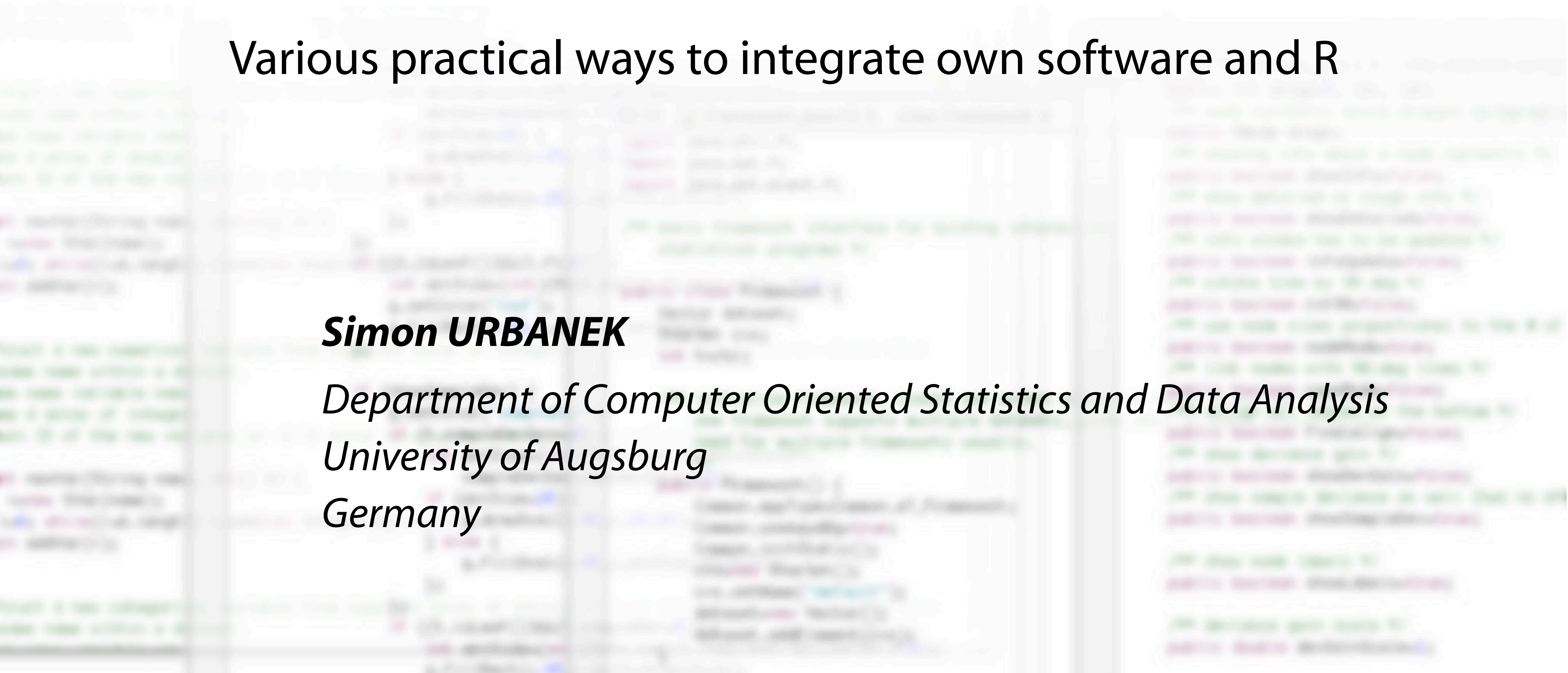
Using R in Other Applications

Various practical ways to integrate own software and R

Simon URBANEK

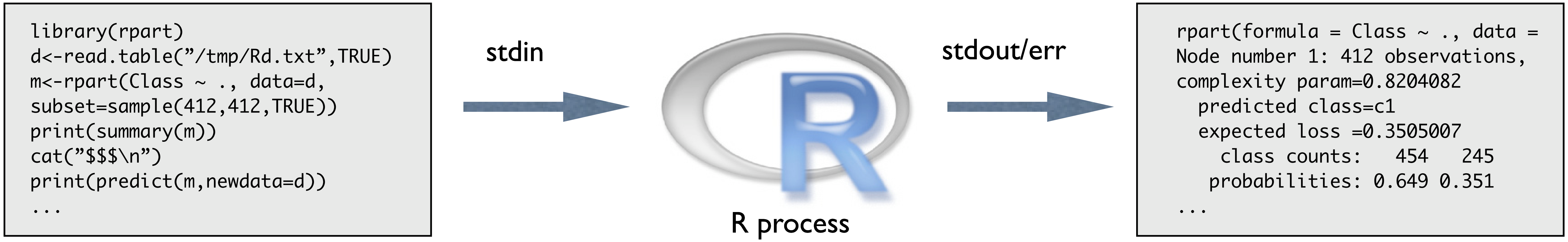
Department of Computer Oriented Statistics and Data Analysis
University of Augsburg
Germany



Communicating with R

- **R batch mode (stdin/out/err)**
- **connections and sockets**
- **C/Fortran interface**
  - linking external code into R (e.g. packages)
  - using R shared library in other programs
- **3rd party packages and projects**
  (use mainly C/Fortran interface)

R batch mode



```
library(rpart)
d<-read.table(”/tmp/Rd.txt”,TRUE)
m<-rpart(Class ~ ., data=d,
subset=sample(412,412,TRUE))
print(summary(m))
cat(”$$$\n”)
print(predict(m,newdata=d))
...
```

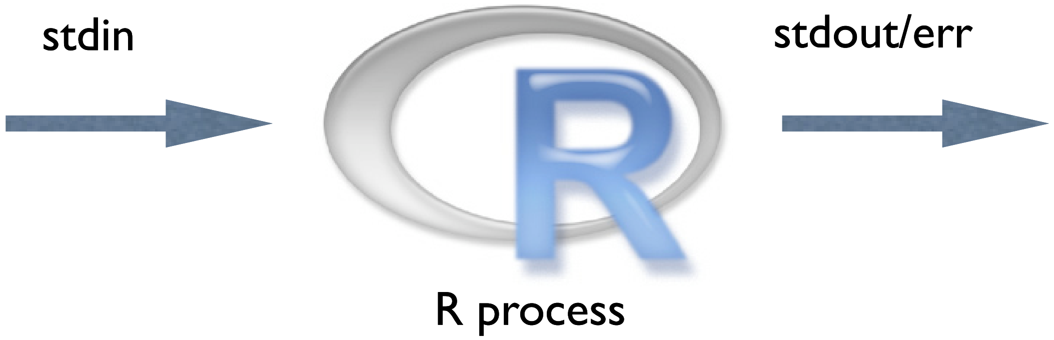
```
rpart(formula = Class ~ ., data =
Node number 1: 412 observations,
complexity param=0.8204082
  predicted class=c1
  expected loss =0.3505007
    class counts:    454    245
   probabilities: 0.649 0.351
...
```

Example: tiny CGI-script

```
#!/usr/bin/perl
use Cgi;
$cmd=$Cgi::command;
$cmd=~s/\\/\\\\/g; $cmd=~s/\”/\\\”/g;
$tfn=”/tmp/demo”.int(rand(10000)).”.R”;
open OUT,”>$tfn”;
print OUT “library(mylib)\nprocessCmd(\”$cmd\”)\n”;
close OUT;
$res=`R --no-save --slave < $tfn 2>&1`;
unlink $tfn;
print “Content-type: text/html\r\n\r\n”;
print $res;
```

R batch mode

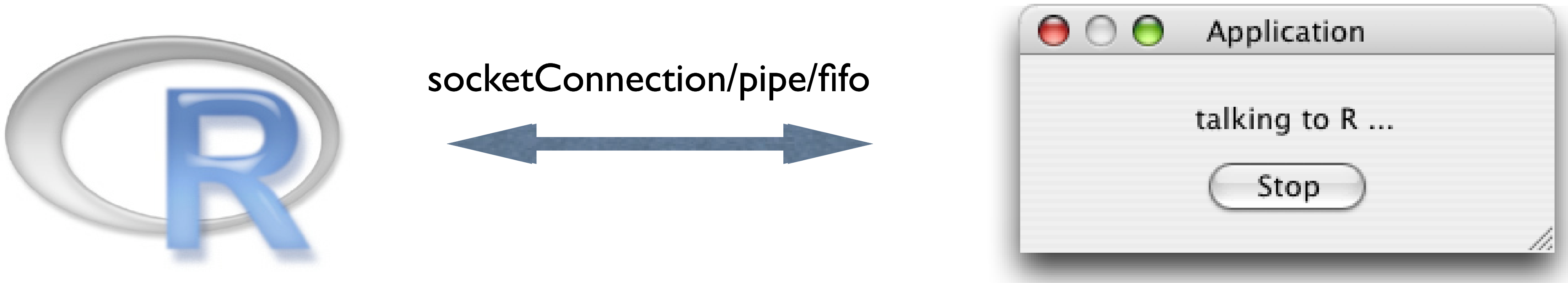
```
library(rpart)
d<-read.table(”/tmp/Rd.txt”,TRUE)
m<-rpart(Class ~ ., data=d,
subset=sample(412,412,TRUE))
print(summary(m))
cat(”$$$\n”)
print(predict(m,newdata=d))
...
```



```
rpart(formula = Class ~ ., data =
Node number 1: 412 observations,
complexity param=0.8204082
  predicted class=c1
  expected loss =0.3505007
    class counts:   454   245
   probabilities: 0.649 0.351
...
```

- **advantage**
  - this “interface” is easy to use
- **potential drawbacks**
  - slow response: full initialization of a new R instance is necessary
  - data and code must be stored (mostly as text) prior to processing
  - results must be parsed if further processing is desired

Connections and sockets



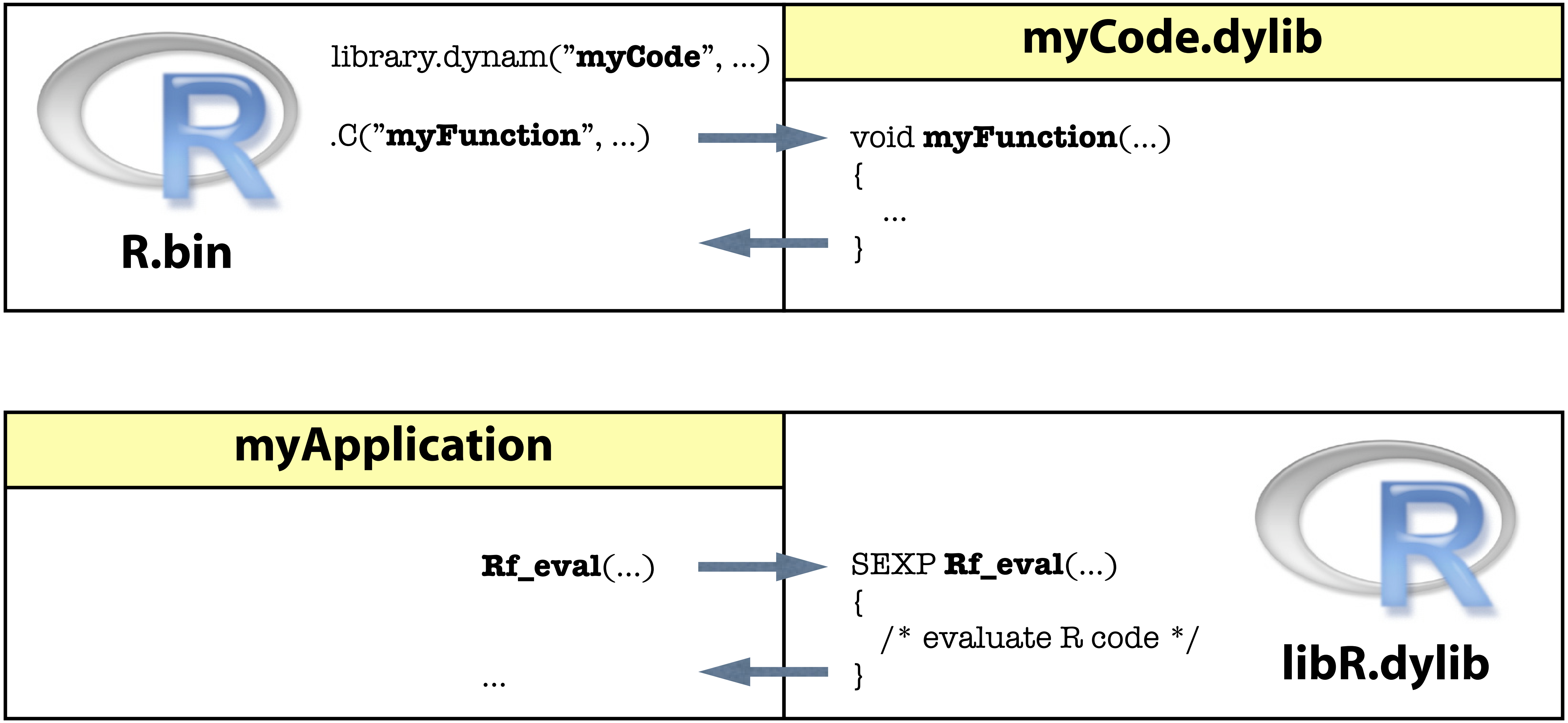
Example: tiny R-web-server

```
co <- socketConnection(port=8080, server=TRUE, blocking=TRUE)
s <- req <- readLines(co,1)
cl <-0
while (nchar(s) > 0) {
  s <- readLines(co,1)
  if (length(grep("Content-length:", s, ignore.case=TRUE)) > 0)
    cl <- as.integer(sub("Content-length:[ \t]*([0-9]+)","\\1",s))
}
ct <- if (cl>0) readChar(co, cl) else NA
rfn <- sub("^[A-Z]+ ([^ ]+) .*","\\1",req)

# request for the file “rfn” to be handled here ...

close(co)
```

Connections and sockets

- **advantages**
  - code written entirely in R
  - R has nice functions for transporting entire R objects (*readBin/writeBin, save/load, serialize*-package)
    this is especially useful when talking to another R instance
  - no initialization delay per request
- **possible drawbacks**
  - R is not really powerful tool for string-parsing tasks
  - parallel processing of requests is very hard
  - slow communication (depends on connection type and task)

C/Fortran interface



R.bin

library.dynam("**myCode**", ...)

.C("**myFunction**", ...)

myCode.dylib

```
void myFunction(...)
{
   ...
}
```

myApplication

Rf_eval(...)

...

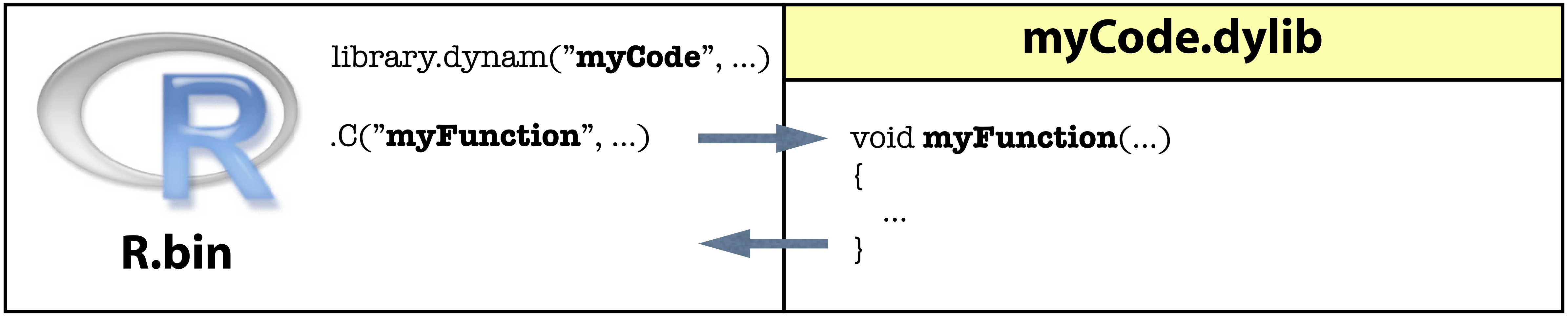
```
SEXP Rf_eval(...)
{
   /* evaluate R code */
}
```

libR.dylib

C/Fortran interface

- **advantages**
  - very fast
  - shortcuts and optimizations possible
    (e.g. skip parsing step, keep intermediate objects)
  - direct data access
- **possible drawbacks**
  - dangerously low-level, good R knowledge as well as good programming practice necessary
  - R is not entirely re-entrant, parallelization must be well thought out
  - some aspects (e.g. initialization of the R dylib) are platform-dependent

Integrating C code into R

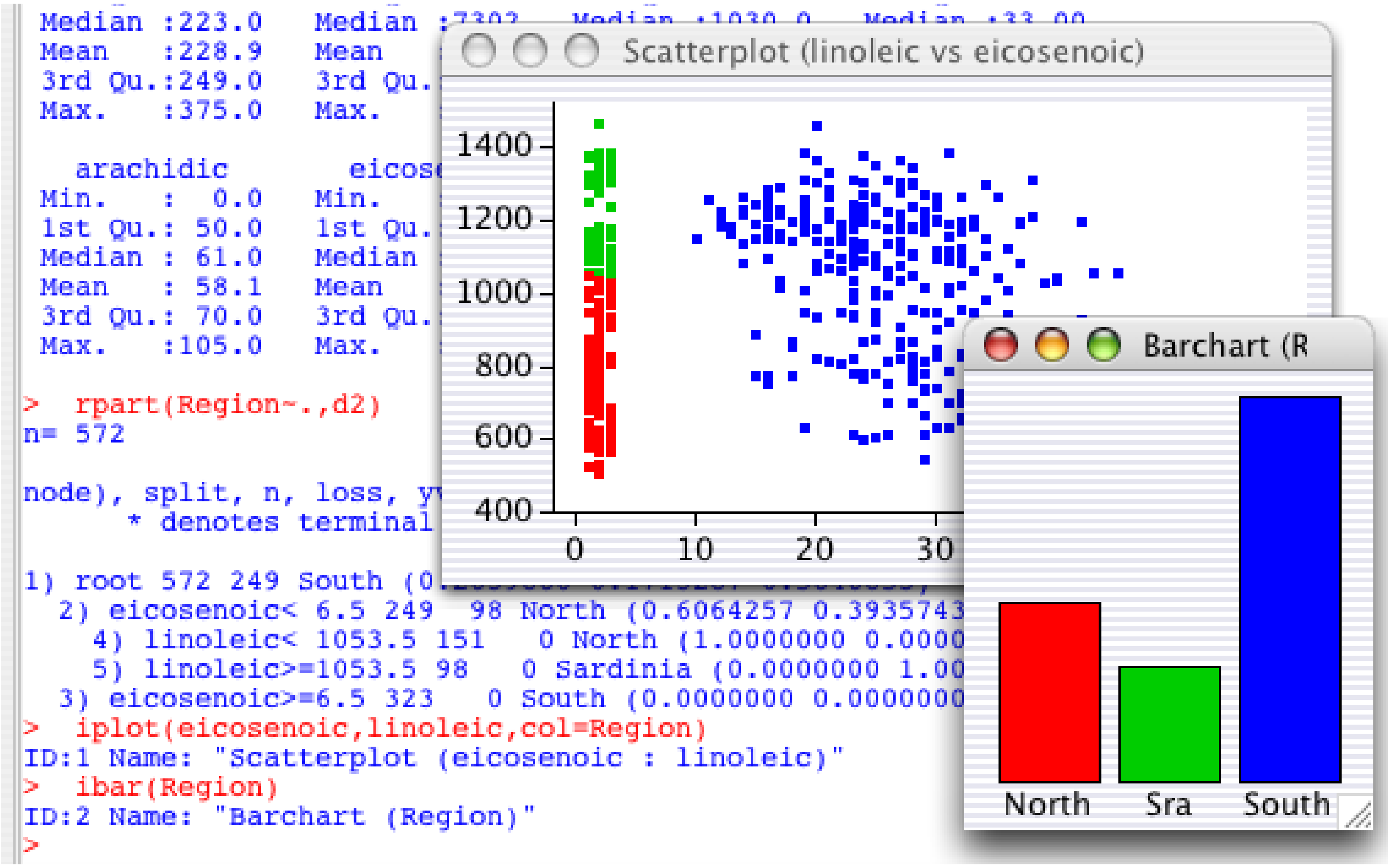


```
library.dynam("myCode", ...)

.C("myFunction", ...)
```

```
void myFunction(...)
{
    ...
}
```

Examples of integrated C code

- **most R packages use C/Fortran code for computation**
- **Rggobi integrates ggobi.dylib into R**
- **iPlots integrate interactive graphics into R**



```
 Median :223.0    Median :7302    Median :1030.0    Median :33.00
 Mean   :228.9    Mean
 3rd Qu.:249.0    3rd Qu.
 Max.   :375.0    Max.

   arachidic         eicos
 Min.   :  0.0    Min.
 1st Qu.: 50.0    1st Qu.
 Median : 61.0    Median
 Mean   : 58.1    Mean
 3rd Qu.: 70.0    3rd Qu.
 Max.   :105.0    Max.

>  rpart(Region~.,d2)
n= 572

node), split, n, loss, y
      * denotes terminal

1) root 572 249 South (0.
  2) eicosenoic< 6.5 249  98 North (0.6064257 0.3935743
    4) linoleic< 1053.5 151   0 North (1.0000000 0.0000
    5) linoleic>=1053.5 98   0 Sardinia (0.0000000 1.00
  3) eicosenoic>=6.5 323   0 South (0.0000000 0.0000000
>  iplot(eicosenoic,linoleic,col=Region)
ID:1 Name: "Scatterplot (eicosenoic : linoleic)"
>  ibar(Region)
ID:2 Name: "Barchart (Region)"
>
```

Parts of the interface

function calls:

```
.C(”myFunction”, 10.5, “hello”)      → void myFunction(double *a, char **b)
.Call(”myCall”, 10.5, “hello”)       → SEXP myCall(SEXP a, SEXP b)
.External(”myExt”, 10.5, “hello”)    → SEXP myExt(SEXP args)
```

data allocation and access:

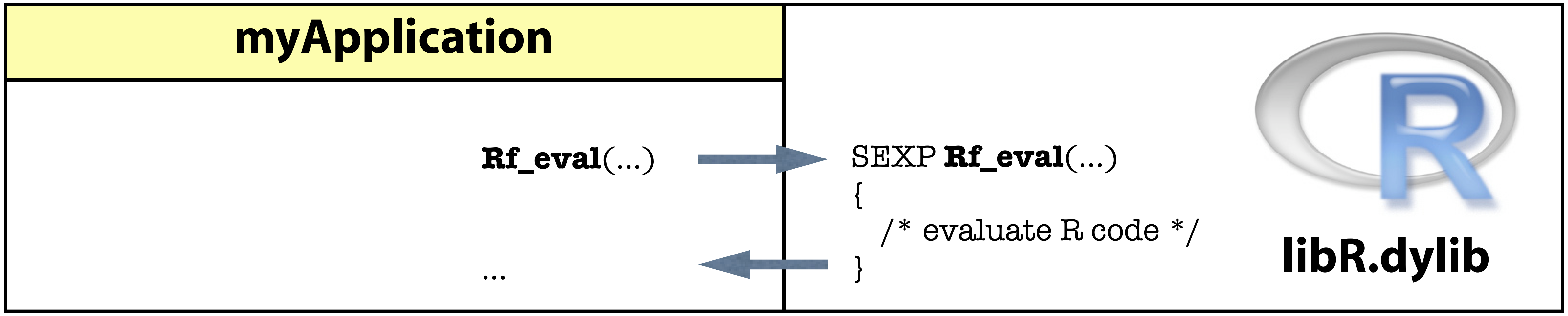
```
allocVector(VECSXP, 10);
SET_VECTOR_ELT(v, 0, install(”x”));
...
```

supporting internal R functions:

```
R_ParseVector(cv, maxParts, status);
eval(expr, rho);
...
```

... for details see *“Writing R Extensions”*

Integrating R into other software



myApplication

Rf_eval(...)

...

```
SEXP Rf_eval(...)
{
   /* evaluate R code */
}
```

libR.dylib

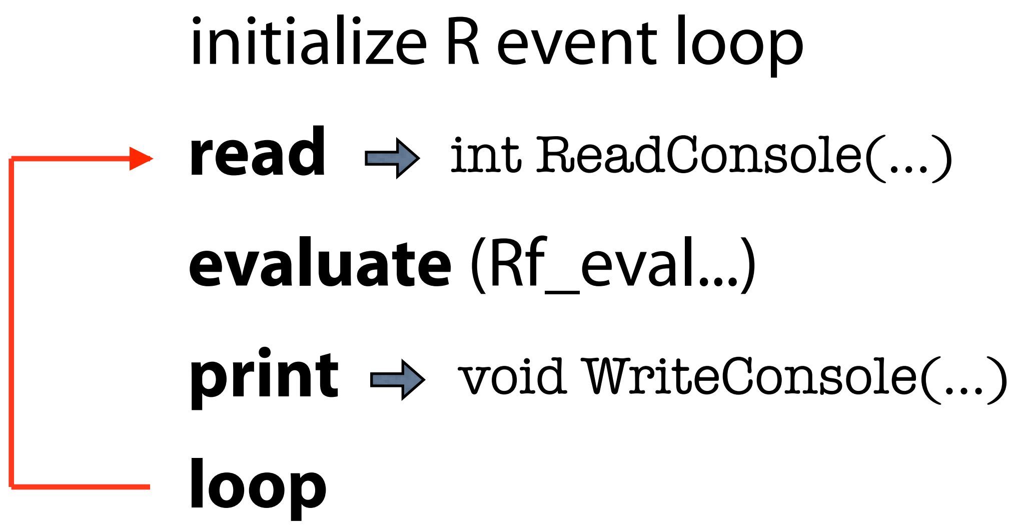
Integrating R into other software

- **load or link to R dynamic library** (libR.so / R.framework / R.dll)
- **initialize R engine**

Call individual R functions

```
char *s="rnorm(100)";
cv=allocVector(STRSXP, 1);
SET_VECTOR_ELT(cv, 0, mkChar(s));
pr=R_ParseVector(cv, 1, status);
exp=eval(VECTOR_ELT(pr, 1),
          R_GlobalEnv);
double *d=REAL(exp);
```

Run R event loop



initialize R event loop

read ⇨ int ReadConsole(...)

evaluate (Rf_eval...)

print ⇨ void WriteConsole(...)

loop

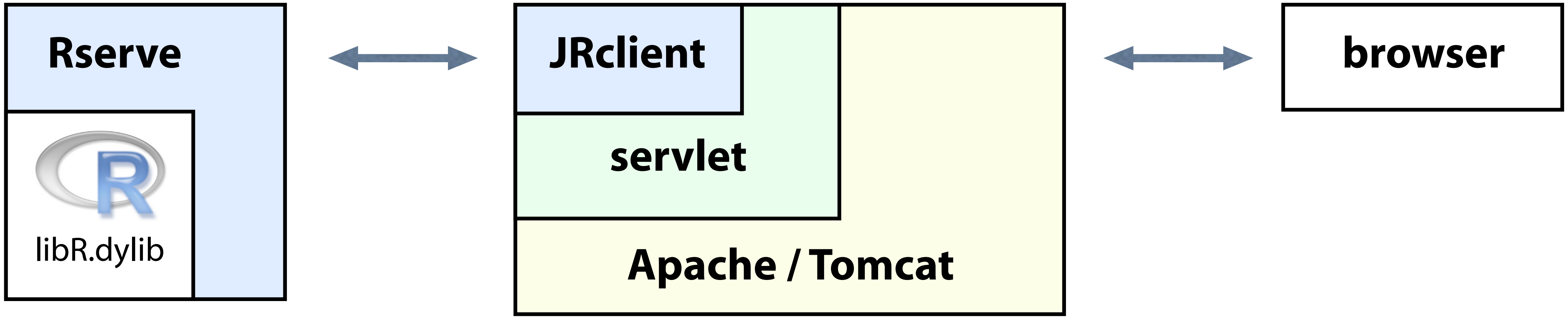
Integrative issues to consider

- **R initialization**
  - is system dependent (see sources of R and other interface projects)
- **R is single-threaded, mostly non-reentrant**
  - R functions should be called only by the R-initializing thread
  - packages usually cannot use threads (platform-dependent)
- **R and its event loop**
  - R handles its own event loop (if run normally) - this involves potential calls of system functions that may interfere with the program
- **R graphics devices**
  - "windowed" devices (X11, Quartz, Windows) need an event loop

some 3rd party R interfaces using R dylib

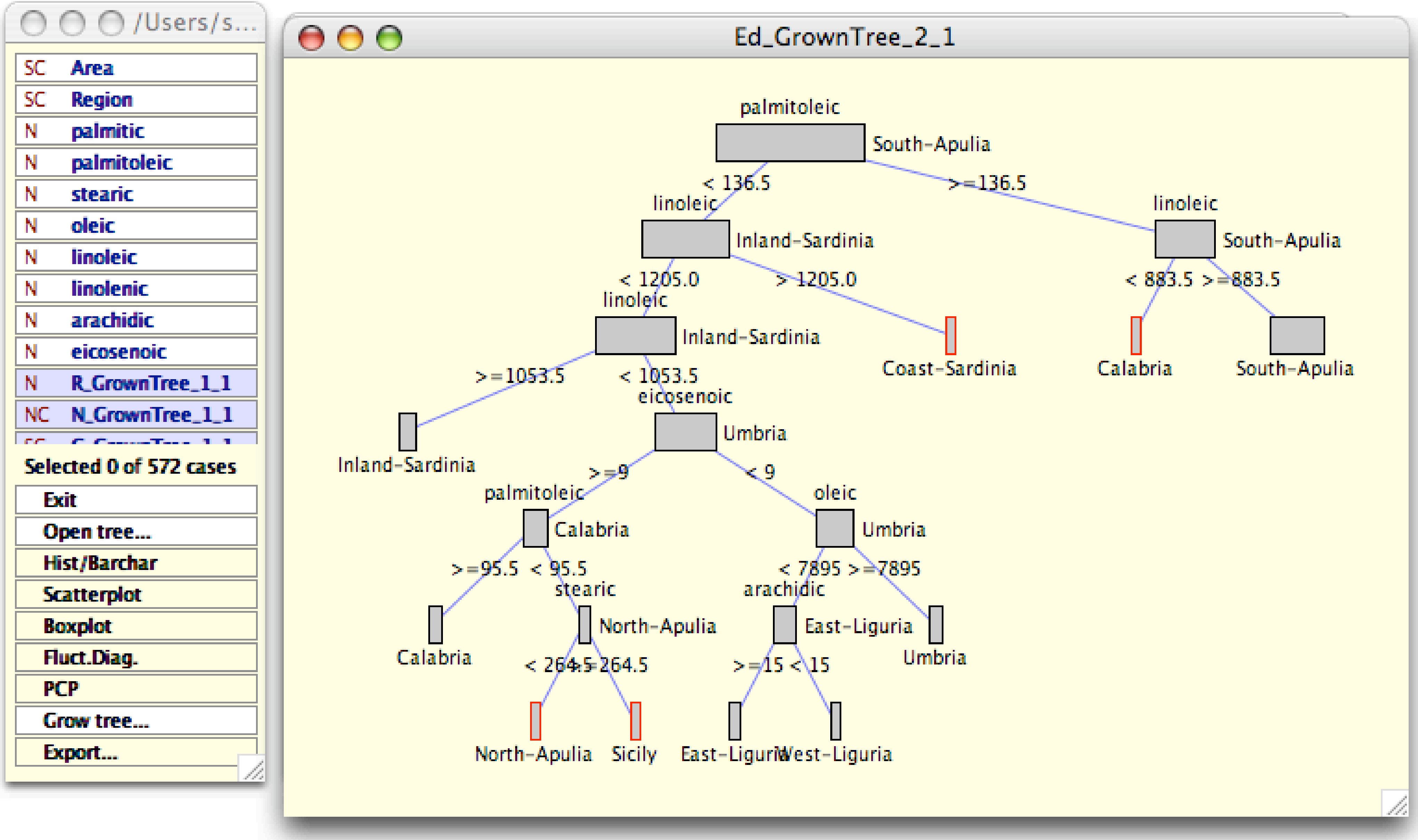
- **(D)COM server**
  - allows Windows programs to access R (ActiveX, Excel plug-in)
- **Omegahat project**
  - general approach to connectivity (R, S, CORBA, Java, perl, Python, ...) [some implementations work well, others are incomplete]
- **Rserve**
  - socket-based server (Java and C clients)
- **JRI**
  - bi-directional Java/R interface (both eval and REPL)
- **Obj-C R framework**
  - Obj-C interface to R (used by Cocoa GUI on Mac OS X)

Example: using R in a Web-application



- **Rserve**
  - offers multiple R instances without initialization delays
- **servlet**
  - prepares necessary data (user input, files, databases...)
  - delegates calculations to R via Rserve
  - builds proper html code as response (incl. image links if necessary)

Example: using R for computations



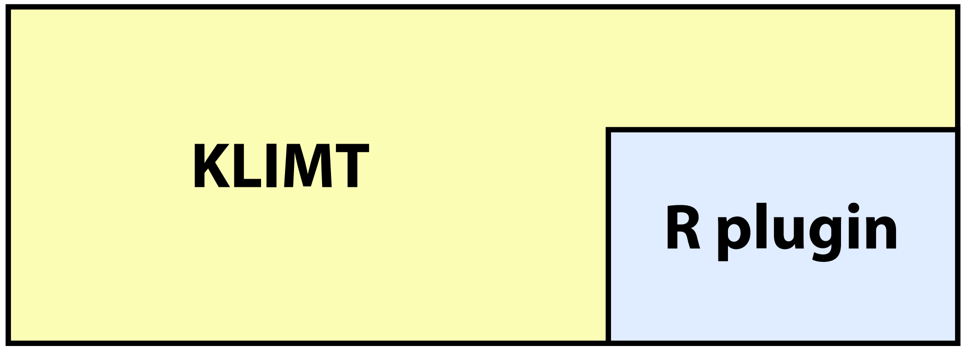
/Users/s...

SC Area
SC Region
N palmitic
N palmitoleic
N stearic
N oleic
N linoleic
N linolenic
N arachidic
N eicosenoic
N R_GrownTree_1_1
NC N_GrownTree_1_1

Selected 0 of 572 cases

Ed_GrownTree_2_1

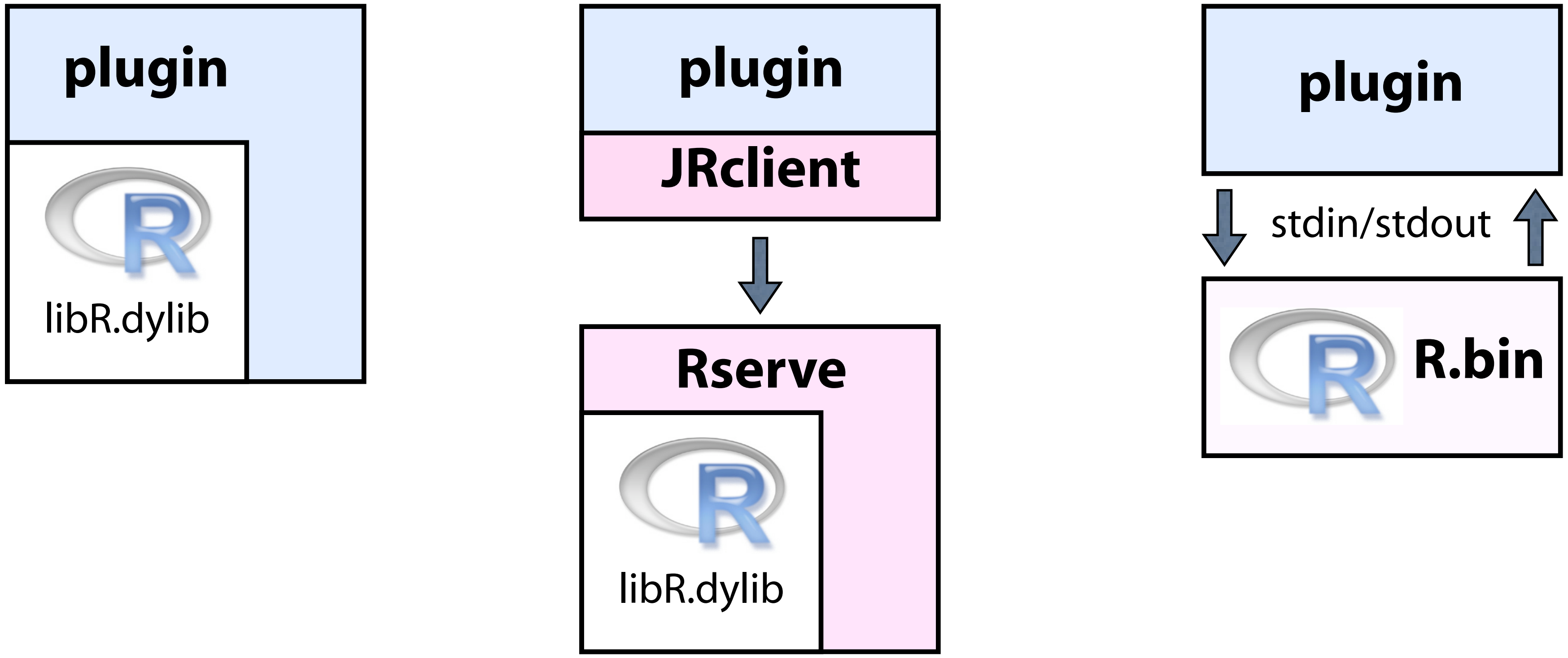
Example: using R for computations



KLIMT

R plugin

various plugins:



plugin

libR.dylib

plugin

JRclient

Rserve

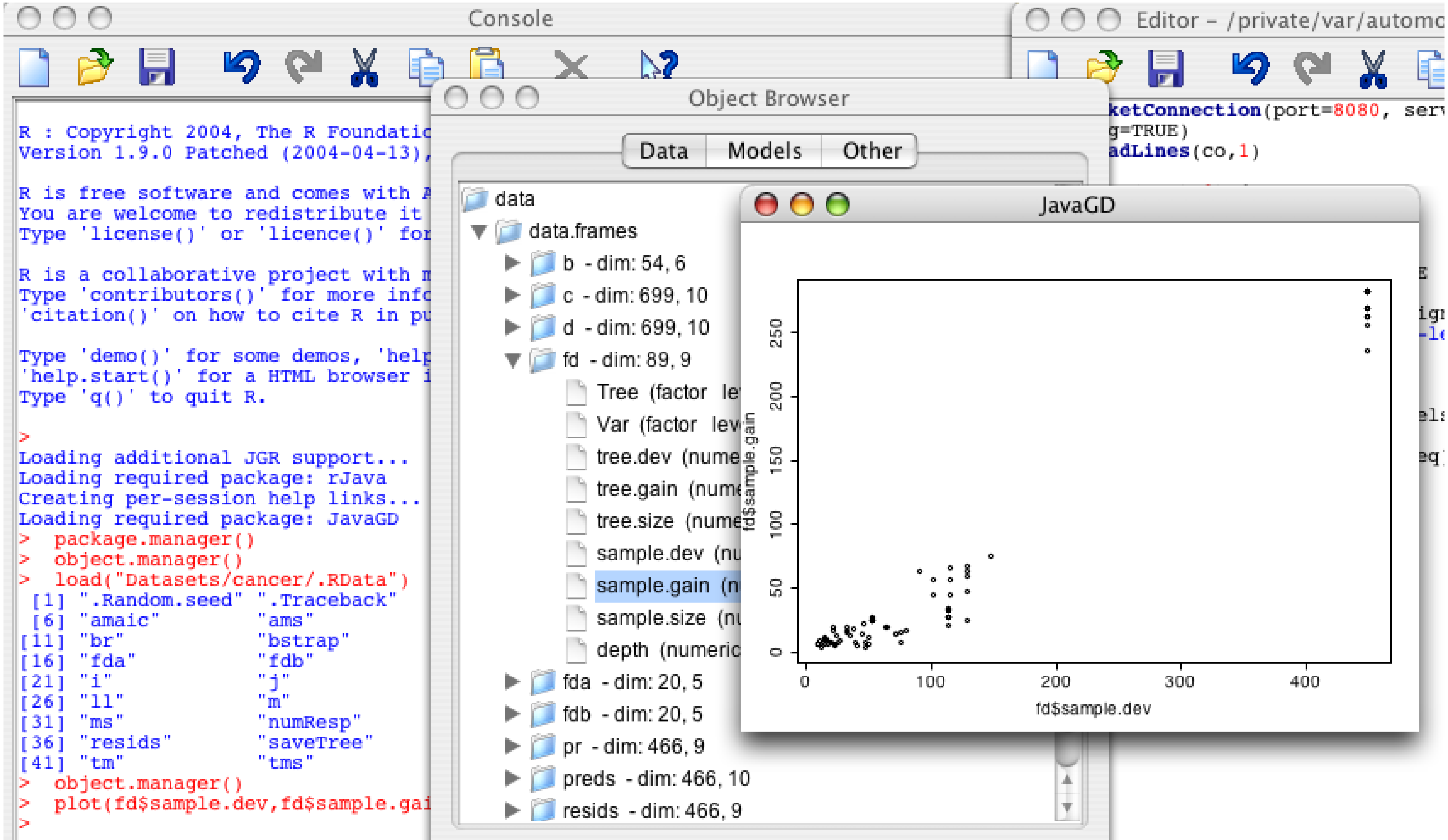
libR.dylib

plugin

stdin/stdout

R.bin

Example: full control of R

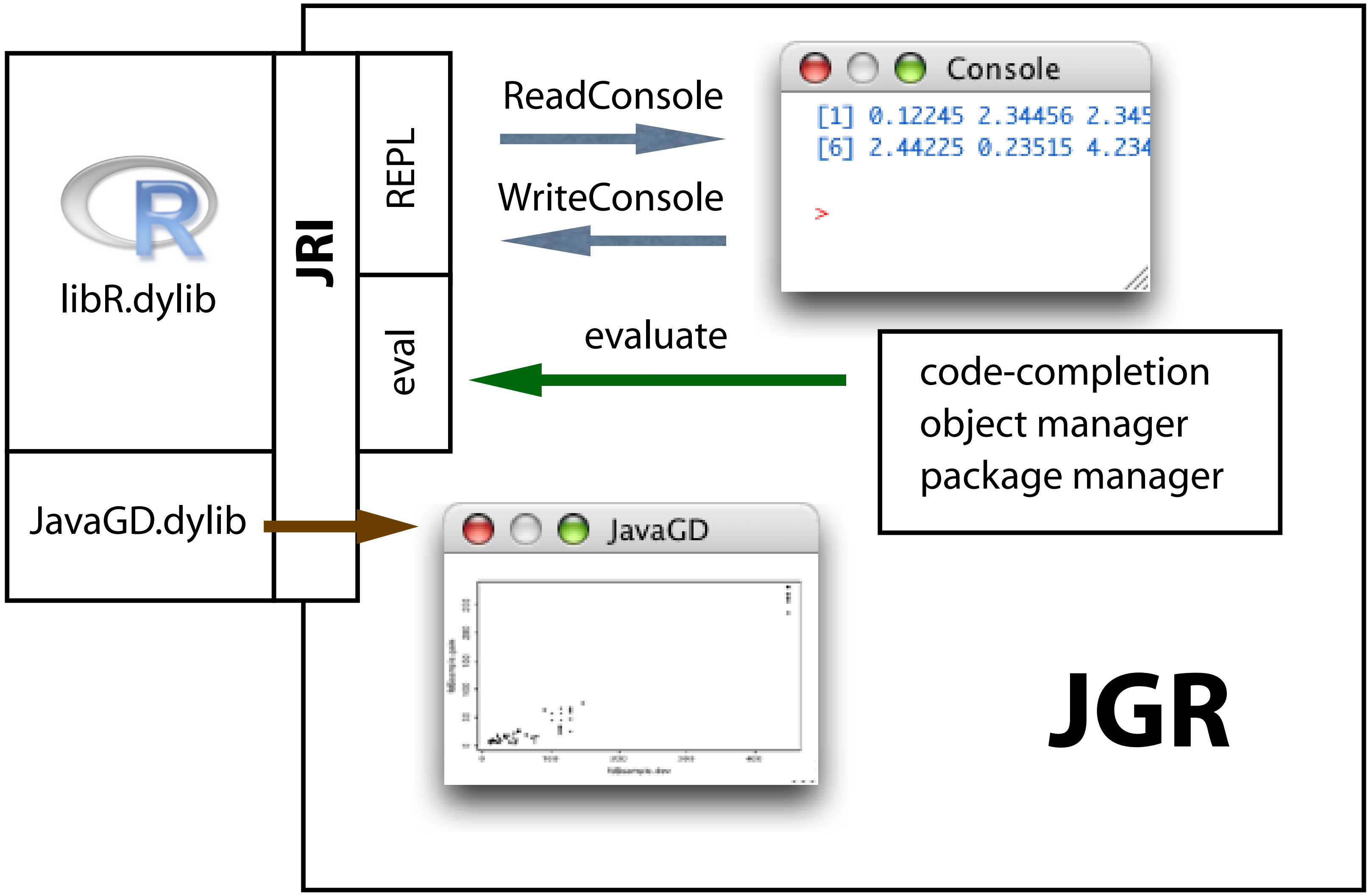


```
R : Copyright 2004, The R Foundatio
Version 1.9.0 Patched (2004-04-13),

R is free software and comes with A
You are welcome to redistribute it
Type 'license()' or 'licence()' for

R is a collaborative project with m
Type 'contributors()' for more info
'citation()' on how to cite R in pu

Type 'demo()' for some demos, 'help
'help.start()' for a HTML browser i
Type 'q()' to quit R.

>
Loading additional JGR support...
Loading required package: rJava
Creating per-session help links...
Loading required package: JavaGD
>  package.manager()
>  object.manager()
>  load("Datasets/cancer/.RData")
 [1] ".Random.seed" ".Traceback"
 [6] "amaic"        "ams"
[11] "br"           "bstrap"
[16] "fda"          "fdb"
[21] "i"            "j"
[26] "ll"           "m"
[31] "ms"           "numResp"
[36] "resids"       "saveTree"
[41] "tm"           "tms"
>  object.manager()
>  plot(fd$sample.dev,fd$sample.gai
>
```

```
data
▼ data.frames
  ▶ b - dim: 54, 6
  ▶ c - dim: 699, 10
  ▶ d - dim: 699, 10
  ▼ fd - dim: 89, 9
      Tree (factor le
      Var (factor lev
      tree.dev (nume
      tree.gain (nume
      tree.size (nume
      sample.dev (nu
      sample.gain (n
      sample.size (nu
      depth (numeric
  ▶ fda - dim: 20, 5
  ▶ fdb - dim: 20, 5
  ▶ pr - dim: 466, 9
  ▶ preds - dim: 466, 10
  ▶ resids - dim: 466, 9
```

```
ketConnection(port=8080, serv
g=TRUE)
adLines(co,1)
```

Example: full control of R



```
[1] 0.12245 2.34456 2.345
[6] 2.44225 0.23515 4.234

>
```

Conclusion

- **R provides three native interfaces to the outer world**
  - stdin/out/err (batch processing) - slow, text-oriented but simple
  - connections/sockets - pure R code, good for specialized tasks
  - C/Fortran interface - fast, but good knowledge of R is needed
- **R supports various ways of integration**
  - embed own code into R (packages, library.dynam)
  - use individual R functions in an own code (libR.dylib)
  - run the R event loop (REPL) - similar to the stdin/out approach
- **The only limiting aspects are initialization and re-entrance**
- **3rd party packages offer additional interfaces for specific tasks**

Contact

Simon URBANEK

simon.urbanek@math.uni-augsburg.de

Department of Computer Oriented Statistics and Data Analysis
University of Augsburg
Germany

http://www.rosuda.org/~urbanek