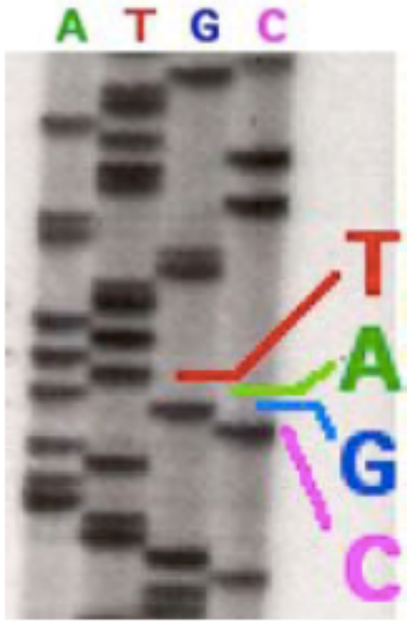
Genomics

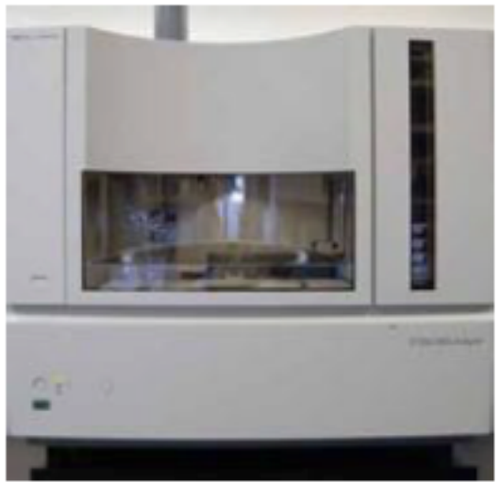
Sequencing tech



1970s: 0th Gen

Radioactive Chain Termination

5000bp / week



1980s-1990s: 1st Gen

Automated Capillary Sequencing

384kbp / day

2000s: 2nd Gen

Pyrosequencing, SOLiD Sequencing-by-Synthesis

1Gbp+ / day

Sequencing tech: next generation

Illumina HiSeq 2000
Sequencing by Synthesis

>60Gbp / day
100bp reads

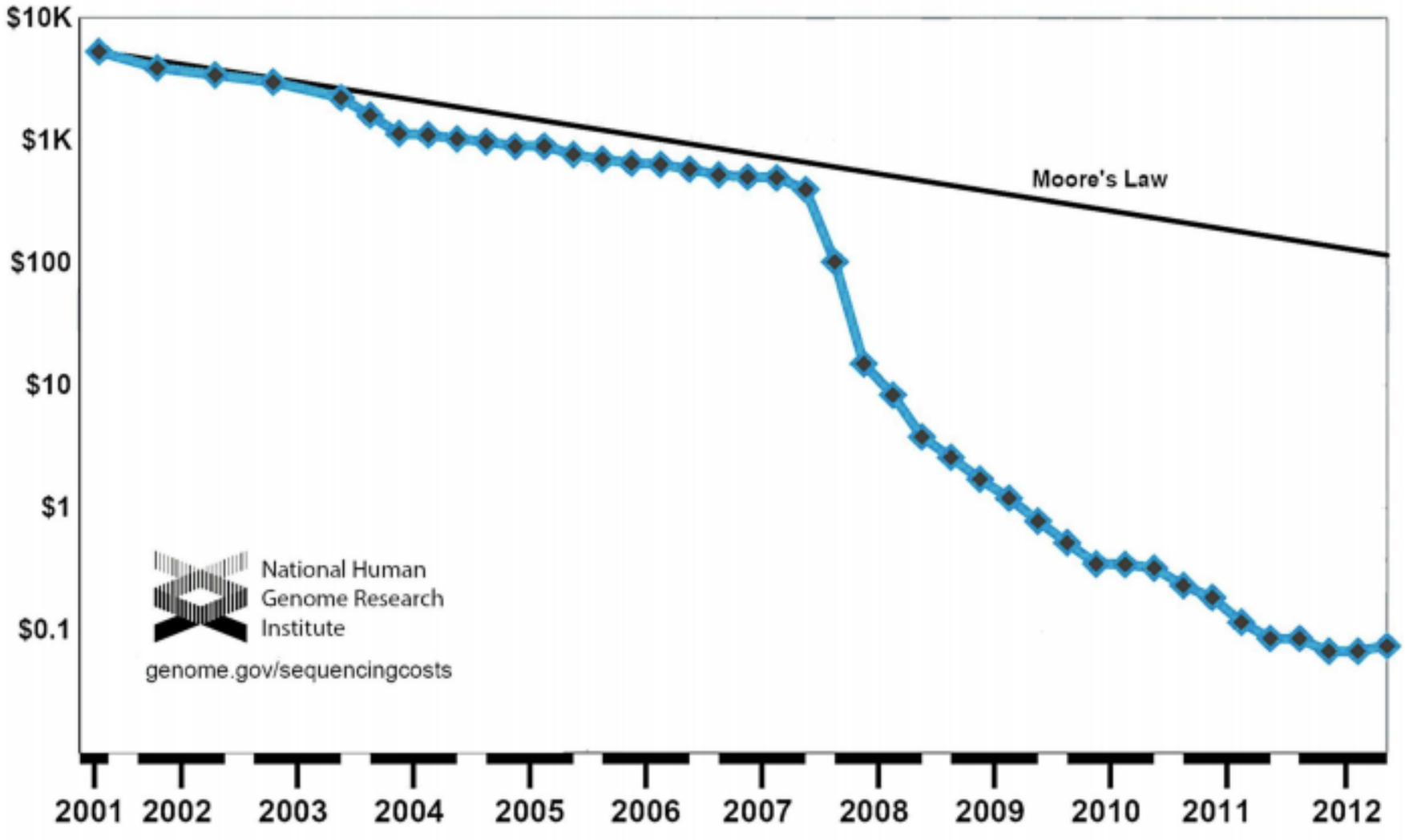
PacBio
SMRT-sequencing

~1Gbp / day
Long Reads

Oxford Nanopore
Nanopore sensing

Many GB / day?
Very Long Reads?

Cost per raw megabase of DNA sequence



Until 2007: Sanger sequencing

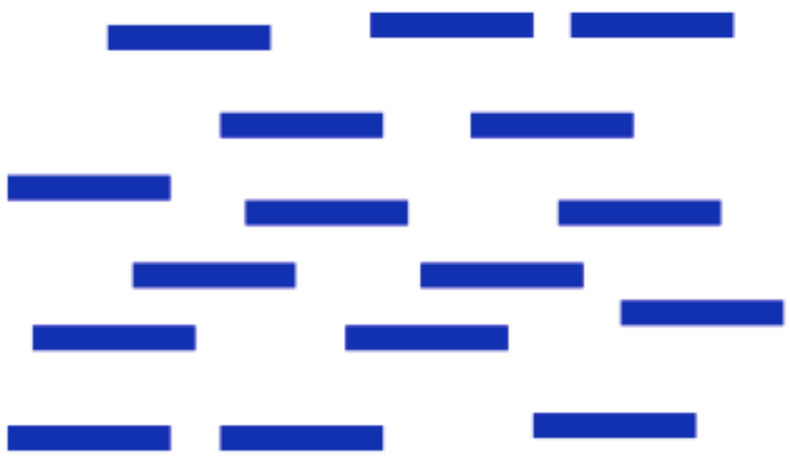
Starting in 2008: next-generation (454, Illumina, SOLiD)

What do we get from sequencing?

Sequencing technologies produce short **reads** from random locations in the DNA sample

How to analyze these reads?

Position of individual reads on the target DNA is not known



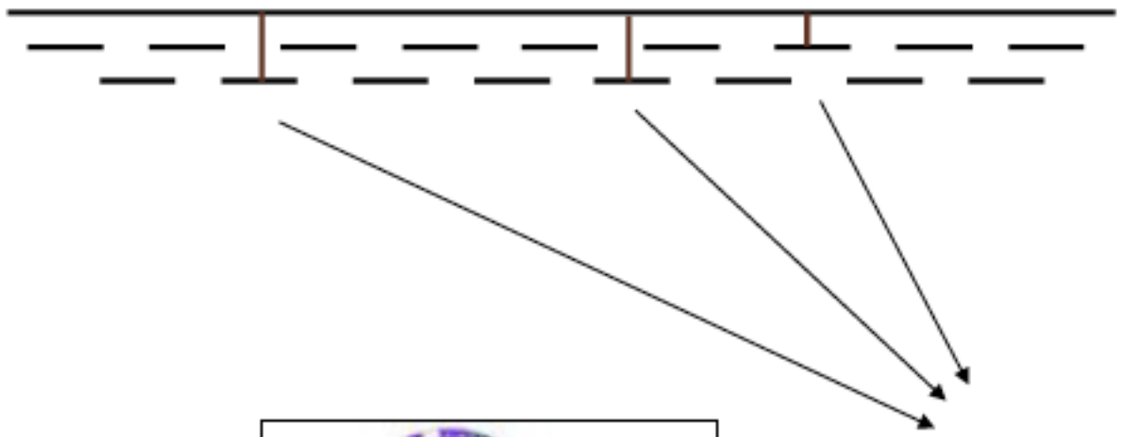
Solved by computational methods:

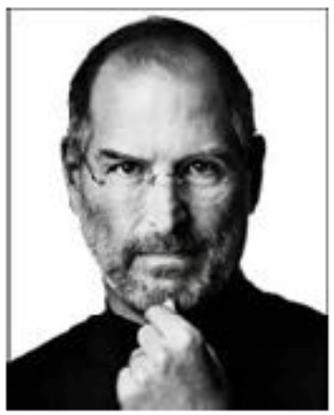
– **mapping** if target DNA is known

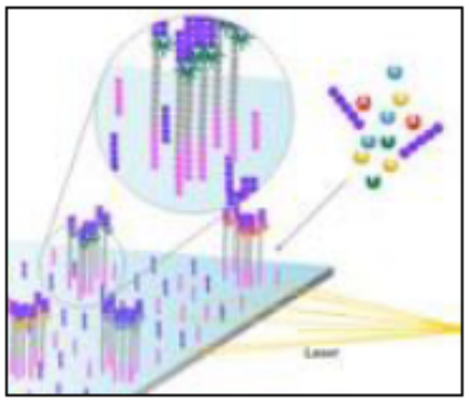
– **assembly** if it is not known

Mutation identification: Mapping

How does your genome compare to the reference?







Cancer
Heart Disease
Brain Disease

Genome projects: Assembly

1976	MS2 (RNA virus) 40 kB
1988	**Human genome sequencing project (15 years)**
1995	bacterium H. influenzae 2 MB, shotgun (TIGR)
1996	S. cerevisiae 10 MB, BAC-by-BAC (Belgium, UK)
1998	C. elegans 100 MB, BAC-by-BAC (Wellcome Trust)
1998	**Celera: human genome in three years!**
2000	D. melanogaster 180 MB, shotgun (Celera, Berkeley)
2001	**2x human genome 3 GB (NIH, Celera)**
after 2001	mouse, rat, chicken, chimpanzee, dog,. . .
2007	**Genomes of Watson and Venter (454)**

Use sequencing for other types of data

Molecular Biology

Mathematics & Computer Science

Computational Biology

Desired measurement → **reduce to sequencing** → Sequence → Solve inverse problem → Analyze

Creativity

Chemistry & Physics

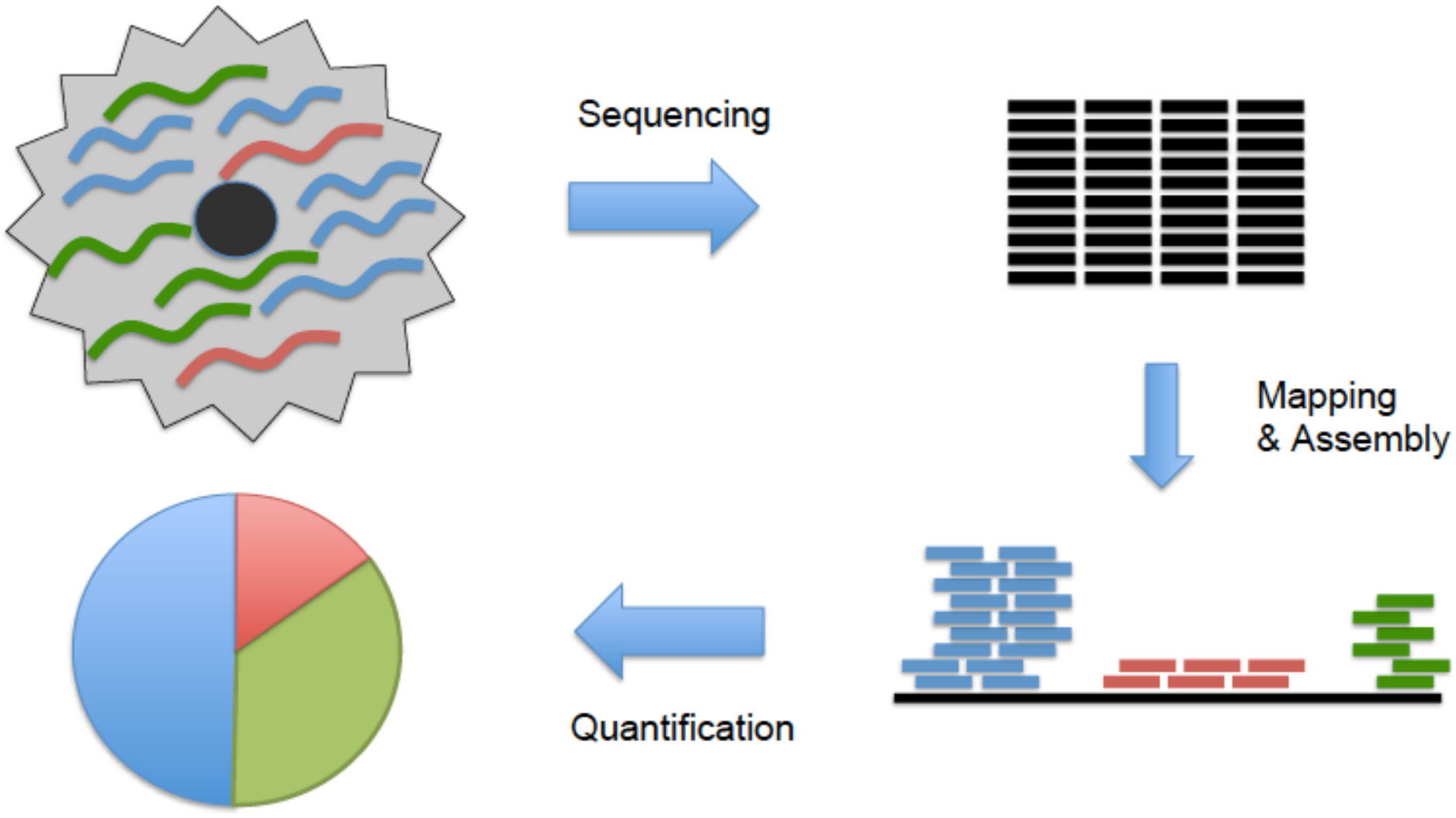
Statistics

Biology

$$\mathbb{P}(f=(p,t,l)) \approx \frac{\lambda_l \cdot \frac{\tau_t}{l(t)} \cdot w_{p|t,l} \cdot \phi_{f|p,t,l}}{\sum_{(q,r,m)\in\mathcal{A}(f)} \lambda_m \cdot \frac{\tau_r}{l(r)} \cdot w_{q|r,m} \cdot \phi_{f|q,r,m}}$$

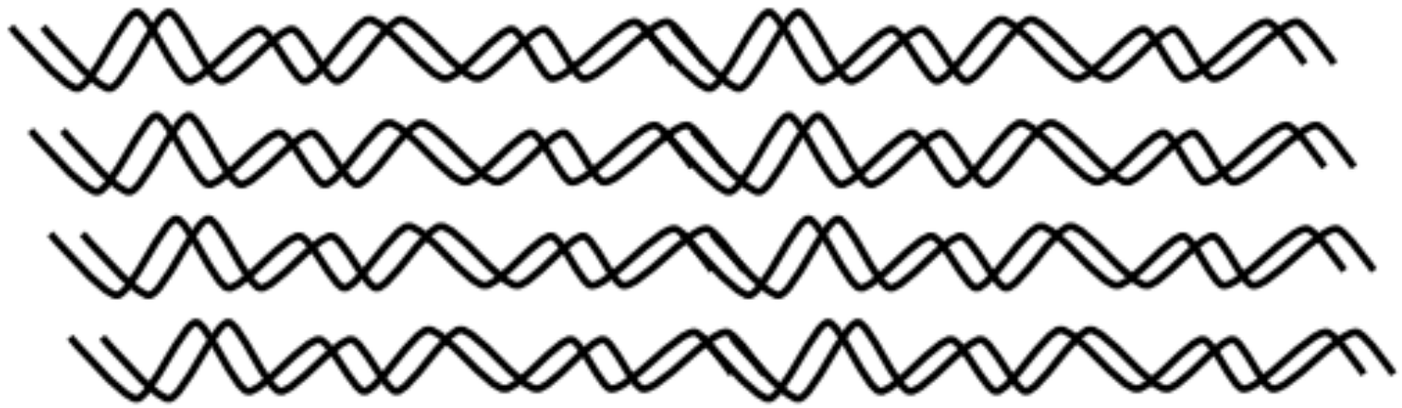
X-seq technology

RNA-seq

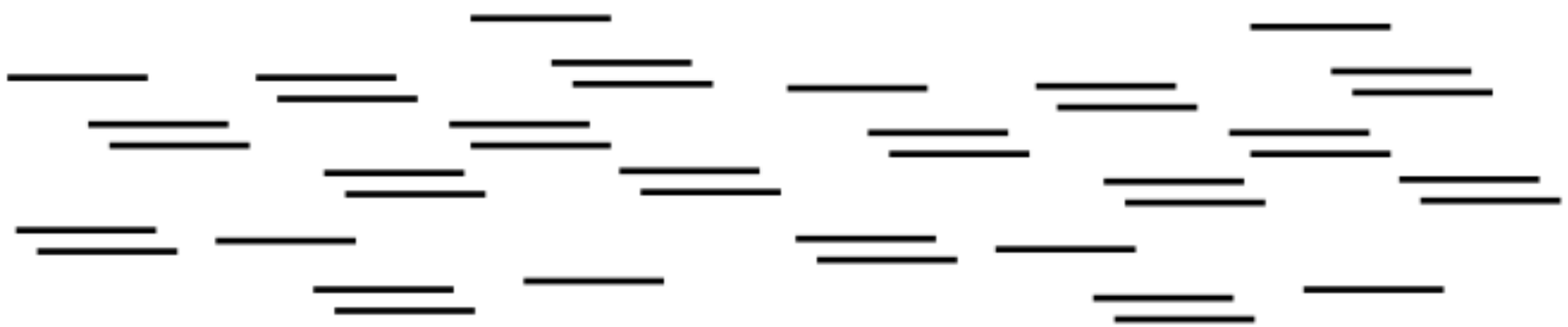


Assembly

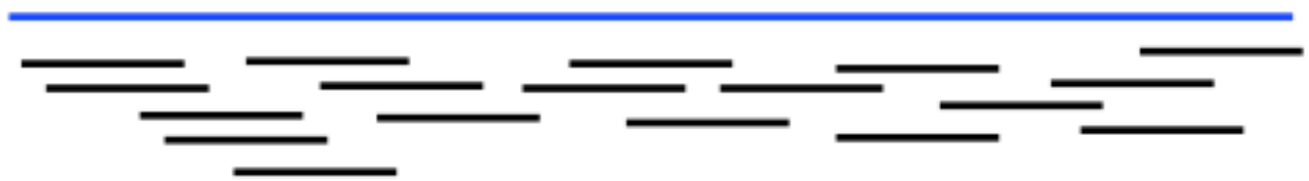
Many copies of the DNA



Shear it, randomly breaking them into many small pieces, read ends of each:



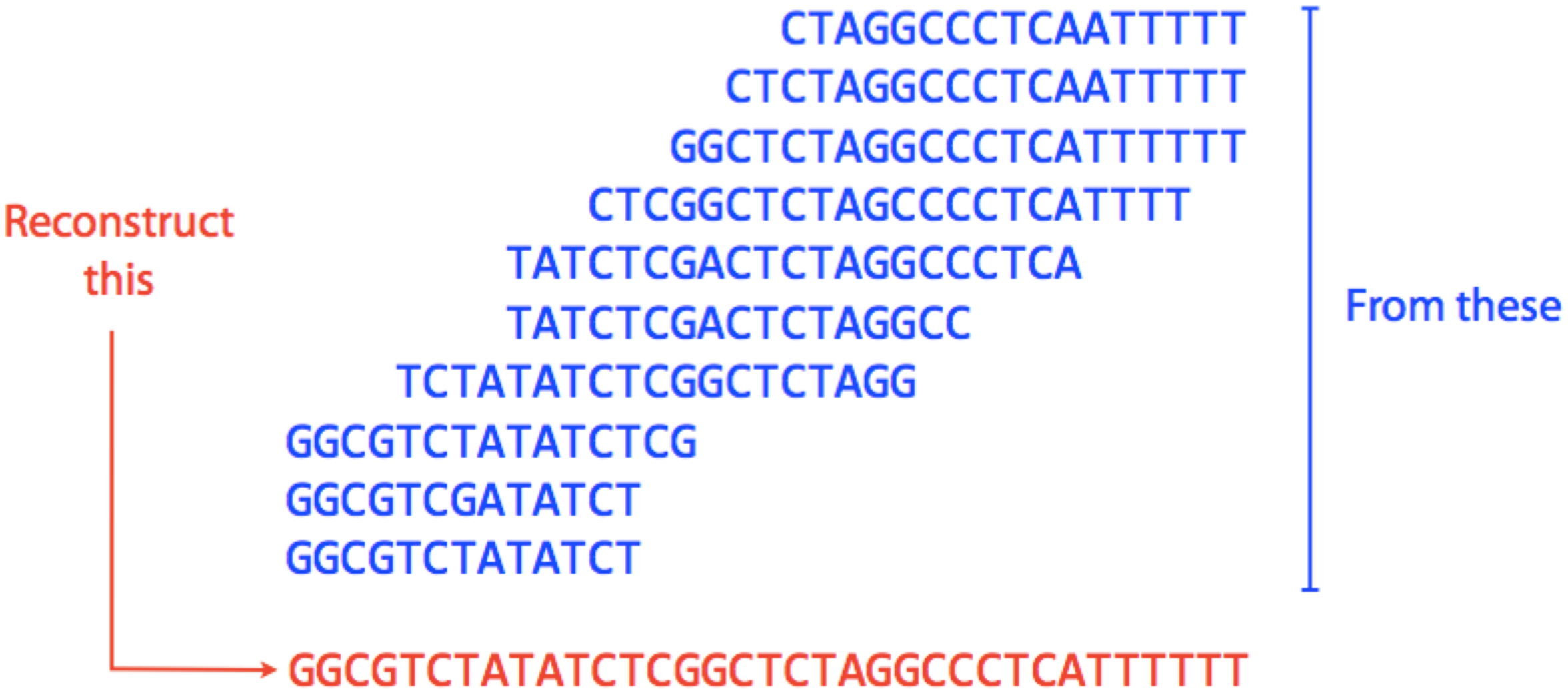
Assemble into original genome:



* Slide from Carl Kingsford

Assembly

<u>Computational Challenge</u>: assemble individual short fragments (reads) into a single genomic sequence ("superstring")



Shortest common superstring

Problem: Given a set of strings, find a shortest string that contains all of them

Input: Strings $s_1, s_2, \ldots, s_n$

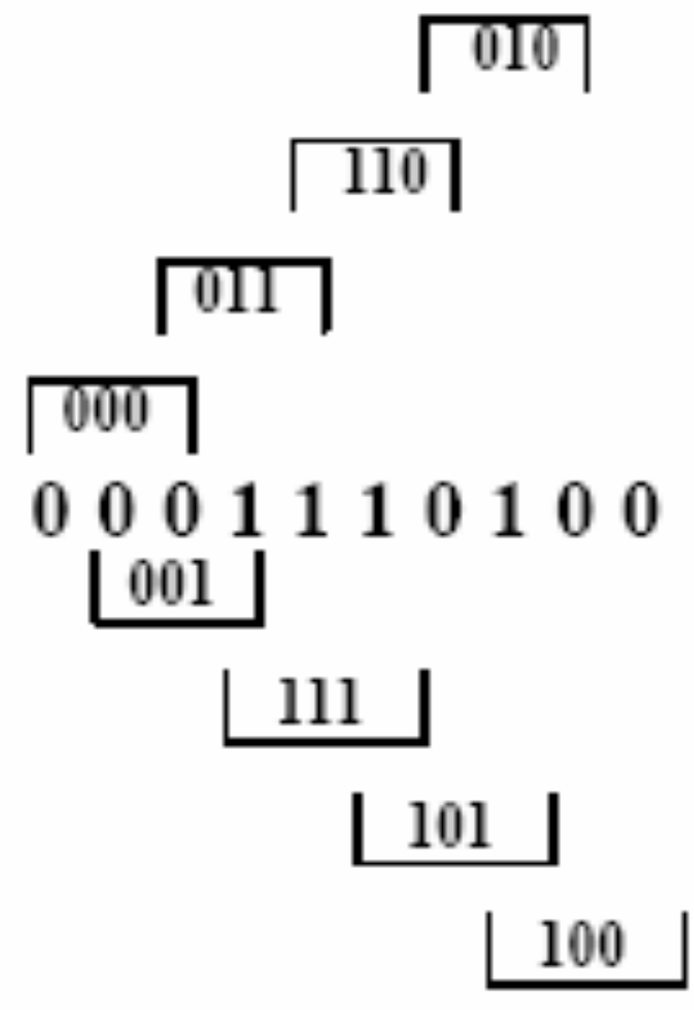
Output: A string s that contains all strings $s_1, s_2, \ldots, s_n$ as substrings, such that the length of s is minimized

Shortest common superstring

Set of strings: {000, 001, 010, 011, 100, 101, 110, 111}

Concatenation Superstring 000 001 010 011 100 101 110 111

Shortest superstring



Any ideas?

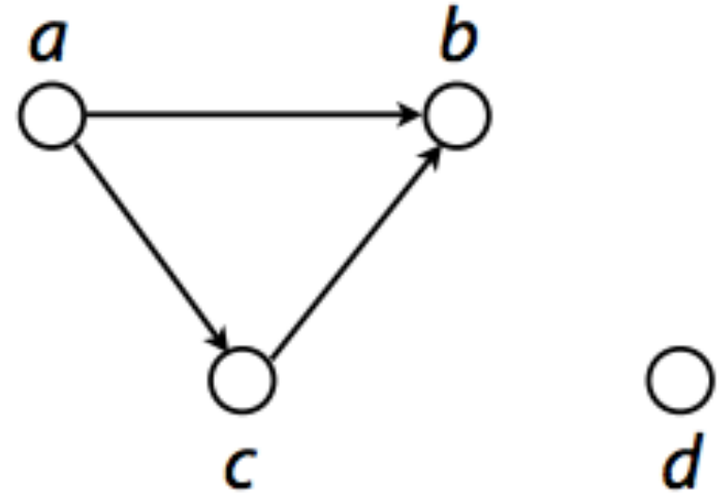
Directed Graph

Directed graph $G(V, E)$ consists of set of *vertices*, V and set of *directed edges*, E

Directed edge is an *ordered pair* of vertices.
First is the *source*, second is the *sink*.

Vertex is drawn as a circle

Edge is drawn as a line with an arrow connecting two circles



Vertex also called *node* or *point*

Edge also called *arc* or *line*

Directed graph also called *digraph*

$V = \{ a, b, c, d \}$

$E = \{ (a, b), (a, c), (c, b) \}$

Source Sink

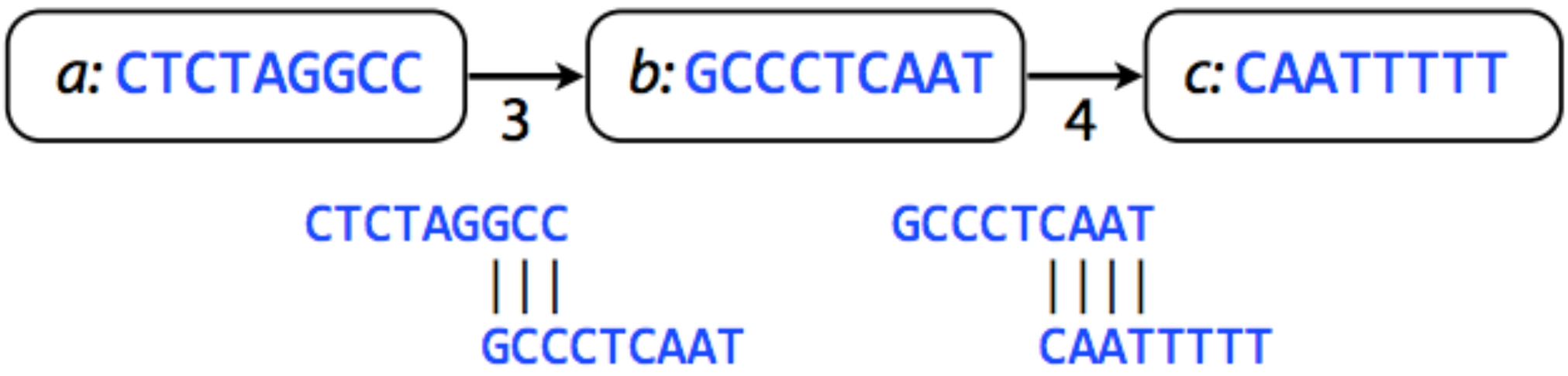
Overlap Graph

Below: overlap graph, where an overlap is a suffix/prefix match of at least 3 characters

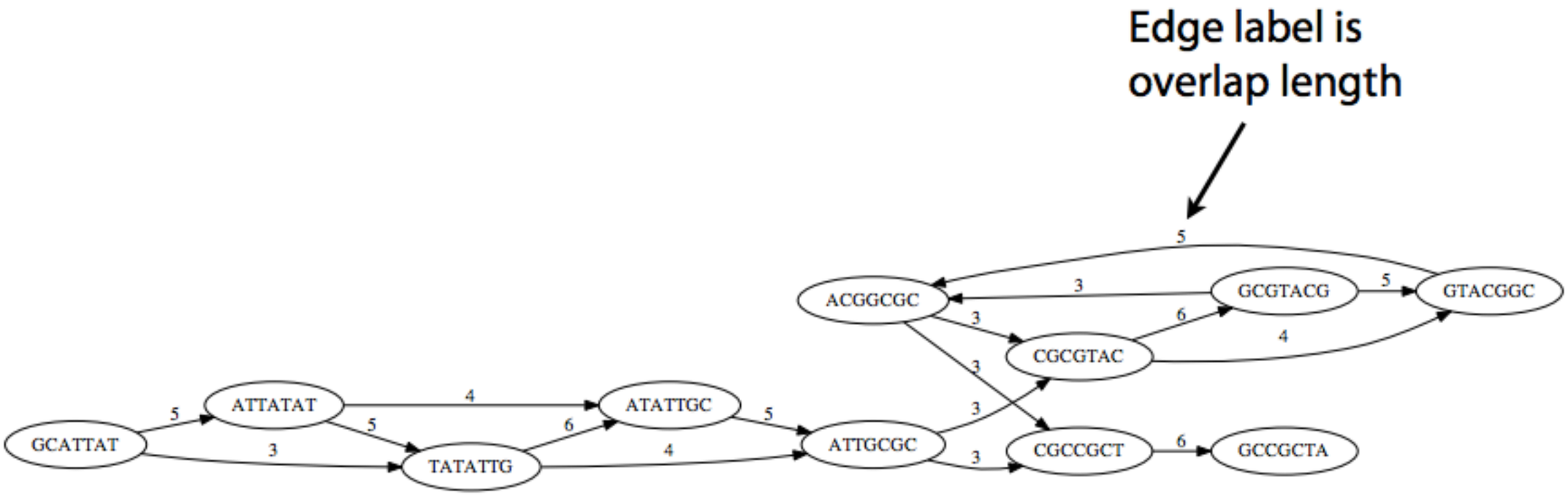
A vertex is a read, a directed edge is an overlap between suffix of source and prefix of sink

Vertices (reads): { *a:* CTCTAGGCC, *b:* GCCCTCAAT, *c:* CAATTTTT }

Edges (overlaps): { (*a*, *b*), (*b*, *c*) }



Example



Original string: GCATTATATATTGCGCGTACGGCGCCGCTACA

Shortest common superstring problem is hard

Can we solve it?

Imagine a modified overlap graph where each edge has cost = - (length of overlap)

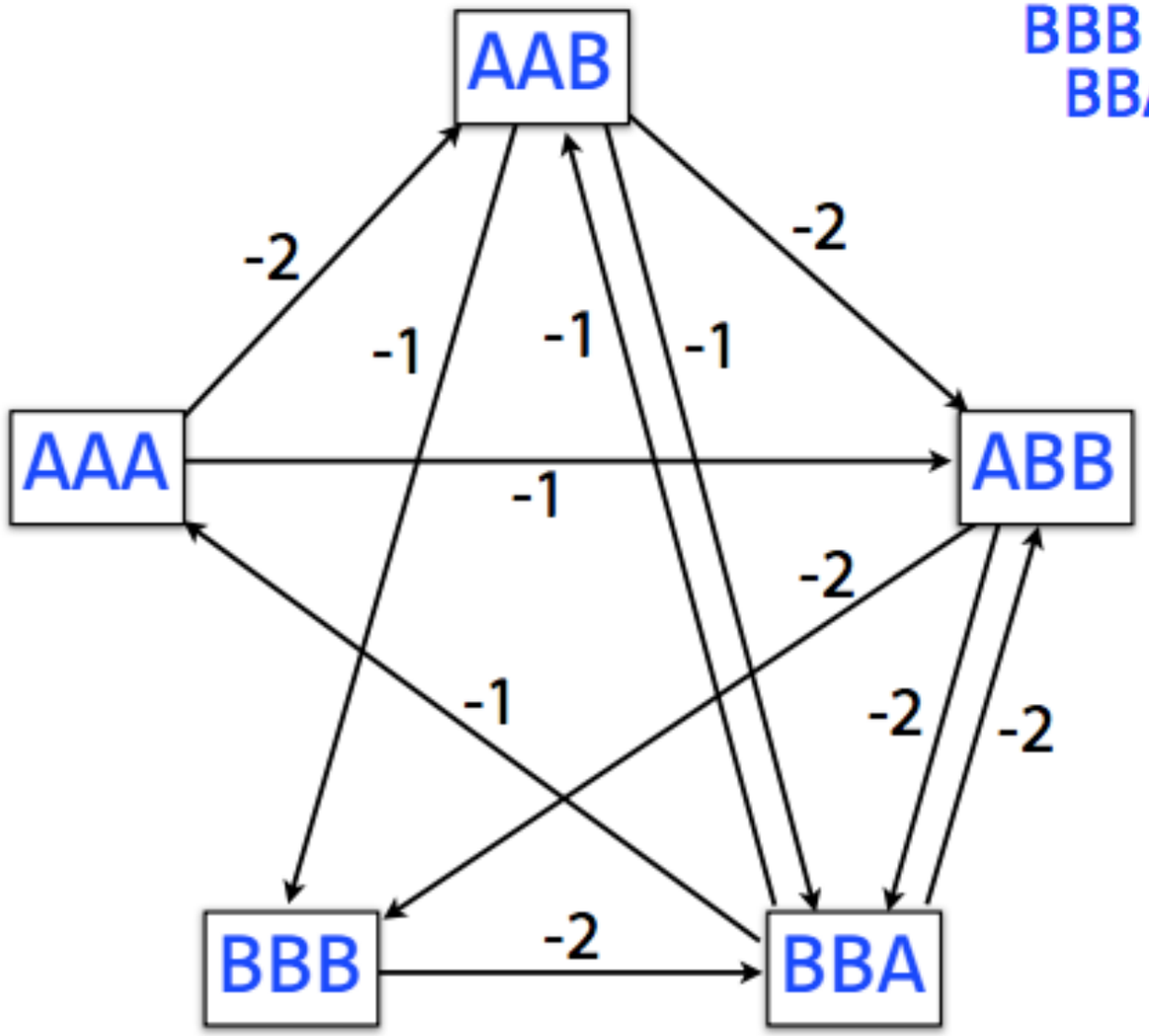
SCS corresponds to a path that visits every node once, minimizing total cost along path

That's the *Traveling Salesman Problem* (*TSP*), which is NP-hard!

S: AAA AAB ABB BBB BBA

SCS(S): AAABBBA

AAA
AAB
ABB
BBB
BBA



Shortest common superstring problem is hard

Say we disregard edge weights and just look for a path that visits all the nodes exactly once

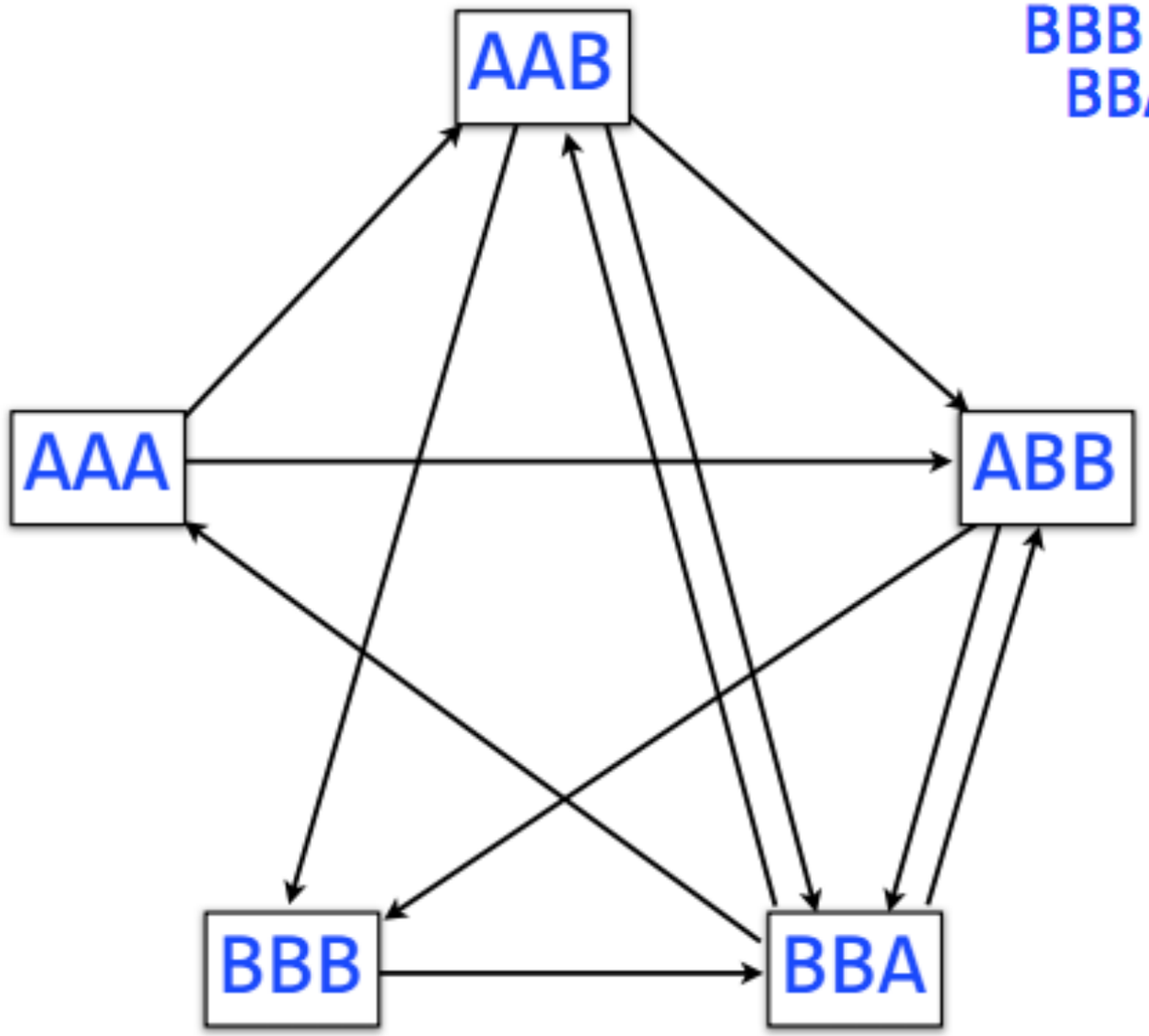
That's the *Hamiltonian Path* problem: NP-complete

Indeed, it's well established that SCS is NP-hard

S: AAA AAB ABB BBB BBA

SCS(S): AAABBBA

AAA
AAB
ABB
BBB
BBA



Is there a better or more feasible way?

Matching a superstring to a set of short reads

Assume we have a set S of reads with length k (k-mers)

Goal: Find a string that can be exactly split in to set S.

$$S = \{\ \mathrm{ATG}\ \ \mathrm{AGG}\ \ \mathrm{TGC}\ \ \mathrm{TCC}\ \ \mathrm{GTC}\ \ \mathrm{GGT}\ \ \mathrm{GCA}\ \ \mathrm{CAG}\ \}$$

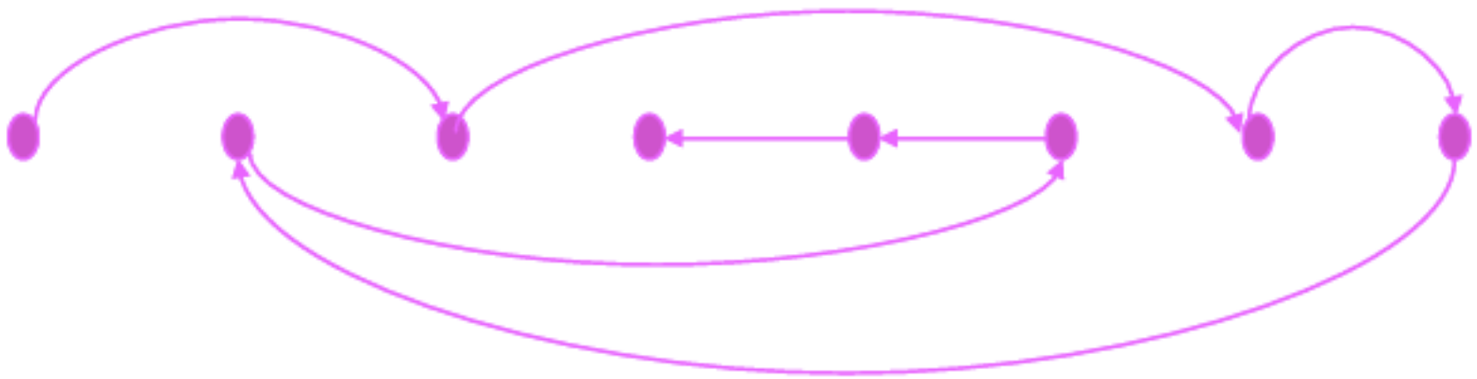
Overlap graph approach

Assume we have a set S of reads with length k (k-mers)

Goal: Find a string that can be exactly split in to set S.

$$S = \{\ \text{ATG}\ \ \text{AGG}\ \ \text{TGC}\ \ \text{TCC}\ \ \text{GTC}\ \ \text{GGT}\ \ \text{GCA}\ \ \text{CAG}\ \}$$

H ATG AGG TGC TCC GTC GGT GCA CAG



ATG C A G G T C C

Path visited every VERTEX once

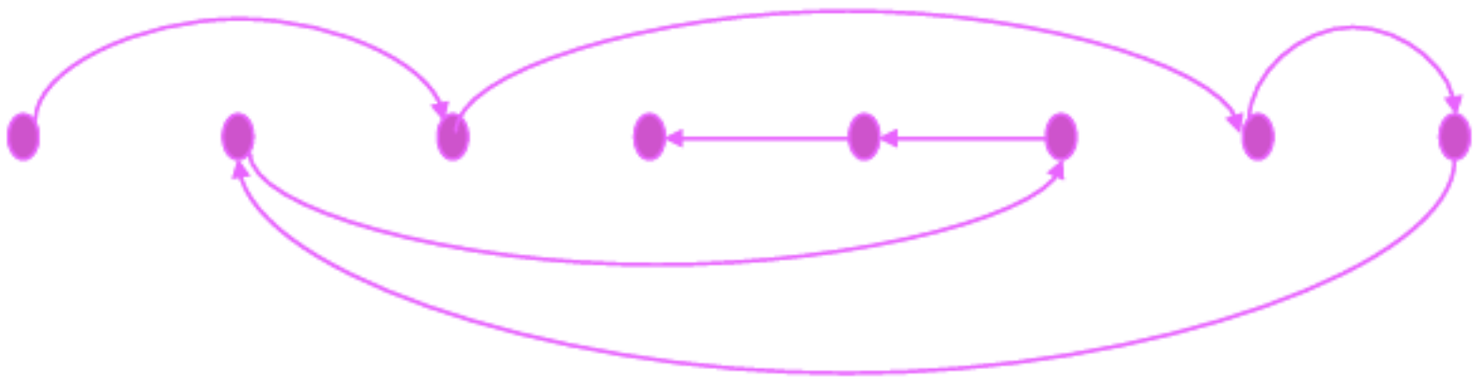
Overlap graph approach is hard

Assume we have a set S of reads with length k (k-mers)

Goal: Find a string that can be exactly split in to set S.

$$S = \{\ \text{ATG}\ \ \text{AGG}\ \ \text{TGC}\ \ \text{TCC}\ \ \text{GTC}\ \ \text{GGT}\ \ \text{GCA}\ \ \text{CAG}\ \}$$

H ATG AGG TGC TCC GTC GGT GCA CAG



ATG C A G G T C C

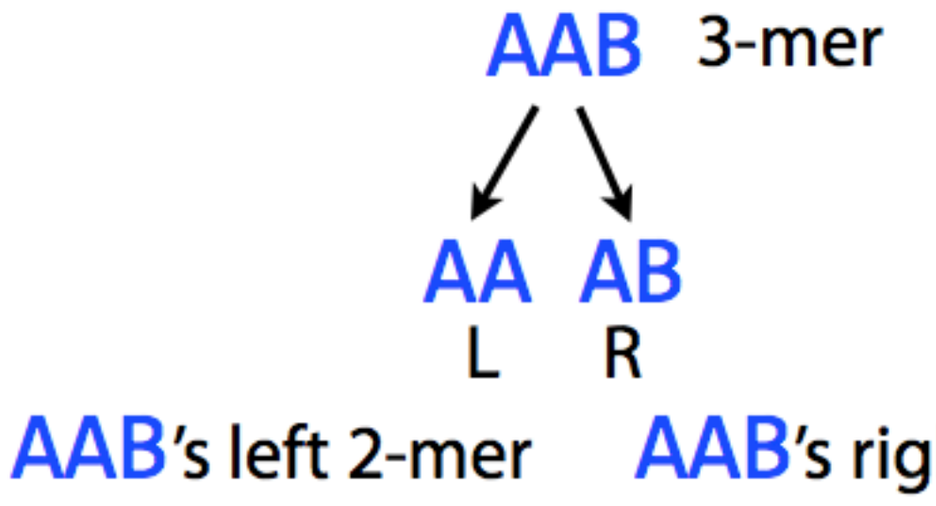
Path visited every VERTEX once

There is an alternative way

As usual, we start with a collection of reads, which are substrings of the reference genome.

AAA, AAB, ABB, BBB, BBA

AAB is a *k*-mer (*k* = 3). AA is its *left* *k*-1-mer, and AB is its right *k*-1-mer.



De Bruijn Graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

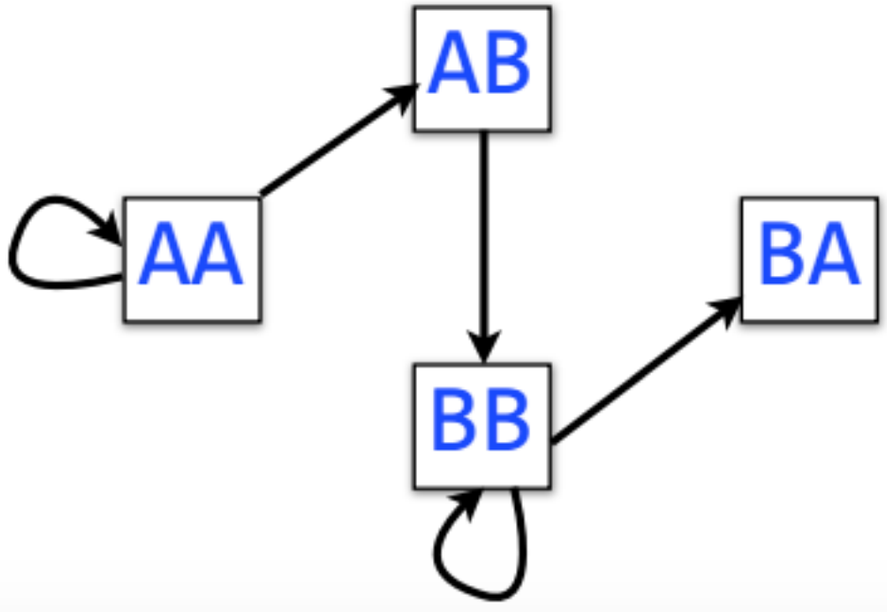
AAABBBA

take all 3-mers: AAA, AAB, ABB, BBB, BBA

form L/R 2-mers: AA, AA, AA, AB, AB, BB, BB, BB, BB, BA

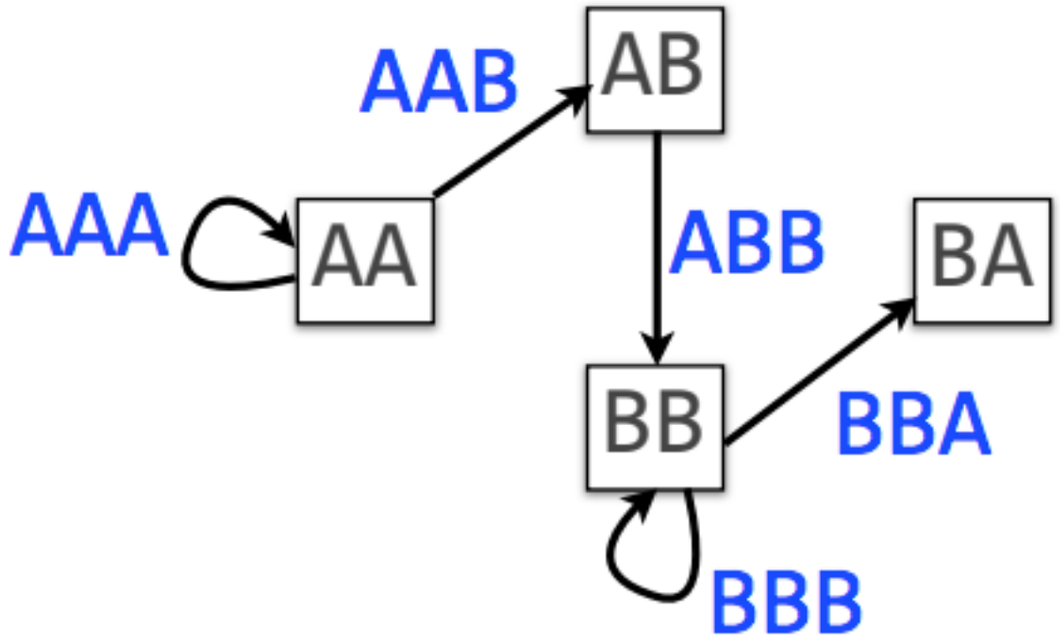
L R L R L R L R L R

Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:



Each *edge* in this graph corresponds to a length-3 input string

De Bruijn Graph

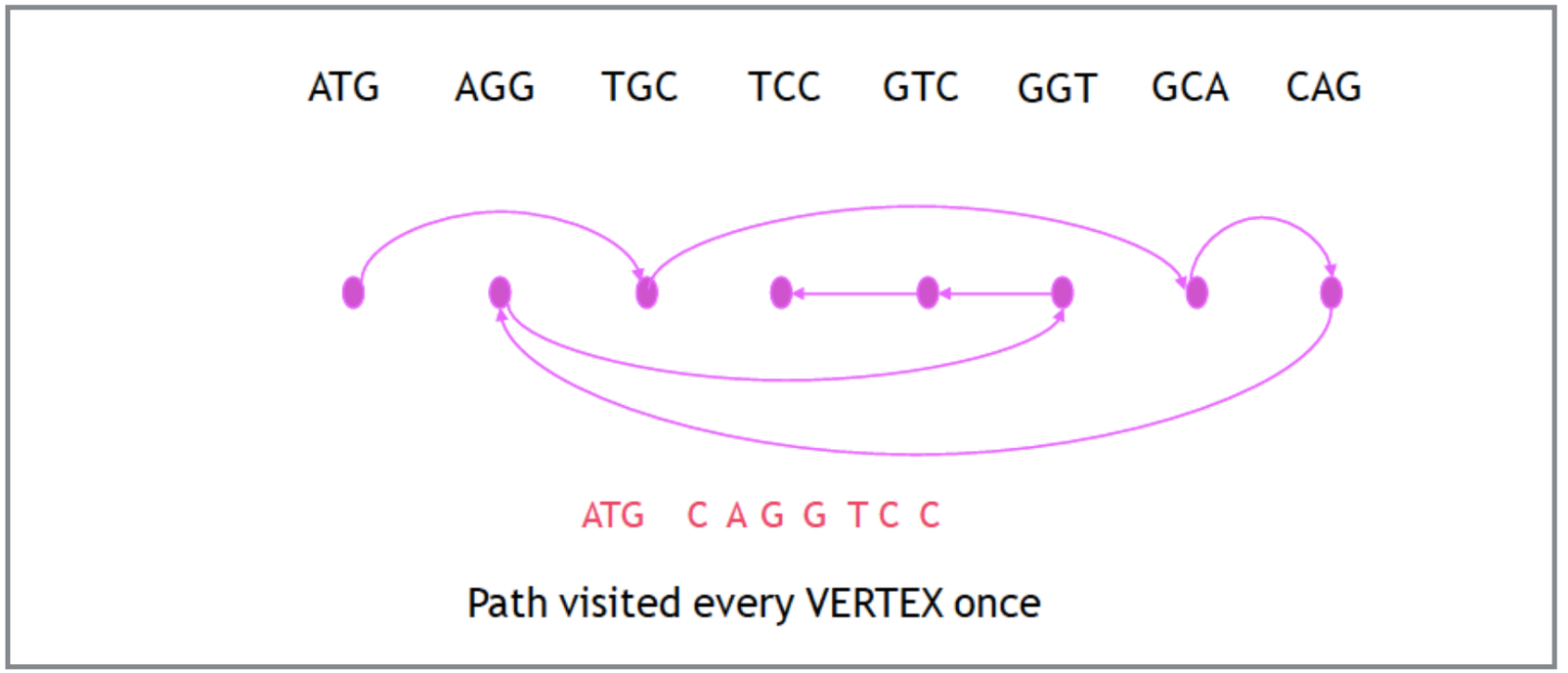


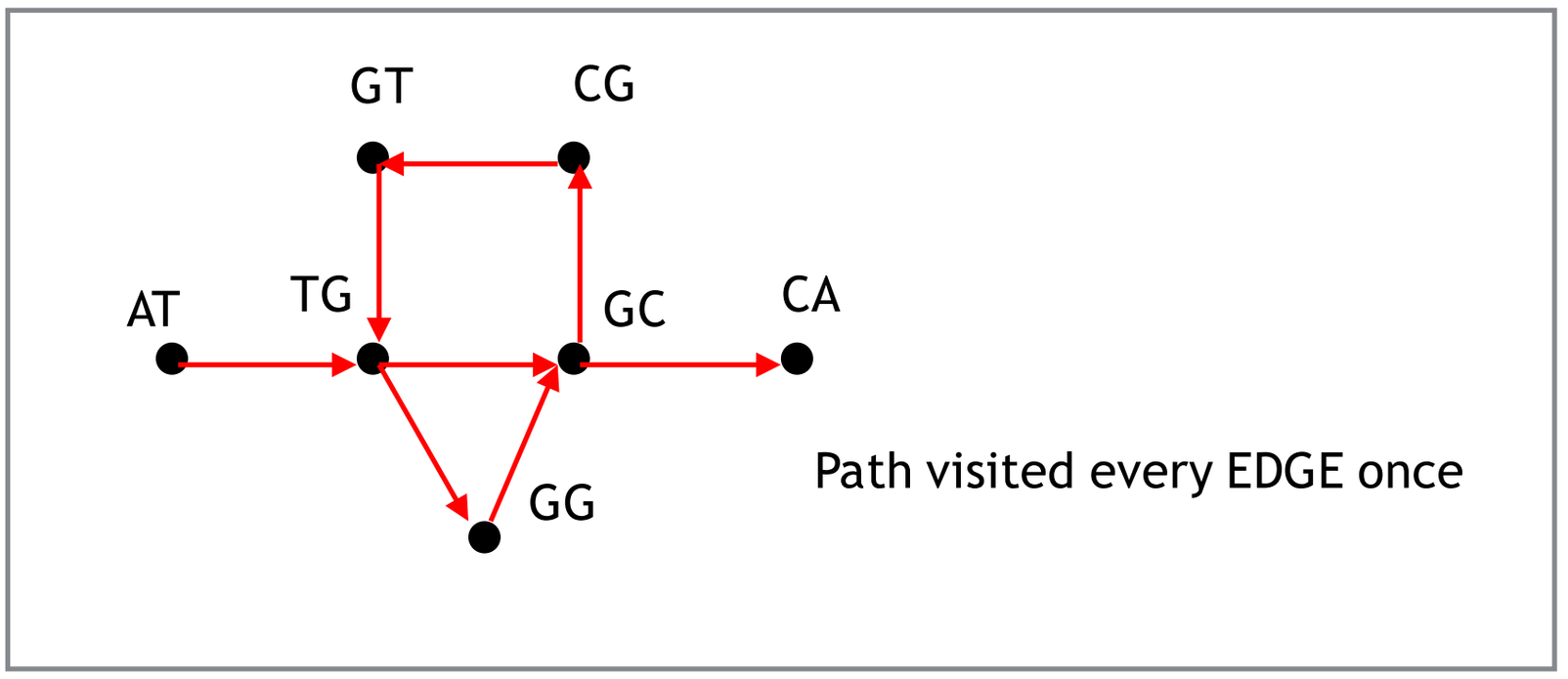
An edge corresponds to an overlap (of length *k*-2) between two *k*-1 mers.
More precisely, it corresponds to a *k*-mer from the input.

What is the goal now?

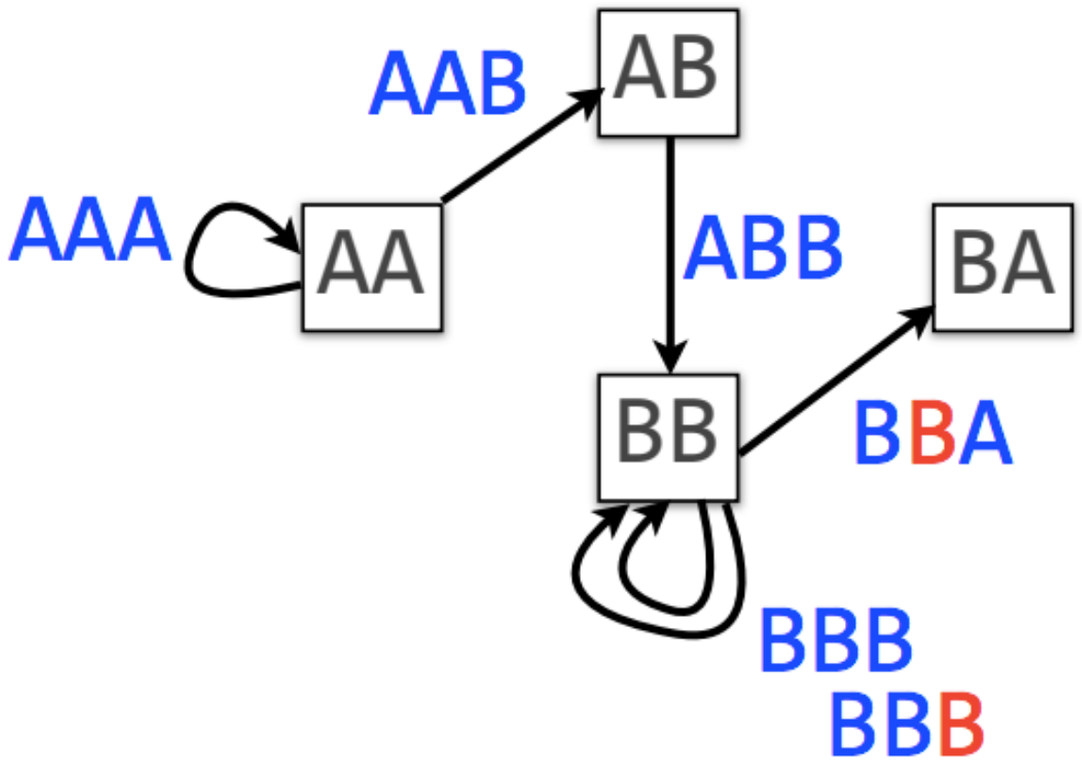
Overlap graph vs De Bruijn graph

$S = \{ ATG \;\; AGG \;\; TGC \;\; TCC \;\; GTC \;\; GGT \;\; GCA \;\; CAG \}$





MultiEdge



If we add one more B to our input string: AAABBBBA, and rebuild the De Bruijn graph accordingly, we get a *multiedge*.

MultiGraph

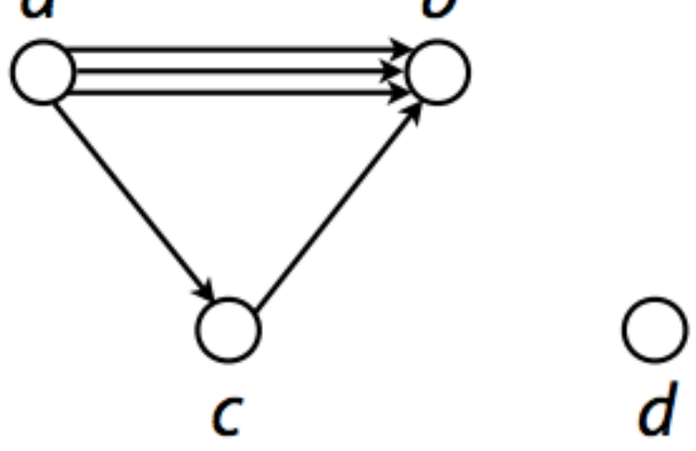
Directed **multigraph** $G(V, E)$ consists of set of *vertices*, V and **multiset** of *directed edges*, E

Otherwise, like a directed graph

Node's *indegree* = # incoming edges

Node's *outdegree* = # outgoing edges

De Bruijn graph is a directed multigraph



$V = \{ a, b, c, d \}$

$E = \{ (a, b), (a, b), (a, b), (a, c), (c, b) \}$

Repeated

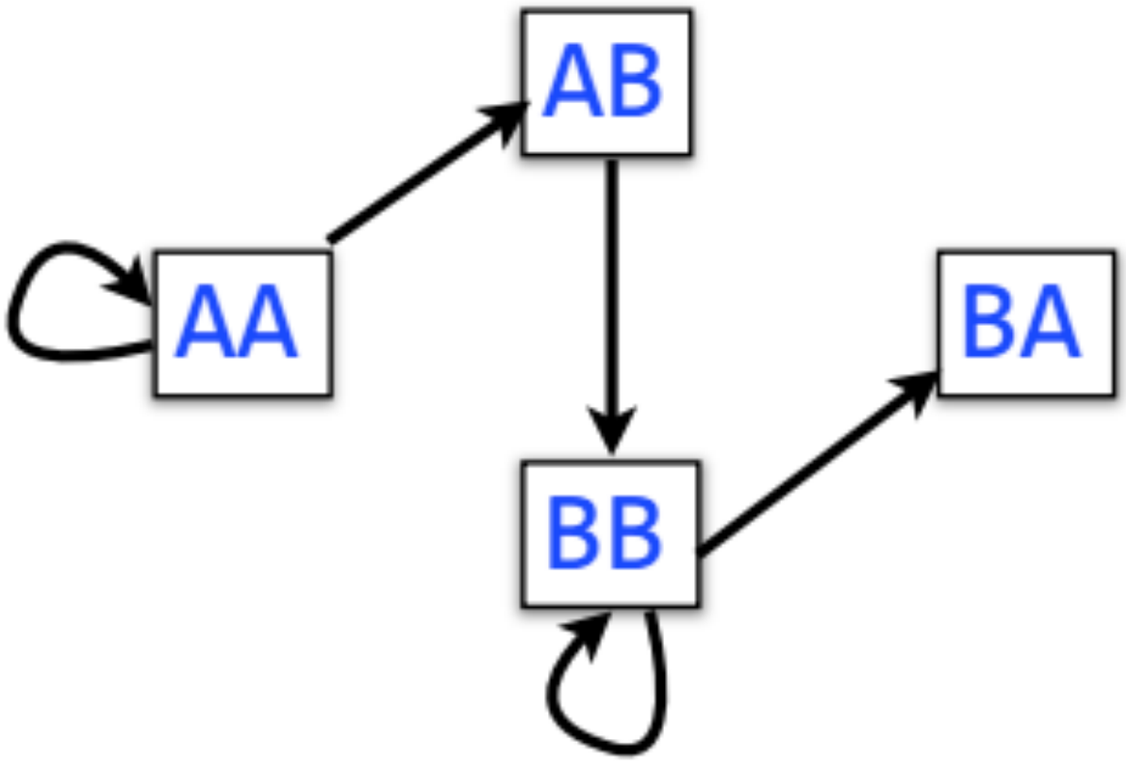
Some definitions

Node is *balanced* if indegree equals outdegree

Node is *semi-balanced* if indegree differs from outdegree by 1

Graph is *connected* if each node can be reached by some other node

Eulerian walk visits each edge exactly once

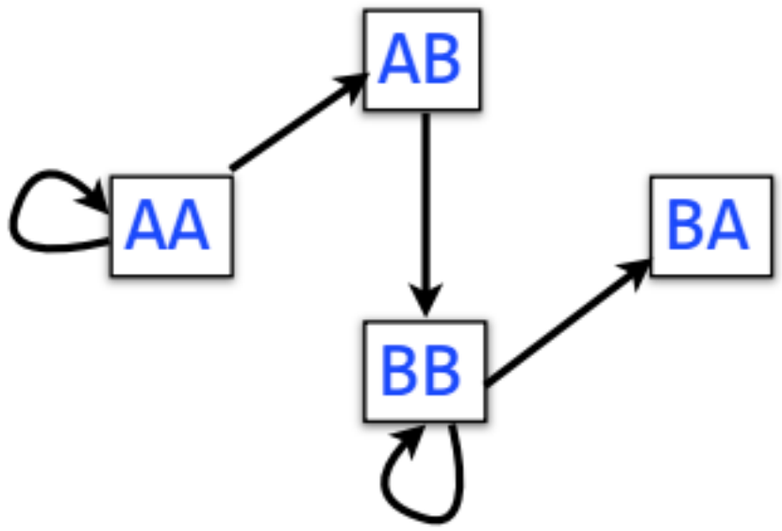


Eulerian walk/path

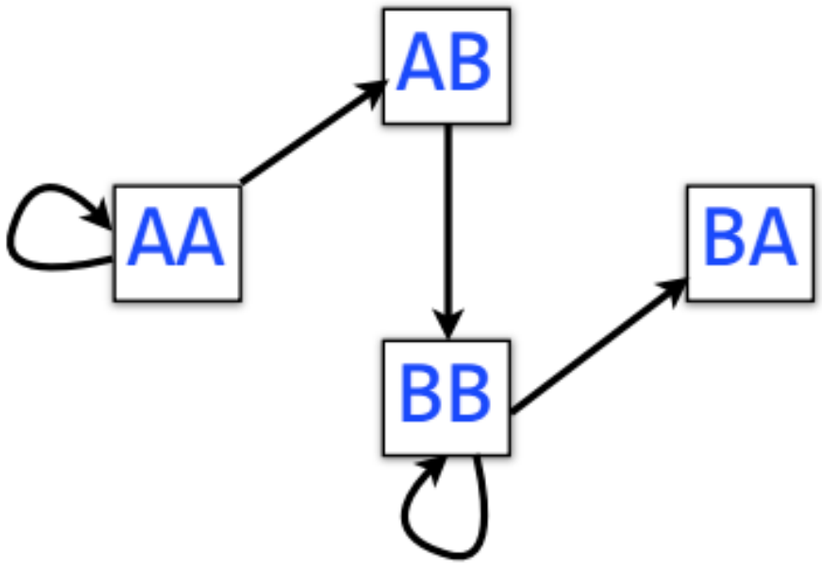
Eulerian walk visits each edge exactly once

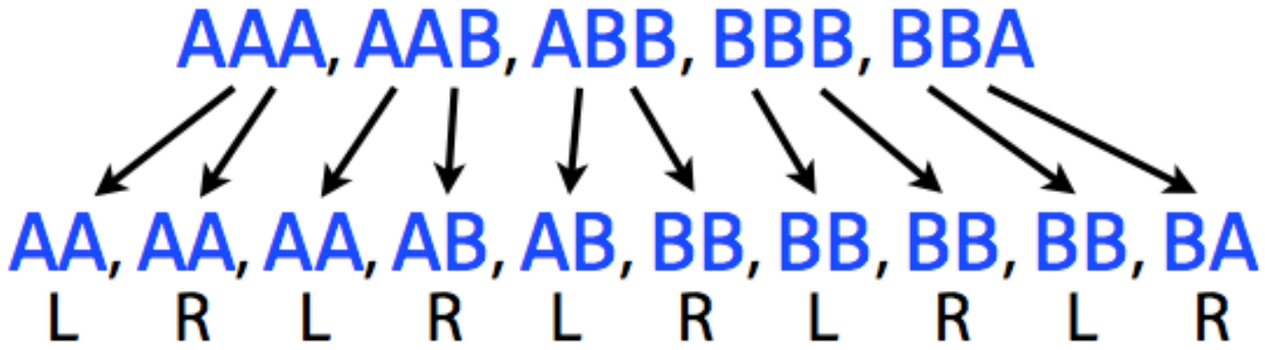
Not all graphs have Eulerian walks. Graphs that do are *Eulerian*.
(For simplicity, we won't distinguish Eulerian from semi-Eulerian.)

A directed, connected graph is Eulerian if and only if it has **zero or** 2 semi-balanced nodes and all other nodes are balanced



Eulerian walk/path





Is it Eulerian? Yes

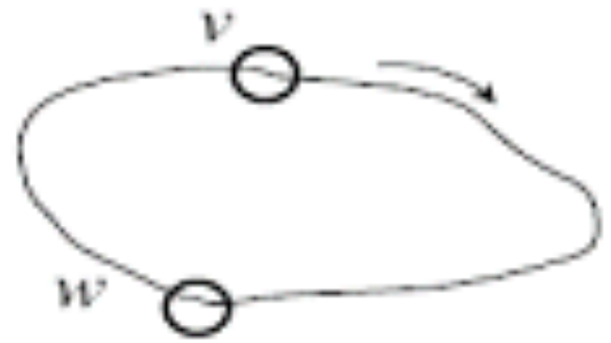
Argument 1: AA → AA → AB → BB → BB → BA

Argument 2: AA and BA are semi-balanced, AB and BB are balanced

Proof? Algorithm?

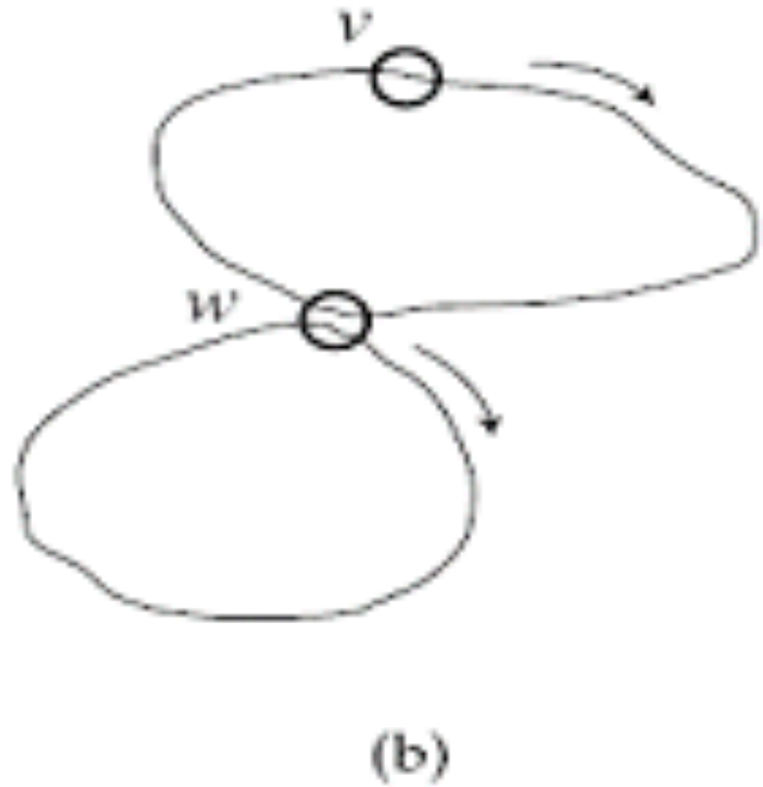
Assume all nodes are balanced

a. Start with an arbitrary vertex *v* and form an arbitrary cycle with unused edges until a dead end is reached. Since the graph is Eulerian this dead end is necessarily the starting point, i.e., vertex *v*.



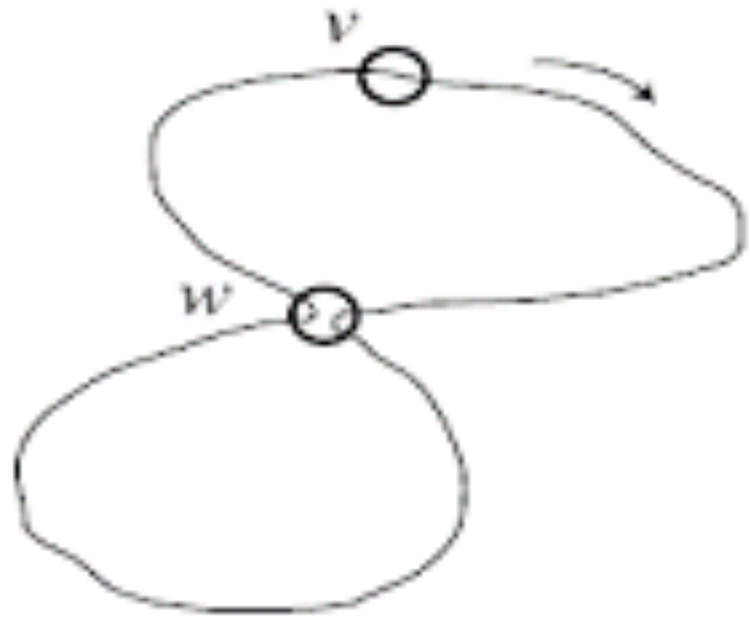
(a)

b. If cycle from (a) is not an Eulerian cycle, it must contain a vertex *w*, which has untraversed edges. Perform step (a) again, using vertex *w* as the starting point. Once again, we will end up in the starting vertex *w*.



(b)

c. Combine the cycles from (a) and (b) into a single cycle and iterate step (b).



(c)

Eulerian path

- A vertex *v* is "semibalanced" if
  | in-degree(*v*) - out-degree(*v*)| = 1

- If a graph has an Eulerian *path* starting from *s* and ending at *t*, then all its vertices are balanced *with the possible exception of s and t*

- Add an edge between two semibalanced vertices: now all vertices should be balanced (assuming there was an Eulerian path to begin with). Find the Eulerian cycle, and remove the edge you had added. You now have the Eulerian path you wanted.