RNA-Seq Alignment/Assembly using STAR

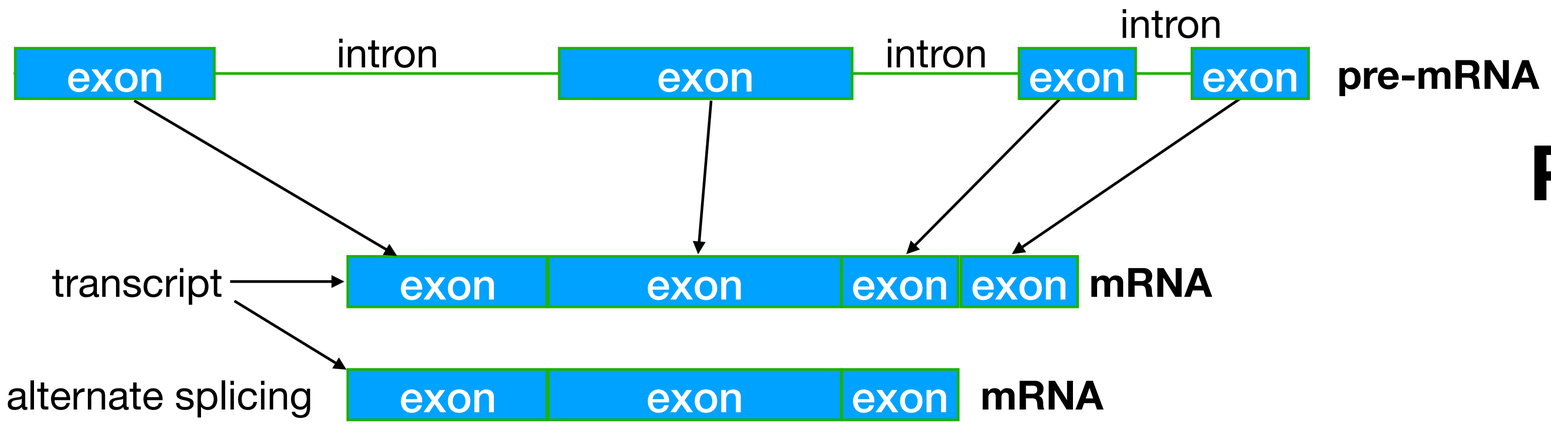
CS 4390/5390

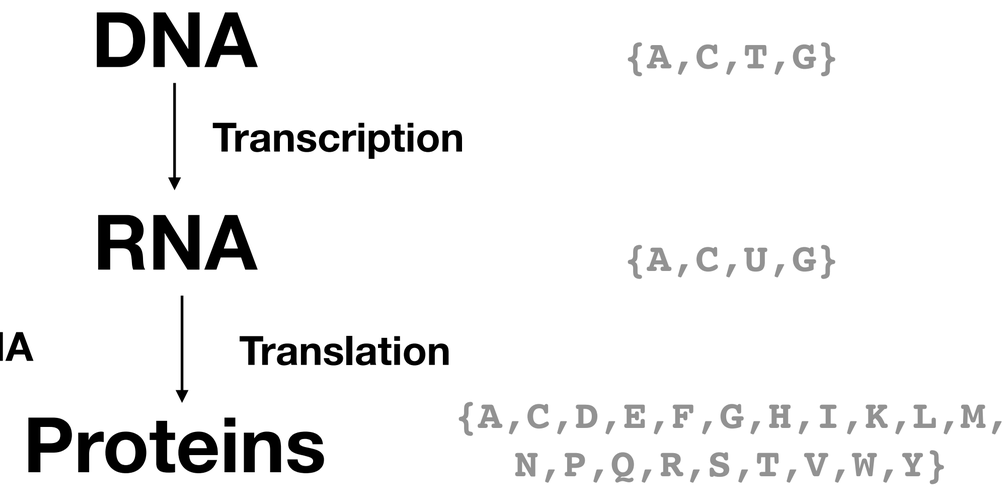
and TopHat if we have time.

The Central Dogma

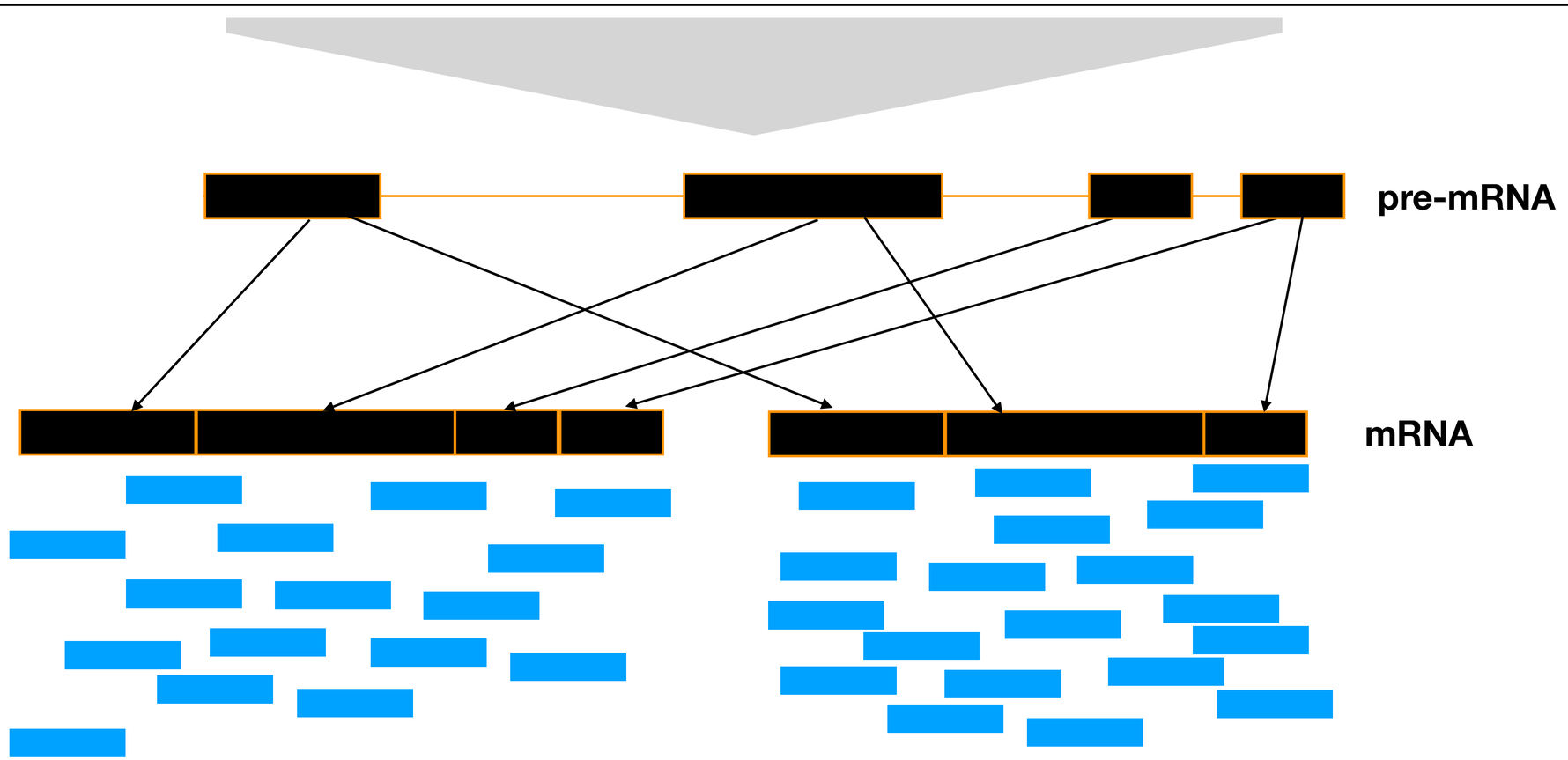
RNA

- pre-mRNA undergo splicing to remove the *introns* and leave only (some) *exons*
- some RNA perform functions on their own and are not spliced, called ncRNA (non-coding RNA)





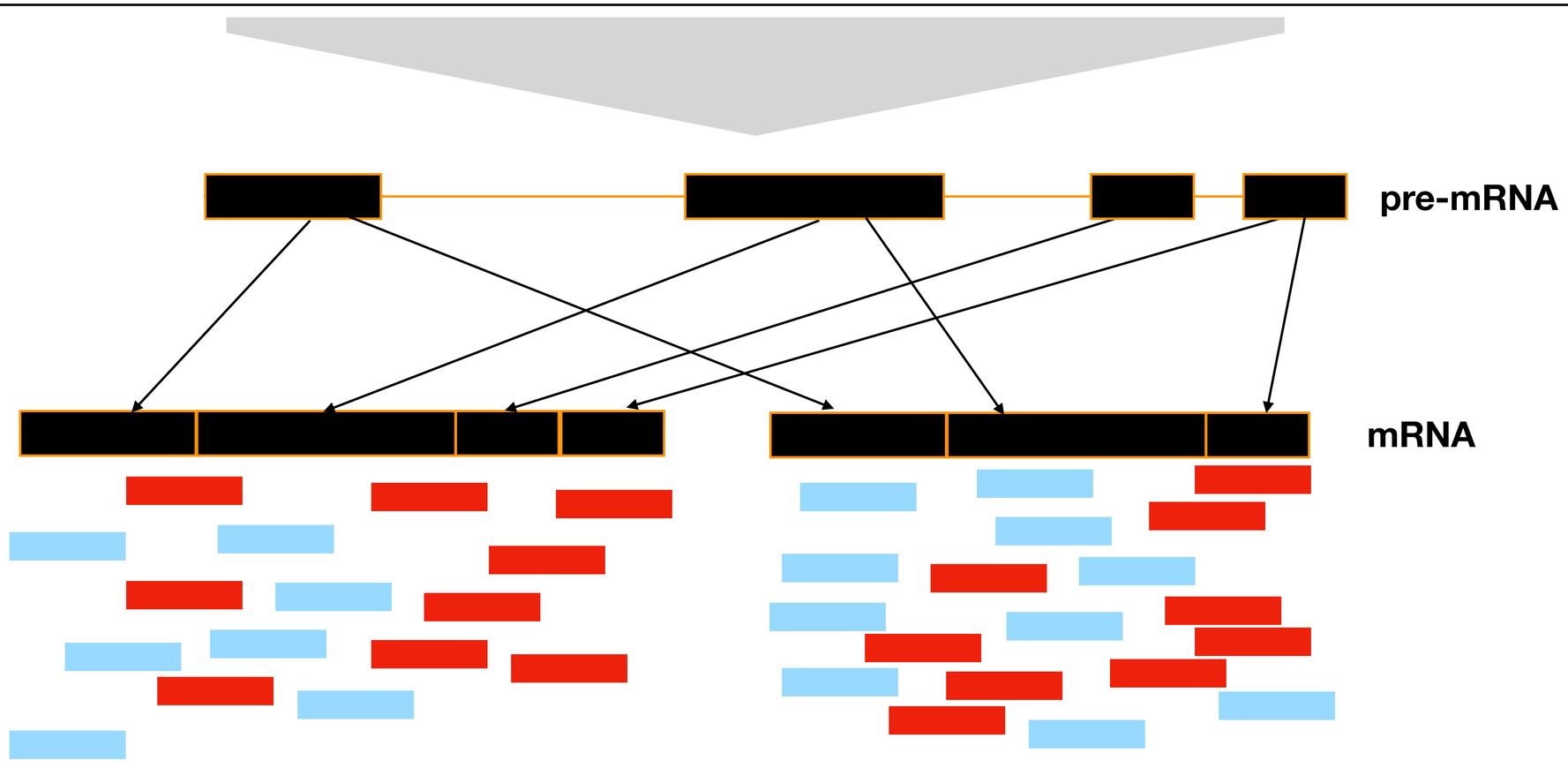
Sequencing Applications



RNA sequencing

adapted from figure 1.2 in Mäkinen, *et al.* 2015

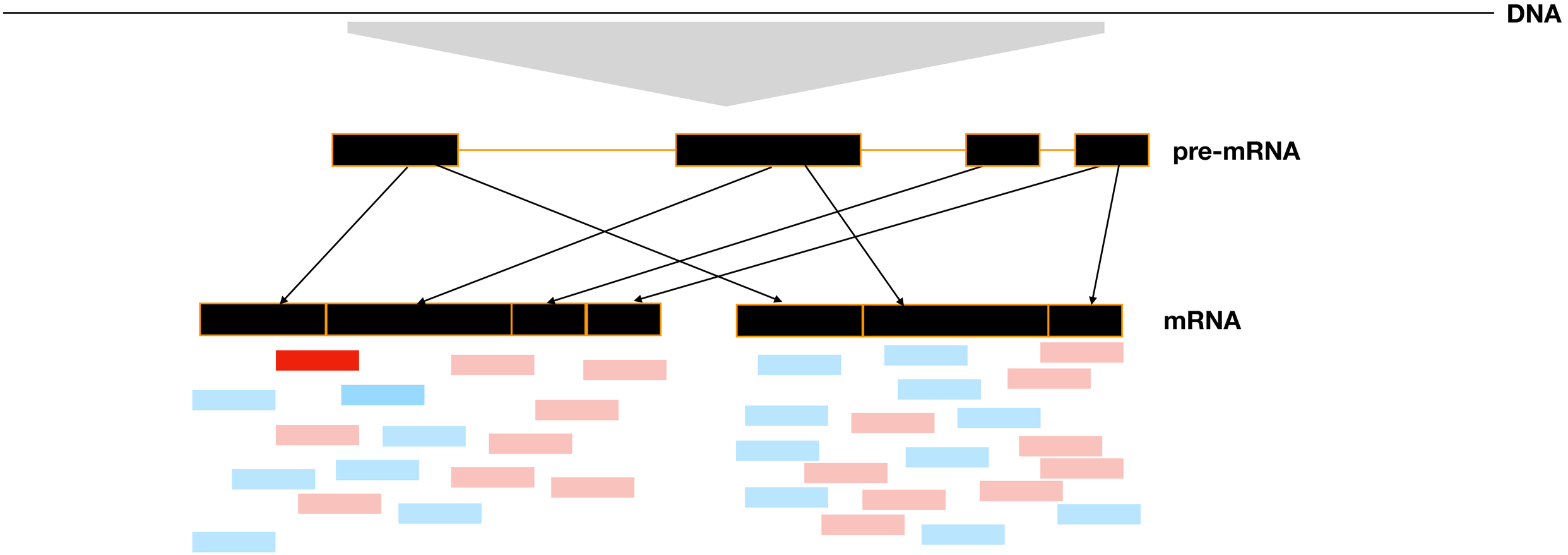
Sequencing Applications



RNA sequencing

adapted from figure 1.2 in Mäkinen, *et al.* 2015

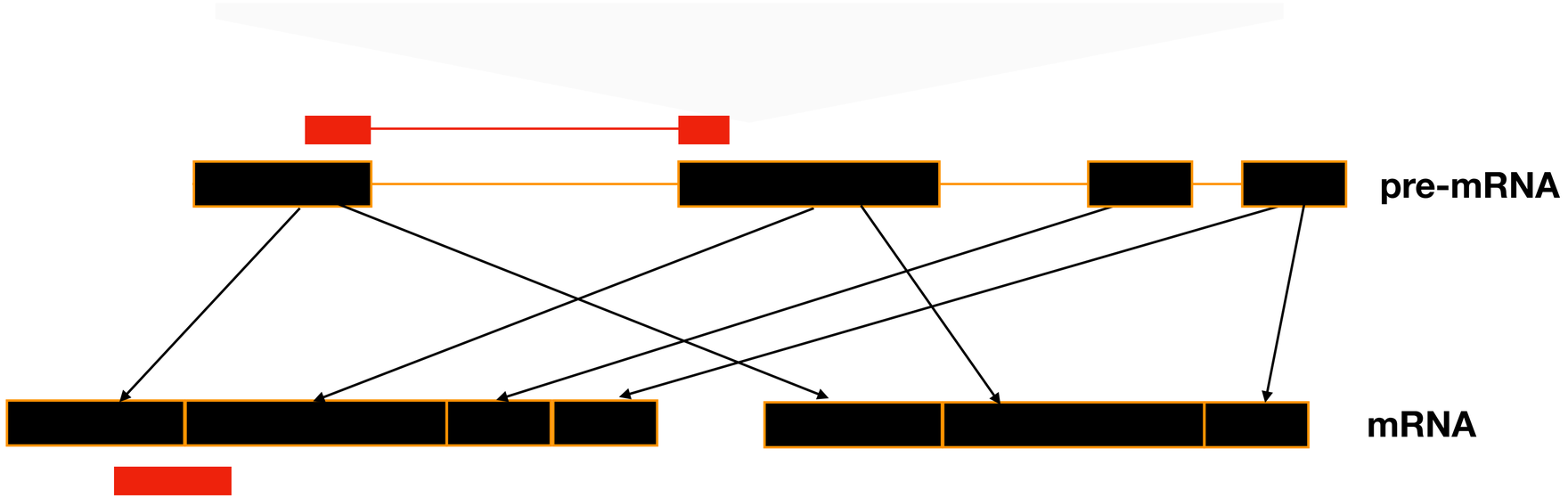
Sequencing Applications



RNA sequencing

adapted from figure 1.2 in Mäkinen, *et al.* 2015

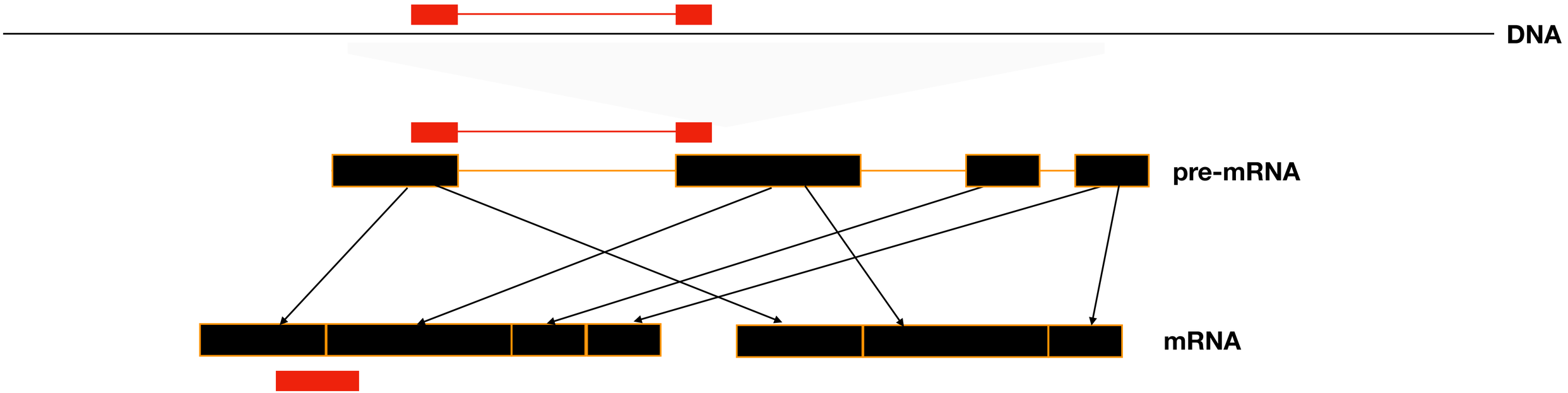
Sequencing Applications



RNA sequencing

adapted from figure 1.2 in Mäkinen, *et al.* 2015

Sequencing Applications



RNA sequencing

adapted from figure 1.2 in Mäkinen, *et al.* 2015

How do you solve this?

How do you solve this?

Split the read into chunks, and align those seperately

- can be done at random positions or predicted intron boundaries
- the chunks end up being small

How do you solve this?

Split the read into chunks, and align those seperately

- can be done at random positions or predicted intron boundaries
- the chunks end up being small

Align to the known transcriptome rather than the genome

- can miss novel transcripts (ones we have not seen before)

How do you solve this?

Split the read into chunks, and align those seperately

- can be done at random positions or predicted intron boundaries
- the chunks end up being small

Align to the known transcriptome rather than the genome

- can miss novel transcripts (ones we have not seen before)

Design a new aligner that takes these complications into account

How do you solve this?

Split the read into chunks, and align those seperately

- can be done at random positions or predicted intron boundaries
- the chunks end up being small

Align to the known transcriptome rather than the genome

- can miss novel transcripts (ones we have not seen before)

Design a new aligner that takes these complications into account

Some combination of the above techniques

Spliced Transcripts Alignment to a Reference (STAR)

Designed to align non-contiguous sections of a read, directly to the reference genome

Consists of two major steps:

- Seed searching
- clustering, stitching, and scoring

BIOINFORMATICS ORIGINAL PAPER

Vol. 29 no. 1 2013, pages 15–21
doi:10.1093/bioinformatics/bts635

Sequence analysis

Advance Access publication October 25, 2012

STAR: ultrafast universal RNA-seq aligner

Alexander Dobin[1,*], Carrie A. Davis[1], Felix Schlesinger[1], Jorg Drenkow[1], Chris Zaleski[1], Sonali Jha[1], Philippe Batut[1], Mark Chaisson[2] and Thomas R. Gingeras[1]

[1]Cold Spring Harbor Laboratory, Cold Spring Harbor, NY, USA and [2]Pacific Biosciences, Menlo Park, CA, USA

Associate Editor: Inanc Birol

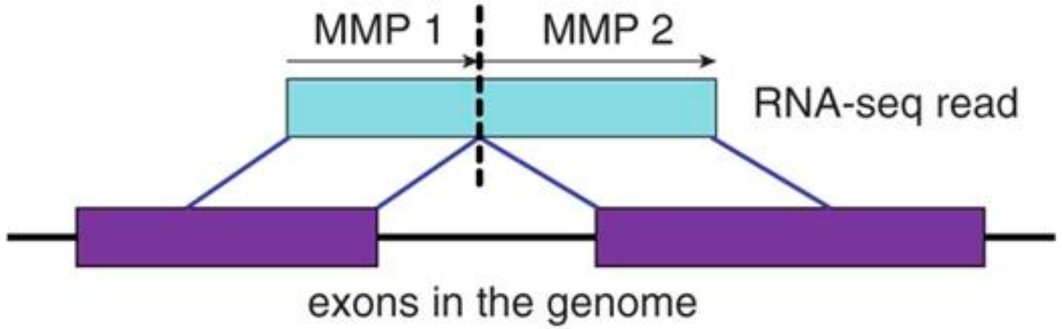
Seed Searching

Maximal Mappable Prefix (*MMP*) for read *R,* read start location *i*, and genome *G*:

- the longest substring $R[i \ldots (i + MML - 1)]$
- such that there exists some set $J = \{j_1, j_2, \ldots, j_n\}$ where for all $j_k \in J$

$$R[i \ldots (i+MML-1)] = G[j \ldots (j_k+MML-1)]$$

- where *MML* is the Maximal Mapping Length



Seed Searching

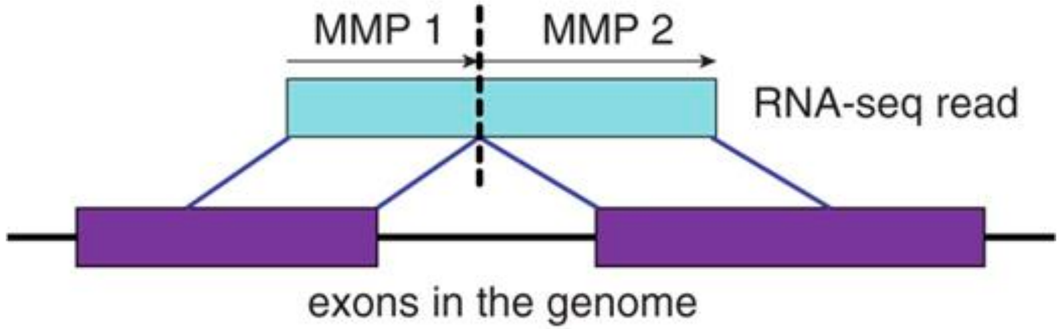
Maximal Mappable Prefix (*MMP*) for read *R,* read start location *i*, and genome *G*:

- the longest substring $R[i \ldots (i + MML - 1)]$
- such that there exists some set $J = \{j_1, j_2, \ldots, j_n\}$ where for all $j_k \in J$

$$R[i \ldots (i+MML-1)] = G[j \ldots (j_k+MML-1)]$$

- where *MML* is the Maximal Mapping Length

This is similar to a Maximal Exact Match (*MEM*) or Maximal Unique Map (*MUM*), the latter requires that only one location in the genome matches that location. In both cases, the value if *i* is not given.



Seed Searching

Maximal Mappable Prefix (*MMP*) for read *R,* read start location *i*, and genome *G*:

- the longest substring $R[i \ldots (i + MML - 1)]$
- such that there exists some set $J = \{j_1, j_2, \ldots, j_n\}$ where for all $j_k \in J$

$$R[i \ldots (i+MML-1)] = G[j \ldots (j_k+MML-1)]$$

- where *MML* is the Maximal Mapping Length

The basic algorithm is

- map from the start of the read as far as possible
- restart searching from the next position to the right

Seed Searching

Maximal Mappable Prefix (*MMP*) for read *R,* read start location *i*, and genome *G*:

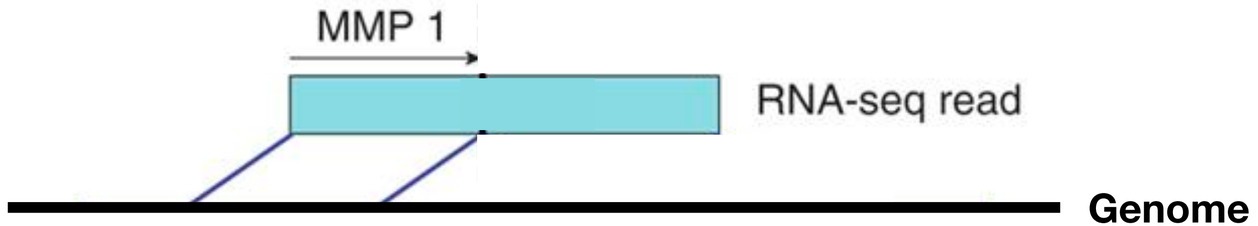
- the longest substring $R[i \ldots (i + MML - 1)]$
- such that there exists some set $J = \{j_1, j_2, \ldots, j_n\}$ where for all $j_k \in J$

$$R[i \ldots (i+MML-1)] = G[j \ldots (j_k+MML-1)]$$

- where *MML* is the Maximal Mapping Length

The basic algorithm is

- map from the start of the read as far as possible
- restart searching from the next position to the right



Seed Searching

Maximal Mappable Prefix (*MMP*) for read *R,* read start location *i*, and genome *G*:

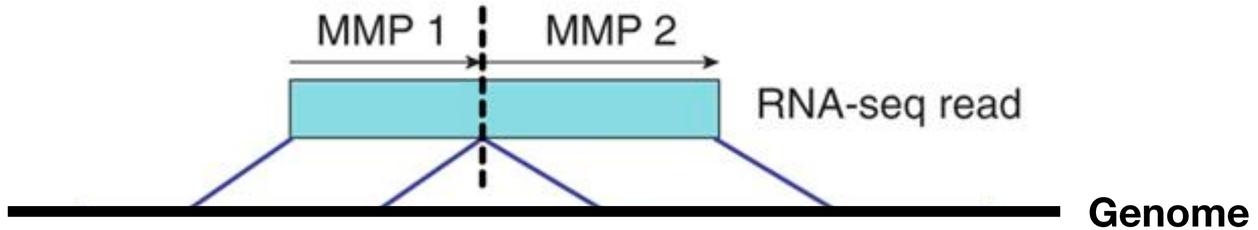
- the longest substring $R[i \ldots (i + MML - 1)]$
- such that there exists some set $J = \{j_1, j_2, \ldots, j_n\}$ where for all $j_k \in J$

$$R[i \ldots (i+MML-1)] = G[j \ldots (j_k+MML-1)]$$

- where *MML* is the Maximal Mapping Length

The basic algorithm is

- map from the start of the read as far as possible
- restart searching from the next position to the right



Seed Searching

Maximal Mappable Prefix (*MMP*) for read *R,* read start location *i*, and genome *G*:

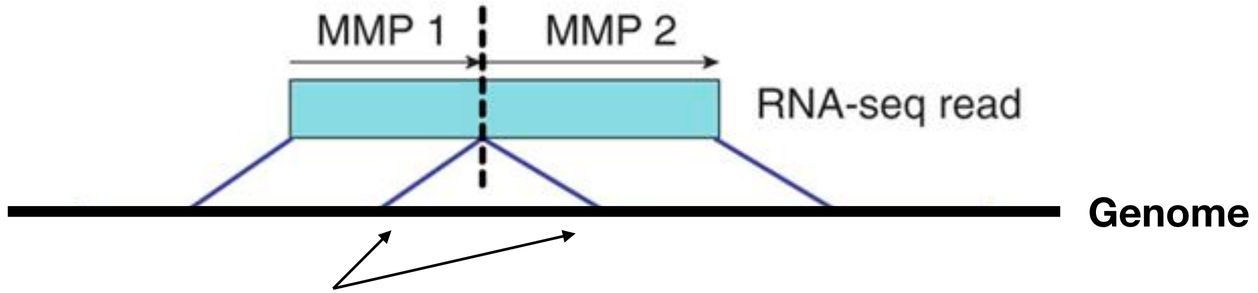
- the longest substring $R[i \ldots (i + MML - 1)]$
- such that there exists some set $J = \{j_1, j_2, \ldots, j_n\}$ where for all $j_k \in J$

$$R[i \ldots (i+MML-1)] = G[j \ldots (j_k+MML-1)]$$

- where *MML* is the Maximal Mapping Length

The basic algorithm is

- map from the start of the read as far as possible
- restart searching from the next position to the right



Seed Searching

Maximal Mappable Prefix (*MMP*) for read *R,* read start location *i*, and genome *G*:

- the longest substring $R[i \ldots (i + MML - 1)]$
- such that there exists some set $J = \{j_1, j_2, \ldots, j_n\}$ where for all $j_k \in J$

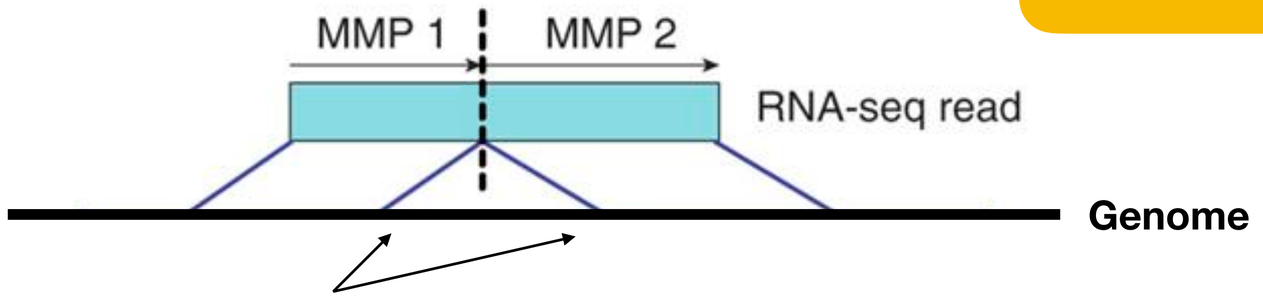
$$R[i \ldots (i+MML-1)] = G[j \ldots (j_k+MML-1)]$$

- where *MML* is the Maximal Mapping Length

The basic algorithm is

- map from the start of the read as far as possible
- restart searching from the next position to the right

The key is that the re-mapping only happens from the end of MMP1 rather than finding all maximal matchings then stitching



the "splice junctions" are inferred from the alignment

Seed Searching

Maximal Mappable Prefix (*MMP*) for read *R,* read start location *i*, and genome *G*:

- the longest substring $R[i \ldots (i + MML - 1)]$
- such that there exists some set $J = \{j_1, j_2, \ldots, j_n\}$ where for all $j_k \in J$

$$R[i \ldots (i+MML-1)] = G[j \ldots (j_k+MML-1)]$$

- where *MML* is the Maximal Mapping Length

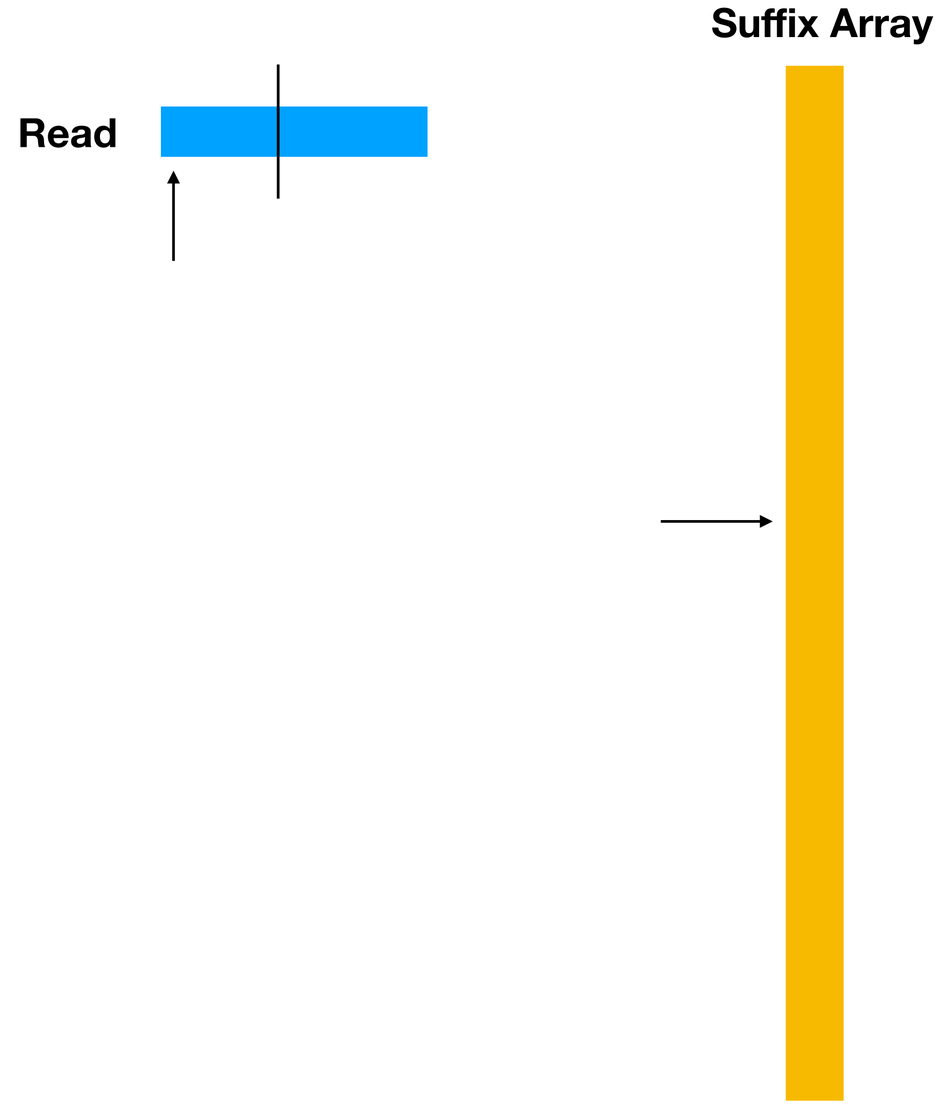
The basic algorithm is

- map from the start of the read as far as possible
- restart searching from the next position to the right

The search is implemented using an uncompressed suffix array(s)

- one for each chromosome

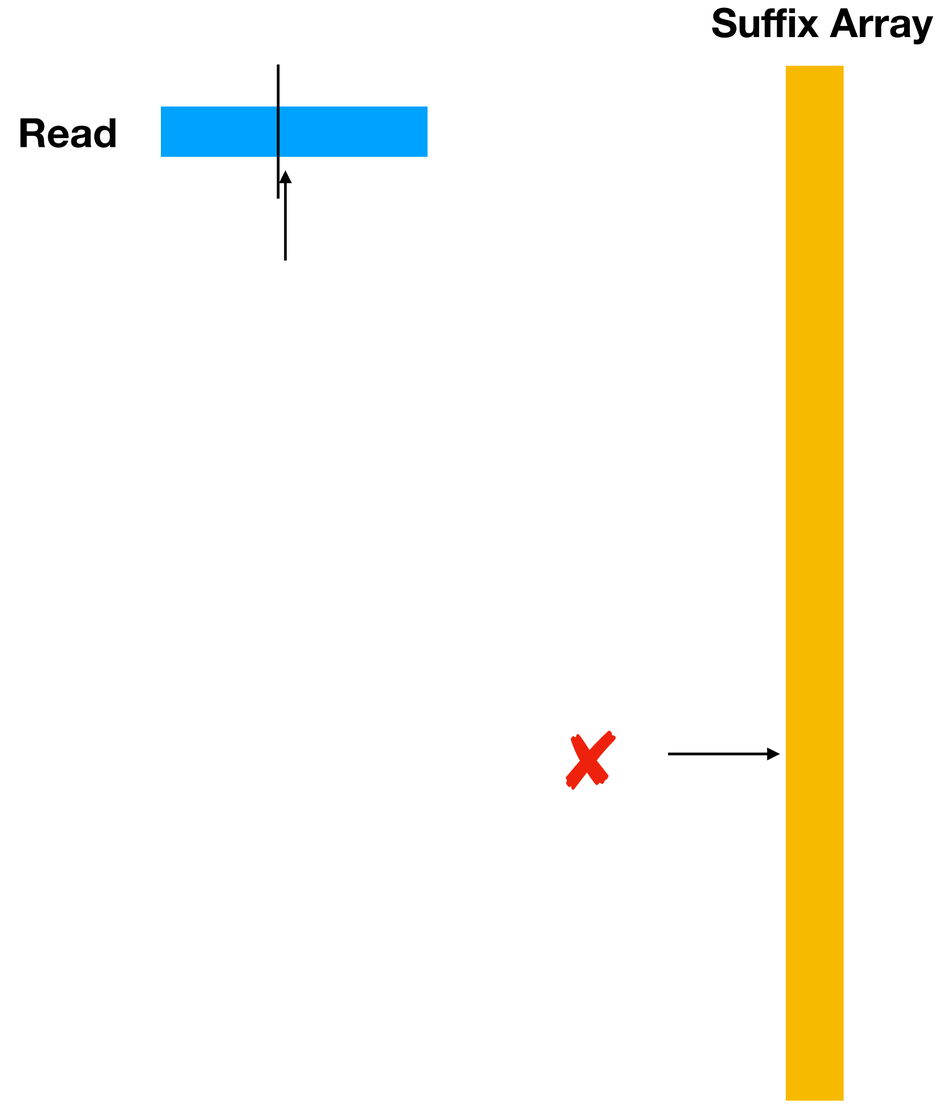
Seed Search



The search is implemented using an uncompressed suffix array(s)

- one for each chromosome
- *MMP* mapping comes "free" with binary search

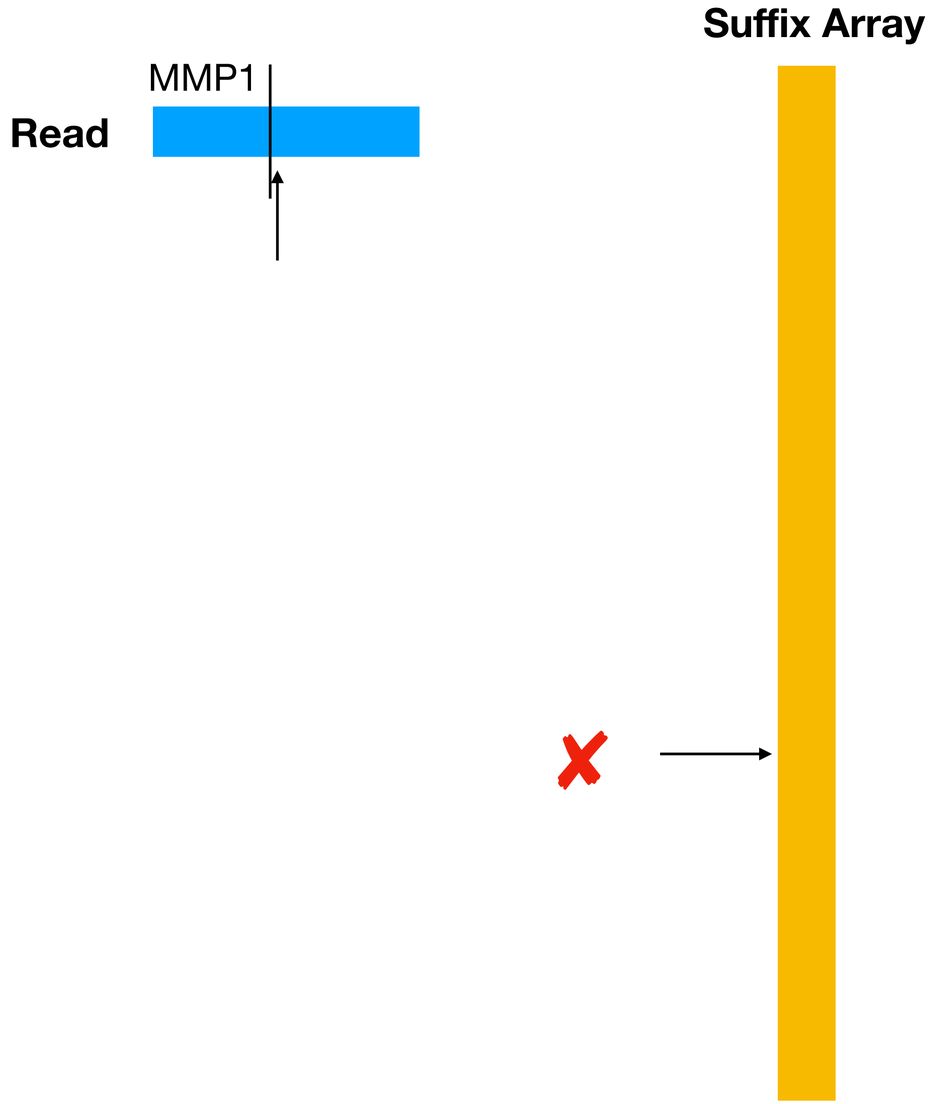
Seed Search



The search is implemented using an uncompressed suffix array(s)

- one for each chromosome
- *MMP* mapping comes "free" with binary search

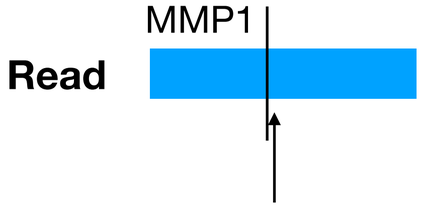
Seed Search



The search is implemented using an uncompressed suffix array(s)

- one for each chromosome
- *MMP* mapping comes "free" with binary search

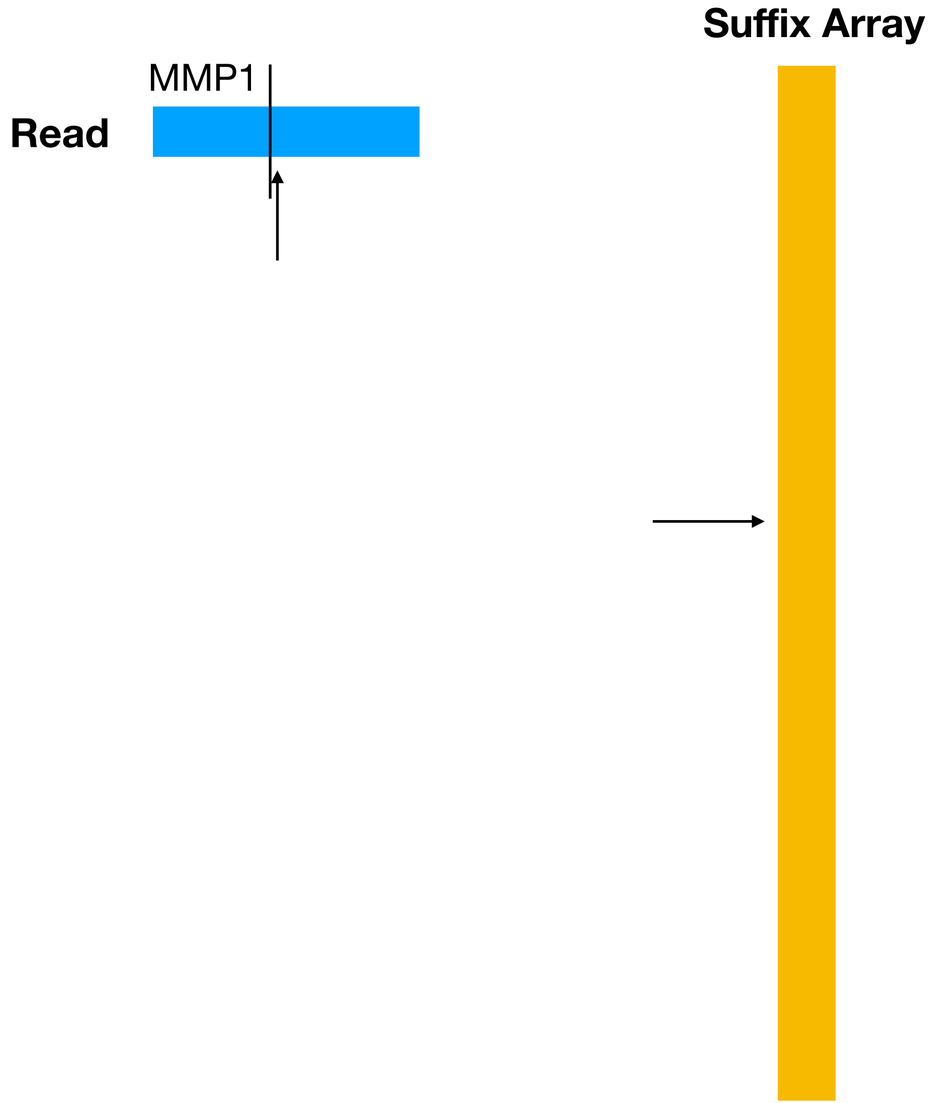
Seed Search



The search is implemented using an uncompressed suffix array(s)

- one for each chromosome
- *MMP* mapping comes "free" with binary search

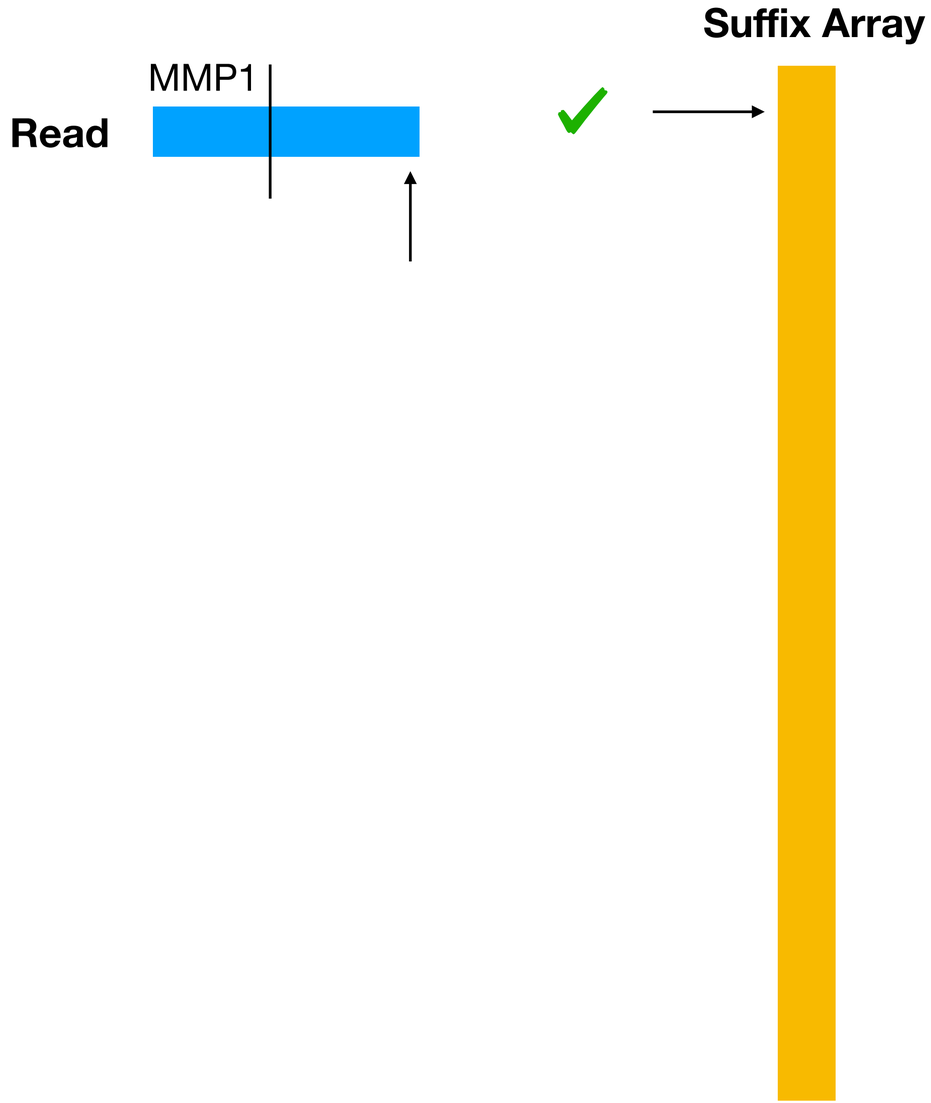
Seed Search



The search is implemented using an uncompressed suffix array(s)

- one for each chromosome
- *MMP* mapping comes "free" with binary search

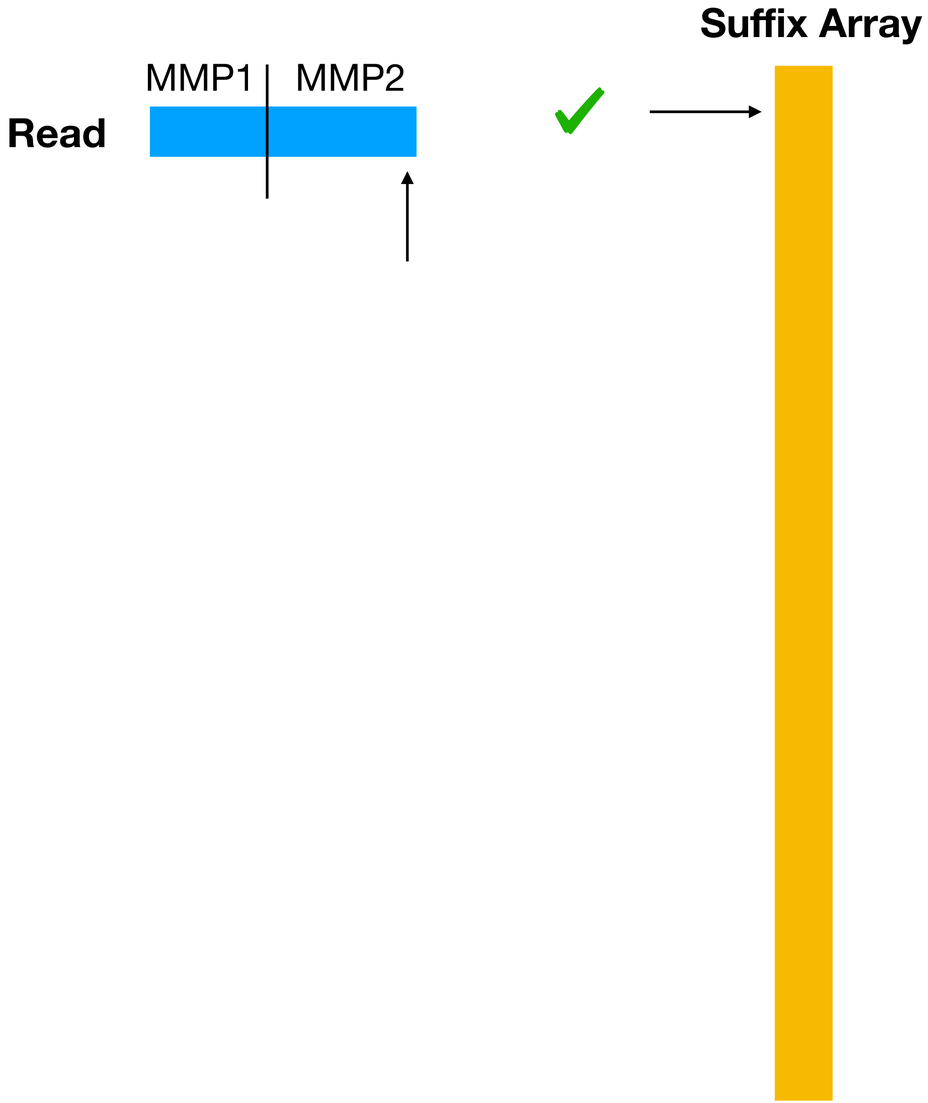
Seed Search



The search is implemented using an uncompressed suffix array(s)

- one for each chromosome
- *MMP* mapping comes "free" with binary search

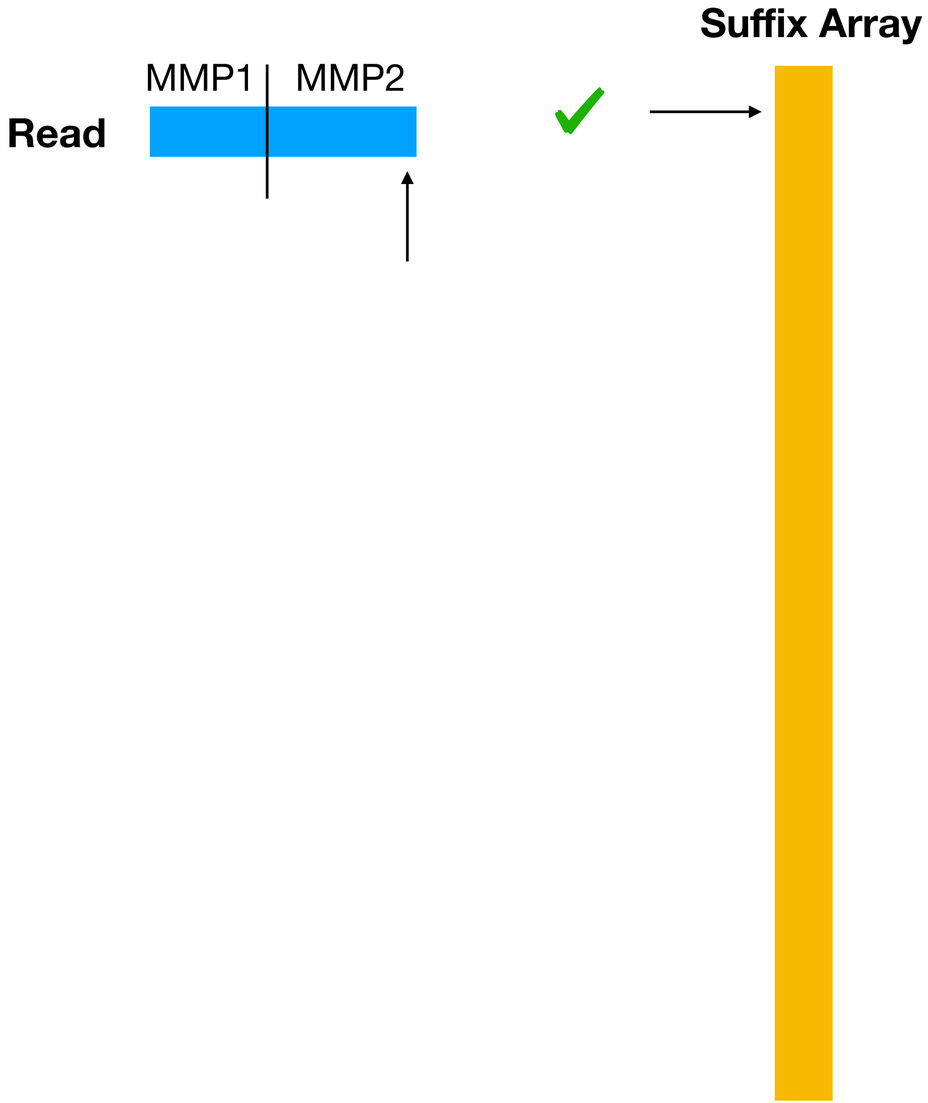
Seed Search



The search is implemented using an uncompressed suffix array(s)

- one for each chromosome
- *MMP* mapping comes "free" with binary search

Seed Search



The search is implemented using an uncompressed suffix array(s)

- one for each chromosome
- *MMP* mapping comes "free" with binary search

needs to be performed using both the read and it's reverse complement

Seed Search

The method described:

Seed Search

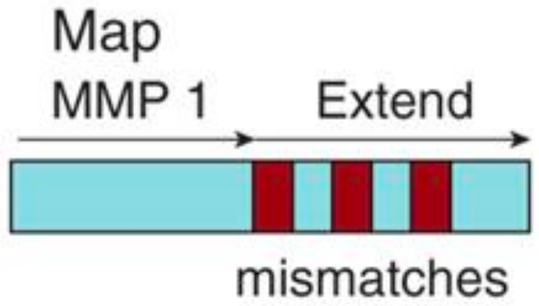
The method described:

- Easily identifies "multi-mapped" reads since they are together in the SA

Seed Search

The method described:

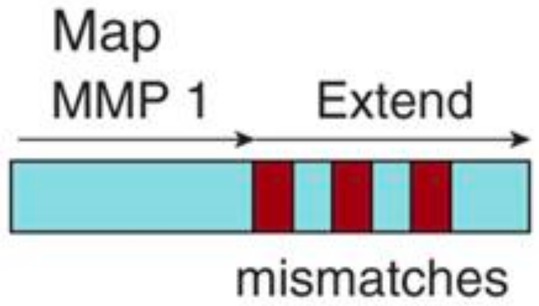
- Easily identifies "multi-mapped" reads since they are together in the SA
- Find mismatches internal to the read



Seed Search

The method described:

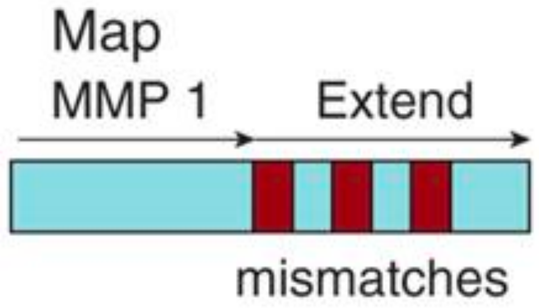
- Easily identifies "multi-mapped" reads since they are together in the SA
- Find mismatches internal to the read
- Overcomes issues with poly-A tails, library adaptors, and low quality

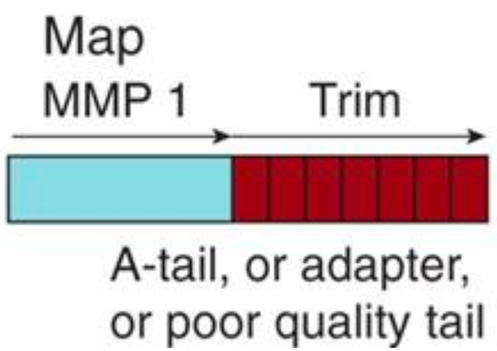


Seed Search

The method described:

- Easily identifies "multi-mapped" reads since they are together in the SA
- Find mismatches internal to the read
- Overcomes issues with poly-A tails, library adaptors, and low quality
- Has high speed, but large memory footprint compared to compressed SAs





Clustering, stitching, and scoring

A set of the MMPs are selected as "anchors"

- "In the current implementation, all the alignments that map less than a user defined value (typically 20-50) are selected as anchors." 🤷‍♂️

Clustering, stitching, and scoring

A set of the MMPs are selected as "anchors"

- "In the current implementation, all the alignments that map less than a user defined value (typically 20-50) are selected as anchors." 🤷‍♂️
- Anchors are those that map to less than 50 locations in the genome

Clustering, stitching, and scoring

A set of the MMPs are selected as "anchors"

- "In the current implementation, all the alignments that map less than a user defined value (typically 20-50) are selected as anchors." 🤷‍♂️
- Anchors are those that map to less than 50 locations in the genome

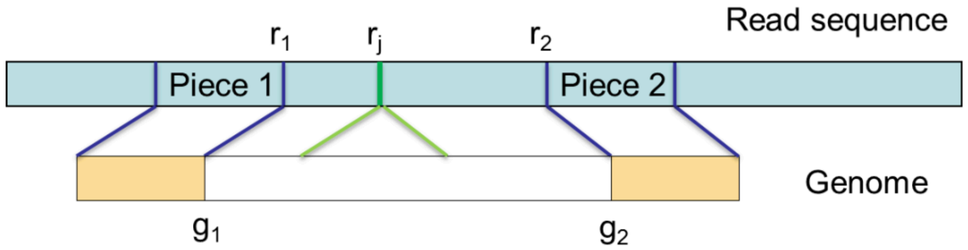
Alignment "windows" are then defined as regions around anchors

- all of the MMPs in those windows will be stitched together linearly
- the size of these windows determines the maximum intron size

Clustering, stitching, and scoring

They stitch two MMPs together allowing

- one gap (of multiple bases) in the genome, and
- minimal mismatches.



Δ is the difference in the size of the space between the MMPs in the genome and the read

- this is how long the gap is going to be

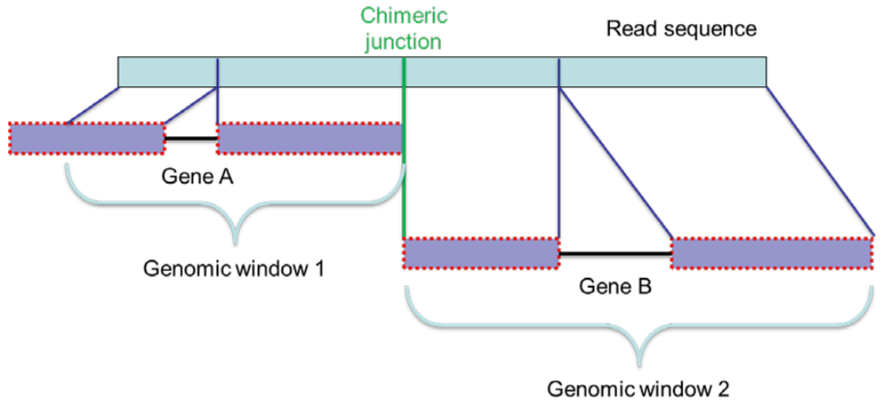
$$\max_{r_1<r_j<r_2}\left\{\sum_{r'=1}^{(r_j-r_1)} Match(r,\Delta) - P_{gap}(r_j)\right\}$$

Match counts the number of bases that are better aligned before the gap minus the count of those that are better placed after

$$\Delta =: (g_2 - g_1) - (r_2 - r_1)$$

$$Match(r',\Delta) =: \begin{cases} 1 & \textbf{if } R(r_1+r') = G(g_1+r') \textbf{ \& } R(r_1+r'+\Delta) \neq G(g_1+r'+\Delta) \\ -1 & \textbf{if } R(r_1+r') = G(g_1+r') \textbf{ \& } R(r_1+r'+\Delta) \neq G(g_1+r'+\Delta) \\ 0 & otherwise \end{cases}$$

Clustering, stitching, and scoring



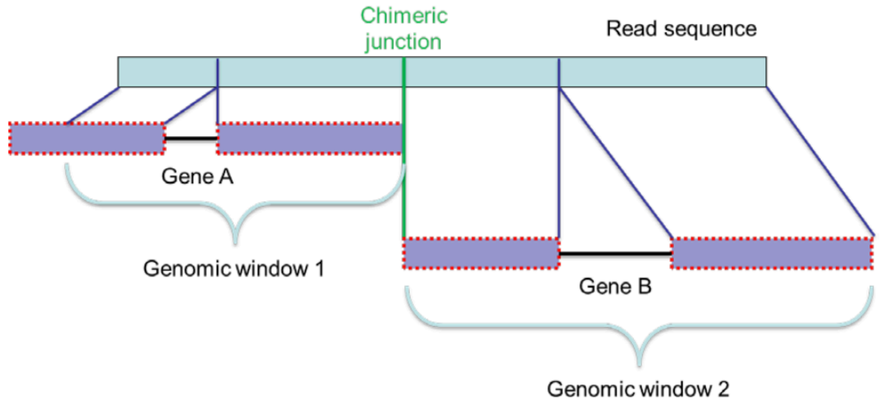
Mate pairs are initially mapped independently as long as they are on the same strand

If the whole read is not covered in 1 window

- find two (or more) non-overlapping windows that map to the read
- create a chimeric mapping

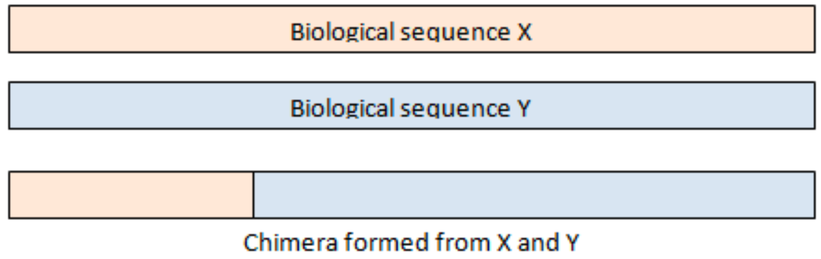
Clustering, stitching, and scoring

Mate pairs are initially mapped independently as long as they are on the same strand



If the whole read is not covered in 1 window

- find two (or more) non-overlapping windows that map to the read
- create a chimeric mapping



A "chimeric sequence" or "chimera" is a sequence made by combining two (or more other sequences). For example, this can happen with mistakes in crossover.

Clustering, stitching, and scoring

The score of each mapped read is:

$$S(\mathbb{A}) = mt_{\mathbb{A}} - ms_{\mathbb{A}} - in_{\mathbb{A}} - dl_{\mathbb{A}} - gp_{\mathbb{A}}$$

Clustering, stitching, and scoring

The score of each mapped read is:

$$S(\mathbb{A}) = mt_{\mathbb{A}} - ms_{\mathbb{A}} - in_{\mathbb{A}} - dl_{\mathbb{A}} - gp_{\mathbb{A}}$$

match and mismatch scores
as we saw before

Clustering, stitching, and scoring

The score of each mapped read is:

$$S(\mathbb{A}) = mt_{\mathbb{A}} - ms_{\mathbb{A}} - in_{\mathbb{A}} - dl_{\mathbb{A}} - gp_{\mathbb{A}}$$

match and mismatch scores as we saw before

affine gaps with different scores for the read and the genome

Clustering, stitching, and scoring

The score of each mapped read is:

$$S(\mathbb{A}) = mt_{\mathbb{A}} - ms_{\mathbb{A}} - in_{\mathbb{A}} - dl_{\mathbb{A}} - gp_{\mathbb{A}}$$

match and mismatch scores as we saw before

affine gaps with different scores for the read and the genome

splice junction score based on the sequences at the end of the junctions: GT/AG, GC/AG, & AT/AC are normal, all others have higher penalties (this is the P_{gap} score from earlier)

Clustering, stitching, and scoring

The score of each mapped read is:

$$S(\mathbb{A}) = mt_{\mathbb{A}} - ms_{\mathbb{A}} - in_{\mathbb{A}} - dl_{\mathbb{A}} - gp_{\mathbb{A}}$$

match and mismatch scores as we saw before

affine gaps with different scores for the read and the genome

splice junction score based on the sequences at the end of the junctions: GT/AG, GC/AG, & AT/AC are normal, all others have higher penalties (this is the P_{gap} score from earlier)

For each read, the mapping with the highest score is retained

Speed vs. Space tradeoff of the SA

Table 1. Mapping speed and RAM benchmarks on the experimental RNA-seq dataset

Aligner	Mapping speed: million read pairs/hour		Peak physical RAM, GB	
	6 threads	12 threads	6 threads	12 threads
STAR	309.2	549.9	27.0	28.4
STAR sparse	227.6	423.1	15.6	16.0
TopHat2	8.0	10.1	4.1	11.3
RUM	5.1	7.6	26.9	53.8
MapSplice	3.0	3.1	3.3	3.3
GSNAP	1.8	2.8	25.9	27.0

Alignment Quality

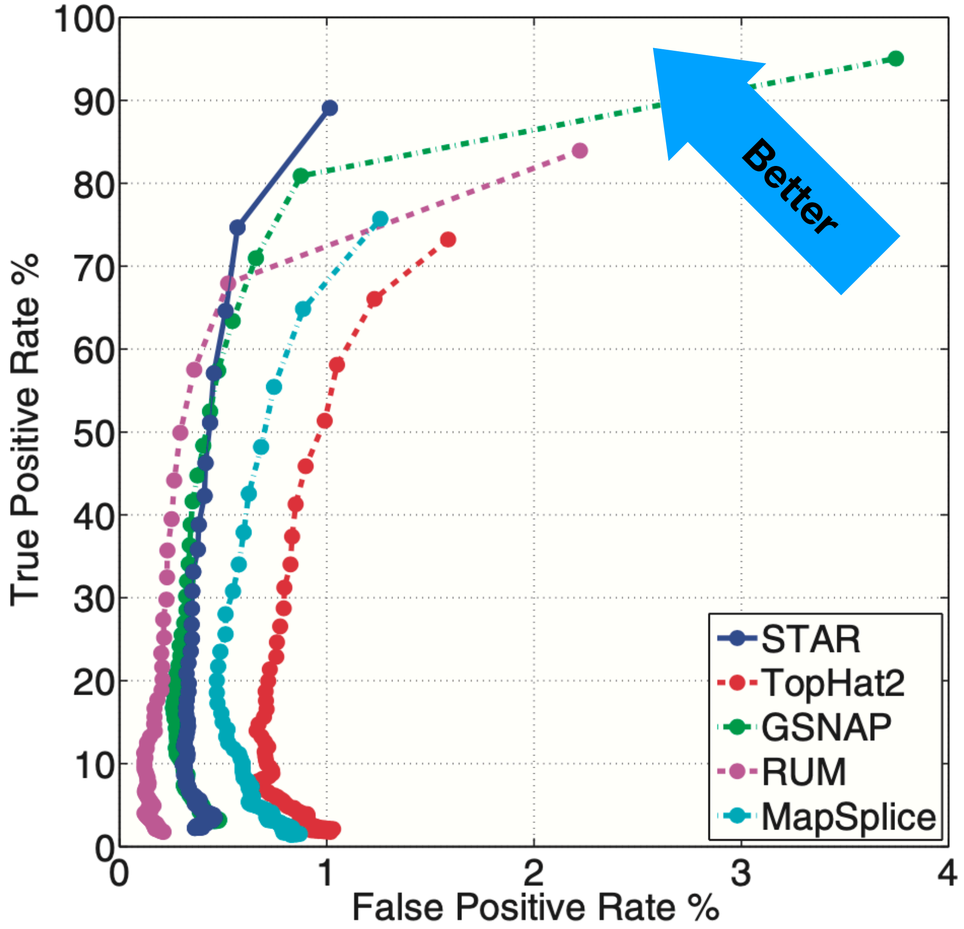


Fig. 2. True-positive rate versus false-positive rate (ROC-curve) for simulated RNA-seq data for STAR, TopHat2, GSNAP, RUM and MapSplice

Take Aways for STAR

Non-contiguous alignment for RNA-Seq is not a totally solved problem

STAR is specifically designed to take introns into account during alignment

Algorithm is extendable to longer read lengths since it can ignore poor quality regions and chimeric reads

Large memory consumption, but fast due to the use of uncompressed SAs

TopHat

Creates an alignment in stages:

- find all high quality whole read alignments to the genome
- identifying possible splice locations
- aligning initially unmapped reads to induced sequences

BIOINFORMATICS ORIGINAL PAPER

Vol. 25 no. 9 2009, pages 1105–1111
doi:10.1093/bioinformatics/btp120

Sequence analysis

TopHat: discovering splice junctions with RNA-Seq

Cole Trapnell[1,*], Lior Pachter[2] and Steven L. Salzberg[1]

[1]Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD 20742 and
[2]Department of Mathematics, University of California, Berkeley, CA 94720, USA

Received on October 23, 2008; revised on February 24, 2009; accepted on February 26, 2009

Advance Access publication March 16, 2009

Associate Editor: Ivo Hofacker

Aligning Reads using Bowtie

Using strict alignment critera, TopHat uses Bowtie to align reads to the whole genome

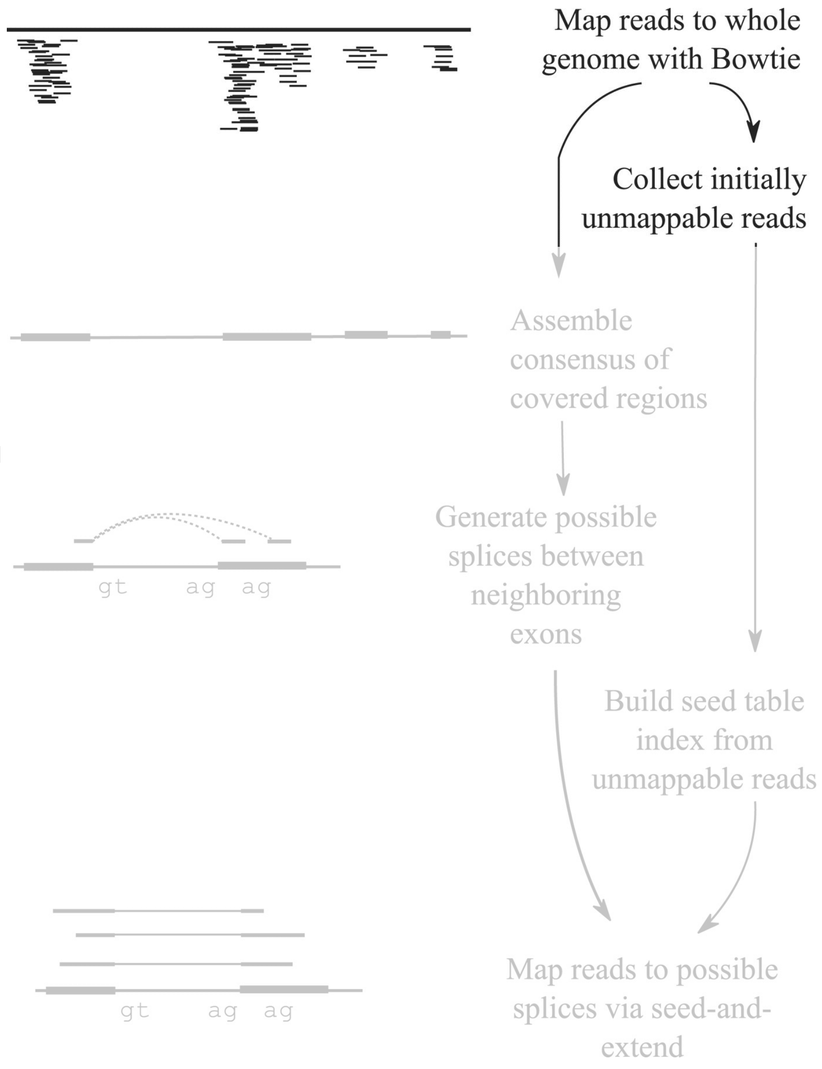
- they include multi-mapped reads

The reads which don't map are called "Initially Unmapped" or IUM reads

- likely don't map because they cross introns
- will be used later to confirm splice junctions

"Low complexity" reads are discarded

- these are likely highly repetitive



Construct potential exons

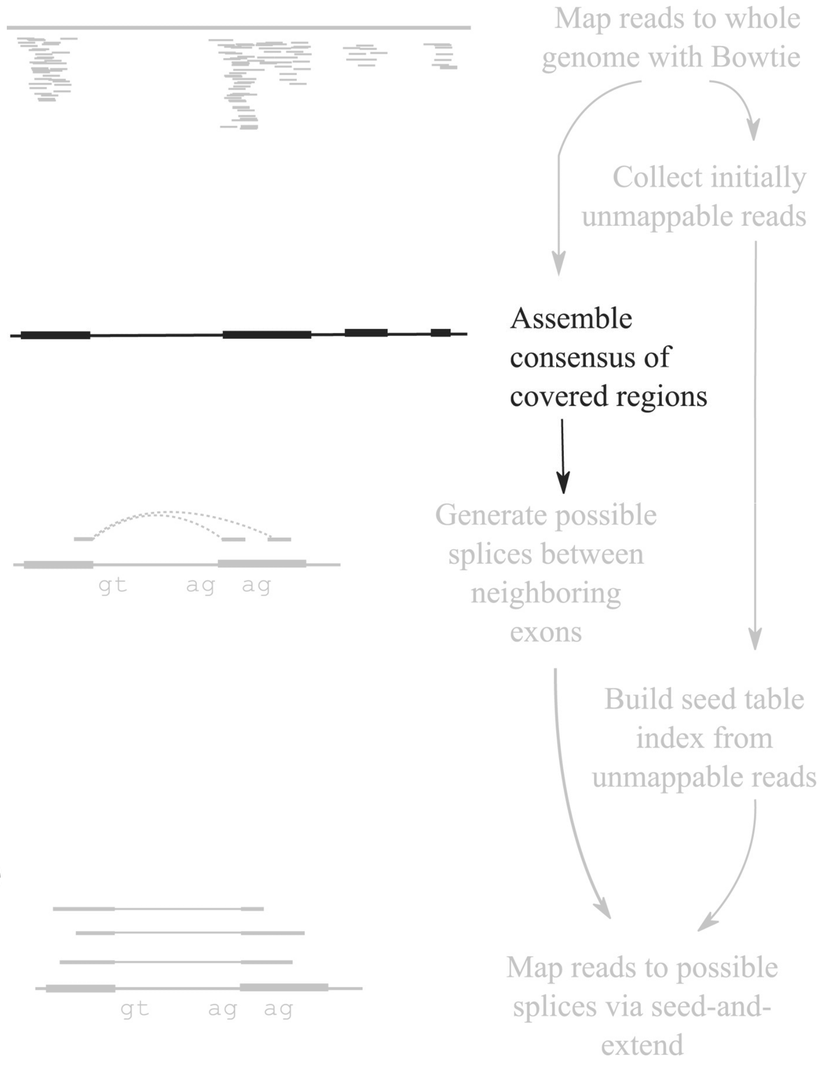
Construct the set of mapped sequences

- the "islands" of sequence that map to the genome
- using the assemble functionality of MAQ

In low coverage regions replace any mismatches with the original genomic sequence

Append a small amount of the flanking sequence from the genome to the end of each cluster

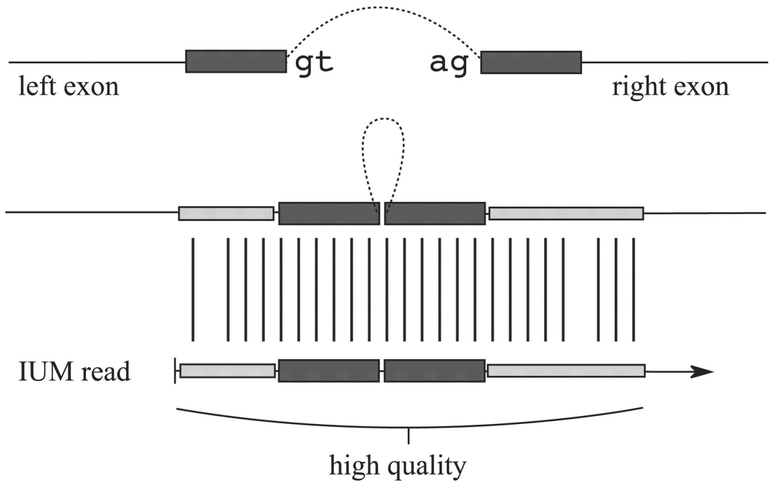
- there may be some parts of the end of an exon that are only covered by spliced reads

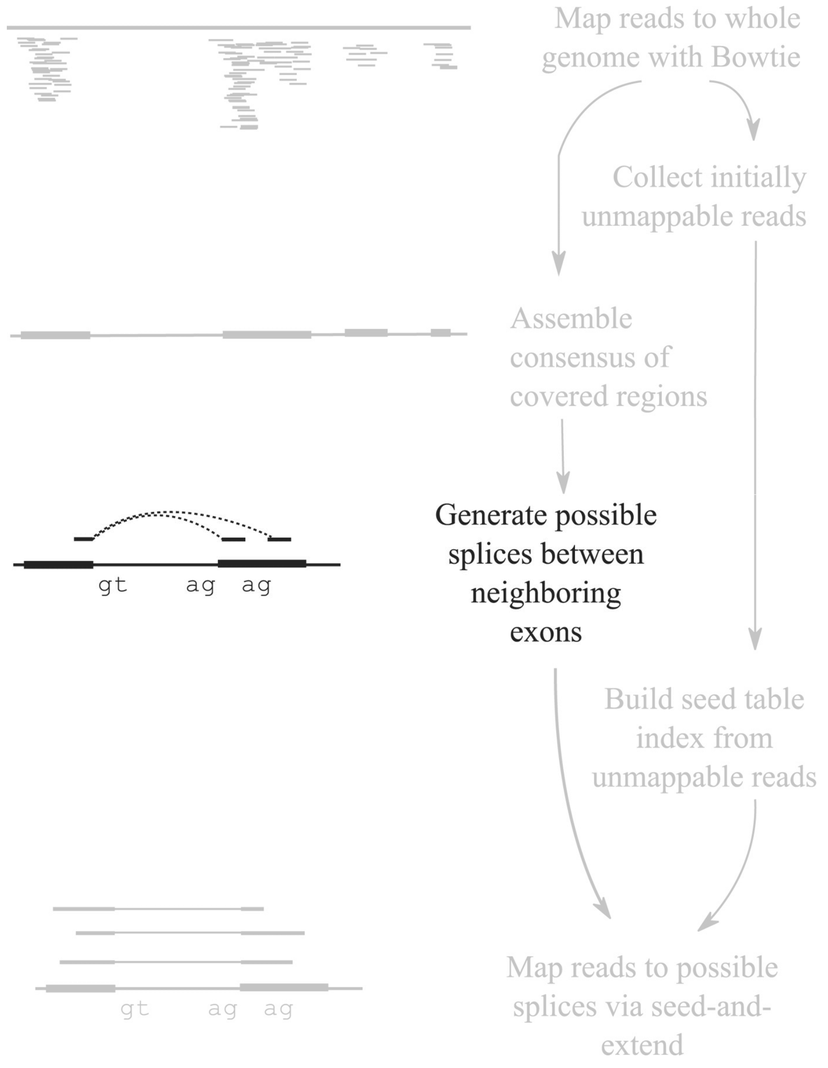


Create potential splice locations

Splice junctions usually happen with predictable bases

- consider all possible pairs as potential splice locations
- create a set of new sequences
- store the *k*-mer surrounding such locations as a seed for mapping





Create potential splice locations

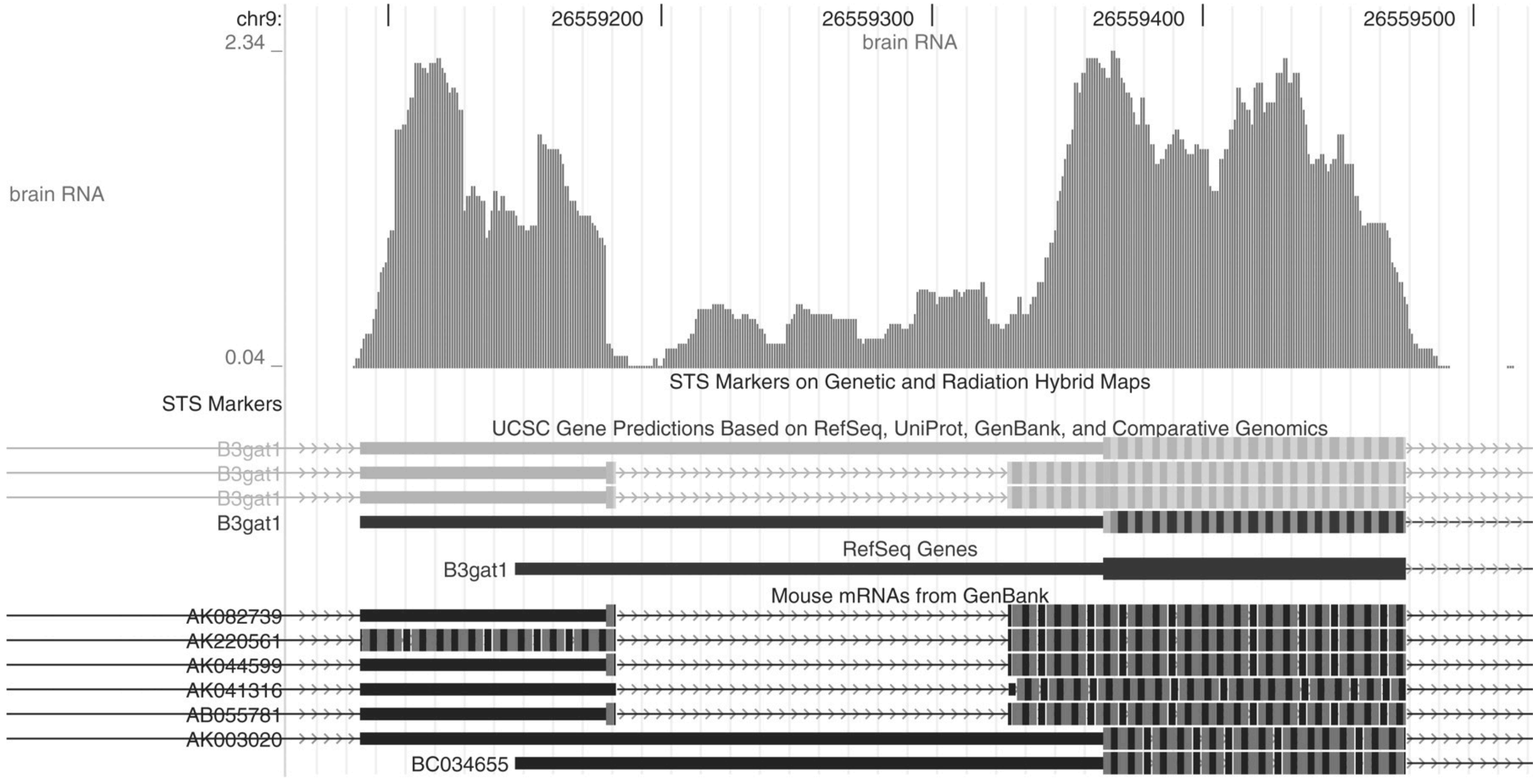
Creating all possible combinations may be too many

- only find regions that are high coverage or at the ends of the potential exons

probability of including potential junction *i* to *j*

$$D_{ij} = \frac{\sum_{m=1}^{j} d_m}{j - i} \cdot \frac{1}{\sum_{m=0}^{n} d_m}$$

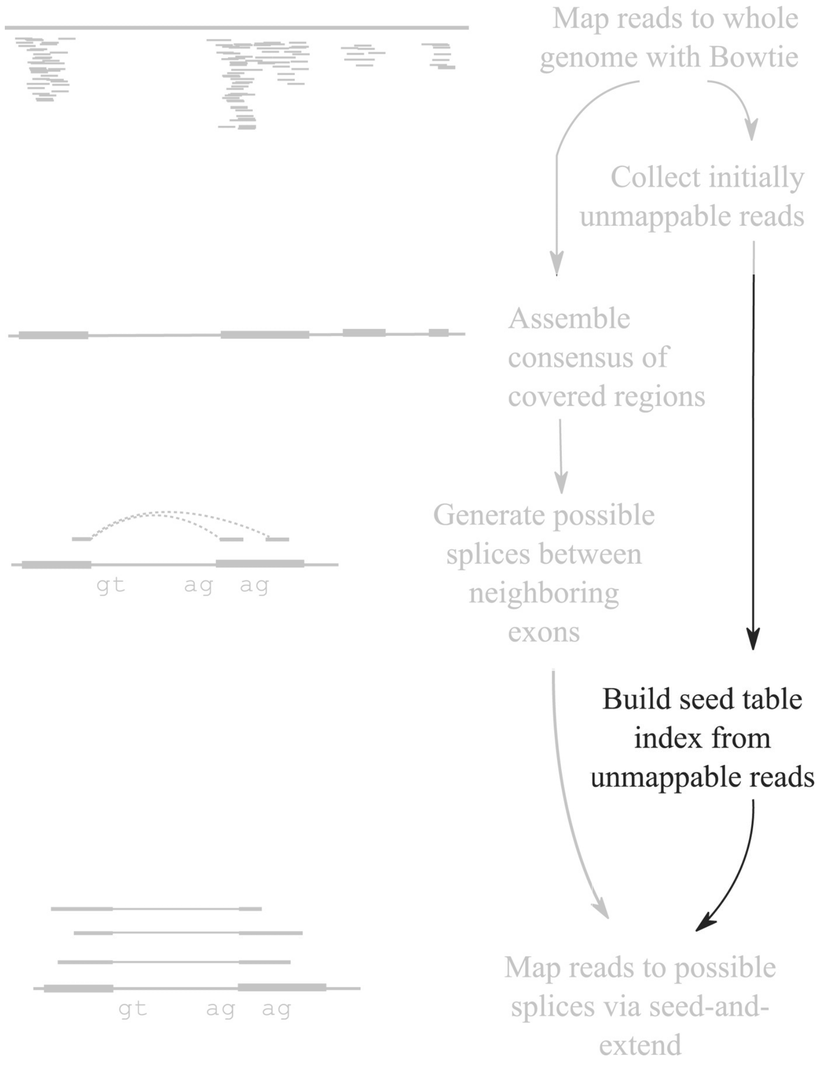
coverage at location *m*



Index IUM reads

For each unmapped read

- extract all unique *k*-mers from the "high quality" region
- here *k~10*

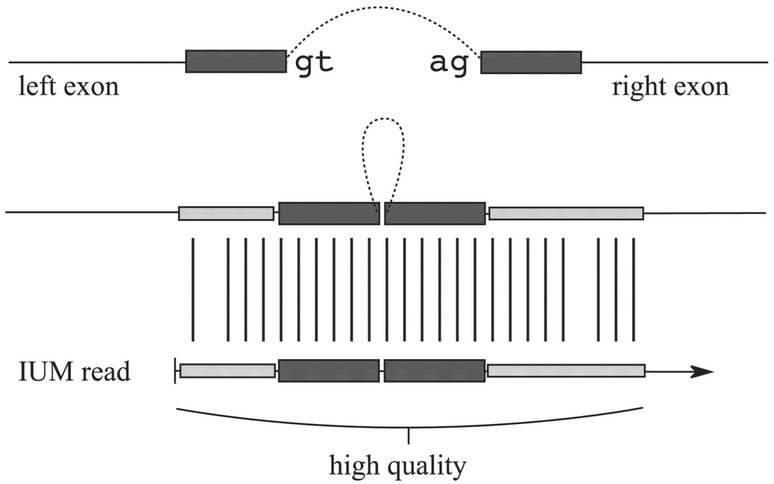


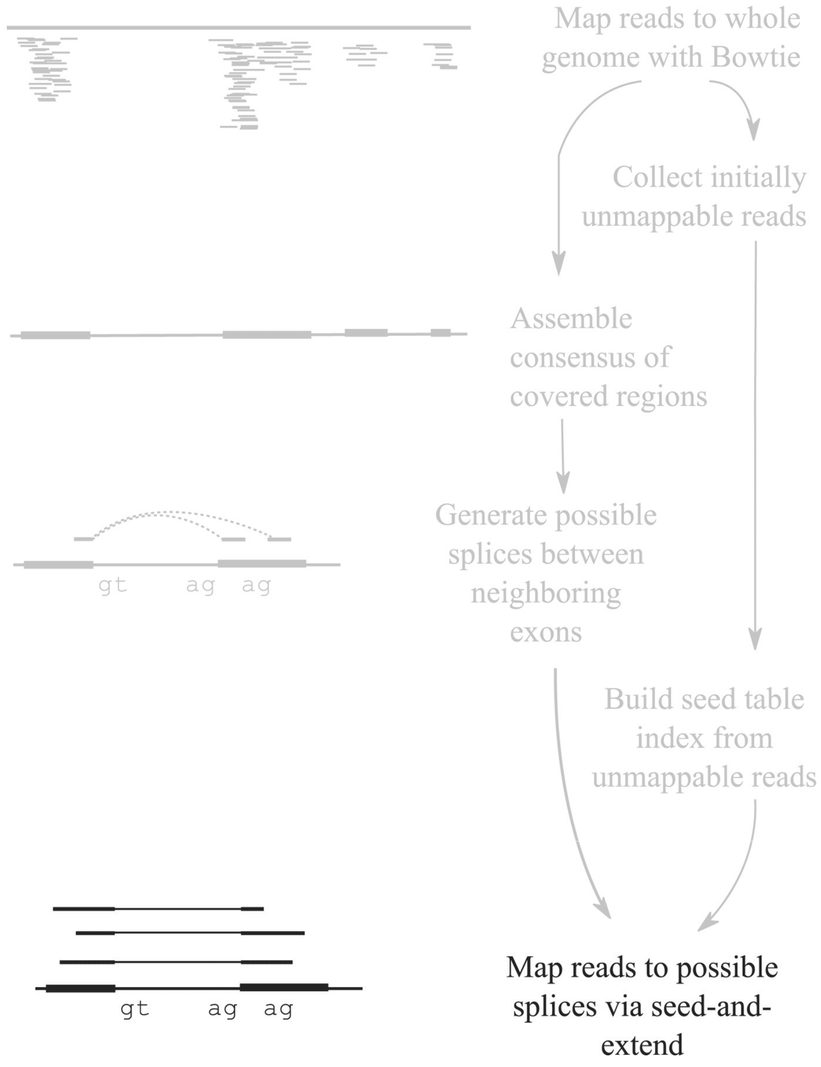
Align remaining reads

Find all matches between *k*-mers

- from the IUM set and
- the newly created potential junction locations

Extend left and right from the *k*-mer to find an alignment





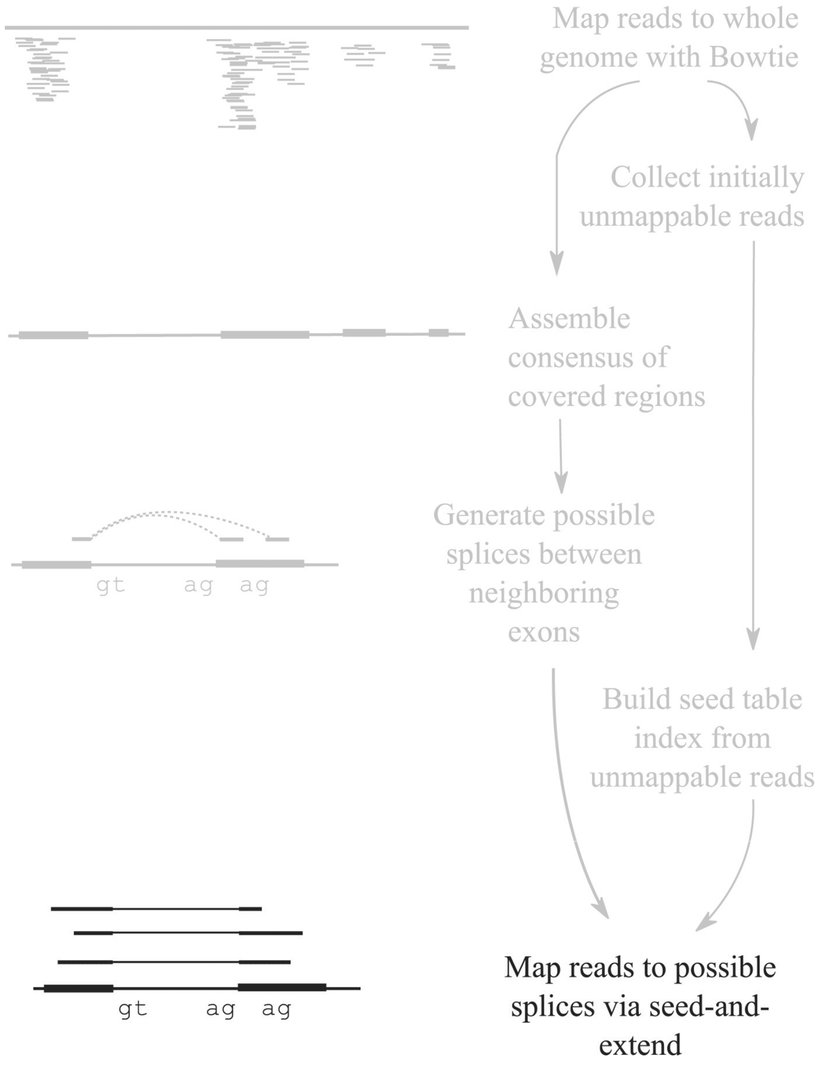
Align remaining reads

Find all matches between *k*-mers
- from the IUM set and
- the newly created potential junction locations

Extend left and right from the *k*-mer to find an alignment

Will miss alignments with mismatches near splice location

Discard junctions that are predicted to be anomalies:
- occurrence rate is lower than some percentage of the coverage of the flanking regions



Take Aways from TopHat

Uses existing software to do some of the heavy lifting

Strict parameters on the splice junctions make the algorithm fast

Limited in the splice junction sequence

TopHat2

Aligns reads in stages of increasing time requirements:

- first to the transctiptome (set of all known transcripts),
- then whole reads to the genome,
- then in chunks

Kim *et al. Genome Biology* 2013, **14**:R36
http://genomebiology.com/2013/14/4/R36

Genome Biology

METHOD Open Access

TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions

Daehwan Kim[1,2,3*], Geo Pertea[3], Cole Trapnell[5,6], Harold Pimentel[7], Ryan Kelley[8] and Steven L Salzberg[3,4]

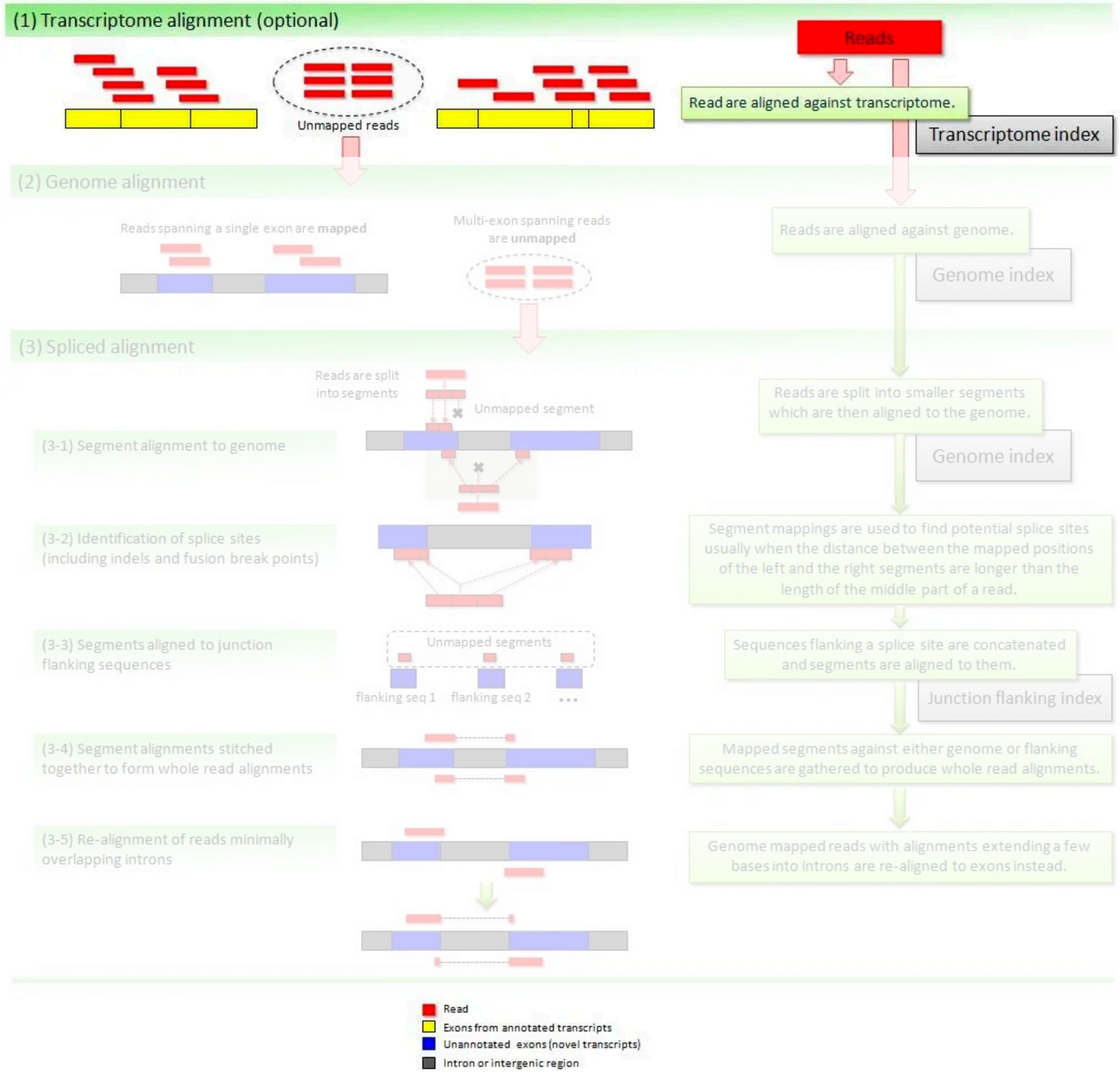
Align to transcriptome

Aligned using Bowtie

The transcriptome is the set of all known transcripts

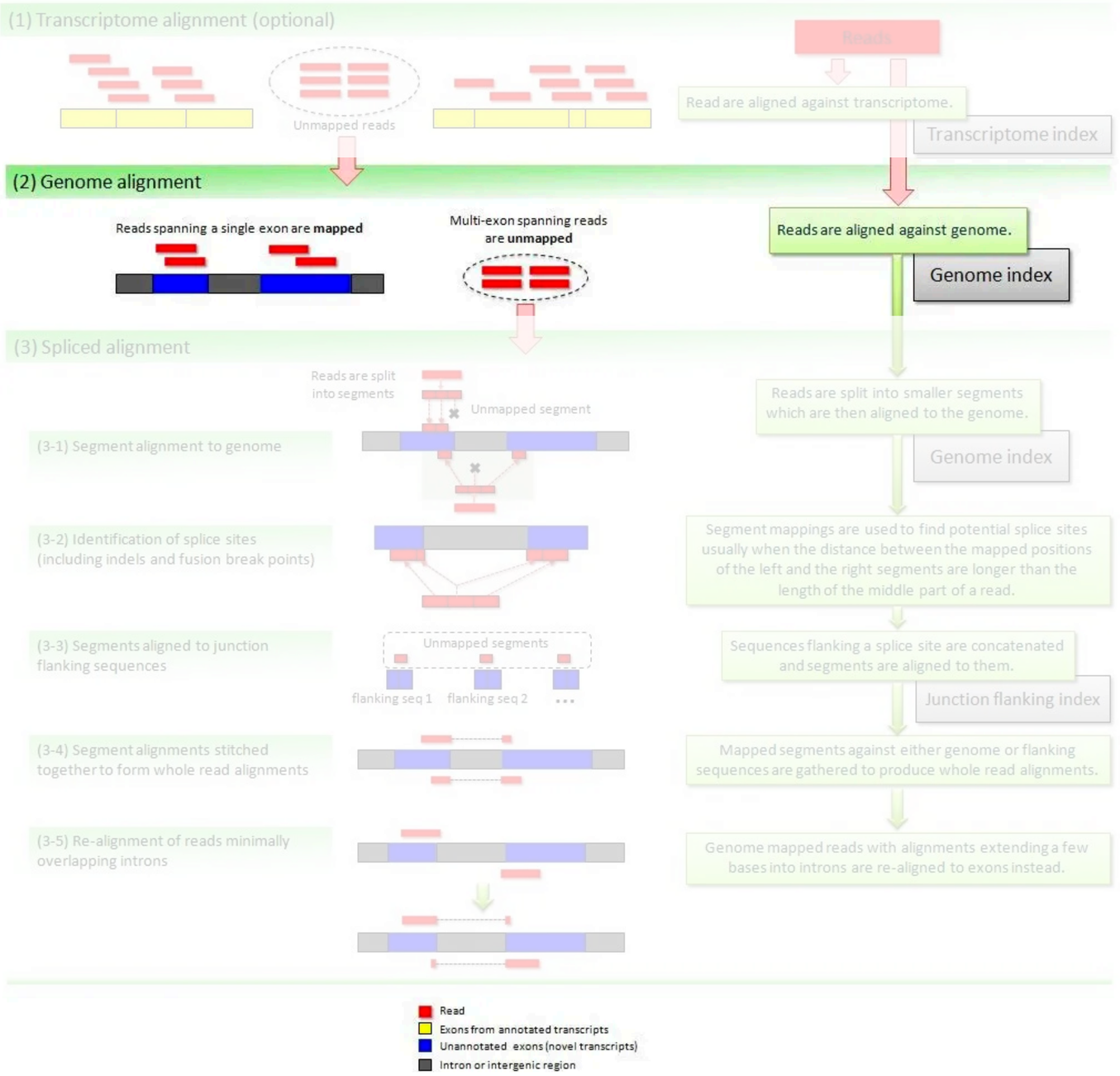
- missing some novel transcripts

This step identifies some of the spliced reads before we go into the TopHat(1) pipeline



Align unmapped reads to genome

Same as the first step of TopHat(1)

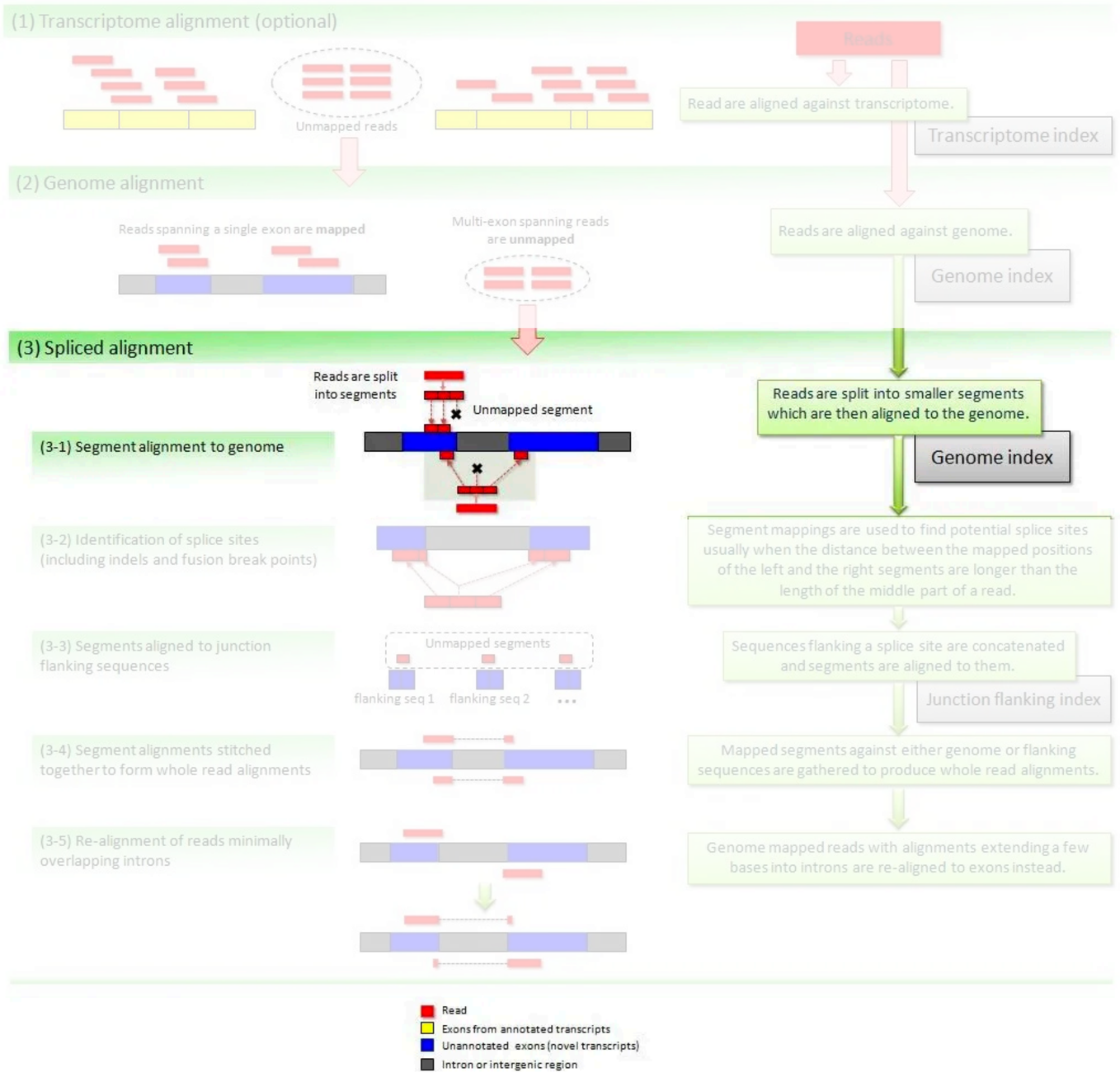


Align chunks of unmapped reads

Additional step from TopHat(1)

Split read into subsequences

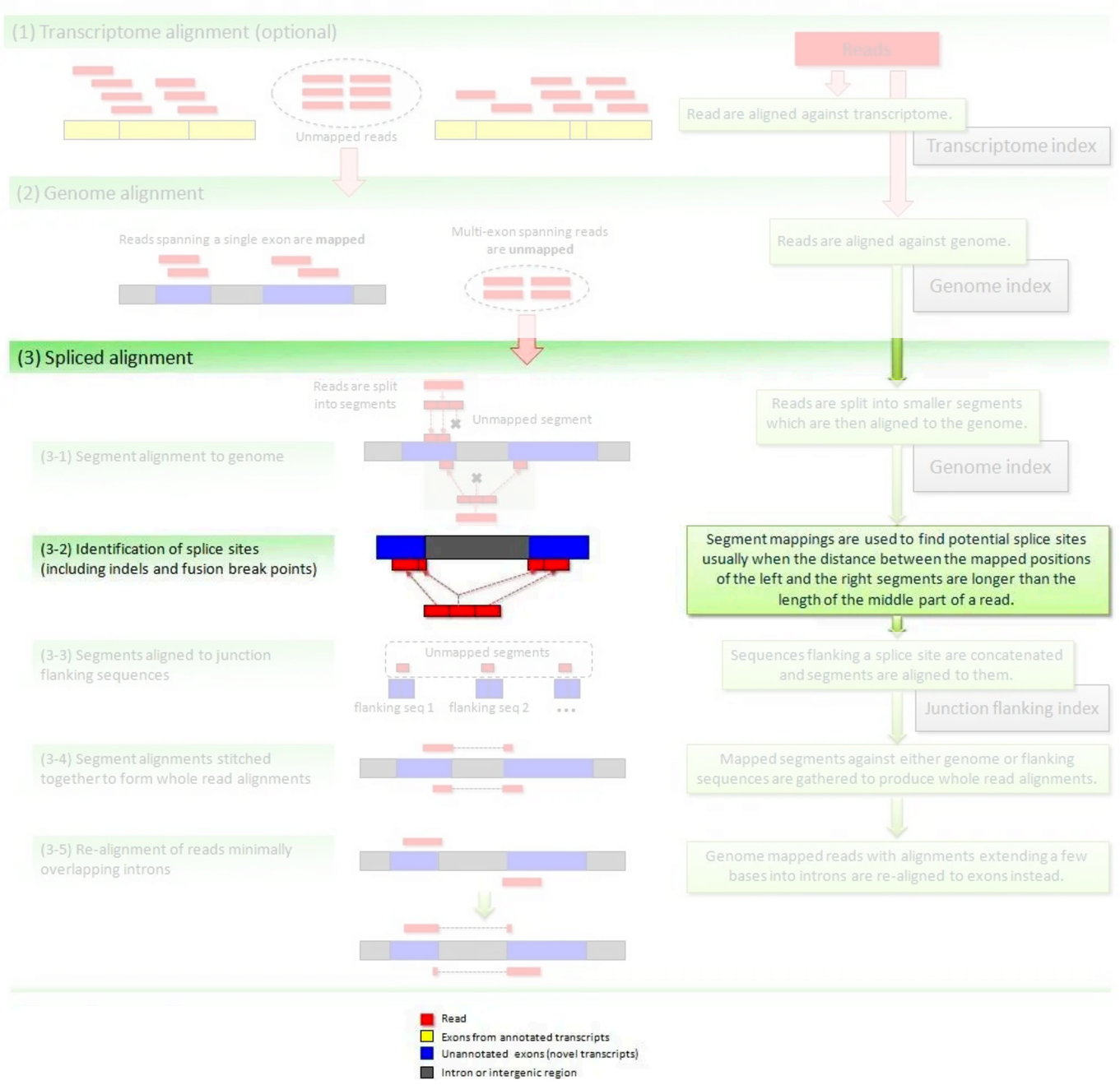
- align these subsequences using Bowtie



Identify potential splice locations

Use these mapped reads to limit the number of potential splice locations

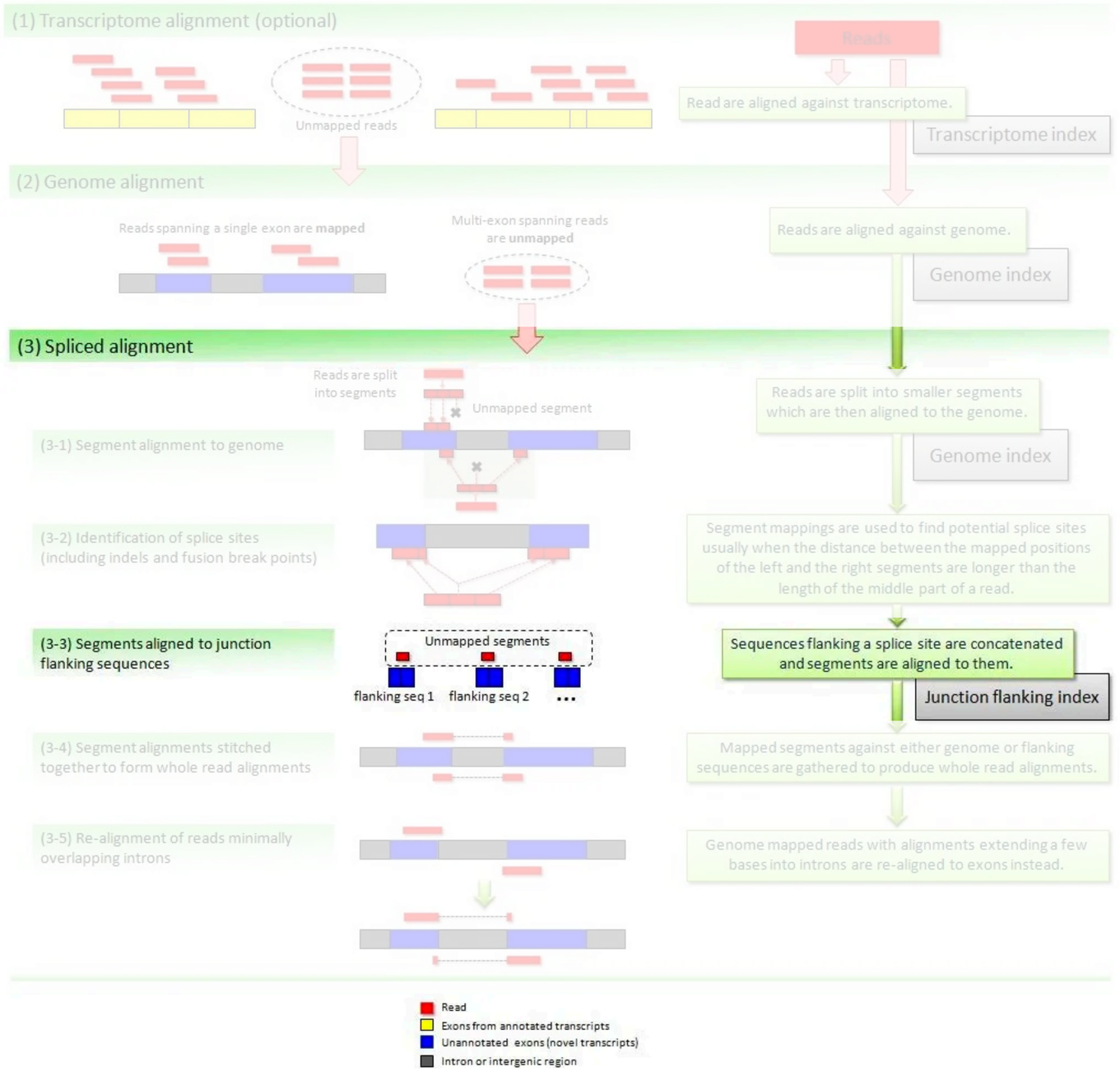
- consider more pairs of ending 2-mers



Align unaligned chunks

Creating a new index of these flanking regions

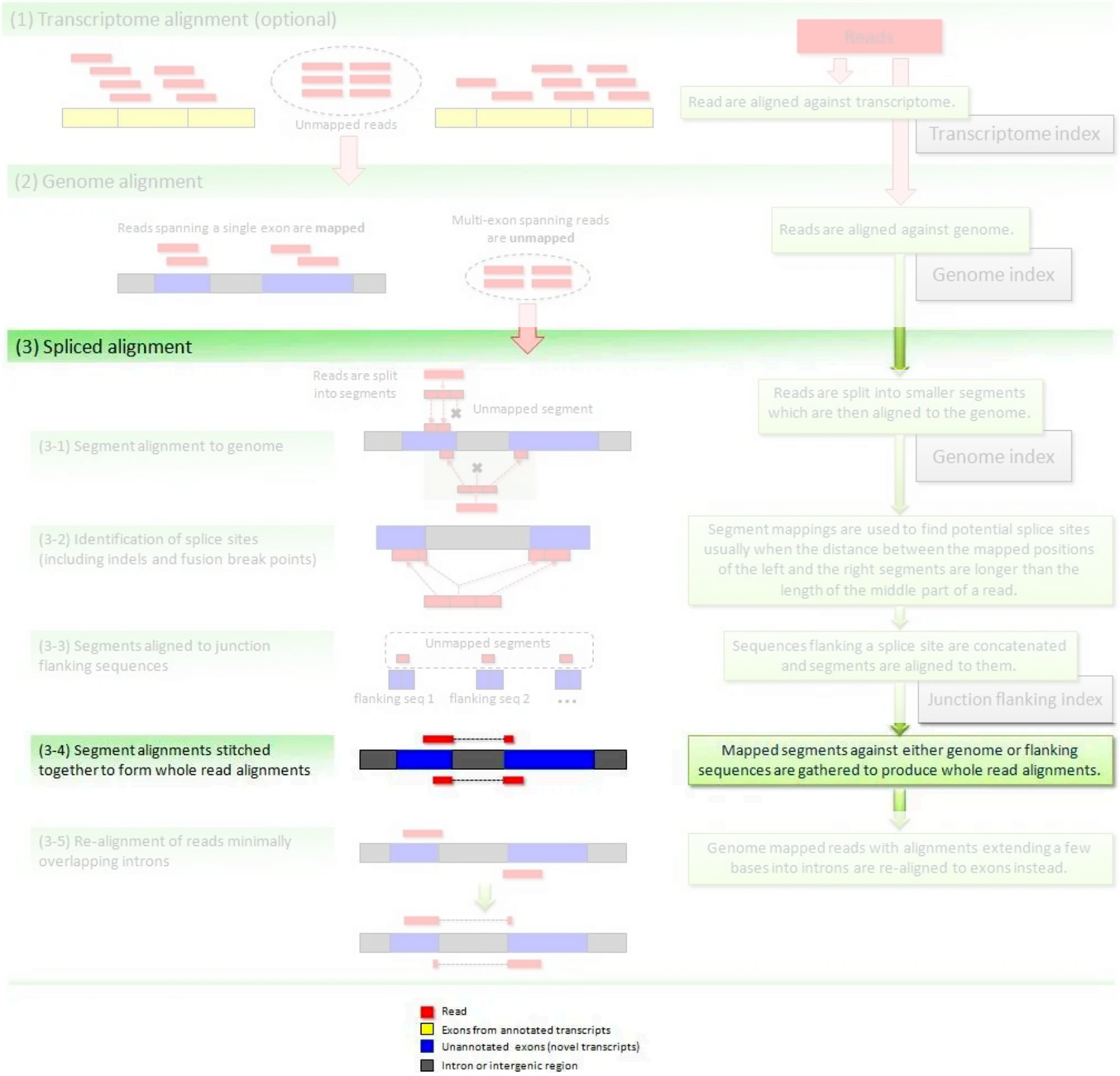
- align the unaligned chunks of all of the reads to find the location of the splices



Recreate whole read alignments

Using the flanking mapping or the original mapping

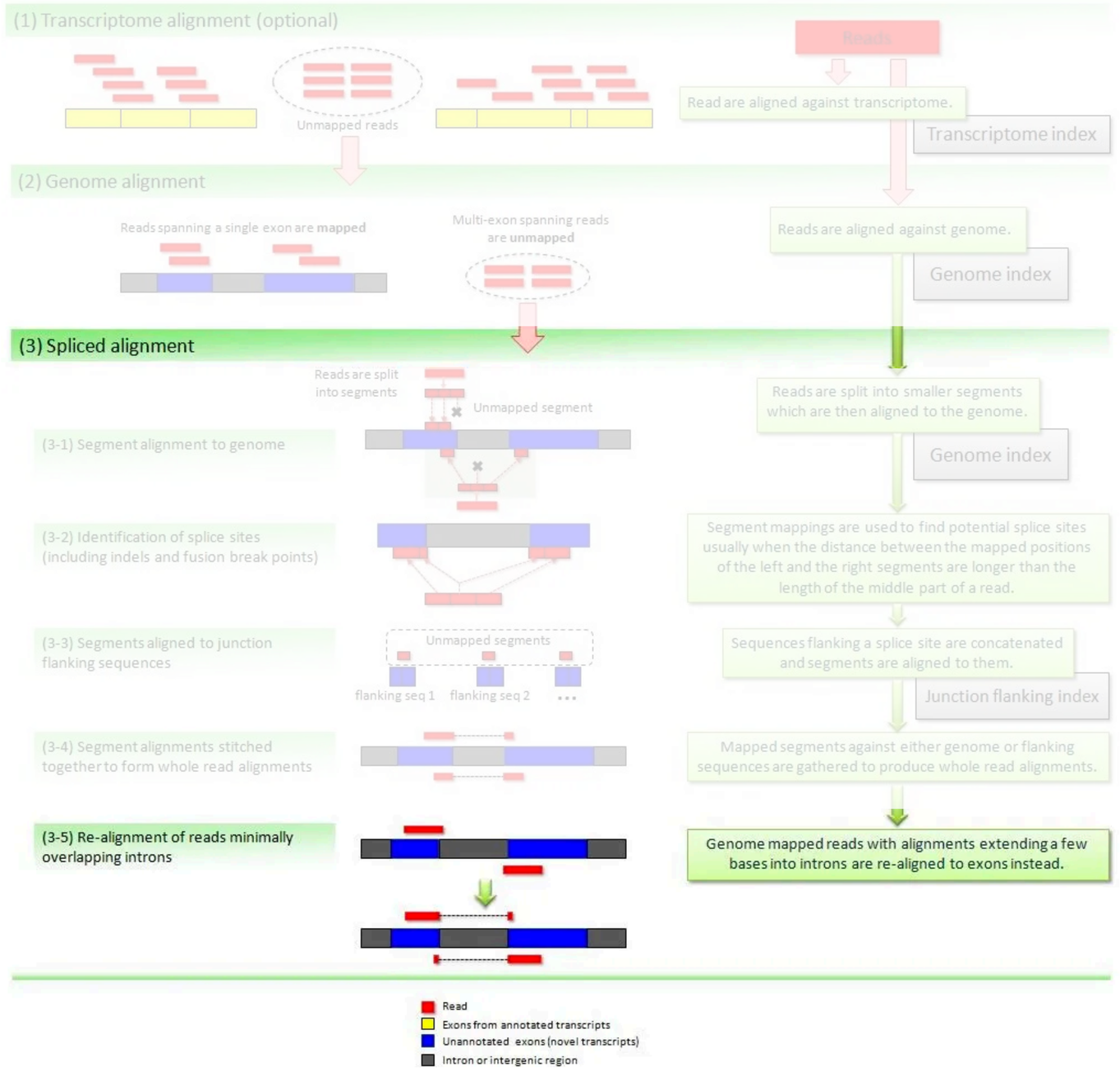
Stitch together to get the whole read mapping



Re-map reads

This step will help align small overlaps between the two adjoining exons

- similar to the last step in TopHat(1)



Take Aways for TopHat2

Built on Bowtie(2), so the actual mapping is very accurate

Annotation allows TopHat2 to better align reads in known transcripts