Gene expression analysis with R and Bioconductor:
from measurements to annotated lists of interesting genes

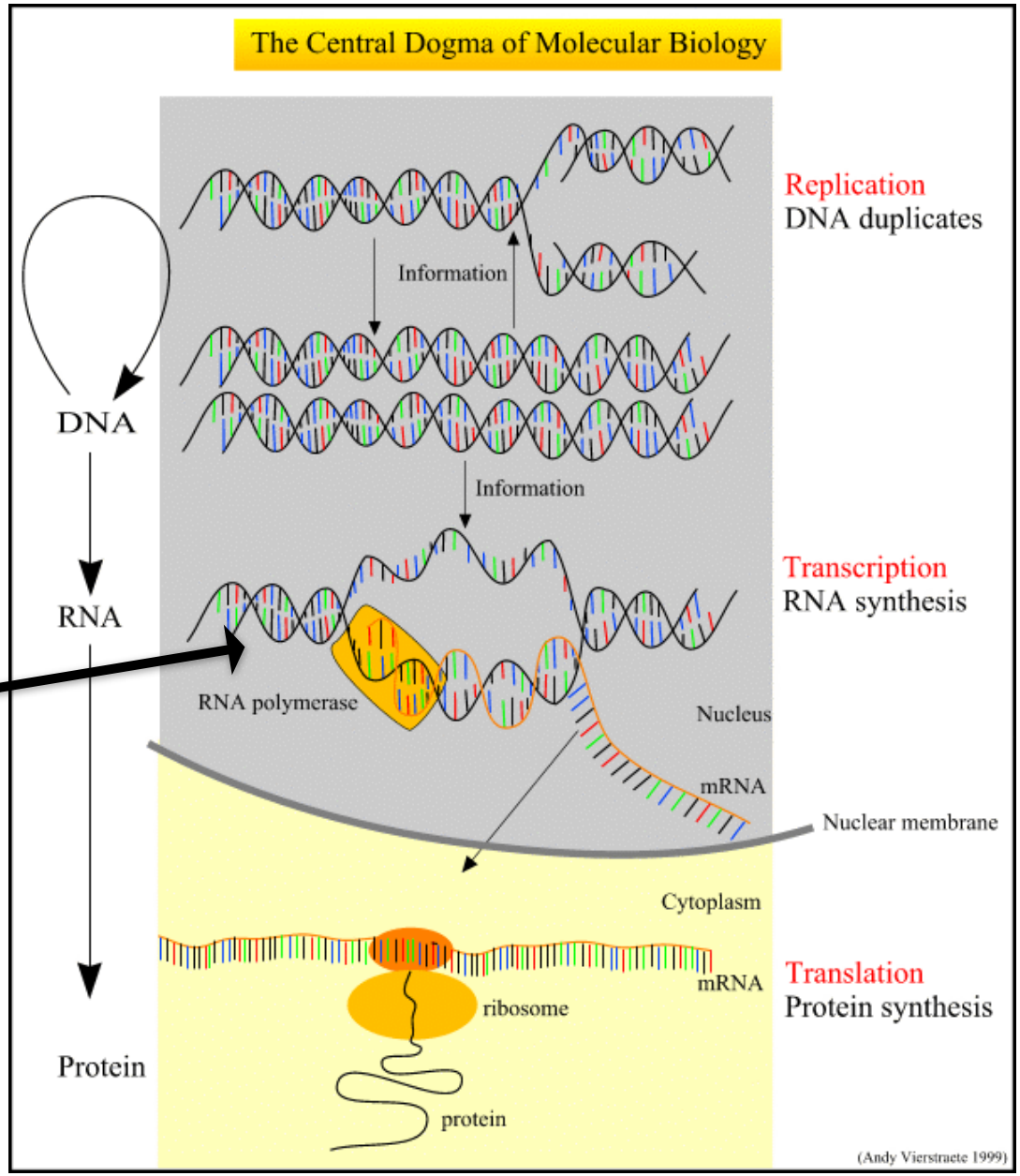
Héctor Corrada Bravo
Dept. of Computer Science
Center for Bioinformatics and Computational Biology
University of Maryland

Computational Systems Biology and Functional Genomics
Spring 2012

What Do They Measure?

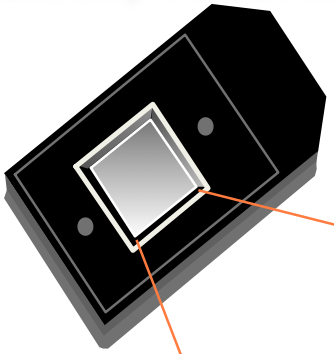
Microarrays

Gene Expression Arrays
Exon Arrays



Affymetrix GeneChip® Arrays

GeneChip Probe Array



1.28cm

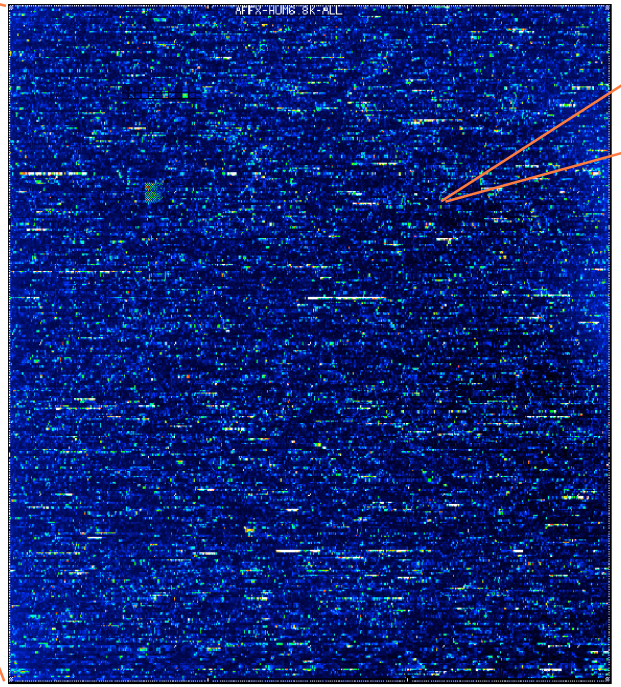
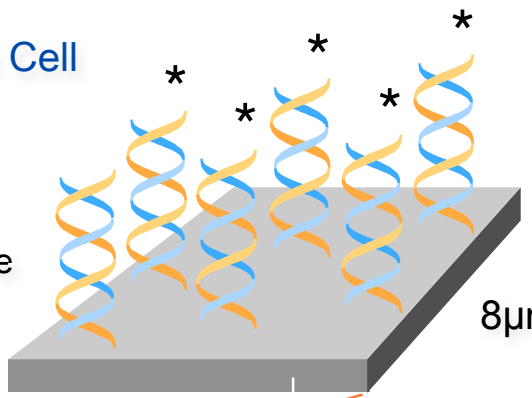


Image of Hybridized Probe Array

Hybridized Probe Cell

Single stranded,
labeled RNA target

Oligonucleotide probe

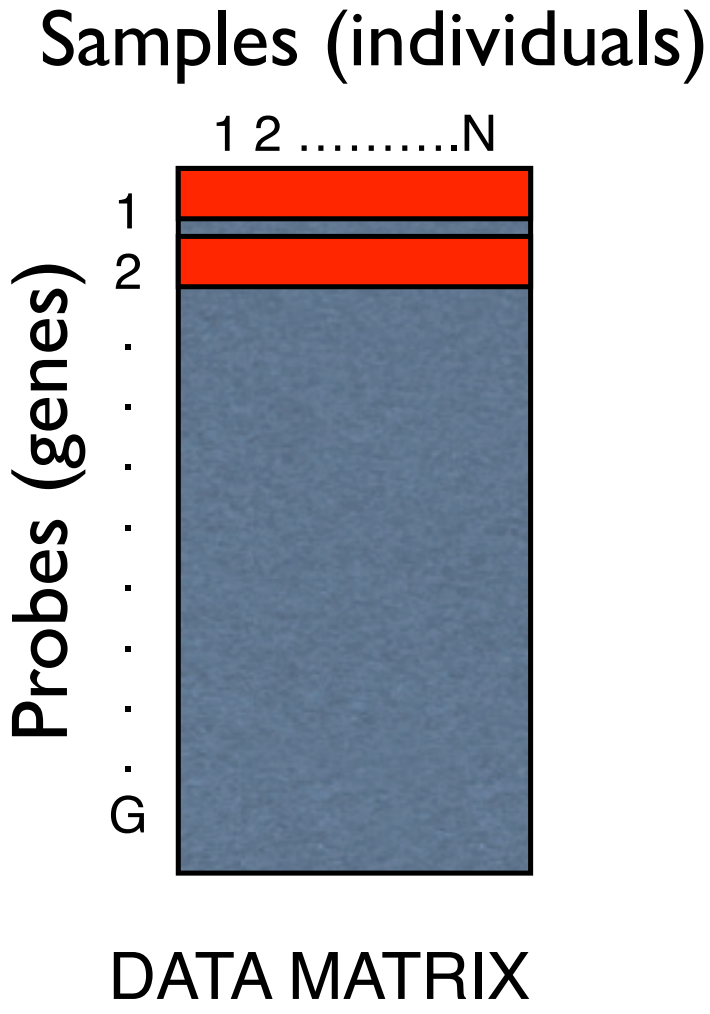


8µm

Millions of copies of a specific oligonucleotide probe

>1,000,000 different complementary probes

Measurements



For today

- Given expression data from two groups
- Find genes that are interesting
  - *differentially expressed*
- Annotate list in meaningful ways
  - *Gene Ontology, Pathways, Pubmed, etc.*

For today...

- Hopefully you've run the setup script sent earlier today. You can get it here:
  `ftp://ftp.umiacs.umd.edu/pub/data/hcorrada/setup.R`
- We will analyze a breast cancer dataset, the setup script above downloads automatically to the current directory
- If that didn't work, you can get it here:
  `ftp://ftp.umiacs.umd.edu/pub/data/hcorrada/chang03.rda`
- While we wait to get started, make sure the following code works for you

```
> library(affy)
> load("chang03.rda")
> show(chang03)
ExpressionSet (storageMode: lockedEnvironment)
assayData: 12625 features, 24 samples
  element names: exprs, exprs.mas5, exprs.mas5.err
protocolData: none
phenoData
  rowNames: GSM4901 GSM4902 ...
    GSM4924 (24 total)
  varLabels: Patient disease.state
```

Gene Expression Analysis with R and Bioconductor: from measurements to annotated lists of interesting genes

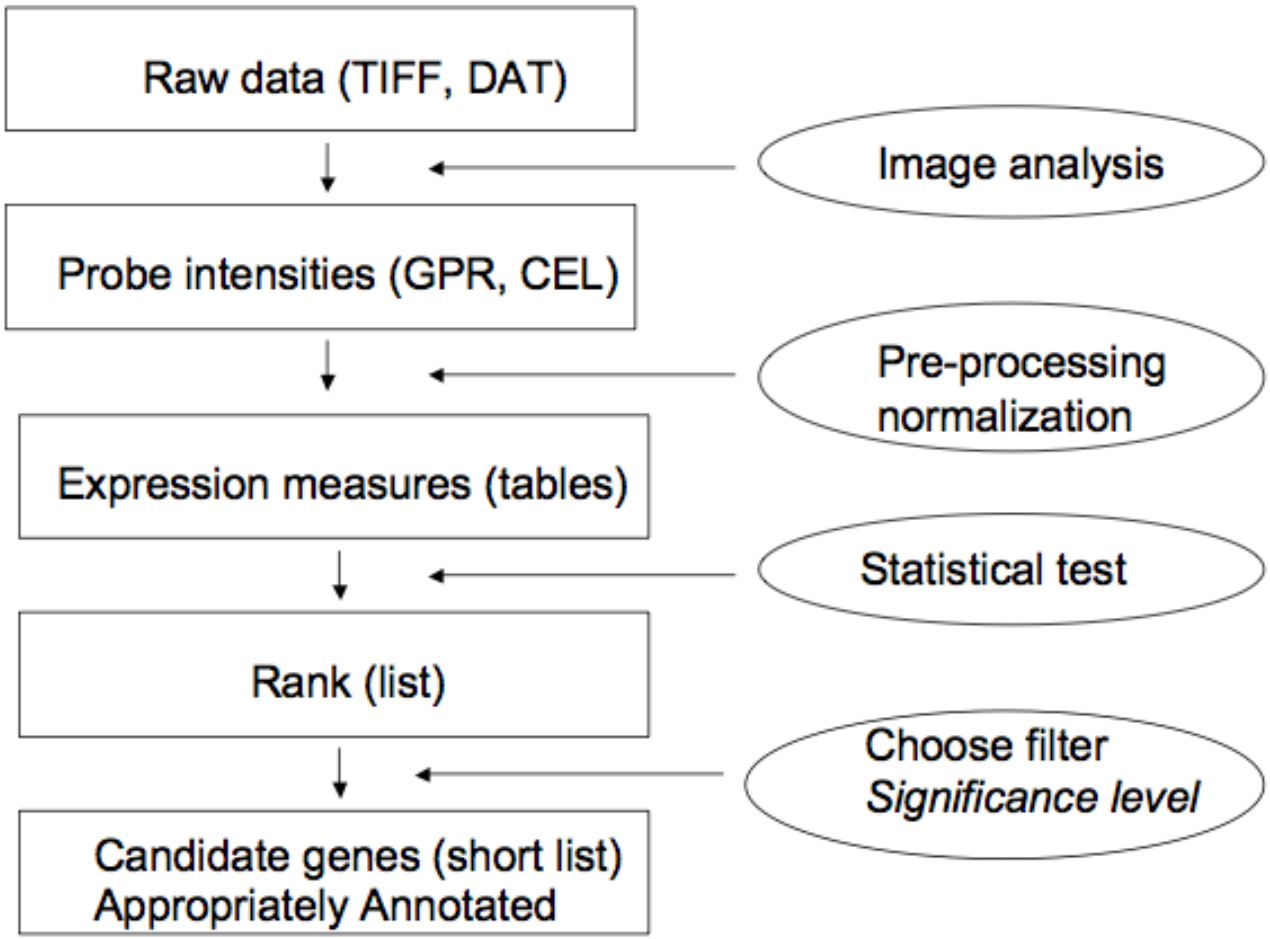
Héctor Corrada Bravo
based on slides developed by
Rafael A. Irizarry and Hao Wu

Computational Systems Biology and Functional Genomics
Spring 2012

Differential Expression and Annotation

Finding differentially expressed genes.

Workflow


Raw data (TIFF, DAT)
Image analysis
Probe intensities (GPR, CEL)
Pre-processing
normalization
Expression measures (tables)
Statistical test
Rank (list)
Choose filter
Significance level
Candidate genes (short list)
Appropriately Annotated

The dataset

- We will use a breast cancer dataset obtained from: `http://pierotti.group.ifom-ieo-campus.it/biocdb/data/experiment/` where you can find a large collection of free microarray data sets for breast and ovarian cancer.
- The data for this experiment have already been normalized for us. This is an extremely important step you have to be very careful about, but we will skip it today.

Setup

- Let's start by loading packages

```
> library(Biobase)
> library(genefilter)
> library(affy)
```

- and the data

```
> load("chang03.rda")
> show(chang03)
ExpressionSet (storageMode: lockedEnvironment)
assayData: 12625 features, 24 samples
  element names: exprs, exprs.mas5, exprs.mas5.err
protocolData: none
phenoData
  rowNames: GSM4901 GSM4902 ...
    GSM4924 (24 total)
  varLabels: Patient disease.state
    ... experiment.type (15 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
  pubMedIds: 12907009
```

OOP

- Object oriented programming (OOP) is a powerful programming paradigm. Object oriented programming allows us to construct modular pieces of code which can be utilized as building blocks for large systems.
- R is a functional language, not particular object oriented, but support exists for programming in an object oriented style.
- The Bioconductor project uses OOP extensively, and it is important to understand basic features to work effectively with Bioconductor.
- `R` has two different OOP systems, known as S3 and S4. These two systems are quite different, with S4 being more object oriented, but sometimes harder to work with.
- In both systems, the object oriented system is much more method-centric than languages like Java and Python - R's system is very Lisp-like.

Why?

As a (Bioconductor) user, it is important to have an understanding of S3 and S4.

- In order to understand and use a package unfamiliar to you.
- In order to diagnose and fix when things break (as they tend to do).

Pay close attention to how to get help, how to examine the definition of a class and a method, and how to examine the code.

S3 Classes

First we will take a look at S3 classes. Base `R` uses S3 more or less exclusively.

- "The greatest use of object oriented programming in R is through print methods, summary methods and plot methods. These methods allow us to have one generic function call, plot say, that dispatches on the type of its argument and calls a plotting function that is specific to the data supplied." – R Manual (referring to the S3 system).
- An S3 class is (most often) a `list` with a `class` attribute. It is constructed by the following code `class(obj) <- "class.name"`.

S3 Classes

```
> xx <- rnorm(1000)
> class(xx)
> plot(xx)
> yy <- ecdf(xx)
> class(yy)
> plot(yy)
> plot
> plot.ecdf
> plot.default
> methods("plot")
> getS3method("plot", "histogram")
```

What plot does, depends on the `class` of the `x` argument. It is a method. `plot.ecdf` is the `ecdf` method for `plot`.

Constructing a new S3 Class

```
> jim <- list(height = 2.54*12*6/100, weight = 180/2.2, name = "
> class(jim) <- "person"
> class(jim)
```

We have now made an object of class `person`. We now define a print method.

```
> print(jim)
> print.person <- function(x, ...) {
+     cat("name:", x$name, "\n")
+     cat("height:", x$height, "meters", "\n")
+     cat("weight:", x$weight, "kilograms", "\n")
+ }
> print(jim)
```

Note the method/class has the "dot" naming convention of `method.class`.

S3 classes are not robust

```
> fit <- lm(rnorm(100) ~ 1)
> class(fit)
> print(fit)
> class(fit) <- "something"
> print(fit)
> class(fit) <- "person"
> print(fit)
```

In case print does not have a method for the class, it dispatches to the default method, print.default.
S3 does not have the concept of type checking – there is no way to formally define a class and ensure that the object conform to the definition.

S3 classes and the help system

S3 classes are traditionally documented in the help page for the function that creates them. Example: `lm`.
Methods have a generic help page (often not very informative), sometimes with more specific help under `?method.class`. Example: `plot.lm`.

Inheritance in S3

In S3, inheritance is achieved by the class attribute being a vector. A canonical example is

```
> fit <- glm(rpois(100, lambda = 1) ~ 1, family = "poisson")
> class(fit)
> methods("residuals")
> methods("model.matrix")
```

If no method for the first is found, the second class is checked.

Useful S3 Method Functions

- `methods("print")` and `methods(class = "lm")`
- `getS3method("print","person")` : Gets the appropriate method associated with a class, useful to see how a method is implemented. Try: `getS3method("residuals", "lm")`.
- In emacs using ESS or in the R Gui we can use TAB completion to determine what methods are available. This can be quite useful for getting help on the specific method (we will see more of this later). In TextMate use ctrl-shift-H.
- Sometimes, methods are non-visible, because they are hidden in a namespace. Use getS3method or getAnywhere to get around this.

```
> residuals.HoltWinters
> getS3method("residuals.HoltWinters")
> getAnywhere("residuals.HoltWinters")
```

Replacement Methods

- As we have already seen R has a somewhat strange type of function that allows us to modify objects in place.
- It is uncommon to define new replacement functions, however they are used quite frequently in day to day programming of R.
- Two examples are: names and colnames. Type "colnames" into the R window and hit "tab", notice the function "colnames<-"?

```
> a <- matrix(1:16, nrow = 4, ncol = 4)
> colnames(a) <- paste("V", 1:4, sep = ".")
> colnames(a)
> point <- list("x" = 1, "y" = 2)
> x.val <- function(x, value) {
+     x$x <- value
+ }
> "x.val<-" <- function(x, value) {
+     x$x <- value
+     return(x)
+ }
```

Replacement Methods

```
> x.val(point, 10)
> print(point)
> x.val(point) <- 10
> print(point)
```

What does the first print statement print? What about the second?

S4 classes, why?

- Although S3 classes can be quite useful and powerful and fast they do not facilitate the type of modularization and type safety that a true object oriented system intends.
- S4 classes are more a traditional object oriented system with type checking, multiple-dispatch, and inheritance.
- S4 is implemented in the methods package in base R.
- For thorough information on S4, read Chambers (1998) "Programming with data" (also known as the green book) (first chapter available at `http://www.omegahat.org/RSMethods/Intro.pdf`) or Chambers (2008) "Software for Data Analysis: Programming with R".
- There are also several good, short, tutorials on the net.

Defining an S4 class

```
> myRep <- representation(height = "numeric", weight = "numeric"
+                         name = "character")
> setClass("personS4", representation = myRep)
> getClass("personS4")
> jimS4 <- new("personS4")
> jimS4
> jimS4 <- new("personS4", height = 2.54*12*6/100, weight = 180/
> jimS4
> jimS4@name
> validObject(jimS4)
> jimS4@height <- "2"
```

Notes on the S4 class example

- It is rare for *users* to define their own S4 classes.
- The use of `new` to *instantiate* a new member of the class is not always needed, often there are explicit constructor functions (see later).
- The use of `@` to access the class *slots* is heavily discouraged, instead use *accessor* functions (see later).

Defining the print method

For completion, we define the print method for `personS4`. For S4 classes, it is not print, but rather show.

```
> setMethod("show", signature("personS4"),
+           function(object) {
+               cat("name:", object@name, "\n")
+               cat("height:", object@height, "meters", "\n")
+               cat("weight:", object@weight, "kilograms", "\n")
+               })
> jimS4
> getMethod("show", signature("personS4"))
```

S4 Generics

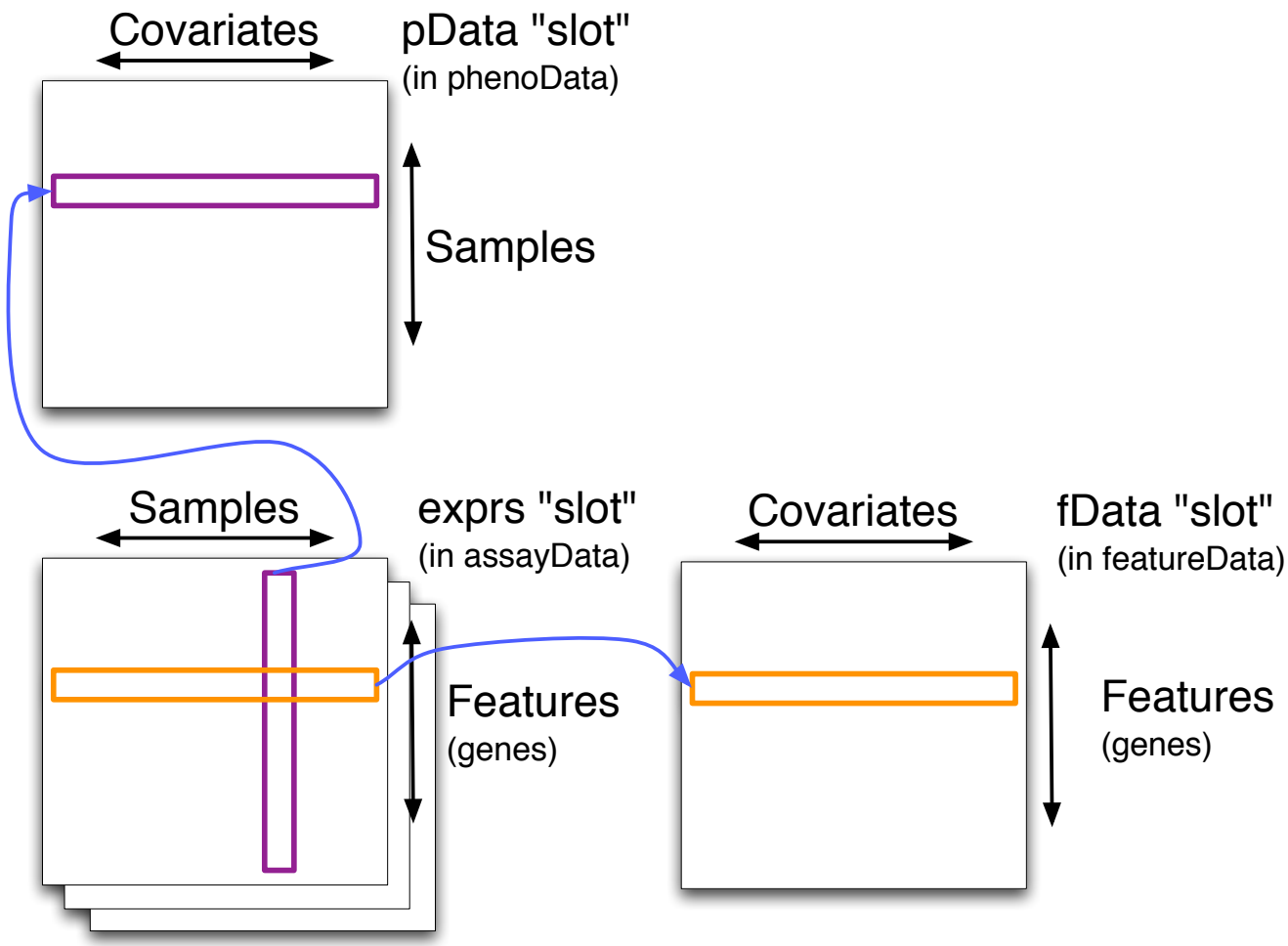
In order to make a new generic we need to call the function setGeneric.

```
> setGeneric("BMI", function(object) standardGeneric("BMI"))
> setMethod("BMI", "personS4", function(object) {
+     object@weight / object@height^2
+ })
> BMI(jimS4)
```

ExpressionSet

We will now deconstruct the ExpressionSet from the package Biobase. This is a very important – and complicated – class from Bioconductor. It will be very profitable to feel comfortable with this class, which is also an excellent example of the power of S4 (and sometimes the frustration of S4).
Some history: the ExpressionSet class is a new design, expanding the older (deprecated) exprSet class (which you still see referenced). There is a fair amount of historical baggage associated with this package.

ExpressionSet: Basic idea



Exploring Biobase

Loading

```
> require(Biobase)
> library(help = Biobase)
> getClass("ExpressionSet")
> data(sample.ExpressionSet)
> sample.ExpressionSet
> head(exprs(sample.ExpressionSet))
> head(pData(sample.ExpressionSet))
> head(fData(sample.ExpressionSet))
```

phenoData / AnnotatedDataFrame

An AnnotatedDataFrame is essentially a versioned data.frame with some descriptive labels of the columns.

```
> getClass("AnnotatedDataFrame")
> sample.phenoData <- phenoData(sample.ExpressionSet)
> sample.phenoData
> pData(sample.phenoData)
> varLabels(sample.phenoData)
> sampleNames(sample.phenoData)
> sample.phenoData$type
```

(Note the last one).
This also works directly from the ExpressionSet:

```
> pData(sample.ExpressionSet)
> varLabels(sample.ExpressionSet)
> sampleNames(sample.ExpressionSet)
> sample.ExpressionSet$type
> head(featureNames(sample.ExpressionSet))
```

Accessing the ExpressionSet

- Accessing the relevant data involves calling accessor functions. We should try to avoid ever accessing the data directly with the "@" accessor because it is less future-proof. Unlike many object oriented programming languages R does not provide a mechanism for protecting data, such as "private" member variables in many languages.
- Have a look at `?ExpressionSet` to see what other methods are available.

```
> featureNames(sample.ExpressionSet)
> sampleNames(sample.ExpressionSet)
> head(exprs(sample.ExpressionSet))
```

ExpressionSet 2

An ExpressionSet contains information

- About characteristics of the samples (`phenoData` / `pData`).
- About gene-level measurements (`assayData` / `exprs`).
- About the microarray (`featureData` / `fData`) (rarely used).

All linked together appropriately. Linking allows for easy subsetting.
The expression matrix has dimension $N_{\text{features}} \times N_{\text{arrays}}$

Subsetting ExpressionSets

We can subset the ExpressionSet object just as we can subset a matrix. Columns refer to samples and rows refer to features.

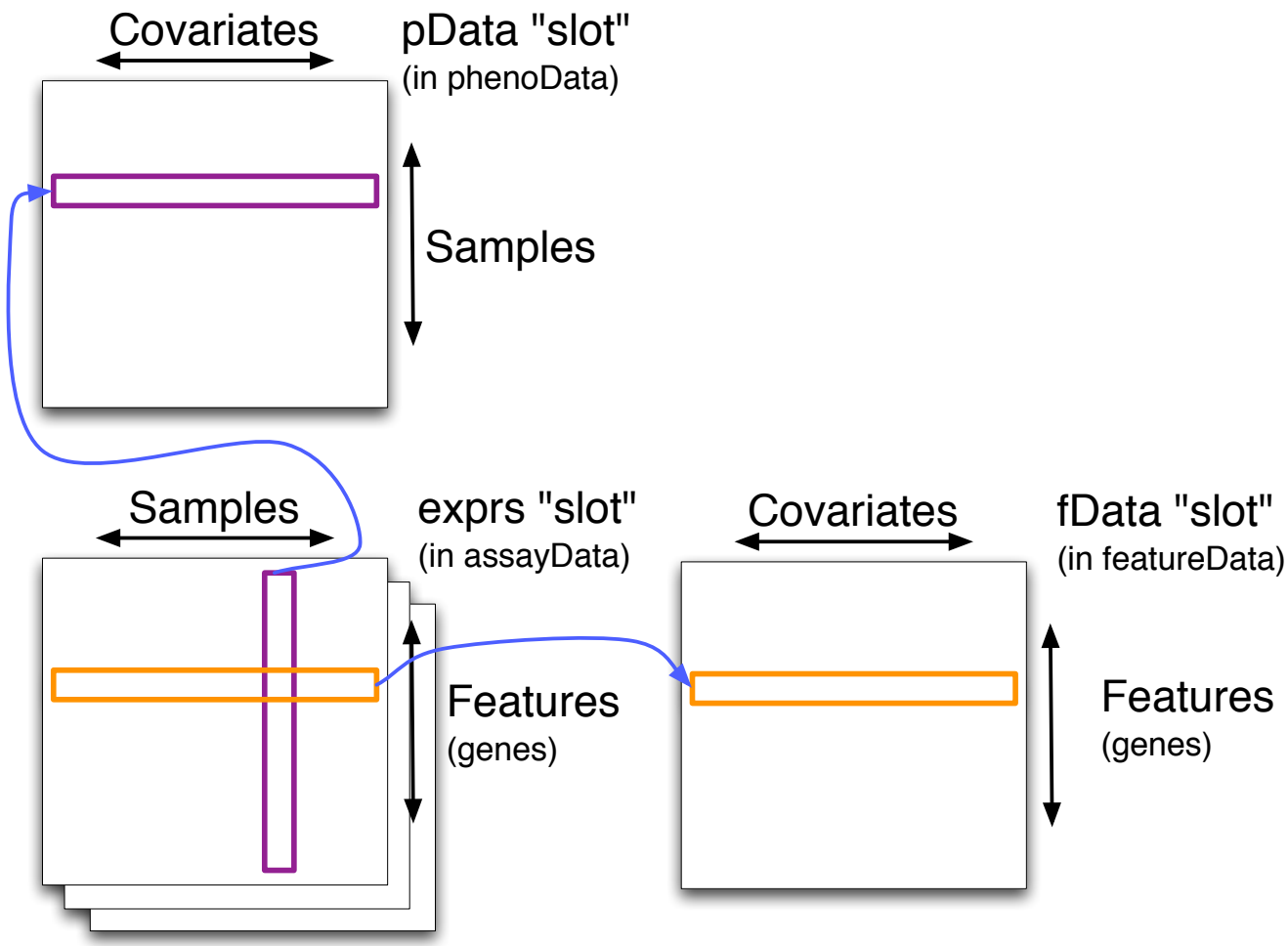
```
> Type <- phenoData(sample.ExpressionSet)$type
> cases <- grep("Case", Type)
> controls <- grep("Control", Type)
> casesEx <- sample.ExpressionSet[,cases]
> controlsEx <- sample.ExpressionSet[,controls]
```

What is the class of casesEx and controlsEx?

```
> sample.ExpressionSet[sample(nrow(sample.ExpressionSet), size =
```

Subsetting is used *a lot*!
Remember to read the "ExpressionSet Introduction" vignette in Biobase.

ExpressionSet: again



Our new dataset

Let's get information about the experiment

```
> cat(abstract(experimentData(chang03)))
```

BACKGROUND: Systemic chemotherapy for operable breast cancer substantially decreases the risk of death. Patients often have de novo resistance or incomplete response to docetaxel, one of the most active agents in this disease. We postulated that gene expression profiles of the primary breast cancer can predict the response to docetaxel. METHODS: We took core biopsy samples from primary breast tumours in 24 patients before treatment and then assessed tumour response to neoadjuvant docetaxel (four cycles, 100 mg/m2 daily for 3 weeks) by cDNA analysis of RNA extracted from biopsy samples using HgU95-Av2 GeneChip. FINDINGS: From the core biopsy samples, we extracted sufficient total RNA (3-6 microg) for cDNA array analysis using HgU95-Av2 GeneChip. Differential patterns of expression of 92 genes correlated with docetaxel response (p=0.001). Sensitive tumours had higher expression of genes involved in cell cycle, cytoskeleton, adhesion, protein transport, protein modification, transcription, and stress or apoptosis; whereas resistant tumours showed increased expression of some transcriptional and signal transduction genes. In leave-one-out cross-validation analysis, ten of 11 sensitive tumours (90% specificity) and 11 of 13 resistant tumours (85%

Our new dataset

sensitivity) were correctly classified, with an accuracy of 88%. This 92-gene predictor had positive and negative predictive values of 92% and 83%, respectively. Correlation between RNA expression measured by the arrays and semiquantitative RT-PCR was also ascertained, and our results were validated in an independent set of six patients. INTERPRETATION: If validated, these molecular profiles could allow development of a clinical test for docetaxel sensitivity, thus reducing unnecessary treatment for women with breast cancer.

Our new dataset

- Find the dimensions of the expression data

```
> dim(exprs(chang03))
[1] 12625    24
```

- Find the dimensions of the measured covariates

```
> dim(pData(chang03))
[1] 24 15
```

- Look at the names of the measured covariates

```
> names(pData(chang03))
 [1] "Patient"
 [2] "disease.state"
 [3] "Tumour.type..IMC.invasive.mammary.carcinoma..IDC.invas
 [4] "Age..years."
 [5] "Menopausal.status"
 [6] "Ethnic.origin"
 [7] "Bidimensional.tumour.size..cm."
 [8] "Clinical.axillary.nodes"
 [9] "Oestrogen..receptor.status"
[10] "Progesterone..receptor.status"
[11] "HER.2..immunhistochemical.analysis."
[12] "species"
```

Our new dataset

- Make a table of the `disease.state` variable

```
> table(pData(chang03)$disease.state)

docetaxel resistant tumor
                       14
docetaxel sensitive tumor
                       10
```

- Look at disease state by progestorone receptor status

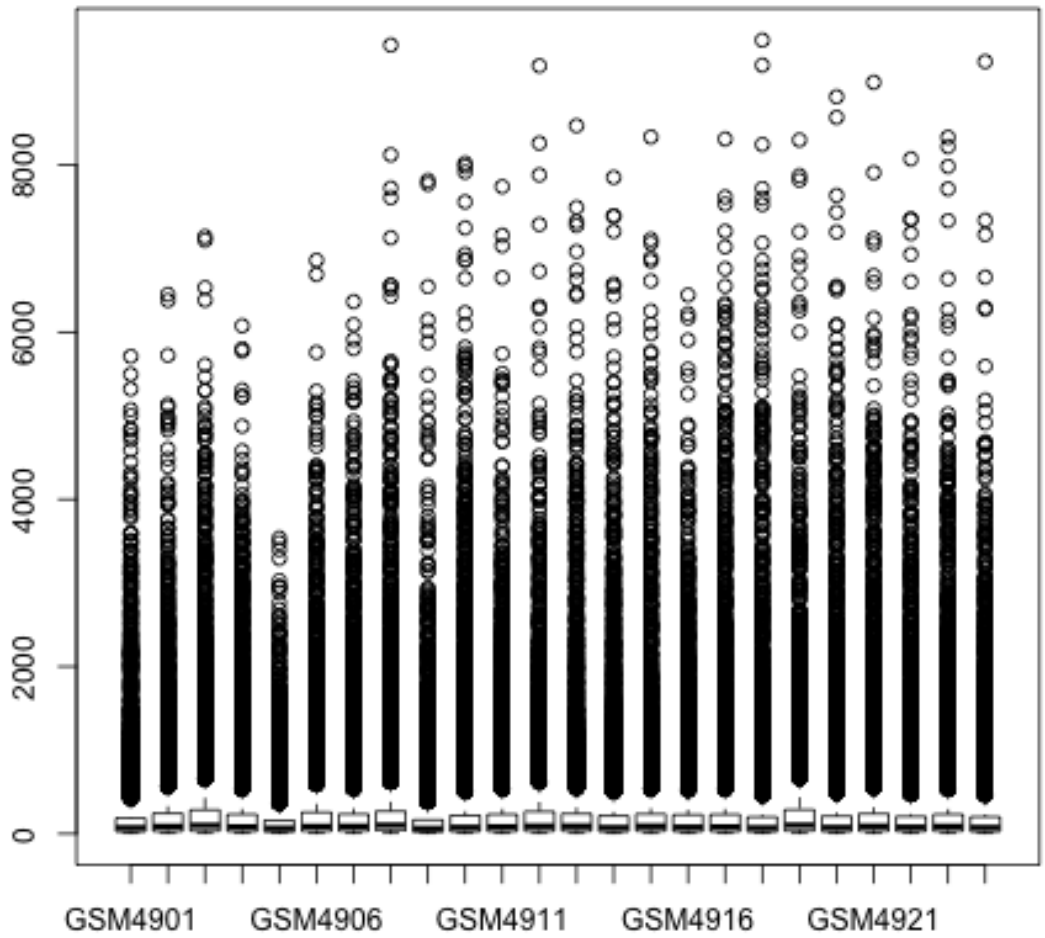
```
> table(pData(chang03)$disease.state, pData(chang03)$Progest

                                 + -
  docetaxel resistant tumor 8 6
  docetaxel sensitive tumor 6 4
```

More Exploration

Let's do a boxplot of the expression measurements

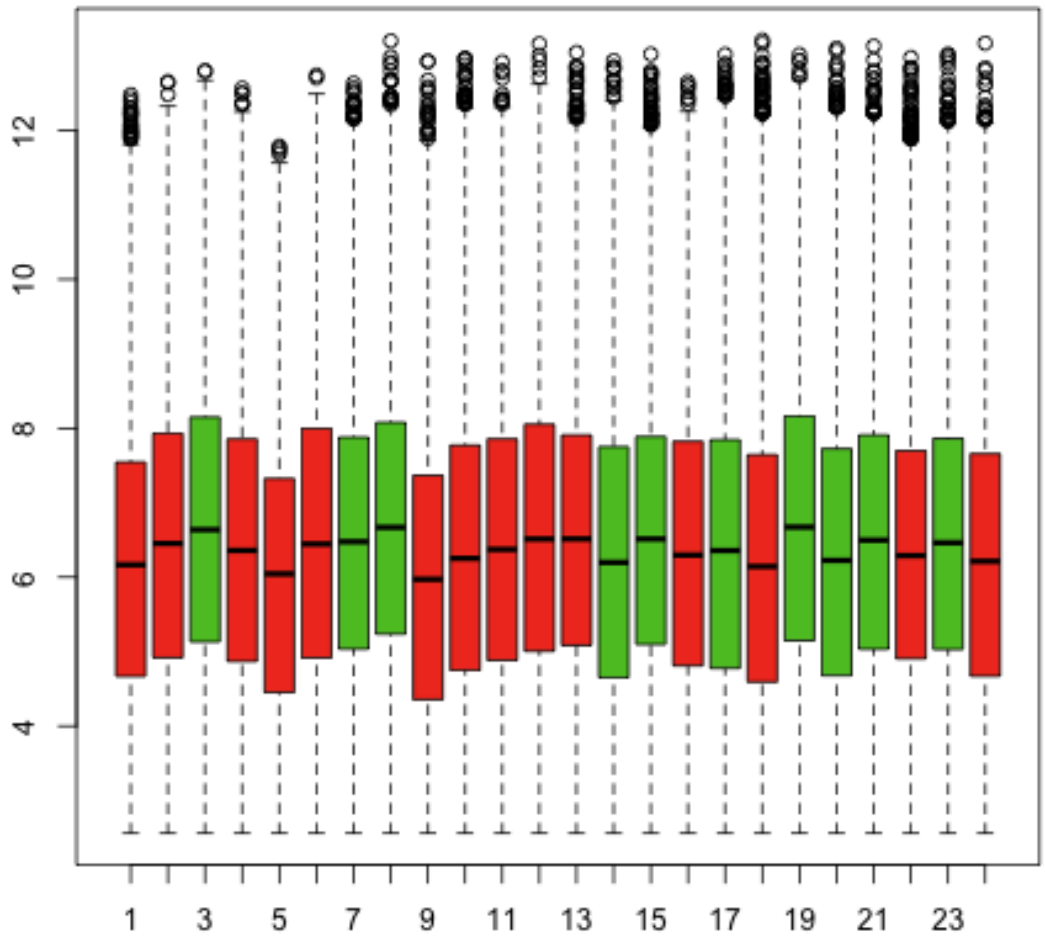
```
> boxplot(exprs(chang03))
```



More Exploration

Oh yeah, work in log space

```
> y <- log2(exprs(chang03))
> boxplot(y, col=as.numeric(pData(chang03)$disease.state)+1)
```



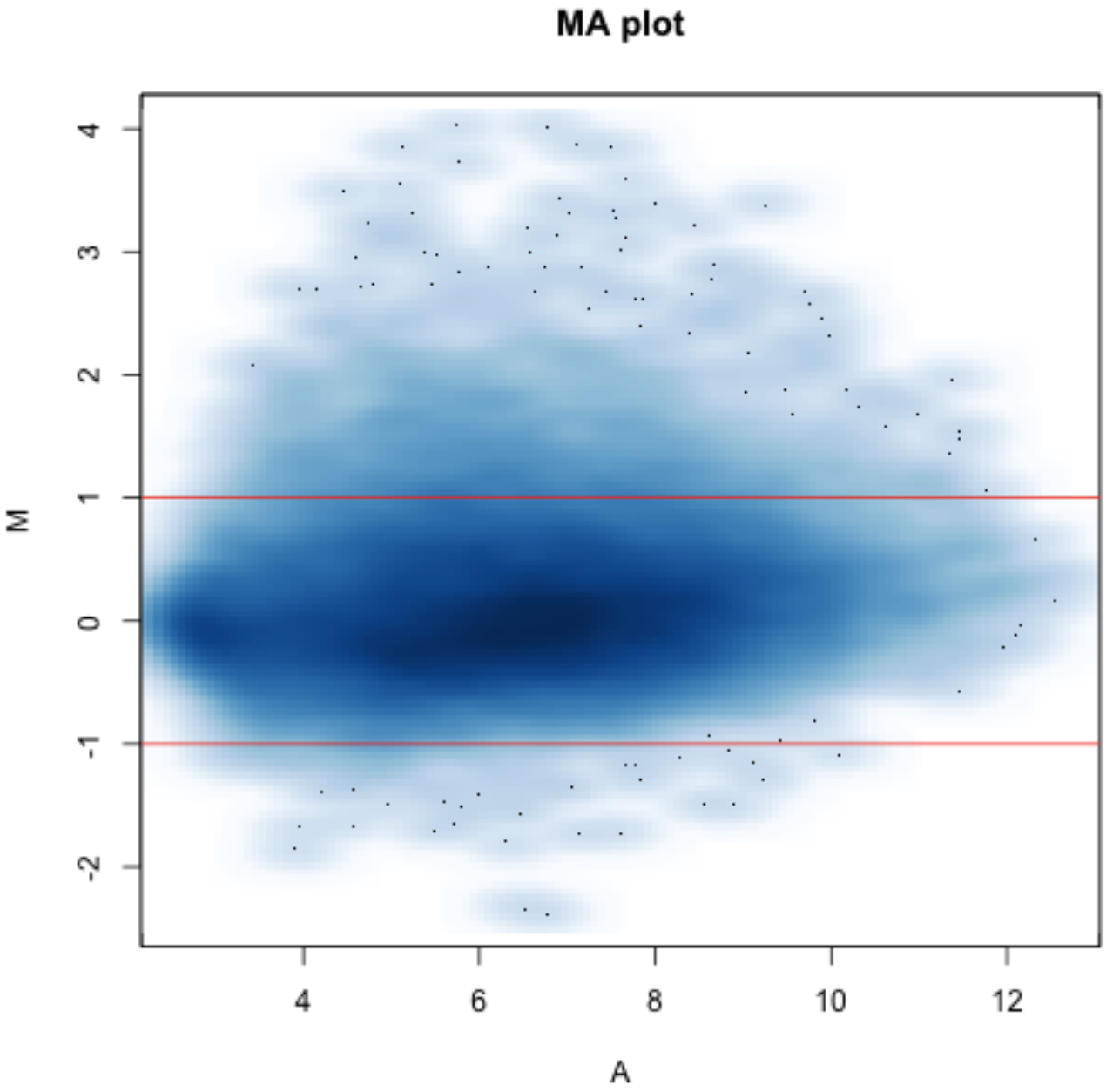
Exploration: The MA plot

- We are interested in genes with large differences between docetaxel resistant patients and docetaxel sensitive patients
- One way to measure those differences are with *old-changes, e.g., there was a two-fold increase in average expression of gene G in docetaxel resistant patients*
- So, why not look at average log ratios?
- We can make MA plots:
  - M: difference in average log intensities
  - A: average log intensities

More Exploration: MA plot

```
> Index <- as.numeric(pData(chang03)$disease.state)
> d <- rowMeans(y[,Index==2]) - rowMeans(y[, Index==1])
> a <- rowMeans(y)
> smoothScatter(a, d, main="MA plot", xlab="A", ylab="M")
> abline(h=c(-1,1), col="red")
```

More Exploration: MA plot



Differential Expression Analysis

- Observations: $X_1, X_2, \ldots, X_M$ and $Y_1, Y_2, \ldots, Y_N$
- Averages:

$$\bar{X} = \frac{1}{M}\sum_{i=1}^{M} X_i \qquad \bar{Y} = \frac{1}{N}\sum_{i=1}^{N} Y_i$$

- Variances:

$$s_X^2 = \frac{1}{M-1}\sum_{i=1}^{M}(X_i - \bar{X})^2$$

$$s_Y^2 = \frac{1}{N-1}\sum_{i=1}^{N}(Y_i - \bar{Y})^2$$

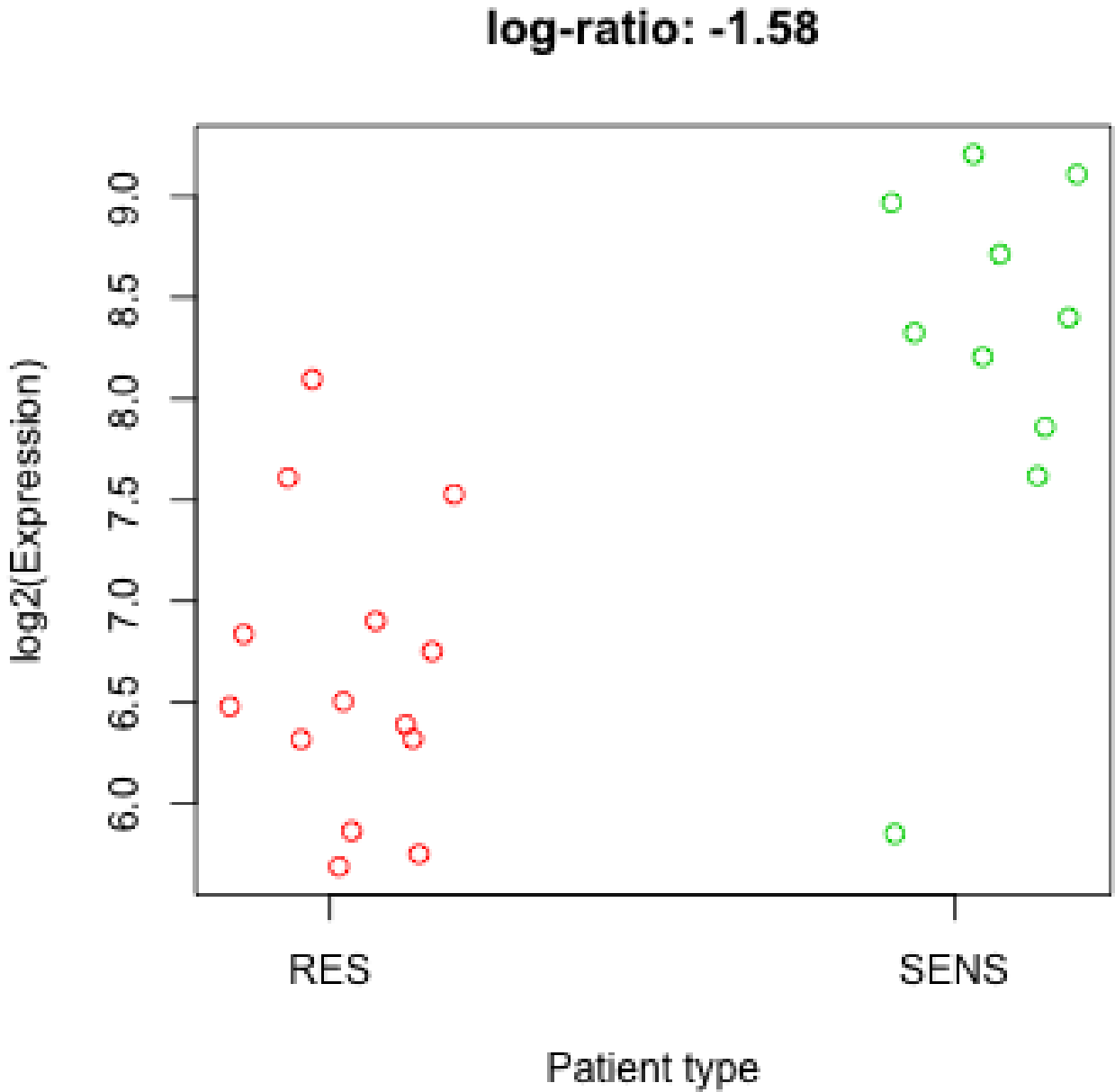
Differential Expression Analysis

Let's look at one gene to decipher the math

```
> g <- y[23,]
> m <- mean(g[Index==1])-mean(g[Index==2])
> plot(jitter(Index), g, col=Index+1, xaxt="n", xlab="Patient ty
> axis(1, labels=c("RES", "SENS"), at=1:2)
```

Differential Expression Analysis

Let's look at one gene to decipher the math



Differential Expression Analysis

The t-statistic:

$$\frac{\bar{Y} - \bar{X}}{\sqrt{\frac{s_Y^2}{N} + \frac{s_X^2}{M}}}$$

Estimating the variance

- If different genes (or probes) have different variation, then it is not a good idea to use average log ratios even if we do care about significance
- Under a random model, we need to estimate SD
- The t-test divides by SD
- But, with few replicates, estimates of SD are not stable
- This explains why the t-test is not powerful
- There are many proposals for estimating variation
- Many borrow strength across genes
- Empirical Bayes approaches are popular
- SAM, an ad-hoc procedure, is even more popular

Differential Expression

Let's use limma (Empirical Bayes) moderated t-test

```
> library(limma)
> design <- model.matrix(~factor(chang03$disease.state))
> fit <- lmFit(y, design)
> ebayes <- eBayes(fit)
```

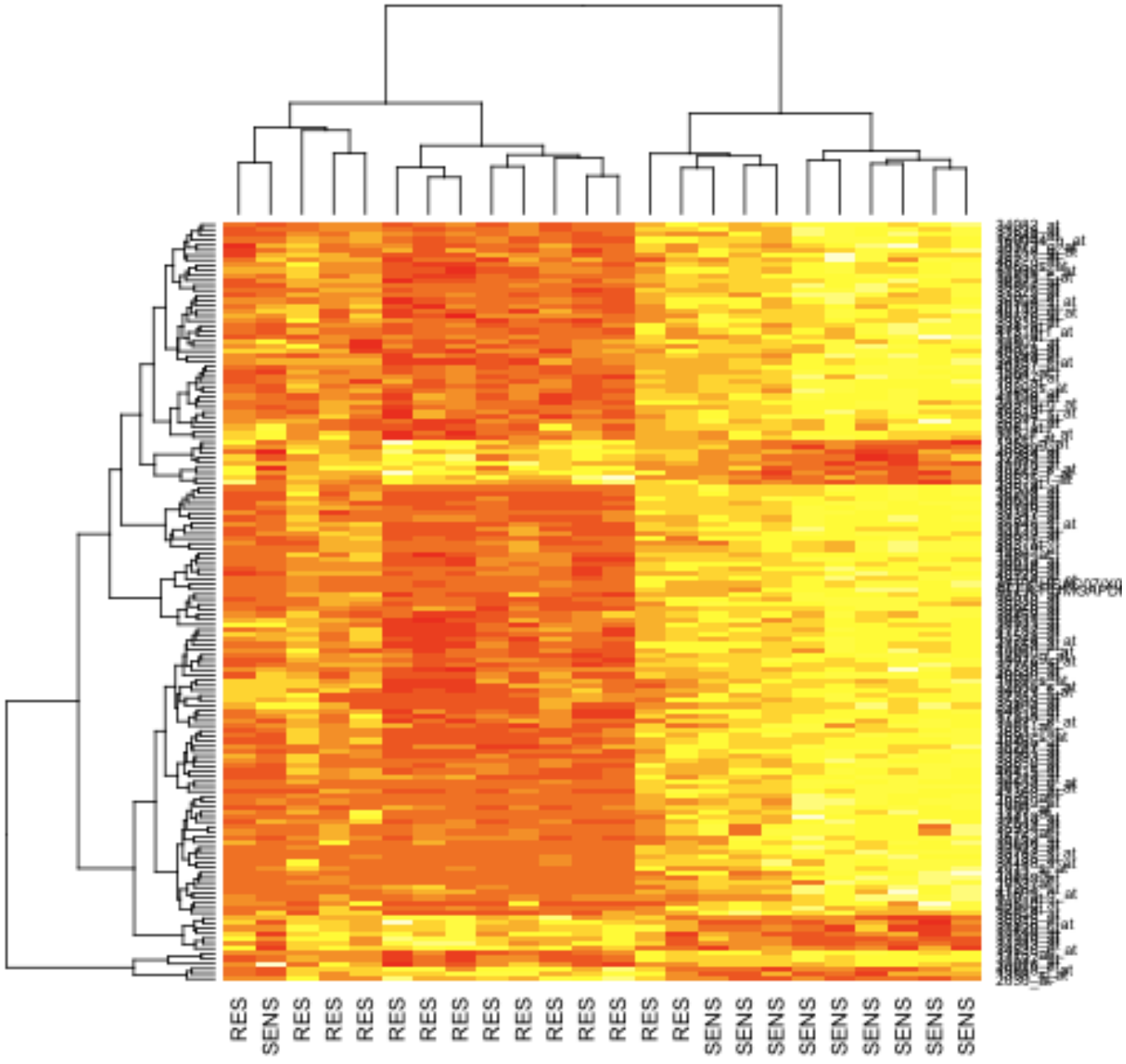
Heatmap

Let's look at the top 150 differentially expressed genes

- The function topTable makes this very easy
- Note that we adjust for false discovery rate (fdr). Another fundamentally important issue in genomic data analysis I won't discuss today
- Then make a heatmap

```
> tab <- topTable(ebayes, coef=2, adjust="fdr", n=150)
> labCol <- c("RES", "SENS")[Index]
> heatmap(y[tab$ID,], labCol=labCol)
```

Heatmap


RES
SENS
RES
RES
RES
RES
RES
RES
RES
RES
RES
RES
RES
RES
SENS
SENS
SENS
SENS
SENS
SENS
SENS
SENS
SENS

Annotation

- One of the largest challenges in analyzing genomic data is associating the experimental data with the available biological metadata, e.g., sequence, gene annotation, chromosomal maps, literature.
- AND MAKING THAT DATA AVAILABLE FOR COMPUTATION
- Bioconductor provides three main packages for this purpose:
  - annotate (end-user)
  - AnnBuilder (developer)
  - annaffy (end-user)

WWW Resources

- Nucleotide databases: e.g. GenBank
- Gene databases: e.g. Entrez Gene, UniGene
- Protein sequence and structure databases: e.g. SwissProt, Protein DataBank (PDB)
- Literature databases: PubMed, OMIM
- Chromosome maps: e.g., NCBI Map Viewer
- Pathways: e.g., KEGG
- Entrez is a search and retrieval system that integrates information from databases at NCBI (National Center for Biotechnology Information)
- If you know of some we should be using, please let us know

annotate: matching IDs

Important tasks

- Associate manufacturers or in-house probe identifiers to other available identifiers, e.g.
  - Affymetrix IDs $\longrightarrow$ Entrez Gene IDs
  - Affymetrix IDs $\longrightarrow$ GenBank accession number
- Associate probes with biological data such as chromosomal position, pathways

annotate: matching IDs

Affy ID	41046_s_at
Entrez Gene ID	9203
GenBank accession #	X95808
Gene symbol	ZMYM3
PubMed ID	8817323, 8889548, 9205841
Chromosomal Location	X, Xq13.1

Annotation data packages

- Bioconductor provides annotation data packages that contain many different mappings to interesting data
  - Mappings between Affy IDs and other probe IDs: hgu95av2.db for HGU95Av2 GeneChip series, also, hgu133a.db, hu6800.db, etc.
  - Affy CDF data packages, e.g. hgu95av2cdf
  - Probe sequence data packages, e.g. hgu95av2probe
- These packages are updated and expanded regularly as new data becomes available
- They can be installed through biocLite()
- AnnBuilder provides tools for building annotation data packages

annotate: matching IDs

- Find out and load annotation package we need

```
> annotation(chang03)
[1] "hgu95av2"
> library(annotate)
> library(hgu95av2.db)
```

- Let's get matching IDs for the first 3 probesets on our list using the lookUp function

```
> probeset <- as.character(tab$ID[1:3])
> lookUp(probeset, "hgu95av2.db", "ACCNUM")
$`36125_s_at`
[1] "L38696"

$`33781_s_at`
[1] "AF075599"

$`40549_at`
[1] "L04658"
```

annotate: matching IDs

```
> lookUp(probeset, "hgu95av2.db", "SYMBOL")
$`36125_s_at`
[1] "RALY"

$`33781_s_at`
[1] "UBE2M"

$`40549_at`
[1] "CDK5"
> lookUp(probeset, "hgu95av2.db", "GENENAME")
$`36125_s_at`
[1] "RNA binding protein, autoantigenic (hnRNP-associated wi

$`33781_s_at`
[1] "ubiquitin-conjugating enzyme E2M"

$`40549_at`
[1] "cyclin-dependent kinase 5"
```

annotate: matching IDs

```
> lookUp(probeset, "hgu95av2.db", "UNIGENE")
$`36125_s_at`
[1] "Hs.136947"

$`33781_s_at`
[1] "Hs.406068"

$`40549_at`
[1] "Hs.647078"
> lookUp(probeset, "hgu95av2.db", "CHR")
$`36125_s_at`
[1] "20"

$`33781_s_at`
[1] "19"

$`40549_at`
[1] "7"
```

annotate: matching IDs

```
> lookUp(probeset, "hgu95av2.db", "CHRLOC")
$`36125_s_at`
      20
32581732

$`33781_s_at`
       19
-59067080

$`40549_at`
         7          7
-150750903 -150750899
> lookUp(probeset, "hgu95av2.db", "MAP")
```

annotate: matching IDs

```
$`36125_s_at`
[1] "20q11.21-q11.23"

$`33781_s_at`
[1] "19q13.43"

$`40549_at`
[1] "7q36"
```

```
> sapply(lookUp(probeset, "hgu95av2.db", "PMID"), head)
```

```
     36125_s_at 33781_s_at 40549_at
[1,] "7533788"  "9694792"  "1181841"
[2,] "8125298"  "10207026" "1330687"
[3,] "9373149"  "10722740" "1639063"
[4,] "9376072"  "10828074" "7566346"
[5,] "10500250" "12477932" "7834371"
[6,] "11780052" "12522145" "7949095"
```

annotate: matching IDs

For some common IDs, you can use more user-friendly functions provided by annotate

```
> getSYMBOL(probeset, "hgu95av2.db")
```

```
36125_s_at 33781_s_at   40549_at
    "RALY"    "UBE2M"     "CDK5"
```

```
> gg <- getGO(probeset, "hgu95av2.db")
> getGOdesc(gg[[1]][[1]]$GOID, "ANY")
```

```
$`GO:0000398`
GOID: GO:0000398
Term: nuclear mRNA splicing, via
    spliceosome
Ontology: BP
Definition: The joining together of
    exons from one or more primary
    transcripts of nuclear
    messenger RNA (mRNA) and the
    excision of intron sequences,
```

annotate: matching IDs

```
    via a spliceosomal mechanism,
    so that mRNA consisting only of
    the joined exons is produced.
Synonym: mRNA splicing
Synonym: nuclear mRNA splicing via
    U12-type spliceosome
Synonym: nuclear mRNA splicing via
    U2-type spliceosome
Synonym: pre-mRNA splicing
Synonym: splicing AT-AC intron
Synonym: splicing GT-AG intron
Synonym: GO:0006374
Synonym: GO:0006375
Secondary: GO:0006374
Secondary: GO:0006375
```

annotate: PubMed example

- Let's use all the genes on our list

  ```
  > probenames <- as.character(tab$ID)
  ```

- Load the XML package, and get pubmed abstracts for the first 2 genes

  ```
  > library(XML)
  > absts <- pm.getabst(probenames[1:2], "hgu95av2.db")
  > absts[[1]][[1]]
  ```

  ```
  An object of class 'pubMedAbst':
  Title: Epstein-Barr virus-induced autoimmune responses. I.
       Immunoglobulin M autoantibodies to proteins mimicking a
       mimicking Epstein-Barr virus nuclear antigen-1.
  PMID: 7533788
  Authors: JH Vaughan, JR Valbracht, MD Nguyen, HH Handley, RS
       Patrick, GH Rhodes
  Journal: J Clin Invest
  Date: Mar 1995
  ```

- Let's look at the titles

annotate: PubMed example

```
> titl <- pm.titles(absts[1])
> strwrap(titl, simplify=FALSE)

[[1]]
[1] "c(\"Epstein-Barr virus-induced autoimmune responses. I.
[2] "autoantibodies to proteins mimicking and not mimicking
[3] "virus nuclear antigen-1.\", \"Oligo-capping: a simple m
[4] "the cap structure of eukaryotic mRNAs with oligoribonuc
[5] "\"Construction and characterization of a full length-en
[6] "5'-end-enriched cDNA library.\", \"The p542 gene encodes
[7] "that cross-reacts with EBNA-1 of the Epstein Barr virus
[8] "be a heterogeneous nuclear ribonucleoprotein.\","
```

annaffy

- Provides simplified mappings between Affymetrix IDs and annotation data
- Relies on chip-level annotation packages created by AnnBuilder
- Supplies functions to produce mappings for almost all environments in a given annotation package
- Primary function of annaffy is to produce very nice HTML or text tables containing
  - Links to databases
  - Statistics
  - Expression measures (color-coded to intensity for easy viewing)

annaffy

- Load some more annotation databases we will use

```
> library("KEGG.db")
> library("GO.db")
> library("annaffy")
```

- Make a table

```
> atab <- aafTableAnn( probenames, "hgu95av2.db", aaf.handle
```

- Save it as HTML

```
> saveHTML(atab, file="report2.html")
> browseURL("report2.html")
```

Examining the R session

```
> sessionInfo()

R version 2.14.1 (2011-12-22)
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

locale:
[1] en_US.utf-8/en_US.utf-8/en_US.utf-8/C/en_US.utf-8/en_US.utf-

attached base packages:
[1] stats     graphics  grDevices
[4] utils     datasets  methods
[7] base

other attached packages:
 [1] GO.db_2.6.1
 [2] hgu95av2.db_2.6.3
 [3] org.Hs.eg.db_2.6.4
 [4] RSQLite_0.11.1
 [5] DBI_0.2-5
```

Examining the R session

```
 [6] annotate_1.32.1
 [7] AnnotationDbi_1.16.10
 [8] limma_3.10.1
 [9] genefilter_1.36.0
[10] affy_1.32.0
[11] Biobase_2.14.0
[12] RColorBrewer_1.0-5

loaded via a namespace (and not attached):
[1] affyio_1.22.0
[2] BiocInstaller_1.2.1
[3] IRanges_1.12.5
[4] preprocessCore_1.16.0
[5] splines_2.14.1
[6] survival_2.36-10
[7] tools_2.14.1
[8] xtable_1.6-0
[9] zlibbioc_1.0.0
```