

# Exact String Matching and searching for SNPs

CMSC423

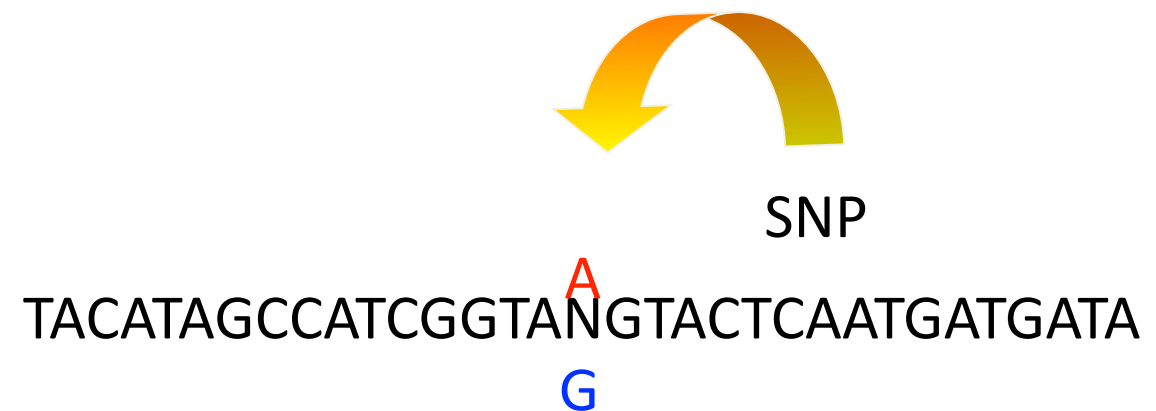
# What makes them different?



Much human variation is due to difference in  $\sim 6$  million base pairs (0.1 % of genome) referred to as SNPs

# Single Nucleotide Polymorphism (SNP)

Genomic DNA:



Three genotypes

AA

Mother

TACATAGCCATCGGTAAGTACTCAATGATGATA  
ATGTATCGGTAGCCATTTCATGAGTTACTACTAT

Father

TACATAGCCATCGGTAAGTACTCAATGATGATA  
ATGTATCGGTAGCCATTTCATGAGTTACTACTAT

AG

Mother

TACATAGCCATCGGTAAGTACTCAATGATGATA  
ATGTATCGGTAGCCATTTCATGAGTTACTACTAT

Father

TACATAGCCATCGGTAAGGTACTCAATGATGATA  
ATGTATCGGTAGCCATCCATGAGTTACTACTAT

GG

Mother

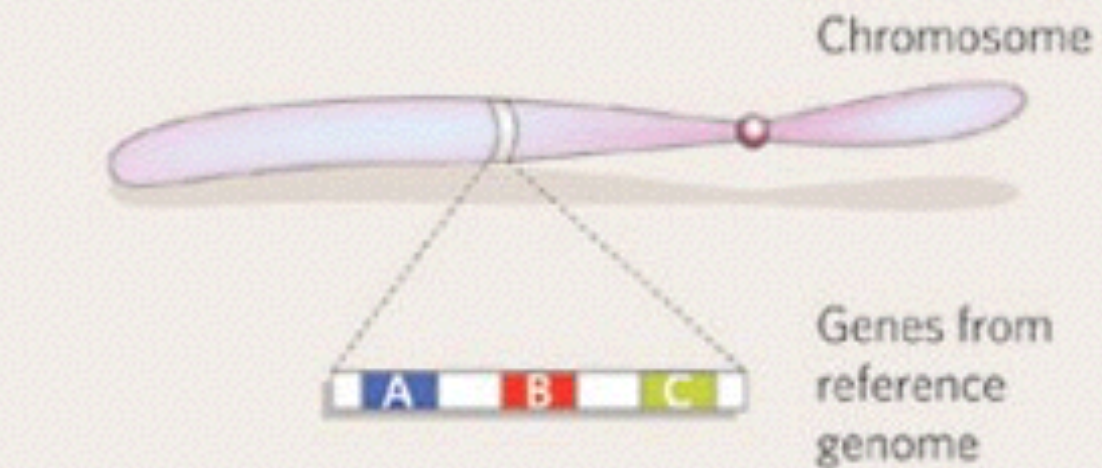
TACATAGCCATCGGTA**G**GTACTCAATGATGATA  
ATGTATCGGTAGCCAT**C**CATGAGTTACTACTAT

Father

TACATAGCCATCGGTA**G**GTACTCAATGATGATA  
ATGTATCGGTAGCCAT**C**CATGAGTTACTACTAT



## VARIATIONS IN OUR GENOMES



Deletion



Insertion



Inversion



Copy-number variant



Segmental duplication



[Check, Nature 437]



# DNA Sequence Variation in a Gene Can Change the Protein Produced by the Genetic Code

*Gene A from Person 1*

GCA AGA GAT AAT TGT...  
↓ ↓ ↓ ↓ ↓  
Ala Arg Asp Asn Cys ...  
1 2 3 4 5

Protein Products



*Gene A from Person 2*

Codon change made no difference in amino acid sequence

GCG AGA GAT AAT TGT...  
↓ ↓ ↓ ↓ ↓  
Ala Arg Asp Asn Cys ...  
1 2 3 4 5

*Gene A from Person 3*

Codon change resulted in a different amino acid at position 2

GCA AAA GAT AAT TGT...  
↓ ↓ ↓ ↓ ↓  
Ala Lys Asp Asn Cys ...  
1 2 3 4 5

OR





# Health or Disease?

**Person 1**

*DNA Sequence*  
**A A A T T T**



**Normal protein**



Some  
DNA  
variations  
have no  
negative  
effects

**Person 2**

**A A T T T T**



**Low or  
nonfunctioning protein**



Other  
variations  
lead to  
disease (e.g., sickle cell)  
or increased susceptibility  
to disease (e.g., lung cancer)

**Person 3**

**A A C T T T**





# PERSONAL GENOMICS



23andMe genetics just got personal.

Go

[log in](#)

| [claim codes](#)

| [blog](#)

| [help](#)

| [your cart](#)

welcome

how it works

genetics 101

store

about us

illumina<sup>®</sup>



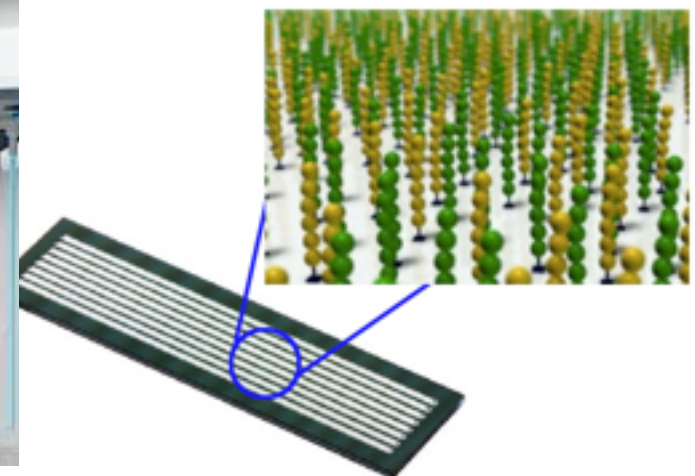
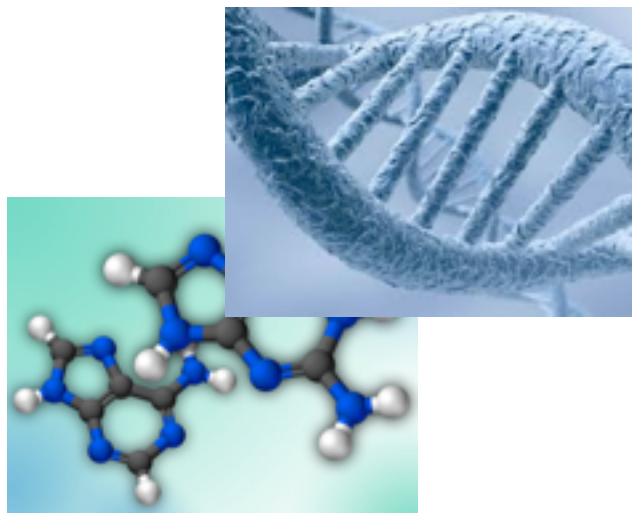
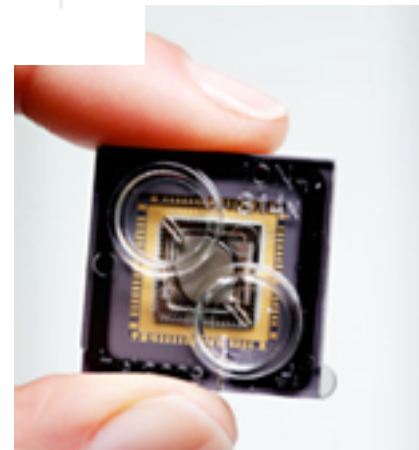
# Next-gen Sequencing



Semiconductor Sequencing for Life™

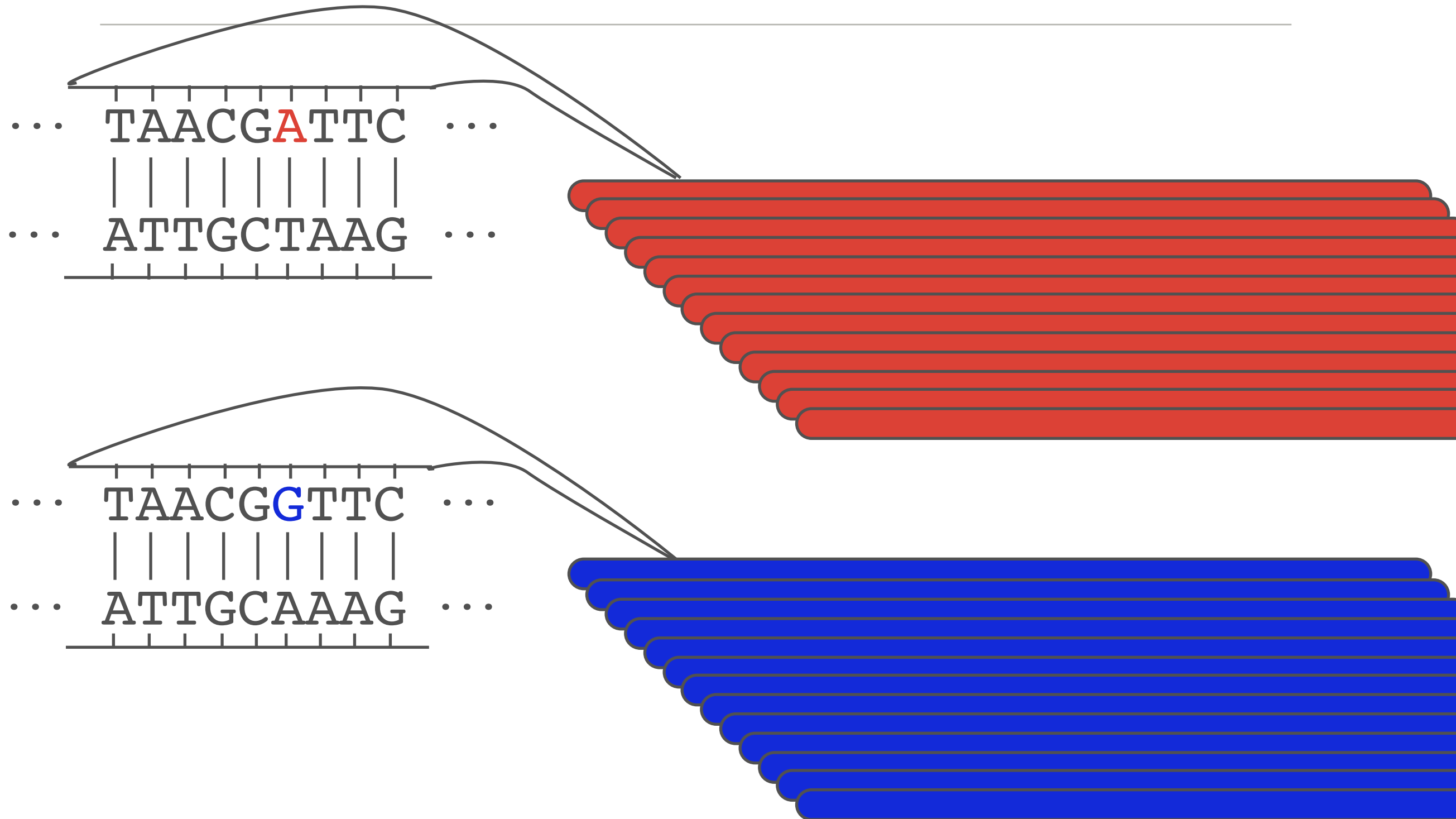


454  
SEQUENCING



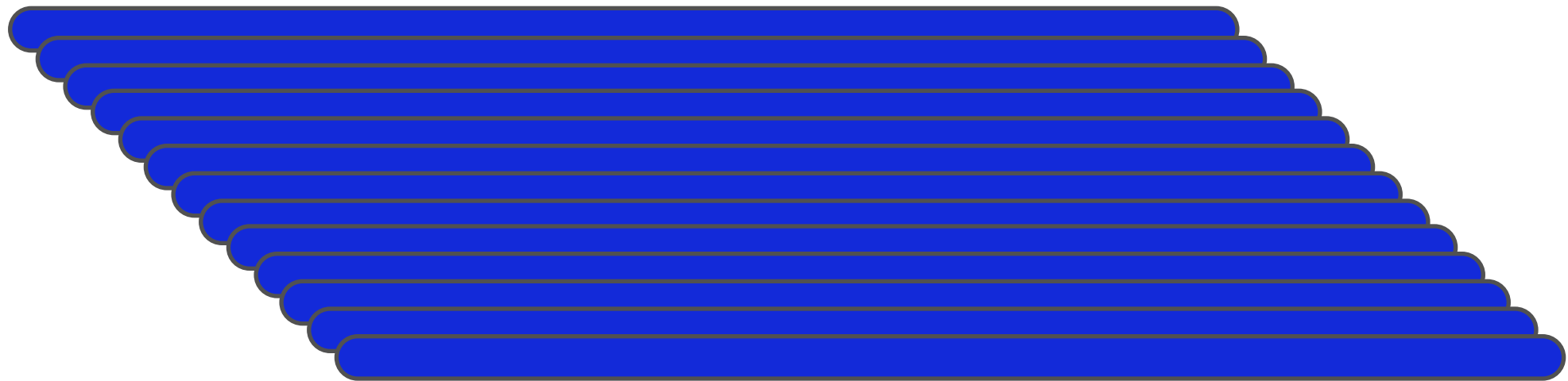
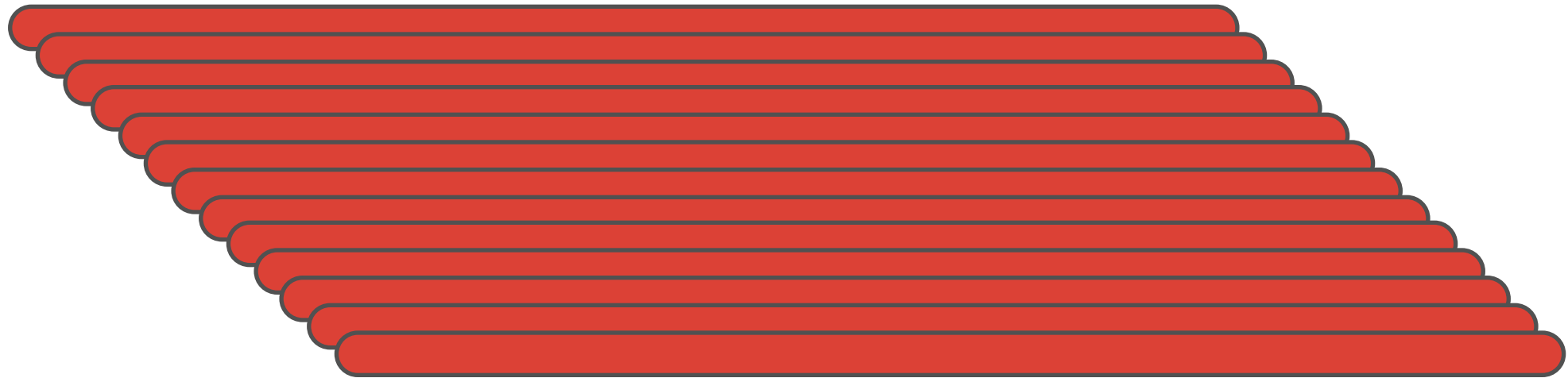


# SEC-GEN SEQUENCING FOR SNPs



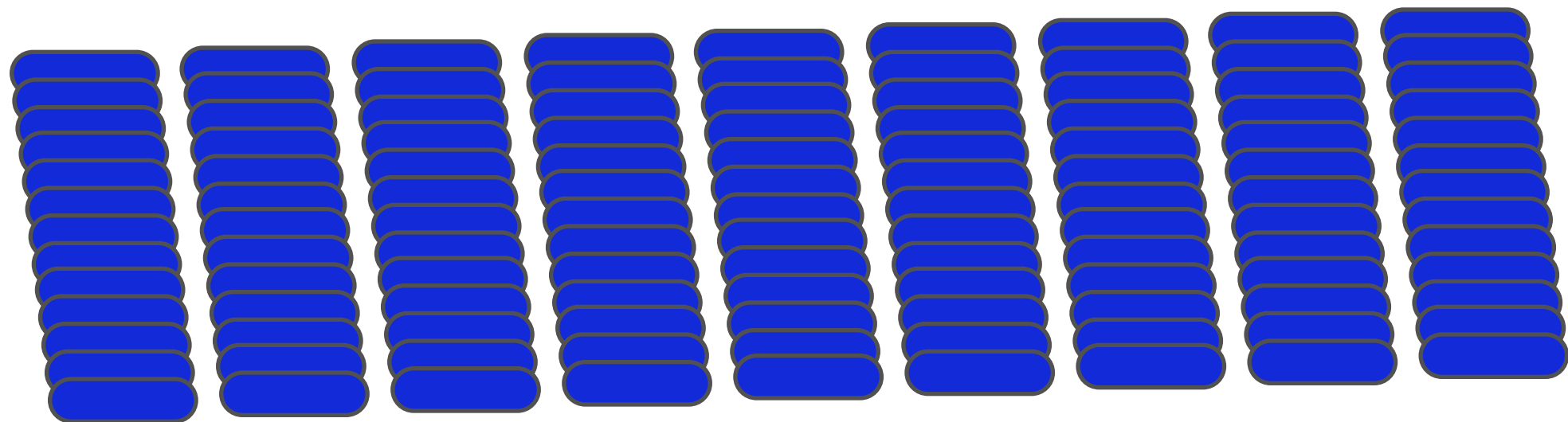
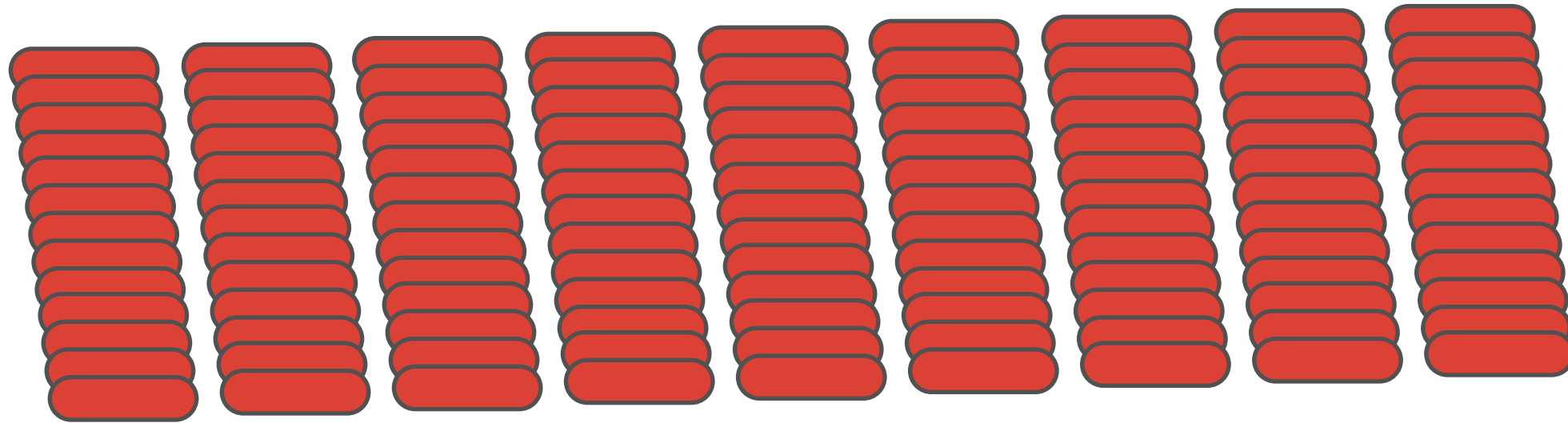
# SEC-GEN SEQUENCING FOR SNPs

---



# SEC-GEN SEQUENCING FOR SNPs

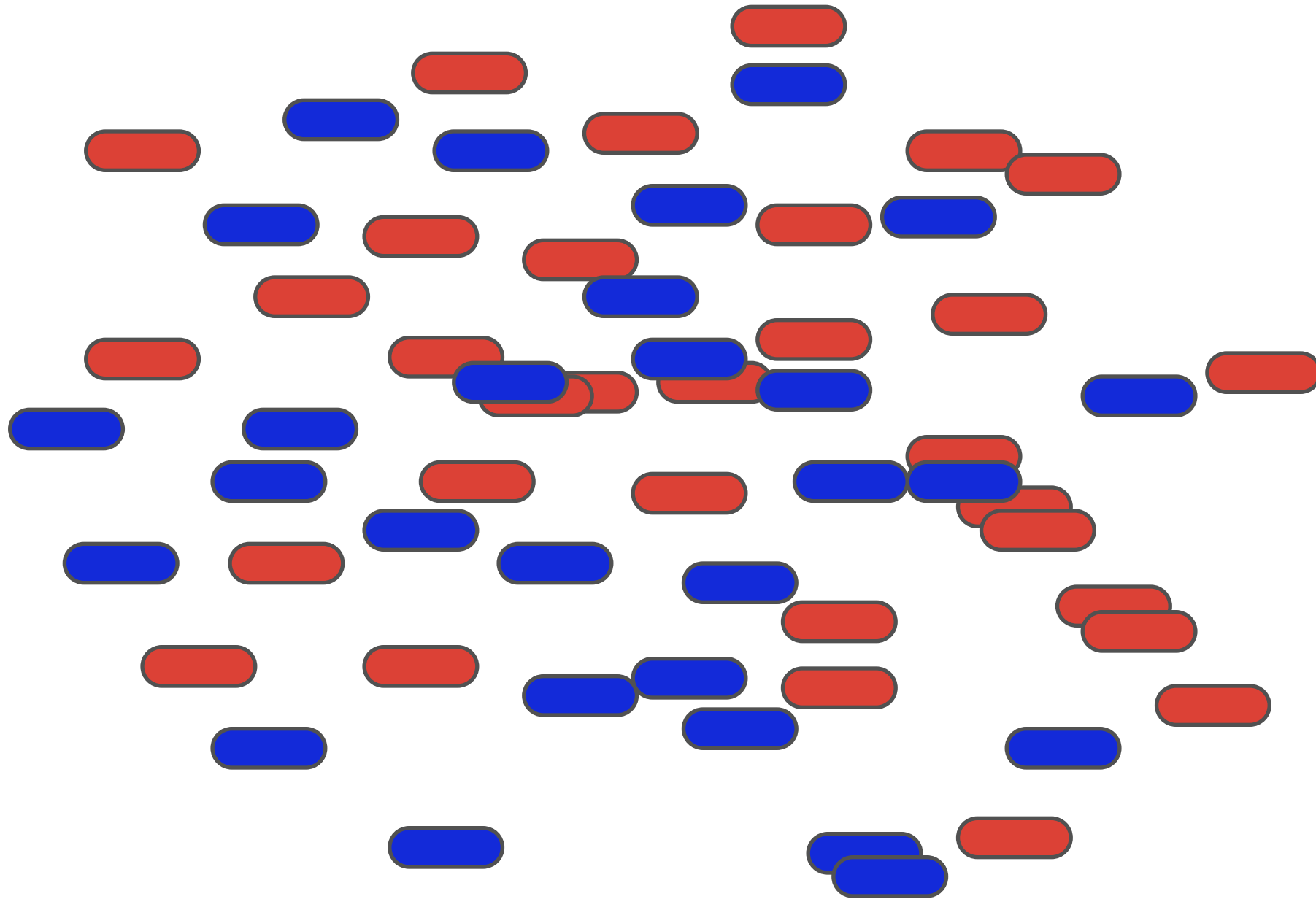
---





# SEC-GEN SEQUENCING FOR SNPs

---



# SEC-GEN SEQUENCING FOR SNPs

```
@HWI-EAS146:5:1:1:961#0/1
TCCGAGGCCAACCGAGGCTCCGCGCGCTGNNNNNNNNNNNNNNNN
+
BBBBA>A7B@;@BBBBA=BA=A\NNNNNNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:1:1595#0/1
TCAGGAGCAGGAGAGCTGGTCAGCAGNNNNNNNNNNNGNNNN
+
B9B@B<;BAA<@AB9=1>\NNNNNNNNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:1:1048#0/1
CTGGACTGCATCCTACCACCAACTCGTCCAANNNNNNNNNN
+
A=0767:>B@;A>79:<;>747\NNNNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:1:1607#0/1
CTCCTCTCAGGTCCCAGAGCACAGCCAAANNNNNANTNNCTNN
+
BBCCCCCBB7C7C=7>+<=>B7C7\NNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:1:1719#0/1
CACGATCTGGTTTATTGTAACCTCCGCTCANNNGNTNAGNNN
+
BCC7+<B=7BB5=ABA7B6BBB4BB7B\NNNNNNNNNNNN
@HWI-EAS146:5:1:2:947#0/1
CCAGGAGAAAGCATGTTTCAGTTCGAGCGNNANANCTGANNN
+
BBB9@77A7>AAB@>7B=7@.>B7B7\NNNNNNNNNNNNNN
@HWI-EAS146:5:1:2:563#0/1
CCAGCCCTCCCCATCTCCACCCTGTACCTNANCCCTGANNN
+
BBABAAB;AAABA77@SAAA:77>\NNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:2:1631#0/1
TGGGAACGAGCTACACTCTCCAGGCTCCTNCTCCGTNNNN
+
BBB@6@BBB@BBB@BBBABAABBB7;9BB@BA5<B:\NNNN
@HWI-EAS146:5:1:2:1420#0/1
CTCAAACCTCTGACCTTGGTGATCCACCGCCTNCGCCTCANN
+
BBB:BBBBAABAA7:(<=A8@>AAA7AB7>\NNNNNNNNNN
@HWI-EAS146:5:1:1:961#0/1
TCCGAGGCCAACCGAGGCTCCGCGCGCTGNNNNNNNNNNNN
+
BBBBA>A7B@;@BBBBA=BA=A\NNNNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:1:1595#0/1
TCAGGAGCAGGAGAGCTGGTCAGCAGNNNNNNNNNNNGNNNN
+
B9B@B<;BAA<@AB9=1>\NNNNNNNNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:1:1048#0/1
CTGGACTGCATCCTACCACCAACTCGTCCAANNNNNNNNNN
+
A=0767:>B@;A>79:<;>747\NNNNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:1:1607#0/1
CTCCTCTCAGGTCCCAGAGCACAGCCAAANNNNNANTNNCTNN
+
BBCCCCCBB7C7C=7>+<=>B7C7\NNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:1:1719#0/1
CACGATCTGGTTTATTGTAACCTCCGCTCANNNGNTNAGNNN
+
BCC7+<B=7BB5=ABA7B6BBB4BB7B\NNNNNNNNNNNN
@HWI-EAS146:5:1:2:947#0/1
CCAGGAGAAAGCATGTTTCAGTTCGAGCGNNANANCTGANNN
+
BBB9@77A7>AAB@>7B=7@.>B7B7\NNNNNNNNNNNNNN
@HWI-EAS146:5:1:2:563#0/1
CCAGCCCTCCCCATCTCCACCCTGTACCTNANCCCTGANNN
+
BBABAAB;AAABA77@SAAA:77>\NNNNNNNNNNNNNNNN
@HWI-EAS146:5:1:2:1631#0/1
TGGGAACGAGCTACACTCTCCAGGCTCCTNCTCCGTNNNN
+
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

Align

```
GTCGCAGTANCTGTCT
||||| |||||
GTCGCAGTATCTGTCT

GGATCTCGATATACC
||||| |||||
GGATCT-CGATATACC

AATCTGATCTTATTTT
||||| |||||
AATCTGATCTTATTTT

ATATATATATATATAT
||||| |||||
ATATATATATATATAT

TCTCTCCANNAGAGC
||||| |||||
TCTCTCCAGGAGAGC
```

Aggregate

Statistics

"Pileup" or "Coverage plot"

"Depth of coverage" = 14

Call: HET A, G  
p-value: 0.0023

"Coverage"

```
GTCGCAGTATCTGTCT
GTCGCAGTATCTGTNN
TGTCGCAGTATCTGTC
TATGTCGCAGTATCTG
TATATCGCAGTATCTT
TATATCGCAGTATCTG
NATATCGCAGTATNTG
CCCTATATCGCAGTAT
ACACCCTATGTCGCA
ACACCCTATCTCGCA
ACACCCTATGTCGCA
GA-CACCCTATGTCGC
CCGGA-CACCCTATAT
CCGGA-CACCCTATAT
GCCGGA-CACCCTATG
```

Reference

(slide courtesy of Ben Langmead)

TGGTATTTTTCGTCTGGGGGGTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTCGCAGTATCTGTCTTTGATTCTCCTGCCTCATCCTATTAT

# The problem

- Given:
  - 100's of millions of short reads: 100-200bp reads
  - A long reference genome (~3Bbp for human)
- Do:
  - Find high scoring scoring (fitting) alignments for each read
- What we know:
  - Dynamic programming solution for fitting alignment:
    - $1e8 * 1e9 * 1e2$  operations,  $1e9 * 1e2$  memory

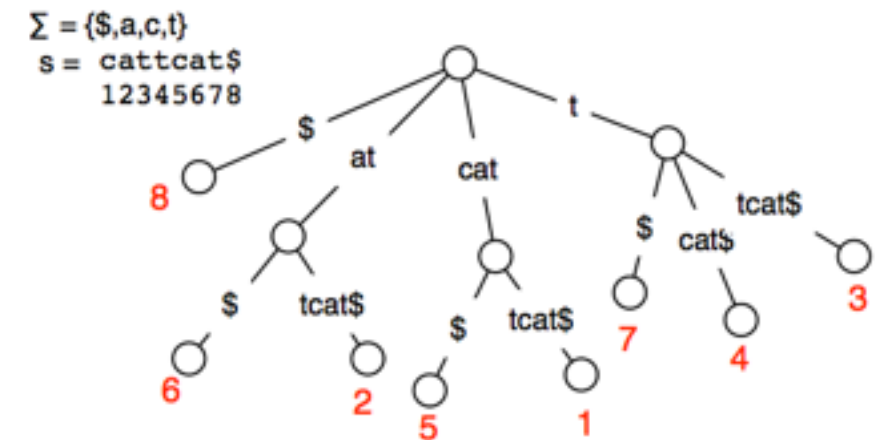


# Strategies

- What if we only allow a small number of substitutions?
  - Let's first try to find *exact* matches and work from those (the  $d+1$  trick in the midterm)
- We are aligning to the same reference 100's of millions of time
  - Is there preprocessing we can do to amortize time?
- Genomes are repetitive
  - Can we search for matches in the genome in a smart way?
  - Can we compress the genome, and search over the compressed representation?

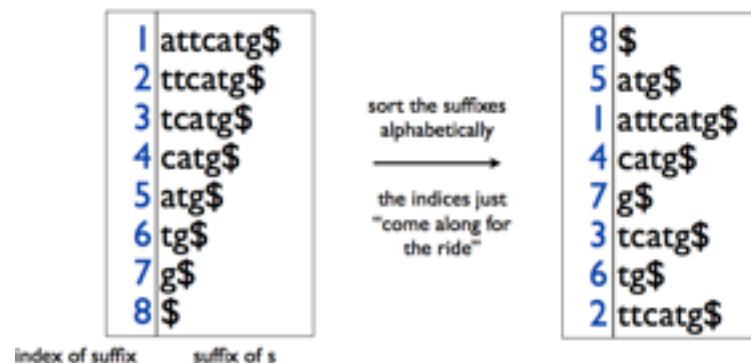
# Exact Matching

- Suffix Tries/Trees



- Suffix Arrays

$s = \text{attcatg\$}$



- Idea: lexicographically sort all the suffixes.
- Store the starting indices of the suffixes in an array.

- The Burrows-Wheeler transform

**banana**

banana\$  
 anana\$b  
 nana\$ba  
 ana\$ban  
 na\$bana  
 a\$banan  
 \$banana

sort

\$banana  
 a\$banan  
 ana\$ban  
 anana\$b  
 banana\$  
 nana\$ba  
 na\$banan

# State of the Art

- Bowtie: ultra-fast mapping of short reads to reference genome

**Bowtie**

An ultrafast memory-efficient short read aligner



**Bowtie** is an ultrafast, memory-efficient short read aligner. It aligns short DNA sequences (reads) to the human genome at a million 35-bp reads per hour. Bowtie indexes the genome with a Burrows-Wheeler index to keep its memory footprint small: 1 GB for the human genome (2.9 GB for paired-end).

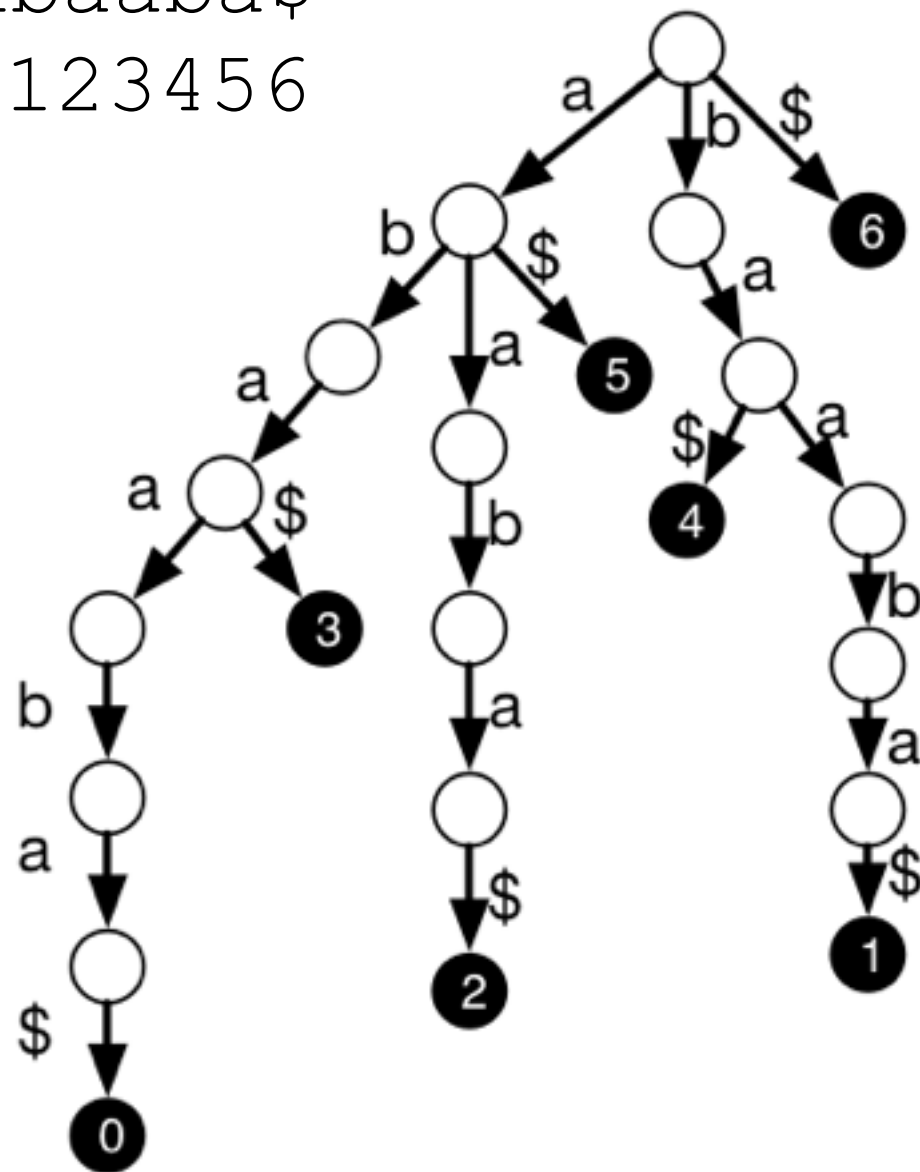
- <http://bowtie-bio.sourceforge.net>



First, we get some inspiration from KMP algorithm and  
fundamental preprocessing

# Suffix Trie

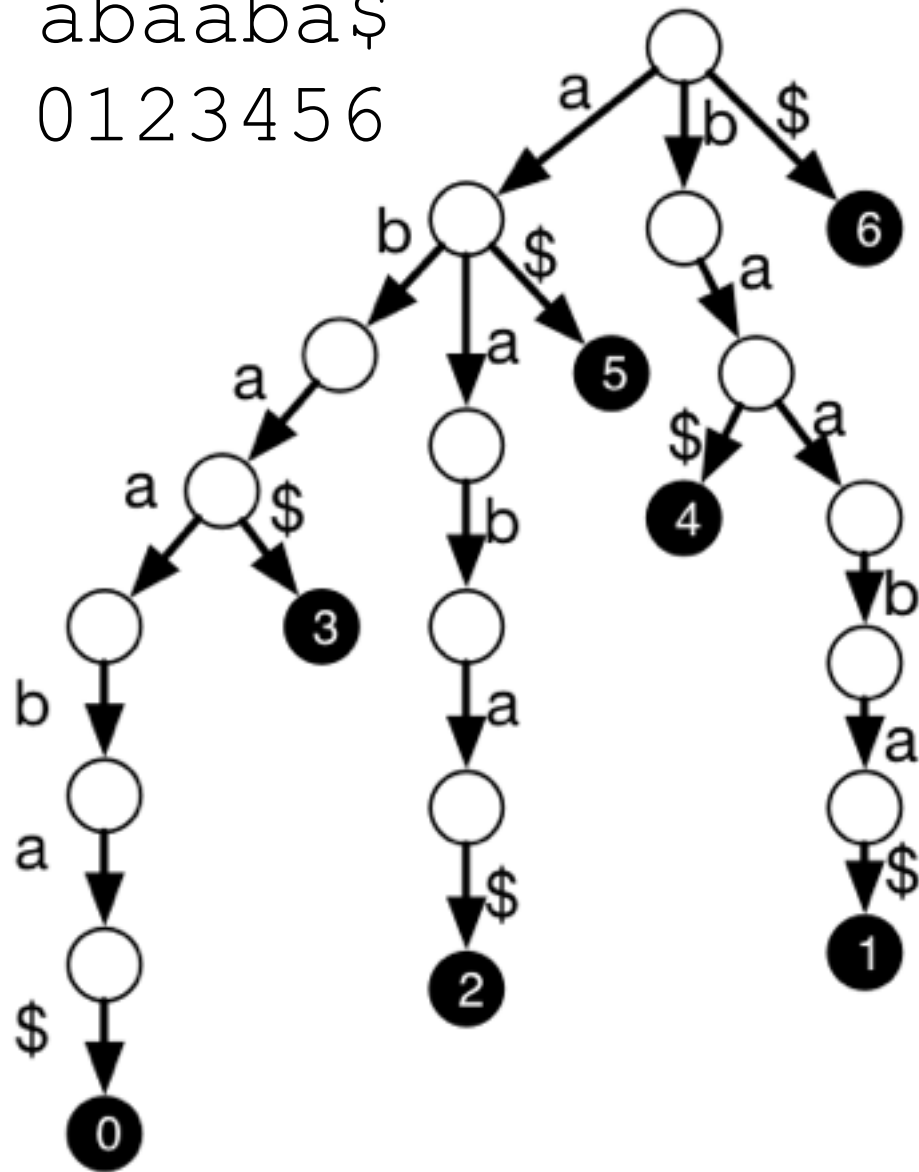
T: abaaba\$  
0123456



- Edges labeled with characters from alphabet.
- Each path from root to leaf corresponds to a *suffix* of T.

# Suffix Trie

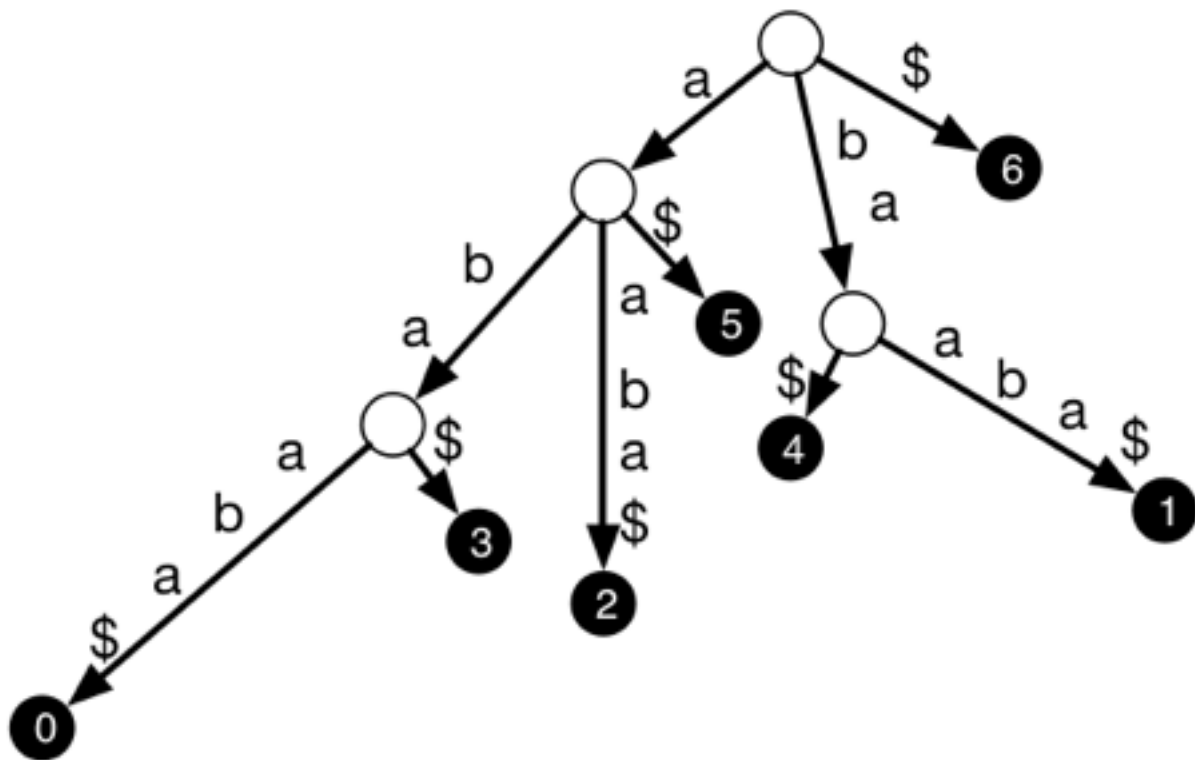
T:   abaaba\$  
      0123456



- Naive construction algorithm is  $O(|T|^2)$ 
  - There are linear time construction algorithms (see Gusfield)
- Memory requirement is also  $O(|T|^2)$
- Time to find matches  $O(|P|)$

# Suffix Tree

T: abaaba\$  
0123456

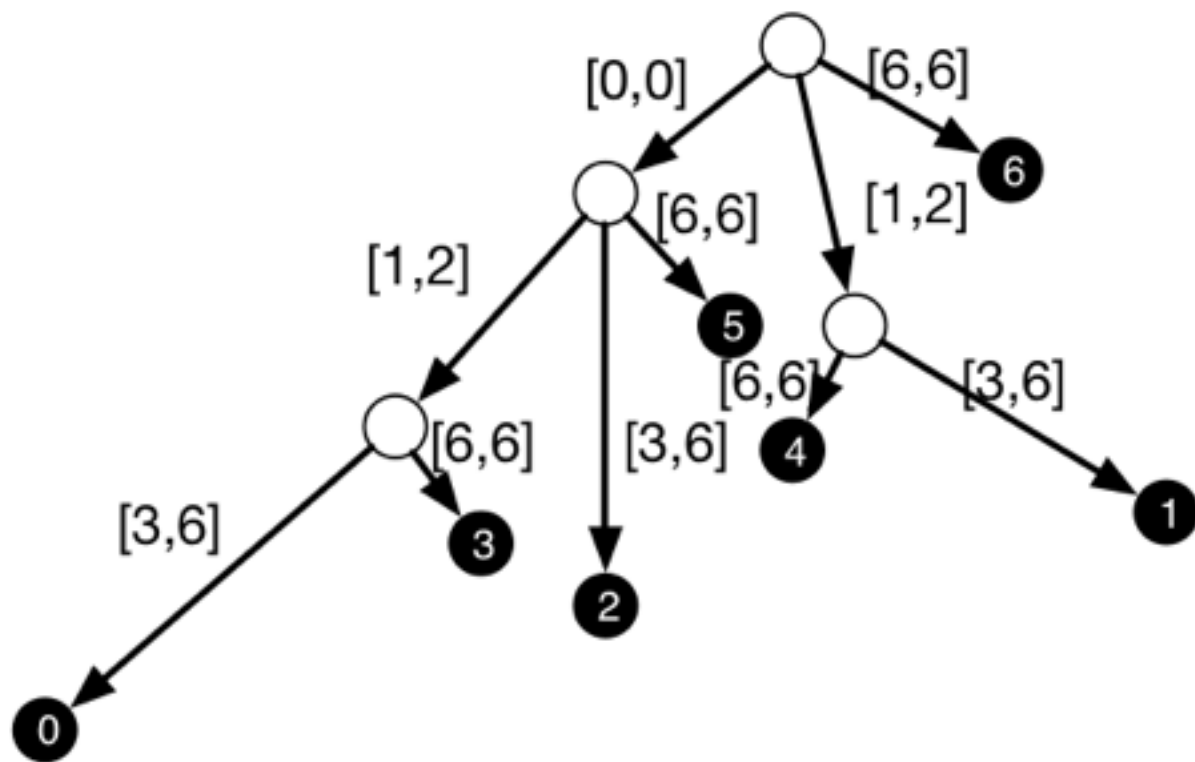


- Collapse non-branching nodes
  - #nodes  $O(|T|)$
- Memory requirement is *not*  $O(|T|)$ 
  - In the worst case, space required for edge labels is  $O(|T|)$



# Suffix Tree

T: abaaba\$  
0123456



- Collapse non-branching nodes
  - #nodes  $O(|T|)$
- Label edges with substring  $[start, end]$ 
  - $O(1)$  per edge
- Memory now  $O(|T|)$
- Construction algorithm  $O(|T|)$  (see Gusfield)

# Suffix Arrays

- Even though Suffix Trees are  $O(n)$  space, the constant hidden by the big-Oh notation is somewhat “big”:  $\approx 20$  bytes / character in good implementations.
- If you have a 10Gb genome, 20 bytes / character = 200Gb to store your suffix tree. “Linear” but large.
- Suffix arrays are a more efficient way to store the suffixes that can do most of what suffix trees can do, but just a bit slower.
- Slight space vs. time tradeoff.

# Example Suffix Array

$s = \text{attcatg\$}$

- Idea: lexicographically sort all the suffixes.
- Store the starting indices of the suffixes in an array.

1	attcatg\$
2	ttcatg\$
3	tcatg\$
4	catg\$
5	atg\$
6	tg\$
7	g\$
8	\$

index of suffix

suffix of  $s$

sort the suffixes  
alphabetically



the indices just  
“come along for  
the ride”

8	\$
5	atg\$
1	attcatg\$
4	catg\$
7	g\$
3	tcatg\$
6	tg\$
2	ttcatg\$

# Example Suffix Array

$s = \text{attcatg\$}$

- Idea: lexicographically sort all the suffixes.
- Store the starting indices of the suffixes in an array.

1	attcatg\$
2	ttcatg\$
3	tcatg\$
4	catg\$
5	atg\$
6	tg\$
7	g\$
8	\$

index of suffix

suffix of s

sort the suffixes  
alphabetically



the indices just  
“come along for  
the ride”

8
5
1
4
7
3
6
2



# Another Example Suffix Array

$s = \text{cattcat\$}$

- Idea: lexicographically sort all the suffixes.
- Store the starting indices of the suffixes in an array.

1	cattcat\$
2	attcat\$
3	ttcat\$
4	tcat\$
5	cat\$
6	at\$
7	t\$
8	\$

index of suffix

suffix of s

sort the suffixes  
alphabetically



the indices just  
“come along for  
the ride”

8	\$
6	at\$
2	attcat\$
5	cat\$
1	cattcat\$
7	t\$
4	tcat\$
3	ttcat\$

# Another Example Suffix Array

$s = \text{cattcat\$}$

- Idea: lexicographically sort all the suffixes.
- Store the starting indices of the suffixes in an array.

1	cattcat\$
2	attcat\$
3	ttcat\$
4	tcat\$
5	cat\$
6	at\$
7	t\$
8	\$

index of suffix

suffix of s

sort the suffixes  
alphabetically



the indices just  
“come along for  
the ride”

8
6
2
5
1
7
4
3

# Search via Suffix Arrays

$s = \text{cattcat\$}$

8	\$	
6	at\$	
2	attcat\$	← ✓
5	cat\$	
1	cattcat\$	←
7	t\$	
4	tcats\$	
3	ttcat\$	

- Does string “at” occur in  $s$ ?
- Binary search to find “at”.
- What about “tt”?

# Counting via Suffix Arrays

$s = \text{cattcat\$}$

8	\$
6	at\$
2	attcat\$
5	cat\$
1	cattcat\$
7	t\$
4	tcat\$
3	ttcat\$

- How many times does “at” occur in the string?
- All the suffixes that start with “at” will be next to each other in the array.
- Find one suffix that starts with “at” (using binary search).
- Then count the neighboring sequences that start with at.

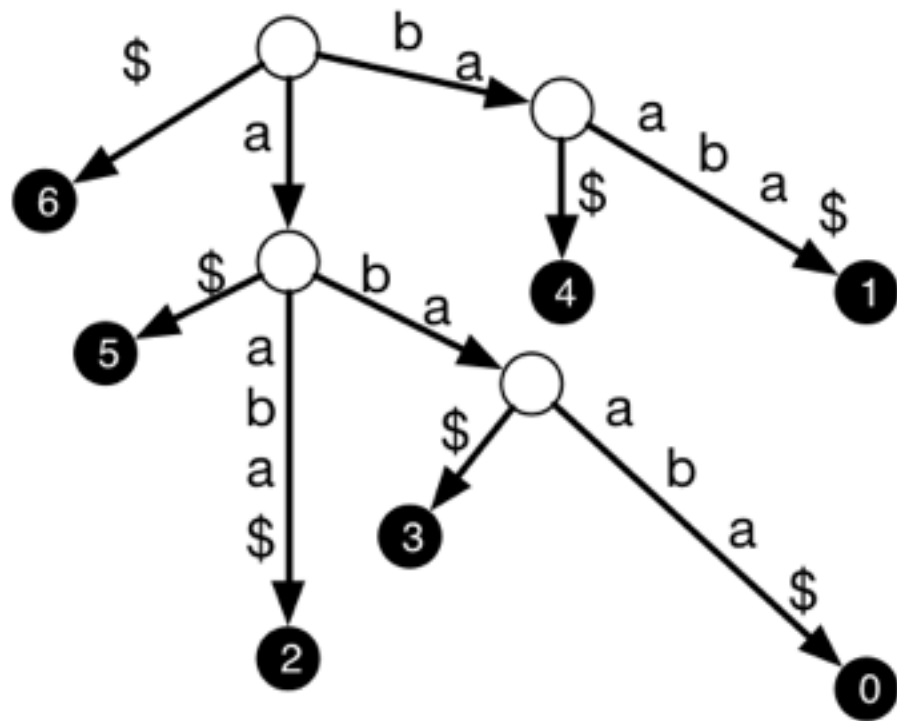
# Constructing Suffix Arrays

- Easy  $O(n^2 \log n)$  algorithm:  
sort the  $n$  suffixes, which takes  $O(n \log n)$  comparisons,  
where each comparison takes  $O(n)$ .
- There are several direct  $O(n)$  algorithms for constructing suffix arrays that use very little space.
- An simple  $O(n)$  algorithm: build the suffix tree, and exploit the relationship between suffix trees and suffix arrays (next slide)



# Relationship between Suffix Arrays and Suffix Trees

T: abaaba\$  
0123456



6	\$
5	a\$
2	aaba\$
3	aba\$
0	abaaba\$
4	ba\$
1	baaa\$

Build suffix trees with edge labels sorted lexicographically  
Order of leaves: 6,5,2,3,0,4,1

# Recap

Structure	Processing Time	Memory	Search
Suffix Trie	$O( T )$	$O( T ^2)$	$O( P )$
Suffix Tree	$O( T )$	$O( T )^*$	$O( P )$
Suffix Array	$O( T )$	$O( T )$ (but <b>much</b> smaller than Suffix Tree)	$O( P \log_2 T )$

\*In best implementations about 20 bytes per character (as opposed to 4 bytes for suffix array)