Exploring Isoform-specific Expression

With R and Bioconductor

Michael Lawrence

Genentech

February 28, 2012

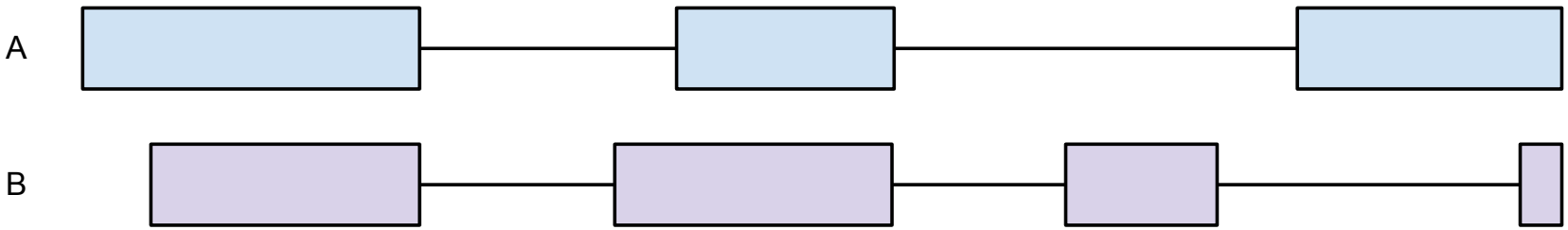
Outline

Outline

Questions of Interest

RNA-seq alignments permit us to ask questions about transcript:

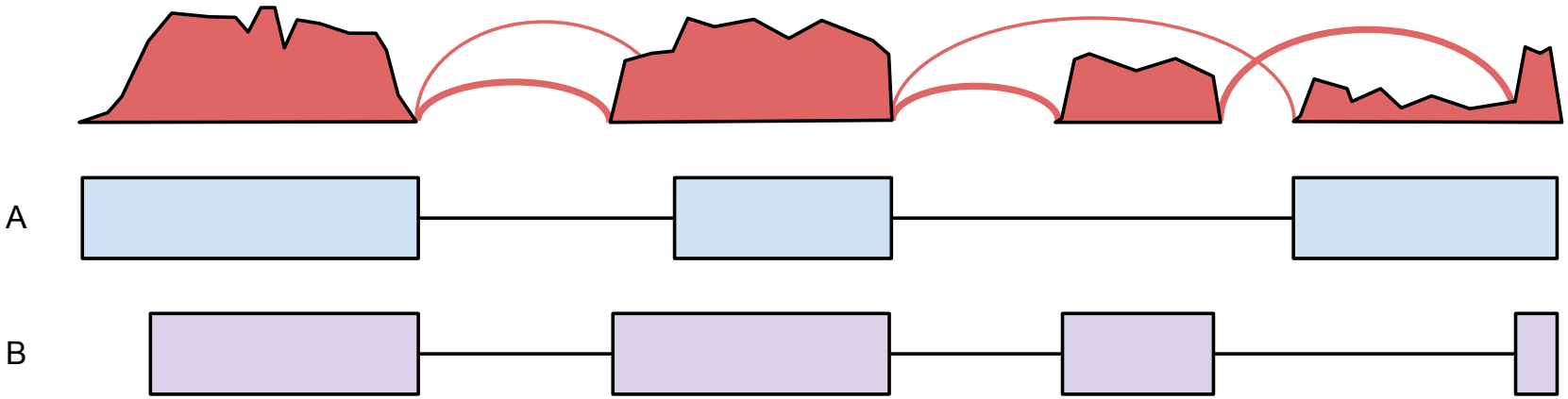
- Expression levels (from counts over exons and junctions)
- Structure (from the alignments)
- Variants (from the sequences)



Questions of Interest

RNA-seq alignments permit us to ask questions about transcript:

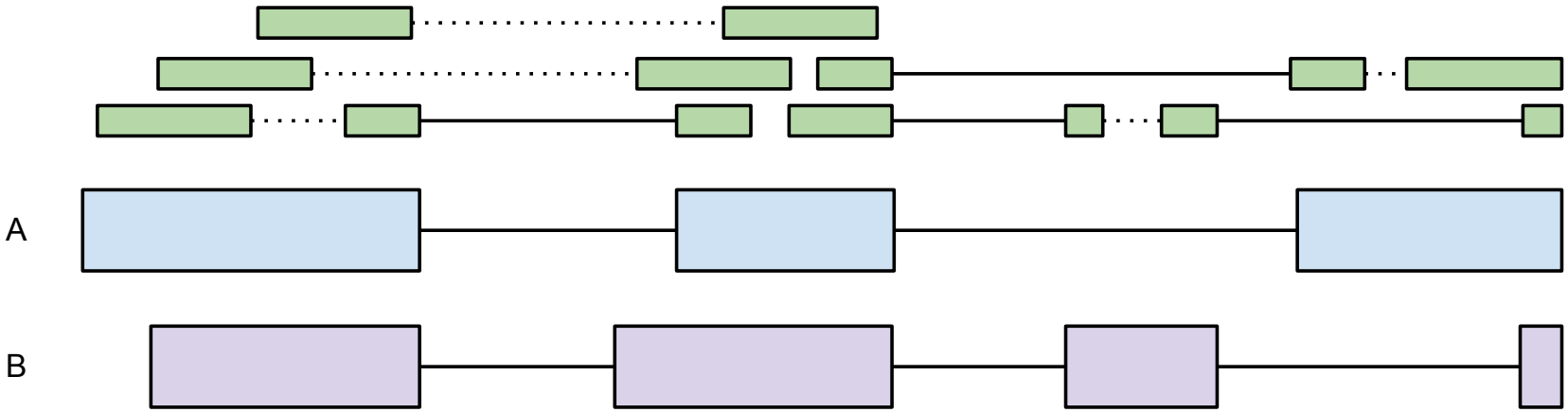
- Expression levels (from counts over exons and junctions)
- Structure (from the alignments)
- Variants (from the sequences)



Questions of Interest

RNA-seq alignments permit us to ask questions about transcript:

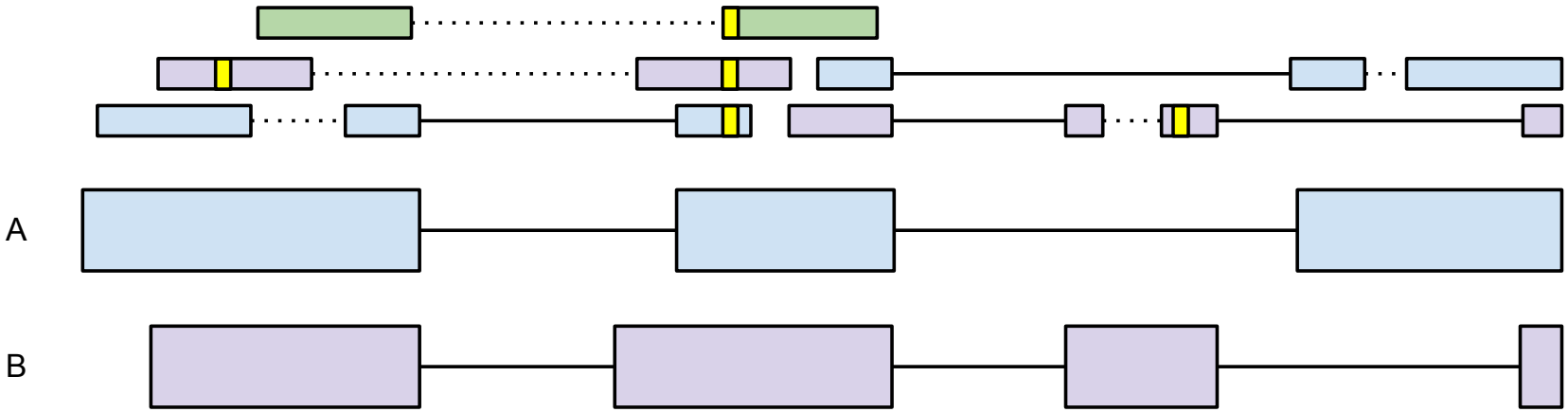
- Expression levels (from counts over exons and junctions)
- Structure (from the alignments)
- Variants (from the sequences)



Questions of Interest

RNA-seq alignments permit us to ask questions about transcript:

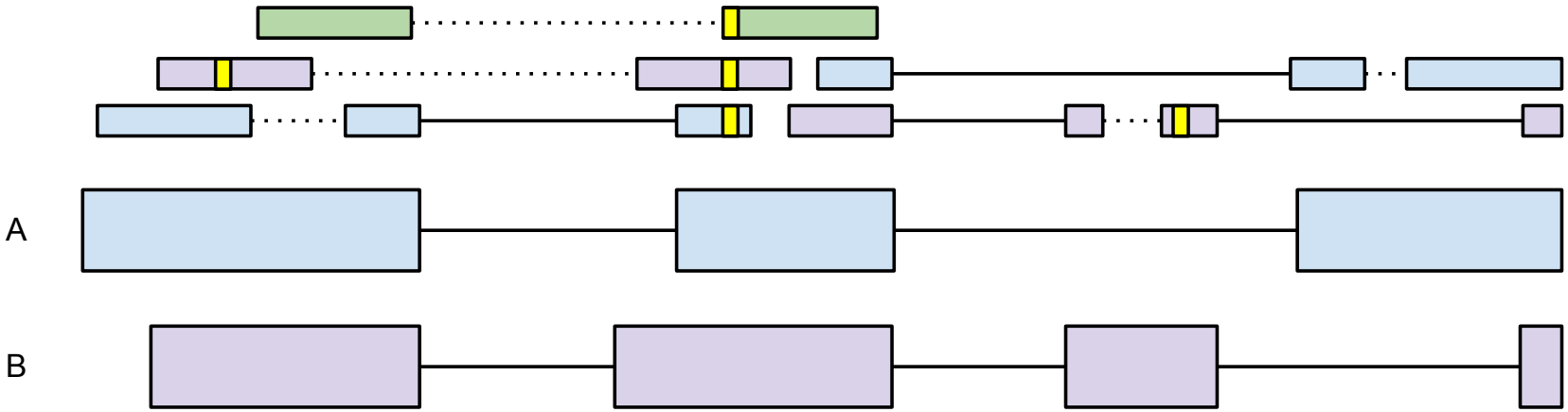
- Expression levels (from counts over exons and junctions)
- Structure (from the alignments)
- Variants (from the sequences)



Questions of Interest

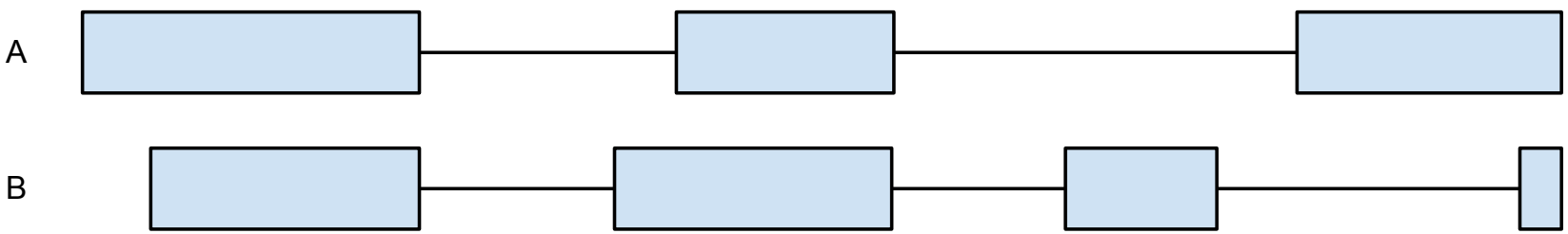
RNA-seq alignments permit us to ask questions about transcript:

- Expression levels (from counts over exons and junctions)
- Structure (from the alignments)
- Variants (from the sequences)



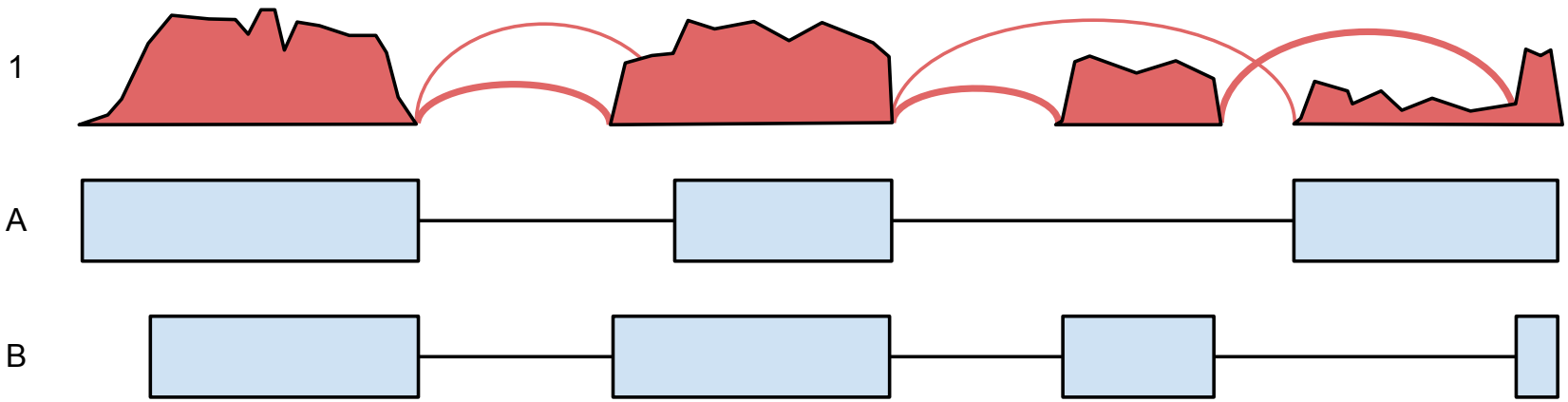
Questions about Isoform Expression

- Is an isoform expressed?
- What is the expression level of a particular isoform, and how can we compare it to that of other isoforms?
- Is the balance in isoform expression different across samples?



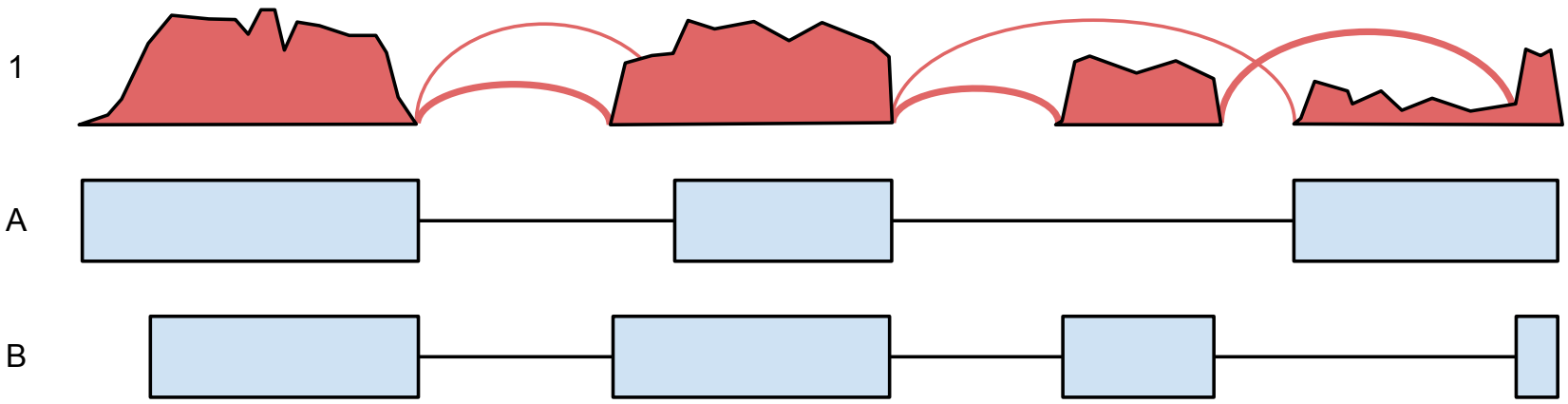
Questions about Isoform Expression

- Is an isoform expressed?
- What is the expression level of a particular isoform, and how can we compare it to that of other isoforms?
- Is the balance in isoform expression different across samples?



Questions about Isoform Expression

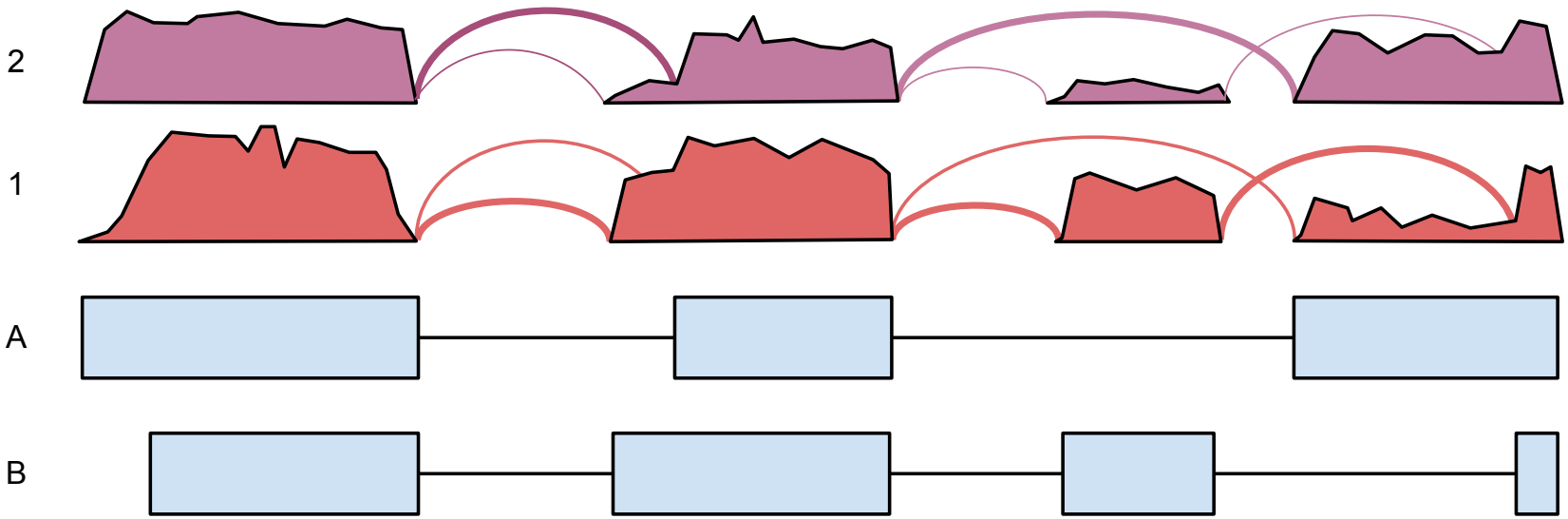
- Is an isoform expressed?
- What is the expression level of a particular isoform, and how can we compare it to that of other isoforms?
- Is the balance in isoform expression different across samples?



	Isoform A	Isoform B
Sample 1	5	12

Questions about Isoform Expression

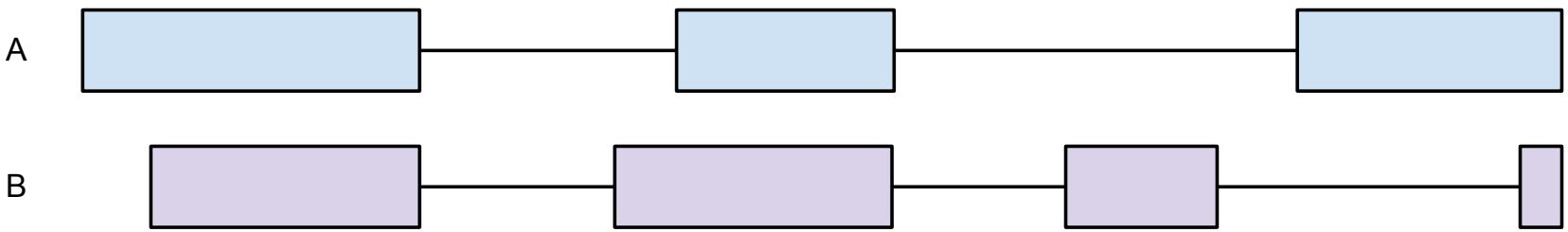
- Is an isoform expressed?
- What is the expression level of a particular isoform, and how can we compare it to that of other isoforms?
- Is the balance in isoform expression different across samples?



	Isoform A	Isoform B
Sample 1	5	12
Sample 2	15	3

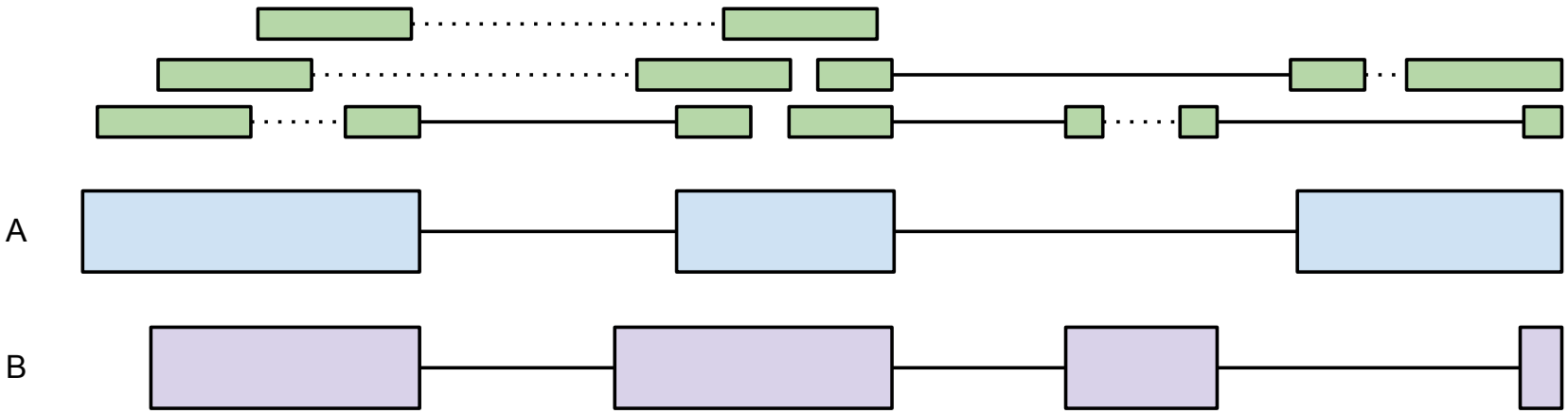
Questions about Isoform Structure

- Are there any unannotated isoforms present?
- How do we use them to derive novel structures? Is assembly feasible?



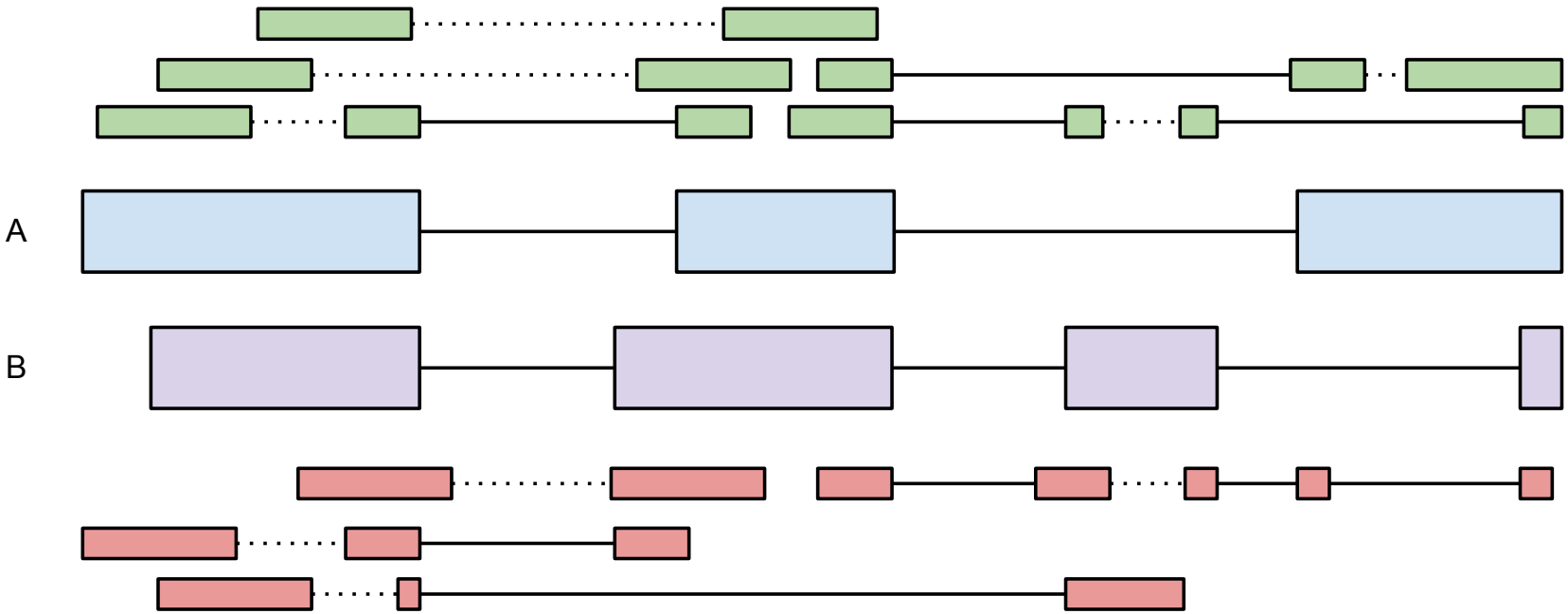
Questions about Isoform Structure

- Are there any unannotated isoforms present?
- How do we use them to derive novel structures? Is assembly feasible?



Questions about Isoform Structure

- Are there any unannotated isoforms present?
- How do we use them to derive novel structures? Is assembly feasible?



Outline

Overview of Algorithm

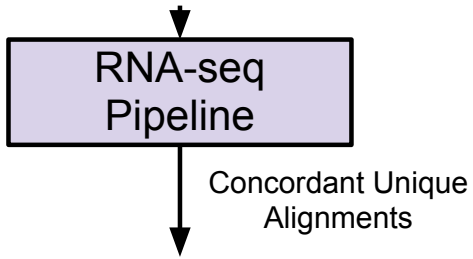
Steps

1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites

Overview of Algorithm

Steps

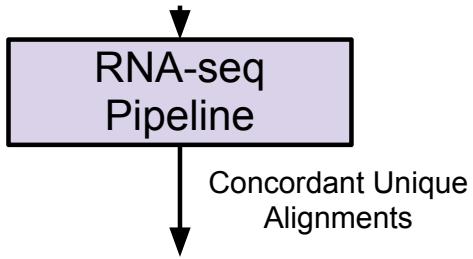
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Overview of Algorithm

Steps

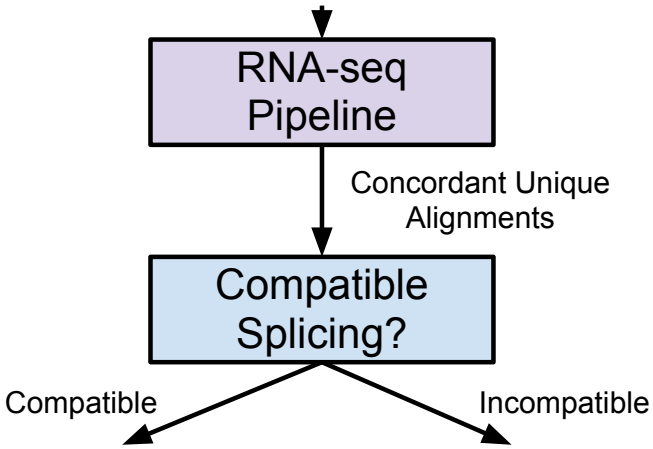
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Overview of Algorithm

Steps

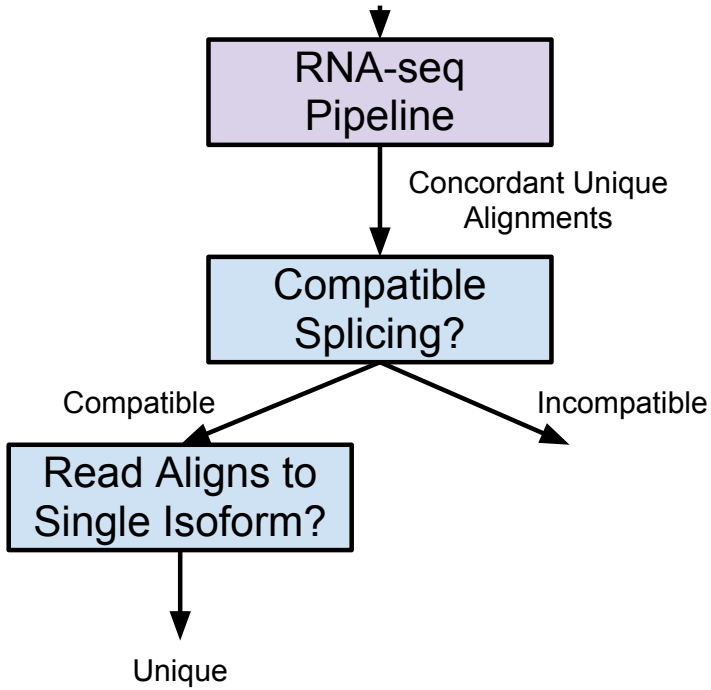
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Overview of Algorithm

Steps

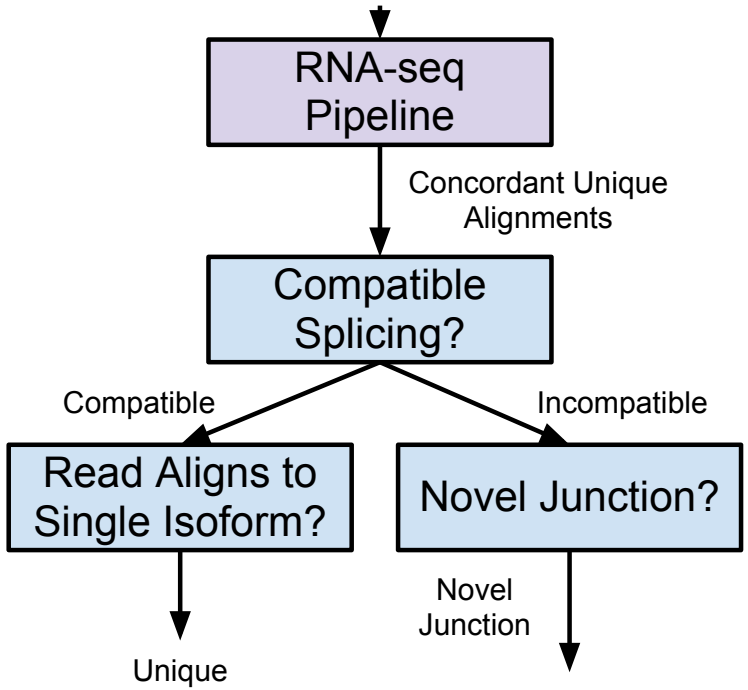
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Overview of Algorithm

Steps

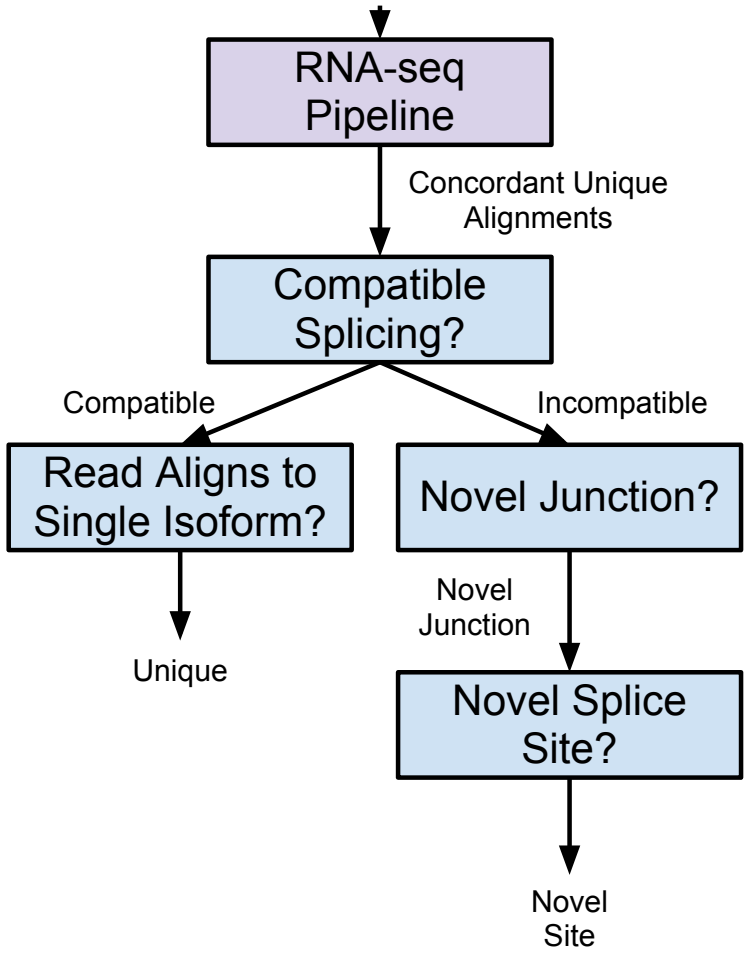
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Overview of Algorithm

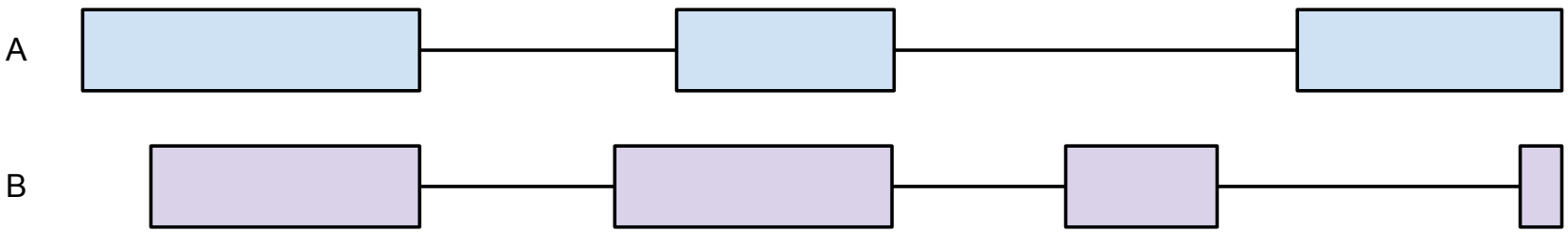
Steps

1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



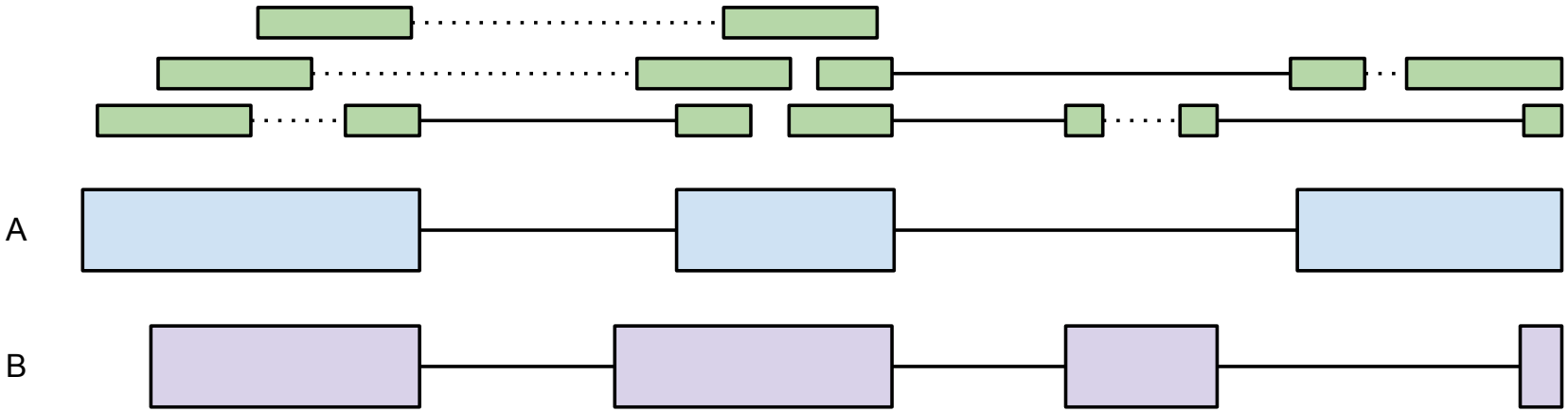
Compatibility and Uniqueness

- To be *compatible* a read must align completely within the exons and the read gaps should exactly match the introns over the read extent
- To be *uniquely mapped* a read must be compatible with only a single isoform



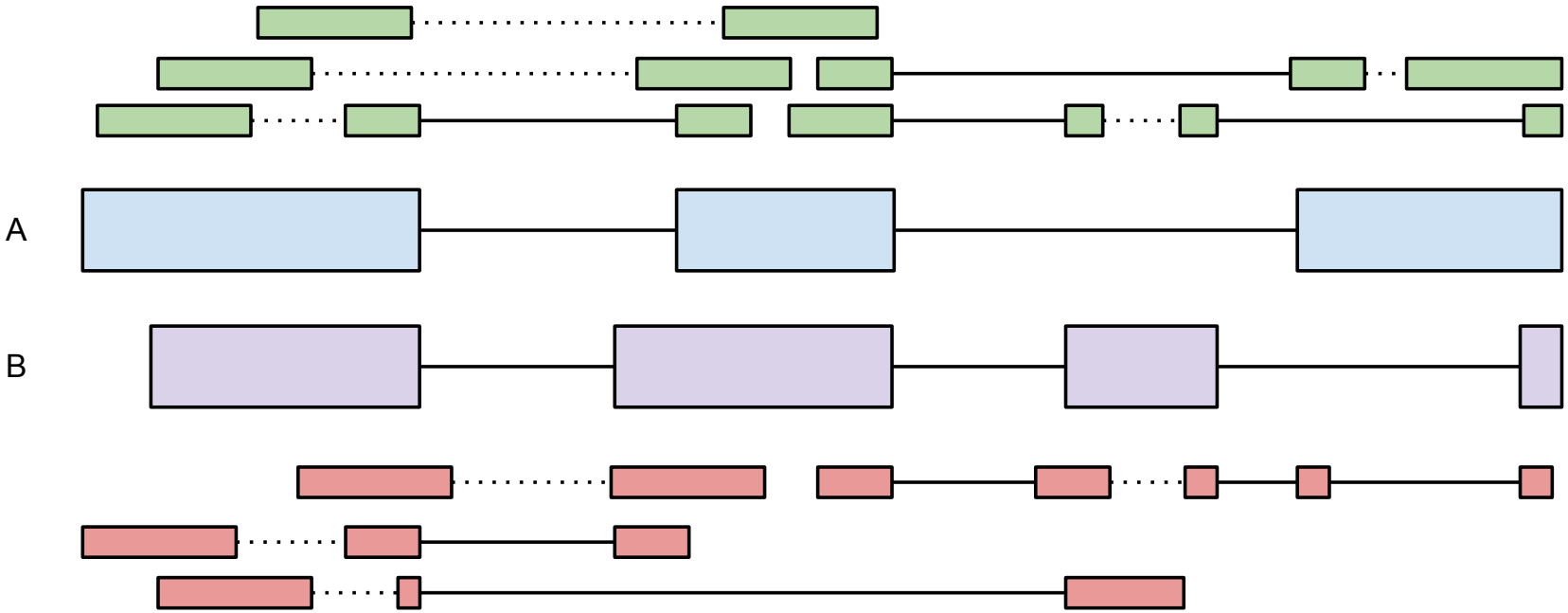
Compatibility and Uniqueness

- To be *compatible* a read must align completely within the exons and the read gaps should exactly match the introns over the read extent
- To be *uniquely mapped* a read must be compatible with only a single isoform



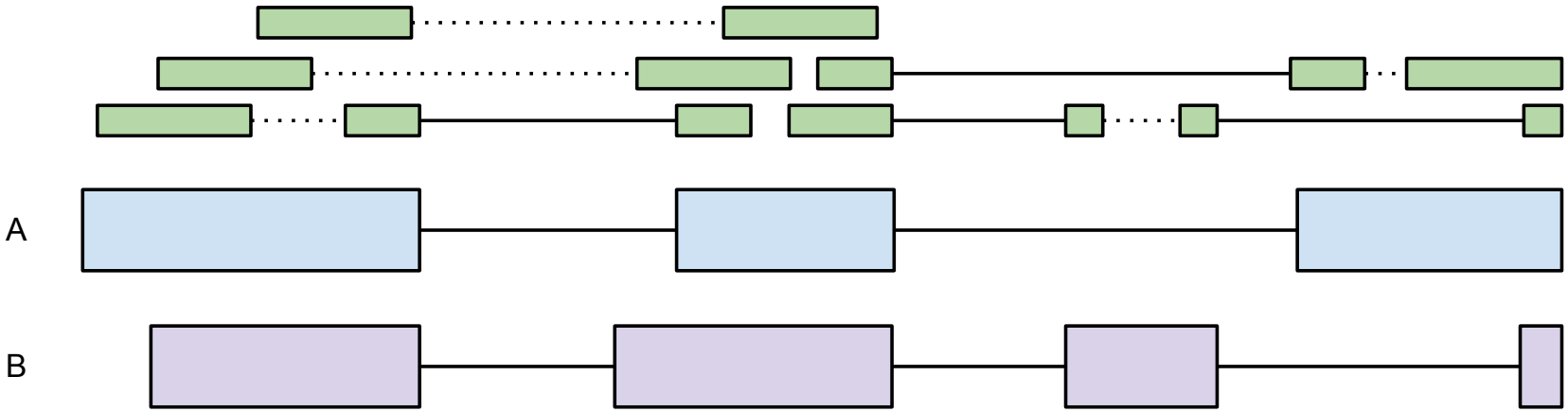
Compatibility and Uniqueness

- To be *compatible* a read must align completely within the exons and the read gaps should exactly match the introns over the read extent
- To be *uniquely mapped* a read must be compatible with only a single isoform



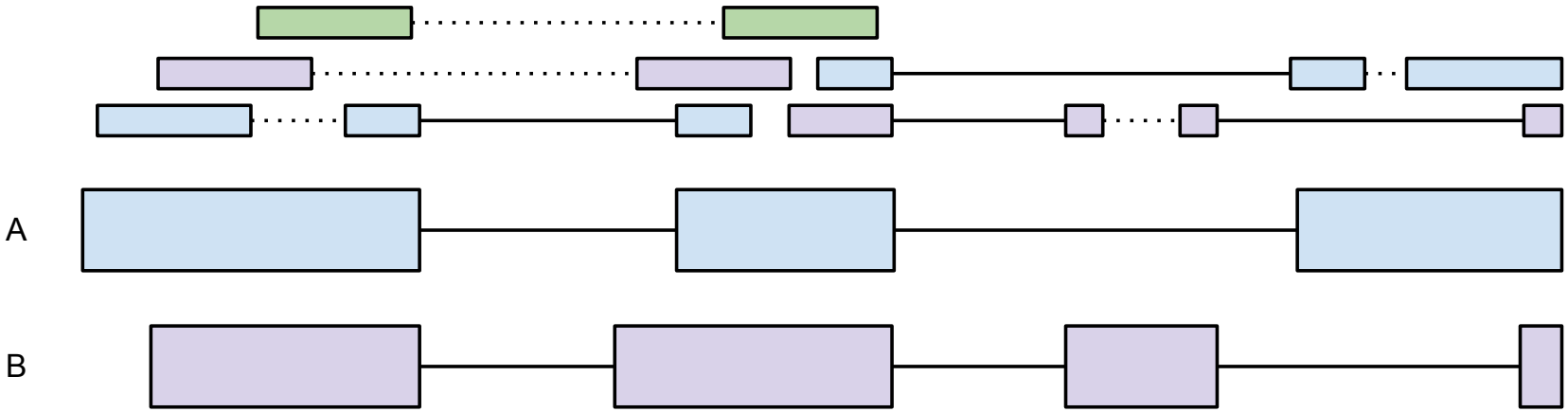
Compatibility and Uniqueness

- To be *compatible* a read must align completely within the exons and the read gaps should exactly match the introns over the read extent
- To be *uniquely mapped* a read must be compatible with only a single isoform



Compatibility and Uniqueness

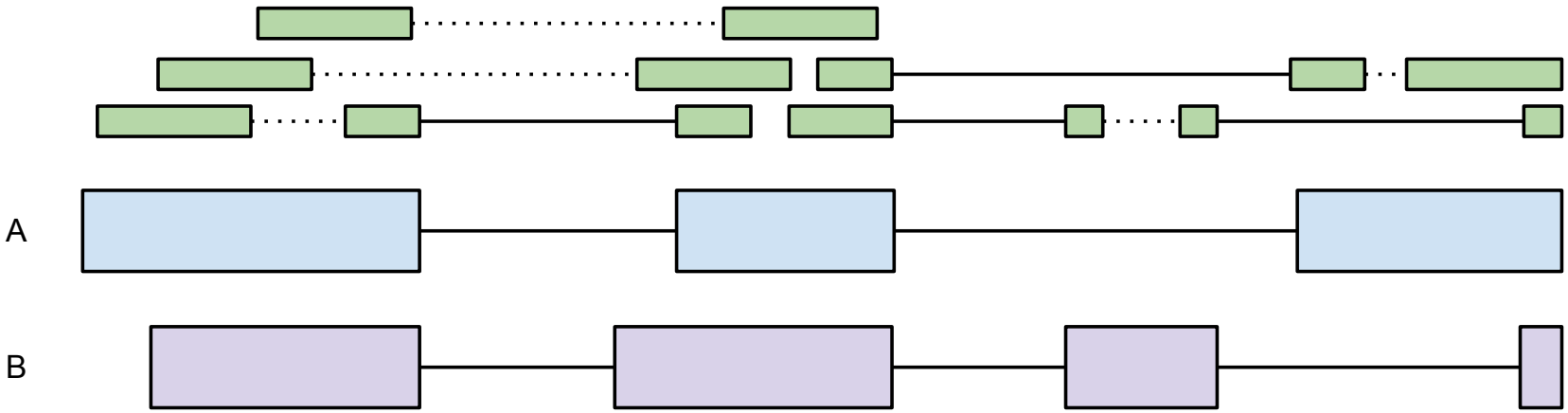
- To be *compatible* a read must align completely within the exons and the read gaps should exactly match the introns over the read extent
- To be *uniquely mapped* a read must be compatible with only a single isoform



Novelty: Grabbing the Low-hanging Fruit

Levels of novelty

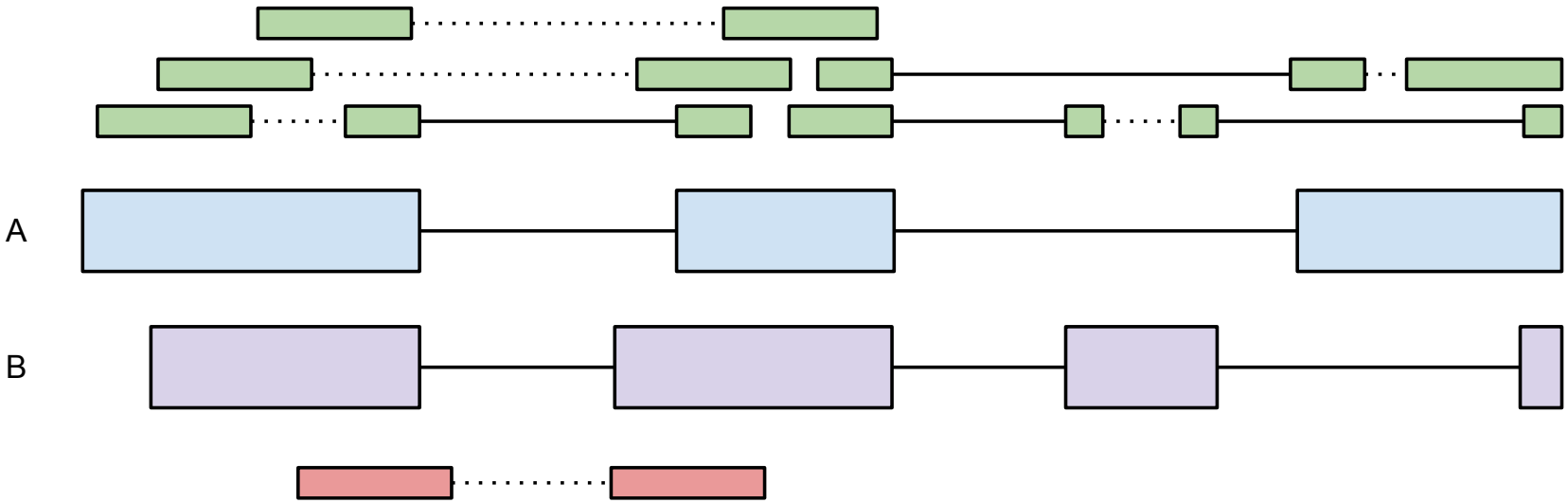
- Inconclusive
- Novel combination of events
- Novel junction
- Novel splice site



Novelty: Grabbing the Low-hanging Fruit

Levels of novelty

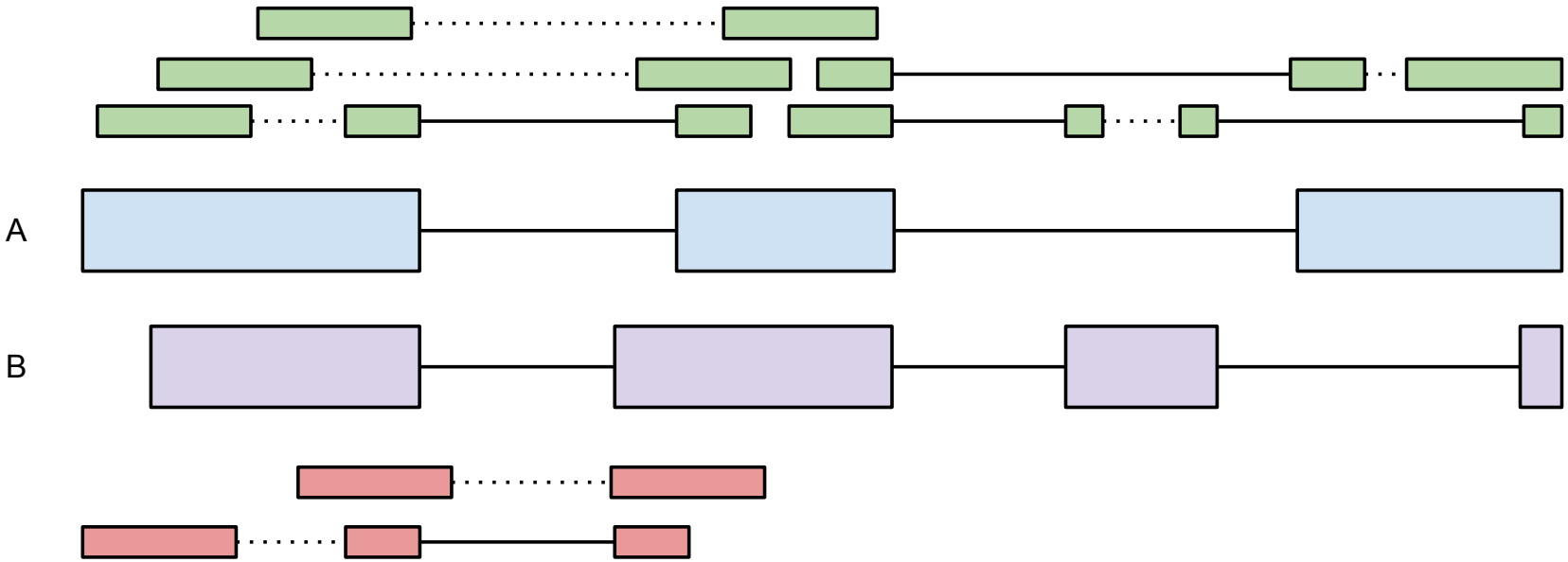
- Inconclusive
- Novel combination of events
- Novel junction
- Novel splice site



Novelty: Grabbing the Low-hanging Fruit

Levels of novelty

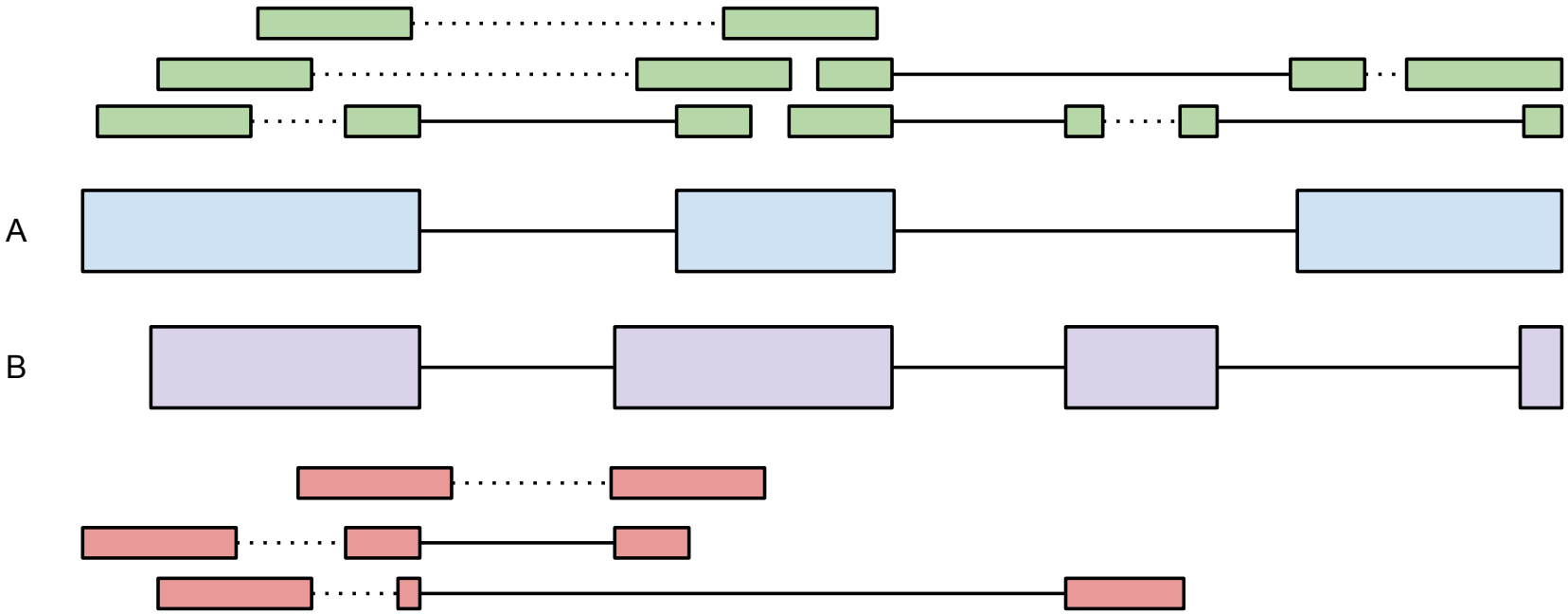
- Inconclusive
- Novel combination of events
- Novel junction
- Novel splice site



Novelty: Grabbing the Low-hanging Fruit

Levels of novelty

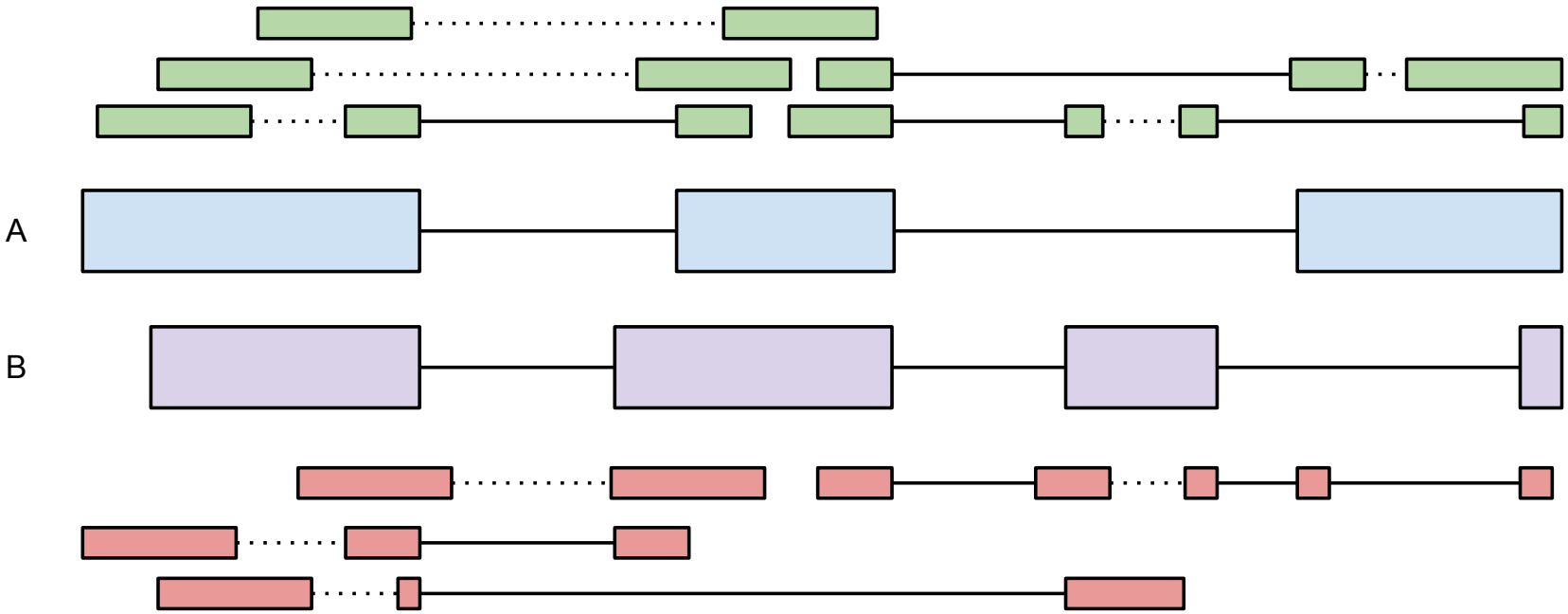
- Inconclusive
- Novel combination of events
- Novel junction
- Novel splice site



Novelty: Grabbing the Low-hanging Fruit

Levels of novelty

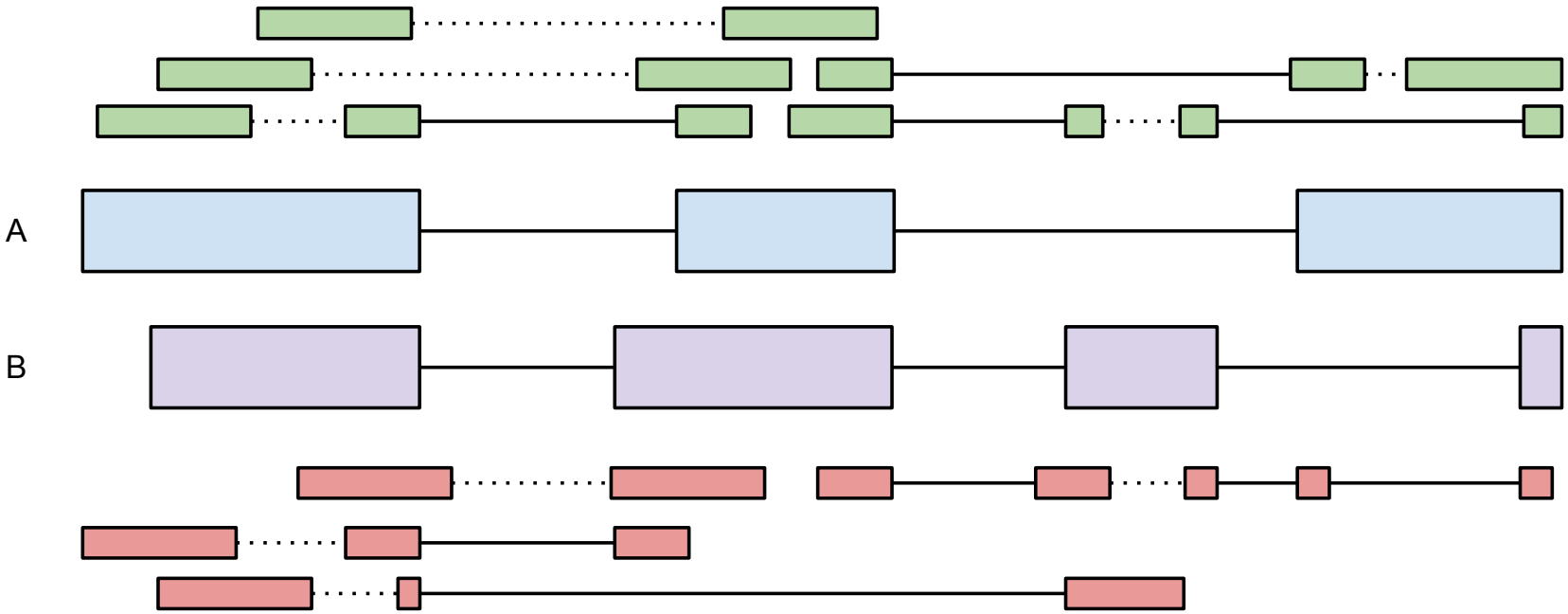
- Inconclusive
- Novel combination of events
- Novel junction
- Novel splice site



Novelty: Grabbing the Low-hanging Fruit

Levels of novelty

- Inconclusive
- Novel combination of events
- Novel junction
- Novel splice site



Outline

Example

- Illumina paired-end 75nt RNA-seq reads from two matched human tissue samples: tumor and normal
- Strand inferred during alignment from splice directions
- Stored in a BAM file
- Subset to the ALDOA (aldolase) gene

Outline

Data Loading Overview

We need two types of data for the analysis:

1. Transcript annotations (gene models)
2. Read alignments from the two samples

For this demonstration, we will focus on the ALDOA gene.

Finding ALDOA

To load the data, we need the genomic interval of ALDOA.

```
> aldoa_eg <- org.Hs.egSYMBOL2EG$ALDOA
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
> aldoa_gr <- exons(txdb, vals = list(gene_id = aldoa_eg),
+                   columns = c("tx_id", "gene_id"))
> aldoa_range <- range(aldoa_gr)
```

Forming the Transcript Models

Simplest method is calling `exonsBy`, but its API does not support restrictive queries, or extra columns. Thus, we form our models directly from `aldoa_gr`:

```
> aldoa_vals <- values(aldoa_gr)
> values(aldoa_gr) <- NULL
> tx <- multisplit(aldoa_gr, aldoa_vals$tx_id)
> tx_to_val <- match(names(tx), unlist(aldoa_vals$tx_id))
> values(tx)$gene_id <-
+   rep(unlist(aldoa_vals$gene_id),
+       elementLengths(aldoa_vals$tx_id))[tx_to_val]
> values(tx)$tx_id <- names(tx)
```

Exercise: `exonsBy`

Problem

As an alternative to last slide, use `exonsBy` and `subsetByOverlaps` to get the basic transcript structures for the ALDOA gene. *Bonus: get the `gene_id` my merging with `transcripts` return value.*

Exercise: exonsBy

Problem

As an alternative to last slide, use exonsBy and subsetByOverlaps to get the basic transcript structures for the ALDOA gene. *Bonus: get the gene_id my merging with transcripts return value.*

Solution

```
> exons_grl <- exonsBy(txdb)
> ans <- subsetByOverlaps(exons_grl, aldoa_range)
> values(ans)$tx_id <- names(ans)
> tx_gr <- transcripts(txdb, columns = c("tx_id", "gene_id"))
> values(ans)$gene_id <-
+   drop(values(tx_gr)$gene_id)[match(names(ans),
+                                     values(tx_gr)$tx_id)]
```

Merging the Gene Symbols

Would be trivial in our case, but here is a general demonstration:

```
> gene_id_keys <-
+   values(tx)$gene_id[!is.na(values(tx)$gene_id)]
> tx_gene_sym <- rep.int(NA, length(tx))
> tx_gene_sym[!is.na(values(tx)$gene_id)] <-
+   unlist(mget(gene_id_keys, org.Hs.egSYMBOL,
+               ifnotfound = NA),
+          use.names = FALSE)
> values(tx)$gene_sym <- tx_gene_sym
```

Selecting Only the CCDS Models

Just take my word for it:

```
> tx <- tx[c(2, 4, 5, 7)]
```

Reading the Alignments

We obtain the normal BAM file for demonstration purposes:

```
> extdatadir <- system.file("extdata",
+                           package = "isoformExprTutorial")
> files <- tools::list_files_with_exts(extdatadir, "bam")
> names(files) <- tools::file_path_sans_ext(basename(files))
> bamFiles <- Rsamtools::BamFileList(files)
> bam <- bamFiles$normal
```

And read it into a *GappedAlignments* object:

```
> param <- ScanBamParam(tag = "XS", which = aldoa_range)
> ga <- readGappedAlignments(path(bam),
+                            use.names = TRUE,
+                            param = param)
```

We request the *XS* tag, the strand as determined by the aligner from the splice directions.

Ranges from the Alignments

The `grglist` function returns the alignment ranges as a *GRangesList*.

```
> reads <- grglist(ga)
> metadata(reads)$bamfile <- bam
```

We store the BAM path in the metadata to track provenance.

Exercise: `metadata`

Problem

Store the *ScanBamParam* object in the metadata.

Exercise: `metadata`

Problem

Store the *ScanBamParam* object in the metadata.

Solution

```
> metadata(reads)$param <- param
```

Splices

We use a custom function, `elementGaps`, to find the gaps within each element of the *GRangesList*:

```
> splices <- elementGaps(reads)
> values(splices)$XS <- values(reads)$XS
```

All metadata needs to be carried over explicitly.

elementGaps: Simple Implementation

```
> elementGaps <- function(reads) {
+   psetdiff(unlist(range(reads), use.names=FALSE), reads)
+ }
```

Problem: the call to `range(reads)` is painfully slow.

elementGaps: *Optimization* via partitioning

```
> elementGaps <- function(x) {
+   x_flat <- unlist(x, use.names = FALSE)
+   egaps <- gaps(ranges(x))
+   first_segment <- start(PartitioningByWidth(x))
+   sn <- seqnames(x_flat)[first_segment][togroup(egaps)]
+   strand <- strand(x_flat)[first_segment][togroup(egaps)]
+   relist(GRanges(sn, unlist(egaps, use.names = FALSE),
+                  strand, seqlengths = seqlengths(x)),
+          egaps)
+ }
```

Assumption

Strand and chromosome is the same within the elements, i.e., no trans-splicing, genomic rearrangements, etc.

Exercise: Partitionings

Problem

Use a *PartitioningByWidth* to find the start of the first exon in each transcript.

Exercise: Partitionings

Problem

Use a *PartitioningByWidth* to find the start of the first exon in each transcript.

Solution

```
> tx_part <- PartitioningByWidth(tx)
> tx_flat <- unlist(tx, use.names = FALSE)
> start(tx_flat)[start(tx_part)]

[1] 30064411 30075600 30075818 30076994
```

Pairing the Reads

We depend on each end of a pair sharing the same name:

```
> pairs <- split(unlist(reads, use.names=FALSE),
+                factor(names(reads)[togroup(reads)],
+                       unique(names(reads))))
> metadata(pairs) <- metadata(reads)
```

XS information is shared within pairs:

```
> xs <- values(reads)$XS
> has_xs <- !is.na(xs)
> pair_xs <- setNames(rep.int(NA, length(pairs)),
+                     names(pairs))
> pair_xs[names(reads)[has_xs]] <- xs[has_xs]
> values(pairs)$XS <- unname(pair_xs)
```

Resolving the Strand from *XS*

The mapping is direct, except `NA` maps to `*`.

```
> xs <- values(pairs)$XS
> strand <- ifelse(!is.na(xs) & xs != "?", xs, "*")
> strand(pairs) <- relist(Rle(strand, elementLengths(pairs)),
+                         pairs)
```

Pairing and Stranding the Splices

We have factored the preceding code into two functions.

```
> splices <- pairReadRanges(splices)
> splices <- strandFromXS(splices)
```

Batch Processing Both Tumor and Normal

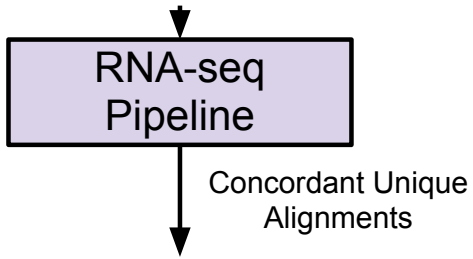
```
> normal <- readReadRanges(bamFiles$normal)
> tumor <- readReadRanges(bamFiles$tumor)
```

Outline

Overview of Algorithm

Steps

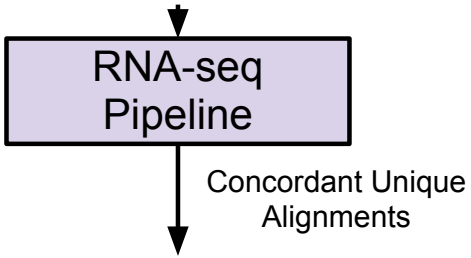
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Overview of Algorithm

Steps

1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Finding Overlaps

```
> hits <- findOverlaps(pairs, tx)
```

Extracting the Hits

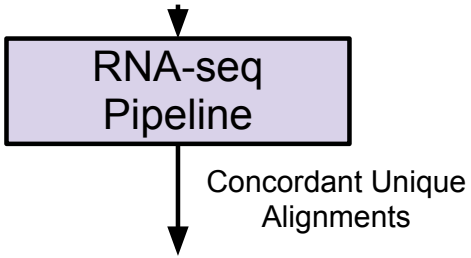
```
> hit_pairs <- ranges(pairs)[queryHits(hits)]
> hit_splices <- ranges(splices)[queryHits(hits)]
> hit_tx <- ranges(tx)[subjectHits(hits)]
```

Memory inefficient, but permits vectorized operations

Overview of Algorithm

Steps

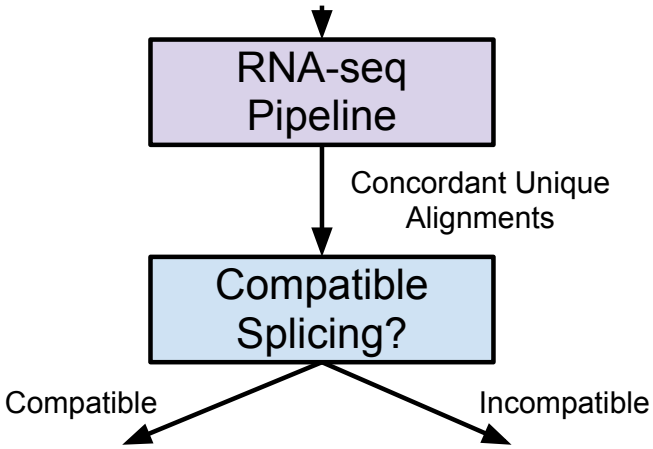
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Overview of Algorithm

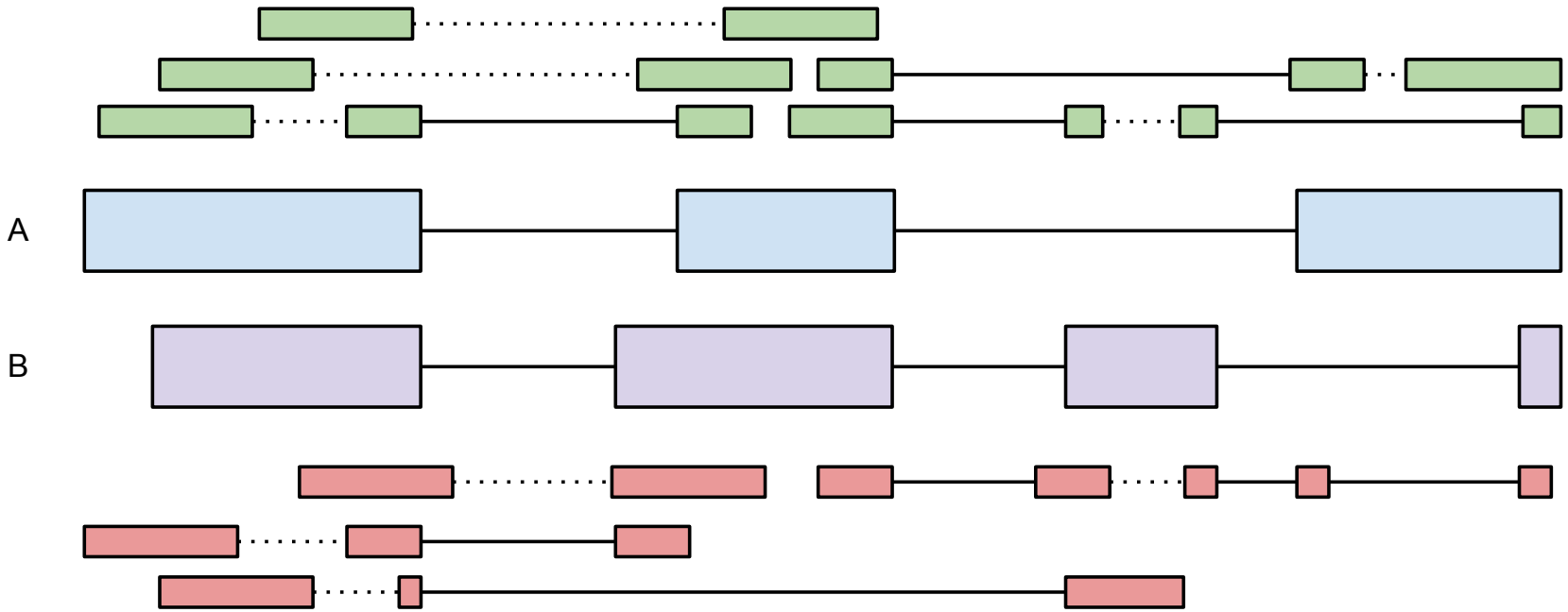
Steps

1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Compatibility Checking

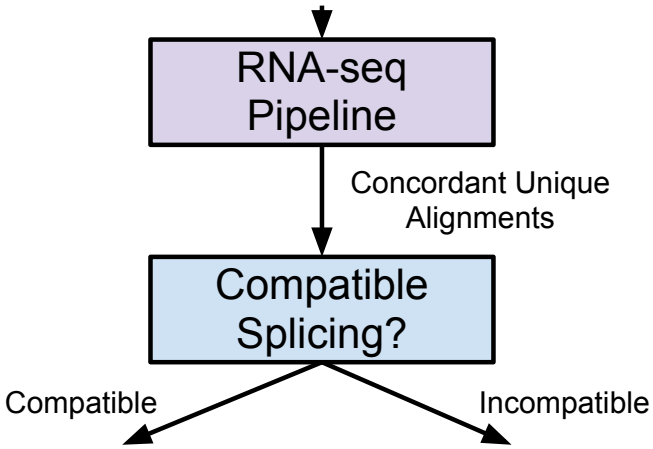
```
> read_within <-
+   elementLengths(setdiff(hit_pairs, hit_tx)) == 0L
> tx_within <-
+   elementLengths(intersect(hit_tx, hit_splices)) == 0L
> compatible <- read_within & tx_within
```



Overview of Algorithm

Steps

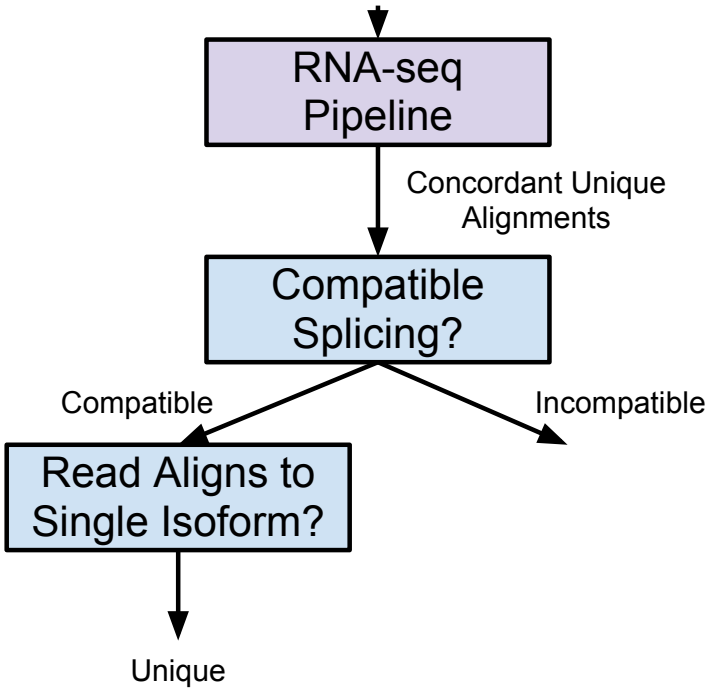
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Overview of Algorithm

Steps

1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Uniquely Mapped Reads

We find the compatible hits where the read is represented only once:

```
> compat_hits <- hits[compatible]
> reads_unique <- tabulate(queryHits(compat_hits),
+                          queryLength(compat_hits)) == 1L
> unique <- logical(length(hits))
> unique[compatible] <- reads_unique[queryHits(compat_hits)]
```

Annotating the Overlaps

Like any other *Vector*, *Hits* can have element metadata:

```
> strand_specific <-
+   all(strand(pairs) != "*")[queryHits(hits)]
> values(hits) <- DataFrame(strand_specific,
+                           compatible,
+                           unique)
```

Exercise: Add Spliced Indicator

Problem

Add a column to the `hits` element metadata indicating whether the read had a splice (ignoring XS tag and assuming all reads are paired).

Exercise: Add Spliced Indicator

Problem

Add a column to the `hits` element metadata indicating whether the read had a splice (ignoring XS tag and assuming all reads are paired).

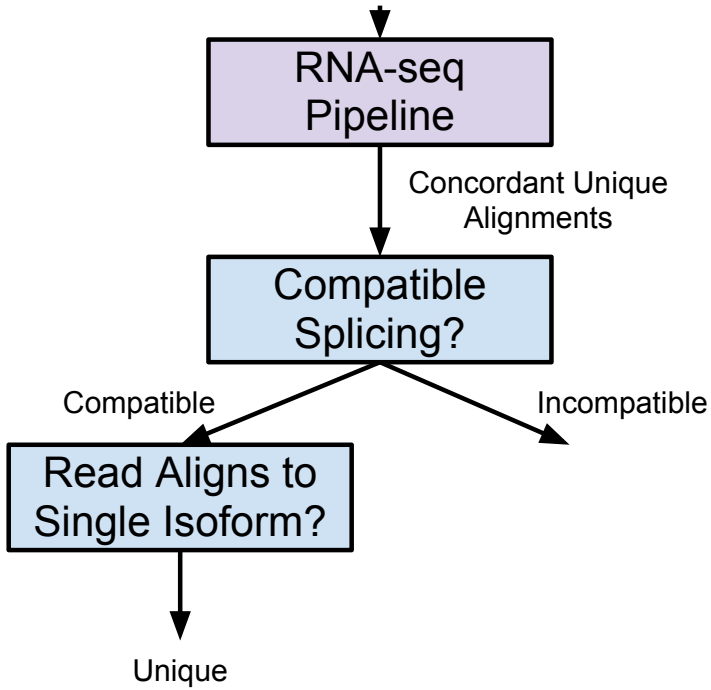
Solution

```
> values(hits)$spliced <-
+   (elementLengths(pairs) > 2)[queryHits(hits)]
```

Overview of Algorithm

Steps

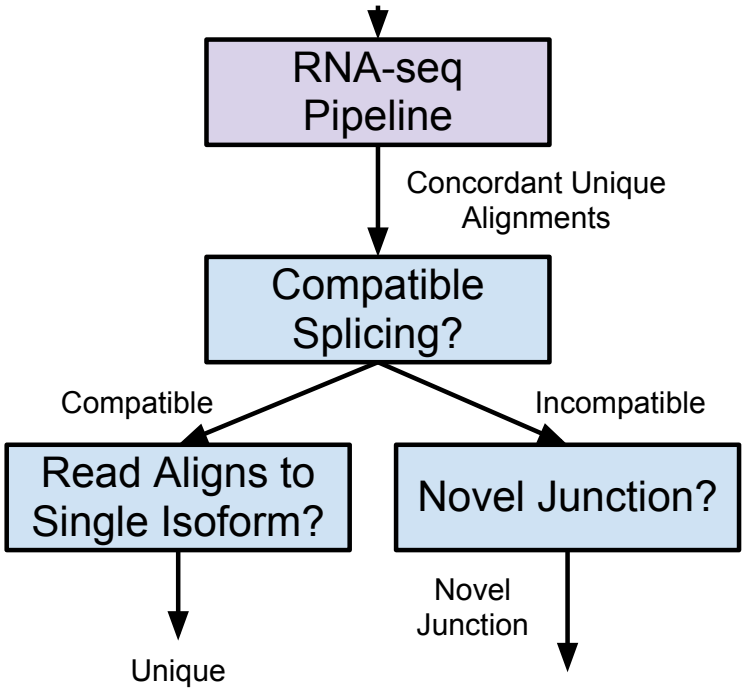
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Overview of Algorithm

Steps

1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and isoforms with any exonic overlap
3. Split the alignments into two bins: *compatible* with splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform
5. For the *incompatible*, find those indicating *novel* junctions or splice sites



Junction Counting

Simplest algorithm: form keys from *GRanges* and hashing.

Key generator

```
> gr2key <- function(x) {
+   paste(seqnames(x), start(x), end(x), strand(x),
+         sep = ":")
+ }
```

Key Parser

```
> key2gr <- function(x, ...) {
+   key_mat <- matrix(unlist(strsplit(x, ":", fixed=TRUE)),
+                     nrow = 4)
+   GRanges(key_mat[1,],
+           IRanges(as.integer(key_mat[2,]),
+                   as.integer(key_mat[3,])),
+           key_mat[4,], ...)
+ }
```

Junction Counting

```
> introns <- elementGaps(tx)
> introns_flat <- unlist(introns, use.names = FALSE)
> tx_keys <- gr2key(introns_flat)

> splices_flat <- unlist(splices, use.names = FALSE)
> splice_table <- table(gr2key(splices_flat))
> splice_summary <-
+   key2gr(names(splice_table),
+          score = as.integer(splice_table),
+          novel = !names(splice_table) %in% tx_keys,
+          seqlengths = seqlengths(splices))
```

Aggregating over the Transcripts

```
> countByTx <- function(x) {
+   tabulate(subjectHits(hits)[x], subjectLength(hits))
+ }
> compatible_strand <-
+   countByTx(with(values(hits),
+                  compatible & strand_specific))
> counts <- DataFrame(compatible_strand,
+                     lapply(values(hits)[-1], countByTx))
```

Batch Processing the Samples

```
> normal_hits <- findIsoformOverlaps(normal)
> normal_counts <- countIsoformHits(normal_hits)
> normal_splices <- summarizeSplices(normal)
> tumor_hits <- findIsoformOverlaps(tumor)
> tumor_counts <- countIsoformHits(tumor_hits)
> tumor_splices <- summarizeSplices(tumor)
```

Outline

Combining the Samples

We reshape the data into a *SummarizedExperiment*:

```
> assays <- mapply(cbind, normal_counts, tumor_counts,
+                  SIMPLIFY = FALSE)
> colData <- DataFrame(tumorStatus = c("tumor", "normal"))
> rownames(colData) <- colData$tumorStatus
> se <- SummarizedExperiment(assays, tx, colData)
```

Comparing the Top Two Isoforms

We use a Fisher Test to check whether the balance is shifting between top two isoforms (by uniquely mapped reads), between tumor and normal.

```
> uc <- assay(se, "unique")
> uc_ord <- order(rowSums(uc), decreasing = TRUE)
> uc_top <- uc[head(uc_ord, 2),]
> fisher.test(uc_top)$estimate

odds ratio
  8.430957
```

Plotting the Isoform Coverage

Setup

Get Uniquely Mapped Reads

```
> getUniqueReads <- function(reads, hits) {
+   sel <- values(hits)$unique &
+          subjectHits(hits) %in% c(1, 4)
+   reads[unique(queryHits(hits)[sel])]
+ }
> normal_uniq <- getUniqueReads(normal, normal_hits)
> tumor_uniq <- getUniqueReads(tumor, tumor_hits)
> both_uniq <- mstack(normal = unlist(normal_uniq),
+                     tumor = unlist(tumor_uniq))
```

Plotting the Isoform Coverage

Setup

Get Uniquely Mapped Splices

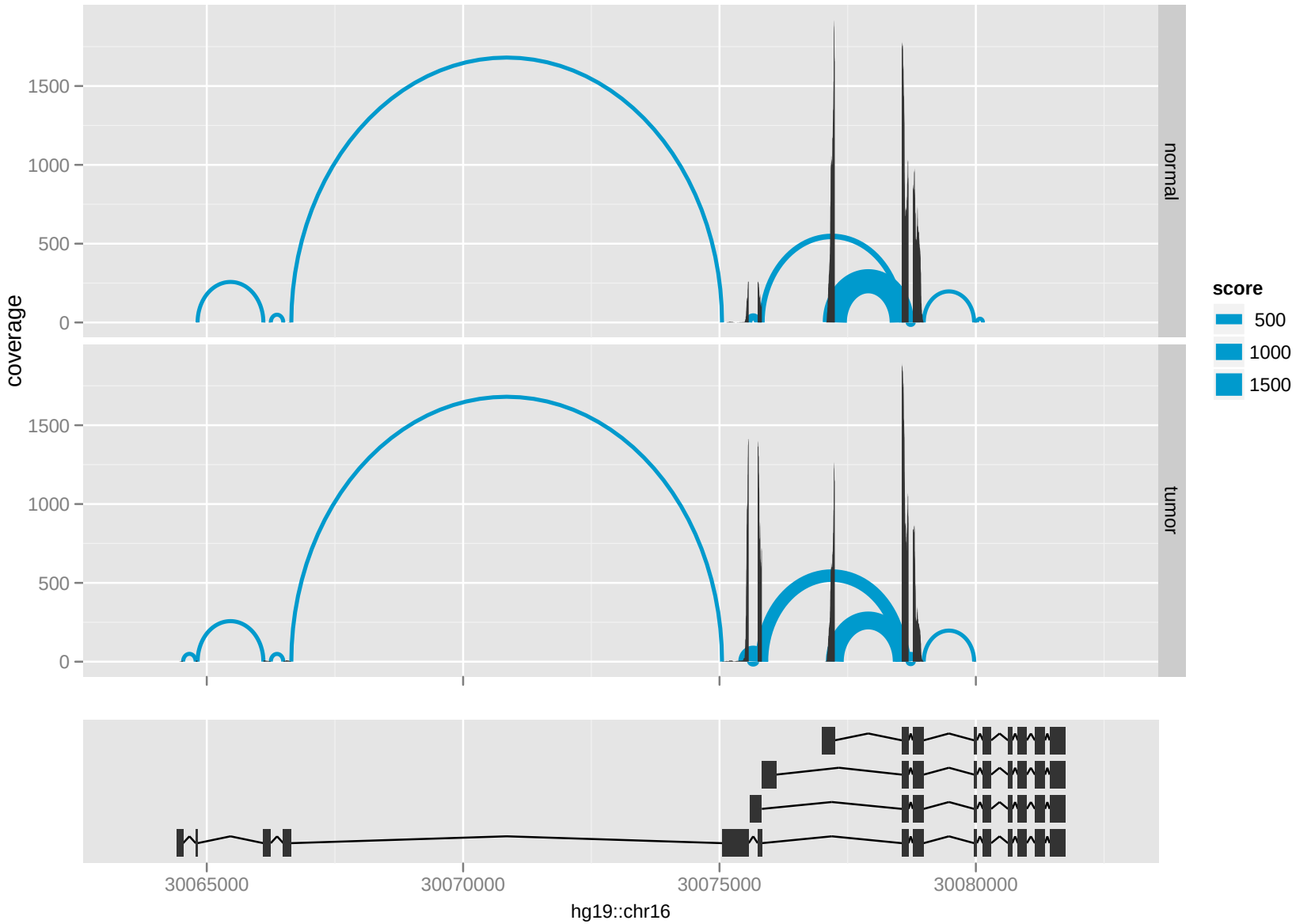
```
> normal_uniq_splices <- summarizeSplices(normal_uniq)
> tumor_uniq_splices <- summarizeSplices(tumor_uniq)
> uniq_splices <- mstack(normal = normal_uniq_splices,
+                        tumor = tumor_uniq_splices)
```

Plotting the Isoform Coverage

We leverage the `autoplot` function from *ggbio*:

```
> read_track <- autoplot(uniq_splices, geom = "arch",
+                        aes(size = score,
+                            height = width / 5),
+                        color = "deepskyblue3",
+                        ylab = "coverage",
+                        facets = name ~ .) +
+   stat_coverage(both_uniq, facets = name ~ .)
> tx_16 <- keepSeqlevels(tx, "chr16")
> tx_track <- autoplot(tx_16, geom = "alignment", ylab = "")
> tracks(read_track, tx_track, heights = c(3, 1))
```

Coverage Plot

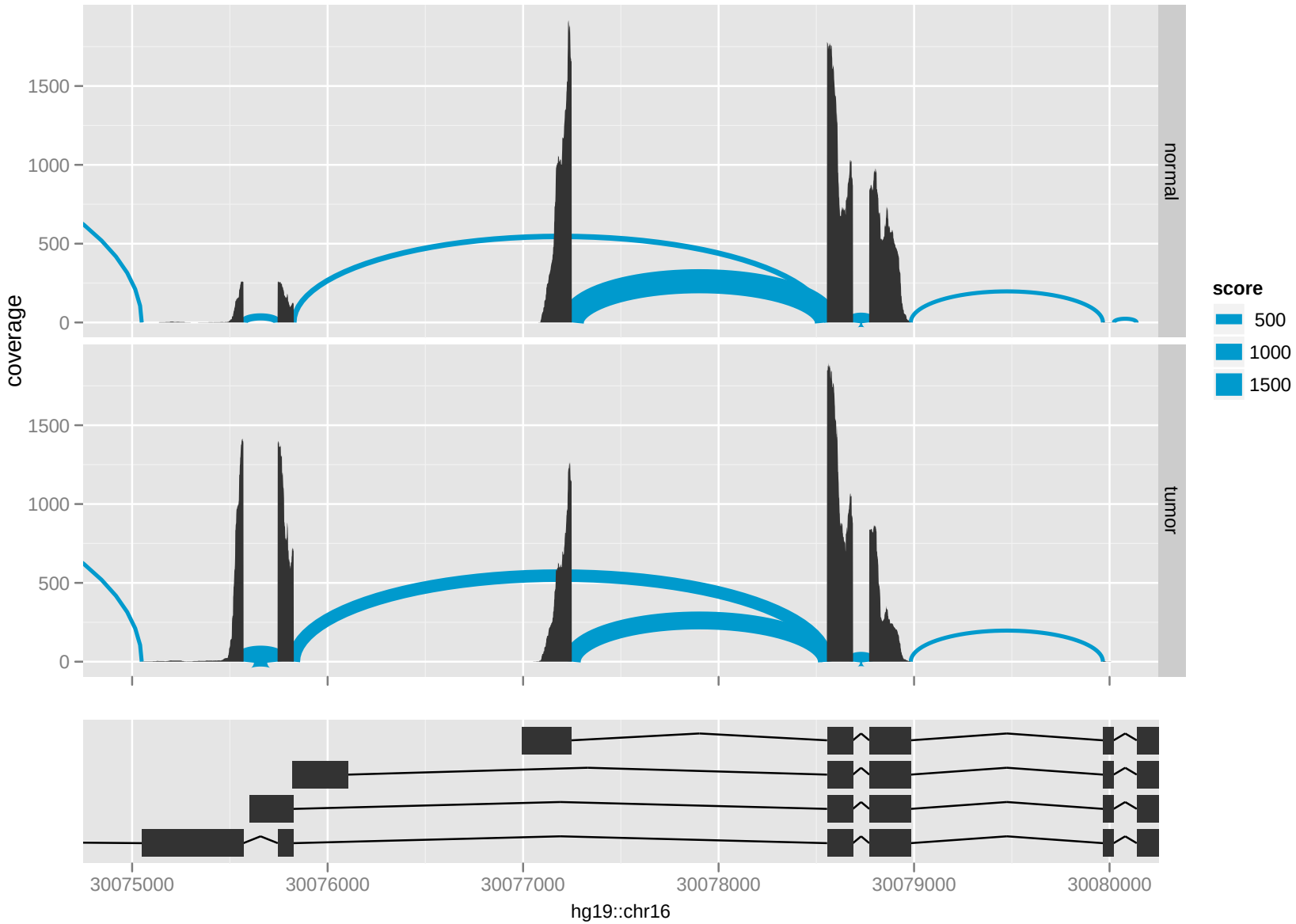


Zooming into the Plot

The `xlim` argument restricts the genomic interval:

```
> tracks(read_track,
+        tx_track, heights = c(3, 1),
+        xlim = c(30075000, 30080000))
```

Zoomed Coverage Plot



Exercise: Zoom Using the Coverage

Problem

Zoom to a similar region of interest by finding a window around the tallest peaks. Hint: `coverage` and `slice` would be useful here.

Exercise: Zoom Using the Coverage

Problem

Zoom to a similar region of interest by finding a window around the tallest peaks. Hint: `coverage` and `slice` would be useful here.

Solution

```
> cov_chr16 <- coverage(both_uniq)$chr16
> roi <- range(ranges(slice(cov_chr16, 1000)))
> roi <- roi + 500
> tracks(read_track,
+        tx_track, heights = c(3, 1),
+        xlim = roi)
```

Plotting the Novel Junctions

Setup

Get the Novel Splices

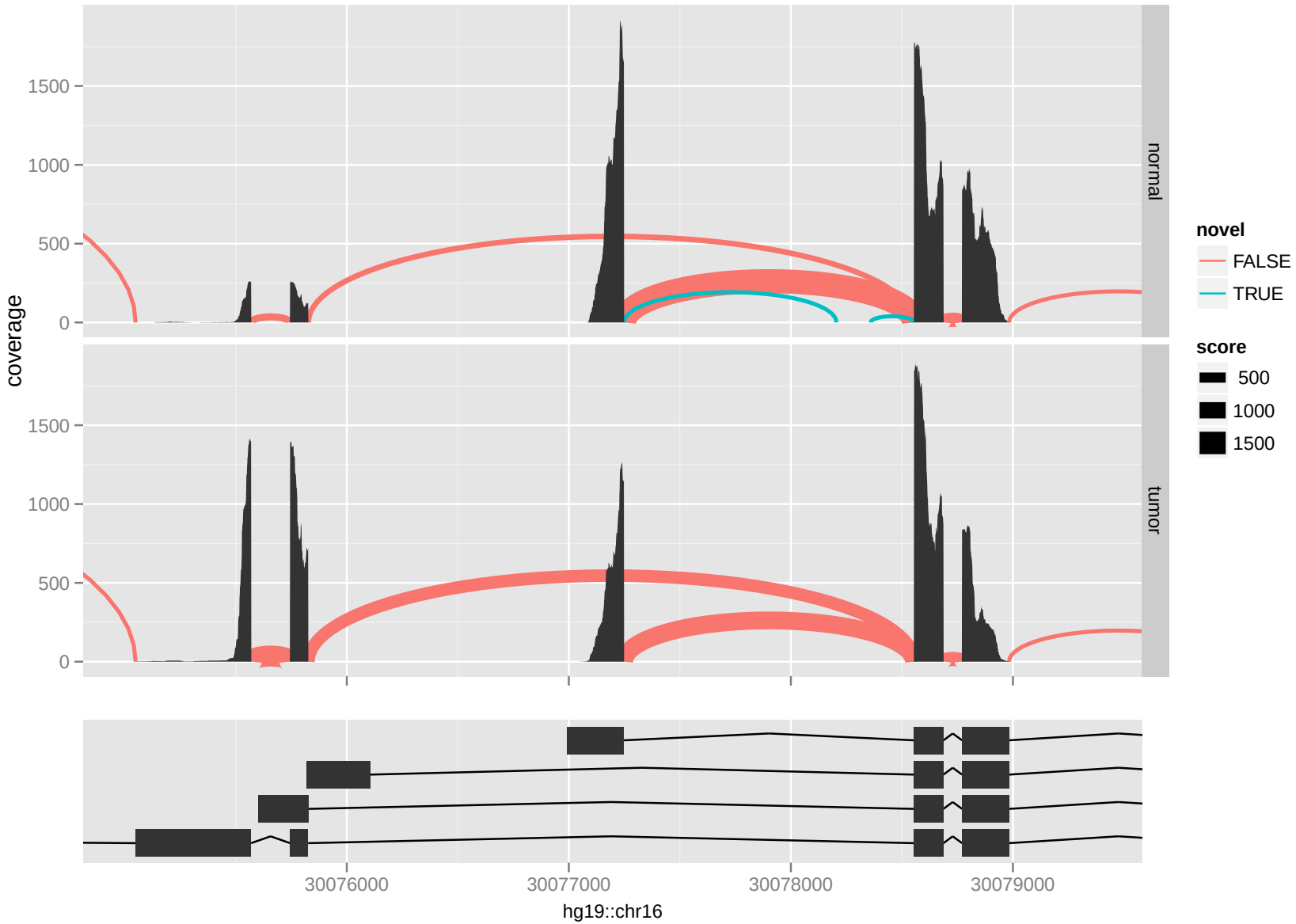
```
> all_splices <- mstack(normal = normal_splices,
+                       tumor = tumor_splices)
> novel_splices <-
+   all_splices[values(all_splices)$novel &
+               values(all_splices)$score == 9]
> uniq_novel_splices <- c(uniq_splices, novel_splices)
```

Plotting the Novel Junctions

This time, we specify the `color = novel` aesthetic:

```
> novel_track <- autoplot(uniq_novel_splices, geom = "arch",
+                         aes(size = score,
+                             height = width / 5,
+                             color = novel),
+                         ylab = "coverage",
+                         facets = name ~ .) +
+   stat_coverage(both_uniq, facets = name ~ .)
> tracks(novel_track, tx_track, heights = c(3, 1),
+        xlim = roi)
```

Coverage Plot, with Novel Junctions



Outline

Take Home Tips

- Use the right data structure for the right job
- Take advantage of metadata facilities
- Long ragged arrays: partitioning faster than looping over lists