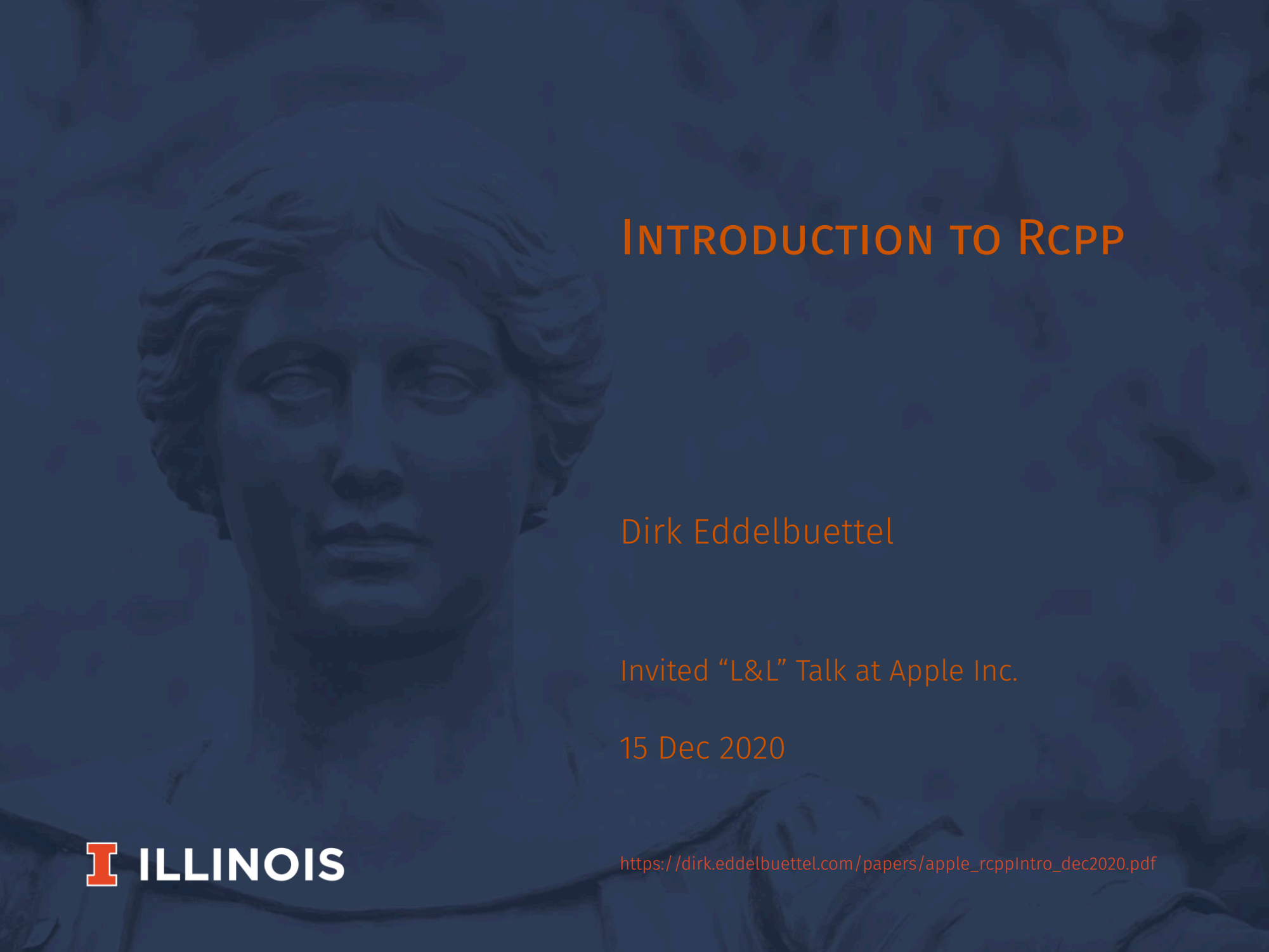


Introduction to Rcpp

Dirk Eddelbuettel

Invited “L&L” Talk at Apple Inc.

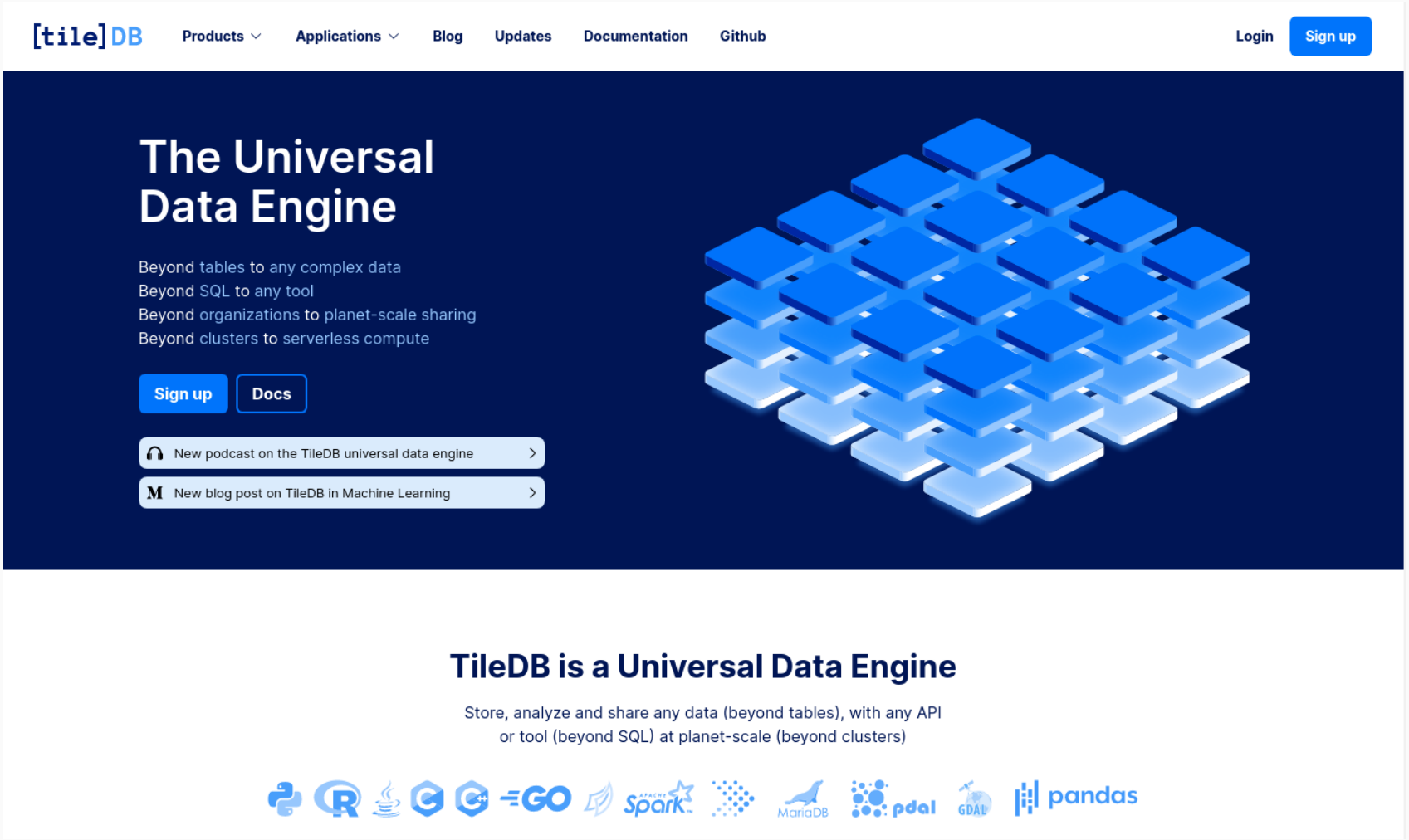
15 Dec 2020

ILLINOIS

https://dirk.eddelbuettel.com/papers/apple_rcppIntro_dec2020.pdf

Who am I ?

My day job


[tile]DB
Products
Applications
Blog
Updates
Documentation
Github
Login
Sign up
The Universal Data Engine
Beyond tables to any complex data
Beyond SQL to any tool
Beyond organizations to planet-scale sharing
Beyond clusters to serverless compute
Sign up
Docs
New podcast on the TileDB universal data engine
New blog post on TileDB in Machine Learning
TileDB is a Universal Data Engine
Store, analyze and share any data (beyond tables), with any API
or tool (beyond SQL) at planet-scale (beyond clusters)
GO
spark
MariaDB
pdal
GDAL
pandas

My other jobs

Academic

- (Adjunct) Clinical Professor, University of Illinois
  - teaching a Data Science Programming Methods class

Open Source

- Debian developer
  - since 1995, currently maintaining about 175 packages
- R package author
  - since 2002, author or maintainer of over 60 CRAN packages
  - R Foundation Board Member
- Rocker Project co-founder
  - Docker for R, including official ‘r-base’ image

Introduction to Rcpp

Very Broad Outline

Overview

- Why ?
- How ?

Introduction: Why?

So why Rcpp?

Three key reasons

- Speed, Performance, ...
- Do things you could not do before
- Easy to extend R this way

Simple Example

R Version of 'is this number odd or even'

```
isOdd_r <- function(num = 10L) {
   result = (num %% 2L == 1L)
   return(result)
}
isOdd_r(42L)
```

```
## [1] FALSE
```

Simple Example (cont.)

C++ Version of 'is this number odd or even'

```cpp
bool isOdd_cpp(int num = 10) {
    bool result = (num % 2 == 1);
    return result;
}
```

Free-standing code, not yet executable...

Simple Example (cont.)

Rcpp Version of 'is this number odd or even'

```
Rcpp::cppFunction("
bool isOdd_cpp(int num = 10) {
    bool result = (num % 2 == 1);
    return result;
}")
isOdd_cpp(42L)
```

```
## [1] FALSE
```

Simple Example (cont.)

In R

```
##
isOdd_r <- function(n=10L) {
   res = (n %% 2L == 1L)
   return(res)
}
isOdd_r(42L)
```

```
## [1] FALSE
```

In C++ via Rcpp

```
Rcpp::cppFunction("
bool isOdd_cpp(int n=10) {
   bool res = (n % 2 == 1);
   return res;
}")
isOdd_cpp(42L)
```

```
## [1] FALSE
```

Second Example: VAR(1)

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.

Second Example: VAR(1)

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

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

Second Example: VAR(1)

```
cppFunction('arma::mat cppSim(arma::mat B, arma::mat E) {
    int m = E.n_rows; int n = E.n_cols;
    arma::mat X(m,n);
    X.row(0) = arma::zeros<arma::mat>(1,n);
    for (int r=1; r<m; r++) {
        X.row(r) = X.row(r-1) * B + E.row(r);
    }
    return X; }', depends="RcppArmadillo")
a <- matrix(c(0.5,0.1,0.1,0.5),nrow=2)
e <- matrix(rnorm(10000),ncol=2)
benchmark(cppSim(a,e), rSim(a,e), order="relative")[,1:4]
```

```
##           test replications elapsed relative
## 1 cppSim(a, e)          100   0.009    1.000
## 2   rSim(a, e)          100   0.696   77.333
```

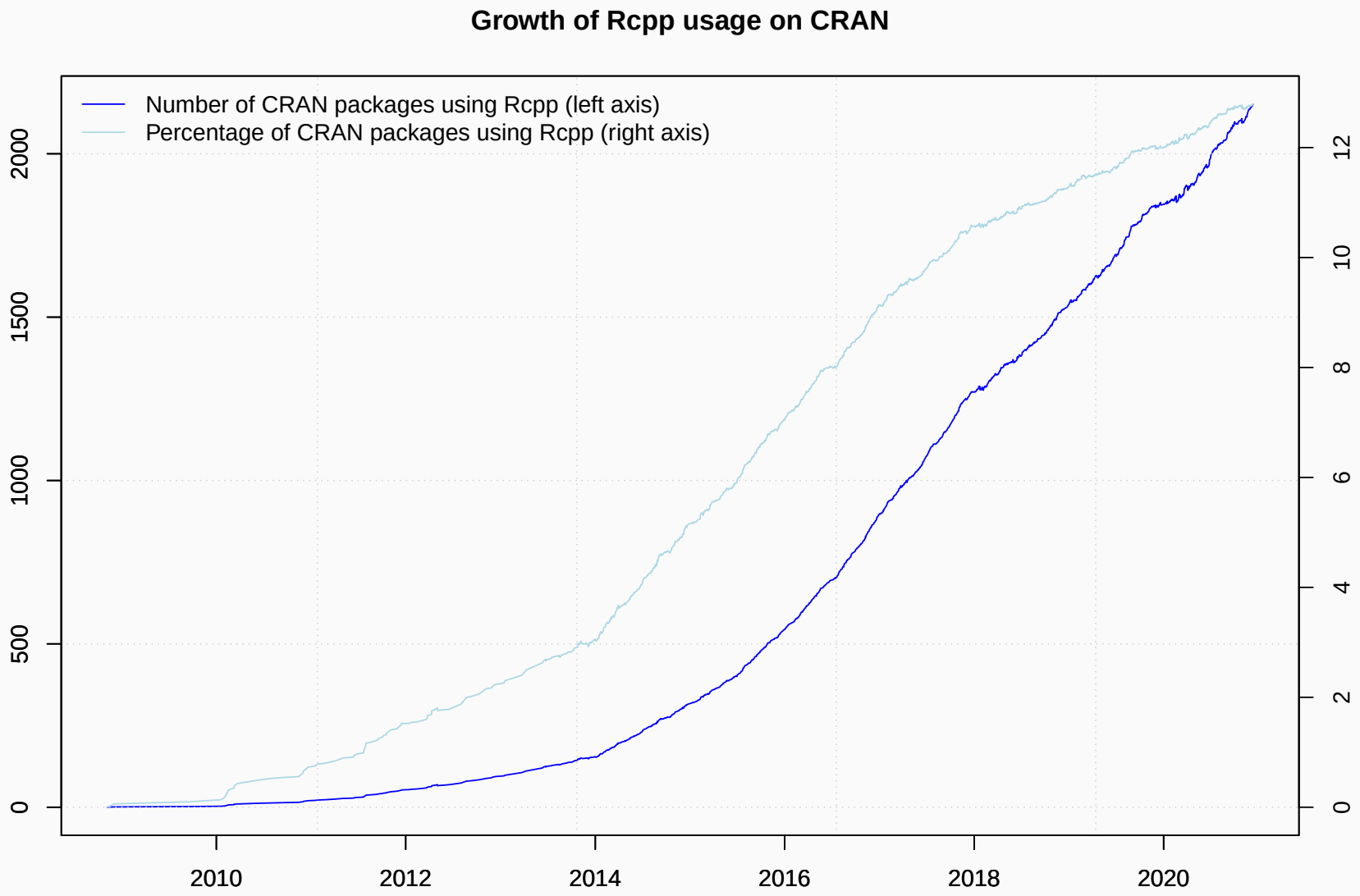
So when do we use Rcpp?

New things: Easy access to C/C++ libraries

- Sometimes speed is not the only reason
- C & C++ provide numerous libraries + APIs we may want to use
- Easy to provide access to as Rcpp eases data transfer

An Aside

Growth



Data current as of December 13, 2020.

Users on Core Repositories

Rcpp is currently used by

- 2151 CRAN packages
- 207 BioConductor packages
- an unknown (but “large”) number of GitHub projects

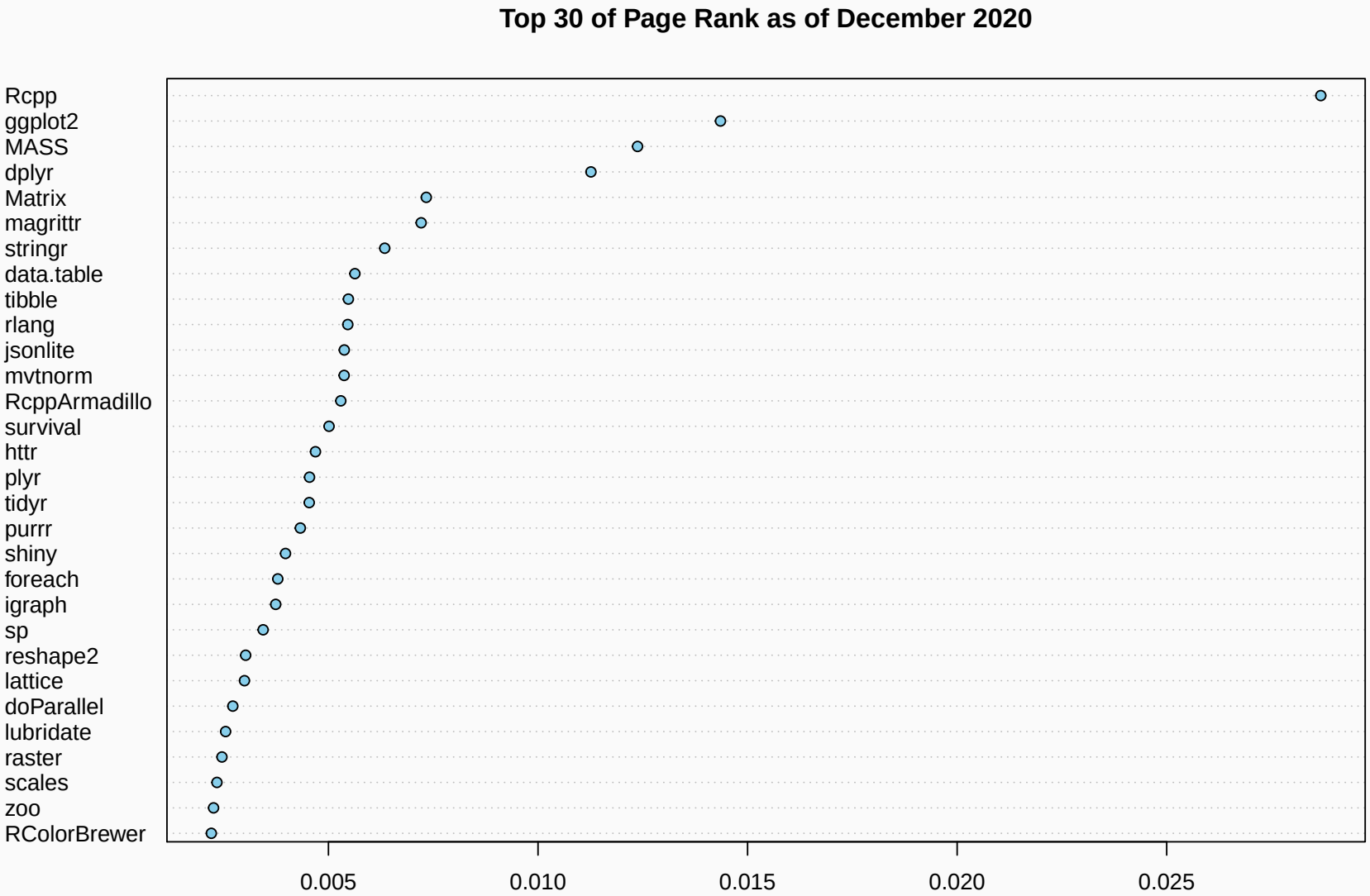
Pagerank

```
suppressMessages(library(utils))
library(pagerank)   # cf github.com/andrie/pagerank

cran <- "http://cloud.r-project.org"
pr <- compute_pagerank(cran)
round(100*pr[1:5], 3)
```

```
##    Rcpp ggplot2    MASS   dplyr  Matrix
##   2.868   1.435   1.237   1.126   0.733
```

Pagerank


Top 30 of Page Rank as of December 2020
Rcpp
ggplot2
MASS
dplyr
Matrix
magrittr
stringr
data.table
tibble
rlang
jsonlite
mvtnorm
RcppArmadillo
survival
httr
plyr
tidyr
purrr
shiny
foreach
igraph
sp
reshape2
lattice
doParallel
lubridate
raster
scales
zoo
RColorBrewer
0.005
0.010
0.015
0.020
0.025

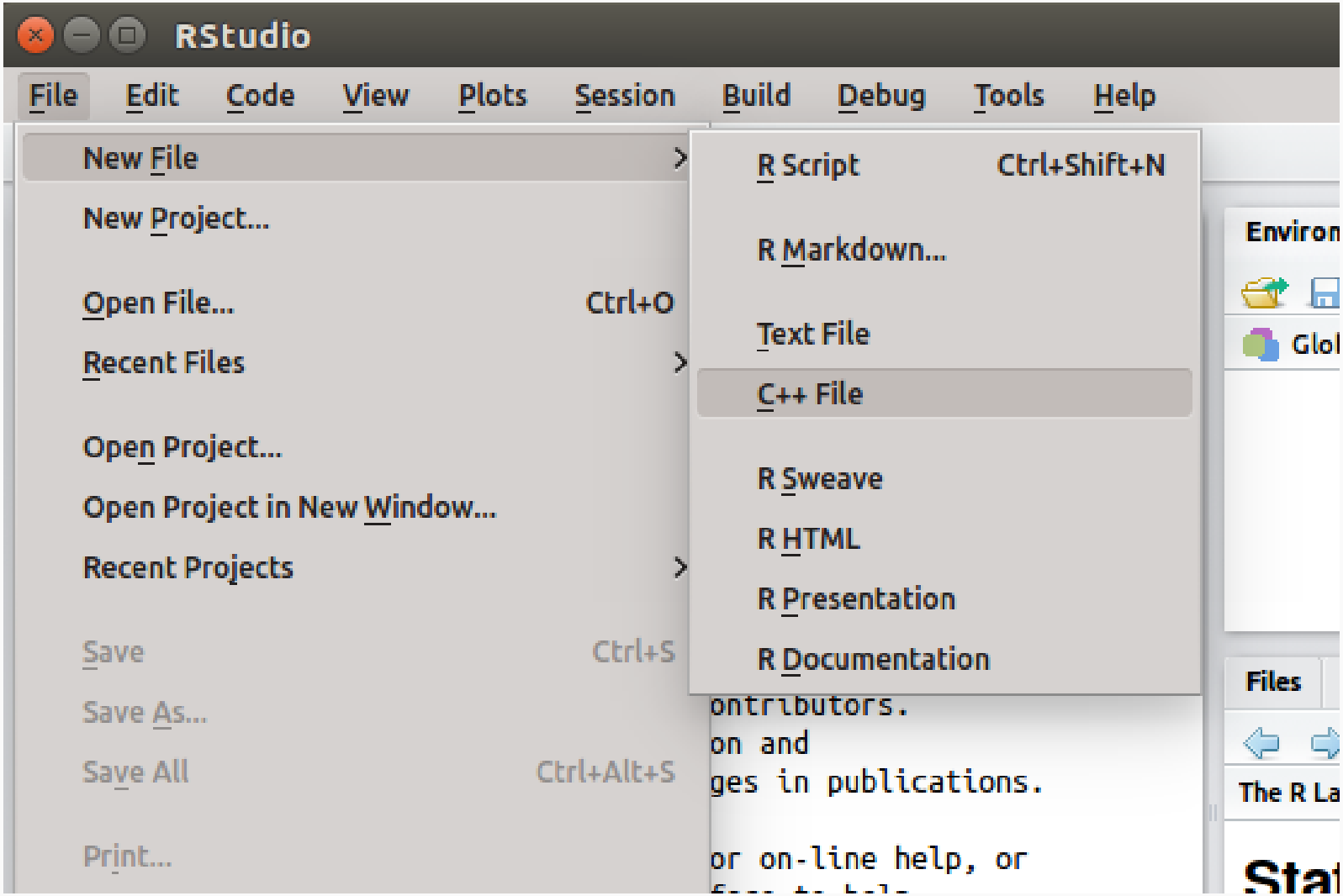
Percentage of Compiled Packages

```
db <- tools::CRAN_package_db()   # added in R 3.4.0
db <- db[!duplicated(db[,1]),]   # rows: nb of pkgs,
nTot <- nrow(db)                 # cols: diff attributes
nRcpp <- length(tools::dependsOnPkgs("Rcpp",recursive=FALSE,
                                     installed=db))
nCompiled <- table(db[, "NeedsCompilation"])["yes"]
propRcpp <- nRcpp / nCompiled * 100
data.frame(tot=nTot, totRcpp = nRcpp,
           totCompiled = nCompiled,
           RcppPctOfCompiled = propRcpp)
```

```
##     tot totRcpp totCompiled RcppPctOfCompiled
## 1 16794    2151        4060           52.9803
```

Introduction: How?

Jumping Right In: Via RStudio



A First Example: Cont'ed

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

// This 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). ...

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

// You can include R code blocks in C++ files processed with sourceCpp
// (useful for testing and development). The R code will be automatically
// run after the compilation.

/*** R
timesTwo(42)
*/
```

A First Example: Cont'ed

So what just happened?

- We defined a simple C++ function
- It operates on a numeric vector argument
- We ask Rcpp to 'source it' for us
- Behind the scenes Rcpp creates a wrapper
- Rcpp then compiles, links, and loads the wrapper
- The function is available in R under its C++ name

Another Example: Focus on Speed

Consider a function defined as

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

An Introductory Example: Simple R Implementation

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
```

An Introductory Example: Timing R Implementation

Timing:

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

```
##    test replications elapsed relative
## 1 f(10)          100   0.010      1.0
## 2 f(15)          100   0.097      9.7
## 3 f(20)          100   1.177    117.7
```

An Introductory Example: C++ Implementation

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

deployed as

```
Rcpp::cppFunction('int g(int n) {
   if (n < 2) return(n);
   return(g(n-1) + g(n-2)); }')
## Using it on first 11 arguments
sapply(0:10, g)
```

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

An Introductory Example: Comparing timing

Timing:

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

```
##    test replications elapsed relative
## 1 f(20)          100   1.142      571
## 2 g(20)          100   0.002        1
```

A nice gain of a few orders of magnitude.

Some Background

R Type mapping

Standard R types (integer, numeric, list, function, ... and compound objects) are mapped to corresponding C++ types using extensive template meta-programming – it just works:

```
library(Rcpp)
cppFunction("NumericVector la(NumericVector x){
  return log(abs(x));
}")
la(seq(-5, 5, by=2))
```

Also note: vectorized C++! `log(abs())` on vectors as R would.

STL Type mapping

Use of `std::vector<double>` and STL algorithms:

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

inline double f(double x) { return ::log(::fabs(x)); }

// [[Rcpp::export]]
std::vector<double> logabs2(std::vector<double> x) {
  std::transform(x.begin(), x.end(), x.begin(),  f);
  return x;
}
```

Not vectorized but `std::transform()` 'sweeps' `f()` across.

STL Type mapping

Used via

```
library(Rcpp)
sourceCpp("code/logabs2.cpp")
logabs2(seq(-5, 5, by=2))
```

Type mapping is seamless

Simple outer product of a col. vector (using RcppArmadillo):

```
library(Rcpp)
cppFunction("arma::mat v(arma::colvec a) {
              return a*a.t();}",
            depends="RcppArmadillo")
v(1:3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    4    6
## [3,]    3    6    9
```

Uses implicit conversion via `as<>` and `wrap` – cf vignette Rcpp-extending.

C++11: lambdas, auto, and much more

We can simplify the `log(abs(...))` example further:

```
#include <Rcpp.h>
// [[Rcpp::plugins(cpp11)]]

using namespace Rcpp;

// [[Rcpp::export]]
std::vector<double> logabs3(std::vector<double> x) {
   std::transform(x.begin(), x.end(), x.begin(),
                  [](double x) {
                    return ::log(::fabs(x));
                  } );
   return x;
}
```

How to: Main Usage Patterns

Basic Usage: evalCpp()

`evalCpp()` evaluates a single C++ expression. Includes and dependencies can be declared.

This allows us to quickly check C++ constructs.

```
library(Rcpp)
evalCpp("2 + 2")      # simple test
```

```
## [1] 4
```

```
evalCpp("std::numeric_limits<double>::max()")
```

```
## [1] 1.79769e+308
```

Basic Usage: cppFunction()

`cppFunction()` creates, compiles and links a C++ file, and creates an R function to access it.

```
cppFunction("
    int exampleCpp11() {
        auto x = 10;
        return x;
}", plugins=c("cpp11"))
exampleCpp11()  # same identifier as C++ function
```

Basic Usage: sourceCpp()

`sourceCpp()` is the actual workhorse behind `evalCpp()` and `cppFunction()`. It is described in more detail in the package vignette Rcpp-attributes.

`sourceCpp()` builds on and extends `cxxfunction()` from package inline, but provides even more ease-of-use, control and helpers – freeing us from boilerplate scaffolding.

A key feature are the plugins and dependency options: other packages can provide a plugin to supply require compile-time parameters (cf RcppArmadillo, RcppEigen, RcppGSL).

Basic Usage: Packages

Package are *the* standard unit of R code organization.

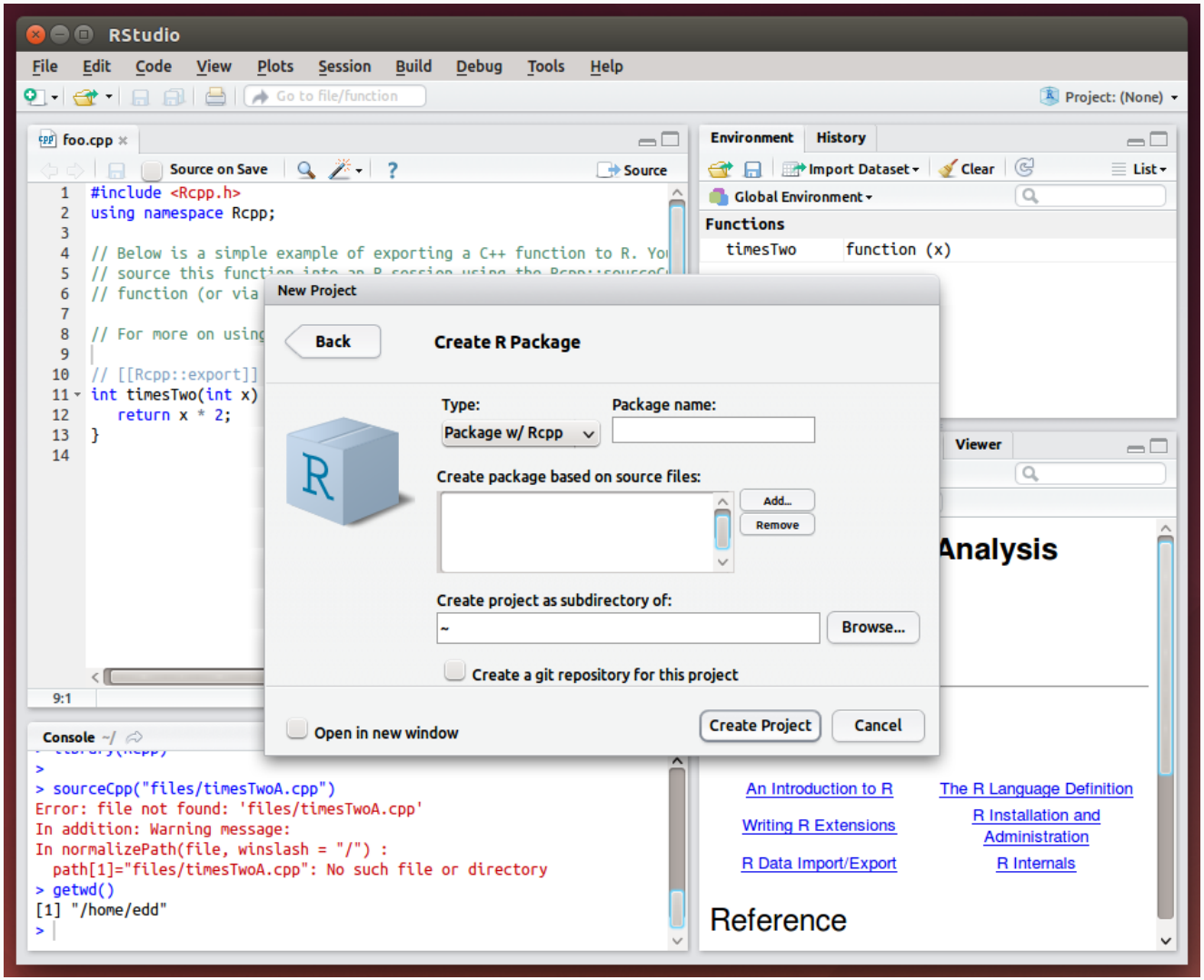
Creating packages with Rcpp is easy; an empty one to work from can be created by `Rcpp.package.skeleton()`

The vignette Rcpp-packages has fuller details.

As of December 2020, there are 2151 CRAN and 207 BioConductor packages which use Rcpp all offering working, tested, and reviewed examples.

Packages and Rcpp

Best way to organize R code with Rcpp is via a package:



Packages and Rcpp

`Rcpp.package.skeleton()` and its derivatives as e.g. `RcppArmadillo.package.skeleton()` create working packages.

```
// another simple example: outer product of a vector,
// returning a matrix
//
// [[Rcpp::export]]
arma::mat rcpparma_outerproduct(const arma::colvec & x) {
    arma::mat m = x * x.t();
    return m;
}

// and the inner product returns a scalar
//
// [[Rcpp::export]]
double rcpparma_innerproduct(const arma::colvec & x) {
    double v = arma::as_scalar(x.t() * x);
    return v;
}
```

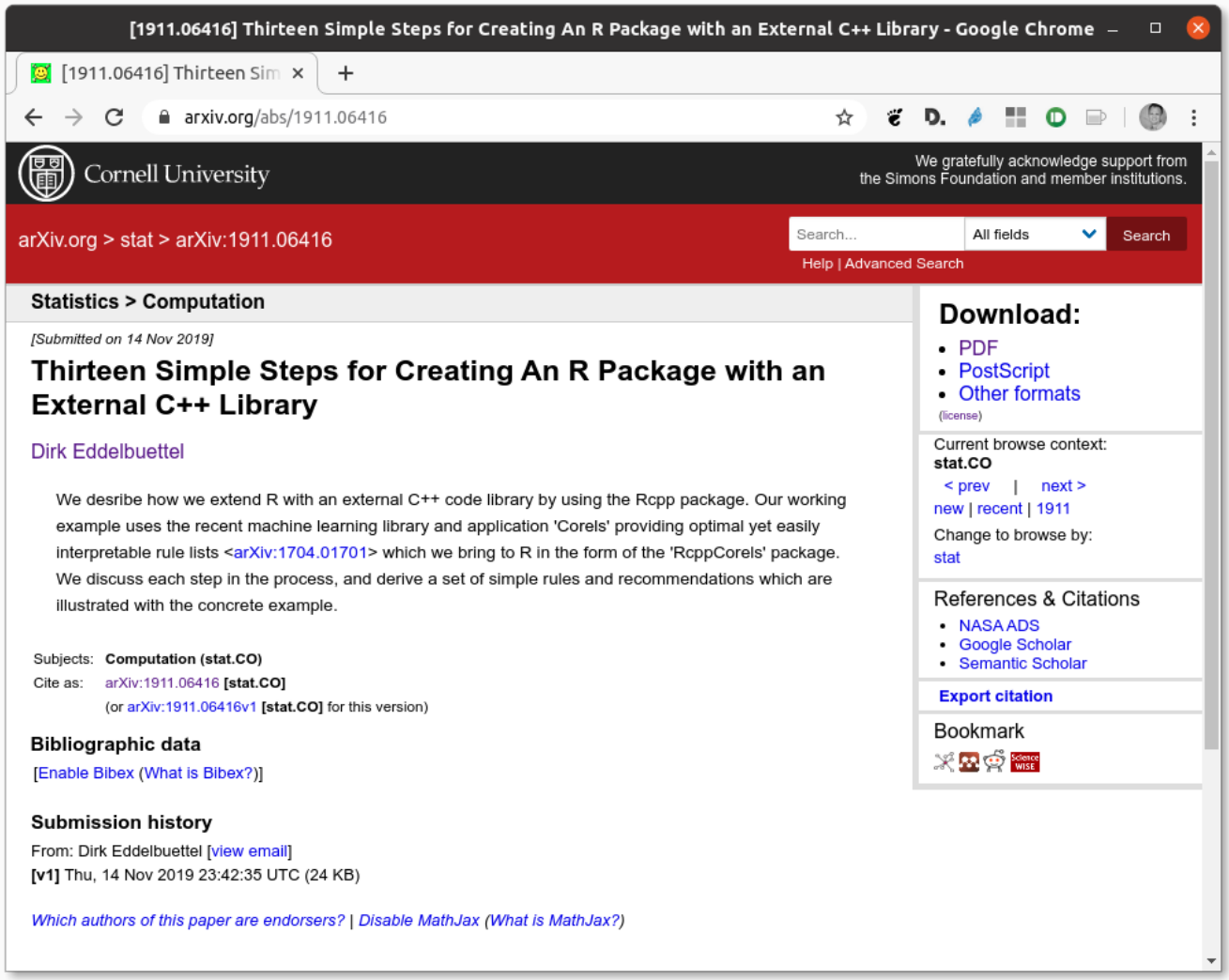
Packages and Rcpp

Two (or three) ways to link to external libraries

- *Full copies:* Do what mlpack does and embed a full copy; larger build time, harder to update, self-contained
- *With linking of libraries:* Do what *e.g.* RcppGSL does and use hooks in the package startup to store compiler and linker flags which are passed to environment variables
- *With C++ template headers only:* Do what RcppArmadillo and other do and just point to the headers

More details in extra vignettes.

New vignette and recent paper



Big Picture

Should you use Rcpp? Or not?

Choice is yours

- Code generation helps remove tedium
- Interfaces are shorter / simpler / more R like
    - recall the `is_odd` function earlier
- Plain C API to R is of course *perfectly fine*
- But IMHO requires more work
    - more manual steps for type conversion
    - additional required memory protection
    - all of which is error prone

Compare

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

SEXP convolve2(SEXP a, SEXP b) {
    int na, nb, nab;
    double *xa, *xb, *xab;
    SEXP ab;

    a = PROTECT(coerceVector(a, REALSXP));
    b = PROTECT(coerceVector(b, REALSXP));
    na = length(a);
    nb = length(b);
    nab = na + nb - 1;
    ab = PROTECT(allocVector(REALSXP, nab));
    xa = REAL(a);
    xb = REAL(b);
    xab = REAL(ab);
    for(int i = 0; i < nab; i++)
        xab[i] = 0.0;
    for(int i = 0; i < na; i++)
        for(int j = 0; j < nb; j++)
            xab[i + j] += xa[i] * xb[j];
    UNPROTECT(3);
    return ab;
}
```

```c
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::NumericVector convolve2cpp(Rcpp::NumericVector a,
                                 Rcpp::NumericVector b) {
    int na = a.length(),
        nb = b.length();
    Rcpp::NumericVector ab(na + nb - 1);
    for (int i = 0; i < na; i++)
        for (int j = 0; j < nb; j++)
            ab[i + j] += a[i] * b[j];
    return(ab);
}
```

You always have a choice between the code (from Section 5.10.1 of *Writing R Extensions*) on the left, or the equivalent Rcpp code on the right.

Machine Learning

Overview

Among the 2150+ Rcpp + CRAN packages, several wrap ML libraries.

Here are three:

- RcppShark based on Shark (but archived in March 2018)
- dlib based on DLib
- mlpack brings us MLPACK

MLPACK

High-level:

- Written by Ryan Curtin et al, Georgia Tech
- Uses Armadillo, and like Armadillo, “feels right”
- Qiang Kou created ‘RcppMLPACK v1’, it is on CRAN

MLPACK ‘v2/v3’

High-level:

- A few of us were trying to update RcppMLPACK to ‘v2’
- Instead of embedding, an external library is used
- This makes deployment a little tricker on Windows and macOS
- We are still waiting on macOS installation of libraries

MLPACK 'v2/v3'

High-level:

- A few of us were trying to update RcppMLPACK to 'v2'
- Instead of embedding, an external library is used
- This makes deployment a little tricker on Windows and macOS
- We are still waiting on macOS installation of libraries

Now following GSoC 2020:

- Integrates new wrappers from the MLPACK side
- At last on CRAN for just about a week (!!)
- Nice effort making R a formal interface language for MLPACK
- GH repo Yashwants19/RcppMLPACK tracked the progress

List of Algorithms:

- Collaborative filtering (with many decomposition techniques)
- Decision stumps (one-level decision trees)
- Density estimation trees
- Euclidean minimum spanning tree calculation
- Gaussian mixture models
- Hidden Markov models
- Kernel Principal Components Analysis (optionally with sampling)
- k-Means clustering (with several accelerated algorithms)
- Least-angle regression (LARS/LASSO)
- Linear regression (simple least-squares)
- Local coordinate coding
- Locality-sensitive hashing for approximate nearest neighbor search
- Logistic regression
- Max-kernel search
- Naive Bayes classifier
- Nearest neighbor search with dual-tree algorithms
- Neighborhood components analysis
- Non-negative matrix factorization
- Perceptrons
- Principal components analysis (PCA)
- RADICAL (independent components analysis)
- Range search with dual-tree algorithms
- Rank-approximate nearest neighbor search
- Sparse coding with dictionary learning

(old) RcppMLPACK: K-Means Example

```
#include "RcppMLPACK.h"

using namespace mlpack::kmeans;
using namespace Rcpp;

// [[Rcpp::depends(RcppMLPACK)]]

// [[Rcpp::export]]
List cppKmeans(const arma::mat& data, const int& clusters) {

    arma::Col<size_t> assignments;
    KMeans<> k;     // Initialize with the default arguments.
    k.Cluster(data, clusters, assignments);

    return List::create(Named("clusters") = clusters,
                        Named("result")   = assignments);
}
```

Timing

Table 1: Benchmarking result

test	replications	elapsed	relative	user.self	sys.self
mlKmeans(t(wine), 3)	100	0.028	1.000	0.028	0.000
kmeans(wine, 3)	100	0.947	33.821	0.484	0.424

Table taken 'as is' from RcppMLPACK vignette.

mlpack: Linear Regression Example

```
suppressMessages({library(utils); library(mlpack)})
data("trees", package="datasets")
X <- with(trees, cbind(log(Girth), log(Height)))
y <- with(trees, log(Volume))
lmfit <- lm(y ~ X)
# summary(fitted(lmfit))

lr <- linear_regression(training=X,
                        training_responses=as.matrix(y))
lrpred <- linear_regression(input_model=lr$output_model, test=X)
mlfit <- as.vector(lrpred$output_predictions)
# summary(mlfit)
all.equal(unname(fitted(lmfit)),  mlfit)

## [1] TRUE
```

mlpack 3.* now on CRAN

- There is more, much *much* more
- Due to the awesome work of our GSoC student Yashwant
- *Every* mlpack algo is now accessible
- Now I just have to update a few slides :)

Sugar

Syntactic 'sugar': Simulating π in R

Draw (x, y), compute distance to origin. Do so repeatedly, and ratio of points below one to number N of simulations will approach $\pi/4$ as we fill the area of 1/4 of the unit circle.

```
piR <- function(N) {
    x <- runif(N)
    y <- runif(N)
    d <- sqrt(x^2 + y^2)
    return(4 * sum(d <= 1.0) / N)
}
set.seed(5)
sapply(10^(3:6), piR)
```

```
## [1] 3.15600 3.15520 3.13900 3.14101
```

Syntactic 'sugar': Simulating π in C++

Rcpp sugar enables us to write C++ code that is almost as compact.

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

// [[Rcpp::export]]
double piSugar(const int N) {
  NumericVector x = runif(N);
  NumericVector y = runif(N);
  NumericVector d = sqrt(x*x + y*y);
  return 4.0 * sum(d <= 1.0) / N;
}
```

The code is essentially identical.

Syntactic 'sugar': Simulating π

And by using the same RNG, so are the results.

```
library(Rcpp)
sourceCpp("code/piSugar.cpp")
set.seed(42); a <- piR(1.0e7)
set.seed(42); b <- piSugar(1.0e7)
identical(a,b)
```

```
## [1] TRUE
```

```
print(c(a,b), digits=7)
```

```
## [1] 3.140899 3.140899
```

Syntactic 'sugar': Simulating π

The performance is close with a small gain for C++ as R is already vectorised:

```
library(rbenchmark)
sourceCpp("code/piSugar.cpp")
benchmark(piR(1.0e6), piSugar(1.0e6))[,1:4]
```

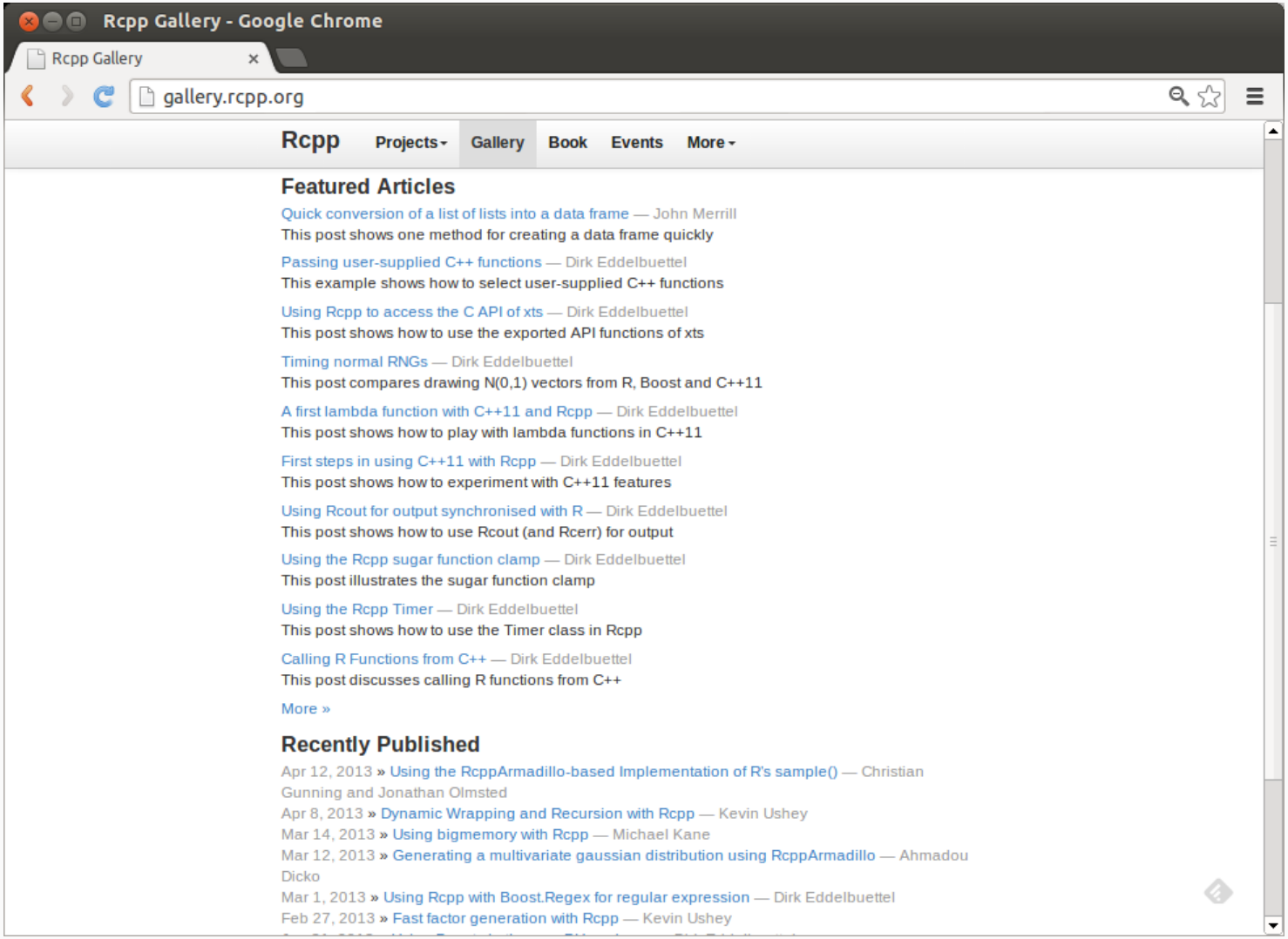
```
##             test replications elapsed relative
## 1     piR(1e+06)          100    4.79     3.09
## 2 piSugar(1e+06)          100    1.55     1.00
```

More

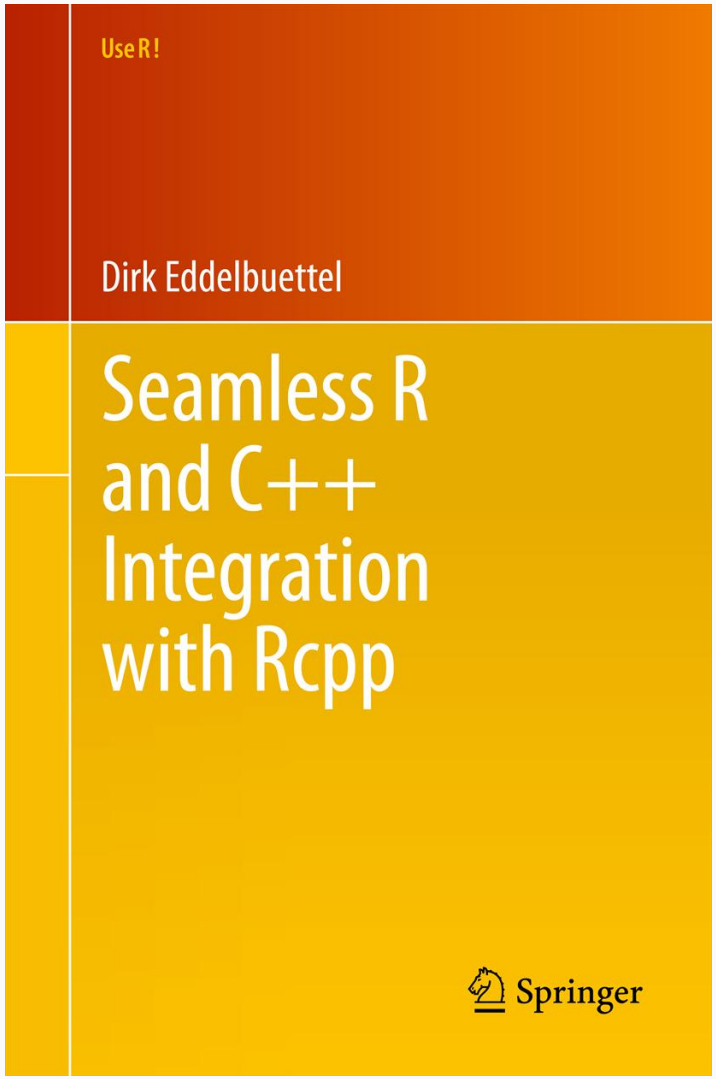
Documentation

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

Rcpp Gallery



The Rcpp book



On sale since June 2013.

Thank you!

slides	https://dirk.eddelbuettel.com/presentations/
web	https://dirk.eddelbuettel.com/
mail	dirk@eddelbuettel.com
github	@eddelbuettel
twitter	@eddelbuettel