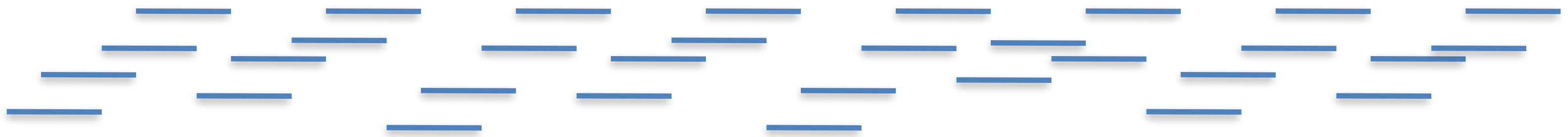
Whole genome shotgun sequencing
DeNovo Assembly

Strategies to sequence long DNA molecules: Shotgun Sequencing

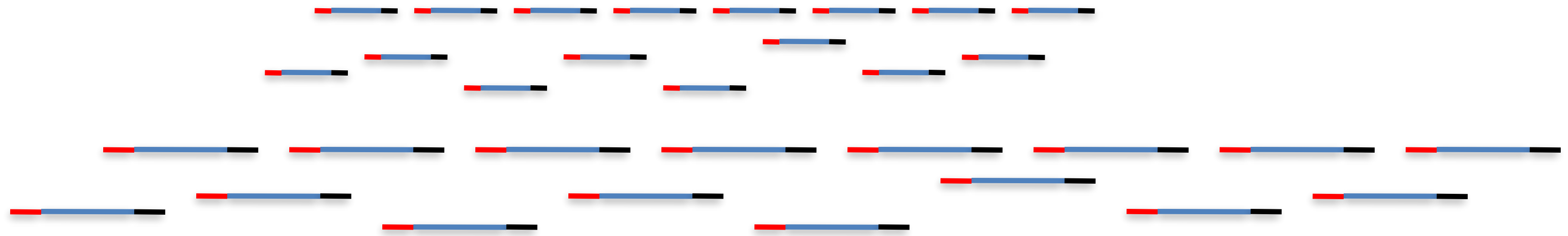
1. Randomly break template DNA into pieces

Strategies to sequence long DNA molecules: Shotgun Sequencing



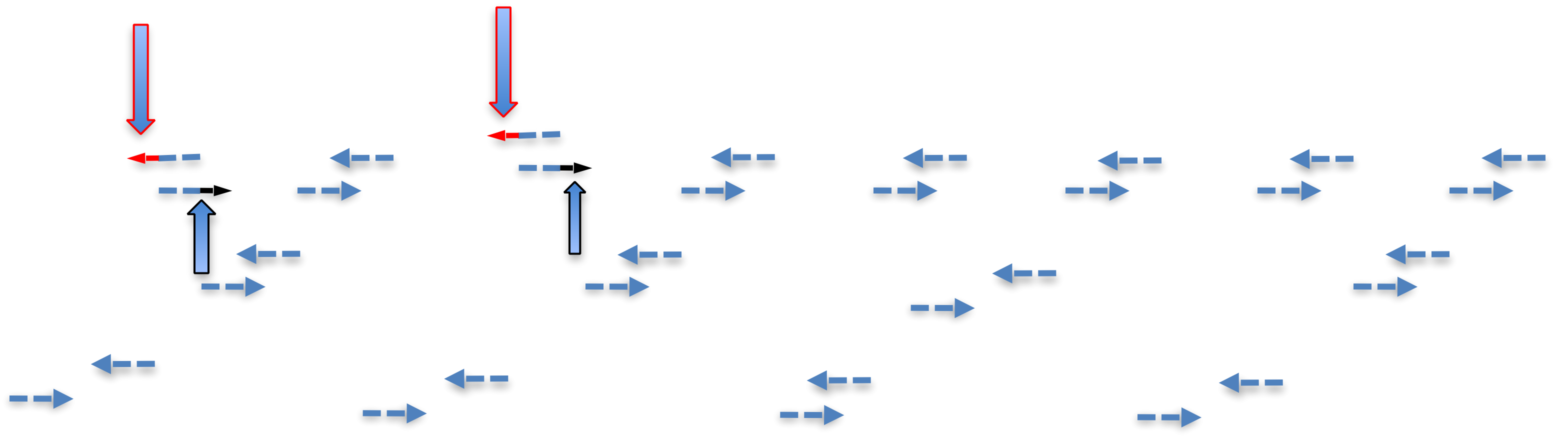
1. Randomly break template DNA into pieces

Strategies to sequence long DNA molecules: Shotgun Sequencing



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends and size select

Strategies to sequence long DNA molecules: Shotgun Sequencing
Sometimes adapter sequences remain!

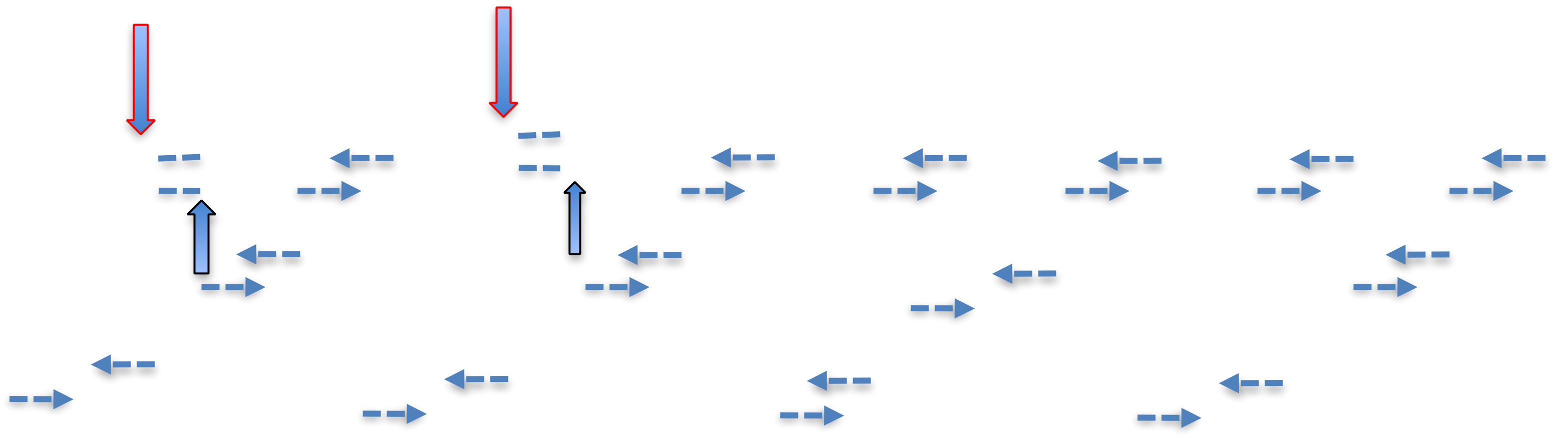


1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments

Identifying these sequences is simple when we ignore the complexity of the search

The problem is, what sequence(s) are we looking for?

Strategies to sequence long DNA molecules: Shotgun Sequencing
Sometimes adapter sequences remain!

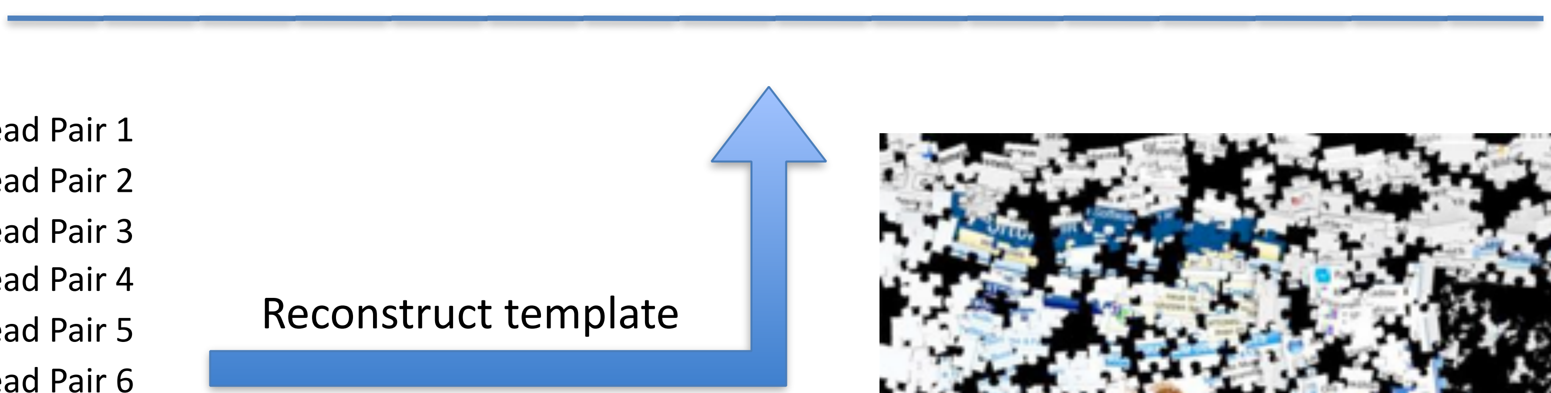


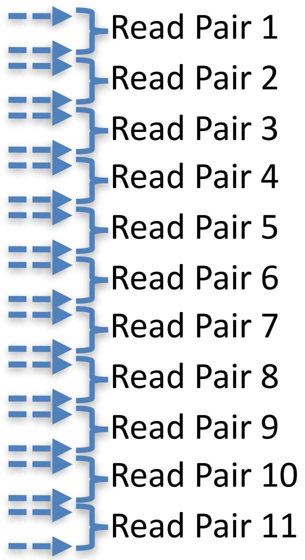
1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments

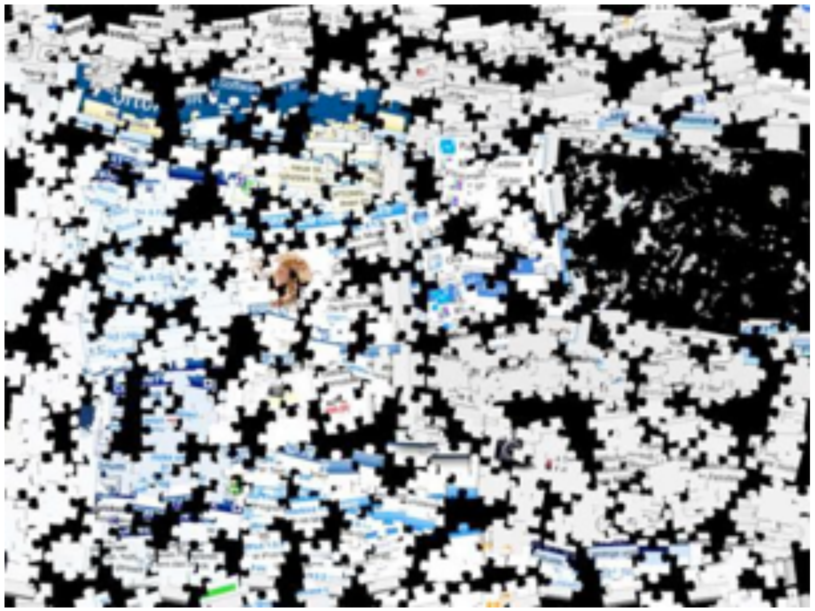
Identifying these sequences is simple when we ignore the complexity of the search

Clip adapter sequences

Strategies to sequence long DNA molecules: Shotgun Sequencing

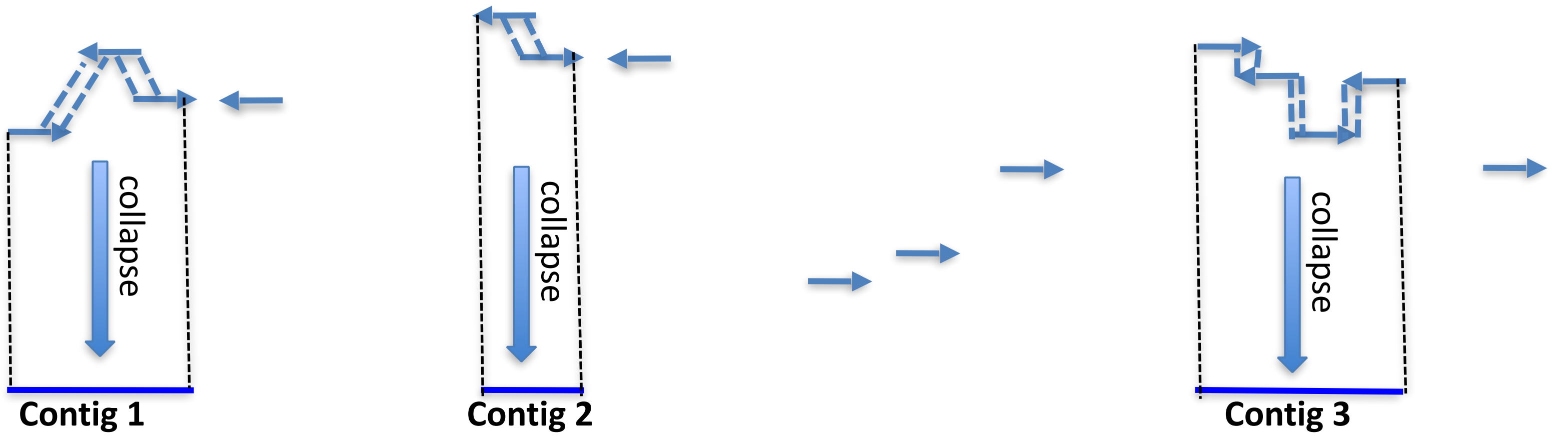






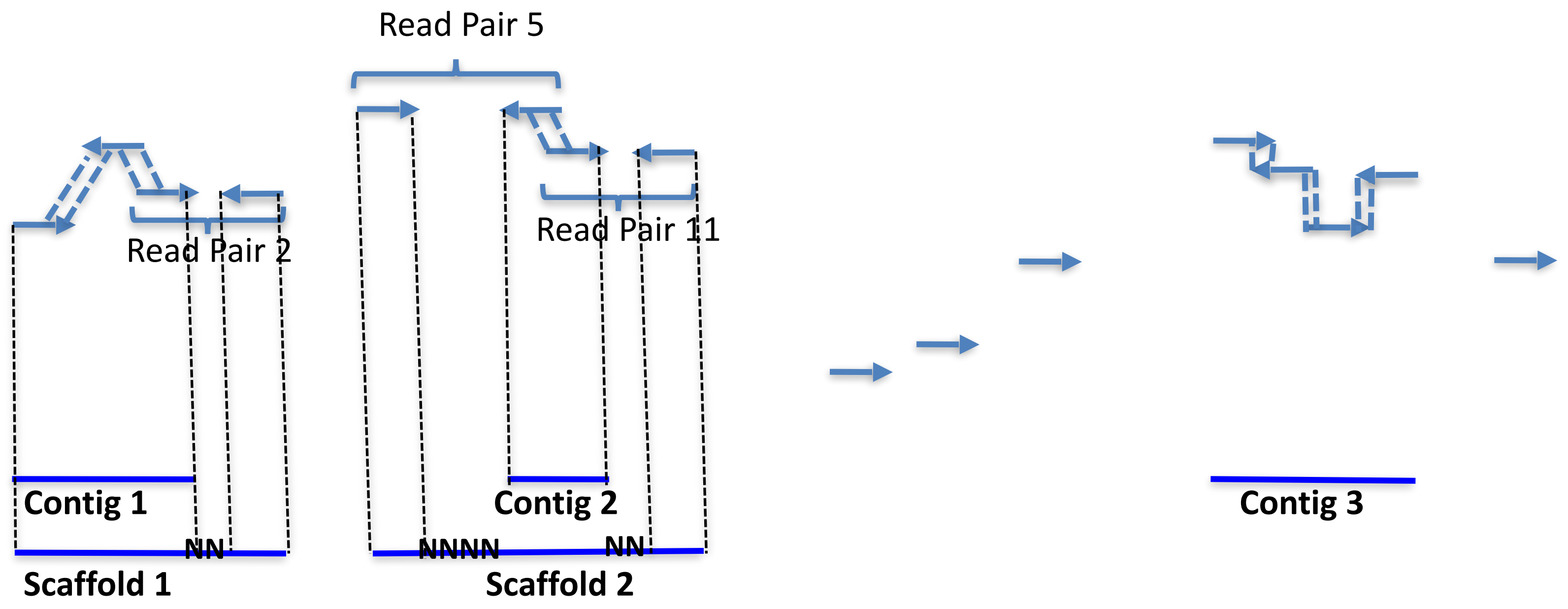
1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments
4. Identify and remove adapter part from the determined sequences
5. Reconstruct template sequence from the sequence reads

Strategies to sequence long DNA molecules: Shotgun sequencing and **de-novo assembly** of the sequence reads



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments
4. Remove adapter part from the determined sequences
5. Reconstruct template sequence from the sequence reads
    1. Reference guided sequence assembly: map reads to a reference sequence
    2. De-novo sequence assembly: determine overlap between sequence reads and assemble overlapping sequences into **contigs.**

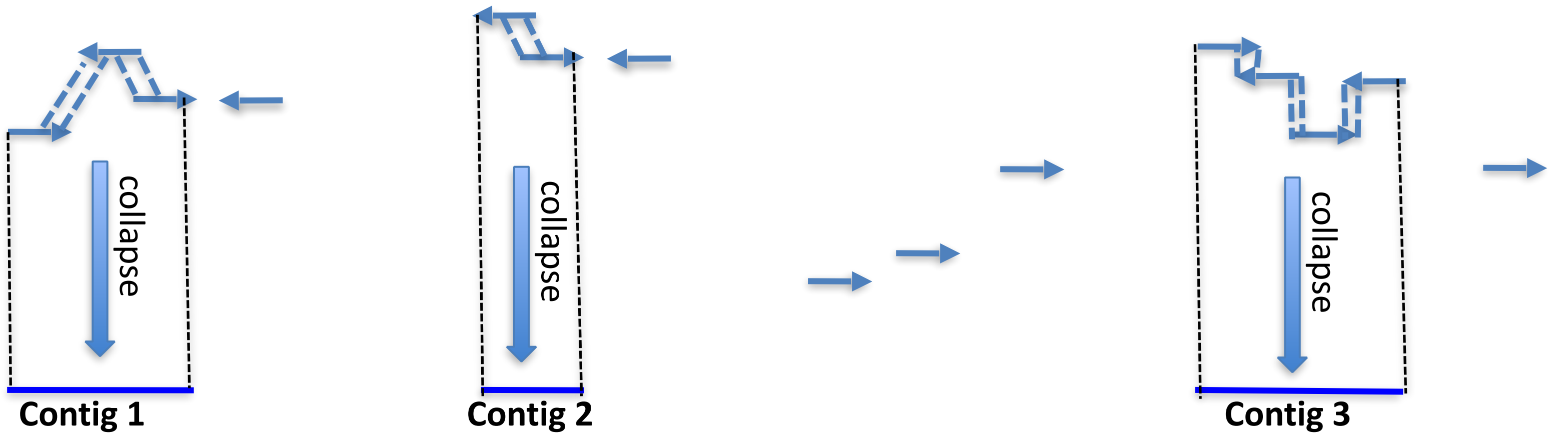
Strategies to sequence long DNA molecules: Shotgun sequencing and **de-novo assembly** of the sequence reads



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments
4. Remove adapter part from the determined sequences
5. Reconstruct template sequence from the sequence reads
    1. Reference guided sequence assembly: map reads to a reference sequence
    2. De-novo sequence assembly: determine overlap between sequence reads and assemble overlapping sequences into contigs**.** **Mate pair information** can then be used to build **super-contigs** (**scaffolds**) from physically non-overlapping contigs**.**

Strategies to sequence long DNA molecules: Shotgun sequencing and **de-novo assembly** of the sequence reads

Some summary statistics to describe assemblies



1. Coverage: The average number of reads covering a position in the sequenced template DNA.

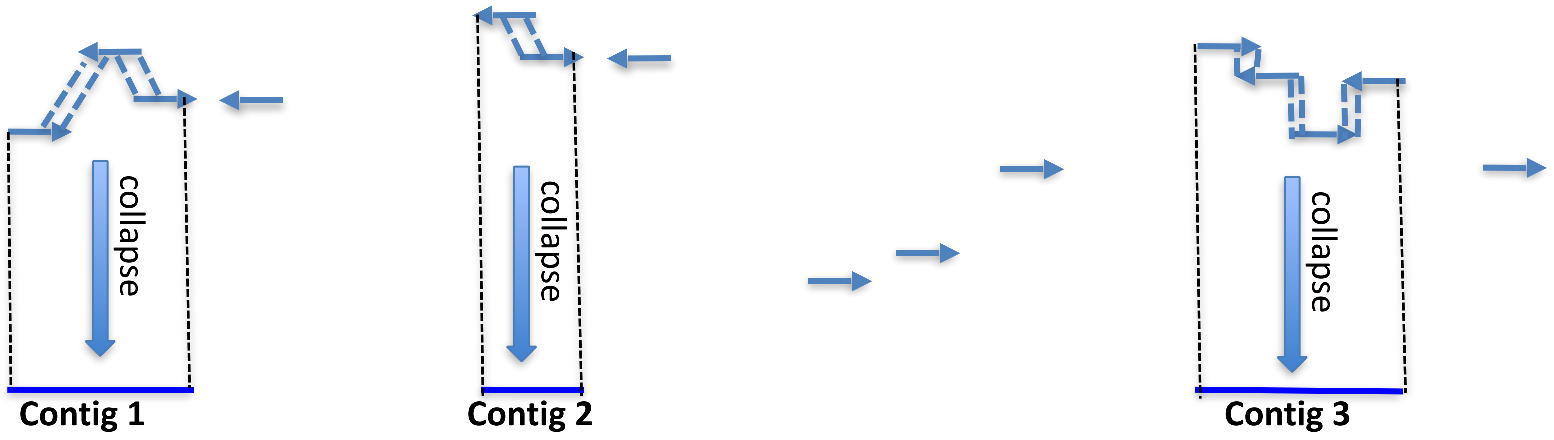
Length of genomic segment:	L		
Number of reads:	n	Coverage	C = n l / L
Length of each read:	l		

How much coverage is enough?

The higher the coverage the better (provided unlimited computational resources)!
The more uniform the coverage distribution the better!

Strategies to sequence long DNA molecules: Shotgun sequencing and **de-novo assembly** of the sequence reads

Some summary statistics to describe assemblies

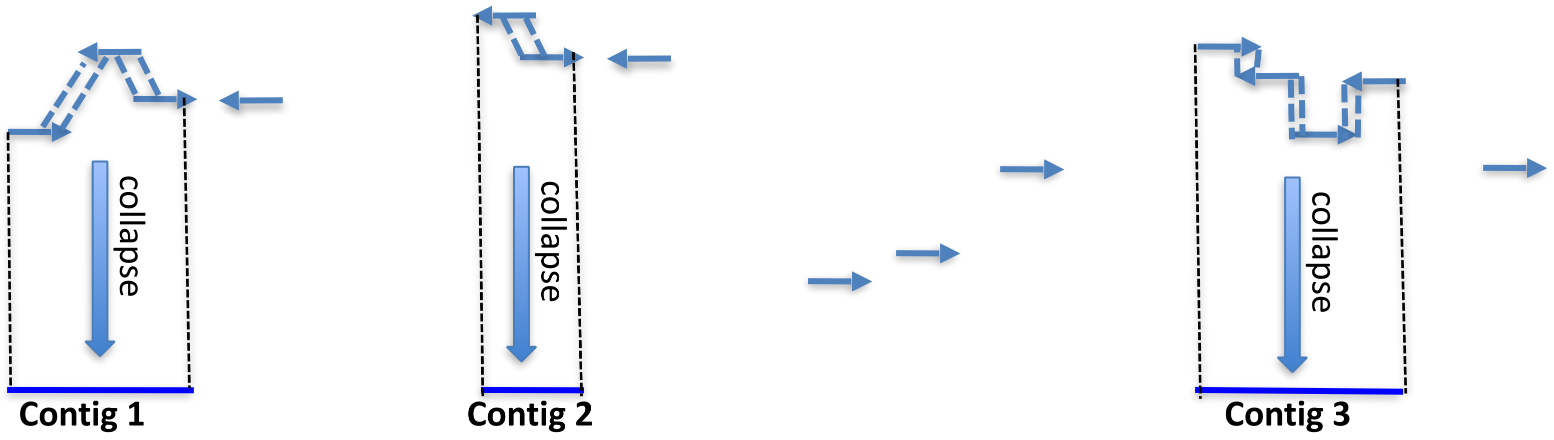


2. **N50-size:** More than 50% of the bases in your assembly reside in contigs with **at least** the size determined by the N50 value. **NOTE:** You can of course specify any other *N-value*.

What now tells us the N50 size exactly?
Is it a quality measure as people frequently use it?
When does it make sense to mention the N50 size (just consider RNAseq assemblies)?

Strategies to sequence long DNA molecules: Shotgun sequencing and **de-novo assembly** of the sequence reads

Some summary statistics to describe assemblies



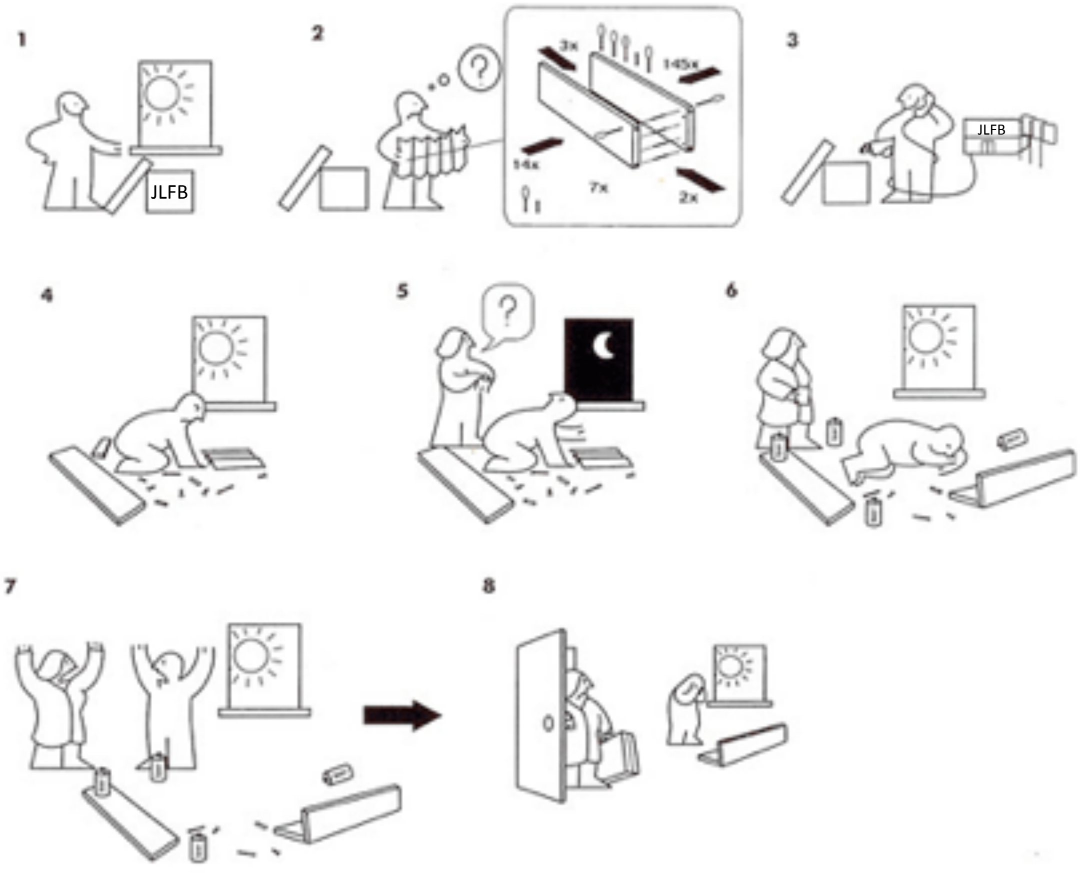
3. **Contig length distribution**

Note: All these statistics can be used for scaffolds as well!

Literature on de-novo assemblies

- J.R. Miller et al. Genomics 95 (2010) 315–327
- P. Compeau et al. Nature Biotechnology 29 (2011) 987-991
- Zerbino and Birney. Genome Res 18 (2008) 821-829 Velvet

DeNovo Assembly

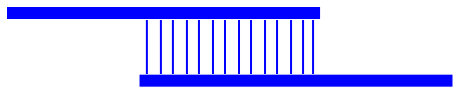


The assembly problem…

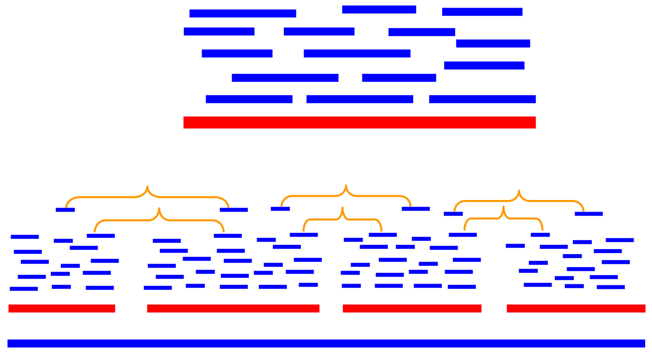
Overlap-Layout-Approach

Assemblers:ARACHNE, PHRAP, CAP, TIGR, CELERA, MIRA

Overlap: find potentially overlapping reads



Layout: merge reads into contigs and
contigs into supercontigs



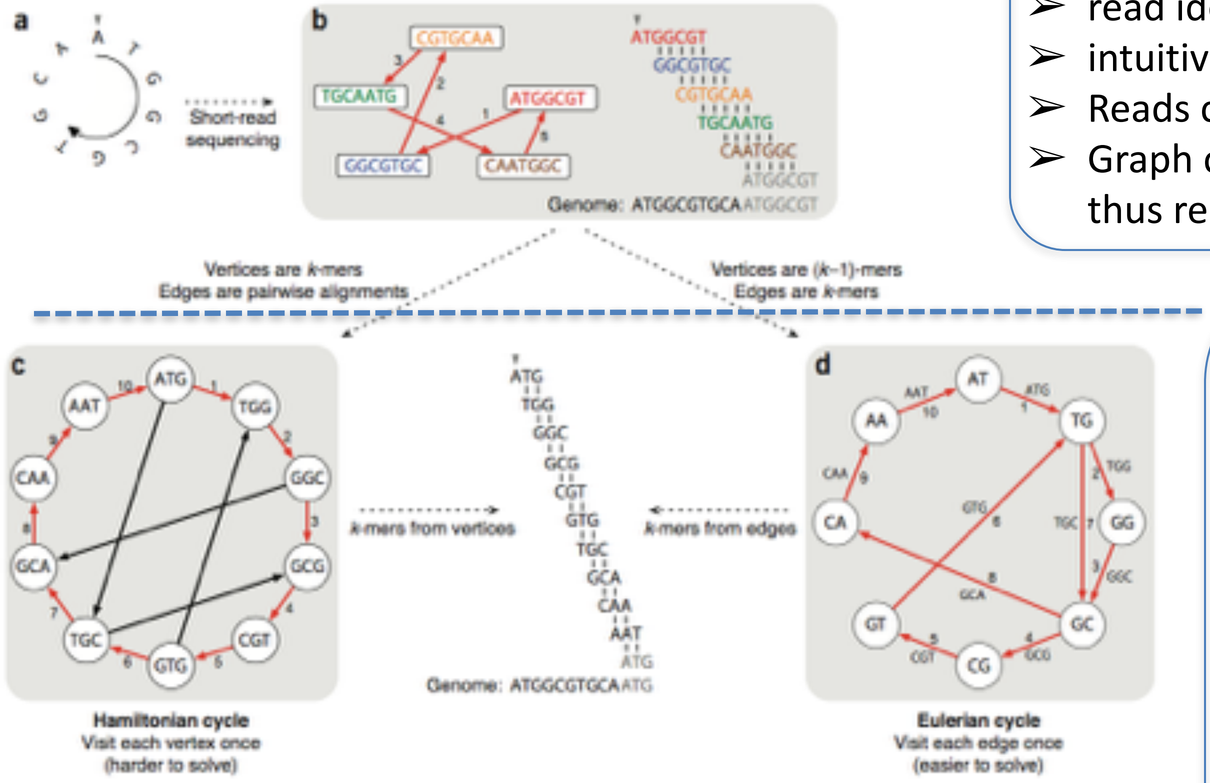
Consensus: derive the DNA sequence and
correct read errors

..ACGATTACAATAGGTT..

Overlap

- Find the best match between the suffix of one read and the prefix of another
- Due to sequencing errors, need to use dynamic programming to find the optimal *overlap alignment*
- Apply a filter to remove pairs of fragments that do not share a significantly long common substring

Different approaches to the sequence assembly problem



Overlap based assembly

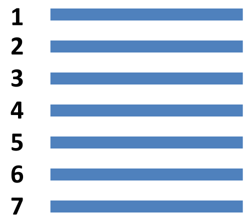
- read identity is maintained
- intuitive
- Reads can be organised in an overlap graph
- Graph complexity increases with coverage, thus read redundancy inflates the graph

Word-based approaches

- read identity is (temporarily) lost…
- Reads are organised in deBruijn graphs
- Graph complexity depends on word size
- Graph complexity is (by and large) independent from coverage, read redundancy is naturally handled
- repeats are represented only once in the graph with explicit links to the different start and end points

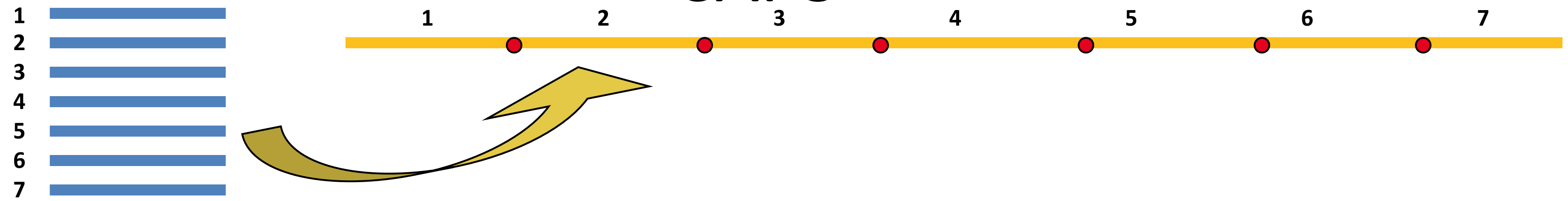
modfied from Compeau et al. (2011) Nature Biotechnology **29**(11)

De-Novo Sequence Assembly: CAP3



Shotgun Sequence Reads

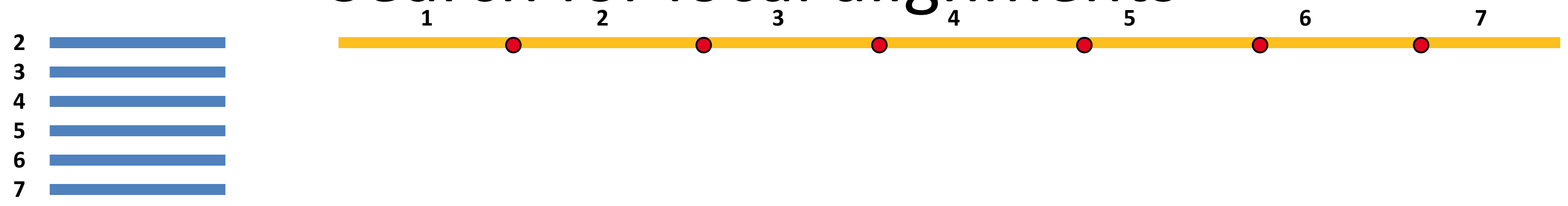
De-Novo Sequence Assembly: CAP3



Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

De-Novo Sequence Assembly (CAP3) Search for local alignments

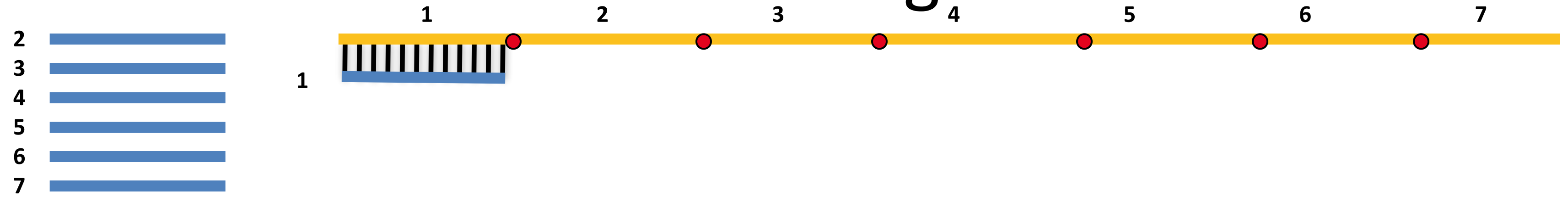


Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

De-Novo Sequence Assembly (CAP3)
Search for local alignments

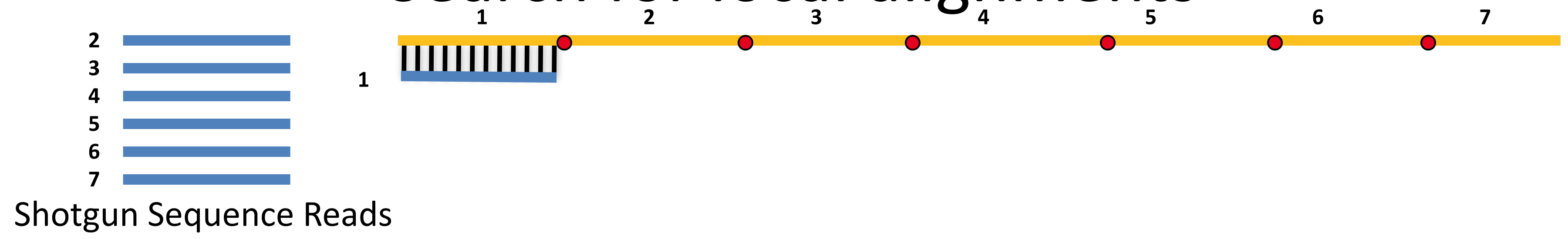


Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

De-Novo Sequence Assembly (CAP3)
Search for local alignments

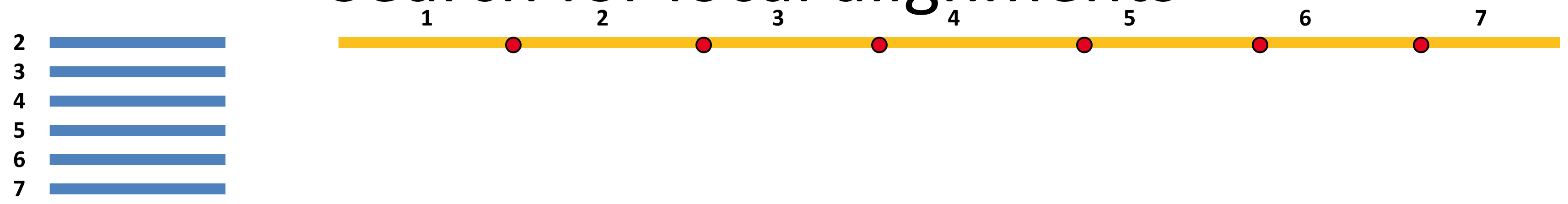


1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

Remove the trivial solution (alignment against itself)

De-Novo Sequence Assembly (CAP3) Search for local alignments



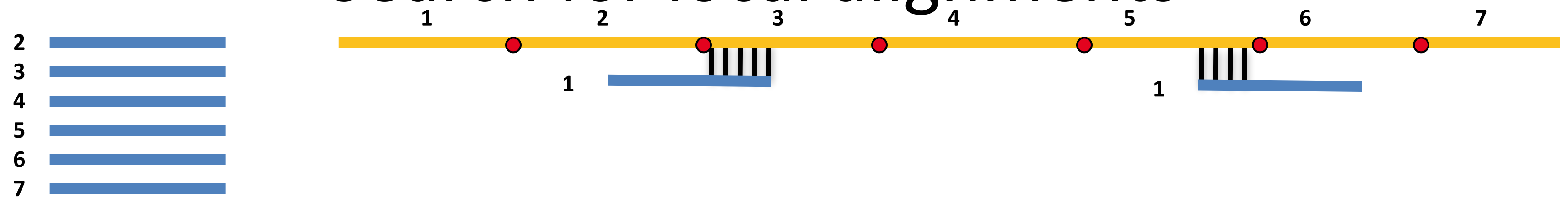
Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

Remove the trivial solution (alignment against itself)

De-Novo Sequence Assembly (CAP3)
Search for local alignments

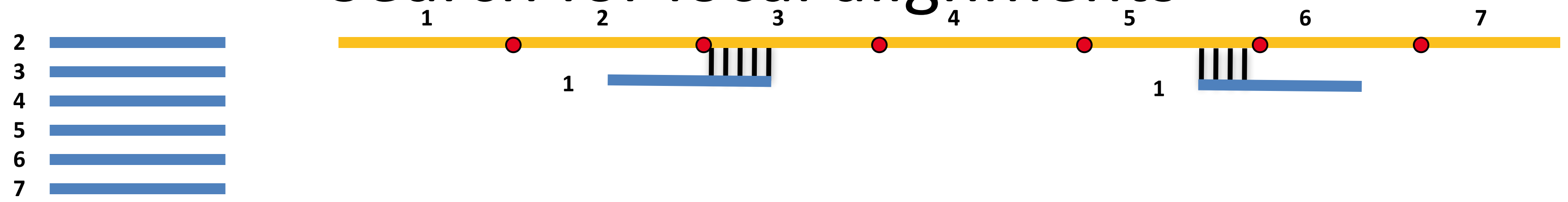


Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

De-Novo Sequence Assembly (CAP3)
Search for local alignments

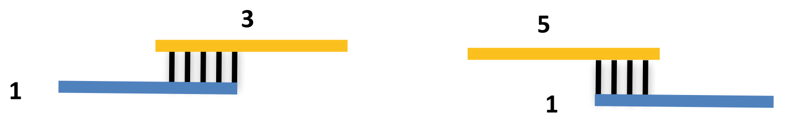


Shotgun Sequence Reads

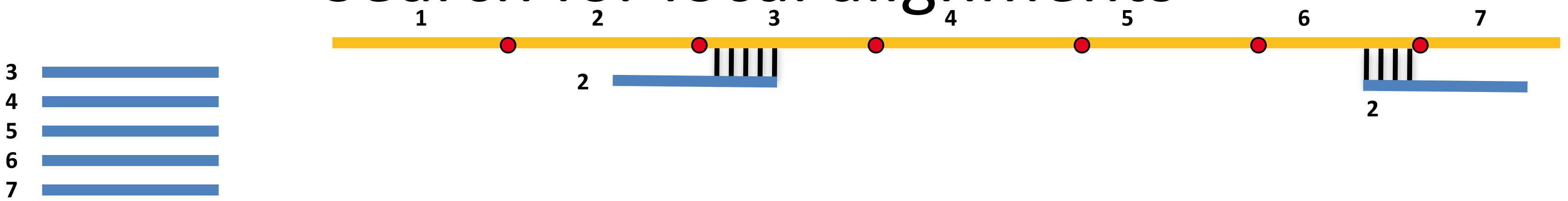
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

Candidate pairs for read 1:



De-Novo Sequence Assembly (CAP3) Search for local alignments

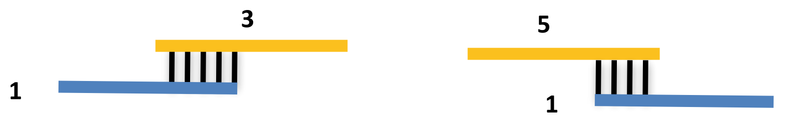


Shotgun Sequence Reads

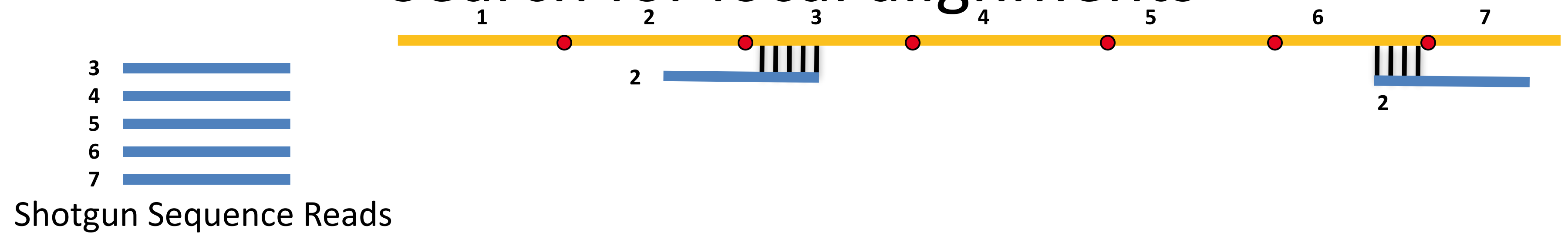
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

Candidate pairs for read 1:



De-Novo Sequence Assembly (CAP3)
Search for local alignments

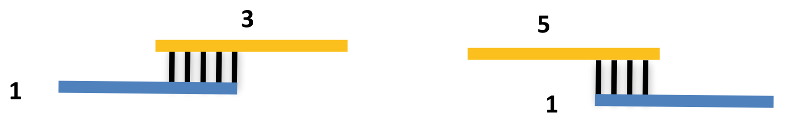


Shotgun Sequence Reads

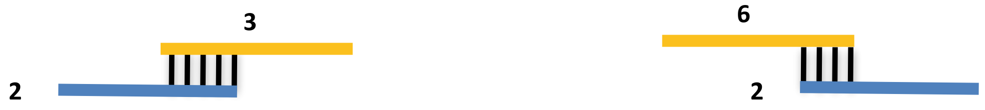
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

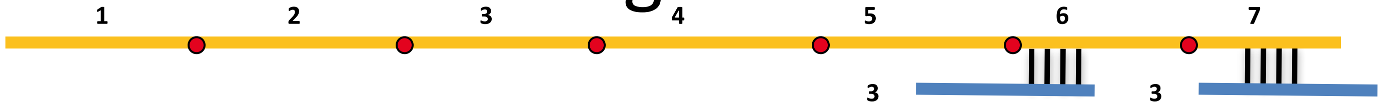
Candidate pairs for read 1:



Candidate pairs for read 2:



De-Novo Sequence Assembly (CAP3)
Search for local alignments

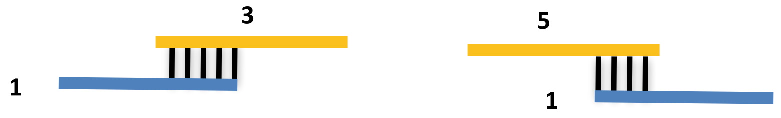


Shotgun Sequence Reads

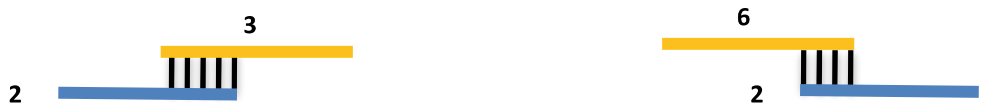
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

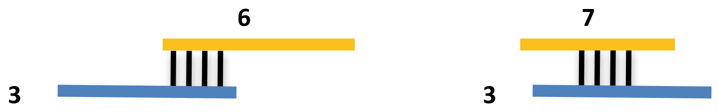
Candidate pairs for read 1:



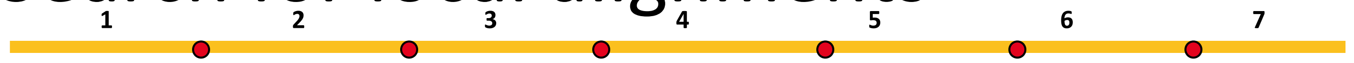
Candidate pairs for read 2:



Candidate pairs for read 3:



De-Novo Sequence Assembly (CAP3) Search for local alignments

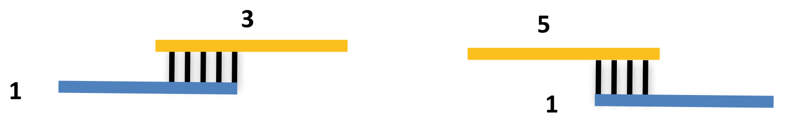


Shotgun Sequence Reads

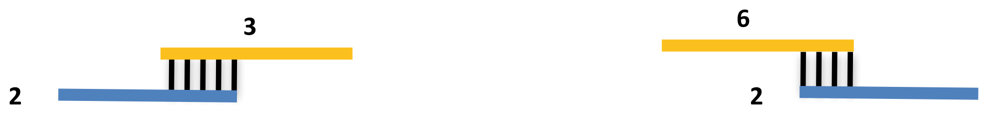
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

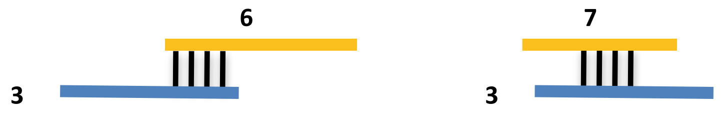
Candidate pairs for read 1:



Candidate pairs for read 2:



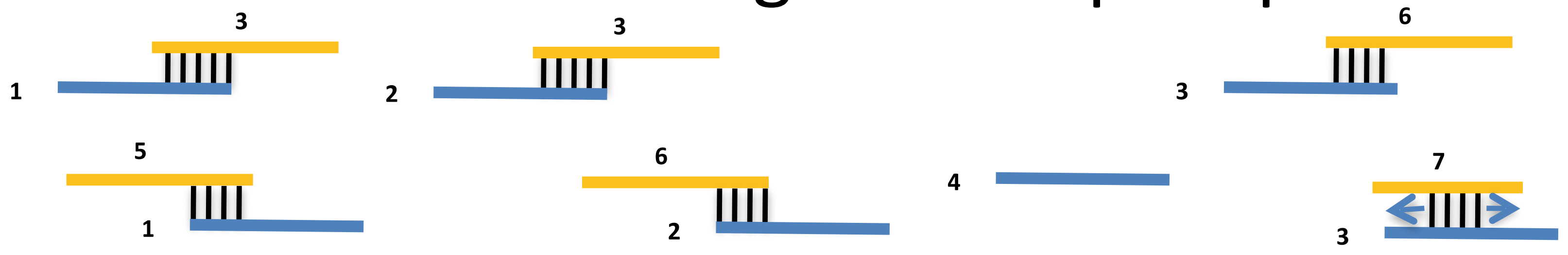
Candidate pairs for read 3:



.
.
.

Candidate pairs for read 7:

De-Novo Sequence Assembly (CAP3)
Search for local alignments: post-processing



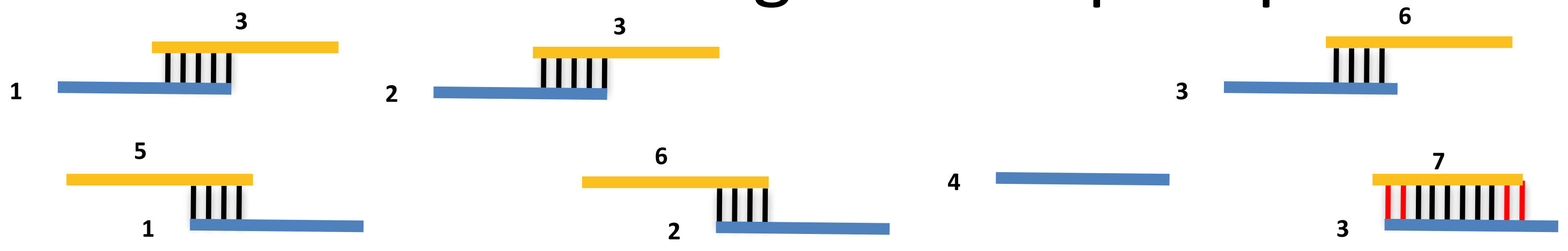
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

3. Remove poor quality sequence ends

4. Compute global alignment for the high quality sequence pairs to verify overlaps.

De-Novo Sequence Assembly (CAP3)
Search for local alignments: post-processing



1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

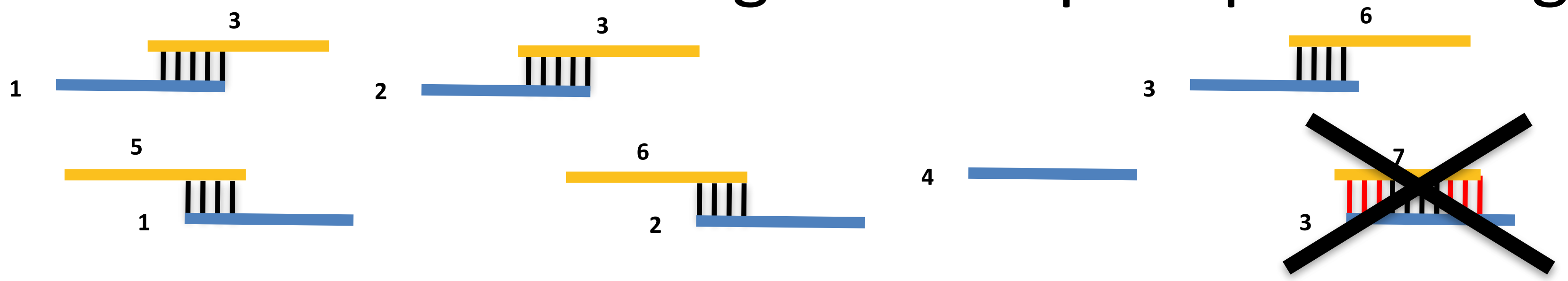
2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

3. Remove poor quality sequence ends

4. Compute global alignment for the high quality sequence pairs to verify overlaps. Evaluate according to the following criteria:
    1. minimum length
    2. minimum identity
    3. minimum similarity
    4. number of high-quality mismatches

Remove sequence pairs that do not meet the thresholds for 4.1 to 4.4

De-Novo Sequence Assembly (CAP3)
Search for local alignments: post-processing



1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

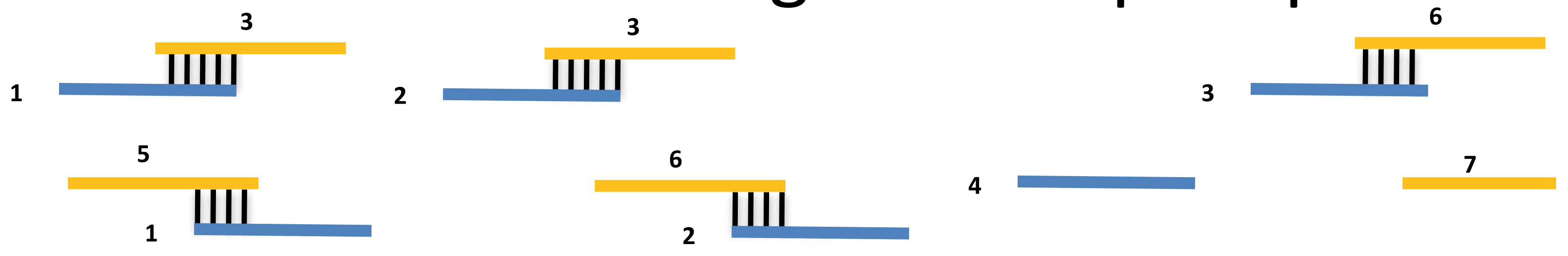
2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

3. Remove poor quality sequence ends

4. Compute global alignment for the high quality sequence pairs to verify overlaps. Evaluate according to the following criteria:
   1. minimum length
   2. minimum identity
   3. minimum similarity
   4. number of high-quality mismatches

   Remove sequence pairs that do not meet the thresholds for 4.1 to 4.4

De-Novo Sequence Assembly (CAP3)
Search for local alignments: post-processing



1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

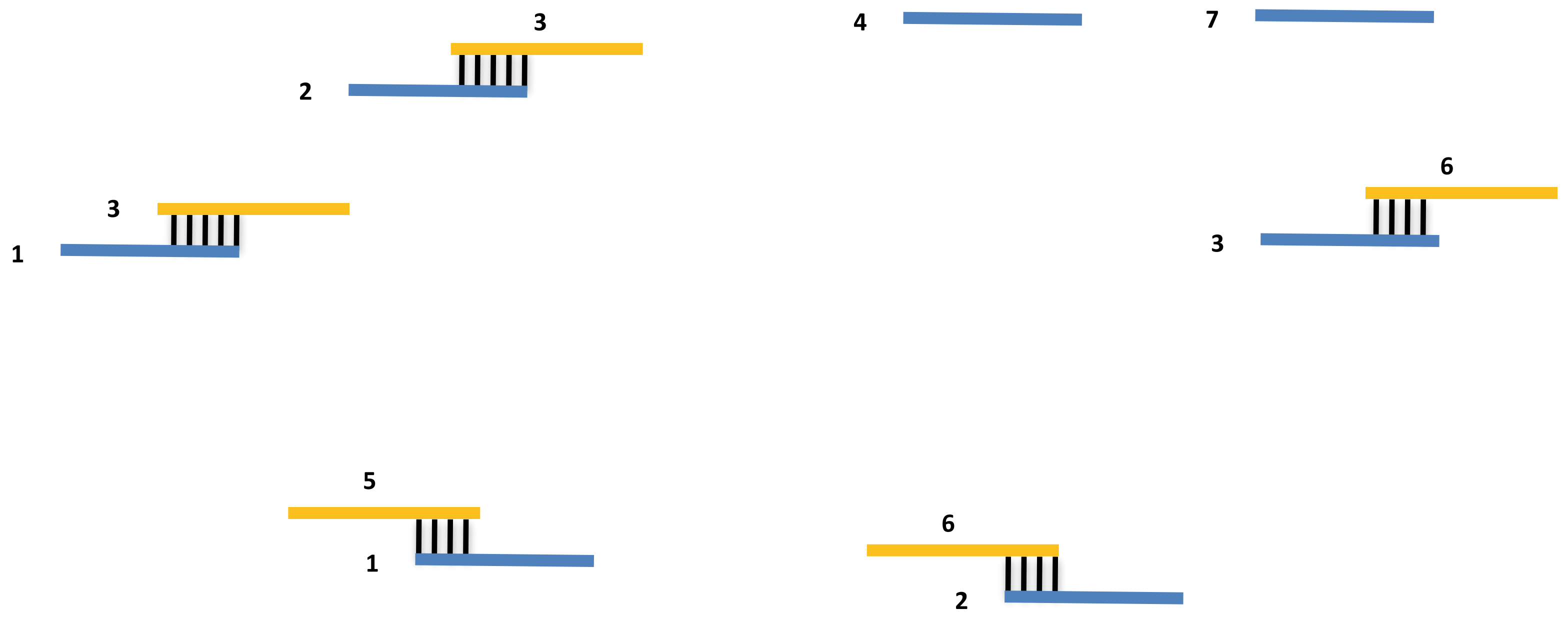
2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

3. Remove poor quality sequence ends

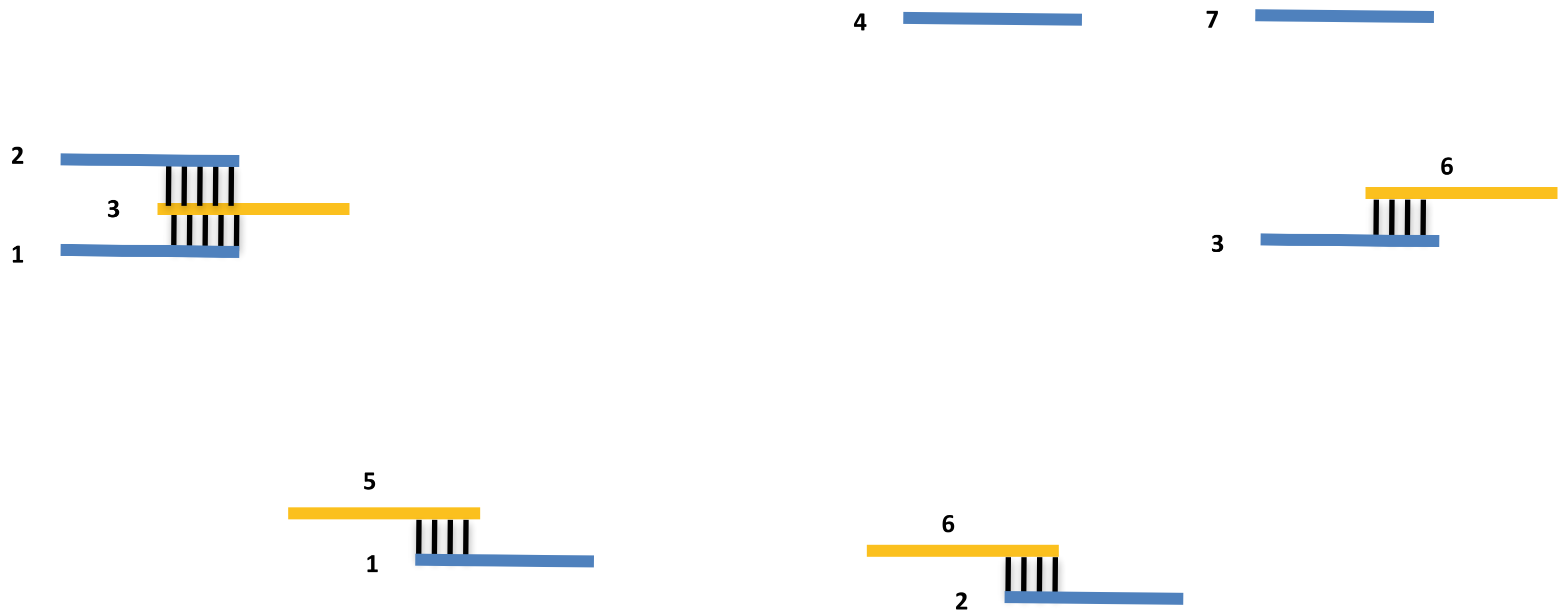
4. Compute global alignment for the high quality sequence pairs to verify overlaps. Evaluate according to the following criteria:
   1. minimum length
   2. minimum identity
   3. minimum similarity
   4. number of high-quality mismatches

   Remove sequence pairs that do not meet the thresholds for 4.1 to 4.4

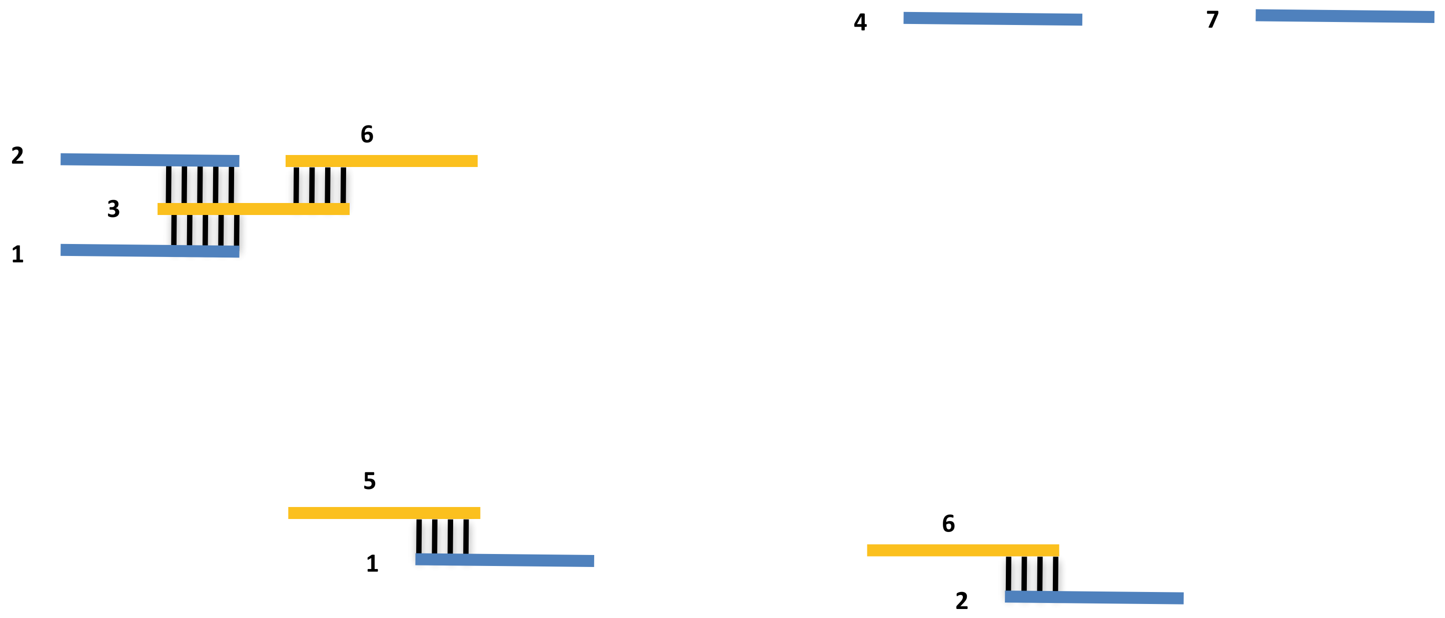
CAP3: Contig Building



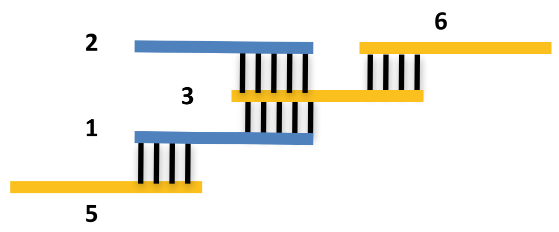
CAP3: Contig Building

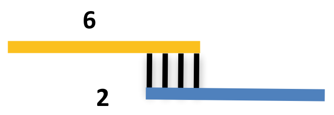


CAP3: Contig Building

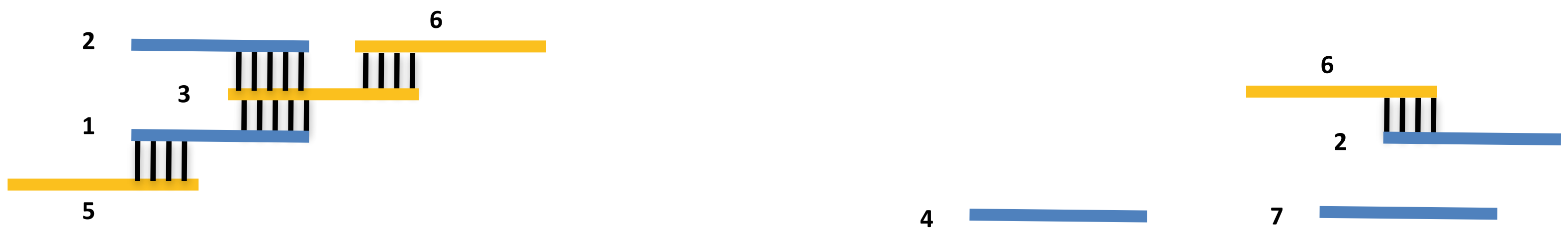


CAP3: Contig Building



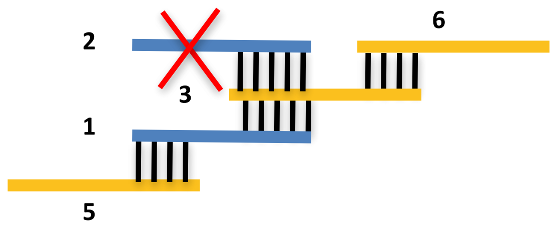


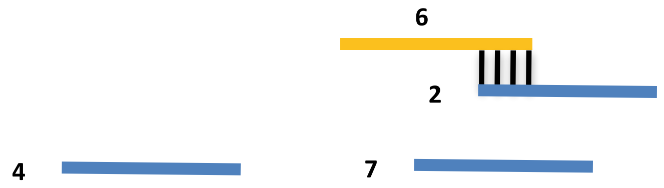
CAP3: Contig Building



1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities.

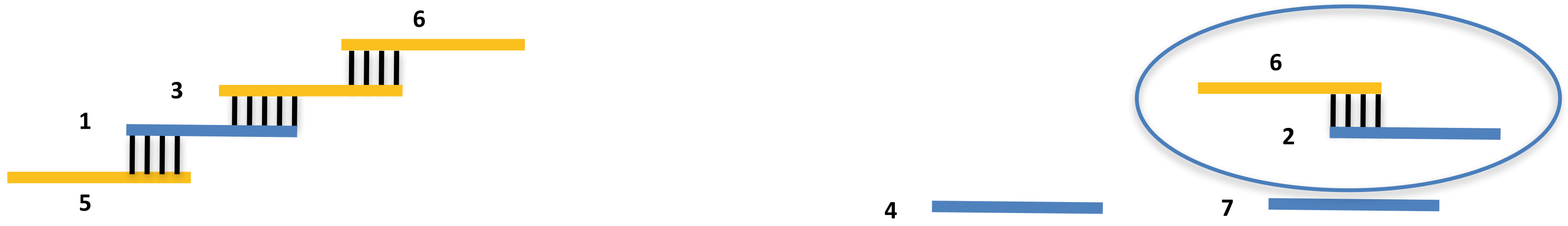
CAP3: Contig Building





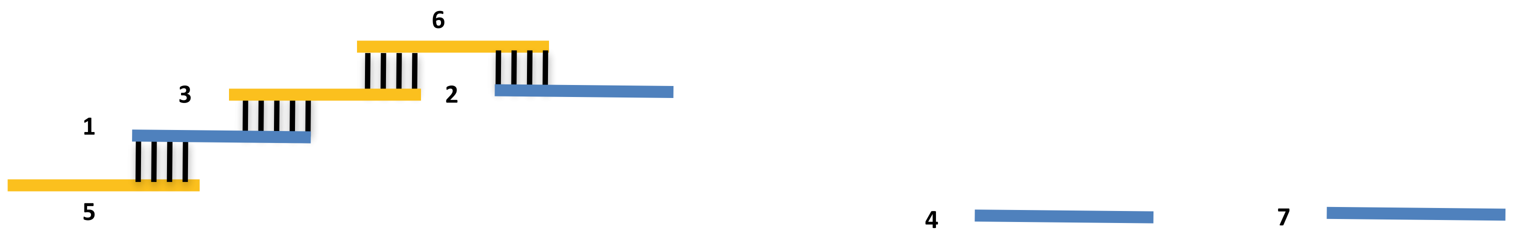
1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities.
   1) sequence read 1 and 2 are incompatible since they could not be aligned.

CAP3: Contig Building



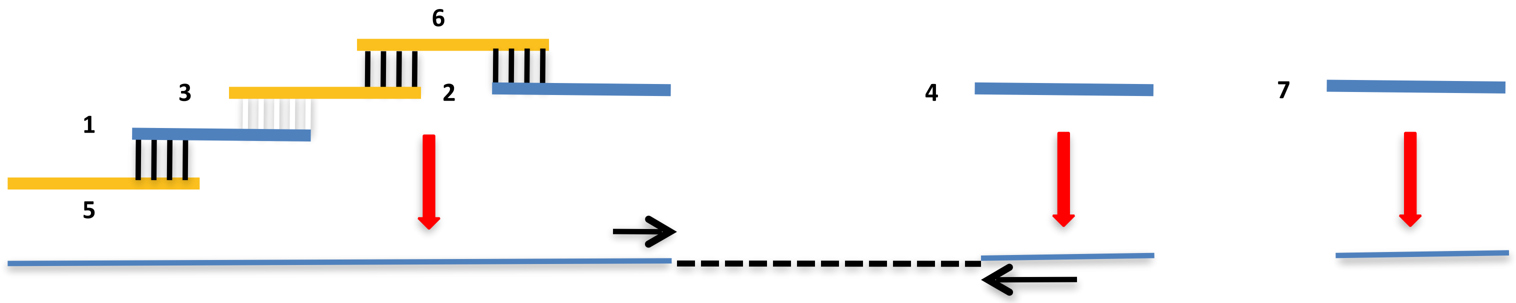
1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities.
   1) sequence read 1 and 2 are incompatible since they could not be aligned.
   2) resolve incompatibility
   3) check for new possible layouts

CAP3: Contig Building



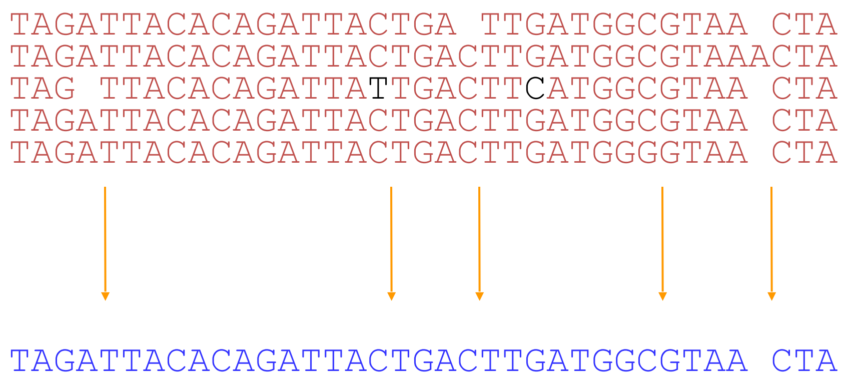
1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities.
    1) sequence read 1 and 2 are incompatible since they could not be aligned.
    2) resolve incompatibility
    3) check for new possible layouts

Further steps in genome sequencing



1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities, remove incompatible reads and align.
3) Build a consensus sequence for each contigs.
4) Order and orient contigs if possible using additional information, e.g., paired end reads.

Derive Consensus Sequence



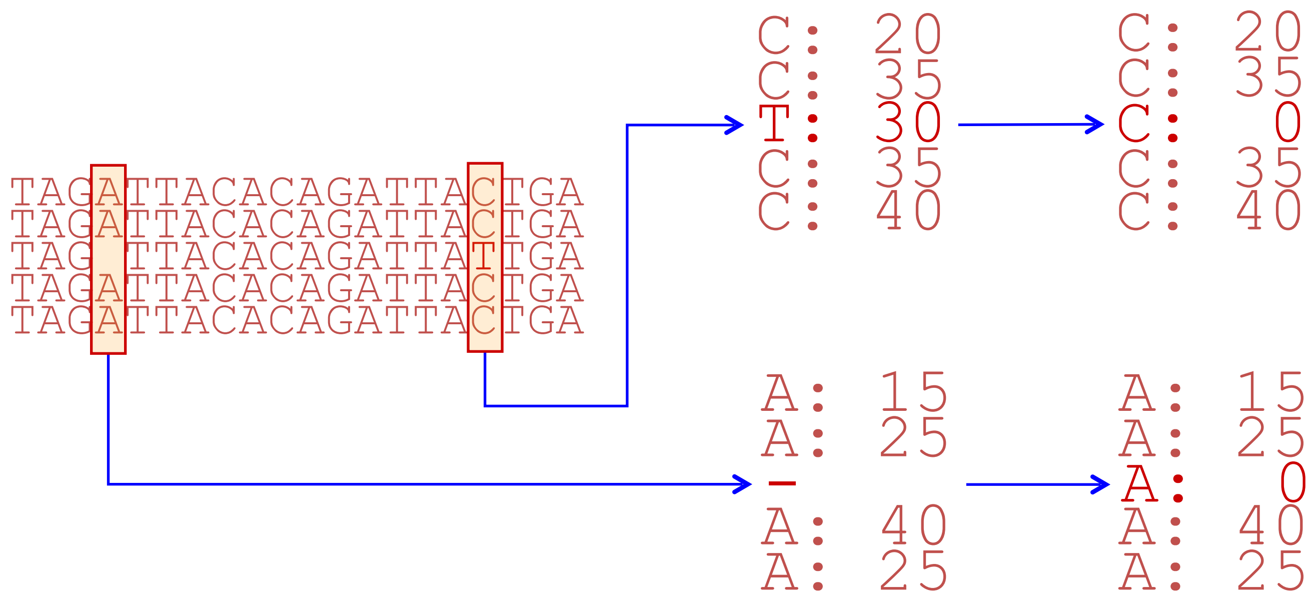
```
TAGATTACACAGATTACTGA TTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAAACTA
TAG TTACACAGATTATTGACTTCATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGGGTAA CTA

TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```

Derive multiple alignment from pairwise read alignments

Derive each consensus base by weighted voting

Error correction by weighted voting

- Correct errors using multiple alignment

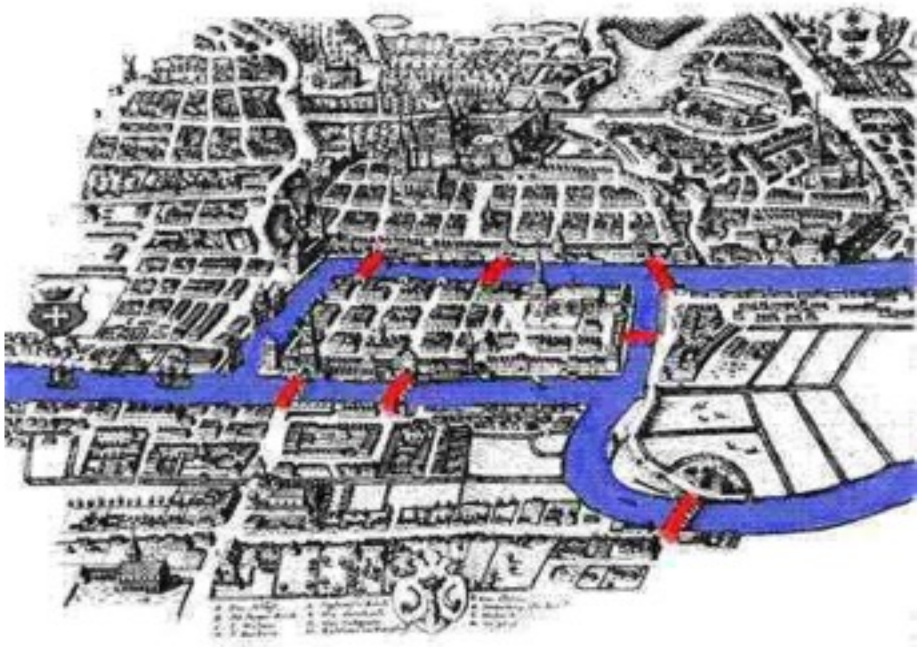


- Score alignments
- Accept alignments with good scores

Consensus (cont’d)

- A consensus sequence is derived from a profile of the assembled fragments
- A sufficient number of reads is required to ensure a statistically significant consensus
- Reading errors are corrected

Sequence assembly with De Bruijn graphs



Sequence assembly can be abstracted to the shortest superstring problem (Nicolas de Bruijn 1946)

Problem: find the shortest (circular) superstring that contains all possible substrings of length *K* over a given alphabet.

for *K = 4* and a two letter alphabet *A={0,1}* we have 16 different words:

0000, 0001, 0010, 0100, 1000, 0011, 0110, 1100, 1001, 1010, 0101, 0111, 1011, 1101, 1110, 1111

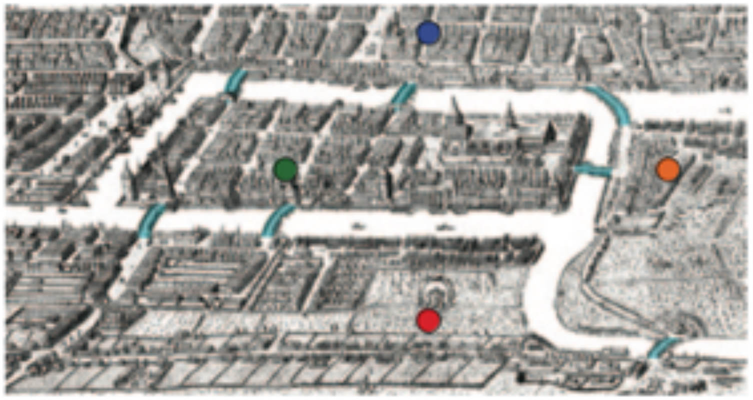
Sequence assembly can be abstracted to the shortest superstring problem (Nicolas de Bruijn 1946)

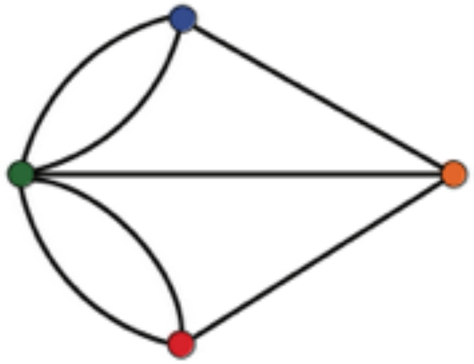
Problem: find the shortest (circular) superstring that contains all possible substrings of length *K* over a given alphabet.

for *K = 4* and a two letter alphabet *A={0,1}* we have 16 different words:

0000, 0001, 0010, 0100, 1000, 0011, 0110, 1100, 1001, 1010, 0101, 0111, 1011, 1101, 1110, 1111

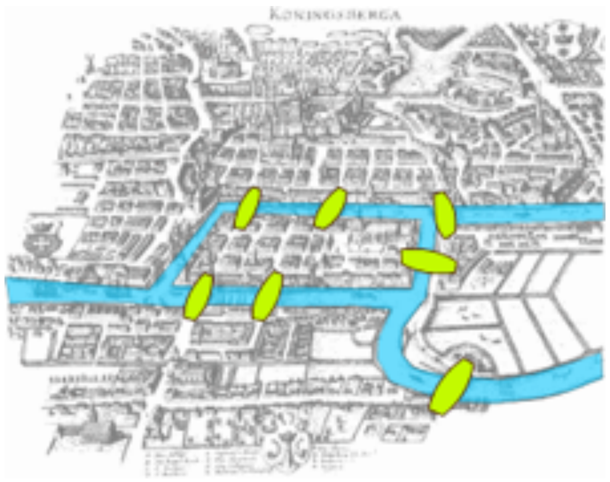
To solve this problem, de Bruijn borrowed from Euler who solved 1735 the 'Königsberg' problem, i.e. the question whether it is possible to visit each island by crossing each bridge exactly once (Eulerian cycle)

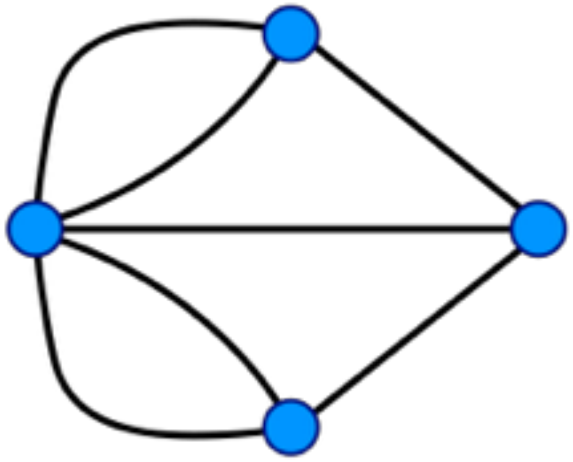




Eulerian Cycle Problem

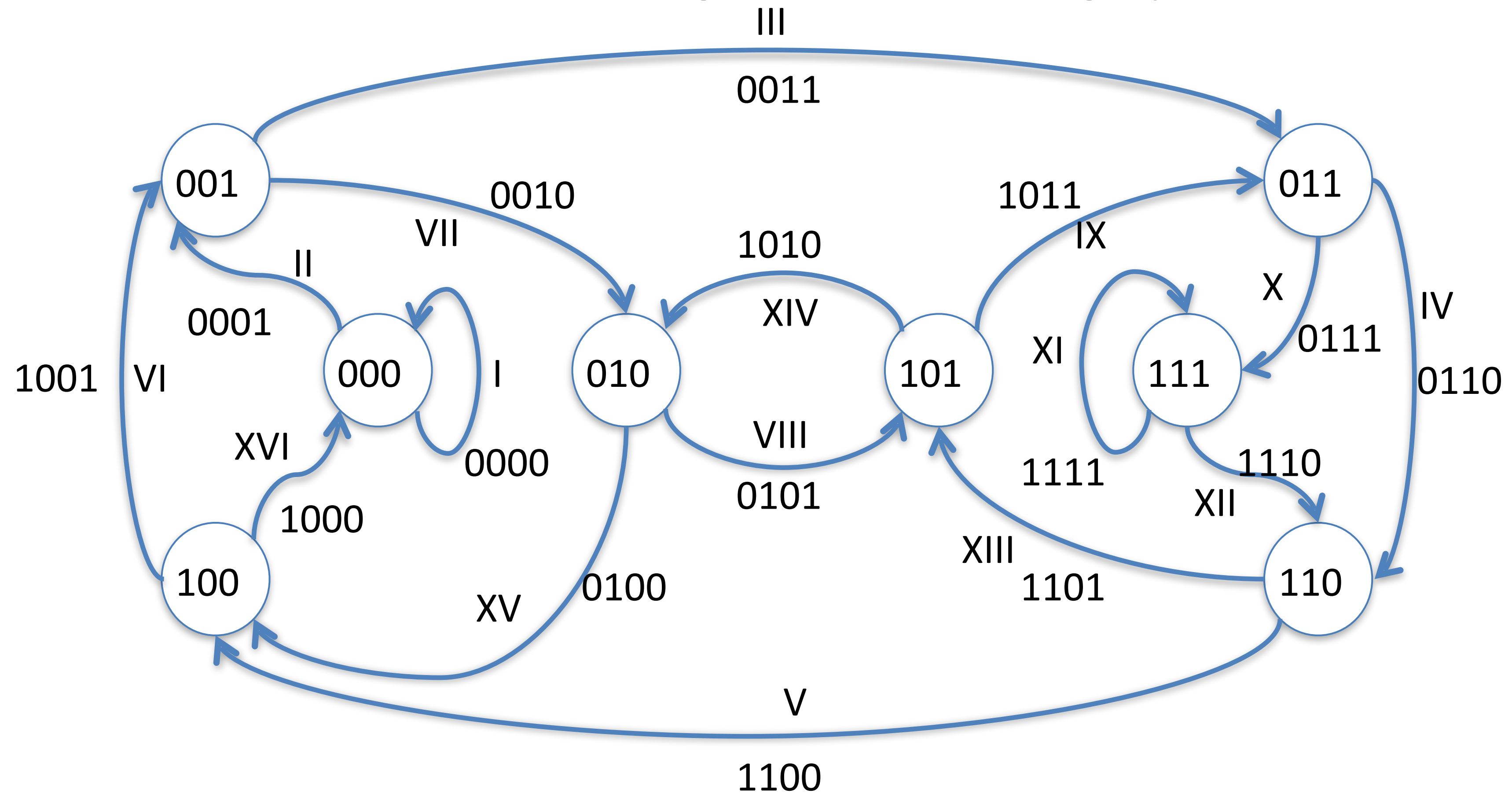
- Find a cycle that visits every ***edge*** exactly once (Linear time)
- An Eulerian Cycle exists if the number of 'outgoing' edges for a node equals the number of 'incoming' edges*.
- The graph may have 2 nodes with an odd number of edges connected to it. In this case an Eulerian path rather than an Eulerian cycle can be found.





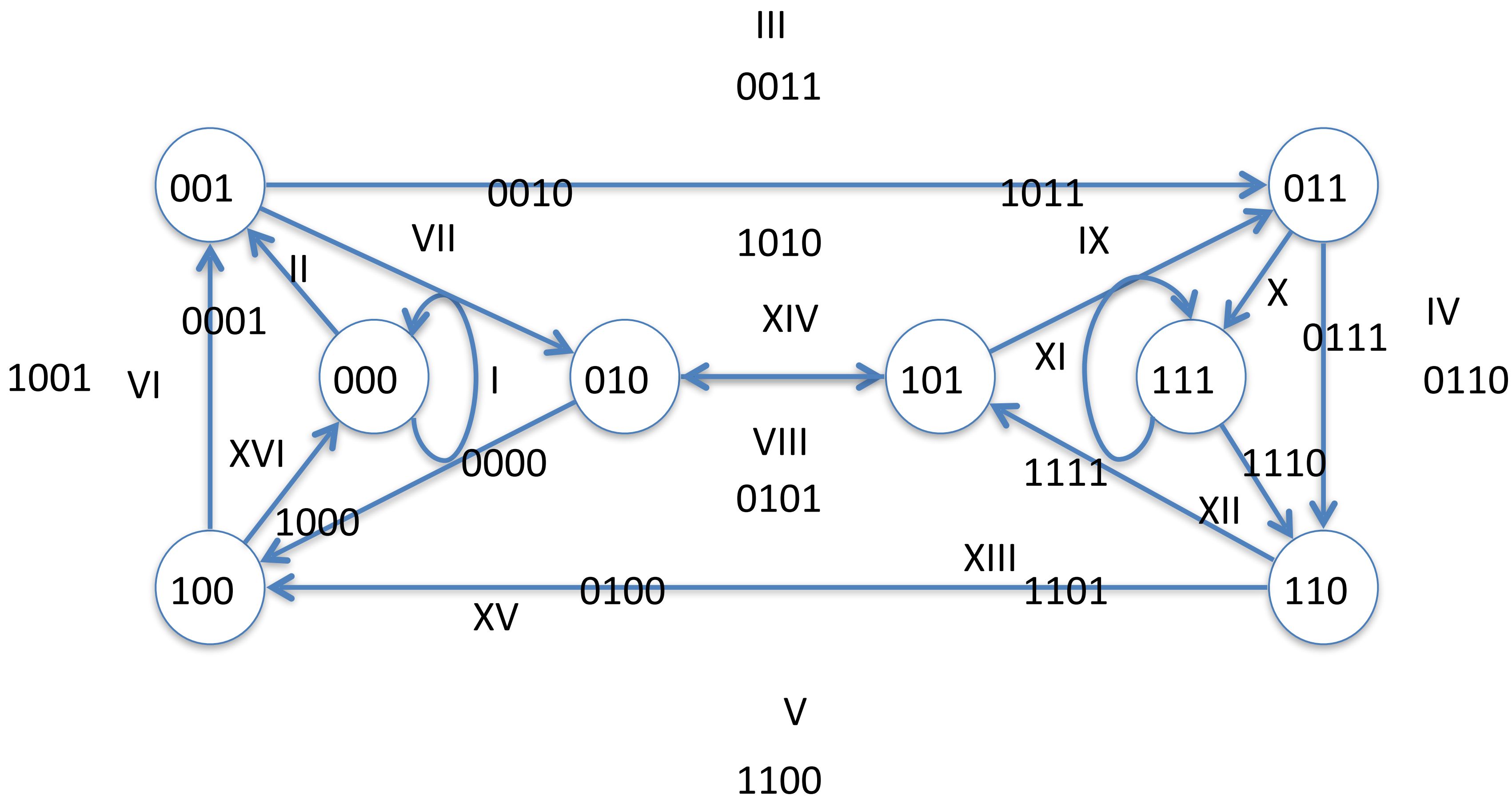
*Proof by C. Hierholzer

de Bruijn solved the problem by representing *K*-1 mers as nodes and *K* mers as edges in a directed graph.



By doing so, he related the problem of finding a shortest common superstring to the already solved problem of finding an Eulerian cycle in a graph.

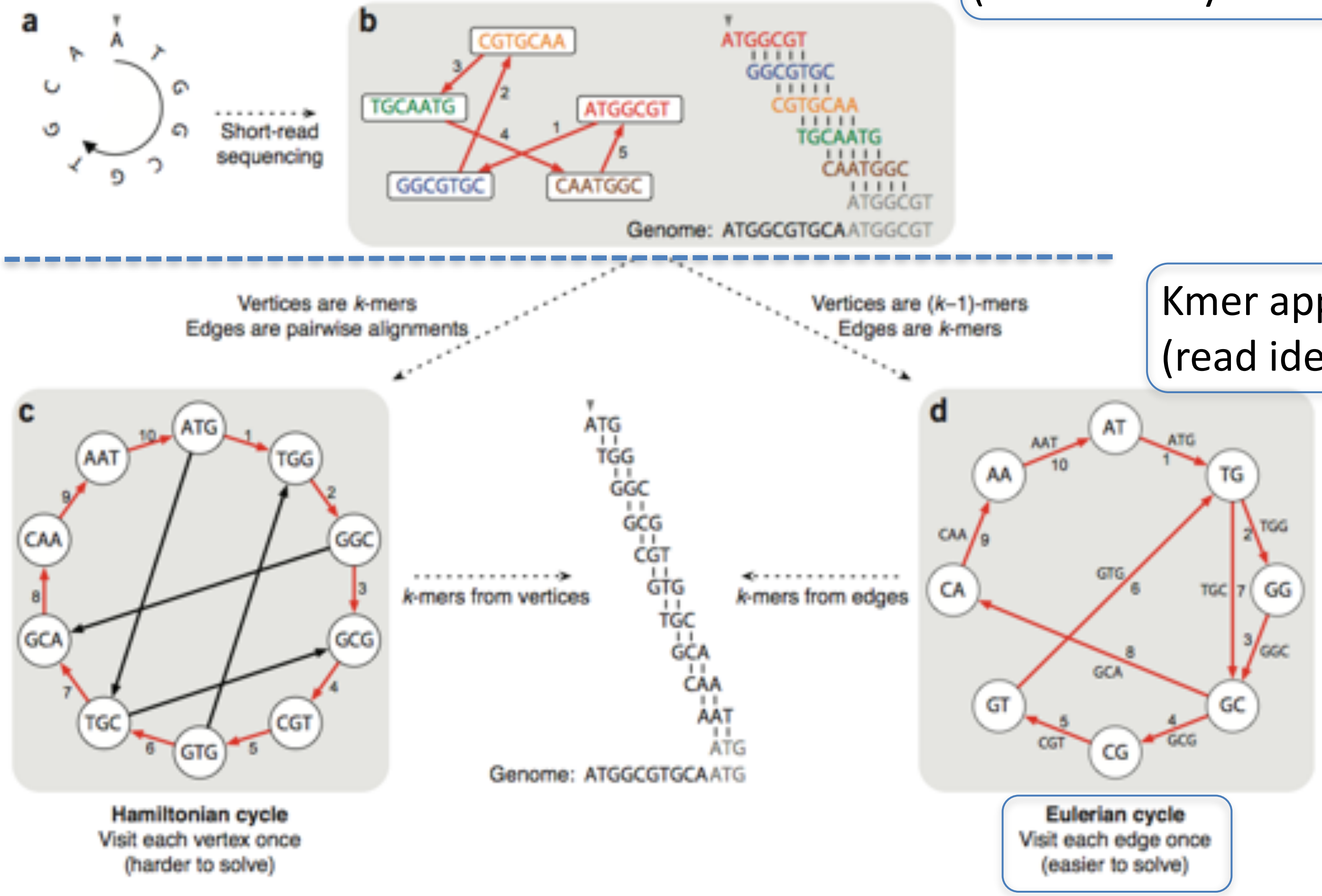
Passing through the edges by following the roman numbers reconstructs the superstring using each word exactly once!



I: 0000, II: 0001, III: 0011; IV: 0110; V: 1100; VI: 1001; VII: 0010; VIII: 0101; IX: 1011; X: 0111; XI: 1111; XII: 1110; XIII: 1101; XIV: 1010; XV: 0100; XVI: 1000

$$\boxed{0000110010111101}$$

Advancing to DNA sequence assembly is straightforward



Classical overlap assembly (read identity is maintained)

Kmer approaches (read identity is lost)

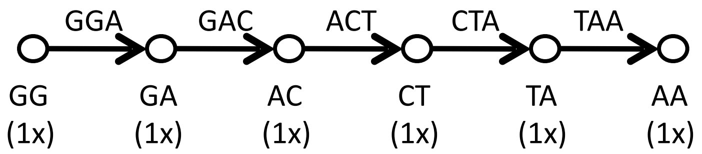
modfied from Compeau et al. (2011) Nature Biotechnology **29**(11)

De Bruijn Graph Example

Shred reads into k-mers (k = 3)

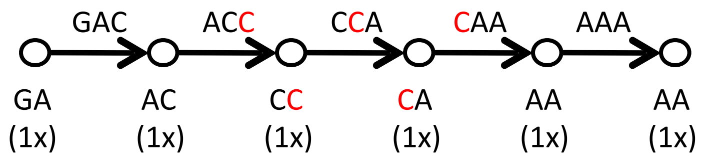
Read 1

```
G   G   A   C   T   A   A

G   G   A
    G   A   C
        A   C   T
            C   T   A
                T   A   A
```



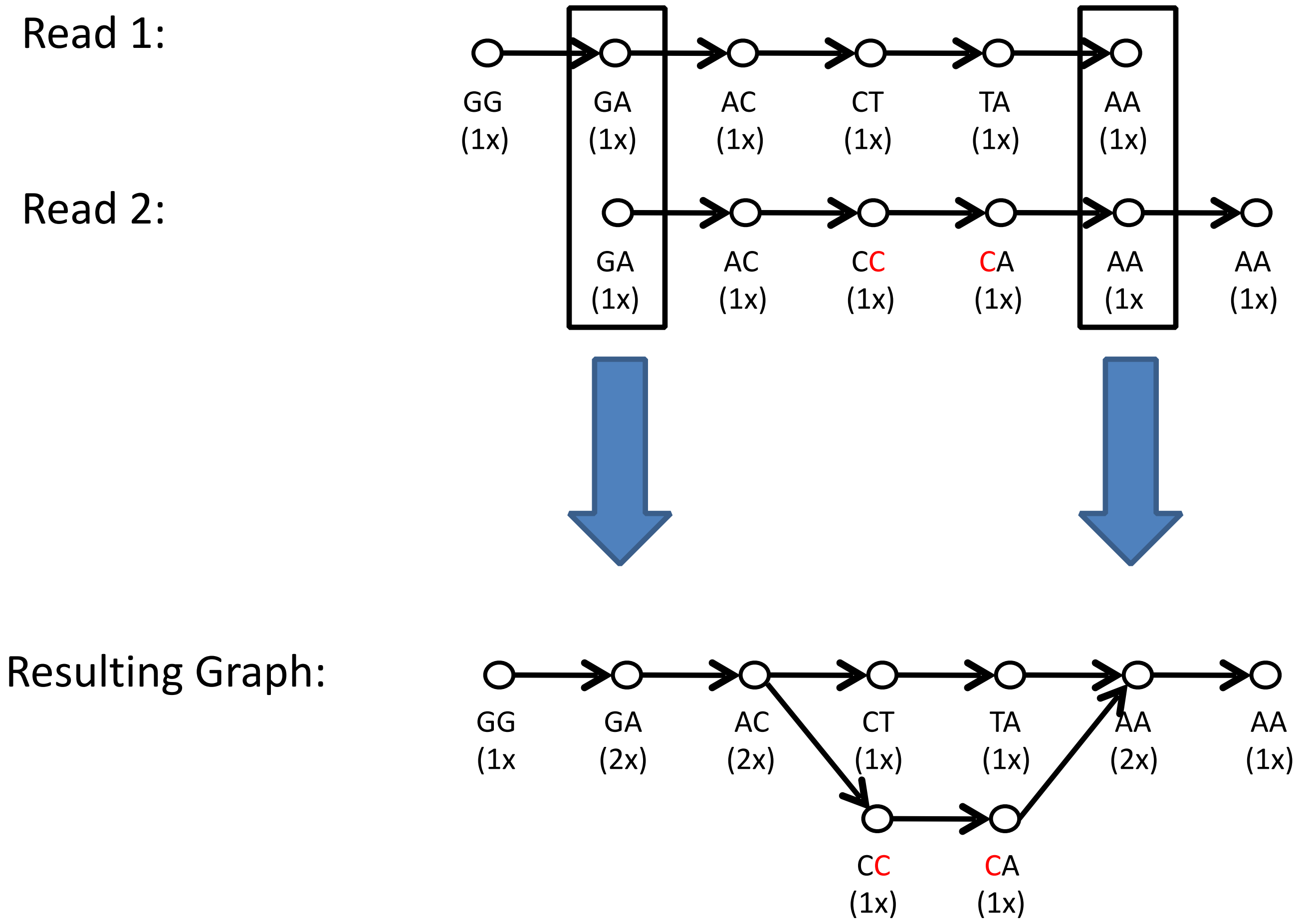
Read 2

```
G   A   C   C   A   A   A

G   A   C
    A   C   C
        C   C   A
            C   A   A
                A   A   A
```



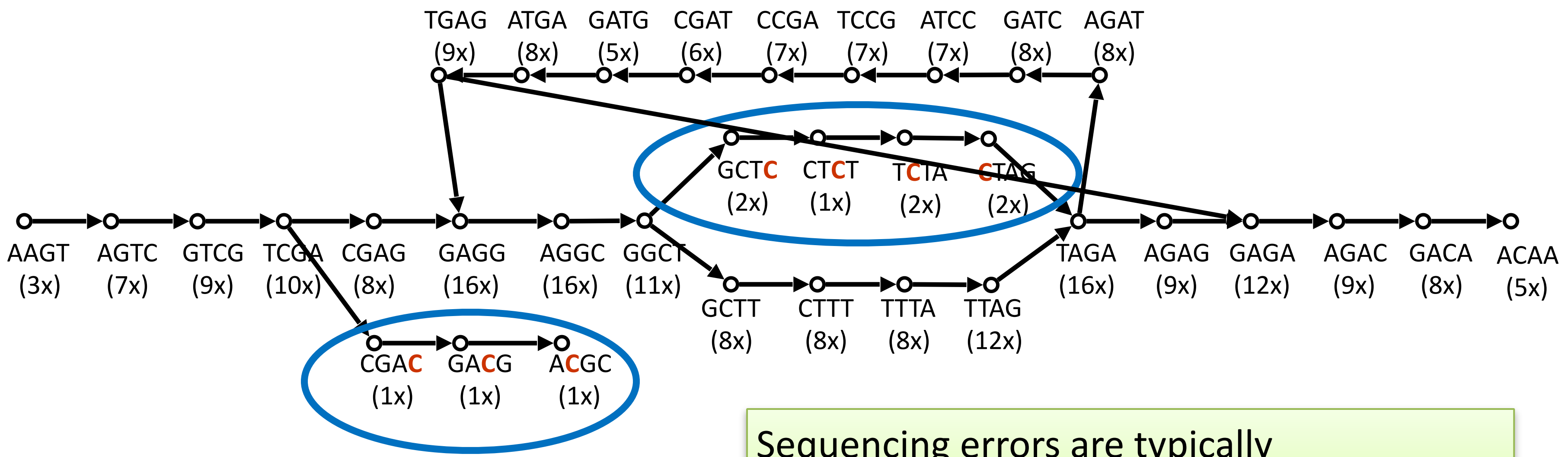
De Bruijn Graph Example

Merge vertices labeled by identical (k-1)-mers



Another Example

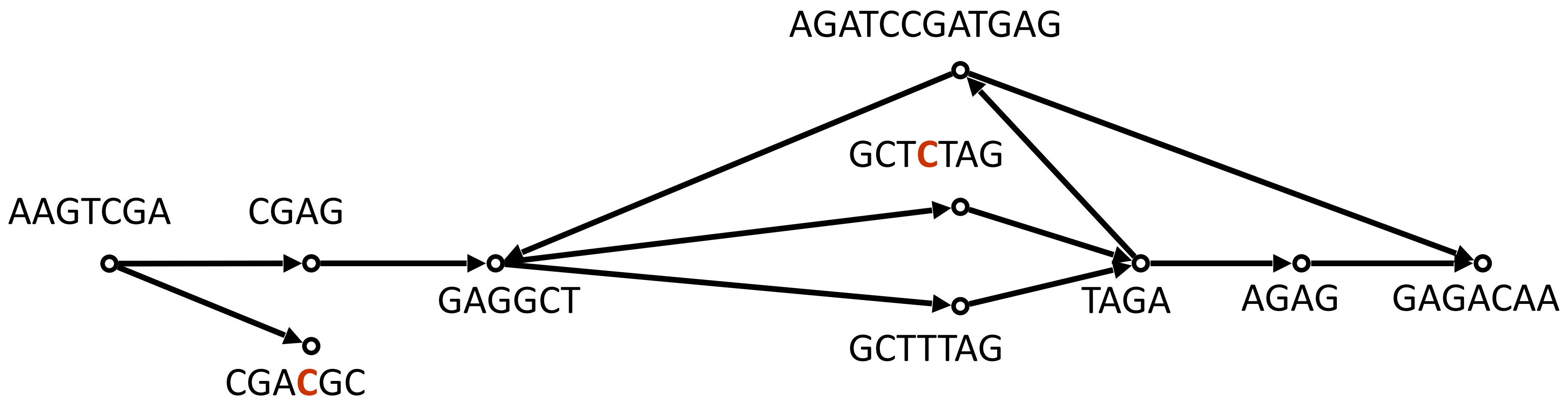
Construct the graph (k = 5)



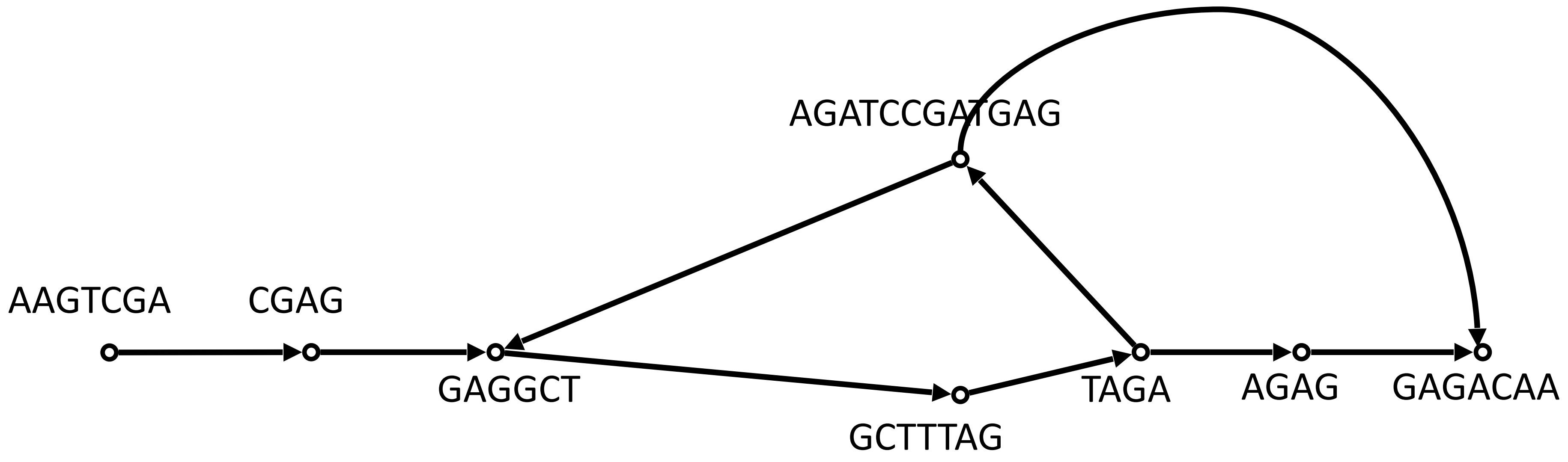
Sequencing errors are typically detected by a coverage cutoff threshold

A branching vertex is caused by either a repeat in the original sequence or a sequencing error

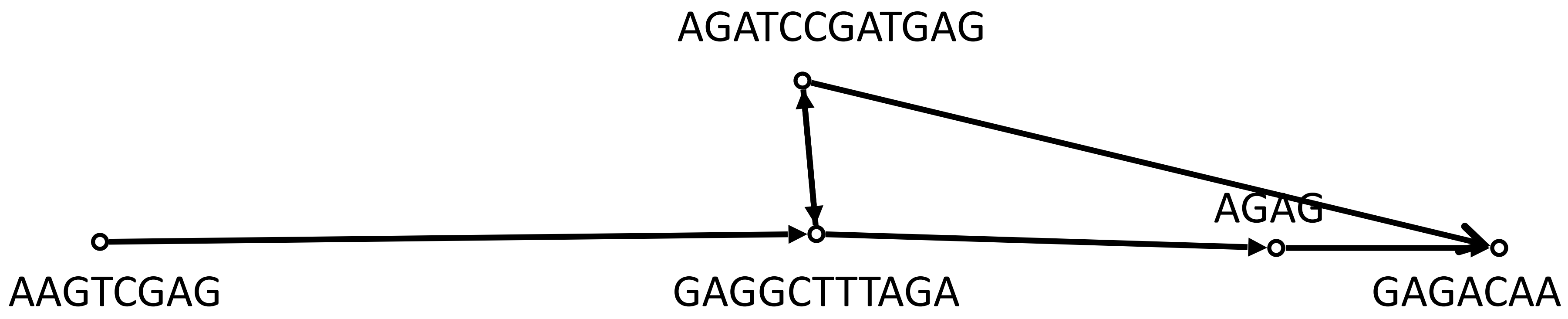
Condense unbranched runs in the graph



Correct sequencing errors using a coverage threshold



After recondensation



Any non-branching path in this graph corresponds to a contig in the original sequence.

Taking the risk of following arbitrary branching paths may create chimeric species

Source: Serafim Batzoglou

Basic concepts of de Bruijn graph based assemblers

- The sequence is treated as a consecutive string of words of length *K*
- Sequence reads are no longer considered to represent a consecutive string of nucleotides. Thus read length as well as read overlap become, in principle, irrelevant.
- Sequence reads are only used to identify words of length *K* occurring in the sequence.
- Given perfect data – error-free K-mers providing full coverage and spanning every repeat – the K-mer graph would be a de Bruijn graph and it would contain an Eulerian path, that is, a path that traverses each edge exactly once.

The magic ‘*Kmer*’ gives most users of graph based assembly algorithms a very hard time as they have to decide on the size of *K.*

To give an informed statement we need to make sure to understand what *K* should represent and what the algorithmic requirements of de Bruijn graph assemblers are

- *K* must represent a word that occurs **only once** in the sequence that should be assembled. Thus, *K* must be **sufficiently large**.
- de Bruijn graph based assemblers assume that **each word** of length *K* occurring in the genome is also represented in the graph. As *Kmers* are collected from a finite set of sequence reads, ***K* must not be too large**.

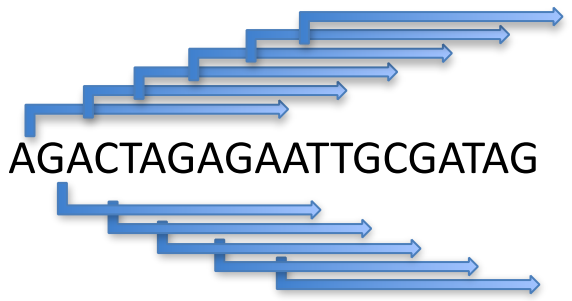
- consider a DNA word of *K=2,* how often does it on average occur in a string of 16 bp?

Take home message: If *K* is only sufficiently large the chance for any *Kmer* to occur more than once in a (repeat-free) genome approaches 0.

Why not using simply the read length as *K?*

Default for Trinity (Grabherr et al. 2011)

Why K must not be too large



A sequence of length 20 contains 11 different words of length 10!

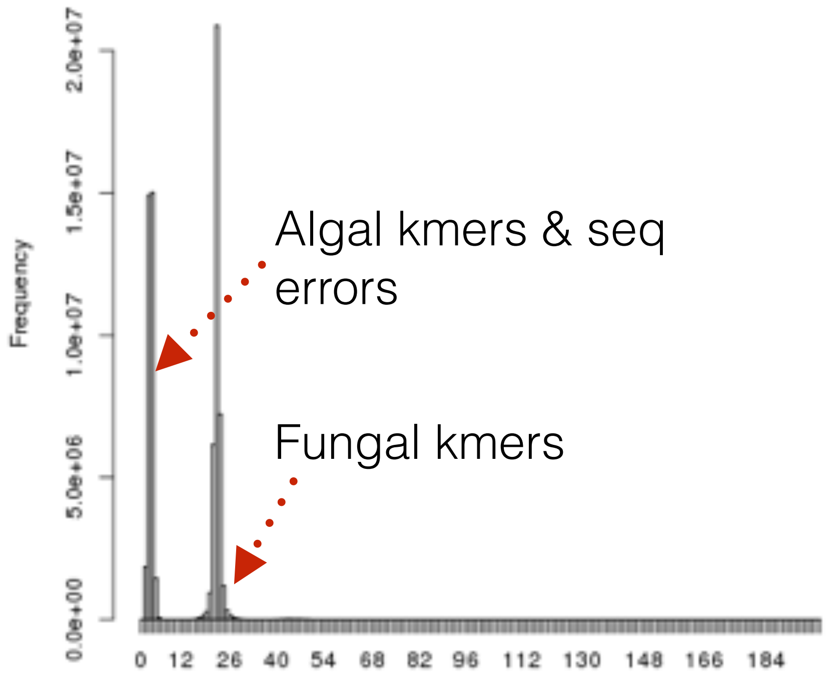
Now, consider the sequence is spanned by 2 reads of length 13:

```
T:   AGACTAGAGAATTGCGATAG
R1:  AGACTAGAGAATT
R2:         AGAATTGCGATAG
```

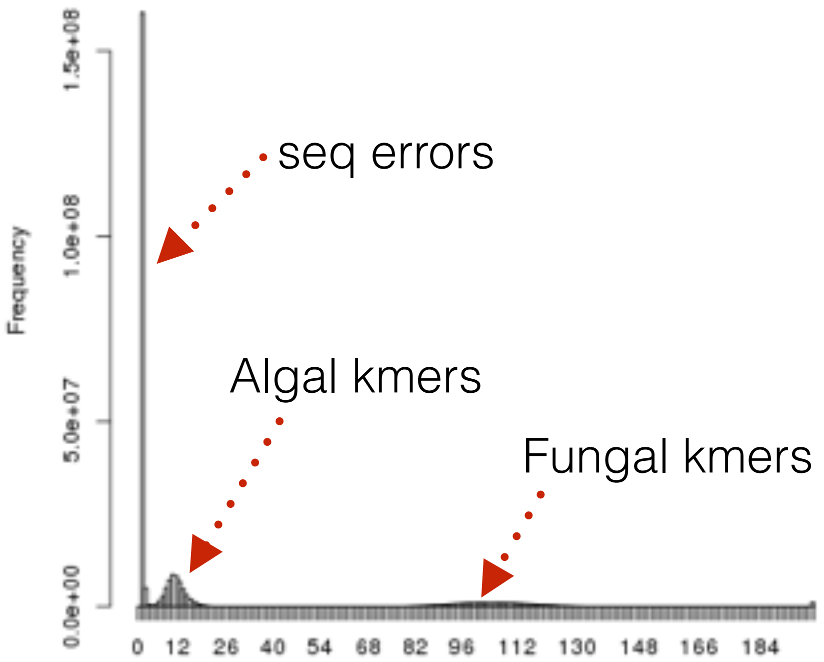
It is easy to see that not all 11 words of length 10 can be reconstructed with the two reads.
This violates the key assumption of the de Bruijn graphs

It is also easy to see that reducing *K* ameliorates the problem and eventually gets rid of it (just consider *K*=1…)

Assembly parameter optimization using maximization of average contig length (N50) as objective can preclude an entire genome from being assembled



k = 151
N50: 57 kbp
Reference: 40%



k = 51
N50: 22 kbp
Reference: 99%