Rcpp

Making R Applications Go Faster and Further

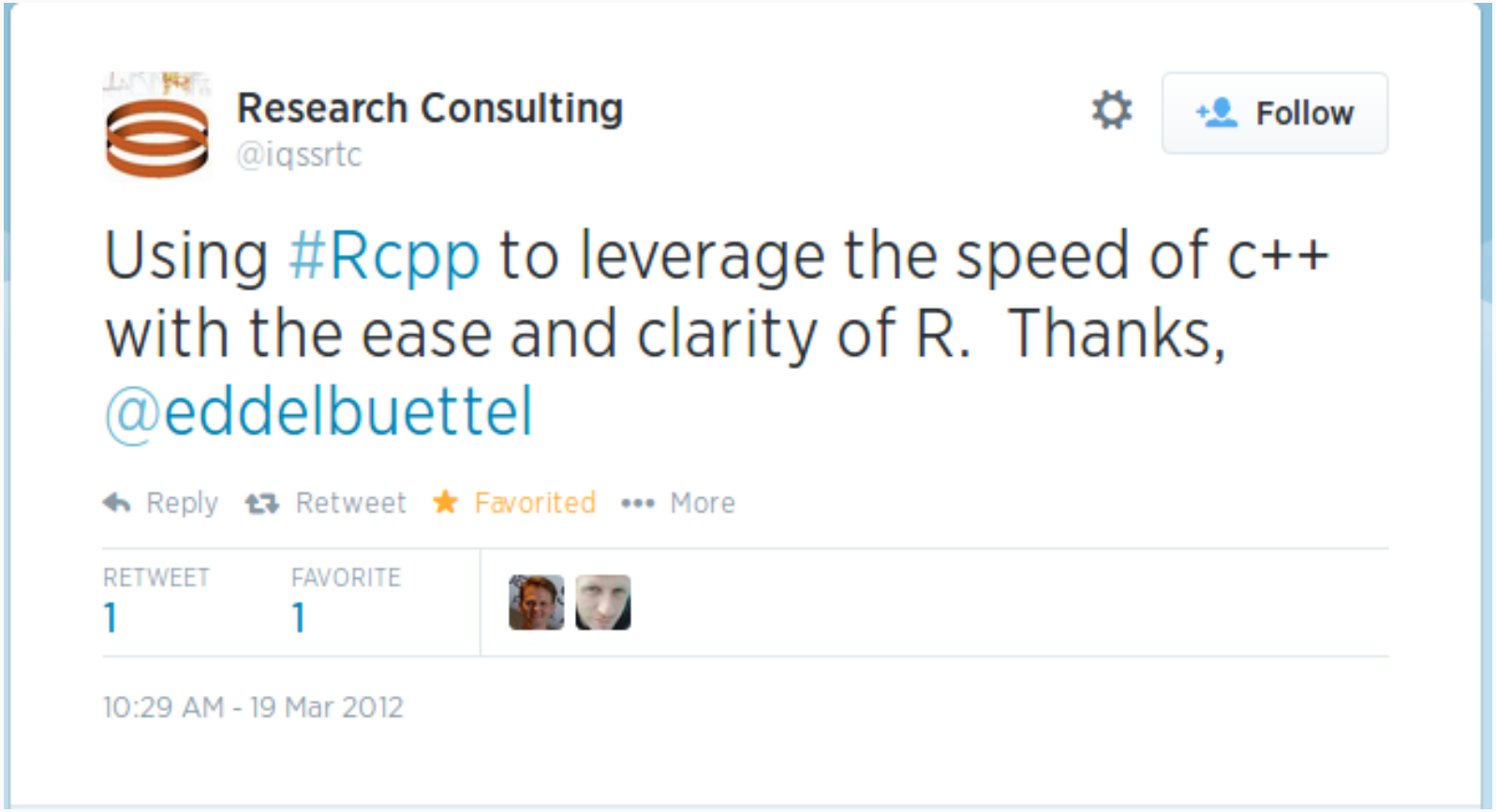
Dirk Eddelbuettel

EARL 2015 Keynote Address

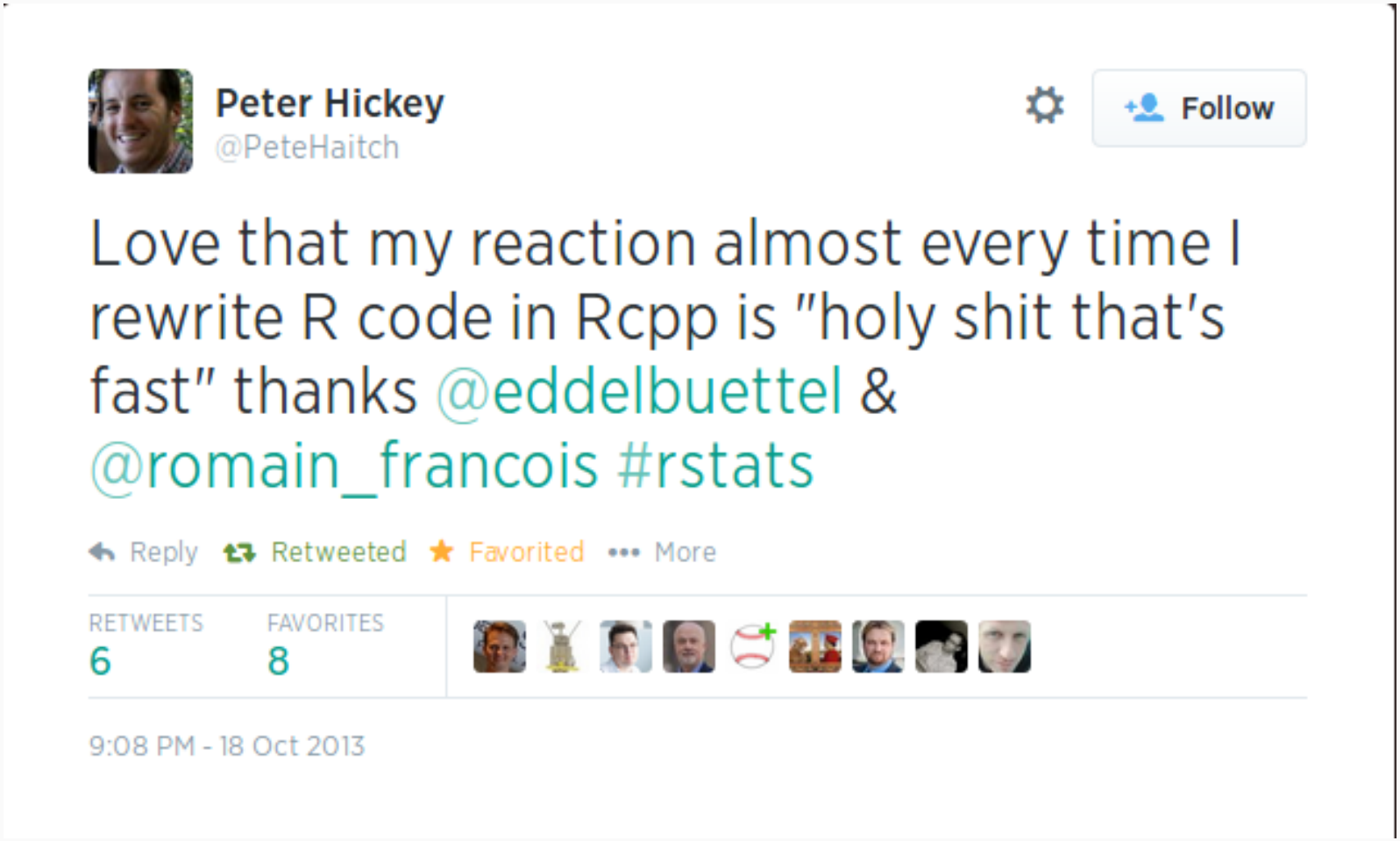
September 16, 2015

Introduction

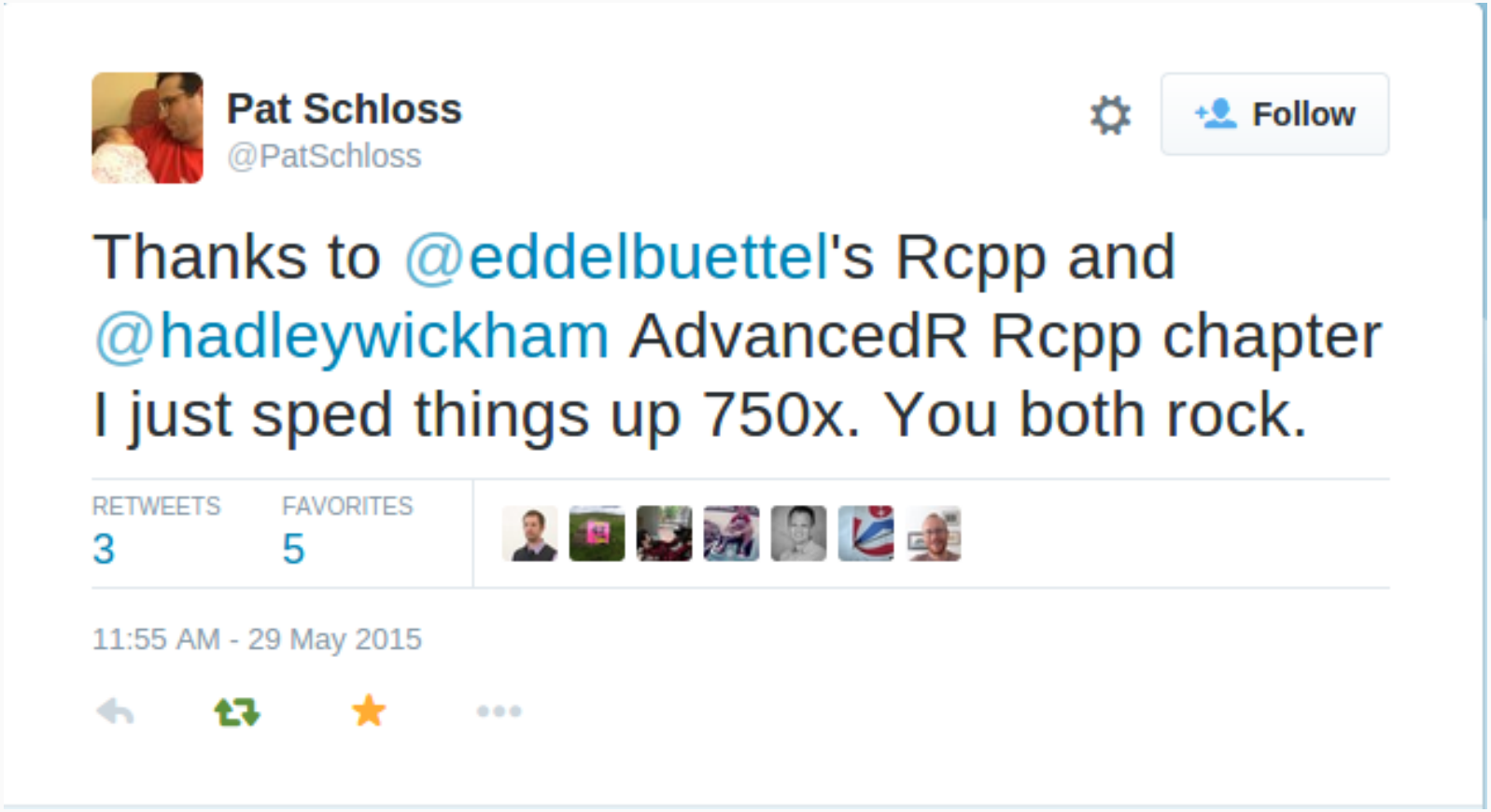
A Very Kind Tweet



And Another Tweet



And Yet Another Tweet

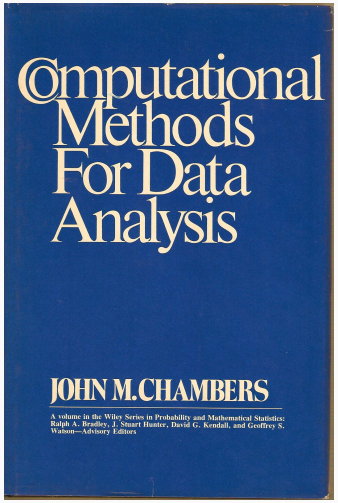


And Why Not Another Tweet

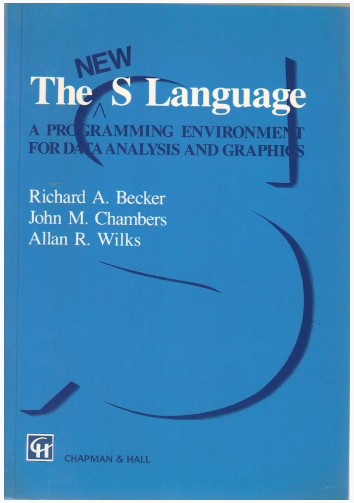
And Last But Not Least

Extending R

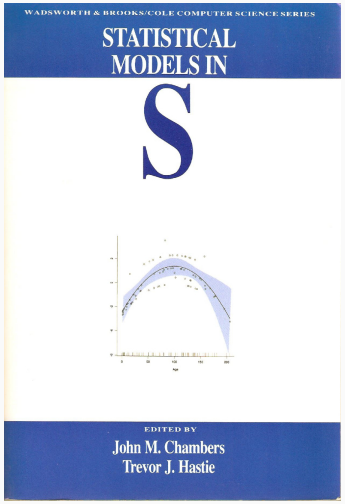
Why R? : *Programming with Data*



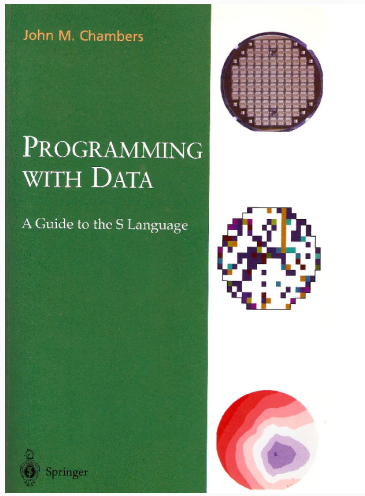
Chambers, *Computational Methods for Data Analysis*. Wiley, 1977.



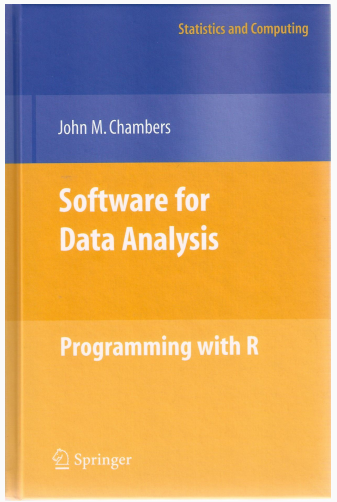
Becker, Chambers, and Wilks. *The New S Language*. Chapman & Hall, 1988.



Chambers and Hastie. *Statistical Models in S*. Chapman & Hall, 1992.



Chambers. *Programming with Data*. Springer, 1998.



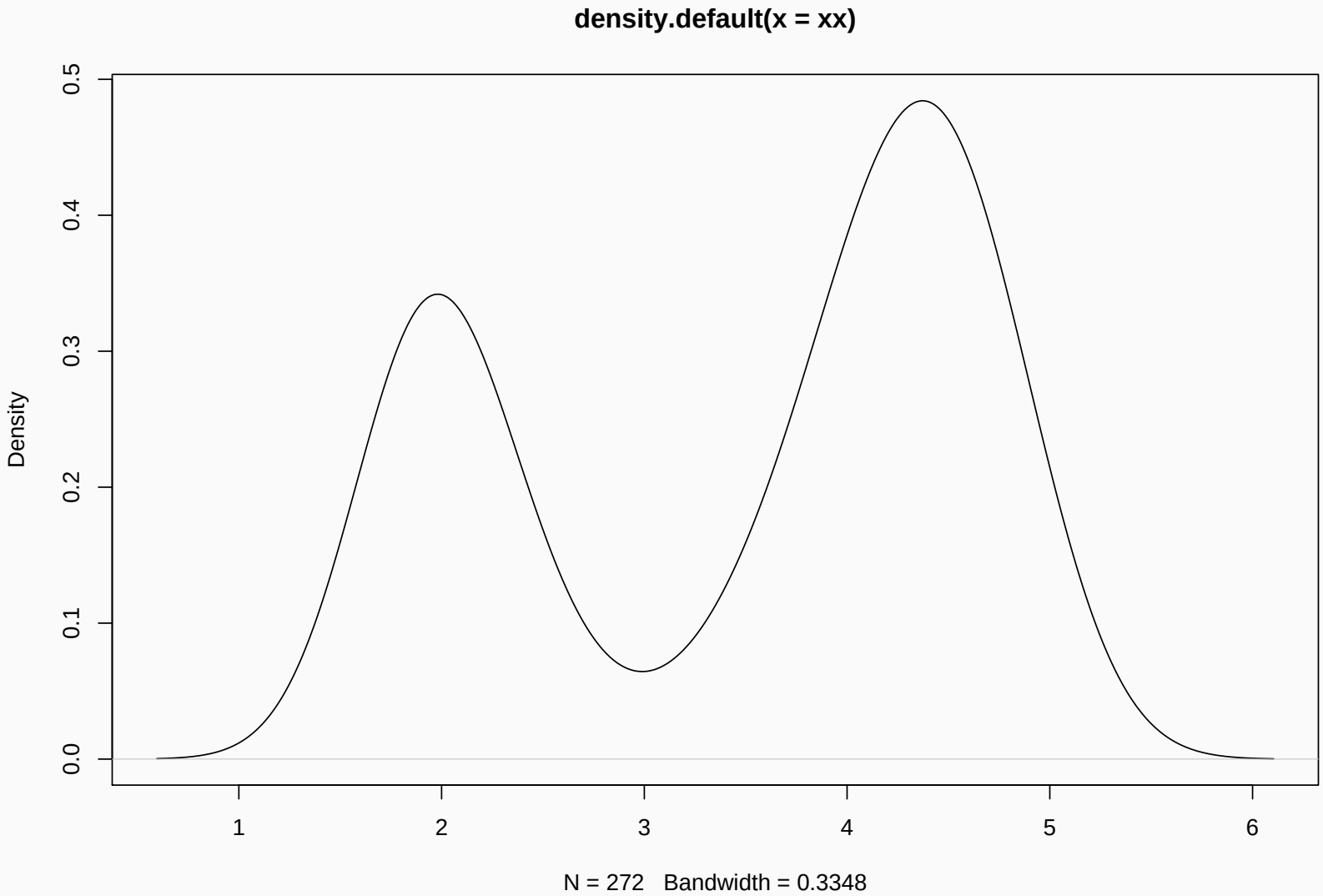
Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008

Thanks to John Chambers for sending me high-resolution scans of the covers of his books.

A Simple Example

```
xx <- faithful[,"eruptions"]
fit <- density(xx)
plot(fit)
```

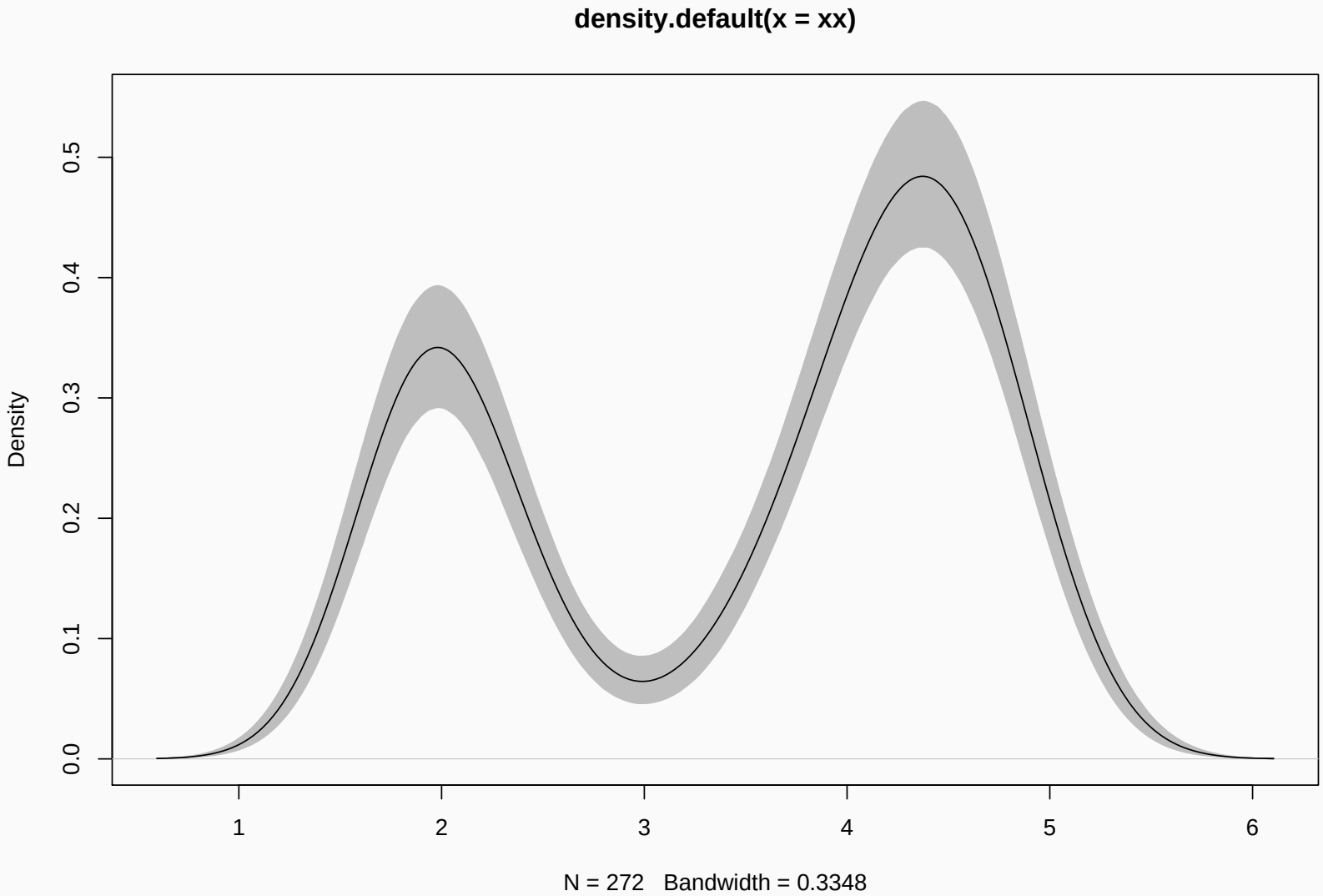
A Simple Example



A Simple Example - Refined

```
xx <- faithful[,"eruptions"]
fit1 <- density(xx)
fit2 <- replicate(10000, {
    x <- sample(xx,replace=TRUE);
    density(x, from=min(fit1$x), to=max(fit1$x))$y
})
fit3 <- apply(fit2, 1, quantile,c(0.025,0.975))
plot(fit1, ylim=range(fit3))
polygon(c(fit1$x,rev(fit1$x)), c(fit3[1,],rev(fit3[2,])),
    col='grey', border=F)
lines(fit1)
```

A Simple Example - Refined



So Why R?

R enables us to

- work interactively
- explore and visualize data
- access, retrieve and/or generate data
- summarize and report into pdf, html, ...

making it the key language for statistical computing, and a preferred environment for many data analysts.

So Why R?

R has always been extensible via

- C via a bare-bones interface described in *Writing R Extensions*
- Fortran which is also used internally by R
- Java via `rJava` by Simon Urbanek
- C++ but essentially at the bare-bones level of C

So while *in theory* this always worked – it was tedious *in practice*

Why Extend R?

Chambers (2008), opens Chapter 11 *Interfaces I: Using C and Fortran*:

> *Since the core of R is in fact a program written in the C language, it's not surprising that the most direct interface to non-R software is for code written in C, or directly callable from C. All the same, including additional C code is a serious step, with some added dangers and often a substantial amount of programming and debugging required. You should have a good reason.*

Why Extend R?

Chambers (2008), opens Chapter 11 *Interfaces I: Using C and Fortran*:

> *Since the core of R is in fact a program written in the C language, it's not surprising that the most direct interface to non-R software is for code written in C, or directly callable from C. All the same, including additional C code is a serious step, with some added dangers and often a substantial amount of programming and debugging required. You should have a good reason.*

Why Extend R?

Chambers proceeds with this rough map of the road ahead:

- Against:
  - It's more work
  - Bugs will bite
  - Potential platform dependency
  - Less readable software
- In Favor:
  - New and trusted computations
  - Speed
  - Object references

Why Extend R?

The *Why?* boils down to:

- speed: Often a good enough reason for us ... and a focus for us in this workshop.
- new things: We can bind to libraries and tools that would otherwise be unavailable in R
- references: Chambers quote from 2008 foreshadowed the work on *Reference Classes* now in R and built upon via Rcpp Modules, Rcpp Classes (and also RcppR6)

And Why C++?

- Asking Google leads to about $\sim$ 50 million hits.
- Wikipedia: *C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose, powerful programming language*
- C++ is industrial-strength, vendor-independent, widely-used, and *still evolving*
- In science & research, one of the most frequently-used languages: If there is something you want to use / connect to, it probably has a C/C++ API
- As a widely used language it also has good tool support (debuggers, profilers, code analysis)

Why C++?

Scott Meyers: *View C++ as a federation of languages*

- *C* provides a rich inheritance and interoperability as Unix, Windows, ... are all build on C.
- *Object-Oriented C++* (maybe just to provide endless discussions about exactly what OO is or should be)
- *Templated C++* which is mighty powerful; template meta programming unequalled in other languages.
- *The Standard Template Library* (STL) is a specific template library which is powerful but has its own conventions.
- *C++11* (and C++14 and beyond) add enough to be called a fifth language.

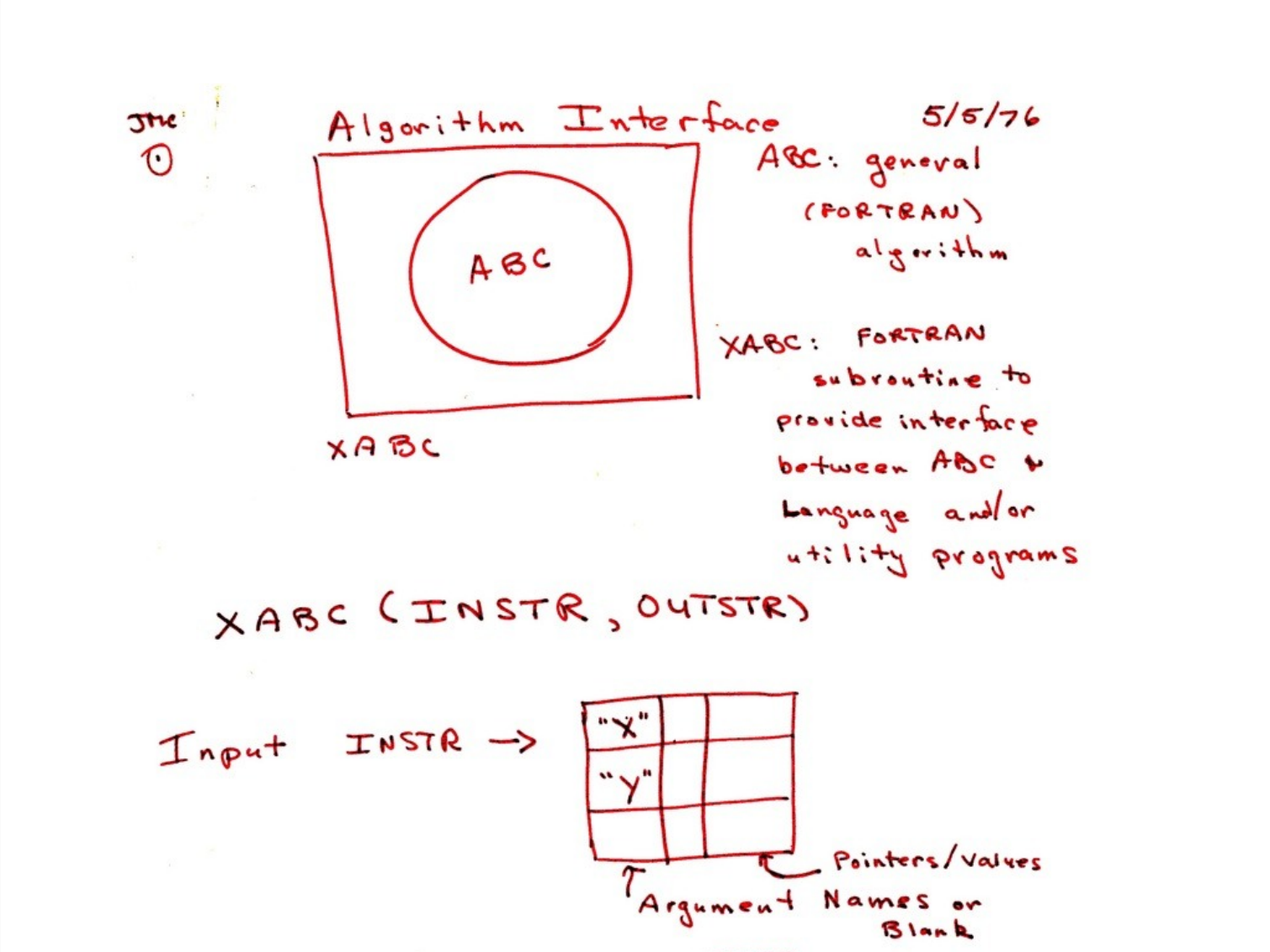
NB: Meyers original list of four languages appeared years before C++11.

Why C++?

- Mature yet current
- Strong performance focus:
  - *You don't pay for what you don't use*
  - *Leave no room for another language between the machine level and C++*
- Yet also powerfully abstract and high-level
- C++11 is a big deal giving us new language features
- While there are complexities, Rcpp users are mostly shielded

Interface Vision

Bell Labs, May 1976



Interface Vision

R offers us the best of both worlds:

- Compiled code with
  - Access to proven libraries and algorithms in C/C++/Fortran
  - Extremely high performance (in both serial and parallel modes)
- Interpreted code with
  - An accessible high-level language made for *Programming with Data*
  - An interactive workflow for data analysis
  - Support for rapid prototyping, research, and experimentation

Why Rcpp?

- Easy to learn as it really does not have to be that complicated – we will see numerous few examples
- Easy to use as it avoids build and OS system complexities thanks to the R infrastrucure
- Expressive as it allows for *vectorised* C++ using *Rcpp Sugar*
- Seamless access to all R objects: vector, matrix, list, S3/S4/RefClass, Environment, Function, ...
- Speed gains for a variety of tasks Rcpp excels precisely where R struggles: loops, function calls, ...
- Extensions greatly facilitates access to external libraries using eg *Rcpp modules*

Speed

Speed Example (due to StackOverflow)

Consider a function defined as

$$\mathrm{f}(n) \quad \text{such that} \quad \begin{cases} n & \text{when} \quad n < 2 \\ \mathrm{f}(n-1) + \mathrm{f}(n-2) & \text{when} \quad n \geq 2 \end{cases}$$

Speed Example in R

R implementation and use:

```
f <- function(n) {
    if (n < 2) return(n)
    return(f(n-1) + f(n-2))
}

## Using it on first 11 arguments
sapply(0:10, f)
```

```
##  [1]  0  1  1  2  3  5  8 13 21 34 55
```

Speed Example Timed

Timing:

```
library(rbenchmark)
benchmark(f(10), f(15), f(20))[,1:4]
```

```
##    test replications elapsed relative
## 1 f(10)          100   0.026    1.000
## 2 f(15)          100   0.327   12.577
## 3 f(20)          100   3.796  146.000
```

Speed Example in C / C++

A C or C++ solution can be equally simple

```
int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2));
}
```

But how do we call it from R?

Matt's Example from useR! 2015

```c
#include <R.h>
#include <Rinternals.h>

int fibonacci_c_impl(int n) {
    if (n < 2) return n;
    return fibonacci_c_impl(n - 1) + fibonacci_c_impl(n - 2);
}

SEXP fibonacci_c(SEXP n) {
    SEXP result = PROTECT(allocVector(INTSXP, 1));
    INTEGER(result)[0] = fibonacci_c_impl(asInteger(n));
    UNPROTECT(1);
    return result;
}

/*
## need to compile, link, load, ...
fibonacci <- function(n) .Call("fibonacci_c", n)
sapply(0:10, fibonacci)
*/
```

One Minor Modification to Matt's Example

```
#include <R.h>
#include <Rinternals.h>

int fibonacci_c_impl(int n) {
    if (n < 2) return n;
    return fibonacci_c_impl(n - 1) + fibonacci_c_impl(n - 2);
}

// [[Rcpp::export]]
SEXP fibonacci_c(SEXP n) {
    SEXP result = PROTECT(allocVector(INTSXP, 1));
    INTEGER(result)[0] = fibonacci_c_impl(asInteger(n));
    UNPROTECT(1);
    return result;
}

/*** R
sapply(0:10, fibonacci_c)
*/
```

Speed Example in C / C++

But Rcpp makes this *much* easier:

```
Rcpp::cppFunction("int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2)); }")
sapply(0:10, g)
```

```
##  [1]  0  1  1  2  3  5  8 13 21 34 55
```

Speed Example Comparing R and C++

Timing:

```
Rcpp::cppFunction("int g(int n) {
   if (n < 2) return(n);
   return(g(n-1) + g(n-2)); }")
library(rbenchmark)
benchmark(f(25), g(25), order="relative")[,1:4]
```

```
##    test replications elapsed relative
## 2 g(25)          100   0.099    1.000
## 1 f(25)          100  47.787  482.697
```

A nice gain of a few orders of magnitude.

Another Angle on Speed

Run-time performance is just one example.

Time to code is another metric.

We feel quite strongly that helps you code more succinctly, leading to fewer bugs and faster development.

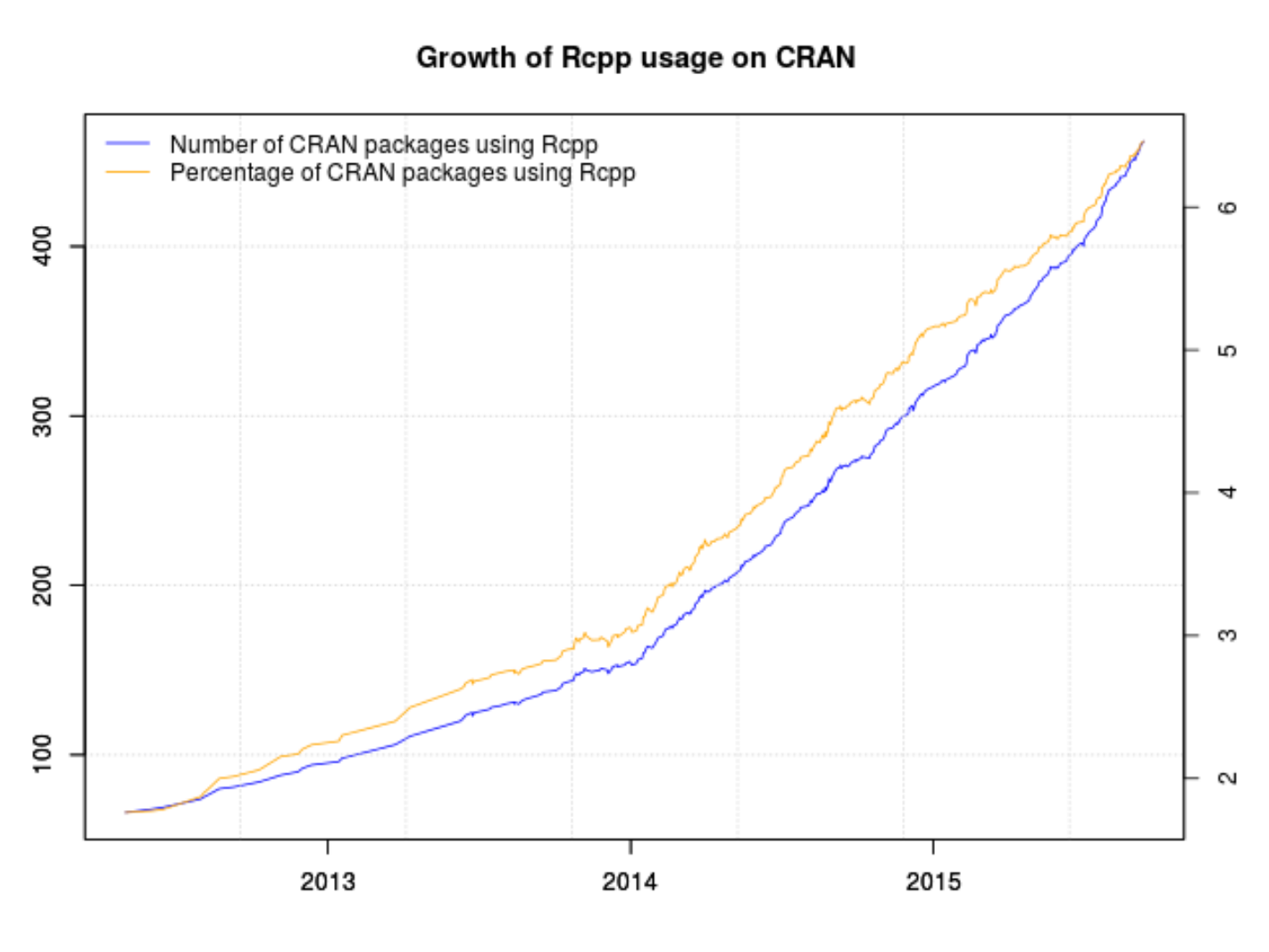
A good environment helps. RStudio integrates R and C++ development quite nicely (eg the compiler error message parsing is very helpful) and also helps with package building.

Speed Example Footnote Also Due to Matt

```
#include <Rcpp.h>

// [[Rcpp::plugins("cpp11")]]

constexpr int fibonacci_recursive_constexpr(const int n) {
    return n < 2 ? n : (fibonacci_recursive_constexpr(n - 1) +
                        fibonacci_recursive_constexpr(n - 2));
}

// [[Rcpp::export]]
int constexprFib() {
    const int N = 42;
    constexpr int result = fibonacci_recursive_constexpr(N);
    return result;
}
```

Popularity

Used by 462 CRAN Packages as of last weekend



Page Rank One (According to Andrie de Vries)

Application Spotlight: Rblpapi

History: Basic package using the C (Dirk)

r_lim_bloomberg.pdf

1 of 19 61.84%

Programming with financial data: Connecting R to Lim and Bloomberg

Dirk Eddelbuettel

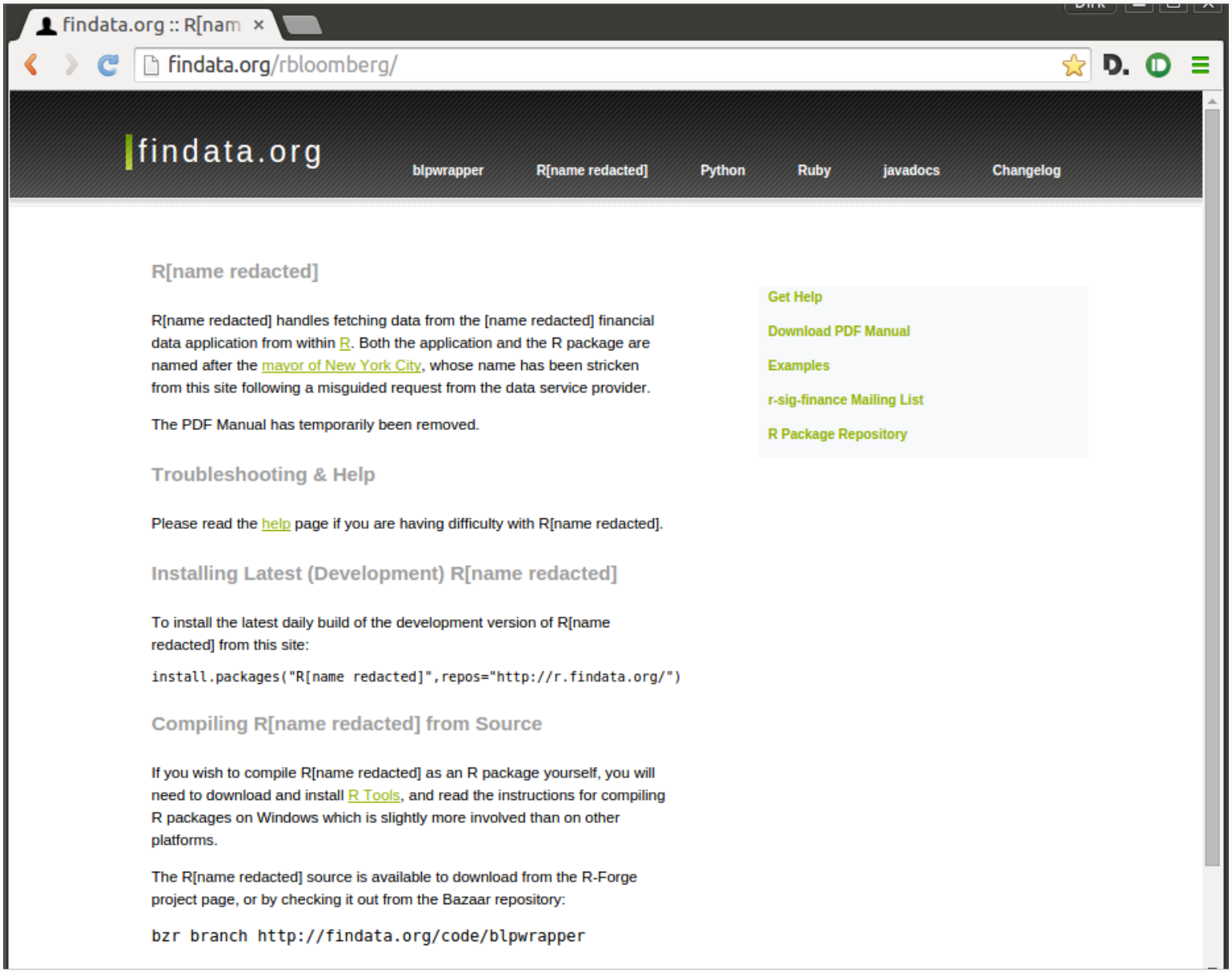
B of A and Debian

edd@debian.org

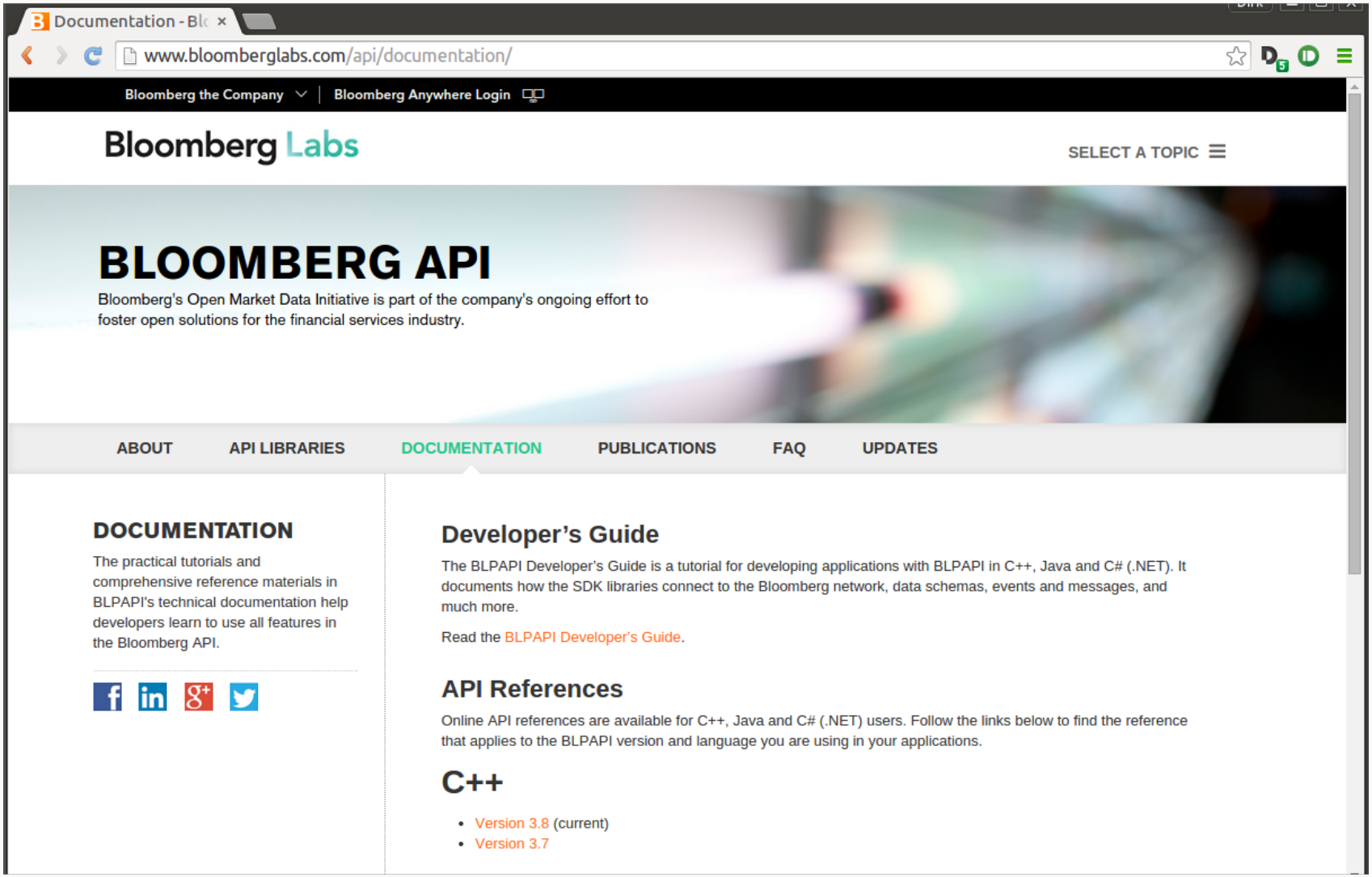
Presentation at the

conference, Vienna, May 20-22, 2004

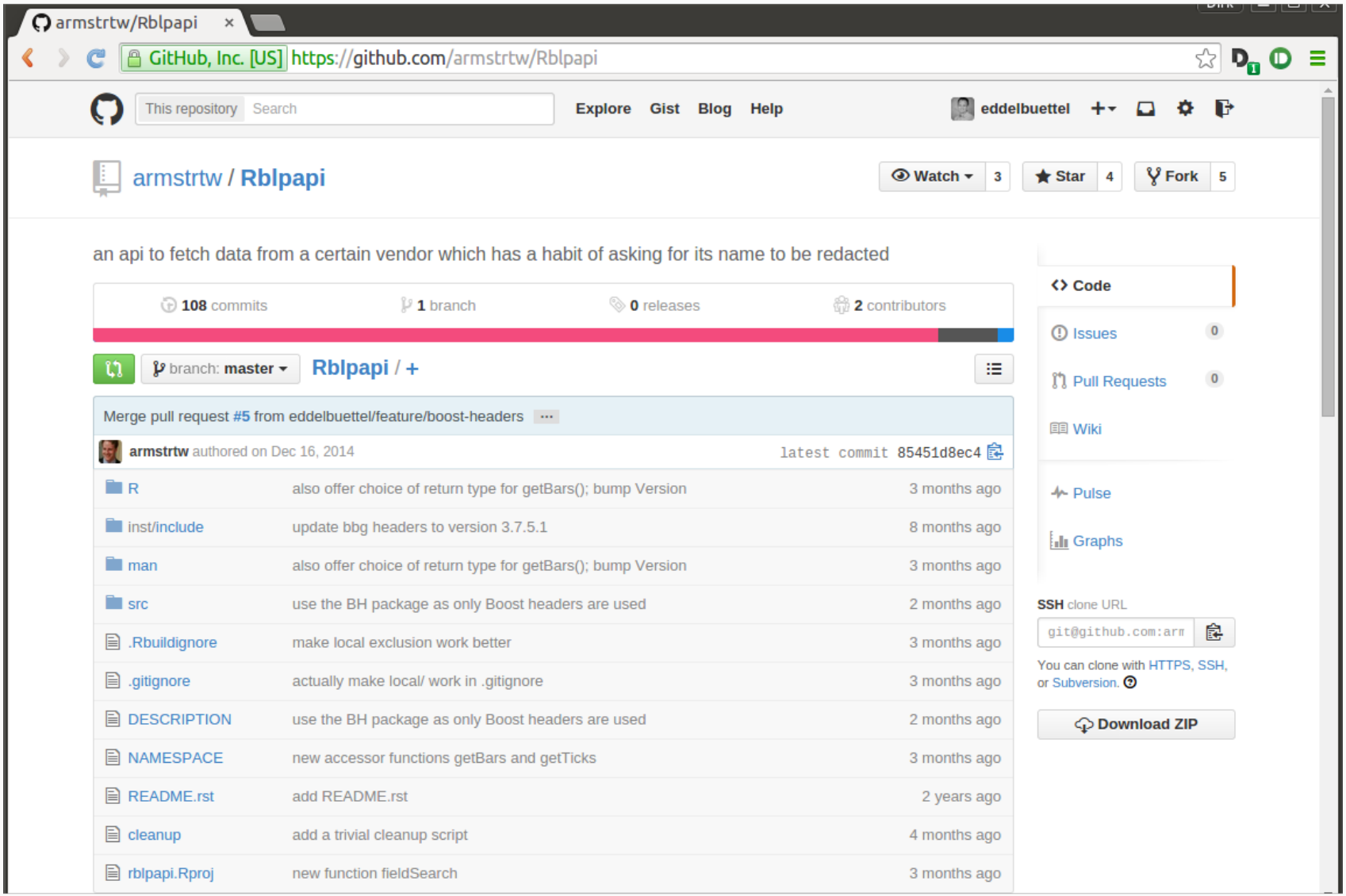
History: Key package using Java (Ana, then John)



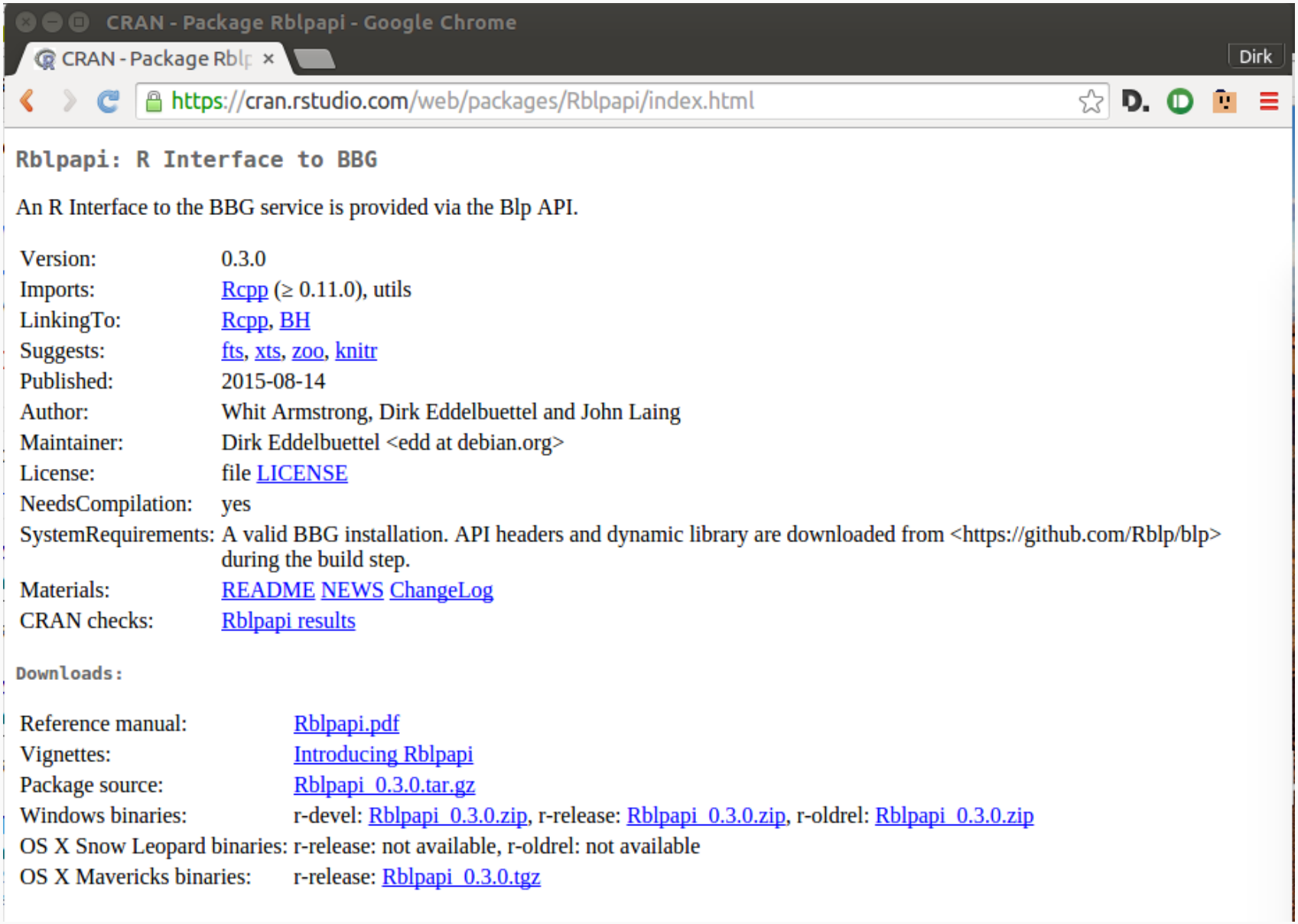
History: But the vendor API keeps improving



Present .. and Future (Whit, Dirk, and John)



Even Better Present and Future



Third Time Lucky: The Rblpapi package

The new rewrite is different:

- Lighter – no longer uses or requires Java
- Simpler – leverages Rcpp
- More flexible – easy to add new functionality with C++

Status: The Rblpapi package

Where we are at now:

- Robust and fast
- Implements most widely-used features
- (Basic) documentation for everything
- Travis CI integration
- On GitHub and in the ghrr repository

NB: And now on CRAN too, see below.

Examples

Core Functions known from other API accessors:

- `bdp(c("ESA Index", "SPY US Equity"), c("PX_LAST", "VOLUME"))`
- `bds("GOOG US Equity", "TOP_20_HOLDERS_PUBLIC_FILINGS")`
- `bdh("SPY US Equity", c("PX_LAST", "VOLUME"), start.date=Sys.Date()-31)`
- `getBars("ESA Index", startTime=ISOdatetime(2015,1,1,0,0,0))`
- `getTicks("ESA Index", "TRADE", Sys.time()-60*60))`
- `fieldSearch("VWAP")`

[Then] Current Status of the Rblpapi package

Things we addressed

- Fixed-dimension retrieval very easy
- Now include shared library with `rpath`-encoded path
- Builds "everywhere" including on Travis CI

[Then] Current Status of the Rblpapi package

Things we [then] need[ed] to address:

- DataFrame class caused trouble, need something new
- Builds on "that other OS" very difficult while (vendor) API library built with VC++
- More features: subscriptions, screens, portfolios...
- Pull requests welcome!

[Then] Summary: The Rblpapi package

Concluding:

- Bloomberg provides a first-rate API and infrastructure
- So the R Community came up with good packages
- Language/OS choice matter: some vendors still "different"
- We prefer Open Source; package may not go onto CRAN
- But we have alternatives in GitHub-hosted repositories

NB: That was then ...

[Now] Summary and State of the Rblpapi Package

What we learned:

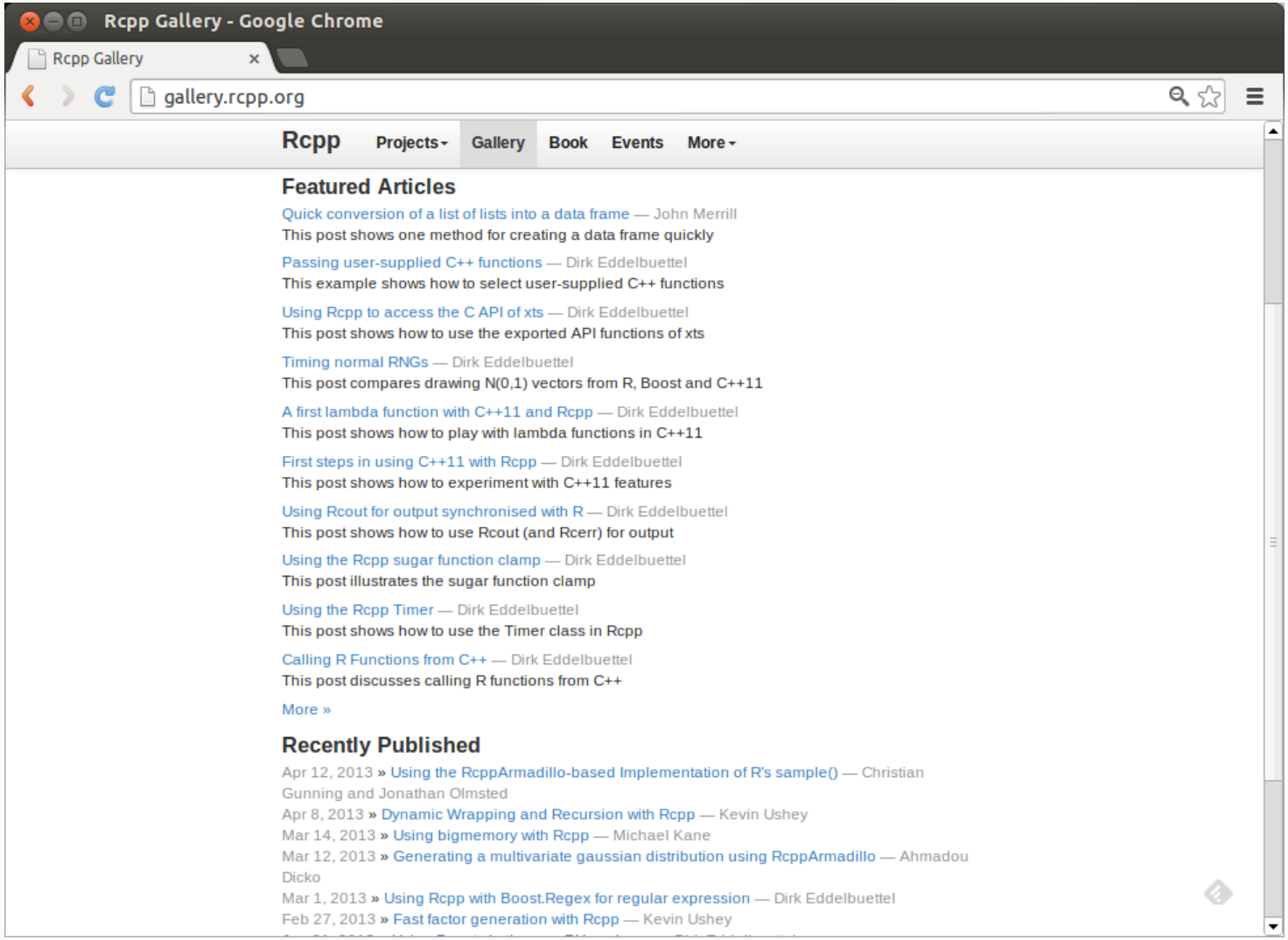
- Powerful APIs are compelling
- Providing working code is key
- Unexpectedly, we got a pull request for Windows support
- With some additional work, this got onto CRAN
- Supporting Linux, OS X and Windows "out of the box"

The End

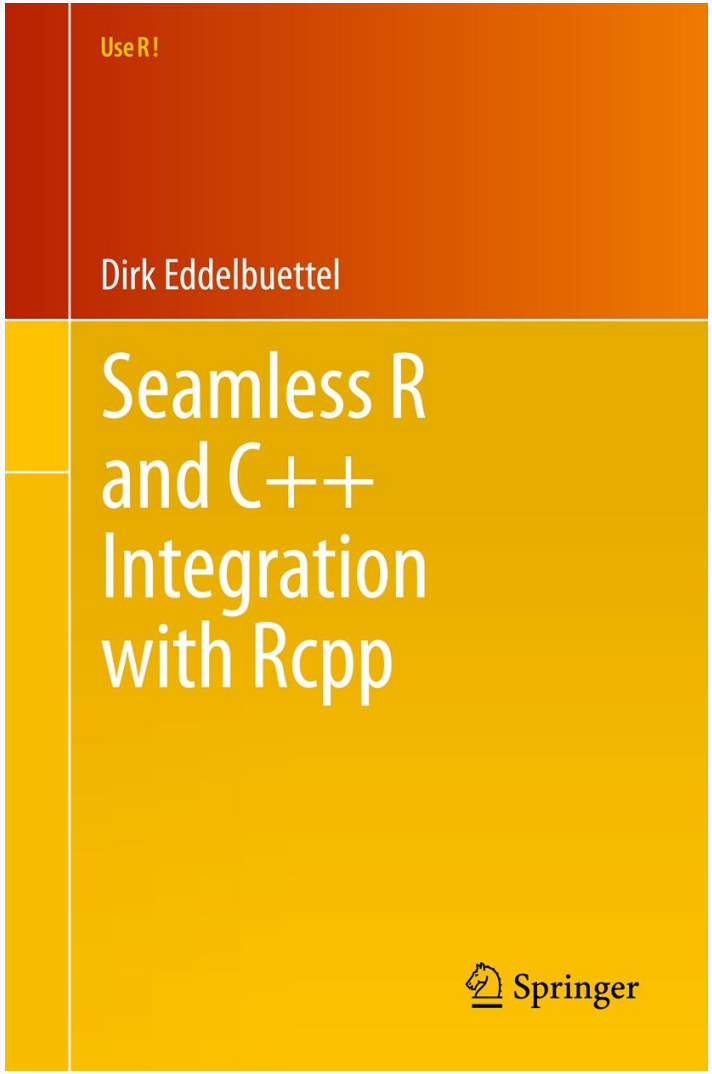
Documentation

- The Rcpp package comes with nine pdf vignettes, and numerous help pages.
- The introductory vignettes are now published (for Rcpp and RcppEigen in *J Stat Software*, for RcppArmadillo in *Comp Stat & Data Anlys*)
- The rcpp-devel list is *the* recommended resource, generally very helpful, and fairly low volume.
- StackOverflow has almost 900 posts too.
- And a number of blog posts introduce/discuss features.

Rcpp Gallery



The Rcpp book



Thank You!

dirk@eddelbuettel.com

http://dirk.eddelbuettel.com/presentations/

Collophon

Made using

- TeXlive 20141024
- Beamer with mtheme
- Pandoc 1.12.4.2
- R 3.2.2
- rmarkdown 0.7
- Emacs 24.4
- Ubuntu 15.04