>eRka()
Entuzjastów R
Krakowska Alternatywa

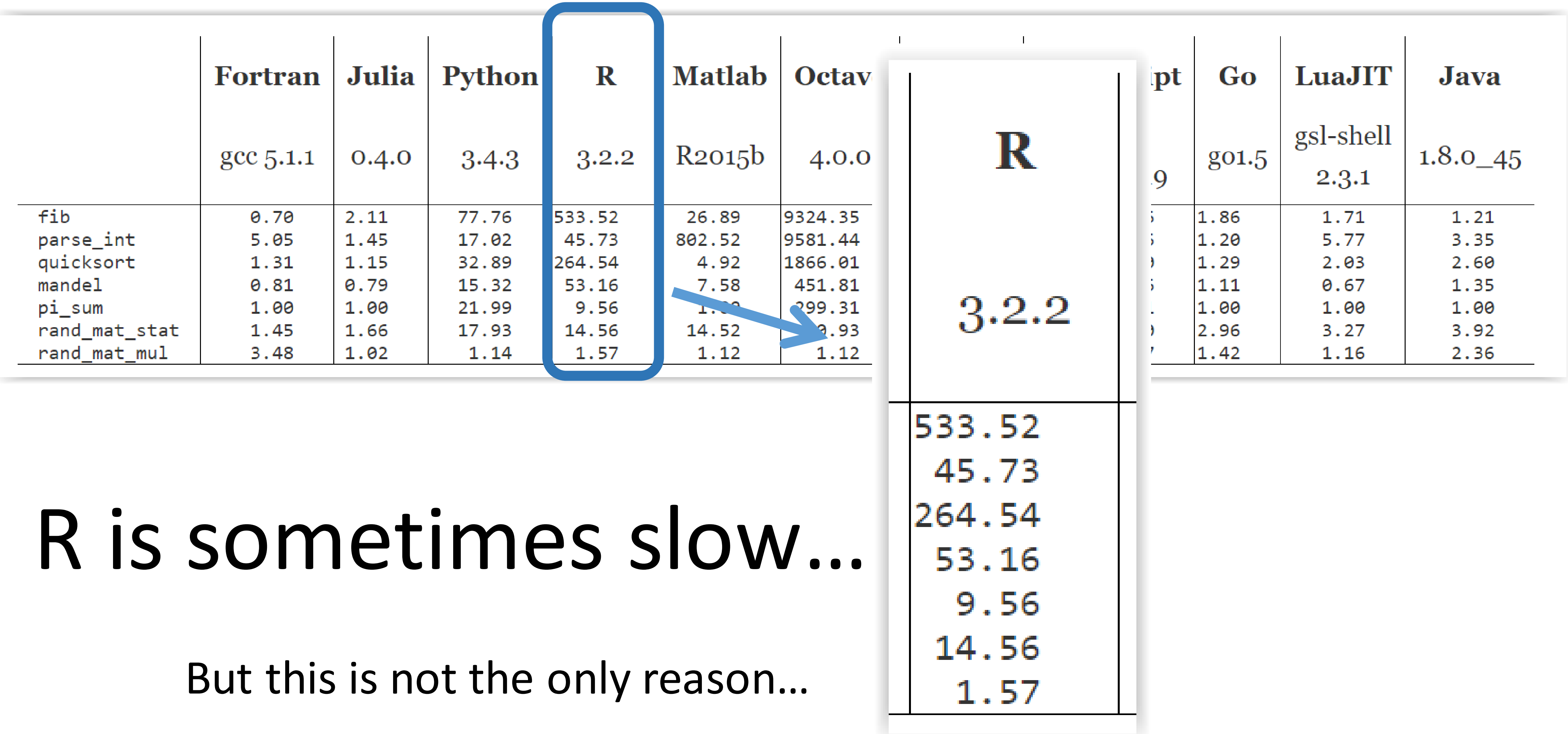
# I feel need. The need for speed. In other words - C++ inside R

Zygmunt Zawadzki

# Quick tour

- R – why we need C++?
- C++ - (subjective) facts and myths.
- Rcpp – how you can easily connect R and C++?

Dlaczego potrzebujemy wsparcia C++?



| | Fortran<br>gcc 5.1.1 | Julia<br>0.4.0 | Python<br>3.4.3 | R<br>3.2.2 | Matlab<br>R2015b | Octav<br>4.0.0 | ipt<br>9 | Go<br>go1.5 | LuaJIT<br>gsl-shell<br>2.3.1 | Java<br>1.8.0_45 |
|---|---|---|---|---|---|---|---|---|---|---|
| fib | 0.70 | 2.11 | 77.76 | 533.52 | 26.89 | 9324.35 | [illegible] | 1.86 | 1.71 | 1.21 |
| parse_int | 5.05 | 1.45 | 17.02 | 45.73 | 802.52 | 9581.44 | [illegible] | 1.20 | 5.77 | 3.35 |
| quicksort | 1.31 | 1.15 | 32.89 | 264.54 | 4.92 | 1866.01 | [illegible] | 1.29 | 2.03 | 2.60 |
| mandel | 0.81 | 0.79 | 15.32 | 53.16 | 7.58 | 451.81 | [illegible] | 1.11 | 0.67 | 1.35 |
| pi_sum | 1.00 | 1.00 | 21.99 | 9.56 | [illegible] | 299.31 | [illegible] | 1.00 | 1.00 | 1.00 |
| rand_mat_stat | 1.45 | 1.66 | 17.93 | 14.56 | 14.52 | [illegible].93 | [illegible] | 2.96 | 3.27 | 3.92 |
| rand_mat_mul | 3.48 | 1.02 | 1.14 | 1.57 | 1.12 | 1.12 | [illegible] | 1.42 | 1.16 | 2.36 |

| R<br>3.2.2 |
|---|
| 533.52 |
| 45.73 |
| 264.54 |
| 53.16 |
| 9.56 |
| 14.56 |
| 1.57 |

# R is sometimes slow...

But this is not the only reason...

# R is not the only one. There are some nice things in other languages too.

Why not use them?

C++ can be a bridge.

**Armadillo**
C++ linear algebra library

mlpack

NVIDIA.
CUDA.

redis

And lot’s more!

C++ ma dostęp do bogatego zbioru struktur danych:

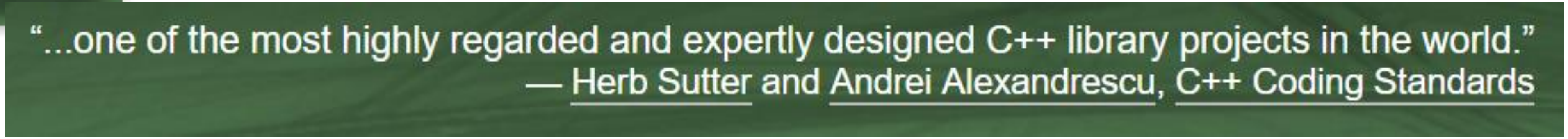
Vectors, sets, maps, stacks...

Besides C++ has boost

C++ LIBRARIES

boost

C++ LIBRARIES



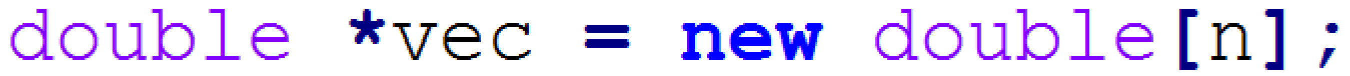
“...one of the most highly regarded and expertly designed C++ library projects in the world.”
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

# C++ - (subjective) facts and myths

C++ is too close to the raw memory for the casual user. Memory leaks is everyday life …

… unless you go to the right path of the STL.

# The allocation of a vector in C++:



```
double *vec = new double[n];
```

# NO!

# NIE!

Use of the new can be very dangerous.

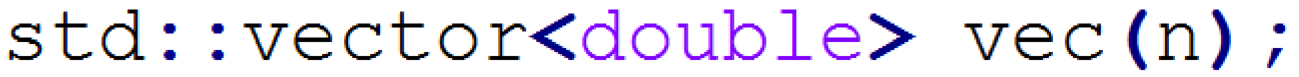
Memory management becomes complex and the lack of calls the delete will lead to a memory leak.

At the beginning it is better to avoid pointers, new and delete.

Tutorials often introduce pointers earlier than necessary. Therefore, the novice can get the impression* that you have to write the code with pointers.

* - I had such impression…

# Righteous path of STL:



```
std::vector<double> vec(n);
```

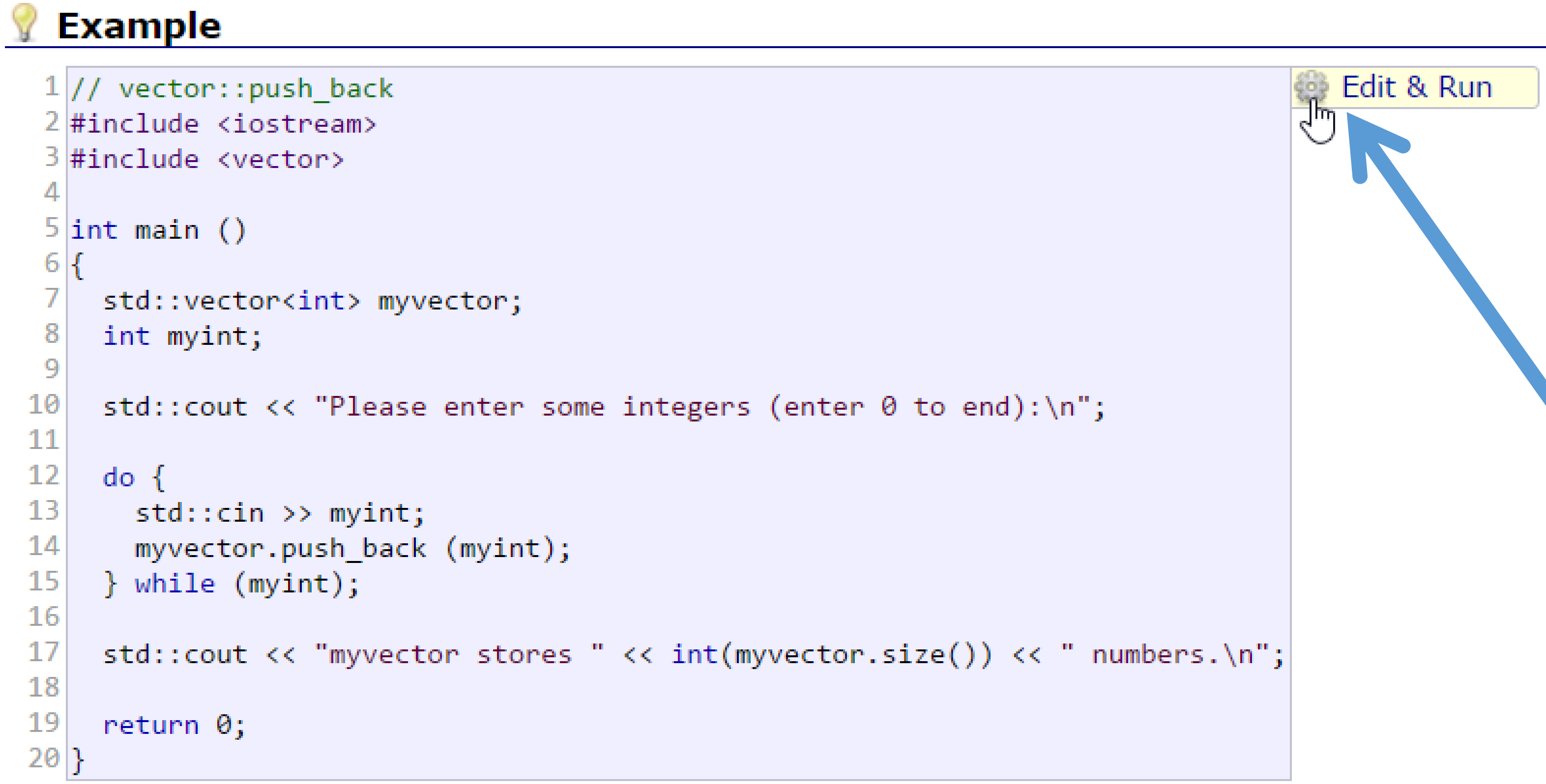
- Memory management.
- Efficient implementations.
- Hundreds of algorithms.

# STL

# Some C++ resources:

- http://www.cplusplus.com/
- http://en.cppreference.com/w/

# http://www.cplusplus.com/reference/vector/vector/push_back/



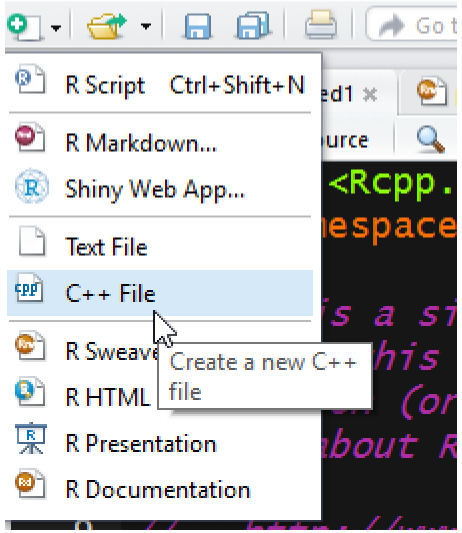
**Example**

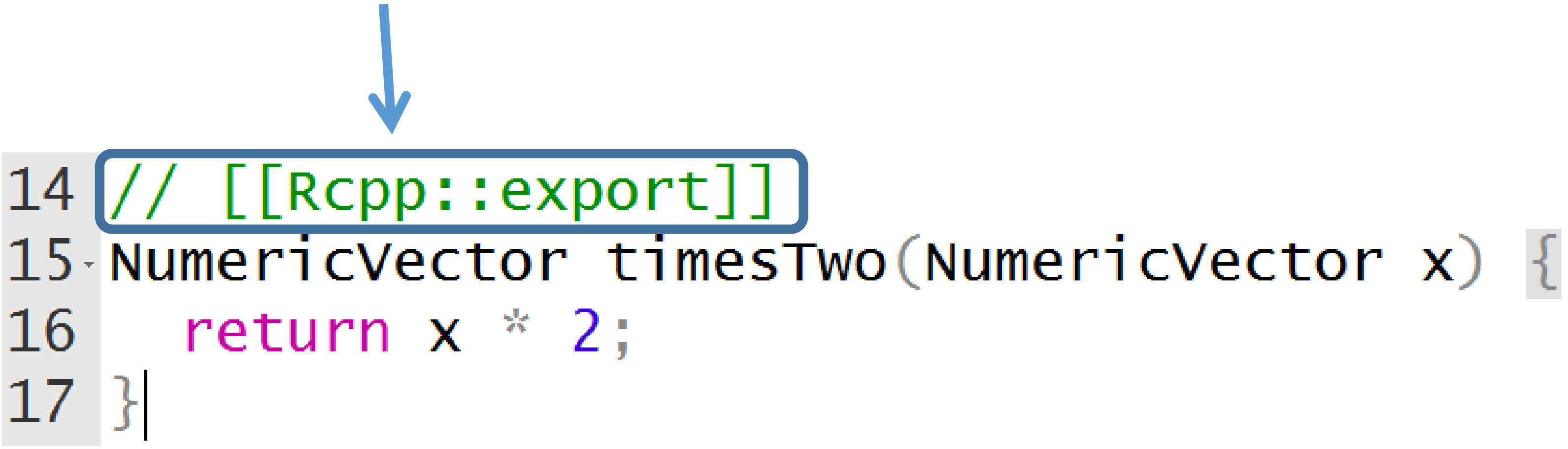
```
// vector::push_back
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector;
  int myint;

  std::cout << "Please enter some integers (enter 0 to end):\n";

  do {
    std::cin >> myint;
    myvector.push_back (myint);
  } while (myint);

  std::cout << "myvector stores " << int(myvector.size()) << " numbers.\n";

  return 0;
}
```

Run and compile in the browser!

# Rcpp – how you can easily connect R and C++?

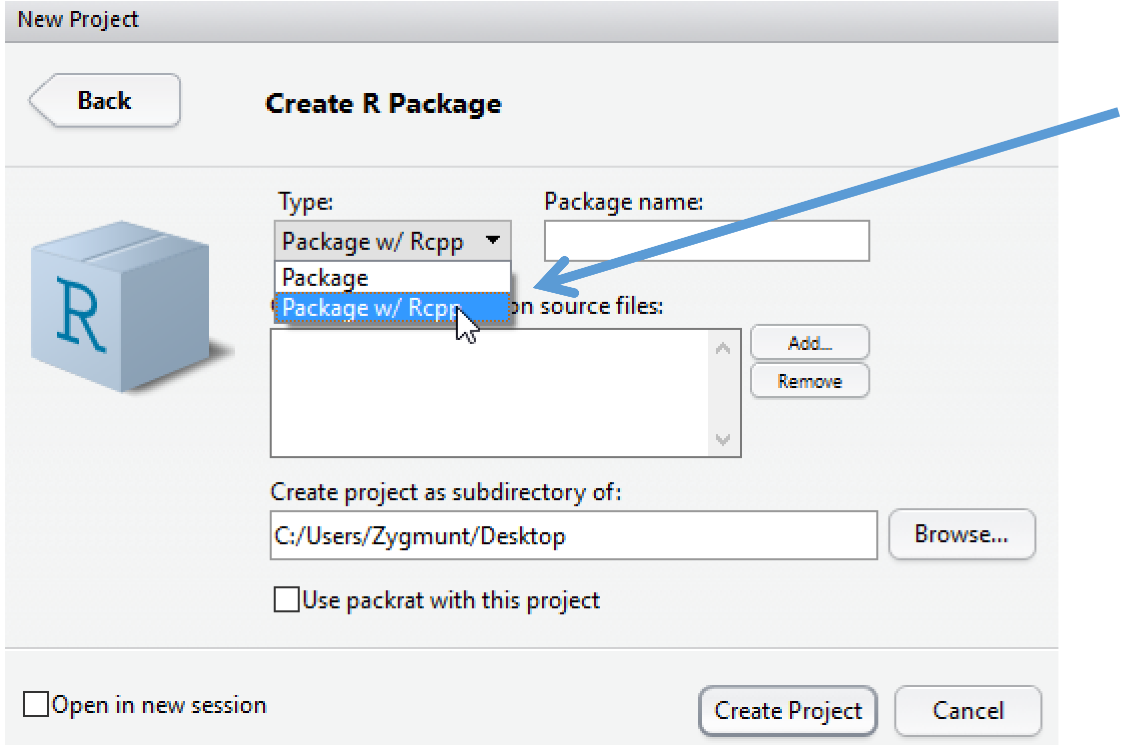
All the functions that contain such entry above its definition will be exported to R.





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

# R package with C++:



Type: Package w/ Rcpp

Package name:

Create project as subdirectory of: C:/Users/Zygmunt/Desktop

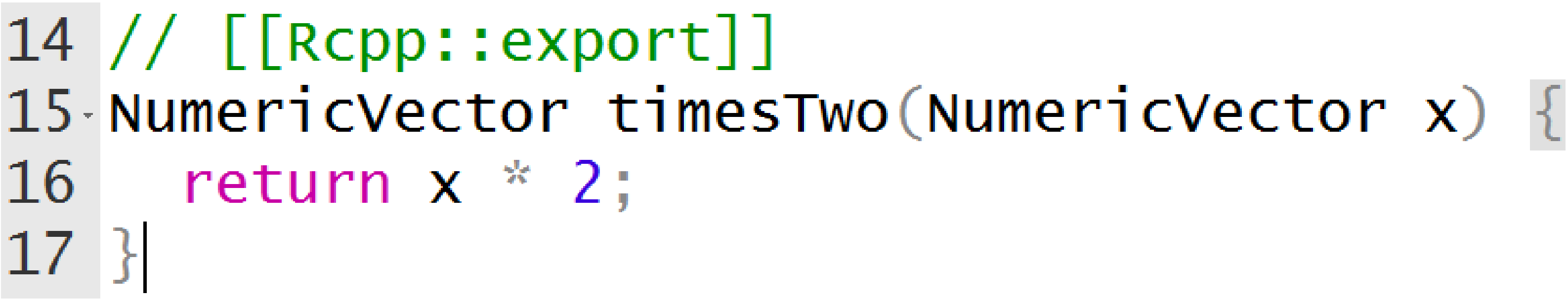
☐ Use packrat with this project

☐ Open in new session

At the stage of creating the package, select the Package in/Rcpp and that’s all.

# Structures from R in Rcpp

| | | |
|---|---|---|
| numeric | ↔ | NumericVector |
| integer | ↔ | IntegerVector |
| character | ↔ | CharacterVector |
| matrix | ↔ | NumericMatrix |
| list | ↔ | List |



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

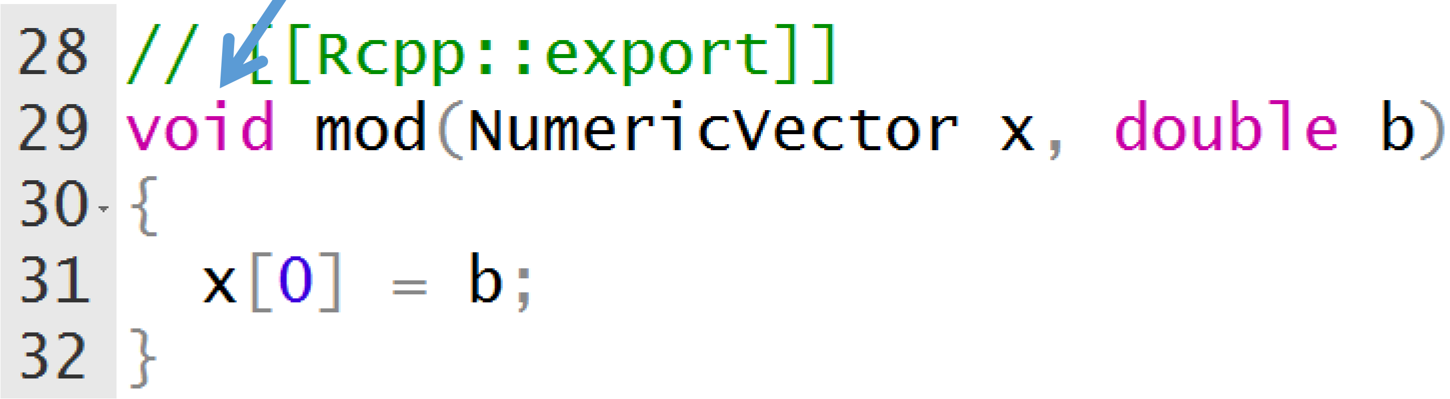
# WARNING!

# WARNING!

Rcpp by default passes object by reference.

There is no copy!.

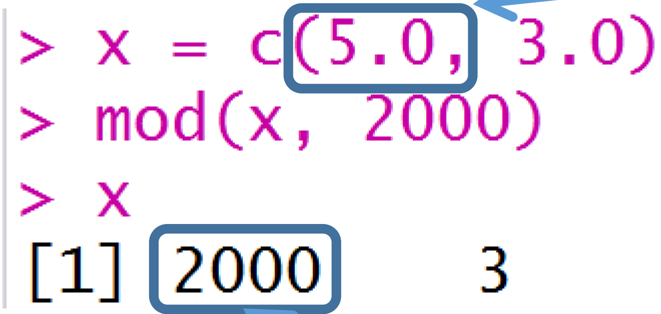
Function returns nothing.



```
// [[Rcpp::export]]
void mod(NumericVector x, double b)
{
  x[0] = b;
}
```

Modifying object in C++ means modifying it in R!

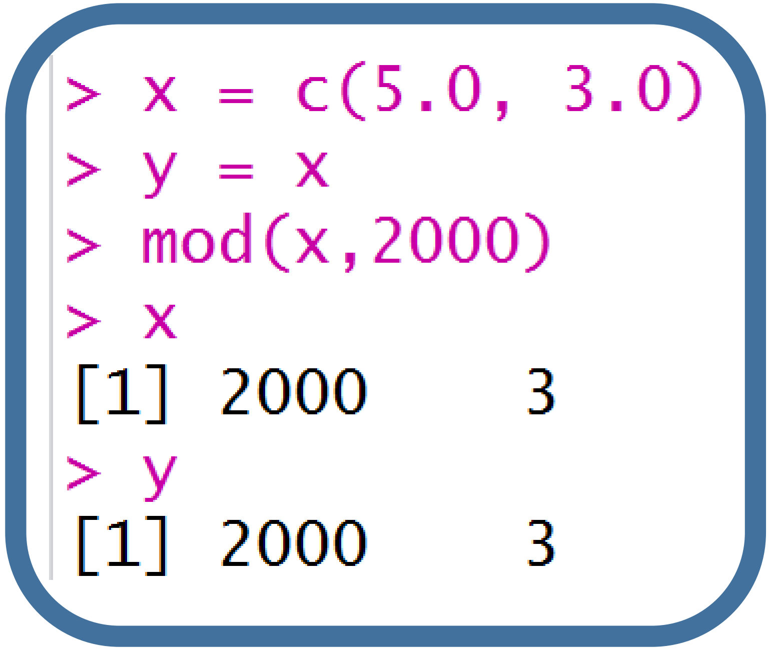
Should be…



```
> x = c(5.0, 3.0)
> mod(x, 2000)
> x
[1] 2000     3
```

… but there is…

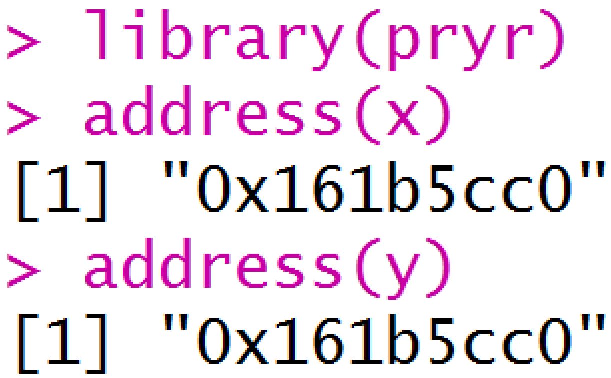
R not always copies the objects…



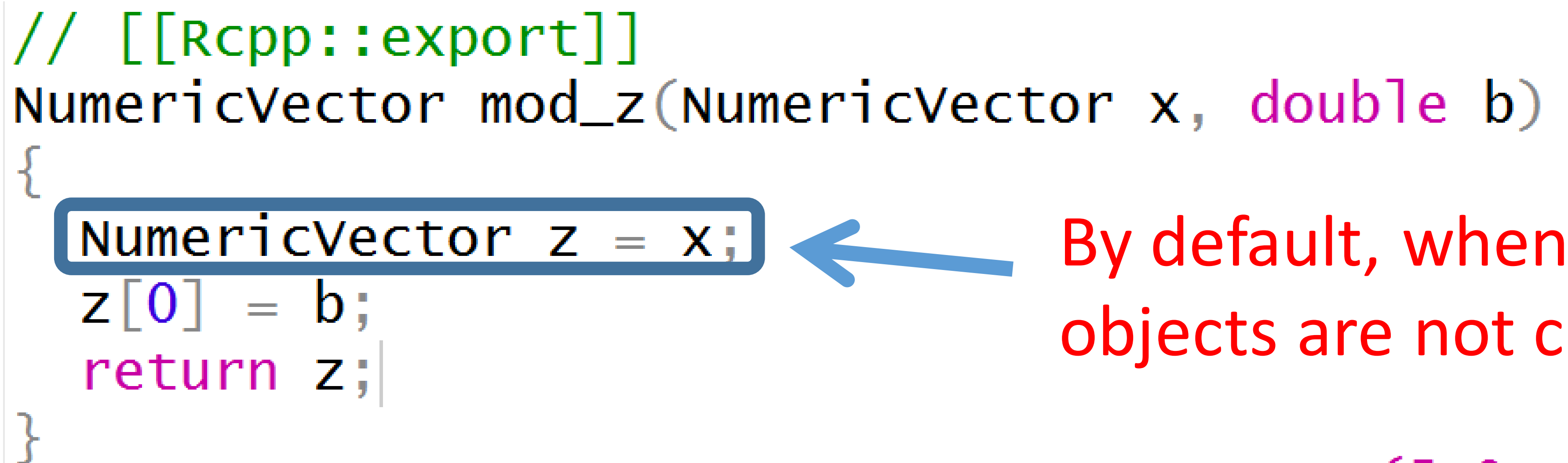
```
> x = c(5.0, 3.0)
> y = x
> mod(x,2000)
> x
[1] 2000    3
> y
[1] 2000    3
```

… and it leads to tragedy…

pryr allows you to better understand how R manages memory.

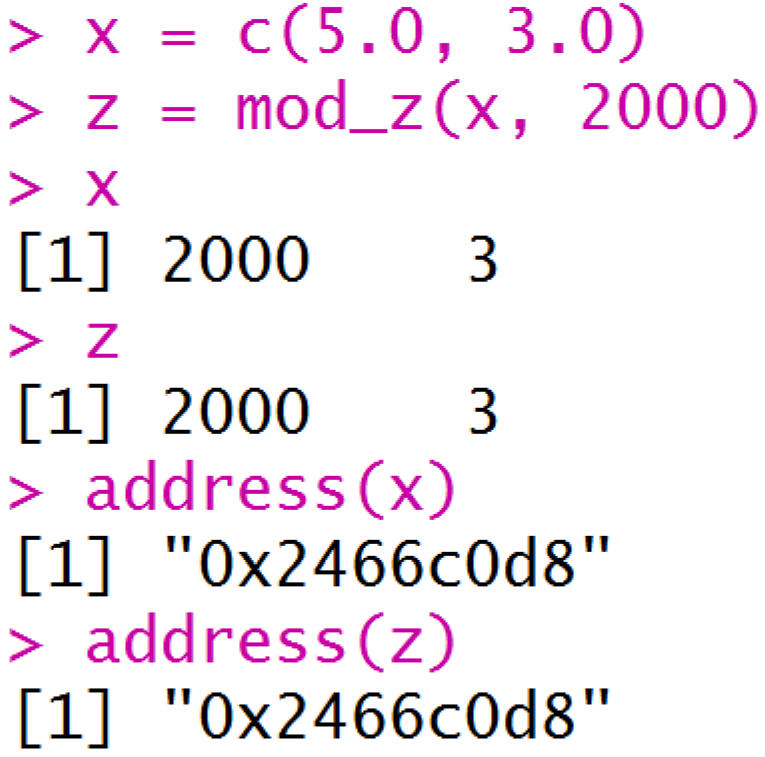


```
> library(pryr)
> address(x)
[1] "0x161b5cc0"
> address(y)
[1] "0x161b5cc0"
```



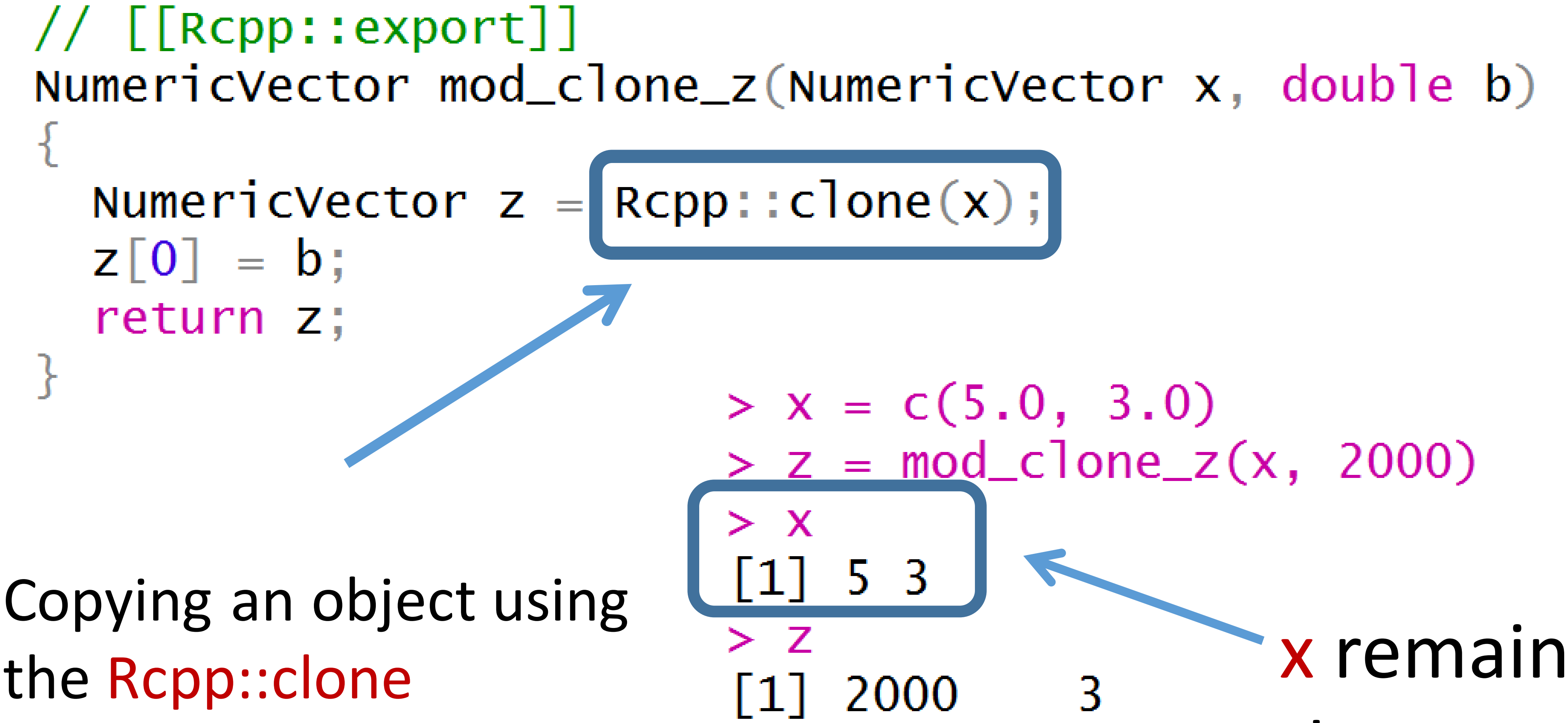
```
// [[Rcpp::export]]
NumericVector mod_z(NumericVector x, double b)
{
  NumericVector z = x;
  z[0] = b;
  return z;
}
```

By default, when assigning objects are not copied!



```
> x = c(5.0, 3.0)
> z = mod_z(x, 2000)
> x
[1] 2000    3
> z
[1] 2000    3
> address(x)
[1] "0x2466c0d8"
> address(z)
[1] "0x2466c0d8"
```

Z created in C++ has the same address as X.



```
// [[Rcpp::export]]
NumericVector mod_clone_z(NumericVector x, double b)
{
  NumericVector z = Rcpp::clone(x);
  z[0] = b;
  return z;
}
```

```
> x = c(5.0, 3.0)
> z = mod_clone_z(x, 2000)
> x
[1] 5 3
> z
[1] 2000    3
```

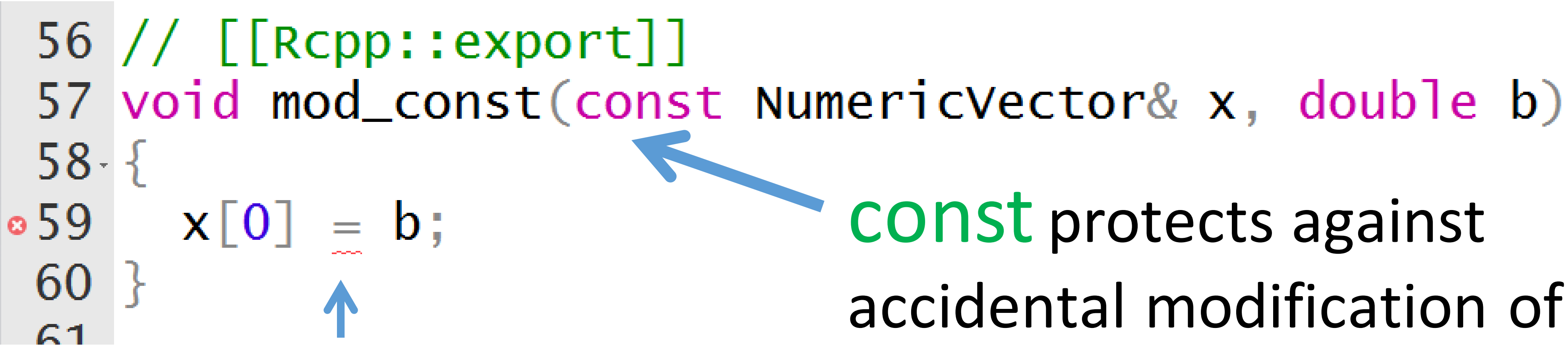
Copying an object using the Rcpp::clone

x remained the same!

# Const reference.

When writing in C++ a good habit to pass all arguments by the constant reference.

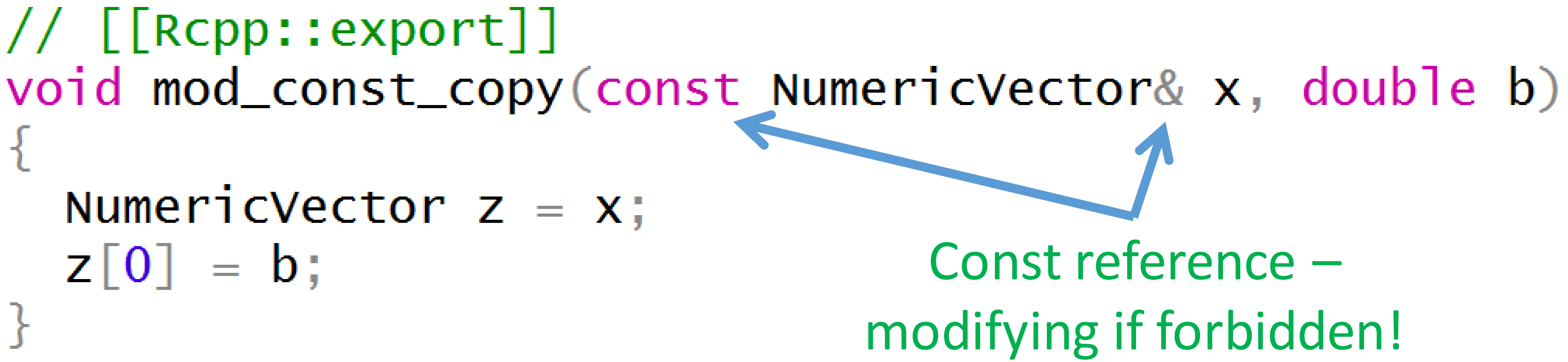
- Reference, makes that the objects are not copied unnecessarily.
- const protects against accidental modification of the object.



```
// [[Rcpp::export]]
void mod_const(const NumericVector& x, double b)
{
  x[0] = b;
}
```

Won’t compile!

const protects against accidental modification of the object.

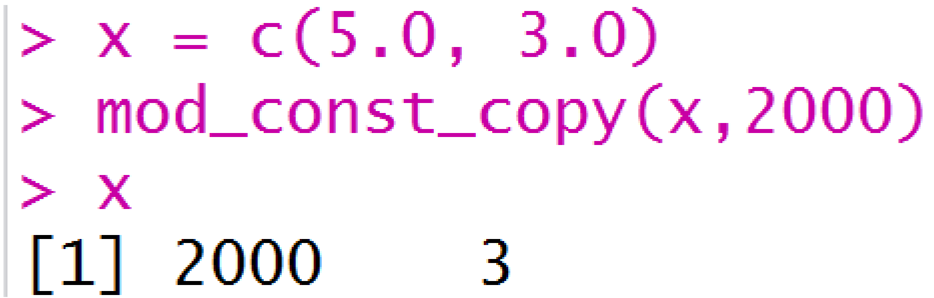


```
// [[Rcpp::export]]
void mod_const_copy(const NumericVector& x, double b)
{
  NumericVector z = x;
  z[0] = b;
}
```

Const reference – modifying if forbidden!

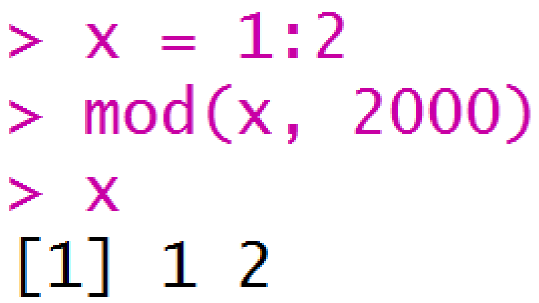
You can complie this!!!

Constant value was modified!



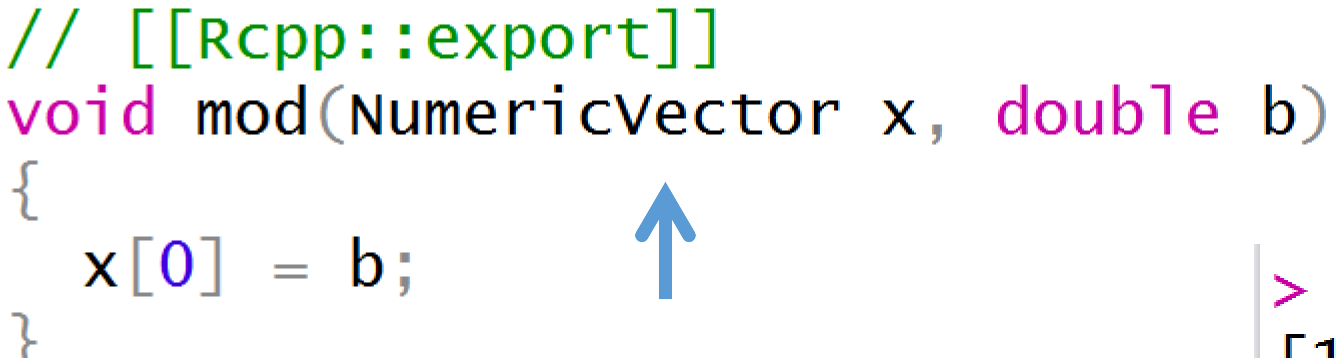
```
> x = c(5.0, 3.0)
> mod_const_copy(x,2000)
> x
[1] 2000    3
```

# Automatic conversions in Rcpp



```
> x = 1:2
> mod(x, 2000)
> x
[1] 1 2
```

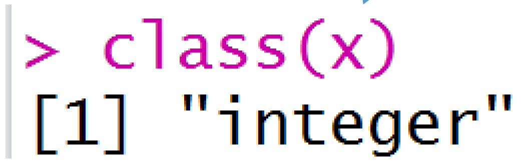
In some cases modifying in C++ doesn't work!



```
// [[Rcpp::export]]
void mod(NumericVector x, double b)
{
  x[0] = b;
}
```

Expected NumericVector

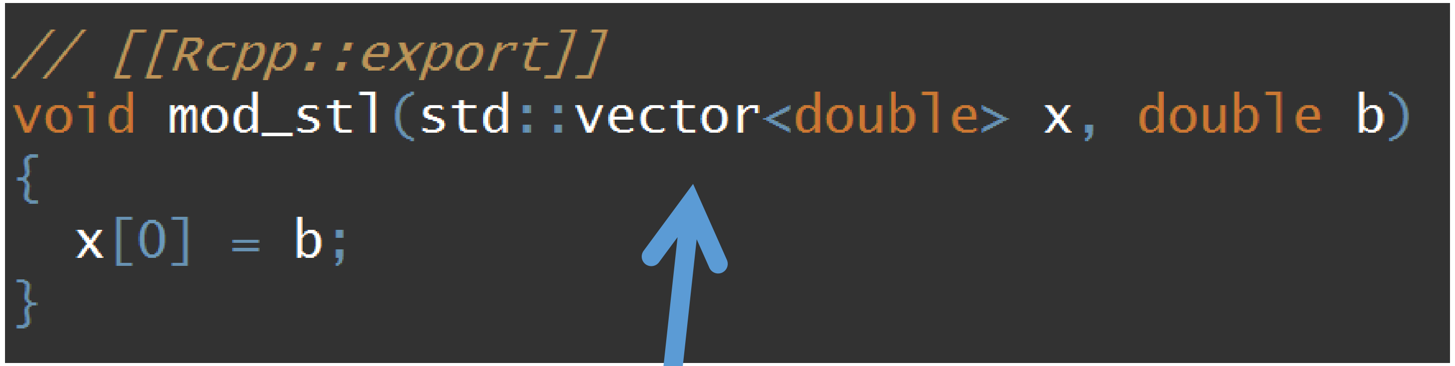
x is an IntegerVector in R



```
> class(x)
[1] "integer"
```

If the type of object passed to function is other than expected Rcpp tries to convert input. In such situation object is copied.

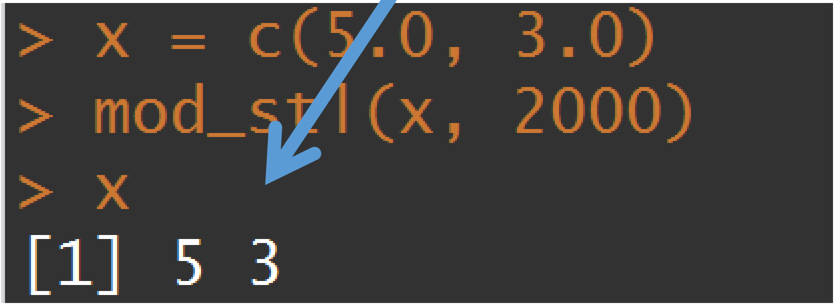
# We can always copy R object by using STL.



```
// [[Rcpp::export]]
void mod_stl(std::vector<double> x, double b)
{
  x[0] = b;
}
```

In Rcpp there's no problem with using STL vector as function argument!!!

X wasn't modified!



```
> x = c(5.0, 3.0)
> mod_stl(x, 2000)
> x
[1] 5 3
```

# Thanks for your attention!

>eRka()

Entuzjastów R
Krakowska Alternatywa

# STL