Introduction to Bioinformatics for Computer Scientists

Lecture 4

BLAST & Genome assembly

Alexey Kozlov

Staff Scientist

alexey.kozlov@h-its.org

Exelixis Lab

November 16, 2022

Karlsruhe Institute of Technology

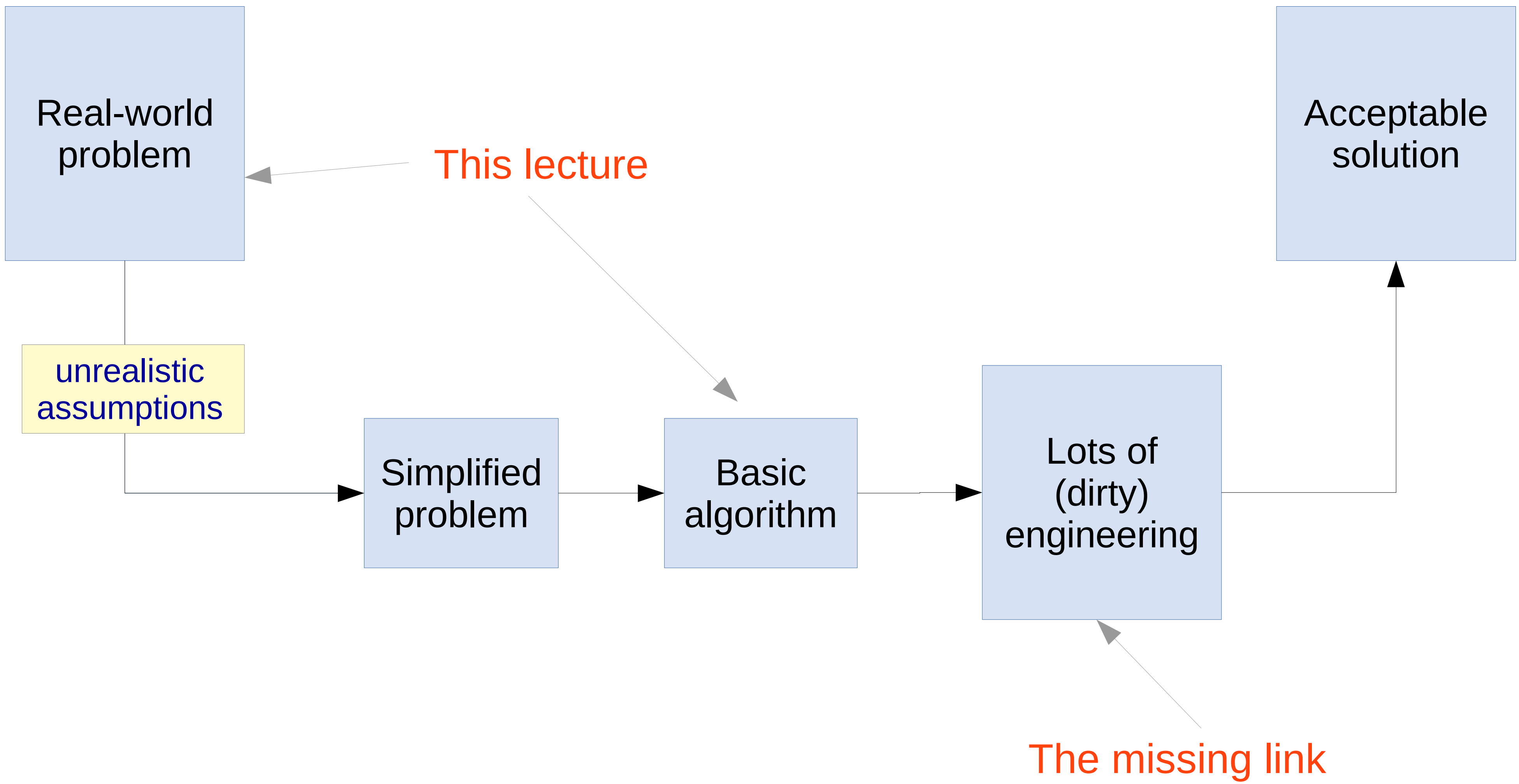
Heidelberg Institute for Theoretical Studies

HITS

Today's agenda

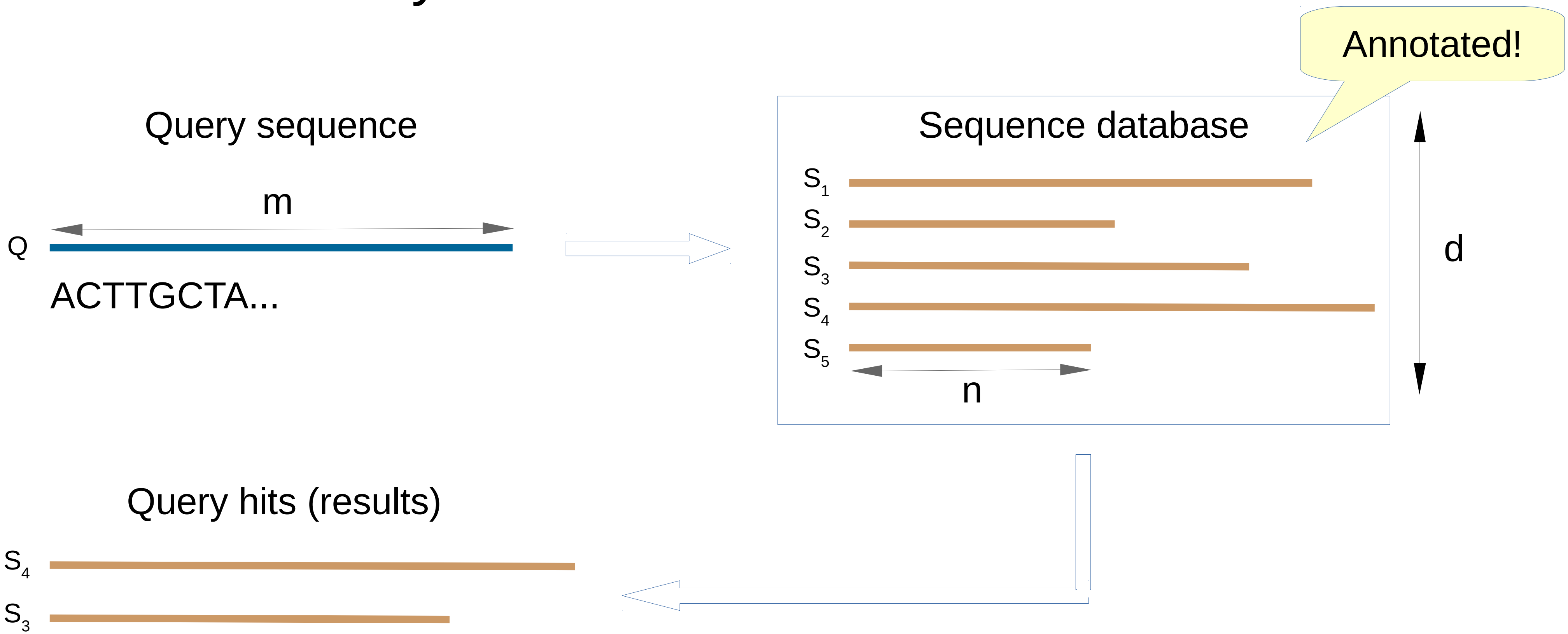
- Search methods for biological sequences
  - Public sequence databases
  - BLAST algorithm
  - Alternative search algorithms
- Genome assembly
  - *De novo* assembly
  - By-reference assembly

How Bioinformatics works



Searching for similar sequences

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

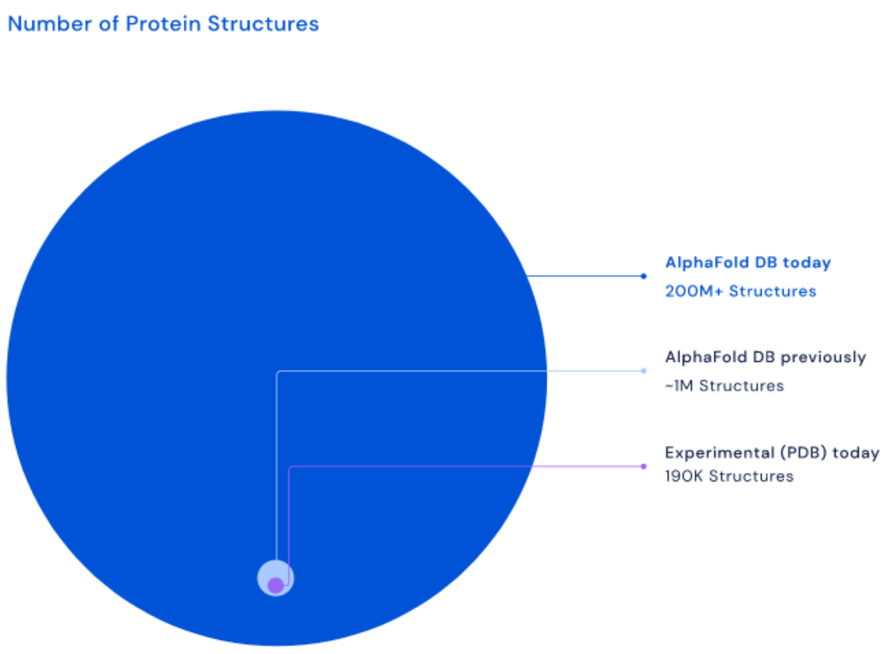


Sequence databases: proteins

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

Sequence databases: proteins

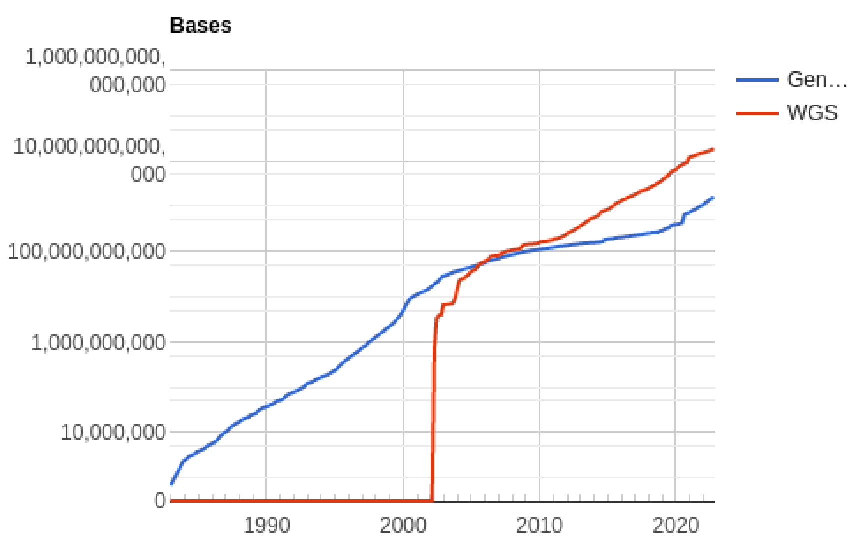
- AlphaFold DB
  - 200M protein structures → high-quality prediction
  - https://alphafold.ebi.ac.uk/

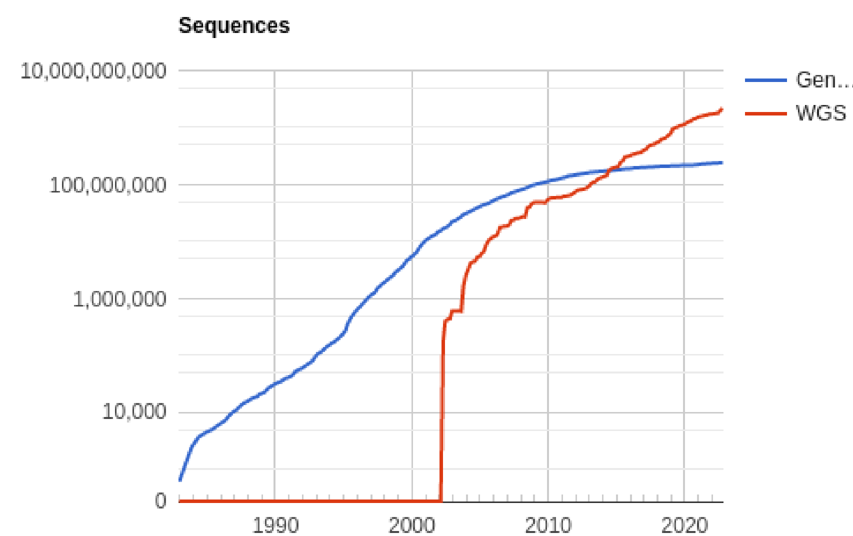


Q8W3K0: A potential plant disease resistance protein. Mean pLDDT 82.24.

Sequence databases: GenBank

- NCBI GenBank/INSDC →”all” DNA+proteins
  - > 240M “regular” sequences as of Oct. 2022
  - ~ 2.1B whole-genome-shotgun (WGS)
  - “primary” database → sequences submitted and annotated by the respective authors (biologists)





Sequence annotation in GenBank

Bacteroides galacturonicus 16S ribosomal RNA gene, partial sequence

GenBank: DQ497994.1

FASTA Graphics

Go to:

```
LOCUS       DQ497994                1431 bp    DNA     linear   BCT 01-JUN-2006
DEFINITION  Bacteroides galacturonicus 16S ribosomal RNA gene, partial
            sequence.
ACCESSION   DQ497994
VERSION     DQ497994.1  GI:95062834
KEYWORDS    .
SOURCE      Bacteroides galacturonicus
  ORGANISM  Bacteroides galacturonicus
            Bacteria; Bacteroidetes; Bacteroidia; Bacteroidales;
            Bacteroidaceae; Bacteroides.
REFERENCE   1  (bases 1 to 1431)
  AUTHORS   Song,Y., Liu,C. and Finegold,S.M.
  TITLE     Reclassification of Bacteroides galacturonicus
  JOURNAL   Unpublished
REFERENCE   2  (bases 1 to 1431)
  AUTHORS   Song,Y., Liu,C. and Finegold,S.M.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-APR-2006) VA Medical Center West Los Angeles, 11301
            Wilshire Blvd. Bldg 304 Rm E3-237, Los Angeles, CA 90504, USA
FEATURES             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

accession number
(~ sequence ID)

organism name and taxonomy (=biological classification)

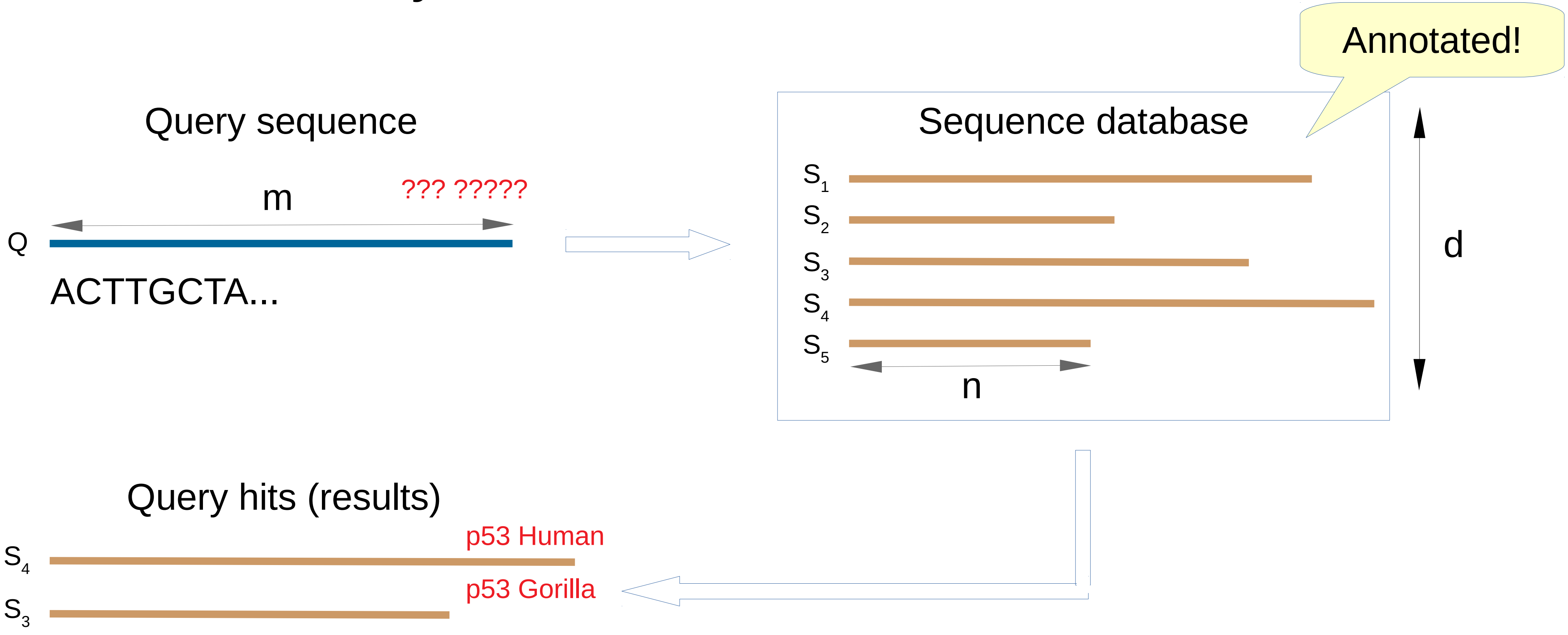
sequence

Sequence databases: barcodes

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

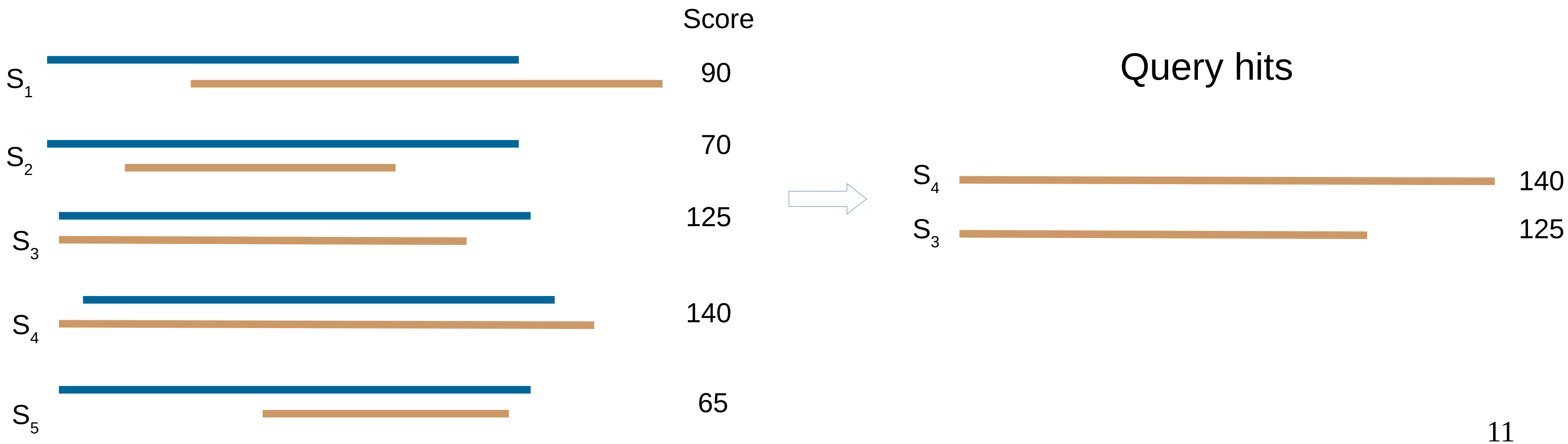
Searching for similar sequences

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



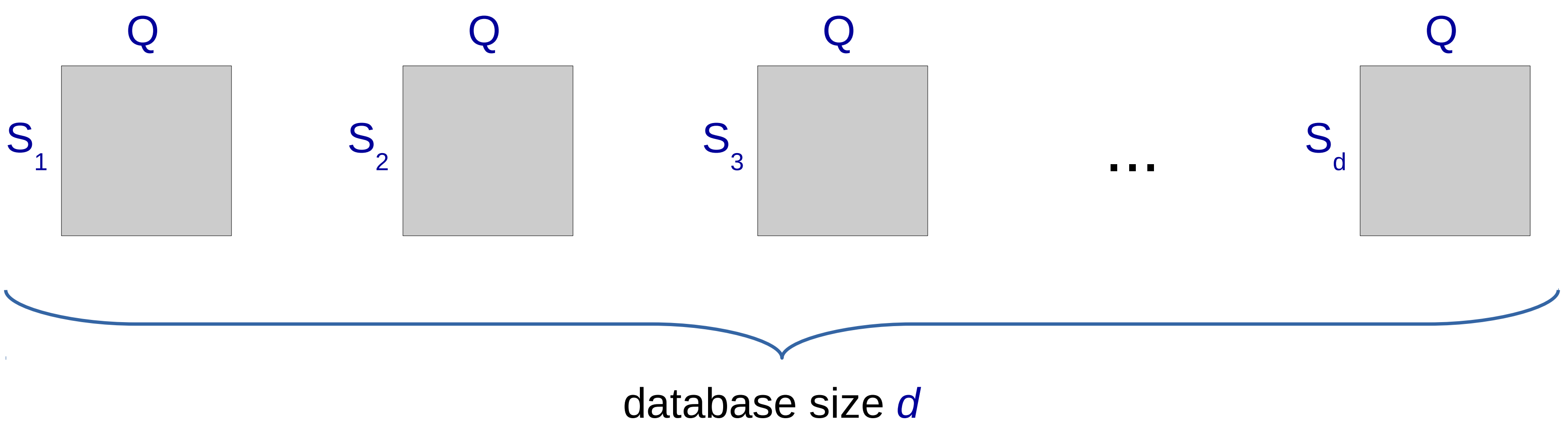
Naïve approach

- Use Smith-Waterman algorithm to build a pair-wise alignment of query sequence *Q* with every sequence in the database $S_1 \ldots S_d$
- 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
  - Two main papers from 1990s have ~180 000 citations[1]
  - cf. RAxML: ~40 000, Human Genome Sequence: ~45 000
- Available as both stand-alone application and web service
  - BLAST on NCBI GenBank database:

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

[1] Altschul et al (1990), Altschul et al (1997)

BLAST: algorithm outline

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

BLAST: seeding

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

```
Query:  ACTTGCTAAC
        ACTTG
         CTTGC
          TTGCT
           TGCTA
             CTAAC
```

L_1

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)

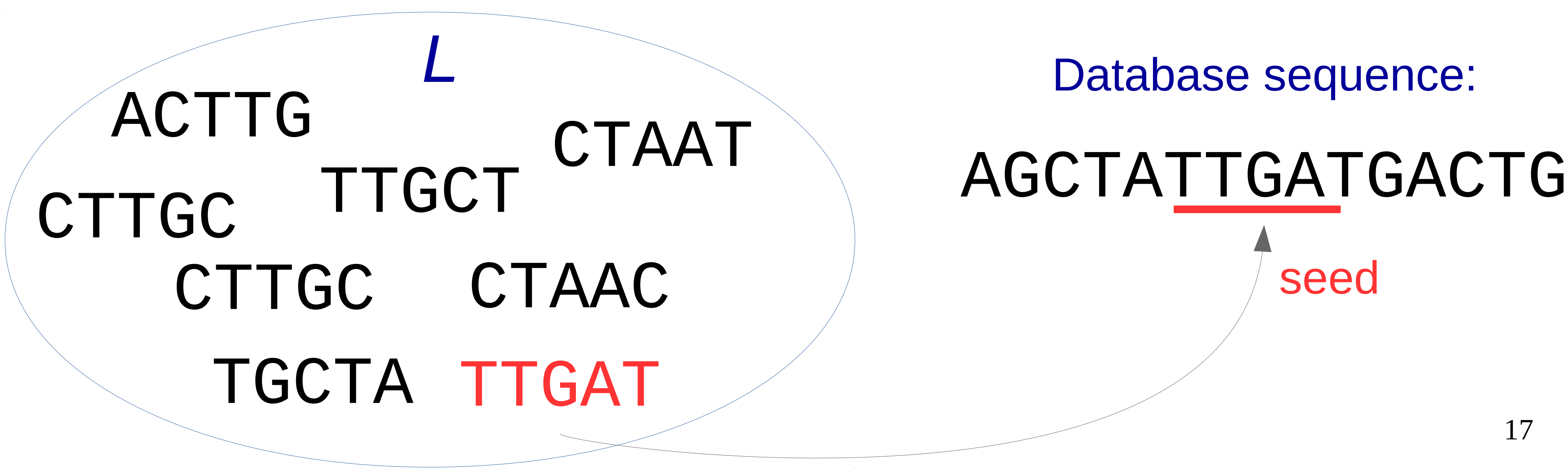
- Combine all subwords' neighborhoods of a single query sequence

$$L_2 = \cup\ N_i$$

- Build a final list by adding the subwords themselves

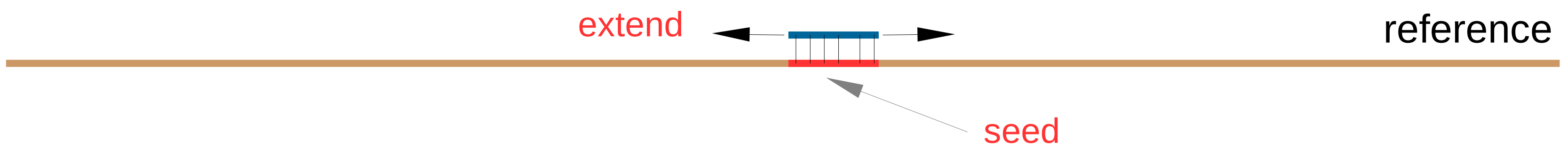
$$L = L_1 \cup\ L_2$$

- 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*

seed

DB sequence:	A	G	C	T	A	T	T	G	A	T	G	A	C	T	G
Query:			C	A	C	T	T	G	C	T	A	A	C		
				-3	-3	+2	+2	+2	-3	+2	-3	+2	+2		

Current score: 6 Max score: 6

High-scoring segment (HSP):

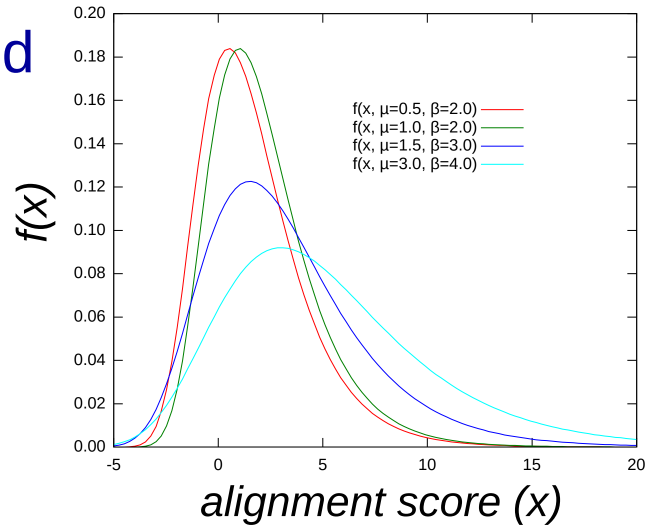
A	G	C	T	A	T	T	G	A	T	G	A	C	T	G
		C	A	C	T	T	G	C	T	A	A	C		

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

BLAST: evaluation

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

EVD prob. density function



- Probability *p* of observing a score *S* equal to or greater than *x by chance* 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*:

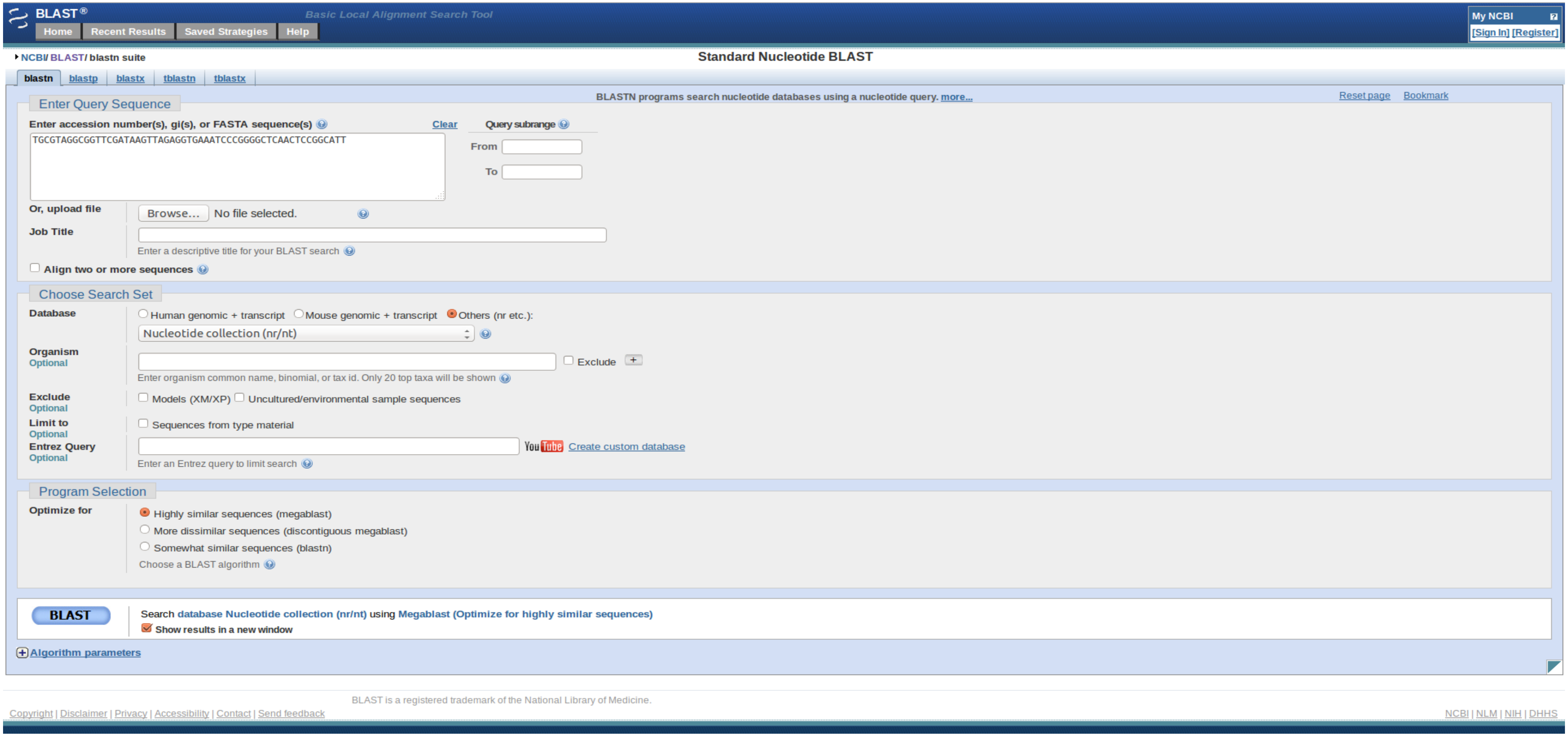
$$E \approx 1 - \mathrm{e}^{-p(S > x)d}$$

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

Beyond BLAST

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

BLAST live demo

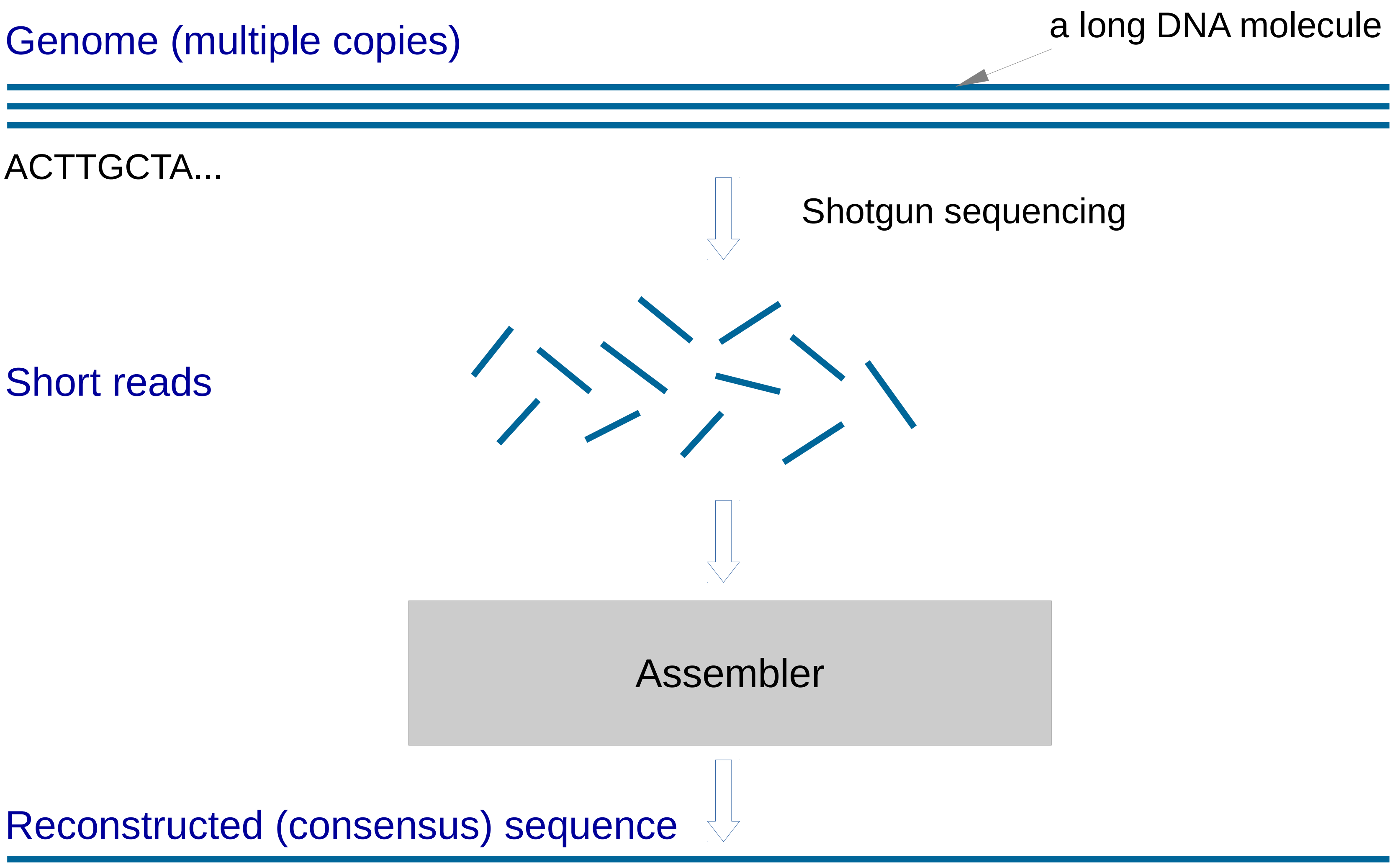


Today's agenda

- 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

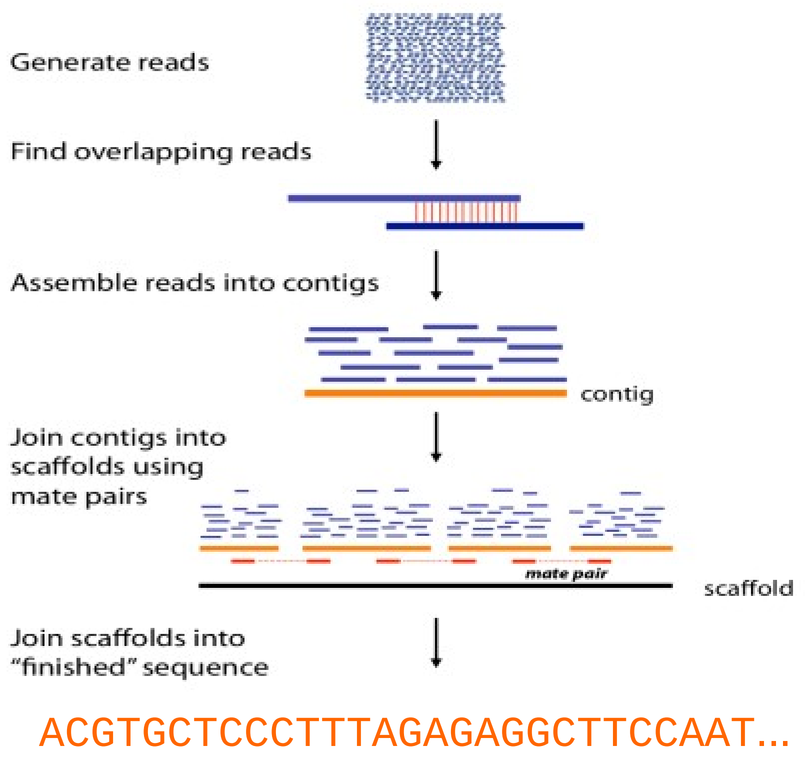
- A very simplified workflow:



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



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

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

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)

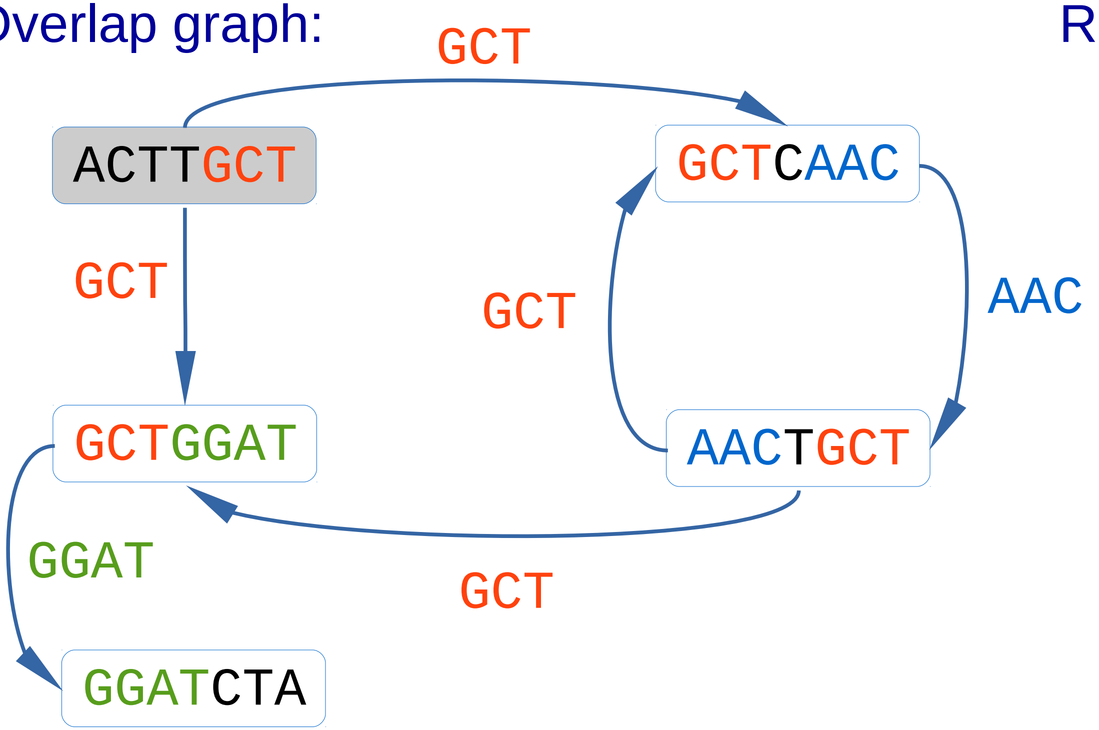
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
       AACTGCT
    GCTCAAC
           GCTGGAT
              GGATCTA
```

Overlap graph:



Reconstructed sequence:

ACTTGCT

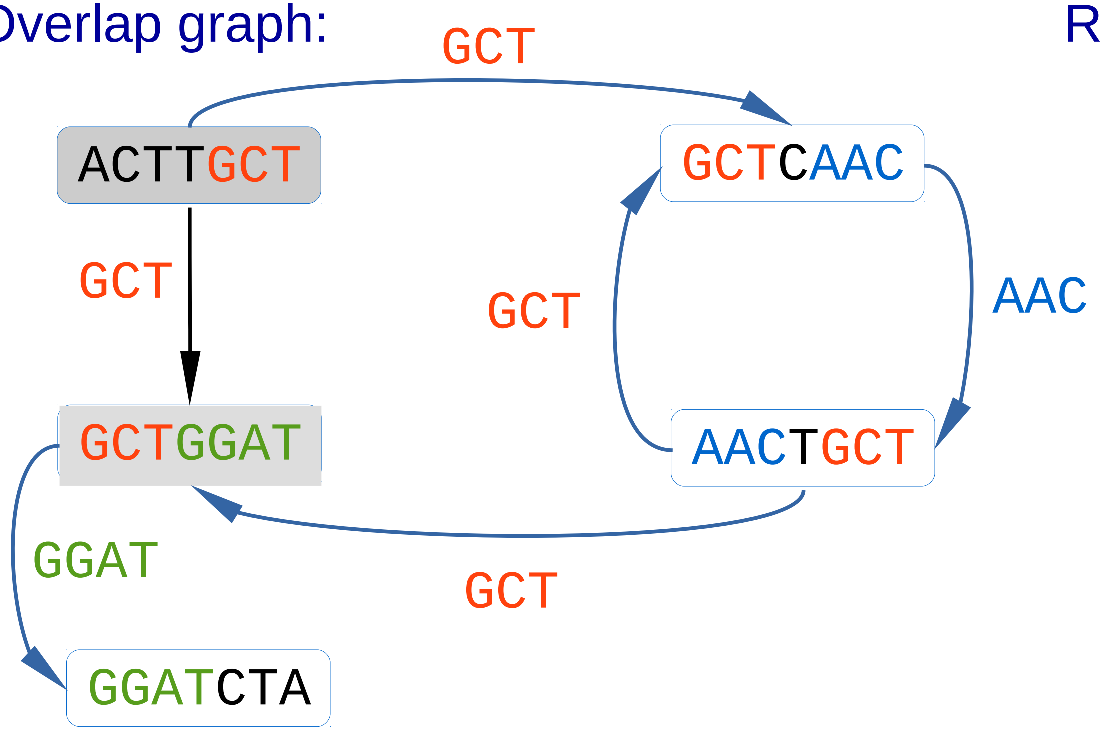
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
          AACTGCT
     GCTCAACT
               GCTGGAT
                   GGATCTA
```

Overlap graph:



Reconstructed sequence:

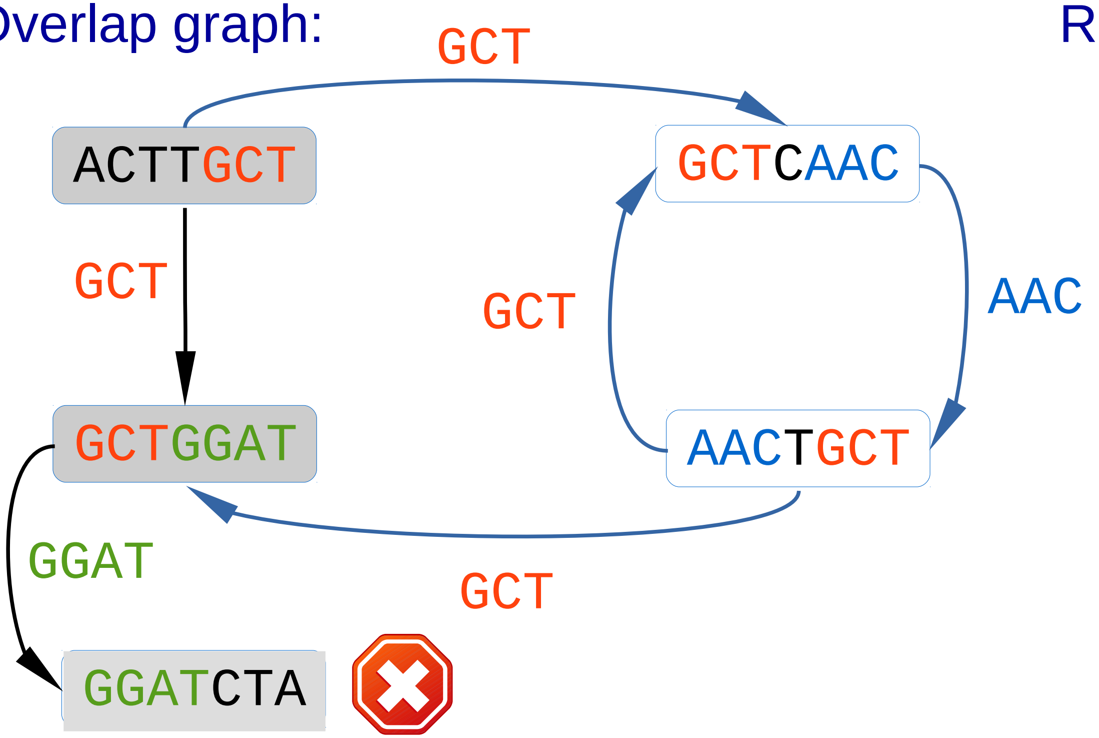
ACTTGCT
ACTTGCTGGAT

Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:
```
ACTTGCT
           AACTGCT
     GCTCAAC
                GCTGGAT
                    GGATCTA
```

Overlap graph:



Reconstructed sequence:

ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA

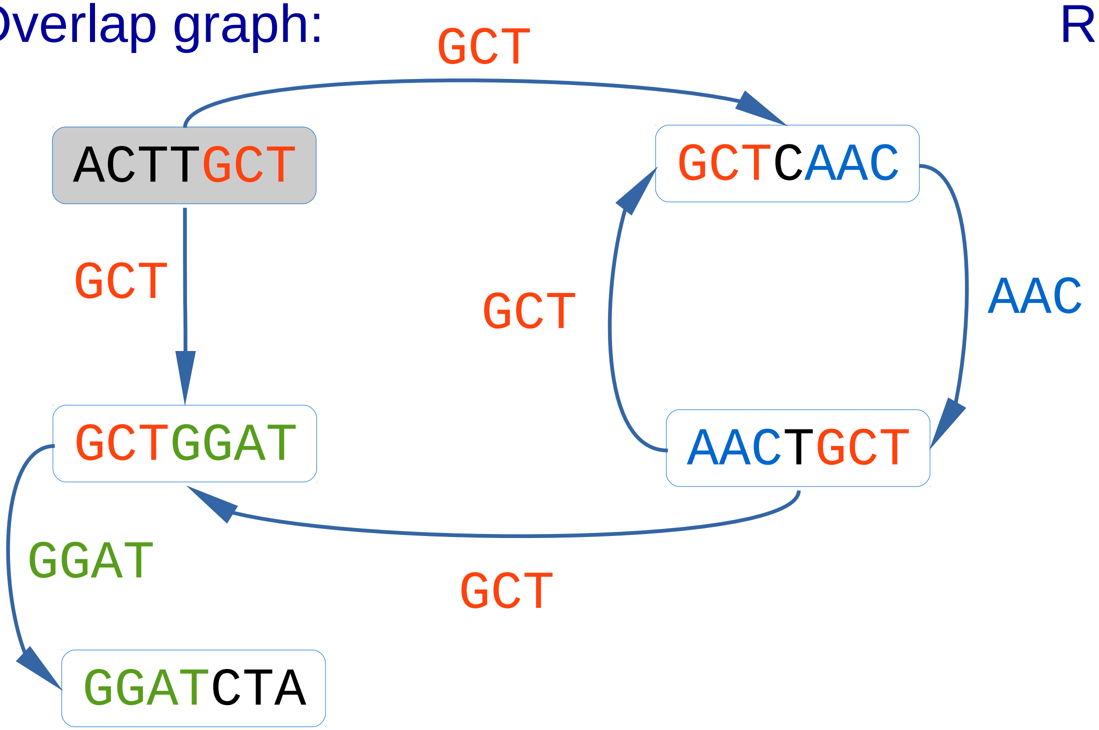
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
          AACTGCT
     GCTCAAC
               GCTGGAT
                   GGATCTA
```

Overlap graph:



Reconstructed sequence:

ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT

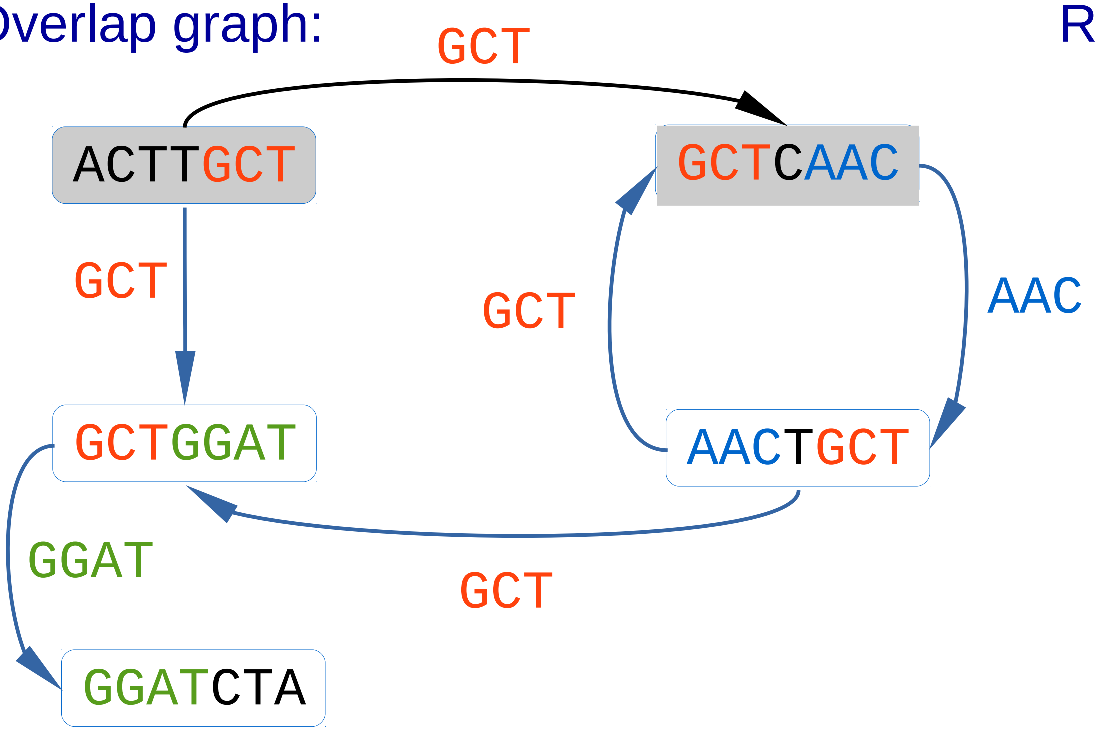
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
          AACTGCT
     GCTCAAC
               GCTGGAT
                   GGATCTA
```

Overlap graph:



Reconstructed sequence:

ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT
ACTTGCTCAAC

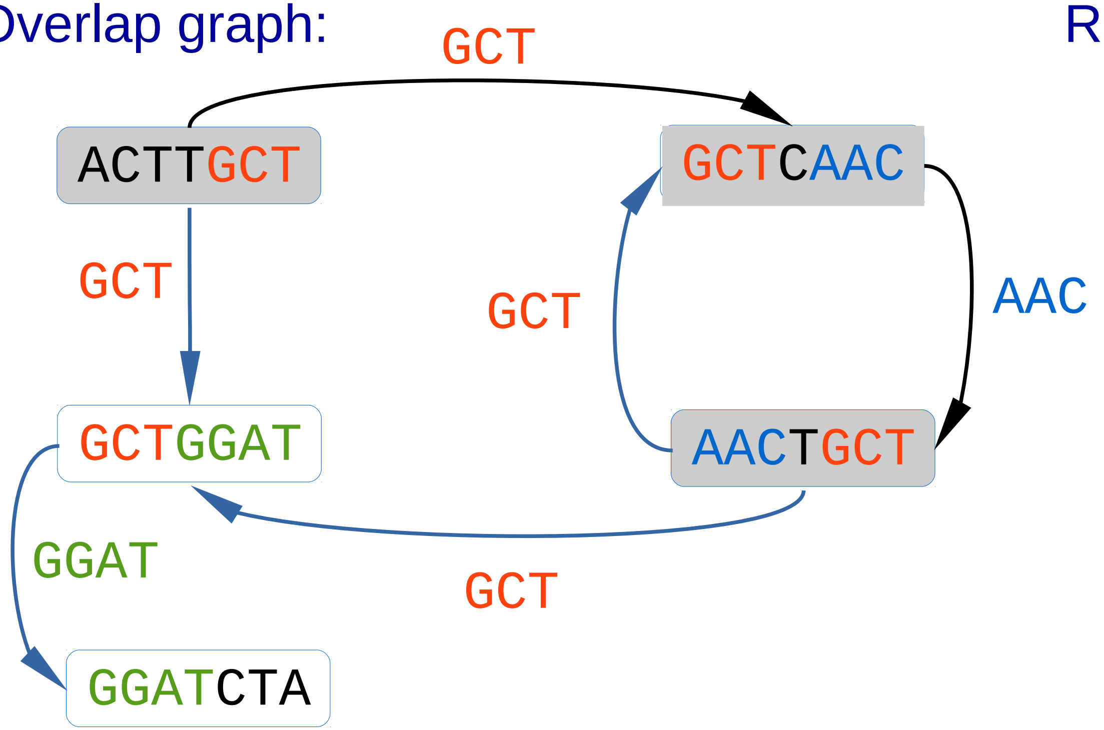
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
       AACTGCT
   GCTCAAC
           GCTGGAT
              GGATCTA
```

Overlap graph:



Reconstructed sequence:

ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT
ACTTGCTCAAC
ACTTGCTCAACTGCT

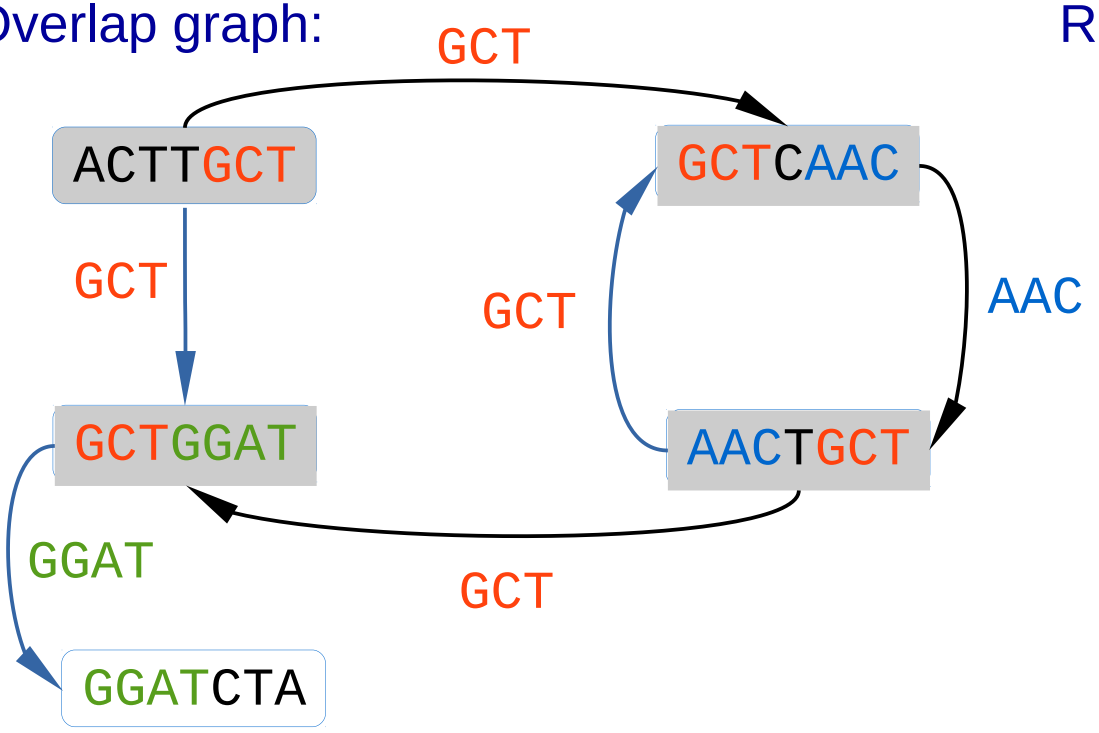
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
       AACTGCT
    GCTCAACT
           GCTGGAT
              GGATCTA
```

Overlap graph:



Reconstructed sequence:

ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT
ACTTGCTCAAC
ACTTGCTCAACTGCT
ACTTGCTCAACTGCTGGAT

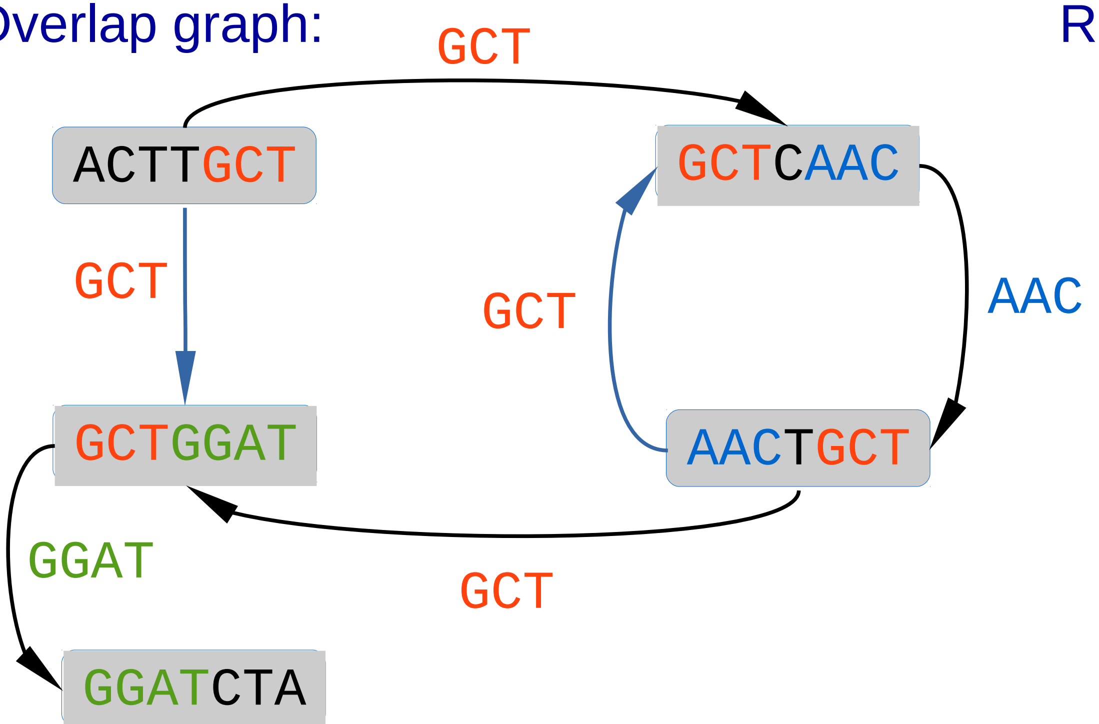
Overlap graphs: example

Genome: ACTTGCTCAACTGCTGGATCTA

Reads:

```
ACTTGCT
          AACTGCT
     GCTCAAC
               GCTGGAT
                   GGATCTA
```

Overlap graph:



Reconstructed sequence:

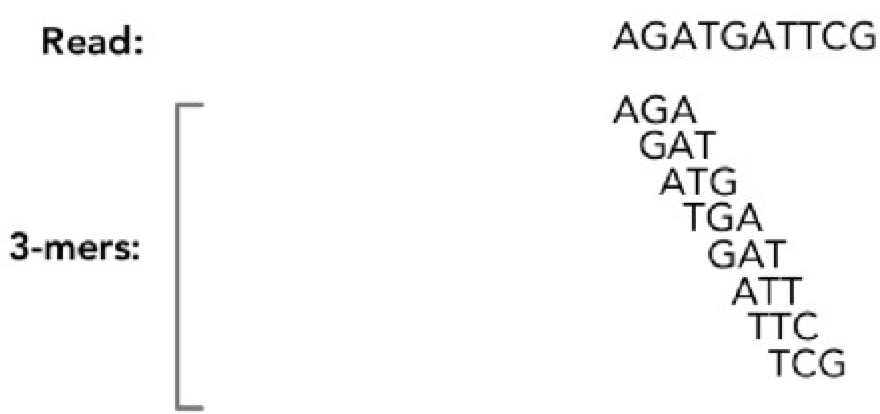
```
ACTTGCT
ACTTGCTGGAT
ACTTGCTGGATCTA
ACTTGCTGGAT
ACTTGCT
ACTTGCTCAAC
ACTTGCTCAACTGCT
ACTTGCTCAACTGCTGGAT
ACTTGCTCAACTGCTGGATCTA
ACTTGCTCAACTGCTGGATCTA
```

Overlap graphs: problems

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

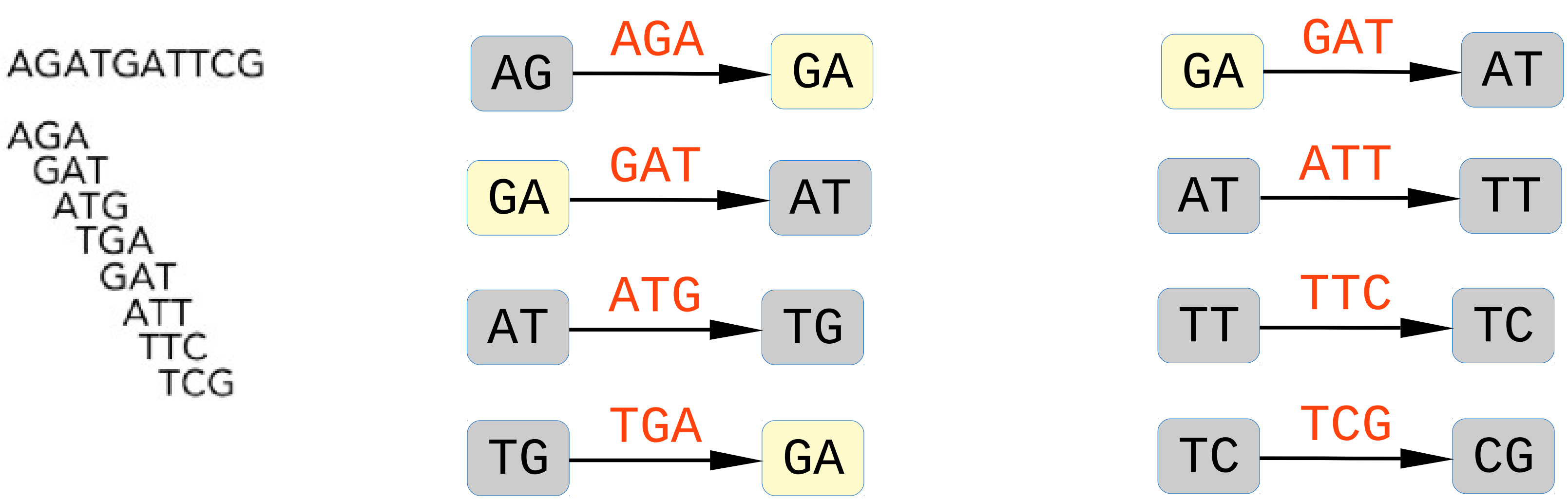
De Bruijn graphs

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



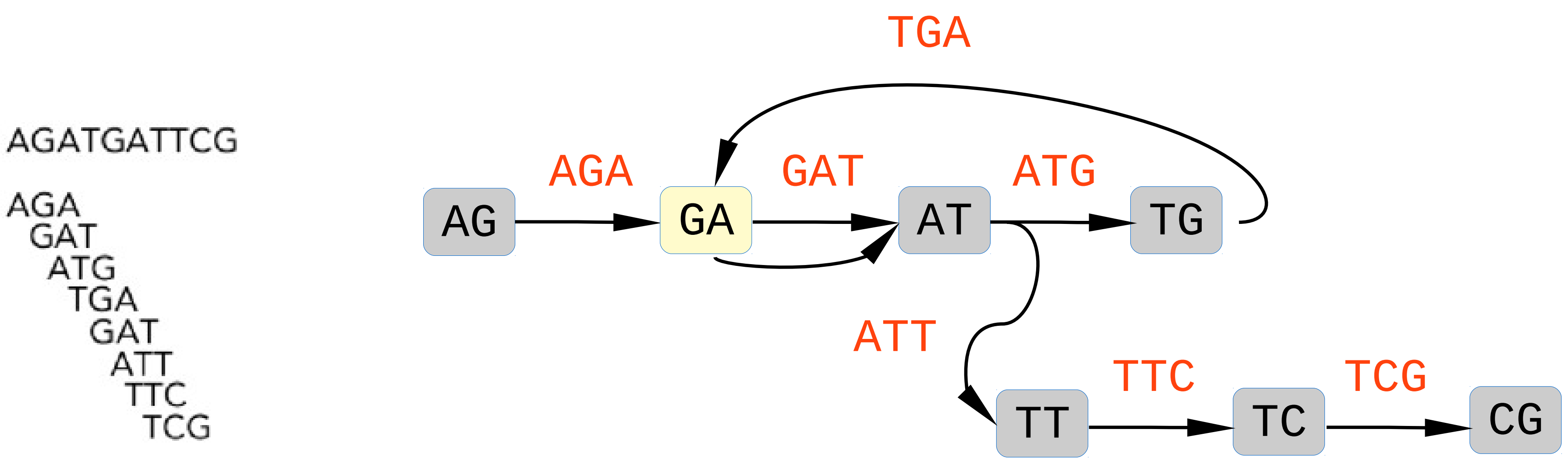
De Bruijn graphs

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



De Bruijn graphs

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



De Bruijn graphs: advantages

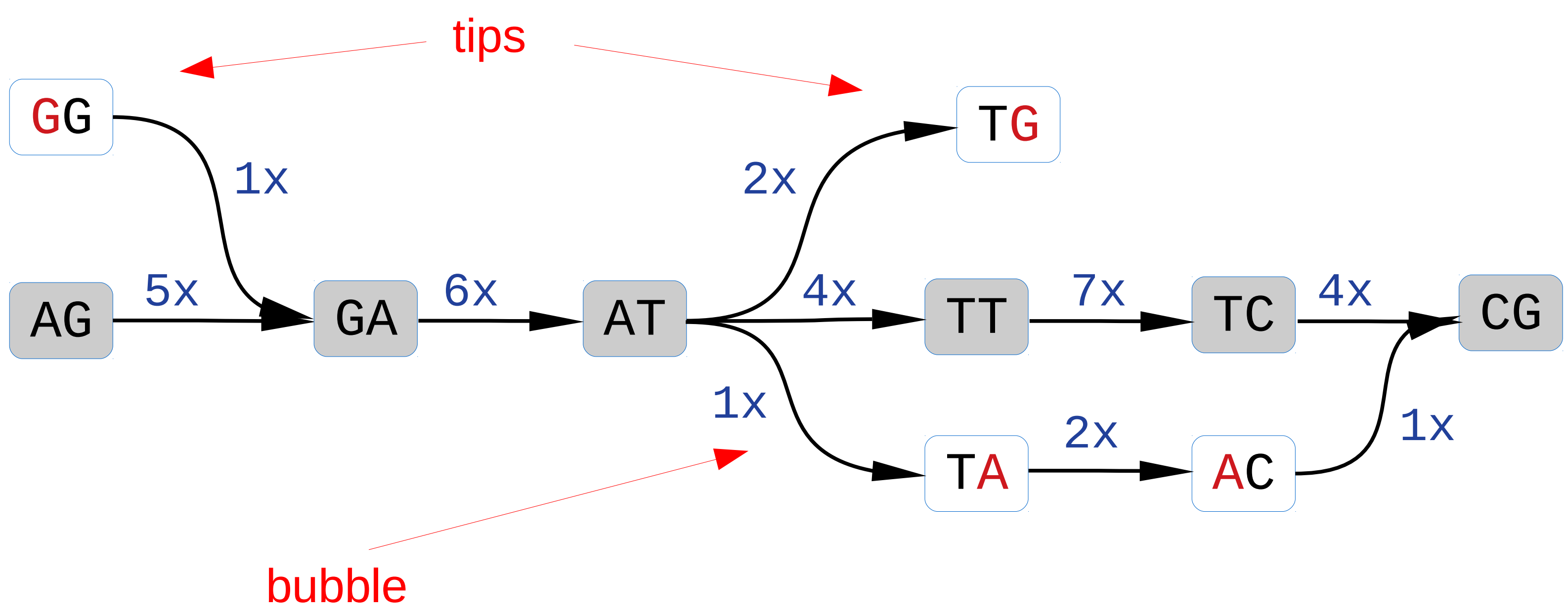
- Compact representation of repeats
  - Duplicate *(k-1)-mers* are represented with a single node
  - Longer repeats form a single series of adjacent nodes
- Building De Bruijn graph has linear complexity
  - Time: $O(N)$, N = total length of all reads
  - Space: $O(min(G,N))$, G = genome size
- There exists an efficient algorithm to find an Eulerian path

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

De Bruijn graphs: problems

- Information loss due to k-mer extraction
  - Repeats are (even) harder to resolve
  - Some paths are not consistent with source reads
- Eulerian path always exists if
  - reads are error-free
  - all genome positions are evenly covered
- There can be multiple alternative paths due to repeats!
- Graph can be disjoint due to lack of (sufficient) coverage of some genome regions
- In practice: assemble as much as possible into (multiple) contigs, and try to join them later on using e.g. mate pairs (cf. Overview)

De Bruijn graphs: error correction



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

Outlook: 3rd generation sequencing

- Further advancement after NGS
  - Pacific Biosciences (PacBio SMRT)
  - Oxford Nanopore (MinION)
- 3GS/PacBio vs NGS/Illumina:
  - Longer reads (>20Kb vs. 0.5Kb)
  - Higher *raw* error rates (5-20% vs. 0.1%)
    but: *down to <0.1% w/consensus*
  - Lower throughput (5-50 Gb vs. 1.8 Tb)
- Change in assembly methods
  - Back to overlap graphs, hybrid approaches
  - Resolve long repeats / bridge contigs

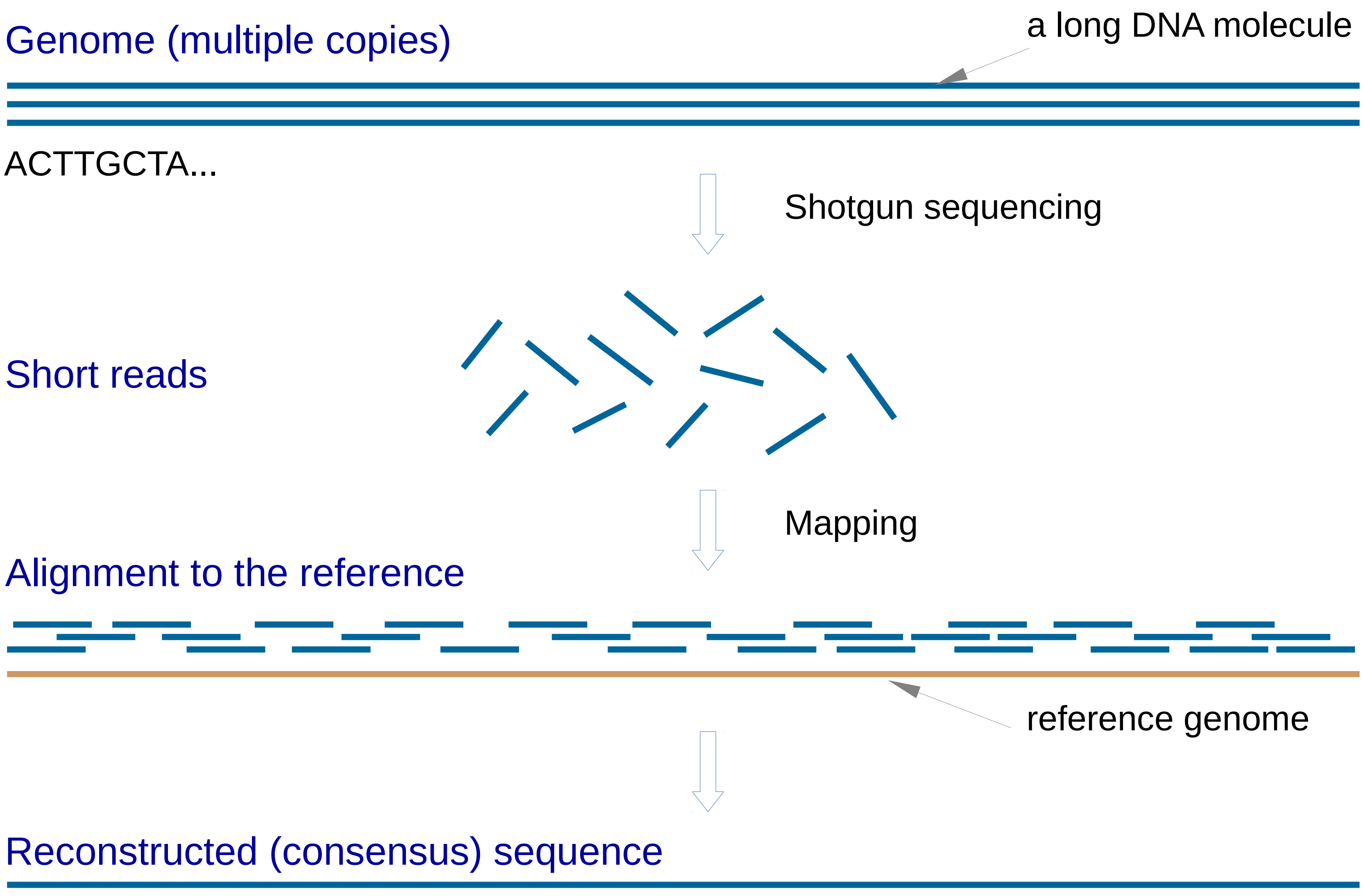
Image: wired.com

Image: genewiz.com

Today's agenda

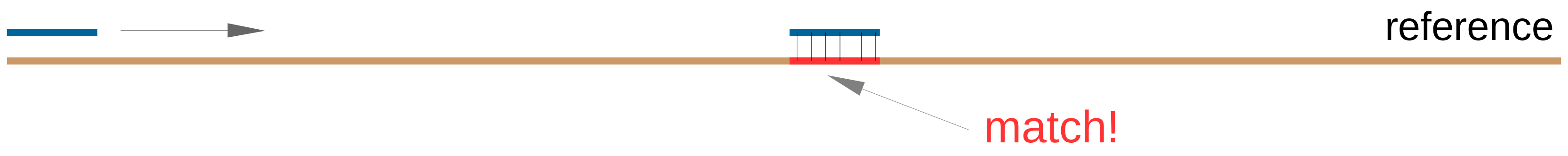
- Search methods for biological sequences
  - Public sequence databases
  - BLAST algorithm
  - Alternative search algorithms
- Genome assembly
  - *De novo* assembly
    - Overlap graphs
    - De Brujin graphs
  - By-reference assembly
    - Hash indexes
    - Burrows-Wheeler transform

By-reference assembly



Sliding window approach

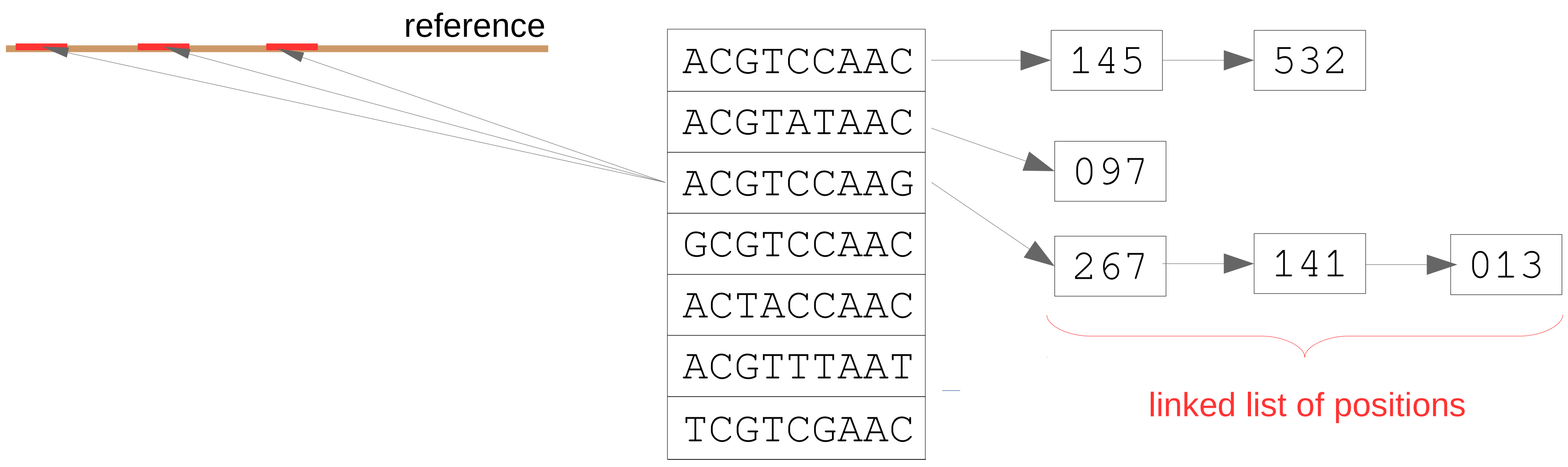
- Slide each read along the reference genome and mark all positions where there is a match
  - if gaps are allowed, one has to resort to the classical DP algorithms like Smith-Waterman



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

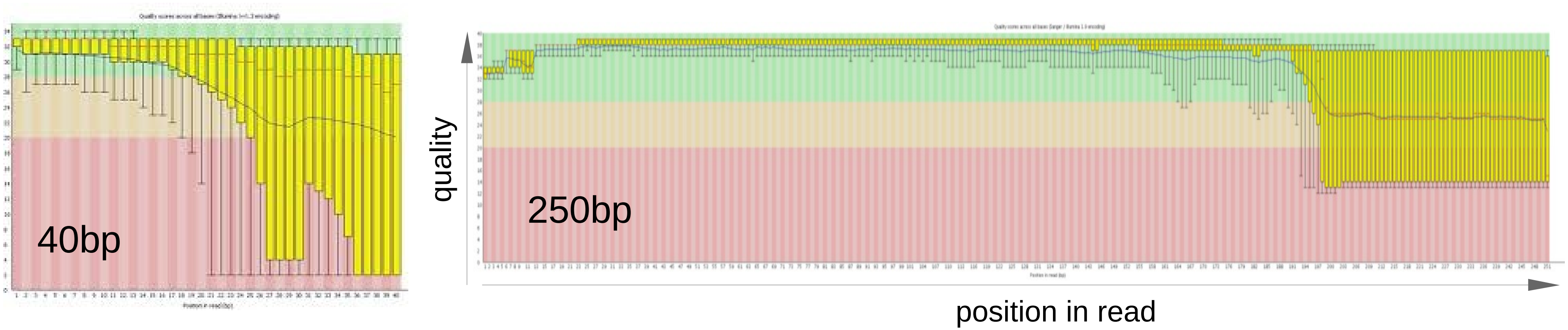
Hashing: build genome index

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



Hashing: search strategy

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



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

Hashing: optimizations & variants

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

111001 ATCGGT ⇔ ATCACT

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

By-reference assembly: BWT

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

Building a BWT

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

S: `mississippi$`

end-of-line marker

rotate

```
m i s s i s s i p p i $
$ m i s s i s s i p p i
i $ m i s s i s s i p p
p i $ m i s s i s s i p
p p i $ m i s s i s s i
i p p i $ m i s s i s s
s i p p i $ m i s s i s
s s i p p i $ m i s s i
i s s i p p i $ m i s s
s i s s i p p i $ m i s
s s i s s i p p i $ m i
i s s i s s i p p i $ m
```

sort

```
$ m i s s i s s i p p i
i $ m i s s i s s i p p
i p p i $ m i s s i s s
i s s i p p i $ m i s s
i s s i s s i p p i $ m
m i s s i s s i p p i $
p i $ m i s s i s s i p
p p i $ m i s s i s s i
s i p p i $ m i s s i s
s i s s i p p i $ m i s
s s i p p i $ m i s s i
s s i s s i p p i $ m i
```

BWT(S):

```
i
p
s
s
m
$
p
i
s
s
i
i
```

BWT properties

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

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

Why BWT?

- Space complexity H. sapiens (3Gb)
  - Suffix tree: ~ 20 * |Genome| 60 GB
  - Suffix array: ~ 4 * |Genome| 12 GB
  - BWT-FM: ~ 0.5 * |Genome| 1.5 GB
- Sounds good, but how can we search?
- FM-index (Ferragina and Manzini, 2000)

FM-Index: LF mapping

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

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

s_0 i_1 p_1 p_0 i_0 $\$_0$

F	L
$\$_0$	i_0
i_0	p_0
i_1	s_0
i_2	s_1
i_3	m_0
m_0	$\$_0$
p_0	p_1
p_1	i_1
s_0	s_2
s_1	s_3
s_2	i_2
s_3	i_3

FM Index

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

FO	F		L
0	$	m i s s i s s i p p	i
1	i	$ m i s s i s s i p	p
	i	p p i $ m i s s i s	s
	i	s s i p p i $ m i s	s
	i	s s i s s i p p i $	m
5	m	i s s i s s i p p i	$
6	p	i $ m i s s i s s i	p
	p	p i $ m i s s i s s	i
8	s	i p p i $ m i s s i	s
	s	i s s i p p i $ m i	s
	s	s i p p i $ m i s s	i
	s	s i s s i p p i $ m	i

i	m	p	s
1	0	0	0
1	0	1	0
1	0	1	1
1	0	1	2
1	1	1	2
1	1	1	2
1	1	2	2
2	1	2	2
2	1	2	3
2	1	2	4
3	1	2	4
4	1	2	4

FM Index

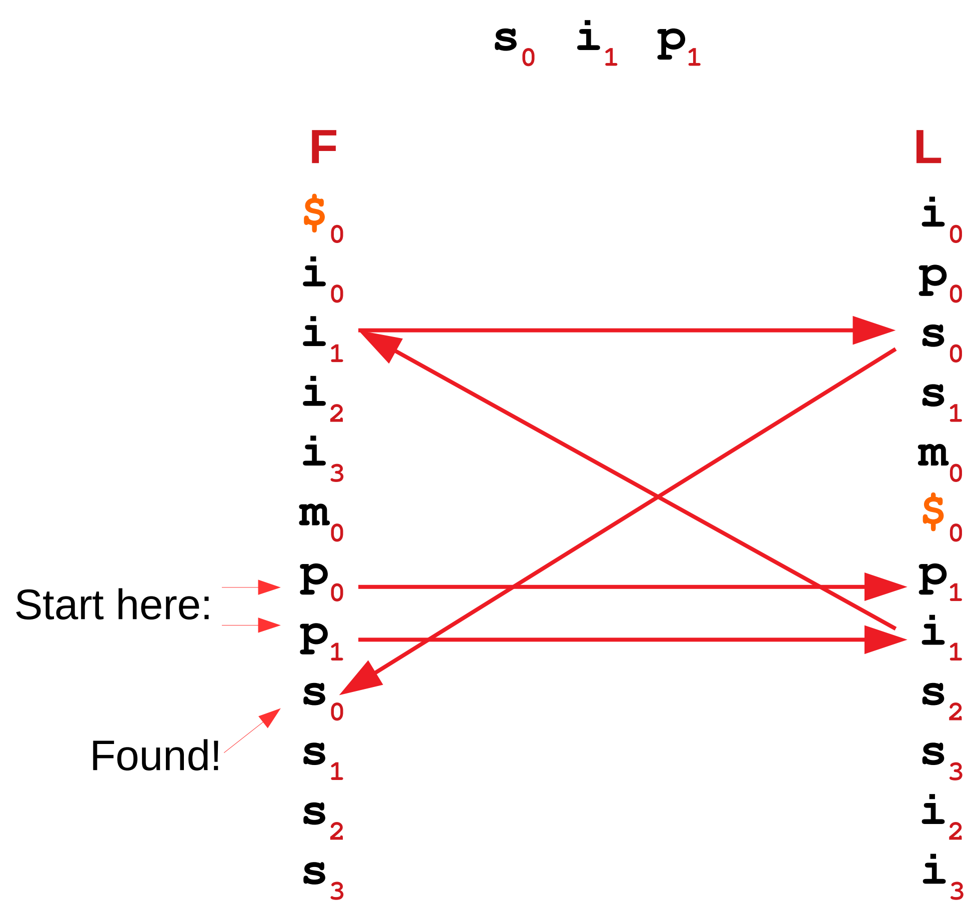
- In addition to BWT itself, the FM-index stores:
  - F offsets → FO, L rank checkpoints → LRC, SA sample → SAS

SAS	FO	F											L	LRC: i	LRC: m	LRC: p	LRC: s
	0	$	m	i	s	s	i	s	s	i	p	p	i	1	0	0	0
10	1	i	$	m	i	s	s	i	s	s	i	p	p				
		i	p	p	i	$	m	i	s	s	i	s	s				
4		i	s	s	i	p	p	i	$	m	i	s	s				
		i	s	s	i	s	s	i	p	p	i	$	m				
0	5	m	i	s	s	i	s	s	i	p	p	i	$	1	1	1	2
	6	p	i	$	m	i	s	s	i	s	s	i	p				
8		p	p	i	$	m	i	s	s	i	s	s	i				
6	8	s	i	p	p	i	$	m	i	s	s	i	s				
		s	i	s	s	i	p	p	i	$	m	i	s				
		s	s	i	p	p	i	$	m	i	s	s	i	3	1	2	4
2		s	s	i	s	s	i	p	p	i	$	m	i				

FM-Index: Pattern search

Find pattern: **sip**

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



Learn from the experts!

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

Questions?

References

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

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

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

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

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

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

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

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

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

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

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

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