de novo assembly

Rayan Chikhi

CNRS

Workshop On Genomics - Cesky Krumlov - January 2015

Your instructor is..

- CNRS researcher in Bioinformatics, France
- Postdoc at Penn State, USA

Research:

- I've made software and methods for *de novo* assembly of short reads:
  - Minia
  - KmerGenie
  - DSK
- I've worked with biologists on real assembly projects

@RayanChikhi on Twitter

`http://rayan.chikhi.name`

Questions to the audience

- Already have data to assemble?
- Plans to sequence *de novo*?
- RNA-Seq?
- PacBio?

Course structure

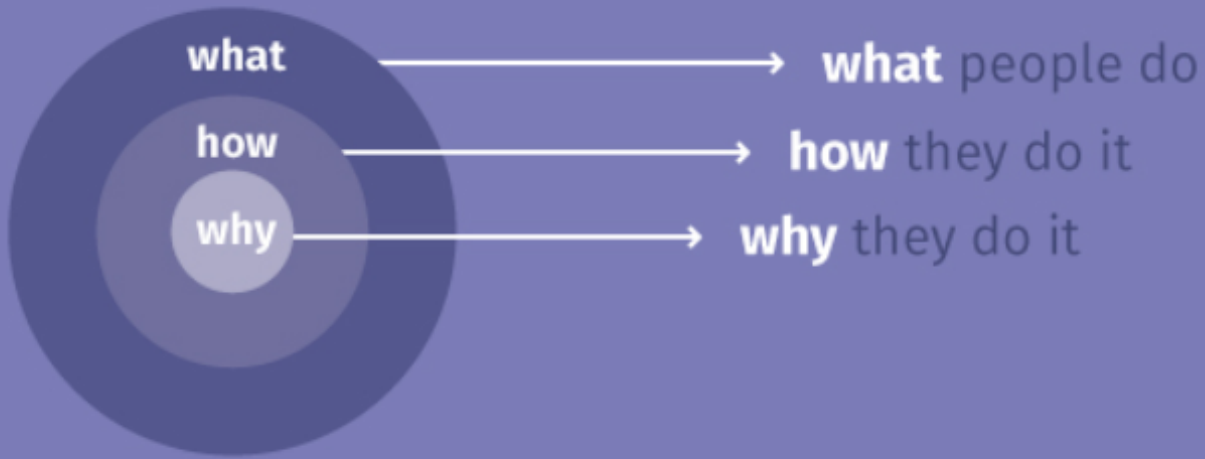
- Short intro
- Basic definitions
- Fundamentals: understanding why assemblies are as they are
- Metrics: methods to evaluate an assembly
- RNA-Seq: how Trinity works
- Pipelines: pre and post assembly

Motivation

"The Golden Circle"

TED TALK: How great leaders inspire action
by Simon Sinek

The "Golden Circle" discusses three topics that come up in presentations:



Source: 8 Ted Talks That Teach Public Speaking (INFOGRAPHIC)

The "Why"

- Create a reference genome / assemble a transcriptome
- Just interested in the genes
- Find novel insertions
- Make sense of un-mapped reads
- Discover SNPs on non-model organisms
- Validate breakpoints
- Recover a specific region of interest
- Explore metagenomics
- ..

"What" and "How" based on "Why"

Scenario 1:

What FASTA file containing "The Best Possible Assembly" of bacteria X

How high-coverage coverage of PacBio

Why Obtain a reference geome

Scenario 2:

What FASTA file containing some draft assembly of organism X

How couple of Illumina lanes

Why A biological question: gene content? viral insertions?

Assembly difficulty

DNA assembly is still a difficult problem in 2015.

1. High computational resources requirements
2. Hard to obtain good assemblies with Illumina data

   Conclusions of the GAGE benchmark : in terms of assembly quality, there is no single best de novo assembler

3. PacBio methods are quite preliminary

State of the research

1. Data-specific assemblers (esp. PacBio)
2. Low-memory assemblers
3. Best practices (protocols) papers
4. Assembly techniques for other purposes (e.g. variant calling)

Plan

Definition of an **assembly** (a trickier question than it seems)

Set of sequences which best approximate the original sequenced material.

Some assembly intuition

Simple facts, an assembly is generally:

- smaller than the reference,
- fragmented

Some assembly intuition

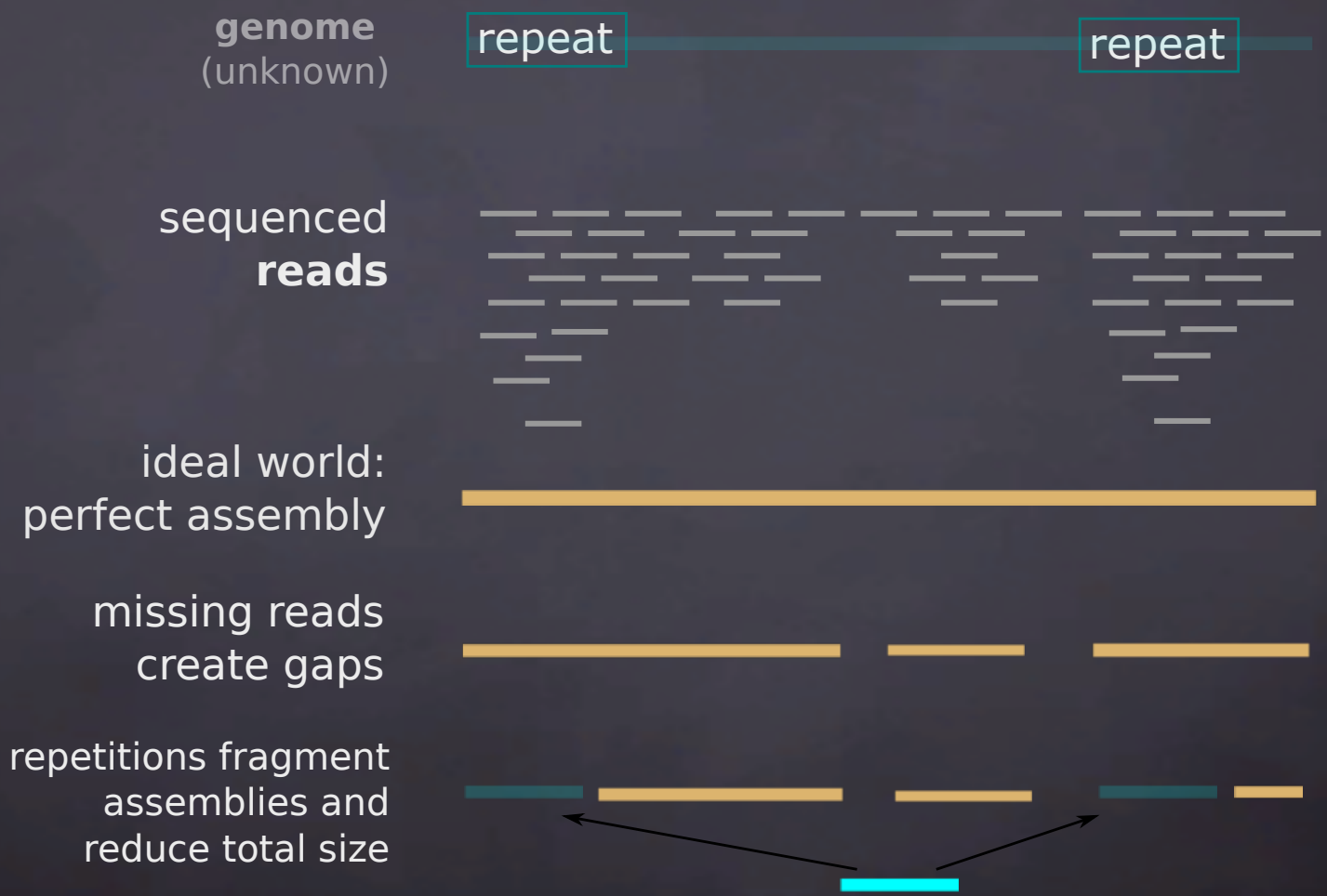
Simple facts, an assembly is generally:

- smaller than the reference,
- fragmented

Some assembly intuition

Simple facts, an assembly is generally:

- smaller than the reference,
- fragmented



Some vocabulary:

Read	Any sequence that comes out of the sequencer
Paired read	$read_1$, gap $\leq$ 500 bp, $read_2$
Mate-pair	$read_1$, gap $\geq$ 1 kbp, $read_2$
Single read	Unpaired read
***k*-mer**	Any sequence of length k
Contig	gap-less assembled sequence
Scaffold	sequence which may contain gaps (N)

Exercise

Here is a set of reads:

```
TACAGT
  CAGTC
   AGTCA
      CAGA
```

1. How many k-mers are in these reads (including duplicates), for $k = 3$?
2. How many *distinct* k-mers are in these reads?
   - (i) for $k = 2$
   - (ii) for $k = 3$
   - (iii) for $k = 5$
3. It appears that these reads come from the (toy) genome TACAGTCAGA. What is the largest k such that the set of distinct k-mers in the genome is exactly the set of distinct k-mers in the reads above?
4. For any value of k, is there a mathematical relation between N, the number of k-mers (incl. duplicates) in a sequence, and L, the length of that sequence?

Exercise (solution)

Here is a set of reads:

```
TACAGT
  CAGTC
   AGTCA
      CAGA
```

1. How many k-mers are in these reads (including duplicates), for $k = 3$? 12
2. How many *distinct* k-mers are in these reads?
   - (i) for $k = 2$, 7
   - (ii) for $k = 3$, 7
   - (iii) for $k = 5$, 4
3. It appears that these reads come from the (toy) genome TACAGTCAGA. What is the largest k such that the set of distinct k-mers in the genome is exactly the set of distinct k-mers in the reads above? 3; for k=4, TCAG does not appear in the reads
4. For any value of k, is there a mathematical relation between N, the number of k-mers (incl. duplicates) in a sequence, and L, the length of that sequence? $N = L - k + 1$

Plan

Graphs

A graph is a set a nodes and a set of edges (directed or not).

Graphs for sequencing data

Overlaps between reads is the fundamental information used to assemble.
Graphs permit to represent these overlaps.

Two different types of graphs for sequencing data are known:

- de Bruijn graphs — Used with Illumina data
- string graphs — Used with PacBio and 454 data

A bioinformatician who knows those graphs will understand:

- how to set the parameters of an assembler
- the type of errors that assemblers make
- why assemblies do not retain variants
- why some heterozygous sites appear twice

de Bruijn graphs

This is going to be fundamental for **Illumina** data. A **de Bruijn** graph for a fixed integer k:

1. **Nodes** = all *k-mers* (k-length sub-strings) present in the reads.
2. For each $(k+1)$-mer x present in the reads, there is an **edge**[1] between the k-mer prefix of x and the k-mer suffix of x.

Exemple for $k = 3$ and a single read:

```
ACTG
```

[1] In this lecture, I am using the edge-centric de Bruijn graph definition. The node-centric definition is when edges correspond to exact $(k-1)$-overlaps between nodes, and $(k+1)$-mers are never considered.

de Bruijn graphs

Example for many reads and still $k = 3$.

```
ACTG
 CTGC
  TGCT
```

de Bruijn graphs: redundancy

What happens if we add redundancy?

```
ACTG
ACTG
 CTGC
 CTGC
 CTGC
  TGCT
  TGCT
```

dBG, $k = 3$:

de Bruijn graphs: errors

How is a sequencing error (at the end of a read) impacting the de Bruijn graph?

```
ACTG
 CTGC
 CTGA
  TGCT
```

dBG, $k = 3$:

de Bruijn graphs: SNPs

What is the effect of a SNP (or a sequencing error inside a read) on the graph?

```
AGTCTGA
AGTTTGA
```

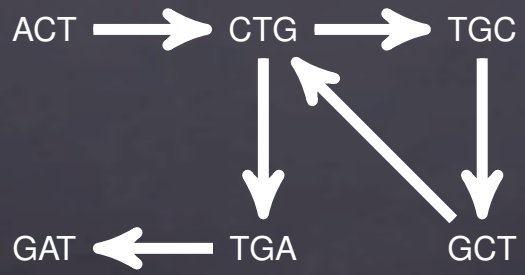
dBG, $k = 3$:

de Bruijn graphs: repeats

What is the effect of a small repeat on the graph?

```
ACTG
 CTGC
  TGCT
   GCTG
    CTGA
     TGAT
```

dBG, $k = 3$:



String graphs: overlap graphs

This is going to be fundamental for **PacBio** data. Definition of an **overlap graph**. It is *almost* a string graph.

1. **Nodes** = reads.
2. Two nodes are linked by an **edge** if both reads overlap[2].

Example for $k = 3$ and a single read:

```
ACTG
```

ACTG

[2] The definition of overlap is voluntarily fuzzy, there are many possible definitions.

Overlap graphs

Given $k > 0$, we say that r and r' **overlap** if a suffix of r of length $l > k$ is *exactly* a prefix of r' of similar length.

Overlap graph for $k = 3$,

`ACTGCT`
`CTGCT` (overlap of length 5)
`GCTAA` (overlap of length 3)

String graphs: overlap graphs

A **string graph** is obtained from an overlap graph by removing redundancy:

- redundant reads (those fully contained in another read)
- transitively redundant edges(if $a \to c$ and $a \to b \to c$, then remove $a \to c$)

From Overlap graphs to String graphs

Overlap graph for $k = 3$,

String graph for $k = 3$,

The read CTGCT is contained in ACTGCT, so it is redundant

Comparison string graph / de Bruijn graph

On the same example, compare the de Bruijn graph with the string graph:

```
ACTGCT
 CTGCTA
   GCTAA
```

String graph, $k = 3$:

de Bruijn graph, $k = 3$:

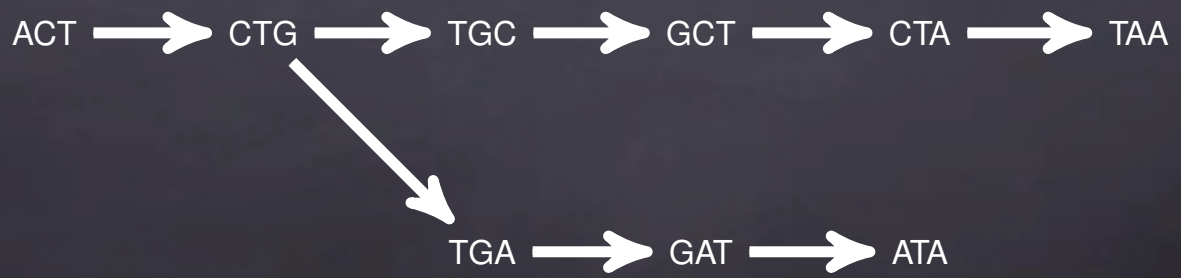
String graph / de Bruijn graph (2)

Let's add an error:

```
ACTGCT
 CTGATA
   GCTAA
```

String graph, $k = 3$:

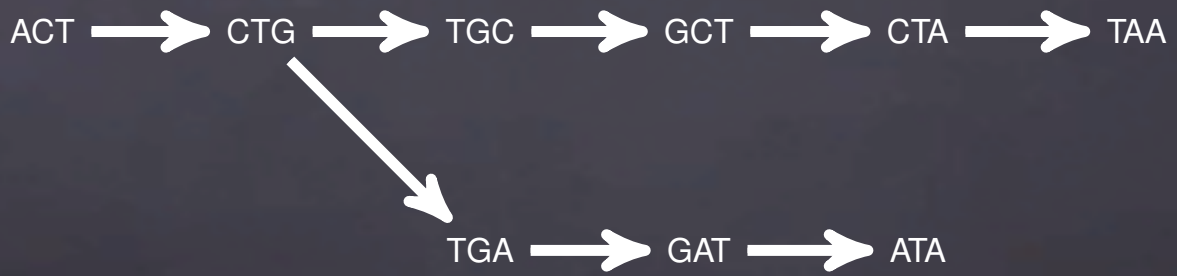
de Bruijn graph, $k = 3$:



String graph / de Bruijn graph (2)

How to "fix" the string graph?
$\rightarrow$ use a relaxed definition of overlaps.
String graph where overlaps may ignore 1 error, $k = 3$:

de Bruijn graph, $k = 3$:



String graph / de Bruijn graph (3)

So, which is better?

- String graphs capture whole read information
- de Bruijn graphs are conceptually simpler:
  - single node length
  - single overlap definition

Historically, string graphs were used for long reads and de Bruijn graphs for short reads.

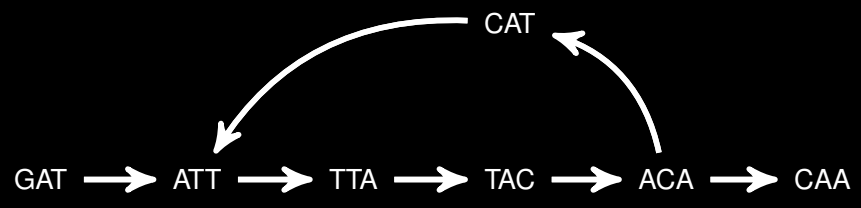
For raw PacBio data, there are too many indels to input them to a string graph. The solution is to input corrected data.

How does one assemble using a graph?

Assembly in theory [Nagarajan 09]

Return a path of *minimal length* that traverses **each node at least once**.

Illustration

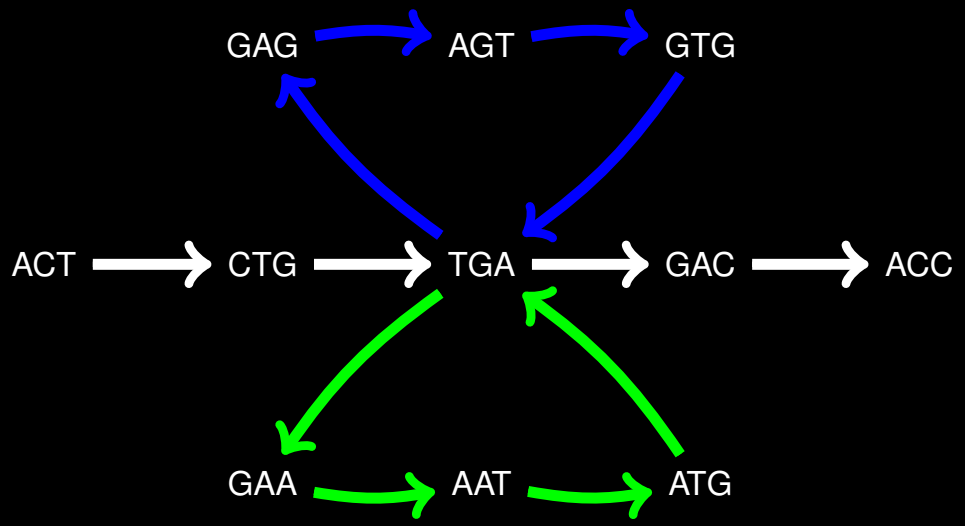


The only solution is GATTACATTACAA.

Assembly in practice

Because of ambiguities and low-coverage region, a single path is almost never found is theory, and is really never found in practice.

Example of ambiguities



Assembly in practice

Return a **set of paths** covering the graph, such that *all possible assemblies* contain these paths.

Solution of the example above

The assembly is the following set of paths:

$$\{\texttt{ACTGA}, \texttt{TGACC}, \texttt{TGAGTGA}, \texttt{TGAATGA}\}$$

Contigs construction

Contigs construction from a graph (de Bruijn graph or string graph, same methods).

The naive way is to enumerate all *node-disjoint* **simple paths**.

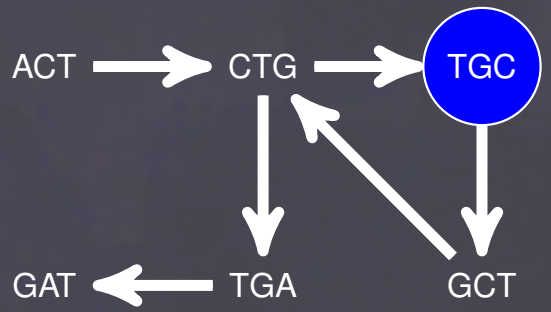
A *simple path* is a path where all internal nodes have one out-edge and one in-edge.

Node-disjoint means that two different paths cannot share a node.

(could also be done with *edge-disjoint* simple paths).

Contigs construction example

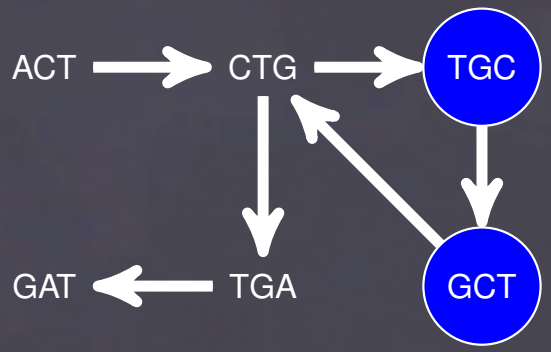
dBG, $k = 3$:



Contigs:

Contigs construction example

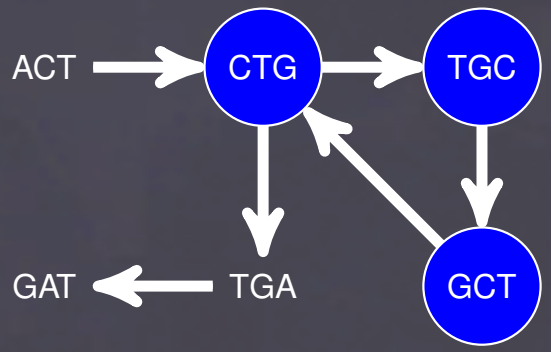
dBG, $k = 3$:



Contigs:

Contigs construction example

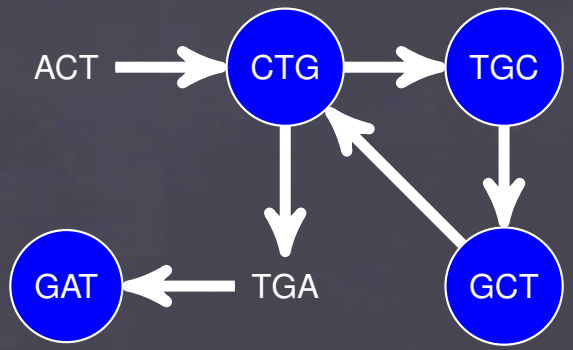
dBG, $k = 3$:



Contigs:
CTGCTG

Contigs construction example

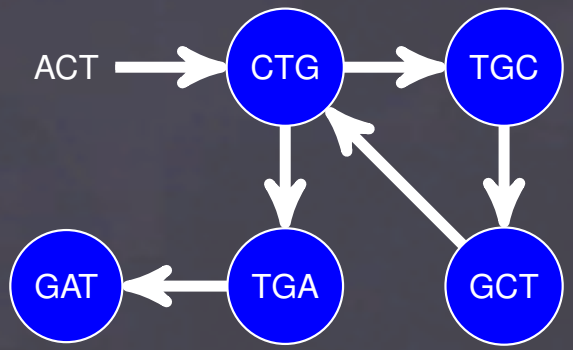
dBG, $k = 3$:



Contigs:
CTGCTG

Contigs construction example

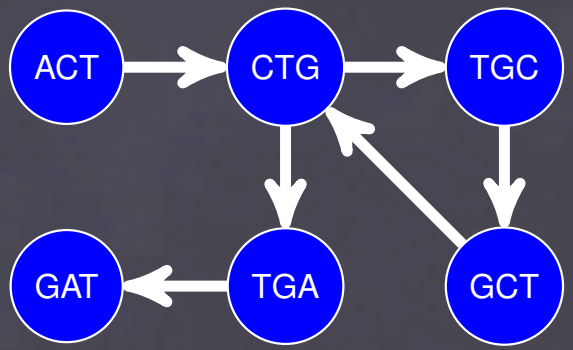
dBG, $k = 3$:



Contigs:
CTGCTG
TGAT

Contigs construction example

dBG, $k = 3$:



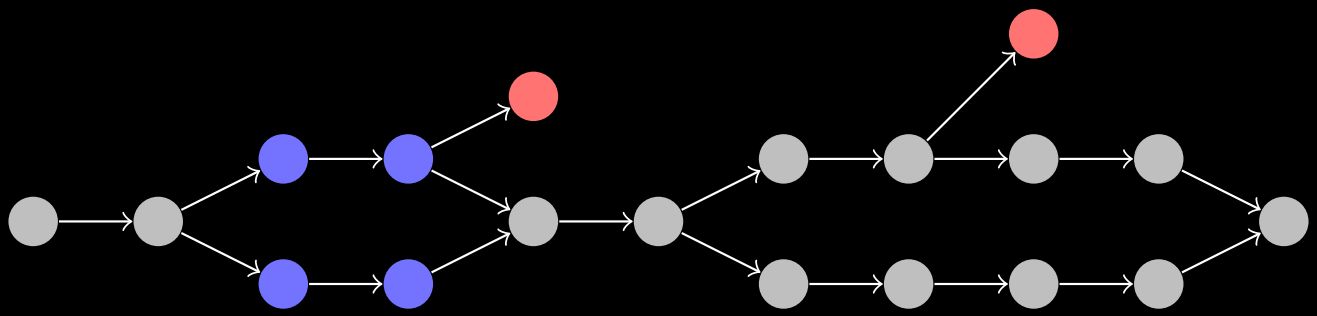
Contigs:
CTGCTG
TGAT
ACT

How an assembler works

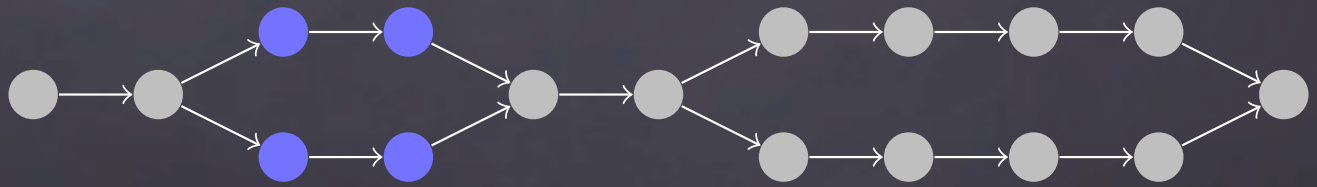
[HGAP, SPAdes, Velvet, ABySS, SOAPdenovo, SGA ..]

1) Maybe correct the reads. (SPAdes, HGAP, SGA)
2) Construct a graph from the reads.

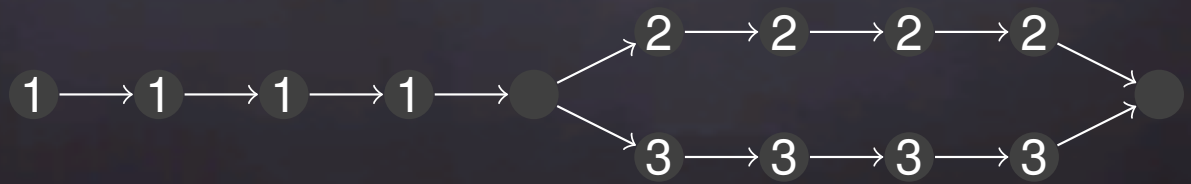
Assembly graph with variants & errors



3) Likely sequencing errors are removed.



3) Known biological events are removed.
4) Finally, **simple paths** (i.e. contigs) are returned.



Short note on reverse complements

Because sequencing isn't strand-directed:

> *In assembly, we always identify a read with its reverse complement.*

E.g: AAA = TTT, ATG = CAT

Exercise

In this exercise, for simplicity, ignore reverse complements.
Reads:

```
TACAGT
  CAGTC
   AGTCAG
     TCAGA
```

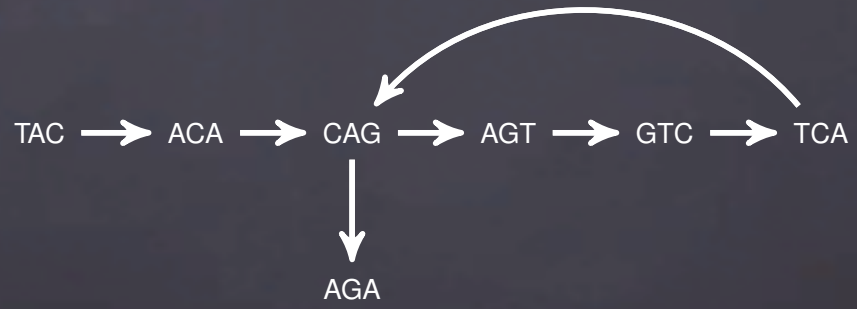
1. Construct the de Bruijn graph for $k = 3$.
   (Reminder: nodes are k-mers and edges correspond to $(k+1)$-mers)
2. How many contigs can be created? (stopping at any branching)
3. At which value of k is there a single contig?
4. (optional) Find a mathematical relationship between k_a, the smallest value of k for which a genome can be assembled into a single contig, and ℓ_r, the length of the longest exactly repeated substring in that genome.

Exercise (solution)

In this exercise, for simplicity, ignore reverse complements.
Reads:

```
TACAGT
  CAGTC
   AGTCAG
     TCAGA
```

1. Construct the de Bruijn graph for $k = 3$.
   The 3-mers (nodes) are: TAC, ACA, CAG, AGT, GTC, TCA, AGA
   The 4-mers (edges) are: TACA, ACAG, CAGT, AGTC, GTCA, TCAG, CAGA



2. How many contigs can be created? (stopping at any branching) 3
3. At which value of k is there a single contig? 4
4. Find a mathematical relationship between k_a, the smallest value of k for which a genome can be assembled into a single contig, and ℓ_r, the length of the longest exactly repeated substring in that genome.
   $k_a = \ell_r + 1$

Plan

Metrics

Preamble: There is no trivial total order (i.e. ranking) between assemblies.

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
```

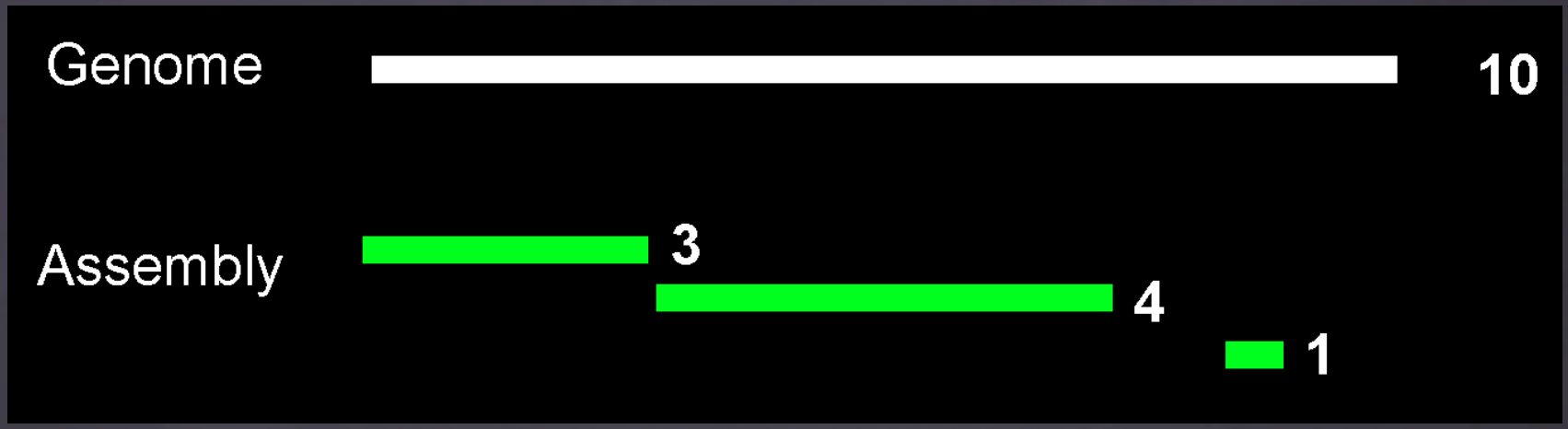
Recent assembly metrics are mostly based on:

- internal consistency
- likelihood of an assembly given the reads

Reference-free metrics: N50

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



If you didn't know N50, write down the definition, there will be an exercise !

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, its length is the N50 value.
- Else, consider the two largest contigs, do they cover 50%?
- If yes, then the N50 is the length of the second largest contig.
- And so on..

Internal consistency

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

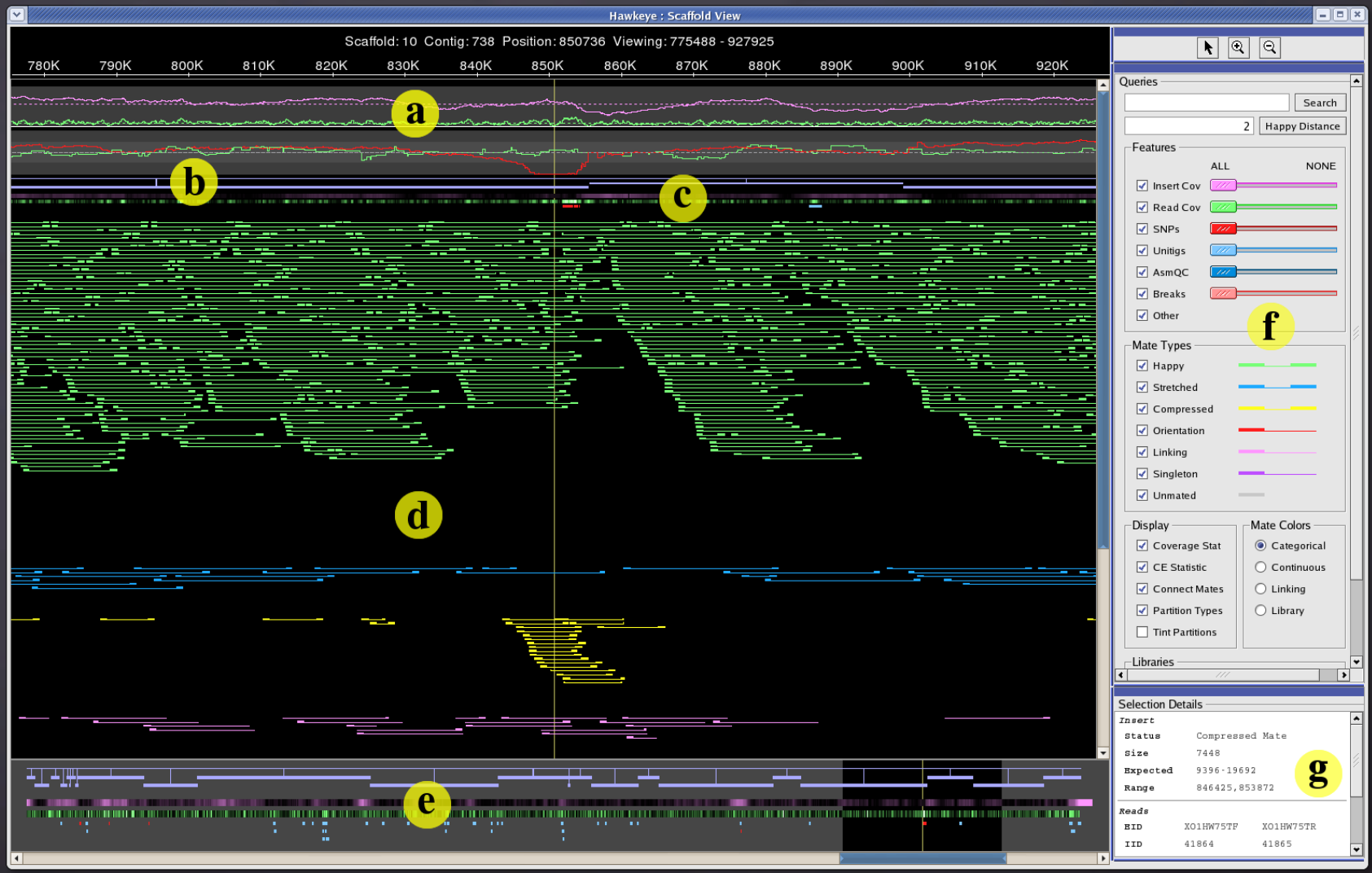
Internal consistency : Percentage of paired reads correctly aligned back to the assembly (*happy* pairs).

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

- REAPR[3] [M Hunt, .. (Gen. Biol.) 2013]
- FRCurve[4] [F. Vezzi, .. (Plos One) 2013]

[3] Google: REAPR assembly
[4] Google: FRCurve

INTERNAL CONSISTENCY: EXAMPLE



Hawkeye software

Assembly likelihood (1)

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

$$\mathcal{L}(A|R) = P(R|A) = \prod_i P(r_i|A)$$

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	-23.44×10^8	22096466	82.22	2.1	2
ALLPATHS-LG	-22.77×10^8	23122569	97.24	81647	36.5
CABOG	-21.26×10^8	23433424	98.32	393	45.3
SOAPdenovo	[a]	[a]	98.17	455	14.7
Reference	-19.04×10^8	23978017	-	-	-

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

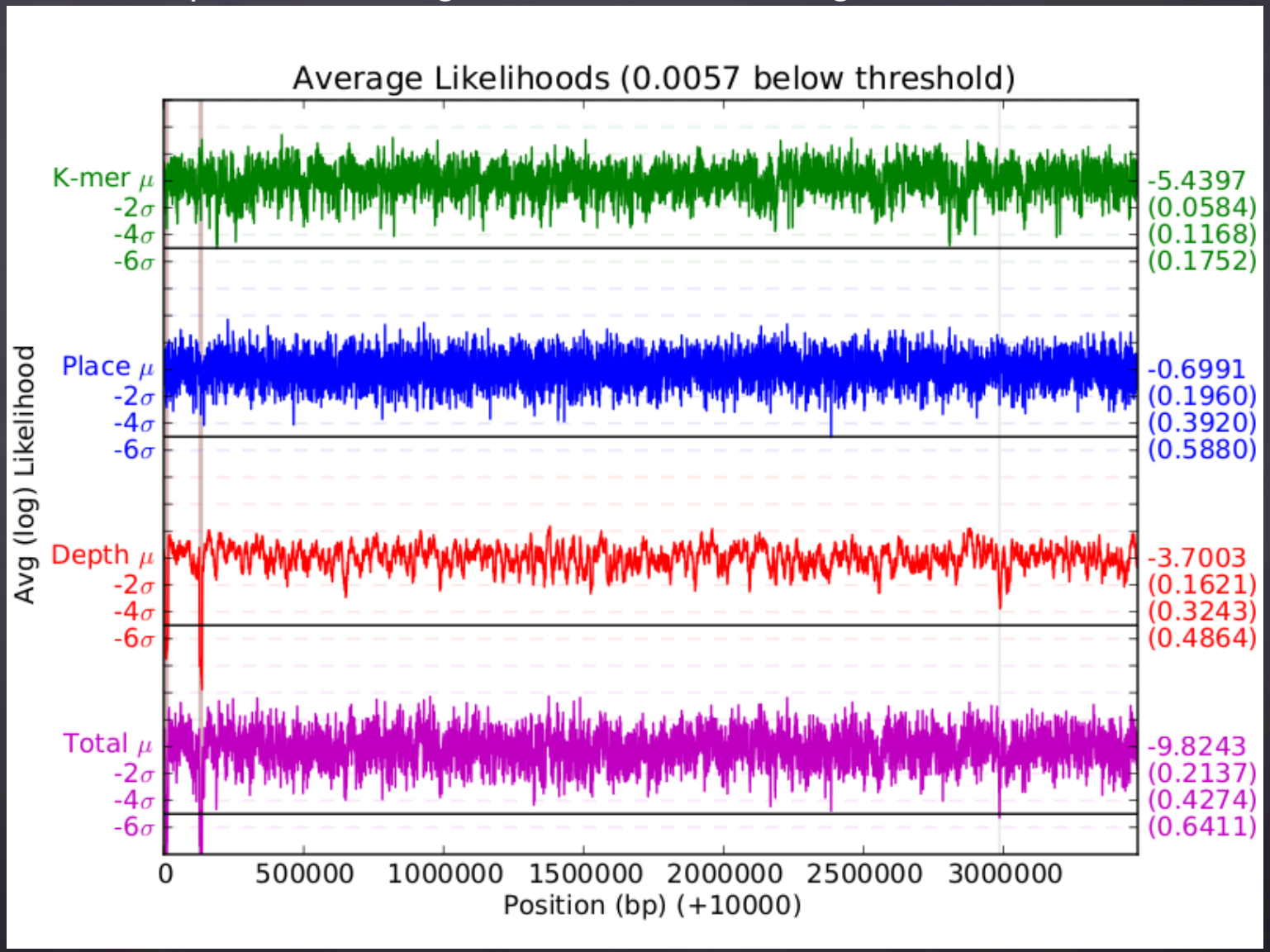
Rahman and Pachter *Genome Biology* 2013 **14**:R8 doi:10.1186/gb-2013-14-1-r8

(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:

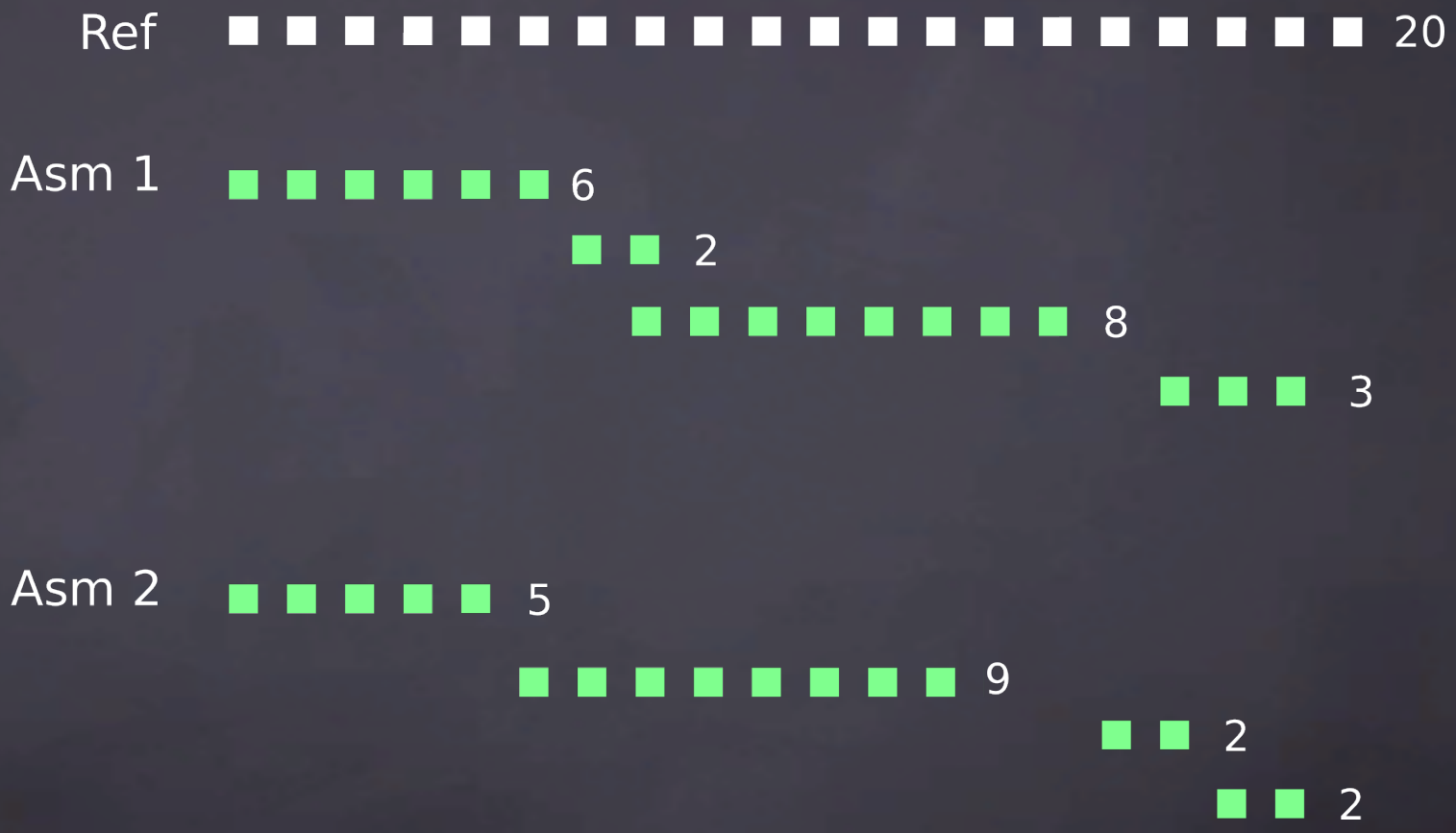
- No total order for metrics
- Use QUAST
- Use CEGMA
- Try ALE

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

EXERCISE

At some point in life, one may need to compare assemblies.

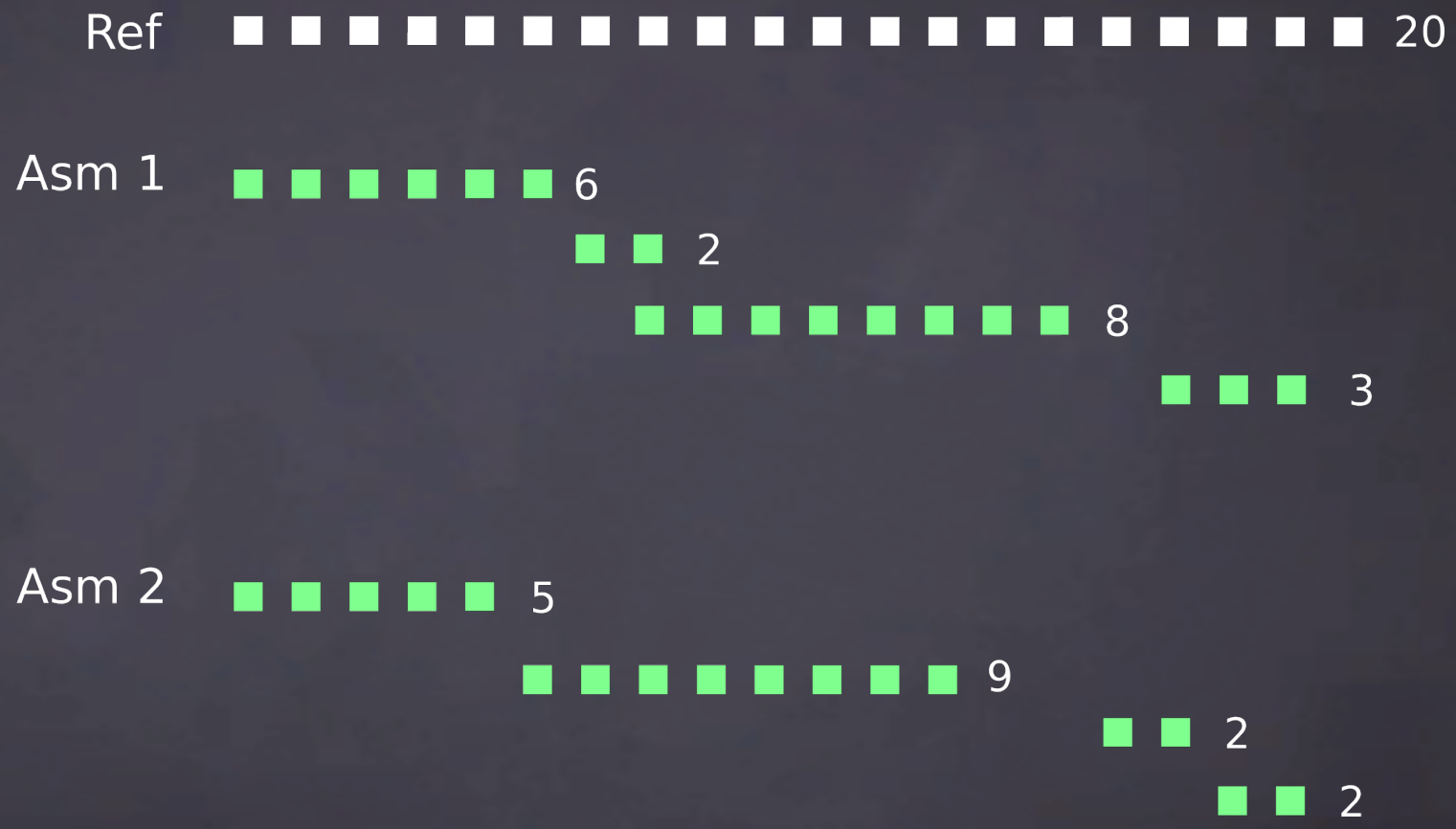
Here are two assemblies, aligned to the same reference:



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

Exercise (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: higher NG50, less contigs, same coverage as the other. But: has some redundancy (maybe a highly heterozygous locus))

Plan

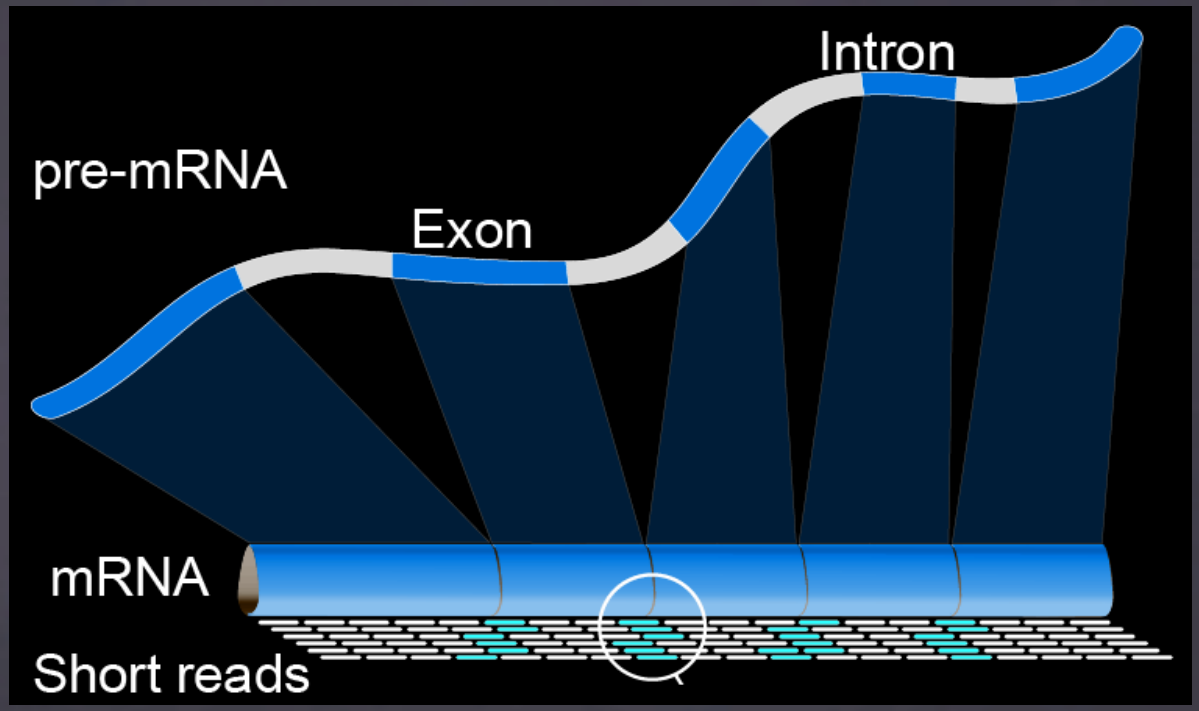
Landscape of assemblers

- Before the Illumina Hi-Seq: 454 (Newbler), Illumina reads $<$ 100 bp (any de Bruijn graph assembler).
- newer Illumina: **200-500** bp reads (when merged), **high** coverage, mate pairs: grey area for assembly software.
- PacBio: $>$ 2 kbp reads, **low** coverage: Gaining momentum for DNA-seq. **Do not use** a de Bruijn graph assembler. Use any string graph assembler (with pre-assembly error-correction).

Personal experience (for Illumina assembly)

Your data follows the Broad recipe	Allpaths-LG
Small (meta)genome	SPAdes
To get a second opinion	SOAPdenovo2
If not enough memory	Minia
454	Newbler
PacBio	HGAP
RNA-Seq	Trinity
Large metagenome	Megahit

RNA-Seq and assembly



Goal: reconstruct mRNA sequences

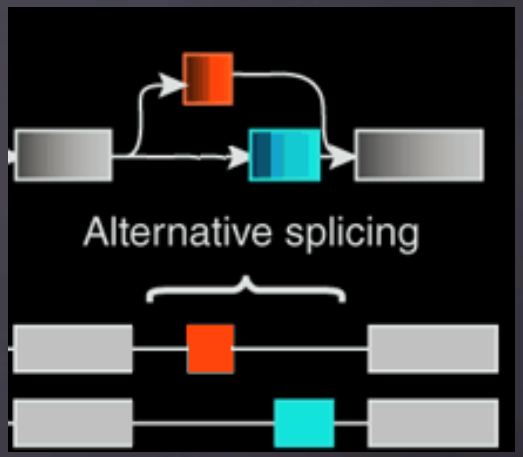
RNA-Seq Assembly

- Short contigs
- Uneven coverage
- Contigs are re-used

average mRNA length: 2 kbp

varying expression levels

alternative splicing



RNA-Seq Assembly

Despite these differences, DNA-seq assembly methods apply:

- Construct a de Bruijn graph (same as DNA)
- Output contigs (same as DNA)
- Allow to re-use the same contig in many different transcripts (new part)

RNA-Seq Assembly: Trinity

Quick overview of Trinity steps:

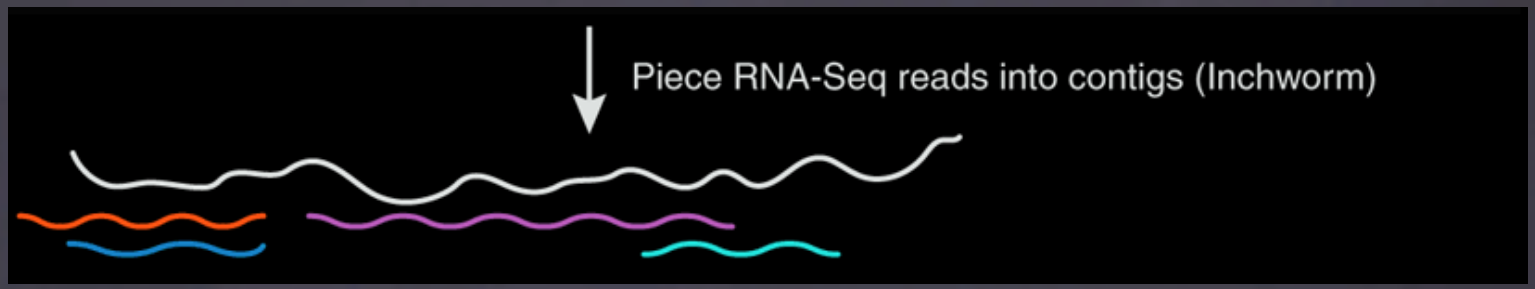
- Inchworm
- Chrysalis
- Butterfly

RNA-Seq Assembly: Trinity

- ~~Inchworm~~ de Bruijn graph construction, part 1
- ~~Chrysalis~~ de Bruijn graph construction, part 2, then partitioning
- ~~Butterfly~~ Graph traversal using reads, isoforms enumeration

RNA-Seq Assembly: Trinity - 1

- **Inchworm** - de Bruijn graph construction, part 1

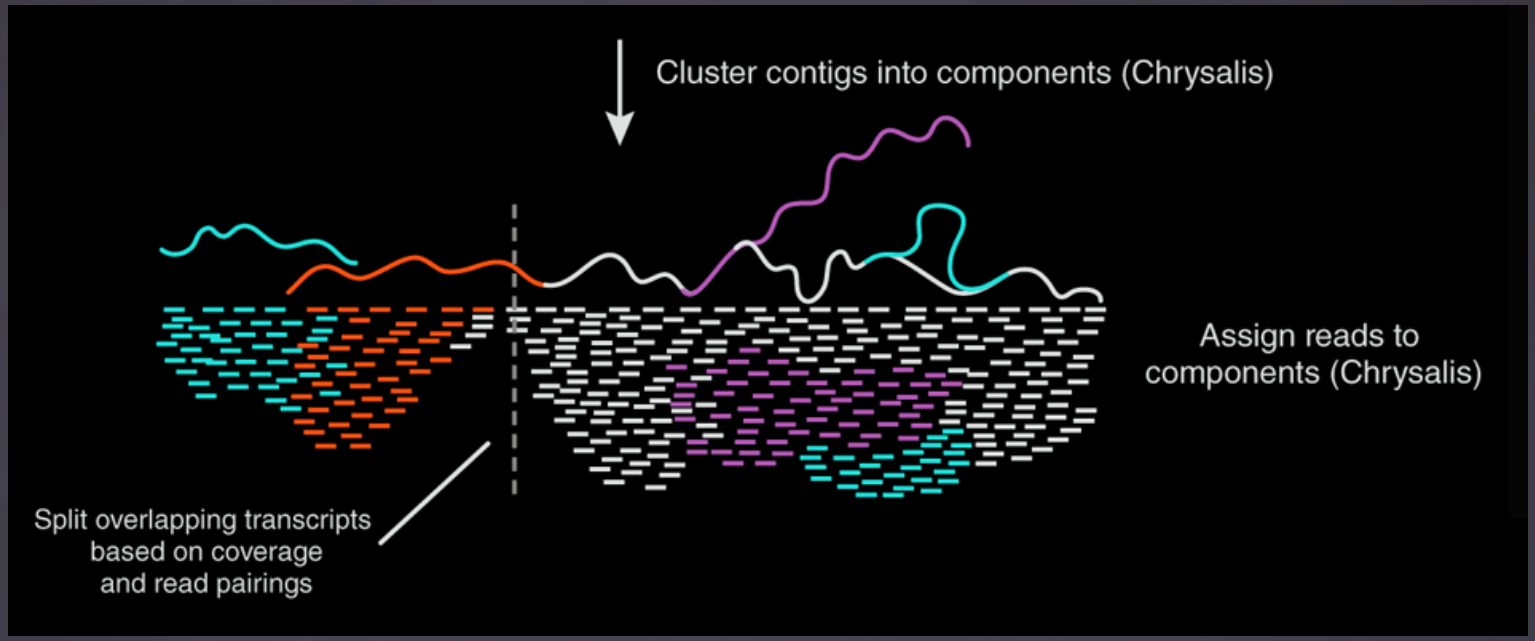


Using k-mers, construct contigs carelessly.

Contigs might correspond to the most abundant isoform, but no guarantee.

RNA-Seq Assembly: Trinity - 2

- **Chrysalis** - de Bruijn graph construction, part 2, then reads partitioning



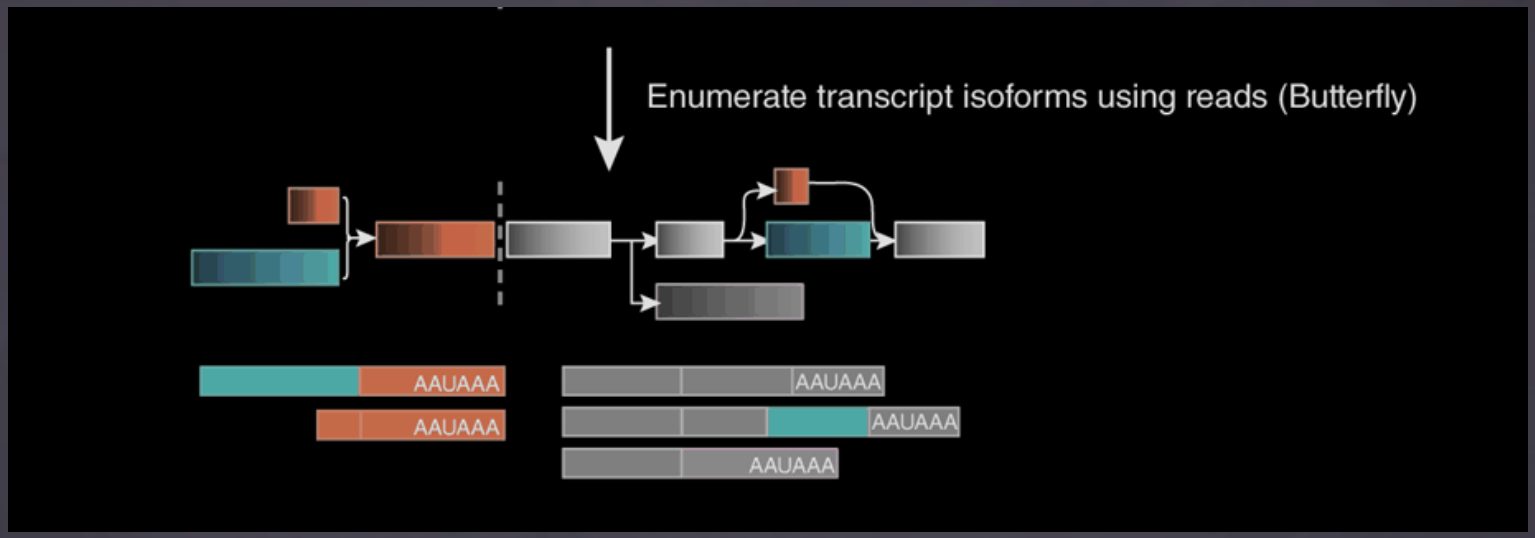
By overlapping Inchworm contigs, construct the true de Bruijn graph.

Then,

Partition the graph and output the reads aligning to each partition.

RNA-Seq Assembly: Trinity - 3

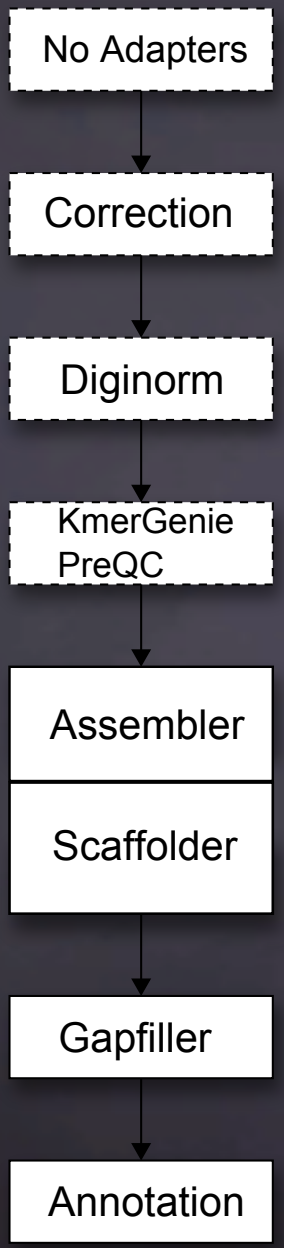
- **Butterfly** - Graph traversal using reads, isoforms enumeration



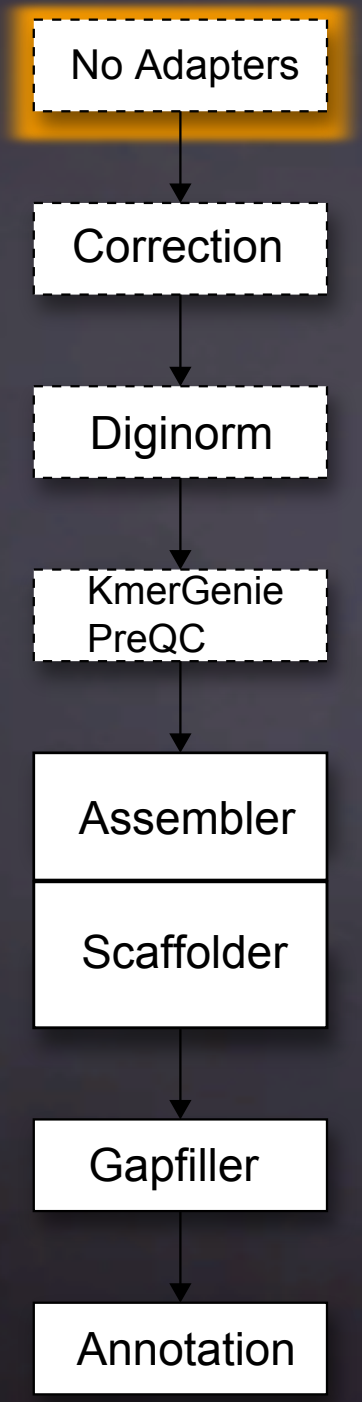
Traverse each de Bruijn graph partition to output isoforms

Difference with DNA-seq assembly: isoforms are, by definition, not *k*-mer-disjoint.

ASSEMBLY PIPELINES



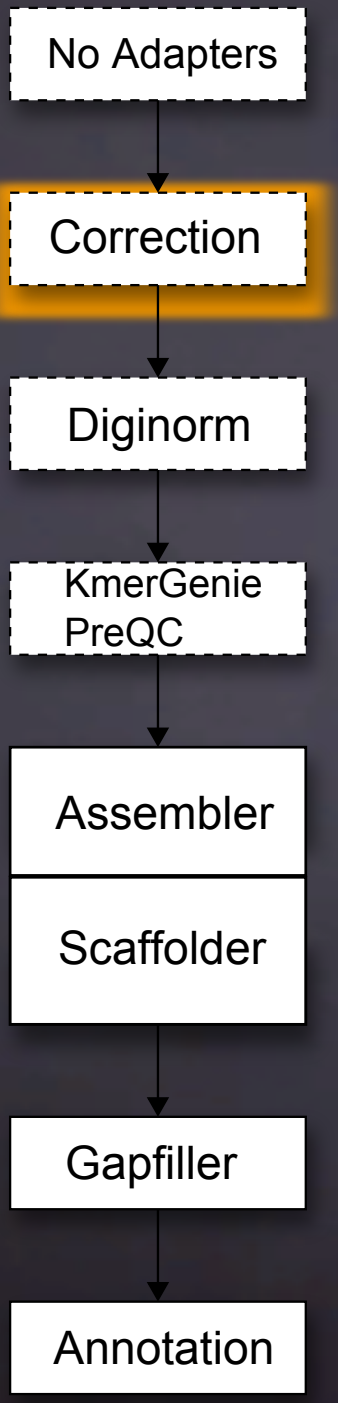
INITIAL STEPS



A good assembly is typically done with several pre-correction stages:

- low-quality reads removal
- trimming
- overlapping paired reads merged into single reads

Error correction

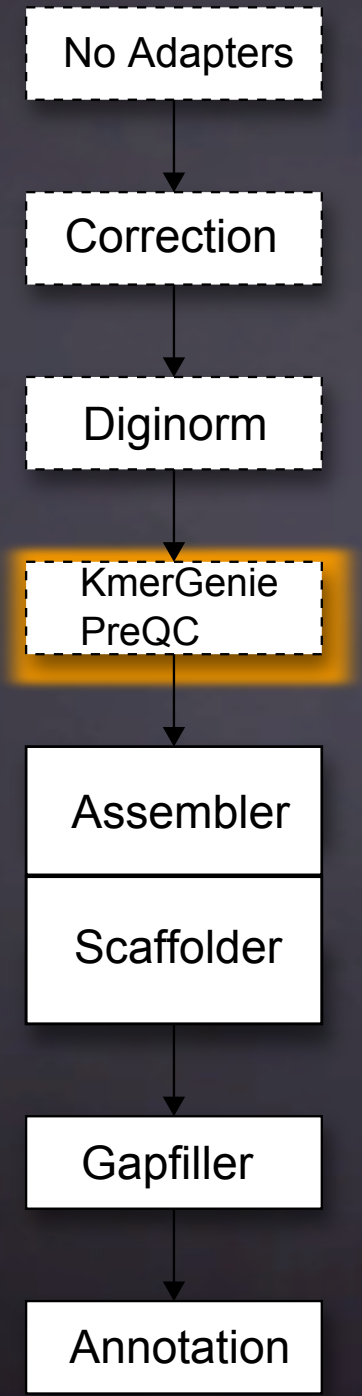


Error-correction generally improves Illumina assemblies.
It is mandatory (and automatic) for PacBio data.
For Illumina:

- SPAdes does it automatically
- For larger genomes, Allpaths-LG stand-alone error corrector (highly recommended)
- SOAPdenovo stand-alone corrector
- Quake

In my experience, with high-coverage data, finding good parameters for the assembler achieves similar effects as error correction.

The k parameter (Illumina)

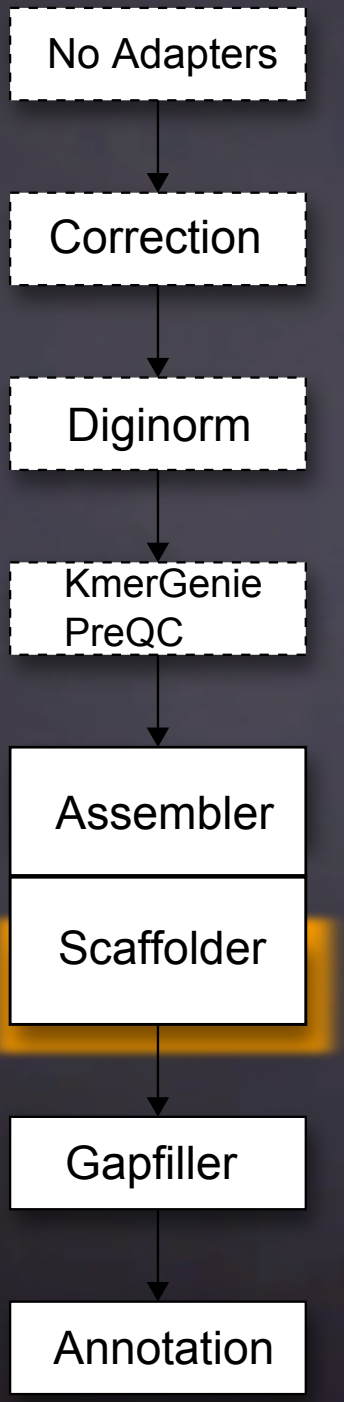


The optimal k-mer size varies with each dataset.

A few things to keep in mind:

- Low limit: For common genomes sizes (10 Mbp - 1 Gbp), there is a high chance that any $\approx$ 12-mer will be repeated in many locations ($4^{12} = 16 \cdot 10^6$).
- High limit: Read length
- Ideally, you want to set k as high as possible, such that, in the reads, non-erroneous k-mers are present significantly more than erroneous k-mers.
- Practically try at least two k values (e.g. $31, 61$).
- My tool KmerGenie can help choose k. Jared Simpson's PreQC does pre-assembly quality control.

Scaffolders



(Not for RNA-Seq)

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

Most assemblers include a scaffolder (SOAPdenovo2, SGA, ABySS, Velvet, Newbler..).

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

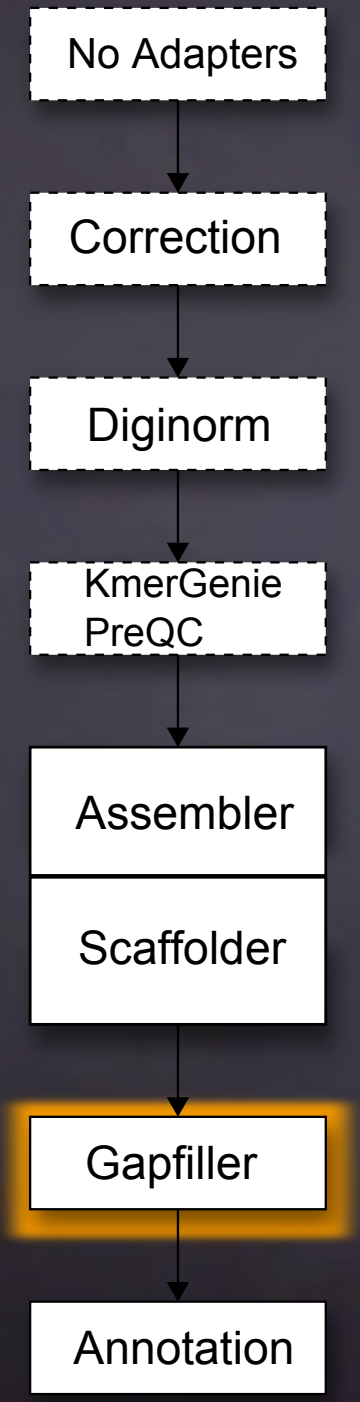
For better assemblies, you may try to:

- Use another assembler's scaffolder (SOAPdenovo2's)
- Use a stand-alone scaffolders (e.g. SSPACE, BESST, Bambus 2, Opera, etc..)
- Simply skip scaffolding, sometimes contigs are good enough.

SSPACE is easy to use:

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

GapFillers



(Not for RNA-Seq)

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 (SOAPdenovo2, Allpaths-LG).

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

GapCloser is quite easy to use:

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

To create a draft genome from short reads

My recommendations:

1. Sequencing strategies:
   - Broad recipe (many Illumina libraries)
   - PacBio high coverage
2. Read either the GAGE paper, GAGE-B (Illumina, bacteria), Assemblathon 2 (large genomes), HGAP paper (PacBio, bacteria), or Twitter/blogs (PacBio assembly)
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

Last Exercise

Reads:

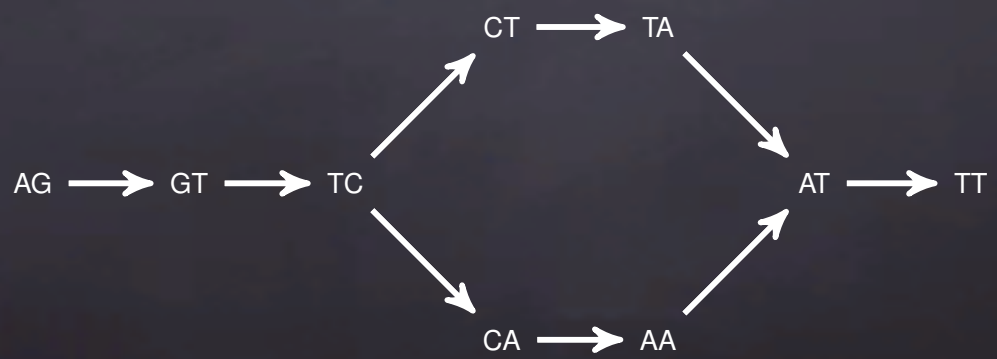
1. AGTC
2. TCAA
3. AATT
4. GTCT
5. TATT
6. TCTA
7. TCAA
8. TCTA

1. Assemble these reads
2. What was special about this genome?

Last Exercise (detailed solution)

Step by step:

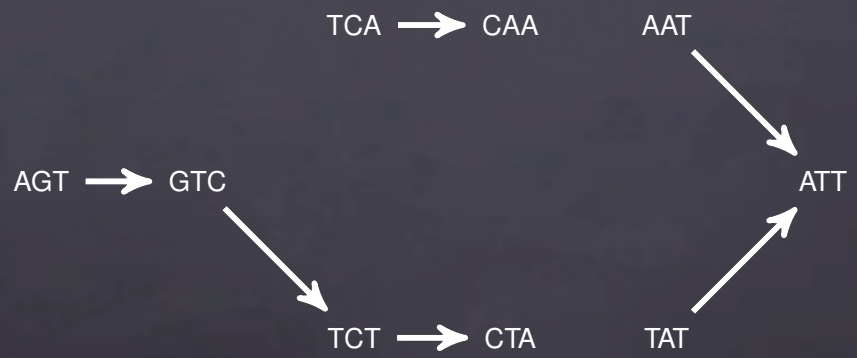
- **Choose an assembly model**: de Bruijn graph or string graph
- *The reads are short, let's choose the de Bruijn model*
- **Choose a k-mer size**:
- *Tempting to use $k = 3$, as it is the highest value such that $(k+1)$-mers exist in the reads. However, to obtain a good assembly, all 4-mers from the (unknown) sequenced genome need to be seen in the reads. Take for instance the 3-mer CAA, there is no 4-mer starting with it, so one could guess that coverage is insufficient. Hence, let's pick a smaller k, $k = 2$.*
- The **nodes** of the graph are all the distinct 2-mers in the reads: AG, GT, TC, CT, TA, CA, AA, AT, TT, GT
- The **edges** of the graph are all the distinct 3-mers in the reads: AGT, GTC, TCA, CAA, AAT, ATT, TCT, TAT, CTA
- Those last two pieces of information are sufficient to draw the graph:



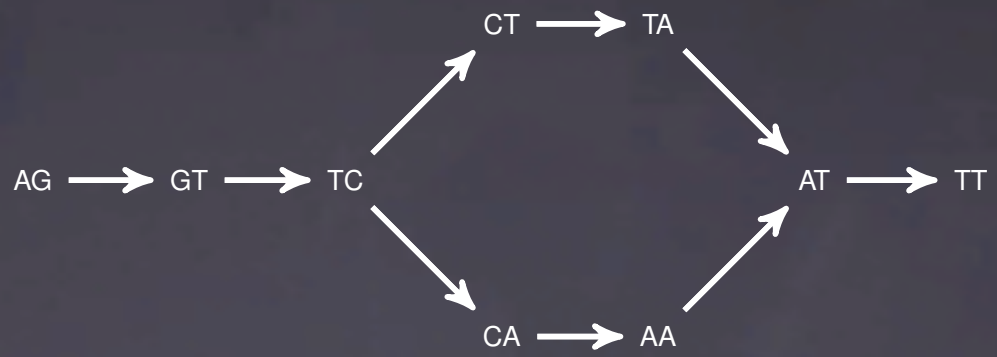
Last Exercise (detailed solution)

Just out of curiosity, let's draw the de Bruijn graph for $k = 3$:

- The **nodes** of the graph are all the distinct 3-mers: AGT, GTC, TCA, CAA, AAT, ATT, TCT, TAT, CTA
- The **edges** of the graph are all the distinct 4-mers (here, the distinct reads): AGTC. TCAA, AATT, GTCT, TATT, TCTA
- We have less edges than nodes, so clearly the read coverage is not sufficient for the graph to be connected.



Last Exercise (detailed solution)



- *To assemble this graph, using the contigs construction used before, there would be 4 contigs. Depending on where one includes branching k-mers (TC, AT) in contigs, a possible solution is: AGTC, CTA, CAA, ATT.*
- But we can actually do better. There are two ways to traverse this graph, yielding an assembly of two haplotypes:

  `AGTCAATT`

  `AGTCTATT`
- This could be a tiny diploid genome with an heterozygous SNP. The bubble is unlikely to be a sequencing error, as I have purposely added reads 7 and 8, to have larger coverage in both paths of the bubble. If it was a sequencing error, one of the paths would typically have low coverage.
- An assembler would collapse this bubble and output only one of the two haplotypes.

Conclusion, what we have seen

- What is a good assembly?
  - No total order
  - Main metrics: N50, coverage, accuracy
  - Use QUAST
- How are assemblies made?
  - Typically, using a de Bruijn graph (Illumina) or a string graph (PacBio)
  - Errors and small variants are removed from the graph.
  - Contigs are just simple paths from the graph.
  - Scaffolds are linked contigs, misassemlies often happen there.
- Assembly software
  - Recommended for Illumina: SPAdes (small genomes), Allpaths-LG
  - Recommended for PacBio: HGAP (small genomes)
  - Many tools for custom needs: Minia for low-memory, SGA for very accurate assembly, etc..
- A few tips
  - Try a different assembler
  - Try a few k values
  - An assembly is not the **absolute truth**, it is a mostly complete, generally fragmented and mostly accurate **hypothesis**