



Genômica Computacional

Montagem de genomas

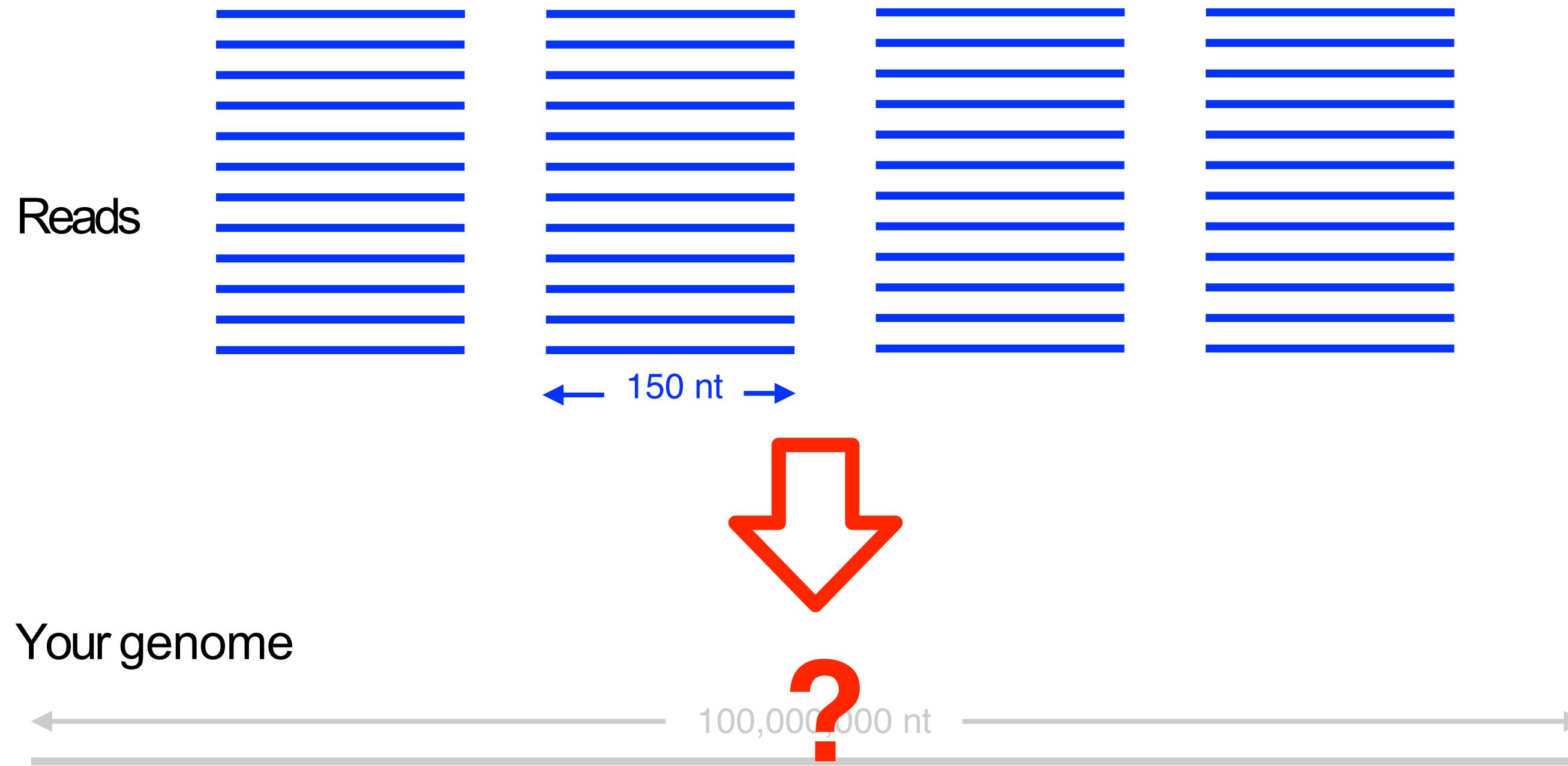
Professor: Ricardo A. Vialle
CS31 - Genômica Computacional

1 de Novembro de 2023

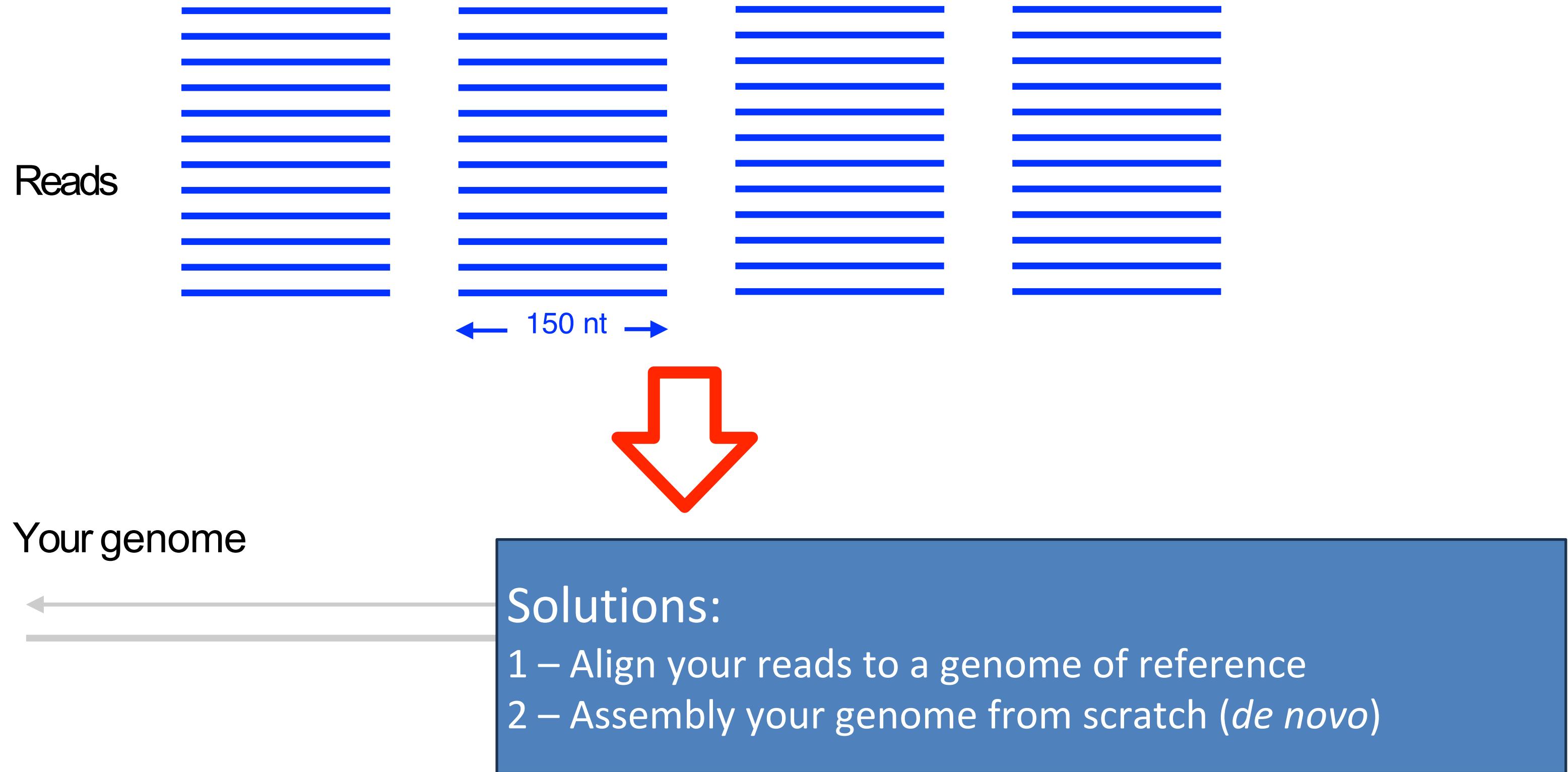
Cronograma

Data	Tema
11-Oct	Introdução a Genômica, Sequenciamento (teórica)
18-Oct	Bioinformática - Linux - Processamento de dados de sequenciamento (teórico-prática)
1-Nov	Montagem de genomas (teórico-prática)
8-Nov	Anotação de genomas (teórico-prática)
22-Nov	Analise de variabilidade genética (teórico-prática)

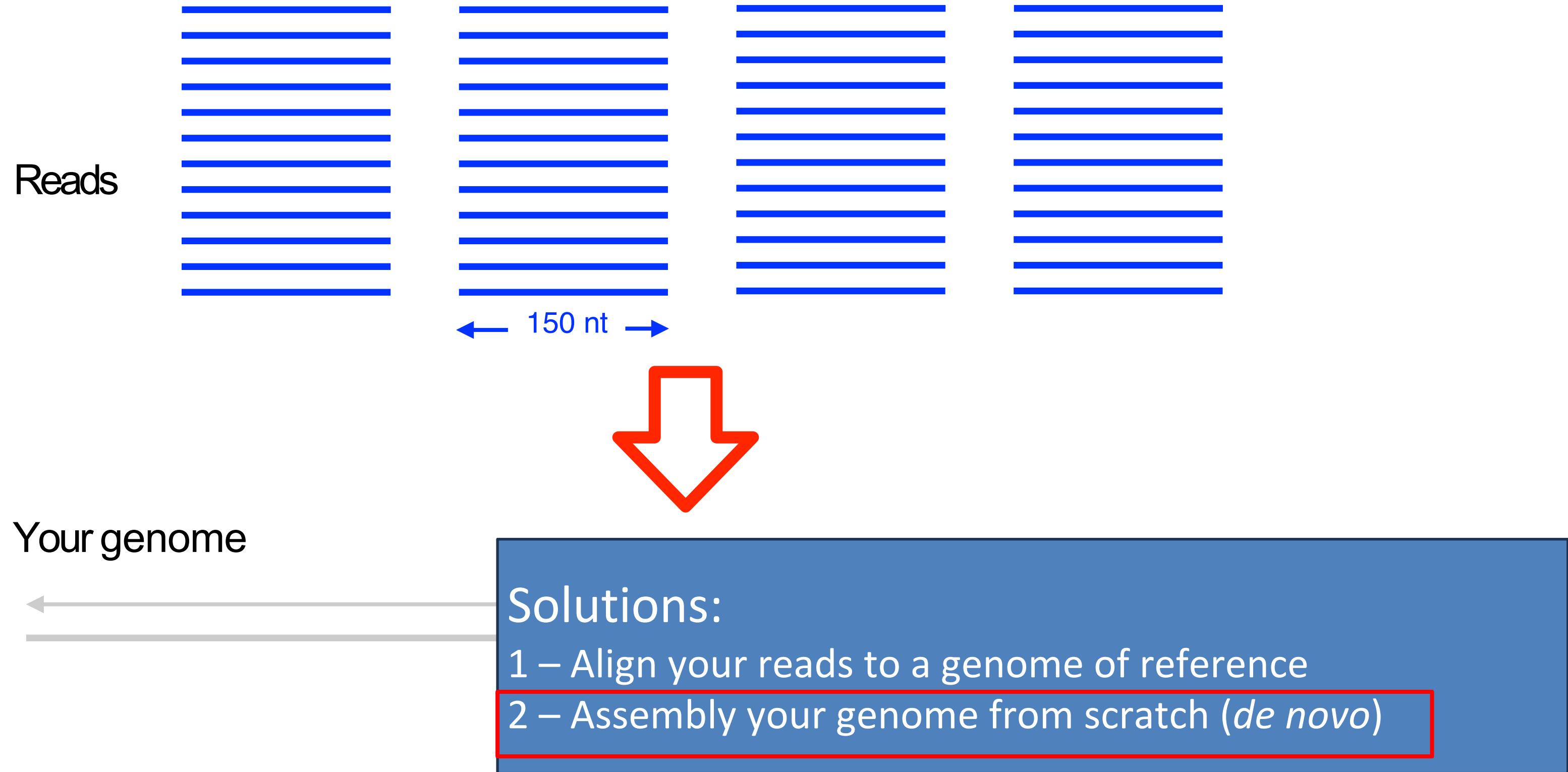
Genome sequenced, what next?



Genome sequenced, what next?



Genome sequenced, what next?



Concepts - Coverage

A vertical green bar is positioned to the left of the sequence, extending from the top blue sequence down to the bottom red sequence. The bar is approximately one-third of the way from the top of the slide to the bottom.

CTAGGCCCTCAATTTT
CTCTAGGCCCTCAATTTT
GGCTCTAGGCCCTCATTTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTT

Coverage = 5

Coverage

CTAGGCCCTCAATTTT
CTCTAGGCCCTCAATTTT
GGCTCTAGGCCCTCATTTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGCTCTAGGCCCTCATTTTT

Coverage = 5

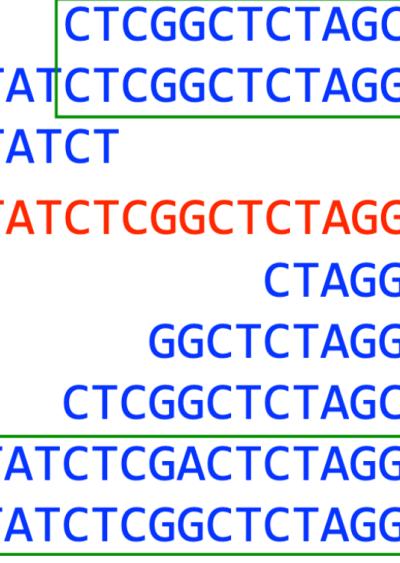


More coverage leads to more and longer overlaps

CTAGGCCCTCAATTTT
CTCGGCTCTAGCCCCTCATTTT
TCTATATCTCGGCTCTAGG
GGCGTCGATATCT
GGCGTCTATATCTCGCTCTAGGCCCTCATTTTT
CTAGGCCCTCAATTTT
GGCTCTAGGCCCTCATTTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCTATATCT

less coverage

more coverage



Overlaps

If a suffix of read A is similar to a prefix of read B...

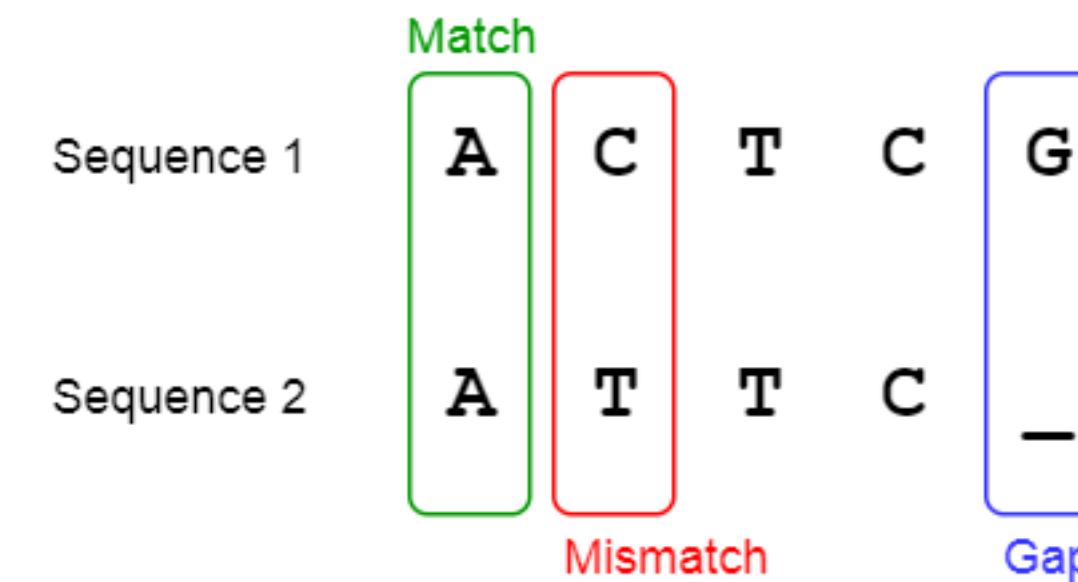
TCTATATCTCGGCTCTAGG
||||||| |||||
TATCTCGACTCTAGGCC

...then A and B might *overlap* in the genome

TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTT
TATCTCGACTCTAGGCC

Sequence alignment

	Substitution	Insertion	Deletion
Original sequence	T G G C A G	T G G C A G	T G G G A G
Mutated sequence	T G G T A G	T G G T A T C A G	T G G G



Sequence alignment

The Needleman-Wunsch Algorithm (1970)

Let's align 2 sequences

CTCGCAGC and CATTCAC

STEP 1: Initialization of score matrix.

0	0	C	T	C	G	C	A	G	C
0									
C									
A									
T									
T									
C									
A									
C									

Sequence alignment

The Needleman-Wunsch Algorithm

- Score matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5								
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

- Direction matrix:

0	0	C	T	C	G	C	A	G	C
0	0	Left							
C	Up								
A	Up								
T	Up								
T	Up								
C	Up								
A	Up								
C	Up								

Let's assume:

Match= 10

Mismatch= -2

Gap Penalty= -5

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

		SequenceB									
		C	T	C	G	C	A	G	C		
0		0	-5	-10	-15	-20	-25	-30	-35	-40	
C		-5									
A		-10									
T		-15									
T		-20									
C		-25									
A		-30									
C		-35									

Score [i, j] = max ((score [i-1, j] + gap penalty), (score [i-1, j-1] + s(ai, bj)), (score [i, j-1] + gap penalty))

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

		SequenceB									
		C	T	C	G	C	A	G	C		
SequenceA		0	0	-5	-10	-15	-20	-25	-30	-35	-40
	0										
	C	-5									
	A	-10									
	T	-15									
	T	-20									
	C	-25									
	A	-30									
	C	-35									

Score [i, j] = max ((score [i-1, j] + gap penalty), (score [i-1, j-1] + s(ai, bj)), (score [i, j-1] + gap penalty))

Up

Diag

Left

i = 1 # row

j = 1 # column

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

		SequenceB									
		C	T	C	G	C	A	G	C		
SequenceA		0	0	-5	-10	-15	-20	-25	-30	-35	-40
	0										
	C	-5									
	A	-10									
	T	-15									
	T	-20									
	C	-25									
	A	-30									
	C	-35									

$$\text{Score}[i, j] = \max ((\text{score}[i-1, j] + \text{gap penalty}), (\text{score}[i-1, j-1] + s(a_i, b_j)), (\text{score}[i, j-1] + \text{gap penalty}))$$

Up Diag Left

i = 1 # row

j = 1 # column

SequenceA(i) = SequenceA(1) = C

SequenceB(j) = SequenceB(1) = C

Here since Seq1[i]=Seq2[j]: **Match= 10**

Left = -5 + (-5) = -10 , Up = -5 + (-5) = -10 , Diag = 0 + 10 = 10

Maximum of the 3 values obtained = 10.

Therefore, the value at **Score[i][j] = 10** and **Direction [i][j] = Diag**.

Sequence alignment

The Needleman-Wunsch Algorithm

Score Matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10							
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

• Direction matrix

0	0	C	T	C	G	C	A	G	C
0	0	Left							
C	Up	Diag							
A	Up								
T	Up								
T	Up								
C	Up								
A	Up								
C	Up								

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

• Score Matrix:

		SequenceB									
		C	T	C	G	C	A	G	C		
SequenceA		0	0	-5	-10	-15	-20	-25	-30	-35	-40
	0	-5	10								
	C	-10									
	A	-15									
	T	-20									
	C	-25									
	A	-30									
	C	-35									

$$\text{Score } [i, j] = \max ((\text{score } [i-1, j] + \text{gap penalty}), (\text{score } [i-1, j-1] + s(a_i, b_j)), (\text{score } [i, j-1] + \text{gap penalty}))$$

Up Diag Left

i = 1 # row

j = 2 # column

SequenceA(i) = SequenceA(1) = C

SequenceB(j) = SequenceB(2) = T

Sequence alignment

The Needleman-Wunsch Algorithm

Scoring:
Match= 10
Mismatch= -2
Gap Penalty= -5

		SequenceB									
		C T C G C A G C									
Score Matrix:		0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40		
C	-5	10									
A	-10										
T	-15										
T	-20										
C	-25										
A	-30										
C	-35										

$$\text{Score}[i, j] = \max ((\text{score}[i-1, j] + \text{gap penalty}), (\text{score}[i-1, j-1] + s(a_i, b_j)), (\text{score}[i, j-1] + \text{gap penalty}))$$

Up Diag Left

i = 1 # row

j = 2 # column

SequenceA(i) = SequenceA(1) = C

SequenceB(j) = SequenceB(2) = T

Here since Seq1[i] is not equal to Seq2[j]:

$$\text{Left} = 10 + (-5) = 5, \text{Up} = -10 + (-5) = -15, \text{Diag} = -5 + (-2) = -7$$

Maximum of the 3 values obtained = 10.

Therefore, the value at **Score[i][j] = 5** and **Direction [i][j] = Left**.

Sequence alignment

The Needleman-Wunsch Algorithm

Score Matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5						
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

- Direction matrix

0	0	C	T	C	G	C	A	G	C
0	0	Left							
C	Up	Diag	Left						
A	Up								
T	Up								
T	Up								
C	Up								
A	Up								
C	Up								

Sequence alignment

The Needleman-Wunsch Algorithm

- Score matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5	0	-5	-2	-7
T	-20	-5	10	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

- Direction matrix:

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag	Left	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Diag	Left	Diag/left	Diag/left	Diag	Left	Left
T	Up	Up	Diag	Left	Left	Left	Up/left	Diag	Left
T	Up	Up	Diag	Diag	Diag/Left	Diag/left	Diag/left	Left/Up	Diag
C	Up	Diag	Up	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Up	Up	Diag	Diag/left/up	Diag	Left	Left
C	Up	Diag	up	Diag	Diag/Up	Diag	Diag/Up	Diag	Diag

Sequence alignment

Trace backing to find the sequence

- Score matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5	0	-5	-2	-7
T	-20	-5	10	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

start with the last cell of the table.

- Direction matrix:

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag	Left	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Diag	Left	Diag/left	Diag/left	Diag	Left	Left
T	Up	Up	Diag	Left	Left	Left	Up/left	Diag	Left
T	Up	Up	Diag	Diag	Diag/Left	Diag/left	Diag/left	Left/Up	Diag
C	Up	Diag	Up	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Up	Up	Diag	Diag/left/up	Diag	Left	Left
C	Up	Diag	up	Diag	Diag/Up	Diag	Diag/Up	Diag	Diag



Sequence alignment

The Needleman-Wunsch Algorithm

- Score matrix:

SequenceB

	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5	0	-5	-2	-7
T	-20	-5	10	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

- Direction matrix:

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag	Left	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Diag	Left	Diag/left	Diag/left	Diag	Left	Left
T	Up	Up	Diag	Left	Left	Left	Up/left	Diag	Left
T	Up	Up	Diag	Diag	Diag/Left	Diag/left	Diag/left	Left/Up	Diag
C	Up	Diag	Up	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Up	Up	Diag	Diag/left/up	Diag	Left	Left
C	Up	Diag	up	Diag	Diag/Up	Diag	Diag/Up	Diag	Diag

This is the path obtained.

Therefore, the reverse sequences obtained are:

SequenceA

A	C
-	C

SequenceB

Sequence alignment

The Needleman-Wunsch Algorithm

- Score matrix:

0	0	C	T	C	G	C	A	G	C
0	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5	0	-5	-2	-7
T	-20	-5	10	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

- Direction matrix:

0	0	C	T	C	G	C	A	G	C
0	0	Left	Left	Left	Left	Left	Left	Left	Left
C	Up	Diag	Left	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Diag	Left	Diag/left	Diag/left	Diag	Left	Left
T	Up	Up	Diag	Left	Left	Left	Up/left	Diag	Left
T	Up	Up	Diag	Diag	Diag/Left	Diag/left	Diag/left	Left/Up	Diag
C	Up	Diag	Up	Diag	Left	Diag	Left	Left	Diag
A	Up	Up	Up	Up	Diag	Diag/left/up	Diag	Left	Left
C	Up	Diag	up	Diag	Diag/Up	Diag	Diag/Up	Diag	Diag

This is the path obtained.

Therefore, the reverse sequences obtained are:

C	-	A	C	T	-	T	A	C
C	G	A	C	G	C	T	-	C

To obtain final results, we reverse the results.

Final sequences obtained are:

C	A	T	-	T	C	A	-	C
C	-	T	C	G	C	A	G	C

In the above example we considered diagonal and 2 direction in the table.

Sequence alignment

The Needleman-Wunsch Algorithm

Complexity: $O(mn)$. Since we are using 2 for loops to traverse the matrices.

Algorithm warnings:

NW dynamic programming algorithm guarantees us optimal global alignment with the parameter we supply (gap_penalty, match_award, mismatch_penalty)

However:

- Different parameters give different alignments.
- Correct alignment might not be the optimal one.
- Correct alignment might correspond only to part of global alignments.

Sequence alignment

The Smith-Waterman Algorithm (local alignment)

		Initialize the scoring matrix								
		T	G	T	T	A	C	G	G	
G	0	0	0	0	0	0	0	0	0	
	0									
	0									
	0									
	0									
	0									
	0									
	0									
	0									
	0									

Substitution matrix:

$$S(a_i, b_j) = \begin{cases} +3, & a_i = b_j \\ -3, & a_i \neq b_j \end{cases}$$

Gap penalty:

$$W_k = kW_1$$
$$W_1 = 2$$

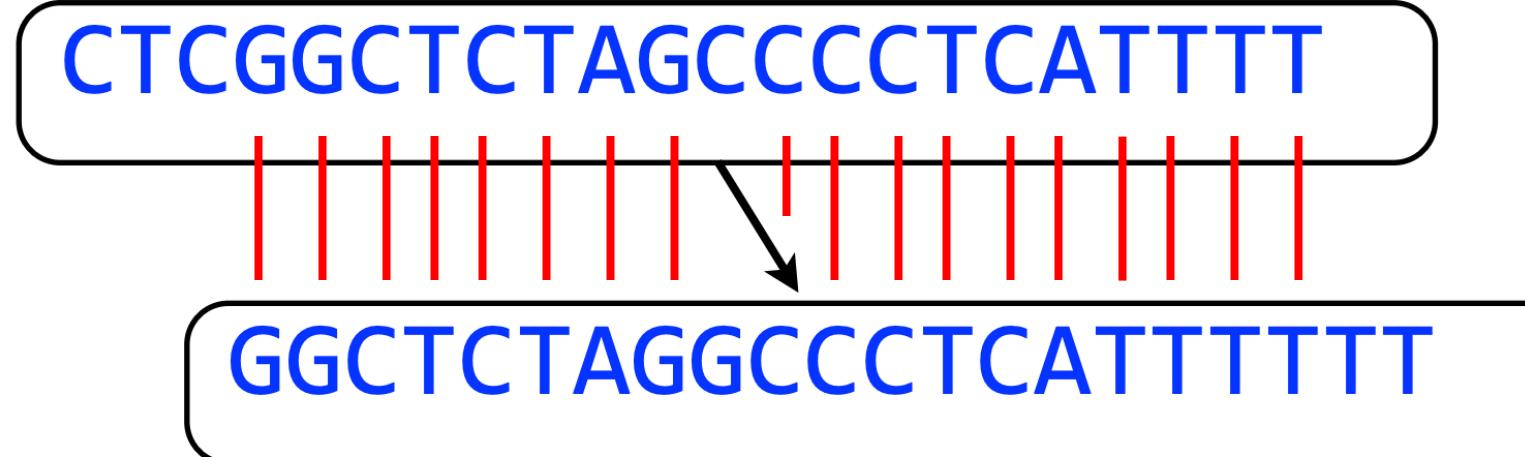
	Smith–Waterman algorithm	Needleman–Wunsch algorithm
Initialization	First row and first column are set to 0	First row and first column are subject to gap penalty
Scoring	Negative score is set to 0	Score can be negative
Traceback	Begin with the highest score, end when 0 is encountered	Begin with the cell at the lower right of the matrix, end at top left cell

Overlaps as graphs

Each node is a read

CTCGGCTCTAGCCCCCTCATTTC

Draw edge A -> B when suffix of A overlaps prefix of B



Overlaps graphs

“ k -mer” is a substring of length k

S : **GGCGATTCA**T TCG

A 4-mer of S : **ATTC**

All 3-mers of S : **GGC**

GCG

CGA

GAT

ATT

TTC

TCA

CAT

ATC

TCG

mer: from Greek meaning “part”

I'll use “ k -1-mer” to refer to a substring of length $k - 1$

Overlaps graphs

Nodes: all 6-mers from **GTACGTACGAT**

GTACGTACGA

GTACGT

TACGTA

ACGTAC

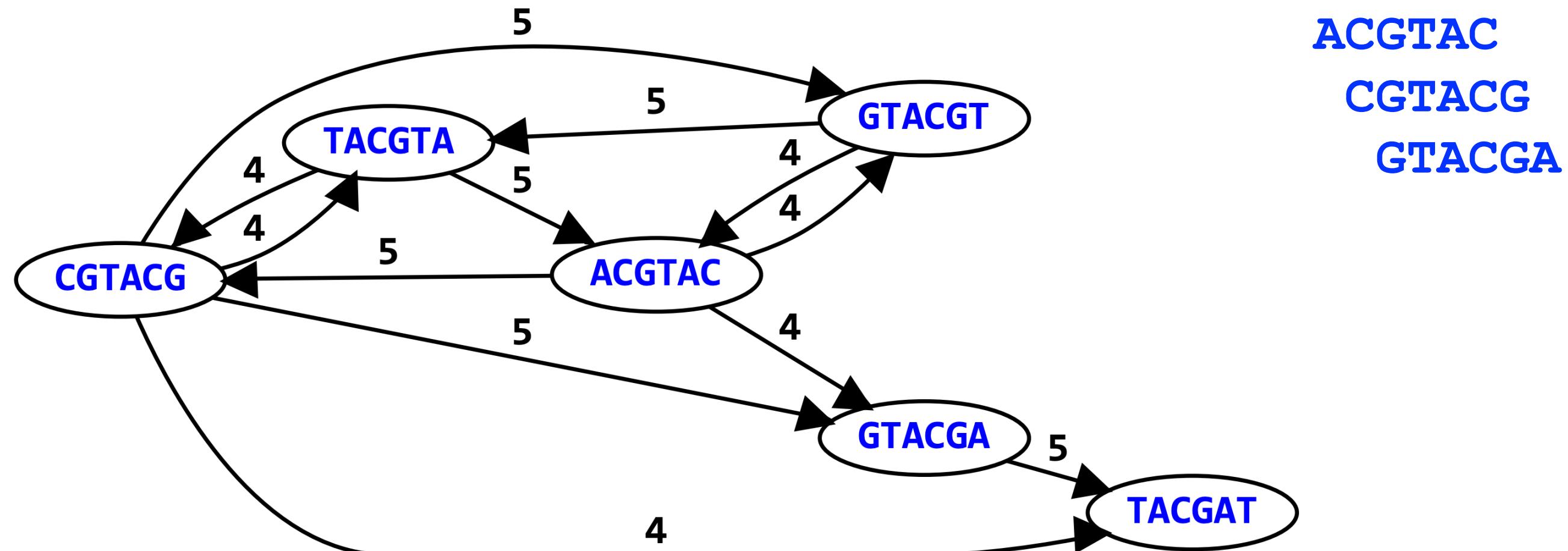
CGTACG

GTACGA

Overlaps graphs

Nodes: all 6-mers from **GTACGTACGAT**

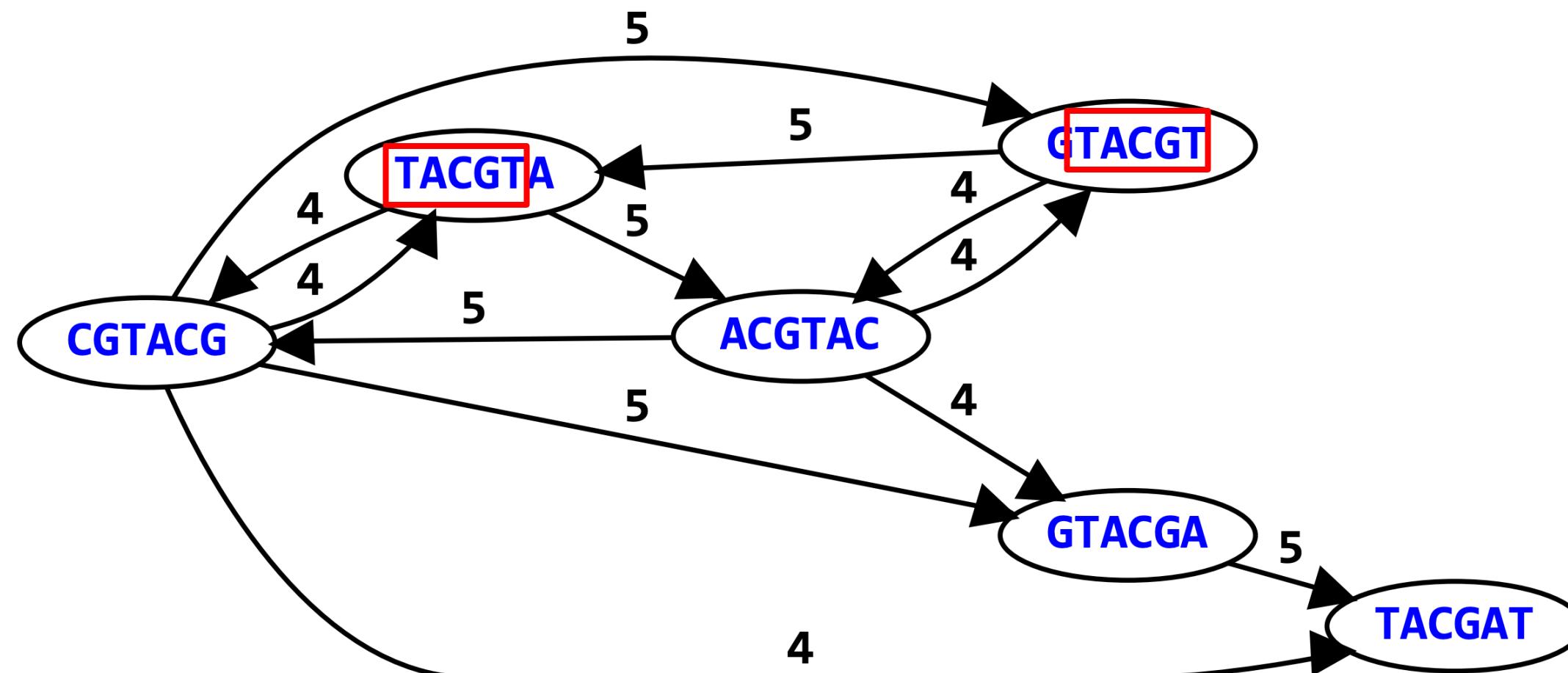
Edges: overlaps of length ≥ 4



Overlaps graphs

Nodes: all 6-mers from **GTACGTACGAT**

Edges: overlaps of length ≥ 4

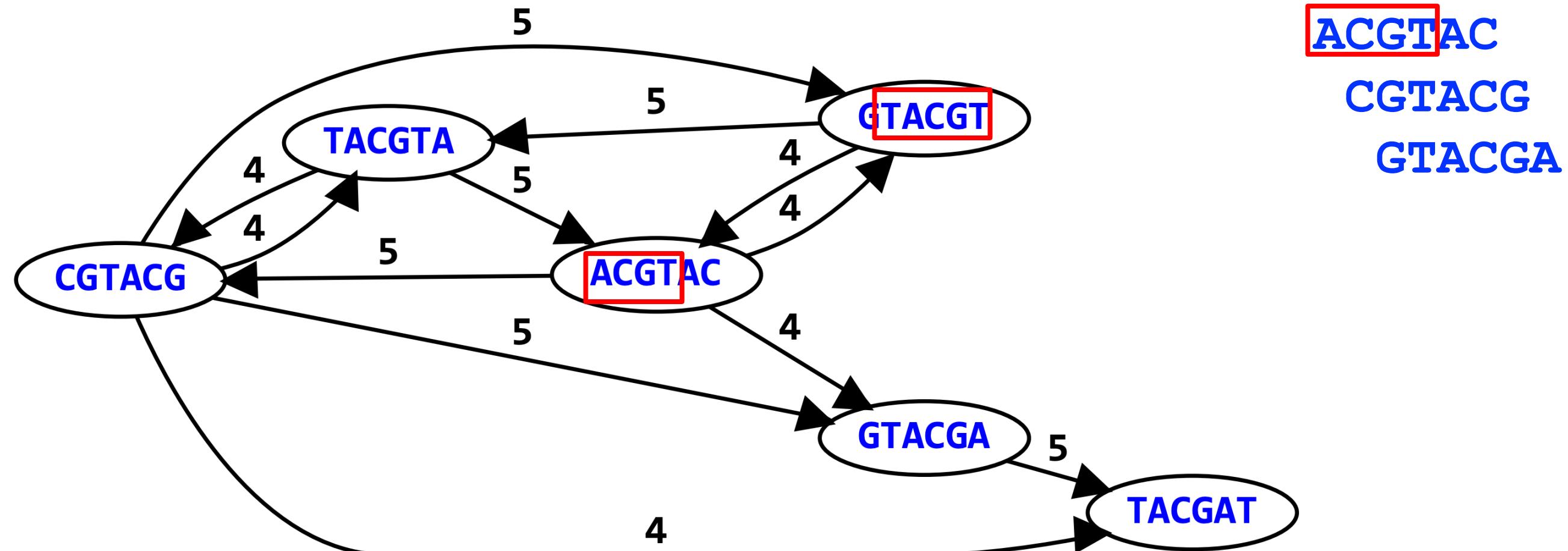


GTACGTACGA
GTACGT
TACGTA
ACGTAC
CGTACG
GTACGA

Overlaps graphs

Nodes: all 6-mers from **GTACGTACGAT**

Edges: overlaps of length ≥ 4

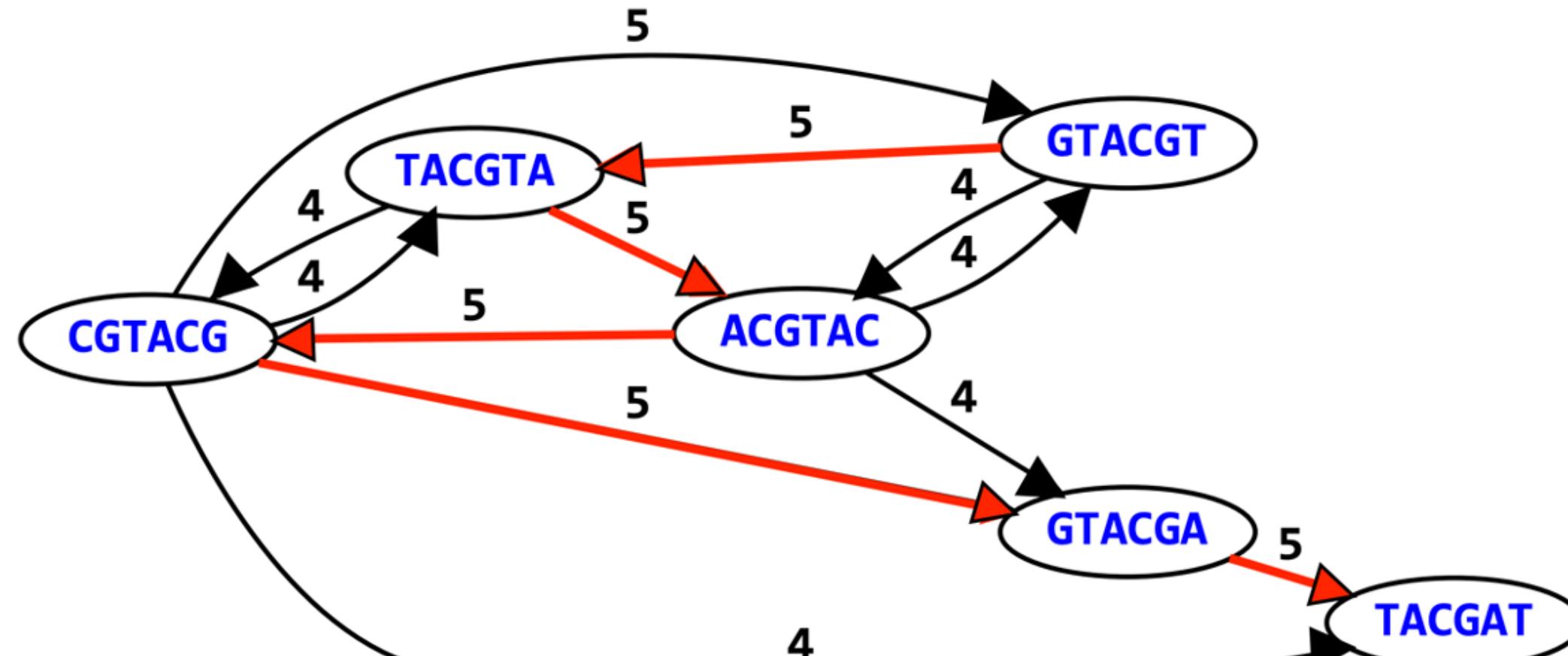


Overlaps graphs

Nodes: all 6-mers from **GTACGTACGAT**

Edges: overlaps of length ≥ 4

GTACGTACGA
GTACGT
TACGTA
ACGTAC
CGTACG
GTACGA



De Bruijn graph

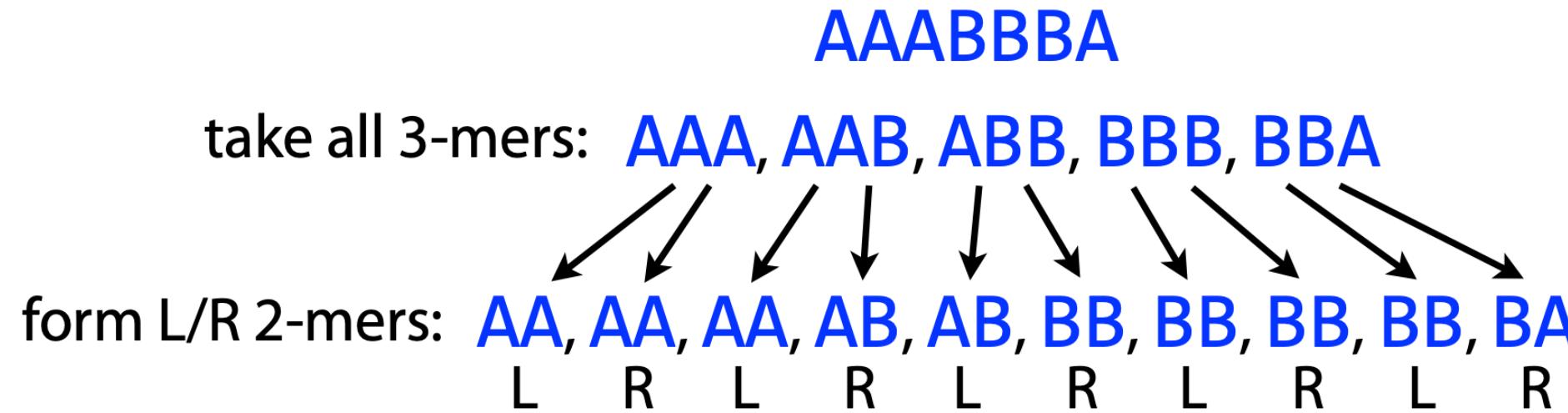
Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

AAABBBBA

take all 3-mers: AAA, AAB, ABB, BBB, BBA

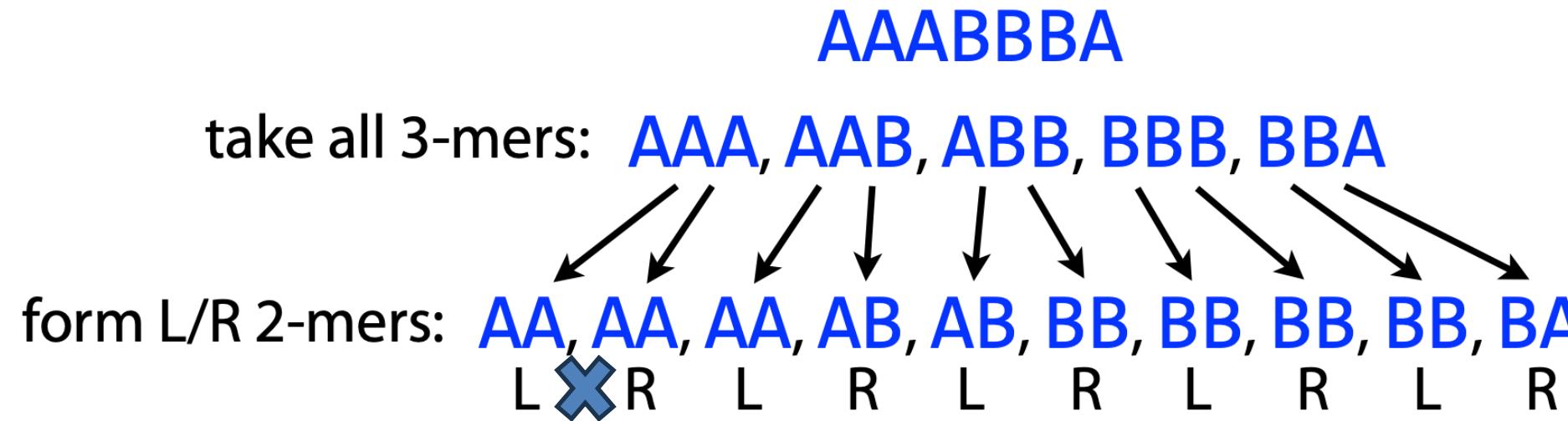
De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

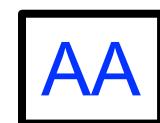


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

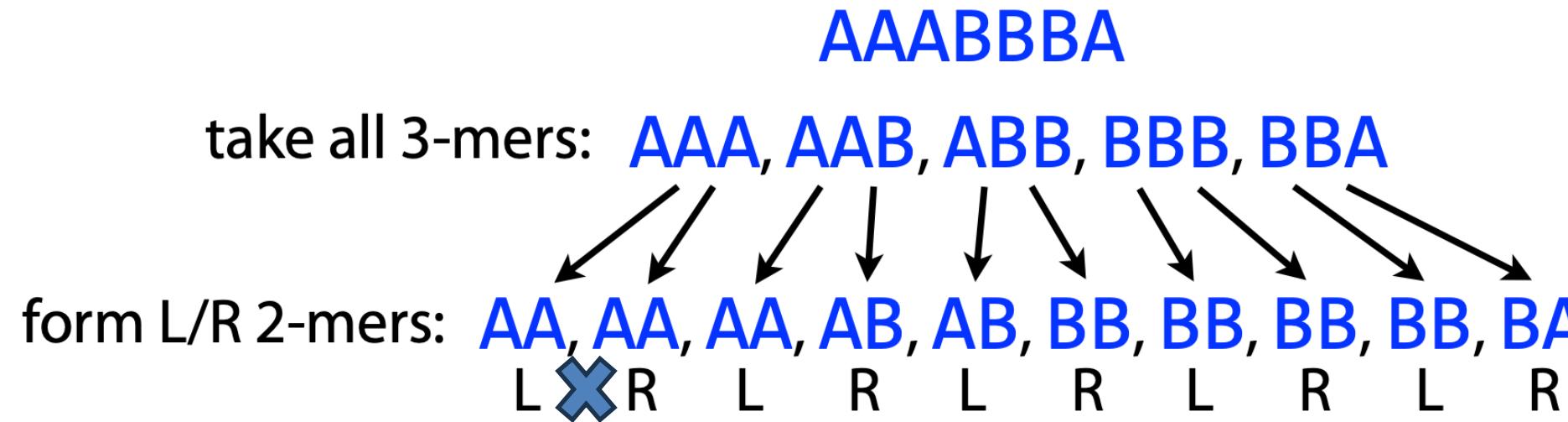


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

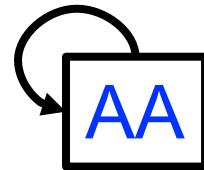


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

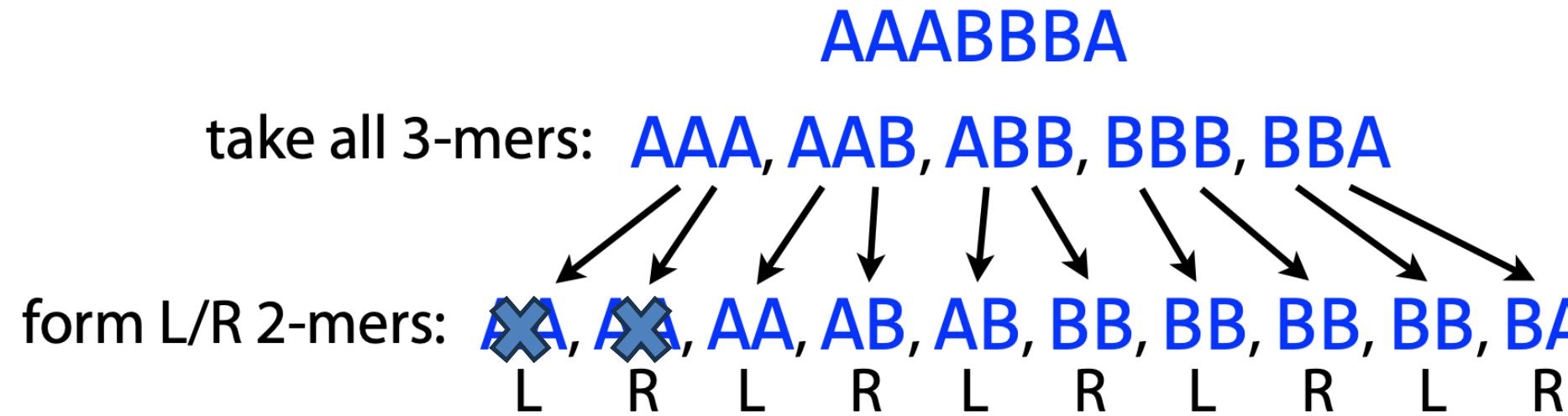


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

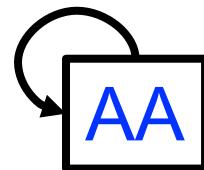


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

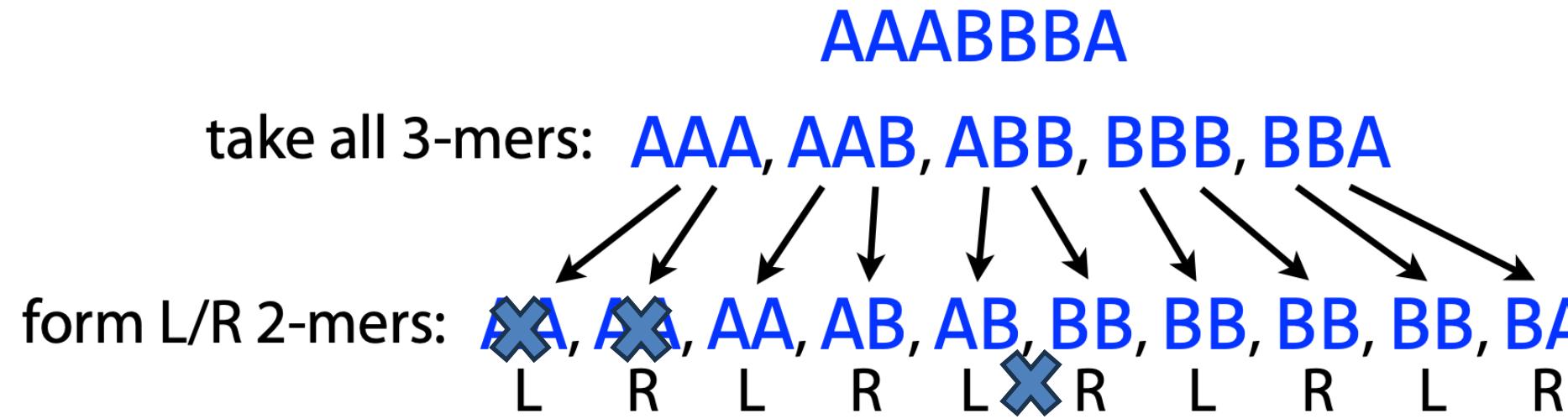


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

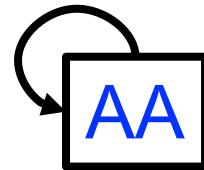


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

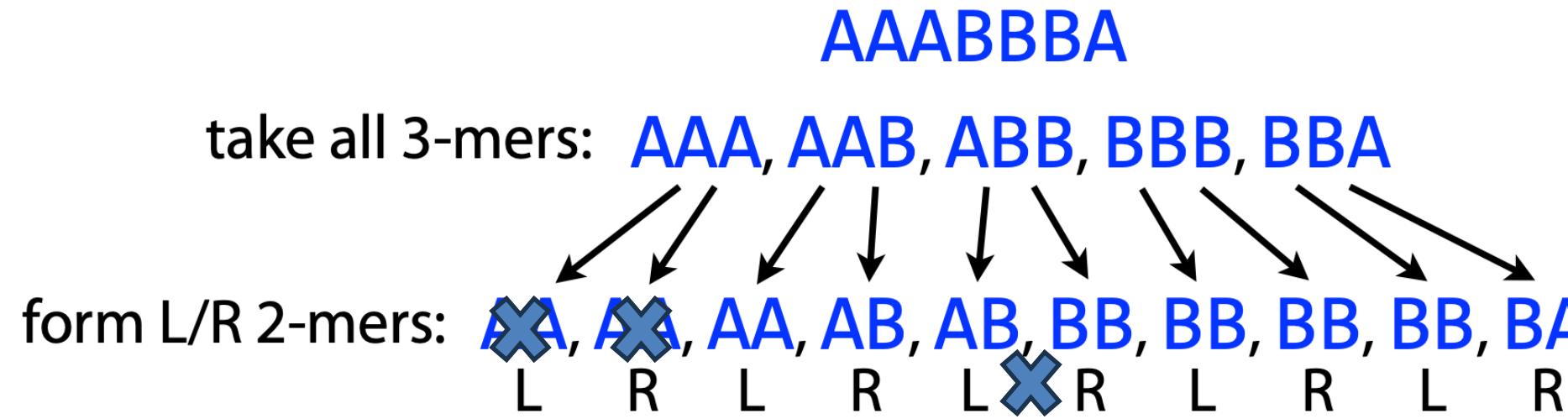


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

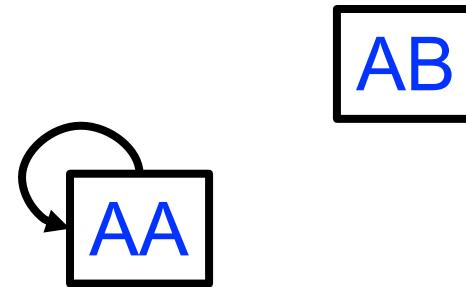


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

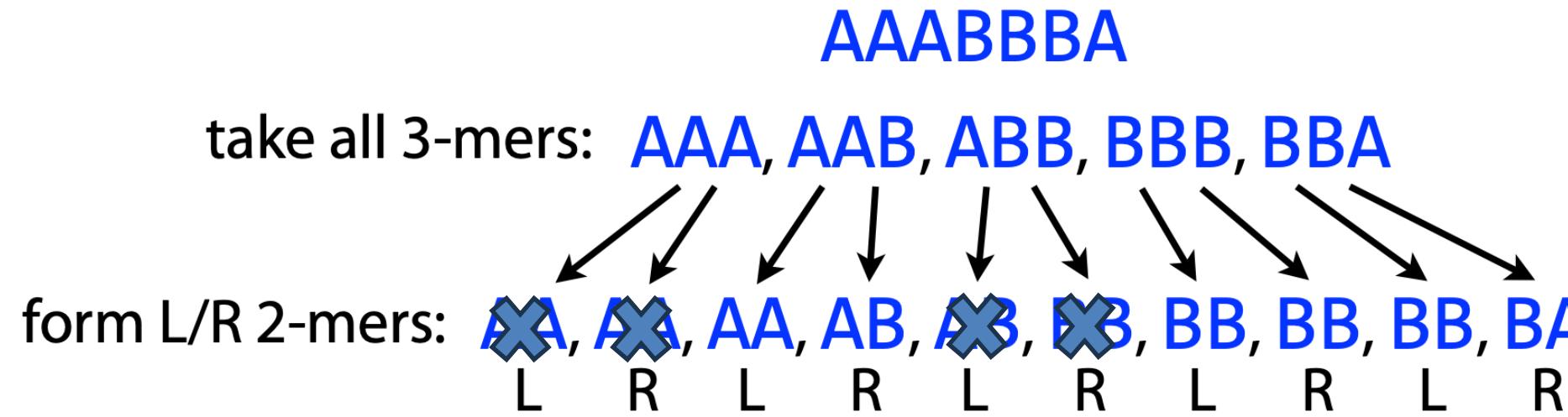


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

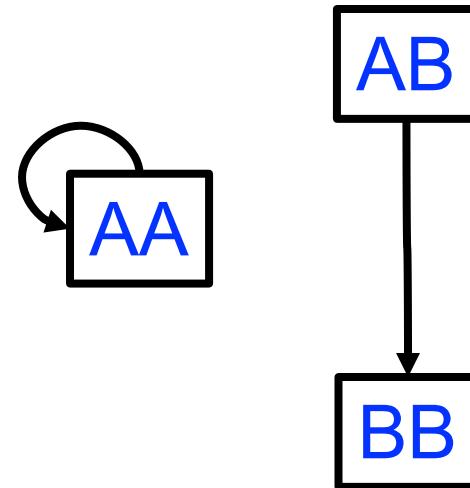


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

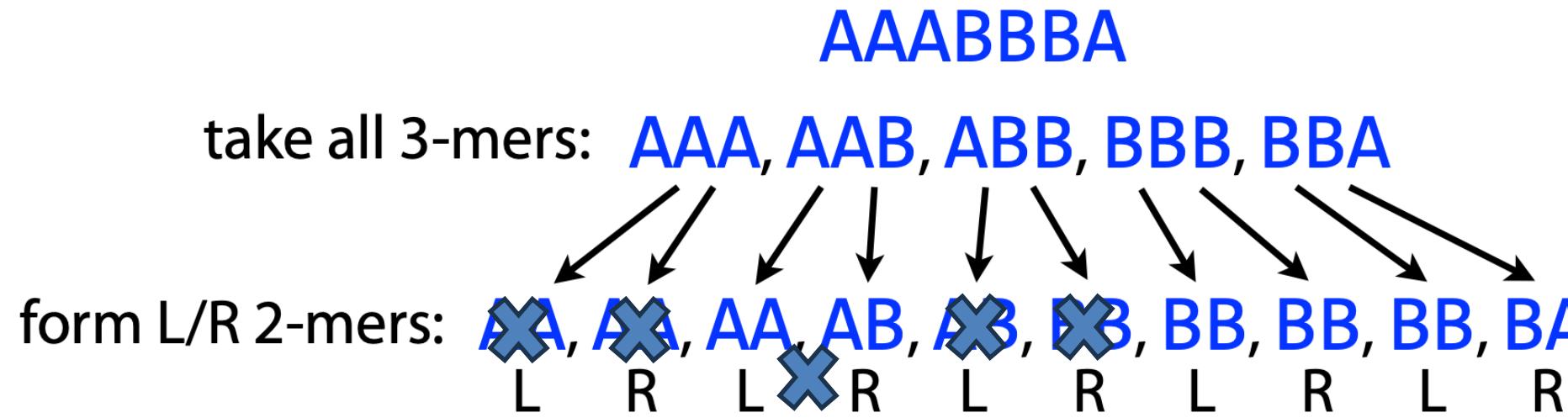


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

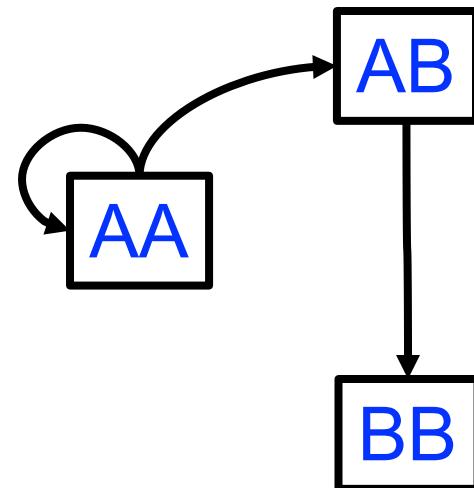


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

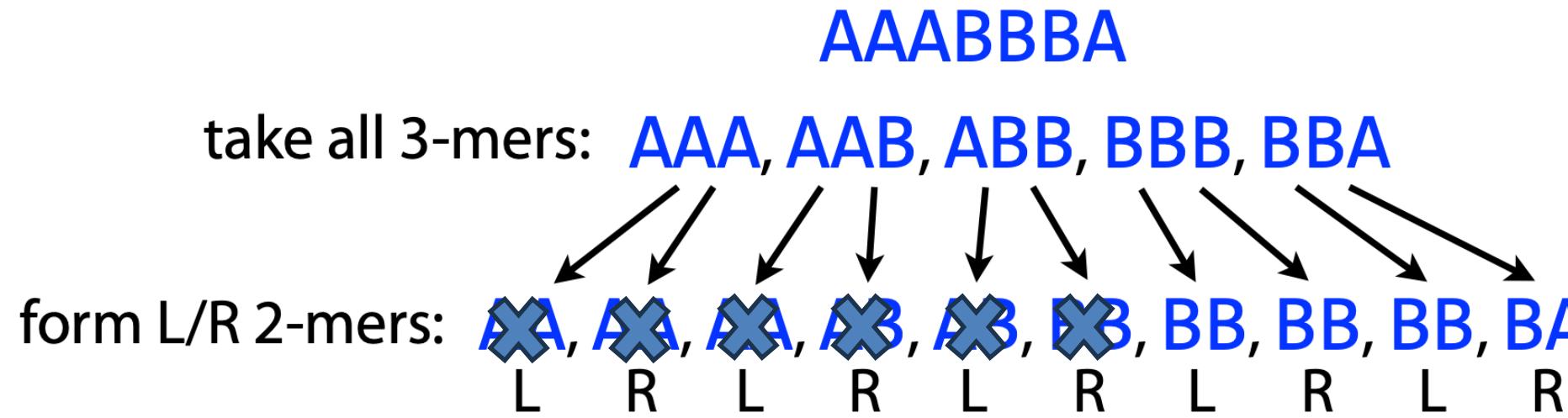


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

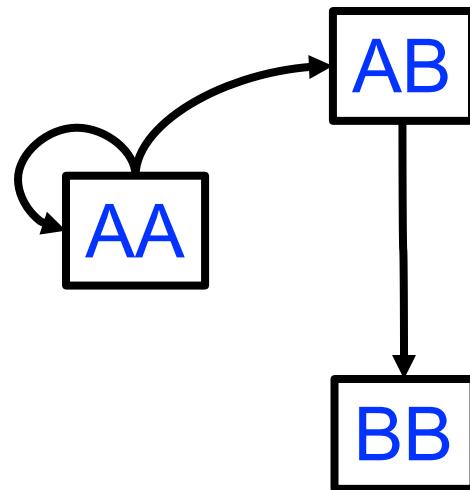


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

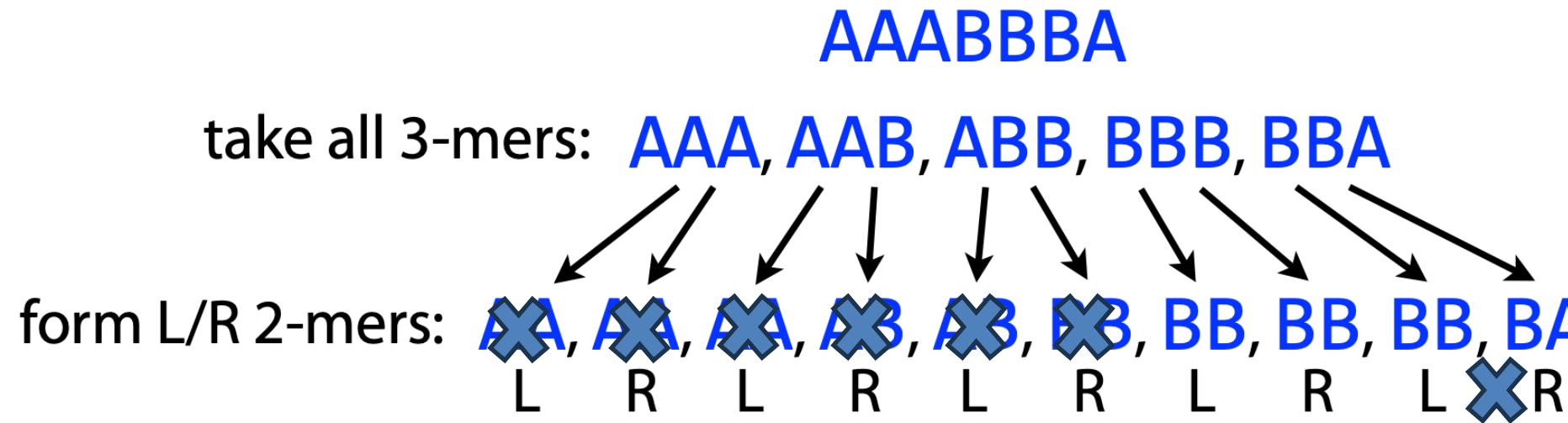


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

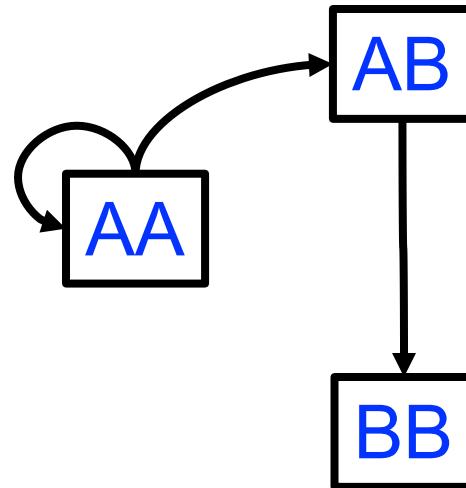


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

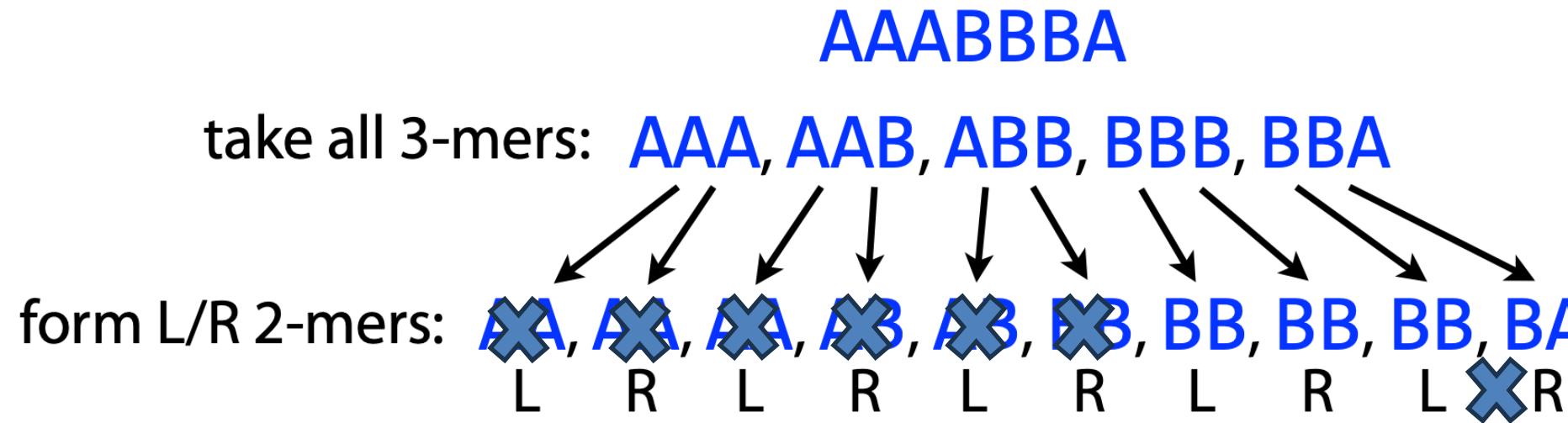


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

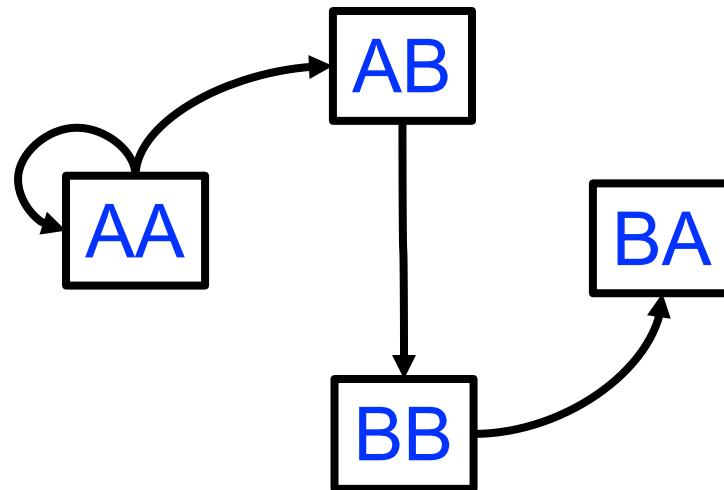


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

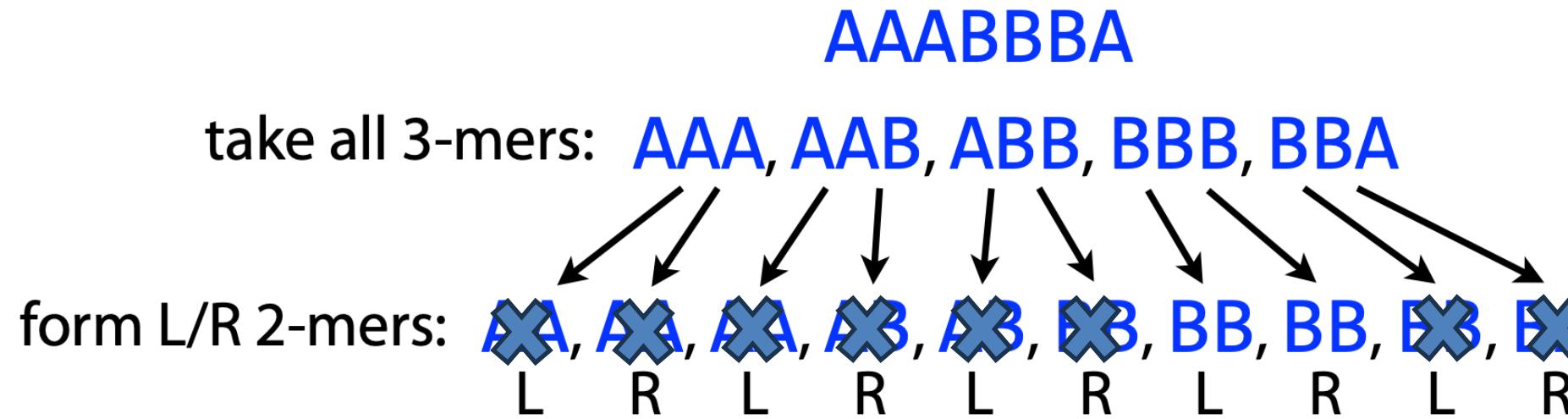


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

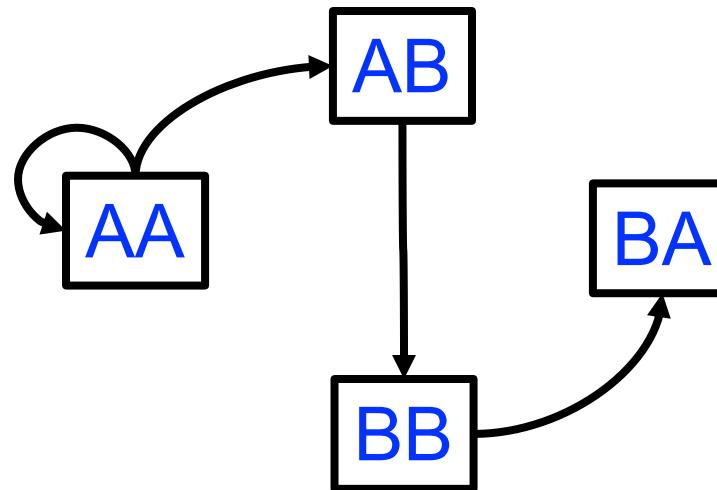


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

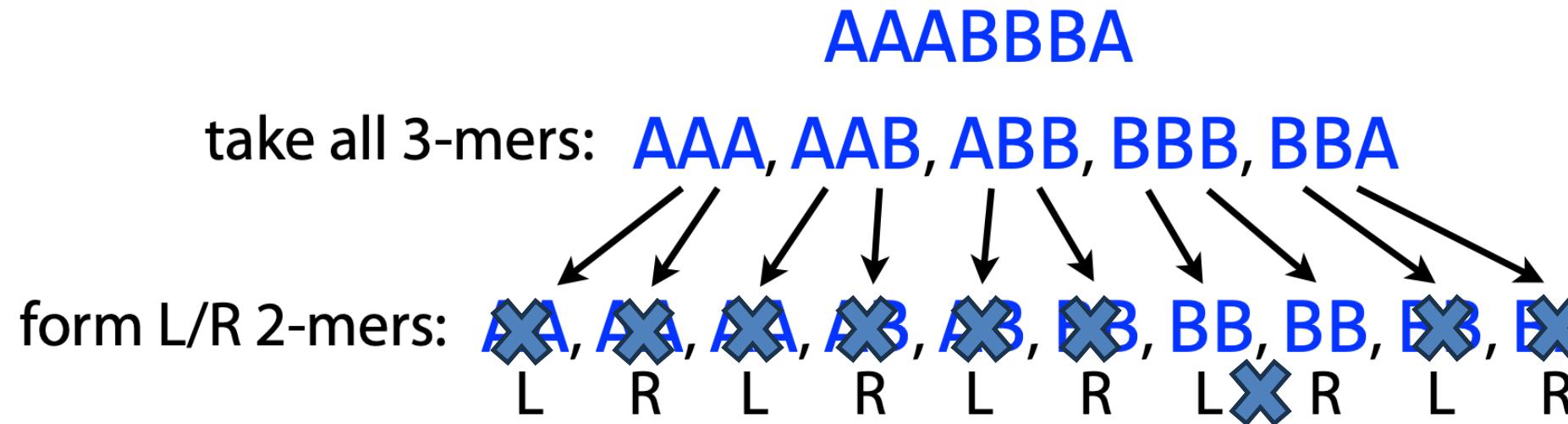


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

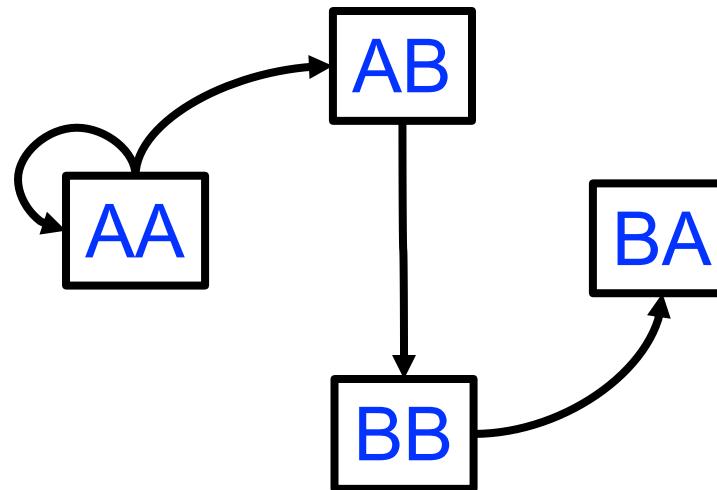


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

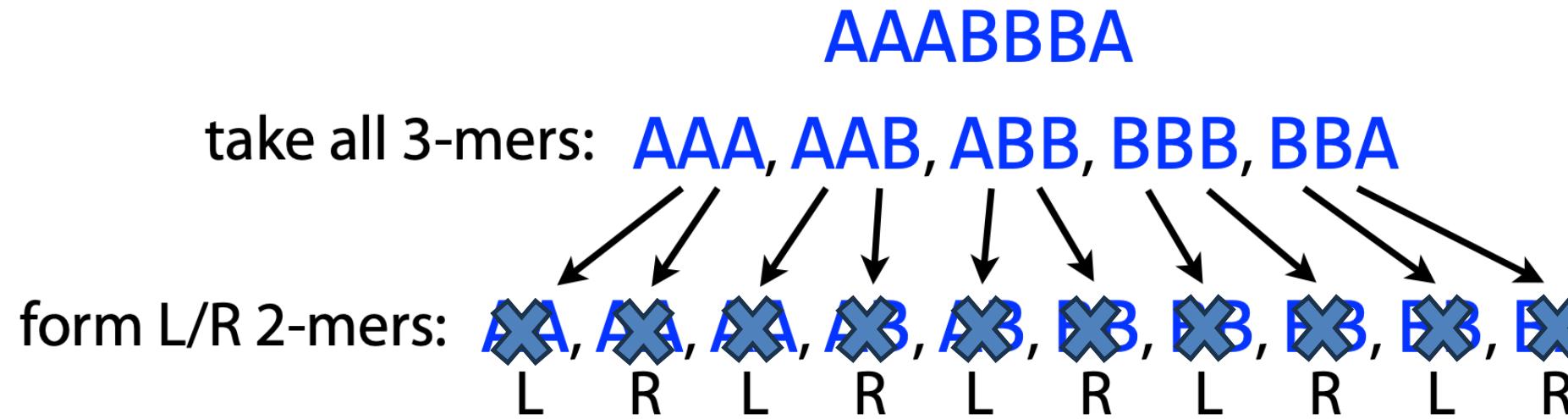


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

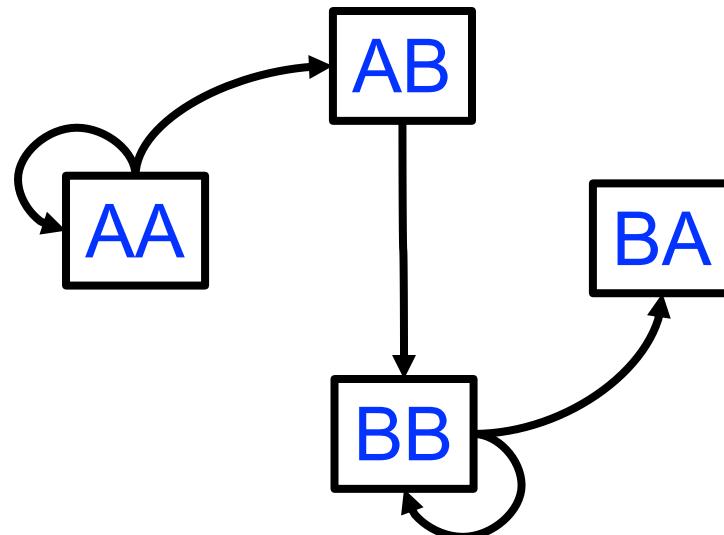


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

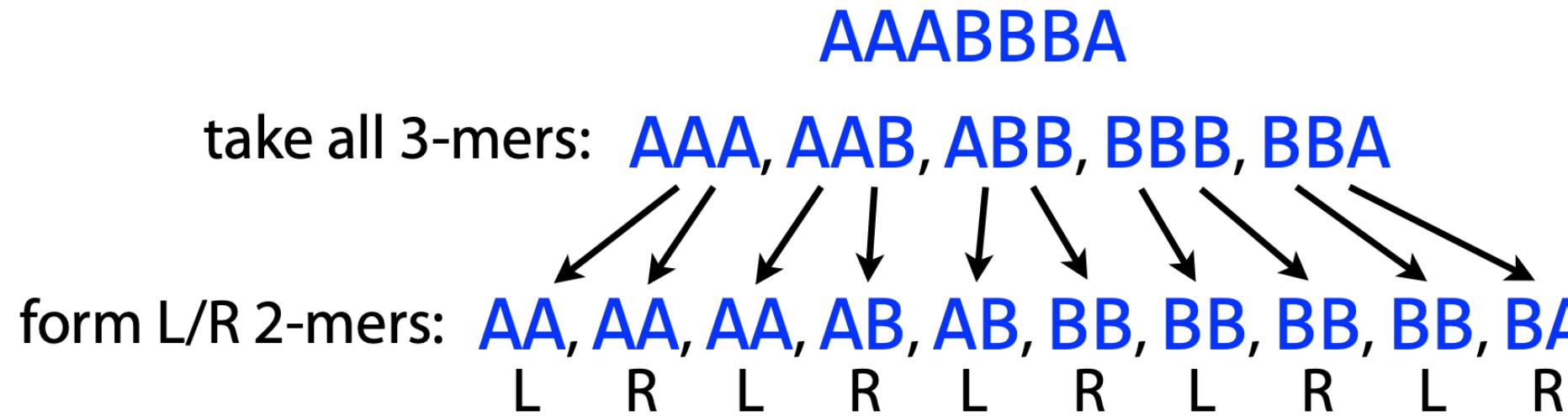


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

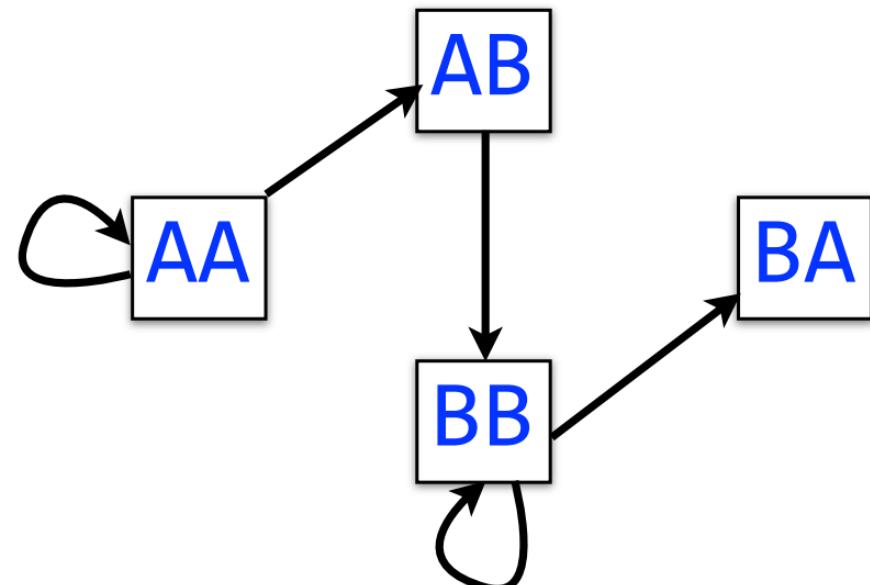


De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right* 2-mers.

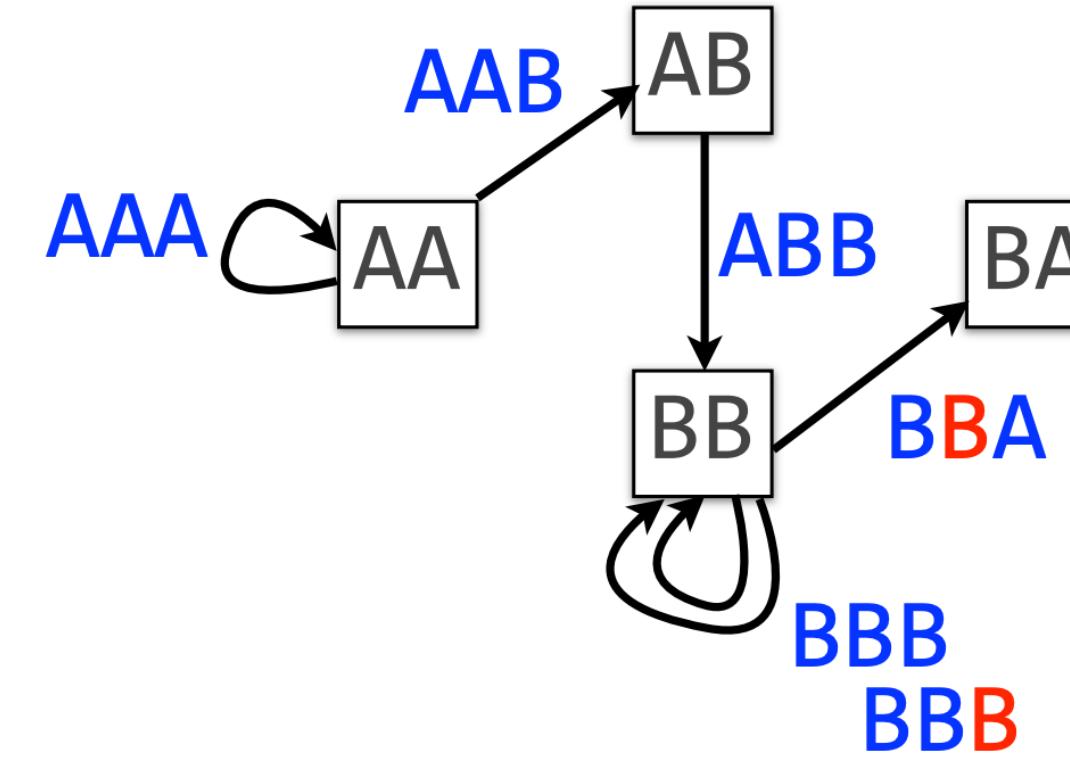


Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:



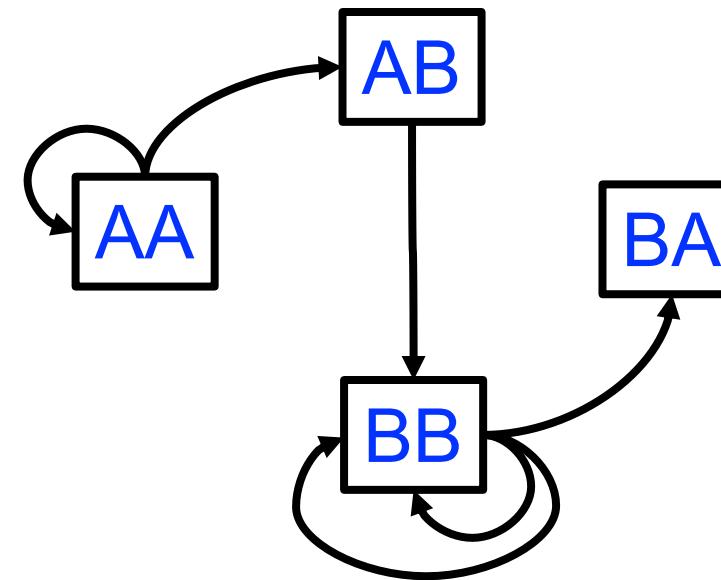
Each *edge* in this graph corresponds to a length-3 input string

De Bruijn graph



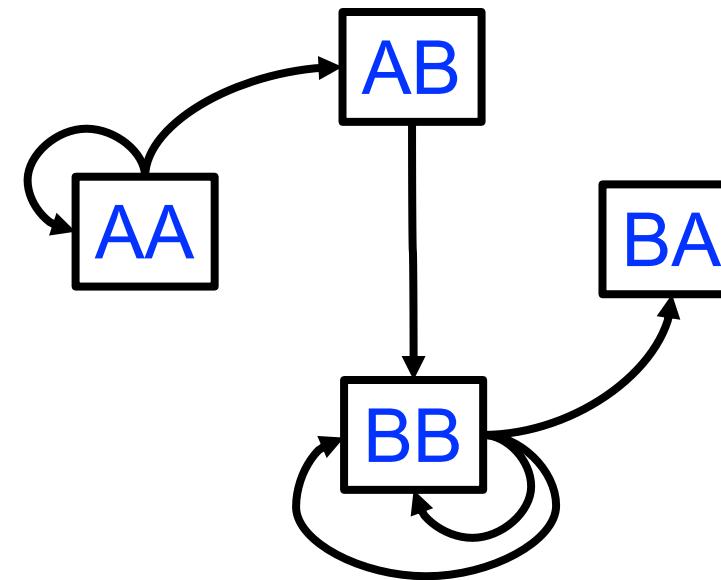
If we add one more B to our input string: **AAABBBA**, and rebuild the De Bruijn graph accordingly, we get a *multiedge*.

De Bruijn graph



Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

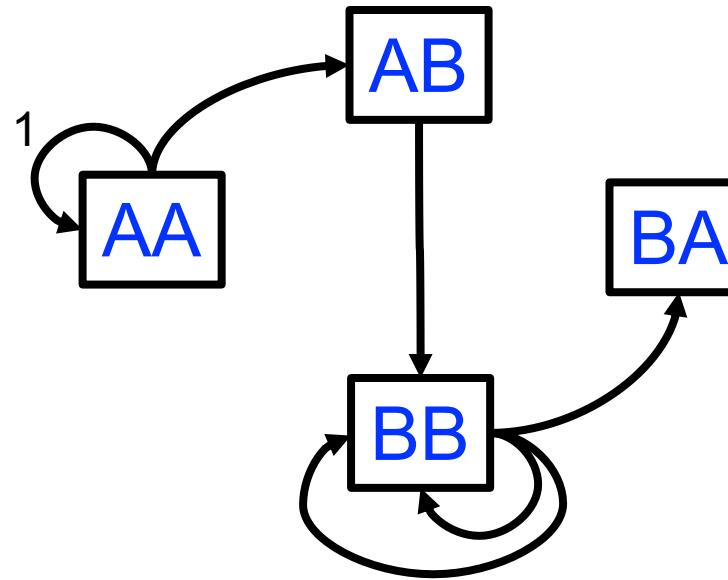
De Bruijn graph



AA

Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

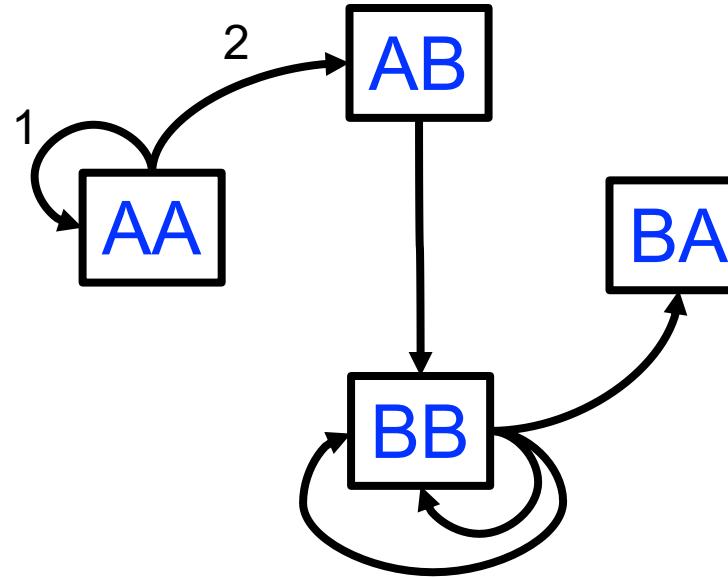
De Bruijn graph



AAA

Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

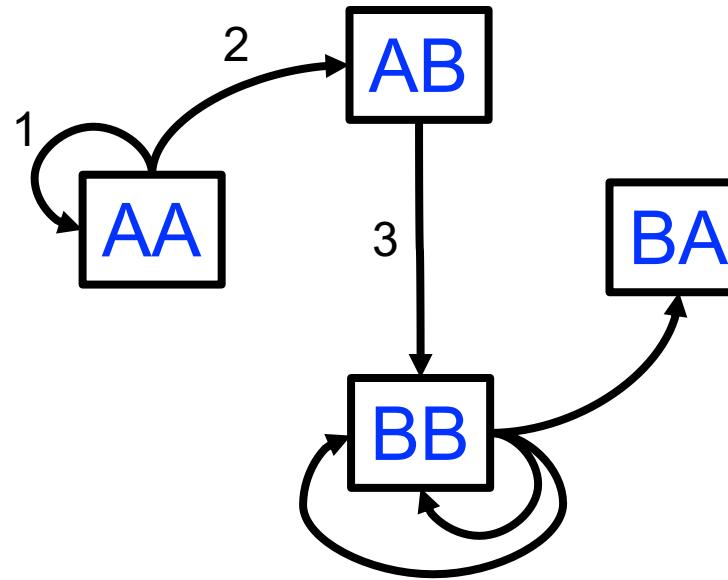
De Bruijn graph



AAAB

Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

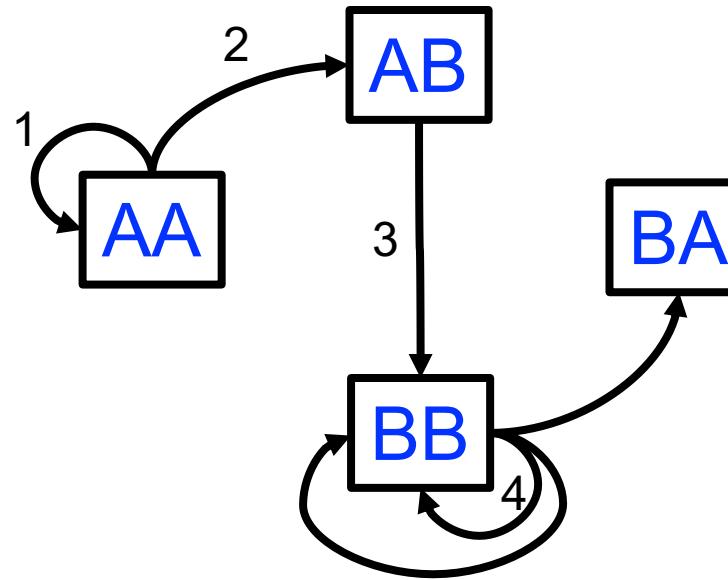
De Bruijn graph



AAABB

Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

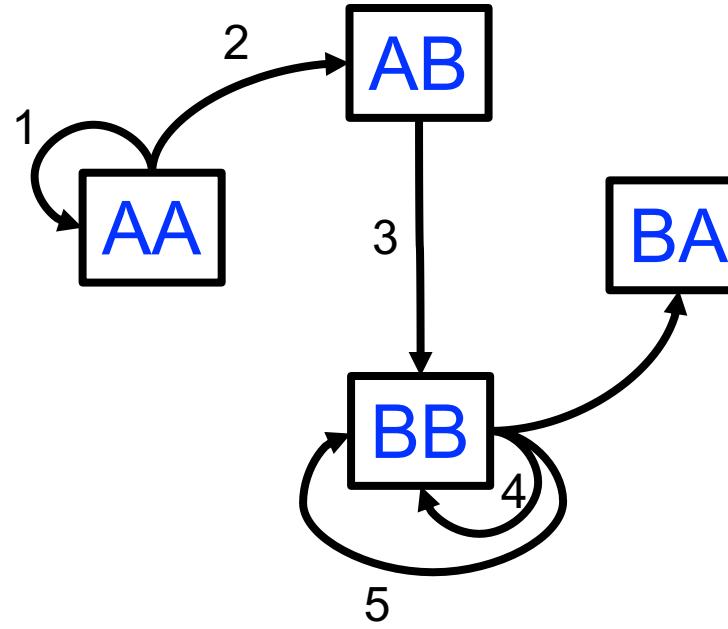
De Bruijn graph



AAABBB

Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

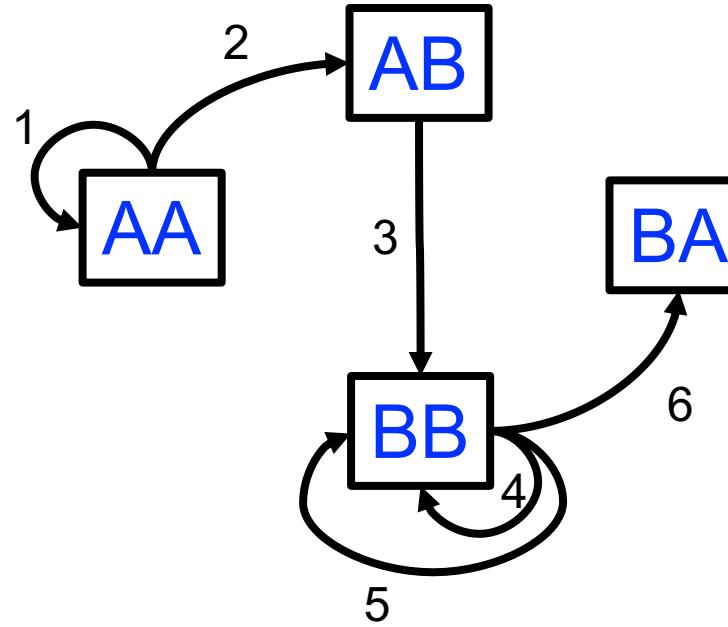
De Bruijn graph



AAABBBB

Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

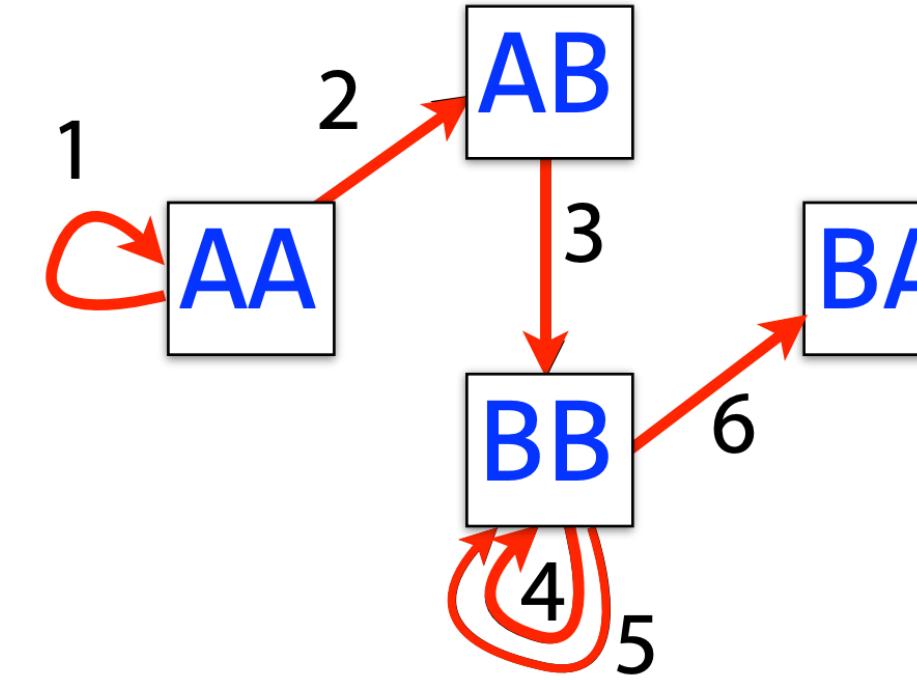
De Bruijn graph



AAABBBBA

Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

De Bruijn graph



AAABBBBA

Walk crossing each edge exactly once gives a reconstruction of the genome. This is an *Eulerian walk*.

De Bruijn graph

A procedure for making a De Bruijn graph
for a genome

Assume *perfect sequencing* where each length- k
substring is sequenced exactly once with no errors

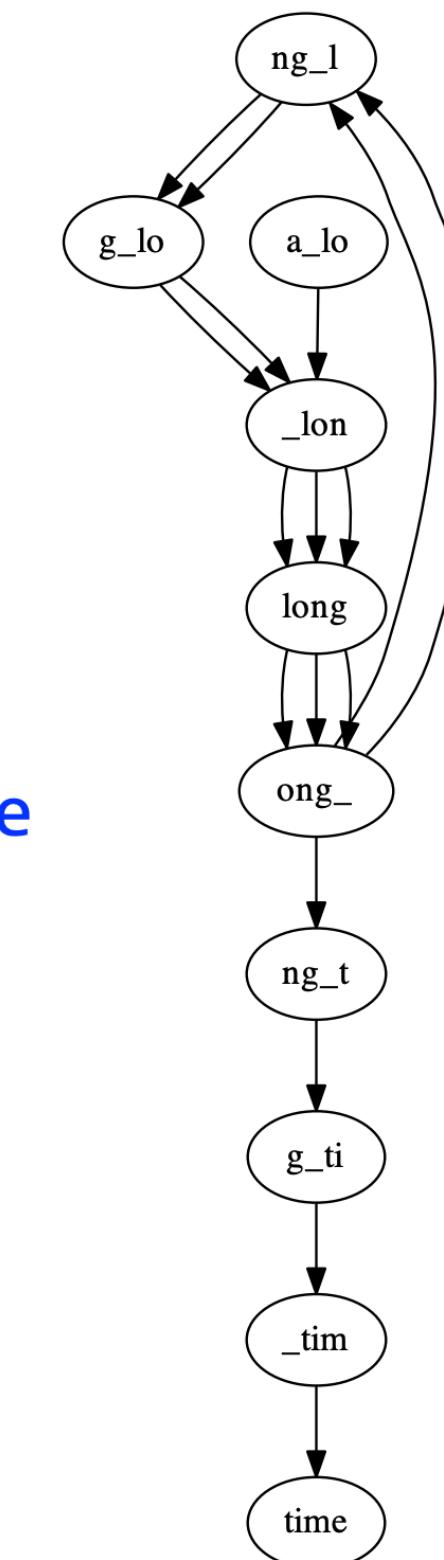
Pick a substring length k : 5

Start with an input string: a_long_long_long_time

Take each k mer and split
into left and right $k-1$ mers

long
long ong_

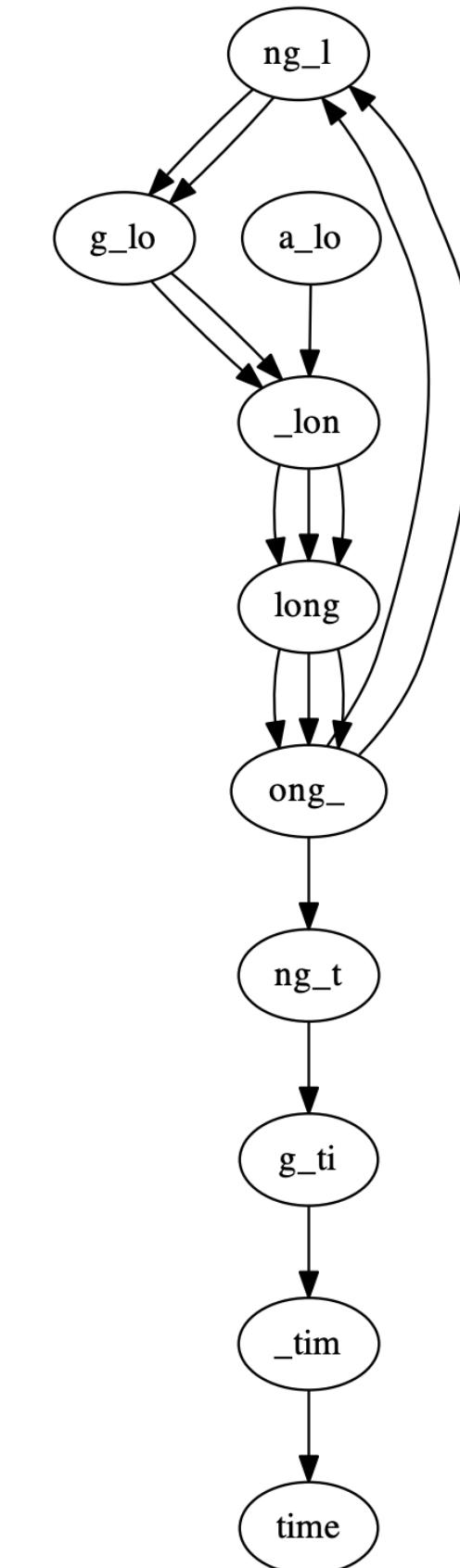
Add $k-1$ mers as nodes to De Bruijn graph
(if not already there), add edge from left $k-1$
mer to right $k-1$ mer



De Bruijn graph

Assuming perfect sequencing, procedure yields graph with Eulerian walk that can be found efficiently.

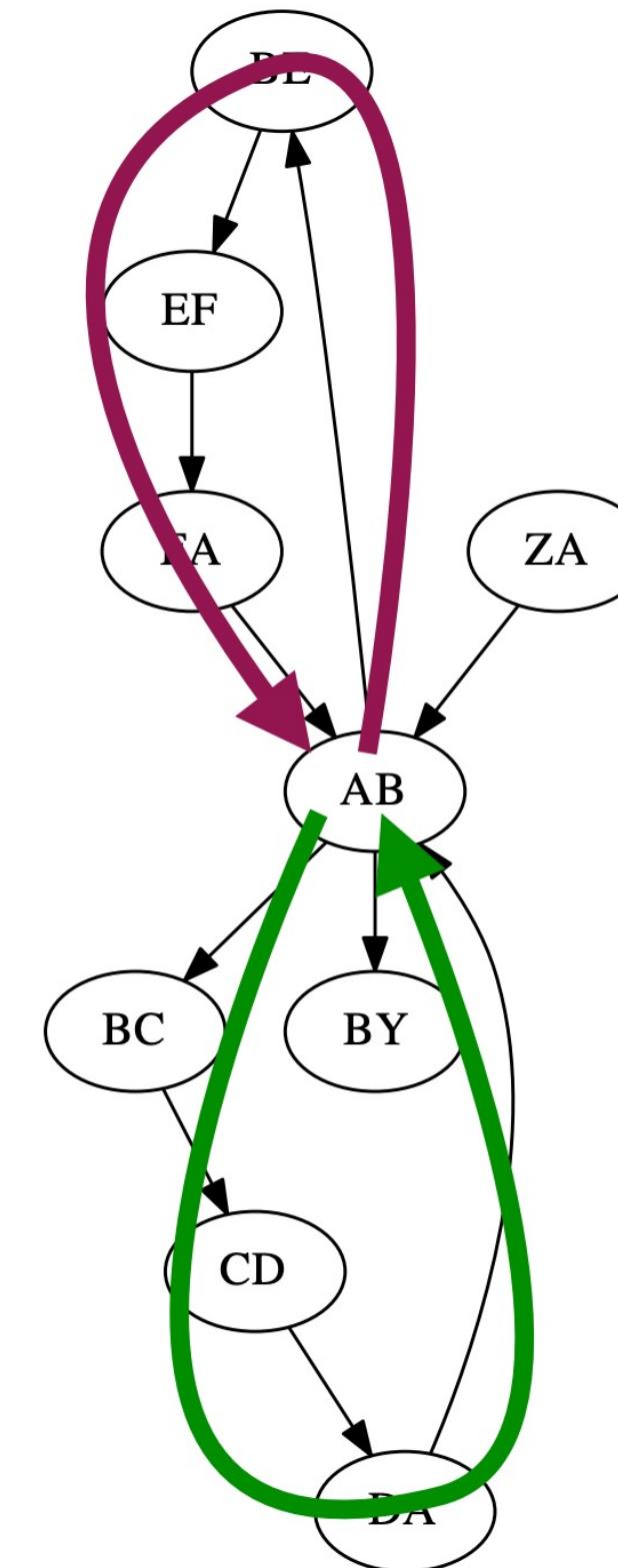
We saw cases where Eulerian walk corresponds to the original superstring. Is this always the case?



De Bruijn graph

No: graph can have multiple Eulerian walks, only one of which corresponds to original superstring

Right: graph for **ZABCDEFABY**, $k = 3$



De Bruijn graph

No: graph can have multiple Eulerian walks, only one of which corresponds to original superstring

Right: graph for **ZABCDABEFA_BY**, $k = 3$

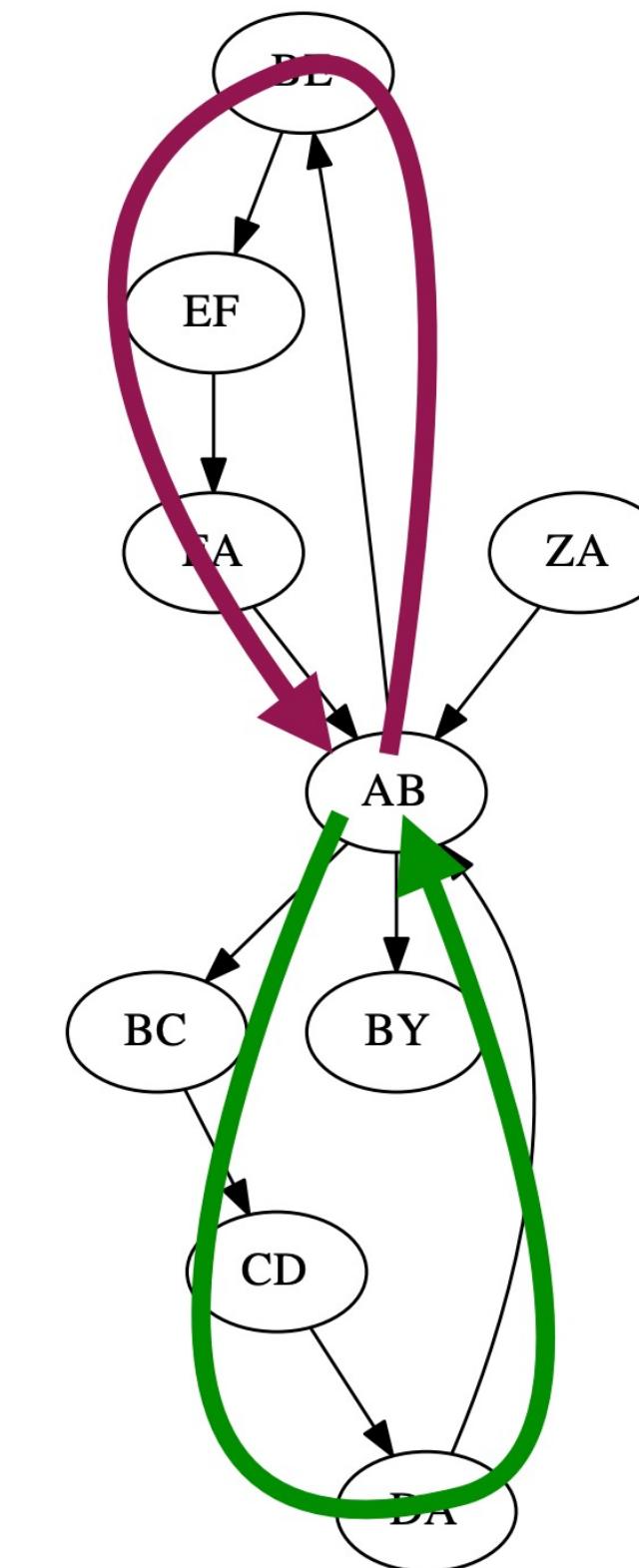
Alternative Eulerian walks:

ZA → **AB** → **BE** → **EF** → **FA** → **AB** → **BC** → **CD** → **DA** → **AB** → **BY**

ZA → **AB** → **BC** → **CD** → **DA** → **AB** → **BE** → **EF** → **FA** → **AB** → **BY**

These correspond to two edge-disjoint directed cycles joined by node **AB**

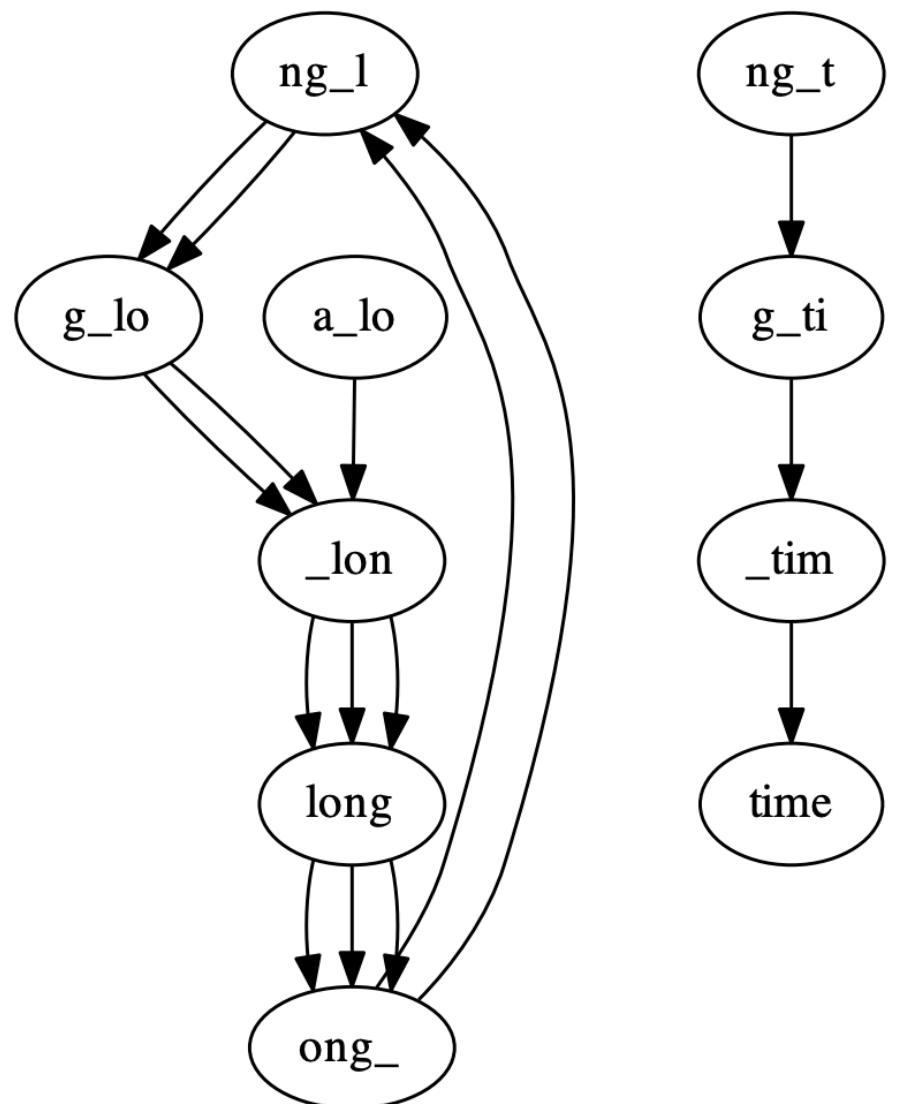
AB is a repeat: **ZABCD**A**BEFA_BY**



De Bruijn graph

Gaps in coverage can lead to *disconnected* graph

Graph for `a_long_long_long_time`, $k=5$ but *omitting* `ong_t`:



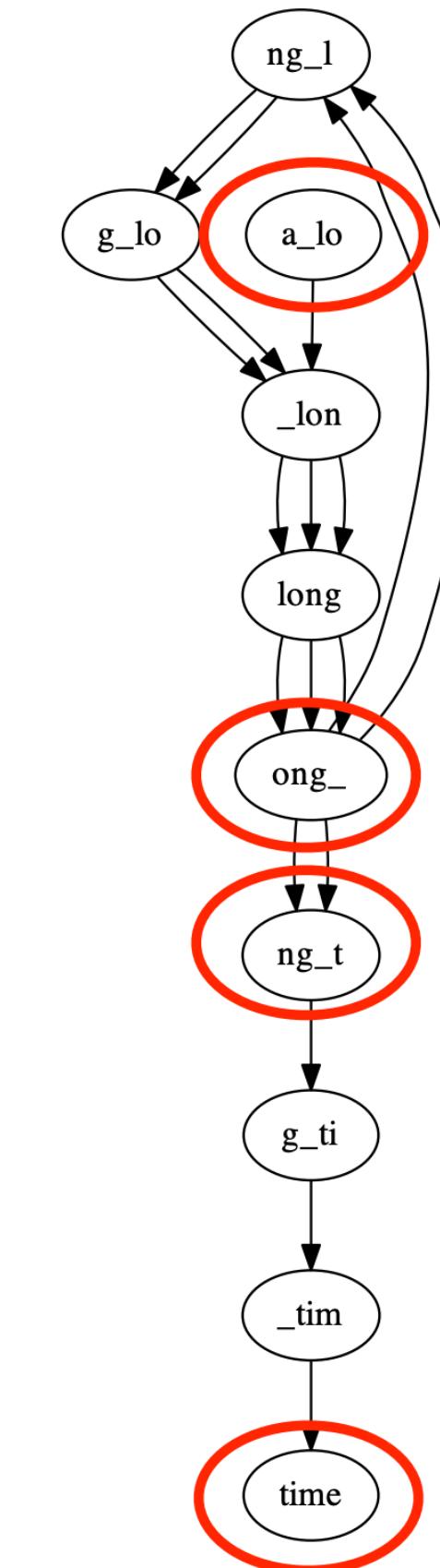
Connected components are individually Eulerian, overall graph is not

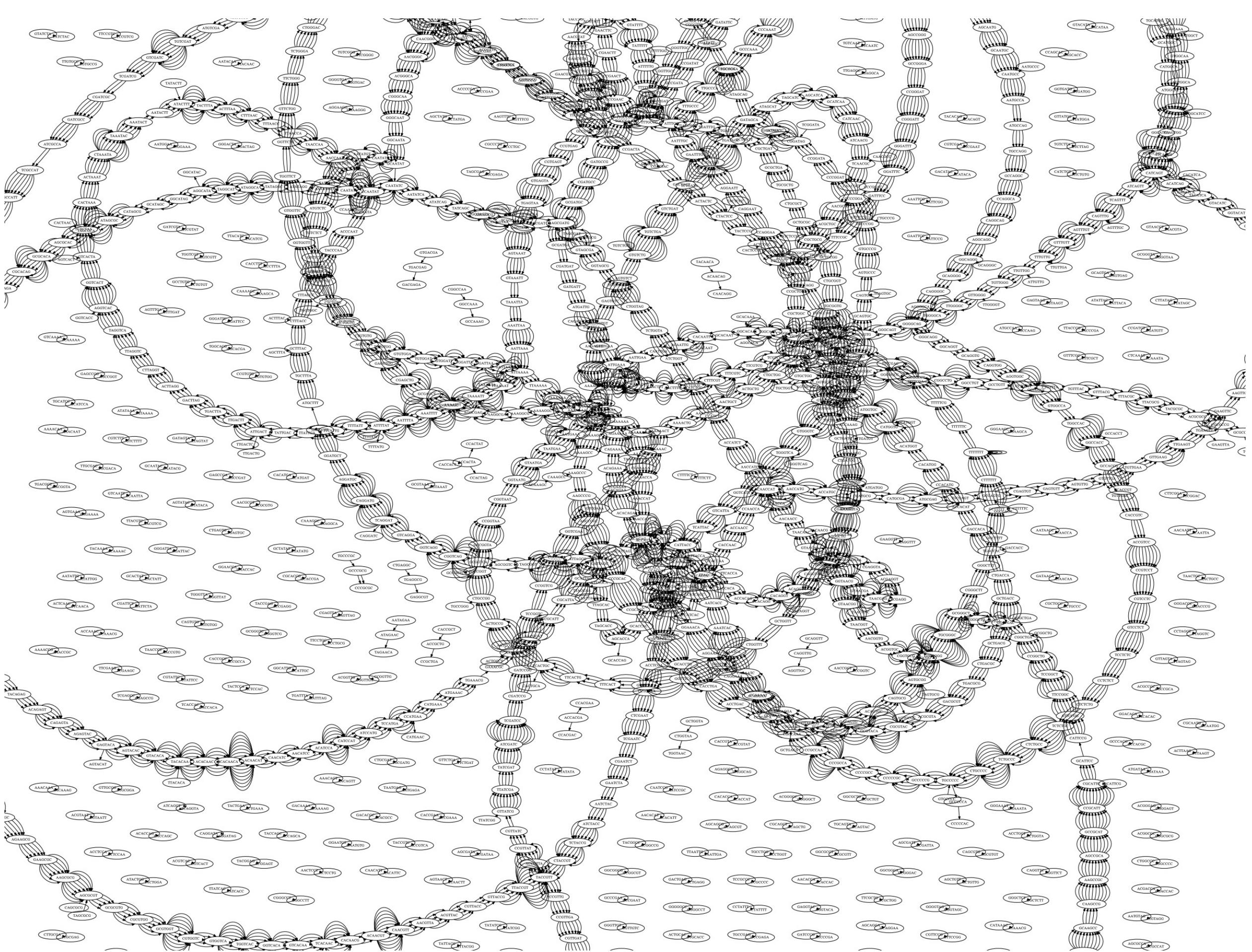
De Bruijn graph

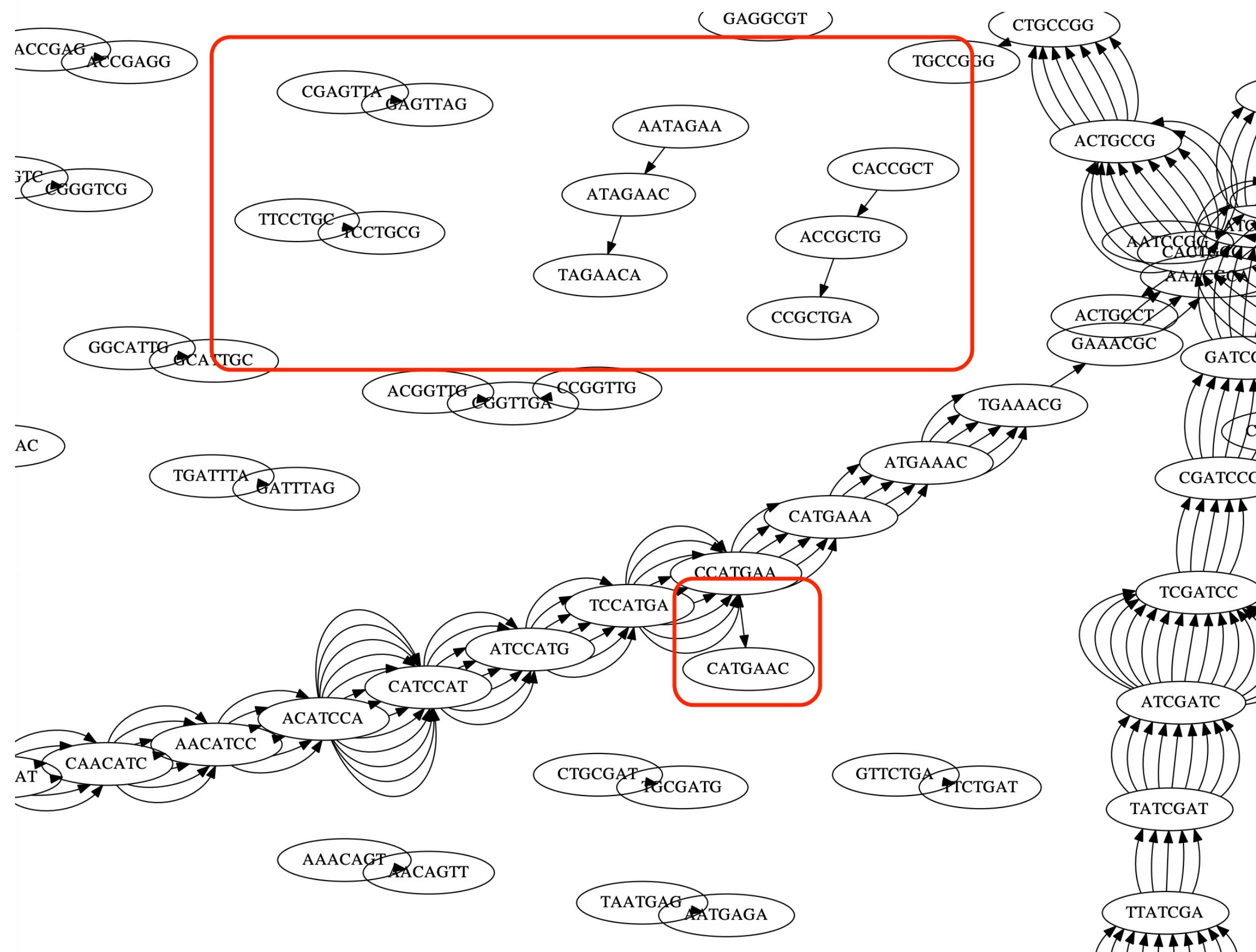
Differences in coverage also lead to non-Eulerian graph

Graph for `a_long_long_long_time`,
 $k = 5$ but with *extra copy* of `ong_t` :

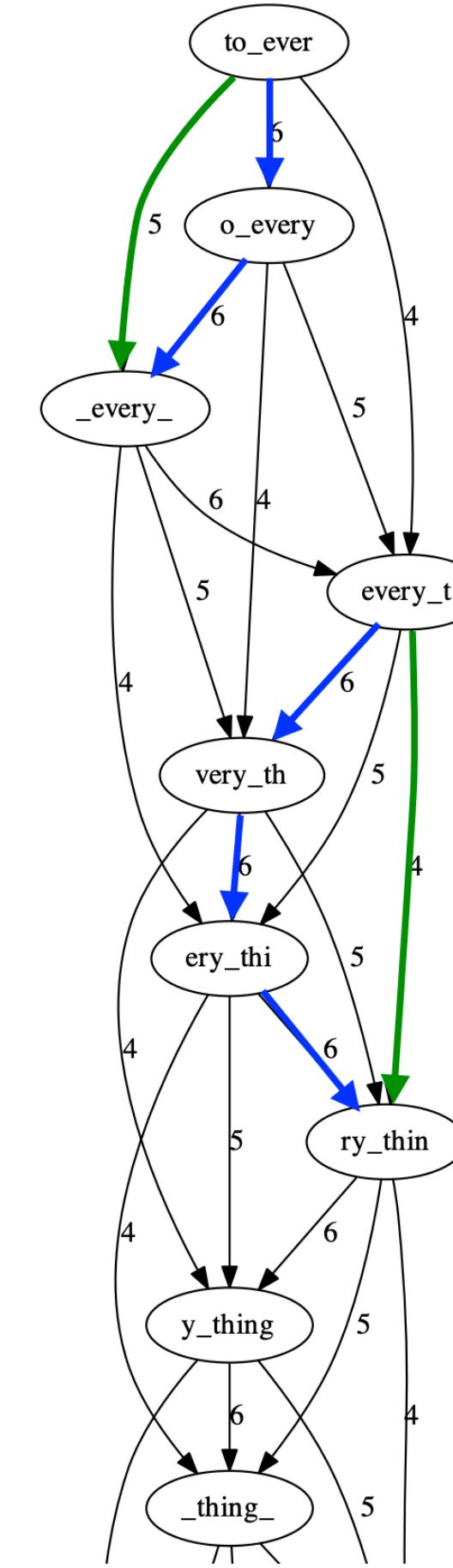
Graph has 4 **semi-balanced** nodes,
isn't Eulerian

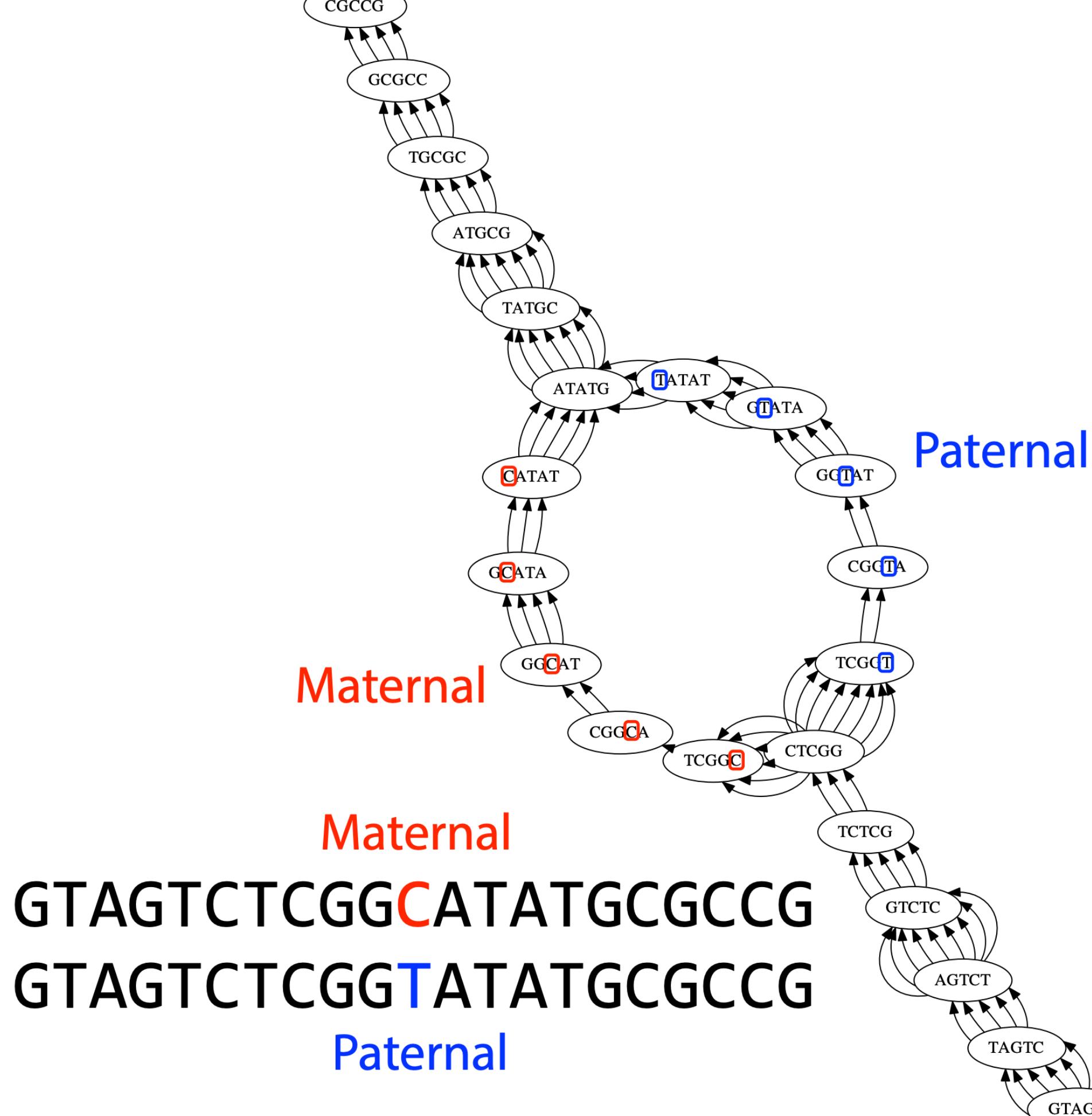


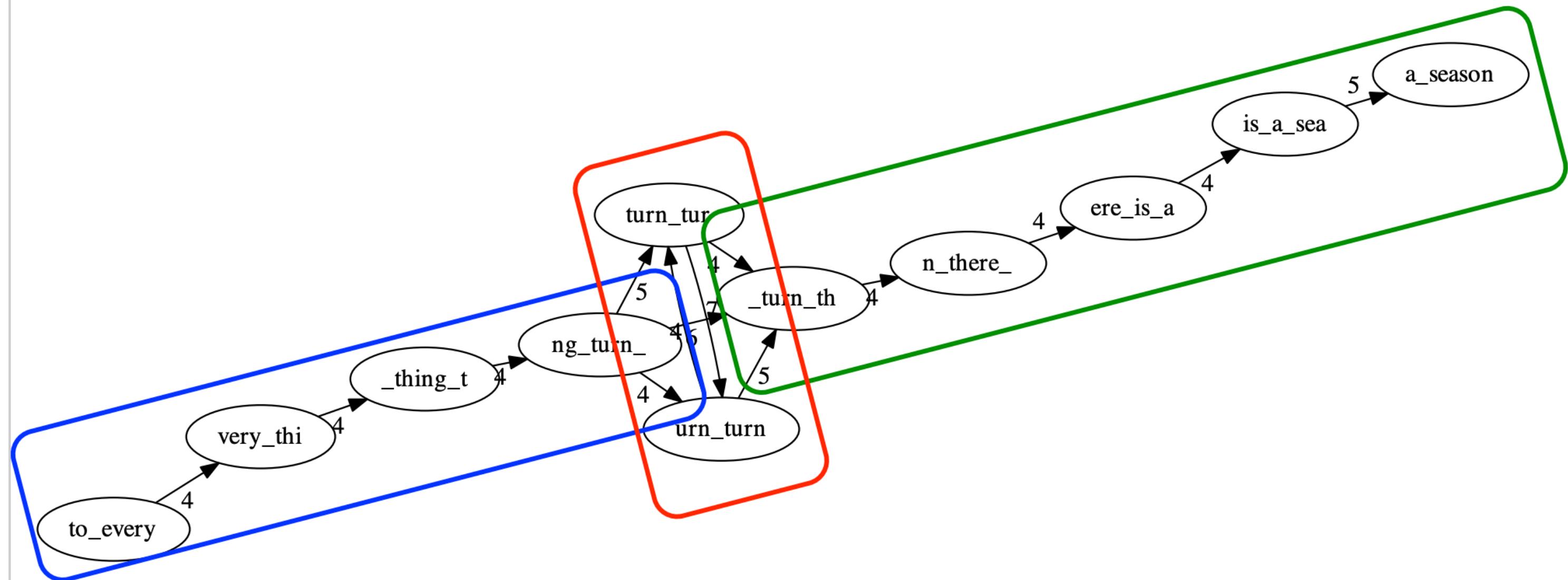




Green edges can be inferred from blue

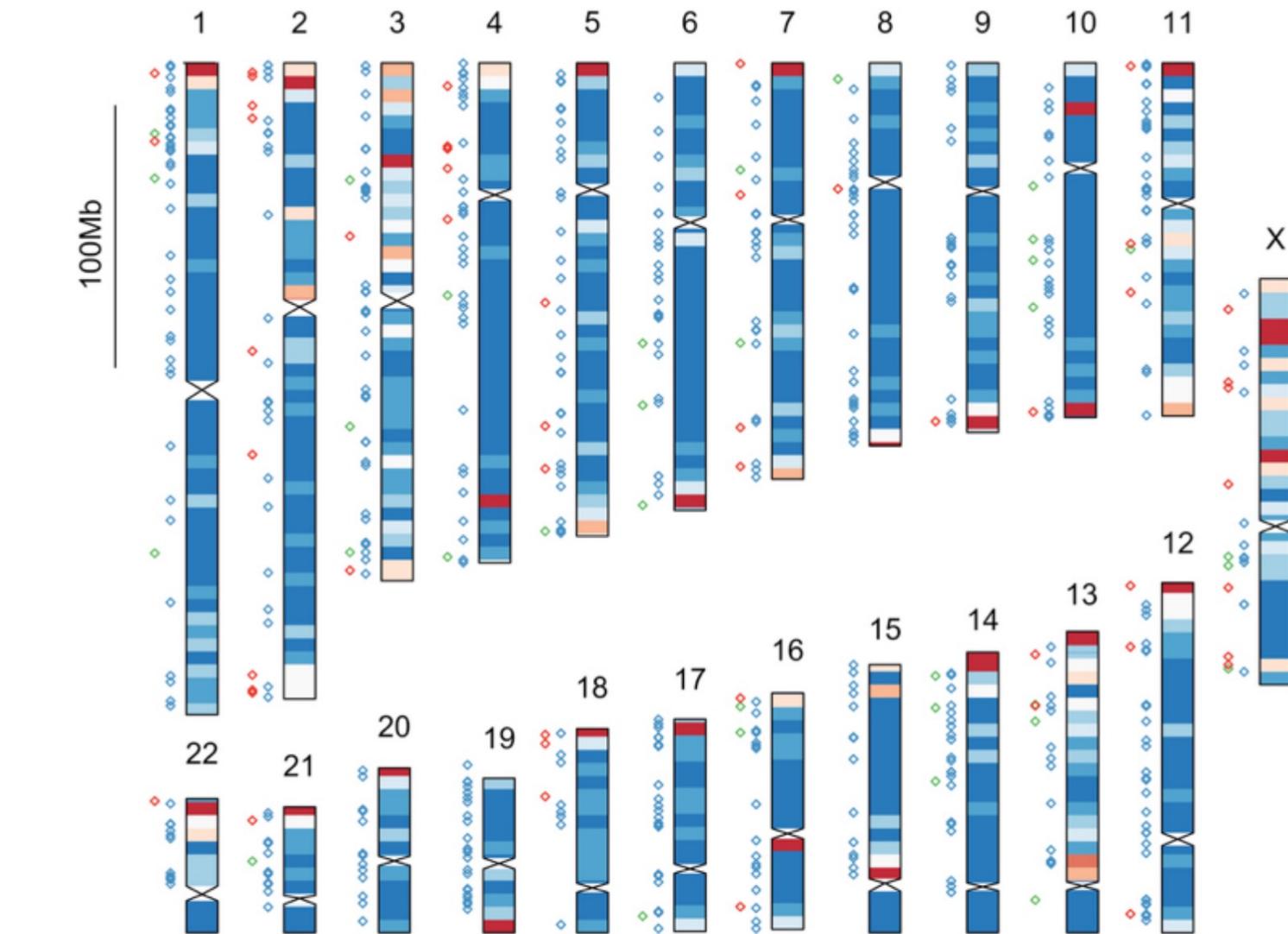
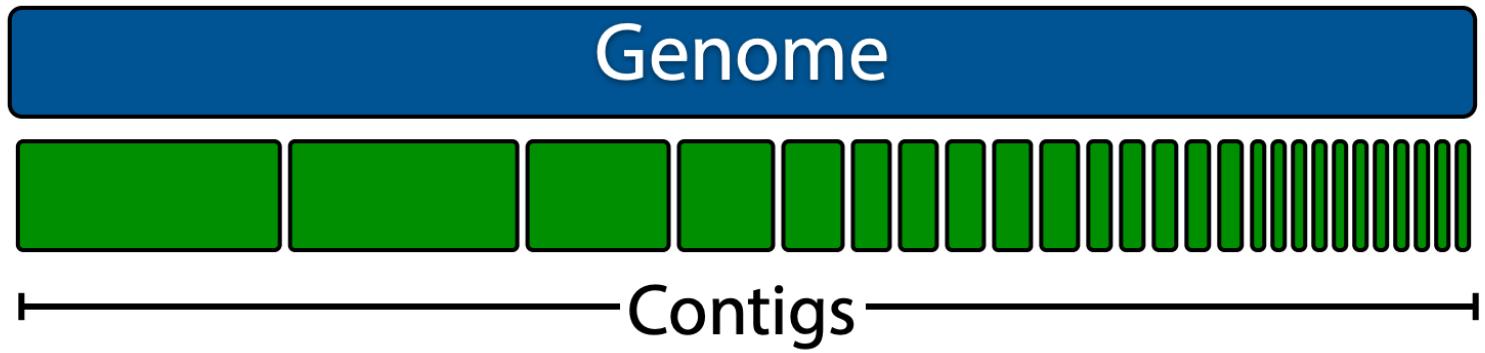






to_every_thing_turn_

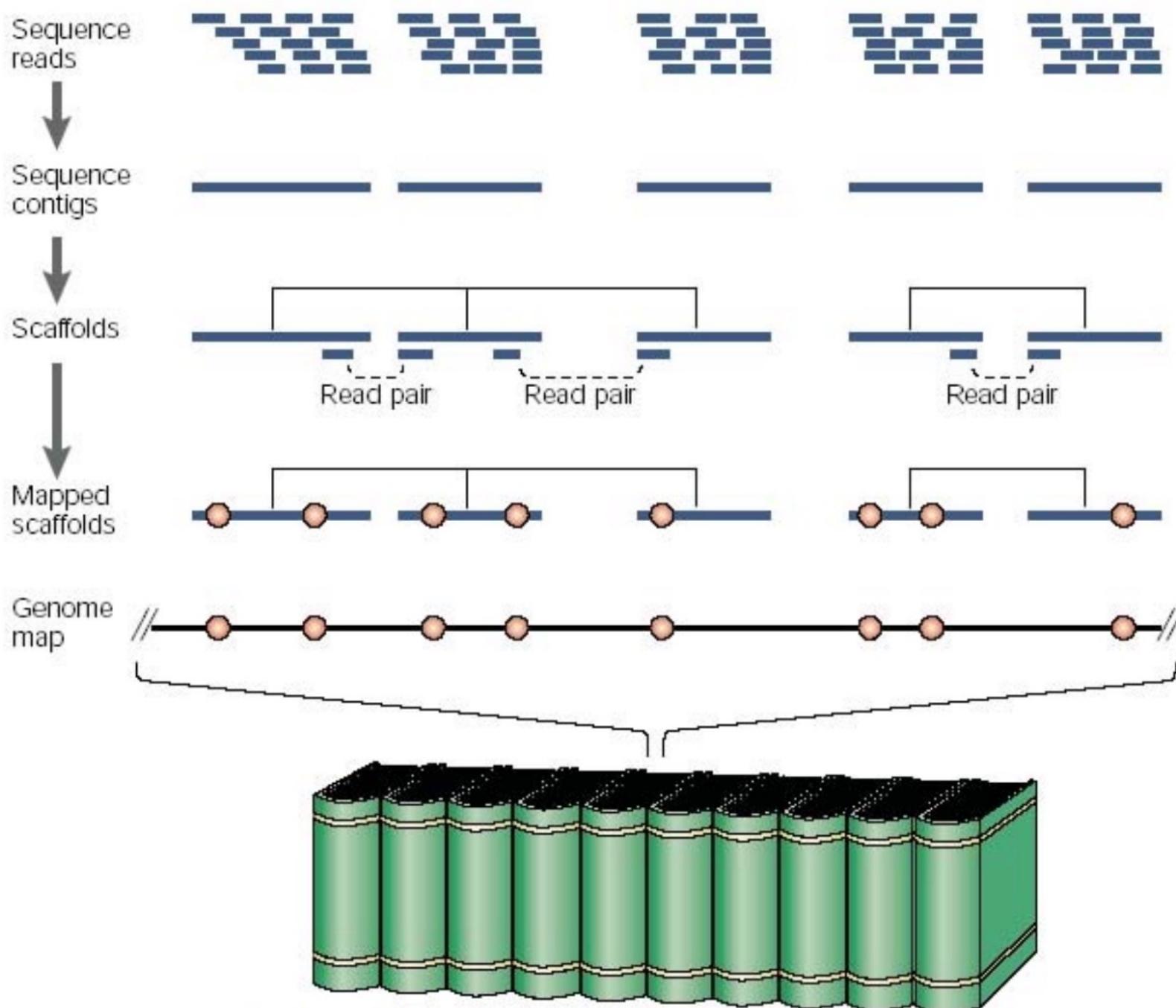
_turn (repeated)



- ◆ Complex event
- ◆ Inversion
- ◆ Closed gap
- STR Density
- High
- Low

Chaisson MJ, et al. Resolving the complexity of the human genome using single-molecule sequencing. *Nature*. 2015 Jan 29;517(7536):608-11.

de novo whole-genome shotgun assembly



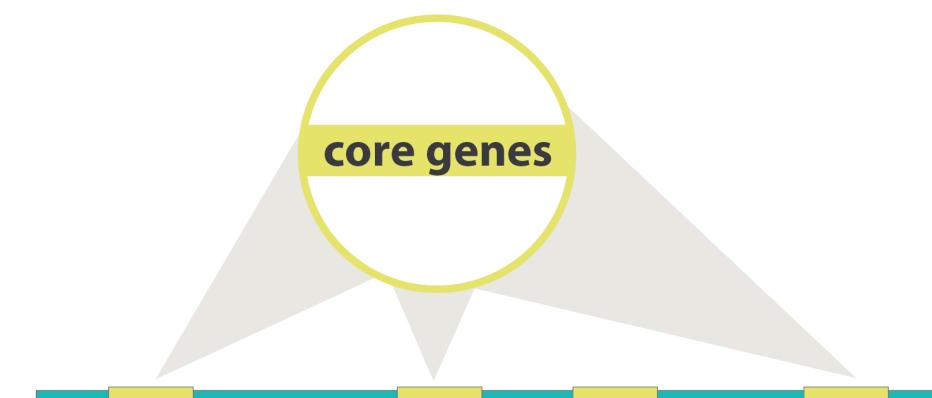
Courtesy of Nature Education. Used with permission.
Source: Green, Eric D. "Strategies for the Systematic Sequencing of Complex Genomes." *Nature Reviews Genetics* 2, no. 8 (2001): 573-83.

Genome assembly quality control, or “the 3C”

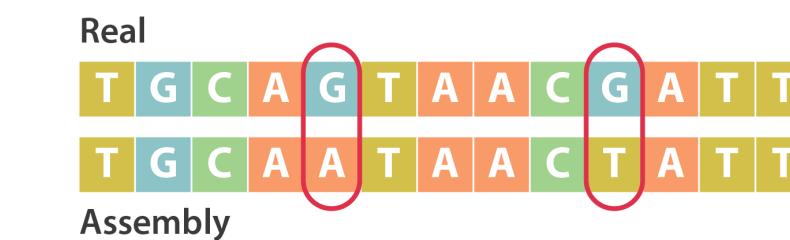
Contiguity



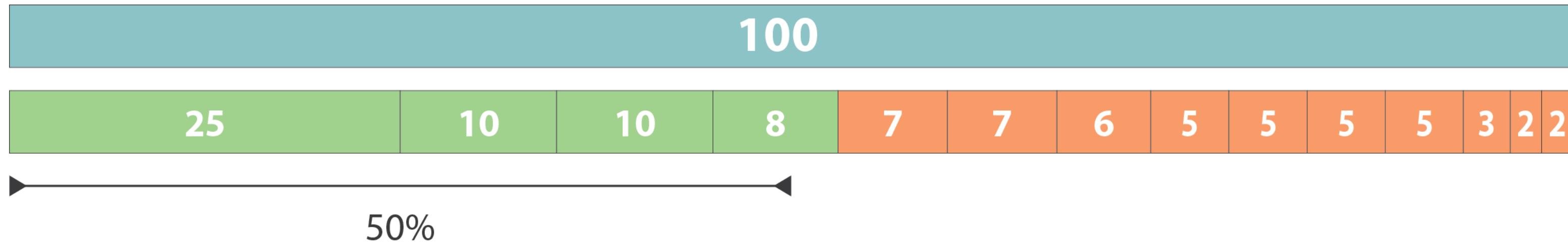
Completeness



Correctness



N50 & L50



N50: given a set of sequences of varying lengths, the N50 is defined as **the length L of the shortest contig for which longer and equal length contigs cover at least 50% of the assembly.**

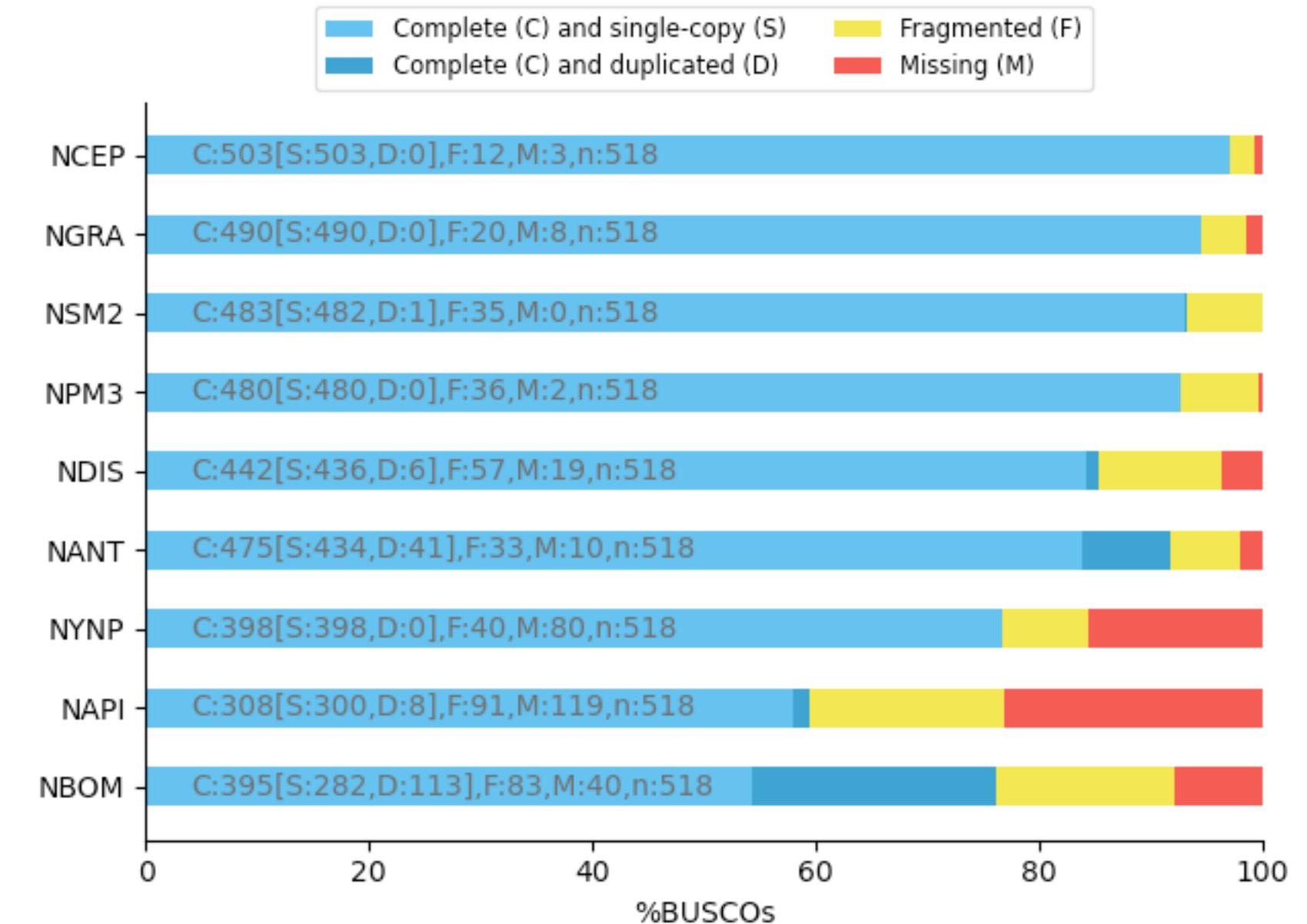
L50: given a set of sequences of varying lengths, the L50 is defined as **count of smallest number of sequences whose length sum makes up 50% of the assembly.**

N50 describes a sequence length whereas L50 describes a number of sequences.

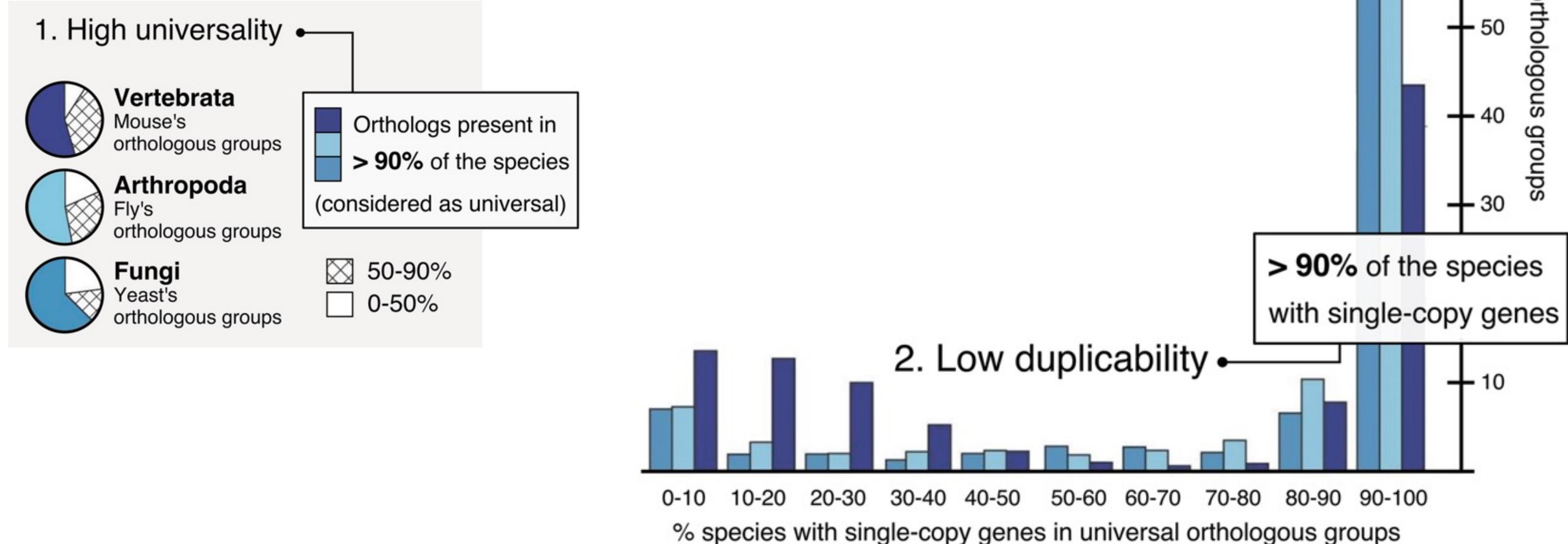
$$\text{N50} = 8 \text{ and } \text{L50} = 4$$

“Core” genes

Core genes in assembly
Core genes in reference database



“Core” genes



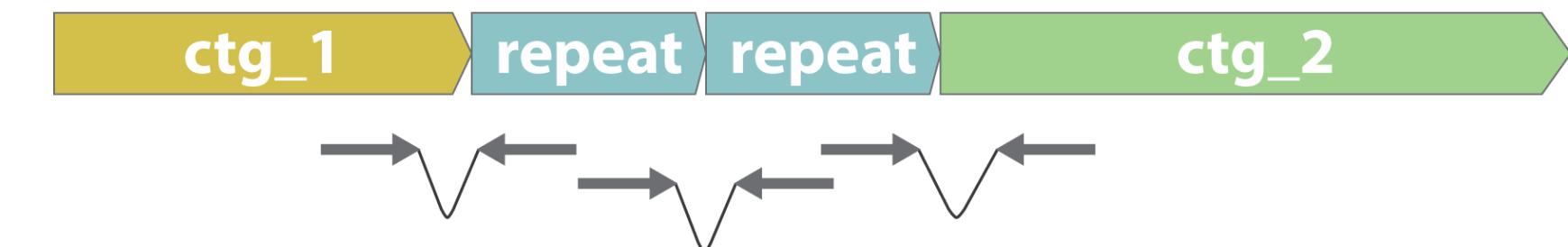
Mistakes into the assembly

- Indels / SNPs
- Mis-joins
- Repeat compressions
- Unnecessary duplications
- Rearrangements

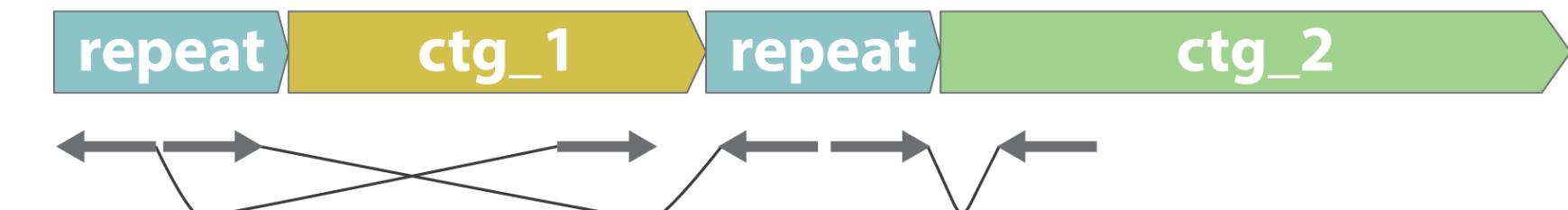
Assembly T | A | C | A | G | T | A | A | C | G | A | T | T

R1	T	G	T	A	-	T	A	A	C	T	A	T	T
R2	T	G	T	A	-	T	A	A	C	T	A	T	T
R3	T	G	T	A	-	T	A	A	C	T	A	T	T
R4	T	A	T	A	-	T	A	A	C	T	A	T	T
R5	T	G	T	A	A	T	A	A	C	C	A	T	T

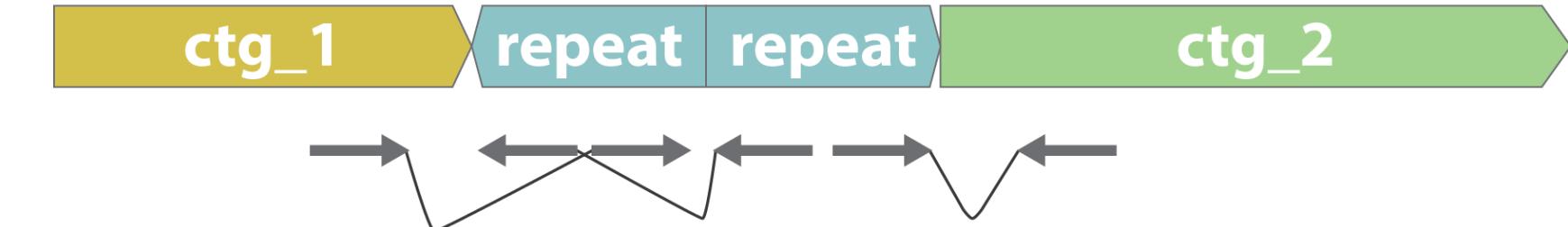
Correct assembly



Rearrangement



Inversion



Adding biological info to sequences

ribosome
binding site

delta toxin
PubMed: 15353161

ACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGA
AAAGCAGCCTCCTGACTTCCCTCGCTTGGTGGTTGAGTGGACCTC
CCAGGCCAGTGCCGGCCCCCTCATAGGAGAGGAAGCTCGGGAGGTG
GCCAGGCCAGGAAGGCGACCCCCCAGCAATCCGCGCGCCGGG
ACAGAATGCCCTGCAGGAACCTTCTAGAACGACCTTCCTCCTG
CAAATAAAACCTCACCCATGAATGCTCACGCAAGTTAATTACAGA
CCTGAAACAAGATGCCATTGTCCCCGGCCTCCTGCTGCTGCTGCT
CTCCGTCCGTCCGTGGGCCACGGCCACCGCTTTTTTTGCC

transfer RNA
Leu-(UUR)

tandem repeat
 $CCGT \times 3$

homopolymer
 $10 \times T$