Module 2 - Assembly

Lecture 10a: Genomics

Bioinformatics Algorithms CSC4181/6802

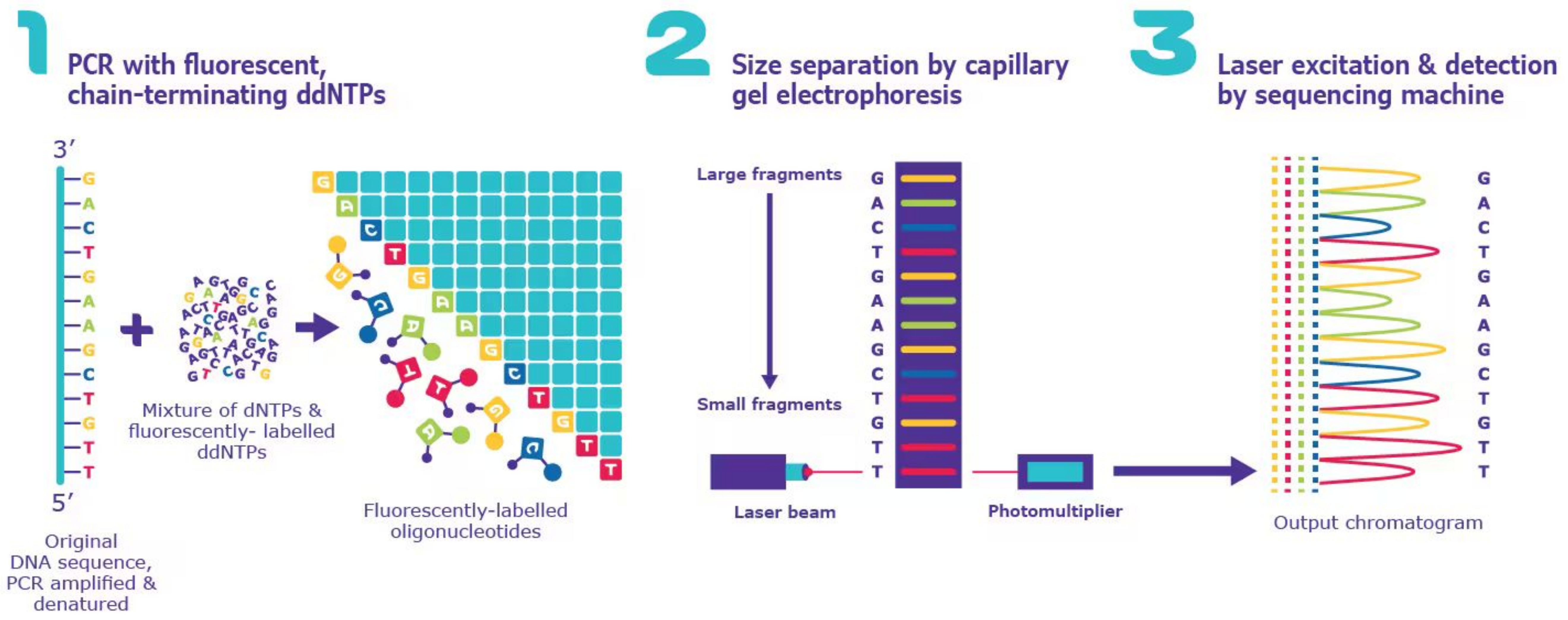
Most slides used are from Ben Langmead's Teaching Materials (www.langmead-lab.org/teaching-materials)

Sequencing Technology

Sanger sequencing
Maxam and Gilbert
Sanger chain termination

https://www.pacb.com/blog/the-evolution-of-dna-sequencing-tools/

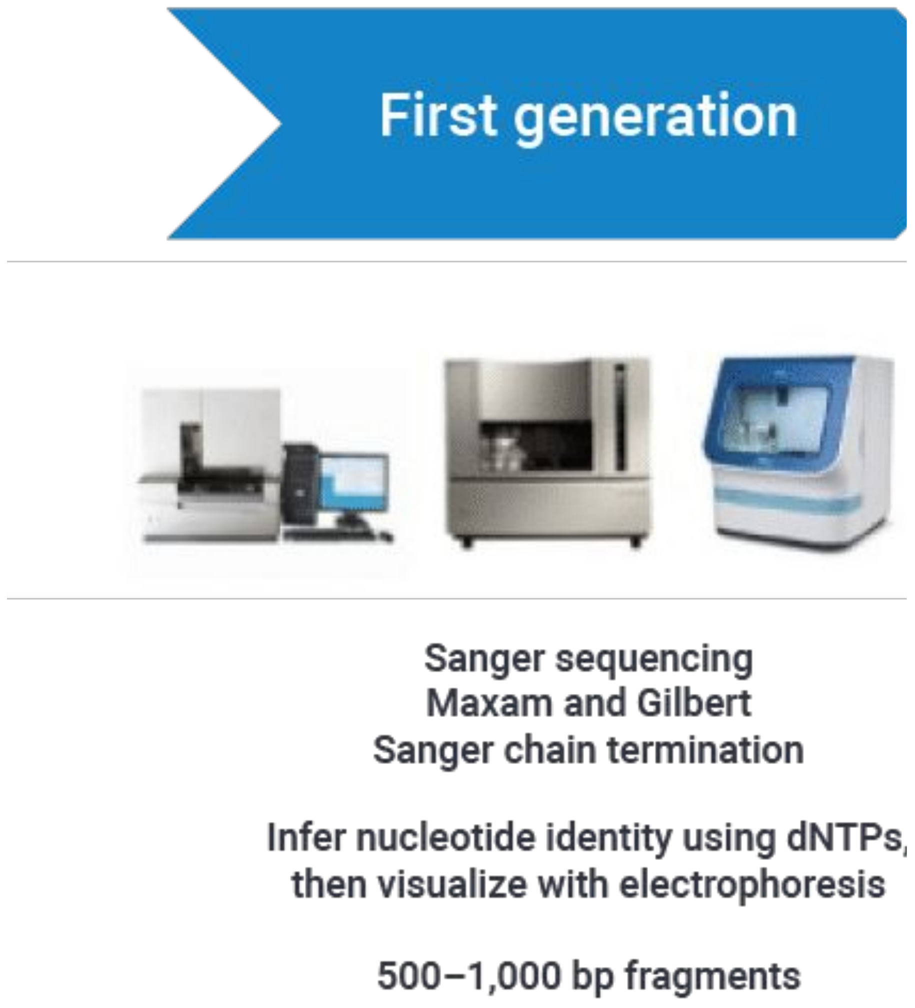
Sanger Sequencing



1 PCR with fluorescent, chain-terminating ddNTPs
2 Size separation by capillary gel electrophoresis
3 Laser excitation & detection by sequencing machine
3′
5′
G A C T G A A G C T G T T
Original DNA sequence, PCR amplified & denatured
Mixture of dNTPs & fluorescently- labelled ddNTPs
Fluorescently-labelled oligonucleotides
Large fragments
Small fragments
Laser beam
Photomultiplier
Output chromatogram

https://www.sigmaaldrich.com/CA/en/technical-documents/protocol/genomics/sequencing/sanger-sequencing

Sequencing Technology



First generation

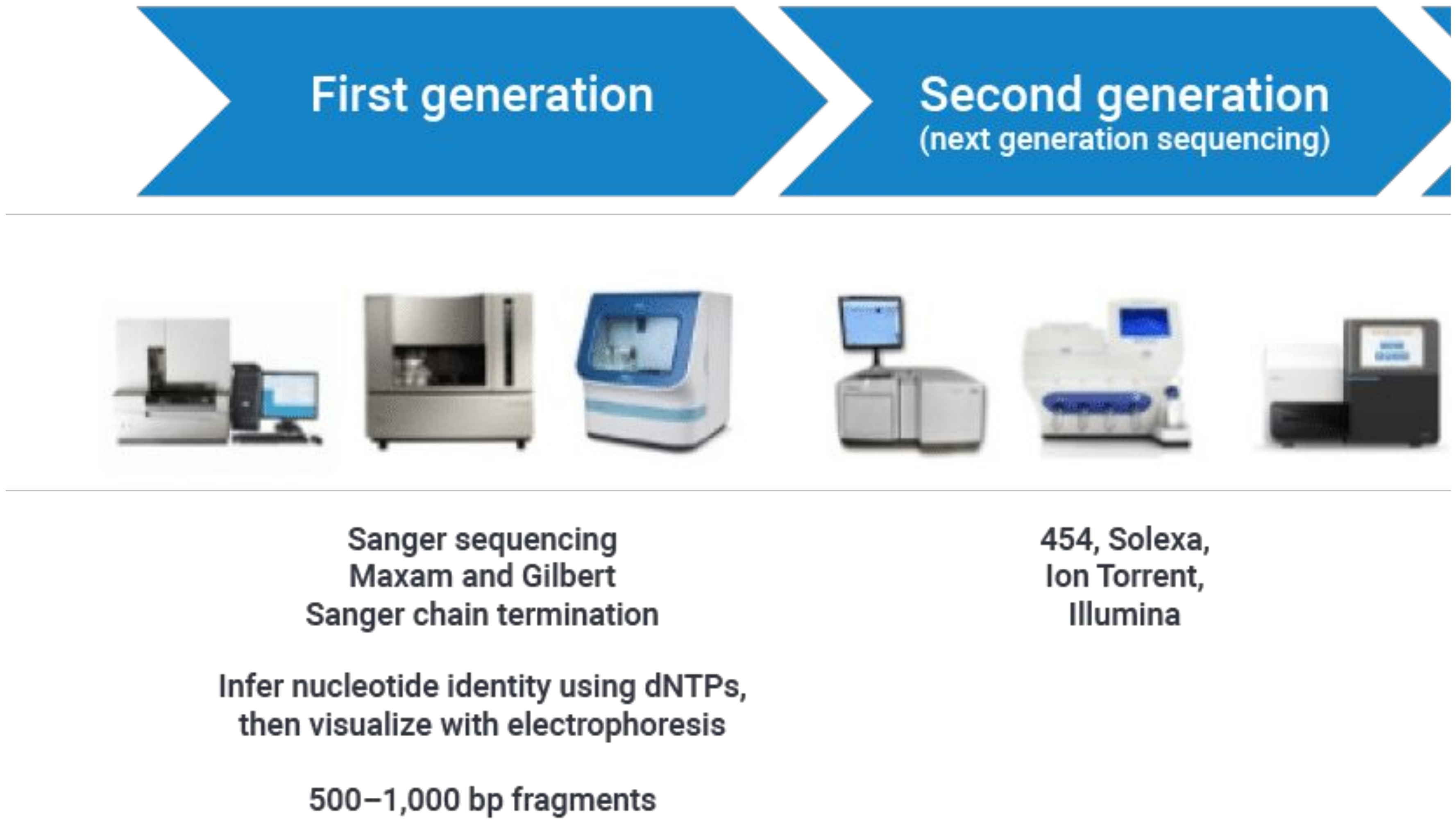
Sanger sequencing
Maxam and Gilbert
Sanger chain termination

Infer nucleotide identity using dNTPs,
then visualize with electrophoresis

500–1,000 bp fragments

https://www.pacb.com/blog/the-evolution-of-dna-sequencing-tools/

Sequencing Technology



Sanger sequencing
Maxam and Gilbert
Sanger chain termination

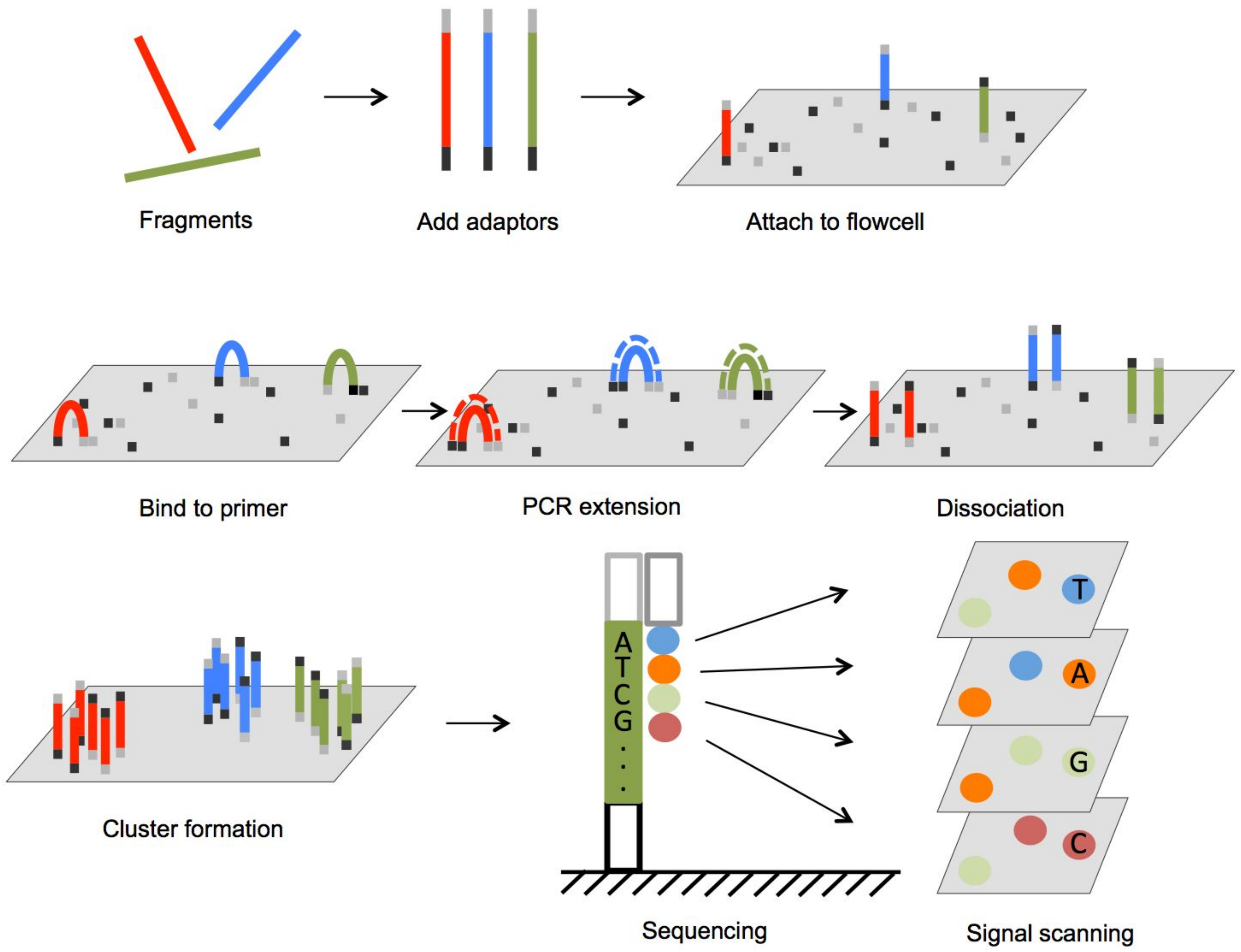
Infer nucleotide identity using dNTPs, then visualize with electrophoresis

500–1,000 bp fragments

454, Solexa, Ion Torrent, Illumina

https://www.pacb.com/blog/the-evolution-of-dna-sequencing-tools/

Sequencing by Synthesis



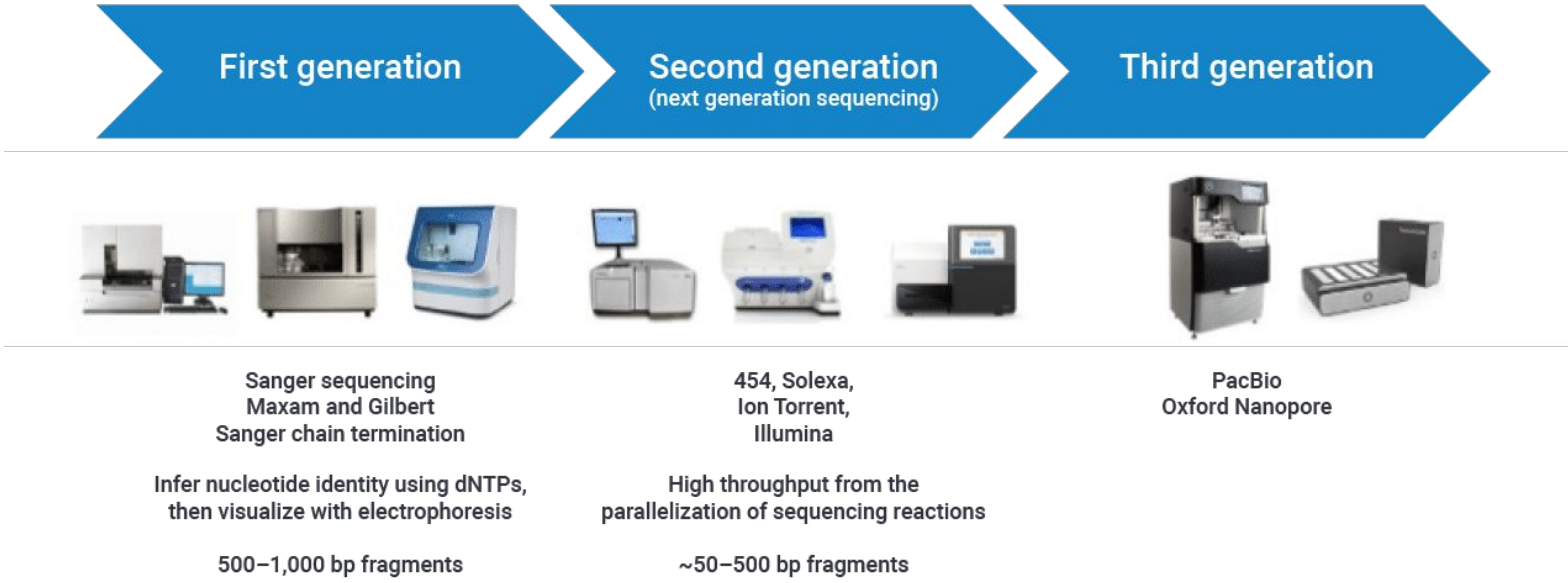
https://www.intechopen.com/chapters/49419

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

https://www.pacb.com/blog/the-evolution-of-dna-sequencing-tools/

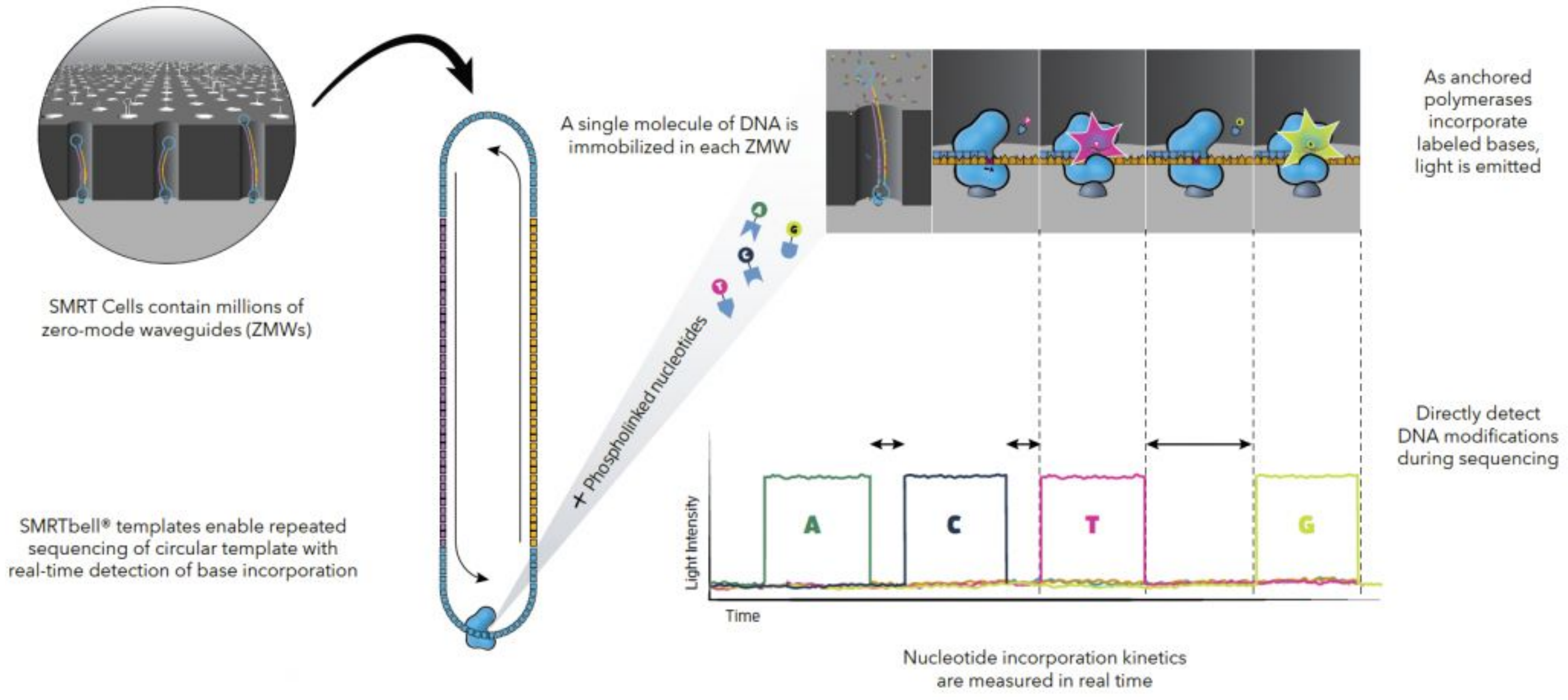
Sequencing Technology



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	
500–1,000 bp fragments	~50–500 bp fragments	

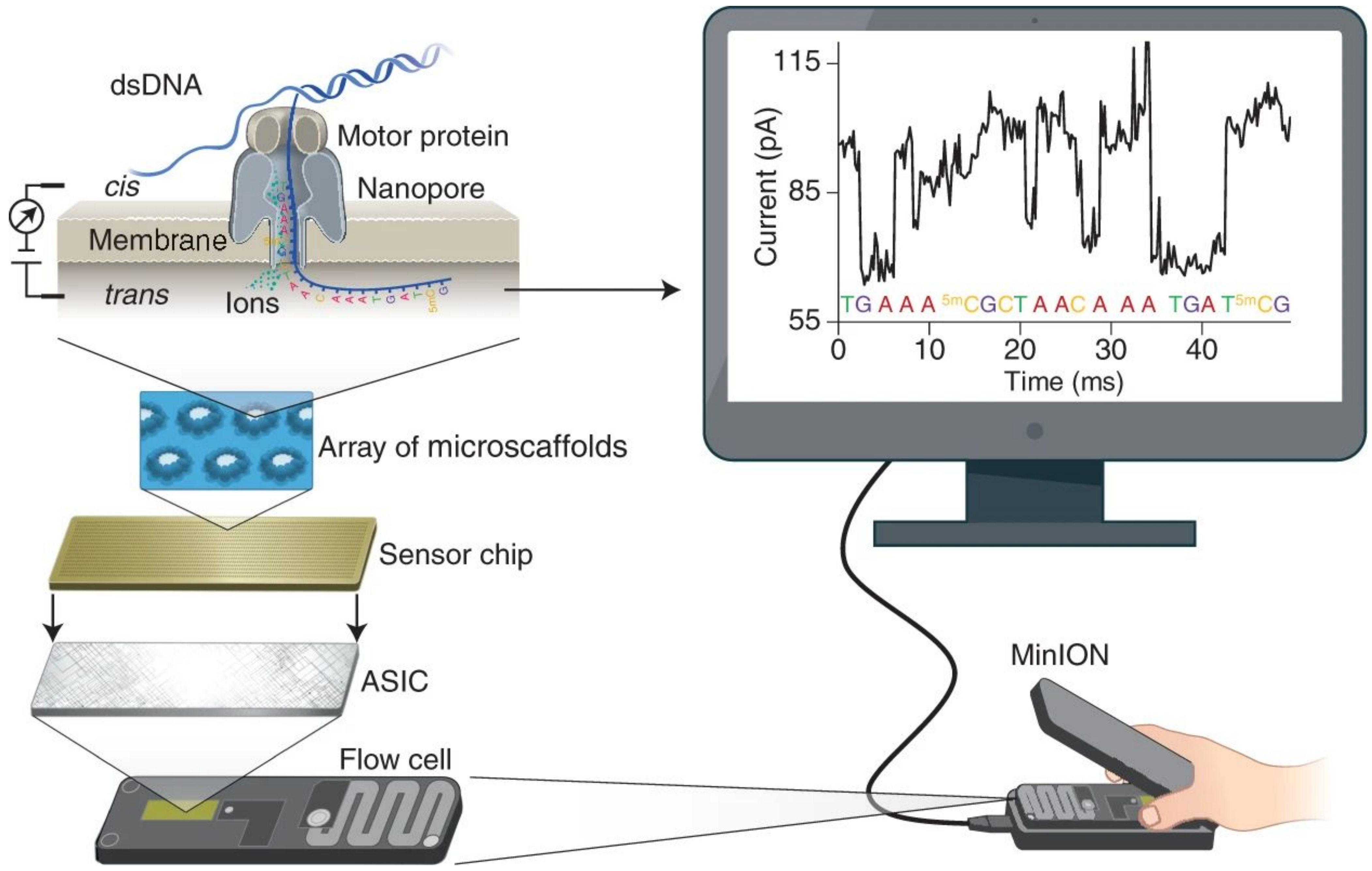
https://www.pacb.com/blog/the-evolution-of-dna-sequencing-tools/

PacBio Sequencing


SMRT Cells contain millions of zero-mode waveguides (ZMWs)
SMRTbell® templates enable repeated sequencing of circular template with real-time detection of base incorporation
A single molecule of DNA is immobilized in each ZMW
+ Phospholinked nucleotides
As anchored polymerases incorporate labeled bases, light is emitted
Directly detect DNA modifications during sequencing
A
C
T
G
Light Intensity
Time
Nucleotide incorporation kinetics are measured in real time

https://www.pacb.com/wp-content/uploads/SMRT-Sequencing-Brochure-Delivering-highly-accurate-long-reads-to-drive-discovery-in-life-science.pdf

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

https://www.pacb.com/blog/the-evolution-of-dna-sequencing-tools/

Sequencing Technology

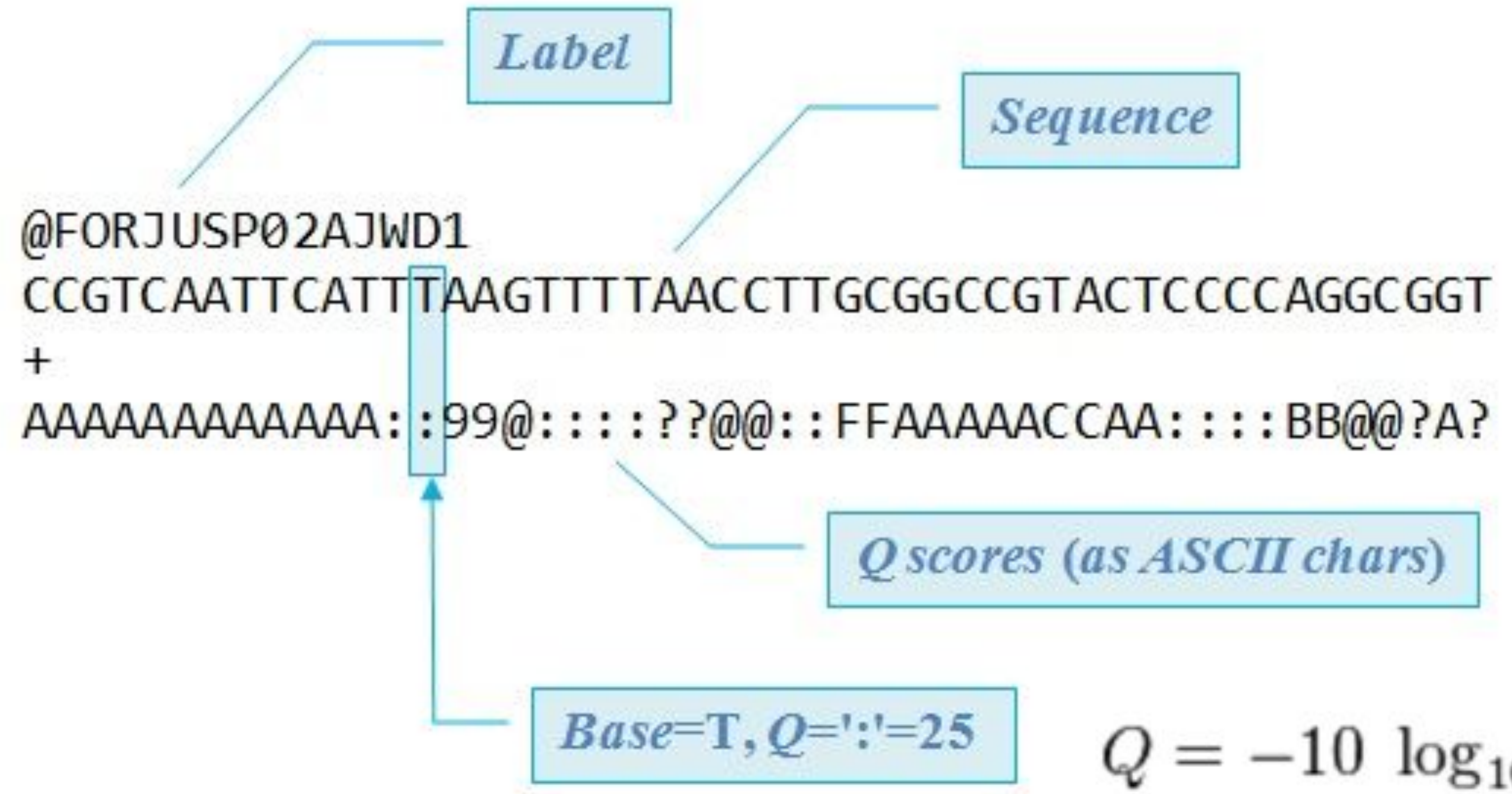
First generation	Second generation (next generation sequencing)	Third generation
Sanger sequencing Maxam and Gilbert Sanger chain termination	454, Solexa, Ion Torrent, Illumina	PacBio Oxford Nanopore
Infer nucleotide identity using dNTPs, then visualize with electrophoresis	High throughput from the parallelization 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

https://www.pacb.com/blog/the-evolution-of-dna-sequencing-tools/

Capturing measurement error: FASTQ



```
@FORJUSP02AJWD1
CCGTCAATTCATTTAAGTTTTAACCTTGCGGCCGTACTCCCCAGGCGGT
+
AAAAAAAAAAAA::99@::::??@@::FFAAAAACCAA::::BB@@?A?
```

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

$$Q = -10 \log_{10} P \quad \Longrightarrow \quad P = 10^{\frac{-Q}{10}}$$

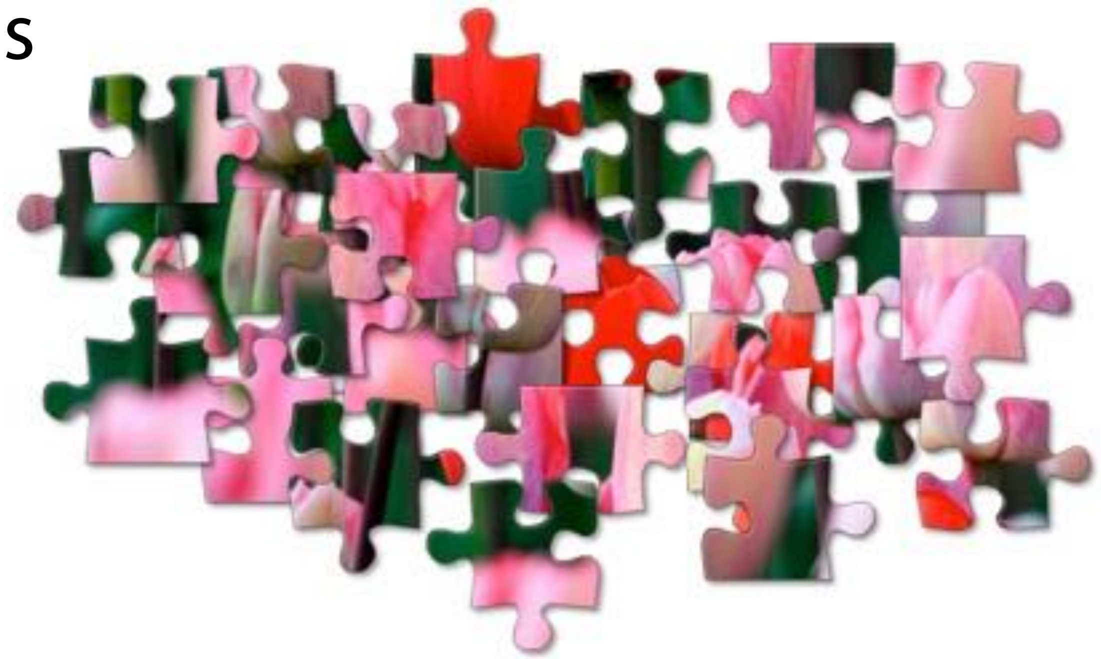
Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10000	99.99%
50	1 in 100000	99.999%

https://www.drive5.com/usearch/manual/fastq_files.html

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

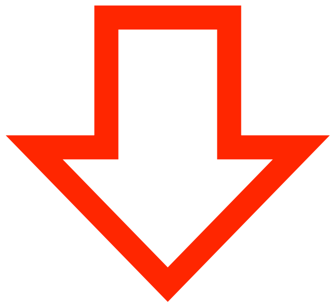
Assembly

Reads



+

Reference genome

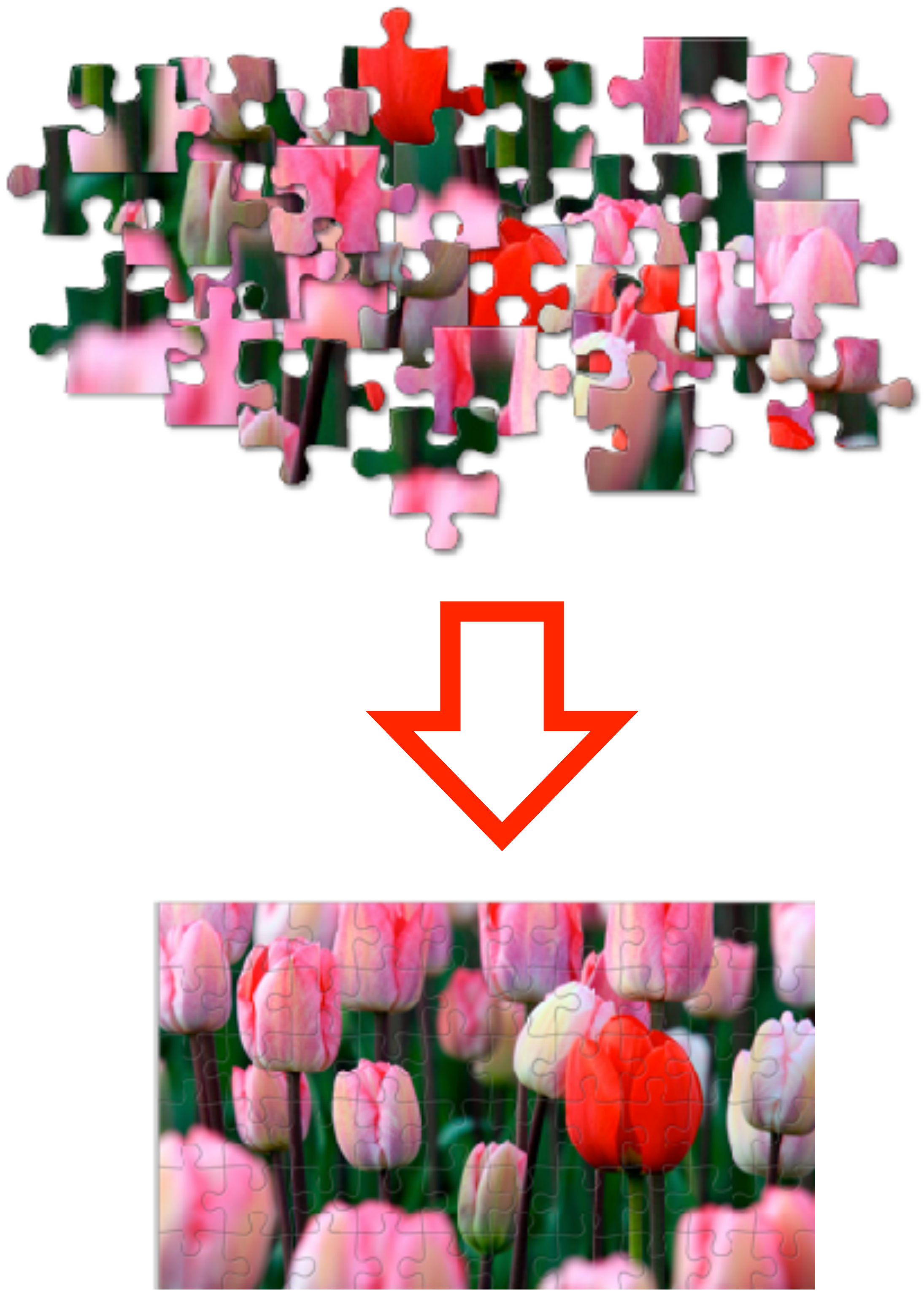


Input DNA

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:   GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Copy:    GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
         GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
         GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
         GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

Then fragments it:

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

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

Assembly

From these

```
                  CTAGGCCCTCAATTTTT
                CTCTAGGCCCTCAATTTTT
              GGCTCTAGGCCCTCATTTTTT
           CTCGGCTCTAGCCCCTCATTTT
        TATCTCGACTCTAGGCCCTCA
        TATCTCGACTCTAGGCC
    TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
```

Reconstruct this

```
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

Assembly

```
CTAGGCCCTCAATTTTT
GGCGTCTATATCT
CTCTAGGCCCTCAATTTTT
TCTATATCTCGGCTCTAGG
GGCTCTAGGCCCTCATTTTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
GGCGTCGATATCT
TATCTCGACTCTAGGCC
GGCGTCTATATCTCG
```

From these

Reconstruct this

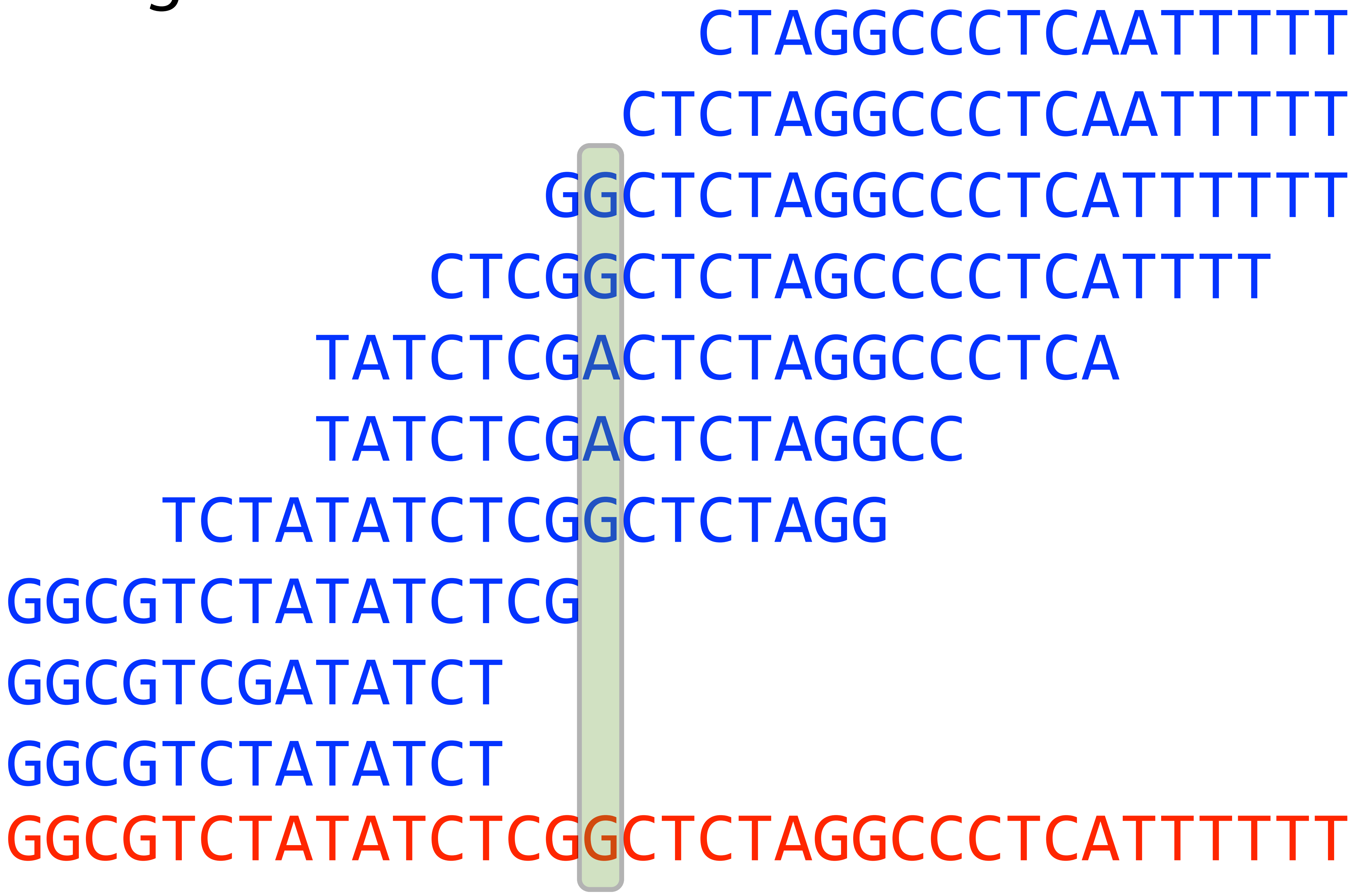
???????????????????????????????????

Coverage

```
                  CTAGGCCCTCAATTTTT
                CTCTAGGCCCTCAATTTTT
              GGCTCTAGGCCCTCATTTTTT
           CTCGGCTCTAGCCCCTCATTTT
        TATCTCGACTCTAGGCCCTCA
        TATCTCGACTCTAGGCC
    TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

Coverage = 5

Coverage



```
                  CTAGGCCCTCAATTTTT
                CTCTAGGCCCTCAATTTTT
              GGCTCTAGGCCCTCATTTTTT
           CTCGGCTCTAGCCCCTCATTTT
        TATCTCGACTCTAGGCCCTCA
        TATCTCGACTCTAGGCC
    TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

Coverage = 5

```
                  CTAGGCCCTCAATTTTT
                CTCTAGGCCCTCAATTTTT
              GGCTCTAGGCCCTCATTTTTTT
           CTCGGCTCTAGCCCCTCATTTT
        TATCTCGACTCTAGGCCCTCA
        TATCTCGACTCTAGGCC              177 bases
    TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT                           35 bases
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

Average coverage = 177 / 35 ≈ 5-fold

```
TCTATATCTCGGCTCTAGG
    TATCTCGACTCTAGGCC
```

```
TCTATATCTCGGGCTCTAGG
    ||||||| |||||||
    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
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
        TATCTCGACTCTAGGCC
```

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

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

```
                  CTAGGCCCTCAATTTTT
           CTCGGCTCTAGCCCCTCATTTT
    TCTATATCTCGGCTCTAGG
GGCGTCGATATCT
```

less coverage

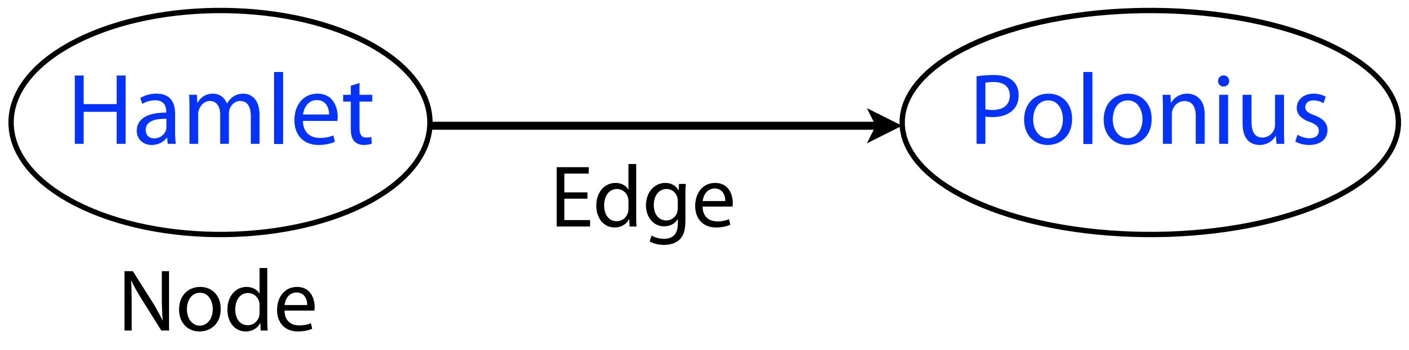
```
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
                  CTAGGCCCTCAATTTTT
              GGCTCTAGGCCCTCATTTTTT
           CTCGGCTCTAGCCCCTCATTTT
        TATCTCGACTCTAGGCCCTCA
    TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCTATATCT
```

more coverage

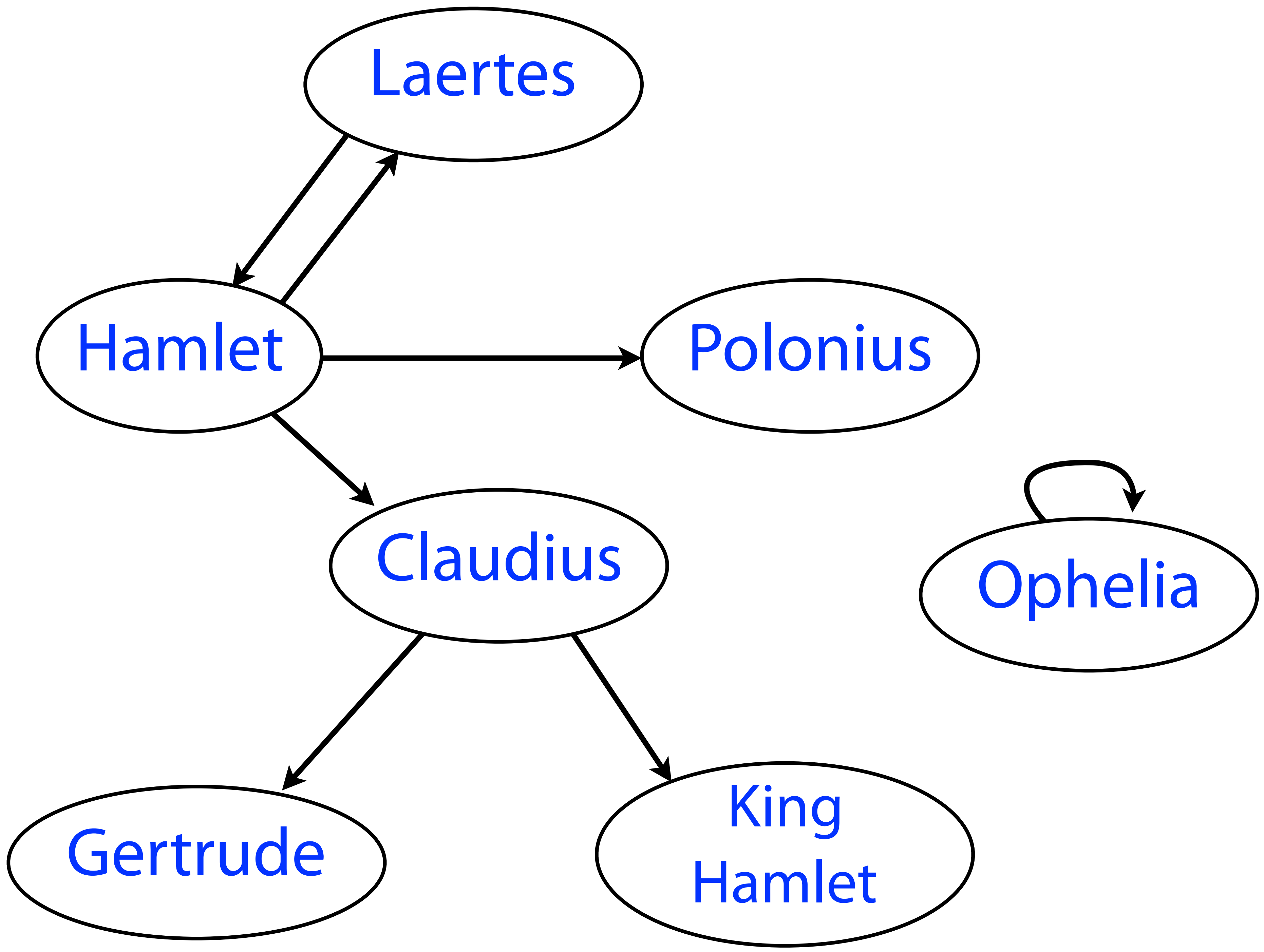
```
TCTATATCTCGGCTCTAGG
    ||||||| |||||||
    TATCTCGACTCTAGGCC
```

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


    TATCTCGACTCTAGGCC
       |||| |||||| ||
       CTCGGCTCTAGCCCCTCAT
```

Directed graph



Directed graph



Overlap graph

Each node is a read

$$\boxed{\texttt{CTCGGCTCTAGCCCCTCATTTT}}$$

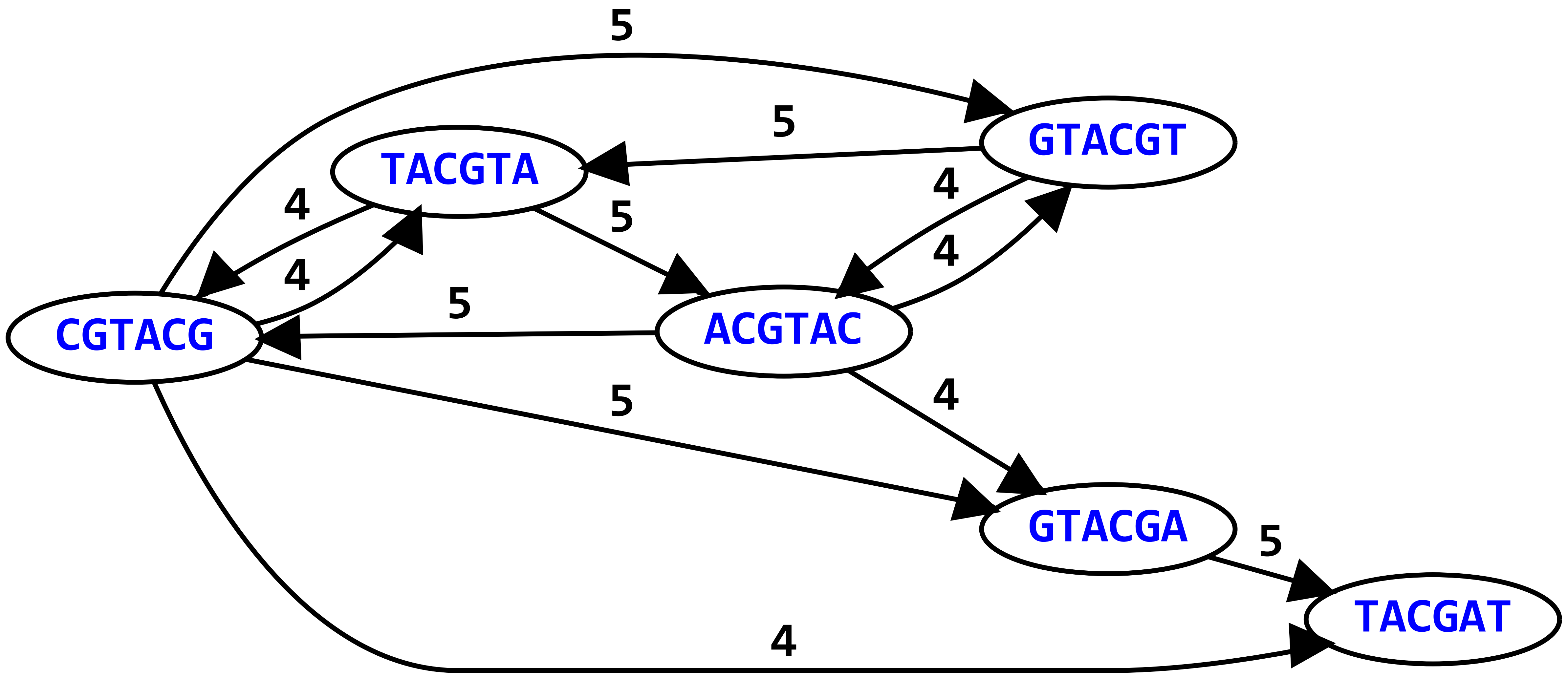
Draw edge A -> B when suffix of A overlaps prefix of B

$$\begin{tikzcd}
\boxed{\texttt{CTCGGCTCTAGCCCCTCATTTT}} \arrow[d] \\
\boxed{\texttt{GGCTCTAGGCCCTCATTTTTTT}}
\end{tikzcd}$$

Overlap graph

Nodes: all 6-mers from GTACGTACGAT

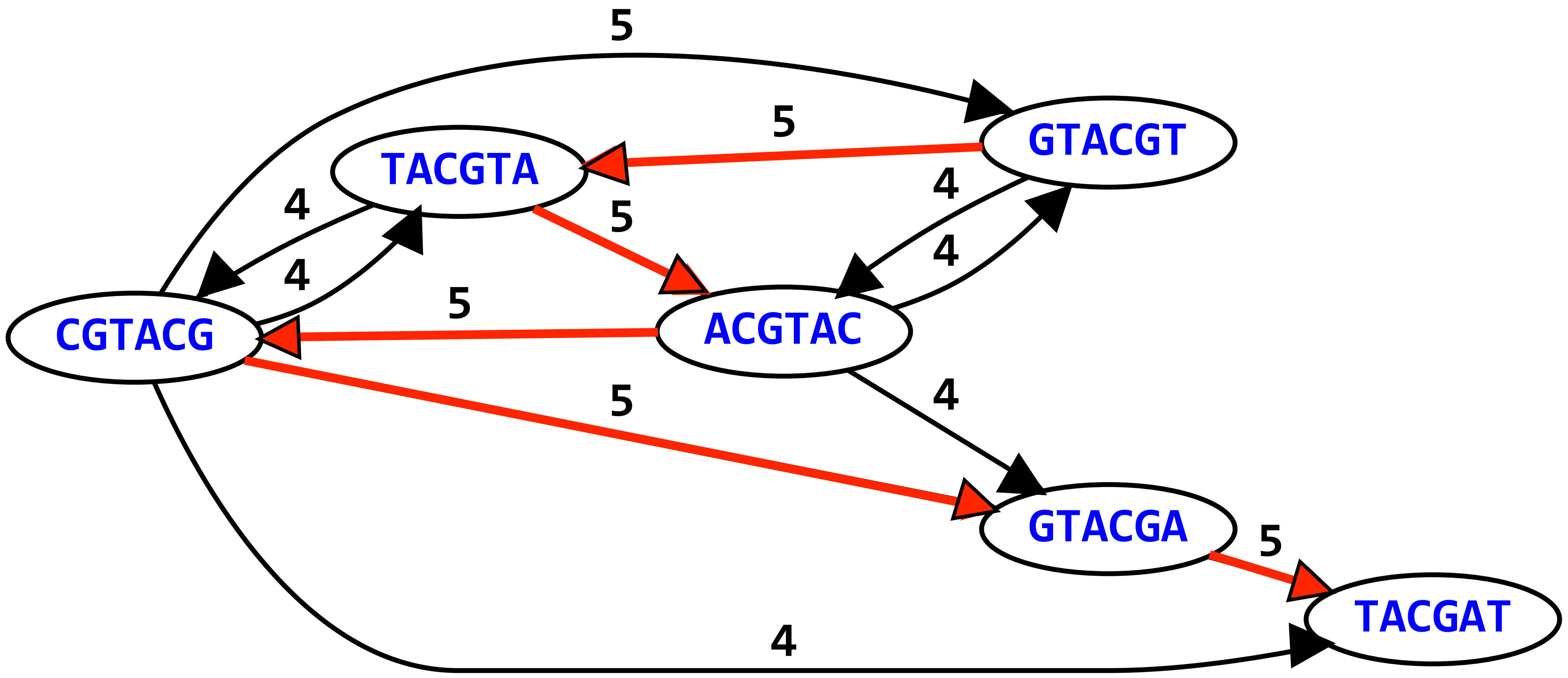
Edges: overlaps of length ≥4



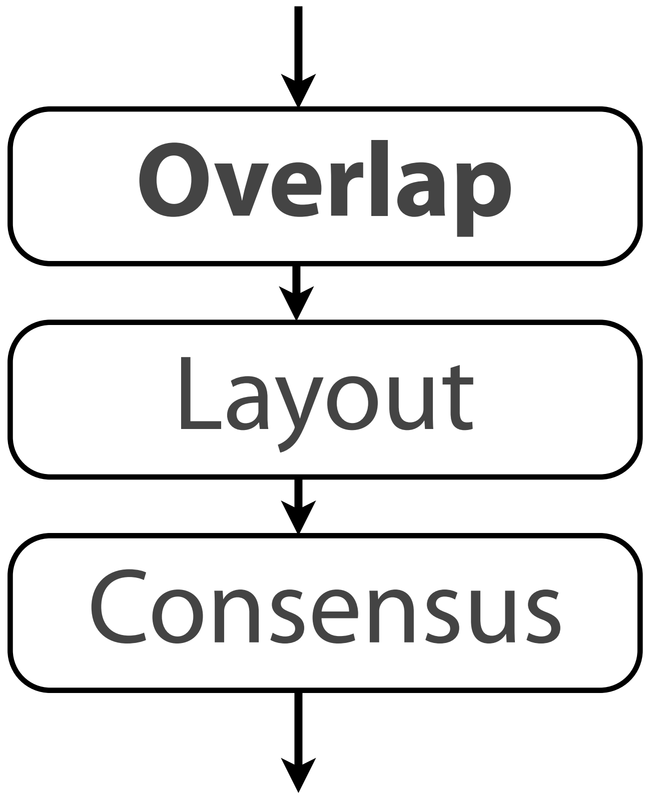
Overlap graph

Nodes: all 6-mers from GTACGTACGAT

Edges: overlaps of length ≥4



Overlap Layout Consensus



Build overlap graph

Coalesce paths into *contigs*

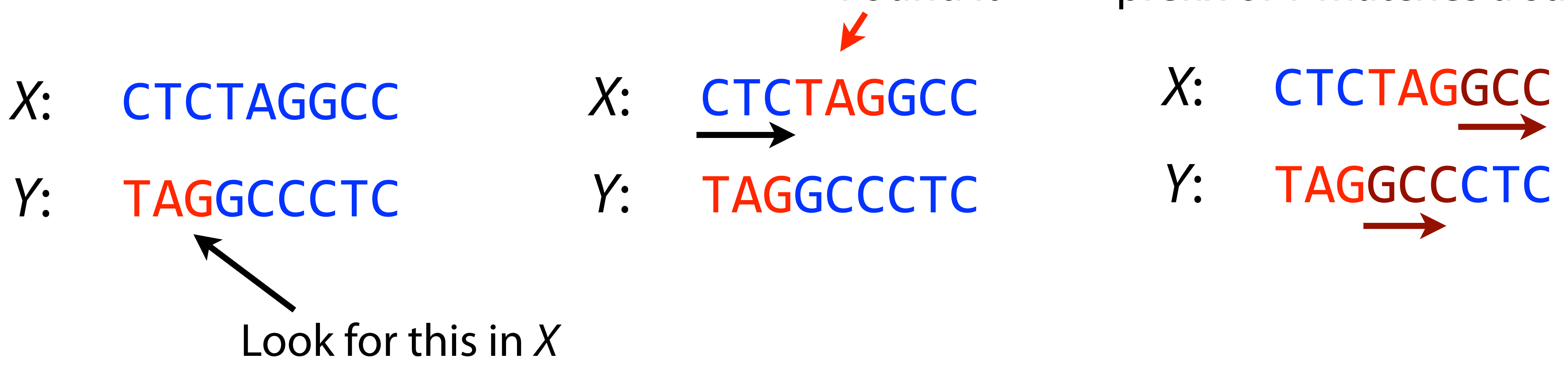
Pick nucleotide sequence for each contig

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 $l = 3$



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

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

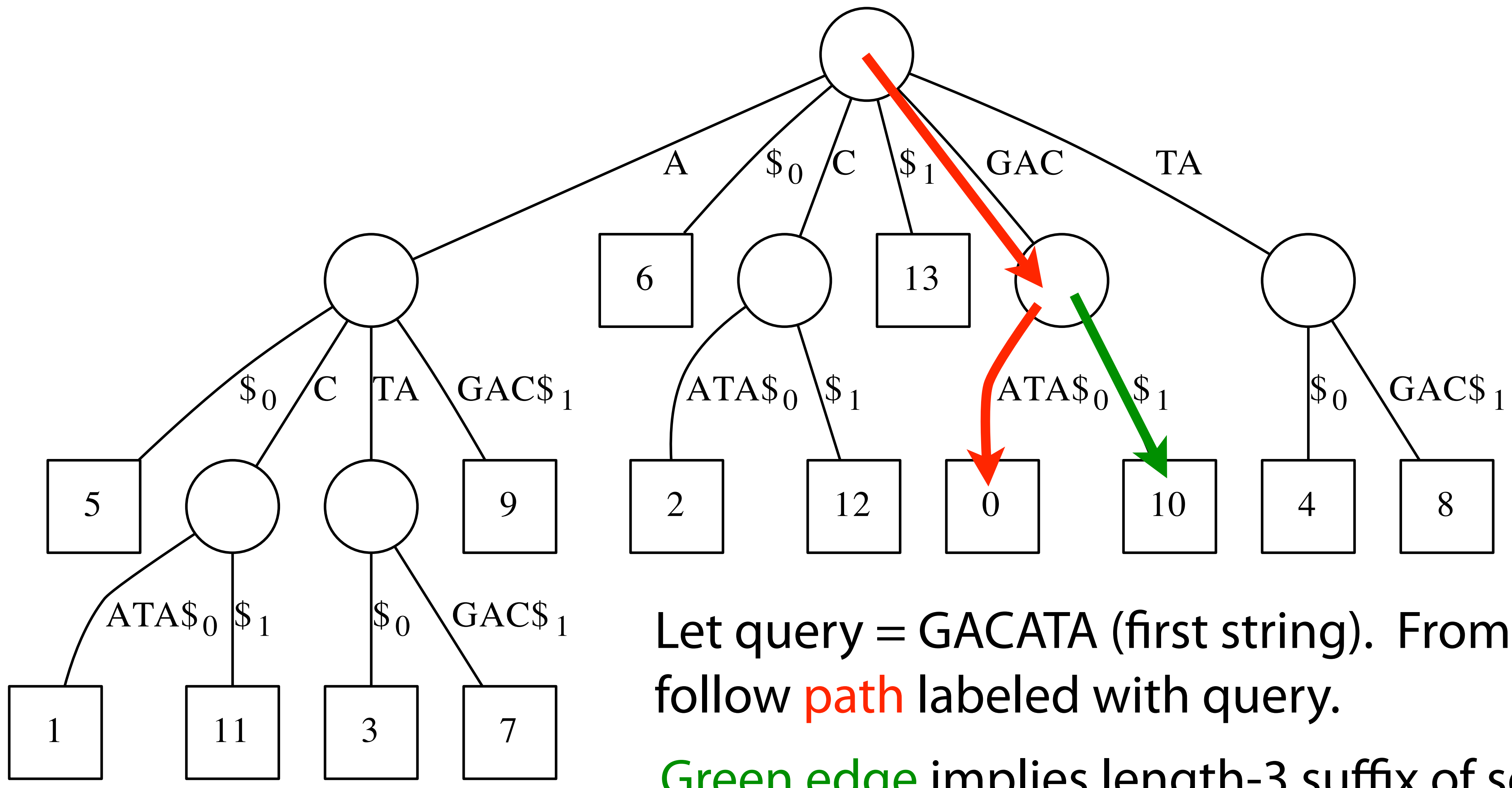
Finding overlaps

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$_0$ATAGAC$_1$



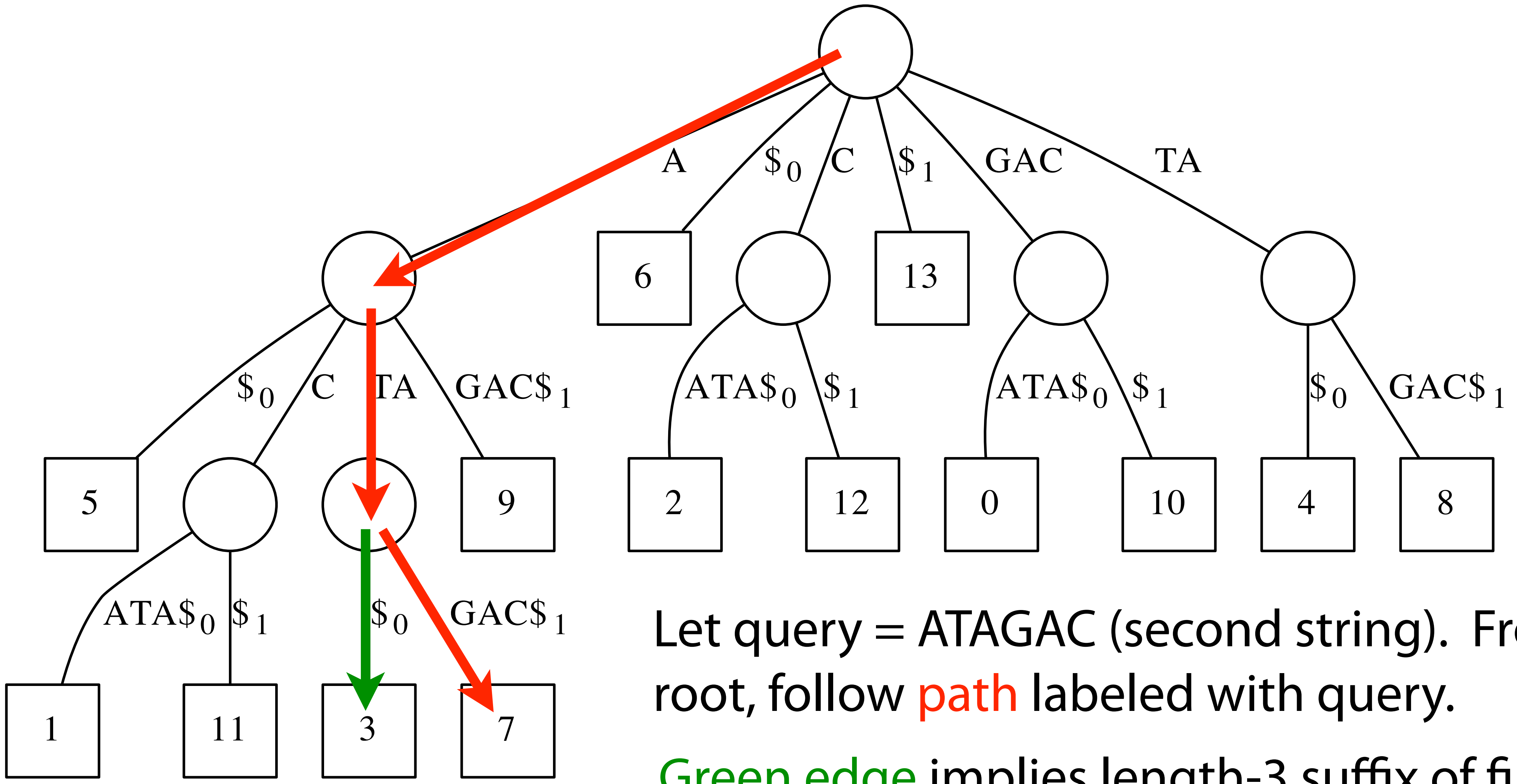
Let query = GACATA (first string). From root, follow path labeled with query.

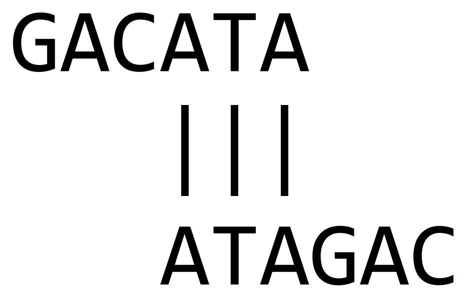
Green edge implies length-3 suffix of second string equals length-3 prefix of query

```
ATAGAC
   |||
   GACATA
```

Finding overlaps with suffix tree

Generalized suffix tree for { "GACATA", "ATAGAC" }   GACATA$\$_0$ATAGAC$\$_1$



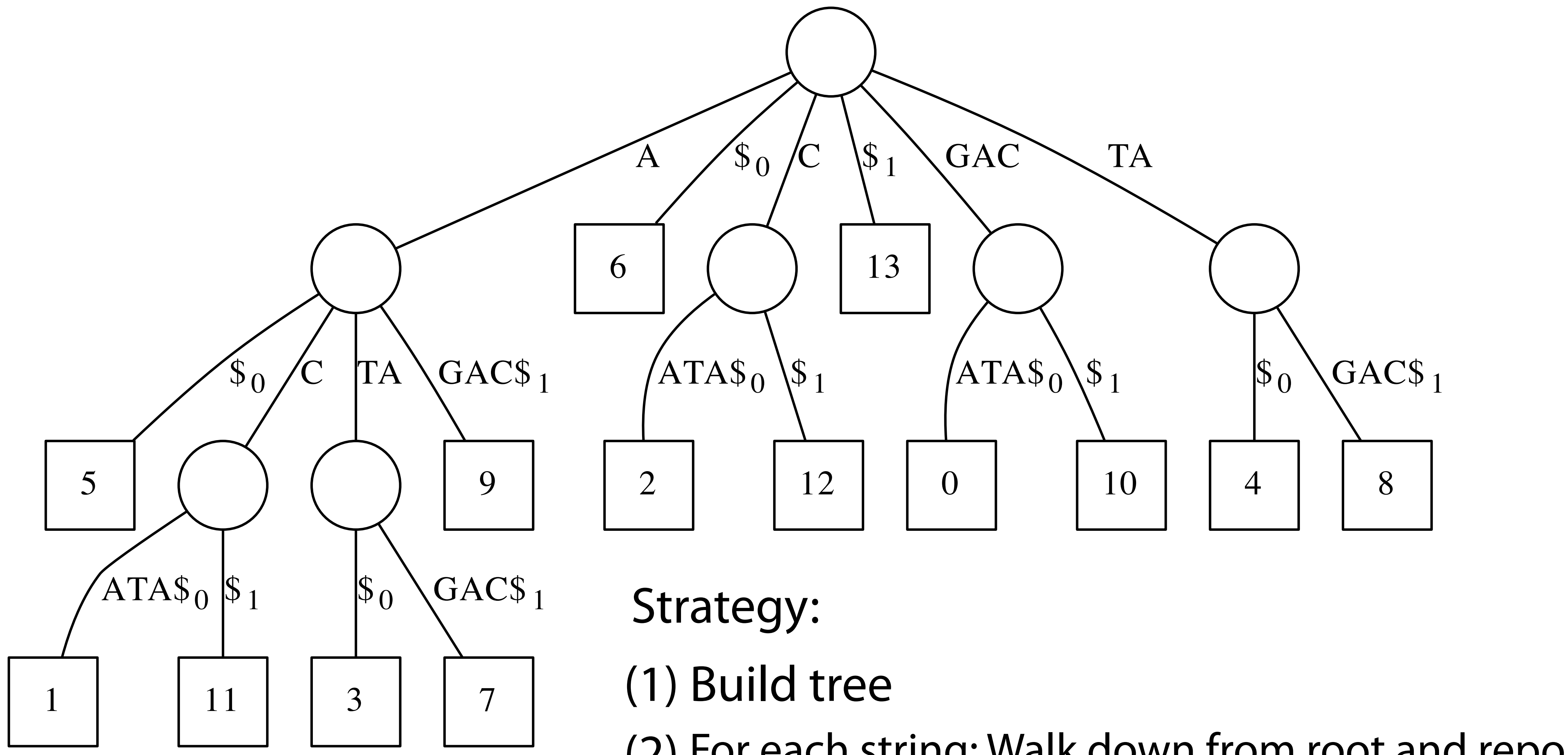


Let query = ATAGAC (second string). From root, follow path labeled with query.

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$\$_0$ATAGAC$\$_1$

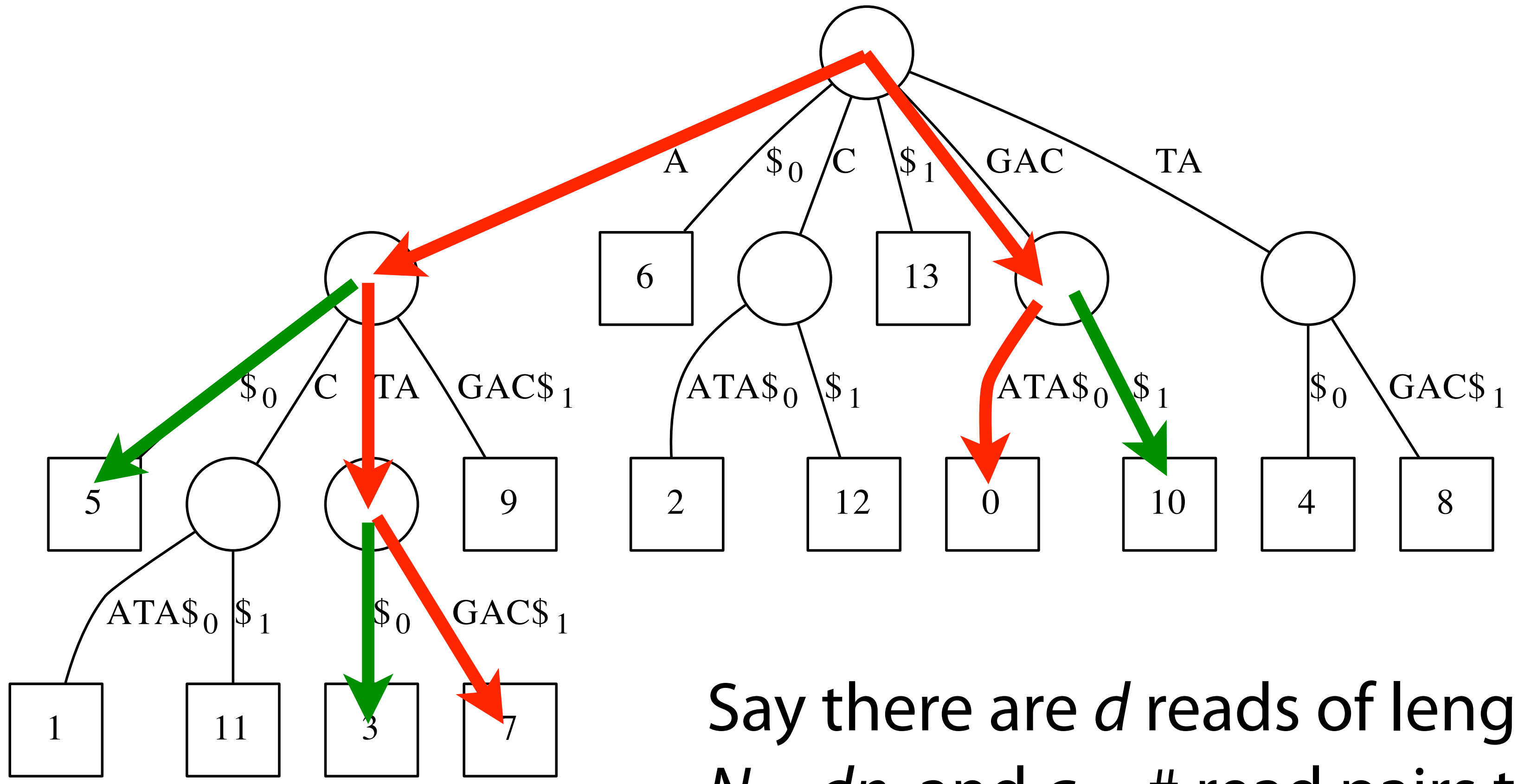


Strategy:

(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:    GGCTCTAGGCCC
```

Dynamic programming

Finding overlaps with dynamic programming

```
X: CTCGGCCCTAGG
      ||| |||||
Y:    GGCTCTAGGCCC
```

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

$s(a, b)$

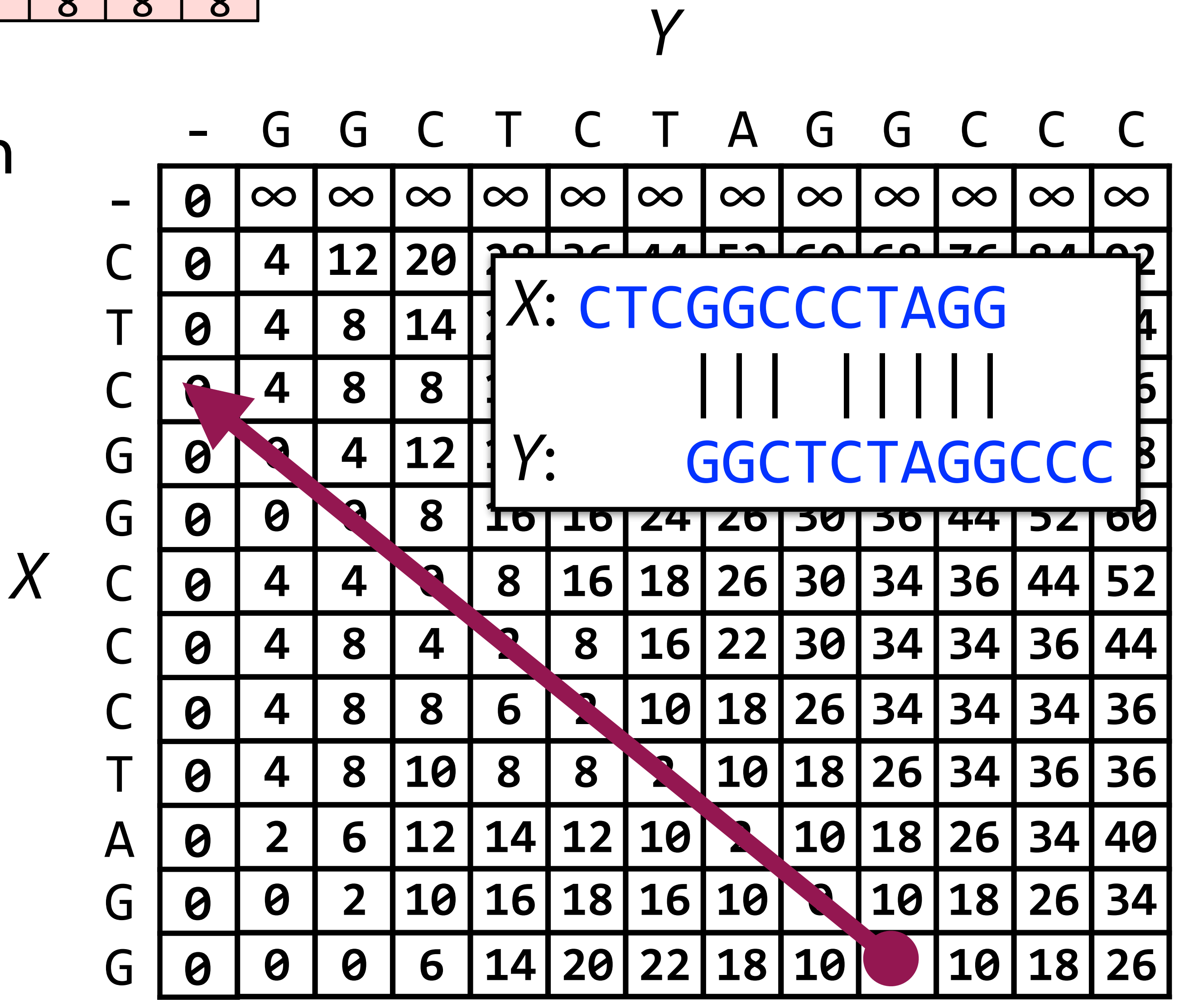
	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	

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 ∞s (must be a prefix of *Y*)

Backtrace from last row



Y

X

	-	G	G	C	T	C	T	A	G	G	C	C	C
-	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
C	0	4	12	20	28	36	44	52	60	68	76	84	92
T	0	4	8	14	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
C	0	4	8	8	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
G	0	0	4	12	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
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	2	8	16	22	30	34	34	36	44
C	0	4	8	8	6	2	10	18	26	34	34	34	36
T	0	4	8	10	8	8	2	10	18	26	34	36	36
A	0	2	6	12	14	12	10	2	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	[illegible]	10	18	26

```
X: CTCGGCCCTAGG
       |||  |||||
Y:     GGCTCTAGGCCC
```

Finding overlaps with dynamic programming

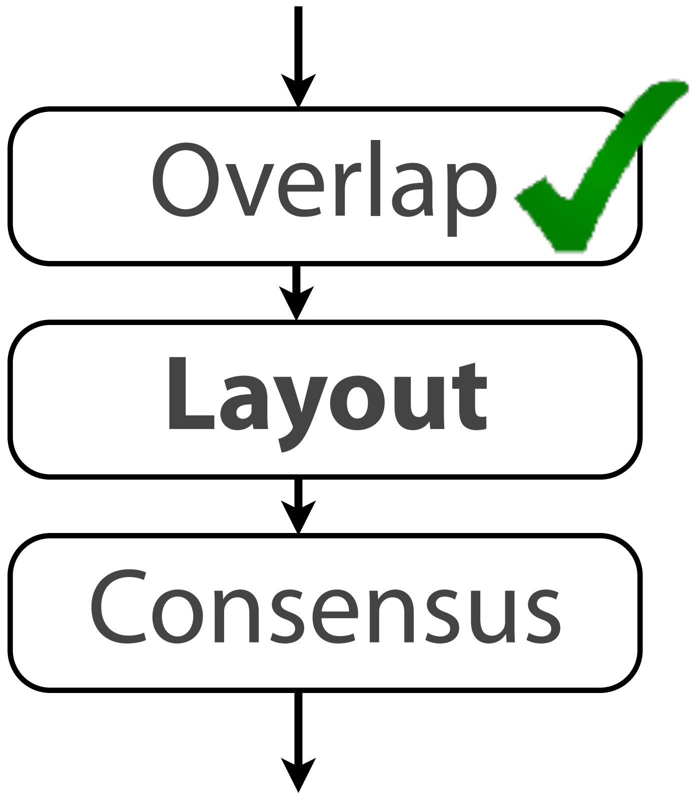
Say there are d reads of length n, total length $N = dn$, and a is total number of pairs with an overlap

# 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



Build overlap graph

Coalesce paths into *contigs*

Pick nucleotide sequence for each contig

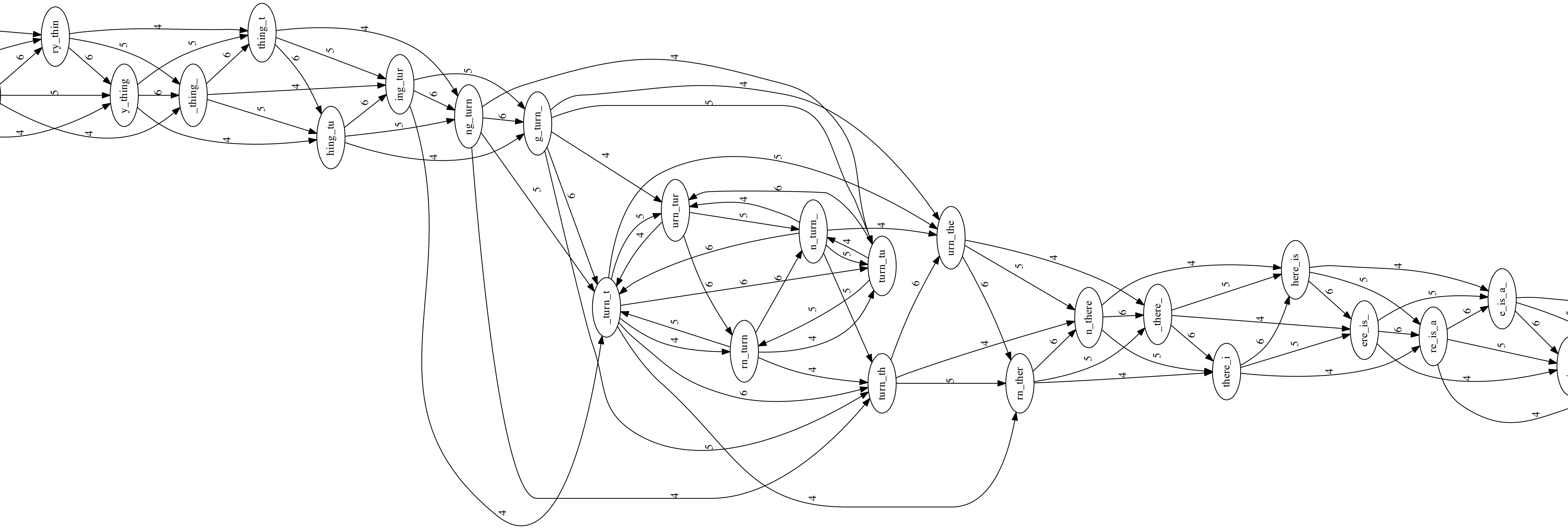
Layout

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

Below: part of the overlap graph for

`to_every_thing_turn_turn_turn_there_is_a_season`

$l = 4, k = 7$

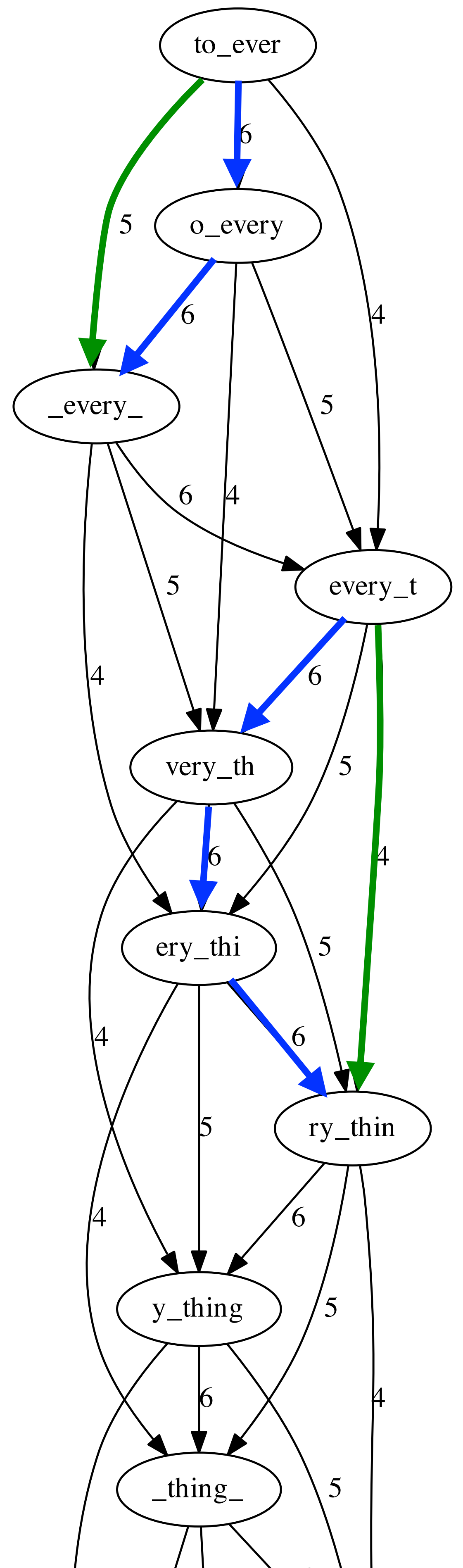


Layout

Anything redundant about this part of the overlap graph?

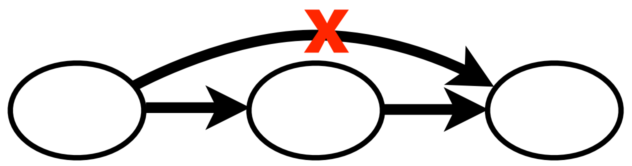
Some edges can be *inferred* (*transitively*) from other edges

E.g. green edge can be inferred from blue

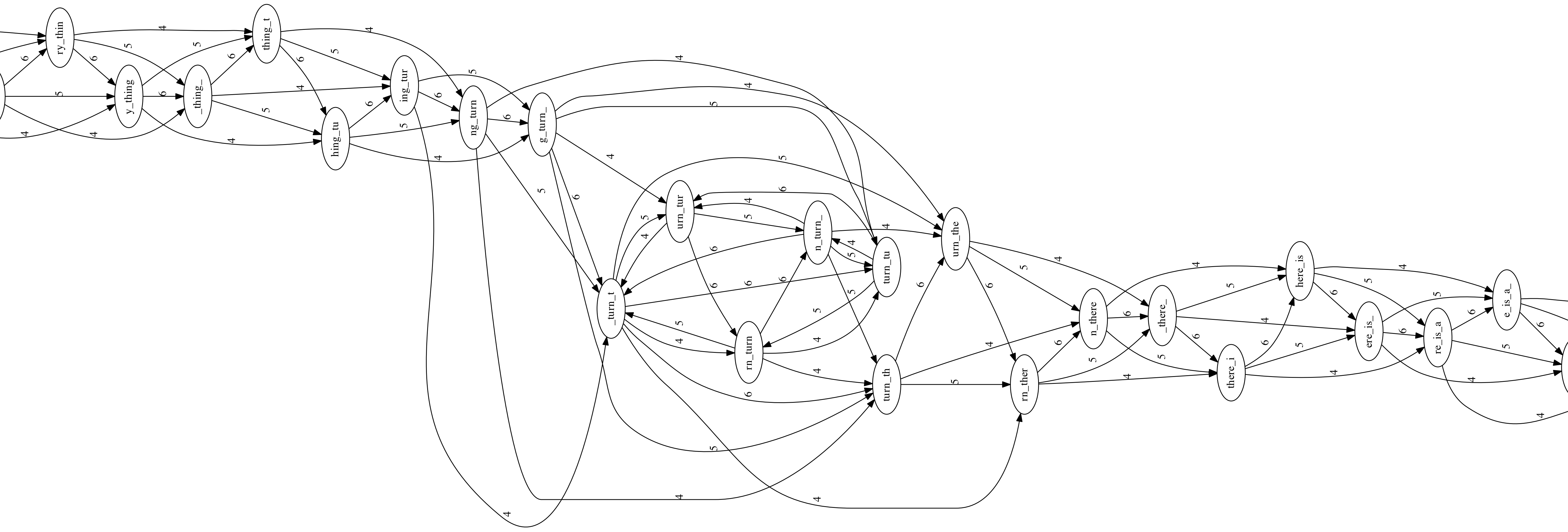


Layout

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

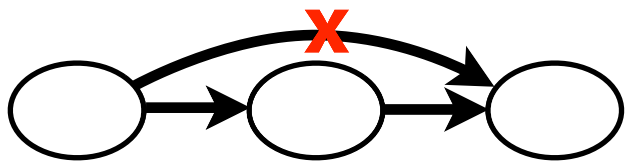


Before:

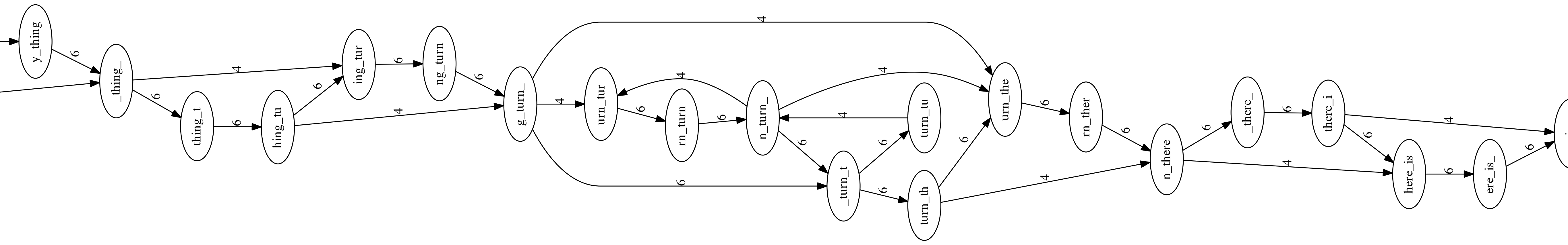


Layout

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

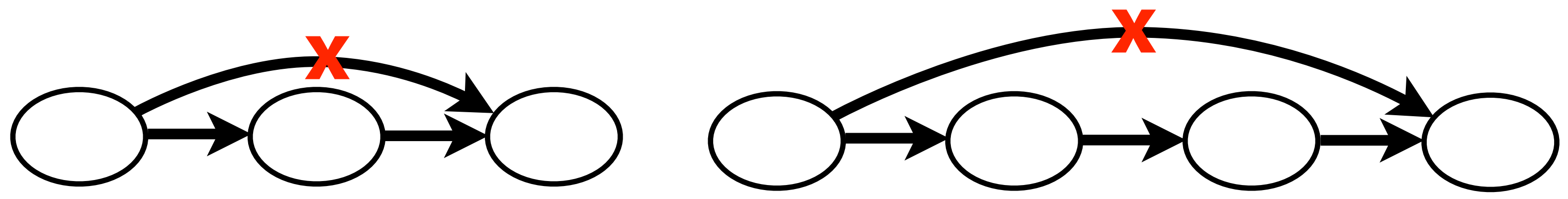


After:

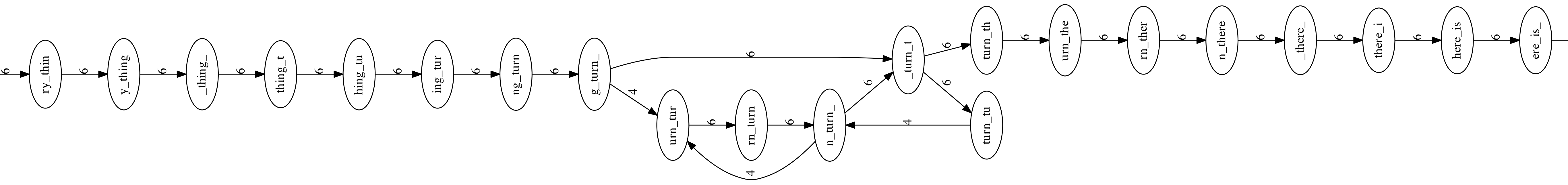


Layout

Now remove edges that skip one or two nodes:



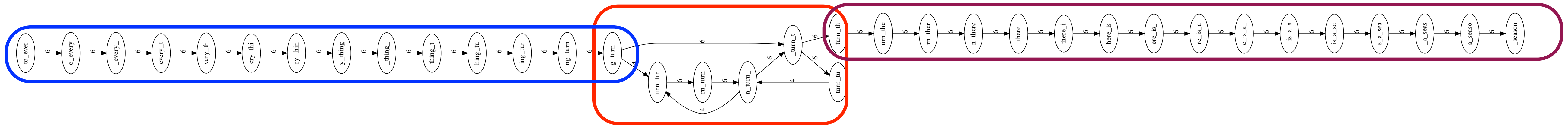
After:



Even simpler

Layout

Emit *contigs* corresponding to the non-branching stretches



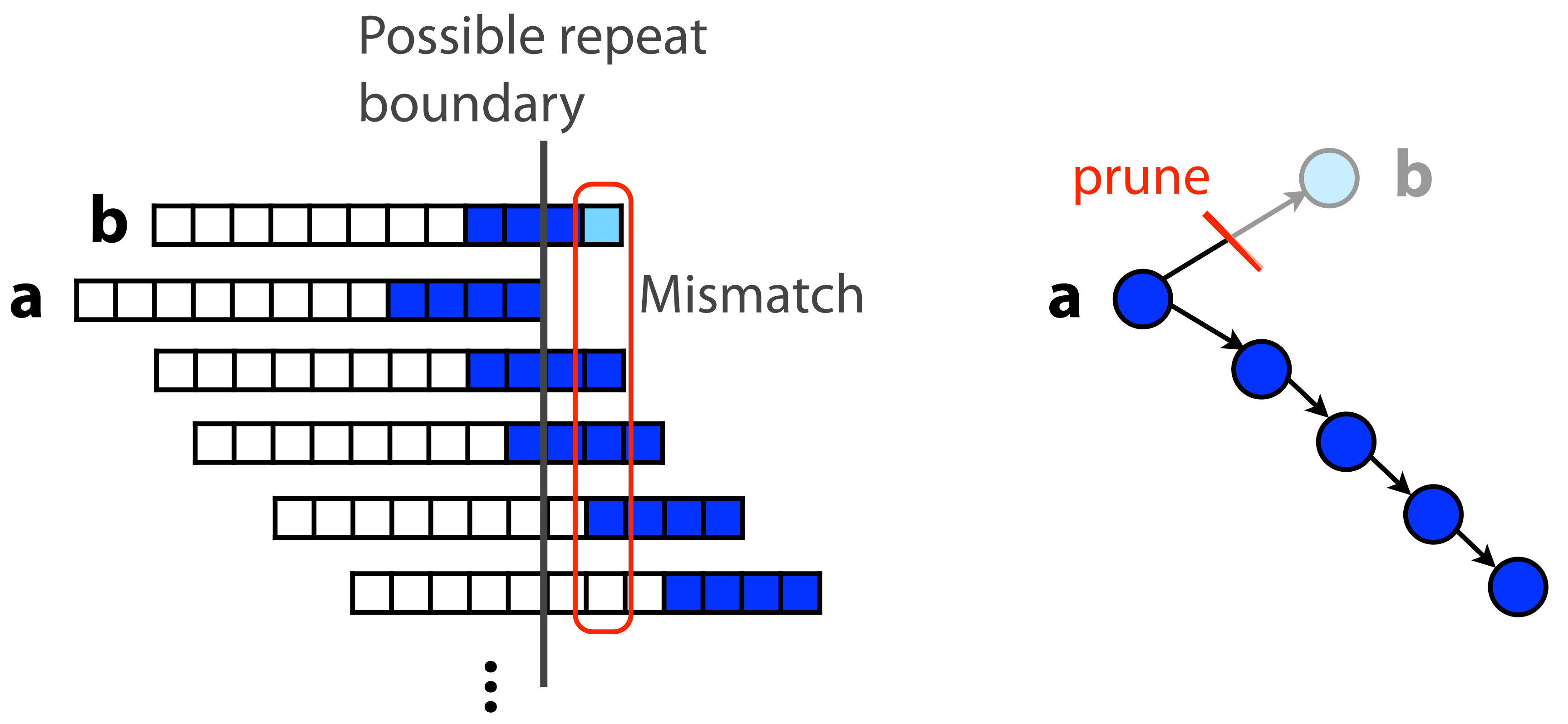
Contig 1
`to_every_thing_turn_`

Contig 2
`turn_there_is_a_season`

Unresolvable repeat

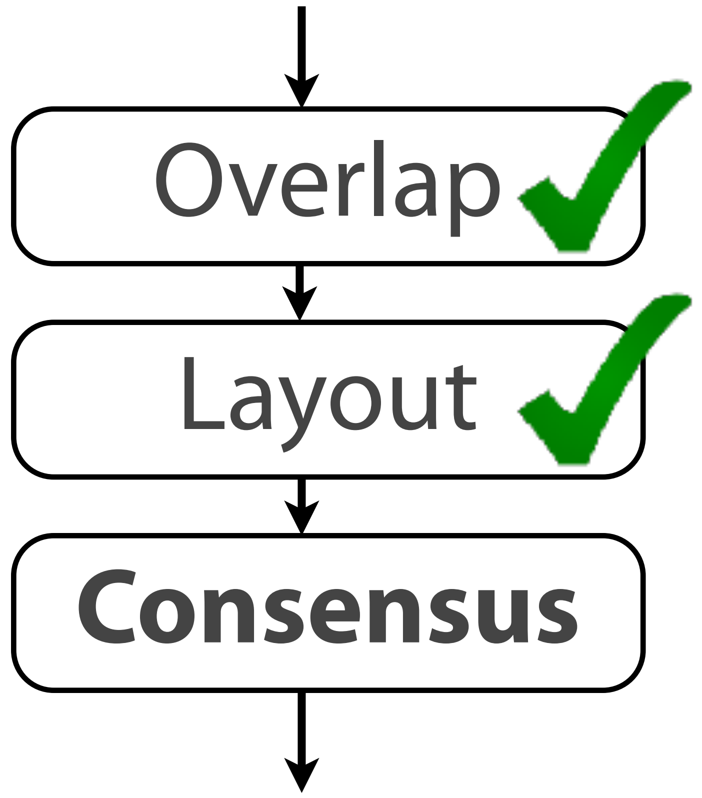
Layout

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



Build overlap graph

Bundle stretches of the overlap graph into *contigs*

Pick most likely nucleotide sequence for each contig

Consensus

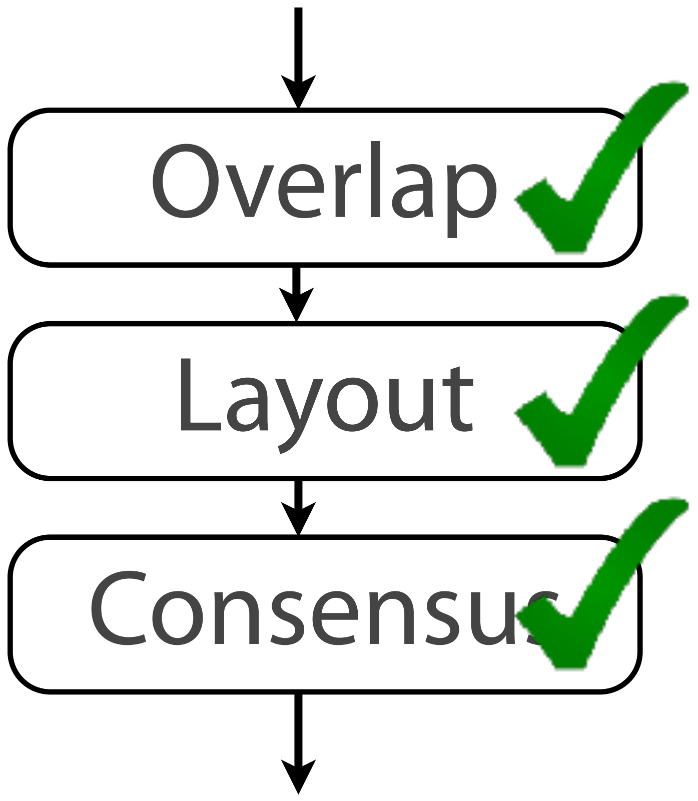
```
TAGATTACACAGATTACTGA TTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAAACTA
TAG TTACACAGATTATTGACTTCATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA

TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```

Take reads that make up a contig and line them up

Take *consensus*, i.e. majority vote

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

Overlap Layout Consensus



Build overlap graph

Coalesce paths into *contigs*

Pick nucleotide sequence for each contig

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