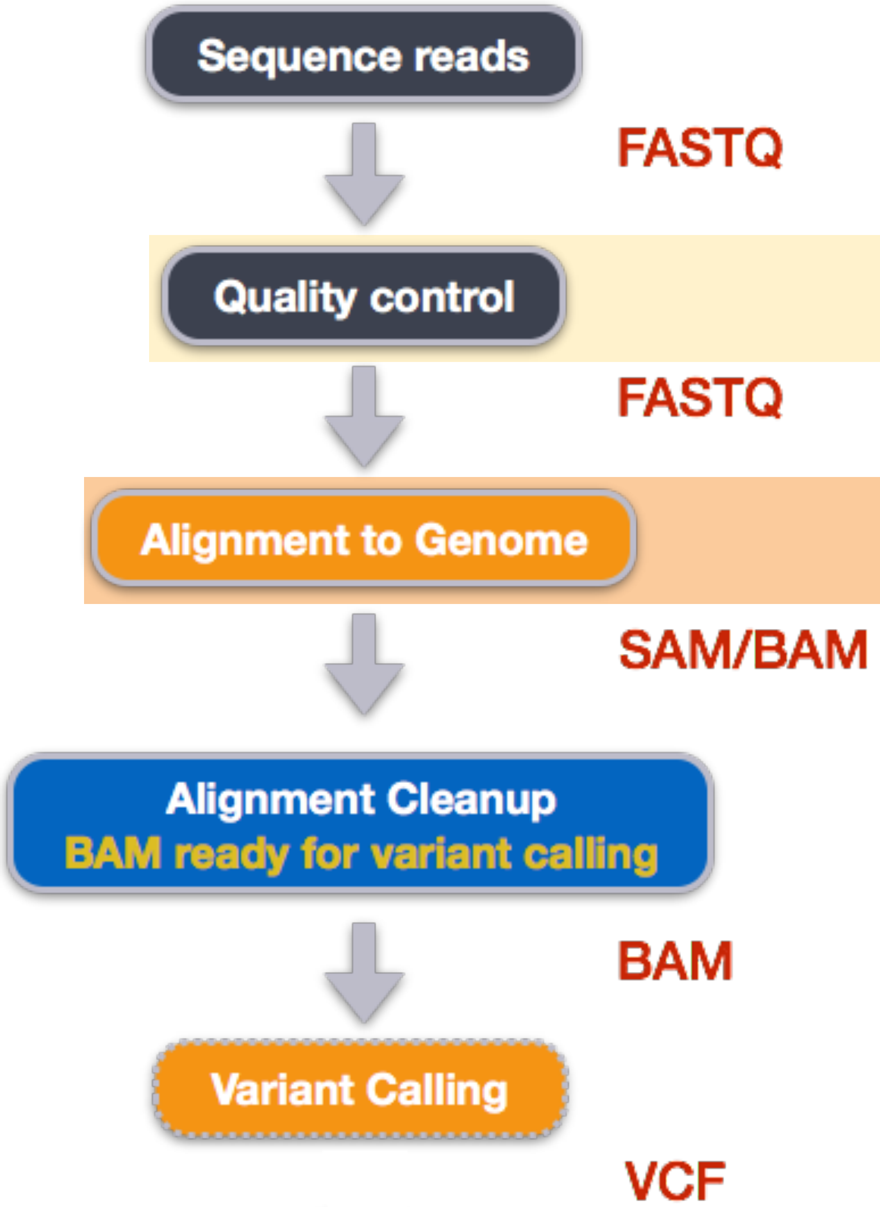
```
# STEP 1: FASTQC raw data

# STEP 2: TRIM and FILTER

# STEP 3: FASTQC filtered data

# STEP 4: Index reference genome

# STEP 5: Align trimmed reads

# STEP 6: Convert from SAM to BAM

# STEP 7: Sort BAM file

# STEP 8: Calculate read coverage of positions in genome

# STEP 9: Detect SNPs

# STEP 10: Filter and report SNPs in VFC
```

Building our Pipeline:



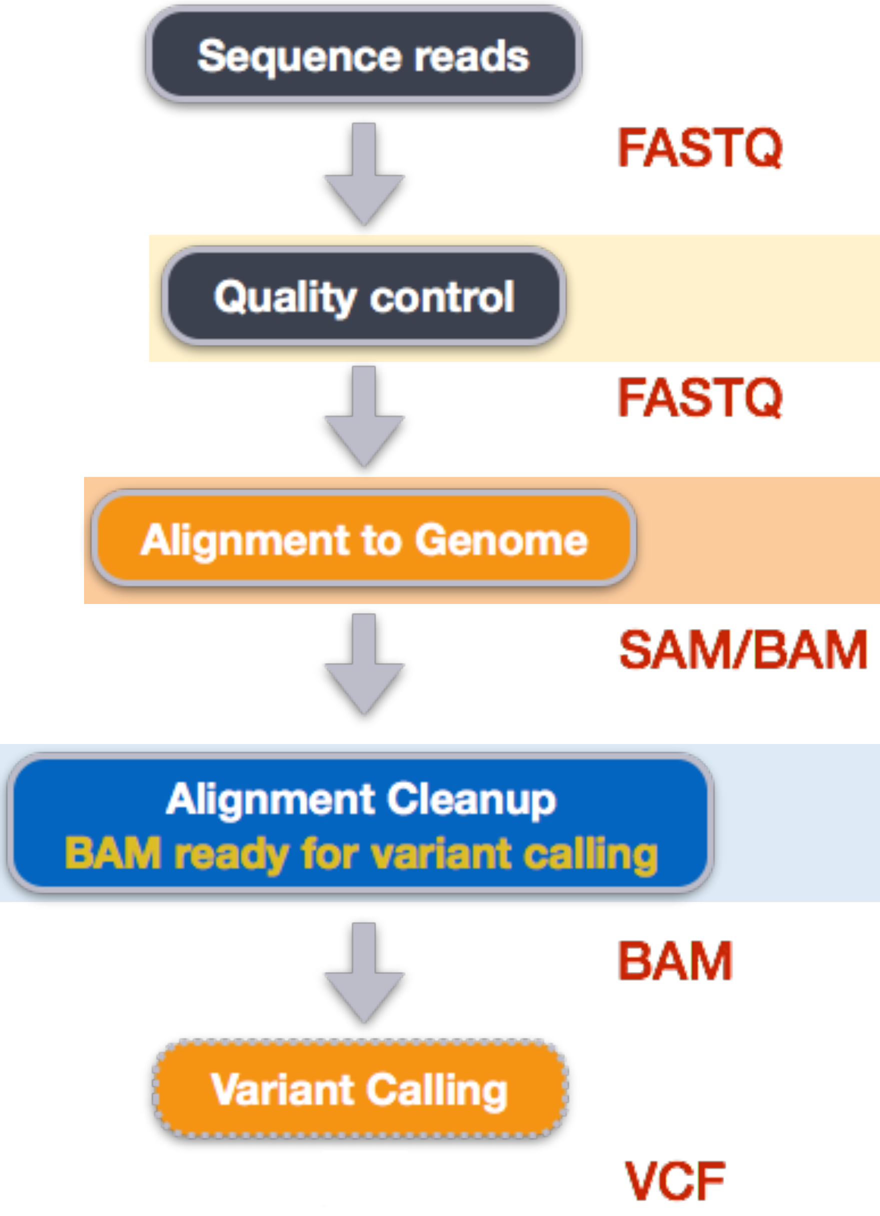
```
# STEP 1: FASTQC raw data

fastqc *.fastq*               # Run FASTQC

for filename in *.zip         # unzip fastqc .zip files
do
unzip $filename
done

cat */summary.txt > fastqc_summaries.txt         # obtain fastqc summary
grep FAIL fastqc_summaries.txt > fastqc_FAIL.txt  # identify problem samples

# STEP 2: TRIM and FILTER

for infile in *_1.fastq.gz                # Run trimmomatic
do
  base=$(basename ${infile} _1.fastq.gz)
  trimmomatic PE ${infile} ${base}_2.fastq.gz \
               ${base}_1.trim.fastq.gz ${base}_1un.trim.fastq.gz \
               ${base}_2.trim.fastq.gz ${base}_2un.trim.fastq.gz \
               SLIDINGWINDOW:4:20 MINLEN:25 ILLUMINACLIP:NexteraPE-PE.fa:2:40:15
done

# STEP 3: FASTQC filtered data

fastqc ~/dc_workshop/data/trimmed_fastq*.fastq*            # Run FASTQC
```

Building our Pipeline:



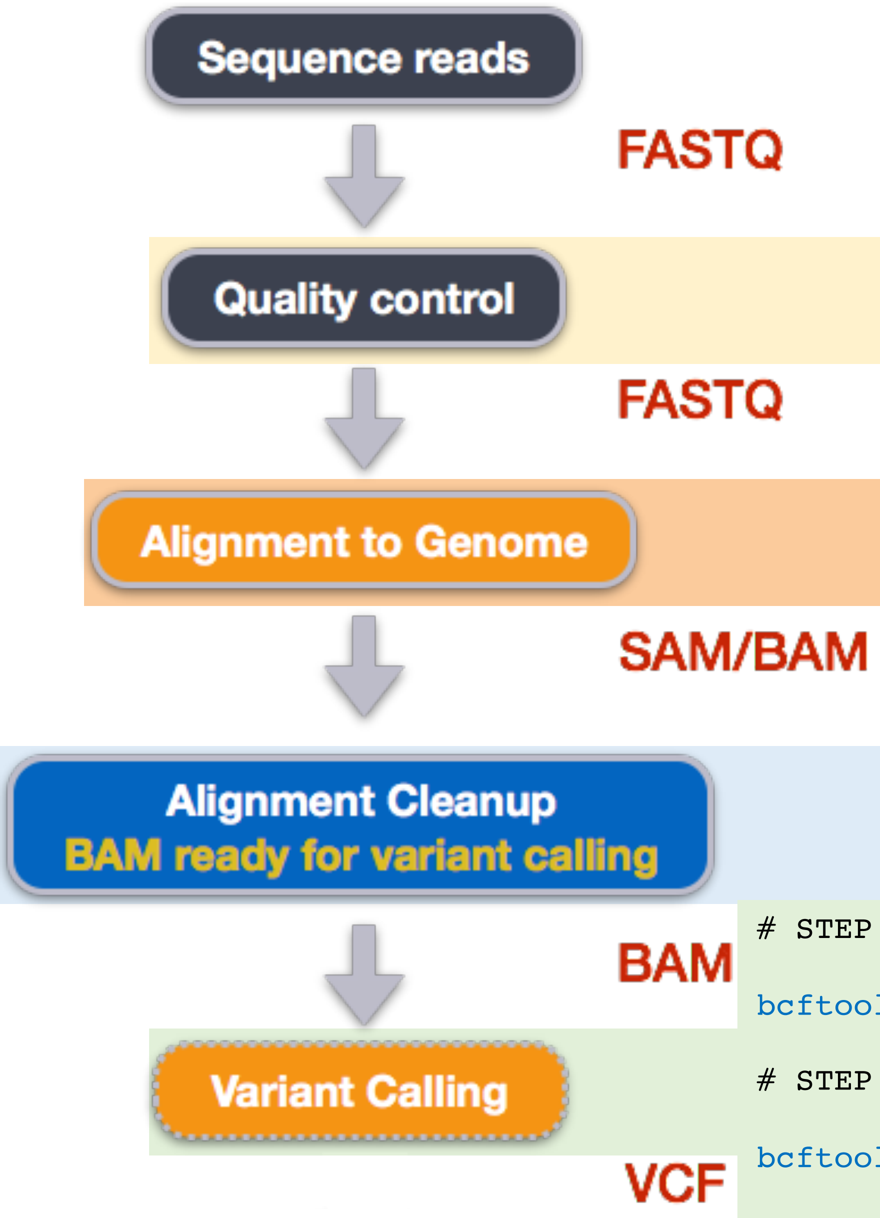
```
# STEP 1: FASTQC raw data

# STEP 2: TRIM and FILTER

# STEP 3: FASTQC filtered data

# STEP 4: Index reference genome

bwa index PATH/ref_genome/ref_genome.fasta

# STEP 5: Align trimmed reads

bwa mem ref_genome.fasta input_file_R1.fastq input_file_R2.fastq > output.sam
```

Building our Pipeline:



```
# STEP 1: FASTQC raw data

# STEP 2: TRIM and FILTER

# STEP 3: FASTQC filtered data

# STEP 4: Index reference genome

# STEP 5: Align trimmed reads

# STEP 6: Convert from SAM to BAM

samtools view -S -b PATH/sam/output.sam > PATH/bam/output.bam

# STEP 7: Sort BAM file

samtools sort -o PATH/bam/output.sorted.bam PATH/bam/output.bam
```

Building our Pipeline:

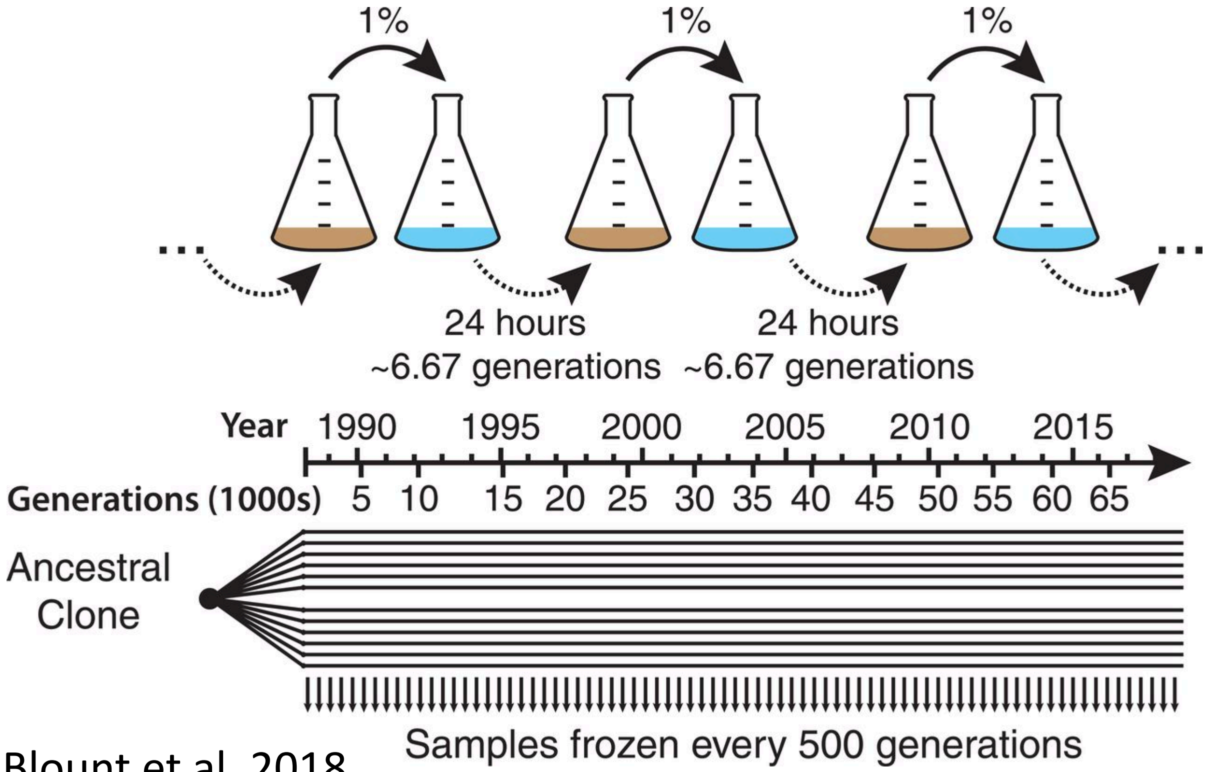


```
# STEP 1: FASTQC raw data

# STEP 2: TRIM and FILTER

# STEP 3: FASTQC filtered data

# STEP 4: Index reference genome

# STEP 5: Align trimmed reads

# STEP 6: Convert from SAM to BAM

# STEP 7: Sort BAM file

# STEP 8: Calculate read coverage of positions in genome

bcftools mpileup -O b -o PATH/SAMPLE_raw.bcf -f PATH/ref_genome.fasta PATH/output.sorted.bam

# STEP 9: Detect SNPs

bcftools call --ploidy 1 -m -v -o PATH/bcf/SAMPLE_variants.vcf PATH/bcf/SAMPLE_raw.bcf

# STEP 10: Filter and report SNPs in VFC

vcfutils.pl varFilter PATH/bcf/SAMPLE_variants.vcf  > PATH/vcf/SAMPLE_final_variants.vcf
```

Sample Automated Variance Calling Pipeline:

- Step 1: Project Management (ProjectTemplate.sh)
- Step 2: Data Download (DataDownload.sh)
- Step 3: VC Pipeline (FullVCPipeline.sh)

Long-Term Evolution Experiment with *E. coli:*


1%
1%
1%
24 hours
~6.67 generations
24 hours
~6.67 generations
Year 1990 1995 2000 2005 2010 2015
Generations (1000s) 5 10 15 20 25 30 35 40 45 50 55 60 65
Ancestral Clone
Samples frozen every 500 generations

Blount et al. 2018

Wikipedia

- Glucose-limited media
- High concentration of citrate
  - Normally not usable under aerobic conditions.
  - Between generations 31,000 and 31,500, a spontaneous citrate using mutant appeared
  - Certain regions of the genome became hypermutable

Our data:

- Subset of the larger experiment
- We will work with three sample events of the Ara-3 strain
    - Generations 5,000, 15,000, and 50,000
- Did the population change through the experiment?

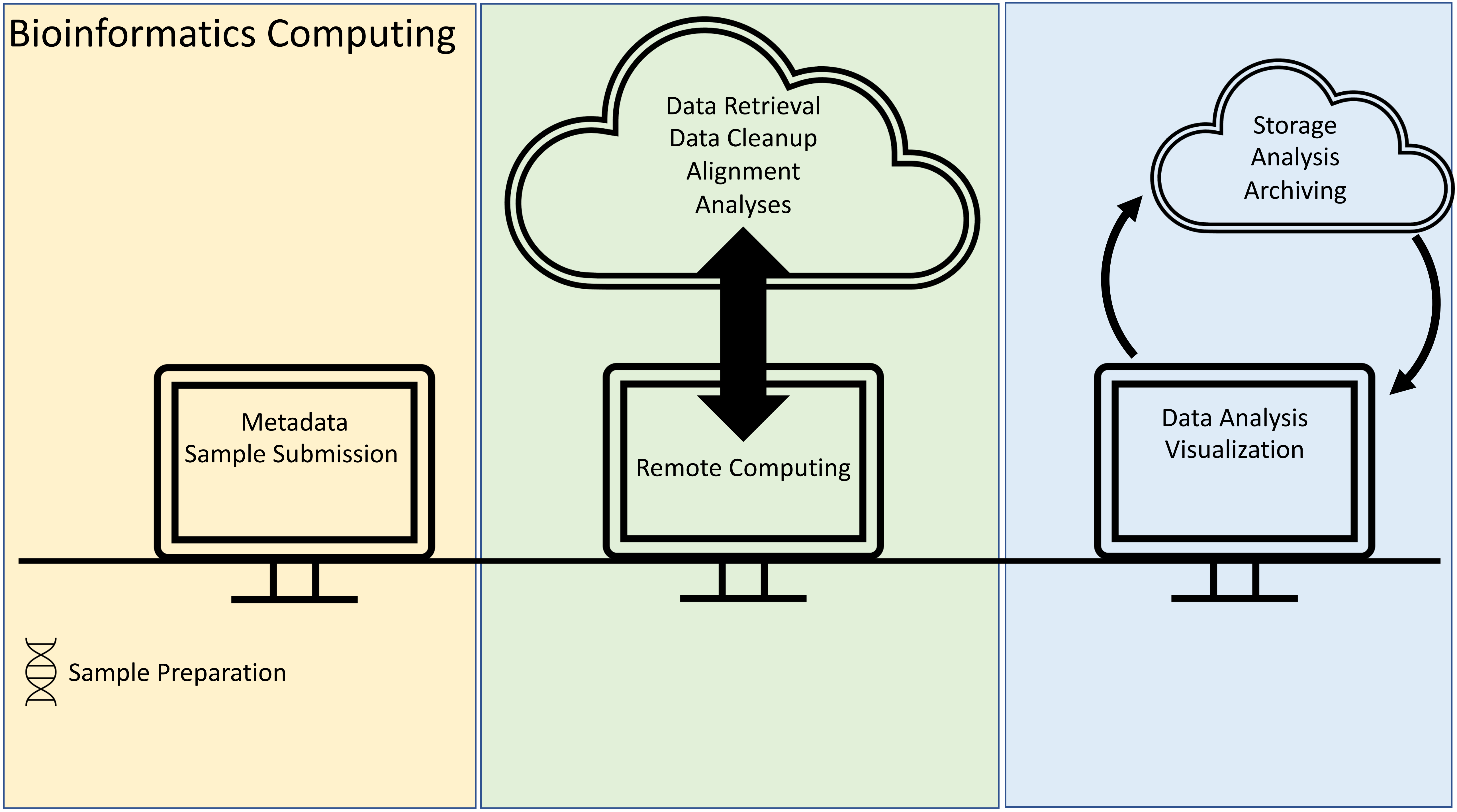
So far:

- Project Organization
- Background and metadata
- Assessing sequencing read quality
- Trimming and filtering reads based on read quality
- Variant calling workflow
- Automating a variant calling workflow
- Cloud computing

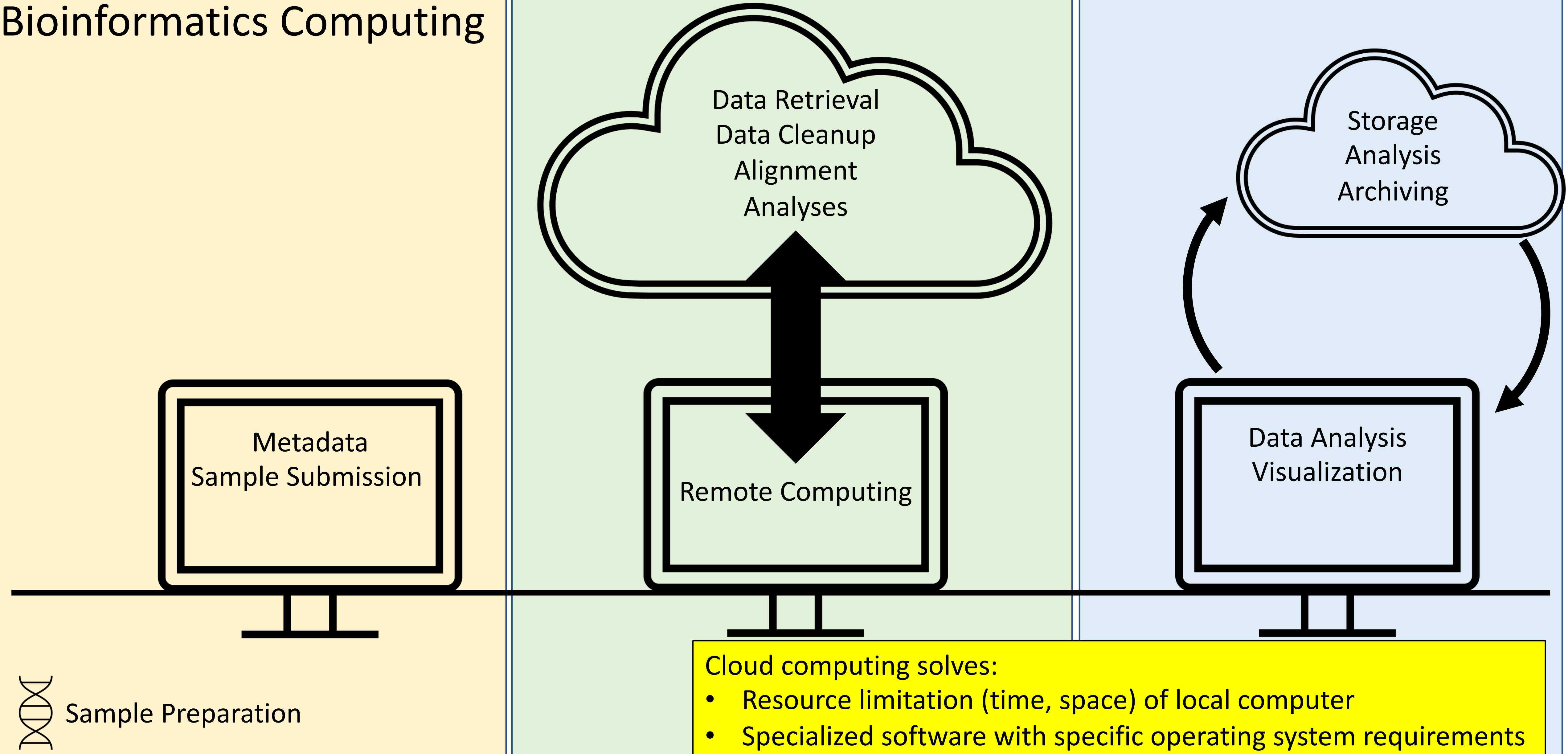
Objectives:

- What is cloud computing?
- What are the tradeoffs of cloud computing?
- What are the benefits?

Bioinformatics Computing



Bioinformatics Computing



Cloud computing solves:

- Resource limitation (time, space) of local computer
- Specialized software with specific operating system requirements
- Access to data from multiple computers
- Continued computing while you are away
- Parallel computing

Choosing a cloud platform:

- Advantages –
  - Access large amounts of computing power on demand
  - Full administrative rights (can install anything)
  - Use pre-configured “images” (machine snapshots where operating system and software are already installed)
  - Your local operating system doesn’t matter – once you connect to the cloud you can run any UNIX software
- Disadvantages –
  - Takes time to upload data and download results
  - Cloud computing costs money (you must keep track of your costs)
  - If you need help, you may not have a system administrator
  - Images may be poorly documented
  - Form of payment is required
  - Understanding Amazon’s billing and payment: https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-getting-started.html

  - Data sensitivity is an important consideration: Are you working with human data?

Logging onto Cloud:

- If you are interested in launching your own instance: https://datacarpentry.org/cloud-genomics/LaunchingInstances/

Choosing a cloud platform:

- Scientific clouds can be free or allocate resources competitively
- Commercial clouds can be powerful, but there are many choices and can be expensive
  - More flexibility usually means more expense

Choosing a cloud platform:
- Scientific clouds can be free or allocate resources competitively
- Commercial clouds can be powerful, but there are many choices and can be expensive
    - More flexibility usually means more expense

Three main platform types:
- University/Corporate Computing Clusters
- Open Science Clouds
- Commercial Clouds

University/Corporate Computing Clusters (High Performance Computing Clusters, HPCCs)

- Often available to students and staff at low or no cost
- Limited by:
    - Number of processors a user can use at once
    - Amount of disk storage per user
    - Amount of time a single process can run
    - What programs can be installed
    - Who can have accounts and access data

University/Corporate Computing Clusters (High Performance Computing Clusters, HPCCs)

- Often available to students and staff at low or no cost
- Limited by:
  - Number of processors a user can use at once
  - Amount of disk storage per user
  - Amount of time a single process can run
  - What programs can be installed
  - Who can have accounts and access data
- Shared Resource:
  - Jobs don’t always run immediately
  - Requesting the right resources (how many processors and how long) can take some trial and error

University/Corporate Computing Clusters (High Performance Computing Clusters, HPCCs)

- Often available to students and staff at low or no cost
- Limited by:
    - Number of processors a user can use at once
    - Amount of disk storage per user
    - Amount of time a single process can run
    - What programs can be installed
    - Who can have accounts and access data
- Shared Resource:
    - Jobs don’t always run immediately
    - Requesting the right resources (how many processors and how long) can take some trial and error
- Advantages:
    - Generally least expensive
    - Often have free or low-cost training
    - Technical support
    - Often have a scheduler system that allows several jobs to be queued and the idle jobs don’t accumulate charges

Open Science Clouds

- XSEDE: https://www.xsede.org
- Open Science Grid: https://opensciencegrid.org
- Open Science Data Cloud (OSDC): https://www.opensciencedatacloud.org

- Others: https://datacarpentry.org/cloud-genomics/04-which-cloud/index.html

- Advantages:
    - Support for each is typically available
    - Can be less expensive than commercial cloud computing resources
- Disadvantage:
    - Resources are allocated on a competitive basis (requiring an application)
    - You often need to have a specific idea of how much of the resource you need for your analysis, which can be hard to calculate

Commercial Clouds:

- Amazon Web Services (AWS): https://aws.amazon.com/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc
- Google Cloud Platform (GCP): https://cloud.google.com
- Microsoft Azure: https://azure.microsoft.com/en-us/

- Operate by renting or provisioning resources
- Costs are generally comparable, choice often comes down to:
  - Funding options (may provide options for researchers to compute for free via credits)
  - Collaborations (what do others in your field use?)
  - Location of datasets (collaborator and research group-specific)
  - Familiarity (similarities between remote computing you've had experience with)

- Advantages:
  - Essentially "on-demand" access to resources
  - Some resources are free for testing
- Disadvantages:
  - Can be extremely costly
  - You are paying for time that your instance is sitting idle

Goals for Today and Friday:

- Introduce R and RStudio
- R basics
- Analyzing data
- RNAseq data analysis
- Producing reports

Favorite R resources:
Cheatsheets: https://rstudio.com/resources/cheatsheets/
Plotting: https://www.data-to-viz.com

R Syntax

`command()` → `getwd()`

`command(OBJECT)` → `help.search("search term")`

`""` (quotes): Typically needed when calling up a package name or a

Exercise: What do these functions do?

Try the following functions by writing them in your script. See if you can guess what they do, and make sure to add comments to your script about your assumed purpose.

- `dir()`
- `sessionInfo()`
- `date()`
- `Sys.time()`

R Syntax for Creating/Storing objects

```
NAME <- object
```

```
a <- 1
b <- “bat”
B <- b
```

Guidelines:

- No quotes to save numbers to objects
- Use “” to save character information to an object (can be double or single typically)
- No quotes with characters assumes that the characters are already objects

Exercise: Create some objects in R

Create the following objects; give each object an appropriate name (your best guess at what name to use is fine):

1. Create an object that has the value of number of pairs of human chromosomes
2. Create an object that has a value of your favorite gene name
3. Create an object that has this URL as its value: “ftp://ftp.ensemblgenomes.org/pub/bacteria/release-39/fasta/bacteria_5_collection/escherichia_coli_b_str_rel606/”
4. Create an object that has the value of the number of chromosomes in a diploid human cell

Naming objects in R:

- Avoid spaces and special characters (use _, .)
- Use easy to understand names (but not overly long)
- Avoid commonly used names (min, max, mean, c)
- Use the recommended R assignment operator (<- not =). = is usually used to identify arguments in functions

Style Guide:
https://style.tidyverse.org/index.html

Exercise: Create objects and check classification with class()

1. chromosome_name <- ‘chr02’
2. od_600_value <- 0.47
3. chr_position <- ‘1001701’
4. spock <- TRUE
5. pilot <- Earhart

R Syntax for Vectors

Function c() is used to concatenate or combine elements

```
NAME <- c(thing1, thing2, thing3)
```

```
snp_genes <- c("gene1","gene2")
```

Guidelines:

- No quotes to save numbers to objects
- Use “” to save character information to an object (can be double or single typically)
- No quotes with characters assumes that the characters are already objects

Review Exercise 4

Using indexing, create a new vector named `combined` that contains:

- The the 1st value in `snp_genes`
- The 1st value in `snps`
- The 1st value in `snp_chromosomes`
- The 1st value in `snp_positions`

read.csv()

Does your table have column names?

Name of your file

```
function (file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", ...)
```

What symbol separates each data entry?

, = comma-separated

\t = tab-separated

; = semicolon-separated

Subsetting data frames:

Square brackets method:

```
DATAFRAME[row, column]
```

Chr	Gene	Position	Allele
chr2	Ccs	3000200	A
chr2	MTF1	4003910	C
chr3	ATOX1	5183027	G

```
DATAFRAME[3,3]
4003910
```

Subsetting data frames:

Square brackets method:

```
DATAFRAME[row, column]
```

Chr	Gene	Position	Allele
chr2	Ccs	3000200	A
chr2	MTF1	4003910	C
chr3	ATOX1	5183027	G

```
DATAFRAME[DATAFRAME$Chr == “chr2”]
```

Subsetting data frames:

Subset function method:

```
subset(DATAFRAME, COLUMN_NAME == “VALUE”)
```

Chr	Gene	Position	Allele
chr2	Ccs	3000200	A
chr2	MTF1	4003910	C
chr3	ATOX1	5183027	G

```
subset(DATAFRAME, Chr == “chr2”)
```

Chr	Gene	Position	Allele
chr2	Ccs	3000200	A
chr2	MTF1	4003910	C

R Syntax for dplyr select

```
select(DATAFRAME, col1, col2, ...)
```

R Syntax for dplyr pipes

Pipe symbol

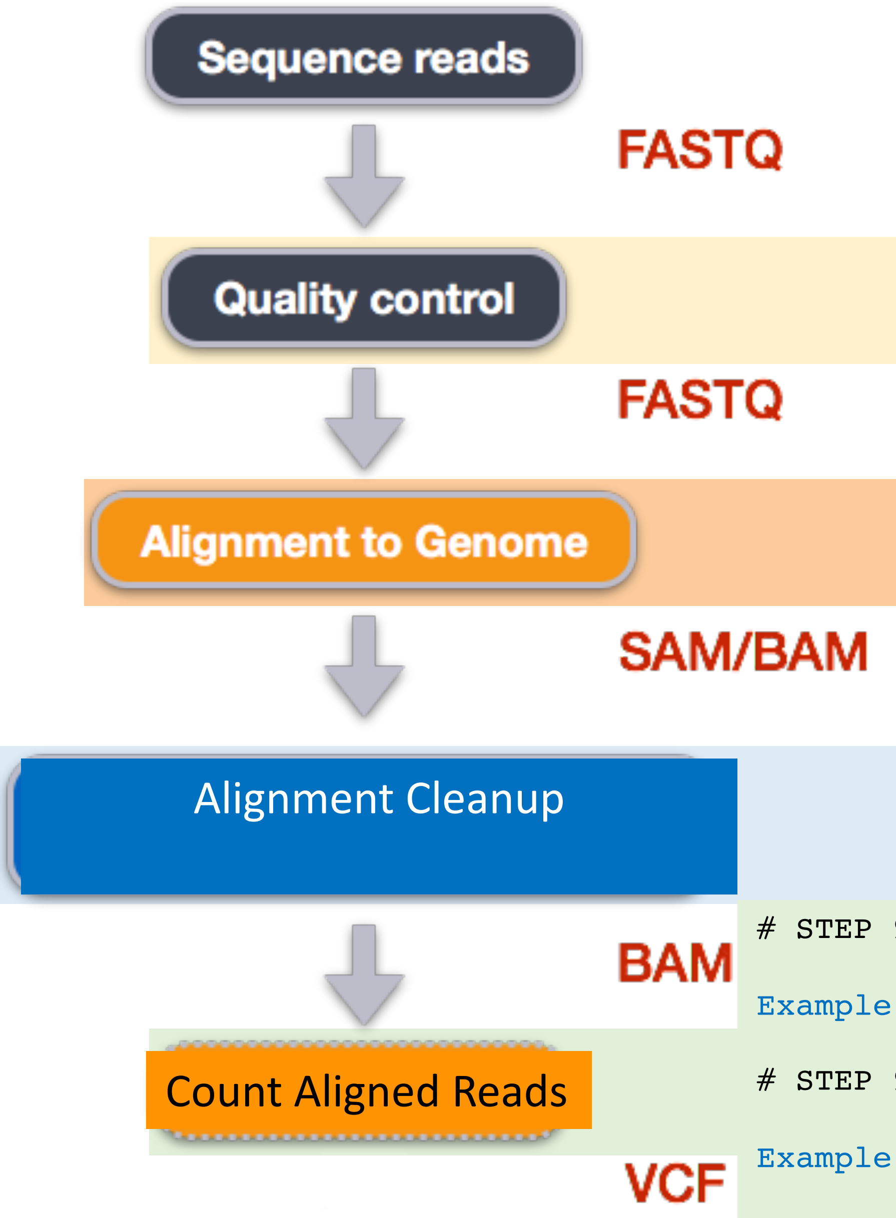
```
DATAFRAME %>%
      filter(sample_id == "SRR2584863") %>%
      select(REF, ALT, DP) %>%
      head()
```

Goals for Wednesday and Friday:

- Introduce R and RStudio
- R basics
- Analyzing data
- RNAseq data analysis
- Producing reports

Favorite R resources:
Cheatsheets: https://rstudio.com/resources/cheatsheets/
Plotting: https://www.data-to-viz.com

Example RNAseq Pipeline:



```
# STEP 1: FASTQC raw data

# STEP 2: TRIM and FILTER

# STEP 3: FASTQC filtered data

# STEP 4: Index reference genome

# STEP 5: Align trimmed reads

# STEP 6: Convert from SAM to BAM

# STEP 7: Sort BAM file

# STEP 9: Generate count-based measure of gene expression

Example: featureCounts (outputs number of reads associated with a feature (gene) of interest

# STEP 9: Differential expression analysis

Example: edgeR
```