CS 68: BIOINFORMATICS

Prof. Sara Mathieson
Swarthmore College
Spring 2018

Outline: Jan 24

- Central dogma of molecular biology
- Sequencing pipeline
- Begin: genome assembly

Note: office hours Monday 3-5pm and Wednesday 1-3pm

Outline: Jan 24

- Central dogma of molecular biology
- Sequencing pipeline
- Begin: genome assembly

Note: office hours Monday 3-5pm and Wednesday 1-3pm

Age of universe: 13.8 billion, Age of earth: 4.5 billion, Age of life: 3.8 billion

The Central Dogma of Molecular Biology

Central Dogma of Molecular Biology

- More correctly stated: “*The central dogma states that information in nucleic acid can be perpetuated or transferred but the transfer of information into protein is irreversible.*” (B. Lewin, 2004)

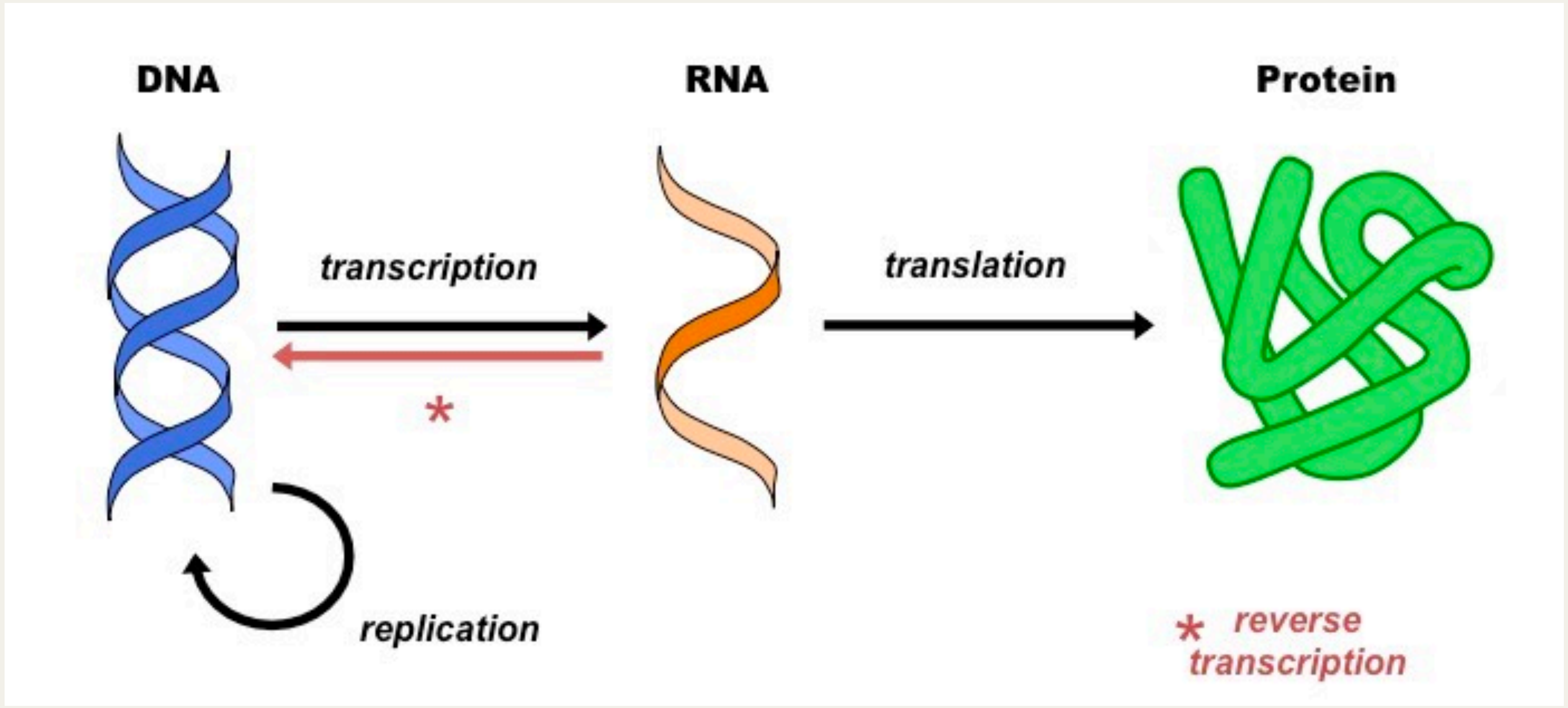


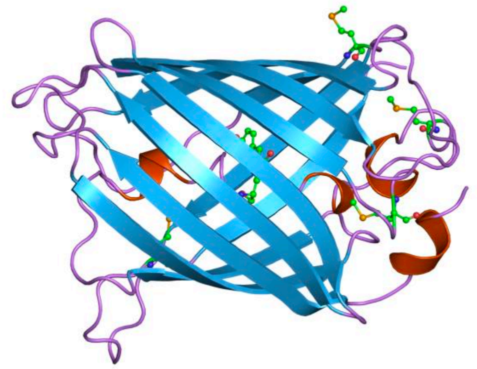
Image: http://ib.bioninja.com.au/

Central Dogma of Molecular Biology

- More correctly stated: "*The central dogma states that information in nucleic acid can be perpetuated or transferred but the transfer of information into protein is irreversible.*" (B. Lewin, 2004)

DNA	ATGCAATCAGATTAG
RNA	CAAUCAGAU
protein	Q S D

GFP protein example



Codons

- Codons are made up of 3 consecutive RNA bases

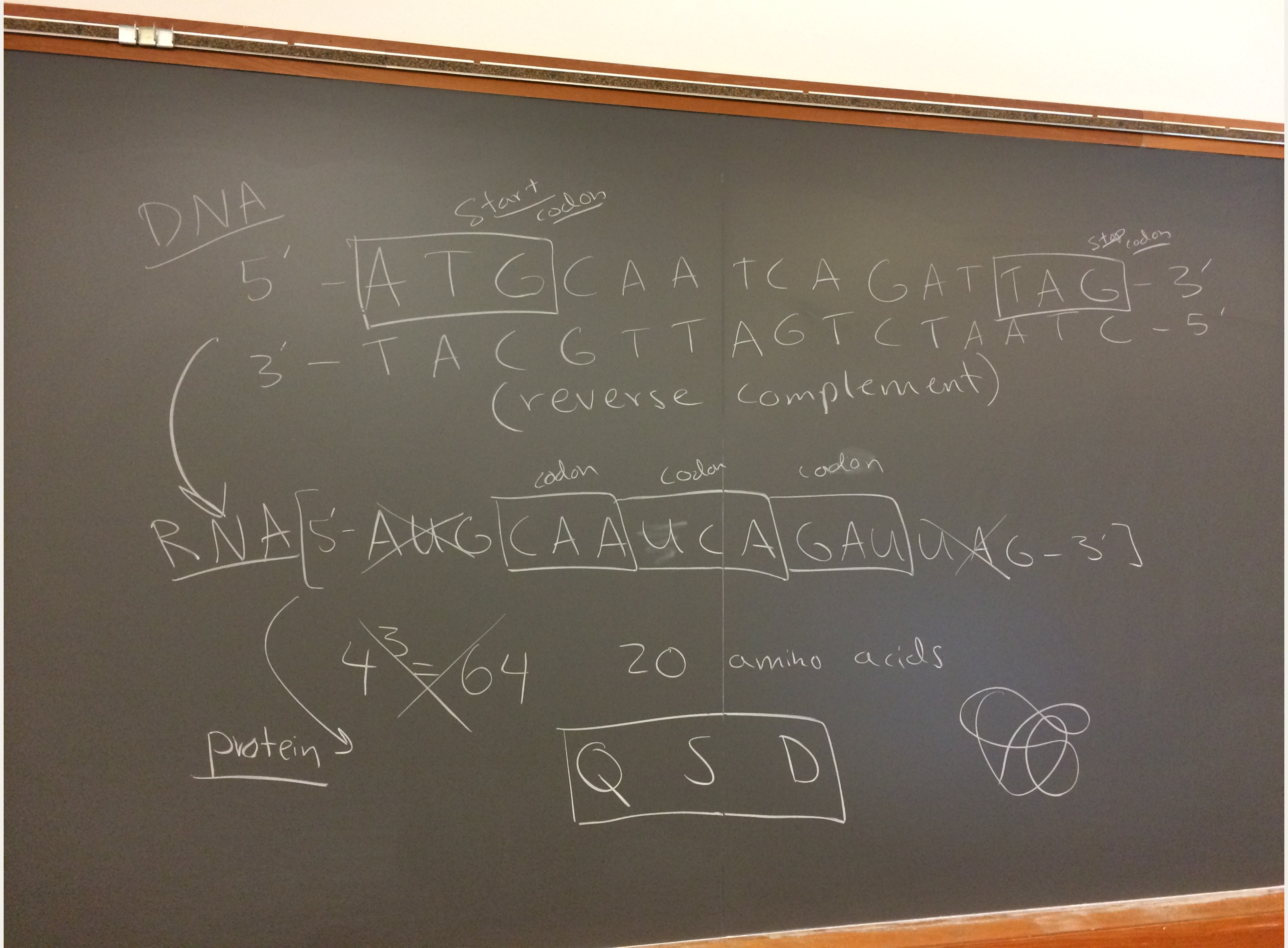
- Transcription begins at the start codon (ATG) and stops at one of the stop codons (TAG, TGA, or TAA)

- Each codon is translated into an amino acid, and amino acids make up proteins

- If each triplet of bases encoded a unique amino acid, how many amino acids would there be?

Codon table

A:	GCU,GCC,GCA,GCG,AGA
R:	CGU,CGC,CGA,CGG,AGG
N:	AAU,AAC
D:	GAU,GAC
C:	UGU,UGC
Q:	CAA,CAG
E:	GAA,GAG
G:	GGU,GGC,GGA,GGG
H:	CAU,CAC
I:	AUU,AUC,AUA
L:	UUA,UUG,CUU,CUC,CUA,CUG
K:	AAA,AAG
M:	AUG
F:	UUU,UUC
P:	CCU,CCC,CCA,CCG
S:	UCU,UCC,UCA,UCG,AGU,AGC
T:	ACU,ACC,ACA,ACG
W:	UGG
Y:	UAU,UAC
V:	GUU,GUC,GUA,GUG



DNA

Start codon

STOP codon

$5' - \boxed{ATG}\ CAA\ TCA\ GAT\ \boxed{TAG} - 3'$

$3' - TAC\ GTT\ AGT\ CTA\ ATC - 5'$

(reverse complement)

codon codon codon

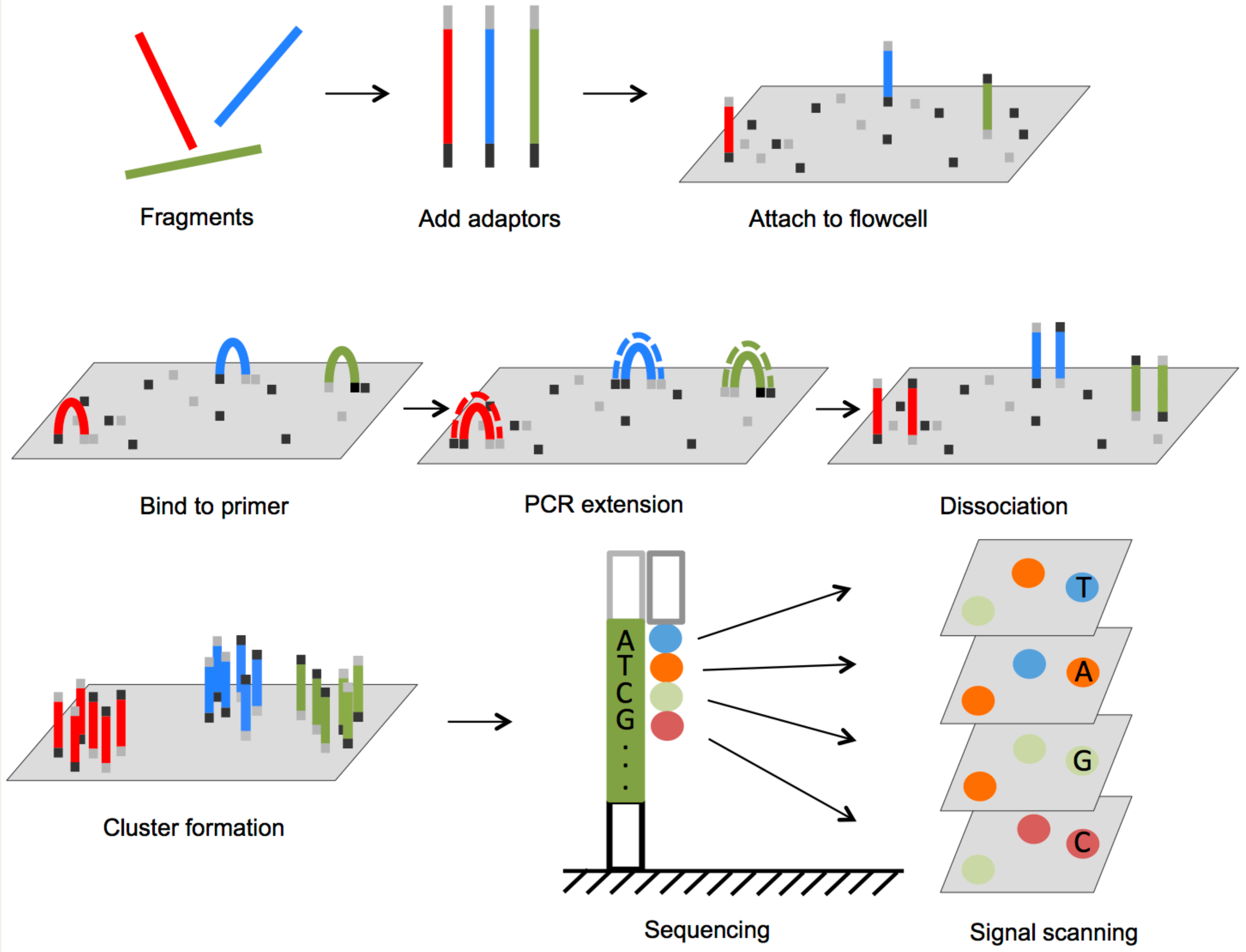
RNA $[5' - \cancel{AU}G\ \boxed{CAA}\ \boxed{UCA}\ \boxed{GAU}\ \cancel{UA}G - 3']$

$\cancel{4^3 = 64}$ 20 amino acids

protein

$\boxed{Q\ S\ D}$

Sequencing Pipeline



InTech open science

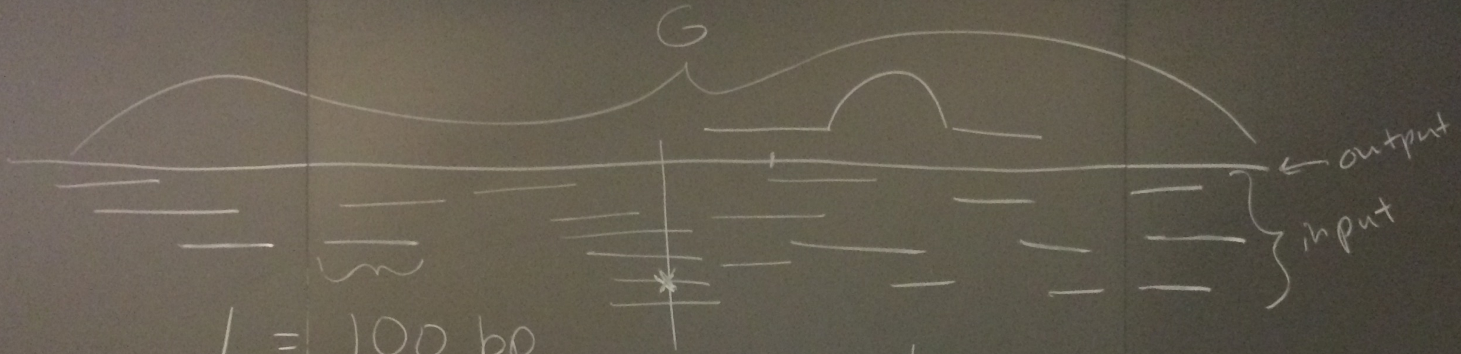
Genome Assembly

Goal of genome assembly

- Input: millions of “reads” from next-generation sequencing
- Output: (ideally) entire “consensus” sequence of the original DNA

Assembly vocabulary

- **Long read**: a fragment that has been "read" from a genomic sequence (DNA for us), usually > 1000 bp
- **Short read**: same as a long read but < 1000 bp (usually 50-100 bp)
- **Paired-end read**: both ends of a fragment are "read", but the portion between them is unknown
- **bp**: base pair
- **kb, Mb, Gb**: kilo bases 10^3, mega bases 10^6, giga bases 10^9
- **Coverage**: number of times (on average) any given base is sequenced. Total number of bases in all reads (n reads × L bases/read), divided by the length of the genome G.



$\underline{L} = 100$ bp

$$C = \frac{n \cdot L}{G}$$

$\uparrow$

coverage (want high coverage)

$$C = \frac{6 \cdot 10}{20}$$

$$\boxed{C = 3}$$

Could you design an algorithm for genome assembly?

1) With a partner, analyze these given reads. What is *L* (length of each read)? What is *n* (number of reads)?

2) Try to assemble these given reads into one continuous string. For these small numbers we can often do this “by eye”, but what if *n* = millions and *L* = 100? How would you tell a computer to assemble them?

3) What is *G* (length of the resulting genome)? From all the numbers, compute the coverage.

Overlap graph assembly

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
 GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
  GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
    GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
     GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
      GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
       GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
        GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
         GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
          GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
               GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
            GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
                 GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
              GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
                  GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
                GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Overlap graph assembly

```
read 1:    GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:    AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
                GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

overlap = 17

Overlap graph assembly

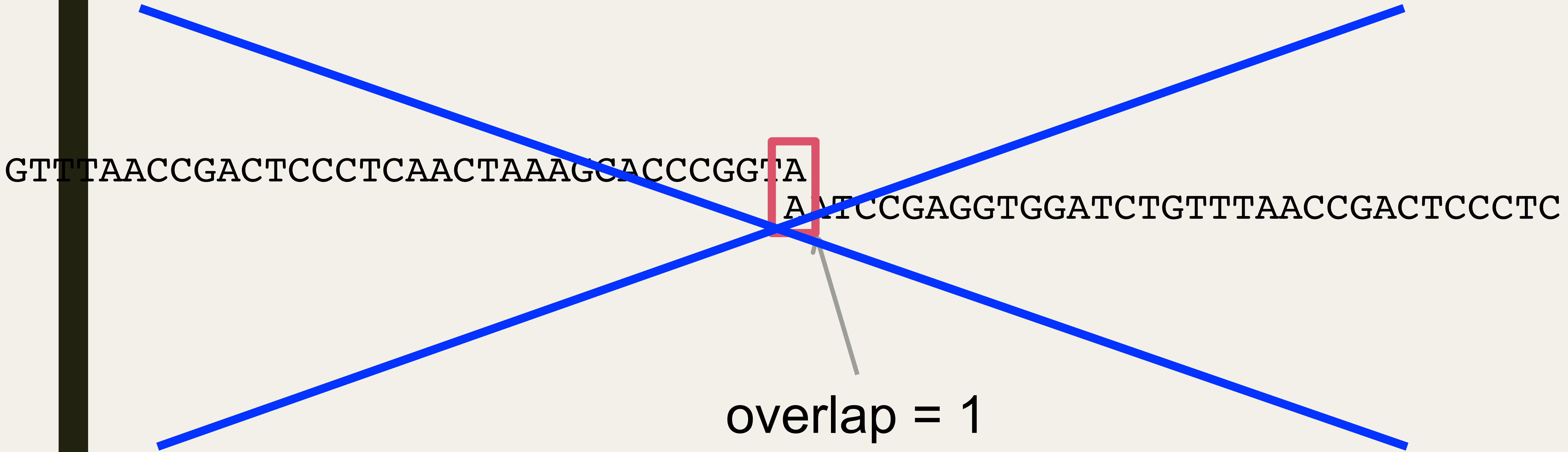
```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

```
GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
                                 AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

overlap = 1

Overlap graph assembly

```
read 1:   GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
read 2:   AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```



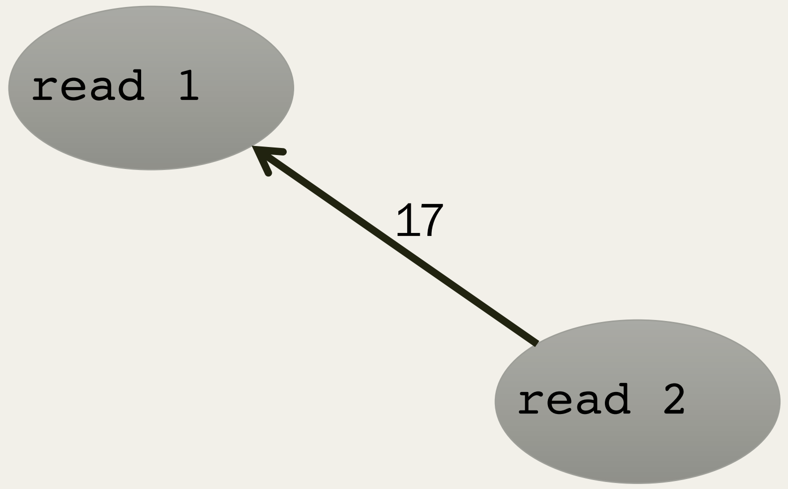
```
GTTTAACCGACTCCCTCAACTAAAGCACCCGGTA
                                 AATCCGAGGTGGATCTGTTTAACCGACTCCCTC
```

Takeaways

- Overlaps should meet some minimum threshold *T* (often 1/3 of the read length)

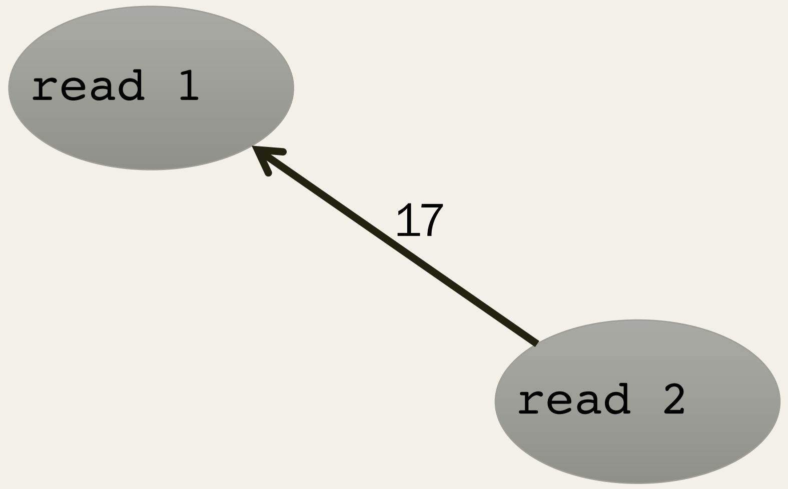
- Overlaps should have a maximum number of errors allows (roughly 2-3 depending on the error rate and overlap threshold)

Overlap graph (directed, why?)



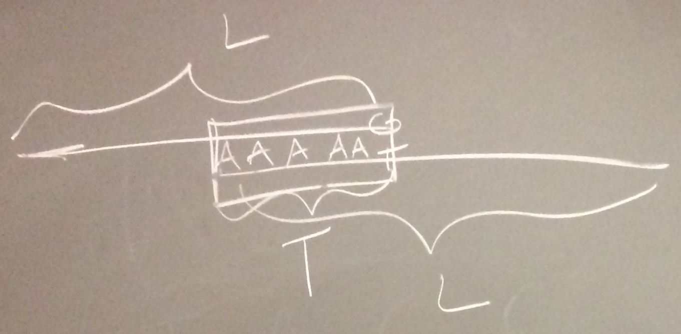
What is the runtime for creating the overlap graph?

Overlap graph (directed, why?)



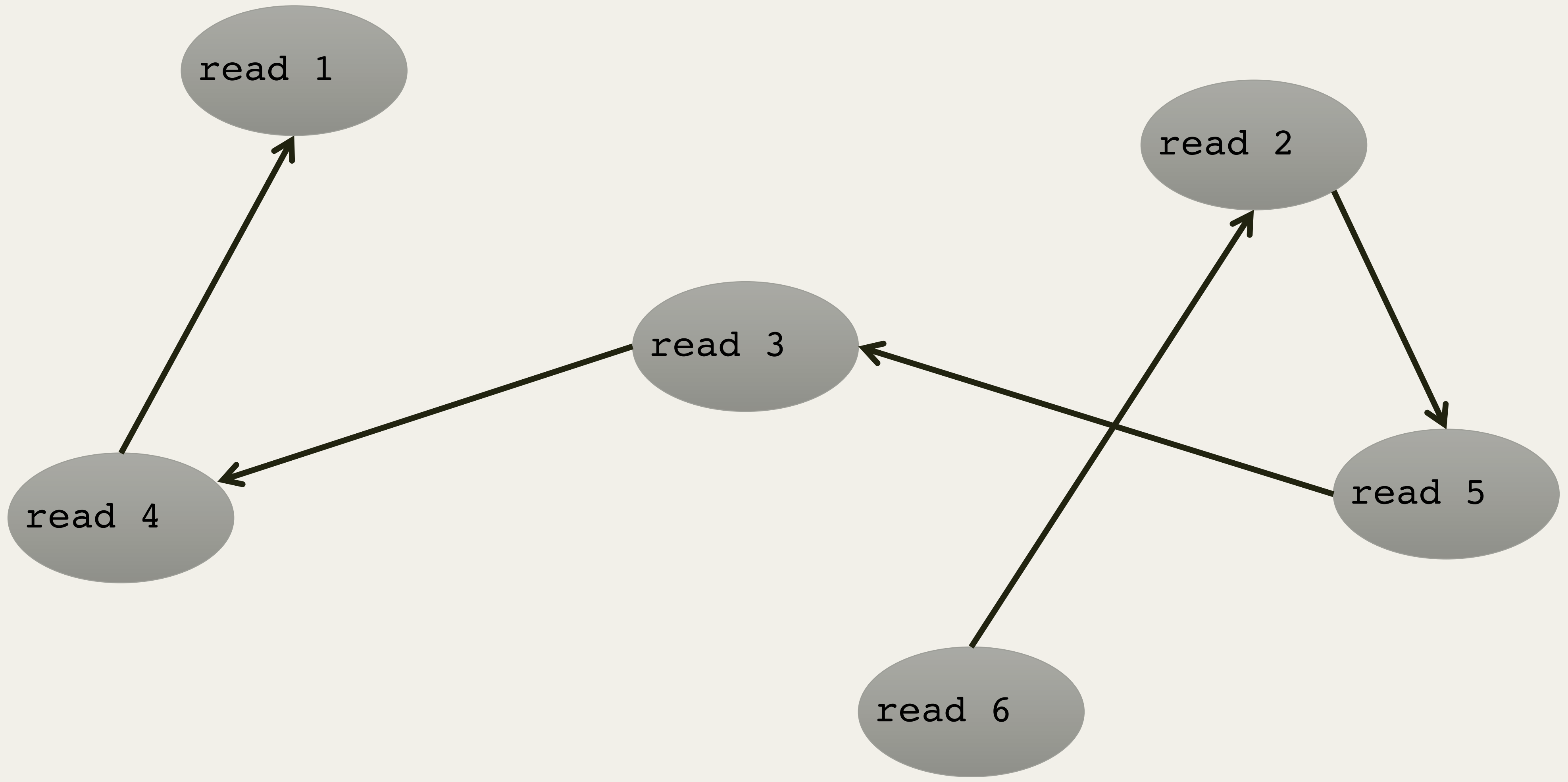
What is the runtime for creating the overlap graph?

O(n^2) pairs, O(L^2) for each pair, => really slow

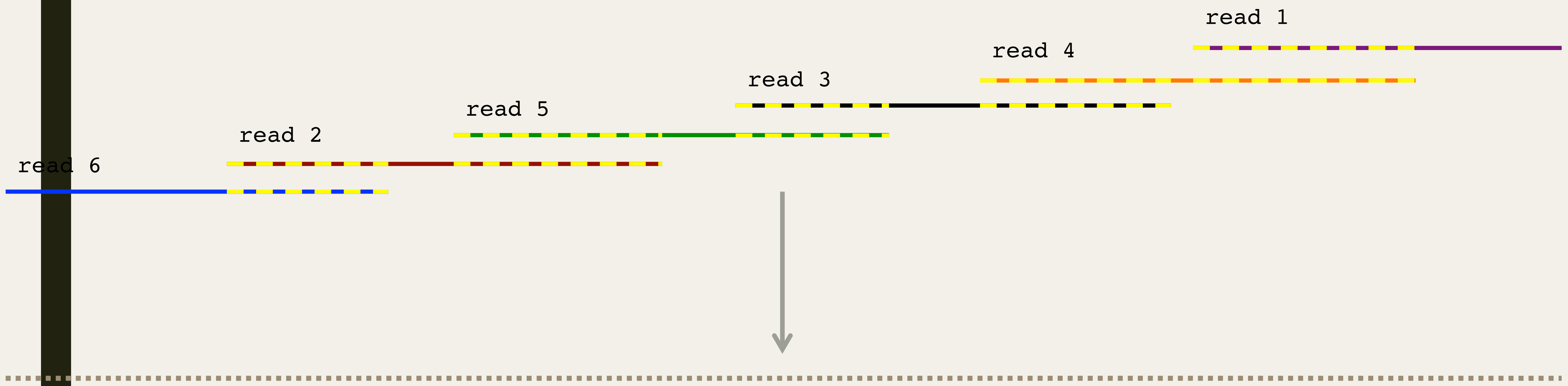


$$\begin{tikzcd}
 & \binom{n}{2}\ \text{pairs} \arrow[r] & O(n^2) \arrow[d] \\
\left.\begin{array}{c} O(L) \\ O(L) \\ \vdots \\ O(L) \end{array}\right\} & O(L^2) \arrow[r] & \boxed{O(L^2 n^2)}
\end{tikzcd}$$

Overlap graph

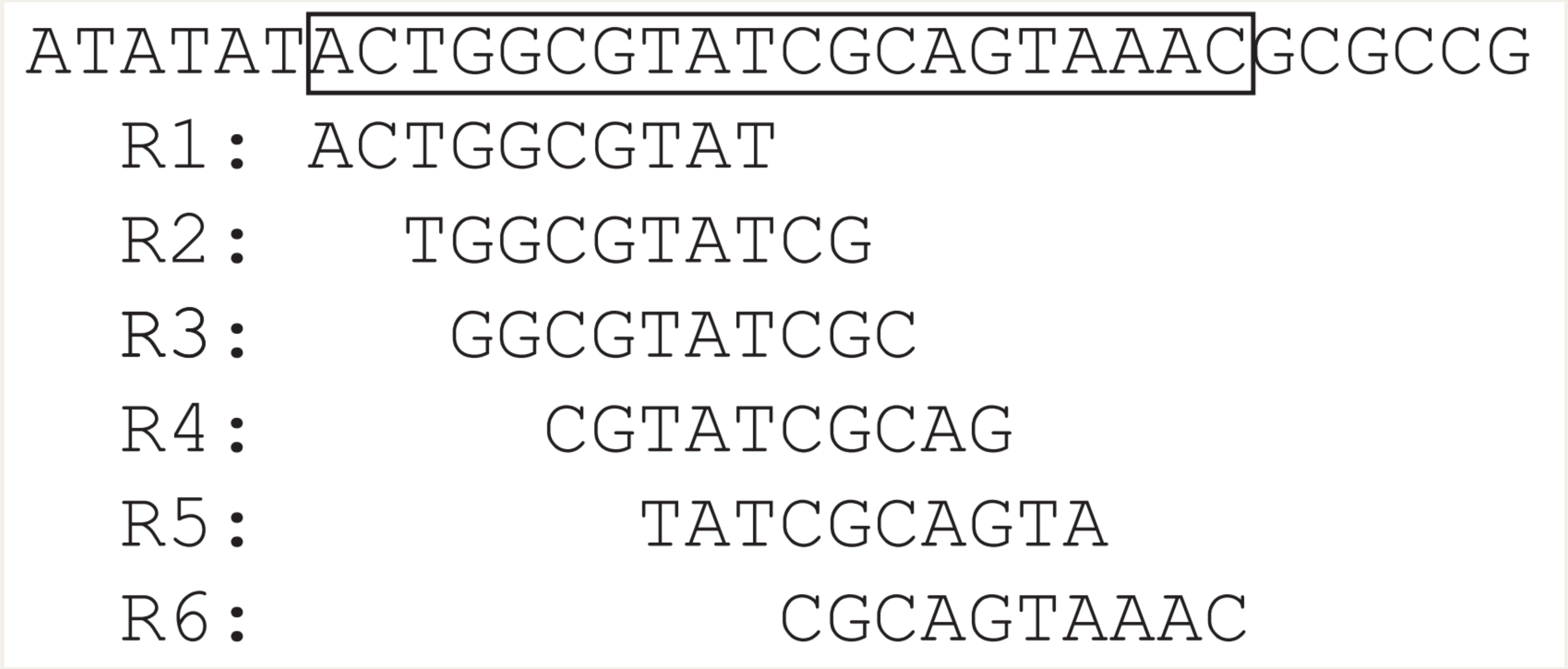


Perfect graph traversal



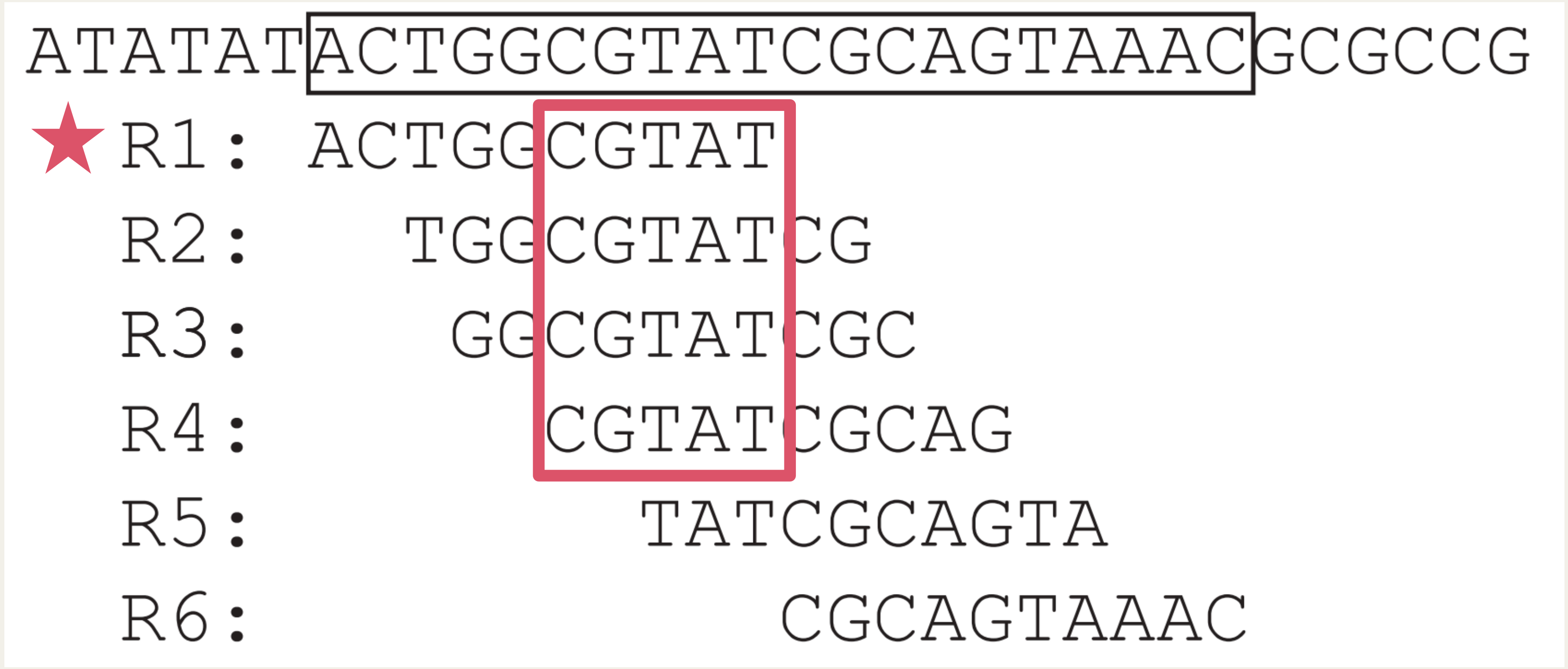
Final sequence

Activity example: L = 10, T = 5

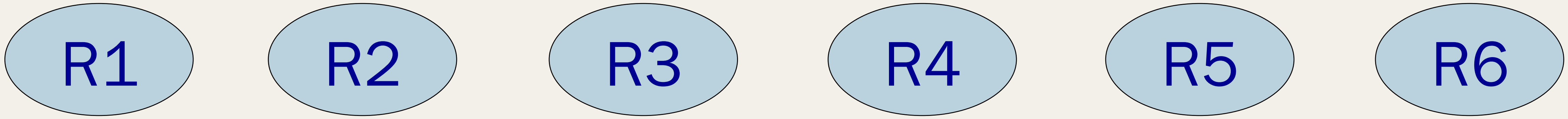


```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
  R1: ACTGGCGTAT
  R2:   TGGCGTATCG
  R3:    GGCGTATCGC
  R4:      CGTATCGCAG
  R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```

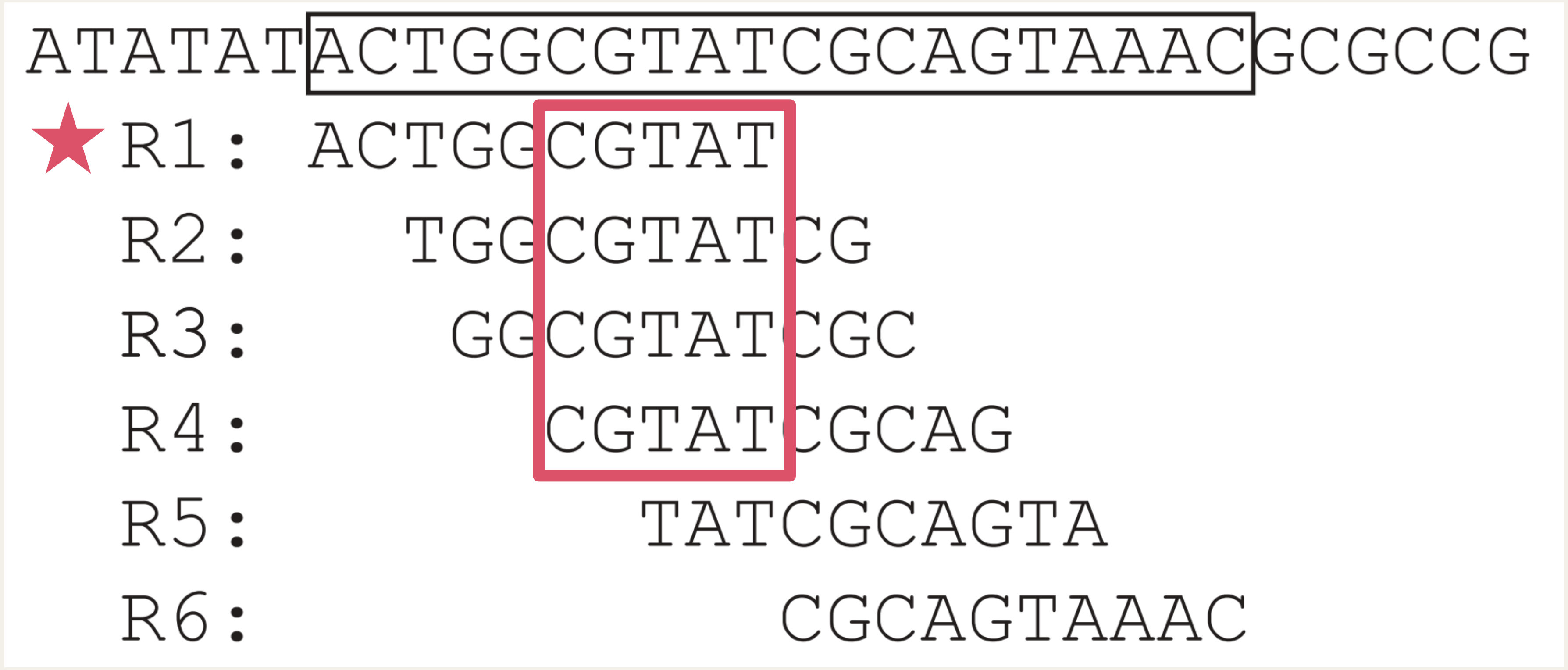
Activity example: L = 10, T = 5



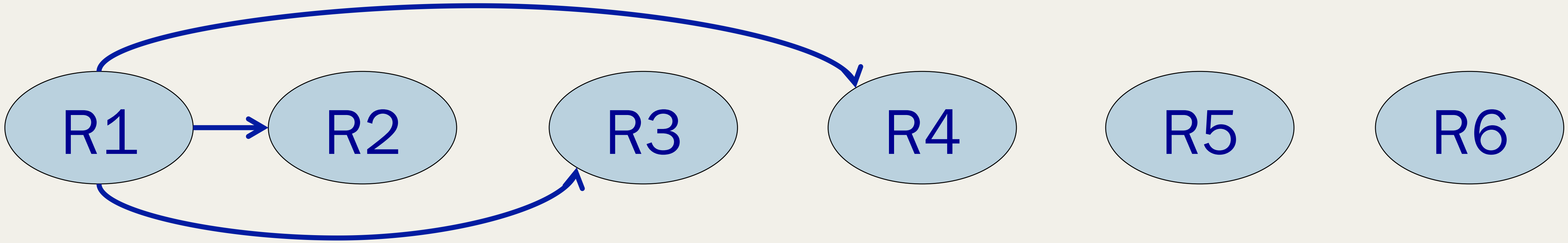
```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
★ R1: ACTGGCGTAT
  R2:   TGGCGTATCG
  R3:    GGCGTATCGC
  R4:      CGTATCGCAG
  R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```



Activity example: *L* = 10, *T* = 5

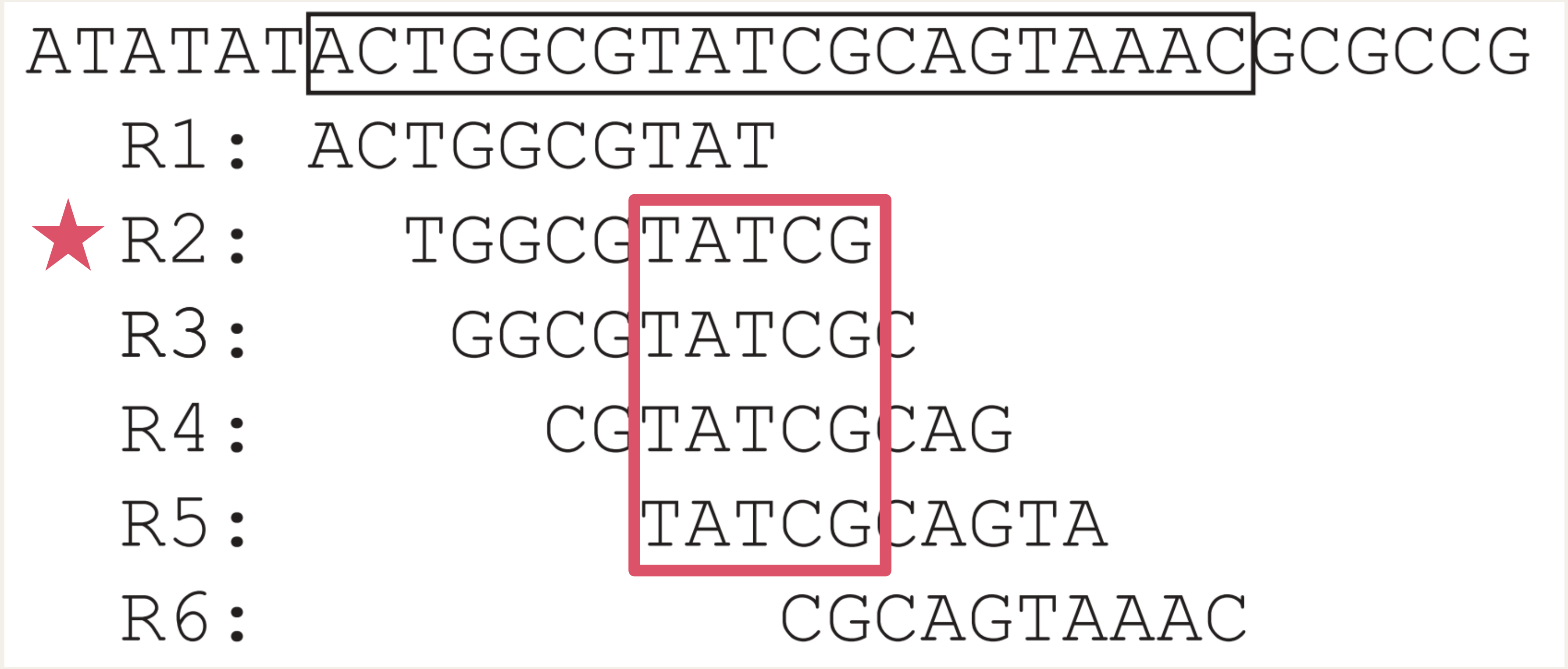


```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
★ R1: ACTGGCGTAT
  R2:   TGGCGTATCG
  R3:    GGCGTATCGC
  R4:      CGTATCGCAG
  R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```



$$\begin{tikzcd}
R1 \arrow[r] \arrow[rr, bend right] \arrow[rrr, bend left] & R2 & R3 & R4 & R5 & R6
\end{tikzcd}$$

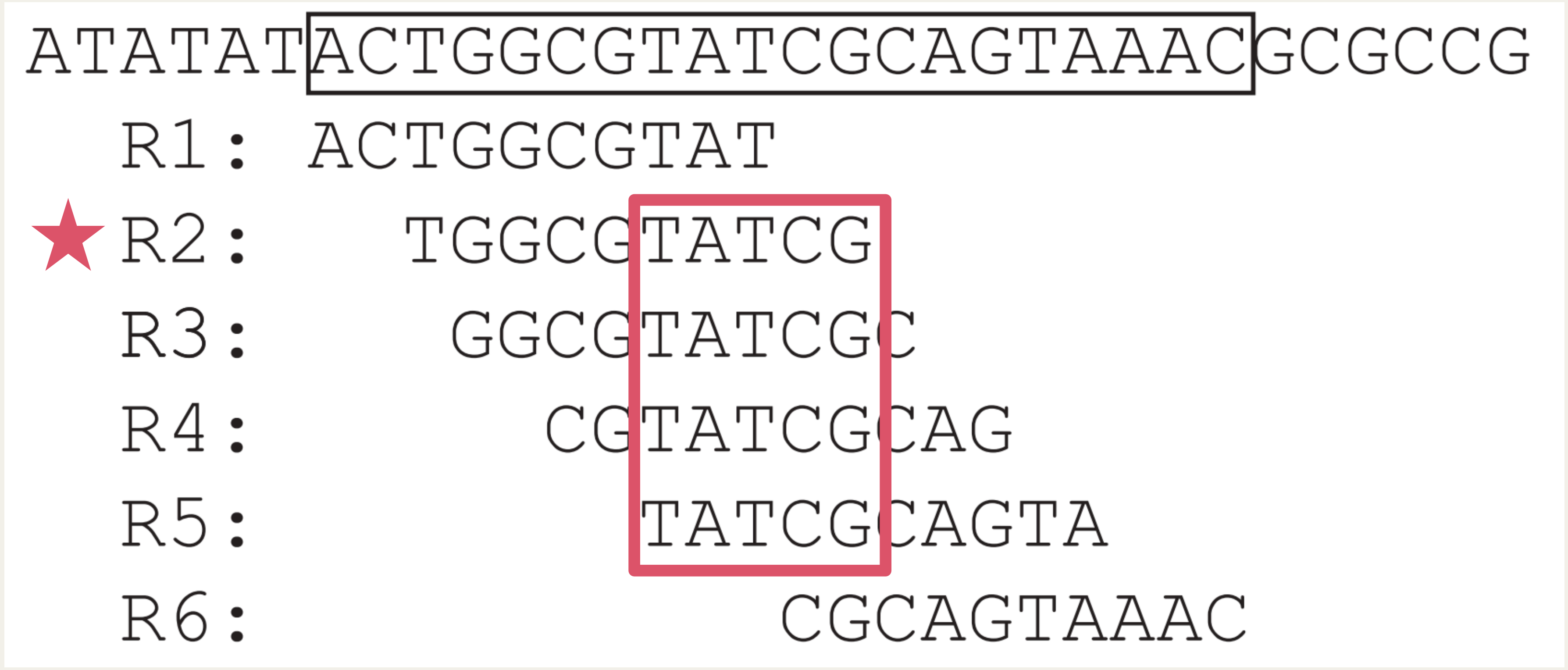
Activity example: L = 10, T = 5



```
      ATATATACTGGCGTATCGCAGTAAACGCGCCG
   R1:      ACTGGCGTAT
★ R2:        TGGCGTATCG
   R3:         GGCGTATCGC
   R4:           CGTATCGCAG
   R5:             TATCGCAGTA
   R6:                CGCAGTAAAC
```

$$\begin{tikzcd}
R1 \arrow[r] \arrow[rr, bend right] \arrow[rrr, bend left] & R2 & R3 & R4 & R5 & R6
\end{tikzcd}$$

Activity example: L = 10, T = 5



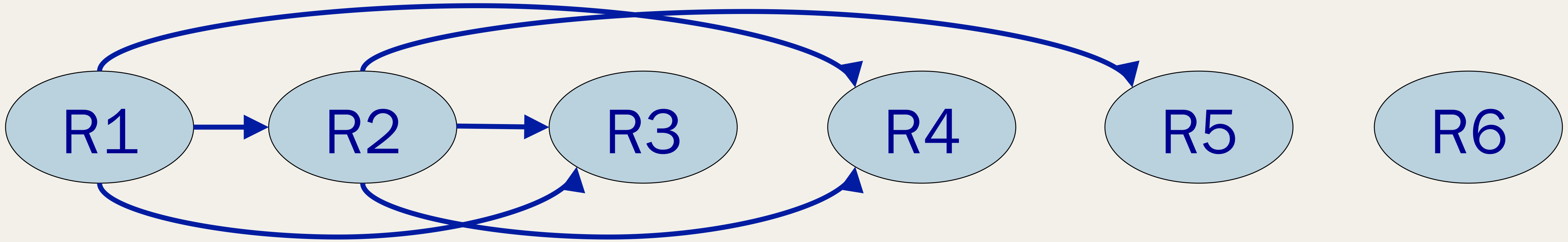
```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
  R1: ACTGGCGTAT
★ R2:   TGGCGTATCG
  R3:    GGCGTATCGC
  R4:      CGTATCGCAG
  R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```

$$\begin{tikzcd}
R1 \arrow[r] \arrow[rr, bend right] \arrow[rrr, bend left] & R2 \arrow[r] \arrow[rr, bend right] \arrow[rrr, bend left] & R3 & R4 & R5 & R6
\end{tikzcd}$$

Li et al, *Briefings in Functional Genomics* (2012)

Activity example: *L* = 10, *T* = 5

```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
  R1: ACTGGCGTAT
  R2:   TGGCGTATCG
★ R3:    GGCGTATCGC
  R4:      CGTATCGCAG
  R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```



Activity example: $L = 10$, $T = 5$

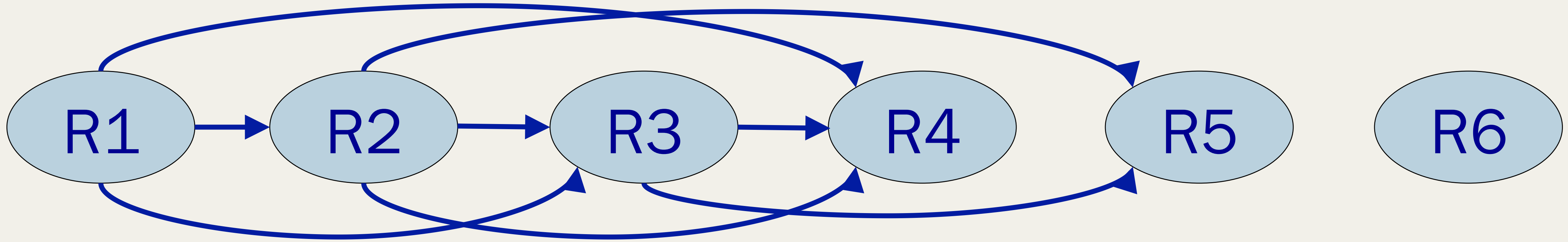
```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
  R1: ACTGGCGTAT
  R2:   TGGCGTATCG
★ R3:    GGCGTATCGC
  R4:      CGTATCGCAG
  R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```

$$\begin{tikzcd}
R1 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R2 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R3 \arrow[r] \arrow[rr, bend right] & R4 & R5 & R6
\end{tikzcd}$$

Li et al, *Briefings in Functional Genomics* (2012)

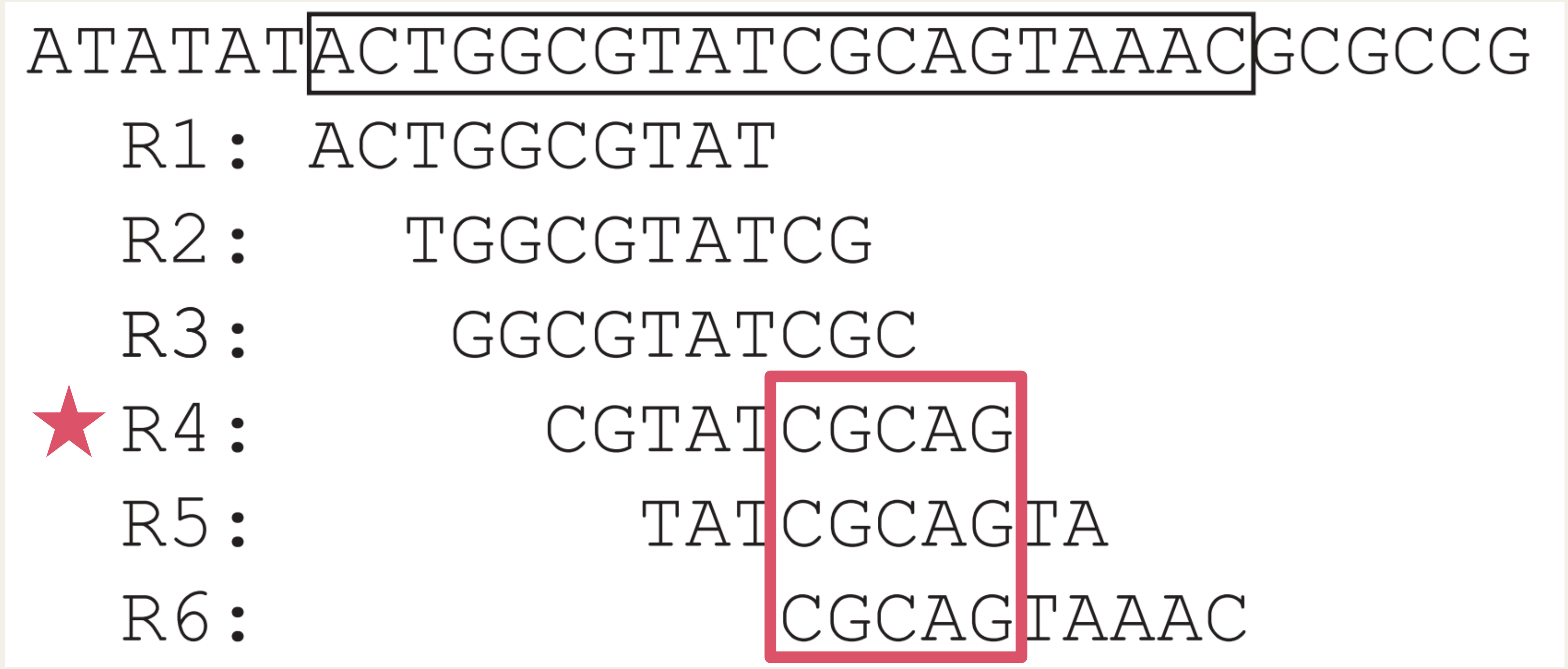
Activity example: $L = 10$, $T = 5$

```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
  R1: ACTGGCGTAT
  R2:   TGGCGTATCG
  R3:    GGCGTATCGC
★ R4:      CGTATCGCAG
  R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```



$$\begin{tikzcd} R1 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R2 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R3 \arrow[r] \arrow[rr, bend right] & R4 & R5 & R6 \end{tikzcd}$$

Activity example: L = 10, T = 5



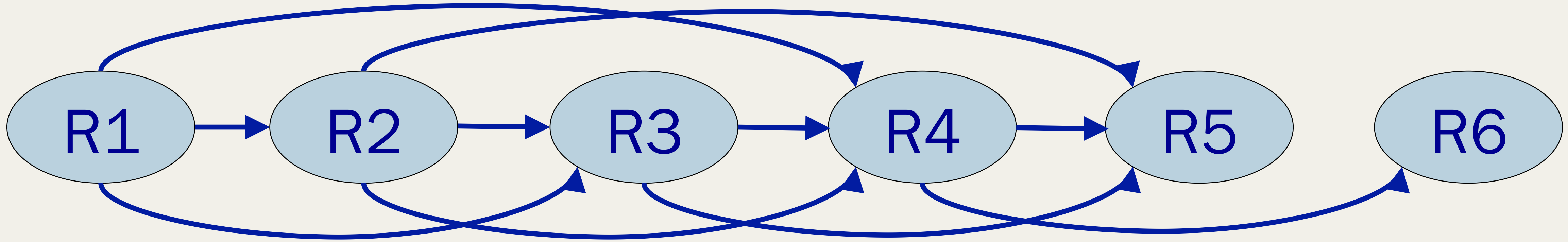
```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
  R1: ACTGGCGTAT
  R2:   TGGCGTATCG
  R3:    GGCGTATCGC
★ R4:      CGTATCGCAG
  R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```

$$\begin{tikzcd}
R1 \arrow[r] \arrow[rr, bend right] \arrow[rrr, bend left] & R2 \arrow[r] \arrow[rr, bend right] \arrow[rrr, bend left] & R3 \arrow[r] \arrow[rr, bend right] & R4 \arrow[r] \arrow[rr, bend right] & R5 & R6
\end{tikzcd}$$

Li et al, *Briefings in Functional Genomics* (2012)

Activity example: L = 10, T = 5

```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
  R1: ACTGGCGTAT
  R2:   TGGCGTATCG
  R3:    GGCGTATCGC
  R4:      CGTATCGCAG
★ R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```



$$\begin{tikzcd}
R1 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R2 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R3 \arrow[r] \arrow[rr, bend right] & R4 \arrow[r] \arrow[rr, bend right] & R5 & R6
\end{tikzcd}$$

Activity example: L = 10, T = 5

```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
  R1: ACTGGCGTAT
  R2:   TGGCGTATCG
  R3:    GGCGTATCGC
  R4:      CGTATCGCAG
★ R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```

$$\begin{tikzcd}
R1 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R2 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R3 \arrow[r] \arrow[rr, bend right] & R4 \arrow[r] \arrow[rr, bend right] & R5 \arrow[r] & R6
\end{tikzcd}$$

Li et al, *Briefings in Functional Genomics* (2012)

Activity example: $L = 10$, $T = 5$

```
ATATATACTGGCGTATCGCAGTAAACGCGCCG
  R1: ACTGGCGTAT
  R2:   TGGCGTATCG
  R3:    GGCGTATCGC
  R4:      CGTATCGCAG
  R5:        TATCGCAGTA
  R6:           CGCAGTAAAC
```

$$\begin{tikzcd}
R1 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R2 \arrow[r] \arrow[rrr, bend left] \arrow[rr, bend right] & R3 \arrow[r] \arrow[rr, bend right] & R4 \arrow[r] \arrow[rr, bend right] & R5 \arrow[r] & R6
\end{tikzcd}$$