R, C++ and Rcpp

Dr. Dirk Eddelbuettel
dirk.eddelbuettel@R-Project.org
edd@debian.org
@eddelbuettel

useR! 2014
Invited Talk
UCLA, 2 July 2014

Acknowledgements

Rcpp would not be where it today without

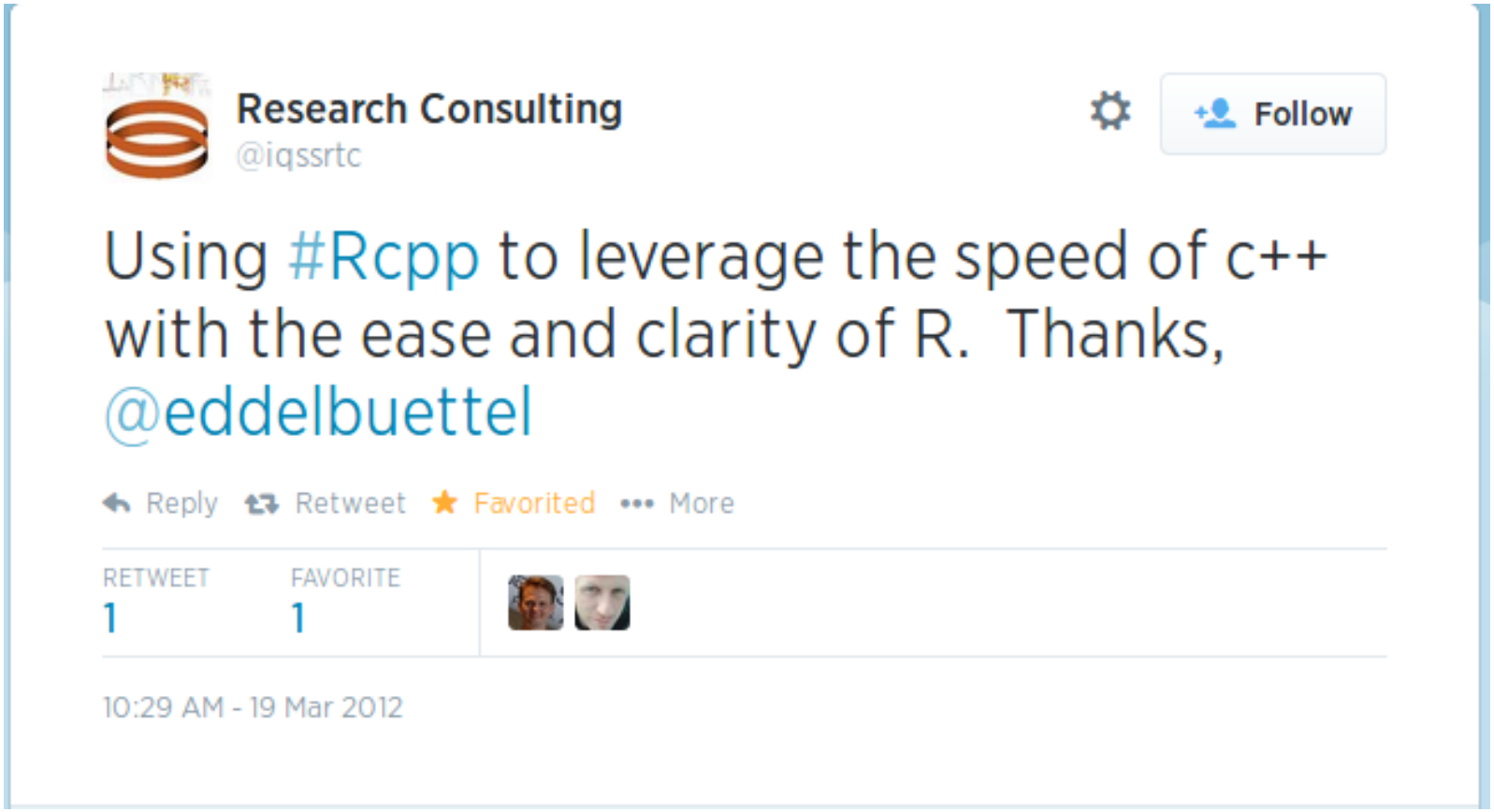
Rcpp Core which is comprised of JJ Allaire, Doug Bates, John Chambers, Dirk Eddelbuettel, Romain Francois and Kevin Ushey

Rcpp contributors listed in the THANKS file

Rcpp and R users in the larger R ecosystem

Outline

A tweet about Rcpp from IQSS / Harvard



R

R excels via

- interactive data analysis
- exploratory work and visualization
- prototyping new methods
- high-level *lingua franca*
- vast array of packages

R and C++

R excels via

- interactive data analysis
- exploratory work and visualization
- prototyping new methods
- high-level *lingua franca*
- vast array of packages

C++ excels via

- native / compiled speed at low overhead
- well-known language with multiple paradigms
- facilities for production deployment of code
- vast array of libraries in C / C++

R and C++

R excels via

- interactive data analysis
- exploratory work and visualization
- prototyping new methods
- high-level *lingua franca*
- vast array of packages

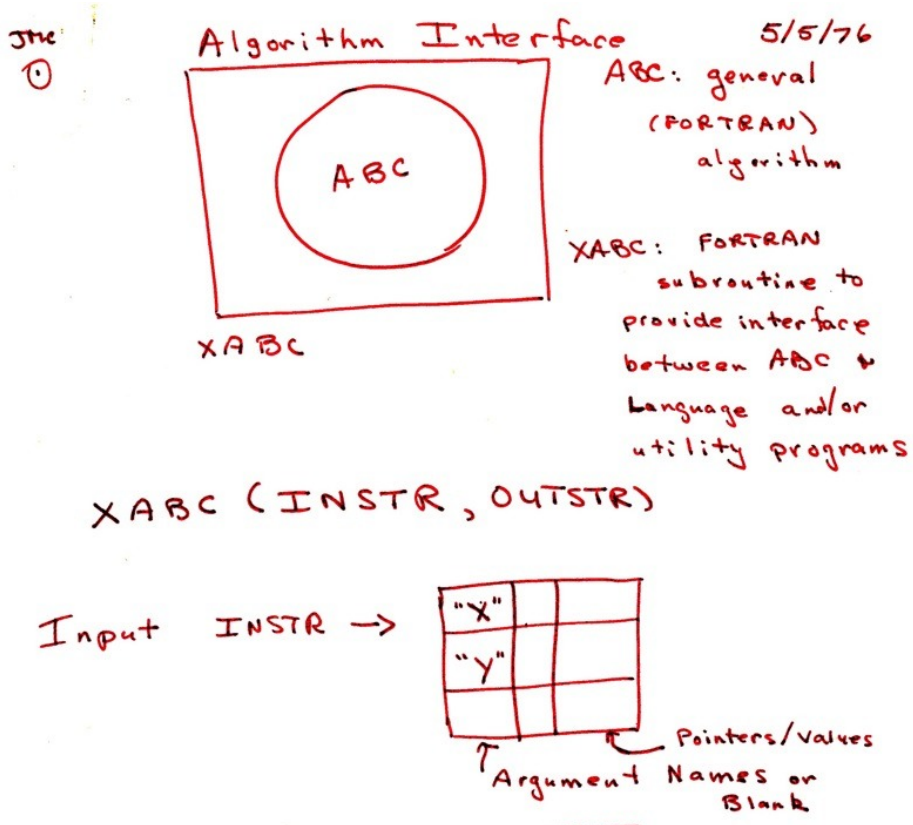
ease and clarity

C++ excels via

- native / compiled speed at low overhead
- well-known language with multiple paradigms
- facilities for production deployment of code
- vast array of libraries in C / C++

speed

R and C++
Interface Vision



Source: John Chambers, personal communication.

R and C++
Interface Vision

- Use trusted numerical libraries (mostly/exclusively written in Fortran)
- Provide environment which statistician could use more easily
- Enable interactive and iterative data exploration
- Pass “objects” seamlessly between both components
- Make it extensible for research into statistical methods
- C.f. John Chambers (2008) regarding “Mission” and “Directive”

Another tweet about Rcpp

Peter Hickey
@PeteHaitch

Follow

Love that my reaction almost every time I rewrite R code in Rcpp is "holy shit that's fast" thanks @eddelbuettel & @romain_francois #rstats

Reply Retweeted Favorited More

RETWEETS 6 FAVORITES 8

9:08 PM - 18 Oct 2013

Outline

CRAN Package cxx (from 2000-2001)

Class definition in header

```
// X.hh

class X {
public:
  X (); ~X ();
};

class Y {
public:
  Y (); ~Y ();
};
```

CRAN Package cxx (from 2000/2001)

Class constructor implementation in source file (could be in header)

```
// X.cc

#include <iostream>
#include "X.hh"

using namespace std;

static Y y;

X::X()  { cout << "constructor X" << endl; }
X::~X() { cout << "destructor X" << endl; }
Y::Y()  { cout << "constructor Y" << endl; }
Y::~Y() { cout << "destructor Y" << endl; }
```

CRAN Package cxx (from 2000/2001)

Glue function with C interface

```
// X_main.cc:

#include "X.hh"

extern "C" {

void X_main () {
  X x;
}

}
```

CRAN Package cxx (from 2000/2001)

R function calling glue function (and note .C() not .Call())

```
cxxtest <- function() {
    invisible(.C("X_main", PACKAGE = "cxx"))
}
```

CRAN Package cxx (from 2000/2001)

It still works

- It still compiles and runs after minor adjustments:
  - C++ side:
    - include wants `iostream` not `iostream.h`
    - Namespaces: add `using namespace std;`
  - R side:
    - `DESCRIPTION` needs Maintainer (added in 0.0-3)
    - `NAMESPACE` needed
- A slightly revised / updated variant is in *Writing R Extensions* – but no corresponding package is maintained

CRAN Package cxx (from 2000/2001)

Lessons

- Pretty sweet to compile 14-year old package
- Only somewhat minor adjustments requirement
- Stable interfaces matter – a lot.
- Stick with standards whenever possible.
- Provides “existence proof” about R and C++.
- Also shows how C++ can provide state via classes with constructor and destructor,

Rcpp Playground: Defined by .Call

R side

```
res <- .Call("nameOfFunction", a, b, c,
             ..., PACKAGE="name")
```

C++ side

```
extern "C"
SEXP nameOfFunction(SEXP a, SEXP b,
                    SEXP c, ...,)
```

Outline

So 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++ just to provide endless discussions about exactly what OO is or should be.

The STL which is a specific template library which is powerful but has its own conventions.

Templated C++ which is mighty powerful; template meta programming unequalled in other languages.

C++1x adds enough to be called a fifth part of this language federation.

Other key aspects

Slightly paraphrasing from talks by Stroustrup and Sutter last fall

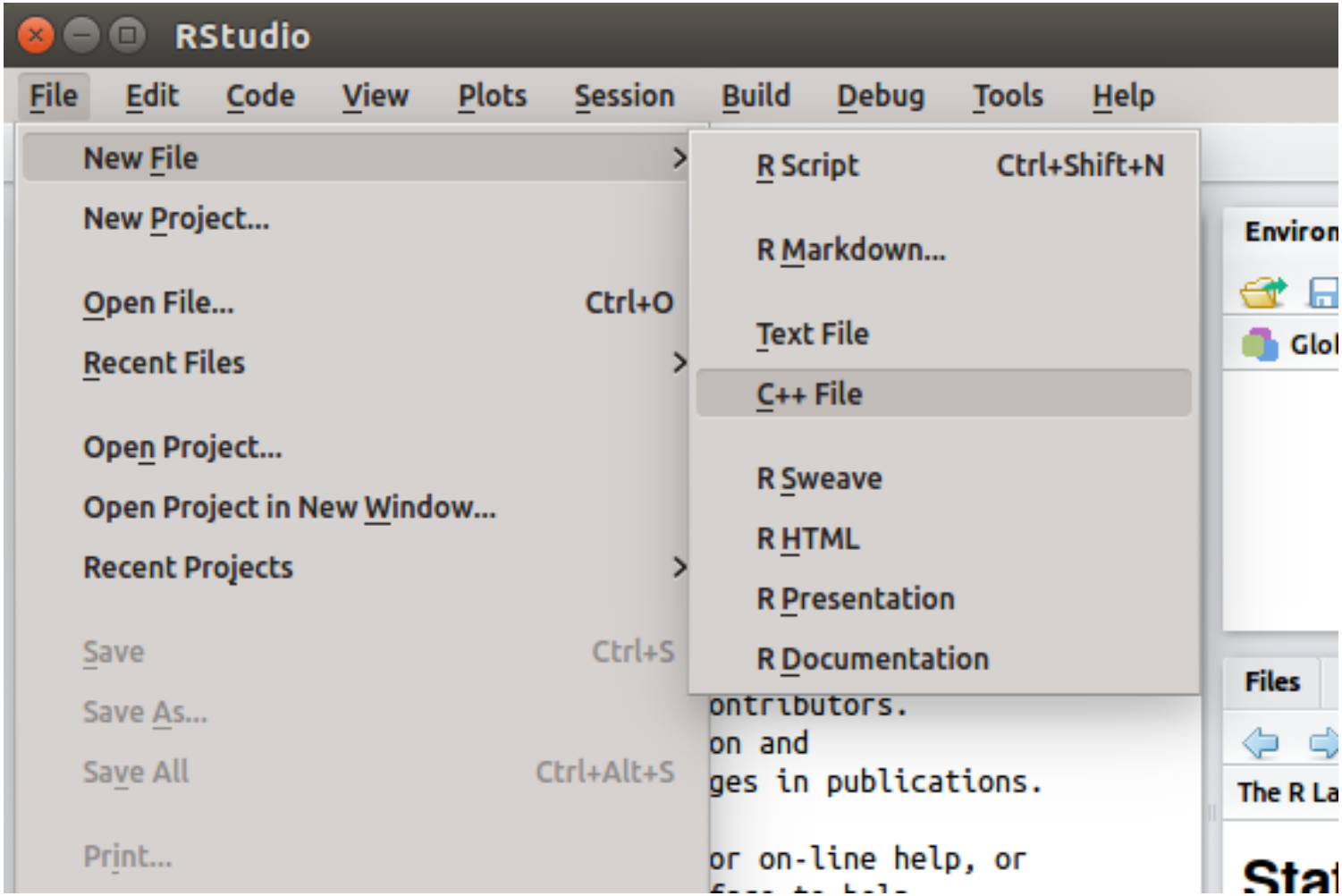
- There should no additional layer (or intermediate language) between C++ and the machine.
- Performance is unsurpassed by other languages.
- "You don't pay for what you don't need": While the language has surely many features, what is not used does not affect performance or resource use.
- C interface / extension major reason for wide and early adoption.
- Very widely used, countless libraries, extensive tools.

Outline

4 Rcpp
- How
- Example: Gibbs
- Example: cumsum
- Example: VAR(1)
- Packages

How do we use Rcpp?

RStudio makes it very easy: Single File



How do we use Rcpp?

RStudio example cont'ed

The following file gets created:

```
#include <Rcpp.h>
using namespace Rcpp;

// Below is a simple example of exporting a C++ function to R.
// You can source this function into an R session using the
// Rcpp::sourceCpp function (or via the Source button on the
// editor toolbar)

// For more on using Rcpp click the Help button on the editor
// toolbar

// [[Rcpp::export]]
int timesTwo(int x) {
   return x * 2;
}
```

How do we use Rcpp?

Rcpp Attributes: evalCpp, cppFunction, sourceCpp

```
## evaluate a C++ expression, retrieve result
evalCpp("2 + 2")

## [1] 4

## create ad-hoc R function 'square'
cppFunction('int square(int x) { return x*x;}')
square(7L)

## [1] 49

## or source an entire file (including R code)
#sourceCpp("code/squareWithRCall.cpp")
```

How do we use Rcpp?

MCMC and Gibbs Samplers

Markov chain Monte Carlo, and the Gibbs sampler in particular are popular to simulate posterior densities.

A set of posts by Darren Wilkinson spawned a cottage industry of comparisons among languages, dialects, variants, ...

We will briefly revisit it here too.

How do we use Rcpp?

Gibbs Sampler Setup

The example by Darren shows a simple MCMC Gibbs sampler of this bivariate density::

$$f(x, y) = kx^2 \exp(-xy^2 - y^2 + 2y - 4x)$$

with conditional distributions

$$\begin{aligned} f(x|y) &\sim \text{Gamma}(3, y^2 + 4) \\ f(y|x) &\sim N\left(\frac{1}{1+x}, \frac{1}{2(1+x)}\right) \end{aligned}$$

i.e. we need repeated RNG draws from both a Gamma and a Gaussian distribution.

How do we use Rcpp?

Gibbs Sampler: R Code

```
## The actual Gibbs Sampler
Rgibbs <- function(N,thin) {
    mat <- matrix(0,ncol=2,nrow=N)
    x <- 0
    y <- 0
    for (i in 1:N) {
        for (j in 1:thin) {
            x <- rgamma(1,3,y*y+4)
            y <- rnorm(1,1/(x+1),1/sqrt(2*(x+1)))
        }
        mat[i,] <- c(x,y)
    }
    mat
}
library(compiler)   ## to byte-compile
RCgibbs <- cmpfun(Rgibbs)
```

Gibbs Sampler: C++ Code

How do we use Rcpp?

```
#include <Rcpp.h>                       // load Rcpp
using namespace Rcpp;                   // shorthand

// [[Rcpp::export]]
NumericMatrix RcppGibbs(int n, int thn) {
    NumericMatrix mat(n, 2);
    double x=0, y=0;
    for (int i=0; i<n; i++) {
        for (int j=0; j<thn; j++) {
            x = R::rgamma(3.0,1.0/(y*y+4));
            y = R::rnorm(1.0/(x+1),1.0/sqrt(2*x+2));
        }
        mat(i,0) = x;
        mat(i,1) = y;
    }
    return mat;                         // Return to R

}
```

Gibbs Sampler: Benchmark

How do we use Rcpp?

```
source("code/gibbs.R")
sourceCpp("code/gibbs.cpp")
library(rbenchmark)
benchmark(Rgibbs(2000,200),
          RCgibbs(2000,200),
          RcppGibbs(2000,200),
          replications=10,
          order="relative")[,c(1,3:4)]
```

```
##                     test elapsed relative
## 3 RcppGibbs(2000, 200)    1.727     1.00
## 2   RCgibbs(2000, 200)   40.911    23.69
## 1    Rgibbs(2000, 200)   44.858    25.98
```

When would we use Rcpp?

Cumulative Sum

A basic looped version:

```
#include <Rcpp.h>
#include <numeric>     // for std::partial_sum
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector cumsum1(NumericVector x){
    // initialize an accumulator variable
    double acc = 0;

    // initialize the result vector
    NumericVector res(x.size());

    for(int i = 0; i < x.size(); i++){
        acc += x[i];
        res[i] = acc;
    }
    return res;

}
```

When would we use Rcpp?

Cumulative Sum

An STL variant:

```
// [[Rcpp::export]]
NumericVector cumsum2(NumericVector x){
    // initialize the result vector
    NumericVector res(x.size());
    std::partial_sum(x.begin(), x.end(),
                     res.begin());
    return res;
}
```

When would we use Rcpp?

Cumulative Sum

Or just Rcpp sugar:

```
// [[Rcpp::export]]
NumericVector cumsum3(NumericVector x){
    return cumsum(x);   // compute + return result vector

}
```

Of course, all results are the same.

```
cppFunction('NumericVector cumsum3(NumericVector x) {
                                    return cumsum(x); }')
x <- 1:10
all.equal(cumsum(x), cumsum3(x))

## [1] TRUE
```

When would we use Rcpp?

Easy speed gain: VAR(1) Simulation

Let's consider a simple possible VAR(1) system of k variables.

For $k = 2$:

$$X_t = X_{t-1} B + E_t$$

where X_t is a row vector of length 2, B is a 2 by 2 matrix and E_t is a row of the error matrix of 2 columns.

When do we use Rcpp?

Easy speedup:: VAR(1) Simulation

In R code, given both the coefficient and error matrices (revealing k and n):

```
rSim <- function(B,E) {
    n <- nrow(E); k <- ncol(E)
    X <- matrix(0, n, k)
    for (r in 2:n) {
        X[r,] = X[r-1, ] %*% B + E[r, ]
    }
    return(X)
}
```

When do we use Rcpp?

Easy speed gain: VAR(1) Simulation

```
cppFunction('
arma::mat cppSim(const arma::mat& B,
                 const arma::mat& E) {
    int n = E.n_rows; int k = E.n_cols;
    arma::mat X = arma::zeros<arma::mat>(n,k);
    for (int r=1; r < n; r++) {
        X.row(r) = X.row(r-1) * B + E.row(r);
    }
    return X;
}', depends="RcppArmadillo")
```

When do we use Rcpp?

Easy speed gain: VAR(1) Simulation

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

// [[Rcpp::export]]
arma::mat cppSim(const arma::mat& B,
                 const arma::mat& E) {
    int n = E.n_rows; int k = E.n_cols;
    arma::mat X = arma::zeros<arma::mat>(n,k);
    for (int r=1; r < n; r++) {
        X.row(r) = X.row(r-1) * B + E.row(r);
    }
    return X;
}
```

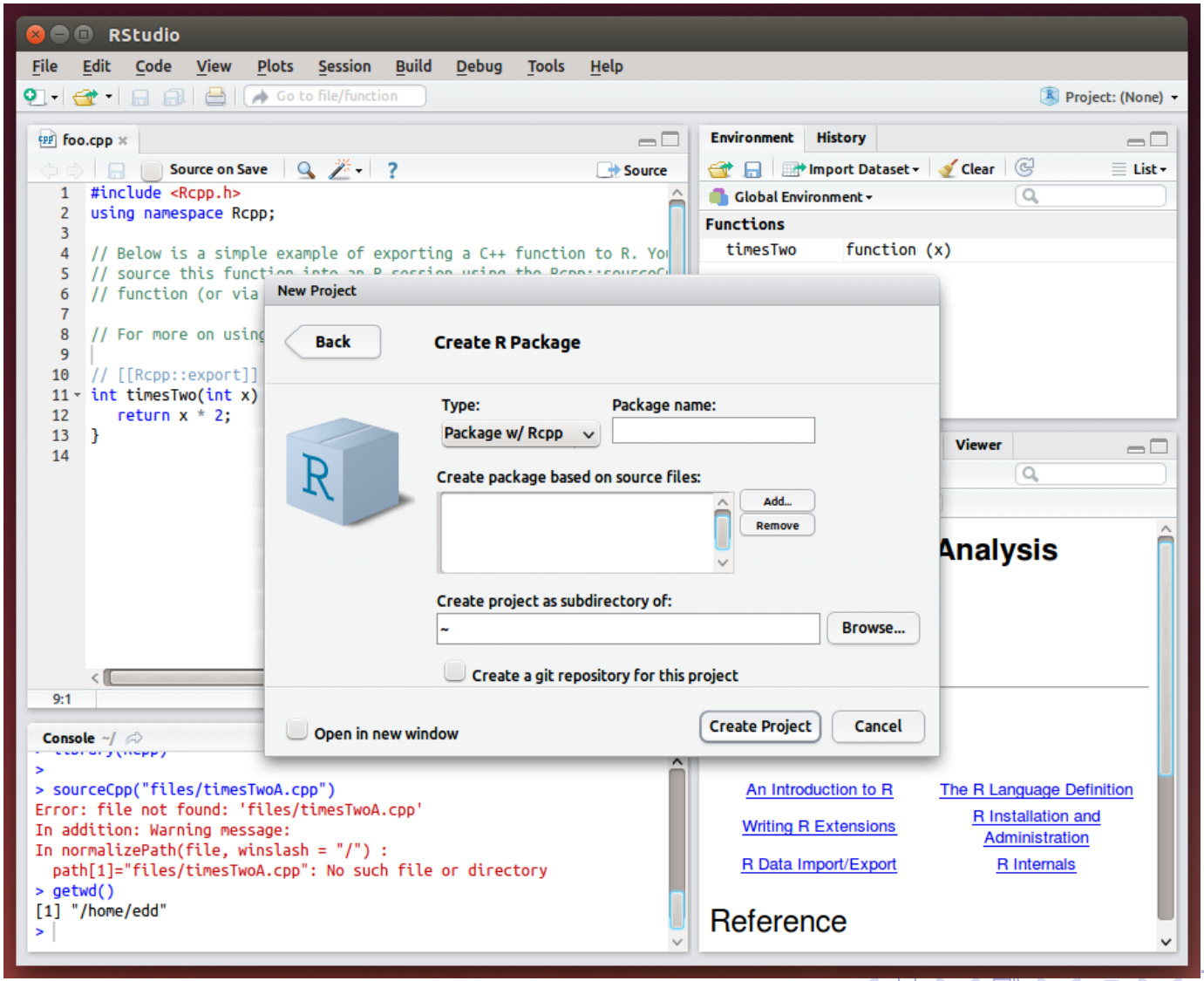
When do we use Rcpp?

Easy speed gain: VAR(1) Simulation

```
a <- matrix(c(0.5,0.1,0.1,0.5),nrow=2)
e <- matrix(rnorm(10000),ncol=2)
all.equal(cppSim(a,e), rSim(a,e))

## [1] TRUE

benchmark(cppSim(a,e), rSim(a,e),
          order="relative")[,1:4]

##           test replications elapsed relative
## 1 cppSim(a, e)          100   0.231     1.00
## 2   rSim(a, e)          100   2.644    11.45
```

How do we use Rcpp?

RStudio makes it very easy (using Rcpp.package.skeleton()



How do we use Rcpp?

Key Features

As we just saw in the small example:

- No build system issues: We just use R
- No operating system dependencies
- No (manual or explicit) memory management
- No complicated C++ code (though we could if we wanted to)
- Easy transition from exploration (“one-liners”) to deployment (“packages”)

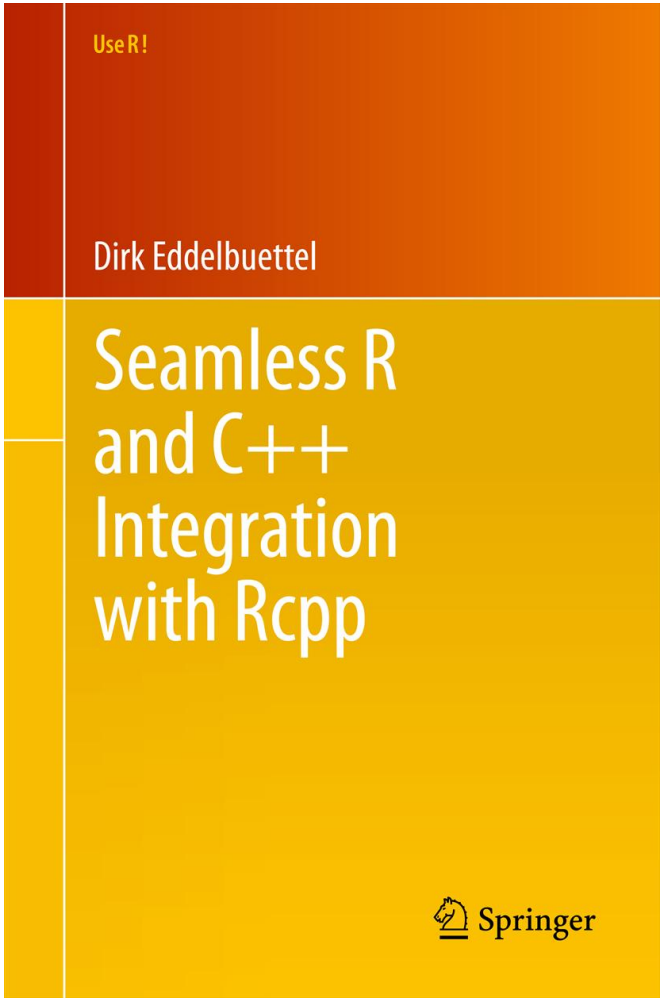
How do we use Rcpp?

Mix and match between packages

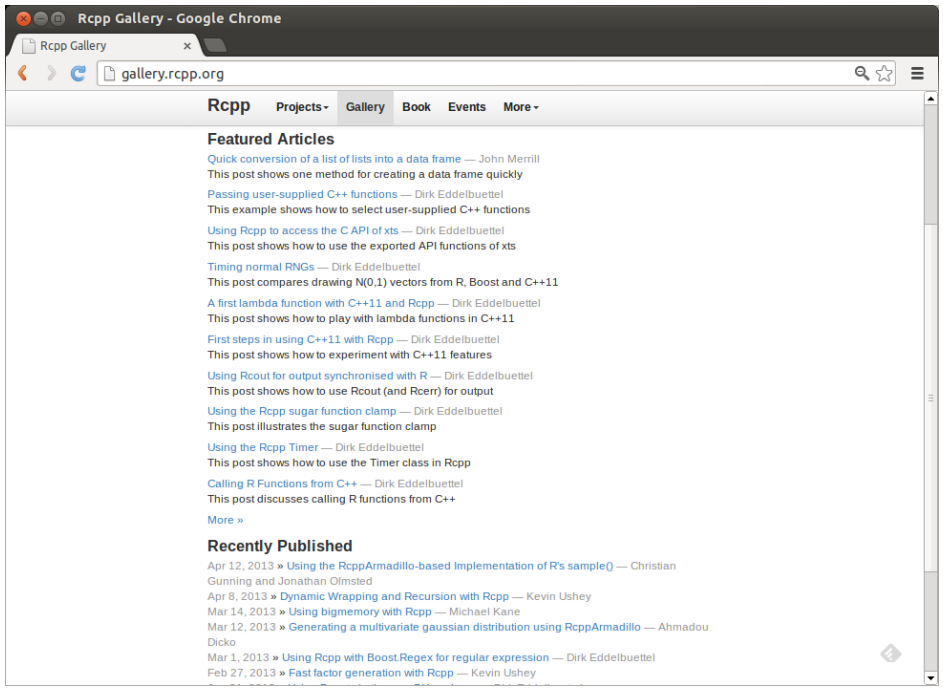
- **plugins** allow us to turn on
  - C++11 as needed without imposing on build system
  - OpenMP to allow for widely-supported and understood parellism
- **depends** allow us to pull in package via
  - headers-only as eg Armadillo, Eigen and Boost
  - actual linking as eg RcppGSL
- **registration** for easier linking on the user-side

Outline

Documentation

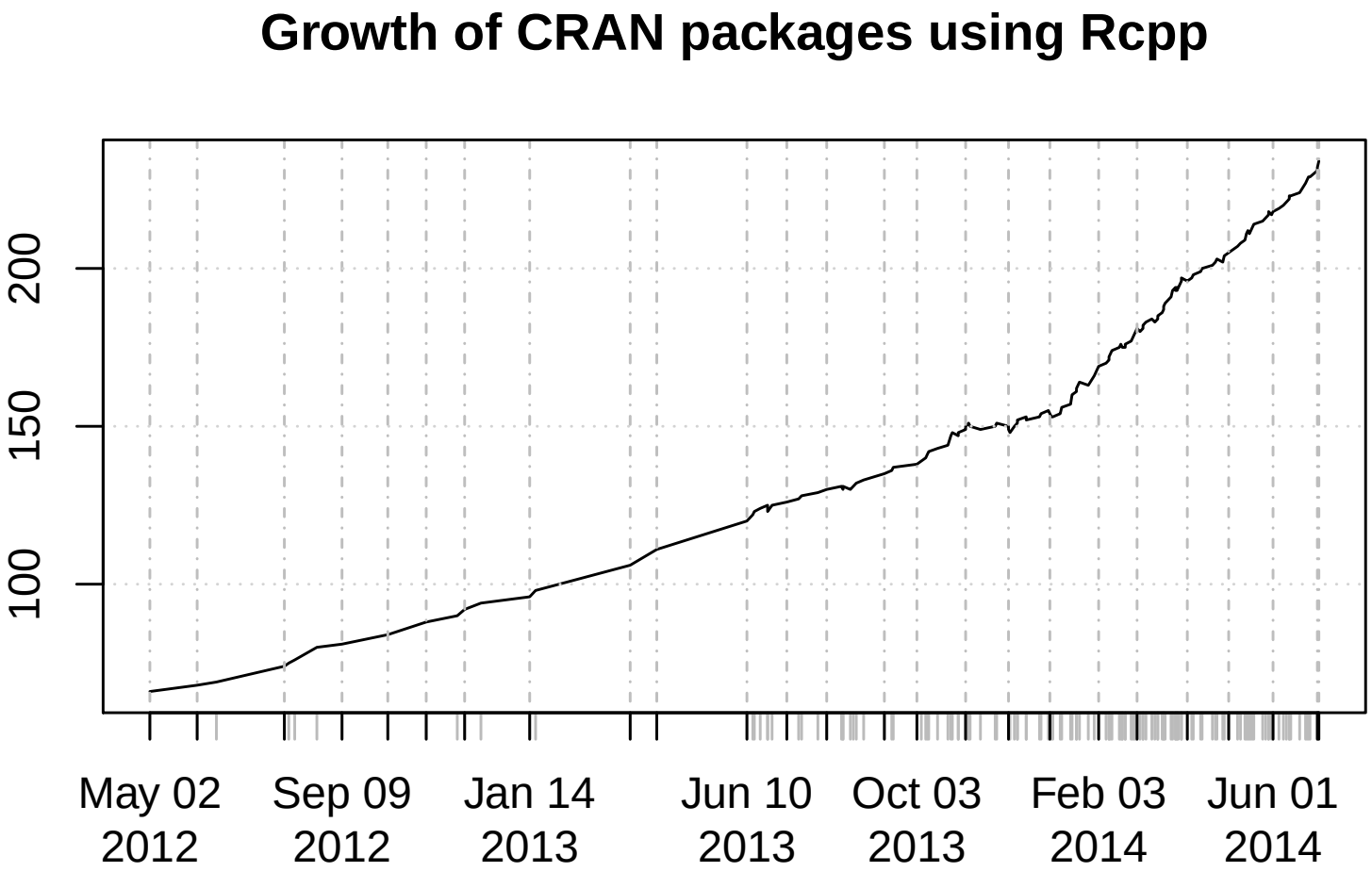


See `gallery.rcpp.org`

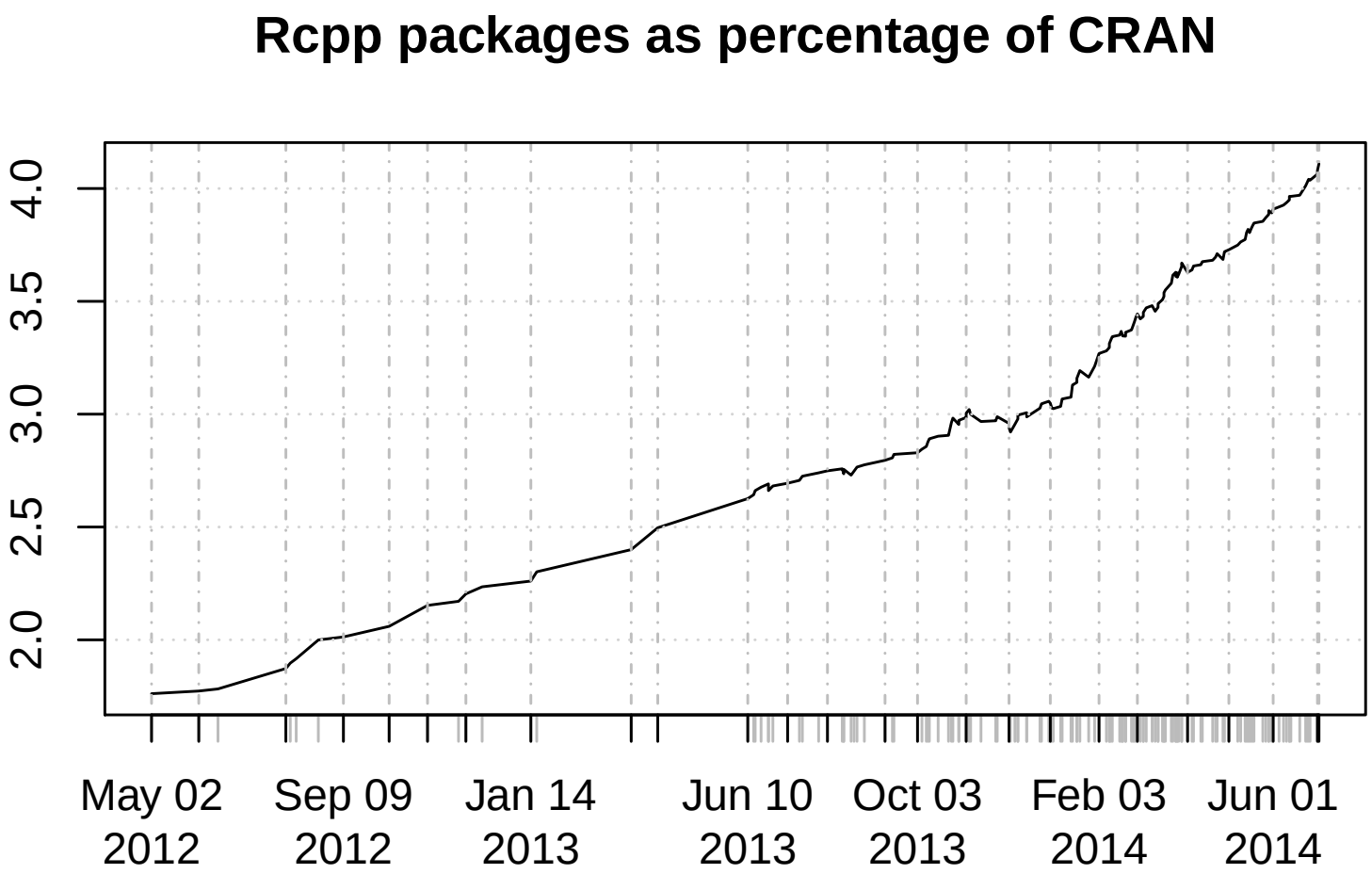


Outline

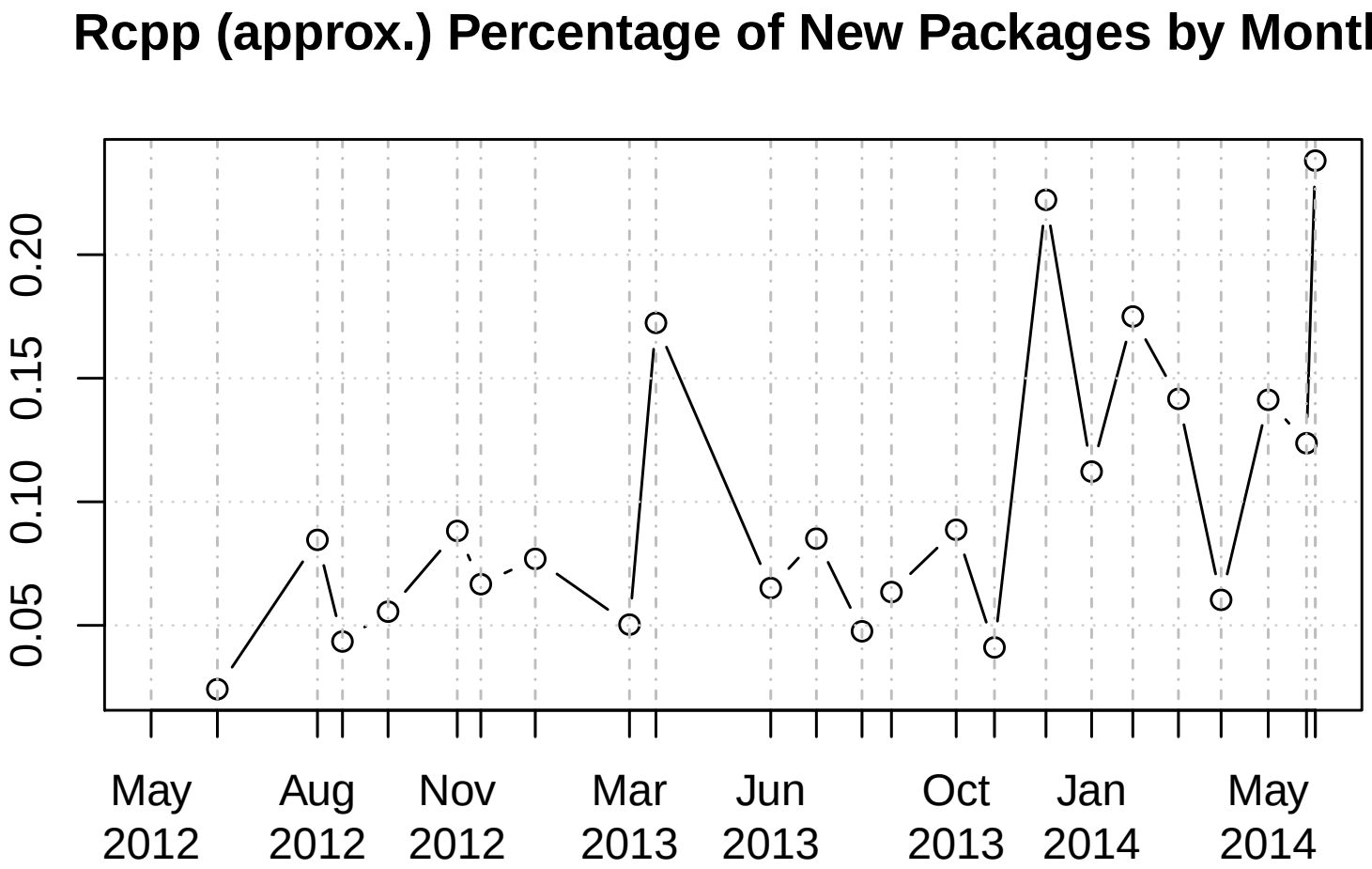
Rcpp on CRAN: Now at 234 packages



Rcpp on CRAN: Or just over 4 per cent



Rcpp on CRAN: 10+ per cent of new pkgs



Growing pains?

From 0.6.0 in late 2008 to 0.11.0 in early 2014

- six major releases from 0.6.0 to 0.11.0
- changed build / deployment system in last release
- disruptive change which took effort to progagate
- we now use C bindings and registered functions
- hope that such breakage may be a thing of the past

Growing pains?

At 250+ user packages across CRAN and BioConductor, we simply cannot make breaking changes.

We test against all direct dependencies.

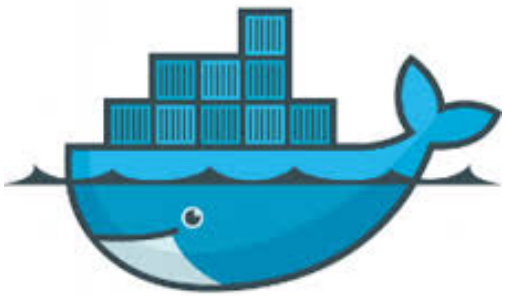
CRAN tests even further.

Some hope that *in the not-so-distant future* we all may have something better...

Outline

7 Docker
- Intro
- Setup
- Commands
- Shells
- Dockerfiles
- Example

Docker

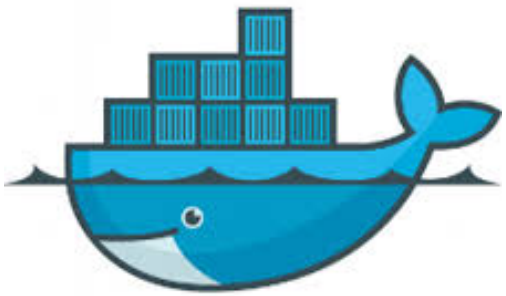


What is Docker?

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments.

Text from `http://docker.com` as of June 24, 2014.

Docker



Ok, seriously, what *is* Docker?

Docker is a very lightweight abstraction using recent Linux kernel features which lets us to run code in **cheap** (to launch) and **easy** (to build) units: **containers**.

We can share containers across OSs and into 'the cloud'.

It changes how we build and test R (and R packages).

Getting started with Docker

Installation

- Most Unix variants have detailed (short) instructions on Docker website. Should just work.
- Or: `sudo apt-get install docker.io`

  (requires Ubuntu 14.04 or Debian testing)
- On Windows or OS X: Use **boot2docker** which installs an appliance for you (containing docker, git, virtualbox, ...).

  Tested on Windows at work. Appears to Just works too.

Getting started with Docker

First steps: Getting images

After installation, run for example

```
docker pull ubuntu
```

to pull a set of pre-built initial images.

Many such image sets are now being defined.

I will describe one for R.

Getting started with Docker

First steps: Listing images

This one call gets us

```
edd@max:~$ docker images
REPOSITORY  TAG       IMAGE ID       CREATED       VIRTUAL SIZE
ubuntu      10.04     3db9c44f4520   9 weeks ago   183 MB
ubuntu      lucid     3db9c44f4520   9 weeks ago   183 MB
ubuntu      12.10     6006e6343fad   3 weeks ago   172.2 MB
ubuntu      quantal   6006e6343fad   3 weeks ago   172.2 MB
ubuntu      13.10     d2099a5ba6c5   3 weeks ago   180.2 MB
ubuntu      saucy     d2099a5ba6c5   3 weeks ago   180.2 MB
ubuntu      14.04     5cf8fd909c6c   3 weeks ago   274.3 MB
ubuntu      latest    5cf8fd909c6c   3 weeks ago   274.3 MB
ubuntu      trusty    5cf8fd909c6c   3 weeks ago   274.3 MB
ubuntu      raring    7656cbf56a8c   3 weeks ago   169.4 MB
ubuntu      13.04     7656cbf56a8c   3 weeks ago   169.4 MB
ubuntu      12.04     cc0067db4f11   3 weeks ago   210.1 MB
ubuntu      precise   cc0067db4f11   3 weeks ago   210.1 MB

edd@max:~$
```

Getting started with Docker

First steps: Running images

Now we can execute commands in all these variants:

```
edd@max:~$ docker run -t ubuntu:12.04 head -1 /etc/motd
Welcome to Ubuntu 12.04.4 LTS (GNU/Linux 3.2.0-58-generic
x86_64)
edd@max:~$ docker run -t ubuntu:12.10 head -1 /etc/motd
Welcome to Ubuntu 12.10 (GNU/Linux 2.6.32-42-server x86_64)
edd@max:~$ docker run -t ubuntu:13.04 head -1 /etc/motd
Welcome to Ubuntu 13.04 (GNU/Linux 2.6.32-46-server x86_64)
edd@max:~$ docker run -t ubuntu:13.10 grep VERSION_ID
/etc/os-release
VERSION_ID="13.10"
edd@max:~$ docker run -t ubuntu:14.04 grep VERSION_ID
/etc/os-release
VERSION_ID="14.04"

edd@max:~$
```

Getting started with Docker

First steps: Running images interactively

Now we can execute commands in all these variants:

```
edd@max:~$ docker run -t -i ubuntu:12.04 /bin/bash
root@fe2fe9a795ff:/$ whoami
root
root@fe2fe9a795ff:/$ exit
exit

edd@max:~$
```

Getting started with Docker

First steps: Running images interactively

We can mount directories for use by the Docker image

```
edd@max:~$ ls -1 | wc
     34       35      380
edd@max:~$ docker run -v `pwd`:/mydir \
                      -t -i ubuntu:12.04 /bin/bash
root@fe2fe9a795ff:/$ ls -1 /mydir
     34       35      380
root@fe2fe9a795ff:/$ exit
exit

edd@max:~$
```

Getting started with Docker

Using a Dockerfile

Dockerfiles are 'recipes' (à la Travis.yml) which modify / create images. Below is 'add-r' which just adds an R installation:

```
## start with the Debian testing
FROM debian:testing
MAINTAINER Dirk Eddelbuettel edd@debian.org

## Remain current
RUN apt-get update -qq
RUN apt-get dist-upgrade -y

RUN apt-get install -y --no-install-recommends \
            r-base r-base-dev r-recommended littler
RUN ln -s /usr/share/doc/littler/examples/install.r \
            /usr/local/bin/install.r

## use via:

## cat Dockerfile | docker build -t debian:testing-w-R -
```

Getting started with Docker

Dockerfile

I have started to create first sets of (partially nested) Dockerfiles creating images

- add-r with basic R as a binary package
- add-r-devel adding R-devel built from source
- add-r-devel-san adding R-devel built from source and enabling Address Sanitizer checks
- add-r-devel-ubsan adding R-devel built from source and enabling Undefined Behavior Sanitizer checks

which are downloadable from `hub.docker.com` under my handle `eddelbuettel`.

Getting started with Docker

Test with R

```
$ docker run -v `pwd`:/mytmp -t b524252a3462  \
     R CMD check --no-manual --no-build-vignettes \
     /mytmp/sanitizers_1.0.tar.gz
* using log directory '//sanitizers.Rcheck'
* using R version 3.1.0 (2014-04-10)
* using platform: i486-pc-linux-gnu (32-bit)
* using session charset: ASCII
* using option '--no-build-vignettes'
* checking for file 'sanitizers/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'sanitizers' version '1.0'
[...]
```

which tests (fine) under the release version.

Getting started with Docker

Test with R devel

```
$ docker run -v `pwd`:/mytmp -t b524252a3462  \
     Rdevel CMD check --no-manual --no-build-vignettes \
     /mytmp/sanitizers_1.0.tar.gz
edd@don:~/Dropbox/src/san-ubsan$ docker run -v `pwd`:/mytmp -t b524252
* using log directory '//sanitizers.Rcheck'
* using R Under development (unstable) (2014-06-20 r65987)
* using platform: i686-pc-linux-gnu (32-bit)
* using session charset: ASCII
* using option '--no-build-vignettes'
* checking for file 'sanitizers/DESCRIPTION' ... OK
[...]
```

which tests under the R-development version (which has been enabled for Sanitizer checks) and ...

Getting started with Docker

Test with R devel

```
[...]
* checking tests ...
  Running 'simple.R'
 ERROR
Running the tests in 'tests/simple.R' failed.
Last 13 lines of output:
    Freed heap region:       fd
    Stack left redzone:      f1
    Stack mid redzone:       f2
    Stack right redzone:     f3
    Stack partial redzone:   f4
    Stack after return:      f5
    Stack use after scope:   f8
    Global redzone:          f9
    Global init order:       f6
    Poisoned by user:        f7
    Contiguous container OOB:fc
    ASan internal:           fe
  ==258==ABORTING
$
```

Getting started with Docker

Enormous Scope and Potential

- Docker provides `hub.docker.com`
- Which integrates with Github
- So we can upload/create/share/modify Dockerfiles
- Triggering rebuilds at Docker for updated images
- Which one can download, deploy as is, or modify
- Deploy locally or on via (growing) list of cloud providers incl. Google Compute Engine

in order to run R, or R-devel, or R-devel with Sanitizers, or R-devel with Profiling, or alternative R implementations ...

Getting started with Docker

Upside for the R Community

- *Linux as a Service*: Getting identical, reliable, replicable test environments
- which we can contain different compilers: gcc-4.8, gcc-4.9, clang-3.4, clang-3.5, icc, ...
- as well as across configurations: enable profiling, sanitizers, valgrind, ...
- possibly using `debian-r.debian.net` and its 6000+ Debian packages spanning CRAN and BioC
- without requiring anyone to install (or learn) Linux
- as the installations are lightweight and disposable
- yet can be used across Linux, Windows, OS X, ... and in the (Google etc) cloud

Conclusion

Rcpp

- mature after 5 1/2 years
- fairly widely used
- supported, tested
- still growing and improving
- *speed of C++ and ease and clarity of R*

Docker

- just had 1.0 release
- lots of momentum and interest
- perfect fit for our (batch, scripted) tests of R
- intend to work on some of this, help welcome