R in Java

FastR: an implementation of the R language


Petr
Maj


Tomas
Kalibera


Jan
Vitek


Floréal
Morandat

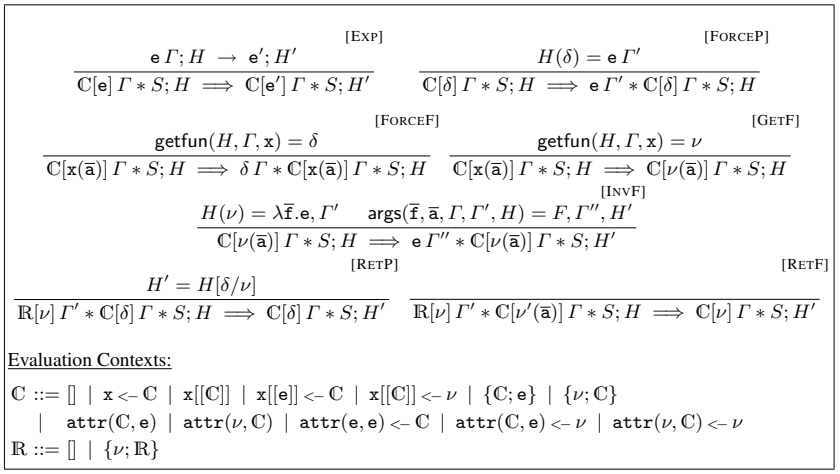

Helena
Kotthaus

Purdue University & Oracle Labs

https://github.com/allr

What we do…

- **`TimeR`** — an instrumentation-based profiler for GNU-R
- **`TracR`** — a trace analysis framework for GNU-R
- **`CoreR`** — a formal semantics for a fragment of R
- **`TestR`** — a testing framework for the R language
- **`FastR`** — **a new R virtual machine written in Java**



$$\frac{\mathsf{e}\,\Gamma; H \;\rightarrow\; \mathsf{e}'; H'}{\mathbb{C}[\mathsf{e}]\,\Gamma * S; H \;\Longrightarrow\; \mathbb{C}[\mathsf{e}']\,\Gamma * S; H'}\;[\textsc{Exp}] \qquad \frac{H(\delta) = \mathsf{e}\,\Gamma'}{\mathbb{C}[\delta]\,\Gamma * S; H \;\Longrightarrow\; \mathsf{e}\,\Gamma' * \mathbb{C}[\delta]\,\Gamma * S; H}\;[\textsc{ForceP}]$$

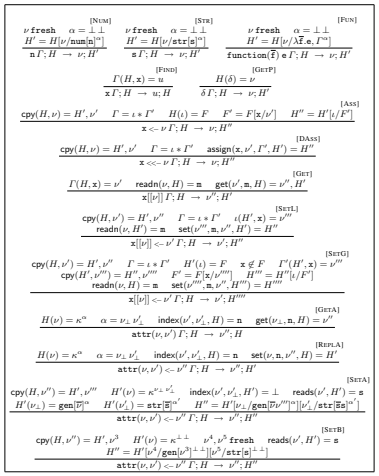
$$\frac{\mathsf{getfun}(H, \Gamma, \mathtt{x}) = \delta}{\mathbb{C}[\mathtt{x}(\overline{\mathsf{a}})]\,\Gamma * S; H \;\Longrightarrow\; \delta\,\Gamma * \mathbb{C}[\mathtt{x}(\overline{\mathsf{a}})]\,\Gamma * S; H}\;[\textsc{ForceF}] \qquad \frac{\mathsf{getfun}(H, \Gamma, \mathtt{x}) = \nu}{\mathbb{C}[\mathtt{x}(\overline{\mathsf{a}})]\,\Gamma * S; H \;\Longrightarrow\; \mathbb{C}[\nu(\overline{\mathsf{a}})]\,\Gamma * S; H}\;[\textsc{GetF}]$$

$$\frac{H(\nu) = \lambda\overline{\mathtt{f}}.\mathsf{e}, \Gamma' \qquad \mathsf{args}(\overline{\mathtt{f}}, \overline{\mathsf{a}}, \Gamma, \Gamma', H) = F, \Gamma'', H'}{\mathbb{C}[\nu(\overline{\mathsf{a}})]\,\Gamma * S; H \;\Longrightarrow\; \mathsf{e}\,\Gamma'' * \mathbb{C}[\nu(\overline{\mathsf{a}})]\,\Gamma * S; H'}\;[\textsc{InvF}]$$

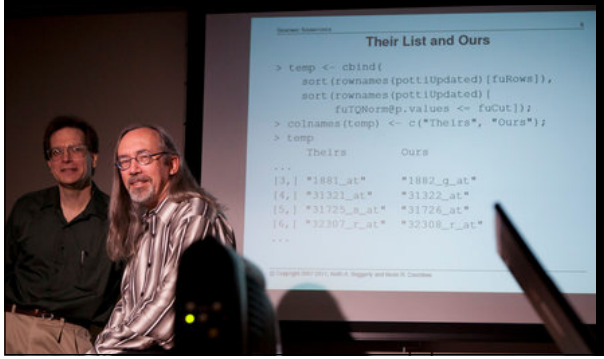
$$\frac{H' = H[\delta/\nu]}{\mathbb{R}[\nu]\,\Gamma' * \mathbb{C}[\delta]\,\Gamma * S; H \;\Longrightarrow\; \mathbb{C}[\delta]\,\Gamma * S; H'}\;[\textsc{RetP}] \qquad \frac{}{\mathbb{R}[\nu]\,\Gamma' * \mathbb{C}[\nu'(\overline{\mathsf{a}})]\,\Gamma * S; H \;\Longrightarrow\; \mathbb{C}[\nu]\,\Gamma * S; H'}\;[\textsc{RetF}]$$

Evaluation Contexts:

$$\mathbb{C} ::= [] \mid \mathtt{x} \mathtt{<-} \mathbb{C} \mid \mathtt{x}[[\mathbb{C}]] \mid \mathtt{x}[[\mathsf{e}]] \mathtt{<-} \mathbb{C} \mid \mathtt{x}[[\mathbb{C}]] \mathtt{<-} \nu \mid \{\mathbb{C}; \mathsf{e}\} \mid \{\nu; \mathbb{C}\}$$
$$\mid \mathtt{attr}(\mathbb{C}, \mathsf{e}) \mid \mathtt{attr}(\nu, \mathbb{C}) \mid \mathtt{attr}(\mathsf{e}, \mathsf{e}) \mathtt{<-} \mathbb{C} \mid \mathtt{attr}(\mathbb{C}, \mathsf{e}) \mathtt{<-} \nu \mid \mathtt{attr}(\nu, \mathbb{C}) \mathtt{<-} \nu$$
$$\mathbb{R} ::= [] \mid \{\nu; \mathbb{R}\}$$



Morandat, Hill, Osvald, Vitek. Evaluating the Design of the R Language. **ECOOP'12**



Why?

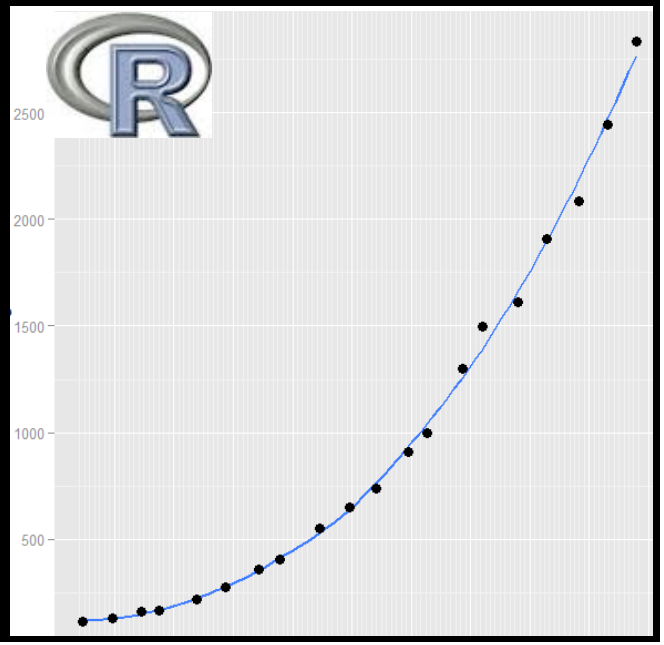
… language for data analysis and graphics

… used in statistics, biology, finance …

… books, conferences, user groups

… 4,338 packages

… 3 millions users



Scripting data

read data into variables

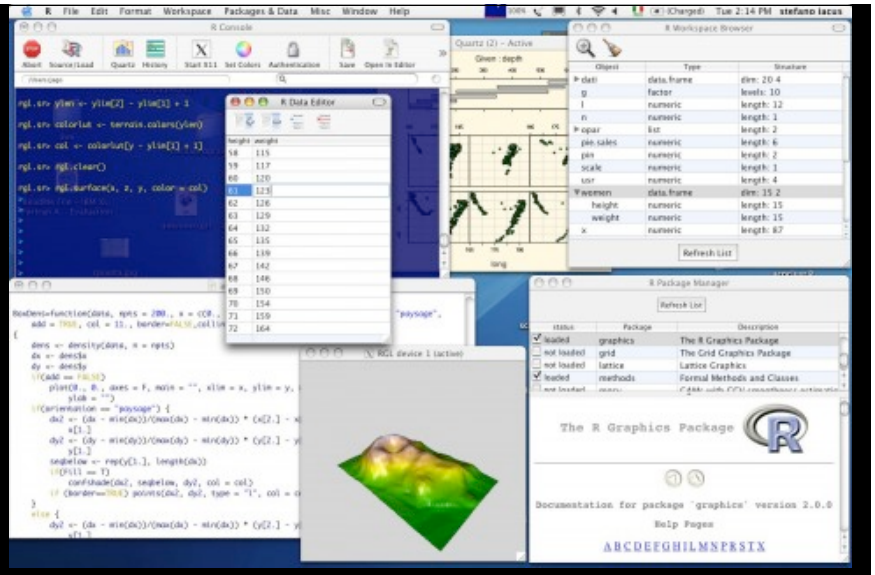
make plots

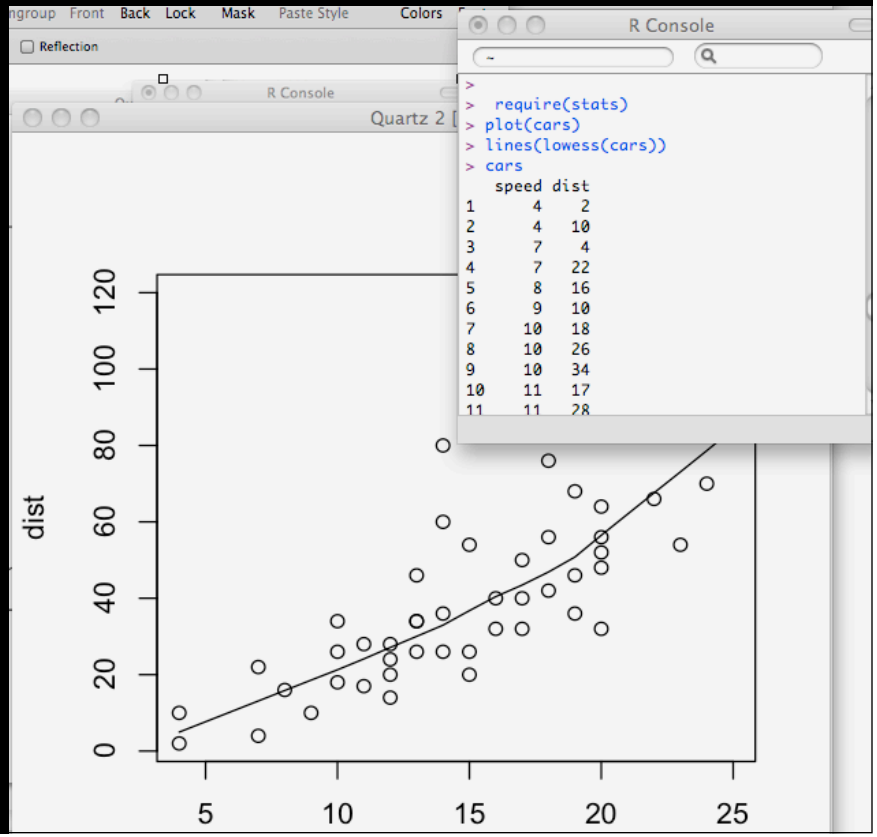
compute summaries

more intricate modeling

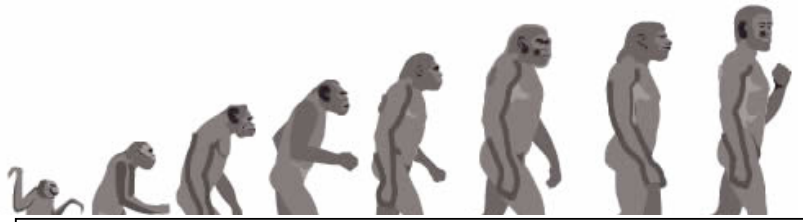
develop simple functions to automate analysis

…





R history



- 1976 **S**

John Chambers @ Bell Labs, then S-Plus (closed-source owned by Tibco)

- 1993 **R**

Ross Ihaka and Robert Gentleman, started R as new language at the University of Auckland, NZ

- Today, **The R project**

Core team ~ 20 people, released under GPL license. Continued development of language & libraries: namespaces ('11), bytecode ('11), indexing beyond 2GB ('13)

http://www.r-project.org
http://cran.r-project.org

What R is…

- vectorized
- functional
- object-oriented
- lazy
- portable
- interactive

What R isn't…

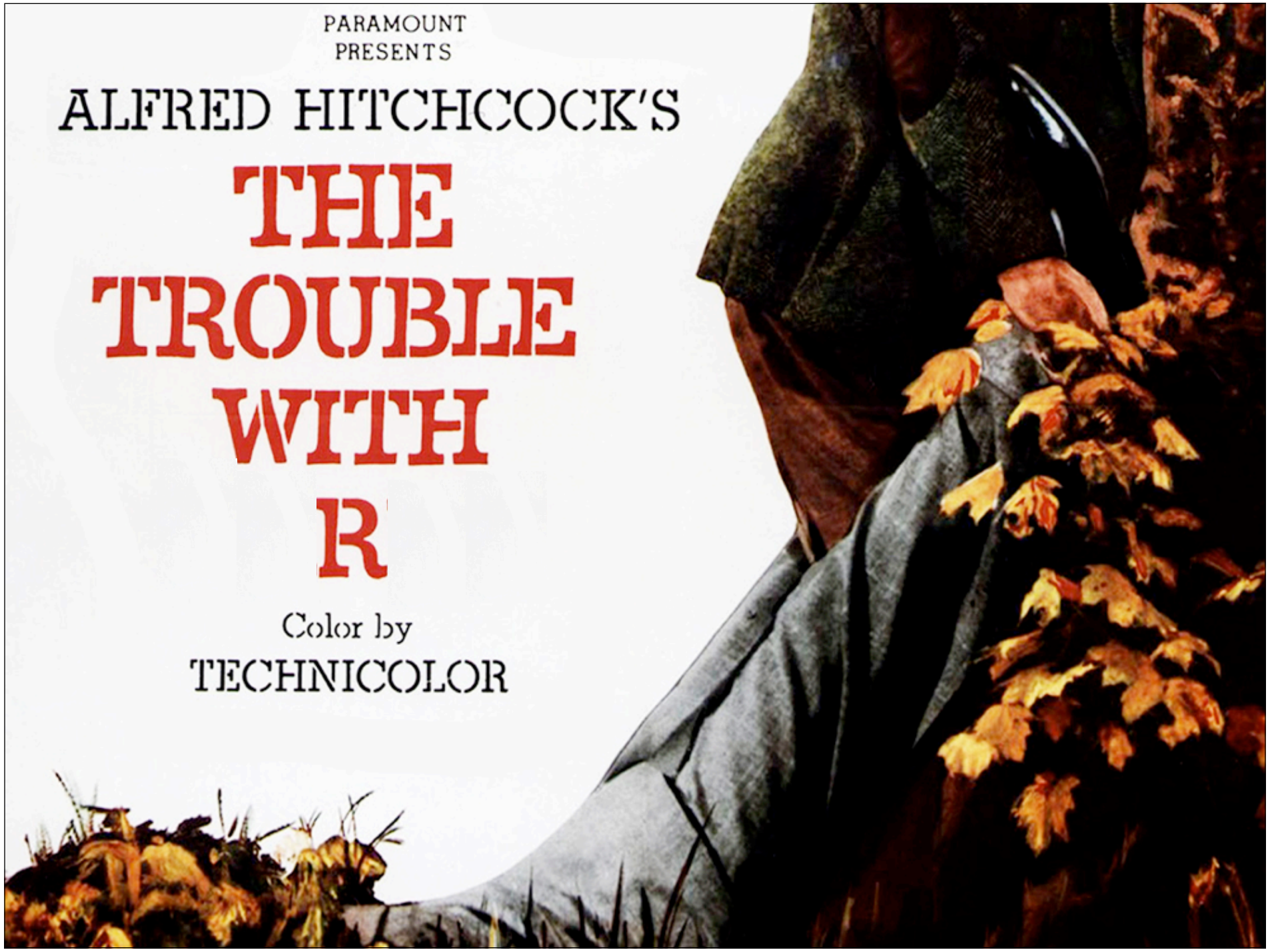
- fast
- low-footprint
- concurrent
- distributed
- formally specified
- standardized

PARAMOUNT
PRESENTS

ALFRED HITCHCOCK'S

THE TROUBLE WITH R

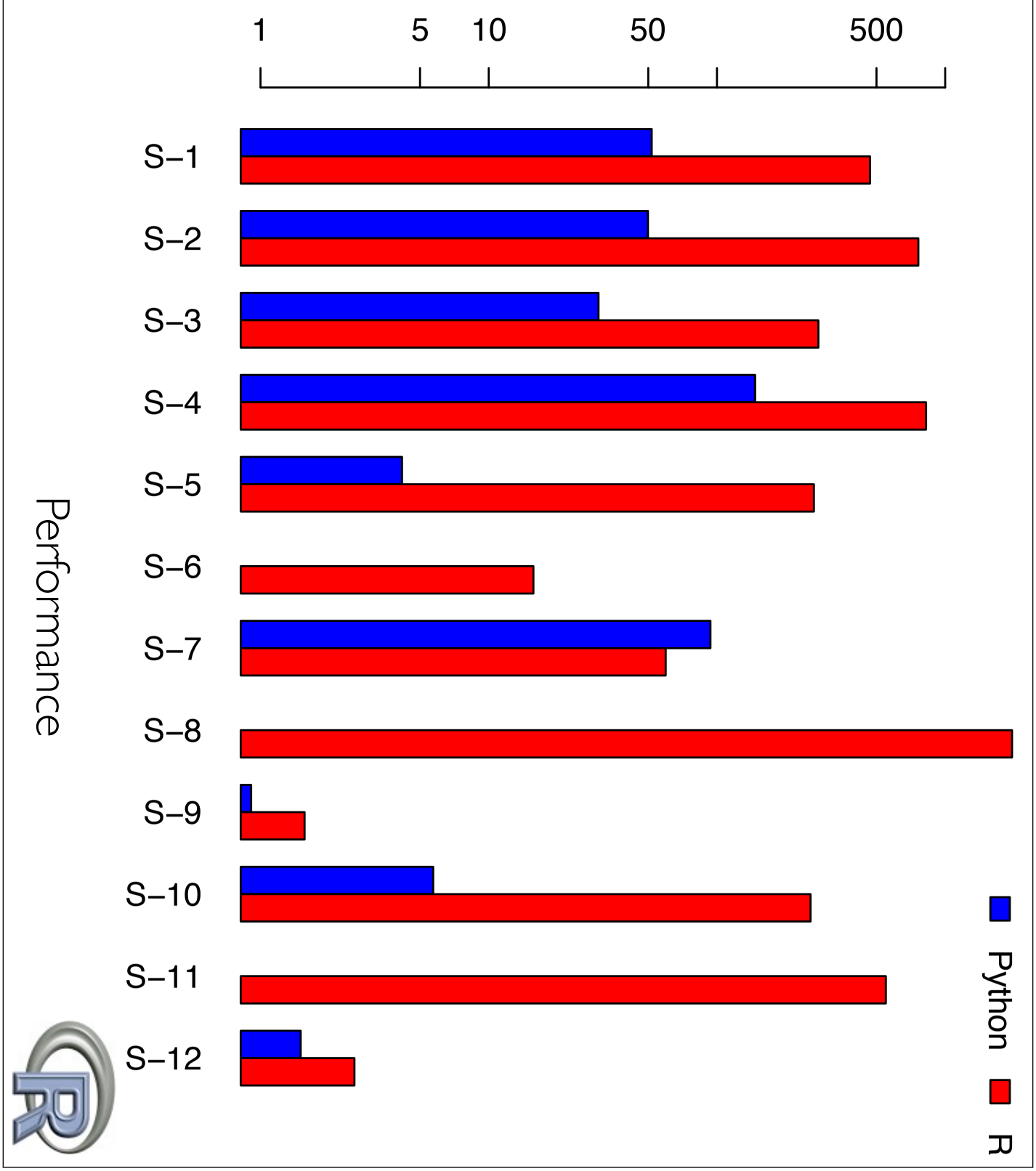
Color by
TECHNICOLOR



Intel X5460. 3.16GHz, Linux 2.6.34. R 2.12.1, GCC v4.4.5

The programming language shootout

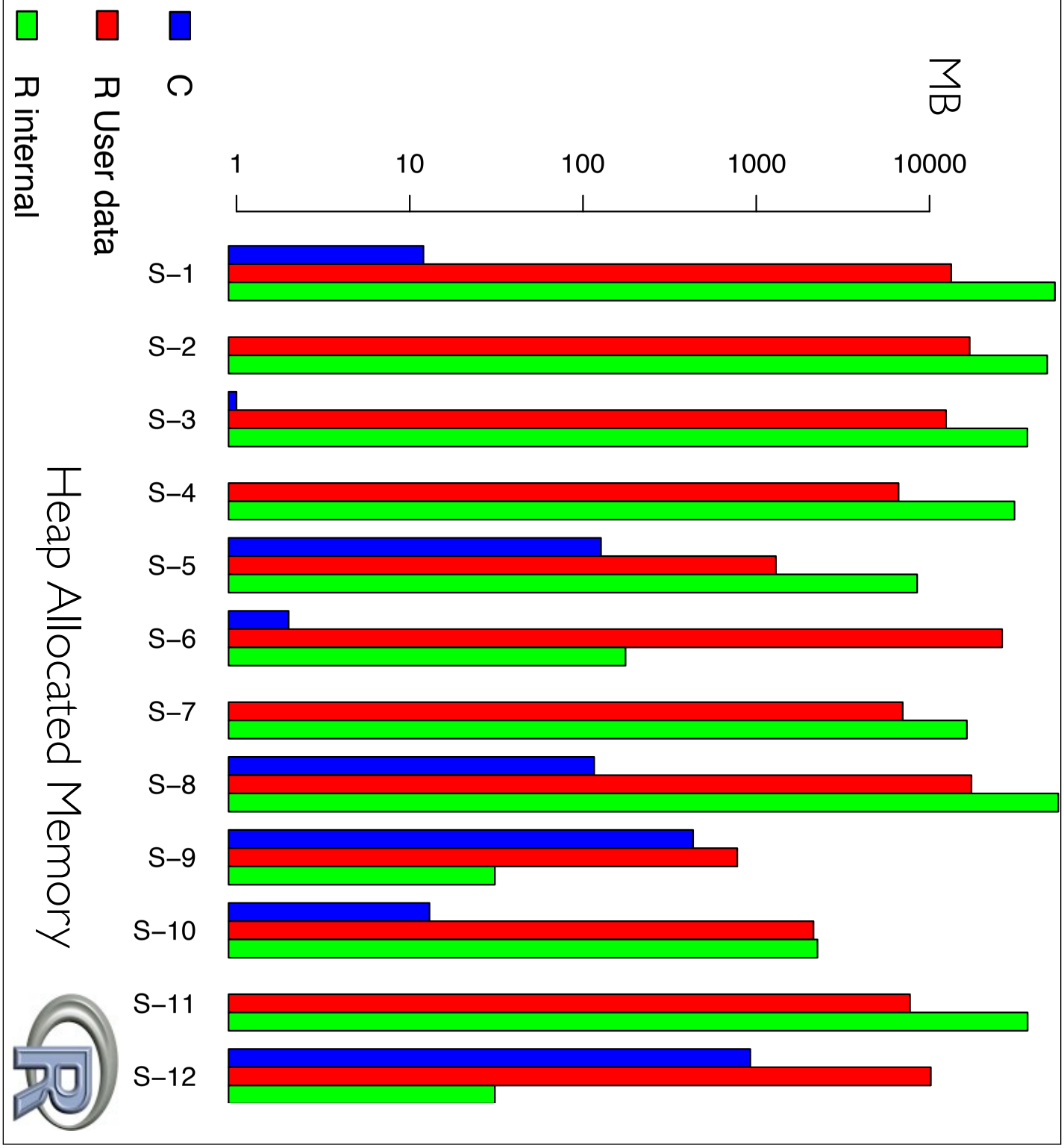
C / Python / R

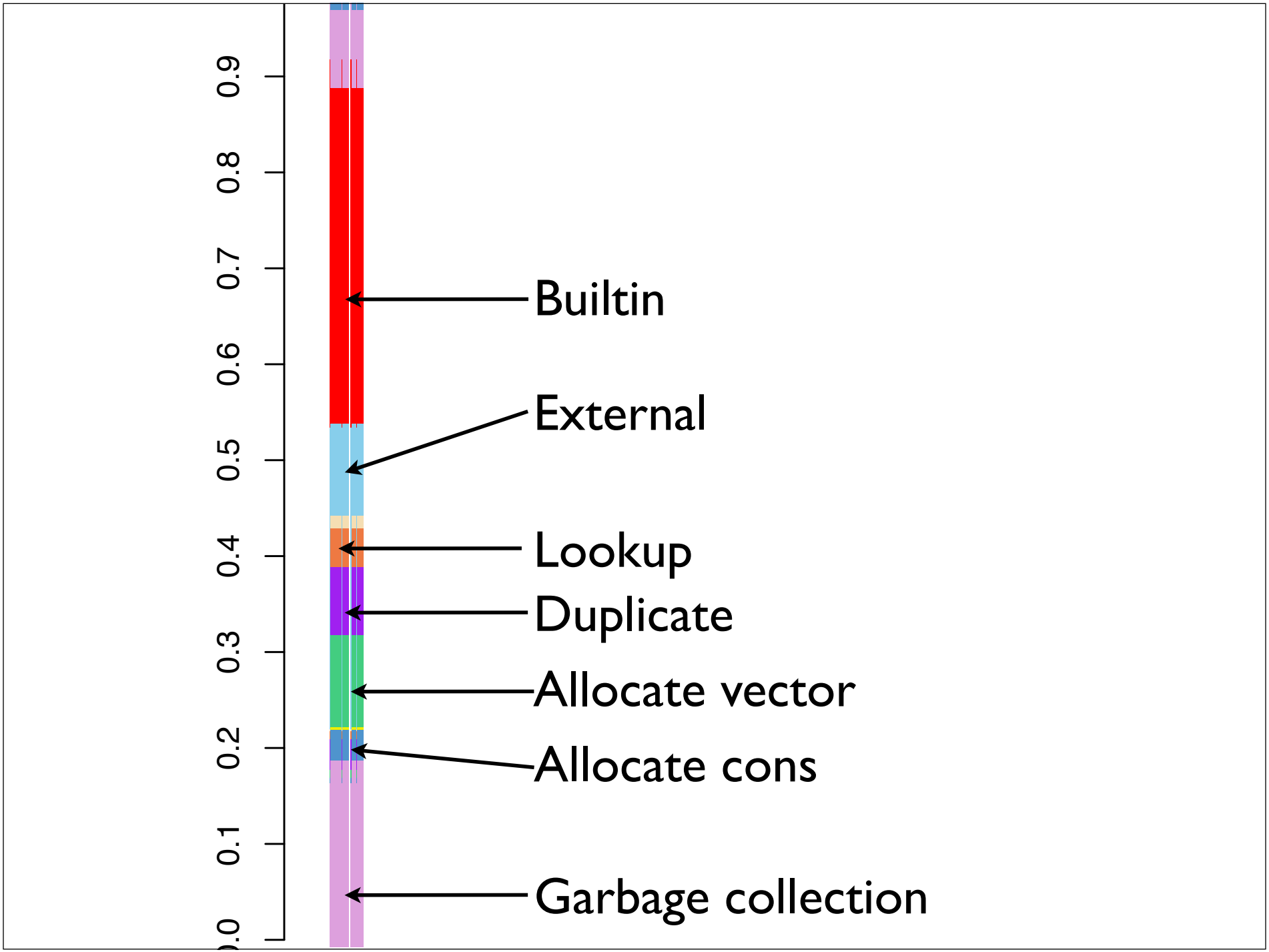


Intel X5460. 3.16GHz, Linux 2.6.34. R 2.12.1, GCC v4.4.5

The programming language shootout

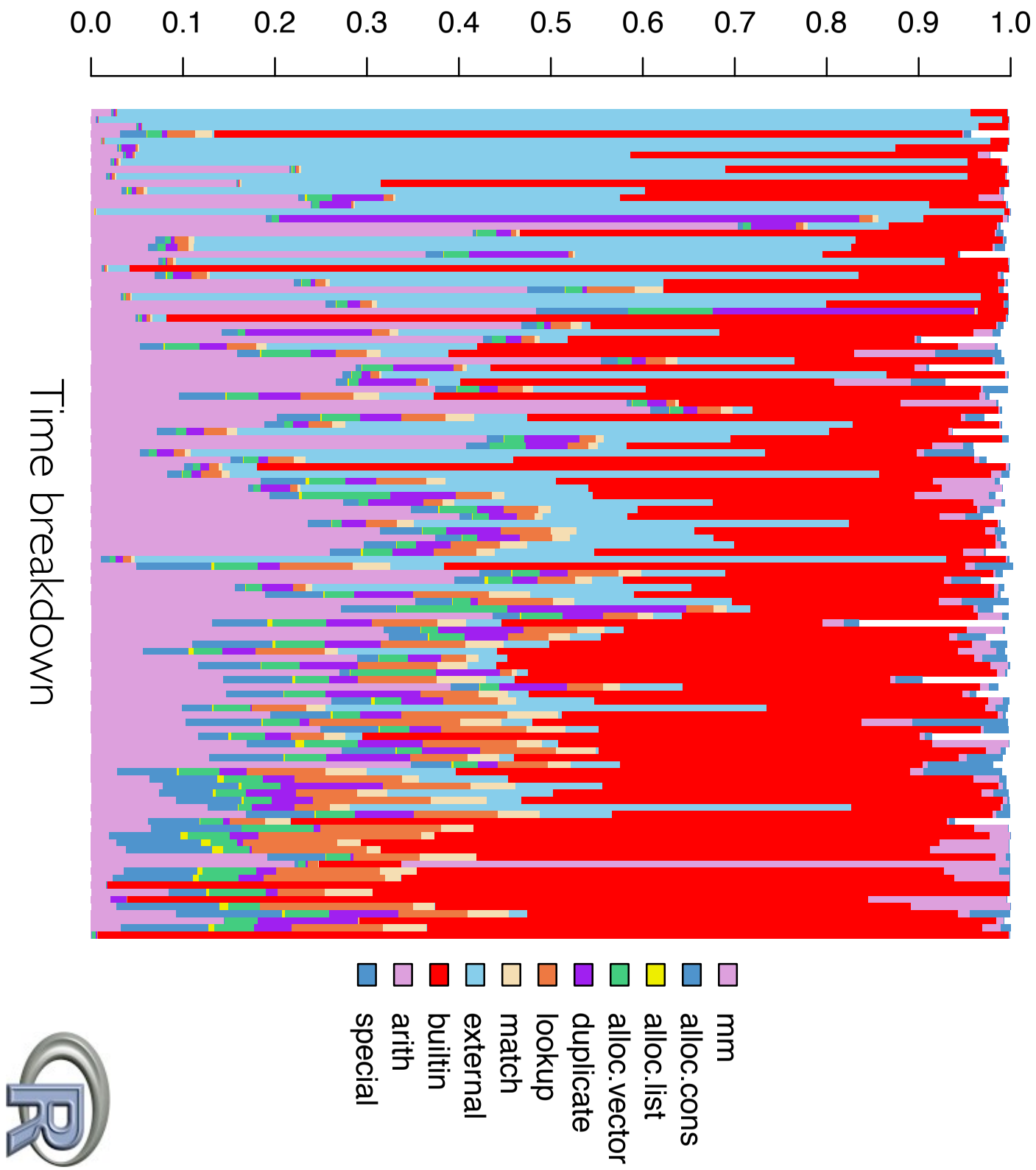
C / R




0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Builtin
External
Lookup
Duplicate
Allocate vector
Allocate cons
Garbage collection

Intel X5460. 3.16GHz, Linux 2.6.34. R 2.12.1, GCC v4.4.5

Bioconductor vignettes



How is R used?

- Extract core semantics by testing
  - *R has no official semantics*
  - *A single reference implementation*
- Observational study based on a large corpus
  - *Many open source programs come with "vignettes"*
  - *Dynamic analysis gives under-approximated behaviors*
  - *Static analysis gives over-approximation*

POPULATION

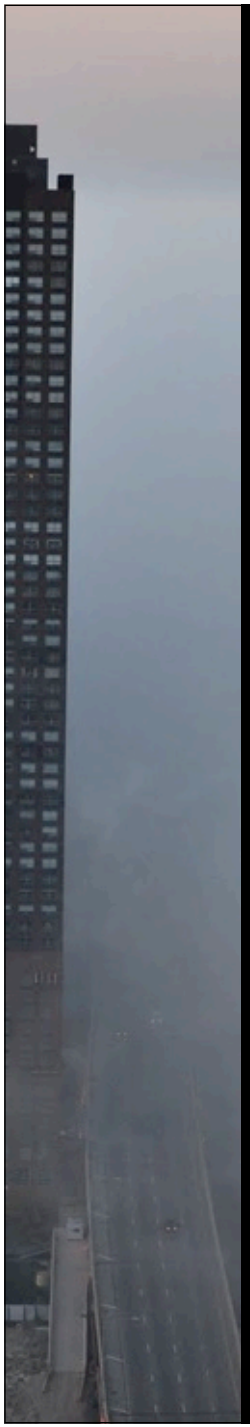
Name	Bioc.	Shoot.	Misc.	CRAN	Base
# Package	630	11	7	1238	27
# Vignettes	100	11	4	–	–
R LOC	1.4M	973	1.3K	2.3M	91K
C LOC	2M	0	0	2.9M	50K

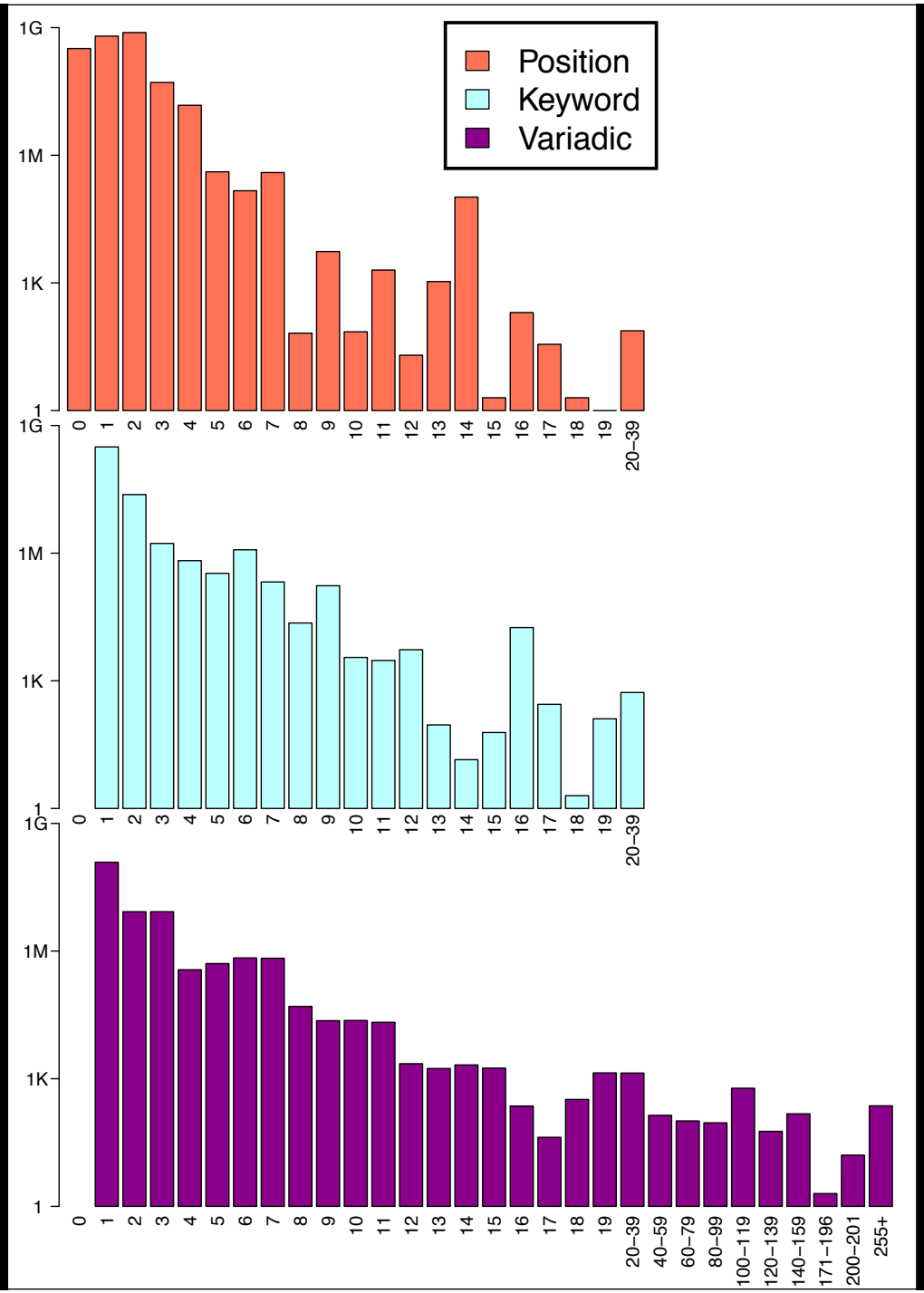
Vectors

```
x <- c(2,7,9,NA,5)

c(1,2,3) + x[1:3]

x[is.na(x)] <- 0
```

Functions

```
q<-function(x=5)x*x*x
q()
q(2)
q(x=4)
p<-function(x=5,...,y=x+1)
```

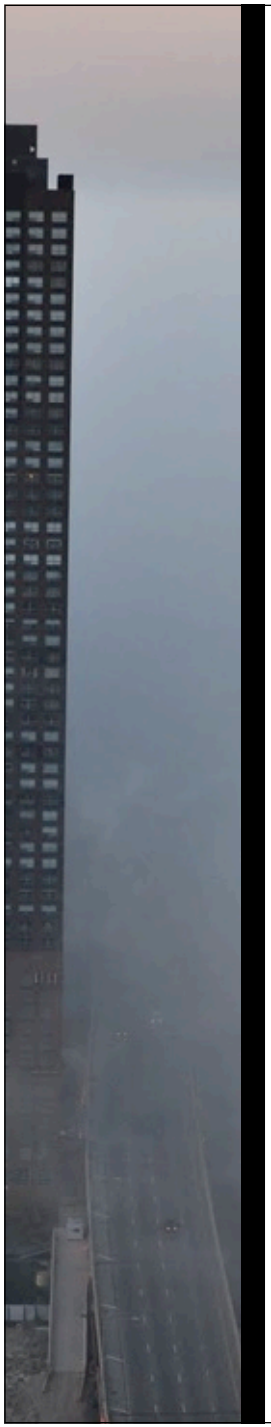


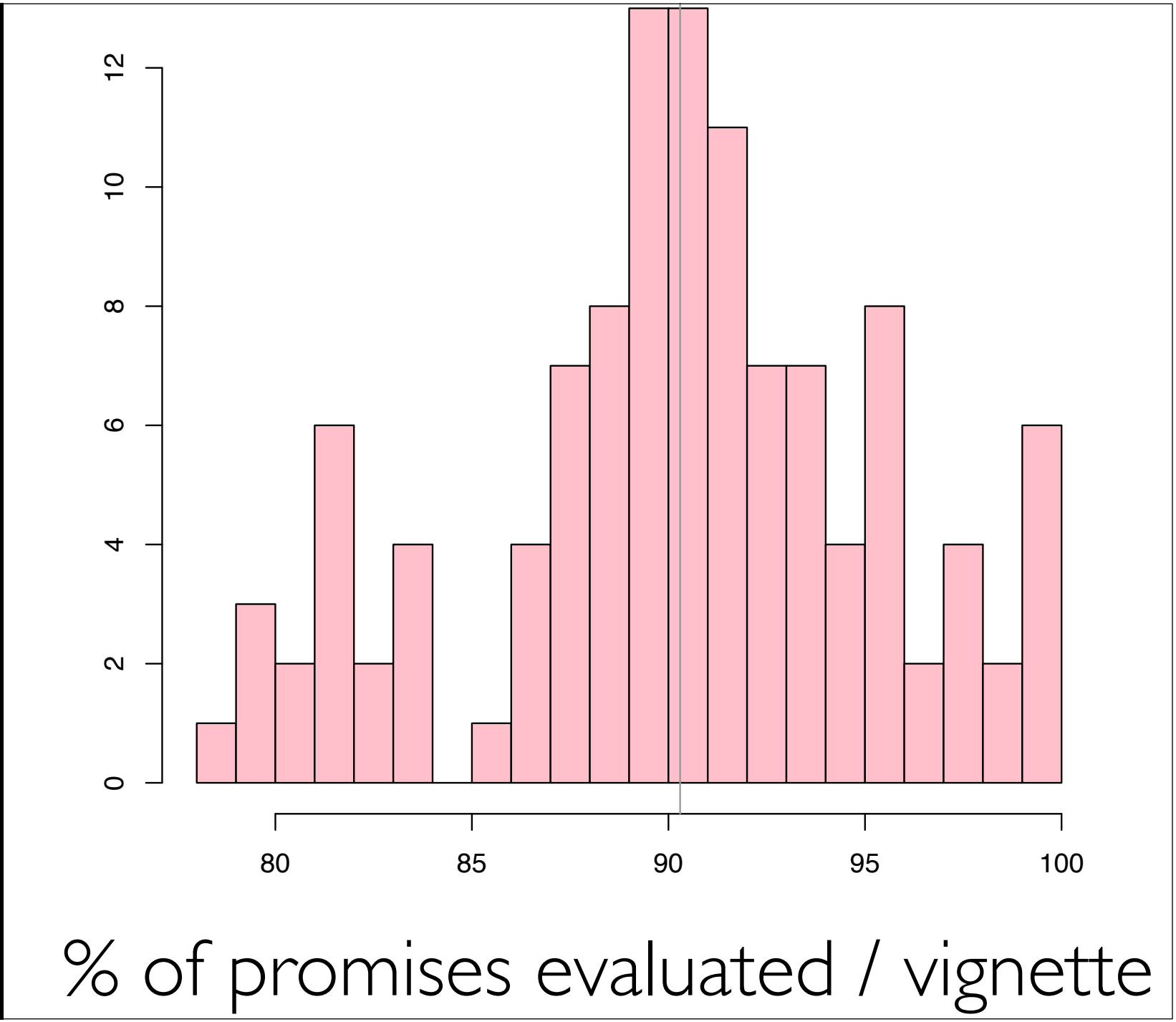



```
f(1, 2)

f(x=1,y=2)
f(y=1,x=2)
f(2,x=1)
f(x=1,2)

c(1,2,3,4)
```

Promises

```
assert<-function(C,P)
            if (C) print(P)

assert( x==42, print("Oops"))
```





% of promises evaluated / vignette

Forcing promises

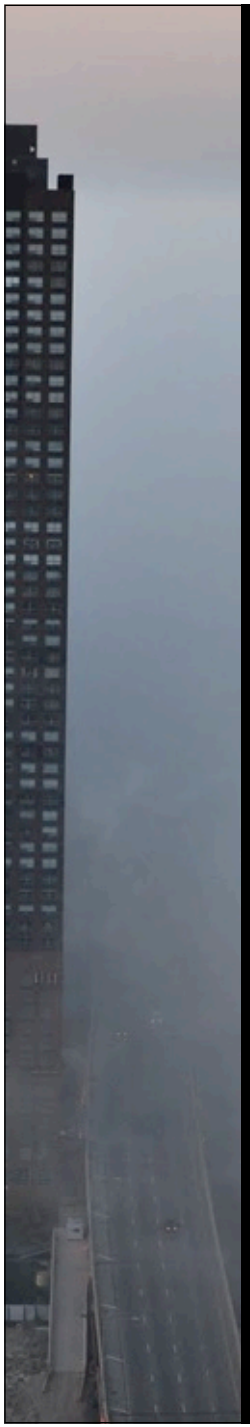
```
x <-  F

x[12] <- F

F ; e

{e ; F}
```

Scoping

Lexical scoping with context sensitive name resolution

```
c <- 42

c(1,2,3)
```

```
c <- 42
d <- c
d(1,2,3)
```



less than 0.05% context sensitive function name lookups

only symbols that rely on it are **`c`** and **`file`**

Referential transparency

```
assert(y[[1]]==5)

f(y)

assert(y[[1]]==5)

f<-function(b){b[[1]]<-0}
```

Assignment

```
x [ 42 ] <- y
```

$$\frac{\mathsf{cpy}(H, \nu') = H', \nu'' \quad \Gamma = \iota * \Gamma' \quad \iota(H', \mathtt{x}) = \nu''' \quad \mathsf{readn}(\nu, H') = \mathtt{m} \quad \mathsf{set}(\nu''', \mathtt{m}, \nu'', H') = H''}{\mathtt{x}[[\nu]] <- \nu' \; \Gamma; H \;\rightarrow\; \nu'; H''}$$

Assignment

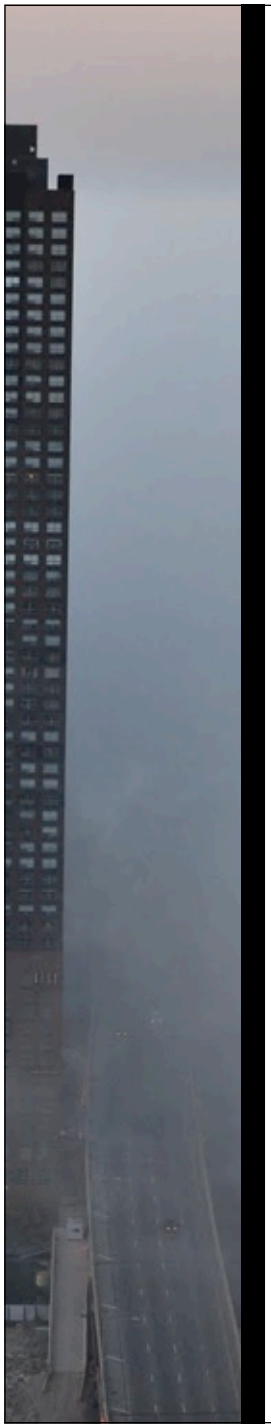
```
y <- c(…)
f <- function() {
   x [ 42 ] <- y
```

$$\frac{\mathsf{cpy}(H, \nu') = H', \nu'' \quad \Gamma = \iota * \Gamma' \quad H'(\iota) = F \quad \mathtt{x} \notin F \quad \Gamma'(H', \mathtt{x}) = \nu''' \quad \mathsf{cpy}(H', \nu''') = H'', \nu'''' \quad F' = F[\mathtt{x}/\nu''''] \quad H''' = H''[\iota/F'] \quad \mathsf{readn}(\nu, H) = \mathtt{m} \quad \mathsf{set}(\nu'''', \mathtt{m}, \nu'', H''') = H''''}{\mathtt{x}[[\nu]] <\!- \nu' \; \Gamma; H \;\rightarrow\; \nu'; H''''}$$

Assignment

```
x [ 42 ] <<- y
```

$$\frac{\mathsf{cpy}(H, \nu) = H', \nu' \qquad \Gamma = \iota * \Gamma' \qquad \mathsf{assign}(\mathtt{x}, \nu', \Gamma', H') = H''}{\mathtt{x} \mathrel{<\!\!<\!\!-} \nu\ \Gamma; H \ \rightarrow\ \nu; H''}$$



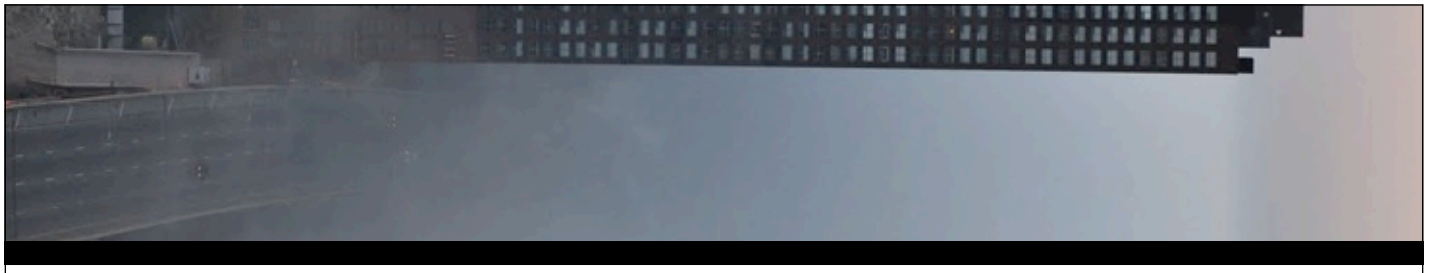
45% of assignments are definitions

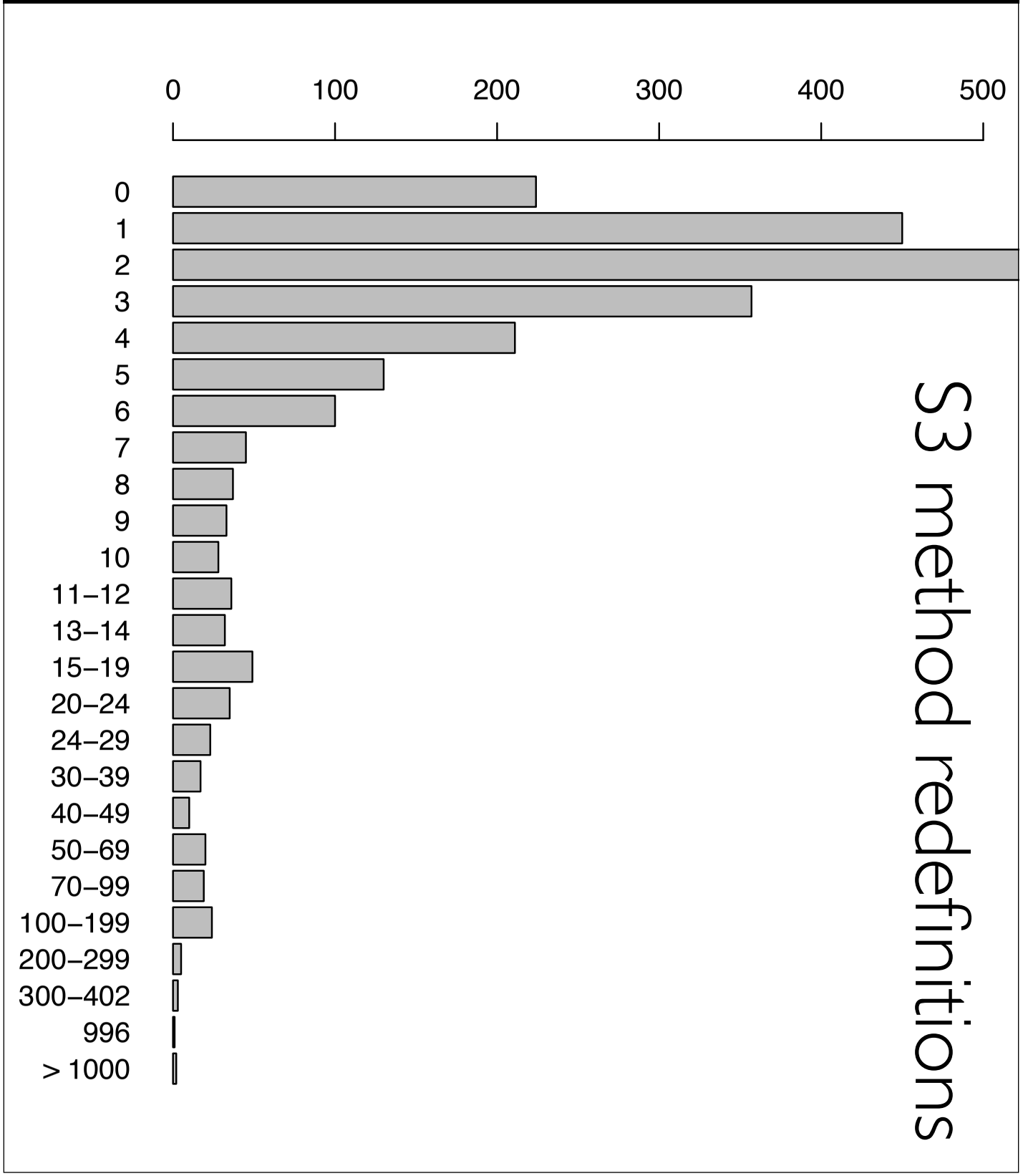
only 2 out of 217 million assignments are definitions in a parent frame

99.9% of side effects are local

Objects

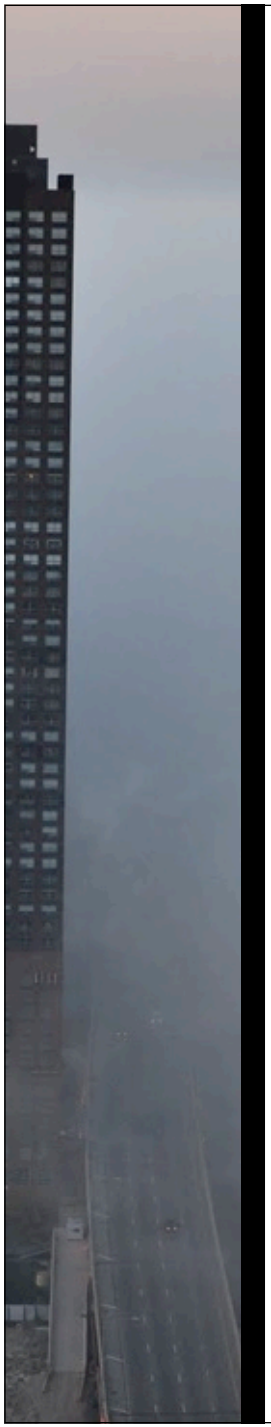
```
who <-function(x) UseMethod("who")

who.man <-function(x) print("Ceasar!")

who.default <-function(x)print("??")

me <- 42;              who(me)
class(me)<-'man'; who(me)
```




S3 method redefinitions
0
100
200
300
400
500
0
1
2
3
4
5
6
7
8
9
10
11–12
13–14
15–19
20–24
24–29
30–39
40–49
50–69
70–99
100–199
200–299
300–402
996
> 1000

Objects

```
setClass("P",representation(x="numeric",y="numeric"))
setClass("C",representation(color="character"))
setClass("CP",contains=c("P","C"))

r <- new("CP", x = 0, y = 0, color = "red")
r@color

setGeneric("add", function(a, b) standardGeneric("add"))
setMethod("add", signature("P","P"),
  function(a, b) new("P", x=a@x+b@x, y=a@y+b@y))
setMethod("add",signature("CP", "CP"),
  function(a, b)new("CP",x=a@x+b@x,y=a@y+b@y,color=a@color)
```



Object usage

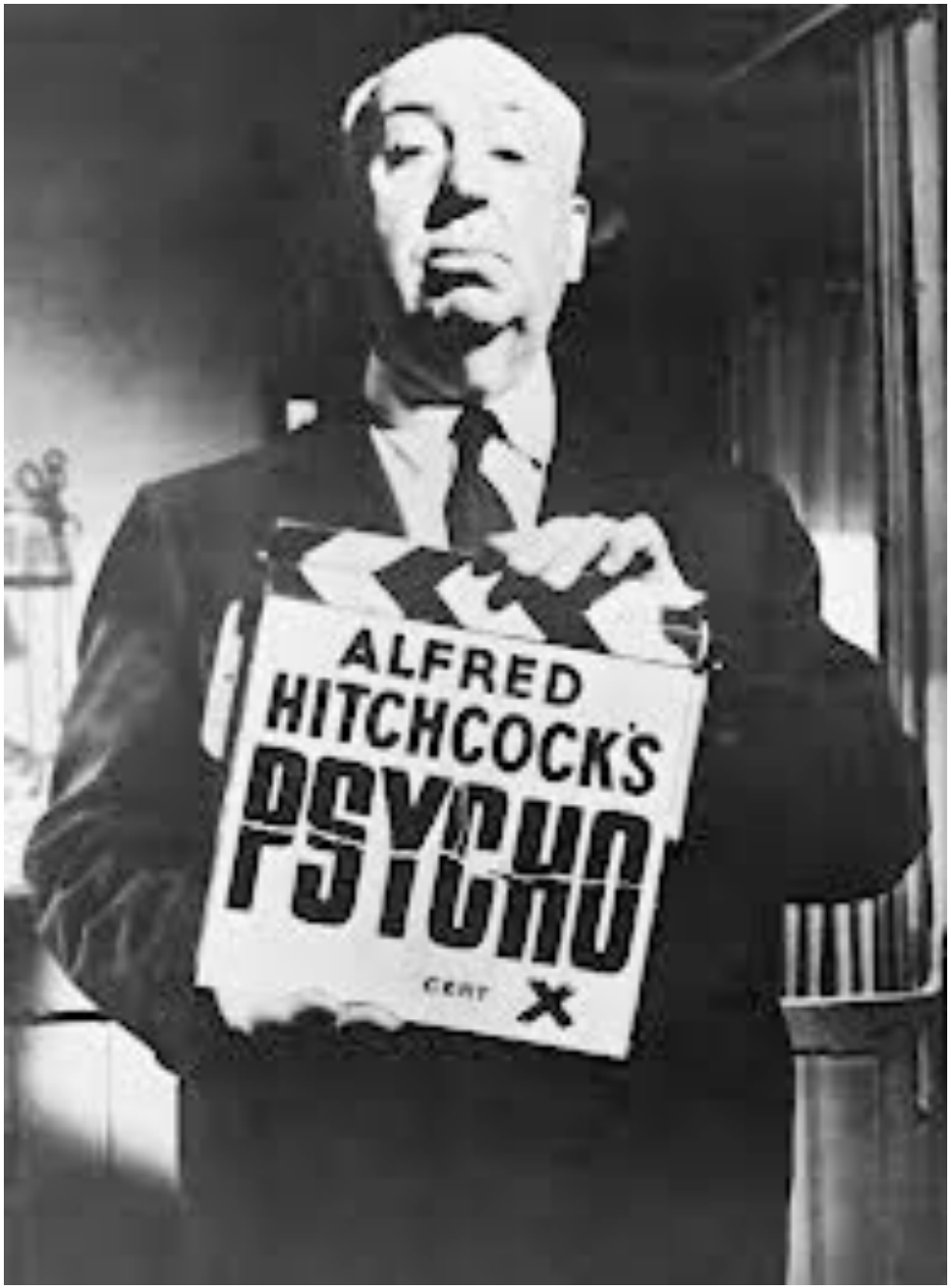
		Bioc	Misc	CRAN	Base	Total
S3	# classes	1 535	0	3 351	191	3 860
	# methods	1 008	0	1 924	289	2 438
	Avg. redef.	6.23	0	7.26	4.25	9.75
	Method calls	13M	58M	-	-	76M
	Super calls	697K	1.2M	-	-	2M
S4	# classes	1 915	2	1 406	63	2 893
	# singleton	608	2	370	28	884
	# leaves	819	0	621	16	1 234
	Hier. depth	9	1	8	4	9
	Direct supers	1.09	0	1.13	0.83	1.07
	# methods	4 136	22	2 151	24	5 557
	Avg. redef.	3	1	3.9	2.96	3.26
	Redef. depth	1.12	1	1.21	1.08	1.14
	# new	668K	64	-	-	668K
	Method calls	15M	266	-	-	15M
	Super calls	94K	0	-	-	94K

The trouble with R ?

- R is slow because it lacks a JIT
- R is a memory hog because it has large objects, allocates profusely and a non-moving garbage collector
- R has tricky semantics rife with cobwebs and dark corners

R is legacy software that must be maintained, yet it must also evolve to meet new challenges

R in Java



Why Java ?

- Because JavaScript may not be fast enough
- Leverage a runtime system with zero-maintenance
- Get a good just-in-time compiler
- OpenJDK is an open source platform

Why not Java ?

- The R community dislikes Java
- Interaction with C is cumbersome
- Complete break with GNU-R

FastR Architecture

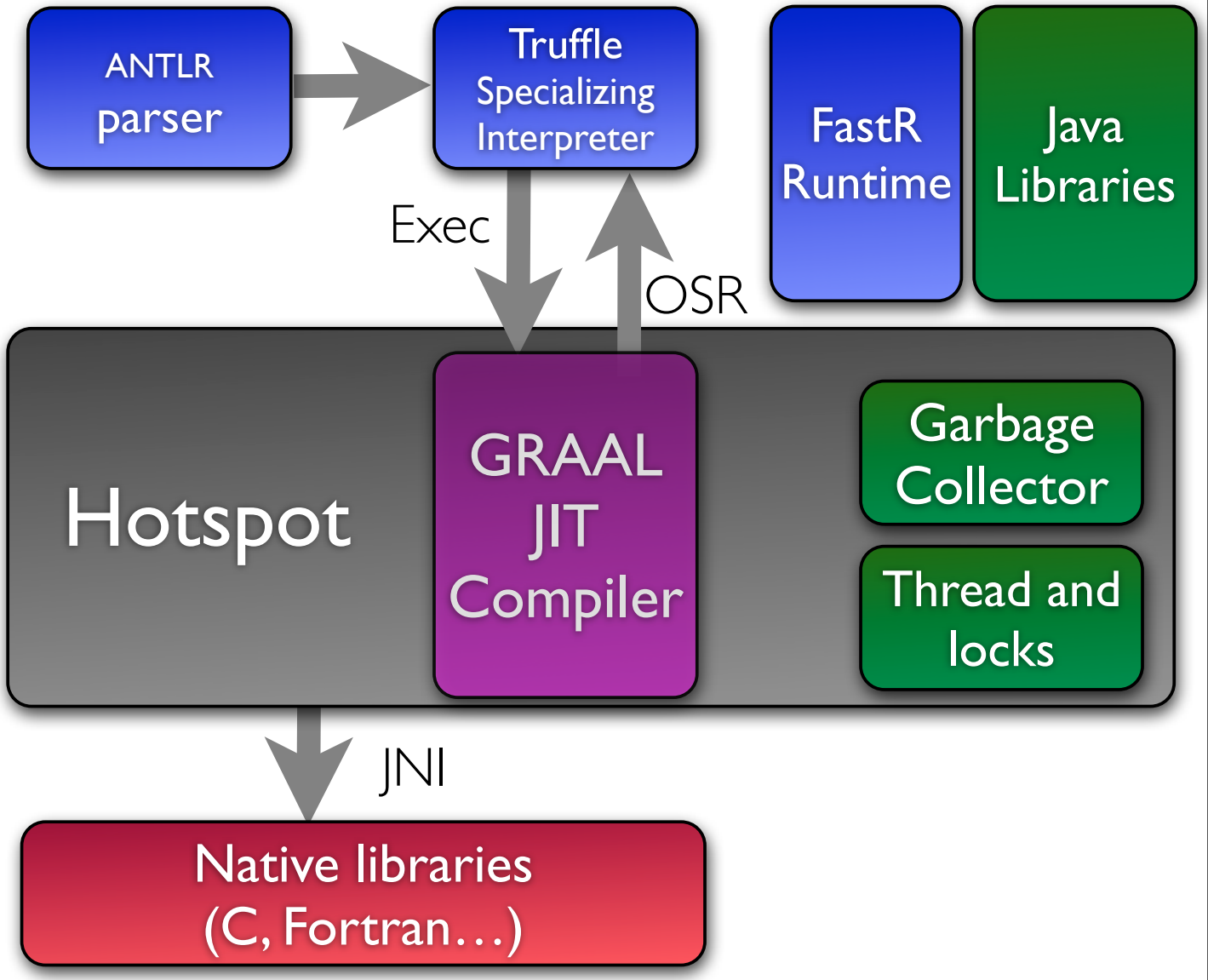
R interpreter written in Java

47 KLOC

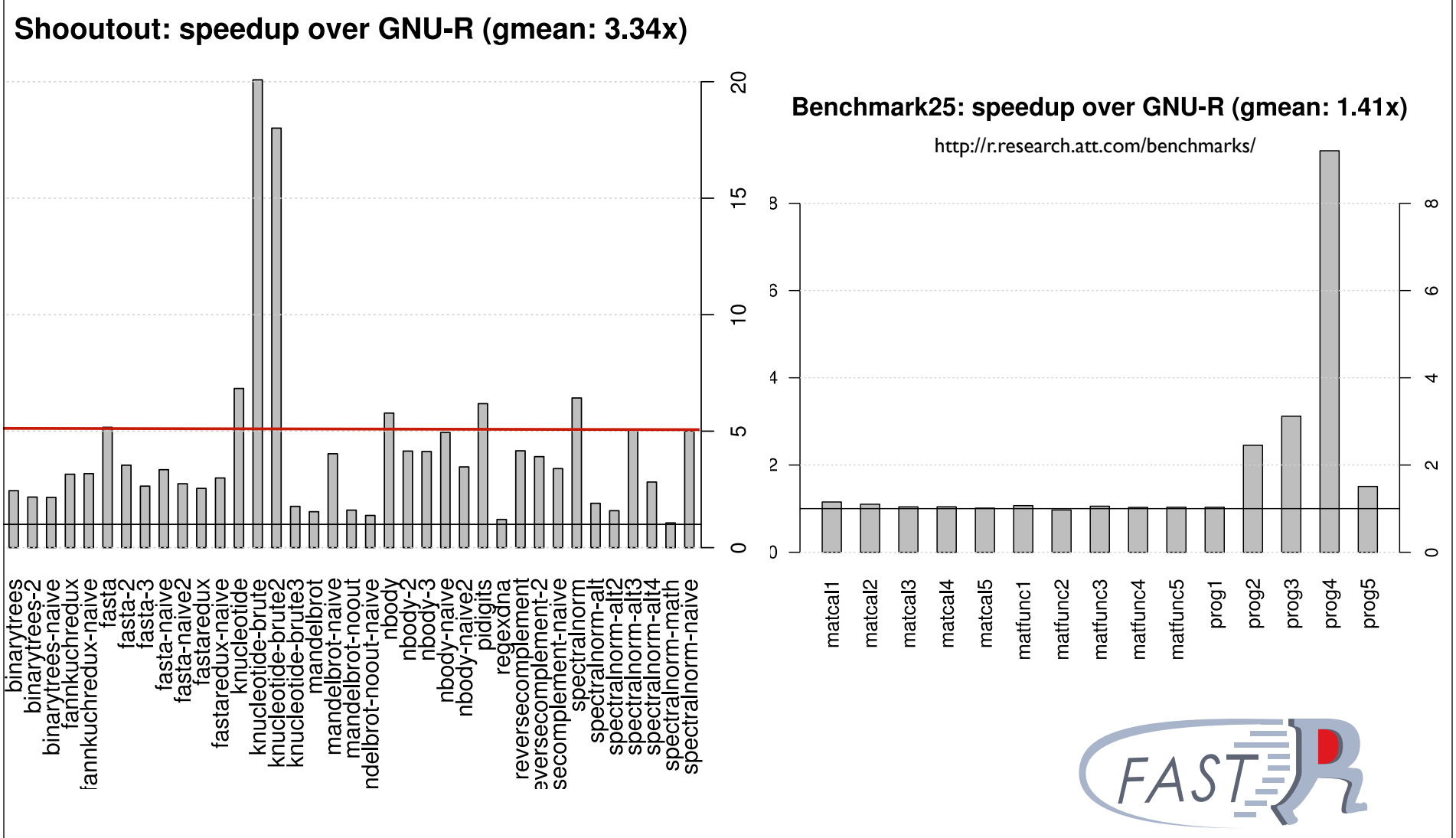
Up to 20x faster on shootout

GPL license

Built on OpenJDK, Graal, Truffle

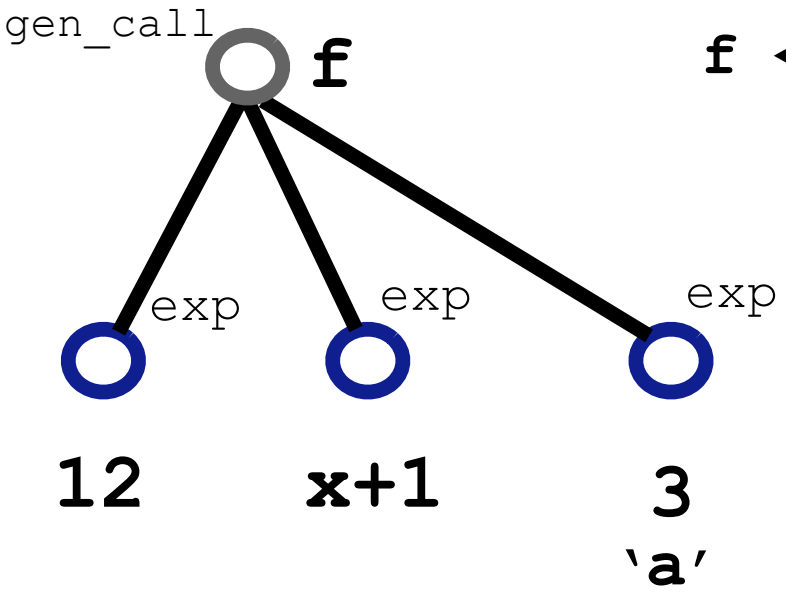


FastR Throughput (w/o JIT)



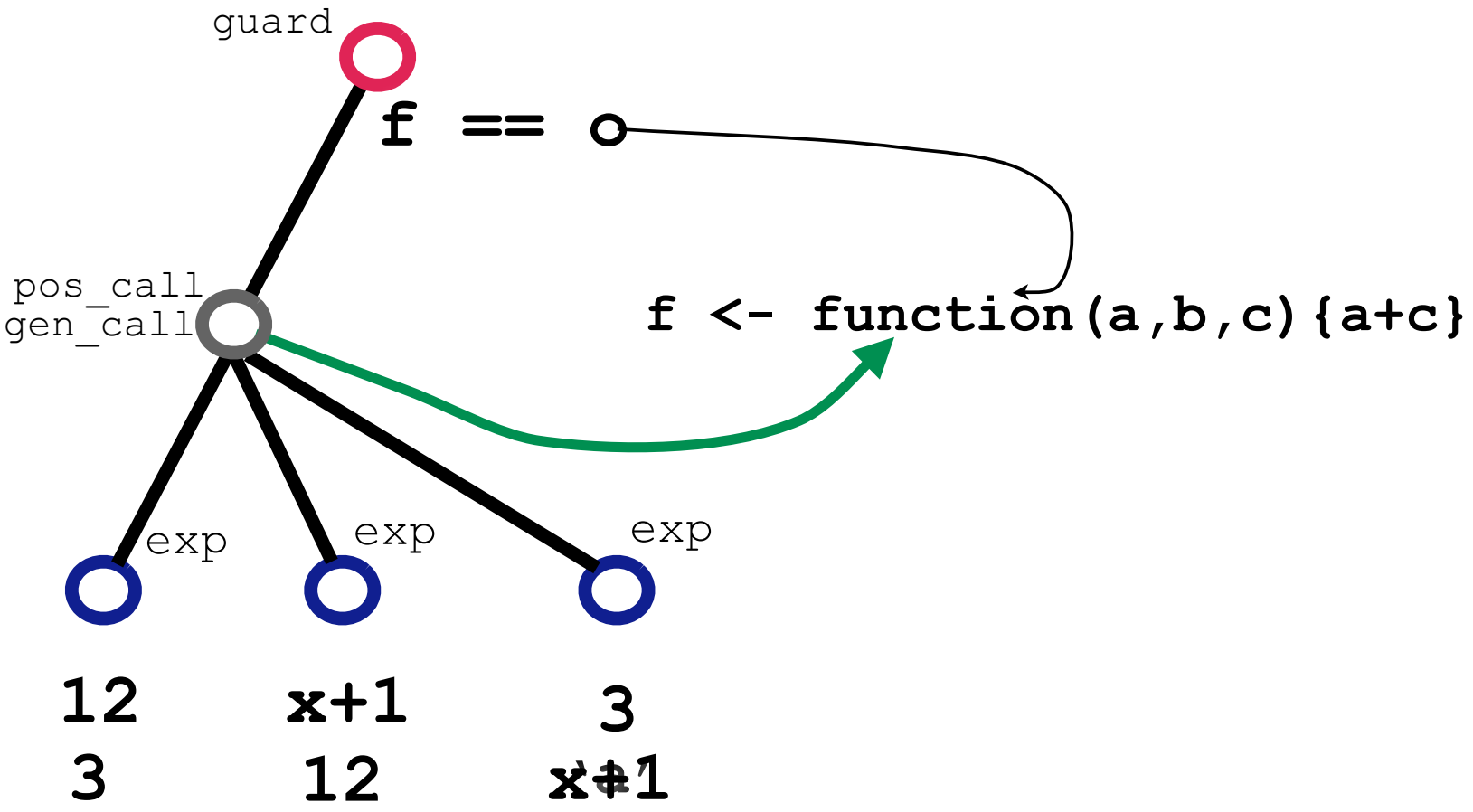
Interpretation

f(12, x+1, a=3)



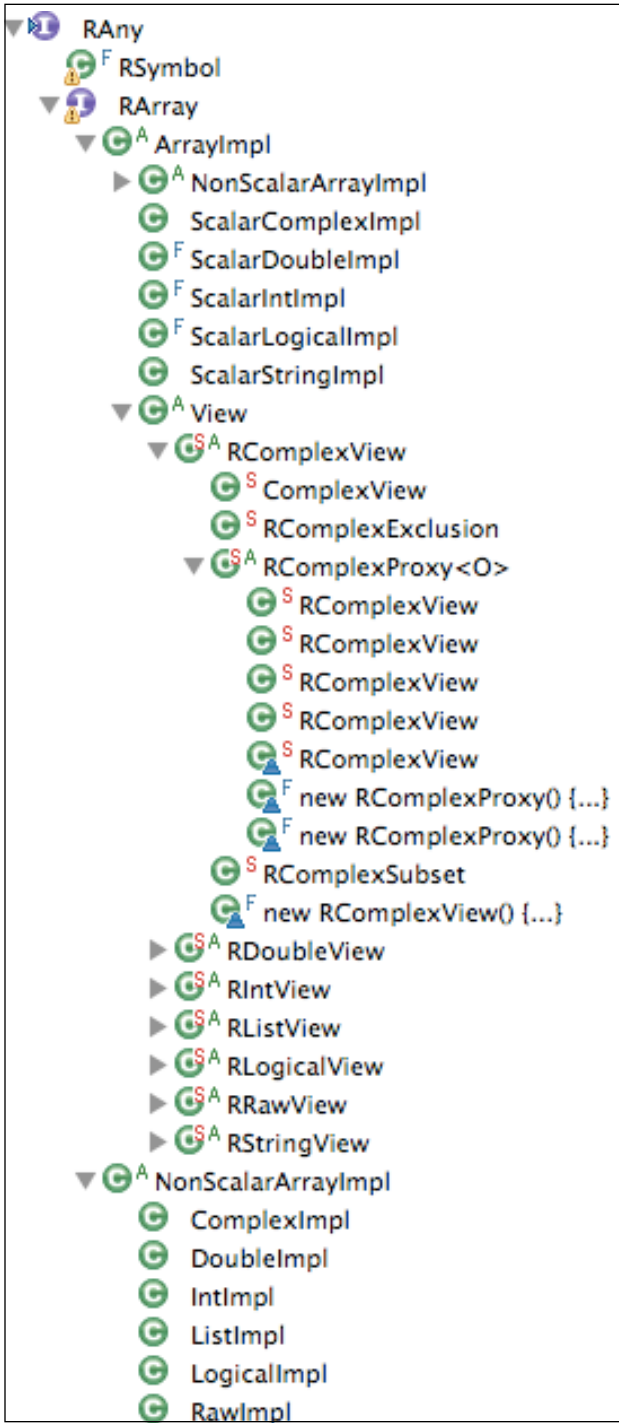

```
f <- function(a,b,c){a+c}
```

Specialization



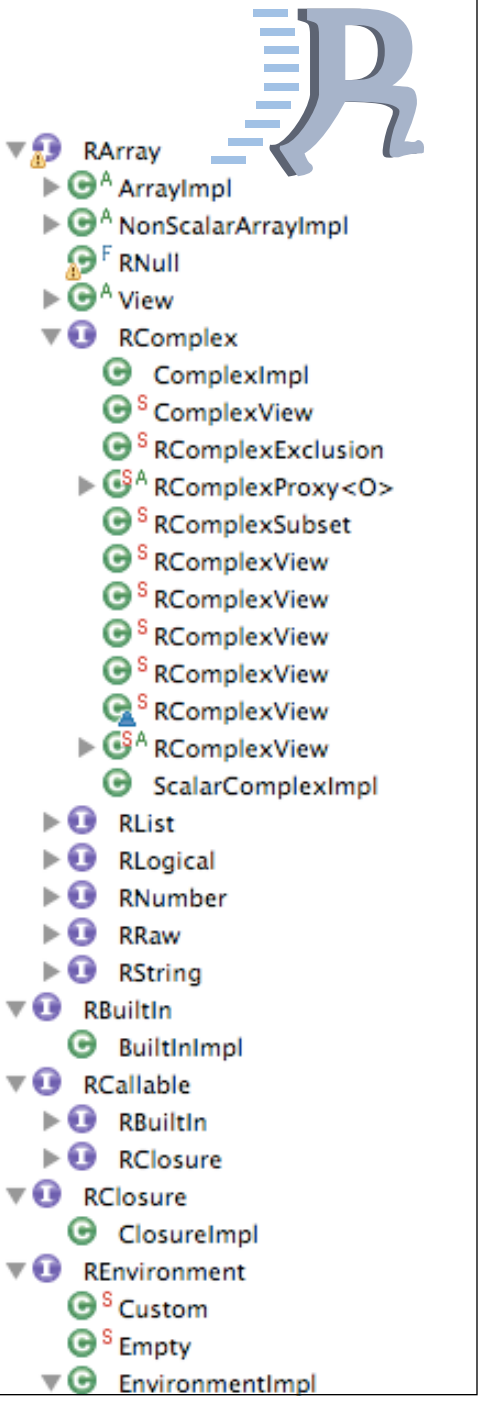
Runtime specialization

```
class If {
  RNode condE, trueB, falseB;

  Object execute(Frame f) {
      try {
        val = condE.executeScalarLogical(frame);
      } catch (UnexpectedResult e) {
        cast = ToLogical.mkNode(condE, e.result());
        replaceChild(condE, cast);
        return execute(frame);
      }
      if (val == TRUE)  return trueB.execute(f);
      if (val == FALSE) return falseB.execute(f);
      throw unexpectedNA();
}
```

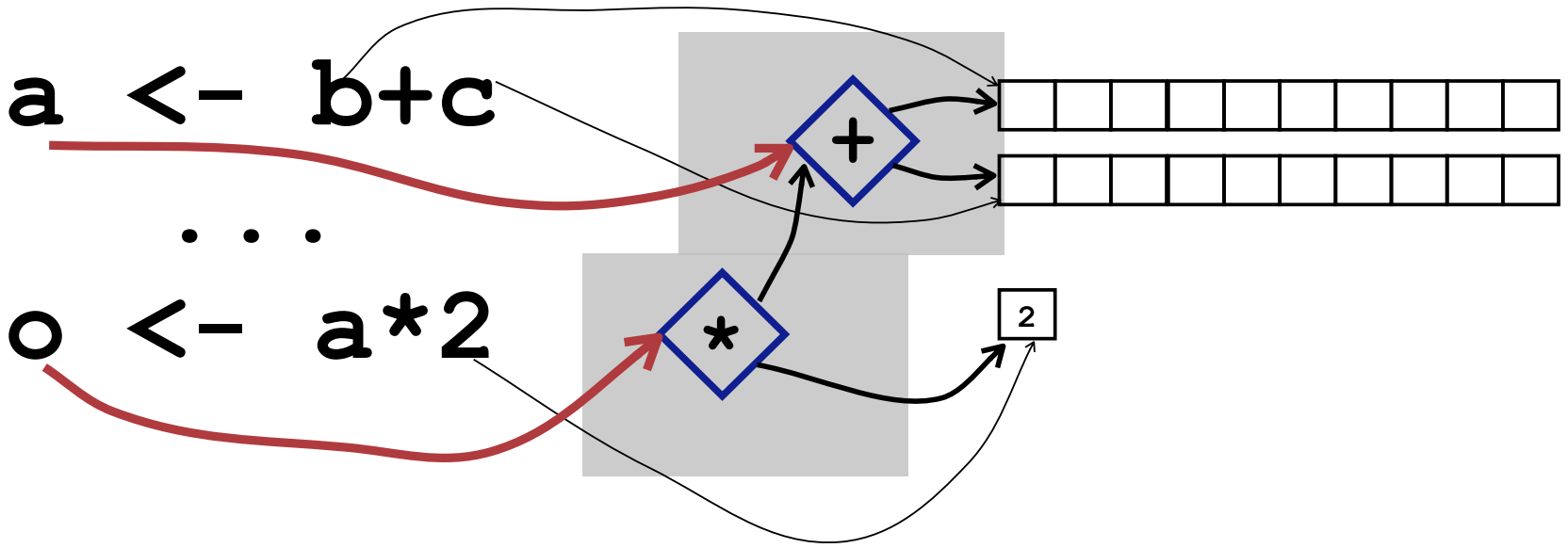


Data types

- Data types are specialized to optimize memory usage and speed
- Scalar are unboxed when possible (soon)
- Memory footprint is improved



Views



- …delay construction of large data objects
- …are a data-flow representation of vectors
- …avoid unnecessary work if a subset of the data is required
- …avoid allocation of temporary objects
- …permit fusion of multiple data traversals into one

Experiments

Inlining

spectralnorm-naive

```
A <- function(i, j) {
  1 / ((i + j) * (i + j + 1) / 2 + i + 1)  }

B <- function(u) { ...
  for (j in 0:n1)
    ret[[i]] <- ret[[i]] + u[[j +
                  *  A(i - 1, j)
```

1.8x

- Inlining is a critical optimization in modern languages
- Replace a function call with it's body
- Guarded inlining leaves a slow path in the code that performs the normal function

Relite

Delite is a compiler framework and runtime for parallel embedded DSLs.

Delite provides:

- Built-in parallel execution patterns
- Optimizers for parallel code
- Code generators for Scala, C++ and CUDA
- A heterogeneous runtime for executing DSLs

Relite is a proof-of-concept R interface to Delite.

`https://github.com/TiarkRompf/Relite`

Relite

```
sapply(1:50000, function(x){sum(1:x)})
```

4.1s GNU-R

1.5s FastR

```
Delite(sapply(1:50000,function(x) {sum(1:x)}))
```

0.4s Relite

```
sapply(1:50000,function(x) {sum((1:x)*0.1)})
```

9.0s GNU-R

2.2s FastR

0.5s Relite

```
sapply(1:50000,
  function(x){ sum(sapply(1:x, function(y) y*0.1))})
```

2395s GNU-R

104s FastR

0.5s Relite

Conclusions

- R is an amazingly successful systems with great mindshare
- The R implementation is hard to maintain and evolve
- The FastR project aims to rethink how to implement R
- We leverage well tested technologies to build a high performance VM
- FastR can be a source of inspiration for GNU R