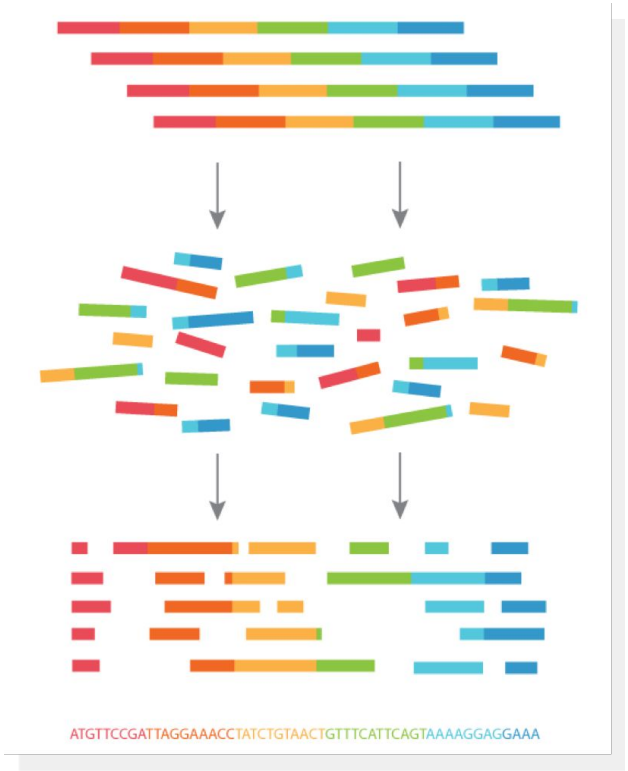


The basic principles of shotgun metagenomics (MGX)

Bram van Dijk - MPI Lecture hall - 24-04-2023

Introduction round

Kristian Ulrich
(IT, course preparation)

Pauline Buffard
(workshop assistant / guinea pig)

Acknowledgements

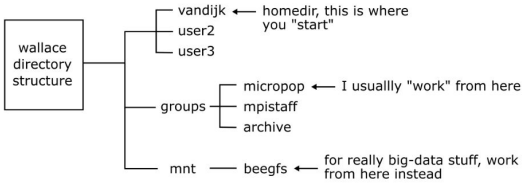
Laura Dijkhuizen

Tina Hauptfeld

Bas Dutilh

Before I even start talking MGX...

Logging in to Wallace: ssh vandijk@wallace



So, to do some stuff in your own directory in micropop:

```
$ cd /groups/micropop
$ mkdir my_directory
$ cd my_directory
$ bash my_script.sh
```

Scripting with Bash (101)

- create/open new script, e.g. nano my_script.sh
- first line should always be: #!/bin/bash
- example script:

```
#!/bin/bash
echo "Start script"
for file in *.fastq.gz; do
   gunzip $file
   echo "Done unzipping $file"
done;
echo "All files are unzipped. Hoping for the best!"
```

To run the script:

```
$ bash my_script.sh
```

Pro tip: if you have a slow script (e.g. one that copies a lot of files), first create a "dummy" script that simply prints the commands without executing them by adding "echo" in front of the steps.

Characters with special meanings:

.	current directory
../	parent directory
../../	parent directory of parent directory
/	root directory
>	pipe output to a file (e.g. ls *.fastq > all_reads.txt)
>>	pipe output to file (append)
|	pipe output to another program (e.g. ls *.fastq | grep "1A")
<	reverse pipe (rarely used, but you may encounter it)
&	when typed after a command, it will put the process "in the background" meaning you will get your prompt back and can type more commands while the other program is running.
$	access a variable (e.g. echo $USER)
$()	store output in variable (e.g. variable=$(command_here))

Useful commands:

command	description	example
cd	change directory	cd /groups/micropop
ls	list content of dir	ls my/data
cp	copy a file	cp myfile.txt /groups/micropop
mv	move (or rename) file	mv myfile.txt myfile_renamed.txt
cat	print file content	cat myfile.txt
wc	count words, lines or chars	wc -l myfile.txt
grep	find lines with pattern	grep "read_001" myfile.txt
cut	split by delimiter	cat myfile.txt | cut -f2
nano	command-line text editor	nano my_script.sh
gedit	graphical text editor	gedit my_script.sh
sort	sort input	cat myfile.txt | sort -n
uniq	remove duplicate items	cat myfile.txt | sort -n | uniq
mkdir	make new directory	mkdir /groups/micropop/b_obama
echo	print something	echo "Hello!"
gzip	zip file	gzip reads.fastq
gunzip	unzip file	gunzip reads.fastq

Example of a chain of commands:

```
ls *.fastq | grep "1A" | cut -d '_' -f2 > all_samples_with_1A.txt
```

Why are MGX “workshops” hard?

Workshops usually have much more of a hands-on feel

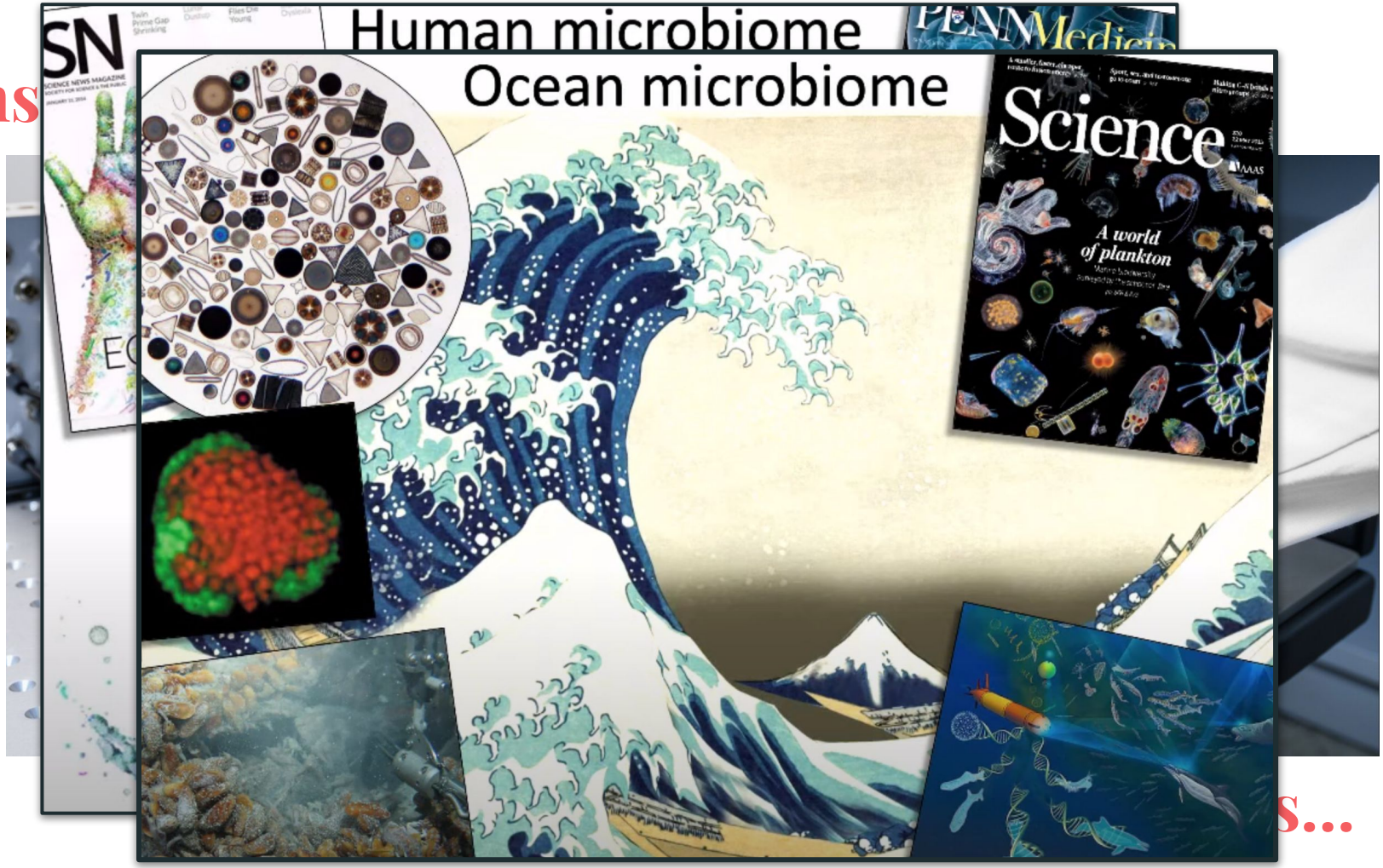
but…

MGX is simply too slow for that.

but…

Let’s at least keep it interactive

Ins


Human microbiome
Ocean microbiome
Science
A world of plankton

s...

Problems with culturing bacteria in the lab

- Hard to find conditions for cultivation
  (super specific conditions for thousands of species)
- Closed systems may be misleading
  (who are we to say what does/doesn't interfere?!)
- Spatial structure matters, and culturing interferes with this!
  (single isolates growing in circles is not solving this problem…)

How to study microbes *in situ*?

- What we need, are "snapshots" from natural systems. If you make a lot of snapshots, you may even compile a movie!
- The most obvious "snapshot" is ofcourse microscopy, but it can only tell you so much …
- MGX tries to make snapshots of “information” → sensu bioinformatics 1972

Bioinformatics?

- “Modern” definition of bioinformatics: the science of collecting and analysing complex biological data such as genetic codes
- Historically, the term bioinformatics did not mean what it means today
- Paulien Hogeweg and Ben Hesper coined bioinformatics to refer to the study of information processes in biotic systems

What this workshop is about (3 goals)

(modern) Bioinformatic skills
(scripting, installing tools, etc.)

Using standard pipelines
(*e.g.* Anvio)

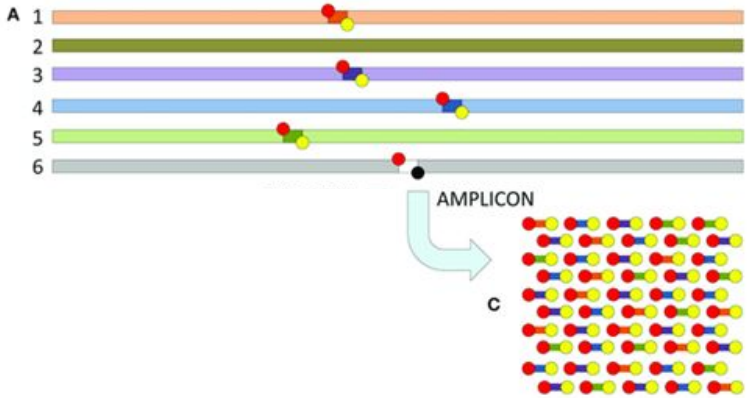
- **Lectures** are focussed on basic steps, concepts and nomenclature. This will get you to the level of a "good collaborator" → you know what you can/can't do with MGX
- The **syllabus/practicals** are meant as “finger exercises”, knowing how to do the steps, and are mostly introductory
- **Consultancy**. I am leaving soon, so get all your questions out . You can email me in the future, but now is the best time to talk about your data! :)

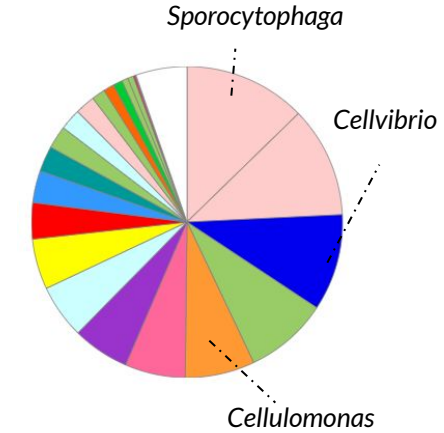
Workshop overview

- Two morning lectures, today and tomorrow (10-12.30)
  > *Today will focus on basic principles*
  > *Tomorrow will focus on HGT and how to detect it*

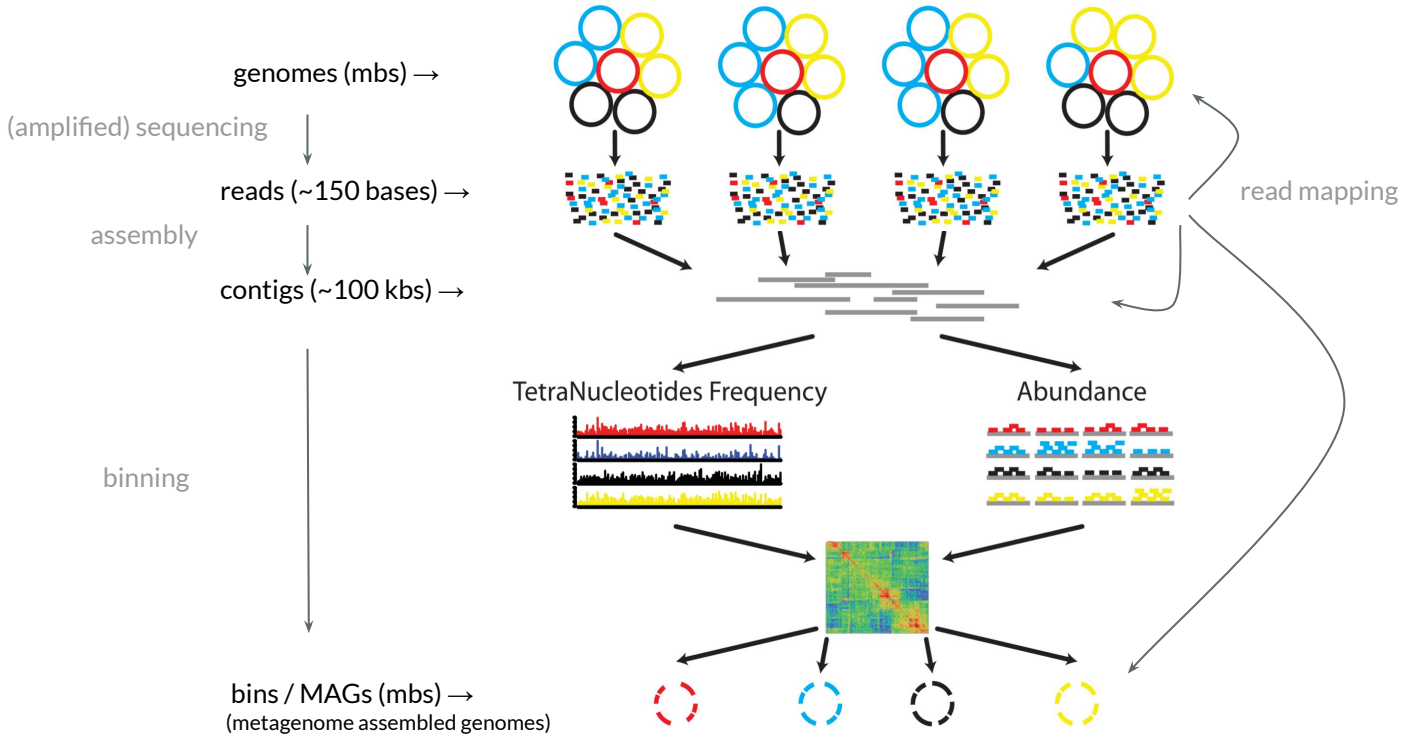
- Two afternoon sessions of hands-on stuff.
  > *Work through the syllabus that helps you get started with MGX*
  > *Try out the xenoseq pipeline for the H/V experimental designs*
  > *Bring your data, ask questions!*

Annotation is often the primary focus: 16S

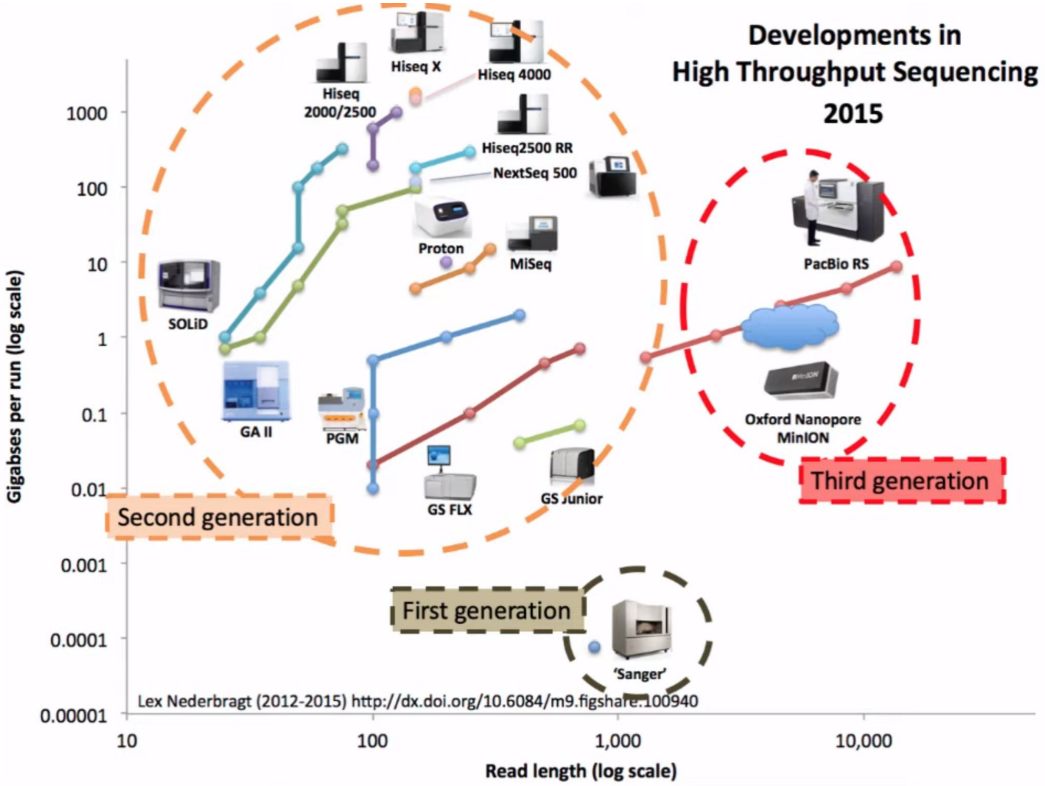




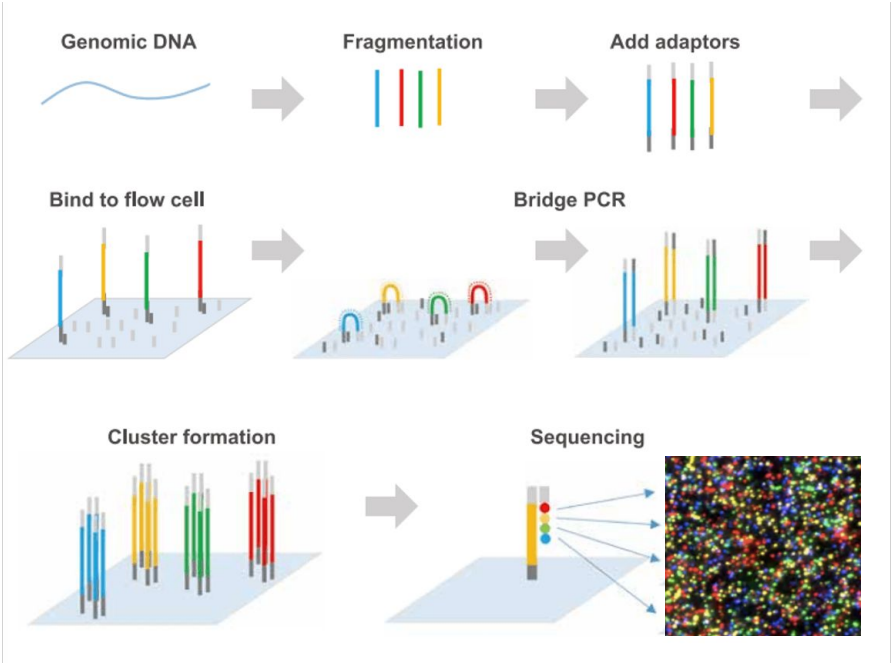
With shotgun MGX, a lot happens before annotation

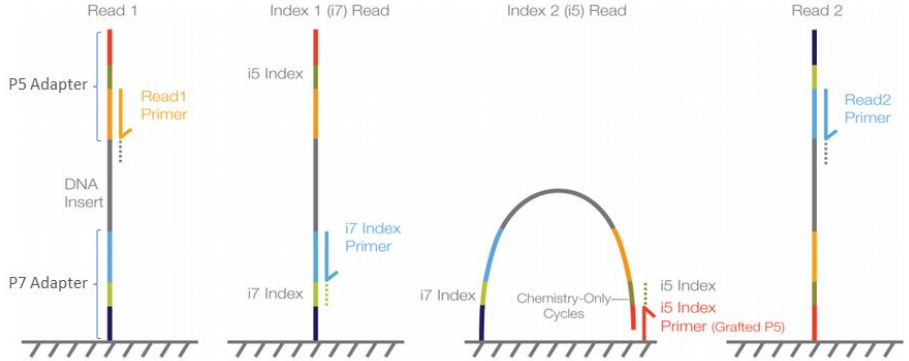


“Next” generation sequencing



Brief recap of illumina sequencing





1. The DNA fragment is not the read
2. Two adapters allow reading in both directions
3. Insert is the bit that isn't sequenced
4. "Quality" → confidence

A fasta file ....

>header sequence

```
>VH00578:2:AAAJVKHHV:1:1101:27055:9216 1:N:0: TCTTATTA+GGCAACCT
GGGCCGCTGGGCGGCCCCGTGTAGCCTTACTGCTGCGAAGCTTCGAACACCTTGCGGAACTCGGCTTTTTCTTCATCGG
TCT
>VH00578:2:AAAJVKHHV:1:1101:32963:9216 1:N:0:TCTTATTA+GGCAACCT
AAGTATGAGTGAGCCCGTTGACAATAGGGTTGTGATATTTTTCTGCTTGATGTTGTATTGGTCCAACTTTTCCTTTAAA
ACT
>VH00578:2:AAAJVKHHV:1:1101:58072:9216 1:N:0:TCTTATTA+GGCAACCT
GTCTGCACATTGCCGGACAAATTGCCAGAACCCTCAGTCCGGTGCCTGATAAAAACAGGAAGCTCCTGATAACCGGCGG
TCC
>VH00578:2:AAAJVKHHV:1:1101:24499:9235 1:N:0:TCTTATTA+GGCAACCT
CAGTATACCTACACTATCAGCACACCAACGCTTCTCCATTCTCCTCAAACCACTCTCCGCCATCTTAGCAGCCATGCCA
CTC
>VH00578:2:AAAJVKHHV:1:1101:28892:9235 1:N:0:TCTTATTA+GGCAACCT
TCATAATAATTCGCAAACTATATCCGGGTAGAGGTCCGGATGTCAACCAAAAATATACACTGTGAGCTGGATCGAACGC
AAG
>VH00578:2:AAAJVKHHV:1:1101:31240:9235 1:N:0:TCTTATTA+GGCAACCT
GAGTTCGATGCCCTGGACGAGCTGCGCGGCTTCGCCGCCACCCTCGGCCGGCGCCTGGGCGAGATGCATCAGGTGCTCG
```

Raw s

```
@MM123:002:FC123AB:3:2208:3330:9840 2:Y:18:ATCACG
AGGATACTAGCATAGATACCCTAGATAGTCATAGATCATGATAGGGAGATCTA
+
IJJJJJJIIIIIJIIIIIFFFEEEEEDDDDDDCABBBBB@@00))))*(*&%!
```

@header

@VH00

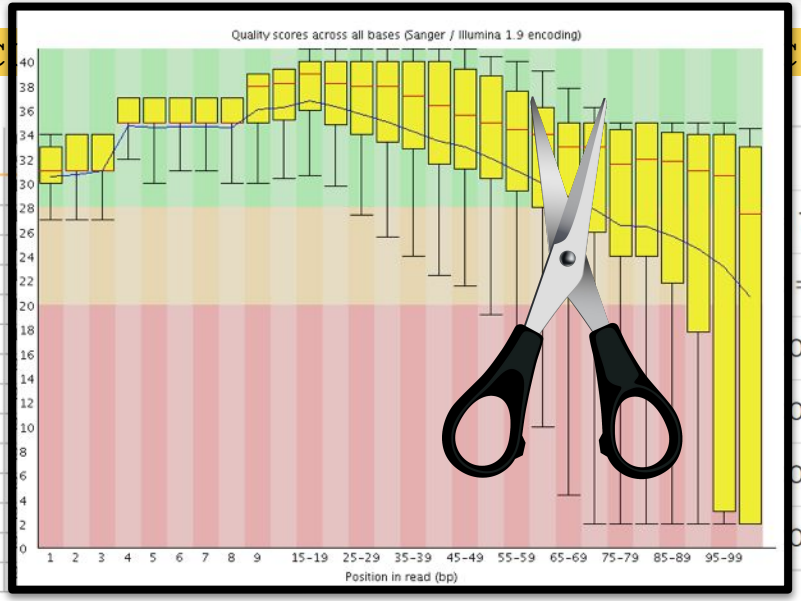
GGGCCGCTGGGCGGCCCCGTGTAGCCTTACTGCTGCGAAGCTTCGAACACCTTGCGGAACTCGGCTTTTTCTTC

+

CCCCCCCCCCCCCCCCCCC ;CCCCCCC-CCCCC;-CCCCCC

Table 1 ASCII Characters Encoding Q-scores 0–40

Symbol	ASCII Code	Q-Score	Symbol	ASCII Code
!	33	0	/	47
"	34	1	0	48
#	35	2	1	49
$	36	3	2	50
%	37	4	3	51
&	38	5	4	52
'	39	6	5	53
(	40	7	6	54
)	41	8	7	55
*	42	9	8	56
+	43	10	9	57
,	44	11	:	58
-	45	12	;	59
.	46	13	<	60



Error	Accuracy (1 - Error)
10%	90%
= 1%	99%
0 = 0.1%	99.9%
00 = 0.01%	99.99%
000 = 0.001%	99.999%
0000 = 0.0001%	99.9999%

Fastq ➳ Anvio: “Ready-to-use” pipeline

Anvio is a great tool to do “everything”

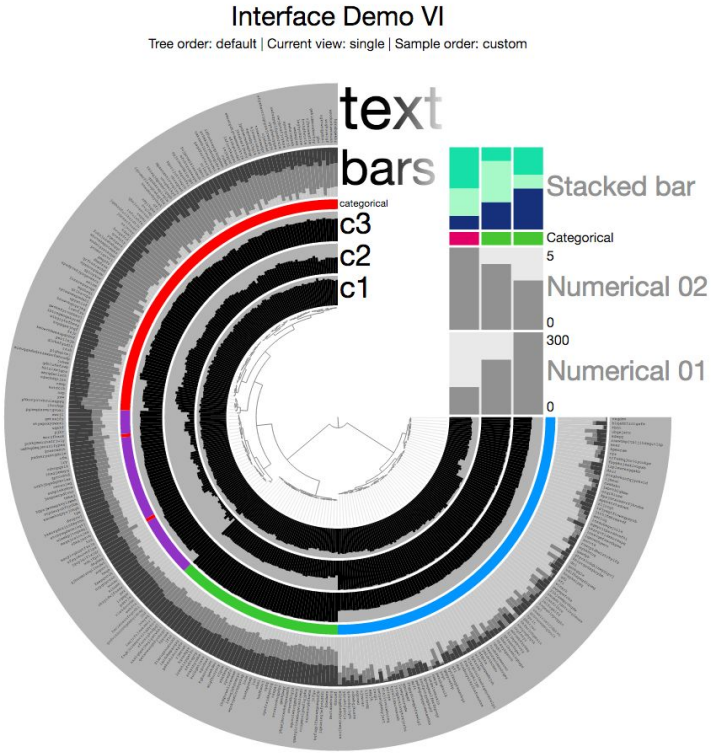
https://anvio.org/

Why I won’t get into Anvio

“Doing everything” doesn’t imply “getting all the knowledge”
I want you to know what you’re doing first
It’s not as easy-to-use as advertised
I am not an expert on Anvio

Other tools

MetaWrap, Squeezemeta, MG-Rast (online!), nf-core-MAG



Bash (bourne-again shell)

- Bash is integral to everything you do in the terminal:
  ls to list files, cp to copy files, mv to move file, *etc.*
- You can write a series of these commands in a BASH script
- For example:

```
#!/bin/bash

echo "Let's roll a 6-sided die"

# generate a random number between 1 and 6
die=$((RANDOM % 6 + 1)) # e.g. if RANDOM is 13, modulo 6 equals 1, plus 1 equals 2

# print the result
echo "You threw a $die"
```

Write me a bash script that loops through all files ending in ".fastq.gz" and unzips the files to a new directory called "unzipped_reads".

Here's a bash script that should do what you're asking for:

bash Copy code

```bash
#!/bin/bash

# create the "unzipped_reads" directory if it doesn't already exist
mkdir -p unzipped_reads

# loop through all files in the current directory ending in ".fastq.gz"
for file in *.fastq.gz
do
    # extract the base filename without the ".fastq.gz" extension
    base=$(basename "$file" .fastq.gz)

    # unzip the file to the "unzipped_reads" directory with the same base fi
    gunzip -c "$file" > "unzipped_reads/${base}.fastq"
done
```

This script first creates the "unzipped_reads" directory using `mkdir -p` to avoid errors in case the directory already exists. It then loops through all files in the current directory that end with ".fastq.gz", and for each file it extracts the base filename without the ".fastq.gz" extension using the `basename` command. It then unzips the file to the "unzipped_reads" directory with the same base filename using `gunzip -c`. The `-c` flag tells `gunzip` to write the output to stdout instead of a file, and the output is redirected to a file with the same base filename and the ".fastq" extension in the "unzipped_reads" directory using the `>` operator.

Note that this script assumes that the files are compressed using gzip compression and that the `gunzip` command is available on your system. If your files are compressed using a different compression algorithm or if you don't have `gunzip` installed, you'll need to modify the script accordingly.

Rewrite the script to add the name of a random famous clown to every file.

Sure, here's a modified version of the script that adds a random famous clown's name to the beginning of each unzipped file:

bash Copy code

```bash
#!/bin/bash

# create the "unzipped_reads" directory if it doesn't already exist
mkdir -p unzipped_reads

# list of famous clowns to choose from
clowns=("Bozo" "Ronald McDonald" "Krusty the Clown" "Pennywise" "Emmett Kell

# loop through all files in the current directory ending in ".fastq.gz"
for file in *.fastq.gz
do
    # extract the base filename without the ".fastq.gz" extension
    base=$(basename "$file" .fastq.gz)

    # select a random clown name from the list
    clown=${clowns[$RANDOM % ${#clowns[@]}]}

    # unzip the file to the "unzipped_reads" directory with the clown name a
    gunzip -c "$file" > "unzipped_reads/${clown}_${base}.fastq"
done
```

```
$ ls raw_reads_course
5316_AD_run721_CCTCGCAG_S499_L001_R1_001.fastq.gz  5316_AD_run721_CCTCGCAG_S499_L002_R2_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L002_R1_001.fastq.gz  5316_CX_run721_ACAAAGTG_S565_L001_R2_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L001_R1_001.fastq.gz  5316_EB_run722_TGACAACC_S332_L002_R2_001.fastq.gz
5316_AD_run721_CCTCGCAG_S499_L001_R2_001.fastq.gz  5316_BN_run721_TCTTATTA_S530_L001_R1_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L002_R2_001.fastq.gz  5316_CX_run721_ACAAAGTG_S565_L002_R1_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L001_R2_001.fastq.gz
5316_AD_run721_CCTCGCAG_S499_L002_R1_001.fastq.gz  5316_BN_run721_TCTTATTA_S530_L001_R2_001.fastq.gz
5316_CX_run721_ACAAAGTG_S565_L001_R1_001.fastq.gz  5316_CX_run721_ACAAAGTG_S565_L002_R2_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L002_R1_001.fastq.gz
```

Step 1: the “rosetta stone”

Refers to a **black basalt stone** found in 1799 which is being used to this day to decipher hieroglyphics

```
$ ls raw_reads_course
5316_AD_run721_CCTCGCAG_S499_L001_R1_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L002_R1_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L001_R1_001.fastq.gz
5316_AD_run721_CCTCGCAG_S499_L001_R2_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L002_R2_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L001_R2_001.fastq.gz
5316_AD_run721_CCTCGCAG_S499_L002_R1_001.fastq.gz
5316_CX_run721_ACAAAGTG_S565_L001_R1_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L002_R1_001.fastq.gz
5316_AD_run721_CCTCGCAG_S499_L002_R2_001.fastq.gz
5316_CX_run721_ACAAAGTG_S565_L001_R2_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L002_R2_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L001_R1_001.fastq.gz
5316_CX_run721_ACAAAGTG_S565_L002_R1_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L001_R2_001.fastq.gz
5316_CX_run721_ACAAAGTG_S565_L002_R2_001.fastq.gz
```

```
$ cat rosetta_stone.txt
5316_AD    T1_C1_7amp_H
5316_BN    T2_C1_7amp_H
5316_CX    T3_C1_7amp_H
5316_EB    Tminus1_C1_powersoil
```

Step 1: the “rosetta stone”

Pseudo-code of 01_retrieve_data.sh: copy files from the archive to your own directory.

1. Make directory called ‘raw_reads_renamed’
2. Then, go through the rosetta_stone.txt file line by line, doing:

   a. Find **all** file(s) containing “5316_AD” in Wallace’s archive (or for the workshop, from the ‘raw_reads_course’ directory)

   b. Unzip the file(s) to a new directory under the name “T1_C1_7amp_H.fastq”

   c. Go to the next line

Step 1: the “rosetta stone”

(the actual code)

```
#!/bin/bash

#########################################################
# MGX_BASICS - Part 1 - Retrieving data from the archive
#########################################################

# Usually, this path will be emailed to you by Sven, or the company provides these files. E.g. on the archive, this path could be
'/groups/archive/MPGC/project5316/run721'
path_to_fastq_files="raw_reads_course"
dir_renamed_files="raw_reads_renamed"

# Make the directory for the new files (-p prevents error when dir already exists)
mkdir -p $dir_renamed_files

# The line below is a way to loop over a file line by line
cat rosetta_stone.txt | while read line; do
        # First, extract specific bits of the line (old name, new name)
        old_name=$(echo $line | cut -f1)          # First column contains the old name, store in variable
        new_name=$(echo $line | cut -f2)          # Second column contains new name, store in variable

        # Print something to the terminal so we know what's going on
        echo "Reads with prefix $old_name will be unzipped and merged under the new name $new_name (fastq for both R1 and R2)" # Echoing the new name.

        # Unzip (zcat) all files corresponding to the old_name into a single file
        zcat $path_to_fastq_files/${old_name}_*R1*.fastq.gz > ${dir_renamed_files}/${new_name}_R1.fastq
        zcat $path_to_fastq_files/${old_name}_*R2*.fastq.gz > ${dir_renamed_files}/${new_name}_R2.fastq
done;
```

For the workshop ↠ a lot of scripts are already run

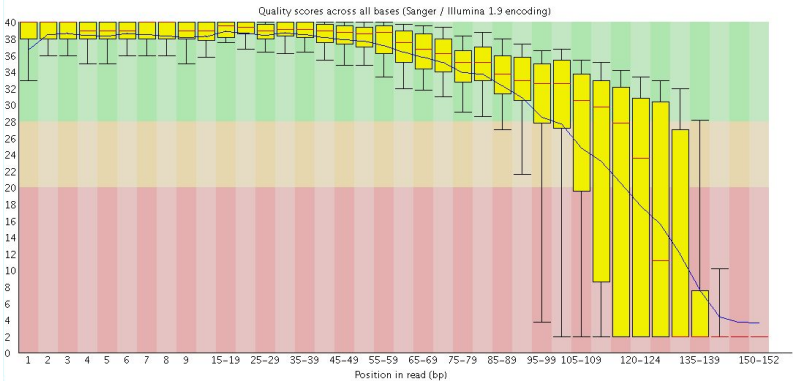
```
vandijk@newnode01_MGX:/groups/micropop/MGX_Workshop_vandijk$ ls -l
total 984
-rwxrwx--- 1 vandijk staffmpi   267 Apr  4 15:37 00_mount_env.sh
-rwxrwx--- 1 vandijk staffmpi   327 Mar 31 13:10 00_setup.rc
-rwxrwx--- 1 vandijk staffmpi   343 Mar 15 13:28 00_setup.sh
-rwxrwx--- 1 vandijk staffmpi  1462 Mar  6 10:51 01_retrieve_data.sh
-rwxrwx--- 1 vandijk staffmpi   927 Mar  8 13:49 02_process_reads.sh
-rwxrwx--- 1 vandijk staffmpi   888 Mar 24 15:12 03_assemble.sh
drwxrwx--- 6 vandijk staffmpi   164 Mar 31 10:54 03_Assembly_output
-rwxrwx--- 1 vandijk staffmpi  1621 Mar 31 11:06 04_map_back.sh
drwxrwx--- 3 vandijk staffmpi    87 Mar 31 15:46 04_Mapping_back
drwxrwx--- 3 vandijk staffmpi   271 Mar 31 15:25 05_Binning
-rwxrwx--- 1 vandijk staffmpi   572 Mar 31 14:05 05_binning.sh
drwxrwx--- 2 vandijk staffmpi   432 Apr  3 11:42 06_Prokka_MAG3
-rwxrwx--- 1 vandijk staffmpi    64 Apr  4 14:49 06_prokka.sh
drwxrwx--- 3 vandijk staffmpi  1385 Apr  4 16:18 07_Contig_annotation
-rwxrwx--- 1 vandijk staffmpi  5786 Apr  4 15:42 07_contig_annotation_tool.sh
-rwxrwx--- 1 vandijk staffmpi    76 Jan 24 10:44 99_nonpareil.sh
drwxrwx--- 2 vandijk staffmpi    43 Mar 24 10:43 99_Other_output
-rwxrwxr-- 1 vandijk staffmpi 11427 Mar  8 13:48 adapter_sequences.fa
-rwxrwx--- 1 vandijk staffmpi  5724 Apr  4 15:35 Contig_Annotation_tool.sh
drwxrwx--- 2 vandijk staffmpi  1072 Mar  6 10:57 raw_reads_course
drwxrwx--- 2 vandijk staffmpi   328 Mar  6 12:01 raw_reads_renamed
drwxrwx--- 2 vandijk staffmpi   592 Mar 24 10:39 reads
-rwxrwx--- 1 vandijk staffmpi    92 Mar  6 11:16 rosetta_stone.txt
```

```
5316_AD_run721_CCTCGCAG_S499_L001_R1_001.fastq.gz
5316_AD_run721_CCTCGCAG_S499_L001_R2_001.fastq.gz
5316_AD_run721_CCTCGCAG_S499_L002_R1_001.fastq.gz
5316_AD_run721_CCTCGCAG_S499_L002_R2_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L001_R1_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L001_R2_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L002_R1_001.fastq.gz
5316_BN_run721_TCTTATTA_S530_L002_R2_001.fastq.gz
5316_CX_run721_ACAAAGTG_S565_L001_R1_001.fastq.gz
5316_CX_run721_ACAAAGTG_S565_L001_R2_001.fastq.gz
5316_CX_run721_ACAAAGTG_S565_L002_R1_001.fastq.gz
5316_CX_run721_ACAAAGTG_S565_L002_R2_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L001_R1_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L001_R2_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L002_R1_001.fastq.gz
5316_EB_run722_TGACAACC_S332_L002_R2_001.fastq.gz
```

```
T1_C1_7amp_H_R1.fastq
T1_C1_7amp_H_R2.fastq
T2_C1_7amp_H_R1.fastq
T2_C1_7amp_H_R2.fastq
T3_C1_7amp_H_R1.fastq
T3_C1_7amp_H_R2.fastq
Tminus1_C1_powersoil_R1.fastq
Tminus1_C1_powersoil_R2.fastq
```

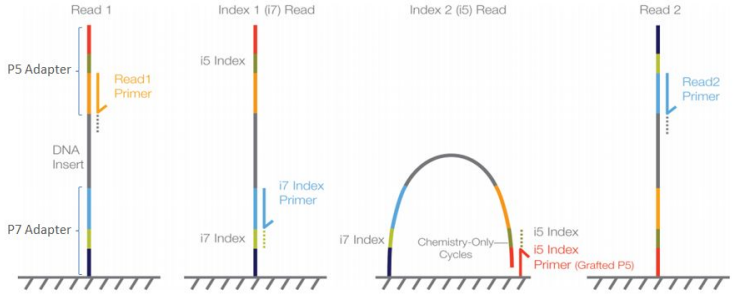
Step 2: read trimming: cleaning up raw reads

Trim the poor-quality / poly-G ends:



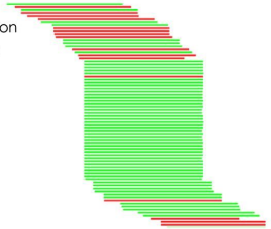
Note: sequencing devices or companies often already do some preliminary cleaning for you!

Remove (remaining) adapters:



Remove PCR duplicates:

- Occur during library preparation
- Don't add unique information
- Optical duplicates



FastQC/FastP: tools that do all of that

1. Generate nice reports of how good your reads are

2. Don’t expect “perfect” reads, every sequencing technology has a few consistent things they perform bad at.

→ We’ll look at a fastp report together in a minute

Step 2: read trimming

```
vandijk@newnode01_MGX:/groups/micropop/MGX_Workshop_vandijk$ ls -l
total 984
-rwxrwx--- 1 vandijk staffmpi   267 Apr  4 15:37 00_mount_env.sh
-rwxrwx--- 1 vandijk staffmpi   327 Mar 31 13:10 00_setup.rc
-rwxrwx--- 1 vandijk staffmpi   343 Mar 15 13:28 00_setup.sh
-rwxrwx--- 1 vandijk staffmpi  1462 Mar  6 10:51 01_retrieve_data.sh
-rwxrwx--- 1 vandijk staffmpi   927 Mar  8 13:49 02_process_reads.sh
-rwxrwx--- 1 vandijk staffmpi   888 Mar 24 15:12 03_assemble.sh
drwxrwx--- 6 vandijk staffmpi   164 Mar 31 10:54 03_Assembly_output
-rwxrwx--- 1 vandijk staffmpi  1621 Mar 31 11:06 04_map_back.sh
drwxrwx--- 3 vandijk staffmpi    87 Mar 31 15:46 04_Mapping_back
drwxrwx--- 3 vandijk staffmpi   271 Mar 31 15:25 05_Binning
-rwxrwx--- 1 vandijk staffmpi   572 Mar 31 14:05 05_binning.sh
drwxrwx--- 2 vandijk staffmpi   432 Apr  3 11:42 06_Prokka_MAG3
-rwxrwx--- 1 vandijk staffmpi    64 Apr  4 14:49 06_prokka.sh
drwxrwx--- 3 vandijk staffmpi  1385 Apr  4 16:18 07_Contig_annotation
-rwxrwx--- 1 vandijk staffmpi  5786 Apr  4 15:42 07_contig_annotation_tool.sh
-rwxrwx--- 1 vandijk staffmpi    76 Jan 24 10:44 99_nonpareil.sh
drwxrwx--- 2 vandijk staffmpi    43 Mar 24 10:43 99_Other_output
-rwxrwxr-- 1 vandijk staffmpi 11427 Mar  8 13:48 adapter_sequences.fa
-rwxrwx--- 1 vandijk staffmpi  5724 Apr  4 15:35 Contig_Annotation_Tool.sh
drwxrwx--- 2 vandijk staffmpi  1072 Mar  6 10:57 raw_reads_course
drwxrwx--- 2 vandijk staffmpi   328 Mar  6 12:01 raw_reads_renamed
drwxrwx--- 2 vandijk staffmpi   592 Mar 24 10:39 reads
-rwxrwx--- 1 vandijk staffmpi    92 Mar  6 11:16 rosetta_stone.txt
```

```
T1_C1_7amp_H_R1.fastq
T1_C1_7amp_H_R2.fastq
T2_C1_7amp_H_R1.fastq
T2_C1_7amp_H_R2.fastq
T3_C1_7amp_H_R1.fastq
T3_C1_7amp_H_R2.fastq
Tminus1_C1_powersoil_R1.fastq
Tminus1_C1_powersoil_R2.fastq
```

```
T1_C1_7amp_H_fastp_report.html
T1_C1_7amp_H_trimmed_R1.fastq
T1_C1_7amp_H_trimmed_R2.fastq
T2_C1_7amp_H_fastp_report.html
T2_C1_7amp_H_trimmed_R1.fastq
T2_C1_7amp_H_trimmed_R2.fastq
T3_C1_7amp_H_fastp_report.html
T3_C1_7amp_H_trimmed_R1.fastq
T3_C1_7amp_H_trimmed_R2.fastq
Tminus1_C1_powersoil_fastp_report.html
Tminus1_C1_powersoil_trimmed_R1.fastq
Tminus1_C1_powersoil_trimmed_R2.fastq
```

Let's take a look!

Step 2: read “trimming”

```
@VH00578:2:AAAJVKHHV:1:1101:27055:9216 1:N:0:TCTTATTA+GGCAACCT
GGGCCGCTGGGCGGCCCCGTGTAGCCTTACTGCTGCGAAGCTTCGAACACCTTGCGGAACTCGGCTTTTTCTTC
+
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
@VH00578:2:AAAJVKHHV:1:1101:32963:9216 1:N:0:TCTTATTA+GGCAACCT
AAGTATGAGTGAGCCCGTTGACAATAGGGTTGTGATATTTTTCTGCTTGATGTTGTATTGGTCCAACTTTTCCT
+
CCCCCCC;CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC;CCCCCCCCCCCCCCCCCCCCCCCCCCCC;CCCC
@VH00578:2:AAAJVKHHV:1:1101:58072:9216 1:N:0:TCTTATTA+GGCAACCT
GTCTGCACATTGCCGGACAAATTGCCAGAACCCTCAGTCCGGTGCCTGATAAAAACAGGAAGCTCCTGATAACC
+
CCCCCCCCC;CCCCCCCCCC-CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
@VH00578:2:AAAJVKHHV:1:1101:24499:9235 1:N:0:TCTTATTA+GGCAACCT
CAGTATACCTACACTATCAGCACACCAACGCTTCTCCATTCTCCTCAAACCACTCTCCGCCATCTTAGCAGCCA
…
```

Intermezzo:
Did we sample “deep enough”?

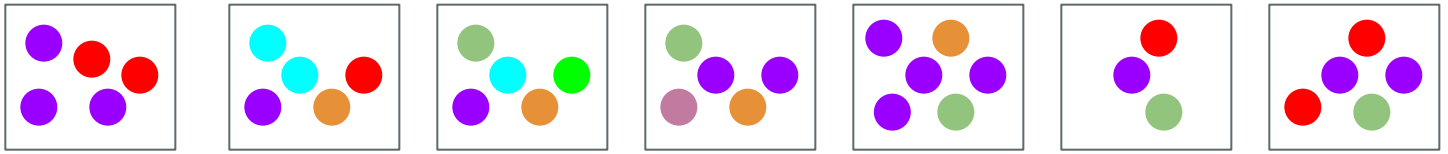
How can we tell if we have “enough reads” ?

1. if you are working with an isolate for which you know the expected genome size, we can calculate this ourselves:

$$\frac{readnumber \cdot readlength}{genomesize}$$

2. however, we often don’t know the genomes size, and we have many **different** players. Now what?

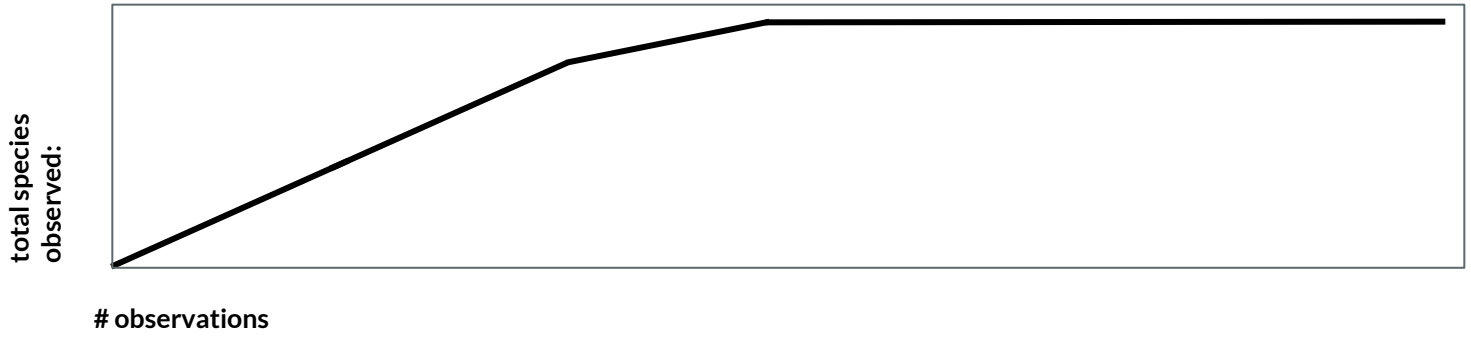
Rarefaction: how ecologists or bird watchers estimate species abundance



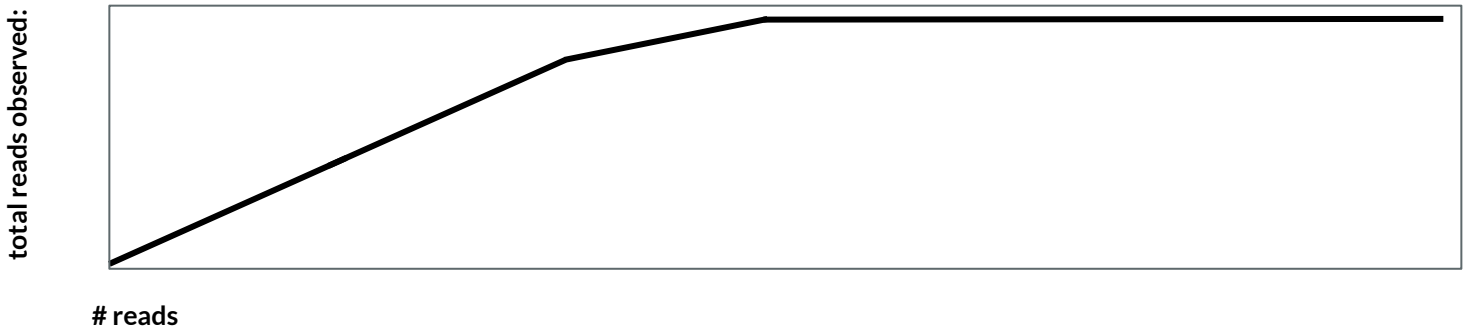
observations:

total species observed:

2 4 6 7 7 7 7

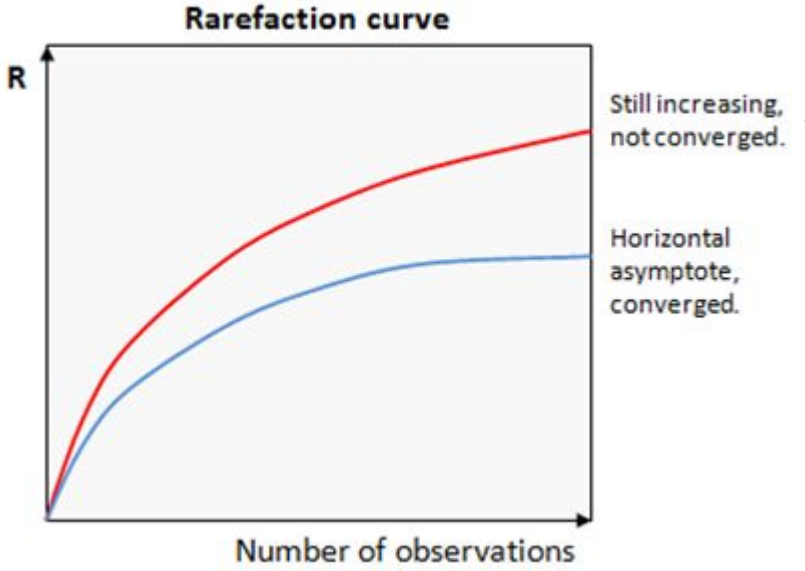


Rarefaction in metagenomics, how to get “observations”?

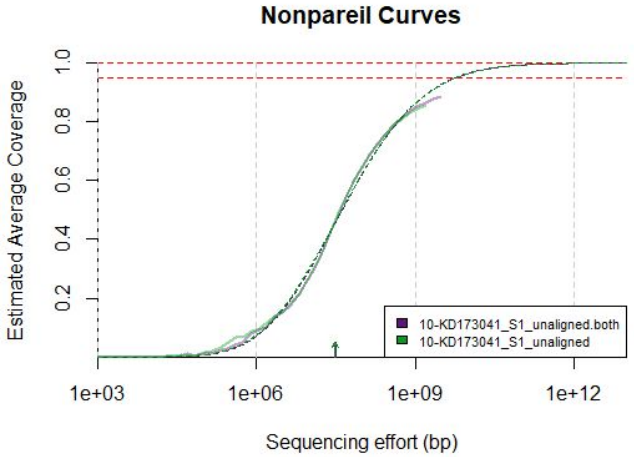


- However, there are 4^{150} possible reads … 2.0e+90. (particles in the universe is ~ 3.0e+80).
- Reads aren’t random, but still…. less combinations would be better…
- Kmers! 24-mers -> 2.8147498e+14

Rarefaction: hopefully, things flatten out



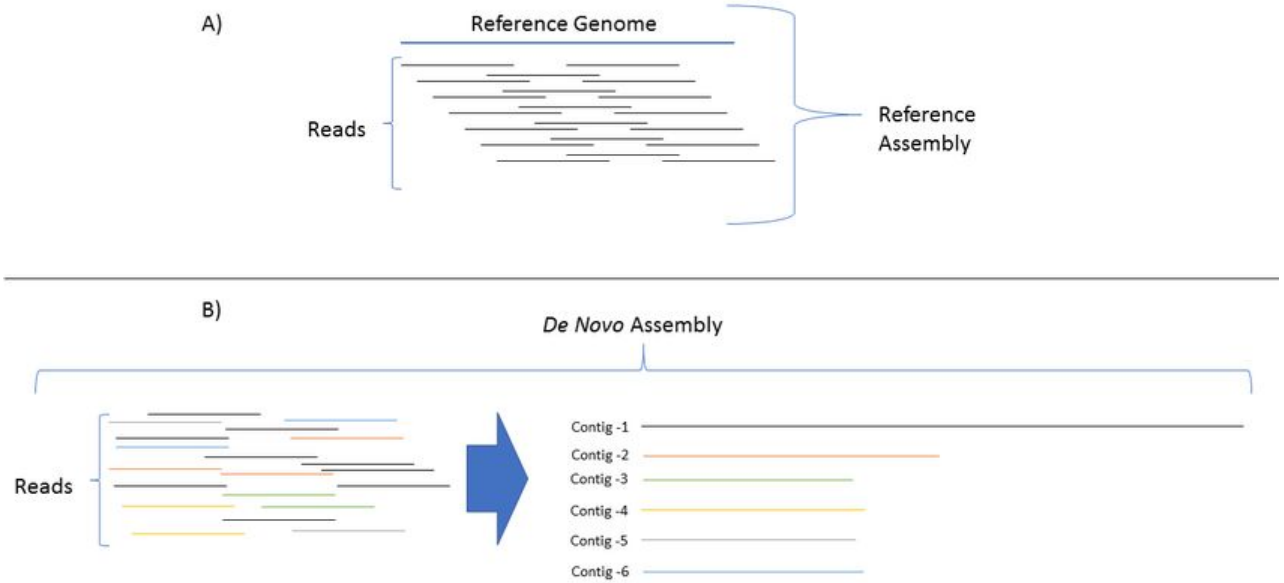
Translated to MGS: if we had more reads, we probably would have “seen more stuff”



This is where it gets interesting

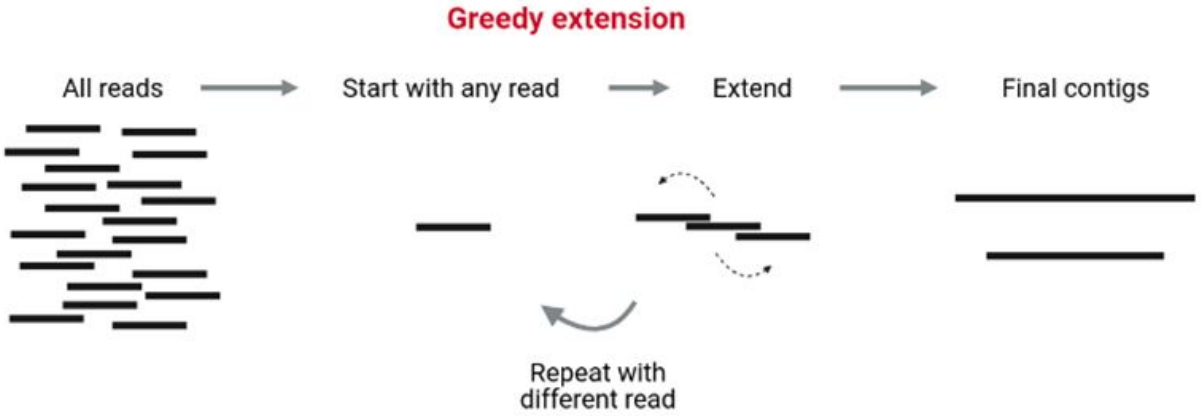
Step 3: assembly of genomes

Two types of assembly: *reference-guided* and *de novo*



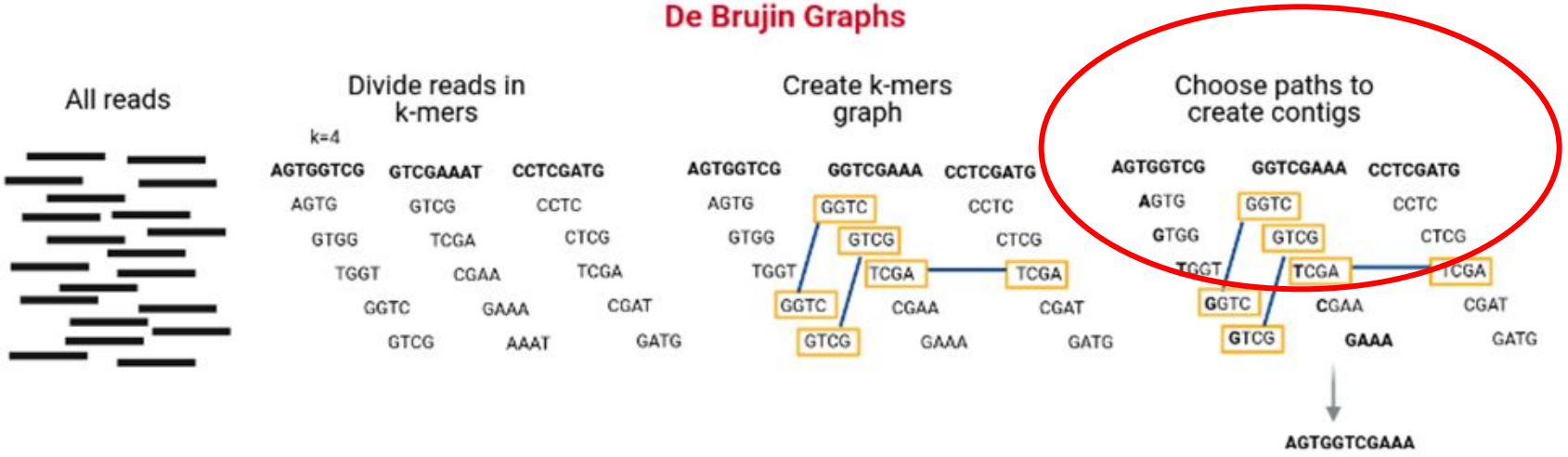
How to de novo assemble a genome?

The naive approach

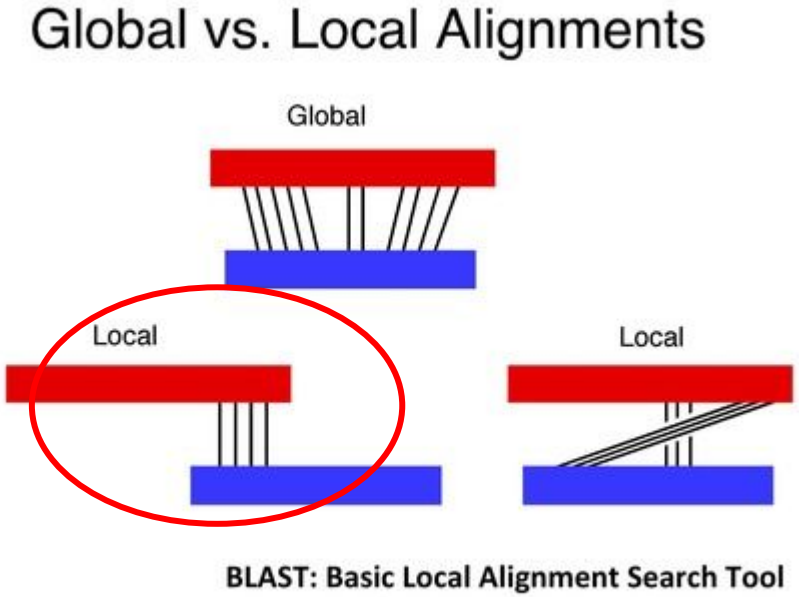


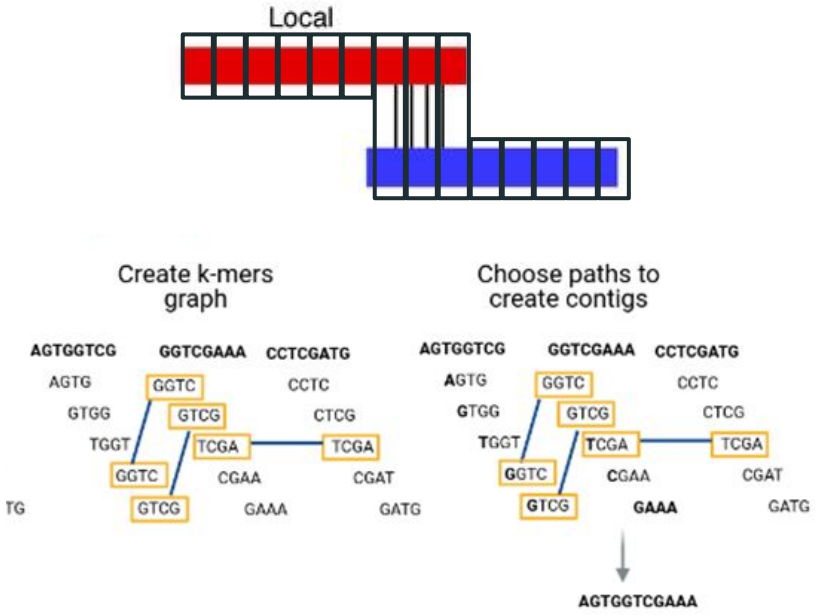
How to de novo assemble a genome?

The actual golden standard: de Bruijn Graphs

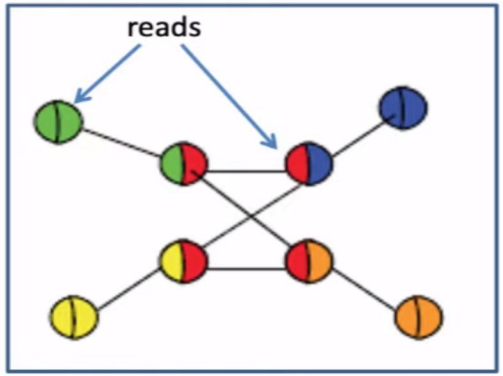


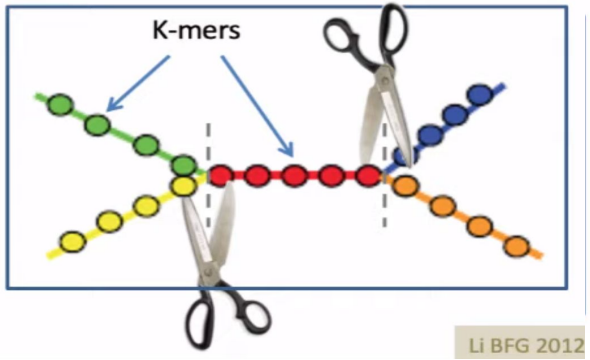
Why Kmers rather than read alignment?





How to know which paths are connected?





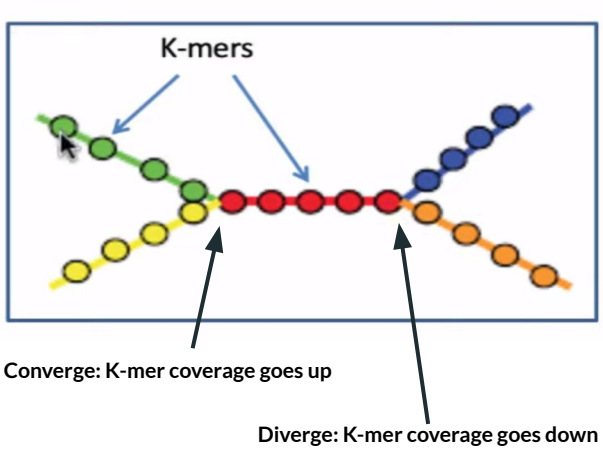
5 contigs
instead of 2 …

Then isn't greedy
extension better?

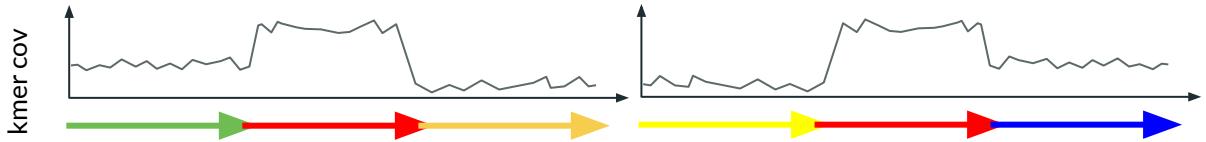
Chimeras are a big problem in metagenomix

This venomous creature eats grass and runs 80 km/h

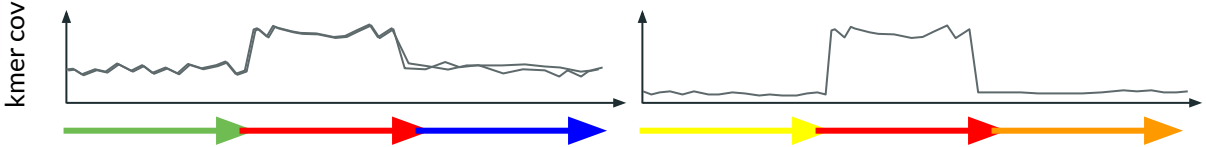
But chimera's are avoidable!



"Wrong" paths: three Kmer coverage patterns

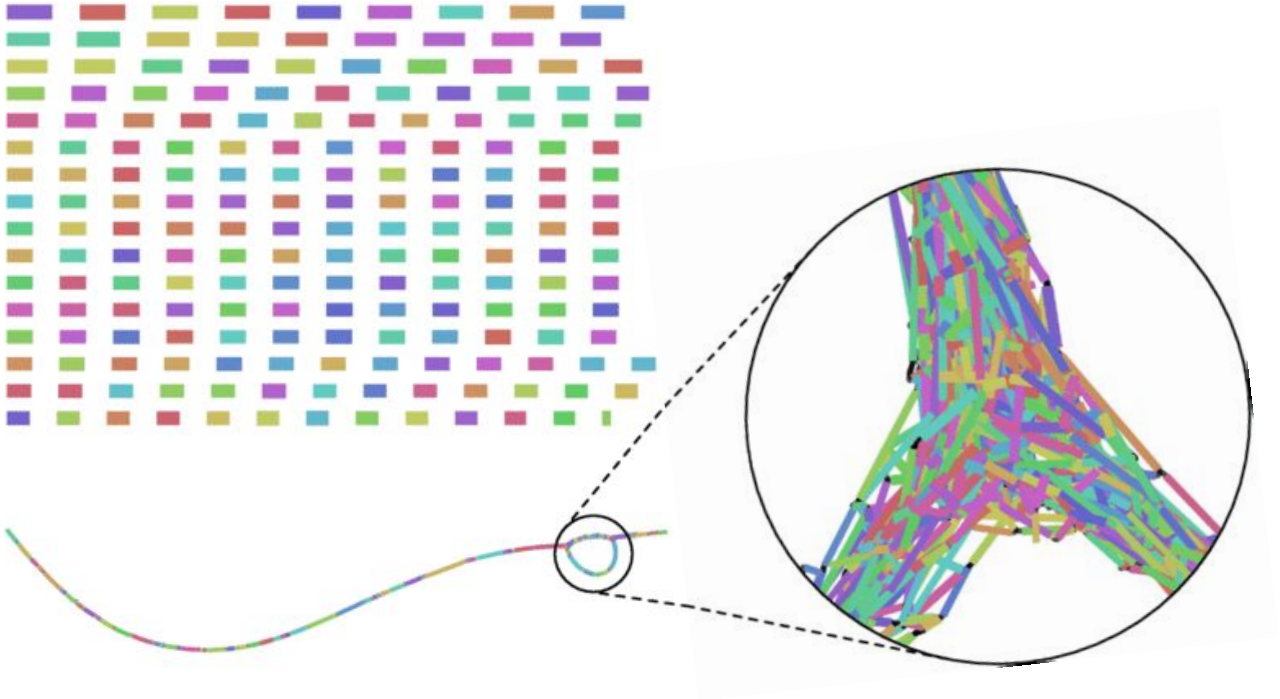


"Correct" paths: two Kmer coverage patterns

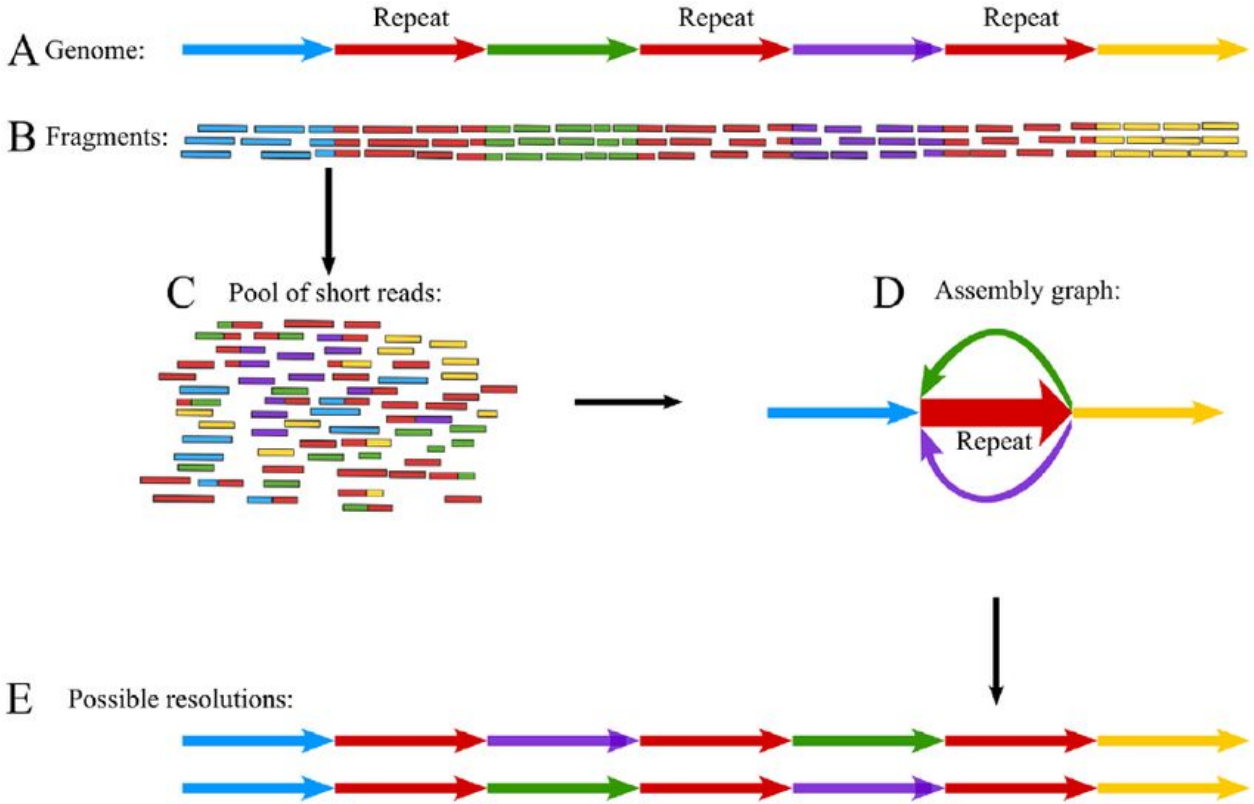


Question: under what circumstance do you still get chimeras?

Good assemblers (megahit, metaspades) take this into account



No matter how hard we try, contigs are still not genomes



Game over for assembly, we will get into "binning" later

Step 3: assembling contigs with megahit

```
#!/bin/bash


# In this script, instead of looping over files, I show how you can write your own function
in BASH. In programming, a function is just a set of instructions that can be repeated with
different inputs. Of course, the other strategy we used in earlier examples (loop over
directories, do something for each) would have worked too, and would essentially have the
same outcome. This is just to illustrate there are different ways to do things :)

read_dir="reads"

assemble() {
        sample=$1 # first argument passed to function is accessed by $1, this is the sample
name
        echo "Now assembling sample $sample..."
        megahit -1 ${read_dir}/${sample}_trimmed_R1.fastq -2
${read_dir}/${sample}_trimmed_R2.fastq -t 8 -o Assembly_${sample}
}

assemble Tminus1_C1_powersoil
assemble T1_C1_7amp_H
assemble T2_C1_7amp_H
assemble T3_C1_7amp_H
```

Step 3: assembling contigs with megahit

How good is your assembly? (N50, etc.)

- Longer contigs are always good, but simply using “average length” is a only a crude estimate: we don’t know the expected length, plus there are plasmids, phages, *etc.*

- Instead, we can use a metric called N50/L50:

N50 = length = 30kb
L50 = number = 5 contigs
(sorry can’t helpt it XD)

- In principle this **works for multiple genomes too**: how many contigs do you need to span 50% of the whole assembly?

Bbstats.sh


```
$ bbstats.sh 03_Assembly_output/Assembly_Tminus1_C1_powersoil_trimmed/final.contigs.fa -format=7#
```

```
A       C       G       T       GC      GC_stdev
0.2165  0.2867  0.2839  0.2129  0.5706  0.1037

Main genome contig total:               70248
Main genome contig sequence total:      51.507 MB
Main genome contig N/L50:               12052/743
Main genome contig N/L90:               54622/353
Max contig length:                      480260
Number of contigs > 1 KB:               6108
% main genome in contigs > 1 KB:        40.22%


Minimum         Number          Total
Contig          of              Contig
Length          Contigs         Length
--------        -------------   -------------
    200                 70248        51507245
    500                 28086        35351508
   1000                  6124        20730498
   2500                  1413        14246144
   5000                   747        11913512
  10000                   314         8884571
  25000                    55         5034830
  50000                    28         4109315
 100000                    13         3095026
 250000                     6         1836172
```

AARGH!

- **NOTE:** These are rough estimates of “quality”, so only use them to compare your own samples with each other (e.g. to test which DNA extraction kit worked better)

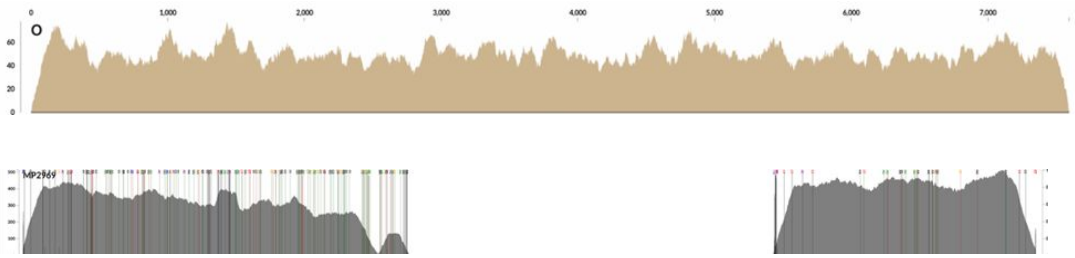
Taking a peek at the assembly file

```
$ head 03_Assembly_output/Assembly_T1_C1_7amp_H_trimmed/final.contigs.fa
>k141_35866 flag=1 multi=3.0000 len=369
GCCATCGAGAGCATCGCGTTTCAAAGCGCCGATCTCCTCGCCGCCATGCAGGCCGACTCG…
>k141_15372 flag=1 multi=2.0000 len=354
CCCTCGTAATTGATGATTTCAGGCGGGTGATGTTCTATCCGAAAGCCATTGCACTGGGCAT…
>k141_0 flag=1 multi=2.0000 len=317
CGCGGCGCCGACCCGACGACATCGAGATATTCGGGAAAGGTTTCCCAGGTCCAGGGAAG…
>k141_5124 flag=1 multi=2.0000 len=456
TACTGTGCTCGGCACCAAGACATCATGTCCCCAGTTGCCGCGTCTCTTGCGGATGCCAGCT…
>k141_30743 flag=1 multi=2.0000 len=395
GGGCTCGCCTTCGTCGCCGAGCAGCTGCTGCCCTACCTGACCGGGCTGGGCGCCGAACCA…
>k141_35963 flag=1 multi=6.0000 len=10346
AGGATGGGGTGATGCAATTGCTTCACGGCGGCGAGCACGTCGGCCTCGCGCGCCGCACGCACGAAGGCGGCGCTCAACGGAAAG
CGGCGGCCGGCGAGGATGGGGTGATGCAATTGCTTCACGGCGGCGAGCACGTCGGCCTCGCGCGCCGCACGCACGAAGGCGGCG
CTCAACGGAAAGCGGCGGCCGGCGAGGATGGGGTGATGCAATTGCTTCACGGCGGCGAGCACGTCGGCCTCGCGCGCCGCACGC
ACGAAGGCGGCGCTCAACGGAAAGCGGCGGCCGGCG …
```

Read mapping: the swiss army knife of MGX

- "Mapping" is similar to "aligning", but is more concerned with **if/where** a small sequence aligns to a larger sequence (whereas aligning is usually referring to how two similar sequences compare)
- That said, mapping is technically the same as aligning (it's just nice to know how people use it)
- Using read mapping, we can determine "coverage", but these come in two flavours:
- **Depth**, average number of reads mapping across the contig (sometimes referred to as vertical coverage)
- **Breadth**, the percentage (or fraction) of base pairs covered by at least X reads.
- These statistics can be used as indicators of "abundance", but we have to be careful!

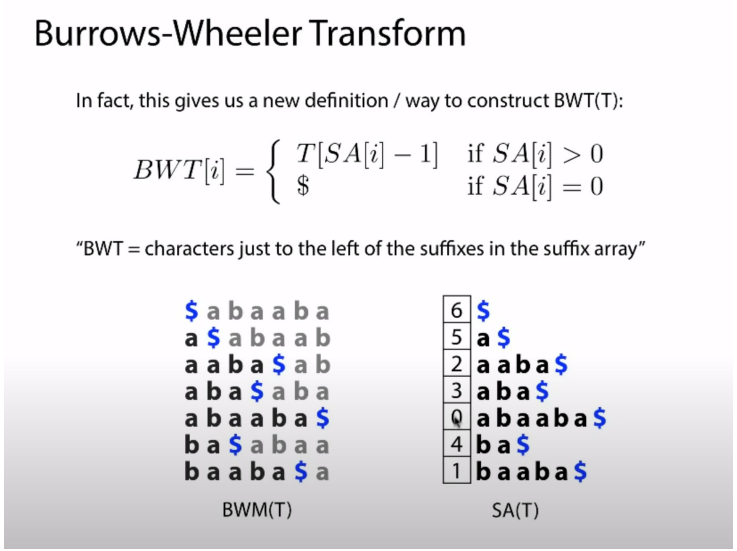
Coverage can indicate abundance



```
depth   ~ 40
breadth ~  1

depth   ~ 250    (!)
breadth ~  0.5   (!)
```

- NOTE: I strongly advice making claims of absolute abundance (this depends too much on sampling quality, PCR cycles, biases…)
- NOTE: It would be unfair to conclude that sequence 2 is “more abundant” than sequence 1
- However: situation 2 will not happen when you **map back** on the sample itself (think about it: how could it?). With cross-sample comparisons, best practice is to combine breadth/depth. And if you base your entire analysis on a few contigs, at least check the coverage distributions

Back-mapping



- BWA is a mapping tool that uses burrows-wheeler transforms
- All you need to know now, it that we must first "make" the transformation matrix, and then read mapping is really fast! (it can map millions of reads in mere minutes!)
- The output it what is a "binary alignment map" (BAM)

```
$ bwa index $contigs
$ bwa mem -t 8 $contigs $read_dir/${sample}_trimmed_R1.fastq
$read_dir/${sample}_trimmed_R2.fastq > $output/read_mapping.bam
```

- The human-readable version of a BAM file is a SAM-file, which stands for "sequence alignment map", which is produced by opening a bam-file with samtools:

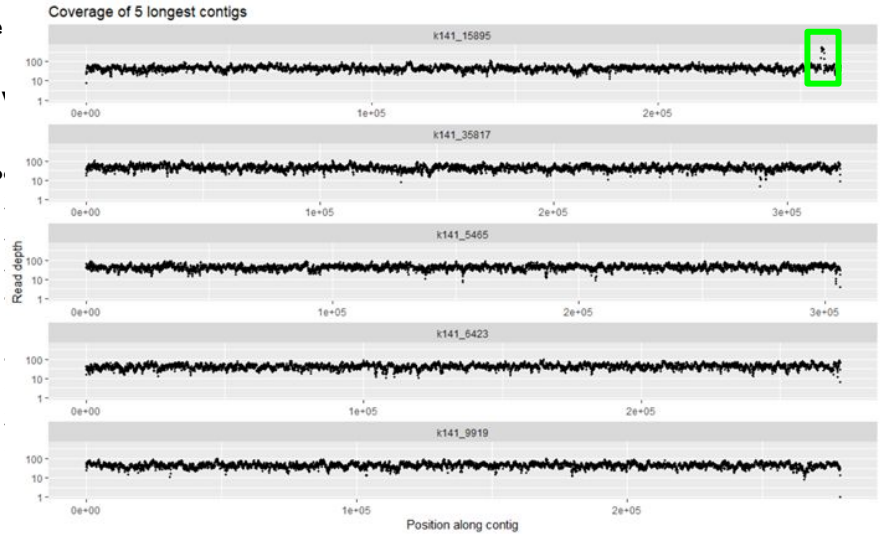
```
$ samtools view read_mapping.bam | head -n 3
VH00578:2:AAAJVKHHV:1:1101:65040:12018  99      k141_29350      2320    60      151M    =
2497    328
CTACCGAACACCGCGGCCGTCACTGGCCACGGGGAGAACTTTACGGGGCGCGGTGTGCTGCGGTTCGCGGAAACACGCGGGTTTGAACGCCCTGTTTC
TGGTTCTGGCGCATAAACCGCCGACCGTCACGACTCGCTTTCCTGATCCATCT
CCCCCCCCCCCCCCCCCCCC;CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCC;CCCCCCCCCCCC;CCCCCCCCCCCCCCCCCCCCCC  NM:i:1  MD:Z:149G1      AS:i:149
XS:i:0
VH00578:2:AAAJVKHHV:1:1101:65040:12018  147     k141_29350      2497    60      151M    =
2320    -328
TCCACGGCCGTTTCCGGCGTTTCGTCCGGGCCGATGACGCCATCGCCCCACGCCTCAACGCACTGGGCGTGCGTGAGCCCGGTCCGCTGCTGCAGCAA
CGCAATGAACCGTGCCTTGAAGGCTTCTTGTGCCTGTCTATCCATTCCTGGCC
-CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC;CCCCCCCCCCCCCCCC  NM:i:0  MD:Z:151        AS:i:151
XS:i:0
VH00578:2:AAAJVKHHV:1:1101:67710:12037  77      *       0       0       *       *       0       0
GGCCGGCAGGAGAGTAACTTCCACCCATACCCTCCGTGATCCGCAATTGTGGTCTGTAGAAACGCCGCAATTATACCGGGTAAAAACTACCGTAAAAG
ATAAAAAGACGGGCCGGGTTTGGGAACAGACCACCCACACTTTCGGACTCCGG
CCCCCCCCCC;C;CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC;C;CCCCCCCC;CCCCCCCCC-CC
```

Back-mapping: getting depths

```
$samtools sort read_mapping.bam > read_mapping.sorted.bam

$samtools coverage
```

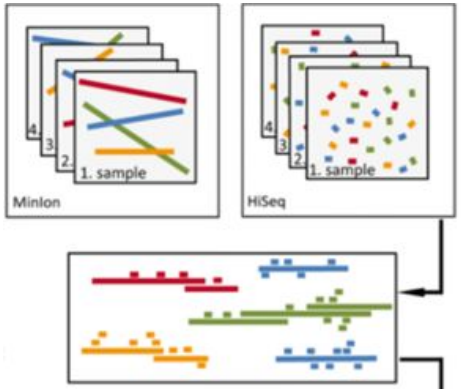
- Which will gi

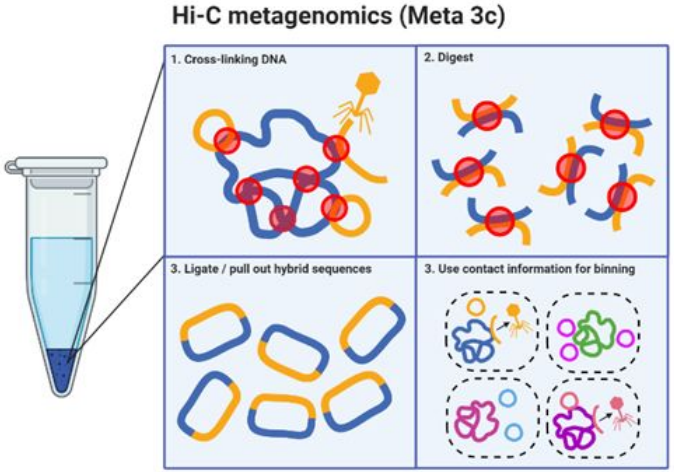
```
#rname startpos endp
k141_35866    1
k141_15372    1
k141_0        1
k141_5124     1
k141_30743    1
k141_25620    1
k141_10248    1
k141_15373    1
k141_20496    1
```



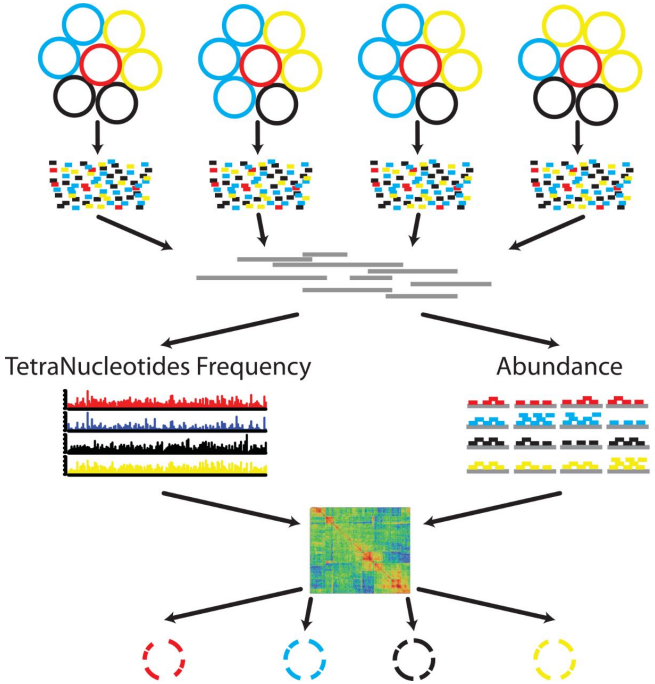
Binning: the biggest challenge in MGX

- Because of fragmentation of contigs (because of repeats), there will be MANY more contigs than there are genomes/replicons in your sample
- How can we determine which ones belong to “the same genome”?





Binning with metabat



Preprocessing

1. Samples from multiple sites or times
2. Metagenome libraries
3. Initial de-novo assembly using the combined library

MetaBAT

4. Calculate TNF for each contig
5. Calculate Abundance per library for each contig
6. Calculate the pairwise distance matrix using pre-trained probabilistic models
7. Forming genome bins iteratively

Now we have “the end product” of MGX

```
$ls 05_Binning
MAG.1.fa
MAG.2.fa
MAG.3.fa
MAG.4.fa
MAG.lowDepth.fa
MAG.tooShort.fa
MAG.unbinned.fa
metabat_abundance_file.txt
```

Very different from 16s:

Before we annotated ANYTHING we did a lot of work to get bigger and bigger collections of DNA, which eventually gave us what is known as….

Metagenome-assembled genome (MAG)

Everything we will do from here is essentially “genomics”

MAGs: what are they?

Not genomes.

How “good” are the MAGs?

BBSTATS

```
$ bbstats.sh MAG.2.fa -format=7
A         C         G         T           GC       GC_stdev
0.3269    0.1734    0.1728    0.3269    0.3462    0.0097

Main genome contig total:                   78
Main genome contig sequence total:          3.096 MB
Main genome contig N/L50:                   16/74535
Main genome contig N/L90:                   48/19106
Max contig length:                          167244
Number of contigs > 1 KB:                   78
% main genome in contigs > 1 KB:            100.00%

Minimum          Number             Total

Contig           of                 Contig

   2772                    78       3095686

  10000                    62       2998612

  25000                    40       2624293

  50000                    22       1960566

 100000                     6        773507
```

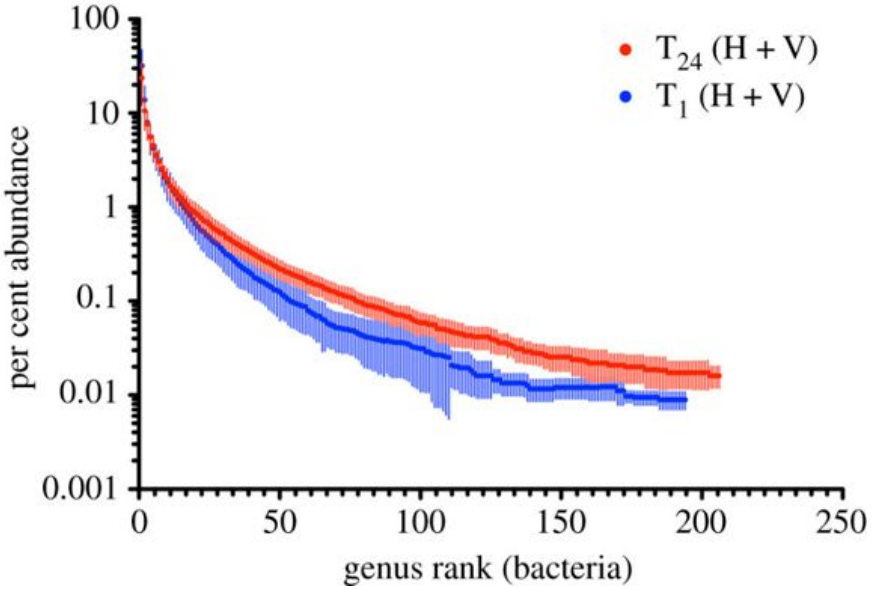
CHECKM

- Estimates “completion” and “contamination” of your MAG based on single-copy marker genes
- As always: a good indication but don’t bet your life on it.

Bin Id	Marker lineage	# genomes	# markers	# marker sets	0	1	2	3	4	5+	Completeness	Contamination	Strain heterogeneity
MAG.3	o__Cytophagales (UID2936)	47	454	336	1	445	7	0	0	1	99.70	2.53	5.88
MAG.2	p__Bacteroidetes (UID2591)	364	303	203	2	299	2	0	0	0	99.01	0.74	0.00
MAG.tooShort	k__Bacteria (UID203)	5449	104	58	7	8	6	12	20	51	88.79	324.33	2.54
MAG.1	f__Xanthomonadaceae (UID4214)	55	659	290	88	562	9	0	0	0	84.68	0.83	0.00
MAG.unbinned	k__Bacteria (UID203)	5449	104	58	14	31	25	17	9	8	78.45	71.41	3.46
MAG.4	k__Bacteria (UID203)	5449	104	58	98	6	0	0	0	0	10.34	0.00	0.00

Why are my MAGs so bad?

Many microbial communities have a long-tailed abundance curve:

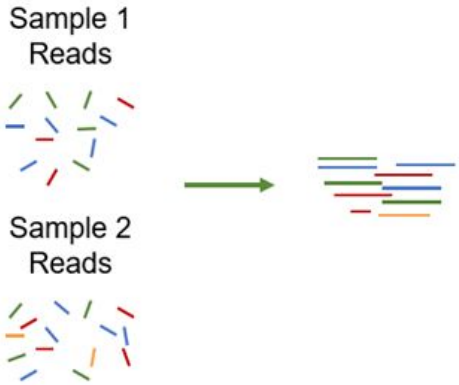


Strategies for improving MAGs

- Sampling MAGs from **many environments** and different conditions (but, dereplication is necessary!)
- **Cross-assembly** (sometimes co-assembly), which is simply pooling reads from many samples into one big file, and assembling the result.

  Your computer will need a LOT of RAM if you do this (sometime terabytes …)

  Also: increases risks of chimera's, so take care !!
- Combine illumina with **long-read sequencing technology** (nanopore, etc.) to get high quality genomes



As said: a lot happens before annotation! :)

- Today: trimming, assembly, and binning
- Tomorrow: annotation and detecting horizontal gene transfer

Hands-on part of the workshop

What to do?

- **Option 1:** follow instructions in the workshop manual to get some finger exercises on how to do each step. Because MGX is not super fast, running the whole suite will take a while…

- **Option 2:** study the scripts from option 1 without running, and go inspect some of the output that was already run for you at: /groups/mpistaff/MGX_workshop/MGX_workshop_vandijk

- **Option 3:** talk to me or Pauline about what you could do with *your* data. Are you stuck? Do you need help finding a tool? Let us know :)

How do do it?

- On Windows you need to either (i) download a terminal emulator (MobaXterm is my hot tip), or (ii) use our webvpn service: https://webvpn.evolbio.mpg.de/

  - Log into Wallace using ssh <username>@wallace.evolbiompg.de

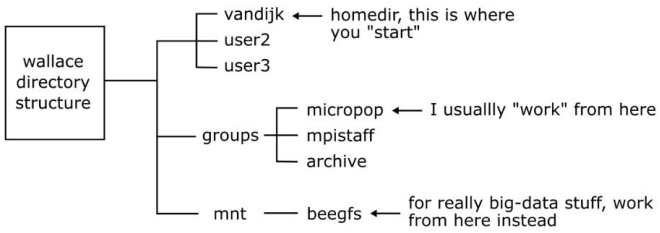
  - Login to either node01-04 using:
    `$ ssh node01`

- For webvpn users: your sessions may time out after some time of inactivity. If this happens, try opening up a 'screen' after you login to wallace like this:
  `$ screen`

  After you timed out (disconnected), you can find your session like this
  `$ screen -list`

  And reconnect like this:
  `$ screen -r <SESSION_ID>`

Logging in to Wallace: ssh vandijk@wallace



So, to do some stuff in your own directory in micropop:

```
$ cd /groups/micropop
$ mkdir my_directory
$ cd my_directory
$ bash my_script.sh
```

Scripting with Bash (101)

- create/open new script, e.g. nano my_script.sh
- first line should always be: #!/bin/bash
- example script:

```
#!/bin/bash
echo "Start script"
for file in *.fastq.gz; do
   gunzip $file
   echo "Done unzipping $file"
done;
echo "All files are unzipped. Hoping for the best!"
```

To run the script:

```
$ bash my_script.sh
```

Pro tip: if you have a slow script (e.g. one that copies a lot of files), first create a "dummy" script that simply prints the commands without executing them by adding "echo" in front of the steps.

Characters with special meanings:

```
.          current directory
../        parent directory
../../     parent directory of parent directory
/          root directory
>          pipe output to a file
           (e.g. ls *.fastq > all_reads.txt)
>>         pipe output to file (append)
|          pipe output to another program
           (e.g. ls *.fastq | grep "1A")
<          reverse pipe
           (rarely used, but you may encounter it)
&          when typed after a command, it will
           put the process "in the background"
           meaning you will get your prompt back
           and can type more commands while the
           other program is running.
$          access a variable
           (e.g. echo $USER)
$()        store output in variable
           (e.g. variable=$(command_here))
```

Useful commands:

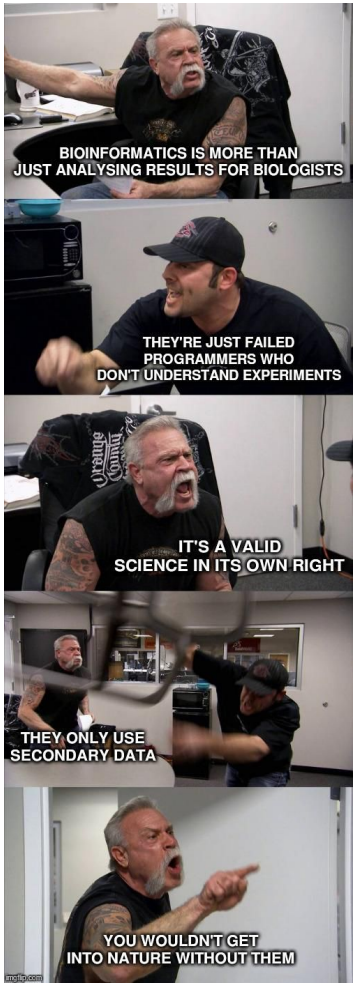
command	description	example
cd	change directory	cd /groups/micropop
ls	list content of dir	ls my/data
cp	copy a file	cp myfile.txt /groups/micropop
mv	move (or rename) file	mv myfile.txt myfile_renamed.txt
cat	print file content	cat myfile.txt
wc	count words, lines or chars	wc -l myfile.txt
grep	find lines with pattern	grep "read_001" myfile.txt
cut	split by delimiter	cat myfile.txt | cut -f2
nano	command-line text editor	nano my_script.sh
gedit	graphical text editor	gedit my_script.sh
sort	sort input	cat myfile.txt | sort -n
uniq	remove duplicate items	cat myfile.txt | sort -n | uniq
mkdir	make new directory	mkdir /groups/micropop/b_obama
echo	print something	echo "Hello!"
gzip	zip file	gzip reads.fastq
gunzip	unzip file	gunzip reads.fastq

Example of a chain of commands:

```
ls *.fastq | grep "1A" | cut -d '_' -f2 > all_samples_with_1A.txt
```

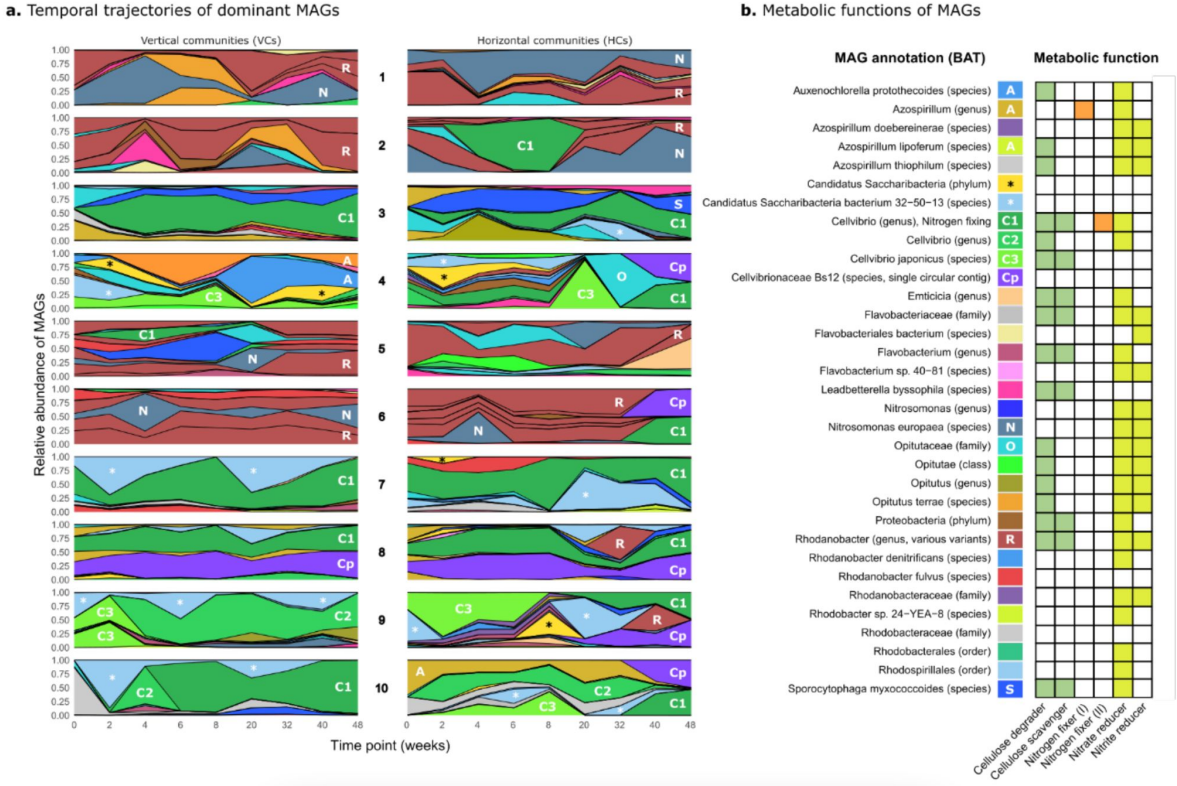
Let's start with questions

PART II
Annotation and horizontal gene transfer

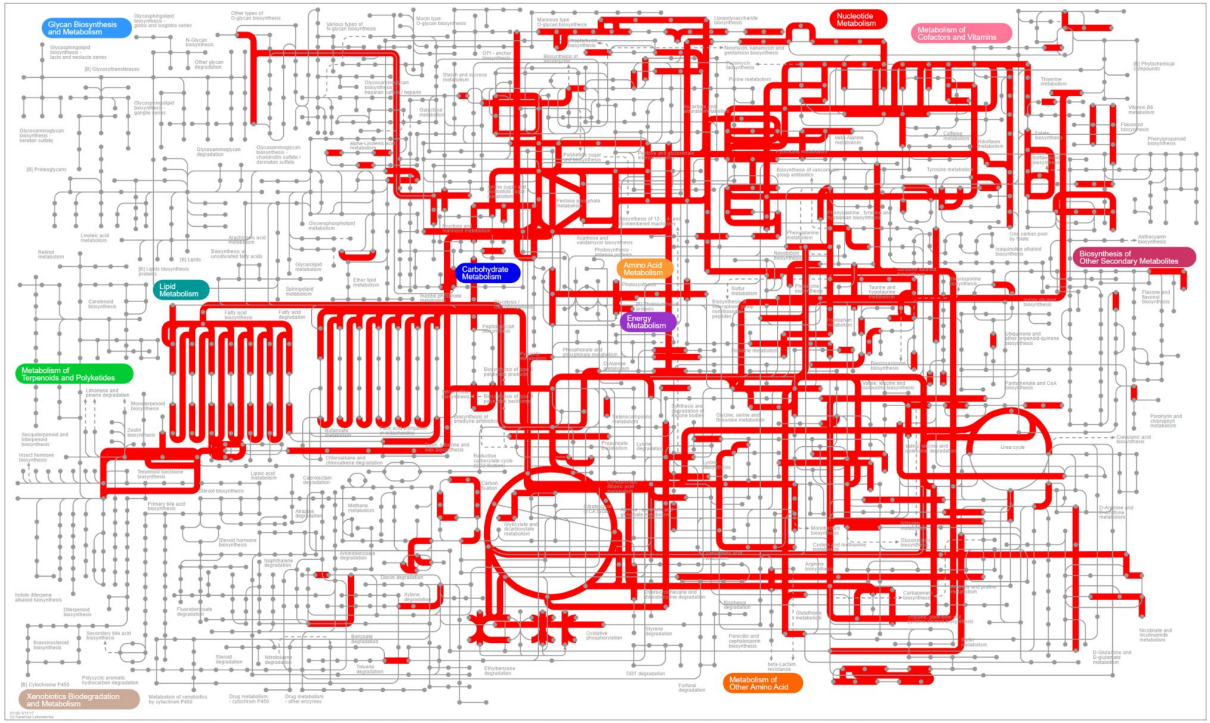


First: Let's copy some files

MAGs from Steven Quistad's compost

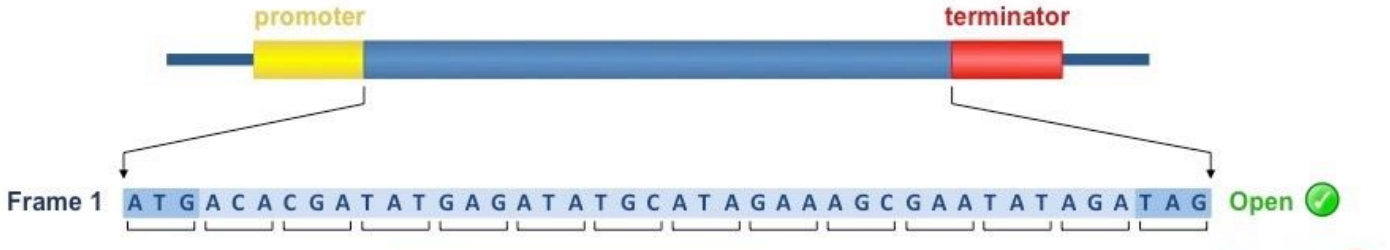


MGX: annotation of gene content and function

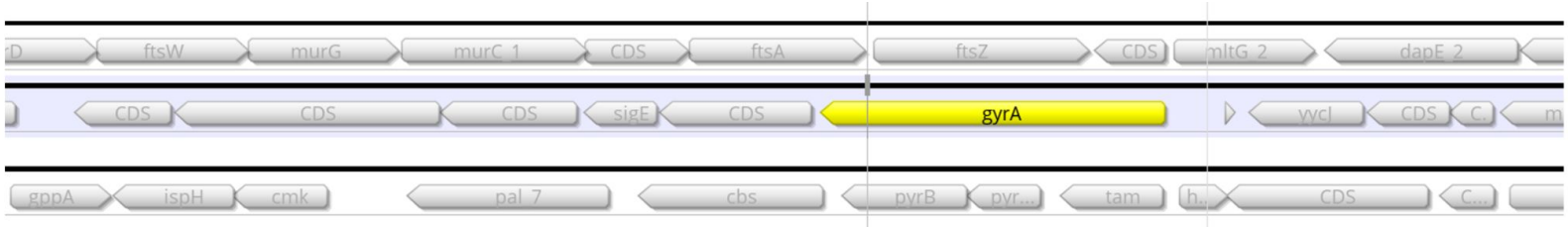


How do you go from sequence to a “function”?

Prodigal: predicting open reading frames (ORFs)



Prokka: prodigal + annotation



Prokka can be run on any fasta file

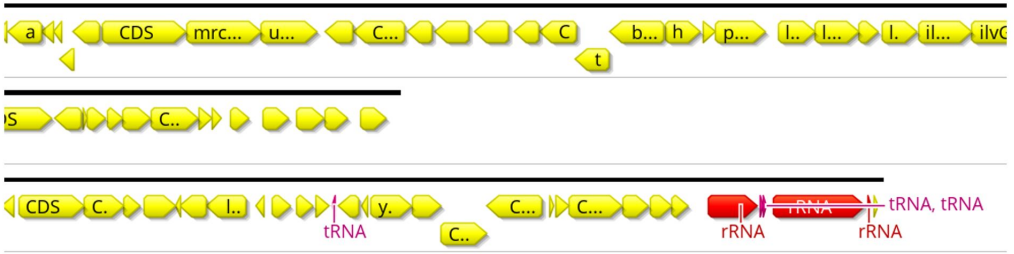
```
$ prokka 05_Binning/MAG.3.fa --prefix 06_Prokka_MAG3
```

```
06_Prokka_MAG3.err
06_Prokka_MAG3.faa
06_Prokka_MAG3.ffn
06_Prokka_MAG3.fna
06_Prokka_MAG3.fsa
06_Prokka_MAG3.gbk
06_Prokka_MAG3.gff
06_Prokka_MAG3.log
06_Prokka_MAG3.sqn
06_Prokka_MAG3.tbl
06_Prokka_MAG3.tsv
06_Prokka_MAG3.txt
```

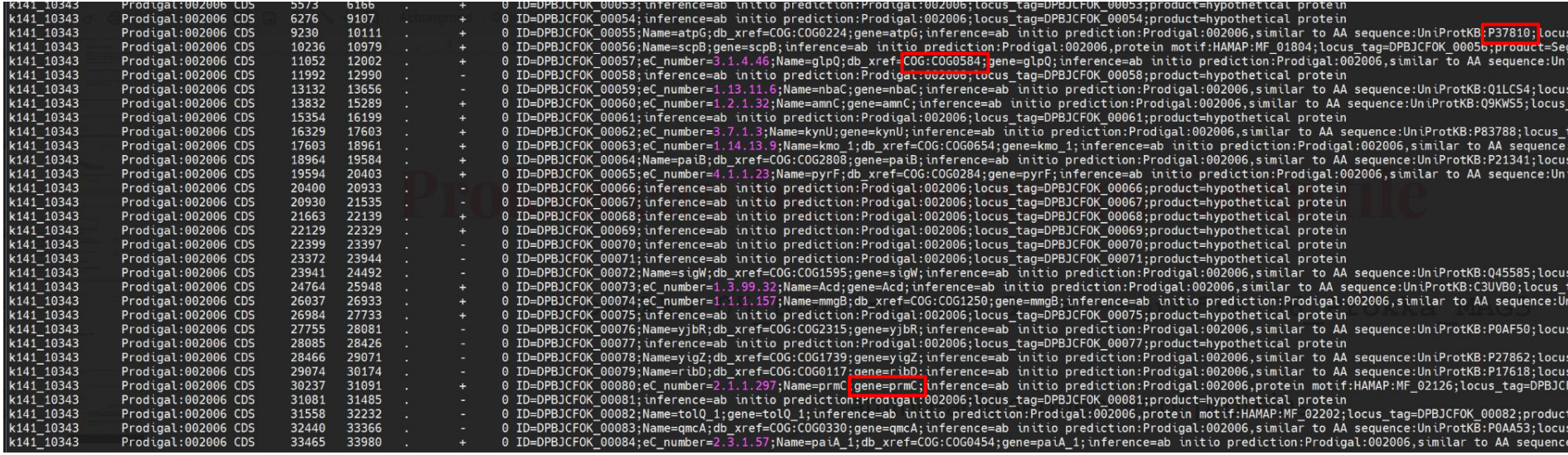
amino-acid sequences of proteins

gbk = genbank file

gff = general-feature format:



The GFF file has a LOT of information in it



```
k141_10343  Prodigal:002006 CDS  5573   6166   .  +  0  ID=DPBJCFOK_00053;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00053;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  6276   9107   .  +  0  ID=DPBJCFOK_00054;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00054;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  9230   10111  .  +  0  ID=DPBJCFOK_00055;Name=atpG;db_xref=COG:COG0224;gene=atpG;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:P37810;locus
k141_10343  Prodigal:002006 CDS  10236  10979  .  +  0  ID=DPBJCFOK_00056;Name=scpB;gene=scpB;inference=ab initio prediction:Prodigal:002006,protein motif:HAMAP:MF_01804;locus_tag=DPBJCFOK_00056;product=Se
k141_10343  Prodigal:002006 CDS  11052  12002  .  +  0  ID=DPBJCFOK_00057;eC_number=3.1.4.46;Name=glpQ;db_xref=COG:COG0584;gene=glpQ;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:Un
k141_10343  Prodigal:002006 CDS  11992  12990  .  -  0  ID=DPBJCFOK_00058;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00058;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  13132  13656  .  -  0  ID=DPBJCFOK_00059;eC_number=1.13.11.6;Name=nbaC;gene=nbaC;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:Q1LCS4;locus
k141_10343  Prodigal:002006 CDS  13832  15289  .  +  0  ID=DPBJCFOK_00060;eC_number=1.2.1.32;Name=amnC;gene=amnC;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:Q9KW55;locus
k141_10343  Prodigal:002006 CDS  15354  16199  .  +  0  ID=DPBJCFOK_00061;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00061;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  16329  17603  .  +  0  ID=DPBJCFOK_00062;eC_number=3.7.1.3;Name=kynU;gene=kynU;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:P83788;locus_
k141_10343  Prodigal:002006 CDS  17603  18961  .  +  0  ID=DPBJCFOK_00063;eC_number=1.14.13.9;Name=kmo_1;db_xref=COG:COG0654;gene=kmo_1;inference=ab initio prediction:Prodigal:002006,similar to AA sequence
k141_10343  Prodigal:002006 CDS  18964  19584  .  +  0  ID=DPBJCFOK_00064;Name=paiB;db_xref=COG:COG2808;gene=paiB;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:P21341;locus
k141_10343  Prodigal:002006 CDS  19594  20403  .  +  0  ID=DPBJCFOK_00065;eC_number=4.1.1.23;Name=pyrF;db_xref=COG:COG0284;gene=pyrF;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:Un
k141_10343  Prodigal:002006 CDS  20400  20933  .  -  0  ID=DPBJCFOK_00066;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00066;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  20930  21535  .  -  0  ID=DPBJCFOK_00067;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00067;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  21663  22139  .  +  0  ID=DPBJCFOK_00068;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00068;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  22129  22329  .  +  0  ID=DPBJCFOK_00069;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00069;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  22399  23397  .  -  0  ID=DPBJCFOK_00070;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00070;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  23372  23944  .  -  0  ID=DPBJCFOK_00071;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00071;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  23941  24492  .  -  0  ID=DPBJCFOK_00072;Name=sigW;db_xref=COG:COG1595;gene=sigW;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:Q45585;locus
k141_10343  Prodigal:002006 CDS  24764  25948  .  +  0  ID=DPBJCFOK_00073;eC_number=1.3.99.32;Name=Acd;gene=Acd;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:C3UVB0;locus_
k141_10343  Prodigal:002006 CDS  26037  26933  .  +  0  ID=DPBJCFOK_00074;eC_number=1.1.1.157;Name=mmgB;db_xref=COG:COG1250;gene=mmgB;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:Un
k141_10343  Prodigal:002006 CDS  26984  27733  .  +  0  ID=DPBJCFOK_00075;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00075;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  27755  28081  .  -  0  ID=DPBJCFOK_00076;Name=yjbR;db_xref=COG:COG2315;gene=yjbR;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:P0AF50;locus
k141_10343  Prodigal:002006 CDS  28085  28426  .  -  0  ID=DPBJCFOK_00077;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00077;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  28466  29071  .  -  0  ID=DPBJCFOK_00078;Name=yigZ;db_xref=COG:COG1739;gene=yigZ;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:P27862;locus
k141_10343  Prodigal:002006 CDS  29074  30174  .  -  0  ID=DPBJCFOK_00079;Name=ribD;db_xref=COG:COG0117;gene=ribD;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:P17618;locus
k141_10343  Prodigal:002006 CDS  30237  31091  .  +  0  ID=DPBJCFOK_00080;eC_number=2.1.1.297;Name=prmC;gene=prmC;inference=ab initio prediction:Prodigal:002006,protein motif:HAMAP:MF_02126;locus_tag=DPBJC
k141_10343  Prodigal:002006 CDS  31081  31485  .  -  0  ID=DPBJCFOK_00081;inference=ab initio prediction:Prodigal:002006;locus_tag=DPBJCFOK_00081;product=hypothetical protein
k141_10343  Prodigal:002006 CDS  31558  32232  .  -  0  ID=DPBJCFOK_00082;Name=tolQ_1;gene=tolQ_1;inference=ab initio prediction:Prodigal:002006,protein motif:HAMAP:MF_02202;locus_tag=DPBJCFOK_00082;product
k141_10343  Prodigal:002006 CDS  32440  33366  .  -  0  ID=DPBJCFOK_00083;Name=qmcA;db_xref=COG:COG0330;gene=qmcA;inference=ab initio prediction:Prodigal:002006,similar to AA sequence:UniProtKB:P0AA53;locus
k141_10343  Prodigal:002006 CDS  33465  33980  .  +  0  ID=DPBJCFOK_00084;eC_number=2.3.1.57;Name=paiA_1;db_xref=COG:COG0454;gene=paiA_1;inference=ab initio prediction:Prodigal:002006,similar to AA sequenc
```

Bioinformatics is like Lego®

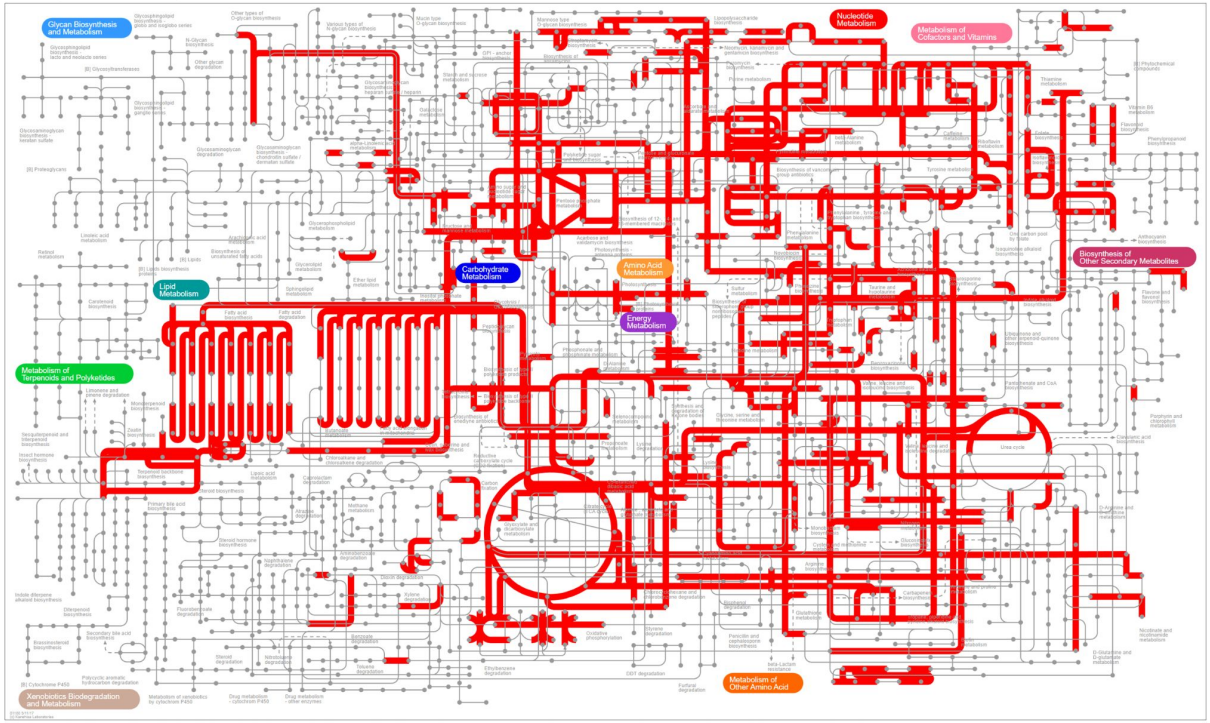
```
# First, run prokka.

prokka 05_Binning/MAG.2.fa --prefix 06_Prokka_MAG2

# Next, I extract the UniProt IDs
grep -o 'UniProt.*' 06_Prokka_MAG2/06_Prokka_MAG2.gff | cut -d';' -f1 |
cut -d':' -f2 | sed 's/^/UNIPROT:/g' > 06_Prokka_MAG2/UniProt_IDs.txt
```

vertical slash
(or “pipe symbol”)

redirect output to a file

```
UNIPROT:P54979,
UNIPROT:Q7AKG9,
UNIPROT:Q0P9D0,
UNIPROT:Q7DBF3,
UNIPROT:Q7WTB1,
UNIPROT:Q0P9C9,
UNIPROT:Q48485,
UNIPROT:P54420,
UNIPROT:P54420,
UNIPROT:G3XD23,
UNIPROT:G3XD01,
UNIPROT:P37744,
UNIPROT:P26391,
UNIPROT:Q8RDI4,
…
```

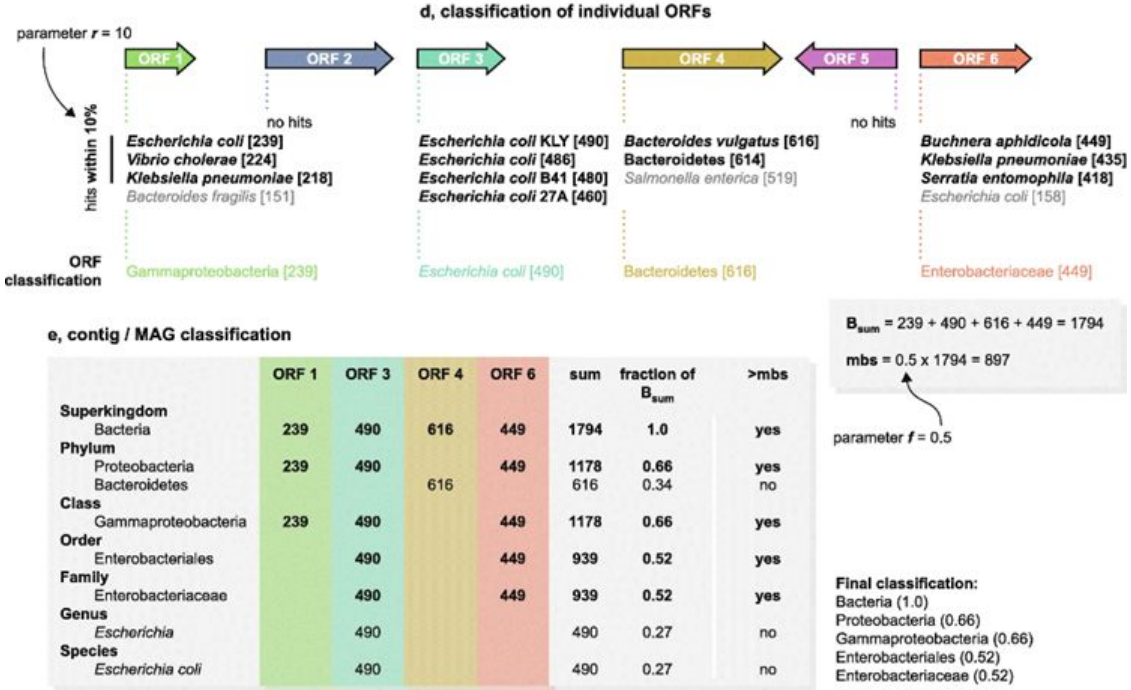
iPath3: visualise metabolism with uniprot IDs



Others (web-based) things you could try this afternoon

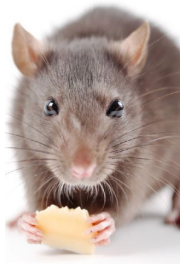
- Interproscan: protein family and domain prediction
- Antismash: predict secondary metabolite gene clusters
- BlastKOALA: predict KEGG Ontology gene categories
- Phaster: prophage prediction
- ICEberg: integrative element prediction
- And a lot more :)

Contig annotation tool (CAT, not cat)



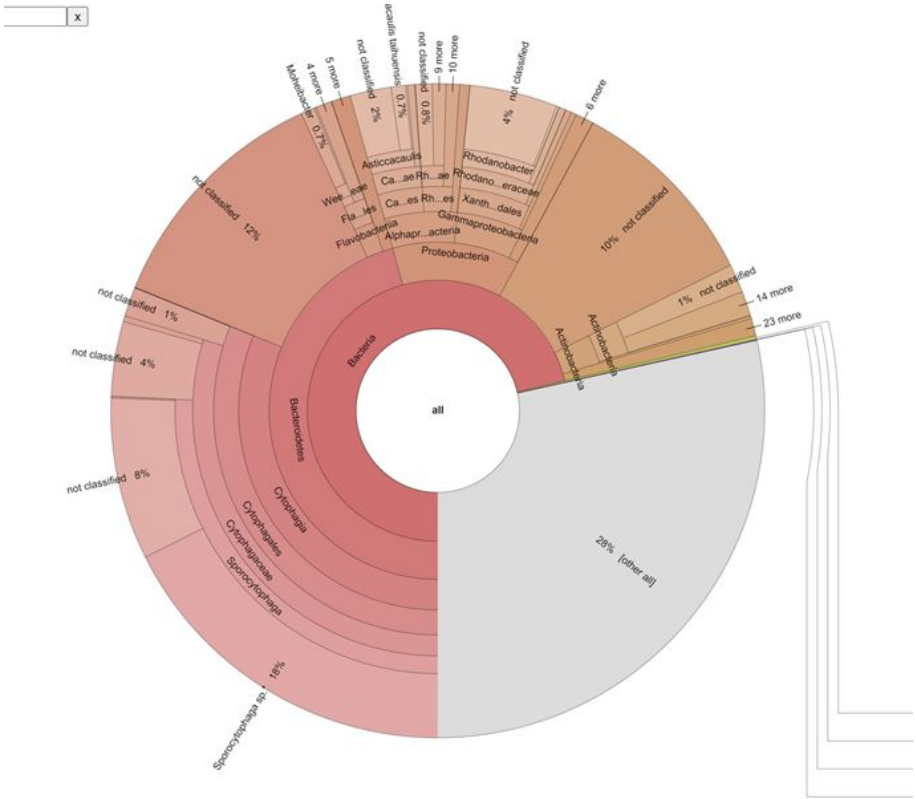
	ORF 1	ORF 3	ORF 4	ORF 6	sum	fraction of B_{sum}	>mbs
Superkingdom							
Bacteria	239	490	616	449	1794	1.0	yes
Phylum							
Proteobacteria	239	490		449	1178	0.66	yes
Bacteroidetes			616		616	0.34	no
Class							
Gammaproteobacteria	239	490		449	1178	0.66	yes
Order							
Enterobacteriales		490		449	939	0.52	yes
Family							
Enterobacteriaceae		490		449	939	0.52	yes
Genus							
Escherichia		490			490	0.27	no
Species							
Escherichia coli		490			490	0.27	no

BAT



RAT

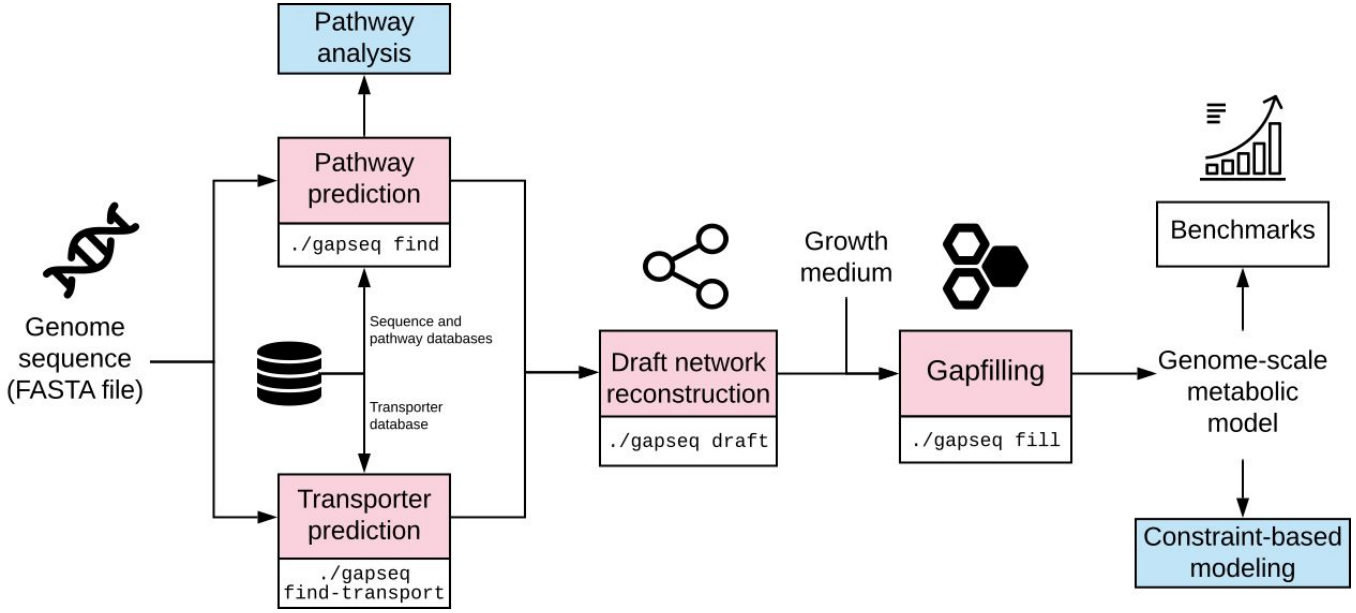
CAT + RAT + Krona



RAT is used to estimate the abundances of contigs (again, this is simply read mapping!)

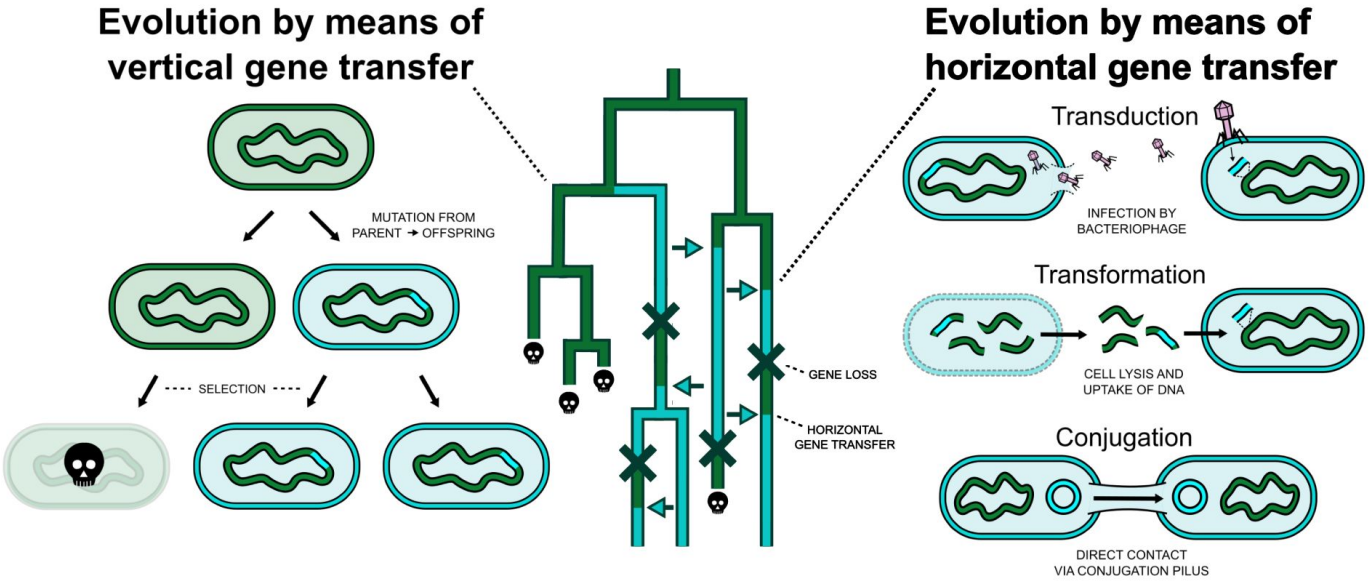
Then, the results of **CAT** can be plotted in a hierarchical diagram, like a Krona plot ←

Metabolic modelling



Predicting the reactions present in MAGs and trying to make models “generate biomass”

Horizontal Gene Transfer



"The walls that divide bacteria from one another are far from solid. Taken to extremes, the preponderance of HGT could even imply that microbiomes are better conceptualized as collections of locally adaptive genes, rather than communities of locally adapted species", – J.P. Hall, 2021

HGT and loss dominate bacterial evolution

Quantify **mutations** that entail whole genes:

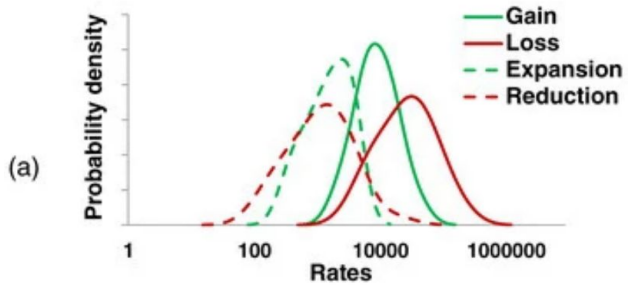
-> Duplication of a gene (**Expansion**)
-> Deletion of a duplicated gene (**Reduction**)
-> Deletion of the only copy of a gene (**Loss**)
-> **Gain** (mostly HGT)

Research article | Open Access | Published: 21 August 2014

Genomes in turmoil: quantification of genome dynamics in prokaryote supergenomes

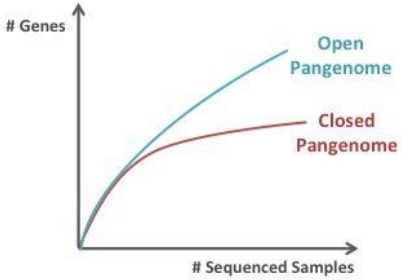
Pere Puigbò, Alexander E Lobkovsky, David M Kristensen, Yuri I Wolf & Eugene V Koonin

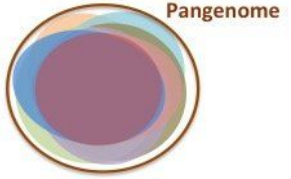
BMC Biology **12**, Article number: 66 (2014) | Cite this article



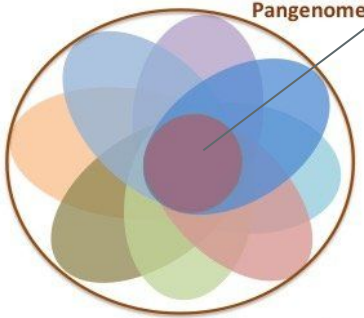
"Indeed, the estimated rates of gene family gain and loss in some groups of bacteria are such that multiple genes appear to come and go over the time required for a single nucleotide substitution to occur in an evolving gene."

The tree of one percent





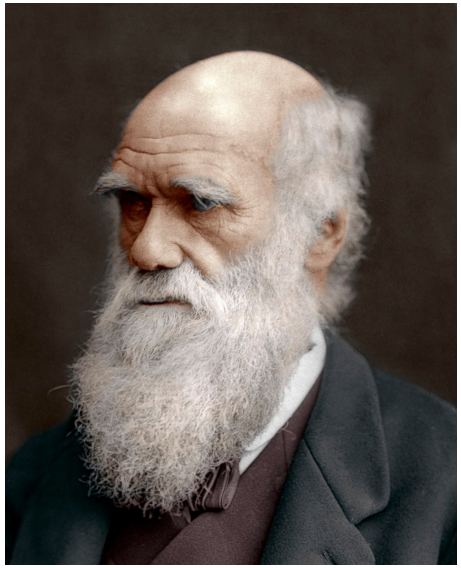
High (Core / Pangenome)



Low (Core / Pangenome)

Tal Dagan:
The prokaryotic tree of life
is a tree of 1% of their genomes…

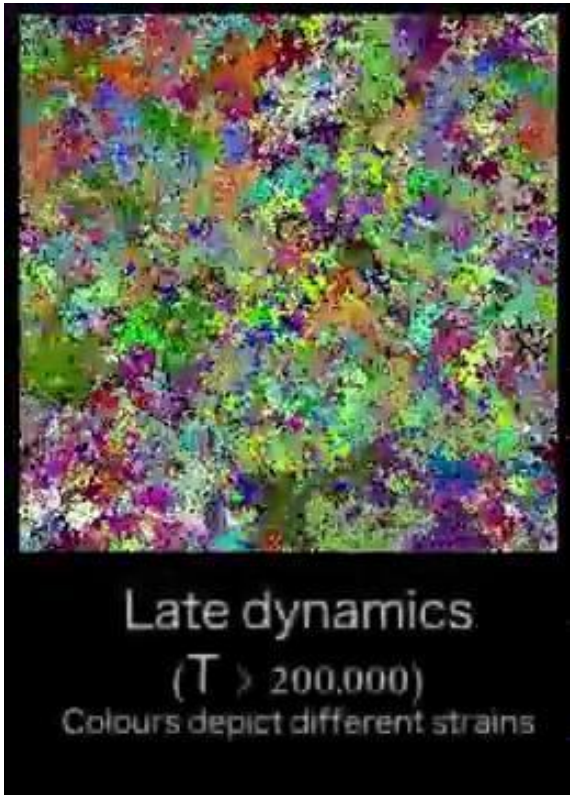
This changes how we think about "descent with modification"



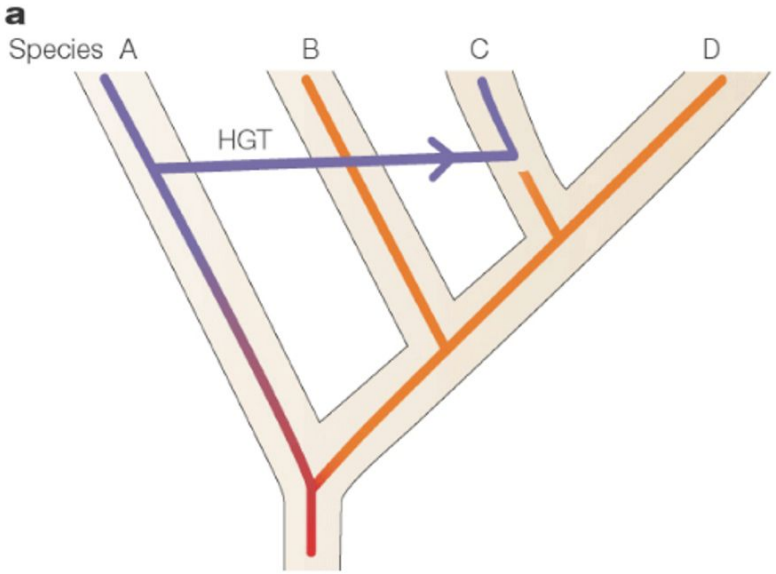
Descent with modification still applies, but to who does it apply? Genes? Bacteria? Groups?

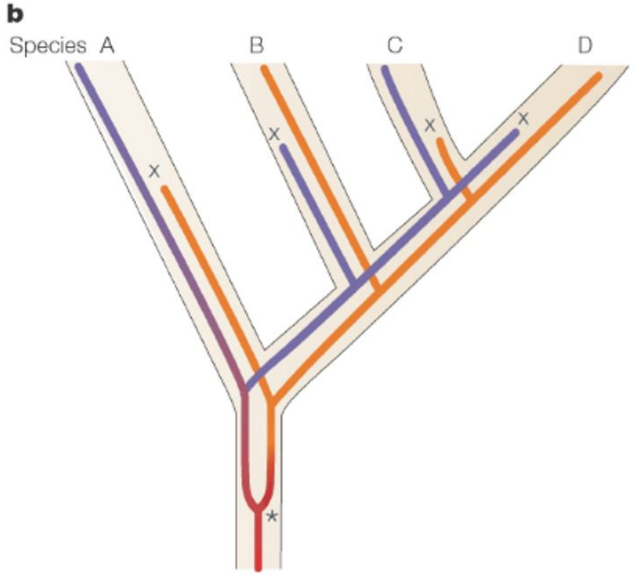
My preferred perspective: to all of them. Selection can act on all these levels in concert: theory of multi-level evolution

Multi-level evolution simulations

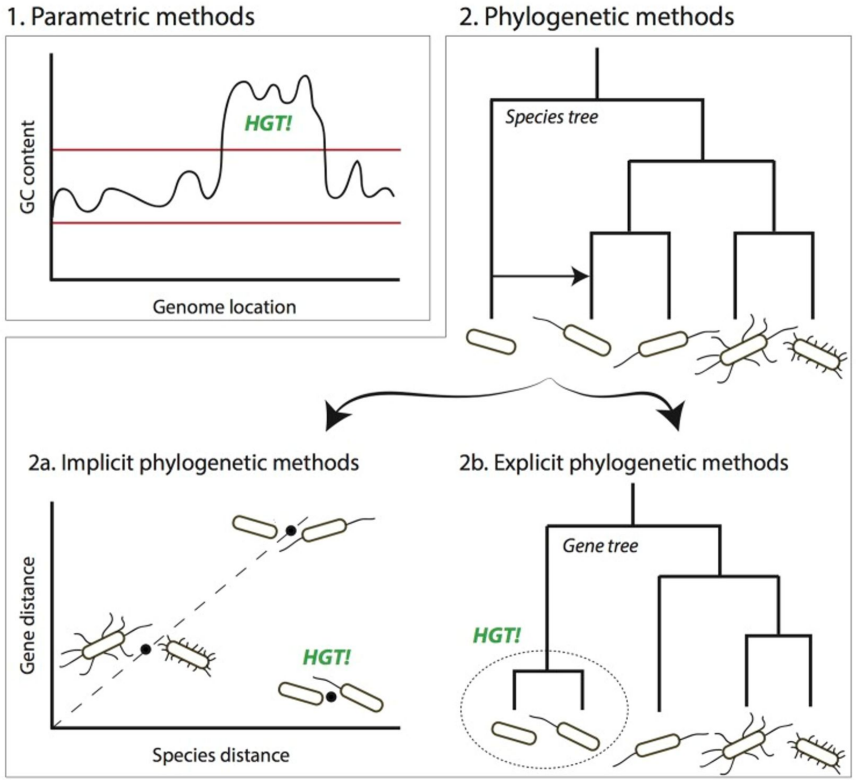


How do detect HGT with MGX?

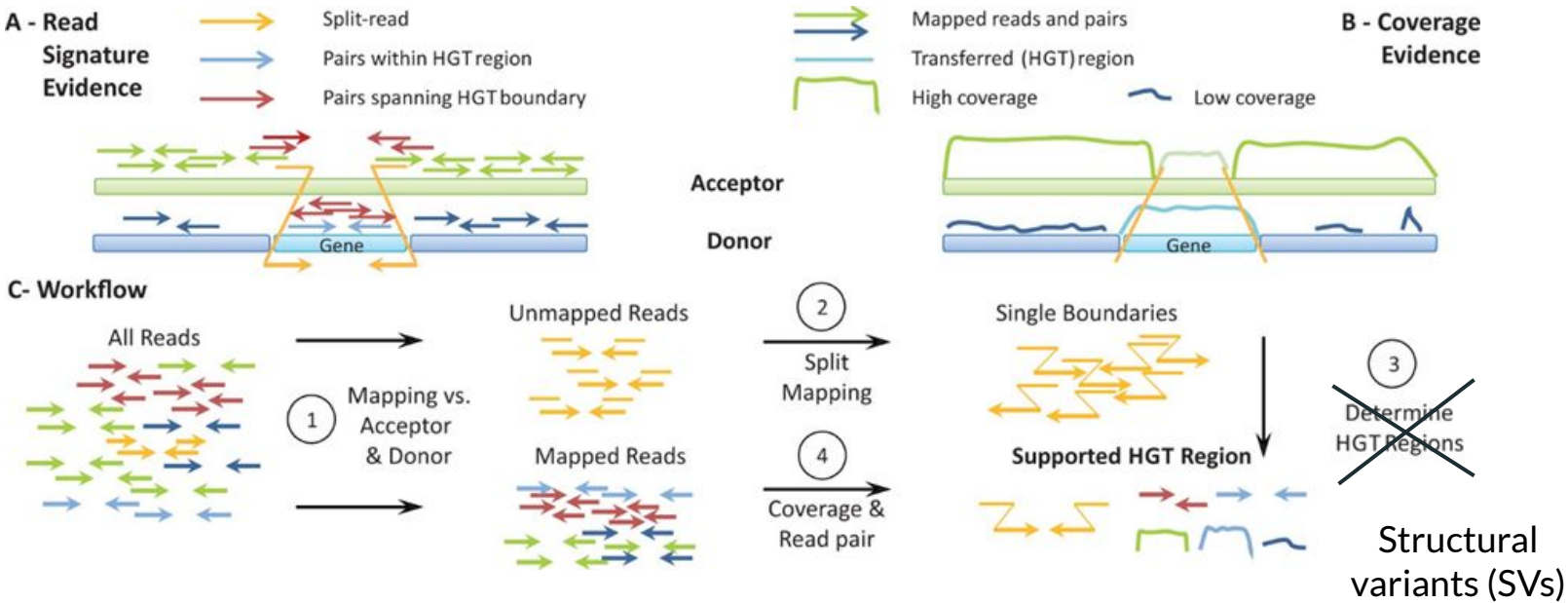




How do detect HGT with MGX?



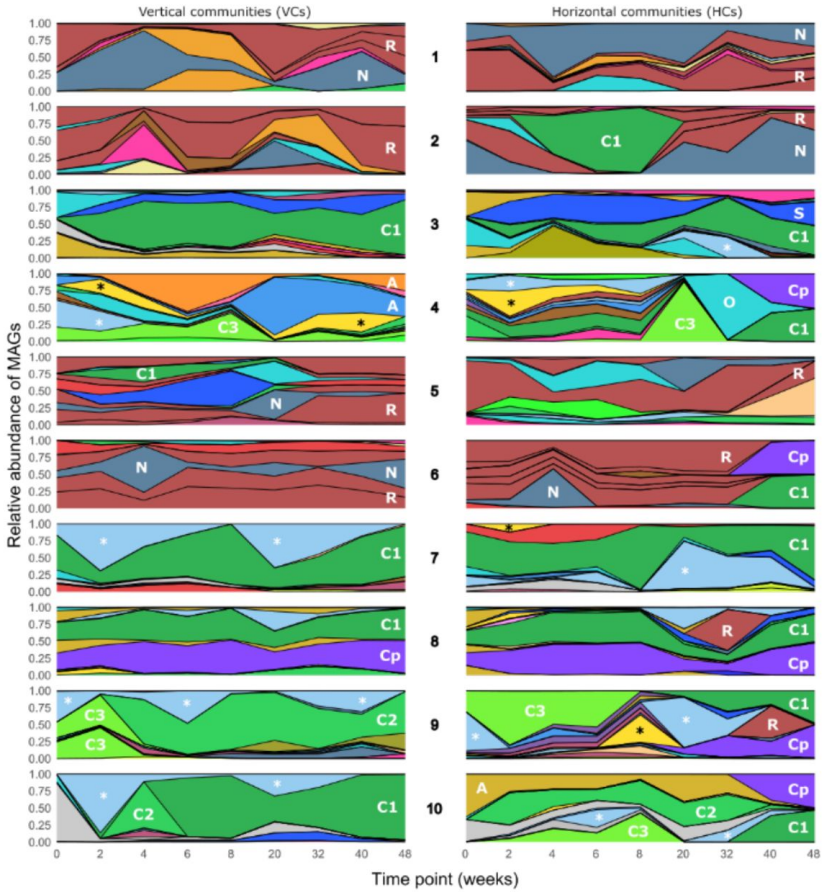
Differential read mapping



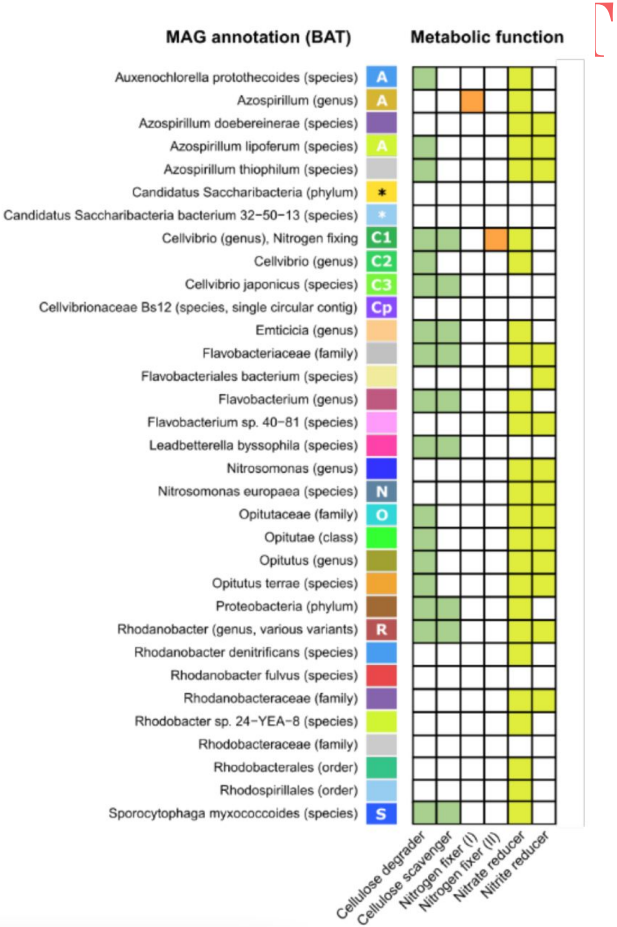
Structural variants (SVs)

Trappe *et al.*, 2016

a. Temporal trajectories of dominant MAGs

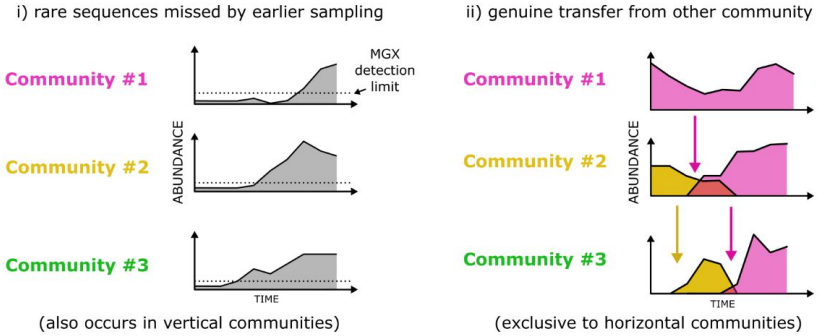


b. Metabolic functions of MAGs

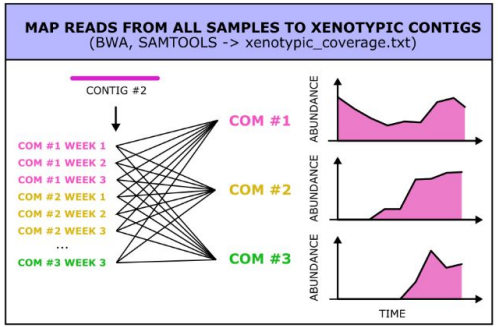


An experimental solution to detect HGT

b. Evolution in horizontal communities yields two types of "unique sequences"



e. Reconstruct dissemination (xenoseq_trace)



How to use xenoseq

1. Install xenoseq (or use the one in the workshop-environment)
2. Prepare a “metadata file”:
3. Run: `$ xenoseq -m xenoseq_metadata.txt -p reads -r _R*.fastq -l -t -o 08_Xenoseq`
4. More help and list of all options: `$xenoseq -h`

5. If you get stuck, I’ve build in an AI that can help you by sending an email to bramvandijk88@gmail.com, but it’s pretty slow sometimes … :(

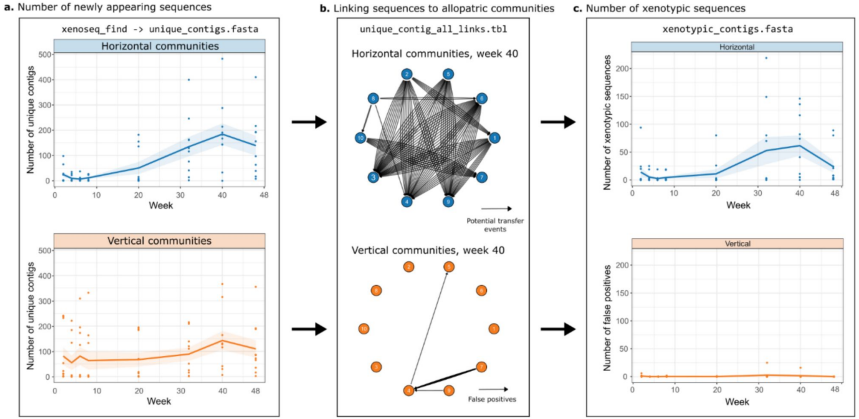
```
#query                  reference
Community_1_T1          Ancestral_1
Community_1_T2          Ancestral_1
Community_1_T3          Ancestral_1
Community_2_T1          Ancestral_2
Community_2_T2          Ancestral_2
Community_2_T3          Ancestral_2
```

Caution: xenoseq works, but biology is messy

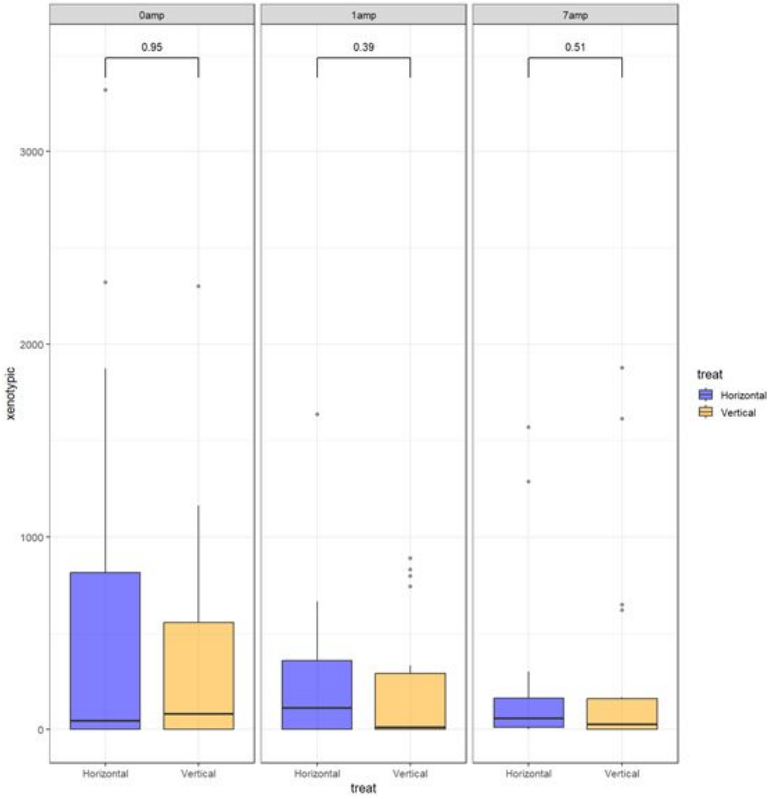
The simulated mock data test:

```
Xenoseq completed for all samples
Found 18 unique contigs in Horizontal1. 16 of these are xenotypic.
Found 10 unique contigs in Horizontal2. 8 of these are xenotypic.
Found 17 unique contigs in Horizontal3. 17 of these are xenotypic.
Found 12 unique contigs in Horizontal4. 10 of these are xenotypic.
Found 0 unique contigs in Vertical1. 0 of these are xenotypic.
Found 1 unique contigs in Vertical2. 0 of these are xenotypic.
Found 0 unique contigs in Vertical3. 0 of these are xenotypic.
Found 0 unique contigs in Vertical4. 0 of these are xenotypic.
Xenoseq finished! :)
```

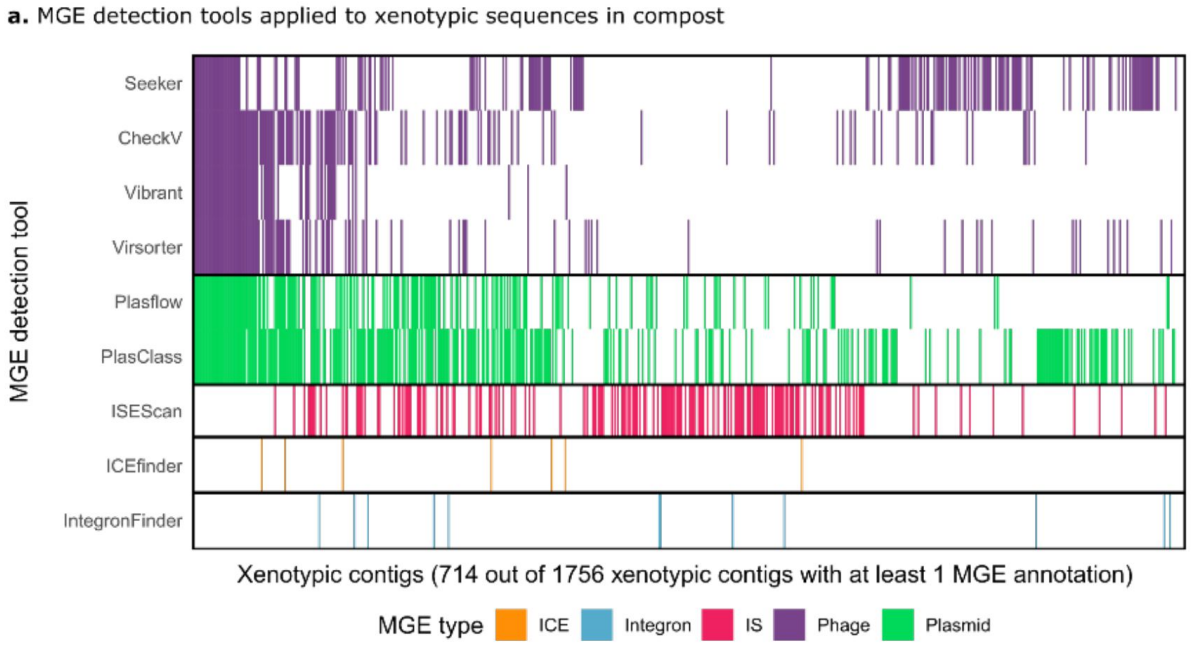
Steven's compost communities:



Ampicilin experiments...

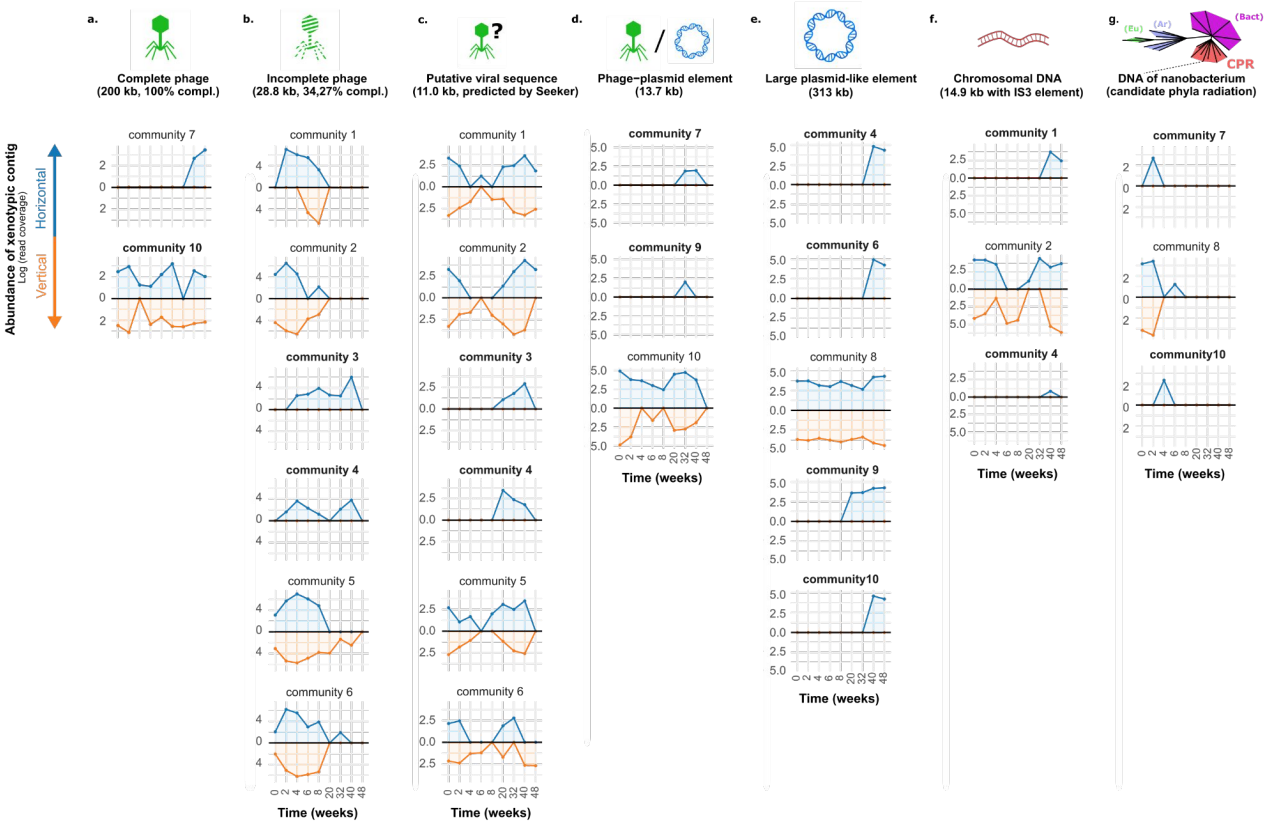


Limits of MGE detection tools:



Movement of MGEs across communities:

"xenotypic_coverage.txt"



MGX take home messages

- Microbes are “too flexible” to simply assume function based on their annotated species. It’s time to move beyond 16S / read annotation
- Instead: try to reserve “annotation” for later stages in the process
- Assembly and read mapping is 90% of what you’ll do first
- It can’t be done with 100% computational tools: experimental approaches are necessary to improve our ability to detect HGT!

Hand-on part of the workshop

What to do?

- **Option 1:** follow instructions in the workshop manual to get some finger exercises on how to do each step. Because MGX is not super fast, running the whole suite will take a while…

- **Option 2:** study the scripts from option 1 without running, and go inspect some of the output that was already run for you at: /groups/mpistaff/MGX_workshop/MGX_workshop_vandijk

- **Option 3:** talk to me or Pauline about what you could do with *your* data. Are you stuck? Do you need help finding a tool? Let us know :)

How do do it?

- On Windows you need to either (i) download a terminal emulator (MobaXterm is my hot tip), or (ii) use our webvpn service: https://webvpn.evolbio.mpg.de/

- Log into Wallace using ssh <username>@wallace.evolbiompg.de

- For webvpn users: your sessions may time out after some time of inactivity. If this happens, try opening up a 'screen' after you login to wallace like this:

```
$ screen
```

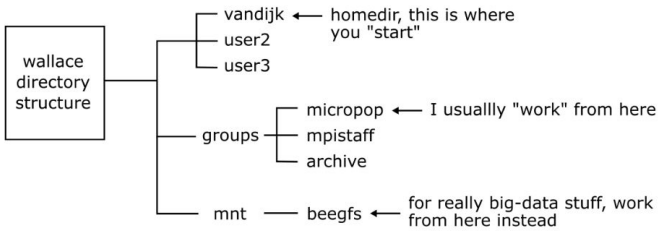
After you timed out (disconnected), you can find your session like this

```
$ screen -list
```

And reconnect like this:

```
$ screen -r <SESSION_ID>
```

Logging in to Wallace: ssh vandijk@wallace



So, to do some stuff in your own directory in micropop:

```
$ cd /groups/micropop
$ mkdir my_directory
$ cd my_directory
$ bash my_script.sh
```

Scripting with Bash (101)

- create/open new script, e.g. nano my_script.sh
- first line should always be: #!/bin/bash
- example script:

```
#!/bin/bash
echo "Start script"
for file in *.fastq.gz; do
   gunzip $file
   echo "Done unzipping $file"
done;
echo "All files are unzipped. Hoping for the best!"
```

To run the script:

```
$ bash my_script.sh
```

Pro tip: if you have a slow script (e.g. one that copies a lot of files), first create a "dummy" script that simply prints the commands without executing them by adding "echo" in front of the steps.

Characters with special meanings:

```
.          current directory
../        parent directory
../../     parent directory of parent directory
/          root directory
>          pipe output to a file
           (e.g. ls *.fastq > all_reads.txt)
>>         pipe output to file (append)
|          pipe output to another program
           (e.g. ls *.fastq | grep "1A")
<          reverse pipe
           (rarely used, but you may encounter it)
&          when typed after a command, it will
           put the process "in the background"
           meaning you will get your prompt back
           and can type more commands while the
           other program is running.
$          access a variable
           (e.g. echo $USER)
$()        store output in variable
           (e.g. variable=$(command_here))
```

Useful commands:

command	description	example
cd	change directory	cd /groups/micropop
ls	list content of dir	ls my/data
cp	copy a file	cp myfile.txt /groups/micropop
mv	move (or rename) file	mv myfile.txt myfile_renamed.txt
cat	print file content	cat myfile.txt
wc	count words, lines or chars	wc -l myfile.txt
grep	find lines with pattern	grep "read_001" myfile.txt
cut	split by delimiter	cat myfile.txt | cut -f2
nano	command-line text editor	nano my_script.sh
gedit	graphical text editor	gedit my_script.sh
sort	sort input	cat myfile.txt | sort -n
uniq	remove duplicate items	cat myfile.txt | sort -n | uniq
mkdir	make new directory	mkdir /groups/micropop/b_obama
echo	print something	echo "Hello!"
gzip	zip file	gzip reads.fastq
gunzip	unzip file	gunzip reads.fastq

Example of a chain of commands:

```
ls *.fastq | grep "1A" | cut -d '_' -f2 > all_samples_with_1A.txt
```