# Genome Annotaion

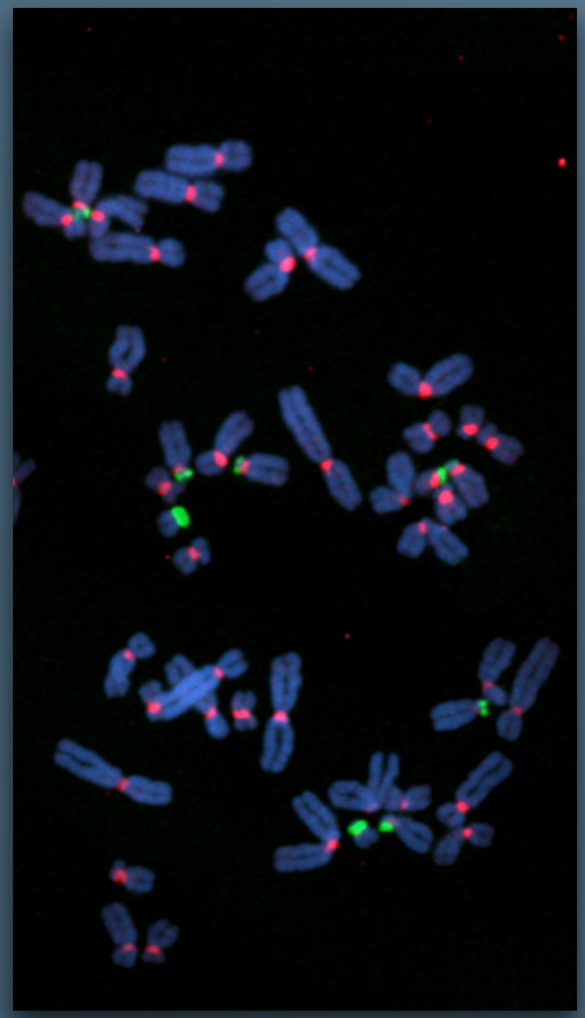
# GETTING SEQUENCES IS EASY

TGCATCGATCGTAGCTAGCTAGCGCATGCTAGCTAGCTAGCTAGCTACGATGCATCG
TGCATCGATCGATGCATGCTAGCTAGCTAGCTAGCATGCTAGCTAGCTAGCTATTGG
CGCTAGCTAGCATGCATGCATGCATCGATGCATCGATTATAAGCGCGATGACGTCAG
CGCGCGCATTATGCCGCGGCATGCTGCGCACACACAGTACTATAGCATTAGTAAAAA
GGCCGCGTATATTTTACACGATAGTGCGGCGCGGCGCGTAGCTAGTGCTAGCTAGTC
TCCGGTTACACAGGTAGCTAGCTAGCTGCTAGCTAGCTGCTGCATGCATGCATTAGT
AGCTAGTGTAGCTAGCTAGCATGCTGCTAGCATGCAGCATGCATCGGGCGCGATGCT
GCTAGCGCTGCTAGCTAGCTAGCTAGCTAGGCGCTAATTATTTATTTTGGGGGGTTA
AAAAAAAAAATTTCGCTGCTTATACCCCCCCCCACATGATGATCGTTAGTAGCTACT
AGCTCTCATCGCGCGGGGGGATGCTTAGCGTGGTGTGTGTGTGTGGTGTGTGTGGTC
CTATAATTAGTGCATCGGCGCATCGATGGCTAGTCGATCGATCGATTTTATATATCT
AAAGACCCCATCTCTCTCTCTTTTCCCTTCTCTCGCTAGCGGGCGGTACGATTTACC
GGCCGCGTATATTTTACACGATAGTGCGGCGCGGCGCGTAGCTAGTGCTAGCTAGTC
AGCTCTCATCGCGCGGGGGGATGCTTAGCGTGGTGTGTGTGTGTGGTGTGTGTGGTC
TGCATCGATCGATGCATGCTAGCTAGCTAGCTAGCATGCTAGCTAGCTAGCTATTGG
CTATAATTAGTGCATCGGCGCATCGATGGCTAGTCGATCGATCGATTTTATATATCT
CGCTAGCTAGCATGCATGCATGCATCGATGCATCGATTATAAGCGCGATGACGTCAG
TCCGGTTACACAGGTAGCTAGCTAGCTGCTAGCTAGCTGCTGCATGCATGCATTAGT

# GETTING SEQUENCES IS EASY - RELATIVELY

The shortest human chromosome (21) is 45.1 Mb long

The longest single read achieved so far is “only” 4 Mb long. It was obtained using nanopore technology



# Sequence assembly

# Sequence assembly

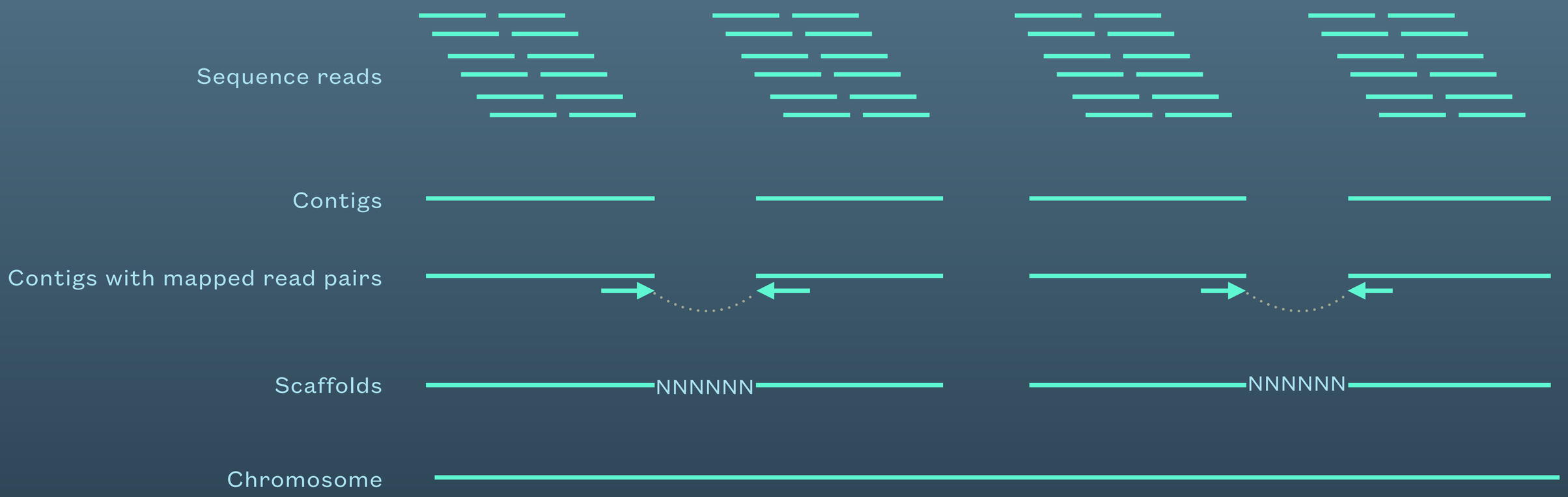
- A fundamental goal of DNA sequencing is to generate large, continuous regions of DNA sequence – CONTIGS
- In principle, assembling a sequence is just a matter of finding overlaps and combining them.
- In practice:
  - most genomes contain multiple copies of many sequences,
  - there are random mutations (either naturally occurring cell-to-cell variation or generated by PCR or cloning),
  - there are sequencing errors

# Genome assembly

- Whole-genome shotgun sequencing starts with copying (NGS) and fragmenting the DNA
- “Shotgun” refers to the random fragmentation of the whole genome; like it was fired from a gun

| | |
|---|---|
| Source material | GGTCTCTAAGCGCTAGACTAGGACTGAAAATC |
| Copies | GGTCTCTAAGCGCTAGACTAGGACTGAAAATC<br>GGTCTCTAAGCGCTAGACTAGGACTGAAAATC<br>GGTCTCTAAGCGCTAGACTAGGACTGAAAATC<br>GGTCTCTAAGCGCTAGACTAGGACTGAAAATC |
| Fragments | GGTCT CTAAGC GCTAGACTAGGACTG AAAATC<br>GGTCTCTAAGC GCTAGACTAGGACTGAAAA TC<br>GG TCTC TAAGCGCT AGACTAGG ACTGAAAATC<br>GGTCTCTAAG CGCTAGACT AGGACTGAAAATC |

# Genome assembly



# Genome assembly: coverage

**Coverage** it is a short for *average coverage*: the average number of reads covering a position in the genome

```
GGTCTCTAAGCGCTAGACTAGGACTGAAAATC
GGTCTCTA
GGTCTCTAAG
          AAGCGCTAGACTA
          AAGCGCTAG
             GCGCTAGAC
             GCGCTAGACTAGG
                          AGGACTGAAA
                          AGGACTGAAAA
                           GGACTGAAAATC
```

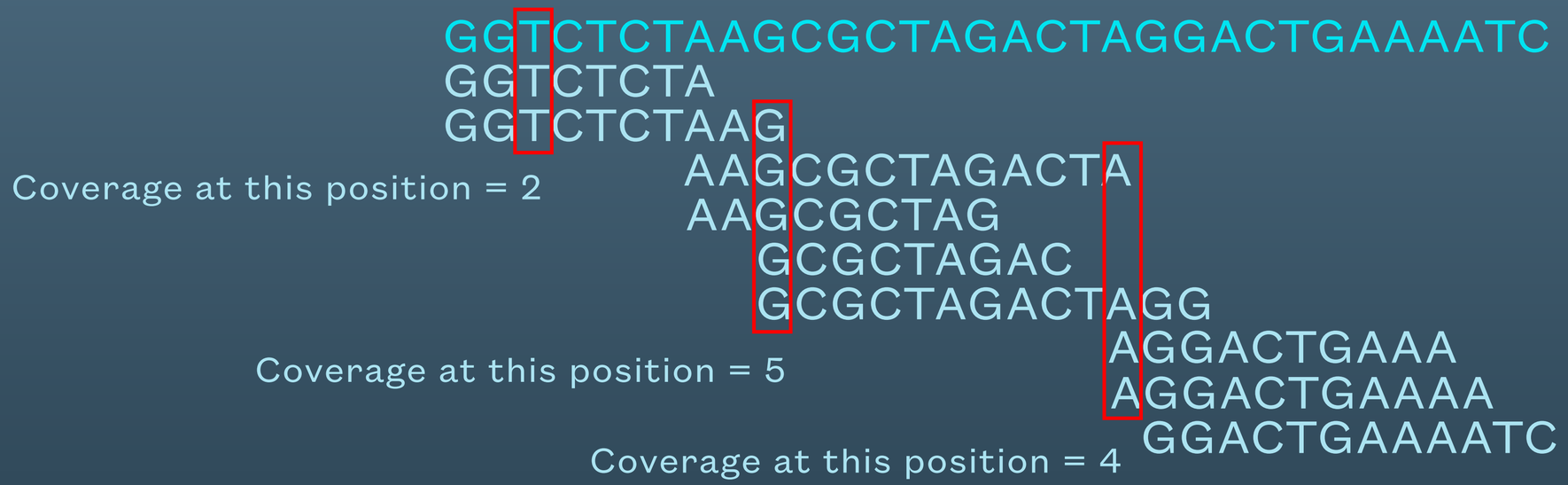
32 nucleotides

95 nucleotides

Average coverage = 95 / 32 = 3x

# Genome assembly: coverage

**Coverage** can also refer to the number of reads covering a particular position in the genome



# Genome assembly: overlapping reads

Let’s assume that two reads truly originate from the same genomic region.

Why might there be a difference?

```
AAGCGCTAGACTA
AAGCGCAAG
   GCGCAAGAC
   GCGCTAGACTAGG
```

# Genome assembly: overlapping reads

Let’s assume that two reads truly originate from the same genomic region.
Why might there be a difference?

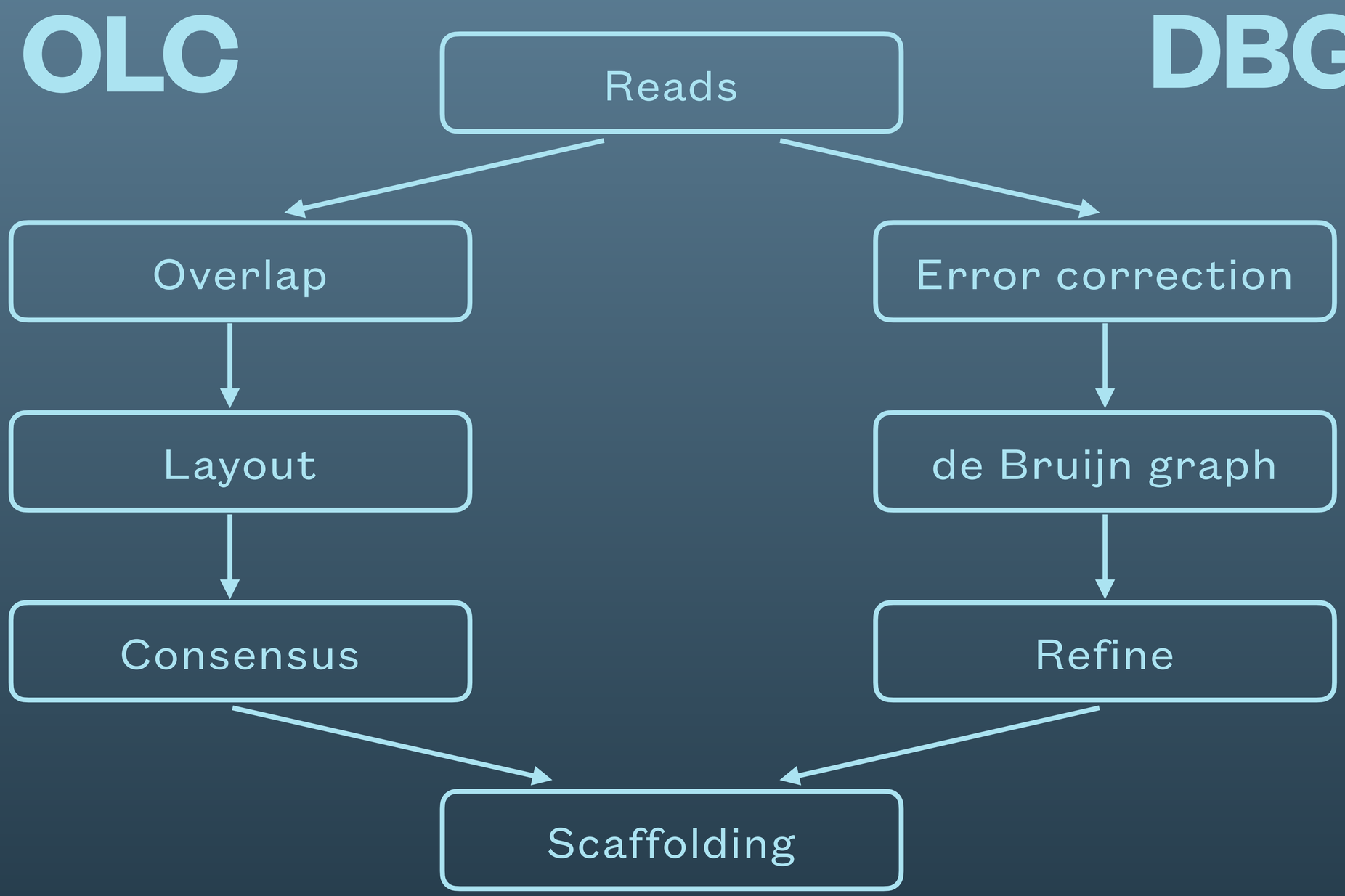
```
AAGCGCTAGACTA
AAGCGCAAG
   GCGCAAGAC
   GCGCTAGACTAGG
```

1. Sequencing error
2. Difference between inherited copies of a chromosome

| | |
|---|---|
| Maternal chromosome | AAGCGCTAGACTA |
| Paternal chromosome | AAGCGCAAGACTA |

# Two approaches to genome assembly

- Overlap-Layout-Consensus (OLC) - string graph assemblers
  - construct overlap graph directly from reads, eliminating redundant reads;
  - trace path in graph for assembly
  - examples: Arachne, Canu, Celara Assembler, HiCanu, SGA (String Graph Assembler)
- de Bruijn graph assemblers
  - construct *k*-mer graph from reads; original reads are discarded
  - trace path in graph for assembly
  - examples: ABySS, Euler, EULER-SR, SOAPdenovo, Velvet

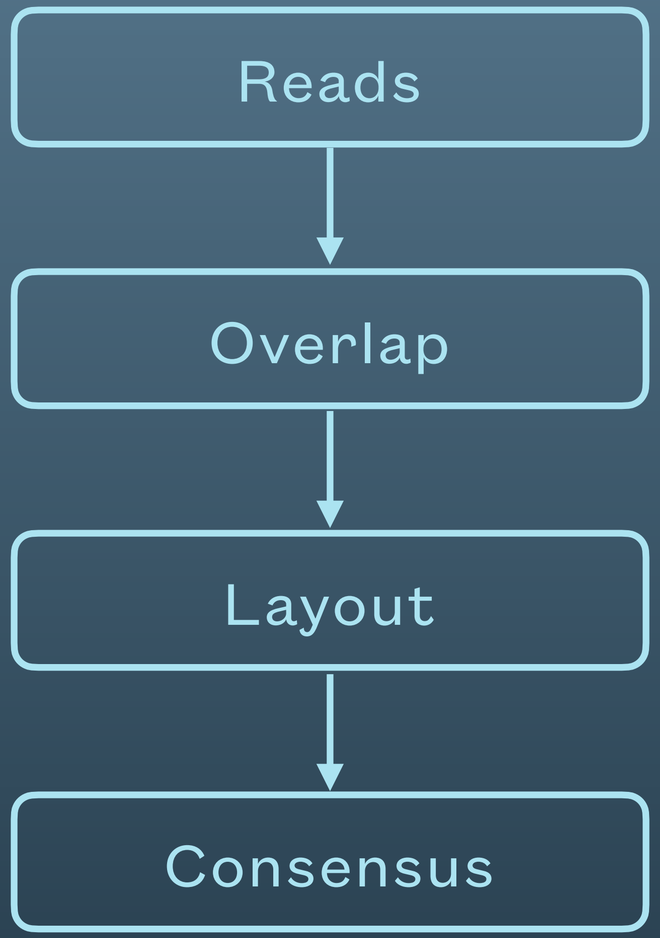

OLC
DBG
Reads
Overlap
Layout
Consensus
Error correction
de Bruijn graph
Refine
Scaffolding

# Genome assembly - OLC approach

Build overlap graph

Bundle stretches of the overlap graph into contigs

Pick most likely nucleotide sequence for each contig



# Building overlap graph

Finding all overlaps is similar to building a directed graph where directed edges connect overlapping nodes (reads)

AGACTAGGACTG
AGGACTGAAAATC

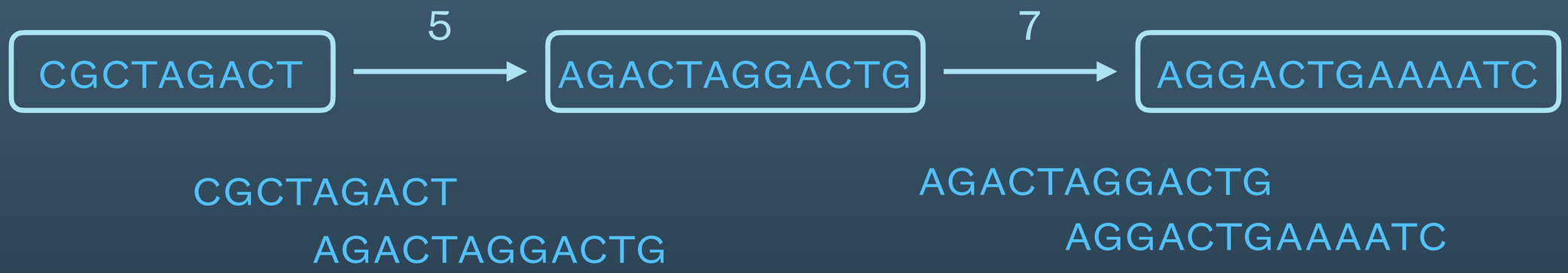
Suffix of one read is similar to a prefix of another read

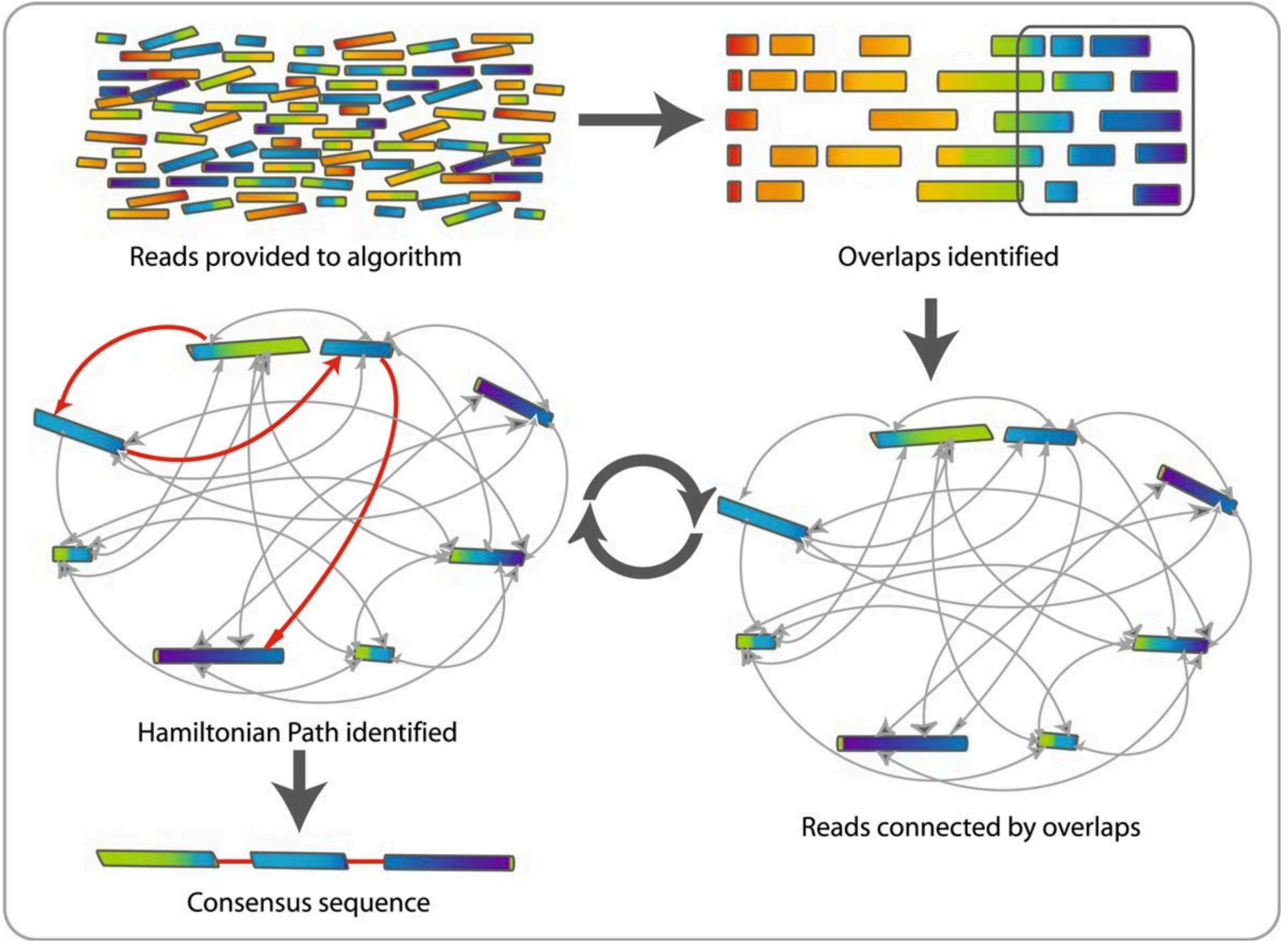
# An overlap graph, where an overlap is a suffix/prefix match of at least 5 characters

A vertex is a read, a directed edge is an overlap between suffix of *source* and prefix of *sink.*

Vertices (reads): {a: CGCTAGACT, b: AGACTAGGACTG, c: AGGACTGAAAATC}

Edges (overlaps): {(a,b),(b,c)}




Reads provided to algorithm
Overlaps identified
Hamiltonian Path identified
Reads connected by overlaps
Consensus sequence

# Genome assembly - OLC approach

- Efficient computation of all read overlaps is a key to success
- Finding overlaps is computationally demanding and OLC-based assembly tends to be slow.
  - For example assembly of a human genome after Illumina (short read) sequencing with 1.2 billion reads took 1427 CPU-hours or 140 hours of real time using  SGA assembler

# Genome assembly - de Bruijn graph

*k*-mer is a substring of length k

*mer* - from Greek meaning “part”

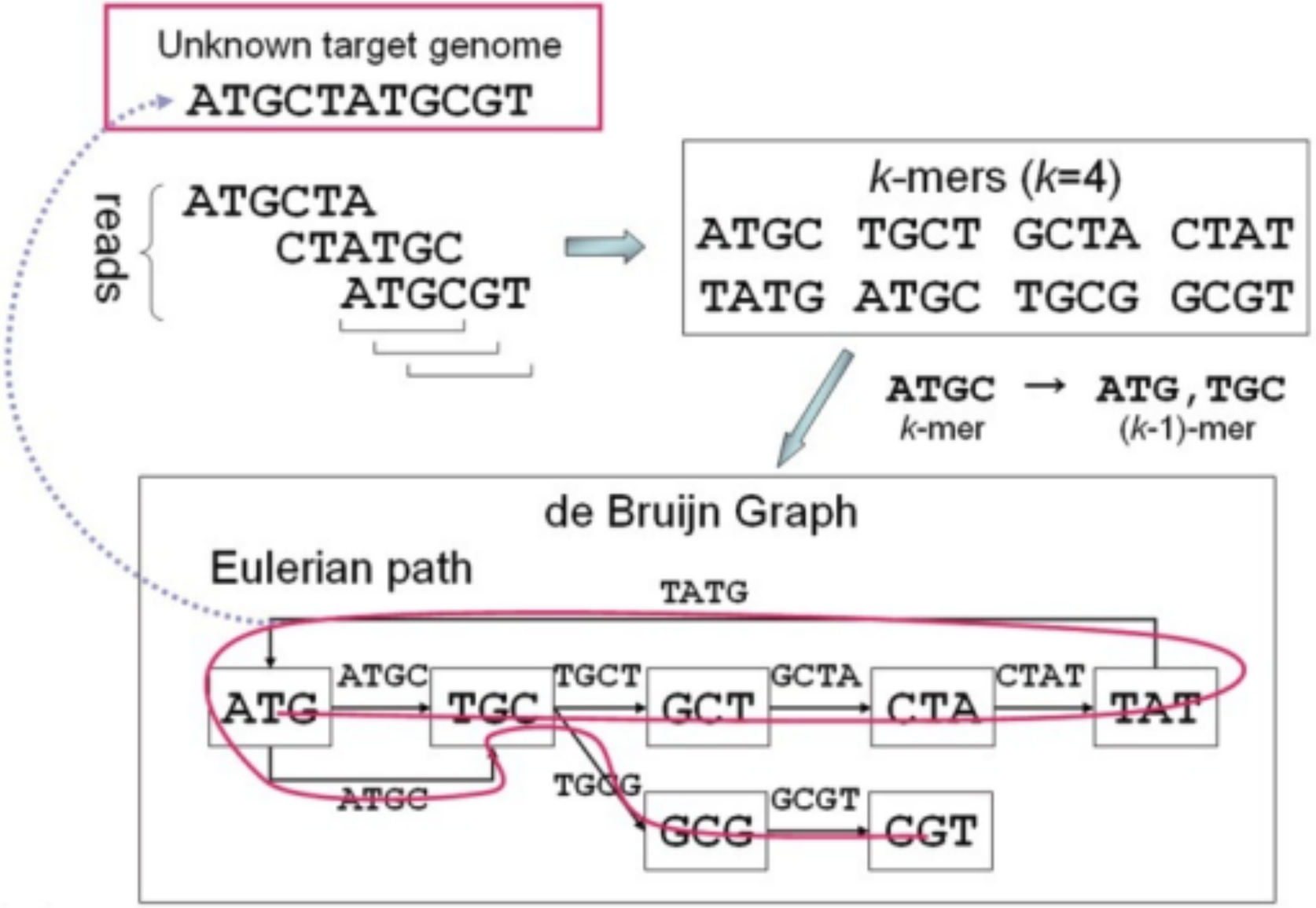
S: AGACTAGGACTG

All *4*-mers of S

AGAC
GACT
ACTA
CTAG
TAGG
AGGA
GGAC
GACT
ACTG

All *3*-mers of S

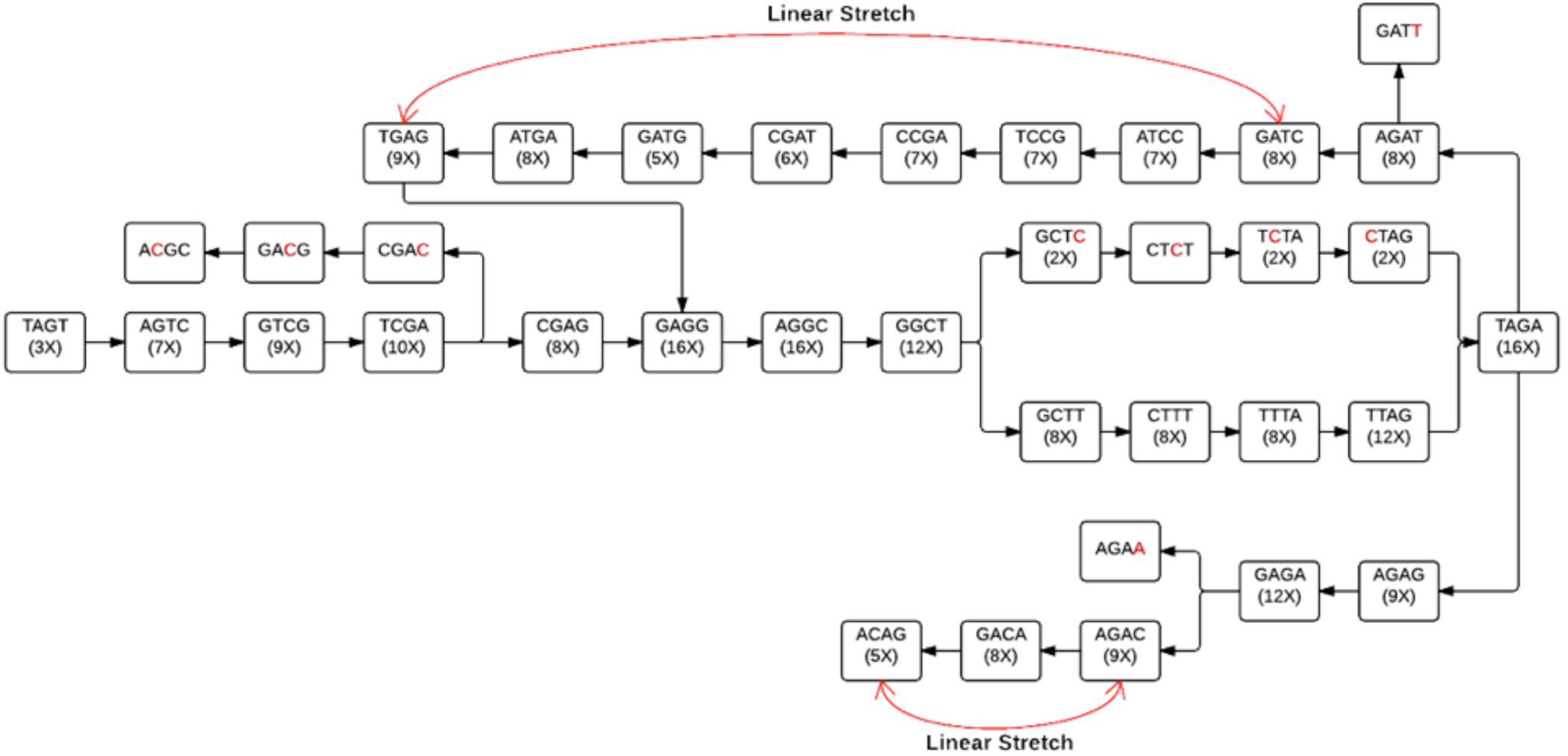
AGA
GAC
ACT
CTA
TAG
AGA
GGA
GAC
ACT
CTG




Continuous linear stretches within the graph

Assembler keeps information about reads coverage for each k-mer/node.

Flicek & Birney (2009) Nat Meth, 6: S6-S12.




Graph is simplified to combine nodes that are associated with the continuous linear stretches into single, larger nodes of various k-mer sizes.
Error correction removes the tips and bubbles that result from sequencing errors.
Sequencing errors are low frequency tips in the graph.

Flicek & Birney (2009) Nat Meth, 6: S6-S12.

# Genome assembly - de Bruijn

- de Bruijn graph approach limitations
  - reads are immediately split into shorter *k*-mers and consequently cannot resolve repeats very well
  - they don’t deal with sequencing errors very well
  - reads coherence is lost and some paths through de Bruijn graph are inconsistent with respect to input reads.

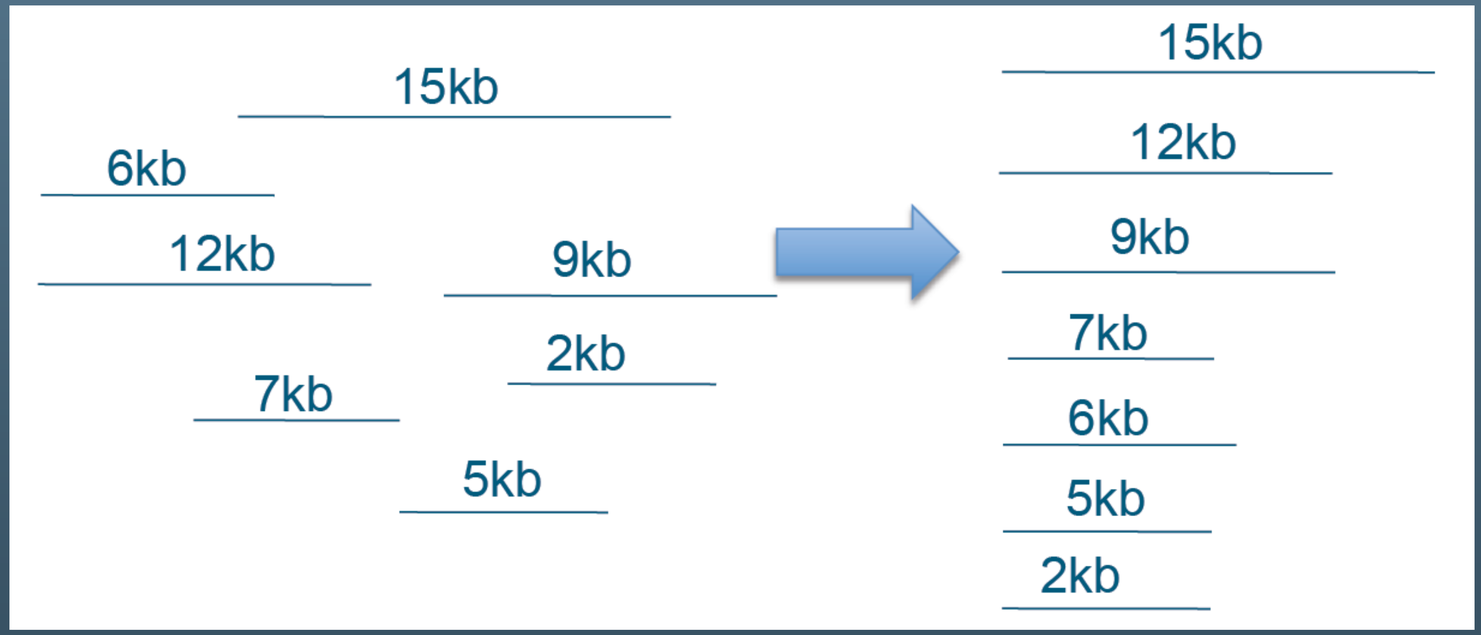
# Comparison of OLC and de Bruijn graph assembly

Assembly statistics for *Caenorhabditis elegans* dataset (33.8M 100-nt read pairs)

| | **SGA** (OLC ) | **Velvet** (de Bruijn) |
|---|---|---|
| **Contig N50 size** | 16.8 kb | 13.6 kb |
| **Scaffold N50 size** | 26.3 kb | 31.3 kb |
| **Sum aligned contig size** | 96.8 Mb | 95.2 Mb |
| **Reference bases covered** | 96.2 Mb | 94.8 Mb |
| **Mismatch rate at assembled bases** | 1 per 21.5 kb | 1 per 8.8 kb |
| **Total CPU time** | 41 hours | 2 hours |
| **Max memory usage** | 4.5 GB | 23 GB |

Genome Res. 2012. 22: 549-556

# Assembly evaluation - N50




If one orders the set of contigs produced by the assembler by size, then N50 is the size of the contig such that 50% of the total bases are in contigs of equal or greater size.

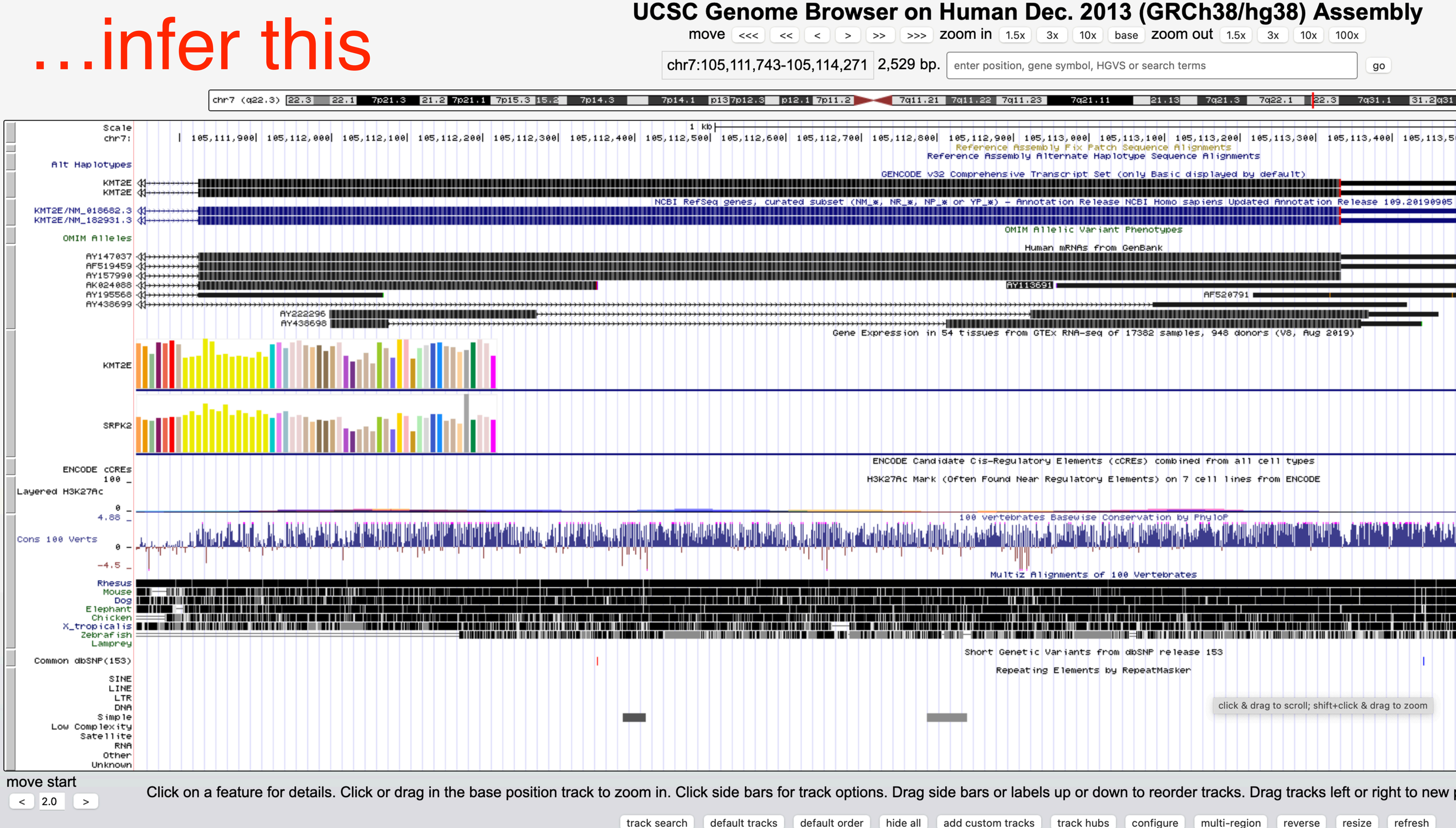
15+12+9+7+6+5+2 = 56

56/2 = 28 -> N50 is 9kb (15+12 = 27 is less than 50%)

# CHALLENGE: HOW FROM THIS…

TGCATCGATCGTAGCTAGCTAGCGCATGCTAGCTAGCTAGCTAGCTACGATGCATCG
TGCATCGATCGATGCATGCTAGCTAGCTAGCTAGCATGCTAGCTAGCTAGCTATTGG
CGCTAGCTAGCATGCATGCATGCATCGATGCATCGATTATAAGCGCGATGACGTCAG
CGCGCGCATTATGCCGCGGCATGCTGCGCACACACAGTACTATAGCATTAGTAAAAA
GGCCGCGTATATTTTACACGATAGTGCGGCGCGGCGCGTAGCTAGTGCTAGCTAGTC
TCCGGTTACACAGGTAGCTAGCTAGCTGCTAGCTAGCTGCTGCATGCATGCATTAGT
AGCTAGTGTAGCTAGCTAGCATGCTGCTAGCATGCAGCATGCATCGGGCGCGATGCT
GCTAGCGCTGCTAGCTAGCTAGCTAGCTAGGCGCTAATTATTTATTTTGGGGGGTTA
AAAAAAAAAATTTCGCTGCTTATACCCCCCCCCCACATGATGATCGTTAGTAGCTACT
AGCTCTCATCGCGCGGGGGGGATGCTTAGCGTGGTGTGTGTGTGTGGTGTGTGTGGTC
CTATAATTAGTGCATCGGCGCATCGATGGCTAGTCGATCGATCGATTTTATATATCT
AAAGACCCCATCTCTCTCTCTTTTCCCTTCTCTCGCTAGCGGGCGGTACGATTTACC
GGCCGCGTATATTTTACACGATAGTGCGGCGCGGCGCGTAGCTAGTGCTAGCTAGTC
AGCTCTCATCGCGCGGGGGGGATGCTTAGCGTGGTGTGTGTGTGTGGTGTGTGTGGTC
TGCATCGATCGATGCATGCTAGCTAGCTAGCTAGCATGCTAGCTAGCTAGCTATTGG
CTATAATTAGTGCATCGGCGCATCGATGGCTAGTCGATCGATCGATTTTATATATCT
CGCTAGCTAGCATGCATGCATGCATCGATGCATCGATTATAAGCGCGATGACGTCAG
TCCGGTTACACAGGTAGCTAGCTAGCTGCTAGCTAGCTGCTGCATGCATGCATTAGT

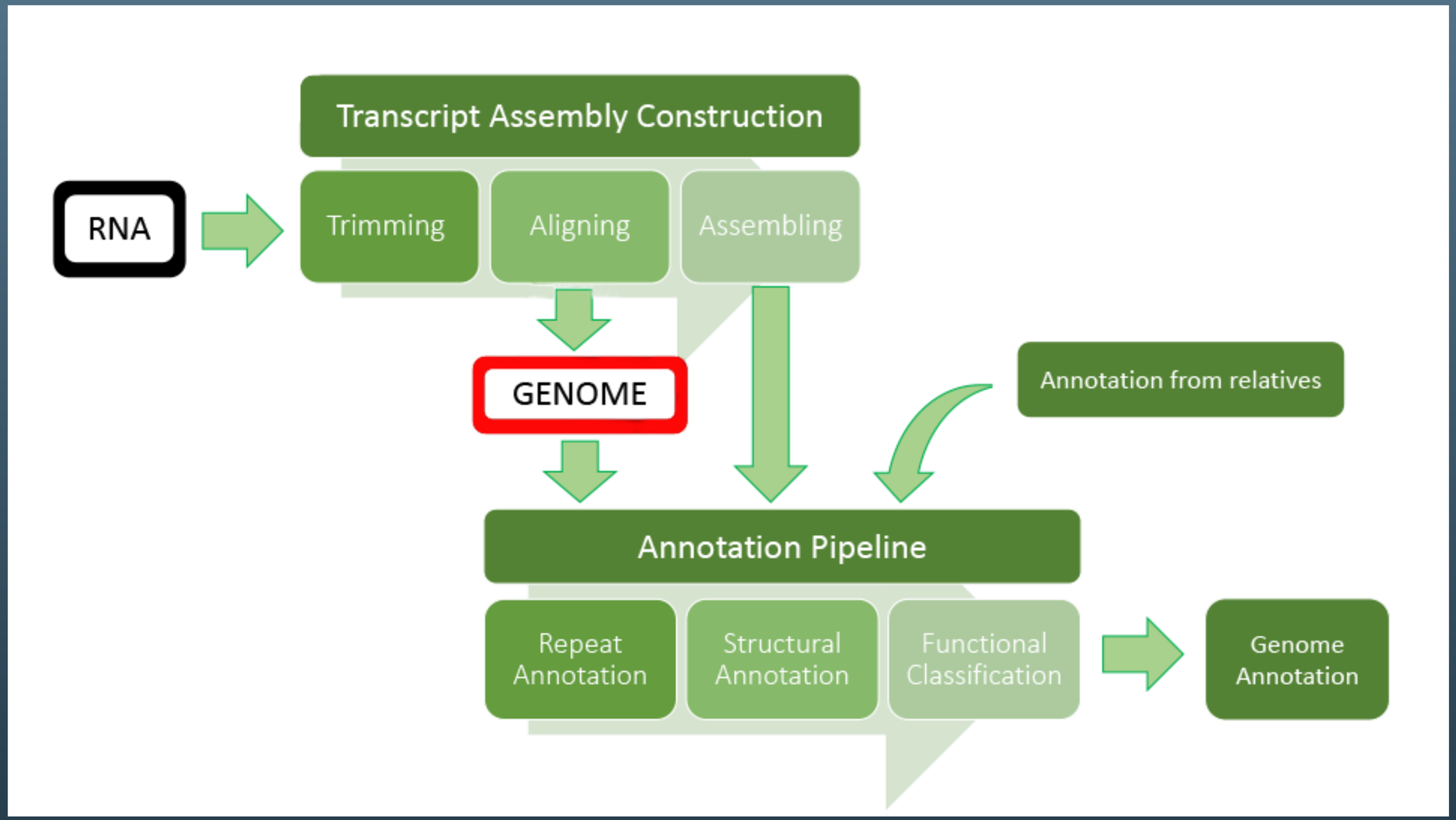
…infer this


UCSC Genome Browser on Human Dec. 2013 (GRCh38/hg38) Assembly
move <<< << < > >> >>> zoom in 1.5x 3x 10x base zoom out 1.5x 3x 10x 100x
chr7:105,111,743-105,114,271 2,529 bp.
enter position, gene symbol, HGVS or search terms
go
chr7 (q22.3)
Scale
chr7:
1 kb
105,111,900 105,112,000 105,112,100 105,112,200 105,112,300 105,112,400 105,112,500 105,112,600 105,112,700 105,112,800 105,112,900 105,113,000 105,113,100 105,113,200 105,113,300 105,113,400
Reference Assembly Fix Patch Sequence Alignments
Reference Assembly Alternate Haplotype Sequence Alignments
Alt Haplotypes
GENCODE v32 Comprehensive Transcript Set (only Basic displayed by default)
KMT2E
KMT2E
NCBI RefSeq genes, curated subset (NM_*, NR_*, NP_* or YP_*) - Annotation Release NCBI Homo sapiens Updated Annotation Release 109.20190905
KMT2E/NM_018682.3
KMT2E/NM_182931.3
OMIM Alleles
OMIM Allelic Variant Phenotypes
Human mRNAs from GenBank
AY147037
AF519459
AY157990
AK024088
AY195568
AY438699
AY113691
AF520791
AY222296
AY438698
Gene Expression in 54 tissues from GTEx RNA-seq of 17382 samples, 948 donors (V8, Aug 2019)
KMT2E
SRPK2
ENCODE cCREs
ENCODE Candidate Cis-Regulatory Elements (cCREs) combined from all cell types
100
Layered H3K27Ac
H3K27Ac Mark (Often Found Near Regulatory Elements) on 7 cell lines from ENCODE
0
4.88
100 vertebrates Basewise Conservation by PhyloP
Cons 100 Verts
0
-4.5
Multiz Alignments of 100 Vertebrates
Rhesus
Mouse
Dog
Elephant
Chicken
X_tropicalis
Zebrafish
Lamprey
Common dbSNP(153)
Short Genetic Variants from dbSNP release 153
Repeating Elements by RepeatMasker
SINE
LINE
LTR
DNA
Simple
Low Complexity
Satellite
RNA
Other
Unknown
click & drag to scroll; shift+click & drag to zoom
move start
< 2.0 >
Click on a feature for details. Click or drag in the base position track to zoom in. Click side bars for track options. Drag side bars or labels up or down to reorder tracks. Drag tracks left or right to new
track search default tracks default order hide all add custom tracks track hubs configure multi-region reverse resize refresh

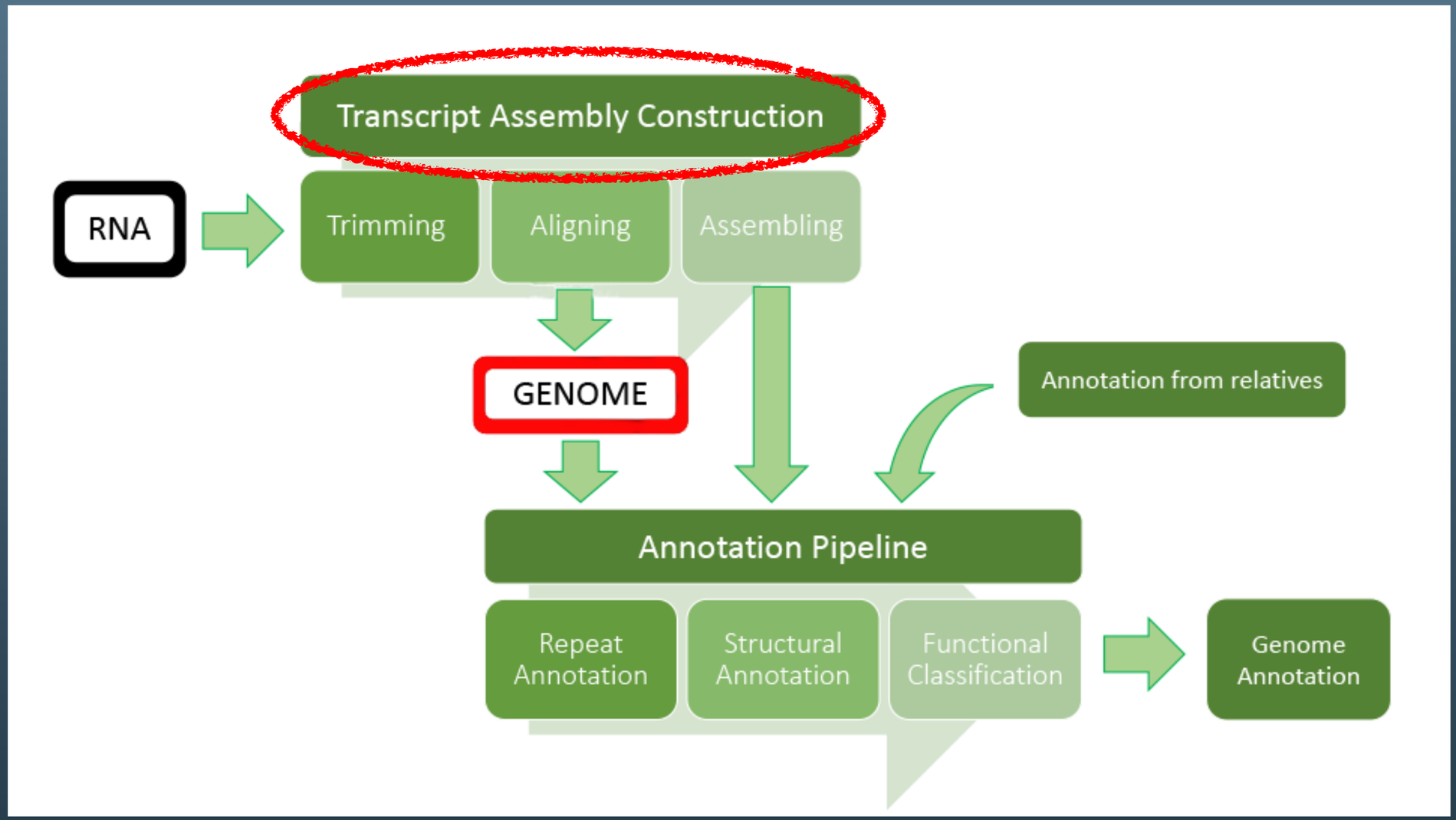
# What are we looking for?

- protein-coding genes
- RNA-coding genes
- gene promoters
- regulatory elements
- repetitive elements including transposons

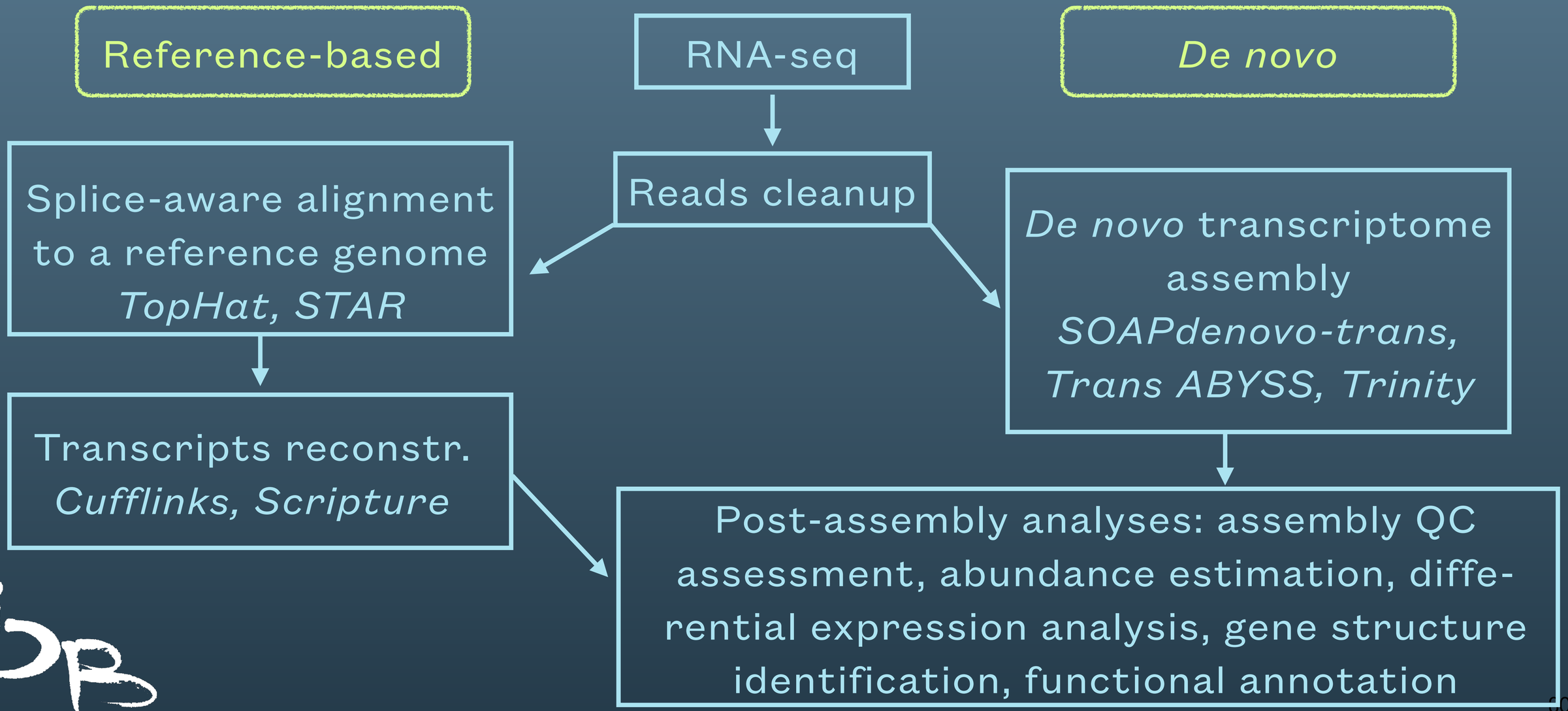
# Annotation workflow



# Annotation workflow

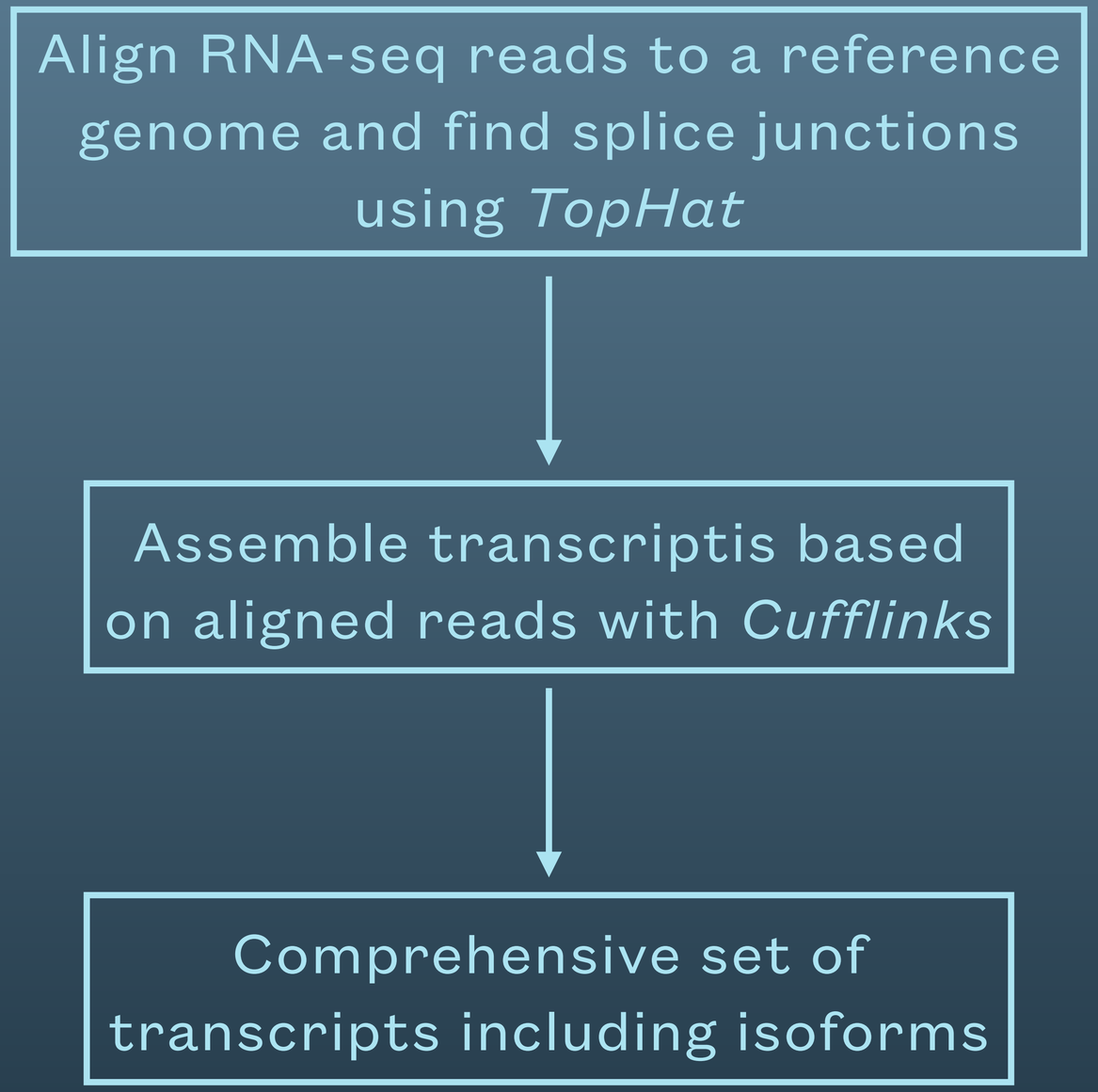


# Transcriptome assembly

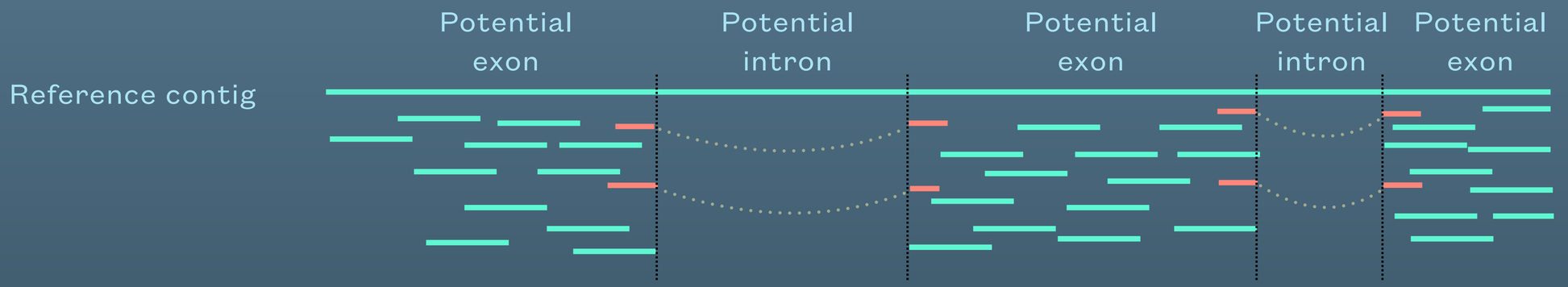


# Reference-based transcriptome assembly

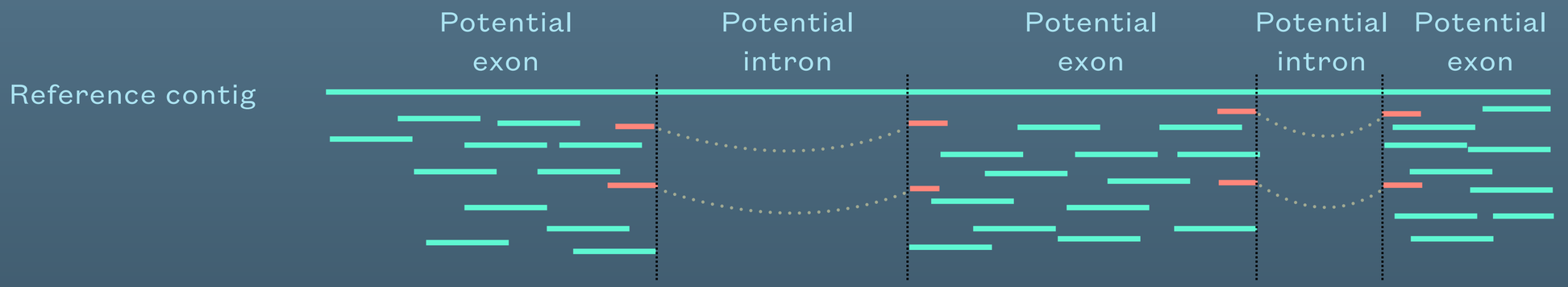
## Tuxedo Suite



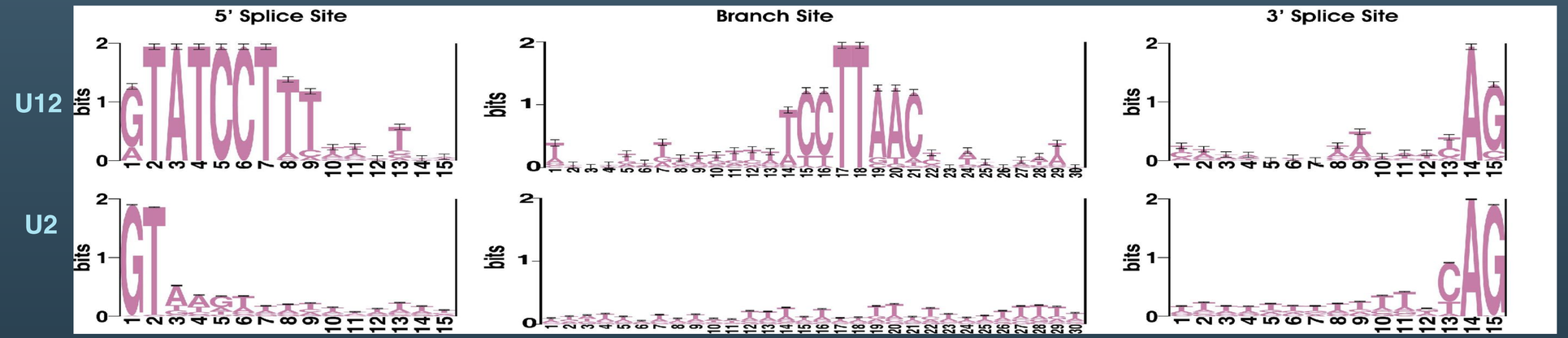
# Transcript assembly



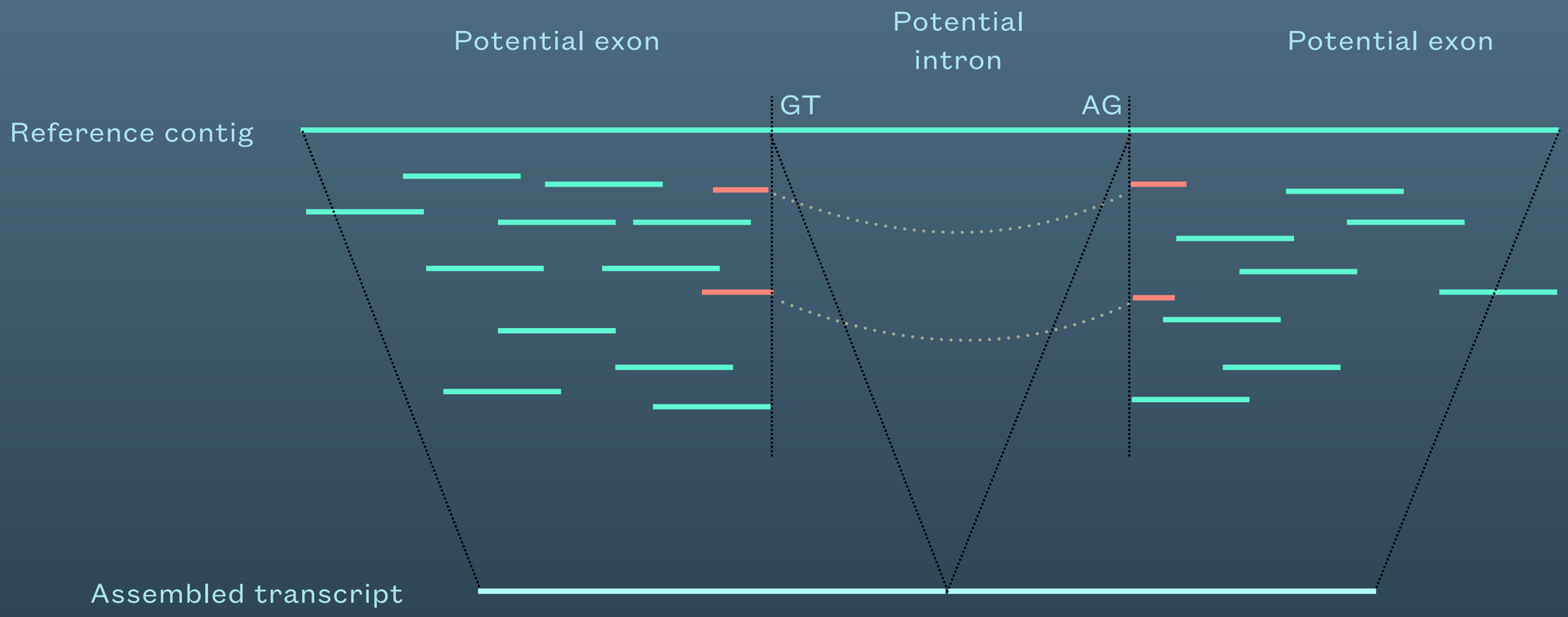
# Transcript assembly




Splicing signals



# Transcript assembly

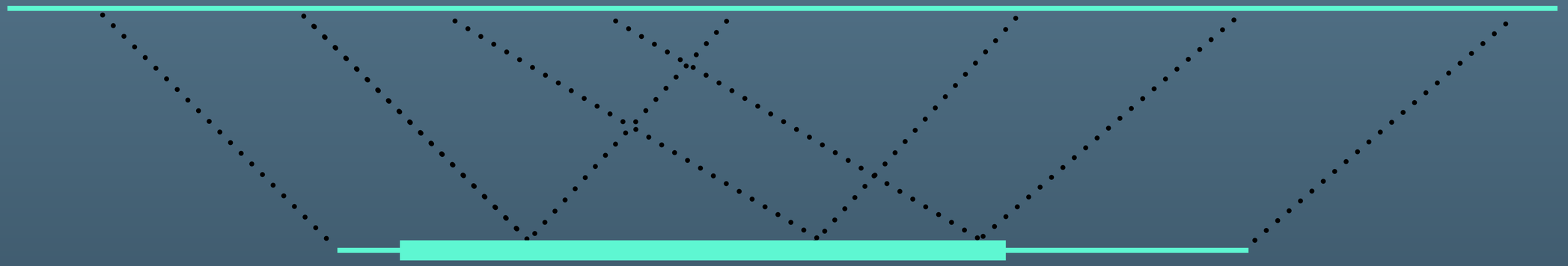


# Do we need *de novo* transcript assembly if we have a genome sequenced?

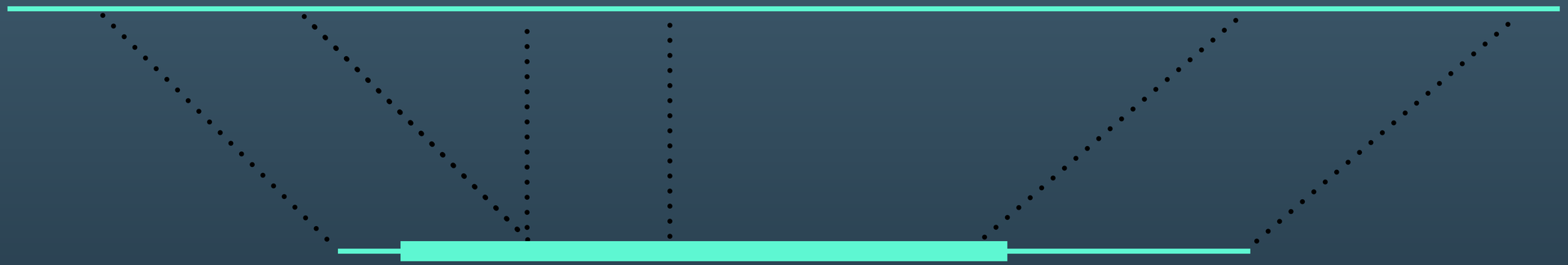

Yes, we do

# Miss- assembly

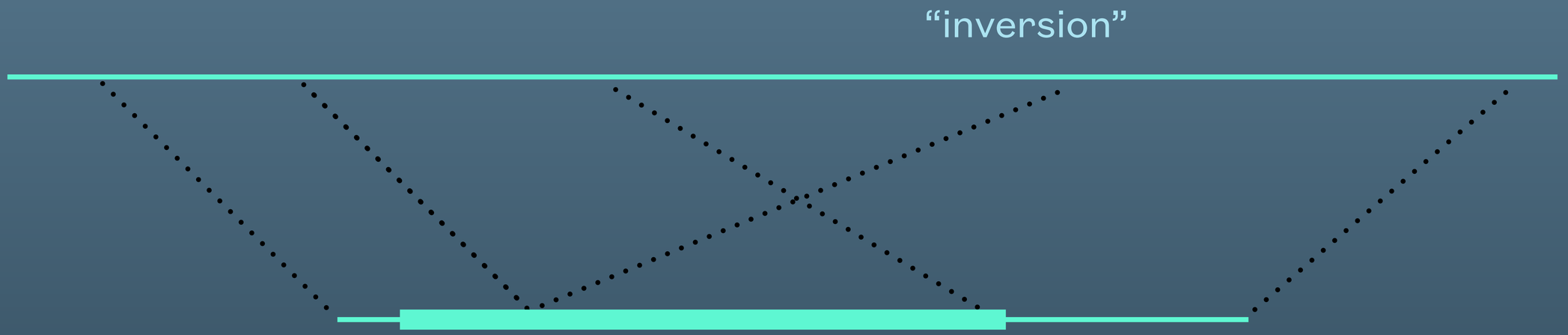
wrong contig fragments order



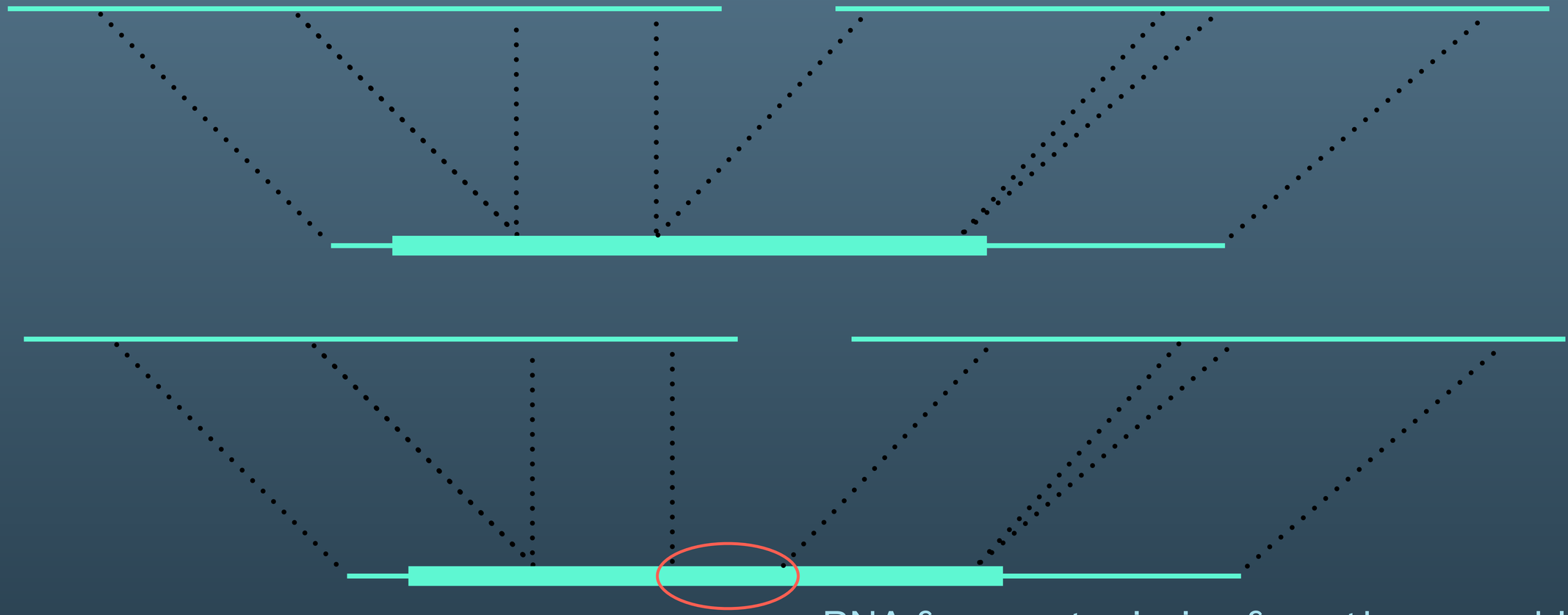
missing pieces of DNA



# Miss- assembly

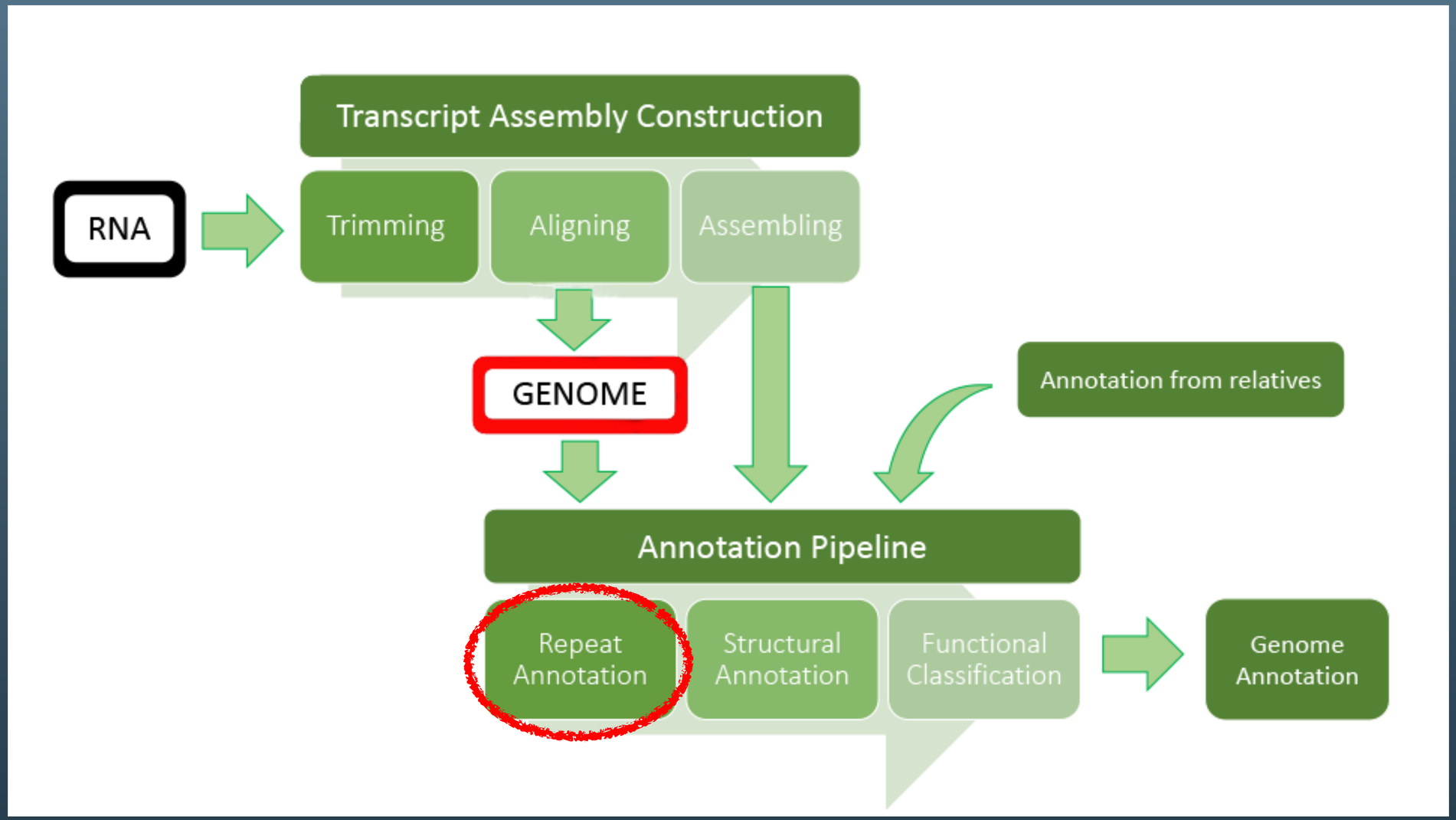


# Gene on different contigs



mRNA fragment missing from the assembly

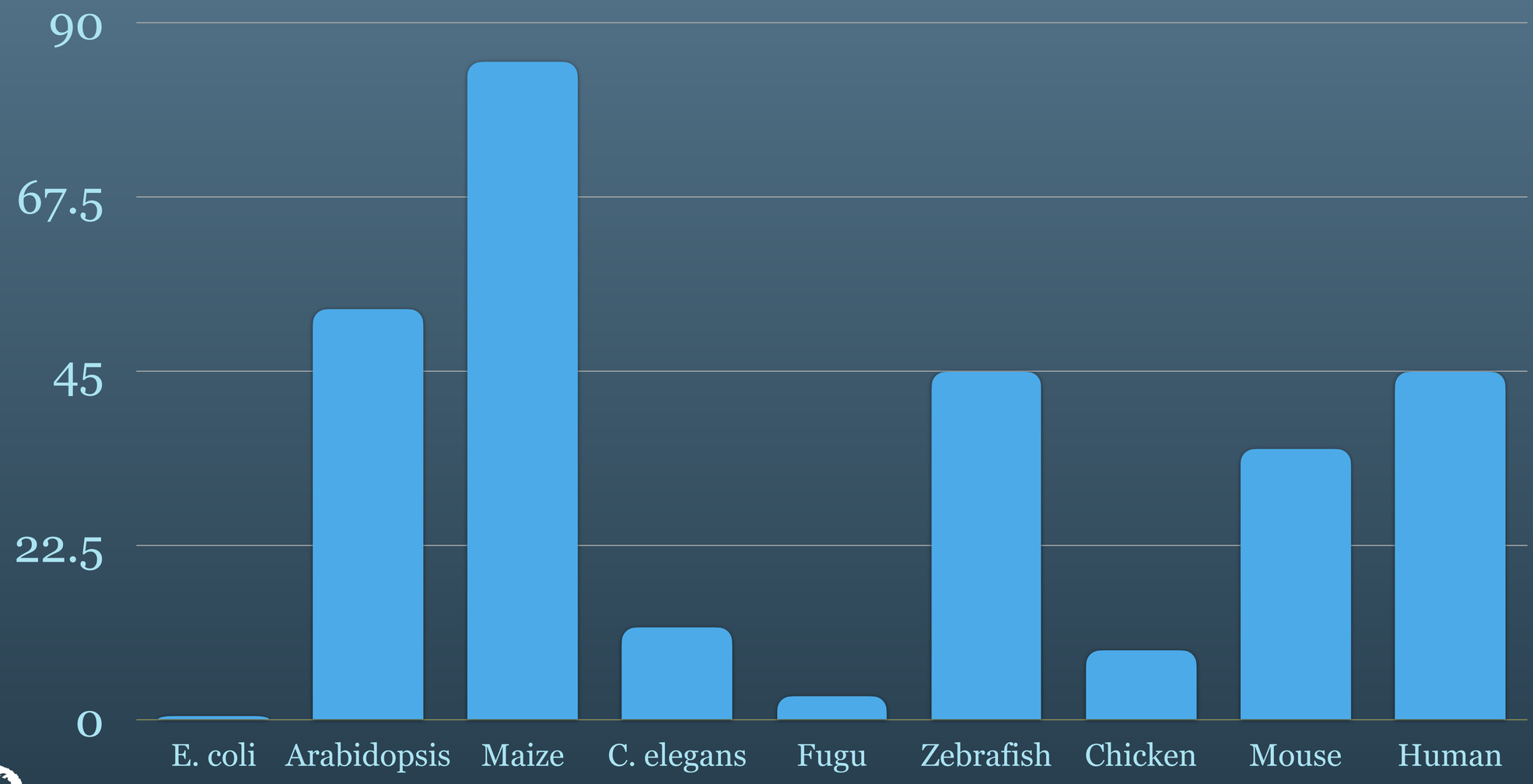
# Annotation workflow



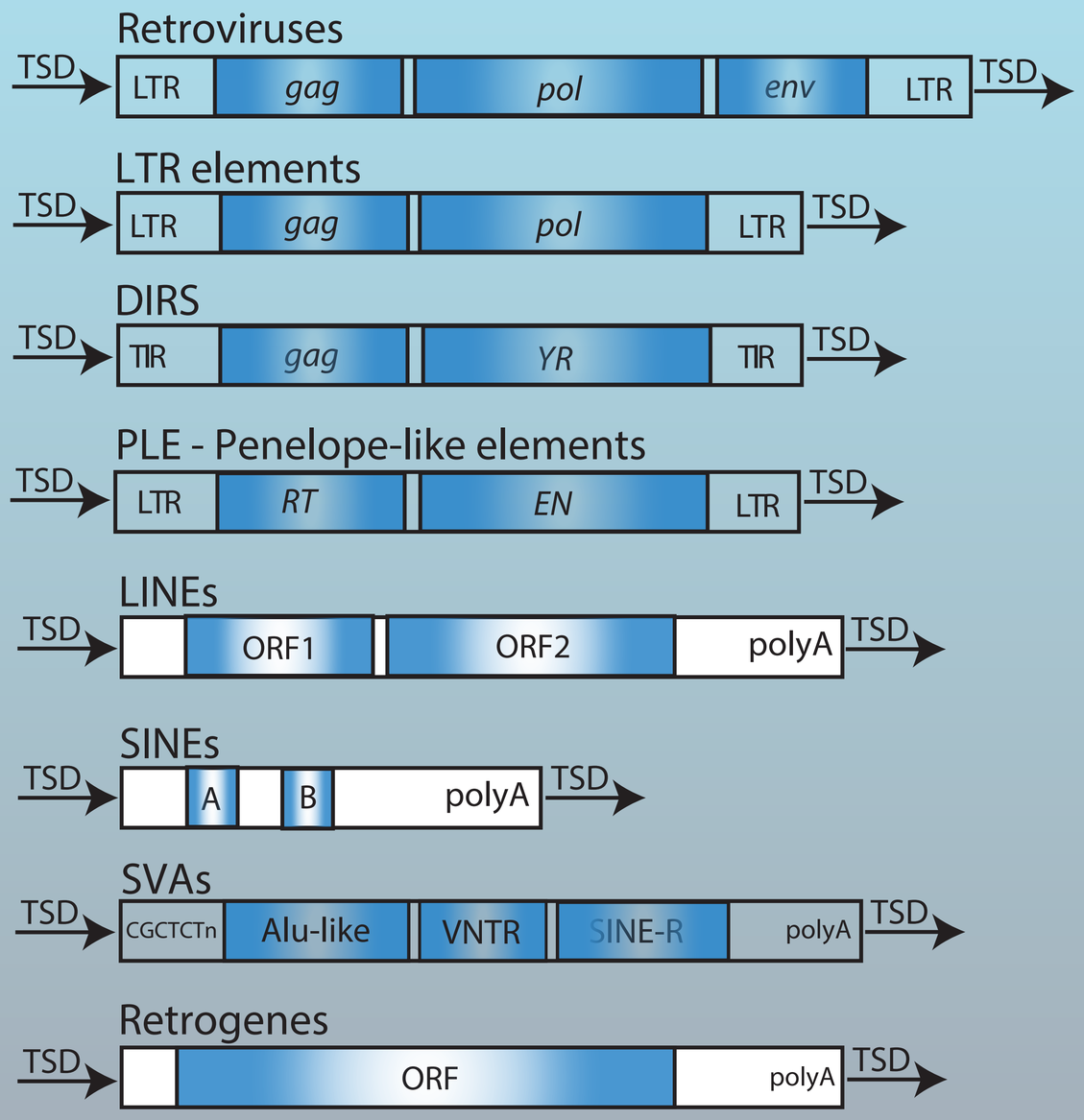
# Type of repeats

- Simple repeats
  - e.g. (AT)n, (TCT)n, microsatellites, etc.
- Interspersed repeats
  - Gene families, e.g. rRNA-coding
  - Transposable elements
- Low complexity regions

# TE-content in different genomes



Class I




Class I: Retrotransposons

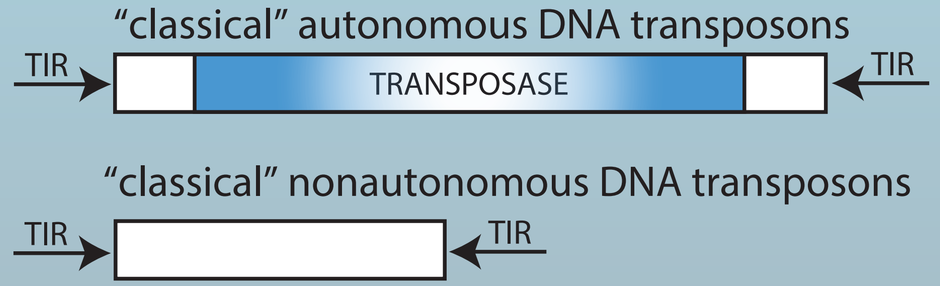
"copy-and-paste" transposition

Class II: DNA transposons

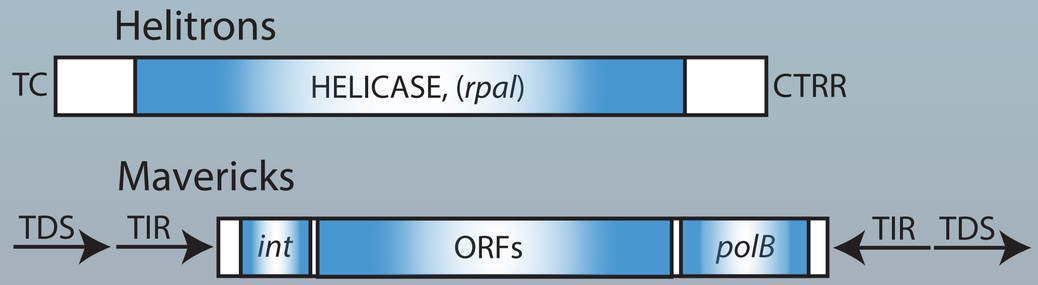
"cut-and-paste" transposition

Both classes are represented by autonomous and non-autonomous elements

Class II - subclass 1




Class II - subclass 2



# Identification of repeats

## Similarity-based

- RepeatMasker
- Censor

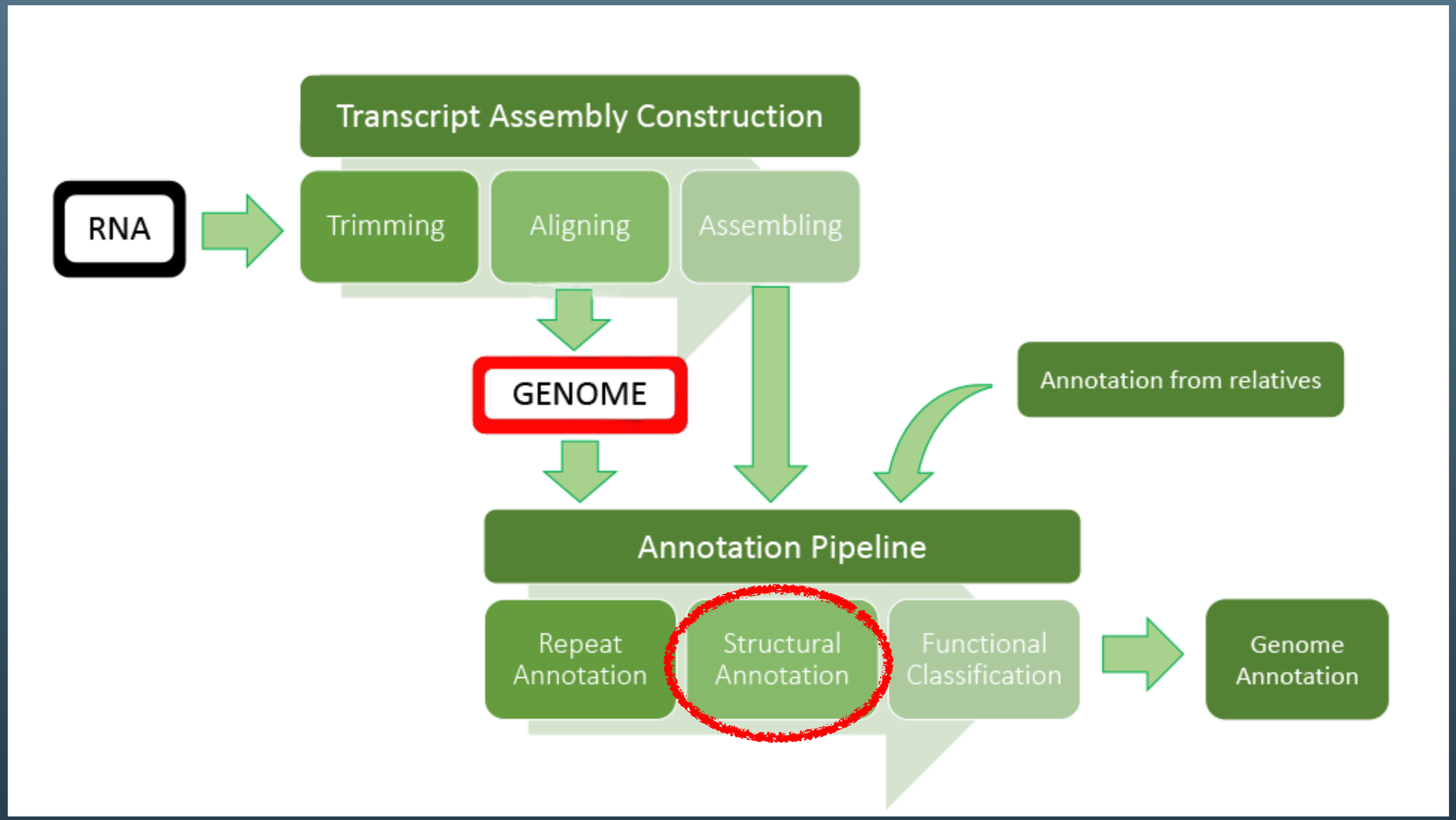
## *De novo*

- self-comparison approach (RECON, PILER, BLASTER)
- k-mer approach: sequences are scanned for overrepresentation of strings of certain length (REPuter, Vmatch, RepeatScout)
- RepeatModeler2 - a pipeline that combines different approaches and software, such as RepeatScout and RECON

# Classification of TEs

| Approach | Target TEs | Software (reference collection) |
|---|---|---|
| **Similarity-based** | General | Censor (RepBase, custom library) |
| | General | Repeat Masker (RepBase, Dfam, custom library) |
| **Signature-based** | LTR transposons | LTR_STRUC, LTR_par, find_LTR, LTR_FINDER, LTRharvest |
| | MITEs | FINDMITE, MUST |
| | Helitrons | HelitronFinder, HelSearch |
| **Machine learning** | General | TEclass (binary classification of TE types)<br>TERL (neural networks) |

# Annotation workflow



# Structural annotation

To determine position and structure of different genomic elements:

- RNA coding genes (ncRNA genes)
  - tRNA, snRNA, lncRNA, rRNA, etc.
- Protein coding genes
- Promoters
- Long-range regulatory elements
  - enhancers, repressors/silencers. insulators

# Different approaches

## Molecular techniques

- quite laborious
- time consuming
- relatively expensive
- low rate false positive
- relatively high rate of false negatives
- comprehensive approach in large-scale projects, e.g. ENCODE

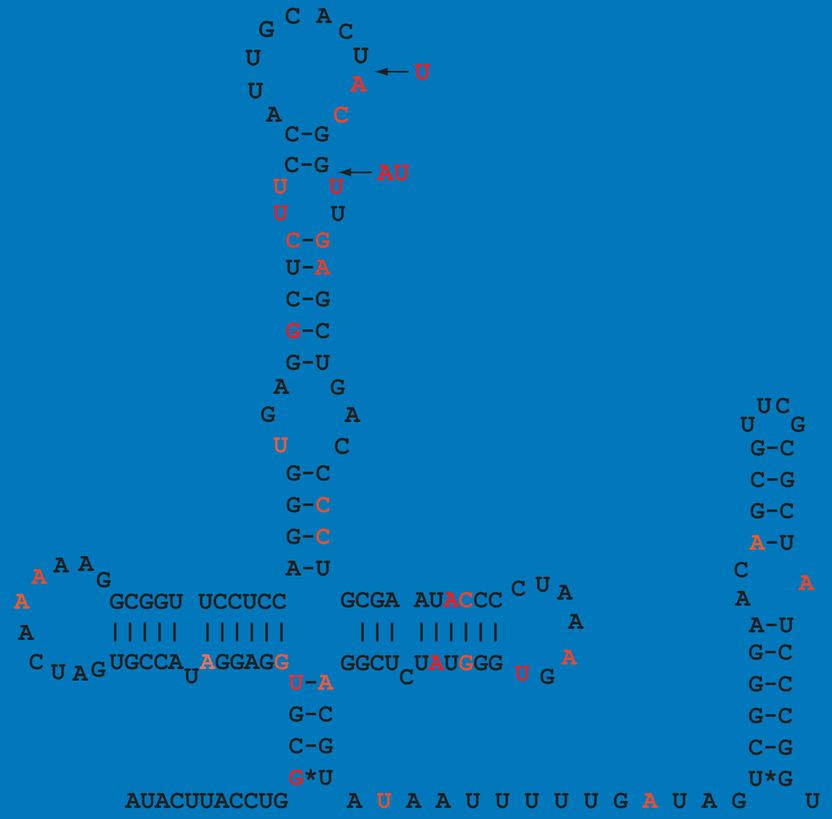
## Computational methods

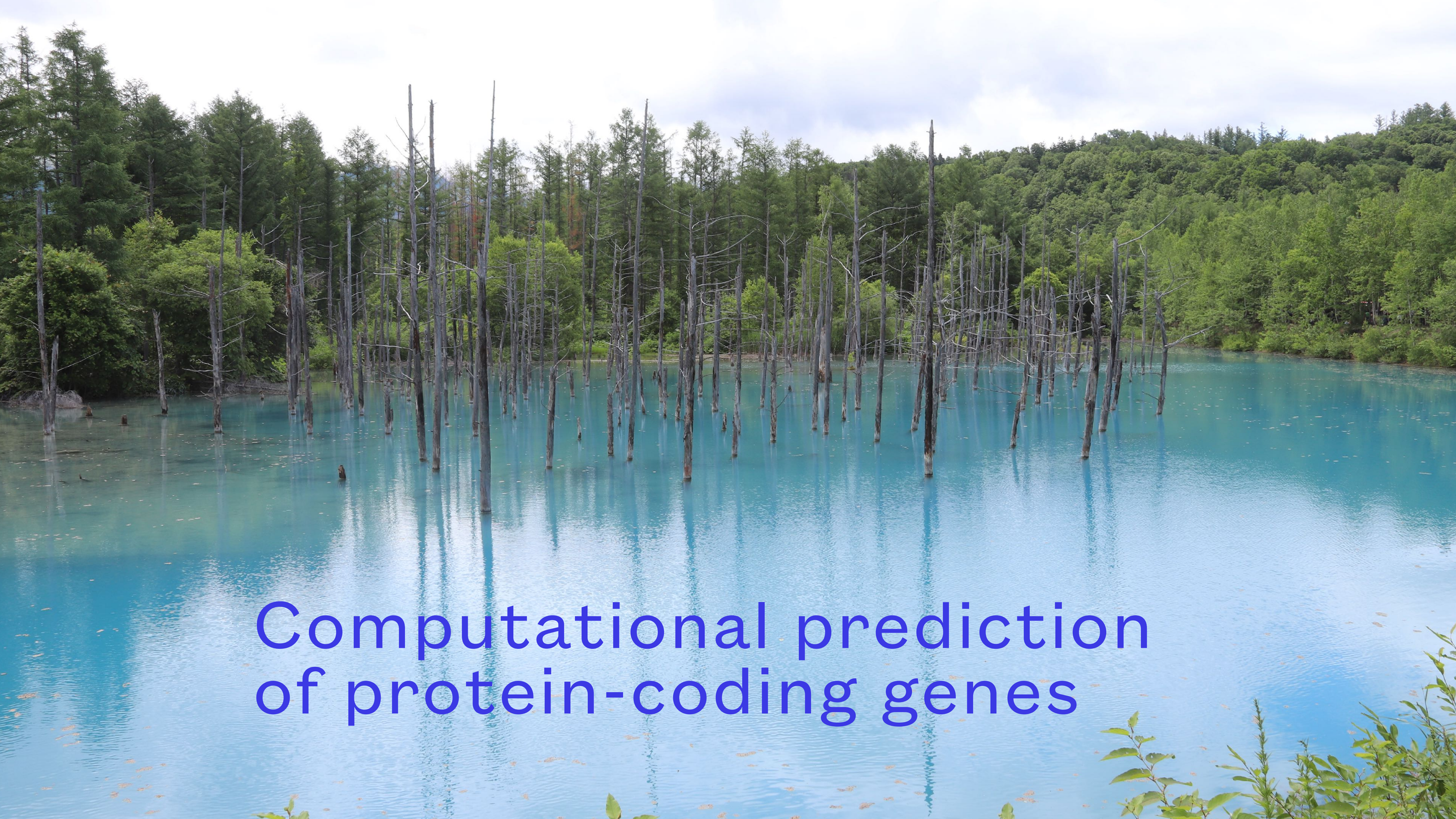
- fast
- relatively low cost
- high rate of false positives
- poor performance on less typical genes
- usually only coding sequence (CDS) can be determined

# Structural annotation

RNA coding genes (ncRNA genes)

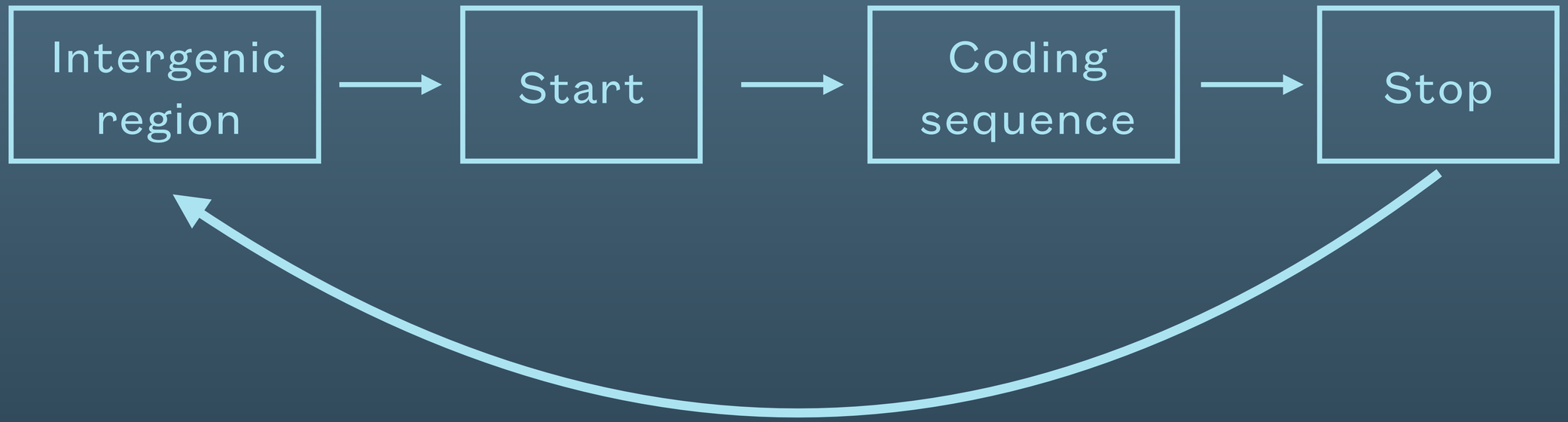
- tRNA, snRNA, rRNA, lncRNA, etc.
- usually secondary structure more conserved than nucleotide sequence
- covariance and HMM models work very well
- specialized software (tRNA-scan) or more generic, e.g infernal



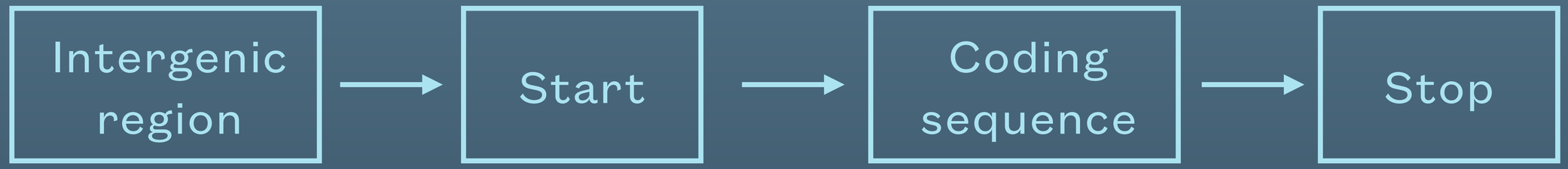


# Computational prediction of protein-coding genes

# General model of protein-coding gene



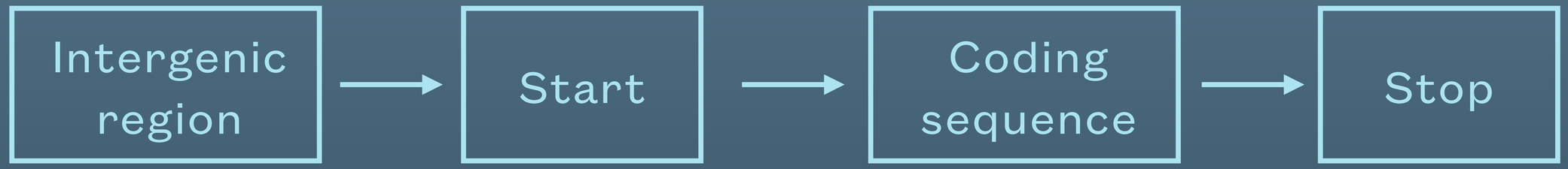
# General model of protein-coding gene



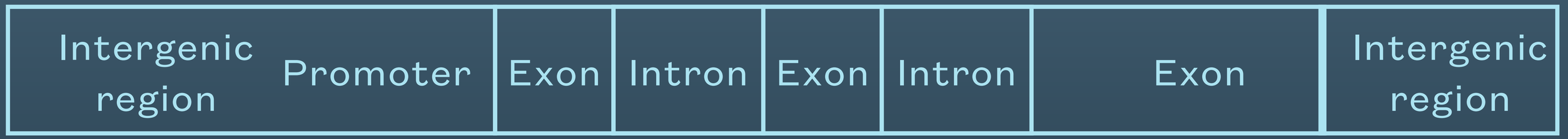

*Prokaryotic gene structure*

| | | | | | | |
|---|---|---|---|---|---|---|
| Intergenic region | Promoter | 5’ UTR | ATG | series of codons | Stop | 3’ UTR | Intergenic region |

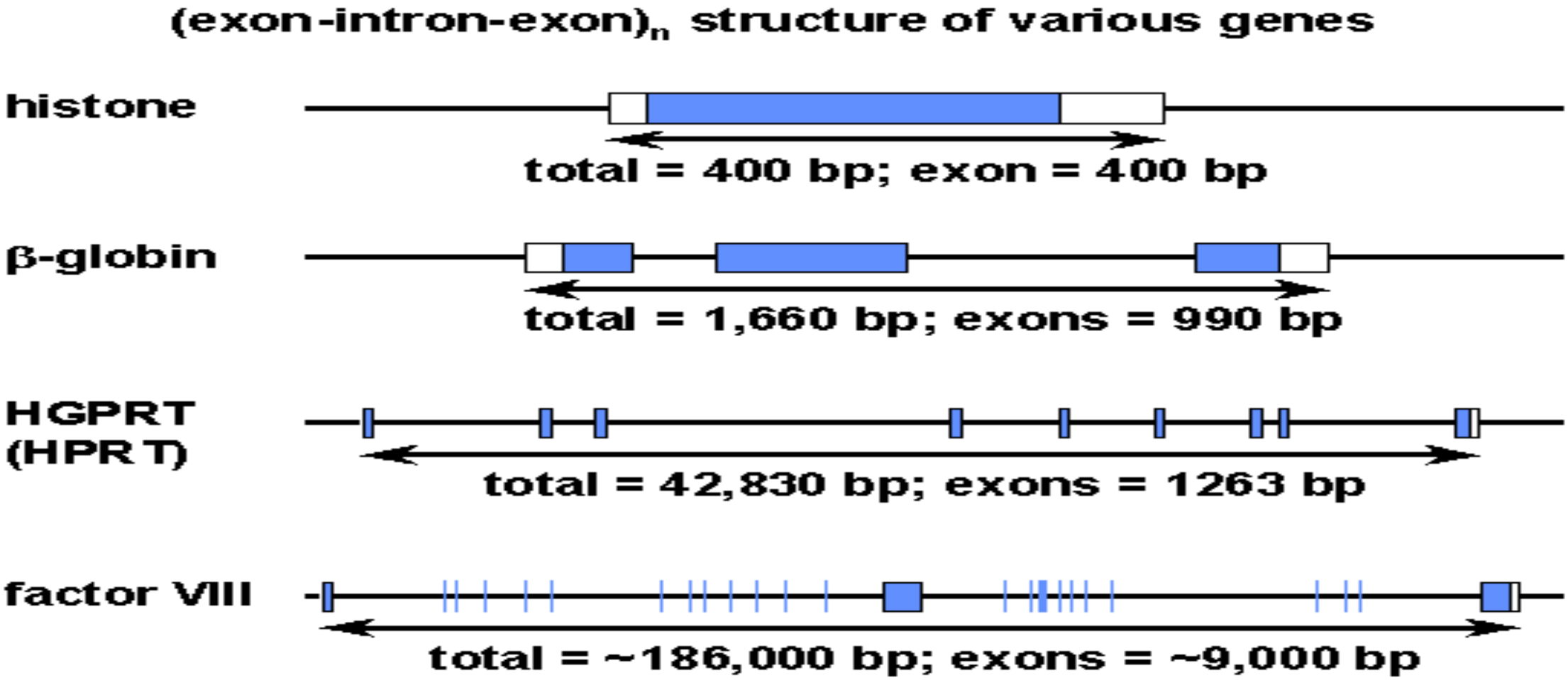
# General model of protein-coding gene



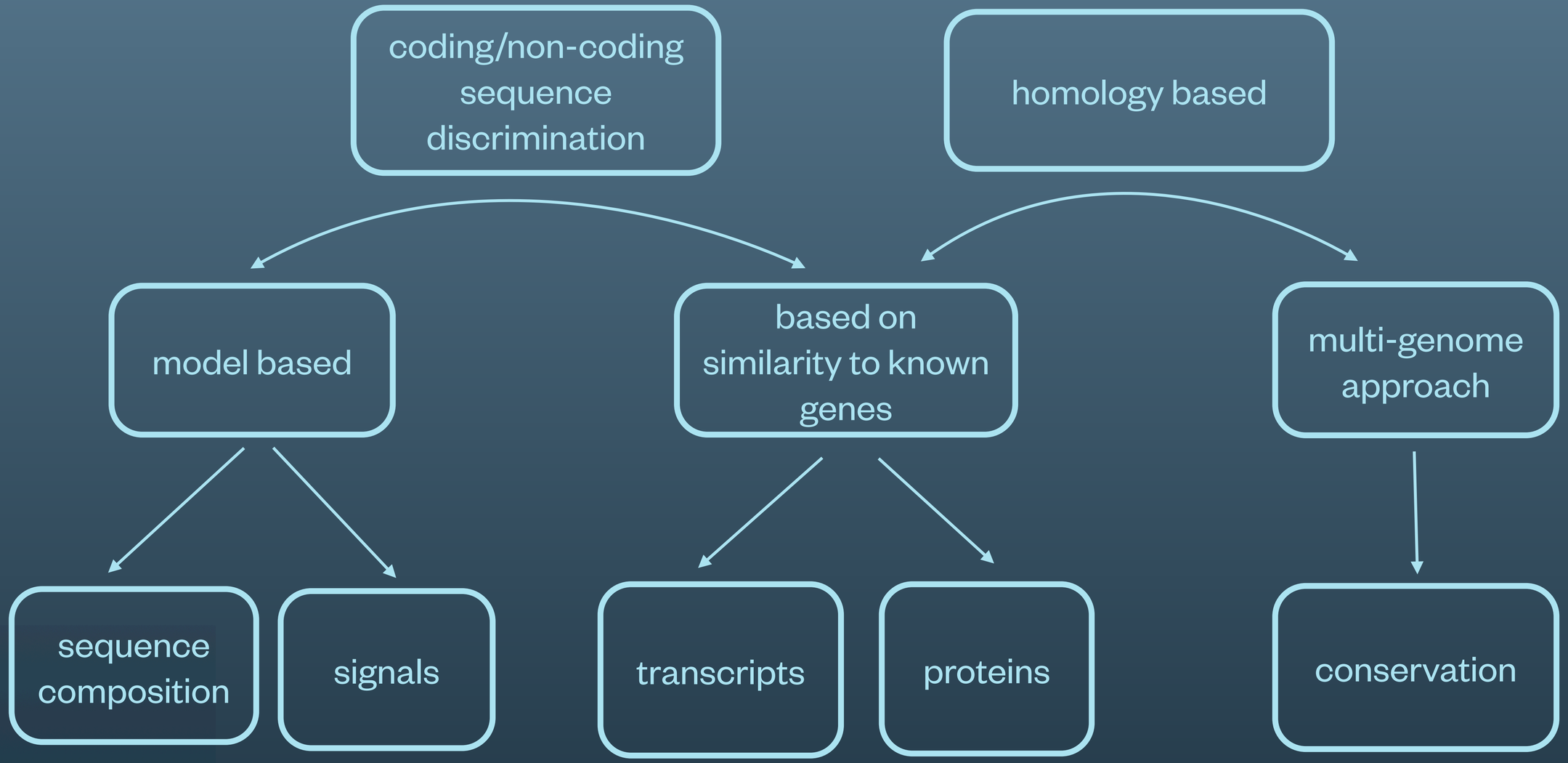

*Eukaryotic gene structure*



# Eukaryotic gene structure



# Gene finding methods



# Gene finding methods

We take advantage of what we already learned about gene structures and features of coding sequences. Based on this knowledge we can build theoretical model, develop an algorithm to search for important features, train it on known data and use to search for coding sequences in anonymous genomic fragments.

*However, we should remember that all models are wrong, but some are useful.*

*George E. P. Box*

# How to build the model

Basically, we can only discriminate between coding and non-coding sequences.

> We can check if sequence in particular ORF has some other features, which could tell us if this is a putative coding sequence or the ORF is false positive. We can look at the sequence content and compare it with known coding sequence and non-coding sequence and check to which of these two the ORF sequence is more similar to.

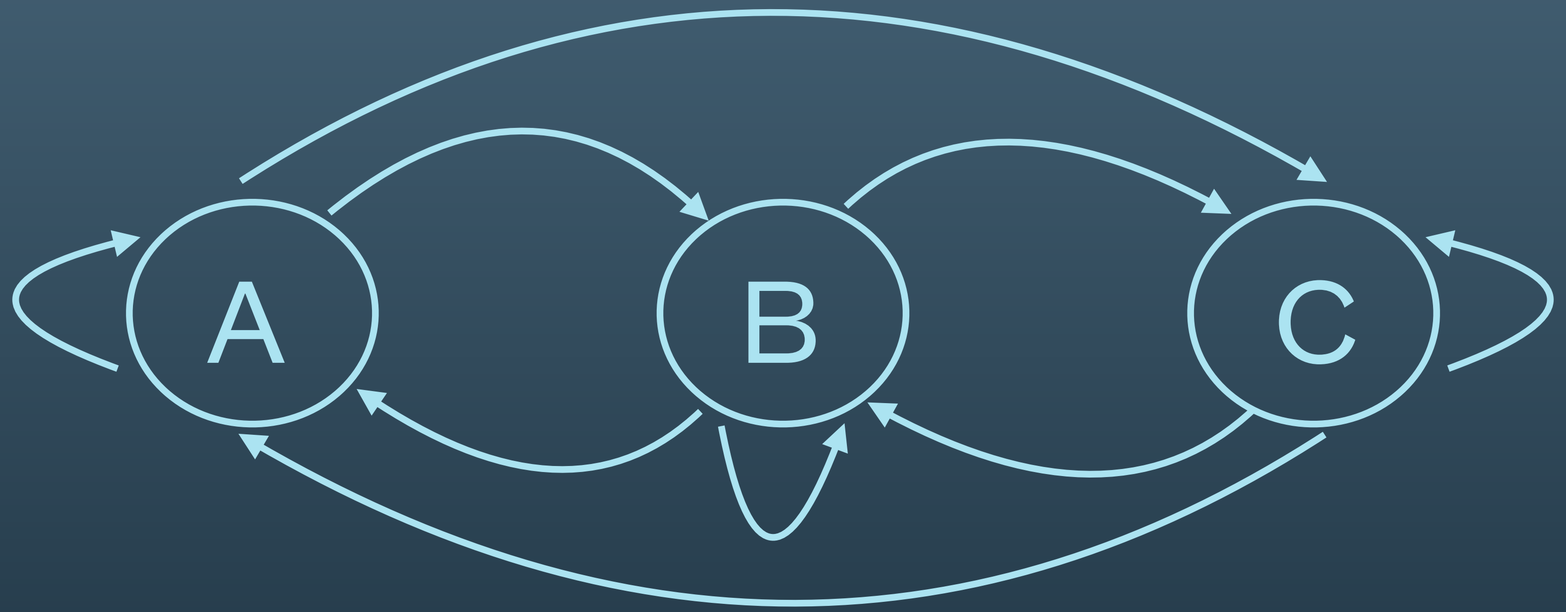
We can also model some regulatory signals, such as promoters, transcription binding sites, splicing signals, etc.

# Hidden Markov Models

- HHM is a statistical model for an ordered sequence of symbols, acting as a stochastic state machine that generates a symbol each time a transition is made from one state to the next. Transitions between states are specified by transition probabilities. A Markov process is a process that moves from state to state depending on the previous n states.
- HHM has been previously used very successfully for speech recognition.
- In biology it is used to produce multiple sequence alignments, in generating sequence profiles, to analyze sequence composition and patterns, to produce a protein structure prediction, and to locate genes.
- In gene identification HMM is a model of periodic patterns in a sequence, representing, for example, patterns found in the coding parts of a gene. HMM provides a measure of how close the data pattern in the sequence resemble the data used to train the model.

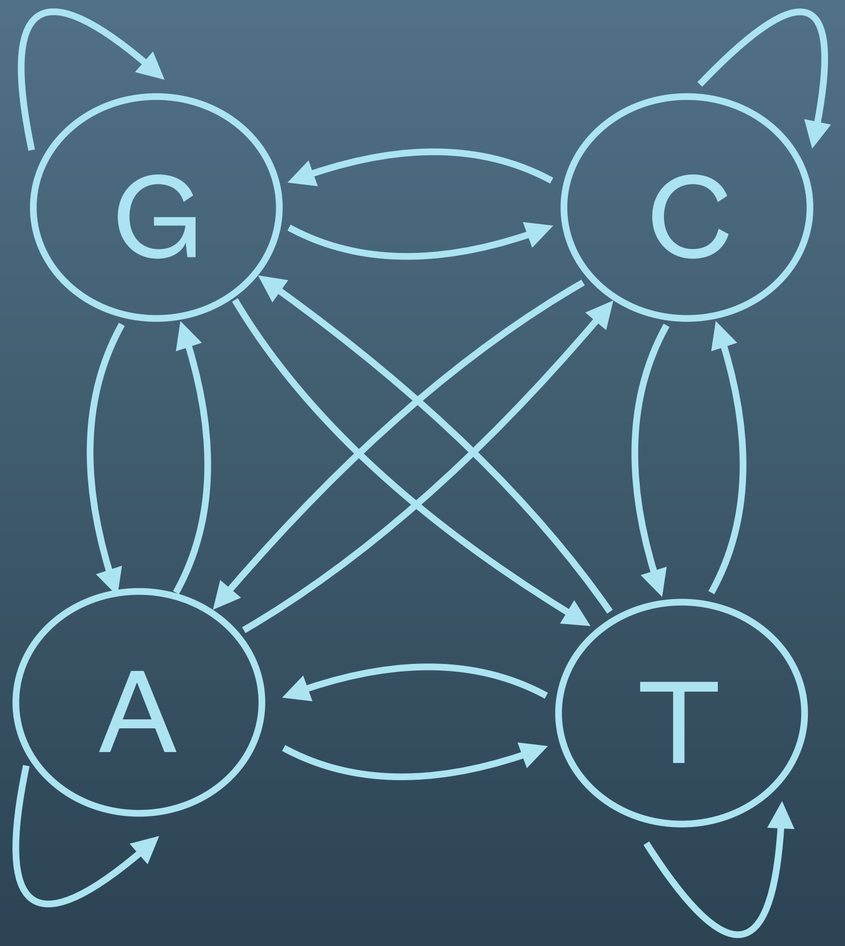
# Markow chains

A Markov Chain is a non-deterministic system in which it is assumed that the probability of moving from one state to another doesn’t vary with time. This means the current state and transition does not depend on what happened in the past. The Markov Chain is defined by probabilities for each occurring transition.



# Markow chains

In a sequence analysis we look at probabilities of transitions from one nucleotide to another. We can check, for example, if certain patterns of transition are more frequent in coding sequences than in non coding sequences.



# Order of Markov chains

GCGCTAGCGCCGATCATCTACTCG

Zero order - the current nucleotide is totally independent of the previous nucleotide.

For example, a probability of “G” in a given sequence.

# Order of Markov chains

GCGCTAGCGCCGATCATCTACTCG

GCGCTAGCGCCGATCATCTACTCG

First order - the current nucleotide only depends on the previous nucleotide.

For example, a probability of having “G” in the sequence if the previous nucleotide is “A”.

# Order of Markov chains

GCGCTAGCGCCGATCATCTACTCG

GCGCTAGCGCCGATCATCTACTCG

GCGCTAGCGCCGATCATCTACTCG

Second order - the current nucleotide depends on the previous two nucleotides.

For example, a probability of having “G” in the sequence if the previous nucleotides are “TA”.

# Order of Markov chains

GCGCTAGCGCCGATCATCTACTCG

GCGCTAGCGCCGATCATCTACTCG

GCGCTAGCGCCGATCATCTACTCG

GCGCTAGCGCCGATCATCTACTCG

Fifth order - the current nucleotide depends on the previous five nucleotides.

For example, a probability of having “G” in the sequence if the previous nucleotides are “CGCTA”.

# How far we can go?

- Order of our model will have influence on specificity and sensitivity of our program.
- Too short sequences may not be specific enough and program may return a lot of false positives.
- Long chains may be too specific and our program will not be sensitive enough returning false negatives.

# Order of Markov chains

GCGCTAGCGCCGATCATCTACTCG

GCGCTAGCGCCGATCATCTACTCG

First order - the current nucleotide only depends on the previous nucleotide (a probability of having “G” in the sequence if the previous nucleotide is “A”)

In our example the sequence is 24 nt long.
For “G” we would have the following probability matrix:

p(G,A) = 1/23 = 0.043
p(G,T) = 0/23 = 0
p(G,C) = 4/23 = 0.174
p(G,G) = 0/23 = 0

# Probability matrix

Number of probabilities in a DNA matrix of a given order can be calculated according to the following formula:

$$4k+1$$

where 4 represents number of letters in the DNA alphabet and $k$ stands for the order number.

Hence, first order Markov Model matrix consists of $4^2 = 16$ probabilities

p(A/A), p(A/T), p(A/C), p(A/G),
p(T/A), p(T/T), p(T/C), p(T/G),
p(C/A), p(C/T), p(C/C), p(C/G),
p(G/A), p(G/T), p(G/C), p(G/G)

# Probability matrix

Frequencies of transitions may depend on in which codon position (1st, 2nd, or 3rd) is a given nucleotide (state). This increases number of probabilities to be calculated. For first order Markov chains it would be:

$$3\,(4^{1+1}) = 3 \times 4^2 = 48$$

Codon position 1

| | A | C | G | T |
|---|---|---|---|---|
| A | .36 | .27 | .35 | .18 |
| C | .21 | .23 | .24 | .27 |
| G | .19 | .14 | .23 | .23 |
| T | .24 | .35 | .19 | .31 |

Codon position 2

| | A | C | G | T |
|---|---|---|---|---|
| A | .16 | .19 | .15 | .07 |
| C | .28 | .44 | .41 | .33 |
| G | .40 | .12 | .27 | .45 |
| T | .16 | .25 | .17 | .16 |

Codon position 3

| | A | C | G | T |
|---|---|---|---|---|
| A | .22 | .33 | .24 | .13 |
| C | .21 | .29 | .27 | .21 |
| G | .44 | .15 | .37 | .53 |
| T | .13 | .22 | .12 | .13 |

# Calculating coding potential of a sequence

To estimate if the sequence (S) is coding we have to calculate probability that sequence is coding ($P_c$) and probability the sequence is non-coding ($P_{nc}$). Next we calculate logarithm from the ratio of these two probability values.

$$LP(S) = \log \frac{P_c\,(S)}{P_{nc}\,(S)}$$

If the calculated value is > 0 the likelihood that the sequence is coding is higher than the sequence is not coding, if value is < 0 there is higher likelihood that sequence is not coding.

# Coding versus non-coding sequence

| A/A | C/A | G/A | T/A | |
|---|---|---|---|---|
| 0.36 | 0.21 | 0.19 | 0.24 | coding |

| A/A | C/A | G/A | T/A | |
|---|---|---|---|---|
| 0.25 | 0.25 | 0.25 | 0.25 | non-coding* |

* it is common to assume that probability of each transition is equal but it would be more realistic to use nucleotide frequencies of the analyzed sequence

# Coding versus non-coding sequence

$$LP(S) = \log \frac{P_i(S)}{P_0(S)}$$

**S=AGGACG**

| Codon position 1 | A | C | G | T |
|---|---|---|---|---|
| A | .36 | .27 | .35 | .18 |
| C | .21 | .23 | .24 | .27 |
| G | .19 | .14 | .23 | .23 |
| T | .24 | .35 | .19 | .31 |

| Codon position 2 | A | C | G | T |
|---|---|---|---|---|
| A | .16 | .19 | .15 | .07 |
| C | .28 | .44 | .41 | .33 |
| G | .40 | .12 | .27 | .45 |
| T | .16 | .25 | .17 | .16 |

| Codon position 3 | A | C | G | T |
|---|---|---|---|---|
| A | .22 | .33 | .24 | .13 |
| C | .21 | .29 | .27 | .21 |
| G | .44 | .15 | .37 | .53 |
| T | .13 | .22 | .12 | .13 |

$$P(S)^1 = f(A,1)F(G,A)^1F(G,G)^2F(A,G)^3F(C,A)^1F(G,C)^2$$

$$P(S) = 0.27 \times 0.19 \times 0.27 \times 0.24 \times 0.21 \times 0.12 = 0.00008377$$

$$P(S) = 0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25 = 0.0002441$$

$$LP(S) = \log(0.00008377/0.0002441) = -0.4644$$

* in the case of the first position in the analyzed sequence we put the frequency of a particular letter in the analyzed genome

# Coding versus non-coding sequence

$$LP(S) = \log \frac{P^{i}(S)}{P_{0}(S)}$$

| Codon position 1 | A | C | G | T |
|---|---|---|---|---|
| A | .36 | .27 | .35 | .18 |
| C | .21 | .23 | .24 | .27 |
| G | .19 | .14 | .23 | .23 |
| T | .24 | .35 | .19 | .31 |

| Codon position 2 | A | C | G | T |
|---|---|---|---|---|
| A | .16 | .19 | .15 | .07 |
| C | .28 | .44 | .41 | .33 |
| G | .40 | .12 | .27 | .45 |
| T | .16 | .25 | .17 | .16 |

| Codon position 3 | A | C | G | T |
|---|---|---|---|---|
| A | .22 | .33 | .24 | .13 |
| C | .21 | .29 | .27 | .21 |
| G | .44 | .15 | .37 | .53 |
| T | .13 | .22 | .12 | .13 |

**S=AGGACG**

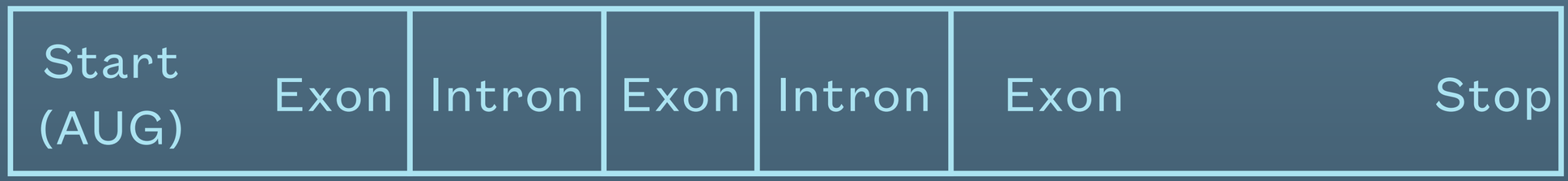
$$LP(S) = \log\frac{0.27}{0.25} + \log\frac{0.19}{0.25} + \log\frac{0.27}{0.25} + \log\frac{0.24}{0.25} + \log\frac{0.21}{0.25} + \log\frac{0.12}{0.25}$$

$$LP(S) = \log 1.08 + \log 0.76 + \log 1.08 + \log 0.96 + \log 0.84 + \log 0.48$$

$$LP(S) = 0.0334 + (-0.1191) + 0.0334 + (-0.0177) + (-0.0757) + (-0.3187)$$

$$LP(S) = -0.4644$$

# Eukaryotic model of protein-coding gene

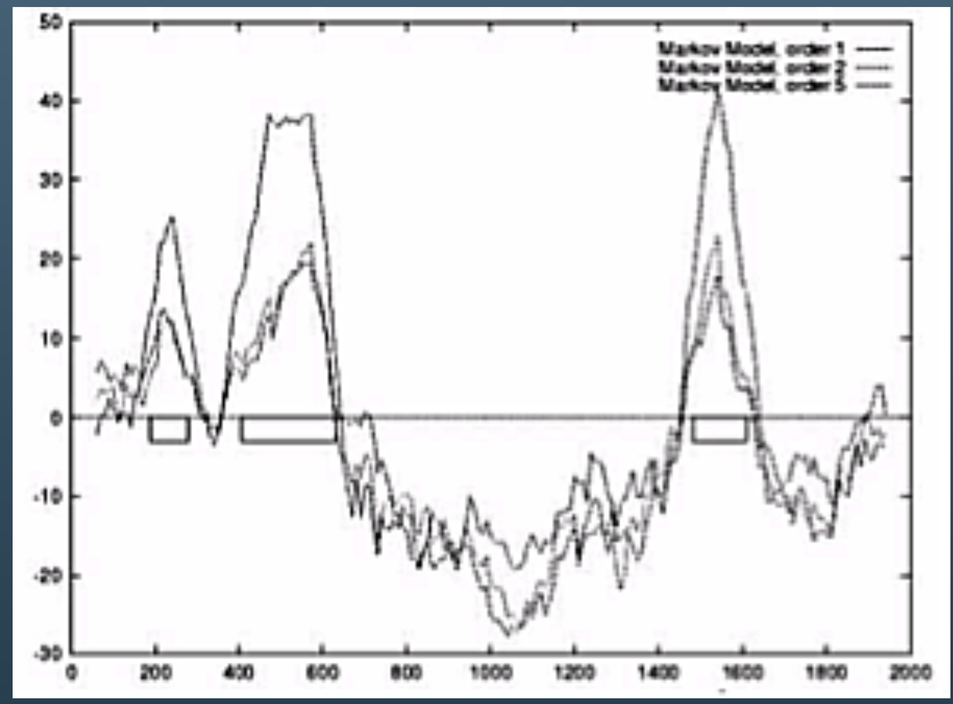



In this case we do not want check if a given sequence fragment is coding or not but we rather want to identify coding fragments in a long sequence. In most cases, this is done by calculating statistics in overlapping windows.

# Eukaryotic model of protein-coding gene

AGTACGATATTAGCGGCAATCGTATGACTACGTCTTGCTACGTCTTCTCTCGTCTGCTCTAG

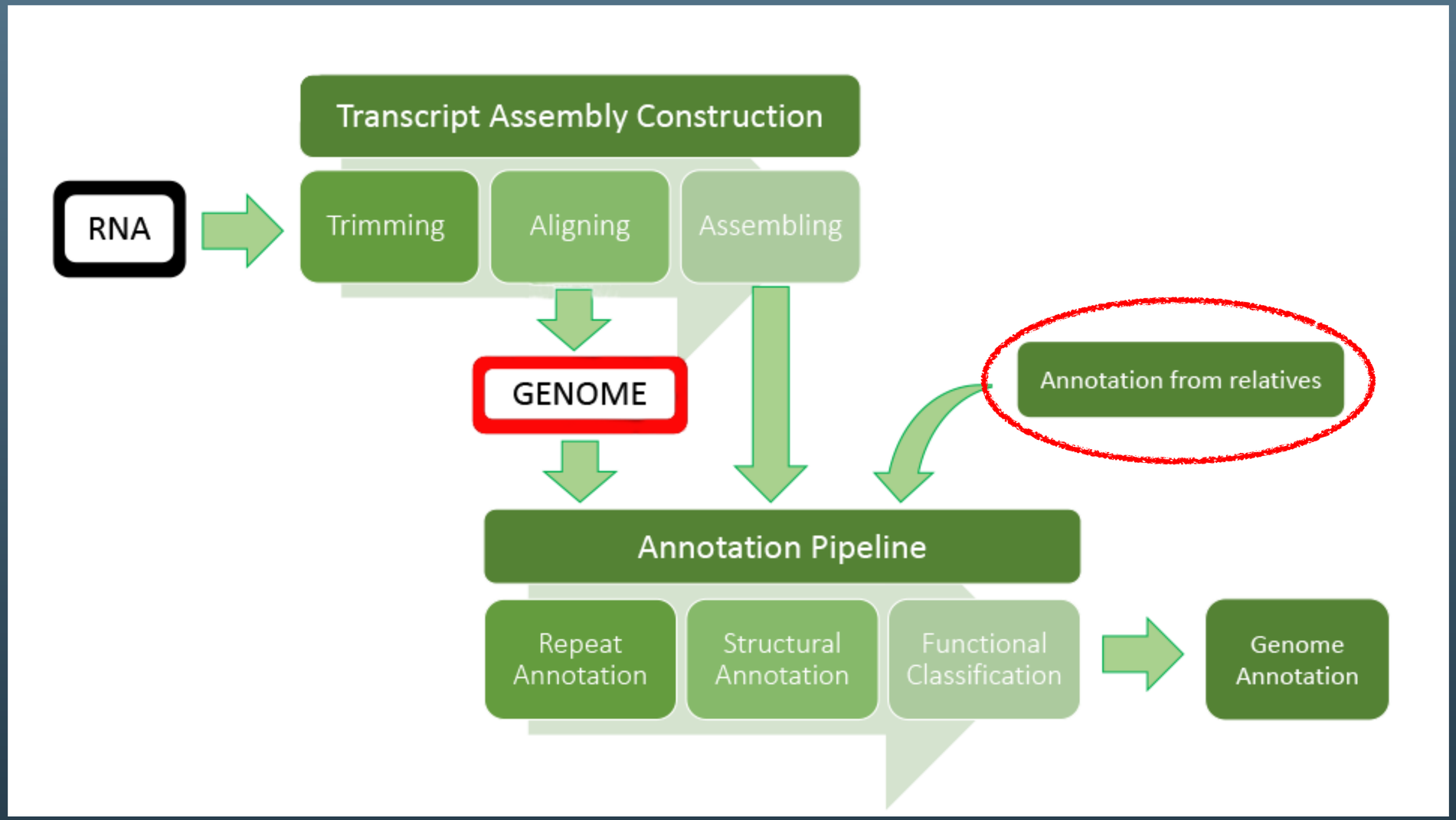
This example shows a profile for a sequence analyzed using a 120-bp window and a 10-bp step.



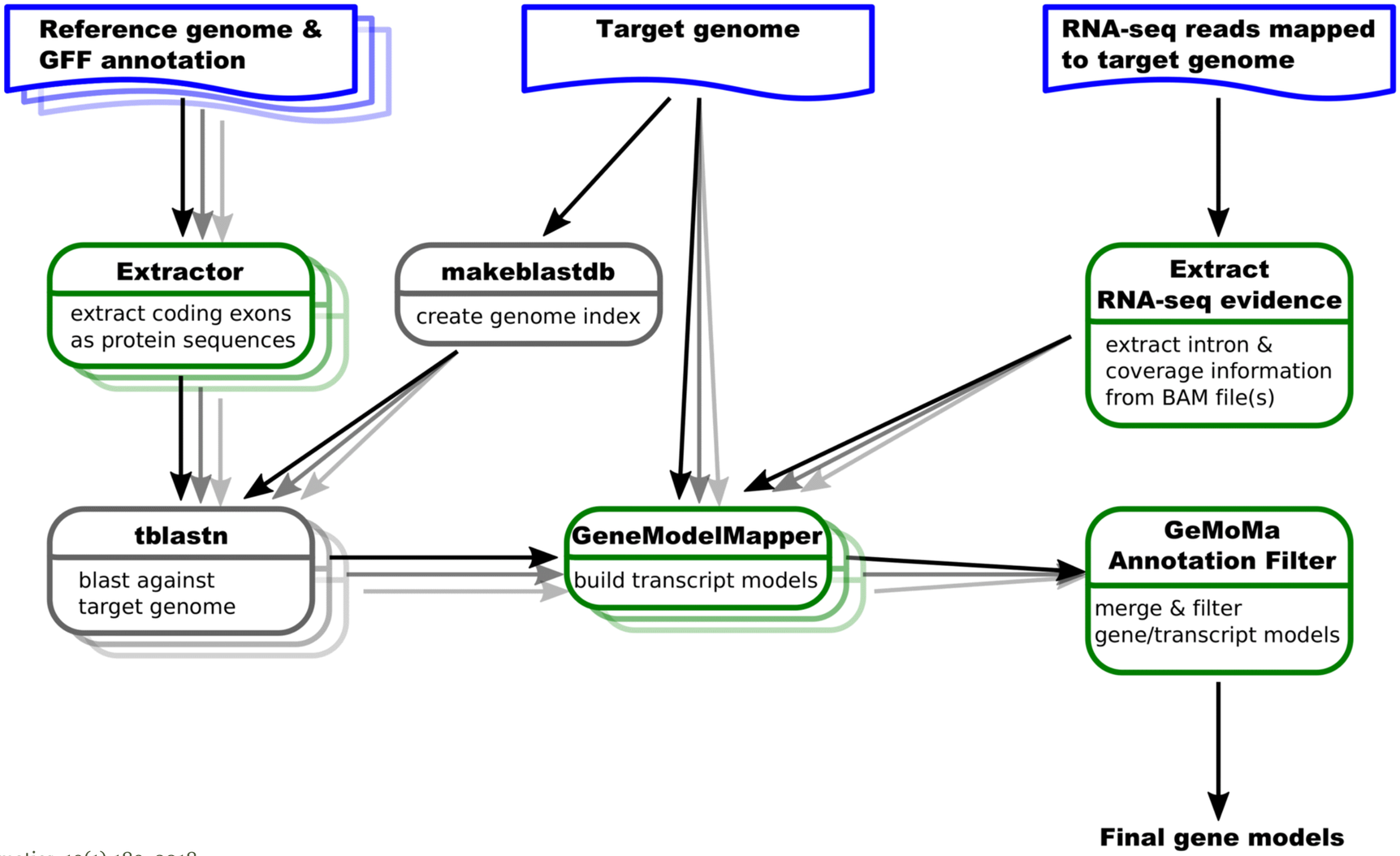
# Rule-based methods

Minimal length ORF → Splicing sites → Coding potential → Putative exons

# Annotation workflow



# Gene Model Mapper (GeMoMa)

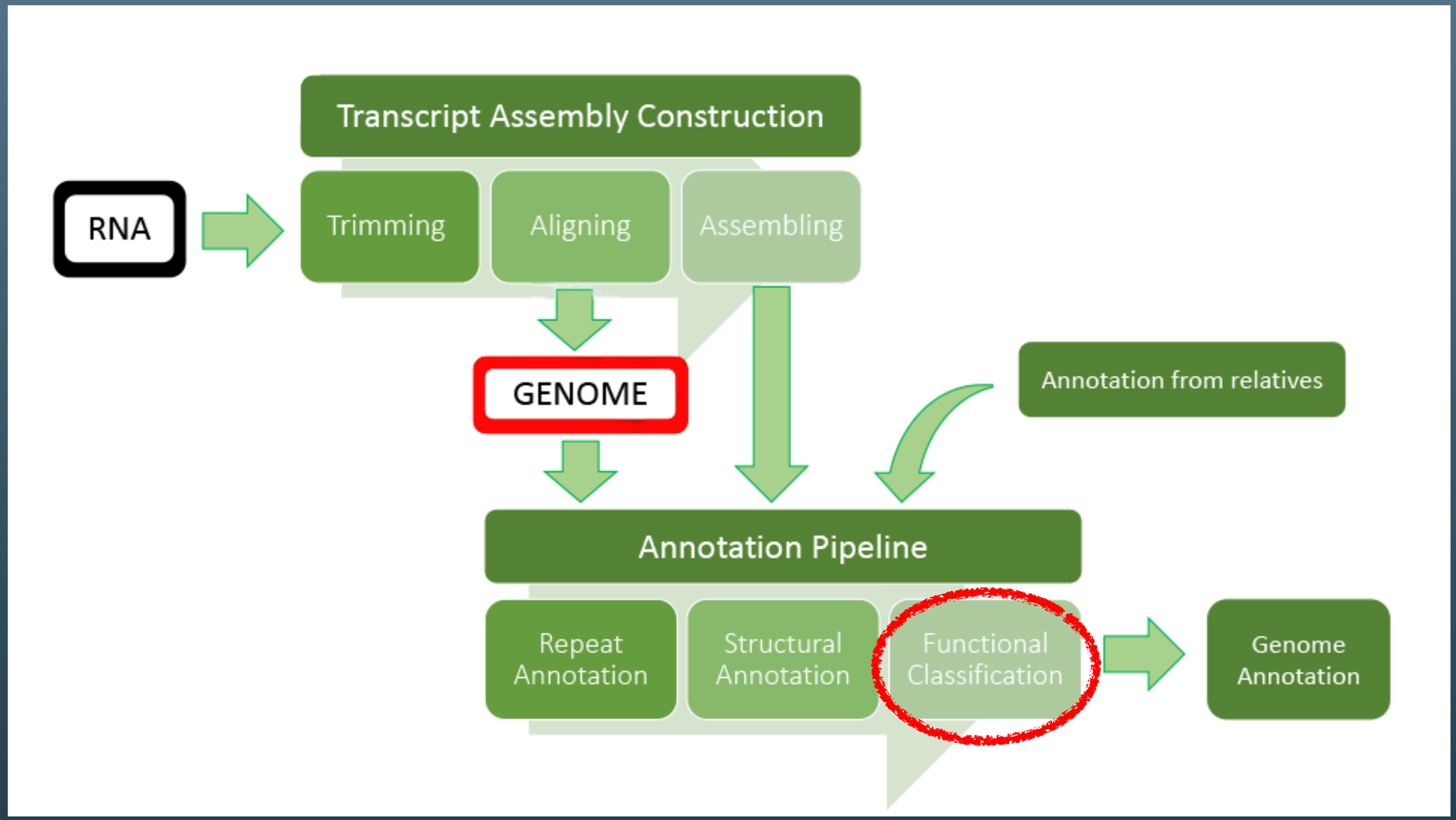



Keilwagen et al. BMC bioinformatics, 19(1):189, 2018

# GeMoMa annotation of *Pogonomyrmex californicus*

| Reference species | Number of proteins in the reference genome | Number of proteins predicted in the target genome | Number of proteins in the final annotation |
|---|---|---|---|
| ***Pogonomyrmex barbatus*** | 19,128 | 14,851 | 12,865 |
| ***Solenopsis invicta*** | 21,118 | 14,160 | 3,697 |
| ***Camponotus floridanus*** | 18,824 | 13,769 | 2,549 |
| ***Apis melifera*** | 22,451 | 10,752 | 1,095 |
| | Number of merged GeMoMa predictions | | **20,170** |

# Annotation workflow



# Functional annotation

The functional annotation of the detected genes includes protein identifications based on similarities to well annotated proteins, their molecular function (GO-annotation) and pathways they are involved in.

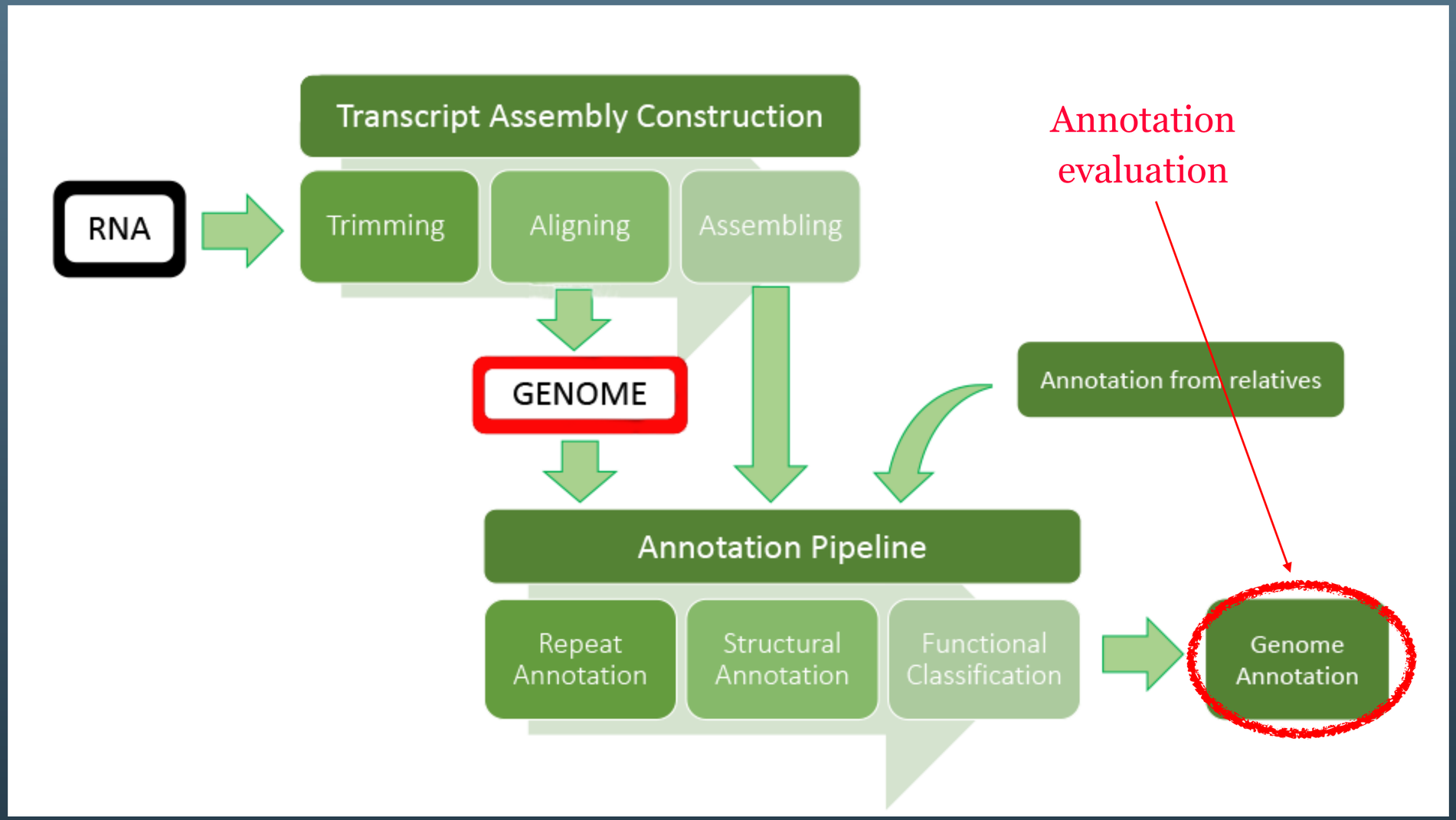
This is usually done based on a series of similarity searches against well annotated databases.

- Uniprot
- NCBI Refsec collection
- Pfam or other protein domains database
- KEGG

# Functional annotation of 27,264 of *Pogonomyrmex californicus* proteins

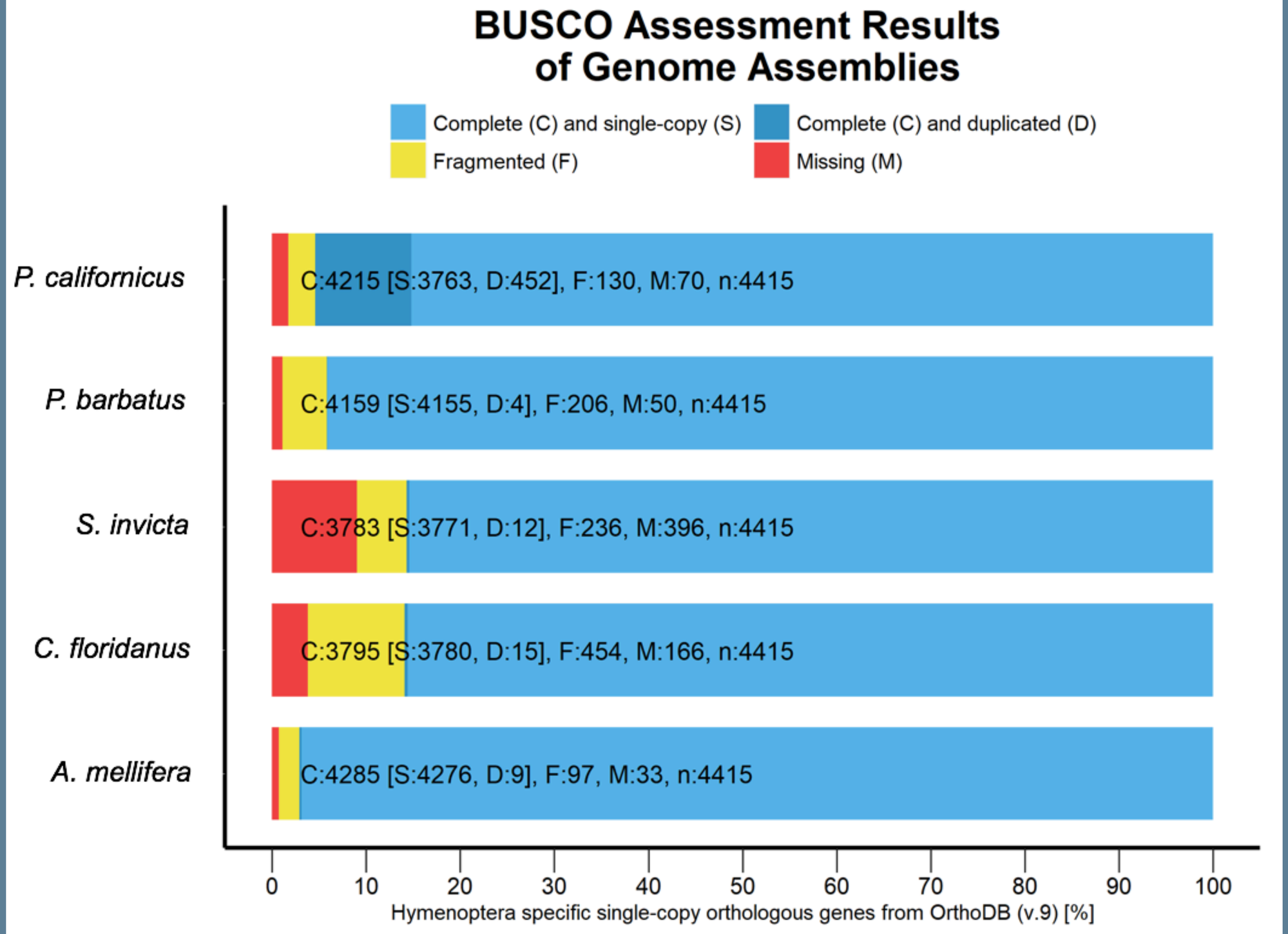
| Run | Database | DB size | Detected functions | Unknown proteins |
|---|---|---|---|---|
| **1** | Uniport | 557,992 | 12,615 | 14,649 |
| **2** | Refseq (Pogonomyrmex) | 12,578 | 1,849 | 12,800 |
| **3** | NCBI nr | 181,118,669 | 2,148 | 10,653 |
| | Final | | 16,612 | 10,653 |

# Annotation workflow



# Genome assembly and annotation evaluation

Benchmarking Universal Single-Copy Orthologs (BUSCO) is a tool for measure the completeness of genome assembly data, annotated gene sets or transcriptomes in terms of expected gene content while comparing the data to core sets of orthologous groups with genes present as single-copy orthologs.


BUSCO Assessment Results
of Genome Assemblies
Complete (C) and single-copy (S)
Complete (C) and duplicated (D)
Fragmented (F)
Missing (M)
P. californicus
C:4215 [S:3763, D:452], F:130, M:70, n:4415
P. barbatus
C:4159 [S:4155, D:4], F:206, M:50, n:4415
S. invicta
C:3783 [S:3771, D:12], F:236, M:396, n:4415
C. floridanus
C:3795 [S:3780, D:15], F:454, M:166, n:4415
A. mellifera
C:4285 [S:4276, D:9], F:97, M:33, n:4415
0
10
20
30
40
50
60
70
80
90
100
Hymenoptera specific single-copy orthologous genes from OrthoDB (v.9) [%]

# BIOINFORMATICS CREED

Remember about biology

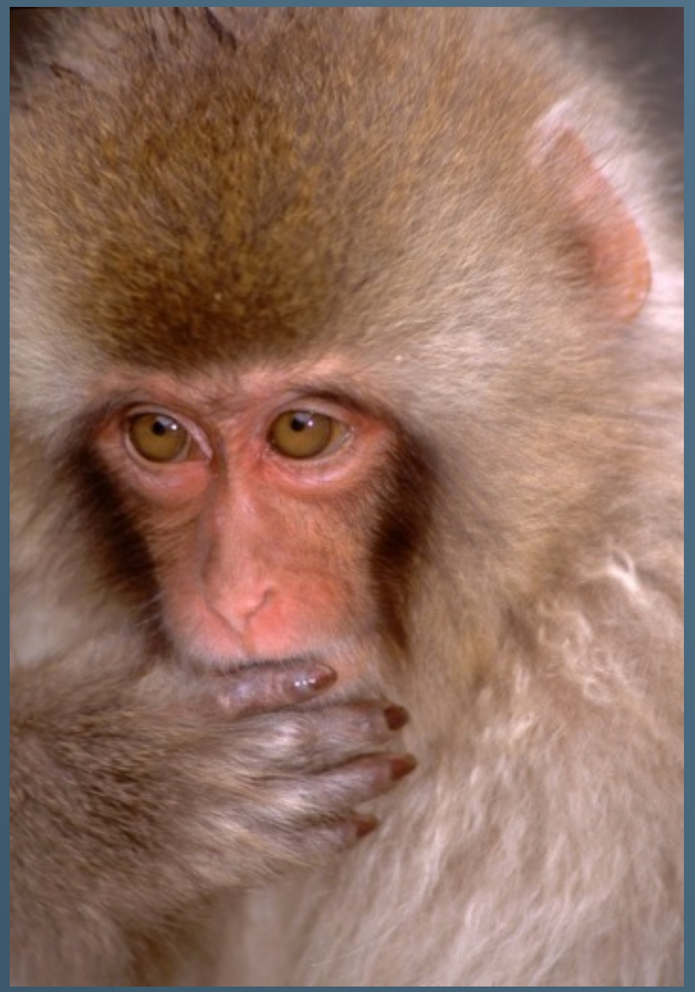
Do not trust the data

Use comparative approach

Use statistics

Know the limits

Remember about biology!!!



UKM Blutspende

In Münster für Münster

# Das liegt dir im Blut!

In der UKM Blutspende kommen alle Blutspenden unmittelbar den Patientinnen und Patienten am UKM zugute – in Münster für Münster!

Für deine Spende erhältst du eine Aufwandsentschädigung!

Domagkstraße 11, 48149 Münster
+49 251 83-58000
www.ukm-blutspende.de