Bioinformatics

Lecture 3

Weidong Tian, School of Life Sciences, Fudan University

Sept 19, 2024

Outline

- Review of last lecture
- Database search
- Reads mapping

PMA1 mutation probability matrix

Mutational probability matrix derived by Dayhoff for the 20 amino acids

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	9867	2	9	10	3	8	17	21	2	6	4	2	6	2	22	35	32	0	2	18
R	1	9913	1	0	1	10	0	0	10	3	1	19	4	1	4	6	1	8	0	1
N	4	1	9822	36	0	4	6	6	21	3	1	13	0	1	2	20	9	1	4	1
D	6	0	42	9859	0	6	53	6	4	1	0	3	0	0	1	5	3	0	0	1
C	1	1	0	0	9973	0	0	0	1	1	0	0	0	0	1	5	1	0	3	2
Q	3	9	4	5	0	9876	27	1	23	1	3	6	4	0	6	2	2	0	0	1
E	10	0	7	56	0	35	9865	4	2	3	1	4	1	0	3	4	2	0	1	2
G	21	1	12	11	1	3	7	9935	1	0	1	2	1	1	3	21	3	0	0	5
H	1	8	18	3	1	20	1	0	9912	0	1	1	0	2	3	1	1	1	4	1
I	2	2	3	1	2	1	2	0	0	9872	9	2	12	7	0	1	7	0	1	33
L	3	1	3	0	0	6	1	1	4	22	9947	2	45	13	3	1	3	4	2	15
K	2	37	25	6	0	12	7	2	2	4	1	9926	20	0	3	8	11	0	1	1
M	1	1	0	0	0	2	0	0	0	5	8	4	9874	1	0	1	2	0	0	4
F	1	1	1	0	0	0	0	1	2	8	6	0	4	9946	0	2	1	3	28	0
P	13	5	2	1	1	8	3	2	5	1	2	2	1	1	9926	12	4	0	0	2
S	28	11	34	7	11	4	6	16	2	2	1	7	4	3	17	9840	38	5	2	2
T	22	2	13	4	1	3	2	2	1	11	2	8	6	1	5	32	9871	0	2	9
W	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	9976	1	0
Y	1	0	3	0	3	0	1	0	4	1	1	0	0	21	0	1	1	2	9945	1
V	13	2	1	1	3	2	2	3	3	57	11	1	17	1	3	2	10	0	2	9901

For clarity, the values have been multiplied by 10000

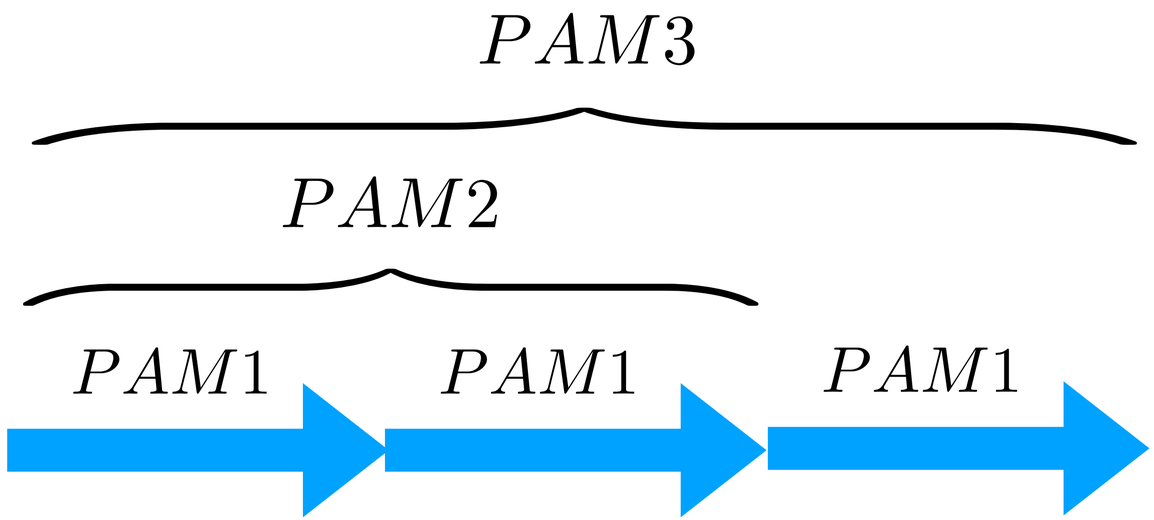
This matrix corresponds to an evolution time period giving 1 mutation/100 amino acids, and is refered to as the **PAM1 matrix.**

$$P(C_i, i \neq j | C_j) = \lambda m_j \cdot \frac{A_{ij}}{\sum_{k \neq j} A_{kj}}$$

How to estimate the mutation probability matrix at PAM2?

Source: Dayhoff, 1978

Markovian Assumption in Evolution



$$PAM3 = PAM1 \times PAM1 \times PAM1 = PAM1^3$$

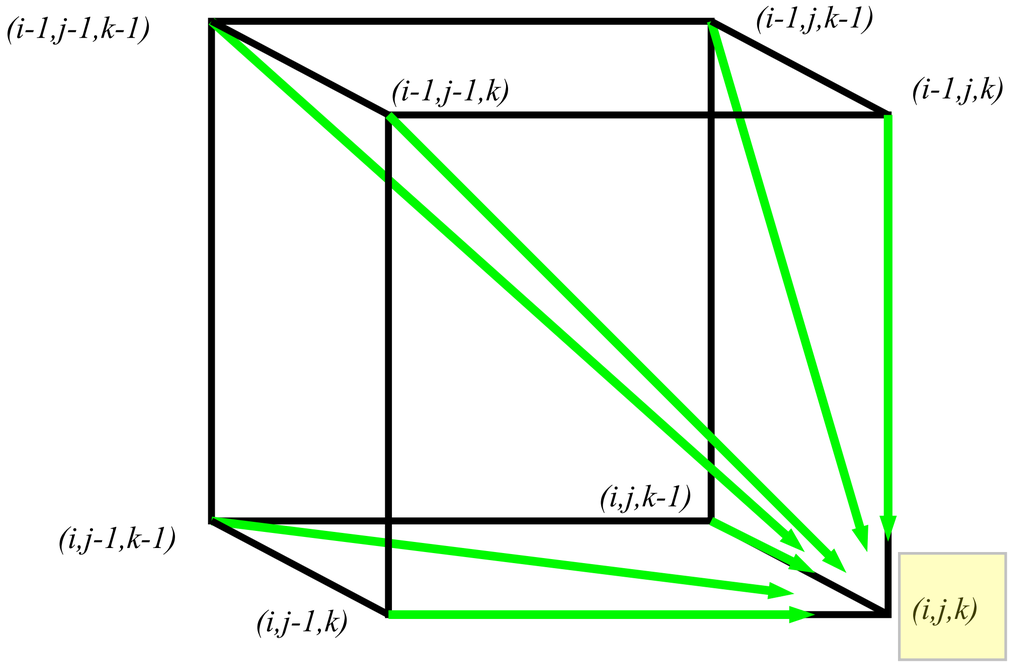
The PAM250 log-odds scoring matrix

$$s(a,b) = \log\left(\frac{p_{ab}}{q_a q_b}\right) = log\frac{f(j)M^n(i,j)}{f(i)f(j)} = log\frac{M^n(i,j)}{f(i)}$$

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	12															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	-2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6					
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

FIGURE 3.14 Log-odds matrix for PAM250. High PAM values (e.g., PAM250) are useful for aligning very divergent sequences. A variety of algorithms for pairwise alignment, multiple sequence alignment, and database searching (e.g., BLAST) allow you to select an assortment of PAM matrices such as PAM250, PAM70, and PAM30. Adapted from NCBI, ftp://ftp.ncbi.nlm.nih.gov/blast/matrices/.

Use a 3-D “Manhattan Cube” to align three sequences by dynamic programming



$$s_{i,j,k} = \max \begin{cases} s_{i-1,j-1,k-1} + \delta(v_i, w_j, u_k) & \text{cube diagonal: no indels} \\ s_{i-1,j-1,k} + \delta(v_i, w_j, \_) & \\ s_{i-1,j,k-1} + \delta(v_i, \_, u_k) & \text{face diagonal: one indel} \\ s_{i,j-1,k-1} + \delta(\_, w_j, u_k) & \\ s_{i-1,j,k} + \delta(v_i, \_, \_) & \\ s_{i,j-1,k} + \delta(\_, w_j, \_) & \text{edge diagonal: two indels} \\ s_{i,j,k-1} + \delta(\_, \_, u_k) & \end{cases}$$

$\delta(x, y, z)$ is an entry in the 3-D scoring matrix

A typical progressive alignment

- Progressive alignment is a variation of greedy algorithm with a somewhat more intelligent strategy for choosing the order of alignments.

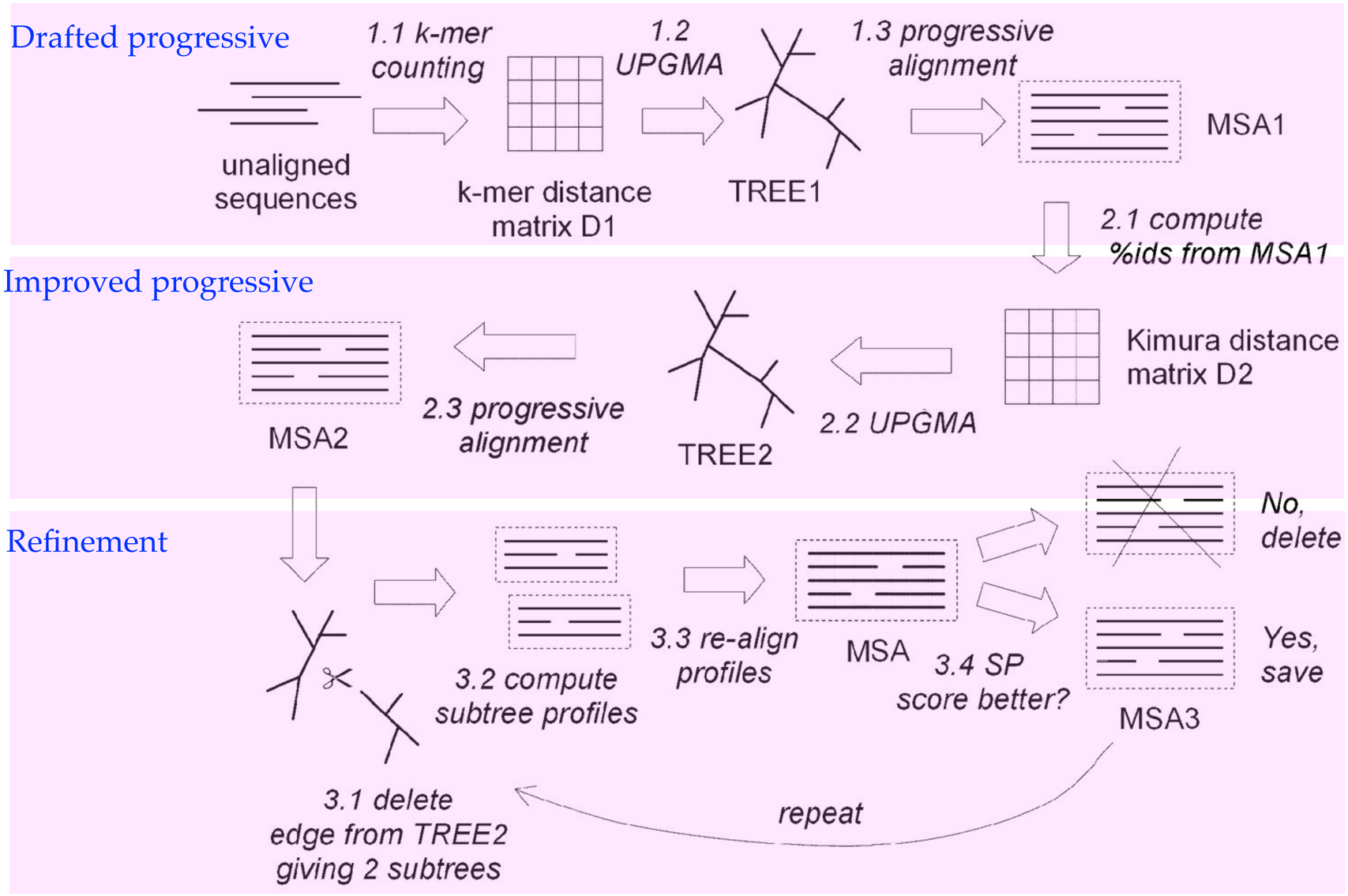
- Three-step process

  1.) Construct pairwise alignments

  2.) Build Guide Tree

  3.) Progressive Alignment guided by the tree

Current widely used MSA tool—MUSCLE



MAFFT

Nucleic Acids Research, 2002, Vol. 30 No. 14 3059–3066

MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform

Kazutaka Katoh, Kazuharu Misawa[1], Kei-ichi Kuma and Takashi Miyata*

Department of Biophysics, Graduate School of Science, Kyoto University, Kyoto 606-8502, Japan and
[1]Institute of Molecular Evolutionary Genetics, Pennsylvania State University, University Park, PA 16802, USA

Received April 8, 2002; Revised and Accepted May 24, 2002

ABSTRACT

A multiple sequence alignment program, MAFFT, has been developed. The CPU time is drastically reduced as compared with existing methods. MAFFT includes two novel techniques. (i) Homologous regions are rapidly identified by the fast Fourier transform (FFT), in which an amino acid sequence is converted to a sequence composed of volume and polarity values of each amino acid residue. (ii) We propose a simplified scoring system that performs well for reducing CPU time and increasing the accuracy of alignments even for sequences having large insertions or extensions as well as distantly related sequences of similar length. Two different heuristics, the progressive method (FFT-NS-2) and the iterative refinement method (FFT-NS-i), are implemented in MAFFT. The performances of FFT-NS-2 and FFT-NS-i were compared with other methods by computer simulations and benchmark tests; the CPU time of FFT-NS-2 is drastically reduced as compared with CLUSTALW with comparable accuracy. FFT-NS-i is over 100 times faster than T-COFFEE, when the number of input sequences exceeds 60, without sacrificing the accuracy.

difficulty, various heuristic methods, including progressive methods (3) and iterative refinement methods (4–6), have been proposed to date. They are mostly based on various combinations of successive two-dimensional DP, which takes CPU time proportional to N^2.

Even if these heuristic methods successfully provide the optimal alignments, there remains the problem of whether the optimal alignment really corresponds to the biologically correct one. The accuracy of resulting alignments is greatly affected by the scoring system. Thompson *et al.* (7) developed a complicated scoring system in their program CLUSTALW, in which gap penalties and other parameters are carefully adjusted according to the features of input sequences, such as sequence divergence, length, local hydropathy and so on. Nevertheless, no existing scoring system is able to process correctly global alignments for various types of problems including large terminal extension of internal insertion (8). Considerable improvements in the accuracy have recently been made in CLUSTALW (7) version 1.8, the most popular alignment program with excellent portability and operativity, and T-COFFEE (9), which provides alignments of the highest accuracy among known methods to date.

On the other hand, few improvements have been made successfully to reduce the CPU time, since the proposal of the progressive method by Feng and Doolittle (3). A high-speed computer program applicable to large-scale problems is becoming more important with the rapid increase in the number of protein and DNA sequences. In order to improve

- MAFFT incorporates a **Fast Fourier Transform (FFT)** algorithm, which speeds up the alignment process by converting the sequences into frequency space. In this space, sequence similarities are computed more quickly. This step dramatically reduces the computational time required for large-scale alignments, particularly when comparing thousands of sequences.

- MAFFT uses a p**rogressive alignment strategy with iterative refinement**, leading to better handling of complex sequences with gaps, insertions, and deletions.

- MAFFT excels at aligning a set of sequences (profile) with another existing alignment, a feature that allows the efficient merging of large datasets. This capability is crucial for incremental or hierarchical alignment tasks that require combining several smaller alignments into one comprehensive alignment.

Clustal omega

Molecular Systems Biology 7; Article number 539; doi:10.1038/msb.2011.75
Citation: ***Molecular Systems Biology*** **7: 539**

www.molecularsystemsbiology.com

molecular systems biology

REPORT

Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega

Fabian Sievers[1,8], Andreas Wilm[2,8], David Dineen[1], Toby J Gibson[3], Kevin Karplus[4], Weizhong Li[5], Rodrigo Lopez[5], Hamish McWilliam[5], Michael Remmert[6], Johannes Söding[6], Julie D Thompson[7] and Desmond G Higgins[1,*]

[1] School of Medicine and Medical Science, UCD Conway Institute of Biomolecular and Biomedical Research, University College Dublin, Dublin, Ireland, [2] Computational and Systems Biology, Genome Institute of Singapore, Singapore, [3] Structural and Computational Biology Unit, European Molecular Biology Laboratory, Heidelberg, Germany, [4] Department of Biomolecular Engineering, University of California, Santa Cruz, CA, USA, [5] EMBL Outstation—European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge, UK, [6] Gene Center Munich, University of Munich (LMU), Muenchen, Germany and [7] Département de Biologie Structurale et Génomique, IGBMC (Institut de Génétique et de Biologie Moléculaire et Cellulaire), CNRS/INSERM/Université de Strasbourg, Illkirch, France
[8] These authors contributed equally to this work
* Corresponding author. UCD Conway Institute of Biomolecular and Biomedical Research, University College Dublin, Belfield, Dublin 4, Ireland. Tel.: +353 1 716 6833; Fax: +353 1 716 6713; E-mail: des.higgins@ucd.ie

Received 23.7.11; accepted 6.9.11

Multiple sequence alignments are fundamental to many sequence analysis methods. Most alignments are computed using the progressive alignment heuristic. These methods are starting to become a bottleneck in some analysis pipelines when faced with data sets of the size of many thousands of sequences. Some methods allow computation of larger data sets while sacrificing quality, and others produce high-quality alignments, but scale badly with the number of sequences. In this paper, we describe a new program called Clustal Omega, which can align virtually any number of protein sequences quickly and that delivers accurate alignments. The accuracy of the package on smaller test cases is similar to that of the high-quality aligners. On larger data sets, Clustal Omega outperforms other packages in terms of execution time and quality. Clustal Omega also has powerful features for adding sequences to and exploiting information in existing alignments, making use of the vast amount of precomputed information in public databases like Pfam.

Molecular Systems Biology **7**: 539; published online 11 October 2011; doi:10.1038/msb.2011.75
Subject Categories: bioinformatics
Keywords: bioinformatics; hidden Markov models; multiple sequence alignment

- Clustal Omega is accurate but also **allows alignments of almost any size** to be produced. It can generate alignments of over 190 000 sequences on a single processor in a few hours.

- Clustal Omega uses a modified version of **mBed** (Blackshields et al, 2010), which has complexity of O(N log N), and which produces guide trees that are just as accurate as those from conventional methods. **mBed works by 'emBedding' each sequence in a space of n dimensions where n is proportional to log N. Each sequence is then replaced by an n element vector, where each element is simply the distance to one of n 'reference sequences.'** These vectors can then be clustered extremely quickly by standard methods such as K-means or UPGMA. In Clustal Omega, the alignments are then computed using the very accurate HHalign package (So¨ding, 2005), which aligns two profile hidden Markov models (Eddy, 1998).

T-coffe

doi:10.1006/jmbi.2000.4042 available online at http://www.idealibrary.com on IDEAL® *J. Mol. Biol.* (2000) **302**, 205–217

JMB

T-Coffee: A Novel Method for Fast and Accurate Multiple Sequence Alignment

Cédric Notredame[1,2,3]*, Desmond G. Higgins[4] and Jaap Heringa[1]

[1]*National Institute for Medical Research, The Ridgeway, Mill Hill, London NW7 1AA, UK*

[2]*ISREC, 155, Ch. des Boveresses, CH, 1066 Epalinges/s Lausanne Switzerland*

[3]*Information Genetique et Structurale, CNRS-UMR 1889 31 Ch. Joseph Aiguier 13402 Marseille, France*

[4]*Department of Biochemistry University College, Cork Ireland*

**Corresponding author*

We describe a new method (T-Coffee) for multiple sequence alignment that provides a dramatic improvement in accuracy with a modest sacrifice in speed as compared to the most commonly used alternatives. The method is broadly based on the popular progressive approach to multiple alignment but avoids the most serious pitfalls caused by the greedy nature of this algorithm. With T-Coffee we pre-process a data set of all pair-wise alignments between the sequences. This provides us with a library of alignment information that can be used to guide the progressive alignment. Intermediate alignments are then based not only on the sequences to be aligned next but also on how all of the sequences align with each other. This alignment information can be derived from heterogeneous sources such as a mixture of alignment programs and/or structure superposition. Here, we illustrate the power of the approach by using a combination of local and global pair-wise alignments to generate the library. The resulting alignments are significantly more reliable, as determined by comparison with a set of 141 test cases, than any of the popular alternatives that we tried. The improvement, especially clear with the more difficult test cases, is always visible, regardless of the phylogenetic spread of the sequences in the tests.

Keywords: pair-wise alignment; progressive alignment; local alignment; global alignment; multiple sequence alignment

- One of the main reasons T-Coffee is highly accurate is its **consistency-based scoring method**. Unlike traditional MSA algorithms, which align sequences directly, T-Coffee checks the "consistency" of the alignments with respect to multiple pairwise comparisons.

- T-Coffee builds a **library of pairwise alignments** (including global and local alignments), which serves as a guide during the multiple sequence alignment process. Each alignment is checked against this library to ensure that each alignment step agrees with the majority of pairwise alignments in the set. This consistency improves the reliability of the final alignment, reducing the impact of errors that might arise from aligning distant sequences directly.

T-coffee's consistency checking

b)Primary Library

```
SeqA GARFIELD THE LAST FAT CAT
SeqB GARFIELD THE FAST CAT ---
```
Prim. Weight = 88

```
SeqA GARFIELD THE LAST FA-T CAT
SeqC GARFIELD THE VERY FAST CAT
```
Prim. Weight = 77

```
SeqA GARFIELD THE LAST FAT CAT
SeqD -------  THE ---- FAT CAT
```
Prim. Weight =100

```
SeqB GARFIELD THE ---- FAST CAT
SeqC GARFIELD THE VERY FAST CAT
```
Prim Weight = 100

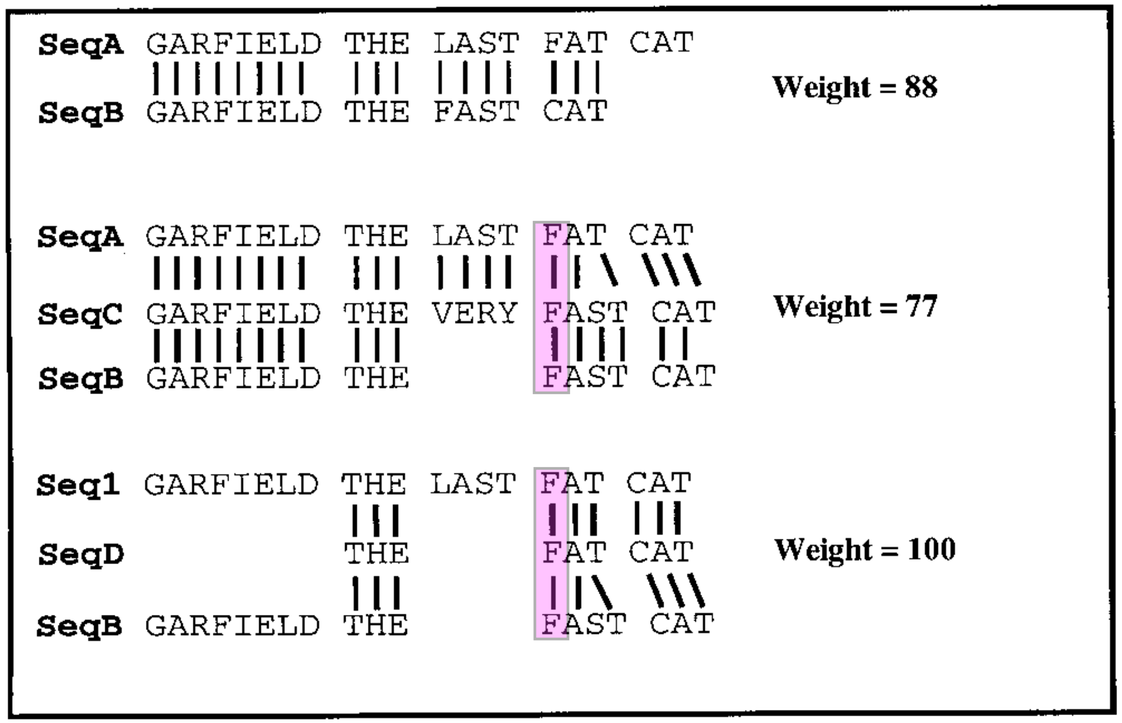
```
SeqB GARFIELD THE FAST CAT
SeqD -------  THE FA-T CAT
```
Prim. Weight = 100

```
SeqC GARFIELD THE VERY FAST CAT
SeqD -------  THE ---- FA-T CAT
```
Prim. Weight = 100

- Primary weight between a pair of sequences is the sequence identity between them.
- Sequence identity = Matched residues/aligned residues
- Each pair of residues will be assigned the Primary weight.

T-coffee's consistency checking

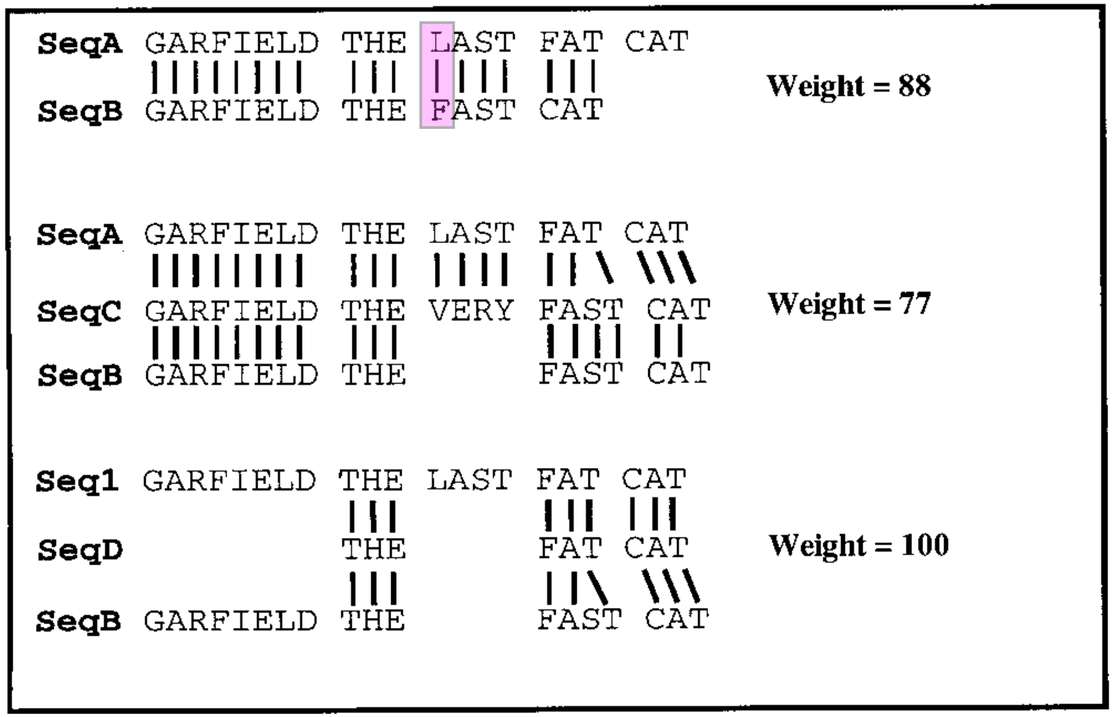
c)Extended Library for seq1 and seq2



- For A-B: F(AT):F(AST)
- In A-B: W(F:F) = 0
- In A-C-B: W(F:F) = min(77(w(A,C), 100(w(C,B))) = 77
- In A-D-B: W(F:F) = min(100(w(A,D), 100(w(D,B))) = 100
- Final A-B: F(AT):F(AST) = 77 + 100 = 177

T-coffee's consistency checking

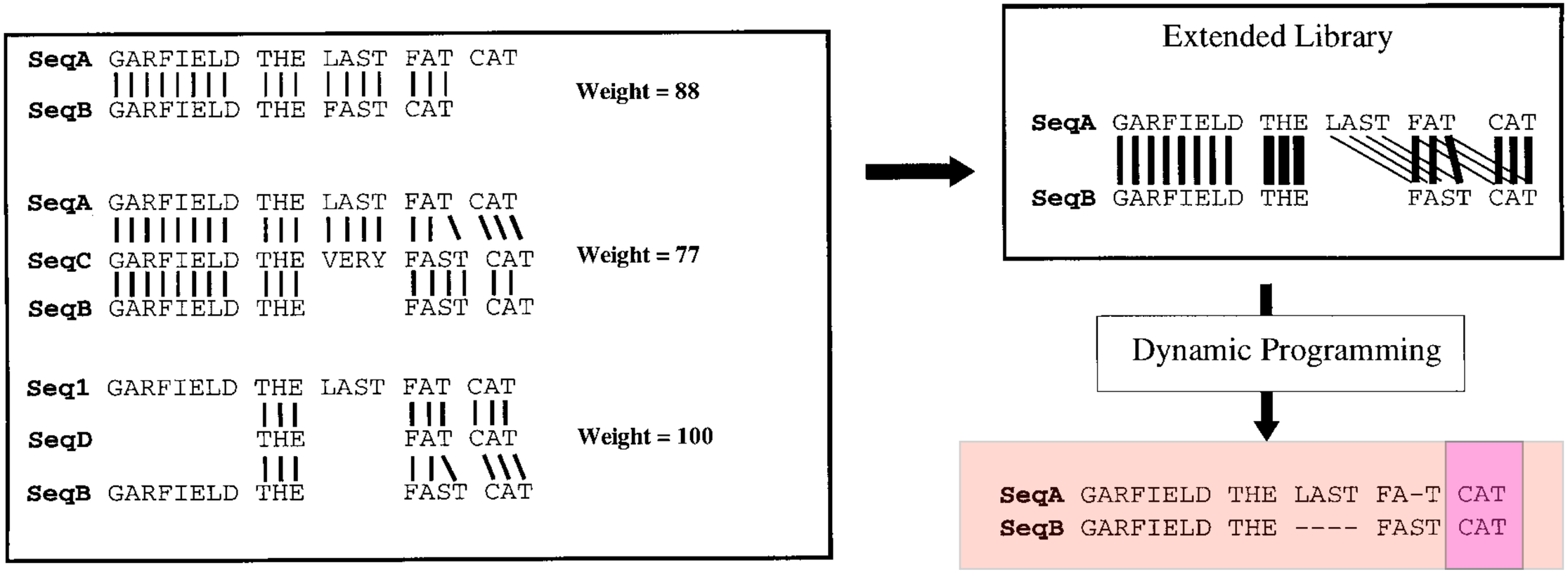
c)Extended Library for seq1 and seq2



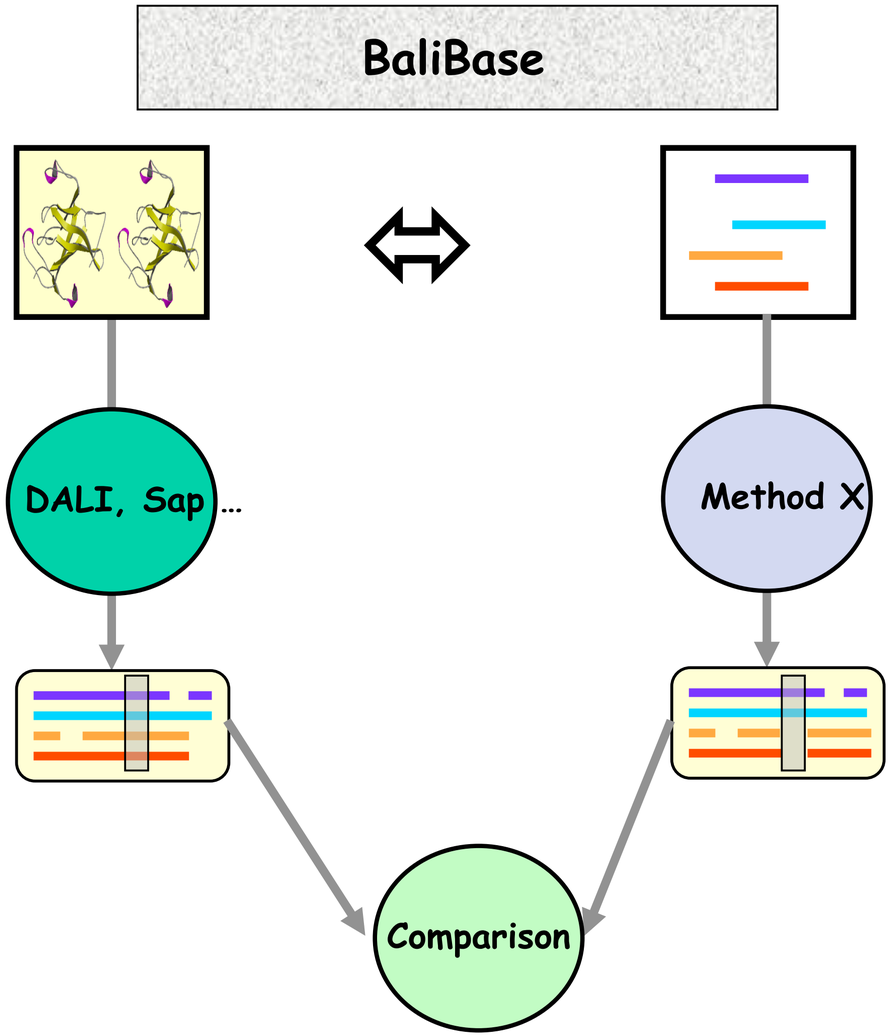
- For A-B: L(AST):F(AST)
- In A-B: W(L:F) = 88
- In A-C-B: W(L:F) = min(0,0) = 0
- In A-D-B: W(L:F) = min(0,0) = 0
- Final A-B: L(AST):F(AST) = 88 + 0 = 88

T-coffee's consistency checking

c)Extended Library for seq1 and seq2



Direct assessment of the quality of an MSA



Structure alignment method

Sequence alignment method

Indirect assessment of the quality of a MSA

- Quality measures of MSA
  - Number of matches (multiple longest common subsequence score)
  - Entropy score
  - Sum of pairs (SP-Score)

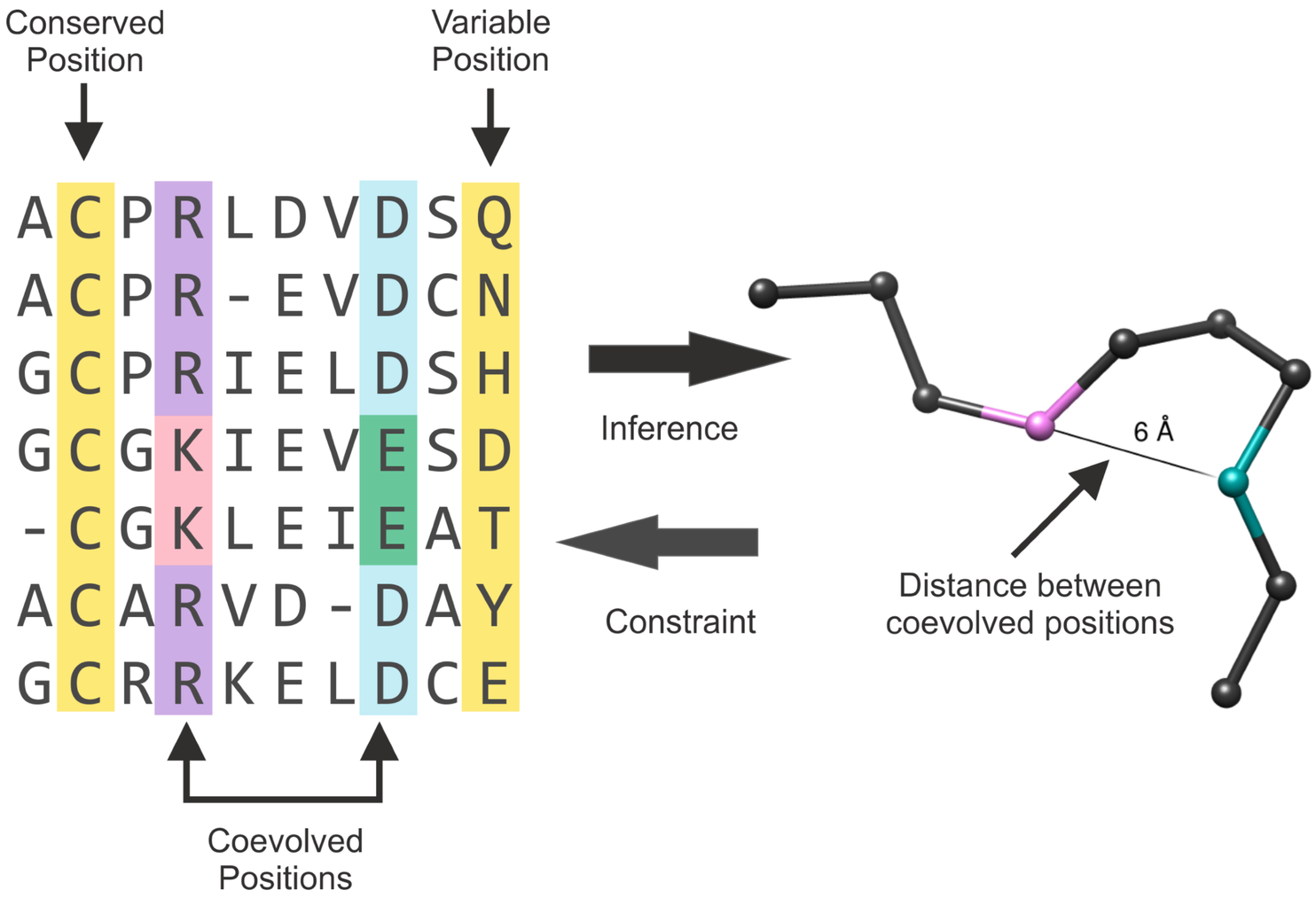
Entropy score

The entropy $\mathcal{H}(P)$ of a probability distribution $P = (p_1, \ldots, p_n)$ is defined by

$$\mathcal{H}(P) = \mathbf{E}(-\log P) = -\sum_{i=1}^{n} p_i \log p_i. \tag{B.1}$$

The units used to measure entropy depend on the base used for the logarithms. When the base is 2, the entropy is measured in bits. The entropy measures the prior uncertainty in the outcome of a random experiment described by P, or the information gained when the outcome is observed. It is

Entropy score

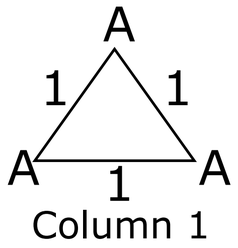


Sum of Pairs Score (SP-Score)

a_1 ATG–C–AAT
· A–G–CATAT
a_k ATCCCATTT

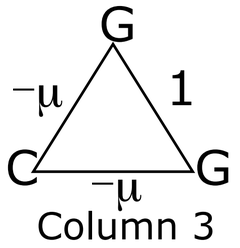
To calculate each column:

$$s'(a_1 \ldots a_k) = \sum_{i,j} s^*(a_i, a_j) \longleftarrow \binom{n}{2} \text{ Pairs of Sequences}$$



Score=3

Column 1

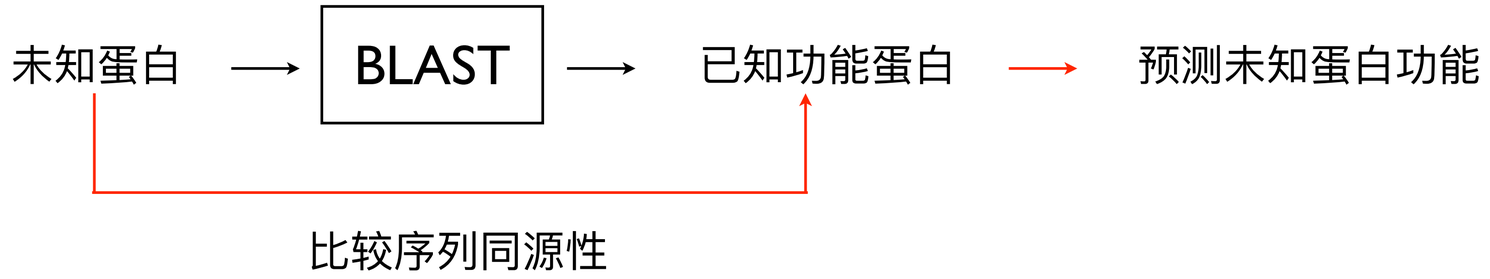


Column 3

Sequence database search

Sequence database search

- Given a query sequence
- Goal: identify all sequences in the database that are homologous to the query sequence, and if the homologous sequence's function is known, infer the function of the query sequence.

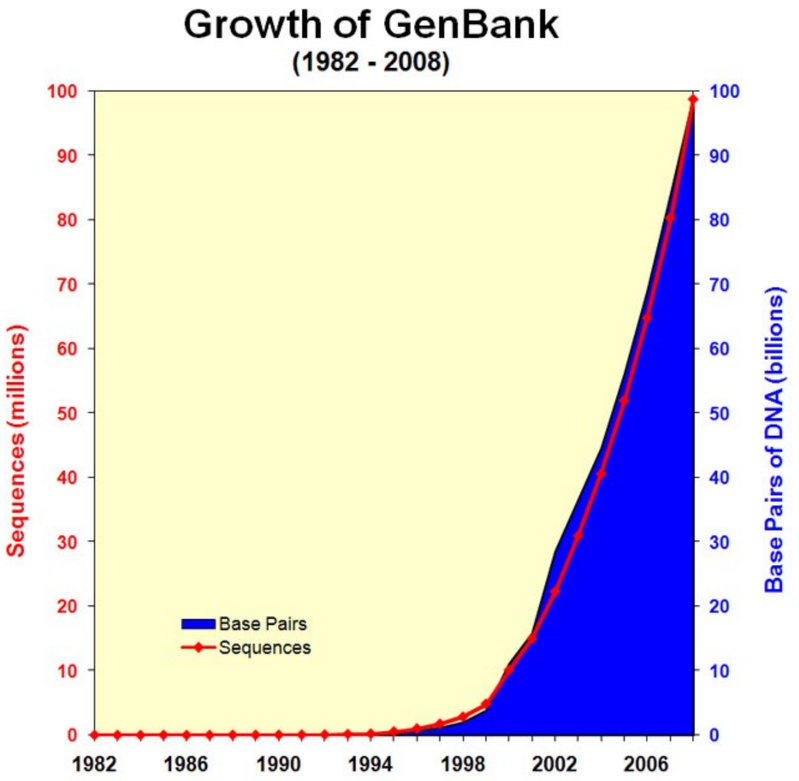


Sequence database search

- Approach: compare the query sequence with each of the sequence in the database by pairwise sequence comparison, and determine the homologous sequences whose alignment scores are above a certain threshold.

- Problem: The time complexity of dynamic programming is *O(nm)* for a pair of sequences. Given a total number of *k* sequences in the database, the complexity would be *O(knm).* However, k is increasing exponentially over the years.

- Dynamic programming for all pairs of sequences is computationally impossible.



GenBank has grown exponentially and currently doubles in size approximately every 2 years

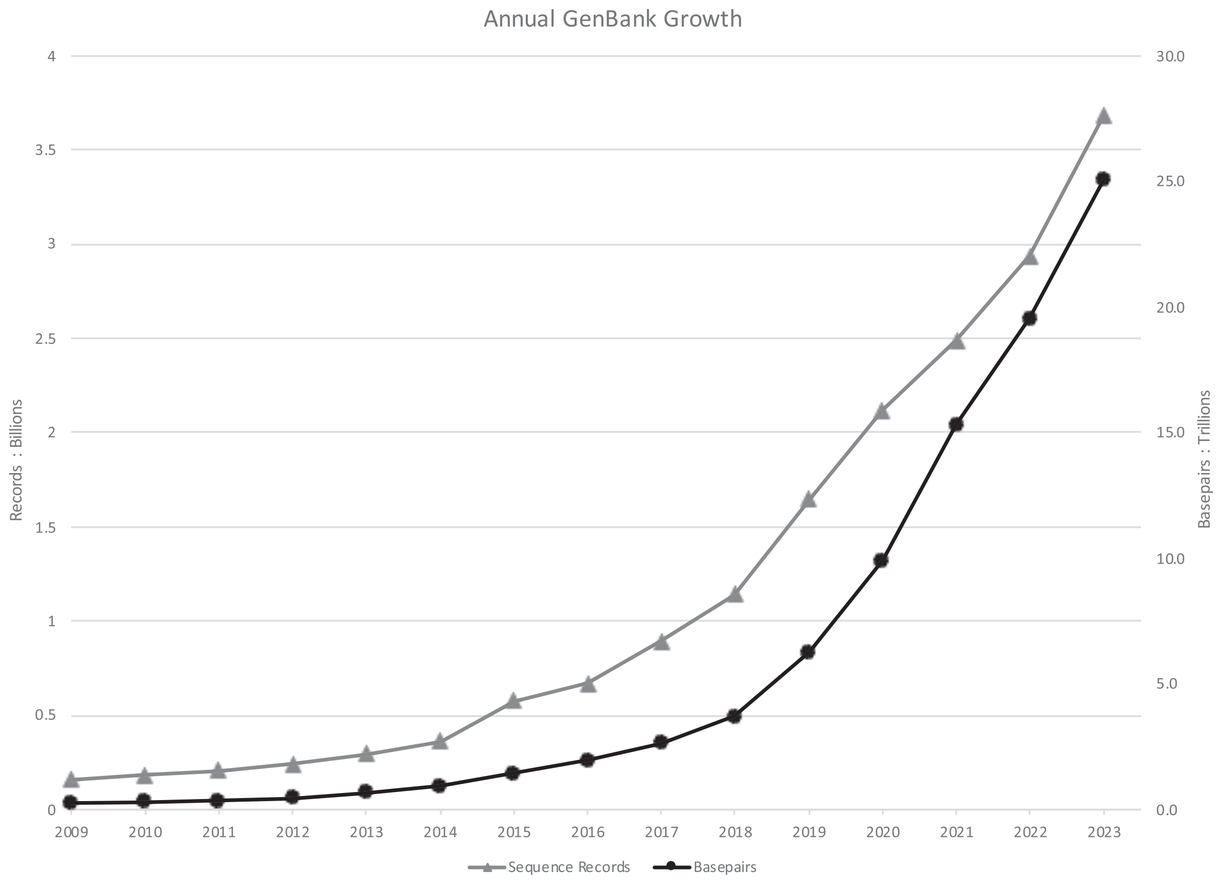
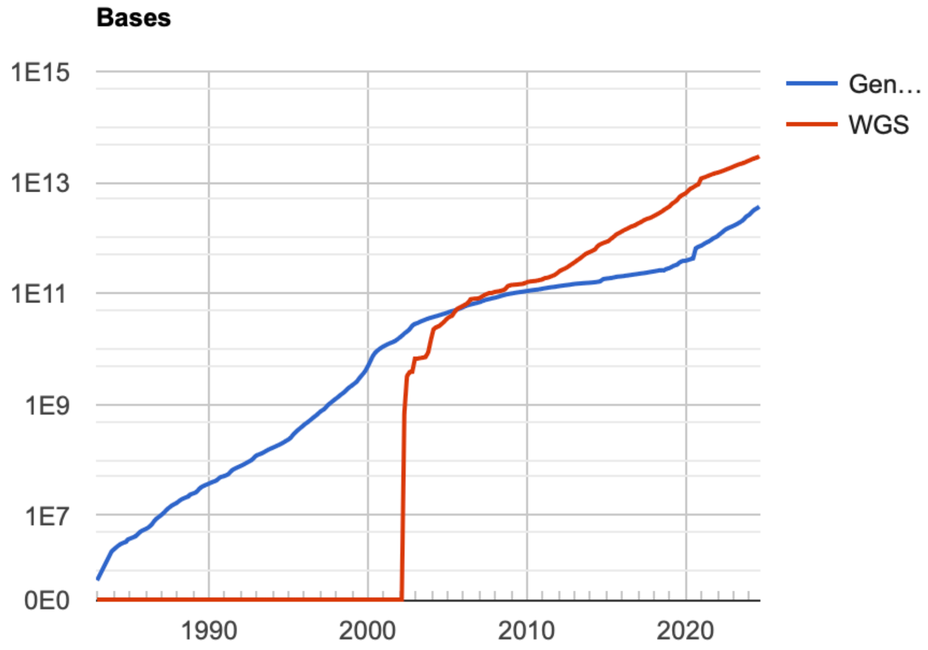
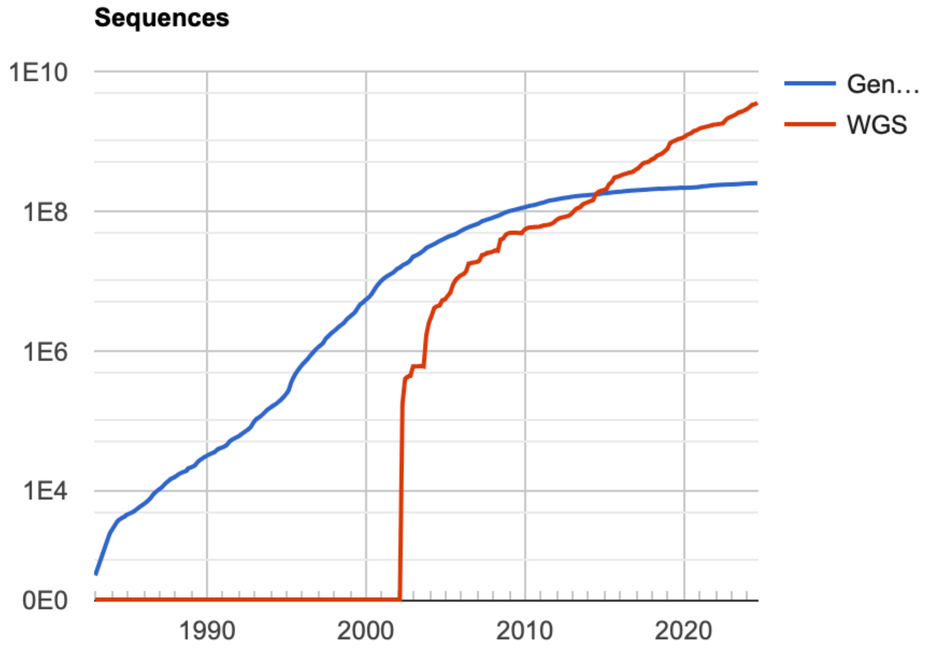


Figure 1. Growth of GenBank recorded in both base pairs (triangles) and the number of sequence records (circles). Each point represents the GenBank release in August of each year, starting with release 173 (August 2009).

GenBank and WGS Statistics





The heuristic approach to solve the database search problem

- A heuristic technique, often called simply a heuristic, is any approach to problem solving or self-discovery that employs a practical method, not guaranteed to be optimal, perfect, or rational, but instead sufficient for reaching an immediate goal. Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution.

- Basic assumption for sequence database search: most sequences in the database are not homologous to the query sequence.

- The heuristic idea: avoiding comparison with most sequences.

The heuristic approach to solve the database search problem

- Knowledge about pairwise sequence comparison between homologous sequences: two homologous sequences must have some common subsequences of absolute identity (short lengths of exact matches).

- No need to compare with most sequences

  - First, identify very short exact matches.

  - Next, the best hits from the first step are extended to longer regions of similarity.

  - Finally, the best hits are optimized.

BLAST

- BLAST, the Basic Local Alignment Search Tool (Altschul et al., 1990), is perhaps the most widely used bioinformatics tool ever written. It is an alignment heuristic that determines “local alignments” between a *query* and a *database*. It uses an approximation of the Smith-Waterman algorithm.

- BLAST consists of two components: a search algorithm and computation of the statistical significance of the alignments.

BLAST

Step 1.A: Compile a List of Words

§ Given a word length ***w***...

- Word length is analogous to window length

$$w = 3$$

VHR = first word

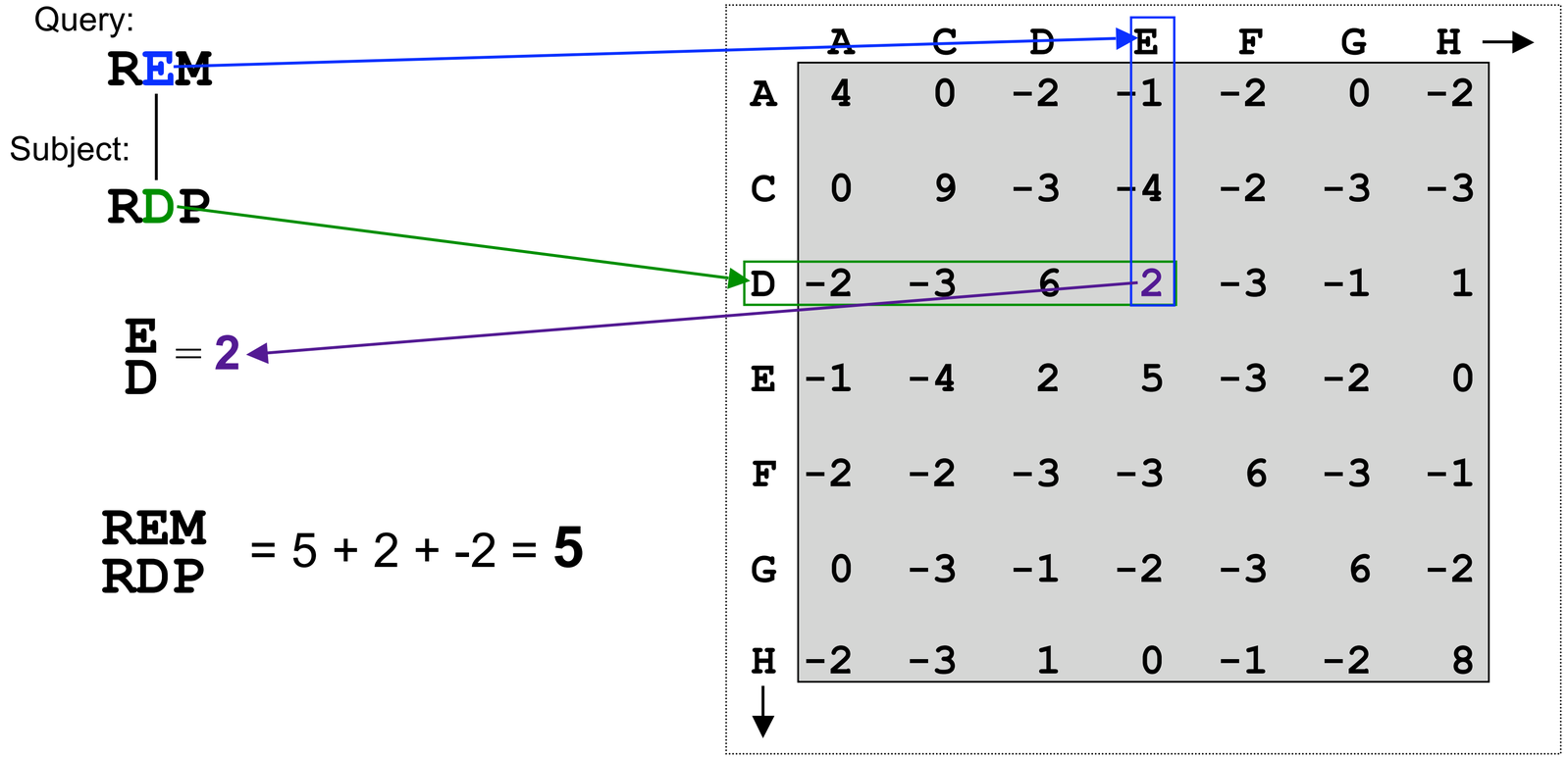
HRE = second word

REM = third word

BLAST

Step 1.B: Choose a Scoring Matrix to Calculate Word Scores

§ Choose an appropriate substitution matrix to calculate word scores



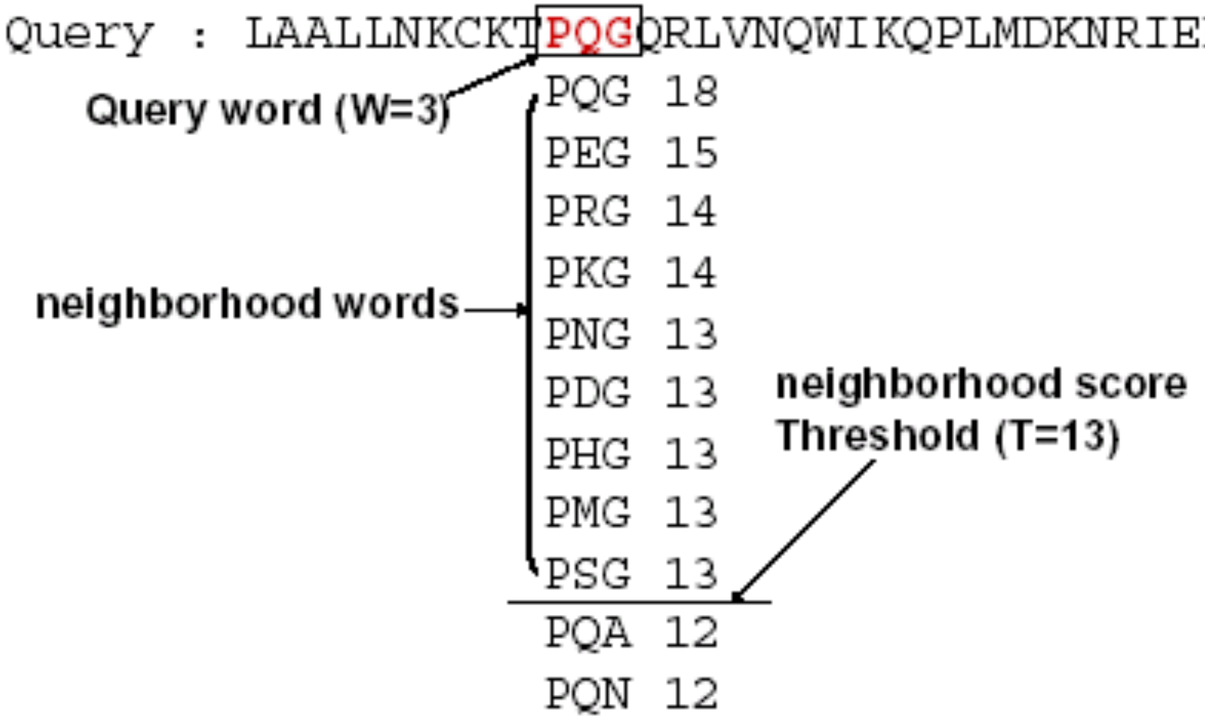
BLAST

Step 1.C: Compile List of High-Scoring Words

§ Calculate score of all possible variations for each word

§ Select word variations that have score higher than ***T***

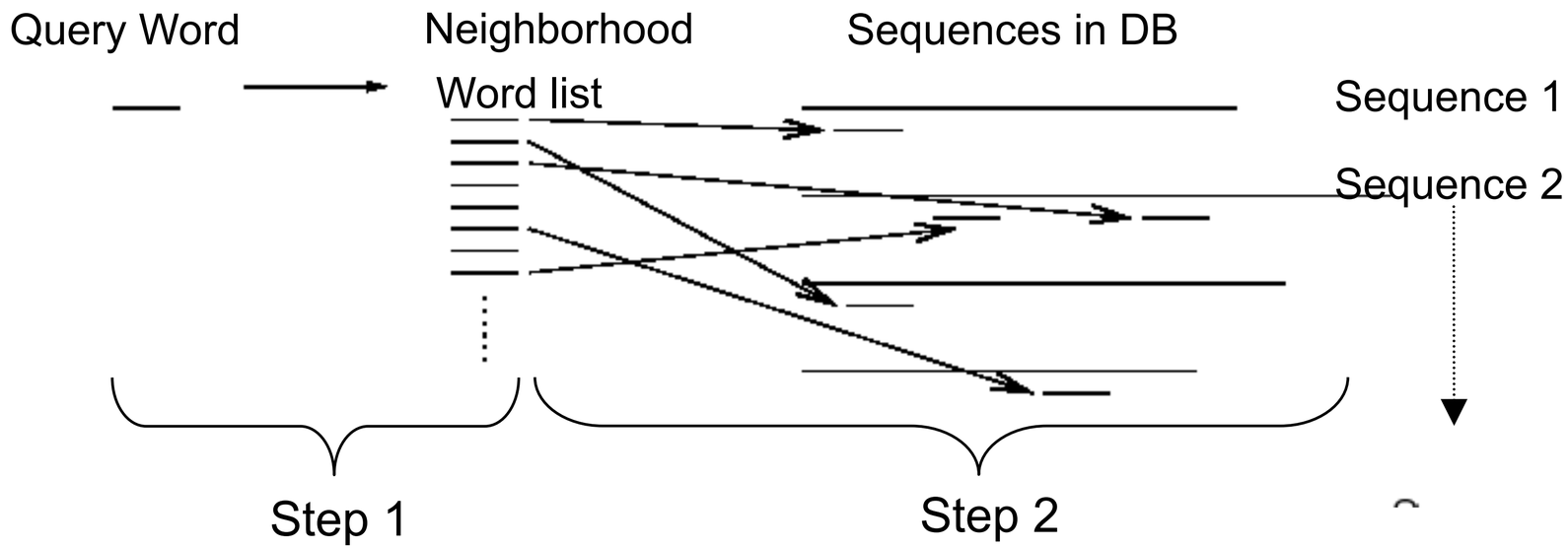
- ***T*** is chosen empirically



BLAST

- Step 2: Scanning DB

  For each words list, identify all exact matches with DB sequences



The purpose of Step 1 and 2 is as same as FASTA

BLAST

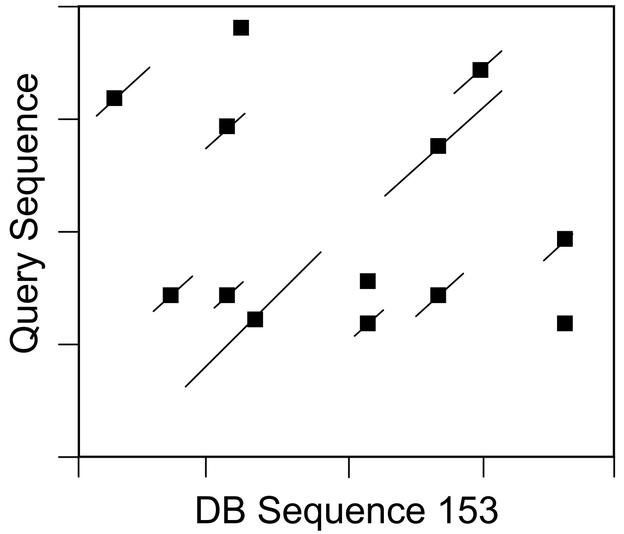
Step 3: Extend Hits to Differentiate Random Hits from Meaningful Hits

§ Extend hits by summing residue pairs from both sides of the word boundary

§ Extension stops when score drops below a threshold ($\mathbf{X_U}$) of the best score yet observed

§ Original BLAST extended each hit to find ungapped segments

§ All extended hits above the minimum score ***S*** are reported



Starting Score = 15

Query:	V	H	R	E	M	H	A	R	T	S	P	L	R	P	L
	-2	-2	2				1	1	0	0	7	2	-3	-4	-4
DB 153:	K	W	K	D	M	H	S	Q	A	D	P	I	I	F	G

BLAST

Step 3.B: Gapped BLAST -- Triggering Extensions

For Gapped BLAST:

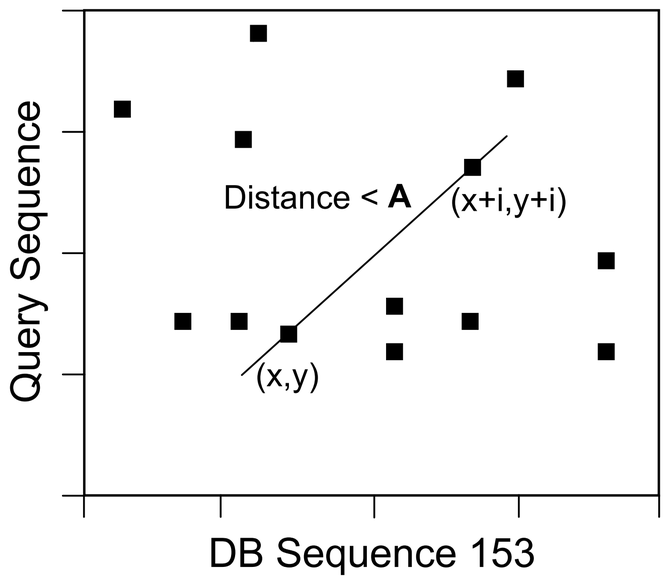
§ Extensions triggered only by two or more hits on the same diagonal

§ Hits must also be less than distance ***A*** from each other to trigger extensions

§ Serves to reduce the number of extensions

§ Typically the Dynamic Programming sequence alignment method is used to find gapped alignment

§ Gapped BLAST is more sensitive and selective than the original, ungapped BLAST



Ungapped vs. gapped alignment in BLAST

- **Ungapped alignment**: An ungapped alignment does not allow gaps (insertions or deletions) and focuses on finding the longest contiguous sequence of identical or similar residues between two sequences, making it faster to compute. BLAST uses ungapped alignments in its initial seeding phase, extending high-scoring matches (seeds) without gaps, and if these alignments meet a score threshold, BLAST proceeds with a gapped extension.

- **Gapped alignment**: A gapped alignment allows gaps (insertions or deletions) to account for indels, making it more accurate for aligning sequences that have evolved over time. If an ungapped match meets a certain score threshold, BLAST performs a gapped extension to improve the alignment. Though more computationally intensive, gapped alignment is used in the final stage to refine and score the alignment.

The raw score of a BLAST alignment

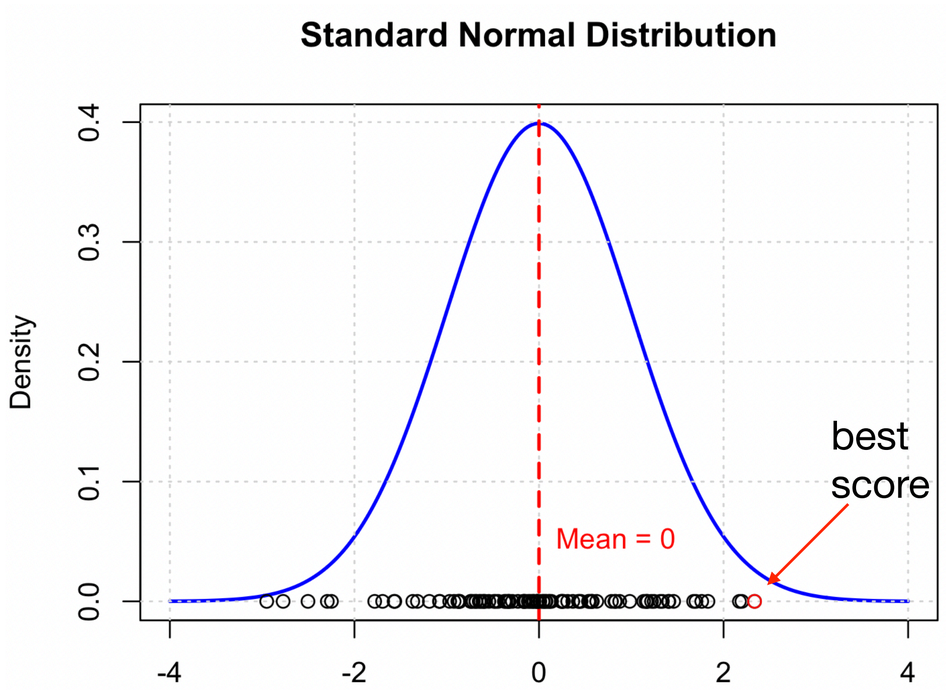
$$S = \sum(\text{score of aligned residues}) - (\text{gap penalties})$$

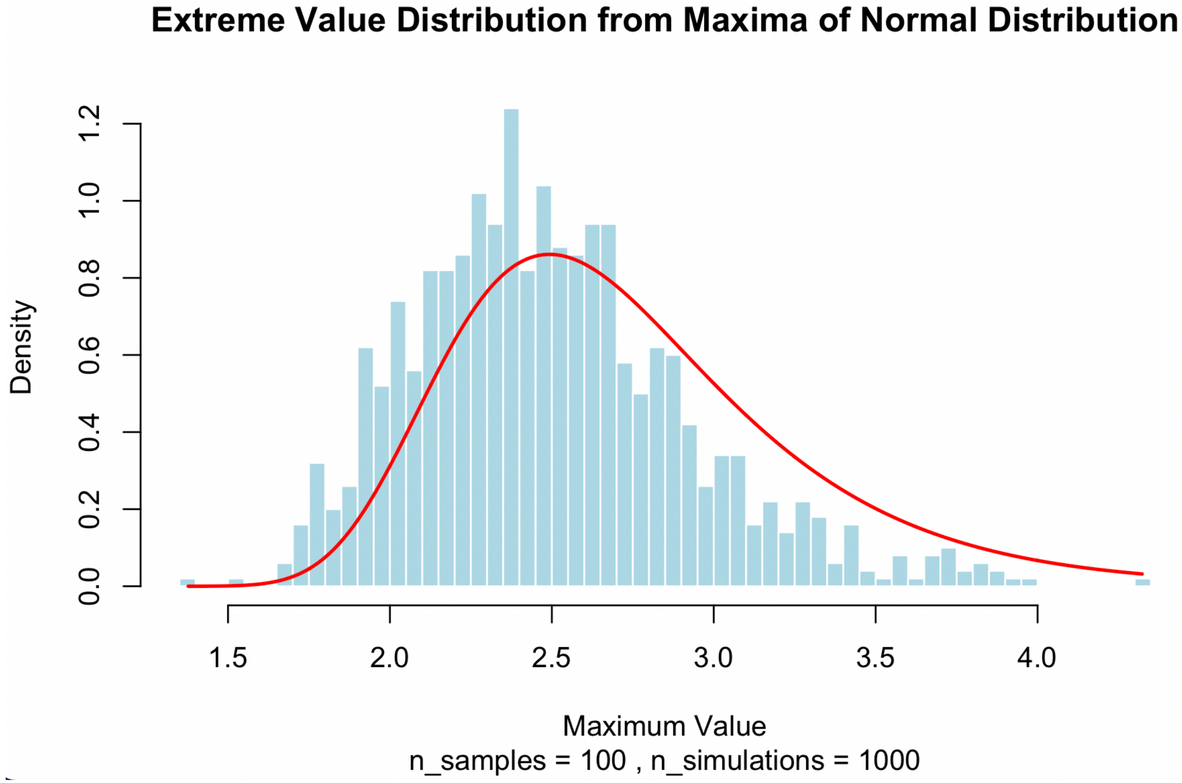
BLOSUM62 matrix

Example: Gap opening = -10. Gap extension = -1.

- The raw score in BLAST reflects the degree of similarity between the query sequence and a hit sequence, but it does not directly indicate the significance of homology on its own.
  - Raw Score Depends on Length.
  - Raw scores are not normalized for different scoring matrices, database sizes, or sequence composition, which makes it difficult to compare results from different searches or databases.
- To assess biological significance and potential homology, use the bit score and E-value, which normalize and adjust for factors like random matches and alignment length.

Extreme Value Distribution

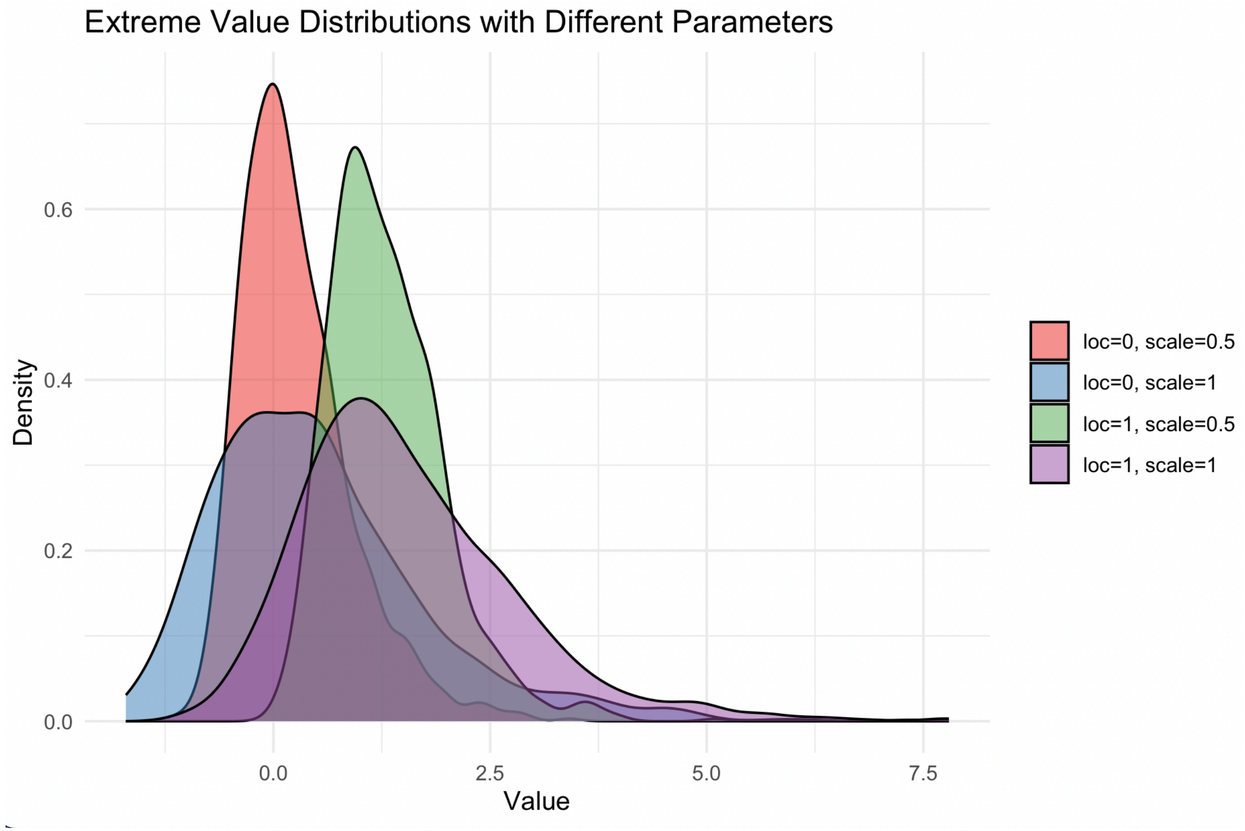




- "Randomly sample 100 values from a standard normal distribution (mean = 0, sd = 1) and identify the best score.

- Repeat this process 1000 times. What is the resulting distribution of these best scores?"

Extreme Value Distribution



$$f(x;\mu,\beta) = \frac{1}{\beta}\exp\left(-\frac{x-\mu}{\beta} - \exp\left(-\frac{x-\mu}{\beta}\right)\right)$$

- **Location Parameter** μ shifts the entire distribution along the x-axis. If μ increases, the distribution shifts to the right. In BLAST: μ is influenced by the search space (related to the parameter K) and shifts the expected scores accordingly.

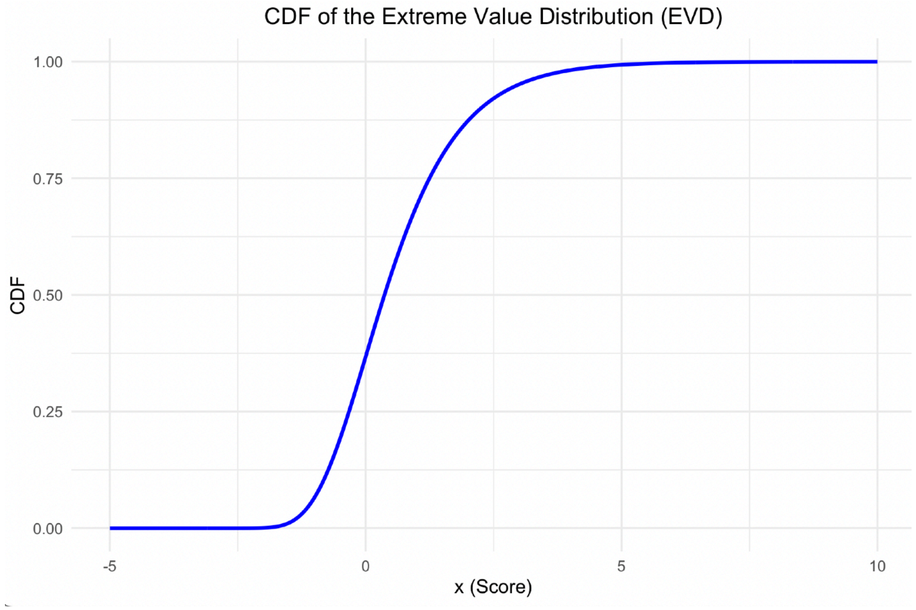
- **Scale Parameter** β controls the spread (or the "width") of the distribution. A larger β leads to a wider, more spread-out distribution, meaning there's more variability in the extreme values. In BLAST: The scale parameter β is analogous to λ, the scaling factor derived from the scoring matrix and gap penalties. It determines how quickly the probability of observing a high alignment score decreases.

Extreme Value Distribution

- The EVD describes the probability of extreme deviations (either maximum or minimum values) from a dataset. Specifically, the right tail of the distribution (for maxima) represents unusually high values, and that’s what the EVD models.

- In BLAST, each sequence alignment generates a score based on how well the query sequence matches a target sequence. Because we’re interested in the best matches (those with the highest scores), BLAST uses an extreme value distribution to model these highest alignment scores.

Extreme Value Distribution



$$\mathrm{P}(S \geq S_{\mathrm{obs}}) = 1 - \mathrm{CDF}(S_{\mathrm{obs}})$$

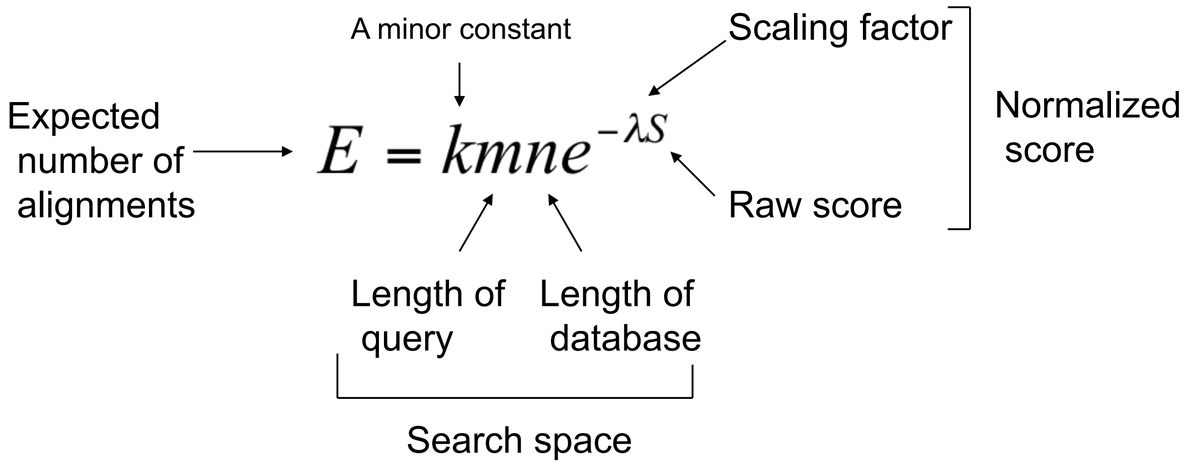
The probability of finding one such alignment by random chance, given the EVD distribution.

$$E = m \times n \times \mathrm{P}(S \geq S_{\mathrm{obs}})$$

- BLAST searches involve multiple comparisons between the query sequence and the sequences in the database.
- *m* is the length of the query sequence, meaning there are potentially *m* different starting positions in the query sequence where alignments can be initiated.
- *n* is the total length of the sequences in the database, which represents the number of residues or nucleotides available to match against the query. This expands the search space significantly.
- The E-value adjusts for the number of these comparisons, and estimates how many alignments of the given score would occur by chance across the entire search.

The BLAST E-value

The Karlin-Altschul Equation



- λ is the scaling factor that determines the steepness of the score distribution. It directly affects how quickly the probability of achieving high alignment scores decreases. It is derived from the scoring matrix (e.g., BLOSUM, PAM) and gap penalties used in the alignment process.
- K is related to the search space size and corresponds to the location parameter in EVD. As K increases in BLAST (due to a larger database or longer query sequence), the distribution of alignment scores is shifted to account for the increased probability of finding high-scoring alignments by chance.
- Large e-value (5 or 10) indicates that the alignment is probably by chance.
- E-values of 0.1 or 0.05 are typical cutoff values for database search.

Raw score vs. bit score

$$S' = \frac{\lambda S - \ln K}{\ln 2}$$

$$E = mn\,2^{-S'}$$

- Raw scores: The raw score is calculated directly from the alignment of two sequences using the substitution matrix (such as BLOSUM62 or PAM) and any penalties for gaps (insertions or deletions).
  - The raw score is dependent on the scoring matrix and gap penalties used.
  - It does not adjust for the size of the database or query sequence.
  - A higher raw score indicates a better alignment.
- Bit scores: The bit score is a normalized version of the raw score that accounts for the statistical properties of the scoring system, including the size of the search space. This makes bit scores comparable across different searches, even if they use different databases, scoring matrices, or gap penalties.

E-value vs. P-value

$$P = 1 - e^{-E}$$

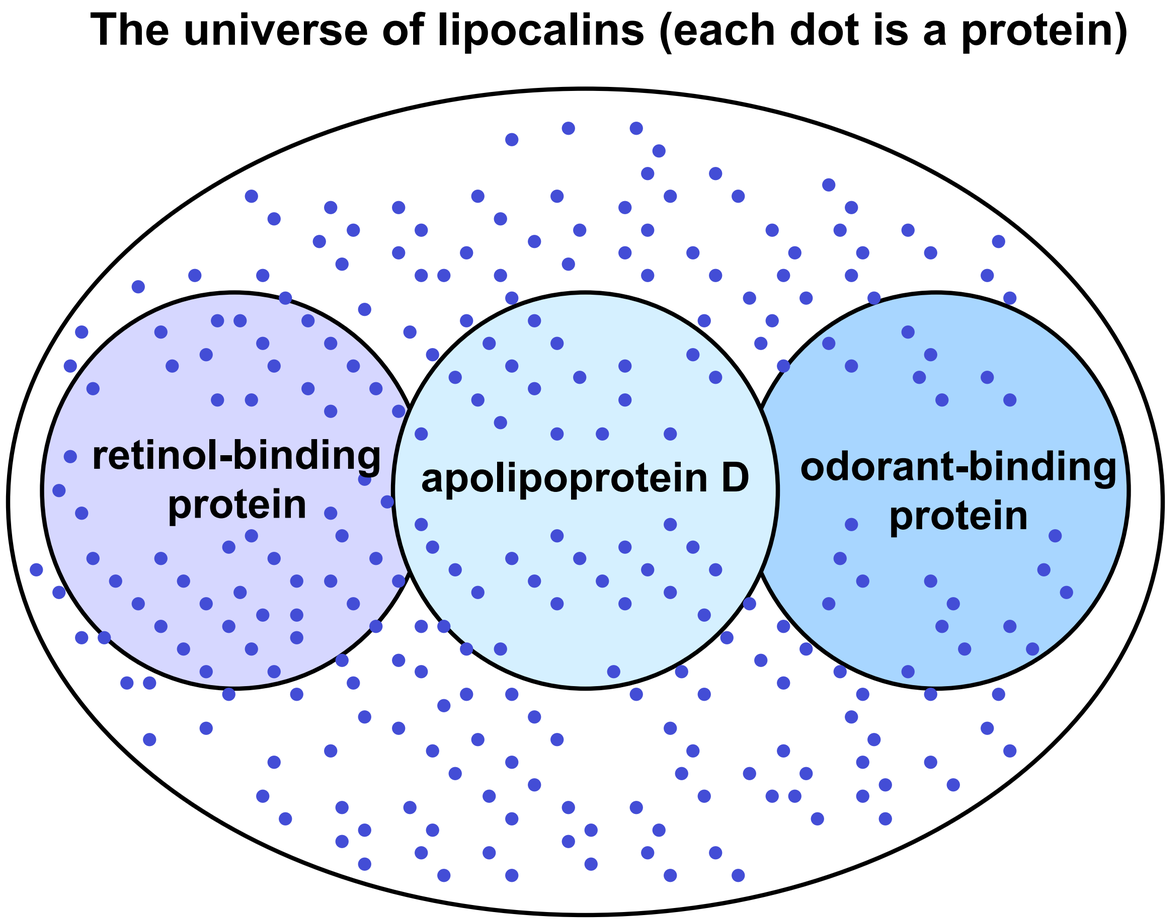
$$E = -\ln(1 - P)$$

Note that E ~= P if either value < $1e^{-5}$

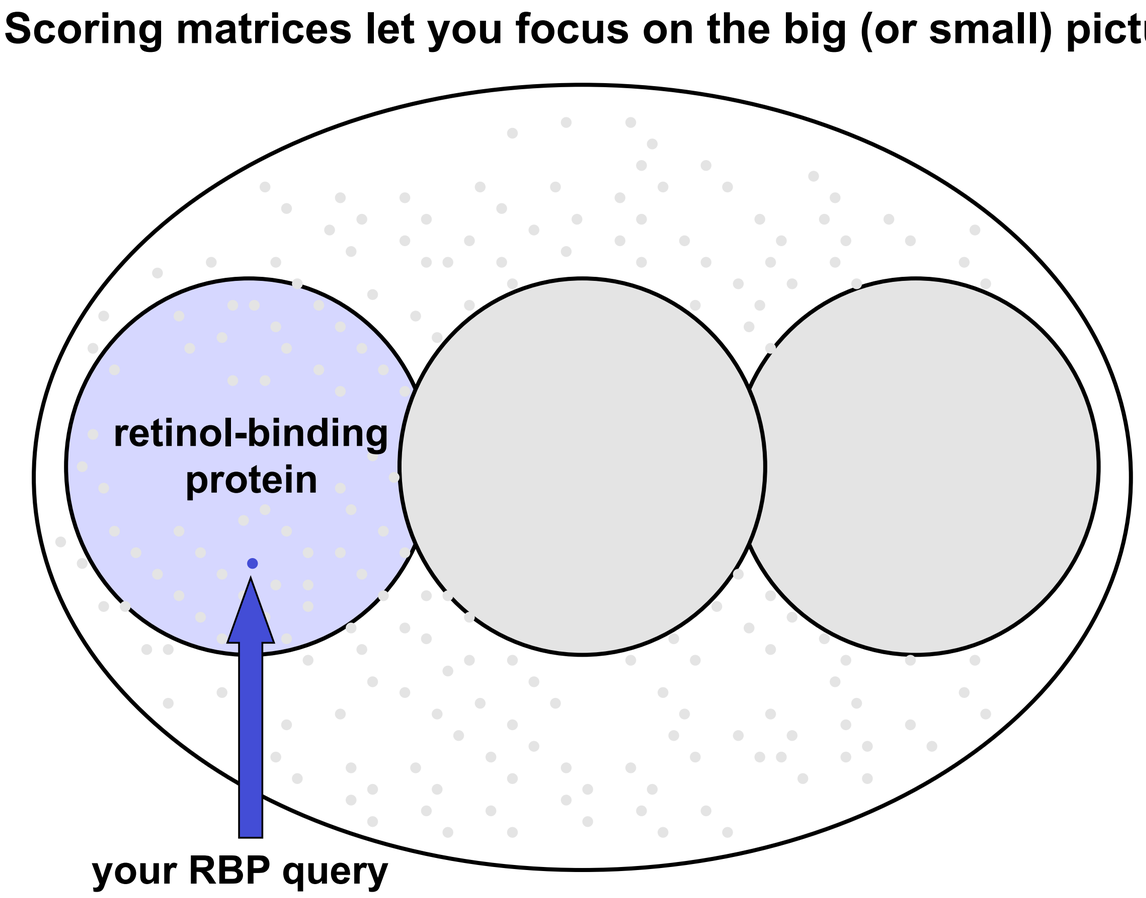
- E = 5, E = 10 versus P = 0.993 and P = 0.99995
- When E < 0.01 P-values and E-values are nearly identical

- **E-value** is used in BLAST because it directly accounts for the size of the database being searched. As the database size increases, the likelihood of finding a match purely by chance increases, and the E-value reflects this.
- **P-value**, on the other hand, gives a measure of significance without adjusting for database size. In sequence searches, the size of the database significantly affects the probability of random matches, so the E-value is preferred.
- The **E-value** is conceptually similar to the P-value but more practical for large-scale database searches since it estimates the number of expected hits by chance, allowing users to filter results accordingly.

The BLAST result of a typical search

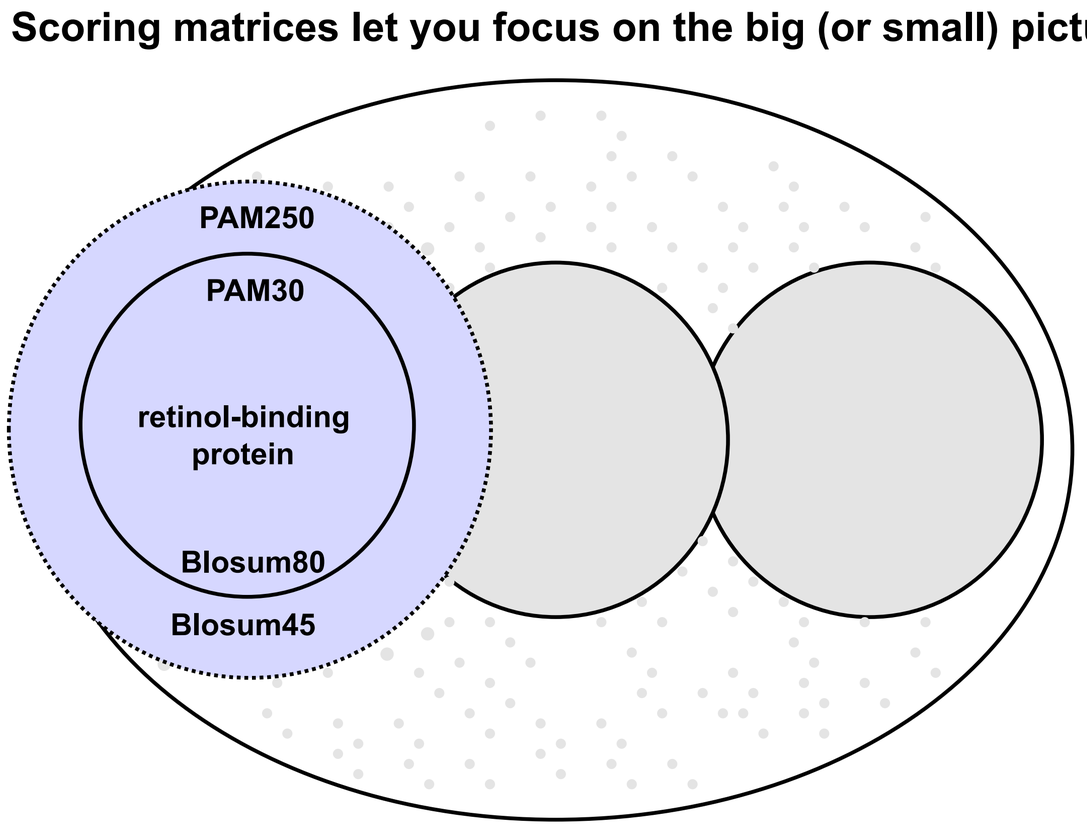


The BLAST result of a typical search



BLAST will find sequences that are homologous to the query sequence mostly within the family level

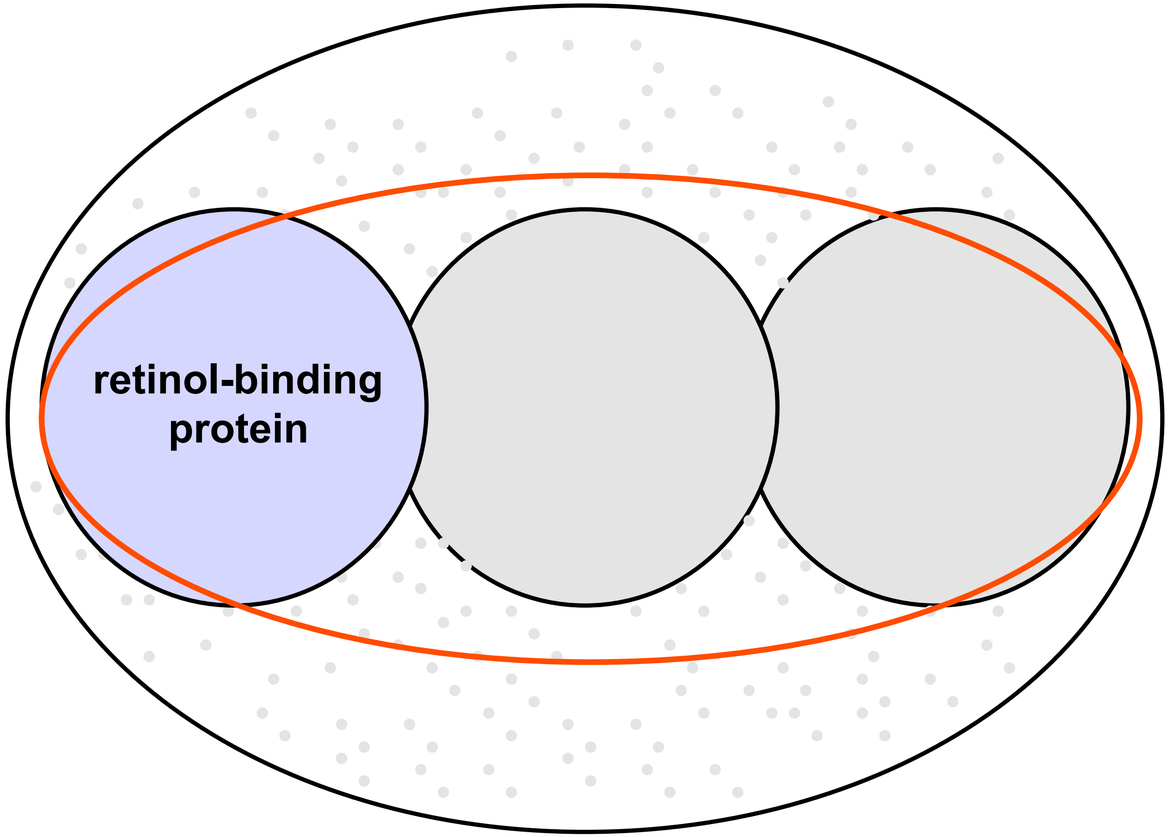
The BLAST result of a typical search



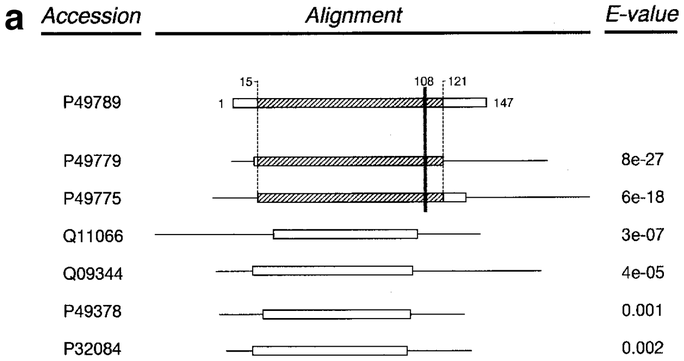
Choosing matrices at distant evolutionary time may find some distantly related homologous sequences by BLAST. However, it is unlikely to find distantly related homologous sequences at superfamily level by BLAST.

How can we improve BLAST search to detect more distantly related homologous proteins?

PSI-BLAST generates scoring matrices more powerful than PAM or BLOSUM



A multiple sequence alignment can be prepared from BLAST output



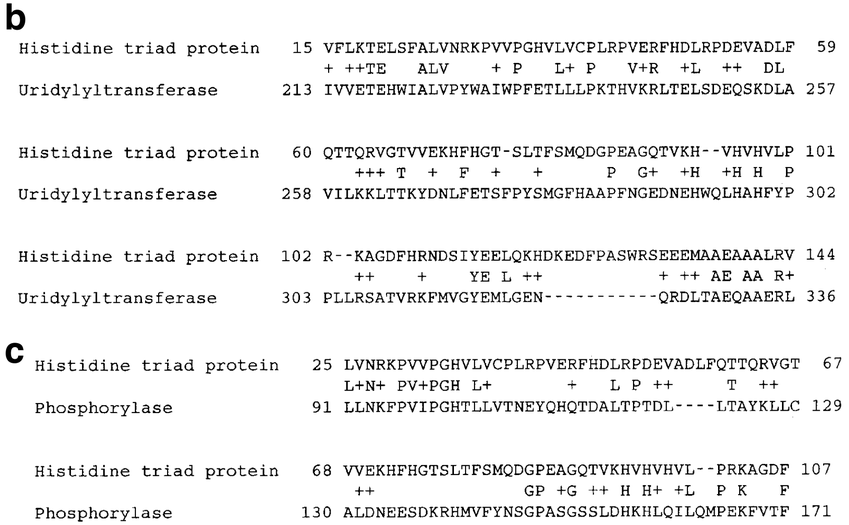
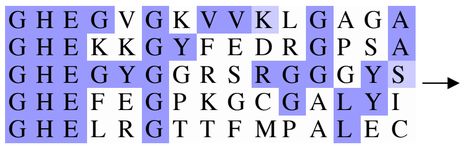


Figure 5. (**a**) The multiple alignment generated by PSI-BLAST when the human fragile histidine triad (HIT) protein (61) (SWISS-PROT accession no. P49789) is compared to SWISS-PROT. All pairwise local alignments have *E*-value ≤0.01, and are identified in SWISS-PROT as belonging to the HIT family. Thick bars within the six database sequences represent segments that align with various segments from the query. In constructing sequence weights for the indicated multiple alignment column, corresponding to residue 108 of the query, only the shaded portions of the multiple alignment are used. (**b**) A local alignment of the human HIT protein and *H.influenzae* galactose-1-phosphate uridylyltransferase (63) (SWISS-PROT accession no. P31764). In its first position-specific iteration, PSI-BLAST gives this alignment a score of 45.4 bits, corresponding to an *E*-value of 4×10^{-5}. '+' symbols reflect positive BLOSUM-62 matrix scores, even though a position-specific matrix is used to construct the alignment. (**c**) A local alignment of the human HIT protein and yeast 5′,5‴-P1,P4-tetraphosphate phosphorylase I (64) (SWISS-PROT accession no. P16550). In its second position-specific iteration, PSI-BLAST gives this alignment a score of 43.4 bits, corresponding to an *E*-value of 2×10^{-4}.

Multiple alignment M has the same length (column length) as the query (Pair-wise alignment columns with query involves inserted gap are ignored).

A position Specific Scoring Matrix (PSSM) can be calculated from the multiple sequence alignment

- A *PSSM* is based on the *frequencies* of each residue in a specific position of a multiple alignment.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	0	0	0	0	0	0	0	0	0	0	0	2	1	0	2
C	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	5	0	1	0	0	0	1	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
G	5	0	0	2	0	5	1	0	1	0	2	3	1	1	0
H	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
K	0	0	0	1	1	0	1	1	0	1	0	0	0	0	0
L	0	0	0	1	0	0	0	0	0	0	1	0	2	0	0
M	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	1	0	0	1	0	1	1	0	0	0	0
S	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
T	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
V	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0

- Column 1: $f_{A,1} = \frac{0}{5} = 0$, $f_{G,1} = \frac{5}{5} = 1$, ...
- Column 2: $f_{A,2} = \frac{0}{5} = 0$, $f_{H,2} = \frac{5}{5} = 1$, ...
- ...
- Column 15: $f_{A,15} = \frac{2}{5} = 0.4$, $f_{C,15} = \frac{1}{5} = 0.2$, ...

The inclusion of pseudo counts when preparing the PSSM

The need of pseudocounts.

- Some observed frequencies may equal 0, which is a consequence of the **limited number** of sequences that is present in a MSA.
- An observed frequency of 0 will exclude the corresponding residue at this position.
- To avoid such situation, pseudocount, a small number, is usually needed.
- One simple use of pseudocounts is to add 1.

Column 1: $f'_{A,1} = \frac{0+1}{5+20} = 0.04$, $f'_{G,1} = \frac{5+1}{5+20} = 0.24$, ...

Column 2: $f'_{A,2} = \frac{0+1}{5+20} = 0.04$, $f'_{H,2} = \frac{5+1}{5+20} = 0.24$, ...

...

Column 15: $f'_{A,15} = \frac{2+1}{5+20} = 0.12$, $f'_{C,15} = \frac{1+1}{5+20} = 0.08$, ...

Construction of the PSSM by PSI-BLAST

The log odds for amino acid *i* at column c is computed by:

$\log (Q_i/P_i)$ Qi: estimated probability for residue i to be found in column *C*
Pi: the probability for residue i in database

The log odds score is scaled by: $Score_i = \frac{1}{\lambda_u} ln(\frac{Q_i}{P_i})$

observed frequency of residue *i*

pseudocount frequency of residue *i*

$g_i = \sum_j \frac{f_j}{P_j} q_{ij}.$

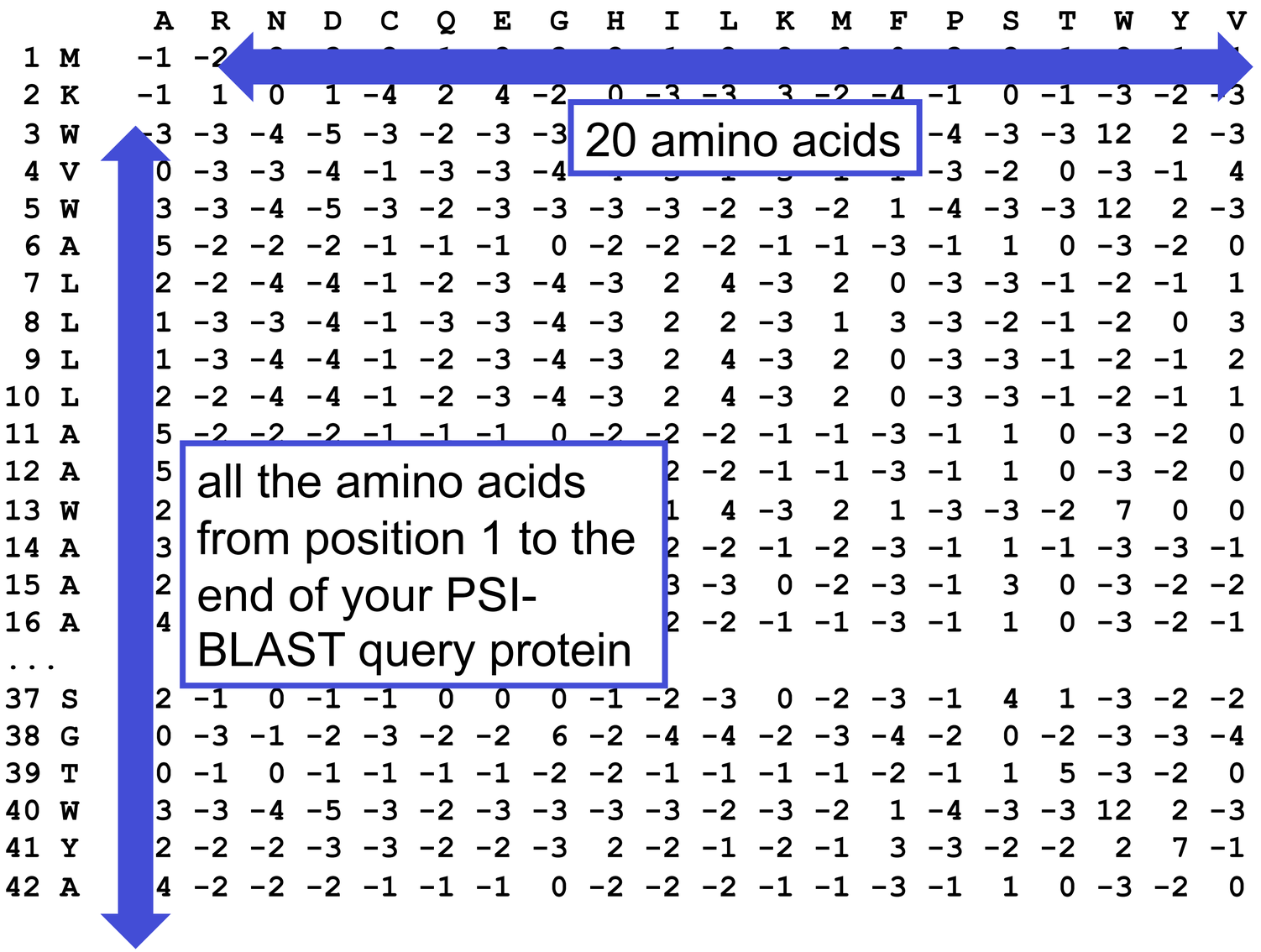
$q_{ij} = P_i P_j e^{\lambda_u s_{ij}},$

$$Q_i = \frac{\alpha f_i + \beta g_i}{\alpha + \beta}$$

$\alpha = N_C - 1$ *Nc* = number of distinct residues in column *C*

β= pseudocount parameter = 10

Example of a PSSM



		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
1	M	-1	-2																		
2	K	-1	1	0	1	-4	2	4	-2	0	-3	-3	3	-2	-4	-1	0	-1	-3	-2	-3
3	W	-3	-3	-4	-5	-3	-2	-3	-3							-4	-3	-3	12	2	-3
4	V	0	-3	-3	-4	-1	-3	-3	-4							-3	-2	0	-3	-1	4
5	W	-3	-3	-4	-5	-3	-2	-3	-3	-3	-3	-2	-3	-2	1	-4	-3	-3	12	2	-3
6	A	5	-2	-2	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	0
7	L	2	-2	-4	-4	-1	-2	-3	-4	-3	2	4	-3	2	0	-3	-3	-1	-2	-1	1
8	L	1	-3	-3	-4	-1	-3	-3	-4	-3	2	2	-3	1	3	-3	-2	-1	-2	0	3
9	L	1	-3	-4	-4	-1	-2	-3	-4	-3	2	4	-3	2	0	-3	-3	-1	-2	-1	2
10	L	2	-2	-4	-4	-1	-2	-3	-4	-3	2	4	-3	2	0	-3	-3	-1	-2	-1	1
11	A	5	-2	-2	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	0
12	A	5									2	-2	-1	-1	-3	-1	1	0	-3	-2	0
13	W	2									1	4	-3	2	1	-3	-3	-2	7	0	0
14	A	3									2	-2	-1	-2	-3	-1	1	-1	-3	-3	-1
15	A	2									3	-3	0	-2	-3	-1	3	0	-3	-2	-2
16	A	4									2	-2	-1	-1	-3	-1	1	0	-3	-2	-1
...																					
37	S	2	-1	0	-1	-1	0	0	0	-1	-2	-3	0	-2	-3	-1	4	1	-3	-2	-2
38	G	0	-3	-1	-2	-3	-2	-2	6	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4
39	T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-3	-2	0
40	W	3	-3	-4	-5	-3	-2	-3	-3	-3	-3	-2	-3	-2	1	-4	-3	-3	12	2	-3
41	Y	2	-2	-2	-3	-3	-2	-2	-3	2	-2	-1	-2	-1	3	-3	-2	-2	2	7	-1
42	A	4	-2	-2	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	0

Example of a PSSM

		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
1	M	-1	-2	-2	-3	-2	-1	-2	-3	-2	1	2	-2	6	0	-3	-2	-1	-2	-1	1
2	K	-1	1	0	1	-4	2	4	-2	0	-3	-3	3	-2	-4	-1	0	-1	-3	-2	-3
3	W	-3	-3	-4	-5	-3	-2	-3	-3	-3	-3	-2	-3	-2	1	-4	-3	-3	12	2	-3
4	V	0	-3	-3	-4	-1	-3	-3	-4	-4	3	1	-3	1	-1	-3	-2	0	-3	-1	4
5	W	-3	-3	-4	-5	-3	-2	-3	-3	-3	-3	-2	-3	-2	1	-4	-3	-3	12	2	-3
6	A	5	-2	-2	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	0
7	L	-2	-2	-4	-4	-1	-2	-3	-4	-3	2	4	-3	2	0	-3	-3	-1	-2	-1	1
8	L	-1	-3	-3	-4	-1	-3	-3	-4	-3	2	2	-3	1	3	-3	-2	-1	-2	0	3
9	L	-1	-3	-4	-4	-1	-2	-3	-4	-3	2	4	-3	2	0	-3	-3	-1	-2	-1	2
10	L	-2	-2	-4	-4	-1	-2	-3	-4	-3	2	4	-3	2	0	-3	-3	-1	-2	-1	1
11	A	5	-2	-2	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	0
12	A	5	-2	-2	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	0
13	W	-2	-3	-4	-4	-1	-3	-3	-4	-3	2	2	-3	1	1	-3	-3	-2	7	0	0
14	A	3	-2	-1	0	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	-1	-3	-3	-1
15	A	2	-1	0	-1	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	3	0	-3	-2	-2
16	A	4	-2	-1	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	-1
...																					
37	S	2	-1	0	-1	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	4	1	-3	-2	-2
38	G	0	-3	-1	-2	-1	-2	-2	6	-3	-3	-3	-2	-3	-4	-2	0	-2	-3	-3	-4
39	T	0	-1	0	-1	-1	-1	-1	0	-2	-2	-2	-1	-1	-2	-1	1	5	-3	-2	0
40	W	-3	-3	-4	-5	-3	-2	-3	-3	-3	-3	-2	-3	-2	1	-4	-3	-3	12	2	-3
41	Y	-2	-2	-2	-3	-3	-2	-2	-3	2	-2	-1	-2	-1	3	-3	-2	-2	2	7	-1
42	A	4	-2	-2	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	0

note that a given amino acid (such as alanine) in your query protein can receive different scores for matching alanine—depending on the position in the protein

Example of a PSSM

		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
1	M	-1	-2	-2	-3	-2	-1	-2	-3	-2	1	2	-2	6	0	-3	-2	-1	-2	-1	1
2	K	-1	1	0	1	-4	2	4	-2	0	-3	-3	3	-2	-4	-1	0	-1	-3	-2	-3
3	W	-3	-3	-4	-5	-3	-2	-3	-3	-3	-3	-2	-3	-2	1	-4	-3	-3	12	2	-3
4	V	0	-3	-3	-4	-1	-3	-3	-4	-4	3	1	-3	1	-1	-3	-2	0	-3	-1	4
5	W	-3	-3	-4	-5	-3	-2	-3	-3	-3	-3	-2	-3	-2	1	-4	-3	-3	12	2	-3
6	A	5	-2	-2	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	0
7	L	-2	-2	-4	-4	-1	-2	-3	-4	-3	2	4	-3	2	0	-3	-3	-1	-2	-1	1
8	L	-1	-3	-3	-4	-1	-3	-3	-4	-3	2	2	-3	1	3	-3	-2	-1	-2	0	3
9	L	-1	-3	-4	-4	-1									0	-3	-3	-1	-2	-1	2
10	L	-2	-2	-4	-4	-1									0	-3	-3	-1	-2	-1	1
11	A	5	-2	-2	-2	-1									-3	-1	1	0	-3	-2	0
12	A	5	-2	-2	-2	-1									-3	-1	1	0	-3	-2	0
13	W	-2	-3	-4	-4	-2											-3	-2	7	0	0
14	A	3	-2	-1	-2	-1									-3	1	1	-1	-3	-3	-1
15	A	2	-1	0	-1	-2									-3	-1	3	0	-3	-2	-2
16	A	4	-2	-1	-2	-1									-3	-1	1	0	-3	-2	-1
...																					
37	S	2	-1	0	-1	-1									-3	-1	4	1	-3	-2	-2
38	G	0	-3	-1	-2	-3									-4	-2	0	-2	-3	-3	-4
39	T	0	-1	0	-1	-1									-2	-1	1	5	-3	-2	0
40	W	-3	-3	-4	-5	-3									1	-4	-3	-3	12	2	-3
41	Y	-2	-2	-2	-3	-3									3	-3	-2	-2	2	7	-1
42	A	4	-2	-2	-2	-1	-1	-1	0	-2	-2	-2	-1	-1	-3	-1	1	0	-3	-2	0

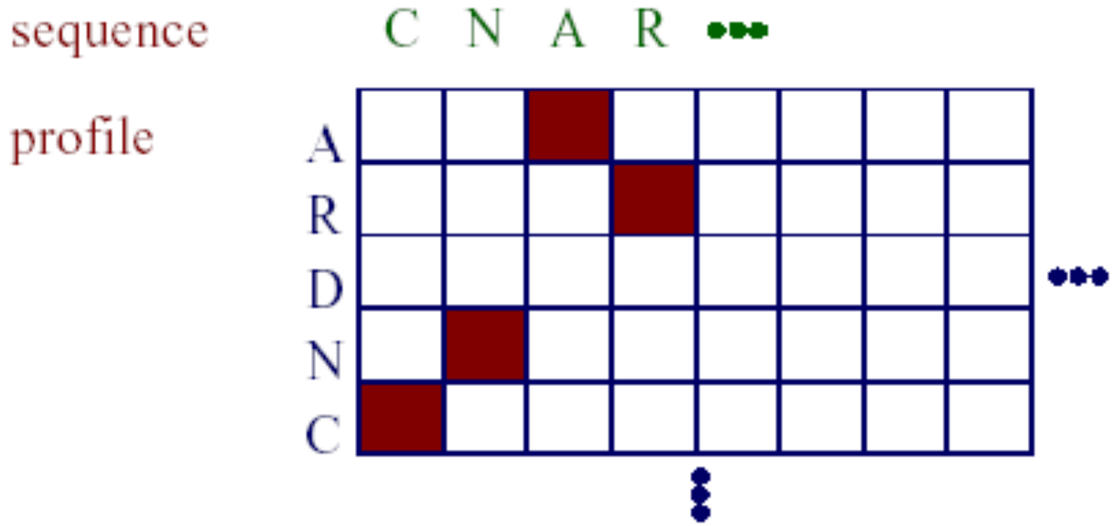
note that a given amino acid (such as tryptophan) in your query protein can receive different scores for matching tryptophan—depending on the position in the protein

Advantages of PSSM

- Position-specific: PSSMs account for specific residue variability and conservation at each position.
- Adaptive: PSSMs dynamically evolve based on the sequence set, making them more customized to the query.
- Remote homology detection: PSSMs are better at detecting distant homologs compared to fixed matrices.
- Context awareness: PSSMs capture important sequence-specific features such as conserved motifs.
- Iterative refinement: PSSMs improve through multiple iterations, becoming more accurate with each round.

The use of a PSSM to search database

- Searching with a Profile
- aligning profile matrix to a simple sequence
  - like aligning two sequences
  - except score for aligning a character with a matrix position is given by the matrix itself
  - not a substitution matrix



Position-Specific Iterated BLAST (PSI-BLAST)

(1) PSI-BLAST takes as an input a single protein sequence and compares it to a protein database, using the gapped BLAST program

(2) The program constructs a multiple alignment, and then a profile, from any significant local alignments found. The original query sequence serves as a template for the multiple alignment and profile, whose lengths are identical to that of the query. Different numbers of sequences can be aligned in different template positions.

(3) The profile is compared to the protein database, again seeking local alignments. After a few minor modifications, the BLAST algorithm can be used for this directly.

(4) PSI-BLAST estimates the statistical significance of the local alignments found. Because profile substitution scores are constructed to a fixed scale, and gap scores remain independent of position, the statistical theory and parameters for gapped BLAST alignments remain applicable to profile alignments

(5) Finally, PSI-BLAST iterates, by returning to step (2), an arbitrary number of times or until convergence.

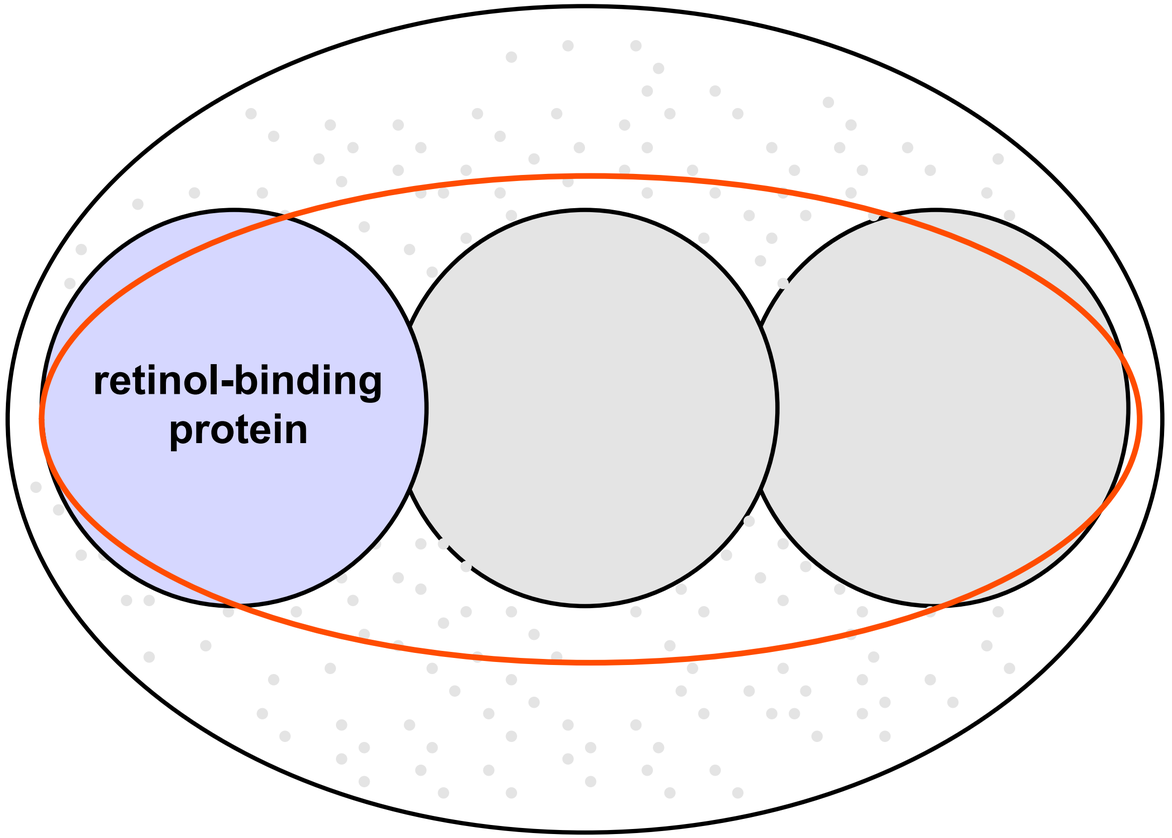
NCBI: http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-2.html

Results of a PSI-BLAST search

Iteration	# hits	# hits > threshold
1	104	49
2	173	96
3	236	178
4	301	240
5	344	283
6	342	298
7	378	310
8	382	320

Improve BLAST search to more distantly related homologous proteins using PSI-BLAST

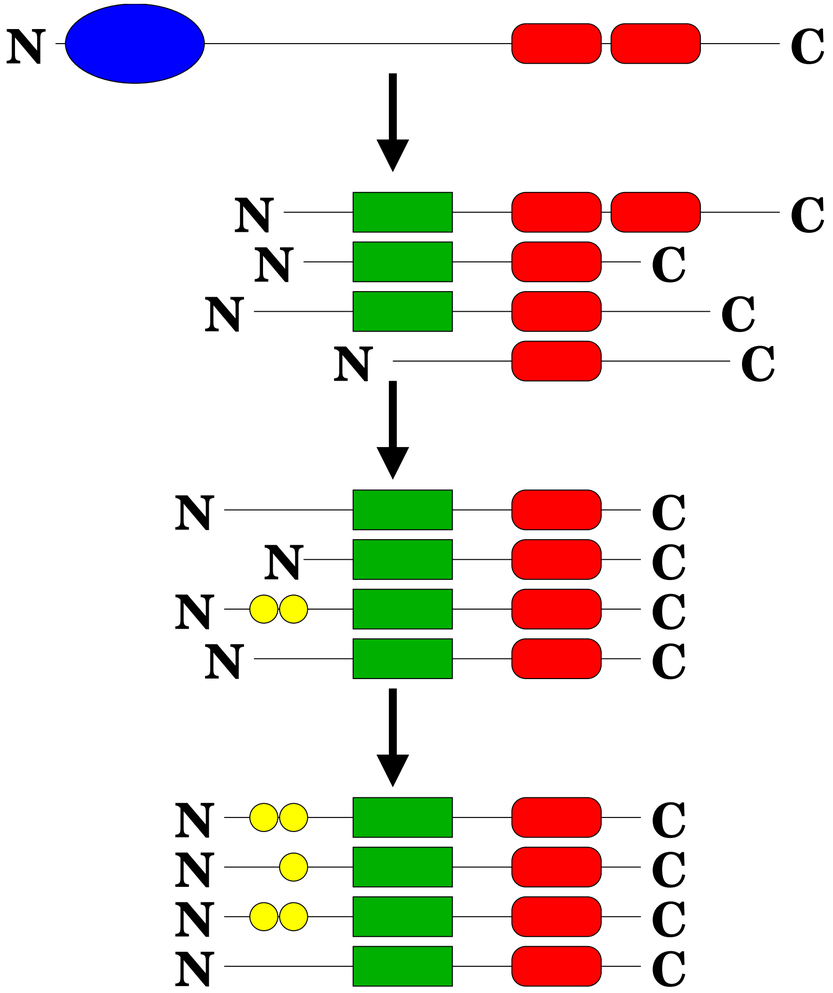
PSI-BLAST generates scoring matrices more powerful than PAM or BLOSUM



Advantages of PSI-BLAST

- A much sensitive scoring system. each position has its own pattern probabilities .
- Different weight to conserved positions.
- Important motifs are bounded
- Lowers the level of random noise.
- Finding distant relatives.
- Disadvantages: Errors are easily amplified by iterations.

PSI-BLAST may lead to wrong annotations



Query sequence

1st iteration

2nd iteration

3rd iteration

- A much sensitive scoring system. each position has its own pattern probabilities .
- Different weight to conserved positions.
- Important motifs are bounded
- Lowers the level of random noise.
- Finding distant relatives.
- Disadvantages: Errors are easily amplified by iterations.

https://blast.ncbi.nlm.nih.gov/doc/blast-help/downloadblastdata.html

BLAST+ executables

Do you have difficulties running high volume BLAST searches? Do you have proprietary sequence data to search and cannot use the NCBI BLAST web site? Do you have access to your own server? Do you have your own research pipeline? Have security or IP concerns about sending searches outside of your organization? If you answered yes to any of these questions, read on!

The NCBI provides a suite of command-line tools to run BLAST called BLAST+. This allows users to perform BLAST searches on their own server without size, volume and database restrictions. BLAST+ can be used with a command line so it can be integrated directly into your workflow.

What are the next steps?

Download and install BLAST+. Installers and source code are available from https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/. Download the databases you need, (see database section below), or create your own. Start searching.

For more details, please see the BLAST+ user manual, the BLAST Help manual, the BLAST releases notes, and the article in BMC Bioinformatics (PubMed link). See our versioning policy.

The BLAST+ suite is the currently supported package. The older C toolkit executables are no longer supported. See our versioning policy.

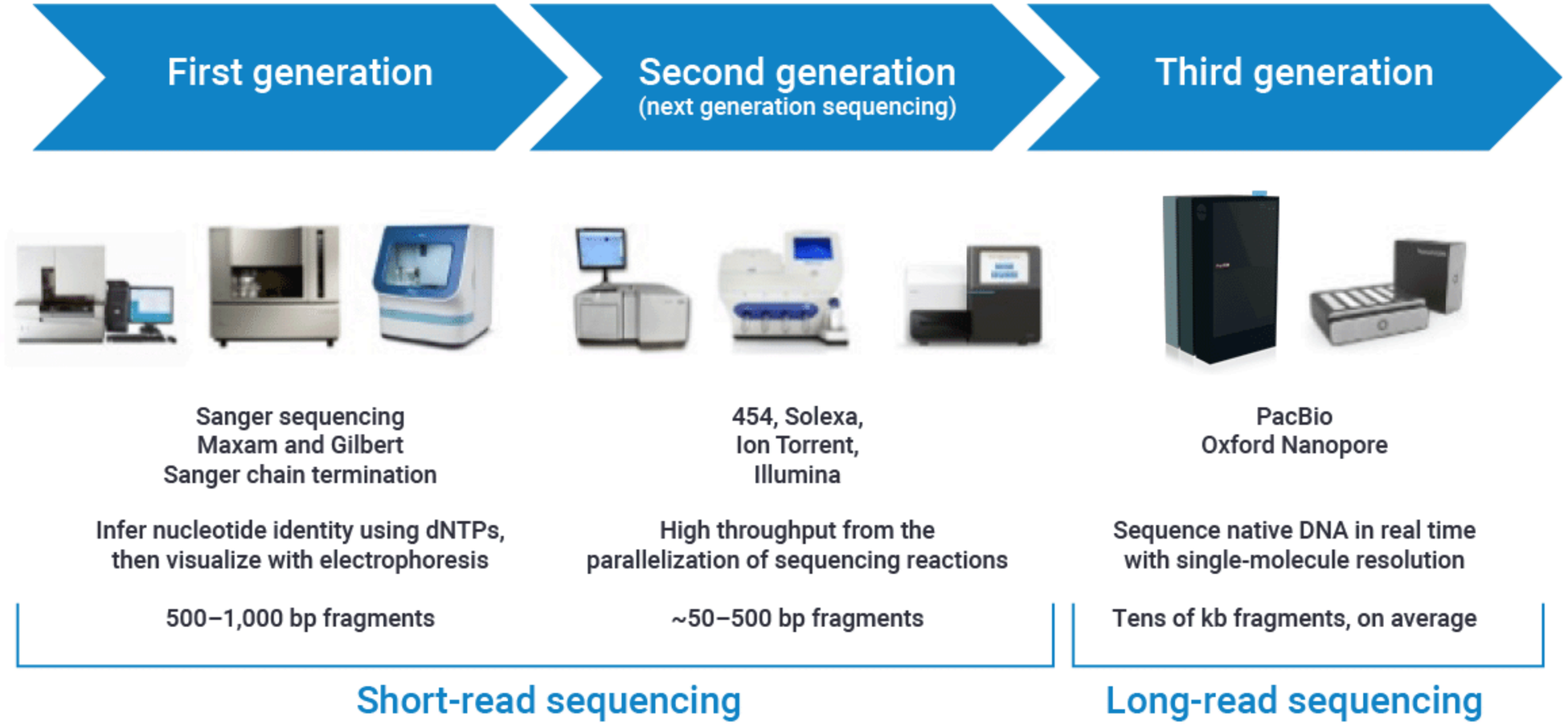
We are always listening and welcome your feedback at BLAST Support Center.

Summary for database search

- BLAST is a heuristic solution for sequence database searches.
- In gapped BLAST, dynamic programming will be initiated to produce gapped alignment when two hits are within a certain distance.
- BLAST alignment scores are evaluated for statistical significance based on the extreme value distribution.
- PSI-BLAST employs PSSM to search against the database.

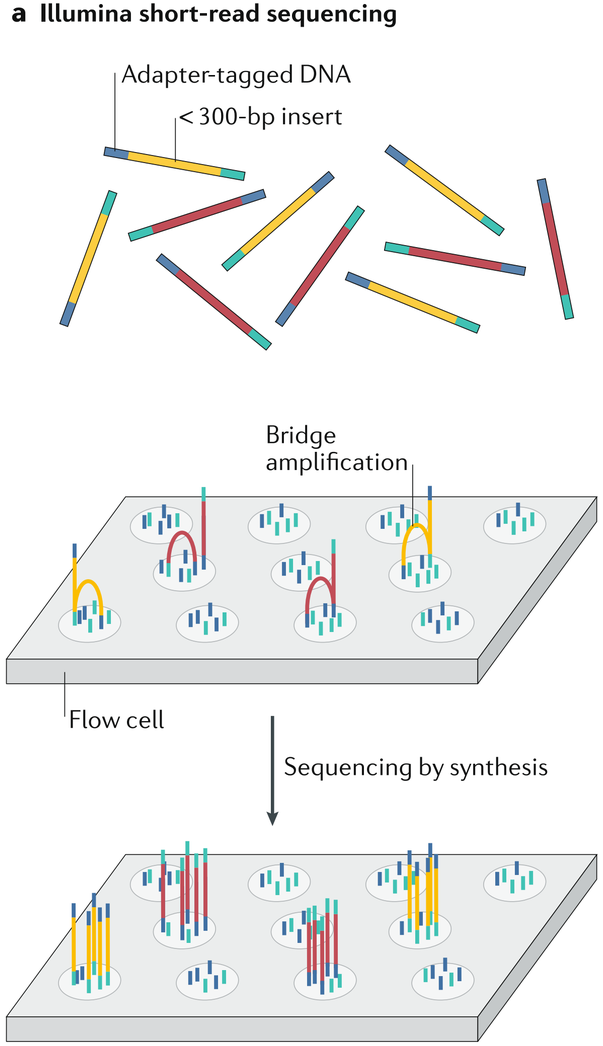
Reads mapping

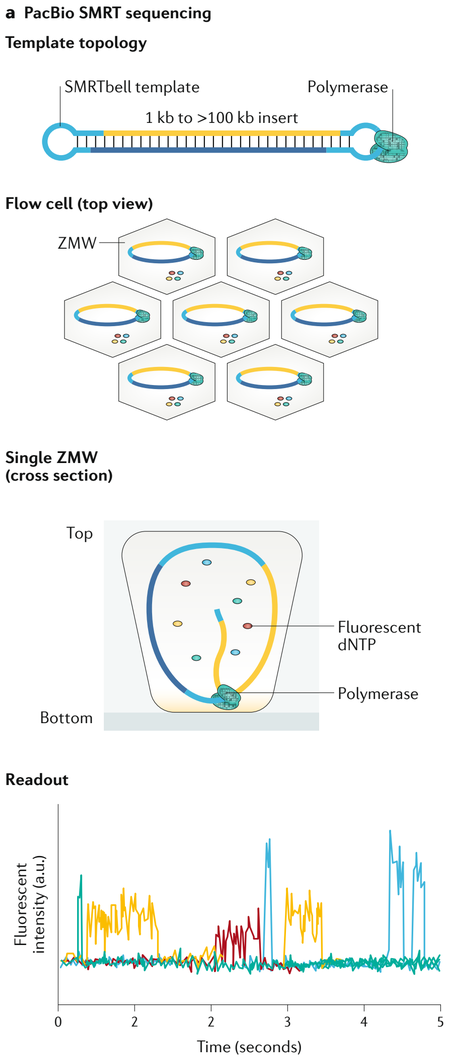
测序技术快速发展

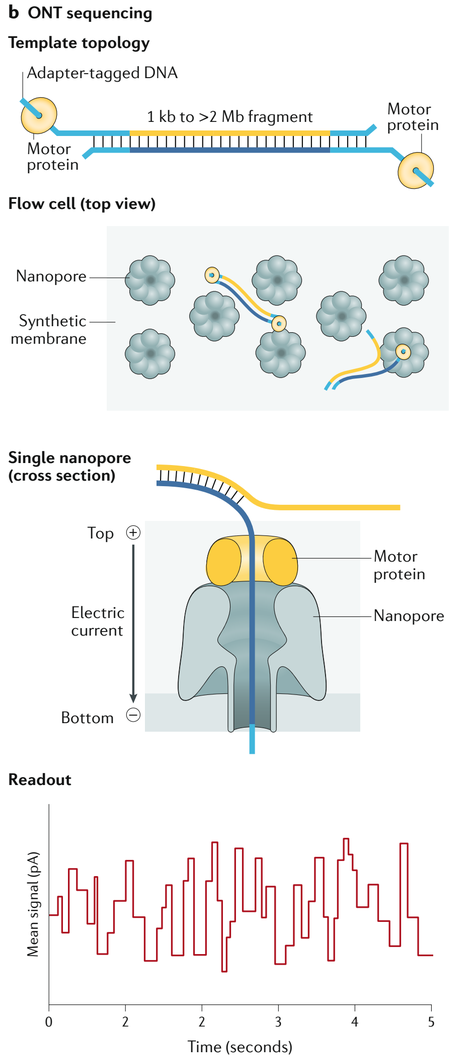


https://www.pacb.com/blog/the-evolution-of-dna-sequencing-tools/

二代和三代测序技术







- 二代测序采用合成测序（sequencing by synthesis, SBS）技术，测序时每次只能合成和读取一个碱基，通过荧光标记的核苷酸逐个加入并检测。每个循环都需要清洗、检测、合成下一个碱基，循环次数越多，错误累积的几率越大。因此，读长受限于反应周期和累积的化学反应效率，读段不能太长。

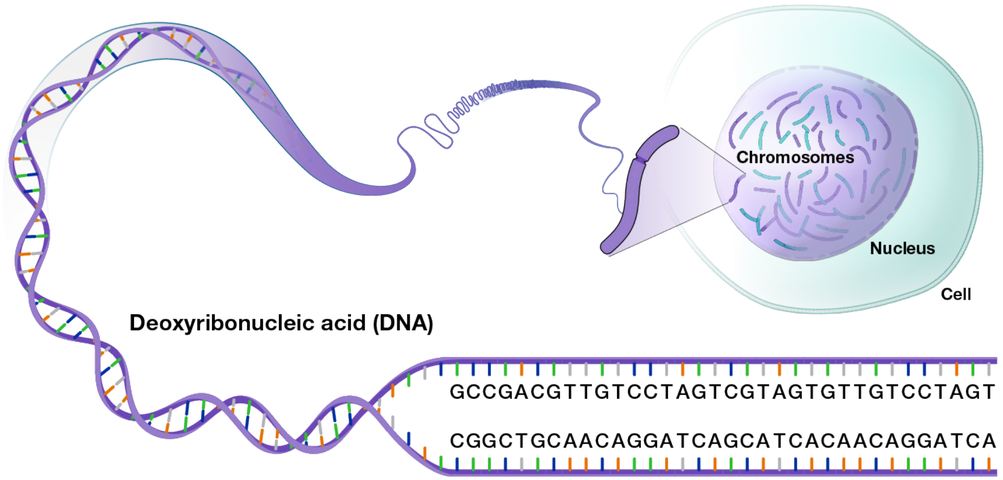
- 三代测序通过单分子实时测序（SMRT, PacBio）或纳米孔测序（Oxford Nanopore），可以直接读取长DNA分子，无需合成或循环检测。由于不需要多次反应，长读段序列的检测变得更加容易。

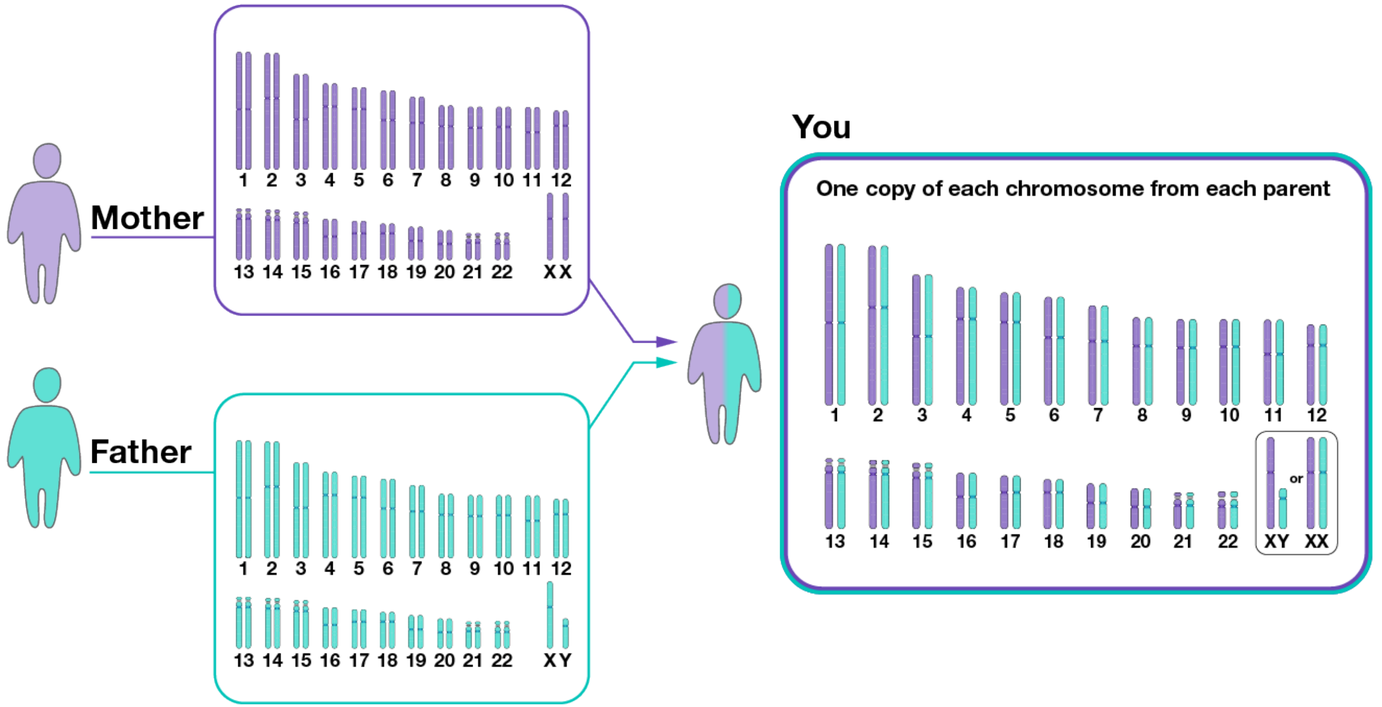
Feature	Second-Generation Sequencing (Illumina)	PacBio (SMRT Sequencing)	Nanopore Sequencing
Sequencing Principle	Sequencing by Synthesis (SBS), detecting each base via fluorescence	Single Molecule Real-Time (SMRT), detecting base incorporation via fluorescence	Nanopore sequencing, detecting electrical current changes as DNA passes through the nanopore
Read Length	Short reads, 100-300 bp	Long reads, 10-30 kb, HiFi reads 15 kb, up to 100 kb	Ultra-long reads, theoretically unlimited, often ¿10 kb, up to 2 Mb
Error Rate	Low (0.1%-1%)	Low in HiFi mode (0.1%-1%), higher error rate in standard mode (10%-15%)	Higher error rate (5%-20%), depending on platform and corrections
Throughput	High, generating billions of short reads per run	Lower throughput, especially in HiFi mode, slower sequencing speed	High, real-time data generation, more flexible throughput
Application Scenarios	Short-read sequencing, suitable for WGS, WES, RNA-seq	High-accuracy genome assembly, structural variant detection, complex repeat region analysis	Ultra-long read sequencing, real-time data generation, field diagnostics, large structural variation studies
Equipment Cost	High, suitable for large-scale genome sequencing projects	Expensive, suited for large labs with high-accuracy needs	Lower cost, portable devices (e.g., MinION) suitable for small labs or field use
Flexibility	Generally low, fixed sequencing workflow and read length	More flexible read length, but requires longer sequencing time	Highly flexible, real-time data, unlimited read length

Table 1: Comparison of Second-Generation (Illumina), PacBio (SMRT Sequencing), and Nanopore Sequencing Technologies

- 二代测序（Illumina）以高通量、低错误率著称，适合大规模短读段测序项目，如全基因组测序和转录组分析。其设备成本较高，但技术成熟，广泛应用于各类基因组研究。
- PacBio（SMRT测序）则通过较长的读段和高保真度（HiFi模式），在复杂的基因组结构变异和基因组组装中占据优势，但其通量较低，设备昂贵，适合高精度需求的项目。
- Nanopore（纳米孔测序）具有极高的灵活性，能够生成超长读段，适合现场检测和大规模结构变异分析。虽然错误率较高，但通过高覆盖度和后续校正，依然能够在特定应用中展现优势。

人类基因组





- 基因组是每个细胞中完整的DNA集合，由四种核苷酸（A、T、C、G）组成的序列提供细胞功能的指令。
- 在人类中，基因组包含大约30亿个核苷酸，分布在23条染色体上，大多数细胞有两套基因组副本（即二倍体），分别从父母继承而来。

https://www.genome.gov/about-genomics/educational-resources/fact-sheets/human-genomic-variation

基因组变异与表型

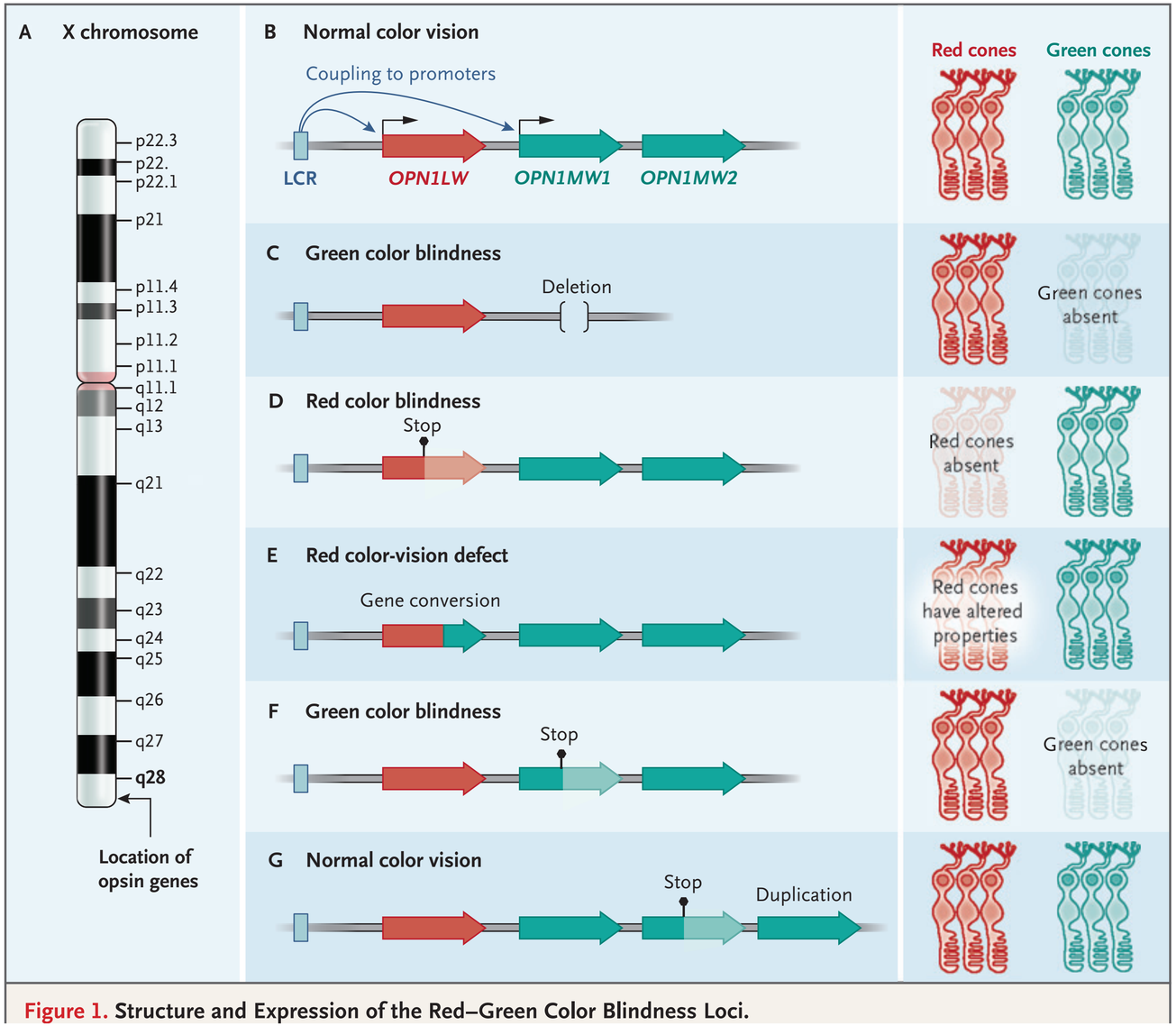


Figure 1. Structure and Expression of the Red–Green Color Blindness Loci.

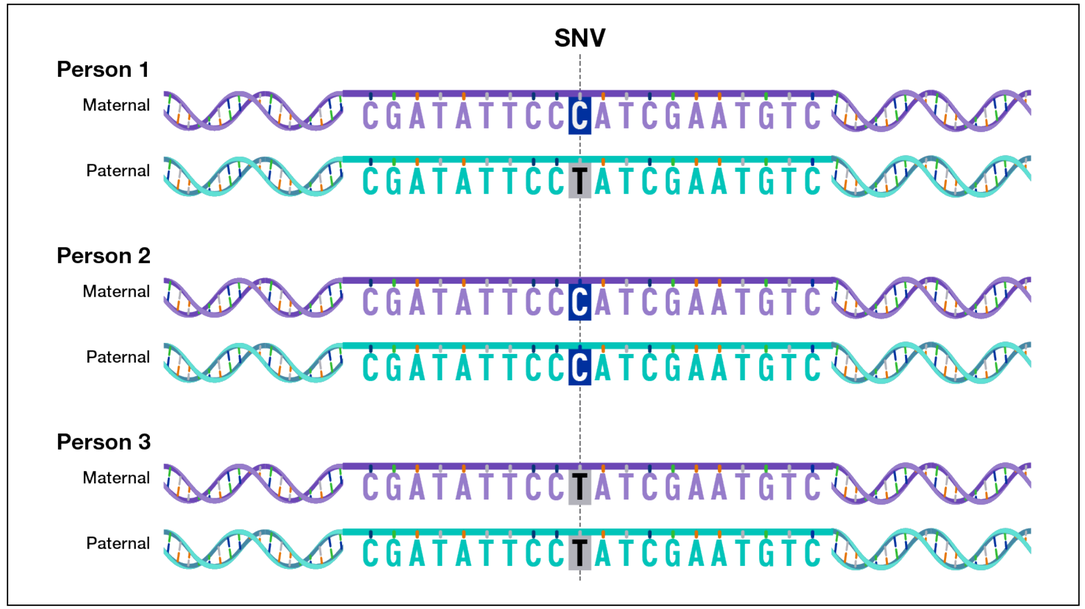
- 红绿色盲（红绿色色觉异常）是一种遗传性视力缺陷，主要由X染色体上的**视蛋白基因变异**引起。**OPN1LW基因**（位于X染色体上，编码红光敏感视蛋白）。**OPN1MW基因**（位于X染色体上，编码绿光敏感视蛋白）。因为这些基因位于X染色体上，红绿色盲是**X连锁隐性遗传**。

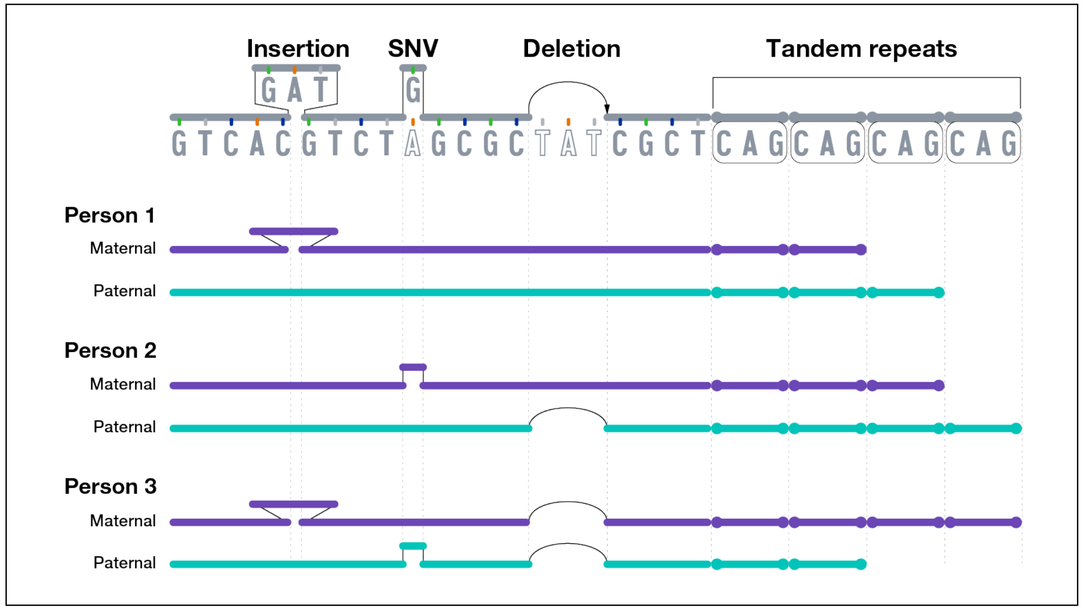
- 由于OPN1LW和OPN1MW基因的位置非常接近，且它们的序列高度相似，在染色体重组时容易发生交叉错误。这可能导致某些人会缺失OPN1LW或OPN1MW基因，导致对红色或绿色不敏感。有时，部分OPN1LW和OPN1MW基因会融合形成一个杂合基因，产生功能异常的视蛋白，从而引发颜色感知异常。

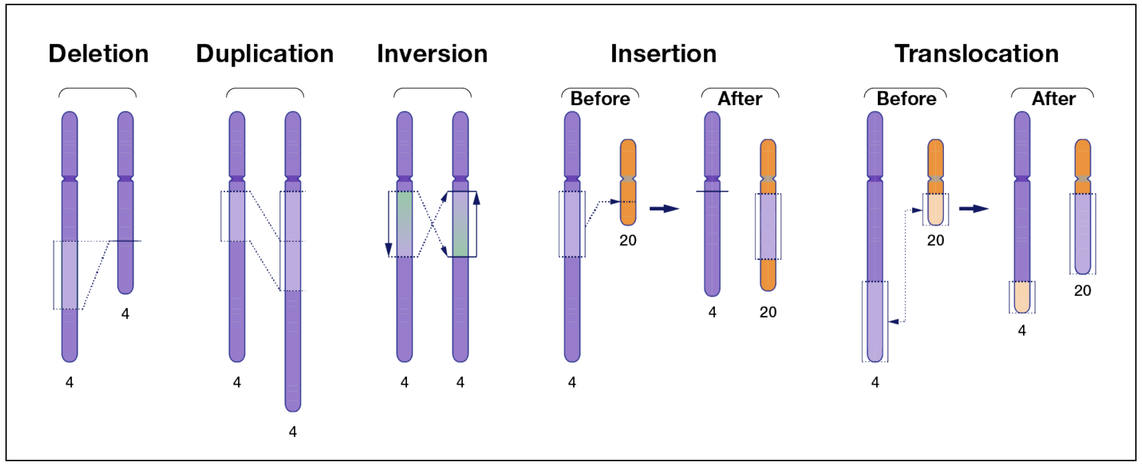
- OPN1LW或OPN1MW基因中的点突变也会导致蛋白质功能异常。这些突变可能改变视蛋白的结构，影响它们对特定波长光的敏感性，进而影响红色或绿色的感知。

N Engl J Med. 2019 Jul 4;381(1):64-74. doi: 10.1056/NEJMra1809315

基因组变异类型

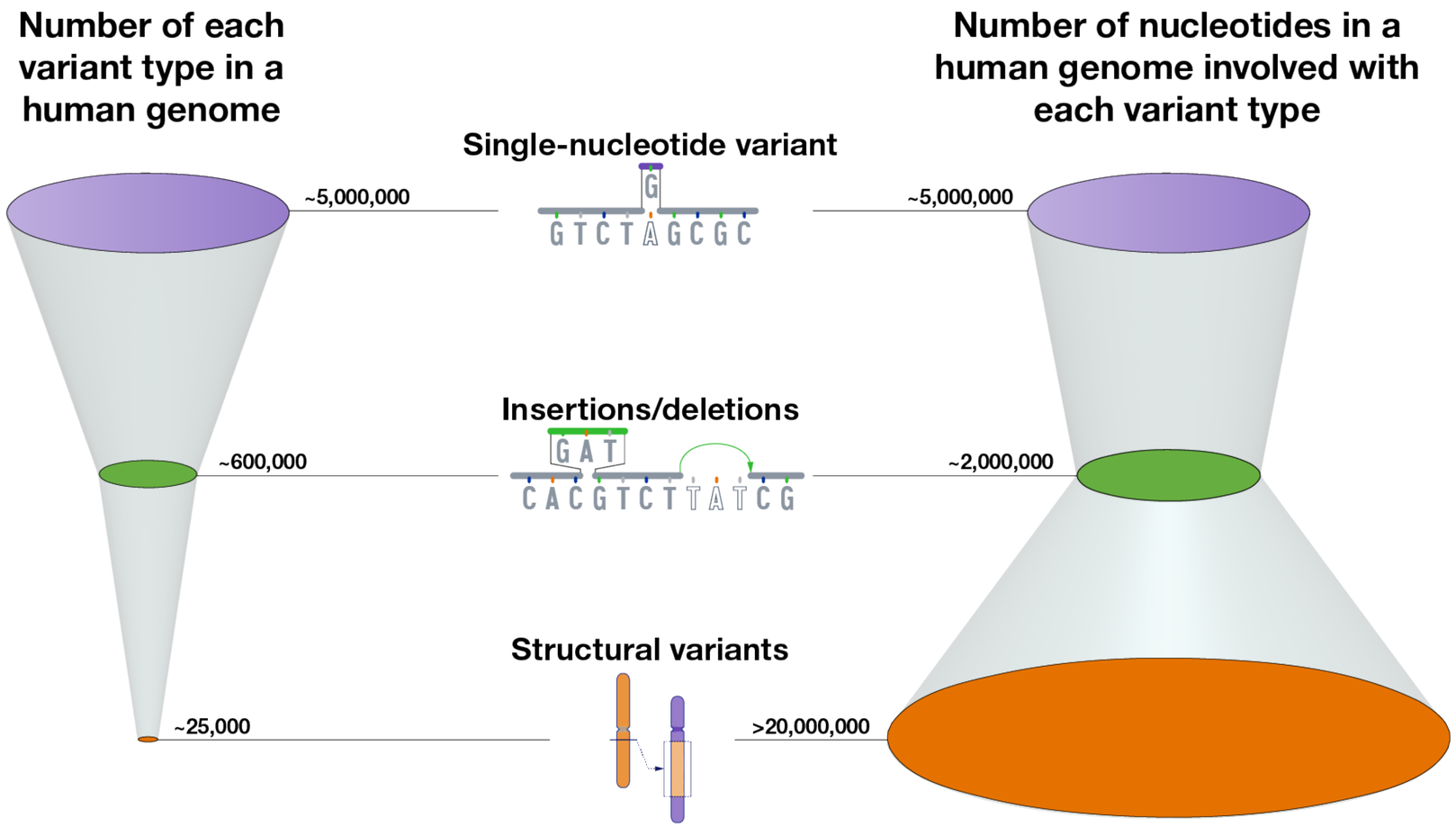






https://www.genome.gov/about-genomics/educational-resources/fact-sheets/human-genomic-variation

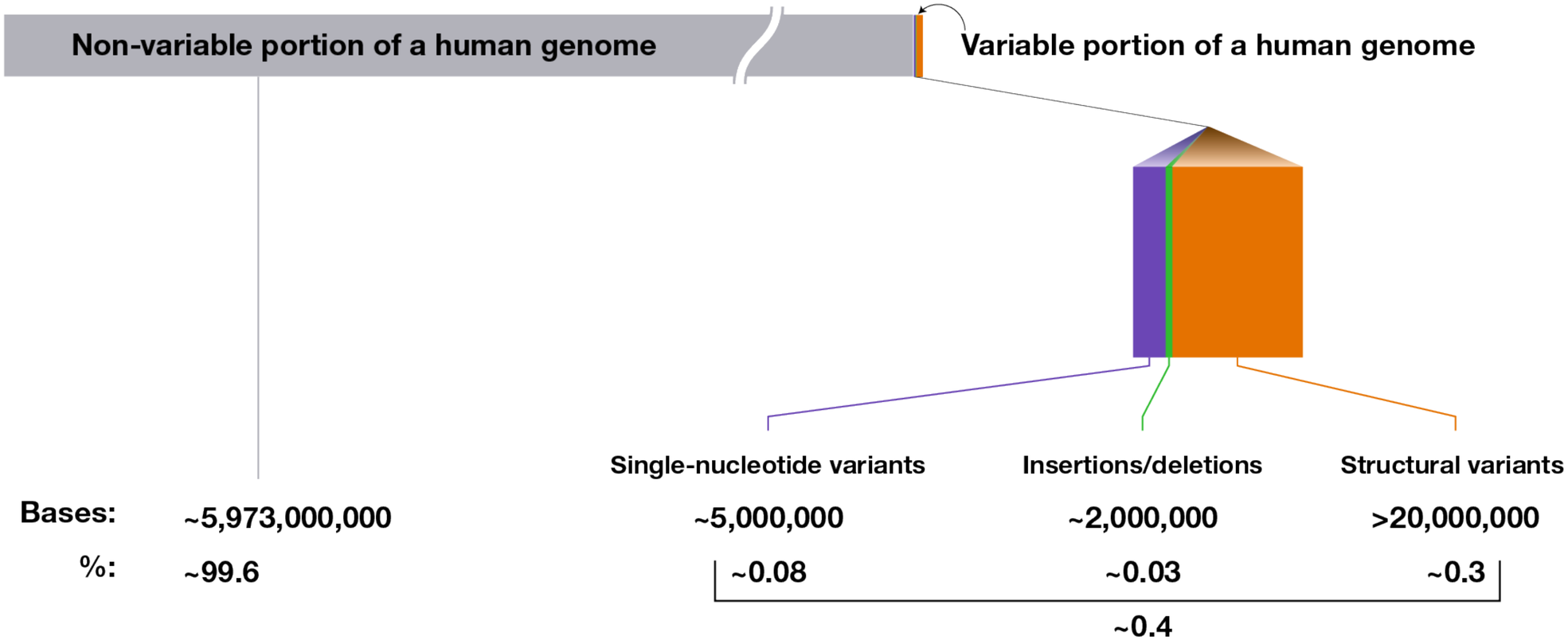
基因组变异数量



- 平均而言，一个人的基因组序列包含三种主要类型的基因组变异，其总核苷酸数量各不相同。单核苷酸变异（SNV）涉及单个核苷酸，而插入/缺失变异和结构变异涉及多个核苷酸，其中结构变异涉及的核苷酸数量最多，但发生频率最低。

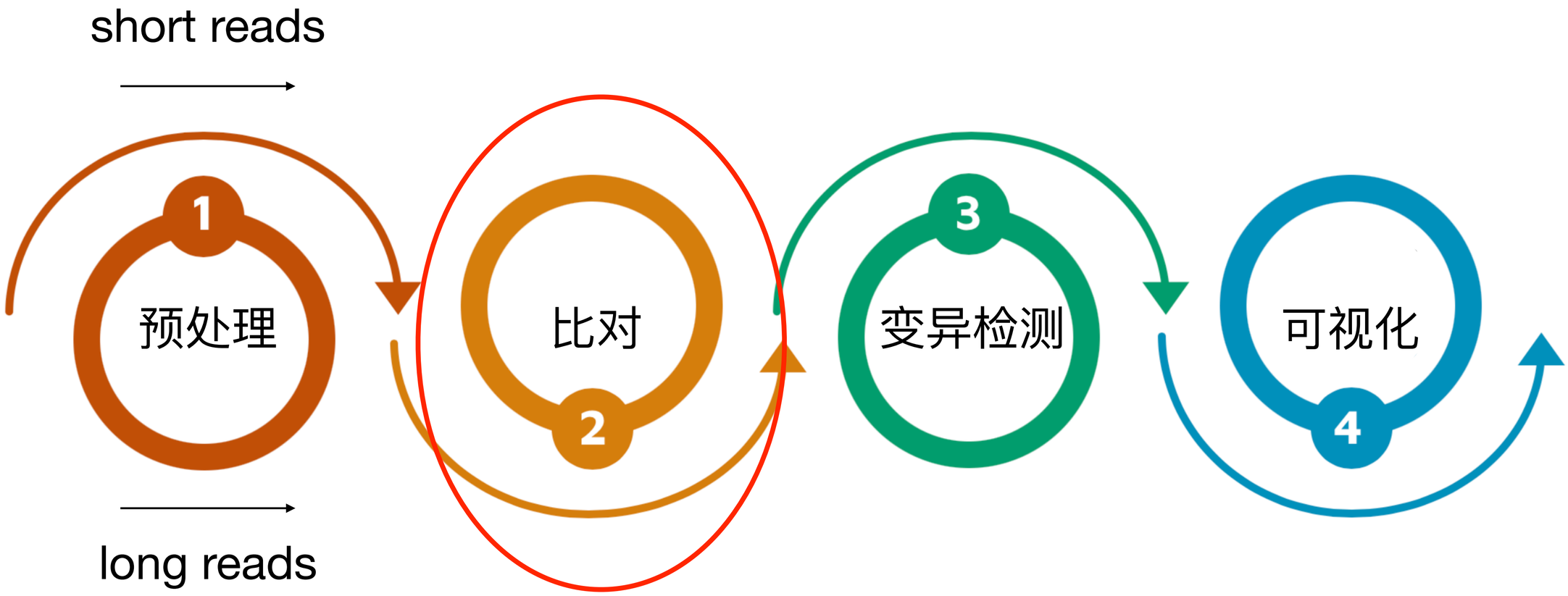
https://www.genome.gov/about-genomics/educational-resources/fact-sheets/human-genomic-variation

基因组变异数量

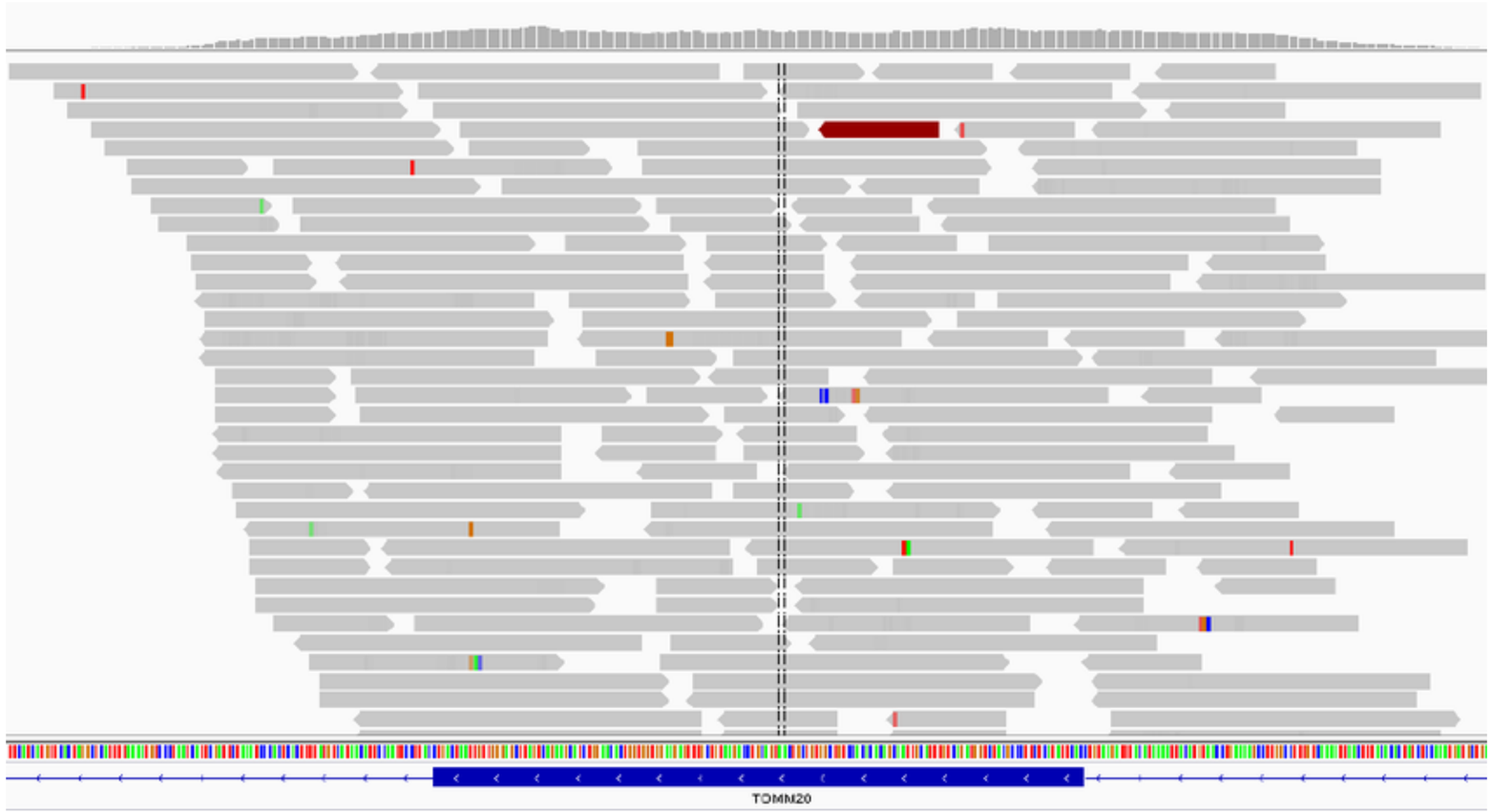


- 人类基因组序列与参考基因组的相似度约为99.6%，可变部分占0.4%，包括三种主要类型的基因组变异。直接比较两个人的基因组序列时，也会得到类似的结果。

基因组数据处理流程



Reads mapping/alignment



- Goal: accurately and rapidly mapping reads to reference genomes.

Why BLAST is not appropriate for reads mapping?

- **Inefficiency**: BLAST is designed for high-sensitivity local alignments of individual or few sequences, making it slower when processing millions or billions of short reads from next-generation sequencing (NGS).

- **Overly Complex Algorithm**: BLAST involves complex processes like seed extension and local alignment scoring (e.g., Smith-Waterman), which are unnecessary for short-read mapping.

- **Suboptimal for Short Reads**: BLAST is less suited for very short sequences (typically 100-150 bp), as its seed matching and extension strategies are better for longer sequences.

- **Higher Memory and Storage Requirements**: BLAST requires more memory and storage for its word-based indexing, making it less efficient for large-scale short-read data.

Reads mapping is essentially a string matching problem

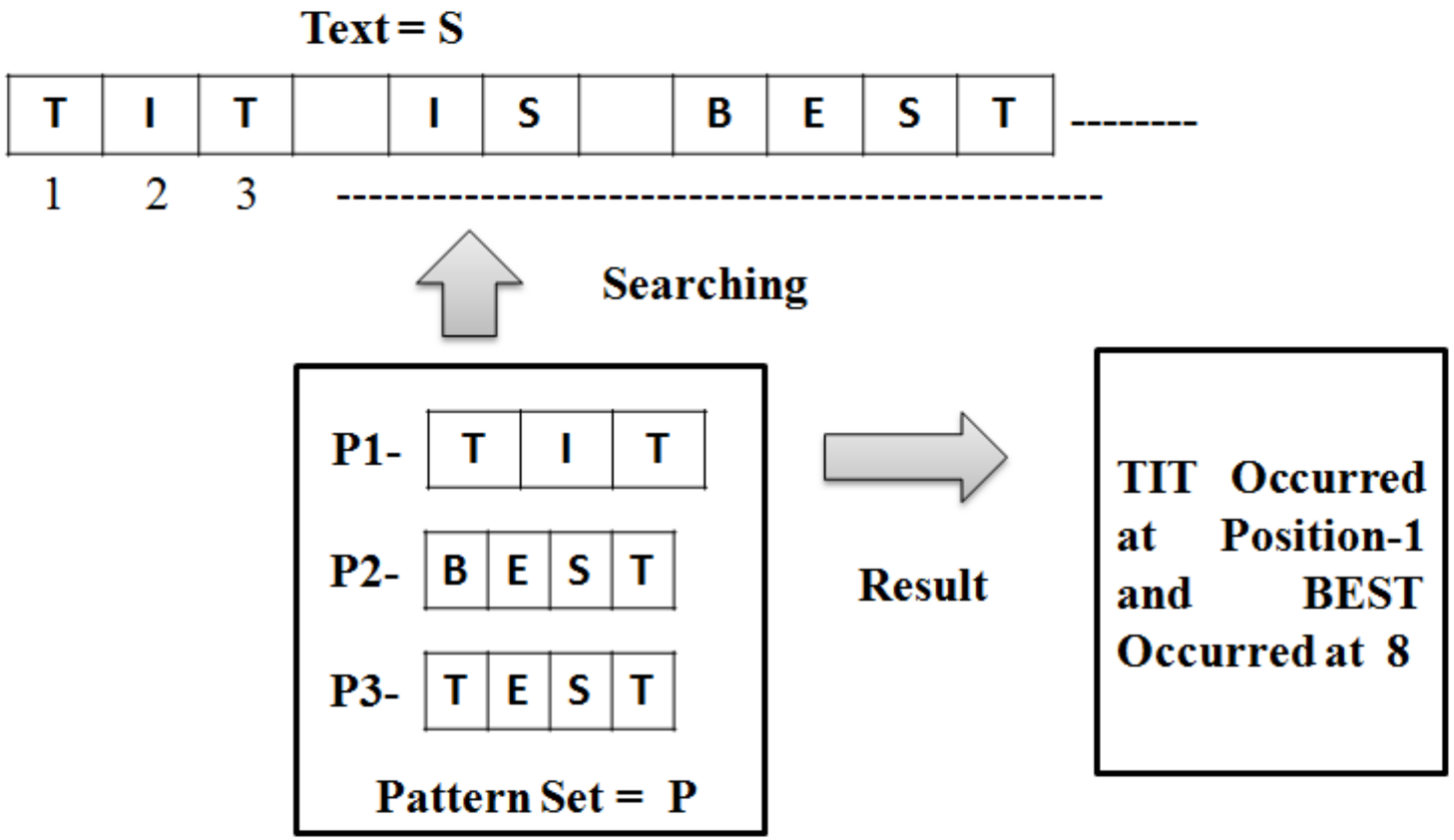
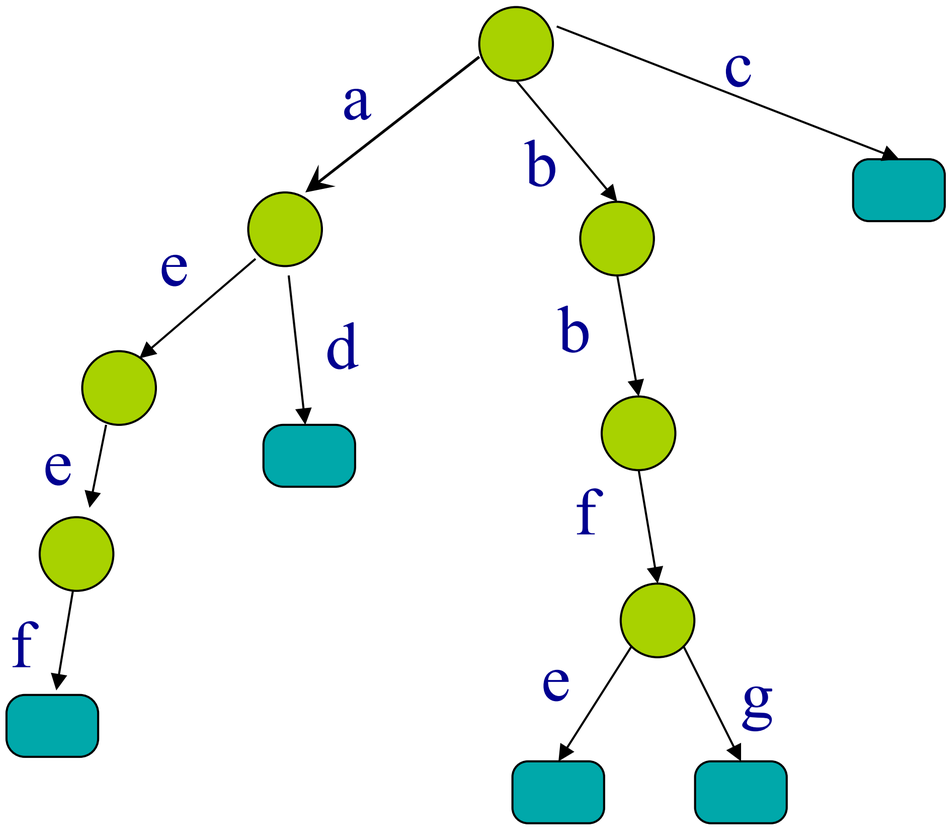


Figure 1 : String Matching Problem

A Trie（字典树）structure to store and search strings

{
aeef
ad
bbfe
bbfg
c
}



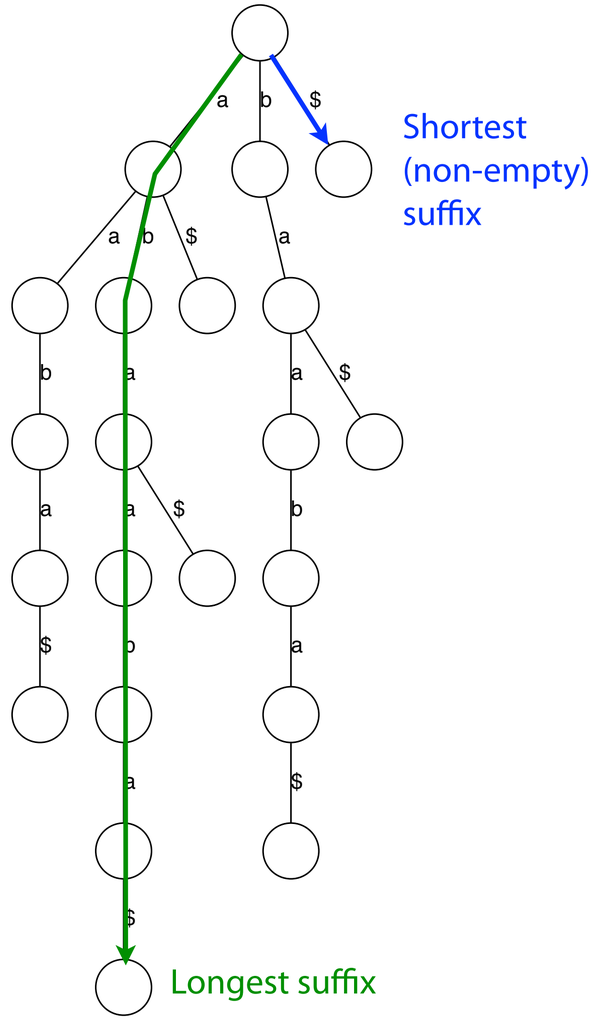
- A **Trie** is a **tree-based** data structure used to efficiently store and retrieve strings. Each path from the root of the Trie to a leaf node represents a complete string.
- It is also known as a **prefix tree** because the strings are stored in such a way that all common prefixes between strings are represented by a shared path.
- **Shared Prefixes**: If two strings share a common prefix, they share the same path from the root down until their characters diverge. This reduces the redundancy of storing common parts of different strings.
- **Fast Searching**: Searching for a string or a prefix is efficient because it follows the characters in the search string as a path down the tree. The time complexity for searching a string is proportional to the length of the string (O(m), where m is the length of the string).
- **Space Complexity**: Tries can require a lot of memory, especially when dealing with a large number of strings or when the alphabet size is large. This is because every character in the string requires a node, even for short words.

Suffix Trie for the string matching problem

T: abaaba *T$:* abaaba**$**

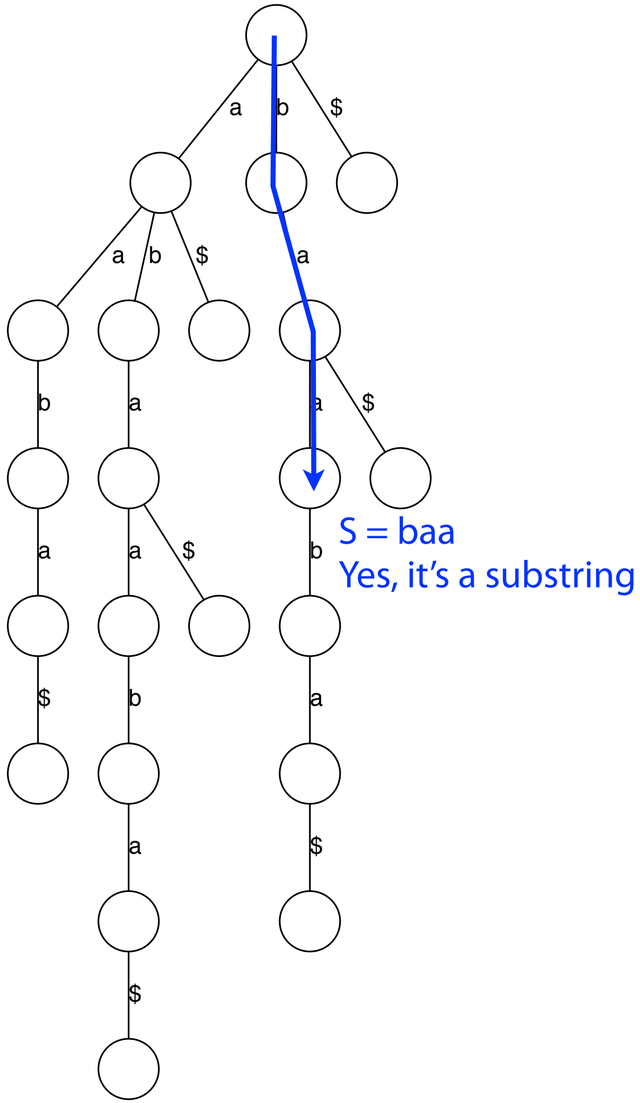
All suffixes to T

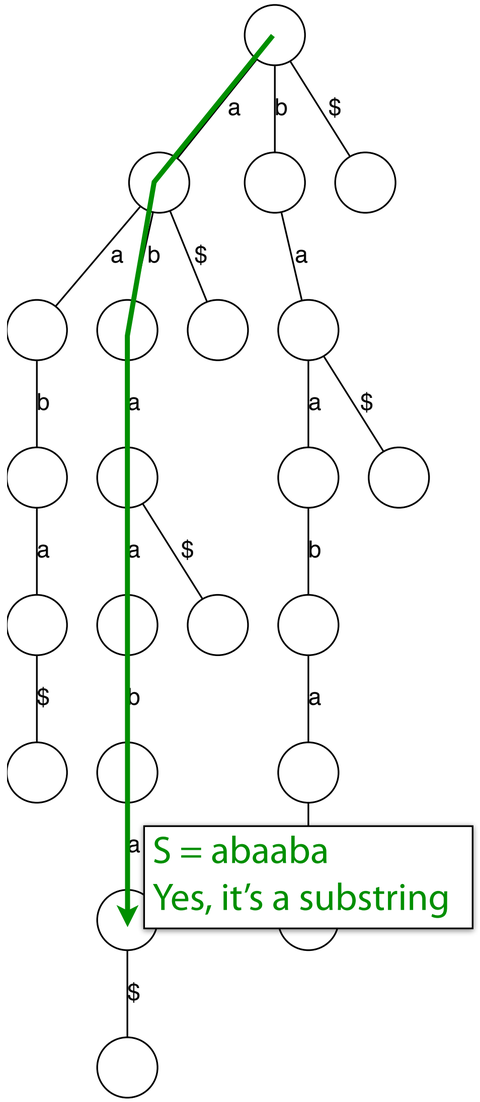
$	$
a$	a$
ba$	aaba$
aba$	aba$
aaba$	abaaba$
baaba$	ba$
abaaba$	baaba$

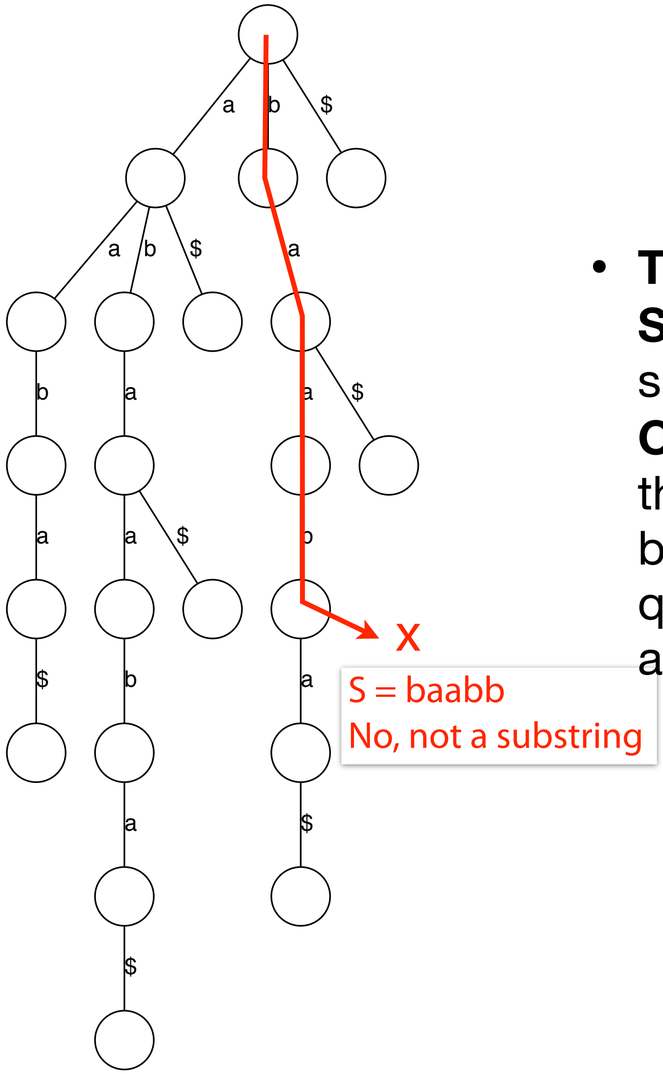


- A **Suffix Trie** is a Trie where each path from the root to a leaf node represents a suffix of the input string. Every suffix of the string is inserted into the Trie, with each character of the suffix stored as a node connected by edges. The Suffix Trie is a complete representation of all possible suffixes of the string, making it a powerful tool for searching for substrings.I
- **Time Complexity for Searching**: Searching for a substring in a Suffix Trie takes **O(m)**, where **m** is the length of the query string. This is because each character of the query is checked sequentially as we traverse the Trie.

Suffix Trie for the string matching problem





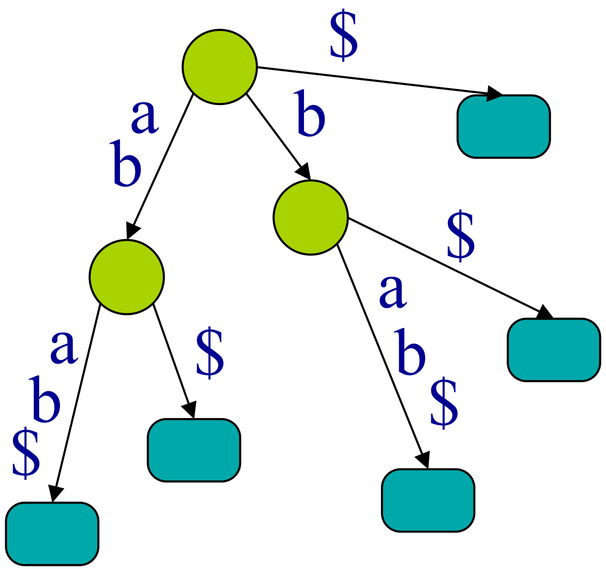


- **Time Complexity for Searching**: Searching for a substring in a Suffix Trie takes **O(m)**, where **m** is the length of the query string. This is because each character of the query is checked sequentially as we traverse the Trie.

Suffix Tree

Let s=abab. A suffix tree of s is a compressed trie of all suffixes of s=abab$

{
$
b$
ab$
bab$
abab$
}



Limitations of suffix tree for reads mapping

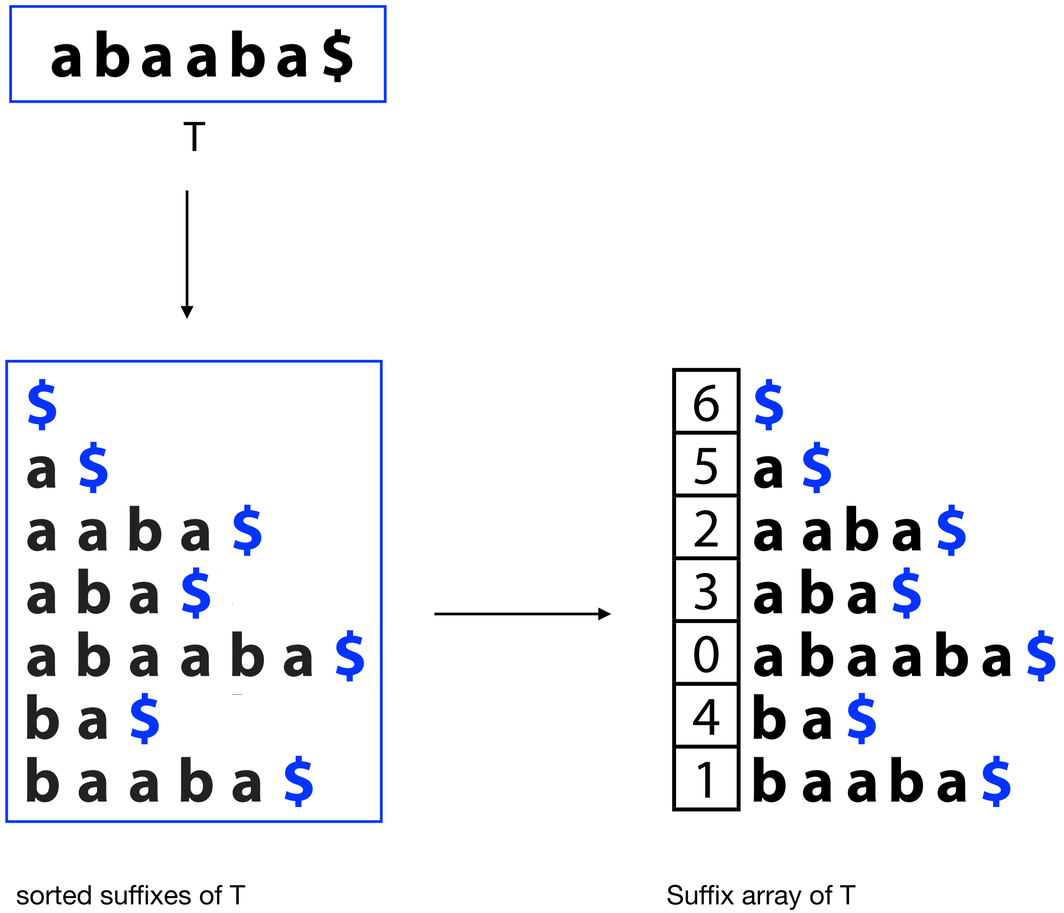
- **High Memory Usage**: The biggest drawback of a Suffix Trie is its space complexity. For long strings, like entire books or genomes, a Suffix Trie can use an enormous amount of memory, making it impractical for large-scale applications.

    - Using a suffix trie to represent the human genome (around 3 billion base pairs) could require several terabytes of memory due to the large number of nodes and edges needed to store all suffixes explicitly.

    - In comparison, a suffix tree, while more space-efficient, can still consume 10 to 30 times the size of the original string, potentially needing 30–90 GB of RAM, making it impractical for large-scale applications.

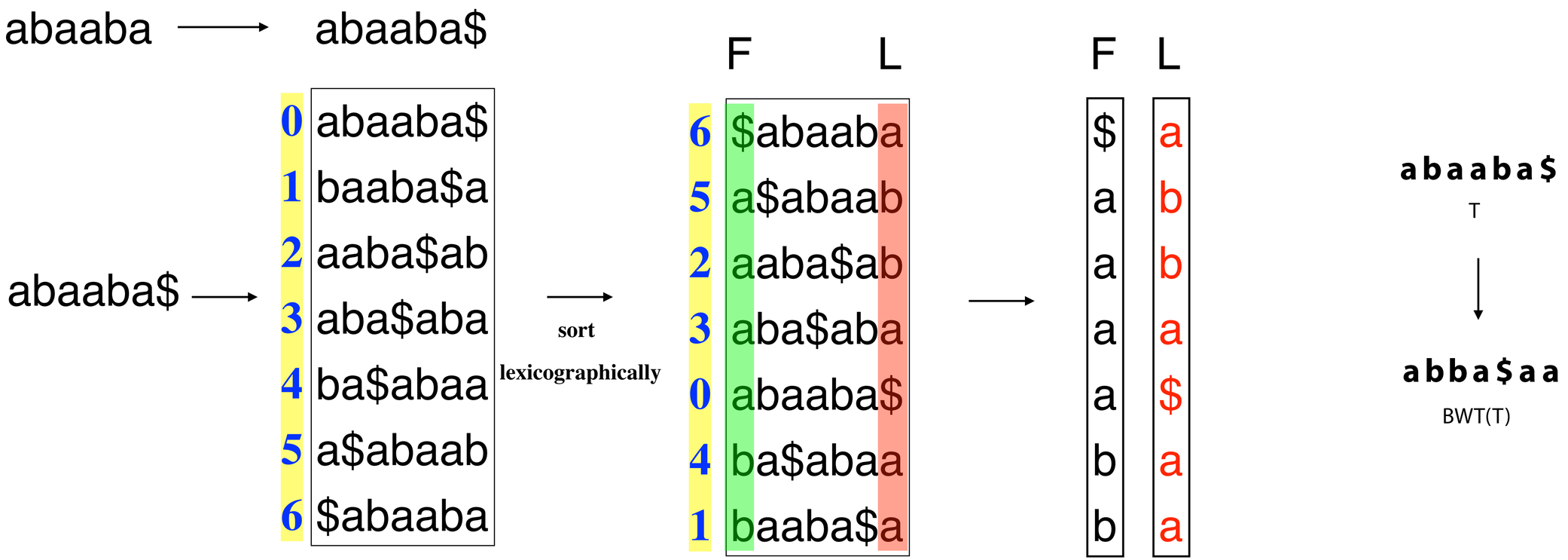
- **Construction Time**: Building a Suffix Trie takes **$O(n^2)$** time in the worst case, which is slow for long strings. The construction time of a suffix tree is **$O(n)$**, which is faster than a suffix trie. However, it still requires complex algorithms, such as Ukkonen's algorithm, to achieve this linear time. The construction of a suffix tree for a large dataset, like a genome, can still be slow in practice due to high constant factors and the overhead of managing memory for large data structures.

Suffix Array



- A **Suffix Array** is a space-efficient array that stores the starting positions of all suffixes of a string in lexicographical order.
- **Memory Efficiency**: Suffix Arrays are much more **space-efficient** than Suffix Tries. The size of a Suffix Array is **O(n)**, where **n** is the length of the string, making it practical for large datasets, such as entire genomes.
- Modern algorithms can build a Suffix Array in **O(n log n)** or even **O(n)** time, making it faster to construct than a Suffix Trie, especially for long strings.
- Searching for a substring in a Suffix Array can be done in **O(m log n)** time using binary search, where **m** is the length of the query and **n** is the length of the text. This is generally slower than searching in a Suffix Trie, but the reduced memory footprint often makes this tradeoff acceptable.

From Suffix Array to Burrows-Wheeler Transform (BWT)



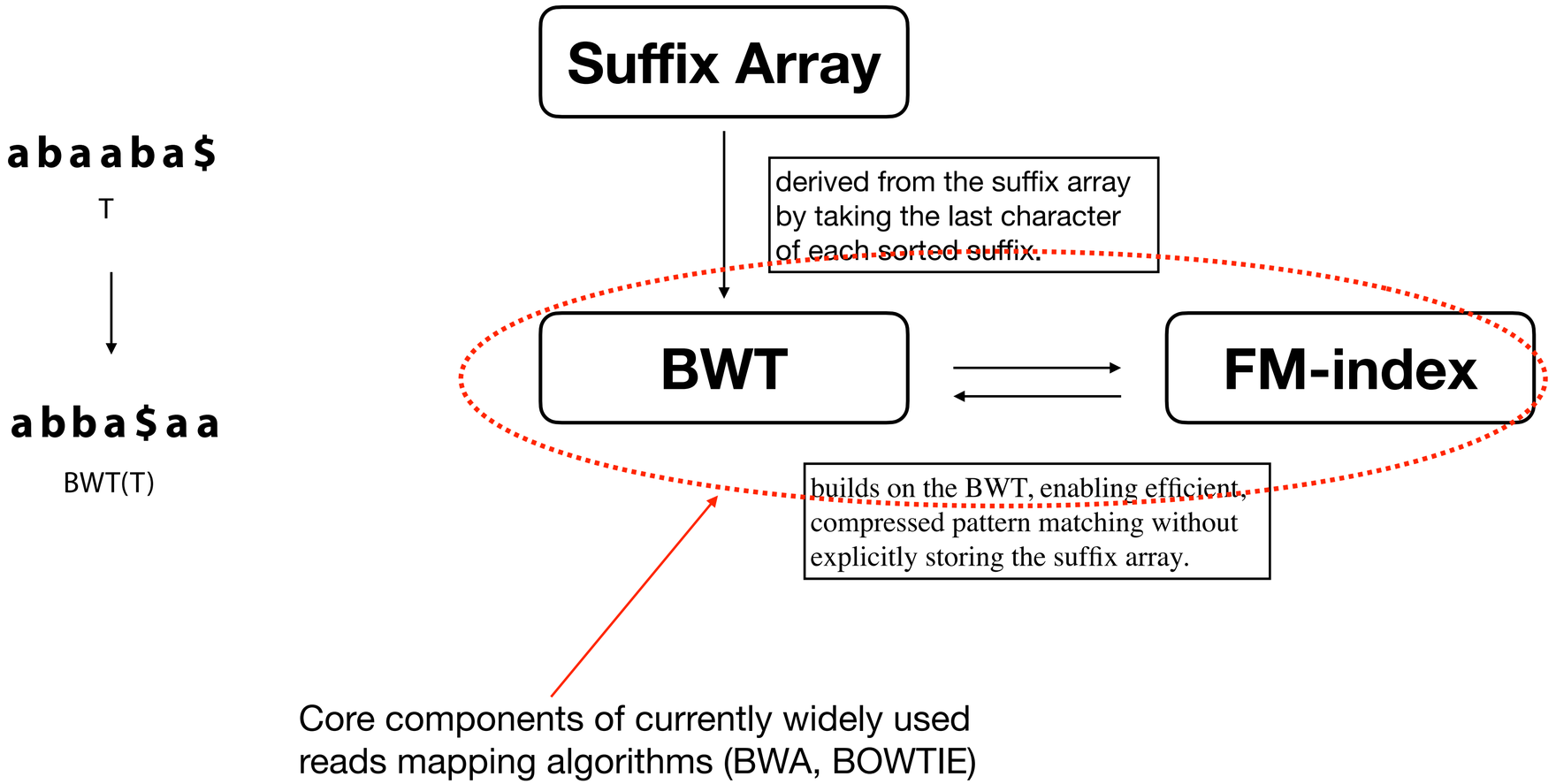
abaaba$

T

abba$aa

BWT(T)

From Suffix Array to Burrows-Wheeler Transform (BWT) and FM-index



From Suffix Array to Burrows-Wheeler Transform (BWT) and FM-index

- The **Burrows-Wheeler Transform** (BWT) is a reversible transformation of a string, generated from its suffix array. The BWT is created by taking the last character of each suffix from the sorted suffix array, forming a transformed string. The BWT facilitates efficient compression and string matching due to its tendency to cluster similar characters together.

- **BWT** is derived from the suffix array by taking the last character of each sorted suffix.

- The **FM-index** is built directly from the BWT. It uses backward searching, a technique that leverages the repetitive patterns in the BWT to reconstruct matching substrings efficiently.

- BWT and FM-index are the core components of BWA and BOWTIE, the currently widely used reads mapping algorithms.

The Last-to-First (LF) mapping property in BWT

Give each character in *T* a rank, equal to # times the character occurred previously in *T*. Call this the *T-ranking*.

$\mathbf{a_0\ b_0\ a_1\ a_2\ b_1\ a_3\ \$}$

BWM with T-ranking:

F	*L*
$ a_0 b_0 a_1 a_2 b_1	$\mathbf{a_3}$
$\mathbf{a_3}$ $ a_0 b_0 a_1 a_2	b_1
$\mathbf{a_1}$ a_2 b_1 a_3 $ a_0	b_0
$\mathbf{a_2}$ b_1 a_3 $ a_0 b_0	$\mathbf{a_1}$
$\mathbf{a_0}$ b_0 a_1 a_2 b_1 a_3	$
b_1 a_3 $ a_0 b_0 a_1	$\mathbf{a_2}$
b_0 a_1 a_2 b_1 a_3 $	$\mathbf{a_0}$

as occur in the same order in *F* and *L*: $\mathbf{a_3}$, $\mathbf{a_1}$, $\mathbf{a_2}$, $\mathbf{a_0}$

BWM with T-ranking:

F	*L*
$ a_0 b_0 a_1 a_2 b_1	a_3
a_3 $ a_0 b_0 a_1 a_2	$\mathbf{b_1}$
a_1 a_2 b_1 a_3 $ a_0	$\mathbf{b_0}$
a_2 b_1 a_3 $ a_0 b_0	a_1
a_0 b_0 a_1 a_2 b_1 a_3	$
$\mathbf{b_1}$ a_3 $ a_0 b_0 a_1	a_2
$\mathbf{b_0}$ a_1 a_2 b_1 a_3 $	a_0

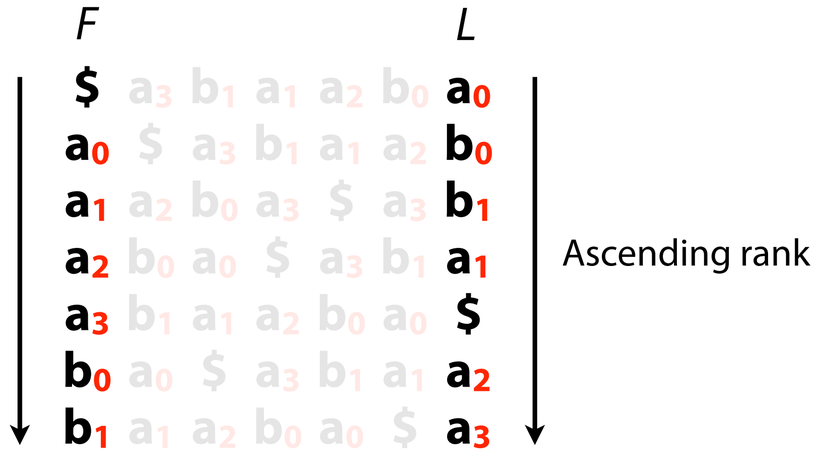
Same with **b**s: $\mathbf{b_1}$, $\mathbf{b_0}$

The Last-to-First (LF) mapping property in BWT

BWM with T-ranking:

F						*L*
$	a_0	b_0	a_1	a_2	b_1	a_3
a_3	$	a_0	b_0	a_1	a_2	b_1
a_1	a_2	b_1	a_3	$	a_0	b_0
a_2	b_1	a_3	$	a_0	b_0	a_1
a_0	b_0	a_1	a_2	b_1	a_3	$
b_1	a_3	$	a_0	b_0	a_1	a_2
b_0	a_1	a_2	b_1	a_3	$	a_0

BWM with B-ranking:

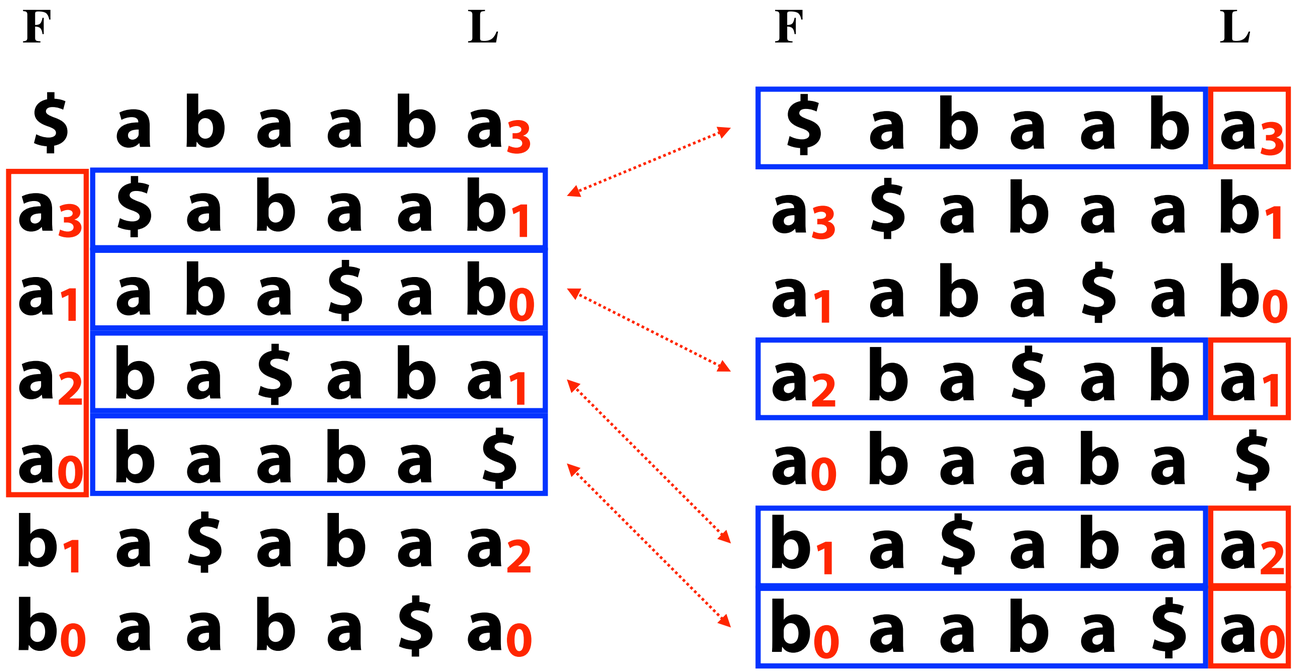


LF Mapping: The i^{th} occurrence of a character *c* in *L* and the i^{th} occurrence of *c* in *F* correspond to the *same* occurrence in *T*

However we rank occurrences of *c*, ranks appear in the same order in *F* and *L*

F now has very simple structure: a **$**, a block of **a**s *with ascending ranks*, a block of **b**s *with ascending ranks*

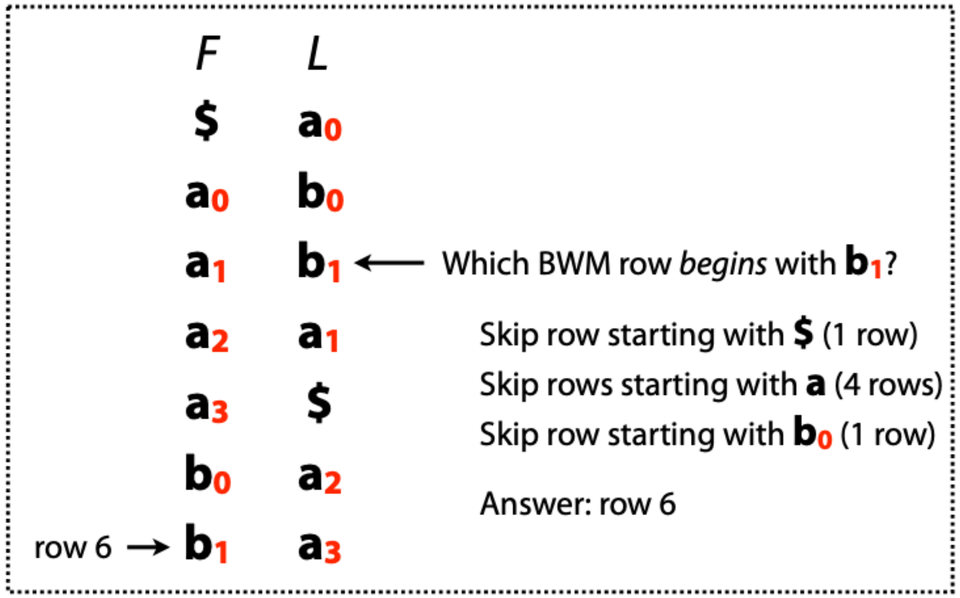
The Last-to-First (LF) mapping property in BWT



- **LF mapping** (Last-to-First mapping) is a key concept in the **Burrows-Wheeler Transform (BWT)** that enables efficient backward searching in strings. It relates the position of a character in the **last column** (BWT-transformed string) to its corresponding position in the **first column** (sorted suffixes) of the conceptual matrix used to construct the BWT.
- **LF mapping** connects the last column (BWT string) with the first column (sorted suffixes), enabling powerful backward searching capabilities in string matching algorithms

a_s are sorted by the same context in the F and the L columns

Use *LF Mapping to find the ranks of specific characters*



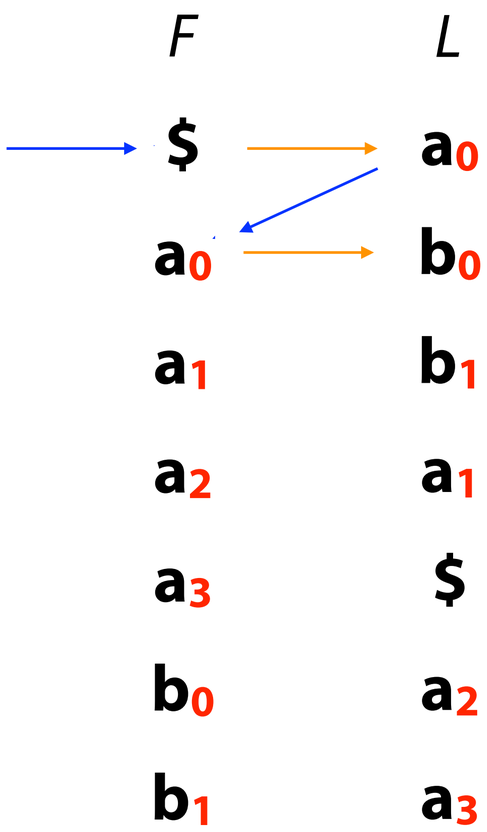
Say *T* has 300 **A**s, 400 **C**s, 250 **G**s and 700 **T**s and **$ < A < C < G < T**

Which BWM row (0-based) begins with **G100**? (Ranks are B-ranks.)

Skip row starting with **$** (1 row)
Skip rows starting with **A** (300 rows)
Skip rows starting with **C** (400 rows)
Skip 100 rows starting with **G** (100 rows)

Answer: row 1 + 300 + 400 + 100 = **row 801**

Use LF mapping to generate original strings from BWT

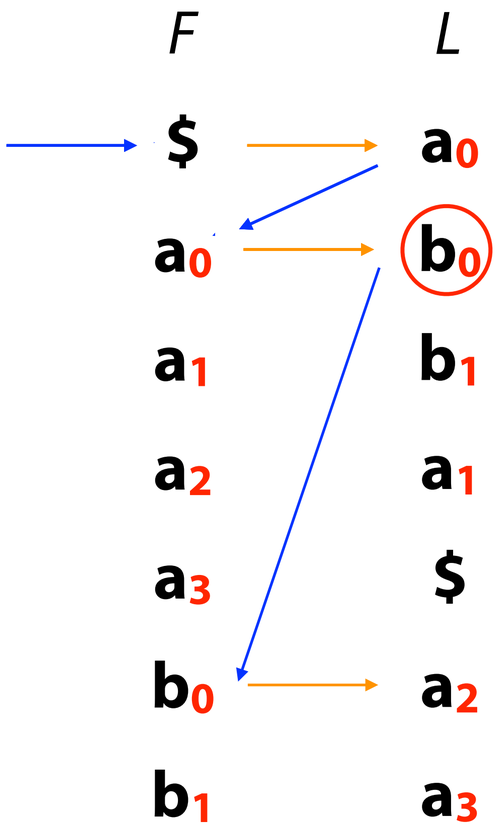


Start in first row. *F* must have **$**. *L* contains character just prior to **$**: **a0**

a0: LF Mapping says this is same occurrence of **a** as first **a** in *F*. Jump to row *beginning* with **a0**. *L* contains character just prior to **a0**: **b0**.

b0 a0 $

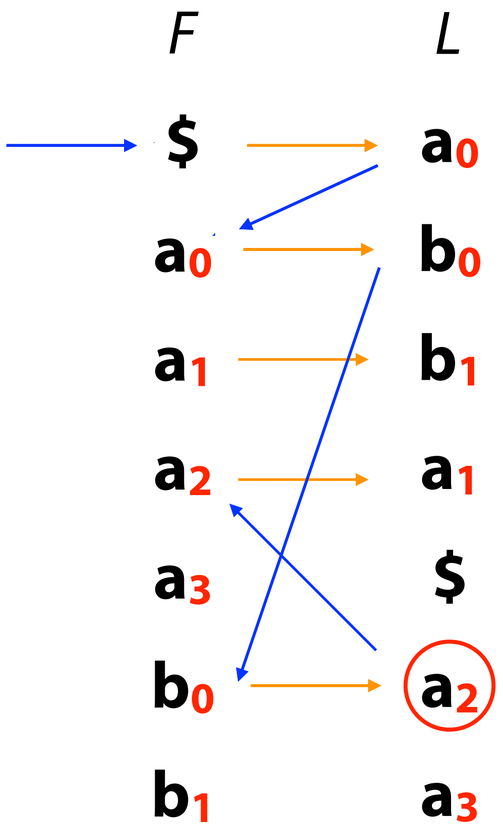
Use LF mapping to generate original strings from BWT



Repeat for b_0, get a_2

$a_2\ b_0\ a_0\ \$$

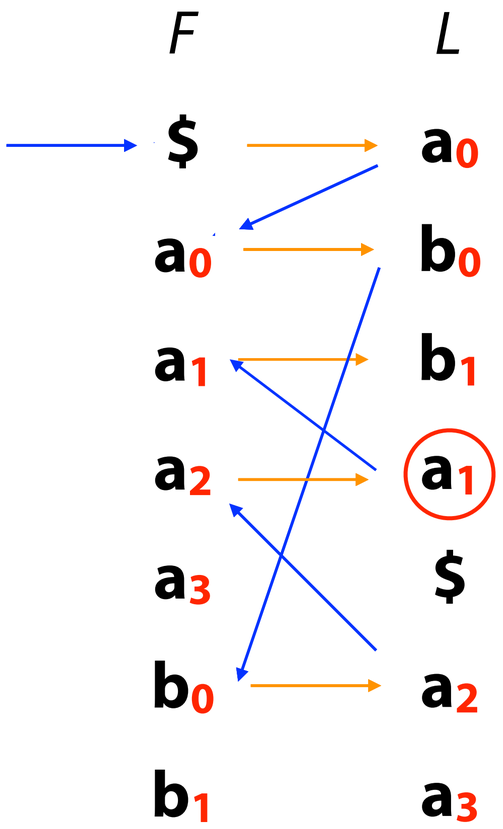
Use LF mapping to generate original strings from BWT



Repeat for a_2, get a_1

$a_1\ a_2\ b_0\ a_0\ \$$

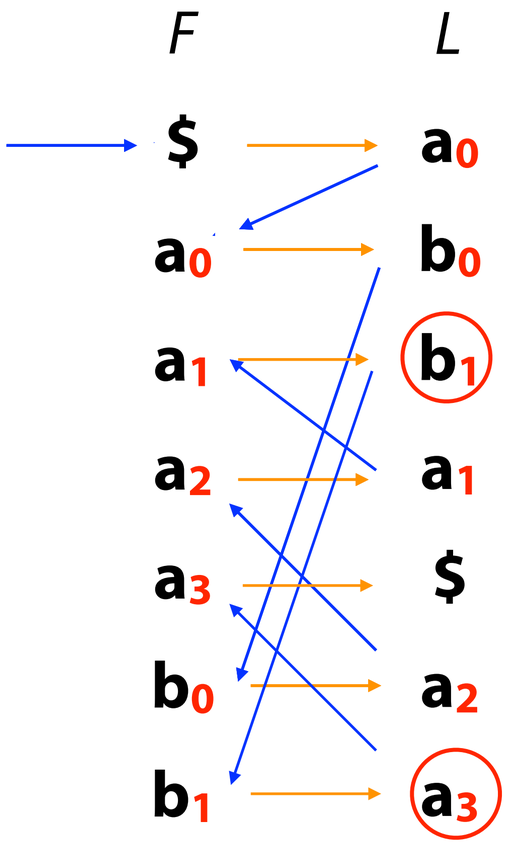
Use LF mapping to generate original strings from BWT



Repeat for a_1, get b_1

$b_1\ a_1\ a_2\ b_0\ a_0\ \$$

Use LF mapping to generate original strings from BWT



Repeat for $\mathbf{b}_1$, get $\mathbf{a}_3$

Repeat for $\mathbf{a}_3$, get $, done

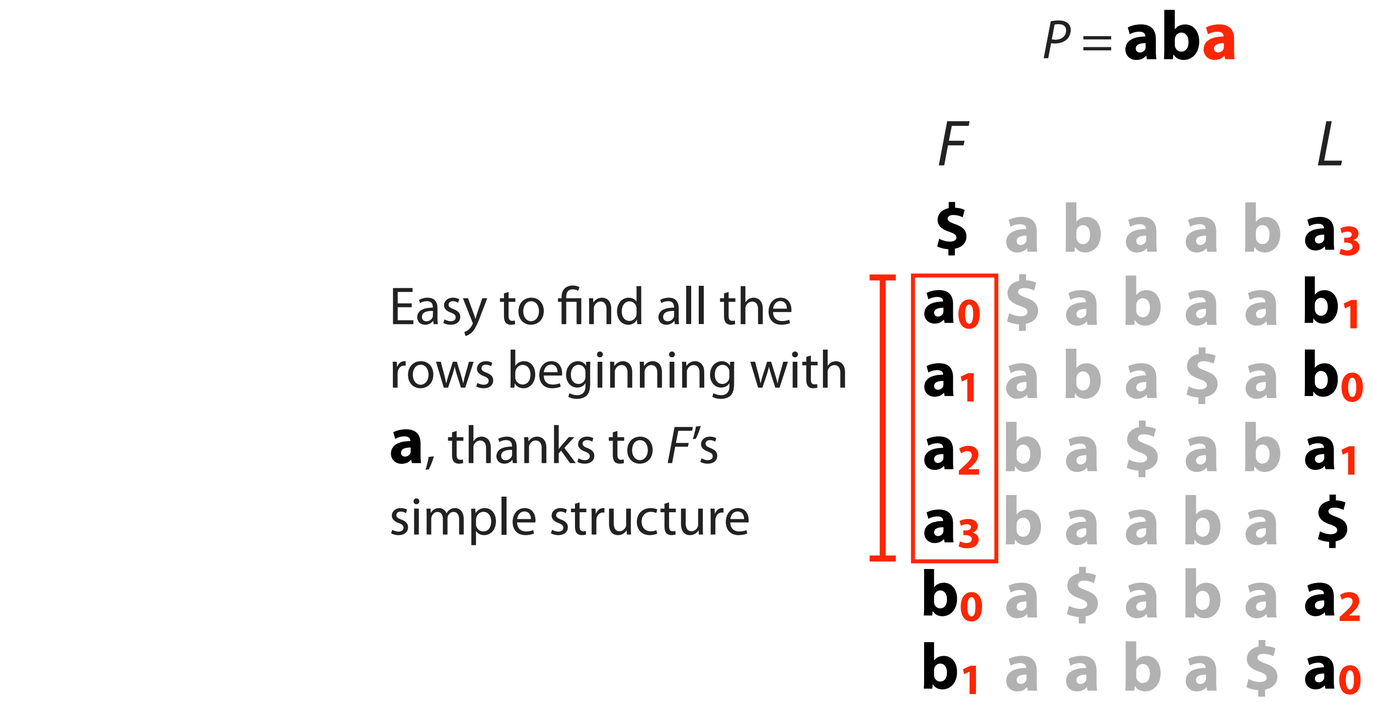
$\mathbf{a}_3\,\mathbf{b}_1\,\mathbf{a}_1\,\mathbf{a}_2\,\mathbf{b}_0\,\mathbf{a}_0\,\$ = T$

Repeated applications of LF Mapping, recreating *T* from right to left

Query a sequence pattern with a BWT sequence

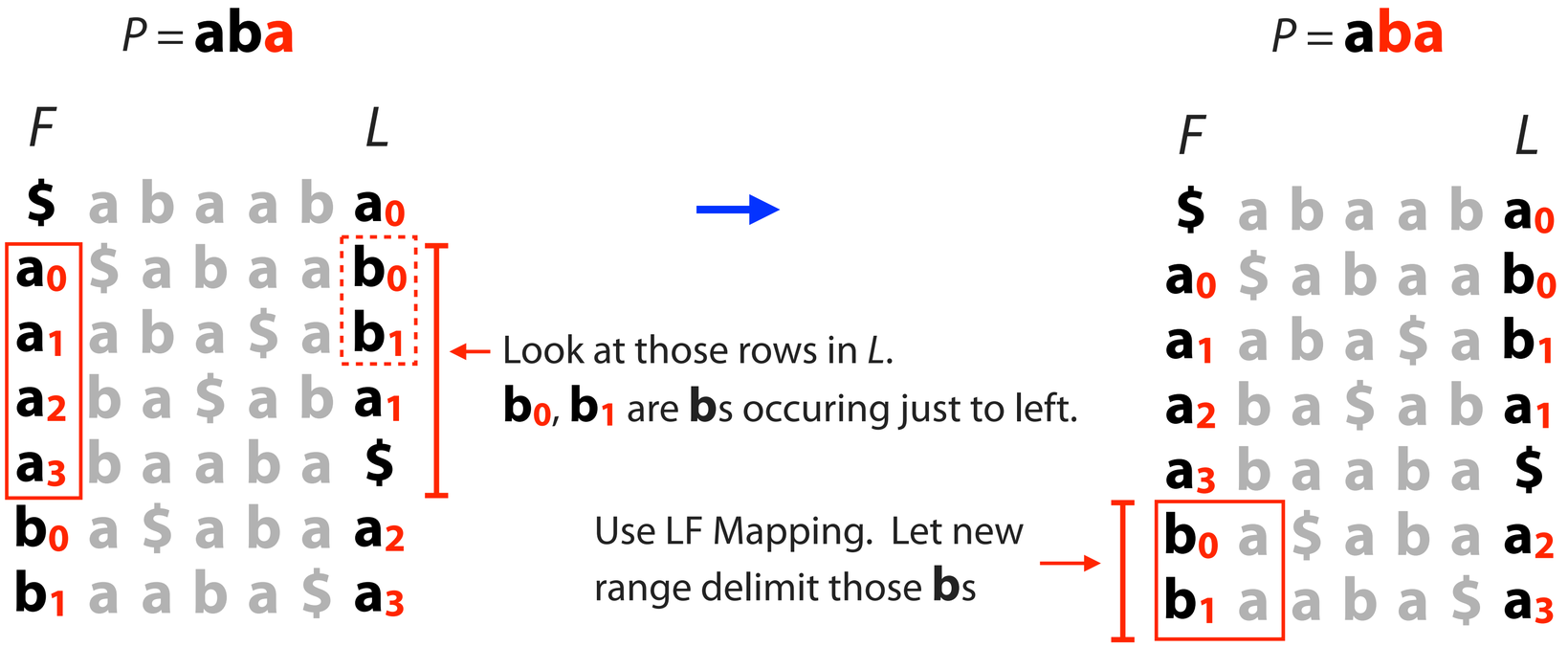
Look for range of rows of BWM(T) with *P* as prefix

Do this for *P*'s shortest suffix, then extend to successively longer suffixes until range becomes empty or we've exhausted *P*



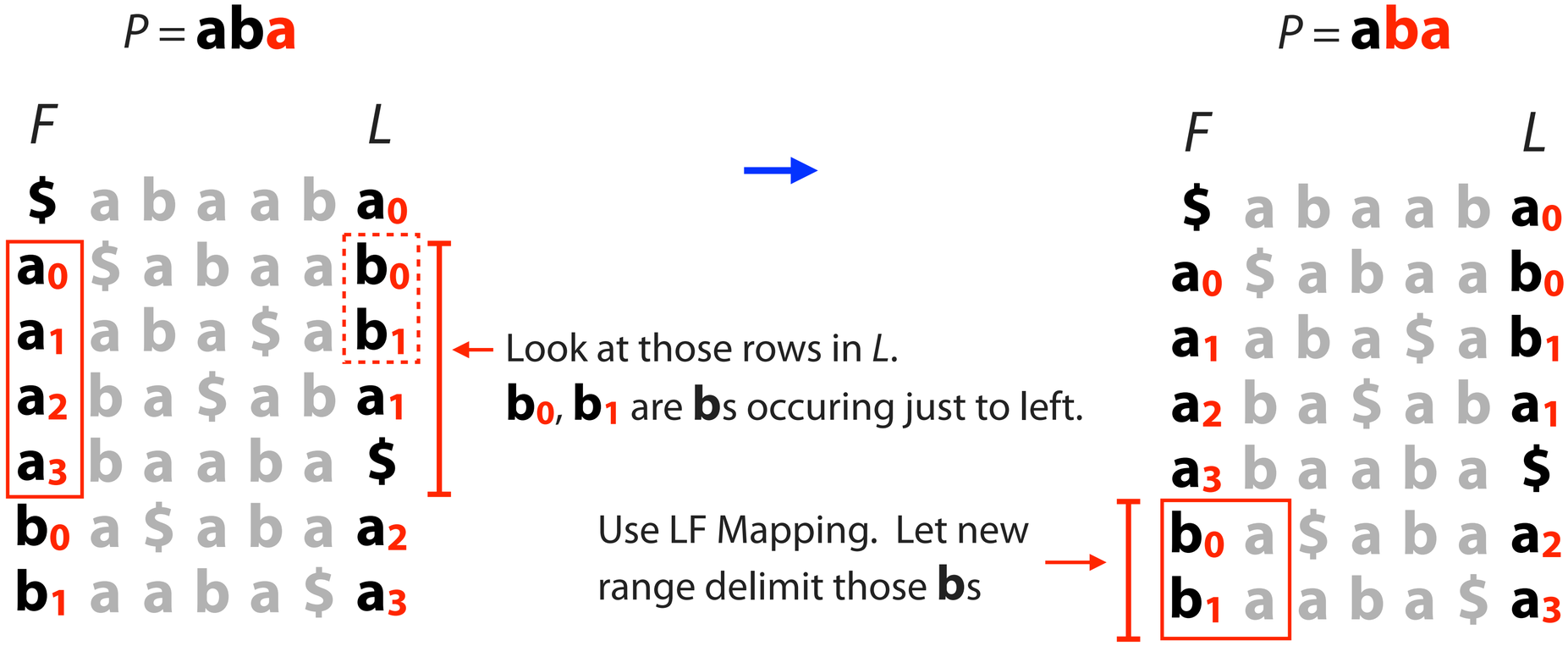
Query a sequence pattern with a BWT sequence

We have rows beginning with **a**, now we seek rows beginning with **ba**



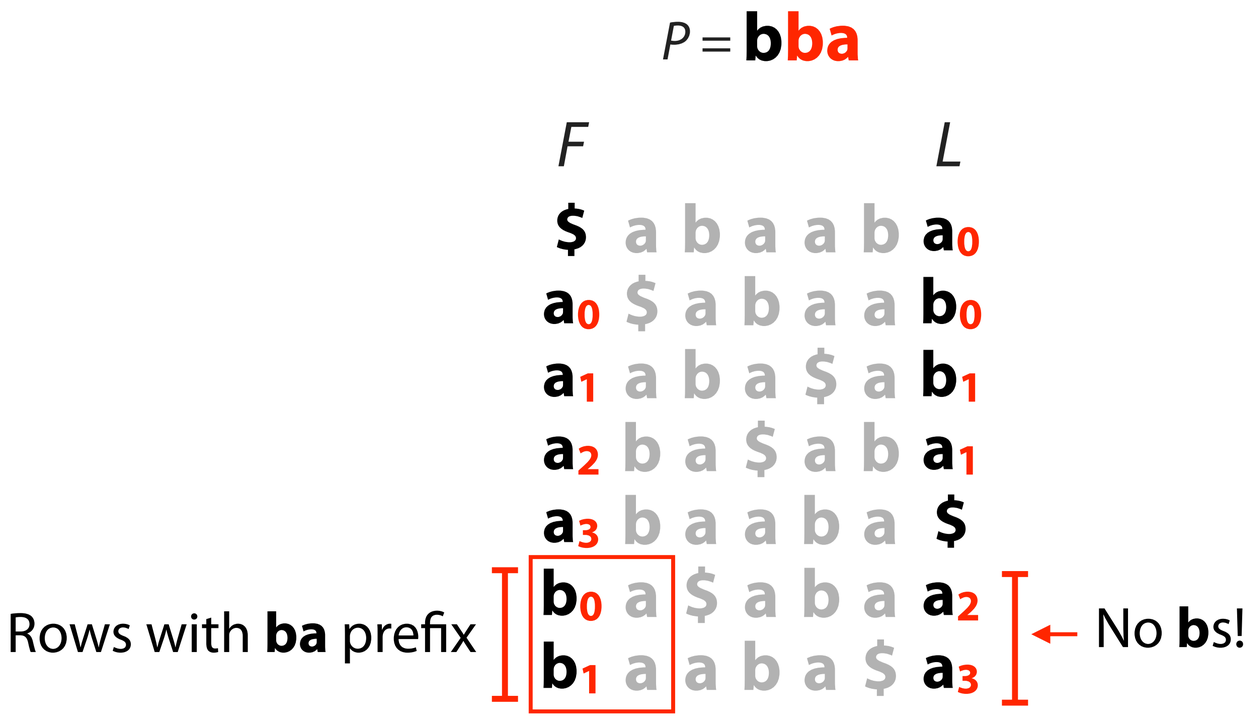
Query a sequence pattern with a BWT sequence

We have rows beginning with **a**, now we seek rows beginning with **ba**



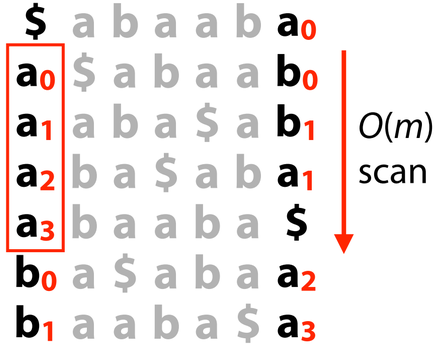
Query a sequence pattern with a BWT sequence

When *P* does not occur in *T*, we will eventually fail to find the next character in *L*:



Issues related to query matching

(1) Scanning for preceding character is slow

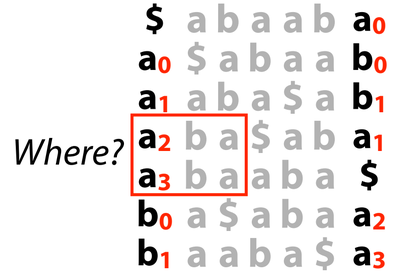


(2) Storing ranks takes too much space

m integers

```
def reverseBwt(bw):
    """ Make T from BWT(T) """
    ranks, tots = rankBwt(bw)
    first = firstCol(tots)
    rowi = 0
    t = "$"
    while bw[rowi] != '$':
        c = bw[rowi]
        t = c + t
        rowi = first[c][0] + ranks[rowi]
    return t
```

(3) Need way to find where matches occur in *T:*

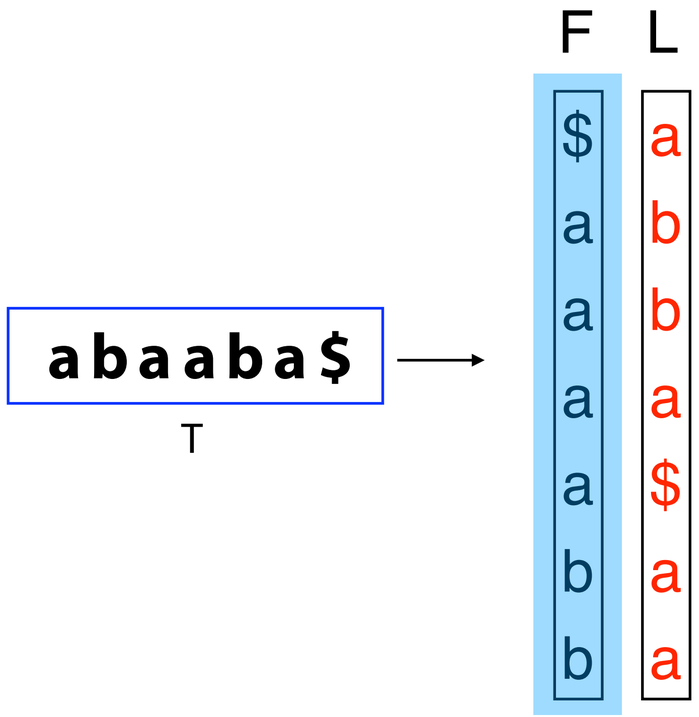


- (1)To find the positions of **a**, you would need to scan the entire BWT string linearly.After finding all occurrences of **a**, you then proceed to find the preceding character **b**, which again requires a scan. Finally, you search for the last **a** in the pattern. This requires repeated linear scans over the BWT, which is inefficient and time-consuming, especially for large texts like the human genome.
- (2) When matching the pattern **“aba”**, you need to know how many times each character (such as **a**) appears in the BWT up to a certain position. This requires storing the **rank** of each character at every position in the BWT.
- (3) After finding the pattern **“aba”** in the BWT, you need to map the match back to its original position in the text. The BWT alone doesn’t store the original positions of the characters, making it difficult to determine where the pattern appears in the text.

FM-index

- The **FM-index** is a data structure designed to support fast and memory-efficient pattern searching in large strings or texts, such as genomes. It is built on top of the **Burrows-Wheeler Transform (BWT)** and provides a way to search for substrings (patterns) in a compressed space, while still allowing efficient navigation through the transformed string.
- FM-index addresses key issues in pattern matching with the BWT, such as:
  - Slow scanning for preceding characters.
  - Storing character ranks takes too much space.
  - Difficulty in finding match positions in the original text.

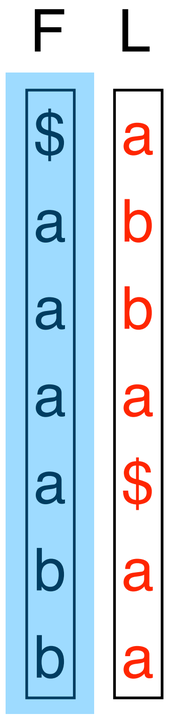
The C-array in FM-index



- $ starts at position 0 -> C($) = 0.
- a starts at position 1 -> C(a) = 1.
- b starts at position 5 -> C(b) = 5

- The C array **C(ch)**: How many characters in the BWT are lexicographically smaller than **ch**. It helps in efficiently mapping the BWT to the first column (sorted suffixes), enabling fast pattern matching.
- For example, if you're trying to find where **a** appears in the sorted first column, you use **C(a) = 1** to know that the block of **a's** starts at position **1** in the first column.

The Occ function in FM-index



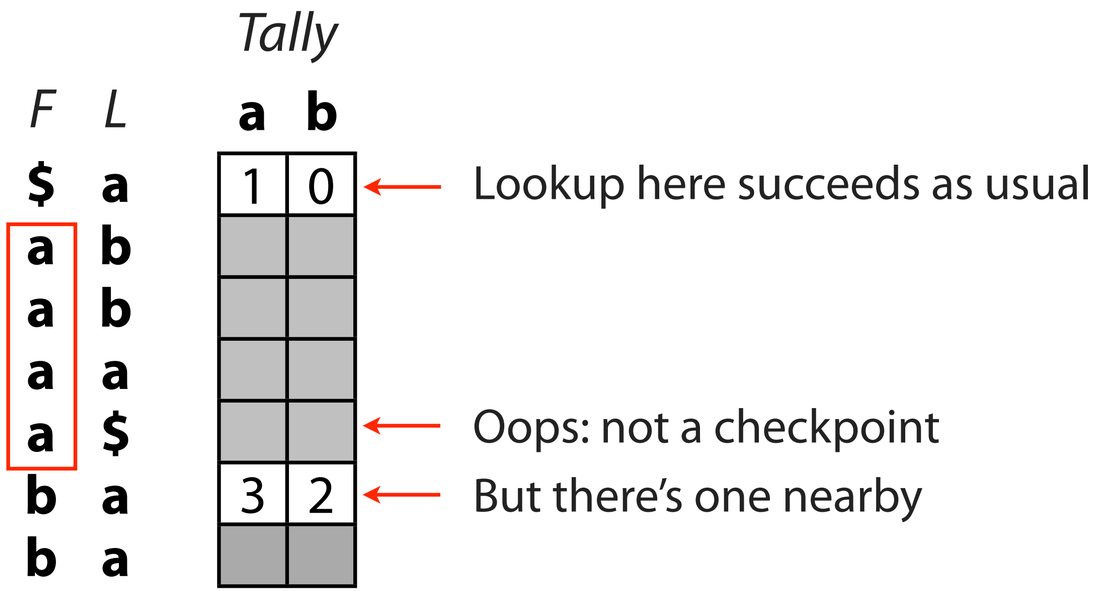
Position (i)	BWT	Occ(a, i)	Occ(b, i)	Occ($, i)
0	a	1	0	0
1	b	1	1	0
2	b	1	2	0
3	a	2	2	0
4	$	2	2	1
5	a	3	2	1
6	a	4	2	1

Table 1: The Occ function for the BWT of "abaaba$" (BWT = "abba$aa")

- **Occ(ch, i)**: For a character **ch** and a position **i** in the BWT, the **Occ function** returns the number of times **ch** has appeared in the BWT from the start (position 0) up to and including position **i.**
- This function allows quick lookup of how many occurrences of a character have been seen up to a specific point in the BWT, which is essential for navigating through the BWT string during the search.

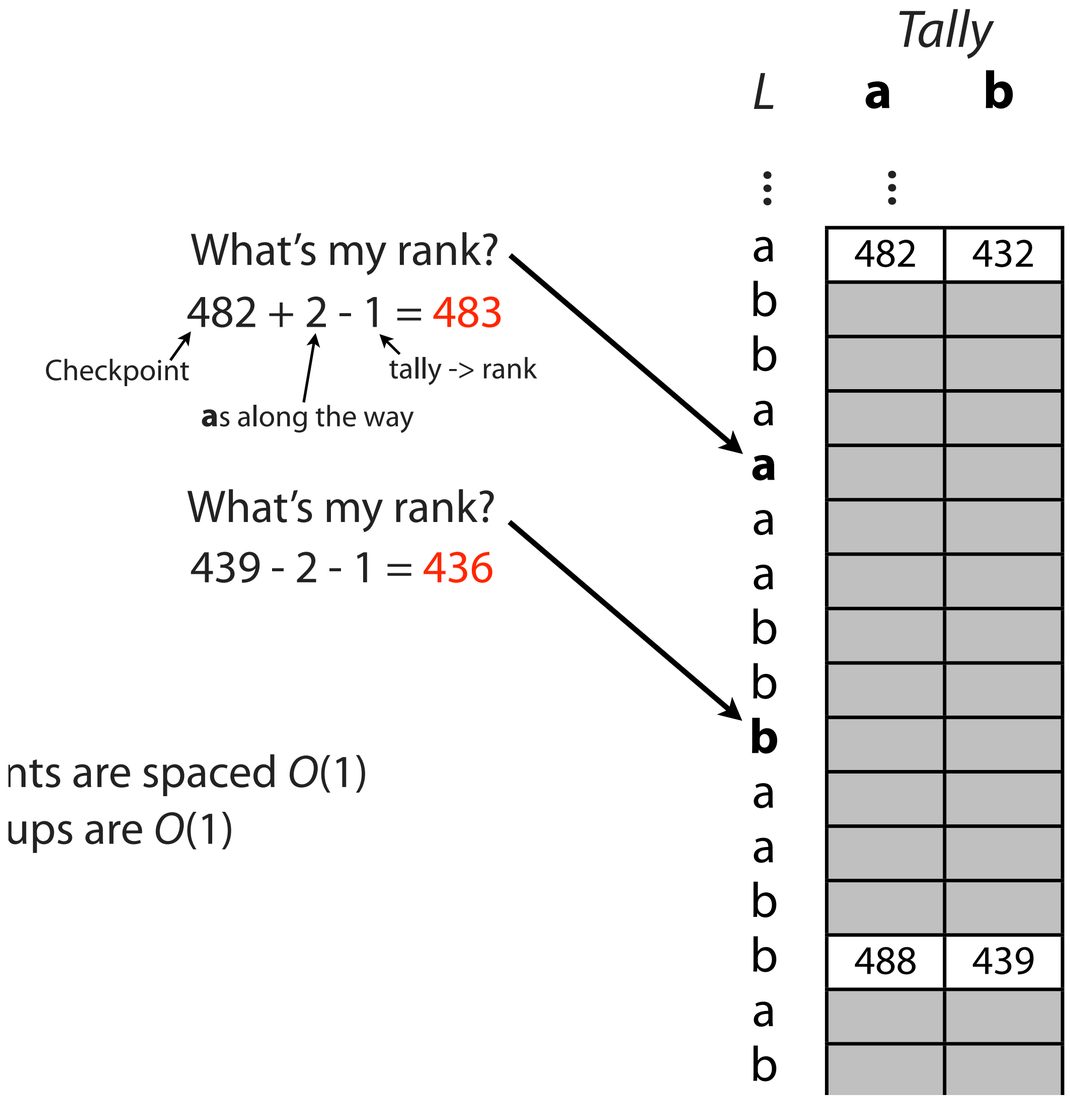
The Occ function with checkpoint in FM-index

Another idea: pre-calculate # **a**s, **b**s in *L* up to *some* rows, e.g. every 5th row.
Call pre-calculated rows *checkpoints*.



To resolve a lookup for character *c* in non-checkpoint row, scan along *L* until we get to nearest checkpoint. Use tally at the checkpoint, *adjusted for # of* c*s we saw along the way*.

The Occ function with checkpoint in FM-index

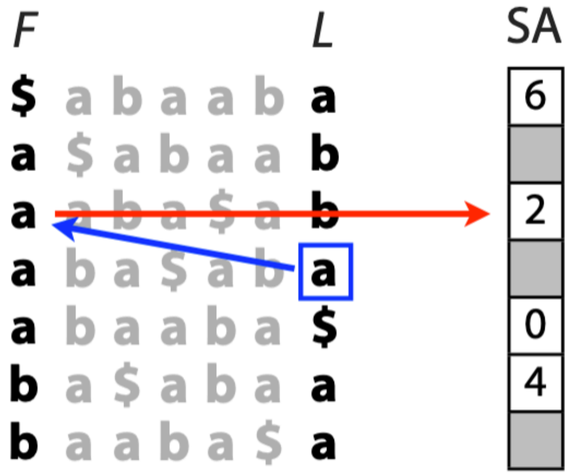


- **Checkpoints** reduce memory usage by storing **Occ values at intervals**. When a value between checkpoints is needed, it can be reconstructed by a short scan of the BWT between the checkpoint and the desired position, making the overall process much faster than scanning the entire BWT.
- **Direct Access**: For positions at a checkpoint, the **Occ function** can directly return the cumulative count. **Efficient Reconstruction**: For positions between checkpoints, the algorithm computes the cumulative count by using the nearest checkpoint and **scanning a small portion** of the BWT between the checkpoint and the current position. This makes it much faster than scanning the entire BWT.

FM-index uses partial suffix array to map where match occurs

6	$
5	a $
2	a a b a $
3	a b a $
0	a b a a b a $
4	b a $
1	b a a b a $

Suffix array of T



- **Suffix Array** tells you where each suffix of the original string starts.
- After finding the range of matching rows in the BWT using backward search, the **suffix array** is used to map those rows back to the original text, giving the starting positions of the pattern.

BWT and FM-index

- BWT: Transforms text to improve compressibility. Forms the basis for efficient text indexing and searching. Provides a regular structure that facilitates fast pattern matching.

- FM-index: Combines BWT with additional data structures for a space-efficient, compressed index. Supports fast pattern matching with low time complexity. Handles large datasets effectively with minimal memory overhead.Provides a practical solution for searching and querying in massive texts, such as genomes.

- Memory usage: BWT: ~3 GB (same as the genome size). FM-index: ~6-15 GB for the human genome, depending on compression and checkpoint intervals.

- Complexity: Searching for a pattern of length **m** in the genome using the FM-index is done in **O(m)** time, independent of the genome's size.

BWA

- Burrows-Wheeler transform algorithm with FM-index using suffix arrays.
- BWA can map low-divergent sequences against a large reference genome, such as the human genome. It consists of three algorithms:
  - BWA-backtrack (Illumina sequence reads up to 100bp)
  - BWA-SW (more sensitive when alignment gaps are frequent)
  - BWA-MEM (maximum exact matches)
- BWA SW and MEM can map longer sequences (70bp to Mbp), but BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate.
- BWA-MEM also has better performance than BWA-backtrack for 70-100bp Illumina reads.

STAR

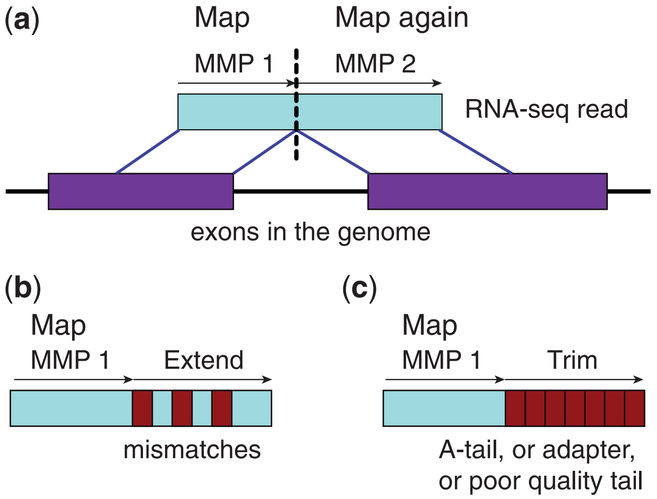
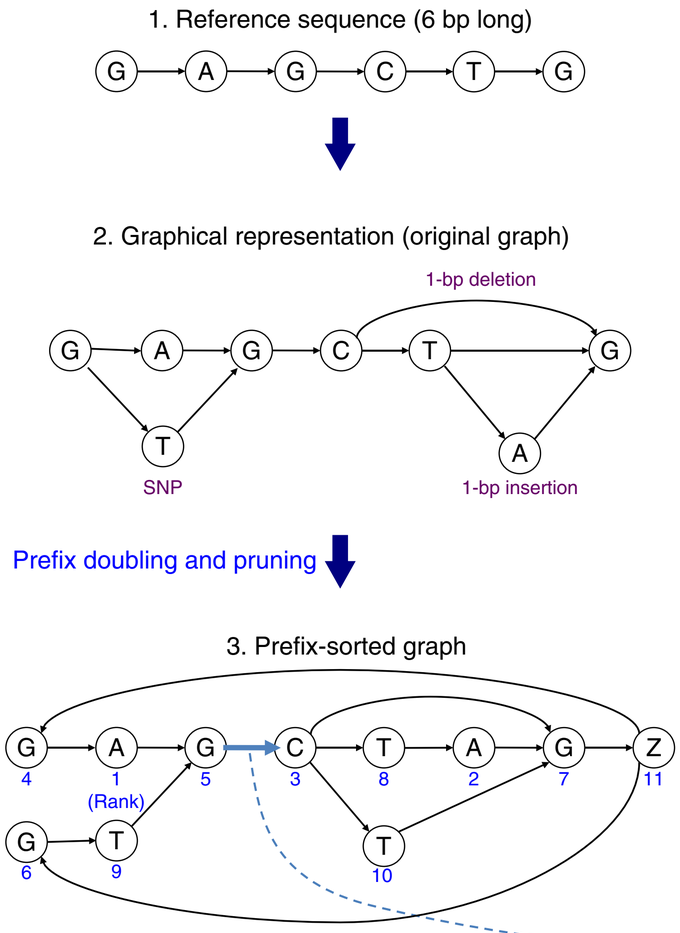


Fig. 1. Schematic representation of the Maximum Mappable Prefix search in the STAR algorithm for detecting (**a**) splice junctions, (**b**) mismatches and (**c**) tails

- Splicing Transcripts Alignment to a Reference
- Two steps: Seed searching and clustering/stitching/ scoring (find MMP -maximal mappable prefix using Suffix Arrays)
- Fast splice aware aligner, high memory (RAM) footprint
- Can detect chimeric transcripts

HISAT2

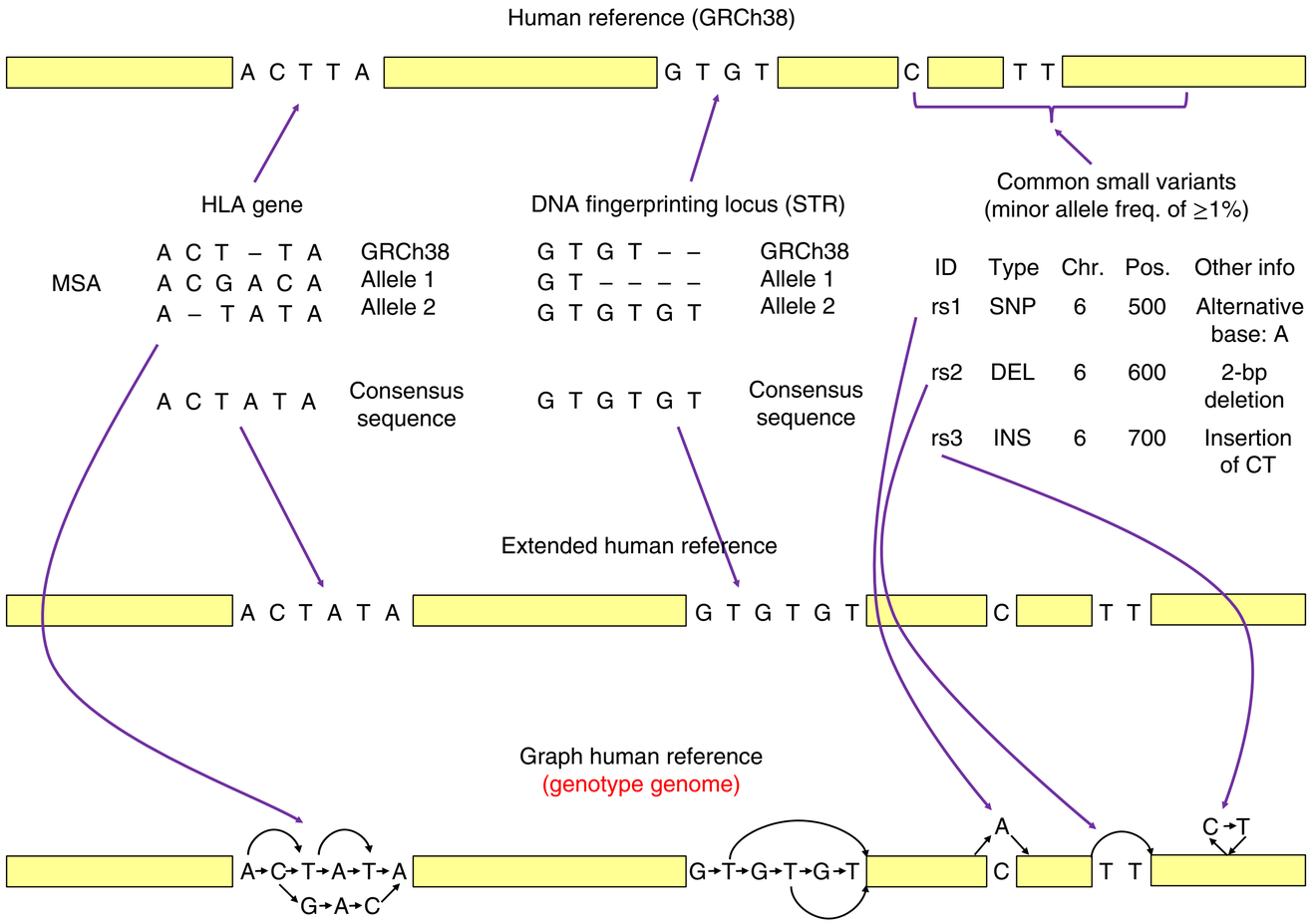


4. Tabular representation of the prefix-sorted graph

Outgoing edge(s)			Incoming edge(s)	
Node rank	First		Last	Node rank
1	A		G	1
2	A		T	2
3	C C C		G Z A	3 4 5
4	G		T	
5	G		Z	6
6	G		A C T	7
7	G			
8	T			
9	T		C	8
10	T		G	9
11	Z Z		C G	10 11

- HISAT2 is a successor to the original HISAT (Hierarchical Indexing for Spliced Alignment of Transcripts) tool and was developed to provide fast and accurate alignment of sequencing data, particularly in the context of RNA sequencing.
- Based on an extension of BWT for graphs, HISAT2 implemented a graph FM index (GFM) that represents a population of human genomes as well as a large set of small GFM indexes that collectively cover the whole genome.

HISAT2



Construction of the graph human reference, that is a Genotype genome.

Summary

- BLAST
- BWT and FM-index

Next lecture

- Phylogenetic tree construction
- Hidden Markov Model