

# A note about the project check in

The project check in is due tomorrow, this should be quite short and only need to explain what you plan to do for the project

For example, if you're planning on doing project 1 it should

- list the algorithms you're planning to implement
- 1 sentence describing what you're planning to do for each (expose parameters, etc)
- if you're working in a team who you're working with

Everyone should hand in their own, but they can be the same content

# Office Hour Update

Weeks of 12 November & 19 November

- Normal (no change)

Week of 26 November

- No Wednesday Hours (day before thanksgiving)

Week of 3 December

- Extra office hours on Thurs. (3 Dec, 2-3p) and Fri. (4 Dec, 10a-11a)

Week of 10 December (final exams/project due)

- Extra office hours on Monday (9 December, 2-3p)
- No office hours Tuesday or Wednesday.

# De Bruijn Graph, Overlap-Layout-Consensus, & *de novo* assembly

CS 4390/5390

# De Bruijn graphs

though we call them De Bruijn graphs they were independently described by Nicolaas Govert de Bruijn and Irving John Good in 1946

they are used to encode sequence information as paths in a graph

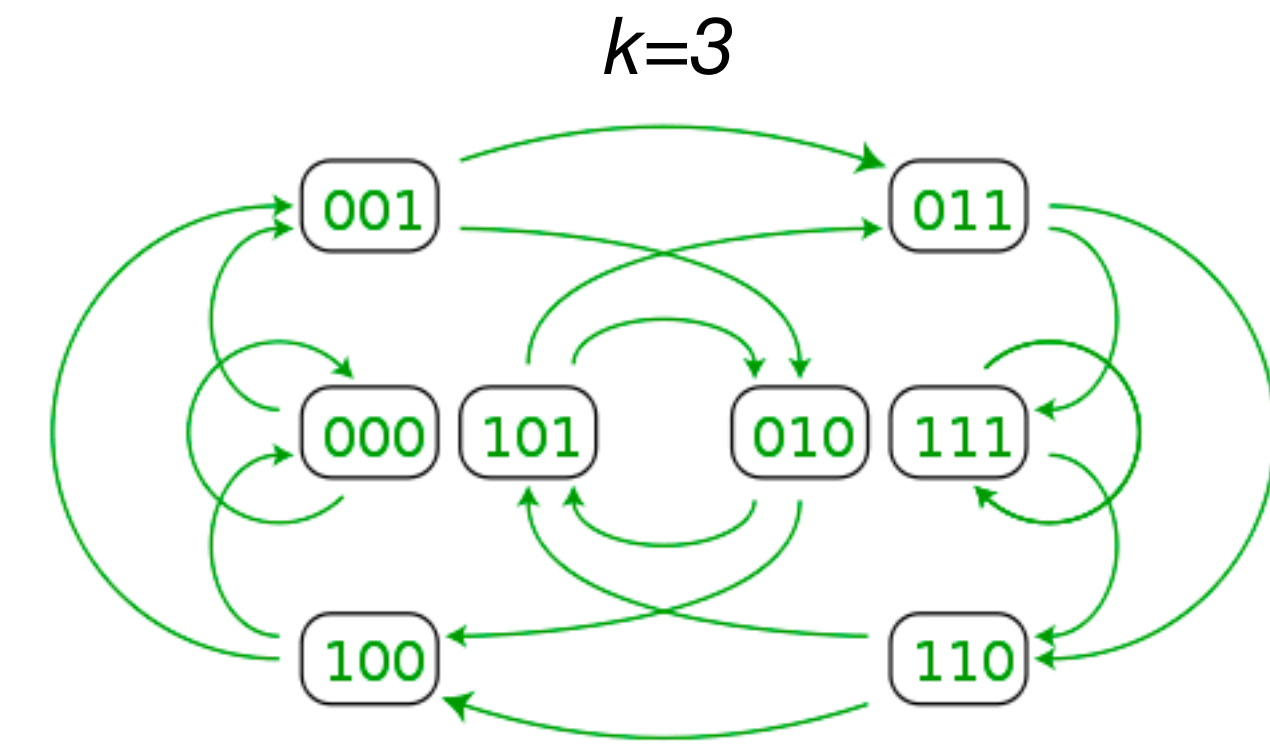
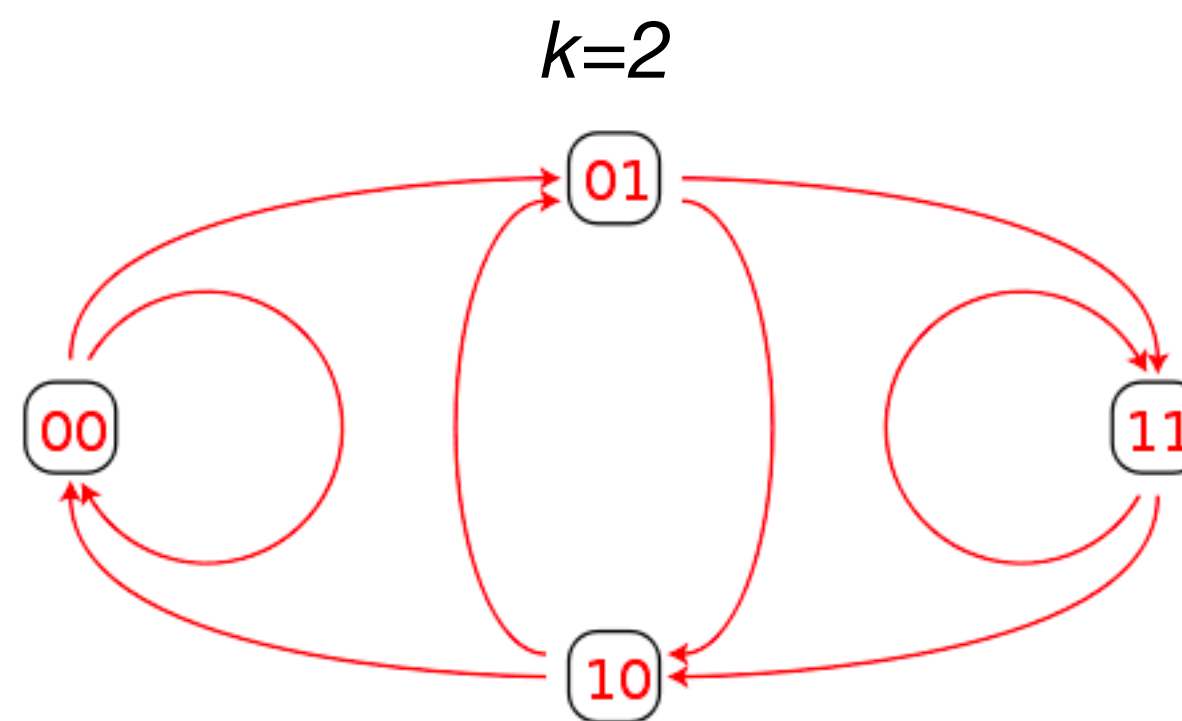
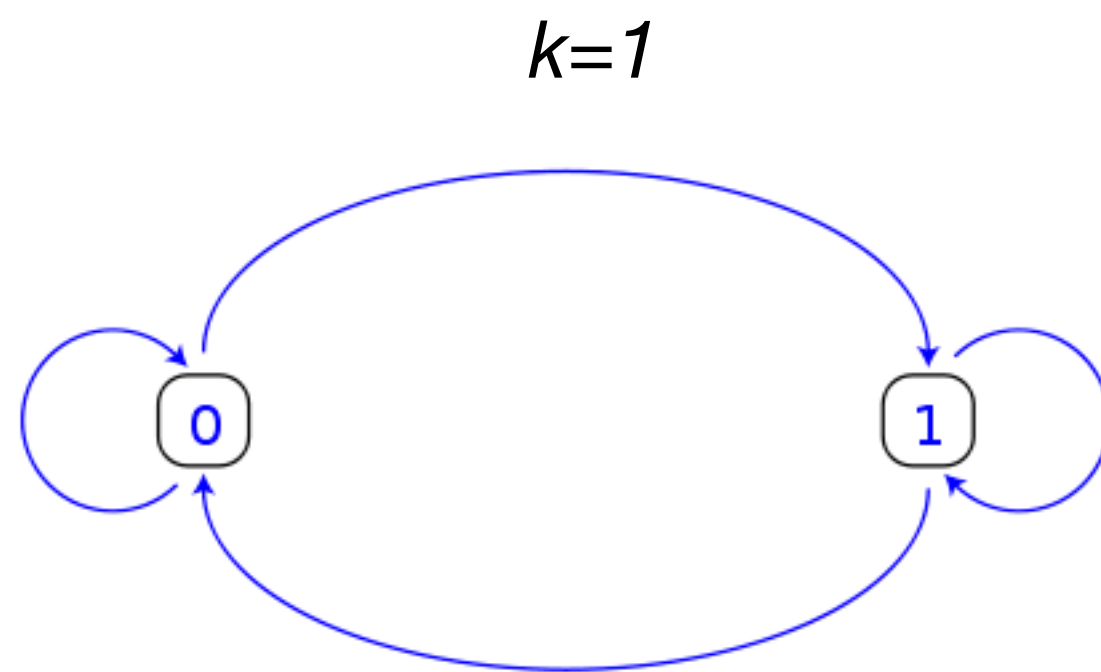
**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ ,  
there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

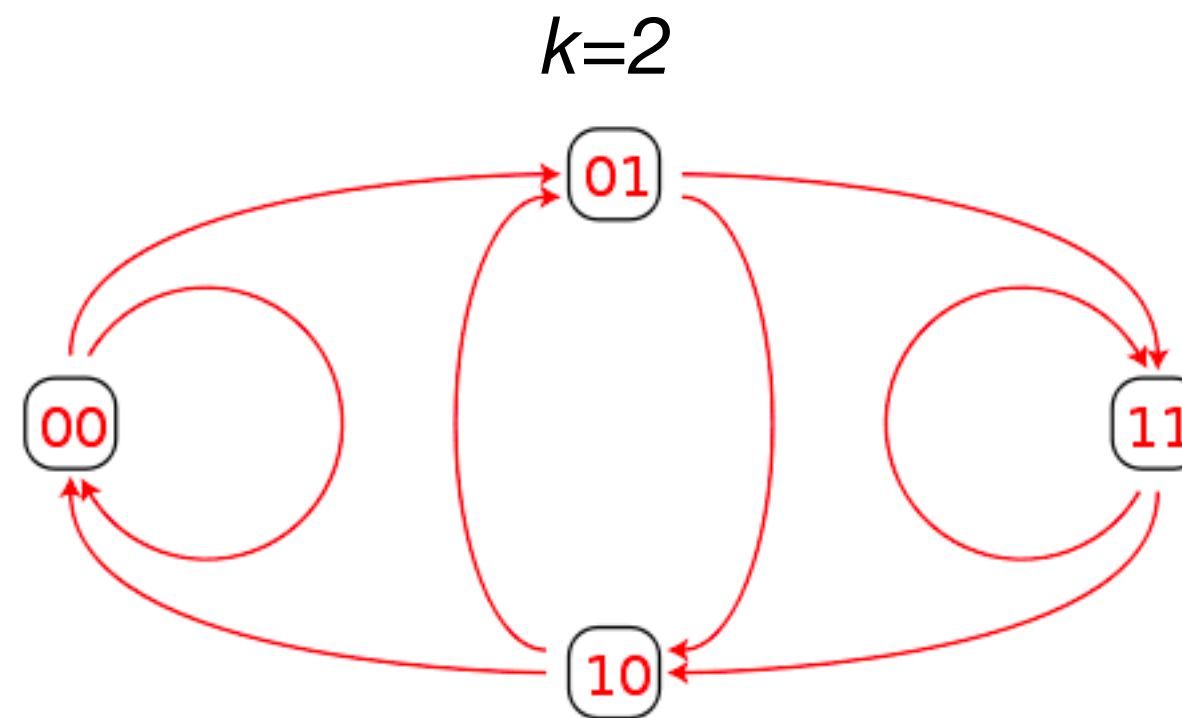


# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges

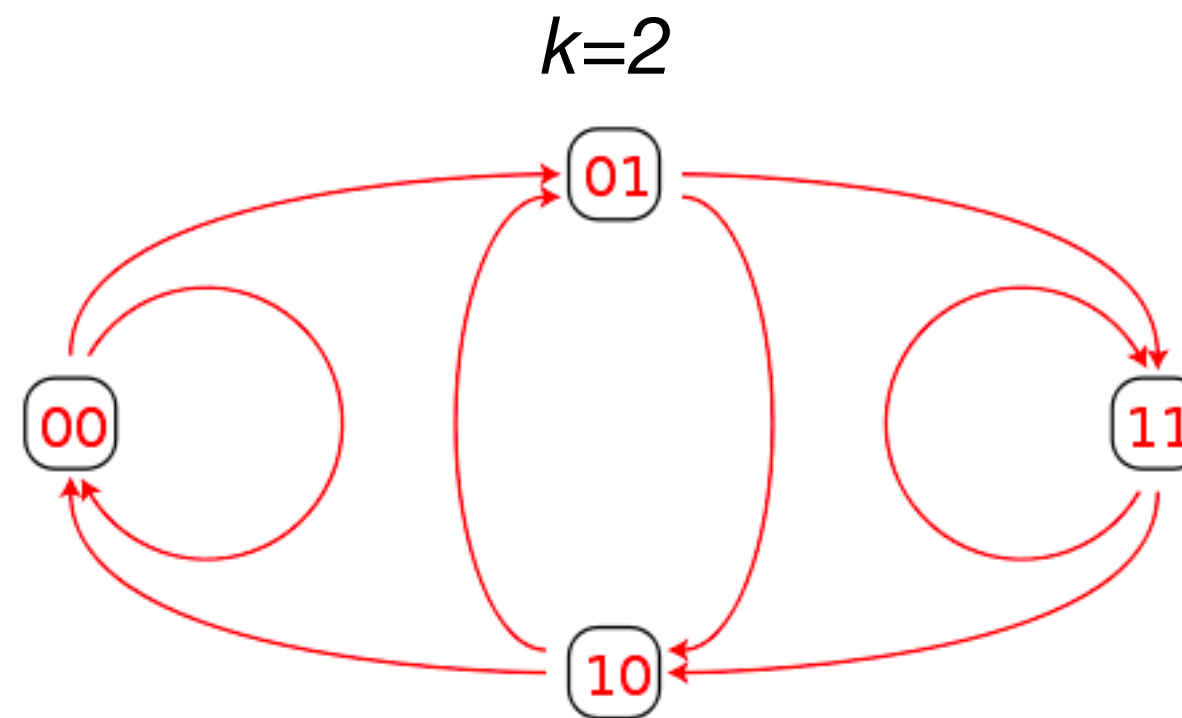


# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

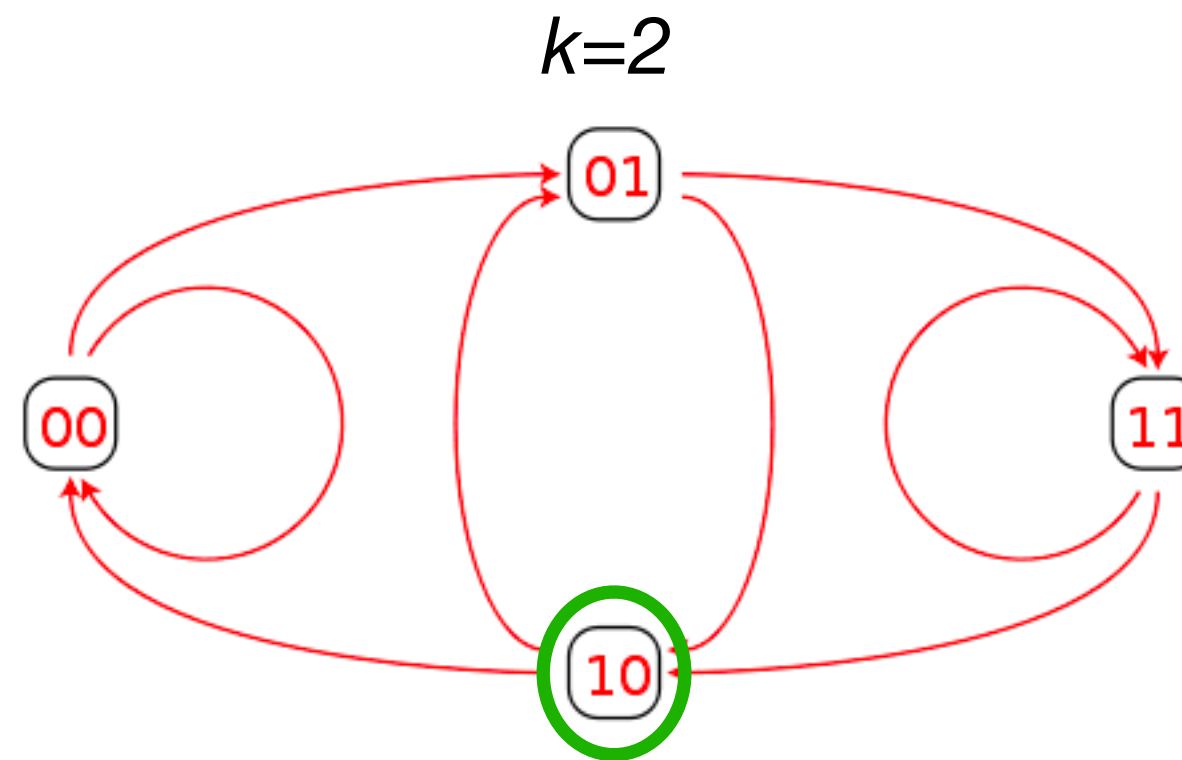
**Example:** 1011000

# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

**Example:** 1011000

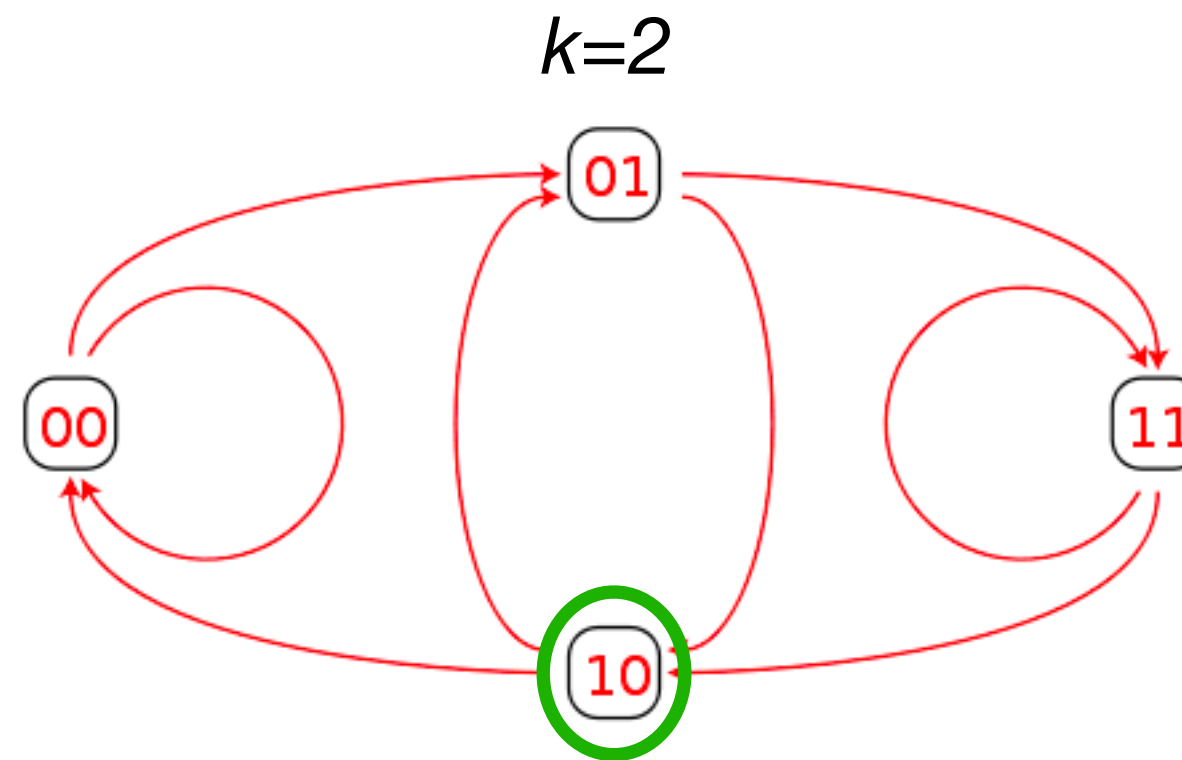


# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

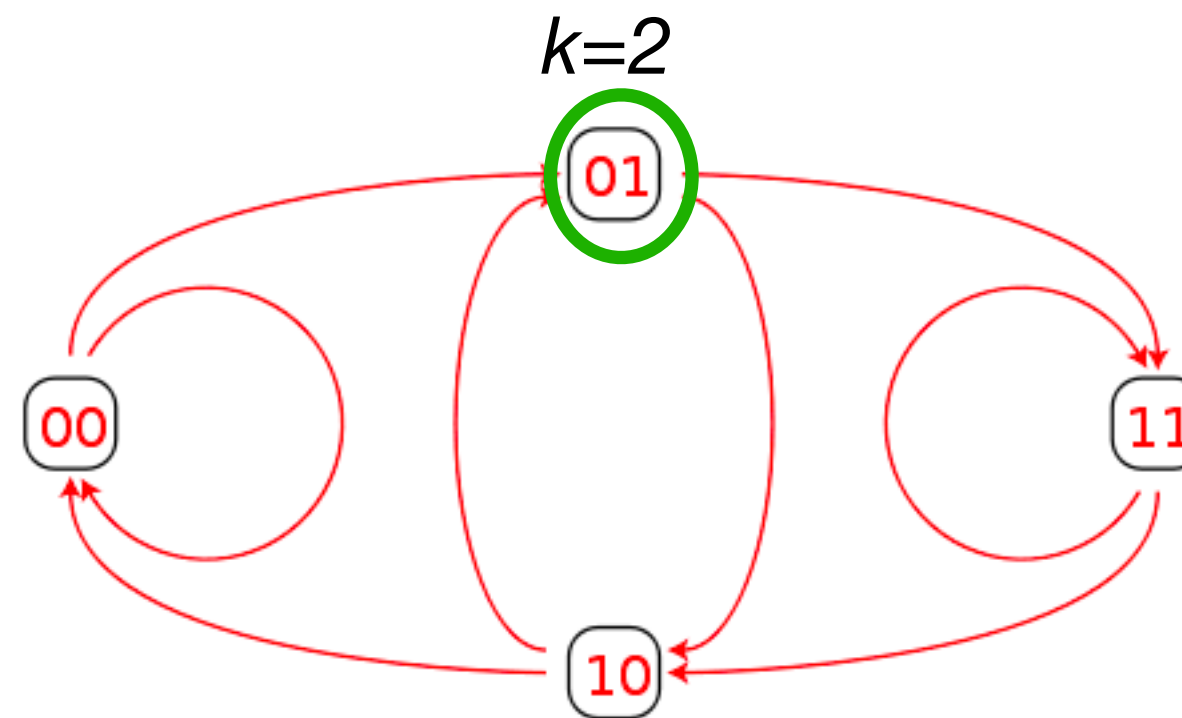
**Example:** 1011000

# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

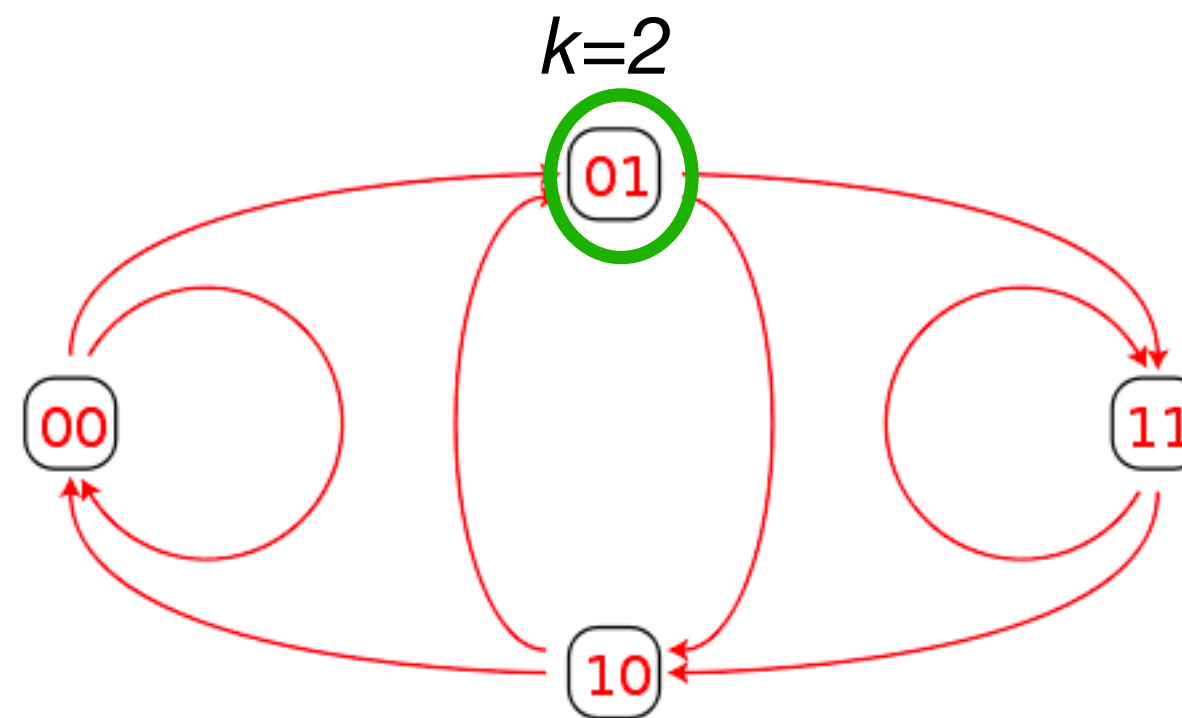
**Example:** 1011000

# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

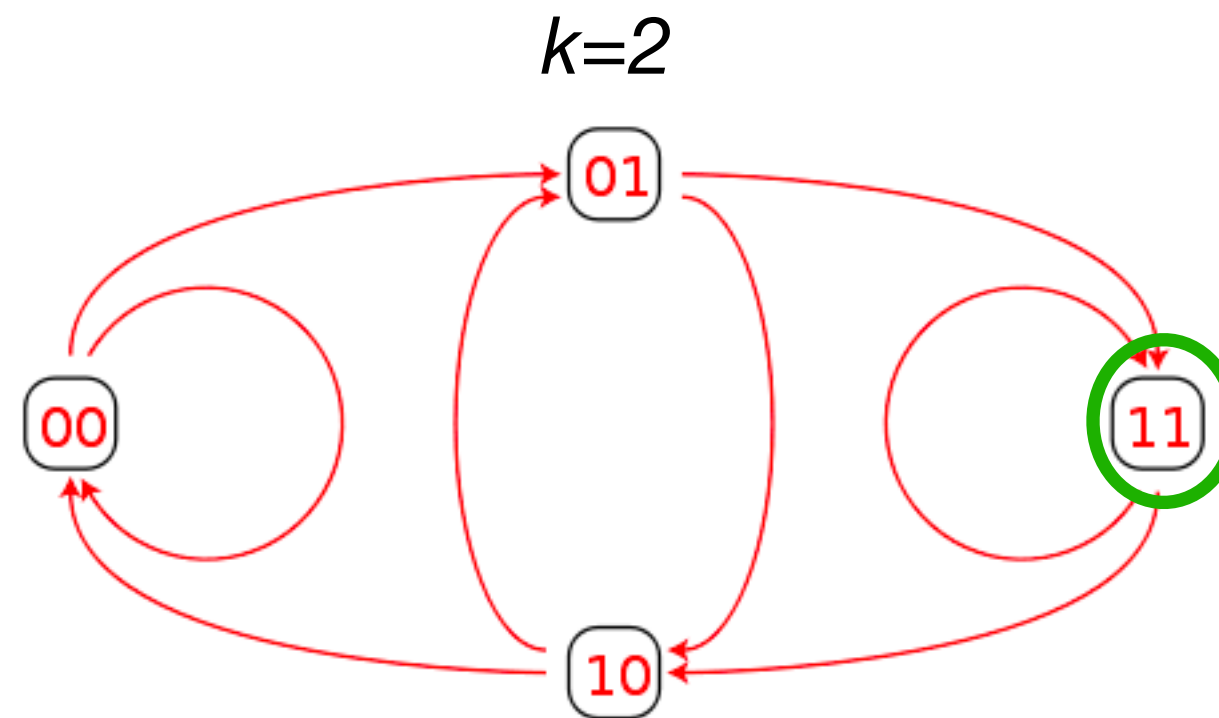
**Example:** 1011000

# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

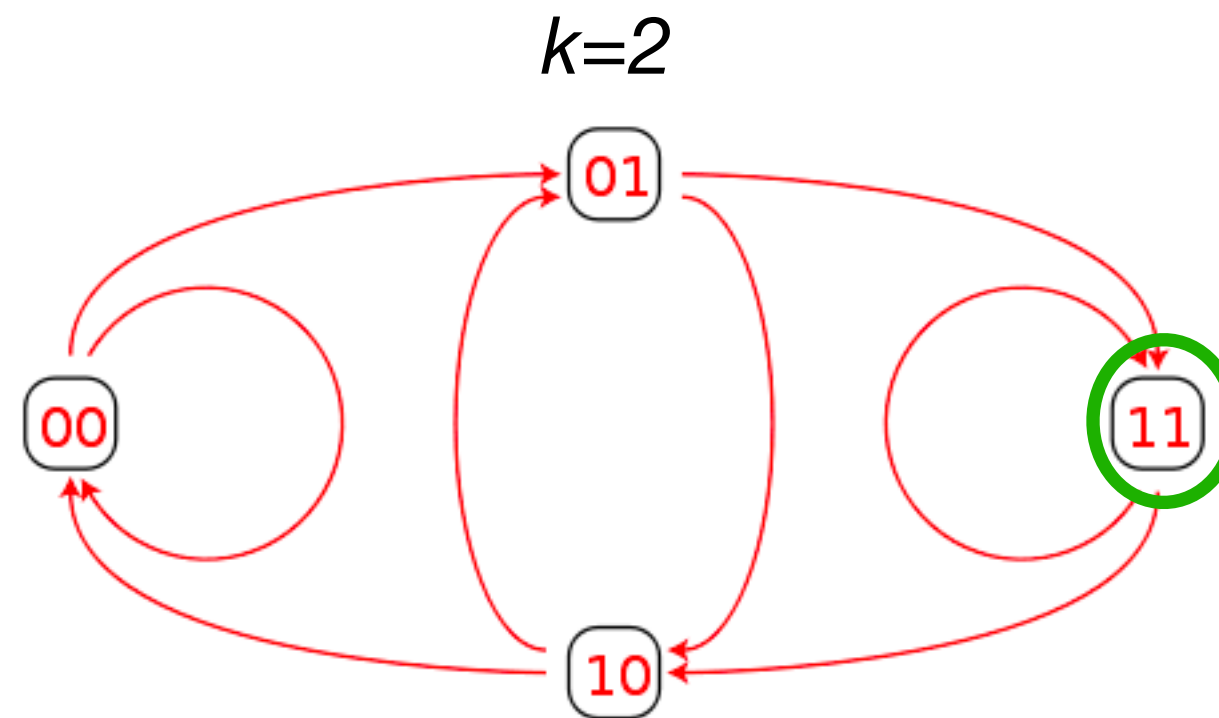
**Example:** 1011000

# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

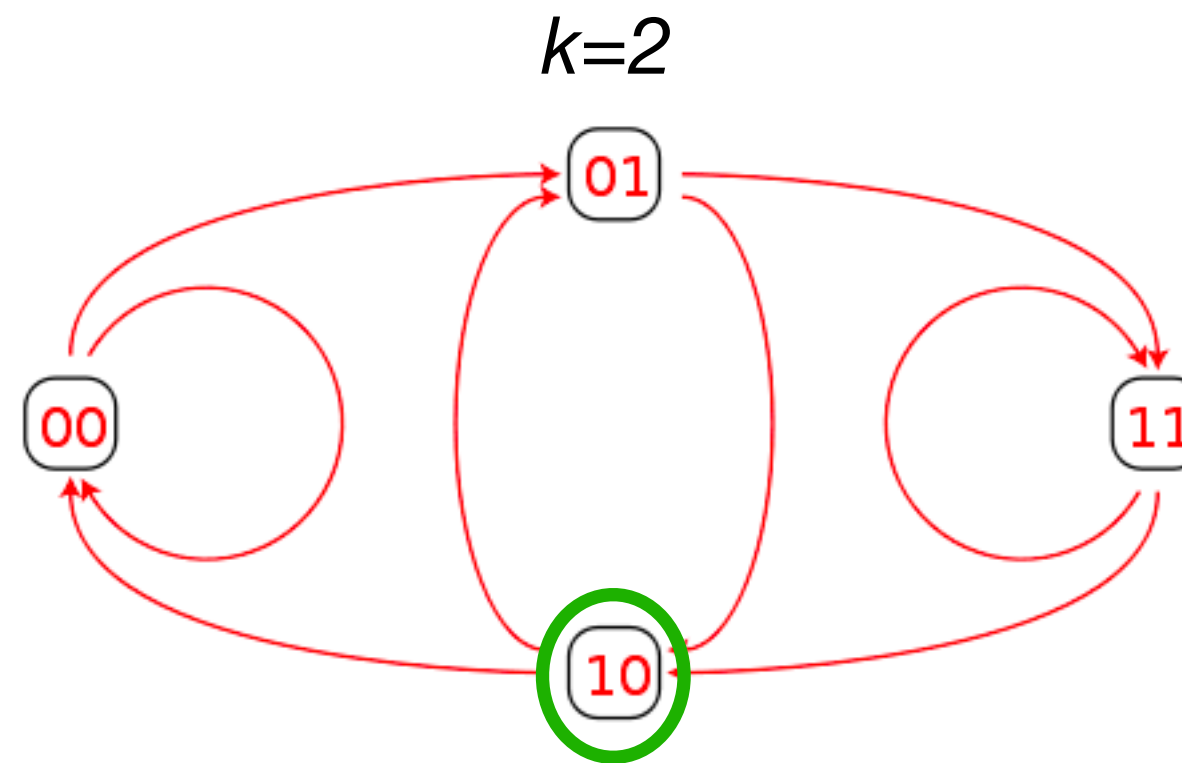
**Example:** 10**11**000

# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

**Example:** 10**11**000

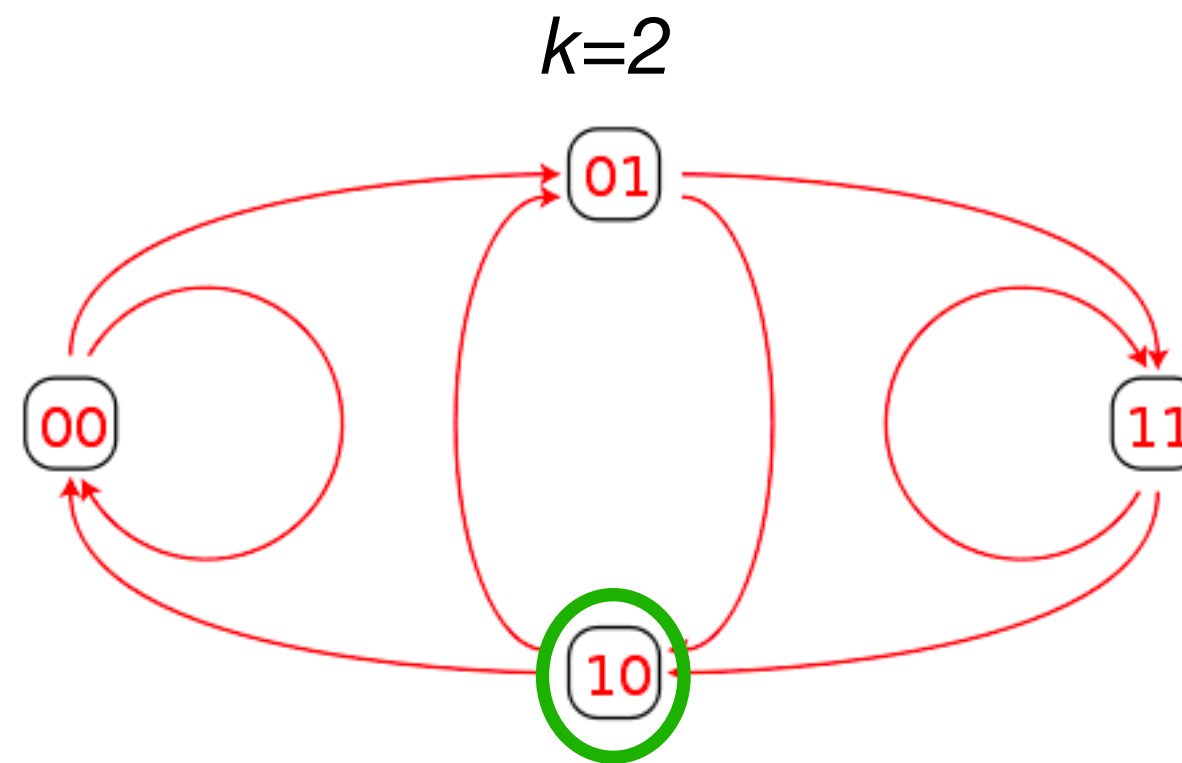


# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

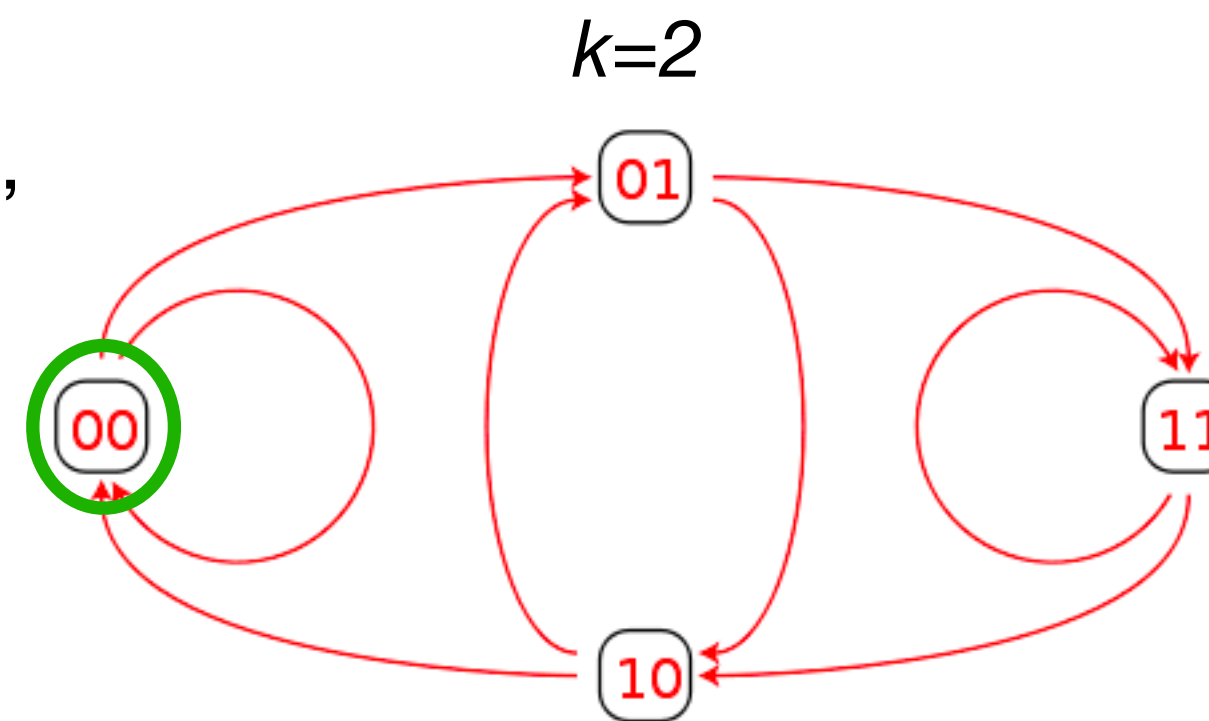
**Example:** 1011000

# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

**Example:** 1011000

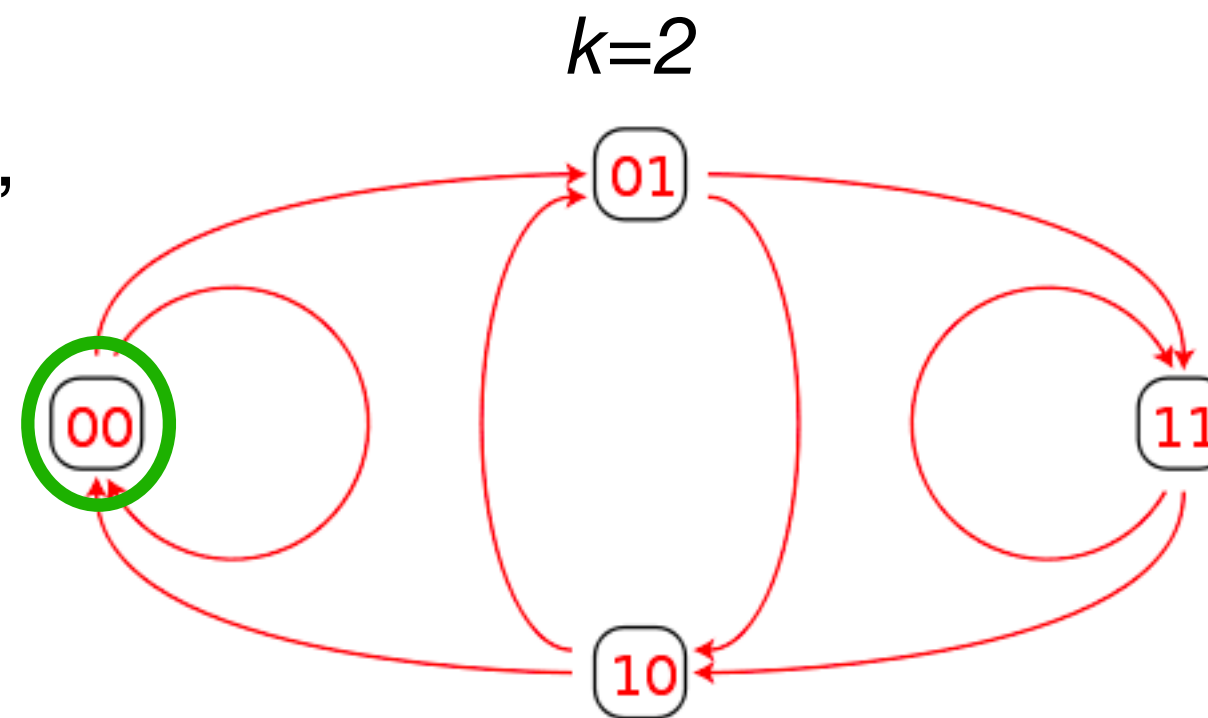


# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges



Any string over the alphabet  
can be encoded as a path on the DBG

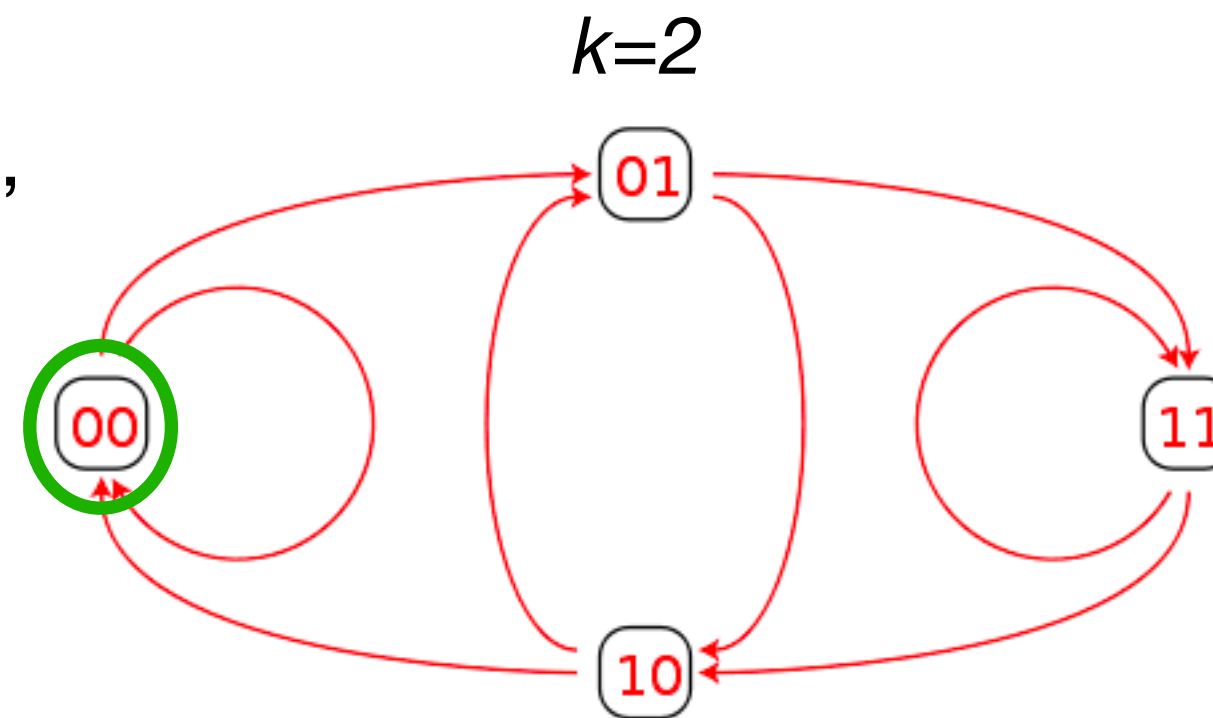
**Example:** 1011000

# De Bruijn Graphs

**Definition** a  $k$ -order de Bruijn Graph (DBG)  $D = (V, E)$  has:

- $V = \Sigma^k$  -- there is a vertex for each possible  $k$ -mer
- $E = \{ax \rightarrow xb \mid a, b \in \Sigma, x \in \Sigma^{(k-1)}\}$  -- for each  $(k+1)$ -mer  $axb$ , there is an edge from the  $k$ -mer  $ax$  to the  $k$ -mer  $xb$

Each node has  $\sigma$  outgoing edges,  
and  $\sigma$  incoming edges

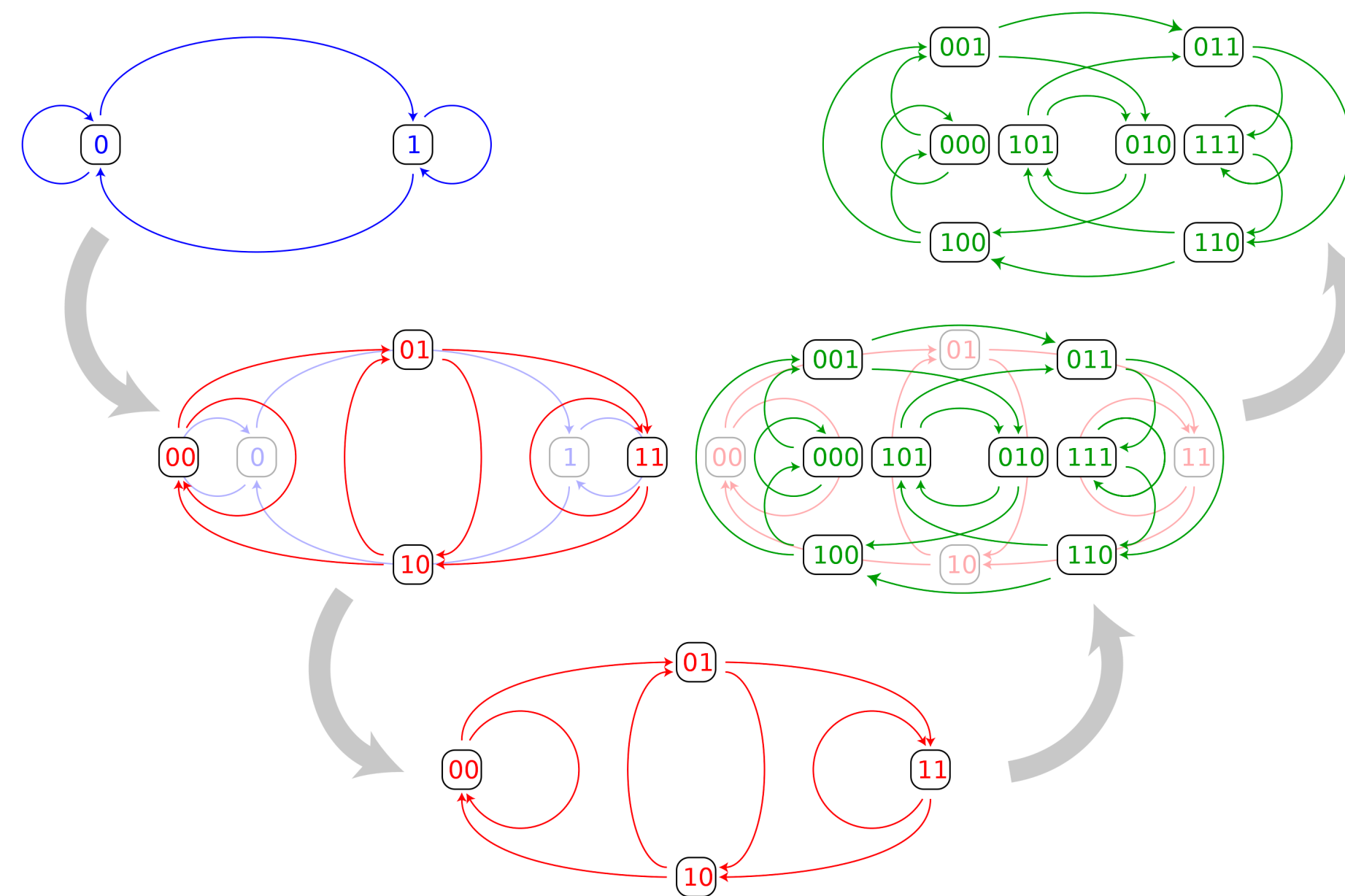


Any string over the alphabet  
can be encoded as a path on the DBG

**Example:** 1011000

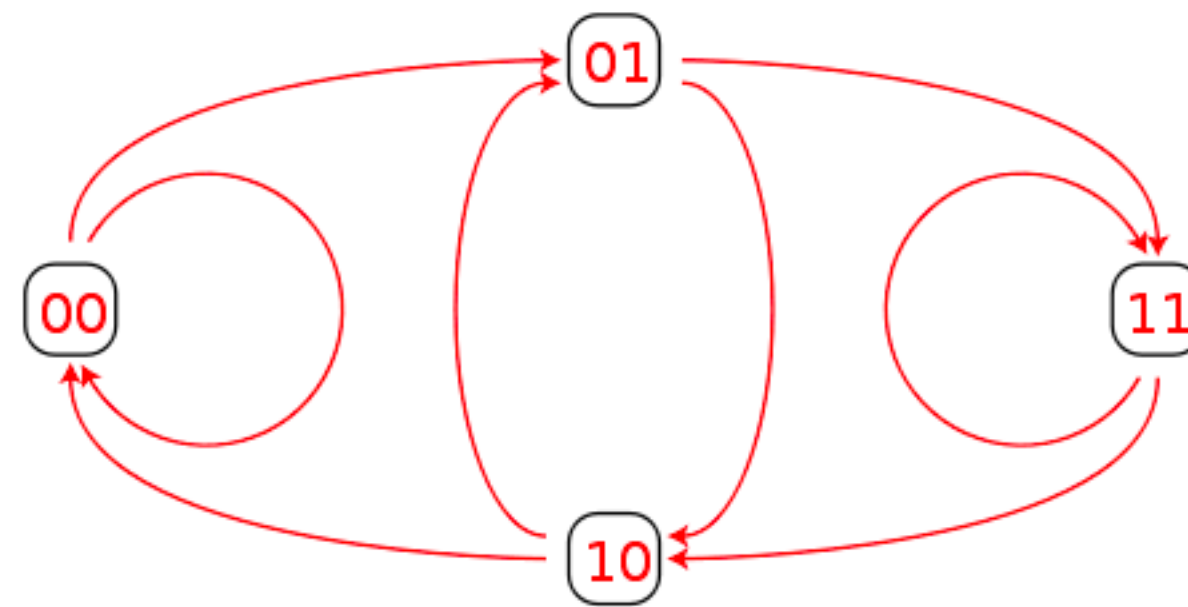
# Other properties for DBGs we won't use for assembly

the de Bruijn of order  $k$  is a **line graph** of the debrujin graph of order  $k-1$



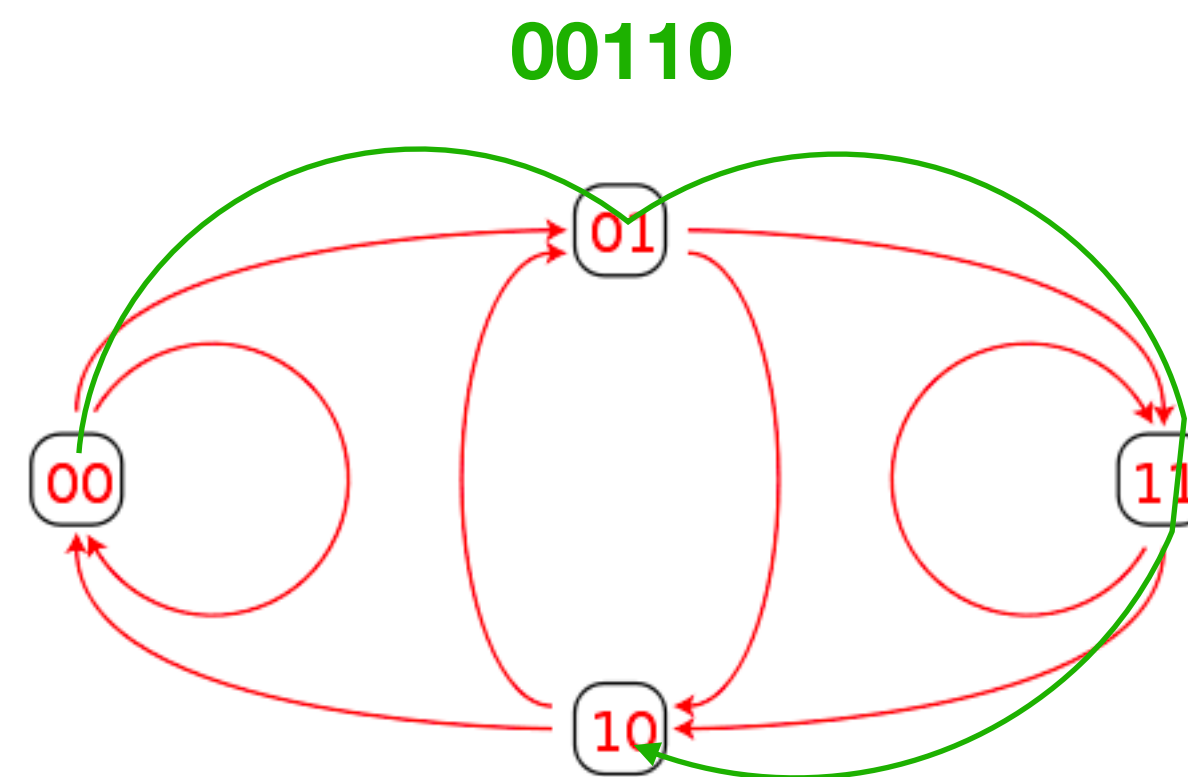
# Other properties for DBGs we won't use for assembly

A de Bruijn *sequence* is an *Hamiltoninan* path of the graph, meaning it contains all  $k$ -mers exactly once



# Other properties for DBGs we won't use for assembly

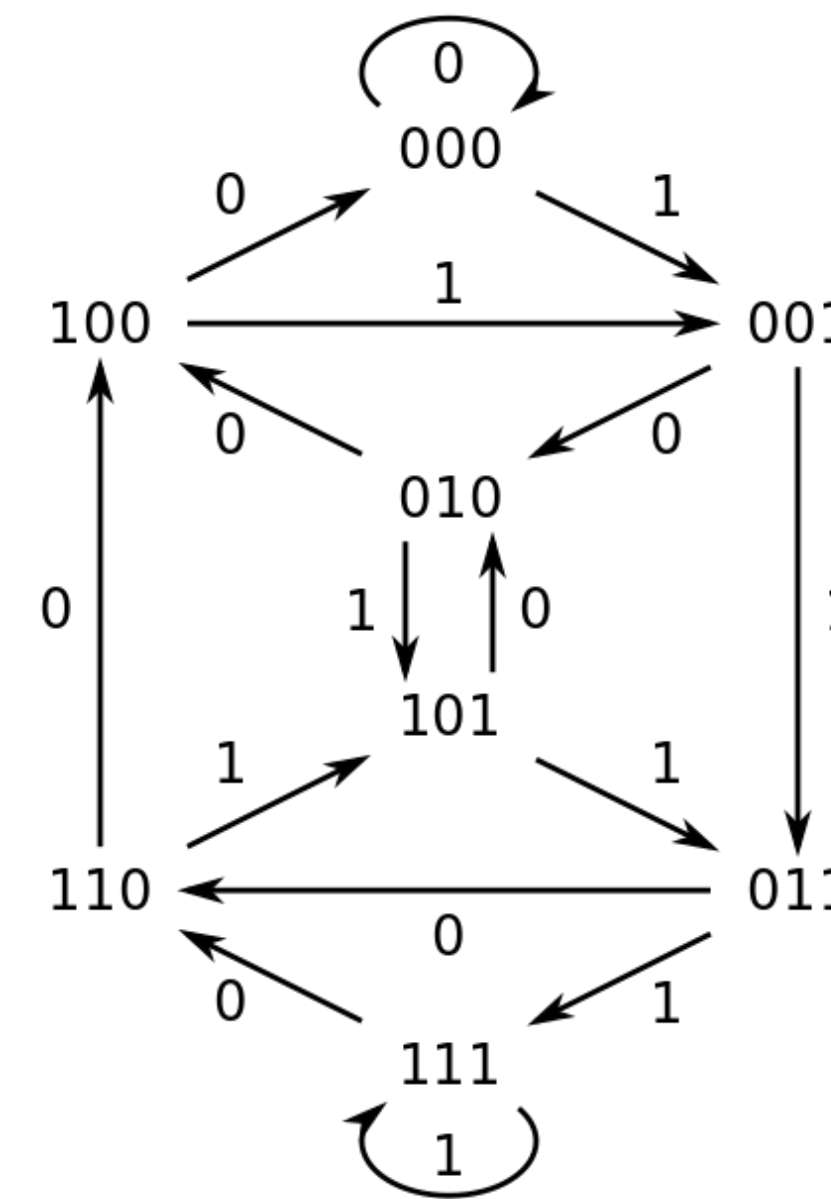
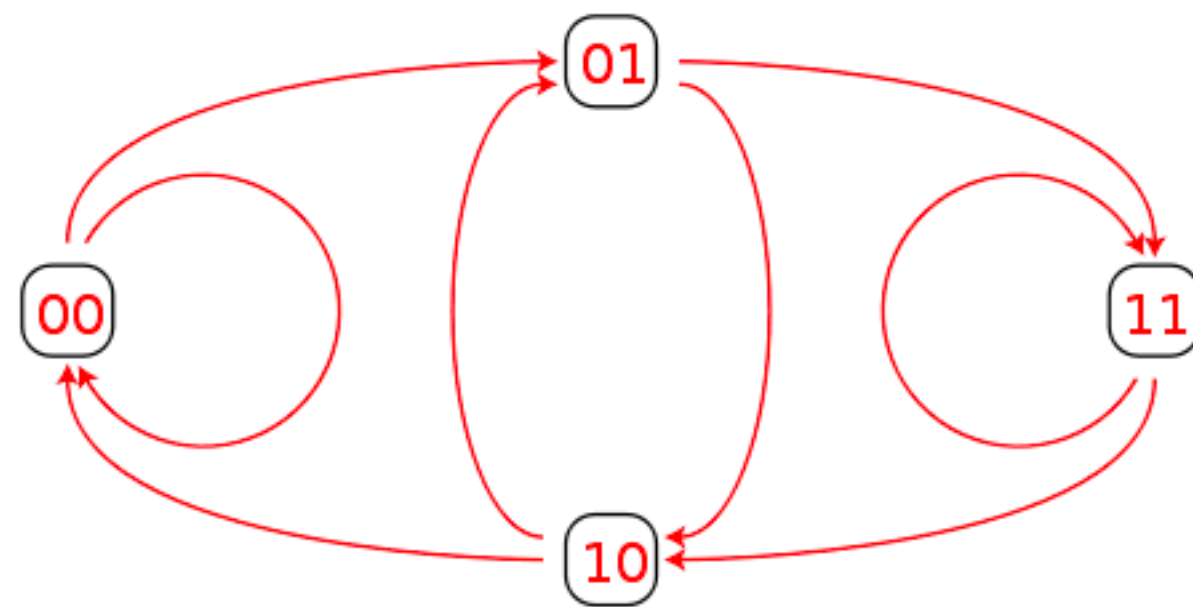
A de Bruijn *sequence* is an *Hamiltoninan* path of the graph, meaning it contains all  $k$ -mers exactly once



# Other properties for DBGs we won't use for assembly

A de Bruijn *sequence* is an *Hamiltonian* path of the graph, meaning it contains all  $k$ -mers exactly once

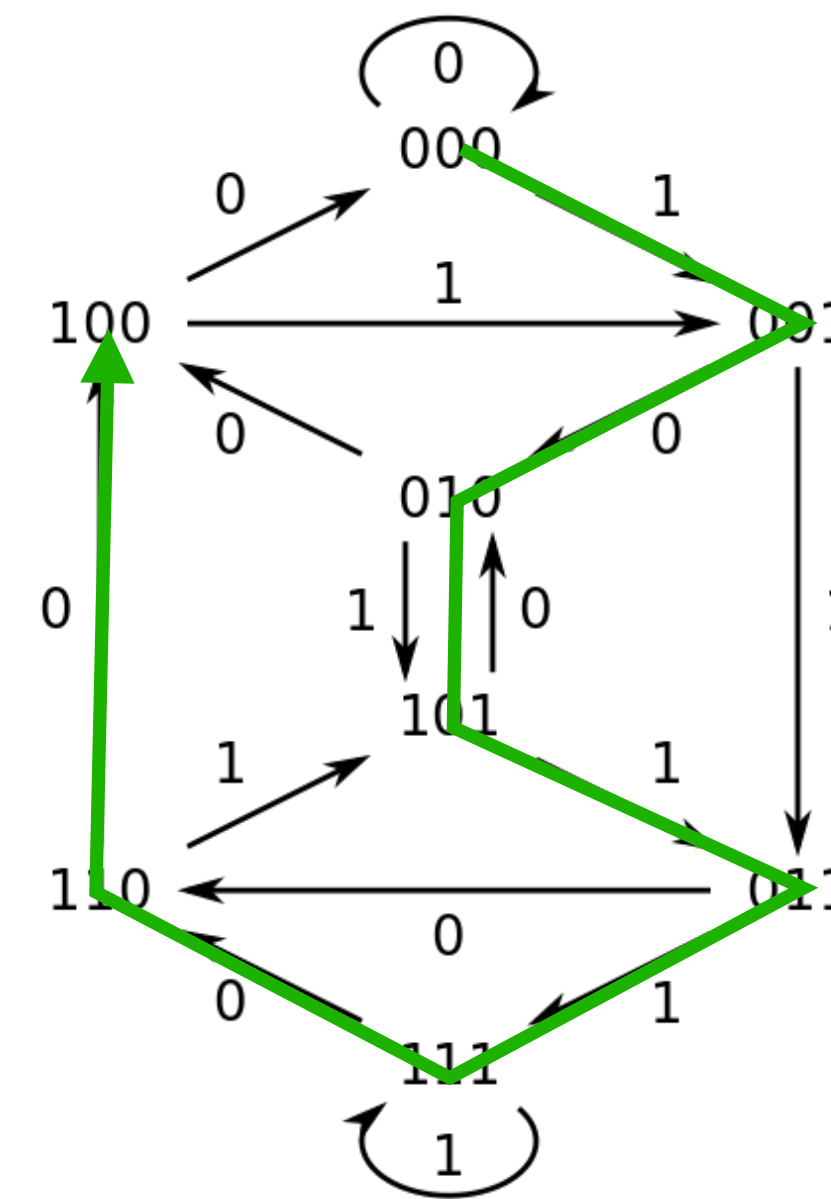
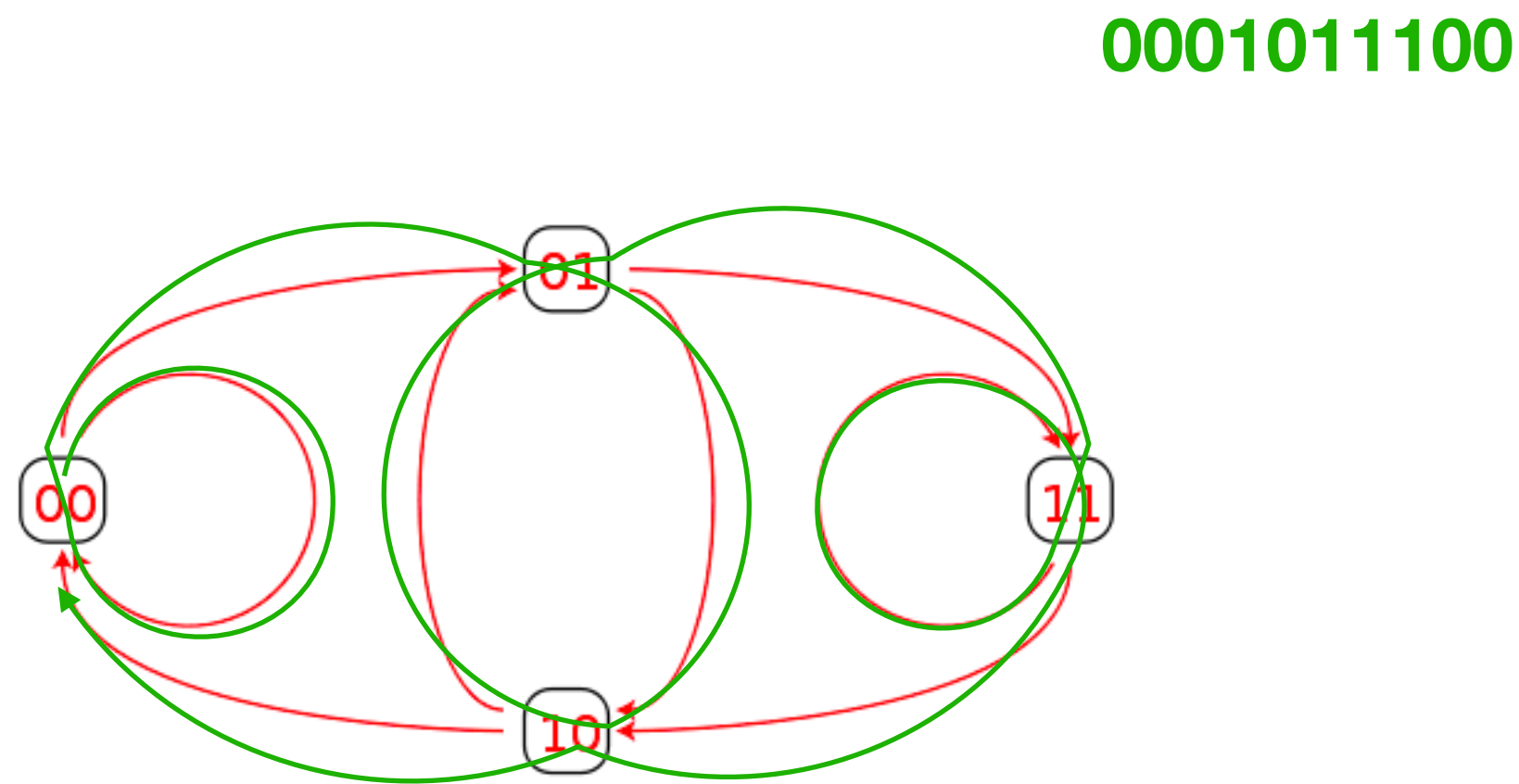
- or the *Eulerian* path of the graph of  $k-1$



# Other properties for DBGs we won't use for assembly

A de Bruijn *sequence* is an *Hamiltoninan* path of the graph, meaning it contains all  $k$ -mers exactly once

- or the *Eulerian* path of the graph of  $k-1$

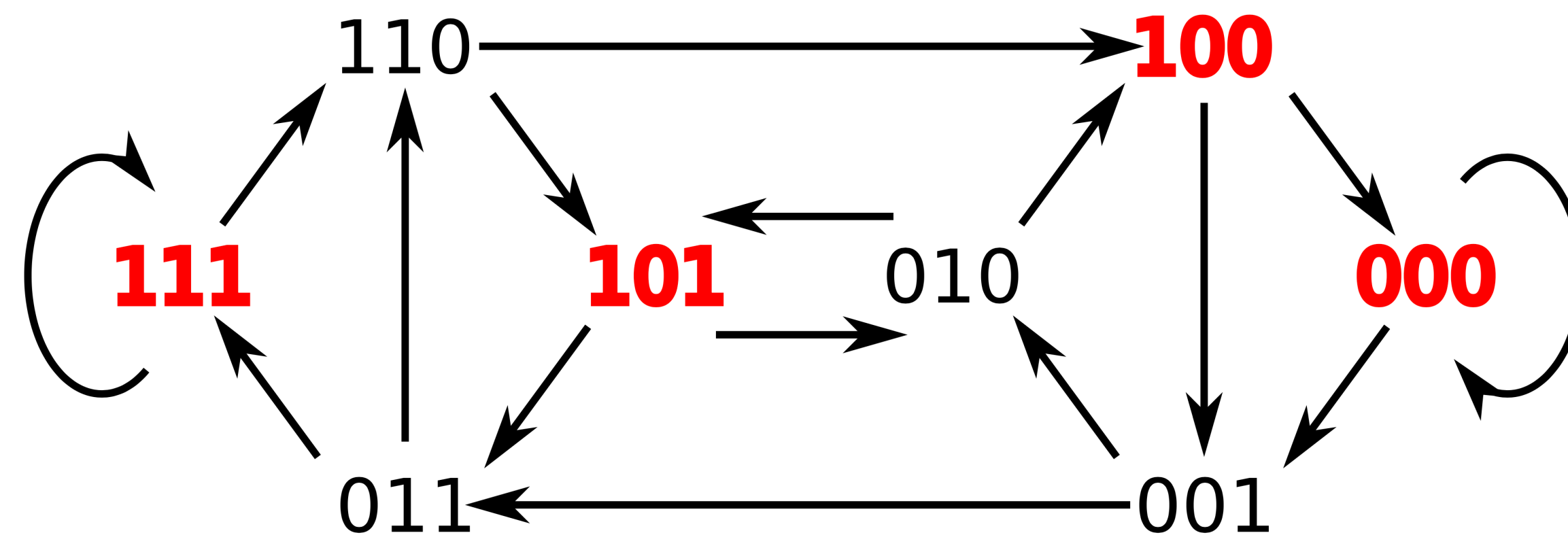




# Other properties for DBGs we won't use for assembly

a decycling set of edges a de Bruijn graph is a set of nodes that when removed leave a DAG

- this set of  $k$ -mers is guaranteed to exist in all long enough sequences



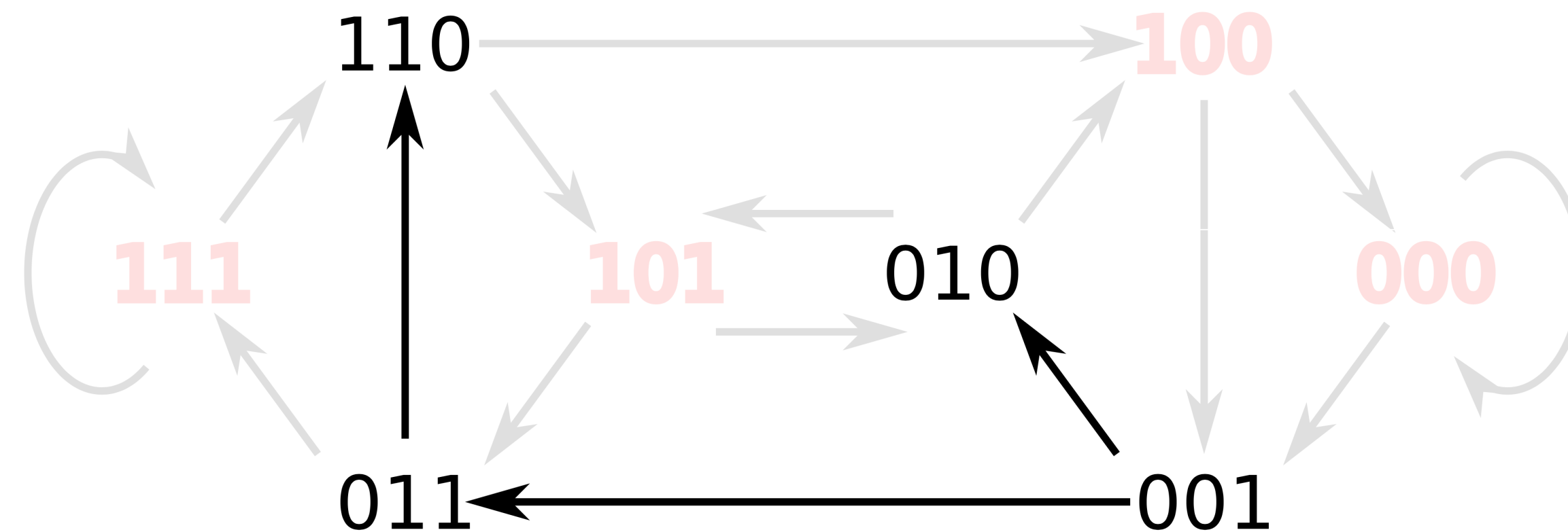
Can you find a length 6 binary sequence,  
which does not intersect one of the red  $k$ -mers?



# Other properties for DBGs we won't use for assembly

a decycling set of edges a de Bruijn graph is a set of nodes that when removed leave a DAG

- this set of  $k$ -mers is guaranteed to exist in all long enough sequences

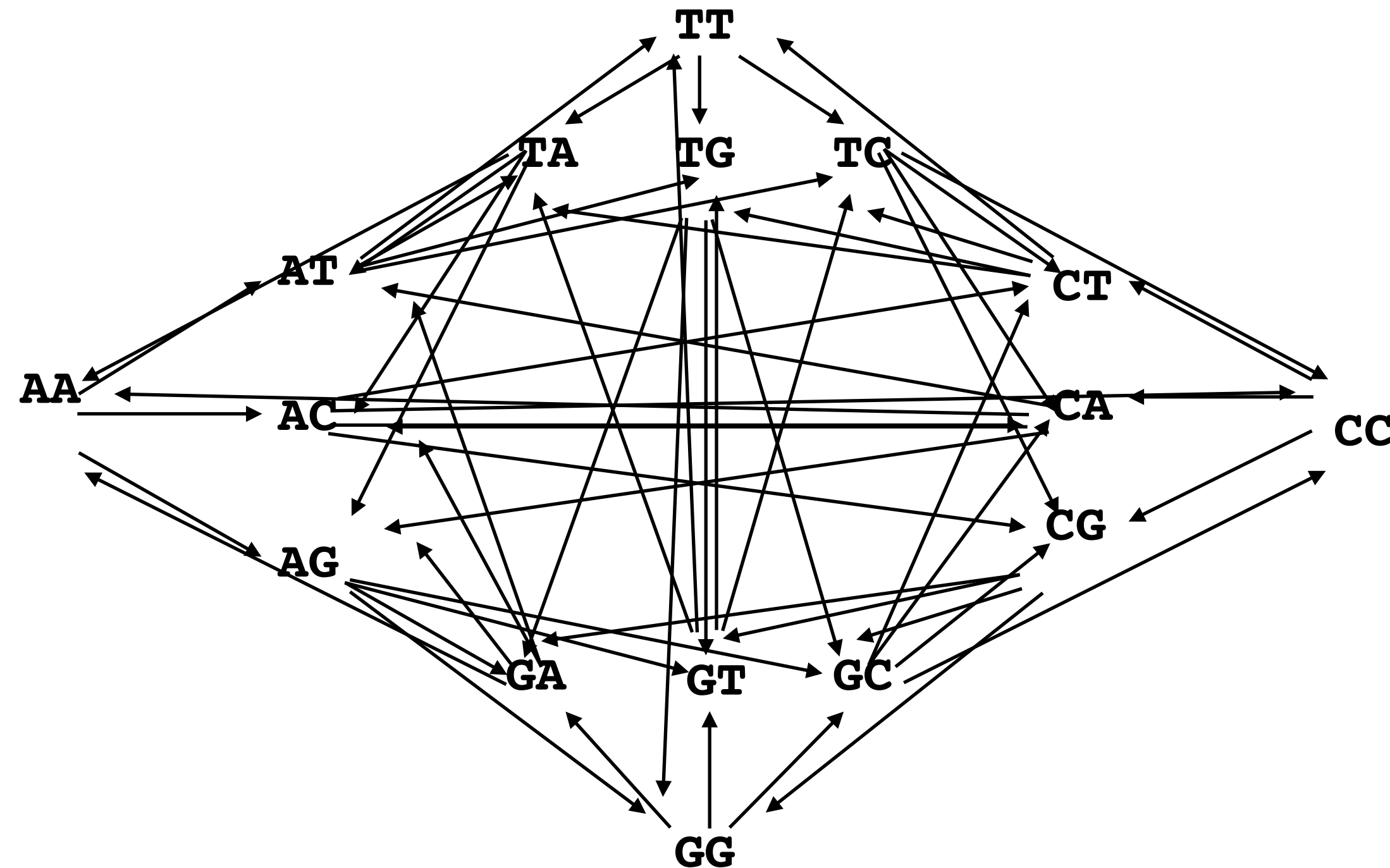


Can you find a length 6 binary sequence,  
which does not intersect one of the red  $k$ -mers?

# DBG for DNA

What we have seen in the previous slides was the DBG for  $\Sigma = \{1,0\}$

For DNA ( $\Sigma = \{A, C, T, G\}$ ) the graph is a little more complicated



# *Sequence* de Bruijn Graphs

What is most commonly used in practice for genome assembly is a subset of the DBG based on a given sequence

This is sometimes in literature referred to as simply a de Bruijn Graph

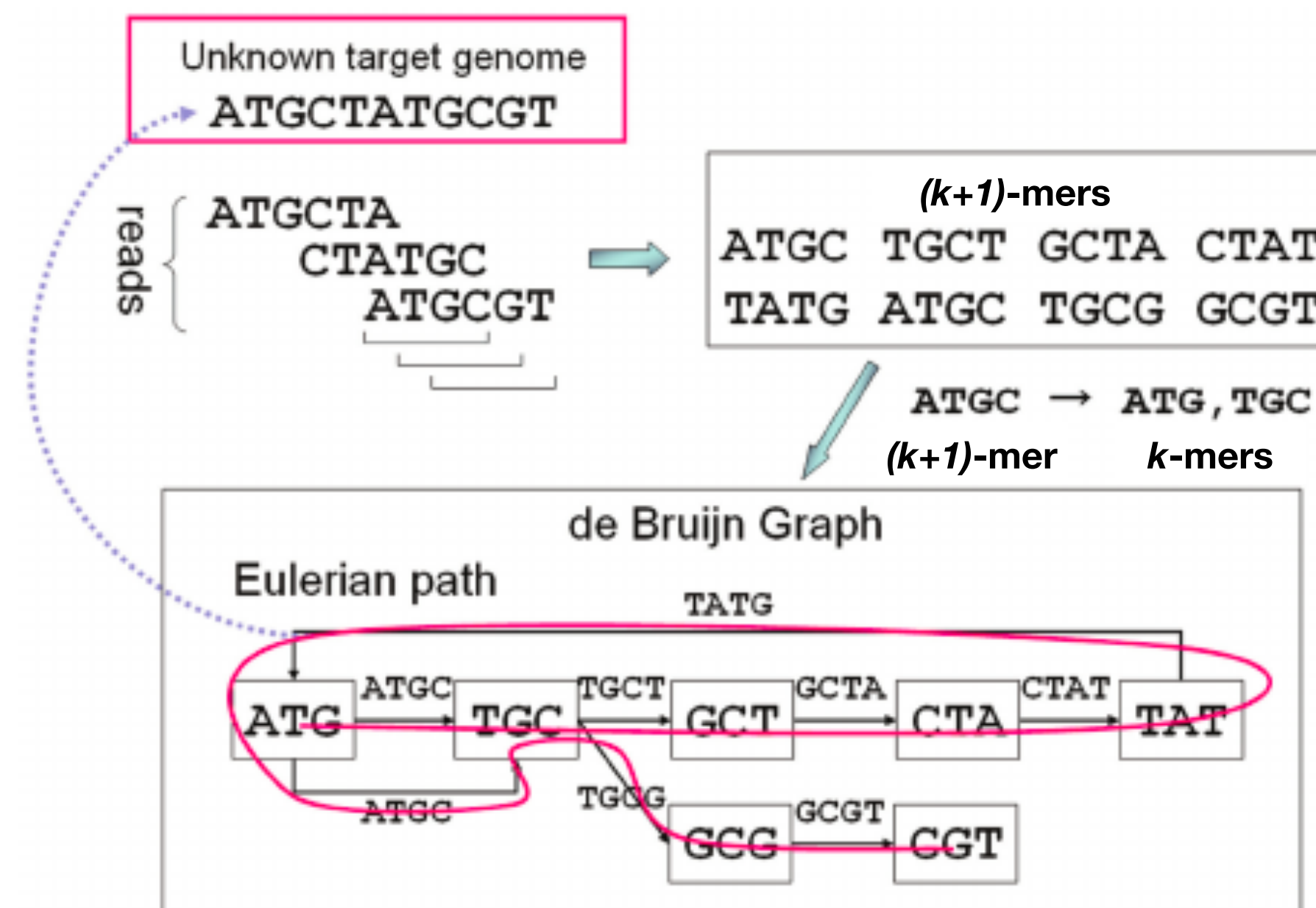
AAAGGCGTTGAGGTT  
AAAG  
AAGG  
AGGC  
GGCG  
GCGT  
CGTT  
GTTG  
TTGA  
TGAG  
GAGG  
AGGT  
GGTT



# *Sequence* de Brujin Graphs

What is most commonly used in practice for genome assembly is a subset of the DBG based on a given sequence

This is sometimes in literature referred to as simply a de Bruijn Graph



# *Sequence* de Bruijn Graphs

Casting assembly as Eulerian walk is appealing, but not practical

Uneven coverage, sequencing errors, etc make graph non-Eulerian

Even if graph were Eulerian, repeats yield many possible walks

Kingsford, Carl, Michael C. Schatz, and Mihai Pop. "Assembly complexity of prokaryotic genomes using short reads." *BMC bioinformatics* 11.1 (2010): 21.

*De Bruijn Superwalk Problem* (DBSP) seeks a walk over the De Bruijn graph, where walk contains each read as a *subwalk*

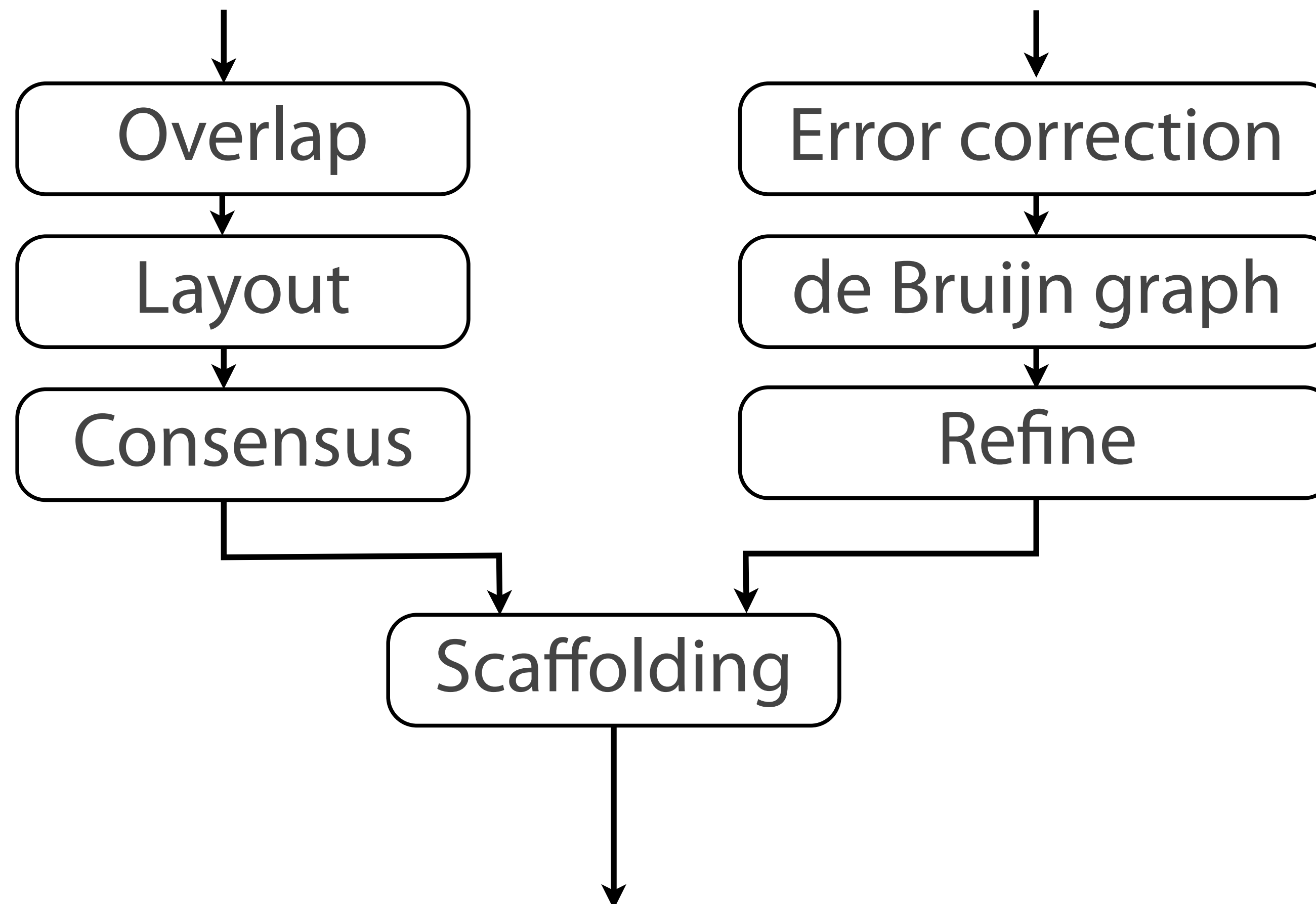
Proven NP-hard!

Medvedev, Paul, et al. "Computability of models for sequence assembly." *Algorithms in Bioinformatics*. Springer Berlin Heidelberg, 2007. 289-301.

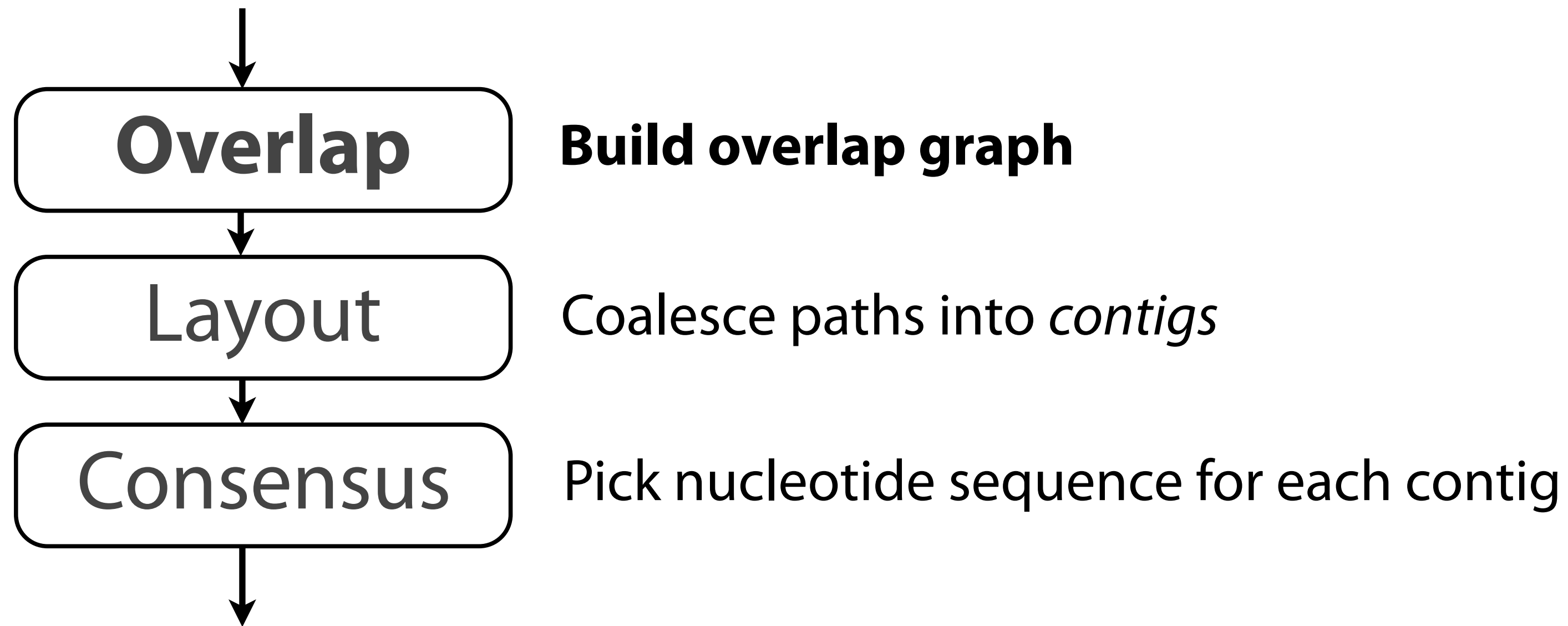
# Assembly alternatives

Alternative 1: Overlap-Layout-Consensus (OLC) assembly

Alternative 2: De Bruijn graph (DBG) assembly



# 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  $X$

$X$ : CTCTAGGCC

$Y$ : TAGGCCCTC

Found it

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

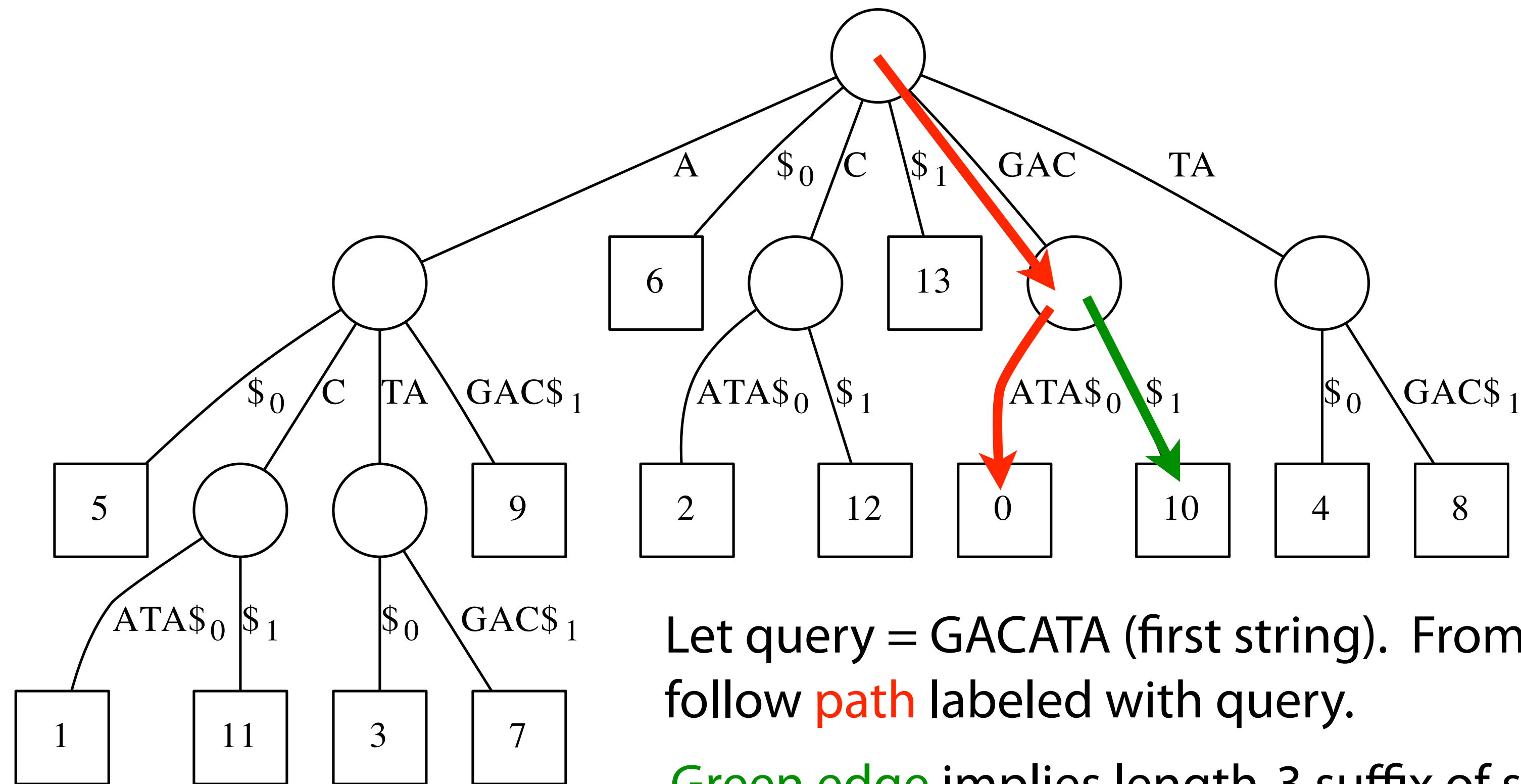
With suffix tree?

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>



ATAGAC  
|||  
GACATA

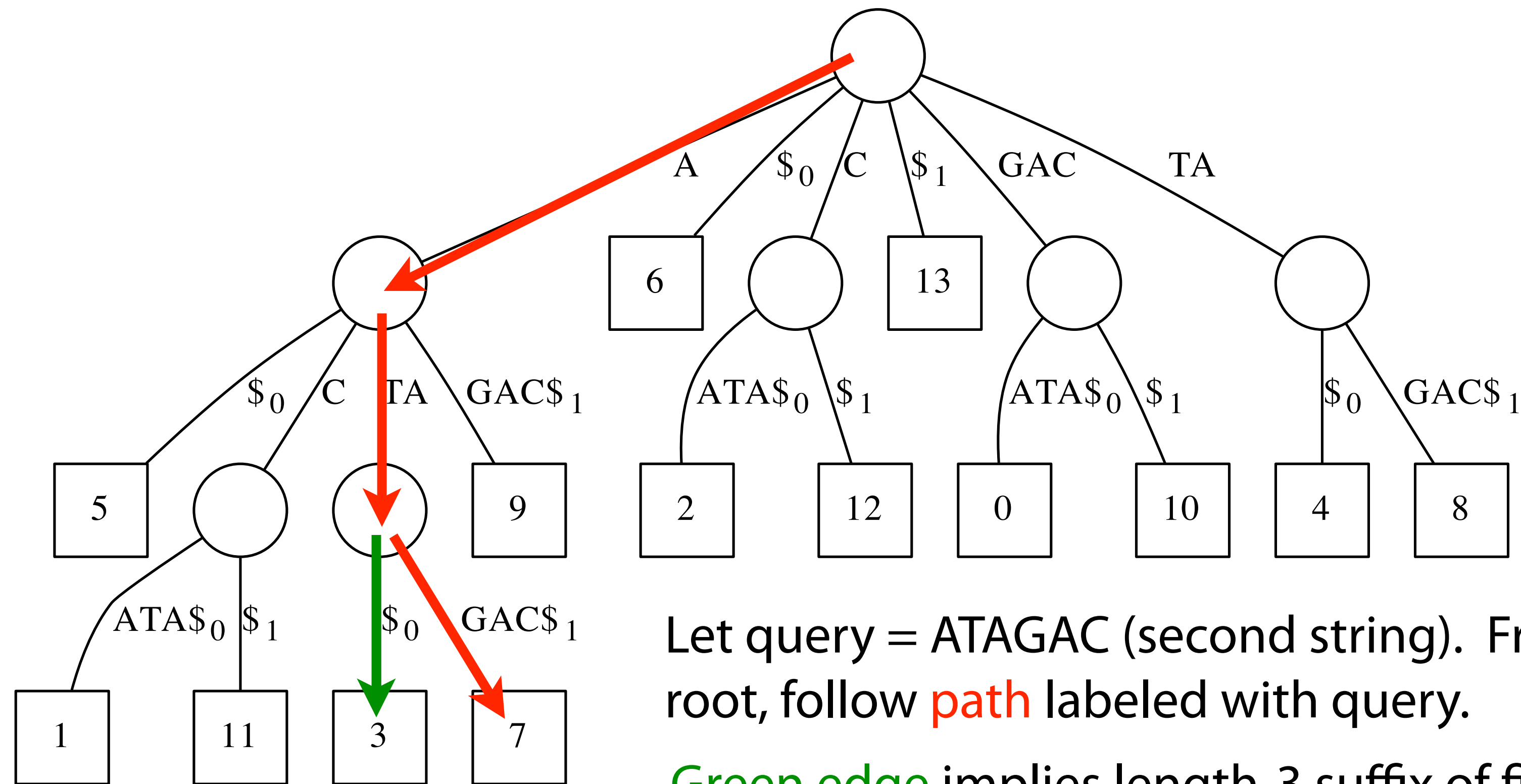
Let query = GACATA (first string). From root, follow **path** labeled with query.

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

# Finding overlaps with suffix tree

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

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



GACATA  
|||  
ATAGAC

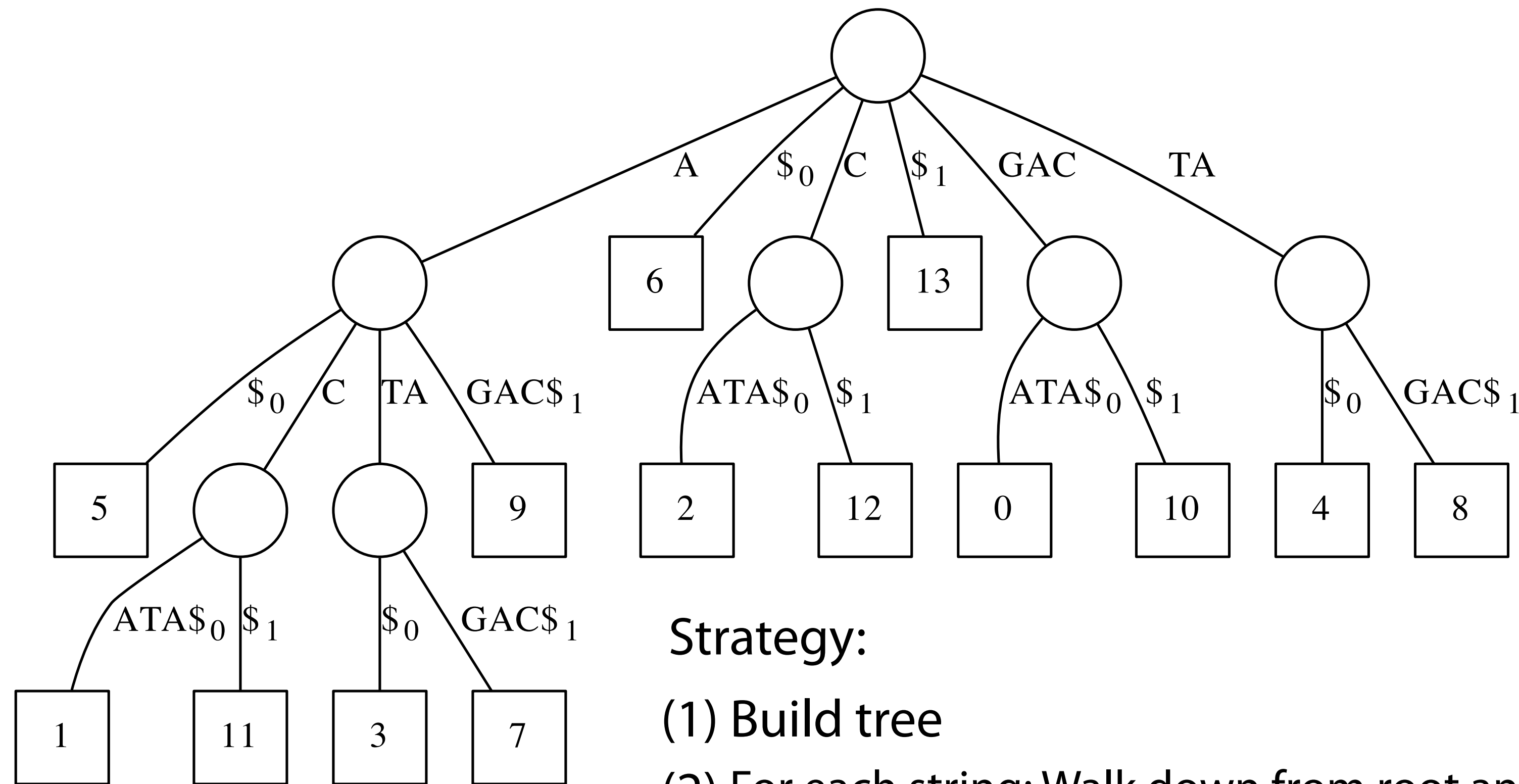
Let query = ATAGAC (second string). From root, follow **path** labeled with query.

**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" }

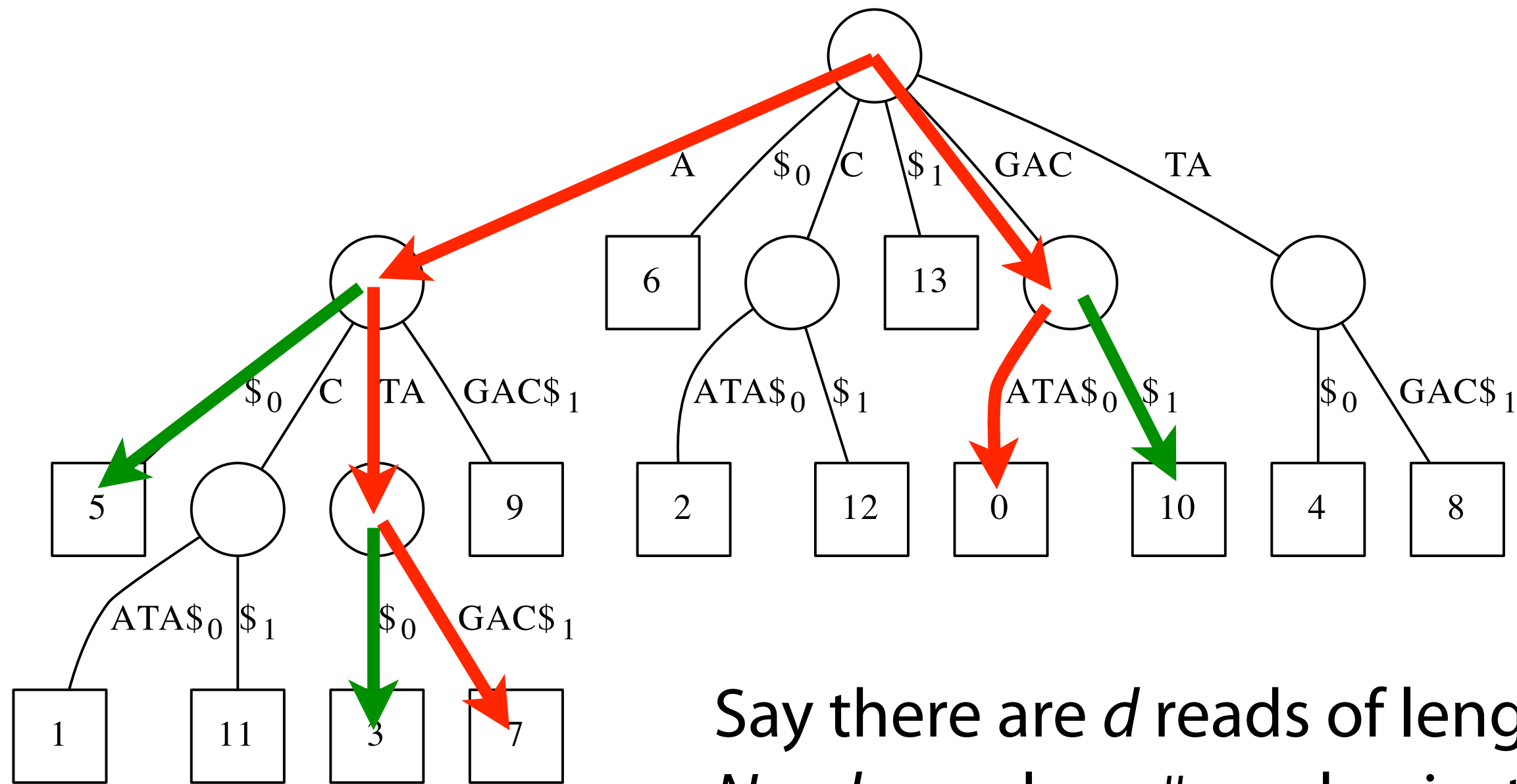
GACATA\$<sub>0</sub>ATAGAC\$<sub>1</sub>



## 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	16	16	24	26	30	36	44	52	60
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	8	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

$X$

$X$ : 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:  $O(d^2)$

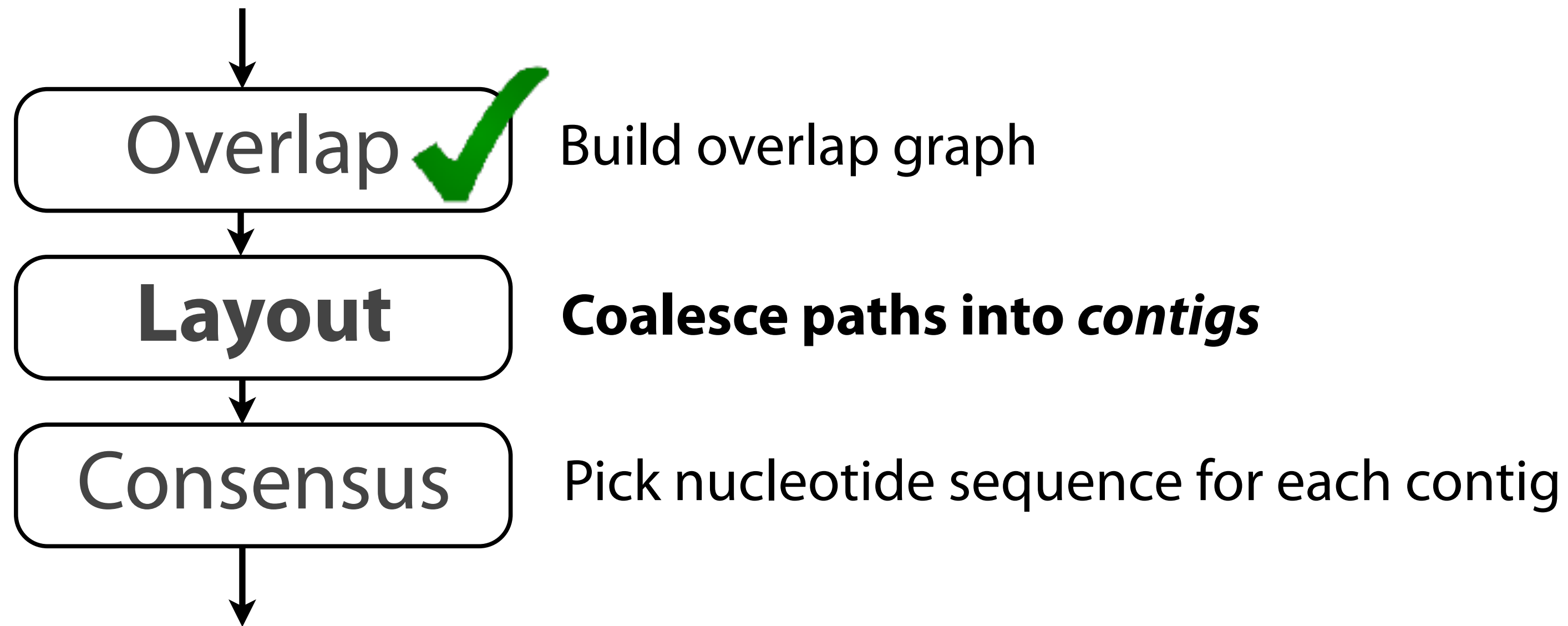
Size of each DP matrix:  $O(n^2)$

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



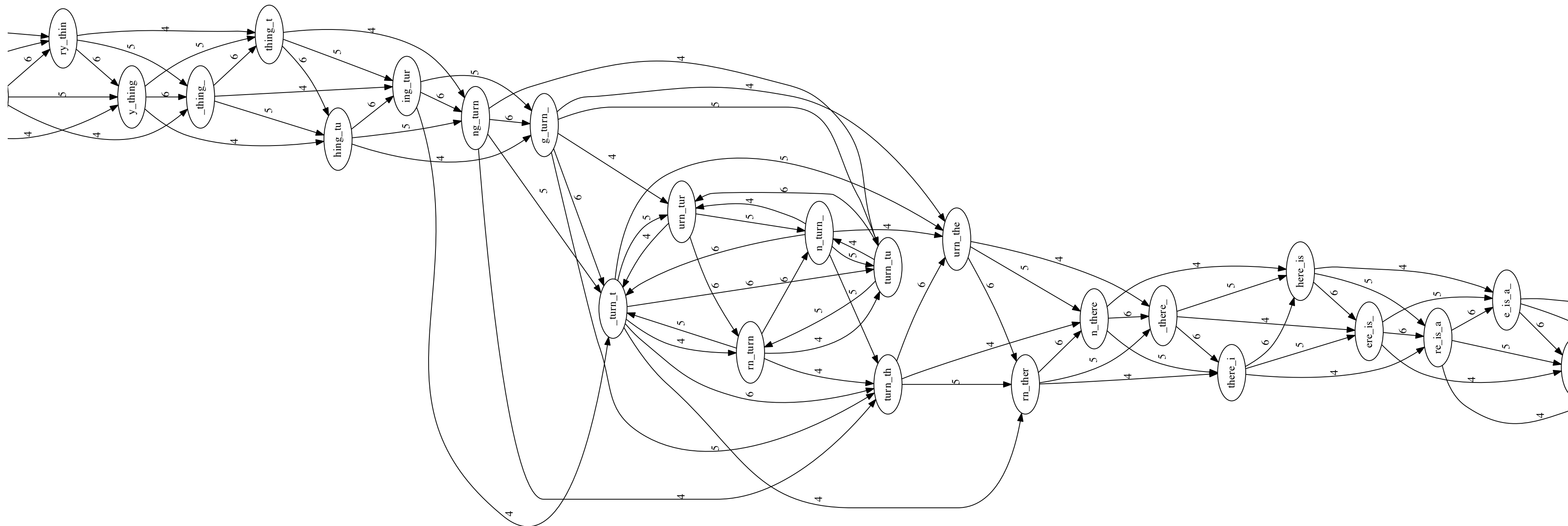
# Layout

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`

$l = 4, k = 7$

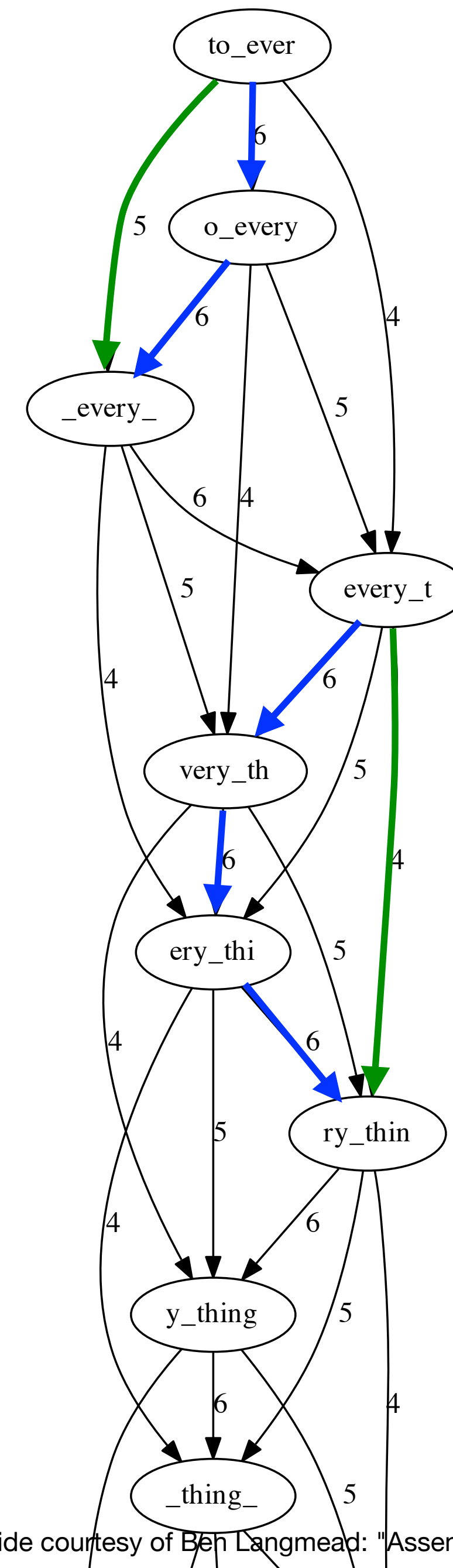


# Layout

Anything redundant about this part of the overlap graph?

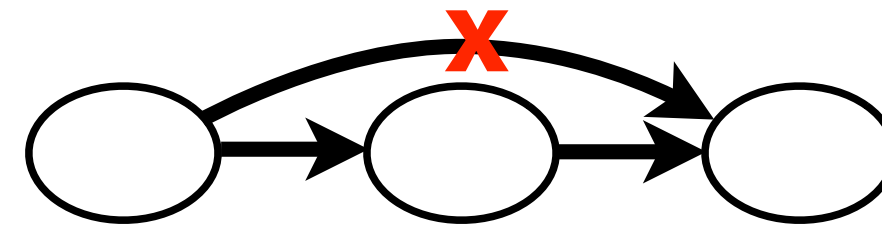
Some edges can be *inferred (transitively)* from other edges

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

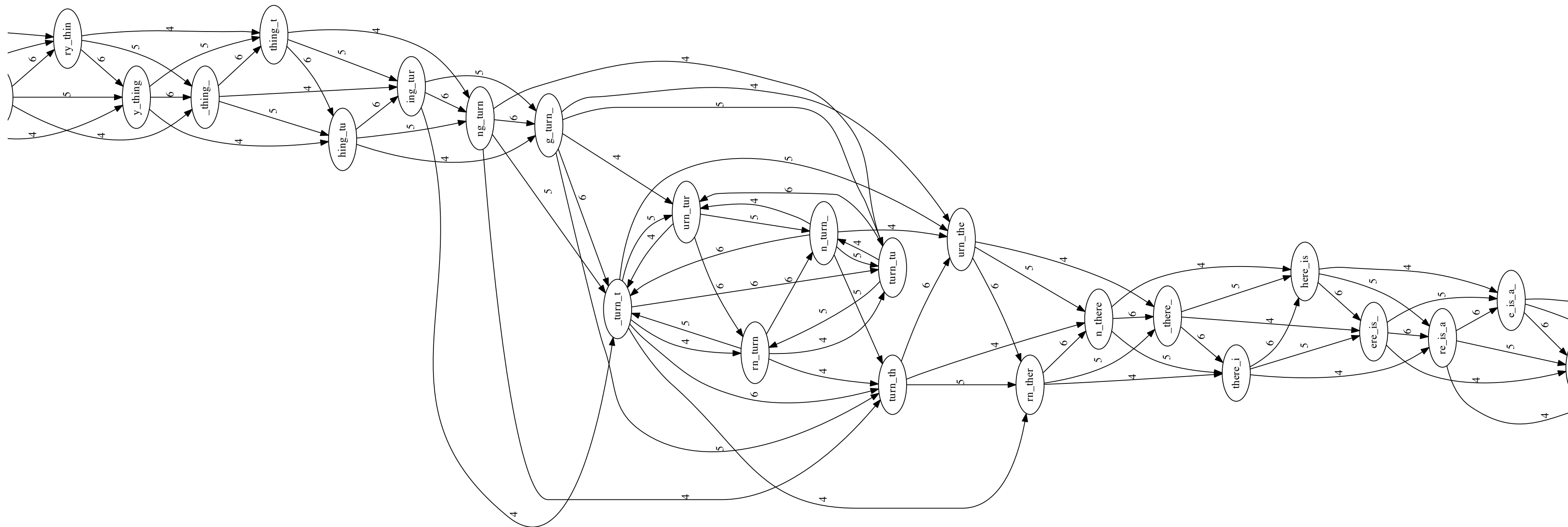


# Layout

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

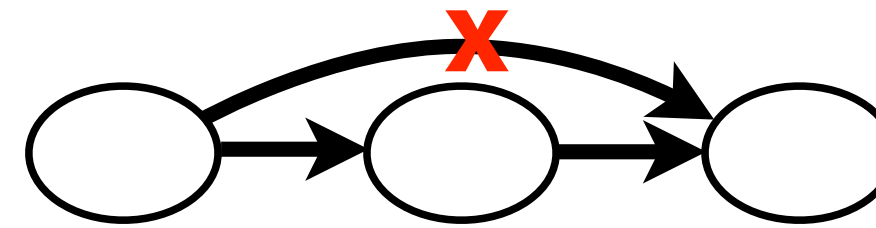


Before:

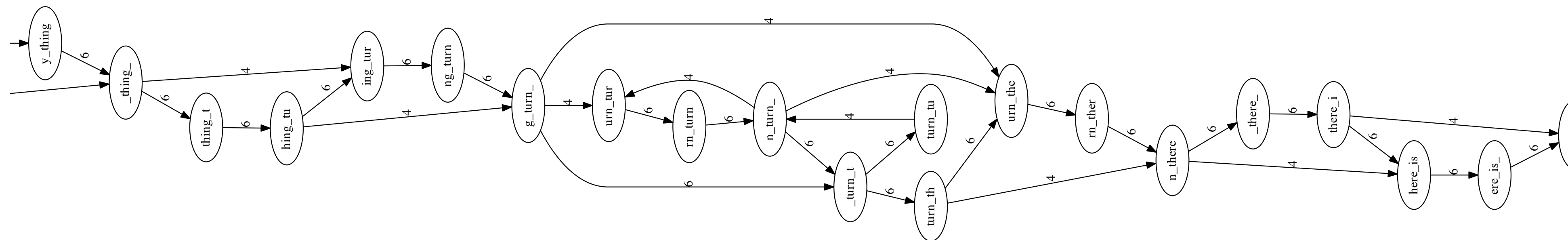


# Layout

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

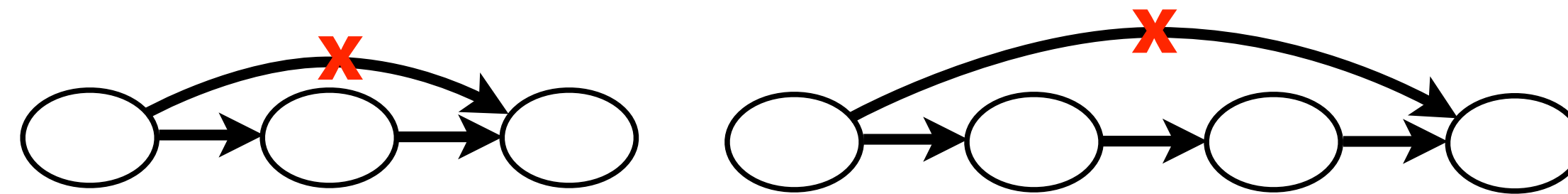


After:

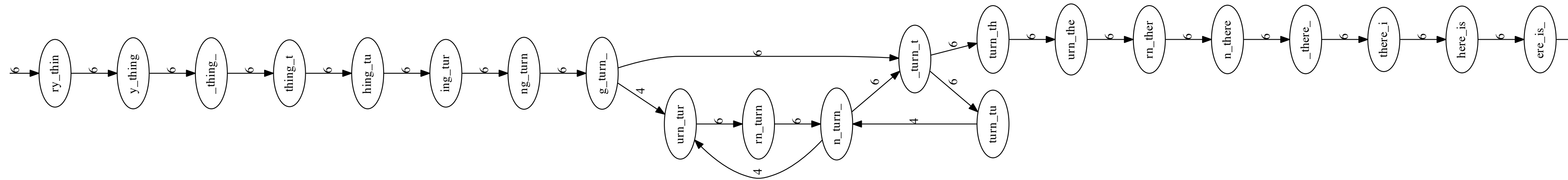


# Layout

Now remove edges that skip one or two nodes:



After:

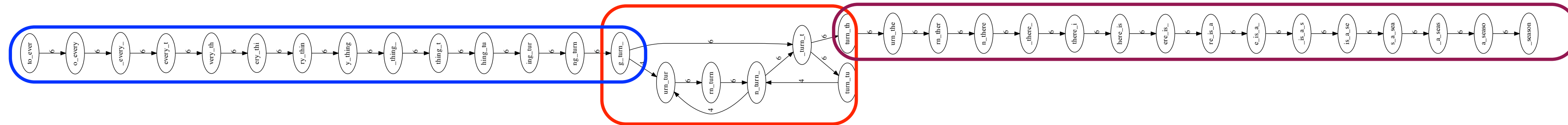


Even simpler



# Layout

Emit *contigs* corresponding to the non-branching stretches



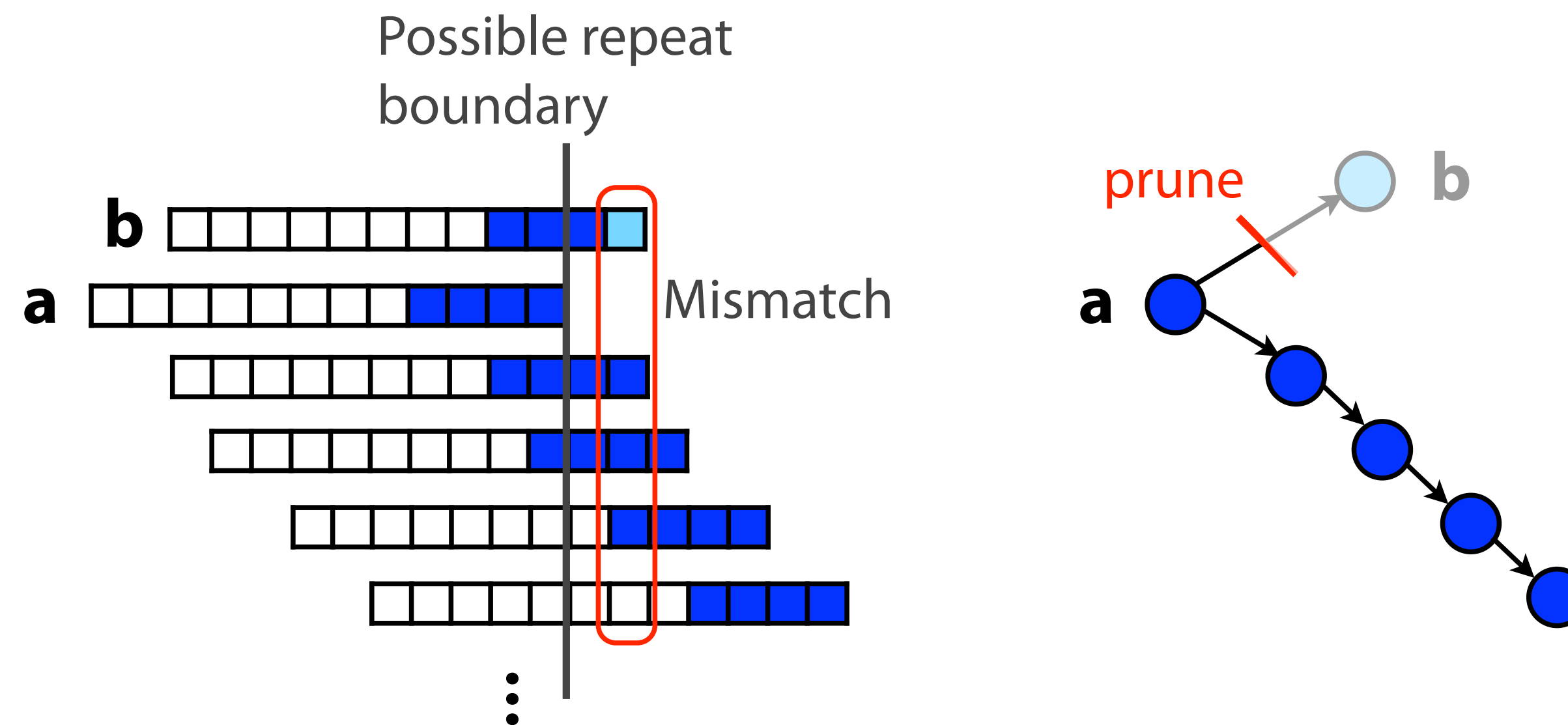
Contig 1  
to\_every\_thing\_turn\_

Contig 2  
turn\_there\_is\_a\_season

┌───┐  
Unresolvable repeat

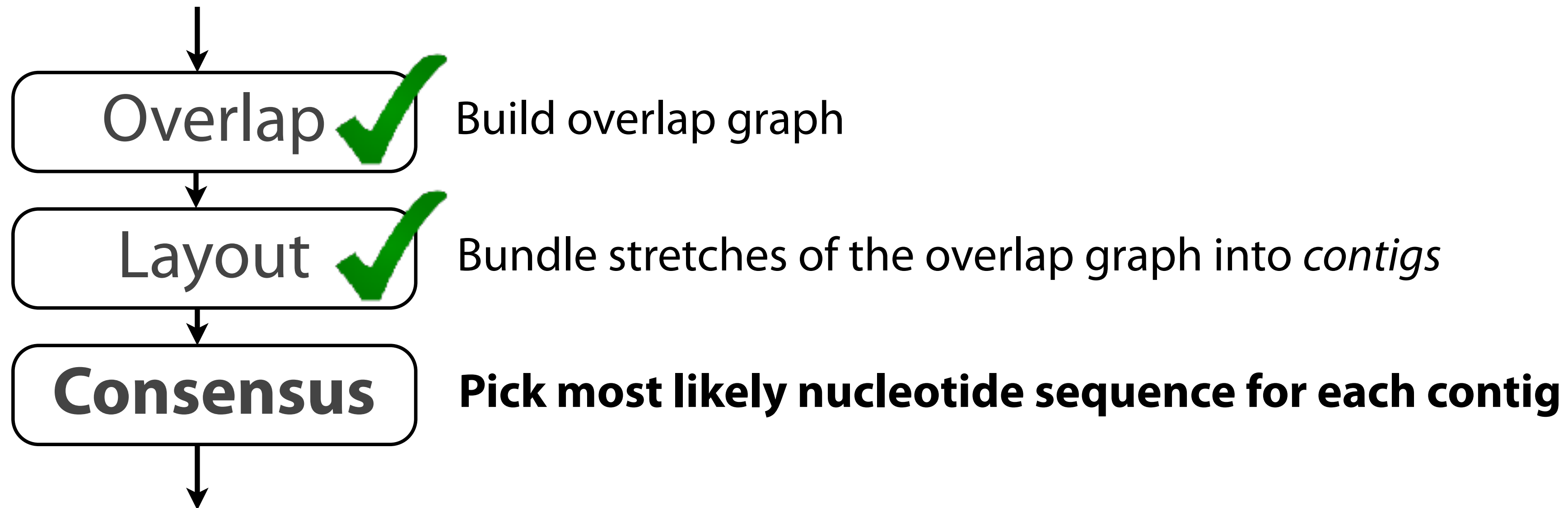
# Layout

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



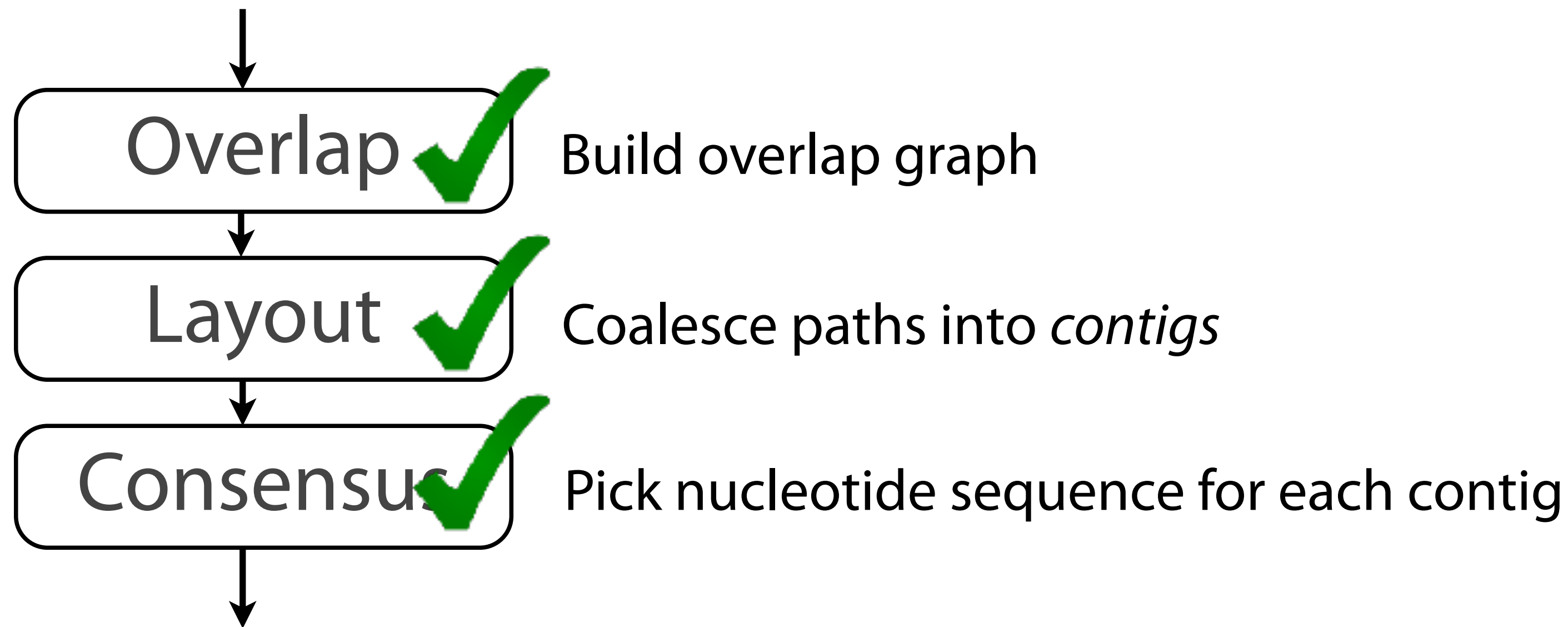
Take reads that make  
up a contig and line  
them up

↓ ↓ ↓ ↓ ↓  
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.

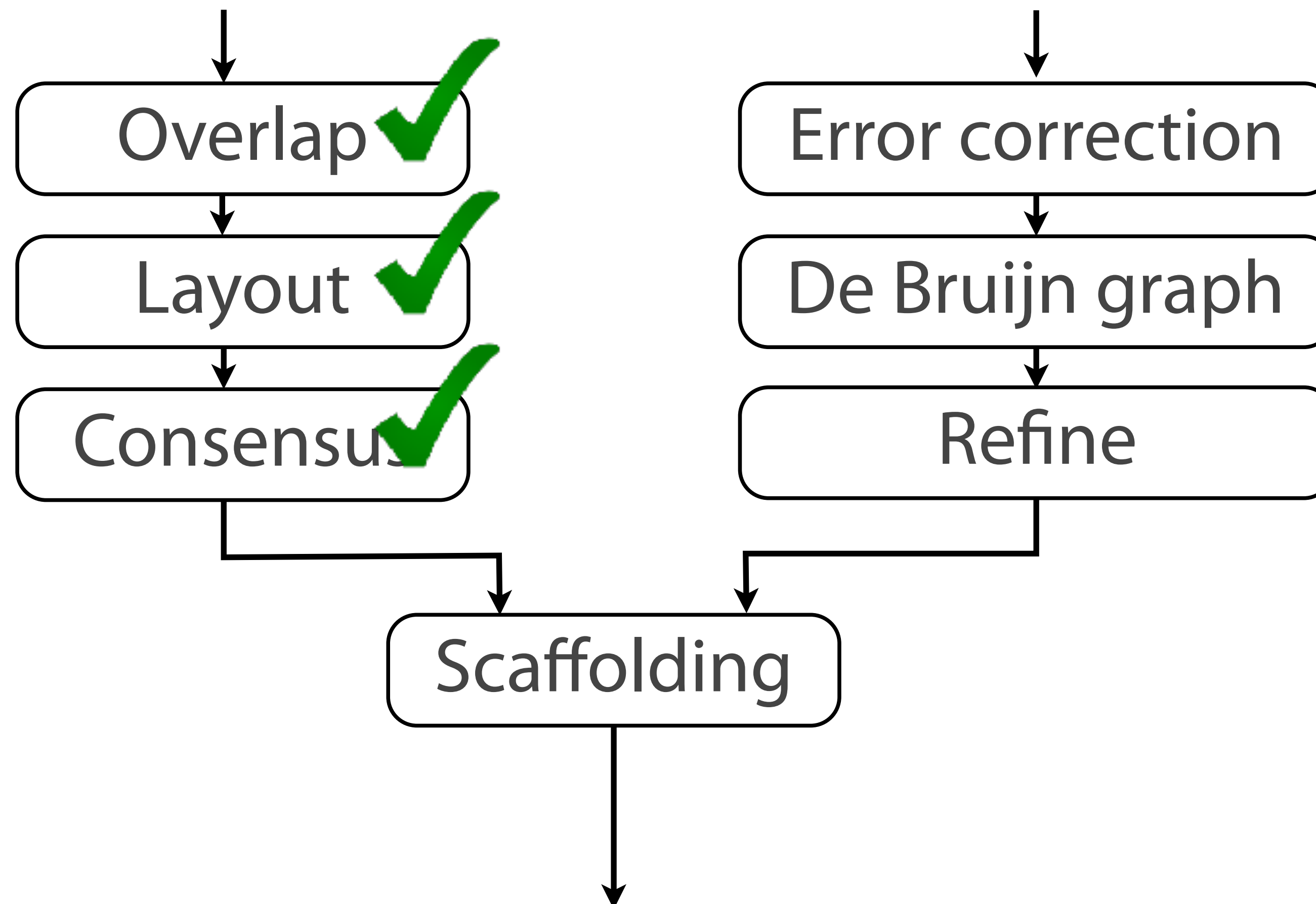
Overlap graph is big; one node per read, # edges can grow superlinearly with # reads

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

# Assembly alternatives

Alternative 1: Overlap-Layout-Consensus (OLC) assembly

Alternative 2: De Bruijn graph (DBG) assembly



# Some quick terminology

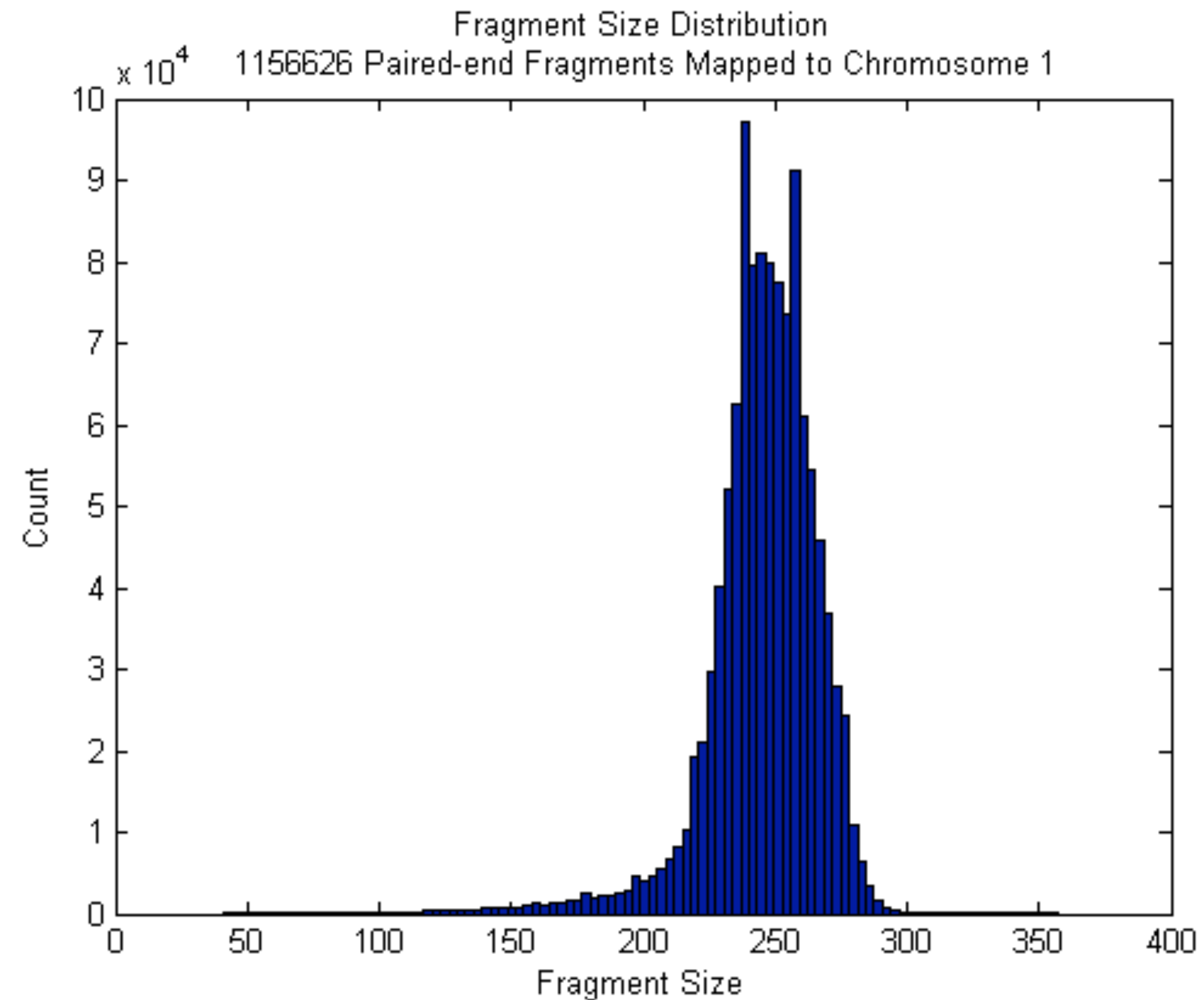
DNA is sequences into short ***reads*** which are parts of the sequence, which are assembled into contiguous unambiguous sections, or ***contigs***, which are not typically the full length of the original sequence.

Using ***paired-end*** reads, we can construct a ***scaffold*** which tell us how far apart the contigs should be with some unknown sequence in the middle (of somewhat known length).

# Scaffolding: paired-end sequencing

Example fragment length distribution

Fragments are not exactly the same length, but there's a clear peak around 250 nt, very few < 150 nt or > 300 nt

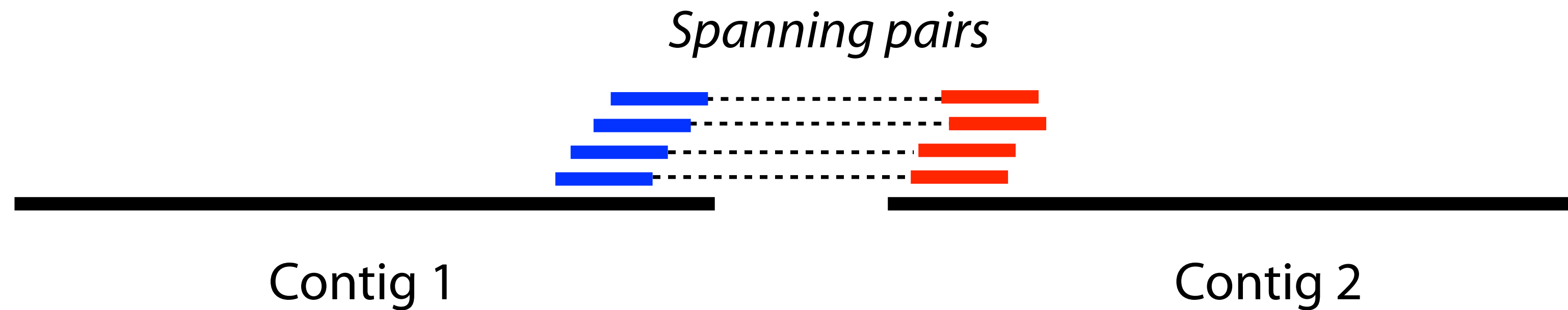




# Scaffolding: paired-end sequencing

Say we have a collection of pairs and we assemble them as usual

Assembly yields two contigs:



...and we discover that some of the mates at one edge of contig 1 are paired with mates in contig 2

Call these *spanning pairs*

# Scaffolding: paired-end sequencing

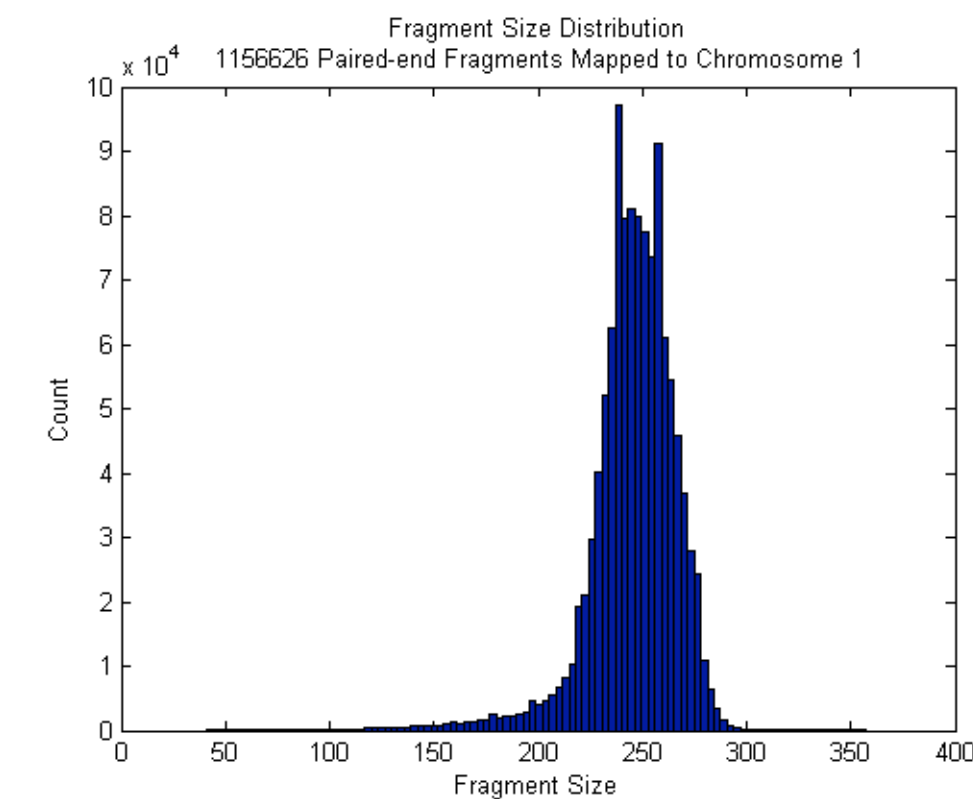


What does this tell us?

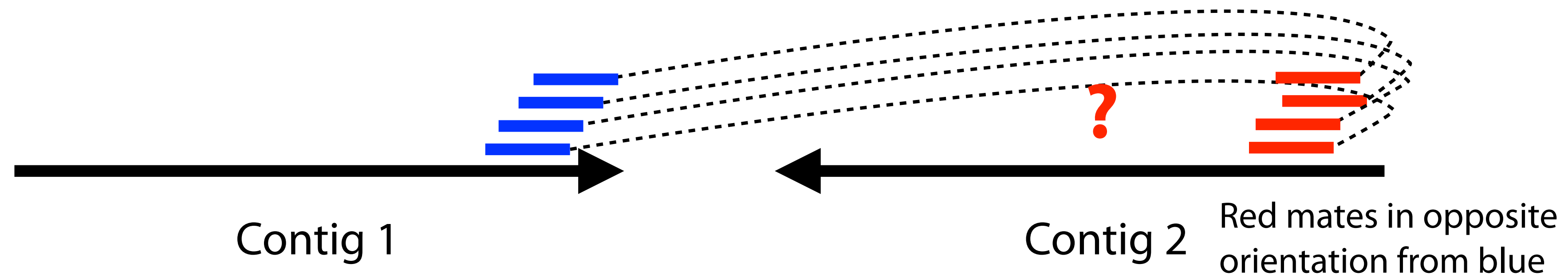
Contig 1 is close to contig 2 in the genome

In fact, we can *estimate distance between contigs* using what we know about fragment length distribution

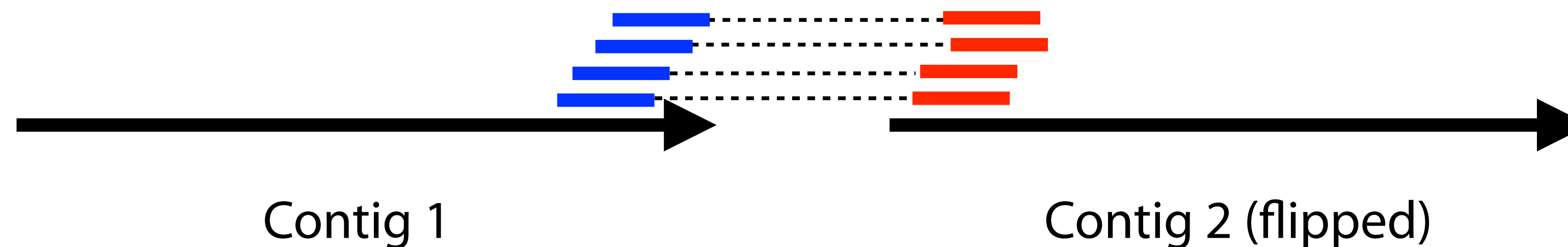
The more spanning pairs we have, the better our estimate



# Scaffolding: paired-end sequencing



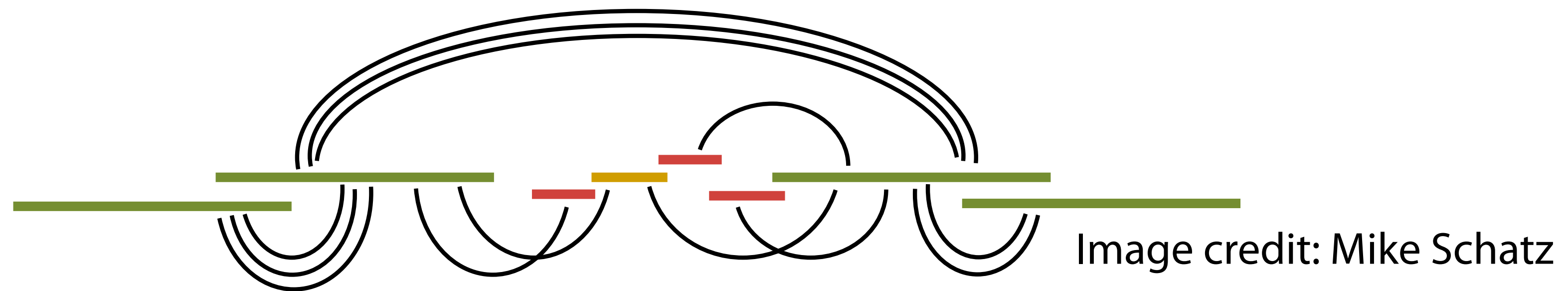
What does the picture look like if contigs 1 and 2 are close, but we assembled contig 2 “backwards” (i.e. reverse complemented)



Pairs also tell us about contigs' relative *orientation*

# Scaffolding

Scaffolding output: collection of *scaffolds*, where a scaffold is a collection of contigs related to each other with high confidence using pairs



# Some quick terminology

DNA is sequences into short ***reads*** which are parts of the sequence, which are assembled into contiguous unambiguous sections, or ***contigs***, which are not typically the full length of the original sequence.

Using ***paired-end*** reads, we can construct a ***scaffold*** which tell us how far apart the contigs should be with some unknown sequence in the middle (of somewhat known length).

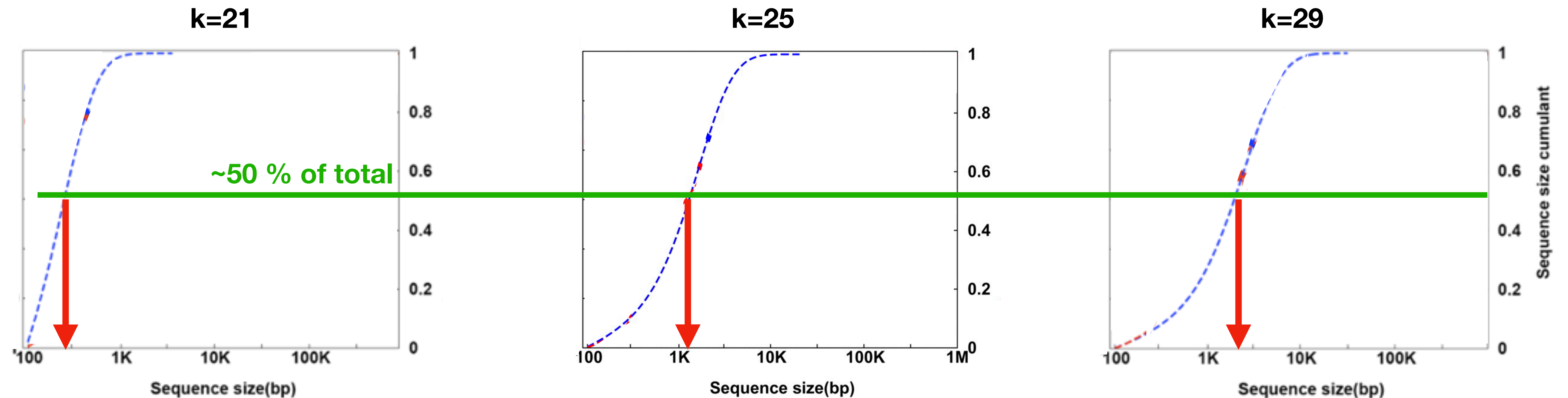
The ***N50*** value of an assembly, is the size of the contig such that half of the total size of the assembled sequences is in smaller contigs. Specifically:

$$N50 =: \left\{ |c_i| \mid \begin{array}{l} \sum_{1 \leq j \leq i} |c_j| \geq \sum_{i < j \leq k} |c_j| \wedge \\ \sum_{1 \leq j < i} |c_j| \leq \sum_{i \leq j \leq k} |c_j| \end{array} \right\}$$

assuming the  $k$  contigs  $c_1, c_2, \dots, c_k$  are sorted.

# N50

Assuming increasing  $k$ -mer sizes are used and sequencing is perfect, this is how large the conigs are expected to be



# short oligonucleotide alignment program, *de novo* (SOAPdenovo)

## Resource

---

### De novo assembly of human genomes with massively parallel short read sequencing

Ruiqiang Li,<sup>1,2,3</sup> Hongmei Zhu,<sup>1,3</sup> Jue Ruan,<sup>1,3</sup> Wubin Qian,<sup>1</sup> Xiaodong Fang,<sup>1</sup>  
Zhongbin Shi,<sup>1</sup> Yingrui Li,<sup>1</sup> Shengting Li,<sup>1</sup> Gao Shan,<sup>1</sup> Karsten Kristiansen,<sup>1,2</sup>  
Songgang Li,<sup>1</sup> Huanming Yang,<sup>1</sup> Jian Wang,<sup>1</sup> and Jun Wang<sup>1,2,4</sup>

<sup>1</sup>Beijing Genomics Institute at Shenzhen, Shenzhen 518083, China; <sup>2</sup>Department of Biology, University of Copenhagen, Copenhagen DK-2200, Denmark



# Sequencing and error correction

The first step taken in this (and several algorithms of this type) is read-error correction.

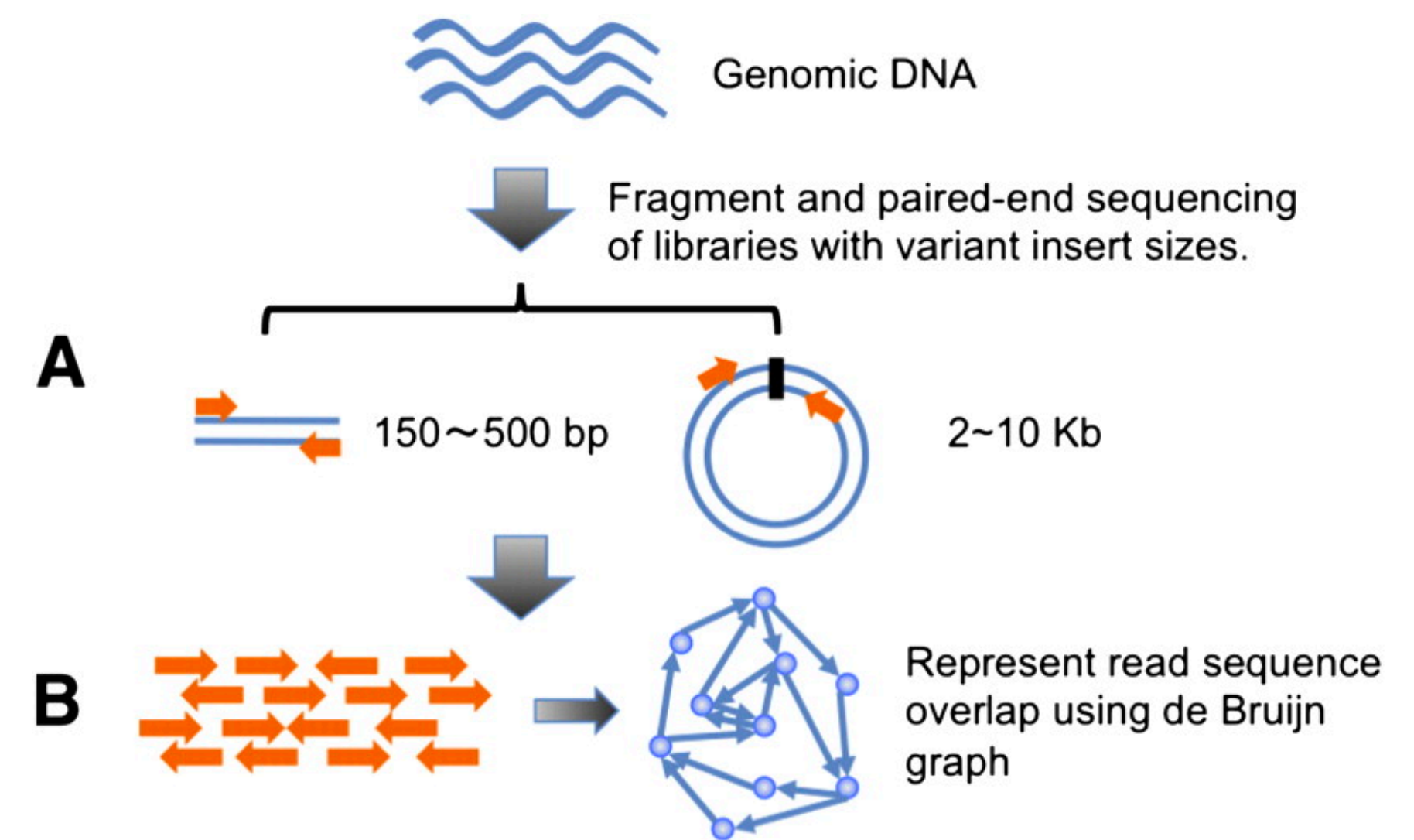
Correct  $k$ -mers are going to appear multiple times in the reads set

- random sequencing error-containing  $k$ -mers have low frequency

Build a hash table to store the frequency of all 17-mers

- for each read, start from the high-frequency regions and extended both sides to infer potential erroneous sites of low-frequency ( $<3$ ) 17-mers
- for each inferred erroneous site, test the impact of changing it to the other three allele types
- accept if all overlapping  $k$ -mers had a frequency equal to or over 3

Dynamic programming was used to find the optimal solution with minimal changes





# Graph correction

## Tips

- any path that is a "dead end" and less  $<2k$  in length is removed

## Low Coverage Links

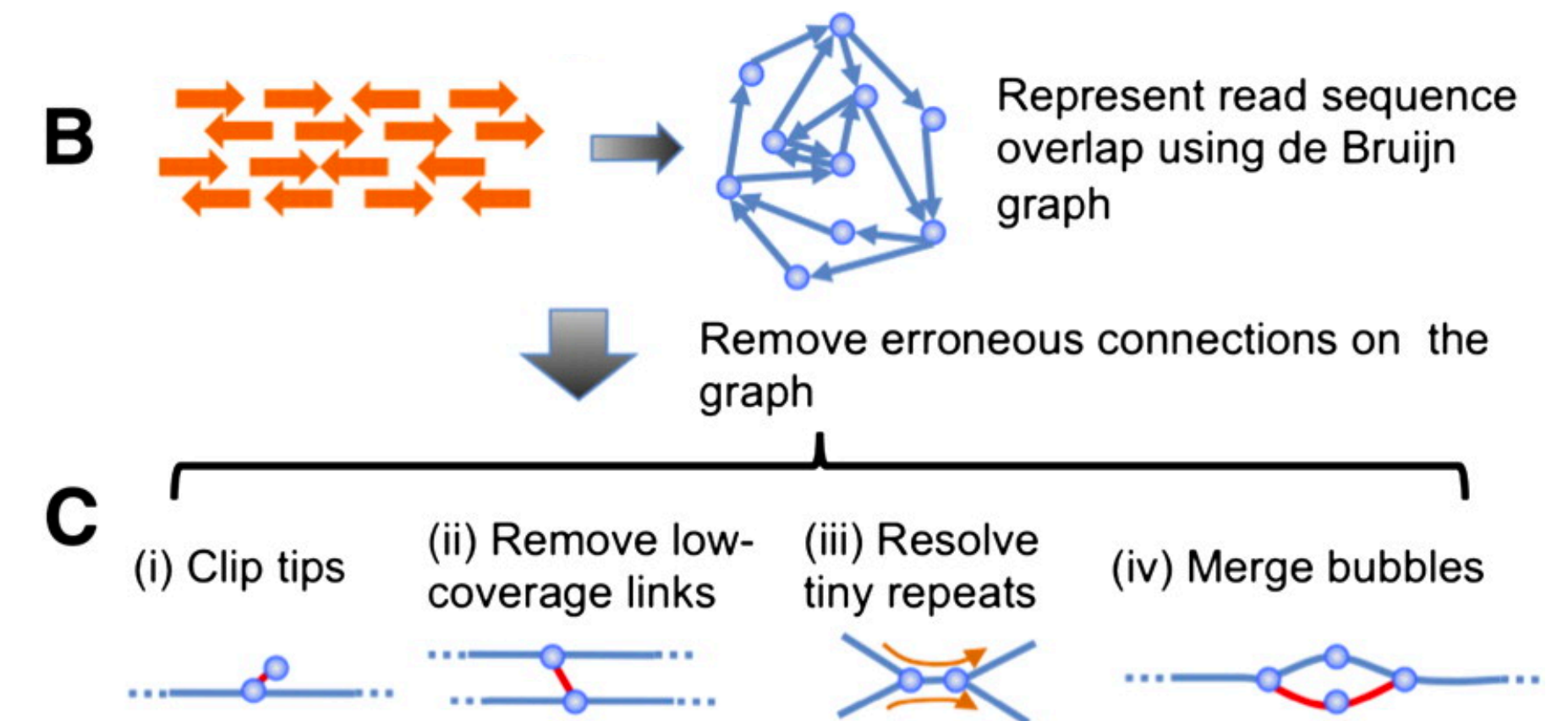
- any node that is only in 1 read should be removed

## Tiny Repeats

- if shorter than a read length, use a read to resolve the repeated sequence paths

## Bubbles

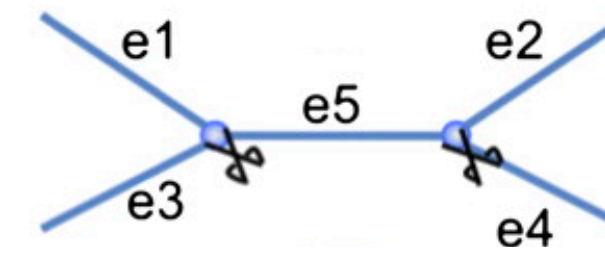
- if the two paths only have one base change, or 90% identity remove the lower coverage path



# Initial Contigs

Remaining repeat sections are split into separate contigs

**D**



Break at repeat boundaries  
and output contigs

# Scaffold Construction

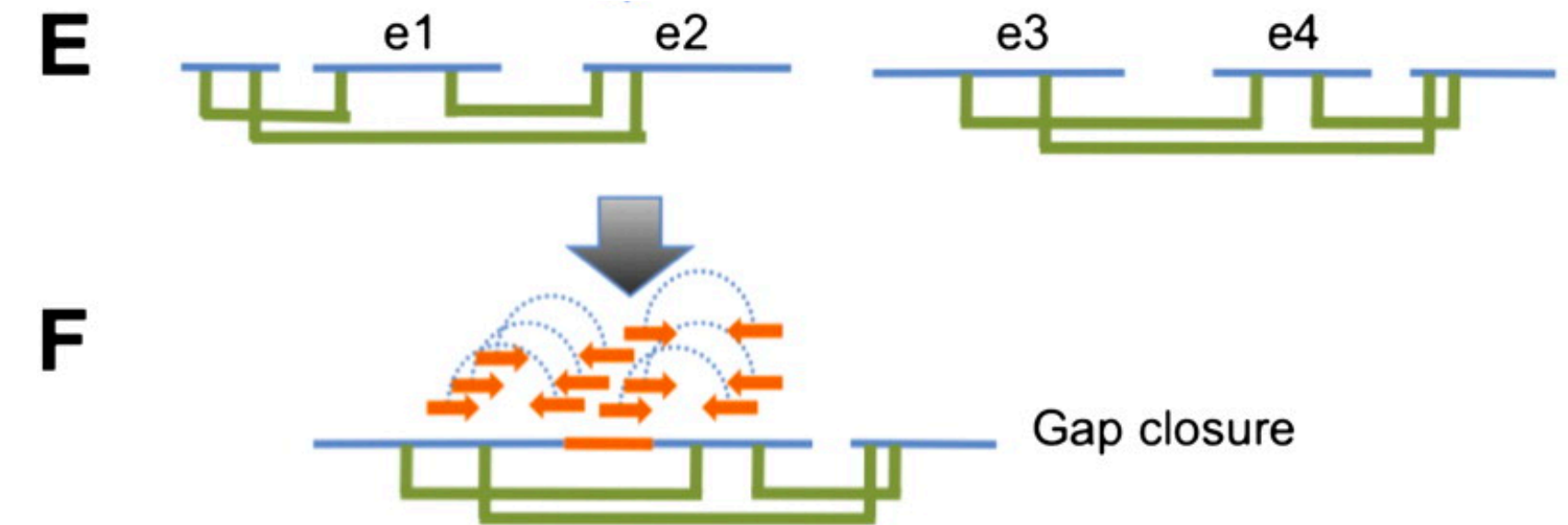
Reads are remapped onto the contigs

- the reads and their mate pair create a graph connecting them

Then the graph is linearized by:

- grouping compatible transitive links
- unresolvable repeat structures are masked (i.e. replaced with unknown characters for a specific length)

When possible, polish the gaps by finding read pairs that map to known sections on one end and the repeat section on another



# The final assemblies

Data set	Step	Sequence depth	N50 (bp)	N90 (bp)	Total length	Genome coverage	Gene coverage
Asian genome	Contig	52×	1050	205	2,146,837,026	80.3%	93.4%
	Scaffold (135&440bp PE)	26×	17,331	3838	2,510,643,840	80.3%	93.4%
	Scaffold (+2.6 kb PE)	5×	103,474	21,431	2,718,204,301	80.3%	93.4%
	Scaffold (+6 kb PE)	4×	230,544	47,127	2,800,570,159	80.3%	93.4%
	Scaffold (+9.6 kb PE)	2×	446,283	78,405	2,874,204,399	80.3%	93.4%
	Contig after gap closure		7384	1376	2,457,434,692	87.4%	95.5%
African genome	Contig	40×	886	185	2,098,284,706	79.8%	87.7%
	Scaffold (200bp PE)	40×	4474	936	2,375,357,508	79.8%	87.7%
	Scaffold (+2 kb PE)	4×	61,880	5994	2,696,443,788	79.8%	87.7%
	Contig after gap closure		5909	1004	2,367,973,949	85.4%	89.2%

All read sequences were used in contig assembly, while paired-end libraries with different insert sizes were used step-by-step additively on scaffold construction. N50 of contig or scaffold was calculated by ordering all sequences, then adding the lengths from longest to shortest until the summed length exceeded 50% of the total length of all sequences. N90 is similarly defined. NCBI build 36.1 was used as the reference genome and RefSeq was used as the gene set to evaluate genome and gene region coverage. Since both genomes were sequenced of male individuals, chromosomes X and Y only have half-sequencing depths of the autosomes, and hence were excluded in calculation genome and gene coverage. For calculating scaffold N50 and total length, the intrascaffold gaps were included.

# Compute time

**Table 4.** Statistics of computational complexity at each assembly step

Step	Human African			Human Asian		
	Peak memory (Gb)	No. of CPUs	Time (h)	Peak memory (Gb)	No. of CPUs	Time (h)
Preassembly error correction	96	40	22	96	40	24
Construct de Bruijn graph	140	16	8	140	16	10
Simplify graph and output contigs	62	1	3	108	1	6
Remap reads	43	8	2	74	8	4
Scaffolding	23	1	4	15	1	3
Gap closure	35	8	1	53	8	1
Total	140	—	40	140	—	48

The assemblies were performed on a supercomputer with eight Quad-core AMD 2.3 GHz CPUs with 512 Gb of memory installed, and used the Linux operating system.

**Supplementary table 2.** Statistics of contig size by graph simplification step by step.

Step	Longest (bp)	N50 (bp)	N90 (bp)
Initial de Bruijn graph	425	29	25
Tips clipped	3,836	29	25
Low-coverage removed	3,946	32	25
Tiny repeats solved	15,933	54	25
Bubbles merged	18,483	127	25
Contigs (>=100bp)	18,483	1,050	205