The *Ranges Suite

Application to the analysis of RNA-seq data

Michael Lawrence

Genentech

July 23, 2012

Outline

Sequencing Approaches

Source	Genome, transcriptome, synthetic, ...
Enrichment	WGS, ChIP, PCR, poly-A, exome capture, ...

RNA-seq

- High-throughput sequencing of cDNA libraries
- Usually there is some enrichment for mRNAs (e.g. poly-A)
- Usually aligned to genome with splicing tolerance (e.g. GSNAP)

General Sequence Analysis Workflow

1. QA on raw instrument output, see *ShortRead*
2. Usually, external alignment of data, i.e., gSNAP via upcoming *gmapR*
3. Import of alignments and/or sequences into R
4. Analysis of sequences, alignments and enrichment patterns

The *Ranges Packages

IRanges

Base of the sequence analysis infrastructure in Bioconductor

- Data structures for interval datasets and genome-scale vectors
- Routines for finding regions of enrichment and overlap between features

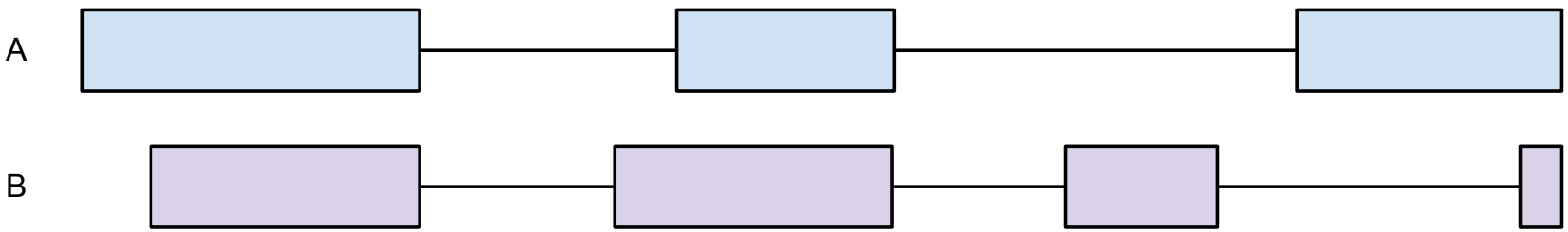
GenomicRanges

Extension of *IRanges* for genomic (biological) datasets, including sequence annotations and read alignments, with biology-specific overlap algorithms and other utilities.

Questions of Interest in RNA-seq

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

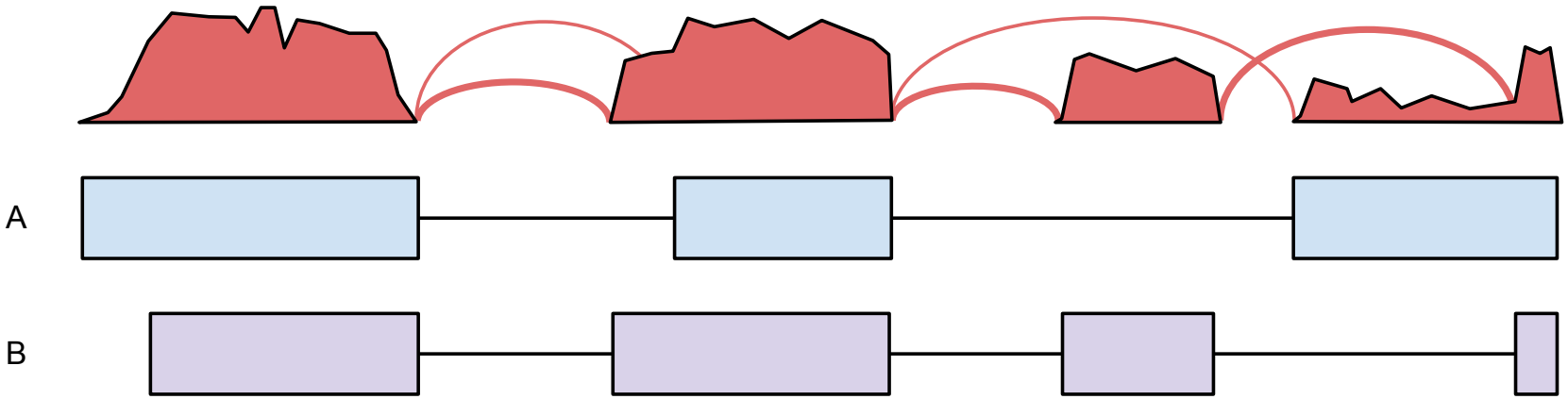
- Expression levels (from counting overlaps with gene annotations) and their differences between samples, alleles and splicing isoforms.
- Structure, novel splicing, gene fusions (from the alignments)
- Variants, RNA-editing (from the sequences)



Questions of Interest in RNA-seq

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

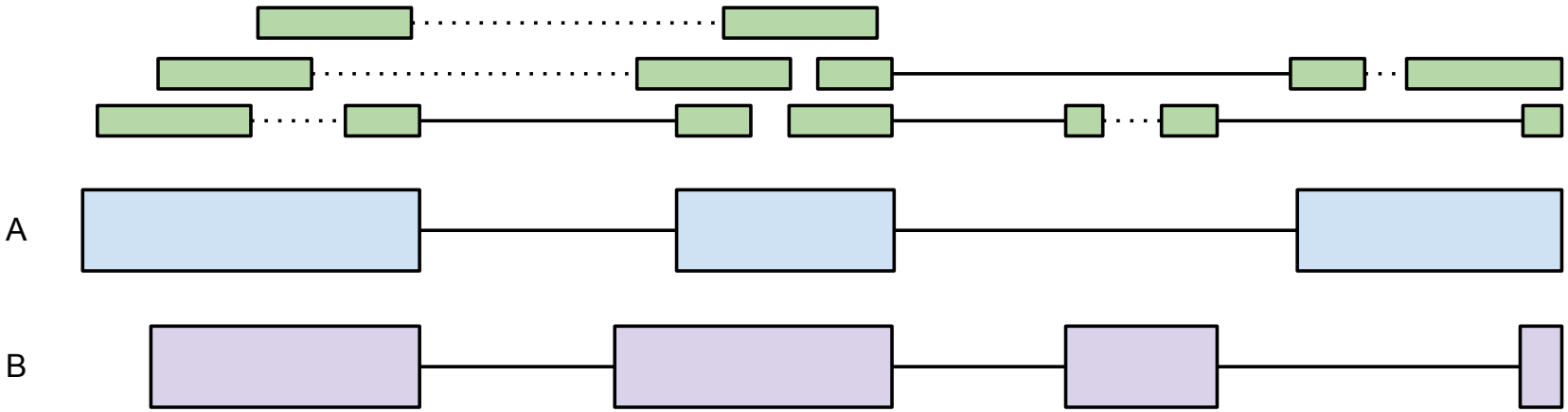
- Expression levels (from counting overlaps with gene annotations) and their differences between samples, alleles and splicing isoforms.
- Structure, novel splicing, gene fusions (from the alignments)
- Variants, RNA-editing (from the sequences)



Questions of Interest in RNA-seq

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

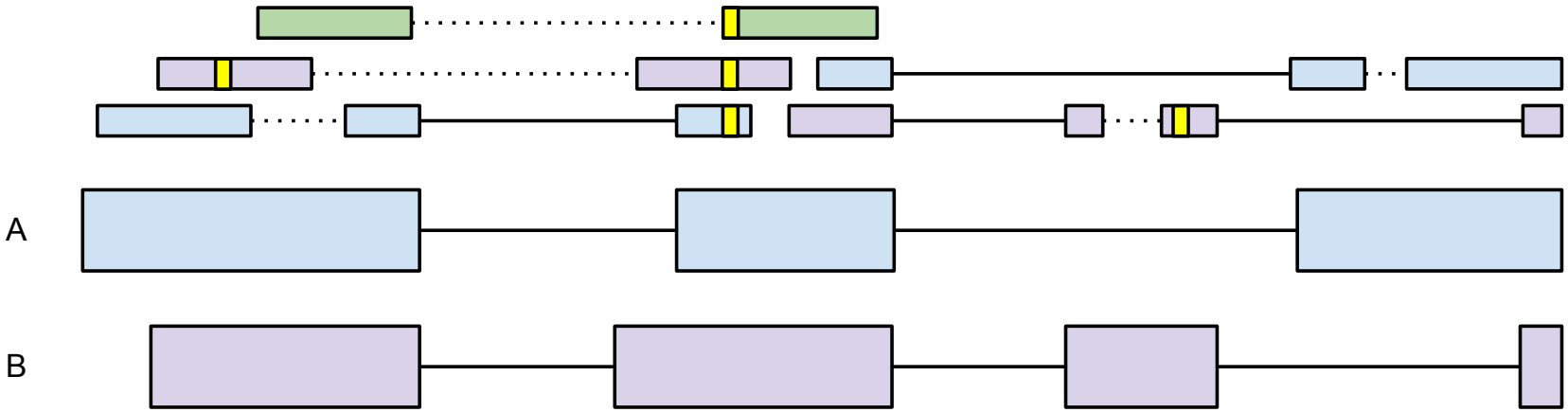
- Expression levels (from counting overlaps with gene annotations) and their differences between samples, alleles and splicing isoforms.
- Structure, novel splicing, gene fusions (from the alignments)
- Variants, RNA-editing (from the sequences)



Questions of Interest in RNA-seq

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

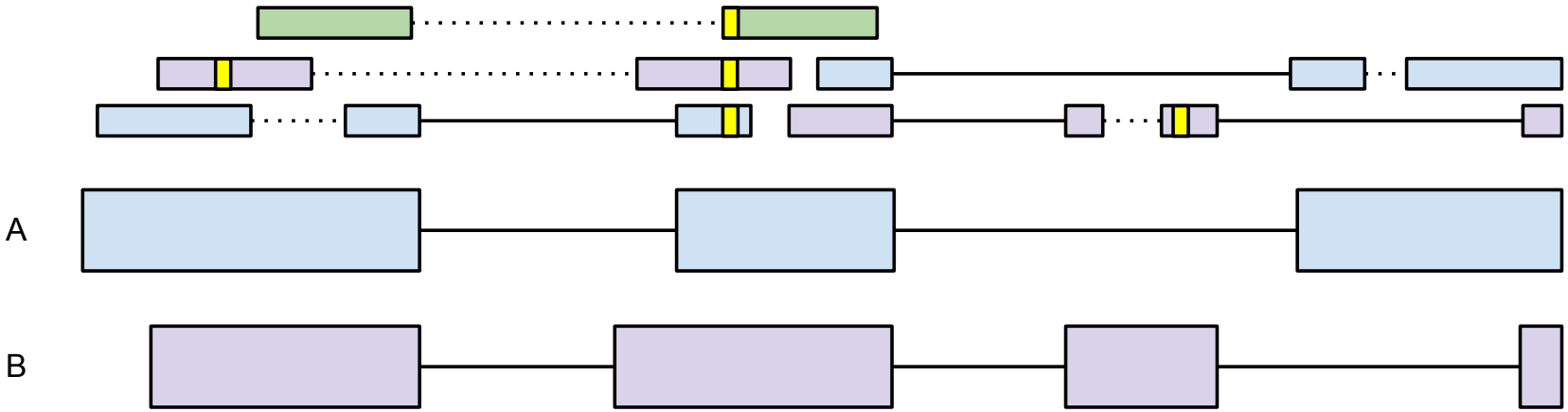
- Expression levels (from counting overlaps with gene annotations) and their differences between samples, alleles and splicing isoforms.
- Structure, novel splicing, gene fusions (from the alignments)
- Variants, RNA-editing (from the sequences)



Questions of Interest in RNA-seq

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

- Expression levels (from counting overlaps with gene annotations) and their differences between samples, alleles and splicing isoforms.
- Structure, novel splicing, gene fusions (from the alignments)
- Variants, RNA-editing (from the sequences)



Outline

RNA-seq Expression Analysis Workflow

- QA (*ShortRead*), alignment (*gmapR*)
- Import of alignments
- Counting of alignments over various features, including genes, transcripts and exons.
- Differential expression between samples (*DEseq*, *edgeR*), isoforms (*DEXseq*), or alleles
- Further down-stream analyses, including GSEA

RNA-seq Expression Analysis Workflow

- QA (*ShortRead*), alignment (*gmapR*)
- Import of alignments
- Counting of alignments over various features, including genes, transcripts and exons.
- Differential expression between samples (*DEseq*, *edgeR*), isoforms (*DEXseq*), or alleles
- Further down-stream analyses, including GSEA

Data from *leeBamViews*

- Four samples from a Yeast RNA-seq experiment
- Two wildtype, two RLP mutants
- Illumina 36bp single-end reads
- Alignments in *leeBamViews* for positions 800000 to 900000 on chromosome XIII
- Stored as BAM files

Disclaimer

For Demonstration Purposes Only

The alignments in *leeBamViews* are for a reference that is slightly different from (about two weeks older than) the UCSC sacCer2 build, for which we have gene annotations.

Outline

Representing SAM/BAM in R

Could store SAM alignments in a *GRanges*, but alignments are common enough for a formal representation.

The *GappedAlignments* Class

A *Vector* of *unpaired* read alignments with SAM-specific fields

- Load with `readGappedAlignments(file)`
- Access `cigar`, `qpos`, `start`, etc.
- Some utilities, like `coverage`.
- For overlap computations, convert to gapped ranges with `grglist`, get intronic gaps with `introns`.

Loading a BAM File

```
> bams <- dir(system.file("bam", package="leeBamViews"),
+             full = TRUE, pattern = "bam$")
> reads_ga <- readGappedAlignments(bams[[1]])
> head(reads_ga, 1)

GappedAlignments with 1 alignment and 0 elementMetadata cols:
     seqnames strand       cigar    qwidth
        <Rle>  <Rle> <character> <integer>
  [1]  Scchr13      -         36M        36
         start       end     width      ngap
     <integer> <integer> <integer> <integer>
  [1]    799975    800010        36         0
  ---
  seqlengths:
   Scchr01 Scchr02 Scchr03 ... Scchr16  Scmito
    230208  813178  316617 ...  948062   85779
```

Outline

Representing Read Alignments as Ranges

For finding overlaps between reads and genomic features, we need to represent read alignments as intervals on stranded sequences.

Sequence Name	Start	End	Strand	metadata...
chr10	3012936	3012959	+	
chr10	3012941	3012964	+	
chr10	3012944	3012967	+	

Representing Read Alignments as Ranges

For finding overlaps between reads and genomic features, we need to represent read alignments as intervals on stranded sequences.

Sequence Name	Start	End	Strand	metadata...
chr10	3012936	3012959	+	
chr10	3012941	3012964	+	
chr10	3012944	3012967	+	

All genomic data fits this basic format

Genomic Datasets in R

The *GRanges* Class

A *Vector* of genomic intervals, with metadata

- Constructor: `GRanges(seqnames, ranges, strand, ...)`
- `seqnames(x)`: sequence name
- `start(x)`, `end(x)`, `width(x)`: interval information
- `strand(x)`: strand (+/-/*)
- `values(x)`: a *DataFrame* of metadata columns, like score or gene
- `seqinfo(x)`: a *Seqinfo* with information about the sequences

Group multiple *GRanges* in a *GRangesList*, representing ranges with gaps.

Representing Alignments as Ranges with Gaps

- Read alignments may contain gaps, e.g., cross a splice junction.
- The possibly multiple ranges per read need to be grouped together.
- Use *GRangesList*, with one *GRanges* per read.

Creating a *GRangesList* from *GappedAlignments*

```
> reads_grl <- grglist(reads_ga)
```

Representing Transcript Models

- Many ways to represent transcript models as ranges:
  - Transcripts
  - Exons, introns
  - CDS, UTRs
- Reference annotations, so prefer persistent storage

Transcript Models in R

The *TxDb* Class

Reference to SQLite DB with transcript information

- `transcripts(x)`: whole transcript ranges
- `exons(x)`: exon ranges
- `cds(x)`: coding exon ranges
- `exonsBy(x)`, `cdsBy(x)`: *GRangesList* object, grouping by transcript or gene

For common organisms/models, *TxDb.** packages are available.

Obtaining the Yeast Transcripts

We get the coding exons grouped by gene, and individually. Using the coding exons is usually a good idea, since the UTRs can be problematic.

```
> library(TxDb.Scerevisiae.UCSC.sacCer2.sgdGene)
> sc2_txdb <- TxDb.Scerevisiae.UCSC.sacCer2.sgdGene
> sc2_cds <- cds(sc2_txdb)
> sc2_cds_gene <- cdsBy(sc2_txdb, by = "gene")
```

Exercise

Problem

Get the coding exons grouped by transcript, instead of gene.

Exercise

Problem

Get the coding exons grouped by transcript, instead of gene.

Solution

```
> sc2_cds_tx <- cdsBy(sc2_txdb, by = "tx")
```

Getting Gene IDs for the Transcripts

For interpretation, it is useful to have the gene ID for each transcript:

```
> sc2_tx <- transcripts(sc2_txdb,
+                       columns = c("tx_id", "gene_id"))
> tx_id_match <- match(names(sc2_cds_tx),
+                      values(sc2_tx)$tx_id)
> gene_id <- values(sc2_tx)$gene_id[tx_id_match]
```

Theoretically possible for transcript to belong to multiple genes, so `gene_id` is a *CharacterList*. Typically there is one gene per transcript, so we drop it to a *character* vector:

```
> all(elementLengths(gene_id) <= 1)

[1] TRUE

> values(sc2_tx)$gene_id <- drop(gene_id)
```

Count Reads in Exons, by Gene

Reconcile Chromosome Names

```
> sc2_cds_gene <- keepSeqlevels(sc2_cds_gene, "chrXIII")
> sc2_cds_gene <- renameSeqlevels(sc2_cds_gene,
+                                 c(chrXIII = "Scchr13"))
```

Call `countOverlaps`

```
> sc2_counts <- countOverlaps(sc2_cds_gene, reads_grl,
+                             ignore.strand = TRUE)
```

We pass `ignore.strand = TRUE`, because the strand is the strand of alignment, not transcription.

Some Limitations of our Simple Approach

We have the count of reads overlapping with the coding regions of every gene, but:

- It only works for a single BAM file (need a convenience function),
- A read is counted for every feature it overlaps, which works reasonably well for genes but does not help with distinguishing transcript isoforms, and
- There is no consideration of splicing, i.e., checking for compatibility with known transcript structures.

Processing Multiple BAM Files

```
> sc2_counts <- seqapply(bams, function(bam) {
+   countOverlaps(sc2_cds_gene,
+                 grglist(readGappedAlignments(bam)),
+                 ignore.strand = TRUE)
+ })
```

The results land in a simple list of integer vectors; we need something more formal.

Representing Rectangular Counts

The *SummarizedExperiment* class

The *SummarizedExperiment* class is the *eSet* analog in the *IRanges* framework. It is a rectangular, self-contained data structure with the following components:

`assays` A list of matrices, each feature X sample, containing counts or other measurements

`rowData` A *GRanges* or *GRangesList* with annotations on the features; *each feature must then be a genomic range (possibly with gaps).*

`colData` A *DataFrame* with phenotypic data on the samples

`exptData` A list of metadata objects describing the experiment

SummarizedExperiment API

- `SummarizedExperiment(assays, rowData, colData, exptData, ...)`: construct an instance with the given components
- `assay(x, i)`: Get the `i`th assay matrix
- `rowData(x)`, `colData(x)`, `exptData(x)`: get the corresponding metadata component
- `x[i, j]`: Rectangular subset by feature and sample.
- `x$name`: Get the named column in `colData`

Storing the Counts in a *SummarizedExperiment*

```
> assay <- list(counts = do.call(cbind, as.list(sc2_counts)))
> rowData <- sc2_cds_gene
> samples <- gsub("_.*", "", basename(bams))
> colData <- DataFrame(genotype = gsub(".$", "", samples),
+                      lane = gsub(".*(.)$", "\\1", samples),
+                      row.names = samples)
> se <- SummarizedExperiment(assay, rowData, colData)
```

Calculating RPKMs

```
> sc2_count_mat <- assay(se)
> k <- sum(width(rowData(se)))
> m <- countBam(BamFileList(bams))[,"records"] / 1e6
> rpkm <- sc2_count_mat / k /
+   rep(m, each = nrow(sc2_count_mat))
```

Potential Shortcut: `summarizeOverlaps`

The `summarizeOverlaps` function:

- Counts overlaps between BAM files and a set of genomic features,
- Always discards reads that map to multiple features, with several different algorithms for determining uniqueness, and
- Returns the result as a *SummarizedExperiment*.

Potential Shortcut: `summarizeOverlaps`

The `summarizeOverlaps` function:

- Counts overlaps between BAM files and a set of genomic features,
- Always discards reads that map to multiple features, with several different algorithms for determining uniqueness, and
- Returns the result as a *SummarizedExperiment*.

Potential Shortcut: `summarizeOverlaps`

The `summarizeOverlaps` function:

- Counts overlaps between BAM files and a set of genomic features,
- Always discards reads that map to multiple features, with several different algorithms for determining uniqueness, and
- Returns the result as a *SummarizedExperiment*.

Filters do not consider splicing, i.e., splicing compatibility and junction hits.

Feeding into DE Algorithms

DESeq

```
> library(DESeq)
> rowDataDf <- as.data.frame(unlist(range(rowData(se))))
> cond <- as.data.frame(colData(se))
> featureData <- AnnotatedDataFrame(rowDataDf)
> deseq <- newCountDataSet(assay(se), cond = cond,
+                          featureData = featureData)
```

edgeR

```
> library(edgeR)
> edger <- DGEList(assay(se), group = colData$genotype,
+                  genes = rowDataDf)
```

Find Transcript Hits for Each Read

Using `findOverlaps`

```
> hits <- findOverlaps(reads_grl, sc2_cds_gene,
+                      ignore.strand = TRUE)
> values(reads_grl)$hits <- as(hits, "List")
> head(table(elementLengths(values(reads_grl)$hits)))

    0     1     2
 1043 32558   948
```

Outline

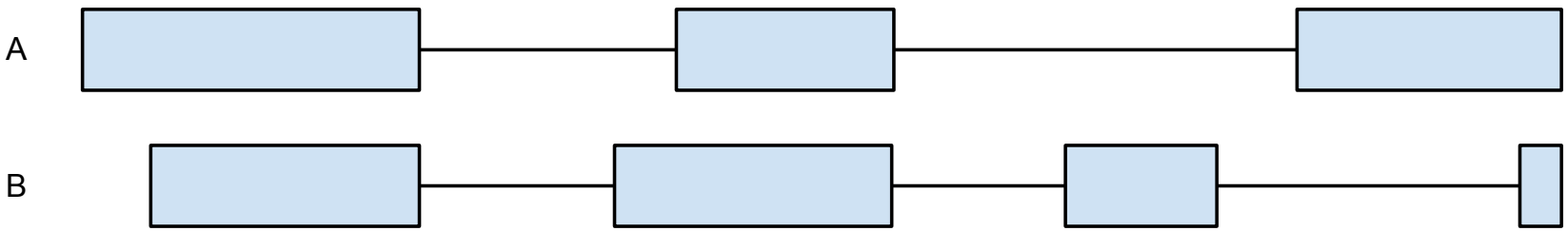
Outline

Focus: *Annotated* Transcript Isoforms

We are primarily interested in the expression of annotated isoforms and deviations from known structures.

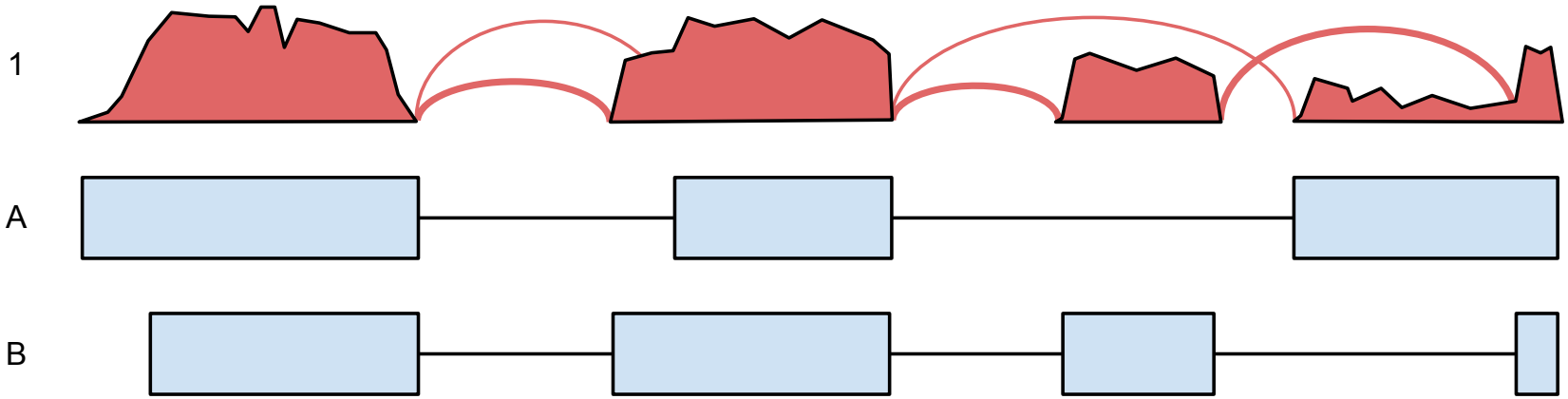
Questions about Isoform Expression

- 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?



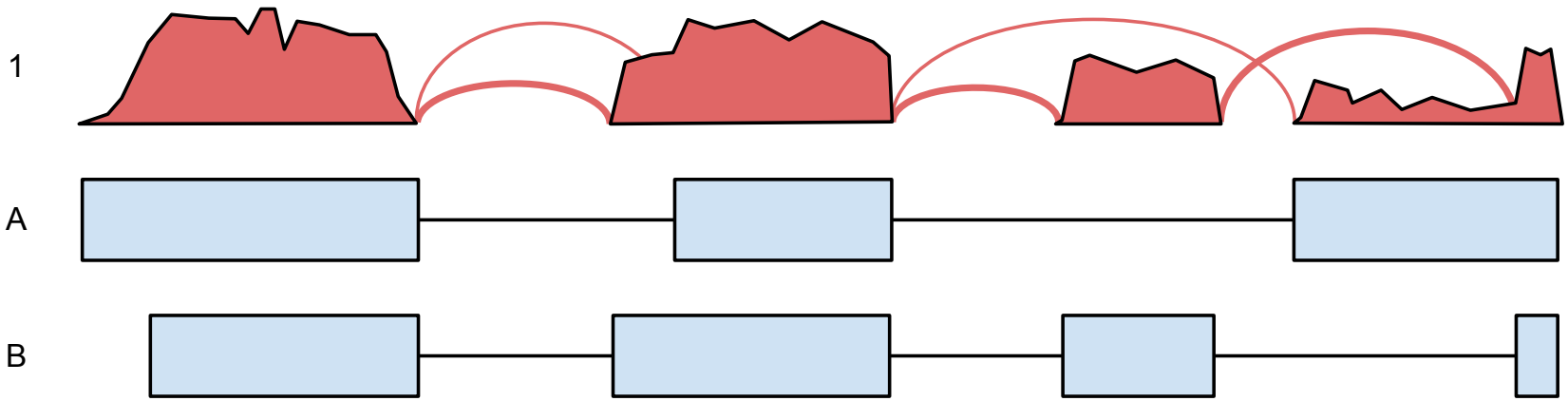
Questions about Isoform Expression

- 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

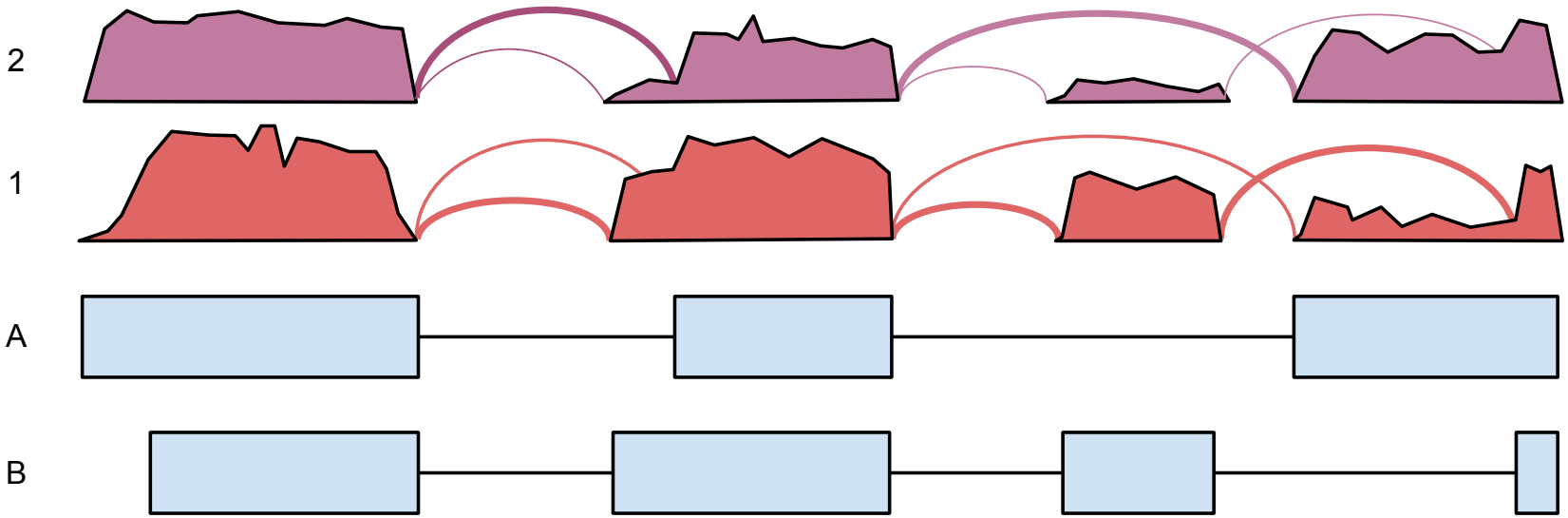
- Is an isoform expressed?
- What is the expression level of a particular isoform, and how can we compare it to that of other isoforms?
- Is the balance in isoform expression different across samples?



	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

Fragment Length Matters

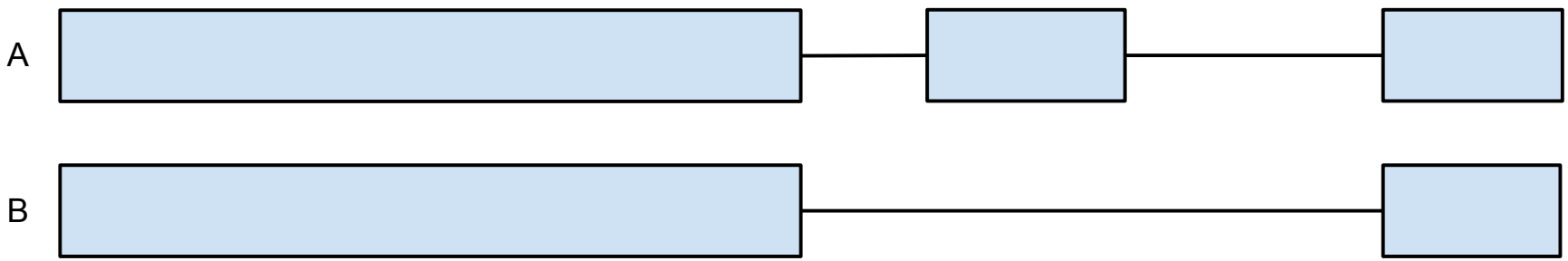
Note

Differences in fragment length between regions and samples can bias counts in multiple ways, including:

- Coverage at ends of transcript is reduced; the affected region grows with fragment length. The bias is most significant for short genes and those with variable TSS/TSE.
- Any counts of transcript sub-regions (exons) or splice junctions are further affected, because the probability of overlap depends on the fragment length.

Fragment Length Affects Capture of Cassette Exon

w/ Robert Gentleman

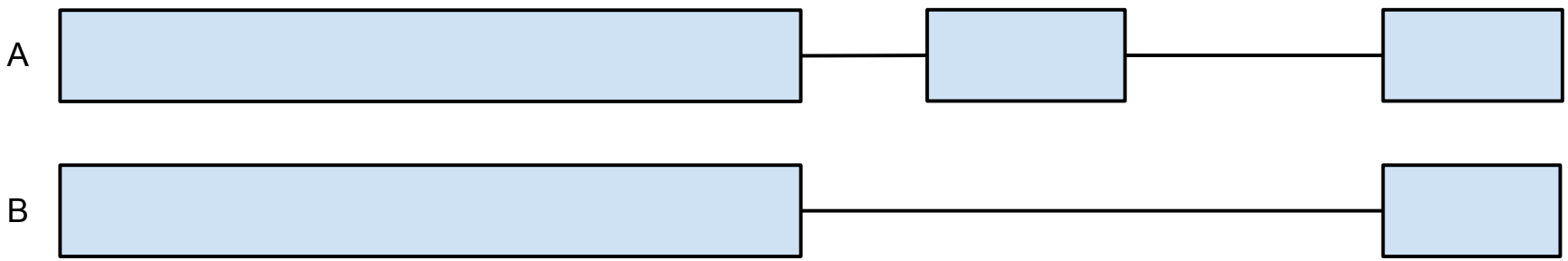


Fragment Length Affects Capture of Cassette Exon

w/ Robert Gentleman

Capture probability P for isoform A is proportional to FL + EL and is capped at 2RL + 2EL. Isoform B follows the same rule, where EL := 0.

FL: Fragment Length
EL: Cassette Exon Length
RL: Read Length

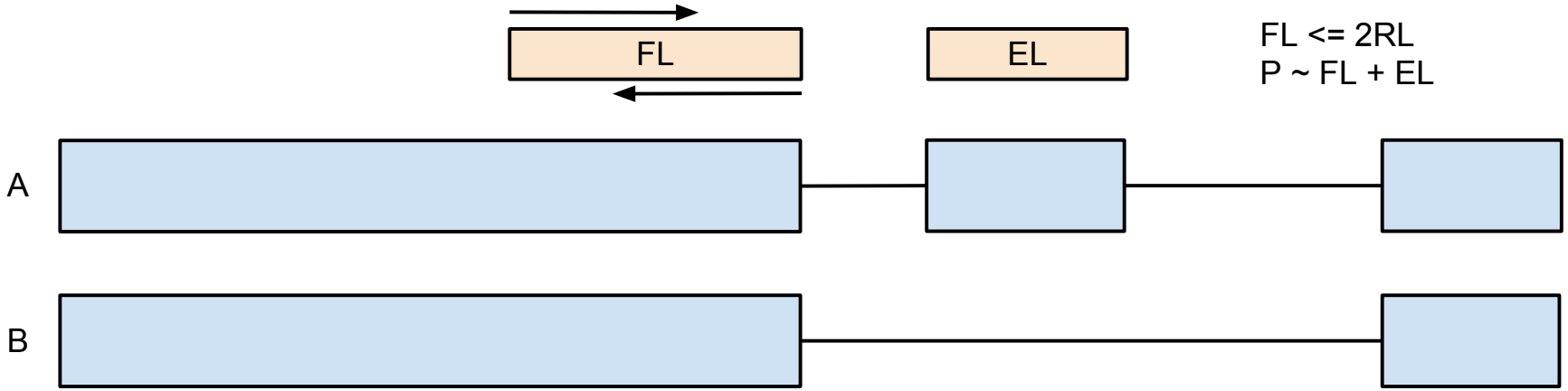


Fragment Length Affects Capture of Cassette Exon

w/ Robert Gentleman

Capture probability P for isoform A is proportional to FL + EL and is capped at 2RL + 2EL. Isoform B follows the same rule, where EL := 0.

FL: Fragment Length
EL: Cassette Exon Length
RL: Read Length

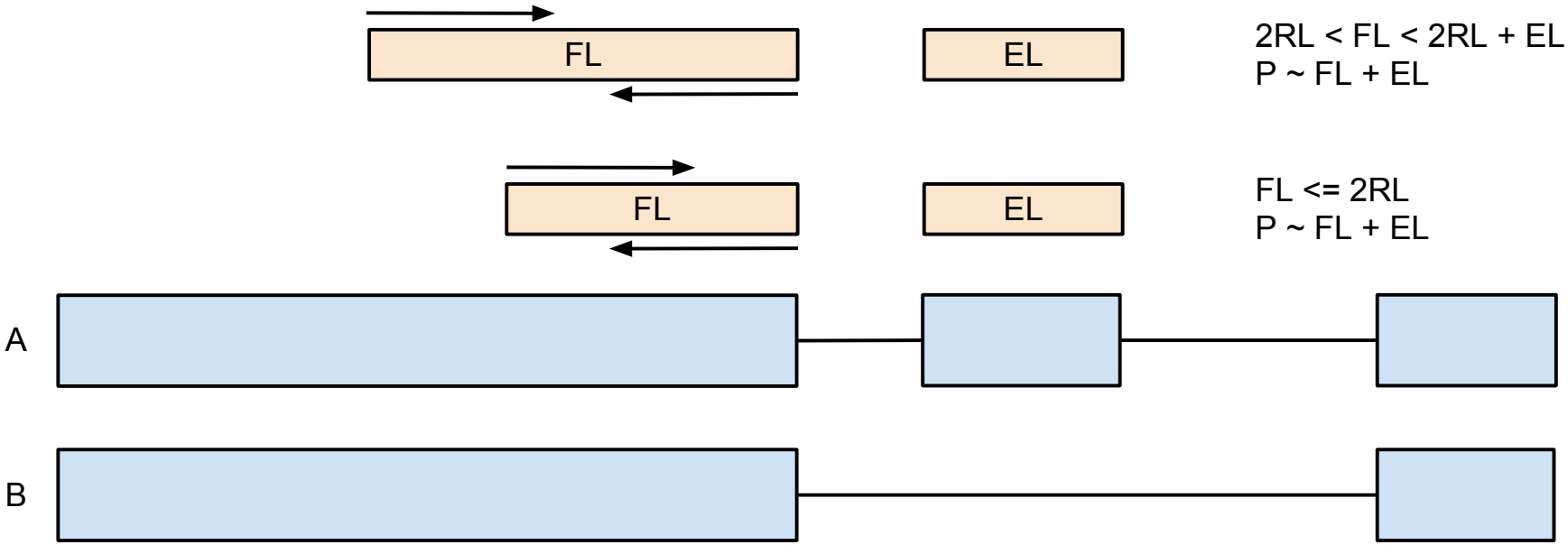


Fragment Length Affects Capture of Cassette Exon

w/ Robert Gentleman

Capture probability P for isoform A is proportional to FL + EL and is capped at 2RL + 2EL. Isoform B follows the same rule, where EL := 0.

FL: Fragment Length
EL: Cassette Exon Length
RL: Read Length

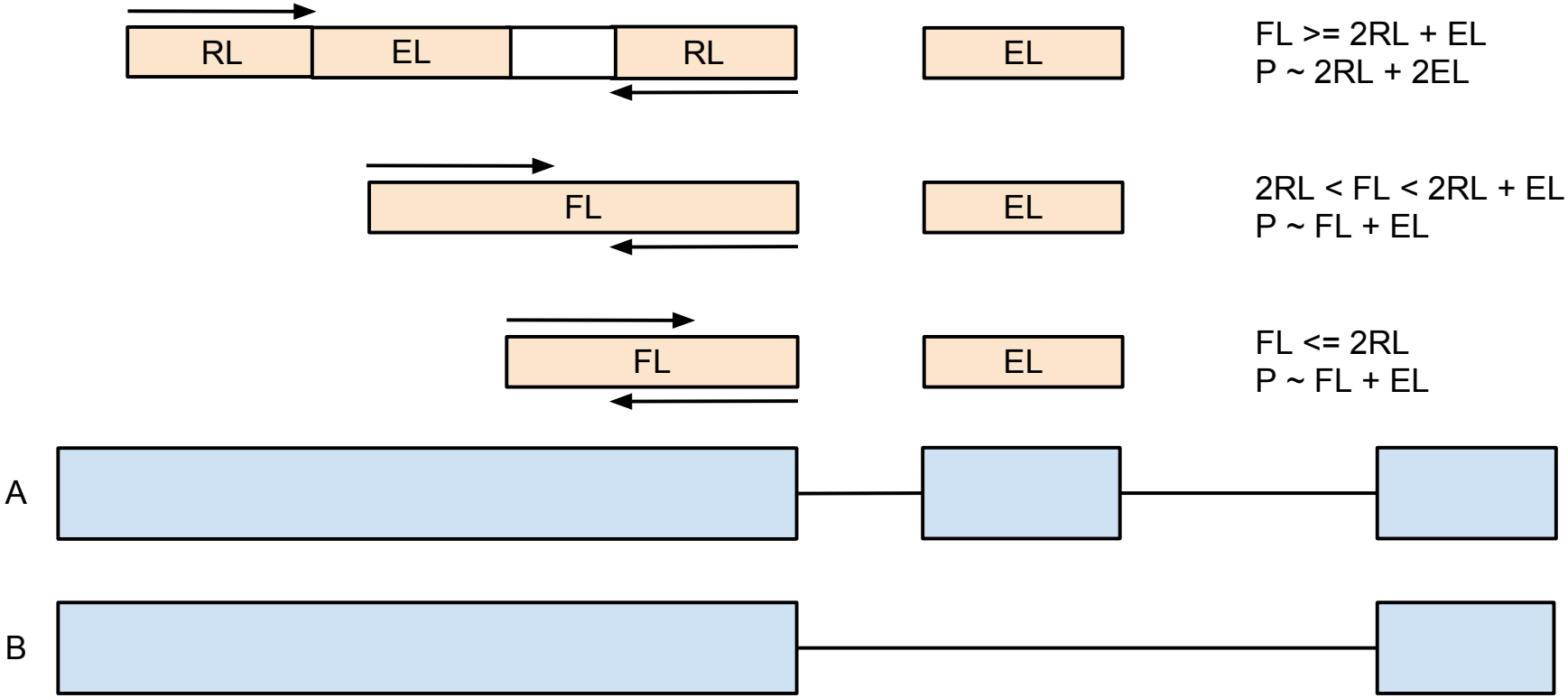


Fragment Length Affects Capture of Cassette Exon

w/ Robert Gentleman

Capture probability P for isoform A is proportional to FL + EL and is capped at 2RL + 2EL. Isoform B follows the same rule, where EL := 0.

FL: Fragment Length
EL: Cassette Exon Length
RL: Read Length



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

Getting the Annotations for ALDOA

Finding the ALDOA Region

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

Grouping the Exons by Transcript

```
> aldoa_range <- range(aldoa_exons)
> aldoa_exons_tx <- multisplit(aldoa_exons,
+                              values(aldoa_exons)$tx_id)
```

Getting the Annotations for ALDOA

Getting the Coding Extents

```
> aldoa_cds <- cds(txdb, vals = list(gene_id = aldoa_eg),
+                  columns = c("gene_id", "tx_id"))
> aldoa_cds_tx <- multisplit(aldoa_cds,
+                            values(aldoa_cds)$tx_id)
> aldoa_exons_tx <- aldoa_exons_tx[names(aldoa_cds_tx)]
```

Selecting Only the CCDS Models

Just take my word for it:

```
> aldoa_exons_tx <- aldoa_exons_tx[c(1, 3, 4, 6)]
> aldoa_cds_tx <- aldoa_cds_tx[c(1, 3, 4, 6)]
```

Reading the Alignments

Finding the BAMs

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

Load the normal BAM

```
> bam <- bamFiles$normal

> param <- ScanBamParam(tag = "XS", what = "isize",
+                       which = aldoa_range)
> ga <- readGappedAlignmentPairs(path(bam), param = param)
```

We request the *XS* tag, the strand as determined by the aligner from the splice directions, as well as the `isize` (TLEN) field from the BAM file, which is the fragment length.

Representing Paired-end Alignments

The *GappedAlignmentPairs* Class

The *GappedAlignmentPairs* class is composed of two parallel *GappedAlignments* objects: one for the first read of the pair, the other for the last. It provides the following API:

- `first(x)`, `last(x)`: gets the *GappedAlignments* for the first or last reads of every pair
- `left(x)`, `right(x)`: gets the *GappedAlignments* for the left or right member of every pair, in terms of genomic position
- `grglist(x)`: converts to a *GRangesList*, with gaps for the introns
- `introns(x)`: gets the intronic gaps as a *GRangesList*

Getting the Fragment Length

- The algorithm for computing isize/TLEN differs by aligner, and some aligners do not populate the field.
- GSNAP does exactly what we want: it takes the extent of the paired-end alignment, excluding clipped regions and intronic gaps (N in CIGAR). So we can simply use the absolute value of the `isize` field (the sign depends on the relative read positions):

```
> fraglen <- abs(values(first(ga))$isize)
```

Any introns in the inter-read gap are not accounted for (because we do not know the original transcript).

Exercise: Computing Fragment Length

Problem

Use *GappedAlignmentPairs* to compute the fragment length, without worrying about the introns.

Exercise: Computing Fragment Length

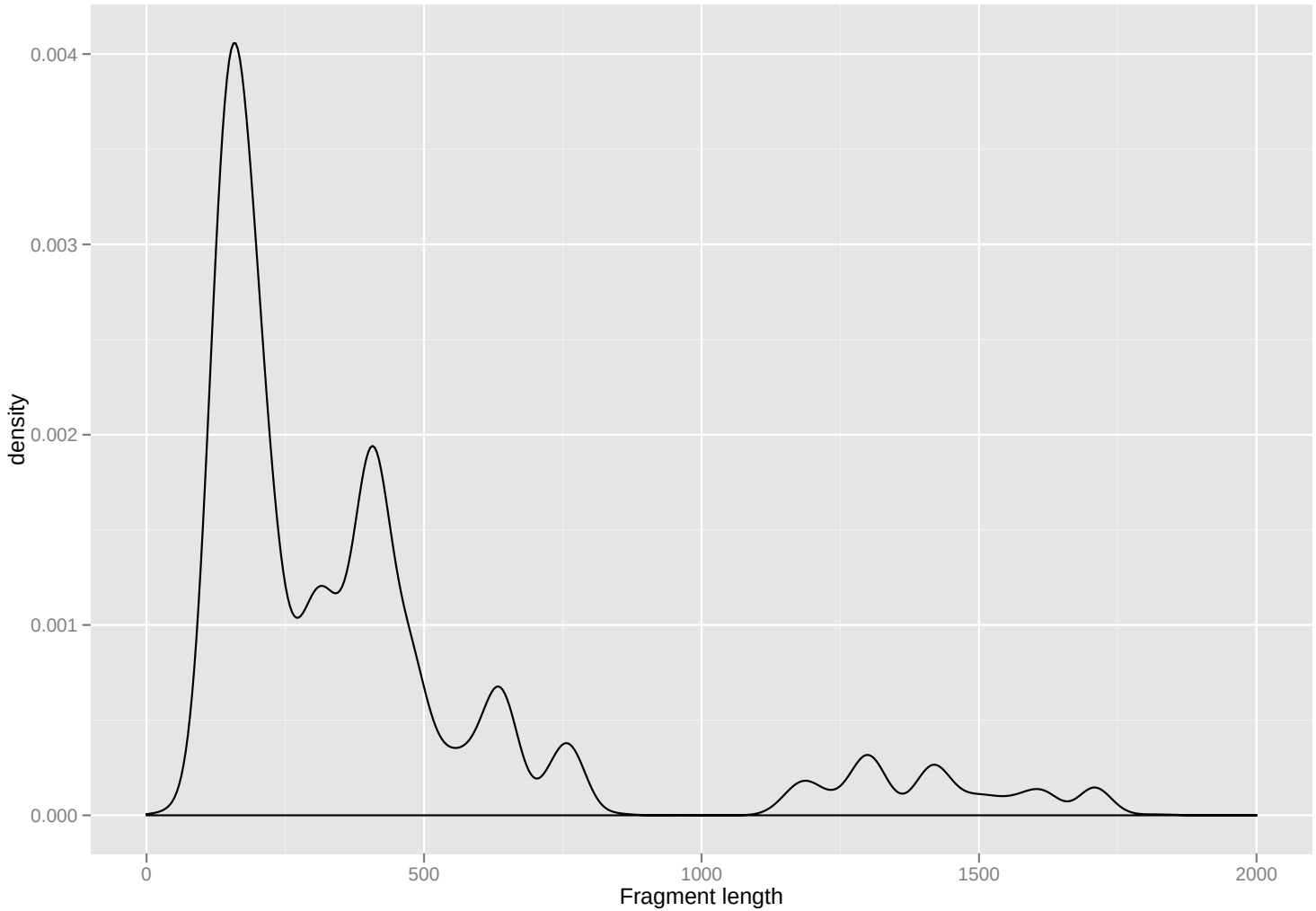
Problem

Use *GappedAlignmentPairs* to compute the fragment length, without worrying about the introns.

Solution

```
> fraglen_simple <- end(right(ga)) - start(left(ga)) + 1L
```

Density of Fragment Length



Outline

Overview of Algorithm

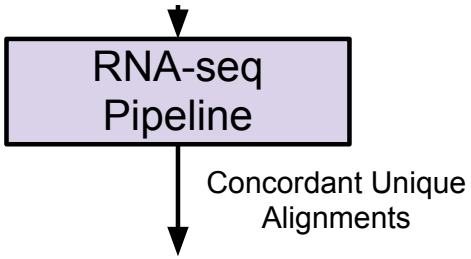
Steps

1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and annotated transcripts with any exonic overlap
3. Split the alignments into two bins: *compatible* with annotated splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform

Overview of Algorithm

Steps

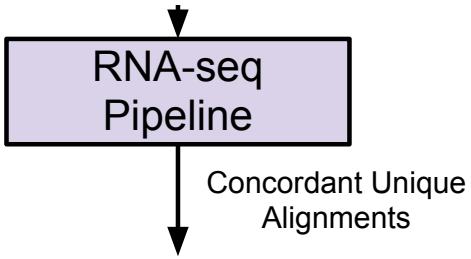
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and annotated transcripts with any exonic overlap
3. Split the alignments into two bins: *compatible* with annotated splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform



Overview of Algorithm

Steps

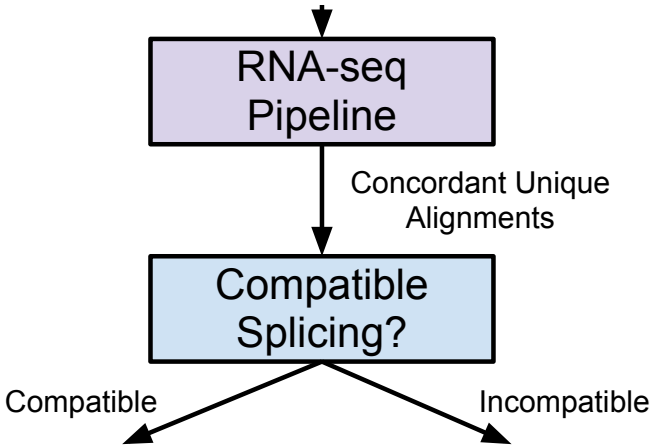
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and annotated transcripts with any exonic overlap
3. Split the alignments into two bins: *compatible* with annotated splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform



Overview of Algorithm

Steps

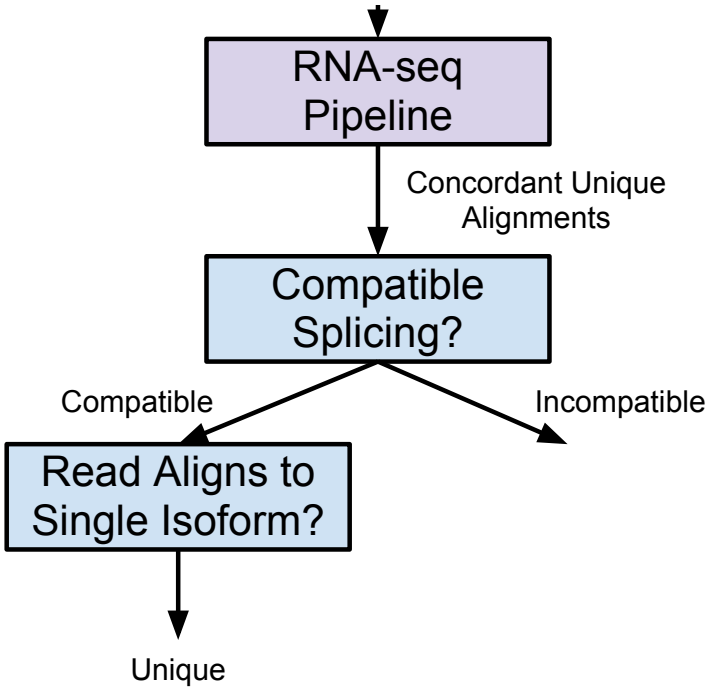
1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and annotated transcripts with any exonic overlap
3. Split the alignments into two bins: *compatible* with annotated splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform



Overview of Algorithm

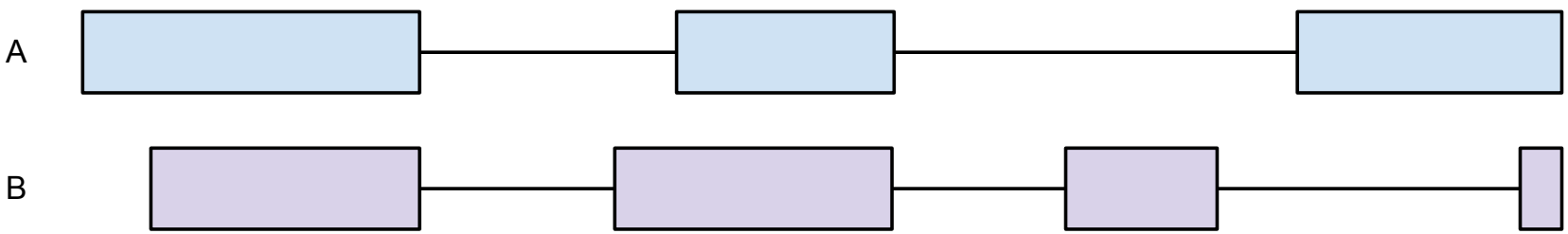
Steps

1. Run RNA-seq pipeline, and load *concordant unique* alignments
2. Find reads and annotated transcripts with any exonic overlap
3. Split the alignments into two bins: *compatible* with annotated splicing, and *incompatible*
4. For the *compatible*, find reads that map *uniquely* to an isoform



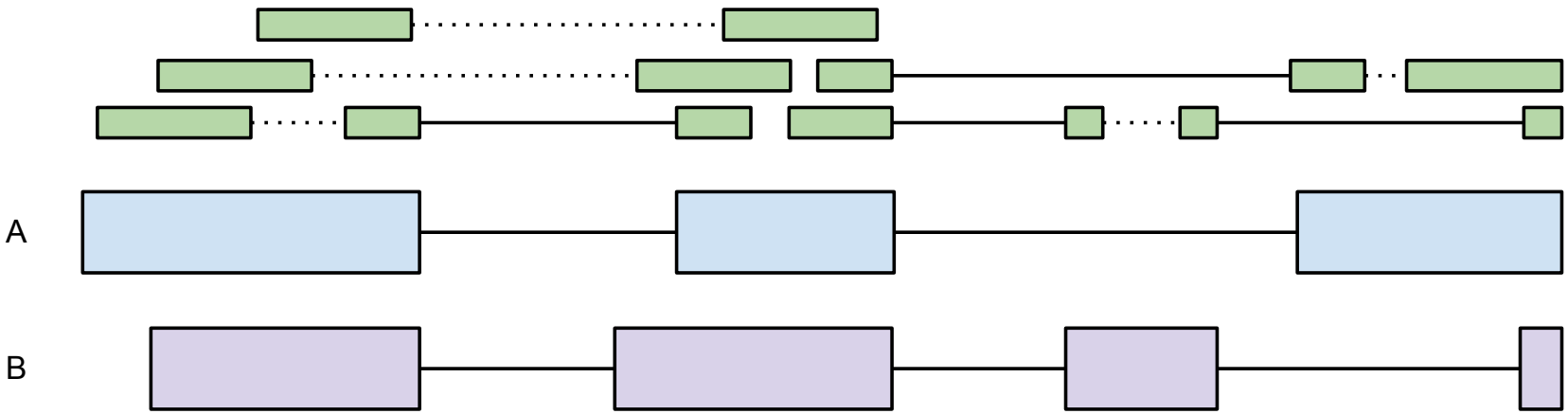
Compatibility and Uniqueness

- 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



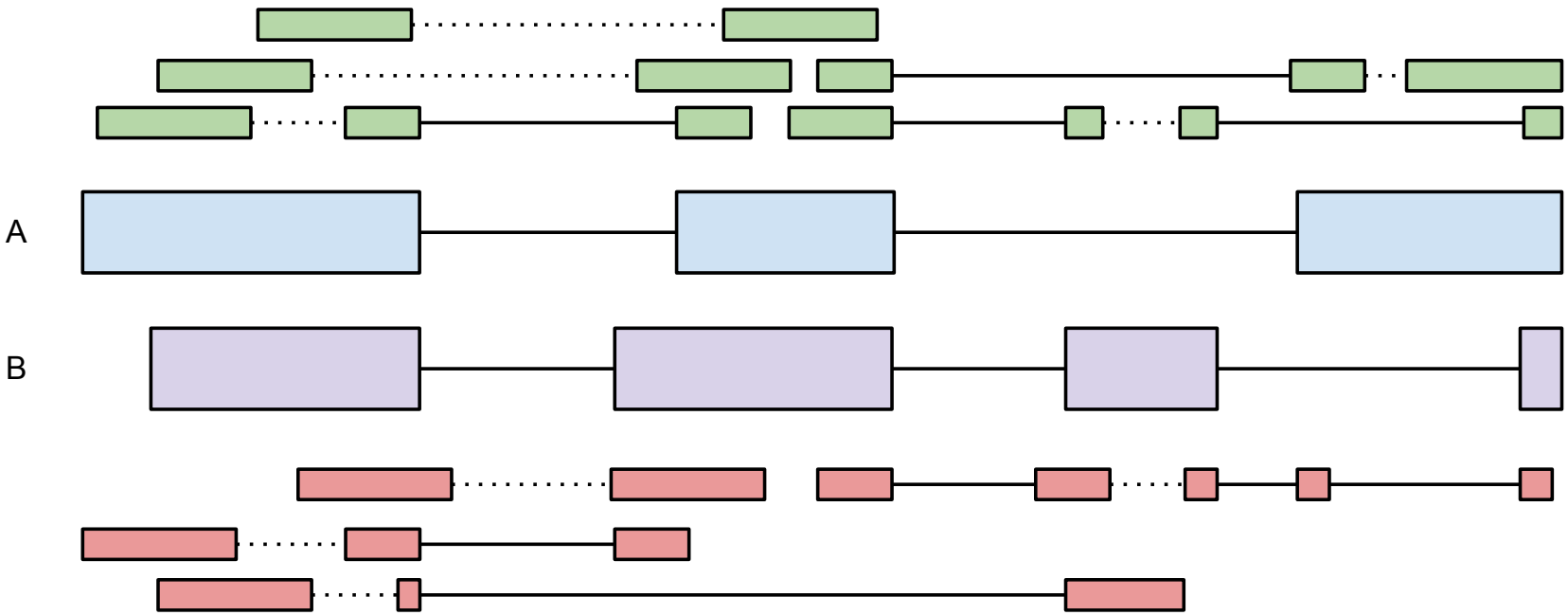
Compatibility and Uniqueness

- 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



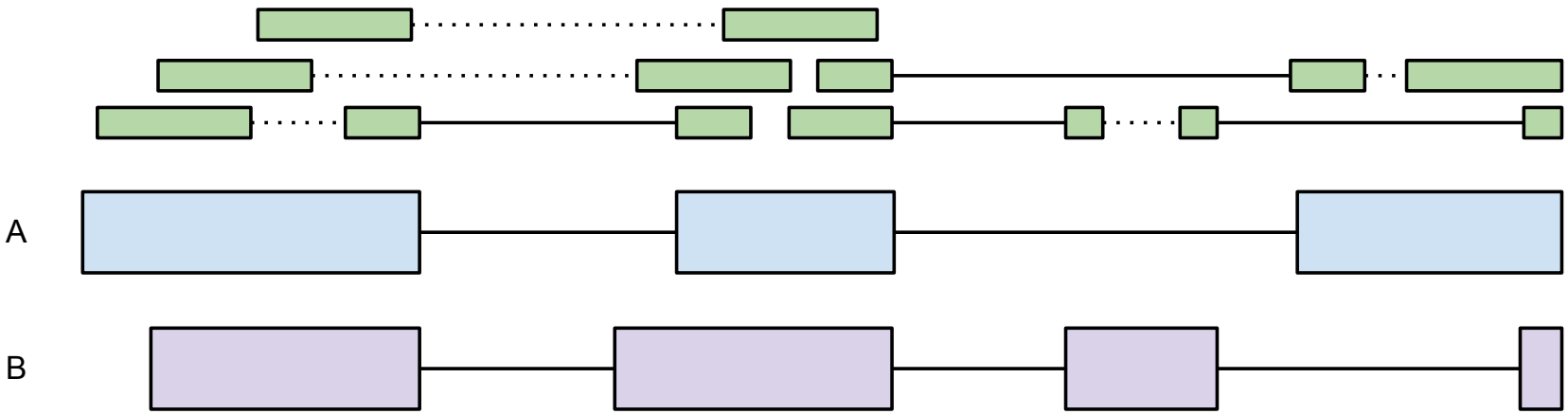
Compatibility and Uniqueness

- 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



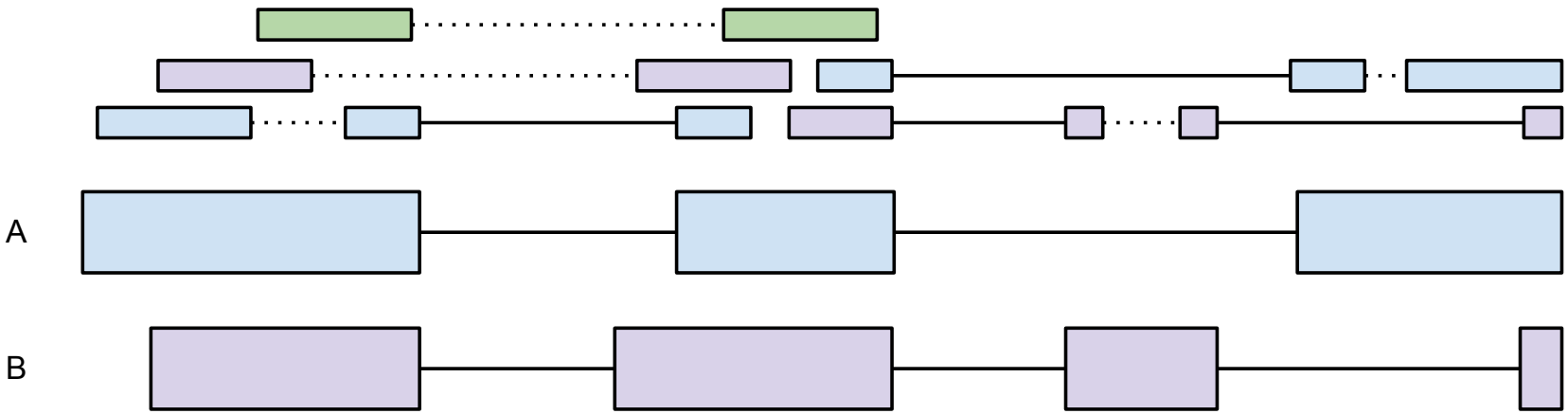
Compatibility and Uniqueness

- 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



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



Finding Overlaps Using Splicing

The `findSpliceOverlaps` function is an experimental, work-in-progress that annotates the return value of the basic `findOverlaps` including the following logical variables:

compatible The read (pair) exactly matches the splicing pattern of the transcript, over the extent of the alignment. If this is ever `TRUE` for a read, all of its incompatible hits are discarded.

unique The hit is compatible and is the only one for this read, i.e., it maps only to a single transcript.

coding The hit is at least partially to a coding region.

Calling `findSpliceOverlaps`

```
> hits <- findSpliceOverlaps(bamFiles$normal, aldoa_exons_tx,
+                            cds = aldoa_cds_tx,
+                            pairedEnd = TRUE)
```

Counting the Hits

Assume we are interested in the total count, the compatible count and the unique (compatible) count, restricting to coding regions:

```
> countSpliceHits <- function(hits) {
+   coding_hits <- hits[values(hits)$coding]
+   compatible_hits <-
+     coding_hits[values(coding_hits)$compatible]
+   unique_hits <- coding_hits[values(coding_hits)$unique]
+   cbind(coding = countSubjectHits(coding_hits),
+         compatible = countSubjectHits(compatible_hits),
+         unique = countSubjectHits(unique_hits))
+ }
> counts <- countSpliceHits(hits)
```

Exercise: Summarizing Over Multiple Samples

Challenge

Write a function called `summarizeSpliceOverlaps` that takes a *BamFileList* (`bamFiles`), computes the unique counts and returns a *SummarizedExperiment*. **Bonus**: compute and store the total and compatible counts, as well.

Exercise: Summarizing Over Multiple Samples

Solution

```
> summarizeSpliceOverlaps <- function(bams, tx, cds) {
+   counts <- lapply(bams, function(bam) {
+     hits <- findSpliceOverlaps(bam, tx, cds = cds)
+     counts <- countSpliceHits(hits)
+     split(counts, colnames(counts)[col(counts)])
+   })
+   assays <- do.call(mapply,
+                     c(cbind, counts, SIMPLIFY = FALSE))
+   colData <- DataFrame(tumorStatus = c("tumor", "normal"))
+   rownames(colData) <- colData$tumorStatus
+   SummarizedExperiment(assays, tx, colData)
+ }
> se <- summarizeSpliceOverlaps(bamFiles, aldoa_exons_tx,
+                               aldoa_cds_tx)
```

Outline

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
  4.337407
```

Plotting the Isoform Coverage

1. Load alignments, merging XS (strand of transcription) within pairs.
2. Convert to ranges, resolving XS to strand value.
3. Filter for reads that map to a single transcript.

Loading the Alignments

Propagation of XS From Reads to Pairs

```
> propagateXS <- function(ga) {
+   first_xs <- values(first(ga))$XS
+   last_xs <- values(last(ga))$XS
+   xs <- first_xs
+   xs[is.na(xs)] <- last_xs[is.na(xs)]
+   values(ga)$XS <- xs
+   ga
+ }
```

Loading the Alignments

Load Alignments

```
> loadAlignments <- function(bam) {
+   ga <- readGappedAlignmentPairs(path(bam),
+                                  param = param)
+   propagateXS(ga)
+ }
> ga_normal <- loadAlignments(bamFiles$normal)
> ga_tumor <- loadAlignments(bamFiles$tumor)
```

Converting Reads to Ranges

Resolving Strand from XS

```
> resolveStrandFromXS <- function(reads,
+                                 xs = values(reads)$XS)
+ {
+   strand <- ifelse(!is.na(xs), xs, "*")
+   strand(reads) <- relist(Rle(strand, elementLengths(reads)),
+                           reads)
+   reads
+ }
```

Converting Reads to Ranges

```
> getReadRanges <- function(ga) {
+   resolveStrandFromXS(grglist(ga))
+ }
> reads_normal <- getReadRanges(ga_normal)
> reads_tumor <- getReadRanges(ga_tumor)
```

Filter For Uniquely Mapped Reads

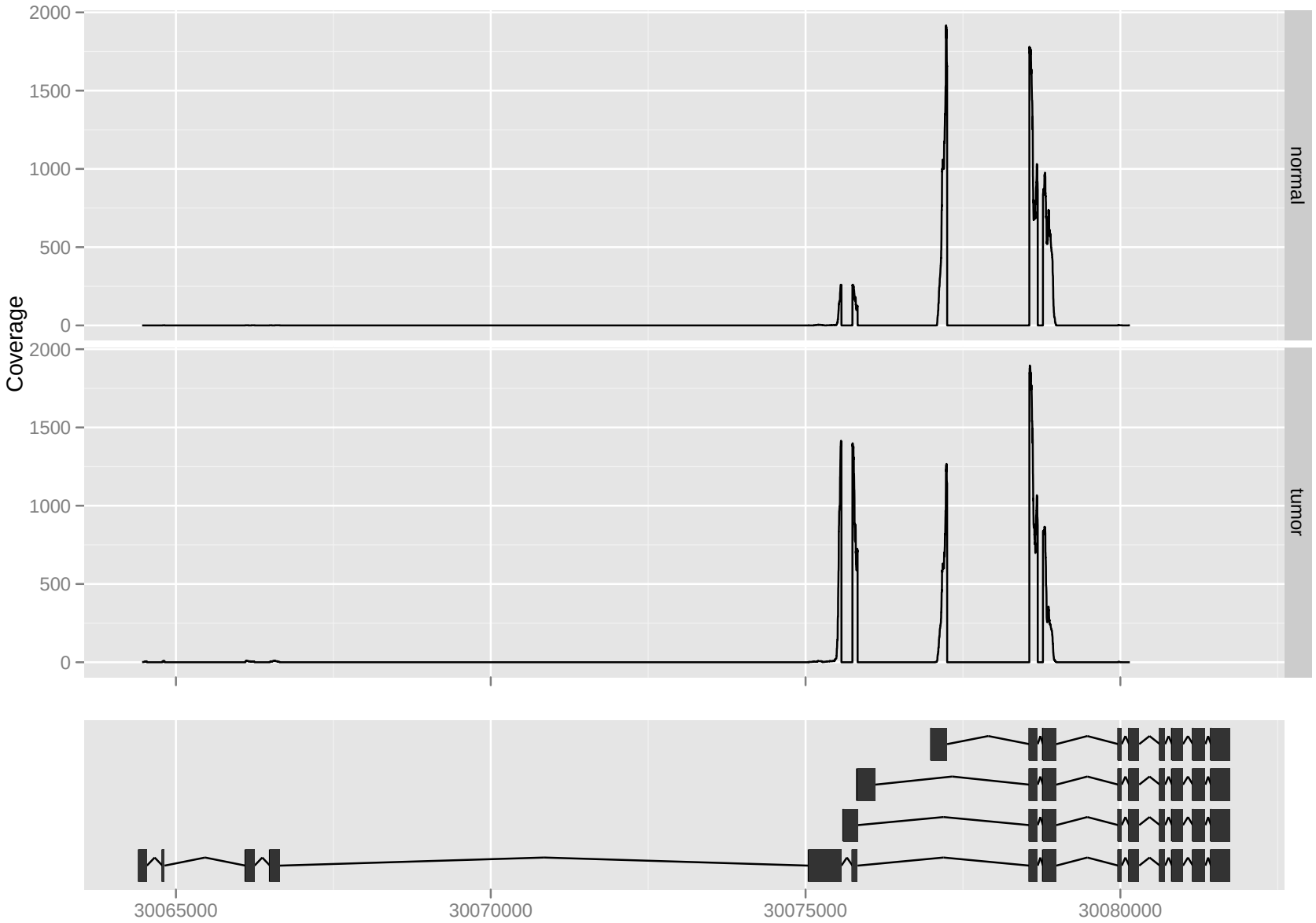
```
> getUniques <- function(hits) {
+   unique(queryHits(hits[values(hits)$unique &
+                         subjectHits(hits) %in% c(1, 4)]))
+ }
> unique_normal <- getUniques(hits_normal)
> unique_tumor <- getUniques(hits_tumor)
> unique_reads_normal <- reads_normal[unique_normal]
> unique_reads_tumor <- reads_tumor[unique_tumor]
> unique_reads <-
+   mstack(normal = unlist(unique_reads_normal),
+          tumor = unlist(unique_reads_tumor))
```

Plotting the Isoform Coverage

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

```
> read_track <- autoplot(unique_reads, stat = "coverage",
+                        facets = name ~ .)
> tx_track <- autoplot(aldoa_exons_tx, geom = "alignment",
+                      ylab = "")
> cov_tracks <- tracks(read_track, tx_track,
+                      heights = c(3, 1))
> cov_tracks
```

Coverage Plot

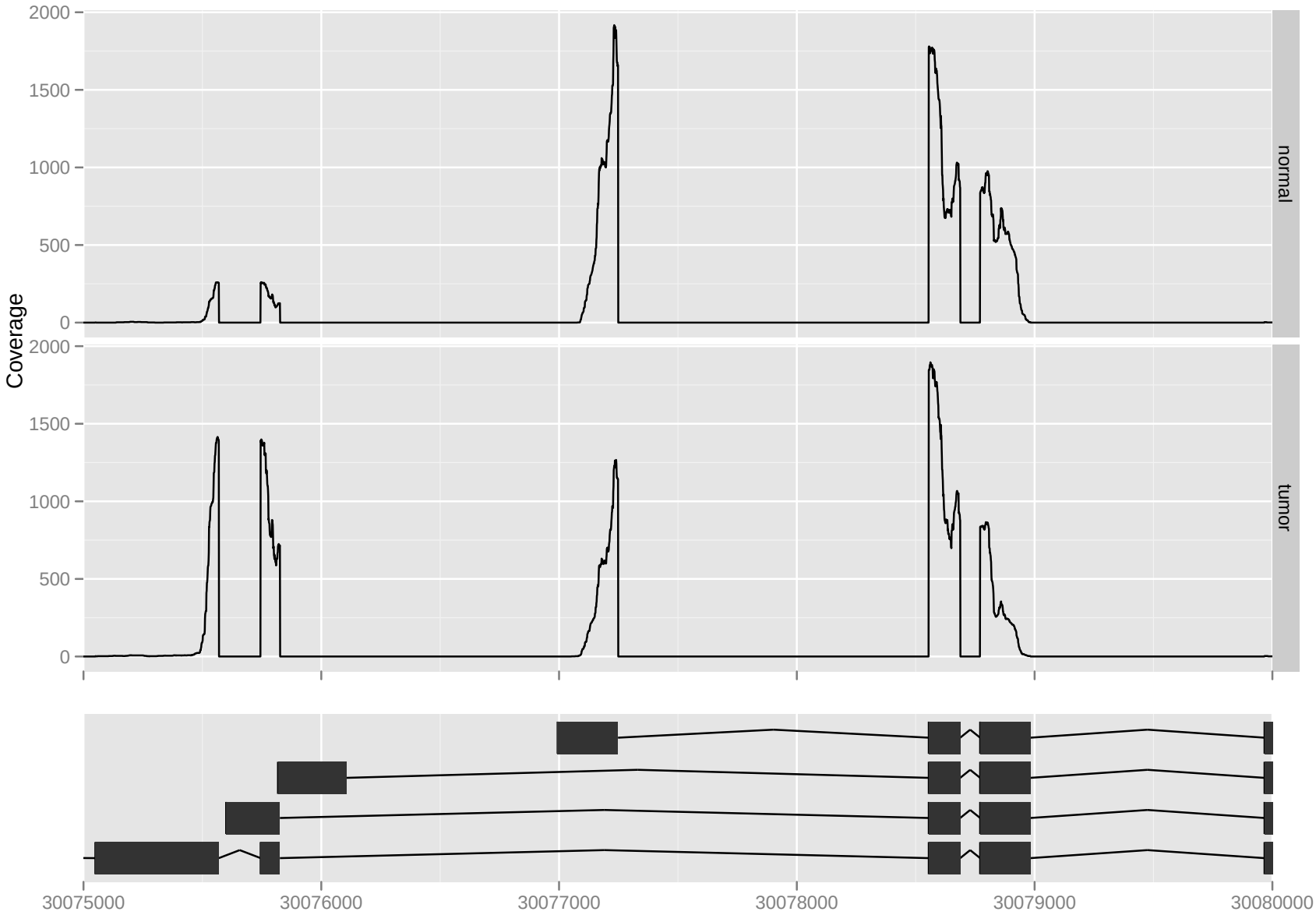


Zooming into the Plot

The `xlim` argument restricts the genomic interval:

```
> xlim(cov_tracks) <- c(30075000, 30080000)
> cov_tracks
```

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.

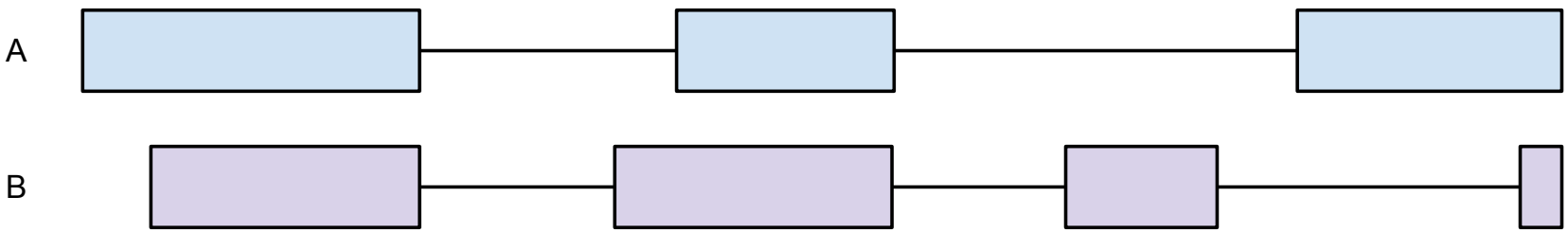
Solution

```
> cov_chr16 <- coverage(unique_reads)$chr16
> roi <- range(ranges(slice(cov_chr16, 1000)))
> roi <- roi + 500
> xlim(cov_tracks) <- roi
> cov_tracks
```

Outline

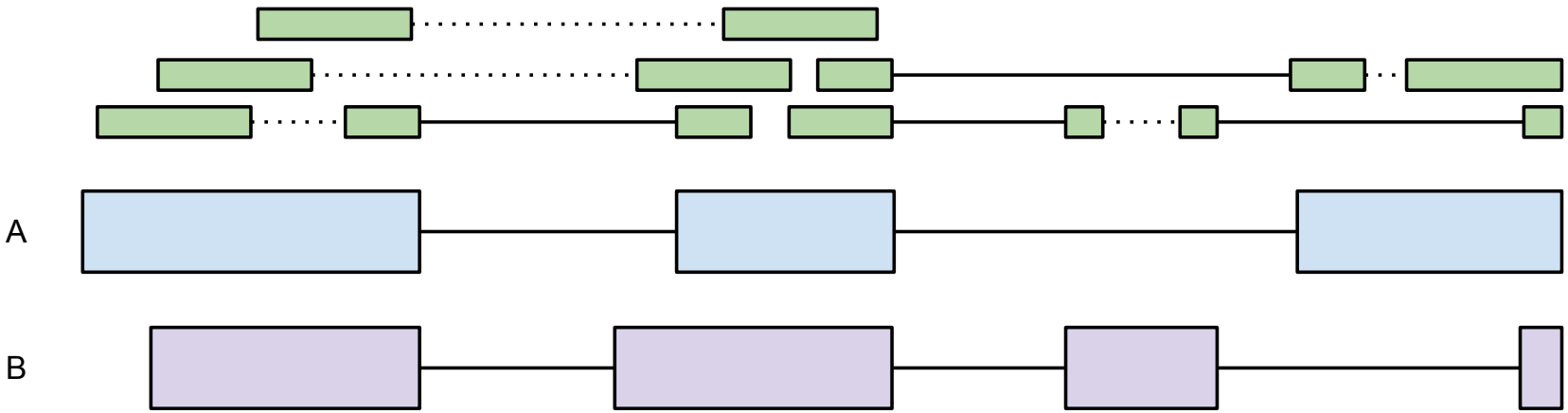
Questions about Isoform Structure

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



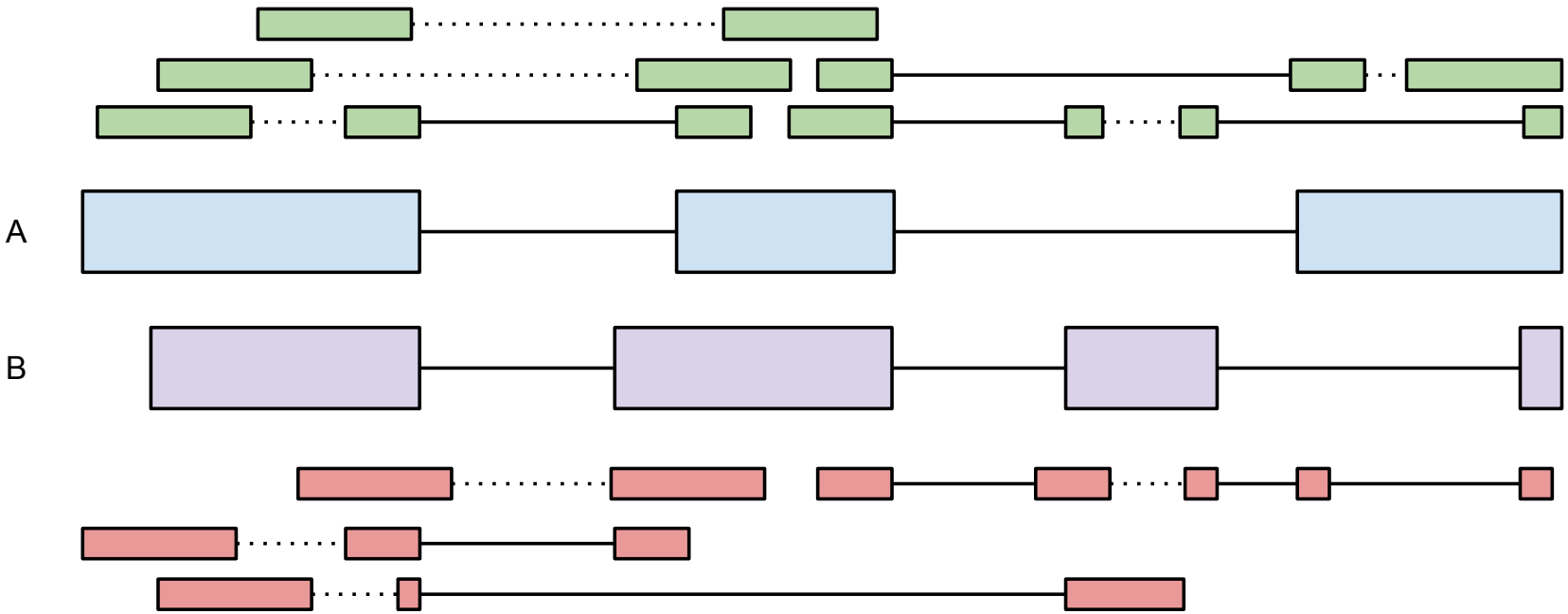
Questions about Isoform Structure

- 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?



Finding the Splices in the Alignments

We use the `introns` function to find the intronic gaps in the read alignments (N runs in the CIGAR):

```
> splices <- resolveStrandFromXS(introns(ga_normal),
+                                values(ga_normal)$XS)
```

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 <- psetdiff(range(aldoa_exons_tx), aldoa_exons_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))
```

Plotting the Isoform Coverage

Setup

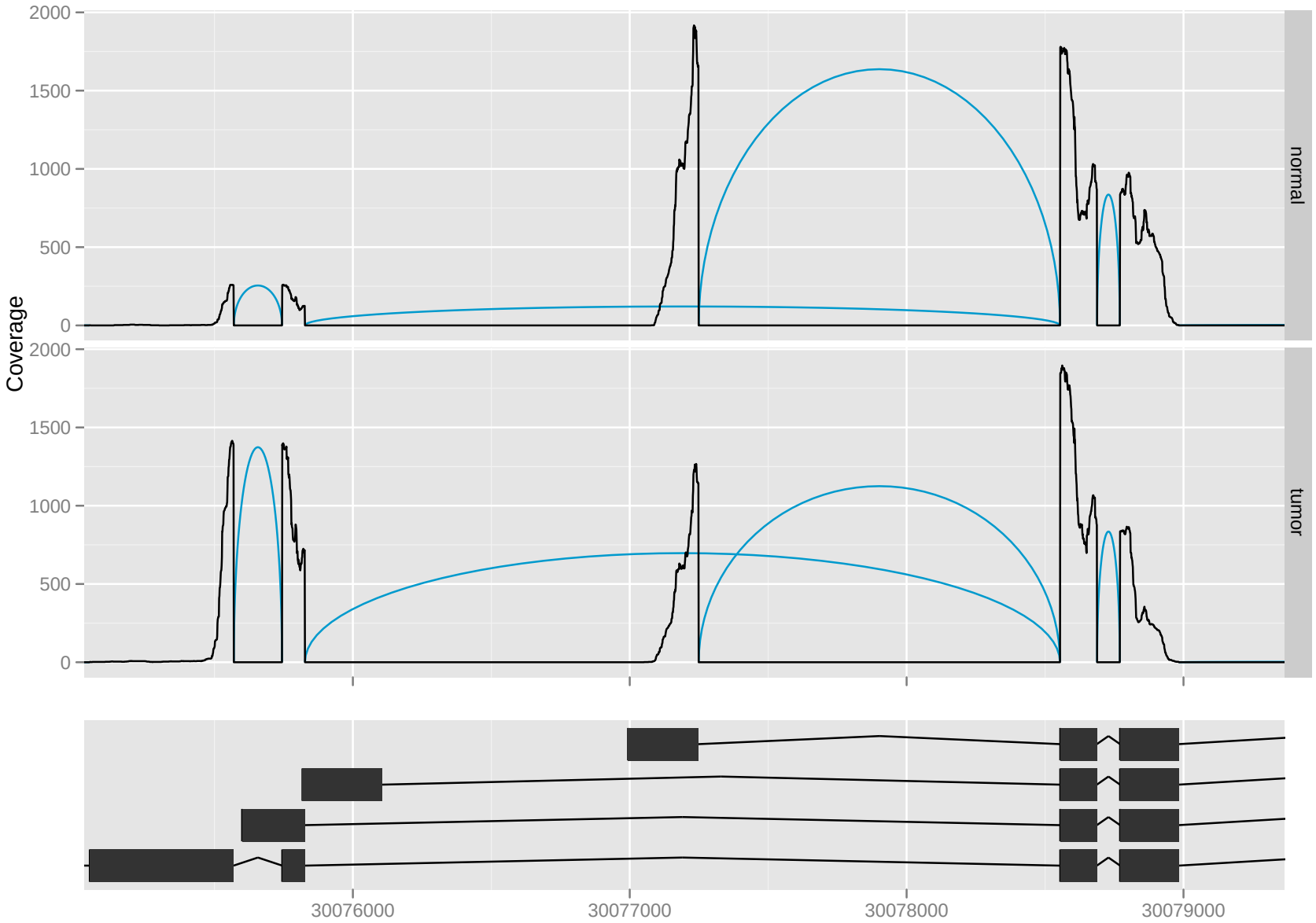
Get Uniquely Mapped Splices

```
> unique_splices_normal <-
+   summarizeSplices(ga_normal[unique_normal])
> unique_splices_tumor <-
+   summarizeSplices(ga_tumor[unique_tumor])
> unique_splices <- mstack(normal = unique_splices_normal,
+                          tumor = unique_splices_tumor)
```

Plotting Splice Counts: Score as Height

```
> read_track <- autoplot(unique_splices, geom = "arch",
+                        aes(height = score),
+                        color = "deepskyblue3",
+                        ylab = "coverage",
+                        facets = name ~ .) +
+   stat_coverage(unique_reads, facets = name ~ .)
> tracks(read_track, tx_track, heights = c(3, 1),
+        xlim = roi)
```

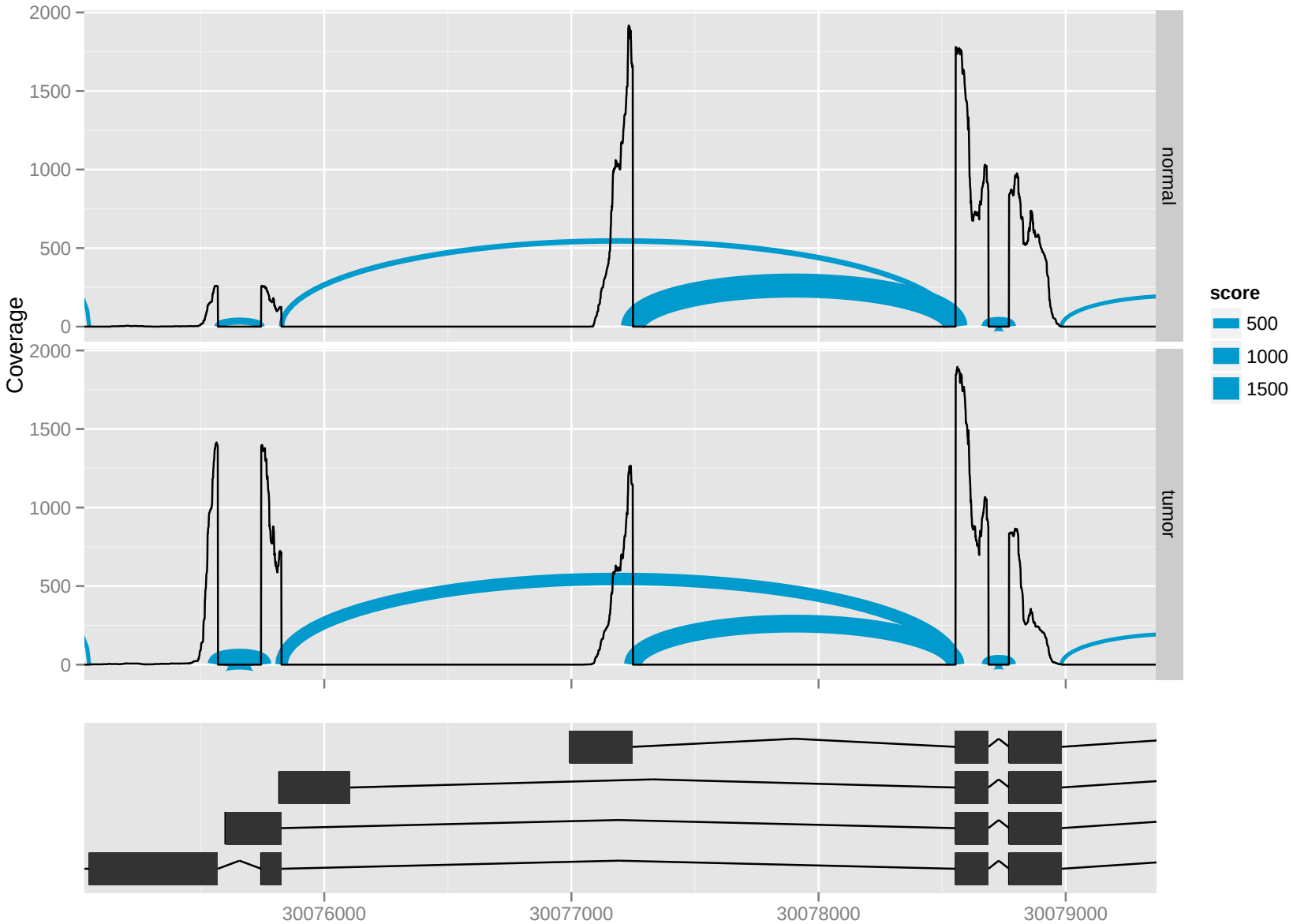
Coverage Plot, Splice Count as Height



Plotting the Splice Counts: Score as Size

```
> read_track <- autoplot(unique_splices, geom = "arch",
+                        aes(size = score,
+                            height = width / 5),
+                        color = "deepskyblue3",
+                        ylab = "coverage",
+                        facets = name ~ .) +
+   stat_coverage(unique_reads, facets = name ~ .)
> tracks(read_track, tx_track, heights = c(3, 1),
+        xlim = roi)
```

Coverage Plot, Splice Count as Size



Plotting the Novel Junctions

Setup

Get the Novel Splices

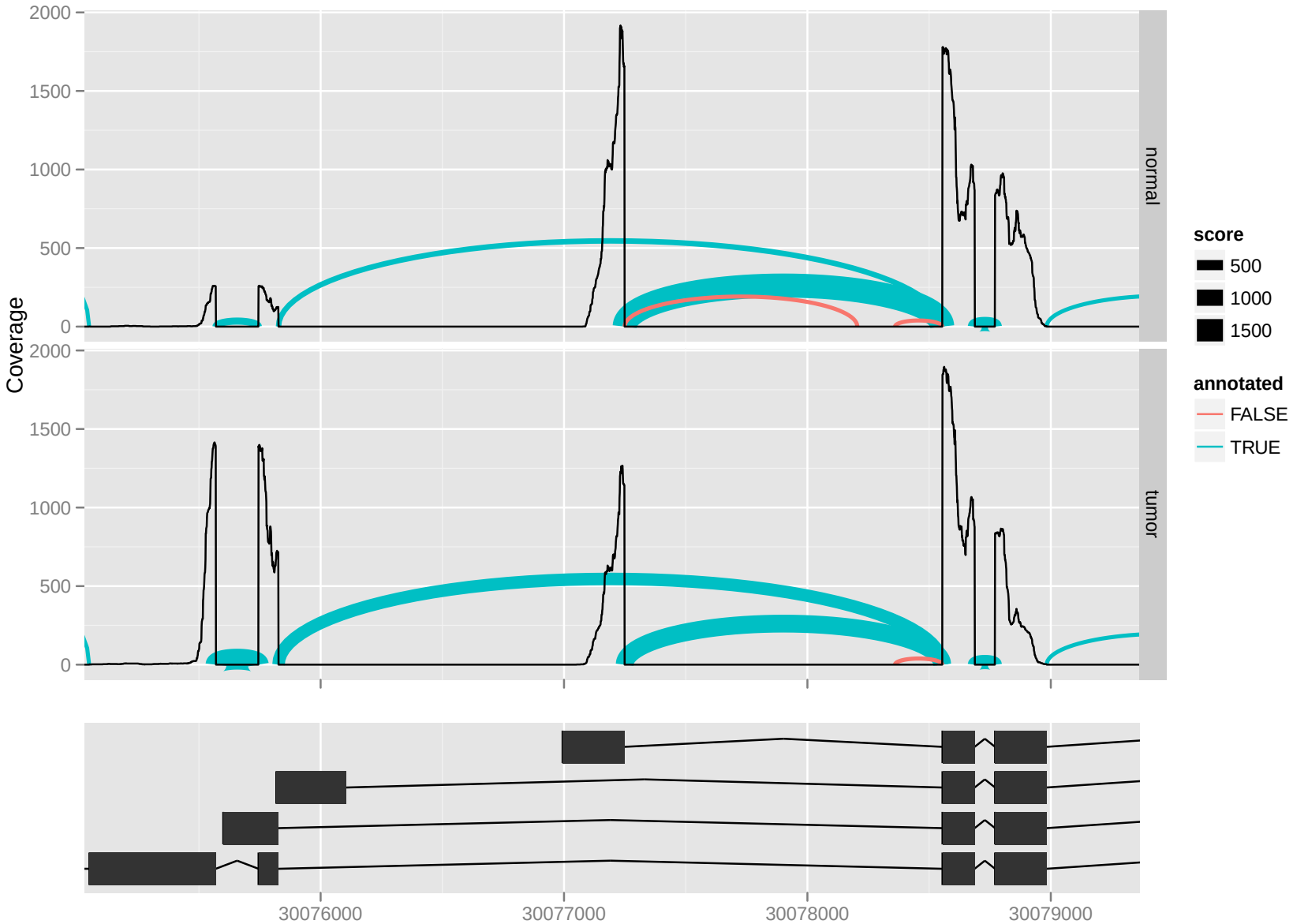
```
> splices_normal <- summarizeSplices(ga_normal)
> splices_tumor <- summarizeSplices(ga_tumor)
> all_splices <- mstack(normal = splices_normal,
+                       tumor = splices_tumor)
> novel_splices <-
+   all_splices[values(all_splices)$novel &
+               values(all_splices)$score >= 9]
> unique_novel_splices <- c(unique_splices, novel_splices)
> values(unique_novel_splices)$annotated <-
+   !values(unique_novel_splices)$novel
```

Plotting the Novel Junctions

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

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

Coverage Plot, with Novel Junctions



Acknowledgements

Ranges Packages
Herve Pages
Patrick Aboyoun
Valerie Obenchain

ggbio
Tengfei Yin
Dianne Cook

Robert Gentleman