Intro to Transcriptomics and Transcriptome Assembly

Programming for Biologists

CSHL 2025

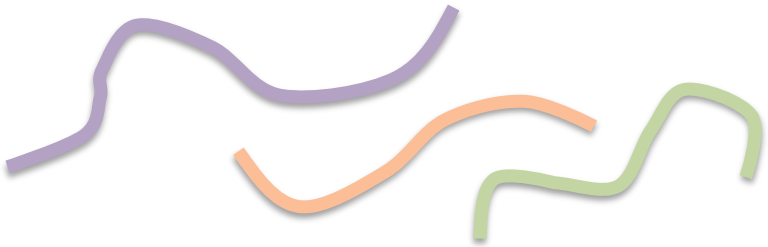
Brian Haas, Ph.D.

Principal Computational Scientist

Broad Institute

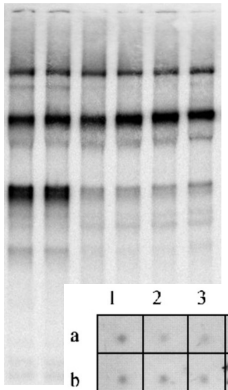
Biological Investigations Empowered by Transcriptomics

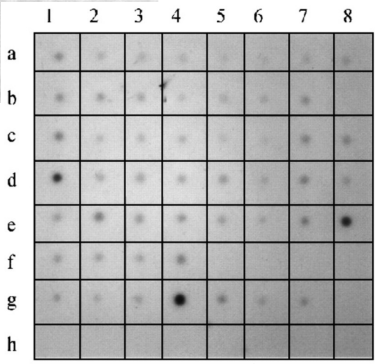
Extract RNA,
… some protocol for processing, …



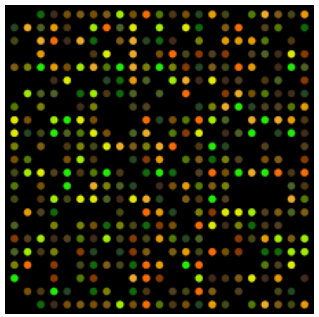
Analysis Method
(pick your favorite)

Northern

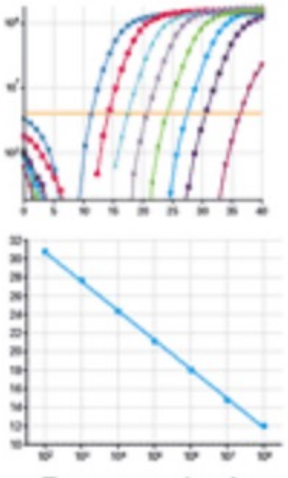




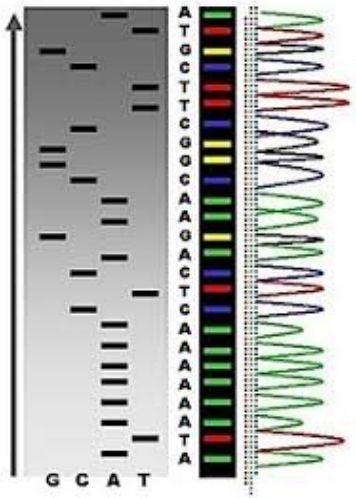
Dot Blot



Microarray



qRT-PCR



Sanger Sequencing

Other…

Minion

MinION

Historical Timeline to Modern Transcriptomics (from 1970)

Reverse Transcription (1970)

Northern Blot
Sanger Sequencing
(1977)

Expressed Sequence Tags (1992)

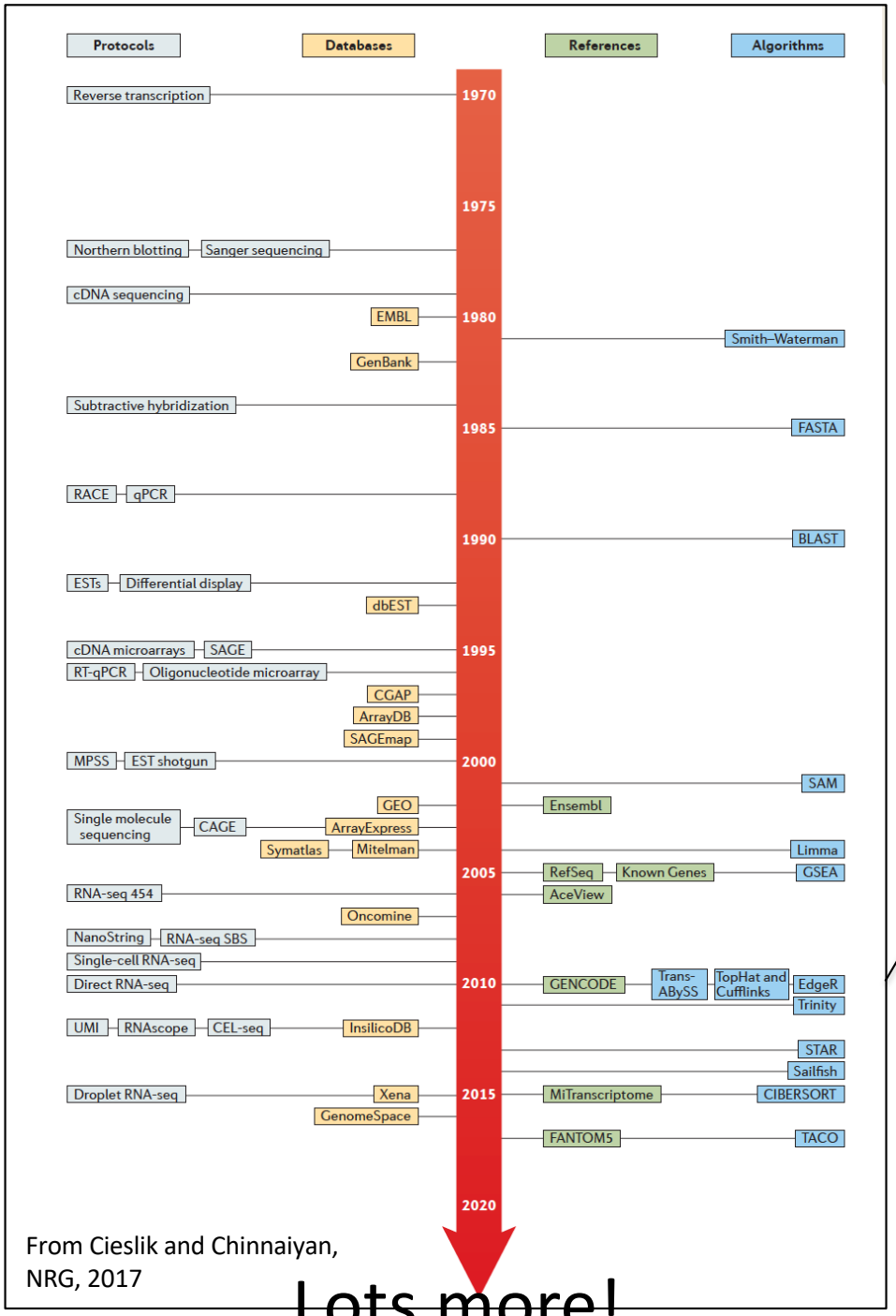
cDNA microarrays (1995)

RNA-Seq (2006-2008)

PacBio IsoSeq (2014)

Droplet single cell RNA-Seq (2015)

Direct RNA Seq Nanopore (2018)



From Cieslik and Chinnaiyan, NRG, 2017

Lots more!

Note: Just a small sampling of what's available.

Smith Waterman (1981)

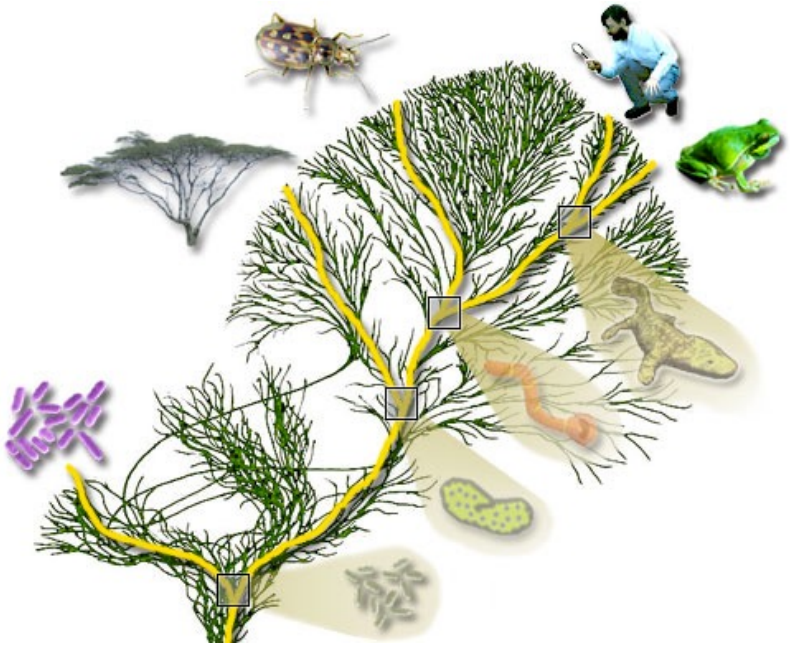
BLAST (1990)

Tophat/Cufflinks (2010)

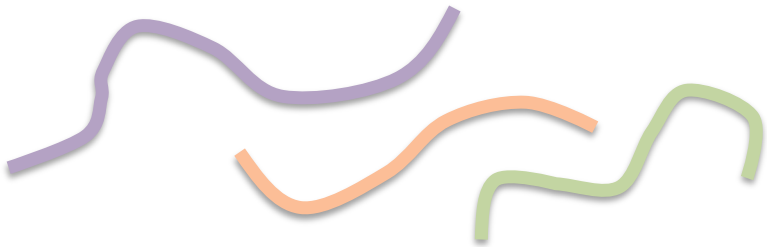
RSEM
(2011)

Kallisto (2016)
Salmon (2017)

Modern Transcriptome Studies Empowered by RNA-seq

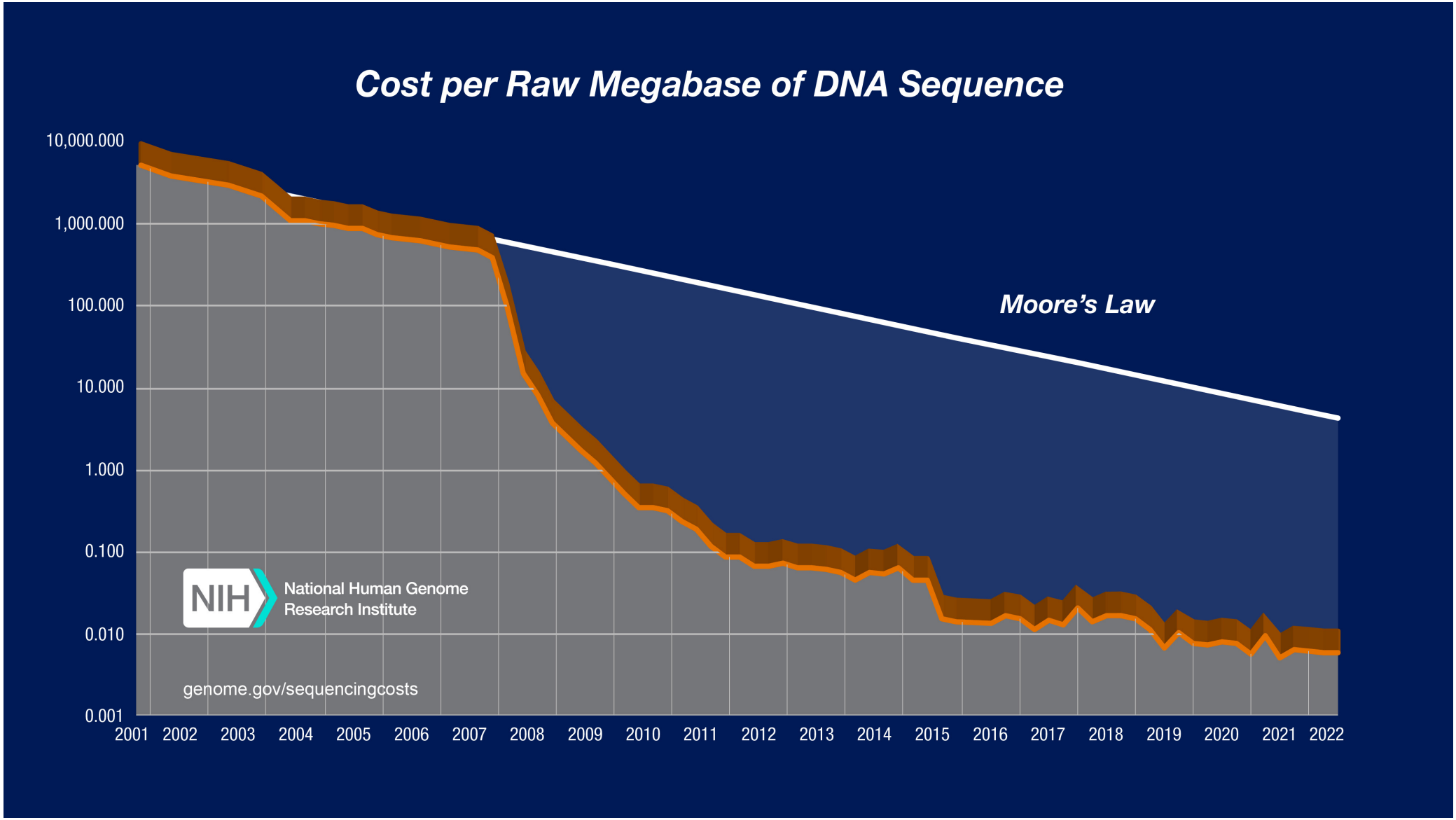


Extract RNA, convert to cDNA



Next-gen Sequencer
(pick your favorite)

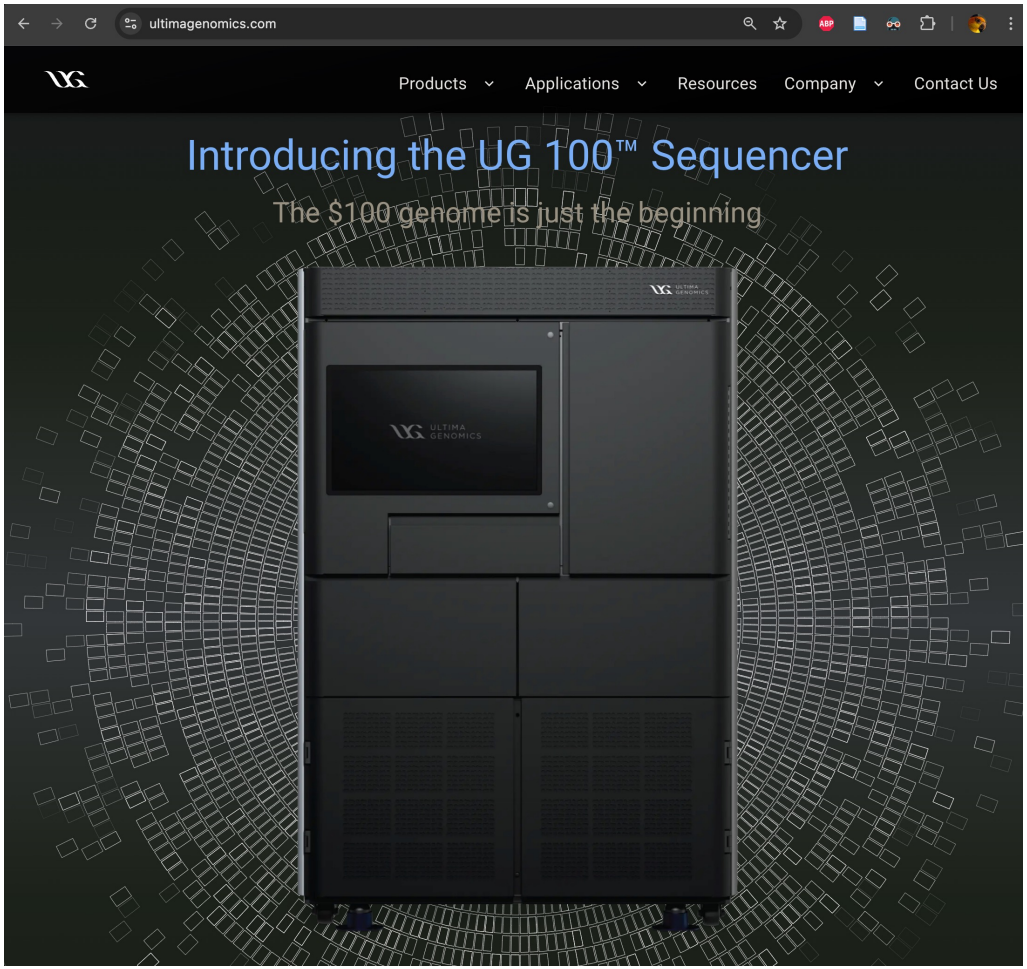
Millions to Billions of Reads



From https://www.genome.gov/sequencingcostsdata/

Ultima Genomics – high throughput, low cost genomics at scale

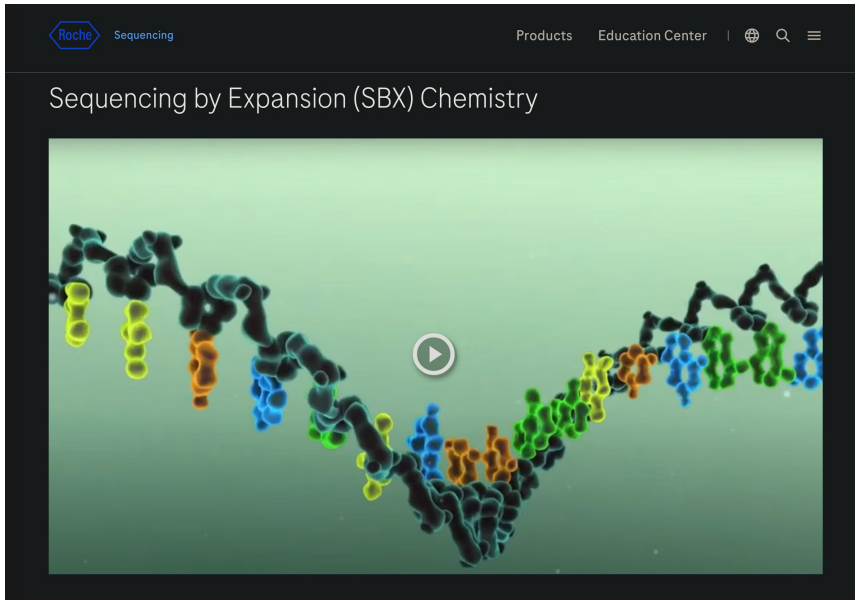
(Unveiled mid-2022)



The UG 100 sequencer can read up to 20,000 human genomes per year for $100 per genome

On the horizon: Roche SBX

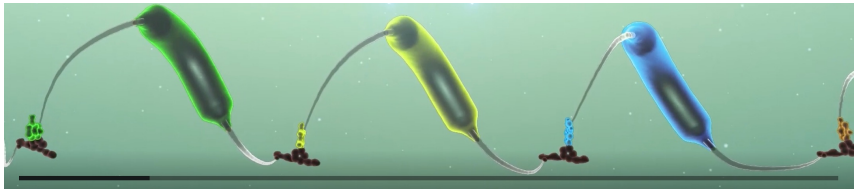
(unveiled Feb, 2025 at AGBT)

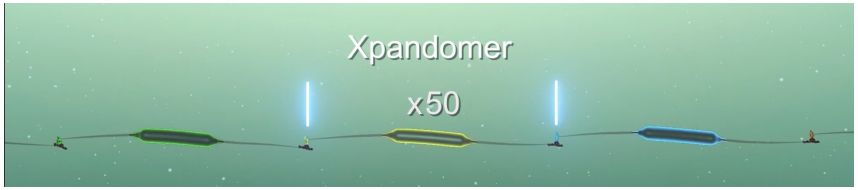


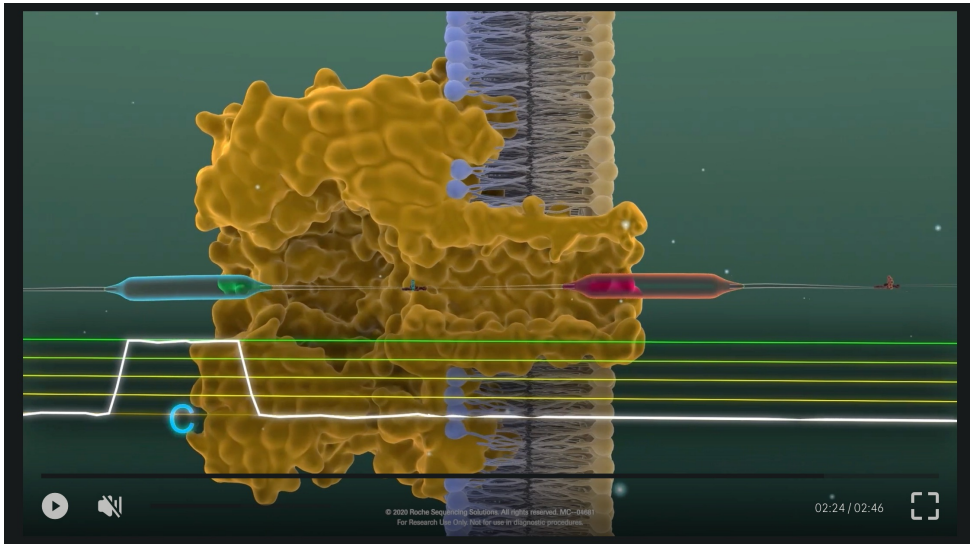
Sequencing by Expansion (SBX)

Stratos Genomics, a Roche company, has developed a novel next generation sequencing (NGS) chemistry called sequencing by expansion (SBX). This chemistry translates the sequence of DNA into a simple to measure surrogate molecule called an Xpandomer. Much like with polymerase chain reaction (PCR), Xpandomer synthesis is based on the natural function of DNA replication where expandable nucleoside triphosphates (X-NTPs) act as substrates for template-dependent, polymerase-based replication.

Four easily differentiated X-NTPs (also called High Signal-to-Noise Reporters) are used during Xpandomer synthesis, one for each DNA base, and engineered polymerases incorporate the X-NTPs into an Xpandomer, which serves as a surrogate for the complement of the nucleic acid template. As the Xpandomer molecule transits through a nanopore, the distinct electrical signal of each base reporter is easily identifiable to enable highly accurate and high throughput nanopore-based nucleic acid sequencing.



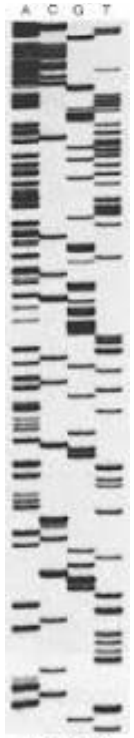


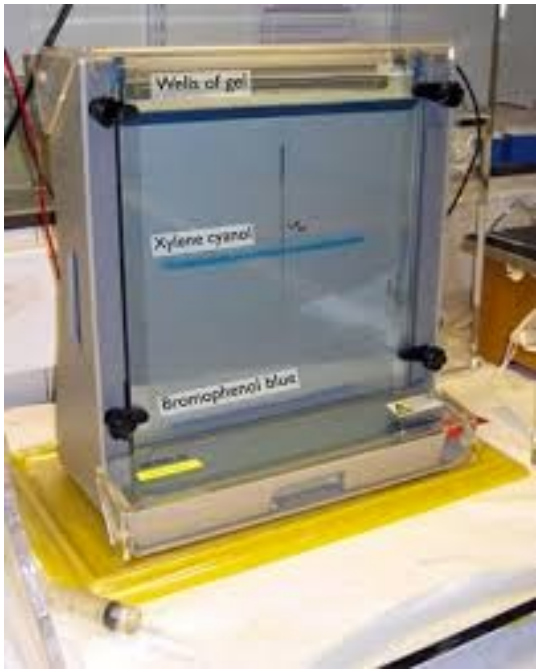


https://sequencing.roche.com/global/en/article-listing/sequencing-platform-technologies.html

Personal Reflections...

Circa 1995





Generating RNA-Seq: *How to Choose?*

Platform	iSeq Project Firefly 2018	MiniSeq	MiSeq	Next Seq 550	HiSeq 2500 RR	Hiseq 2500 V3	HiSeq 2500 V4	HiSeq 4000	HiSeq X	Nova Seq S1 2018	Nova Seq S2	Nova Seq S4	5500 XL	318 HiQ 520	Ion 530	Ion Proton P1	PGM HiQ 540	RS P6-C4	Sequel	R&D end 2018	Smidg ION RnD	Mini ION R9.5	Grid ION X5	Prome thION RnD	Prome thION theor etical	QiaGen Gene Reader	BGI SEQ 500	BGI SEQ 50	#
Reads: (M)	4	25	25	400	600	3000	4000	5000	6000	3300	6600	20000	1400	3-5	15-20	165	60-80	5.5	38.5	--	--	--	--	--	--	400	1600	1600	--
Read length: (paired-end*)	150*	150*	300*	150*	100*	100*	125*	150*	150*	150*	150*	150*	60	200 400	200 400	200	200	15K	12K	32K	--	--	--	--	--	--	100*	50	--
Run time: (d)	0.54	1	2	1.2	1.125	11	6	3.5	3	1.66	1.66	1.66	7	0.37	0.16	--	0.16	4.3	--	--	--	2	2	2	--	--	1	0.4	--
Yield: (Gb)	1	7.5	15	120	120	600	1000	1500	1800	1000	2000	6000	180	1.5	7	10	12	12	5	150	4	8	40	2400	11000	80	200	8	--
Rate: (Gb/d)	1.85	7.5	7.5	100	106.6	55	166	400	600	600	1200	3600	30	5.5	50	--	93.75	2.8	--	--	--	4	20	1200	5500	--	200	20	--
Reagents: ($K)	0.1	1.75	1	5	6.145	23.47	29.9	--	--	--	--	--	10.5	0.6	--	1	1.2	2.4	--	1	--	0.5	1.5	--	--	0.5	--	--	--
per-Gb: ($)	100	233	66	50	51.2	39.1	31.7	20.5	7.08	18	15	5.8	58.33	--	--	100	--	200	80	6.6	--	62.5	37.5	20	4.3	--	--	--	--
hg-30x: ($)	12000	28000	8000	5000	6144	4692	3804	2460	849.6	1800	1564	700	7000	--	--	12000	--	24000	9600	1000	--	7500	4500	2400	500	--	600	--	
Machine: ($)	30K	49.5K	99K	250K	740K	690K	690K	900K	1M	999K	999K	999K	595K	50K	65K	243K	242K	695K	350K	350K	--	--	125K	75K	75K	--	200K	--	--
#Page maintained by http://twitter.com/albertvilella http://tinyurl.com/ngslytics #Editable version: http://tinyurl.com/ngsspecsshared																													
#curl "https://docs.google.com/spreadsheets/d/1GMMfhyLK0-q8Xklo3YxlWaZA5vVMuhU1kg41g4xLkXc/export?gid=4&format=csv" | grep -v '^#' | grep -v '^"' | column -t -s\, | less -S																													

Stats circa 2018

Note, stats may be outdated but demonstrates diversity and considerations

From: https://tinyurl.com/wbgcs65

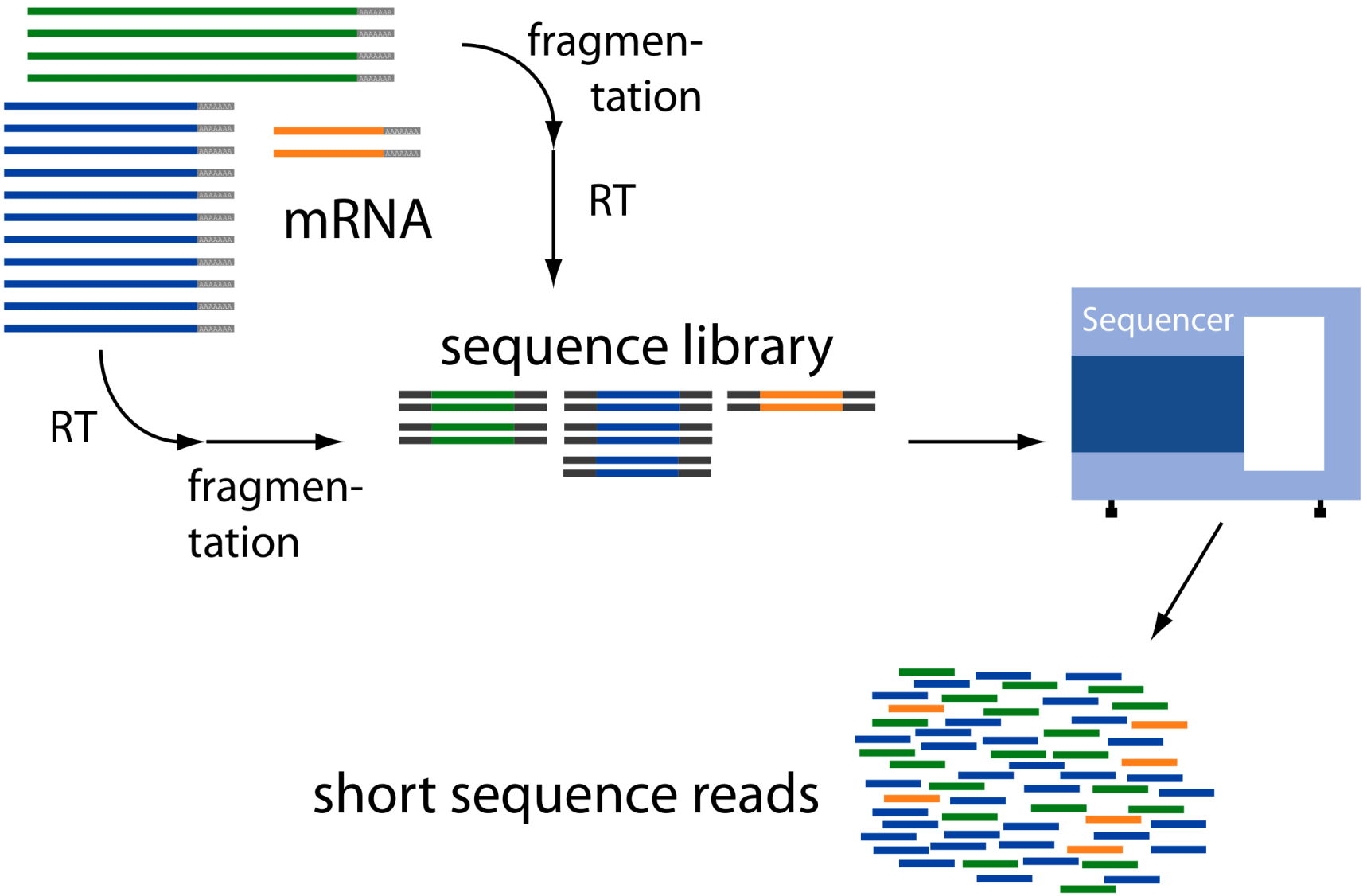
*Not all shown at scale

Read length? **Base accuracy?** **Paired or single-end reads?** **Single molecule?** **$?**

Each has pros/cons

The OG PacBio Sequencer
(way smaller now)

Overview of RNA-Seq



From: http://www2.fml.tuebingen.mpg.de/raetsch/members/research/transcriptomics.html

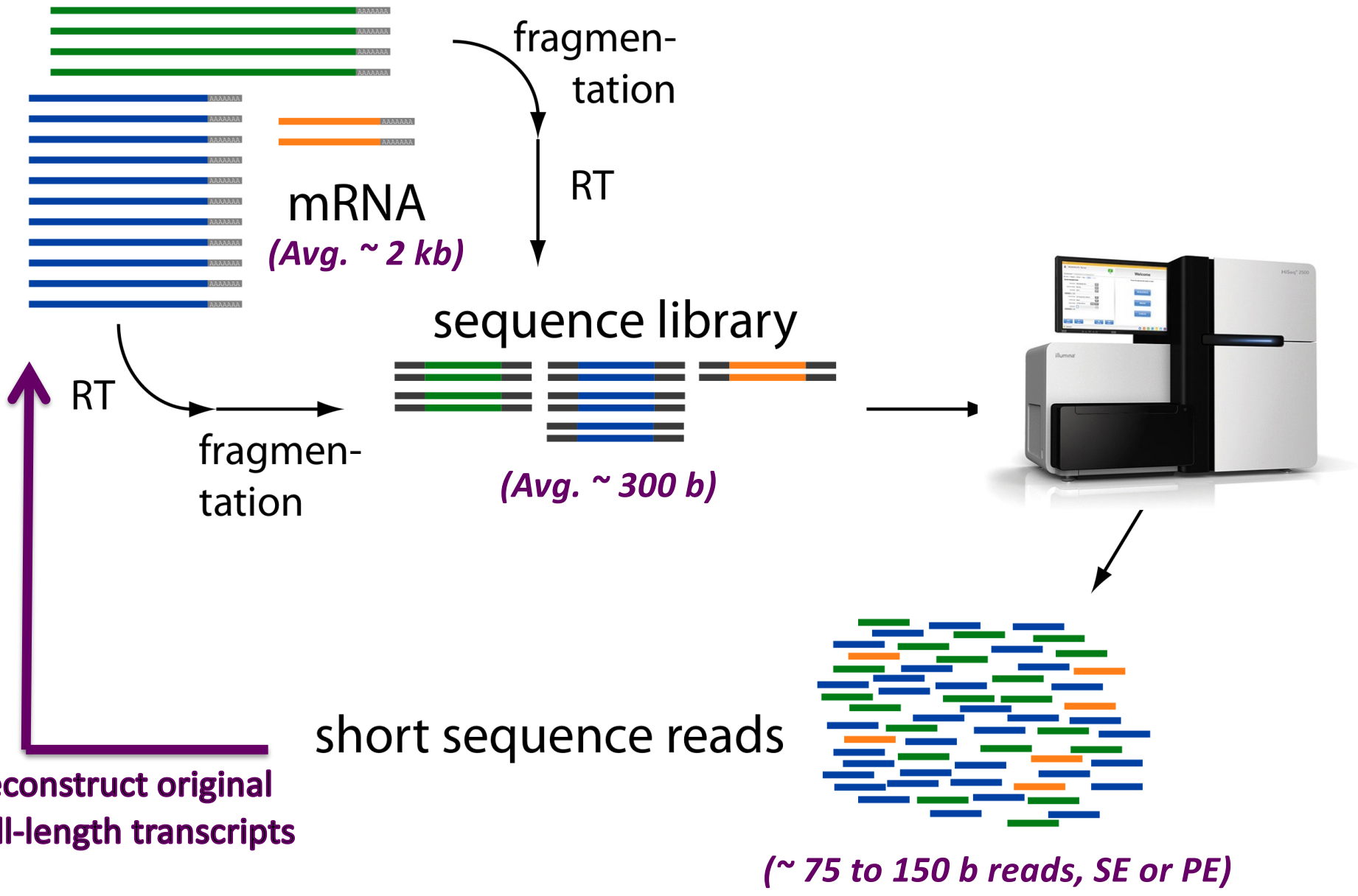
Paired-end Sequences

Two FastQ files, read name indicates left (/1) or right (/2) read of paired-end

```
@61DFRAAXX100204:1:100:10494:3070/1
AAACAACAGGGCACATTGTCACTCTTGTATTTGAAAAACACTTTCCGGCCAT
+
ACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBC?CCCCCCCCC@@CACCCCCA
```

```
@61DFRAAXX100204:1:100:10494:3070/2
CTCAAATGGTTAATTCTCAGGCTGCAAATATTCGTTCAGGATGGAAGAACA
+
C<CCCCCCCACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBCCCC
```

RNA-Seq Challenge: Transcript Reconstruction



Adapted from: http://www2.fml.tuebingen.mpg.de/raetsch/members/research/transcriptomics.html

Transcript Reconstruction from RNA-Seq Reads

Advancing RNA-Seq analysis

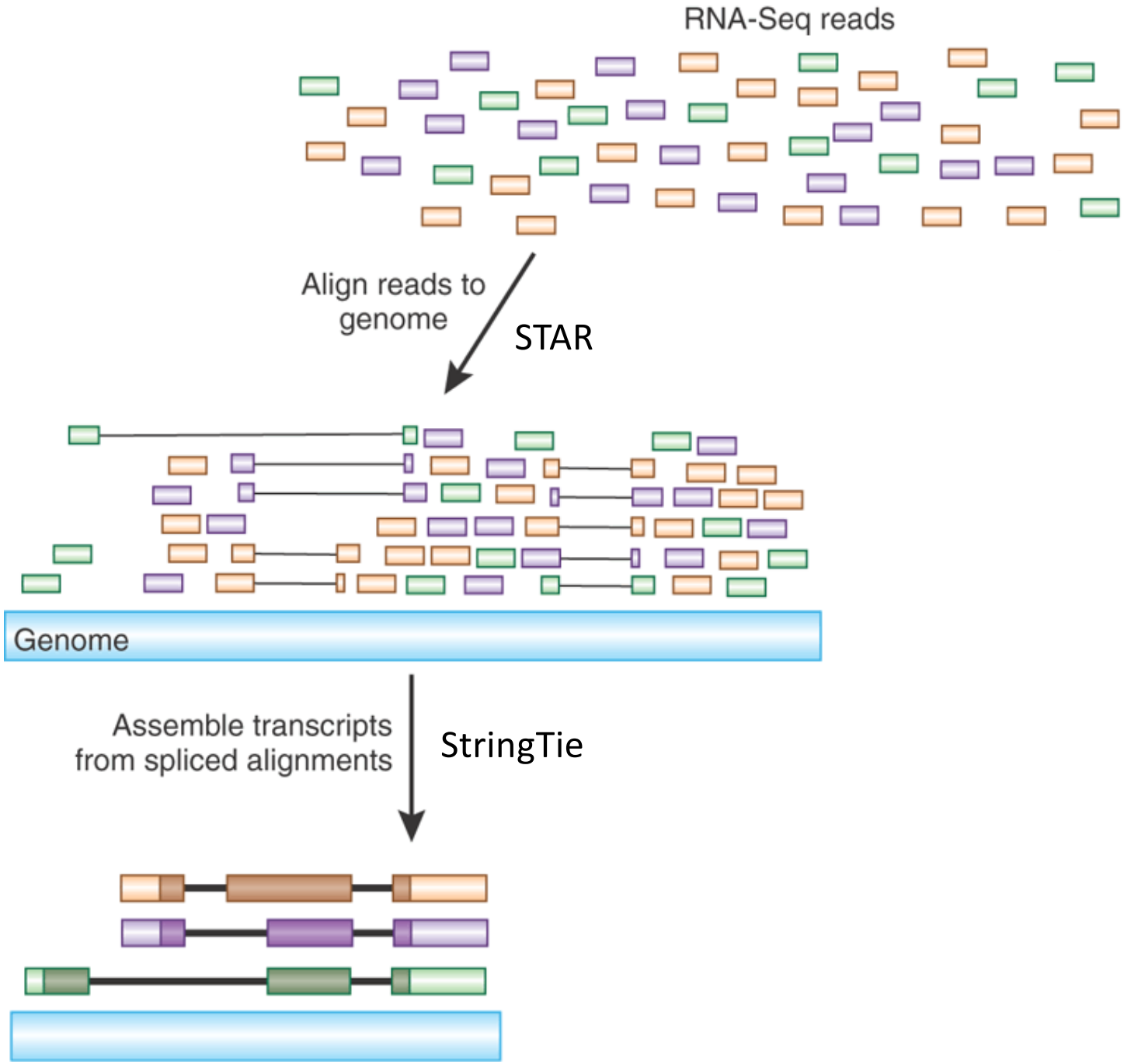
Brian J Haas & Michael C Zody

Nature Biotech, 2010

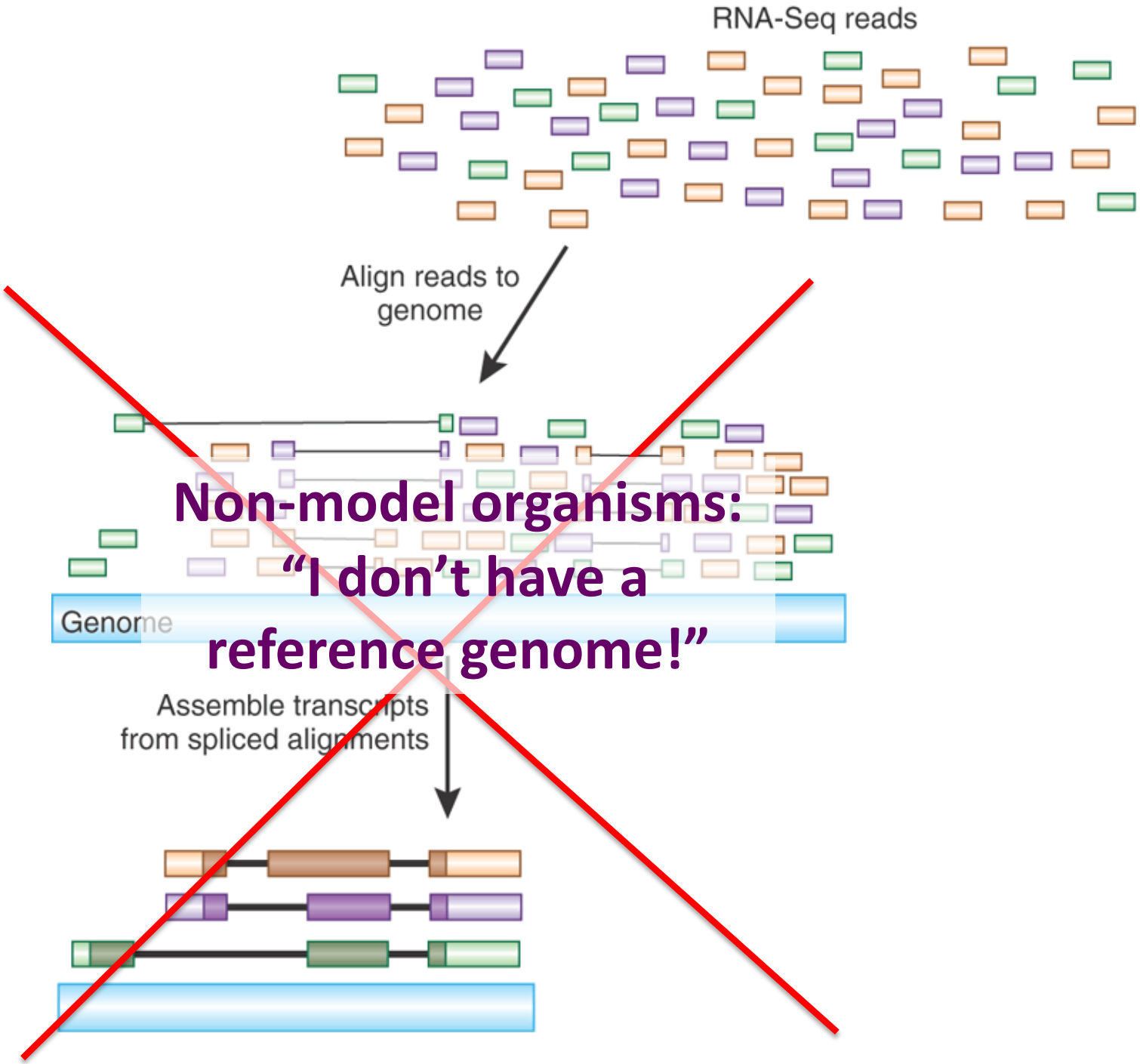
New methods for analyzing RNA-Seq data enable *de novo* reconstruction of the transcriptome.

Transcript Reconstruction from RNA-Seq Reads

Transcript Reconstruction from RNA-Seq Reads

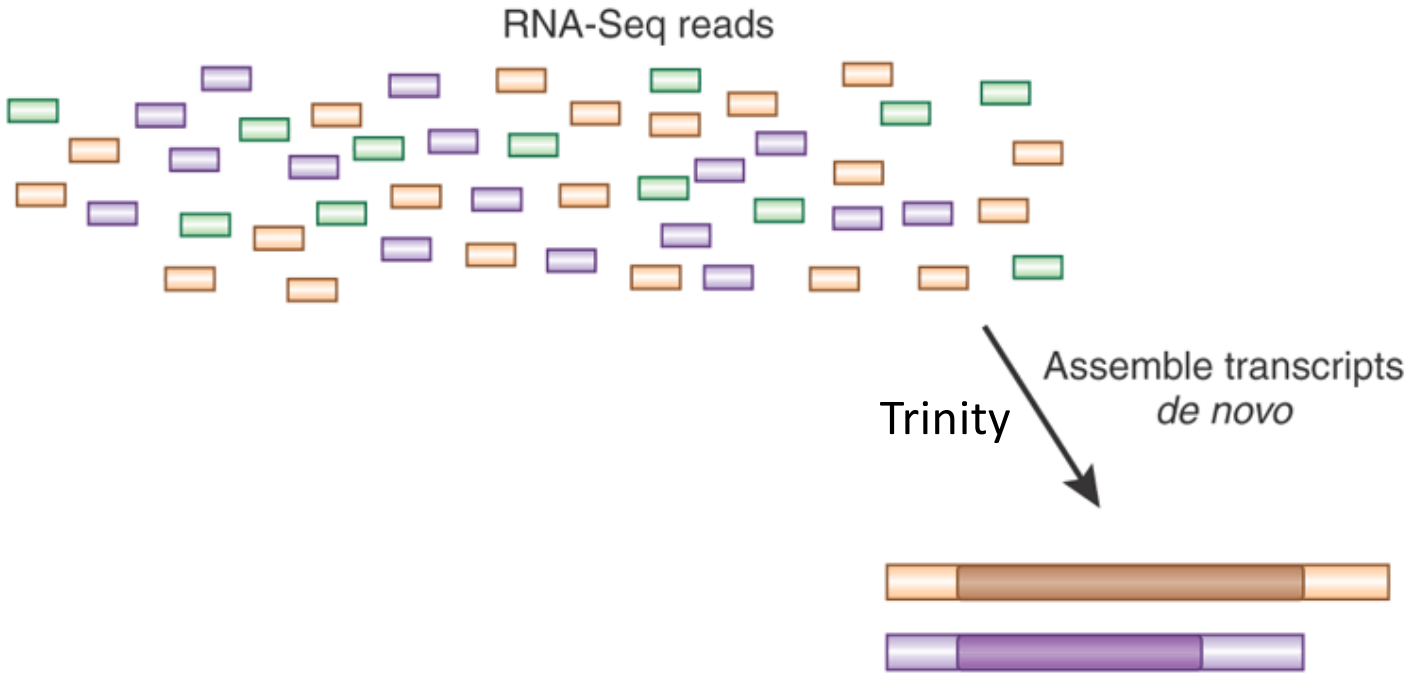


Transcript Reconstruction from RNA-Seq Reads

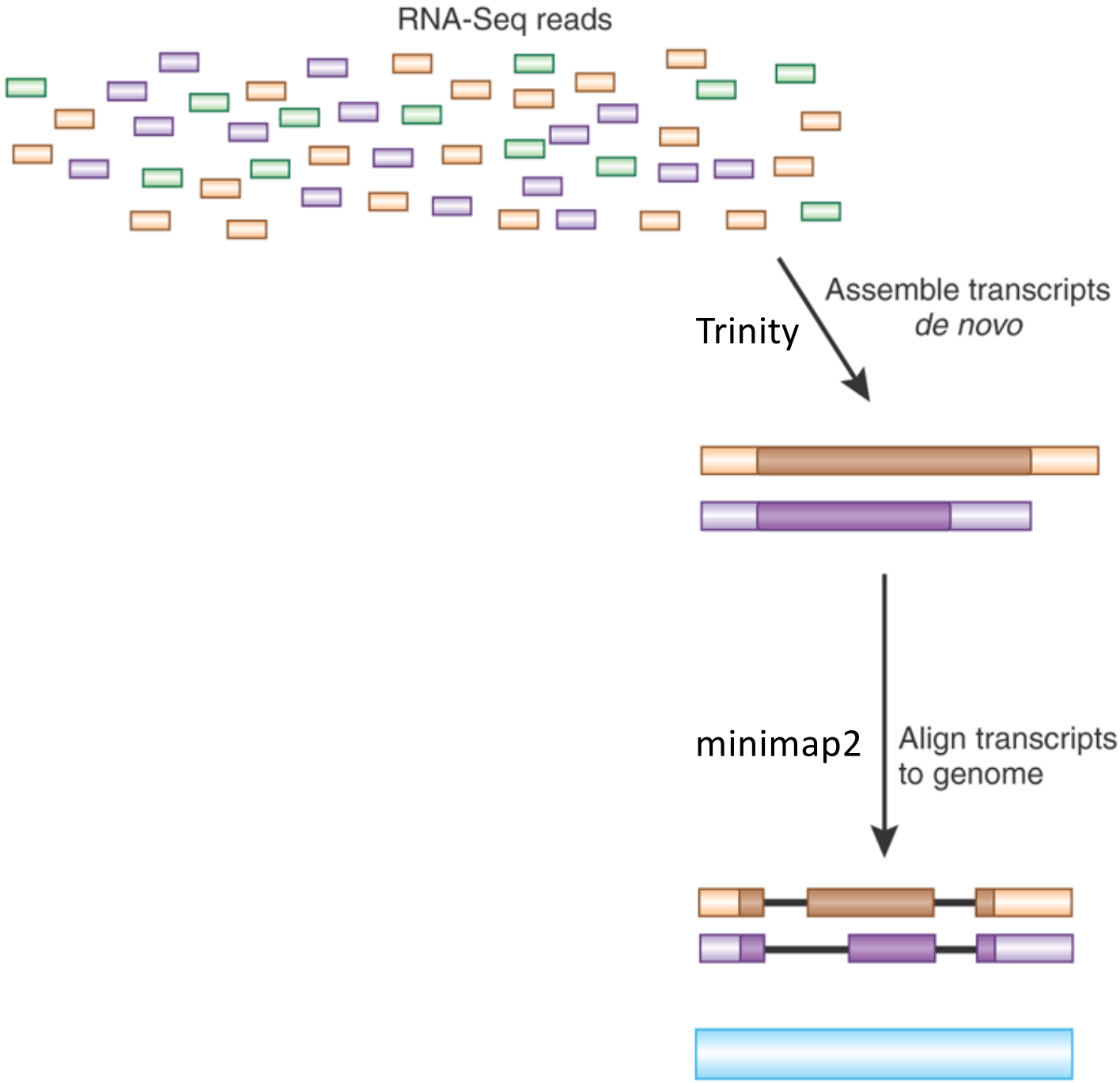


Non-model organisms: "I don't have a reference genome!"

Transcript Reconstruction from RNA-Seq Reads



Transcript Reconstruction from RNA-Seq Reads



Transcript Reconstruction from RNA-Seq Reads

End-to-end **Transcriptome**-based
RNA-Seq Analysis
Software Package

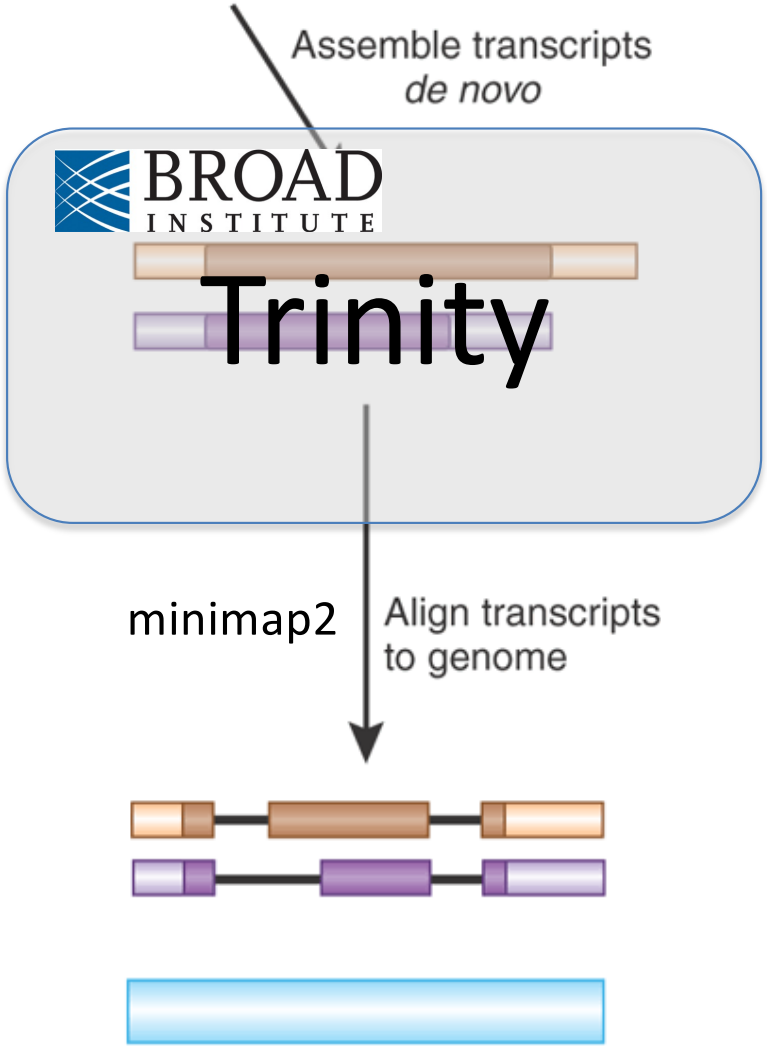
NATURE PROTOCOLS | PROTOCOL

De novo transcript sequence reconstruction from RNA-seq using the Trinity platform for reference generation and analysis

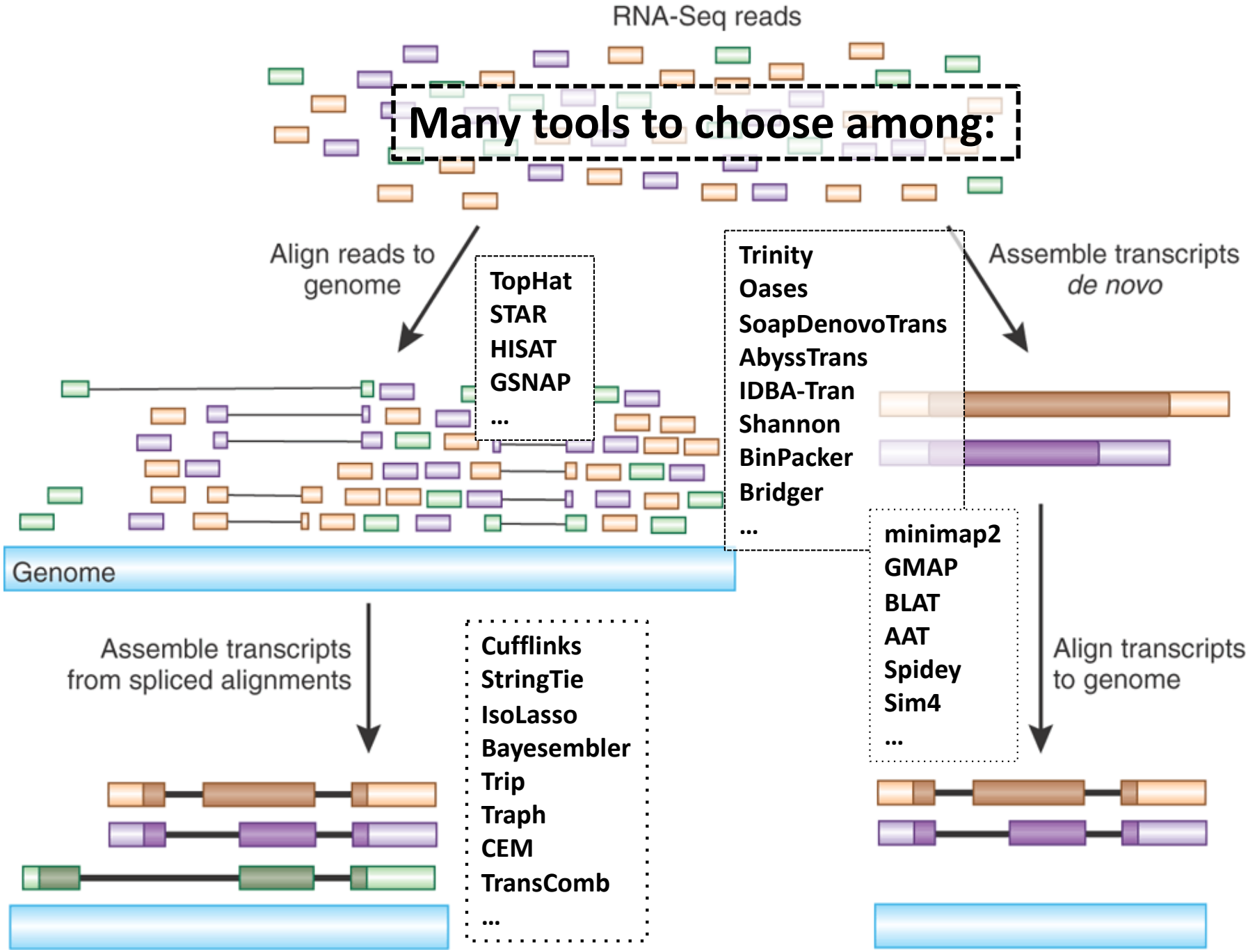
Brian J Haas, Alexie Papanicolaou, Moran Yassour, Manfred Grabherr, Philip D Blood, Joshua Bowden, Matthew Brian Couger, David Eccles, Bo Li, Matthias Lieber, Matthew D MacManes, Michael Ott, Joshua Orvis, Nathalie Pochet, Francesco Strozzi, Nathan Weeks, Rick Westerman, Thomas William, Colin N Dewey, Robert Henschel, Richard D LeDuc, Nir Friedman & Aviv Regev

Affiliations | Contributions | Corresponding authors

Nature Protocols **8**, 1494–1512 (2013) | doi:10.1038/nprot.2013.084
Published online 11 July 2013



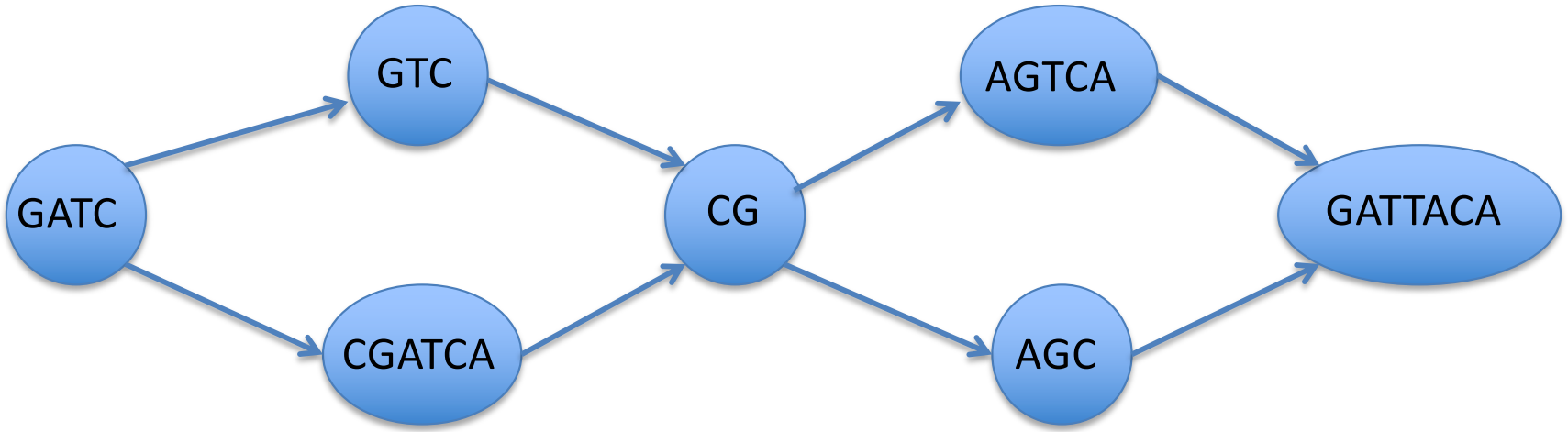
Transcript Reconstruction from RNA-Seq Reads



Graph Data Structures Commonly Used For Assembly

- Sequence
- Order
- Orientation (+, -)
- Overlap

Reads to Graph

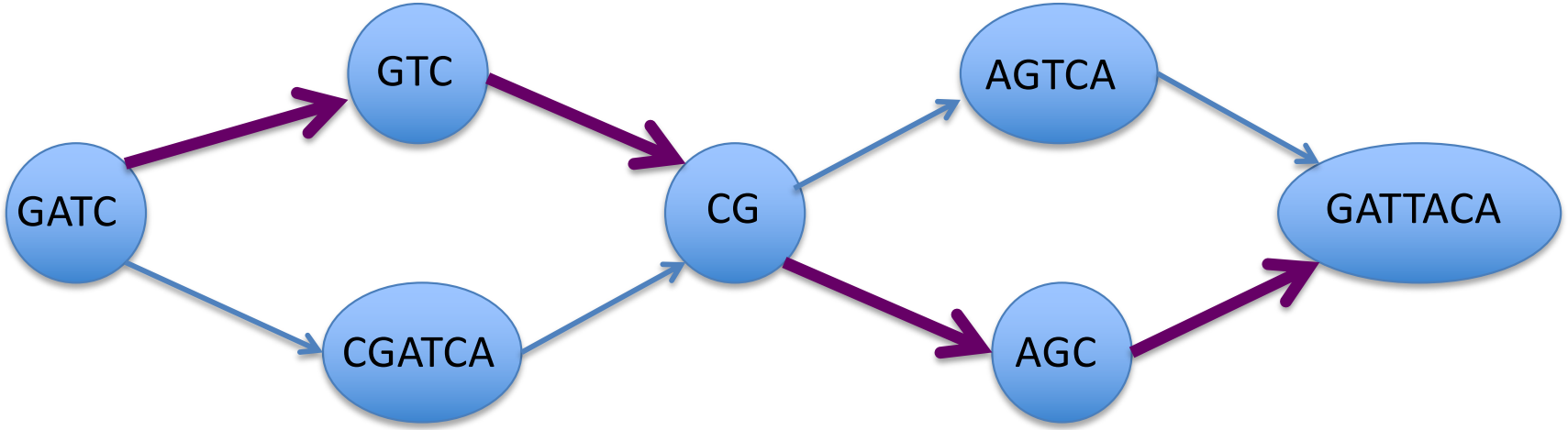


Nodes = sequence (+/-)
Edges = order, overlap

Graph Data Structures Commonly Used For Assembly

- Sequence
- Order
- Orientation (+, -)
- Overlap

Reads to Graph

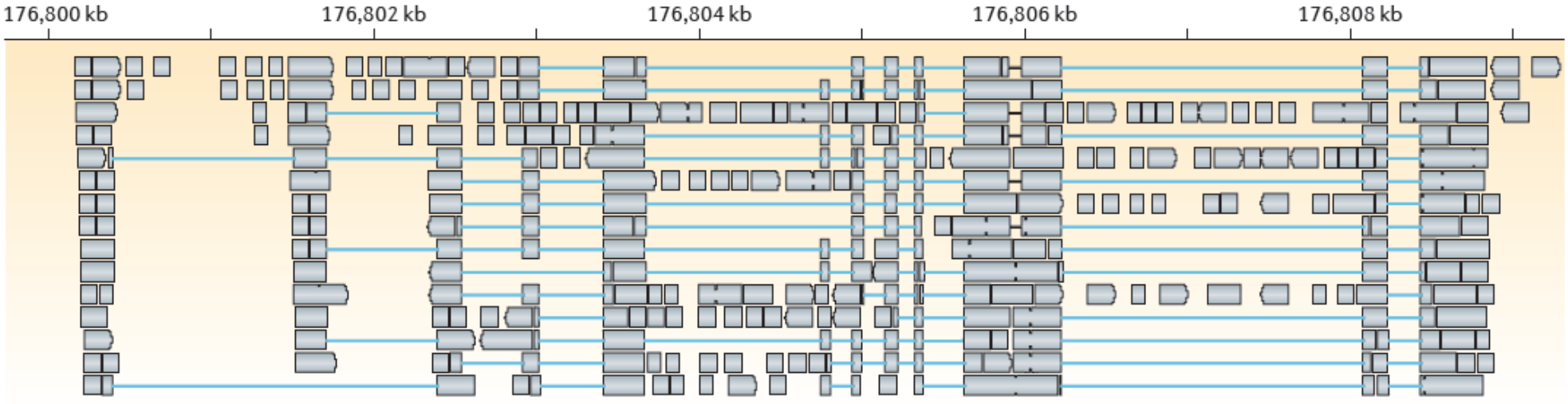


GATCGTCCGAGCGATTACA

Nodes = sequence (+/-)
Edges = order, overlap

Genome-Guided Transcript Reconstruction

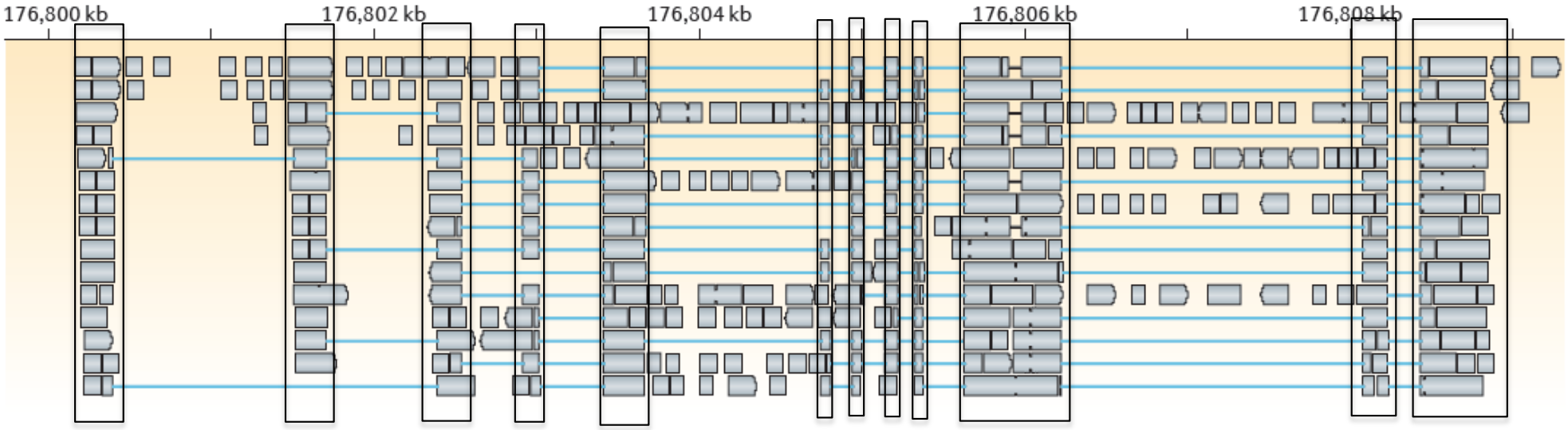
Splice-align reads to the genome



From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

Splice-align reads to the genome

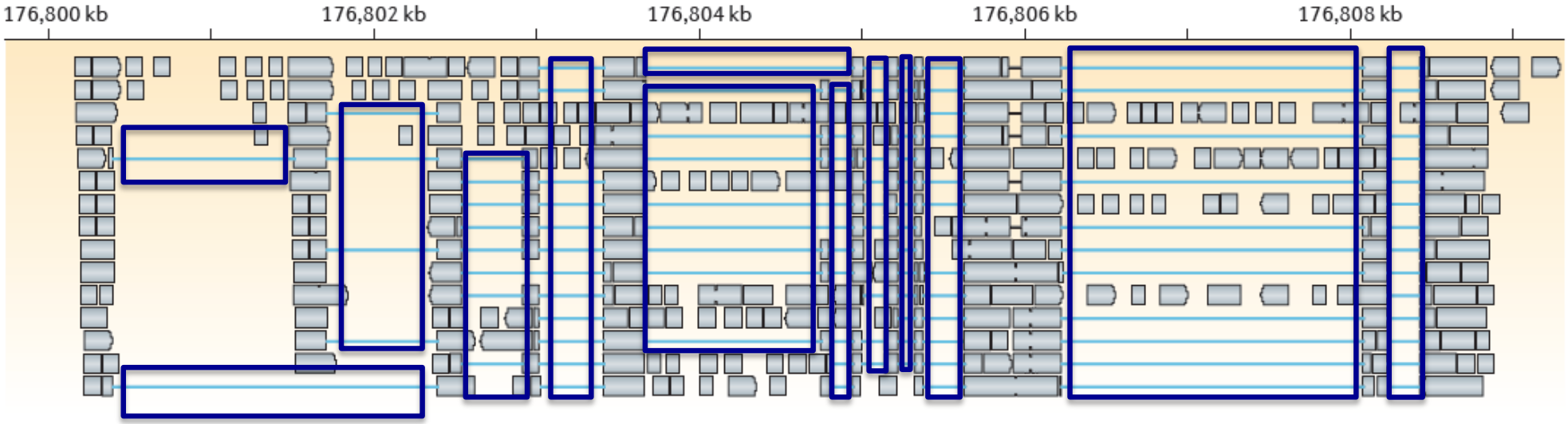


Alignment segment piles => exon regions

From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

Splice-align reads to the genome

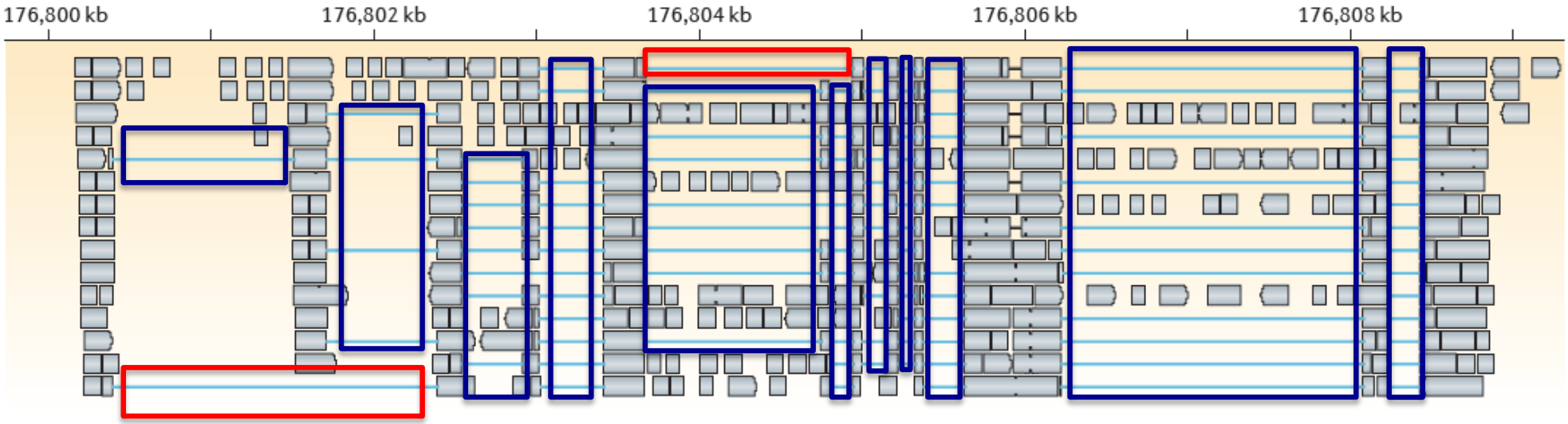


Large alignment gaps => introns

From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

Splice-align reads to the genome

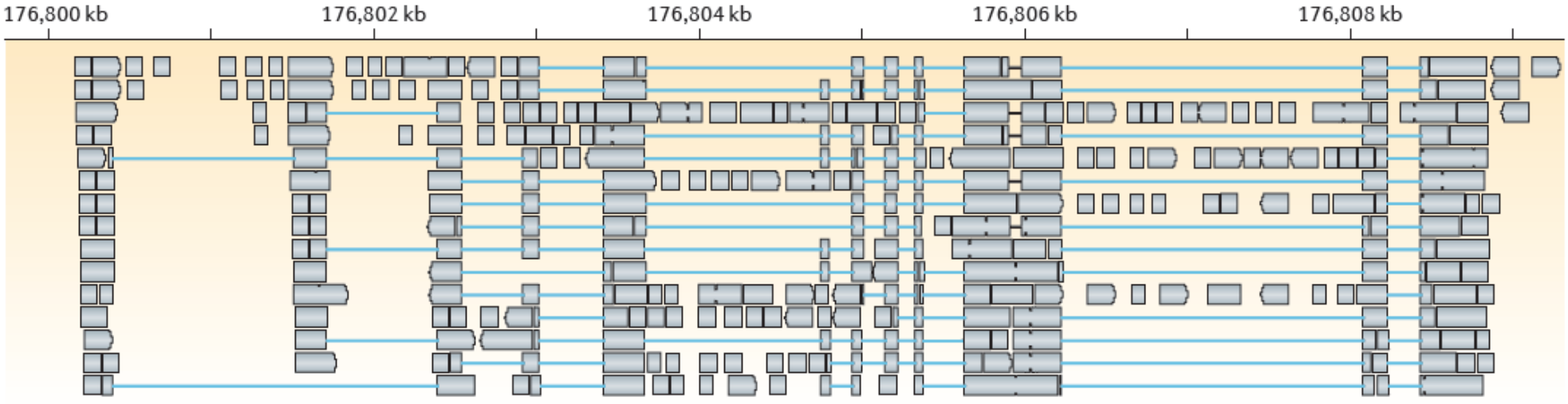


Overlapping but different introns = evidence of alternative splicing

From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

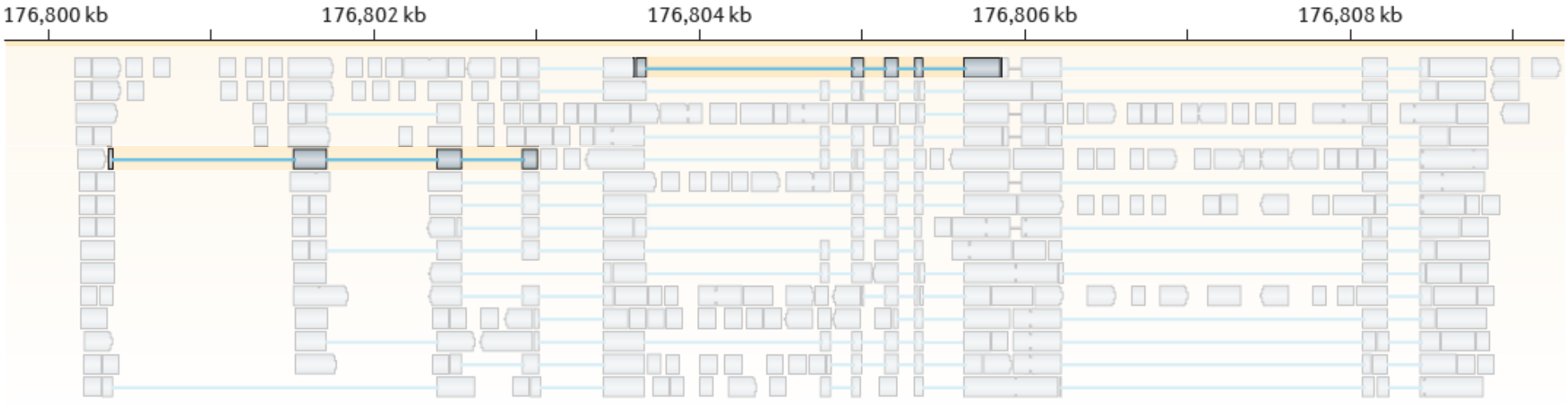
Splice-align reads to the genome



From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

Splice-align reads to the genome

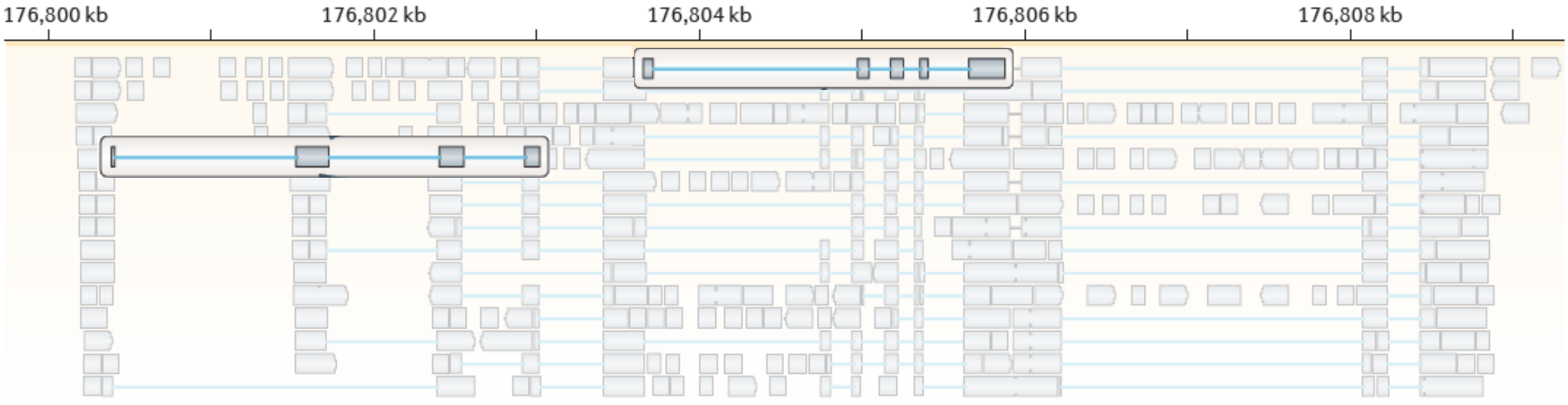


Individual reads can yield multiple exon and intron segments (splice patterns)

From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

Splice-align reads to the genome

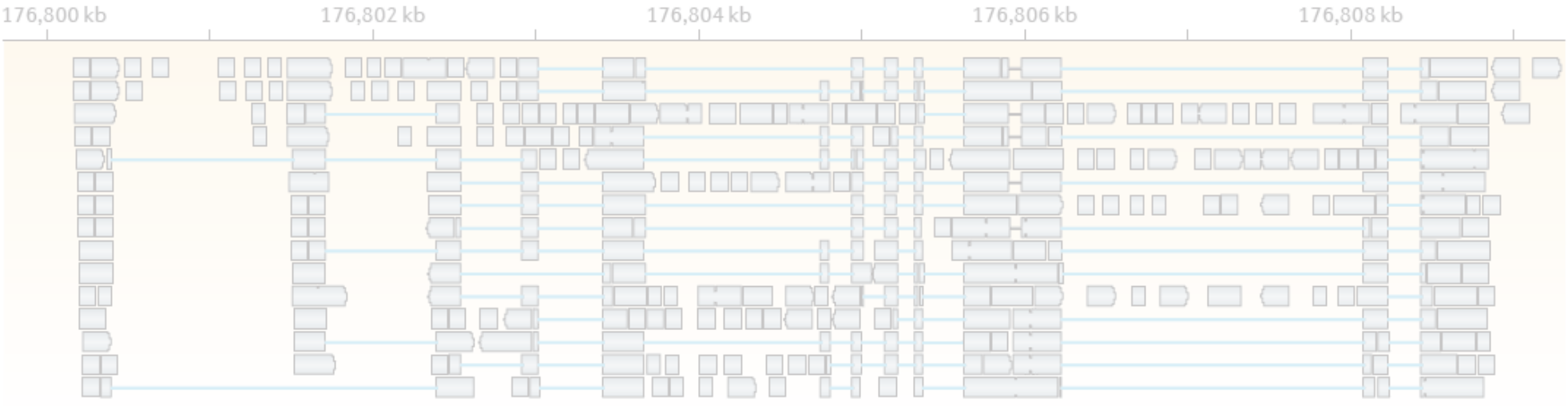


Nodes = unique splice patterns

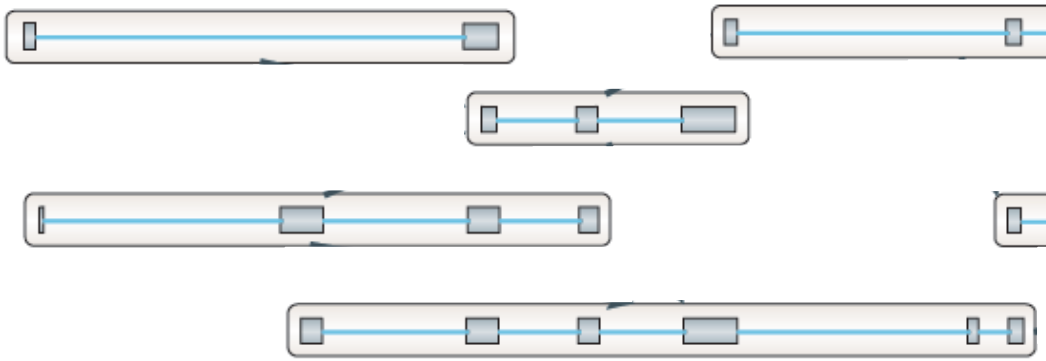
From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

Splice-align reads to the genome



Construct graph from unique splice patterns of aligned reads.

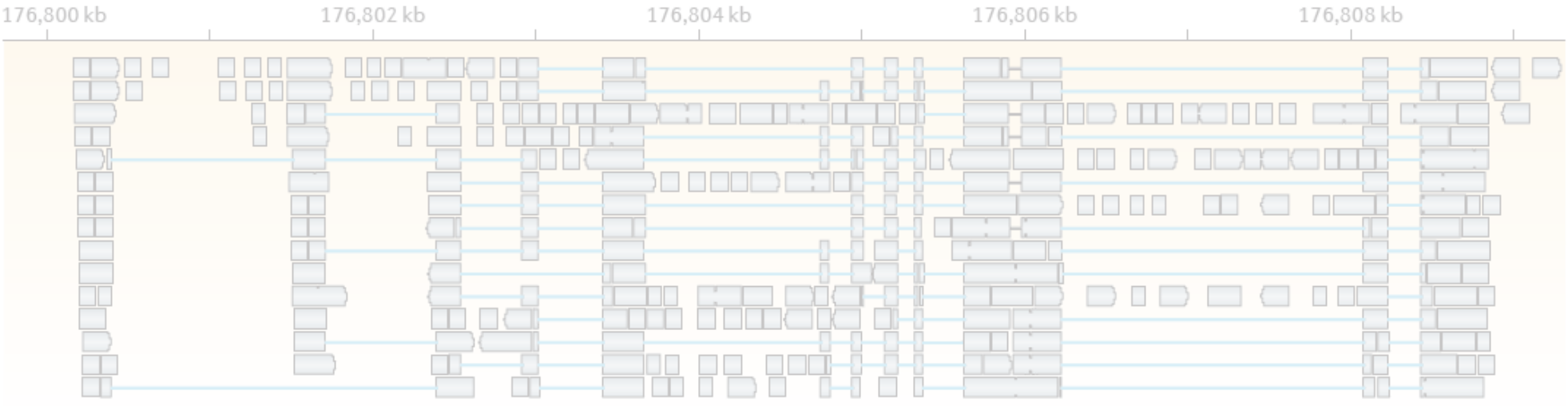


Nodes = unique splice patterns

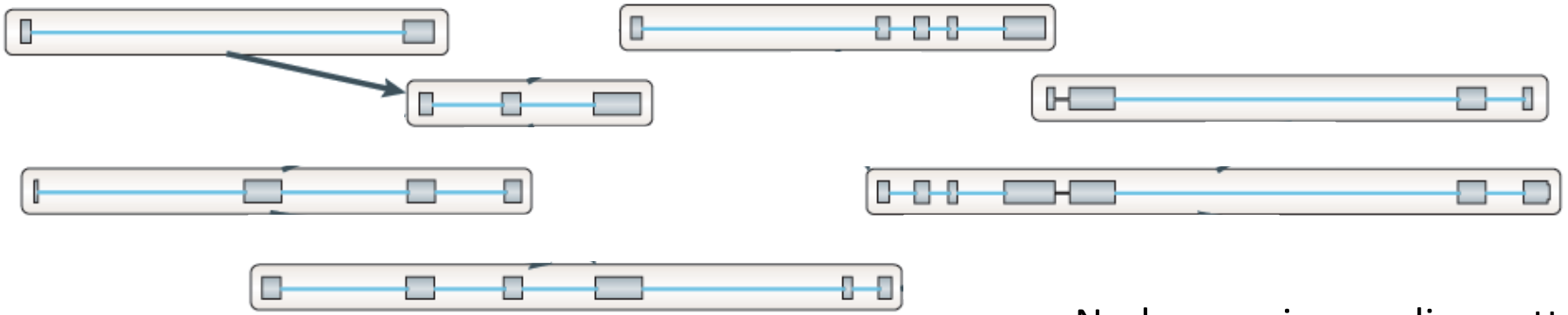
From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

Splice-align reads to the genome



Construct graph from unique splice patterns of aligned reads.



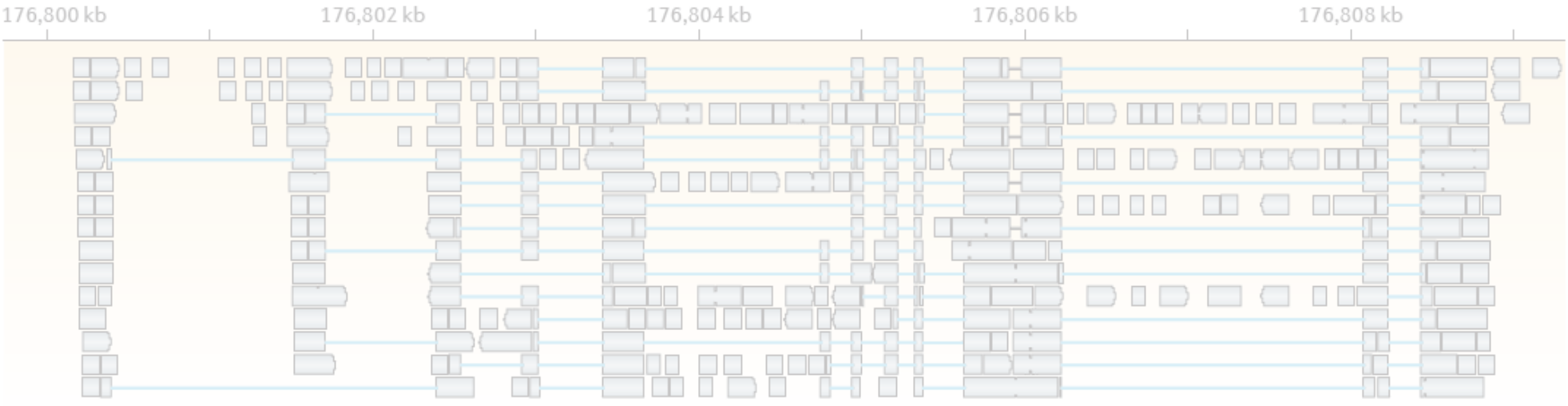
Nodes = unique splice patterns

Edges = compatible patterns

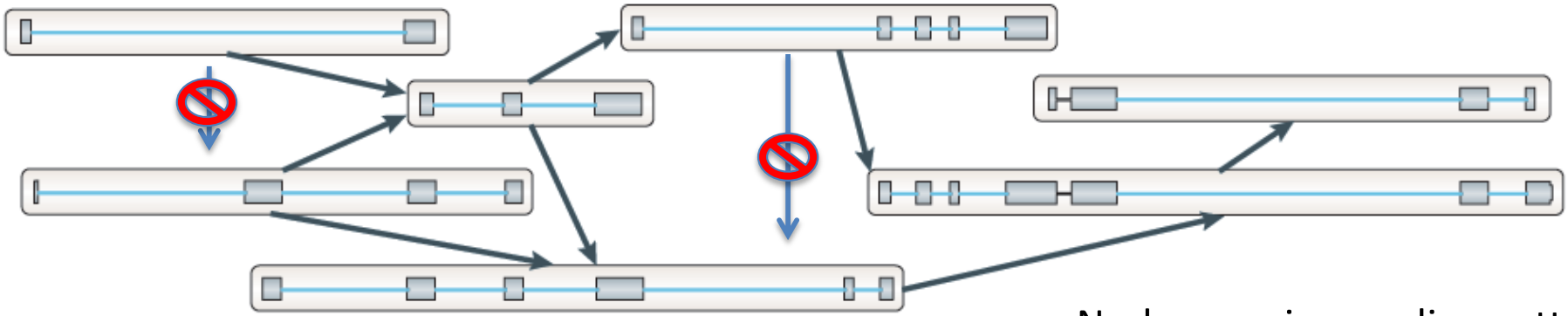
From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

Splice-align reads to the genome



Construct graph from unique splice patterns of aligned reads.



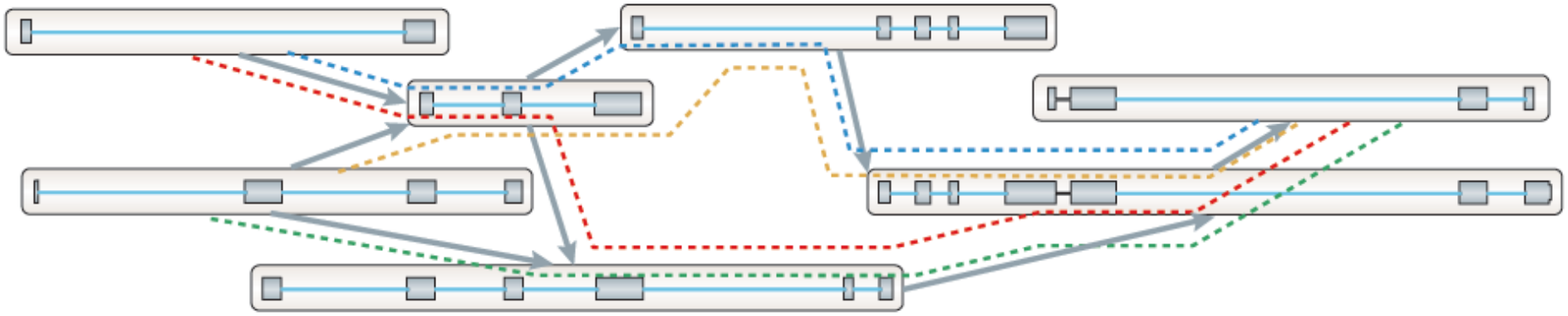
Nodes = unique splice patterns

Edges = compatible patterns

From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

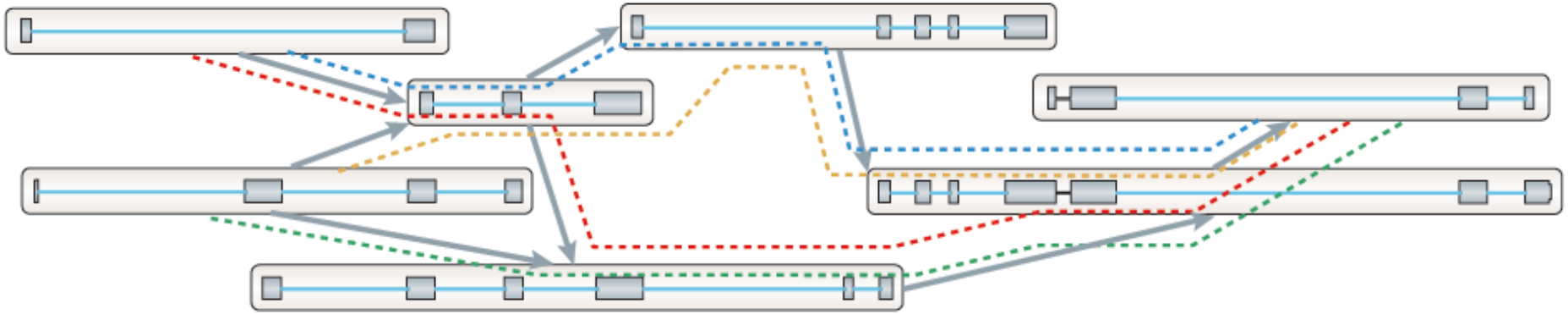
Traverse paths through the graph to assemble transcript isoforms



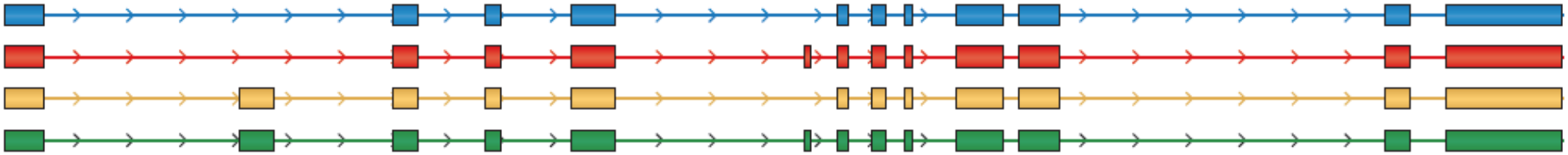
From Martin & Wang. Nature Reviews in Genetics. 2011

Genome-Guided Transcript Reconstruction

Traverse paths through the graph to assemble transcript isoforms



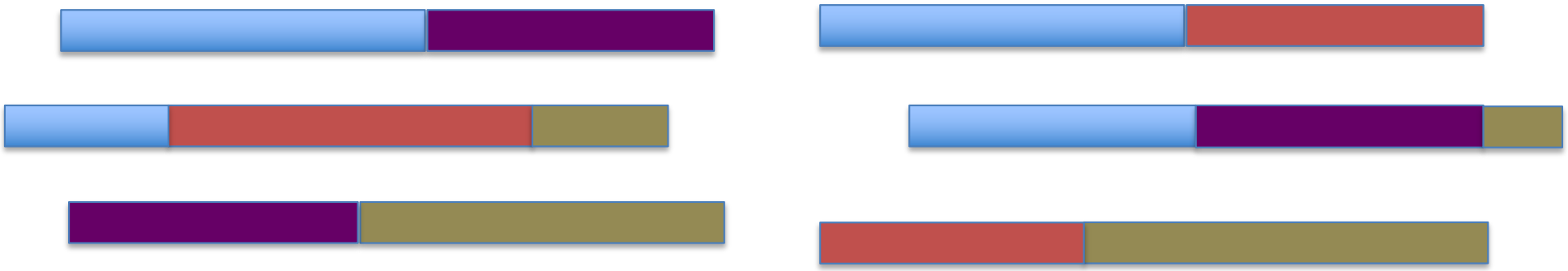
Reconstructed isoforms



From Martin & Wang. Nature Reviews in Genetics. 2011

What if you don’t have a high quality reference genome sequence?

Read Overlap Graph: Reads as nodes, overlaps as edges



Read Overlap Graph: Reads as nodes, overlaps as edges

Node = read
Edge = overlap

Read Overlap Graph: Reads as nodes, overlaps as edges

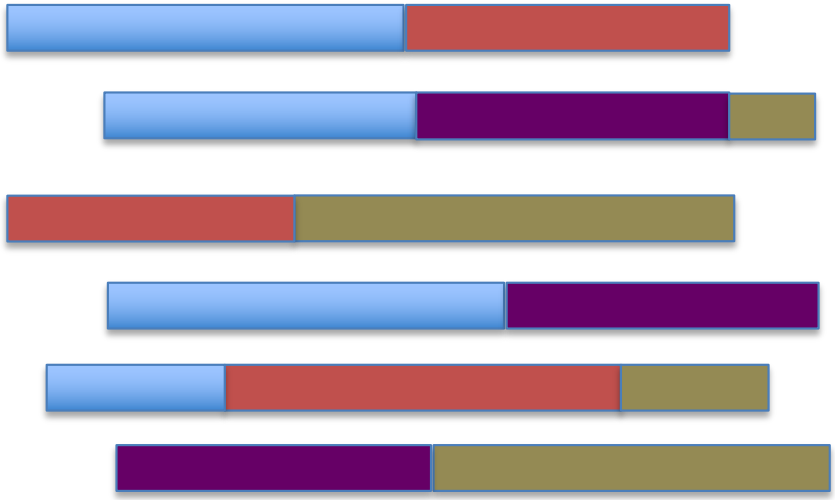
Transcript A

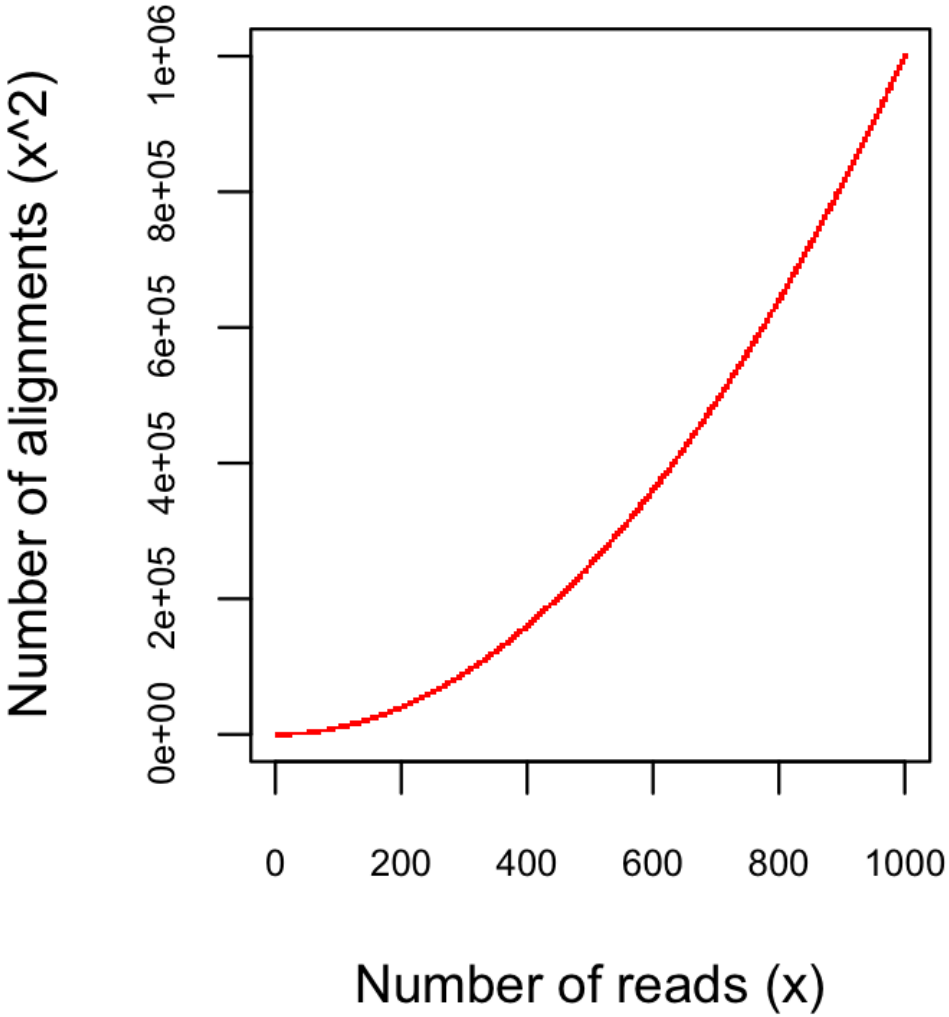
Generate consensus sequence where reads overlap

Node = read
Edge = overlap

Transcript B

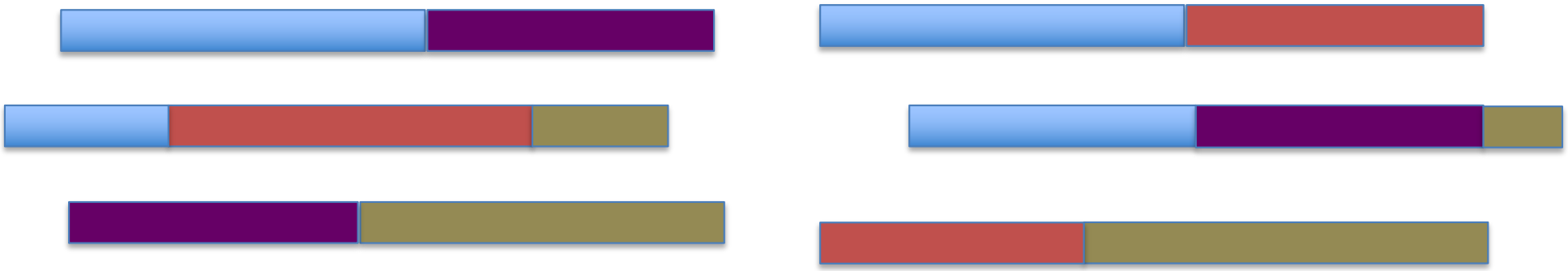
Finding pairwise overlaps between *n* reads involves ~ n^2 comparisons.





Impractical for typical RNA-Seq data (50M reads)

No genome to align to… De novo assembly required



Want to avoid n^2 read alignments to define overlaps

Use a de Bruijn graph

Sequence Assembly via de Bruijn Graphs

Generate all substrings of length k from the reads

From Martin & Wang, Nat. Rev. Genet. 2011

Sequence Assembly via De Bruijn Graphs

Generate all substrings of length k from the reads

From Martin & Wang, Nat. Rev. Genet. 2011

Sequence Assembly via De Bruijn Graphs

Generate all substrings of length k from the reads

Construct the de Bruijn graph

Nodes = unique k-mers

From Martin & Wang, Nat. Rev. Genet. 2011

Sequence Assembly via De Bruijn Graphs

Generate all substrings of length k from the reads

Construct the de Bruijn graph

From Martin & Wang, Nat. Rev. Genet. 2011

Nodes = unique k-mers
Edges = overlap by (k-1)

Sequence Assembly via De Bruijn Graphs

Generate all substrings of length k from the reads

Construct the de Bruijn graph

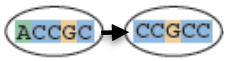
From Martin & Wang, Nat. Rev. Genet. 2011

Nodes = unique k-mers
Edges = overlap by (k-1)

Sequence Assembly via De Bruijn Graphs

Generate all substrings of length k from the reads

Construct the de Bruijn graph

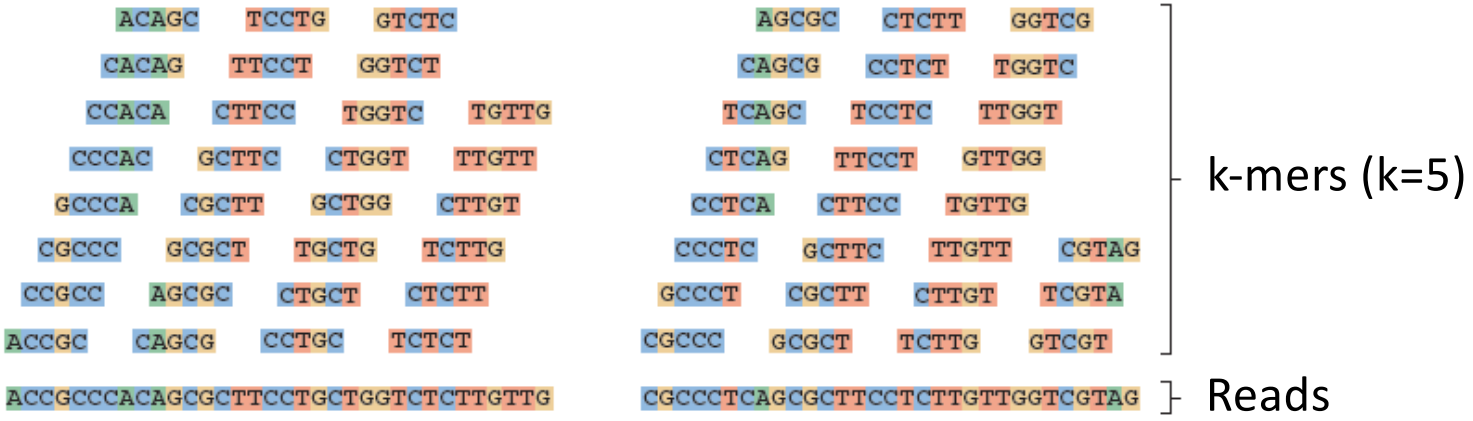


From Martin & Wang, Nat. Rev. Genet. 2011

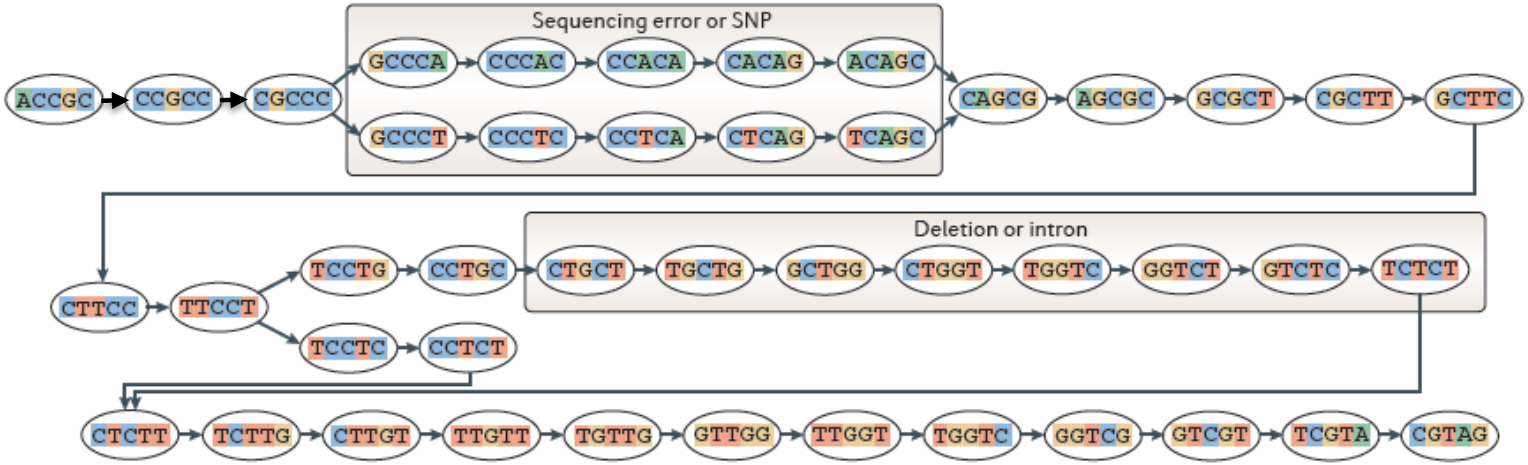
Nodes = unique k-mers
Edges = overlap by (k-1)

Sequence Assembly via De Bruijn Graphs

Generate all substrings of length k from the reads



Construct the de Bruijn graph

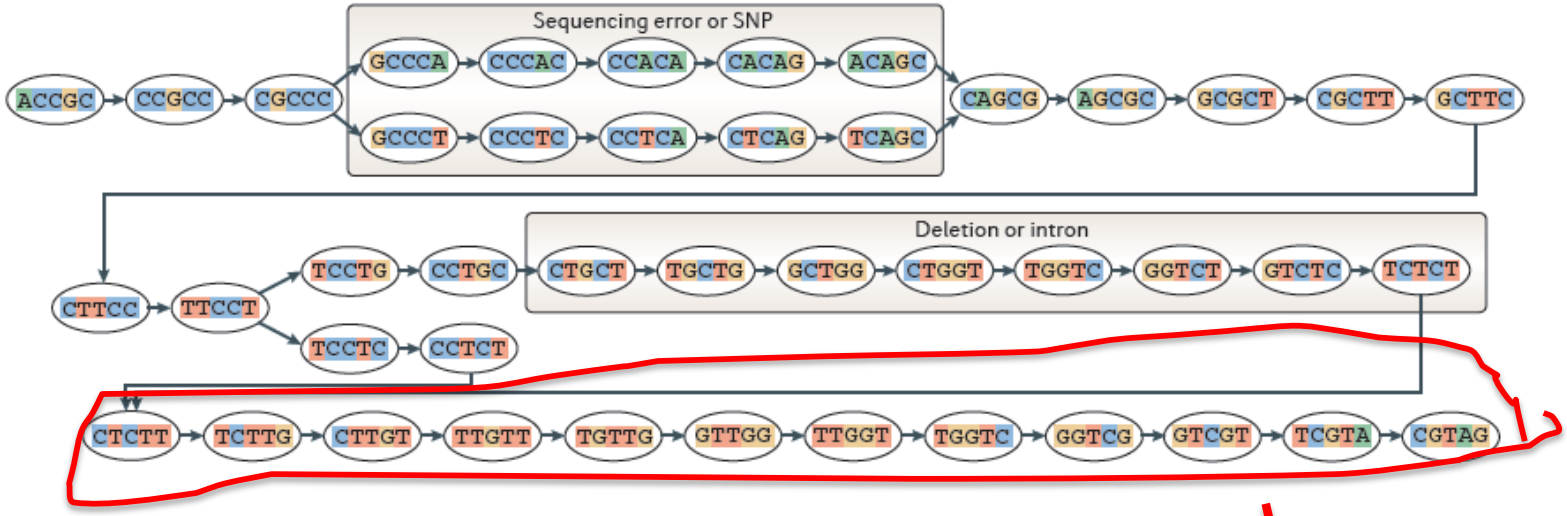


Nodes = unique k-mers

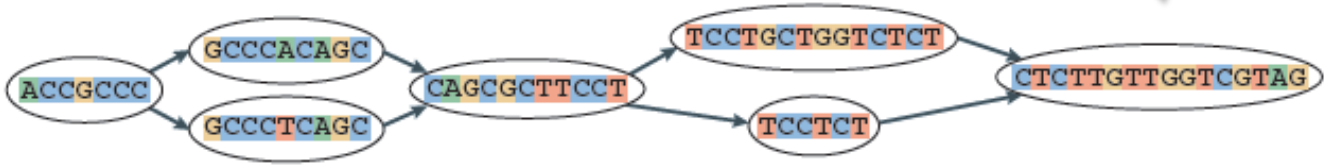
Edges = overlap by (k-1)

From Martin & Wang, Nat. Rev. Genet. 2011

Construct the de Bruijn graph

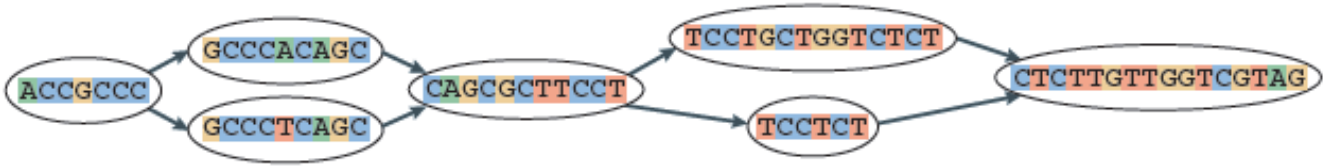


Collapse the de Bruijn graph

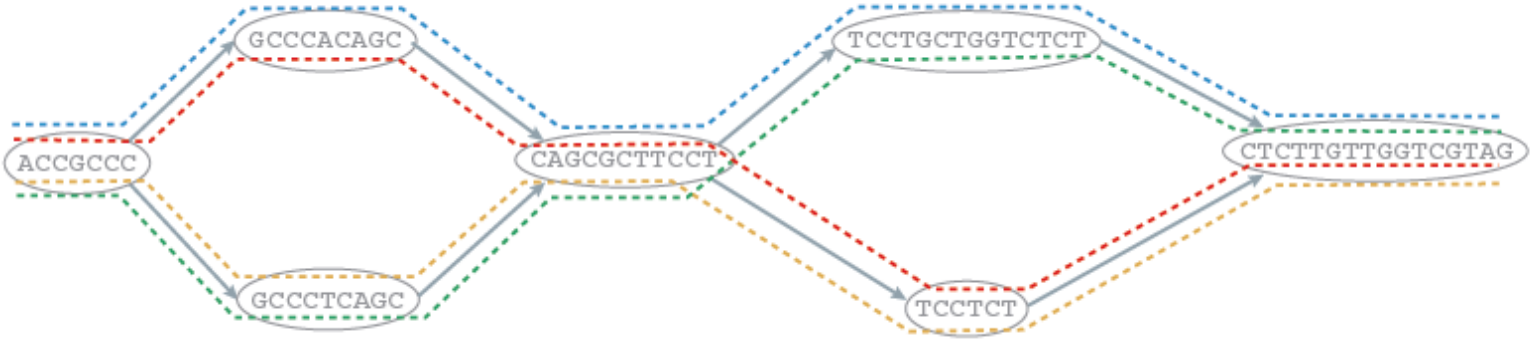


From Martin & Wang, Nat. Rev. Genet. 2011

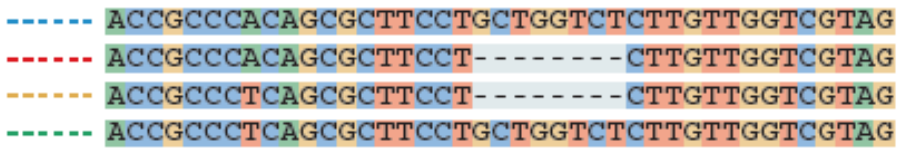
Collapse the de Bruijn graph



Traverse the graph



Assemble Transcript Isoforms



From Martin & Wang, Nat. Rev. Genet. 2011

Contrasting Genome and Transcriptome *De novo* Assembly

Genome Assembly

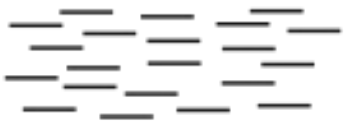
- Uniform coverage
- Single contig per locus
- Assemble small numbers of large Mb-length chromosomes
- Double-stranded data

Transcriptome Assembly

- Exponentially distributed coverage levels
- Multiple contigs per locus (alt splicing)
- Assemble many thousands of Kb-length transcripts
- Strand-specific data available

Trinity – How it works:

RNA-Seq reads → **Linear contigs** → **de-Bruijn graphs** → **Transcripts + Isoforms**



...CTTCGCAA...TGATCGGAT...
...ATTCGCAA...TCATCGGAT...

Thousands of disjoint graphs

Inchworm Algorithm

- Decompose all reads into overlapping Kmers => hashtable(kmer, count)

Read: **AATGTGAAAACTGGATTACATGCTGGTATGTC**…

AATGTGA

ATGTGAA Overlapping kmers of length (k)

TGTGAAA

…

Kmer Catalog (hashtable)

Kmer	Count among all reads
AATGTGA	4
ATGTGAA	2
TGTGAAA	1
GATTACA	9

Inchworm Algorithm

- Decompose all reads into overlapping Kmers => hashtable(kmer, count)
- Identify seed kmer as most abundant Kmer, ignoring low-complexity kmers.

https://en.wikipedia.org/wiki/Gattaca

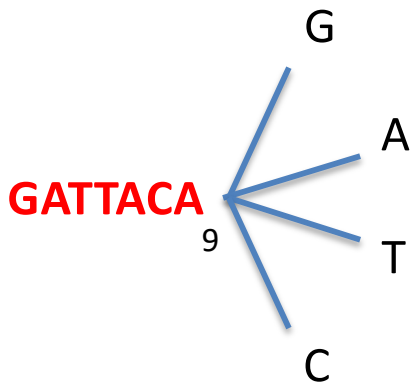
GATTACA
9

Kmer Catalog (hashtable)

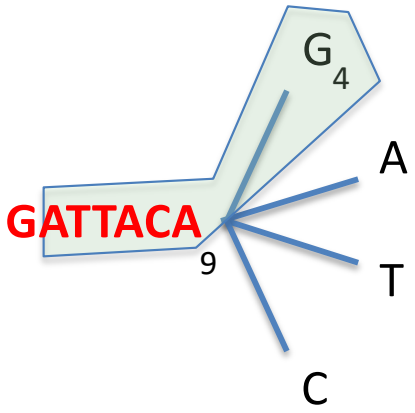
Kmer	Count among all reads
AATGTGA	4
ATGTGAA	2
TGTGAAA	1
GATTACA	9

Inchworm Algorithm

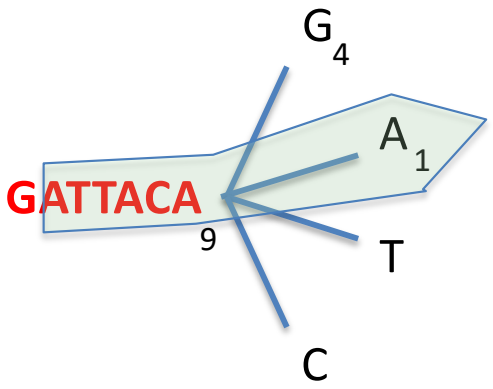
- Decompose all reads into overlapping Kmers => hashtable(kmer, count)
- Identify seed kmer as most abundant Kmer, ignoring low-complexity kmers.
- Extend kmer at 3’ end, guided by coverage.



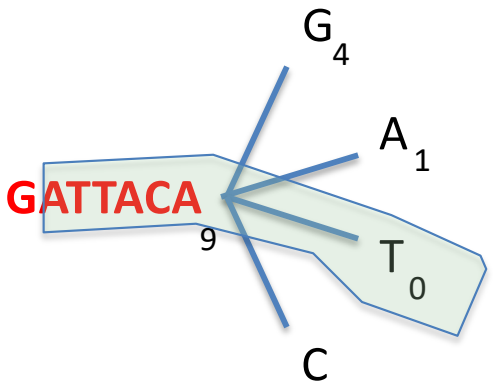
Inchworm Algorithm



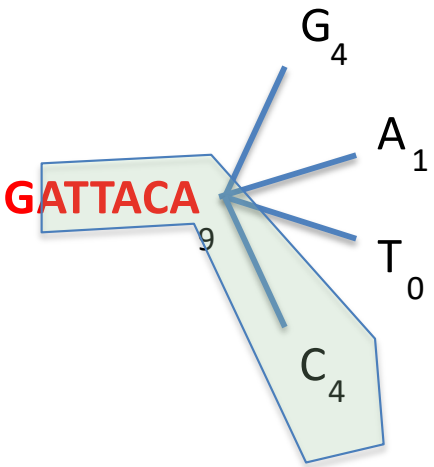
Inchworm Algorithm



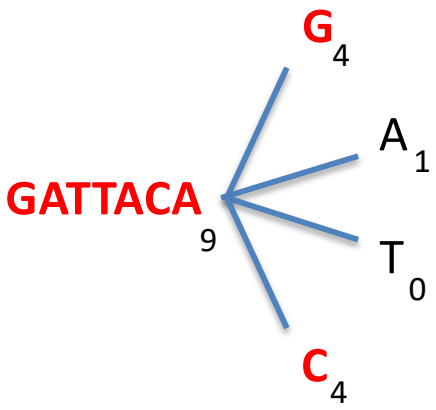
Inchworm Algorithm



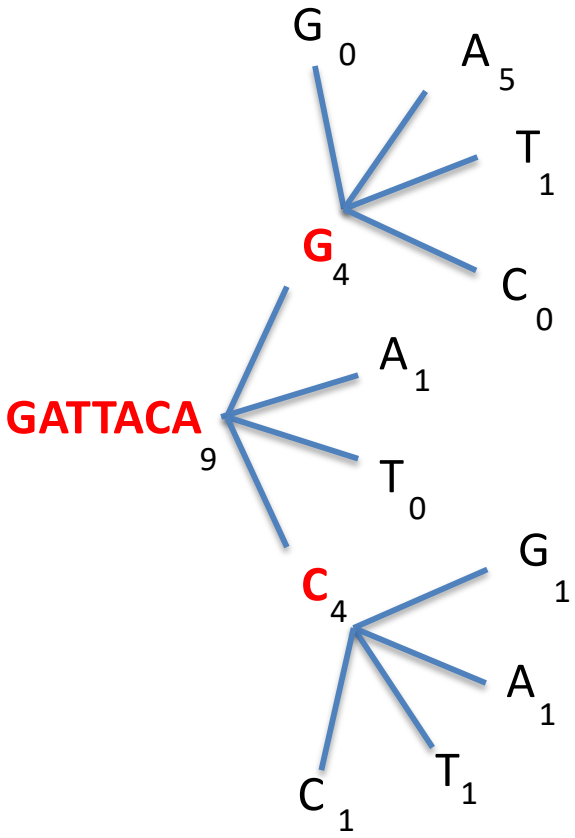
Inchworm Algorithm



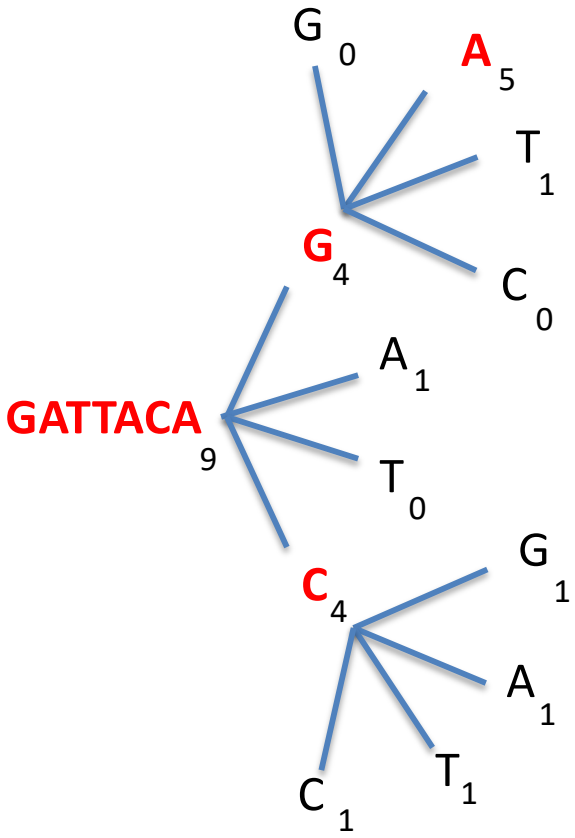
Inchworm Algorithm



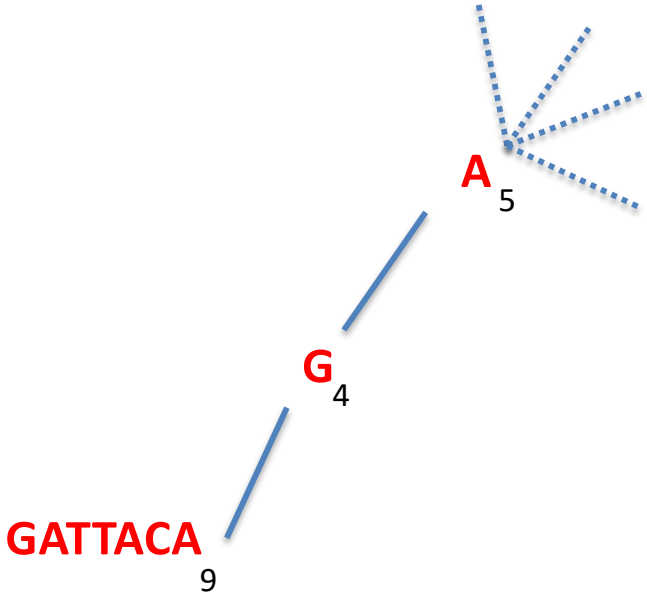
Inchworm Algorithm



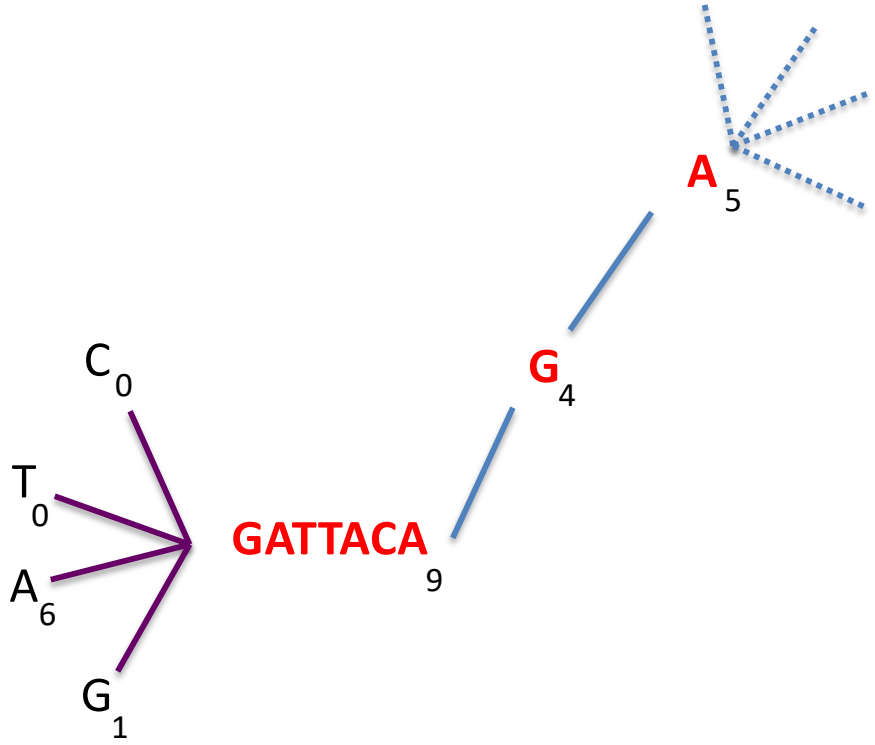
Inchworm Algorithm



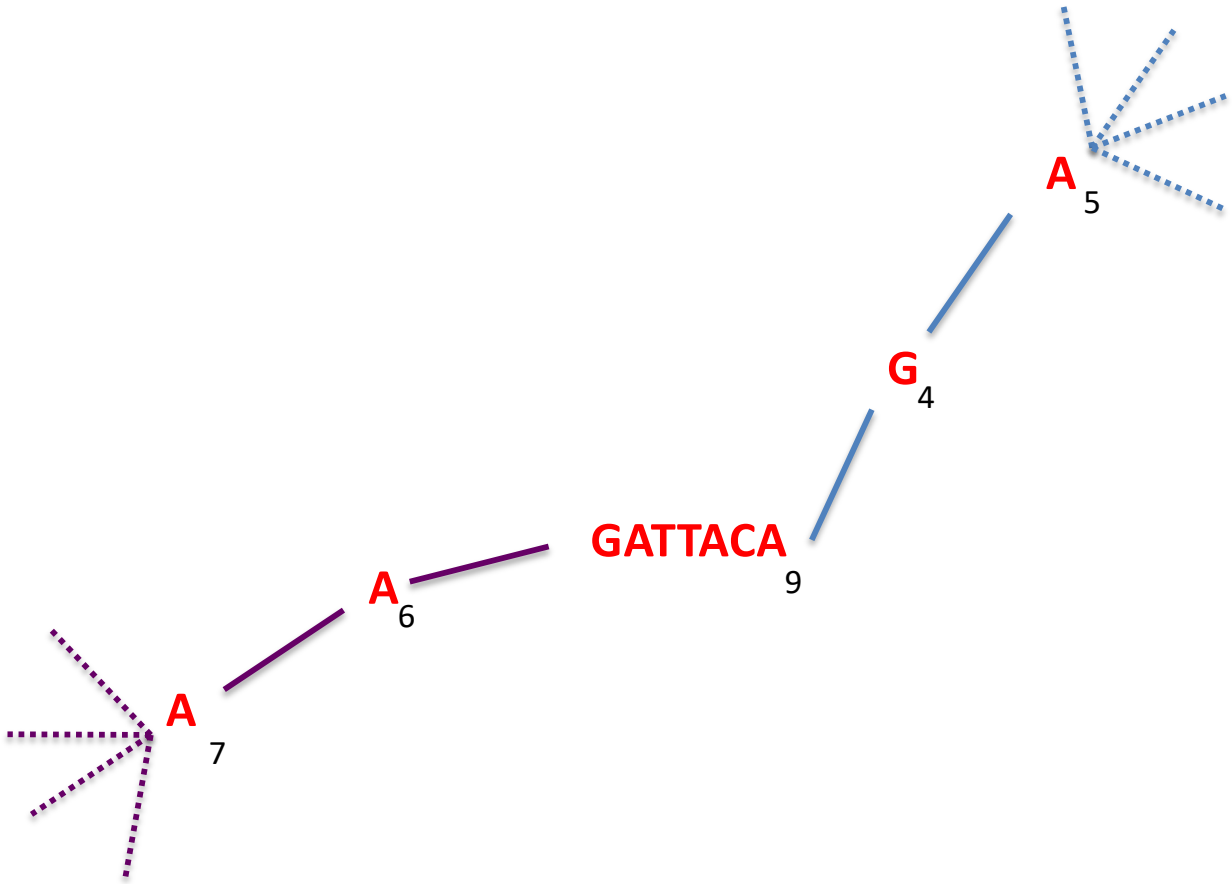
Inchworm Algorithm



Inchworm Algorithm



Inchworm Algorithm



Report contig: ….AAGATTACAGA….

Remove assembled kmers from catalog, then repeat the entire process.

Chrysalis

>a121:len=5845

>a122:len=2560

>a123:len=4443

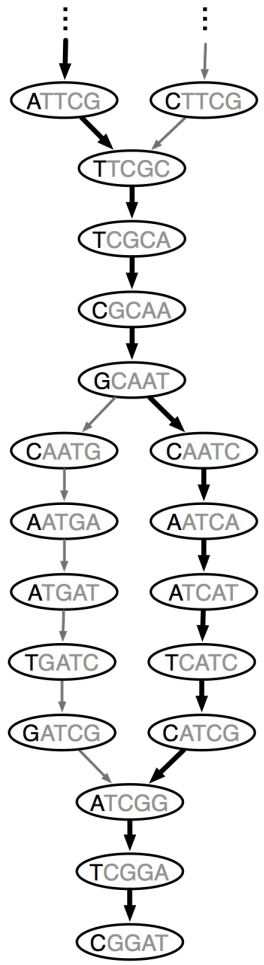
>a124:len=48

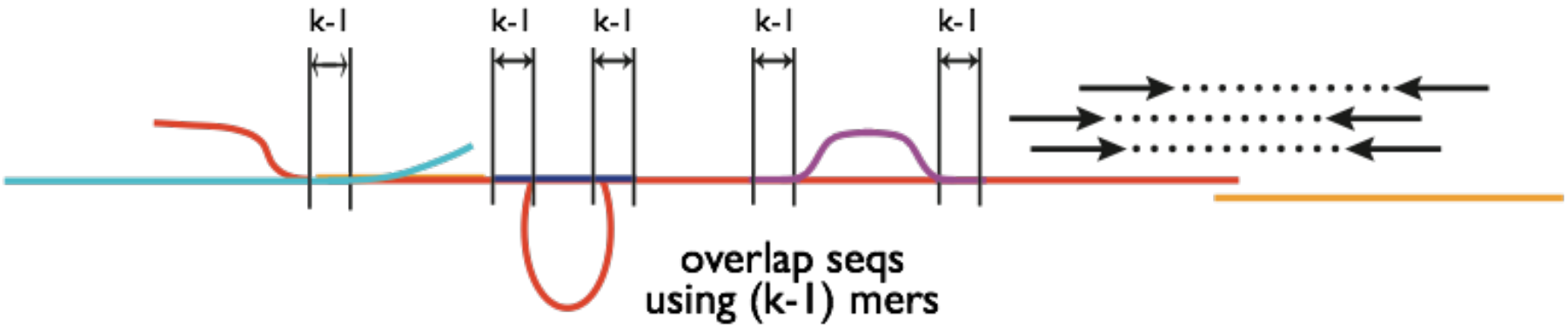
>a125:len=8876

>a126:len=66

Integrate isoforms
via k-1 overlaps

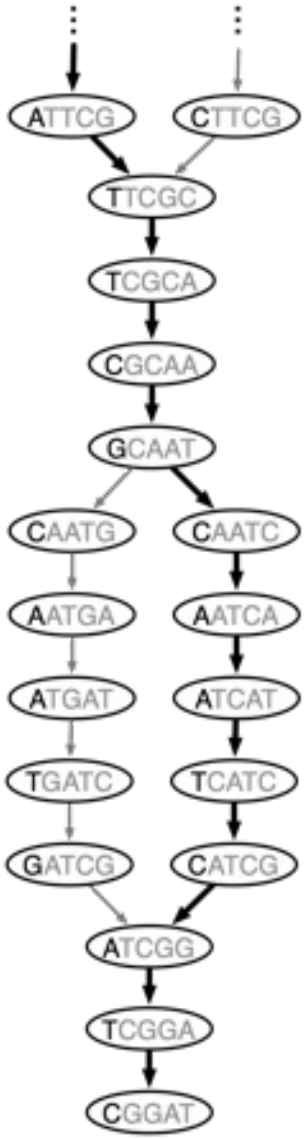
Build de Bruijn Graphs
(ideally, one per gene)





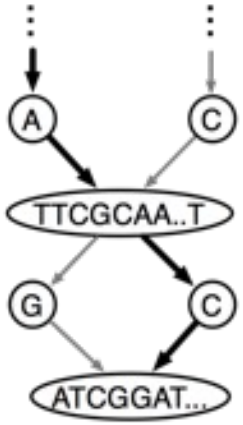

Thousands of Chrysalis Clusters

Butterfly



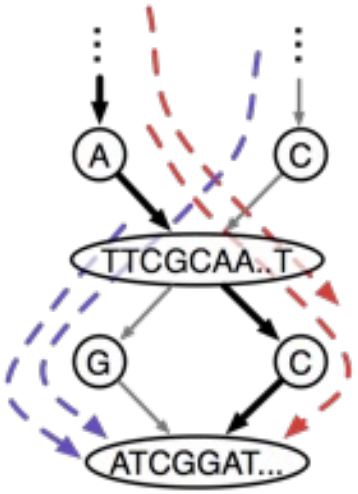
de Bruijn graph

compacting



compact graph

finding paths



compact graph with reads

extracting sequences

..CTTCGCAA..TGATCGGAT...

..ATTCGCAA..TCATCGGAT...

sequences (isoforms and paralogs)

Trinity output: A multi-fasta file

nature protocols

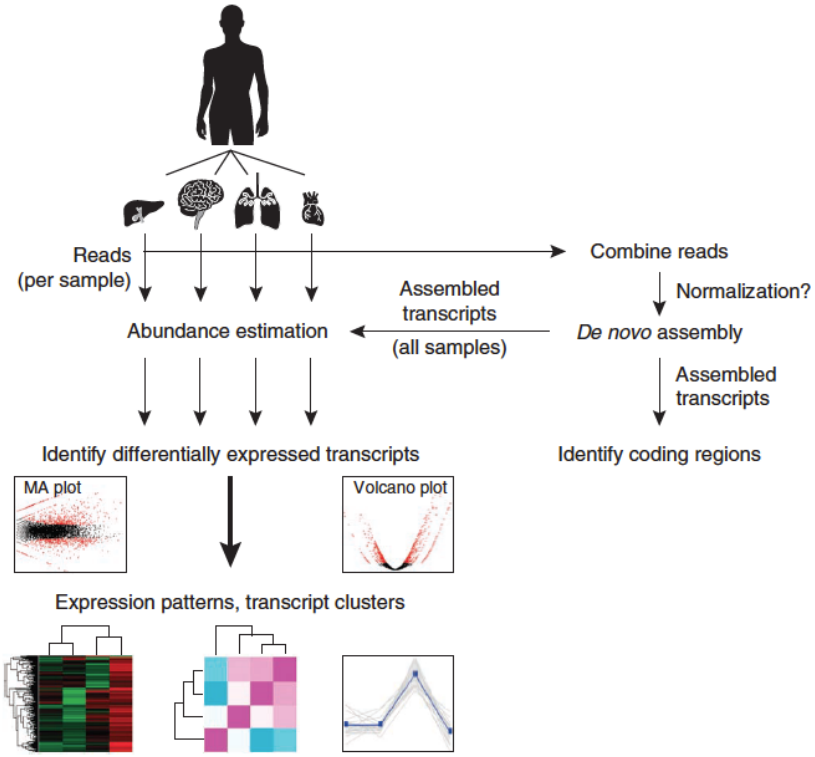
NATURE PROTOCOLS | PROTOCOL

De novo transcript sequence reconstruction from RNA-seq using the Trinity platform for reference generation and analysis

Brian J Haas, Alexie Papanicolaou, Moran Yassour, Manfred Grabherr, Philip D Blood, Joshua Bowden, Matthew Brian Couger, David Eccles, Bo Li, Matthias Lieber, Matthew D MacManes, Michael Ott, Joshua Orvis, Nathalie Pochet, Francesco Strozzi, Nathan Weeks, Rick Westerman, Thomas William, Colin N Dewey, Robert Henschel, Richard D LeDuc, Nir Friedman & Aviv Regev

Affiliations | Contributions | Corresponding authors

Nature Protocols **8**, 1494–1512 (2013) | doi:10.1038/nprot.2013.084
Published online 11 July 2013



RNA-Seq De novo Assembly Using Trinity

Quick Guide for the Impatient

Trinity assembles transcript sequences from Illumina RNA-Seq data.

Download Trinity here.

Build Trinity by typing 'make' in the base installation directory.

Assemble RNA-Seq data like so:

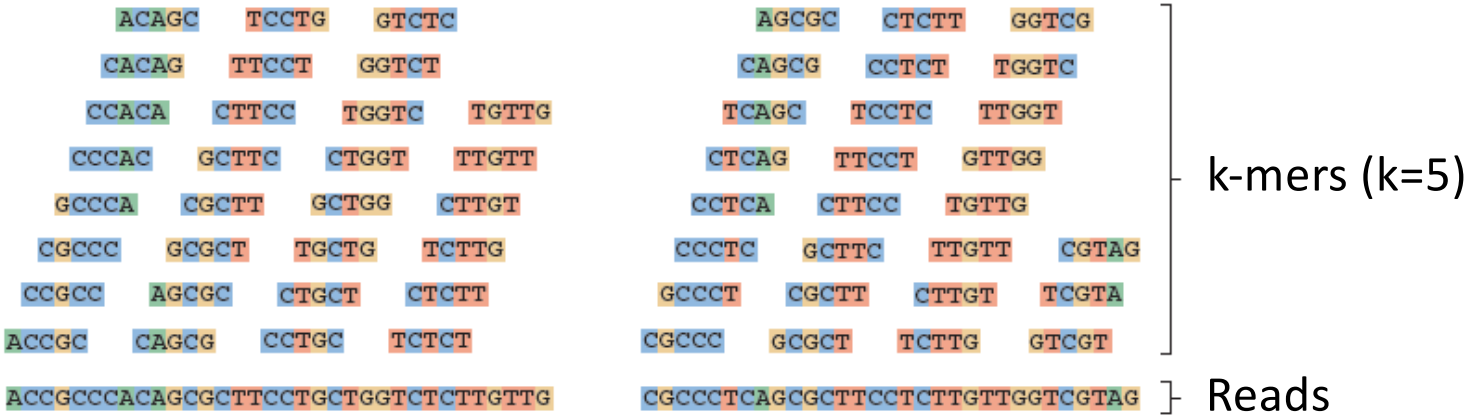
```
Trinity --seqType fq --left reads_1.fq --right reads_2.fq --CPU 6 --max_memory 20G
```

Find assembled transcripts as: 'trinity_out_dir/Trinity.fasta'

Use the documentation links in the right-sidebar to navigate this documentation, and contact our Google group for technical support.

ProgForBio Exercise : Build a program that counts k-mers

Generate all substrings of length k from the reads



Using a kmer size of 8 and reporting the top 10 kmers and their counts:

```
kmer_counter.py 8 reads.left.fq 10
```

.

```
TTTTTTTT        1743
AAAAAAAA        1204
CCAGCCTT        666
GAAGCTGG        627
CAGGCAGG        549
TTTGCTGT        536
GCCTGCTG        533
CTTGGTCT        525
AAGCTGGA        518
TTTTATTT        504
```

https://github.com/trinityrnaseq/CSHLProgForBio/tree/main/Exercise-counting_kmers

Some Important Python Coding Conventions

PEP8 Summary from Google AI

Python coding conventions are outlined in PEP 8, the official style guide for Python code. Here's a summary of the key points:

Naming Conventions:

- Variables and Functions: Use lowercase with underscores for separation (e.g., my_variable, calculate_average).
- Constants: Use uppercase with underscores (e.g., MAX_VALUE, PI).
- Classes: Use CamelCase (e.g., MyClass, ShoppingCart).
- Modules: Use lowercase (e.g., mymodule, utils).

Whitespace:

- *Indentation: Use 4 spaces per indentation level.*
- Spaces: Use spaces around operators and after commas (e.g., x = 1 + 2).
- Blank Lines: Use blank lines to separate functions and classes, and within functions to separate logical blocks.

Line Length:

Keep lines under 79 characters.

Imports:

- Put imports at the top of the file.
- Group imports in this order: standard library, third-party libraries, and local modules.

Docstrings:

- Use docstrings to document modules, classes, functions, and methods.

Other Important Conventions:

- Avoid using global variables.
- Use meaningful variable names.
- Write clear, concise code.
- Test your code thoroughly.

https://peps.python.org/pep-0008/

A useful structure of a python script for shareable functions

(How B. Haas often does it)

Import modules you need in this script

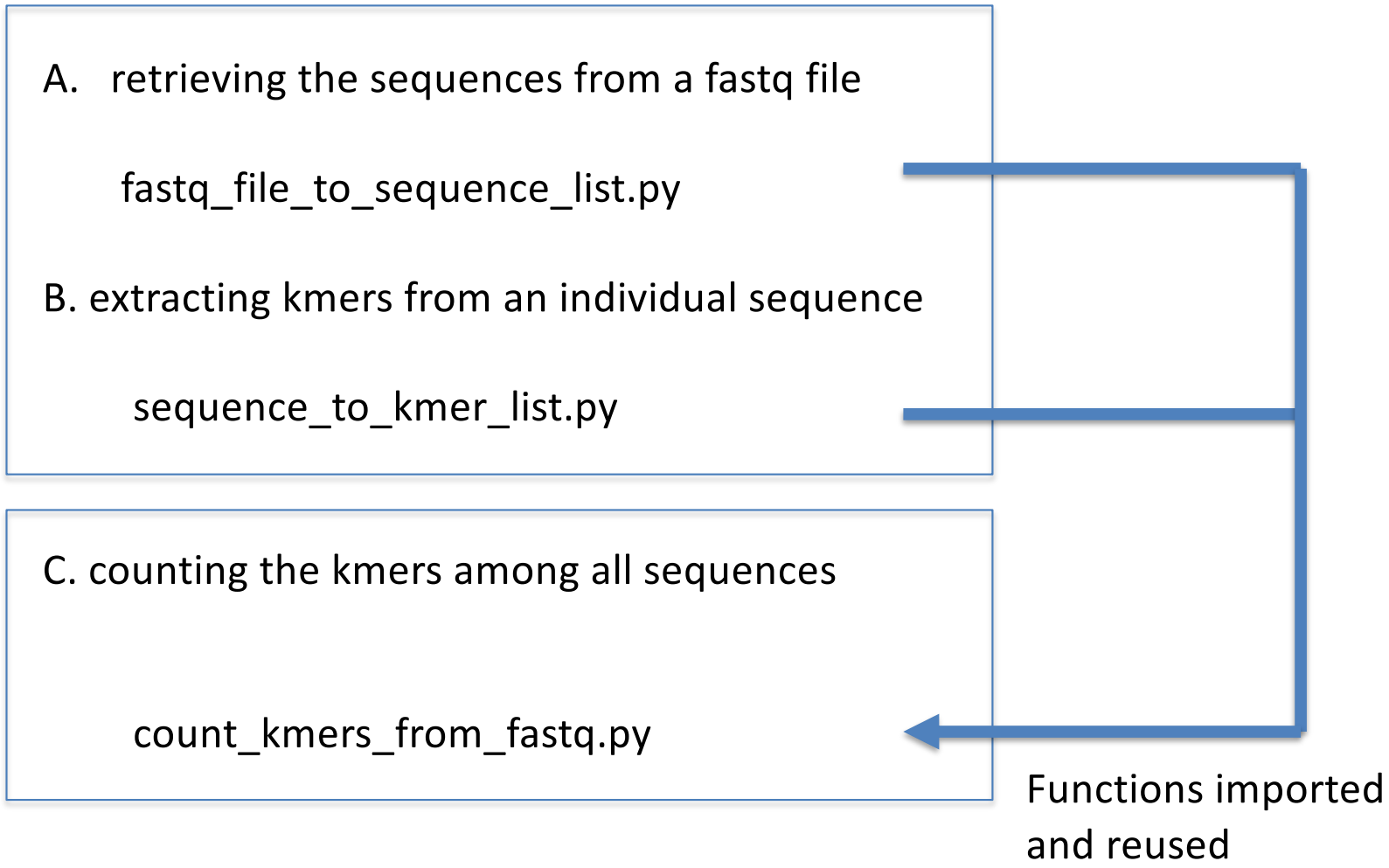
Function definitions

Driver

Driver only run if the script containing it is directly executed (not when imported)

```
#!/usr/bin/env python3

Import sys, os, re

def func_A(myvar):
    …
    …
    return something

def func_B(myvar):
   …
   …
   return something



def main():
     # test func_A
     …..

      # test func_B
      ….


If __name__==‘__main__’:
    main()
```

Build in 3 phases – demonstrating function reuse



Useful structure of a python script for shareable functions

my_module.py (can run directly or use as a function library)

```
#!/usr/bin/env python3

import sys

def decorate_name_val(name_val):
    decorated = " ".join([get_unicorn(), name_val, get_unicorn()])
    return(decorated)

def get_unicorn():
    # returns unicorn symbol
    return("\U0001f984")


if __name__ == '__main__':
    print("my_module.py running as driver: ", decorate_name_val(__name__))
```

executed

```
(base) wm4ca-d15:whatsupwith__name__? bhaas$ ./my_module.py
my_module.py running as driver:  🦄 __main__ 🦄
```

When a script is executed, the special __name__ variable is automatically set to '__main__' within the scope of that file.

https://github.com/trinityrnaseq/CSHLProgForBio/tree/main/whatsupwith__name__%3F

Useful structure of a python script for shareable functions

my_module.py (can run directly or use as a function library)

```
#!/usr/bin/env python3

import sys

def decorate_name_val(name_val):
    decorated = " ".join([get_unicorn(), name_val, get_unicorn()])
    return(decorated)

def get_unicorn():
    # returns unicorn symbol
    return("\U0001f984")


if __name__ == '__main__':
    print("my_module.py running as driver: ", decorate_name_val(__name__))
```

```
(base) wm4ca-d15:whatsupwith__name__? bhaas$ ./my_module.py
my_module.py running as driver:  🦄 __main__ 🦄
```

my_script.py *imported*

```
#!/usr/bin/env python

from my_module import *

print("my_script.py: checking __name__ value:", decorate_name_val(__name__))
```

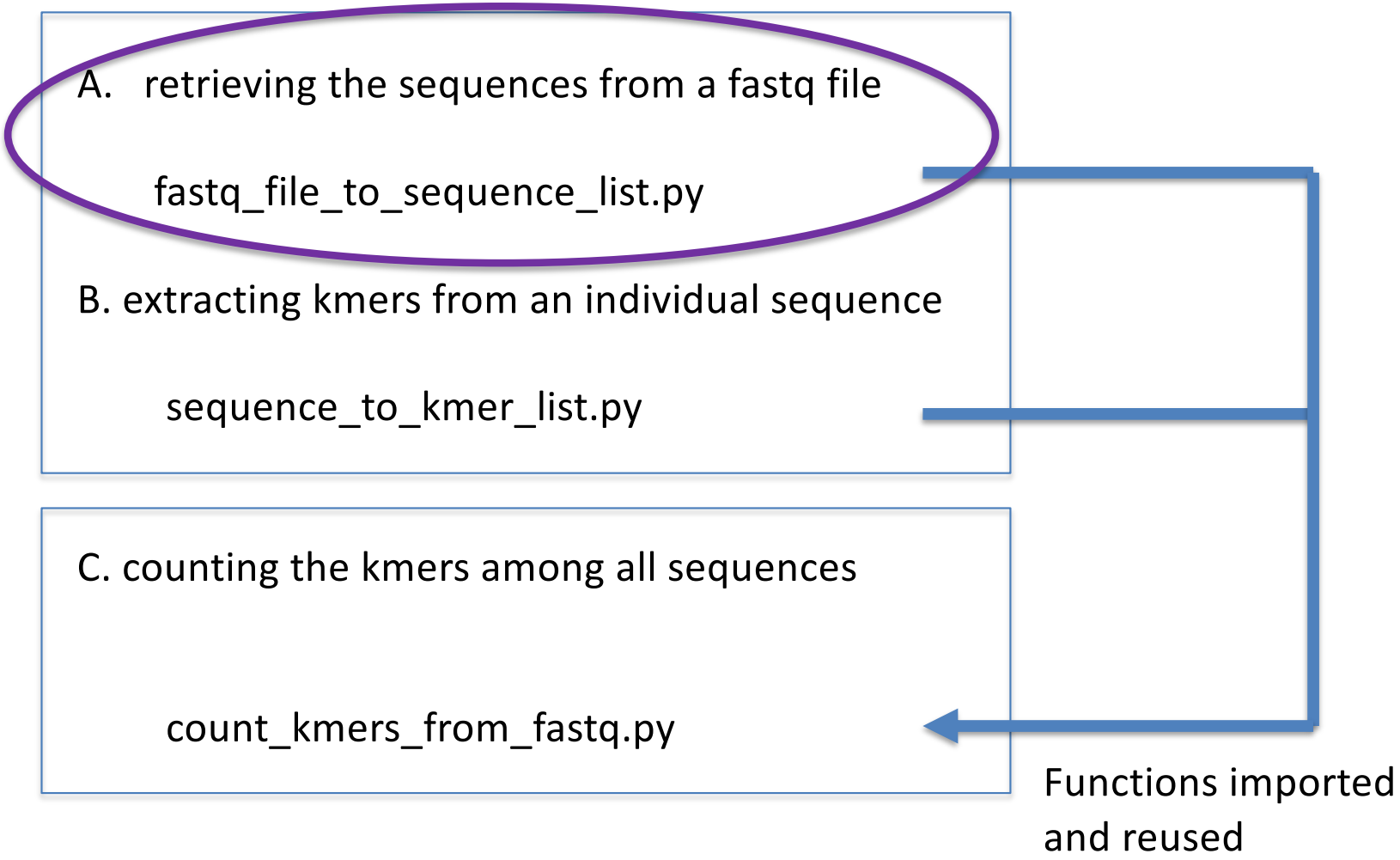
executed

Uses imported decorate_name_val()

```
(base) wm4ca-d15:whatsupwith__name__? bhaas$ ./my_script.py
my_script.py: checking __name__ value: 🦄 __main__ 🦄
```

https://github.com/trinityrnaseq/CSHLProgForBio/tree/main/whatsupwith__name__%3F

Build in 3 phases – demonstrating function reuse



Part A: Retrieve sequences from a fastq file

Write a python script that retrieves a list of all read sequences from a fastq file.

A script fastq_file_to_sequence_list.py is provided as a starting point. Fill in the missing code.

A fastq file <reads.fq> is provided as input.

The script usage is:

```
usage: ./fastq_file_to_sequence_list.py filename.fastq num_seqs_show
```

Running it like so:

```
fastq_file_to_sequence_list.py reads.fq 10
```

Should produce the following output:

['ACTGCATCCTGGAAAGAATCAATGGTGGCCGGAAAGTGTTTTTCAAATACAAGAGTGACAATGTGCCCTGTTGTTT',
'GTAATTTCCGTACCTGCCACAGTGTGGGCTCACCCTGCTTAGAGGACAGGGAAGGACCCTAAAGGTAGGCTGATGC',
'CTGGGCTGCAGCTAAGTTCTCTGCATCCTCCTTCTTGCTTGTGGCTGGGAAGAAGACAATGTTGTCGATGGTCTGG',
'CACGTTTTCTAAGCAGTTTGTACCAGATCGTGCTAACTGCTCATTGTCTTGTTGTACACACCAGTAAAGCTGGGCA',
'TGCTCATTGTCTTGTTGTACACACCAGTAAAGCTGGGCAAAAATATCATCCAAAAGTACATCGCTGAGAACTCCTA',
'CCCACCTGAAAACATTTTCTACATCCACTGTTATATGGAATGCTTGATAAGCTTTTCATTCTAACCATCAGAGCAC',
'TCTGAATAAGTCCTGCCACCAATGTTTTTCATAAGTGTGGCCATATGTTTTCATTATTTCAAACATTACTGTTAAG',
'CTCCGTTTTTTGAGAGTGCAACACATAGATACTGCTTGATAGCATTAATAAACATCTCATTTGTCCTGAAAACAGG',
'GCCTGAGTGTGCAAAAATCTTCAGAGTAAGAATACCATAGTTGCTAAATATCTTTTACCATGAGCAATAATTTTTT',
'TCTGGTGCAGCTAGATGGAATACTGAGAAAATGTTCTTCCATCCTGAACGAATATTTGCAGCCTGAGAATTAACCA']

GitHub link: https://github.com/trinityrnaseq/CSHLProgForBio/tree/main/Exercise-counting_kmers#part-a-retrieve-sequences-from-a-fastq-file

A. fastq_file_to_sequence_list.py

Reusable function part:

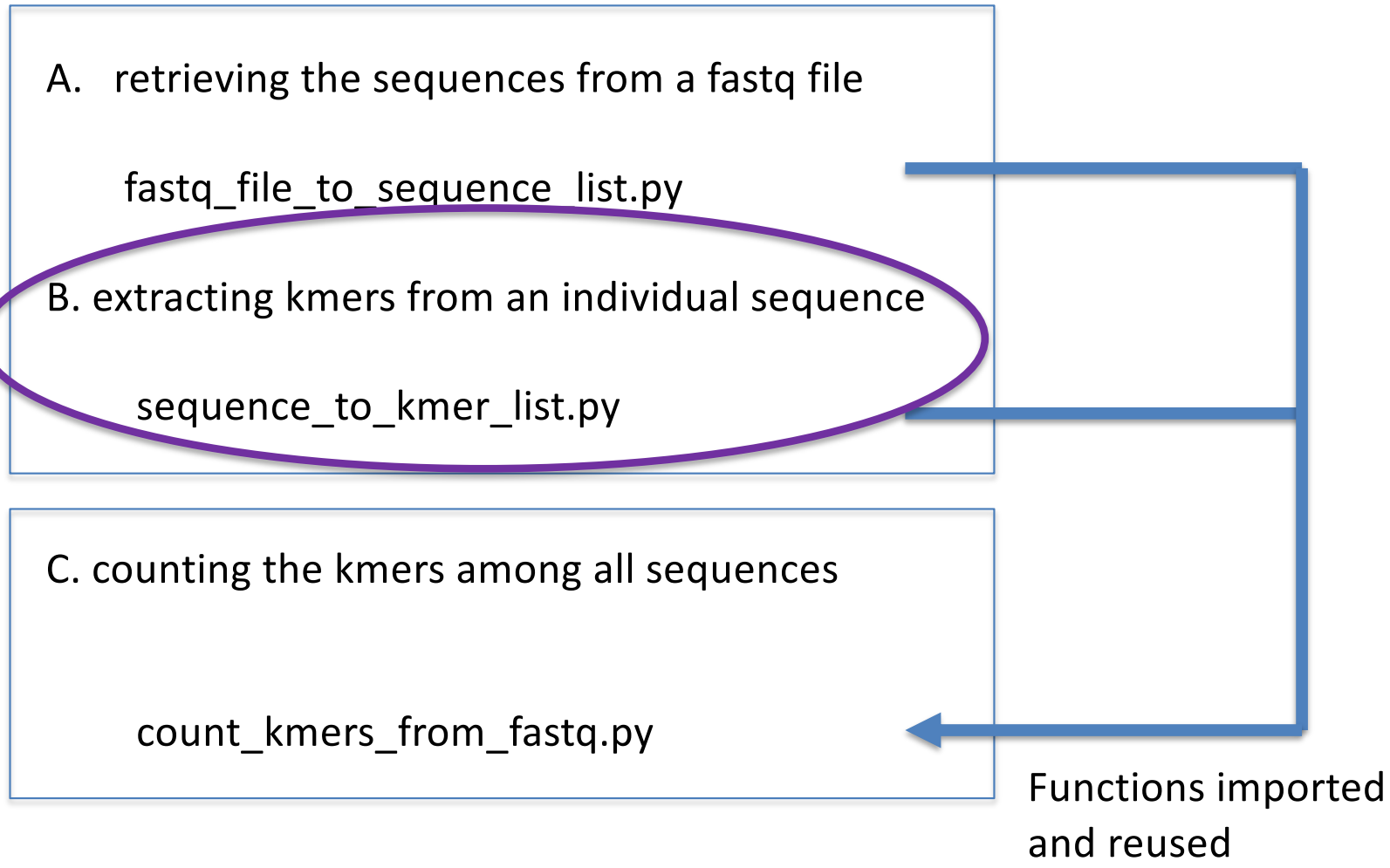
```
6
7     ## method: seq_list_from_fastq_file(fastq_filename)
8     ##
9     ##  Extracts the sequence lines from a fastq file and returns a list
10    ##  of the sequence lines
11    ##
12    ##  input parameters:
13    ##
14    ##  fastq_filename :  name of the fastq file (type: string)
15    ##
16    ##  returns seq_list : list of read sequences.
17    ##                   ie.  ["GATCGCATAG", "CGATGCAG", ...]
18
19    def seq_list_from_fastq_file(fastq_filename):
20
21        seq_list = list()
22
23        ## begin your code
24
25
26
27
28
29
30        ## end your code
31
32        return seq_list
33
```

Driver part for testing:

```
35
36    def main():
37
38        progname = sys.argv[0]
39
40        usage = "\n\n\tusage: {} filename.fastq num_seqs_show\n\n\n".format(progname)
41
42        if len(sys.argv) < 3:
43            sys.stderr.write(usage)
44            sys.exit(1)
45
46        # capture command-line arguments
47        fastq_filename = sys.argv[1]
48        num_seqs_show = int(sys.argv[2])
49
50        seq_list = seq_list_from_fastq_file(fastq_filename)
51
52        print(seq_list[0:num_seqs_show])
53
54        sys.exit(0)  # always good practice to indicate worked ok!
55
56
57
58    if __name__ == '__main__':
59        main()
60
```

https://github.com/trinityrnaseq/CSHLProgForBio/blob/main/Exercise-counting_kmers/fastq_file_to_sequence_list.py

Build in 3 phases – demonstrating function reuse



Part B: Extracting kmers from a sequence

Write a python script to extract all kmers of a specified length from a nucleotide sequence.

A script fastq_file_to_sequence_list.py is provided as a starting point. Fill in the missing code.

The script usage is:

```
usage: ./sequence_to_kmer_list.py sequence kmer_length
```

Running it like so:

```
sequence_to_kmer_list.py ACTGCATCCTGGAAAGAATCAATGGTGGCCGGAAAGTGTTTTTCAAATACAAGAGTGACAATGTGCCCTGTTGTTT 6
```

Should produce the following output:

['ACTGCA', 'CTGCAT', 'TGCATC', 'GCATCC', 'CATCCT', 'ATCCTG', 'TCCTGG', 'CCTGGA', 'CTGGAA', 'TGGAAA', 'GGAAAG', 'GAAAGA', 'AAAGAA', 'AAGAAT', 'AGAATC', 'GAATCA', 'AATCAA', 'ATCAAT', 'TCAATG', 'CAATGG', 'AATGGT', 'ATGGTG', 'TGGTGG', 'GGTGGC', 'GTGGCC', 'TGGCCG', 'GGCCGG', 'GCCGGA', 'CCGGAA', 'CGGAAA', 'GGAAAG', 'GAAAGT', 'AAAGTG', 'AAGTGT', 'AGTGTT', 'GTGTTT', 'TGTTTT', 'GTTTTT', 'TTTTTC', 'TTTTCA', 'TTTCAA', 'TTCAAA', 'TCAAAT', 'CAAATA', 'AAATAC', 'AATACA', 'ATACAA', 'TACAAG', 'ACAAGA', 'CAAGAG', 'AAGAGT', 'AGAGTG', 'GAGTGA', 'AGTGAC', 'GTGACA', 'TGACAA', 'GACAAT', 'ACAATG', 'CAATGT', 'AATGTG', 'ATGTGC', 'TGTGCC', 'GTGCCC', 'TGCCCT', 'GCCCTG', 'CCCTGT', 'CCTGTT', 'CTGTTG', 'TGTTGT', 'GTTGTT', 'TTGTTT']

B. sequence_to_kmer_list.py

Reusable function part

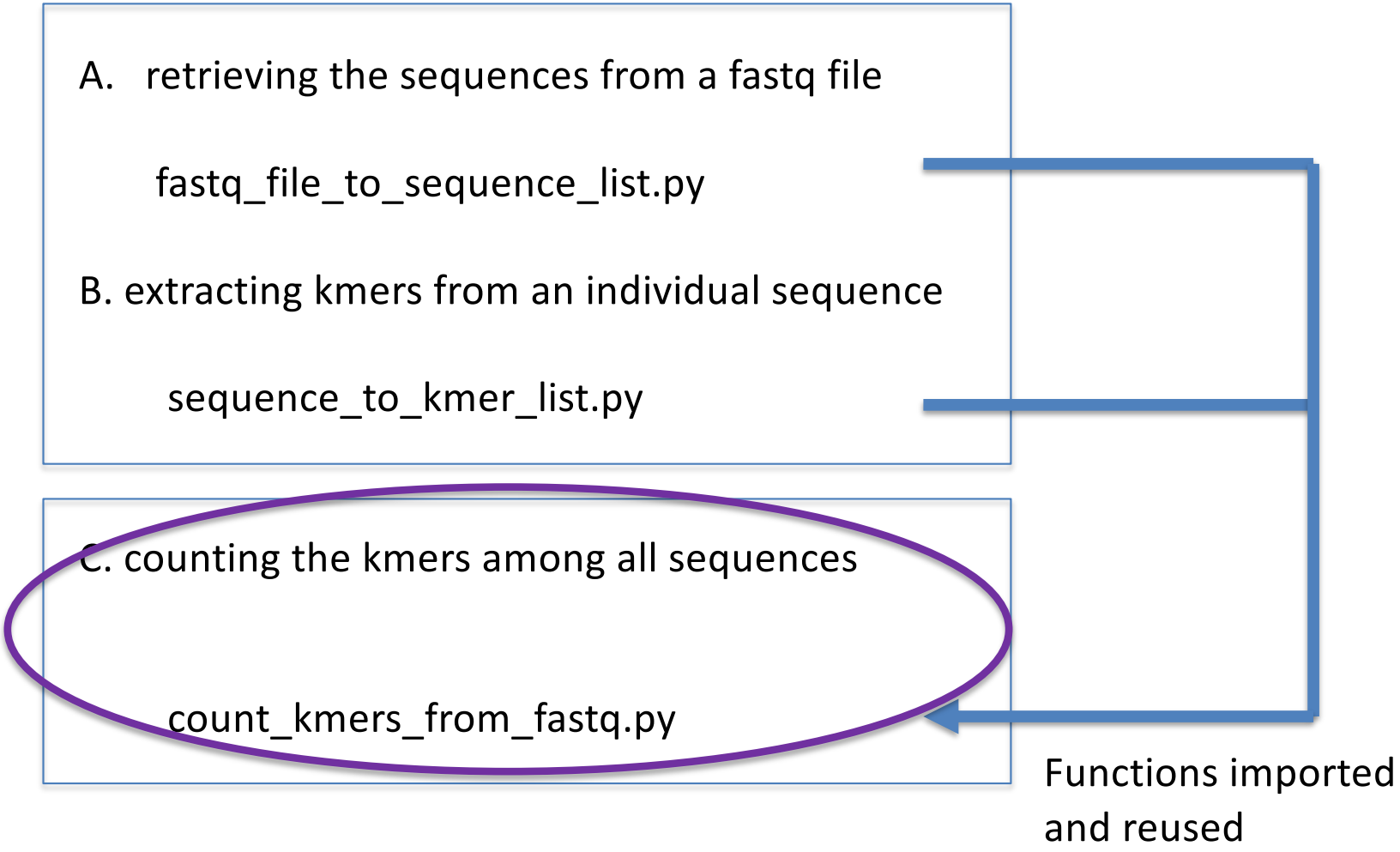
```
6
7     ## method: sequence_to_kmer_list(sequence, kmer_length)
8     ##
9     ##  Extracts all kmers of a specified length from a sequence
10    ##
11    ##  ie.  sequence: GATCGATCGATCGA
12    ##   and given kmer_length = 4
13    ##   would yield
14    ##                GATC
15    ##                 ATCG
16    ##                  TCGA
17    ##                   .... and so forth
18    ##
19    ##  input parameters:
20    ##
21    ##  seuqence : nucleotide sequence (type: string)
22    ##
23    ##  returns kmer_list : list of kmer sequences.
24    ##                   ie.  ["GATC", "ATCG", ...]
25
26    def sequence_to_kmer_list(sequence, kmer_length):
27
28        kmers_list = list()
29
30        ## begin your code
31
32
33
34
35
36        ## end your code
37
38        return kmers_list
39
40
```

Driver part for testing:

```
41
42    def main():
43
44        progname = sys.argv[0]
45
46        usage = "\n\n\tusage: {} sequence kmer_length\n\n\n".format(progname)
47
48        if len(sys.argv) < 3:
49            sys.stderr.write(usage)
50            sys.exit(1)
51
52        # capture command-line arguments
53        sequence = sys.argv[1]
54        kmer_length = int(sys.argv[2])
55
56        kmers  = sequence_to_kmer_list(sequence, kmer_length)
57
58        print(kmers)
59
60        sys.exit(0)  # always good practice to indicate worked ok!
61
62
63
64    if __name__ == '__main__':
65        main()
66
```

https://github.com/trinityrnaseq/CSHLProgForBio/blob/main/Exercise-counting_kmers/sequence_to_kmer_list.py

Build in 3 phases – demonstrating function reuse



Part C: Counting all kmers from all sequences in a fastq file

Now, let's count all kmers in all sequences. We can leverage each of the methods implemented above. Because of the way we wrote the above scripts, we can leverage them as a code library and simply import them for use in a new script.

Use the script count_kmers_from_fastq.py as the starting point. You'll see at the top of this script:

```
from sequence_to_kmer_list import *
from fastq_file_to_sequence_list import *
```

Those lines import the methods we implemented earlier so that we can just reuse them without having to rewrite or copy/paste any code in this new script.

The usage of our script is:

```
    usage: ./count_kmers_from_fastq.py filename.fastq kmer_length num_top_kmers_show
```

And when we run it like so:

```
    count_kmers_from_fastq.py reads.fq 6 10
```

It should produce the output:

```
TTTTTT: 3085
CTTCTT: 2550
AAAAAA: 2498
CTGCTG: 2446
AGCTGG: 2400
CAGCAG: 2265
CAGCTG: 2243
TCTTCT: 2208
CTGGAG: 2174
TGCTGT: 2156
```

C. count_kmers_from_fastq.py

Import earlier functions

```
#!/usr/bin/env python

import os, sys

from sequence_to_kmer_list import *
from fastq_file_to_sequence_list import *

```

New function to implement

```

## method: count_kmers(kmer_list)
##
##  Counts the frequency of each kmer in the given list of kmers
##
##  input parameters:
##
##  kmer_list : list of kmers (type: list)
##              ie.  ["GATC", "TCGA", "GATC", ...]
##
##
##  returns kmer_counts_dict : dict containing ( kmer : count )
##                   ie.  {  "GATC" : 2,
##                           "TCGA" : 1,
##                            ...      }


def count_kmers(kmer_list):

    kmer_count_dict = dict()

    ##################
    ## Step 2:
    ## begin your code


    Step 2


    ## end your code
    ###############

    return kmer_count_dict

```

Main code block

```

def main():

    progname = sys.argv[0]

    usage = "\n\n\tusage: {} filename.fastq kmer_length num_top_kmers_show\n\n\n".format(
        progname
    )

    if len(sys.argv) < 4:
        sys.stderr.write(usage)
        sys.exit(1)

    # capture command-line arguments
    fastq_filename = sys.argv[1]
    kmer_length = int(sys.argv[2])
    num_top_kmers_show = int(sys.argv[3])

    seq_list = seq_list_from_fastq_file(fastq_filename)

    all_kmers = list()

    ######################
    ## Step 1:
    ## begin your code, populate 'all_kmers' list with the
    ## collection of kmers from all sequences

    Step 1

    ## end your code
    ######################

    kmer_count_dict = count_kmers(
        all_kmers
    )  # see step 2 above. You implement this. :-)

    unique_kmers = list(kmer_count_dict.keys())

    ########################
    ## Step 3: sort unique_kmers by abundance descendingly
    ## (Note, you can run and test without first implementing Step 3)
    ## begin your code       hint: see the built-in 'sorted' method documentation


    ## end your code

    Step 3

    ## printing the num top kmers to show
    top_kmers_show = unique_kmers[0:num_top_kmers_show]

    for kmer in top_kmers_show:
        print("{}: {}".format(kmer, kmer_count_dict[kmer]))

    sys.exit(0)  # always good practice to indicate worked ok!


if __name__ == "__main__":
    main()

```

https://github.com/trinityrnaseq/CSHLProgForBio/blob/main/Exercise-counting_kmers/count_kmers_from_fastq.py

Extra Credit

It should produce the output:

```
TTTTTT: 3085
CTTCTT: 2550
AAAAAA: 2498
CTGCTG: 2446
AGCTGG: 2400
CAGCAG: 2265
CAGCTG: 2243
TCTTCT: 2208
CTGGAG: 2174
TGCTGT: 2156
```

Extra credit section:

If you've accomplished the above, here's another challenge!

Note that the top-most kmer is of low complexity. If we are going to perform downstram operations like assembly and want to start with a seed kmer, we might want to avoid low complexity kmers as they would lack specificity.

Challenge: include another method that computes the complexity of each kmer using Shannon's Entropy (example: see: https://en.wikipedia.org/wiki/Sequence_logo#Logo_creation), and picture the kmer as representing one column of the seqlogo for which you would get one entropy calculation.

Add the entropy value as another column in the above printing.

Shannon Entropy for a Nucleotide Sequence (in bits)

$$H = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Where:

- n: number of unique nucleotides (typically 4: A, C, G, T)
- p_i: probability (frequency) of nucleotide i in the sequence
- $\log_2$: base-2 logarithm (entropy measured in bits)

Interpretation:

- $H = 0$: completely uniform (only one nucleotide present)
- $H = 2$: maximum entropy for DNA (all nucleotides equally likely, $p_i = 0.25$)

Example:

$$\text{Sequence: AACGT}$$

$$p_A = 0.4, \quad p_C = p_G = p_T = 0.2$$

$$H = -[0.4 \log_2(0.4) + 3(0.2 \log_2(0.2))] \approx 1.92 \text{ bits per symbol.}$$

Higher entropy indicates greater sequence complexity or diversity, while lower entropy reflects bias or repetitiveness (e.g., homopolymers)

No use of coding help from artificial intelligence (AI) before dinner!!

After dinner, we'll unleash the power of AI