Ranges, sequences and alignments

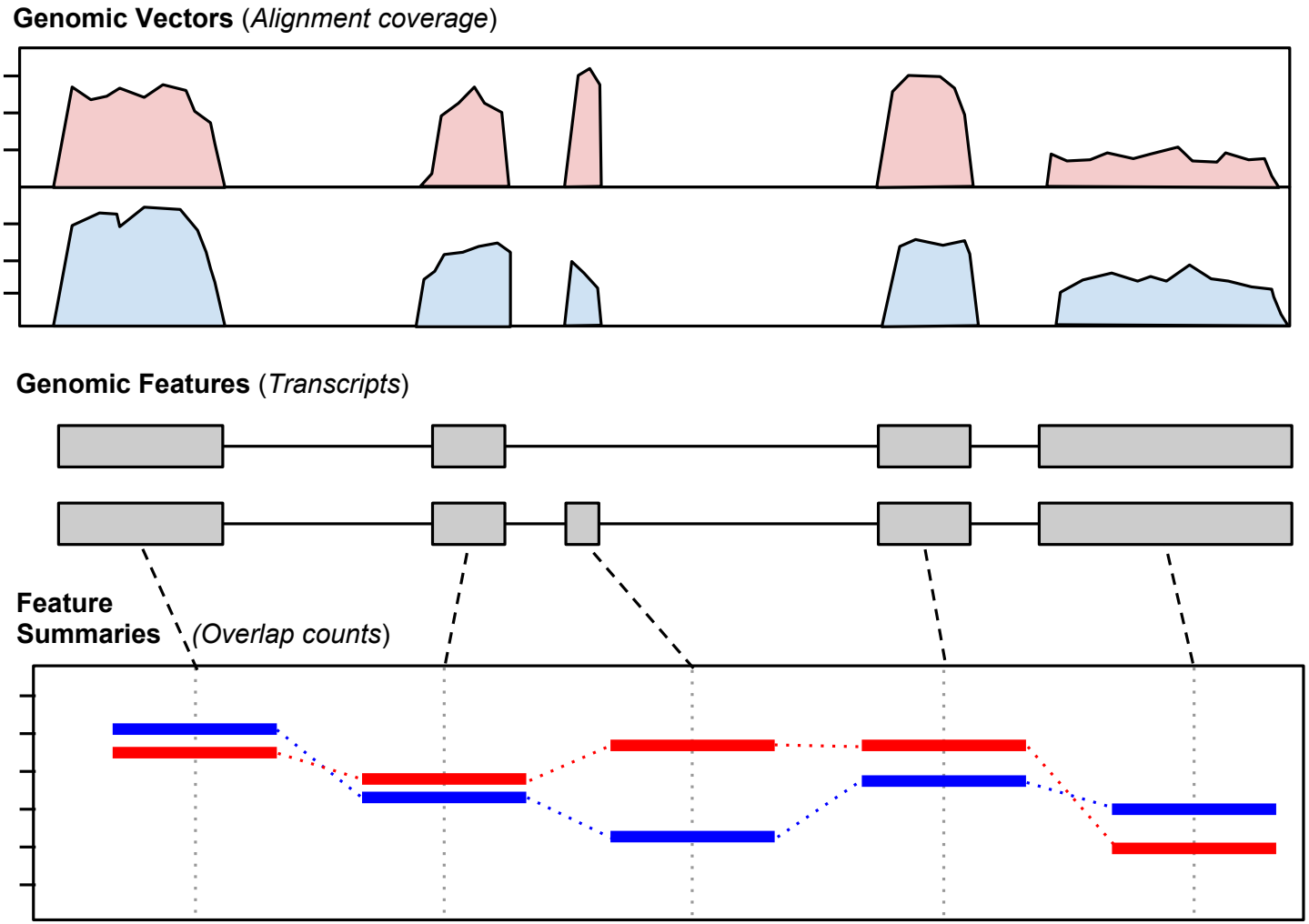
Michael Lawrence

June 23, 2014

Outline

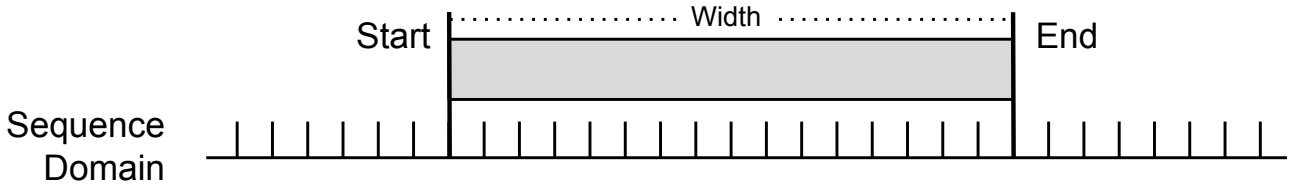
Outline

Genomic data falls into three types



The range: grand unifier of genomic data

- We define the genomic range by:
  - Sequence domain (e.g., chromosome, contig)
  - Start and end
  - Strand
  - Annotations (e.g., score, or name)



- The genomic range
  - Represents genomic features, like genes and alignments
  - Indexes into genomic vectors, like sequence and coverage
  - Links summaries, like RPKMs, to genomic locations
- The genome acts as a scaffold for data integration
- Ranges have a specialized structure and algebra, requiring specialized data types and algorithms

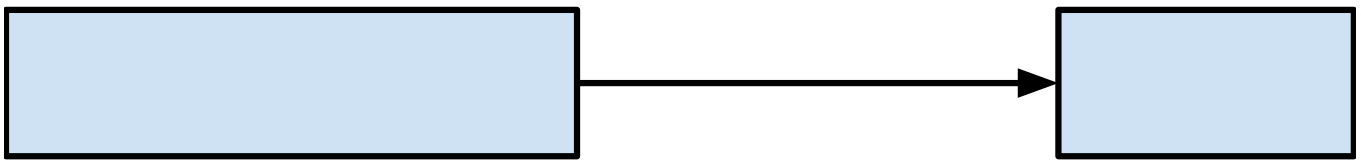
The IRanges and GenomicRanges packages

Collaborative effort with Bioconductor

- Define core classes for representing ranges, like:
  - *GRanges* for simple ranges (exons)
  - *GRangesList* for compound ranges (multi-exon transcripts)
- Algorithms for transforming, comparing, summarizing ranges.
- Run-length encoding of genome-length vectors: *Rle*
- Encapsulation of feature-level experimental summaries and metadata: *SummarizedExperiment*.

Representing a transcript with *GRanges*

We can represent any type of genomic range with *GRanges*, including the exons of a transcript



```
tx1
```

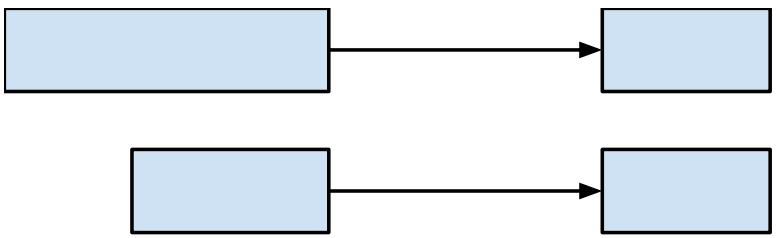
```
GRanges with 2 ranges and 1 metadata column:
      seqnames        ranges strand |     tx_name
         <Rle>     <IRanges>  <Rle> | <character>
  [1]        1 [1000, 2000]      + |           A
  [2]        1 [3000, 3500]      + |           A
  ---
```

Finding the unspliced transcript using range()

```
unspliced <- range(tx1)
```

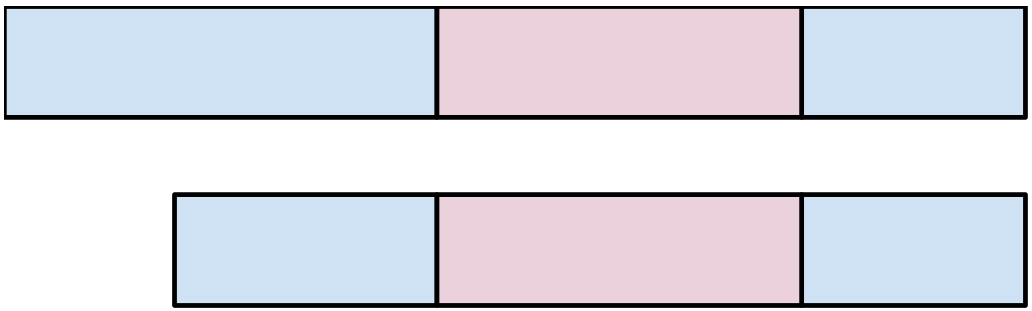
Combining multiple transcripts in a *GRangesList*

```
txList <- GRangesList(tx1, tx2)
```



Finding both unspliced transcripts using range()

```
unspliced <- range(txList)
```



range() returns the appropriate result given the type of the input.

Classes are important for complex data

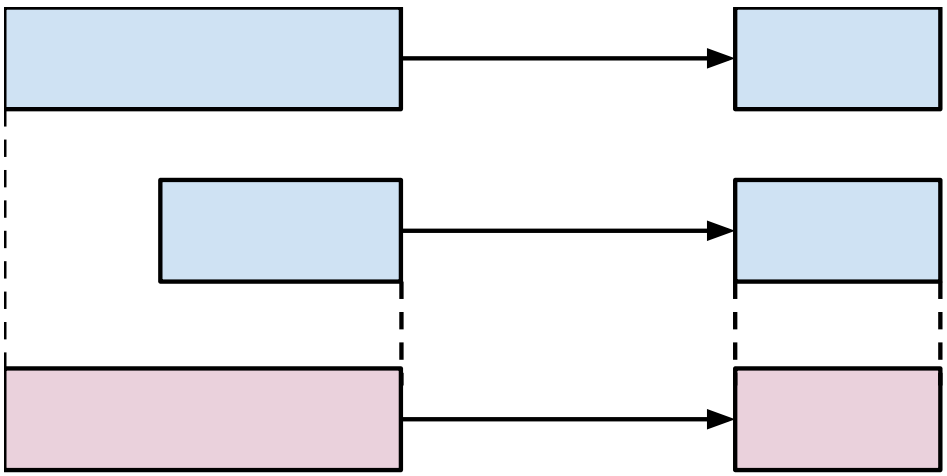
- Ensure the integrity/validity of data (strong typing)
- Hide implementation and enable code to express algorithms in an abstract way (polymorphism)
- Support analysis by better representing the semantics of the biological entity compared to an ordinary *data.frame*
- Science defies rigidity: we need hybrid objects that combine strongly typed fields with arbitrary user-level metadata

Ranges algebra

Arithmetic	`shift`, `resize`, `restrict`, `flank`
Set operations	`intersect`, `union`, `setdiff`, `gaps`
Summaries	`coverage`, `reduce`, `disjoin`
Comparison	`findOverlaps`, `findMatches`, `nearest`, `order`

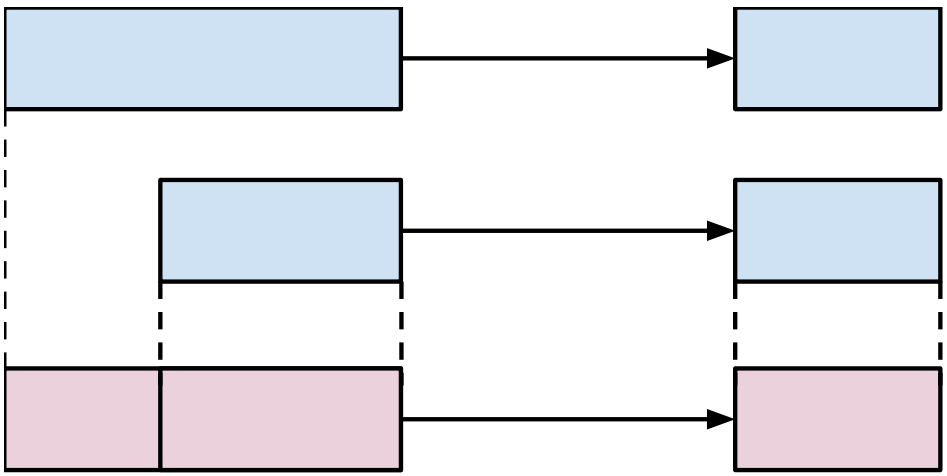
Finding "gene" regions using reduce()

```
exon.bins <- reduce(unlist(txList))
```



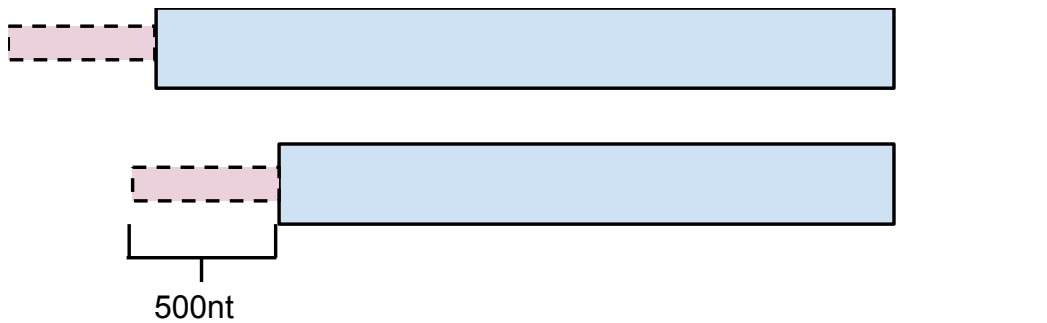
Generating DEXseq counting bins using `disjoin()`

```
exon.bins <- disjoin(unlist(txList))
```



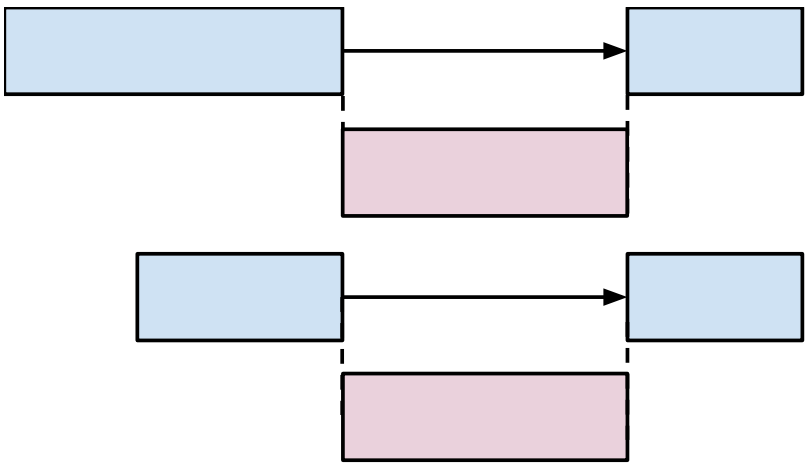
Finding promoters using `flank()`

```
promoters <- flank(unspliced, 500)
```



Finding the introns using psetdiff()

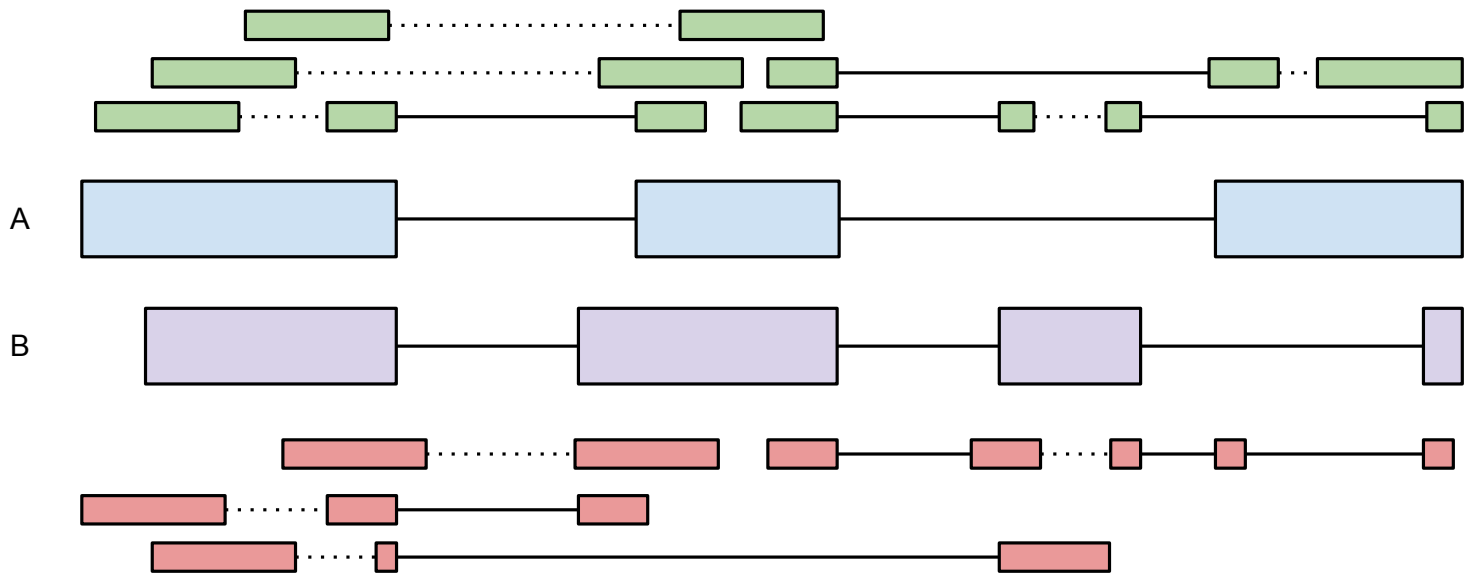
```
introns <- psetdiff(unspliced, txList)
```



Outline

Counting compatible alignments

- The `findSpliceOverlaps()` function in GenomicAlignments finds *compatible* overlaps between transcripts and RNA-seq read alignments.
- To be *compatible* a read must align completely within the exons and the read gaps should exactly match the introns over the read extent

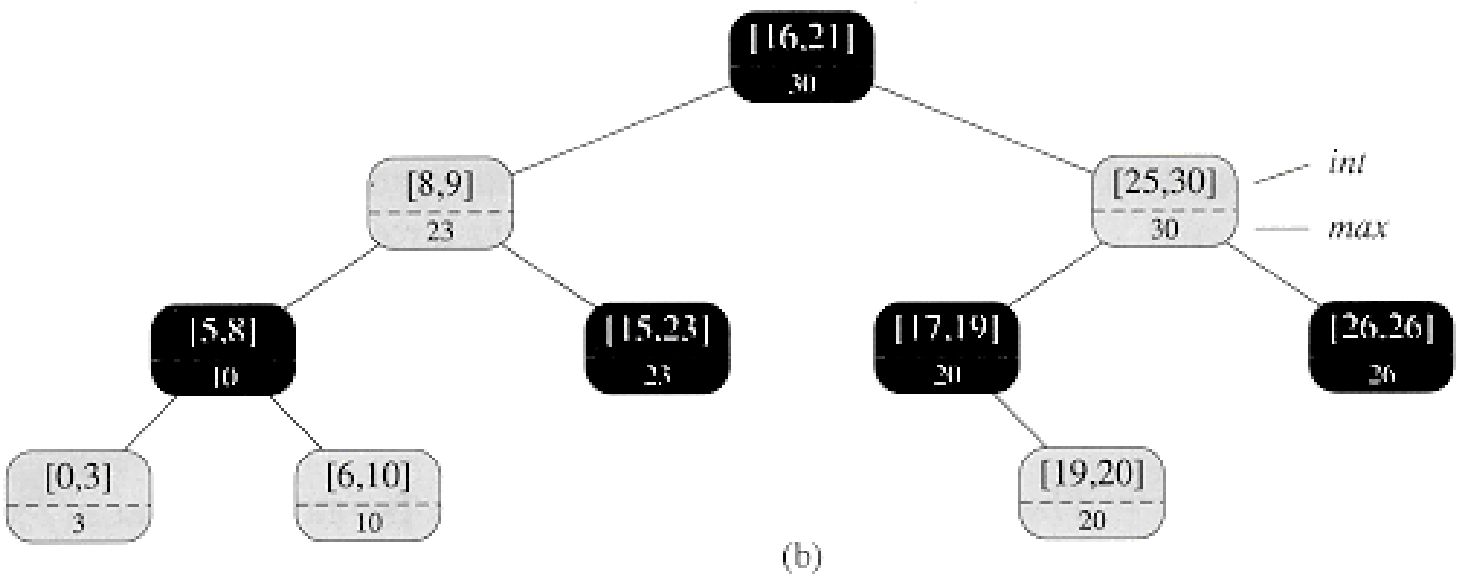


The `findSpliceOverlaps()` algorithm

1. Match read alignments to transcripts by any overlap.
2. For each match, check that the alignment segments and exons are identical over the range of the alignment.

Overlap detection algorithm

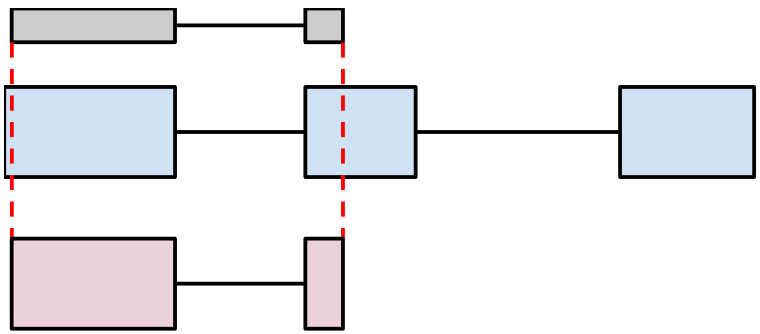
- Fast overlap detection based on a textbook interval tree algorithm.
- Extended algorithm for common case of sorted queries (does not need to restart search for each query).
- Index is represented as an *IntervalTree*, which acts like any other *Ranges* object (abstraction).



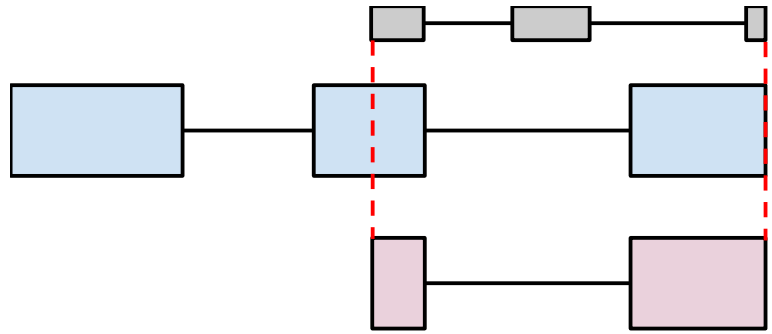
Restrict the problem to range of alignment

```
subtx <- restrict(tx, start(alignments),
                      end(alignments))
```

Hit A



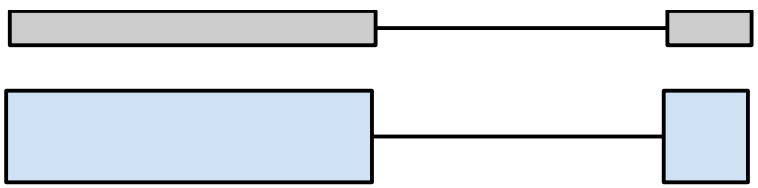
Hit B



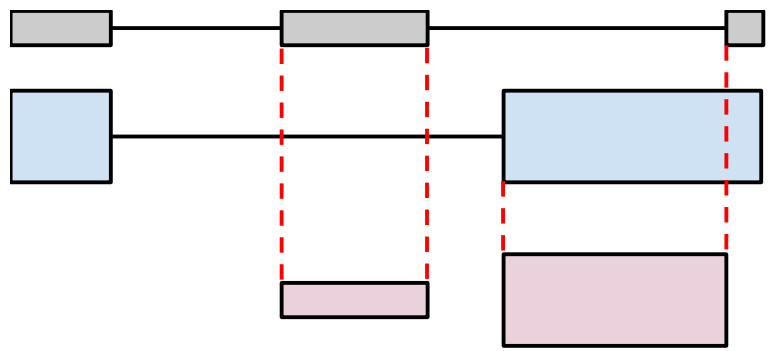
Check that alignments and sub-transcripts are equal

```
sum(width(psetdiff(alignments, subtx))) == 0L &
sum(width(psetdiff(subtx, alignments))) == 0L
```

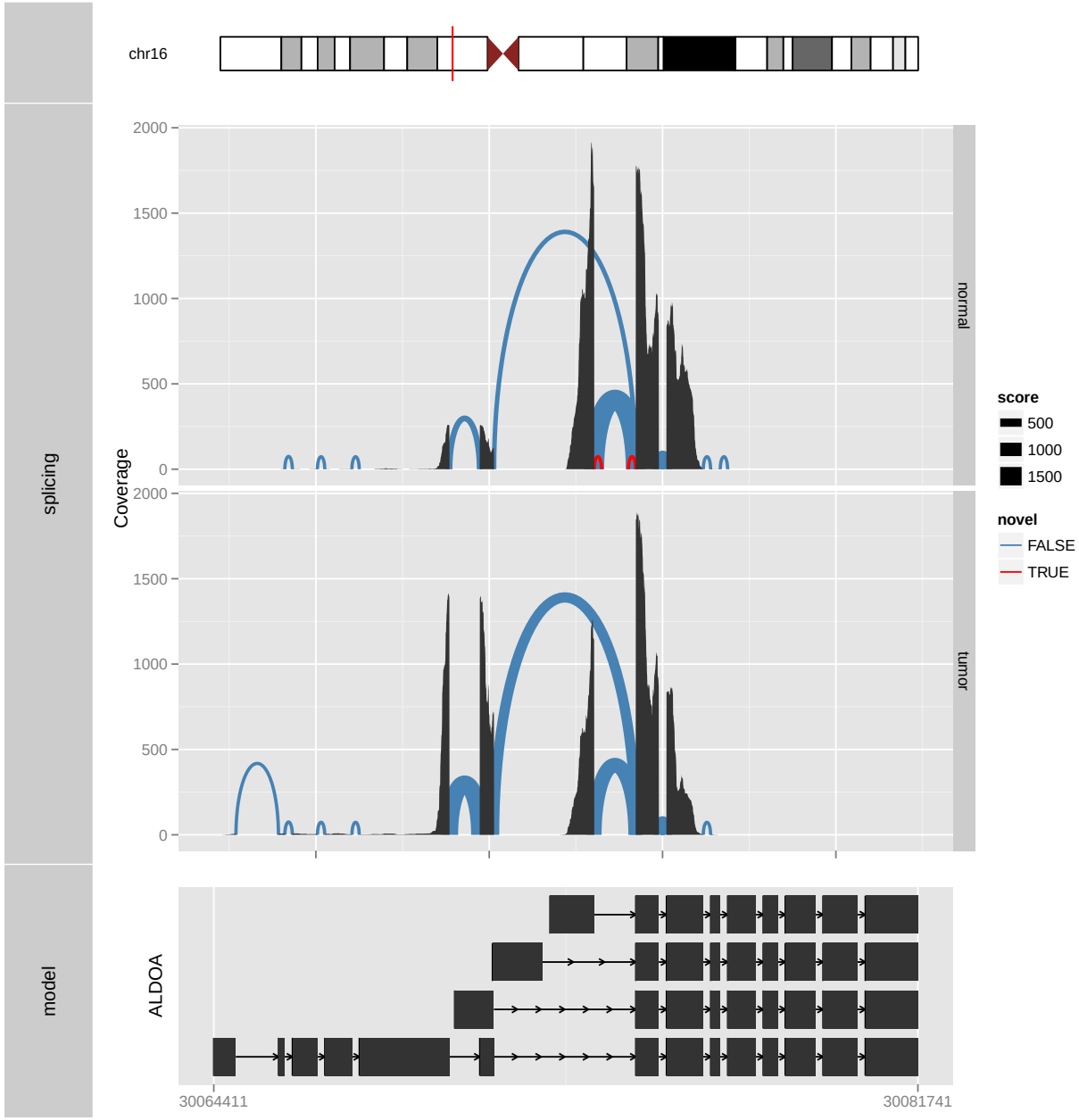
Hit A: Compatible



Hit B: Incompatible



Summary plot with ggbio



Outline

Example junction counting workflow

Steps

1. Load alignments from BAM
2. Tabulate junctions in alignments
3. Retrieve splice site sequences from reference assembly
4. Store intron locations, counts and annotations in a single object
5. Obtain splice site sequences and annotate known splices

Assumption

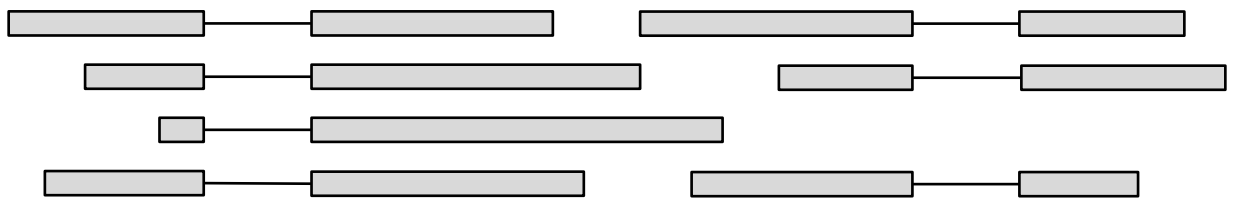
The sequences were generated by a strand-specific protocol.

Existing tools

When doing this for real, see `junctions()` in GenomicAlignments, which is much fancier and can infer the strand based on canonical splice site motifs.

Loading alignments from a BAM file

```
ga <- readGAlignments("my.bam")
reads <- grglist(ga)
```



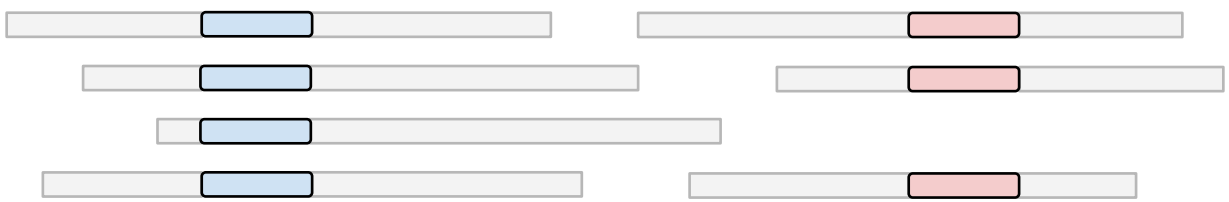
Tabulating junctions

Find the unique junctions

```
read.junctions <- psetdiff(range(reads), reads)
unique.junctions <- unique(read.junctions)
```

Count matches to unique junctions

```
counts <- countMatches(unique.junctions, read.junctions)
```



Storing summarized counts: *SummarizedExperiment*

The *SummarizedExperiment* object enables integration of feature by sample measurements with feature and sample annotations.

```
assays <- list(junction_count=cbind(A=count))
se <- SummarizedExperiment(assays, unique.junctions)
se
```

```
class: SummarizedExperiment
dim: 20024 1
exptData(0):
assays(1): 'junction_count'
rownames: NULL
colnames(1): A
colData names(0):
```

Retrieving splice site sequences

Finding the 5' splice sites

```
splice.sites <- resize(rowData(se), 2)
```

Getting and recording the sequences

```
library(BSgenome.Hsapiens.UCSC.hg19)
rowData(se)$splice.seqs <- getSeq(Hsapiens, splice.sites)
```

Example of storing arbitrary annotations on the rows/features, a feature supported by most GenomicRanges containers.

Annotate for known splices

- Reference transcript annotations are stored as *TranscriptDb* objects and distributed in individual packages.
- We can load the transcript structures as ranges and compare their introns to those derived from the reads.

Deriving the known junctions

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
tx <- exonsBy(TxDb.Hsapiens.UCSC.hg19.knownGene)
known.junctions <- psetdiff(range(tx), tx)
```

Annotating junctions for matches to reference set

```
rowData(se)$known <- se %in% known.junctions
```

Outline

Summary

- The range integrates the different types of genomic data.
- IRanges and GenomicRanges define the fundamental abstractions, data types and utilities for representing, manipulating, comparing, and summarizing ranges.
- The data structures support storage of arbitrary metadata, and are well integrated with reference annotation sources and visualization packages.
- We applied these tools to the analysis of transcript expression and junction counting in the context of RNA-seq data.
- Broader applications include: variant calling, ChIP-seq, proteomics, and even general fields like time series analysis.

Acknowledgements

- Herve Pages
- Patrick Aboyoun
- Valerie Oberchain
- Martin Morgan
- Robert Gentleman