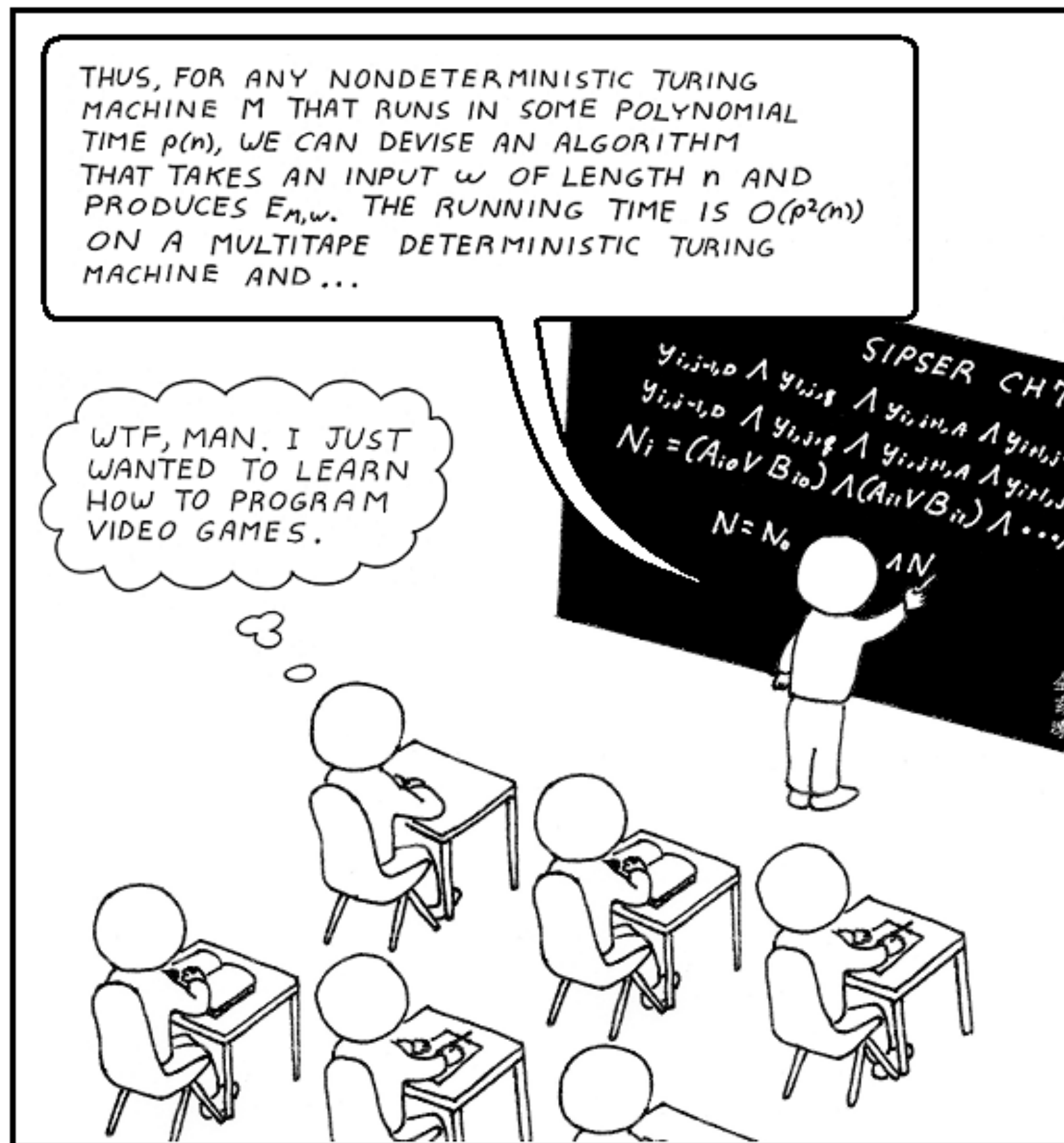


# CSE 549: Computational Biology

Computer Science for ~~Biologists~~ Biology

# What is Computer Science?



# What is Computer Science?

Not actually simple to define constructively

Still debate whether certain areas constitute CS

Computer science is the scientific and practical approach to computation and its applications. It is the systematic study of the feasibility, structure, expression, and mechanization of the methodical procedures (or algorithms) that underlie the acquisition, representation, processing, storage, communication of, and access to information\* ...

## What isn't Computer Science?

**Don't** install operating systems (may develop them)

**Don't** set up the office network (may study / design network protocols)

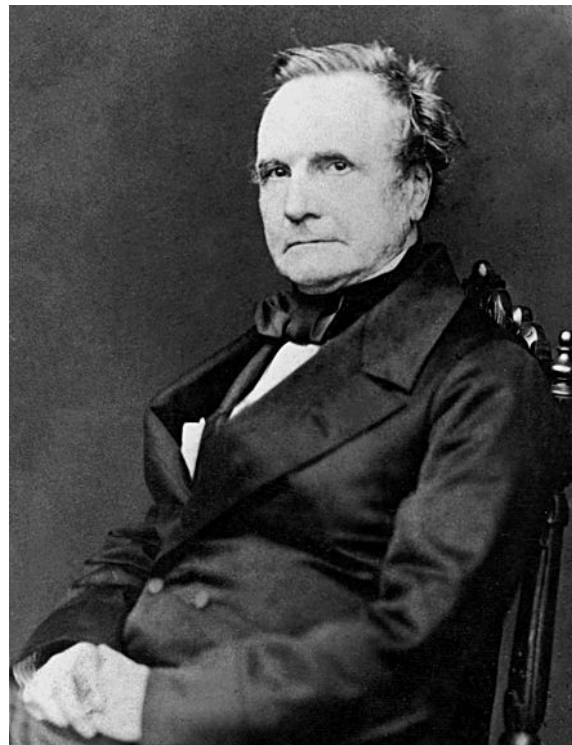
**Not about** Hacking together a program or learning a web-framework — programming  $\neq$  CS (may study formal languages and develop new programming languages)

\*<http://www.cs.bu.edu/AboutCS/WhatIsCS.pdf>

# What is Computer Science?

Started as a branch of Mathematics — early computing machines

Charles Babbage (1791-1871)



Difference engine →  
Analytical engine\*

Ada Lovelace (1791-1871)



Commonly considered the first  
“programmer”; developed an algorithm  
for the analytical engine to compute the  
Bernoulli numbers

\*Analytical engine (would have been the first Turing-complete, general purpose computer) was never completed



# What is Computer Science?

What is “computable”? Early 20th century & birth of “modern” CS



# What is Computer Science?

What is “computable”? Early 20th century & birth of “modern” CS



Kurt  
Gödel

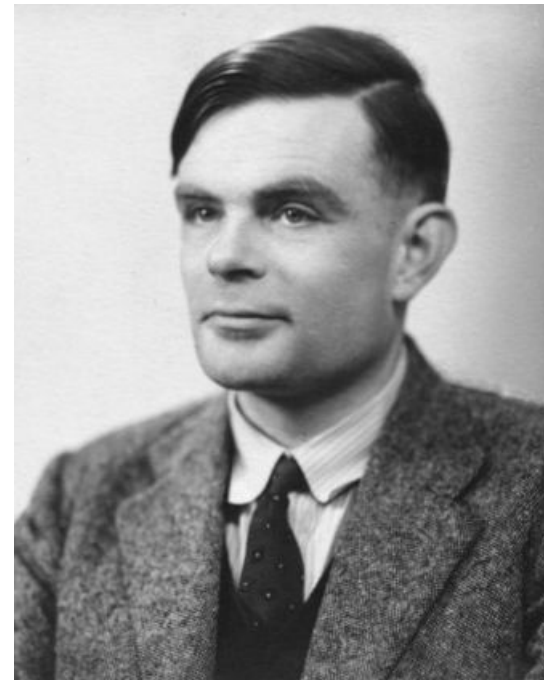


# What is Computer Science?

What is “computable”? Early 20th century & birth of “modern” CS



Kurt  
Gödel



Alan  
Turing





# What is Computer Science?

What is “computable”? Early 20th century & birth of “modern” CS



Kurt  
Gödel



Alan  
Turing



Alonzo  
Church





# What is Computer Science?

What is “computable”? Early 20th century & birth of “modern” CS



Kurt  
Gödel



Alan  
Turing



Alonzo  
Church



John  
von Neumann

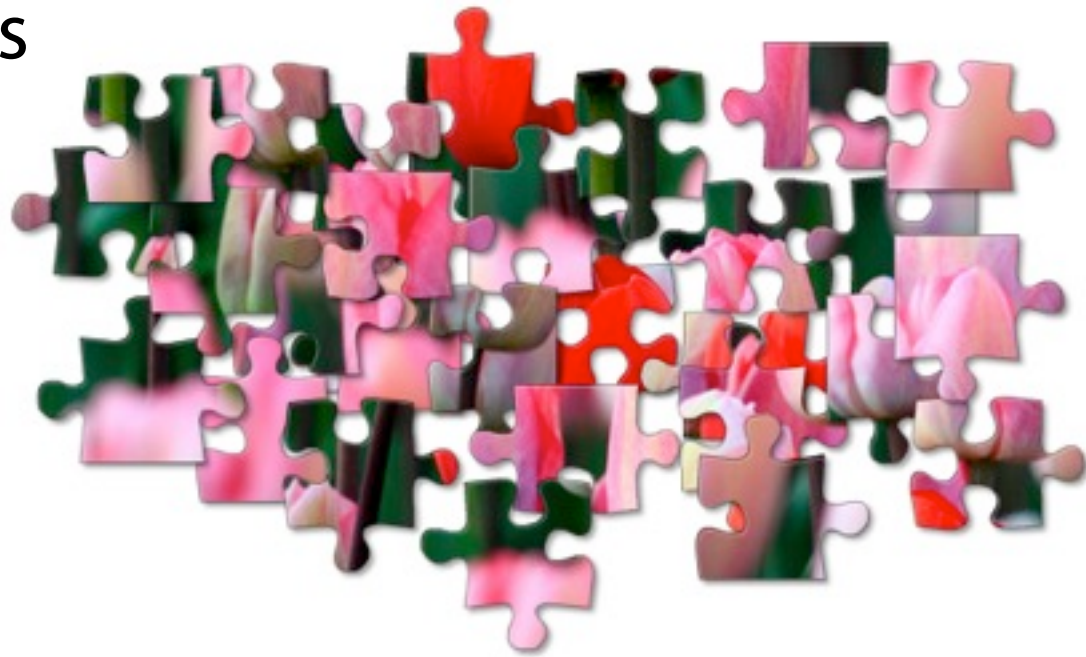
# What is Computer Science?

Concerned with the development of provably **correct** and **efficient** computational procedures (algorithms & data structures) to answer **well-specified** problems.

To answer a computational question, we first need a well-formulated problem.

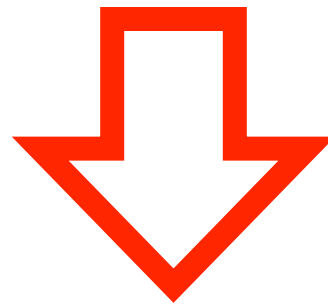
# Assembly

Reads



+

Reference genome



Input DNA



How to assemble  
puzzle without the  
benefit of knowing  
what the finished  
product looks like?

# Assembly

Whole-genome “shotgun” sequencing starts by copying and fragmenting the DNA

(“Shotgun” refers to the random fragmentation of the whole genome; like it was fired from a shotgun)

Input: GCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT

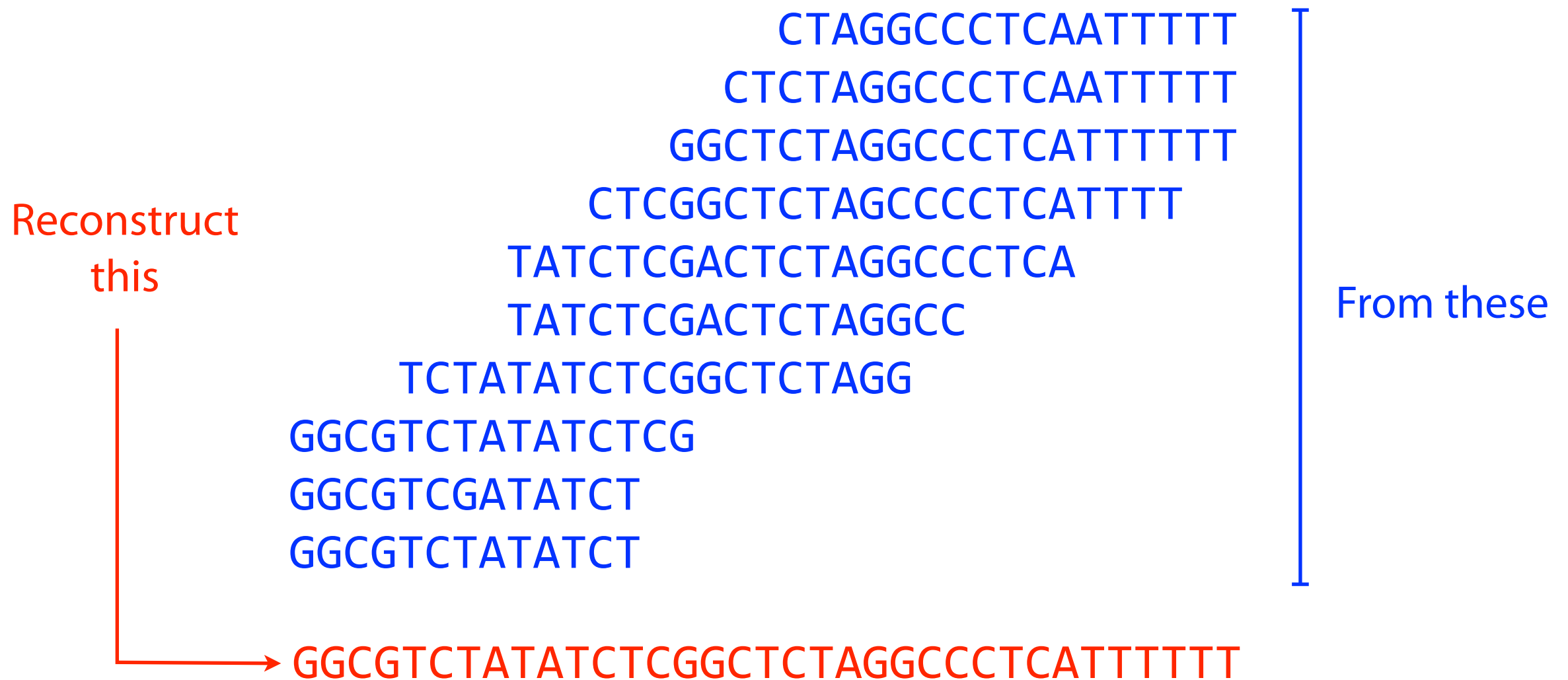
Copy: GCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT  
GCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT  
GCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT  
GCGTCTATATCTCGGCTCTAGGCCCTCATTTTTTT

Fragment: GCGTCTA TATCTCGG CTCTAGGCCCTC ATTTTTT  
GGC GTCTATAT CTCGGCTCTAGGCCCTCA TTTTTT  
GCGTC TATATCT CGGCTCTAGGCCCT CATTTTTT  
GCGTCTAT ATCTCGGCTCTAG GCCCTCA TTTTTT



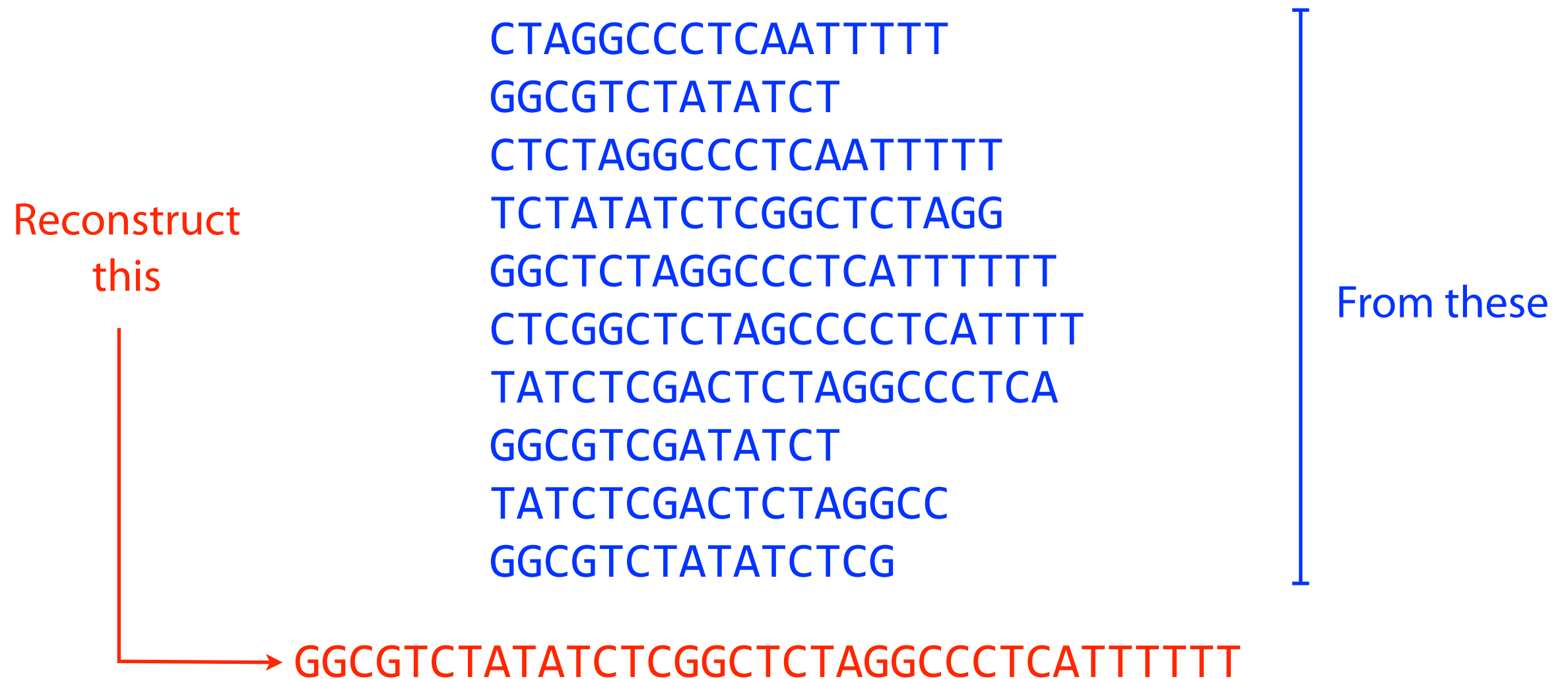
# Assembly

Assume sequencing produces such a large # fragments that almost all genome positions are *covered* by many fragments...



# Assembly

...but we don't know what came from where



# What is Computer Science?

Concerned with the development of provably **correct** and **efficient** computational procedures (algorithms & data structures) to answer **well-specified** problems.

To answer a computational question, we first need a well-formulated problem.

**Given:** a collection,  $R$ , of sequencing reads (strings)

**Find:** The genome (string),  $G$ , that generated them

# What is Computer Science?

Concerned with the development of provably **correct** and **efficient** computational procedures (algorithms & data structures) to answer **well-specified** problems.

To answer a computational question, we first need a well-formulated problem.

**Given:** a collection  $R$ , of sequencing reads (strings)

**Find:** The genome (string),  $G$ , that generated them

**Not** well-specified.

What makes one genome more likely than another?

What constraints do we place on the space of solutions?



# What is Computer Science?

Concerned with the development of provably **correct** and **efficient** computational procedures (algorithms & data structures) to answer **well-specified** problems.

To answer a computational question, we first need a well-formulated problem.

**Given:** a collection,  $R$ , of sequencing reads (strings)

**Find:** The shortest genome (string),  $G$ , that contains all of them



# Shortest Common Superstring

**Given:** a collection,  $S = \{s_1, s_2, \dots, s_k\}$ , of sequencing reads (strings)

**Find\*:** The shortest possible genome (string),  $G$ , such that  $s_1, s_2, \dots, s_k$  are all substrings of  $G$

**How**, might we go about solving this problem?

\*for reasons we'll explore later, this isn't actually a great formulation for genome assembly.

# Shortest common superstring

Given a collection of strings  $S$ , find  $SCS(S)$ : the shortest string that contains all strings in  $S$  as substrings

Without requirement of “shortest,” it’s easy: just concatenate them

Example:  $S$ : BAA AAB BBA ABA ABB BBB AAA BAB

Concatenation: BAAAABBBBAABAABBBBBBAAABAB  
└────────── 24 ─────────┘

$SCS(S)$ : AAABBBBABAA  
└── 10 ─┘

AAA  
AAB  
ABB  
BBB  
BBA  
BAB  
ABA  
BAA

# Shortest common superstring

Can we solve it?

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

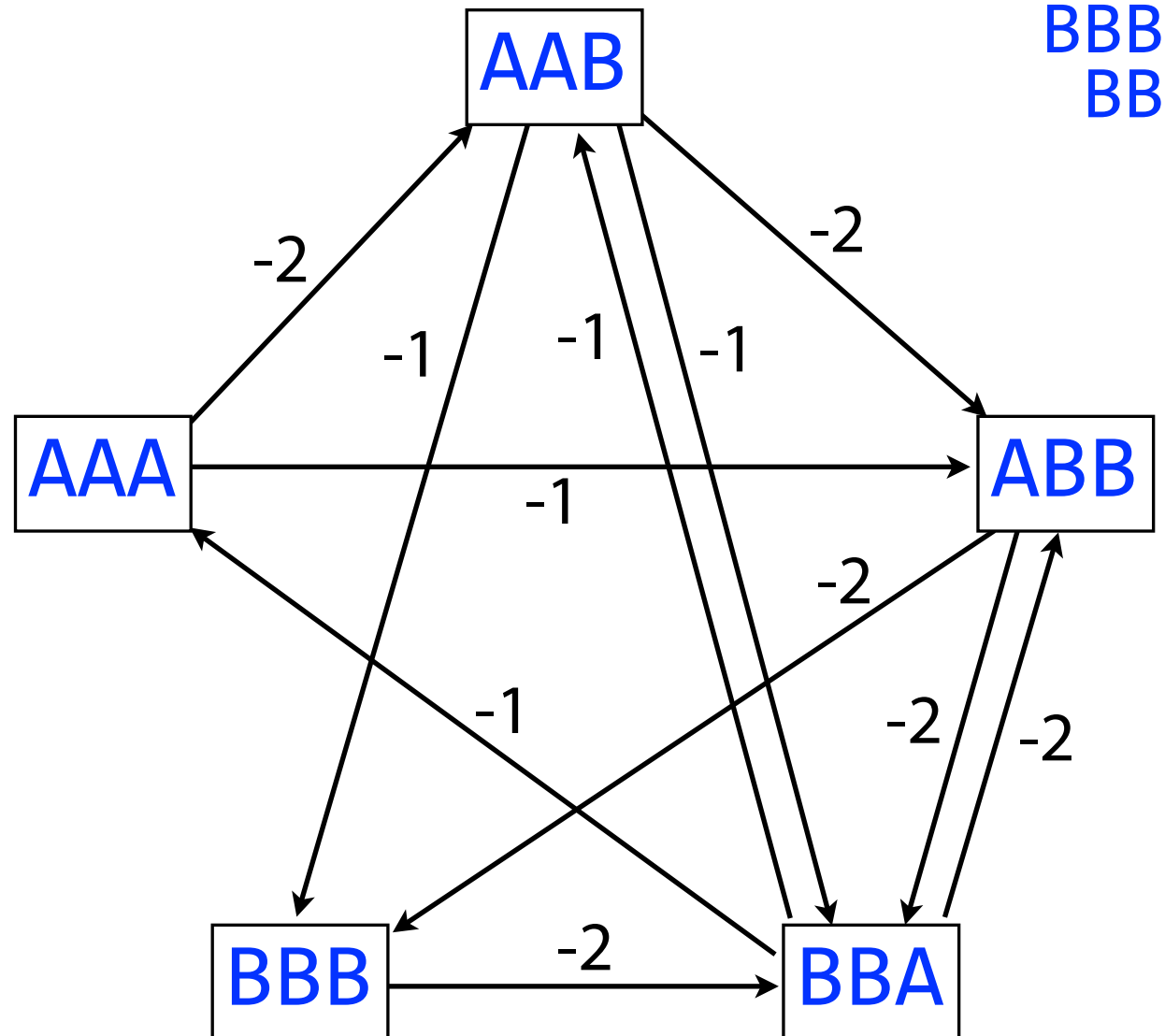
SCS corresponds to a path that visits every node once, minimizing total cost along path

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

S: AAA AAB ABB BBB BBA

SCS(S): AAABBBBA

AAA  
AAB  
ABB  
BBB  
BBA





# Shortest common superstring

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

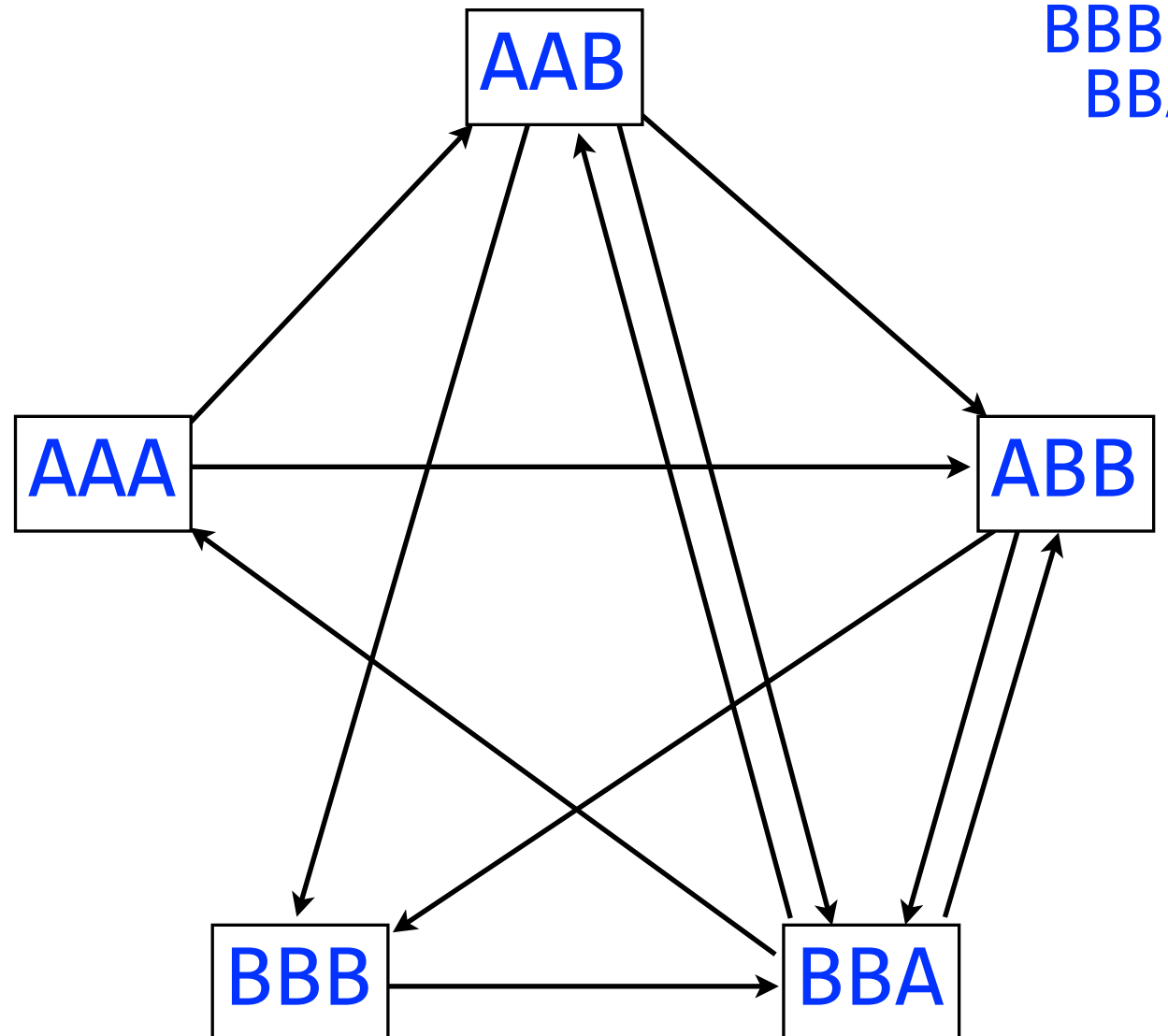
That's the *Hamiltonian Path* problem: NP-complete

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

$S$ : AAA AAB ABB BBB BBA

$SCS(S)$ : AAABBBBA

AAA  
AAB  
ABB  
BBB  
BBA



# Shortest common superstring & friends

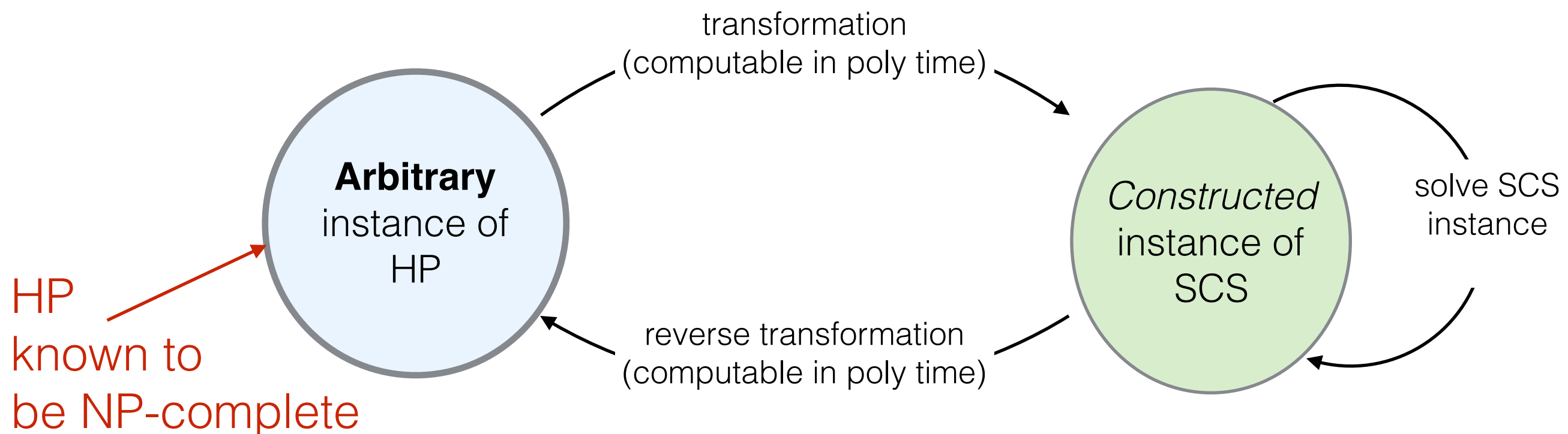
Traveling Salesman, Hamiltonian Path, and Shortest Common Superstring are all NP-hard

For refreshers on Traveling Salesman, Hamiltonian Path, NP-hardness and NP-completeness, see Chapters 34 and 35 of “Introduction to Algorithms” by Cormen, Leiserson, Rivest and Stein, or Chapters 8 and 9 of “Algorithms” by Dasgupta, Papadimitriou and Vazirani (free online: <http://www.cs.berkeley.edu/~vazirani/algorithms>)

**Important note:** The fact that we modeled SCS as NP-hard problems (TSP and HP) **does not** prove that (the decision version of) SCS is NP-complete. To do that, we must **reduce** a known NP-complete problem to **SCS**.

**Given** an instance  $I$  of a known hard problem, **generate** an instance  $I'$  of SCS such that if we can solve  $I'$  in polynomial time, then we can solve  $I$  in polynomial time. This *implies* that SCS is *at least* as hard as the hard problem.

This can be done e.g. with HAMILTONIAN PATH



# Shortest Common Superstring

The fact that SCS is **NP-complete** means that it is unlikely that there exists *any* algorithm that can solve a general instance of this problem in time polynomial in  $n$  — the number of strings.

If we give up on finding the *shortest* possible superstring  $G$ , how does the situation change?



# Shortest Common Superstring

There's a “greedy” *heuristic* that turns out to be an *approximation algorithm* (provides a solution within a constant factor of the optimum)

At *each step*, chose the *pair of strings* with the *maximum overlap*, merge them, and return the merged string to the collection.

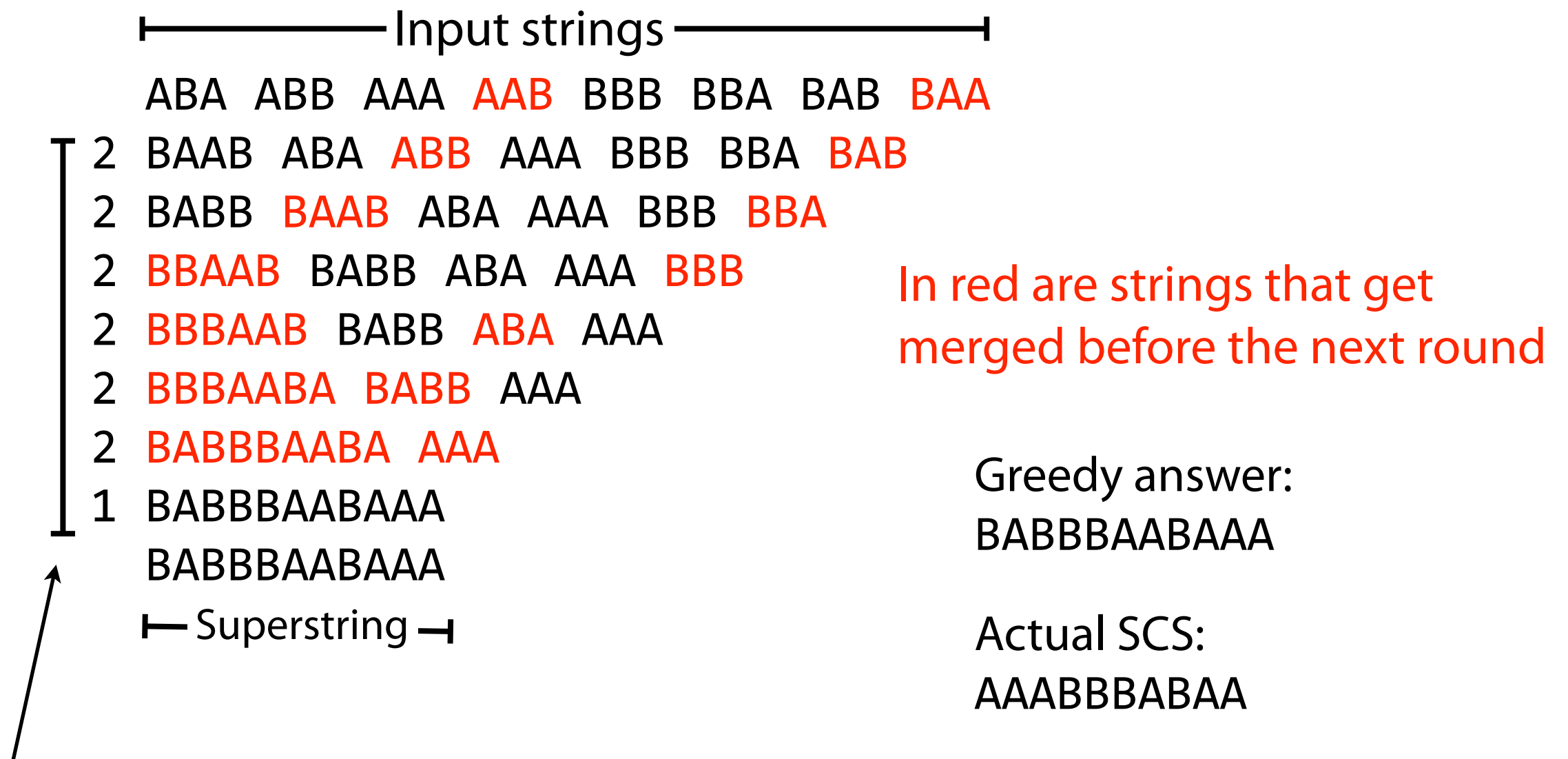
*Greedy conjecture* factor of 2-OPT *is the worst case*

Different approx. (not all greedy)

ratio	authors	year
approximating SCS		
3	Blum, Jiang, Li, Tromp and Yannakakis [4]	1991
$2\frac{8}{9}$	Teng, Yao [23]	1993
$2\frac{5}{6}$	Czumaj, Gasieniec, Piotrow, Rytter [8]	1994
$2\frac{50}{63}$	Kosaraju, Park, Stein [15]	1994
$2\frac{3}{4}$	Armen, Stein [1]	1994
$2\frac{50}{69}$	Armen, Stein [2]	1995
$2\frac{2}{3}$	Armen, Stein [3]	1996
$2\frac{25}{42}$	Breslauer, Jiang, Jiang [5]	1997
$2\frac{1}{2}$	Sweedyk [21]	1999
$2\frac{1}{2}$	Kaplan, Lewenstein, Shafrir, Sviridenko [12]	2005
$2\frac{1}{2}$	Paluch, Elbassioni, van Zuylen [18]	2012
$2\frac{11}{23}$	Mucha [16]	2013

# Shortest common superstring: greedy

Greedy-SCS algorithm in action ( $l = 1$ ):



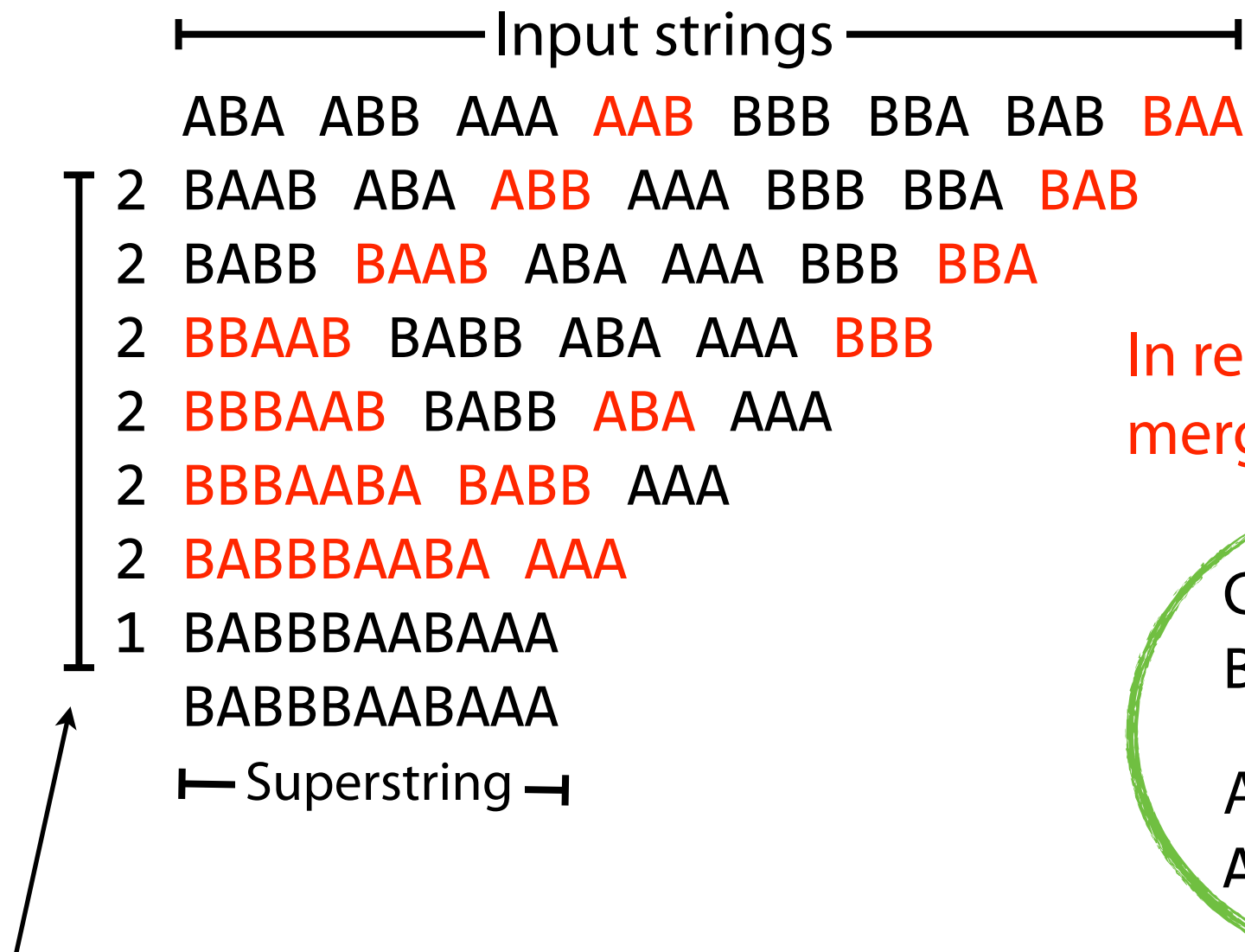
Rounds of merging, one merge per line.

Number in first column = length of overlap merged before that round.

# Shortest common superstring: greedy

Greedy-SCS algorithm in action ( $l = 1$ ):

Note: approx. guarantee  
is on length of the superstring  
Actual result may be very different.



In red are strings that get merged before the next round

Greedy answer:  
BABBBBAABAAA

Actual SCS:  
AAABBBBABAA

Rounds of merging, one merge per line.

Number in first column = length of overlap merged before that round.