Extending R with C++

Motivation, Examples, and Context

Dirk Eddelbuettel

20 April 2018

Debian / R Project / U of Illinois

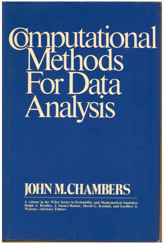
Outline

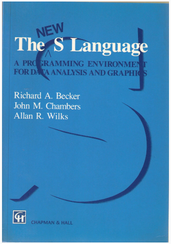
Agenda

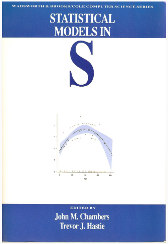
- (Very) Quick R Basics Reminder
- C++ in (way less than) a nutshell
- Extending R with C++ via Rpp
- A Worked Example
- A Case Study

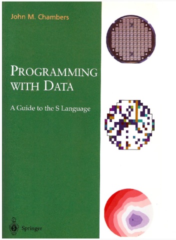
Why R?

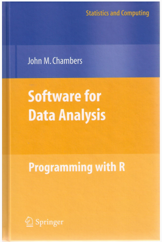
Programming with Data from 1977 to 2016

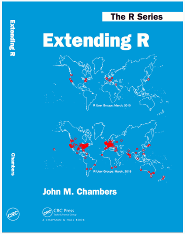










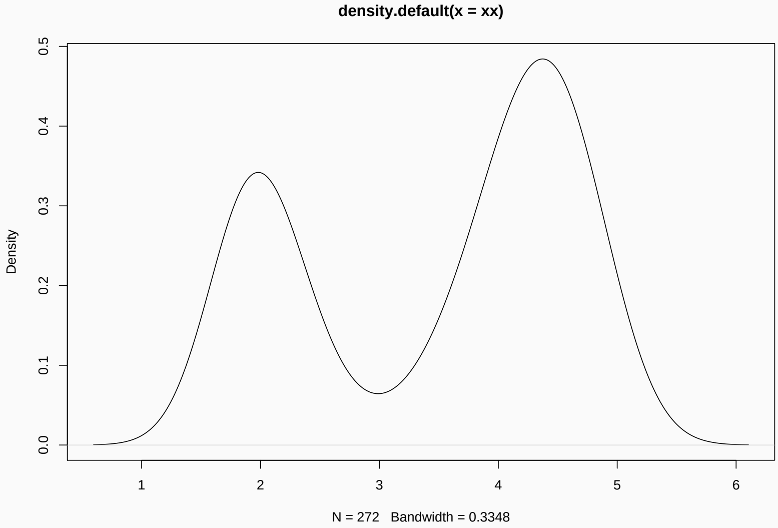


Thanks to John Chambers for high-resolution cover images. The publication years are, respectively, 1977, 1988, 1992, 1998, 2008 and 2016.

A Simple Example

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

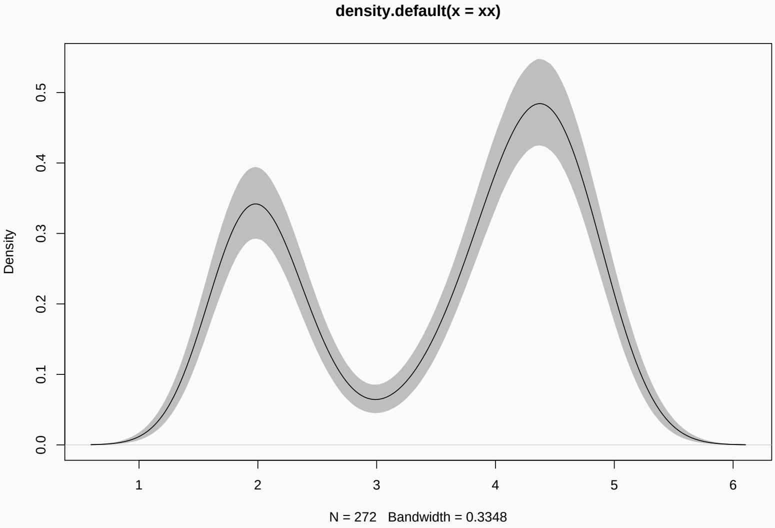
A Simple Example



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.

R as central point

R AS CENTRAL POINT

From any one of

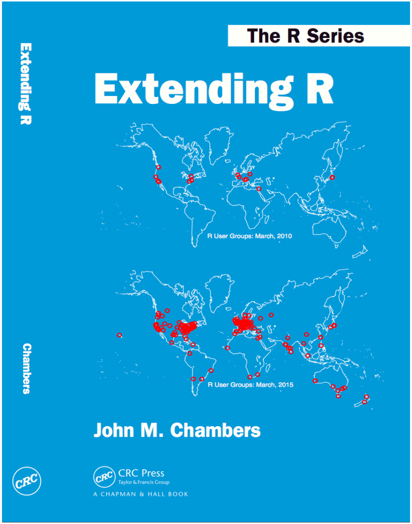
- csv
- txt
- xlsx
- xml, json, …
- web scraping, …
- hdf5, netcdf, …
- sas, stata, spss, …
- various SQL + NOSQL DBs
- various binary protocols

via

into any one of

- txt
- html
- latex and pdf
- html and js
- word
- shiny
- most graphics formats
- other dashboards
- web frontends

R per John Chambers (2016)



R per John Chambers (2016)

Three Principles (Section 1.1)

Object	Everything that exists in R is an object.
Function	Everything that happens in R is a function call.
Interface	Interfaces to other software are part of R.

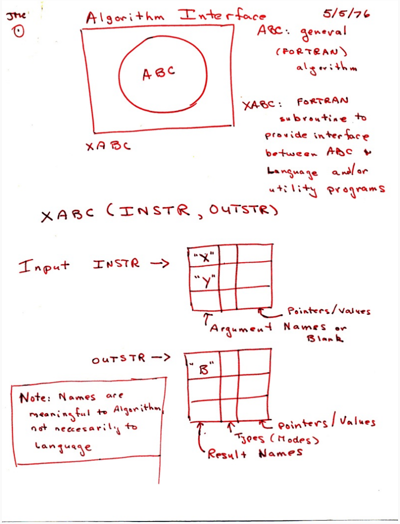
Three Principles (Section 1.1)

Object	Everything that exists in R is an object.
Function	Everything that happens in R is a function call.
Interface	Interfaces to other software are part of R.

R per John Chambers (2016)

That is new. Or is it?

R per John Chambers (2016)



Source: John Chamber, personal communication

Interface 1976

This became the system known as “Interface”, a precursor to S and R.

C++

Why C++?

- Asking Google leads to tens of million of 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, C++14, C++17, … a big deal giving us new language features
- While there are complexities, Rcpp users are mostly shielded

C++ In Too Little Time

Need to compile and link

```
#include <cstdio>

int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

Or streams output rather than `printf`

```
#include <iostream>

int main(void) {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Compiled not Interpreted

`g++ -o` will compile and link

Next: an example with explicit linking of an external library.

Compiled not Interpreted

```
#include <cstdio>

#define MATHLIB_STANDALONE
#include <Rmath.h>

int main(void) {
    printf("N(0,1) 95th percentile %9.8f\n",
           qnorm(0.95, 0.0, 1.0, 1, 0));
}
```

We may need to supply:

- *header location* via `-I`,
- *library location* via `-L`,
- *library* via `-llibraryname`

```
g++ -I/usr/include -c qnorm_rmath.cpp
g++ -o qnorm_rmath qnorm_rmath.o -L/usr/lib -lRmath
```

Statically Typed

- R is dynamically typed: `x <- 3.14; x <- "foo"` is valid.
- In C++, each variable must be declared before first use.
- Common types are `int` and `long` (possibly with `unsigned`), `float` and `double`, `bool`, as well as `char`.
- No standard string type, though `std::string` is close.
- All these variables types are scalars which is fundamentally different from R where everything is a vector.
- `class` (and `struct`) allow creation of composite types; classes add behaviour to data to form `objects`.
- Variables need to be declared, cannot change

C++ is a Better C

- control structures similar to what R offers: `for`, `while`, `if`, `switch`
- functions are similar too but note the difference in positional-only matching, also same function name but different arguments allowed in C++
- pointers and memory management: very different, but lots of issues people had with C can be avoided via STL (which is something Rcpp promotes too)
- sometimes still useful to know what a pointer is …

This is a second key feature of C++, and itis different from S3 and S4.

```
struct Date {
    unsigned int year;
    unsigned int month;
    unsigned int day
};
struct Person {
    char firstname[20];
    char lastname[20];
    struct Date birthday;
    unsigned long id;
};
```

Object-Oriented

Object-orientation matches data with code operating on it:

```
class Date {
private:
    unsigned int year
    unsigned int month;
    unsigned int date;
public:
    void setDate(int y, int m, int d);
    int getDay();
    int getMonth();
    int getYear();
}
```

Generic Programming and the STL

The STL promotes *generic* programming.

For example, the sequence container types `vector`, `deque`, and `list` all support

- `push_back()` to insert at the end;
- `pop_back()` to remove from the front;
- `begin()` returning an iterator to the first element;
- `end()` returning an iterator to just after the last element;
- `size()` for the number of elements;

but only `list` has `push_front()` and `pop_front()`.

Other useful containers: `set`, `multiset`, `map` and `multimap`.

Traversal of containers can be achieved via *iterators* which require suitable member functions `begin()` and `end()`:

```
std::vector<double>::const_iterator si;
for (si=s.begin(); si != s.end(); si++)
    std::cout << *si << std::endl;
```

Another key STL part are *algorithms*:

```
double sum = accumulate(s.begin(), s.end(), 0);
```

Some other STL algorithms are

- `find` finds the first element equal to the supplied value
- `count` counts the number of matching elements
- `transform` applies a supplied function to each element
- `for_each` sweeps over all elements, does not alter
- `inner_product` inner product of two vectors

Template programming provides a ‘language within C++’: code gets evaluated during compilation.

One of the simplest template examples is

```
template <typename T>
const T& min(const T& x, const T& y) {
    return y < x ? y : x;
}
```

This can now be used to compute the minimum between two `int` variables, or `double`, or in fact any *admissible type* providing an `operator<()` for less-than comparison.

Another template example is a class squaring its argument:

```
template <typename T>
class square : public std::unary_function<T,T> {
public:
    T operator()(T t) const {
        return t*t;
    }
};
```

which can be used along with STL algorithms:

```
transform(x.begin(), x.end(), square);
```

Further Reading

Books by Meyers are excellent

I also like the (free) C++ Annotations

C++ FAQ

Resources on StackOverflow such as

- general info and its links, eg
- booklist

Some tips:

- Generally painful, old-school `printf()` still pervasive
- Debuggers go along with compilers: `gdb` for `gcc` and `g++`; `lldb` for the clang / llvm family
- Extra tools such as `valgrind` helpful for memory debugging
- “Sanitizer” (ASAN/UBSAN) in newer versions of `g++` and `clang++`

Extending R with C++

Three key functions

- `evalCpp()`
- `sourceCpp()`
- `cppFunction()`

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] 1.797693e+308
```

Basic Usage: cppFunction()

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

```
cppFunction("
    int simpleExample() {
        int x = 10;
        return x;
}")
simpleExample()  # same identifier as C++ function
```

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
```

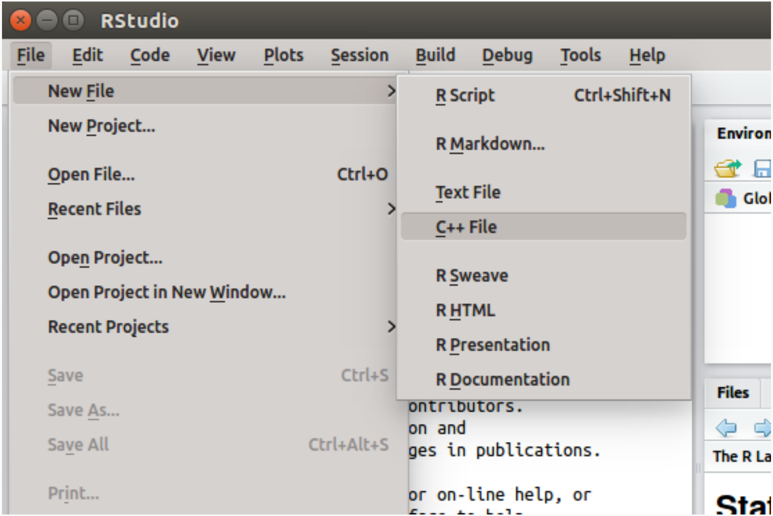
Basic Usage: sourceCpp()

`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 Uage: RStudio



Basic Uage: RStudio (Cont'ed)

The following file gets created:

```
#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)
*/
```

Basic Uage: RStudio (Cont'ed)

So what just happened?

- We defined a simple C++ function
- It operates on a numeric vector argument
- We asked 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

Basic Usage: Packages

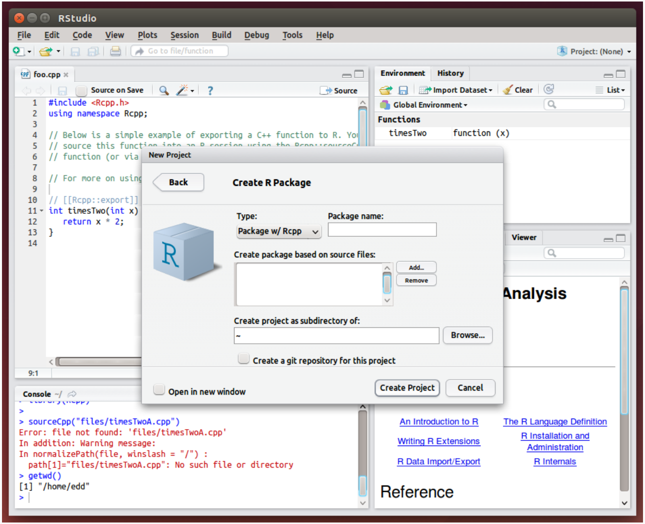
Package are *the* standard unit of R code organization.

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 April 2018, there are over 1300 packages on CRAN which use Rcpp, and a almost 100 more on BioConductor — with working, tested, and reviewed examples.

Packages and Rcpp



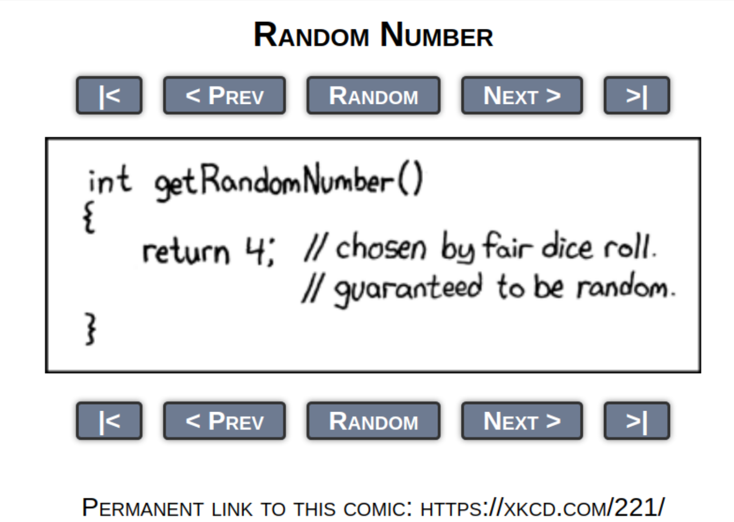
`Rcpp.package.skeleton()` and its derivatives. e.g. `RcppArmadillo.package.skeleton()` create working packages.

```
// 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;
}
```

NICE, BUT DOES IT *REALLY* WORK?

Something self-contained

- Let's talk random numbers!
- We'll look at a quick generator
- And wrap it in plain C / C++

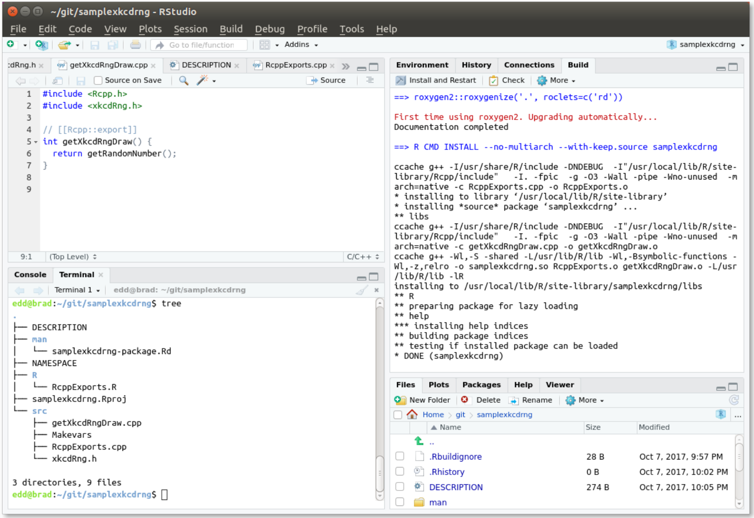

RANDOM NUMBER
|< < PREV RANDOM NEXT > >|
int getRandomNumber()
{
return 4; // chosen by fair dice roll.
// guaranteed to be random.
}
|< < PREV RANDOM NEXT > >|
PERMANENT LINK TO THIS COMIC: HTTPS://XKCD.COM/221/

```
// cf https://xkcd.com/221/
//
//    "RFC 1149.5 specifies 4 as the "
//    "standard IEEE-vetted random number."

int getRandomNumber()
{
    return 4; // chosen by fair dice roll
              // guaranteed to be random
}
```

getXkcdRngDraw()

```
#include <Rcpp.h>
#include <xkcdRng.h>

// [[Rcpp::export]]
int getXkcdRngDraw() {
    return getRandomNumber();
}
```

Package


~/git/samplexkcdrng - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
#include <Rcpp.h>
#include <xkcdRng.h>
// [[Rcpp::export]]
int getXkcdRngDraw() {
return getRandomNumber();
}
==> roxygen2::roxygenize('.', roclets=c('rd'))
First time using roxygen2. Upgrading automatically...
Documentation completed
==> R CMD INSTALL --no-multiarch --with-keep.source samplexkcdrng
* DONE (samplexkcdrng)
edd@brad:~/git/samplexkcdrng$ tree
DESCRIPTION
man
samplexkcdrng-package.Rd
NAMESPACE
R
RcppExports.R
samplexkcdrng.Rproj
src
getXkcdRngDraw.cpp
Makevars
RcppExports.cpp
xkcdRng.h
3 directories, 9 files

What Did We Do?

- An unmodified piece of C / C++ code
- A simple interface function
- Rcpp does the rest

A Case Study

A Benchmark Comparison

A recent blogpost on "finding a needle in a haystack" has a nice story:

```
options(width=50)
set.seed(1)
haystack <- sample(0:12, size = 2000, replace = TRUE)
needle   <- c(2L, 10L, 8L)
haystack[1:60]
```

```
##  [1]  3  4  7 11  2 11 12  8  8  0  2  2  8  4 10
## [16]  6  9 12  4 10 12  2  8  1  3  5  0  4 11  4
## [31]  6  7  6  2 10  8 10  1  9  5 10  8 10  7  6
## [46] 10  0  6  9  9  6 11  5  3  0  1  4  6  8  5
```

A recent blogpost on "finding a needle in a haystack" has a nice story:

```
options(width=50)
set.seed(1)
haystack <- sample(0:12, size = 2000, replace = TRUE)
needle   <- c(2L, 10L, 8L)
haystack[1:60]
```

```
##  [1]  3  4  7 11  2 11 12  8  8  0  2  2  8  4 10
## [16]  6  9 12  4 10 12  2  8  1  3  5  0  4 11  4
## [31]  6  7  6  2 10  8 10  1  9  5 10  8 10  7  6
## [46] 10  0  6  9  9  6 11  5  3  0  1  4  6  8  5
```

First Candidate

```r
forloop_find <- function(needle, haystack) {
    n <- length(needle) - 1L
    for (i in seq(haystack)) {
        if (identical(haystack[i:(i+n)], needle)) {
            return(i)
        }
    }
}
forloop_find(needle, haystack)
```

```
## [1] 34
```

Second Candidate

```
lead_find <- function(needle, haystack) {
    v  <- haystack == needle[1]
    for (i in seq(2, length(needle))) {
        v <- v +
            (dplyr::lead(haystack, i-1L) == needle[i])
    }
    which(v == length(needle))[1L]
}
lead_find(needle, haystack)
```

```
## [1] 34
```

Third Candidate

```r
shift_find <- function(needle, haystack) {
    shifted_haystack <-
        data.table::shift(haystack, type='lead',
                          0:(length(needle)-1))

    v <- Map('==', shifted_haystack, needle)
    v <- Reduce('+', v)
    which(v == length(needle))[1]
}
shift_find(needle, haystack)
```

```
## [1] 34
```

Fourth Candidate

```
Rcpp::cppFunction('int rcpp_find(NumericVector needle,
                                 NumericVector haystack) {
    int nlen = needle.size(), hlen = haystack.size(), j;
    for (int i = 0; i < (hlen - nlen); i++) {
        for (j = 0; j < nlen; j++) {
           if (needle[j] != haystack[i + j]) break;
        }
        if (j == nlen) return(i+1);
    }
    return(0);
}')
rcpp_find(needle, haystack)
```

```
## [1] 34
```

Fifth Candidate

```
Rcpp::cppFunction('
int idiomaticrcpp_find(NumericVector needle,
                       NumericVector haystack) {
    NumericVector::iterator it;
    it = std::search(haystack.begin(), haystack.end(),
                     needle.begin(), needle.end());
    int pos = it - haystack.begin() + 1;
    if (pos > haystack.size()) pos = -1;
    return(pos);
}')
idiomaticrcpp_find(needle, haystack)
```

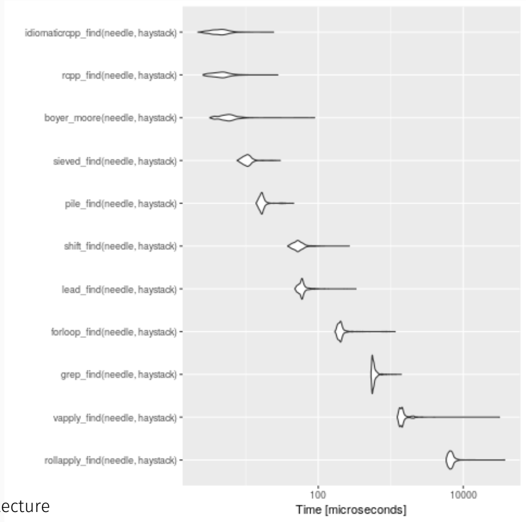
```
## [1] 34
```

Shootout

```
R> res <- microbenchmark::microbenchmark(...) # not shown
R> res
Unit: microseconds
                               expr      min        lq       mean    median        uq       max neval
     rollapply_find(needle, haystack) 5829.484 6355.7290 6918.75051 6719.5825 7114.4770 37348.915  1000
        vapply_find(needle, haystack) 1230.373 1338.3275 1519.16471 1419.7170 1491.9485 31687.188  1000
          grep_find(needle, haystack)  535.059  556.3545  592.86697  572.5000  597.6460  1396.680  1000
       forloop_find(needle, haystack)  169.751  189.5370  213.39386  200.6070  210.2785  1151.483  1000
          lead_find(needle, haystack)   47.571   55.4575   61.54025   59.2370   62.3445   331.499  1000
         shift_find(needle, haystack)   37.939   47.5780   55.14043   52.3475   58.5400   268.533  1000
          pile_find(needle, haystack)   13.883   15.7375   17.10174   16.5590   17.4175    45.757  1000
        sieved_find(needle, haystack)    7.587    9.4575   10.82973   10.4355   11.3620    29.978  1000
        boyer_moore(needle, haystack)    3.197    4.7035    6.06770    5.6540    6.5950    89.414  1000
          rcpp_find(needle, haystack)    2.579    3.6805    4.86756    4.5465    5.3570    27.765  1000
 idiomaticrcpp_find(needle, haystack)    2.183    3.5230    4.62004    4.3555    5.1945    24.235  1000
R>
```

Shootout

```r
ggplot2::autoplot(res)
```



Conclusion

Takeaways on *Extending R with C++*

- clearly possible as the tooling helps greatly
- natural as interfaces are a normal part of R
- not too hard, though balancing two languages
- rewarding in terms of performance
- always measure and profile

Thank you!

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