# Advanced de novo assembly

Rayan Chikhi

Pennsylvania State University

MSU NGS Summer Course - June 2013

# slightly more Advanced de novo assembly

Rayan Chikhi

Pennsylvania State University

MSU NGS Summer Course - June 2013

# Course structure

Q : How to create a draft genome on my own ?

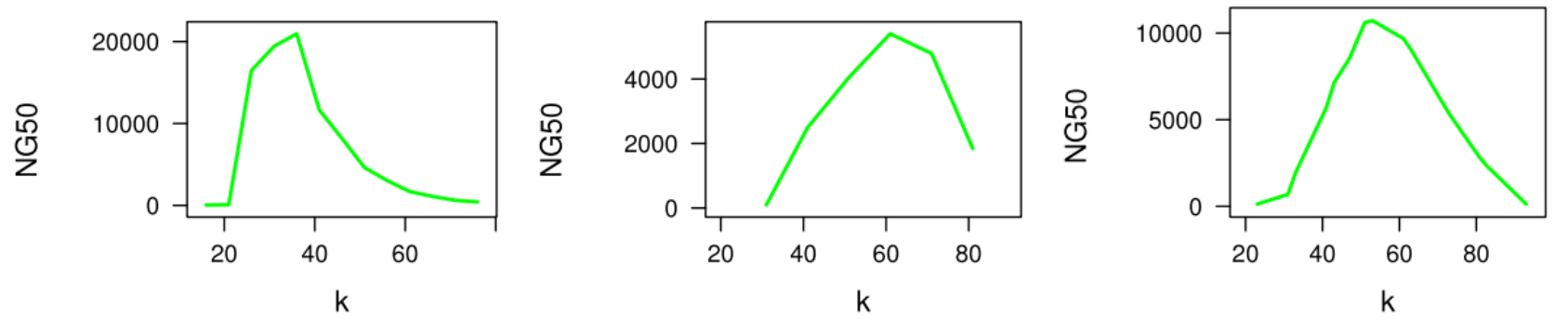
Will talk mostly about tools before/after assembly, but not the biology that follows.

- *k* parameter : how to choose the *k*-mer size (KmerGenie)
- low-memory assembly : running Minia
- Metrics : some recent pointers on assembly evaluation
- Pipelines : there is more than just running Velvet

# THE *k* PARAMETER

Assembly is not robust with respect to the parameter *k*. Because the ideal *k*-mer size depends on :

- sequencing coverage
- sequencing error rate
- genome complexity




*k* vs NG50 for 3 organisms : bacteria (S. aureus), human chr14, whole bumblebee genome (B. impatiens)

# Possibles approaches

There exists two tools to estimate the best $k$, both are designed for Velvet :

- Velvetk : formula based on number of reads, estimated genome size
- VelvetOptimizer : just run Velvet for all values of $k$.

Velvetk does not know about genome complexity and error rate.
VelvetOptimizer takes in the order of CPU-years on $>$ 100 Mbp genomes.

# KmerGenie

Two basic assumptions in DNA/RNA/metaDNA/metaRNA assembly :

- A larger $k$ value allows to resolve more repetitions.
- A smaller $k$ increases the chances of seeing a given $k$-mer.

Thus, one should assemble using the largest $k$-mer size possible, such that the $k$-mer coverage is sufficient.

Apparté : in RNAseq, this partly explains why a single $k$ value is not ideal. Low-abundance transcripts require a small $k$-mer size.

Facts :

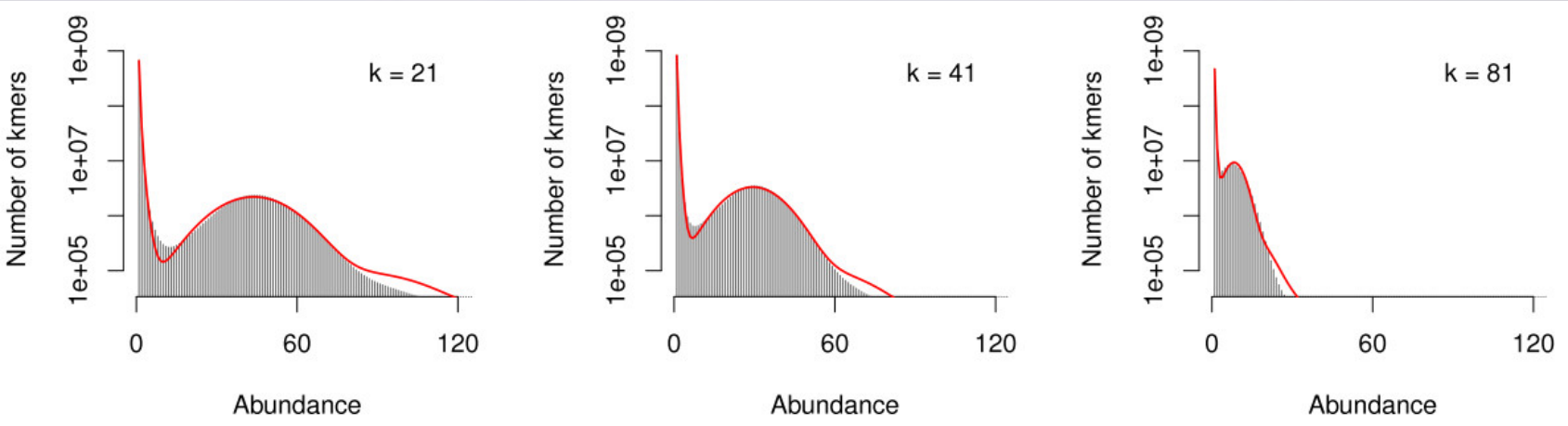
- Resolving repetitions means obtaining more $k$-mers that are present in the genome (*correct k-mers*).
- Increasing the $k$-mer size also means obtaining more erroneous $k$-mers (= $k$-mers containing at least an error).

In conclusion, we want to estimate and maximize the number of correct $k$-mers.

# KMERGENIE

How to estimate the number of correct *k*-mers :

1. Compute *k*-mer histograms for all *k*
2. Correct *k*-mers should be distributed as a Gaussian

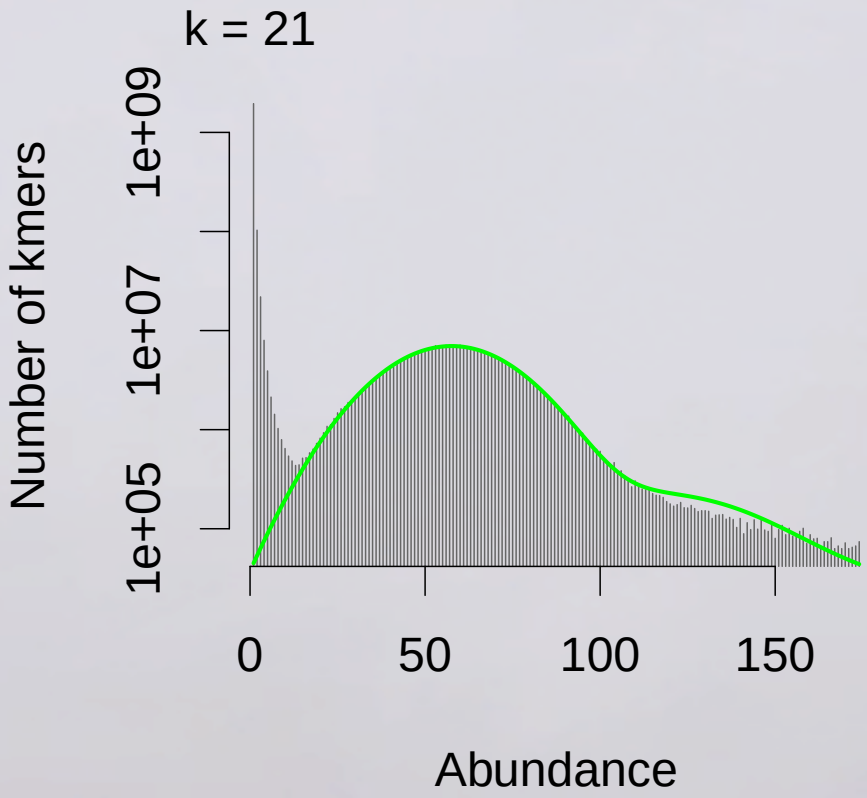



Chr 14 ($\approx$ 88 Mbp ; histograms and fit)

# KmerGenie

How to estimate the number of correct *k*-mers :

1. Compute *k*-mer histograms for all *k*
2. Correct *k*-mers should be distributed as a Gaussian
3. Fit a model to the histograms

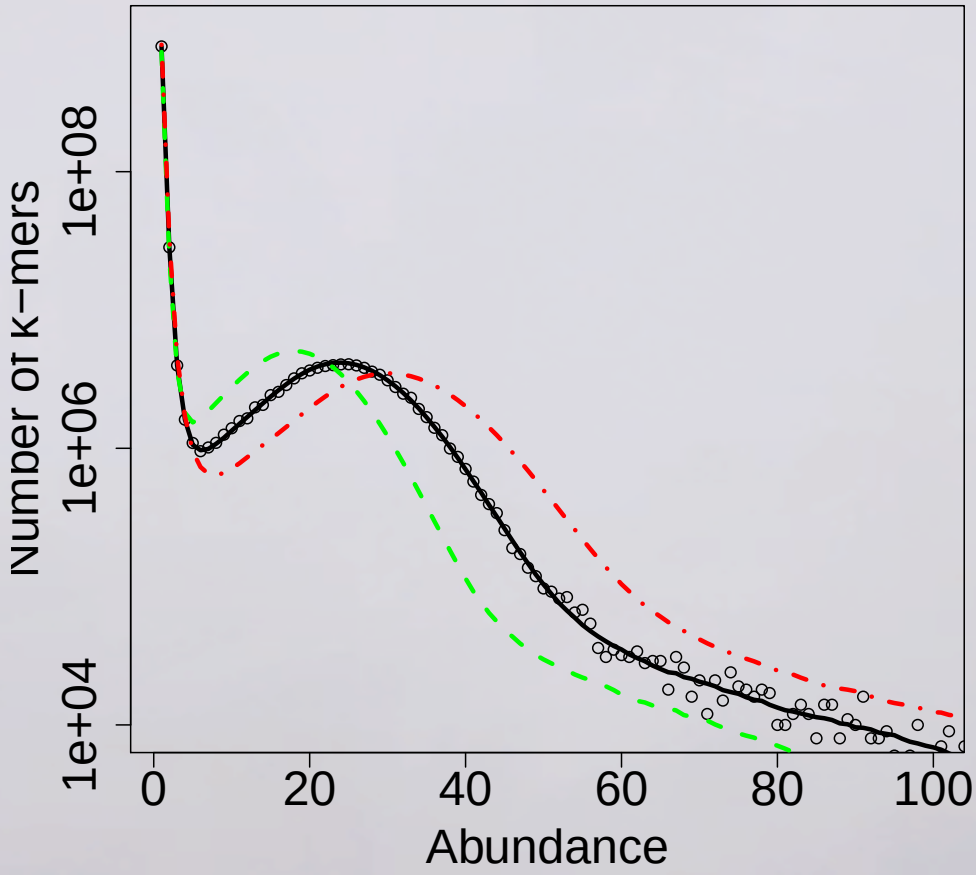



M. persicae (≈ 300 Mbp), k=21 ; green curve is fitted to the putative correct (genomic) *k*-mers component

# KmerGenie

Summary :

- KmerGenie predicts the *k* value which maximizes the assembly size.
- It quickly estimates the histograms using sampling.




Chr 14 (≈ 88 Mbp) ; small dots are the sampled histogram

# KMERGENIE RESULS

Results on the GAGE benchmark :

| Assembly | Contig NG50 (Kbp) | Size (Mbp) | Errors |
|---|---|---|---|
| *S. aureus* (Velvet) | | | |
| $k = 21$ | 0.5 | 7.65 | 0 |
| $\underline{k = 31}$ | **19.4** | **2.83** | 10 |
| $k = 41$ | 11.7 | 2.81 | **6** |
| $k = 51$ | 4.6 | 2.80 | 9 |
| *chr14* (Velvet) | | | |
| $k = 41$ | 2.4 | 74.56 | 764 |
| $k = 51$ | 4.0 | 79.92 | 843 |
| $k = 61$ | **5.4** | **82.10** | 431 |
| $\underline{k = 71}$ | 4.7 | 81.89 | 251 |
| $k = 81$ | 1.8 | 74.18 | **153** |
| *B. impatiens* (SOAPdenovo2) | | | |
| $k = 41$ | 5.4 | 224.05 | |
| $\underline{k = 51}$ | **10.4** | 229.71 | |
| $k = 61$ | 9.5 | **230.36** | |
| $k = 71$ | 5.9 | 226.11 | |
| $k = 81$ | 2.5 | 207.11 | |

**Table 1.** Quality of assemblies for different values of $k$. The value of $k$ predicted by KMERGENIE is underlined.

# USING KMERGENIE

```
curl http://kmergenie.bx.psu.edu/kmergenie-1.5356.tar.gz | tar xz
cd kmergenie-1.5356
make
```

Usage for a single file :

```
./kmergenie reads.fastq
```

Usage for a list of files :

```
ls -1 *.fastq > list_reads
./kmergenie list_reads
```

What is returned :

```
[..]
best k: 47
```

As well as a set of kmer histograms to visualize.

# MINIA

# MINIA

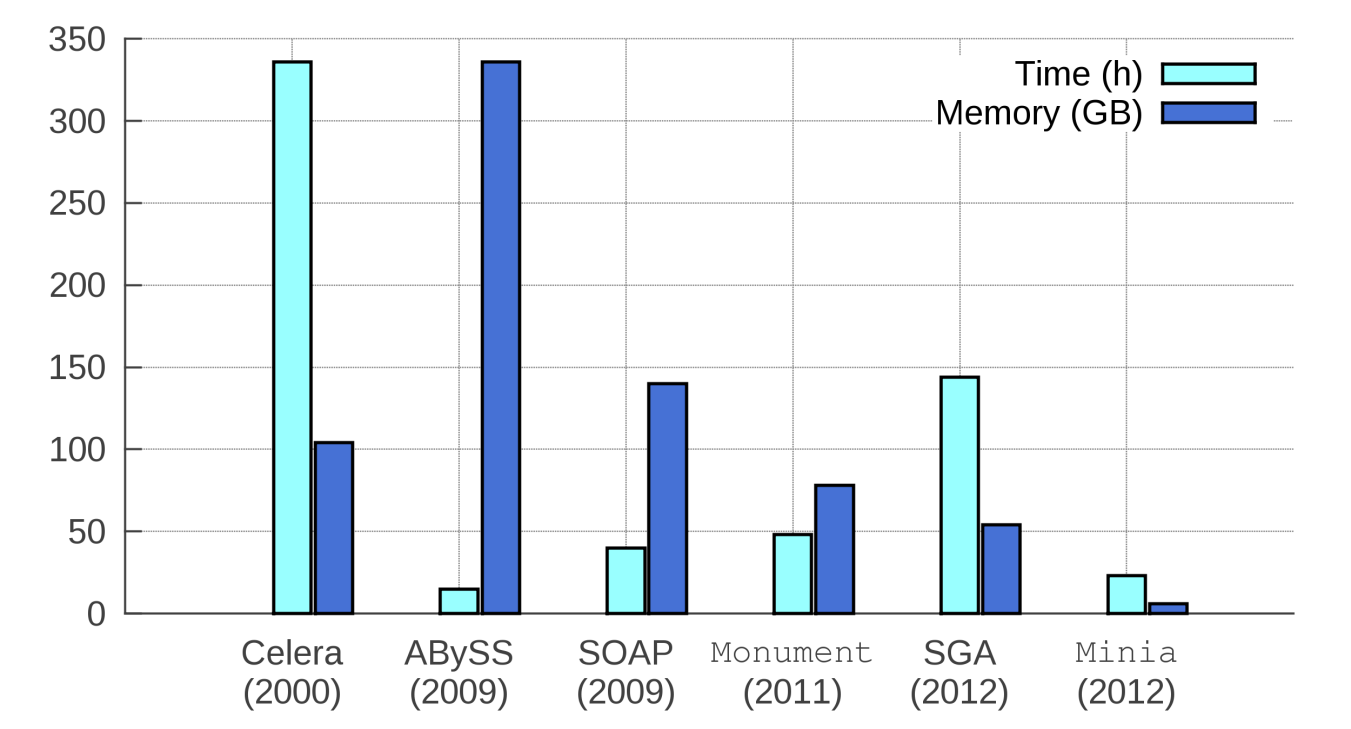
Minia is the lowest-memory de novo genome assembler to date.

History :

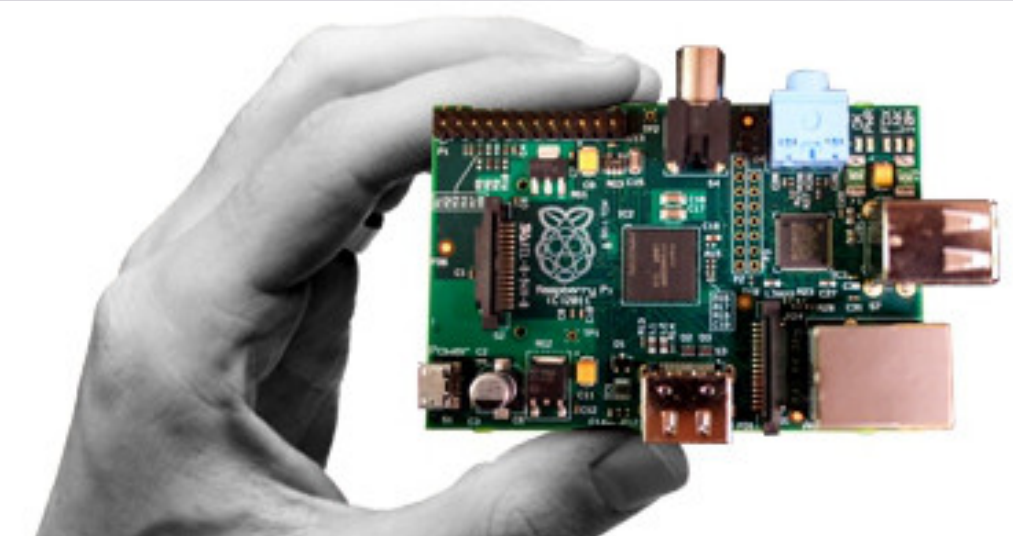
1. Titus' group published a technical khmer paper on arXiv, they obtained inexact de Bruijn graphs extremely efficiently
2. We extended the idea to make the graphs exact and cited their arXiv paper
3. This was applied to genome assembly (Minia), local RNA-seq assembly (KisSplice), and other software in preparation

# MINIA

Minia assembles a human genome in ≈ a day on a desktop computer,




and recently *C. elegans* on a Raspberry PI.



# Assembly quality of Minia

- Minia only creates contigs
- Thus, Minia is not a complete assembly pipeline like SOAPdenovo2 or Allpaths-LG
- Assembly contiguity and quality is $\approx$ Velvet
- No claim to perform better than SOAPdenovo2 or Allpaths-LG

# USING MINIA

Installation :

```
curl http://minia.genouest.org/files/minia-1.5316.tar.gz | tar xz
cd minia-1.5316
make
```

Then launch :

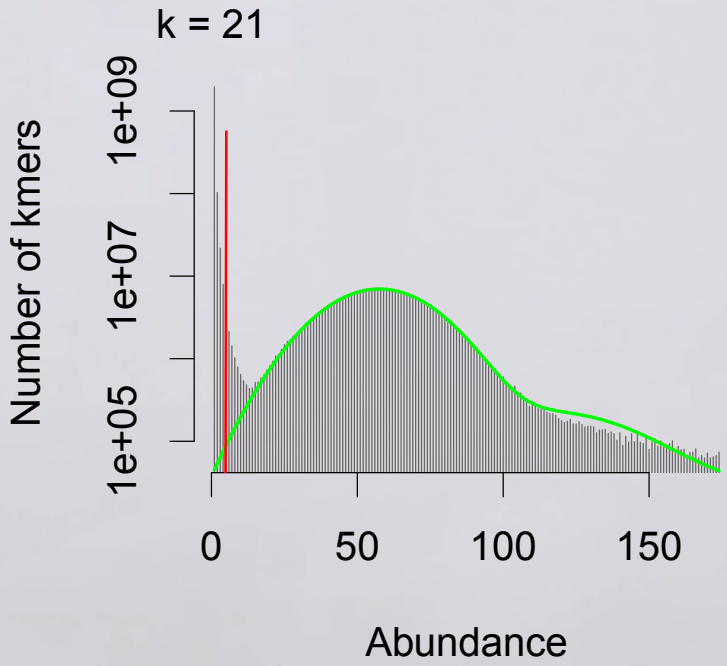
```
./minia reads.fq kmer_size cov_cutoff estimated_genome_size output
```

Setting cov_cutoff to 3 for usual sequencing coverages (30x-100x) is reasonable, but it is good practice to inspect KmerGenie plots.

Result :

```
output.contigs.fa
```

```
# head histograms-k21.histo
1       1950283000
2       103024000
3       21882000
4       7967000
5       3912000
```




A possible cov_cutoff value (5, shown as a red line), realizing a compromise between retaining most correct *k*-mers and discarding many erroneous ones.

# METRICS

# METRICS

**Preamble :** There is no trivial total order (i.e. ranking) between assemblies. A compromise is generally made.

**Why ?** $>$ 2 independent criteria to optimize (e.g., total length, and average size of assembled sequences)

**Example** Would you rather have an assembly with good coverage and short contigs, or an assembly with mediocre coverage and long contigs ?

# Overview of reference-free metrics

Assume you have no close reference genome available.
Metrics serve two purposes :

1. Individually evaluate a single assembly
2. Compare several assemblies made from different parameters or assemblers

Classical metrics :

- Number of contigs/scaffolds
- Total length of the assembly
- Length of the largest contig/scaffold
- Percentage of gaps in scaffolds ('N')
- N50/NG50 of contigs/scaffolds
- Number of predicted genes
- Number of core genes [CEGMA]

An easy tool to compute most of these is QUAST :

```
./quast.py assembly.fa
```

Recent assembly metrics are mostly based on :

- internal consistency
- likelihood of then assembly given the reads

# REFERENCE-FREE METRICS : N50

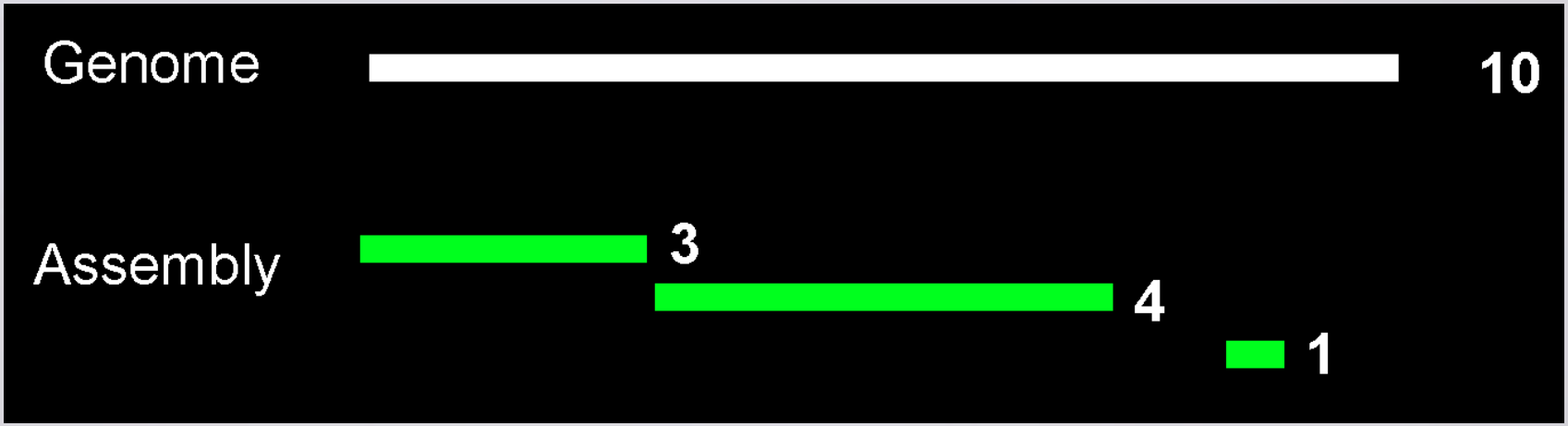
Let's do this slide only if a majority wants.

# REFERENCE-FREE METRICS : N50

Let's do this slide only if a majority wants.

N50 = Largest contig length at which longer contigs cover 50% of the total **assembly** length

NG50 = Largest contig length at which longer contigs cover 50% of the total **genome** length




A practical way to compute N50 :

- Sort contigs by decreasing lengths
- Take the first contig (the largest) : does it cover 50% of the assembly ?
- If yes, this is the N50 value. Else, try the next one (the second largest), and so on..

# INTERNAL CONSISTENCY

Rarely appears in assembly articles but almost the only way to detect errors in *de novo* assemblies.

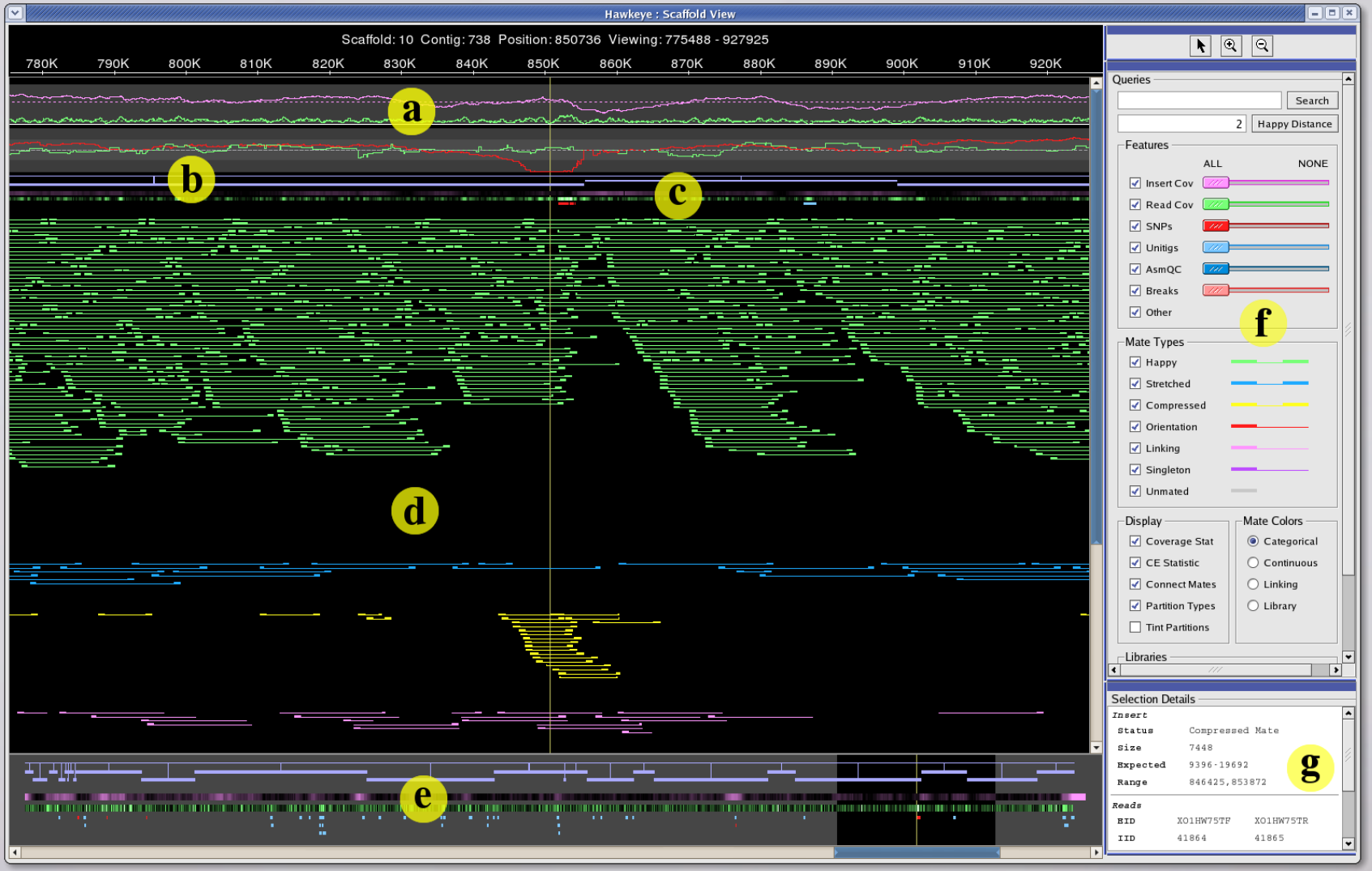
**Internal consistency** : Percentage of paired reads correctly aligned back to the assembly (*happy* pairs).

Can also pinpoint certain misassemblies (mis-joins).
Recent tools :

- REAPR[1] [M Hunt, .. (Gen. Biol.) 2013]
- FRCurve[2] [F. Vezzi, .. (Plos One) 2013]

[1] Google : REAPR assembly
[2] Google : FRCurve

# INTERNAL CONSISTENCY : EXAMPLE




Hawkeye software

# Assembly likelihood (1)

Principle : for an assembly $A$ and a set of reads $R$,

$$P(R|A) = \prod_i P(r_i|A)$$

Where each $p(r_i|A)$,

- is the probability that the read $r_i$ is sequenced if the genome was $A$.
- In practice, $p(r_i|A)$ can be estimated by aligning $r_i$ to the assembly.

Recent software :

- ALE [S. Clark, .. (Bioinf.) 2013]
- CGAL [A. Rahman, .. (Gen. Biol.) 2013]
- a third one from M. Pop's group

# ASSEMBLY LIKELIHOOD (2)

From my exp., ALE is easier to use/faster, but still not fully automated (needs you to pre-align the reads).

```
./ALE reads_aligned_to_assembly.sam assembly.fa
```

Returns :

```
ALE_score: -194582491.814571
```

# ASSEMBLY LIKELIHOOD (3)

**Likelihoods of GAGE assemblies of human chromosome 14**

| Assembler | Likelihood | Number of reads mapped | Coverage (%) | Scaffold N50 (kb) | Contig N50 (kb) |
|---|---|---|---|---|---|
| ABySS | $-23.44 \times 10^8$ | 22096466 | 82.22 | 2.1 | 2 |
| ALLPATHS-LG | $-22.77 \times 10^8$ | 23122569 | 97.24 | 81647 | 36.5 |
| CABOG | $-21.26 \times 10^8$ | 23433424 | 98.32 | 393 | 45.3 |
| SOAPdenovo | [a] | [a] | 98.17 | 455 | 14.7 |
| Reference | $-19.04 \times 10^8$ | 23978017 | - | - | - |

[a] Likelihood not computed as reads could not be mapped with Bowtie 2.

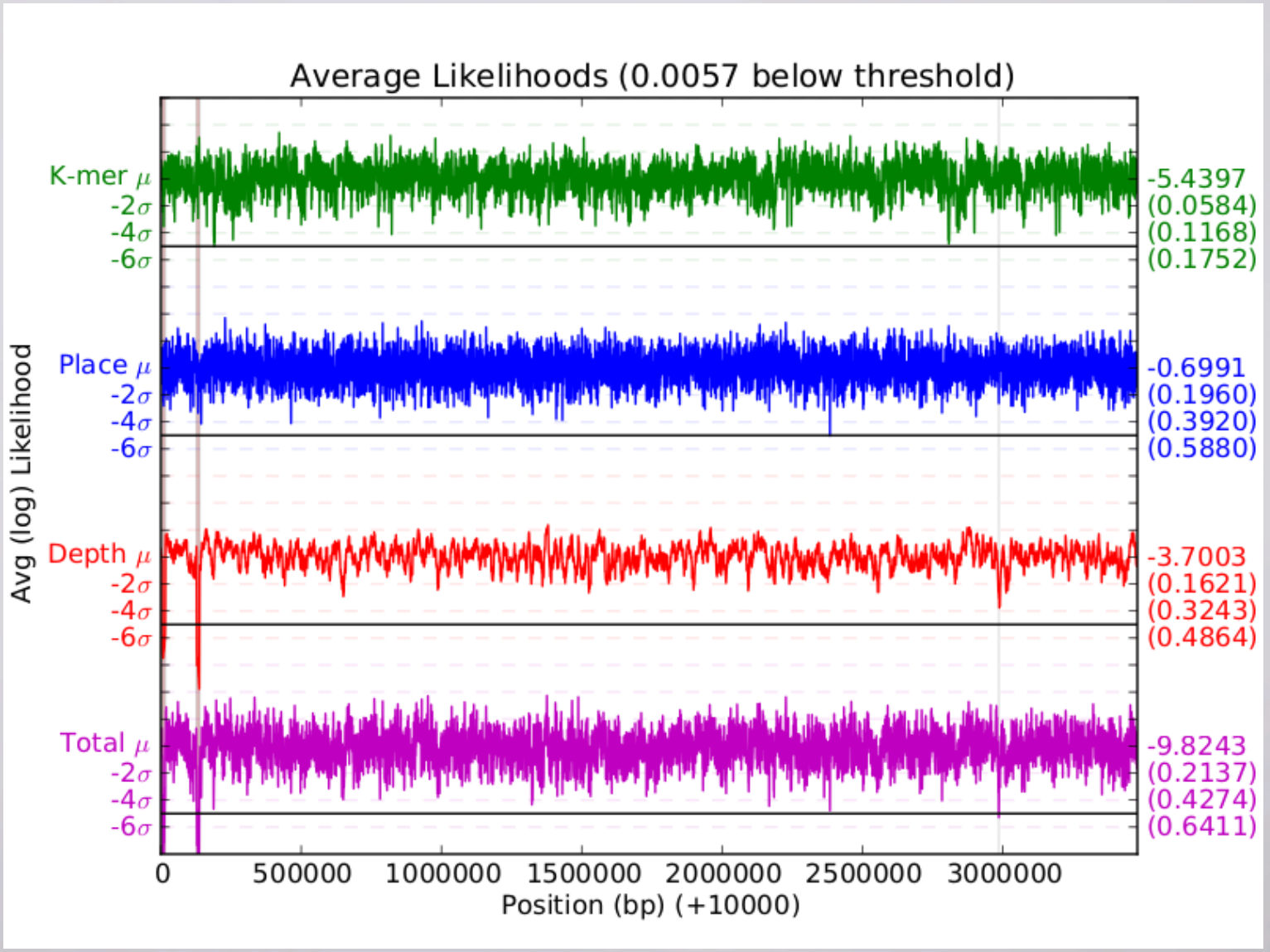
Rahman and Pachter *Genome Biology* 2013 **14**:R8 doi:10.1186/gb-2013-14-1-r8

(higher likelihood is better)

Likelihood-based metrics are comparative ; i.e. computing them for a single assembly would be meaningless.

# ASSEMBLY LIKELIHOOD (4)

ALE can also plot the average likelihood over the genome.



# Summary

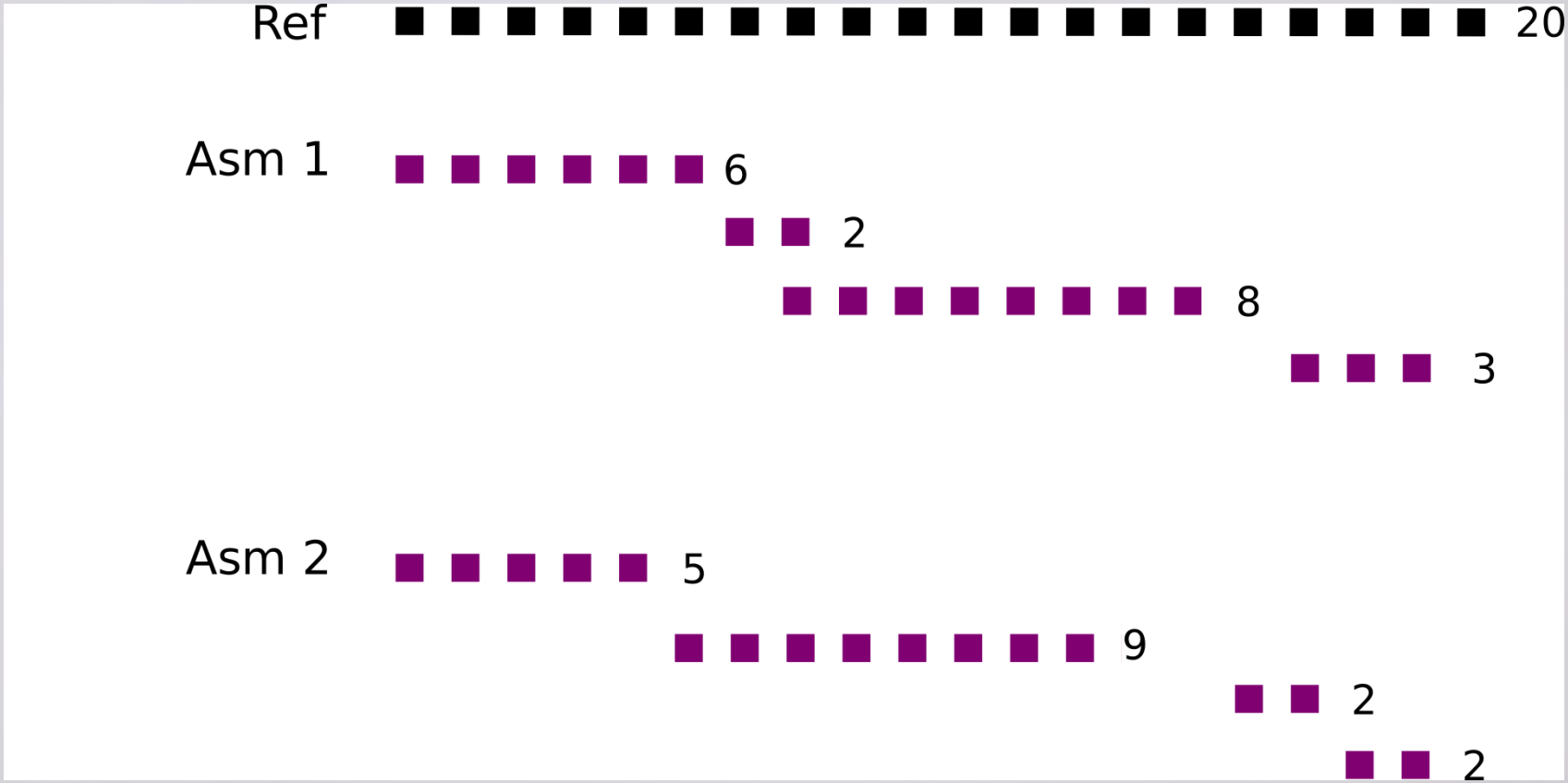
Google 'assembly uncertainty' for a nice summary, blog post by Lex Nederbragt.
In summary :

- No total order for metrics
- Use QUAST
- Use CEGMA
- Try ALE

I am unsure if likelihood-based metrics are very robust indicators, might favor high-coverage assemblies..

# EXERCICE

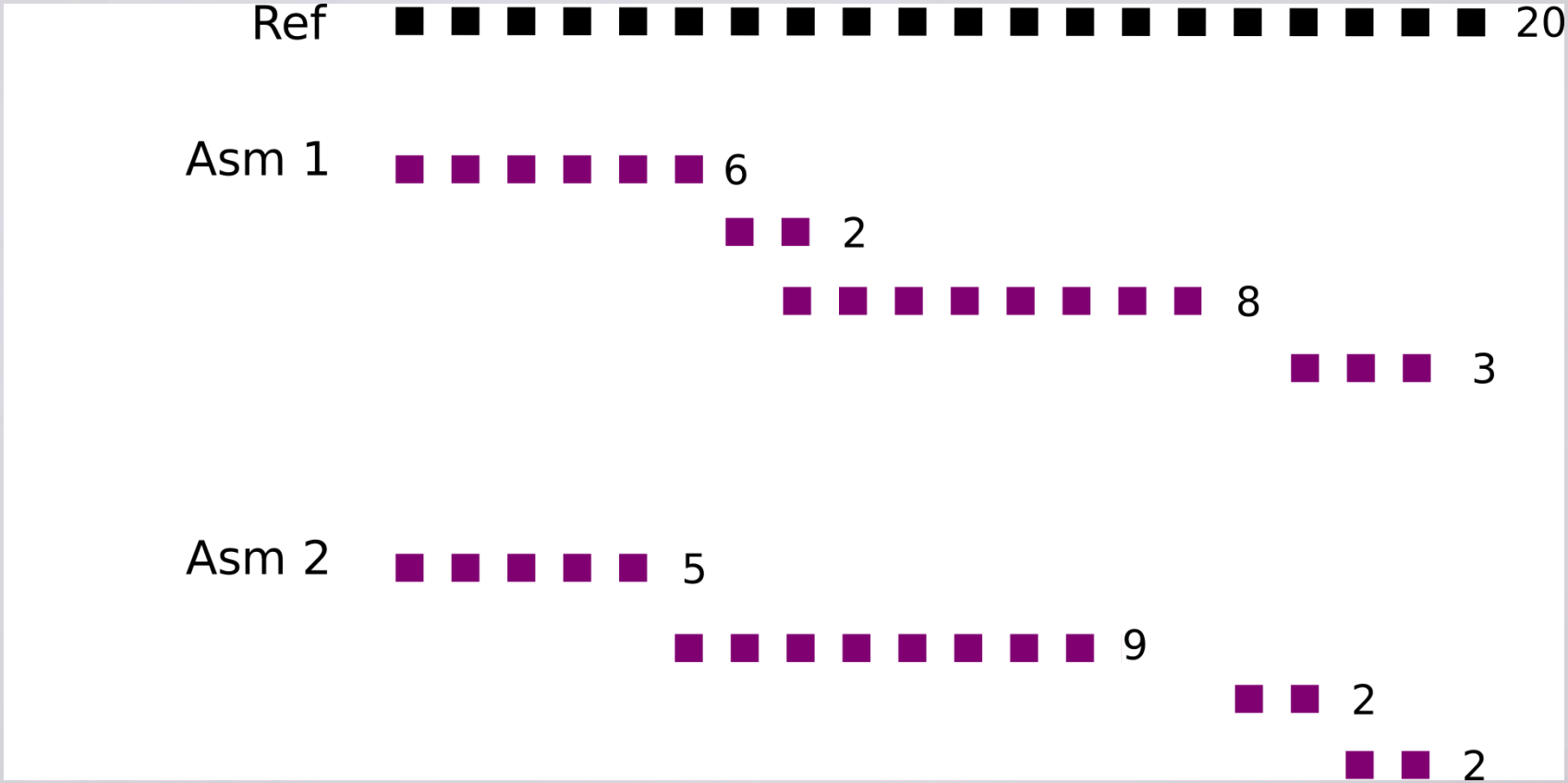
Here are two assemblies, aligned to the same reference :




- For each, compute the following metrics :
  - Total size of the assembly, N50, NG50 (bp)
  - Coverage (%)
- Which one is better than the other ?

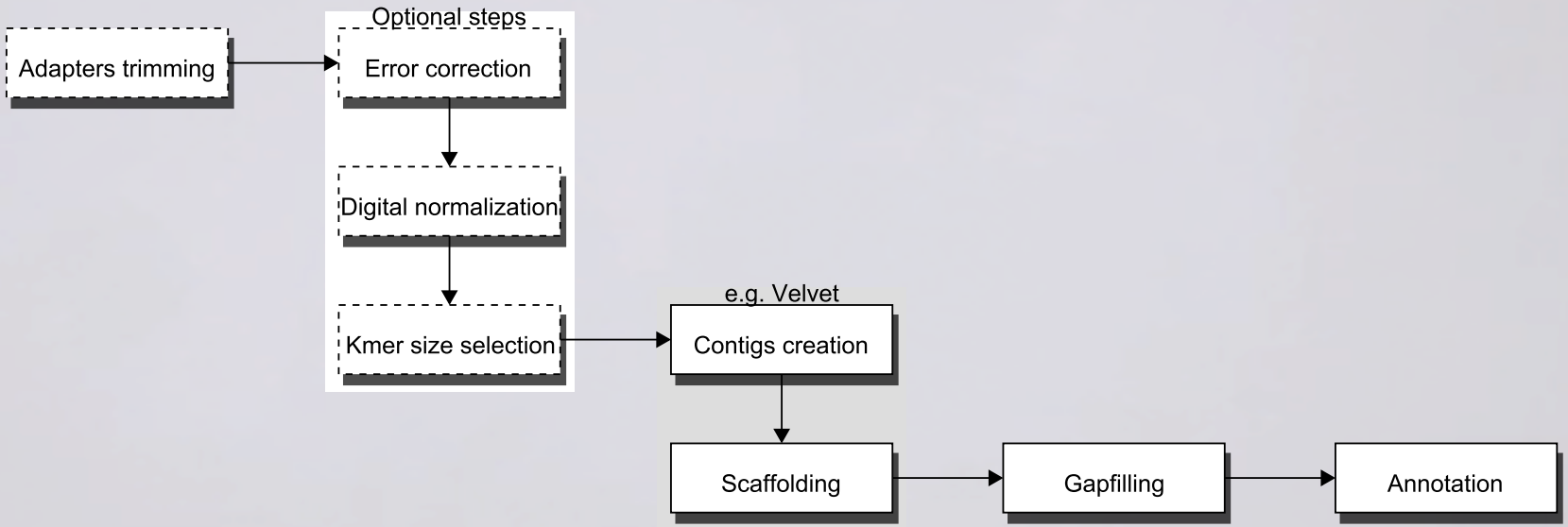
# EXERCICE (SOLUTION)

Here are two assemblies, aligned to the same reference :

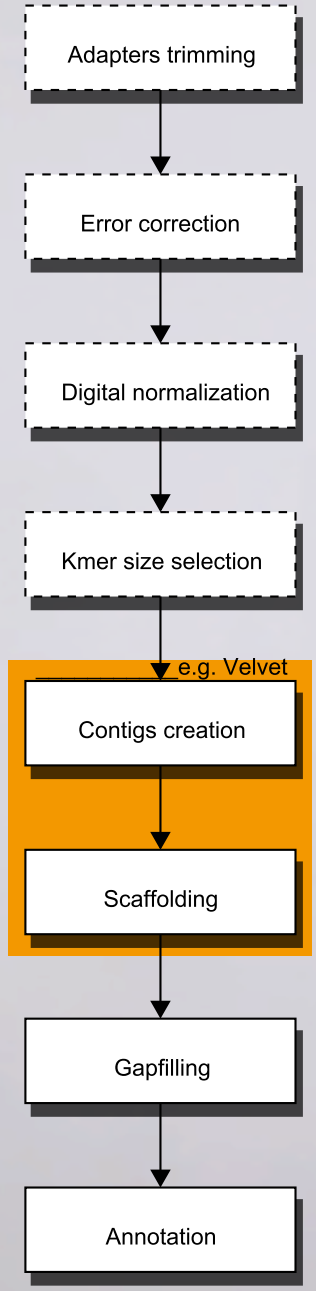



- For each, compute the following metrics :
  - Total size of the assembly (19 bp, 18 bp), N50 (6 bp, 9 bp), NG50 (6 bp, 5 bp)
  - Coverage (%) (90, 90)
- Which one is better than the other ? (I would say first one)

# ASSEMBLY PIPELINES



# SCAFFOLDERS




Scaffolding is the step that maps paired reads to contigs to order them.

Most assemblers include a scaffolder (SOAPdenovo, SGA, ABySS, Velvet, Newbler..).

Scaffolding is where most assembly errors are likely to be made.

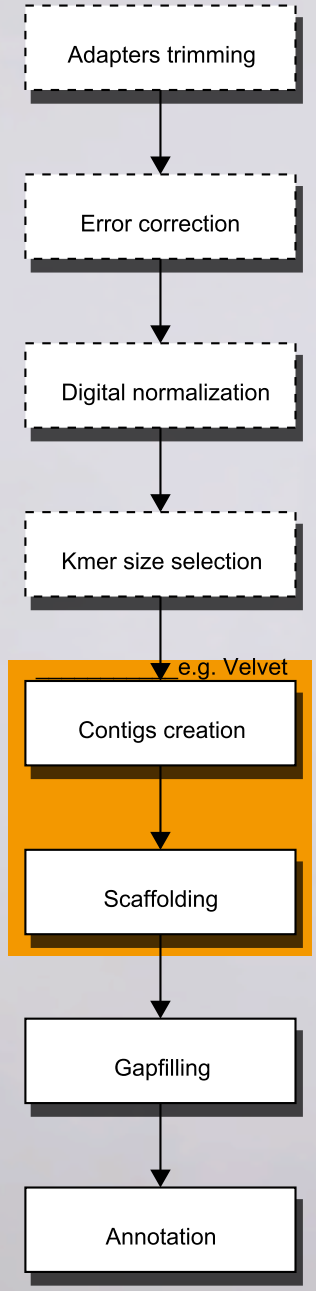
If [your assembler]'s scaffolder did not work for you :

- Use another assembler's scaffolder (e.g. SOAPdenovo2)
- Use a stand-alone scaffolders (e.g. SSPACE, Bambus 2, Opera, etc..)
- Avoid performing scaffolding, for some applications contigs are good enough.

SSPACE is easy to use :

```
perl SSPACE_Basic_v2.0.pl \
  -l small_config_file.txt -s assembly.fa
```

# GapFillers




Gap-filling is the step that fills the gaps inside scaffolds.

Gap-filling can increase contigs length by an order of magnitude. But mistakes may happen at short tandem repeats.

Few assemblers include a gap-filler (SOAPdenovo, Allpaths-LG) :

- SOAPdenovo2 GapCloser can be used standalone, Allpaths not.
- There exists stand-alone gap-fillers (GapFiller, FinIS), but they have limitations.

GapCloser is quite easy to use :

```
./GapCloser -b soap_config_file \
 -a contigs.fa -o scaffolds
```

# Personal experience

If I was in a hurry, and had to choose a single assembler

| | |
|---|---|
| **Your data follows the Broad recipe** | Allpaths-LG |
| **General purpose** | SOAPdenovo2 |
| **If not enough memory** | Minia |
| **454** | Newbler |
| **RNA-Seq** | Trinity |
| **Metagenome** | RayMéta ( ?) |

# SUMMARY : TO CREATE A DRAFT GENOME FROM SHORT READS

Step by step :

1. Prior to sequencing : ask for the Broad recipe, if possible
2. Read the GAGE and/or Assemblathon 2 paper
3. Pick one (two is better) assemblers from the papers above
4. Run each assembler with several sets of parameters
5. Run a program to compare these assemblies

For bacterial genomes, another option is PacBio, it looks increasingly interesting.