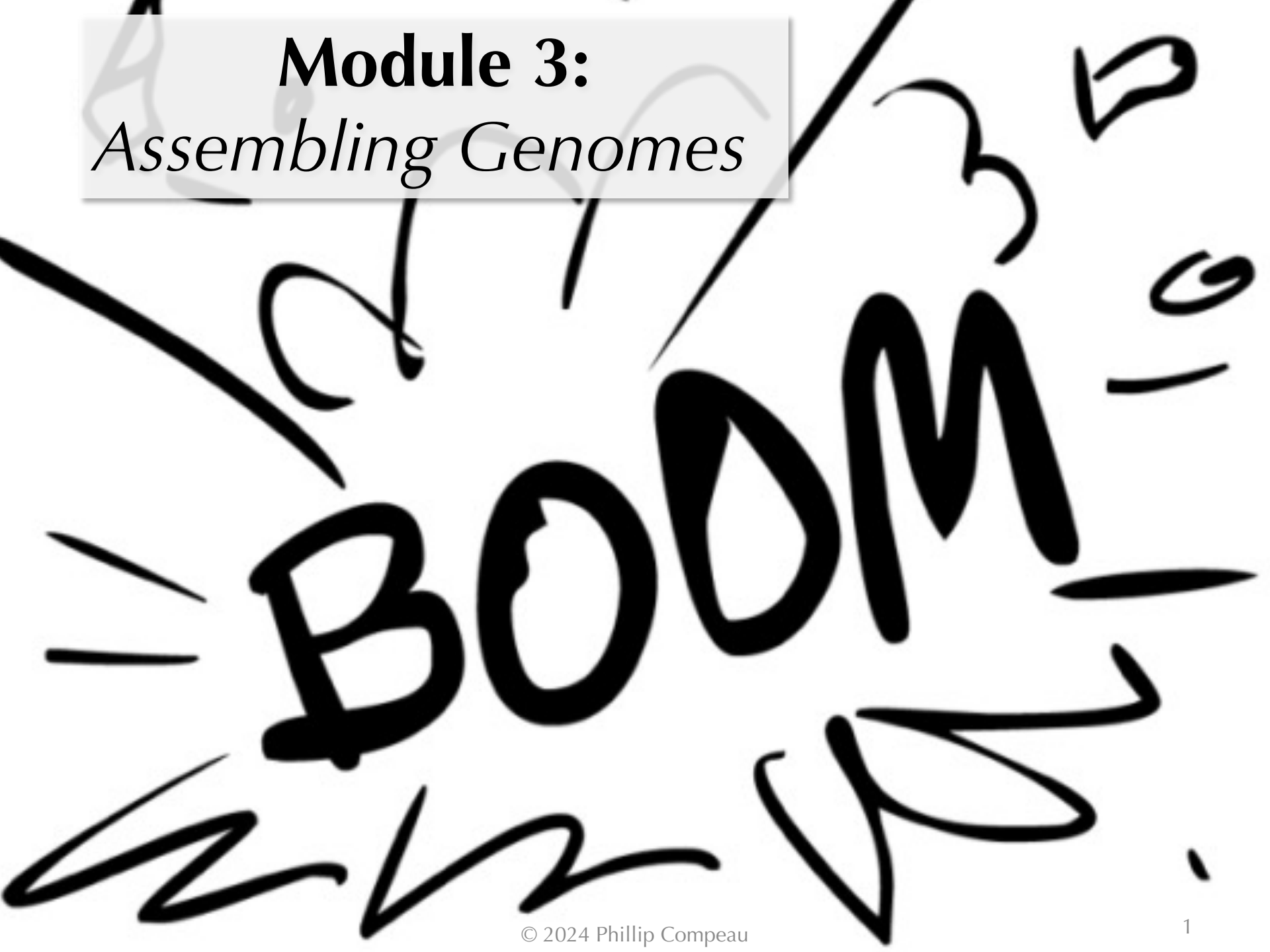
Module 3:
Assembling Genomes



AN INTRODUCTION TO GENOME SEQUENCING

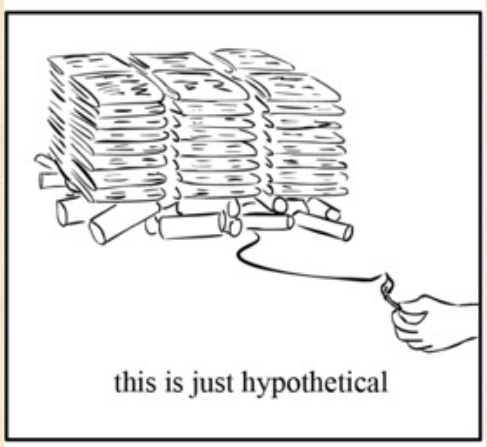
Eternity II: The Highest-Stakes Puzzle in History

Courtesy: Matej Bat'ha

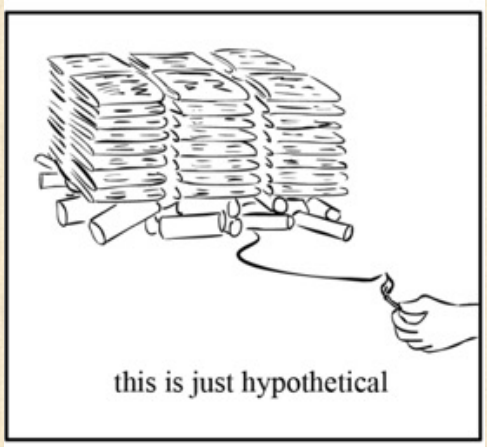
The Newspaper Problem

The Newspaper Problem

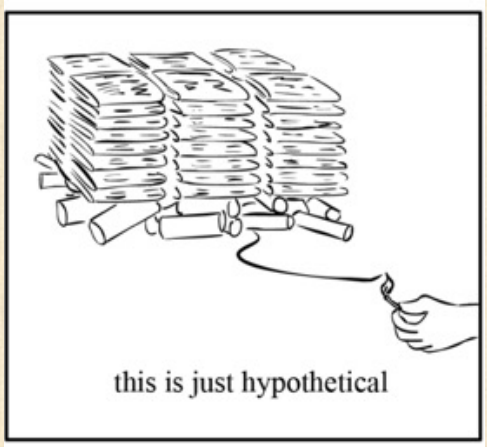
The Newspaper Problem

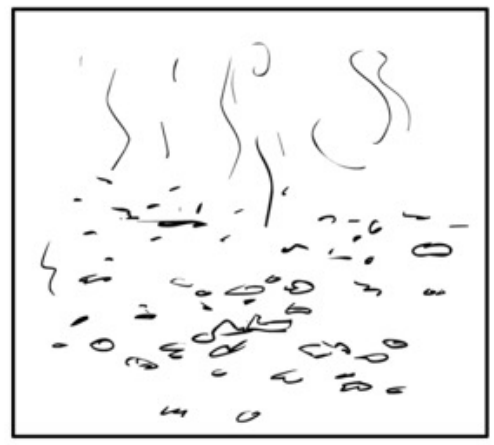


The Newspaper Problem

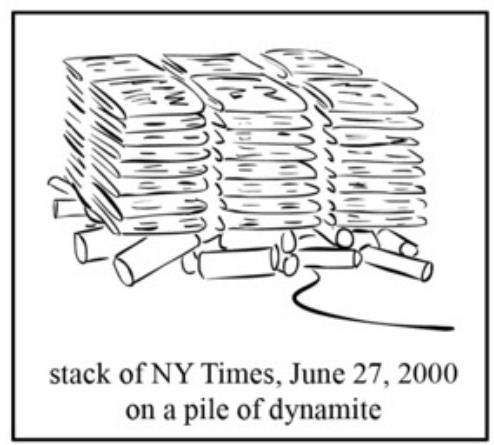


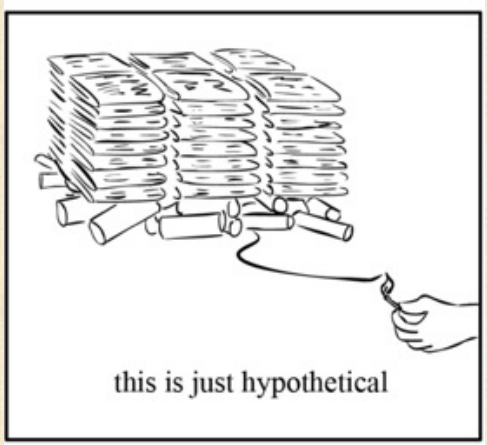
The Newspaper Problem

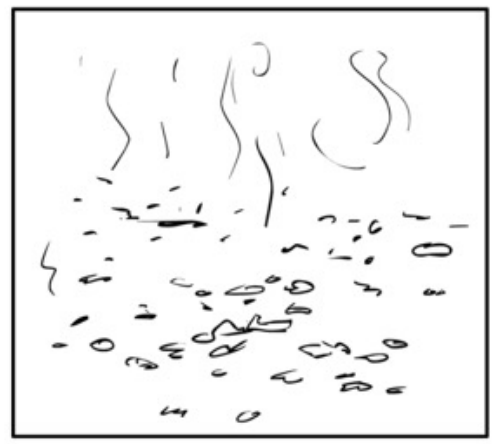


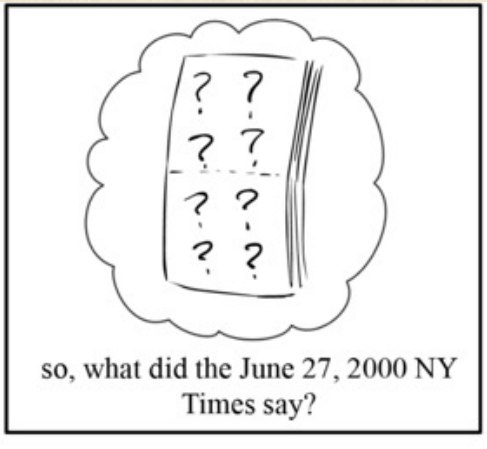


The Newspaper Problem

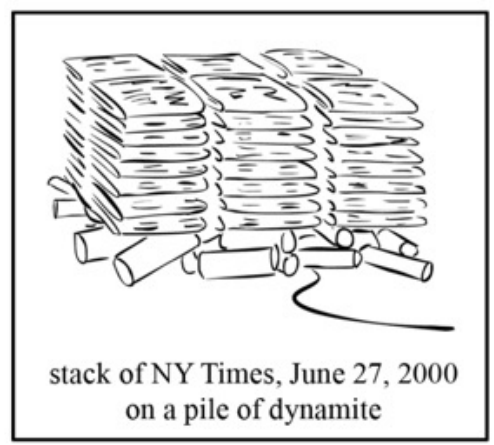


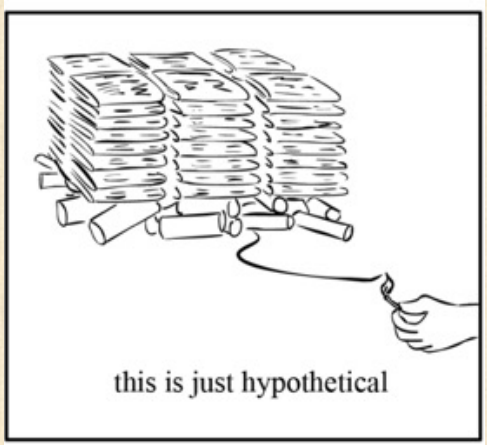


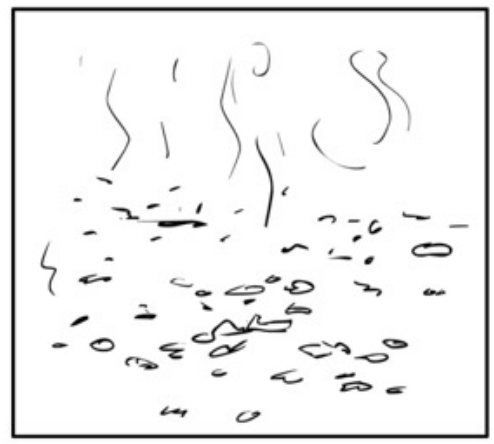


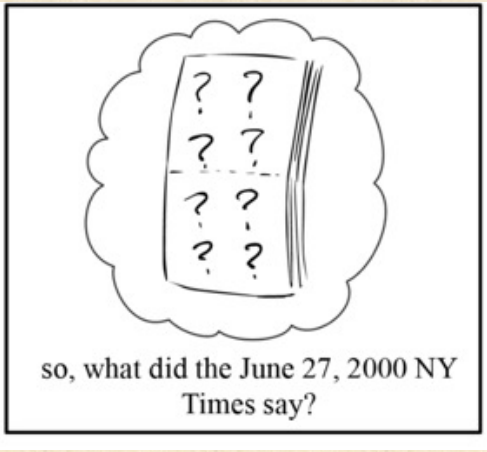


The Newspaper Problem



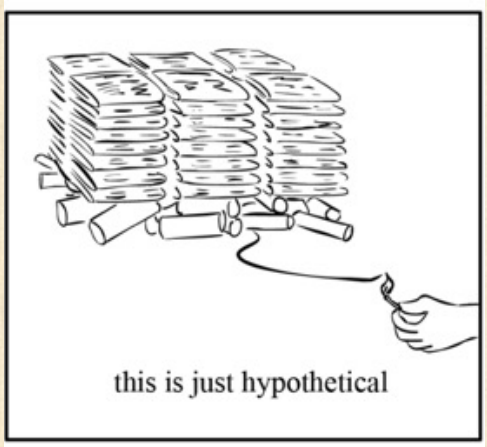


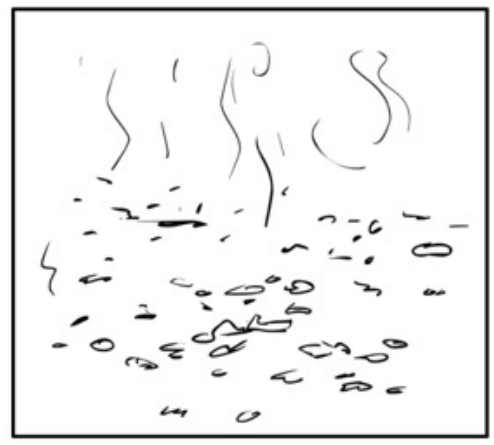


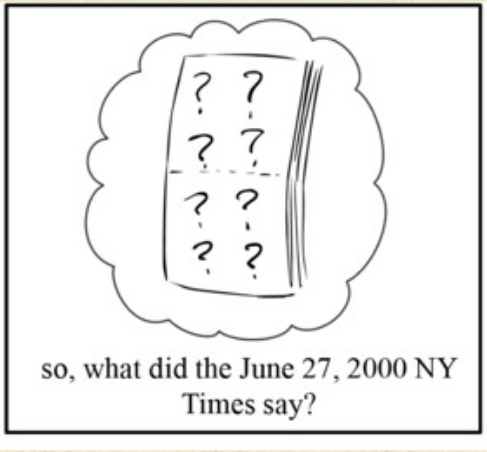


The Newspaper Problem is an **overlap puzzle**.

The Newspaper Problem





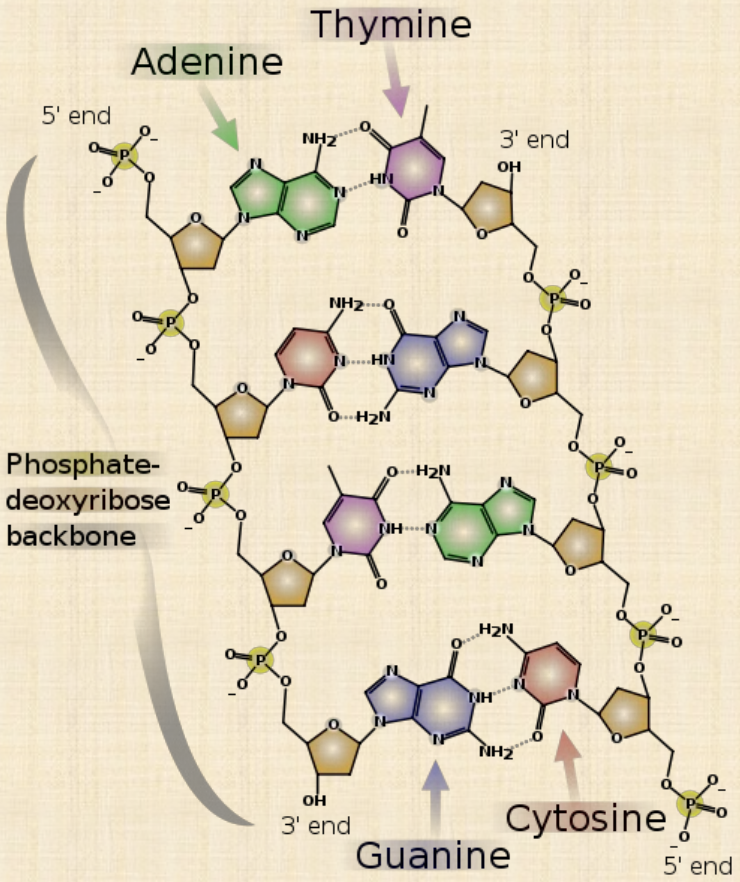


But what does this have to do with biology?

The Order of Nucleotides Determines Genetics

Nucleotide: Half of one "rung" of DNA.

Key point: if we know one strand of DNA, we get the other strand for free because of this "complementarity".



DNA's Molecular Structure

Genome "Sequencing" Means "Reading" the Genome

Genome: The nucleotide sequence read down one side of an organism's chromosomal DNA. A human genome has about 3 billion letters.

```
…CCGTAGTCGCATGGAACAGTATACGAGACAGTACAGATACGATACGATACGATCATTAACCGAGAGTACCAGATTCCAGATCATACG
TTACGCTTAGCTACGGACGTACGATACCCAGATTACGATCCATATAGATATAACCGGTGTGTCTTGCTAATACGTAACGGGGTGCCT
TCGATAGGTCAGAATACCAGATCTCTCGATCTTCTTACAGATACTACGATCCCCAGATACTACCCCTACTGACCCATCGTACGGGTA
CTACTACGGATATGATACCGATGTAGAGGGATCCATATATCCCGAGACGTCTCGCGCATAAGATCATCGTCTAGATACACGTACGTA
CTAGACTAGCGTATGCCTCTTATGATCGTCCCGATCGAGTCGCGTGCTCAGAAAAGCTACGATACGATACCCGATACTAGACCATAG…
```

Genome "Sequencing" Means "Reading" the Genome

Genome: The nucleotide sequence read down one side of an organism's chromosomal DNA. A human genome has about 3 billion letters.

```
…CCGTAGTCGCATGGAACAGTATACGAGACAGTACAGATACGATACGATACGATCATTAACCGAGAGTACCAGATTCCAGATCATACG
TTACGCTTAGCTACGGACGTACGATACCCAGATTACGATCCATATAGATATAACCGGTGTGTCTTGCTAATACGTAACGGGGTGCCT
TCGATAGGTCAGAATACCAGATCTCTCGATCTTCTTACAGATACTACGATCCCCAGATACTACCCCTACTGACCCATCGTACGGGTA
CTACTACGGATATGATACCGATGTAGAGGGATCCATATATCCCGAGACGTCTCGCGCATAAGATCATCGTCTAGATACACGTACGTA
CTAGACTAGCGTATGCCTCTTATGATCGTCCCGATCGAGTCGCGTGCTCAGAAAAGCTACGATACGATACCCGATACTAGACCATAG…
```

Polychaos dubium (an amoeba) has one of the longest known genomes: 670 billion nucleotides.

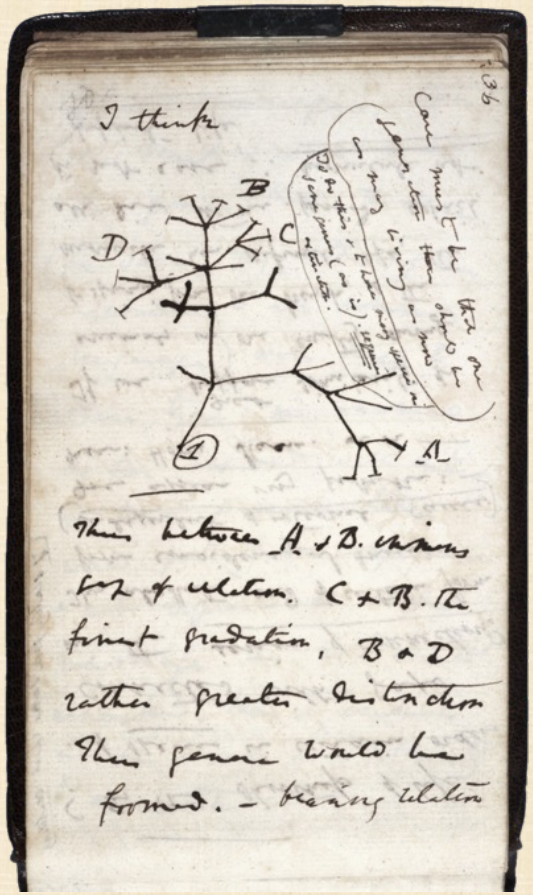
Genome "Sequencing" Means "Reading" the Genome

Genome: The nucleotide sequence read down one side of an organism's chromosomal DNA. A human genome has about 3 billion letters.

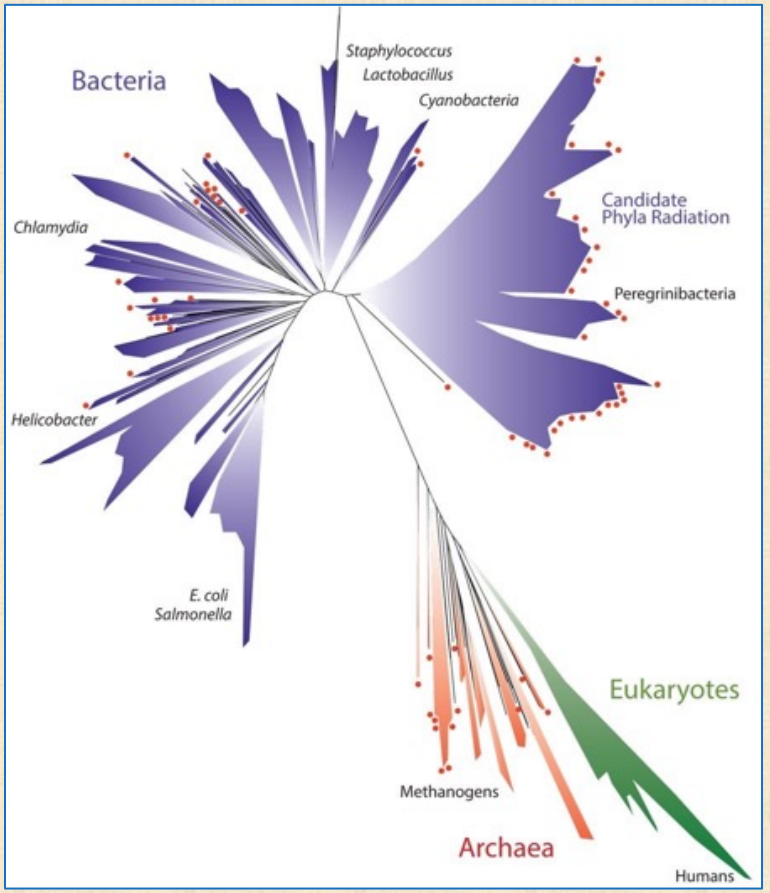
```
…CCGTAGTCGCATGGAACAGTATACGAGACAGTACAGATACGATACGATACGATCATTAACCGAGAGTACCAGATTCCAGATCATACG
TTACGCTTAGCTACGGACGTACGATACCCAGATTACGATCCATATAGATATAACCGGTGTGTCTTGCTAATACGTAACGGGGTGCCT
TCGATAGGTCAGAATACCAGATCTCTCGATCTTCTTACAGATACTACGATCCCCAGATACTACCCCTACTGACCCATCGTACGGGTA
CTACTACGGATATGATACCGATGTAGAGGGATCCATATATCCCGAGACGTCTCGCGCATAAGATCATCGTCTAGATACACGTACGTA
CTAGACTAGCGTATGCCTCTTATGATCGTCCCGATCGAGTCGCGTGCTCAGAAAAGCTACGATACGATACCCGATACTAGACCATAG…
```

Key Point: DNA is submicroscopic! How do we read something that we cannot see?

We Sequence a Species's Genome to Unlock its Genetic Identity



Darwin's notebook c. 1837

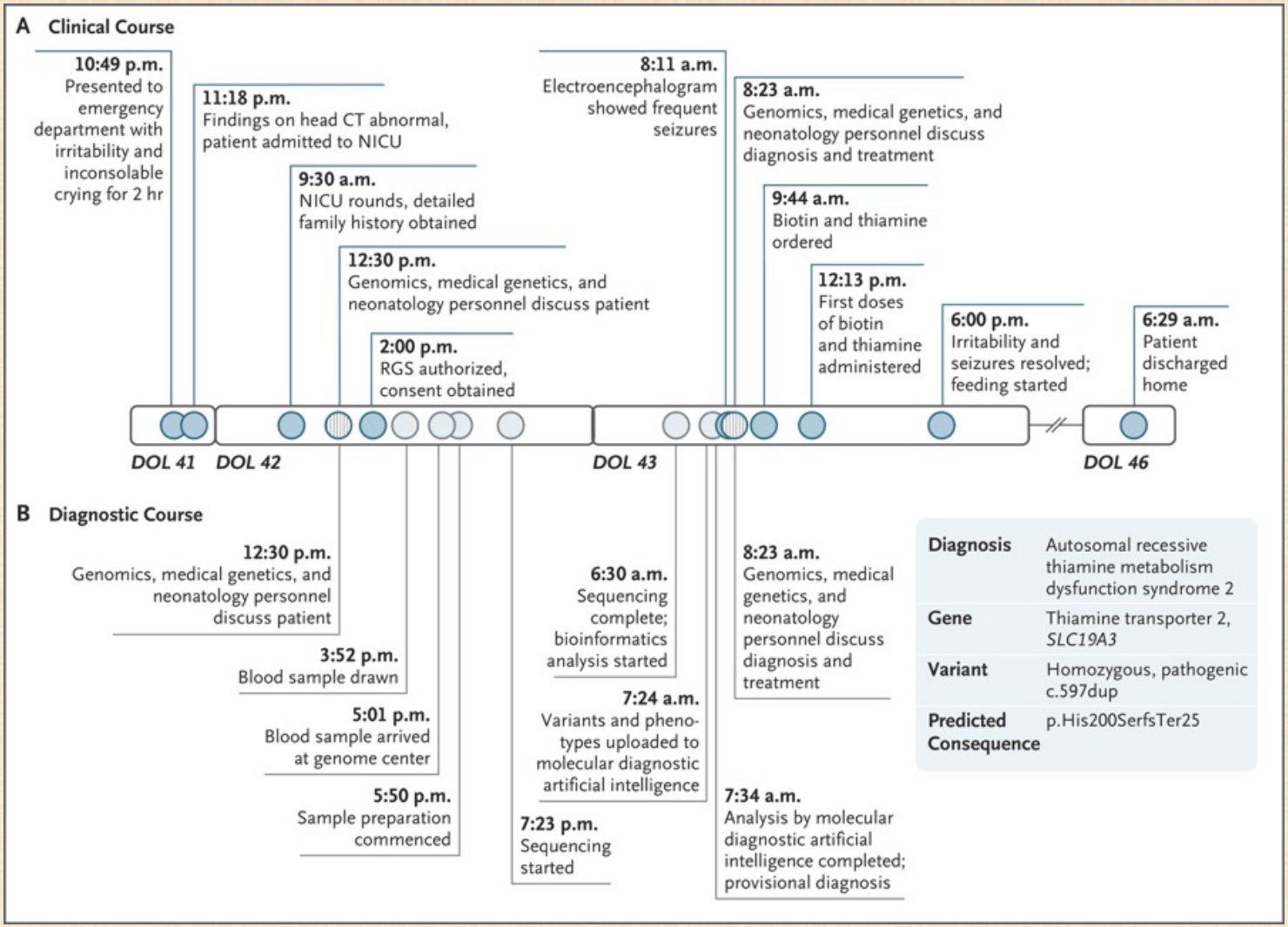


Hug et al., 2016
Nature Biotechnology, Discovery Magazine

We Sequence an Individual's Genome to Find What Makes them Unique

Nicholas Volker:
First human whose life was saved because of genome sequencing (2011).

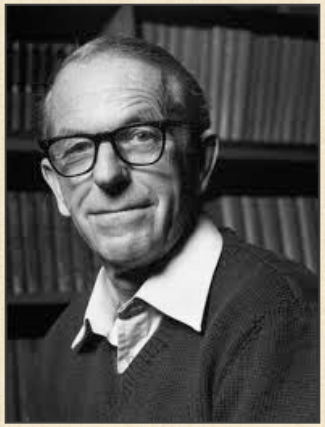
Ten years later, genome sequencing saves a life in 13 hours



History of Genome Sequencing

Late 1970s: Walter Gilbert and Frederick Sanger develop independent sequencing methods.

Walter Gilbert



Frederick Sanger

```
GAGTTTTATCGCTTCCATGACGCAGAAGTTAACACTTTCGGATATTTCTGATGAGTCGAAAAATTATCTTGATAAAGCAGGAATTACTACTGCTTGTTTACGAATTAAATCGAAGTGGACTGCTGGCGGAAAATGAGAAAATTCGACCTATCCTTGCGCAGCT
CGAGAAGCTCTTACTTTGCGACCTTTCGCCATCAACTAACGATTCTGTCAAAAACTGACGCGTTGGATGAGGAGAAGTGGCTTAATATGCTTGGCACGTTCGTCAAGGACTGGTTTAGATATGAGTCACATTTTGTTCATGGTAGAGATTCTCTTGTTGACAT
TTTAAAAGAGCGTGGATTACTATCTGAGTCCGATGCTGTTCAACCACTAATAGGTAAGAAATCATGAGTCAAGTTACTGAACAATCCGTACGTTCCAGACCGCTTTGGCCTCTATTAAGCTCATTCAGGCTTCTGCCGTTTTGGATTTAACCGAAGATGATTT
CGATTTTCTGACGAGTAACAAAGTTTGGATTGCTACTGACCGCTCTCGTGCTCGTCGCTGCGTTGAGGCTTGCGTTTATGGTACGCTGGACTTTGTGGGATACCCTCGCTTTCCTGCTCCTGTTGAGTTTATTGCTGCCGTCATTGCTTATTATGTTCATCCC
GTCAACATTCAAACGGCCTGTCTCATCATGGAAGGCGCTGAATTTACGGAAAACATTATTAATGGCGTCGAGCGTCCGGTTAAAGCCGCTGAATTGTTCGCGTTTACCTTGCGTGTACGCGCAGGAAACACTGACGTTCTTACTGACGCAGAAGAAAACGTGC
GTCAAAAATTACGTGCGGAAGGAGTGATGTAATGTCTAAAGGTAAAAAACGTTCTGGCGCTCGCCCTGGTCGTCCGCAGCCGTTGCAGGTACTAAAGGCAAGCGTAAAGGCGCTCGTCTTTGGTATGTAGGTGGTCAACAATTTTAATTGCAGGGGCTTCGG
CCCCTTACTTGAGGATAAATTATGTCTAATATTCAAACTGGCGCCGAGCGTATGCCGCATGACCTTTCCCATCTTGGCTTCCTTGCTGGTCAGATTGGTCGTCTTATTACCATTTCAACTACTCCGGTTATCGCTGGCGACTCCTTCGAGATGGACGCCGTTG
GCGCTCTCCGTCTTTCTCCATTGCGTCGTGGCCTTGCTATTGACTCTACTGTAGACATTTTTACTTTTTATGTCCCTCATCGTCACGTTTATGGTGAACAGTGGATTAAGTTCATGAAGGATGGTGTTAATGCCACTCCTCTCCCGACTGTTAACACTACTGG
TTATATTGACCATGCCGCTTTTCTTGGCACGATTAACCCTGATACCAATAAAATCCCTAAGCATTTGTTTCAGGGTTATTTGAATATCTATAACAACTATTTTAAAGCGCCGTGGATGCCTGACCGTACCGAGGCTAACCCTAATGAGCTTAATCAAGATGAT
GCTCGTTATGGTTTCCGTTGCTGCCATCTCAAAAACATTTGGACTGCTCCGCTTCCTCCTGAGACTGAGCTTTCTCGCCAAATGACGACTTCTACCACATCTATTGACATTATGGGTCTGCAAGCTGCTTATGCTAATTTGCATACTGACCAAGAACGTGATT
ACTTCATGCAGCGTTACCATGATGTTATTTCTTCATTTGGAGGTAAAACCTCTTATGACGCTGACAACCGTCCTTTACTTGTCATGCGCTCTAATCTCTGGGCATCTGGCTATGATGTTGATGGAACTGACCAAACGTCGTTAGGCCAGTTTTCTGGTCGTGT
TCAACAGACCTATAAACATTCTGTGCCGCGTTTCTTTGTTCCTGAGCATGGCACTATGTTTACTCTTGCGCTTGTTCGTTTTCCGCCTACTGCGACTAAAGAGATTCAGTACCTTAACGCTAAAGGTGCTTTGACTTATACCGATATTGCTGGCGACCCTGTT
TTGTATGGCAACTTGCCGCCGCCGGTGAAAATTTCTATGAAGGATGTTTTCCGTTCTGGTGATTCGTCTAAGAAGTTTAAGATTGCTGAGGGTCAGTGGTATCGTTATGCGCCCTTCGTATGTTTCTCCTGCTTATCACCTTCTTGAAGGCTTCCCATTCATTCAGG
AACCGCCTTCTGGTGATTTGCAAGAACGCGTACTTATTCGCCACCATGATTATGACCAGTGTTTCCAGTCCGTTCAGTTGTTGCAGTGGAATAGTCAGGTTAAATTTAATGTGACCGTTTATCGCAATCTGCCGACCACTCGCGATTCAATCATGACTTCGTG
ATAAAAGATTGAGTGTGAGGTTATAACGCCGAAGCGGTAAAAATTTTAATTTTTGCCGCTGAGGGGTTGACCAAGCGAAGCGCGGTAGGTTTTCTGCTTAGGAGTTTAATCATGTTTCAGACTTTTATTTCTCGCCATAATTCAAACTTTTTTTCTGATAAGC
TGGTTCTCACTTCTGTTACTCCAGCTTCTTCGGCACCTGTTTTACAGACACCTAAAGCTACATCGTCAACGTTATATTTTGATAGTTTGACGGTTAATGCTGGTAATGGTGGTTTTCTTCATTGCATTCAGATGGATACATCTGTCAACGCCGCTAATCAGGT
TGTTTCTGTTGGTGCTGATATTGCTTTTGATGCCGACCCTAAATTTTTTGCCTGTTTGGTTCGCTTTGAGTCTTCTTCGGTTCCGACTACCCTCCCGACTGCCTATGATGTTTATCCTTTGAATGGTCGCCATGATGGTGGTTATTATACCGTCAAGGACTGT
GTGACTATTGACGTCCTTCCCCGTACGCCGGGCAATAACGTTTATGTTGGTTTCATGGTTTGGTCTAACTTTACCGCTACTAAATGCCGCGGATTGGTTTCGCTGAATCAGGTTATTAAAGAGATTATTTGTCTCCAGCCACTTAAGTGAGGTGATTTATGTT
TGGTGCTATTGCTGGCGGTATTGCTTCTGCTCTTGCTGGTGGCGCCATGTCTAAATTGTTTGGAGGCGGTCAAAAAGCCGCCTCCGGTGGCATTCAAGGTGATGTGCTTGCTACCGATAACAATACTGTAGGCATGGGTGATGCTGGTATTAAATCTGCCATT
CAAGGCTCTAATGTTCCTAACCCTGATGAGGCCGCCCCTAGTTTTGTTTCTGGTGCTATGGCTAAAGCTGGTAAAGGACTTCTTGAAGGTACGTTGCAGGCTGGCACTTCTGCCGTTTCTGATAAGTTGCTTGATTTGGTTGGACTTGGTGGCAAGTCTGCCG
CTGATAAAGGAAAGGATACTCGTGATTATCTTGCTGCTGCATTTCCTGAGCTTAATGCTTGGGAGCGTGCTGGTGCTGATGCTTCCTCTGCTGGTATGGTTGACGCCGGATTTGAGAATCAAAAAGAGCTTACTAAAATGCAACTGGACAATCAGAAAGAGAT
TGCCGAGATGCAAAATGAGACTCAAAAAGAGATTGCTGGCATTCAGTCGGCGACTTCACGCCAGAATACGAAAGACCAGGTATATGCACAAAATGAGATGCTTGCTTATCAACAGAAGGAGTCTACTGCTCGCGTTGCGTCTATTATTGGAAAACACCAATCTT
TCCAAGCAACAGCAGGTTTCCGAGATTATGCGCCAAATGCTTACTCAAGCTCAAACGGCTGGTCAGTATTTTACCAATGACCAAATCAAAGAAATGACTCGCAAGGTTAGTGCTGAGGTTGACTTAGTTCATCAGCAAACGCAGAATCAGCGGTATGGCTCTT
CTCATATTGGCGCTACTGCAAAGGATATTTCTAATGTCGTCACTGATGCTGCTTCTGGTGTGGTTGATATTTTTCATGGTATTGATAAAGCTGTTGCCGATACTTGGAACAATTTCTGGAAAGACGGTAAAGCTGATGGTATTGGCTCTAATTTGTCTAGGAA
ATAACCGTCAGGATTGACACCCTCCCAATTGTATGTTTCATGCCTCCAAATCTTGGAGGCTTTTTATGGTTCGTTCTTATTACCCTTCTGAATGTCACGCTGATTATTTTGACTTTGAGCGTATCGAGGCTCTTAAACCTGCTATTGAGGCTTGTGGCATT
TCTACTCTTTCTCAATCCCCAATGCTTGGCTTCCATAAGCAGATGGATAACCGCATCAAGCTCTTGGAAGAGATTCTGTCTTTTCGTATGCAGGGCGTTGAGTTCGATAATGGTGATATGTATGTTGACGGCCATAAGGCTGCTTCTGACGTTCGTGATGAGT
TTGTATCTGTTACTGAAGAAGTTAATGGATGAATTGGCACAATGCTACAATGTGCTCCCCCAACTTGATATTAATAACACTATAGACCACCGCCCCGAAGGGGACGAAAAATGGTTTTTAGAGAACGAGAAGACGGTTACGCAGTTTTGCCGCAAGCTGGCTGC
TGAACGCCCTCTTAAGGATATTCGCGATGAGTATAATTACCCCAAAAAGAAAGGTATTAAGGATGAGTGTTCAAGATTGCTGGAGGCCTCCACTATGAAATCGCGTAGAGGCTTTGCTATTCAGCGTTTGATGAATGCAATGCGACAGGCTCATGCTGATGGT
TGGTTTATCGTTTTTGACACTCTCACGTTGGCTGACGACCGATTAGAGGCGTTTTATGATAATCCCAATGCTTTGCGTGACTATTTTCGTGATATTGGTCGTATGGTTCTTGCTGCCGAGGGTCGCAAGGCTAATGATTCACACGCCGACTGCTATCAGTATT
TTTGTGTGCCTGAGTATGGTACAGCTAATGGCCGTCTTCATTTCCATGCGGTGCACTTTATGCGGACACTTCCTACAGGTAGCGTTGACCCTAATTTTGGTCGTCGGGTACGCAATCGCCGCCAGTTAAATAGCTTGCAAAATACGTGGCCTTATGGTTACAG
TATGCCCATCGCAGTTCGCTACACGCAGGACGCTTTTTCACGTTCTGGTTGGTTGTGGCCTGTTGATGCTAAAGGTGAGCCGCTTAAAGCTACCAGTTATATGGCTGTTGGTTTCTATGTGGCTAAATACGTTAACAAAAAGTCAGATATGGACCTTGCTGCT
AAAGGTCTAGGAGCTAAAGAATGGAACAACTCACTAAAAACCAAGCTGTCGCTACTTCCCAAGAAGCTGTTCAGAATCAGAATGAGCCCGCAACTTCGGGATGAAAATGCTCACAATGACAAATCTGTCCACGGAGTGCTTAATCCAACTTACCAAGCTGGGTT
ACGACGCGACGCCGTTCAACCAGATATTGAAGCAGAACGCAAAAAGAGAGATGAGATTGAGGCTGGGAAAAGTTACTGTAGCCGACGTTTTGGCGGCGCAACCTGTGACGACAAATCTGCTCAAATTTATGCGCGCTTCGATAAAAATGATTGGCGTATCCAA
CCTGCA
```

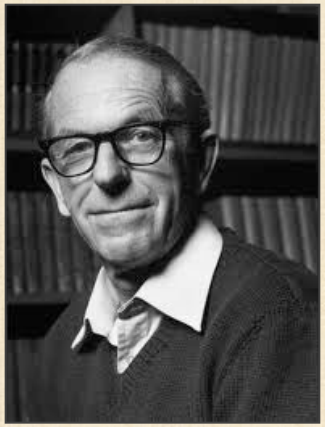
Bacterial phage PhiX174 genome (5,386 nucleotides)

History of Genome Sequencing

Late 1970s: Walter Gilbert and Frederick Sanger develop independent sequencing methods.

1980: They share the Nobel Prize in Chemistry.

Walter Gilbert



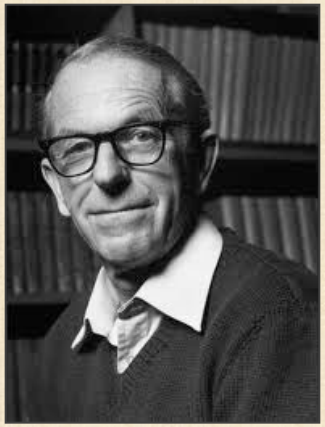
Frederick Sanger

History of Genome Sequencing

Late 1970s: Walter Gilbert and Frederick Sanger develop independent sequencing methods.

Walter Gilbert

1980: They share the Nobel Prize in Chemistry.

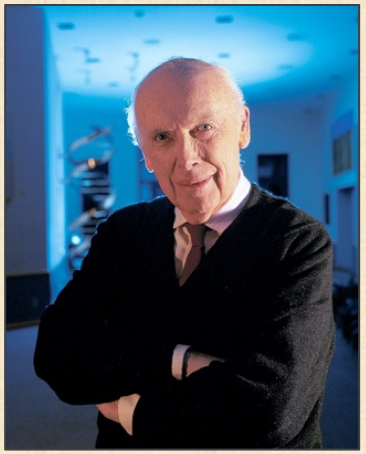


Frederick Sanger

However, their approaches cost about $1 per nucleotide.

The Race to Sequence the Human Genome

1990: Human Genome Project given $3 billion to sequence human genome.



James Watson

The Race to Sequence the Human Genome

1990: Human Genome Project given $3 billion to sequence human genome.

1992: James Watson resigns, replaced by Francis Collins.

Francis Collins

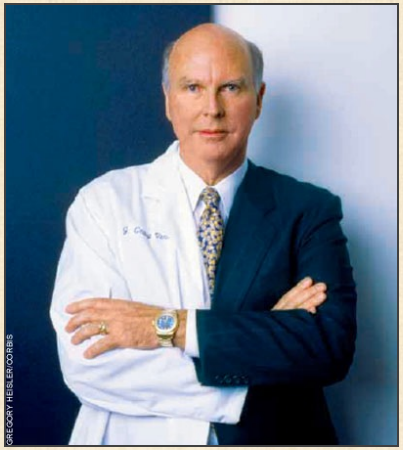
The Race to Sequence the Human Genome

1990: Human Genome Project given $3 billion to sequence human genome.

1992: James Watson resigns, replaced by Francis Collins.

1997: Craig Venter founds Celera Genomics with same goal.

Francis Collins



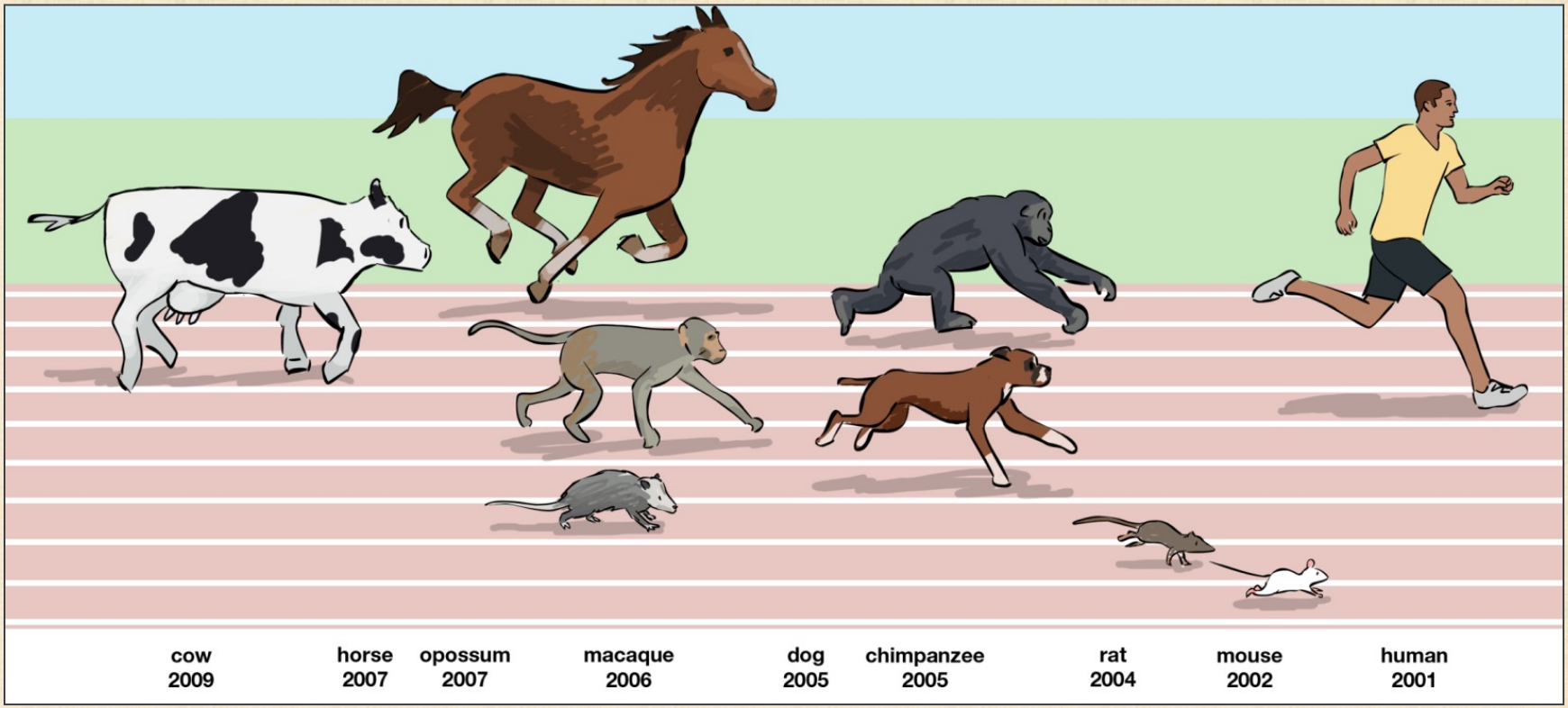
Craig Venter

The Race to Sequence the Human Genome

2000: First draft of human genome published.

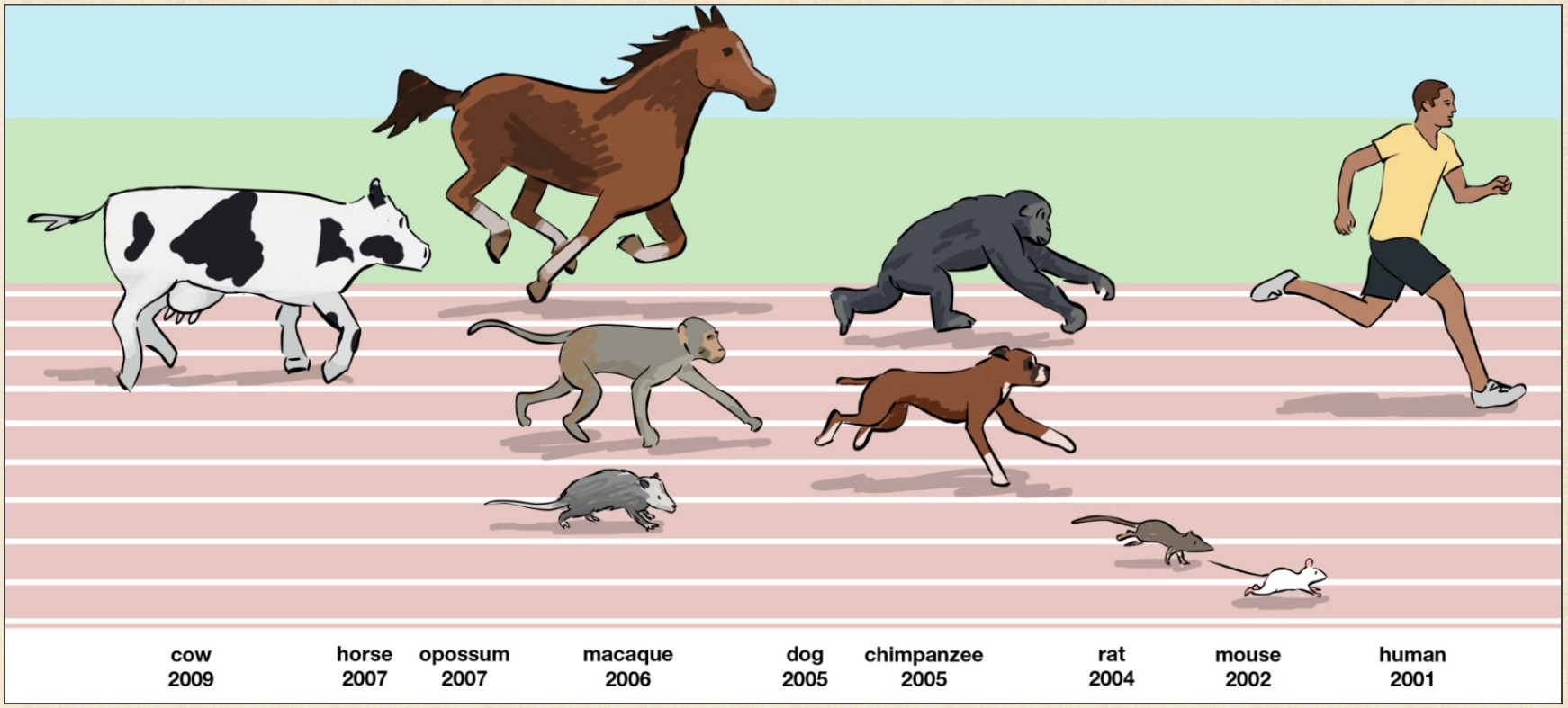
From One Mammal Genome to Many

Early 2000s: Many more mammalian genomes are sequenced using Sanger's approach.

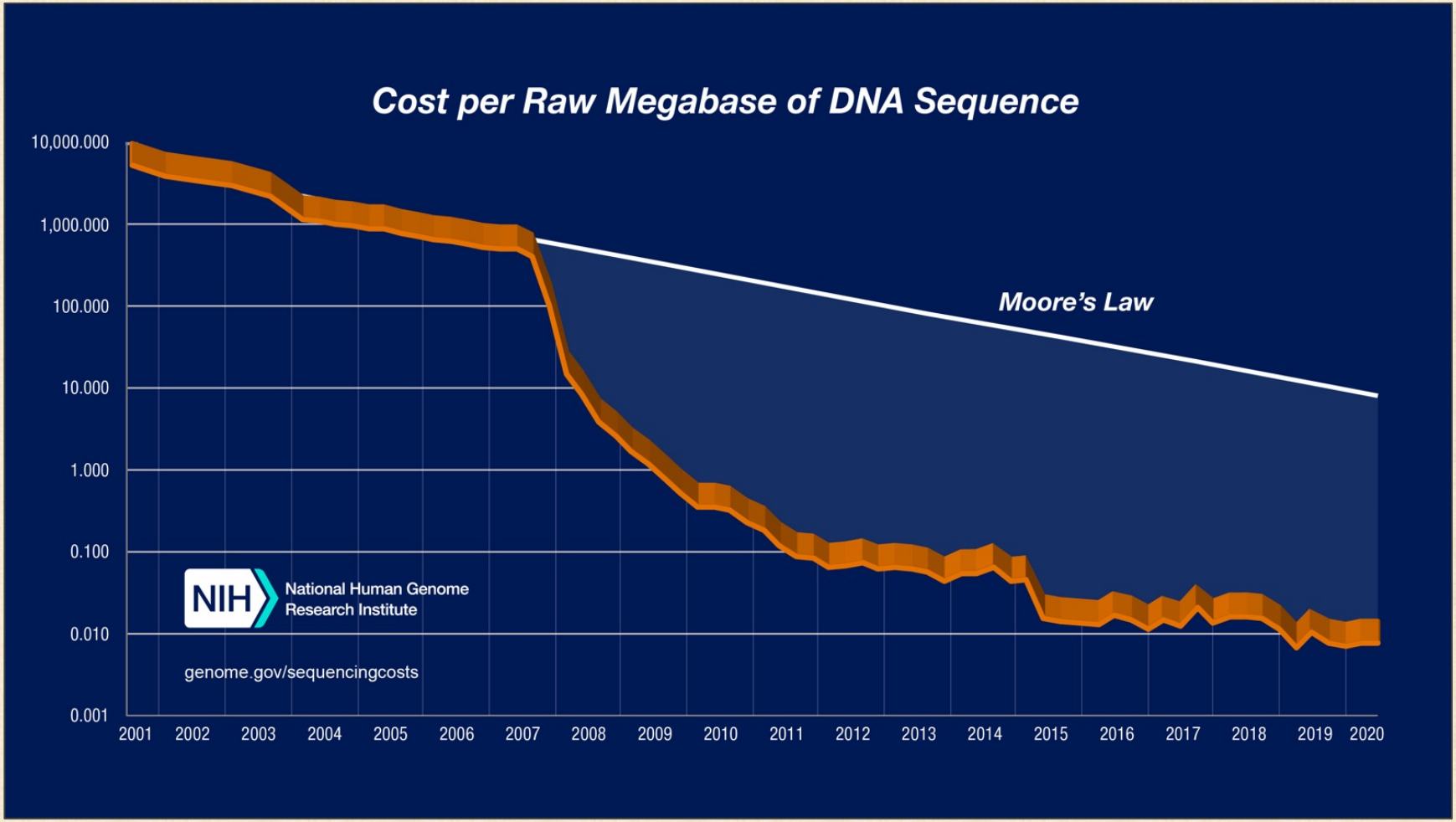


From One Mammal Genome to Many

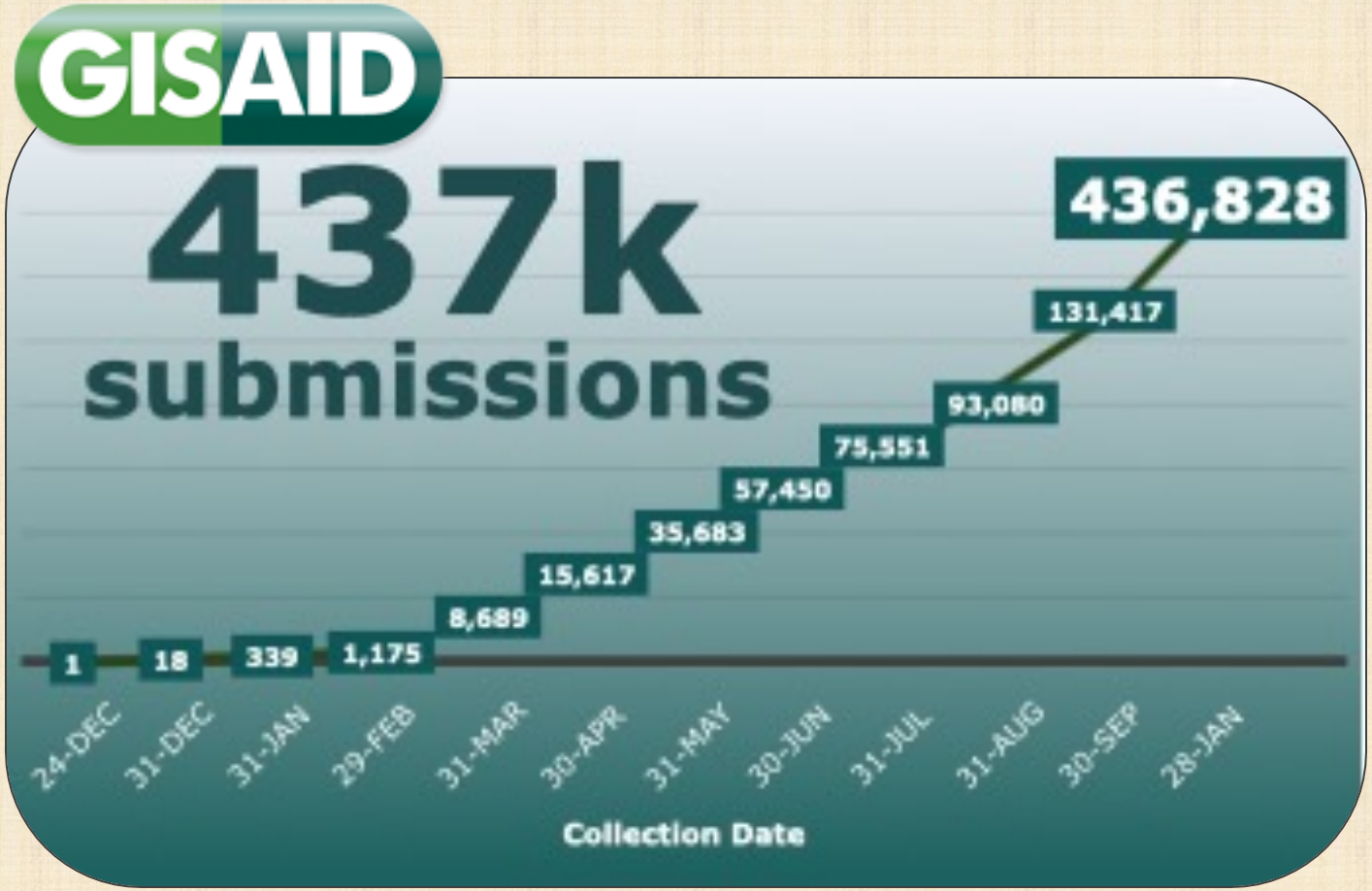
Problem: This approach was just too expensive to scale to thousands of species.



Sequencing Cost Has Fallen Faster than Moore's Law

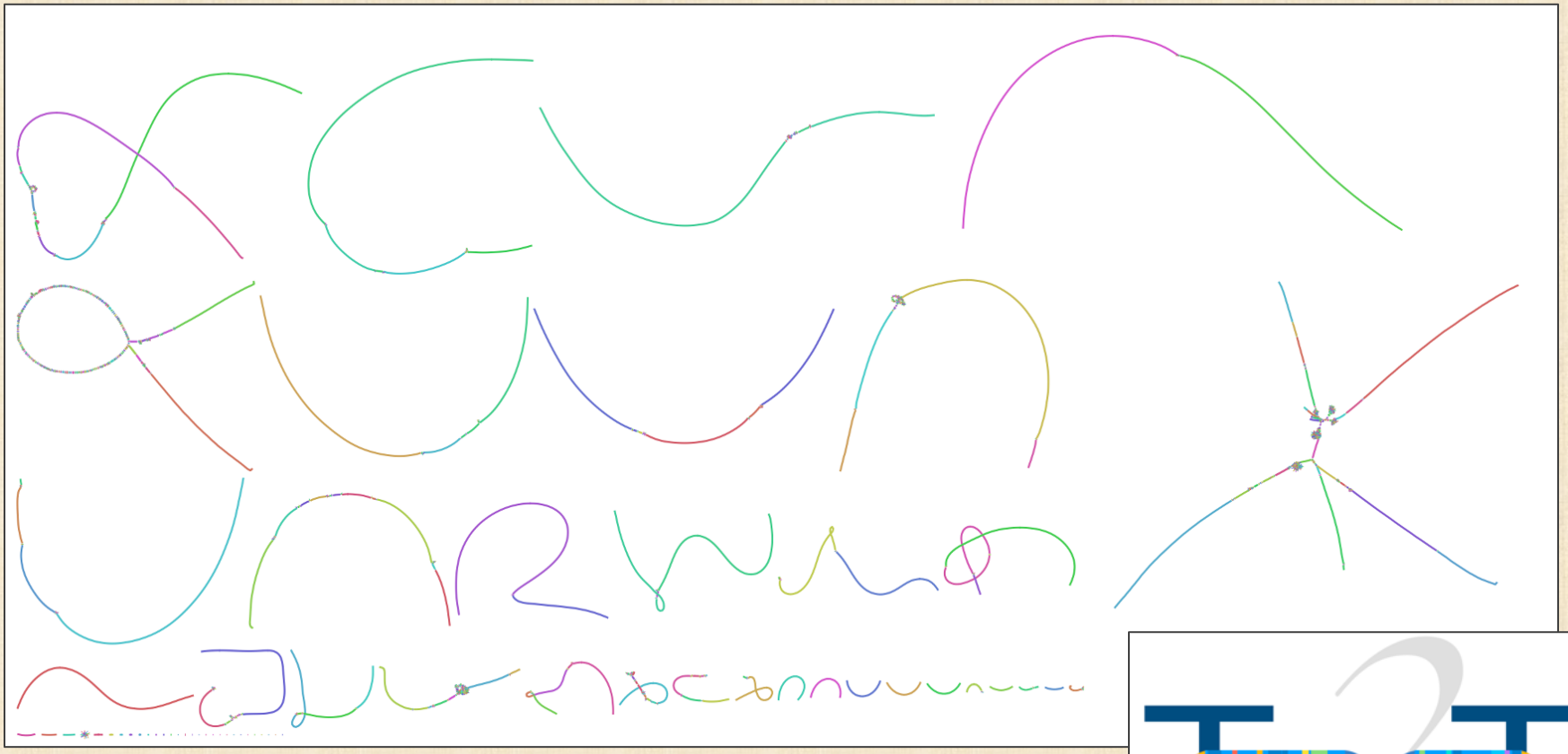


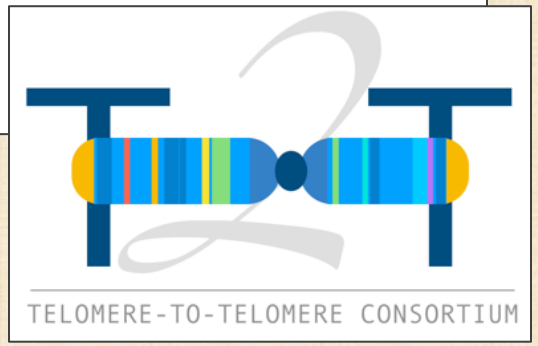
GISAID collects 400k SARS-CoV-2 Genomes in One Year



Scientists aim to sequence 1.5M eukaryotes before 2030

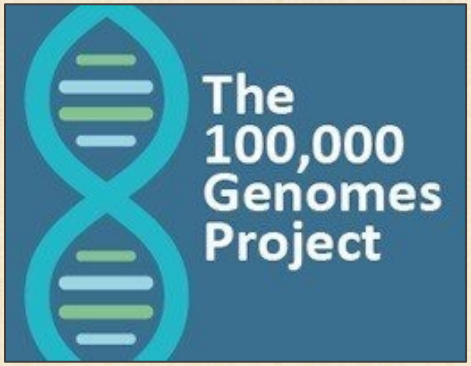
Dark Secret: The First *Full* Human Genome Wasn't Sequenced Until 2020!





We Now Have Over 2 Million Human Genomes

100,000 Genomes: Sequenced 100,000 UK resident genomes (2012-2018).

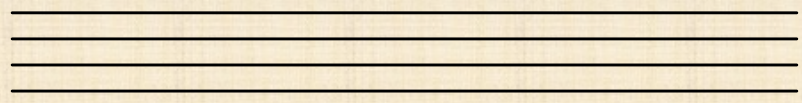


Overview of Genome Sequencing

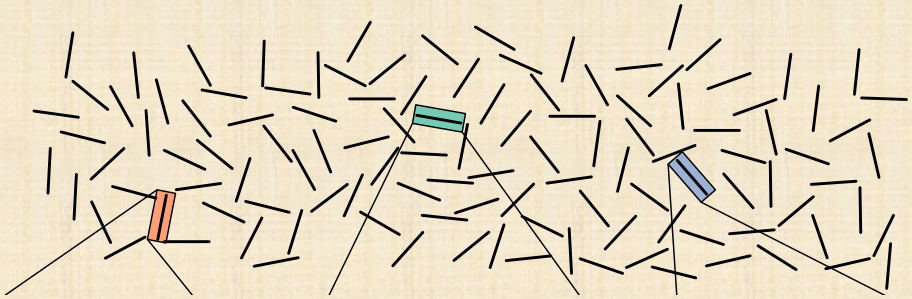
Multiple identical copies of a genome

Overview of Genome Sequencing

Multiple identical copies of a genome



Shatter the genome into reads



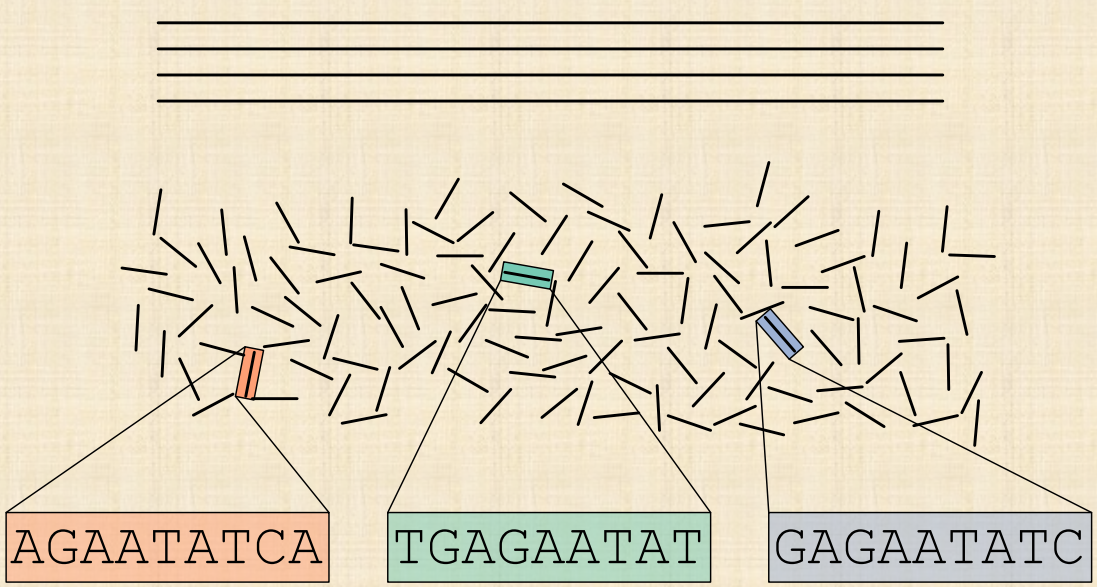
Overview of Genome Sequencing

Multiple identical copies of a genome

Shatter the genome into reads

Sequence the reads

(Lab)



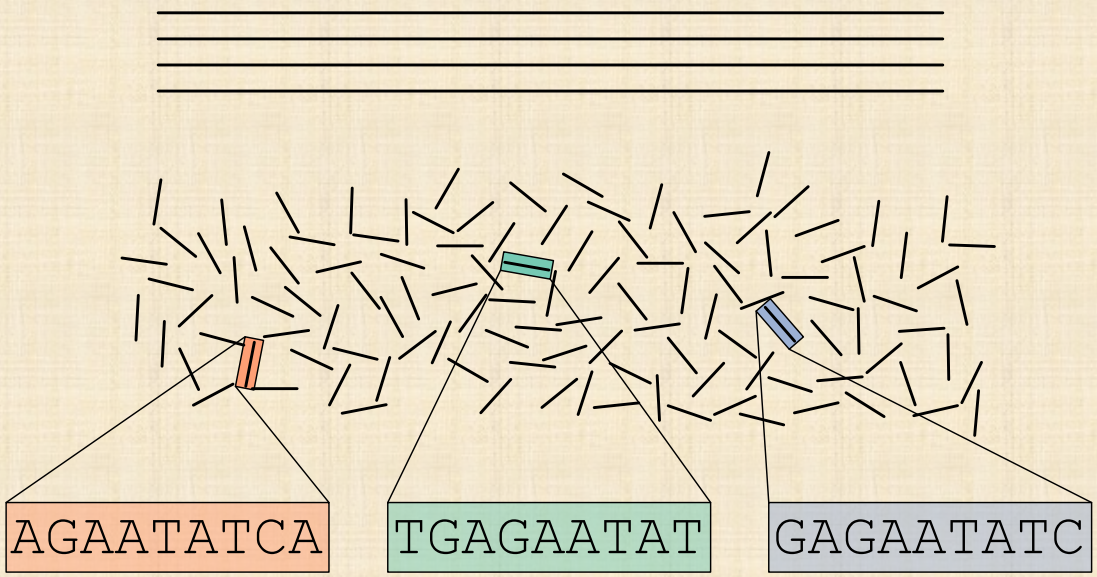
Overview of Genome Sequencing

Multiple identical copies of a genome

Shatter the genome into reads

Sequence the reads (Lab)

Assemble the genome using overlapping reads (Computational)



AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...

Overview of Genome Sequencing

Multiple identical copies of a genome

Shatter the genome into reads

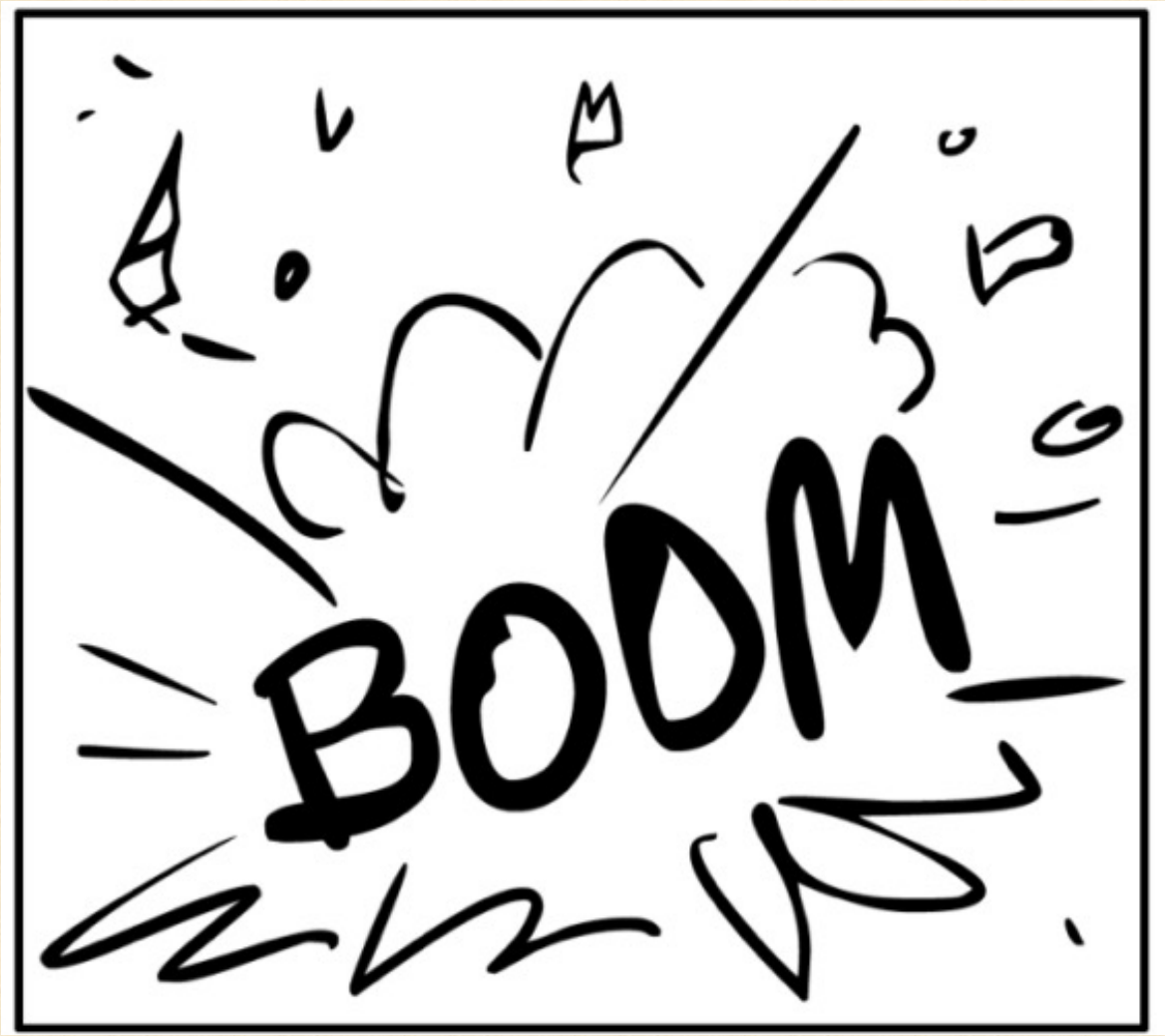
Sequence the reads (Lab)

Assemble the genome using overlapping reads (Computational)

AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...

What does genome sequencing remind you of?

Genome Assembly = Overlap Puzzle



A COMPUTATIONAL PROBLEM FOR GENOME ASSEMBLY

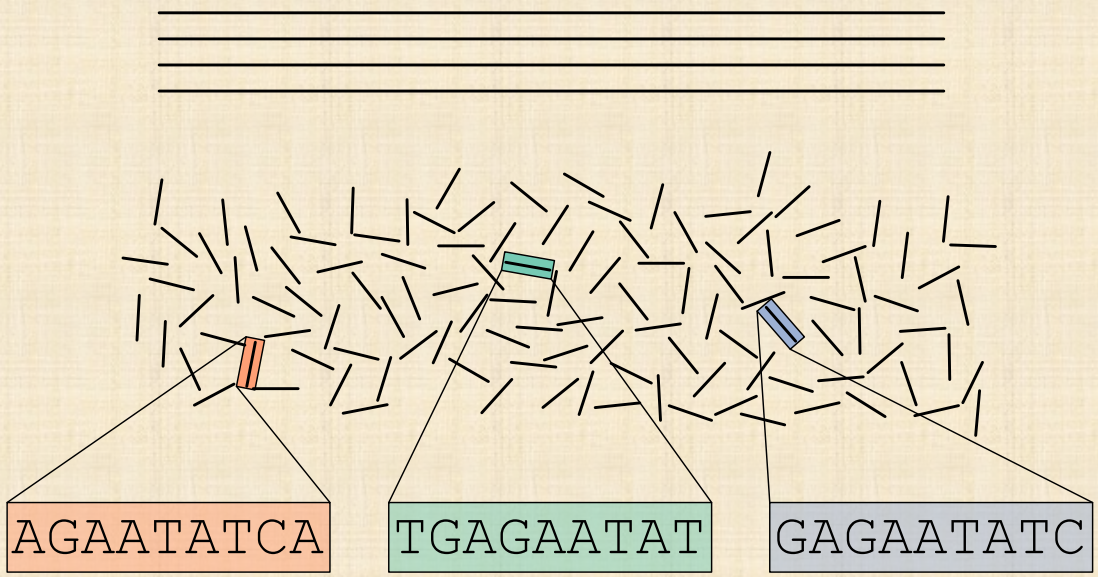
Modeling Our Problem Computationally

Multiple identical copies of a genome

Shatter the genome into reads

Sequence the reads (Lab)

Assemble the genome using overlapping reads (Computational)



AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...

Practical Complications

1. DNA is **double-stranded**.

2. A genome may consist of **multiple chromosomes**.

3. Reads have **imperfect "coverage"** of the underlying genome – not every possible starting position gets sampled by the sequencer.

4. Sequencing machines are **error-prone**.

Making Some Assumptions is OK!

1. DNA is **single-stranded**.

2. A genome consists of a **single chromosome**.

3. Reads have **perfect "coverage"** of the underlying genome – *every* k-mer at every starting position gets sampled by the sequencer.

4. Sequencing machines are **error-free**.

Formulating a Computational Problem for Genome Assembly

Genome Assembly Problem

- **Input:** A collection of strings *Reads*.
- **Output:** A string *Genome* reconstructed from *Reads*.

Formulating a Computational Problem for Genome Assembly

Genome Assembly Problem

- **Input:** A collection of strings *Reads*.
- **Output:** A string *Genome* reconstructed from *Reads*.

STOP: Is this a well-defined problem?

Formulating a Computational Problem for Genome Assembly

Genome Assembly Problem

- **Input:** A collection of strings *Reads*.
- **Output:** A string *Genome* reconstructed from *Reads*.

STOP: Is this a well-defined problem?

Answer: No! We have no sense of what it means to "reconstruct" a genome.

Toward a Computational Problem

The ***k*-mer composition** of a string is its collection of *k*-mers, including repeated substrings.

NANABANANA
NAN
ANA
NAB
ABA
BAN
ANA
NAN
ANA

3-mer composition

Toward a Computational Problem

We want to solve the *reverse* problem: given a collection of strings, find a string having this collection as its *k*-mer composition.

String Reconstruction Problem

- **Input:** A collection of strings *patterns* and an integer *k*.
- **Output:** A string *Text* whose *k*-mer composition is equal to *Patterns*.

Toward a Computational Problem

STOP: Is this a well-defined computational problem? Are we ready to solve it?

String Reconstruction Problem

- **Input:** A collection of strings *patterns* and an integer *k*.
- **Output:** A string *Text* whose *k*-mer composition is equal to *Patterns*.

Toward a Computational Problem

STOP: Is this a well-defined computational problem? Are we ready to solve it?

Answer: Not quite ... what if *Patterns* = {AAA, ZZZ}?

String Reconstruction Problem

- **Input:** A collection of strings *patterns* and an integer *k*.
- **Output:** A string *Text* whose *k*-mer composition is equal to *Patterns* (if such a string exists).

SOLVING THE STRING RECONSTRUCTION PROBLEM?

Toward an Algorithm for Genome Assembly

Exercise: Reconstruct the string corresponding to the following 3-mer composition.

AAT ATG GTT TAA TGT

Toward an Algorithm for Genome Assembly

Exercise: Reconstruct the string corresponding to the following 3-mer composition.

AAT ATG GTT TAA TGT

```
TAA
 AAT
  ATG
   TGT
    GTT
TAATGTT
```

Toward an Algorithm for Genome Assembly

"Greedy" algorithm: for each *k*-mer, look for the *k*-mer of maximum overlap in each direction.

```
TAA
 AAT
  ATG
   TGT
    GTT
TAATGTT
```

Greedy Assembly Algorithm as Pseudocode

```
GreedyAssembler(reads)
        reads2 ← copy of reads
        k ← length of each string in reads2
        genome ← reads2[0]
        reads2 ← remove reads2[0]
        while some read in reads2 has suffix = genome[0, k-1]
                genome ← read[0] + genome
                reads2 ← remove read
        while some read in reads2 has prefix = genome[n–k+1, n]
                // n is length of genome
                genome ← genome + read[length(read) - 1]
                reads2 ← remove read
        return genome
```

Removing an Element from an Array

STOP: Say that we have an array of strings. How can we remove the *i*-th string from the array?

Removing an Element from an Array

STOP: Say that we have an array of strings. How can we remove the *i*-th string from the array?

Answer 1: Create a new array and copy over everything except for the *i*-th string.

Removing an Element from an Array

STOP: Say that we have an array of strings. How can we remove the *i*-th string from the array?

Answer 1: Create a new array and copy over everything except for the *i*-th string.

Answer 2 (better): Use the built-in append operation to append everything after the *i*-th string to everything before.

```
reads2 = append(reads2[:i], reads2[i+1:]…)
```

Implementing Greedy Assembler

```
GreedyAssembler(reads)
        reads2 ← copy of reads
        k ← length of each string in reads2
        genome ← reads2[0]
        reads2 ← remove reads2[0]
        while some read in reads2 has suffix = genome[0, k-1]
                genome ← read[0] + genome
                reads2 ← remove read
        while some read in reads2 has prefix = genome[n–k+1, n]
                // n is length of genome
                genome ← genome + read[length(read) - 1]
                reads2 ← remove read
        return genome
```

Code Challenge: Implement this function.

Let's Test Our Solution Ourselves!

1. Generate a random string.
2. Form the *k*-mer composition of this string.
3. Shuffle the *k*-mers.
4. Apply GreedyAssembler.
5. See if the *k*-mer composition of the resulting string matches the original *k*-mer composition.

First: Generating a Random String

Random DNA String Problem

- **Input:** An integer *n*.
- **Output:** A randomly generated DNA string of length *n*, where each nucleotide has a probability of 0.25 at each position.

Code Challenge: Solve the Random DNA String Problem.

Simple Problems Can Have Big Consequences

It would be easy to solve this problem by concatenating a random symbol each time.

```
func RandomDNAString(n int) string {
    genome := “”
    for i := 0; i < n; i++ {
        genome = genome + string(RandomDNASymbol())
    }
    return genome
}
```

Simple Problems Can Have Big Consequences

It would be easy to solve this problem by concatenating a random symbol each time.

```
func RandomDNAString(n int) string {
    genome := ""
    for i := 0; i < n; i++ {
        genome = genome + string(RandomDNASymbol())
    }
    return genome
}
```

STOP: Let's run this with $n = 3000000$. What's wrong?

Simple Problems Can Have Big Consequences

It would be easy to solve this problem by concatenating a random symbol each time.

```
func RandomDNAString(n int) string {
    genome := “”
    for i := 0; i < n; i++ {
        genome = genome + string(RandomDNASymbol())
    }
    return genome
}
```

Answer: Every time we concatenate strings, a copy of the string is created. This is very slow!

Simple Problems Can Have Big Consequences

Fix: create array of symbols and convert to string at the end.

```
func RandomDNAString(n int) string {
    symbols := make([]byte, n)
    for i := 0; i < length; i++ {
        symbols[i] = RandomDNASymbol()
    }
    return string(symbols)
}
```

Second: Splitting a String into *k*-Mers

The ***k*-mer composition** of a string is its collection of *k*-mers, including repeated substrings.

NANABANANA

```
NAN
 ANA
  NAB
   ABA
    BAN
     ANA
      NAN
       ANA
```

3-mer composition

***k*-Mer Composition Problem**

- **Input:** A string *text* and an integer *k*.
- **Output:** An array containing all *k*-mer substrings of *text*, including repeated strings.

Code Challenge: Solve this problem.

Third: Shuffling the *k*-Mers

String Shuffle Problem

- **Input:** An array of strings *patterns*.
- **Output:** A random "shuffle" of *patterns*.

Hint: The function `rand.Perm()` takes an integer *n* and returns a random *permutation* of the first *n* non-negative integers (e.g., [4, 1, 3, 0, 2]).

func Perm

```
func Perm(n int) []int
```

Perm returns, as a slice of n ints, a pseudo-random permutation of the integers [0,n) from the default Source.

Third: Shuffling the *k*-Mers

String Shuffle Problem

- **Input:** An array of strings *patterns*.
- **Output:** A random "shuffle" of *patterns*.

Hint: The function `rand.Perm()` takes an integer *n* and returns a random *permutation* of the first *n* non-negative integers (e.g., [4, 1, 3, 0, 2]).

Code Challenge: Solve the String Shuffle Problem.

Fourth: Checking Our Solution

We started with a collection of *patterns*. Does our string have this collection as its *k*-mer composition?

String Array Equality Problem:

- **Input:** Two arrays of strings *patterns1* and *patterns2*.
- **Output:** "True" if the collections are the same, and "false" otherwise.

Note that we provide code solving this problem in `helper_functions.go`.

Just One More Thing

Exercise: Apply the greedy algorithm to the 3-mer composition at right by hand. What string do you produce?

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

TAA

AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT

TAA

Issues with Greedy Assembly

TAA

AAT

TAAT

AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
```

STOP: Which one should we choose?

```
TAATG
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
   TGC

TAATGC
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
   TGC
    GCC
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

TAATGCC

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
   TGC
    GCC
     CCA

TAATGCCA
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
   TGC
    GCC
     CCA
      CAT

TAATGCCAT
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
   TGC
    GCC
     CCA
      CAT
       ATG

TAATGCCATG
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
   TGC
    GCC
     CCA
      CAT
       ATG
        TGG

TAATGCCATGG
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
   TGC
    GCC
     CCA
      CAT
       ATG
        TGG
         GGA

TAATGCCATGGA
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
   TGC
    GCC
     CCA
      CAT
       ATG
        TGG
         GGA
          GAT


TAATGCCATGGAT
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

```
TAA
 AAT
  ATG
   TGC
    GCC
     CCA
      CAT
       ATG
        TGG
         GGA
          GAT
           ATG


TAATGCCATGGATG
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

Issues with Greedy Assembly

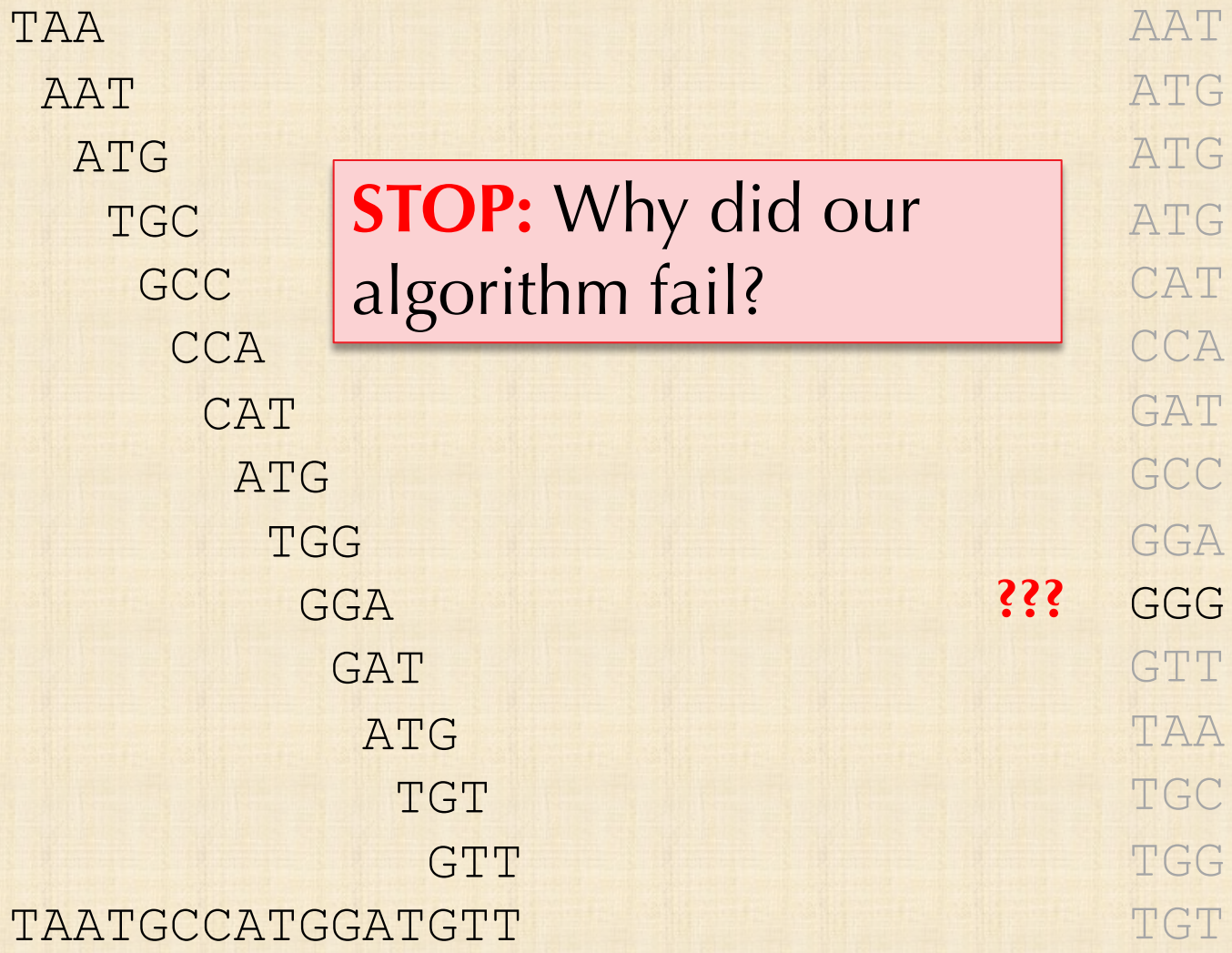
```
TAA
 AAT
  ATG
   TGC
    GCC
     CCA
      CAT
       ATG
        TGG
         GGA
          GAT
           ATG
            TGT

TAATGCCATGGATGT
```

```
AAT
ATG
ATG
ATG
CAT
CCA
GAT
GCC
GGA
GGG
GTT
TAA
TGC
TGG
TGT
```

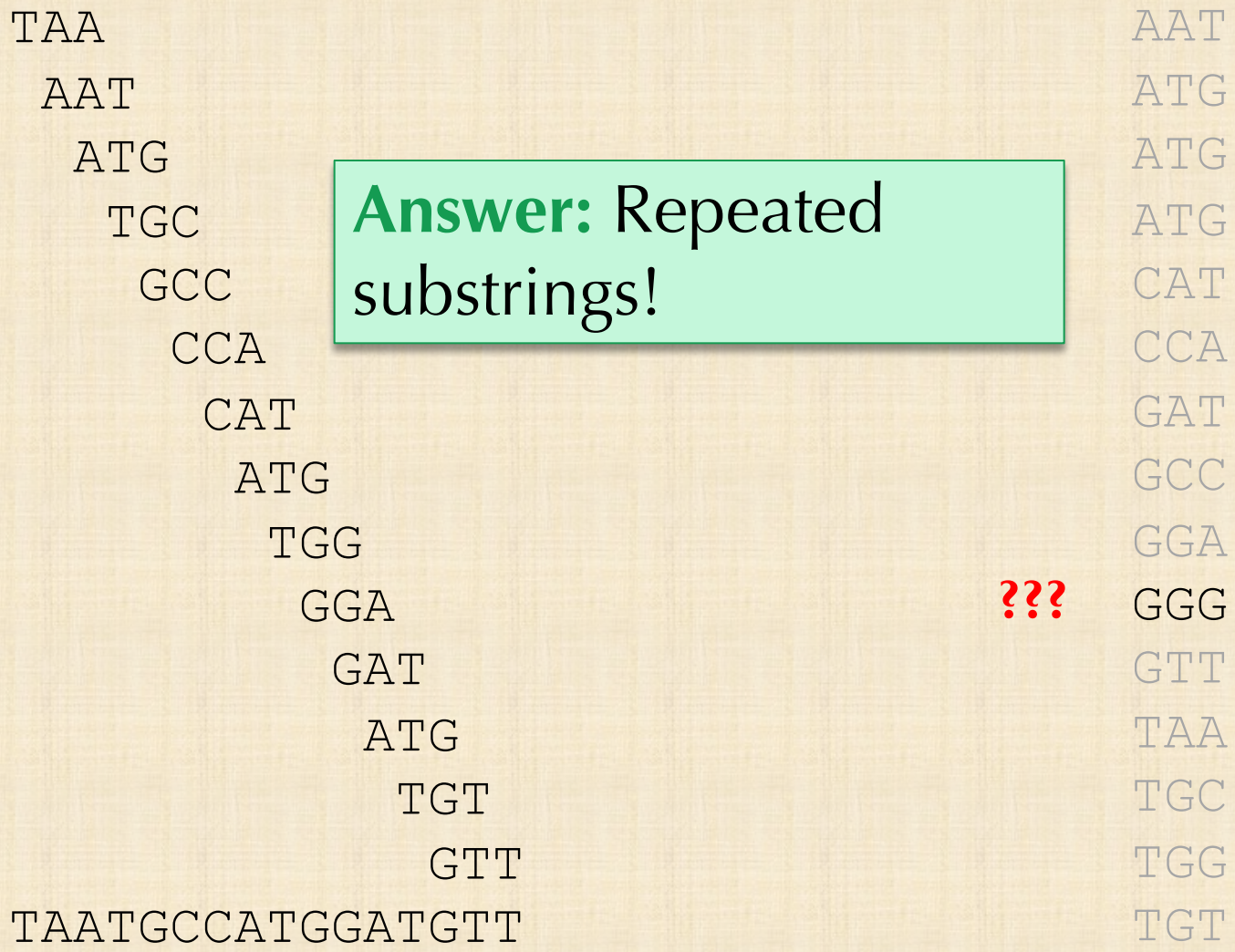
Issues with Greedy Assembly

Issues with Greedy Assembly



STOP: Why did our algorithm fail?

Issues with Greedy Assembly



Answer: Repeated substrings!

Repeats Make Eternity II Unsolvable …

Courtesy: Matej Bat'ha

Even a 16-piece "Triazzle" Can Take a Human Hours to Solve...

... so what hope do we have of assembling a genome?

Courtesy: Dan Gilbert

… and Repeats Complicate Genome Assembly Too ☹

Repeats are very common in genomes; the 300-nucleotide **Alu repeat** occurs over a million times (with minor changes) in every human genome.

So what hope do we have of assembling a genome?

We Haven't Even Addressed Practical Complications

1. DNA is **double-stranded**.

2. A genome may consist of **multiple chromosomes**.

3. Reads have **imperfect "coverage"** of the underlying genome – not every possible starting position gets sampled by the sequencer.

4. Sequencing machines are **error-prone**.

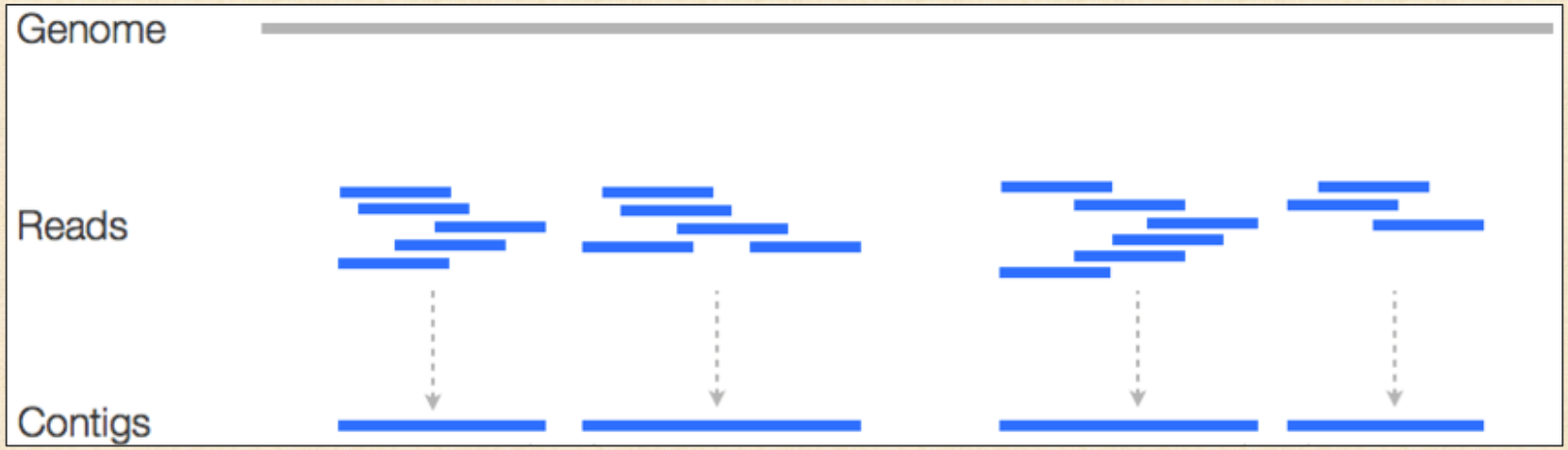
We Need a Heuristic to Work with Real Data …

We will keep the single-strandedness assumption for now but will handle the other practical complications.

Heuristic: a "quick and dirty" approach that doesn't necessarily solve a computational problem but that may provide a good approximate result.

Real Assemblers Produce "Contigs"

Contig: a contiguous substring of DNA that we are confident about in the underlying chromosome.



We Also Need to Handle Errors

	Illumina	PacBio/ Oxford Nanopore	PacBio SMRT Reads
Accuracy	99.9%	~85%	>99.9%
Price	Cheap	Cheap	Expensive
Length	50-300 nt	~10k bp	10k+ bp

STOP: Before, we considered two reads to overlap if they overlapped *perfectly*. How can we use what we have already learned to overlap reads in the presence of errors?

We Also Need to Handle Errors

	Illumina	PacBio/ Oxford Nanopore	PacBio SMRT Reads
Accuracy	99.9%	~85%	>99.9%
Price	Cheap	Cheap	Expensive
Length	50-300 nt	~10k bp	10k-100k bp

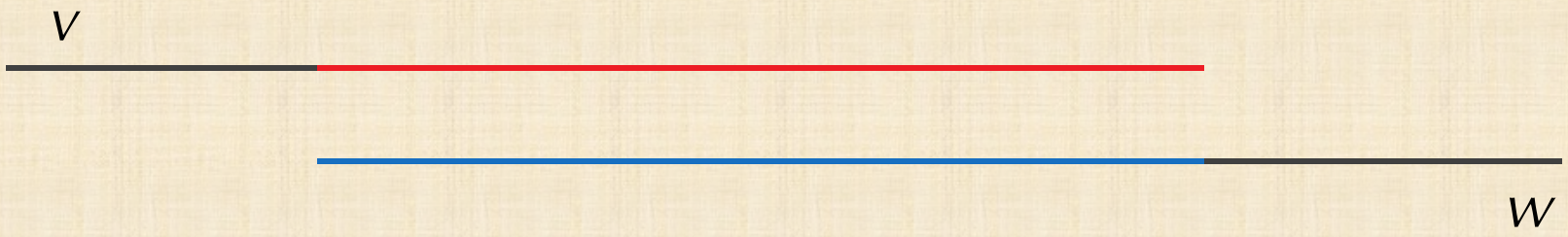
STOP: Before, we considered two reads to overlap if they overlapped *perfectly*. How can we use what we have already learned to overlap reads in the presence of errors?

Answer: We need to use *alignment*!

ADDING ALIGNMENT TO ASSEMBLY

Aligning Overlapping Sequencing Reads

STOP: How could we apply what we have learned about sequence alignment to our genome assembler to compare reads? What problem are we solving?



Formulating an Alignment Problem

Given two strings *v* and *w*, an **overlap alignment** of *v* and *w* is an alignment of a *suffix* of *v* with a *prefix* of *w*.

Overlap Alignment Problem:

- **Input:** Two strings *v* and *w* along with scoring parameters.
- **Output:** A highest-scoring overlap alignment of *v* with *w* over all suffixes of *v* and prefixes of *w*.

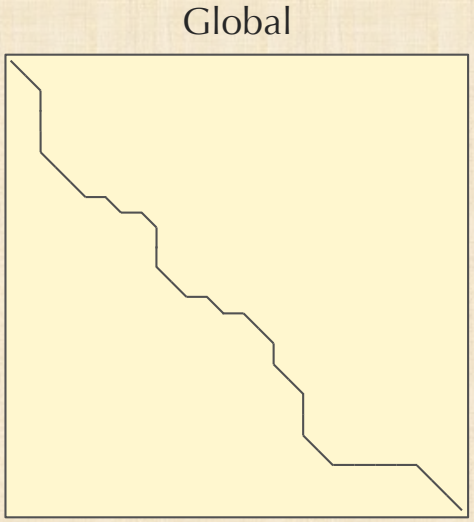
Formulating an Alignment Problem

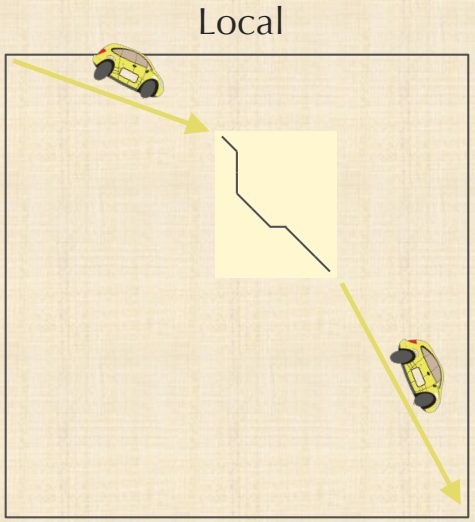
Exercise: How does this problem differ from local alignment? Where are the "free taxi rides" in the alignment network?

Overlap Alignment Problem:

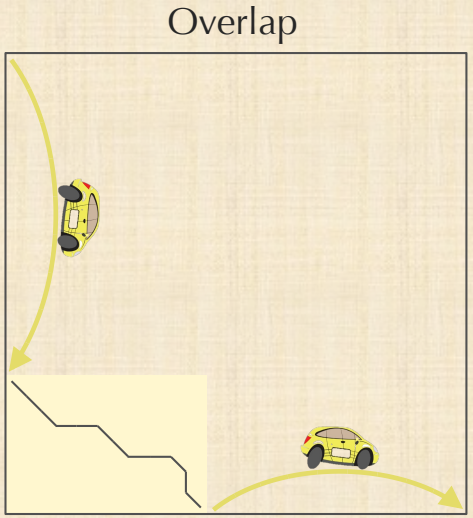
- **Input:** Two strings *v* and *w* along with scoring parameters.
- **Output:** A highest-scoring overlap alignment of *v* with *w* over all suffixes of *v* and prefixes of *w*.

Free Rides for Overlap Alignment

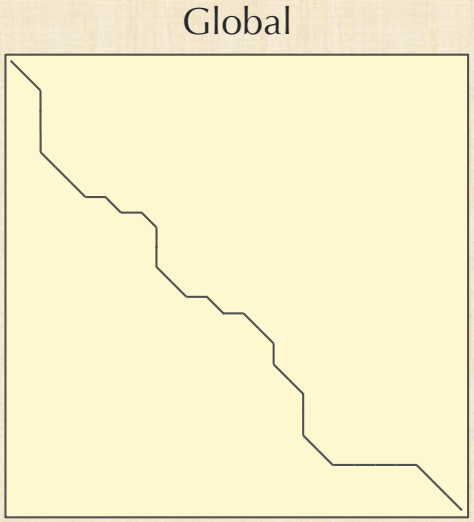


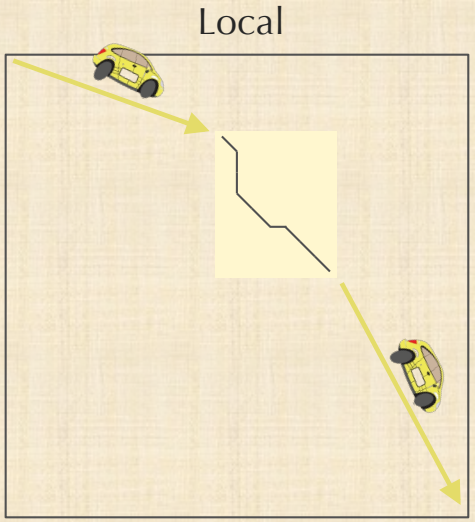


Exercise: Find the recurrence relation(s) for overlap alignment.



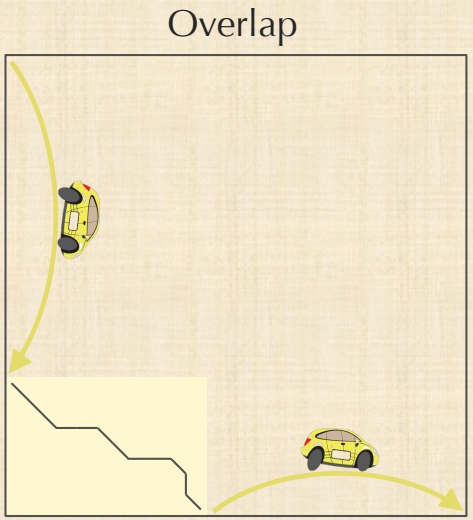
Free Rides for Overlap Alignment





Answer:

- s(*i*,0) = 0.
- other nodes have global alignment recurrence.
- Answer = max of all values in bottom row.



Hacking Overlap Alignment

Overlap Alignment Scoring Problem:

- **Input:** Two strings *v* and *w* along with scoring parameters.
- **Output:** The score of a highest-scoring overlap alignment of *v* with *w* over all suffixes of *v* and prefixes of *w*.

Code Challenge: Solve this problem. Write a subroutine called OverlapScoreTable that computes the 2-D scoring matrix for overlap alignment.

From comparing two reads to many

Overlap Alignment Matrix Problem:

- **Input:** A collection of strings *reads* and scoring parameters.
- **Output:** A matrix S for which $S(i, j)$ is the score of a maximum-score overlap alignment between *reads*[i] and *reads*[j].

STOP: Will the matrix S be symmetric? Why or why not?

From comparing two reads to many

Answer: Not symmetric! Just because the end of *v* overlaps with *w* does not mean the reverse is true…

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	16	87	-47	-90
1	-130	0	-63	72	-30
2	41	-89	0	-19	32
3	-104	-90	61	0	-12
4	-87	46	91	4	0

From comparing two reads to many

Overlap Alignment Matrix Problem:

- **Input:** A collection of strings *reads* and scoring parameters.
- **Output:** A matrix S for which $S(i, j)$ is the score of a maximum-score overlap alignment between $reads[i]$ and $reads[j]$.

Code Challenge: Solve this problem.

Processing our Overlap Matrix

STOP: How can we use this matrix to figure out which strings overlap?

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	16	87	-47	-90
1	-130	0	-63	72	-30
2	41	-89	0	-19	32
3	-104	-90	61	0	-12
4	-87	46	91	4	0

Processing our Overlap Matrix

Answer: We pick a threshold, and only keep pairs of strings whose score is at least equal to the threshold.

threshold = 40

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	16	87	-47	-90
1	-130	0	-63	72	-30
2	41	-89	0	-19	32
3	-104	-90	61	0	-12
4	-87	46	91	4	0

Processing our Overlap Matrix

STOP: How will changing the threshold affect the alignments that we decide to keep?

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	16	87	-47	-90
1	-130	0	-63	72	-30
2	41	-89	0	-19	32
3	-104	-90	61	0	-12
4	-87	46	91	4	0

Processing our Overlap Matrix

Answer: If we lower the threshold, we will keep more pairs of strings.

threshold = 15

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	16	87	-47	-90
1	-130	0	-63	72	-30
2	41	-89	0	-19	32
3	-104	-90	61	0	-12
4	-87	46	91	4	0

Selecting a "Good" Threshold

Exercise: Say we have the following parameters:

- reads of length 100
- $\mu = 1$, $\sigma = 5$
- mismatch error rate of 1% in reads
- indel error rate of 1% in reads

How would you pick a threshold to use? How would your answer change if the reads were longer? What if the error rate were higher?

We will "binarize" our matrix to make it easier to work with

Binarization: converting values to 1 or 0 (in this case based on the threshold).

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	16	87	-47	-90
1	-130	0	-63	72	-30
2	41	-89	0	-19	32
3	-104	-90	61	0	-12
4	-87	46	91	4	0

We will "binarize" our matrix to make it easier to work with

Binarization: converting values to 1 or 0 (in this case based on the threshold).

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	0	1	0	0
1	0	0	0	1	0
2	1	0	0	0	0
3	0	0	1	0	0
4	0	1	1	0	0

We will "binarize" our matrix to make it easier to work with

Overlap Matrix Binarization Problem:

- **Input:** An overlap scoring matrix S for a collection of strings *reads*, along with a threshold value t.
- **Output:** a matrix B in which $B(i, j)$ is 1 if $S(i, j)$ is at least equal to t, and $B(i, j)$ is 0 otherwise.

Just one problem: what if both $S(i,j)$ and $S(j,i)$ are both above the threshold?

STOP: How can we address this issue? Does read 0 overlap with read 2 or vice-versa?

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	16	87	-47	-90
1	-130	0	-63	72	-30
2	41	-89	0	-19	32
3	-104	-90	61	0	-12
4	-87	46	91	4	0

Just one problem: what if both $S(i,j)$ and $S(j,i)$ are both above the threshold?

Answer: If $S(i, j)$ and $S(j, i)$ are both above threshold, only set the larger one to be 1 when we binarize.

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	0	1	0	0
1	0	0	0	1	0
2	0	0	0	0	0
3	0	0	1	0	0
4	0	1	1	0	0

Adjusting our Binarization Problem

Overlap Matrix Binarization Problem:

- **Input:** An overlap scoring matrix S for a collection of strings *reads*, along with a threshold value t.
- **Output:** a matrix B in which $B(i, j)$ is 1 if $S(i, j) \geq t$ and $S(i, j) \geq S(j, i)$, and $B(i, j)$ is 0 otherwise. (If $S(i, j) = S(j, i)$, then set $B(i, j) = 1$ if $j > i$ and set $B(i, j) = 0$ if $i > j$.)

Code Challenge: Solve this problem.

VISUALIZING OVERLAPPING READS WITH THE OVERLAP NETWORK

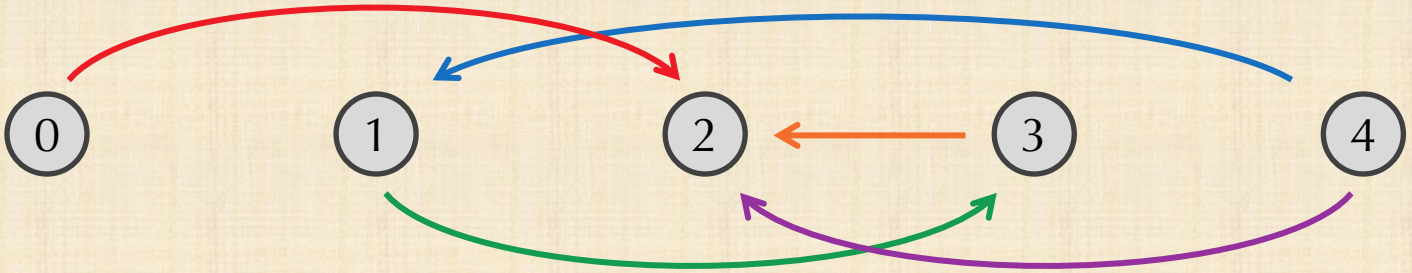
Forming a Network from a Binary Matrix

Overlap network: Each node corresponds to a read; a directed edge connects nodes i to j if $B(i, j) = 1$.

(Hypothetical matrix)

Read	0	1	2	3	4
0	0	0	1	0	0
1	0	0	0	1	0
2	0	0	0	0	0
3	0	0	1	0	0
4	0	1	1	0	0

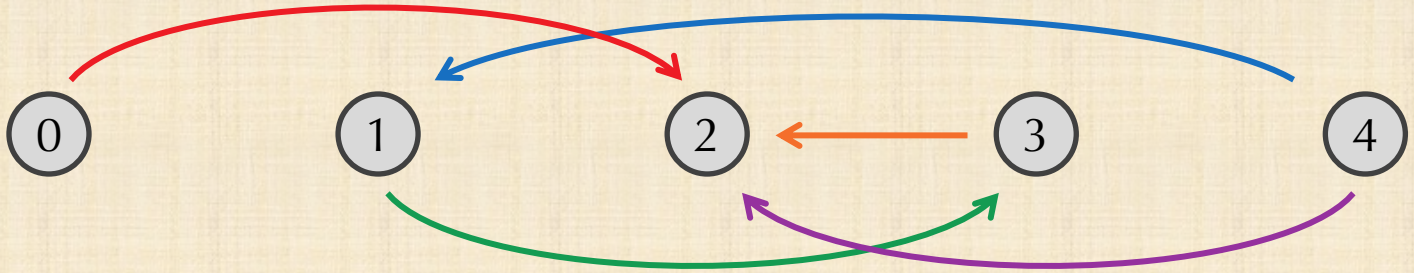
Forming a Network from a Binary Matrix



(Hypothetical matrix)

Read	0	1	2	3	4
0	0	0	1	0	0
1	0	0	0	1	0
2	0	0	0	0	0
3	0	0	1	0	0
4	0	1	1	0	0

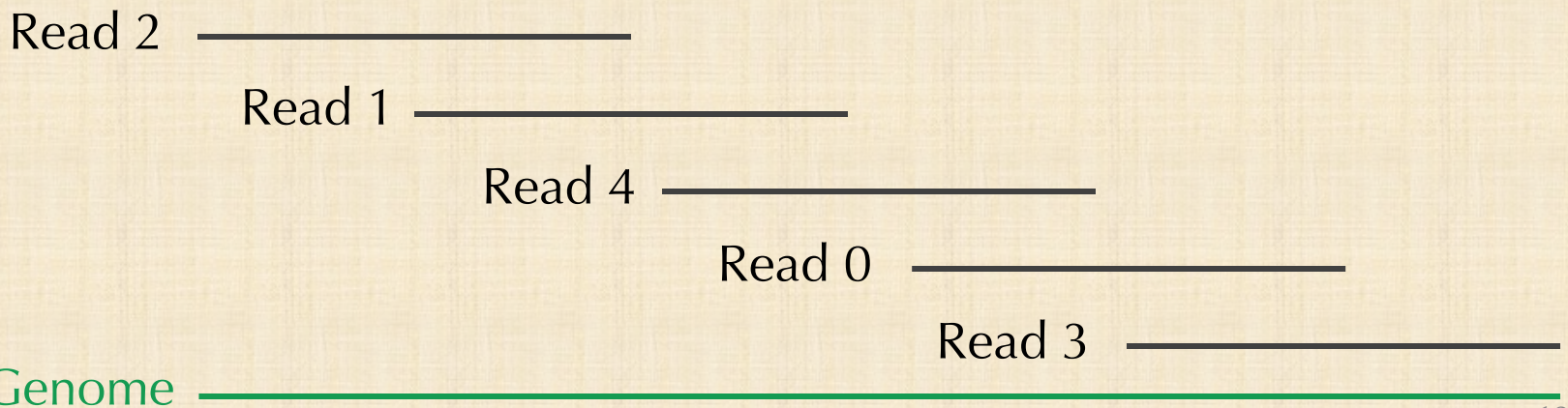
Forming a Network from a Binary Matrix



STOP: Do you think that this is a realistic overlap network? What would we hope to see when we draw an overlap network?

The Best Overlap Network is a Simple Path

If we have a path in which there is one edge in and one edge out of each node, we can start "laying out" the reads and determining the genome.



We Can't Just Tell a Computer to Look at a Picture of a Network ☺

Adjacency list: Each node corresponds to a read; a directed edge connects nodes i to j if $B(i, j) = 1$.

Hypothetical Binary matrix

Reads		0	1	2	3	4
ANANA	0	0	1	0	0	1
ANASB	1	0	0	0	1	0
BANAN	2	1	0	0	0	1
ASBAN	3	0	0	1	0	0
NANAS	4	0	1	0	0	0

Adjacency List

```
ANANA: {ANASB, NANAS}
ANASB: {ASBAN}
BANAN: {ANANA, NANAS}
ASBAN: {BANAN}
NANAS: {ANASB}
```

Forming an Overlap Network Adjacency List from Our Reads

Overlap Network Adjacency List Problem:

- **Input:** A collection *reads* of DNA strings, scoring parameters, and a threshold t.
- **Output:** A map *adjList* (keys = strings, values = slices of strings) such that after producing the binary matrix B with threshold t, if $B(i, j) = 1$, then *reads*[j] is in the slice of strings *adjList*[*reads*[i]].

Code Challenge: Solve this problem.

GENERATING AN OVERLAP NETWORK OF A REAL READ DATASET

Analyzing the Overlap Network

Recall: Once we build the overlap network for a collection of simulated reads, we are hoping that the network is just a single path like the one below.

STOP: In terms of the adjacency list, what would such a graph "look like"?

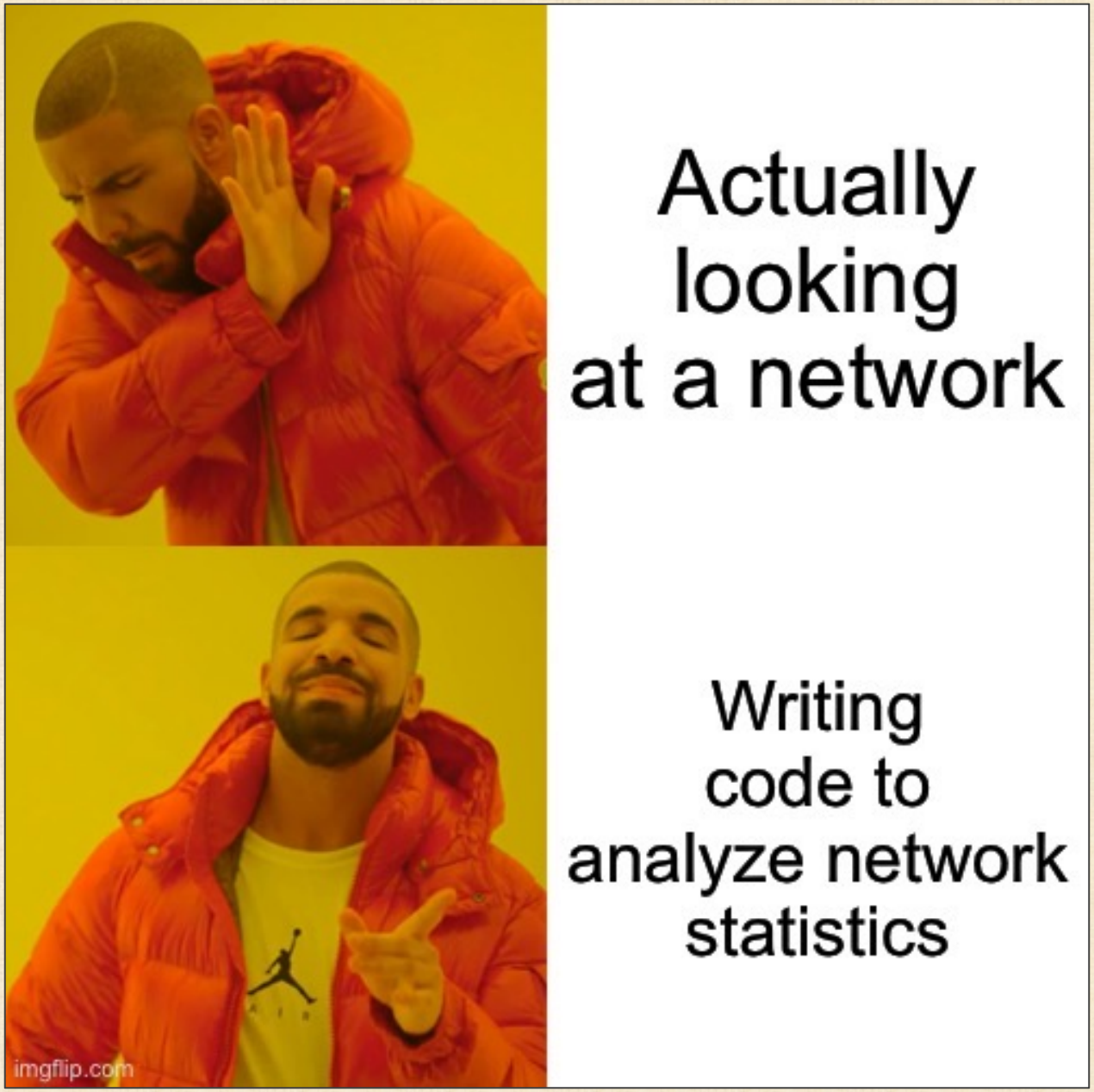
Analyzing the Overlap Network

Recall: Once we build the overlap network for a collection of simulated reads, we are hoping that the network is just a single path like the one below.

Answer: The "outdegree" of each node, or the number of nodes it is connected to, should be at most 1.

Adjacency List

0 : {3}
1 : {4}
2 : {1}
3 : { }
4 : {0}



Analyzing the Overlap Network

The **outdegree** of a node is the number of edges leading out of it.

Average Outdegree Problem:

- **Input:** The adjacency list of an (overlap) network.
- **Output:** The average outdegree of all elements in the adjacency list.

Code Challenge: Solve this problem. There are two versions, depending on whether nodes with outdegree equal to zero contribute to the average.

Let's Build a Network

Let's generate some simulated reads sampled from a known genome and then analyze the overlap network.

We will use the SARS-CoV genome that caused the original 2003 outbreak, since it has a short genome (~30k bp) comparable to SARS-CoV-2.

Sampling Error-Free Reads from a Known Genome

To randomly sample *k*-mers in order to test our algorithm, we will range over a genome, and at every position we will randomly choose the *k*-mer starting at that position with some fixed probability.

Sampling Error-Free Reads from a Known Genome

To randomly sample *k*-mers in order to test our algorithm, we will range over a genome, and at every position we will randomly choose the *k*-mer starting at that position with some fixed probability.

STOP: How many *k*-mers are there in a string of length *n*?

Sampling Error-Free Reads from a Known Genome

To randomly sample *k*-mers in order to test our algorithm, we will range over a genome, and at every position we will randomly choose the *k*-mer starting at that position with some fixed probability.

STOP: How many *k*-mers are there in a string of length *n*?

Answer: $n - k + 1$.

Simulating Clean Reads

Read Generation Problem:

- **Input:** A string *genome* along with integers *readLength* and a decimal *probability*.
- **Output:** An array of strings resulting from taking the substring of length *readLength* at a given position of *genome* with probability *probability*.

Code Challenge: Solve this problem.

Let's apply this to a simulated read dataset!

STOP: Let's apply our approach to simulated reads sampled from the SARS-CoV genome. What happens?

THE MAGIC OF MINIMIZERS

Why Our Algorithm is Slow

STOP: If we look at a typical row of a binarized overlap matrix, what will we see? What does this have to do with our algorithm being slow?

Why Our Algorithm is Slow

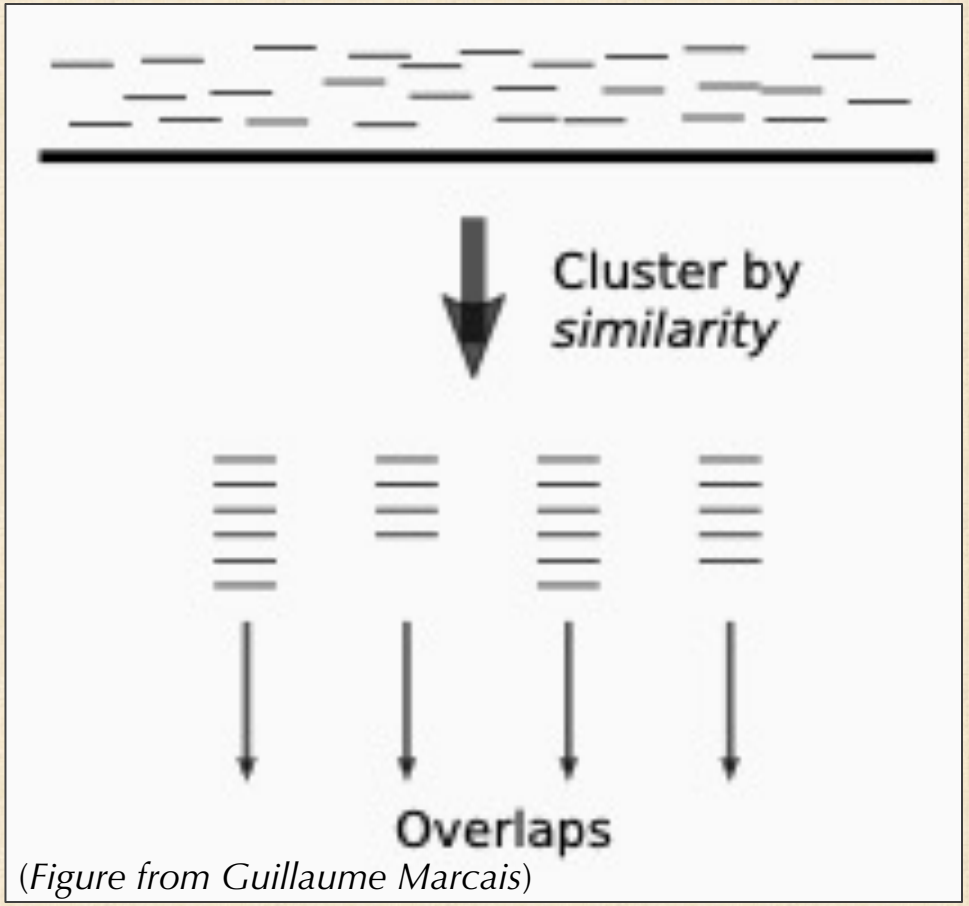
STOP: If we look at a typical row of a binarized overlap matrix, what will we see? What does this have to do with our algorithm being slow?

Answer: Most values in the matrix are 0 because they *don't* overlap. So why are we overlapping all these pairs?

Finding similar strings before overlapping

Idea: make a quick decision at the start to find strings that may be similar.

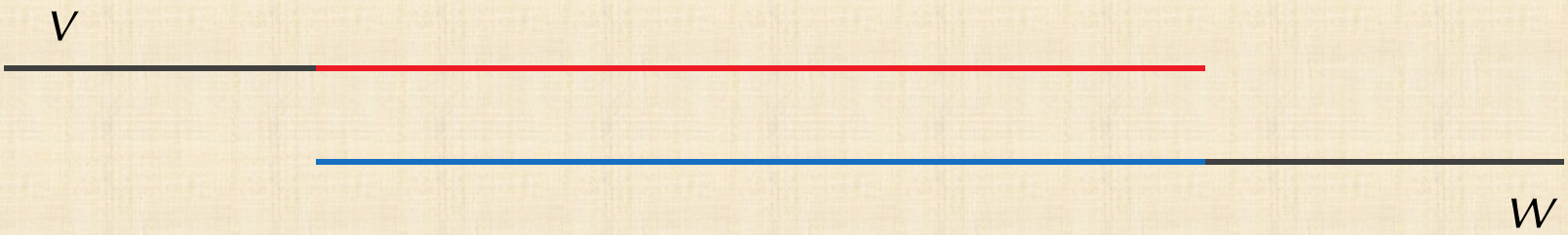
STOP: any ideas on how we could we quickly cluster reads based on similarity?



(*Figure from Guillaume Marcais*)

The Brilliant Insight

The more two strings *overlap*, the more *k*-mers that they will share.

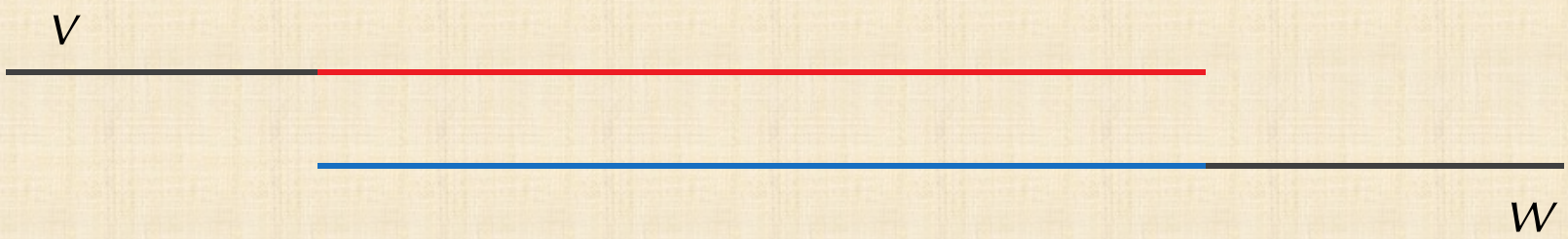


The Brilliant Insight

The more two strings *overlap*, the more *k*-mers that they will share.

Minimizer of `RUINATION` when $k = 3$: `ATI`

Minimizer: The lexicographically smallest *k*-mer in a string.

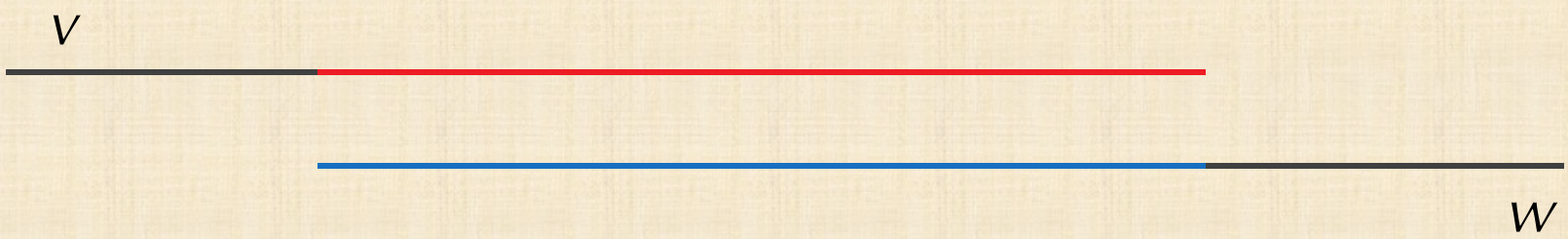


The Brilliant Insight

Key Point: There is a very good chance that the minimizer of *v* is equal to the minimizer of *w*!

Minimizer of `RUINATION` when $k = 3$: `ATI`

Minimizer: The lexicographically smallest *k*-mer in a string.



Solving the Minimizer Problem

Minimizer Problem:

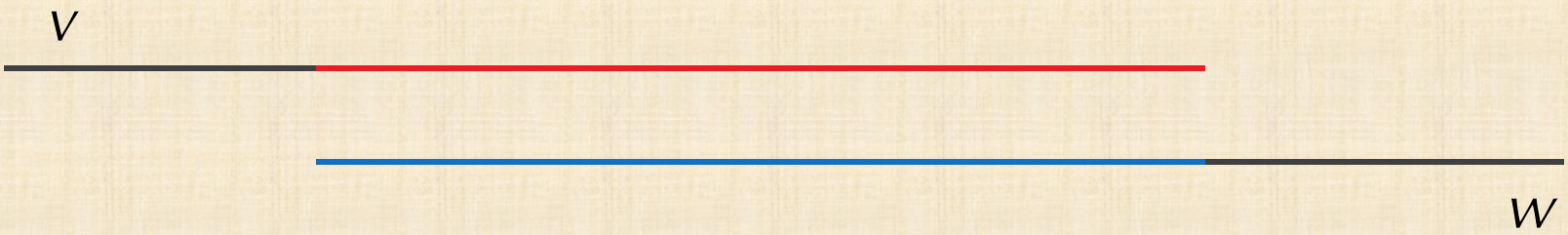
- **Input:** A string *text* and an integer *k*.
- **Output:** The minimizer *k*-mer of *text*.

Minimizer: The lexicographically smallest *k*-mer in a string.

Code Challenge: Solve this problem.

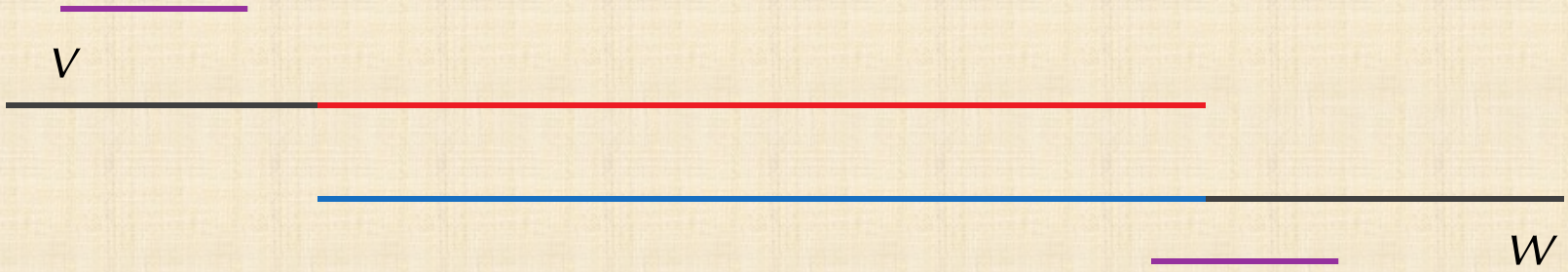
The Issue with Using Minimizers to Find Similarity

STOP: When will *v* and *w* not have the same minimizer?



The Issue with Using Minimizers to Find Similarity

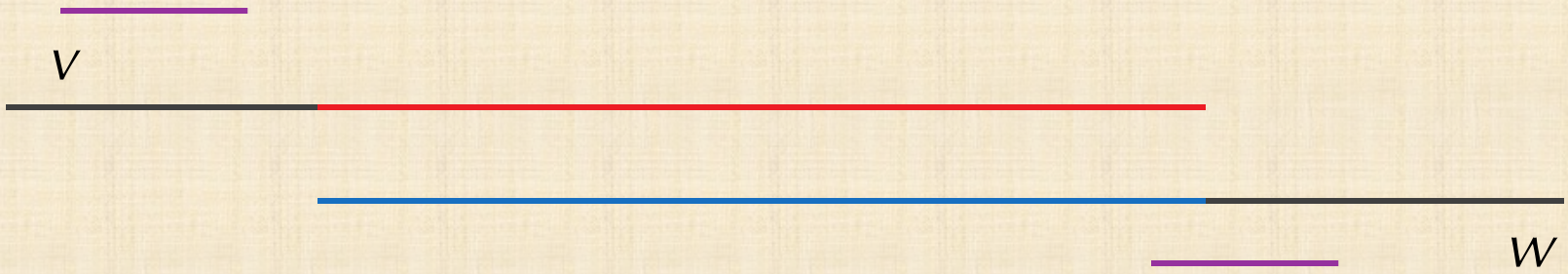
Answer: If the minimizer in *either* string occurs outside the range of what is shared.



The Issue with Using Minimizers to Find Similarity

Answer: If the minimizer in *either* string occurs outside the range of what is shared.

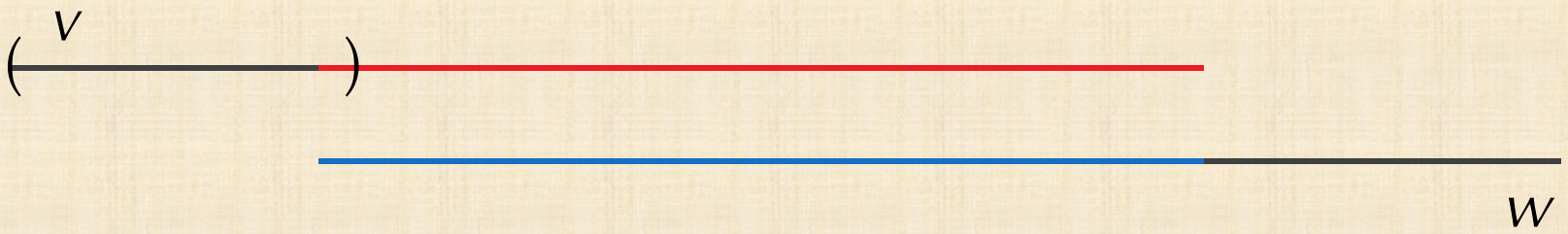
Example: `RUINATION` and `NATIONAL` both share the same minimizer, but `RUINATION` and `NATIONALIZE` don't.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

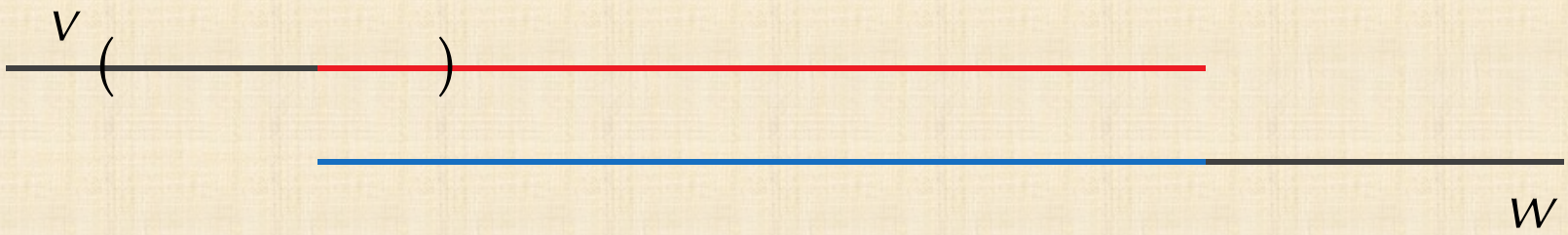
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

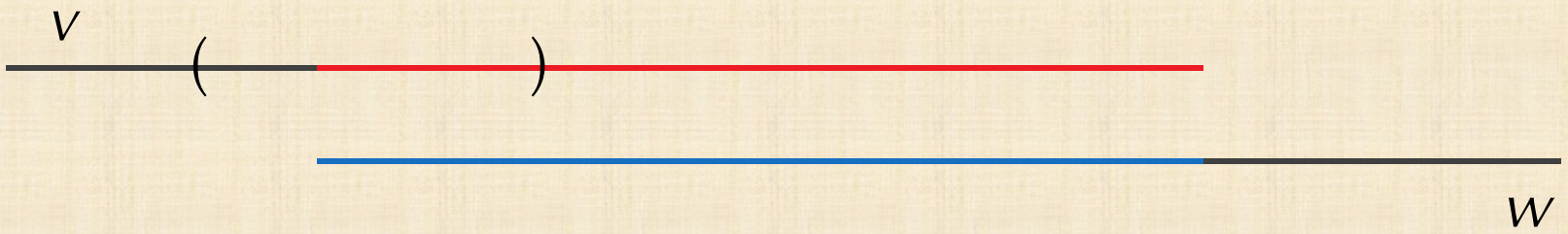
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

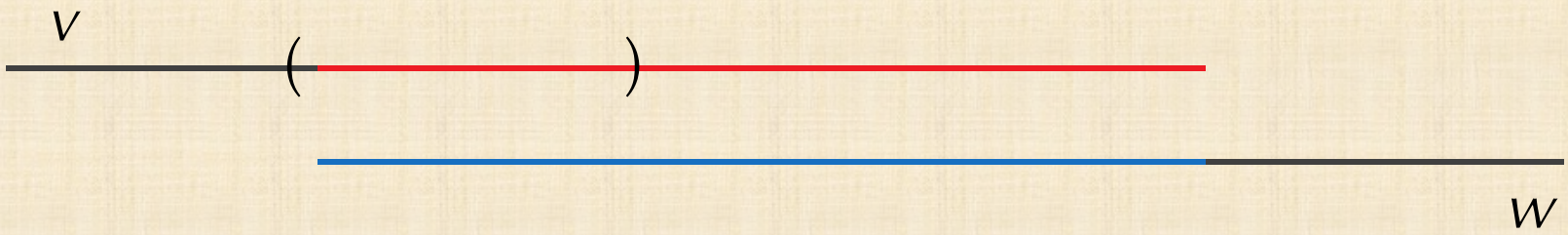
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

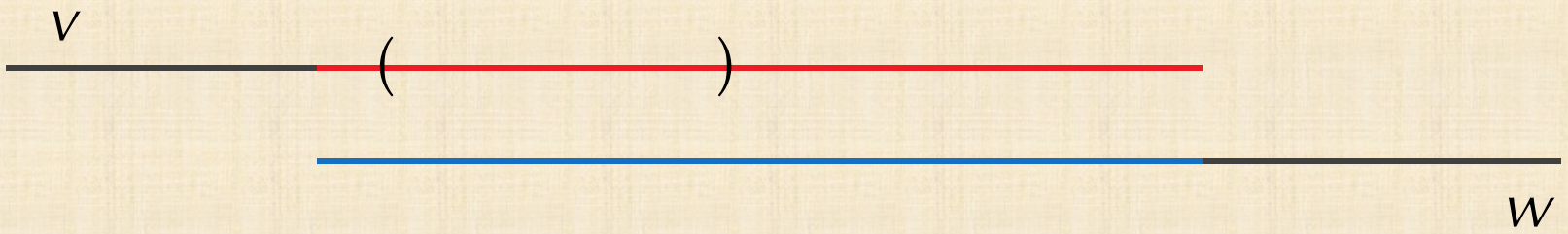
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

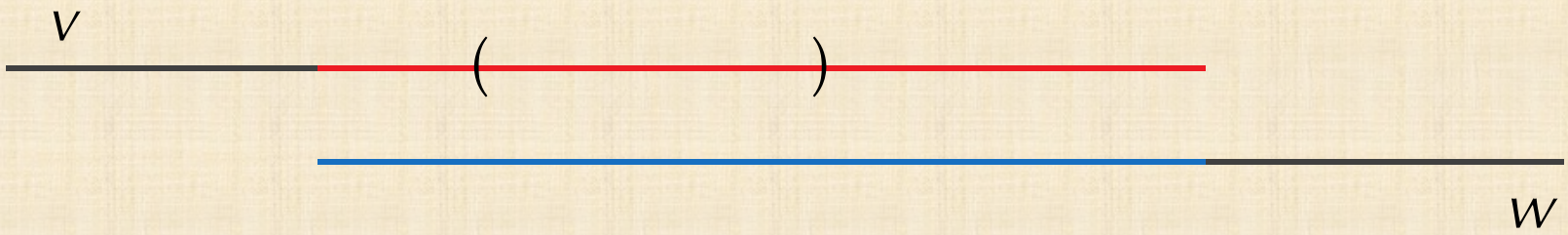
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

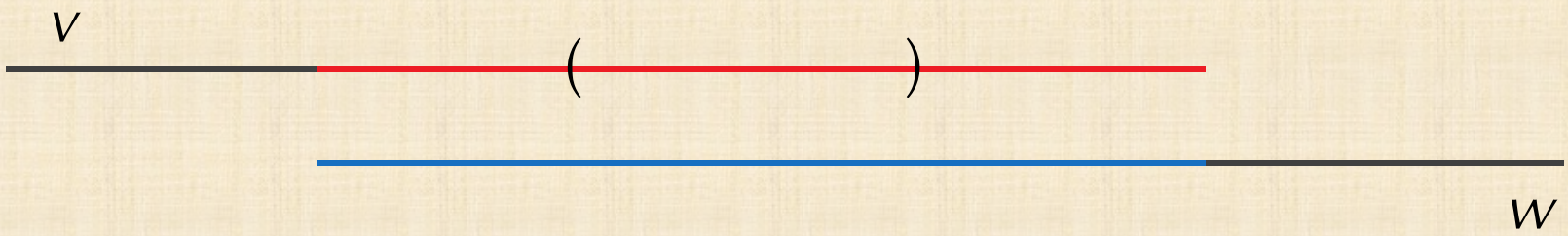
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

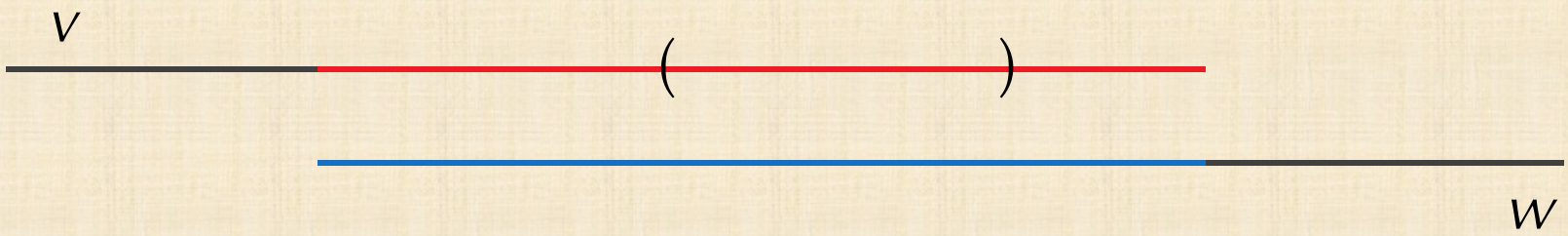
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

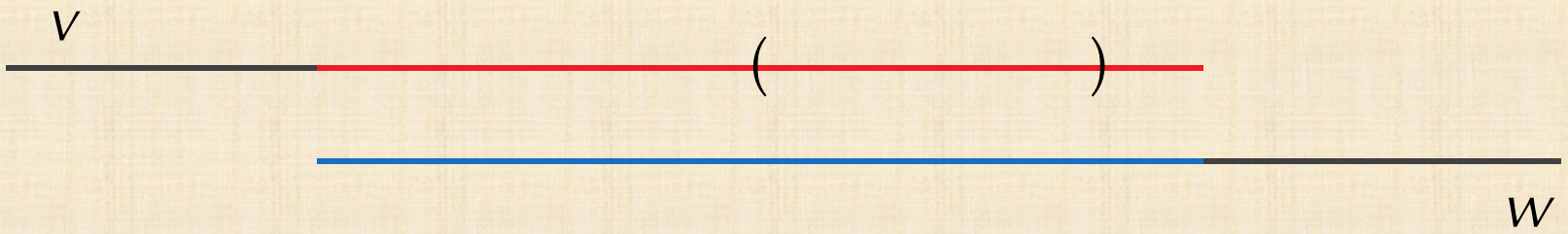
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

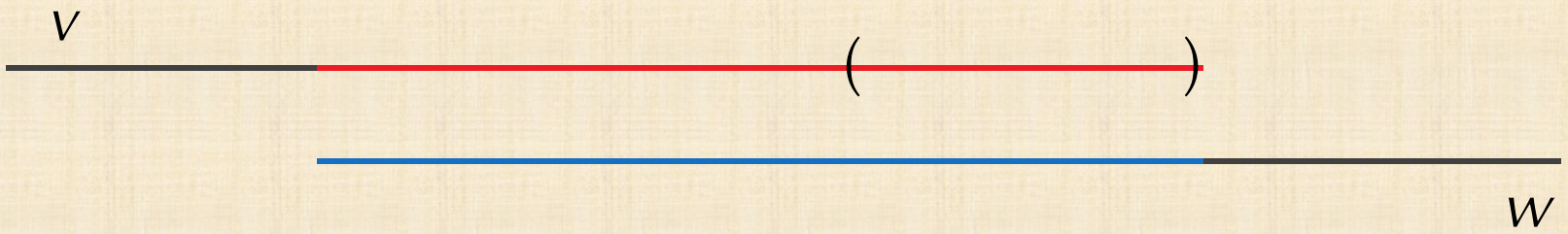
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

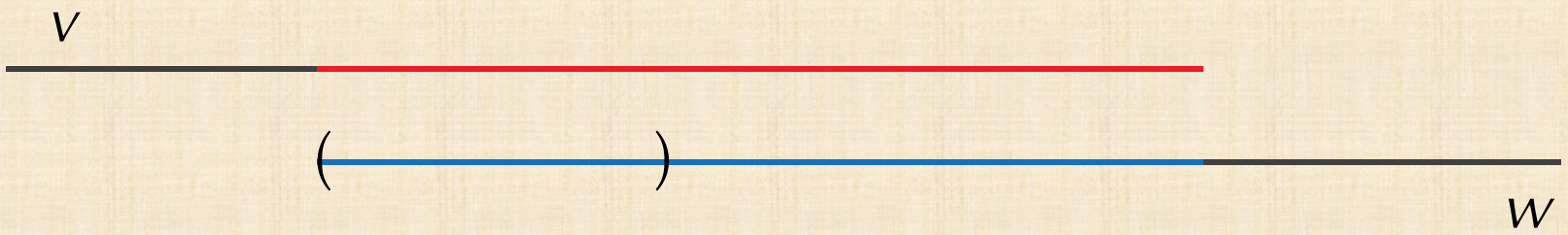
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

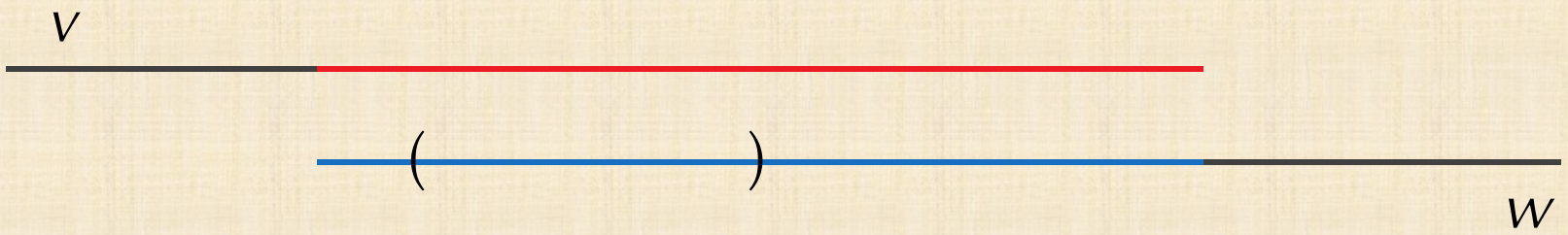
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

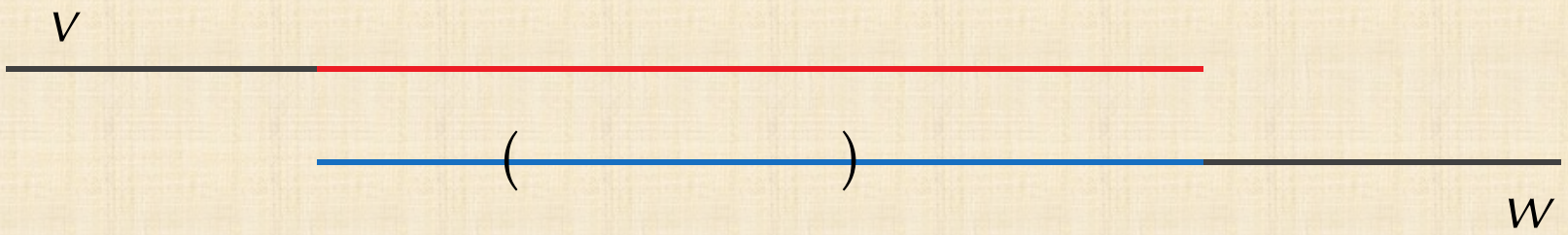
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

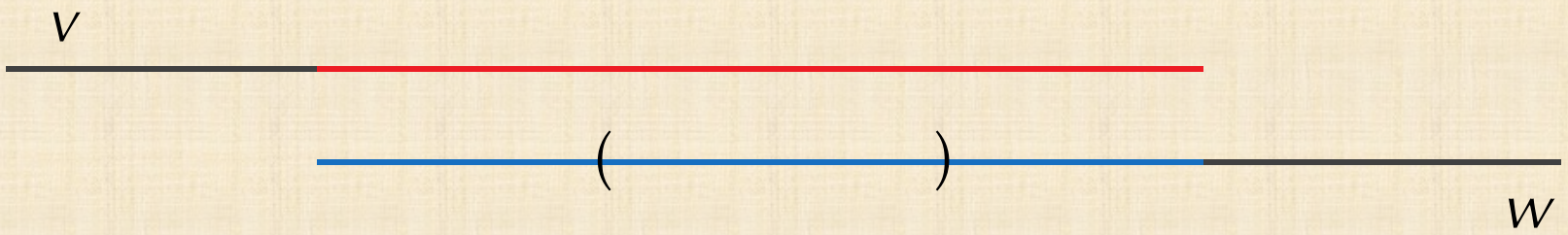
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

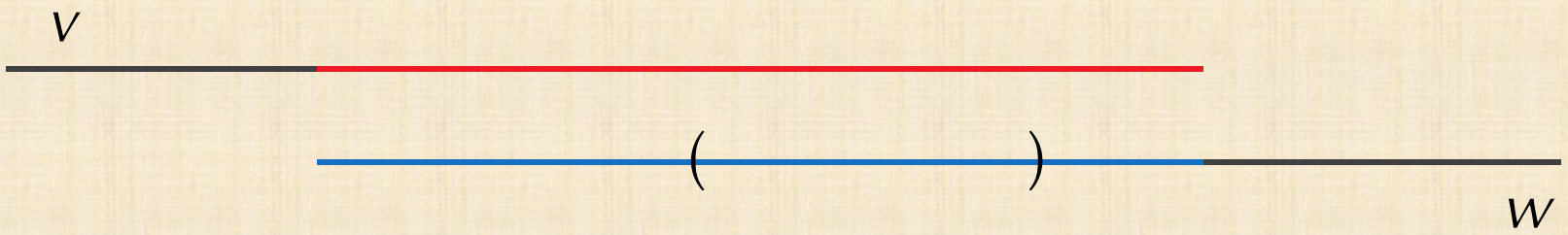
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

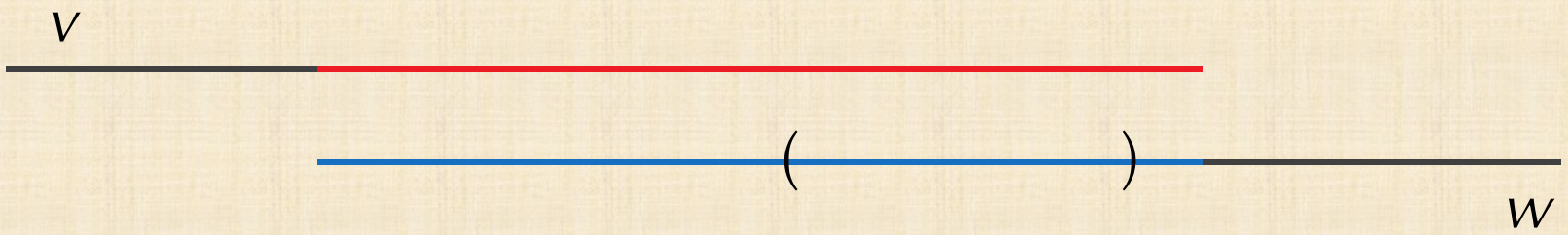
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

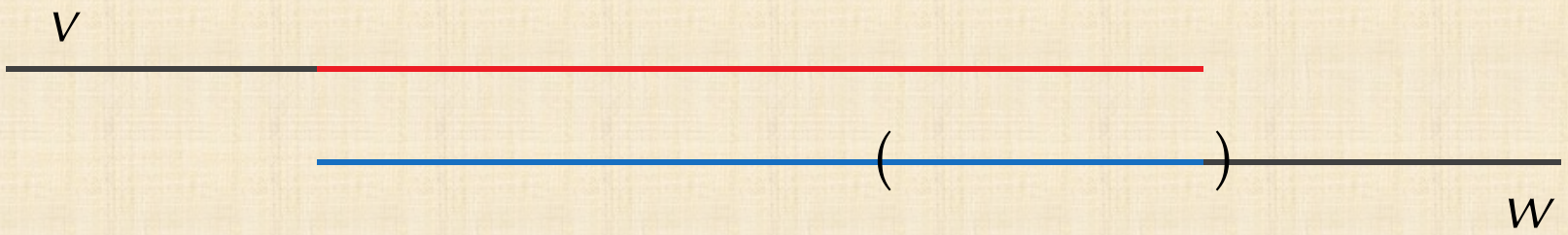
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

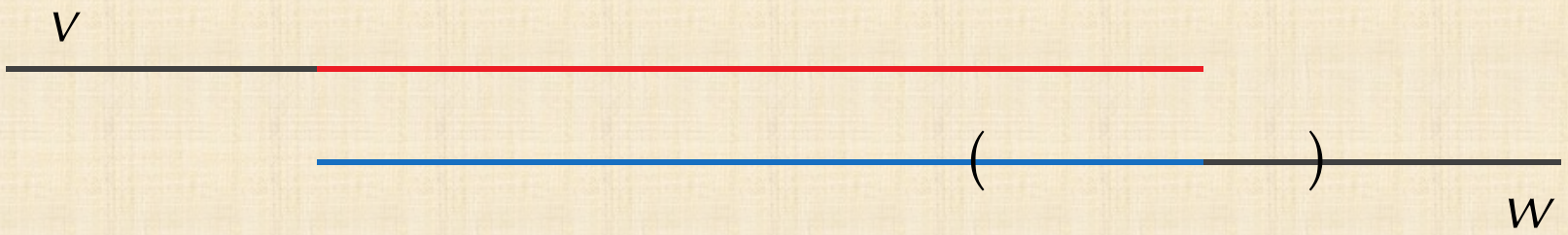
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

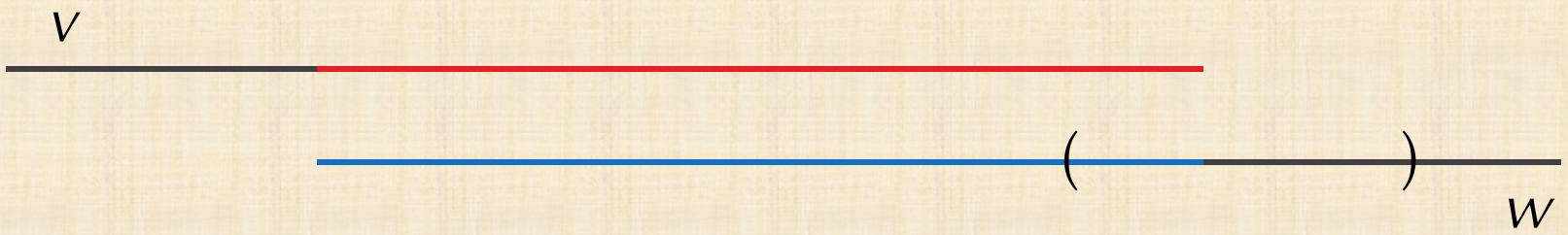
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

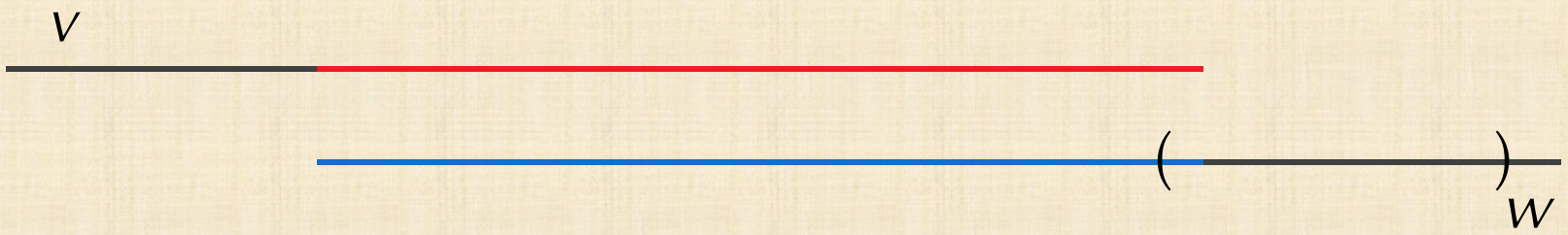
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

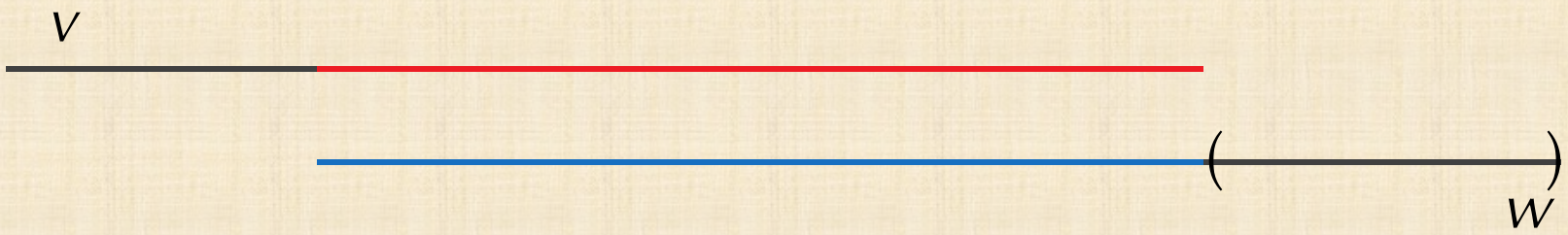
Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Solution: Take *every* minimizer within a given window length

Fix an integer parameter *windowLength*, and then take *all* *k*-mer minimizers of *v* and *w* within substrings of length *windowLength*.

Key point: if *v* and *w* overlap, then they will likely share at least one *k*-mer minimizer.



Returning to our example

Recall that RUINATION and NATIONALIZE don't share the same minimizer. Instead, let's find every 3-mer minimizer of substrings of *windowLength* = 5.

RUINATION

NATIONALIZE

Returning to our example

Recall that RUINATION and NATIONALIZE don't share the same minimizer. Instead, let's find every 3-mer minimizer of substrings of *windowLength* = 5.

RUINATION

RUINA

UINAT

INATI

NATIO

ATION

Minimizers:

ATI, INA

NATIONALIZE

Returning to our example

Recall that RUINATION and NATIONALIZE don't share the same minimizer. Instead, let's find every 3-mer minimizer of substrings of *windowLength* = 5.

```
RUINATION
RUINA
 UINAT
  INATI
   NATIO
    ATION
```

Minimizers:
ATI, INA

```
NATIONALIZE
NATIO
 ATION
  TIONA
   IONAL
    ONALI
     NALIZ
      ALIZE
```

Minimizers:
ALI, ATI, ION

Returning to our example

These strings now share the minimizer ATI! This shared minimizer gives evidence that they overlap.

RUINATION
RUINA
UINAT
INATI
NATIO
ATION

Minimizers:
ATI, INA

NATIONALIZE
NATIO
ATION
TIONA
IONAL
ONALI
NALIZ
ALIZE

Minimizers:
ALI, ATI, ION

From a Good Idea to a Minimizer Map

Minimizer Map: Assigns a string to all read indices having this string as one of its minimizers.

Read index: **147**

ARCHAEBACTERIA

ARCHA
RCHAE
CHAEB
HAEBA
AEBAC
EBACT
BACTE
ACTER
CTERI
TERIA

String	*Indices*
ACT	23, **147**, 230
AEB	14, 68, **147**
ARC	**147**, 200
CHA	24, 57, 68, **147**
CTE	**147**
ERI	7, 70, **147**, 153
etc.	

From a Good Idea to a Minimizer Map

STOP: Which reads do we now want to perform an overlap alignment with this read?

Read index: **147**

ARCHAEBACTERIA

ARCHA
RCHAE
CHAEB
HAEBA
AEBAC
EBACT
BACTE
ACTER
CTERI
TERIA

String	*Indices*
ACT	23, **147**, 230
AEB	14, 68, **147**
ARC	**147**, 200
CHA	24, 57, 68, **147**
CTE	**147**
ERI	7, 70, **147**, 153
etc.	

From a Good Idea to a Minimizer Map

Answer: Anything sharing a minimizer with this read (i.e., reads with indices 23, 230, 14, 68, etc.).

Read index: **147**

ARCHAEBACTERIA

ARCHA
RCHAE
CHAEB
HAEBA
AEBAC
EBACT
BACTE
ACTER
CTERI
TERIA

String	*Indices*
ACT	23, **147**, 230
AEB	14, 68, **147**
ARC	**147**, 200
CHA	24, 57, 68, **147**
CTE	**147**
ERI	7, 70, **147**, 153
etc.	

Example: $k = 2$, *windowLength* $= 5$

	Reads	"Windows"
0	ACCGATC	ACCGA, CCGAT, CGATC
1	AGGTGGG	AGGTG, GGTGG, GTGGG
2	CAGAGCC	CAGAG, AGAGC, GAGCC
3	GGACGTA	GGACG, GACGT, ACGTA
4	TTCGAGG	TTCGA, TCGAG, CGAGG
5	TTGACCA	TTGAC, TGACC, GACCA

Minimizer Map

String	*Indices*
AC	
AG	
AT	
CG	
GG	

Example: $k = 2$, *windowLength* $= 5$

	Reads	"Windows"
0	ACCGATC	ACCGA, CCGAT, CGATC
1	AGGTGGG	AGGTG, GGTGG, GTGGG
2	CAGAGCC	CAGAG, AGAGC, GAGCC
3	GGACGTA	GGACG, GACGT, ACGTA
4	TTCGAGG	TTCGA, TCGAG, CGAGG
5	TTGACCA	TTGAC, TGACC, GACCA

Minimizer Map

String	*Indices*
AC	0
AG	
AT	0
CG	
GG	

Example: $k = 2$, $windowLength = 5$

	Reads	"Windows"
0	ACCGATC	**AC**CGA, CCG**AT**, CG**AT**C
1	AGGTGGG	**AG**GTG, **GG**TGG, GT**GG**G
2	CAGAGCC	CAGAG, AGAGC, GAGCC
3	GGACGTA	GGACG, GACGT, ACGTA
4	TTCGAGG	TTCGA, TCGAG, CGAGG
5	TTGACCA	TTGAC, TGACC, GACCA

Minimizer Map

String	*Indices*
AC	0
AG	1
AT	0
CG	
GG	1

Exercise: Fill in the rest of the map.

Example: $k = 2$, *windowLength* $= 5$

	Reads	"Windows"
0	ACCGATC	**AC**CGA, CCG**AT**, CG**AT**C
1	AGGTGGG	**AG**GTG, **GG**TGG, GT**GG**G
2	CAGAGCC	C**AG**AG, **AG**AGC, G**AG**CC
3	GGACGTA	GGACG, GACGT, ACGTA
4	TTCGAGG	TTCGA, TCGAG, CGAGG
5	TTGACCA	TTGAC, TGACC, GACCA

Minimizer Map

String	*Indices*
AC	0
AG	1, 2
AT	0
CG	
GG	1

Example: $k = 2$, *windowLength* $= 5$

	Reads	"Windows"
0	ACCGATC	**AC**CGA, CCG**AT**, CG**AT**C
1	AGGTGGG	**AG**GTG, **GG**TGG, GT**GG**G
2	CAGAGCC	C**AG**AG, **AG**AGC, G**AG**CC
3	GGACGTA	GG**AC**G, G**AC**GT, **AC**GTA
4	TTCGAGG	TTCGA, TCGAG, CGAGG
5	TTGACCA	TTGAC, TGACC, GACCA

Minimizer Map

String	*Indices*
AC	0, 3
AG	1, 2
AT	0
CG	
GG	1

Example: $k = 2$, *windowLength* $= 5$

	Reads	"Windows"
0	ACCGATC	**AC**CGA, CCG**AT**, CG**AT**C
1	AGGTGGG	**AG**GTG, **GG**TGG, GT**GG**G
2	CAGAGCC	C**AG**AG, **AG**AGC, G**AG**CC
3	GGACGTA	GG**AC**G, G**AC**GT, **AC**GTA
4	TTCGAGG	TT**CG**A, TCG**AG**, CG**AG**G
5	TTGACCA	TTGAC, TGACC, GACCA

Minimizer Map

String	*Indices*
AC	0, 3
AG	1, 2, 4
AT	0
CG	4
GG	1

Example: $k = 2$, *windowLength* $= 5$

	Reads	"Windows"
0	ACCGATC	**AC**CGA, CCG**AT**, CG**AT**C
1	AGGTGGG	**AG**GTG, **GG**TGG, GT**GG**G
2	CAGAGCC	C**AG**AG, **AG**AGC, G**AG**CC
3	GGACGTA	GG**AC**G, G**AC**GT, **AC**GTA
4	TTCGAGG	TT**CG**A, TCG**AG**, CG**AG**G
5	TTGACCA	TTG**AC**, TG**AC**C, G**AC**CA

Minimizer Map

String	*Indices*
AC	0, 3, 5
AG	1, 2, 4
AT	0
CG	4
GG	1

Building a Minimizer Map

Minimizer Map Problem:

- **Input:** A collection of strings *reads* and integers *k* and *windowLength*.
- **Output:** The minimizer map of *k*-mers of *reads* corresponding to considering the *k*-mer minimizer of every substring of length *windowLength* of each string from *reads*.

Code Challenge: Solve this problem.

Using the Minimizer Map for Assembly

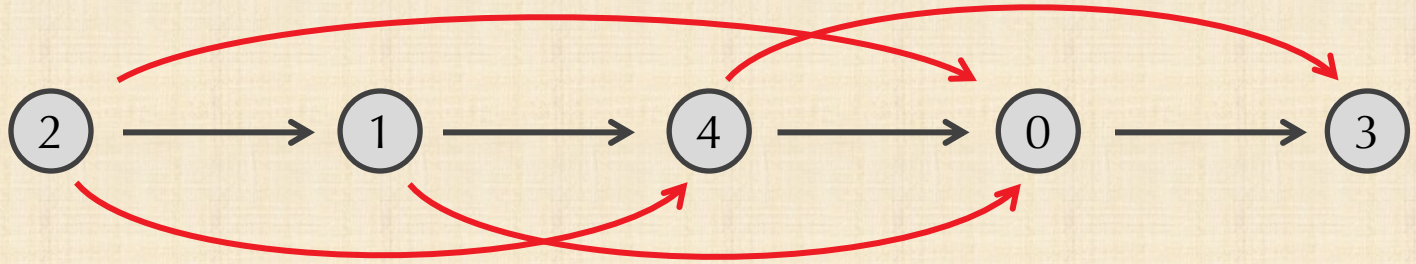
Now we are ready to apply minimizers to our assembly and see if they improve the efficiency of the overlap graph construction!

Note: If we don't overlap two reads, we will set their overlap score to a big negative number.

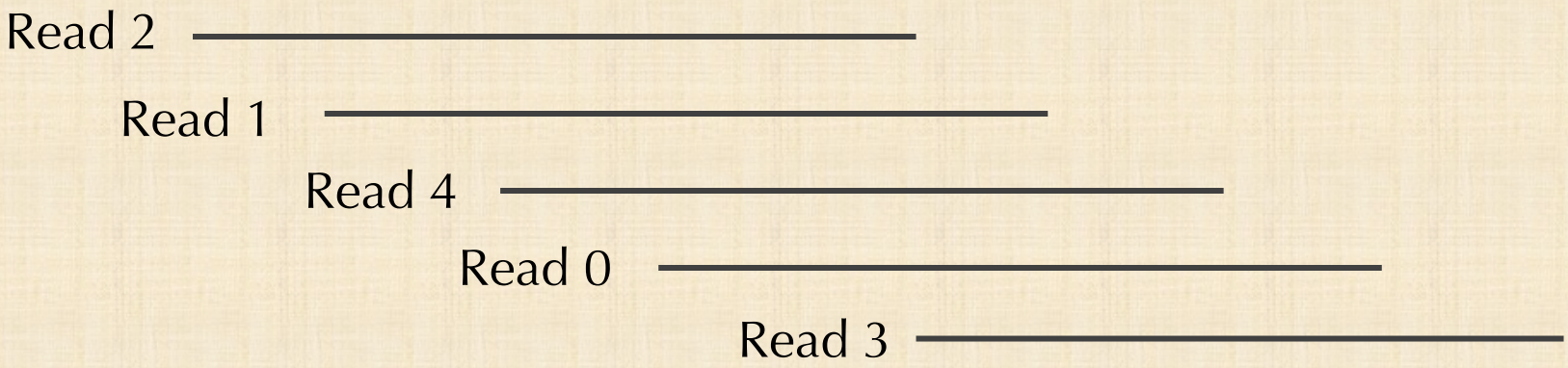
STOP: What is our average outdegree? What does this mean?

TRIMMING THE OVERLAP NETWORK

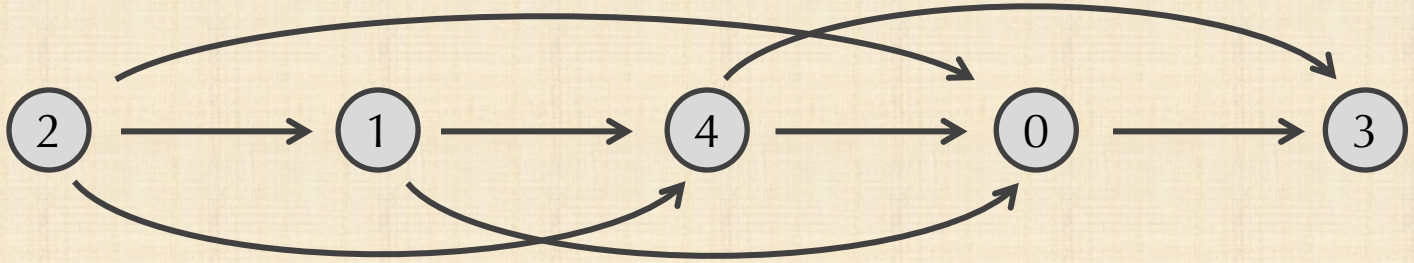
Real Overlap Networks are Tangled



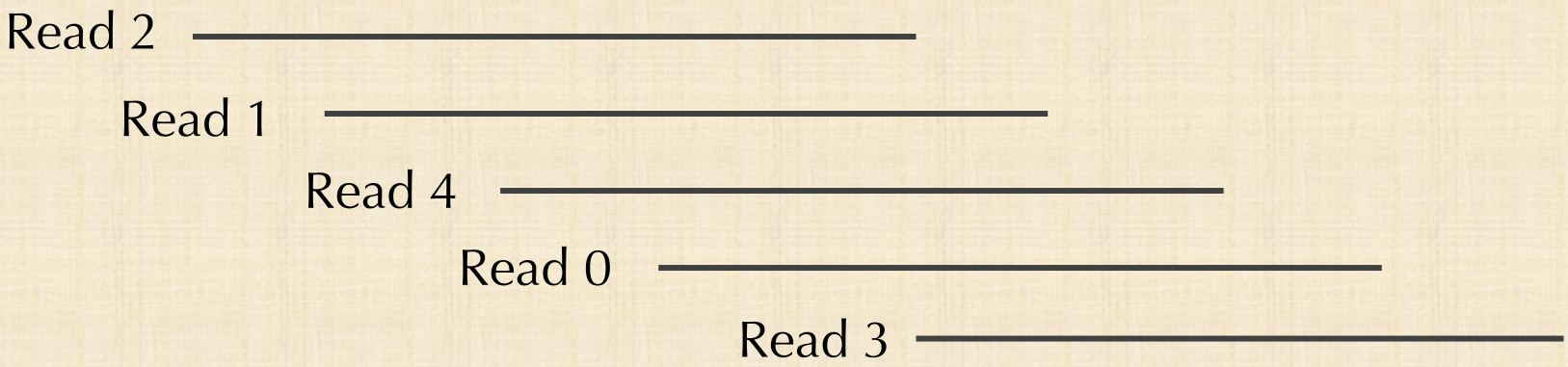
Key Point: If *x* and *y* overlap really well, and *y* and *z* overlap really well, then *x* and *z* will probably overlap really well too.



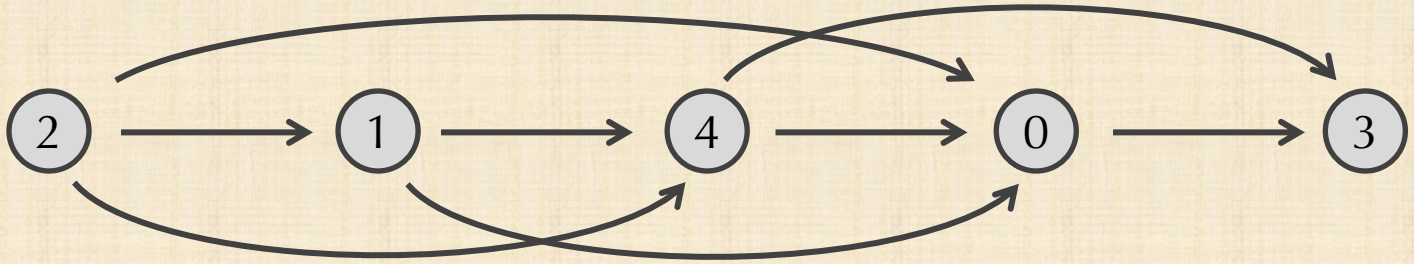
The Reality of the Overlap Network is a Bit Different



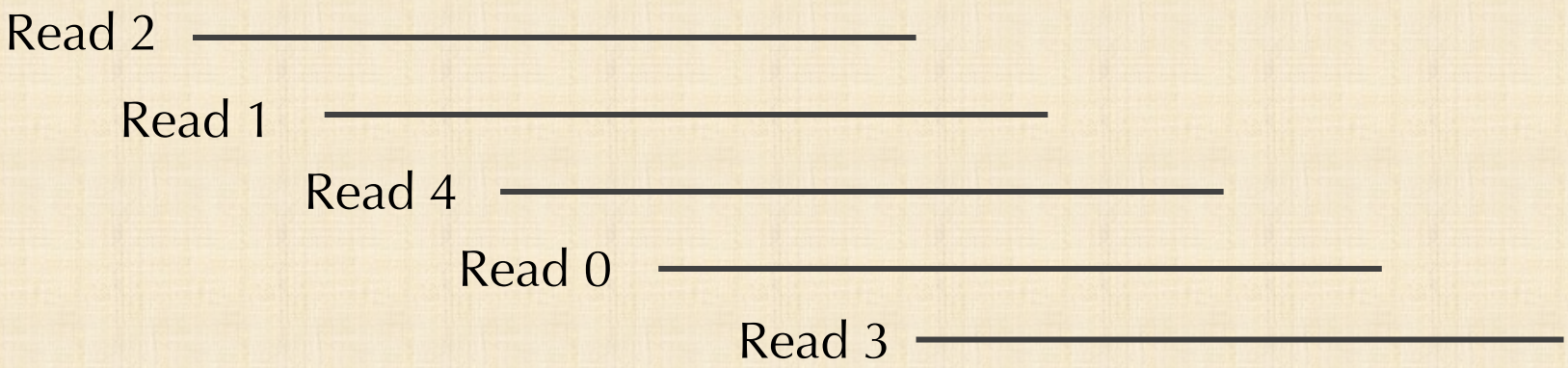
Note: We only see the "correct" path because I have assumed that we know the order of reads when we draw the network.



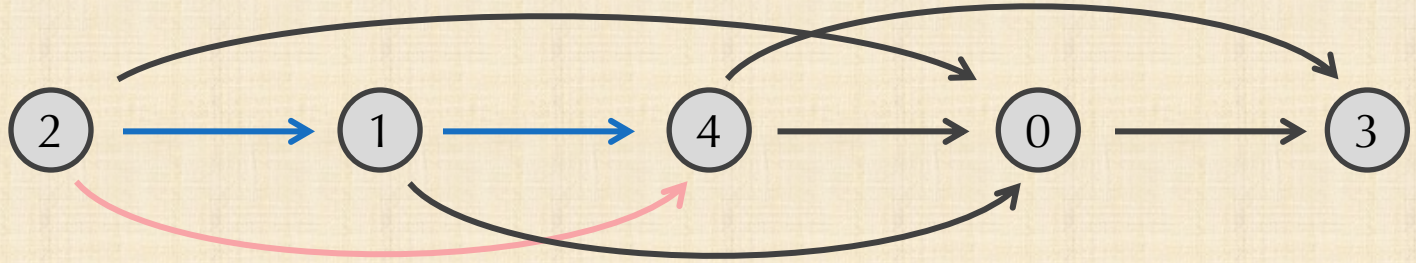
The Reality of the Overlap Network is a Bit Different



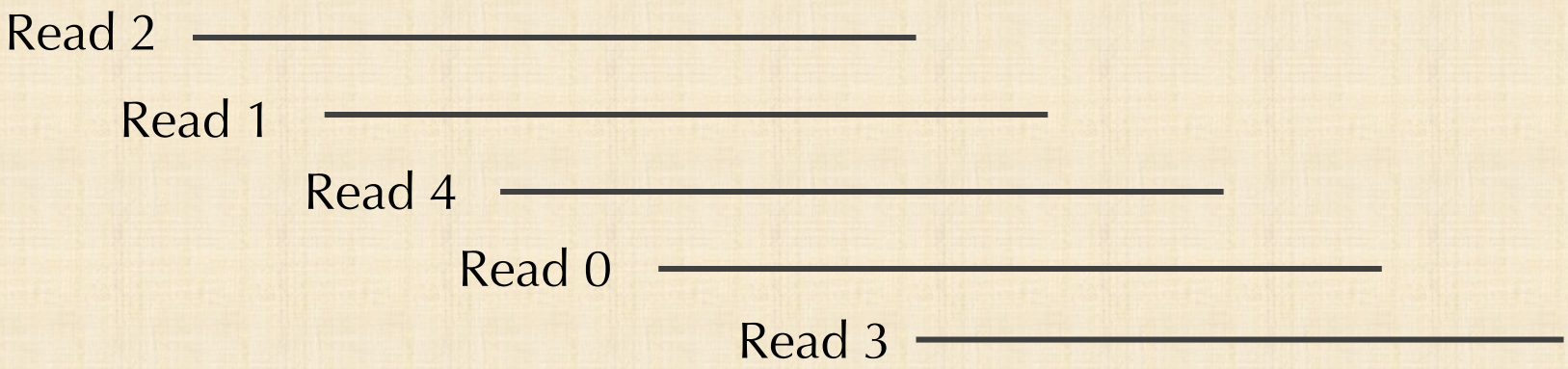
STOP: Any ideas on how we could simplify the network?



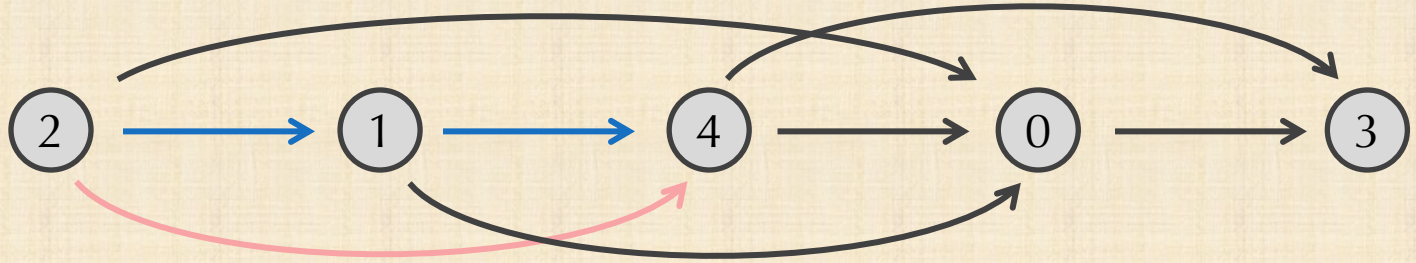
The Reality of the Overlap Network is a Bit Different



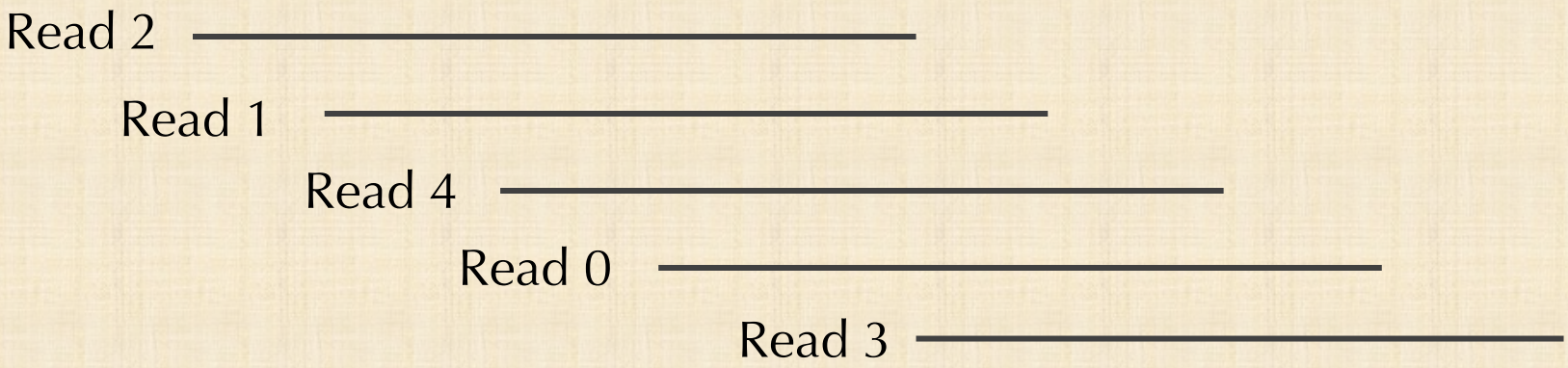
Key Point: If $x \rightarrow y$, $y \rightarrow z$, and $x \rightarrow z$, then the edge $x \rightarrow z$ is probably redundant and can be removed.



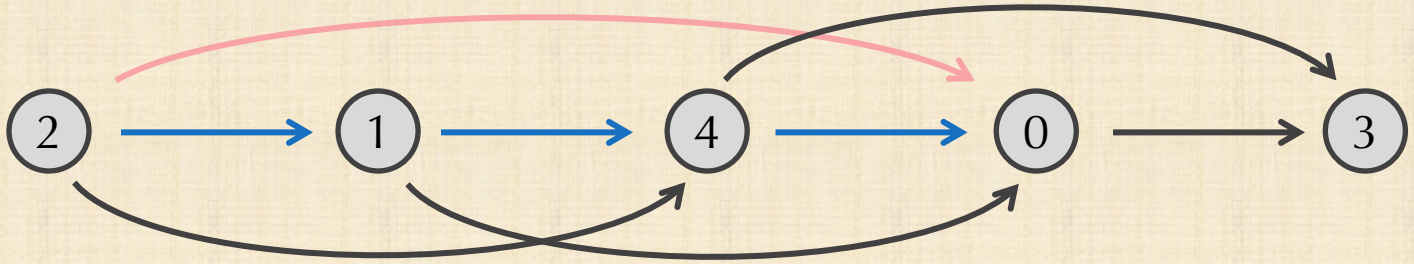
The Reality of the Overlap Network is a Bit Different



STOP: What other edges would be remove according to this rule?



The Reality of the Overlap Network is a Bit Different



Similarly, if $x \rightarrow y$, $y \rightarrow z$, $z \rightarrow w$, and $x \rightarrow w$, then the edge $x \rightarrow w$ is redundant and can be removed.

Read 2

Read 1

Read 4

Read 0

Read 3

Formulating "Edge Trimming" as a Computational Problem

Adjacency List Trimming Problem:

- **Input:** The adjacency list of an (overlap) network, and an integer *maxK*.
- **Output:** The trimmed adjacency list such that if two nodes $x \rightarrow y$ are connected by a path of length between 2 and *maxK*, and $x \rightarrow y$, then we remove the edge $x \rightarrow y$.

Pseudocode for Network Trimming

```
TrimNetwork(adjList, maxK)
        adjlist2 ← blank adjacency list
        for every key pattern in adjList
                adjList2[pattern] ← GetTrimmedNeighbors(pattern,
adjList, maxK)
        return adjList2
```

Here, the function **GetTrimmedNeighbors** finds the updated neighbors of *pattern* in the graph after we perform edge trimming.

Pseudocode for Network Trimming

```
GetTrimmedNeighbors(pattern, adjList, maxK)
        neighbors ← adjList[pattern]
        extendedNeighbors ← ExtendedNbd(pattern, adjList, maxK)
        numNeighbors ← len(neighbors)
        for every integer i between 0 and numNeighbors – 1
                if Contains(extendedNeighbors, neighbors[i])
                        neighbors ← remove neighbors[i]
        return neighbors
```

ExtendedNbd finds all strings that can be reached from *pattern* by paths of length between 2 and *maxK*

Contains returns `true` if the slice contains the value and `false` otherwise.

Building the Extended Neighborhood

Extended Neighborhood Problem:

- **Input:** A string *pattern*, the adjacency list *adjList* of an (overlap) network, and an integer *maxK*.
- **Output:** An array containing every string that can be reached from *pattern* by a path of length between 2 and *maxK*, without duplicates.

Code Challenge: Solve this problem.

Trimming the Network

```
TrimNetwork(adjList, maxK)
        adjlist2 ← blank adjacency list
        for every key pattern in adjList
                adjList2[pattern] ← GetTrimmedNeighbors(pattern,
adjList, maxK)
        return adjList2
```

```
GetTrimmedNeighbors(pattern, adjList, maxK)
        neighbors ← adjList[pattern]
        extendedNeighbors ← ExtendedNbd(pattern, adjList, maxK)
        numNeighbors ← len(neighbors)
        for every integer i between 0 and numNeighbors – 1
                if Contains(extendedNeighbors, neighbors[i])
                        neighbors ← remove neighbors[i]
        return neighbors
```

Code Challenge: Implement these functions.

Returning to the Overlap Network

STOP: Let's re-compute the average outdegree of our adjacency list for our simulated reads *after* trimming. Now what do we see?

From Clean to Messy Reads

To simulate messy reads, we will have three additional parameters corresponding to the probability that a given position will be mutated:

1. A *substitution* rate
2. An *insertion* rate
3. A *deletion* rate

Given a starting position of the genome, we build a read by applying a mutations at each symbol at the given rates until our read is the desired length.

From Clean to Messy Reads

Messy Read Generation Problem:

- **Input:** A string *genome* along with integers *readLength,* a decimal *readProbability,* and three decimals corresponding to the probability of a substitution, insertion, or deletion.
- **Output:** An array of strings resulting from sampling "messy" reads according to our model from the previous slide.

Code Challenge: Let's solve this problem together.

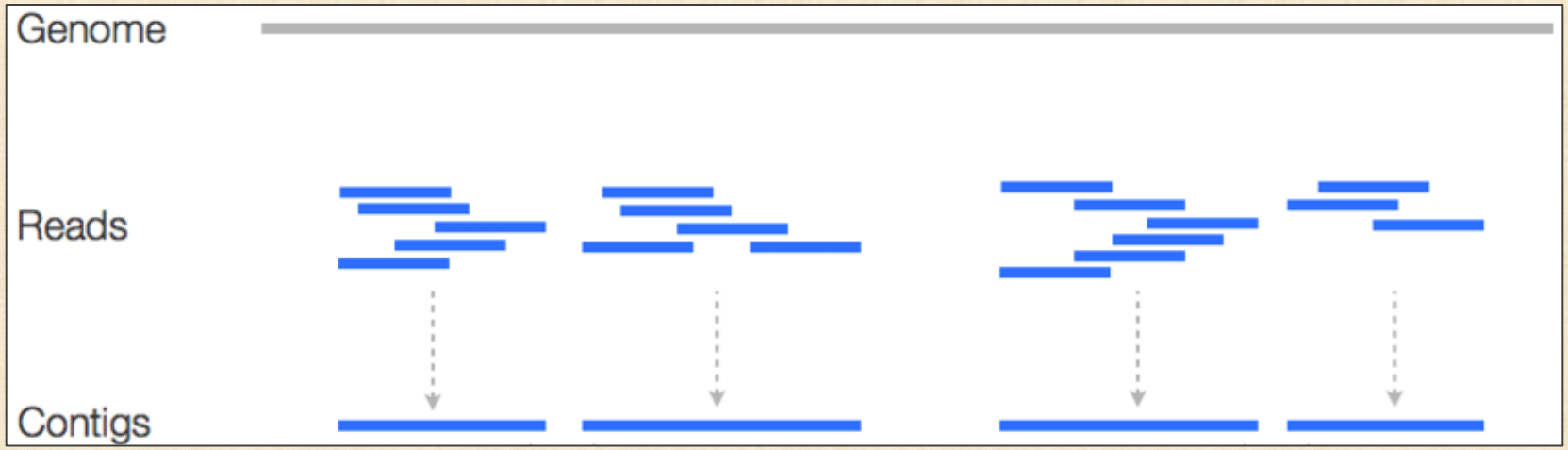
Returning to the Overlap Network

STOP: Now we will add the messy read generation to our pipeline. How does it affect the final average outdegree? What should we do next?

INFERRING CONTIGS FROM OUR NETWORK

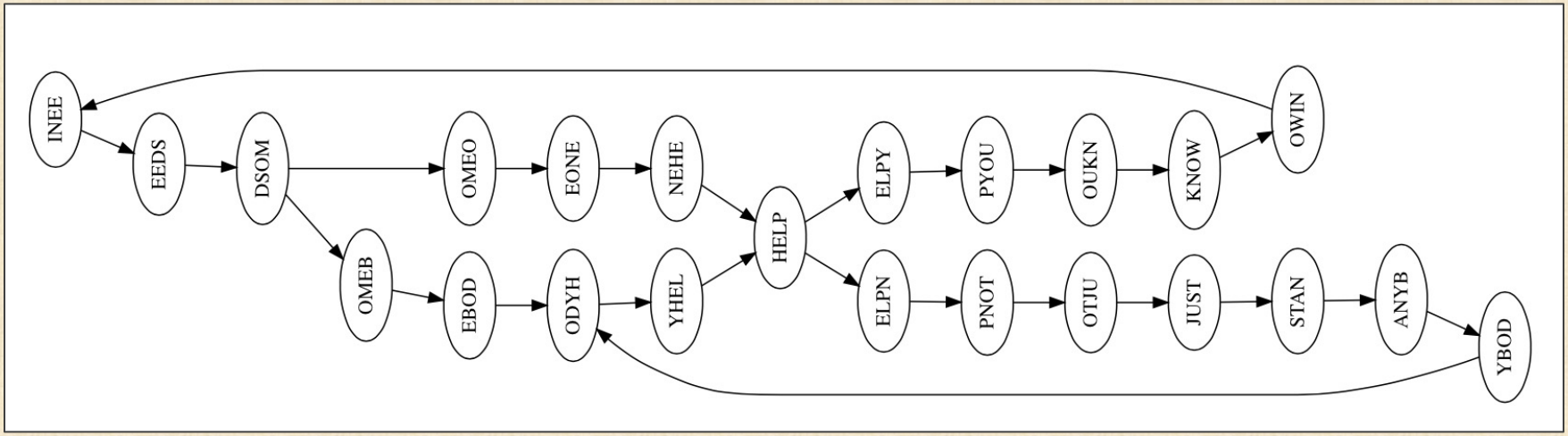
Recall: Real Assemblers Produce "Contigs"

Contig: a contiguous substring of DNA that we are confident about in the underlying chromosome.



A Hypothetical Network after Trimming

Text = `INEEDSOMEBODYHELPNOTJUSTANYBODYHELPYOUKNOWINEEDSOMEONEHELP`

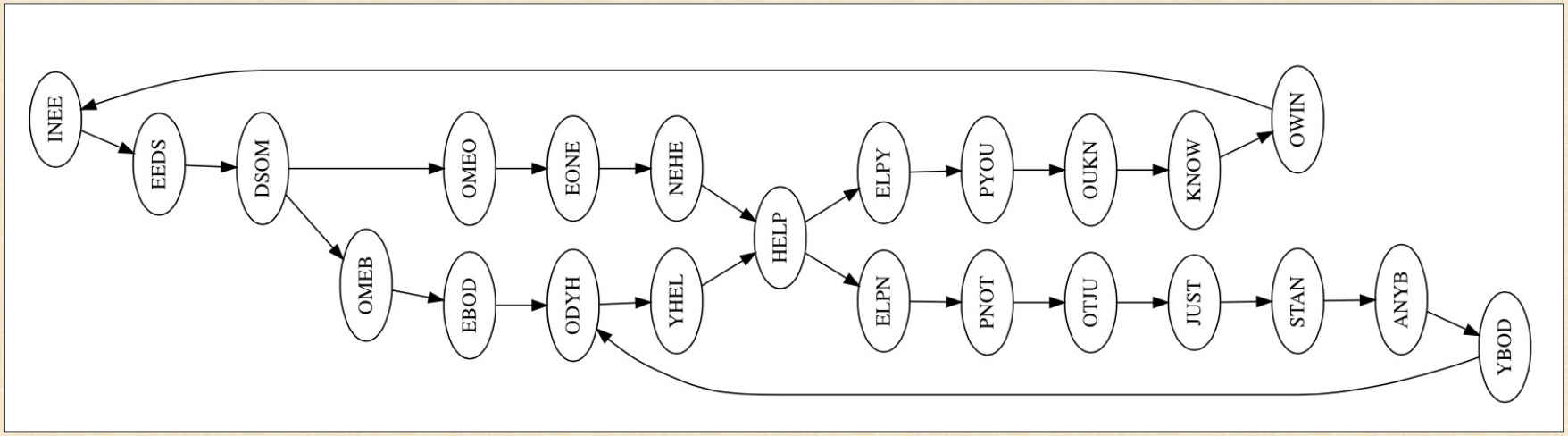


Special thanks: G. Sprowls, Esq.

Note: We could never infer that this is the "correct" text from this network.

A Hypothetical Network after Trimming

Text = `INEEDSOMEBODYHELPNOTJUSTANYBODYHELPYOUKNOWINEEDSOMEONEHELP`

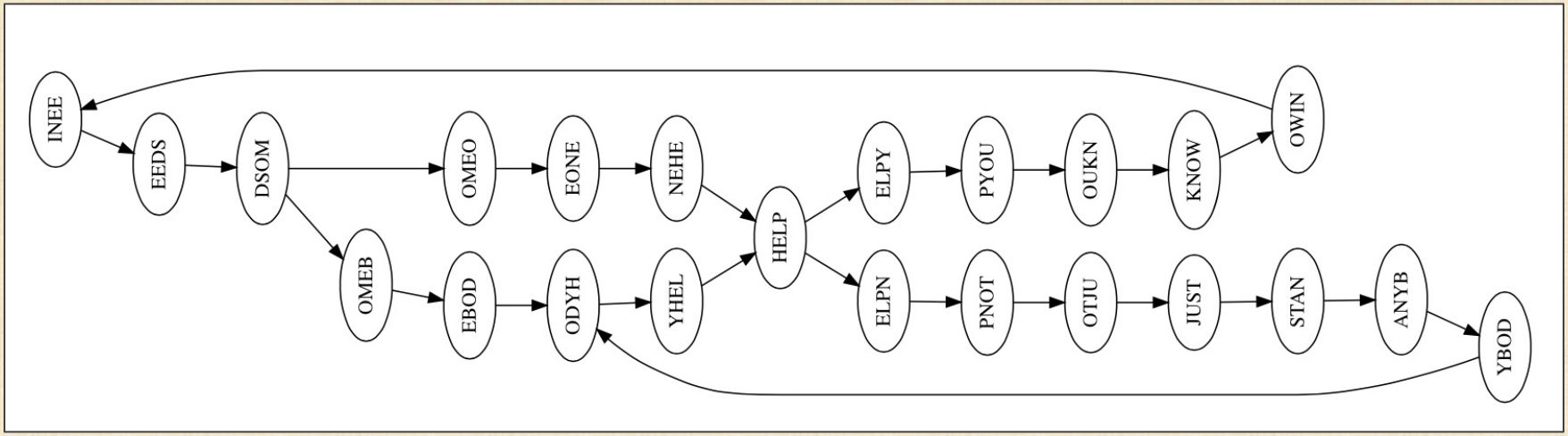


Special thanks: G. Sprowls, Esq.

STOP: What parts of the graph can we safely infer as "correct"?

A Hypothetical Network after Trimming

Text = `INEEDSOMEBODYHELPNOTJUSTANYBODYHELPYOUKNOWINEEDSOMEONEHELP`

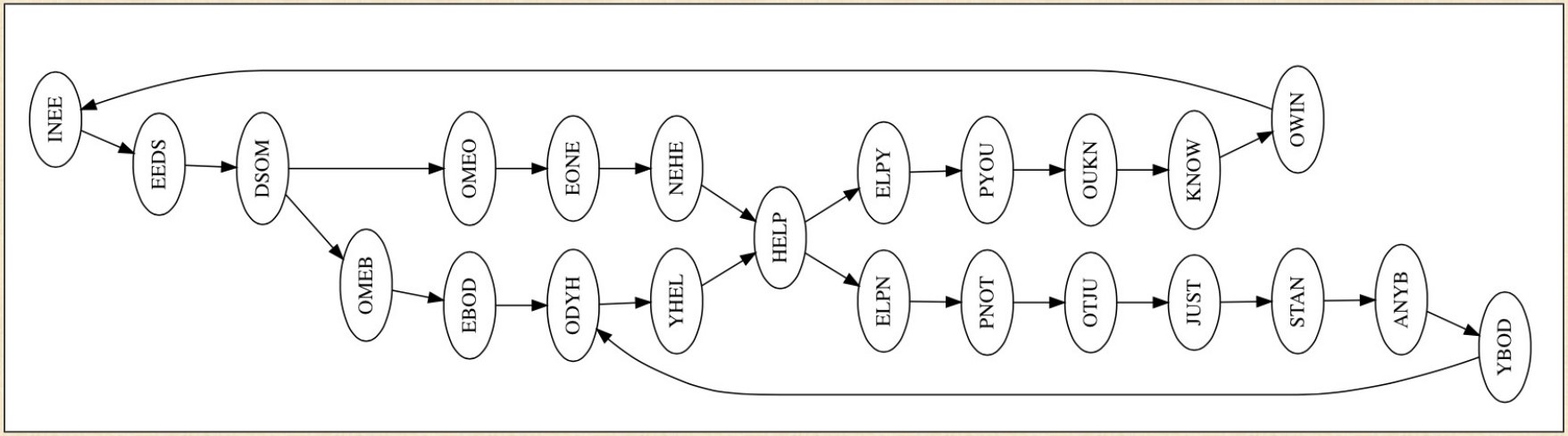


Special thanks: G. Sprowls, Esq.

We are looking for **maximal non-branching paths**: paths in the network for which every node is "1 in, 1 out". These are our contigs!

A Hypothetical Network after Trimming

Text = INEEDSOMEBODYHELPNOTJUSTANYBODYHELPYOUKNOWINEEDSOMEONEHELP

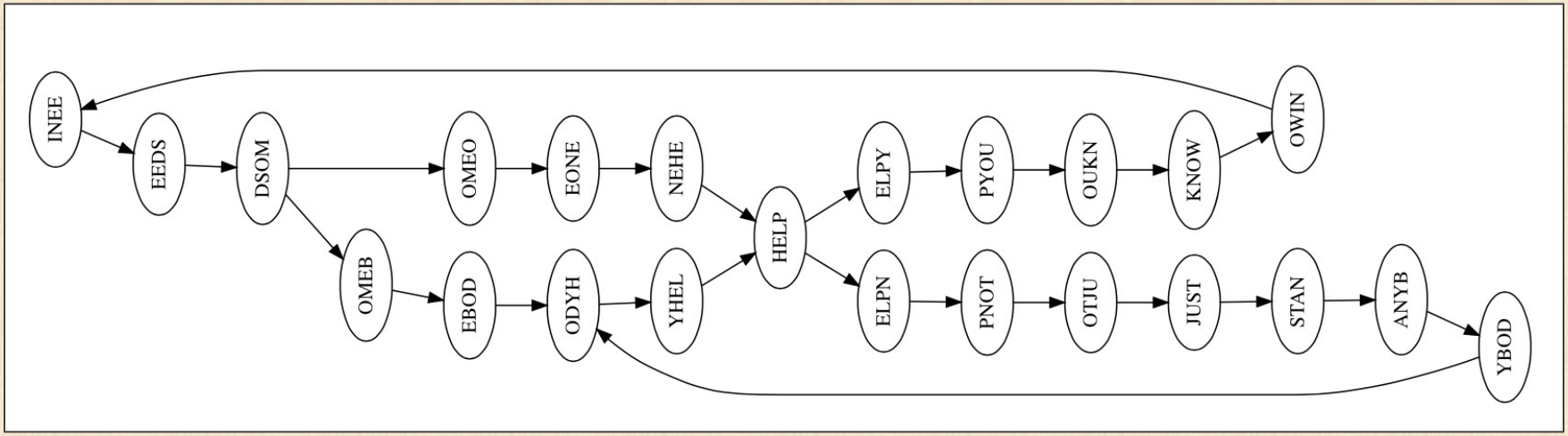


Special thanks: G. Sprowls, Esq.

Exercise: Find the maximal non-branching paths in this graph, and infer contigs from them.

A Hypothetical Network after Trimming

Text = INEEDSOMEBODYHELPNOTJUSTANYBODYHELPYOUKNOWINEEDSOMEONEHELP



Special thanks: G. Sprowls, Esq.

Answer: HELPYOUKNOWINEEDSOM, DSOMEONEHELP, DSOMEBODYH, ODYHELP, HELPNOTJUSTANYBODYH.

Contigs = Non-Branching Paths

Maximal Non-Branching Paths Problem:

- **Input:** The adjacency list *adjList* of an (overlap) network.
- **Output:** An array whose values are arrays of strings, each corresponding to an order of strings in a maximal non-branching path of the overlap network.

Note: This problem is quite challenging, so we will provide a function **GenerateContigs** for you ☺.

Determining Contigs from Simulated Data

STOP: Now we will infer contigs from our pipeline. How many do we have and what should we do now?

Congrats! We have mostly replicated an assembly algorithm from 2016

Minimap and **miniasm**: fast mapping and de novo assembly for noisy long sequences

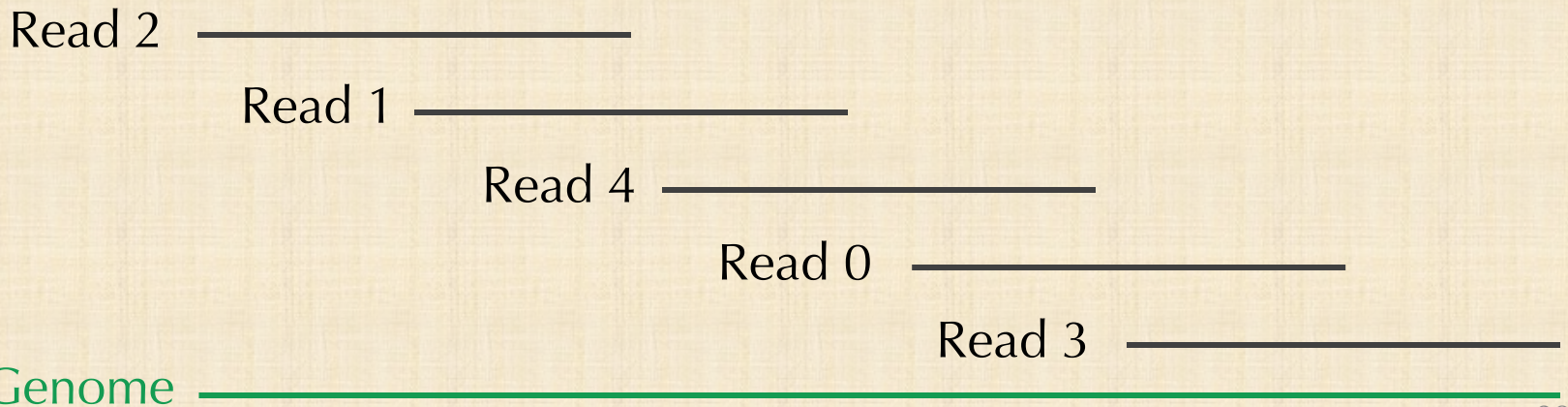
H Li - Bioinformatics, 2016 - academic.oup.com

… Results: We present a new mapper, minimap and a de novo assembler, **miniasm**, for efficiently mapping and assembling SMRT and ONT reads without an error correction stage. They …

☆ Save Cite Cited by 1001 Related articles All 11 versions

There Is Just One Problem

Key Point: It's not clear how to align (possibly error-prone) reads and then infer the genome. We need to know more about multiple alignment!



TO BE CONTINUED ...