Overlap Layout Consensus assembly

Ben Langmead

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Real-world assembly methods

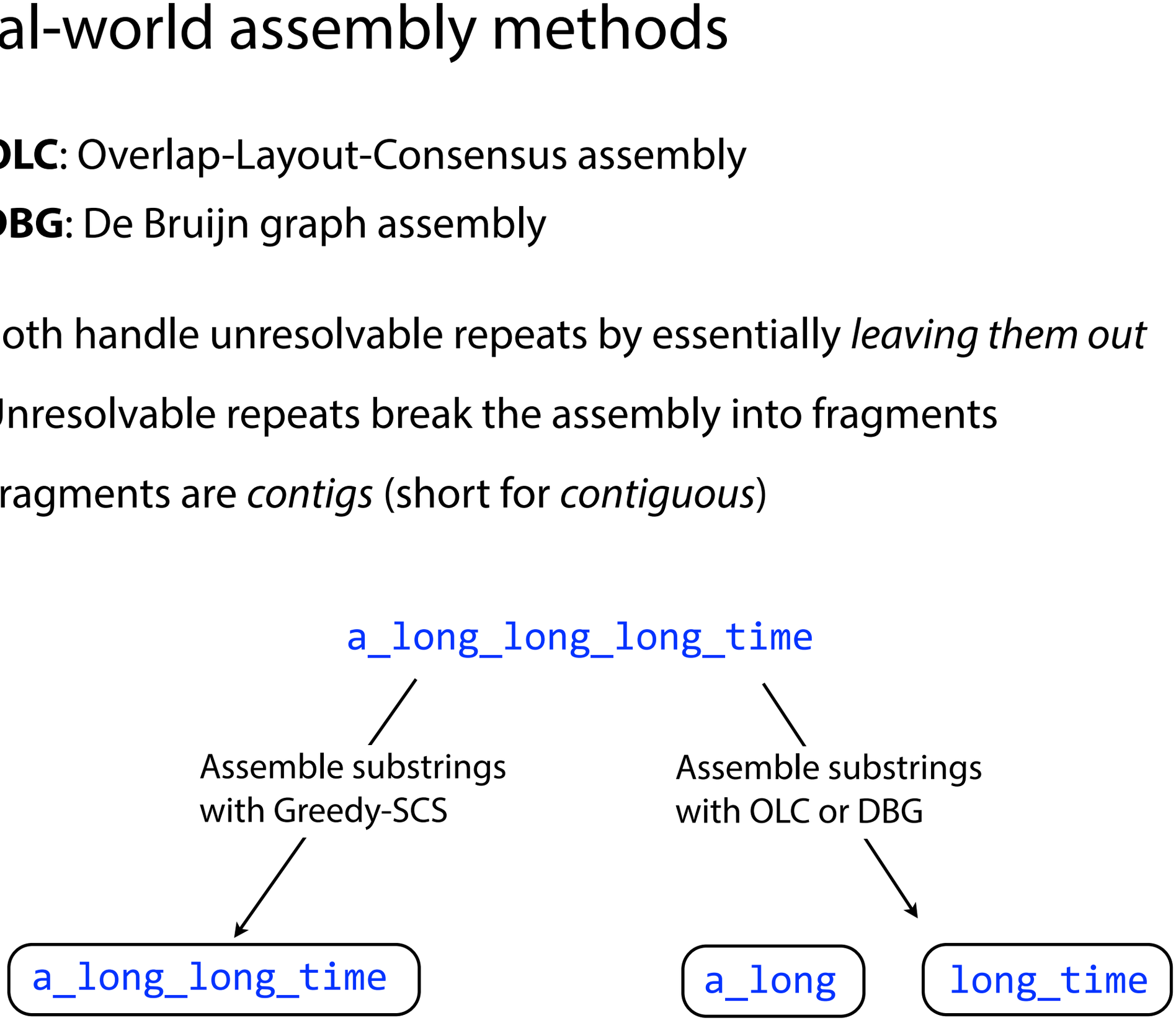
OLC: Overlap-Layout-Consensus assembly

DBG: De Bruijn graph assembly

Both handle unresolvable repeats by essentially *leaving them out*

Unresolvable repeats break the assembly into fragments

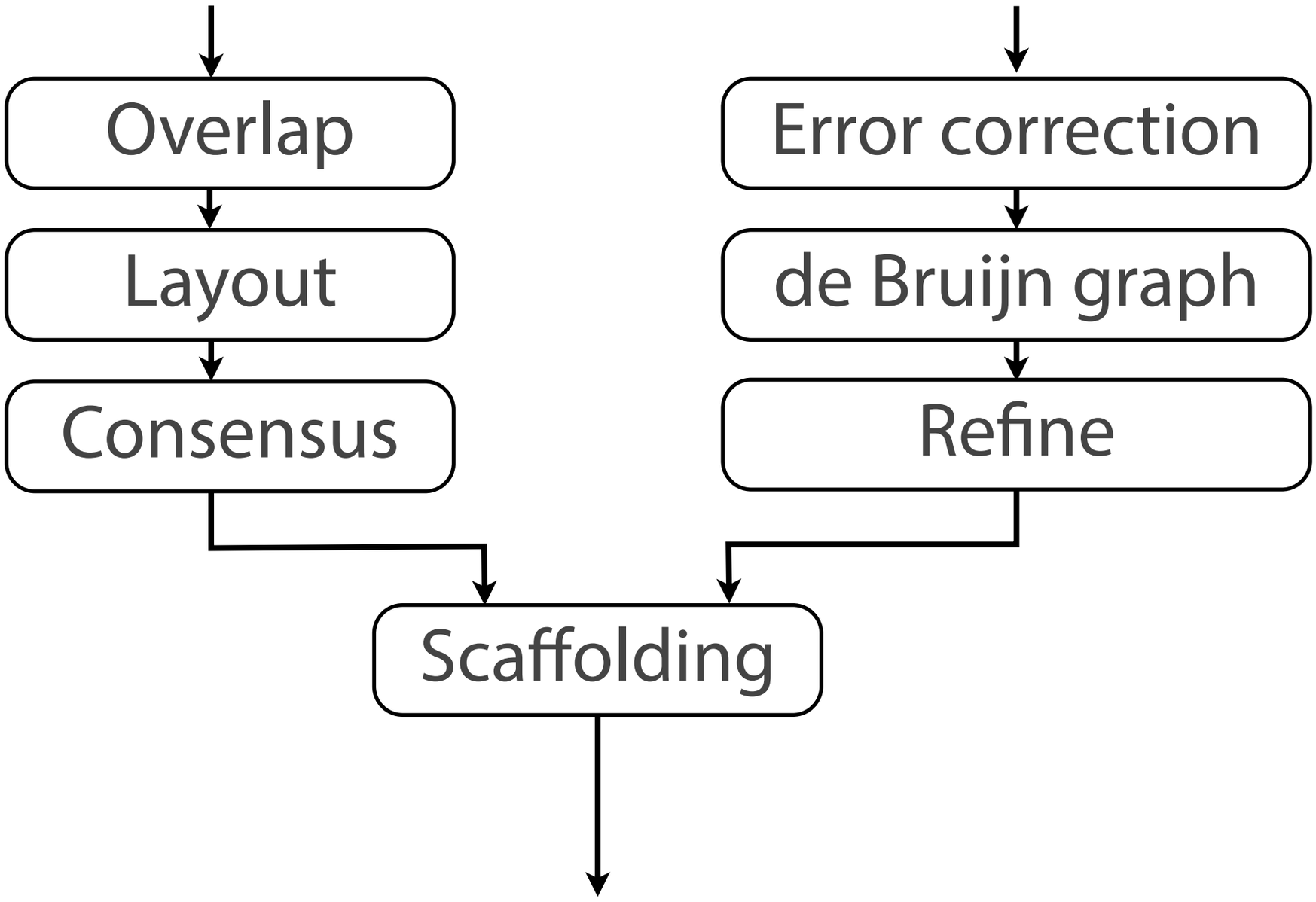
Fragments are *contigs* (short for *contiguous*)



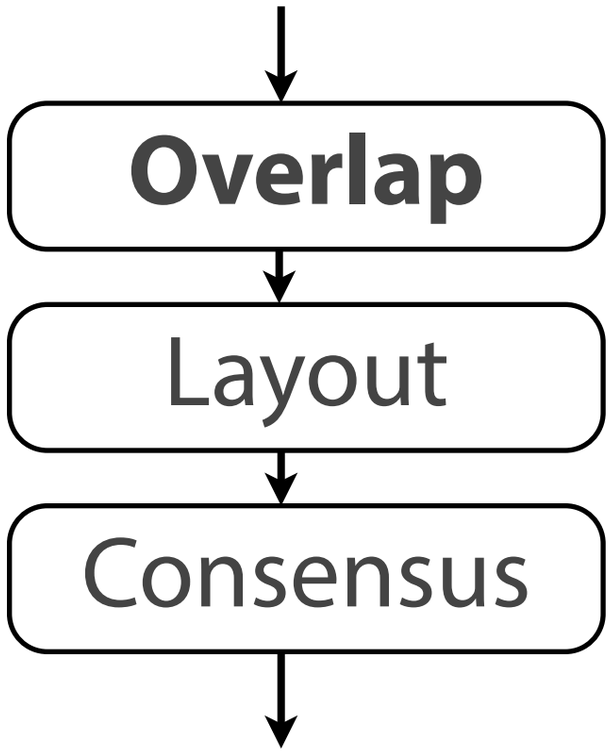
Assembly alternatives

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

Alternative 2: de Bruijn graph (DBG) assembly



Overlap Layout Consensus



Build overlap graph

Bundle stretches of the overlap graph into *contigs*

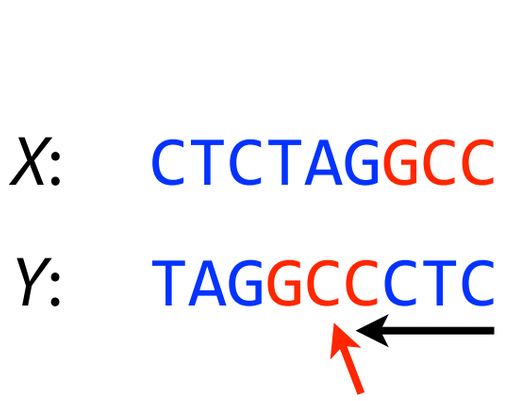
Pick most likely nucleotide sequence for each contig

Finding overlaps

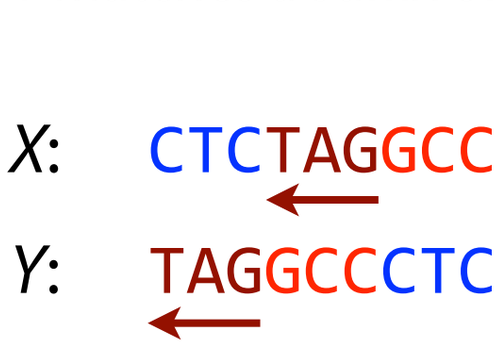
Can we be less naive than this?

Say $l = 3$

Look for this in *Y*, going right-to-left



Extend to left; in this case, we confirm that a length-6 prefix of *Y* matches a suffix of *X*



We're doing this for *every pair* of input strings

Finding overlaps

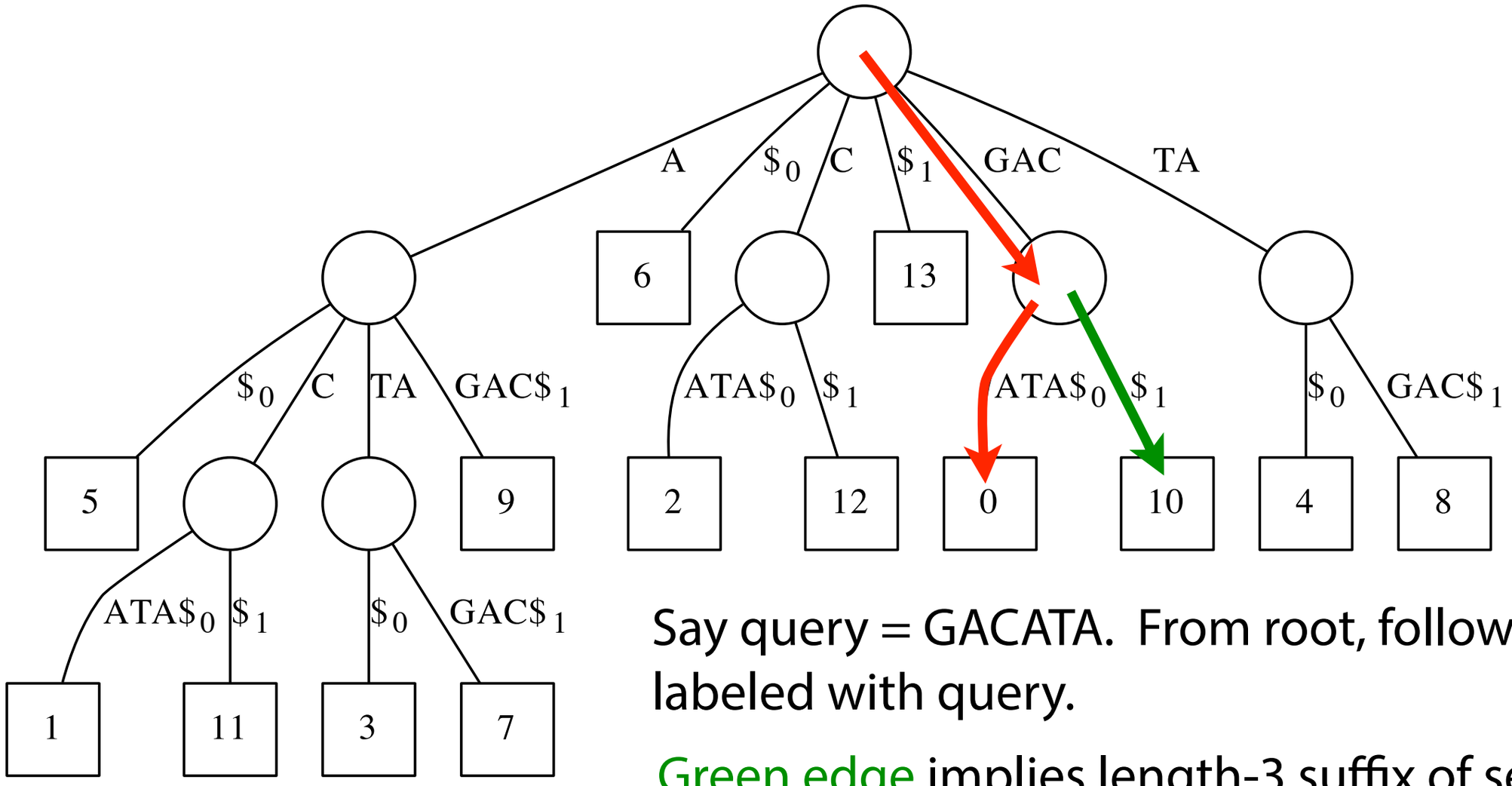
Can we use suffix trees for overlapping?

Problem: 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*

Hint: Build a generalized suffix tree of the strings in *S*

Finding overlaps with suffix tree

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



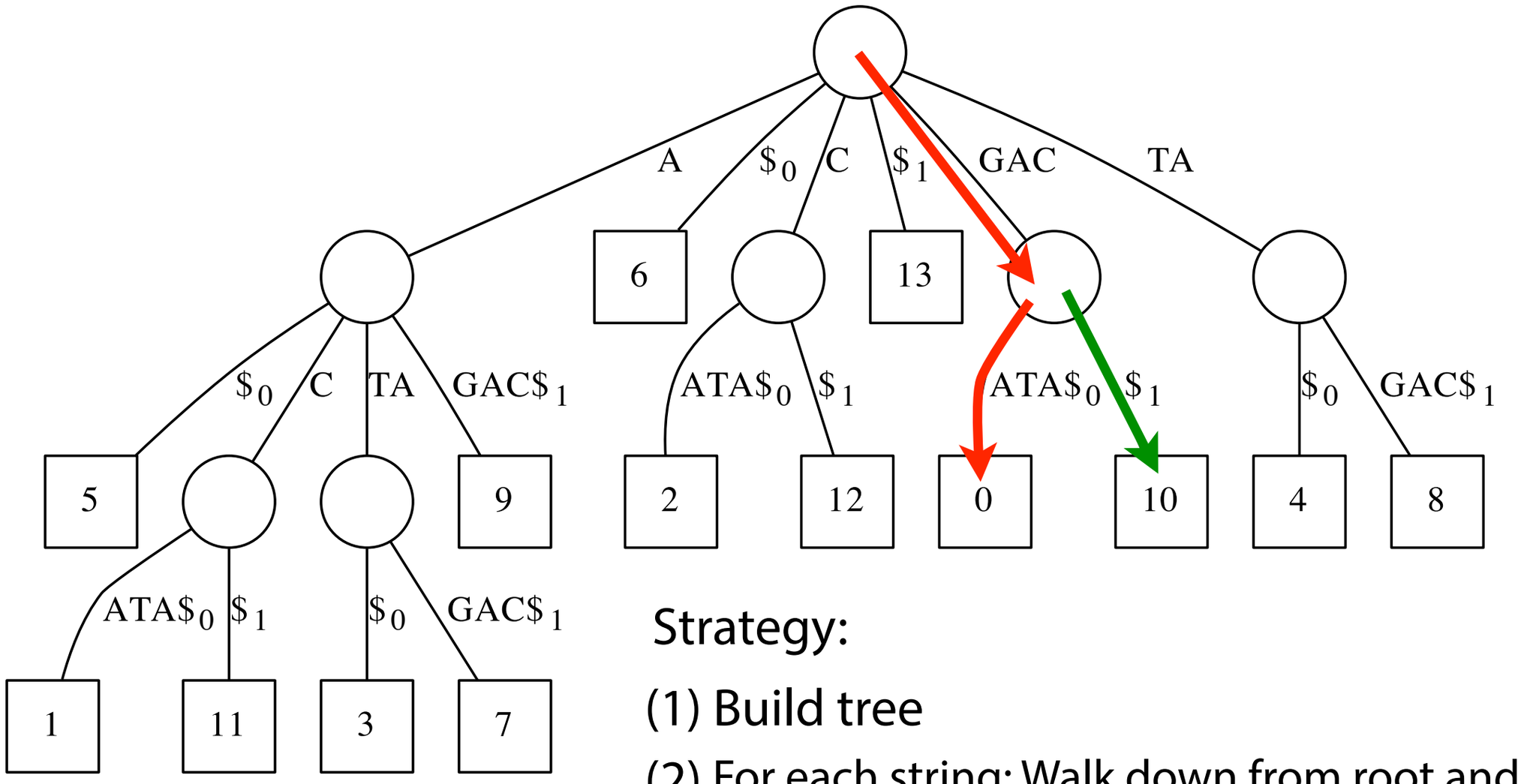
Say query = GACATA. From root, follow path labeled with query.

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$_1$



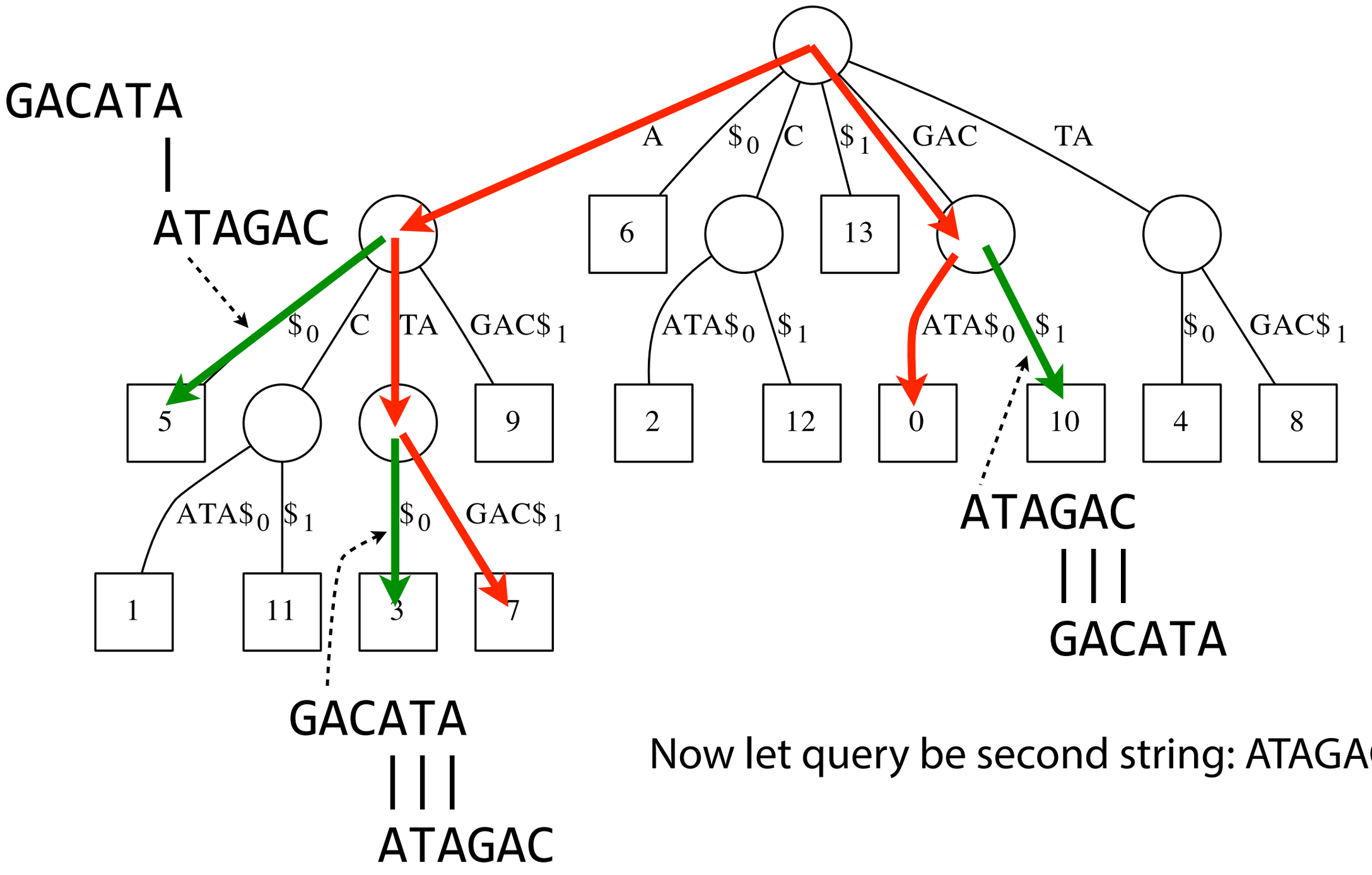
Strategy:

(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.

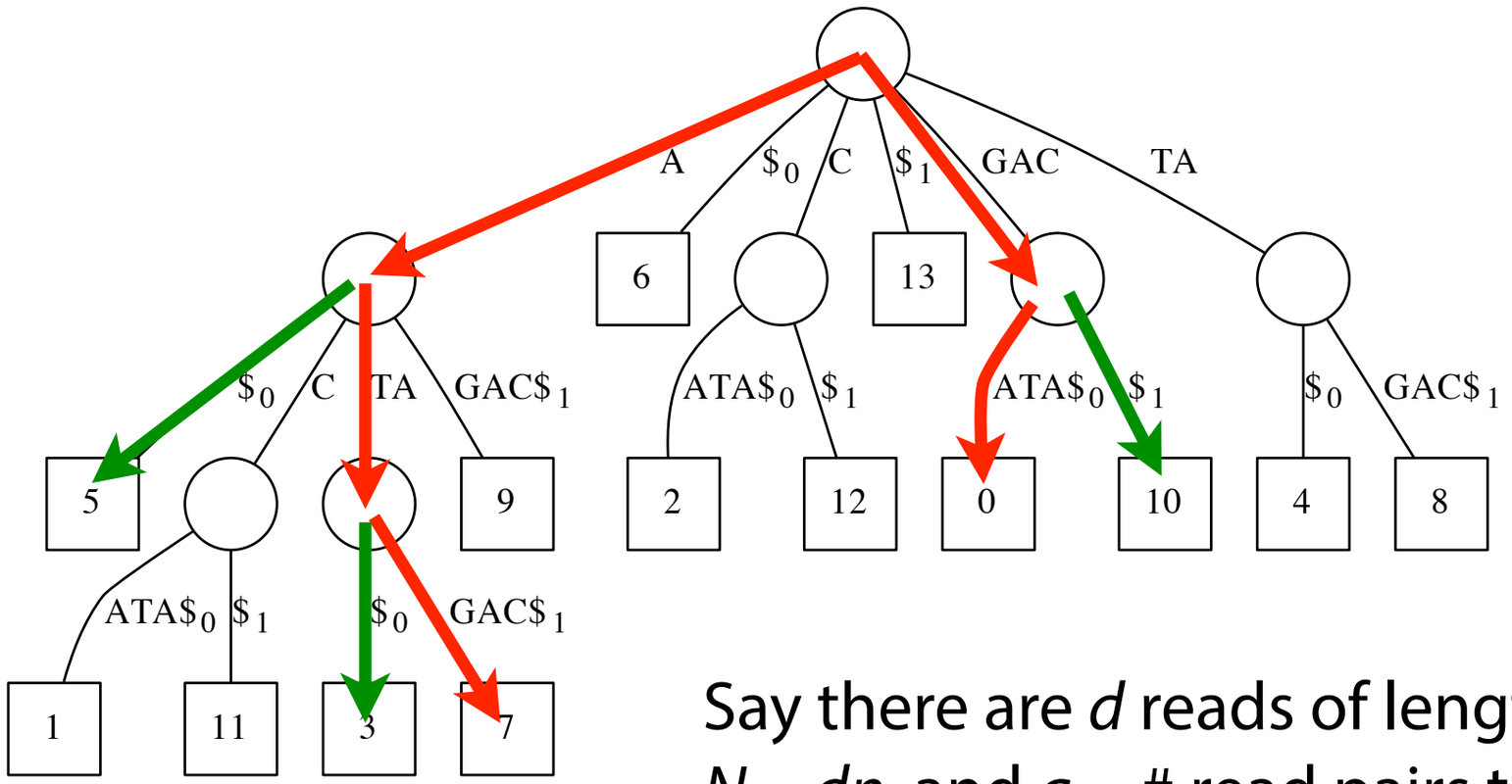
Finding overlaps with suffix tree

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



Now let query be second string: ATAGAC

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:	$O(N + a)$

d^2 doesn't appear explicitly, but a is $O(d^2)$ in worst case

Finding overlaps

What if we want to allow mismatches and gaps in the overlap?

I.e. How do we find the best *alignment* of a suffix of *X* to a prefix of *Y*?

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

Dynamic programming

But we must frame the problem such that only backtraces involving a suffix of *X* and a prefix of *Y* are allowed

Finding overlaps with dynamic programming

Find the best alignment of a suffix of *X* to a prefix of *Y*

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

We'll 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	

But how do we force it to find prefix / suffix matches?

Finding overlaps with dynamic programming

Find the best alignment of a suffix of *X* to a prefix of *Y*

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

$s(a,b)$	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 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

		-	G	G	C	T	C	T	A	G	G	C	C	C
	-	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
	C	0	4	12	20									
	T	0	4	8	14									
	C	0	4	8	8									
	G	0		4	12									
	G	0	0		8	16	16	24	26	30	36	44	52	60
X	C	0	4	4		8	16	18	26	30	34	36	44	52
	C	0	4	8	4		8	16	22	30	34	34	36	44
	C	0	4	8	8	6		10	18	26	34	34	34	36
	T	0	4	8	10	8	8		10	18	26	34	36	36
	A	0	2	6	12	14	12	10		10	18	26	34	40
	G	0	0	2	10	16	18	16	10		10	18	26	34
	G	0	0	0	6	14	20	22	18	10		10	18	26

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

Finding overlaps with dynamic programming

Find the best alignment of a suffix of *X* to a prefix of *Y*

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

$s(a,b)$	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	

Problem: very short matches got high scores by chance...

...which might obscure the more relevant match

Say we want to enforce minimum overlap length $l = 5$

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	20	28	36	44	52	60	68	76	84
C	0	4	8	8	16	20	28	36	44	52	60	68	76
G	0	0	4	12	12	20	24	30	36	44	52	60	68
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	2	10	18	26

Finding overlaps with dynamic programming

Find the best alignment of a suffix of *X* to a prefix of *Y*

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

$s(a,b)$	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	

Solve by initializing certain additional cells to ∞

Cells whose values changed highlighted in red

Now the relevant match is the best candidate

		Y												
		-	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	20	28	36	44	52	60	68	76	84
	C	0	4	8	8	16	20	28	36	44	52	60	68	76
	G	0	0	4	12	12	20	24	30	36	44	52	60	68
	G	0	0	0	8	16	16	24	26	30	36	44	52	60
X	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	∞	4	8	10	8	8	2	10	18	26	34	36	36
	A	∞	12	6	12	14	12	10	2	10	18	26	34	40
	G	∞	20	12	10	16	18	16	10	0	10	18	26	34
	G	∞	∞	∞	∞	∞	20	22	18	10	2	10	18	26

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

Number of overlaps to try:	$O(d^2)$
Size of each dynamic programming matrix:	$O(n^2)$
Overall:	$O(d^2n^2) = O(N^2)$

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

But dynamic programming is more flexible, allowing mismatches and gaps

Real-world overlappers mix the two, using indexes to filter out vast majority of non-overlapping pairs, then using dynamic programming for remaining pairs

Finding overlaps

Overlapping is typically the slowest part of assembly

> Consider a second-generation sequencing dataset with hundreds of millions or billions of reads!

Approaches from alignment unit can be adapted to finding overlaps

> We saw adaptations of naive exact matching, suffix-tree-assisted exact matching, and dynamic programming
>
> Could also have adapted efficient exact matching, approximate string matching, co-traversal, ...

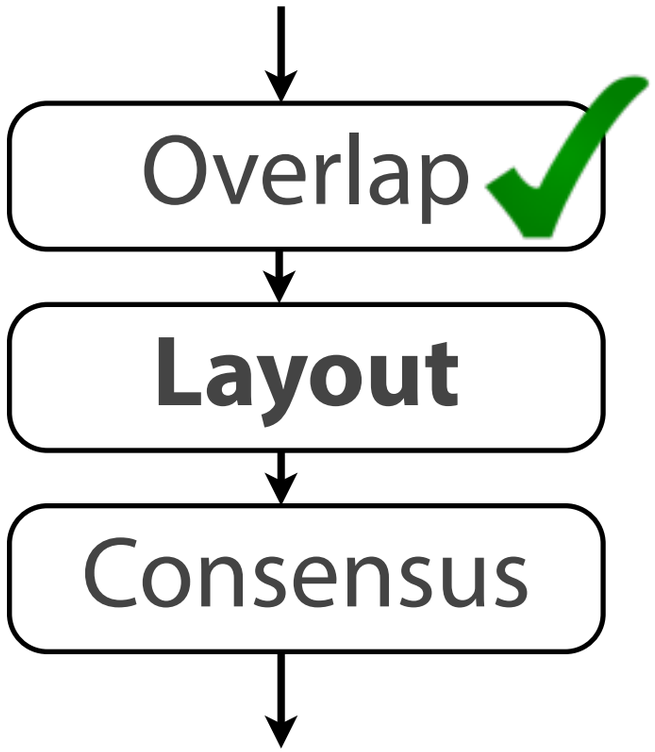
Finding overlaps

Celera Assembler's overlapper is probably the best documented:

- Inverted substring indexes built on batches of reads
- Only look for overlaps between reads that share one or more substrings of some length

http://wgs-assembler.sourceforge.net/wiki/index.php/RunCA#Overlapper

Overlap Layout Consensus



Build overlap graph

Bundle stretches of the overlap graph into *contigs*

Pick most likely nucleotide sequence for each contig

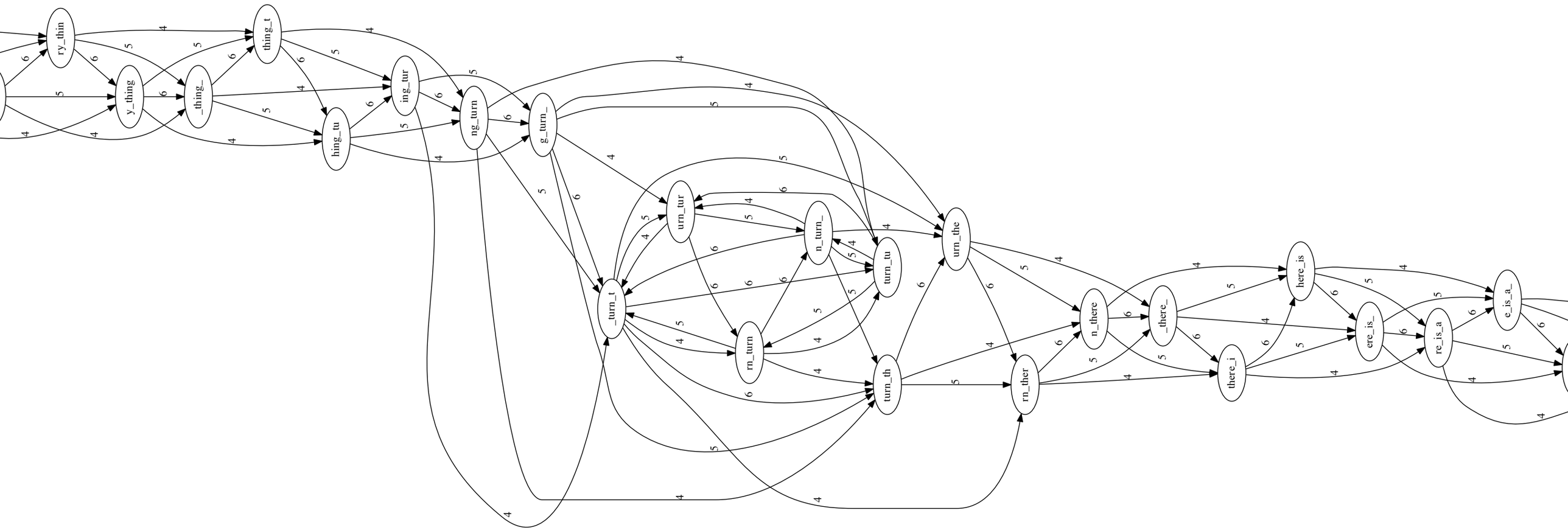
Layout

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`

$l = 4, k = 7$

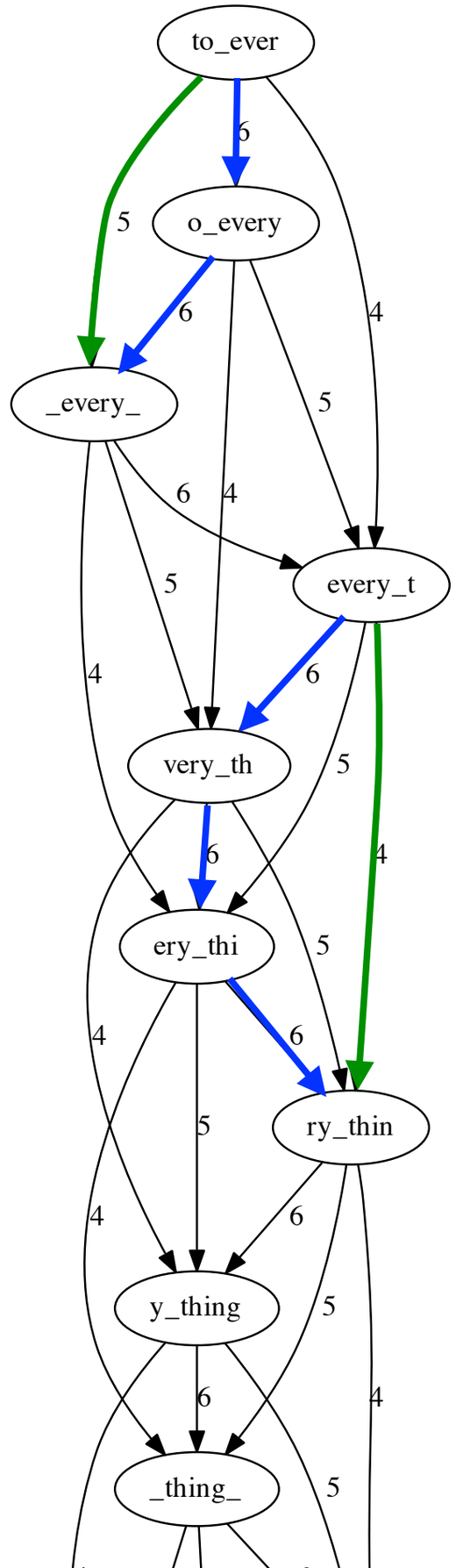


Layout

Anything redundant about this part of the overlap graph?

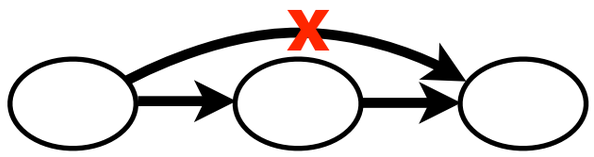
Some edges can be *inferred* (*transitively*) from other edges

E.g. green edge can be inferred from blue



Layout

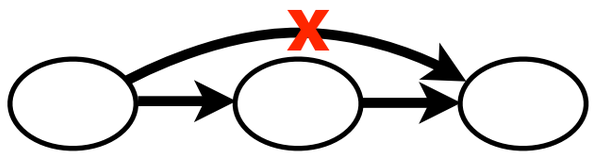
Remove transitively-inferrible edges, starting with edges that skip one node:



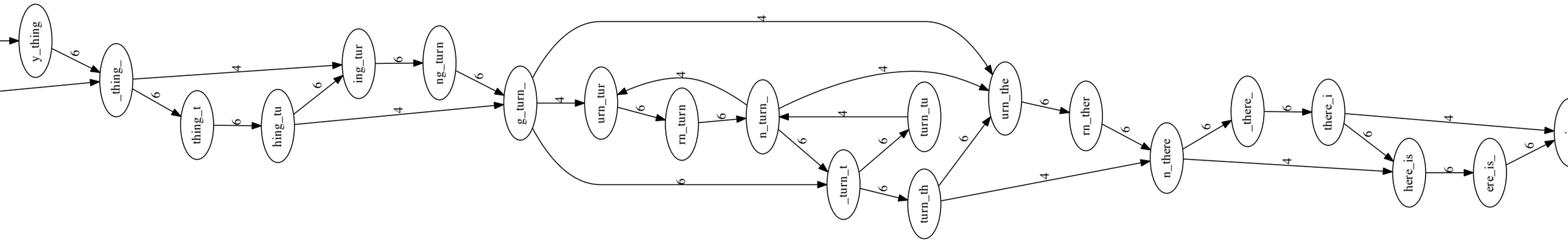
Before:

Layout

Remove transitively-inferrible edges, starting with edges that skip one node:

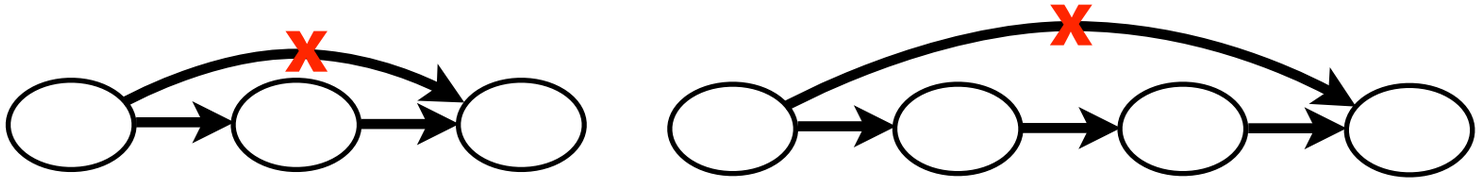


After:



Layout

Remove transitively-inferrible edges, starting with edges that skip one *or two* nodes:

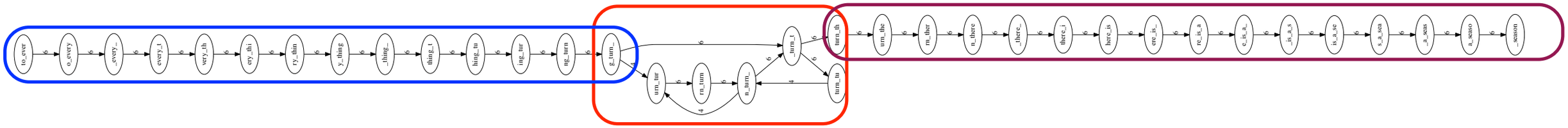


After:

Even simpler

Layout

Emit *contigs* corresponding to the non-branching stretches



Contig 1

`to_every_thing_turn_`

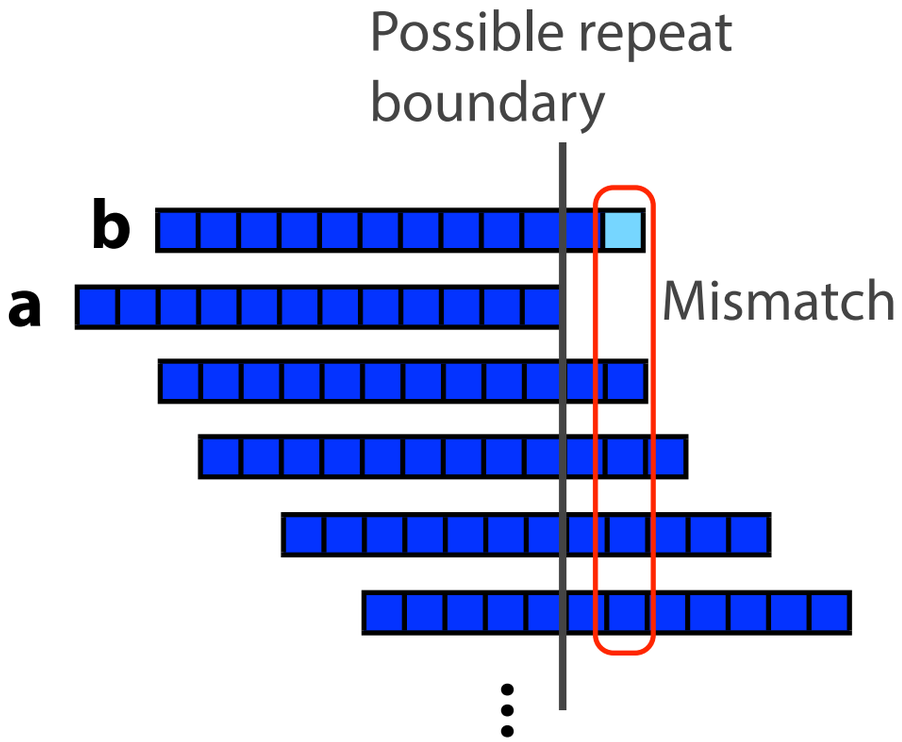
Contig 2

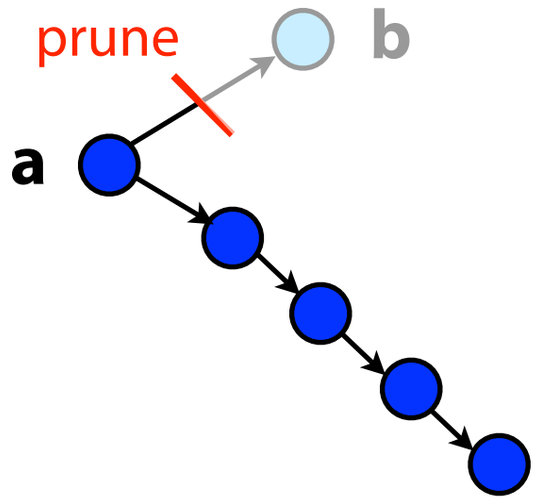
`turn_there_is_a_season`

Unresolvable repeat

Layout

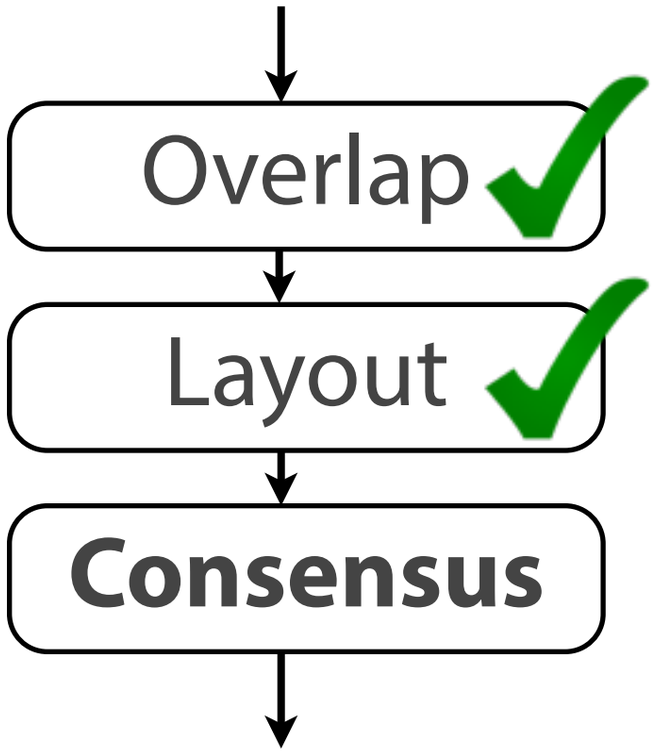
In practice, layout step also has to deal with spurious subgraphs, e.g. because of sequencing error





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

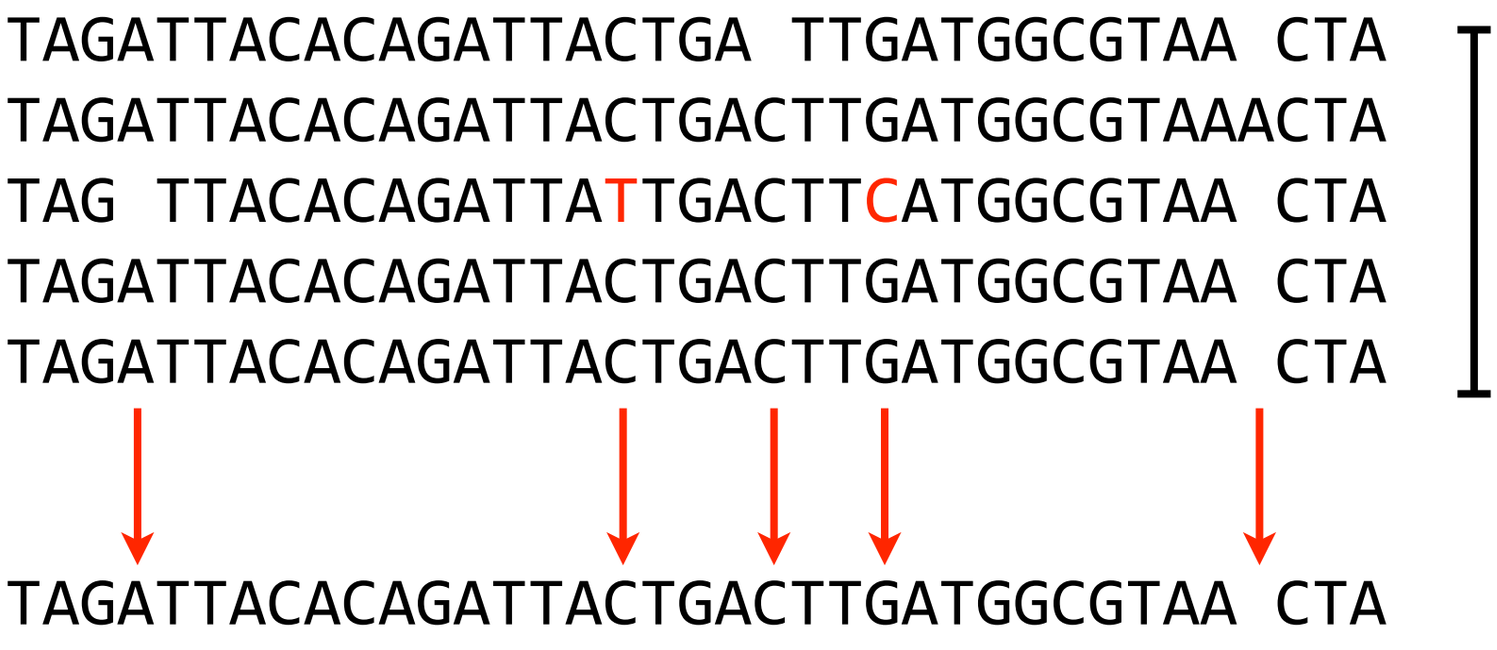


Build overlap graph

Bundle stretches of the overlap graph into *contigs*

Pick most likely nucleotide sequence for each contig

Consensus



Take reads that make up a contig and line them up

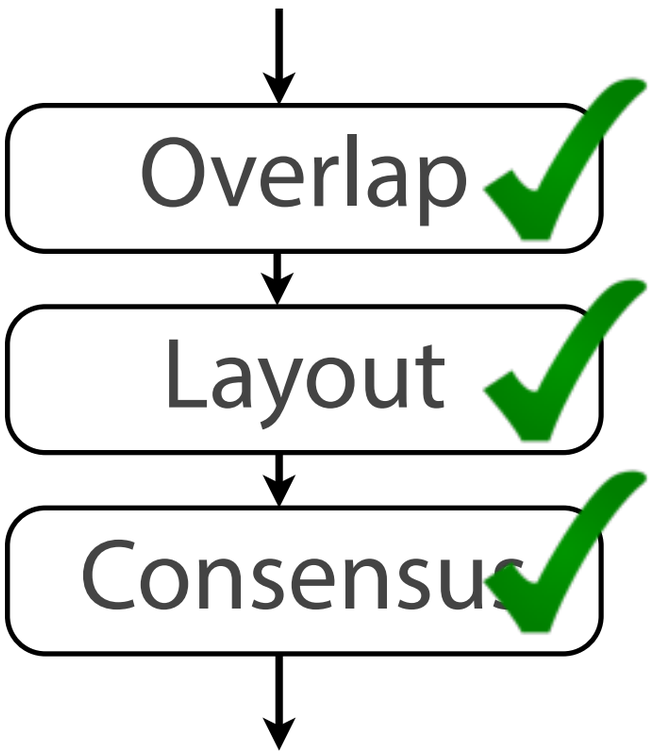
Take *consensus*, i.e. majority vote

At each position, ask: what nucleotide (and/or gap) is here?

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

Say the true genotype is AG, but we have a high sequencing error rate and only about 6 reads covering the position.

Overlap Layout Consensus



Build overlap graph

Bundle stretches of the overlap graph into *contigs*

Pick most likely 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, and in practice # edges grows superlinearly with # reads

2nd-generation sequencing datasets are ~ 100s of millions or billions of reads, hundreds of billions of nucleotides total

Assembly alternatives

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

Alternative 2: de Bruijn graph (DBG) assembly