

# Introduction to Bioinformatics for Computer Scientists

## *Lecture 4*

# BLAST & Genome assembly

Alexey Kozlov

*Staff Scientist*

*alexey.kozlov@h-its.org*

Exelixis Lab

October 30, 2023

# Today's agenda

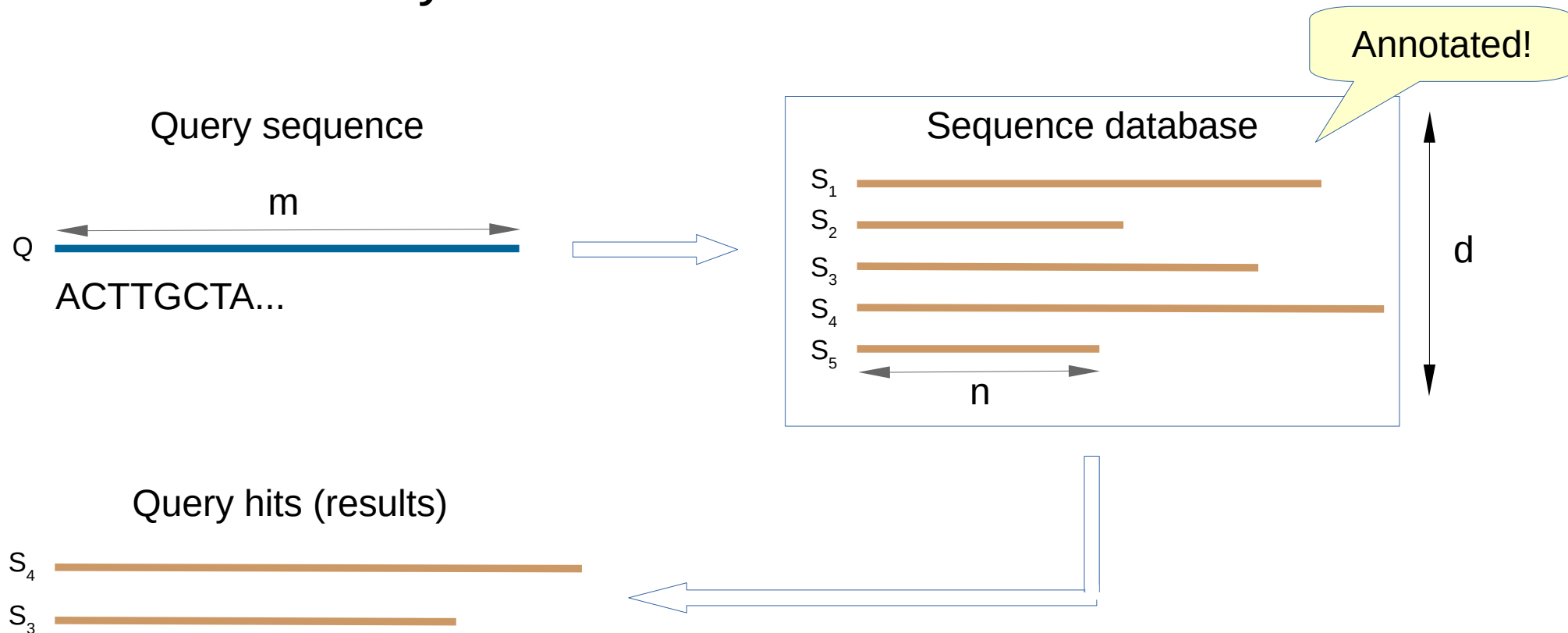
- Search methods for biological sequences
  - Public sequence databases
  - BLAST algorithm
  - Alternative search algorithms
- Genome assembly
  - *De novo* assembly
  - By-reference assembly

# Disclaimer

- Today: Basic algorithms
  - Solving an idealized problem
- Real programs are much more complicated
  - Deal with biological “fuzziness”
  - Heuristics, optimizations...
- I’m not an expert
  - Ask your advanced questions anyways :)
  - Only basic stuff is relevant for the exam

# Searching for similar sequences

- **Assumption:**
  - sequence similarity  $\Leftrightarrow$  evolutionary and/or functionally relatedness



# Sequence databases: proteins

- UniProt

- extensive annotations (3D structure, functions etc.)
- 560K manually curated entries + 180M automatic

## UniProtKB - P04637 (P53\_HUMAN)

### Display

Entry

Publications

Feature viewer

Feature table

None

- Function
- Names & Taxonomy
- Subcellular location
- Pathology & Biotech
- PTM / Processing
- Expression
- Interaction
- Structure

[BLAST](#) [Align](#) [Format](#) [Add to basket](#) [History](#)

**Protein** | Cellular tumor antigen p53

**Gene** | TP53

**Organism** | *Homo sapiens (Human)*

**Status** | Reviewed - Annotation score: ●●●●●● - Experiment

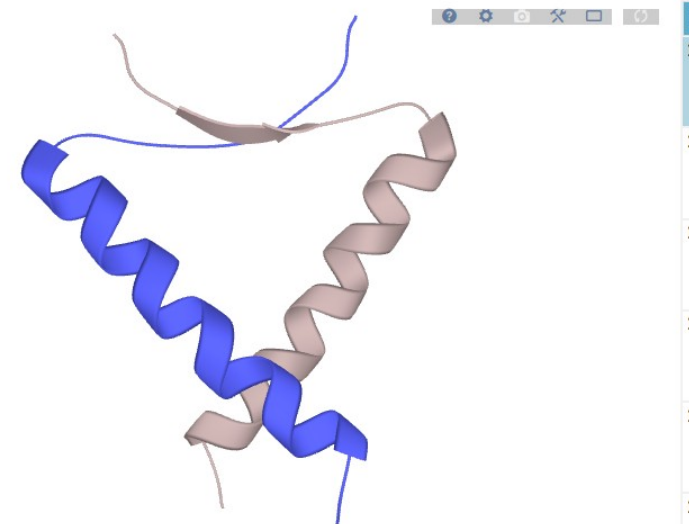
### Function<sup>i</sup>

Acts as a tumor suppressor in many tumor types; induces growth arrest of cells in which a subset of genes required for this process. One of the activated genes is an apoptosis-inducing factor. Apoptotic activity is activated via its interaction with PPP1R13B/ASPP1 (PubMed:12524540). In cooperation with mitochondrial PPIF is involved in the regulation of mitochondrial biogenesis. lincRNA-Mkl1n1. lincRNA-p21 participates in TP53-dependent transcriptional repression of the complex in response to DNA damage, thus stopping cell cycle progression. In cooperation with PIPK1 is involved in cell cycle suppression mediated by isoform 1. Isoform 7 inhibits isoform 1-mediated

### Cofactor<sup>i</sup>

Zn<sup>2+</sup>

**Note:** Binds 1 zinc ion per subunit.



### Secondary structure



Legend: █ Helix █ Turn █ Beta strand █ PDB Structure known for this area

[Show more details](#)

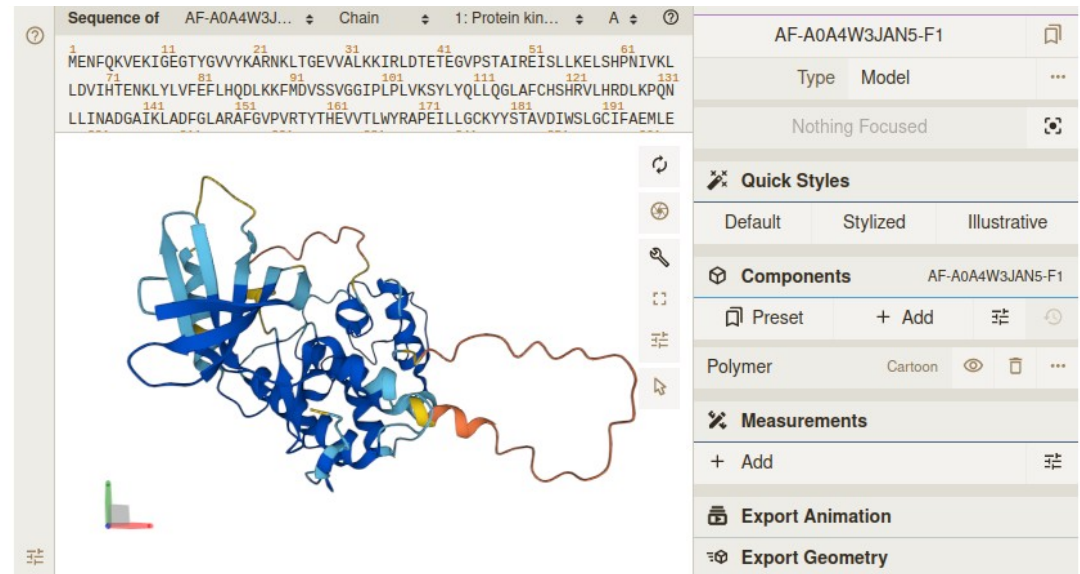
# Sequence databases: proteins

- AlphaFold DB

- 200M protein structures → high-quality prediction
- <https://alphafold.ebi.ac.uk/>

## Information

Protein	Protein kinase domain-containing protein
Gene	cdk2
Source organism	Callorhinchus milii (Ghost shark) <a href="#">go to search</a> <a href="#">↗</a>
UniProt	A0A4W3JAN5 <a href="#">go to UniProt</a> <a href="#">↗</a>
Experimental structures	None available in the PDB
Biological function	Not available. <a href="#">go to UniProt</a> <a href="#">↗</a>



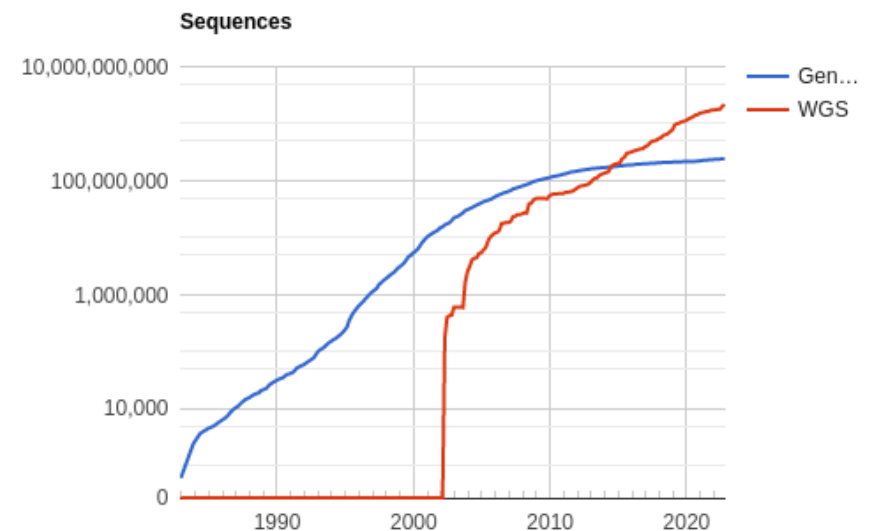
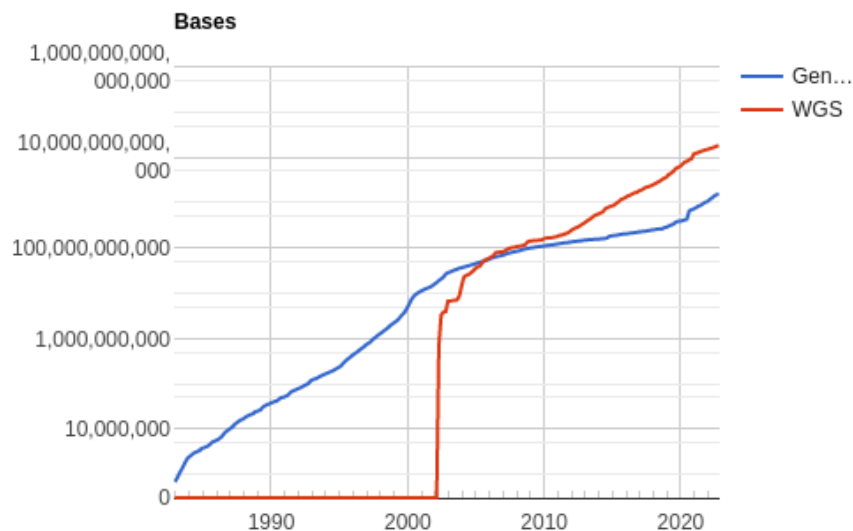
## Model Confidence

- Very high (pLDDT > 90)
- High (90 > pLDDT > 70)
- Low (70 > pLDDT > 50)
- Very low (pLDDT < 50)

AlphaFold produces a per-residue model confidence score (pLDDT) between 0 and 100. Some regions below 50 pLDDT may be unstructured in isolation.

# Sequence databases: GenBank

- **NCBI GenBank/INSDC** → "all" DNA+proteins
  - > 246M "regular" sequences as of Aug. 2023
  - ~ 2.6B whole-genome-shotgun (WGS)
  - "primary" database → sequences submitted and annotated by the respective authors (biologists)



# Sequence annotation in GenBank

## Bacteroides galacturonicus 16S ribosomal RNA gene, partial sequence

GenBank: DQ497994.1

[FASTA](#) [Graphics](#)

Go to:

LOCUS DQ497994 1431 bp DNA linear BCT 01-JUN-2006  
DEFINITION Bacteroides galacturonicus 16S ribosomal RNA gene, partial  
sequence.

ACCESSION DQ497994

VERSION DQ497994.1 GI:95062834

KEYWORDS .

SOURCE Bacteroides galacturonicus  
ORGANISM [Bacteroides galacturonicus](#)  
Bacteria; Bacteroidetes; Bacteroidia; Bacteroidales;  
Bacteroidaceae; Bacteroides.

REFERENCE 1 (bases 1 to 1431)

AUTHORS Song,Y., Liu,C. and Finegold,S.M.  
TITLE Reclassification of Bacteroides galacturonicus  
JOURNAL Unpublished

REFERENCE 2 (bases 1 to 1431)

AUTHORS Song,Y., Liu,C. and Finegold,S.M.  
TITLE Direct Submission  
JOURNAL Submitted (17-APR-2006) VA Medical Center West Los Angeles, 11301  
Wilshire Blvd. Bldg 304 Rm E3-237, Los Angeles, CA 90504, USA

FEATURES  
source Location/Qualifiers  
1..1431  
/organism="Bacteroides galacturonicus"  
/mol\_type="genomic DNA"  
/db\_xref="taxon:384639"  
rRNA <1..>1431  
/product="16S ribosomal RNA"

ORIGIN

```
1 tgcaagtcga acgaagcatt taagacagat tacttcggtt tgaagtc ttt tatgactgag
61 tggcggacgg gtgagtaacg cgtgggtaac ctgcctcata cagggggata gcagctggaa
121 acggctggtg ataccgcata agcgcacagt accacatggt acagtgtgaa aaactccggt
181 ggtatgagat ggaccgcgct ctgattagct tgttggcggg gtaaccggcc accaaggcga
```

entry name:  
usually includes organism  
and gene name

accession number  
(~ sequence ID)

organism name and  
taxonomy (=biological  
classification)

sequence



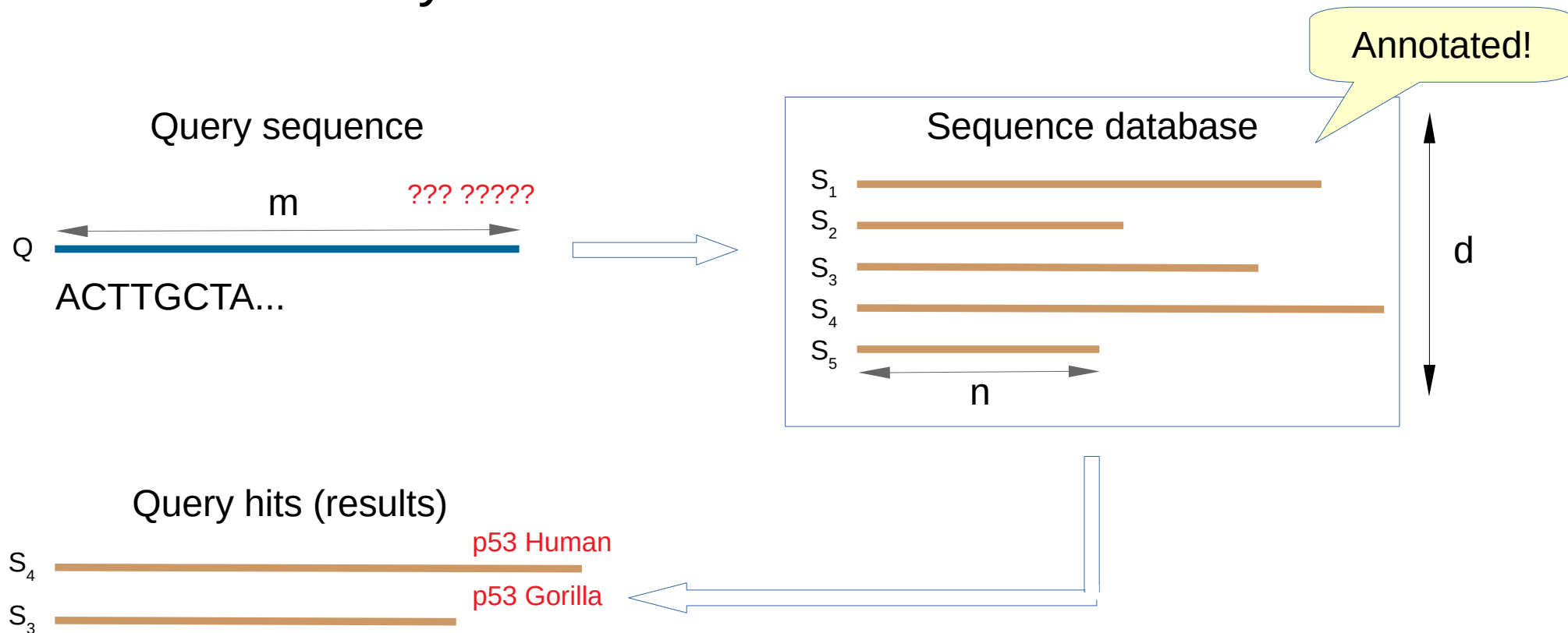
# Sequence databases: barcodes

- “Secondary” databases for barcoding genes
  - quality filtering, curated taxonomy, sample geodata...
  - useful for species identification/classification
- **SILVA** → <https://www.arb-silva.de/>
  - 16S, >10M sequences, focus on Bacteria+Archaea
- **BOLD** → <http://boldsystems.org/>
  - COI, >7M sequences, focus on Animals
- **UNITE** → <https://unite.ut.ee/>
  - ITS, 1.7M sequences, focus on Fungi

# Searching for similar sequences

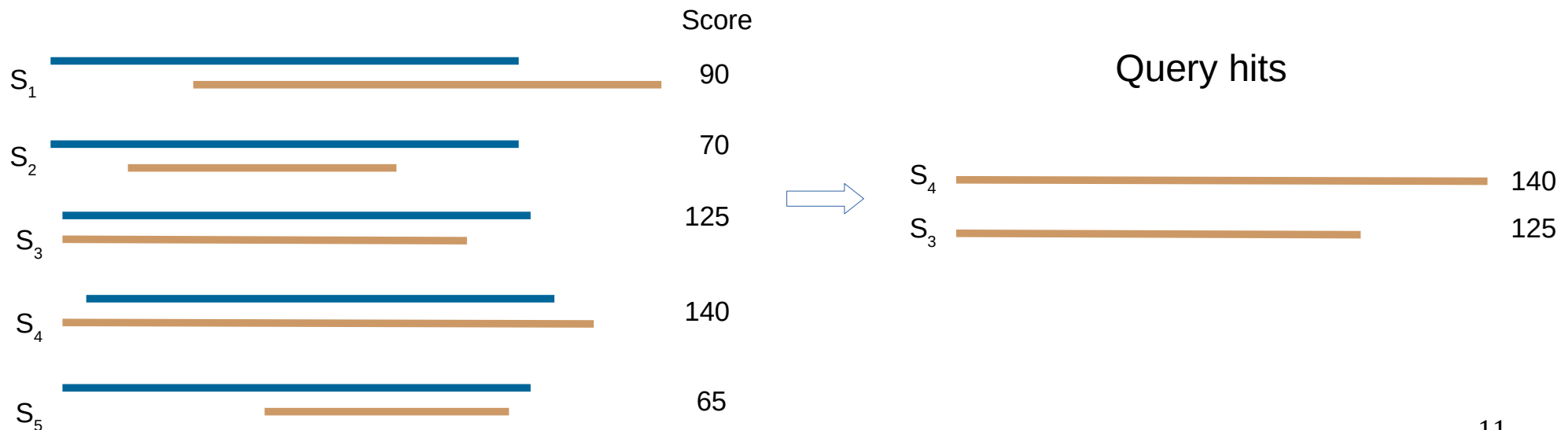
- Assumption:

- sequence similarity  $\Leftrightarrow$  evolutionary and/or functionally relatedness



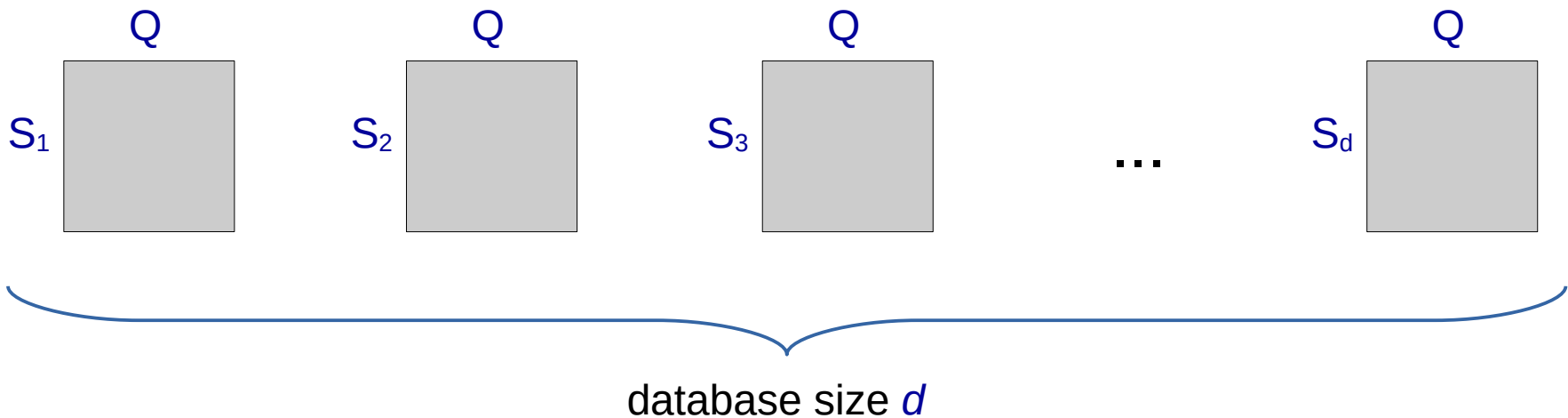
# Naïve approach

- Use **Smith-Waterman** algorithm to build a pairwise alignment of query sequence  $Q$  with every sequence in the database  $S_1 \dots S_d$
- Sort alignments by similarity score
- Report best match(es)



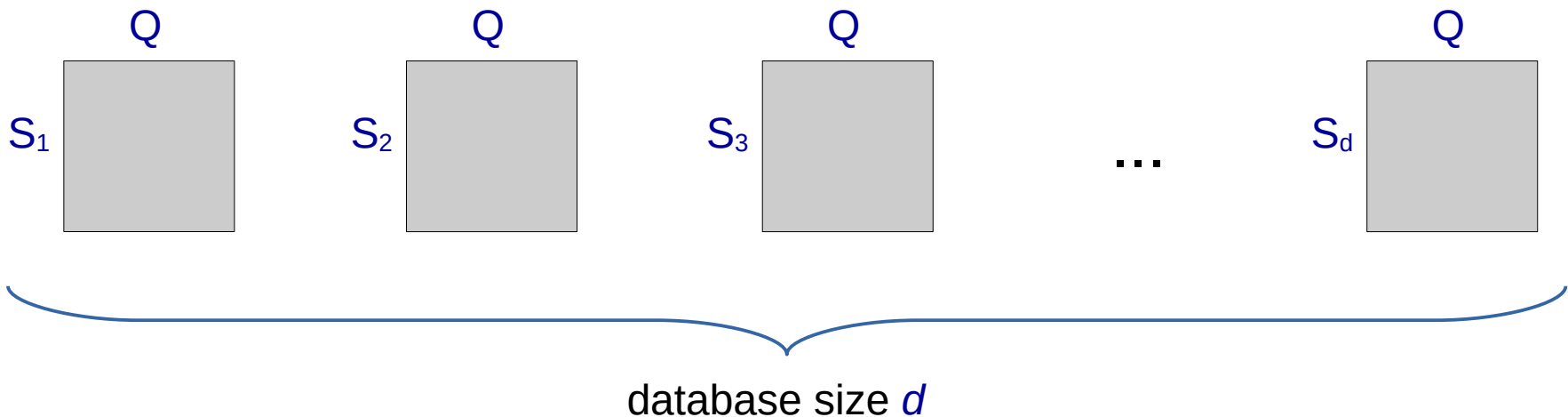
# Naïve approach: complexity

- Smith-Waterman has a complexity of  $O(m \times n)$
- DP matrix for every sequence in database:



# Naïve approach: complexity

- Smith-Waterman has a complexity of  $O(m \times n)$
- DP matrix for every sequence in database:



- NCBI GenBank:  $d = 200$  million
- Do we really need to compute *all* matrices?

# BLAST

- Stands for **B**asic **L**ocal **A**lignment **S**earch **T**ool
- **Fast heuristic** to find similar sequences
- One of the most widely-used algorithms in bioinformatics
  - Two main papers from 1990s have ~180 000 citations<sup>1</sup>
  - cf. RAxML: ~40 000, Human Genome Sequence: ~45 000
- Available as both stand-alone application and web service
  - BLAST on NCBI GenBank database:  
<http://blast.st-va.ncbi.nlm.nih.gov/Blast.cgi>

---

<sup>1</sup> Altschul et al (1990), Altschul et al (1997)

# BLAST: algorithm outline

- **Idea:** reduce search space by ignoring dissimilar regions
- Three-step heuristic:
  - **Seeding:** find common subwords between query and database sequences → *seeds*
  - **Extension:** starting from seeds, extend alignment in both directions → *high-scoring segment pairs (HSP)*
  - **Evaluation:** assess the statistical significance of each HSP

# BLAST: seeding

- Pre-process **query** sequence
- Build a list  $L_1$  of all subwords (factors) of the length  $w$  in the query sequence
- Example with  $w := 5$

Query: ACTTGCTAAC

$L_1$  {  
ACTTG  
CTTGC  
TTGCT  
TGCTA  
CTAAC



# BLAST: seeding (2)

- For each subword  $W_i \in L_1$  build a *neighborhood*  $N_i$  which includes “similar” subwords
- Similarity is defined via a substitution matrix
  - For proteins, BLOSUM matrices can be used
  - For DNA, +2 for a match and -3 for mismatch (or +5/-4)
- Only subwords with similarity score above a threshold  $T$  are added into the neighborhood


# BLAST: seeding (2)

- For each subword  $W_i \in L_1$  build a *neighborhood*  $N_i$  which includes “similar” subwords
- Similarity is defined via a substitution matrix
  - For proteins, BLOSUM matrices can be used
  - For DNA, +2 for a match and -3 for mismatch (or +5/-4)
- Only subwords with similarity score above a threshold  $T$  are added into the neighborhood

$$T := 4$$


$W_i \rightarrow$  T T G C T  
T T G A T

+2 +2 +2 -3 +2 = 5 > 4



$W_i \rightarrow$  T T G C T  
T G C C T

+2 -3 -3 +2 +2 = 0 < 4



# BLAST: seeding (3)

- Combine all subwords' neighborhoods of a single query sequence

$$L_2 = \cup N_i$$

- Build a final list by adding the subwords themselves

$$L = L_1 \cup L_2$$

- Scan the database for **exact matches** of subwords in  $L$

# BLAST: seeding (3)

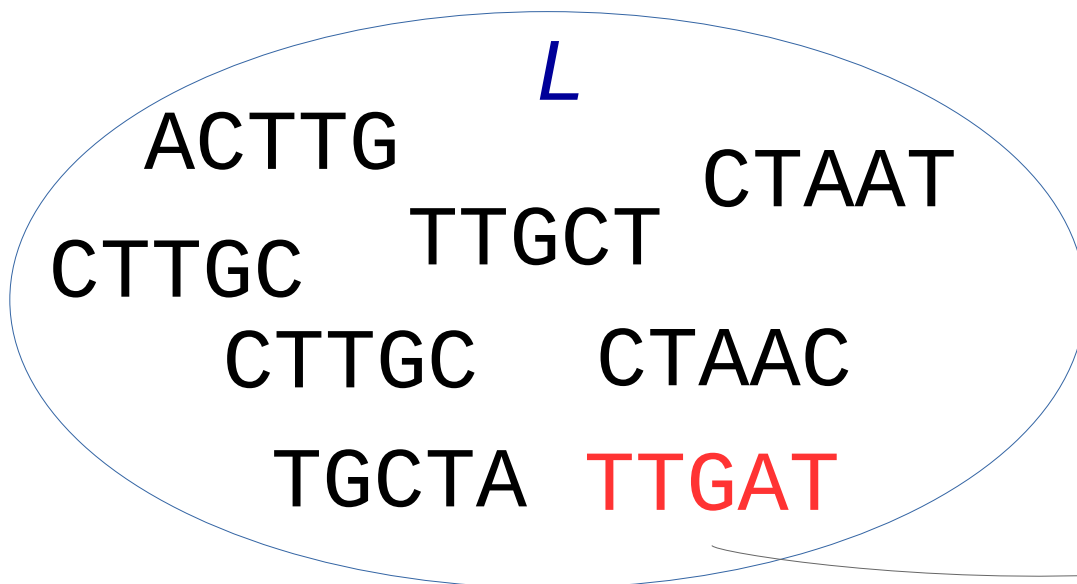
- Combine all subwords' neighborhoods of a single query sequence

$$L_2 = \cup N_i$$

- Build a final list by adding the subwords themselves

$$L = L_1 \cup L_2$$

- Scan the database for **exact matches** of subwords in  $L$



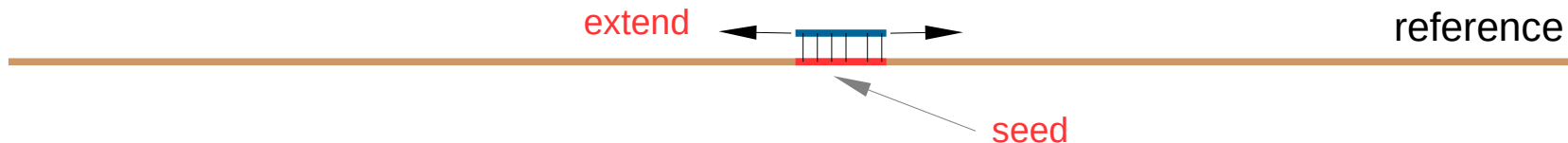
Database sequence:

AGCTATTGATGACTG

seed

# BLAST: extension

- Try to **extend** the alignment to the left and to the right from the seed
- **Stop** if the current total score drops by more than  $X$  compared to the maximum seen so far
- **Trim** alignment back to the maximum score



# BLAST: extension (2)

- Example with  $X := 3$

DB sequence: A G C T A **T T G A T** G A C T G

Query: C A C **T T G C T** A A C

seed

+2 +2 +2 -3 +2

Current score: 5                      Max score: 5                      Diff: 0



# BLAST: extension (2)

- Example with  $X := 3$

DB sequence: A G C T A **T T G A T** G A C T G

Query: C A C **T T G C T** A A C

-3 -3 +2 +2 +2 -3 +2

Current score: -1                      Max score: 5                      Diff: 6



# BLAST: extension (2)

- Example with  $X := 3$

DB sequence: A G C T A **T T G A T** G A C T G

Query: C A C **T T G C T** A A C

seed

+2 +2 +2 -3 +2

Current score: 5                      Max score: 5                      Diff: 0

# BLAST: extension (2)

- Example with  $X := 3$

DB sequence: A G C T A **T T G A T** G A C T G

Query: C A C **T T G C T** A A C

seed

+2 +2 +2 -3 +2 -3

Current score: 2                      Max score: 5                      Diff: 3

# BLAST: extension (2)

- Example with  $X := 3$

DB sequence: A G C T A **T T G A T** G A C T G

Query: C A C **T T G C T** A A C

seed

+2 +2 +2 -3 +2 -3 +2

Current score: 4                      Max score: 5                      Diff: 1

# BLAST: extension (2)

- Example with  $X := 3$

DB sequence: A G C T A **T T G A T** G A C T G

Query: C A C **T T G C T** A A C

seed

+2 +2 +2 -3 +2 -3 +2 +2

Current score: 6                      Max score: 6                      Diff: 0

# BLAST: extension (2)

- Example with  $X := 3$

DB sequence: A G C T A **T T G A T** G A C T G

Query: C A C **T T G C T** A A C

seed

					+2	+2	+2	-3	+2	-3	+2	+2
--	--	--	--	--	----	----	----	----	----	----	----	----

Current score: 6                      Max score: 6

High-scoring segment pair (HSP):

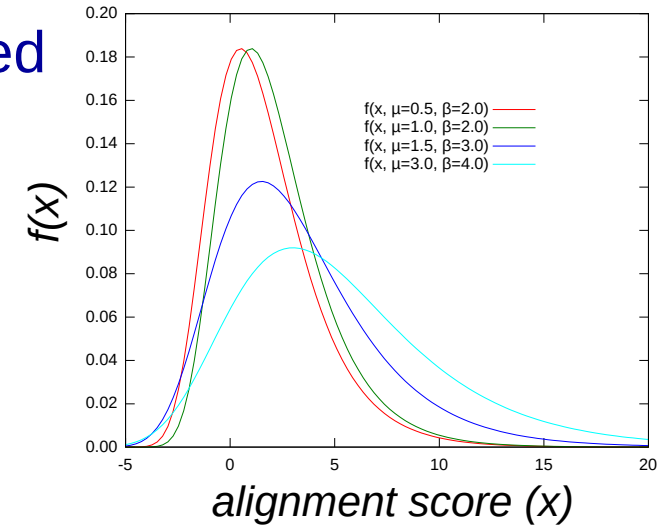
A	G	C	T	A	<b>T</b>	<b>T</b>	<b>G</b>	<b>A</b>	<b>T</b>	<b>G</b>	<b>A</b>	<b>C</b>	T	G	
					<b>C</b>	<b>A</b>	<b>C</b>	<b>T</b>	<b>T</b>	<b>G</b>	<b>C</b>	<b>T</b>	<b>A</b>	<b>A</b>	<b>C</b>

Total score: +2 +2 +2 -3 +2 -3 +2 +2 = 6

# BLAST: evaluation

- Given  $S=6$  → are sequences **biologically related** or are they similar just **by chance**?
- It was shown that Smith-Waterman local alignment scores between two **random** sequences follow the **Gumbel extreme value distribution (EVD)**

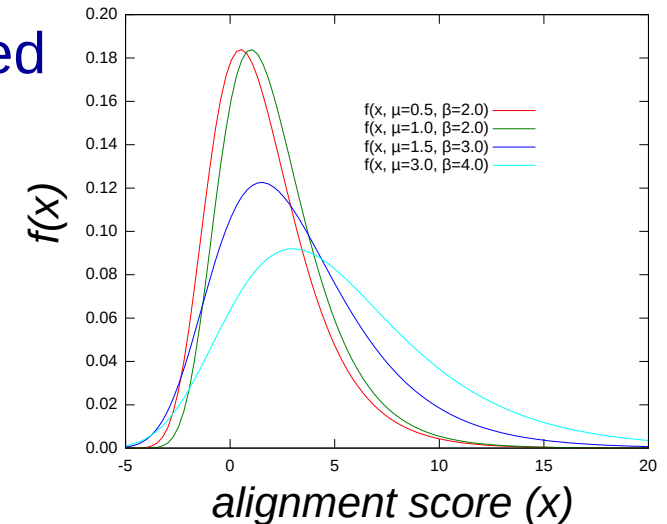
EVD prob. density function



# BLAST: evaluation

- Given  $S=6$  → are sequences **biologically related** or are they similar just **by chance**?
- It was shown that Smith-Waterman local alignment scores between two **random** sequences follow the **Gumbel extreme value distribution (EVD)**

EVD prob. density function



- Probability  $p$  of observing a score  $S$  equal to or greater than  $x$  *by chance* is given by the equation

$$p(S \geq x) = 1 - \exp(-e^{-\lambda(x-\mu)})$$

- Parameters  $\lambda$  and  $\mu$  depend on the substitution matrix, gap penalties, sequence length and nucleotide frequencies

# BLAST: evaluation (2)

- Instead of a probability, BLAST reports a so-called expectation value (**E-value**), which takes into account the database size  $d$ :

$$E \approx 1 - e^{-p(S>x)d}$$

- The E-value is the expected number of times that an **unrelated** database sequence would obtain a score  $S$  higher than  $x$  *by chance*
- Hence, for biologically related sequences, E-value has to be **very small**



# BLAST web service

## Basic Local Alignment Search Tool

BLAST finds regions of similarity between biological sequences. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance. [Learn more](#)

NEWS

ClusteredNR database on BLAST+

The ClusteredNR database is now available for BLAST+

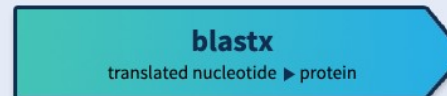
Thu, 24 Aug 2023

[More BLAST news...](#)

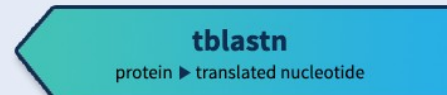
## Web BLAST



**Nucleotide BLAST**  
nucleotide ▶ nucleotide



**blastx**  
translated nucleotide ▶ protein



**tblastn**  
protein ▶ translated nucleotide



**Protein BLAST**  
protein ▶ protein

## BLAST Genomes

Search

[Human](#)

[Mouse](#)

[Rat](#)

[Microbes](#)

## Standalone and API BLAST



Download BLAST

Get BLAST databases and executables



Use BLAST API

Call BLAST from your application



Use BLAST in the cloud

Start an instance at a cloud provider

## Specialized searches

SmartBLAST



Primer-BLAST



Global Align



CD-search



<https://blast.ncbi.nlm.nih.gov/>

# BLAST web service: query

Standard Nucleotide BLAST

blastn blastp blastx tblastn tblastx

BLASTN programs search nucleotide databases using a nucleotide query. more... [Reset page](#) [Bookmark](#)

### Enter Query Sequence

Enter accession number(s), gi(s), or FASTA sequence(s) [Clear](#) Query subrange [?](#)

```
GTTGAACTAATCTGAACCATCCTACCGCTATTGTCTAGTCTGCTTGCCCT
CCCCTCCCTCCAAATCCTCTACATAATAGACGAAATCGACGAACCTGATCTCA
CCCTAAAAGCCATCGGACACCAATGATACTGAACAAAAAAAAAACAGACTT
CAAGGACCTCTCATTTGACTCCTACATAACCCCAACACAGACCTCCCCTA
```

From  To

Or, upload file  No file selected. [?](#)

Job Title   
Enter a descriptive title for your BLAST search [?](#)

Align two or more sequences [?](#)

### Choose Search Set

Database  Standard databases (nr etc.):  rRNA/ITS databases  Genomic + transcript data  Betacoronavirus

Experimental databases [Try experimental taxonomic nt data](#)  
For more info see [What are taxonomic nt data](#)

Nucleotide collection (nr/nt)  [?](#)

Organism Optional   exclude [?](#)  
Enter organism common name, binomial, or tax id. Only 20 top taxa will be shown [?](#)

Exclude Optional  Models (XM/XP)  Uncultured/environmental sample sequences

Limit to Optional  Sequences from type material

Entrez Query Optional  [YouTube](#) [Create](#)  
Enter an Entrez query to limit search [?](#)

### Algorithm parameters

[Restore default search parameters](#)

#### General Parameters

Max target sequences  [?](#)  
Select the maximum number of aligned sequences to display [?](#)

Short queries  Automatically adjust parameters for short input sequences [?](#)

Expect threshold  [?](#)

Word size  [?](#)

Max matches in a query range  [?](#)

#### Scoring Parameters

Match/Mismatch Scores  [?](#)

Gap Costs Existence: 5 Extension: 2 [?](#)

#### Filters and Masking

Filter  Low complexity regions [?](#)  
 Species-specific repeats for:  [?](#)

Mask  Mask for lookup table only [?](#)  
 Mask lower case letters [?](#)

**BLAST** Search database Nucleotide collection (nr/nt) using Megablast (Optimize for highly similar sequences)  
 Show results in a new window

# BLAST web service: results

[Descriptions](#) | [Graphic Summary](#) | [Alignments](#) | [Taxonomy](#)

**Sequences producing significant alignments** Download ▾ Select columns ▾ Show 100 ▾ ?

select all 100 sequences selected 
[GenBank](#) | [Graphics](#) | [Distance tree of results](#) | [MSA Viewer](#)

	Description	Scientific Name	Max Score	Total Score	Query Cover	E value	Per. Ident	Acc. Len	Accession
<input checked="" type="checkbox"/>	<a href="#">Gallus gallus isolate DC1-1 mitochondrion, complete genome</a>	<a href="#">Gallus gallus</a>	180	180	92%	9e-41	86.43%	16785	<a href="#">MK163565.1</a>
<input checked="" type="checkbox"/>	<a href="#">Gallus gallus isolate DC3 mitochondrion, complete genome</a>	<a href="#">Gallus gallus</a>	180	180	92%	9e-41	86.43%	16785	<a href="#">MK163564.1</a>
<input checked="" type="checkbox"/>	<a href="#">Gallus gallus isolate DT3 mitochondrion, complete genome</a>	<a href="#">Gallus gallus</a>	180	180	92%	9e-41	86.43%	16785	<a href="#">MK163562.1</a>
<input checked="" type="checkbox"/>	<a href="#">Gallus gallus isolate SG2 mitochondrion, complete genome</a>	<a href="#">Gallus gallus</a>	180	180	92%	9e-41	86.43%	16784	<a href="#">MK163561.1</a>
<input checked="" type="checkbox"/>	<a href="#">Gallus gallus isolate SG6 mitochondrion, complete genome</a>	<a href="#">Gallus gallus</a>	180	180	92%	9e-41	86.43%	16785	<a href="#">MK163560.1</a>
<input checked="" type="checkbox"/>	<a href="#">Gallus gallus isolate Y4 mitochondrion, complete genome</a>	<a href="#">Gallus gallus</a>	180	180	92%	9e-41	86.43%	16785	<a href="#">MK163559.1</a>

[Descriptions](#) | [Graphic Summary](#) | **[Alignments](#)** | [Taxonomy](#)

Alignment view Pairwise ▾  CDS feature ? [Restore defaults](#) Download ▾

100 sequences selected ?

[Download](#) ▾ [GenBank](#) [Graphics](#) 
 ▾ Next ▲ Previous ◀ Descriptions

**Gallus gallus isolate DC1-1 mitochondrion, complete genome**

Sequence ID: [MK163565.1](#) Length: 16785 Number of Matches: 1

Range 1: 8517 to 8656 [GenBank](#) [Graphics](#) 
 ▾ Next Match ▲ Previous Match

Score	Expect	Identities	Gaps	Strand
180 bits(199)	9e-41	121/140(86%)	19/140(13%)	Plus/Plus

```

Query 1      GTTGAAC TAATCTGAACCATCTACCCGCTATTGTCCTAGTCCTGCTTGCCTCCCTCC
Sbjct 8517    GTTGAAC TAATCTGAACCATCTACCCGCTATTGTCCTAGTCCTGCTTGCCTCCCTCC 8576
Query 61     CTCCAA-----AAATCGACGAACTGATCTCACCTAAAAGCCATC 101
Sbjct 8577    CTCCAAATCCTCTACATAATAGACGAAATCGACGAACTGATCTCACCTAAAAGCCATC 8636
Query 102    GGACACCAATGATACTGAAC 121
Sbjct 8637    GGACACCAATGATACTGAAC 8656
  
```

# Beyond BLAST

- BLAST implicitly assumes that:
  - The number of queries is **relatively small** (compared to DB size)
  - **Pair-wise comparison** is sufficient to detect relatedness
  - We are (also) interested in **remotely-related** sequences
- What if it's not the case?
  - Metagenomics: Querying **millions** of reads against a **fixed-size** reference database
  - Protein families: Find sequences similar to **multiple** queries

# Beyond BLAST

- What if it's not the case?
  - Metagenomics: Querying **millions** of reads against a **fixed-size** reference database
  - Protein families: Find sequences similar to **multiple** queries
- Use **database pre-processing**
  - Seed index: BLAT (*Kent 2002*), MEGABLAST (*Morgulis 2008*), UBLAST
  - Unique words index: USEARCH (*Edgar 2010*)
  - *k*-mer frequencies: RDP Classifier (Wang et al. 2007)
  - Profile HMMs: HMMER (Eddy 2009)
- Trade **query** time for **pre-processing** time

# Today's agenda

- Search methods for biological sequences
  - Public sequence databases
  - BLAST algorithm
  - Alternative search algorithms
- **Genome assembly**
  - *De novo* assembly
    - Overlap graphs
    - De Bruijn graphs
  - By-reference assembly

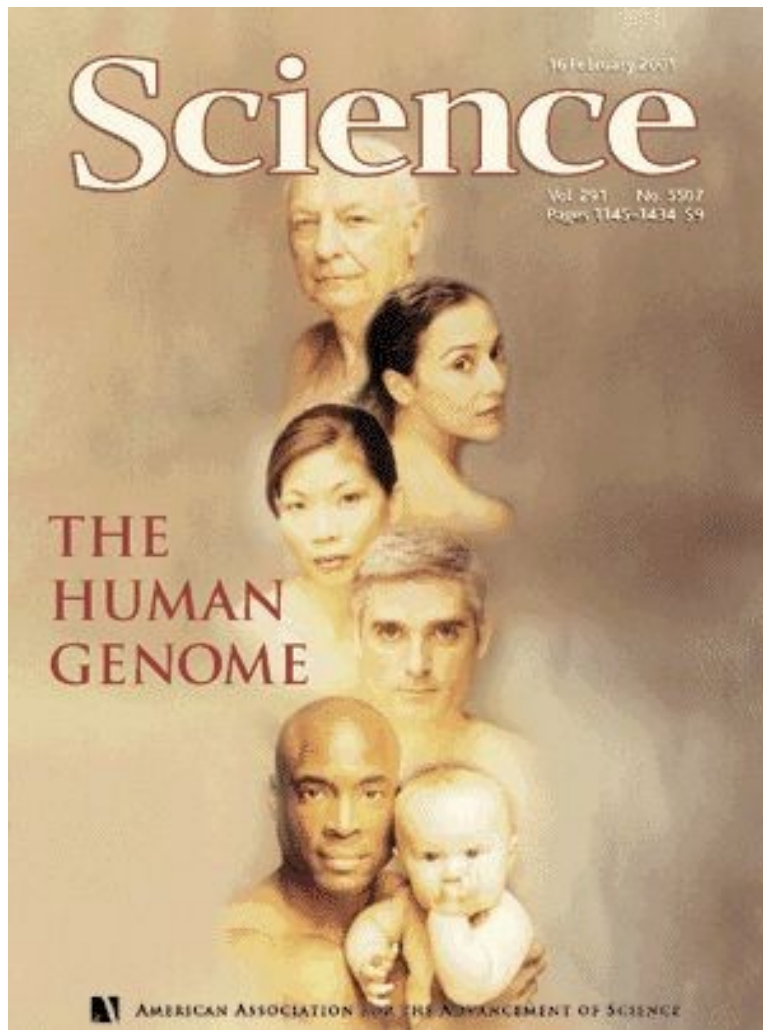
# *De novo* vs. by-reference assembly

- Genome assembly is a process of reconstructing the original DNA sequence from its factors (i.e., short reads)
- *De novo*: exploit read overlaps to build a (novel) genome sequence *from scratch*
  - e.g., assembling the first human genome back in 2000
- *By-reference*: map reads to the known *reference sequence* – usually genome of the same species or a close relative/close relatives
  - e.g., getting your personal genome nowadays
- In terms of time and space complexity, *de novo* assembly is orders of magnitude slower and more memory intensive than mapping assembly



# Human genome: are we there yet?

2001



2022



## RESEARCH ARTICLE

### HUMAN GENOMICS

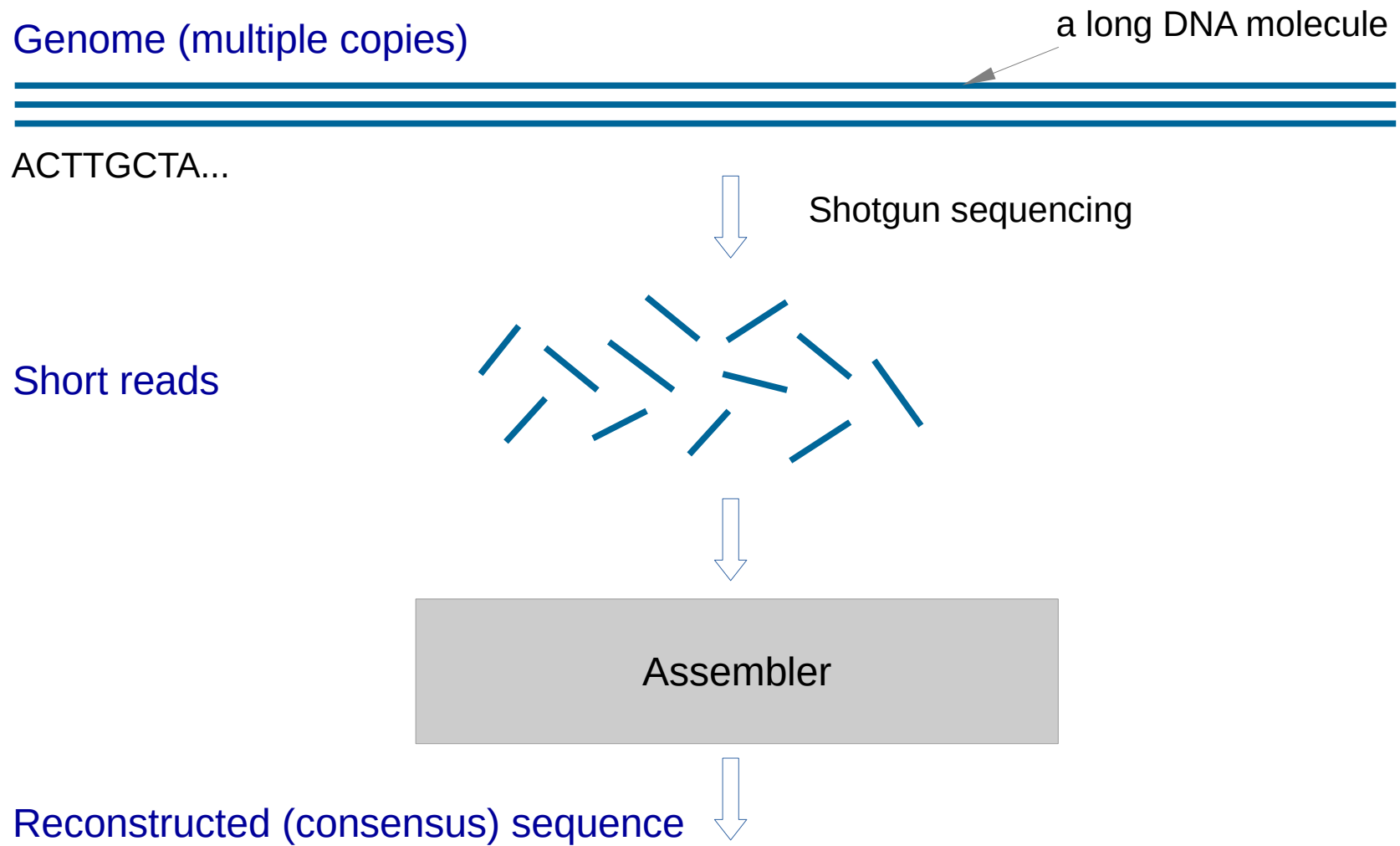
## The complete sequence of a human genome

Sergey Nurk<sup>1†</sup>, Sergey Koren<sup>1†</sup>, Arang Rhie<sup>1†</sup>, Mikko Rautiainen<sup>1†</sup>, Andrey V. Bzikadze<sup>2</sup>, Al Mitchell R. Vollger<sup>4</sup>, Nicolas Altemose<sup>5</sup>, Lev Uralsky<sup>6,7</sup>, Ariel Gershman<sup>8</sup>, Sergey Aganez Savannah J. Hoyt<sup>10</sup>, Mark Diekhans<sup>11</sup>, Glennis A. Logsdon<sup>4</sup>, Michael Alonge<sup>9</sup>, Stylianos E. A. Matthew Barber<sup>13</sup>, Gerard C. Bouffard<sup>14</sup>, Shalini V. Brock<sup>14</sup>, Gina M. Collier<sup>15</sup>, Mao Ch.

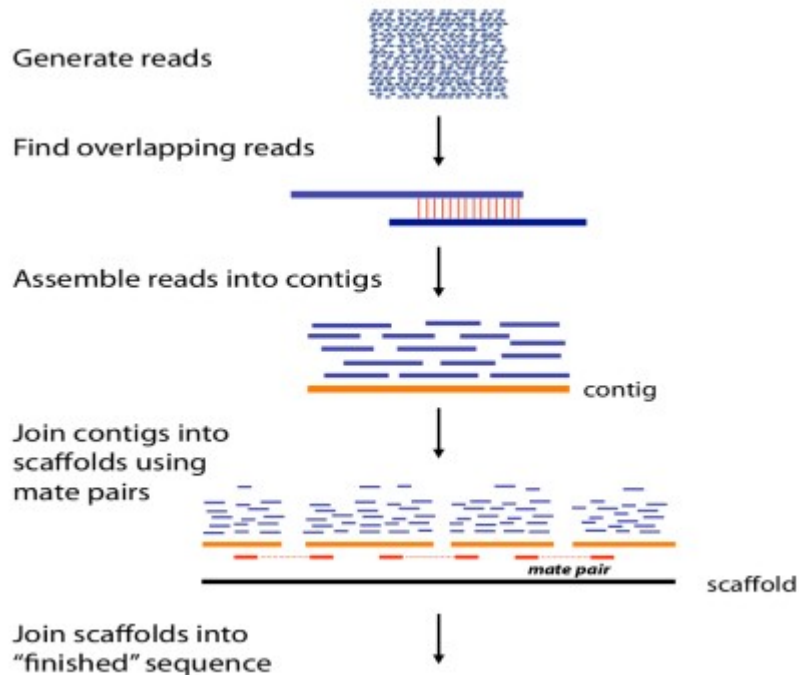


# Genome assembly

- A very simplified workflow:



# De novo assembly: overview



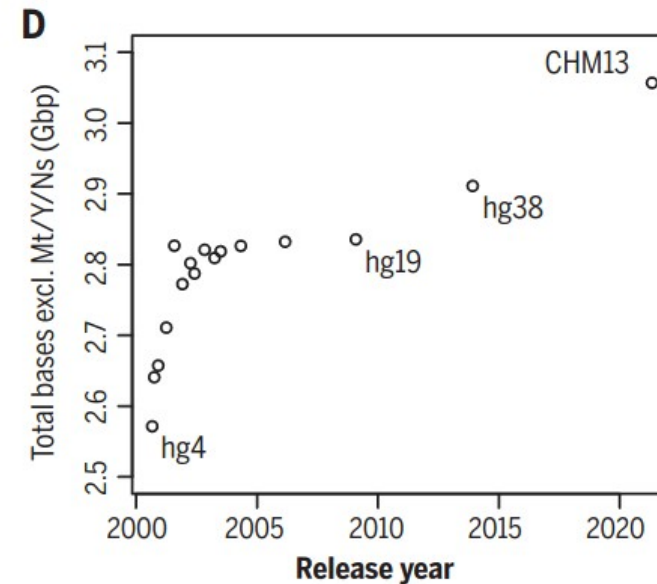
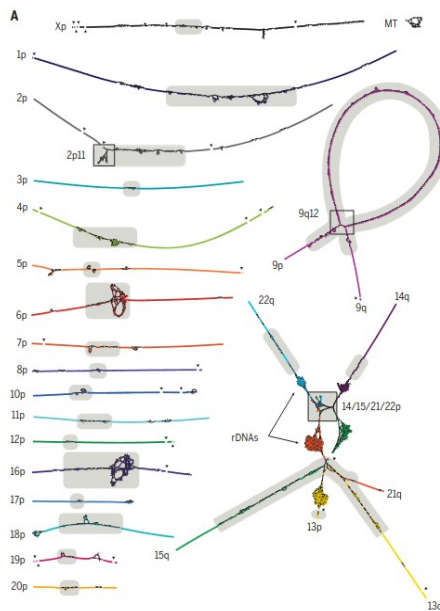
ACGTGCTCCCTTTAGAGAGGCTTCCAAT...

- Assemble reads into **contigs** using overlaps between them
  - **overlap graphs** or **de Bruijn graphs**
- Use mate pairs to combine contigs into **scaffolds**
  - Information from paired-end reads allows to determine contigs' order and orientation
- Joining scaffolds is a manual step
  - Using optical gene maps or other coarse-grained structural information
  - Often impossible due to large repeats

- Length of contigs and scaffolds are important metrics of assembly quality

# Human Genome assembly

STATISTICS	2014 GRCH38	2022 T2T-CHM13	DIFFERENCE (±%)
<b>Summary</b>			
<b>Assembled bases (Gbp)</b>	2.92	3.05	+4.5
<b>Unplaced bases (Mbp)</b>	11.42	0	-100.0
<b>Gap bases (Mbp)</b>	120.31	0	-100.0
<b>Number of contigs</b>	949	24	-97.5
<b>Contig NG50 (Mbp)</b>	56.41	154.26	+173.5
<b>Number of issues</b>	230	46	-80.0
<b>Issues (Mbp)</b>	230.43	8.18	-96.5



# Overlap graphs

- Represent each read as a graph **node**
- Compute all pair-wise alignments between reads and represent overlaps as graph **edges**

# Overlap graphs

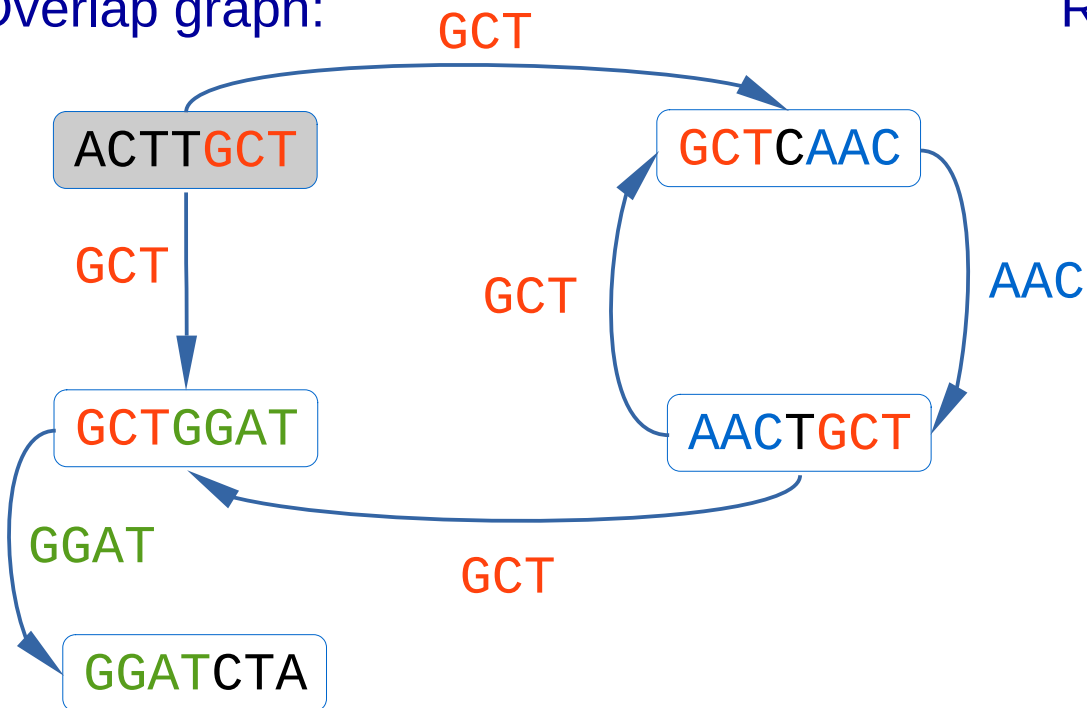
- Represent each read as a graph **node**
- Compute all pair-wise alignments between reads and represent overlaps as graph **edges**
- Walking along the **Hamiltonian path** (visit every node *exactly once*), we can reconstruct the original genomic sequence
  - For a **circular** genome, we can start from any node
  - For a **linear** genome, we should ideally start from a node with no inbounding edges (in practice, there could be none/multiple such nodes → apply heuristics)

# Overlap graphs: example

Genome: **ACTTGCTCAACTGCTGGATCTA**

Reads: {  
ACTTGCT  
AACTGCT  
GCTCAAC  
GCTGGAT  
GGATCTA

Overlap graph:



Reconstructed sequence:

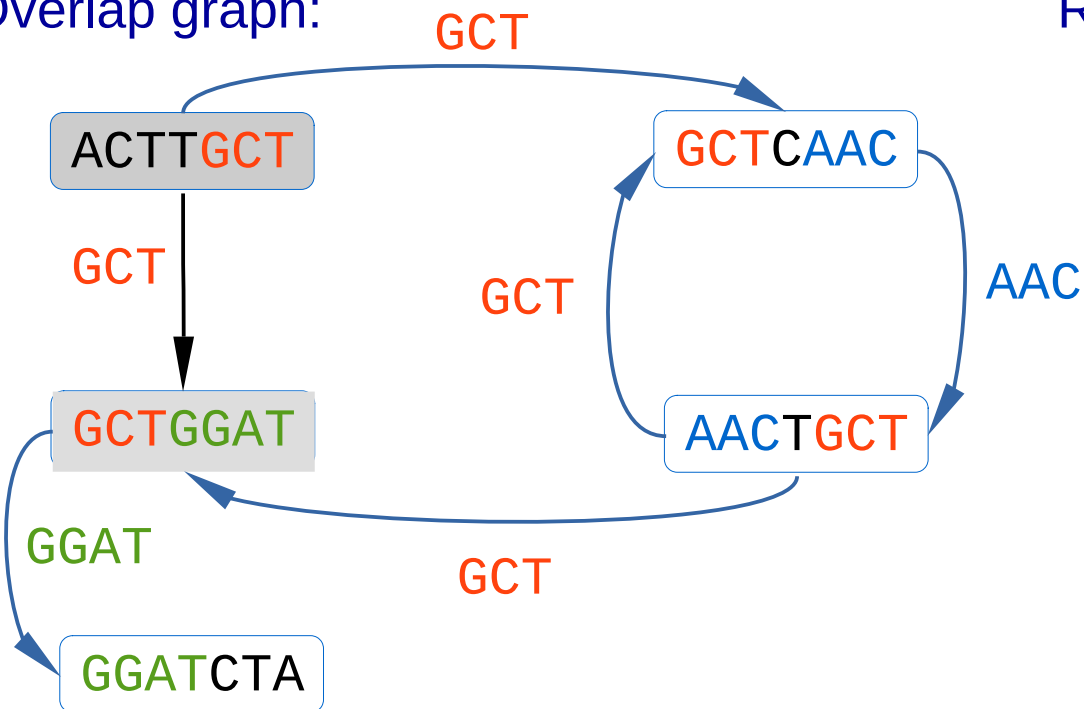
ACTTGCT

# Overlap graphs: example

Genome: **ACTTGCTCAACTGCTGGATCTA**

Reads: {  
ACTTGCT  
AACTGCT  
GCTCAAC  
GCTGGAT  
GGATCTA

Overlap graph:



Reconstructed sequence:

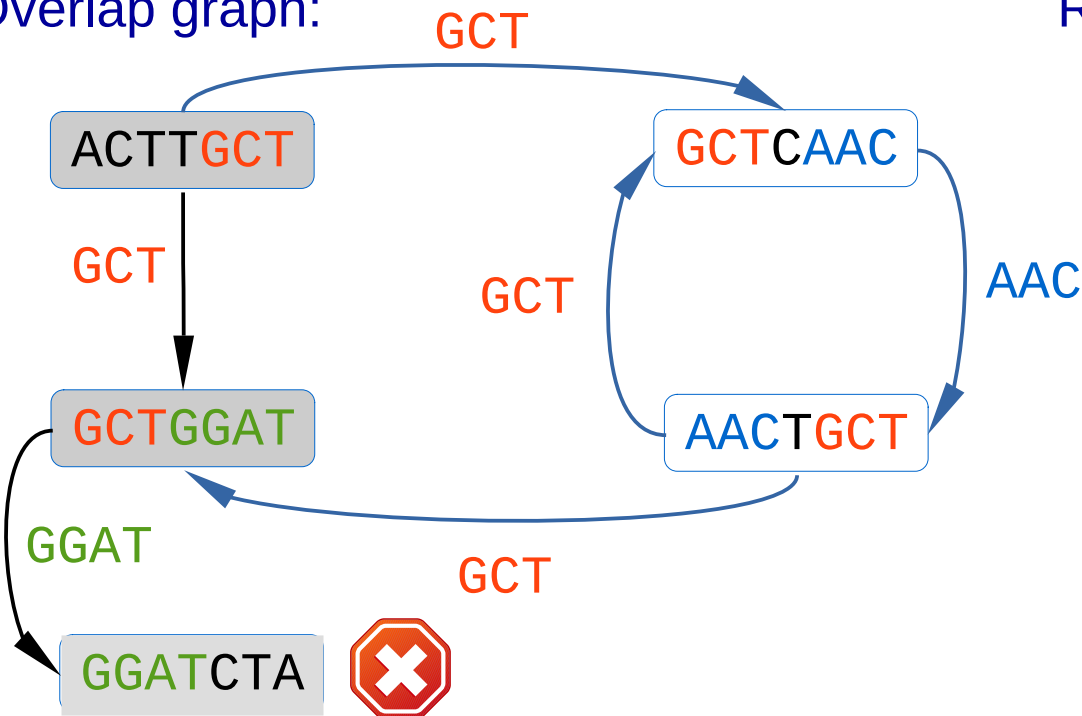
ACTTGCT  
ACTTGCTGGAT

# Overlap graphs: example

Genome: **ACTTGCTCAACTGCTGGATCTA**

Reads: {  
ACTTGCT  
AACTGCT  
GCTCAAC  
GCTGGAT  
GGATCTA

Overlap graph:



Reconstructed sequence:

ACTTGCT  
ACTTGCTGGAT  
ACTTGCTGGATCTA

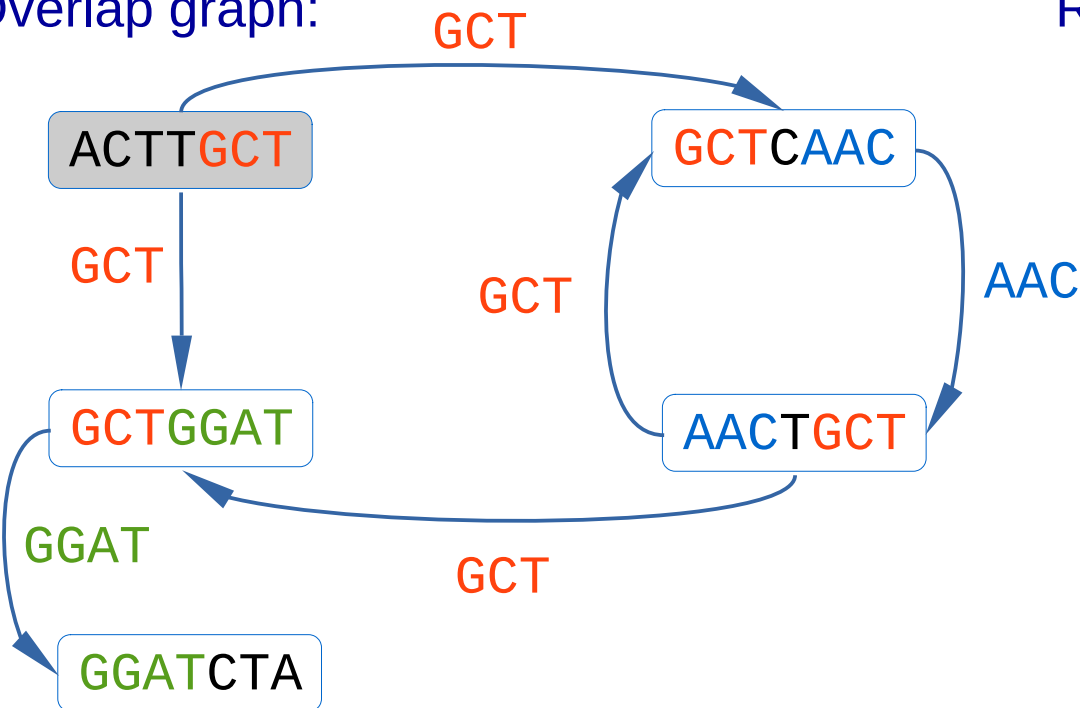


# Overlap graphs: example

Genome: **ACTTGCTCAACTGCTGGATCTA**

Reads: {  
ACTTGCT  
AACTGCT  
GCTCAAC  
GCTGGAT  
GGATCTA

Overlap graph:



Reconstructed sequence:

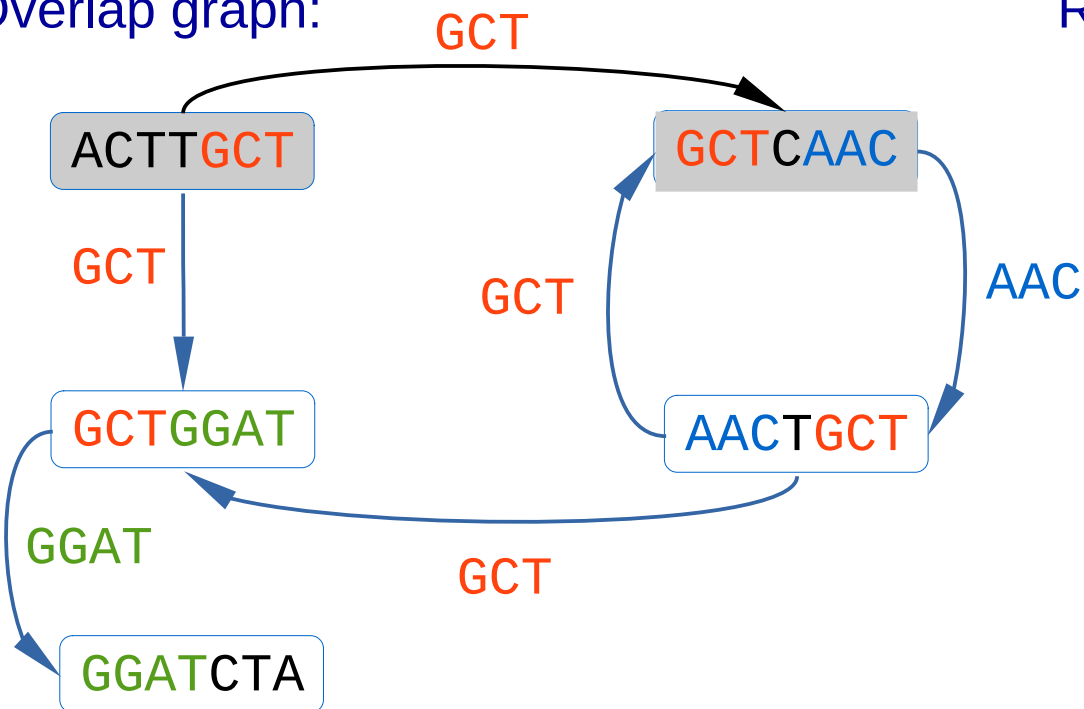
ACTTGCT  
ACTTGCTGGAT  
ACTTGCTGGATCTA  
ACTTGCTGGAT  
ACTTGCT

# Overlap graphs: example

Genome: **ACTTGCTCAACTGCTGGATCTA**

Reads: {  
ACTTGCT  
AACTGCT  
GCTCAAC  
GCTGGAT  
GGATCTA

Overlap graph:



Reconstructed sequence:

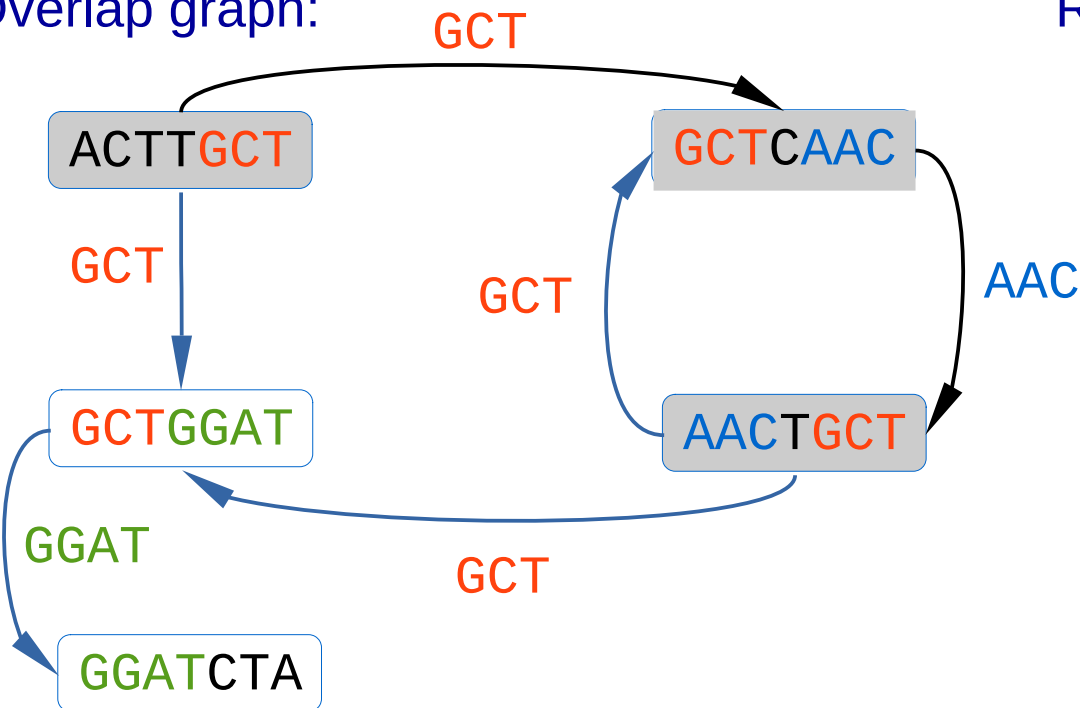
ACTTGCT  
ACTTGCTGGAT  
ACTTGCTGGATCTA  
ACTTGCTGGAT  
ACTTGCT  
ACTTGCTCAAC

# Overlap graphs: example

Genome: **ACTTGCTCAACTGCTGGATCTA**

Reads: {  
 ACTTGCT  
 AACTGCT  
 GCTCAAC  
 GCTGGAT  
 GGATCTA

Overlap graph:



Reconstructed sequence:

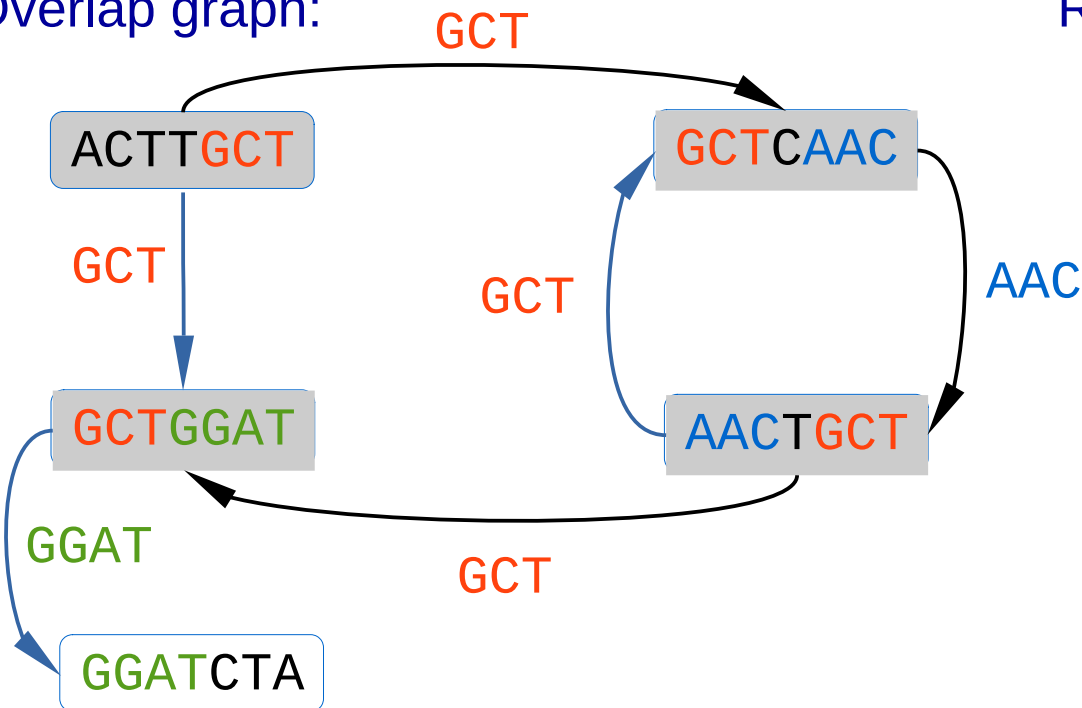
ACTTGCT  
 ACTTGCTGGAT  
 ACTTGCTGGATCTA  
 ACTTGCTGGAT  
 ACTTGCT  
 ACTTGCTCAAC  
 ACTTGCTCAACTGCT

# Overlap graphs: example

Genome: **ACTTGCTCAACTGCTGGATCTA**

Reads: {  
 ACTTGCT  
 AACTGCT  
 GCTCAAC  
 GCTGGAT  
 GGATCTA

Overlap graph:



Reconstructed sequence:

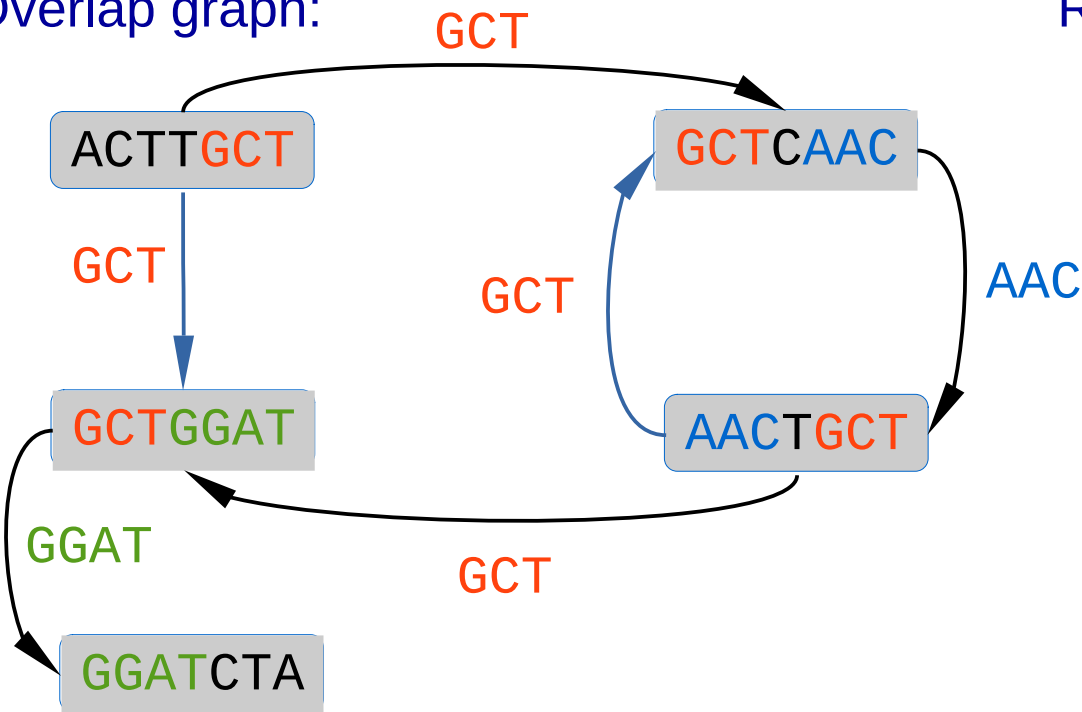
ACTTGCT  
 ACTTGCTGGAT  
 ACTTGCTGGATCTA  
 ACTTGCTGGAT  
 ACTTGCT  
 ACTTGCTCAAC  
 ACTTGCTCAACTGCT  
 ACTTGCTCAACTGCTGGAT

# Overlap graphs: example

Genome: **ACTTGCTCAACTGCTGGATCTA**

Reads: {  
 ACTTGCT  
 AACTGCT  
 GCTCAAC  
 GCTGGAT  
 GGATCTA

Overlap graph:



Reconstructed sequence:

ACTTGCT  
 ACTTGCTGGAT  
 ACTTGCTGGATCTA  
 ACTTGCTGGAT  
 ACTTGCT  
 ACTTGCTCAAC  
 ACTTGCTCAACTGCT  
 ACTTGCTCAACTGCTGGAT  
 ACTTGCTCAACTGCTGGATCTA  
**ACTTGCTCAACTGCTGGATCTA**

# Overlap graphs: problems

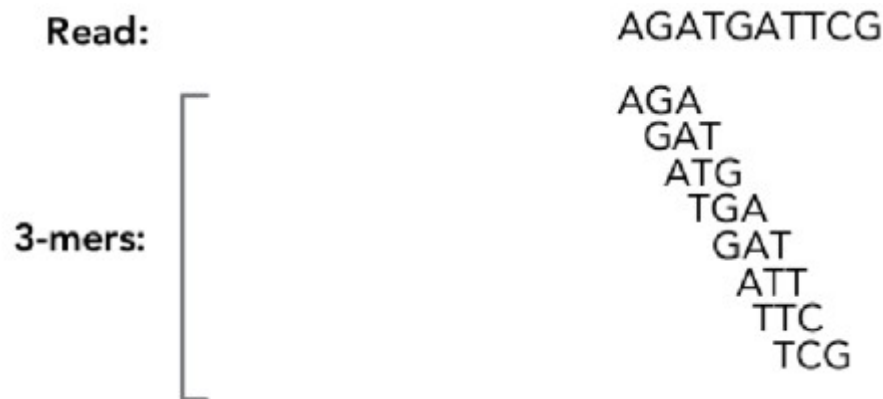
- There is no known **efficient** algorithm for finding a Hamiltonian path
- Complexity of doing the pair-wise alignments is **quadratic** in terms of number of reads

# Overlap graphs: problems

- There is no known **efficient** algorithm for finding a Hamiltonian path
- Complexity of doing the pair-wise alignments is **quadratic** in terms of number of reads
- Overlap graphs work well if there is a **small number** of reads with **significant overlap**
  - e.g., Sanger sequencing (or 3<sup>rd</sup> gen. → later)
- With millions of short NGS reads, this method becomes computationally unfeasible

# De Bruijn graphs

- To build a de Bruijn graph, each read is decomposed into series of *k*-mers
  - In real applications,  $k = 20\text{--}50$  is common
  - Optimal value of  $k$  depends on read length and error rate;  $k < \text{length of the shortest read}$
  - Some methods use multiple  $k \rightarrow \text{SPAdes (Bankevich 2012)}$



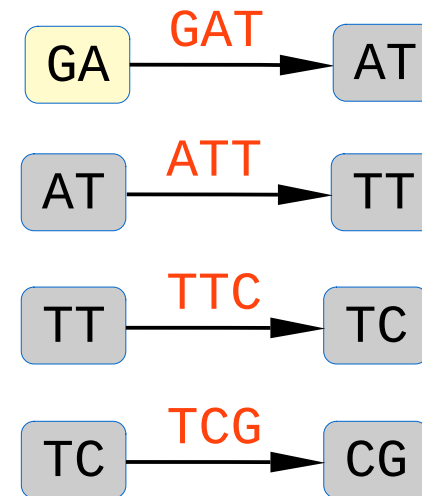
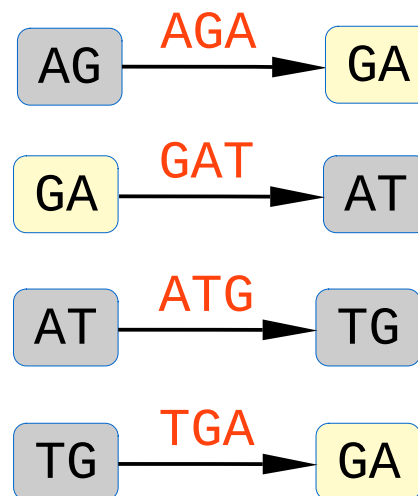


# De Bruijn graphs

- Each **unique  $k$ -mer** is represented by an **edge** of the graph
- Nodes are  **$(k-1)$ -mers**: prefix and suffix of the  **$k$ -mer** associated with the edge connecting them
- Nodes with identical  $(k-1)$ -mers are “glued together”

AGATGATTCG

AGA  
GAT  
ATG  
TGA  
GAT  
ATT  
TTC  
TCG

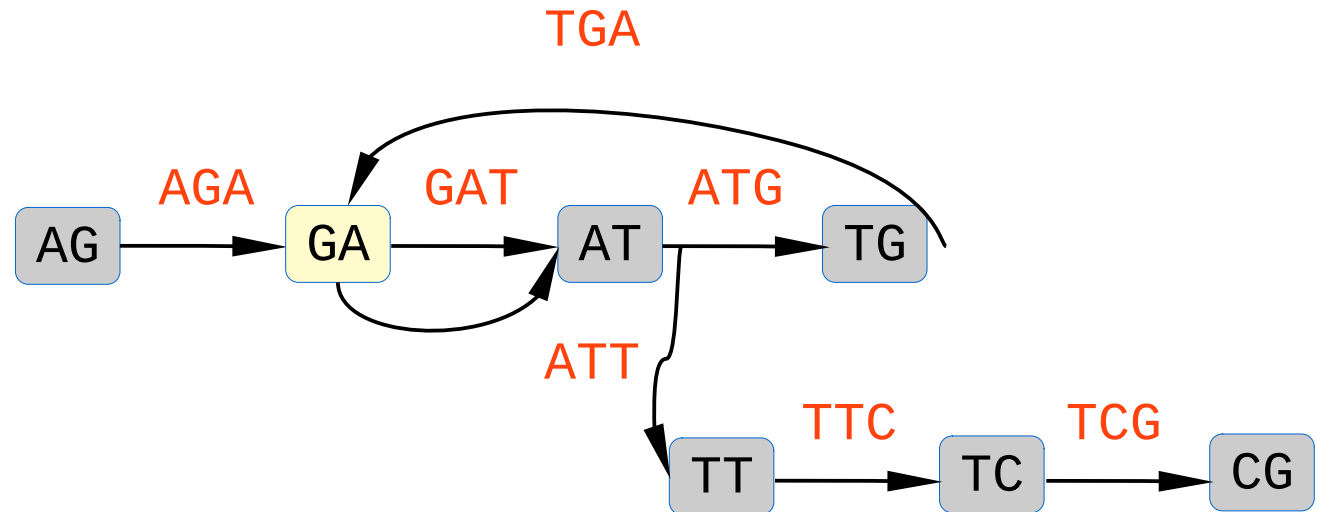


# De Bruijn graphs

- The original sequence can be reconstructed by finding an **Eulerian path** (visit each **edge** exactly once)

AGATGATTCG

AGA  
GAT  
ATG  
TGA  
GAT  
ATT  
TTC  
TCG



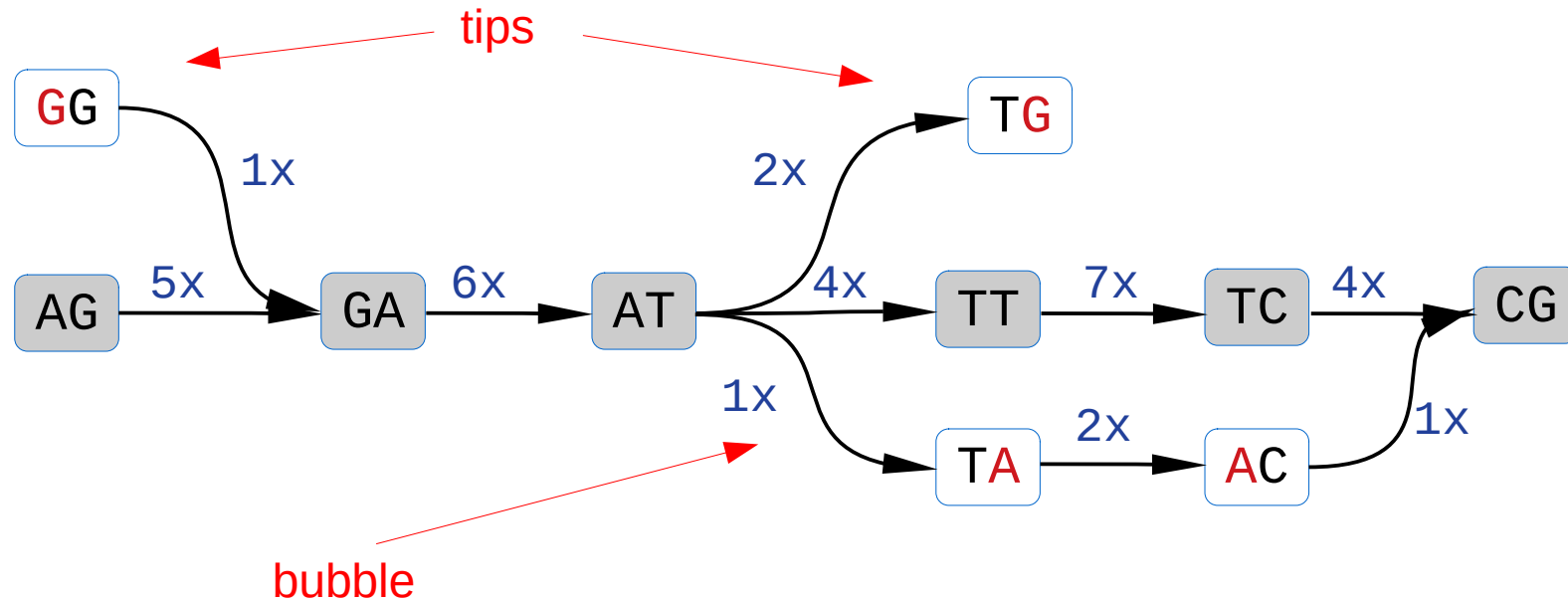
# De Bruijn graphs: advantages

- Compact representation of repeats
  - Duplicate  $(k-1)$ -mers are represented with a single node
  - Longer repeats form a single series of adjacent nodes
- Building De Bruijn graph has **linear** complexity
  - **Time:**  $O(N)$ ,  $N$  = total length of all reads
  - **Space:**  $O(\min(G,N))$ ,  $G$  = genome size
- There exists an efficient algorithm to find an Eulerian path
- For these reasons, most modern NGS assemblers use de Bruijn graphs (either explicitly or implicitly)

# De Bruijn graphs: problems

- Information loss due to k-mer extraction
  - Repeats are (even) harder to resolve
  - Some paths are not consistent with source reads
- Eulerian path **always exists** if
  - reads are error-free
  - all genome positions are evenly covered
- There can be **multiple** alternative paths due to repeats!
- Graph can be **disjoint** due to lack of (sufficient) coverage of some genome regions
- **In practice:** manual post-processing  
(see: [https://www.science.org/doi/suppl/10.1126/science.abj6987/suppl\\_file/science.abj6987\\_sm.pdf](https://www.science.org/doi/suppl/10.1126/science.abj6987/suppl_file/science.abj6987_sm.pdf) )

# De Bruijn graphs: error correction



- In reality, reads contain sequencing errors
  - Errors in the middle of the read → “bubbles”
  - Errors at the ends of the read → “tips”
  - Coverage information (=node/edge multiplicity) can be used to detect and fix errors

# Outlook: 3<sup>rd</sup> generation sequencing

- Further advancement after NGS
  - Pacific Biosciences (**PacBio SMRT**)
  - Oxford Nanopore (**MinION**)
- 3GS/PacBio vs NGS/Illumina:
  - Longer reads (20-100Kb vs. 0.5Kb)
  - Higher *raw* error rates (5-20% vs. 0.1%)  
**but:** *down to <0.1% w/consensus*
  - Lower throughput (~100 Gb vs. 8 Tb)
- Change in assembly methods
  - Back to overlap graphs, hybrid approaches
  - Resolve long repeats / bridge contigs



Image: wired.com

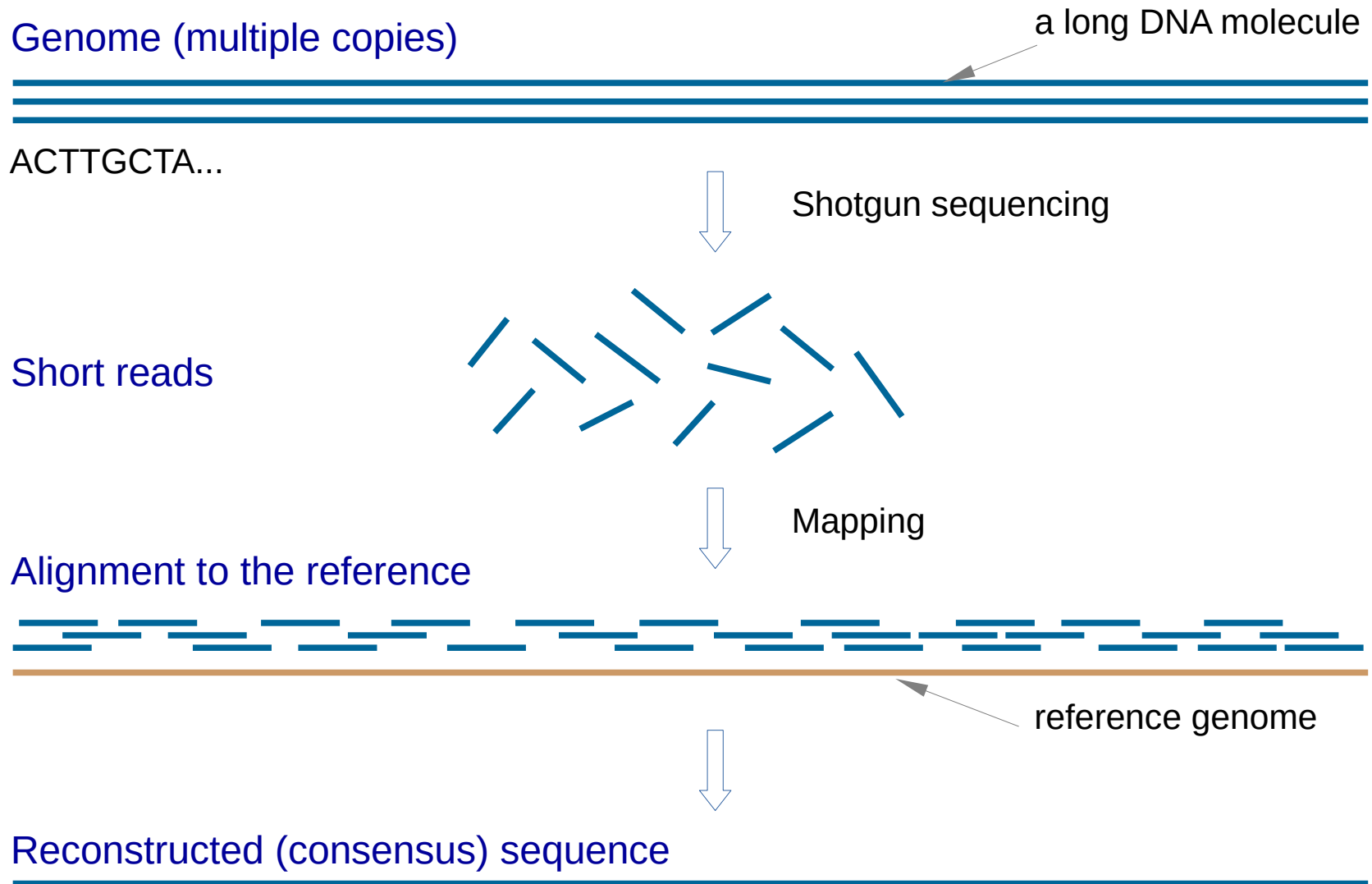


Image: PacBio

# Today's agenda

- Search methods for biological sequences
  - Public sequence databases
  - BLAST algorithm
  - Alternative search algorithms
- Genome assembly
  - *De novo* assembly
    - Overlap graphs
    - De Bruijn graphs
  - **By-reference assembly**
    - Hash indexes
    - Burrows-Wheeler transform

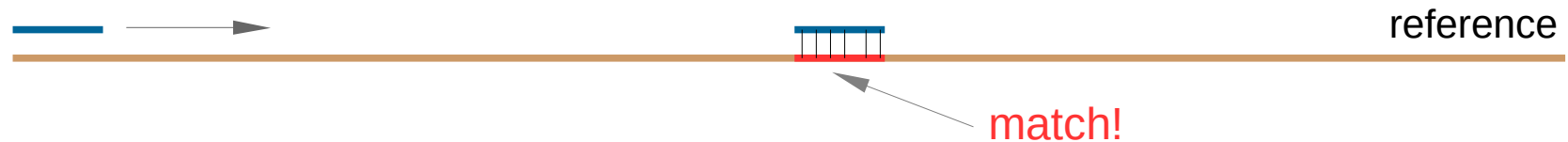
# By-reference assembly





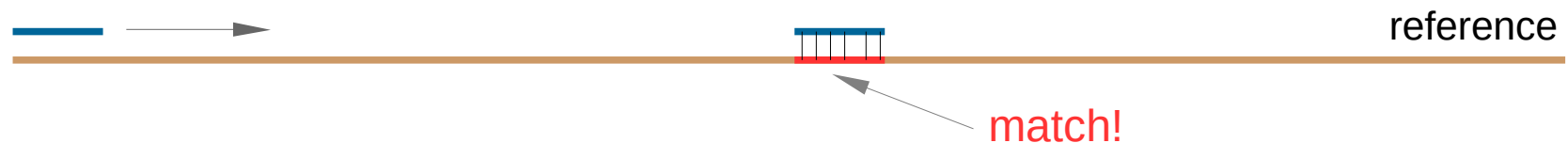
# Sliding window approach

- **Slide** each read along the reference genome and **mark** all positions where there is a match
  - if gaps are allowed, one has to resort to the classical DP algorithms like **Smith-Waterman**



# Sliding window approach

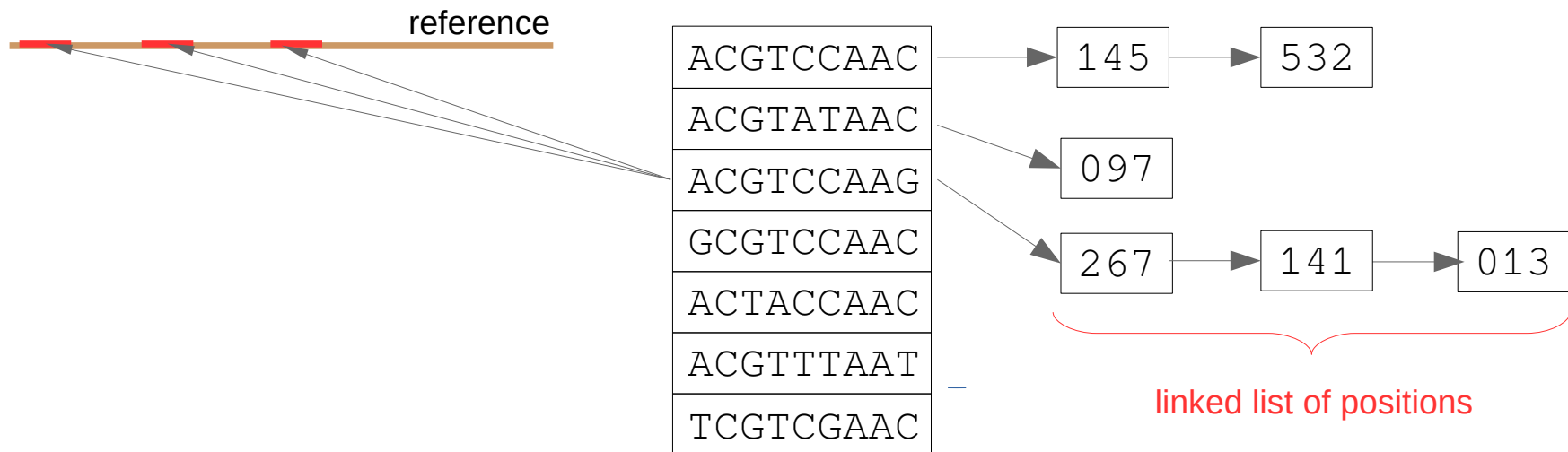
- **Slide** each read along the reference genome and **mark** all positions where there is a match
  - if gaps are allowed, one has to resort to the classical DP algorithms like **Smith-Waterman**



- **Problem:** huge complexity!
  - Recall the BLAST discussion
- Build a reference genome index using:
  - Hashing
  - Burrows-Wheeler-transform

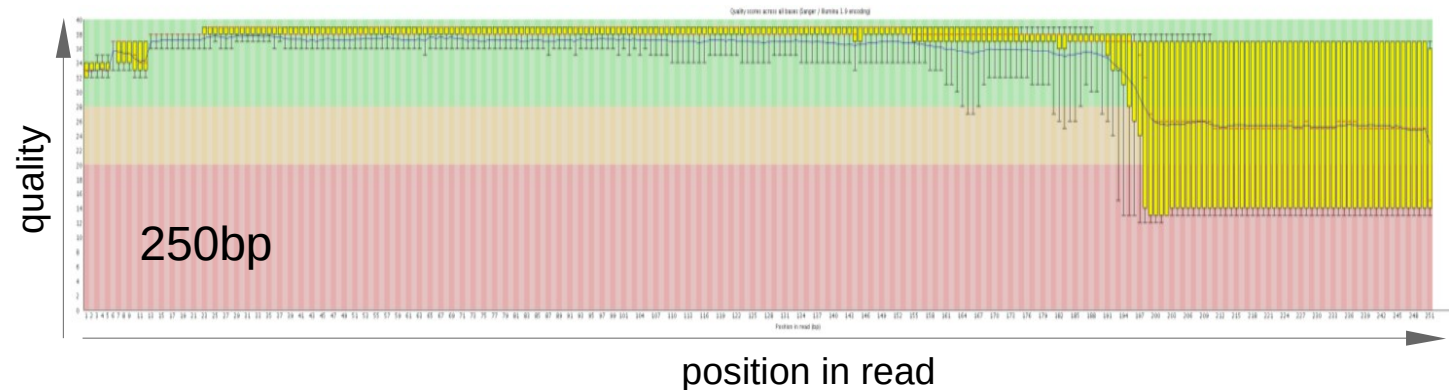
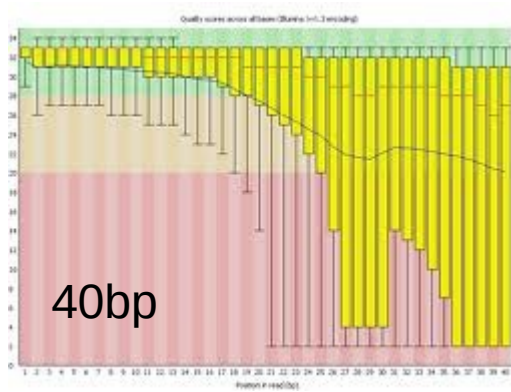
# Hashing: build genome index

- Use hash table to store positions of all *k*-mers in a genome:
  - $k \ll$  read length
  - $k := 9 \rightarrow 4^9 = 262144$  table entries (at most)



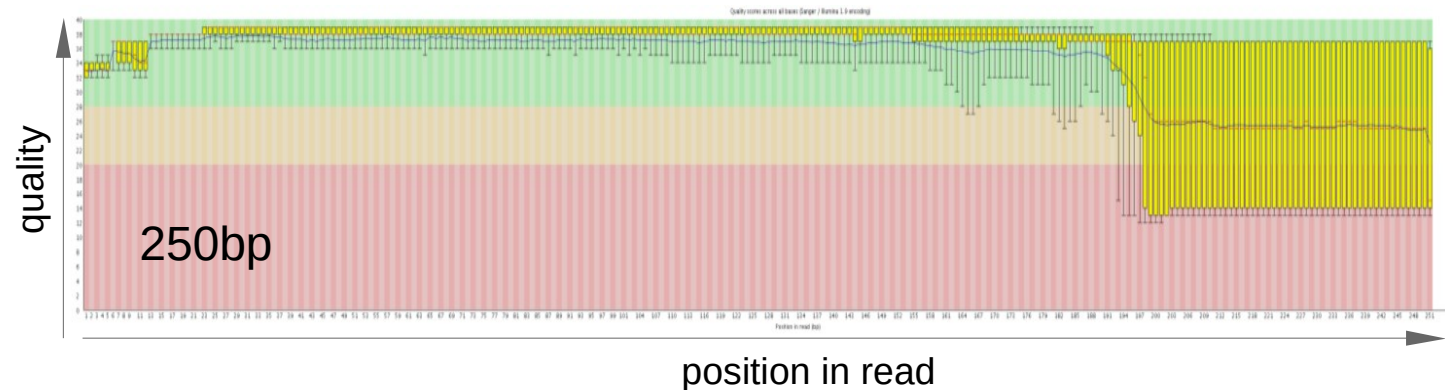
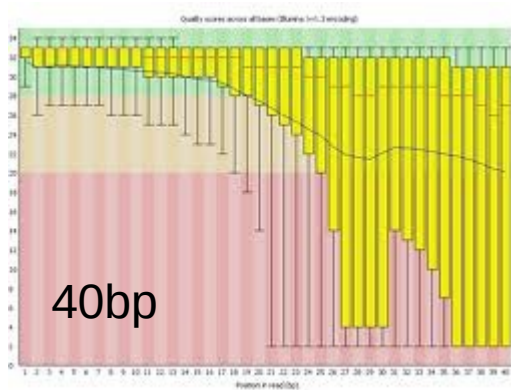
# Hashing: search strategy

- For each read, select **one “proxy”  $k$ -mer**
  - Leftmost (or middle) part is a good choice → better quality



# Hashing: search strategy

- For each read, select one “proxy” *k-mer*
  - Leftmost (or middle) part is a good choice → better quality



- Use hash table lookup to find all positions of this *k-mer* in the genome → seeds
- For each seed, try to extend the alignment (i.e., map the rest of the read) allowing for mismatches/gaps like in Smith-Waterman algorithm

# Hashing: optimizations & variants

- Inexact matches with **spaced seeds**
  - Binary mask defines positions where mismatches are allowed:  
**111001**    **ATCGGT** ↔ **ATCACT**
- Multiple *k-mers* per read
  - Require at least *n* seed matches for a mapping location to be considered
- Inverted approach: Build hash table from the reads and search for *k-mers* present in the reference

# By-reference assembly: BWT

- Most modern mapping tools rely on **Burrows-Wheeler Transform** or BWT (Burrows and Wheeler, 1994) → **Bowtie, BWA, SOAP2**
- BWT indexes offer significant improvements over hash-based methods in terms of both time and memory usage
- Also used in data compression programs such as **bzip2**

# Building a BWT

- Building a BWT consists of three steps:
  - Write down all cyclic rotations of the source string  $S$
  - Sort the rows lexicographically
  - Store the last column  $\rightarrow$   $BWT(S)$

$S$ : `mississippi$`  end-of-line marker



# Building a BWT

- Building a BWT consists of three steps:
  - Write down all cyclic rotations of the source string  $S$
  - Sort the rows lexicographically
  - Store the last column  $\rightarrow$   $BWT(S)$

$S$ : mississippi\$  end-of-line marker

rotate 

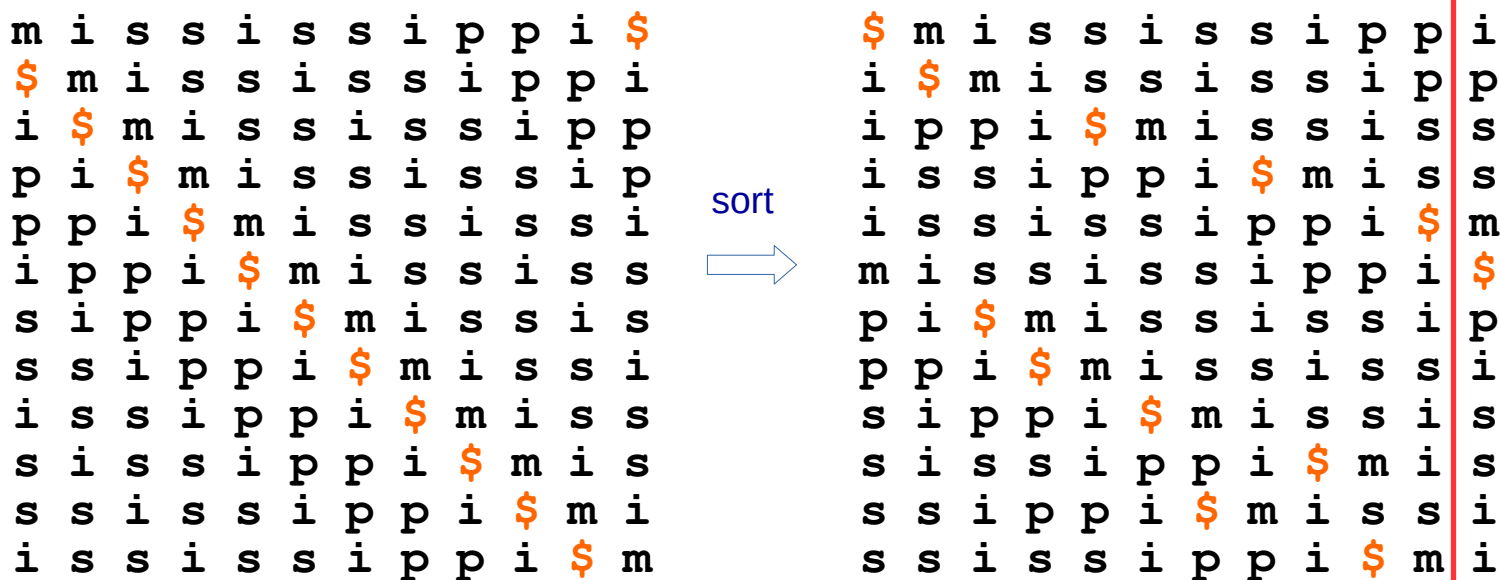
m	i	s	s	i	s	s	i	p	p	i	\$
\$	m	i	s	s	i	s	s	i	p	p	i
i	\$	m	i	s	s	i	s	s	i	p	p
p	i	\$	m	i	s	s	i	s	s	i	p
p	p	i	\$	m	i	s	s	i	s	s	i
i	p	p	i	\$	m	i	s	s	i	s	s
s	i	p	p	i	\$	m	i	s	s	i	s
s	s	i	p	p	i	\$	m	i	s	s	i
i	s	s	i	p	p	i	\$	m	i	s	s
s	i	s	s	i	p	p	i	\$	m	i	s
s	s	i	s	s	i	p	p	i	\$	m	i
i	s	s	i	s	s	i	p	p	i	\$	m

# Building a BWT

- Building a BWT consists of three steps:
  - Write down all cyclic rotations of the source string  $S$
  - Sort the rows lexicographically
  - Store the last column  $\rightarrow$   $BWT(S)$

$S$ : mississippi \$ ← end-of-line marker

rotate 



# Building a BWT

- Building a BWT consists of three steps:
  - Write down all cyclic rotations of the source string  $S$
  - Sort the rows lexicographically
  - Store the last column  $\rightarrow$   $BWT(S)$

$S$ : mississippi \$ ← end-of-line marker

rotate ↓

m	i	s	s	i	s	s	i	p	p	i	\$
\$	m	i	s	s	i	s	s	i	p	p	i
i	\$	m	i	s	s	i	s	s	i	p	p
p	i	\$	m	i	s	s	i	s	s	i	p
p	p	i	\$	m	i	s	s	i	s	s	i
i	p	p	i	\$	m	i	s	s	i	s	s
s	i	p	p	i	\$	m	i	s	s	i	s
s	s	i	p	p	i	\$	m	i	s	s	i
i	s	s	i	p	p	i	\$	m	i	s	s
s	i	s	s	i	p	p	i	\$	m	i	s
s	s	i	s	s	i	p	p	i	\$	m	i
i	s	s	i	s	s	i	p	p	i	\$	m

sort →

\$	m	i	s	s	i	s	s	i	p	p	i
i	\$	m	i	s	s	i	s	s	i	p	p
i	p	p	i	\$	m	i	s	s	i	s	s
i	s	s	i	p	p	i	\$	m	i	s	s
i	s	s	i	s	s	i	p	p	i	\$	m
m	i	s	s	i	s	s	i	p	p	i	\$
p	i	\$	m	i	s	s	i	s	s	i	p
p	p	i	\$	m	i	s	s	i	s	s	i
s	i	p	p	i	\$	m	i	s	s	i	s
s	i	s	s	i	p	p	i	\$	m	i	s
s	s	i	p	p	i	\$	m	i	s	s	i
s	s	i	s	s	i	p	p	i	\$	m	i

→

$BWT(S)$ :

i  
p  
s  
s  
m  
\$  
p  
i  
s  
s  
i  
i

# BWT properties

- BWT has two important features:
  - Rows of the matrix form a **sorted list of suffixes** → allows efficient search for substring occurrences
  - BWT is **reversible** → no need to store the whole matrix, since it can be obtained from the last column only

BWT

\$	m	i	s	s	i	s	s	i	p	p	i
i	\$	m	i	s	s	i	s	s	i	p	p
i	p	p	i	\$	m	i	s	s	i	s	s
i	s	s	i	p	p	i	\$	m	i	s	s
i	s	s	i	s	s	i	p	p	i	\$	m
m	i	s	s	i	s	s	i	p	p	i	\$
p	i	\$	m	i	s	s	i	s	s	i	p
p	p	i	\$	m	i	s	s	i	s	s	i
s	i	p	p	i	\$	m	i	s	s	i	s
s	i	s	s	i	p	p	i	\$	m	i	s
s	s	i	p	p	i	\$	m	i	s	s	i
s	s	i	s	s	i	p	p	i	\$	m	i

# BWT properties

- BWT has two important features:
  - Rows of the matrix form a **sorted list of suffixes** → allows efficient search for substring occurrences
  - BWT is **reversible** → no need to store the whole matrix, since it can be obtained from the last column only

	Suffix array	BWT
11	\$	\$ m i s s i s s i p p i
10	i \$	i \$ m i s s i s s i p p
7	i p p i \$	i p p i \$ m i s s i s s
4	i s s i p p i \$	i s s i p p i \$ m i s s
1	i s s i s s i p p i \$	i s s i s s i p p i \$ m
0	m i s s i s s i p p i \$	m i s s i s s i p p i \$
9	p i \$	p i \$ m i s s i s s i p
8	p p i \$	p p i \$ m i s s i s s i
6	s i p p i \$	s i p p i \$ m i s s i s
3	s i s s i p p i \$	s i s s i p p i \$ m i s
5	s s i p p i \$	s s i p p i \$ m i s s i
2	s s i s s i p p i \$	s s i s s i p p i \$ m i

# BWT properties

- BWT has two important features:
  - Rows of the matrix form a **sorted list of suffixes** → allows efficient search for substring occurrences
  - BWT is **reversible** → no need to store the whole matrix, since it can be obtained from the last column only

32b/char	Suffix array	BWT	2b/char
11	\$	\$ m i s s i s s i p p	i
10	i \$	i \$ m i s s i s s i p p	p
7	i p p i \$	i p p i \$ m i s s i s s	s
4	i s s i p p i \$	i s s i p p i \$ m i s s	s
1	i s s i s s i p p i \$	i s s i s s i p p i \$ m	i
0	m i s s i s s i p p i \$	m i s s i s s i p p i \$	\$
9	p i \$	p i \$ m i s s i s s i p	p
8	p p i \$	p p i \$ m i s s i s s i	s
6	s i p p i \$	s i p p i \$ m i s s i s	s
3	s i s s i p p i \$	s i s s i p p i \$ m i s	i
5	s s i p p i \$	s s i p p i \$ m i s s	i
2	s s i s s i p p i \$	s s i s s i p p i \$ m	i

# Why BWT?

- Space complexity H. sapiens (3Gb)
  - Suffix tree:  $\sim 20 * |\text{Genome}|$  60 GB
  - Suffix array:  $\sim 4 * |\text{Genome}|$  12 GB
  - BWT-FM:  $\sim 0.5 * |\text{Genome}|$  1.5 GB
- Additional data structure needed for search
- **FM-index** (Ferragina and Manzini, 2000)

# Learn from the experts!

- Pavel Pevzner (UC San Diego)
  - co-author of **SPAdes** → de-novo assembler
  - <https://www.youtube.com/c/bioinfallgorithms/videos>
- Ben Langmead (John Hopkins)
  - the author of **Bowtie** → BWT-based read mapper
  - <https://www.youtube.com/user/BenLangmead/playlists>



Questions?

# References

Altschul, Stephen; Gish, Warren; Miller, Webb; Myers, Eugene; Lipman, David (1990). [Basic local alignment search tool](#). *Journal of Molecular Biology* 215 (3): 403–410. doi:10.1016/S0022-2836(05)80360-2. PMID 2231712

Bankevich, Anton et al. (2012) [SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing](#). *J Comput Biol.* 2012 May; 19(5): 455–477. <https://dx.doi.org/10.1089%2Fcmb.2012.0021>

Burrows, Michael; Wheeler, David J. (1994). [A block sorting lossless data compression algorithm](#), Technical Report 124, Digital Equipment Corporation

Darriba D, Flouri T, Stamatakis A (2018) [The State of Software for Evolutionary Biology](#), *Molecular Biology and Evolution*, Volume 35, Issue 5, 1 May 2018, Pages 1037–1046, <https://doi.org/10.1093/molbev/msy014>

Eddy, S. R. (2009). [A New Generation of Homology Search Tools Based on Probabilistic Inference](#). *Genome Inform.*, 23:205-211.

Edgar, Robert C. (2010). [Search and clustering orders of magnitude faster than BLAST](#). *Bioinformatics* (2010) 26 (19): 2460-2461 doi:10.1093/bioinformatics/btq461

Ferragina P, Manzini G (2000). [Opportunistic data structures with applications](#). *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*

Kent, WJ (2002). [BLAT--the BLAST-like alignment tool](#). *Genome Research* 12 (4): 656–664. doi:10.1101/gr.229202. PMC 187518. PMID 11932250

Morgulis A, Coulouris G, Raytselis Y, Madden T, Agarwala R, and Schäffer AA (2008). [Database indexing for production MegaBLAST searches](#). *Bioinformatics* 24 (16): 1757-1764 first published online June 21, 2008 doi:10.1093/bioinformatics/btn322

Nidhi S, Nute GM, Warnow T, Pop M (2018) [Misunderstood parameter of NCBI BLAST impacts the correctness of bioinformatics workflows](#), *Bioinformatics*, bty833, <https://doi.org/10.1093/bioinformatics/bty833>

Wang, Q, G. M. Garrity, J. M. Tiedje, and J. R. Cole (2007). [Naïve Bayesian Classifier for Rapid Assignment of rRNA Sequences into the New Bacterial Taxonomy](#). *Appl Environ Microbiol.* 73(16):5261-5267; doi: 10.1128/AEM.00062-07 [PMID: 17586664]

+ **additional references in the old slide set:** <http://sco.h-its.org/exelixis/web/teaching/lectures2014/lecture4.pdf>

Backup



# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

F													L
\$	m	i	s	s	i	s	s	i	p	p	i		$i_0$
$i_0$	\$	m	i	s	s	i	s	s	i	p	p		
$i_1$	p	p	i	\$	m	i	s	s	i	s	s		
$i_2$	s	s	i	p	p	i	\$	m	i	s	s		
$i_3$	s	s	i	s	s	i	p	p	i	\$	m		
m	i	s	s	i	s	s	i	p	p	i	\$		
p	i	\$	m	i	s	s	i	s	s	i	p		
p	p	i	\$	m	i	s	s	i	s	s	$i_1$		
s	i	p	p	i	\$	m	i	s	s	i	s		
s	i	s	s	i	p	p	i	\$	m	i	s		
s	s	i	p	p	i	\$	m	i	s	s	$i_2$		
s	s	i	s	s	i	p	p	i	\$	m	$i_3$		

F													L
\$													$i_0$
$i_0$													$p_0$
$i_1$													$s_0$
$i_2$													$s_1$
$i_3$													$m_0$
$m_0$													\$
$p_0$													$p_1$
$p_1$													$i_1$
$s_0$													$s_2$
$s_1$													$s_3$
$s_2$													$i_2$
$s_3$													$i_3$

# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

F																			L
\$	m	i	s	s	i	s	s	i	p	p	i								i <sub>0</sub>
i <sub>0</sub>	\$	m	i	s	s	i	s	s	i	p	p								p <sub>0</sub>
i <sub>1</sub>	p	p	i	\$	m	i	s	s	i	s	s								s <sub>0</sub>
i <sub>2</sub>	s	s	i	p	p	i	\$	m	i	s	s								s <sub>1</sub>
i <sub>3</sub>	s	s	i	s	s	i	p	p	i	\$	m								m <sub>0</sub>
m	i	s	s	i	s	s	i	p	p	i	\$								\$ <sub>0</sub>
p	i	\$	m	i	s	s	i	s	s	i	p								p <sub>1</sub>
p	p	i	\$	m	i	s	s	i	s	s	i	i <sub>1</sub>							i <sub>1</sub>
s	i	p	p	i	\$	m	i	s	s	i	s								s <sub>2</sub>
s	i	s	s	i	p	p	i	\$	m	i	s								s <sub>3</sub>
s	s	i	p	p	i	\$	m	i	s	s	i	i <sub>2</sub>							i <sub>2</sub>
s	s	i	s	s	i	p	p	i	\$	m	i	i <sub>3</sub>							i <sub>3</sub>

F																			L
\$ <sub>0</sub>																			\$ <sub>0</sub>
i <sub>0</sub>																			i <sub>0</sub>
i <sub>1</sub>																			p <sub>0</sub>
i <sub>2</sub>																			s <sub>0</sub>
i <sub>3</sub>																			s <sub>1</sub>
m <sub>0</sub>																			m <sub>0</sub>
p <sub>0</sub>																			\$ <sub>0</sub>
p <sub>1</sub>																			p <sub>1</sub>
s <sub>0</sub>																			i <sub>1</sub>
s <sub>1</sub>																			s <sub>2</sub>
s <sub>2</sub>																			s <sub>3</sub>
s <sub>3</sub>																			i <sub>2</sub>
																			i <sub>3</sub>

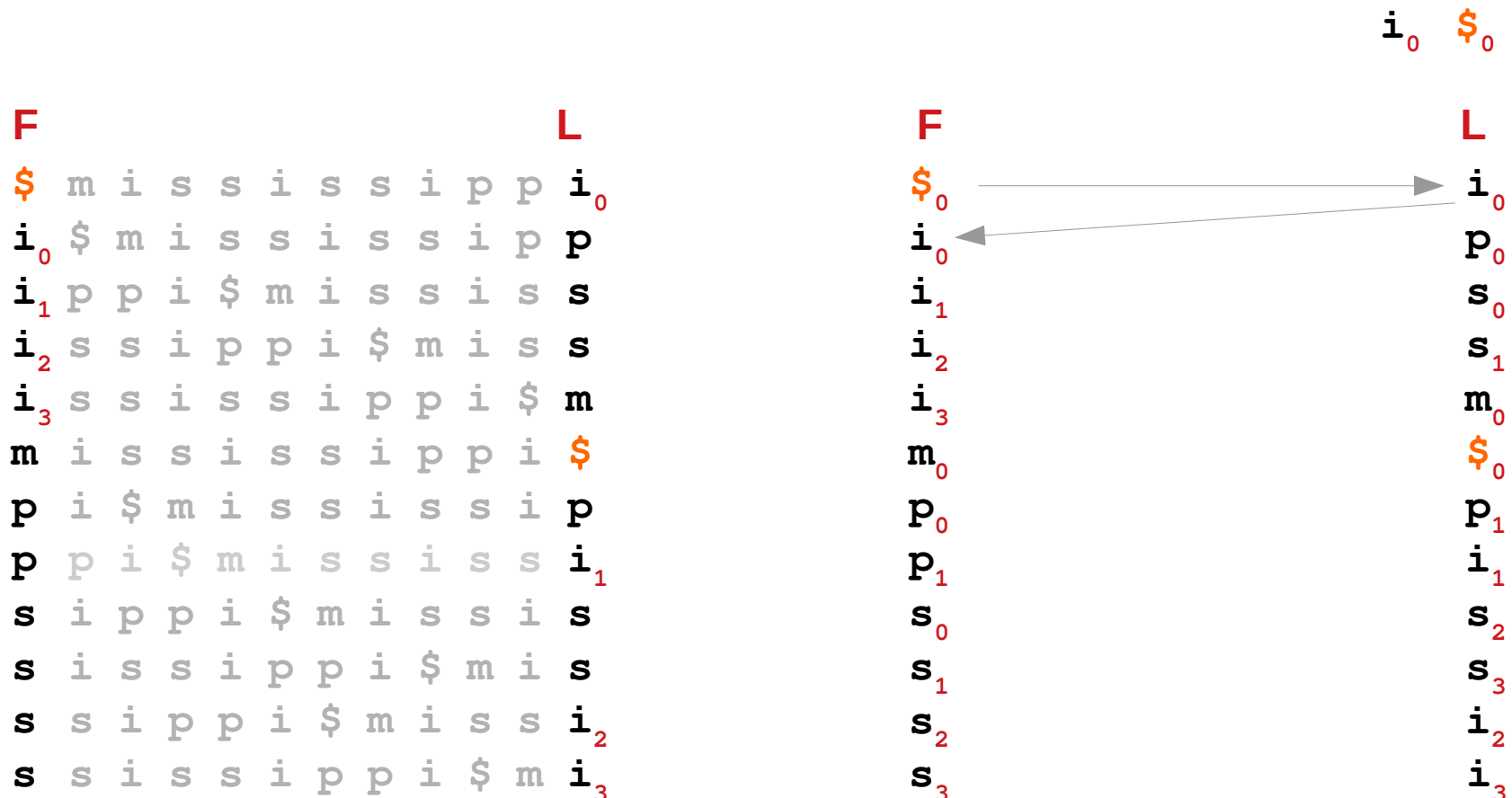
# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns



# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

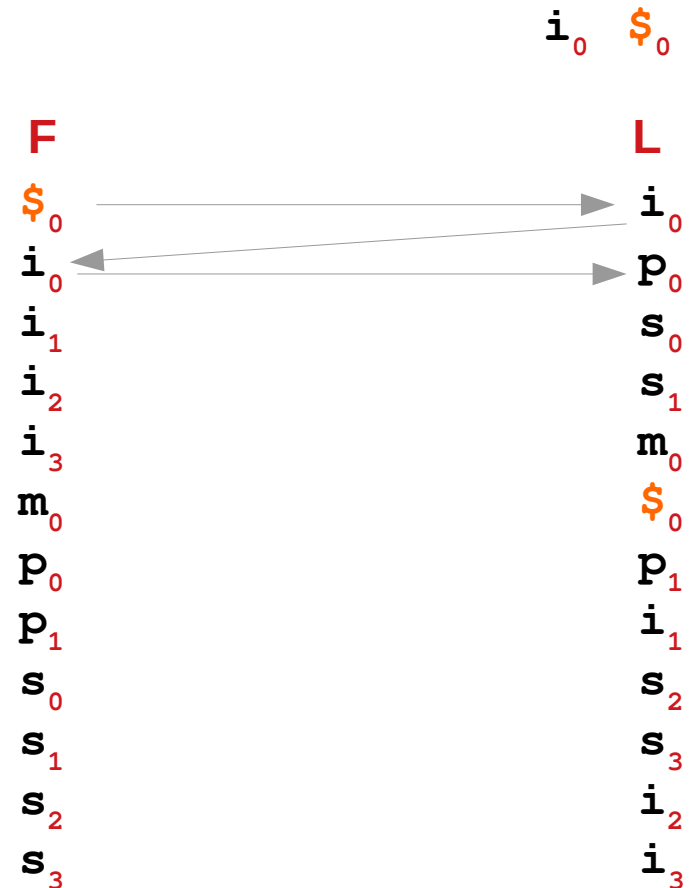




# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

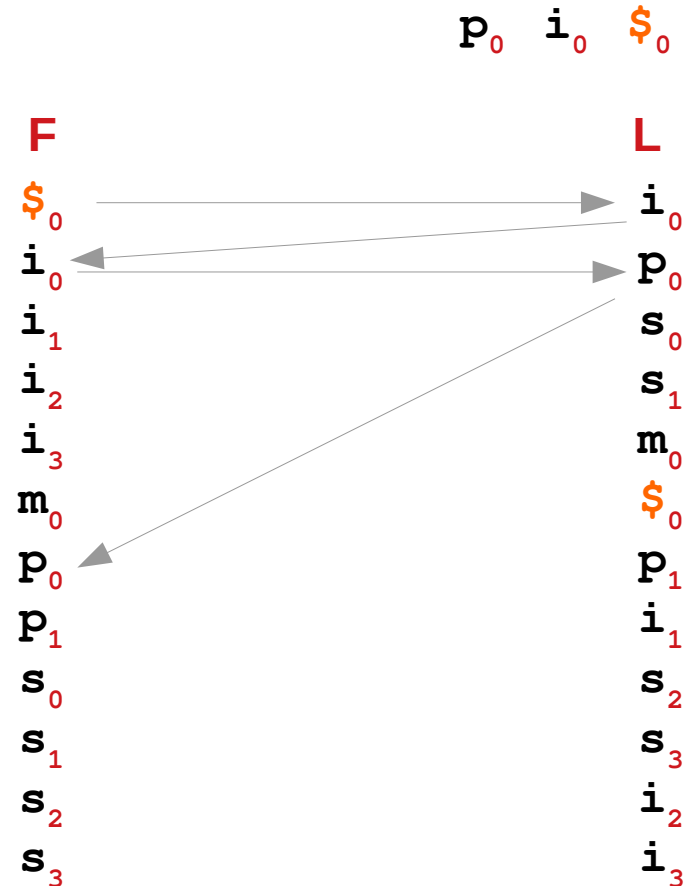
F																		L
\$	m	i	s	s	i	s	s	i	p	p	i							i <sub>0</sub>
i <sub>0</sub>	\$	m	i	s	s	i	s	s	i	p	p							
i <sub>1</sub>	p	p	i	\$	m	i	s	s	i	s	s							
i <sub>2</sub>	s	s	i	p	p	i	\$	m	i	s	s							
i <sub>3</sub>	s	s	i	s	s	i	p	p	i	\$	m							
m	i	s	s	i	s	s	i	p	p	i	\$							
p	i	\$	m	i	s	s	i	s	s	i	p							
p	p	i	\$	m	i	s	s	i	s	s	i	i <sub>1</sub>						
s	i	p	p	i	\$	m	i	s	s	i	s							
s	i	s	s	i	p	p	i	\$	m	i	s							
s	s	i	p	p	i	\$	m	i	s	s	i	i <sub>2</sub>						
s	s	i	s	s	i	p	p	i	\$	m	i	i <sub>3</sub>						



# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

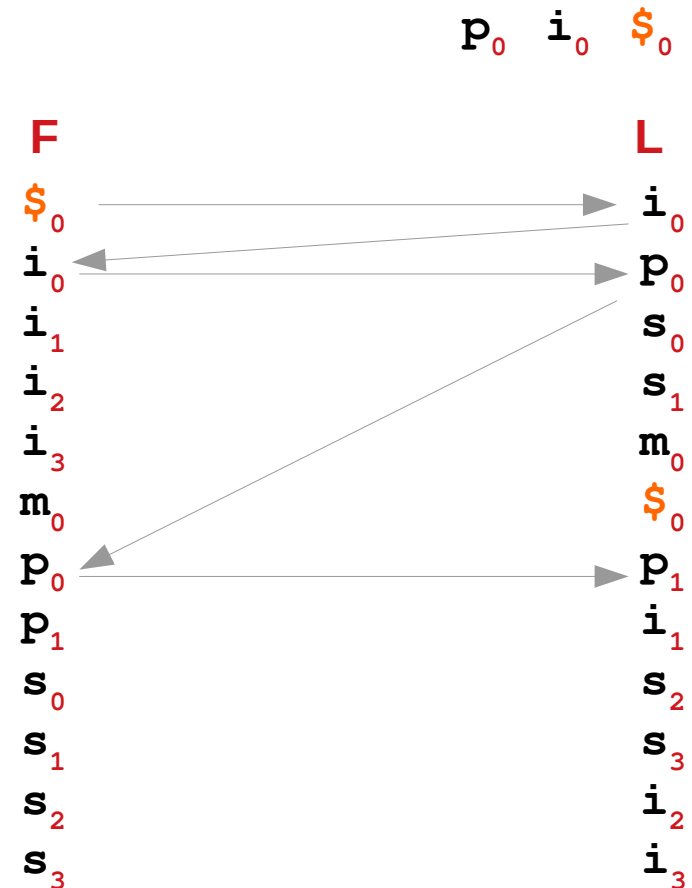
F																				L
\$	m	i	s	s	i	s	s	i	p	p	i									i <sub>0</sub>
i <sub>0</sub>	\$	m	i	s	s	i	s	s	i	p	p									
i <sub>1</sub>	p	p	i	\$	m	i	s	s	i	s	s									
i <sub>2</sub>	s	s	i	p	p	i	\$	m	i	s	s									
i <sub>3</sub>	s	s	i	s	s	i	p	p	i	\$	m									
m	i	s	s	i	s	s	i	p	p	i	\$									
p	i	\$	m	i	s	s	i	s	s	i	p									
p	p	i	\$	m	i	s	s	i	s	s	i	i <sub>1</sub>								
s	i	p	p	i	\$	m	i	s	s	i	s									
s	i	s	s	i	p	p	i	\$	m	i	s									
s	s	i	p	p	i	\$	m	i	s	s	i	i <sub>2</sub>								
s	s	i	s	s	i	p	p	i	\$	m	i	i <sub>3</sub>								



# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

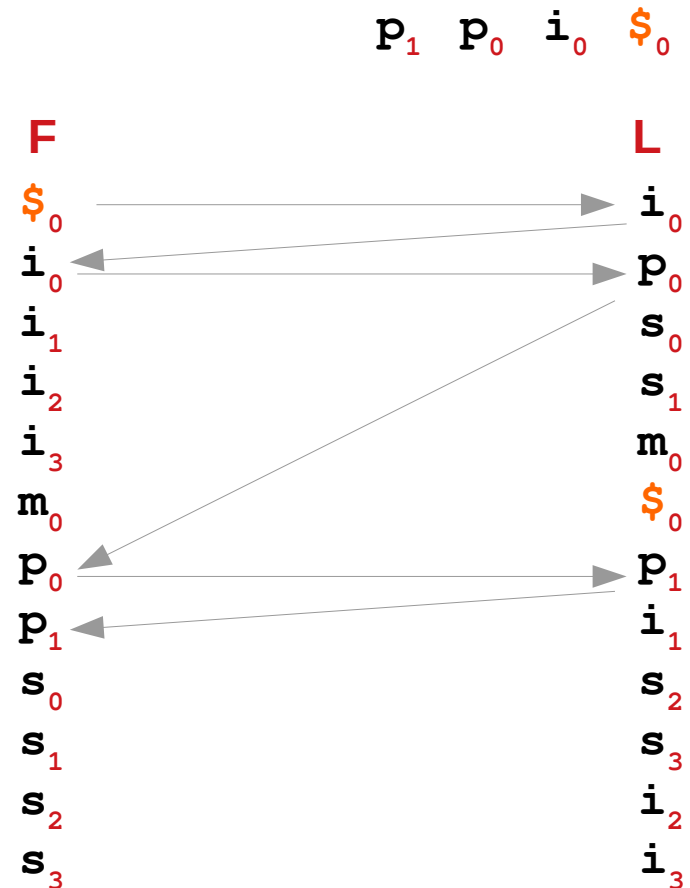
F													L
\$	m	i	s	s	i	s	s	i	p	p	i		$i_0$
$i_0$	\$	m	i	s	s	i	s	s	i	p	p		
$i_1$	p	p	i	\$	m	i	s	s	i	s	s		
$i_2$	s	s	i	p	p	i	\$	m	i	s	s		
$i_3$	s	s	i	s	s	i	p	p	i	\$	m		
m	i	s	s	i	s	s	i	p	p	i	\$		
p	i	\$	m	i	s	s	i	s	s	i	p		
p	p	i	\$	m	i	s	s	i	s	s	i	$i_1$	
s	i	p	p	i	\$	m	i	s	s	i	s		
s	i	s	s	i	p	p	i	\$	m	i	s		
s	s	i	p	p	i	\$	m	i	s	s	i	$i_2$	
s	s	i	s	s	i	p	p	i	\$	m	i	$i_3$	



# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

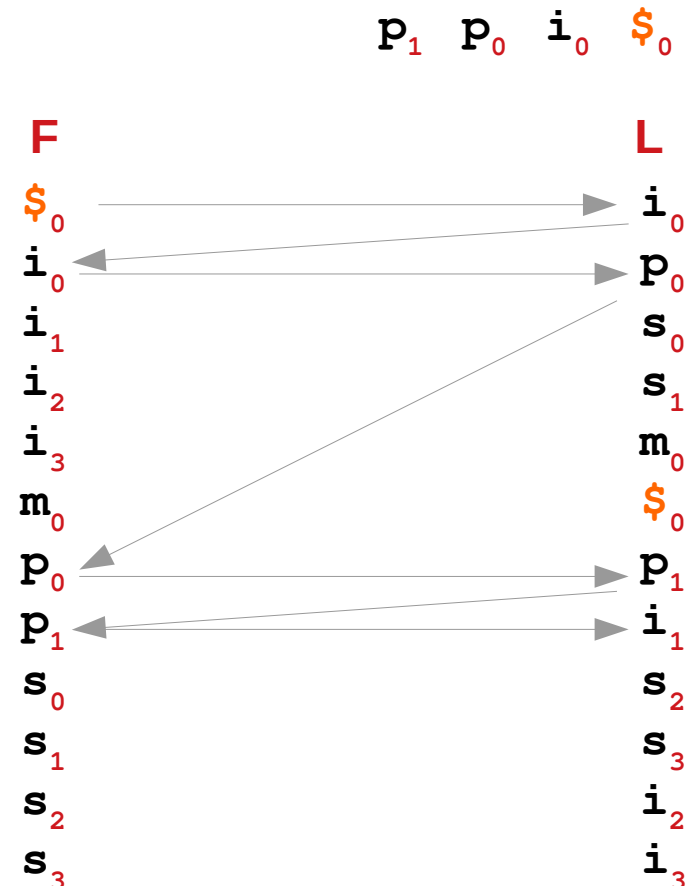
F																			L
\$	m	i	s	s	i	s	s	i	p	p	i								i <sub>0</sub>
i <sub>0</sub>	\$	m	i	s	s	i	s	s	i	p	p								
i <sub>1</sub>	p	p	i	\$	m	i	s	s	i	s	s								
i <sub>2</sub>	s	s	i	p	p	i	\$	m	i	s	s								
i <sub>3</sub>	s	s	i	s	s	i	p	p	i	\$	m								
m	i	s	s	i	s	s	i	p	p	i	\$								
p	i	\$	m	i	s	s	i	s	s	i	p								
p	p	i	\$	m	i	s	s	i	s	s	i <sub>1</sub>								
s	i	p	p	i	\$	m	i	s	s	i	s								
s	i	s	s	i	p	p	i	\$	m	i	s								
s	s	i	p	p	i	\$	m	i	s	s	i <sub>2</sub>								
s	s	i	s	s	i	p	p	i	\$	m	i <sub>3</sub>								



# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

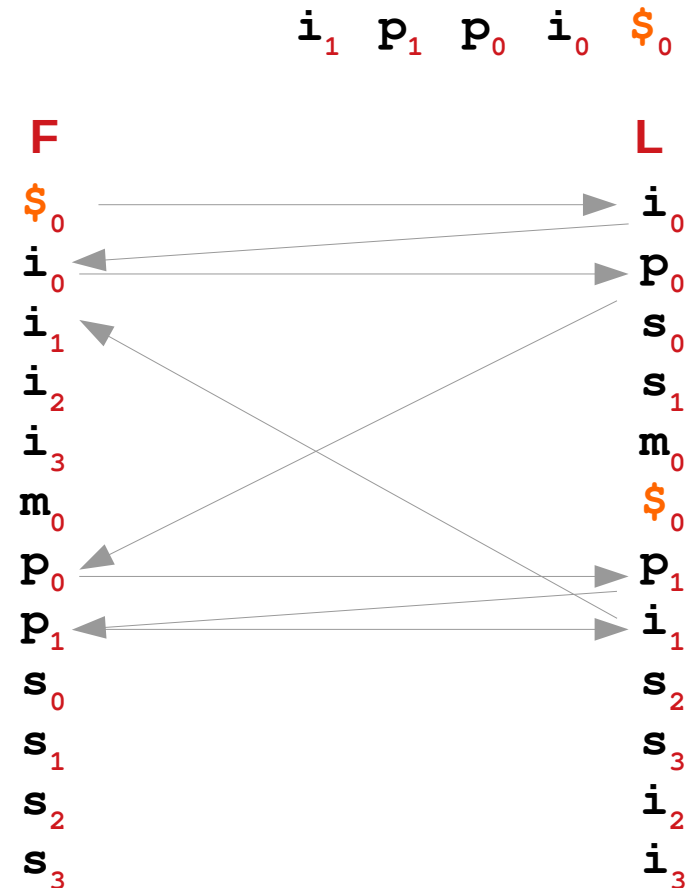
F													L
\$	m	i	s	s	i	s	s	i	p	p	i		i <sub>0</sub>
i <sub>0</sub>	\$	m	i	s	s	i	s	s	i	p	p		
i <sub>1</sub>	p	p	i	\$	m	i	s	s	i	s	s		
i <sub>2</sub>	s	s	i	p	p	i	\$	m	i	s	s		
i <sub>3</sub>	s	s	i	s	s	i	p	p	i	\$	m		
m	i	s	s	i	s	s	i	p	p	i	\$		
p	i	\$	m	i	s	s	i	s	s	i	p		
p	p	i	\$	m	i	s	s	i	s	s	i <sub>1</sub>		
s	i	p	p	i	\$	m	i	s	s	i	s		
s	i	s	s	i	p	p	i	\$	m	i	s		
s	s	i	p	p	i	\$	m	i	s	s	i <sub>2</sub>		
s	s	i	s	s	i	p	p	i	\$	m	i <sub>3</sub>		



# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

F																				L
\$	m	i	s	s	i	s	s	i	p	p	i									$i_0$
$i_0$	\$	m	i	s	s	i	s	s	i	p	p									
$i_1$	p	p	i	\$	m	i	s	s	i	s	s									
$i_2$	s	s	i	p	p	i	\$	m	i	s	s									
$i_3$	s	s	i	s	s	i	p	p	i	\$	m									
m	i	s	s	i	s	s	i	p	p	i	\$									
p	i	\$	m	i	s	s	i	s	s	i	p									
p	p	i	\$	m	i	s	s	i	s	s	$i_1$									
s	i	p	p	i	\$	m	i	s	s	i	s									
s	i	s	s	i	p	p	i	\$	m	i	s									
s	s	i	p	p	i	\$	m	i	s	s	$i_2$									
s	s	i	s	s	i	p	p	i	\$	m	$i_3$									

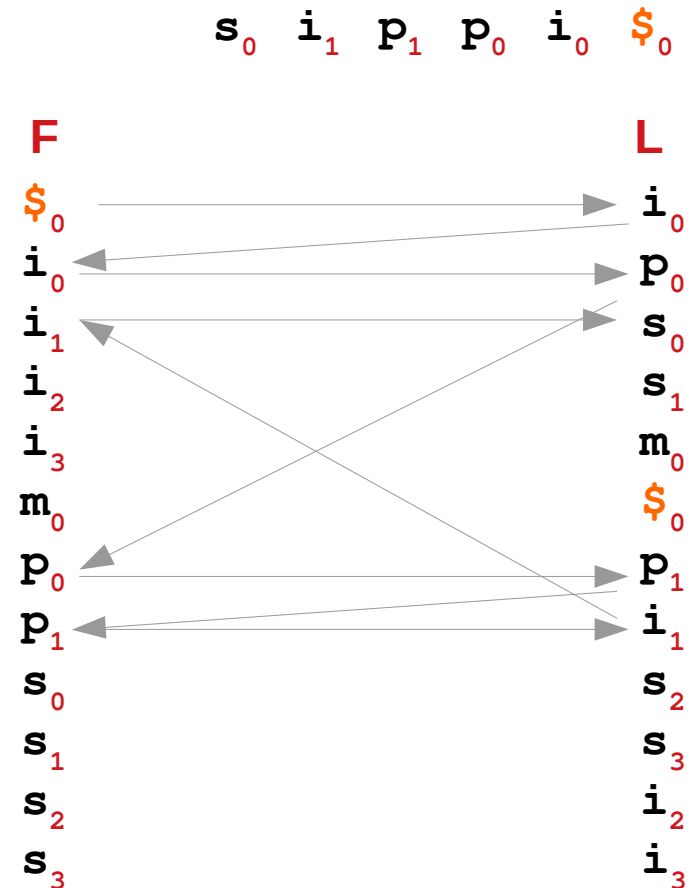




# FM-Index: LF mapping

- Character ranks are consistent between first (F) and last (L) columns

F																				L
\$	m	i	s	s	i	s	s	i	p	p	i									i <sub>0</sub>
i <sub>0</sub>	\$	m	i	s	s	i	s	s	i	p	p									
i <sub>1</sub>	p	p	i	\$	m	i	s	s	i	s	s									
i <sub>2</sub>	s	s	i	p	p	i	\$	m	i	s	s									
i <sub>3</sub>	s	s	i	s	s	i	p	p	i	\$	m									
m	i	s	s	i	s	s	i	p	p	i	\$									
p	i	\$	m	i	s	s	i	s	s	i	p									
p	p	i	\$	m	i	s	s	i	s	s	i <sub>1</sub>									
s	i	p	p	i	\$	m	i	s	s	i	s									
s	i	s	s	i	p	p	i	\$	m	i	s									
s	s	i	p	p	i	\$	m	i	s	s	i <sub>2</sub>									
s	s	i	s	s	i	p	p	i	\$	m	i <sub>3</sub>									





# FM Index

- In addition to BWT itself, the FM-index stores:
  - F offsets → **FO**, L rank checkpoints → **LRC**, SA sample → **SAS**

<b>FO</b>	<b>F</b>	<b>L</b>
0	\$ m i s s i s s i p p i	
1	i \$ m i s s i s s i p p	
	i p p i \$ m i s s i s s	
	i s s i p p i \$ m i s s	
	i s s i s s i p p i \$ m	
5	m i s s i s s i p p i \$	
6	p i \$ m i s s i s s i p	
	p p i \$ m i s s i s s i	
8	s i p p i \$ m i s s i s	
	s i s s i p p i \$ m i s	
	s s i p p i \$ m i s s i	
	s s i s s i p p i \$ m i	

# FM Index

- In addition to BWT itself, the FM-index stores:
  - F offsets → **FO**, L rank checkpoints → **LRC**, SA sample → **SAS**

<b>FO</b>	<b>F</b>	<b>L</b>
0	\$ m i s s i s s i p p i	
1	i \$ m i s s i s s i p p	
	i p p i \$ m i s s i s s	
	i s s i p p i \$ m i s s	
	i s s i s s i p p i \$ m	
5	m i s s i s s i p p i \$	
6	p i \$ m i s s i s s i p	
	p p i \$ m i s s i s s i	
8	s i p p i \$ m i s s i s	
	s i s s i p p i \$ m i s	
	s s i p p i \$ m i s s i	
	s s i s s i p p i \$ m i	

<b>i</b>	<b>m</b>	<b>p</b>	<b>s</b>
1	0	0	0
1	0	1	0
1	0	1	1
1	0	1	2
1	1	1	2
1	1	1	2
1	1	2	2
2	1	2	2
2	1	2	3
2	1	2	4
3	1	2	4
4	1	2	4

# FM Index

- In addition to BWT itself, the FM-index stores:
  - F offsets → **FO**, L rank checkpoints → **LRC**, SA sample → **SAS**

<b>FO</b>	<b>F</b>		<b>L</b>	<b>LRC</b>												
0	\$	m	i	s	s	i	s	s	i	p	p	i	i	m	p	s
1	i	\$	m	i	s	s	i	s	s	i	p	p				
	i	p	p	i	\$	m	i	s	s	i	s	s				
	i	s	s	i	p	p	i	\$	m	i	s	s				
	i	s	s	i	s	s	i	p	p	i	\$	m				
5	m	i	s	s	i	s	s	i	p	p	i	\$	1	1	1	2
6	p	i	\$	m	i	s	s	i	s	i	p					
	p	p	i	\$	m	i	s	s	i	s	s	i				
8	s	i	p	p	i	\$	m	i	s	s	i	s				
	s	i	s	s	i	p	p	i	\$	m	i	s				
	s	s	i	p	p	i	\$	m	i	s	s	i	3	1	2	4
	s	s	i	s	s	i	p	p	i	\$	m	i				

# FM Index

- In addition to BWT itself, the FM-index stores:
  - F offsets → **FO**, L rank checkpoints → **LRC**, SA sample → **SAS**

**SAS**

10
4
0
8
6
2

**FO**

0
1
5
6
8

**F**

**\$** m i s s i s s i p p i  
i \$ m i s s i s s i p p  
i p p i \$ m i s s i s s  
i s s i p p i \$ m i s s  
i s s i s s i p p i \$ m  
m i s s i s s i p p i **\$**  
p i \$ m i s s i s s i p  
p p i \$ m i s s i s s i  
s i p p i \$ m i s s i s  
s i s s i p p i \$ m i s  
s s i p p i \$ m i s s i  
s s i s s i p p i \$ m i

**L**

**LRC**

<b>i</b>	<b>m</b>	<b>p</b>	<b>s</b>
1	0	0	0
1	1	1	2
3	1	2	4

# FM-Index: Pattern search

Find pattern: **sip**

F										L		
\$	m	i	s	s	i	s	s	i	p	p	i	0
i <sub>0</sub>	\$	m	i	s	s	i	s	s	i	p	p	
i <sub>1</sub>	p	p	i	\$	m	i	s	s	i	s	s	
i <sub>2</sub>	s	s	i	p	p	i	\$	m	i	s	s	
i <sub>3</sub>	s	s	i	s	s	i	p	p	i	\$	m	
m	i	s	s	i	s	s	i	p	p	i	\$	
p	i	\$	m	i	s	s	i	s	s	i	p	
p	p	i	\$	m	i	s	s	i	s	s	i	1
s	i	p	p	i	\$	m	i	s	s	i	s	
s	i	s	s	i	p	p	i	\$	m	i	s	
s	s	i	p	p	i	\$	m	i	s	s	i	2
s	s	i	s	s	i	p	p	i	\$	m	i	3

Start here: →

F												L	
\$												i	0
i <sub>0</sub>												p	0
i <sub>1</sub>												s	0
i <sub>2</sub>												s	1
i <sub>3</sub>												m	0
m												\$	0
p												p	1
p												i	1
s												s	2
s												s	3
s												i	2
s												i	3







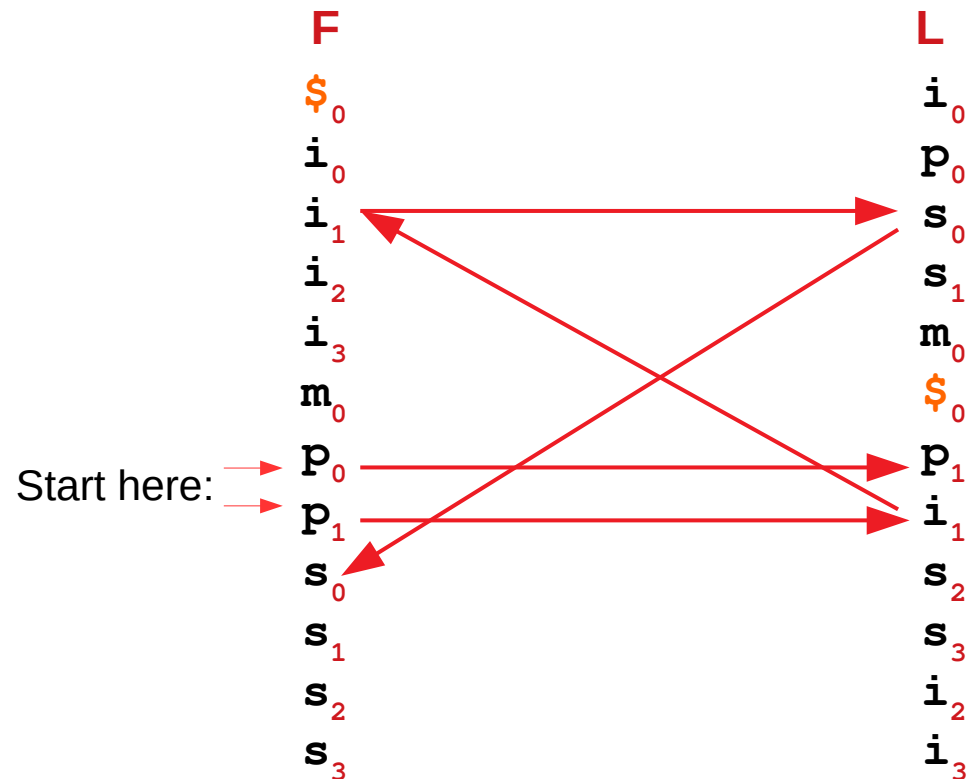


# FM-Index: Pattern search

Find pattern: **sip**

$s_0$   $i_1$   $p_1$

F															L
\$	m	i	s	s	i	s	s	i	p	p	i				$i_0$
$i_0$	\$	m	i	s	s	i	s	s	i	p	p				
$i_1$	p	p	i	\$	m	i	s	s	i	s	s				
$i_2$	s	s	i	p	p	i	\$	m	i	s	s				
$i_3$	s	s	i	s	s	i	p	p	i	\$	m				
m	i	s	s	i	s	s	i	p	p	i	\$				
p	i	\$	m	i	s	s	i	s	s	i	p				
p	p	i	\$	m	i	s	s	i	s	s	i				$i_1$
s	i	p	p	i	\$	m	i	s	s	i	s				
s	i	s	s	i	p	p	i	\$	m	i	s				
s	s	i	p	p	i	\$	m	i	s	s	i				$i_2$
s	s	i	s	s	i	p	p	i	\$	m	i				$i_3$



# FM-Index: Pattern search

Find pattern: **sip**

**s<sub>0</sub> i<sub>1</sub> p<sub>1</sub>**

F																			L
\$	m	i	s	s	i	s	s	i	p	p	i								i <sub>0</sub>
i <sub>0</sub>	\$	m	i	s	s	i	s	s	i	p	p								p <sub>0</sub>
i <sub>1</sub>	p	p	i	\$	m	i	s	s	i	s	s								s <sub>0</sub>
i <sub>2</sub>	s	s	i	p	p	i	\$	m	i	s	s								s <sub>1</sub>
i <sub>3</sub>	s	s	i	s	s	i	p	p	i	\$	m								m <sub>0</sub>
m	i	s	s	i	s	s	i	p	p	i	\$								\$ <sub>0</sub>
p	i	\$	m	i	s	s	i	s	s	i	p								p <sub>1</sub>
p	p	i	\$	m	i	s	s	i	s	s	i								i <sub>1</sub>
s	i	p	p	i	\$	m	i	s	s	i	s								s <sub>2</sub>
s	i	s	s	i	p	p	i	\$	m	i	s								s <sub>3</sub>
s	s	i	p	p	i	\$	m	i	s	s	i								i <sub>2</sub>
s	s	i	s	s	i	p	p	i	\$	m	i								i <sub>3</sub>

