Introduction to Bioinformatics for Computer Scientists

Lecture 4

BLAST & Genome assembly

Alexey Kozlov
with contributions of Solon Pissis and Tomas Flouri

Exelixis Lab

November 9, 2017

Karlsruhe Institute of Technology

Heidelberg Institute for Theoretical Studies

HITS

Today's agenda

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

Sequence databases

- NCBI GenBank/INSDC →”all” DNA+proteins
  - > 200 million sequences as of Nov. 2017
  - “primary” database → sequences submitted and annotated by the respective authors (biologists)
- UniProt → proteins
  - extensive annotations (3D structure, functions etc.)
- SILVA, UNITE, BOLD ... → “barcoding” genes
  - quality filtering, curated taxonomy, sample geodata...
  - useful for species identification/classification

Sequence annotation in GenBank

Bacteroides galacturonicus 16S ribosomal RNA gene, partial sequence

GenBank: DQ497994.1

FASTA Graphics

Go to:

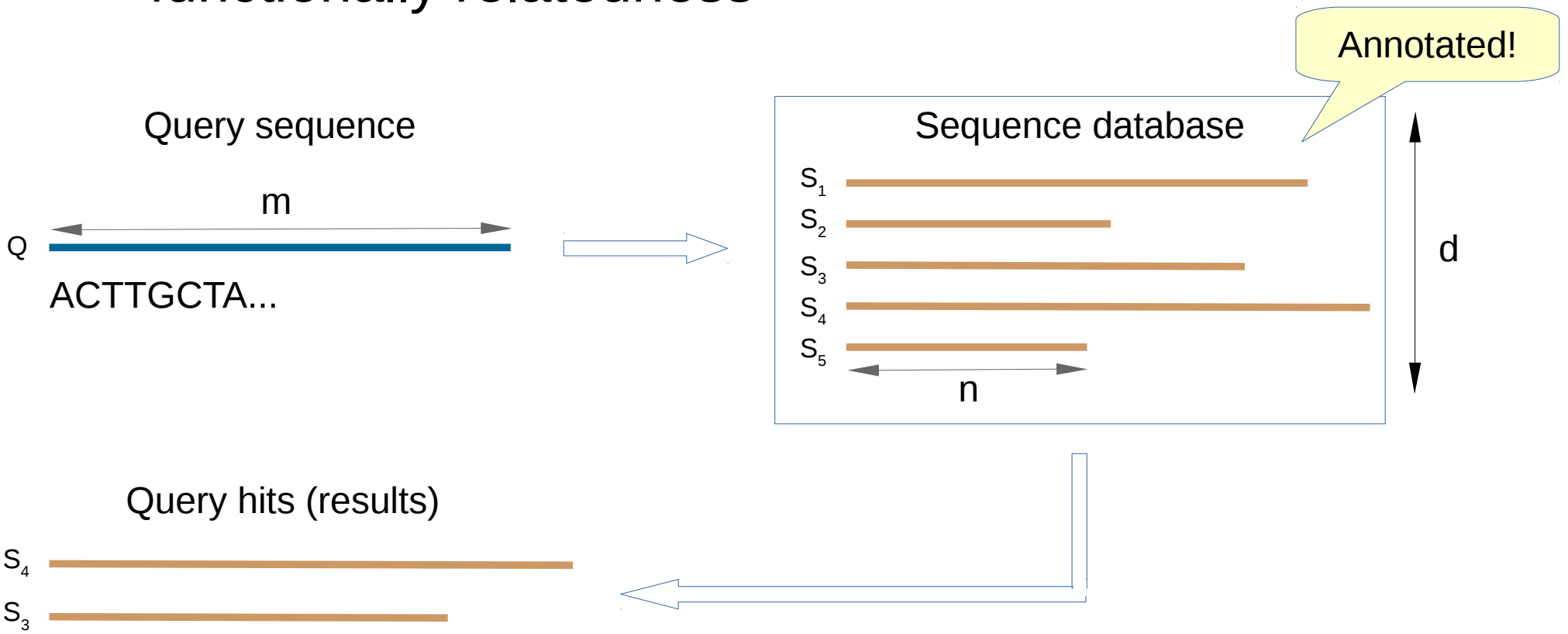
```
LOCUS       DQ497994               1431 bp    DNA     linear   BCT 01-JUN-2006
DEFINITION  Bacteroides galacturonicus 16S ribosomal RNA gene, partial
            sequence.
ACCESSION   DQ497994
VERSION     DQ497994.1  GI:95062834
KEYWORDS    .
SOURCE      Bacteroides galacturonicus
  ORGANISM  Bacteroides 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  JOURNAL   Unpublished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             Location/Qualifiers
     source          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 tgaagtcttt tatgactgag
       61 tggcggacgg gtgagtaacg cgtgggtaac ctgcctcata cagggggata gcagctggaa
      121 acggctggta ataccgcata agcgcacagt accacatggt acagtgtgaa aaactccggt
      181 ggtatgagat ggacccgcgt ctgattagct tgttggcggg gtaacggccc accaaggcga
```

entry name:
usually includes organism and gene name

organism name and taxonomy (=biological classification)

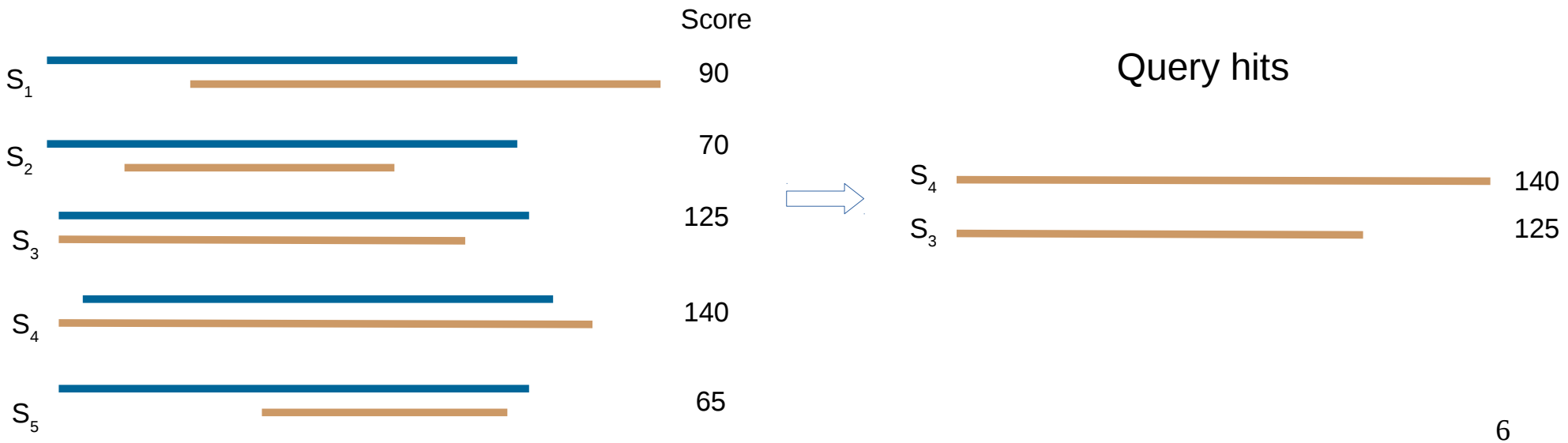
Searching for similar sequences

- Assumption:
  - sequence similarity ⇔ evolutionary and/or functionally relatedness



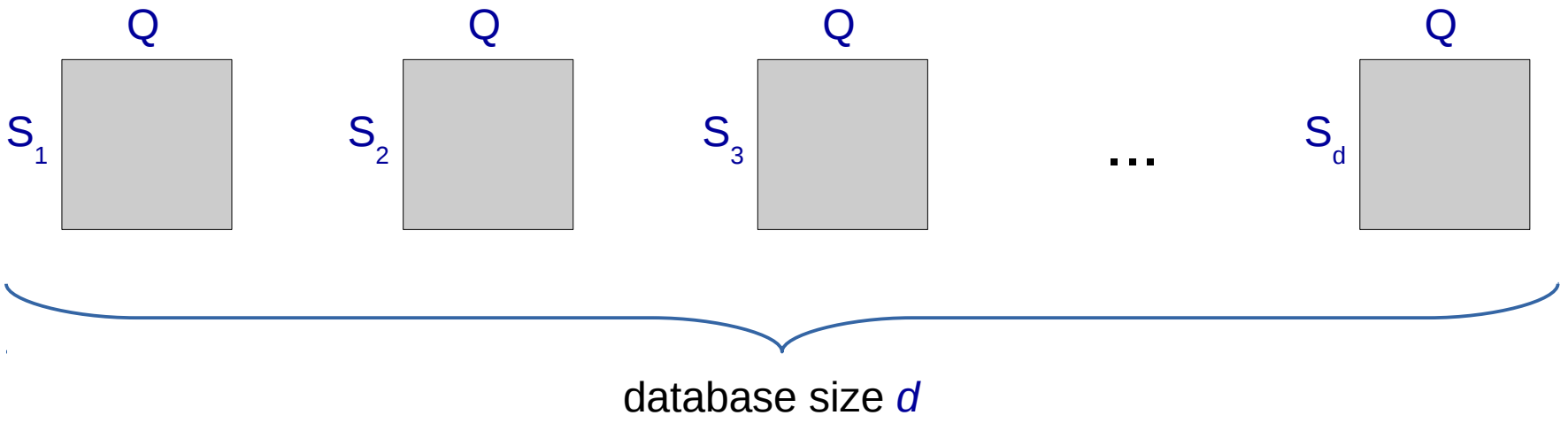
Naïve approach

- Use Smith-Waterman algorithm to build a pair-wise alignment of query sequence *Q* with every sequence in the database $S_1 ... S_d$
- 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:



- 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
  - Original paper from 1990 has over 50 000 citations[1]
- Available as both stand-alone application and web service
  - BLAST on NCBI GenBank database:

    http://blast.st-va.ncbi.nlm.nih.gov/Blast.cgi

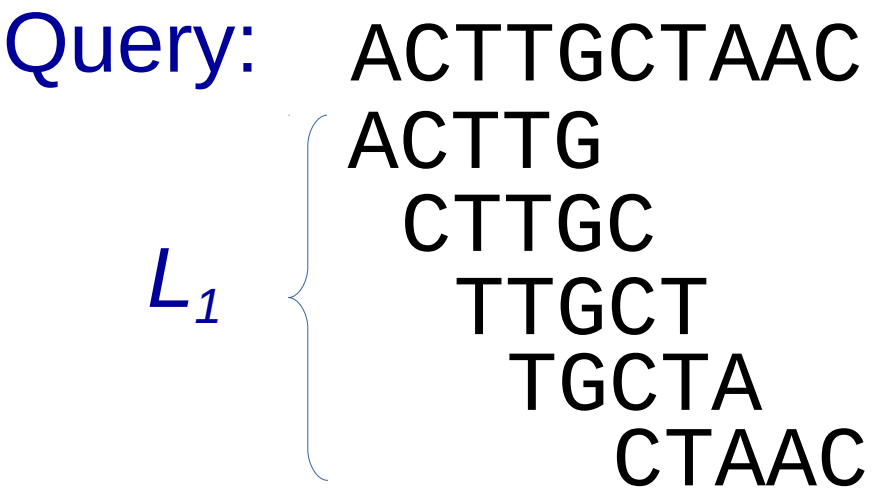
[1] Altschul, Stephen; Gish, Warren; Miller, Webb; Myers, Eugene; Lipman, David (1990). "Basic local alignment search tool". *Journal of Molecular Biology* 215 (3): 403–410. doi:10.1016/S0022-2836(05)80360-2. PMID 2231712

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

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



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 → T T G C T

T T G A T

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

W_i → T T G C T

T G C C T

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

BLAST: seeding (3)

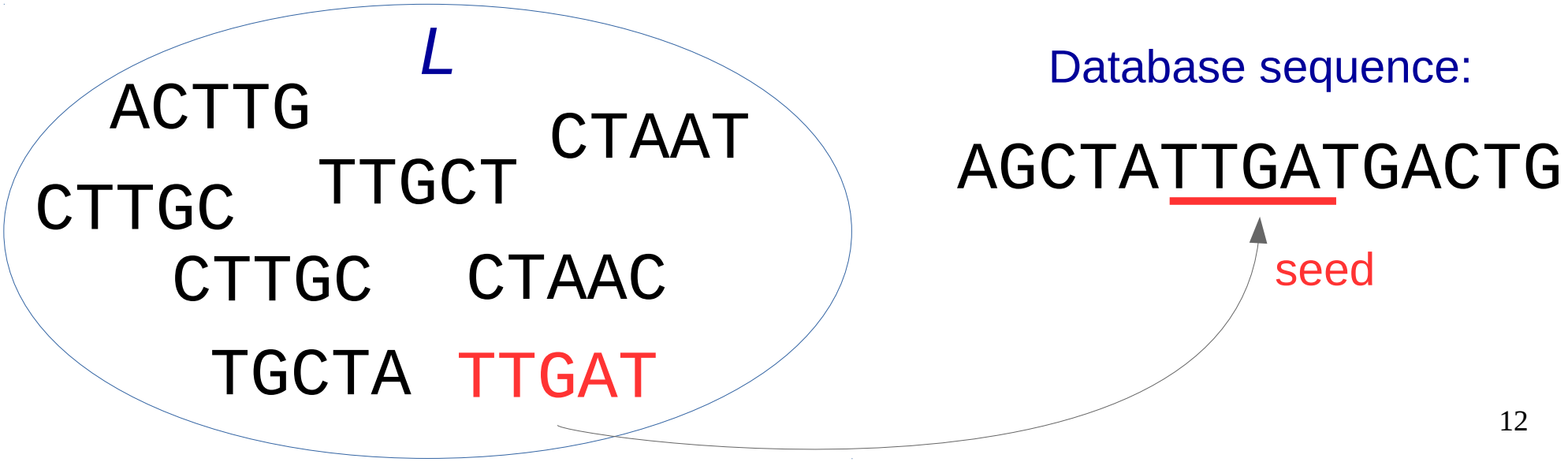
- Combine all subwords' neighborhoods of a single query sequence

  $L_2 = \cup\ N_i$

- Build a final list by adding the subwords themselves

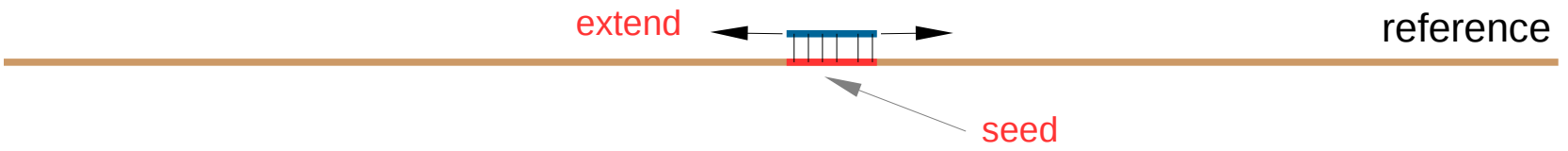
  $L = L_1 \cup L_2$

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



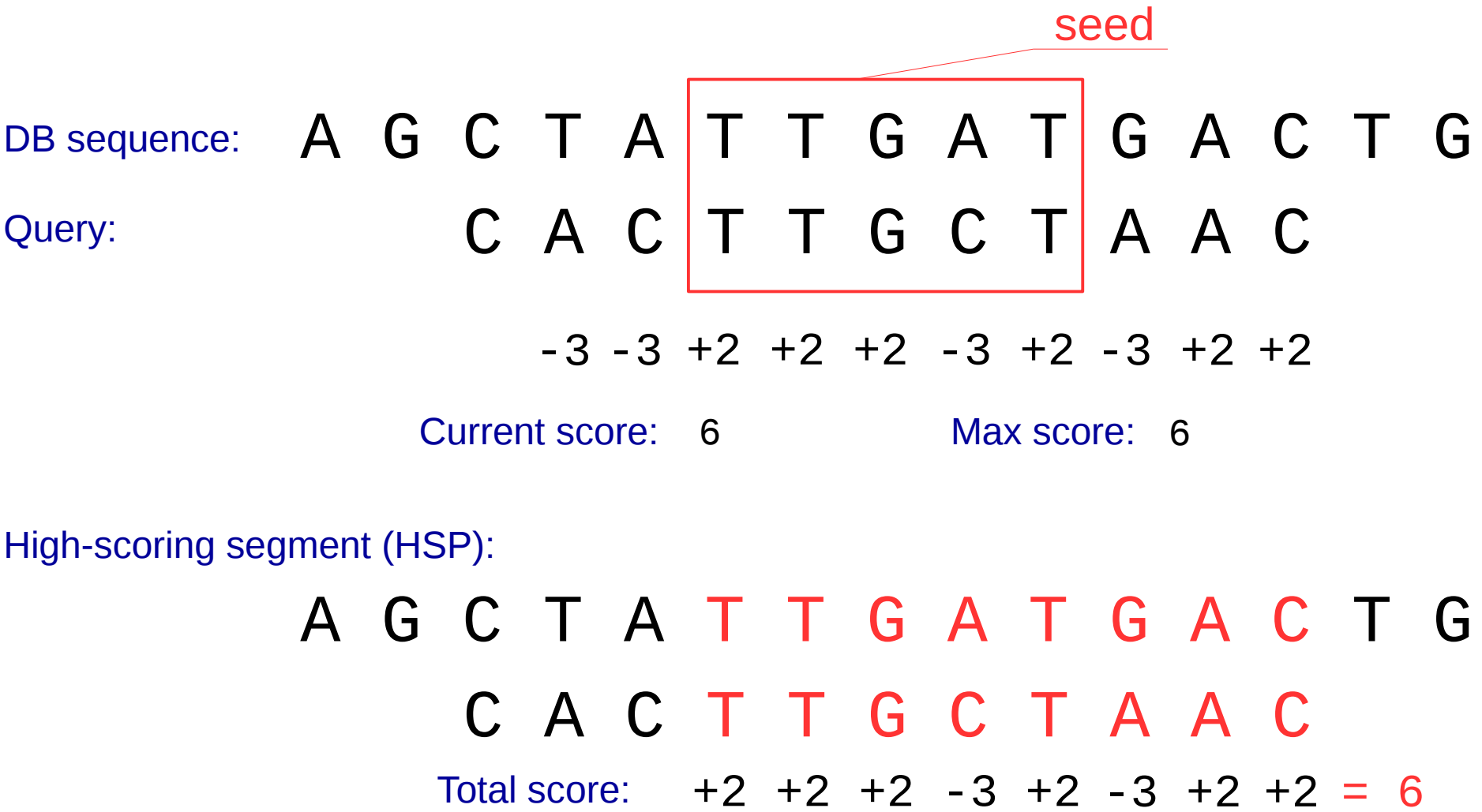
BLAST: extension

- 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)

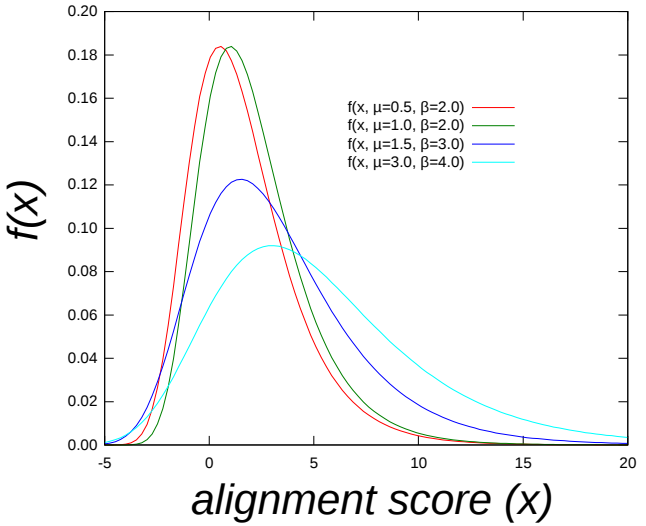
- Example with *X := 3*



BLAST: evaluation

- 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* is given by the equation

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

- Parameters *λ* and *μ* 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*:

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

- The e-value is the expected number of times that an unrelated database sequence would obtain a score *S* higher than *x by chance*

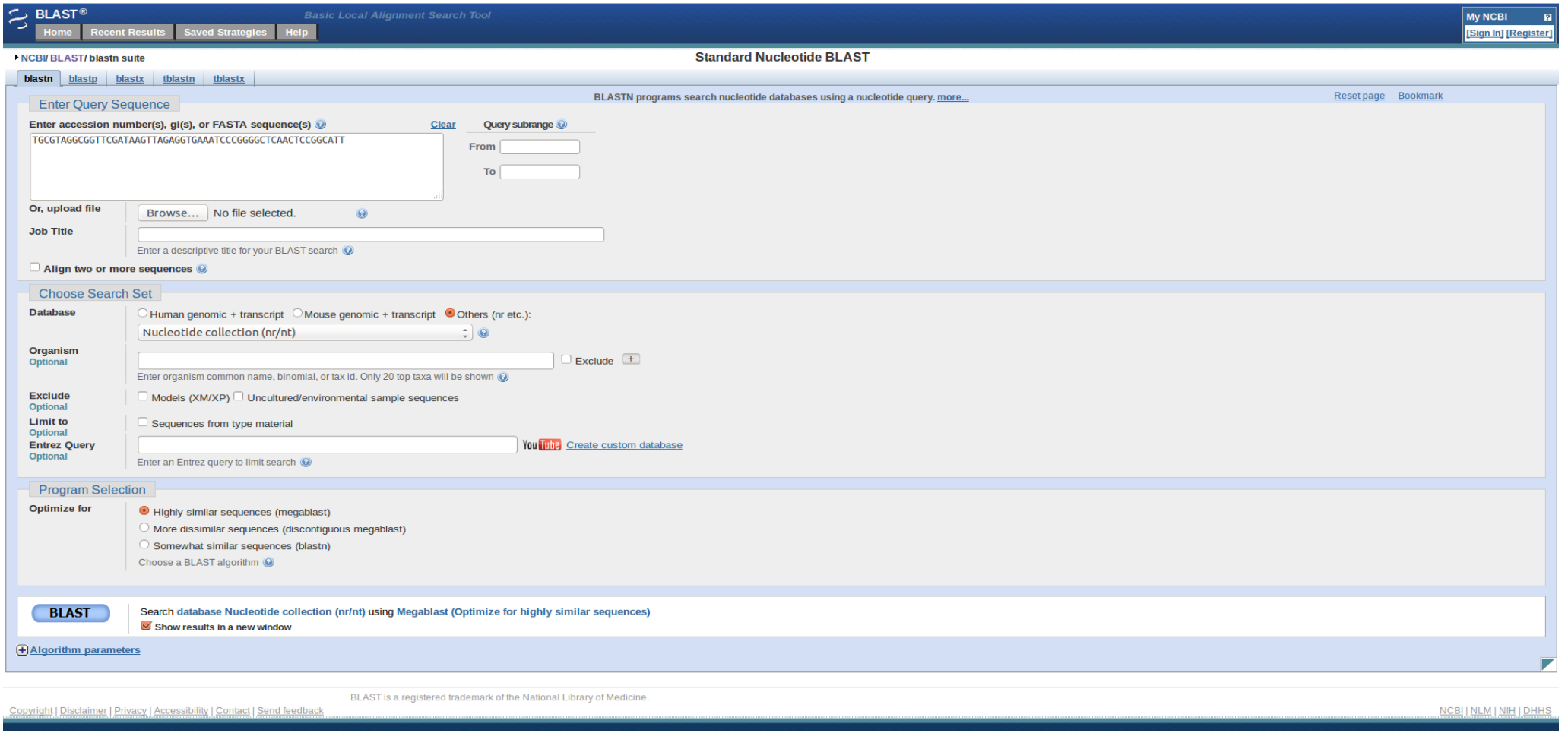
BLAST flavors

- BLASTN: both the database and the query are nucleotide sequences
- BLASTP: both the database and the query are protein sequences
- BLASTX: the database are protein sequences and the query is a nucleotide translated into a protein sequence
- TBLASTN: the database are nucleotides translated into protein sequences and the query is a protein sequence
- TBLASTX: both the database and the query are nucleotides translated into protein sequences

Beyond BLAST

- BLAST implicitly assumes that:
  - The database is large and growing (like GenBank)
  - The number of queries is relatively small (compared to DB size)
  - Pair-wise comparison is sufficient to detect relatedness
- 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
- In this situation, database pre-processing becomes the method of choice:
  - 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 training time

BLAST live demo

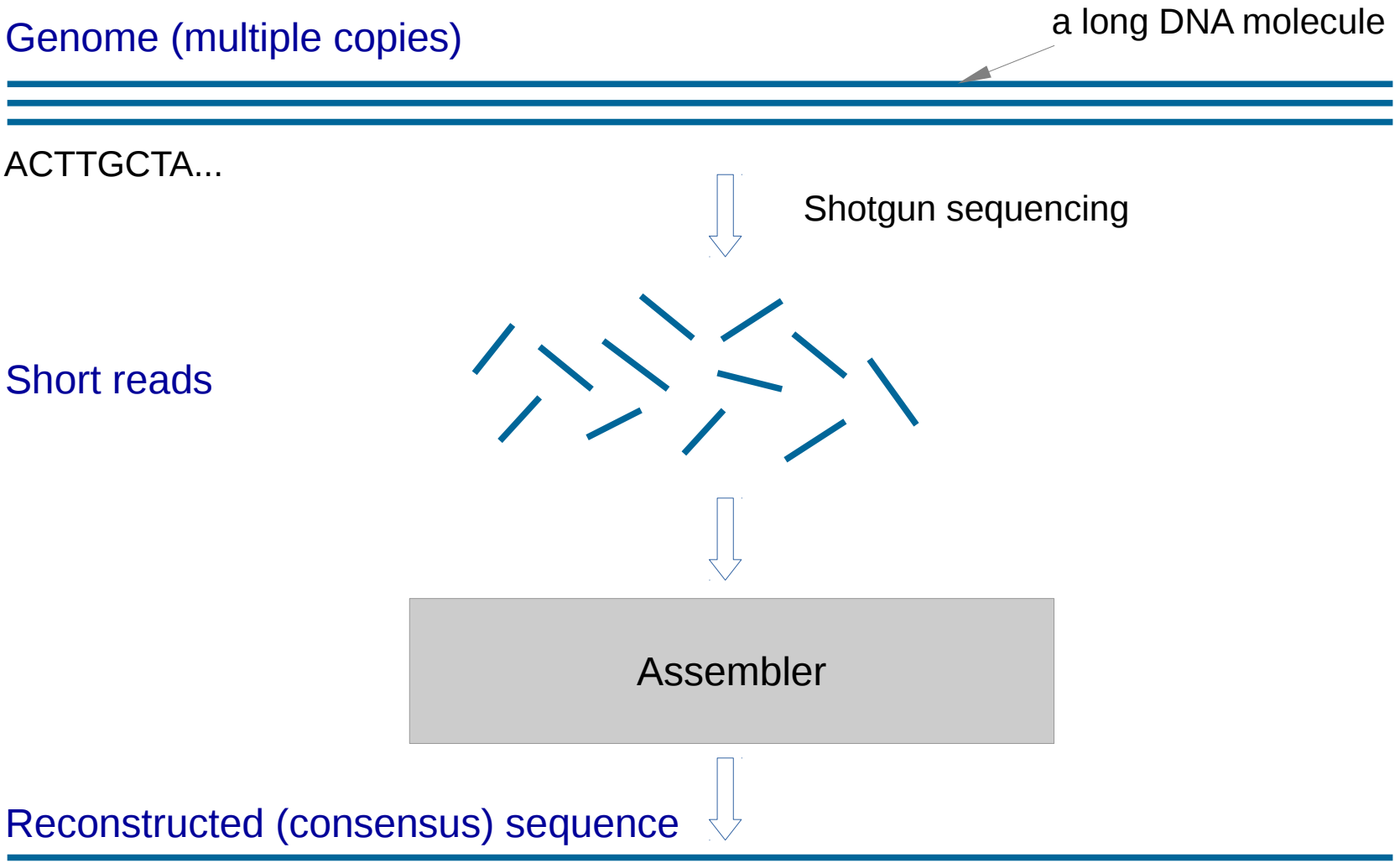


Today's agenda

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

Genome assembly

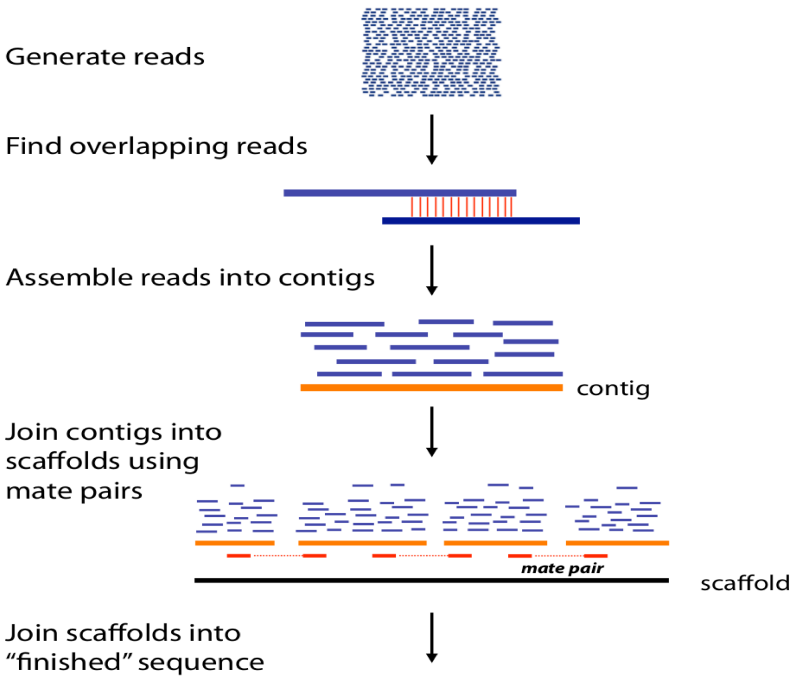
- A very simplified workflow:



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

De novo assembly: overview



ACGTGCTCCCTTTAGAGAGGCTTCCAAT...

- Assemble reads into contigs using overlaps
- 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
- Two basic approaches for *de novo* assembly: overlap graphs and de Brujin graphs

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)

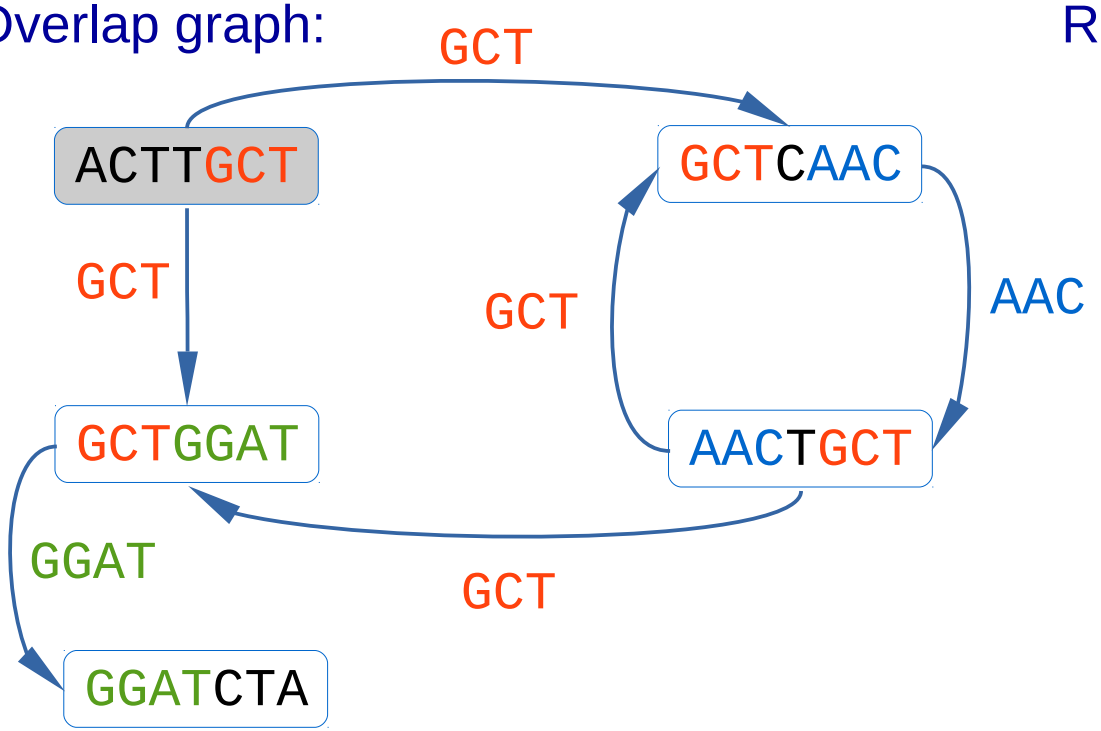
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
      AACTGCT
   GCTCAAC
         GCTGGAT
             GGATCTA
```

Overlap graph:



Reconstructed sequence:

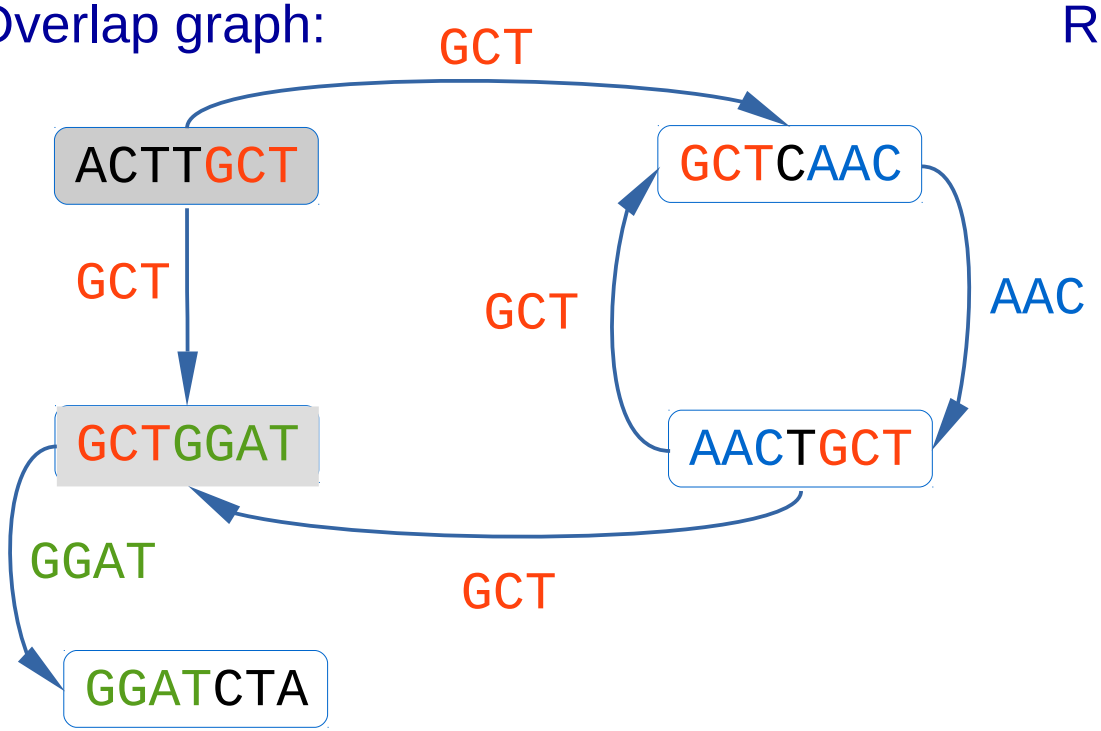
ACTTGCT

Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:
ACTTGCT
AACTGCT
GCTCAAC
GCTGGAT
GGATCTA

Overlap graph:



Reconstructed sequence:

ACTTGCT
ACTTGCTGGAT

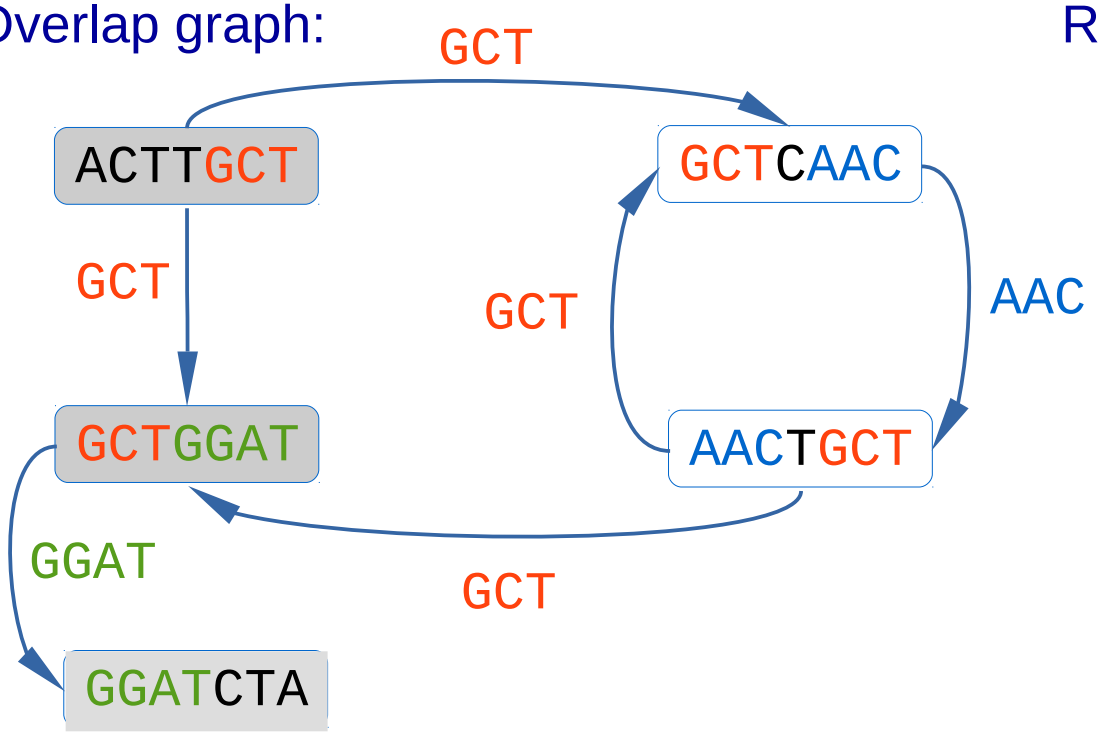
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
      AACTGCT
   GCTCAAC
         GCTGGAT
             GGATCTA
```

Overlap graph:



Reconstructed sequence:

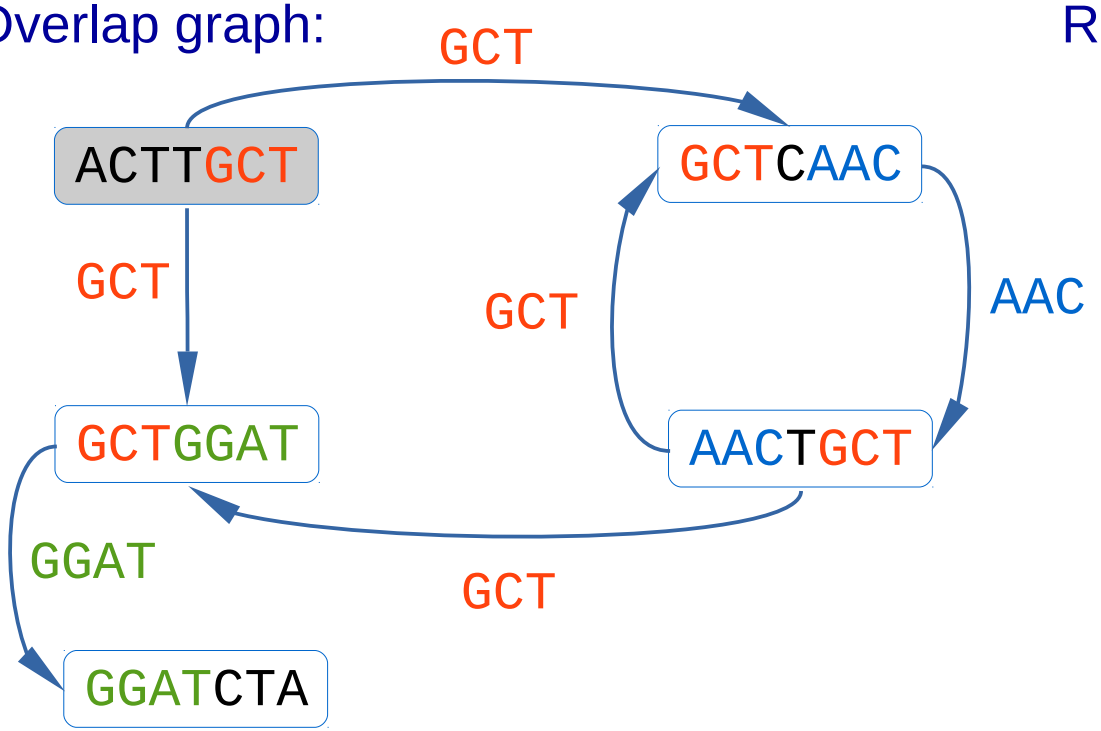
ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA

Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:
ACTTGCT
AACTGCT
GCTCAAC
GCTGGAT
GGATCTA

Overlap graph:



Reconstructed sequence:

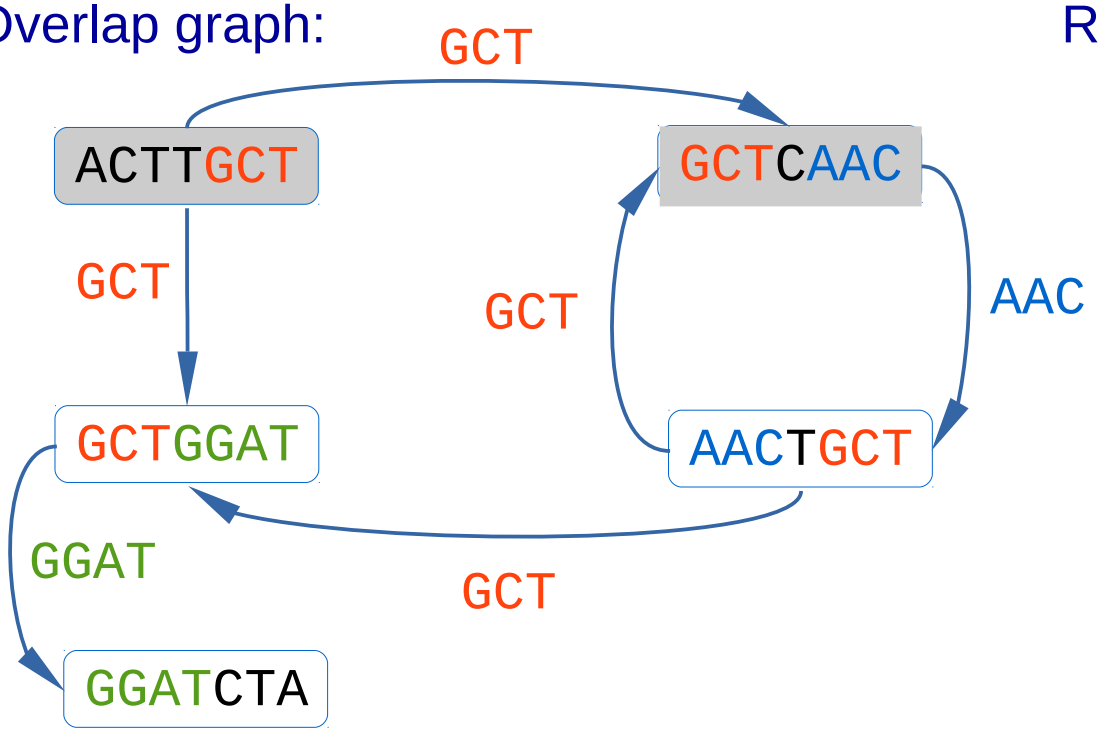
ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT

Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:
ACTTGCT
AACTGCT
GCTCAAC
GCTGGAT
GGATCTA

Overlap graph:



Reconstructed sequence:

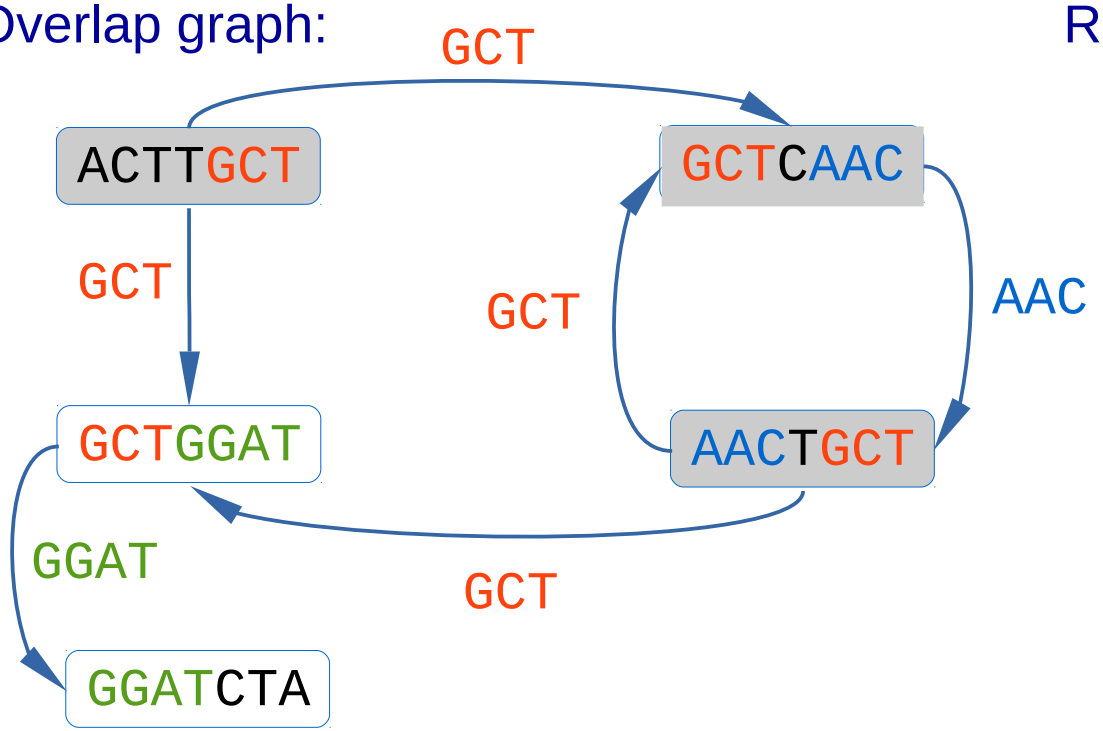
ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT
ACTTGCTCAAC

Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:
ACTTGCT
AACTGCT
GCTCAAC
GCTGGAT
GGATCTA

Overlap graph:



Reconstructed sequence:

ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT
ACTTGCTCAAC
ACTTGCTCAACTGCT

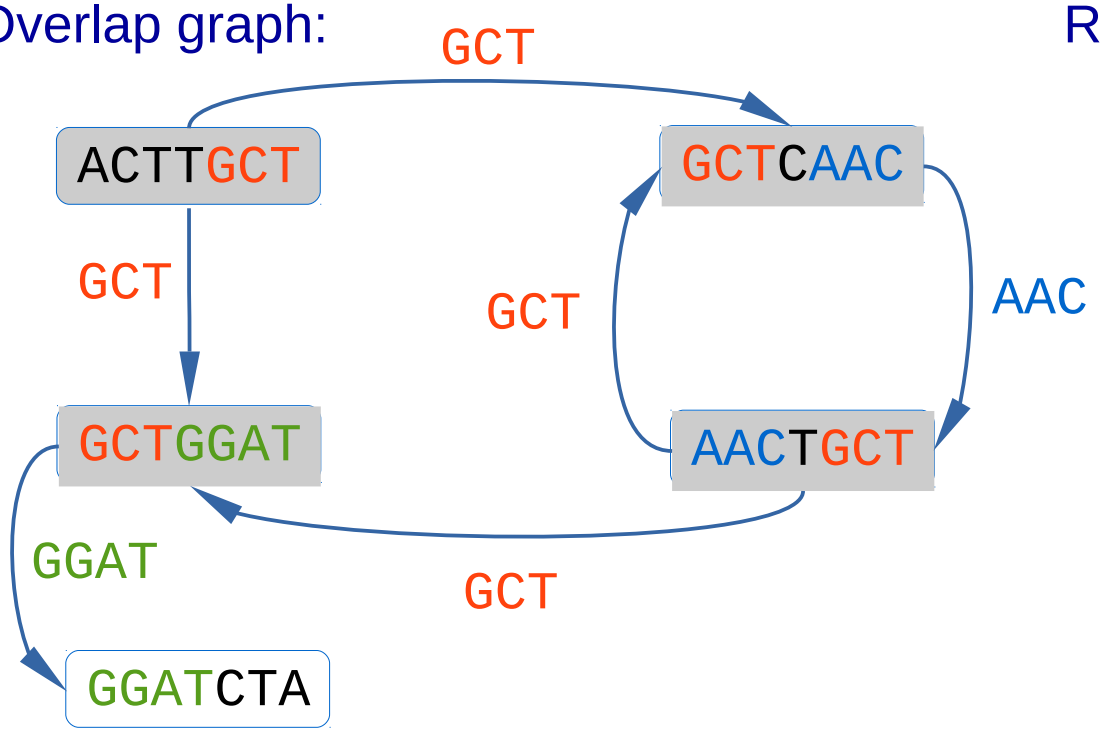
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
      AACTGCT
   GCTCAAC
         GCTGGAT
             GGATCTA
```

Overlap graph:



Reconstructed sequence:

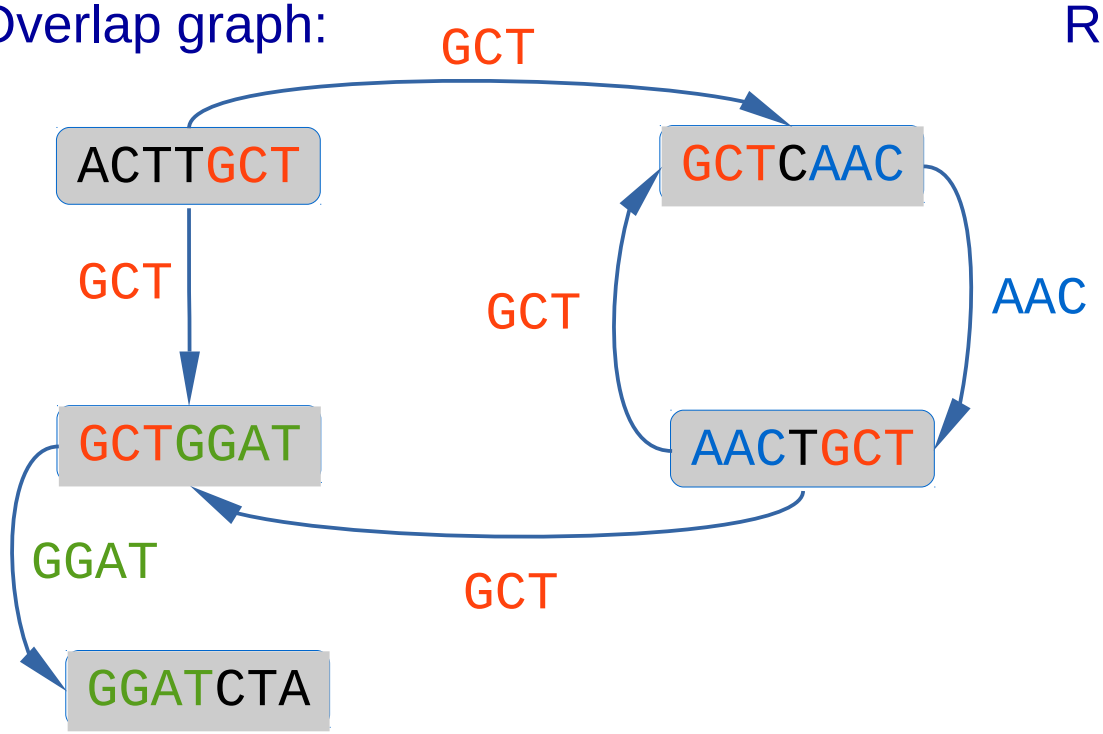
ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT
ACTTGCTCAAC
ACTTGCTCAACTGCT
ACTTGCTCAACTGCTGGAT

Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:
ACTTGCT
AACTGCT
GCTCAAC
GCTGGAT
GGATCTA

Overlap graph:



Reconstructed sequence:

ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT
ACTTGCTCAAC
ACTTGCTCAACTGCT
ACTTGCTCAACTGCTGGAT
ACTTGCTCAACTGCTGGATCTA

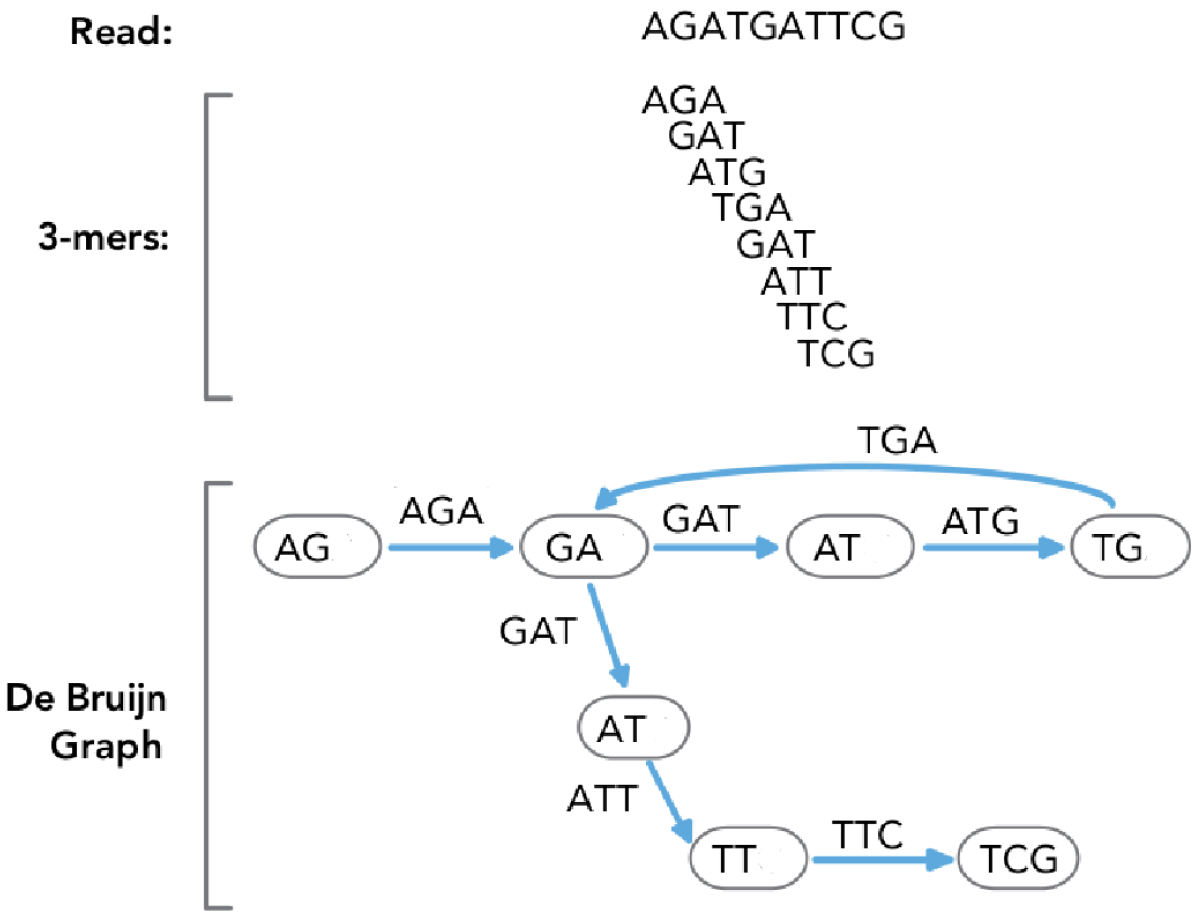
Overlap graphs: problems

- 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
- With millions of short NGS reads, this method becomes computationally unfeasible

De Brujin graphs

- To build a de Brujin graph, each read is decomposed into series of *k-mers*
  - In real applications, the minimum *k* is 19 and the maximum is limited by computational resources
  - Optimal value of *k* depends on read length and error rate
- 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
- The original sequence can be reconstructed by finding an Eulerian path (visit each edge exactly once)

De Brujin graphs: example



De Brujin graphs: advantages

- Compact representation of repeats
  - Duplicate *k-mers* are represented with a single node
  - Longer repeats form a single series of adjacent nodes
- Graph building time is linear to the number of reads
- There exists an efficient algorithm to find an Eulerian path

➔ For these reason, most modern NGS assemblers use de Brujin graphs (either explicitly or implicitly)

Outlook: 3rd generation sequencing

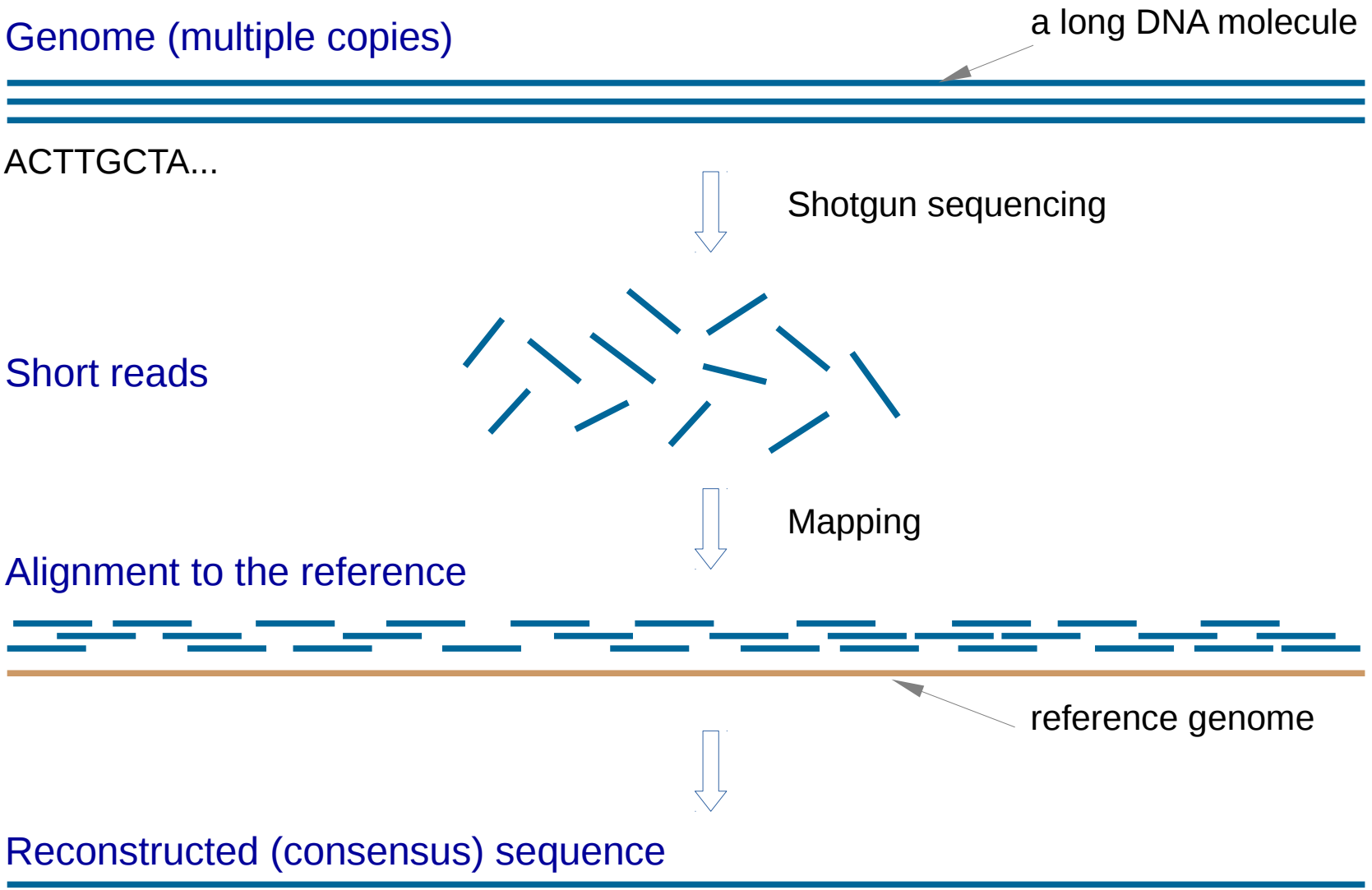
- Further advancement after NGS
  - Pacific Biosciences (PacBio SMRT)
  - Oxford Nanopore (MinION)
- Hallmarks:
  - Long reads (5 – 30 kb)
  - High error rates (15 – 30 %)
- Change in assembly methods
  - Overlap graphs, hybrid approaches

Image: wired.com

Today's agenda

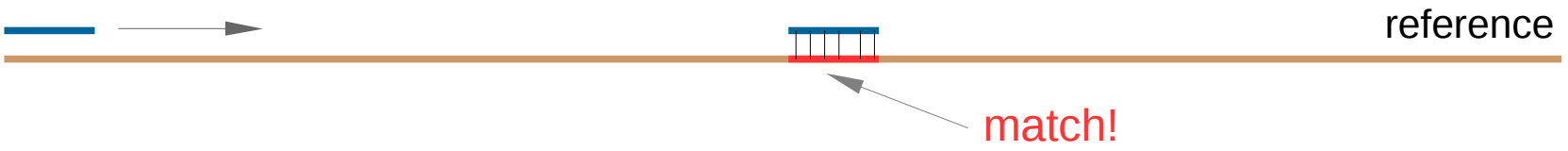
- Search methods for biological sequences
  - Public sequence databases
  - BLAST algorithm
  - Alternative search algorithms
- Genome assembly
  - *De novo* assembly
    - Overlap graphs
    - De Brujin graphs
  - By-reference assembly
    - Hash indexes
    - Burrows-Wheeler transform

By-reference assembly



Sliding window approach

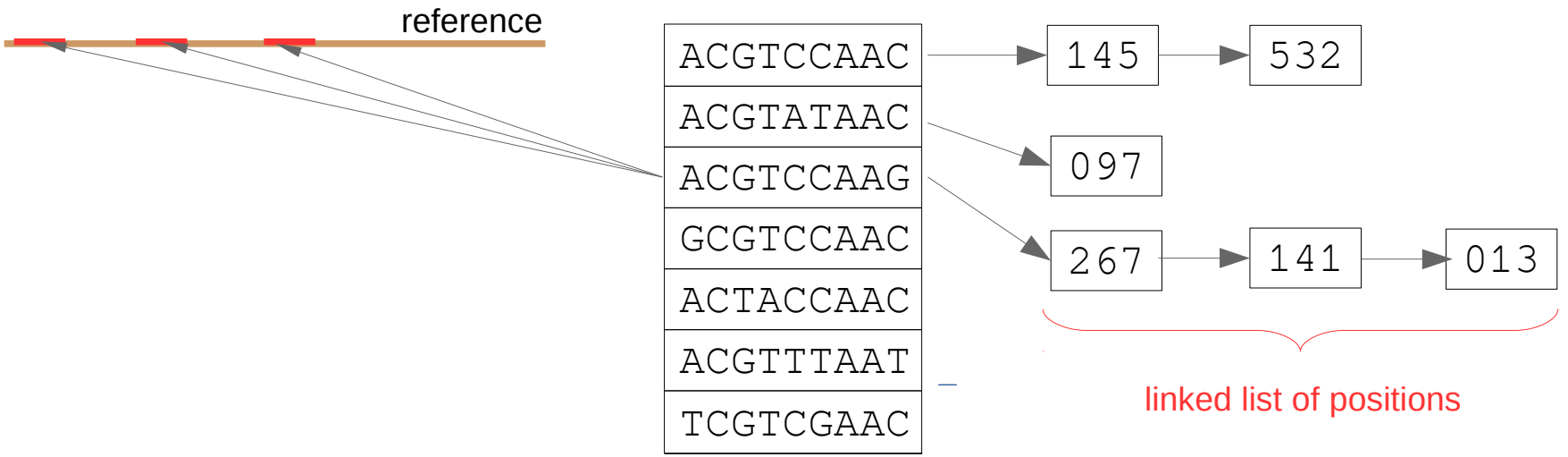
- Slide each read along the reference genome and mark all positions where there is a match
  - if gaps are allowed, one has to resort to the classical DP algorithms like Smith-Waterman



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

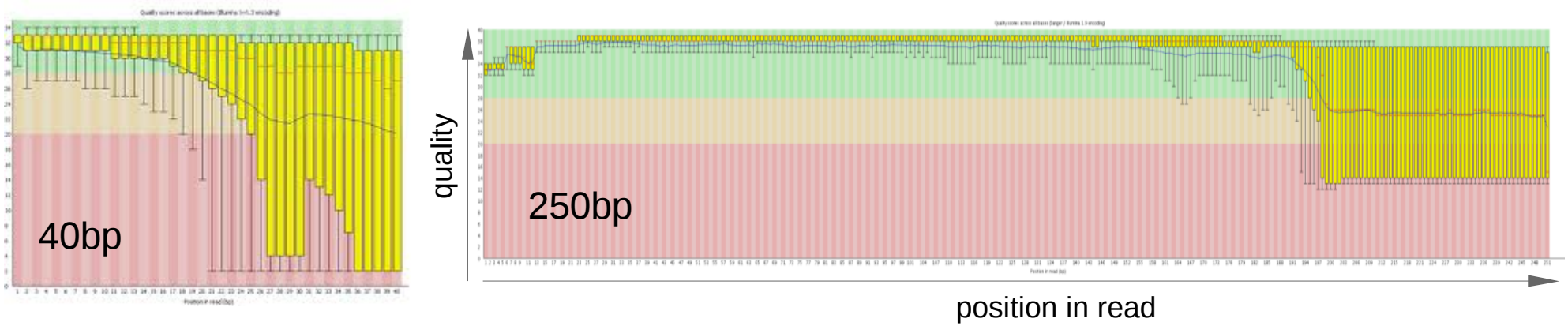
Hashing: build genome index

- Use hash table to store positions of all *k-mers* in a genome*:*
  - *k* << read length
  - *k := 9* → 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



- 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 Needleman-Wunsch algorithm

Hashing: optimizations & variants

- Inexact matches with spaced seeds
  - Binary mask defines positions where mismatches are allowed:

111001 ATCGGT ↔ ATCACT

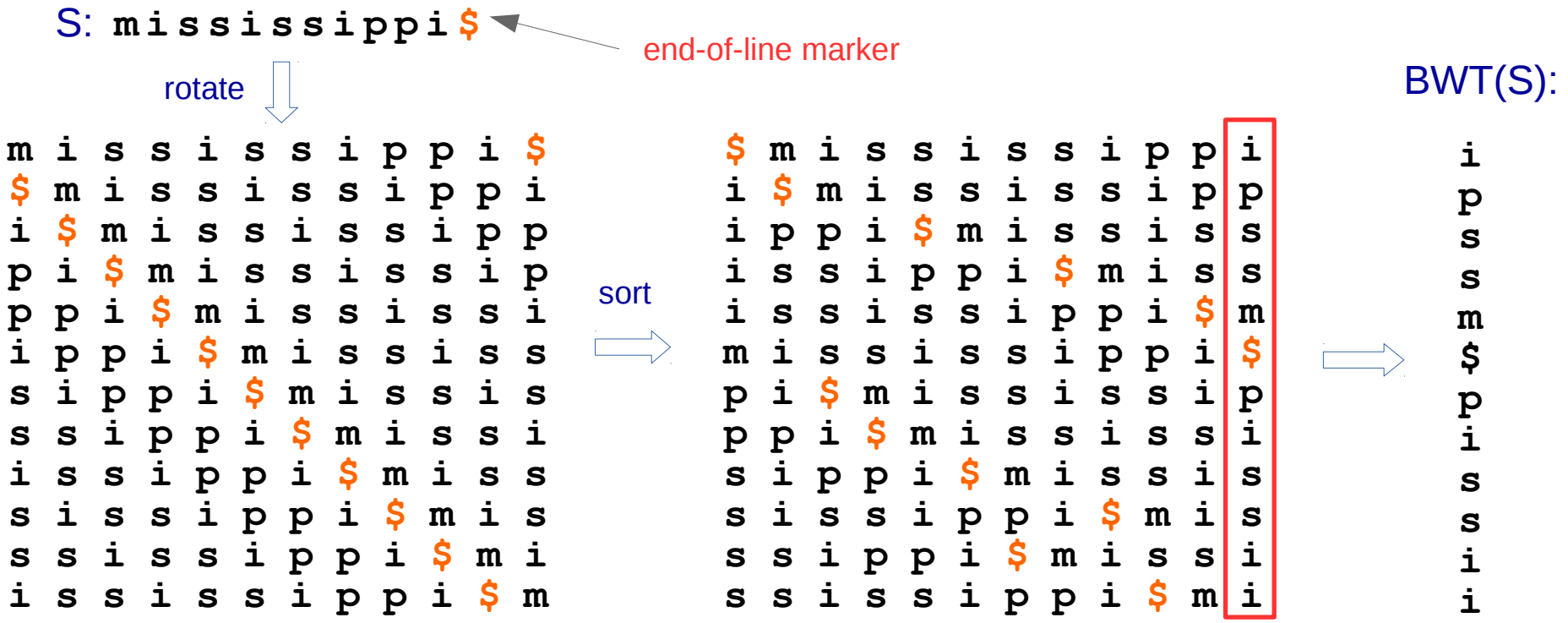
- Multiple *k-mers* per read
  - Require at least *n* seed matches for a mapping location to be considered
- Inverted approach: Build hash table from the reads and search for *k-mers* present in the reference

By-reference assembly: BWT

- Most recent mapping tools rely on Burrows-Wheeler Transform or BWT (Burrows and Wheeler, 1994)
- 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 → BWT(*S*)



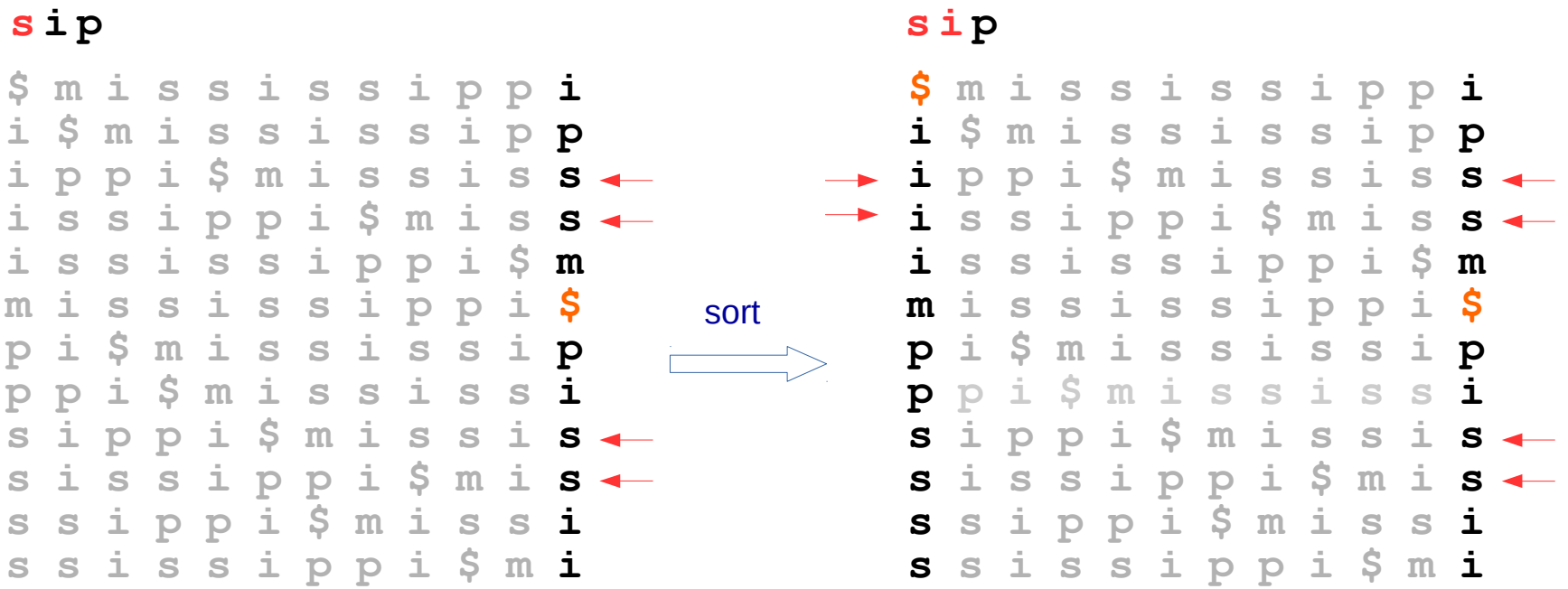
BWT properties

- 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

Pattern search with BWT

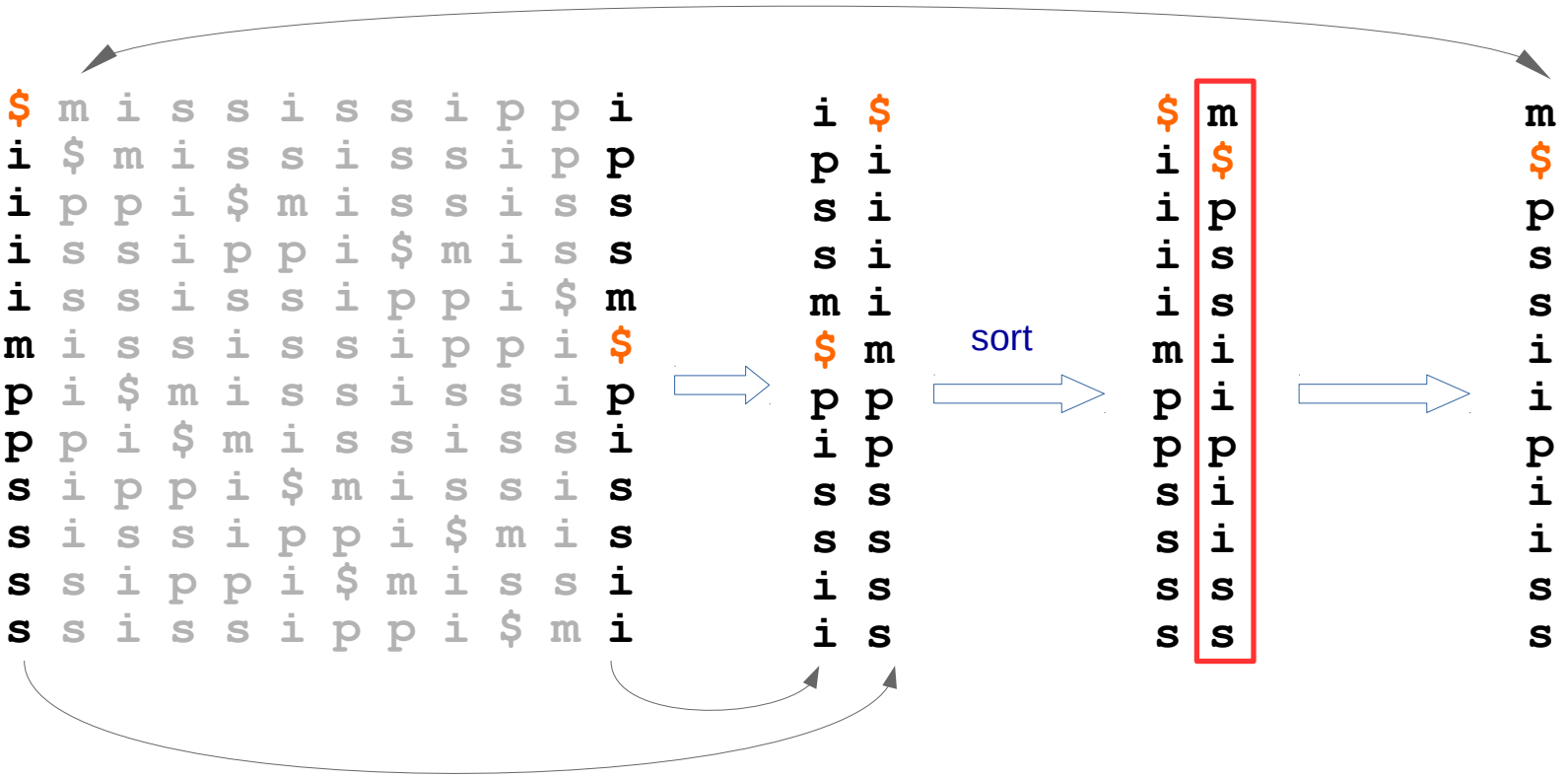
- Trick: first column of the matrix can be obtained by simply sorting the last one (i.e. BWT)

Find: **s i p**



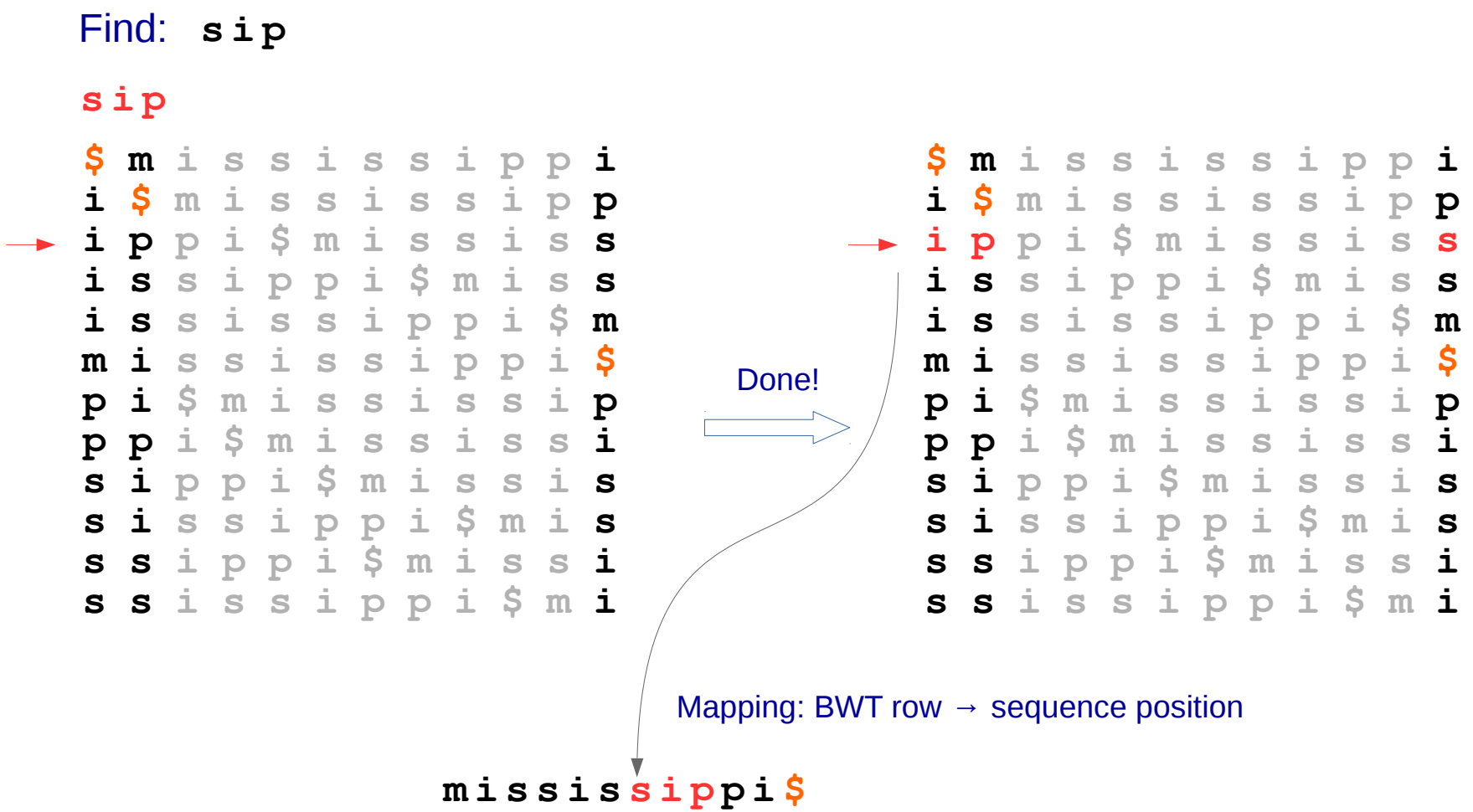
Pattern search with BWT (2)

- Use similar trick to get the 2nd column from the 1st and the last one



Pattern search with BWT (3)

- Proceed with the search



BWT indexes

- The algorithm presented above is not efficient yet → need an index to optimize BWT matrix traversal
- FM-index (Ferragina and Manzini, 2000) is the most well-known index based on BWT
- In addition to BWT itself, the FM-index stores:
  - Last-to-first mapping: allows to avoid the sorting step
  - Mapping between suffix indices and positions in original sequence

BWT- vs. Hash-based tools

Table 1. Benchmark of short read alignment tools

	Software	Reads aligned (%)	Time (paired, s)	Time (single, s)	Memory usage (GB)
BWT	SOAP2	93.6	828	478	5.4
	SOAP	93.8	19 234	14 328	14.7
	MAQ	93.2	22 506	19 847	1.2
BWT	Bowtie	91.7	–	405	2.3

Source: Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang "SOAP2: an improved ultrafast tool for short read alignment" Bioinformatics (2009) 25 (15): 1966-1967 doi:10.1093/bioinformatics/btp336

Further info

- Older slides from SS2014
  - *by Solon Pissis & Tomas Flouri*
  - much more details, “Skript”-like → better for learning
  - http://sco.h-its.org/exelixis/web/teaching/lectures2014/lecture4.pdf
- Online course series: Bioinformatics I - VI
  - *by Phillip E. C. Compeau, Nikolay Vyahhi, Pavel Pevzner*
  - nice explanation of de Brujin graphs, BWT and genome assembly
  - https://www.coursera.org/specializations/bioinformatics

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

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

Aleksandr Morgulis, George Coulouris, Yan Raytselis, Thomas L. Madden, Richa Agarwala, and Alejandro A. Schäffer (2008). Database indexing for production MegaBLAST searches. *Bioinformatics* 24 (16): 1757-1764 first published online June 21, 2008 doi:10.1093/bioinformatics/btn322

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]

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

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

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

Paolo Ferragina, Giovanni Manzini (2000). Opportunistic data structures with applications. *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*

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