

# **Сборка. Часть вторая**

**Алгоритмы в биоинформатике**

**Дмитрий Мелешко**

**meleshko.dmitrii@gmail.com**

# В прошлой лекции

- Секвенирование – случайный процесс
- Сборка геномов из коротких и точных ридов

# В этой лекции

- Задача сборки генома
- Задача SCS
- Жадное решение и overlap graph

# Сборка геномов

Дано:

Множество ридов

Цель:

Найти геном, из которого эти риды получены

CTAGGCCCTCAATT  
CTCTAGGCCCTCAATT  
GGCTCTAGGCCCTCATT  
CTCGGCTCTAGCCCCTCATT  
TATCTCGACTCTAGGCCCTCA  
TATCTCGACTCTAGGCC  
TCTATATCTGGCTCTAGG  
GGCGTCTATATCTCG  
GGCGTCGATATCT  
GGCGTCTATATCT  
GGCGTCTATATCTGGCTCTAGGCCCTCATT

# Сборка геномов

Дано:

Множество ридов

Цель:

Найти геном, из которого эти риды получены

CTAGGCCCTCAATT  
GGCGTCTATATCT  
CTCTAGGCCCTCAATT  
TCTATATCTGGCTCTAGG  
GGCTCTAGGCCCTCATT  
CTCGGCTCTAGCCCCTCATT  
TATCTCGACTCTAGGCCCTCA  
GGCGTCGATATCT  
TATCTCGACTCTAGGCC  
GGCGTCTATATCTCG

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

# Сборка геномов. SCS

Задача поиска SCS (shortest common supersequence) – для заданного множества подстрок, найти такую строку минимальной длины, которая содержит все подстроки.

AAA, AAT, ATT, TTA, TTT

CS = AAAAATATT~~TT~~ATT

Но это не самая короткая строка

# Сборка геномов. SCS

Задача поиска SCS (shortest common supersequence) – для заданного множества подстрок, найти такую строку минимальной длины, которая содержит все подстроки.

AAA, AAT, ATT, TTA, TTT

CS = AAAAATATTATTATT

Но это не самая короткая строка

CSC = AAATTAA

# SCS

SCS (shortest common supersequence) – очень трудная задача.  
NP-трудная!

Как решить?

# SCS

AAA, AAT, ATT, TTA, TTT

Рассмотрим разные порядки

(1, 2, 3, 4, 5) AAA<sub>1</sub>T<sub>2</sub>A<sub>3</sub>T<sub>4</sub>TT<sub>5</sub>

(1, 5, 2, 4, 3) AAAT<sub>1</sub>TT<sub>2</sub>A<sub>3</sub>TT<sub>4</sub>ATT<sub>5</sub>

# SCS

AAA, AAT, ATT, TTA, TTT

Рассмотрим разные порядки

(1, 2, 3, 4, 5) AAA<sub>1</sub>T<sub>2</sub>A<sub>3</sub>T<sub>4</sub>TT<sub>5</sub>

(1, 5, 2, 4, 3) AAAT<sub>1</sub>TT<sub>2</sub>A<sub>3</sub>TT<sub>4</sub>ATT<sub>5</sub>

(1, 2, 3, 5, 4) AAAT<sub>1</sub>T<sub>2</sub>TT<sub>3</sub>A<sub>5</sub>TT<sub>4</sub>

# SCS

AAA, AAT, ATT, TTA, TTT

Рассмотрим разные порядки

(1, 2, 3, 4, 5) AAAT<sub>T</sub>ATT

(1, 5, 2, 4, 3) AAATT<sub>T</sub>AATTATT

(1, 2, 3, 5, 4) AAAT<sub>T</sub>TA

# SCS

AAA, AAT, ATT, TTA, TTT

Рассмотрим разные порядки

(1, 2, 3, 4, 5) AAAT<sub>T</sub>ATT

(1, 5, 2, 4, 3) AAATT<sub>T</sub>AATTATT

(1, 2, 3, 5, 4) AAAT<sub>T</sub>TA

# SCS

## Алгоритм:

- Перебираем все возможные порядки
- Для каждого порядка объединяем склеиваем строки, начиная с первого не перекрывающегося символа в строке

# SCS

## Алгоритм:

- Перебираем все возможные порядки
- Для каждого порядка объединяем склеиваем строки, начиная с первого не перекрывающегося символа в строке

## Замечания:

Сложность такого алгоритма  $O(n!nl^2)$

Если подстроки разной длины то нужно еще проверять, есть ли уже подстрока в строке, иначе контрпример {АТА, Т}

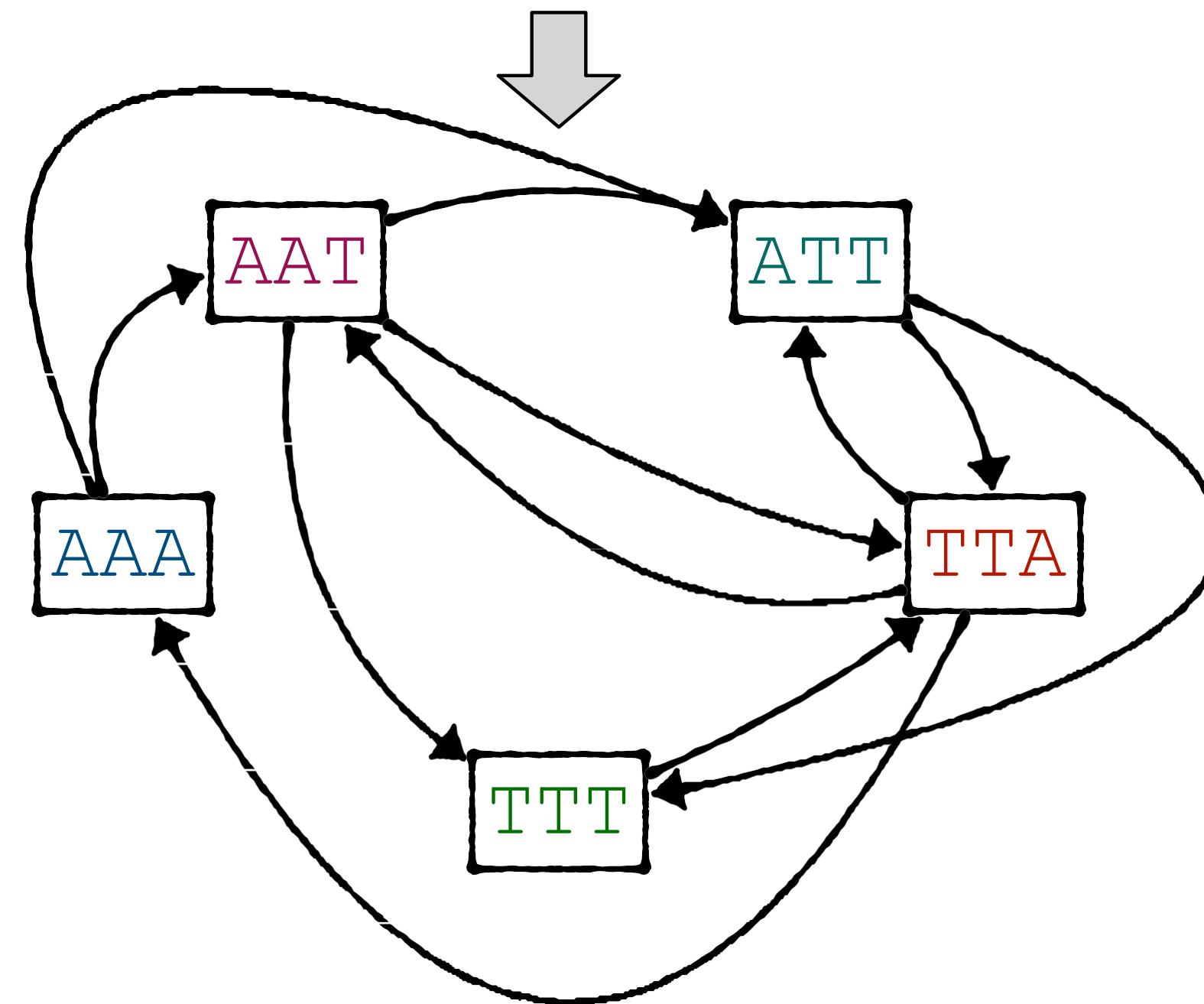
# SCS. Граф перекрытий

*Определение:* для заданного множества строк  $\{S\}$  и порога  $t$ , графом перекрытий  $O(\{S\}, t)$  будем называть такой направленный граф, в котором

- Вершины соответствуют строкам  $S_i$
- Ребро проводится из вершины  $i$  в вершину  $j$  если самый длинный суффикс  $S_i$  совпадающий с префиксом  $S_j$  длиннее чем  $t$
- Вес ребра  $e : (i, j)$  – размер суффикса

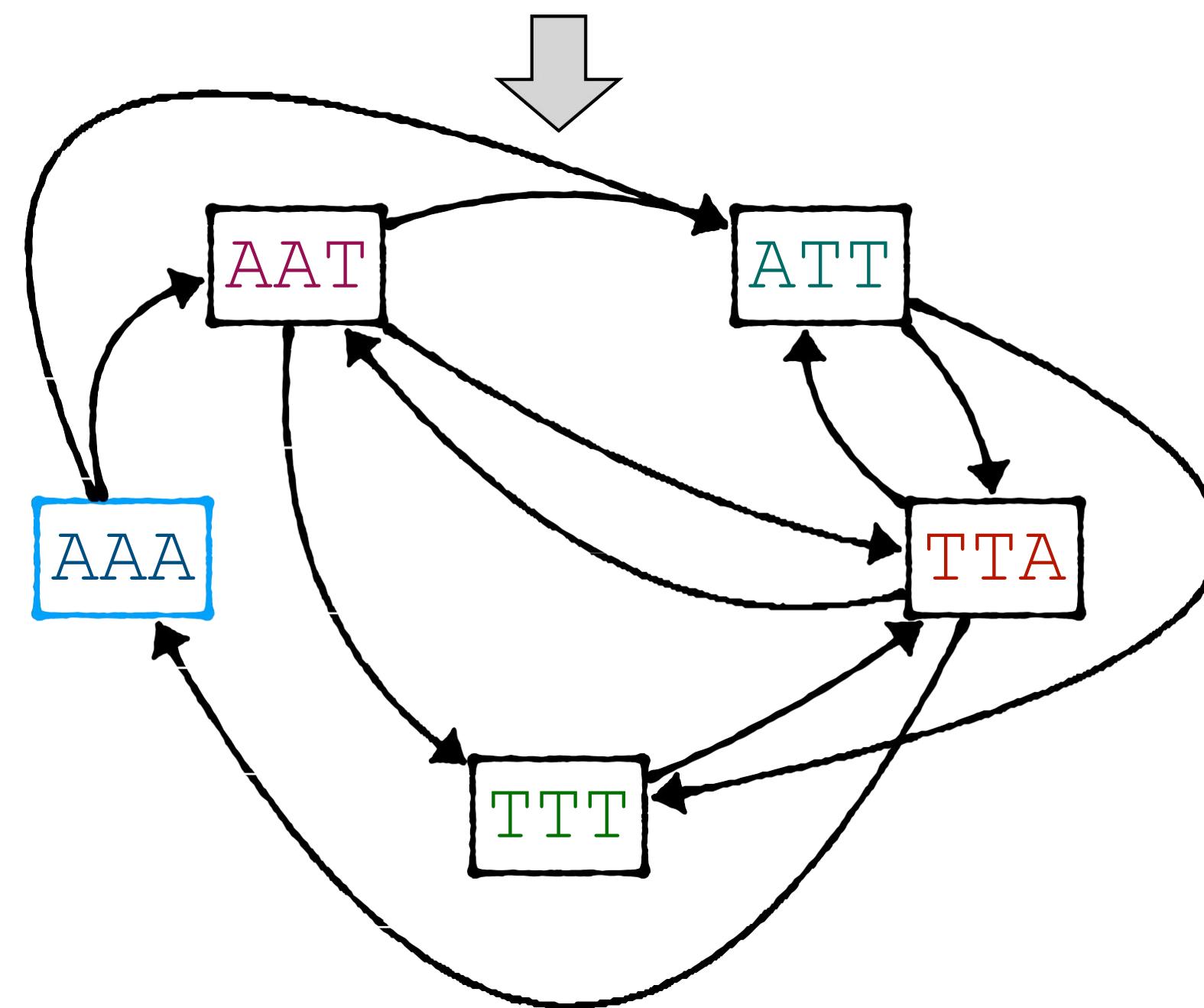
# SCS. Граф перекрытий

AAA, AAT, ATT, TTA, TTT



# SCS. Обход графа перекрытий

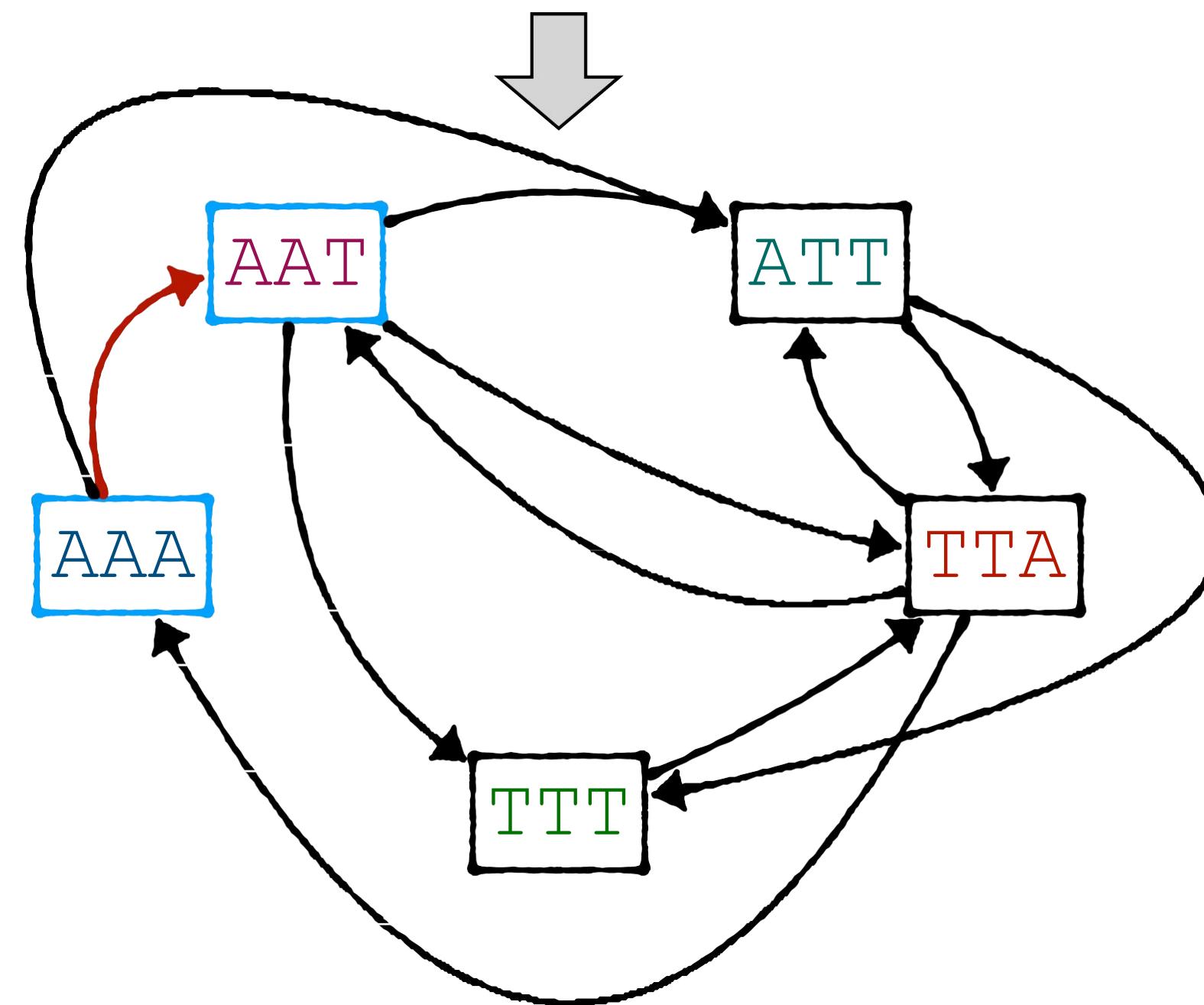
AAA, AAT, ATT, TTA, TTT



AAA

# SCS. Обход графа перекрытий

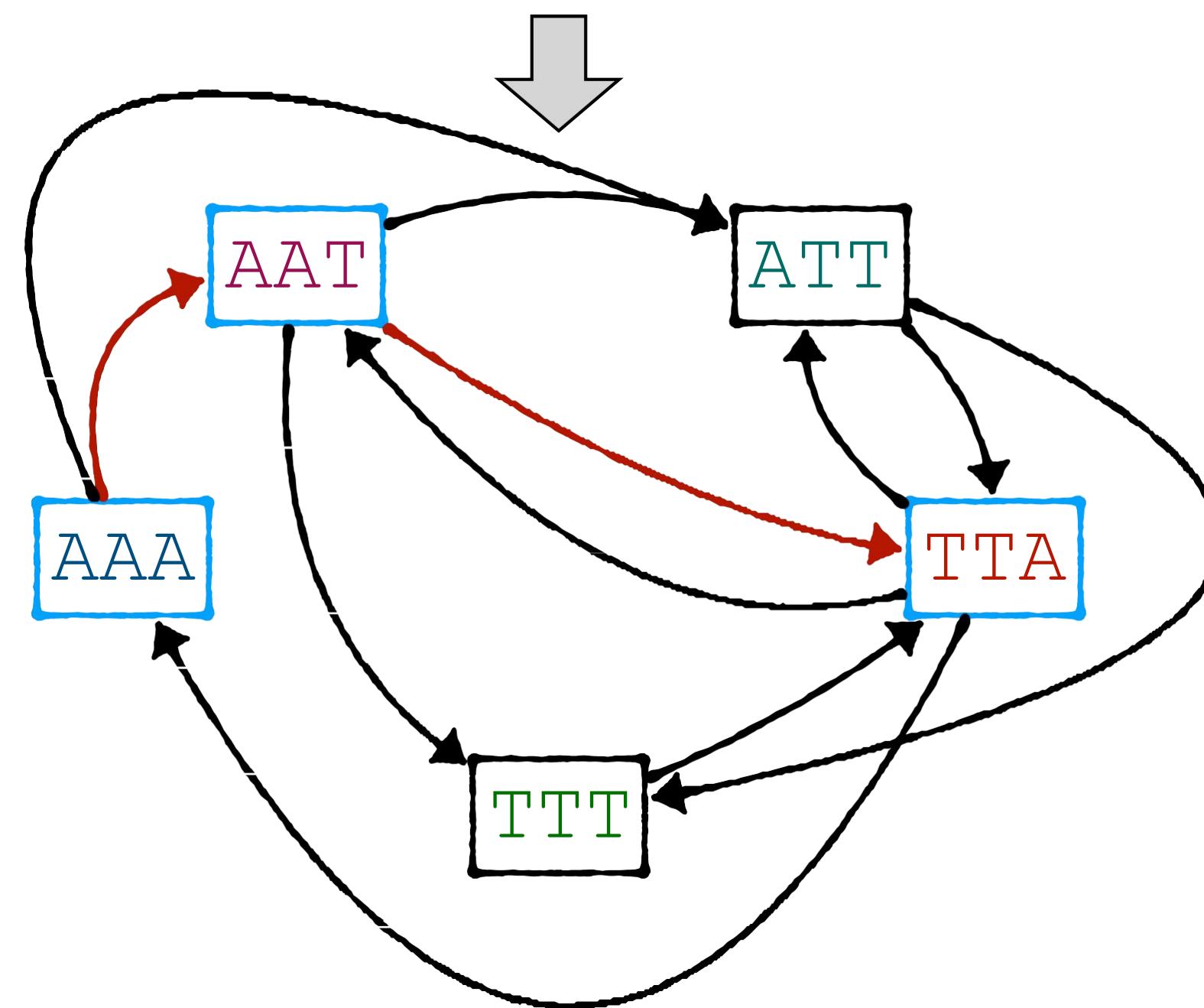
AAA, AAT, ATT, TTA, TTT



AAAAT

# SCS. Обход графа перекрытий

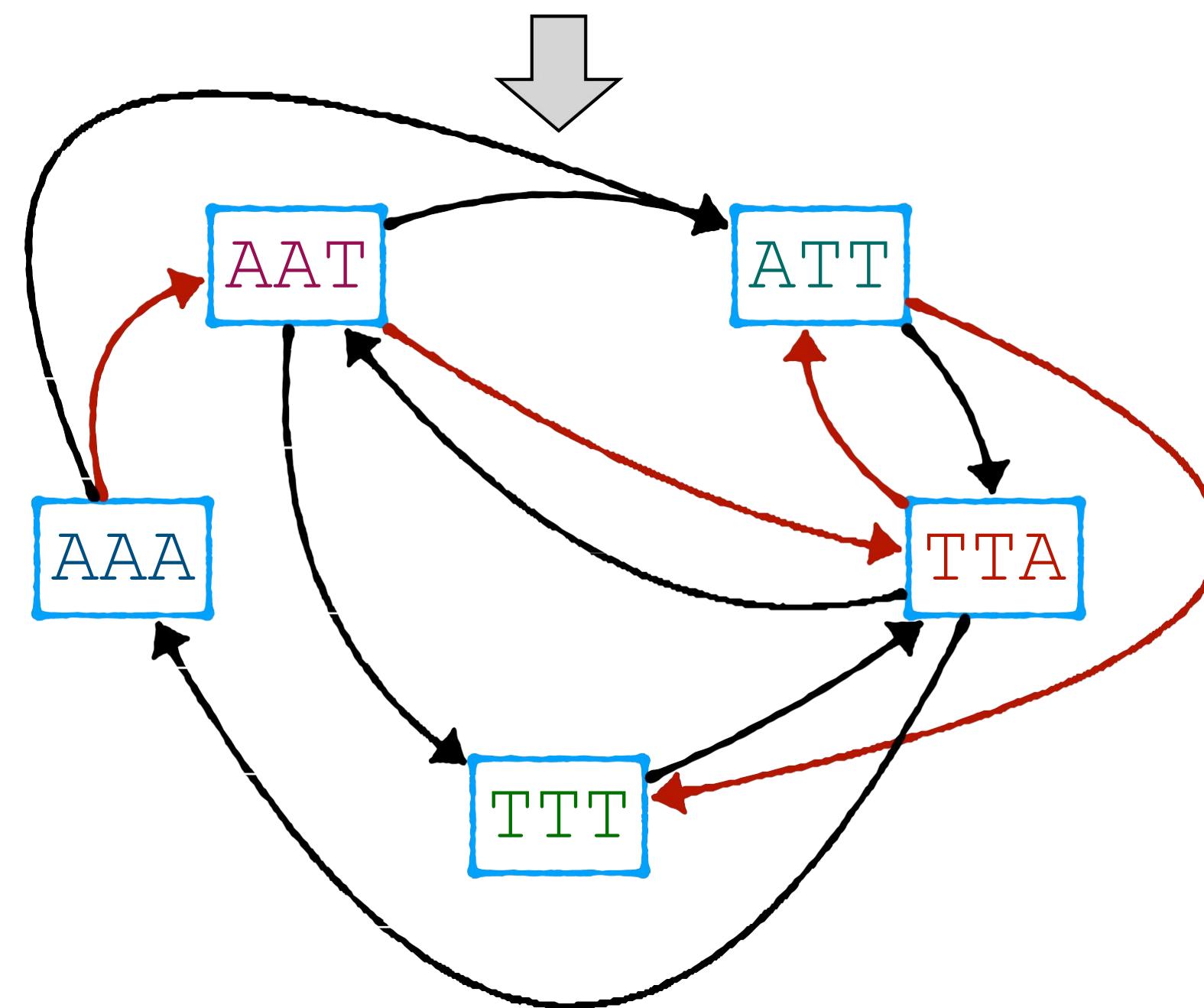
AAA, AAT, ATT, TTA, TTT



AAA<sub>1</sub>AAT<sub>2</sub>TTA<sub>3</sub>

# SCS. Обход графа перекрытий

AAA, AAT, ATT, TTA, TTT



AAA<sub>1</sub>AAT<sub>2</sub>ATT<sub>3</sub>TTA<sub>4</sub>TTT<sub>5</sub>

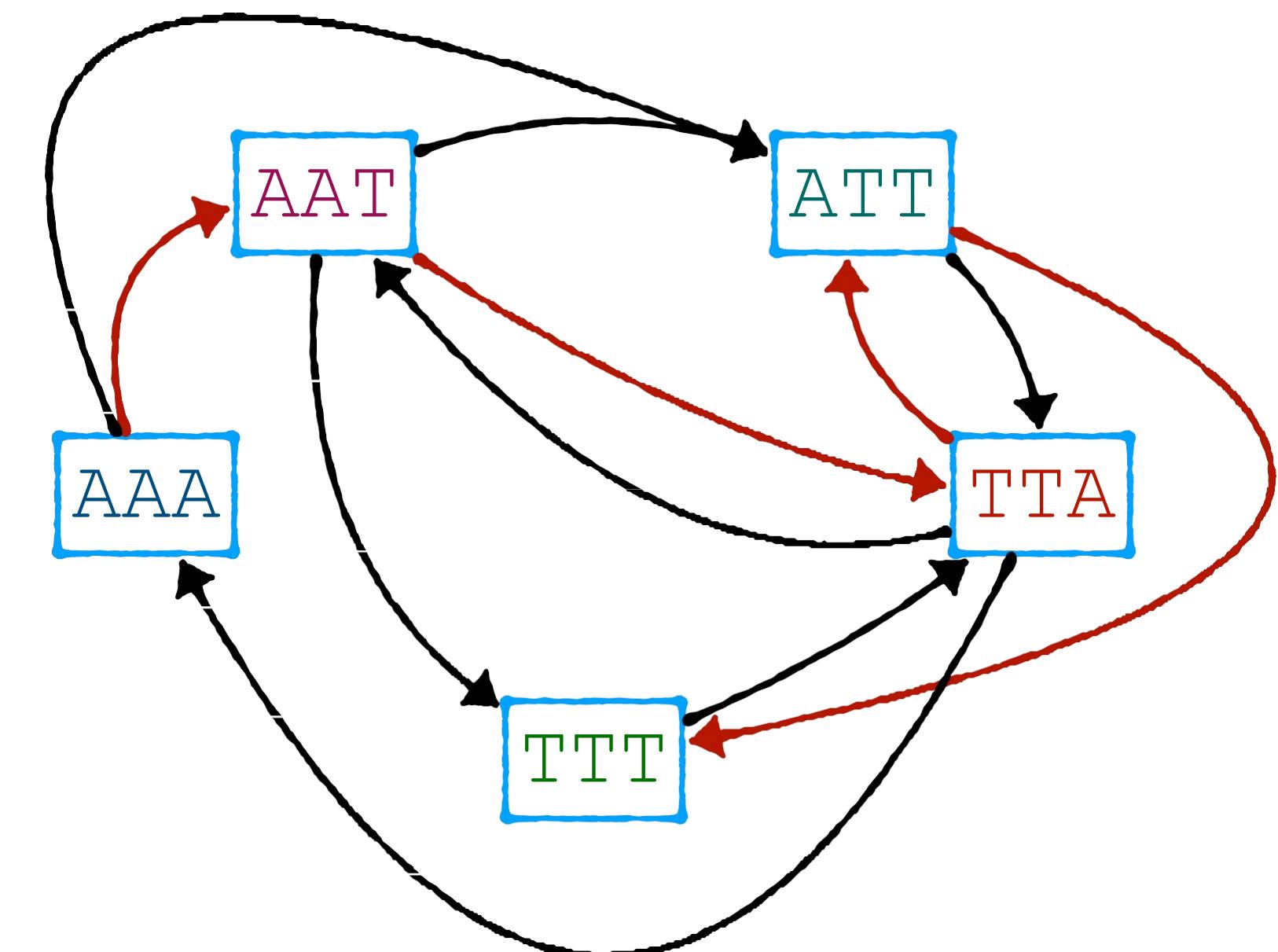
# SCS. Обход графа перекрытий

## Алгоритм

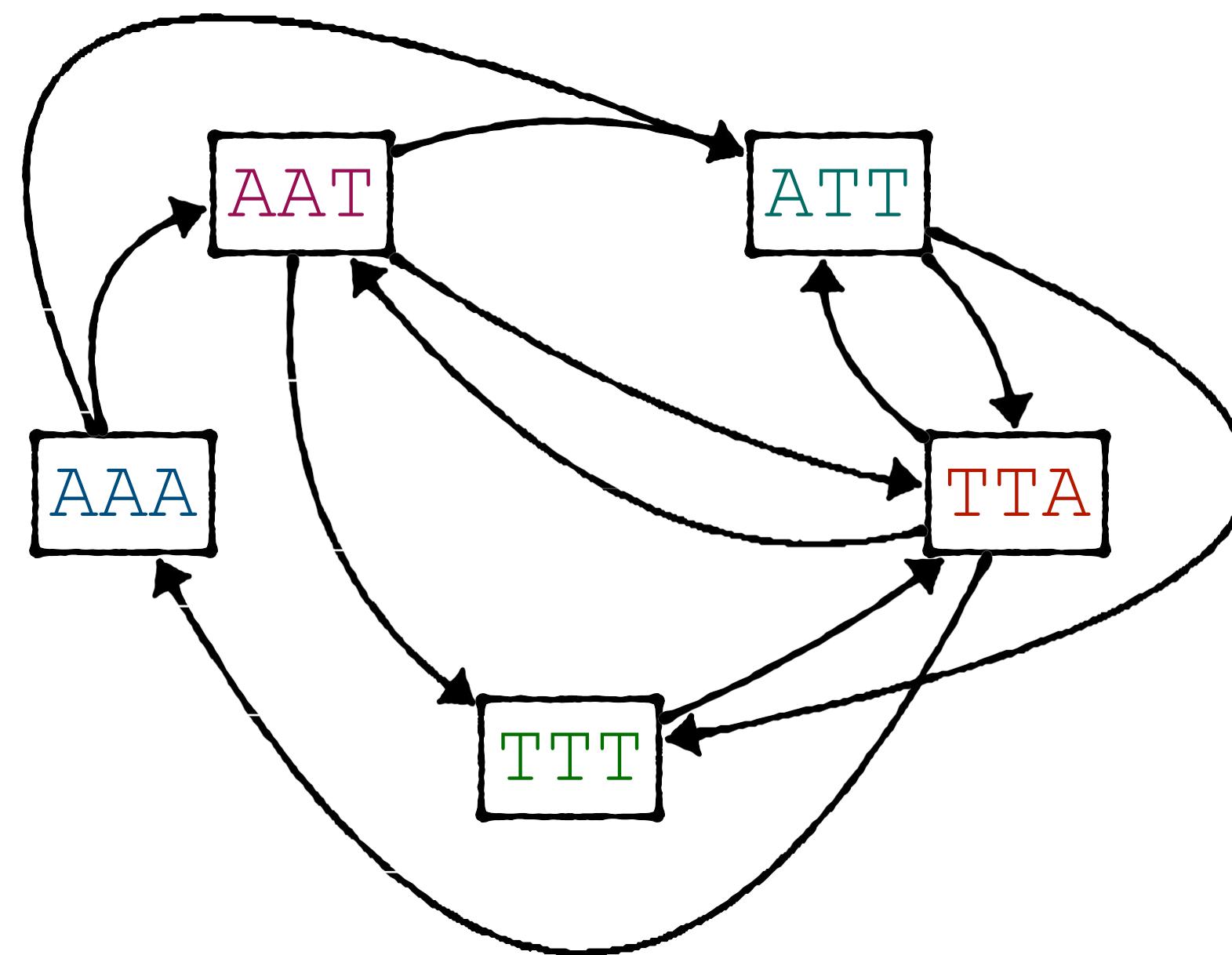
- ⋮
- Найдем Гамильтонов путь в графе перекрытий

## Замечания:

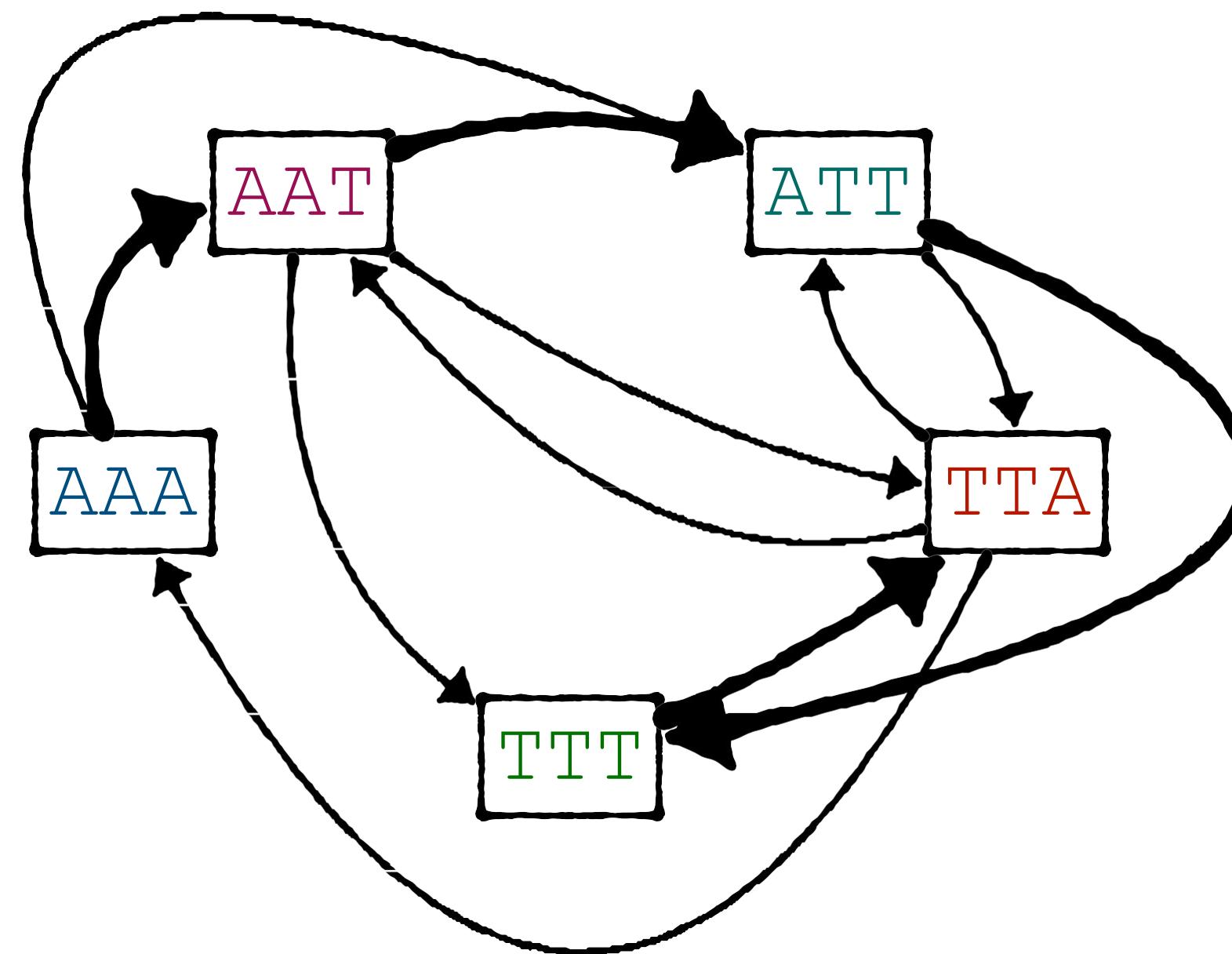
- Находит не кратчайшую суперстроку
- Все еще трудная задача
- Пути может и не существовать



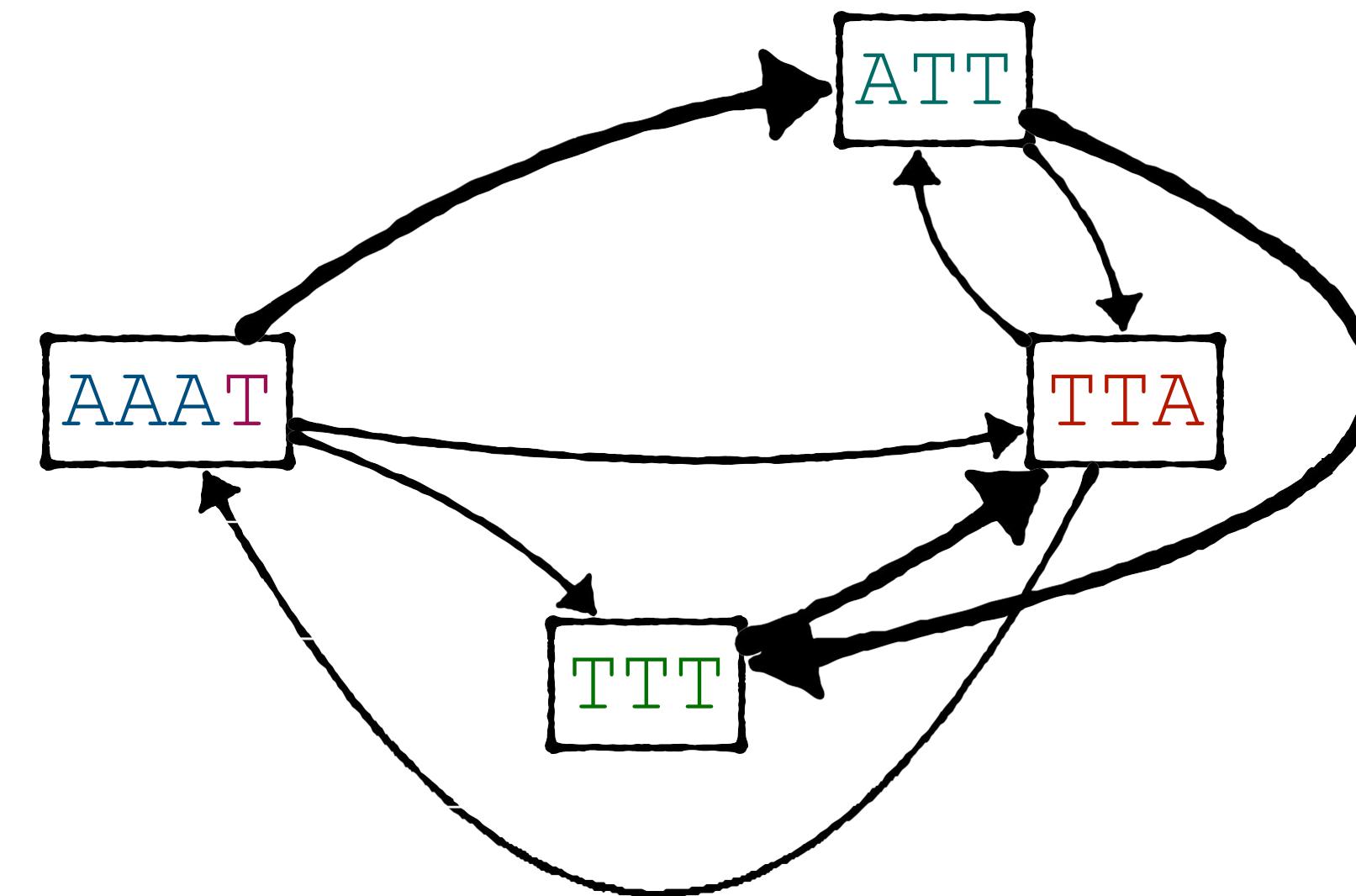
# SCS. Жадный алгоритм



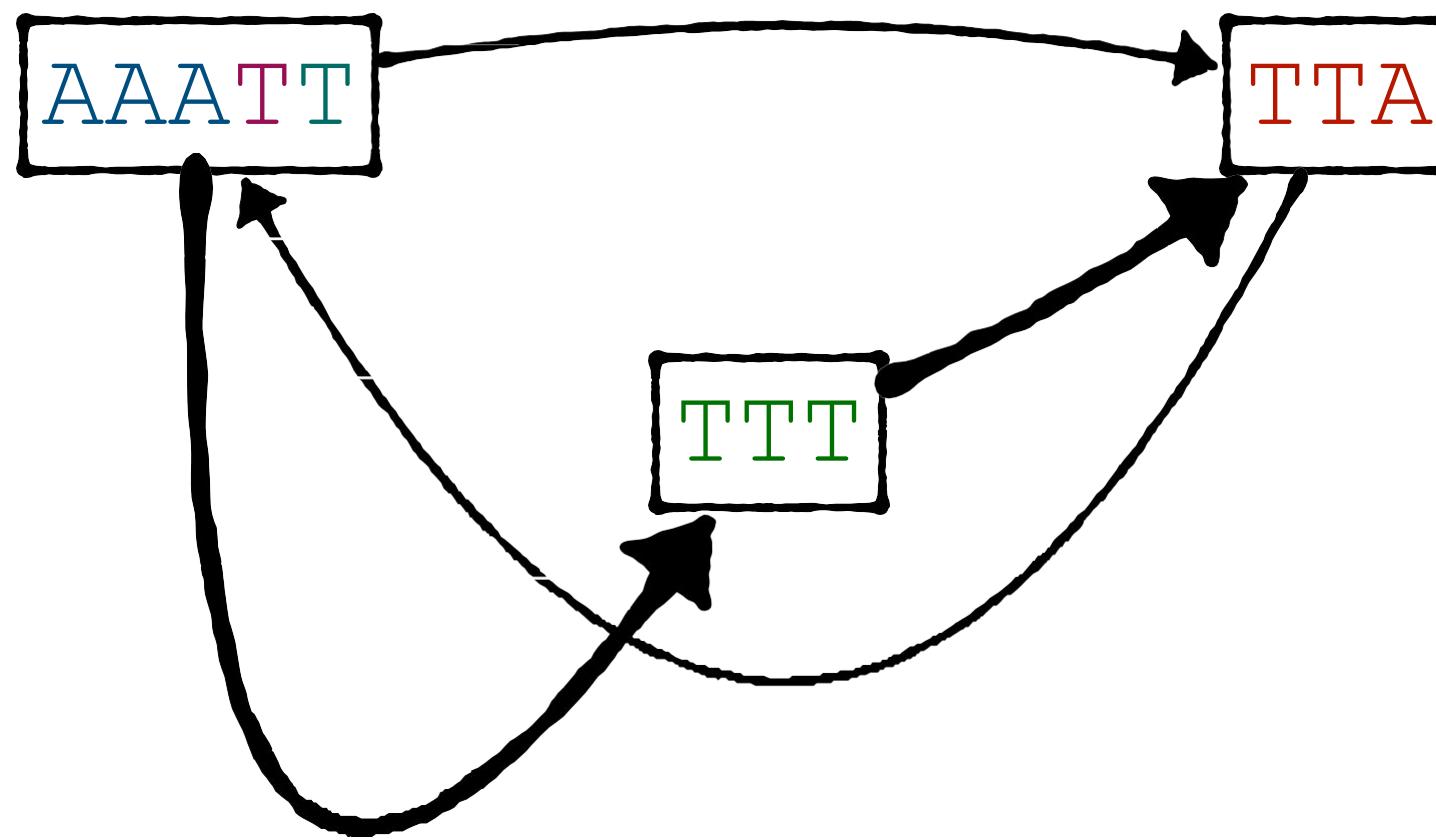
# SCS. Жадный алгоритм



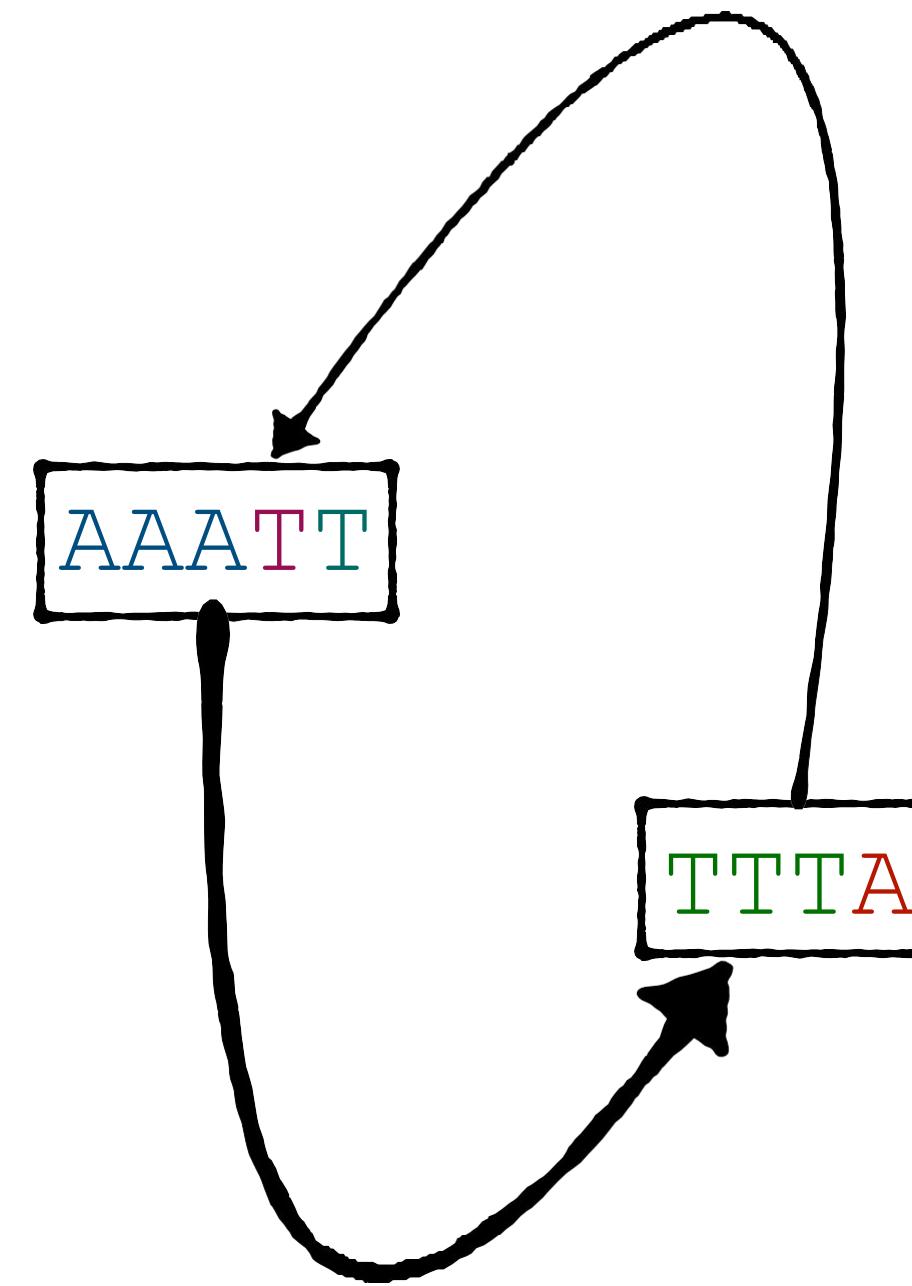
# SCS. Жадный алгоритм



# SCS. Жадный алгоритм



# SCS. Жадный алгоритм



# SCS. Жадный алгоритм

АААТТТА

# **SCS. Жадный алгоритм**

## **Алгоритм:**

- Строим граф перекрытий
- Пока есть вершины объединяем те, что соединены тяжелыми ребрами

## **Замечания:**

- Находит не кратчайшую суперстроку
  - Если ребер нет, а вершины еще остались то просто конкатенируем их
- Задача полиномиальная

# String graph

A **string graph** is obtained from an overlap graph by removing redundancy:

- redundant reads (those fully contained in another read)
- transitively redundant edges (if  $a \rightarrow c$  and  $a \rightarrow b \rightarrow c$ , then remove  $a \rightarrow c$ )

# FROM OVERLAP GRAPHS TO STRING GRAPHS

Overlap graph for  $k = 3$ ,



String graph for  $k = 3$ ,



The read CTGCT is contained in ACTGCT, so it is redundant

# COMPARISON STRING GRAPH / DE BRUIJN GRAPH

On the same example, compare the de Bruijn graph with the string graph:

AGTGCT

GTGCTA

GCTAA

String graph,  $k = 3$ :

AGTGCT → GTGCTA → GCTAA

de Bruijn graph,  $k = 3$ :

AGT → GTG → TGC → GCT → CTA → TAA

# COMPARISON STRING GRAPH / DE BRUIJN GRAPH

Let's add an error:

AGTGCT

GTGATA

GCTAA

String graph,  $k = 3$ :

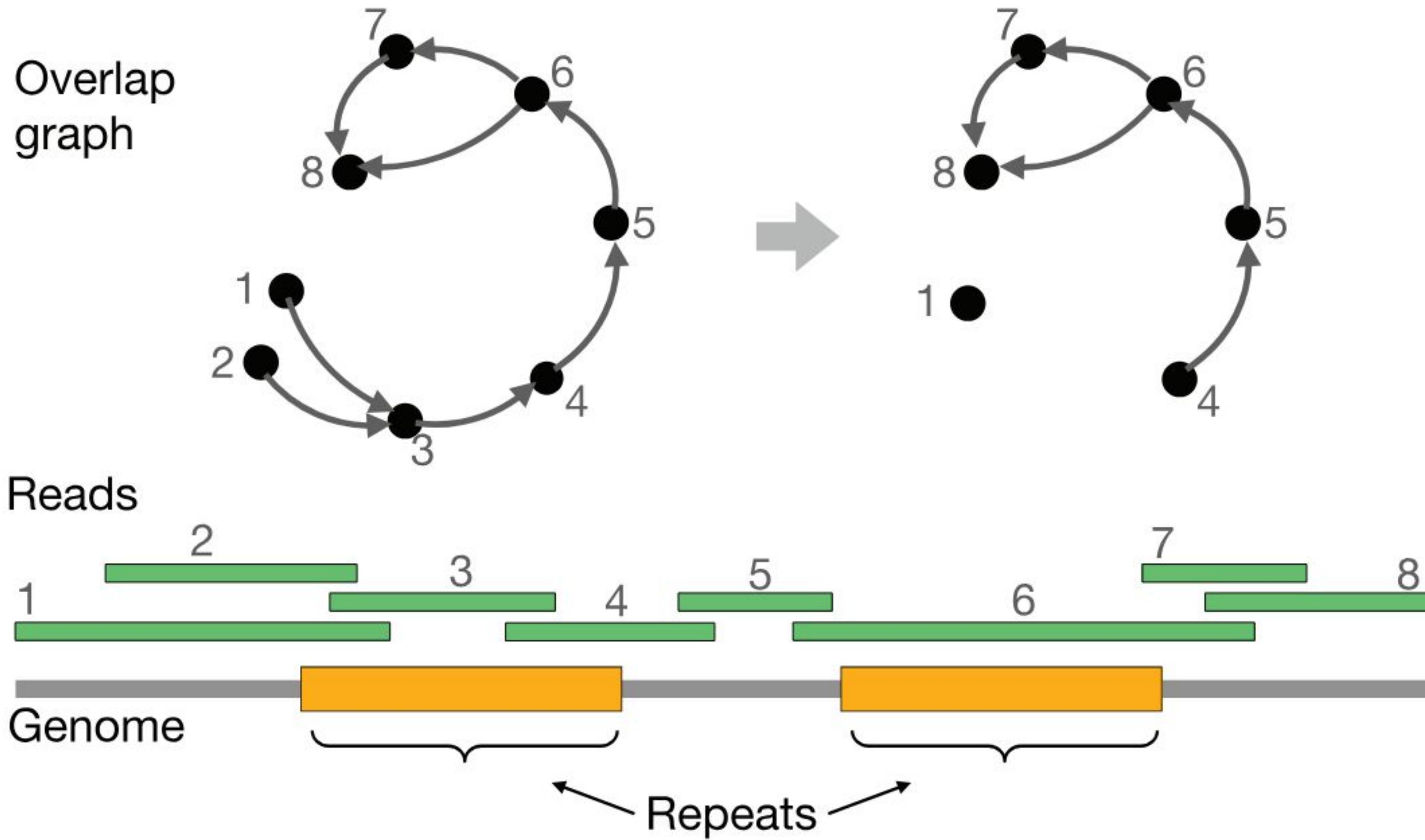


de Bruijn graph,  $k = 3$ :



**All of these don't work actually**

# All of these don't work actually

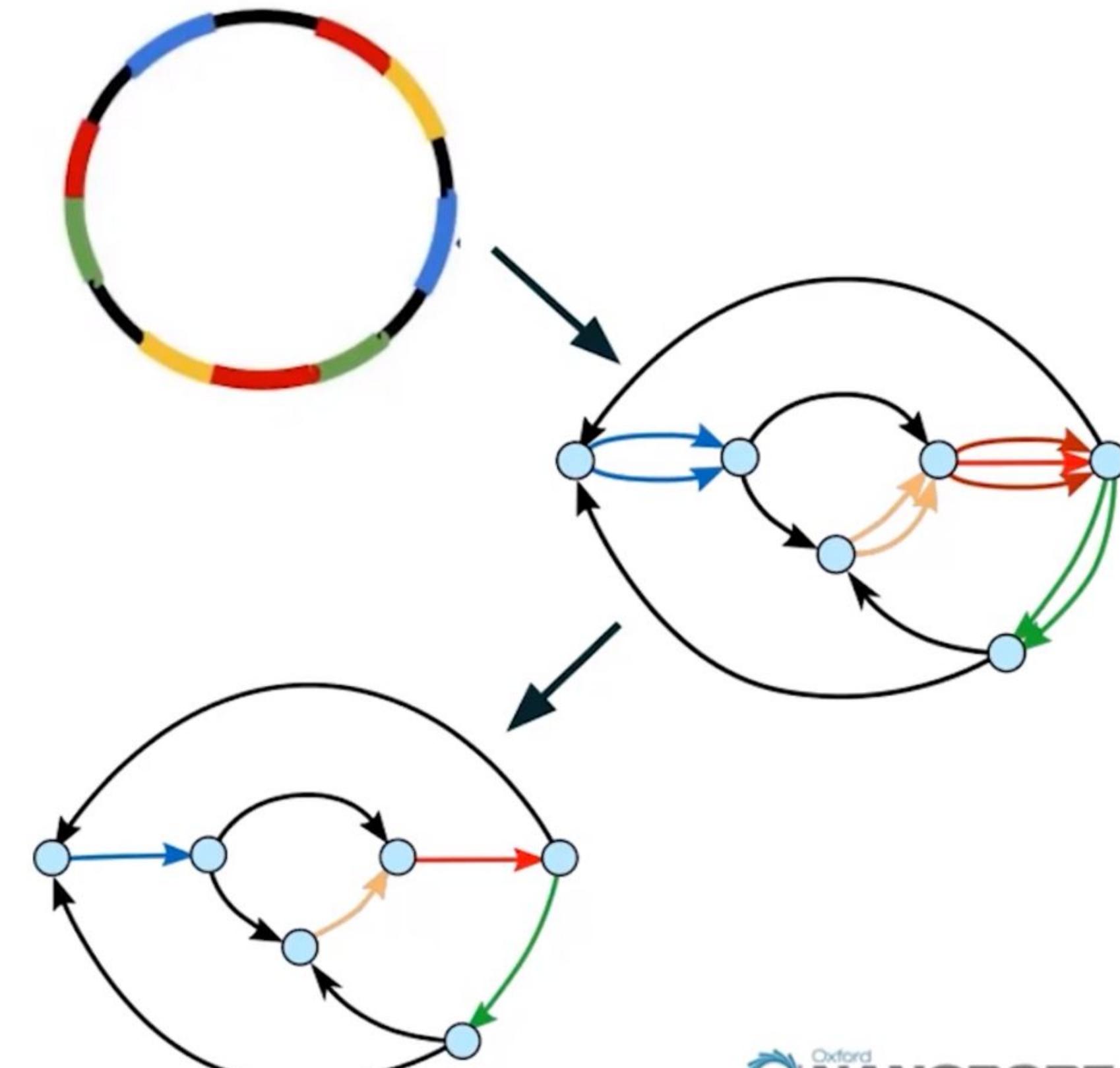


# How modern long read assemblers work?

# Flye: Kolmogorov et al.

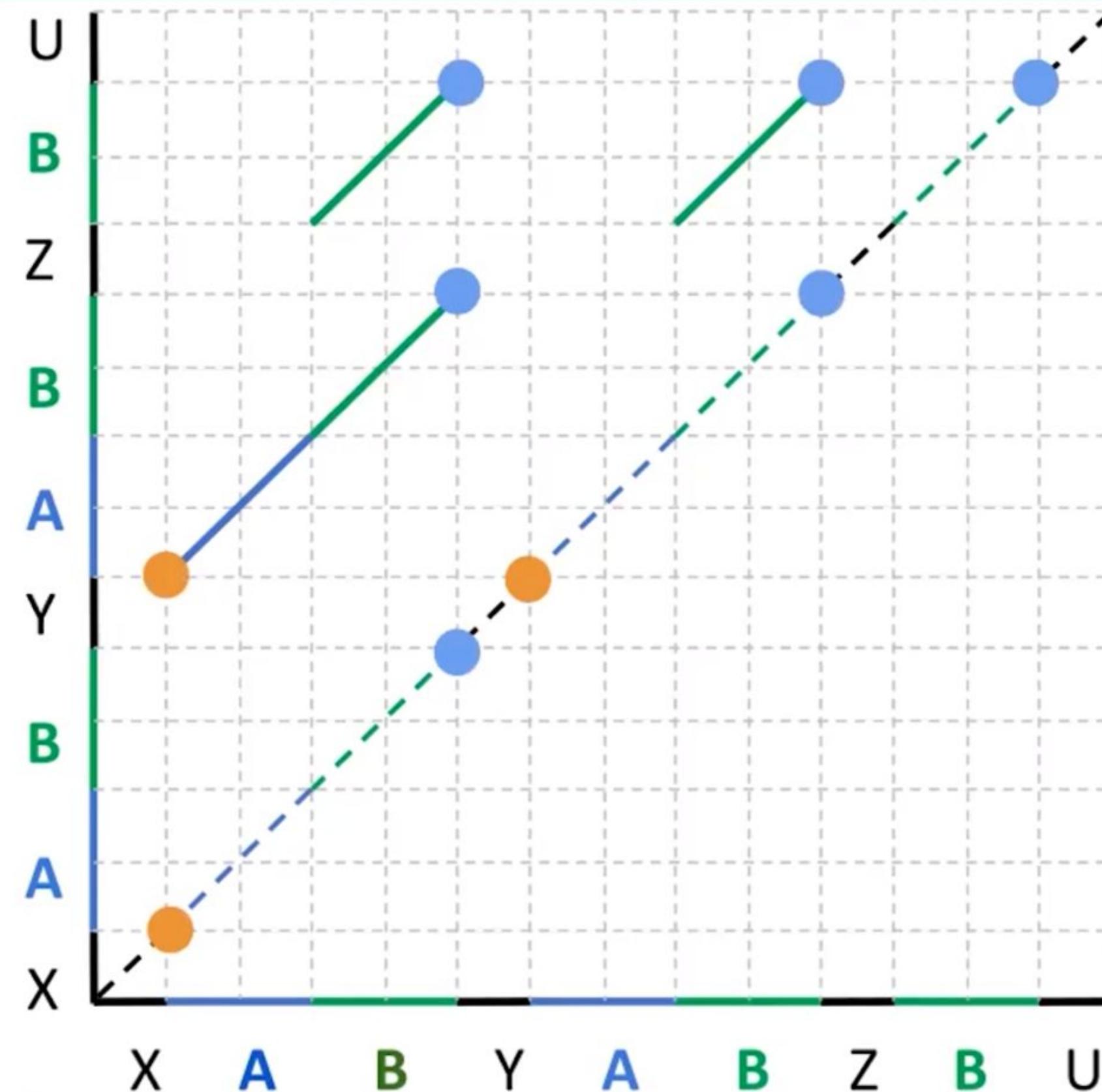
## Repeat Graphs

- Repeat graphs: de Bruijn graphs for approximate matches (Pevzner et al. *Genome Res.* 2004)
- Goal: reveal genomic repeats



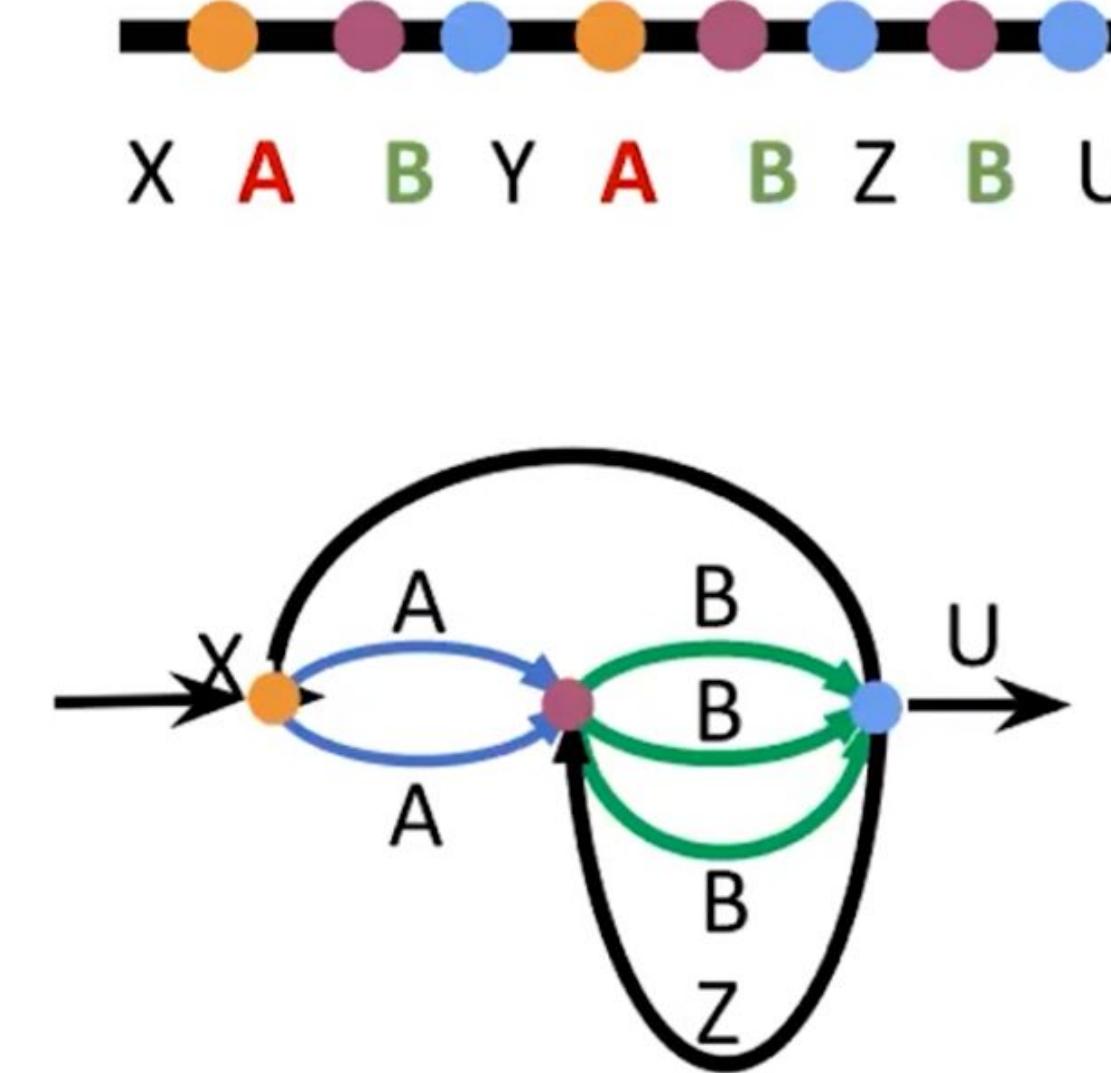
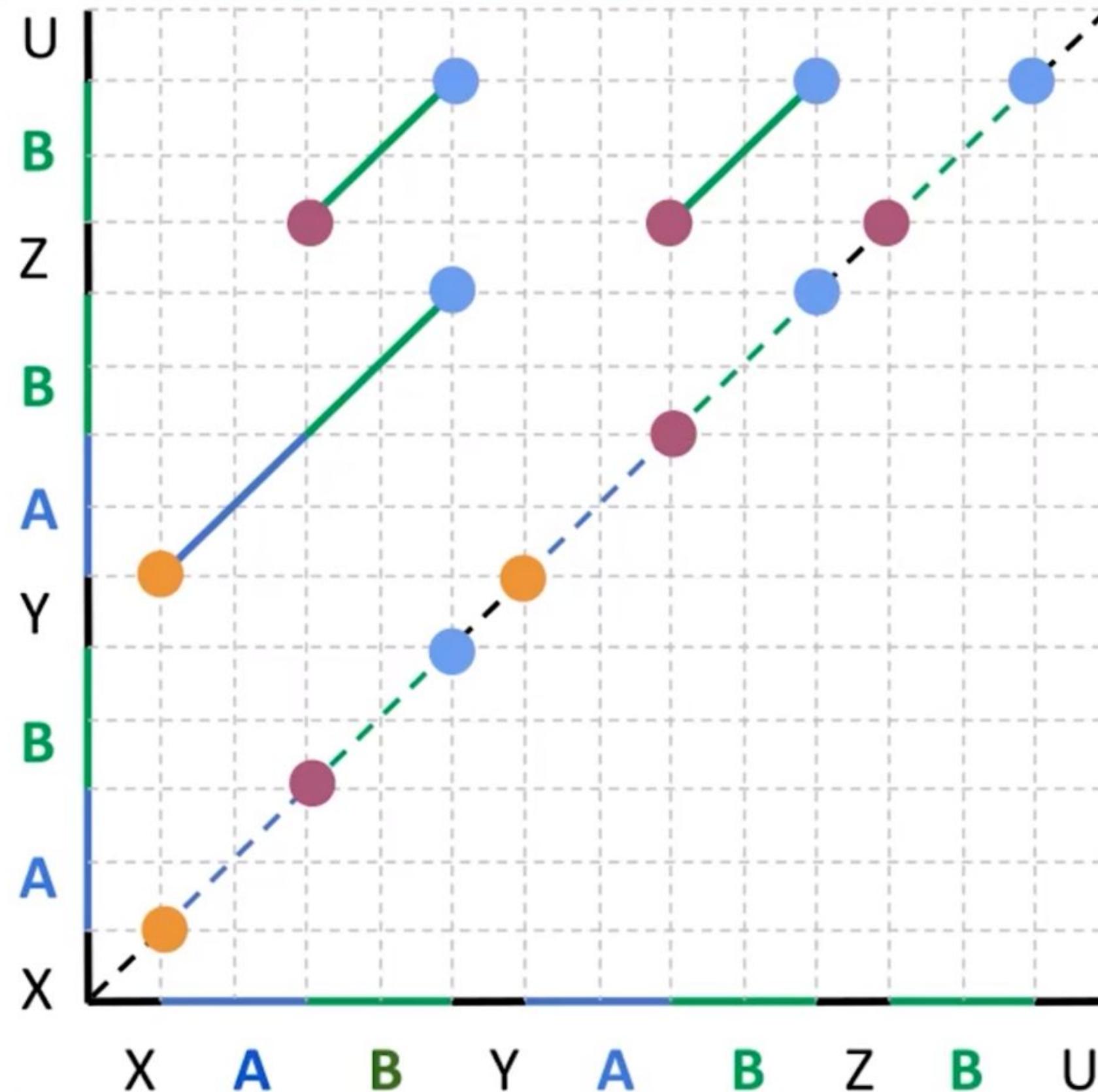
# Flye: Kolmogorov et al.

## Repeat Graph Construction



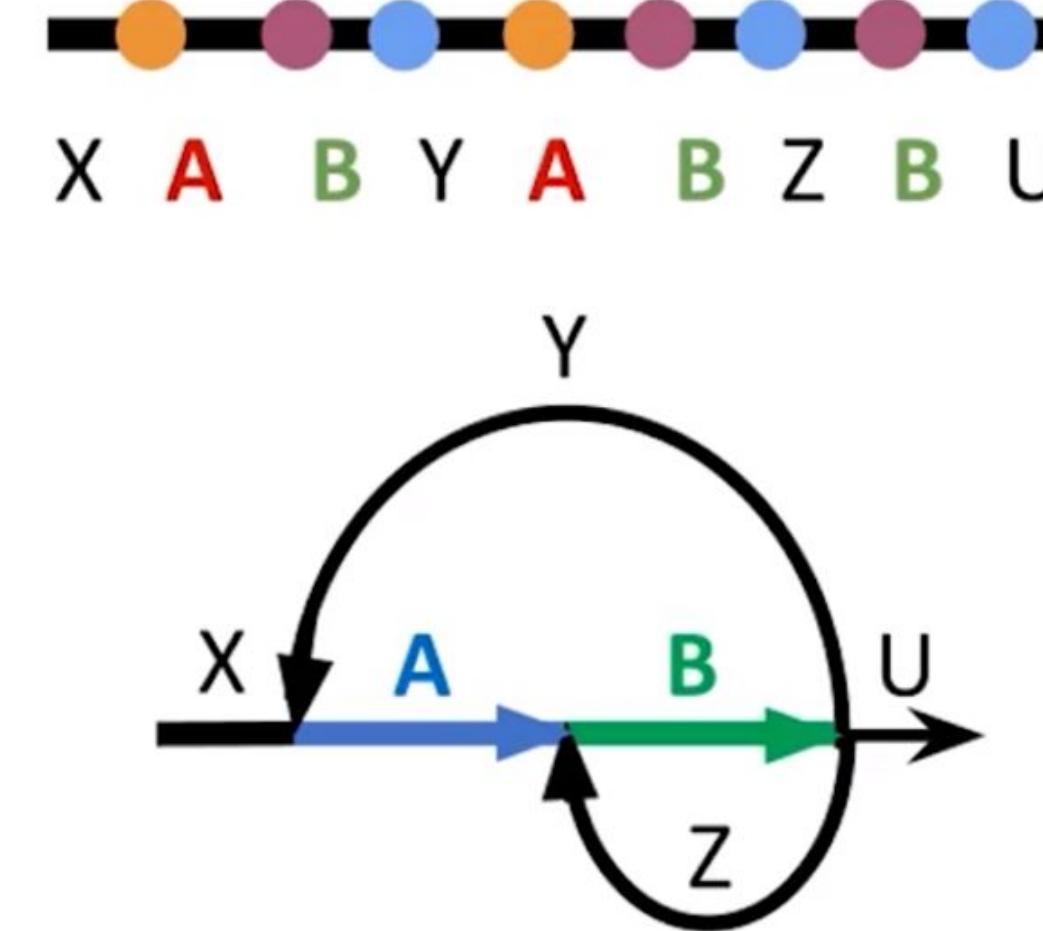
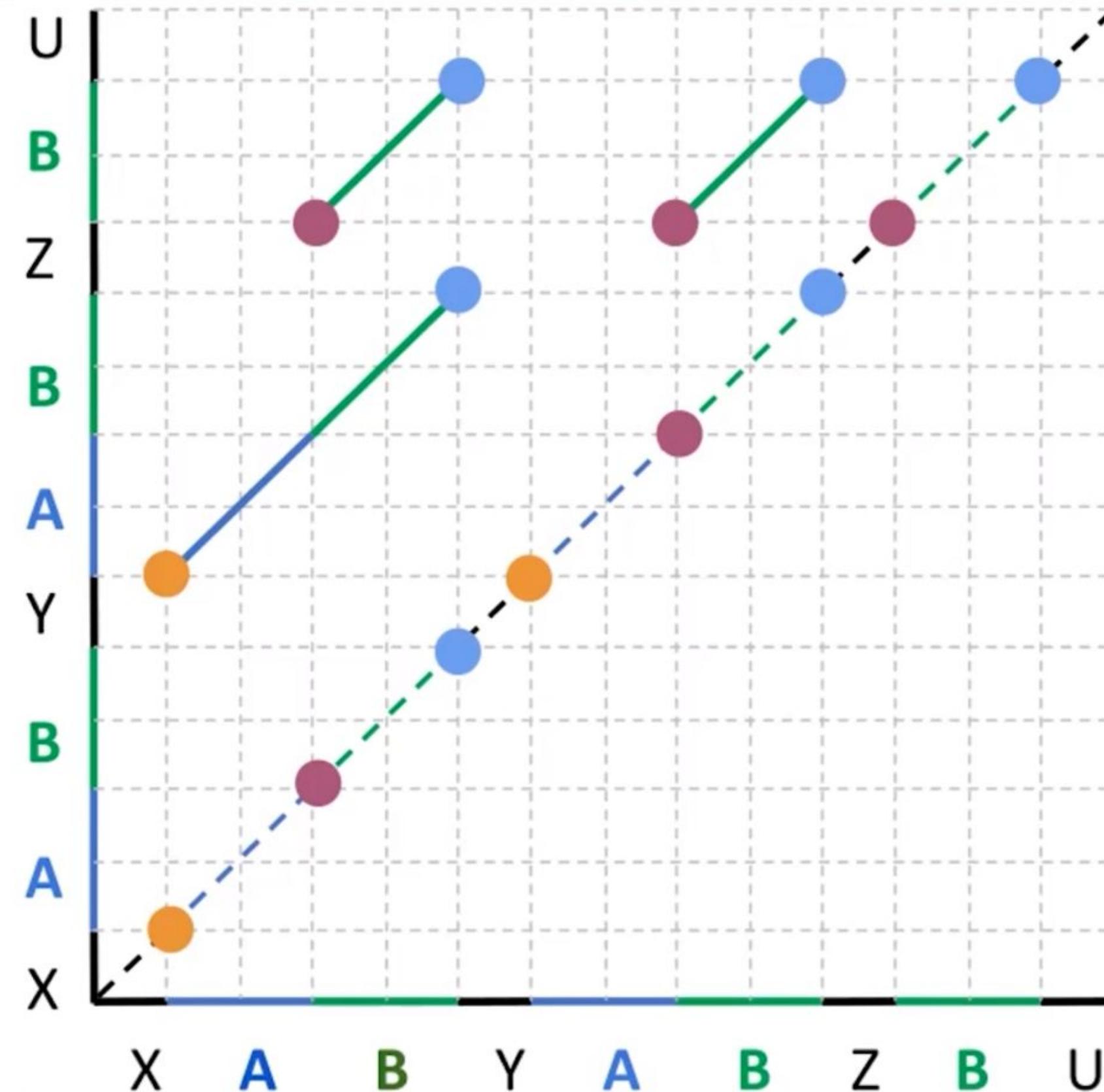
# Flye: Kolmogorov et al.

## Repeat Graph Construction



# Flye: Kolmogorov et al.

## Repeat Graph Construction

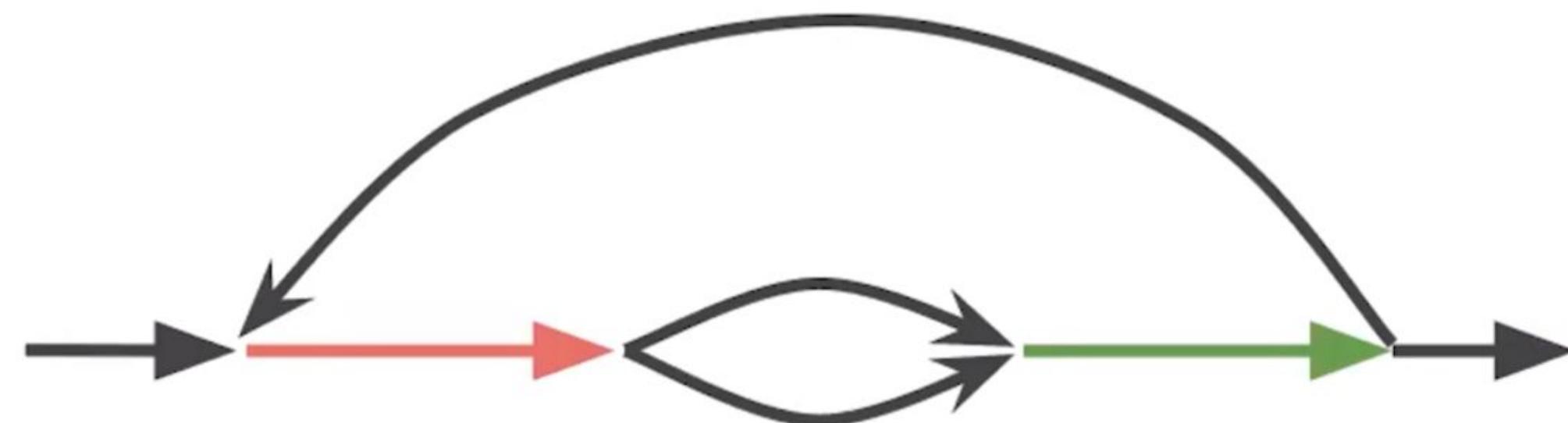


# Flye: Kolmogorov et al.

## From Reads to Repeat Graph

---

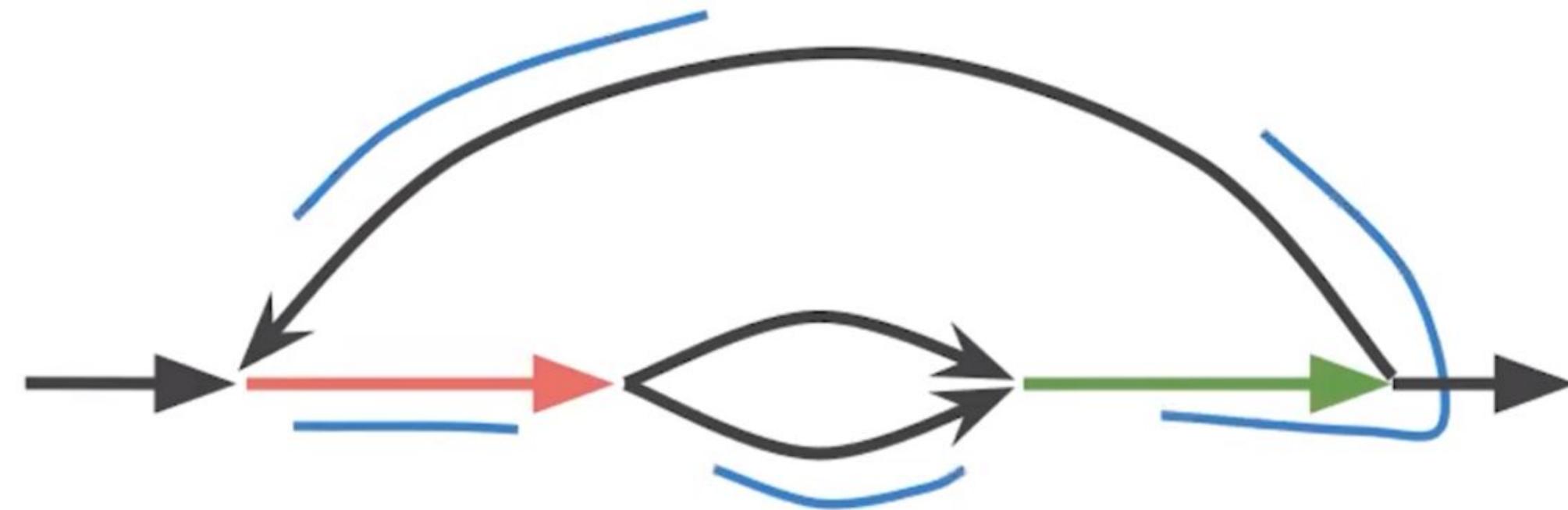
How to generate repeat graph from reads?



# Flye: Kolmogorov et al.

## From Reads to Repeat Graph

How to generate repeat graph from reads?



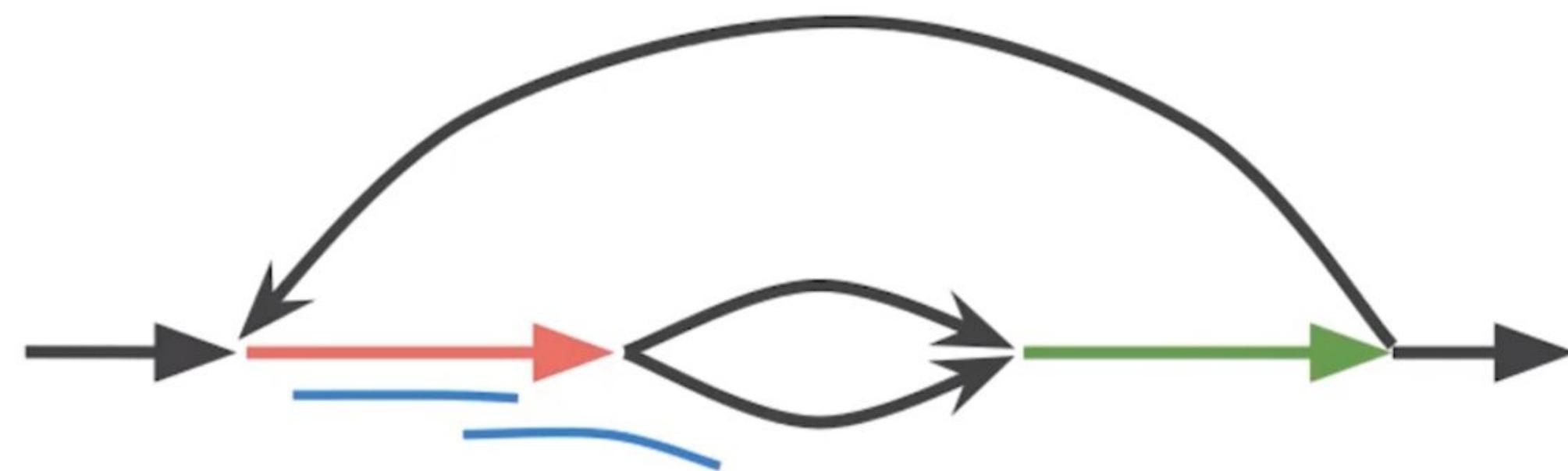
Reads = paths on the graph

# Flye: Kolmogorov et al.

## From Reads to Repeat Graph

---

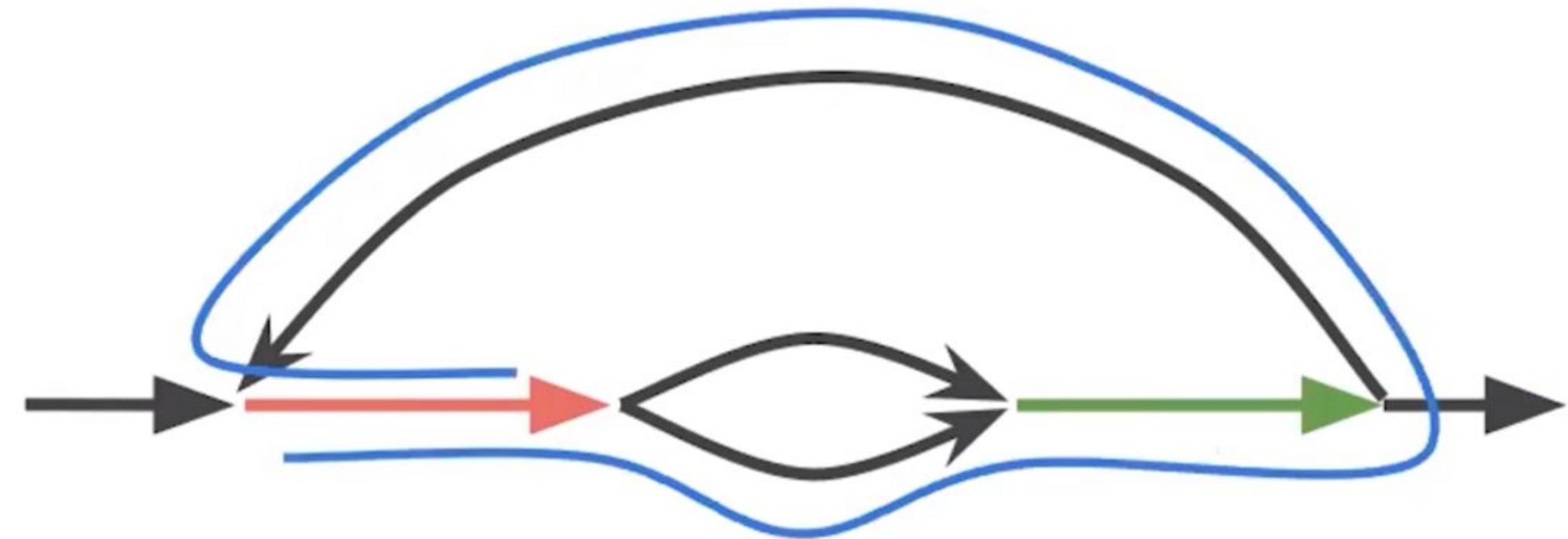
How to generate repeat graph from reads?



# Flye: Kolmogorov et al.

## From Reads to Repeat Graph

How to generate repeat graph from reads?



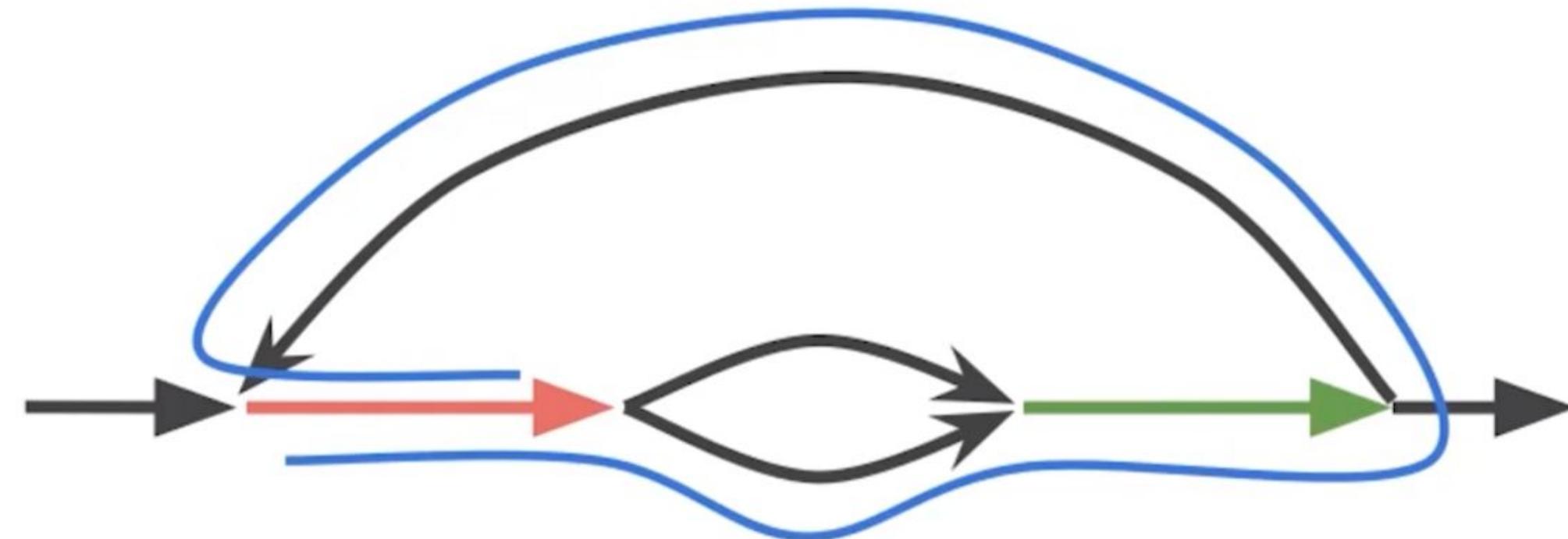
Disjointig = random walk on the graph

# Flye: Kolmogorov et al.

## From Reads to Repeat Graph

How to generate repeat graph from reads?

RepeatGraph[Genome] = RepeatGraph[disjointigs] (Pevzner et al., Genome Res. 2004, Kolmogorov et al., Nature Biotech. 2019)



Disjointig = random walk on the graph

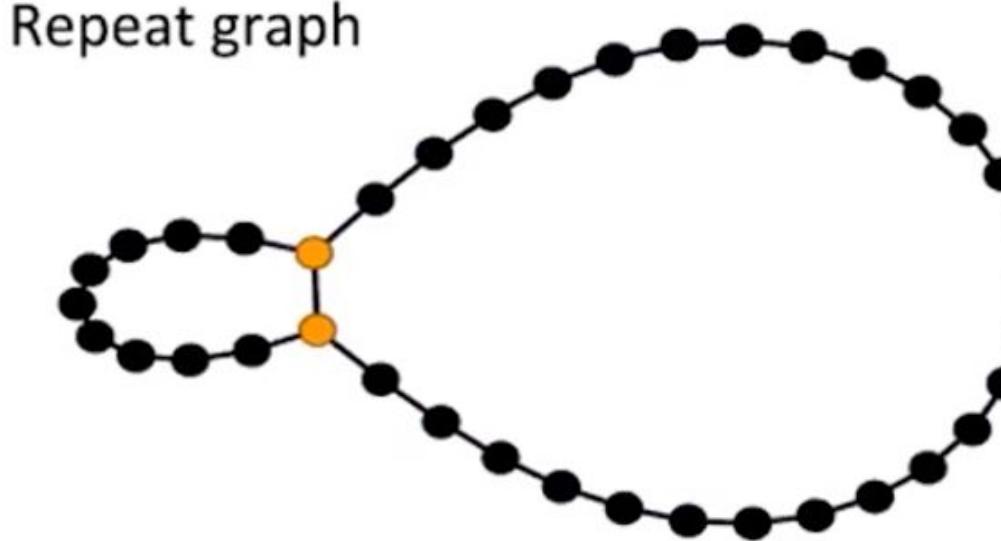
# Flye: Kolmogorov et al.

## Untangling Repeat Graph

Genome with one repeat



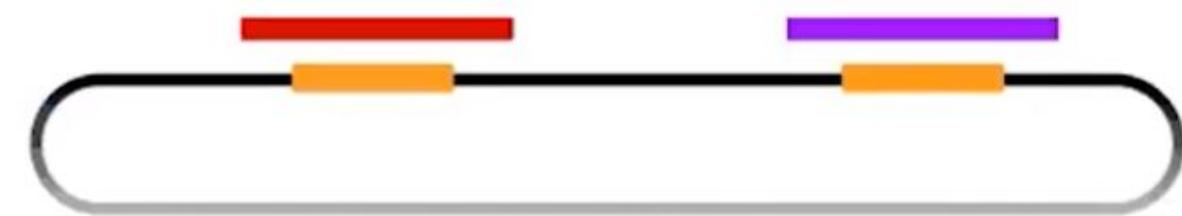
Repeat graph



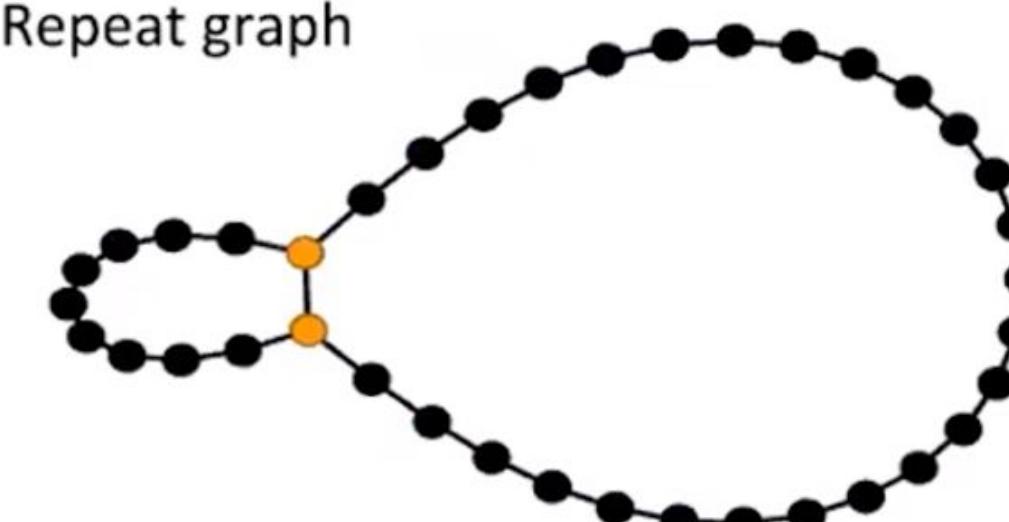
# Flye: Kolmogorov et al.

## Untangling Repeat Graph

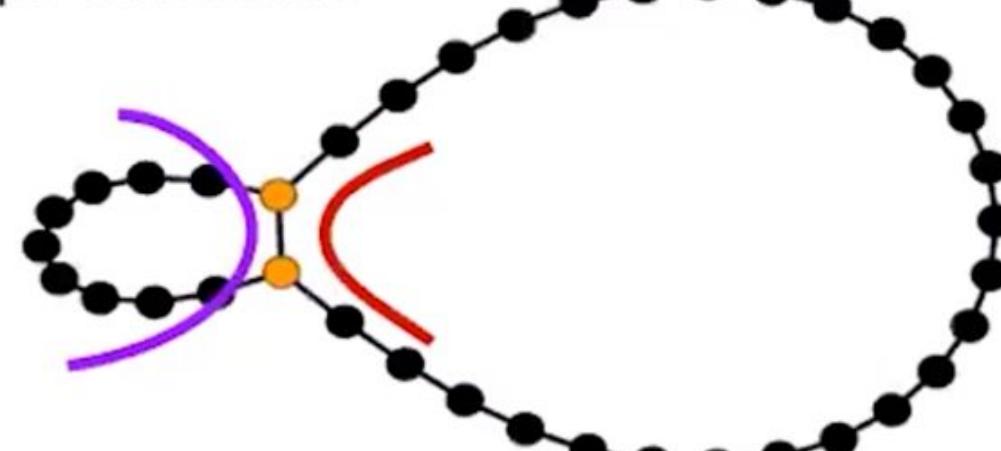
Genome with one repeat



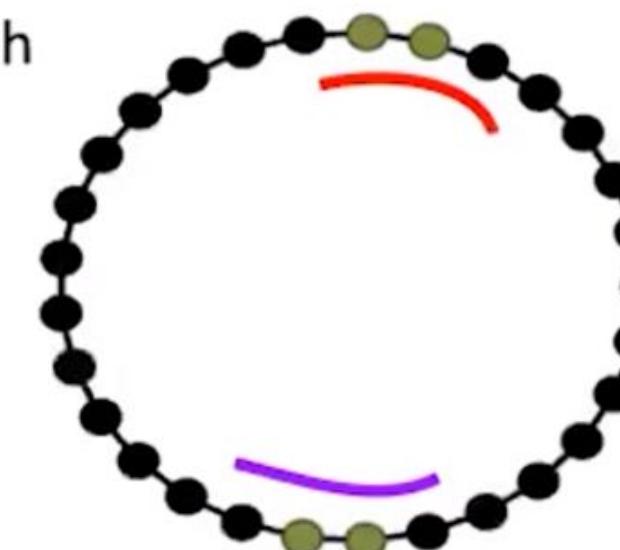
Repeat graph



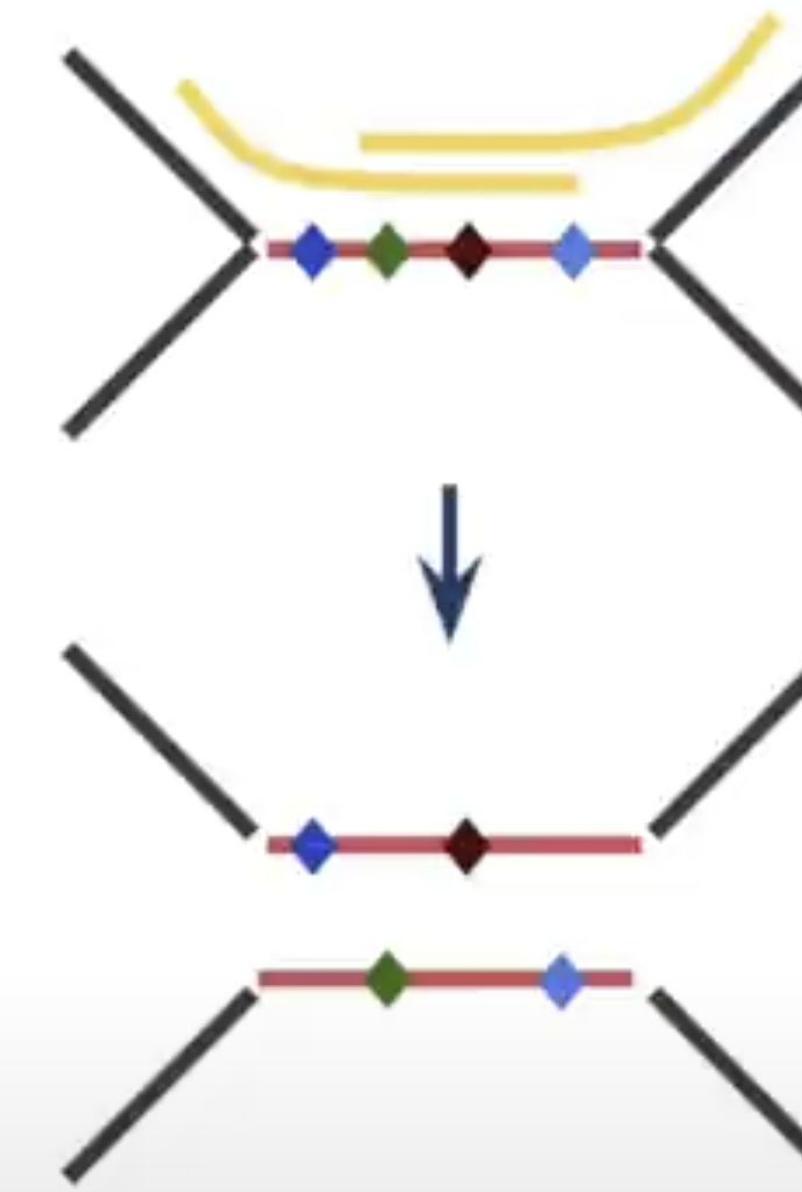
Repeat graph with reads



Simplified graph



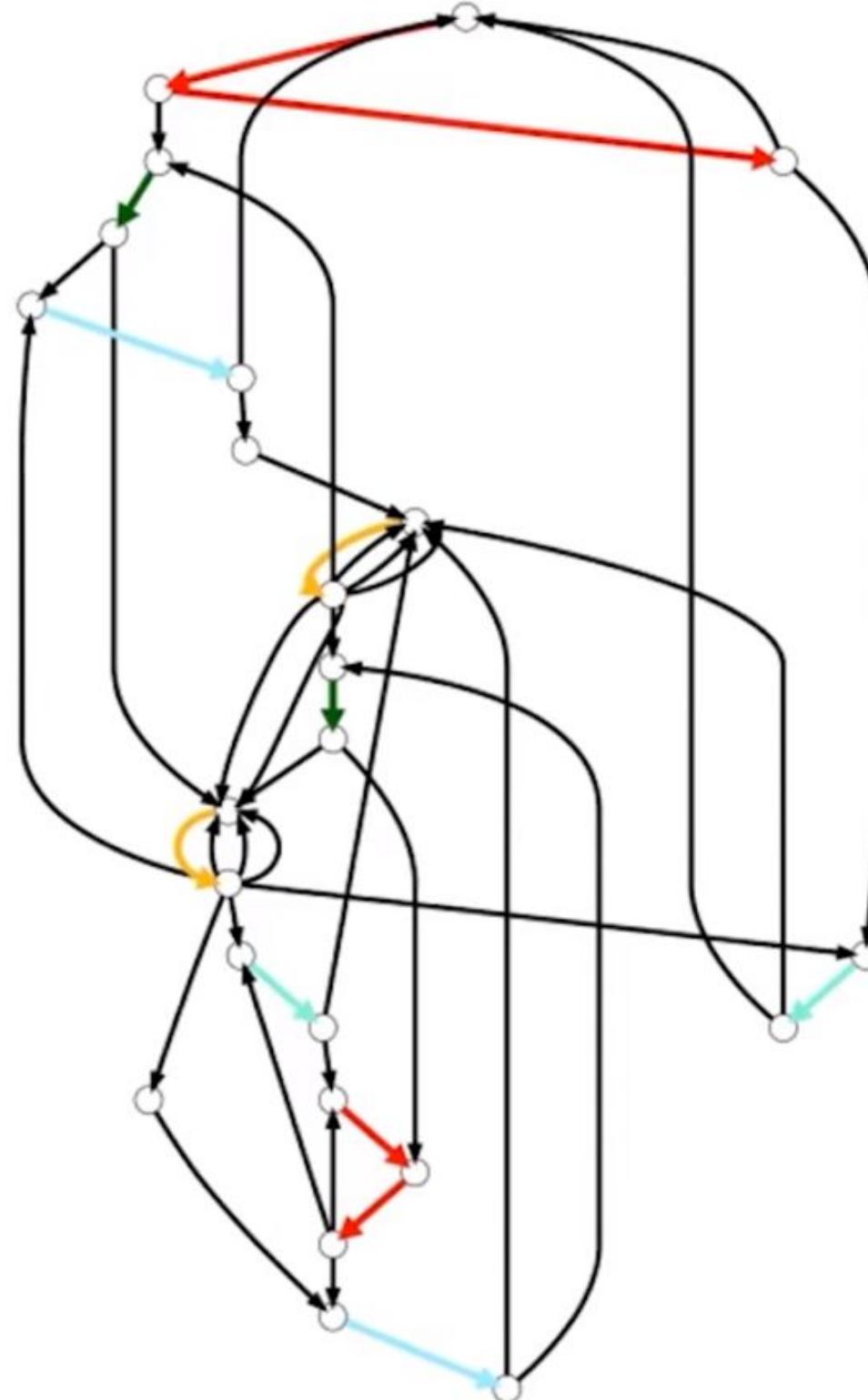
# Flye: Kolmogorov et al.



Trestle algorithm: extend into  
unbridged repeats using  
differences between copies

# Flye: Kolmogorov et al.

## Repeat Graph of *K. pneumoniae*



- **Repetitive** edges are colored
- **Unique** edges are black
- **Contigs** generated from unique edges

# **Assembly quality**

# How good is my assembly?

---

**How much total sequence is in the assembly relative to estimated genome size?**

**How many pieces, and what is their size distribution?**

**Are the contigs assembled correctly?**

**Are the scaffolds connected in the right order / orientation?**

**How were the repeats handled?**

**Are all the genes I expected in the assembly?**

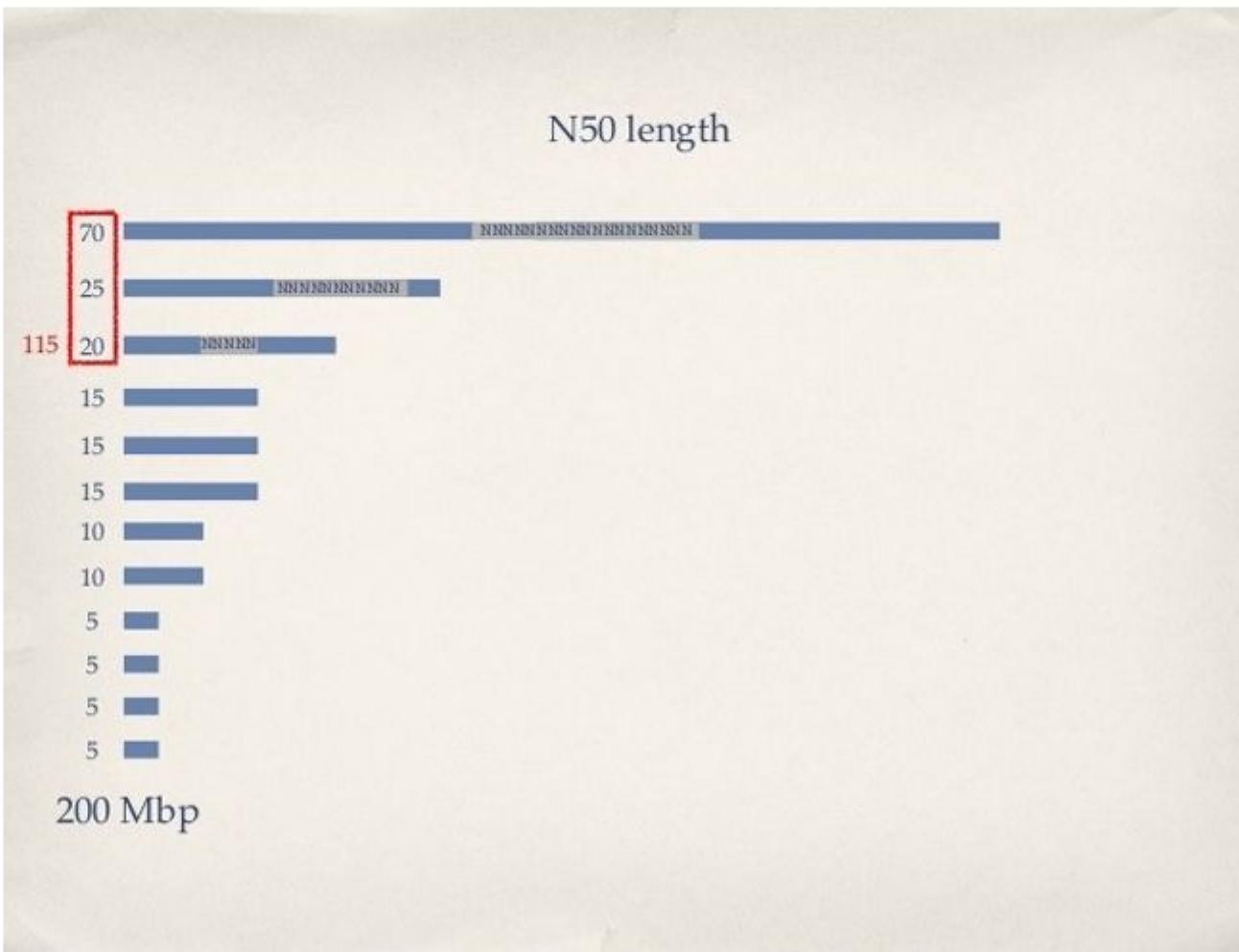
# N50: the most common measure of assembly quality

---

**N50** = length of the shortest contig in a set making up 50% of the total assembly length (*Larger is better*)

**NG50** = length of the shortest contig in a set making up 50% of the estimated genome size

NG50 is generally better



# Comparative analysis

---

Compare against

- A close reference genome
- Results from another assembler
- Self-comparison
- Versions of the same assembly

Whole genome alignment

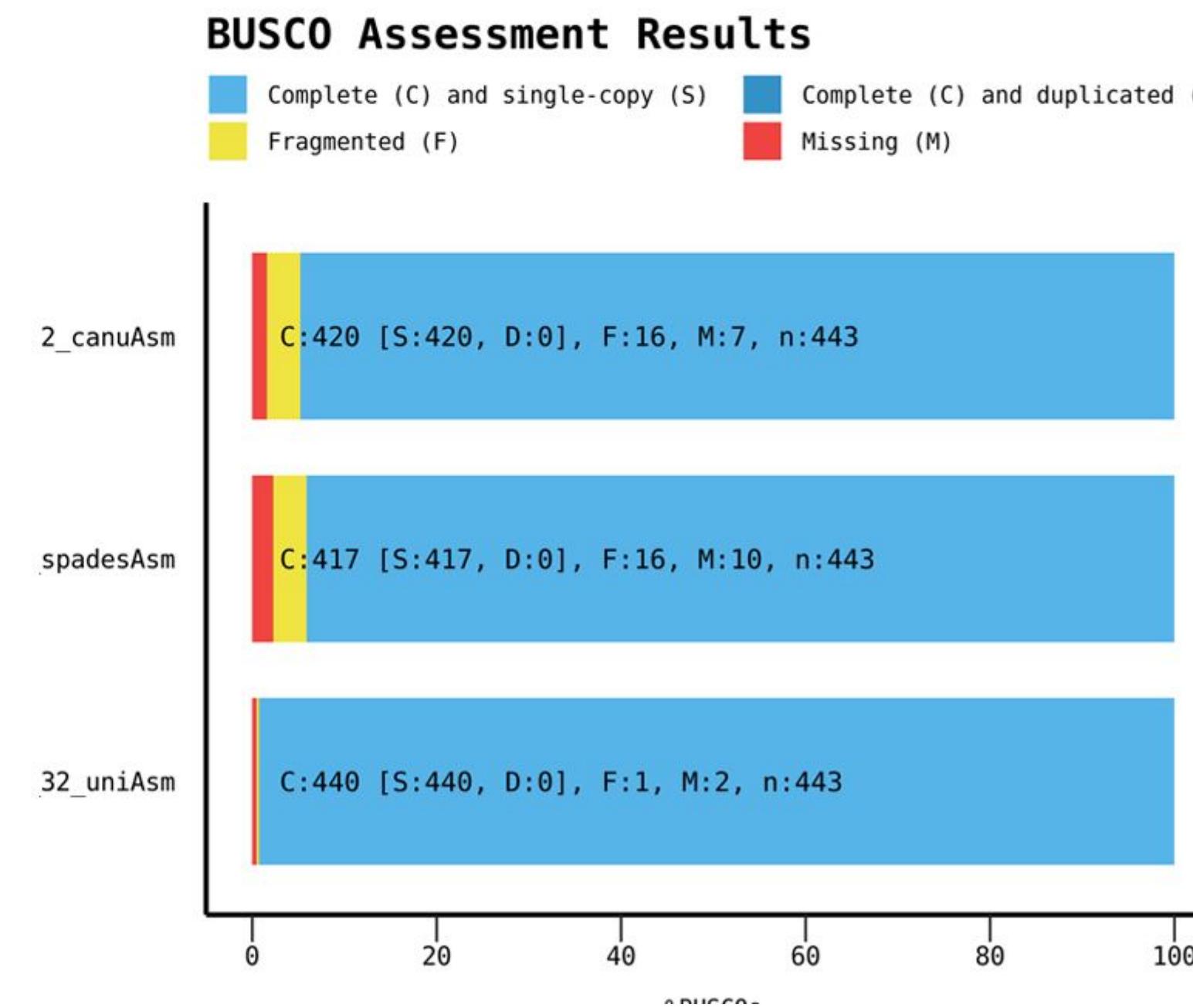
- **MUMmer**
- **Lastz**

Generates an alignment and a ***dot plot***

# BUSCO: conserved gene sets

**BUSCO:** From Evgeny Zdobnov's group,  
University of Geneva

Coverage is indicative of quality  
and completeness of assembly



# QUAST

## QQuality ASsessment Tool

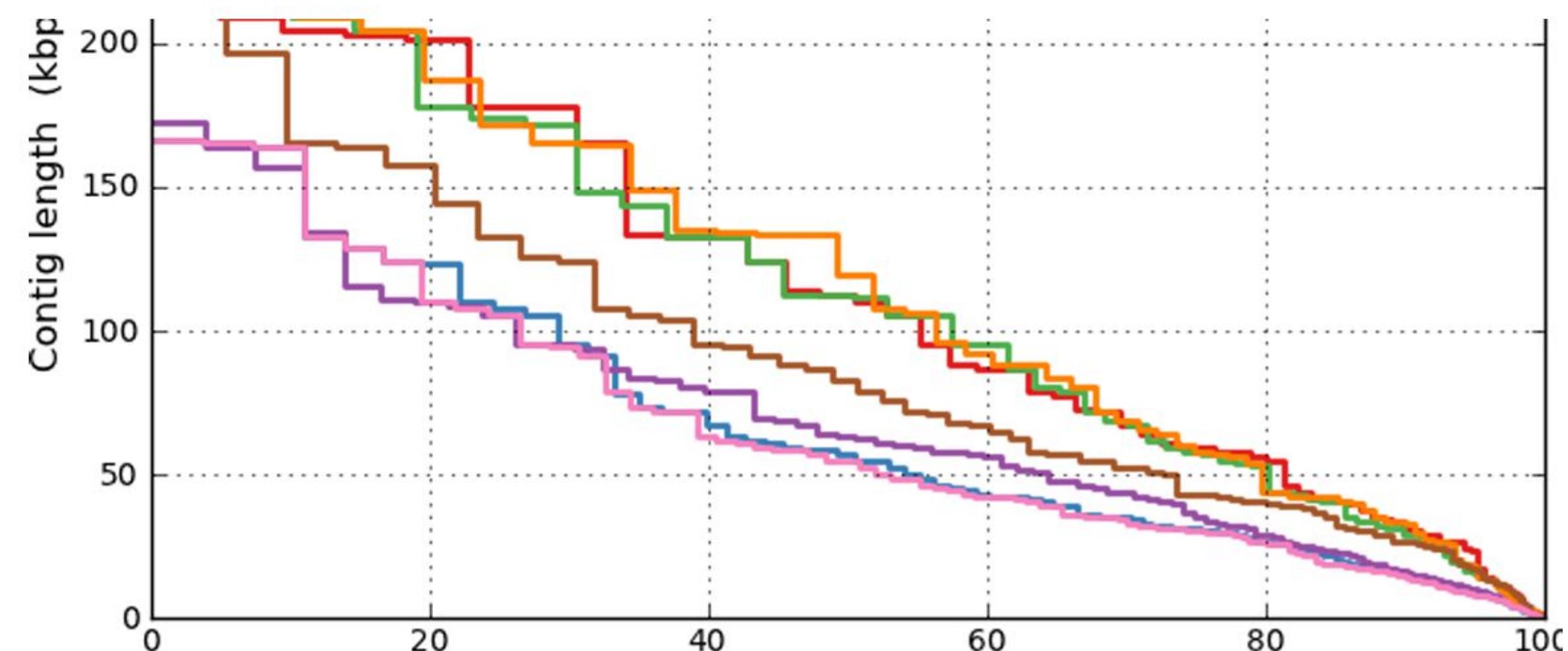
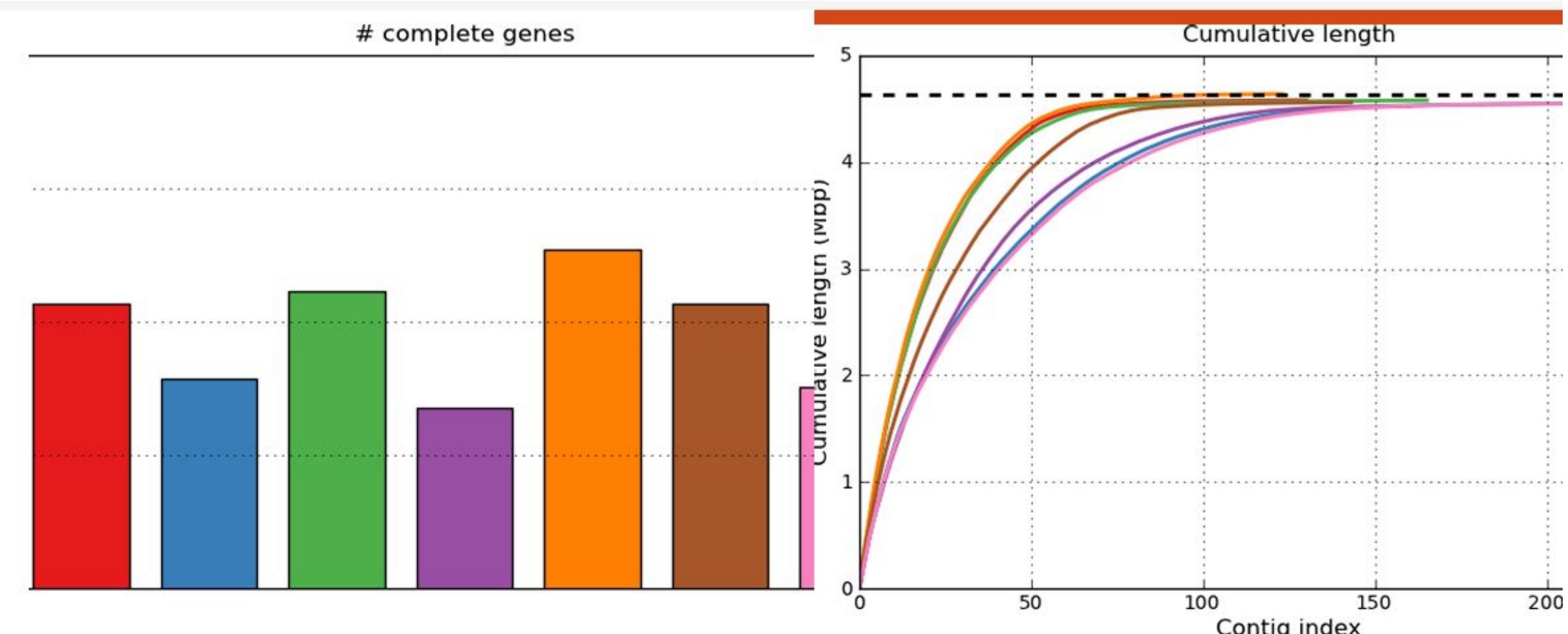
- Small (bacterial, fungal) and large (eukaryotic) genomes
- Metagenomes
- Icarus for contig alignment visualization

Can compare multiple assemblies against one another

Compare against a known (or close) reference

**Optional:** Predict genes or include annotations (checks for odd issues like frameshifts)

Generates a summary HTML report



# Assembly, variant, and pangenome graphs

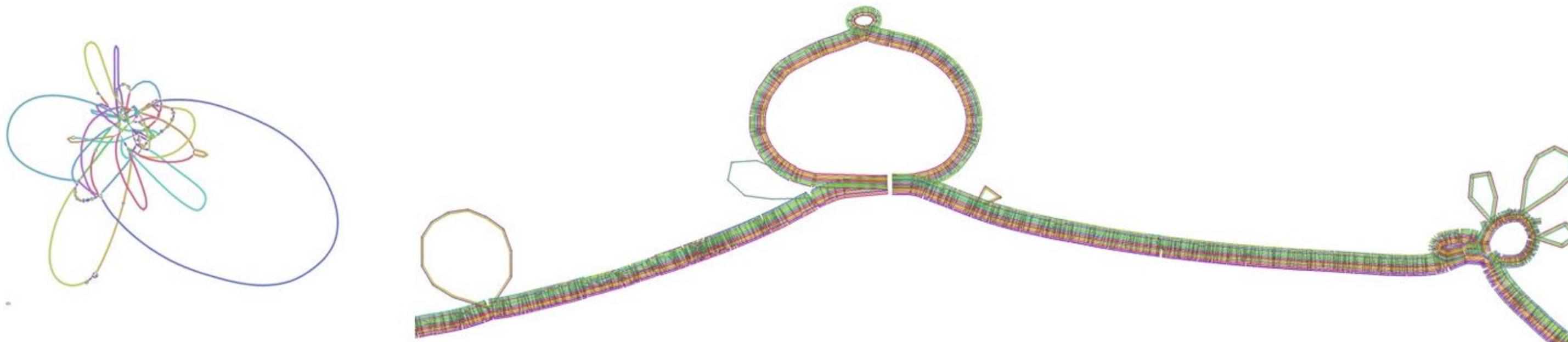
With the release of the latest human genome reference, there is more pressure to represent more data with a genome.

Current representations are mainly **haploid** (one copy)

**Assembly graphs** can retain haplotype information or raw assembly connectivity

**Variant graphs** can be generated from a reference genome and a variant file from other samples

**Pangenome graphs** capture information across populations of samples from the same species



# Резюмируем

- Задача SCS – абстрактное приближение задачи сборки генома является сложной задачей
- overlap graph позволяет реализовать жадное решение задачи поиска SCS
-