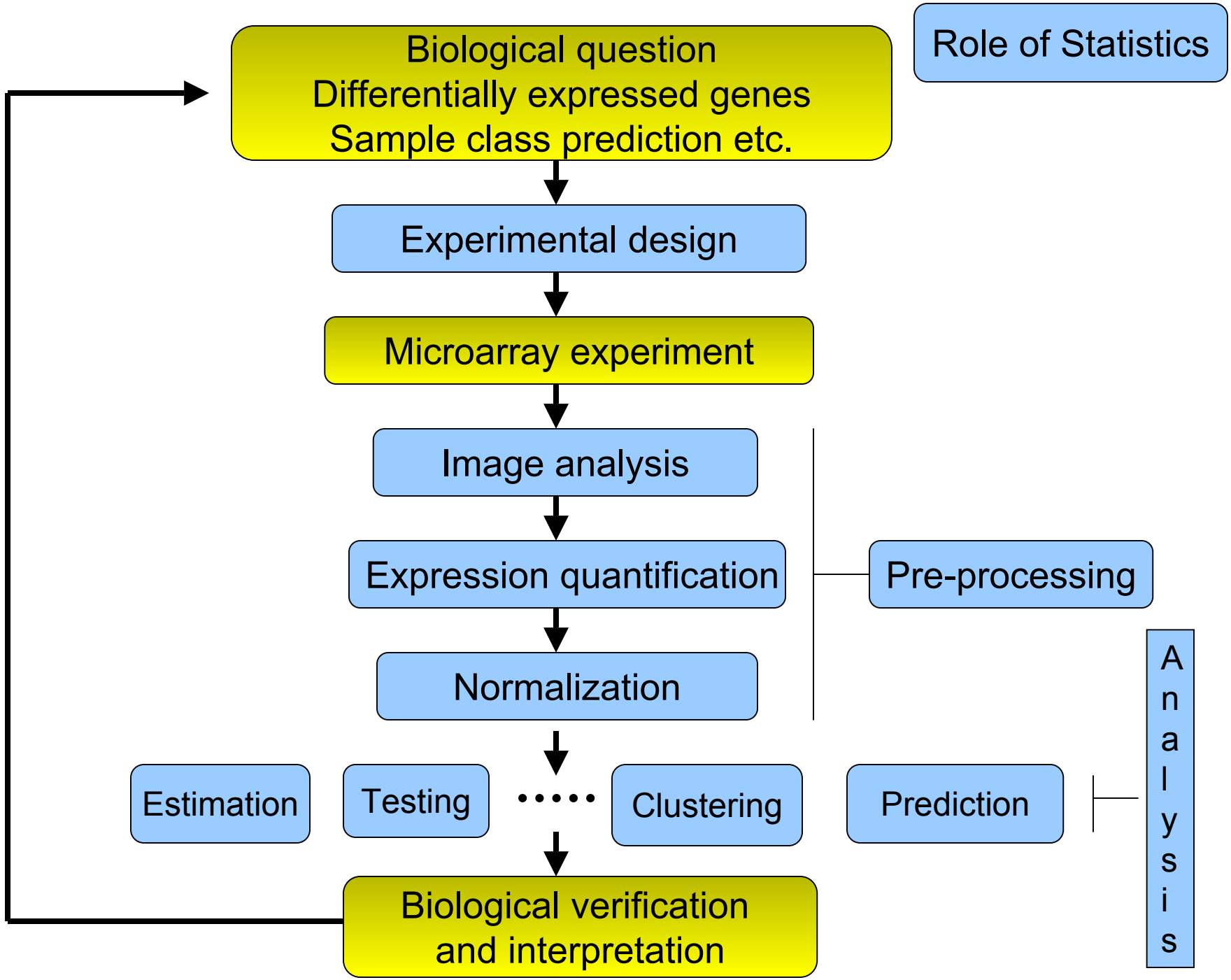
Part III. Overview of the Bioconductor project

Sandrine Dudoit and

Robert Gentleman


Role of Statistics
Biological question
Differentially expressed genes
Sample class prediction etc.
Experimental design
Microarray experiment
Image analysis
Expression quantification
Normalization
Pre-processing
Estimation
Testing
.....
Clustering
Prediction
Analysis
Biological verification
and interpretation

Statistical computing

Everywhere …

- Statistical design and analysis:
  - image analysis, normalization, estimation, testing, clustering, prediction, etc.

- Integration with biological information resources, in house and external databases:
  - gene annotation (GenBank, LocusLink);
  - literature (PubMed);
  - graphical (pathways, chromosome maps).

Computing needs

- Access to a broad range of statistical and graphical methods:

  diagnostic plots, linear and non-linear modeling, survival analysis, multiple testing, model selection, prediction, cluster analysis, resampling, etc.
- Tools for integrating biological metadata in the analysis of microarray data.
- Extensible, scalable, and interoperable software.

Bioconductor

- Bioconductor is an open source and open development software project for the analysis and comprehension of genomic data.

- Software and documentation are available from www.bioconductor.org.

Bioconductor

The broad goals of the project are

- to enable sound and powerful statistical analyses in genomics;
- to provide a computing platform that allows the rapid design and deployment of high-quality software;
- to develop a computing environment for both biologists and statisticians.

Bioconductor

- The project was started in the Fall of 2001 by Robert Gentleman, at the Biostatistics Unit of the Dana Farber Cancer Institute.
- There are currently 21 core developers.
- The first release of 15 packages occurred on May 2nd, 2002.

R

- Most of the early developments are in the form of R packages.

- R is a widely used open source language and environment for statistical computing and graphics

  - GNU's S-Plus.

R

- R is available from www.r-project.org.
- R is available for Unix, Windows, and Macintosh computers.
- Comprehensive R Archive Network - CRAN - www.cran.r-project.org: repository of software packages for a broad range of statistical and graphical techniques.

Bioconductor packages

Release 1.0, May 2nd, 2002

- General infrastructure:
  `Biobase`, `rhdf5`, `tkWidgets`.
- Annotation:
  `annotate`, `AnnBuilder`.
- Graphics:
  `geneplotter`.
- Pre-processing for Affymetrix oligonucleotide chip data:
  `affy`.
- Pre-processing for cDNA microarray data:
  `marrayClasses`, `marrayInput`, `marrayNorm`, `marrayPlots`.
- Differential gene expression:
  `edd`, `genefilter`, `multtest`, `ROC`.

Bioconductor

- Object-oriented class/method design. Allows efficient representation and manipulation of large and complex biological datasets of multiple types.
- Widgets. Small-scale graphical user interfaces, allowing point & click access to specific analysis tasks.
- E.g. File browsing and selection for data input.

Bioconductor

- Interactive tools for linking experimental results to annotation and literature WWW resources in real time.

  E.g. PubMed, GenBank, LocusLink.

- Scenario. For a list of differentially expressed genes obtained from **`multtest`** or **`genefilter`**, use the **`annotate`** package
  - to retrieve and search PubMed abstracts for these genes;
  - to generate an HTML report with links to LocusLink for each gene.

Bioconductor training

Extensive documentation and training resources for R and Bioconductor are available on the WWW.

- R manuals and tutorials are available from CRAN.
- R help system
  - detailed on-line documentation, available in text, HTML, PDF, and LaTeX formats;
  - e.g. **`help(genefilter), ?pubmed.`**
- R demo system
  - user-friendly interface for running demonstrations of R scripts;
  - e.g. **`demo(marrayPlots)`**, **`demo(affy).`**

Bioconductor training

- R vignette system
  - comprehensive repository of step-by-step tutorials covering a wide variety of computational objectives in `/doc` subdirectory;
  - documents generated using the **`Sweave`** function from the **`tools`** package;
  - integrated statistical documents intermixing text, code, and code output (textual and graphical);
  - documents can be automatically updated if either data or analyses are changed.

- Bioconductor short courses
  - modular training segments on software and statistical methodology;
  - lectures and computer labs available on WWW for self-instruction.

R programming

- In order to deliver high quality software, the Bioconductor project relies on a few programming techniques that might not be familiar
  - environments and closures;
  - object oriented programming.

- We review these here for interested programmers (understanding them is not essential but is often very helpful).

Environments and closures

- An environment is an object that contains bindings between symbols and values.
- It is very similar to a hash table.
- Environments can be accessed using the following functions
  - get a listing of objects in the environment **`e`**

    ```
    ls(env=e)
    ```
  - get the value of the object with name **`x`** in the environment **`e`**

    ```
    get("x", env=e)
    ```
  - assign to the name **`x`** the value **`y`** in the environment **`e`**

    ```
    assign("x",y,env=e)
    ```

Environments and closures

- Since these operations are used a great deal in Bioconductor we have provided two helper functions
  - **`multiget`**
  - **`multiassign`**

- These functions get and assign multiple values into the specified environment.

Environments and closures

- Environments can be associated with functions.
- When an environment is associated with a function, then that environment is used to obtain values for any unbound variables.
- The term closure refers to the coupling of the function body with the enclosing environment.
- The **`annotate`**, **`genefilter`**, and other packages take advantage of environments and closures.

Environments and closures

```
x <- 4

e1 <- new.env()
assign("x",10, env=e1)
f <- function() x
environment(f) <- e1

x   # returns 4
f() # returns 10!
```

Object oriented programming

- The Bioconductor project has adopted the OOP paradigm presented in *Programming with Data*, J. M. Chambers, 1998.

- Tools for programming using the class/method mechanism are provided in the `methods` package.

OOP

- A class provides a software abstraction of a real world object. It reflects how we think of certain objects and what information these objects should contain.
- A class defines the structure, inheritance, and initialization of objects.
- Classes are defined in terms of slots which contain the relevant data.
- An object is an instance of a class.

OOP

- A method is a function that performs an action on data (objects).
- A generic function is a dispatcher, it examines its arguments and determines the appropriate method to invoke.
- Examples of generic functions include **`plot`**, **`summary`**, **`print`**.

OOP

- It is important to realize that when calling a generic function (such as `plot`), the actions performed depend on the class of the arguments.
- Methods define how a particular function should behave depending on the class of its arguments.
- Methods allow computations to be adapted to particular data types, i.e., classes.

OOP

- The **`methods`** package contains a number of functions for defining new classes and methods (e.g. **`setClass`**, **`setMethod`**) and for working with these classes and methods.

- A tutorial is available at http://www.omegahat.org/RSMethods/index.html

OOP

- To obtain documentation (on-line help) about
  - a class: `class?classname`

    so, **`class?exprSet`**, will display the help file for the **`exprSet`** class.
  - a method: `methods?methodname`

    so, **`methods?print`**, will display the help file for the **`print`** methods.

OOP

```
> x <- 1:10
> y <- 2*x + 1 + rnorm(10)

> class(x)
[1] "integer"
> plot(x,y)

> fit <- lm(y ~ x)
> class(fit)
[1] "lm"
> plot(fit)
```

OOP

```
> setClass("simple",
     representation(x="numeric",y="matrix"),
     prototype = list(x=numeric(),y=matrix(0)))
> z <- new("simple", x=1:10,
                  y=matrix(rnorm(50),10,5))
> z@x
 [1]  1  2  3  4  5  6  7  8  9 10
> setMethod("plot",
    signature(x="simple", y="missing"),
    function(x, y,...)
     plot(slot(x,"x"),slot(x,"y")[,1]))
> plot(z)
```