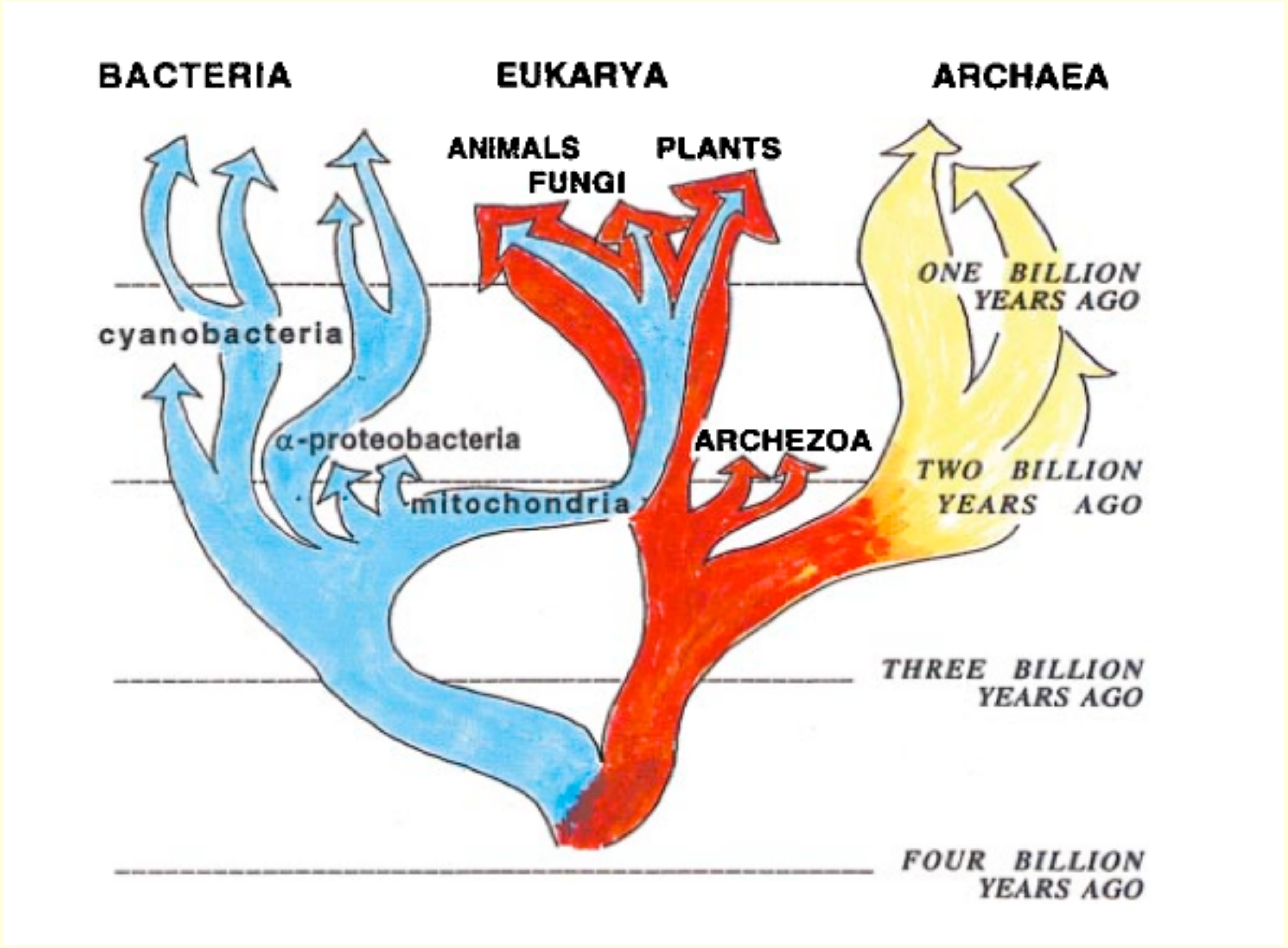
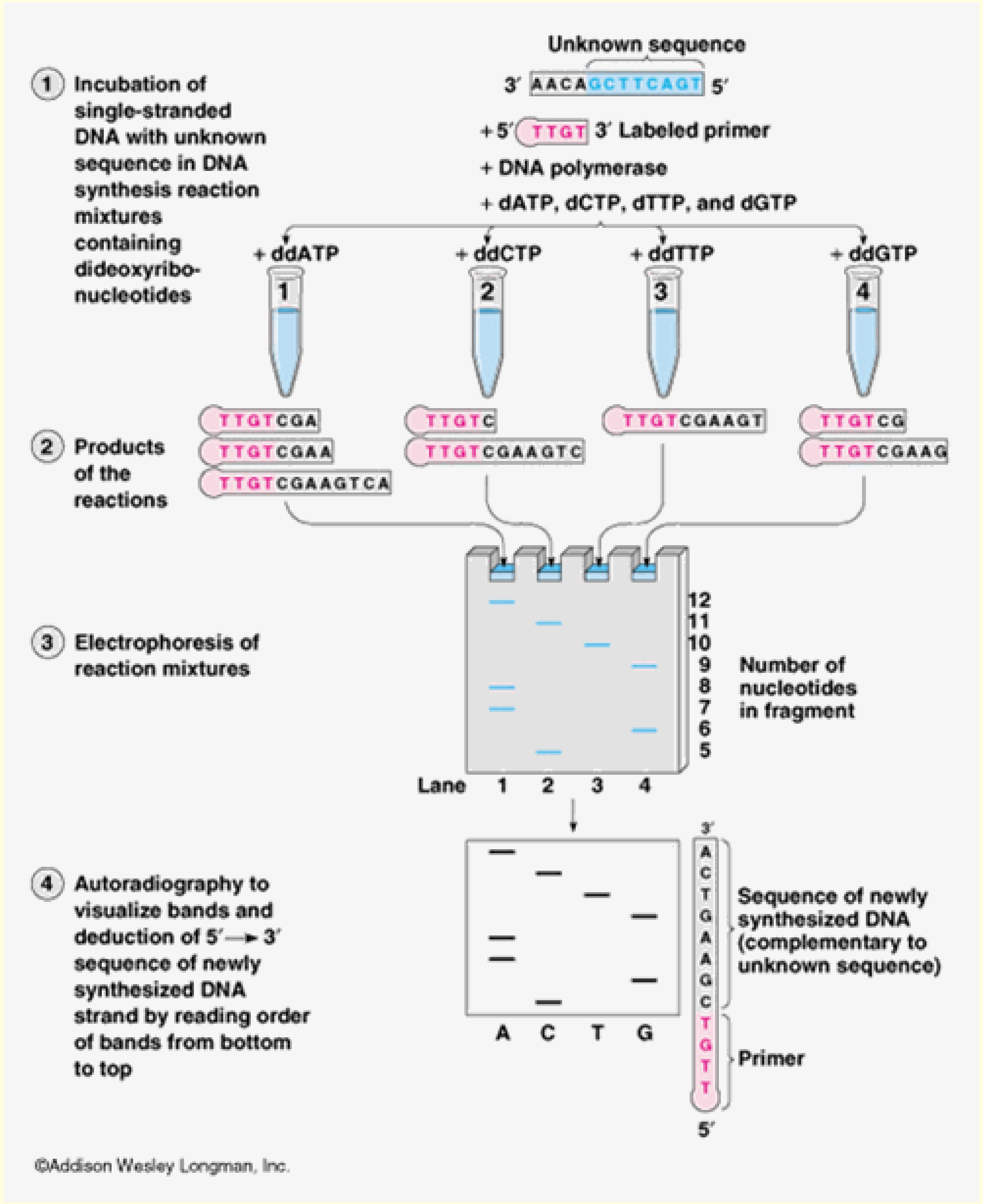
# CMPS 6630: Introduction to Computational Biology and Bioinformatics

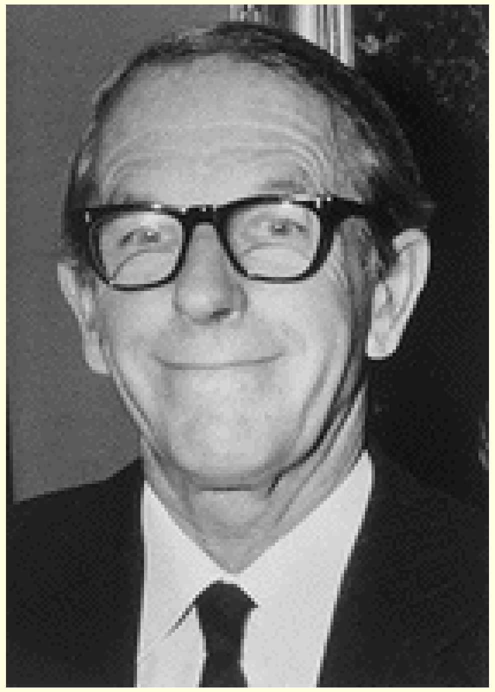
## Sequence Assembly

# Why Genome Sequencing?





Unknown sequence
3′ AACAGCTTCAGT 5′
+5′ TTGT 3′ Labeled primer
+ DNA polymerase
+ dATP, dCTP, dTTP, and dGTP
1 Incubation of single-stranded DNA with unknown sequence in DNA synthesis reaction mixtures containing dideoxyribonucleotides
+ ddATP
+ ddCTP
+ ddTTP
+ ddGTP
1
2
3
4
2 Products of the reactions
TTGTCGA
TTGTCGAA
TTGTCGAAGTCA
TTGTC
TTGTCGAAGTC
TTGTCGAAGT
TTGTCG
TTGTCGAAG
3 Electrophoresis of reaction mixtures
12 11 10 9 8 7 6 5
Number of nucleotides in fragment
Lane 1 2 3 4
4 Autoradiography to visualize bands and deduction of 5′→3′ sequence of newly synthesized DNA strand by reading order of bands from bottom to top
A C T G
3′ A C T G A A G C T G T T 5′
Sequence of newly synthesized DNA (complementary to unknown sequence)
Primer
©Addison Wesley Longman, Inc.



Sanger (1982) introduced chain-termination sequencing.

Main idea: Obtain fragments of all possible lengths, ending in A, C, T, G.

Using gel electrophoresis, we can separate fragments of differing lengths, and then assemble them.

## Automated Sequencing

**Speeding the Gene Hunt: High-Speed DNA Sequencing**

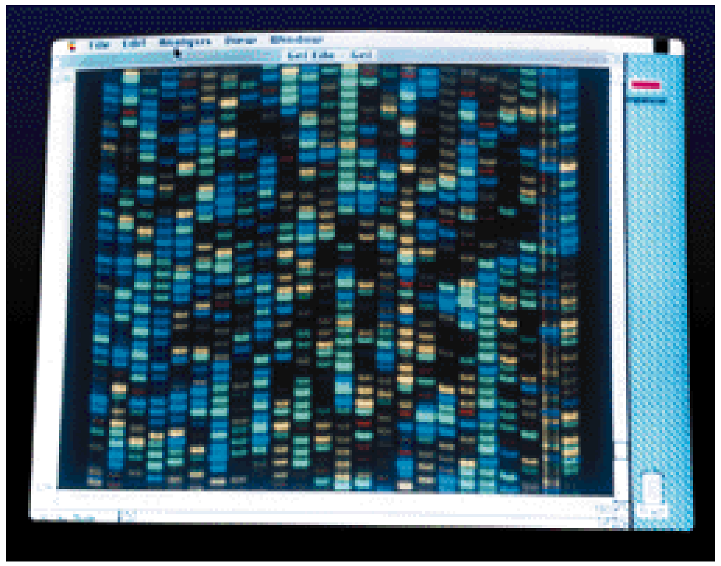
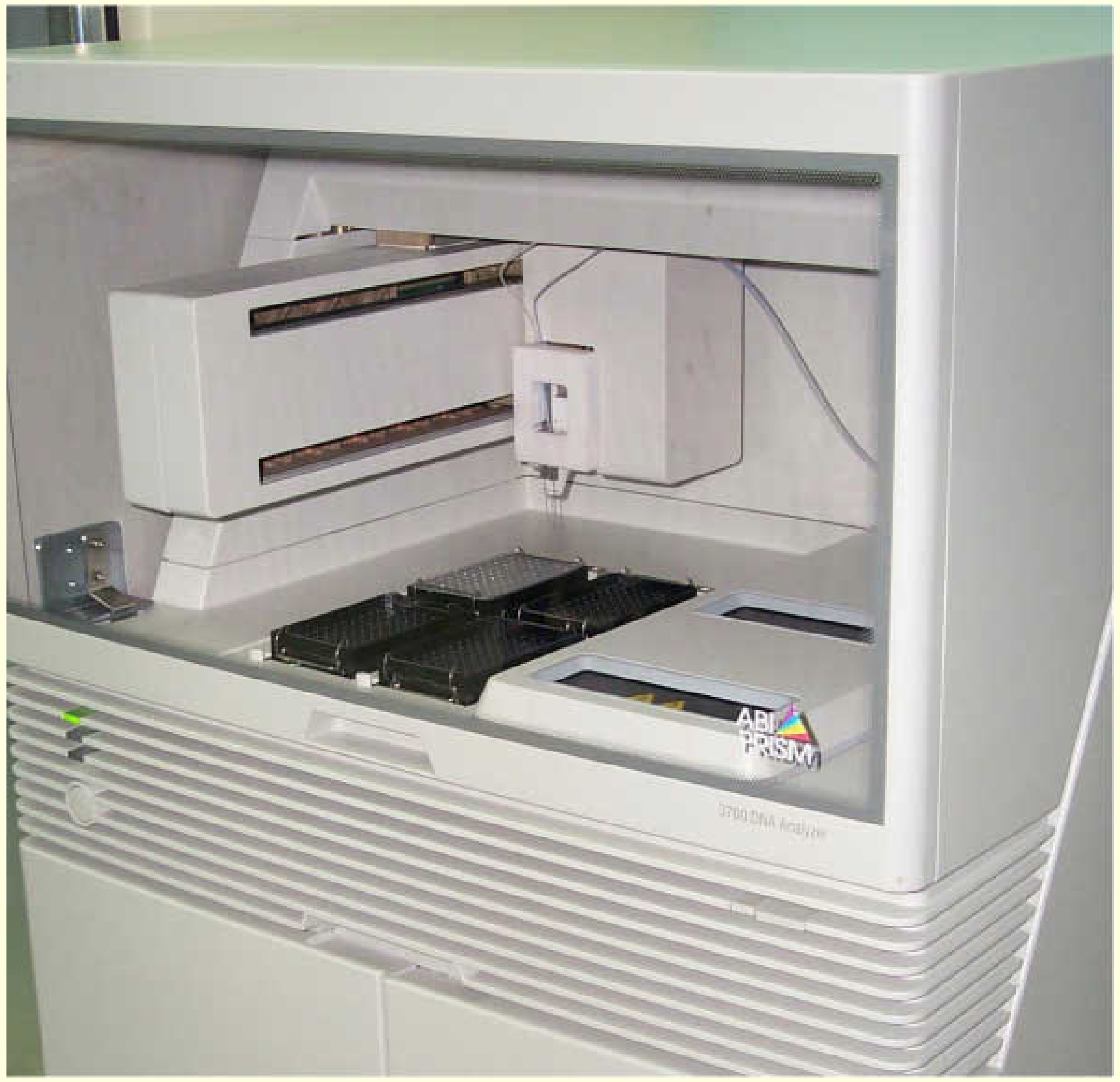


Figure 1. Computer-generated image of fluorescent bands after the fragments are detected by the laser.



Perkin-Elmer 3700:
Can sequence ~500bp with 98.5% accuracy

## DNA Fragmentation

- DNA is first purified and then fragmented.
- Fragments must then be sorted and cloned before they are sequenced.
- Suppose we are able to separate and sequence individual ~500bp fragments, or reads (ignore directionality for now).

## In a Perfect World

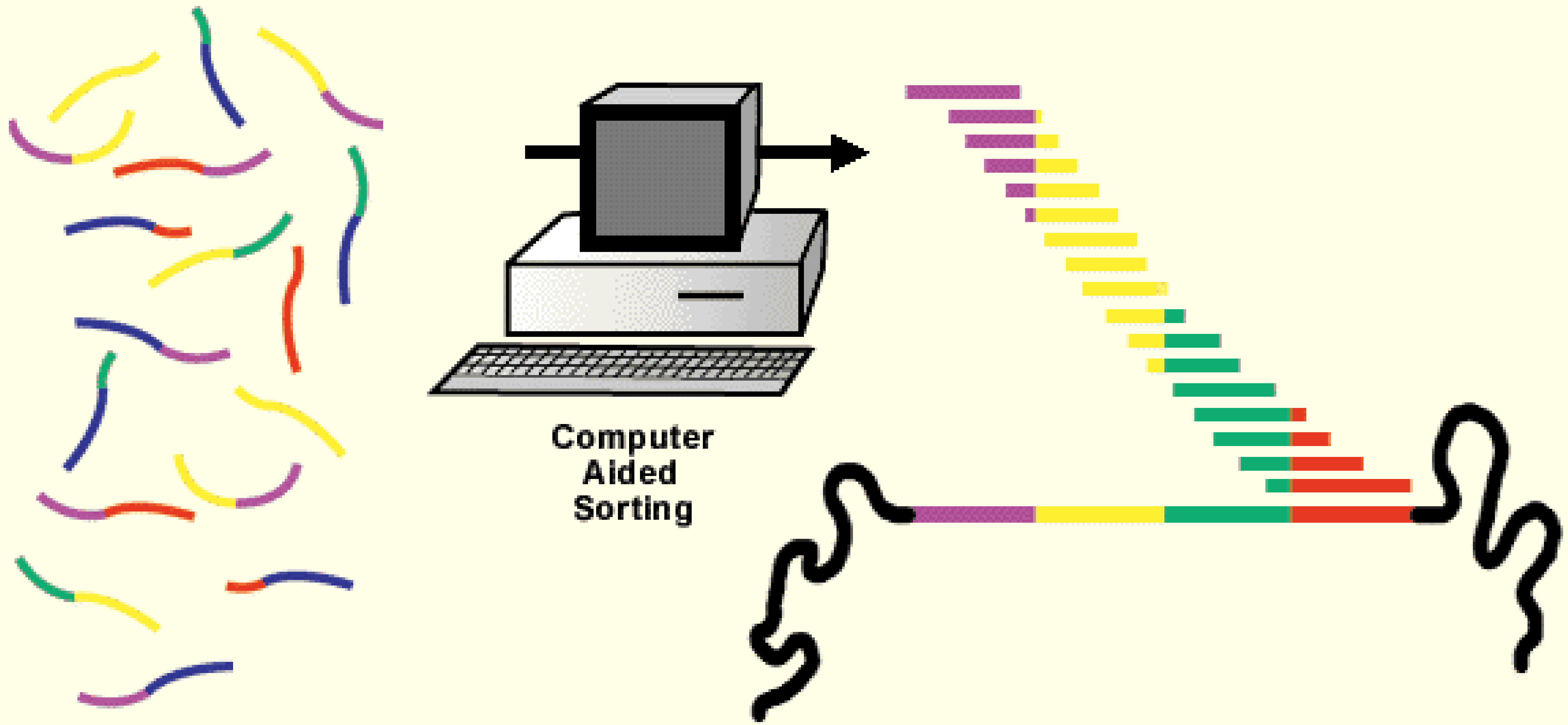


Fig 2: Short fragments of DNA sequence are ordered by overlapping data to recreate the whole genome sequence

## Shortest Common Superstring

Given fragments $s_1, s_2, \ldots s_n$, find a string $S$ such that for all $i$, $s_i \in S$ and $length(S)$ is minimized.

| | | | | | |
|---|---|---|---|---|---|
| `ACGGTG` | `GGTGA` | `GAC` | `AATCC` | `TCCG` | `GAAT` |
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Clearly we must “assemble” $S$ from the given fragments, but how?

# Shortest Common Superstring

Given fragments $s_1, s_2, \ldots s_n$, find a string $S$ such that for all $i$, $s_i \in S$ and $length(S)$ is minimized.

| | | | | | |
|---|---|---|---|---|---|
| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

```
     ACGGTG
       GGTGA
    GAC    AATCC
             TCCG
          GAAT
    -------------
S : GACGGTGAATCCG
```

What makes this the shortest common superstring?

# Shortest Common Superstring

Given fragments $s_1, s_2, \ldots s_n$, find a string $S$ such that for all $i$, $s_i \in S$ and $length(S)$ is minimized.

| | | | | | |
|---|---|---|---|---|---|
| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

```
     ACGGTG
       GGTGA
    GAC    AATCC
             TCCG
          GAAT
    _____________
S : GACGGTGAATCCG
```

What makes this the shortest common superstring?

Maximize Overlap!

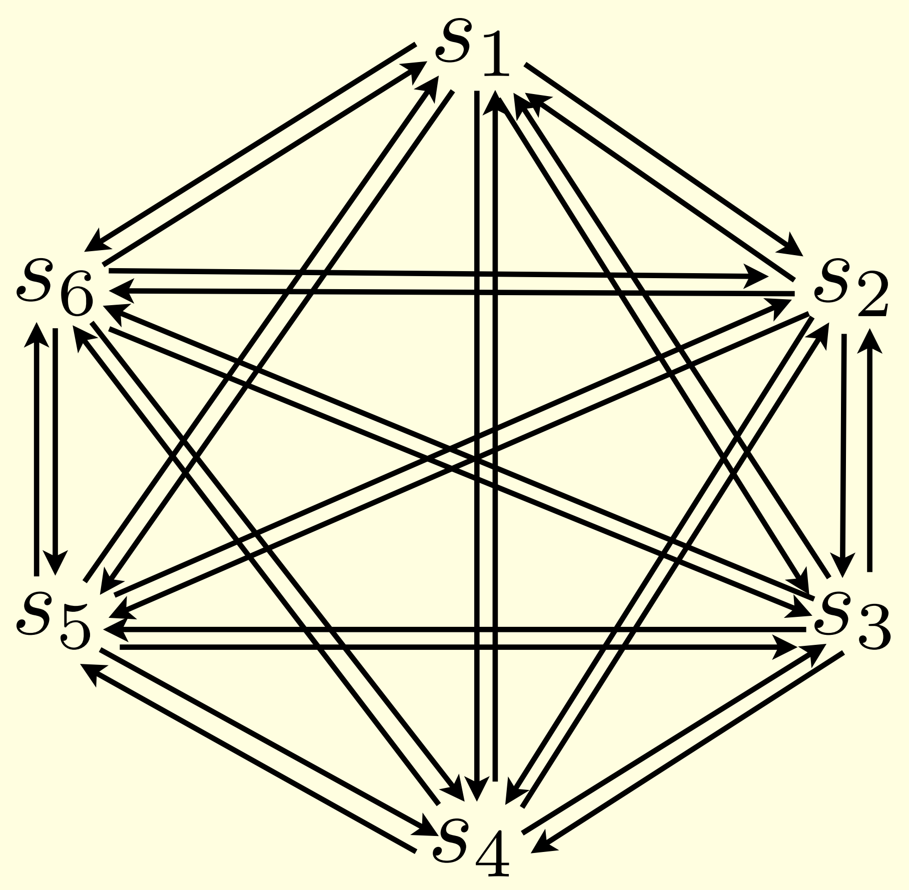
# Shortest Common Superstring

Given fragments $s_1, s_2, \ldots s_n$, find a string $S$ such that for all $i$, $s_i \in S$ and $length(S)$ is minimized.

| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

|  | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | ▨ | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | ▨ | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | ▨ | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | ▨ | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | ▨ | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | ▨ |



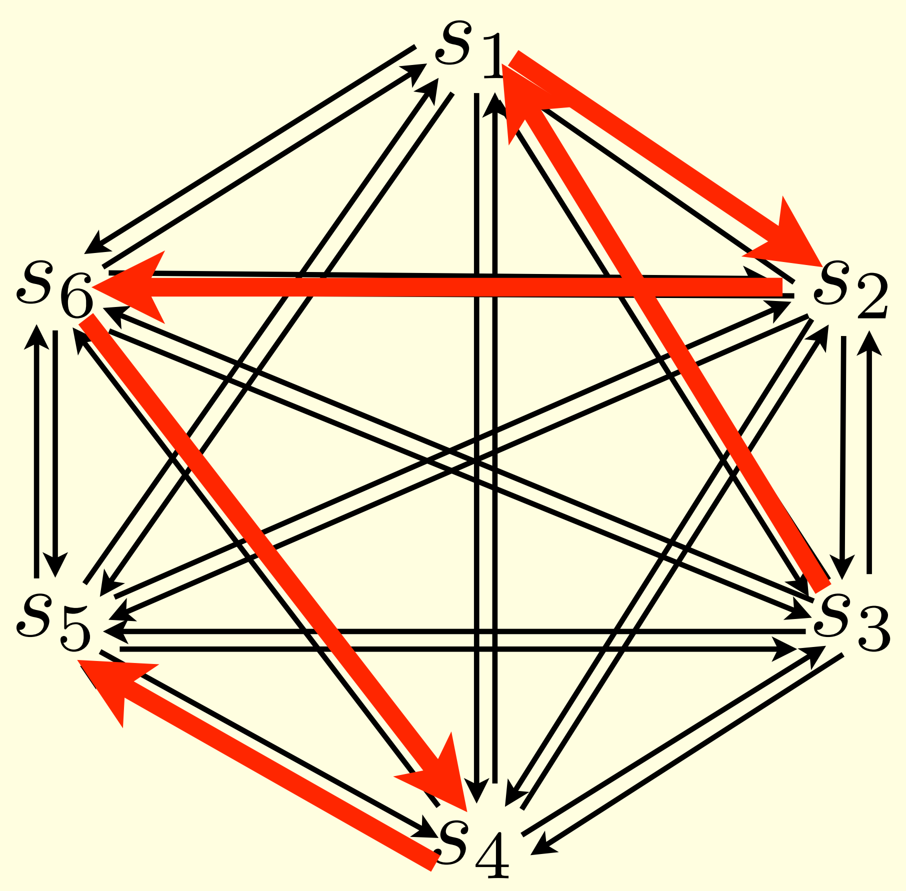
# Shortest Common Superstring

Given fragments $s_1, s_2, \ldots s_n$, find a string $S$ such that for all $i$, $s_i \in S$ and $length(S)$ is minimized.

| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | ▨ | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | ▨ | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | ▨ | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | ▨ | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | ▨ | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | ▨ |



This formulation is the Traveling Salesman Problem.

# Algorithms/Heuristics

TSP and Shortest Common Superstring are both NP-Complete; it is unlikely that there is a polynomial-time algorithm.

| ACGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | ▨ | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | ▨ | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | ▨ | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | ▨ | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | ▨ | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | ▨ |

The greedy heuristic is commonly used to find a superstring.

# Greedy Heuristic

| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | ▨ | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | ▨ | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | ▨ | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | ▨ | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | ▨ | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | ▨ |

ACGGTG

# Greedy Heuristic

| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

|  | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | ▨ | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | ▨ | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | ▨ | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | ▨ | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | ▨ | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | ▨ |

```
ACGGTG
  GGTGA
```

# Greedy Heuristic

| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

|  | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | ▨ | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | ▨ | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | ▨ | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | ▨ | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | ▨ | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | ▨ |

```
ACGGTG
  GGTGA
     GAC
```

# Greedy Heuristic

| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | ▨ | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | ▨ | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | ▨ | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | ▨ | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | ▨ | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | ▨ |

```
ACGGTG
  GGTGA
     GAC
        AATCC
```

# Greedy Heuristic

| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | ▨ | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | ▨ | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | ▨ | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | ▨ | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | ▨ | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | ▨ |

```
ACGGTG
  GGTGA
     GAC
        AATCC
          TCCG
```

# Greedy Heuristic

| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | |

```
ACGGTG
  GGTGA
     GAC
        AATCC
          TCCG
             GAAT
-----------------
ACGGTGACAATCCGAAT
```

The greedy heuristic is conjectured to be 4-approximate; there are guarantees when overlap is “metric”.

# Greedy Heuristic

| ACGGTG | GGTGA | GAC | AATCC | TCCG | GAAT |
|---|---|---|---|---|---|
| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |

Overlap

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ |
|---|---|---|---|---|---|---|
| $s_1$ | ▨ | -4 | -1 | 0 | 0 | 0 |
| $s_2$ | -1 | ▨ | -2 | 0 | 0 | -2 |
| $s_3$ | -2 | 0 | ▨ | 0 | 0 | 0 |
| $s_4$ | 0 | 0 | 0 | ▨ | -3 | 0 |
| $s_5$ | 0 | -1 | -1 | 0 | ▨ | 0 |
| $s_6$ | 0 | 0 | 0 | -3 | -1 | ▨ |

```
ACGGTG
  GGTGA
     GAC
        AATCC
          TCCG
             GAAT
-----------------
ACGGTGACAATCCGAAT
  GACGGTGAATCCG
```

The greedy heuristic is conjectured to be 4-approximate; there are guarantees when overlap is “metric”.

# Current Sequencing Software

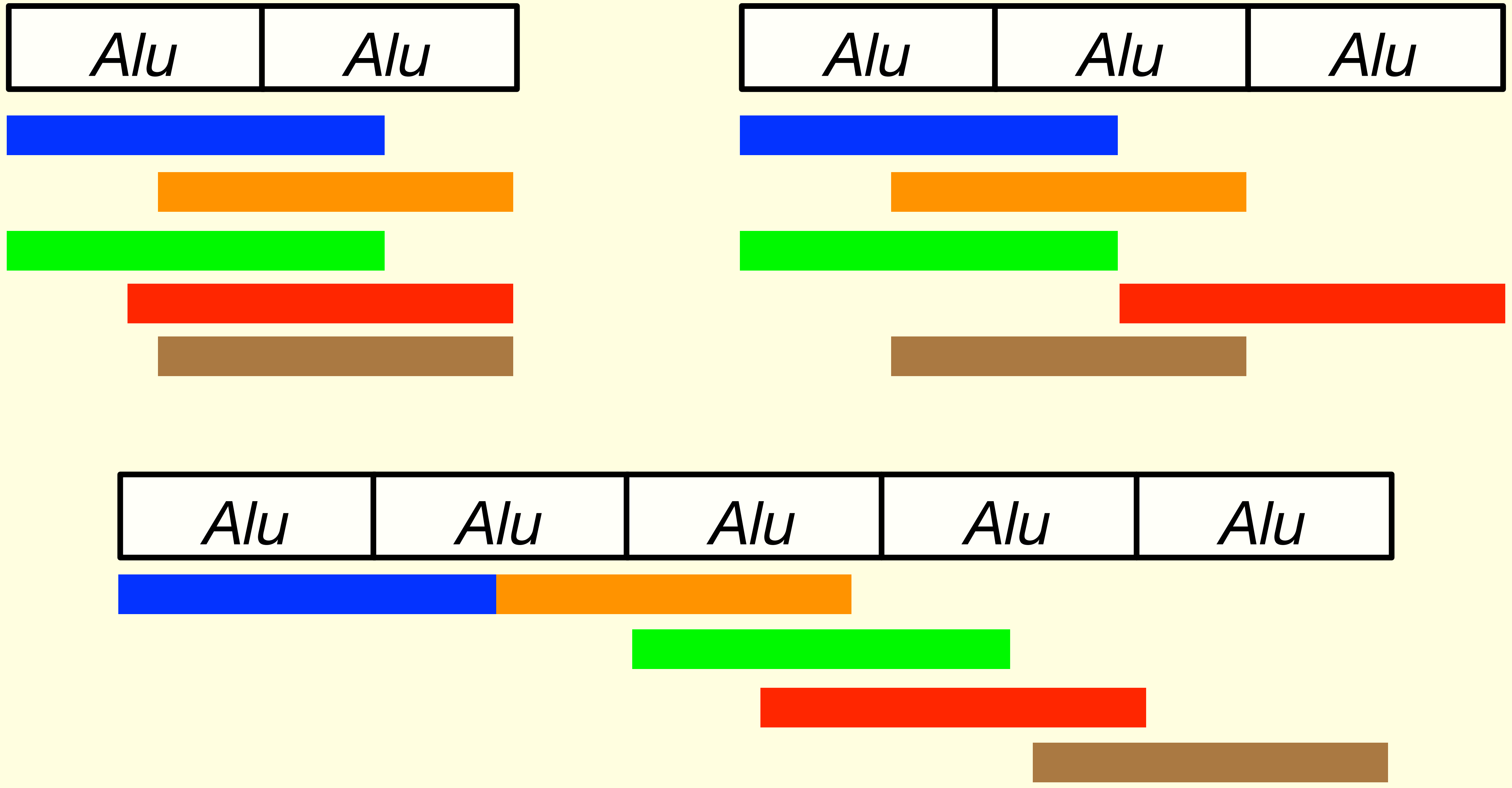
Sequencing software systems (Phrap, Arachne, Celera, etc.) are composed of several parts:

- Basecalling: Create fragments by labeling positions with statisically most likely nucleotides.
- Assembly: Combine fragments using overlaps to create contigs.
- Finishing: Find and close gaps between contigs

# Sequencing in the Real World

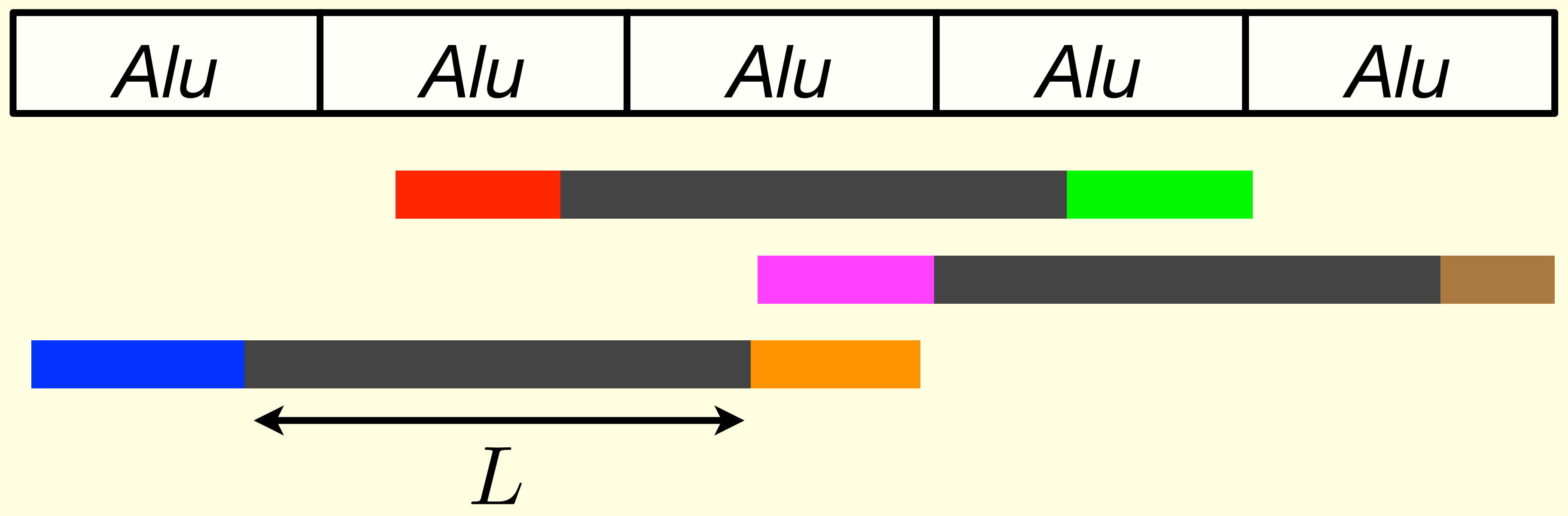
- Sequencers have a 1-3% error rate, so overlaps are imperfect.
- Genomes are evolutionary products:
  - ALU repeats (~300bp, 10%)
  - Transposons/Retrotransposons (50%)
  - Causes: recombination, viruses, etc.
- If a repeat is longer than a read, can we detect it?

# Repeat Detection



There is not always a unique assembly in the presence of repeats.

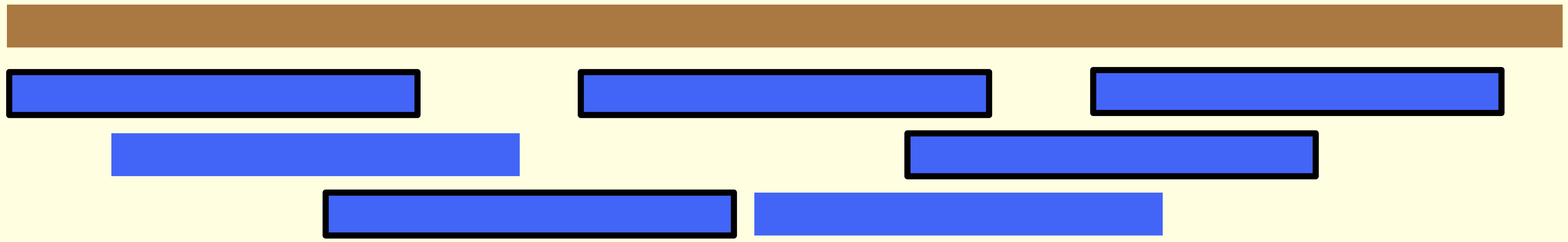
# Mate Pairs



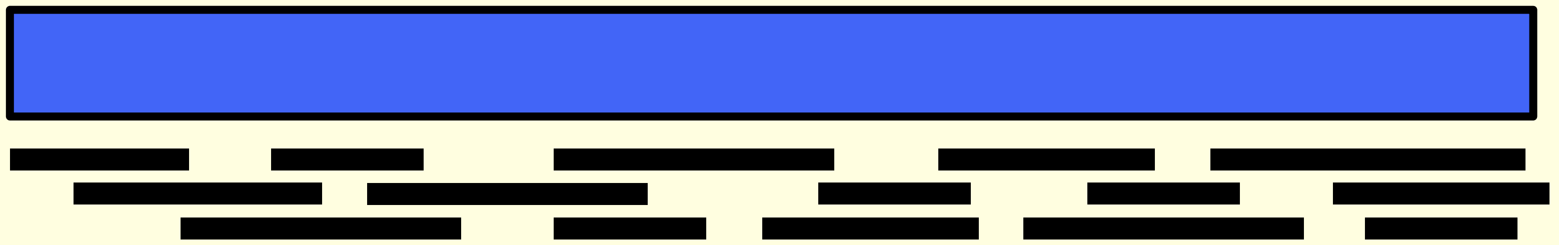
The *mate-pair* method increases the virtual size of a read by forcing reads to be approximately $L$ base pairs apart.

This in turn gives us reads for which only the ends are known; we can recover the rest of the read by guaranteeing extra coverage.

# BAC-by-BAC

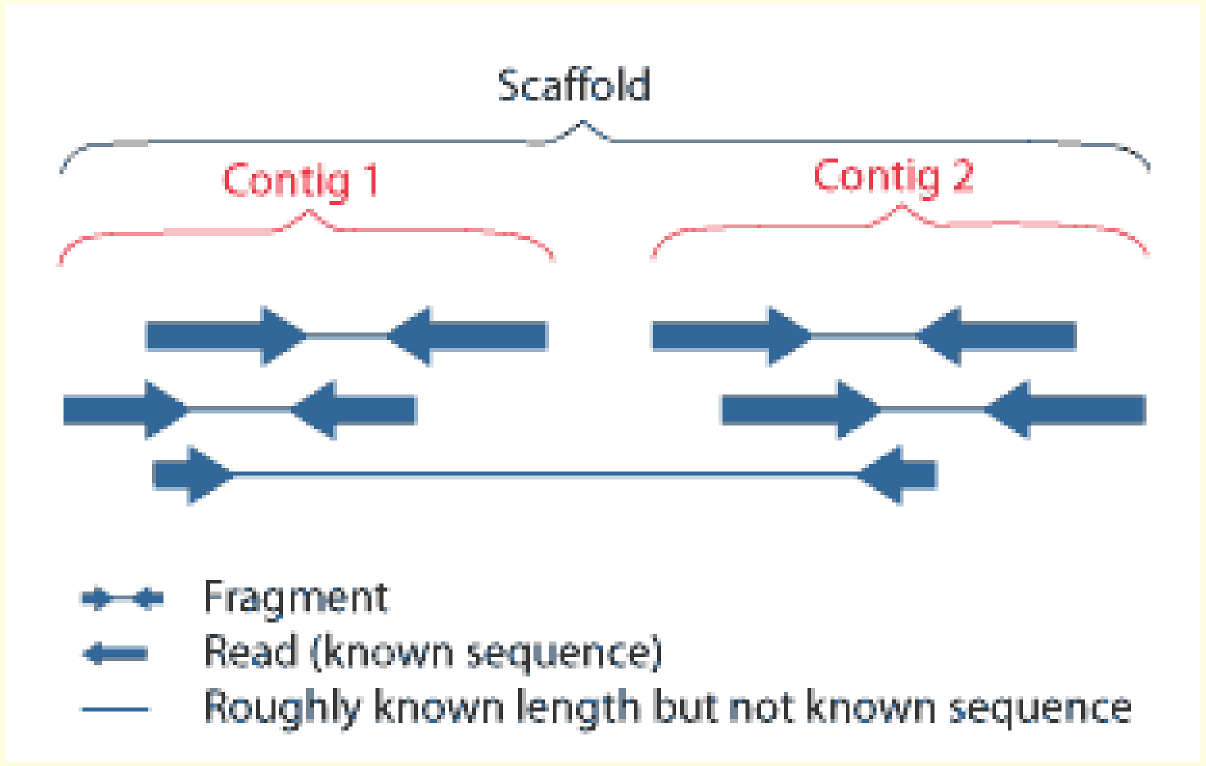


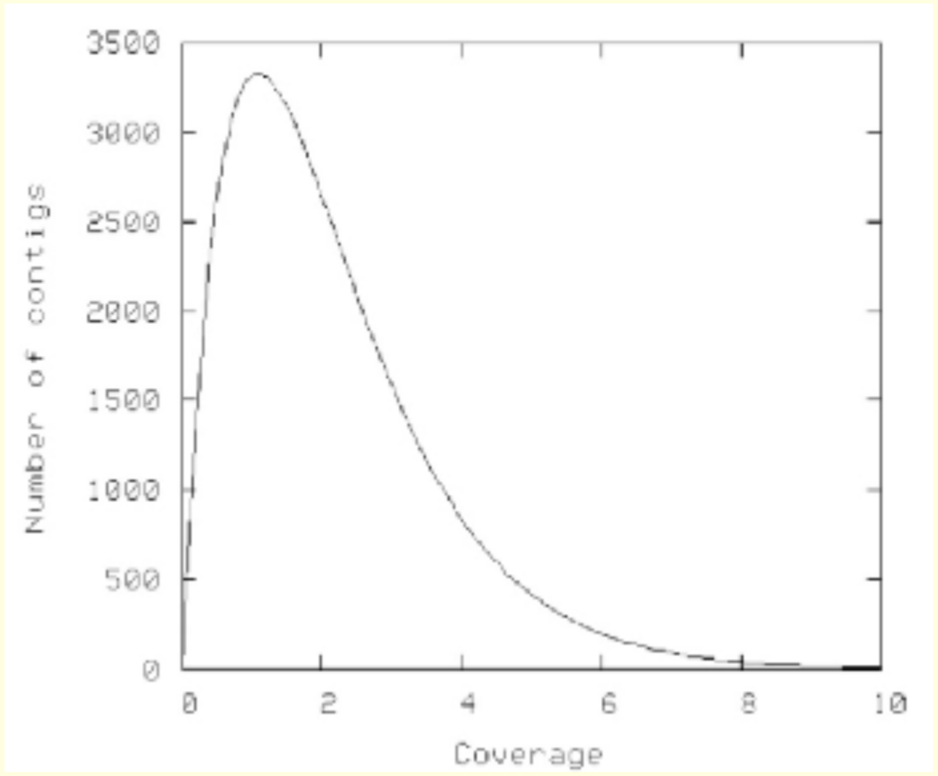
A *minimum tiling path* uses landmarks in the given genome. Each “tile” (~10Kbp) is then treated as a mini-genome (a Bacterial Artificial Chromosome).



Each BAC is then sequenced using the shotgun method (using Phrap). Note that repeats are still an issue, but to a lesser extent.

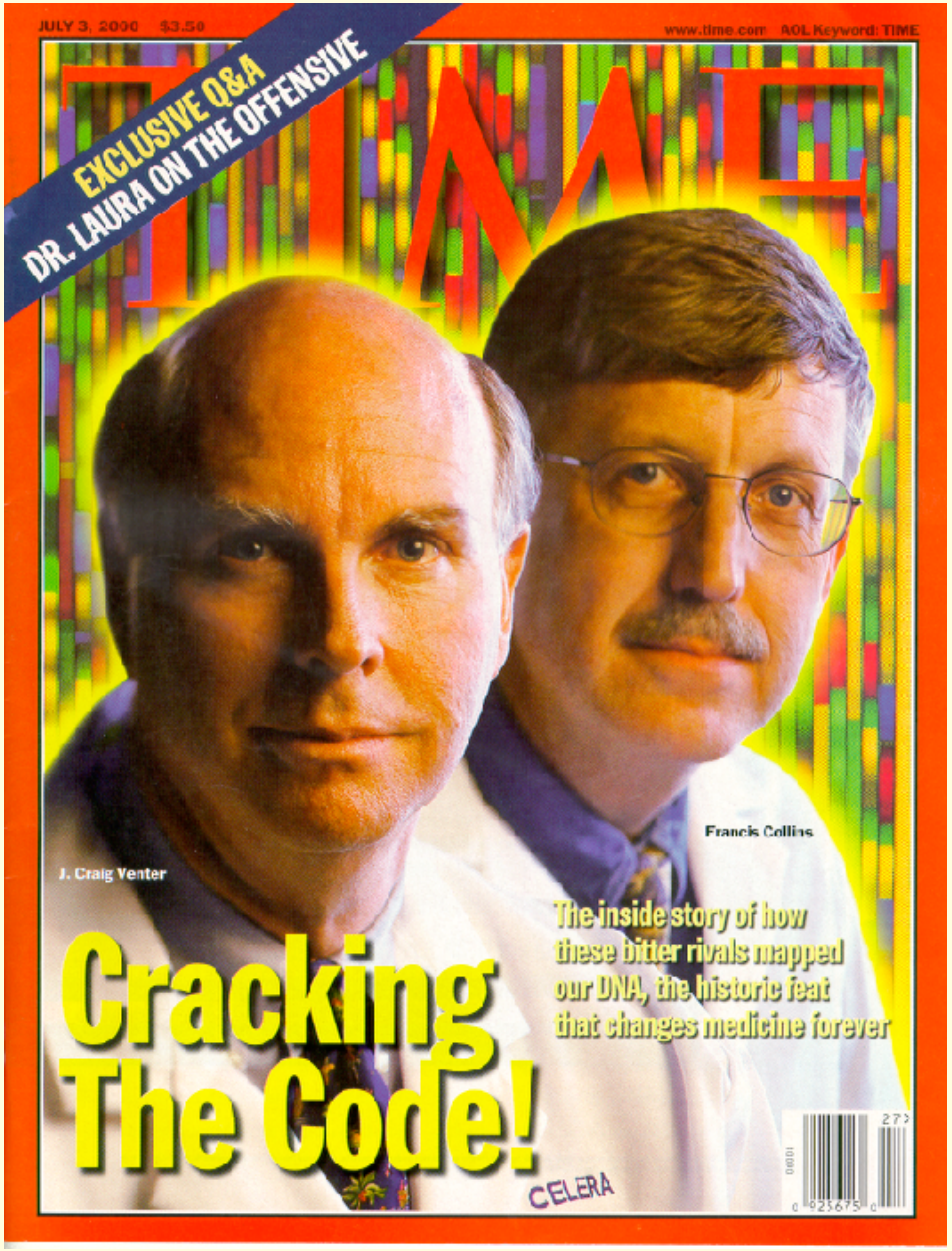
# Shotgun Sequencing





DNA is fragmented into ~2000bp segments, and the ends are sequenced. With enough coverage, the number of contigs can be shown to be low enough for assembly.

Celera proved (with proprietary methods) that the entire genome could be sequenced without BACs.

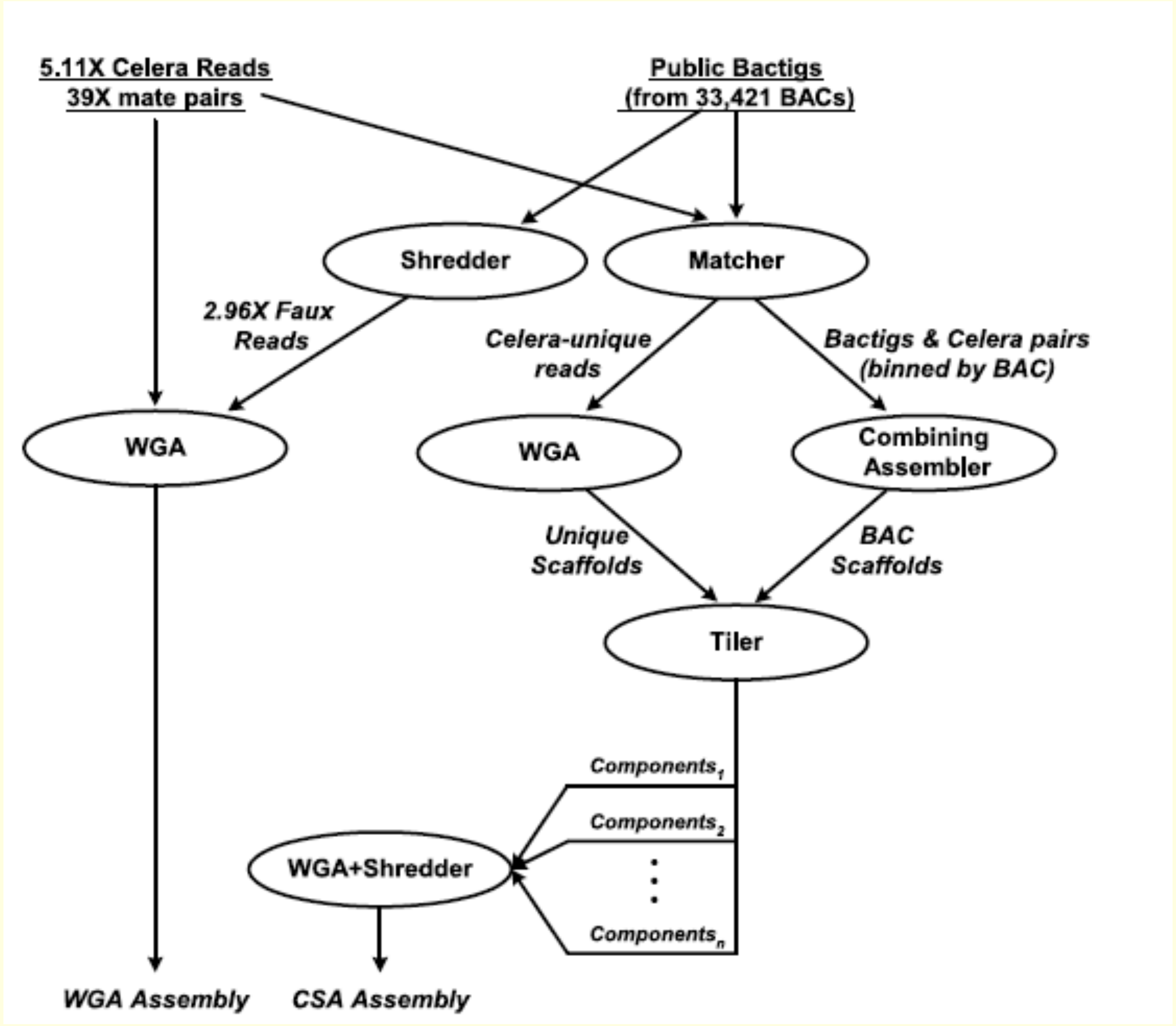

JULY 3, 2000 $3.50
www.time.com AOL Keyword: TIME
TIME
EXCLUSIVE Q&A
DR. LAURA ON THE OFFENSIVE
Francis Collins
J. Craig Venter
The inside story of how these bitter rivals mapped our DNA, the historic feat that changes medicine forever
Cracking The Code!
CELERA

NIH used a BAC-by-BAC strategy; Celera used a whole-genome shotgun assembly.

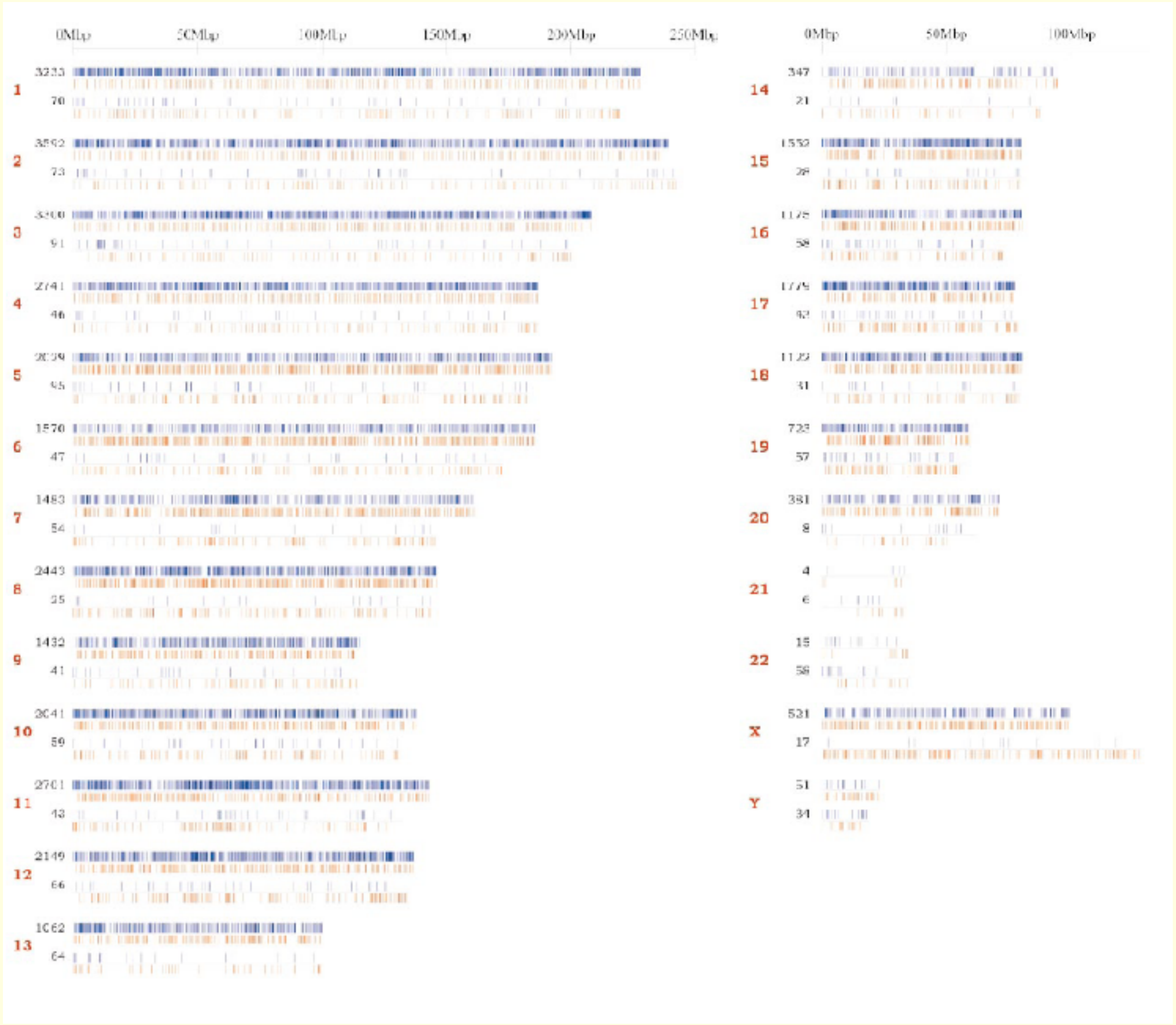
Celera used 300 sequencing machines in parallel to obtain 175,000 reads per day.

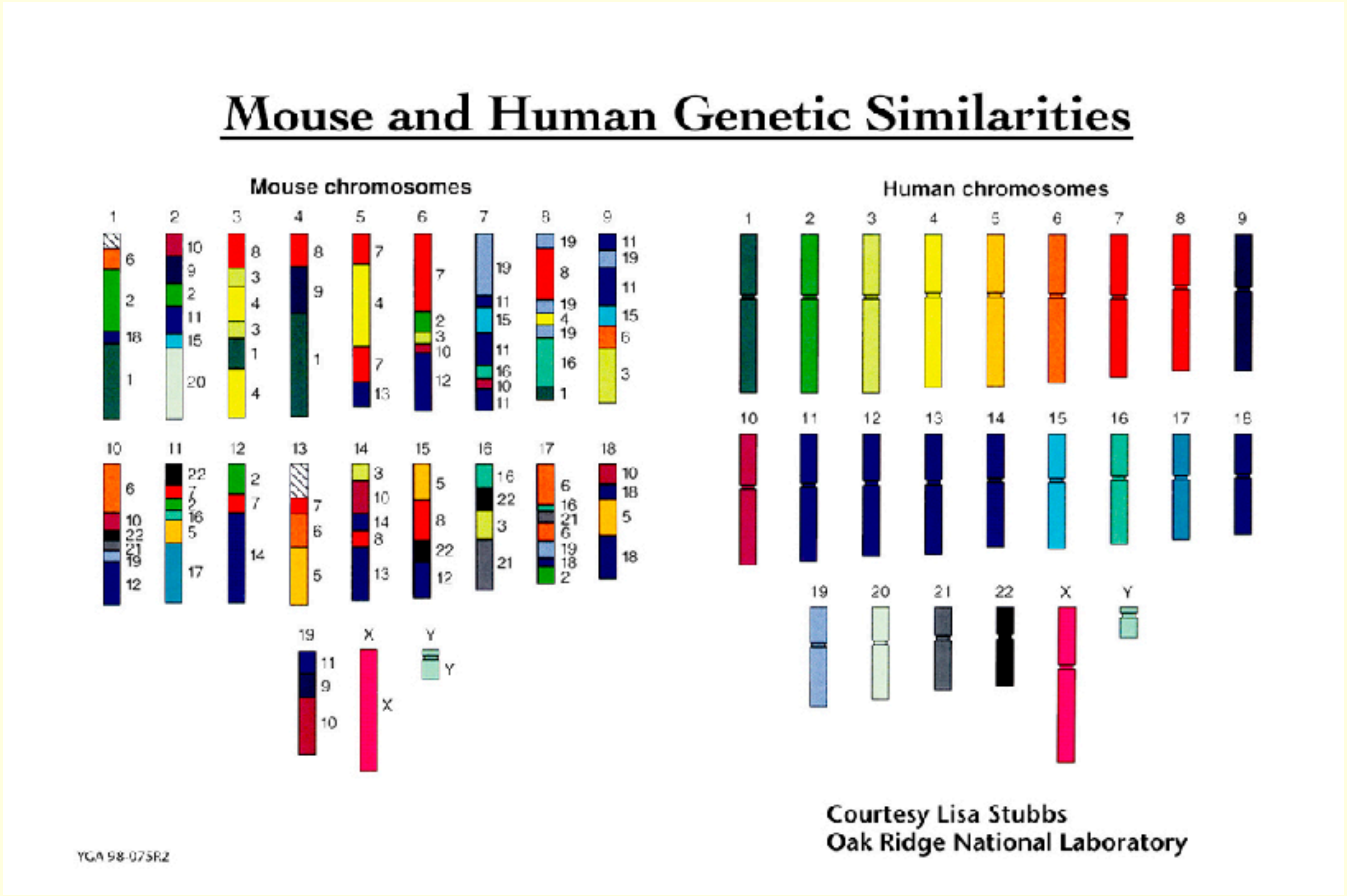
Efforts were combined, resulting in 8x coverage of the human genome; consensus sequence is 2.91 billion base pairs.

Overall timeline~1990-2003


5.11X Celera Reads
39X mate pairs
Public Bactigs
(from 33,421 BACs)
Shredder
Matcher
2.96X Faux Reads
Celera-unique reads
Bactigs & Celera pairs (binned by BAC)
WGA
WGA
Combining Assembler
Unique Scaffolds
BAC Scaffolds
Tiler
Components$_1$
Components$_2$
Components$_n$
WGA+Shredder
WGA Assembly
CSA Assembly

# The Big Picture





The race is on to sequence as many genomes as possible (http://www.genomesonline.org/).

# Cost and Logistics

- The HapMap project aims to capture differences in DNA sequence and chromosome variation between individuals.
- Cheap genome sequencing is necessary to obtain genome-wide sequence “biomarkers”.
- Whole-genome shotgun sequencing requires about 8x coverage for a human, requiring accurate “basecalling” of ~24Gbps. The fastest sequencers currently produce 5Mbps/hour.
- State of the art: 26 hours (fastest), $1000 (cheapest).