Strings and Ranges in R and Bioconductor

Mark Dunning

Cancer Research Uk
Cambridge Research Institute
Robinson Way
Cambridge

November 2, 2013

Outline

Aims

By the end of this lecture and practical you should be familiar with

- How DNA sequences are represented in R
- How to create and compare genomic intervals
- How to read fastq and bam files into R
- Interactions between the packages

Previously....

- Whistle-stop tour of sequencing.
  - fastq
  - bam
- Recap of R and Bioconductor
  - Vectors, data.frames, lists
  - Functions
  - How to get help

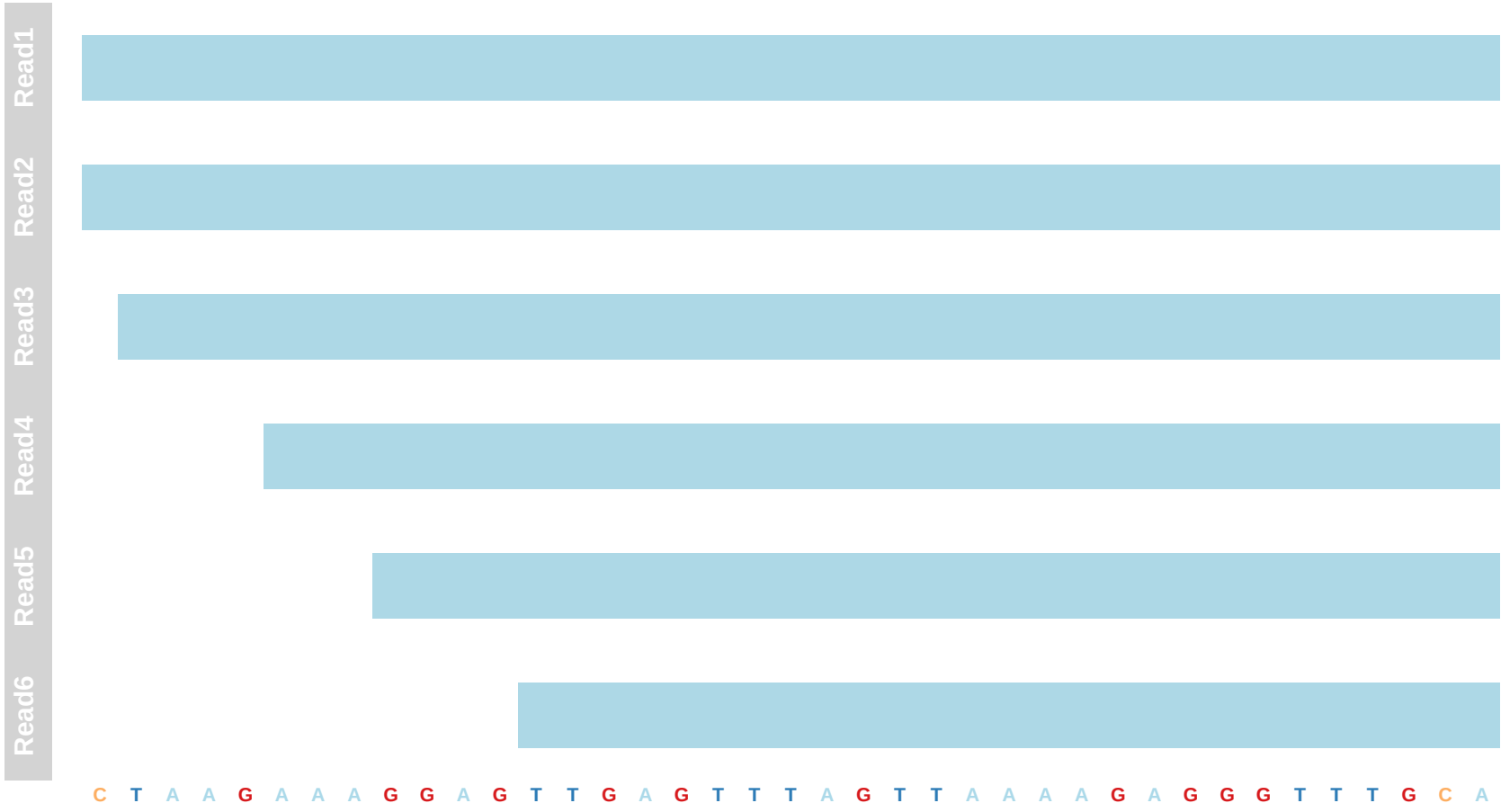
Introduction

Sequencing produces millions of reads. e.g in fastq format

```
Read 1
CTAAGAAAGGAGTTGAGTTTAGTTAAAAGAGGGTTTGCATCGAGATATTAGATTTGGGATAGACATGTACC
Read 2
CTAAGAAAGGAGTTGAGTTTAGTTAAAANAGNGTTTGCATCGAGATATTAGATNTNGGNTNGACATGTNCC
Read 3
TAAGAAAGGAGTTGAGTTTAGTTAAAAGAGGGTTTGCATCCAGATATTAGATTTGGGATAGACATGTACCT
Read 4
AAAGGAGTTGAGTTTAGTTAAAAGAGGGTTTGCATCGAGATATTAGATTTGGGATAGACATGTACCTTTTA
Read 5
GGAGTTGAGTTTAGTTAAAAGAGGGTTTGCATCGAGATATTAGATTTGGGATAGACATGTACCTTTTACAG
Read 6
TTGAGTTTAGTTAAAAGAGGGTTTGCATCGAGATATTAGATTTGGGATAGACATGTACCTTTTACAGCATT
```

These need to be compared to the genome (aligned) and we record the chromosome and coordinates that each sequence aligns to, often with quality information.

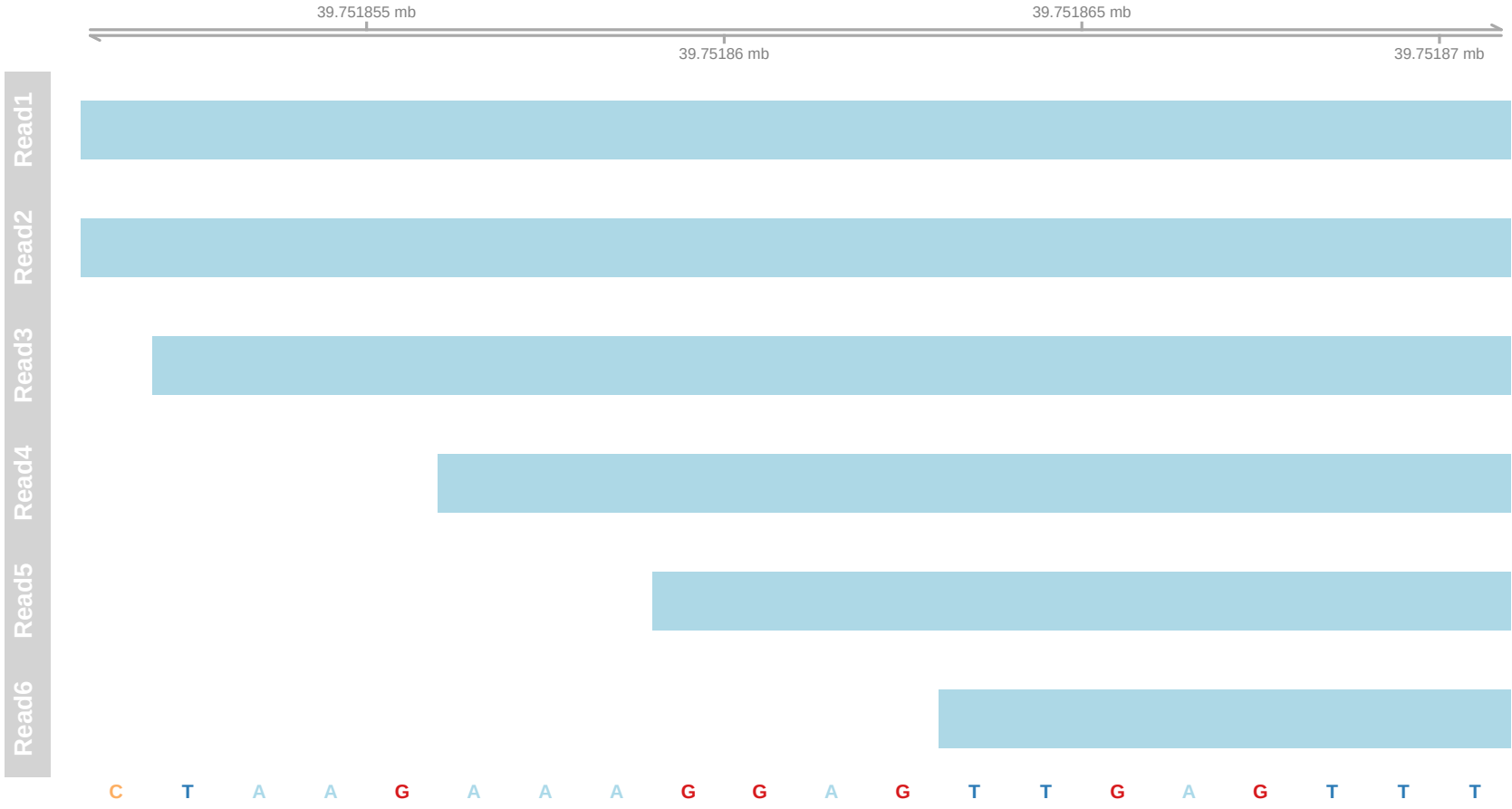
Need consistent representation of (1) genome and (2) reads



Reads come with quality score and IDs that also need to be captured

Associating reads with positions

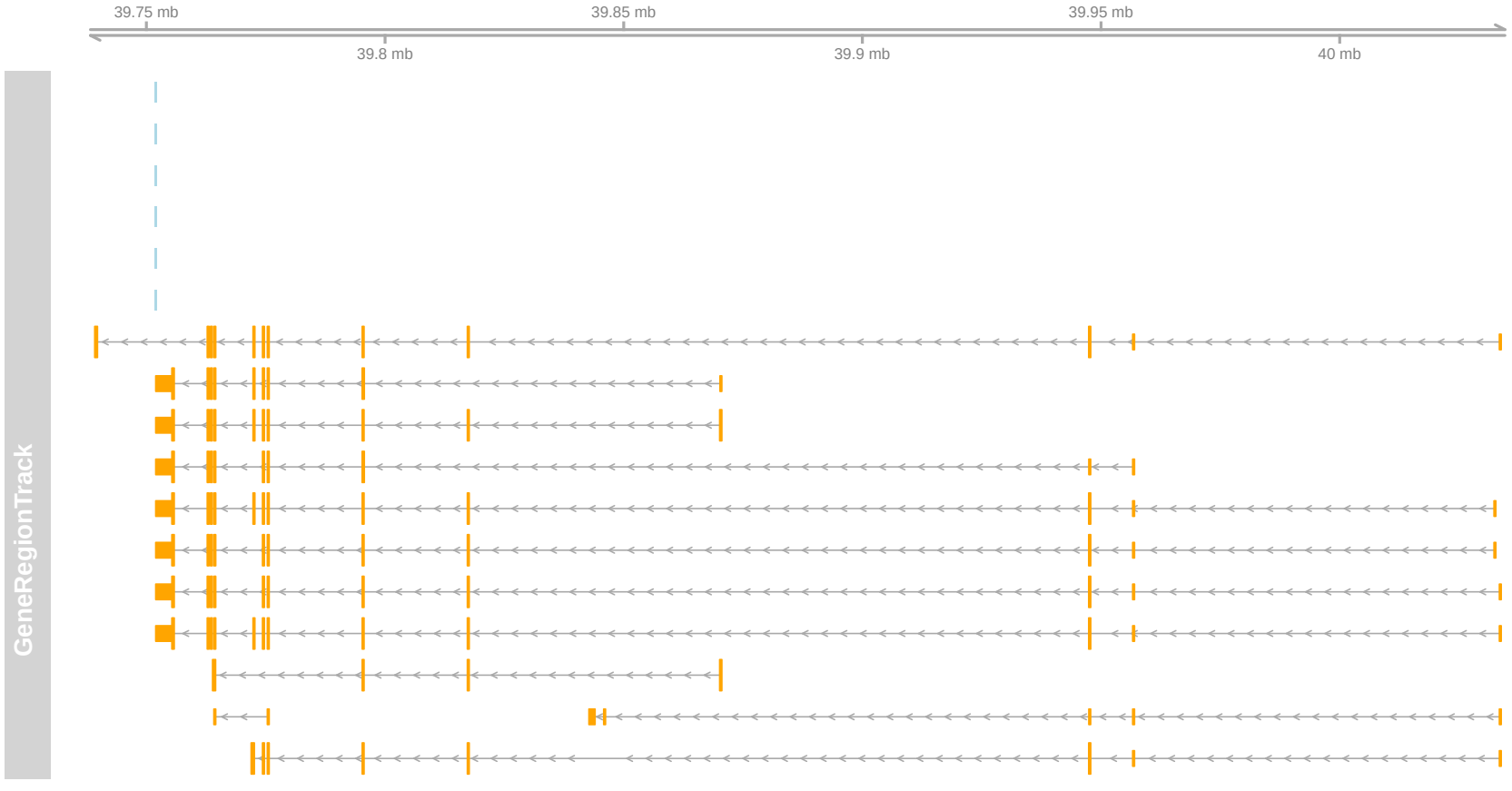
Often we are given the mapped location of reads



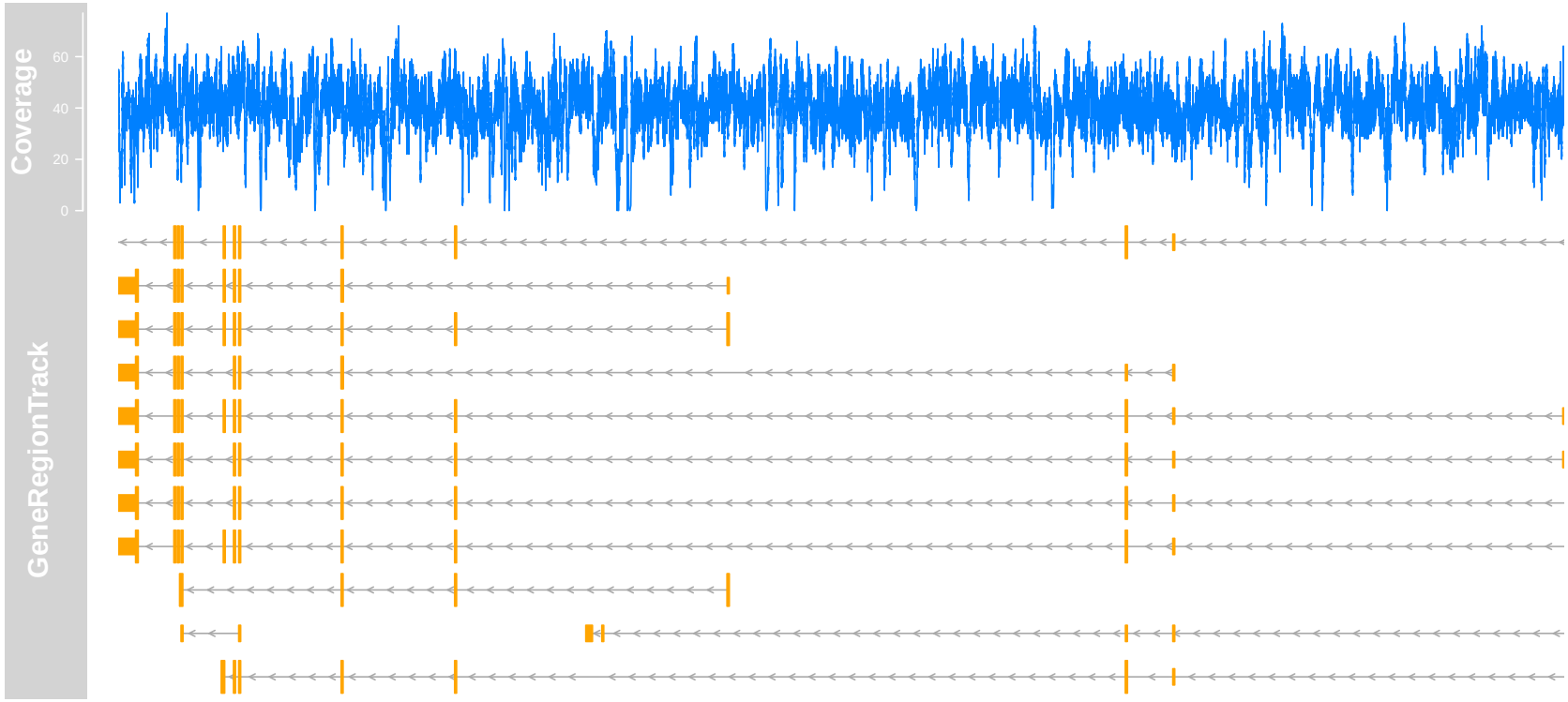
Need a way of representing alignments and associated qualities

Associating with Genomic Features

We will often want to find information about the genomic region around the reads



Or use definitions of genomic regions to interrogate the data



Need representation of genomic regions of interest

Why bother doing these things in R?

- Interactivity and data exploration
- Quality assessment
- Access to exisiting statistical and visualisation techniques (e.g. limma)
- Reproducibility

We will not do alignment of reads in R

Packages we will meet

- Biostrings - Manipulation of DNA sequences in R
- ShortRead - Input / output of fastq files and quality assessment
- IRanges - Low-level classes and functions for dealing with intervals of consecutive values
- GRanges - Functions for representing ranges with sequence and strand information
- Rsamtools - Input of bam files

DNA Sequences

```
## [1] "AGTCTGCTCCAG"
## [2] "CGCAGTCGCGG"
## [3] "TGGTCTTTGTTCACTCTT"
## [4] "AGAAAAAGCCCTTCG"
```

We can represent sequences of A, T, C, G. Several useful operations are possible

```
myseq
```

```
## [1] "AGTCTGCTCCAG"
## [2] "CGCAGTCGCGG"
## [3] "TGGTCTTTGTTCACTCTT"
## [4] "AGAAAAAGCCCTTCG"
```

```
gsub("ATG", "atg", myseq)
```

```
## [1] "AGTCTGCTCCAG"
## [2] "CGCAGTCGCGG"
## [3] "TGGTCTTTGTTCACTCTT"
## [4] "AGAAAAAGCCCTTCG"
```

Biostrings package

However, the Biostrings package is specifically-designed for biological sequences

```
library(Biostrings)
myseq <- DNAStringSet(randomStrings)
```

Biostrings operations

```
myseq

##   A DNAStringSet instance of length 100
##       width seq
##    [1]    12 AGTCTGCTCCAG
##    [2]    11 CGCAGTCGCGG
##    [3]    18 TGGTCTTTGTTCACTCTT
##    [4]    15 AGAAAAAGCCCTTCG
##    [5]    17 GTTAAGATGCTTACTGA
##    ...   ... ...
##   [96]    13 ACTTCCTTTTCTG
##   [97]    12 TAATGTCAAGAG
##   [98]    10 TGACTCTCAA
##   [99]    14 TTATAGACTCTGGA
##  [100]    13 GATCACAGCGCGG
```

Biostrings operations

```
myseq[1:2, ]

##   A DNAStringSet instance of length 2
##     width seq
## [1]    12 AGTCTGCTCCAG
## [2]    11 CGCAGTCGCGG
```

This doesn't work!

```
myseq[, 1:2]
```

```
subseq(myseq, 1, 3)
```

```
##   A DNAStringSet instance of length 100
##       width seq
##   [1]     3 AGT
##   [2]     3 CGC
##   [3]     3 TGG
##   [4]     3 AGA
##   [5]     3 GTT
##   ...   ... ...
##  [96]     3 ACT
##  [97]     3 TAA
##  [98]     3 TGA
##  [99]     3 TTA
## [100]     3 GAT
```

Similar to substr

Biostrings operations

Accessor functions must be used to retrieve the data

```
width(myseq)[1:2]

## [1] 12 11

length(width(myseq))

## [1] 100

table(width(myseq))

##
## 10 11 12 13 14 15 16 17 18 19 20
## 15 13  7  9  8  7  6  6  8 15  6
```

Can subset based on properties of the set

```
myseq[width(myseq) > 19]
```

```
##   A DNAStringSet instance of length 6
##     width seq
## [1]    20 ATTAGCCAGTGTTATGTACT
## [2]    20 CAGGTTGCAATTCATTGGCA
## [3]    20 ACACATGTGTCCTTCTTAAG
## [4]    20 ATGTCGATGAACGTATGGTC
## [5]    20 TTTAGGAAGCAGATGTTCTA
## [6]    20 CCCTCTCGGCAGAACGAGGG
```

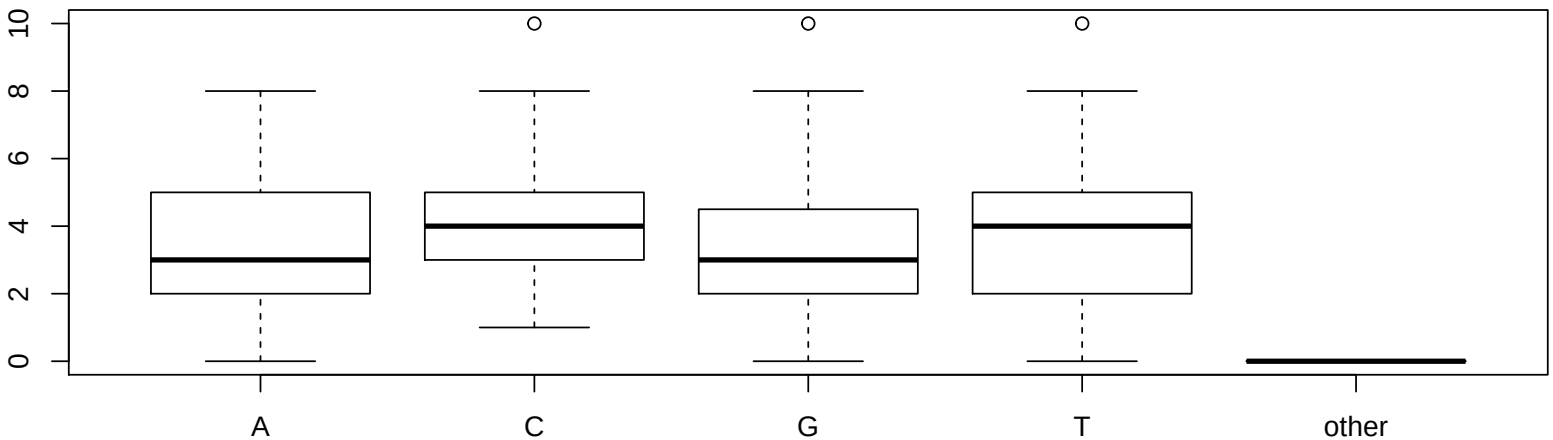
```
myseq[as.character(substr(myseq, 1, 3)) ==
    "TTC"]
```

```
##   A DNAStringSet instance of length 1
##     width seq
## [1]    12 TTCCAGGGTTAC
```

Some useful string operation functions are provided

```
alphabetFrequency(myseq[1:4, ], baseOnly = TRUE)

##      A C G  T other
## [1,] 2 4 3  3     0
## [2,] 1 4 5  1     0
## [3,] 1 4 3 10     0
## [4,] 6 4 3  2     0

af <- alphabetFrequency(myseq, baseOnly = TRUE)
myseq[af[, 1] == 0, ]

##   A DNAStringSet instance of length 3
##     width seq
## [1]    10 CTCCTGGTCC
## [2]    11 TCGCCGCCCCT
## [3]    19 CGGGTGCTCGCTTCCGCCT
```

```
boxplot(af)
```



More-specialised features

```
myseq[1:2, ]
```

```
##   A DNAStringSet instance of length 2
##     width seq
## [1]    12 AGTCTGCTCCAG
## [2]    11 CGCAGTCGCGG
```

```
reverse(myseq[1:2, ])
```

```
##   A DNAStringSet instance of length 2
##     width seq
## [1]    12 GACCTCGTCTGA
## [2]    11 GGCGCTGACGC
```

```
reverseComplement(myseq[1:2, ])
```

```
##   A DNAStringSet instance of length 2
##     width seq
## [1]    12 CTGGAGCAGACT
## [2]    11 CCGCGACTGCG
```

```
translate(myseq[1:4, ])
```

```
##   A AAStringSet instance of length 4
##     width seq
## [1]     4 SLLQ
## [2]     3 RSR
## [3]     6 WSLFTL
## [4]     5 RKSPS
```

The genome as a string - BSGenome

```
library(BSgenome)
head(available.genomes())

## [1] "BSgenome.Alyrata.JGI.v1"
## [2] "BSgenome.Amellifera.BeeBase.assembly4"
## [3] "BSgenome.Amellifera.UCSC.apiMel2"
## [4] "BSgenome.Athaliana.TAIR.04232008"
## [5] "BSgenome.Athaliana.TAIR.TAIR9"
## [6] "BSgenome.Btaurus.UCSC.bosTau3"

available.genomes()[23:25]

## [1] "BSgenome.Hsapiens.UCSC.hg17"
## [2] "BSgenome.Hsapiens.UCSC.hg18"
## [3] "BSgenome.Hsapiens.UCSC.hg19"
```

The human genome

```
library(BSgenome.Hsapiens.UCSC.hg19)
hg19 <- BSgenome.Hsapiens.UCSC.hg19::Hsapiens
hg19

## Human genome
## |
## | organism: Homo sapiens (Human)
## | provider: UCSC
## | provider version: hg19
## | release date: Feb. 2009
## | release name: Genome Reference Consortium GRCh37
## |
## | single sequences (see '?seqnames'):
## |   chr1
## |   chr2
## |   chr3
## |   chr4
## |   chr5
## |   chr6
## |   chr7
## |   chr8
## |   chr9
```

Retrieve Sequences

```
tp53 <- getSeq(hg19, "chr17", 7577851, 7590863)
tp53

##   13013-letter "DNAString" instance
## seq: TTGTATTTTTCAGTAG...GGGGAAAACCCCAATC

as.character(tp53)

## [1] "TTGTATTTTTCAGTAGAGACGGGGTTTCACCGTTAGCCAGGATGGTCTCGATCTCCCAACCTC

alphabetFrequency(tp53, baseOnly = TRUE)

##     A     C     G     T other
##  3102  3375  3025  3511     0

subseq(tp53, 1000, 1010)

##   11-letter "DNAString" instance
## seq: TATAGGTGTGC
```

Timings

Don't need to load the whole genome into memory, so reading a particular sequence is **fast**

```
system.time(tp53 <- getSeq(hg19, "chr17",
    7577851, 7598063))

##    user  system elapsed
##   0.861   0.030   0.895
```

Manipulating sequences

We can now use Biostrings operations to manipulate the sequence

```
translate(subseq(tp53, 1000, 1010))

##   3-letter "AAString" instance
## seq: YRC

reverseComplement(subseq(tp53, 1000, 2000))

##   1001-letter "DNAString" instance
## seq: CCTATGGAAACTGTGA...GTGGTGCACACCTATA
```

Later, we will show how the sequences for genomic features can be extracted

Fastq Recap

Recall that sequence reads are represented in text format

```
readLines(sampleFQ(), n = 10)
```

```
##  [1] "@SRR020521.1 EAS139_33_FC301DUAAXX_0_2_1_206_461/1"
##  [2] "GTCTATAGTTCTCAAGTTTATGTCCATTTGAGCTC"
##  [3] "+"
##  [4] ">>>>>>>>>>>>>>>>>>>><>>>>>+>+:48><"
##  [5] "@SRR020521.3188018 EAS139_33_FC301DUAAXX_0_2_33_1708_1368/1"
##  [6] "CTTGAGAAGATCATCATTGTAAAGAGGCAAACTTG"
##  [7] "+"
##  [8] ">>>>4>>>>>>>>>>>>>>><>>><<><<><>"
##  [9] "@SRR020521.3332221 EAS139_33_FC301DUAAXX_0_2_35_514_899/1"
## [10] "ATCAAATGGAATCGAATGGAATCTTCATCAATTGG"
```

It should be possible to represent these as **Biostrings** objects

The Short Read package

Has convenient functions for reading fastq files and performing quality assessment

```
library(ShortRead)
fq <- readFastq(sampleFQ())
fq

## class: ShortReadQ
## length: 1000000 reads; width: 35 cycles
```

```
sread(fq)[1:3, ]
```

```
##   A DNAStringSet instance of length 3
##     width seq
## [1]    35 GTCTATAGTTCTCA...TCCATTTGAGCTC
## [2]    35 CTTGAGAAGATCAT...AGAGGCAAACTTG
## [3]    35 ATCAAATGGAATCG...CTTCATCAATTGG
```

```
quality(fq)[1:3, ]
```

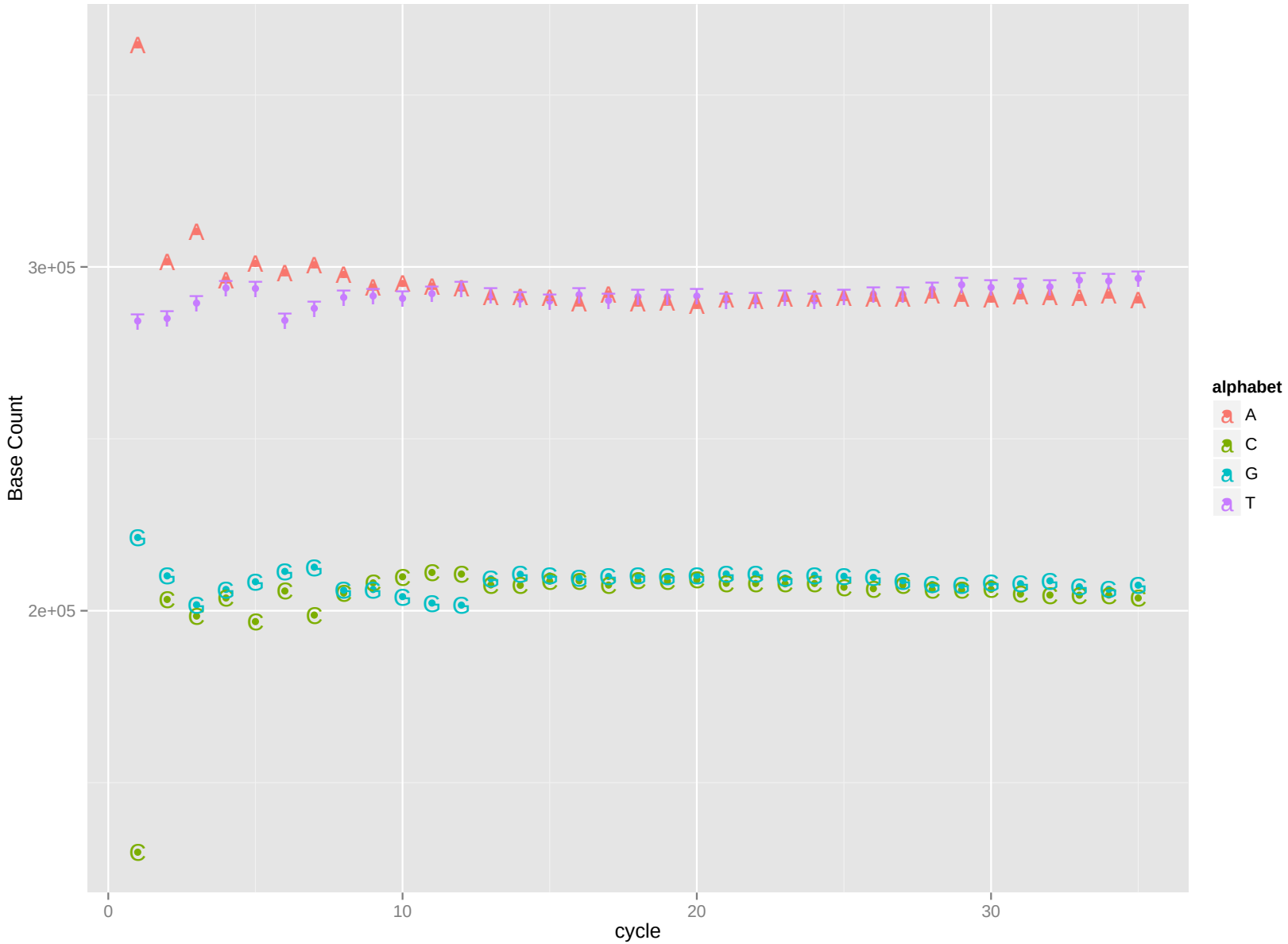
```
## class: FastqQuality
## quality:
##   A BStringSet instance of length 3
##     width seq
## [1]    35 >>>>>>>>>>>>>>...>>>>>+>+:48><
## [2]    35 >>>>4>>>>>>>>>...<>>><<<><<><>
## [3]    35 9>>>6>>49>>>>:...2>>4<<:-:<70%
```

```
id(fq)[1:3]

##   A BStringSet instance of length 3
##     width seq
## [1]    49 SRR020521.1 EA...2_1_206_461/1
## [2]    58 SRR020521.3188...3_1708_1368/1
## [3]    56 SRR020521.3332..._35_514_899/1
```

Could parse the ID for run names, lanes, tiles etc

```
abc <- alphabetByCycle(sread(fq))
abc[1:4, 1:8]
```

```
##          cycle
## alphabet   [,1]   [,2]   [,3]   [,4]
##        A 364639 301566 310341 296242
##        C 129777 203283 198450 203706
##        G 221299 210142 201737 206191
##        T 284285 285009 289472 293861
##          cycle
## alphabet   [,5]   [,6]   [,7]   [,8]
##        A 301006 298385 300656 297781
##        C 196845 205745 198741 205139
##        G 208434 211436 212665 205970
##        T 293715 284434 287938 291110
```


Base Count
3e+05
2e+05
0
10
20
30
cycle
alphabet
A
C
G
T

Conversion of qualities

Phred quality scores are integers from 0 to 50 that are stored as ASCII characters after adding 33. The basic R functions rawToChar and charToRaw can be used to convert

```
phred <- 1:9
phreda <- paste(sapply(as.raw((phred) + 33),
    rawToChar), collapse = "")
phreda

## [1] "\"#$%&'()*"

as.integer(charToRaw(phreda)) - 33

## [1] 1 2 3 4 5 6 7 8 9
```

```
quality(fq)[1]
```

```
## class: FastqQuality
## quality:
##   A BStringSet instance of length 1
##     width seq
## [1]    35 >>>>>>>>>>>>>>...>>>>>+>+:48><
```

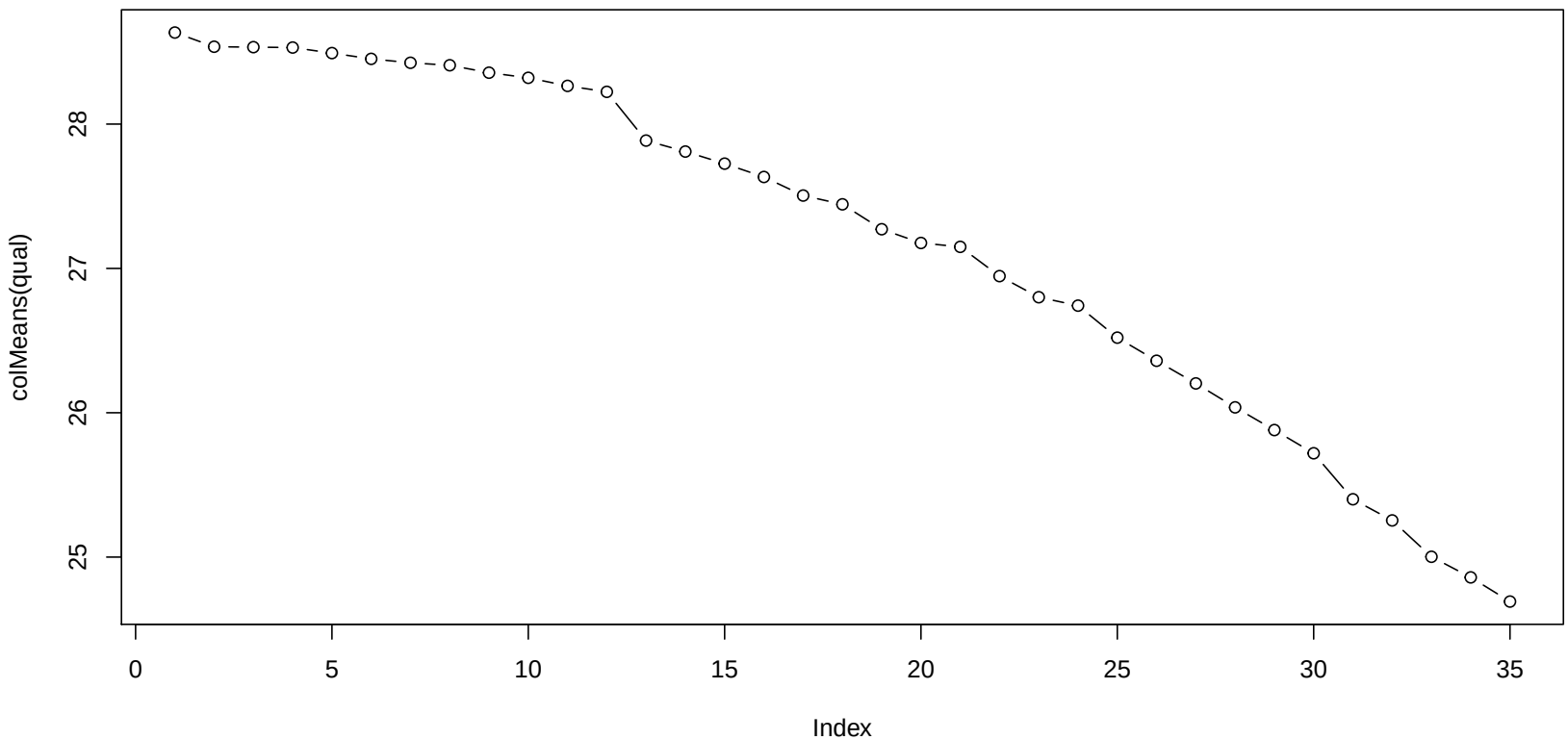
```
as.integer(charToRaw(">>>>>>>>>>>>>>>>>>>>>>>><>>>>>+>+:48><")) -
    33
```

```
##  [1] 29 29 29 29 29 29 29 29 29 29 29 29
## [13] 29 29 29 29 29 29 29 29 29 27 29 29
## [25] 29 29 29 10 29 10 25 19 23 29 27
```

A shortcut

```
qual <- as(quality(fq), "matrix")
dim(qual)

## [1] 1000000      35

qual[1, ]

##  [1] 29 29 29 29 29 29 29 29 29 29 29 29
## [13] 29 29 29 29 29 29 29 29 29 27 29 29
## [25] 29 29 29 10 29 10 25 19 23 29 27
```

```
plot(colMeans(qual), type = "b")
```



Read Occurrence

```
tbl <- tables(fq)
names(tbl)

## [1] "top"          "distribution"

tbl$top[1:5]

## AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
##                                  37
## GAATGGAATGGAATGGAATGGAATGGAATGGAATG
##                                  37
## ATTCCATTCCATTCCATTCCATTCCATTCCATTCC
##                                  36
## GGAATGGAATGGAATGGAATGGAATGGAATGGAAT
##                                  32
## AACCCTAACCCTAACCCTAACCCTAACCCTAACCC
##                                  27
```

```
head(tbl$distribution)
```

```
##   nOccurrences nReads
## 1            1 977827
## 2            2   6801
## 3            3   1428
## 4            4    496
## 5            5    180
## 6            6     72
```

977827 sequences appear only once, 6801 appear twice, etc.

We can trim the reads if required

```
subseq(sread(fq), 1, 10)
```

```
##   A DNAStringSet instance of length 1000000
##             width seq
##       [1]    10 GTCTATAGTT
##       [2]    10 CTTGAGAAGA
##       [3]    10 ATCAAATGGA
##       [4]    10 TCATATCCTA
##       [5]    10 CTAAAGTTTT
##       ...   ... ...
##  [999996]    10 TTGTATGTGC
##  [999997]    10 ATTTCGTCTT
##  [999998]    10 TAATTGTCTA
##  [999999]    10 AAAAACAGAC
## [1000000]    10 TCCTTCTCTC
```

or search for adaptor sequence

```
grep(myAdaptor, sread(fq))
```

And write the resulting files

```
write.XStringSet(...)
```

We could even do some 'aligning' in R

```
system.time(aln <- matchPattern(as.character(sread(fq)[2]),
    hg19[["chr1"]]))

##    user  system elapsed
##   2.561   0.260  16.301

aln

##   Views on a 249250621-letter DNAString subject
## subject: NNNNNNNNNNNNNNN...NNNNNNNNNNNNNNN
## views:
##         start       end width
## [1] 249066163 249066197    35 [CTT...TG]
```

```
sread(fq)[2]

##   A DNAStringSet instance of length 1
##     width seq
## [1]    35 CTTGAGAAGATCAT...AGAGGCAAACTTG

getSeq(hg19, "chr1", 249066163, 249066197)

##   35-letter "DNAString" instance
## seq: CTTGAGAAGATCATCATTGTAAAGAGGCAAACTTG

identical(as.character(sread(fq)[2]), as.character(getSeq(hg19,
    "chr1", 249066163, 249066197)))

## [1] TRUE
```

We might want to know more about the region between 249066163 and 249066197 on chromosome 1, or find other reads in this region. For this we will need a way of representing genomic ranges

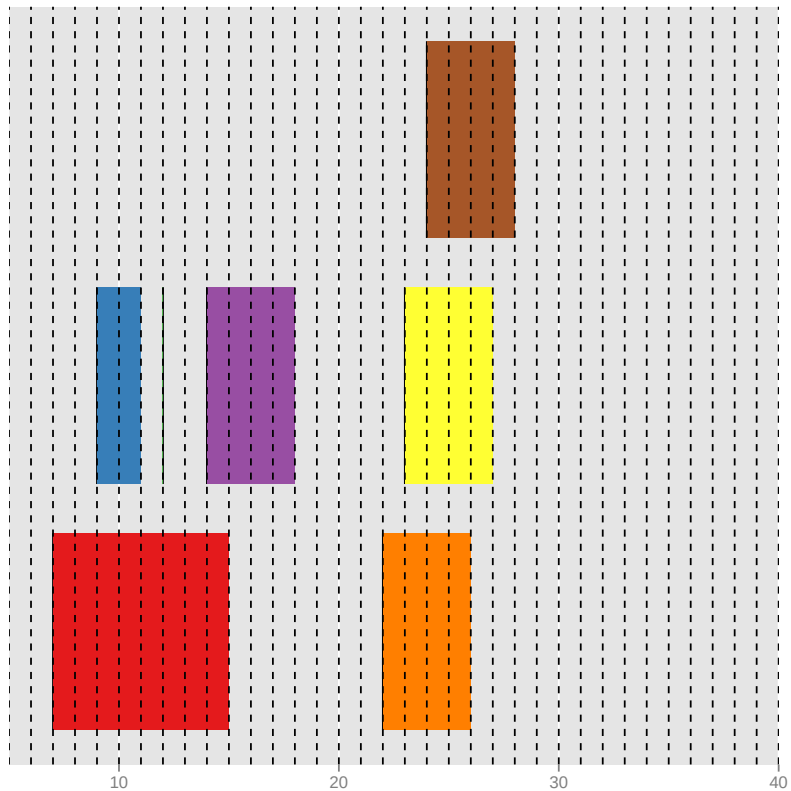
IRanges

- Genome is typically represented as linear sequence
- Ranges are an ordered set of consecutive integers defined by a start and end position
- start $\leq$ end
- Ranges are a common scaffold for many genomic analyses
- Ranges can be associated with genomic information (e.g. gene name) or data derived from analysis (e.g. counts)

Suppose we want to capture information on the following intervals

Start	End
7	15
9	11
12	12
14	18
22	26
23	27
24	28

```
## Object of class "ggbio"
```




```
## NULL
```

```
ir <- IRanges(start = c(7, 9, 12, 14, 22:24),
    end = c(15, 11, 12, 18, 26, 27, 28))
ir
```

```
## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     9  11     3
## [3]    12  12     1
## [4]    14  18     5
## [5]    22  26     5
## [6]    23  27     5
## [7]    24  28     5
```

```
start(ir)
```

```
## [1]  7  9 12 14 22 23 24
```

```
end(ir)
```

```
## [1] 15 11 12 18 26 27 28
```

```
width(ir)
```

```
## [1] 9 3 1 5 5 5 5
```

Ranges as vectors

```
ir

## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     9  11     3
## [3]    12  12     1
## [4]    14  18     5
## [5]    22  26     5
## [6]    23  27     5
## [7]    24  28     5
```

```
ir[1:2]

## IRanges of length 2
##     start end width
## [1]     7  15     9
## [2]     9  11     3

ir[width(ir) == 5]

## IRanges of length 4
##     start end width
## [1]    14  18     5
## [2]    22  26     5
## [3]    23  27     5
## [4]    24  28     5
```

Common Operations

- shift - move ranges by specified amount
- resize - change width, anchoring start, end or mid flank - Regions adjacent to start or end

See GRanges paper

Shifting

We could do this the long way

```
ir2 <- IRanges(start(ir) + 5, end(ir) + 5)
```

But a shortcut is provided by IRanges

```
identical(ir2, shift(ir, 5))
```

```
## [1] TRUE
```

Shifting

e.g. sliding windows

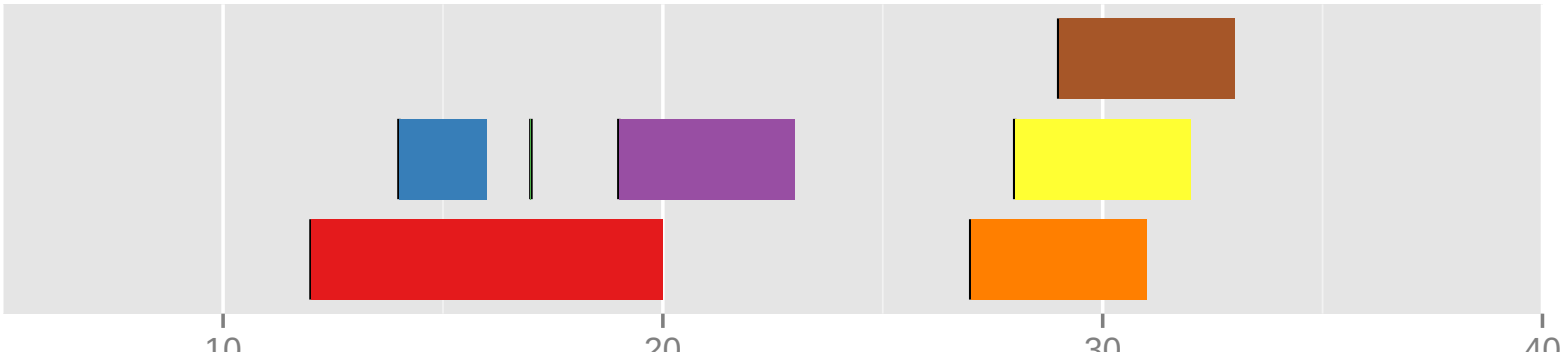
```
ir
```

```
## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     9  11     3
## [3]    12  12     1
## [4]    14  18     5
## [5]    22  26     5
## [6]    23  27     5
## [7]    24  28     5
```

```
shift(ir, 5)
```

```
## IRanges of length 7
##     start end width
## [1]    12  20     9
## [2]    14  16     3
## [3]    17  17     1
## [4]    19  23     5
## [5]    27  31     5
## [6]    28  32     5
## [7]    29  33     5
```

```
## Object of class "ggbio"
```



Shifting

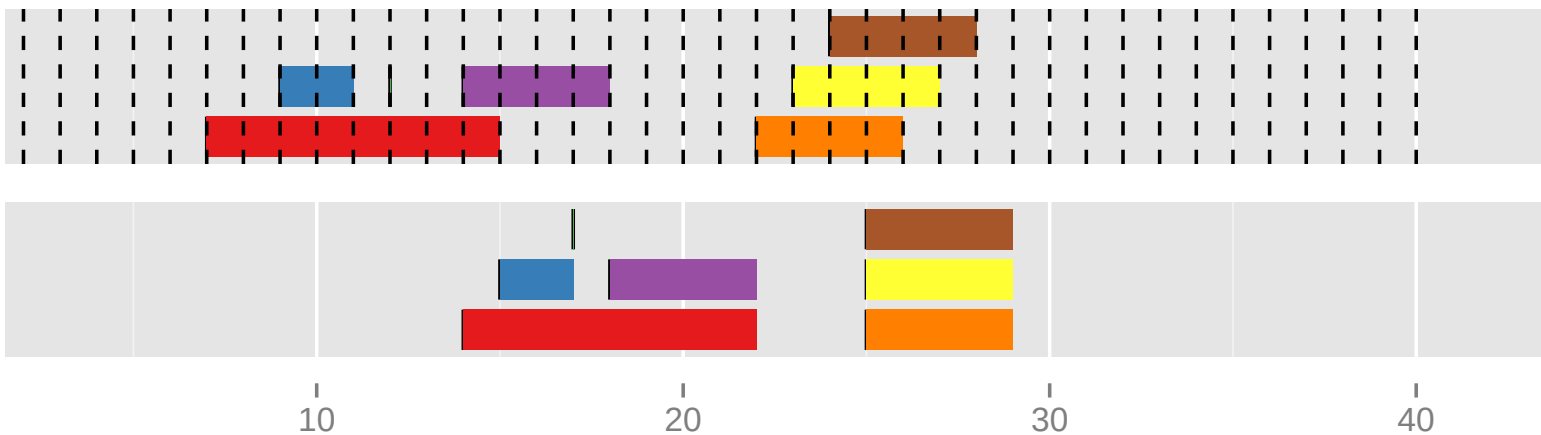
Size of shift doesnt need to be constant

```
ir
```

```
## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     9  11     3
## [3]    12  12     1
## [4]    14  18     5
## [5]    22  26     5
## [6]    23  27     5
## [7]    24  28     5
```

```
shift(ir, 7:1)
```

```
## IRanges of length 7
##     start end width
## [1]    14  22     9
## [2]    15  17     3
## [3]    17  17     1
## [4]    18  22     5
## [5]    25  29     5
## [6]    25  29     5
## [7]    25  29     5
```



Resize

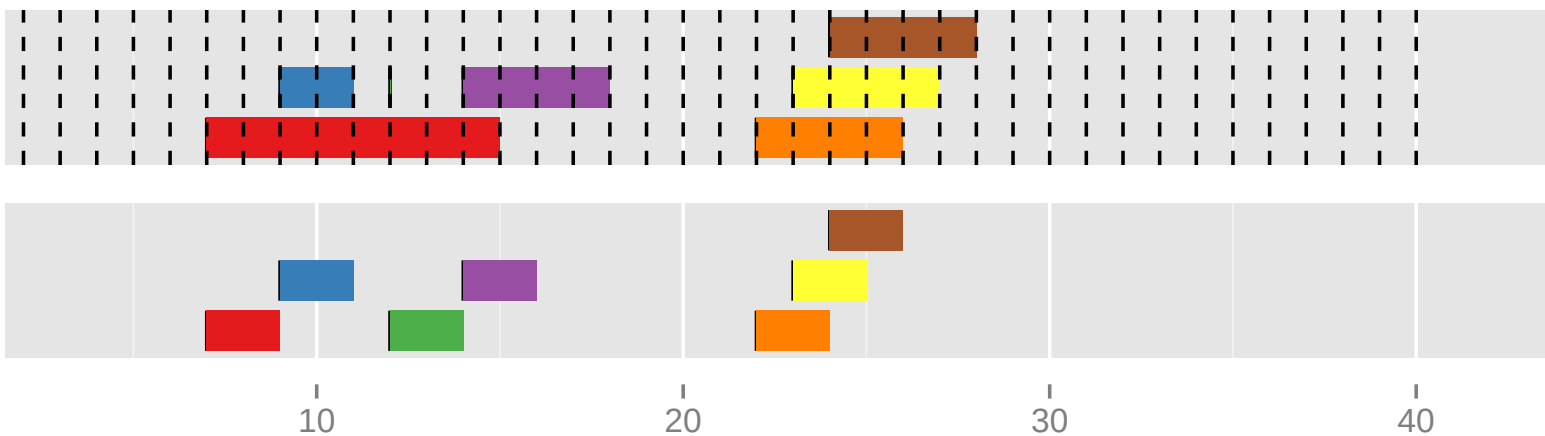
e.g. trimming reads

```
ir
```

```
## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     9  11     3
## [3]    12  12     1
## [4]    14  18     5
## [5]    22  26     5
## [6]    23  27     5
## [7]    24  28     5
```

```
resize(ir, 3)
```

```
## IRanges of length 7
##     start end width
## [1]     7   9     3
## [2]     9  11     3
## [3]    12  14     3
## [4]    14  16     3
## [5]    22  24     3
## [6]    23  25     3
## [7]    24  26     3
```



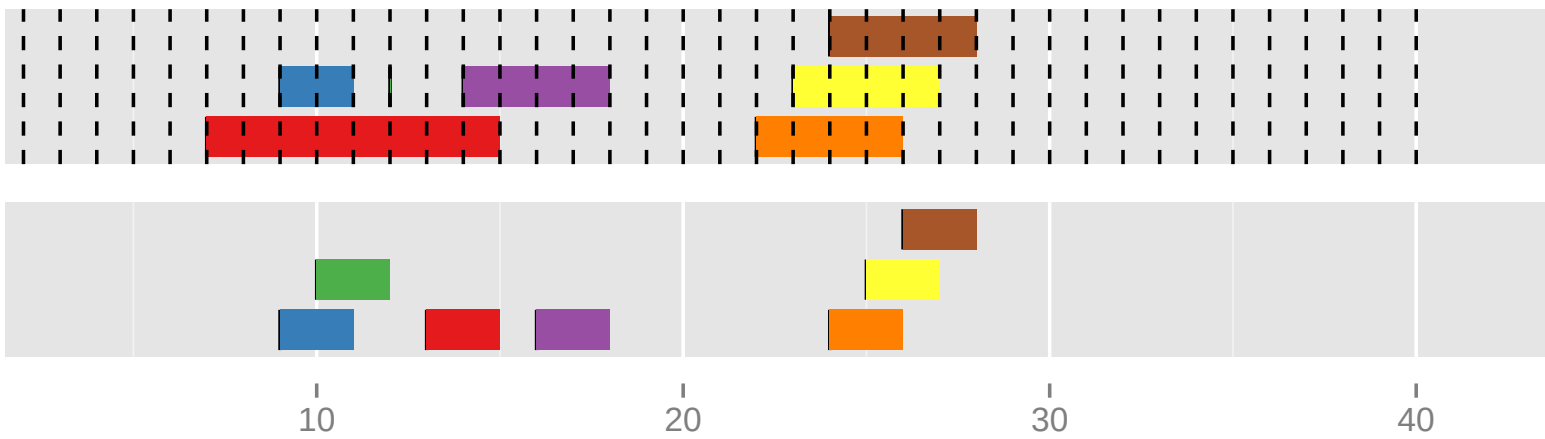
Resize

```
ir
```

```
## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     9  11     3
## [3]    12  12     1
## [4]    14  18     5
## [5]    22  26     5
## [6]    23  27     5
## [7]    24  28     5
```

```
resize(ir, 3, fix = "end")
```

```
## IRanges of length 7
##     start end width
## [1]    13  15     3
## [2]     9  11     3
## [3]    10  12     3
## [4]    16  18     3
## [5]    24  26     3
## [6]    25  27     3
## [7]    26  28     3
```



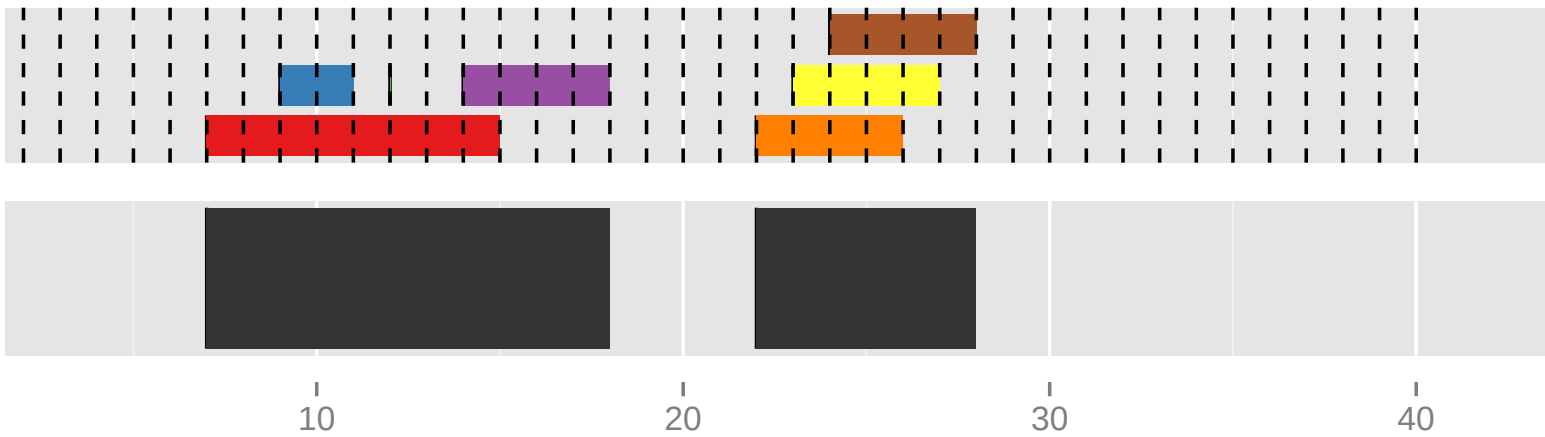
Reducing

```
ir
```

```
## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     9  11     3
## [3]    12  12     1
## [4]    14  18     5
## [5]    22  26     5
## [6]    23  27     5
## [7]    24  28     5
```

```
reduce(ir)
```

```
## IRanges of length 2
##     start end width
## [1]     7  18    12
## [2]    22  28     7
```



Gaps

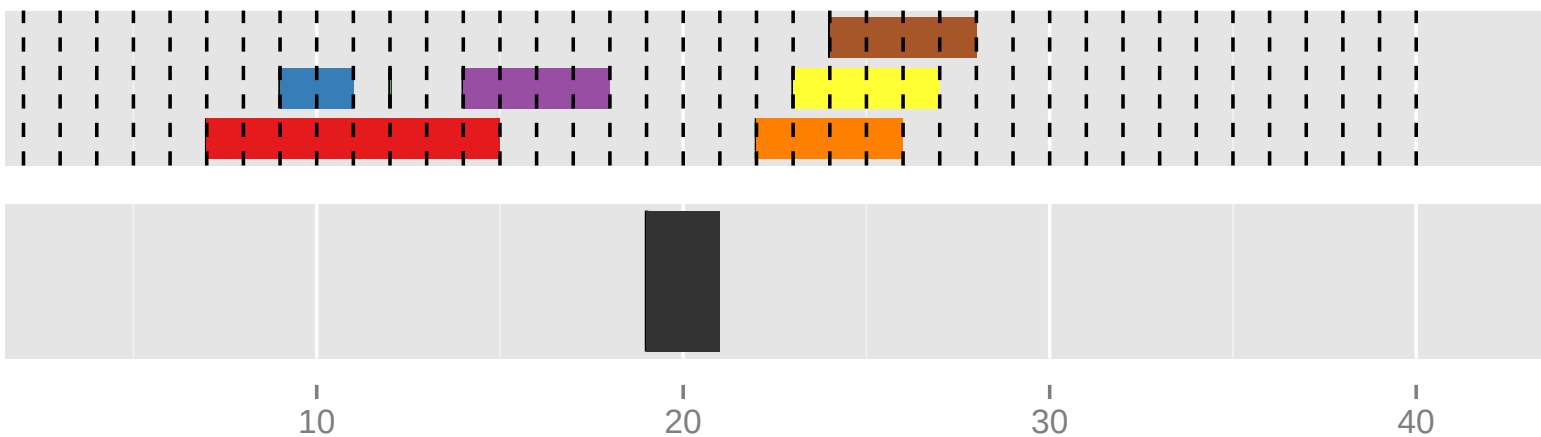
e.g. introns

```
ir
```

```
## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     9  11     3
## [3]    12  12     1
## [4]    14  18     5
## [5]    22  26     5
## [6]    23  27     5
## [7]    24  28     5
```

```
gaps(ir)
```

```
## IRanges of length 1
##     start end width
## [1]    19  21     3
```



Flanking

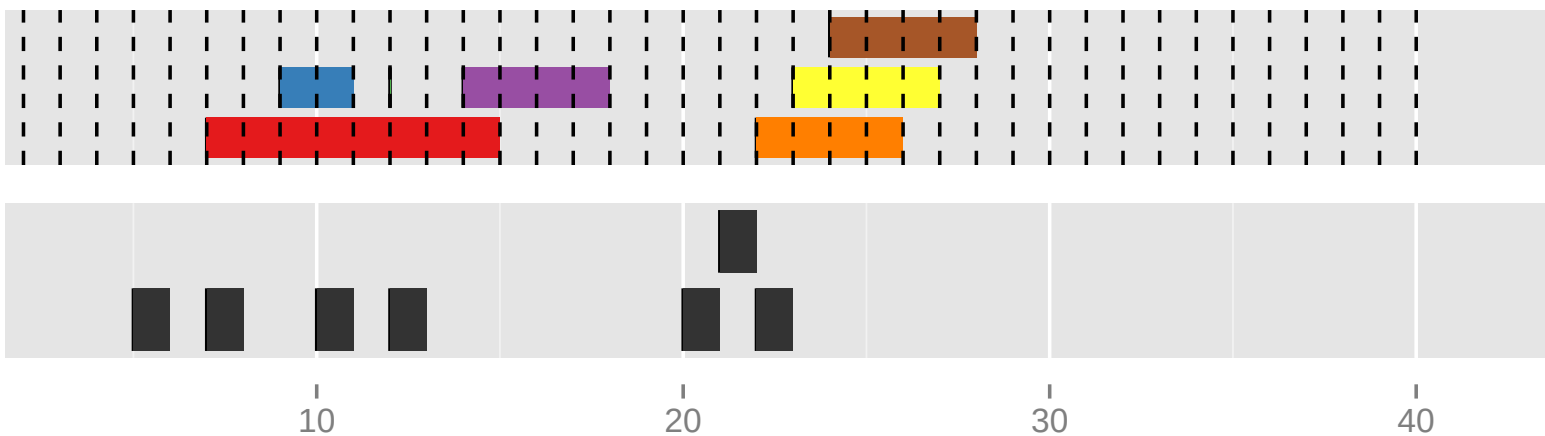
e.g. promoters

```
ir
```

```
## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     9  11     3
## [3]    12  12     1
## [4]    14  18     5
## [5]    22  26     5
## [6]    23  27     5
## [7]    24  28     5
```

```
flank(ir, 2)
```

```
## IRanges of length 7
##     start end width
## [1]     5   6     2
## [2]     7   8     2
## [3]    10  11     2
## [4]    12  13     2
## [5]    20  21     2
## [6]    21  22     2
## [7]    22  23     2
```

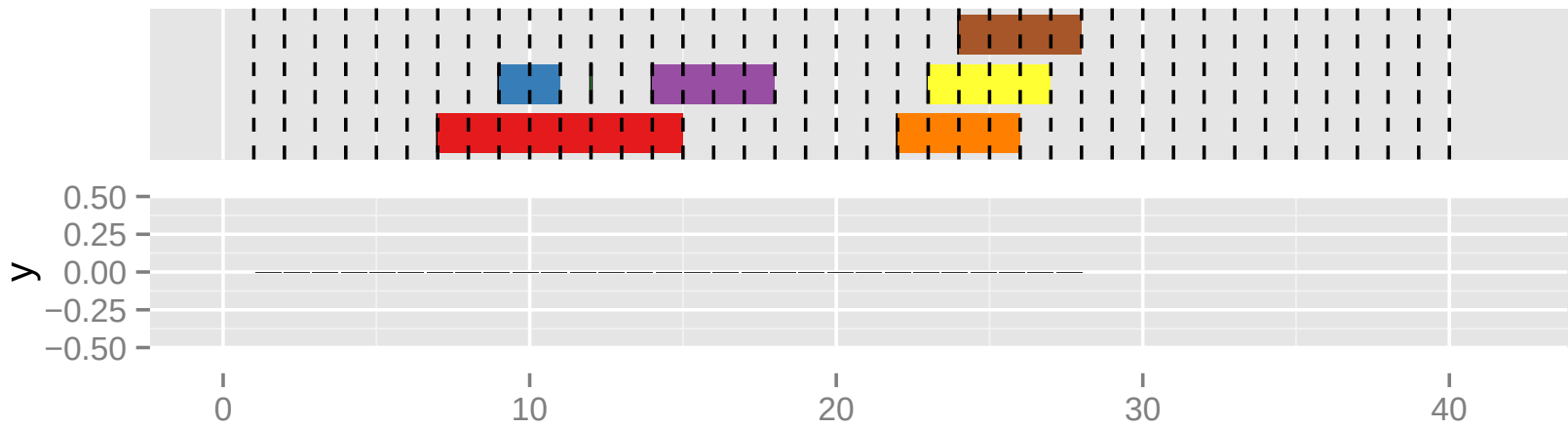


Coverage

coverage returns a Run Length Encoding - an efficient representation of repeated values

```
cvg <- coverage(ir)
cvg

## integer-Rle of length 28 with 12 runs
##   Lengths: 6 2 4 1 2 3 3 1 1 3 1 1
##   Values : 0 1 2 1 2 1 0 1 2 3 2 1

as.vector(cvg[1:12])

##  [1] 0 0 0 0 0 0 1 1 2 2 2 2
```

```
## Default use binwidth:  range/30
## Default use binwidth:  range/30
```



slice to get peaks

```
ranges(slice(coverage(ir), 2))
```

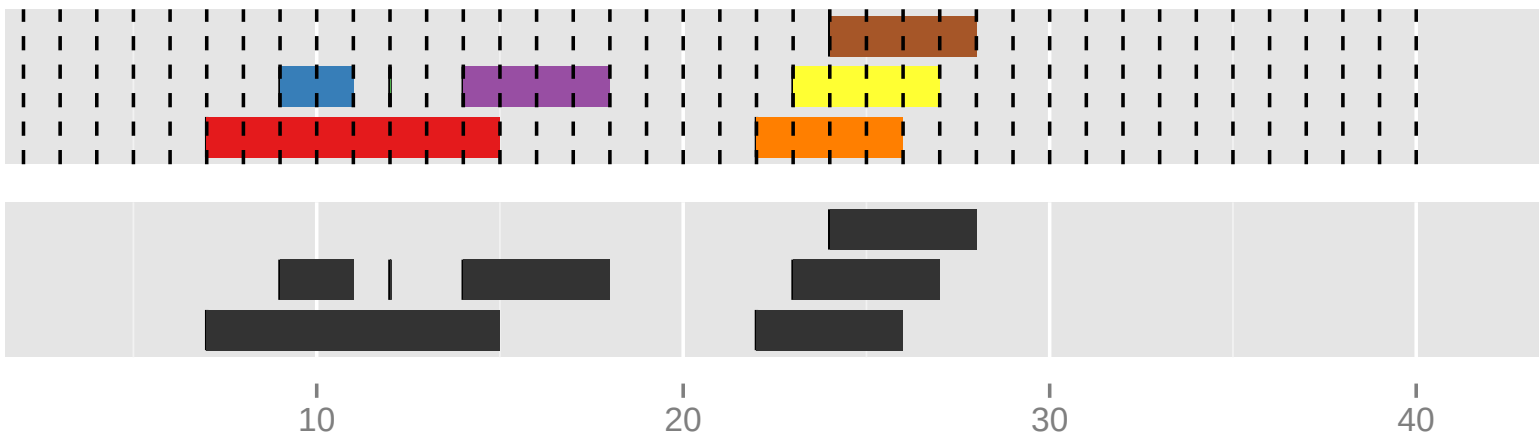
```
## IRanges of length 3
##     start end width
## [1]     9  12     4
## [2]    14  15     2
## [3]    23  27     5
```

Overlaps...

e.g. counting

```
ir3 <- IRanges(start = c(1, 14, 27), end = c(13,
    18, 30))
ir3
```

```
## IRanges of length 3
##     start end width
## [1]     1  13    13
## [2]    14  18     5
## [3]    27  30     4
```



Overlaps

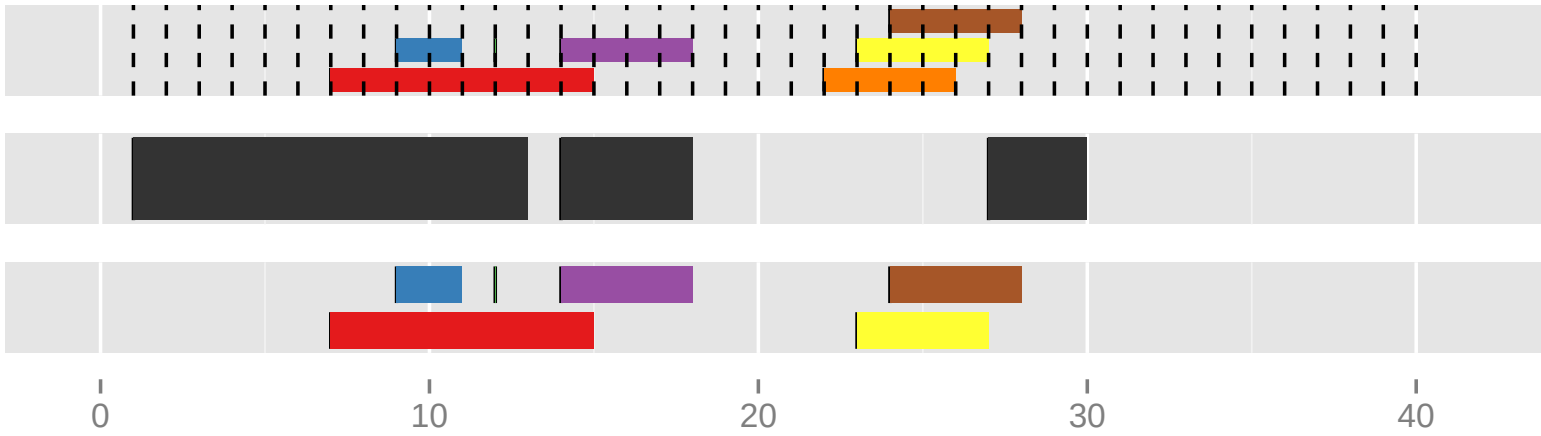
```
query <- ir
subject <- ir3
ov <- findOverlaps(query, subject)
ov

## Hits of length 7
## queryLength: 7
## subjectLength: 3
##   queryHits subjectHits
##    <integer>   <integer>
##  1         1           1
##  2         1           2
##  3         2           1
##  4         3           1
##  5         4           2
##  6         6           3
##  7         7           3
```

```
query[queryHits(ov)]
```

```
## IRanges of length 7
##     start end width
## [1]     7  15     9
## [2]     7  15     9
## [3]     9  11     3
## [4]    12  12     1
## [5]    14  18     5
## [6]    23  27     5
## [7]    24  28     5
```

```
subject[subjectHits(ov)]
```

```
## IRanges of length 7
##     start end width
## [1]     1  13    13
## [2]    14  18     5
## [3]     1  13    13
## [4]     1  13    13
## [5]    14  18     5
## [6]    27  30     4
## [7]    27  30     4
```



Can make the overlap more stringent

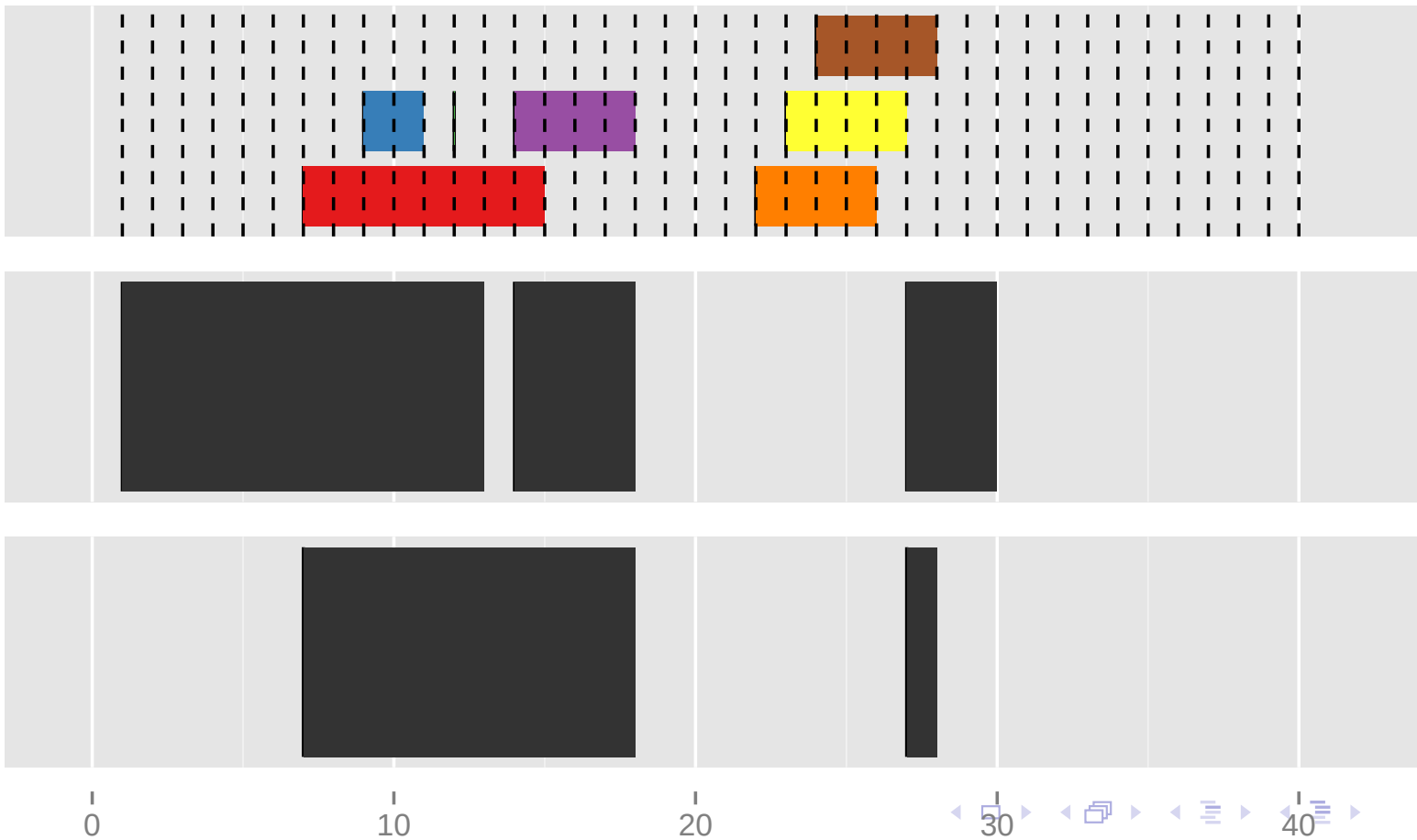
```
findOverlaps(query, subject, type = "within")
```

```
## Hits of length 3
## queryLength: 7
## subjectLength: 3
##   queryHits subjectHits
##    <integer>   <integer>
##  1         2           1
##  2         3           1
##  3         4           2
```

Intersection

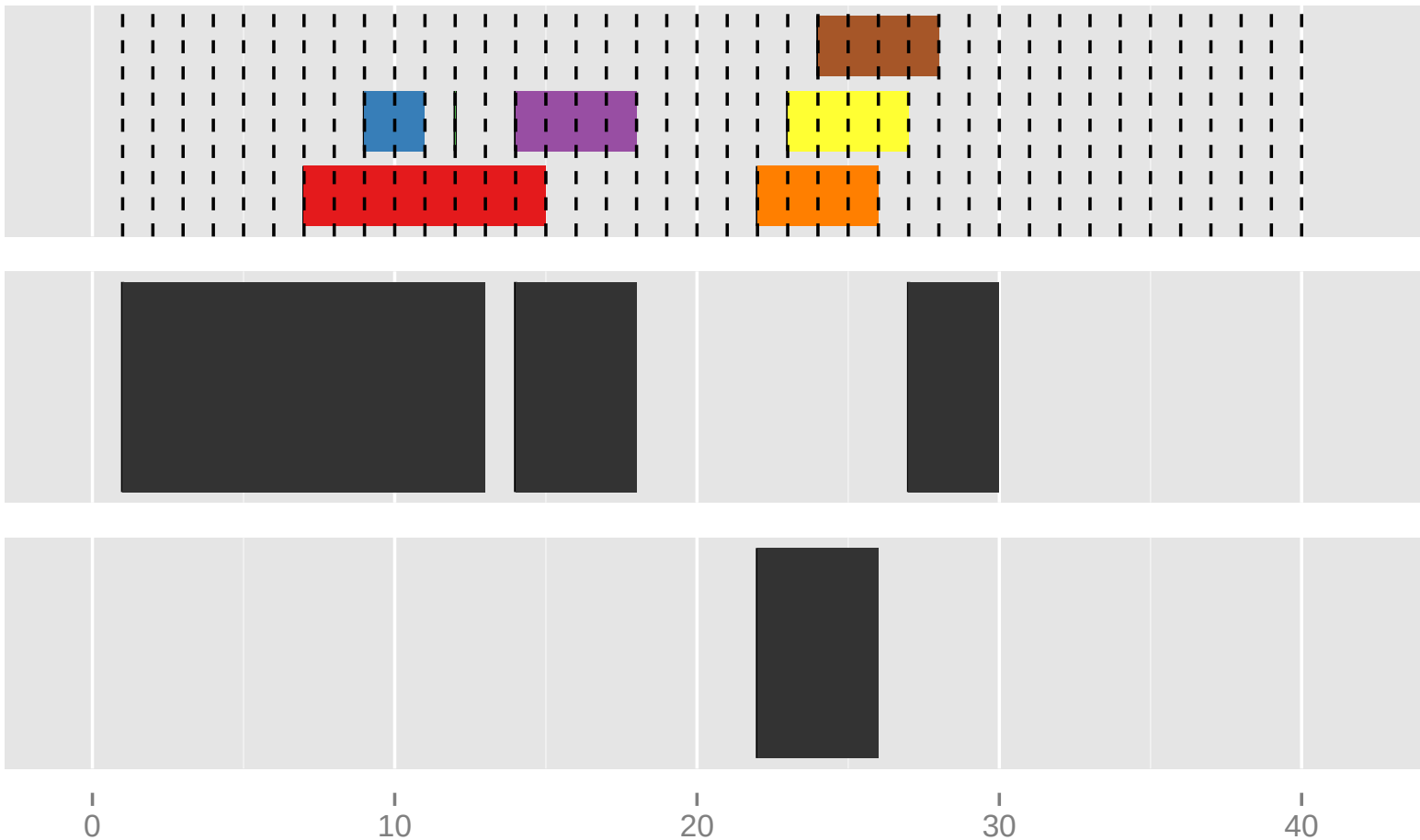
```
intersect(ir, ir3)
```

```
## IRanges of length 2
##     start end width
## [1]     7  18    12
## [2]    27  28     2
```



Subtraction

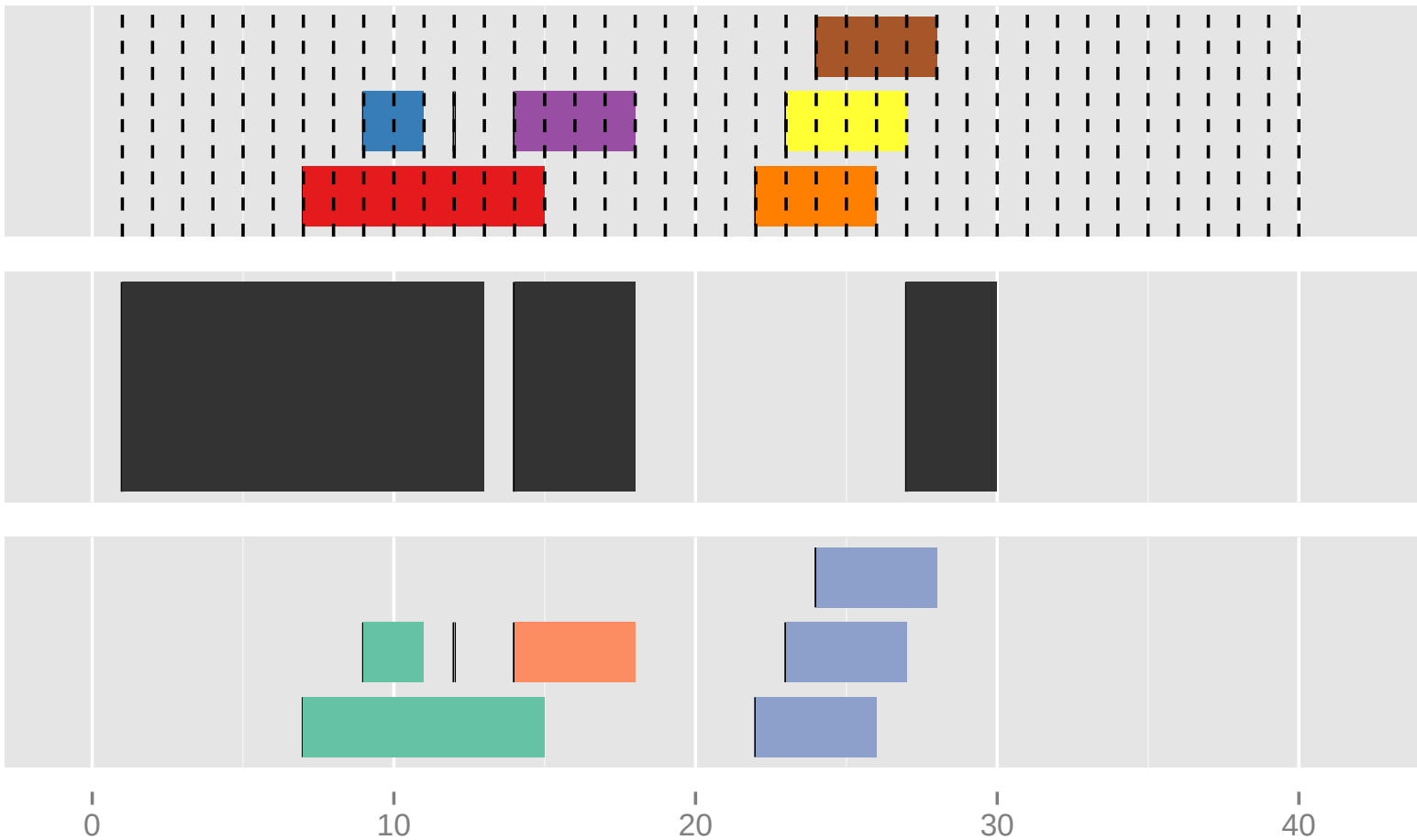
```
setdiff(ir, ir3)

## IRanges of length 1
##     start end width
## [1]    22  26     5
```



Nearest

e.g. Annotating to features...

```
nearest(ir, ir3)

## [1] 1 1 1 2 3 3 3
```



GRanges and Genomic Features

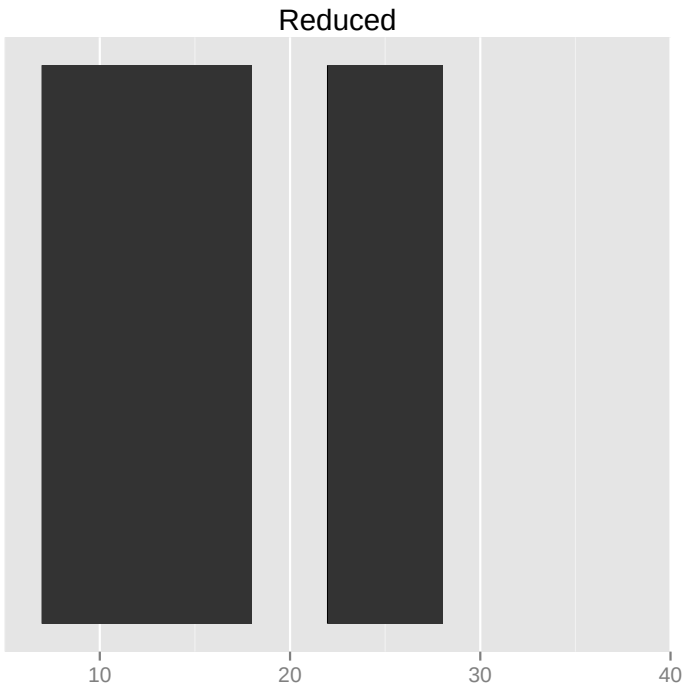
- GRanges provides infrastructure to manipulate genomic intervals in an efficient manner
- GenomicFeatures provides infrastructure to manipulate databases of genomic features (e.g. transcripts, exons)

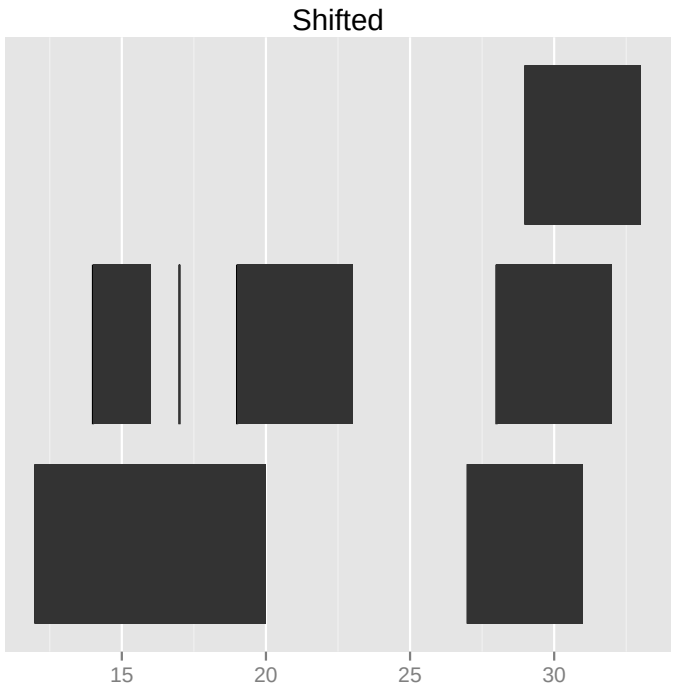
GRanges objects are IRanges with additional metadata (e.g. chromosome name, strand)

```
gr <- GRanges(rep("chr1", length(ir)), ranges = ir)
gr

## GRanges with 7 ranges and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]     chr1  [ 7, 15]      *
##   [2]     chr1  [ 9, 11]      *
##   [3]     chr1  [12, 12]      *
##   [4]     chr1  [14, 18]      *
##   [5]     chr1  [22, 26]      *
##   [6]     chr1  [23, 27]      *
##   [7]     chr1  [24, 28]      *
##   ---
##   seqlengths:
##    chr1
##      NA
```

Reducing

```
reduce(gr)
shift(gr, 5)
```



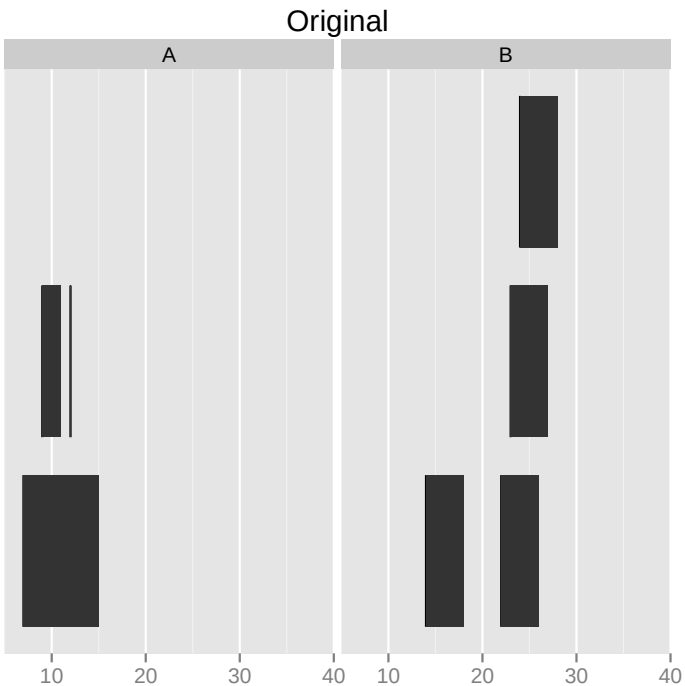


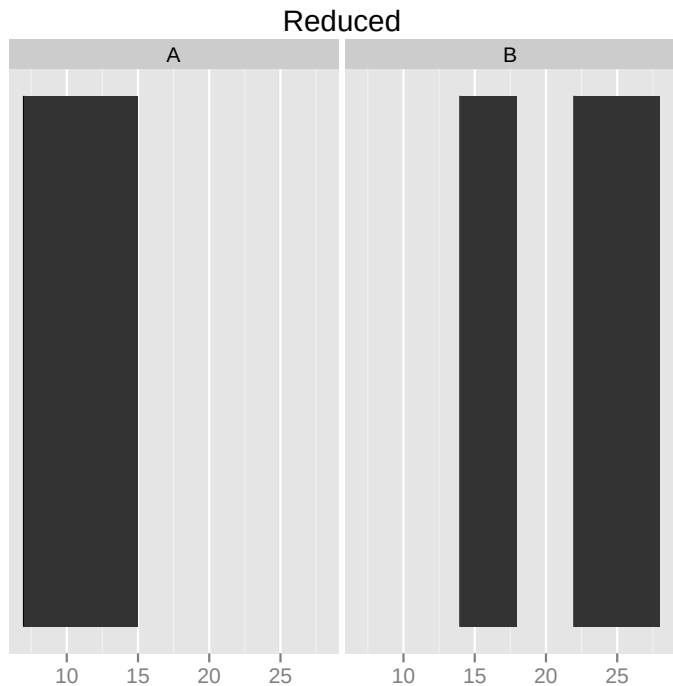
We now define a 'chromosome' for each range

```
gr <- GRanges(c("A", "A", "A", "B", "B",
    "B", "B"), ranges = ir)
gr

## GRanges with 7 ranges and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]        A  [ 7, 15]      *
##   [2]        A  [ 9, 11]      *
##   [3]        A  [12, 12]      *
##   [4]        B  [14, 18]      *
##   [5]        B  [22, 26]      *
##   [6]        B  [23, 27]      *
##   [7]        B  [24, 28]      *
##   ---
##   seqlengths:
##     A  B
##    NA NA
```

Reducing

```
reduce(gr)
```





```
gr
```

```
## GRanges with 7 ranges and 0 metad
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]        A  [ 7, 15]      *
##   [2]        A  [ 9, 11]      *
##   [3]        A  [12, 12]      *
##   [4]        B  [14, 18]      *
##   [5]        B  [22, 26]      *
##   [6]        B  [23, 27]      *
##   [7]        B  [24, 28]      *
##   ---
##   seqlengths:
##     A  B
##    NA NA
```

```
gr2 <- GRanges("A", ir3)
```

```
gr2
```

```
## GRanges with 3 ranges and 0 metada
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]        A  [ 1, 13]      *
##   [2]        A  [14, 18]      *
##   [3]        A  [27, 30]      *
##   ---
##   seqlengths:
##     A
##    NA
```

```
intersect(gr, gr2)
```

```
## GRanges with 1 range and 0 metadat
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]        A   [7, 15]      *
##   ---
##   seqlengths:
##     A  B
```

```
gr
```

```
## GRanges with 7 ranges and 0 metad
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]        A  [ 7, 15]      *
##   [2]        A  [ 9, 11]      *
##   [3]        A  [12, 12]      *
##   [4]        B  [14, 18]      *
##   [5]        B  [22, 26]      *
##   [6]        B  [23, 27]      *
##   [7]        B  [24, 28]      *
##   ---
##   seqlengths:
##     A  B
##    NA NA
```

```
gr3 <- GRanges("B", ir3)
```

```
gr3
```

```
## GRanges with 3 ranges and 0 metada
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]        B  [ 1, 13]      *
##   [2]        B  [14, 18]      *
##   [3]        B  [27, 30]      *
##   ---
##   seqlengths:
##     B
##    NA
```

```
intersect(gr, gr3)
```

```
## GRanges with 2 ranges and 0 metada
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]        B  [14, 18]      *
##   [2]        B  [27, 28]      *
##   ---
##   seqlengths:
```

Naming conventions

Sometimes (well, often) different naming conventions are used for chromosome names

```
seqlevels(gr)

## [1] "A" "B"



gr <- renameSeqlevels(gr, c(A = "chr1", B = "chr2"))
gr

## GRanges with 7 ranges and 0 metadata columns:
##       seqnames    ranges strand
##          <Rle> <IRanges>  <Rle>
##   [1]     chr1  [ 7, 15]      *
##   [2]     chr1  [ 9, 11]      *
##   [3]     chr1  [12, 12]      *
##   [4]     chr2  [14, 18]      *
##   [5]     chr2  [22, 26]      *
##   [6]     chr2  [23, 27]      *
##   [7]     chr2  [24, 28]      *
##   ---
```

Assigning metadata

GRanges objects can also have metadata associated with them

```
meta <- data.frame(SomeVals = runif(n = length(gr),
    100, 200), OtherVals = runif(n = length(gr),
    0, 1), SomeChars = sample(LETTERS, length(gr)))
values(gr) <- meta
```

```
gr[1:5]

## GRanges with 5 ranges and 3 metadata columns:
##       seqnames    ranges strand |
##          <Rle> <IRanges>  <Rle> |
##   [1]     chr1  [ 7, 15]      * |
##   [2]     chr1  [ 9, 11]      * |
##   [3]     chr1  [12, 12]      * |
##   [4]     chr2  [14, 18]      * |
##   [5]     chr2  [22, 26]      * |
##        SomeVals OtherVals SomeChars
##       <numeric> <numeric>  <factor>
##   [1]     145.6   0.81418         O
##   [2]     125.4   0.98168         G
##   [3]     105.7   0.03634         U
##   [4]     143.5   0.56741         W
##   [5]     177.1   0.40172         X
##   ---
##   seqlengths:
##    chr1 chr2
##      NA   NA
```

```
gr[values(gr)$SomeVals > 150]

## GRanges with 2 ranges and 3 metadata columns:
##       seqnames    ranges strand |
##          <Rle> <IRanges>  <Rle> |
##   [1]     chr2  [22, 26]      * |
##   [2]     chr2  [24, 28]      * |
##        SomeVals OtherVals SomeChars
##       <numeric> <numeric>  <factor>
##   [1]     177.1    0.4017         X
##   [2]     198.4    0.4033         N
##   ---
##   seqlengths:
##    chr1 chr2
##      NA   NA
```

```
gr[order(values(gr)$OtherVals)]

## GRanges with 7 ranges and 3 metadata columns:
##       seqnames    ranges strand |
##          <Rle> <IRanges>  <Rle> |
##   [1]     chr1  [12, 12]      * |
##   [2]     chr2  [22, 26]      * |
##   [3]     chr2  [24, 28]      * |
##   [4]     chr2  [23, 27]      * |
##   [5]     chr2  [14, 18]      * |
##   [6]     chr1  [ 7, 15]      * |
##   [7]     chr1  [ 9, 11]      * |
##        SomeVals OtherVals SomeChars
##       <numeric> <numeric>  <factor>
##   [1]     105.7   0.03634         U
##   [2]     177.1   0.40172         X
##   [3]     198.4   0.40332         N
##   [4]     112.9   0.45705         V
##   [5]     143.5   0.56741         W
##   [6]     145.6   0.81418         O
##   [7]     125.4   0.98168         G
##   ---
##   seqlengths:
##    chr1 chr2
```

GRanges can be split according to metadata

```
split(gr, values(gr)$SomeChars)

## GRangesList of length 7:
## $G
## GRanges with 1 range and 3 metadata columns:
##       seqnames    ranges strand |
##          <Rle> <IRanges>  <Rle> |
##   [1]     chr1   [9, 11]      * |
##        SomeVals OtherVals SomeChars
##       <numeric> <numeric>  <factor>
##   [1]     125.4    0.9817         G
##
## $N
## GRanges with 1 range and 3 metadata columns:
##       seqnames   ranges strand |
##   [1]     chr2 [24, 28]      * |
##       SomeVals OtherVals SomeChars
##   [1]    198.4    0.4033         N
##
## $O
## GRanges with 1 range and 3 metadata columns:
##       seqnames  ranges strand |
##   [1]     chr1 [7, 15]      * |
```

Summary values can be computed

```
lapply(split(gr, values(gr)$SomeChars), function(x) mean(values(x)$Some
```

```
## $G
## [1] 125.4
##
## $N
## [1] 198.4
##
## $O
## [1] 145.6
##
## $U
## [1] 105.7
##
## $V
## [1] 112.9
##
## $W
## [1] 143.5
##
## $X
## [1] 177.1
```

Reading alignments

We will assume that the sequencing reads have been aligned and that we are interested in processing the alignments. Rsamtools provides an interface for doing this. But we will use the readGappedAlignments tool in GenomicRanges which extracts the essential information from the bam file.

```
bam <- readGappedAlignments(mybam, use.name = TRUE)
```

The result looks a lot like a GRanges object. In fact, a lot of the same operations can be use

```
bam[1:4]

## GappedAlignments with 4 alignments and 0 metadata columns:
##                      seqnames strand       cigar    qwidth
##                         <Rle>  <Rle> <character> <integer>
##   SRR031715.1138209     chr4      +         37M        37
##    SRR031714.776678     chr4      -         37M        37
##   SRR031715.3258011     chr4      -         37M        37
##   SRR031715.4791418     chr4      +         37M        37
##                         start       end     width      ngap
##                      <integer> <integer> <integer> <integer>
##   SRR031715.1138209       169       205        37         0
##    SRR031714.776678       184       220        37         0
##   SRR031715.3258011       187       223        37         0
##   SRR031715.4791418       193       229        37         0
##   ---
##   seqlengths:
##       chr2L    chr2R    chr3L ...      chrM     chrX   chrYHet
##    23011544 21146708 24543557 ...     19517 22422827    347038
```

Querying alignments

```
table(strand(bam))

##
##     +     -     *
## 84871 90475     0

summary(width(bam))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      37      37      37      59      37   19400

range(start(bam))

## [1]     169 1351760

cigar(bam)[1:10]

##  [1] "37M" "37M" "37M" "37M" "37M" "37M" "37M" "37M" "37M"
## [10] "37M"
```

Manipulation of reads

aligned reads can be manipulated using functions from IRanges

```
shift(ranges(bam), 10)

## IRanges of length 175346
##              start     end width              names
## [1]            179     215    37 SRR031715.1138209
## [2]            194     230    37  SRR031714.776678
## [3]            197     233    37 SRR031715.3258011
## [4]            203     239    37 SRR031715.4791418
## [5]            336     372    37 SRR031715.1138209
## ...            ...     ...   ...                ...
## [175342] 1349718 1349754    37 SRR031714.1650928
## [175343] 1349848 1349884    37 SRR031714.1650928
## [175344] 1351650 1351686    37 SRR031714.5192891
## [175345] 1351650 1351686    37 SRR031715.2351056
## [175346] 1351770 1351806    37  SRR031714.864195
```

Manipulation of reads

aligned reads can be manipulated using functions from IRanges

```
flank(ranges(bam), 100, both = T)

## IRanges of length 175346
##              start     end width              names
## [1]             69     268   200 SRR031715.1138209
## [2]             84     283   200  SRR031714.776678
## [3]             87     286   200 SRR031715.3258011
## [4]             93     292   200 SRR031715.4791418
## [5]            226     425   200 SRR031715.1138209
## ...            ...     ...   ...                ...
## [175342] 1349608 1349807   200 SRR031714.1650928
## [175343] 1349738 1349937   200 SRR031714.1650928
## [175344] 1351540 1351739   200 SRR031714.5192891
## [175345] 1351540 1351739   200 SRR031715.2351056
## [175346] 1351660 1351859   200  SRR031714.864195

coverage(ranges(bam))

## integer-Rle of length 1351796 with 104286 runs
##   Lengths:  168   15    3    6   13 ... 1765   37   83   37
```

Region subset - the naive way

```
bam[start(bam) < 20100 & end(bam) > 20000, ]

## GappedAlignments with 14 alignments and 0 metadata columns:
##                     seqnames strand       cigar    qwidth
##                        <Rle>  <Rle> <character> <integer>
##   SRR031714.4100693     chr4      +  31M7704N6M        37
##   SRR031715.5248298     chr4      +  29M7704N8M        37
##   SRR031714.4092638     chr4      -         37M        37
##   SRR031714.4275537     chr4      -         37M        37
##   SRR031715.1315719     chr4      -         37M        37
##                 ...      ...    ...         ...       ...
##   SRR031715.3358559     chr4      +         37M        37
##   SRR031715.4831822     chr4      +         37M        37
##   SRR031715.4459351     chr4      +         37M        37
##   SRR031715.2716654     chr4      -         37M        37
##   SRR031715.1552693     chr4      +         37M        37
##                       start       end     width      ngap
##                   <integer> <integer> <integer> <integer>
##   SRR031714.4100693     13660     21400      7741         1
##   SRR031715.5248298     13662     21402      7741         1
##   SRR031714.4092638     19968     20004        37         0
##   SRR031714.4275537     19968     20004        37         0
```

The smart way

```
gr <- GRanges("chr4", IRanges(start = 20000, end = 20100))
gr

## GRanges with 1 range and 0 metadata columns:
##       seqnames          ranges strand
##          <Rle>       <IRanges>  <Rle>
##   [1]     chr4 [20000, 20100]      *
##   ---
##   seqlengths:
##    chr4
##      NA
```

```
findOverlaps(gr, bam)

## Hits of length 12
## queryLength: 1
## subjectLength: 175346
##     queryHits subjectHits
##      <integer>   <integer>
##  1           1        6699
##  2           1        6700
##  3           1        6701
##  4           1        6702
##  5           1        6703
##  ...       ...         ...
##  8           1        6706
##  9           1        6707
##  10          1        6708
##  11          1        6709
##  12          1        6710
```

```
bam[subjectHits(findOverlaps(gr, bam))]

## GappedAlignments with 12 alignments and 0 metadata columns:
##                     seqnames strand       cigar    qwidth
##                        <Rle>  <Rle> <character> <integer>
##   SRR031714.4092638     chr4      -         37M        37
##   SRR031714.4275537     chr4      -         37M        37
##   SRR031715.1315719     chr4      -         37M        37
##   SRR031715.1502533     chr4      -         37M        37
##    SRR031714.336402     chr4      -         37M        37
##                 ...      ...    ...         ...       ...
##   SRR031715.3358559     chr4      +         37M        37
##   SRR031715.4831822     chr4      +         37M        37
##   SRR031715.4459351     chr4      +         37M        37
##   SRR031715.2716654     chr4      -         37M        37
##   SRR031715.1552693     chr4      +         37M        37
##                         start       end     width      ngap
##                     <integer> <integer> <integer> <integer>
##   SRR031714.4092638     19968     20004        37         0
##   SRR031714.4275537     19968     20004        37         0
##   SRR031715.1315719     19968     20004        37         0
##   SRR031715.1502533     19968     20004        37         0
##    SRR031714.336402     19971     20007        37         0
##                 ...       ...       ...       ...       ...
```

Alternative

```
bam.sub <- bam[bam %over% gr]
bam.sub

## GappedAlignments with 12 alignments and 0 metadata columns:
##                     seqnames strand       cigar    qwidth
##                        <Rle>  <Rle> <character> <integer>
##   SRR031714.4092638     chr4      -         37M        37
##   SRR031714.4275537     chr4      -         37M        37
##   SRR031715.1315719     chr4      -         37M        37
##   SRR031715.1502533     chr4      -         37M        37
##    SRR031714.336402     chr4      -         37M        37
##                 ...      ...    ...         ...       ...
##   SRR031715.3358559     chr4      +         37M        37
##   SRR031715.4831822     chr4      +         37M        37
##   SRR031715.4459351     chr4      +         37M        37
##   SRR031715.2716654     chr4      -         37M        37
##   SRR031715.1552693     chr4      +         37M        37
##                       start       end     width      ngap
##                   <integer> <integer> <integer> <integer>
##   SRR031714.4092638     19968     20004        37         0
##   SRR031714.4275537     19968     20004        37         0
##   SRR031715.1315719     19968     20004        37         0
```

Read subset of regions

quicker still, we can get the reads directly from the bam file. The region to be read can be specified using the param argument.

```
system.time(bam.sub <- readGappedAlignments(file = mybam, use.names = T
    param = ScanBamParam(which = gr)))

##    user  system elapsed
##   0.085   0.011   0.122
```

Recap

- Ranges can be used to represent continuous regions
- GRanges are special ranges with extra biological context
- GRanges can be manipulated, compared, overlapped with each other
- Aligned reads can be represented by Ranges
- Genome and sequencing reads can be represented efficiently by Biostrings
- The genome can also be accessed using Ranges

To be continued...

- Tomorrow, we will go through an RNA-seq analysis workflow
- We will then look how to relate the regions of interest back to the genome

This talk was brought to you by...

```
sessionInfo()
```

```
## R version 3.0.1 (2013-05-16)
## Platform: x86_64-apple-darwin10.8.0 (64-bit)
##
## locale:
## [1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8
##
## attached base packages:
## [1] grid      parallel  stats     graphics  grDevices
## [6] utils     datasets  methods   base
##
## other attached packages:
##  [1] gridExtra_0.9.1
##  [2] reshape_0.8.4
##  [3] plyr_1.8
##  [4] BSgenome.Hsapiens.UCSC.hg19_1.3.19
##  [5] BiocInstaller_1.10.4
##  [6] TxDb.Hsapiens.UCSC.hg19.knownGene_2.9.2
##  [7] BSgenome.Hsapiens.UCSC.hg18_1.3.19
##  [8] Gviz_1.4.4
```