# Bioinformatics Algorithms
# COS-BIOL-530/630
# Lecture13

| Days & Times | Room | Meeting Dates |
|---|---|---|
| Tu 2:00PM - 3:50PM | Thomas Gosnell Hall (GOS)-2178 | 01/13/2025 - 04/28/2025 |
| Th 2:00PM - 3:50PM | Thomas Gosnell Hall (GOS)-2178 | 01/13/2025 - 04/28/2025 |

Instructor:
Fernando Rodriguez
email: frvsbi@rit.edu
Office: Orange Hall 1311

# RNA-seq
# - Lecture13-

**Quiz 10**: Lectures 12 and 13. Available April 23rd
(Due on April 27th 5 pm)

**Lab 12 Assignments**:
- Discussion12
- Activity12

**Final Project:**
- Draft (due on April 24)
- Report (due on May 1st 11:59 pm)

**No Class on Thursday** April 24th

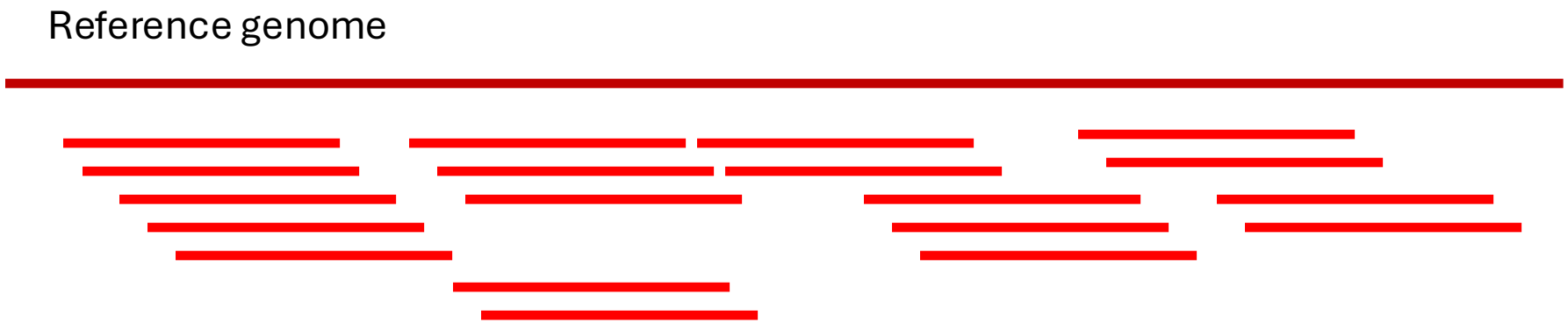
**SRATE evaluation** period will be ending soon (4/29)!
rit.smartevals.com

# RNA-seq
# - Lecture13-

Topics:

- The short-read mapping (alignment) problem
- The Burrows-Wheeler Transform (BWT)
- RNA sequencing

# Short read mapping




HTS technologies produce million of reads = a short substring of the genome

If we already have one reference genome:

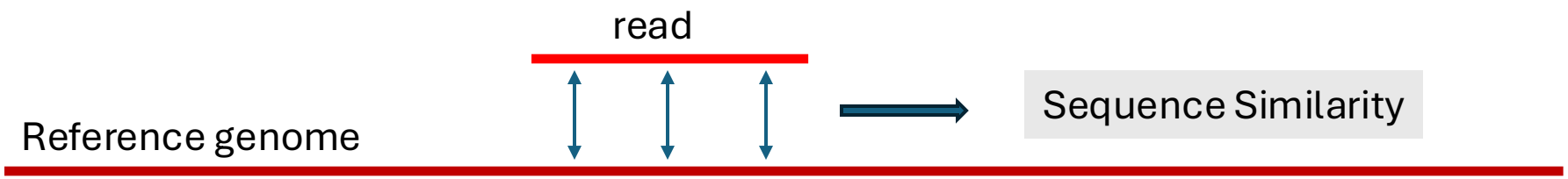
- Map genomic read to the reference (**re-sequencing**)
  - Nucleotide Polymorphism
  - Find structural variants
  - Indels

- Map RNA (coding and non-coding) to reference

- Problem: Million of string searches in a long string

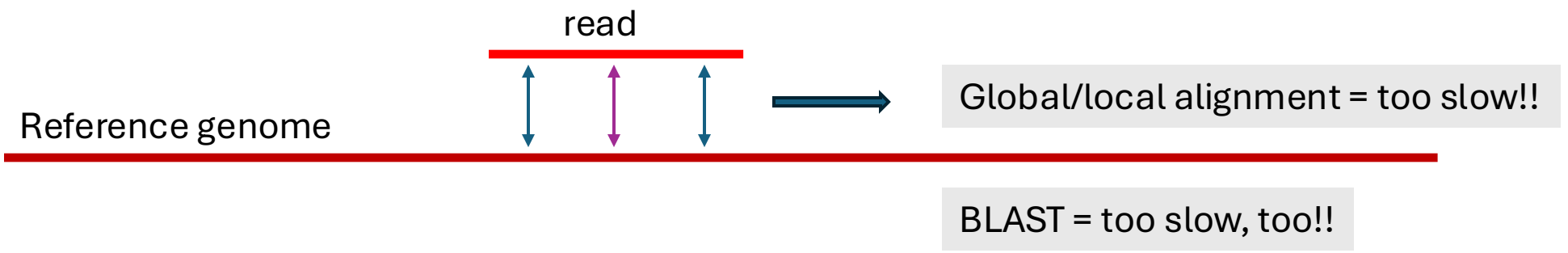
# Short read mapping: the problem

- First, genomic DNA is sheared into fragments that can be size-selected to obtain a template library of uniformly short DNA fragments.
- During sequencing, every molecule is cyclically extended by a single base, and all clusters are read in parallel before the cycle is repeated (Illumina).
- The first step in the analysis of NGS data is to determine where in the genome each of the short reads originated.



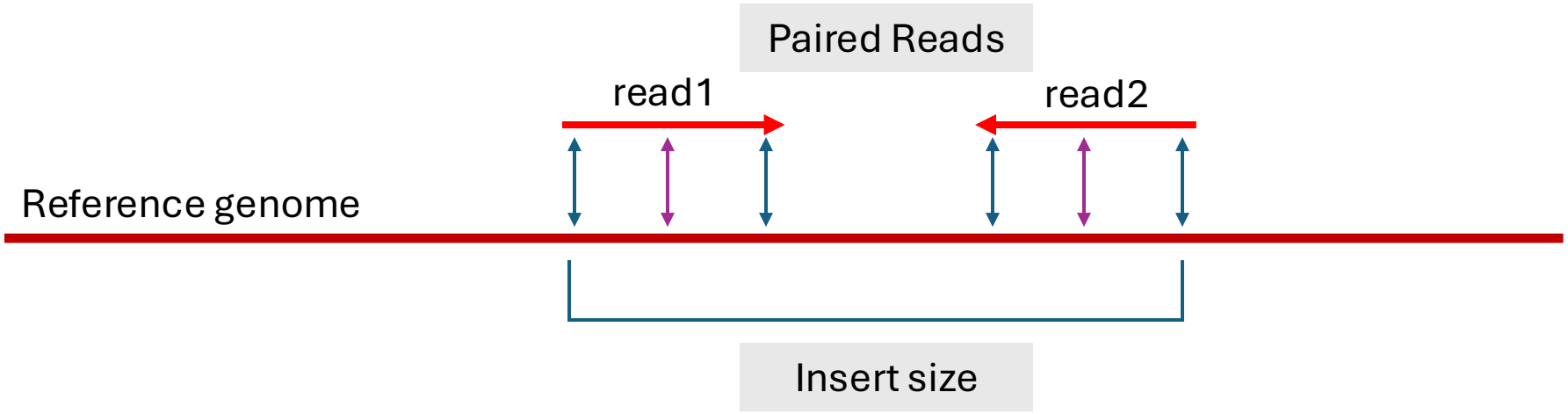
# Short read mapping: the problem

- First, genomic DNA is sheared into fragments that can be size-selected to obtain a template library of uniformly short DNA fragments.
- During sequencing, every molecule is cyclically extended by a single base, and all clusters are read in parallel before the cycle is repeated (Illumina).
- The first step in the analysis of NGS data is to determine where in the genome each of the short reads originated.
- Due to sequencing errors and genuine differences between the reference genome and the sequenced organism, the match is not 100%.

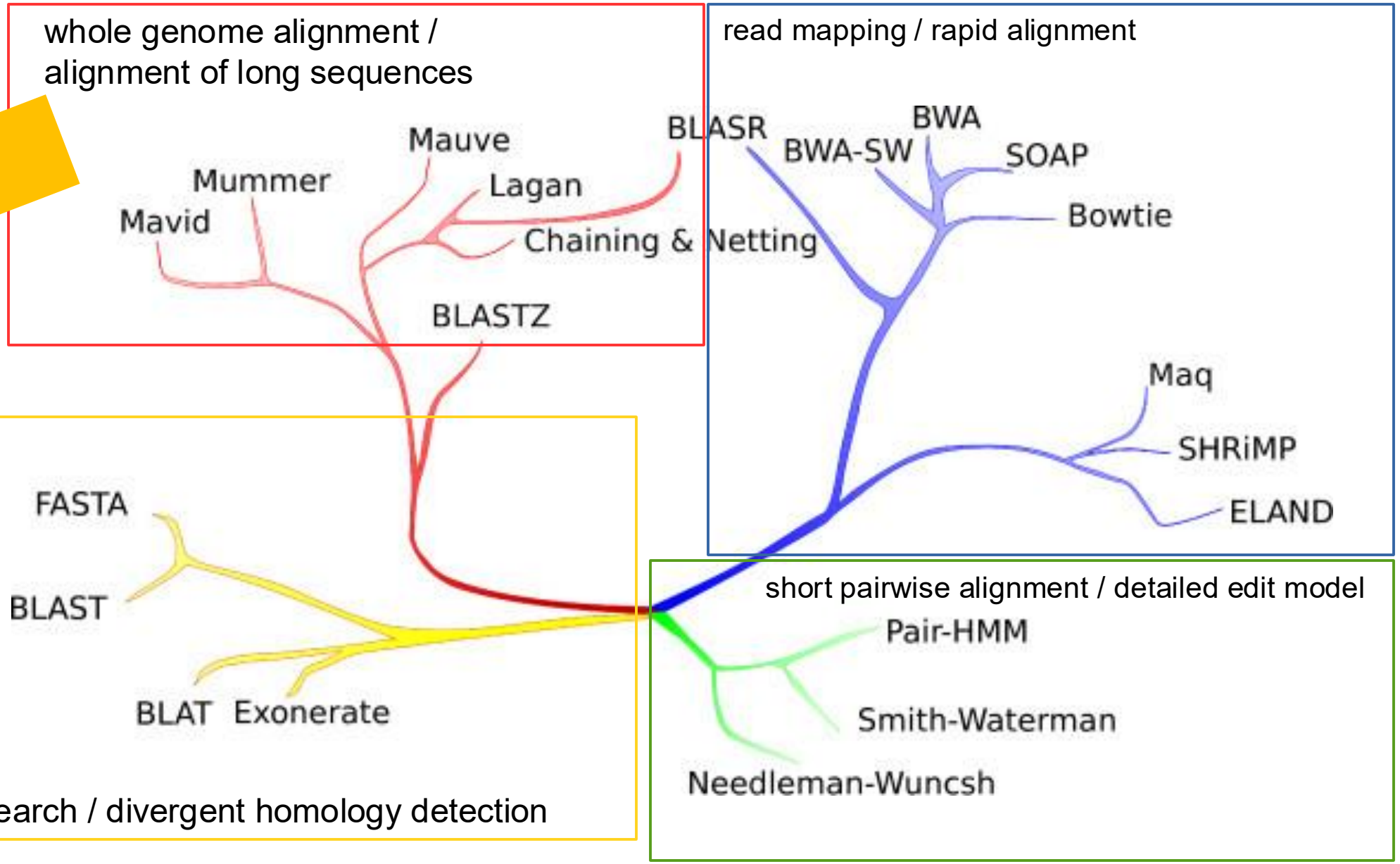


# Short read mapping: the problem

- Big data: sometimes billions of short reads need to be aligned to a large genome. (examples of large genomes?).
- We required new algorithm designs to run orders of magnitude faster.
- Sometimes, we need to incorporate additional information (eg. paired reads)



**Mapping Sequences!!**




Source: Chaisson and Tesler, 2012
https://doi.org/10.1186/1471-2105-13-238

# Read mapping as String Matching

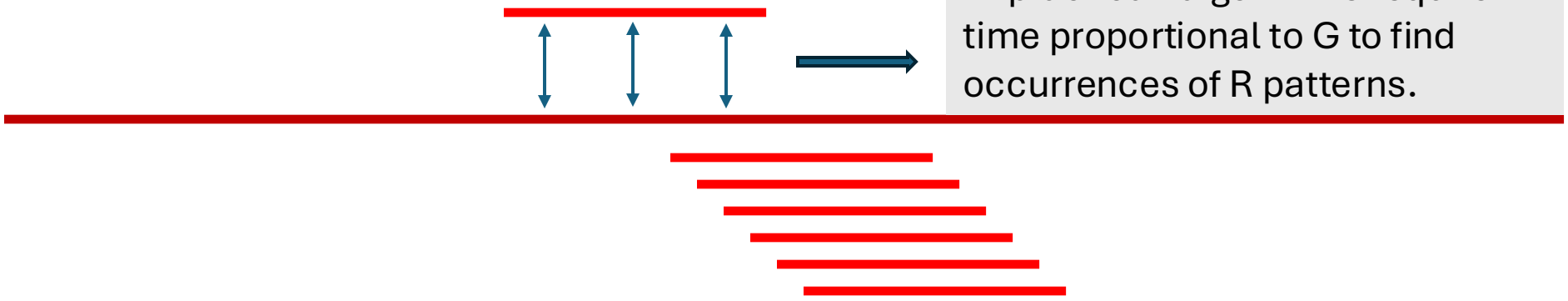
Reference genome: **G**
Substring S: contiguous subsequence

Reads: **R**'s

Alphabet: {A, C, G, T}

**Exact String Matching**

- A read R might be detected through sequence identity of R to a substring in G.
- The problem is finding all occurrences of a pattern in R in G.
- But R's are typically hundreds of reads: classical matching algorithms.



Impractical: algorithms require time proportional to G to find occurrences of R patterns.

# Read mapping as String Matching

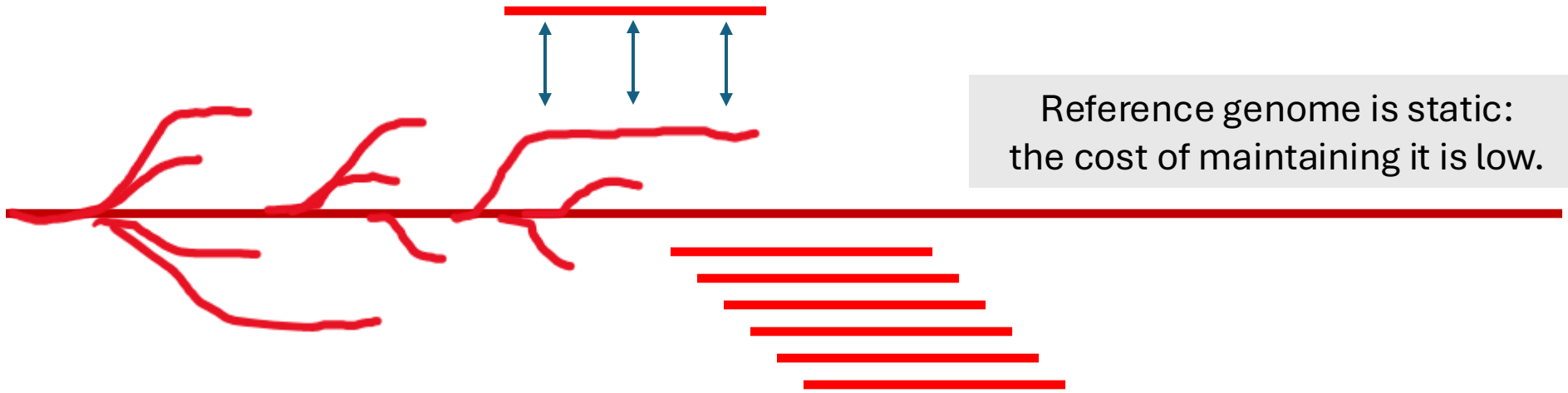
Reference genome: **G**
Substring S: contiguous subsequence

Reads: **R**'s

Alphabet: {A, C, G, T}

**Indexed string matching**

- Builds an auxiliary data structure (index) for the Genome.
- The pattern in R can be found without scanning the complete text in G.
- It speeds up the mapping for millions of reads.
- Minimal cost for constructing the index reference genome.



Reference genome is static:
the cost of maintaining it is low.

Lecture06
Genome comparisons

# Whole Genome Alignment

- A critical process in comparative genomics, facilitating the detection of genetic variants and aiding our understanding of evolution.

- Classification of Whole Genome Alignment Algorithms:

  - **Suffix Tree-Based** Alignment Methods
    - **MUMmer** Technique

  - Anchor based methods
    - LAGAN
    - Mauve
    - BLASTZ
    - LASTZ
    - Minimap2

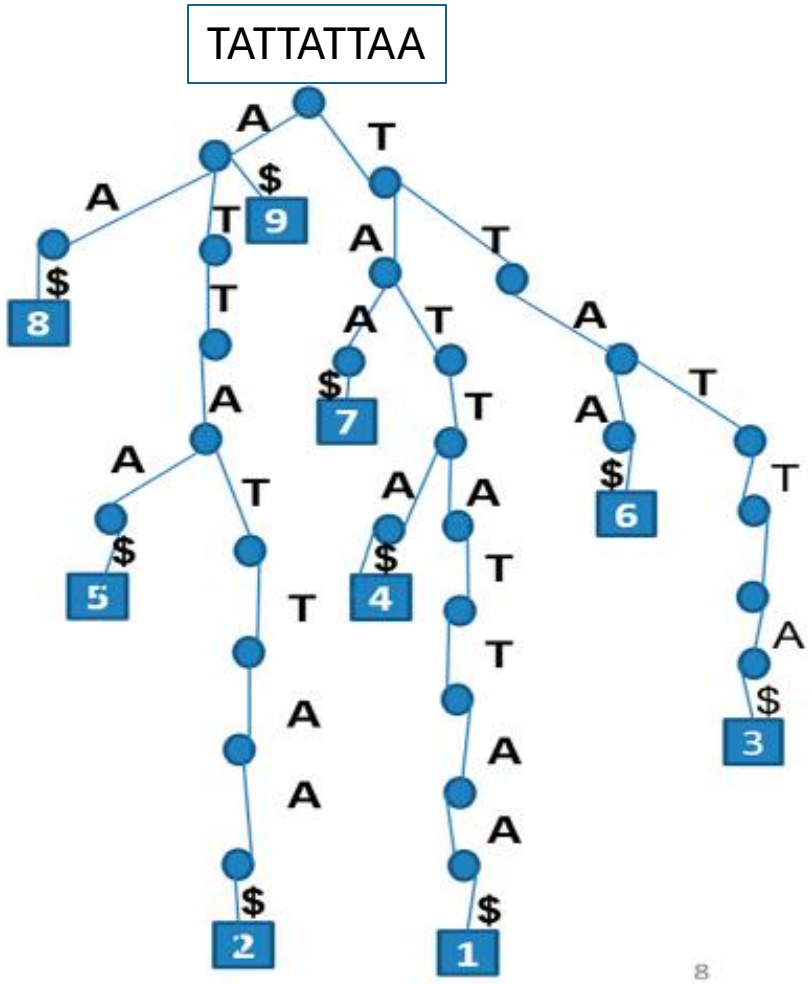
  - Hash-based methods

  - Graph-based methods

Lecture06
Genome comparisons

# Whole Genome Alignment

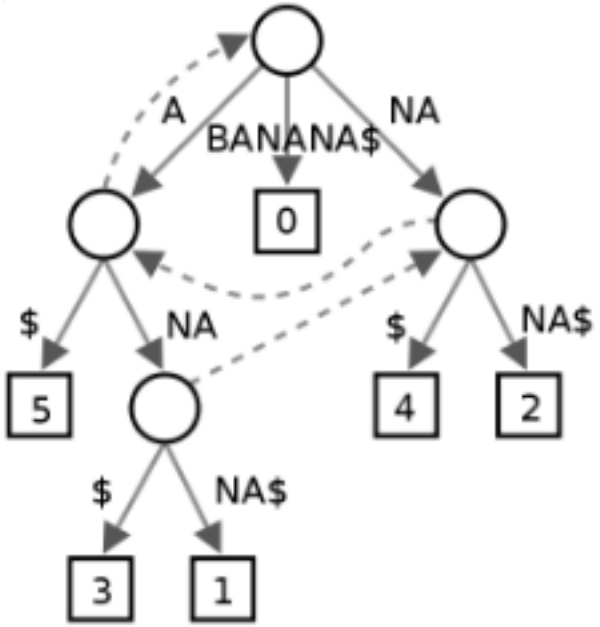
- Suffix Tree-Based Alignment Methods




- It is a compressed data tree structure that represents all the suffixes of a given string.
- The trees save their positions in the text as well as their values.
- TATTATTAA sequence tree with 9 suffixes and $ marking the termination of each suffix with a value.
- Data structured in this way provide fast implementations for string operation.

# Exact pattern matching with indexing

There are many data structures built on suffixes
String: '**banana**'




| | |
|---|---|
| 6 | $ |
| 5 | A$ |
| 3 | ANA$ |
| 1 | ANANA$ |
| 0 | BANANA$ |
| 4 | NA$ |
| 2 | NANA$ |

| | | | | | | |
|---|---|---|---|---|---|---|
| $ | B | A | N | A | N | A |
| A | $ | B | A | N | A | N |
| A | N | A | $ | B | A | N |
| A | N | A | N | A | $ | B |
| B | A | N | A | N | A | $ |
| N | A | $ | B | A | N | A |
| N | A | N | A | $ | B | A |

Suffix Tree

Suffix Array

FM Index

classical *full-text* index structures
Relatively **fast search** functionality
But a large *space requirement*

Ferragina and Manzini developed their full-text index which takes space proportional to the *compressed* text

# Read mapping as String Matching

- **Suffix trees** and **suffix arrays** are classical **full-text index structures**.

- **Suffix arrays** are an array of integers specifying the lexicographical order of the suffixes in the reference genome, and they are more space efficient than suffix trees.

- They permit (almost) optimal *O(n)* search time, proportional to:
  - number of occurrences of a pattern in the genome.
  - Size of the genome.
  - Length of the read.
  - Number of occurrences

- However, both data structures demand memory resources if the index (Genome) is too large.
  - ie. A suffix tree of the human genome needs ~45 GB of space.

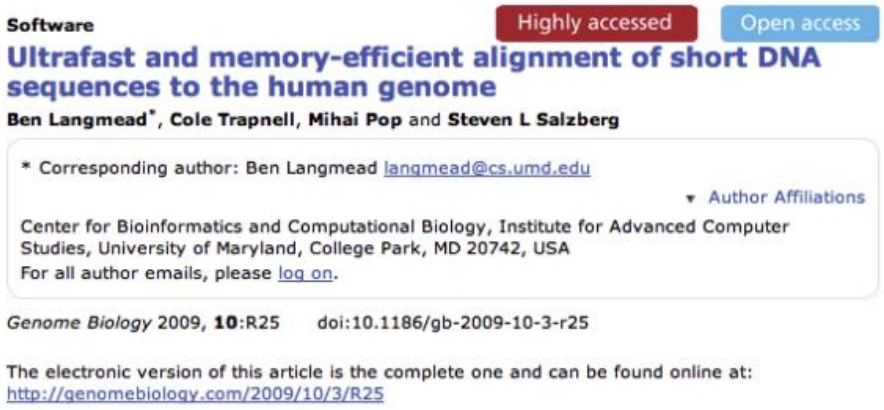
- It wasn't until Ferragina and Manzini (2000) developed the full-text index based on the **Burrows-Wheeler compression algorithm** that the space requirement was reduced while preserving fast functionality.

http://dx.doi.org/10.1109/SFCS.2000.892127

# The Burrows-Wheeler Transform (BWT)

- It is a combination of the Burrows-Wheeler compression algorithm with the the Suffix Array data structure to obtain a sort of compressed suffix array.
- This transformation of a string has been useful in compression: that was its original motivation, and the **b** in bzip2 stands for this.
- BWT also has formed the basis of a number of modern string algorithms that can operate on text using a small amount of space.

**Bowtie**



Software

Highly accessed | Open access

**Ultrafast and memory-efficient alignment of short DNA sequences to the human genome**

**Ben Langmead***, **Cole Trapnell**, **Mihai Pop** and **Steven L Salzberg**

* Corresponding author: Ben Langmead langmead@cs.umd.edu

▾ Author Affiliations

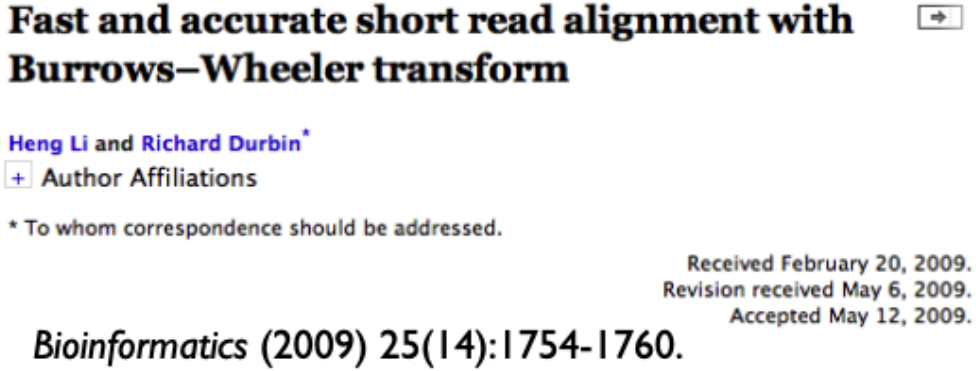
Center for Bioinformatics and Computational Biology, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA

For all author emails, please log on.

*Genome Biology* 2009, **10**:R25 doi:10.1186/gb-2009-10-3-r25

The electronic version of this article is the complete one and can be found online at: http://genomebiology.com/2009/10/3/R25

**bwa**



**Fast and accurate short read alignment with Burrows–Wheeler transform**

Heng Li and Richard Durbin*

+ Author Affiliations

* To whom correspondence should be addressed.

Received February 20, 2009.
Revision received May 6, 2009.
Accepted May 12, 2009.

*Bioinformatics* (2009) 25(14):1754-1760.

## Bowtie alignment performance versus SOAP and Maq

| | Platform | CPU time | Wall clock time | Reads mapped per hour (millions) | Peak virtual memory footprint (megabytes) | Bowtie speed-up | Reads aligned (%) |
|---|---|---|---|---|---|---|---|
| Bowtie -v 2 | Server | 15 m 7 s | 15 m 41 s | 33.8 | 1,149 | - | 67.4 |
| SOAP | | 91 h 57 m 35 s | 91 h 47 m 46 s | 0.10 | 13,619 | 351× | 67.3 |
| Bowtie | PC | 16 m 41 s | 17 m 57 s | 29.5 | 1,353 | - | 71.9 |
| Maq | | 17 h 46 m 35 s | 17 h 53 m 7 s | 0.49 | 804 | 59.8× | 74.7 |
| Bowtie | Server | 17 m 58 s | 18 m 26 s | 28.8 | 1,353 | - | 71.9 |
| Maq | | 32 h 56 m 53 s | 32 h 58 m 39 s | 0.27 | 804 | 107× | 74.7 |

Performance and sensitivity of Bowtie v0.9.6, SOAP v1.10, and Maq v0.6.6 when aligning **8.84 M reads (35 bp)**.

Source: https://doi.org/10.1186/gb-2009-10-3-r25

# The Burrows-Wheeler Transform (BWT)

- Most modern aligners use a full-text index in Minute space, or **FM-Index** as the genome index structure.

- They perform well in both overall runtime and memory usage.

- The FM index's most important component is its use of the Burrows-Wheeler transform (BWT) of the reference genome.

- The BWT is created by first building an **array of suffix rotations in the genome**.

    - starting with the full genome as the first suffix
    - then the next suffix removes the first letter and appends it onto the end.
    - ie. the suffix rotations of the word “banana” and “$” match the end of the string:
        - banana$
        - anana$b
        - nana$ba

# The Burrows-Wheeler Transform (BWT)

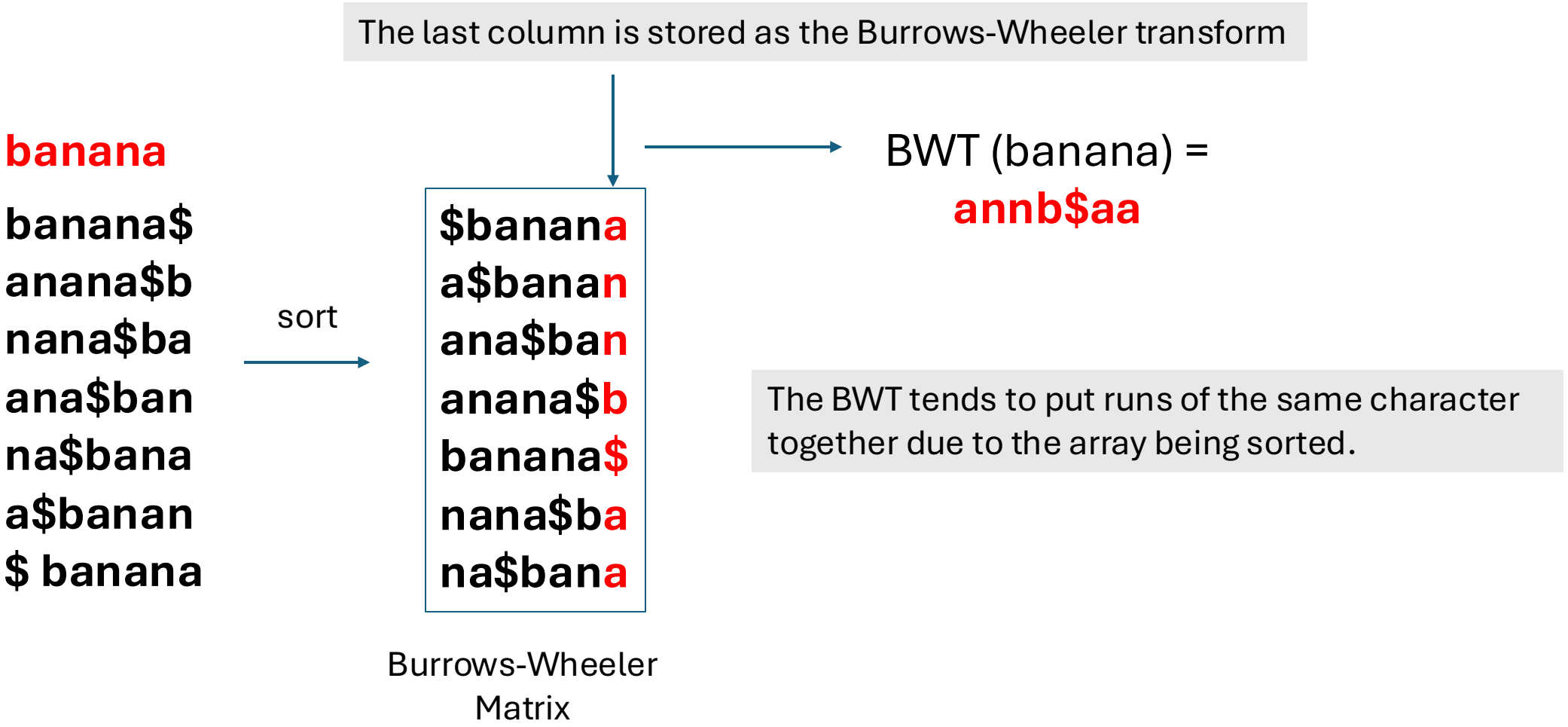
Text transform that is useful for compression & search.

**banana**

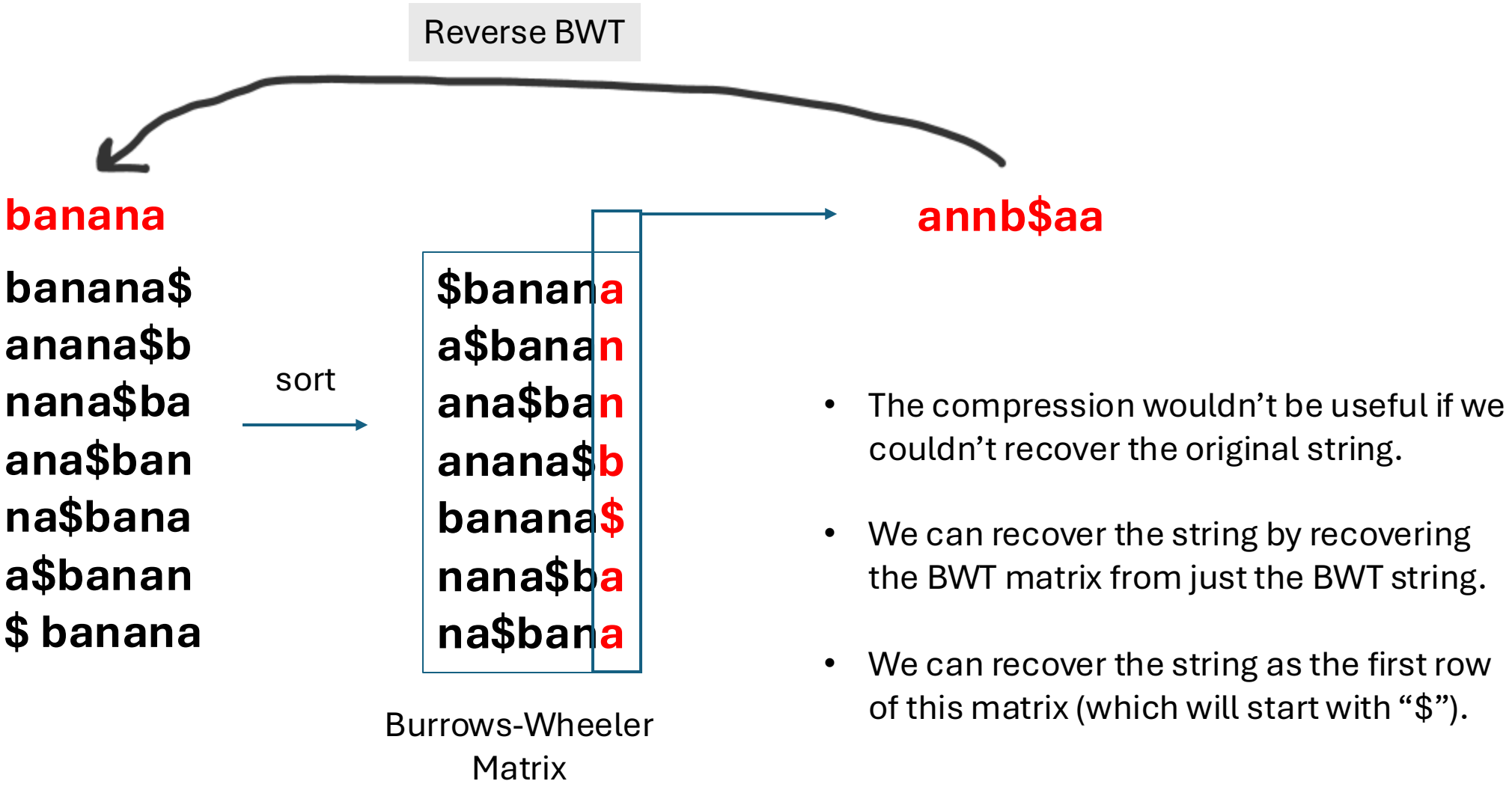
| | | |
|---|---|---|
| **banana$** | | **$banana** |
| **anana$b** | | **a$banan** |
| **nana$ba** | sort → | **ana$ban** |
| **ana$ban** | | **anana$b** |
| **na$bana** | | **banana$** |
| **a$banan** | | **nana$ba** |
| **$banana** | | **na$bana** |

- "**$**" match the end of the string
- generate **all rotations** of the reference sequence.
- **sort** them lexicographically.

# The Burrows-Wheeler Transform (BWT)



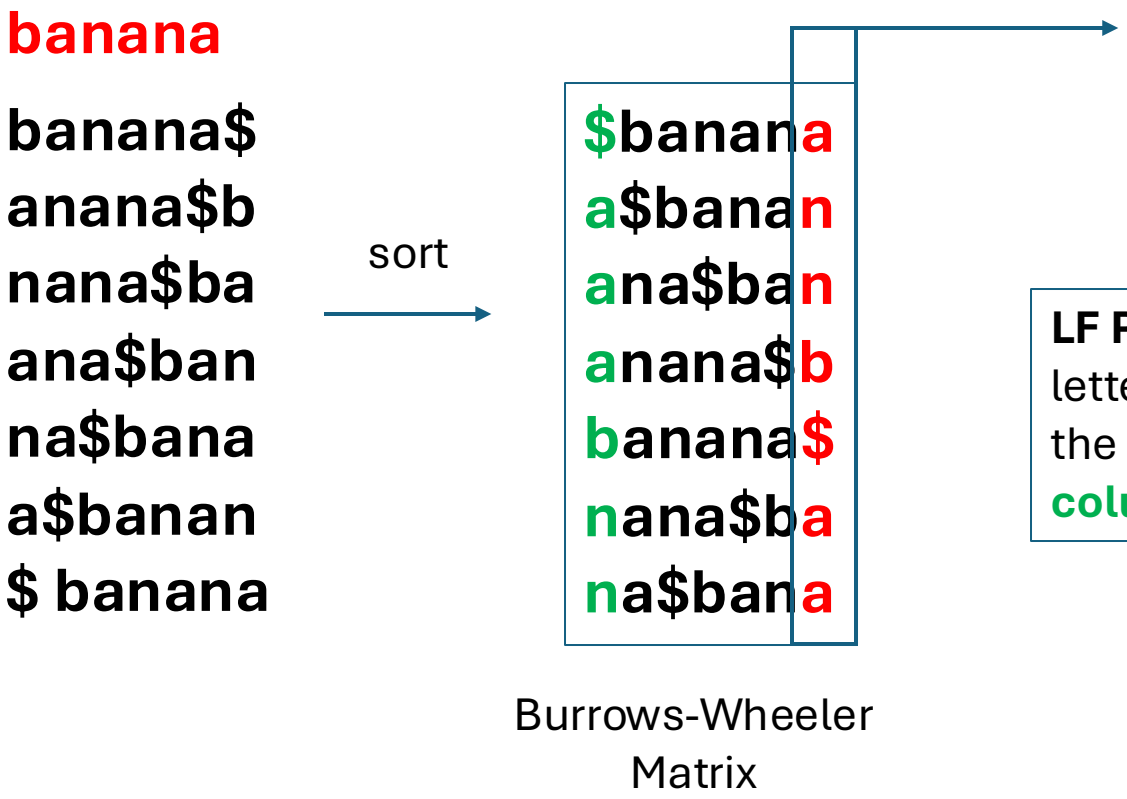
# The Burrows-Wheeler Transform (BWT)




- The compression wouldn't be useful if we couldn't recover the original string.
- We can recover the string by recovering the BWT matrix from just the BWT string.
- We can recover the string as the first row of this matrix (which will start with "$").

# The Burrows-Wheeler Transform (BWT)

Now, we pair up F and L (ranked) to establish a mapping from one column to the other. This will be used for LF-mapping.




BWT (banana) =
**annb$aa**

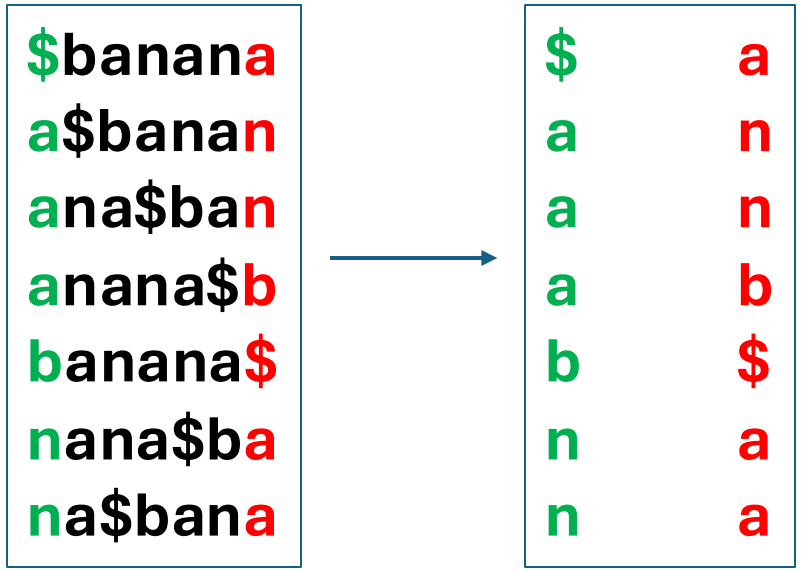
**LF Property**: The order of occurrence of a letter X in the **last column** corresponds to the same order of occurrence of X in the **first column**.

# The Burrows-Wheeler Transform (BWT)

BWT (“banana$”)

**annb$aa**

**T-ranking**
Give each character a rank, equal to # times the character occurred Previously



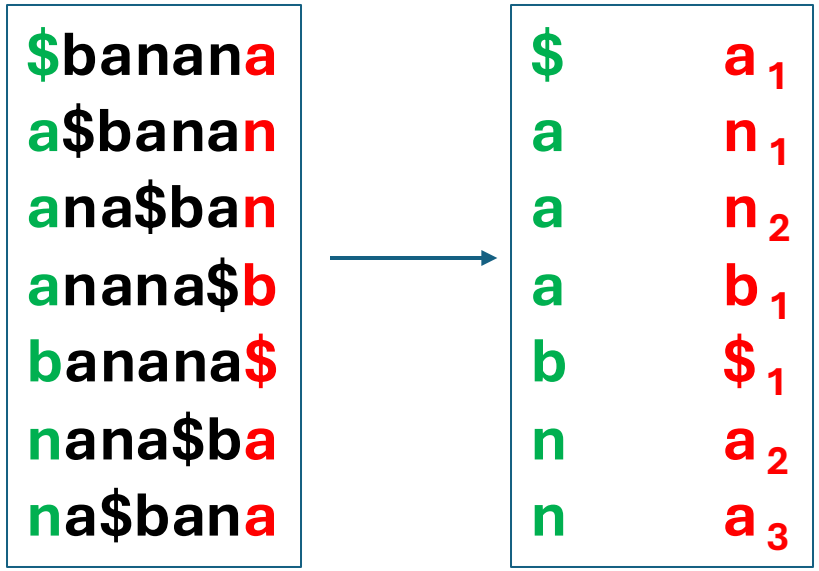

Burrows-Wheeler Matrix

# The Burrows-Wheeler Transform (BWT)

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$

**T-ranking**
Give each character a rank, equal to # times the character occurred Previously




Burrows-Wheeler Matrix

# The Burrows-Wheeler Transform (BWT)

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$

**LF Property**: The order of occurrence of a letter X in the **last column** corresponds to the same order of occurrence of X in the **first column**.

| | |
|---|---|
| $\$_1$ banana$a_1$ | |
| $a_1$ $banan$n_1$ | |
| $a_2$ na$ban$n_2$ | |
| $a_3$ nana$b$b_1$ | |
| $b_1$ anana$$_1$ | |
| $n_1$ ana$ba$a_2$ | |
| $n_2$a$bana$a_3$ | |

Burrows-Wheeler Matrix

| first | last |
|---|---|
| $\$_1$ | $a_1$ |
| $a_1$ | $n_1$ |
| $a_2$ | $n_2$ |
| $a_3$ | $b_1$ |
| $b_1$ | $\$_1$ |
| $n_1$ | $a_2$ |
| $n_2$ | $a_3$ |

| F | L |
|---|---|
| $\$_1$ | $a_1$ |
| $a_1$ | $n_1$ |
| $a_2$ | $n_2$ |
| $a_3$ | $b_1$ |
| $b_1$ | $\$_1$ |
| $n_1$ | $a_2$ |
| $n_2$ | $a_3$ |

**a**'s occur in the same order in F and L.

**n**’s: same order in F and L.

# The Burrows-Wheeler Transform (BWT)

Recovering (inverting) original string using BWT

BWT (“banana$”)

**$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$**



**Reconstruct the first column**
**sort BWT**

| first | last |
|---|---|
| $\$_1$ | $a_1$ |
| a | $n_1$ |
| a | $n_2$ |
| a | $b_1$ |
| b | $\$_1$ |
| n | $a_2$ |
| n | $a_3$ |

# The Burrows-Wheeler Transform (BWT)

Recovering (inverting) original string using BWT

BWT (“banana$”)

**$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$**

**Reconstruct the first column**
**sort BWT**

**LF Property**: The order of occurrence of a letter X in the **last column** corresponds to the same order of occurrence of X in the **first column**.

| first | last |
|---|---|
| $\$_1$ | $a_1$ |
| $a_1$ | $n_1$ |
| $a_2$ | $n_2$ |
| $a_3$ | $b_1$ |
| $b_1$ | $\$_1$ |
| $n_1$ | $a_2$ |
| $n_2$ | $a_3$ |

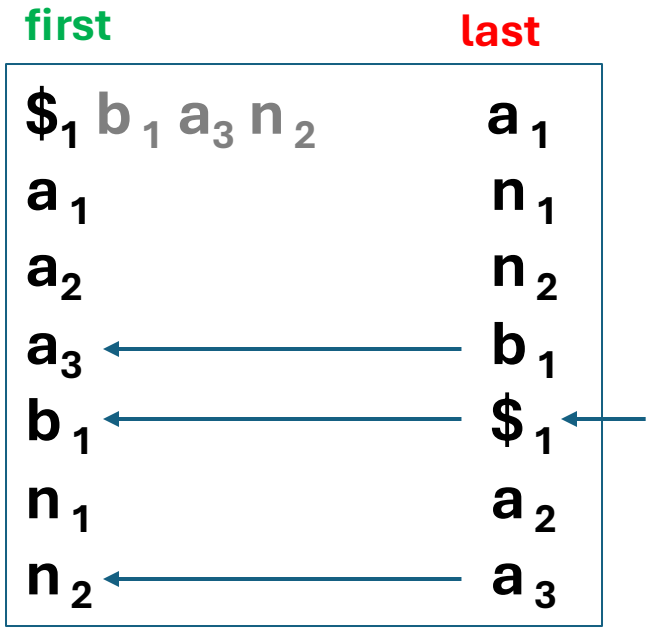
Reconstruct the original string by the following letters in the rotations.

# The Burrows-Wheeler Transform (BWT)

Recovering (inverting) original string using BWT

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$



# The Burrows-Wheeler Transform (BWT)

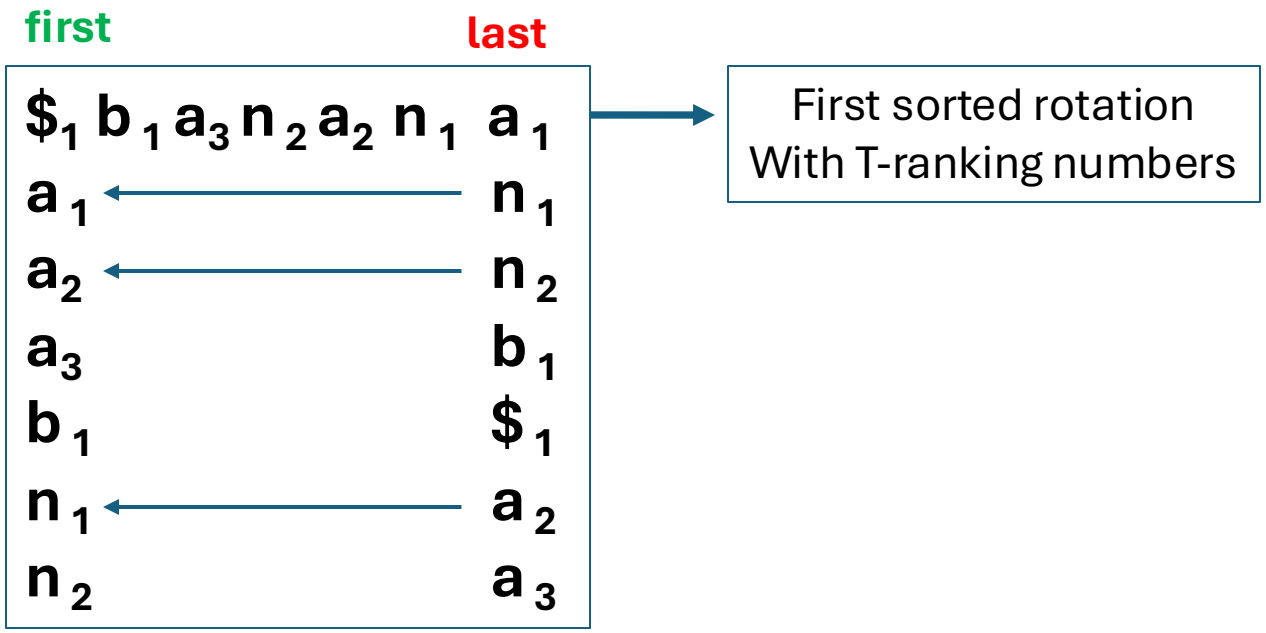
Recovering (inverting) original string using BWT

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$

**original string**

$b_1 a_3 n_2 a_2 n_1 a_1 \$_1$



# The Burrows-Wheeler Transform (BWT)

Pattern matching using BWT

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$

**Reconstruct the first column**
**sort BWT**

| i | first | last | L2F (i) |
|---|---|---|---|
| 0 | $\$_1$ banan | $a_1$ | 1 |
| 1 | $a_1$ \$bana | $n_1$ | 5 |
| 2 | $a_2$ na\$ba | $n_2$ | 6 |
| 3 | $a_3$ nana\$ | $b_1$ | 4 |
| 4 | $b_1$ anana | $\$_1$ | 0 |
| 5 | $n_1$ ana\$b | $a_2$ | 2 |
| 6 | $n_2$ a\$ban | $a_3$ | 3 |

Given a query, I can trace back to the original string

-Keep track of the first and last column

-Provide row numbers

- **First** column
- **Last** to **First** column row numbers

# The Burrows-Wheeler Transform (BWT)

Pattern matching using BWT

BWT ("banana$")

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$



**Reconstruct the first column**



-Keep track of the first and last column
-Provide row numbers

- First column
- Last to First column row numbers

| i | first | last | L2F (i) |
|---|---|---|---|
| 0 | $\$_1$ | $a_1$ | 1 |
| 1 | $a_1$ | $n_1$ | 5 |
| 2 | $a_2$ | $n_2$ | 6 |
| 3 | $a_3$ | $b_1$ | 4 |
| 4 | $b_1$ | $\$_1$ | 0 |
| 5 | $n_1$ | $a_2$ | 2 |
| 6 | $n_2$ | $a_3$ | 3 |

**LF mapping**: The $i^{th}$ occurrence of a character *c* in **L** and the $i^{th}$ occurrence of *c* in **F** correspond to the same occurrence in the original string.

# The Burrows-Wheeler Transform (BWT)

Pattern matching using BWT

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$

top

bottom

| i | first | last | L2F (i) |
|---|---|---|---|
| 0 | $\$_1$ | $a_1$ | 1 |
| 1 | $a_1$ | $n_1$ | 5 |
| 2 | $a_2$ | $n_2$ | 6 |
| 3 | $a_3$ | $b_1$ | 4 |
| 4 | $b_1$ | $\$_1$ | 0 |
| 5 | $n_1$ | $a_2$ | 2 |
| 6 | $n_2$ | $a_3$ | 3 |

- Given a query, I can trace back to the original string.
- In the original BWT matrix, all rotations that started with the same prefix were plumbed together.
- If I start with the top first rotation of the start of my query, and bottom the the last rotation of the start of my query, the range between top and bottom is exactly all instances of my query in the original string.

# The Burrows-Wheeler Transform (BWT)

Pattern matching using BWT

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$

top

bottom

| i | first | last | L2F (i) |
|---|---|---|---|
| 0 | $\$_1$ | $a_1$ | 1 |
| 1 | $a_1$ | $n_1$ | 5 |
| 2 | $a_2$ | $n_2$ | 6 |
| 3 | $a_3$ | $b_1$ | 4 |
| 4 | $b_1$ | $\$_1$ | 0 |
| 5 | $n_1$ | $a_2$ | 2 |
| 6 | $n_2$ | $a_3$ | 3 |

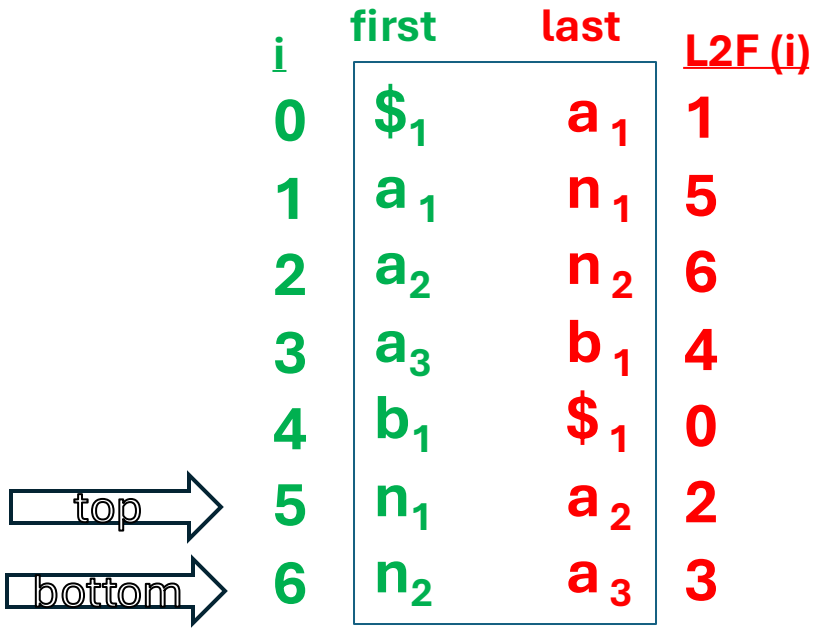
- Given a query, I can trace back to the original string.
- String: ‘an’ (query)
- Search for my word backward.
- First ‘n’: search **rotations** in the **last** column for ‘n’

# The Burrows-Wheeler Transform (BWT)

Pattern matching using BWT

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$



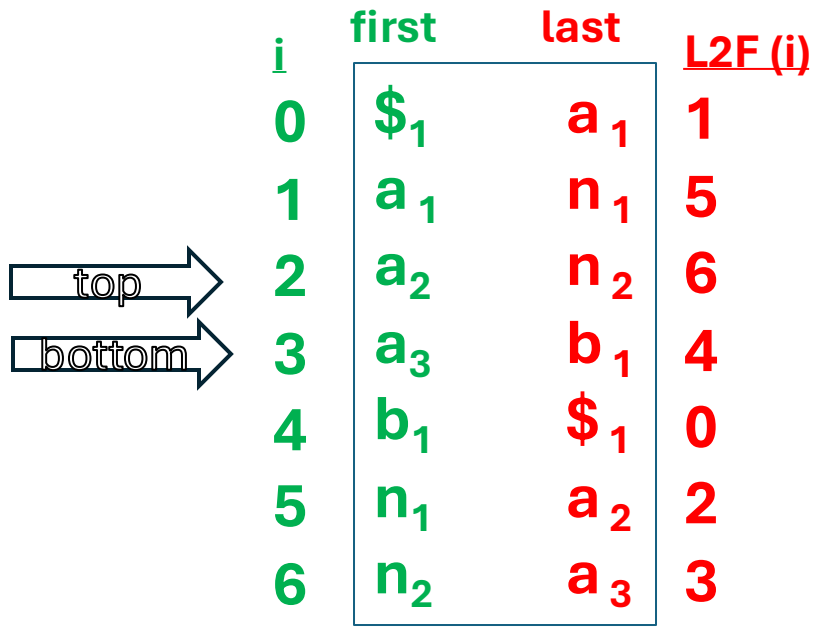

- Given a query, I can trace back to the original string.
- String: ‘an’ (query)
- Search for my word backward.
- First ‘n’
- Look for next ‘a’ between the range (top-bottom).

# The Burrows-Wheeler Transform (BWT)

Pattern matching using BWT

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$




- Given a query, I can trace back to the original string.
- String: ‘an’ (query)
- Search for my word backward.
- First ‘n’
- Look for next ‘a’ between the range (top-bottom).
- $a_2$ **and** $a_3$ are the two instances of my query.
- Map back those rotations to the original string.

**original string**

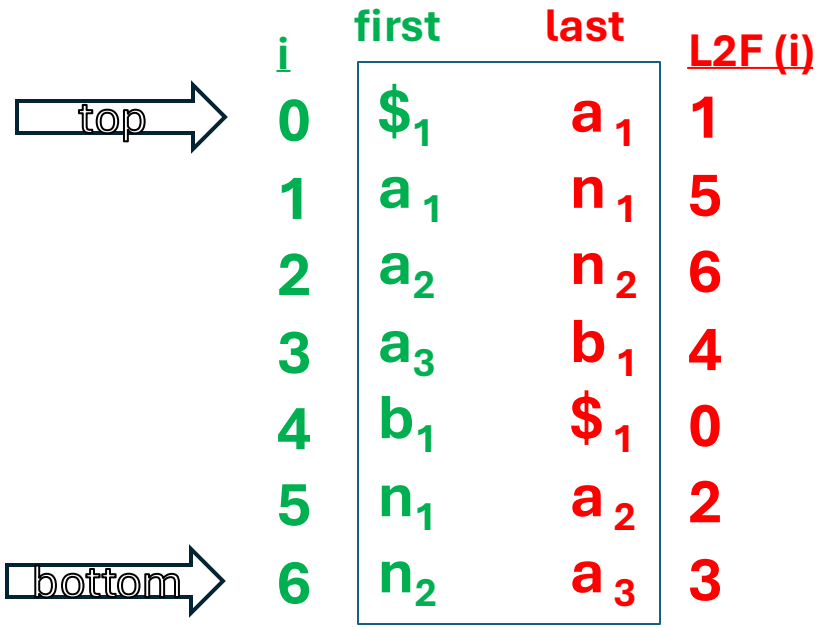
$b_1 a_3 n_2 a_2 n_1 a_1 \$_1$

# The Burrows-Wheeler Transform (BWT)

Pattern matching using BWT

BWT (“banana$”)

$a_1 n_1 n_2 b_1 \$_1 a_2 a_3$




- Given a query, I can trace back to the original string.
- String (query):
  - ‘na’
  - ‘nan’
  - ‘ba’
  - ‘bana’

**original string**

$b_1 a_3 n_2 a_2 n_1 a_1 \$_1$

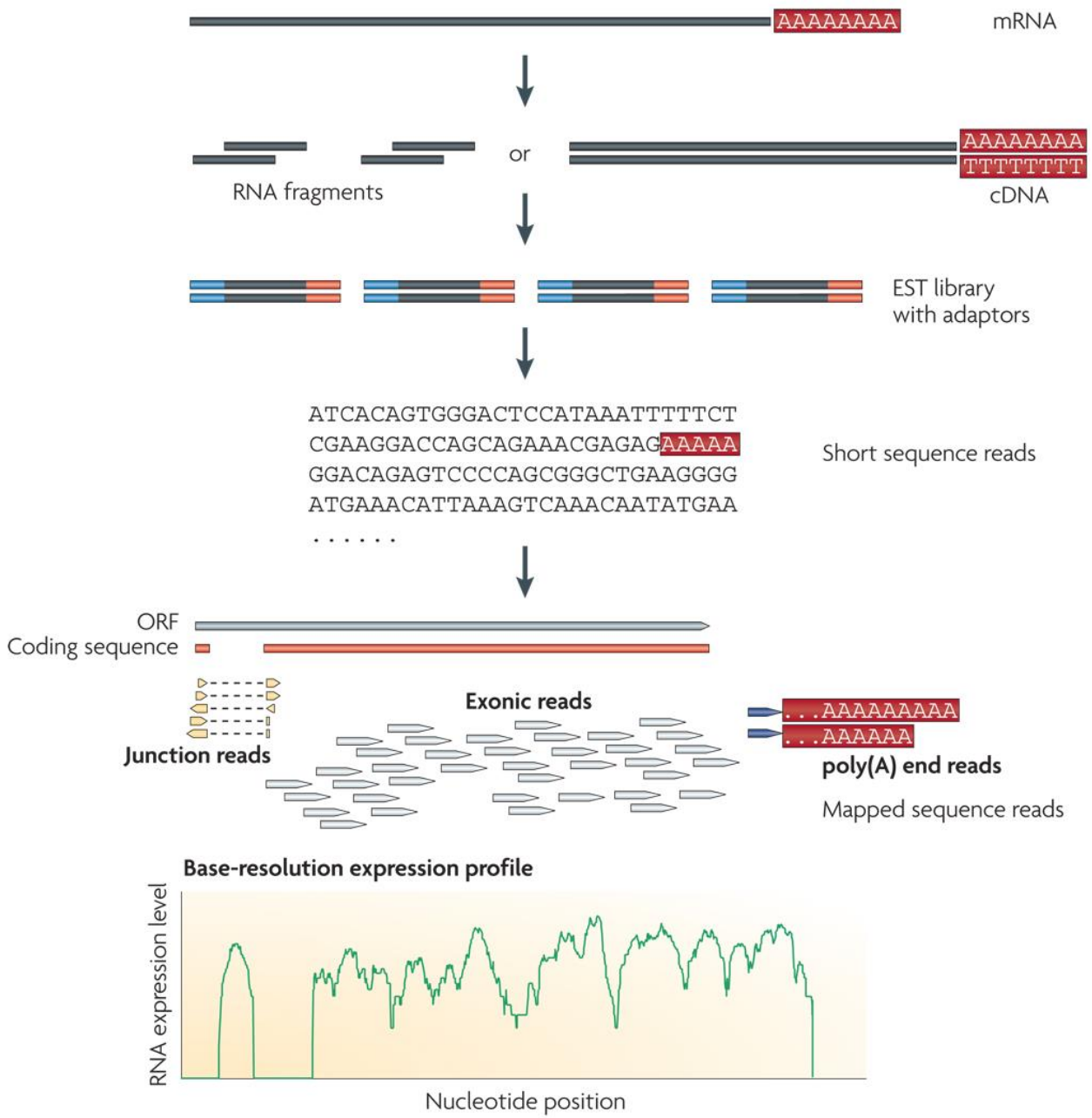
# The Burrows-Wheeler Transform (BWT)

- BWT is useful for pattern searching and compression.
- BTW is invertible: given the BTW of a string, the original string can be reconstructed.
- BWT is computable in *O(n)* time: *O(|pattern|)*
- There is a close relationship between Suffix Trees, Suffix Arrays, and BWT:
    - Suffix Array: order of the suffix numbers of the suffix tree, traversed left to right.
    - BWT: letters at positions given by the suffix array entries.
- After compression, strings can be searched quickly with BWT:
    - Don’t have to store the LF mapping
    - Allows computing searches (another algorithm) in compressed data *on the fly* (with some extra storage)

# RNA-seq

- RNA-seq is the sequencing  of coding and non-coding transcripts.
    - Eukaryotes: polyA tail in mRNAs
    - But also: miRNA, piRNAs, siRNAs, lncRNAs,… is RNA-seq
- Requires reverse transcribe RNA -> DNA -> cDNA library.
- Applications:
    - Gene prediction.
    - *denovo* assembly of transcripts.
    - RNA-seq quantification: gene expression / differential gene expression.
    - Inference of alternating splicing.

# RNA-seq

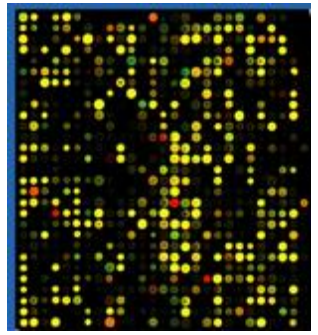



Source: https://doi.org/10.1038/nrg2484

# RNA-seq *vs*. Microarrays

- **Microarrays**: measure transcription using probes for specific genes:
  - One probe: one gene
  - Relies on hybridization
  - Requires knowledge of gene sequences

- But microarrays have
  - low reproducibility
  - Limits of detection at extreme high/low expression

- **RNA-seq** has improved reproducibility.

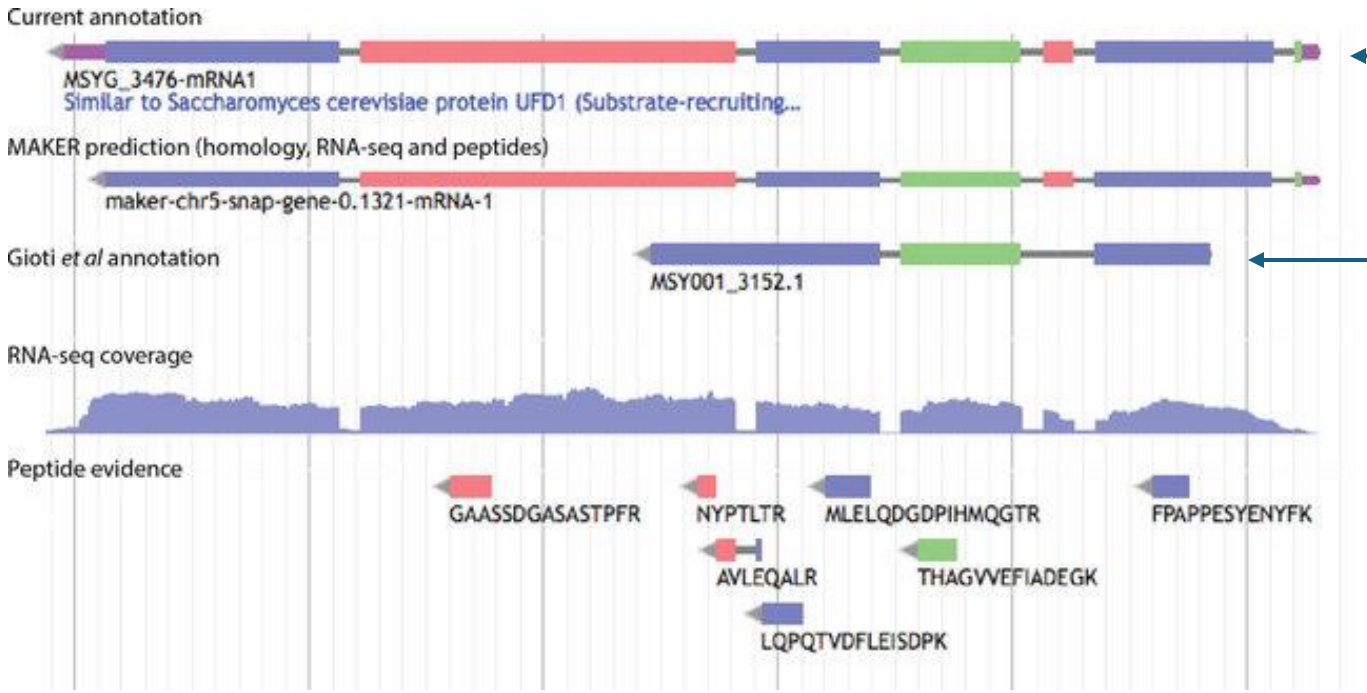
- There are no limitations at high/low expression levels.
  - Low expression is solved by increasing the yield (sequencing depth).

- Now, it is more affordable to perform RNA-seq.

# RNA-seq – Gene Prediction

- Gene finding in eukaryotic genomes is notoriously difficult to automate, even with HMMs.
- Most gene prediction tools incorporate RNA-Seq data into unsupervised/supervised training and subsequently generate *ab initio* gene predictions.

HMMs
+
RNA-seq
+
Protein Homology




Refseq

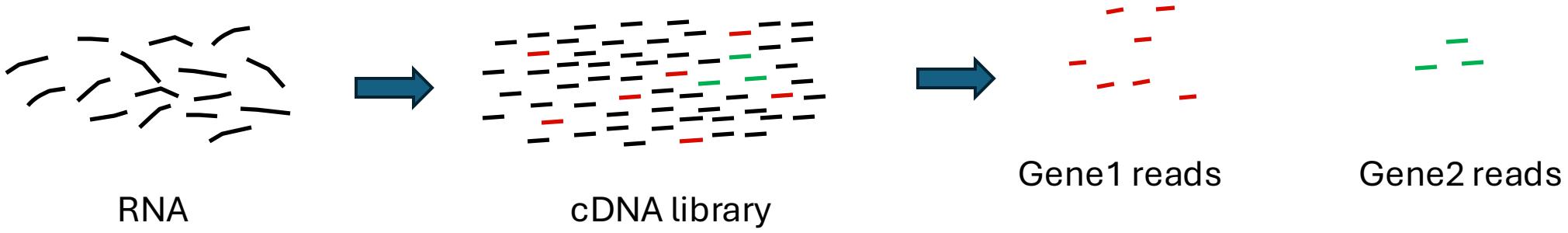
Gene prediction only
HMMs

Protein Homology

http://dx.doi.org/10.1093/nar/gkx006

# RNA-seq quantification

- RNA-seq is a relative abundance measurement technology.
- RNA-seq gives you reads from the ends of a random sample of fragments in your library.
- Without additional data, this only gives information about the **relative** abundances.
- Additional information, such as levels of “spike-in” transcripts, are needed for absolute measurements.



# RNA-seq quantification

Normalization is required to compare samples in these situations.

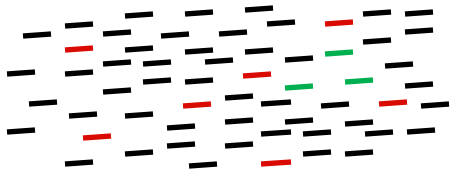
| Gene | Sample 1 absolute abundance | Sample 1 relative abundance | Sample 2 absolute abundance | Sample 2 relative abundance |
|---|---|---|---|---|
| 1 | 20 | 10% | 20 | 5% |
| 2 | 20 | 10% | 20 | 5% |
| 3 | 20 | 10% | 20 | 5% |
| 4 | 20 | 10% | 20 | 5% |
| 5 | 20 | 10% | 20 | 5% |
| 6 | 100 | 50% | 300 | 75% |

# RNA-seq quantification

- In RNA-seq read abundance is also dependent of transcript length.
- The probability of a read coming from a transcript is proportional to the relative abundance x length:

$$\textit{Relative abundance} = \frac{Probability\ of\ read\ from\ transcript}{Transcript\ length} = \frac{\frac{reads-trancript}{total-reads}}{Transcript\ length}$$

| GeneID | Transcript length | # reads |
|---|---|---|
| 1 | 200 | 100 |
| 2 | 60 | 60 |
| | | .... |
| | | Total reads: 200 |



$$\textit{Relative abundance}_1 = \frac{\frac{100}{200}}{200} = \frac{1}{400}$$

$$\textit{Relative abundance}_2 = \frac{\frac{60}{200}}{60} = \frac{2}{400}$$

# RNA-seq quantification

-Basic quantification algorithm:

- Align reads (Burrows-Wheeler Transform) against
  - Genome/gene annotations
  - Set of reference transcripts

- Count the number of reads per feature (gene, CDS, exon, etc..)

- Convert read counts into relative expression levels.

-Normalization methods:

- RPKM - Reads Per Kilobase per Million mapped reads
- FPKM (fragments instead of reads, two reads per fragment for paired-end reads)
- TPM - Transcripts Per Million: normalize for gene length first, and then normalize for sequencing dept.

# RNA-seq quantification

- RPKM - Reads Per Kilobase per Million mapped reads

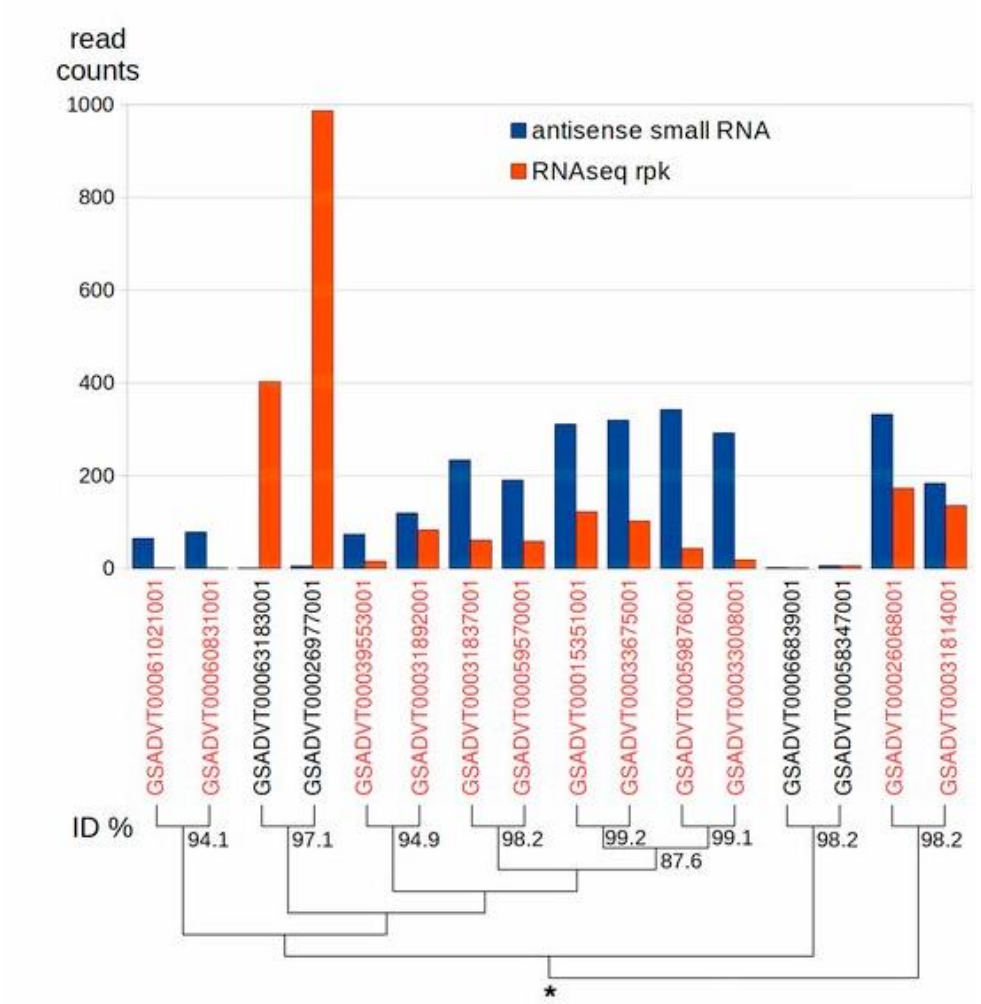
$$RPKM = \frac{ExonMappedReads * 10^9}{TotalMappedReads * ExonLength}$$

- TPM - Transcripts Per Million

$$TPM = \frac{Ni/Li * 10^6}{sum(N1/L1 + N2/L2 + \ldots + Nn/Ln)}$$

$N_i$ is the number of reads compared to the *i*-exon
$L_i$ is the length of the *i*-exon
*Sum(N1/L1+N2/L2+…+Nn/Ln)* is the sum of values of all (n) exons after normalization by length.

# RNA-seq quantification




RNA profiles from the cytochrome P450 node marked with antisense small RNA counts and RNA-seq rpk counts.

Source: https://doi.org/10.1534/genetics.116.186734