Bioconductor packages for short read analyses

Alex Sánchez

Unitat d'Estadística i Bioinformàtica (UEB)

June 8, 2014

Table of Contents

Foreword

- The “core” packages for integrating NGS data anlysis represents a massive structure.
- It is under very active development and often different ways exist to achieve one goal.
  - *e.g RangedData vs. GRanges*
- The trunk of this core starts to reach maturity and redundant branches might be pruned.

ALM

- Introduce all the necessary packages to perfom the QA and the pre-processin of NGS rawdata:
    - **biomRt**
    - **rtracklayer**
    - **Biostrings**
    - **BSgenome**
    - **GenomicFeatures**
    - **GenomicRanges**
    - **IRanges**
    - **Rsamtools**
    - **ShortRead**

Before we “really” start: the Classes in R

- Two kinds: S3 and S4
  - S3 are old and informal, setting the class attribute is enough to “convert” an object into a class
  - S4 is an attempt at making R more object oriented
    - they have specific definitions
    - they define “fields” called “**slots**”
    - they can inherit and be inherited from
    - they can have prototypes, validators
    - they can be virtual
    - etc.
  - Most of the classes described here are of S4 type, except when backward compatibility with the R core required otherwise
  - More information can be foun in the R help page: ?classRepresentation

Methods to browse S4 classes

(Load the IRanges library to run the following example)

```
> require(IRanges)
> ?Classes
> ?Methods
> getClass("RleList")

Virtual Class "RleList" [package "IRanges"]

Slots:

Name:      elementType elementMetadata        metadata
Class:       character DataTableORNULL            list

Extends:
Class "AtomicList", directly
Class "List", by class "AtomicList", distance 2
Class "Vector", by class "AtomicList", distance 3
Class "Annotated", by class "AtomicList", distance 4

Known Subclasses: "RleViews", "CompressedRleList", "SimpleRleList"
```

Methods to browse S4 classes

```
> names(completeSubclasses(getClass("RleList")))

[1] "RleViews"          "CompressedRleList" "SimpleRleList"

> head(showMethods(classes="RleList",printTo=FALSE))

[1] ""                               "Function \"activeView\":"
[3] " <not an S4 generic function>" ""
[5] "Function \"activeView<-\":"     " <not an S4 generic function>"

> showMethods("values",includeDefs=TRUE)

Function: values (package IRanges)
x="QASummary"
function (x, ...)
{
    x@values
}


x="RangedData"
function (x, ...)
{
    .local <- function (x)
    x@values
    .local(x, ...)
}


x="SummarizedExperiment"
function (x, ...)
{
    values(rowData(x), ...)
}
```

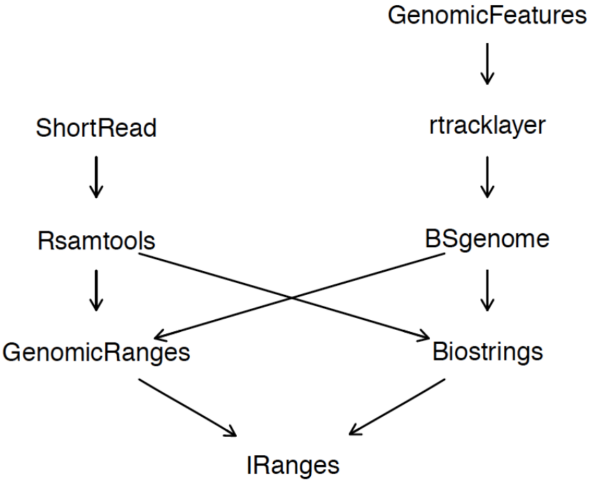
Packages dependencies

- Sometimes packages define the same function resulting in one of the function to be inaccessible anymore.
- When this happens, one needs to contact the packages authors for them to find an appropiate solution
- In the meanwhile, the hack described on the next slides might help
- load the GenomicRanges and the genomeIntervals in that order

Packages dependencies

- For the purpose og the example it is not necessary to understand the actual obects taht are created. We'll come back to them later.
- Create the necessary object
  - grngs $< -$ GRanges (seqnames=c(“chr1”,“chr2”,“ch31”), ranges=IRanges(start=c(3,4,1),end=c(7,5,3)),s trand=c(“+”,“+”,“-”), seqlegths=c(“chr1”=24,“chr2”=18))

Bottom - up approach



GenomicFeatures

ShortRead

rtracklayer

Rsamtools

BSgenome

GenomicRanges

Biostrings

IRanges

Infrastructure package

- **IRanges**
  - Long sequences, compressed and pointer referenced
  - **Views** on long sequences
  - Integer overlap tppñs; e.g. interval overlap
  - Used to define genomic intervals (i.e. RangedData)
- **GenomicRanges** Recent
  - IRanges extension
  - Adds discontiguous genomic interval sets (useful for gapped alignments)
- **genomeIntervals** Not Core
  - Very similar to IRanges
  - Extremely efficient at interval calculations; e.g. interval overlap

Infrastructure Views

- Issue
  - DNA sequences can be very large (think of the human genome)
  - Duplicating them in memory is contra-efficient
- Therefore the views!
  - Views is yet another IRanges class
    - a virtual class for storin set of views (pointers) on a single Sequence object
    - avaliable as RleViews, XStringViews, XIntegerViews, XStringSetViews, etc.
    - it stores the sequence using a “pass-by-reference” semantic and associatesranges to select the subsequences

Infrastructure Running Length Encodings (RLEs)

- Issue
  - Again, memory is the limit. holding a coverage vector at a single bp resolution is inefficient.
- Therefore the concept of RLEs
  - a common compression technique for piecewise constant data
    - 0 0 0 1 1 1 2 2 3 3 3 ... can be compressed in
    - 0(3), 1(3), 2(2), 3(3),...
  - it couples values e.g. 0 with a run length i.e. 3
  - Can be partitioned into RleList, e.g. for storing the coverage of different chromosomes

Infrastructure methods

- get the metods for the Rle S4 class
  - f.list< −showMethods(classes=“Rle”, printTo=FALSE)
- process the result to extract the function name
  - sapply(strsplit(f.list[grep(“Function”,f.list,”), function(l)gsub(’ ’,“,l[[2]])

```
> f.list<-showMethods(classes="Rle", printTo=FALSE)
> length(sapply(strsplit( f.list[grep("Function",
+   f.list)],' '),function(l){gsub('\"|:','',l[[2]])}))

[1] 340
```

Infrastructure methods, some examples

- **Arithmetic** $+, -, *,^{\prime} \%\%, \%/\%, /$
- **Compare** $==, >, <, != , <=, >=$
- **Logic** $\&, |$
- **Math** abs, sign, sqrt, ceiling, floor, trunc, cummax, cummin, cumprod, cumsum, log, log10, log2, log1p, acos, acosh, asin, asinh,...
- **Math2** round,signif
- **Summmary** max, min, range, prod, sum, any, all

Looks intimidating

- Still the point is:
  - whenever you think about a functionality, it probably already exists.

Example 1: coverage

- Coverage calculation

```
> require("ShortRead")
> fl<-system.file("extdata","GSM424494_wt_G2_orc_chip_rep1_S288C_14.mapview.txt.gz",
+                  package="EatonEtAlChiPseq")
> aln<-readAligned(fl,type="MAQMapview")
> cover<-coverage(aln);cover
> cover[["S288C_14"]]
> head(runValue(cover[["S288C_14"]]))
> as.integer(cover[["S288C_14"]])
> smoothCover<-round(runmean(cover,75,endrule="constant"))
> class(smoothCover)
> smoothCover
```

Example 2: slice

- Finding wide regions with elevated coverage

```
> islands<-slice(smoothCover,lower=10)
> islandsWithWidePeaks<- islands[vienMaxs(islands)>=20L &width(islands)>=500L]
> islandsWithWidePeaks
```

What comes on top of IRanges

- We've "covered" **IRanges** and it's low level capabilities.
- Still, High Throughput methods in biology, especially sequencing, are more about sequenes than maths.
- Therefor the **Biostrings** package, build on top of IRanges

Biostrings
IRanges

Biostrings

- All the classes in that package derivers from the XString class

```
> require(Biostrings)
> getClass("XString")

Virtual Class "XString" [package "Biostrings"]

Slots:

Name:           shared          offset          length elementMetadata
Class:       SharedRaw         integer         integer DataTableORNULL

Name:         metadata
Class:            list

Extends:
Class "XRaw", directly
Class "XVector", by class "XRaw", distance 2
Class "Vector", by class "XRaw", distance 3
Class "Annotated", by class "XRaw", distance 4

Known Subclasses: "BString", "DNAString", "RNAString", "AAString"
```

- There are 4 subclasses:
  - **BString**: store strings without alphabet
  - **DNAString**: store strings with an DNA alphabet
  - **RNAString**: store strings with an RNA alphabet
  - **AAString**: store strings with an Amino Acid alphabet

An DNAString example

- The **Biostring** package contains many example datasets

```
> data(package="Biostrings")
> data(yeastSEQCHR1)
> class(yeastSEQCHR1)

[1] "character"

> nchar(yeastSEQCHR1)

[1] 230208

> DNAString(yeastSEQCHR1)

  230208-letter "DNAString" instance
seq: CCACACCACACCCACACACCCACACACCACACCACA...GGTGTGTGGGTGTGGTGTGGGTGTGGTGTGTGTGGG
```

- The obtained DNAString is defined by the DNA alphabet

```
> alphabet(DNAString(yeastSEQCHR1))

 [1] "A" "C" "G" "T" "M" "R" "W" "S" "Y" "K" "V" "H" "D" "B" "N" "-" "+" "."
```

The alphabets

- The **Biostring** package implements the possible alphabets

```
> GENETIC_CODE
TTT TTC TTA TTG TCT TCC TCA TCG TAT TAC TAA TAG TGT TGC TGA TGG CTT CTC CTA CTG
"F" "F" "L" "L" "S" "S" "S" "S" "Y" "Y" "*" "*" "C" "C" "*" "W" "L" "L" "L" "L"
CCT CCC CCA CCG CAT CAC CAA CAG CGT CGC CGA CGG ATT ATC ATA ATG ACT ACC ACA ACG
"P" "P" "P" "P" "H" "H" "Q" "Q" "R" "R" "R" "R" "I" "I" "I" "M" "T" "T" "T" "T"
AAT AAC AAA AAG AGT AGC AGA AGG GTT GTC GTA GTG GCT GCC GCA GCG GAT GAC GAA GAG
"N" "N" "K" "K" "S" "S" "R" "R" "V" "V" "V" "V" "A" "A" "A" "A" "D" "D" "E" "E"
GGT GGC GGA GGG
"G" "G" "G" "G"
> AMINO_ACID_CODE
    A     R     N     D     C     Q     E     G     H     I     L     K     M
"Ala" "Arg" "Asn" "Asp" "Cys" "Gln" "Glu" "Gly" "His" "Ile" "Leu" "Lys" "Met"
    F     P     S     T     W     Y     V     U     O     B     Z     X
"Phe" "Pro" "Ser" "Thr" "Trp" "Tyr" "Val" "Sec" "Pyl" "Asx" "Glx" "Xaa"
> RNA_GENETIC_CODE
UUU UUC UUA UUG UCU UCC UCA UCG UAU UAC UAA UAG UGU UGC UGA UGG CUU CUC CUA CUG
"F" "F" "L" "L" "S" "S" "S" "S" "Y" "Y" "*" "*" "C" "C" "*" "W" "L" "L" "L" "L"
CCU CCC CCA CCG CAU CAC CAA CAG CGU CGC CGA CGG AUU AUC AUA AUG ACU ACC ACA ACG
"P" "P" "P" "P" "H" "H" "Q" "Q" "R" "R" "R" "R" "I" "I" "I" "M" "T" "T" "T" "T"
AAU AAC AAA AAG AGU AGC AGA AGG GUU GUC GUA GUG GCU GCC GCA GCG GAU GAC GAA GAG
"N" "N" "K" "K" "S" "S" "R" "R" "V" "V" "V" "V" "A" "A" "A" "A" "D" "D" "E" "E"
GGU GGC GGA GGG
"G" "G" "G" "G"
> IUPAC_CODE_MAP
     A      C      G      T      M      R      W      S      Y      K      V
   "A"    "C"    "G"    "T"   "AC"   "AG"   "AT"   "CG"   "CT"   "GT"  "ACG"
     H      D      B      N
 "ACT"  "AGT"  "CGT" "ACGT"
```

Set of Strings

- XStrings and subclasses instanves can all be grouped into Sets

```
> names(completeSubclasses(getClass("XStringSet")))

 [1] "BStringSet"                 "DNAStringSet"
 [3] "RNAStringSet"               "AAStringSet"
 [5] "QualityScaledXStringSet"    "XStringQuality"
 [7] "QualityScaledBStringSet"    "QualityScaledDNAStringSet"
 [9] "QualityScaledRNAStringSet" "QualityScaledAAStringSet"
[11] "QualityScaledBStringSet"    "QualityScaledDNAStringSet"
[13] "QualityScaledRNAStringSet" "QualityScaledAAStringSet"
[15] "PhredQuality"               "SolexaQuality"
[17] "IlluminaQuality"
```

- Again, there are data examples withun the **Biostring** package to play with

```
> data(srPhiX174)
> class(srPhiX174)

[1] "DNAStringSet"
attr(,"package")
[1] "Biostrings"
```

```
> head(srPhiX174)

  A DNAStringSet instance of length 6
    width seq
[1]    35 GTTATTATACCGTCAAGGACTGTGTGACTATTGAC
[2]    35 GGTGGTTATTATACCGTCAAGGACTGTGTGACTAT
[3]    35 TACCGTCAAGGACTGTGTGACTATTGACGTCCTTC
[4]    35 GTACGCCGGGCAATAATGTTTATGTTGGTTTCATG
[5]    35 GGTTTCATGGTTTGGTCTAACTTTACCGCTACTAA
[6]    35 GGGCAATAATGTTTATGTTGGTTTCATGGTTTGGT
```

XString Methods

- Basic utilities
  - subsequence selection
    - subseq, Views, narrow (XStringSet, IRanges package)
  - letter frequencies
    - alphabetFrequency, dinucleotideFrequency (tri..., oligo...), uniqueLetters
  - letter consensus
    - consensusMatrix,consensusString
  - letter transformation
    - reverse, complement, reverseComplement, translate, chartr
  - Input/Output
    - read.DNAStringSet (...B...,...RNA...,..AA..)
    - write.XStringSet, save.XStringSet

Xstrings Methods (c'ed)

- Advanced
  - **alignment utilities**:pairwiseAlignment, stringDist
  - **string matching**
    - (v)matchPDict (on a reference or a reference set (v))
    - (v)matchPDict, (v)countPDict,(v)whichPDict
    - matchPattern
    - (v)matchPattern,(v)countPattern, neditStartingAt, neditEndingAt, (which.) isMatchingStartingAt, (which.)isMatchingEndingAt
    - matchPWM(Position Weight Matrix, e.g. for transcription factor binding sites)
    - matchPWM,countPWM
  - **Others**
    - matchLRPatterns, trimLRPatterns,matchProbePair, findPalindromes, findComplementedPalindromes

Example 1: Letter/ alphabet frequencies

Single-letter frequencies

```
> alphabetFrequency(DNAString(yeastSEQCHR1))

    A     C     G     T     M     R     W     S     Y     K     V     H     D
69830 44643 45765 69970     0     0     0     0     0     0     0     0     0
    B     N     -     +     .
    0     0     0     0     0
```

```
> alphabetFrequency(DNAString(yeastSEQCHR1),baseOnly=TRUE)

    A     C     G     T other
69830 44643 45765 69970     0
```

Multi-letter frequencies

```
> dinucleotideFrequency(DNAString(yeastSEQCHR1))

   AA    AC    AG    AT    CA    CC    CG    CT    GA    GC    GG    GT    TA
23947 12493 13621 19769 15224  9218  7089 13112 14478  8910  9438 12938 16181
   TC    TG    TT
14021 15617 24151
```

```
> head(trinucleotideFrequency(DNAString(yeastSEQCHR1)),20)

 AAA  AAC  AAG  AAT  ACA  ACC  ACG  ACT  AGA  AGC  AGG  AGT  ATA  ATC  ATG  ATT
8576 4105 4960 6306 3924 2849 2186 3534 4537 2680 2707 3697 5242 3849 4294 6384
```

Example 2: String manipulation

Standard transformations

```
> head(narrow(srPhiX174,1,9))

  A DNAStringSet instance of length 6
    width seq
[1]     9 GTTATTATA
[2]     9 GGTGGTTAT
[3]     9 TACCGTCAA
[4]     9 GTACGCCGG
[5]     9 GGTTTCATG
[6]     9 GGGCAATAA
```

```
> head(reverse(narrow(srPhiX174,1,9)))

  A DNAStringSet instance of length 6
    width seq
[1]     9 ATATTATTG
[2]     9 TATTGGTGG
[3]     9 AACTGCCAT
[4]     9 GGCCGCATG
[5]     9 GTACTTTGG
[6]     9 AATAACGGG
```

```
> head(reverseComplement(narrow(srPhiX174,1,9))

  A DNAStringSet instance of length 6
    width seq
[1]     9 TATAATAAC
[2]     9 ATAACCACC
[3]     9 TTGACGGTA
[4]     9 CCGGCGTAC
[5]     9 CATGAAACC
[6]     9 TTATTGCCC
```

```
> head(translate(narrow(srPhiX174,1,9)))

  A AAStringSet instance of length 6
    width seq
[1]     3 VII
[2]     3 GGY
[3]     3 YRQ
[4]     3 VRR
[5]     3 GFM
[6]     3 GQ*
```

Example 2: String manipulation

- Bisulffite transformation

```
> alphabetFrequency(chartr("C","T", DNAString(yeastSEQCHR1)),baseOnly=TRUE)

     A      C      G      T  other
 69830      0  45765 114613      0

> alphabetFrequency(DNAString(yeastSEQCHR1),baseOnly=TRUE)

    A     C     G     T other
69830 44643 45765 69970     0
```

Example 3: Consensus

Consensus matrix

```
> snippet<-subseq(head(sort(srPhiX174),5),1,10);snippet

  A DNAStringSet instance of length 5
    width seq
[1]    10 AAATAATGTT
[2]    10 AACGTTATAT
[3]    10 AAGGAATGTG
[4]    10 AAGGACTGTG
[5]    10 AAGGACTGTG
```

```
> consensusMatrix(snippet,baseOnly=TRUE)

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
A        5    5    1    0    4    2    1    0    1     0
C        0    0    1    0    0    2    0    0    0     0
G        0    0    3    4    0    0    0    4    0     3
T        0    0    0    1    1    1    4    1    4     2
other    0    0    0    0    0    0    0    0    0     0
```

Consensus string

```
> consensusString(snippet)

[1] "AAGGAMTGTK"
```

Example 4: String Matching

- Match counting

```
> data(phiX174Phage)
> phiX174Phage


  A DNAStringSet instance of length 6
    width seq                                               names
[1]  5386 GAGTTTTATCGCTTCCATGACGC...ATGATTGGCGTATCCAACCTGCA Genbank
[2]  5386 GAGTTTTATCGCTTCCATGACGC...ATGATTGGCGTATCCAACCTGCA RF70s
[3]  5386 GAGTTTTATCGCTTCCATGACGC...ATGATTGGCGTATCCAACCTGCA SS78
[4]  5386 GAGTTTTATCGCTTCCATGACGC...ATGATTGGCGTATCCAACCTGCA Bull
[5]  5386 GAGTTTTATCGCTTCCATGACGC...ATGATTGGCGTATCCAACCTGCA G97
[6]  5386 GAGTTTTATCGCTTCCATGACGC...ATGATTGGCGTATCCAACCTGCA NEB03


> genome<- phiX174Phage[["NEB03"]]
> negPhiX174<- reverseComplement(srPhiX174)
> posCounts<- countPDict(PDict(srPhiX174),genome)
> negCounts<- countPDict(PDict(negPhiX174),genome)
> table(posCounts,negCounts)


         negCounts
posCounts    0
        0 1030
        1   83
```

Example 4: String Matching

- So we have 1030 reads that do not align either way to the genome and only 83 aligning.
- The match locations can be found using:

```
[[2]] IRanges of length 1 start end width [1] 2746 2780 35

[[3]] IRanges of length 1 start end width [1] 2757 2791 35

... <80 more elements>
```

Example 5: Pairwise alignment

- alignment scores

```
> posScore <- pairwiseAlignment(srPhiX174, genome, type="global-local", scoreOnly=TRUE)
> negScore <- pairwiseAlignment(negPhiX174, genome, type="global-local", scoreOnly=TRUE)
> which(pmin(posScore)<pmin(negScore))
```

```
[1] 932
```

- alignment

```
> pairwiseAlignment(srPhiX174[932],genome,type="global-local")
```

```
Global-Local PairwiseAlignmentsSingleSubject (1 of 1)
pattern:    [1] GCAATAACCTTGCGAGTCATTTCTTTGATTTGGTC
subject: [2804] GCAATAATGTTTATGTTGGTTTCATGG-TTTGGTC
score: -33.31176
```

```
> pairwiseAlignment(negPhiX174[932],genome,type="global-local")
```

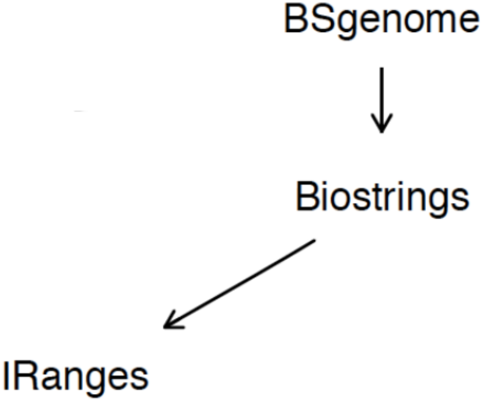
```
Global-Local PairwiseAlignmentsSingleSubject (1 of 1)
pattern:    [1] GACCAAATCAAAGAAATGACTCGCAAGGTTATTGC
subject: [3666] GACCAAATCAAAGAAATGACTCGCAAGGTTAGTGC
score: 61.4804
```

What next?

- We now have seen how to deal with biologically meaningful intervals and objects.
- Many organism have been sequenced and their genome is know.
- An interface in R to easily acces and manipulate such information would be very useful; this is the **BSgenome** package.

BSgenome

- It is not just a data package; it leverages th functionalitis introduced in **Biostrings**



BSgenome

Biostrings

IRanges

Available genomes

- Easy to find out

```
> require(BSgenome)
> head(available.genomes())
```

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

- However, large genomes(i.e. human, mouse, ...) packages might take log to transfer.

BSgenome Class overview (c'ed)

- Important:
  - proper S4 class usage ban accessing a slot through the "@" accessor, except within a package scope.
  - Hence, it is nowhere to be seen on the present slide

```
> library(BSgenome.Dmelanogaster.UCSC.dm3)
> # Dmelanogaster@seqs_dir
> #Dmelanogaster@masks_dir    ERROR
> #dir(Dmelanogaster@masks_dir)
```

BSgenome methods

- **Sequence selection**: [[, $
- **Subsequence selection**: getSeq
- **Accesors**: length, names/seqnames, mseqnames, seqlengths, masknames, sourceUrl
- **Matching**: all Biostings methods
- **SNPs**: injectSNPs, SNPlocs$_{p}kgname, SNPcount, SNPlocs$

Sequence information

- operation that do not load sequences

```
> require(BSgenome.Dmelanogaster.UCSC.dm3)
> head(seqnames(Dmelanogaster))
```

```
[1] "chr2L" "chr2R" "chr3L" "chr3R" "chr4"  "chrX"
```

```
> head(seqlengths(Dmelanogaster))
```

```
   chr2L    chr2R    chr3L    chr3R     chr4     chrX
23011544 21146708 24543557 27905053  1351857 22422827
```

- operation that do

```
> alphabetFrequency(Dmelanogaster[["chr4"]],baseOnly=TRUE)
```

```
     A      C      G      T  other
430227 238155 242039 441336    100
```

Masked vs unmasked

- unmasked package
    - e.g Dmelanogaster
- masked packages
    - e.g Hsapiens

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> Hsapiens[["chr1"]]

  249250621-letter "DNAString" instance
seq: NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN...NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

Extending Biostrings. Example 1

- Applying the Biostrings matching functions:

```
> exclude<-setdiff(seqnames(Hsapiens),c("chr1","chr2"))
> vcountPattern("ACYTANCAGT", Hsapiens, fixed=c(pattern=FALSE, subject=TRUE), exclude=exclude
```

```
  seqname strand count
1    chr1      +  1546
2    chr1      -  1545
3    chr2      +  1722
4    chr2      -  1684
```

```
> #vmatchPattern("ACYTANCAGT",Hsapiens, fixed=c(pattern=FALSE, subject=TRUE), exclude=exclude
> #asRangedData=FALSE)
```

Example 2

- Using a Pattern Dictionary, e.g. a library of microarray probes

```
> library(hgu95av2probe)
> probes<-DNAStringSet(hgu95av2probe$sequence[1:100])
> probes[1:10]

  A DNAStringSet instance of length 10
     width seq
 [1]    25 TGGCTCCTGCTGAGGTCCCCTTTCC
 [2]    25 GGCTGTGAATTCCTGTACATATTTC
 [3]    25 GCTTCAATTCCATTATGTTTTAATG
 [4]    25 GCCGTTTGACAGAGCATGCTCTGCG
 [5]    25 TGACAGAGCATGCTCTGCGTTGTTG
 [6]    25 CTCTGCGTTGTTGGTTTCACCAGCT
 [7]    25 GGTTTCACCAGCTTCTGCCCTCACA
 [8]    25 TTCTGCCCTCACATGCACAGGGATT
 [9]    25 CCTCACATGCACAGGGATTTAACAA
[10]    25 TCCTTGGTACTCTGCCCTCCTGTCA
```

Example 2

```
> counts<-vcountPDict(probes,Hsapiens,exclude=exclude);counts

DataFrame with 400 rows and 4 columns
    seqname strand     index count
      <Rle>  <Rle> <integer> <Rle>
1      chr1      +         1     0
2      chr1      +         2     0
3      chr1      +         3     0
4      chr1      +         4     0
5      chr1      +         5     0
...     ...    ...       ...   ...
396    chr2      -        96     0
397    chr2      -        97     0
398    chr2      -        98     0
399    chr2      -        99     0
400    chr2      -       100     0

> #whichMatch<-seqselect(counts$index,counts$count>0);whichMatch  No existeix seqselect!!
> #matchedProbes<- probes[WhichMatch];matchedProbes
> #matchLocs<-matchPDict(PDict(matchedProbes),Hsapiens$chr2);matchLocs
> #extractAllMatches(Hsapiens$chr2,matchLocs)
```

Example 5

- A new interesting feature is the possibility to inject SNPs!

```
> cat(available.SNPs(),sep="\n")

SNPlocs.Hsapiens.dbSNP.20090506
SNPlocs.Hsapiens.dbSNP.20100427
SNPlocs.Hsapiens.dbSNP.20101109
SNPlocs.Hsapiens.dbSNP.20110815
SNPlocs.Hsapiens.dbSNP.20111119
SNPlocs.Hsapiens.dbSNP.20120608

> library("SNPlocs.Hsapiens.dbSNP.20090506")
> HsWithSNPs<-injectSNPs(Hsapiens,
+           "SNPlocs.Hsapiens.dbSNP.20090506")
>
>
```

Example 5

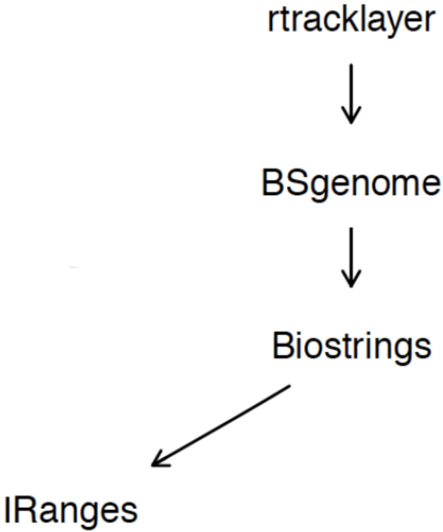
```
> HsWithSNPs

Human genome
|
| organism: Homo sapiens (Human)
| provider: UCSC
| provider version: hg19
| release date: Feb. 2009
| release name: Genome Reference Consortium GRCh37
| with SNPs injected from package: SNPlocs.Hsapiens.dbSNP.20090506
|
| single sequences (see '?seqnames'):
|   chr1                   chr2                   chr3
|   chr4                   chr5                   chr6
|   chr7                   chr8                   chr9
|   chr10                  chr11                  chr12
|   chr13                  chr14                  chr15
|   chr16                  chr17                  chr18
|   chr19                  chr20                  chr21
|   chr22                  chrX                   chrY
|   chrM                   chr1_gl000191_random   chr1_gl000192_random
|   chr4_ctg9_hap1         chr4_gl000193_random   chr4_gl000194_random
|   chr6_apd_hap1          chr6_cox_hap2          chr6_dbb_hap3
|   chr6_mann_hap4         chr6_mcf_hap5          chr6_qbl_hap6
|   chr6_ssto_hap7         chr7_gl000195_random   chr8_gl000196_random
|   chr8_gl000197_random   chr9_gl000198_random   chr9_gl000199_random
|   chr9_gl000200_random   chr9_gl000201_random   chr11_gl000202_random
|   chr17_ctg5_hap1        chr17_gl000203_random  chr17_gl000204_random
|   chr17_gl000205_random  chr17_gl000206_random  chr18_gl000207_random
|   chr19_gl000208_random  chr19_gl000209_random  chr21_gl000210_random
```

What next?

- Now that we can acces genomic information, it would be useful to import the related annotation. That's (one of) the purpose of the following packages:
  - **rtracklayer**
  - **GenomicFeatures**
  - **biomaRt**
  - **genomeIntervals**
- **rtracklayer** offers export function too and as alredy presented, **genomeIntervals** offers interval utilities similar to **IRanges**

rtracklayer



rtracklayer

BSgenome

Biostrings

IRanges

Methods

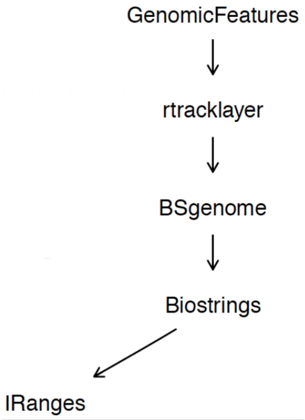
- There are two high level methods
  - import
  - export
- Both accept the following formats:
  - BED: bed, bedGraph, bed15
  - GFF: gff1, 2 and 3
  - WIG
- export works with *RangedData* objects
- import returns a *RangedData* object or *GRanges* object, depending on the (asRangedData) boolean argument.

Methods (c'ed)

- When exporting
  - The naming convention of the *RangedData* column names is crucial.
  - The following column names
    - **names**: for exporting the feature names
    - **scores**: for exporting the feature scores
    - **strand**: for exporting the feature strands
    - **see** ?export.bed for the complete details

GenomicFeatures

- manegement of transcript information
    - using **GenomicRanges**
    - stored into SQLite databases



GenomicFeatures

rtracklayer

BSgenome

Biostrings

IRanges

Constructors and Class

- makeTranscriptDbFromBiomart
- makeTrascriptDbFromUCSC

```
> library(GenomicFeatures)
> head(supportedUCSCtables())
```

```
                       track           subtrack
knownGene         UCSC Genes               <NA>
knownGeneOld3 Old UCSC Genes               <NA>
ccdsGene                CCDS               <NA>
refGene         RefSeq Genes               <NA>
xenoRefGene     Other RefSeq               <NA>
vegaGene          Vega Genes Vega Protein Genes
```

```
> mm9KG<-makeTranscriptDbFromUCSC(genome="mm9",tablename="knownGene")
> saveFeatures(mm9KG,file="mm9KG.sqlite")
```

Constructors and Class

```
> mm9KG<-loadFeatures("mm9KG.sqlite")
> mm9KG


TranscriptDb object:
| Db type: TranscriptDb
| Supporting package: GenomicFeatures
| Data source: UCSC
| Genome: mm9
| Organism: Mus musculus
| UCSC Table: knownGene
| Resource URL: http://genome.ucsc.edu/
| Type of Gene ID: Entrez Gene ID
| Full dataset: yes
| miRBase build ID: NA
| transcript_nrow: 55419
| exon_nrow: 246570
| cds_nrow: 213117
| Db created by: GenomicFeatures package from Bioconductor
| Creation time: 2014-06-07 22:03:43 +0200 (Sat, 07 Jun 2014)
| GenomicFeatures version at creation time: 1.16.2
| RSQLite version at creation time: 0.11.4
| DBSCHEMAVERSION: 1.0
```

Extractors

- ungrouped
  - transcriptBy
  - exonsBy
  - intronsByTranscript
  - fiveUTRsByTranscript
  - threeUTRsByTranscript

Extractors

```
> library(GenomicFeatures)
> txExons<-exonsBy(mm9KG)
> head(txExons)

GRangesList of length 6:
$1
GRanges with 8 ranges and 3 metadata columns:
      seqnames             ranges strand |   exon_id   exon_name exon_rank
         <Rle>          <IRanges>  <Rle> | <integer> <character> <integer>
  [1]     chr1 [4797974, 4798063]      + |         1        <NA>         1
  [2]     chr1 [4798536, 4798567]      + |         2        <NA>         2
  [3]     chr1 [4818665, 4818730]      + |         3        <NA>         3
  [4]     chr1 [4820349, 4820396]      + |         4        <NA>         4
  [5]     chr1 [4822392, 4822462]      + |         5        <NA>         5
  [6]     chr1 [4827082, 4827155]      + |         6        <NA>         6
  [7]     chr1 [4829468, 4829569]      + |         7        <NA>         7
  [8]     chr1 [4831037, 4832908]      + |         9        <NA>         8

$2
GRanges with 9 ranges and 3 metadata columns:
      seqnames             ranges strand | exon_id exon_name exon_rank
  [1]     chr1 [4797974, 4798063]      + |       1      <NA>         1
  [2]     chr1 [4798536, 4798567]      + |       2      <NA>         2
  [3]     chr1 [4818665, 4818730]      + |       3      <NA>         3
  [4]     chr1 [4820349, 4820396]      + |       4      <NA>         4
  [5]     chr1 [4822392, 4822462]      + |       5      <NA>         5
  [6]     chr1 [4827082, 4827155]      + |       6      <NA>         6
  [7]     chr1 [4829468, 4829569]      + |       7      <NA>         7
  [8]     chr1 [4831037, 4831213]      + |       8      <NA>         8
  [9]     chr1 [4835044, 4836816]      + |      10      <NA>         9
```

Usage

- Overlapping with transcripts
  - findOverlaps
  - countOverlaps
  - match
  - %in%
  - subsetByOverlaps
- More about these in the following part about **GenomicRanges**

biomaRt

- Side note to get help from within R:
  - vignette(“biomaRt”,package=“biomaRt”)
- biomaRt is an interface to the collection of databases that implements the bioMart software suite:
  - `http://biomart.org`
  - allow retrieval og huge datasets from different sources through a common interface
  - examples are: Ensembl, HapMap, Uniprot, ...

biomaRt, an example

- Connect the mart database

```
> require(biomaRt)
> ensembl<- useMart("ensembl")
> head(listDatasets(ensembl))

                         dataset                                description
1         oanatinus_gene_ensembl     Ornithorhynchus anatinus genes (OANA5)
2        cporcellus_gene_ensembl            Cavia porcellus genes (cavPor3)
3        gaculeatus_gene_ensembl     Gasterosteus aculeatus genes (BROADS1)
4         lafricana_gene_ensembl         Loxodonta africana genes (loxAfr3)
5 itridecemlineatus_gene_ensembl Ictidomys tridecemlineatus genes (spetri2)
6        choffmanni_gene_ensembl        Choloepus hoffmanni genes (choHof1)
  version
1   OANA5
2 cavPor3
3 BROADS1
4 loxAfr3
5 spetri2
6 choHof1

> ensembl<- useMart("ensembl",dataset="dmelanogaster_gene_ensembl")
> head(listAttributes(ensembl))

                   name           description
1       ensembl_gene_id       Ensembl Gene ID
2 ensembl_transcript_id Ensembl Transcript ID
3    ensembl_peptide_id    Ensembl Protein ID
4       ensembl_exon_id       Ensembl Exon ID
5           description           Description
6       chromosome_name       Chromosome Name
```

biomaRt, an example (c'ed)

- query the database

```
> exon.annotation<-getBM(c("ensemb_gene_id","strand",
+                  "chromosome_name","ensembl_exon_id",
+                  "exon_chrom_start","exon_chrom_end"),
+                  mart=ensembl,filters="chromosome_name",
+                  values="4")
```

- convert into a RangedData / Granges

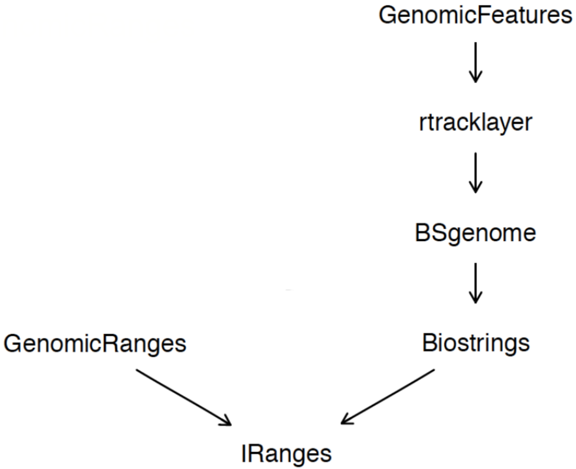
genomeIntervals

- Similar interval implementation to IRanges
  - (+) overall faster, gff function more robust to 'incorrect' format
  - (-) less integrated in R
- Two classes:
  - Genome_intervals
  - Genome_intervals_stranded
- Methods
  - input
    - **readGff3, getGffAttributes**, parseGffAttributes
  - intervals utilities
    - interval_overlap, **interval_complement**, interval_union, interval_intersection

What next?

- We have seen how to get genomic sequences and their annotation
- For processing NGS data, we are now missing the other half of the workflow: loading and manipulating the actual data. For this, three packages are available.
  - **GenomicRanges**
  - **Rsamtools**
  - **ShortRead**

GenomicRanges



GenomicFeatures

rtracklayer

BSgenome

GenomicRanges

Biostrings

IRanges

Naive approach

- Genomic coordinates consist of
  - chromosome
  - position
  - strand
  - additional information
    - GC content
    - etc.
- This caan be represented by a data.frame
  - fine for organism information ($\sim$ 100k exons, 20k genes)
  - not for million of reads

BIOC representation for intervals with data

- *RangedData*
  - used by **rtracklayer**
  - interval grouped by chromosome/conting
  - strand unaware
- *GRanges*
  - used by **GenomicFeatures**
  - intervals not required to be grouped by chromosome/contig
  - strand aware
  - *GRangesList* can hold exons with spliced transcripts

GRanges constructor and slots

- starts and ends defined in an *IRanges* object
- strand, seqnames (chromosome) and seqlenghts (chromosome size) to be provided

```
> grngs<-GRanges(seqnames=c("chr1","chr2","chr1"),
+       ranges=IRanges(start=c(3,4,1),end=c(7,5,3)),
+     strand=c("+","+","-"),seqlengths = c("chr1"=24,"chr2"=18))
> grngs

GRanges with 3 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1    [3, 7]      +
  [2]     chr2    [4, 5]      +
  [3]     chr1    [1, 3]      -
  ---
  seqlengths:
   chr1 chr2
     24   18
```

- additional slots can contain mtadata information

```
> getSlots("GRanges")

       seqnames          ranges          strand elementMetadata         seqinfo
          "Rle"       "IRanges"           "Rle"     "DataFrame"       "Seqinfo"
       metadata
         "list"
```

Interval operations

- Intra-interval
  - flank,resize,shift
- Inter-interval
  - disjoin, gaps, reduce, range
  - coverage
- Between intervals sets
  - union, intersect, setdiff
  - punion,pintersectm psetdiff
  - findOverlaps, countOverlaps, %in%, match
- Low Level
  - start,end,width

Other functions

- Selecting
  - seqselect, [
  - head, tail, window
  - subset, subsetByOverlaps

```
> grngs[strand(grngs)== "-"]

GRanges with 1 range and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1    [1, 3]      -
  ---
  seqlengths:
   chr1 chr2
     24   18

> # seqselect(grngs,strand(grngs)=="-") NO EXISTEIX seqselect
```

Example 1: Intra-interval

- shift

```
> grngs

GRanges with 3 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1    [3, 7]      +
  [2]     chr2    [4, 5]      +
  [3]     chr1    [1, 3]      -
  ---
  seqlengths:
   chr1 chr2
     24   18
```

```
> shift(grngs,1)

GRanges with 3 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1    [4, 8]      +
  [2]     chr2    [5, 6]      +
  [3]     chr1    [2, 4]      -
  ---
  seqlengths:
   chr1 chr2
     24   18
```

- resize

```
> resize(grngs,10)

GRanges with 3 ranges and 0 metadata column
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1   [3, 12]      +
  [2]     chr2   [4, 13]      +
  [3]     chr1   [1,  3]      -
  ---
  seqlengths:
   chr1 chr2
     24   18
```

- flank

```
> flank(grngs,2)

GRanges with 3 ranges and 0 metadata column
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1    [1, 2]      +
  [2]     chr2    [2, 3]      +
  [3]     chr1    [4, 5]      -
  ---
  seqlengths:
   chr1 chr2
     24   18
```

Overlap detection

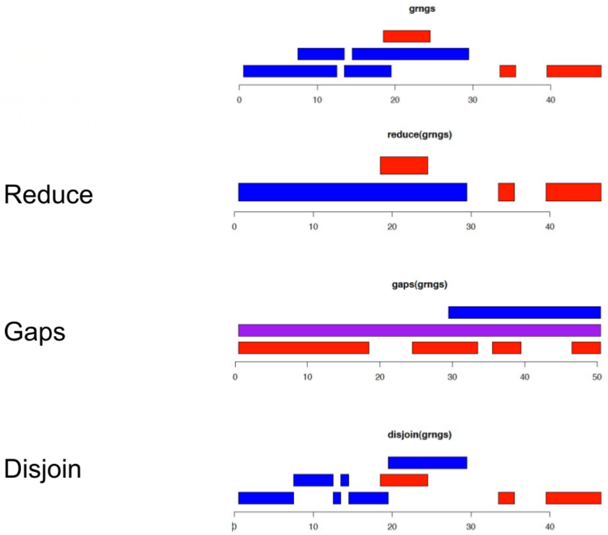
- *findOverlap* and *countOverlaps* produce a mapping and a tabulation of interval overlaps, respectively

```
> ol<-findOverlaps(grngs,reduce(grngs))
> ol

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

Rsamtools

Blue represents the "+" strand, red the"-" strand



samtools and Rsamtools

- samtools
  - Data Format: SAM(text) and BAM (binary)
  - Tools: merge, sort, pileup, view, etc.
- Rsamtools
  - Reads and represents BAMfiles
  - high level: readAligned (type=BAM), readPileup
  - lower level: scanBam, scanBamParam, ScanBamWhat
  - utilities: countBam, sortBam, indexBam, filterBam, scanBamHeader
  - views: BamViews

Input

- readAligned returns an *alignedRead* class
    - described in the following section on **ShortRead**
- scanBam returns a list of list i.e.. one list per column in the SAM file.
    - qname: a *BStringSet* containing the read id
    - seq: a *DNAStringSet* containing the read sequence
    - etc.
    - The possible fields can be found with scanBamWhat()

```
> require(Rsamtools)
> scanBamWhat()
```

```
 [1] "qname"        "flag"         "rname"        "strand"       "pos"
 [6] "qwidth"       "mapq"         "cigar"        "mrnm"         "mpos"
[11] "isize"        "seq"          "qual"         "groupid"      "mate_status"
```

- scanBam is the function called by the **GenomicRanges** readGappedAlignments method

Input (c'ed)

- The input can be controlled using ScanBamParam
  - it has three fields
    - **which**: *GRanges* selecting references, genomic loci, strand, ...
    - **flag**: use the SAM flag to selected paired, mapped, etc. reads.

```
> names(formals(scanBamFlag))
```

```
 [1] "isPaired"                        "isProperPair"
 [3] "isUnmappedQuery"                 "hasUnmappedMate"
 [5] "isMinusStrand"                   "isMateMinusStrand"
 [7] "isFirstMateRead"                 "isSecondMateRead"
 [9] "isNotPrimaryRead"                "isNotPassingQualityControls"
[11] "isDuplicate"                     "isValidVendorRead"
```

    - **what**: fields to retrieve (cf. scanBamWhat)

GappedAlignments vs AlignedRead

- *AlignedRead*
  - reads complete files
  - include sequence, quality, identifier, etc.
  - reads are assumed to be ungapped
- *GappedAlignments*
  - use scanBam
  - genomic coordinates, 'cigar', covered intervals
  - Cigar: an RLE; M(match), I (insertion), D (deleiton), N (skipped), P (padding), S/H (soft/hard clip)
  - direct IRanges accesors (sub-setting, narrowing, coverage)

BamViews

- Acces a set of experiments stored in BAM files
  - for example to query a specific loci
- Check the vignette (”leeViews”)
- Still very unstable!

BamViews

```
> library(leeBamViews)
> bpaths=dir(system.file("bam",package="leeBamViews"),full=TRUE, patt="bam$")
> gt<- do.call(rbind, strsplit(basename(bpaths),"_"))[,1]
> geno<-substr(gt,1,nchar(gt)-1)
> lane<- substr(gt,nchar(gt),nchar(gt))
> pd=DataFrame(geno=geno, lane=lane, row.names=paste(geno,lane,sep="."))
> bs1=BamViews(bamPaths=bpaths, bamSamples=pd, bamExperiment=list(annotation="org.Sc.sgd.db"
> bamPaths(bs1)

                                                                      isowt.5
"/home/alex/R/x86_64-pc-linux-gnu-library/3.1/leeBamViews/bam/isowt5_13e.bam"
                                                                      isowt.6
"/home/alex/R/x86_64-pc-linux-gnu-library/3.1/leeBamViews/bam/isowt6_13e.bam"
                                                                        rlp.5
  "/home/alex/R/x86_64-pc-linux-gnu-library/3.1/leeBamViews/bam/rlp5_13e.bam"
                                                                        rlp.6
  "/home/alex/R/x86_64-pc-linux-gnu-library/3.1/leeBamViews/bam/rlp6_13e.bam"
                                                                        ssr.1
  "/home/alex/R/x86_64-pc-linux-gnu-library/3.1/leeBamViews/bam/ssr1_13e.bam"
                                                                        ssr.2
  "/home/alex/R/x86_64-pc-linux-gnu-library/3.1/leeBamViews/bam/ssr2_13e.bam"
                                                                        xrn.1
  "/home/alex/R/x86_64-pc-linux-gnu-library/3.1/leeBamViews/bam/xrn1_13e.bam"
                                                                        xrn.2
  "/home/alex/R/x86_64-pc-linux-gnu-library/3.1/leeBamViews/bam/xrn2_13e.bam"
```

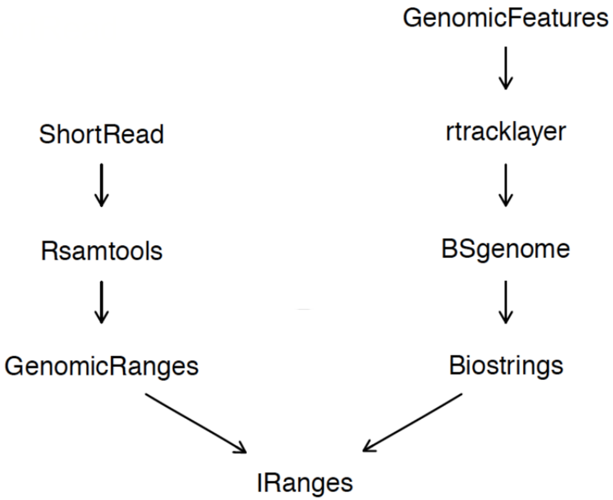
BamViews

```
> bamSamples(bs1)

DataFrame with 8 rows and 2 columns
               geno        lane
        <character> <character>
isowt.5       isowt           5
isowt.6       isowt           6
rlp.5           rlp           5
rlp.6           rlp           6
ssr.1           ssr           1
ssr.2           ssr           2
xrn.1           xrn           1
xrn.2           xrn           2

> sel<-GRanges(seqnames="Scchr13",IRanges(start=861250,end=863000),strand="+")
> # covex=RleList(lapply(bamPaths(bs1),function(x) coverage(readGappedAlignments(x))[[1]]))
```

ShortRead


GenomicFeatures
ShortRead
rtracklayer
Rsamtools
BSgenome
GenomicRanges
Biostrings
IRanges

ShortRead

- Input
  - read most sequence proprietary formats
  - read fastq
  - read BAM
- Exploration
  - contains sequence, quality, id, etc. information
- Manipulation
  - allow the manipulation of the fields with a limited memory impact
- Quality assessment
  - offers quality assessment functionalities

AlignedReadClass

- The main class to store the read information

```
> require(ShortRead)
> showClass("AlignedRead")

Class "AlignedRead" [package "ShortRead"]

Slots:

Name:       chromosome        position          strand    alignQuality
Class:          factor         integer          factor    QualityScore

Name:        alignData         quality           sread              id
Class: AlignedDataFrame    QualityScore     DNAStringSet      BStringSet

Extends:
Class "ShortReadQ", directly
Class "ShortRead", by class "ShortReadQ", distance 2
Class ".ShortReadBase", by class "ShortReadQ", distance 3
```

- All slots can be accessed through accordingly named accessors

SRFilterclass

- Useful tools to filter the reads during or after the import

```
> showClass("SRFilter")

Class "SRFilter" [package "ShortRead"]

Slots:

Name:             .Data            name
Class:         function ScalarCharacter

Extends:
Class "function", from data part
Class ".SRUtil", directly
Class "OptionalFunction", by class "function", distance 2
Class "PossibleMethod", by class "function", distance 2
Class "expressionORfunction", by class "function", distance 2
Class "functionORNULL", by class "function", distance 2
```

- many already implemented
  - idFilter
  - chromosomeFilter
  - positionFilter
  - strandFilter
  - etc.
- They can be combined using compose
  - compose(idFilter,chromosomeFilter)

Other classes

- The package implements many classes to hold the different kind of data

```
> getClasses(where="package:ShortRead")

 [1] "AlignedDataFrame"       "AlignedRead"              "ArrayIntensity"
 [4] "BAMQA"                  "BowtieQA"                 "ExperimentPath"
 [7] "FastqFile"              "FastqFileList"            "FastqFileReader"
[10] "FastqQA"                "FastqQuality"             "FastqSampler"
[13] "FastqSamplerList"       "FastqStreamer"            "FastqStreamerList"
[16] "IntegerQuality"         "Intensity"                "IntensityInfo"
[19] "IntensityMeasure"       "MAQMapQA"                 "MatrixQuality"
[22] "NumericQuality"         "QA"                       ".QA"
[25] ".QA2"                   "QAAdapterContamination"   "QACollate"
[28] "QAData"                 "QAFastqSource"            "QAFiltered"
[31] "QAFlagged"              "QAFrequentSequence"       "QANucleotideByCycle"
[34] "QANucleotideUse"        "QAQualityByCycle"         "QAQualityUse"
[37] "QAReadQuality"          "QASequenceUse"            "QASource"
[40] "QASummary"              "QualityScore"             ".Roche"
[43] "RochePath"              "RocheSet"                 "RtaIntensity"
[46] "SFastqQuality"          "ShortRead"                ".ShortReadBase"
[49] "ShortReadFile"          "ShortReadQ"               "ShortReadQQA"
[52] "Snapshot"               "SnapshotFunction"         "SnapshotFunctionList"
[55] ".Solexa"                "SolexaExportQA"           "SolexaIntensity"
[58] "SolexaIntensityInfo"    "SolexaPath"               "SolexaRealignQA"
[61] "SolexaSet"              "SpTrellis"                "SRError"
[64] "SRFilter"               "SRFilterResult"           "SRList"
[67] "SRSet"                  ".SRUtil"                  "SRVector"
[70] "SRWarn"                 "trellis"
```

Input and accessor examples

Simple walk through

```
> require("EatonEtAlChIPseq")
> fl<-system.file("extdata","GSM424494_wt_G2_orc_chip_rep1_S288C_14.mapview.txt.gz",pa
> aln<-readAligned(fl,type="MAQMapview");aln

class: AlignedRead
length: 478774 reads; width: 39 cycles
chromosome: S288C_14 S288C_14 ... S288C_14 S288C_14
position: 2 4 ... 784295 784295
strand: + - ... + +
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24

> head(sread(aln))

  A DNAStringSet instance of length 6
    width seq
[1]    39 CGGCTTTCTGACCGAAATTAAAAAAAAAAAATGAAAATG
[2]    39 GATTTATGAAAGAAATTAAAAAAAAAAAATGAAAATGAA
[3]    39 CTTTCTGACCGAAATTAAAAAAAAAAAATGAAAATGAAA
[4]    39 TTTCTGACCGAAATTAAAAAAAAAAAATGAAATTGAAAC
[5]    39 TTTATGAAAGAAAATAAAAAAAAAAAATGAAAATGAAAC
[6]    39 TTTCTGAAAGAAATTAAAAAAAAAAAATGAAAATGAAAC
```

Input and accessor examples

- with filters

```
> filter<- compose(chromosomeFilter("S288C_14"),positionFilter(min=1,max=1000))
> alnF<-readAligned(fl,type="MAQMapview",filter=filter);alnF

class: AlignedRead
length: 715 reads; width: 39 cycles
chromosome: S288C_14 S288C_14 ... S288C_14 S288C_14
position: 2 4 ... 997 999
strand: + - ... - -
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
```

Input and accessor examples

```
> head(quality(aln))

class: FastqQuality
quality:
  A BStringSet instance of length 6
    width seq
[1]    39 >>>>>>>>>><>>><<<>>5><<><<<<<<44444///,
[2]    39 ,%//4&/14&&:<<<.<>>>>>>>>>>>>>>>>>>>>>>
[3]    39 >>>>>>>>>>>>>>>><><>><>>><<<<<44444////
[4]    39 <>>>>>:<<><::>><>:><<<>><::::<44%-4//$/
[5]    39 ,,/,1+41+4<<<4<<>>>>>>>>>>>>>>>>>>>>>>>
[6]    39 ,++(*44*/<<<<+><>>>>>>>>>><>>>>>>>>>>>

> head(id(aln))

  A BStringSet instance of length 6
    width seq
[1]    23 X8193_200:5:175:690:668
[2]    22 X8193_200:5:62:612:145
[3]    23 X8193_200:5:206:446:786
[4]    22 X8193_200:5:12:950:859
[5]    23 X8193_200:5:230:400:822
[6]    23 X8193_200:5:258:160:889
```

Manipulation example

- For example to rename chromosome

```
> chrom<-chromosome(alnF)
> i<-sub("S288C_([[:digit:]]+)","\\1",levels(chrom));i

[1] "14"

> levels(chrom)

[1] "S288C_14"

> levels(chrom)<-paste("chr",as.roman(i),sep="")
> levels(chrom)

[1] "chrXIV"

> alnF<-renew(alnF,chromosome=chrom);alnF

class: AlignedRead
length: 715 reads; width: 39 cycles
chromosome: chrXIV chrXIV ... chrXIV chrXIV
position: 2 4 ... 997 999
strand: + - ... - -
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24

>
```

Quality assessment

- Many functions are available in **ShortRead** that can be used for performing QA

```
> f.list<-showMethods(
+ where="package:ShortRead",
+ printTo=FALSE)
```

```
> sapply(strsplit(f.list[grep("Function",f.list)],' '),
+        function(x)x[2])

 [1] "alphabet"                          "alphabetByCycle"
 [3] "alphabetFrequency"                 "alphabetScore"
 [5] "annTrack"                          "append"
 [7] "!"                                 "["
 [9] "[["                                "$<-"
[11] "$"                                 "c"
[13] "chromosome"                        "clean"
[15] "coerce"                            "to=\"classGeneratorFu
[17] "to=\"OptionalFunction\""           "to=\"genericFunction\
[19] "to=\"OptionalFunction\""           "to=\"genericFunction\
[21] "countLines"                        "coverage"
[23] "dim"                               "dustyScore"
[25] "encoding"                          "experimentPath"
[27] "fac"                               "FastqFileList"
[29] "FastqQuality"                      "FastqSamplerList"
[31] "FastqStreamerList"                 "FastqStreamer"
[33] "files"                             "flag"
[35] "functions"                         "getTrellis"
[37] "id"                                "ignore.strand"
[39] "%in%"                              "laneNames"
[41] "lapply"                            "length"
[43] "names<-"                           "names"
[45] "name"                              "narrow"
[47] "pan"                               "pData"
[49] "phenoData"                         "position"
[51] "qa2"                               "QACollate"
[53] "qa"                                "rbind"
[55] "read454"                           "readAligned"
[57] "readBaseQuality"                   "readFastaQual"
```

QA example (yet another one...)

- Using independent functions

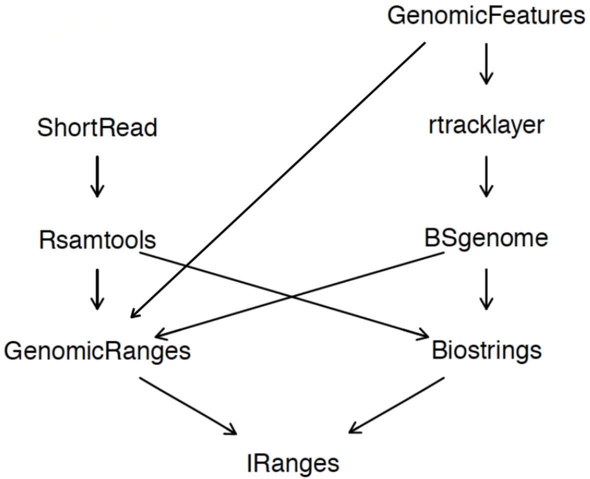
```
> abc<- alphabetByCycle(sread(alnF))
> abc[1:4,1:12]

        cycle
alphabet [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
       A  239  246  251  236  244  244  223  207  184   212   217   230
       C  197  172  180  178  169  194  192  194  202   212   185   182
       G  103   88   87  105   89   93  108  101  114    90   103    83
       T  176  209  197  196  213  184  192  213  215   201   210   220

> abc<-abc[1:4, ]
> par(mfrow=c(1,2))
> matplot(t(abc),type="l",lty=rep(1,4))
> m<-as (quality(alnF),"matrix")
> plot(colMeans(m),type="b")
```

- All these and more are combined into the function: qa()
- These can then be reported using the report() function

Conclusion



GenomicFeatures

ShortRead

rtracklayer

Rsamtools

BSgenome

GenomicRanges

Biostrings

IRanges

Conclusion

- We have seen the two “branches” of the core packages:
  - the one used to get genomic sequence and annotation
  - the one used to load and manipulate NGS data
- Actually, the cit is not so clear ad the packages of these two branches are interacting at different levels.
- They provide numerous functionalities and are getting into a “production” (stable development) state.
- Higher level packaages are being developed to wrap these functionalities into more user friendly packages.

Conclusion

- If you would start today using these packages:
  - go for the BAM format
  - go for GRanges objects
  - Be on the lookout, especially for the *SummarizedExperiment* class in the **GenomicRanges** package.
    - It is a concept similar to the *ExpressionSet* class devoloped for microarray and aims at normalizing the output of NGS experiments within R/Bioconductor

If we were fast..

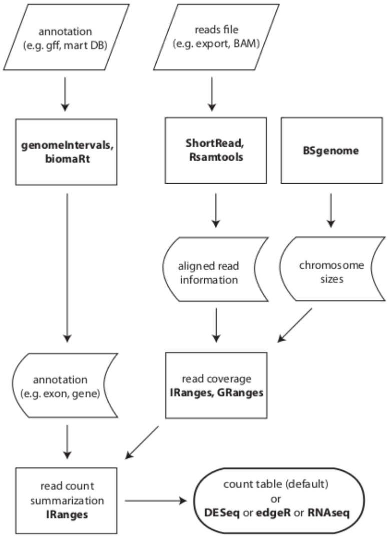
- Another couple of package to mention
    - Rsubread (only on linux)
    - easyRNASeq (self-promotion)

Rsubread

- a package to align short read in R!
- If you have a session on vuori you can try that code slightly modified in the R file to use only chromosome 1

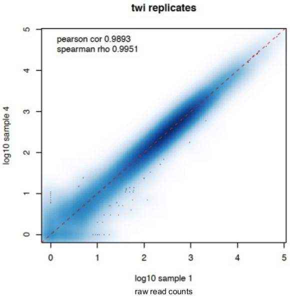
```
> ## write the human genome sequences
> writeXStringSet(Reduce(append,
+ lapply(seqnames(Hsapiens),
+ function(nam)
+ {dss<-DNAStringSet(unmasked(Hsapiens[[nam]]))
+ names(dss)<-nam
+ dss})),file="hg19.fa")
> ##create the indexes
> require(Rsubread)
> dir.create("indexes")
> buildindex(basename=file.path("indexes","hg19"),
+            reference="hg19.fa")
> ## align the reads
> sapply(dir(pattern="*\\.gz$"),function(fil){
+   ## decomplress the files
+   gunzip(fil)
+   ##align
+   align(index=file.path("indexes","hg19"),
+         readfile1=sub("\\.gz$","",fil),
+         nsubreads=2, TH1=1,
+         output_file=sub("\\.fastq\\.gz$","\\,sam",fil))
+   ## create bam files
+   asBAM(file=sib("\\.fastq\\.gz$","\\.sam",fil),
+         destination=sub("\\.fastq\\.gz$","",fil),
+         indexDestination=TRUE)
+   })
```

easyRNASeq package



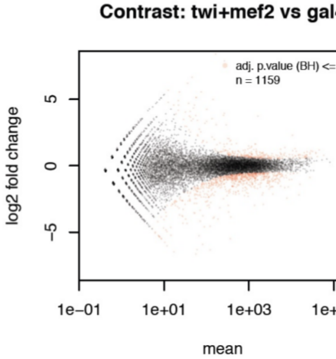
Replicate comparison

- The simplest output is a matrix
  - Comparing replicates is therefore easy
  - Can be done automatically if the user provides the sample information
  - GRAFIC



Normalization

- Three types can be applied
    - **R**eads **P**er feature **K**b per **M**ilion reads in the library
- DESeq
    - based on Negative Binomial
    - fit a model to correct for the library sizes
- edgeR
    - based on Negative Binomial
    - use a trimmed mean og M-values to correct for the library sizes


Contrast: twi+mef2 vs gal
adj. p.value (BH) <=
n = 1159
log2 fold change
1e−01
1e+01
1e+03
1e+
mean