Suffix Arrays and Burrows-Wheeler transform

Advanced Bioinformatics for Next-Generation Sequencing

d. 05-09-2023

Rasmus Amund Henriksen, Abigail Ramsøe and Thorfinn Sand Korneliussen

UNIVERSITY OF COPENHAGEN

Content

- Introduction
- String terminology

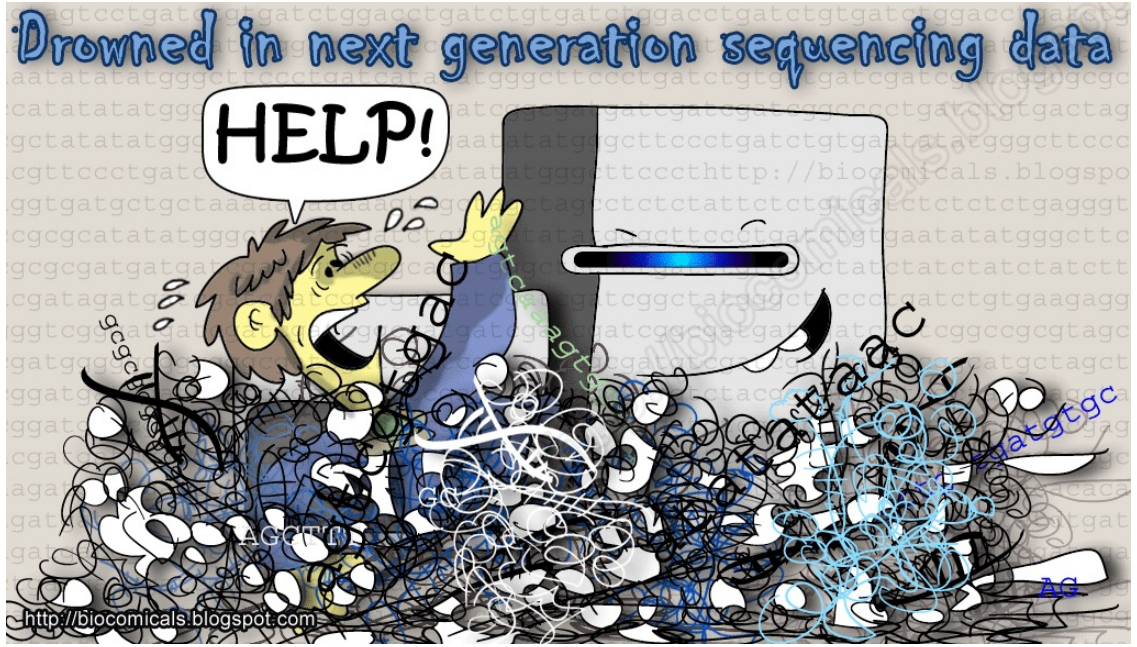
- Suffix trie
- Suffix tree
- Suffix Arrays

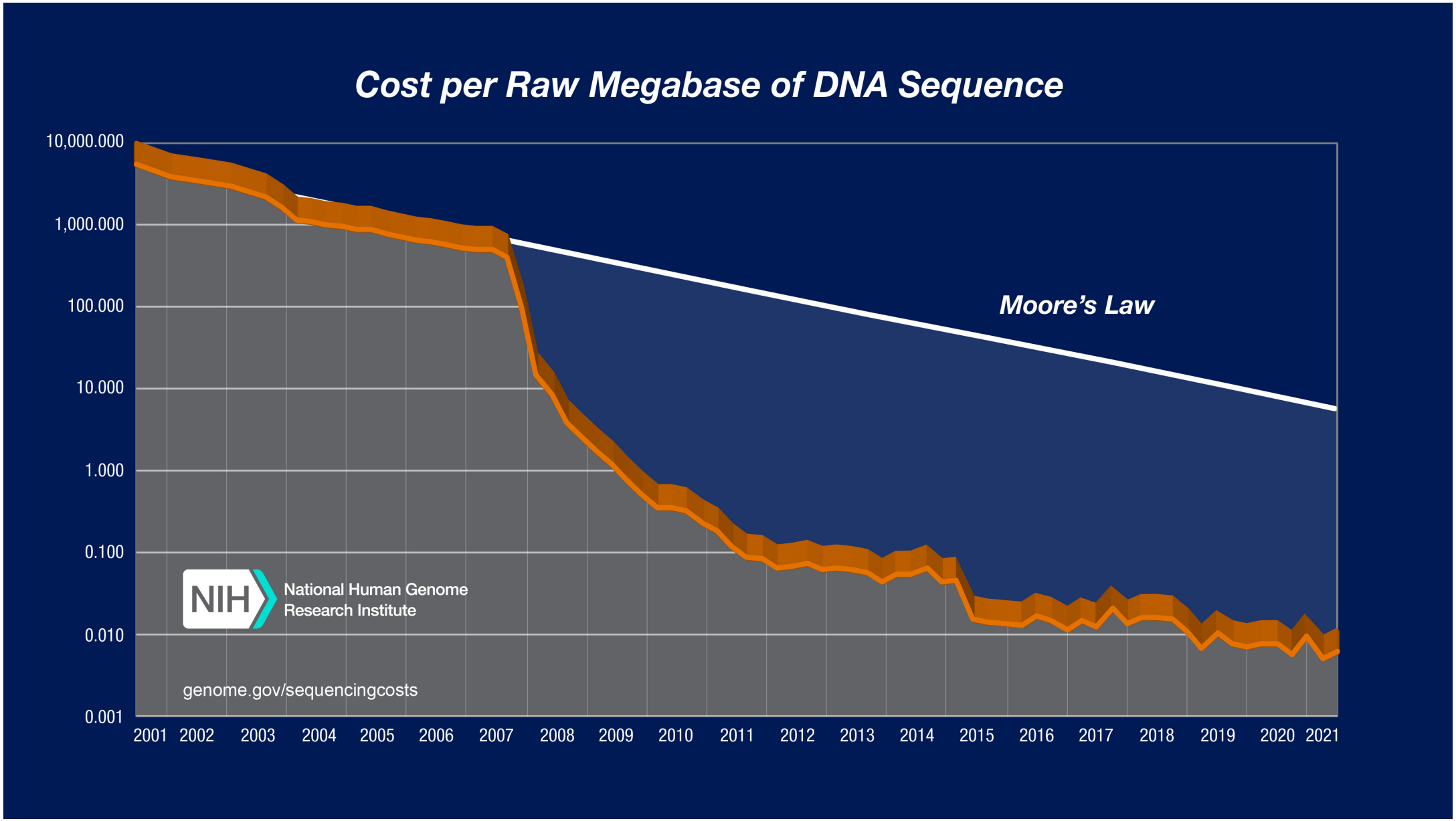
- Burrows–wheeler transform
- Alignment
- Samtools
- Ancient DNA analysis

INTRODUCTION

Motivation

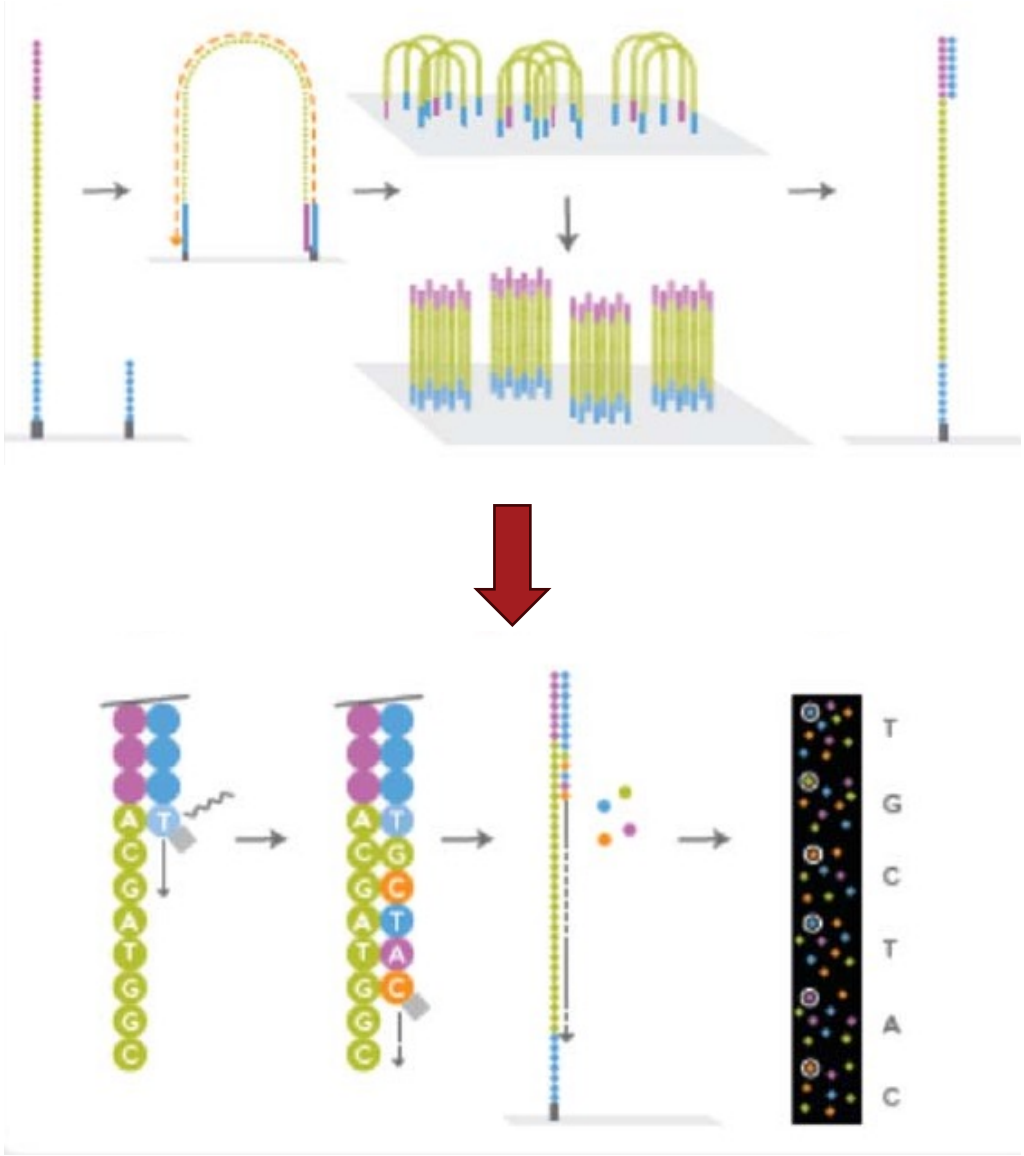
- Next generation sequencing data we can sequence whole genomes or regions
- Yields millions of sequencing reads




Cost per Raw Megabase of DNA Sequence
10,000.000
1,000.000
100.000
10.000
1.000
0.100
0.010
0.001
Moore's Law
NIH
National Human Genome Research Institute
genome.gov/sequencingcosts
2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021

DNA sequencing – 2nd generation

- Illumina Hiseq
  - DNA fragments are loaded to a flow cell
  - The opposite end hybridize to complementary nucleotide
  - Amplified
  - Sequenced using flouroscent nucleotides

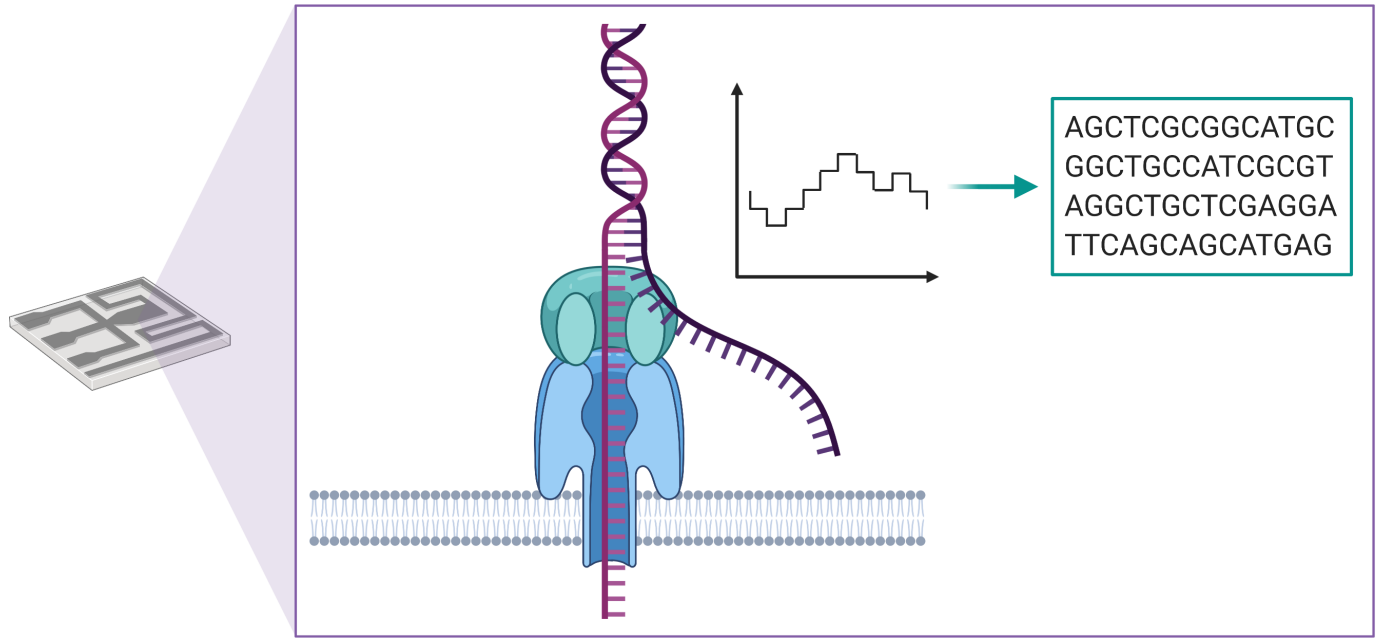


DNA sequencing – 3rd generation

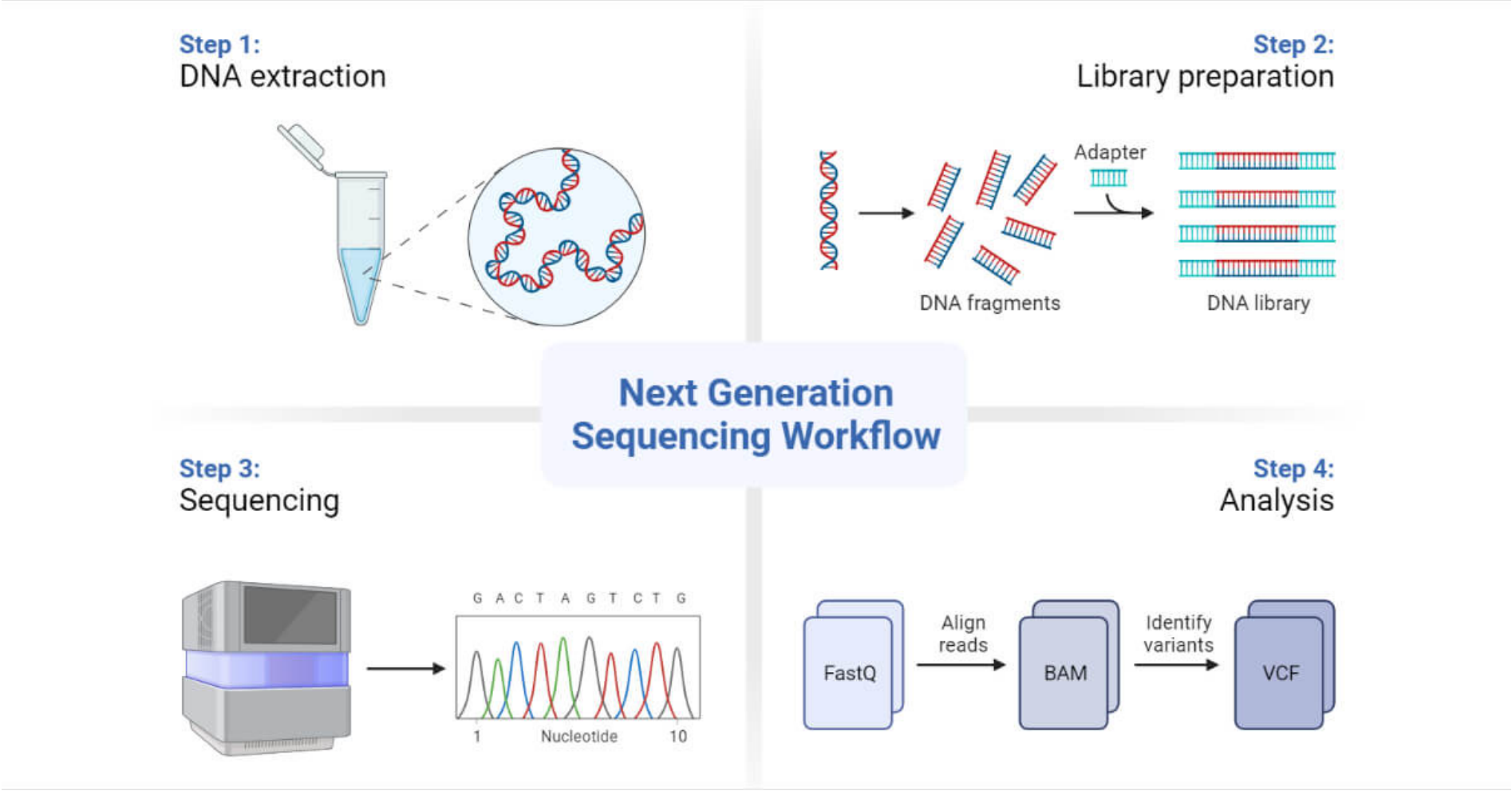
- Long-read sequencing
  oxford nanopore

- Measures the disruption of current across a membrance when the nucleotides passes through a pore

- No upper limt

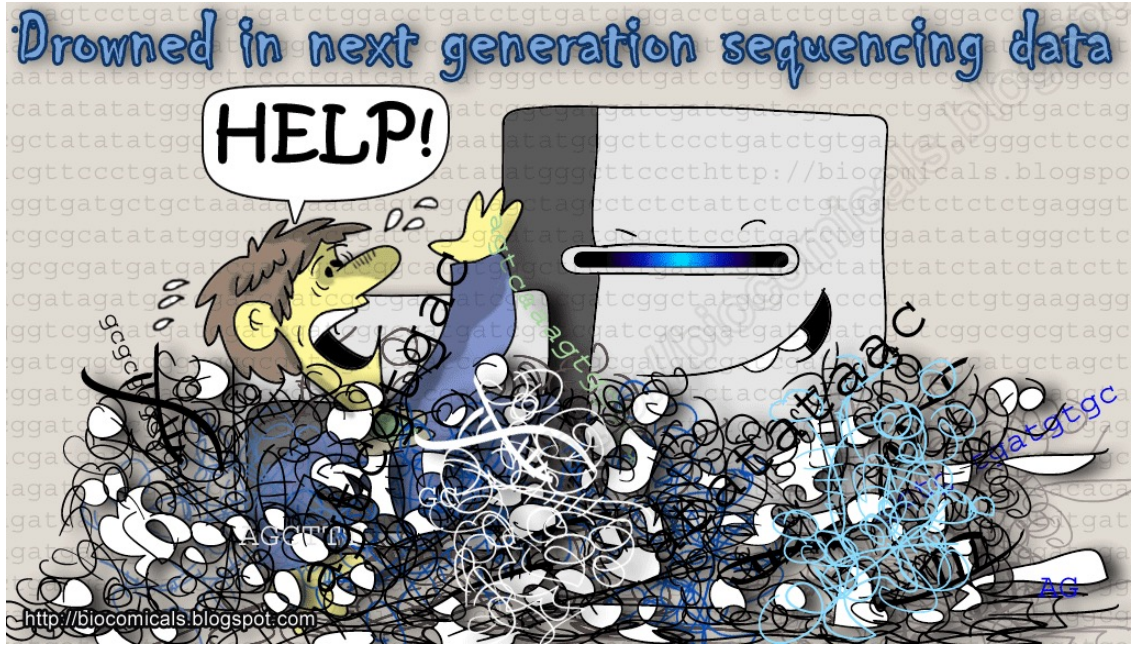


Classical sequencing and bioinformatical pipeline



Motivation

- Sequence reads which can be thought of as random, short substring of the genome
- Match their DNA to a reference genome, to identify the origin. Such as the human genome.
- Called reference-based alignment.



Problem

- Lets say we have a sequencing run with over 100 million reads.
  After processing, the reads are between 75 - 150 nucleotide long.
- We would like to know if these sequences are in the human genome, and if so where. The human genome has a size of 3Gb.
- A naïve approach would be to simply search for a sequence such as TCTGAGCGGAGGAGAG (n=16), this takes ~ 6 seconds
- But querying 100 million reads will take over 20 years

- Metagenomics studies uses genetic material recovered directly from environmental samples so to identify all organism we need a reference databases of 1.5 Tb.
- Which is why we need faster search algorithms and more efficient data structures to perform various string algorithms

Terminology:

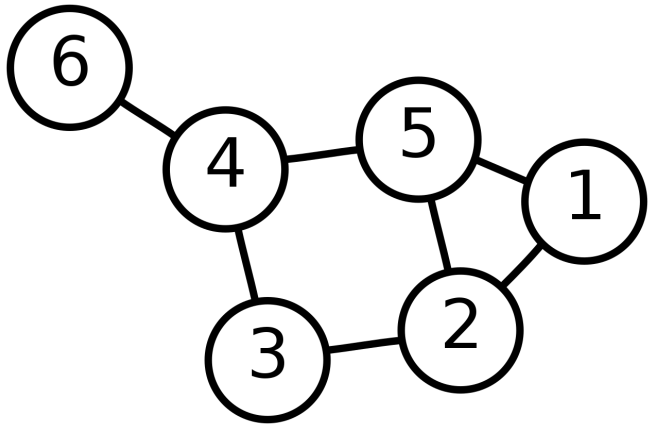
All the algorithms use string sorting and searching to identify the location of the sequencing reads

- String: Sequence of characters
- Substring: contiguous sequence of characters within the string
- Prefix and Suffix: special substrings occurring in the beginning and end of a string

  S = Banana, substring = ana, prefix = Ban, Suffix = nana
- Alphabet of S is $\Sigma = \{a,b,n\}$
- DNA alphabet $\Sigma = \{A,G,C,T\}$ (4)
  Protein alphabet $\Sigma = \{A,R,N,D,C,\ldots,V\}$ (20)

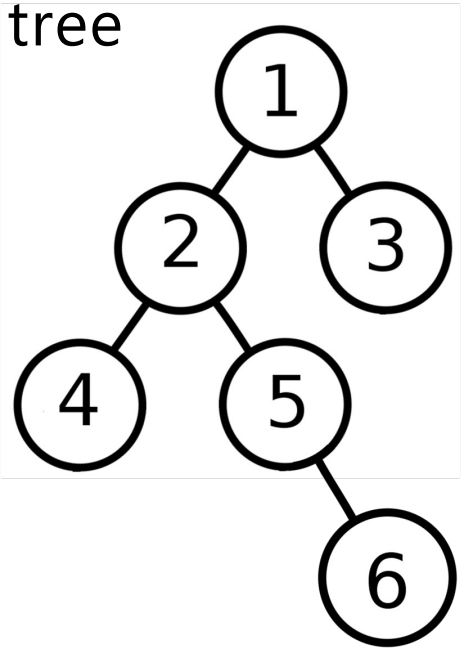
Graph

- Models relation between objects
- Collection of nodes and edges
- Each node can have any number of edges
- A cycle can be formed



Trees

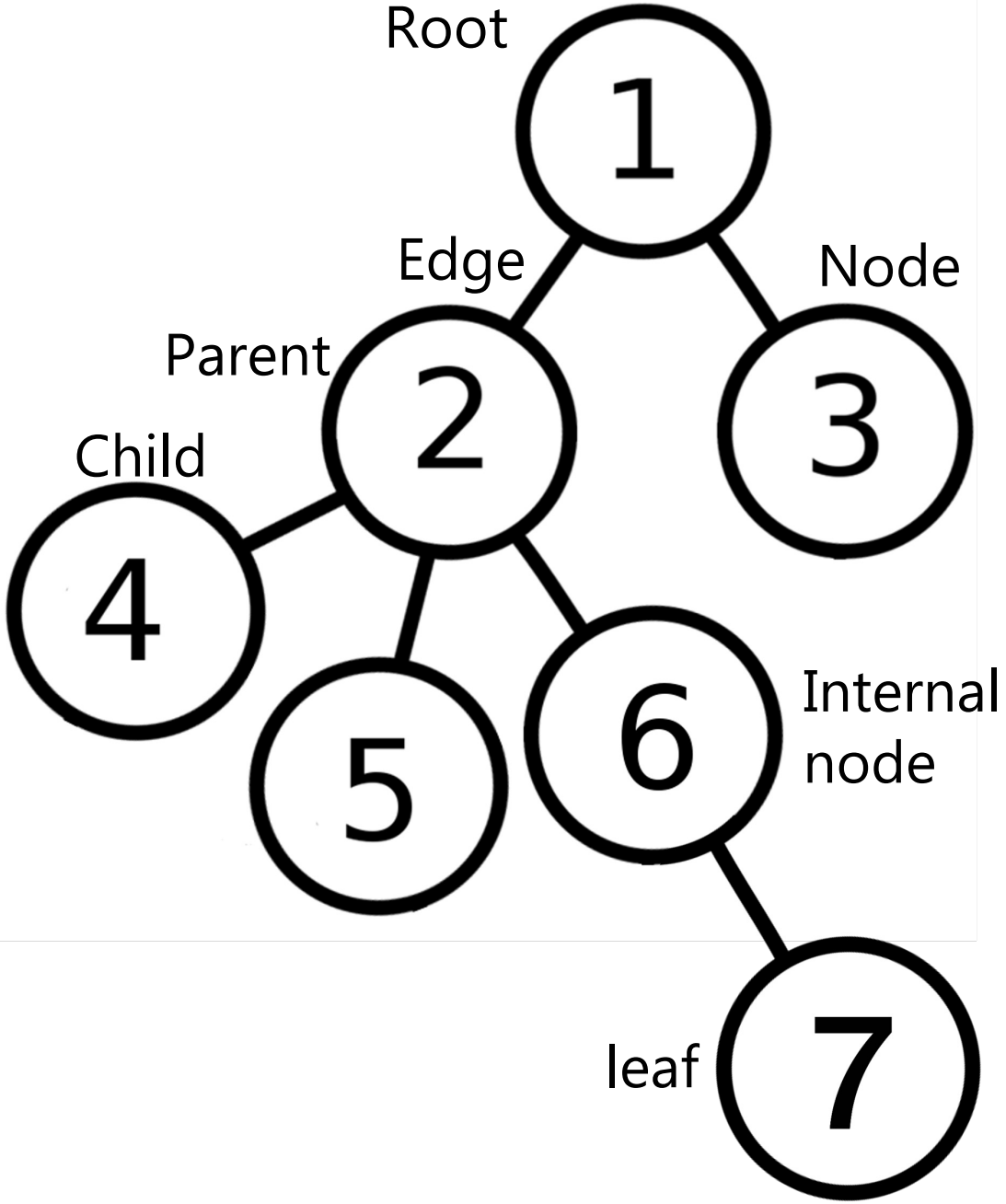
- Terminology from graphs
- Graph where any two nodes are connected by exactly one path
- A cycle cannot be formed
- Binary tree can have most 2 edges down the tree



Rooted Tree

- Set of nodes (n), Set of edges (n-1)
- Root: Top node
- Parent: a node which has a branch from it to any other node
- Child: immediately succeeds the parent
- Leafs: nodes with no children
- Internal nodes: nodes which is not a leaf

- Breadth: number of leaves
- Depth: distance between the node and the root
- Height: distance between the node and the leaf node in longest path.



SUFFIX TRIE, SUFFIX TREE & SUFFIX ARRAY

Suffix – introduction – BANANA

- We need to represents all possible suffixes of a sequence
- BANANA (n=6) with n suffixes
- Termination character $
- avoids confusion when comparing strings of different lengths, in case a suffix is a prefix of another suffix

BANANA$
ANANA$
NANA$
ANA$
NA$
A$

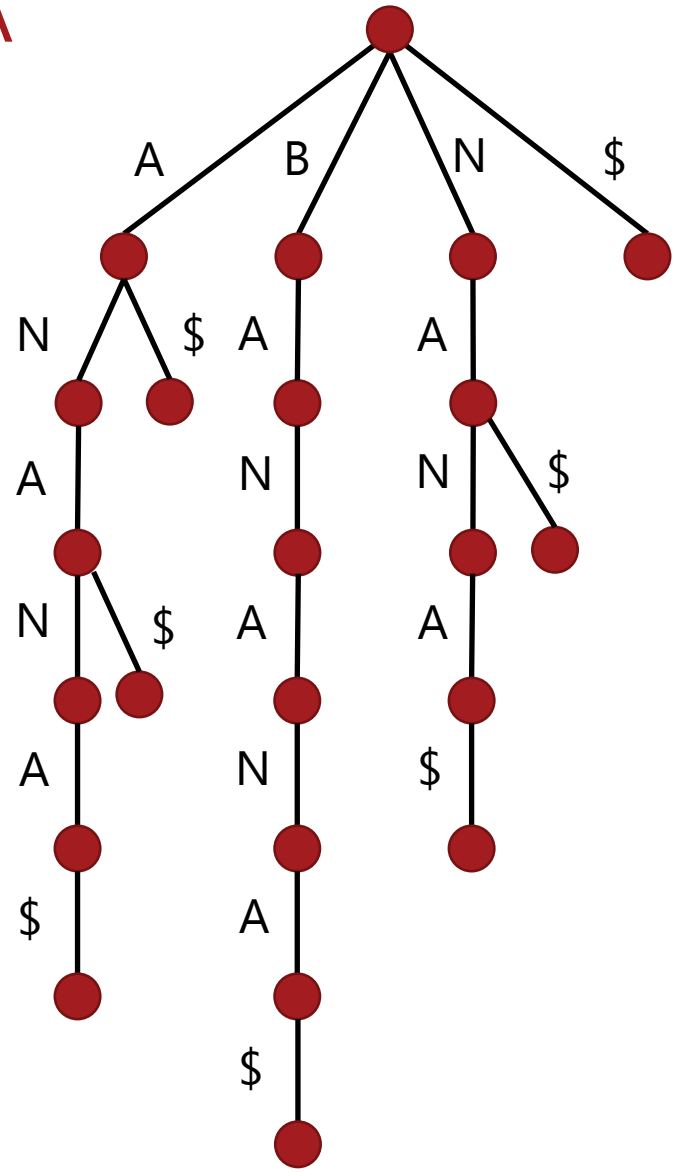
Suffix – introduction – GATCGGATGGCA (n=12)

- Sequence of n characters → n suffixes
- Human genome is 3.2 billion nucleotides
- 100 million read of 150 bp, Long-read sequencing yielding sequences >1kb.
- We need a datastructure to accomplish this - Suffix Trie data structure to contain all suffixes

GATCGGATGGCA$
ATCGGATGGCA$
TCGGATGGCA$
CGGATGGCA$
GGATGGCA$
GATGGCA$
ATGGCA$
TGGCA$
GGCA$
GCA$
CA$
A$

Suffix trie – introduction – BANANA

- A trie (“try”) is a tree representing the all the suffixes
  - Each edge is labelled with a single character from the alphabet Σ = {A,B,N}
  - No two edges starting at same node can have a string label beginning with the same character c ∈ Σ
- Each path from the root to leaf represent one suffix, one character at a time

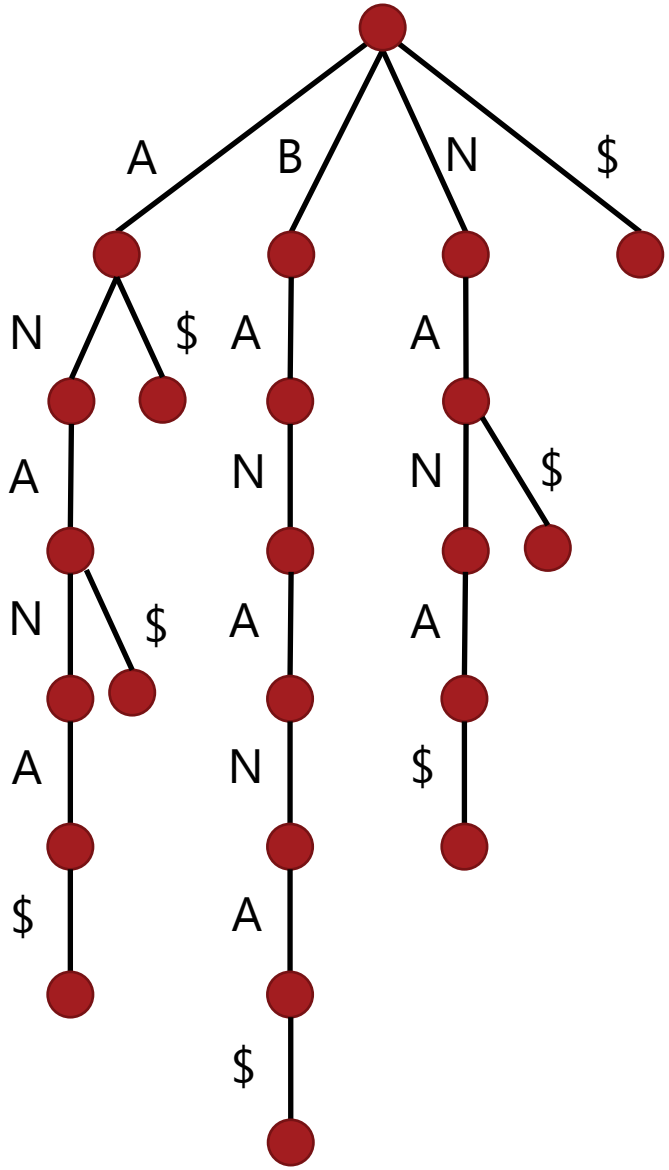


BANANA$
ANANA$
NANA$
ANA$
NA$
A$

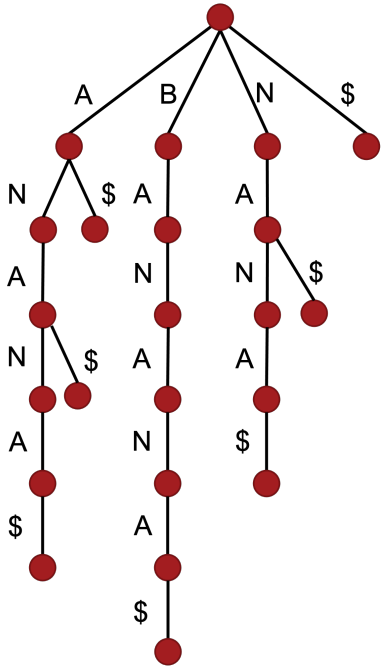
Suffix trie

T = BANANA (reference), S = NA (query)

- How do we check if string S is a substring of T?
  - Start at the root and match characters
- How do we check if string S is a suffix of T?
  - Start at the root and match characters until termination
- How do we count number of times string S occurs as a substring of T?
  - Search through the branch with the prefix of our query sequence
  - Report the number of leaf nodes



Longest common substring problem:



- We want to see if two sequences are related
- Looking for a common substring between two sequences

BANANA
ANANAS

- We could create a suffix trie of one of them and follow the path matching our query

Q: What is the longest substring of these two sequences?

- Longest common substring = ANANA (n=5)

Q: Which longest common substring appear multiple times?

- ANA

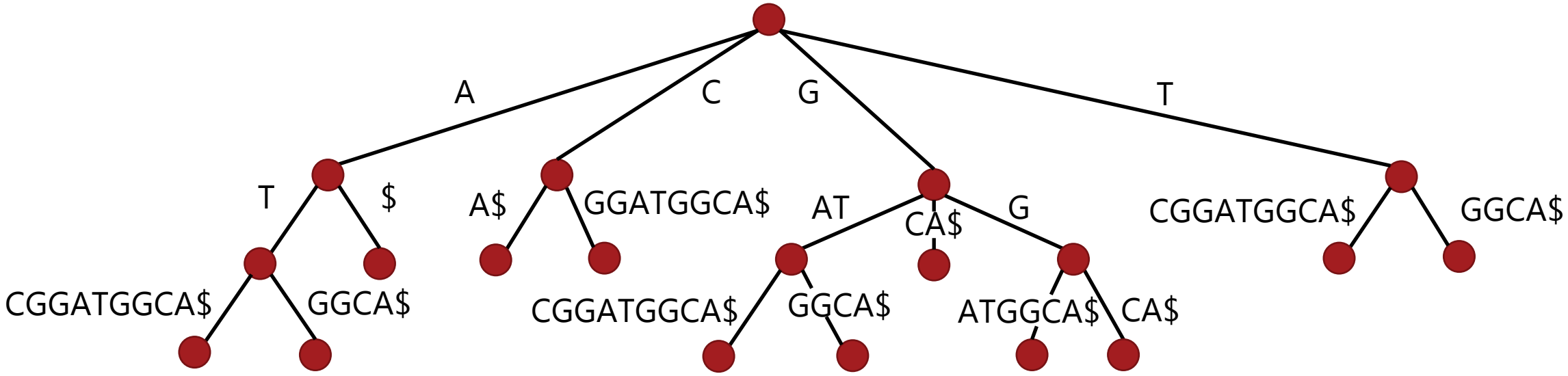
Suffix tree

- Suffix Trie every edge had a character
- We can preprocess the set of suffixes to minimize the number of nodes
- Combine suffixes that share prefixes, to condense the labels of the edges in our suffix trie
- Decrease number of comparisons
- Naïve approach is to build first edge from the root labelled with longest suffix
  then add the 2nd longest, 3rd longest etc.

GATCGGATGGCA$
ATCGGATGGCA$
TCGGATGGCA$
CGGATGGCA$
GGATGGCA$
GATGGCA$
ATGGCA$
TGGCA$
GGCA$
GCA$
CA$
A$

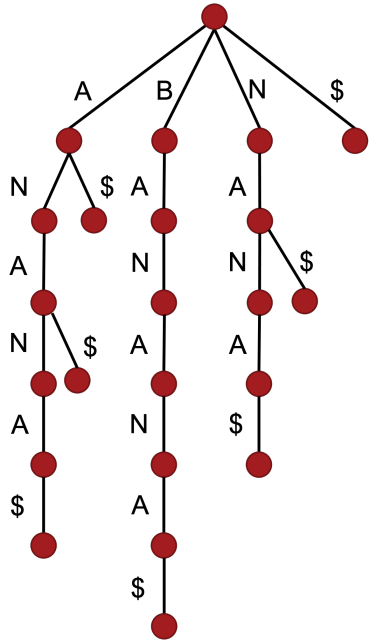
Suffix tree: T = GATCGGATGGCA$

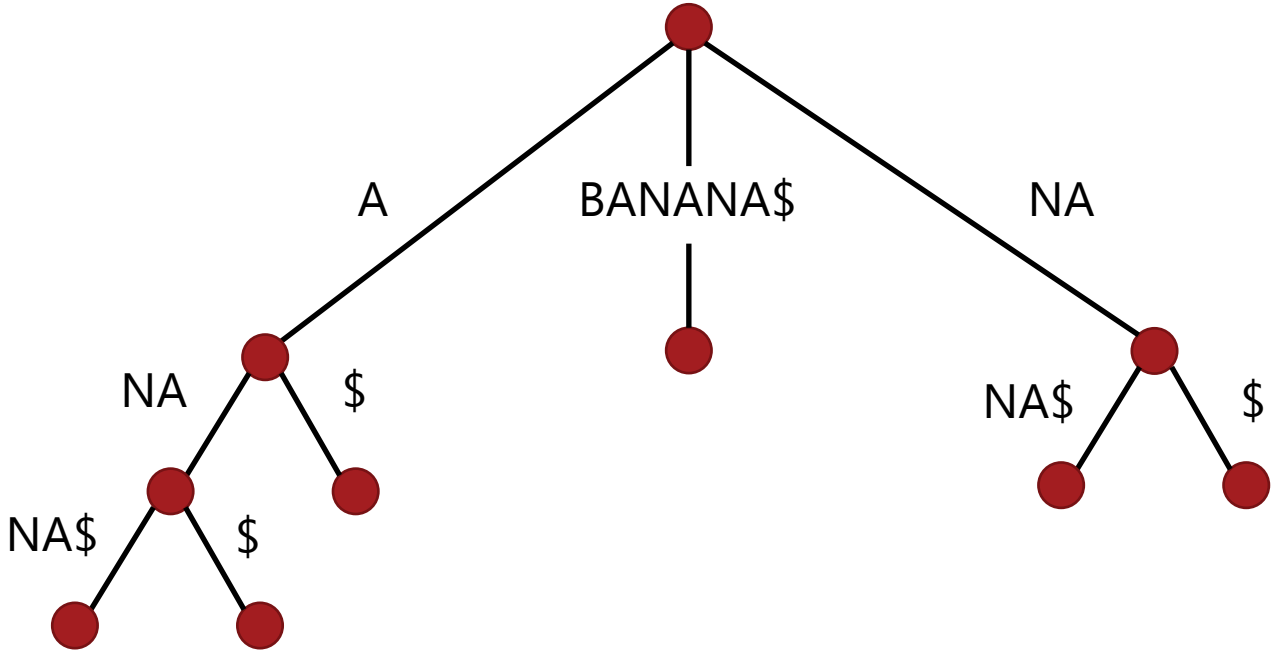
- Each internal nodes has at least 2 children
- No two edges starting at same node can have a string label beginning with the same character c ∈ Σ
- Each path from root to leaf spells out a suffix



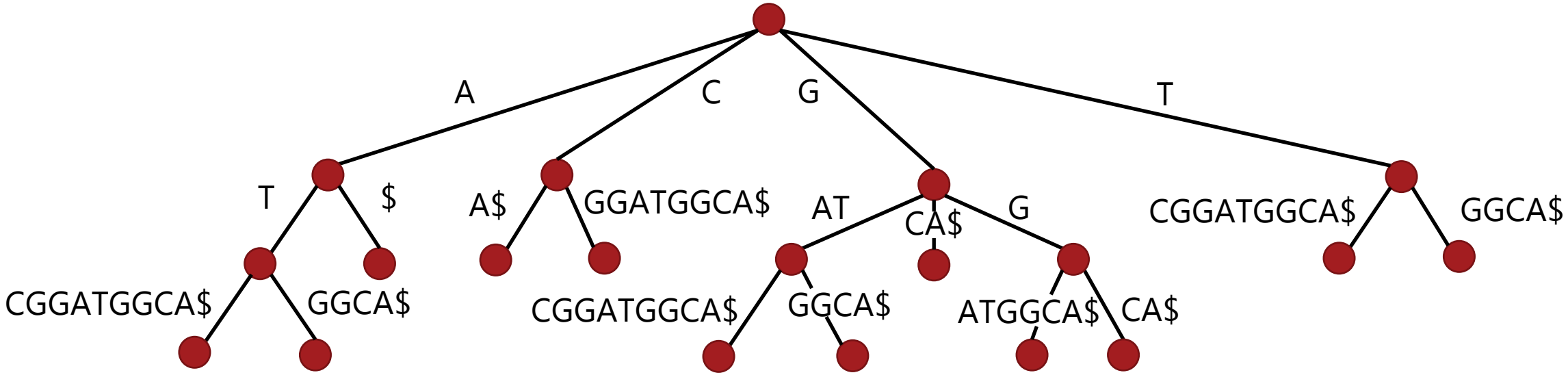
Suffix tree - BANANA$

BANANA$
ANANA$
NANA$
ANA$
NA$
A$





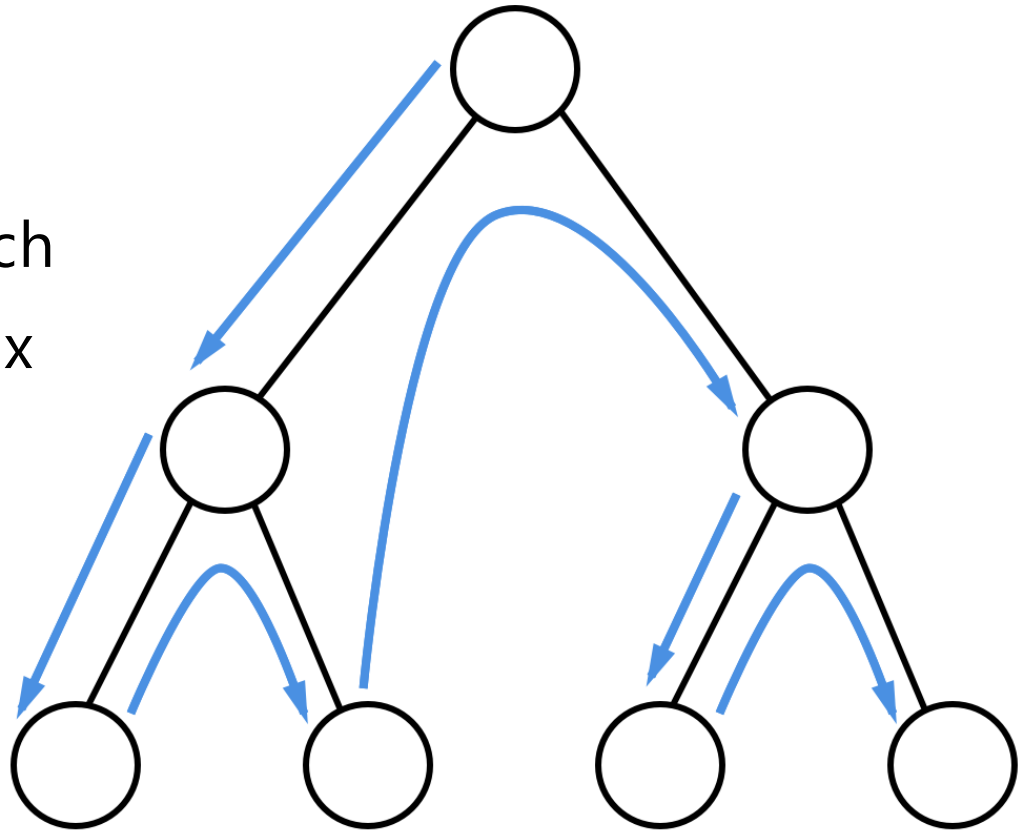
Suffix tree: T = GATCGGATGGCA$



- How do we check if string S is a substring of T?
  How do we check if string S is a suffix of T?
  How do we count number of times string S occurs as a substring of T?
- Similar to what we did in the suffix trie
- We can actually create an even more compressed data structure

Construction of Suffix array from Suffix trees using Pre-order Traverse tree depth-first

- Build suffix tree
- In alphabetically order through the left path first
- Traverse as far as possible along each branch
- Upon reaching a leaf we append suffix index to array



Suffix array

GATCGGATGGCA

- Suffix array of string S contains the offset in the string for which the lexicographical sorted suffixes appear
- Suf column
- Search for matches similar to looking up in a dictionary

- So suffix SA[1] = 6 = ATGGCA meaning pos 6 of original sequence

index	suf	corresponding suffix
0	1	ATCGGATGGCA
1	6	ATGGCA
2	11	A
3	10	CA
4	3	CGGATGGCA
5	0	GATCGGATGGCA
6	5	GATGGCA
7	9	GCA
8	4	GGATGGCA
9	8	GGCA
10	2	TCGGATGGCA
11	7	TGGCA

Big O - describes the limiting behavior of a function

- Suffix trie:
  - Creation: $O(m^2)$ time and space, m = length of full sequence
  - Querying: O(n), n = length of query
- Suffix tree:
  - Creation: O(m) time and space
  - Querying: O(n),
  - Bytes pr input character >15, memory for human genome 3×10^9 bases = 45 Gb
- Suffix array:
  - Creation: O(m) space, time = $O(m^2 \log(m))$
    Querying: is O(n log m)
  - Modern algorithms can also create SA in O(m) time using the suffixes are related through cyclic rotation
  - Bytes pr input character ~4 , for the human genome = 12 Gb

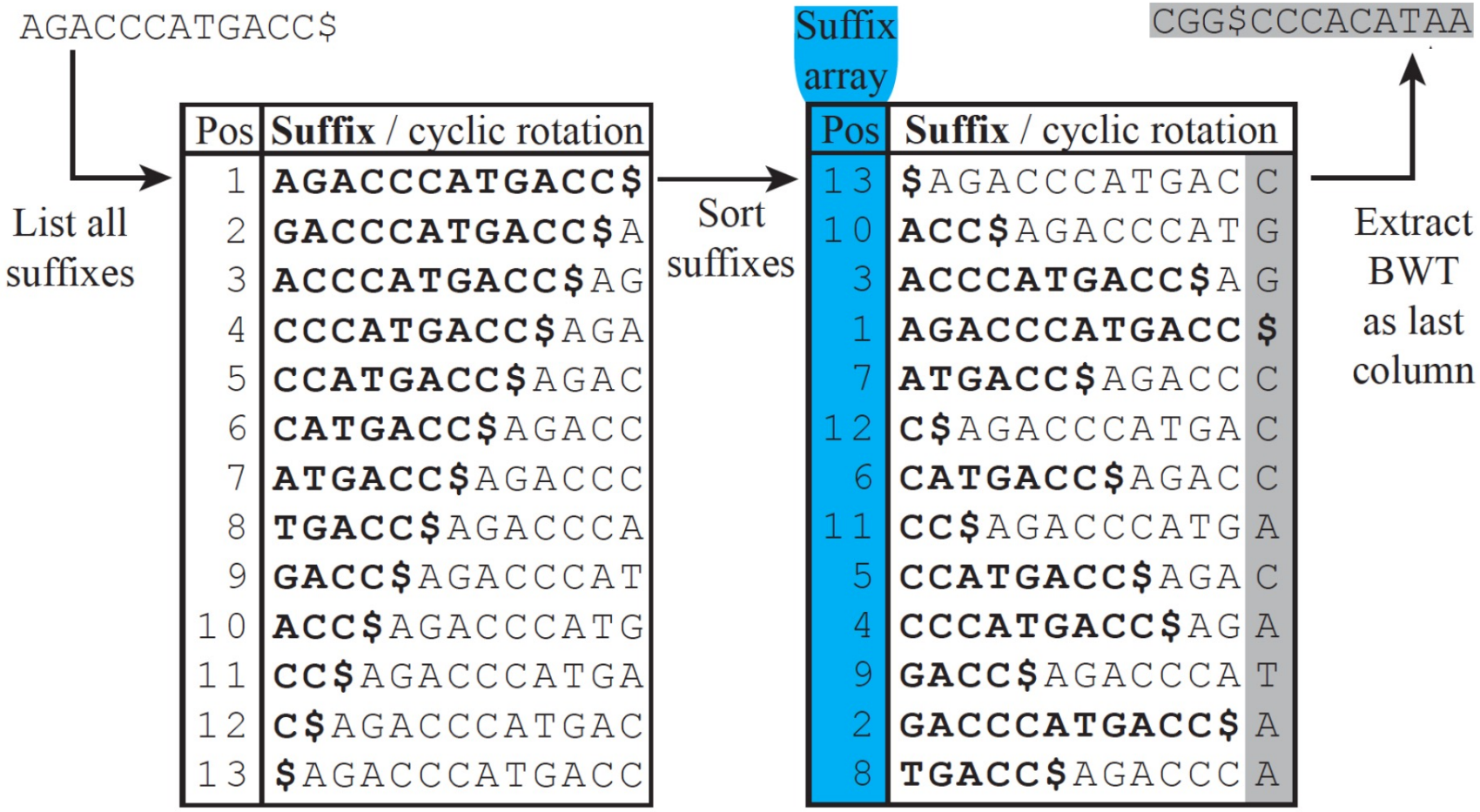
BURROWS-WHEELER TRANSFORM

Burrows-wheeler transform

- Originally created to as a way to rearrange a string into runs of similar characters
- If we have multiple substrings in a string, it will after BWT create a rearranged sequence which will have several places with repeated characters
- BANANA$ ⟶ BWT⟶ annb$aa
- This easier to compress and its reversible without storing additional data

Suffix transformation to Burrows-Wheeler-Transform

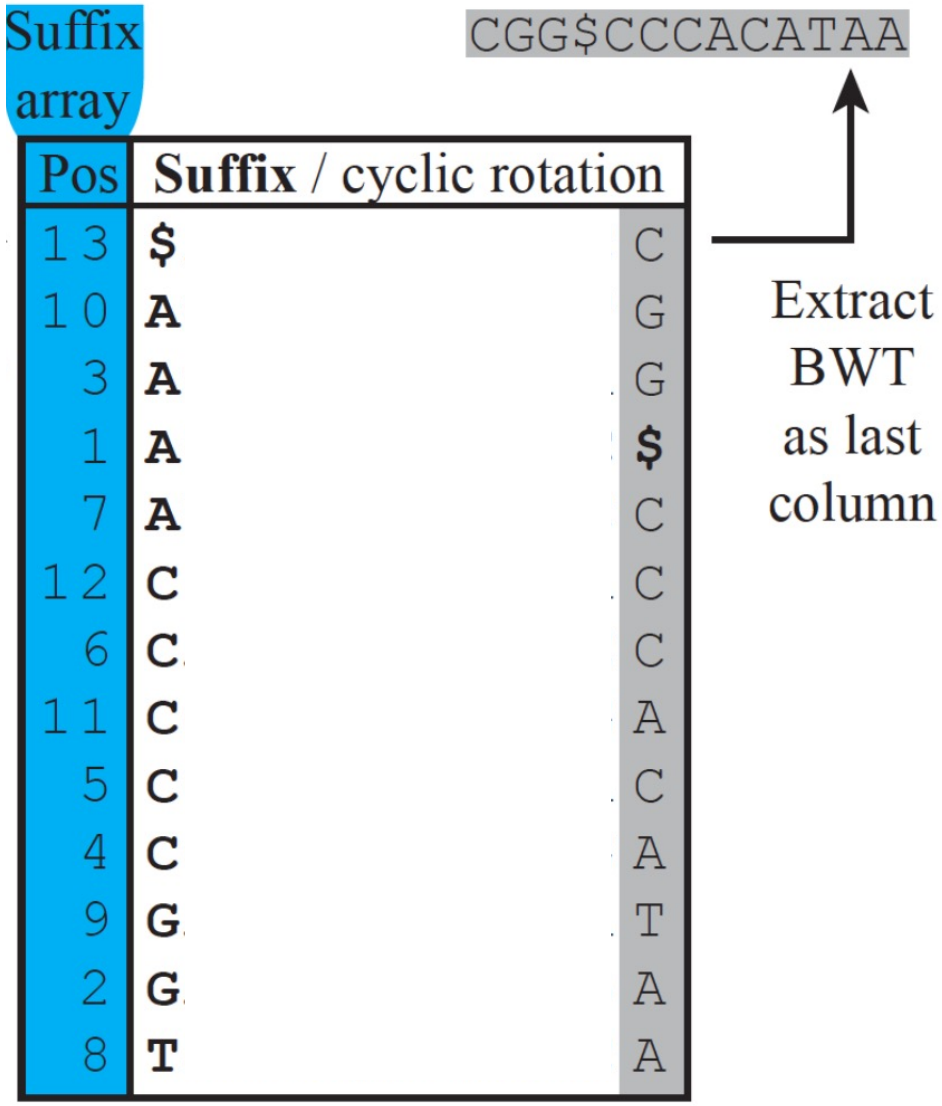
- Arrange the suffixes acoording to the suffix position index
- Creates the Burrows-Wheeler Matrix
- With the last column being the Burrows-Wheeler Transform



Burrows-Wheeler matrix

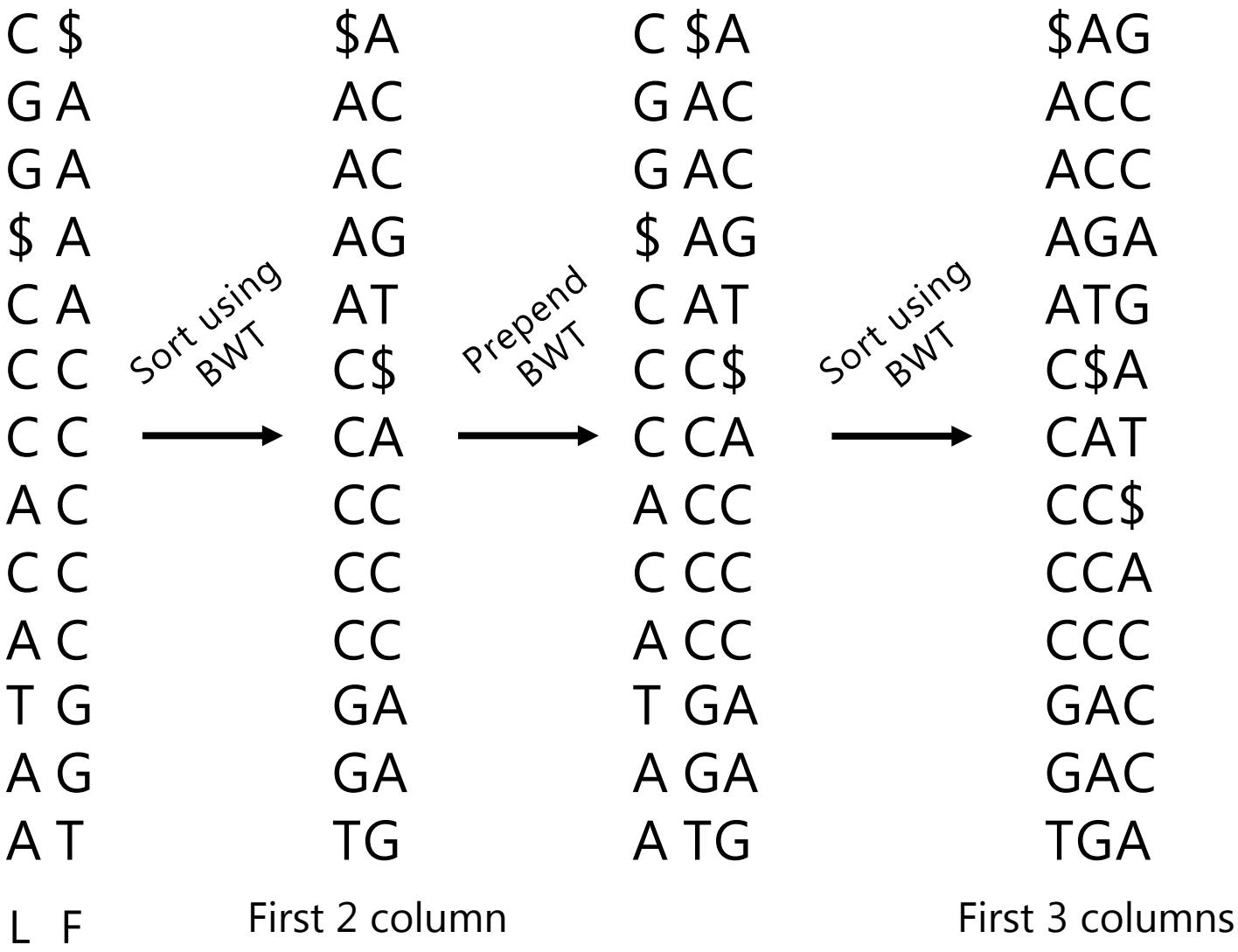
Applications - AGACCCATGACC$

- Compression
- How is it reversible?
  - Last column being the characters just before the first column
- How can it be used to create and index?
  - First to Last: FL - MAPPING
  - From an index i to an index j, such that F[j] = L[i]



Reversible – reconstructing AGACCCATGACC

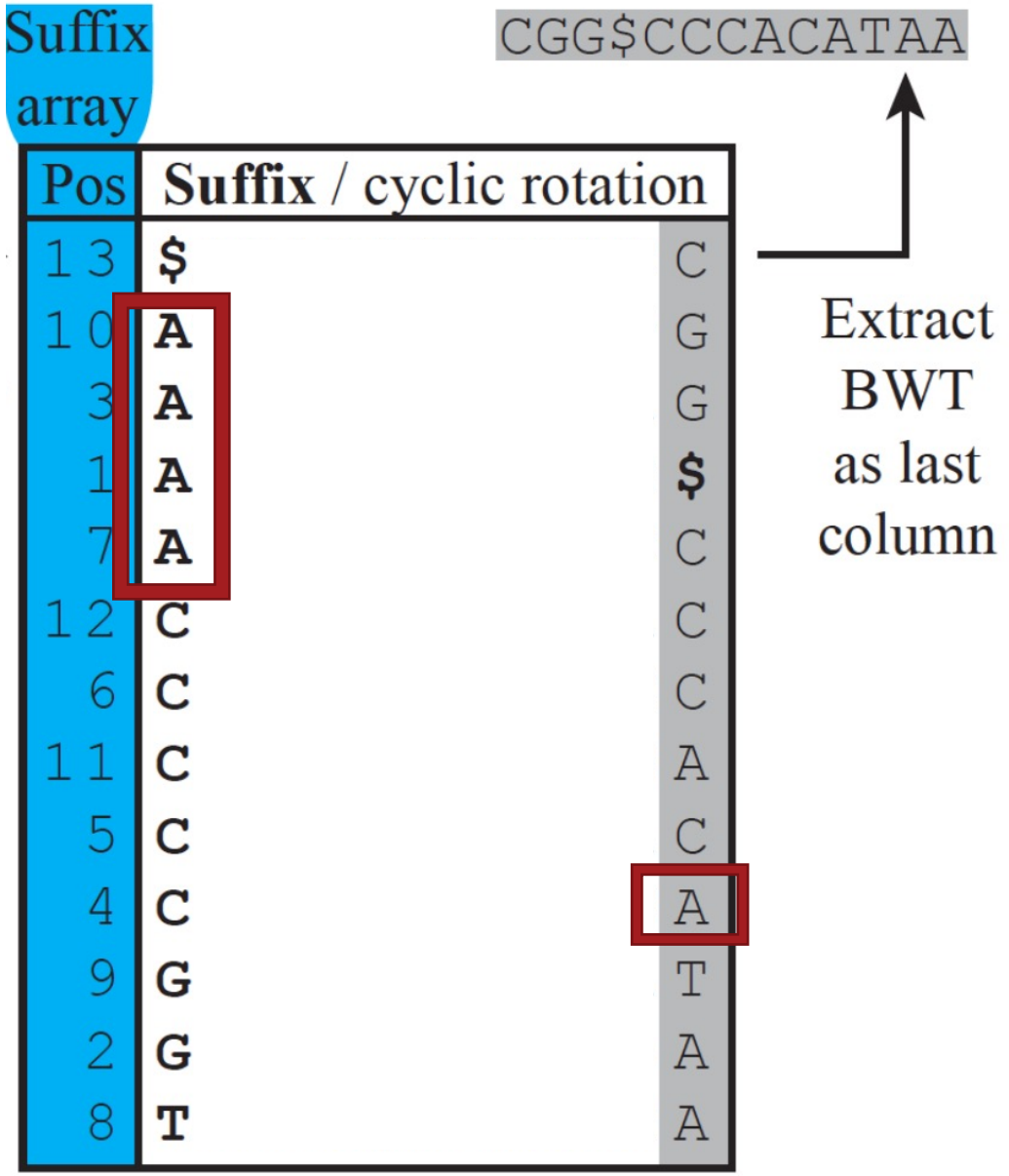
- Use L (BWT)
- Use F (SA)
- To reconstruct the Burrows-Wheeler Matrix



Create index - Last to First mapping

- Substring index based on BWT and similar to Suffix arrays
- Search for a position of each occurrence of a substring

- But where are the corresponding nucleotides in L and F?
- FM index we want to perform a left-first column mapping LF from an index i to an index j, such that F[j] = L[i]
  - Character Table, C[a]
  - Occurrence of Characters Function, Occ(a,i)



FM index

- C[a] is a table
  - contains the number of occurrences of alphabetically smaller characters than those stored in the suffix array.
  - Gives us the position of character in the suffix array
- Occ(a,i) is a function
  - the number of occurrences of character in the prefix of the last column until position i.
- LF(i) = C[L[i]] + Occ(L[i], i)
  - Suffix number corresponding to letter i in the bwt(i)

- The FM-index itself is a compression of L together with C and Occ, and information that maps the selected index in L to the positions in the original string.

C[a] table & Occ function - LF(i) = C[L[i]] + Occ(L[i], i)

- C[a] is a table

$AAAACCCCCGGT

a	$	A	C	G	T
C[a]	0	1	5	10	12

Occ(a,i) is a function

POS	1	2	3	4	5	6	7	8	9	10	11	12	13
BWT	C	G	G	$	C	C	C	A	C	A	T	A	A
$	0	0	0	1	1	1	1	1	1	1	1	1	1
A	0	0	0	0	0	0	0	1	1	2	2	3	4
C	1	1	1	1	2	3	4	4	5	5	5	5	5
G	0	1	2	2	2	2	2	2	2	2	2	2	2
T	0	0	0	0	0	0	0	0	0	0	1	1	1

C[a] table & Occ function - LF(i) = C[L[i]] + Occ(L[i], i)

L(6) = C

LF(6) = C[C] + Occ(C,6)

= 5 + 3 = 8

F(8) = C

L(10) = A

LF(10) = C[A] + Occ(A,10)

= 1 + 2 = 3

F(3) = A

Pos	Suffix / cyclic rotation	
13	$AGACCCATGAC	C
10	ACC$AGACCCAT	G
3	ACCCATGACC$A	G
1	AGACCCATGACC$	
7	ATGACC$AGACC	C
12	C$AGACCCATGA	C
6	CATGACC$AGAC	C
11	CC$AGACCCATG	A
5	CCATGACC$AGA	C
4	CCCATGACC$AG	A
9	GACC$AGACCCA	T
2	GACCCATGACC$	A
8	TGACC$AGACCC	A

Pos 3

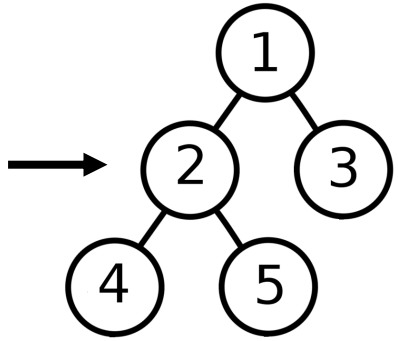
Pos 8

Pos 6

Pos 10

Summary

AGACCCATGACC$
GACCCATGACC$
ACCCATGACC$
CCCATGACC$
CCATGACC$
CATGACC$
ATGACC$
TGACC$
GACC$
ACC$
CC$
C$



Suffix array

Pos	
13	$
10	A
3	A
1	A
7	A
12	C
6	C
11	C
5	C
4	C
9	G
2	G
8	T

Suffix array

Pos	Suffix / cyclic rotation	
13	$AGACCCATGAC	C
10	ACC$AGACCCAT	G
3	ACCCATGACC$A	G
1	AGACCCATGACC$	
7	ATGACC$AGACC	C
12	C$AGACCCATGA	C
6	CATGACC$AGAC	C
11	CC$AGACCCATG	A
5	CCATGACC$AGA	C
4	CCCATGACC$AG	A
9	GACC$AGACCCA	T
2	GACCCATGACC$	A
8	TGACC$AGACCC	A

CGG$CCCACATAA

Extract BWT as last column

Pos	Suffix / cyclic rotation	
13	$	C
10	A	G
3	A	G
1	A	$
7	A	C
12	C	C
6	C	C
11	C	A
5	C	C
4	C	A
9	G	T
2	G	A
8	T	A

Sequence and we create suffixes

Create a

Suffix trie
&
suffix tree

Create a suffix array

Create Burrows-Wheeler matrix and BWT

Create FM index using LF

Big O

- Suffix trie:
  - Creation: $O(m^2)$ time and space, m = length of full sequence
  - Querying: O(n), n = length of query
- Suffix tree:
  - Creation: O(m) time and space, m = length of full sequence
  - Querying: O(n), n = length of query
  - Bytes pr input character >15
  - Human genome 45 Gb
- Suffix array:
  - Creation: O(m) space and time
  - Querying: O(n log m)
  - Bytes pr input character ~4
  - Human genome 12 Gb
- FM index
  - Creation: O(m) time and space
  - Reversing: O(m)
  - Querying: O(n)
  - Bytes pr input character ~0.5
  - Human genome = 1.5 Gb

ALIGNMENT

Alignment

- Assumptions:
  - Most reads originate from the genome in question
  - A read represents a substring of the genome apart from a small number of errors
  - Reads are short

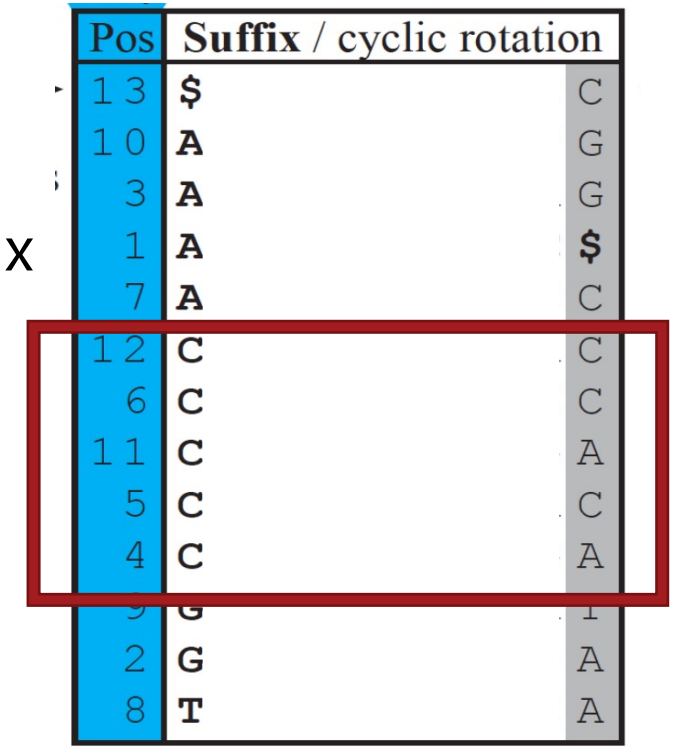
- BWA index the genome using the Burrows-Wheeler transform and then searches for matches of the reads as queries using FM index

Query using FM index

- Look for range of prefix match in the BWT(i)
- Start with shortes suffix and then match to longer suffix

- For query GACC
  - First find all **C** in F column
  - Look at the same rows in BWT for **C**
  - LF mapping to identify same **C**'s in column F to narrow the index

  - Repeat



Pos	Suffix / cyclic rotation	
13	$	C
10	A	G
3	A	G
1	A	$
7	A	C
12	C	C
6	C	C
11	C	A
5	C	C
4	C	A
2	G	A
8	T	A

Formulation of exact match using backwards search

Start with the initial suffix interval matching last letter in our query

$i = C[a] + 1$ to $j = C[a + 1]$

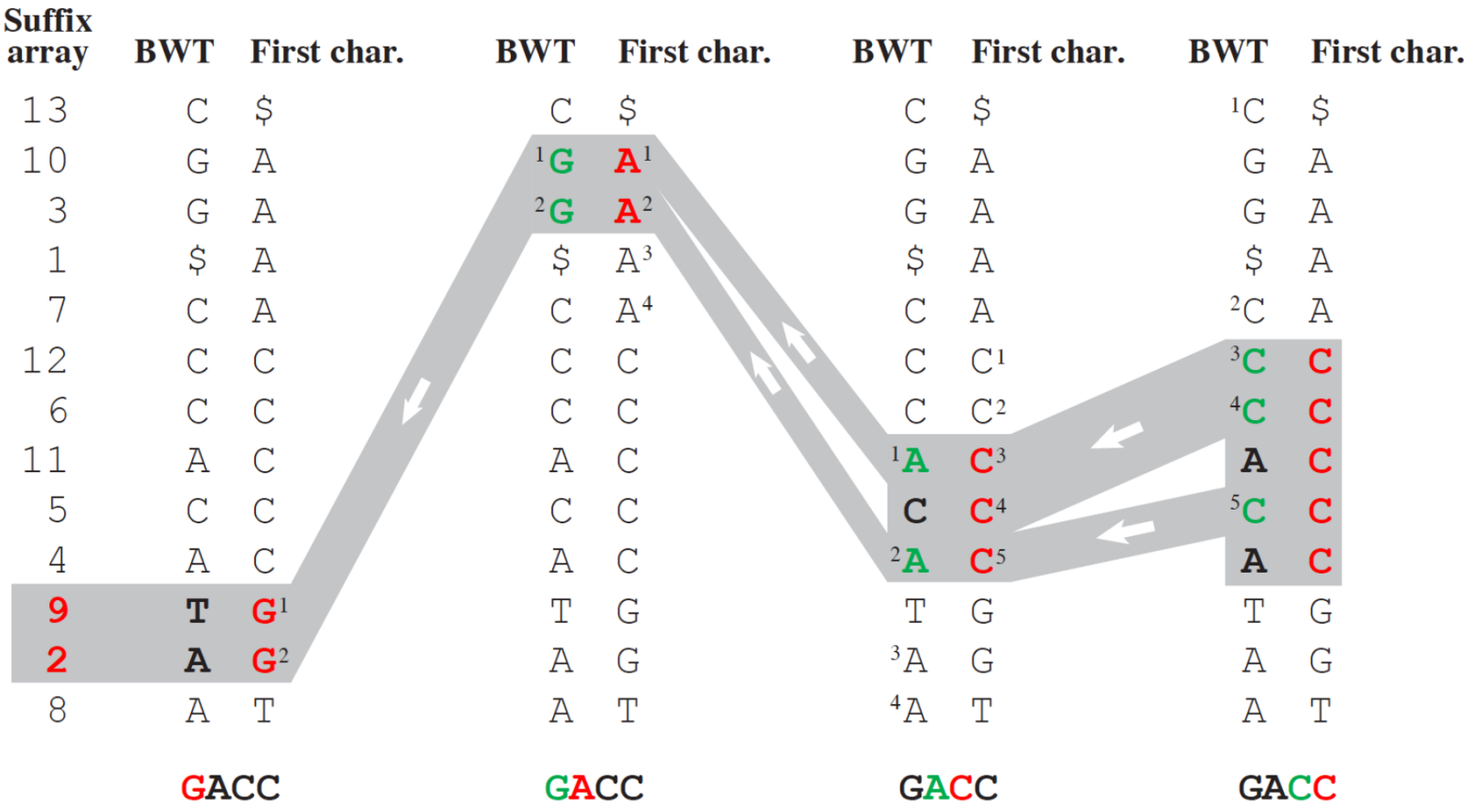
Then go one step backwards and recalculate next interval

$i = C[a] + Occ(a,i-1) + 1$

$j = C[a] + Occ(a,j)$

Continue until first letter in query

Aligning finding the exact match using L (BWT) and F column



i = C[C] + 1 = 5 + 1 = 6

j = C[C + 1] = C[G] = 10

i = C[C] + Occ(C,6-1) + 1

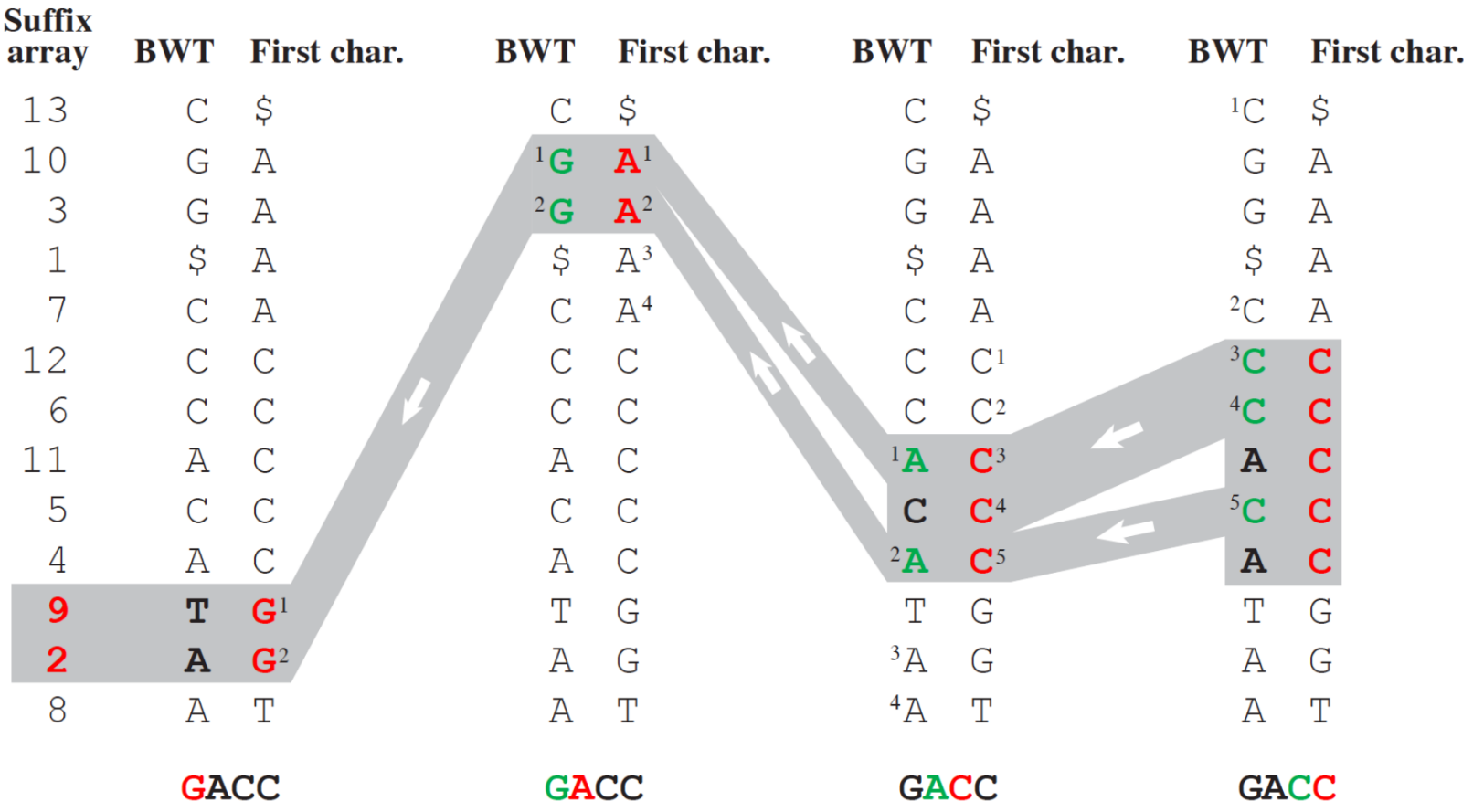
i = 5 + 2 + 1 = 8

j = C[C] + Occ(C,10)

j = 5 + 5 = 10

Aligning finding the exact match using L (BWT) and F column

AGACCCATGACC



Pos	Suffix / cyclic rotation
13	$AGACCCATGAC C
10	ACC$AGACCCAT G
3	ACCCATGACC$A G
1	AGACCCATGACC $
7	ATGACC$AGACC C
12	C$AGACCCATGA C
6	CATGACC$AGAC C
11	CC$AGACCCATG A
5	CCATGACC$AGA C
4	CCCATGACC$AG A
9	GACC$AGACCCA T
2	GACCCATGACC$ A
8	TGACC$AGACCC A

BWA article – Exact match

- Mentioned in the paper they create a suffix array using prefix trie
  - which is identical to a suffix trie if we reverse the string X
- W is a substring of X
- SA interval of to search for a match of W $[\underline{R}(W), \overline{R}(W)]$
- Exacth matches using Backtracking as described above

$$\underline{R}(aW) = C(a) + O(a, \underline{R}(W) - 1) + 1$$

$$\overline{R}(aW) = C(a) + O(a, \overline{R}(W))$$

BWA article – Inexact match (in theory)

Algorithm InexactSearch(W, z)

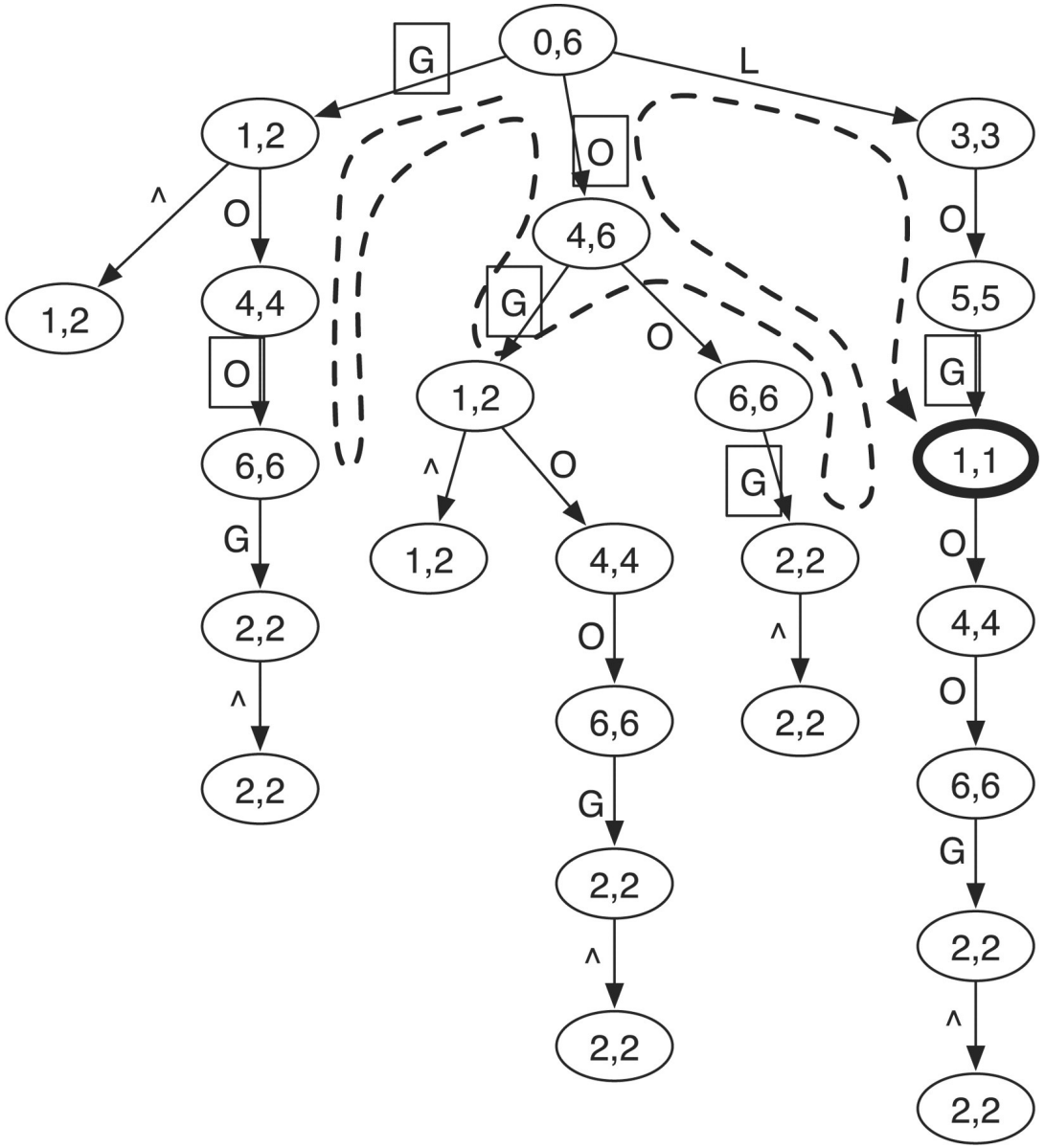
1: CalculateD(W)
2: **return** InexRecur($W, |W| - 1, z, 1, |X| - 1$)

- InexRecur(W , i, z, k, l) returns the SA intervals of substrings in X that match our query string W with no more than z differences
  - W : query
  - i: Search for matches to W [0..i]
  - z max number of mismatches
  - k, l: On the condition that the suffix Wi+1 matches interval [k..l]
- CalculateD(W) generates a is the lower bound of the number of differences in a substring of our query W [0..i] to the best match in the substrings of X

BWA article example (in theory)

- Reference (X) = GOOGOL
- Query (W) = LOL
- Search for mismatches in the prefix until lower bound is found of X
- Given the suffix aligns

1. G branch GOO -> stop search
2. O branch OG & OOG -> stop search
3. L branch LOG -> stop search



BWA ALN, BWA MEM

BWA aln:

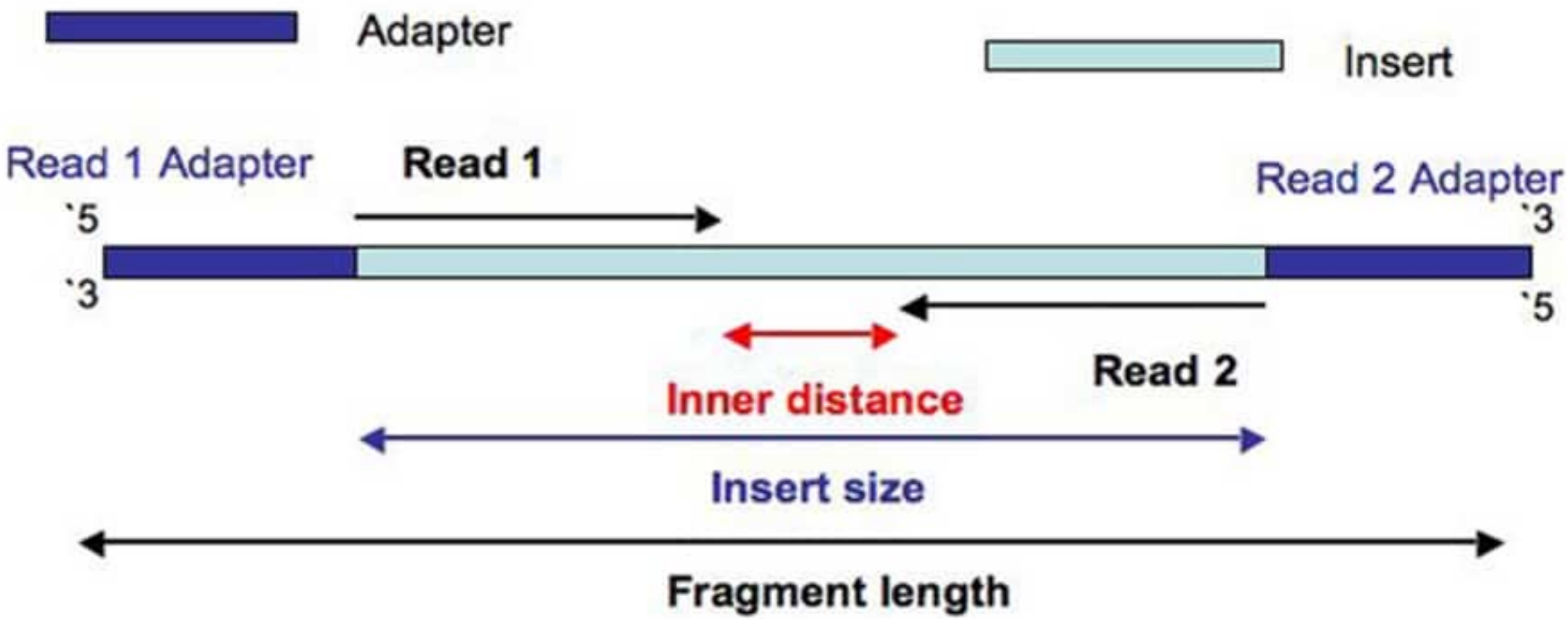
- Seeds the read with the first 32 nucleotides to find a potential match using BWT
- Then extends the alignment within the region

BWA mem:

- Multiple short seeds across the read
- Clustering seeds
- Seed extension using Local sequence alignment

EXAMPLES, SEQUENCE READ, SAMTOOLS

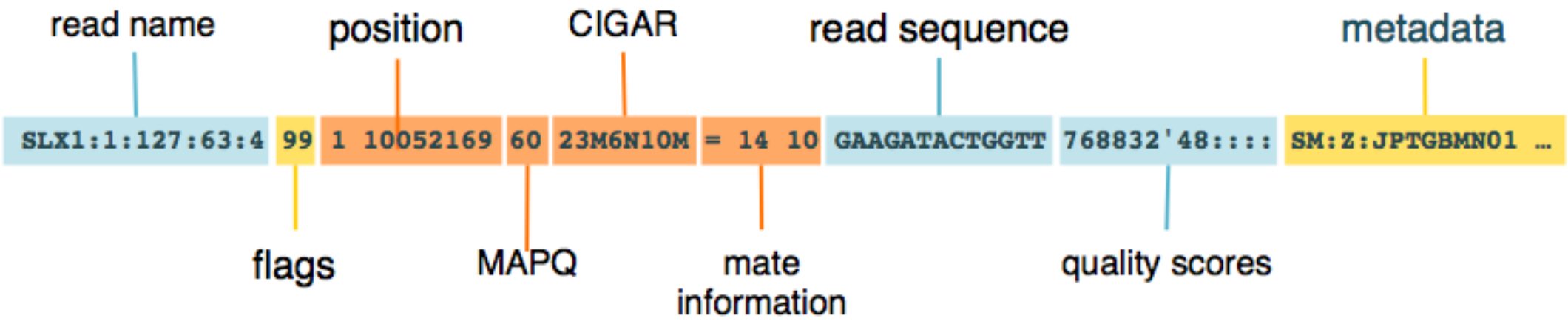
Terminology: Fragment vs Sequence reads, paired-end reads



Sequence Alignment Map Format – SAM files

- Text file format with alignment information for several aligned sequences
- BAM and CRAM files are compressed forms of SAM

HEADER containing metadata (sequence dictionary, read group definitions etc)
RECORDS containing structured read information (1 line per read record)



samtools view -H Data_pe.bam

```
@SQ SN:chr1 LN:249250621
@SQ SN:chr2 LN:243199373
@SQ SN:chr3 LN:198022430
@SQ SN:chr4 LN:191154276
@SQ SN:chr5 LN:180915260
.
.
.
@SQ SN:chrUn_gl000248 LN:39786
@SQ SN:chrUn_gl000249 LN:38502
@RG ID:Data_trim.fastq.gz SM:Data
LB:Data_pe PL:ILLUMINA_HISEQ_4000
@PG ID:bwa PN:bwa VN:0.7.17-r1198-dirty
CL:bwa mem -t 40 -R
@RG\tID:Data_trim.fastq.gz\tSM:Data\tLB:Da
ta_pe\tPL:ILLUMINA_HISEQ_4000 hg19.fa
Data_1_trim.fastq.gz Data_2_trim.fastq.gz
@PG ID:samtools PN:samtools PP:bwa VN:1.10
CL:samtools view -H Data_pe.bam
```

- SQ tag: Reference sequence dictionary'
  - SN: reference sequence name
  - LN: Length of reference sequence
- RG tag: Read group
  - ID: id of alignment record
  - SM: SAMPLE
  - LB: library
  - PL: platform used
- PG tag: program
  - ID: Program identifier
  - PN: Program Name
  - PP: Previous PG tag
  - VN: Program Version
  - CL: Command

CIGAR strings and SAM flags – https://broadinstitute.github.io/picard/explain-flags.html

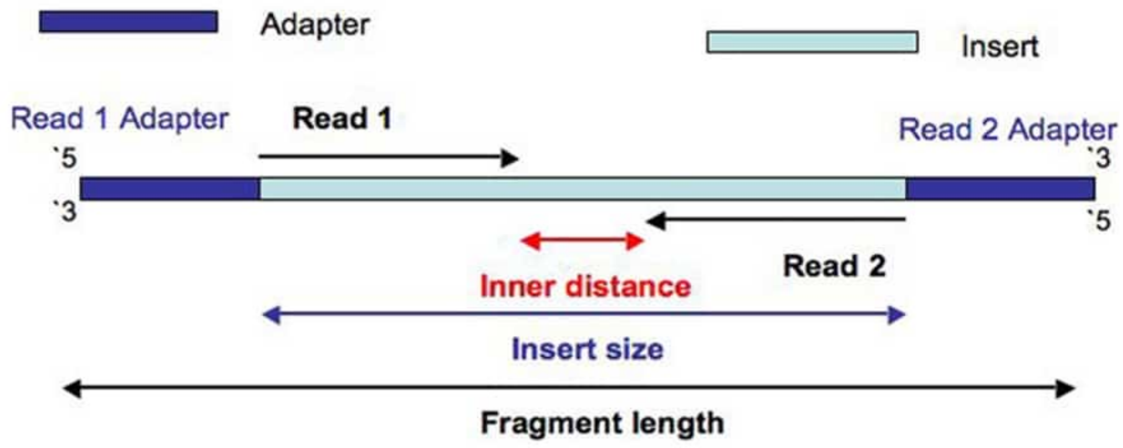
Decimal	Binary	Exp.	Meaning
1	1	2^0	This is a paired read
2	10	2^1	This read is part of a pair that aligned properly*
4	100	2^2	This read was not aligned
8	1000	2^3	This read is part of a pair and its mate was not aligned
16	10000	2^4	This read aligned in the reverse direction**
32	100000	2^5	This read is part of a pair and its mate aligned in the reverse direction**
64	1000000	2^6	This read is the first in the pair (read 1)
128	10000000	2^7	This read is the second in pair (read 2)
256	100000000	2^8	The given alignment is a secondary alignment***
512	1000000000	2^9	Read failed quality check (such as Illumina quality filtering)
1024	10000000000	2^{10}	Read was flagged as a duplicate (such as a PCR duplicate)
2048	100000000000	2^{11}	Supplementary alignment (Exact meaning varies by aligner)

Op	BAM	Description
M	0	alignment match (can be a sequence match or mismatch)
I	1	insertion to the reference
D	2	deletion from the reference
N	3	skipped region from the reference
S	4	soft clipping (clipped sequences present in SEQ)
H	5	hard clipping (clipped sequences NOT present in SEQ)
P	6	padding (silent deletion from padded reference)
=	7	sequence match
X	8	sequence mismatch

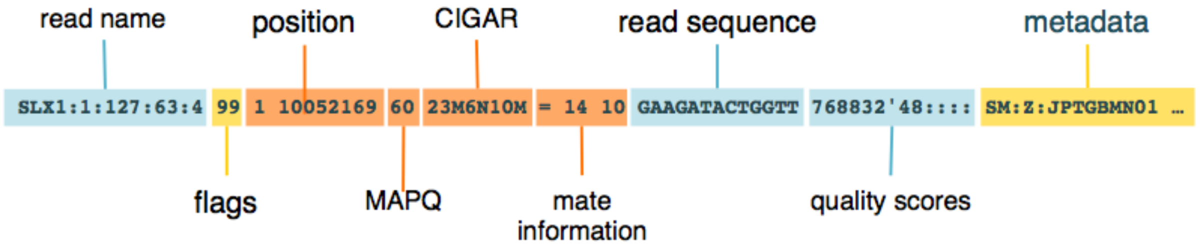
Filter your aligned reads

```
samtools view -b -F 4 -q 30 Data_pe.bam -o Data_pe_filter.bam
```

samtools view -b -F 4 -q 30 Data_pe.bam -o Data_pe_filter.bam



HEADER containing metadata (sequence dictionary, read group definitions etc)
RECORDS containing structured read information (1 line per read record)



- **K00171:751:HW5LWBBXX:8:1101:15767:11055** 99 chr3 191218878 60 101M = 191218953 176 GCTAATGGCAGAATCAAGGTGTTGCTCAATAGCCACTGTAAAGGGTTCTCTCTACCACACCATGTGGCTTTCTATTATTTATATGCATATGTGTTTGTTAT AAFFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ NM:i:2 MD:Z:9G33A57 MC:Z:101M AS:i:91 XS:i:20 RG:Z:Data_trim.fastq.gz

- **K00171:751:HW5LWBBXX:8:1101:15767:11055** 147 chr3 191218953 60 101M = 191218878 -176 TATTTATATGCATATGTGTTTGTTATAGGTTTACATACATTTGAAATGCATACACTATATATACAATGTATACATACATATTATGTGTGTGCATATATGTG FJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJFJJJJJJJJJJJJFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJFFFAA NM:i:1 MD:Z:92G8 MC:Z:101M AS:i:96 XS:i:21 RG:Z:Data_trim.fastq.gz

SAM Flag: 99 Explain

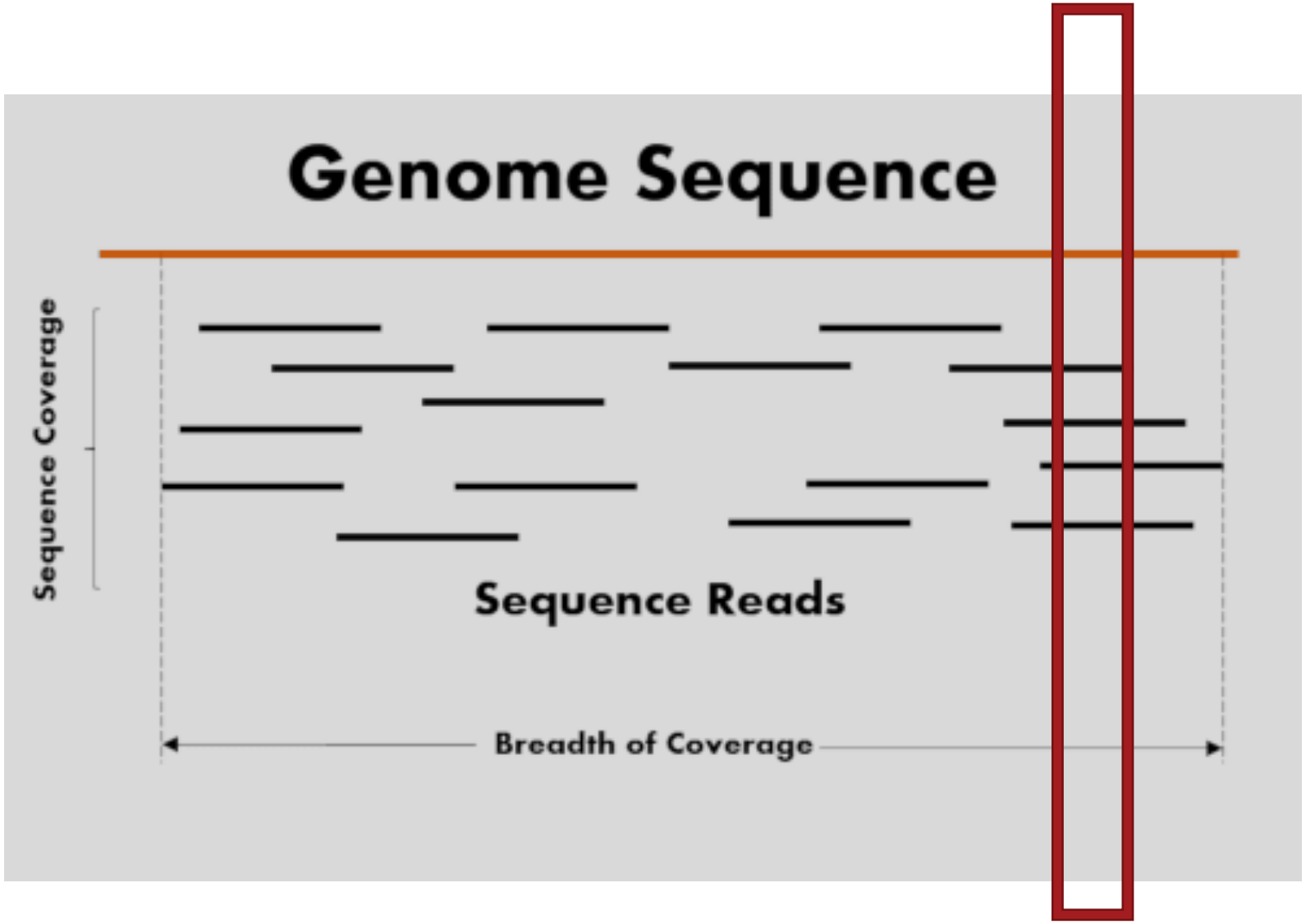
Summary:
read paired (0x1)
read mapped in proper pair (0x2)
mate reverse strand (0x20)
first in pair (0x40)

SAM Flag: 147 Explain

Summary:
read paired (0x1)
read mapped in proper pair (0x2)
read reverse strand (0x10)
second in pair (0x80)

Sequencing Coverage and Depth are used interchangeably

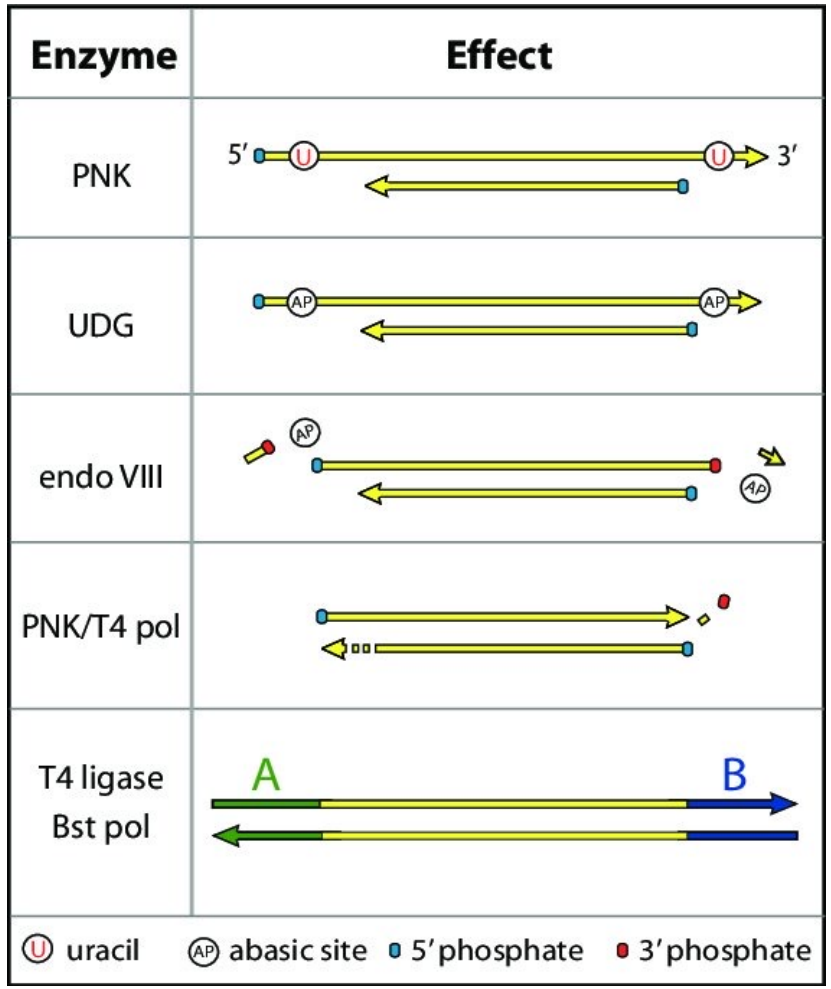
- Number of times a nucleotide is sequenced.
- Greater depth gives greater confidence by distuingishing sequencing errors from variant calling
- 100 Mbp Genome and you have sequenced 5 M reads of 100 bp size gives a genome level coverage of 5x
- **Depth of coverage**
- **Breadth of coverage**: percentage of genome is covered by sequence reads with a certain depth (95%)



Samtools commands

- Samtools view
  - Header: samtools view -H Data_pe.bam
  - Filtering: samtools view -b -F 4 -q 30 Data_pe.bam -o Data_pe_filter.bam
  - File conversion: samtools view -C -T hg19.fa Data_pe.bam -o Data_pe.cram
- Samtools sort
  - samtools sort -@ 10 -m 2G Data_pe.bam -o Data_pe_sort.bam
- Samtools index
  - samtools index Data_pe_sort.bam
- Samtools faidx
  - samtools faidx hg19.fa chr1:100000-100100
    >chr1:100000-100100
    cactaagcacacagagaataatgtctagaatctgagtgccatgttatcaaattgtactgagactcttgcagtcacacagg
    ctgacatgtaagcatcgccat

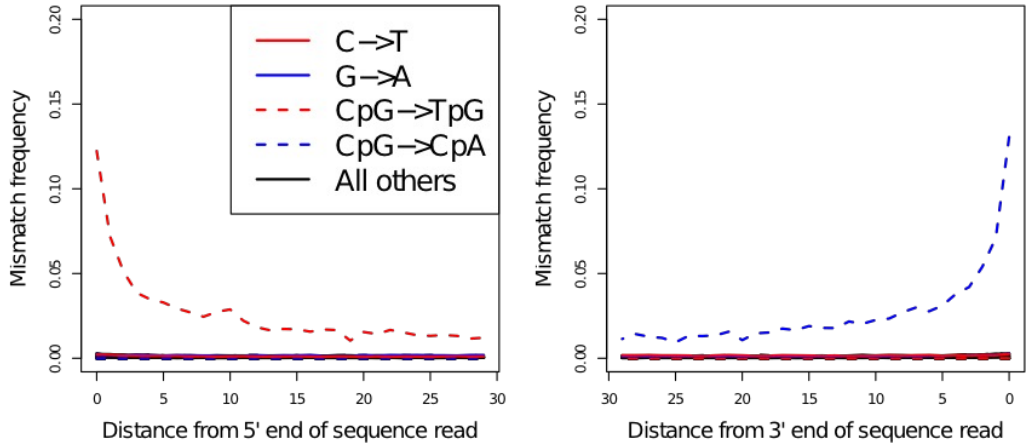
Why is these analysis important for ancient DNA?

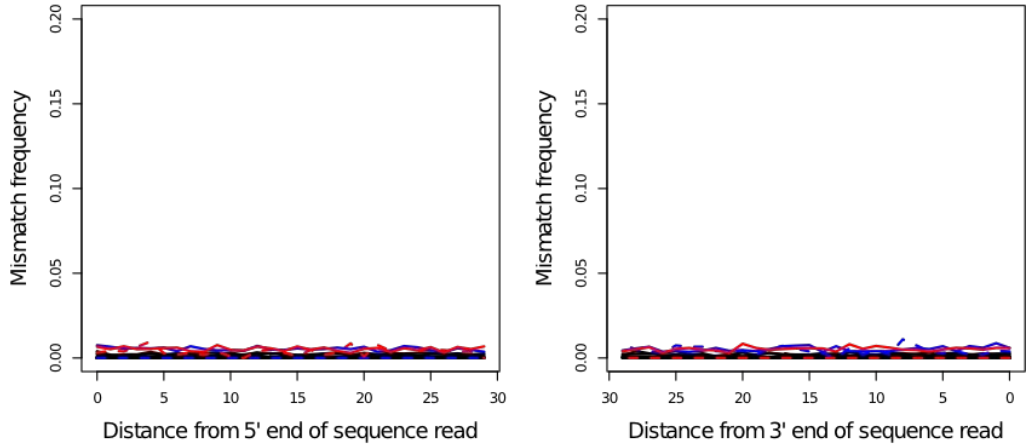


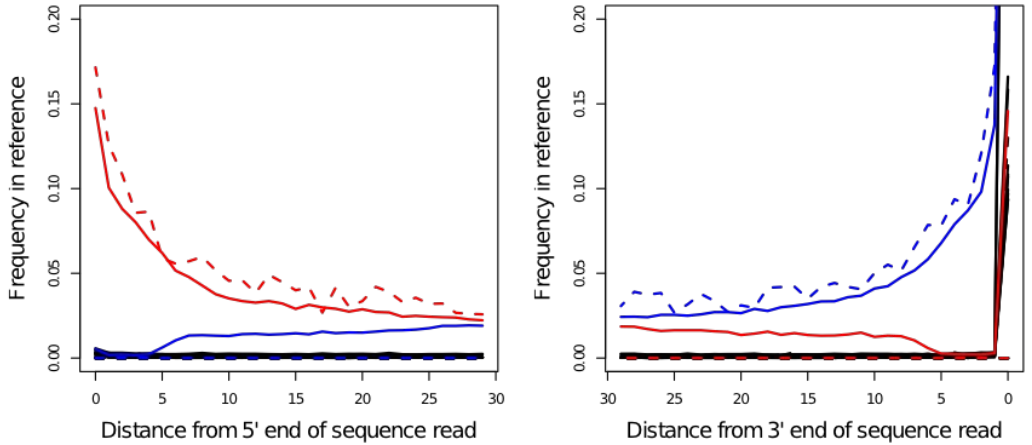
Nuclear DNA
USER-treated

mtDNA
USER-treated

No treatment

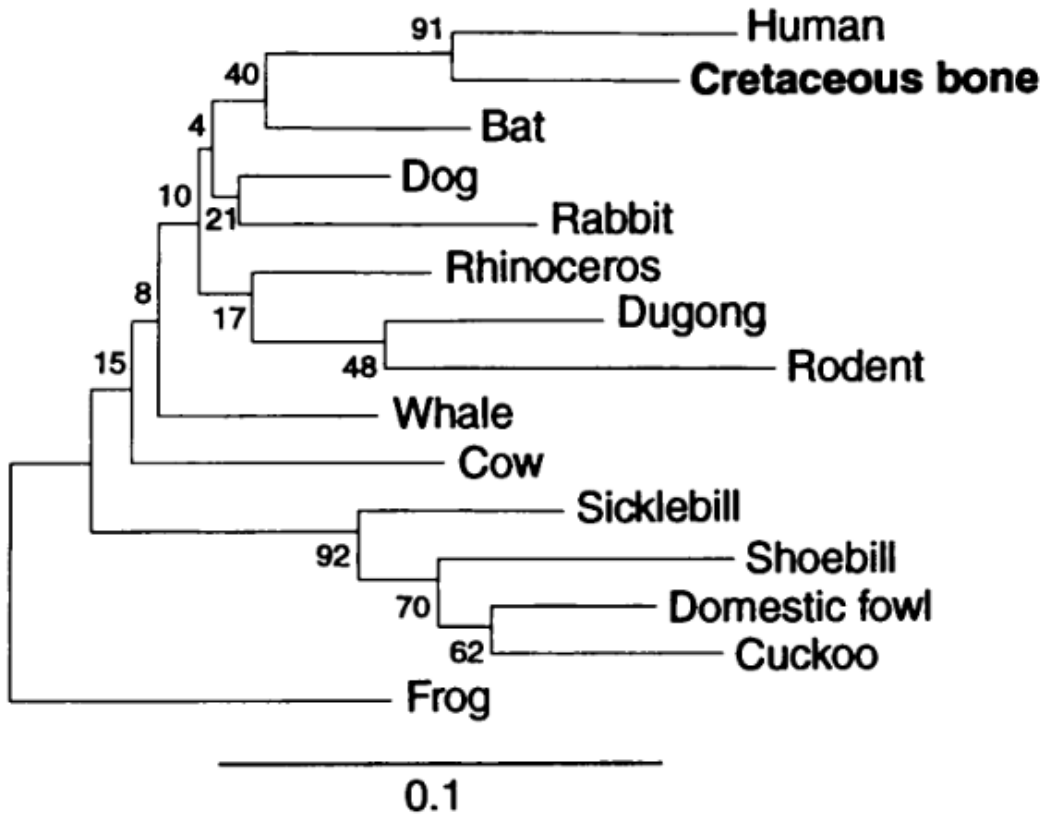




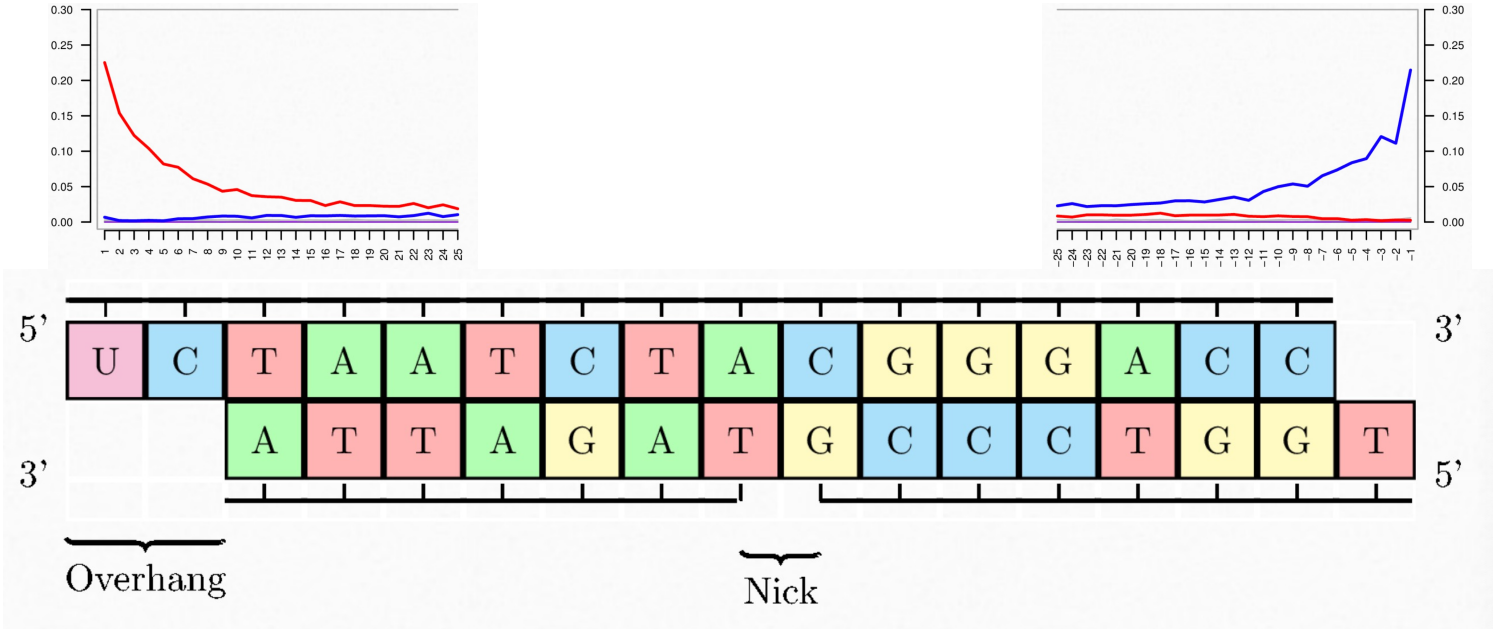


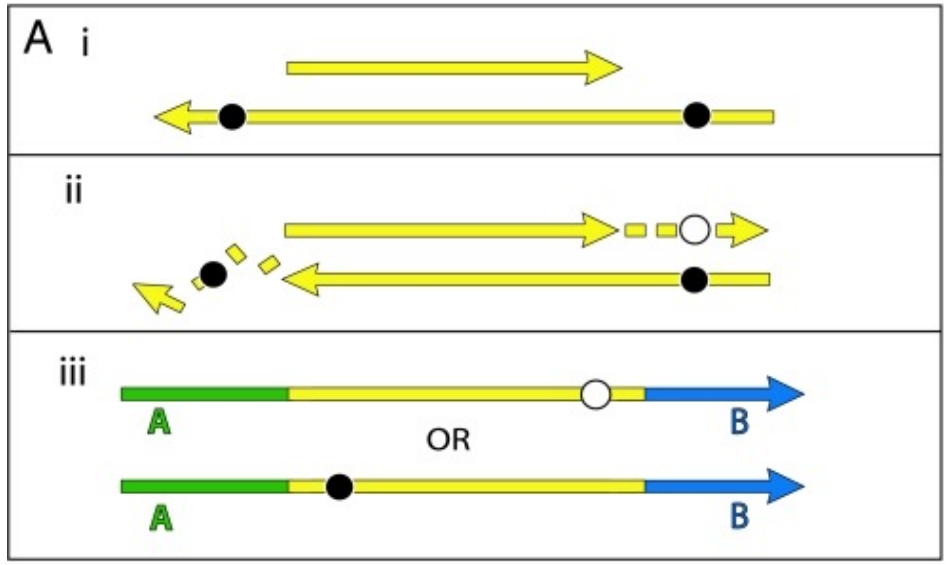
Damage patterns - Authentication

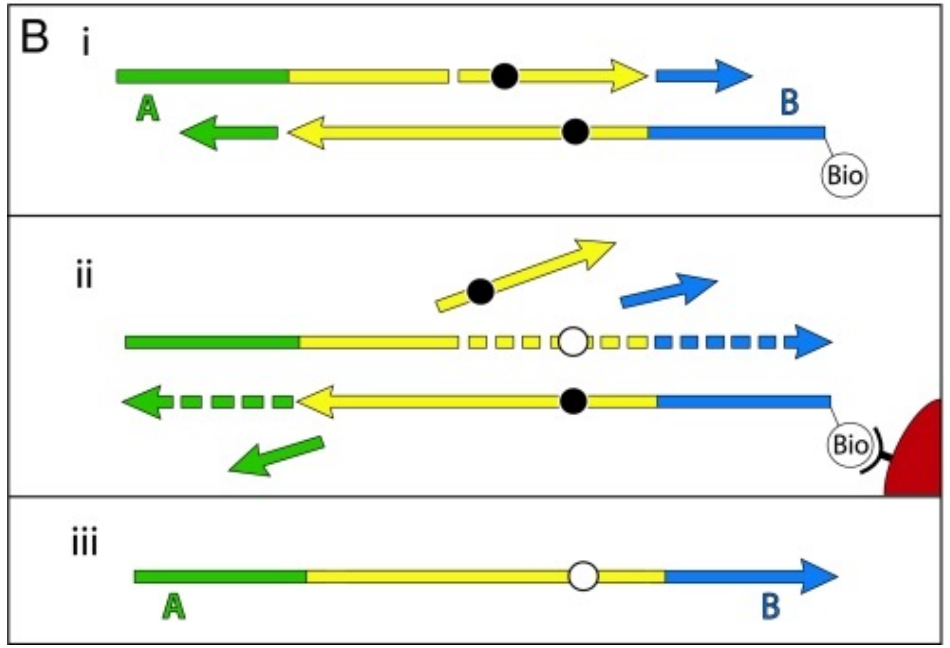
- 80 My Dinosaur DNA from cytochrome B region
- 120 – 135 My old weevil of 315 and 226 basepair fragments
- 250 My bacillus species from salt-crystal
- None of these have been replicated or validated



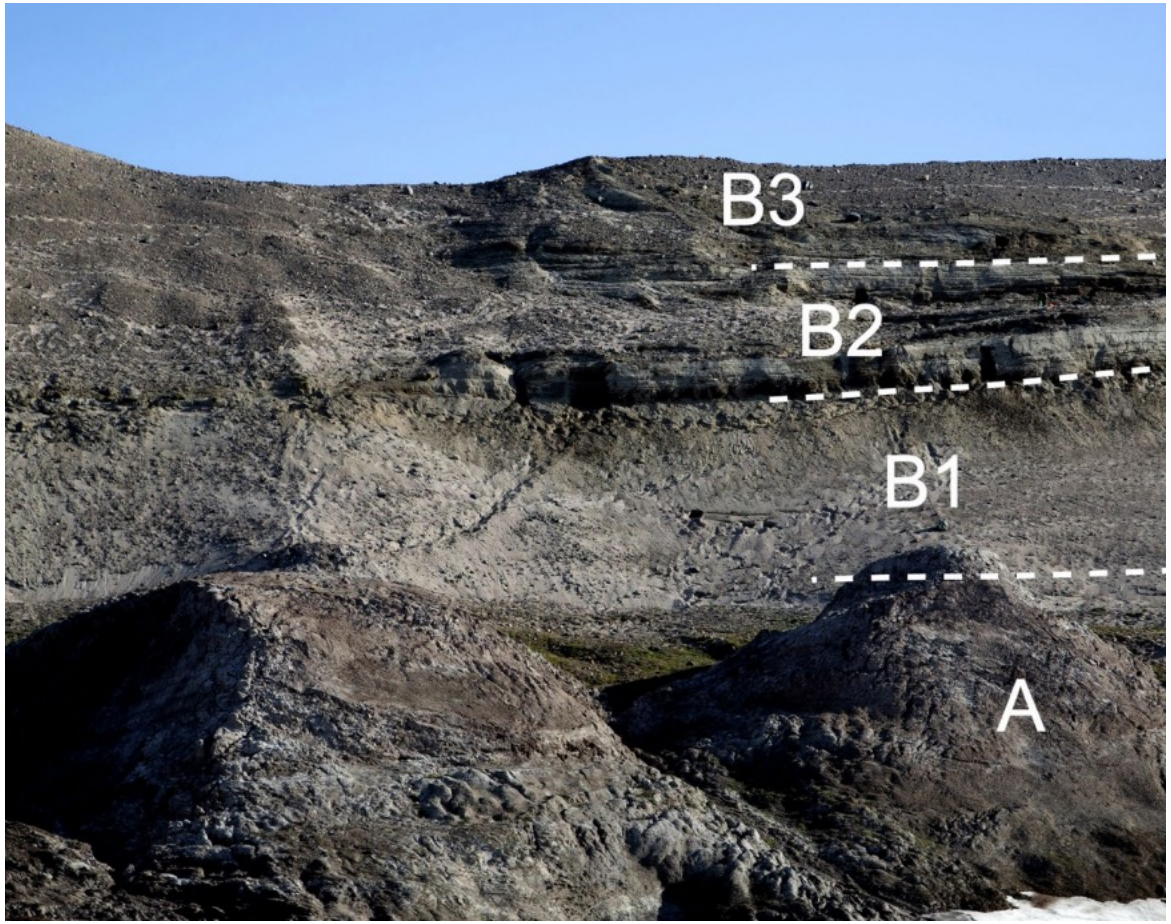
Modelling the damage patterns

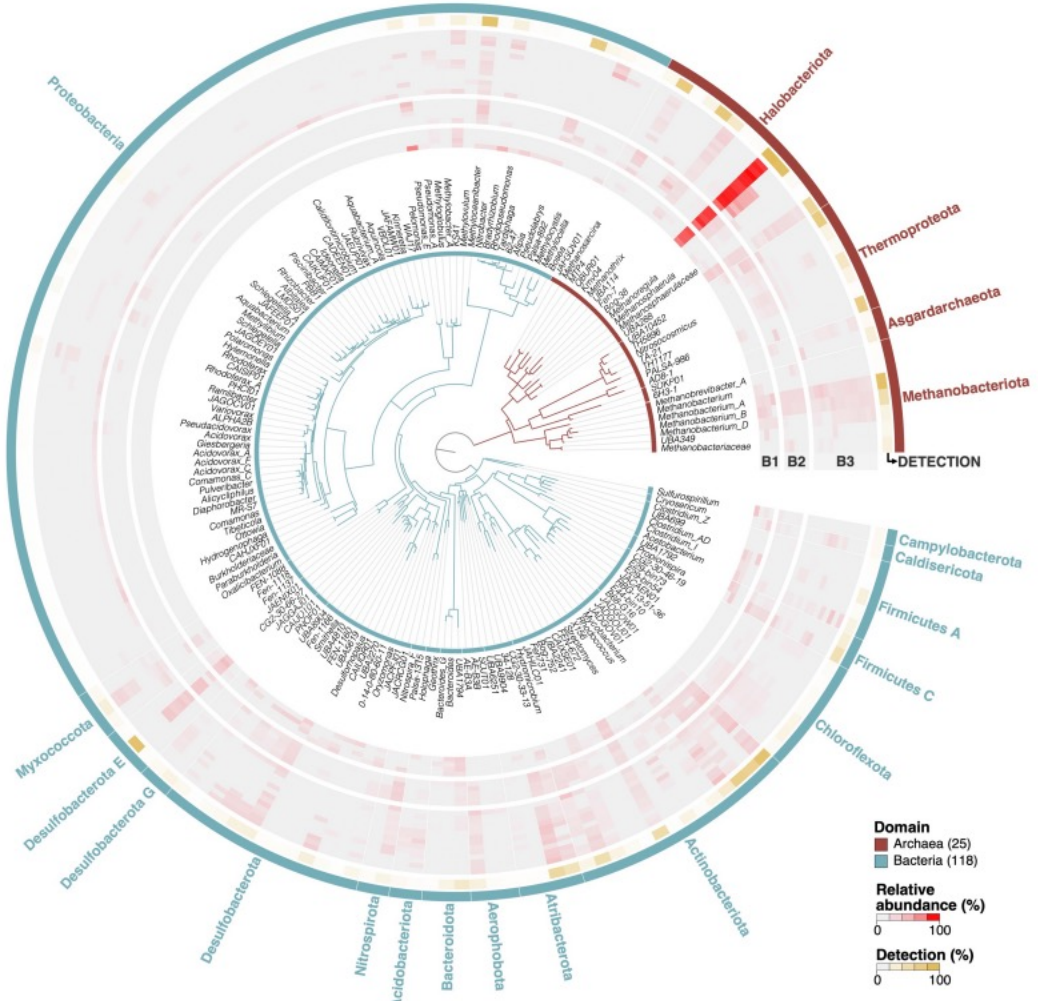






Damage patterns and metaGenomics





Damage patterns and metaGenomics