

# Module 2 - Assembly

## Lecture 10: Genomics

Bioinformatics Algorithms CSC4181/6802

Most slides used are from Ben Langmead's Teaching Materials ([www.langmead-lab.org/teaching-materials](http://www.langmead-lab.org/teaching-materials))

# Sequencing Technology

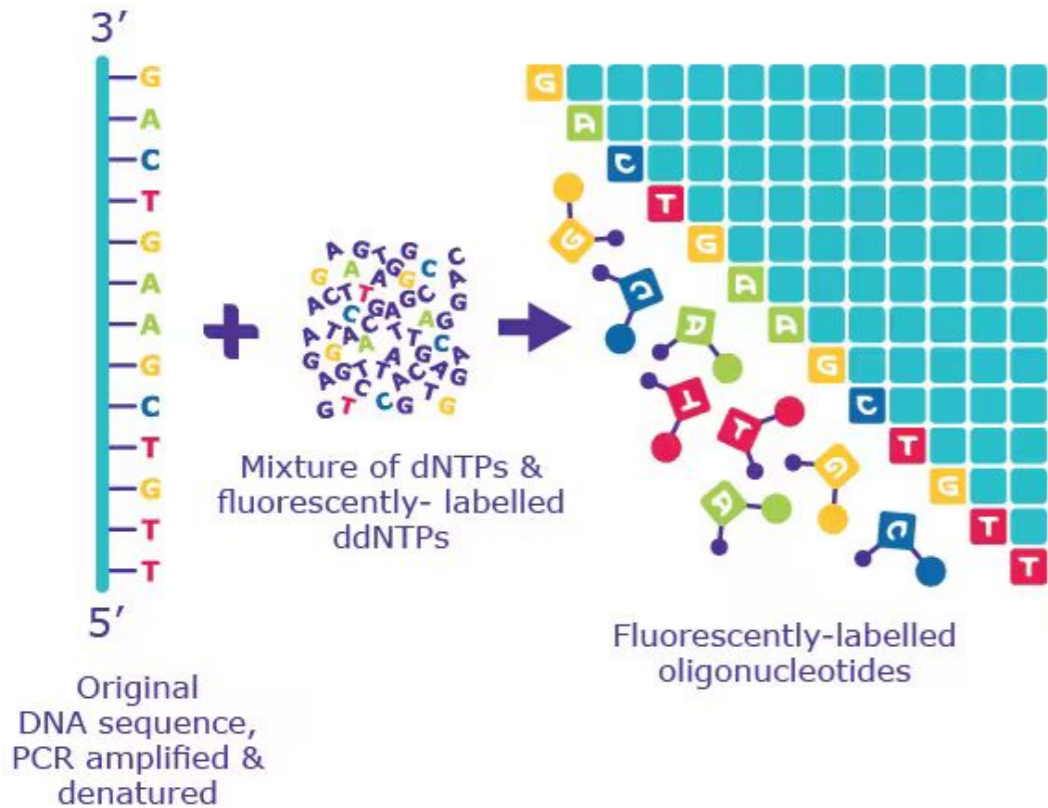
First generation



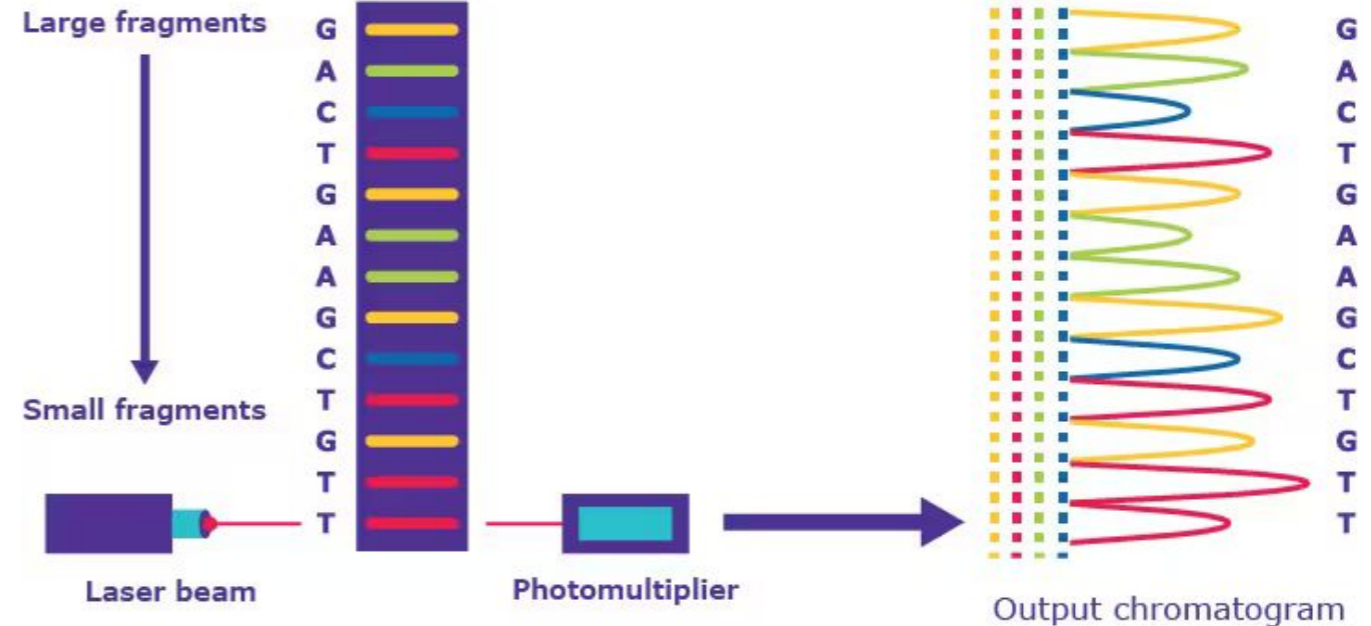
Sanger sequencing  
Maxam and Gilbert  
Sanger chain termination

# Sanger Sequencing

## 1 PCR with fluorescent, chain-terminating ddNTPs



## 2 Size separation by capillary gel electrophoresis



## 3 Laser excitation & detection by sequencing machine

# Sequencing Technology

First generation



Sanger sequencing  
Maxam and Gilbert  
Sanger chain termination

Infer nucleotide identity using dNTPs,  
then visualize with electrophoresis

500–1,000 bp fragments

# Sequencing Technology

First generation

Second generation  
(next generation sequencing)



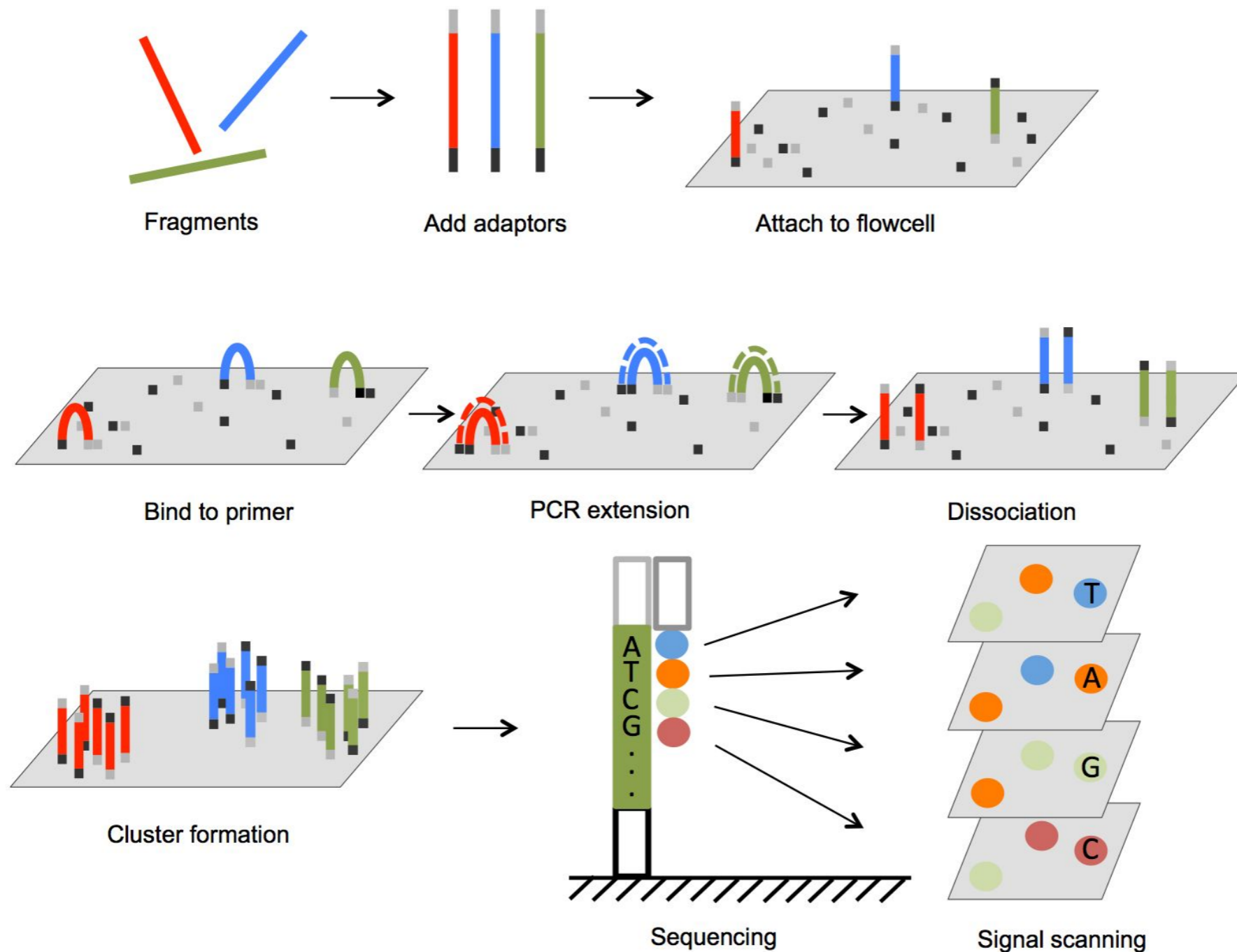
Sanger sequencing  
Maxam and Gilbert  
Sanger chain termination

454, Solexa,  
Ion Torrent,  
Illumina

Infer nucleotide identity using dNTPs,  
then visualize with electrophoresis

500–1,000 bp fragments

# Sequencing by Synthesis



# Sequencing Technology

First generation

Second generation  
(next generation sequencing)



Sanger sequencing  
Maxam and Gilbert  
Sanger chain termination

454, Solexa,  
Ion Torrent,  
Illumina

Infer nucleotide identity using dNTPs,  
then visualize with electrophoresis

High throughput from the  
parallelization of sequencing reactions

500–1,000 bp fragments

~50–500 bp fragments

# Sequencing Technology

First generation

Second generation  
(next generation sequencing)

Third generation



Sanger sequencing  
Maxam and Gilbert  
Sanger chain termination

Infer nucleotide identity using dNTPs,  
then visualize with electrophoresis

500–1,000 bp fragments



454, Solexa,  
Ion Torrent,  
Illumina

High throughput from the  
parallelization of sequencing reactions

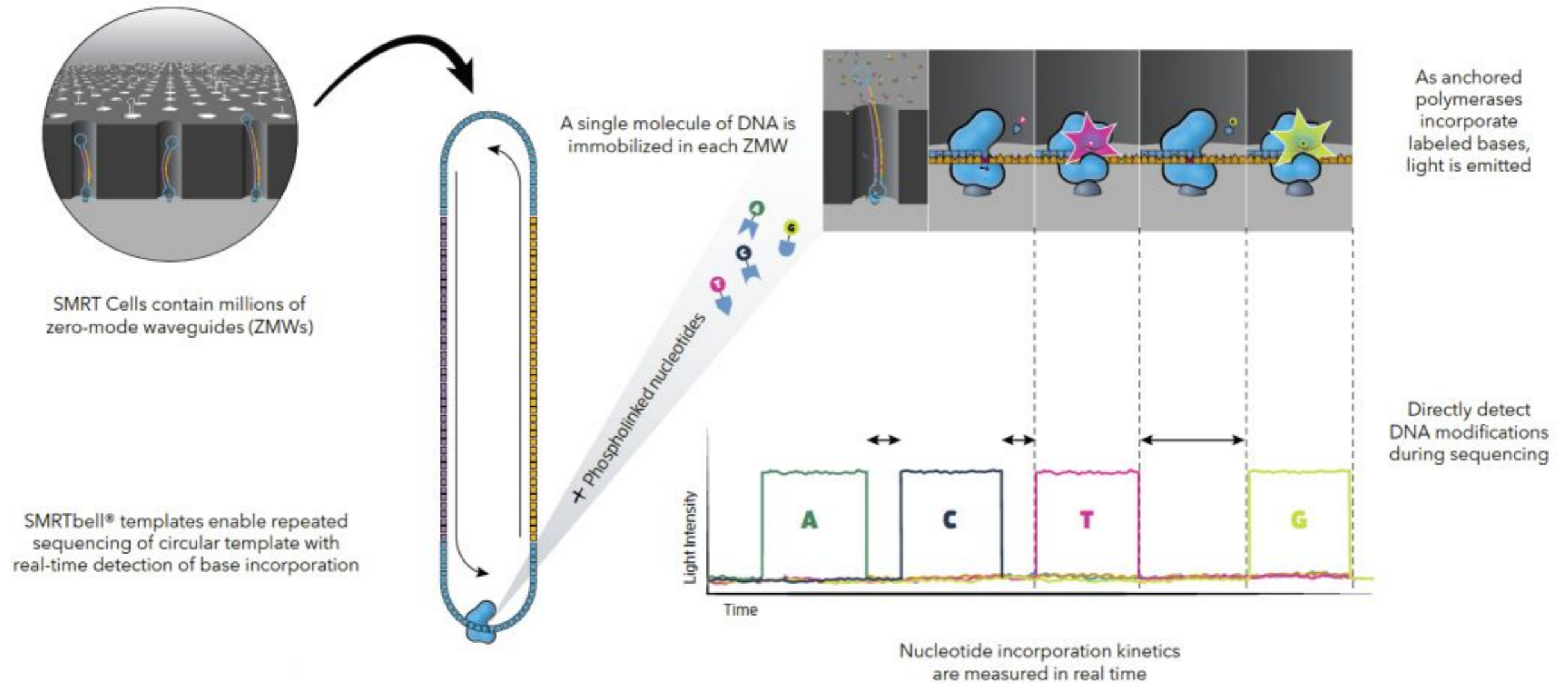
~50–500 bp fragments



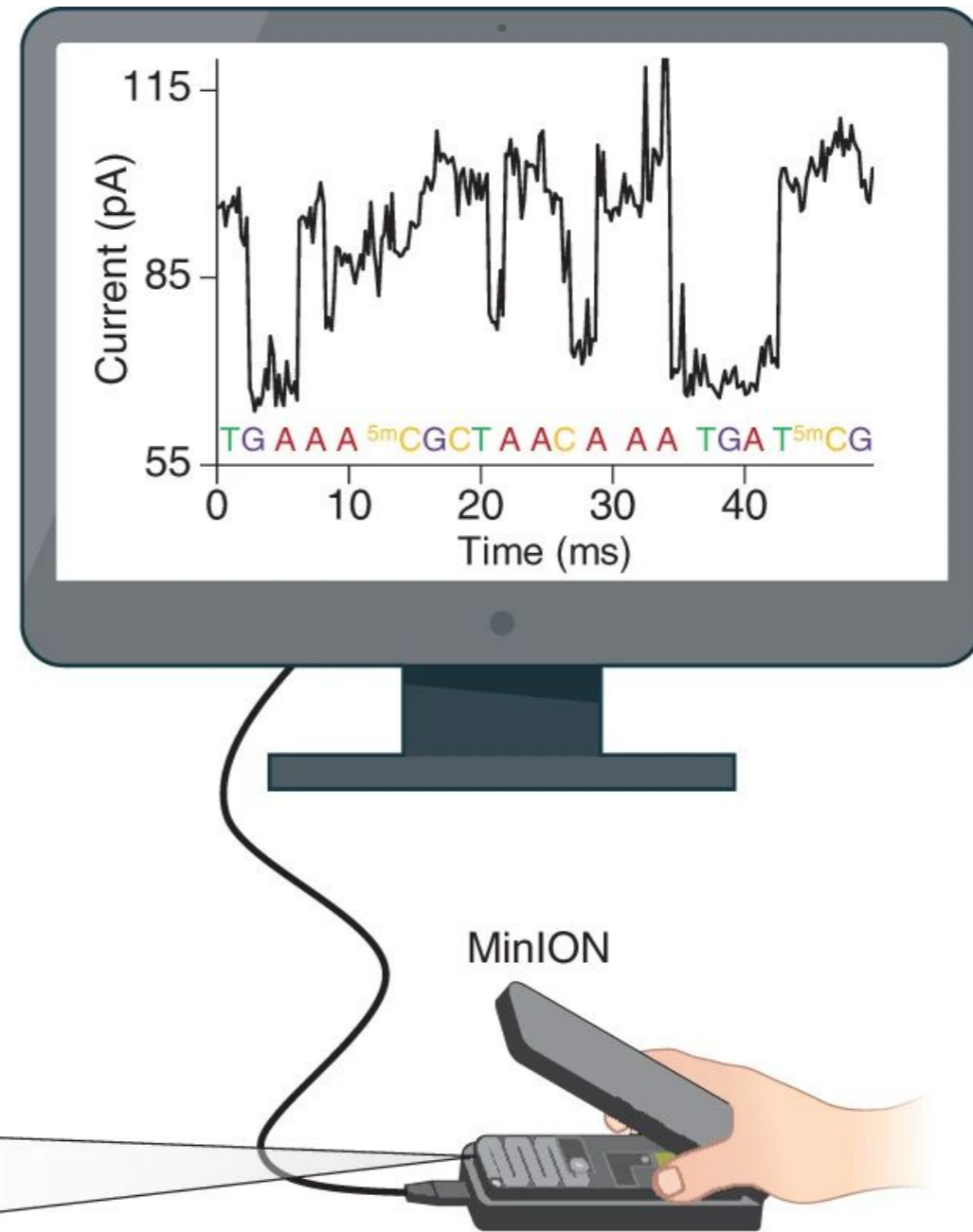
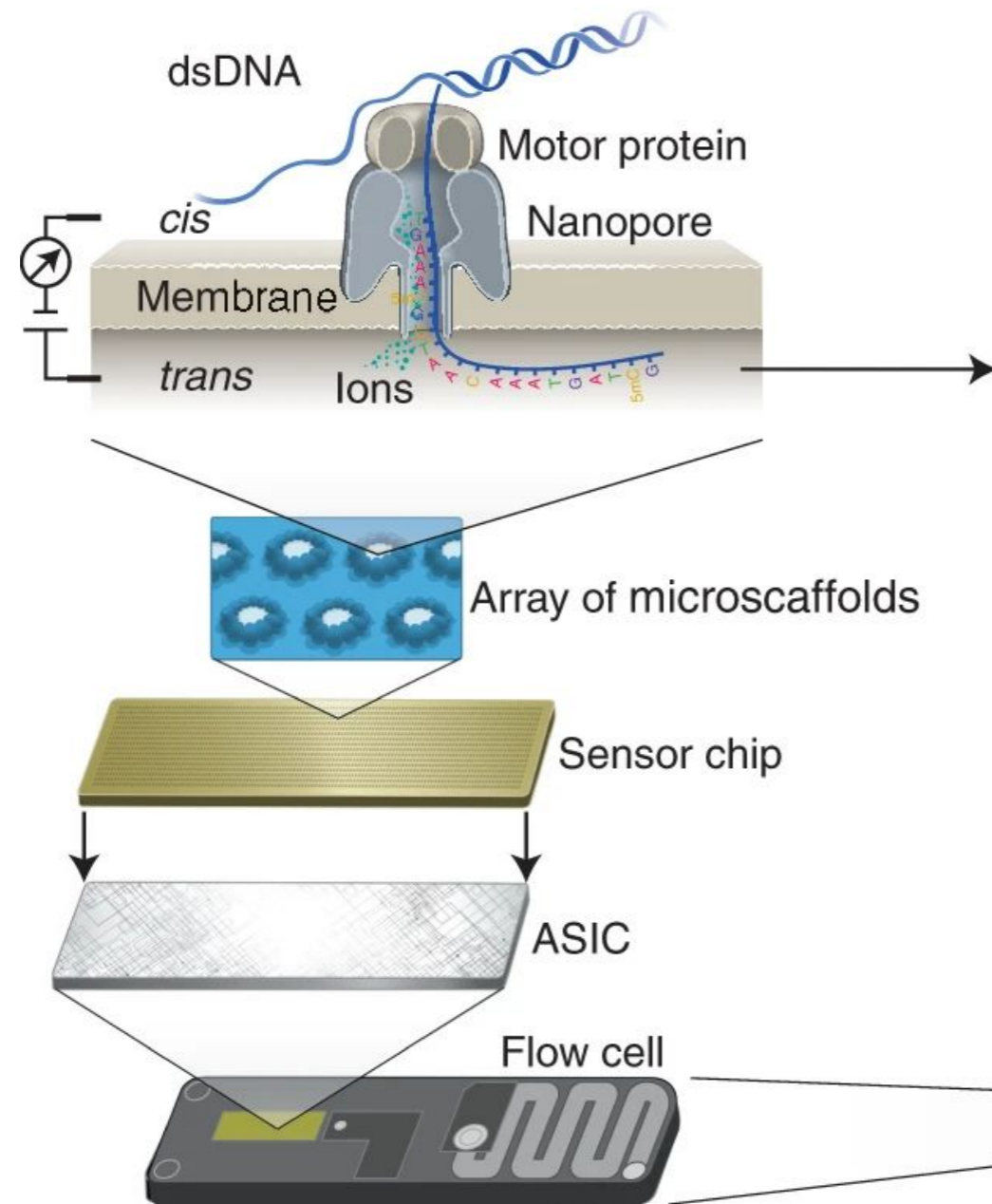
PacBio  
Oxford Nanopore



# PacBio Sequencing



# Nanopore Sequencing



<https://www.nature.com/articles/s41587-021-01108-x/figures/1>

# Sequencing Technology

First generation

Second generation  
(next generation sequencing)

Third generation



Sanger sequencing  
Maxam and Gilbert  
Sanger chain termination

454, Solexa,  
Ion Torrent,  
Illumina

PacBio  
Oxford Nanopore

Infer nucleotide identity using dNTPs,  
then visualize with electrophoresis

High throughput from the  
parallelization of sequencing reactions

Sequence native DNA in real time  
with single-molecule resolution

500–1,000 bp fragments

~50–500 bp fragments

Tens of kb fragments, on average

# Sequencing Technology

First generation

Second generation  
(next generation sequencing)

Third generation



Sanger sequencing  
Maxam and Gilbert  
Sanger chain termination

454, Solexa,  
Ion Torrent,  
Illumina

PacBio  
Oxford Nanopore

Infer nucleotide identity using dNTPs,  
then visualize with electrophoresis

High throughput from the  
parallelization of sequencing reactions

Sequence native DNA in real time  
with single-molecule resolution

500–1,000 bp fragments

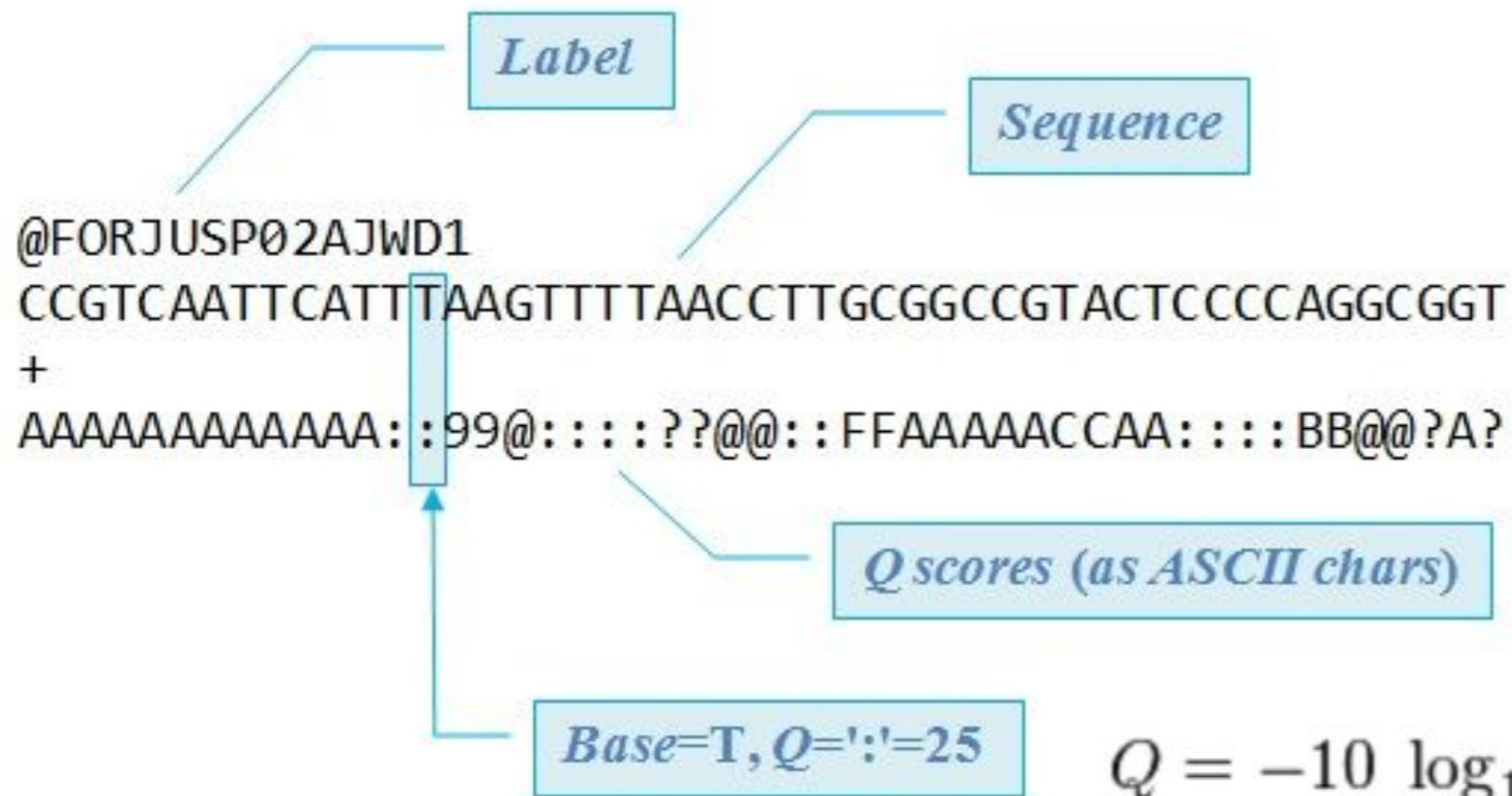
~50–500 bp fragments

Tens of kb fragments, on average

Short-read sequencing

Long-read sequencing

# Capturing measurement error: FASTQ



Quality value Q is an integer representation of the probability p that a corresponding base call is incorrect

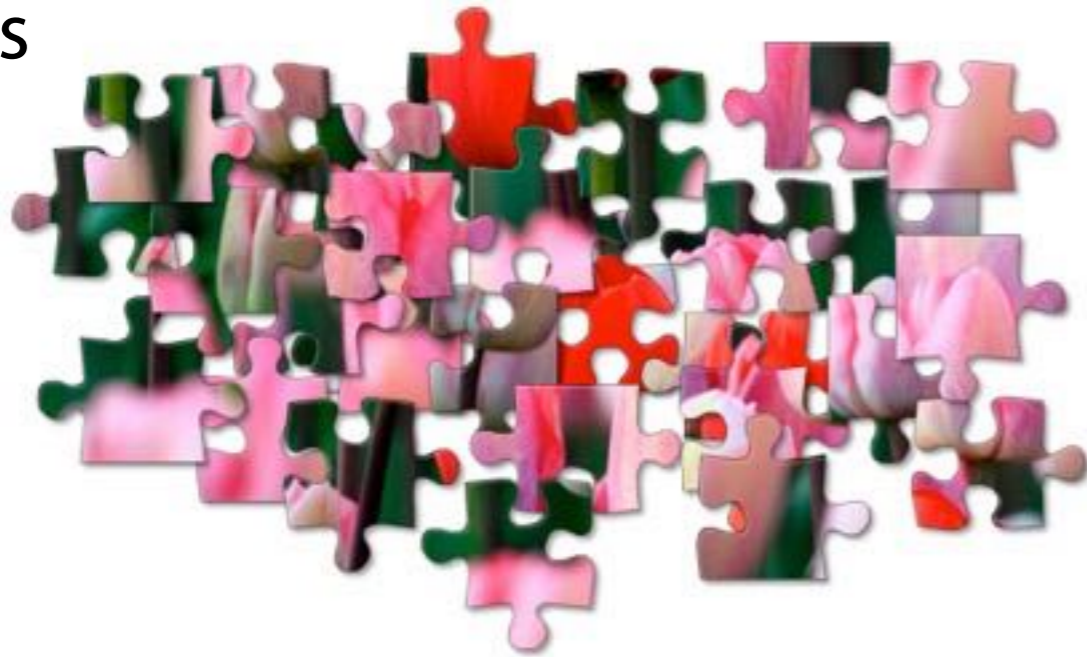
Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10000	99.99%
50	1 in 100000	99.999%

[https://www.drive5.com/usearch/manual/fastq\\_files.html](https://www.drive5.com/usearch/manual/fastq_files.html)

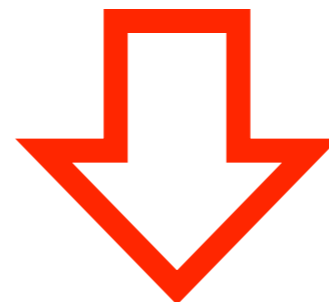
<https://learn.gencore.bio.nyu.edu/ngs-file-formats/quality-scores/>

# Assembly

Reads



Reference genome



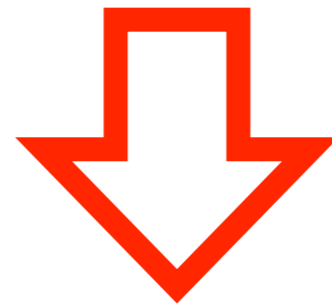
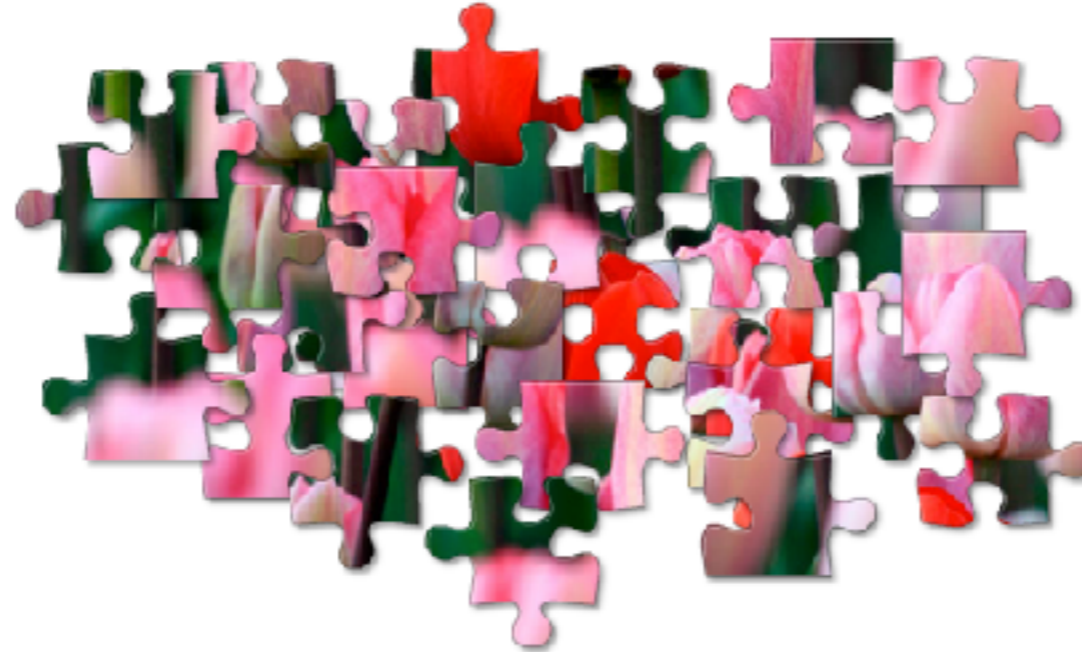
Input DNA



How do we assemble puzzle without the benefit of knowing what the finished product should look like?

(That's what the Human Genome Project had to do!)

# De novo shotgun assembly



# Assembly

Whole-genome “shotgun” sequencing first copies the input DNA:

Input: GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT

Copy: GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT  
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT  
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT  
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT

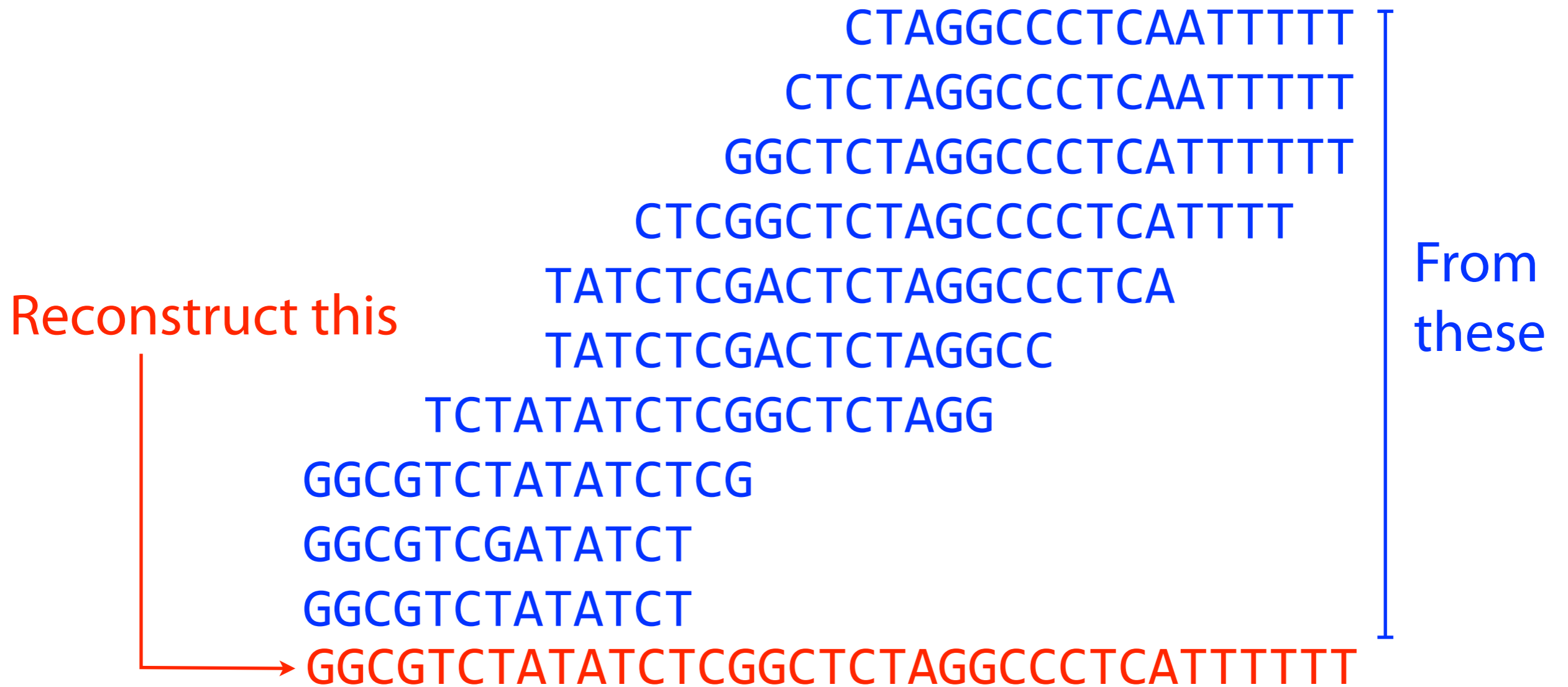
Then fragments it:

Fragment: GGCGTCTA TATCTCGG CTCTAGGCCCTC ATTTTTT  
GGC GTCTATAT CTCGGCTCTAGGCCCTCA TTTTTT  
GGCGTC TATATCT CGGCTCTAGGCCCT CATTTTTTT  
GGCGTCTAT ATCTCGGCTCTAG GCCCTCA TTTTTT

“Shotgun” refers to the random fragmentation of the whole genome; like it was fired from a shotgun



# Assembly

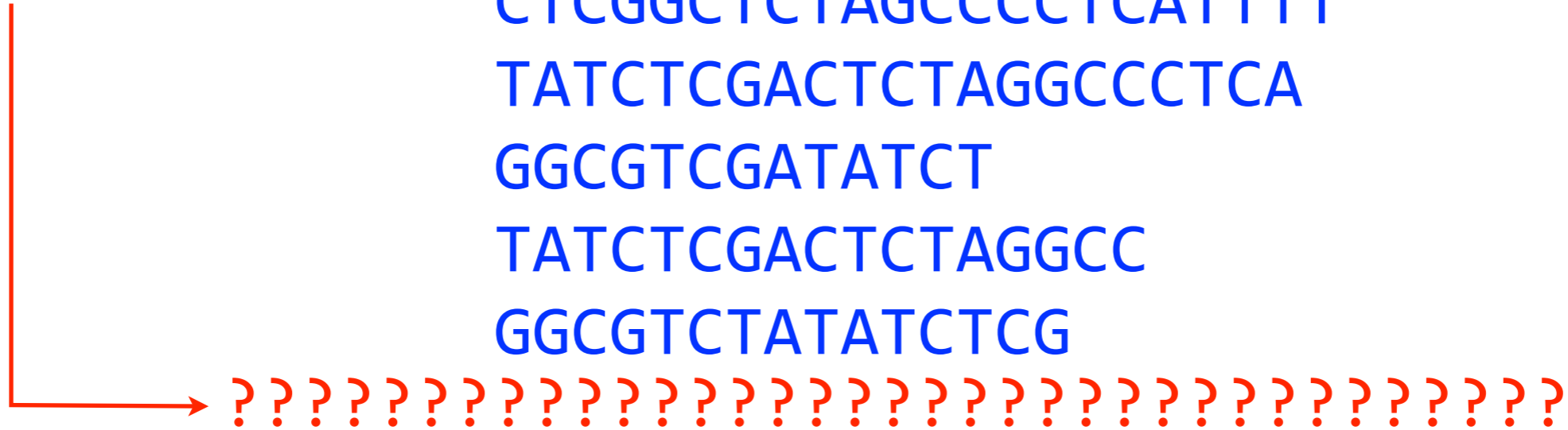


# Assembly

CTAGGCCCTCAATTTTT  
GGCGTCTATATCT  
CTCTAGGCCCTCAATTTTT  
TCTATATCTCGGCTCTAGG  
GGCTCTAGGCCCTCATTTTTT  
CTCGGCTCTAGCCCCTCATT  
TATCTCGACTCTAGGCCCTCA  
GGCGTCGATATCT  
TATCTCGACTCTAGGCC  
GGCGTCTATATCTCG

From  
these

Reconstruct this



# Coverage

CTAGGCCCTCAATTTT  
CTCTAGGCCCTCAATTTT  
GGCTCTAGGCCCTCATTTTT  
CTCGGCTCTAGCCCCTCATT  
TATCTCGACTCTAGGCCCTCA  
TATCTCGACTCTAGGCC  
TCTATATCTCGGCTCTAGG  
GGCGTCTATATCTCG  
GGCGTCGATATCT  
GGCGTCTATATCT  
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTT

Coverage = 5

# Coverage

CTAGGCCCTCAATTTT  
CTCTAGGCCCTCAATTTT  
GGCTCTAGGCCCTCATTTTT  
CTCGGCTCTAGCCCCTCATT  
TATCTCGACTCTAGGCCCTCA  
TATCTCGACTCTAGGCC  
TCTATATCTCGGCTCTAGG  
GGCGTCTATATCTCG  
GGCGTCGATATCT  
GGCGTCTATATCT  
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTT

Coverage = 5

CTAGGCCCTCAATTTT  
CTCTAGGCCCTCAATTTT  
GGCTCTAGGCCCTCATTTTT  
CTCGGCTCTAGCCCCTCATT  
TATCTCGACTCTAGGCCCTCA  
TATCTCGACTCTAGGCC  
TCTATATCTCGGCTCTAGG  
GGCGTCTATATCTCG  
GGCGTCGATATCT  
GGCGTCTATATCT  
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTT

177 bases

35 bases

Average coverage =  $177 / 35 \approx 5$ -fold

TCTATATCTCGGCTCTAGG

TATCTCGACTCTAGGCC

TCTATATCTCGGCTCTAGG

||||| |||||

TATCTCGACTCTAGGCC

# First law of assembly

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

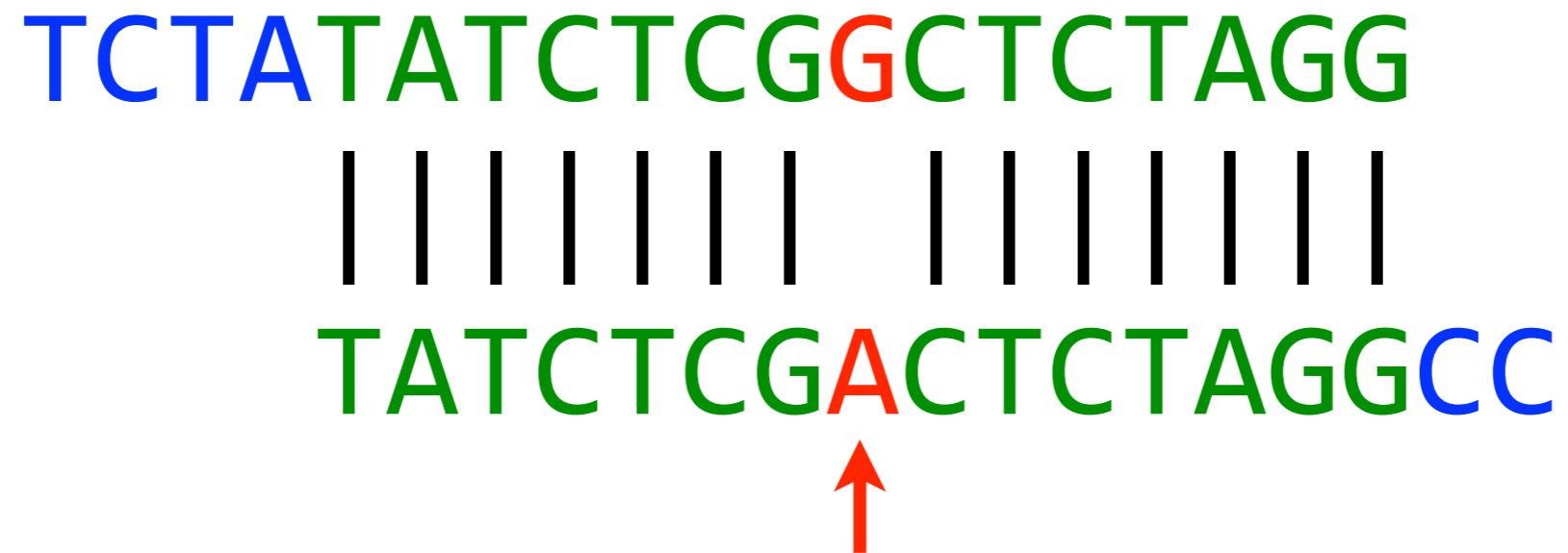
```
TCTATATCTCGGCTCTAGG
| | | | | | | | | |
TATCTCGACTCTAGGCC
```

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

```
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTT
TATCTCGACTCTAGGCC
```



TCTATATCTCGGCTCTAGG  
| | | | | | | | | |  
TATCTCGA CTCTAGGCC



Why the differences?

1. Sequencing errors
2. Ploidy: e.g. humans have 2 copies of each chromosome, and copies can differ



# Second law of assembly

More coverage leads to more and longer overlaps

CTAGGCCCTCAATTTT  
CTCGGCTCTAGGCCCTCATT  
TCTATATCTCGGCTCTAGG  
GGCGTCGATATCT  
GGCGTCTATATCTCGGCTCTAGGCCCTCATT  
CTAGGCCCTCAATTTT  
GGCTCTAGGCCCTCATT  
CTCGGCTCTAGGCCCTCATT  
TATCTCGACTCTAGGCCCTCA  
TCTATATCTCGGCTCTAGG  
GGCGTCTATATCTCG  
GGCGTCTATATCT

less coverage

more coverage

TCTATATCTCGGCTCTAGG

||||| |

TATCTCGACTCTAGGCC

TCTATATCTCGGCTCTAGG

||||| |||||

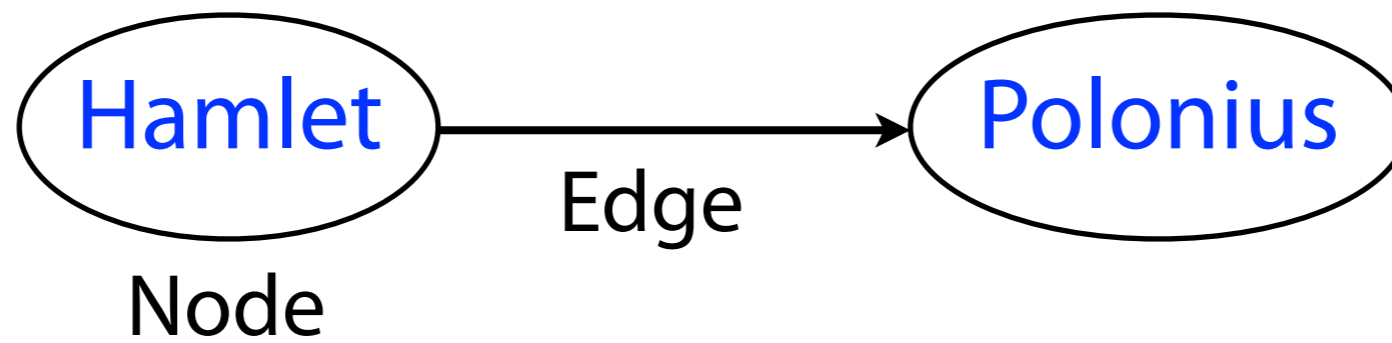
TATCTCGACTCTAGGCC

TATCTCGACTCTAGGCC

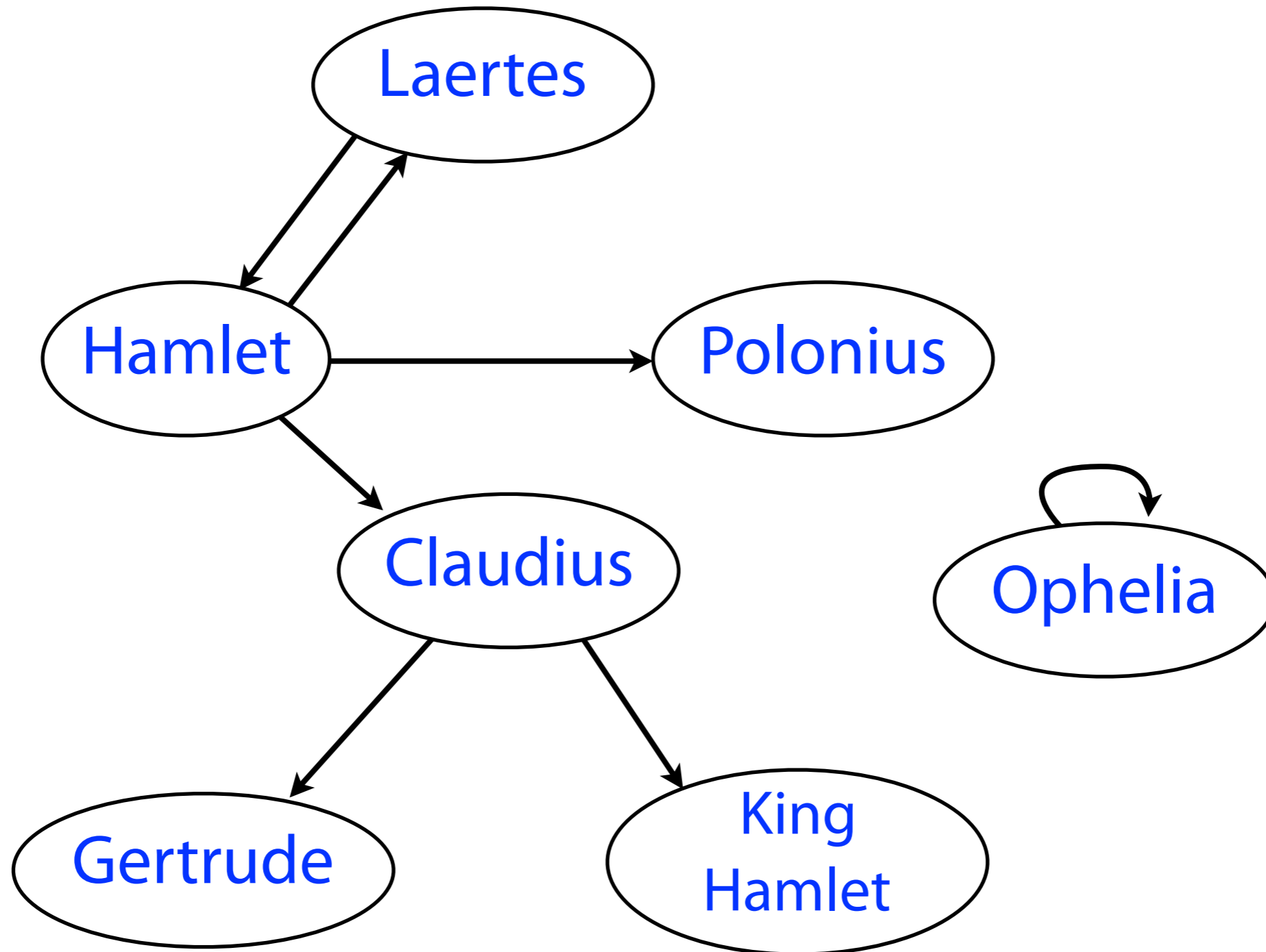
|||| | ||||| ||

CTCGGCTCTAGCCCTCAT

# Directed graph



# Directed graph



# Overlap graph

Each node is a read

CTCGGGCTCTAGCCCCTCATT

Draw edge A  $\rightarrow$  B when suffix of A overlaps prefix of B

CTCGGGCTCTAGCCCCTCATT

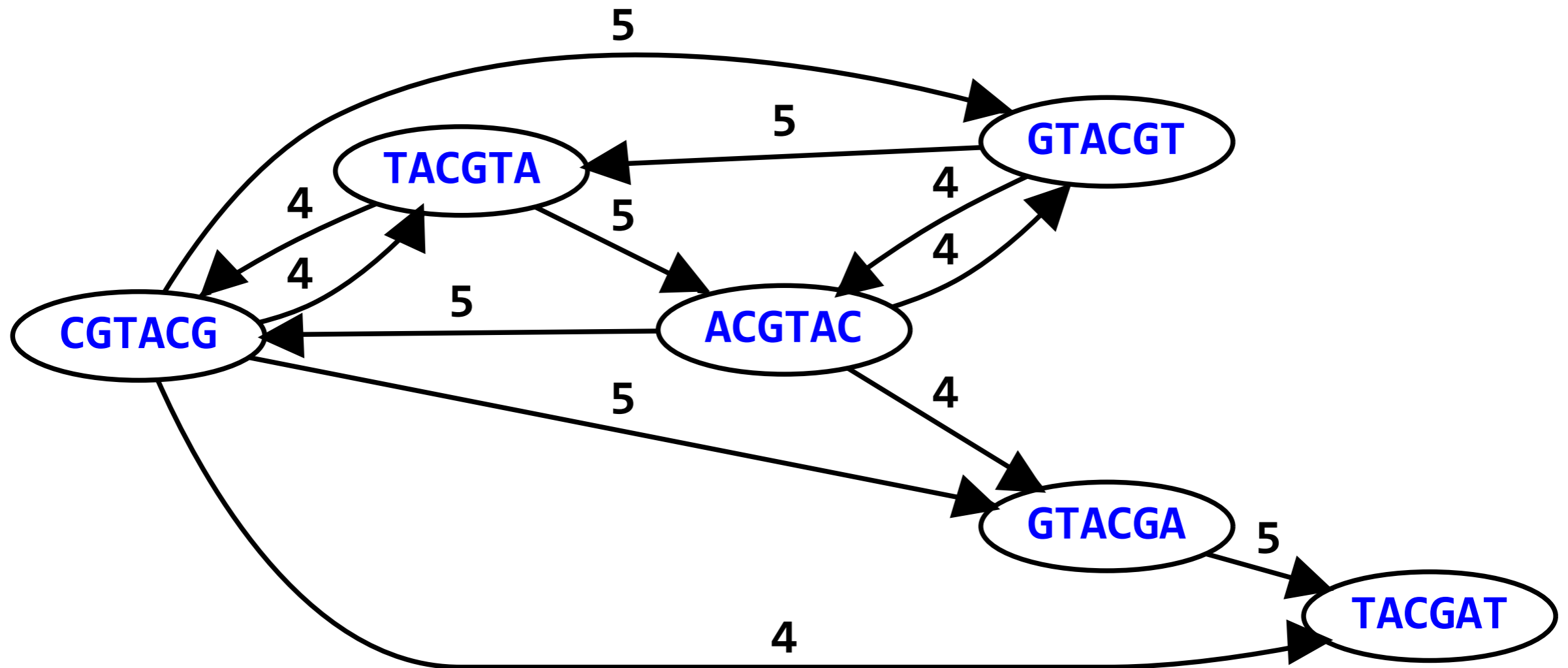
GGCTCTAGCCCCTCATT



# Overlap graph

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

Edges: overlaps of length  $\geq 4$

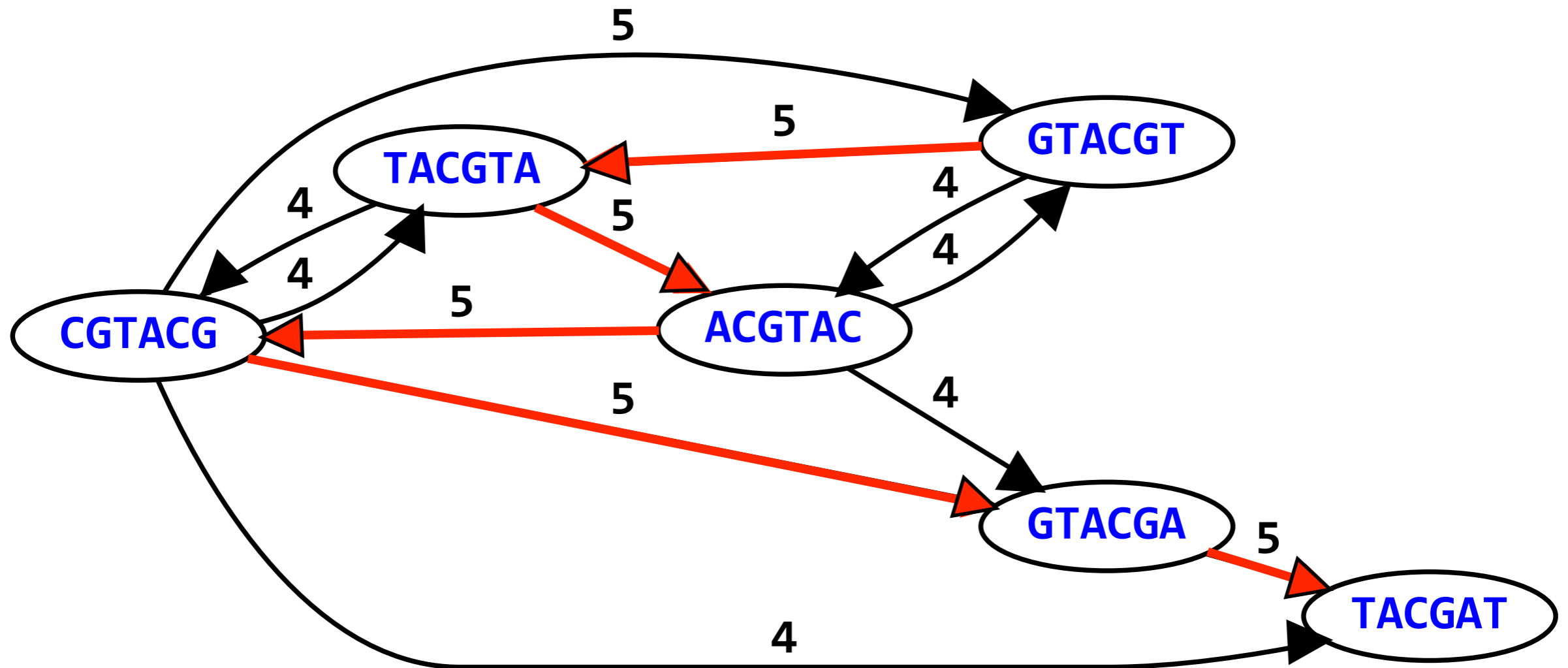




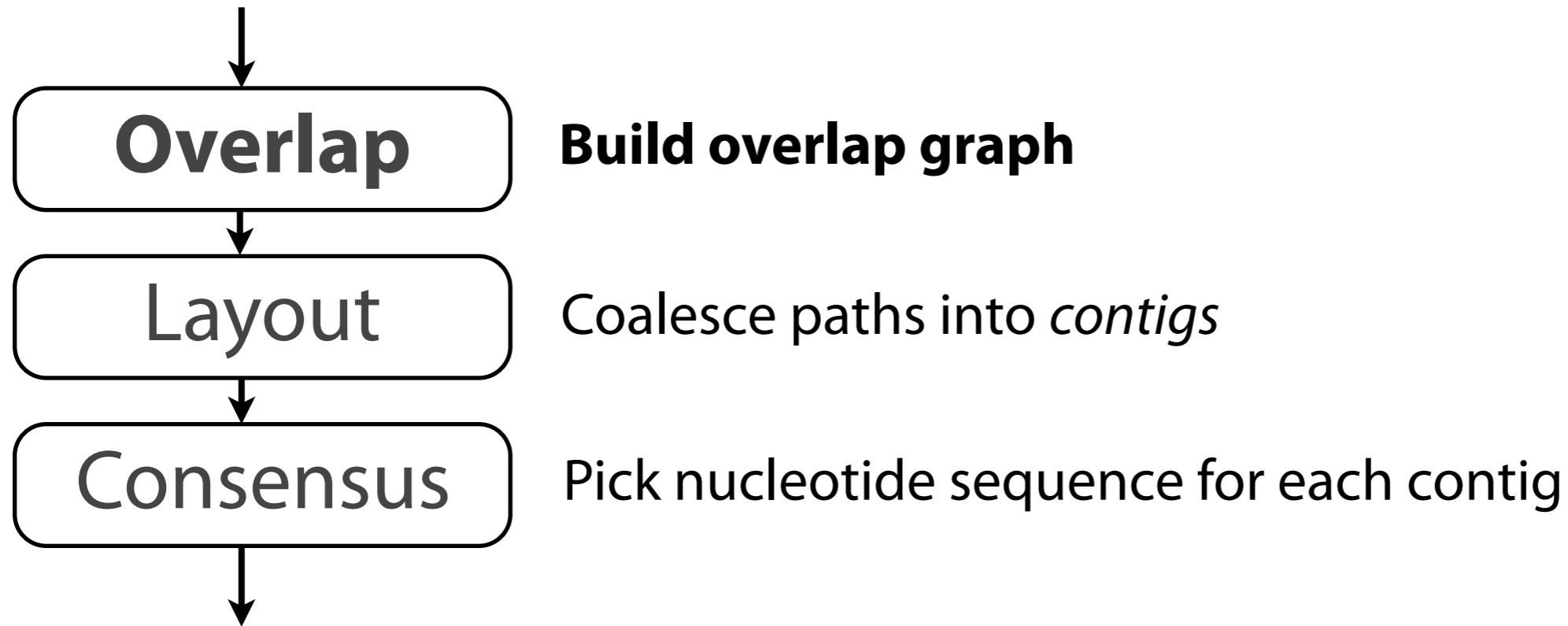
# Overlap graph

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

Edges: overlaps of length  $\geq 4$



# Overlap Layout Consensus



# Finding overlaps

Overlap: Suffix of  $X$  of length  $\geq l$  matches prefix of  $Y$ ;  $l$  is given

Naive: look in  $X$  for occurrences of  $Y$ 's length- $l$  prefix. Extend matches to the right to confirm whether entire suffix of  $X$  matches.

Say  $l = 3$

$X$ : CTCTAGGCC

$Y$ : TAGGCCCTC

Look for this in  $X$

Found it

$X$ : CTCTAGGCC

$Y$ : TAGGCCCTC

Extend to right; confirm a length-6 prefix of  $Y$  matches a suffix of  $X$

$X$ : CTCTAGGCC

$Y$ : TAGGCCCTC

See `suffixPrefixMatch` function in HW5 Q4 (Assembly Challenge)

# Finding overlaps

With suffix tree?

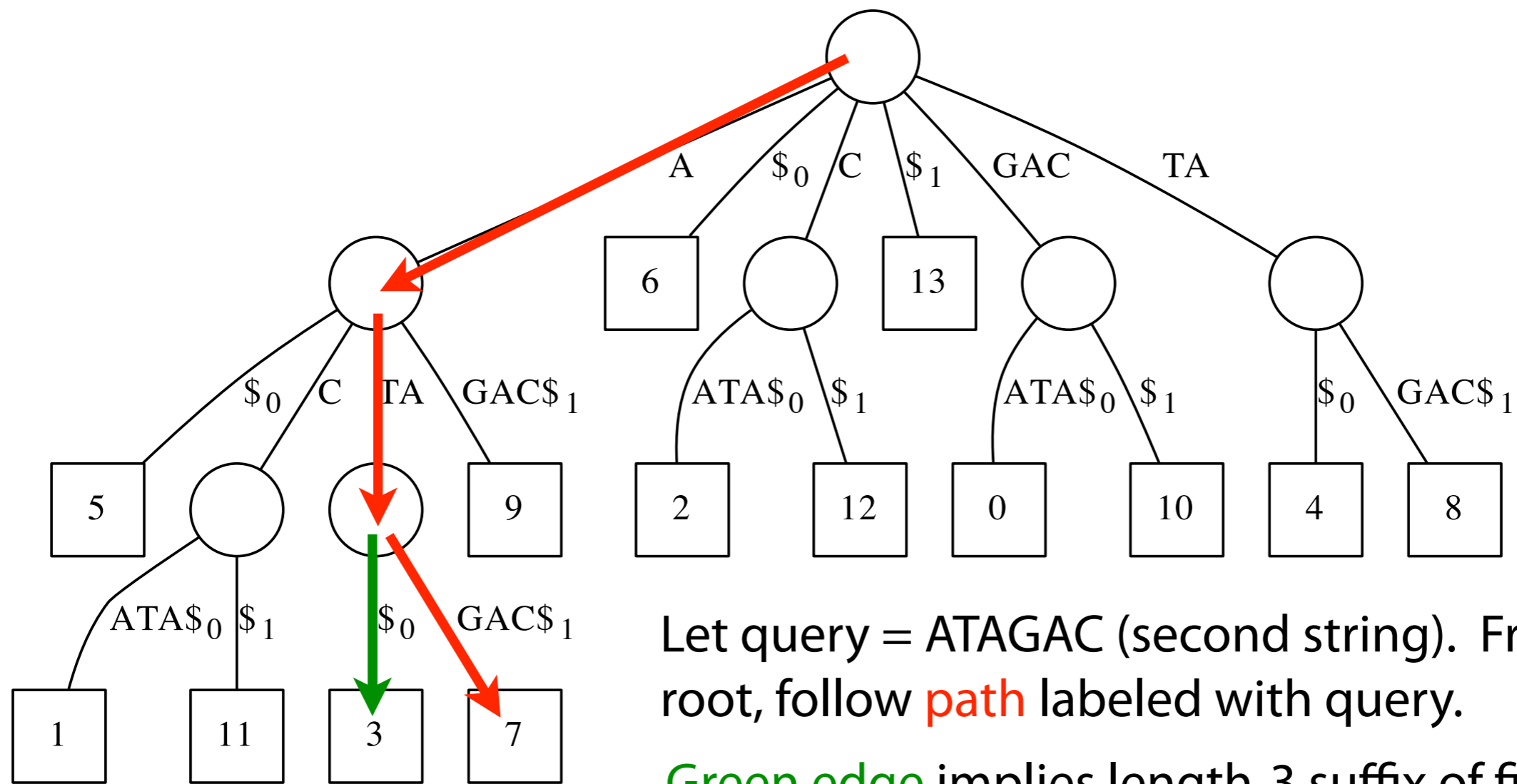
Given a collection of strings  $S$ , for each string  $x$  in  $S$  find all overlaps involving a prefix of  $x$  and a suffix of another string  $y$



# Finding overlaps with suffix tree

Generalized suffix tree for {"GACATA", "ATAGAC"}

GACATA\$<sub>0</sub>ATAGAC\$<sub>1</sub>



GACATA

|||

ATAGAC

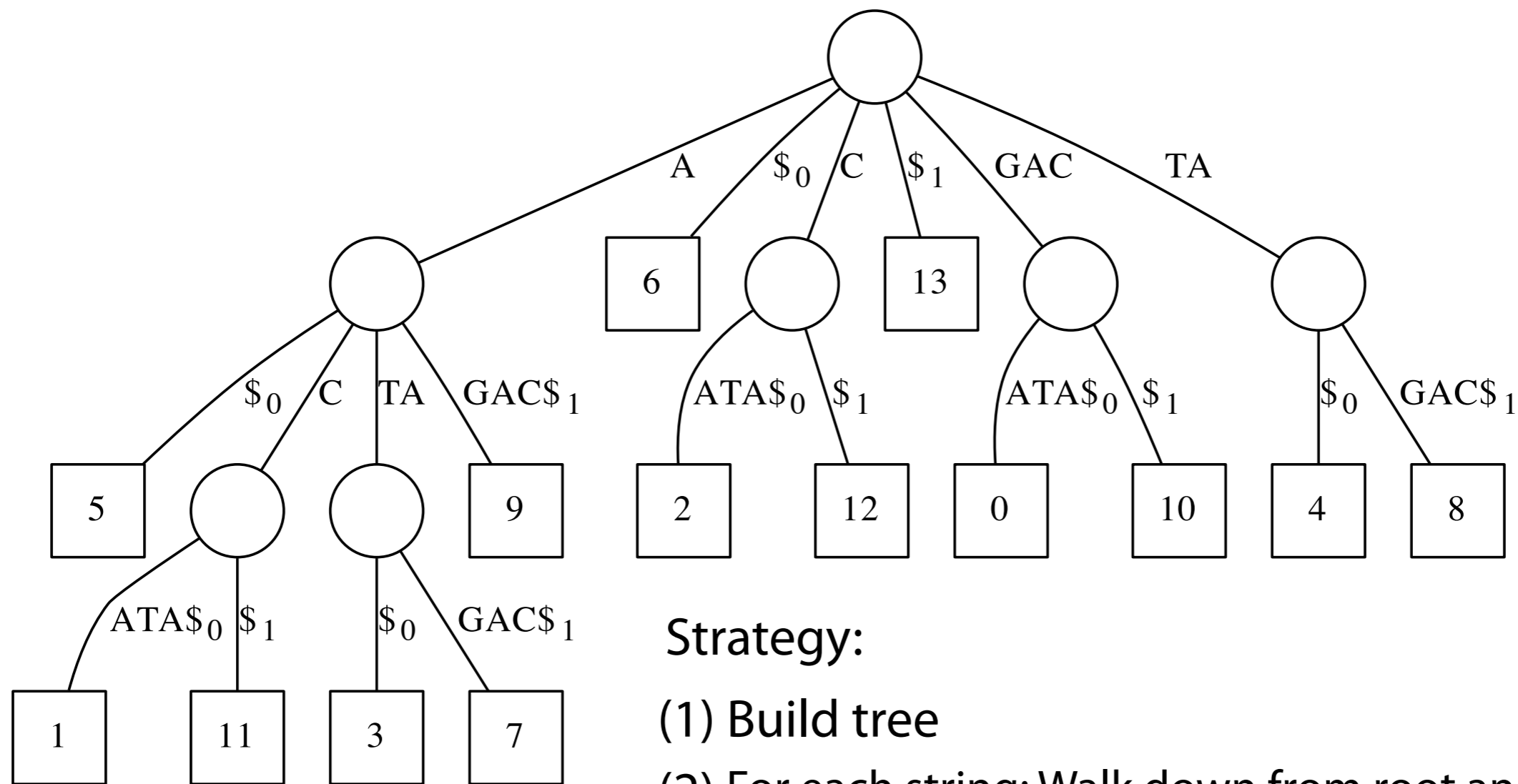
Let query = ATAGAC (second string). From root, follow **path** labeled with query.

**Green edge** implies length-3 suffix of first string equals length-3 prefix of query

# Finding overlaps with suffix tree

Generalized suffix tree for {"GACATA", "ATAGAC"}

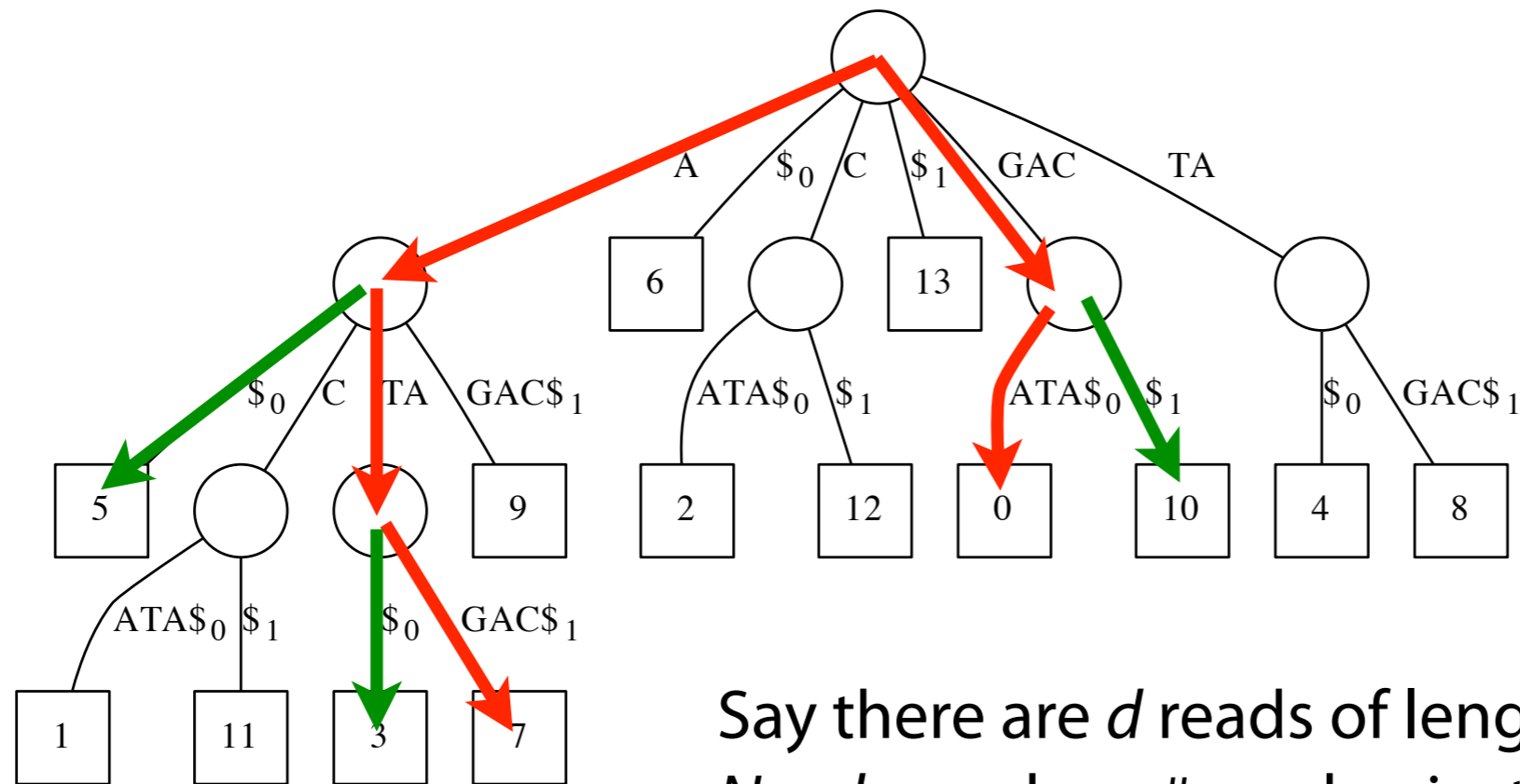
GACATA\$<sub>0</sub>ATAGAC\$<sub>1</sub>



## Strategy:

- (1) Build tree
- (2) For each string: Walk down from root and report any outgoing edge labeled with a separator. Each corresponds to a prefix/suffix match involving prefix of query string and suffix of string ending in the separator.

# Finding overlaps with suffix tree



Say there are  $d$  reads of length  $n$ , total length  $N = dn$ , and  $a = \#$  read pairs that overlap

Assume for given string pair we report only the longest suffix/prefix match

Time to build generalized suffix tree:  $O(N)$

... to walk down red paths:  $O(N)$

... to find & report overlaps (green):  $O(a)$

Overall:  $O(N + a)$



# Finding overlaps

What about *approximate* suffix/prefix matches?

```
X: CTCGGCCCTAGG
      ||| |||||
Y:  GGCTCTAGGCC
```

Dynamic programming

# Finding overlaps with dynamic programming

X: CTCGGCCCTAGG  
 Y: GGCTCTAGGCC

Use *global alignment* recurrence and score function

$$D[i, j] = \min \begin{cases} D[i - 1, j] + s(x[i - 1], -) \\ D[i, j - 1] + s(-, y[j - 1]) \\ D[i - 1, j - 1] + s(x[i - 1], y[j - 1]) \end{cases}$$

$s(a, b)$

	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	

How do we force it to find prefix / suffix matches?

# Finding overlaps with dynamic programming

$$s(a, b)$$

	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	

How to initialize first row & column so suffix of  $X$  aligns to prefix of  $Y$ ?

First column gets 0s  
(any suffix of  $X$  is possible)

First row gets  $\infty$ s  
(must be a prefix of  $Y$ )

Backtrace from last row

$Y$

	-	G	G	C	T	C	T	A	G	G	C	C	C
-	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
C	0	4	12	20	28	36	44	52	60	68	76	84	92
T	0	4	8	14	22	30	38	46	54	62	70	78	86
C	0	4	8	8	16	24	32	40	48	56	64	72	80
G	0	0	4	12	20	28	36	44	52	60	68	76	84
G	0	0	0	8	16	16	24	26	30	36	44	52	60
C	0	4	4	0	8	16	18	26	30	34	36	44	52
C	0	4	8	4	0	8	16	22	30	34	34	36	44
C	0	4	8	8	6	0	10	18	26	34	34	34	36
T	0	4	8	10	8	0	0	10	18	26	34	36	36
A	0	2	6	12	14	12	10	0	10	18	26	34	40
G	0	0	2	10	16	18	16	10	0	10	18	26	34
G	0	0	0	6	14	20	22	18	10	0	10	18	26

$X$ : CTCGGCCCTAGG

||| ||||

$Y$ : GGCTCTAGGCC

# Finding overlaps with dynamic programming

Say there are  $d$  reads of length  $n$ , total length  $N = dn$ , and  $a$  is total number of pairs with an overlap

# overlaps to try:  $O(d^2)$

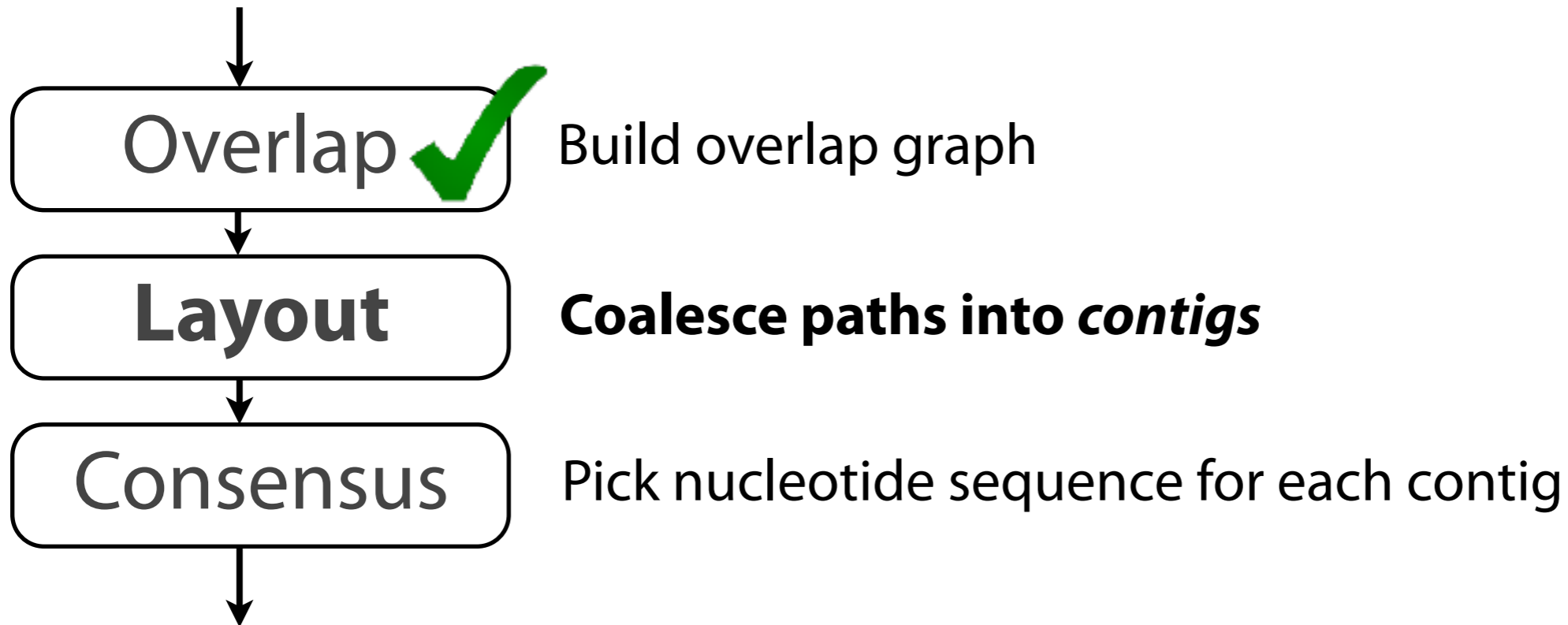
Size of each DP matrix:  $O(n^2)$

Overall:  $O(d^2n^2)$ , or  $O(N^2)$

Contrast  $O(N^2)$  with suffix tree:  $O(N + a)$ , but where  $a$  is worst-case  $O(d^2)$

Real-world overlappers mix the two; index filters out vast majority of non-overlapping pairs, dynamic programming used for remaining pairs

# Overlap Layout Consensus



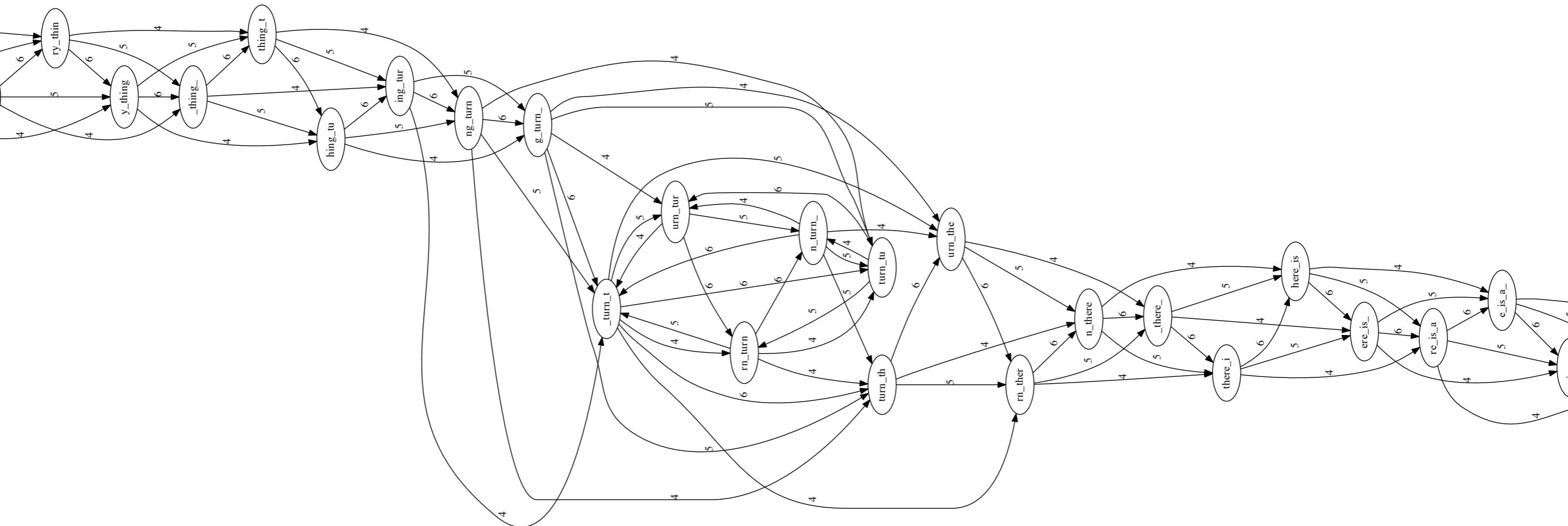
# Layout

Overlap graph is big and messy. Contigs don't "pop out" at us.

Below: part of the overlap graph for

`to_everything_turn_turn_turn_there_is_a_season`

$l = 4, k = 7$

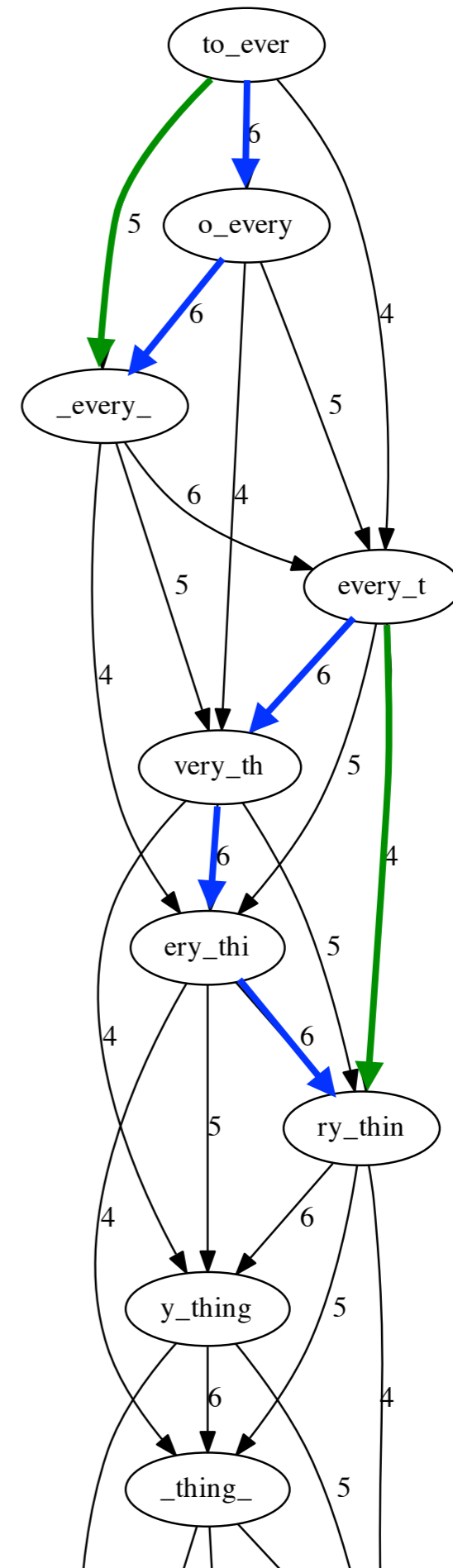


# Layout

Anything redundant about this part of the overlap graph?

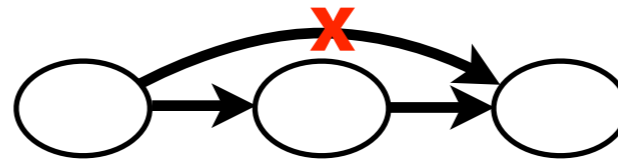
Some edges can be *inferred (transitively)* from other edges

E.g. **green** edge can be inferred from **blue**

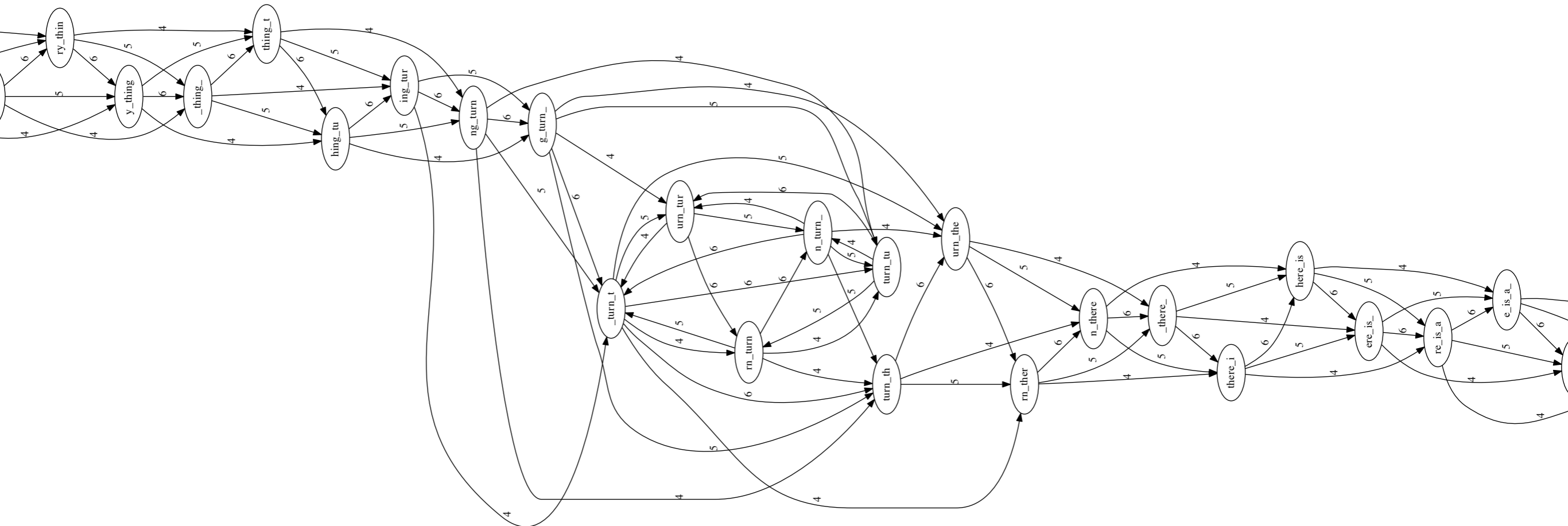


# Layout

Remove transitively inferrable edges, starting with edges that skip one node:



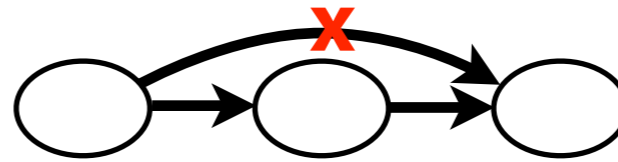
Before:



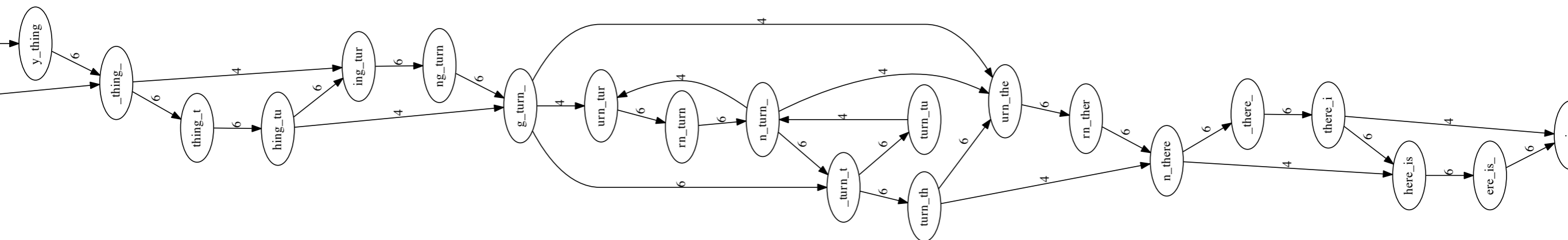


# Layout

Remove transitively inferrable edges, starting with edges that skip one node:

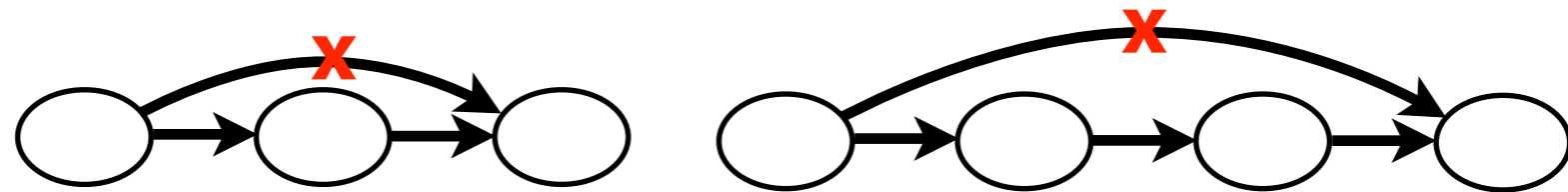


After:

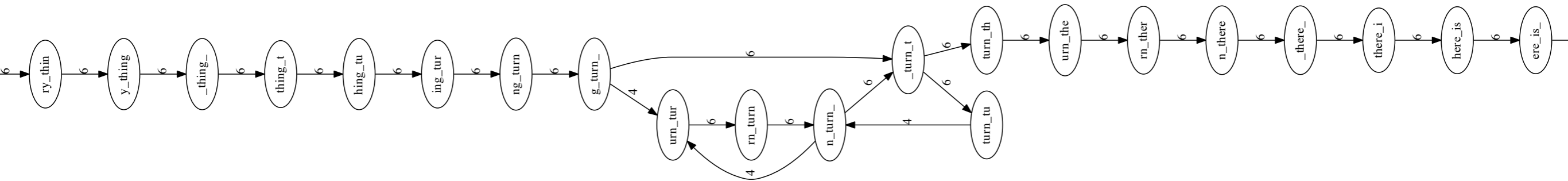


# Layout

Now remove edges that skip one or two nodes:



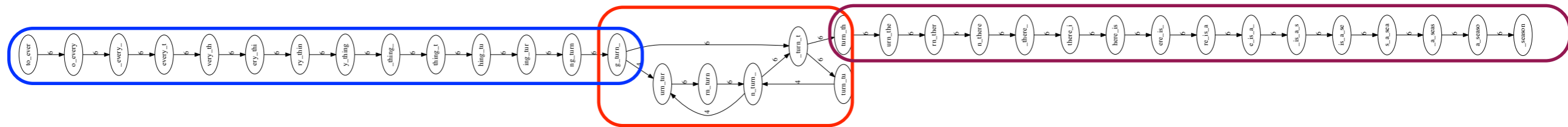
After:



Even simpler

# Layout

Emit *contigs* corresponding to the non-branching stretches



Contig 1

to\_every\_thing\_turn\_

Contig 2

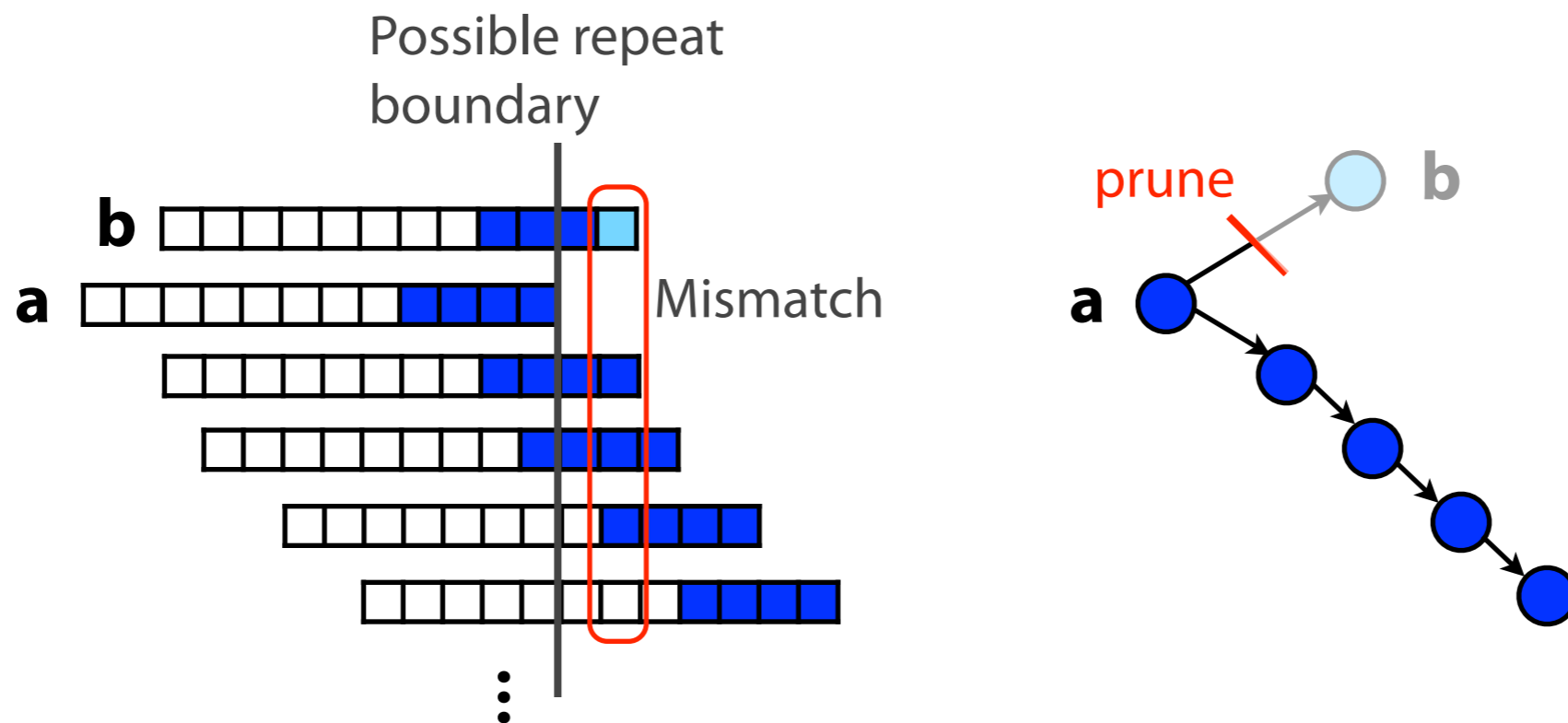
turn\_there\_is\_a\_season



Unresolvable repeat

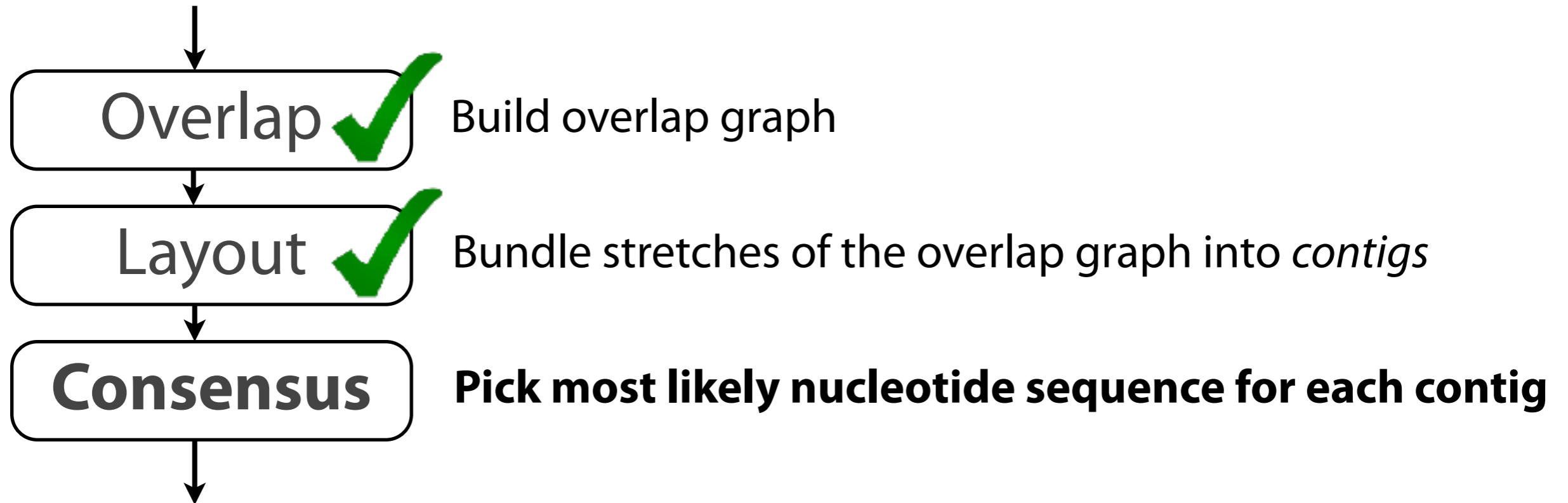
# Layout

Must handle subgraphs that are spurious, e.g. because of sequencing error



Mismatch could be due to sequencing error or repeat. Since the path through **b** ends abruptly we might conclude it's an error and prune **b**.

# Overlap Layout Consensus



# Consensus

TAGATTACACAGATTACTGA TTGATGGCGTAA CTA  
TAGATTACACAGATTACTGACTTGATGGCGTAACTA  
TAG TTACACAGATTATTGACTTCATGGCGTAA CTA  
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA  
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA

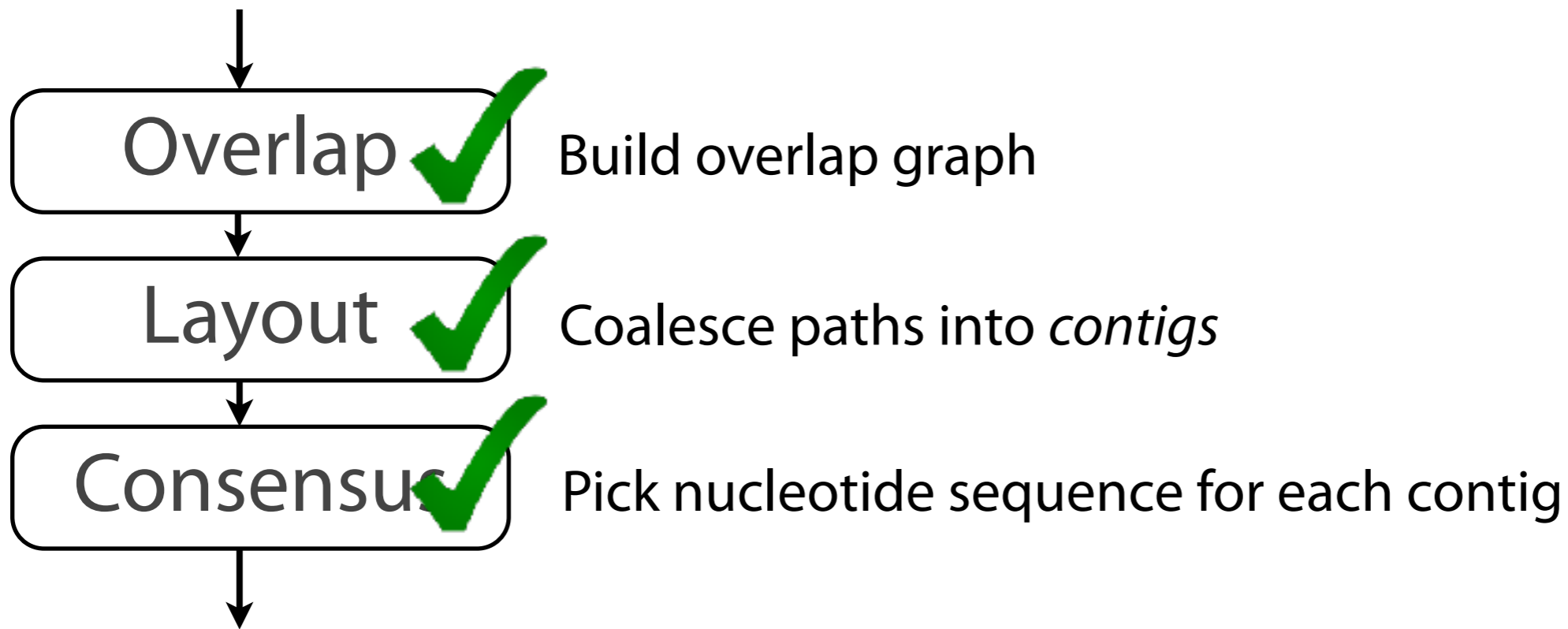
Take reads that make up a contig and line them up

↓ ↓ ↓ ↓ ↓  
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA

Take *consensus*, i.e. majority vote

Complications: (a) sequencing error, (b) ploidy

# Overlap Layout Consensus



## OLC drawbacks

Building overlap graph is slow. We saw  $O(N + a)$  and  $O(N^2)$  approaches.

Overlap graph is big; one node per read, # edges can grow superlinearly with # reads

Sequencing datasets are ~ 100s of millions or billions of reads