Rcpp11

Romain François

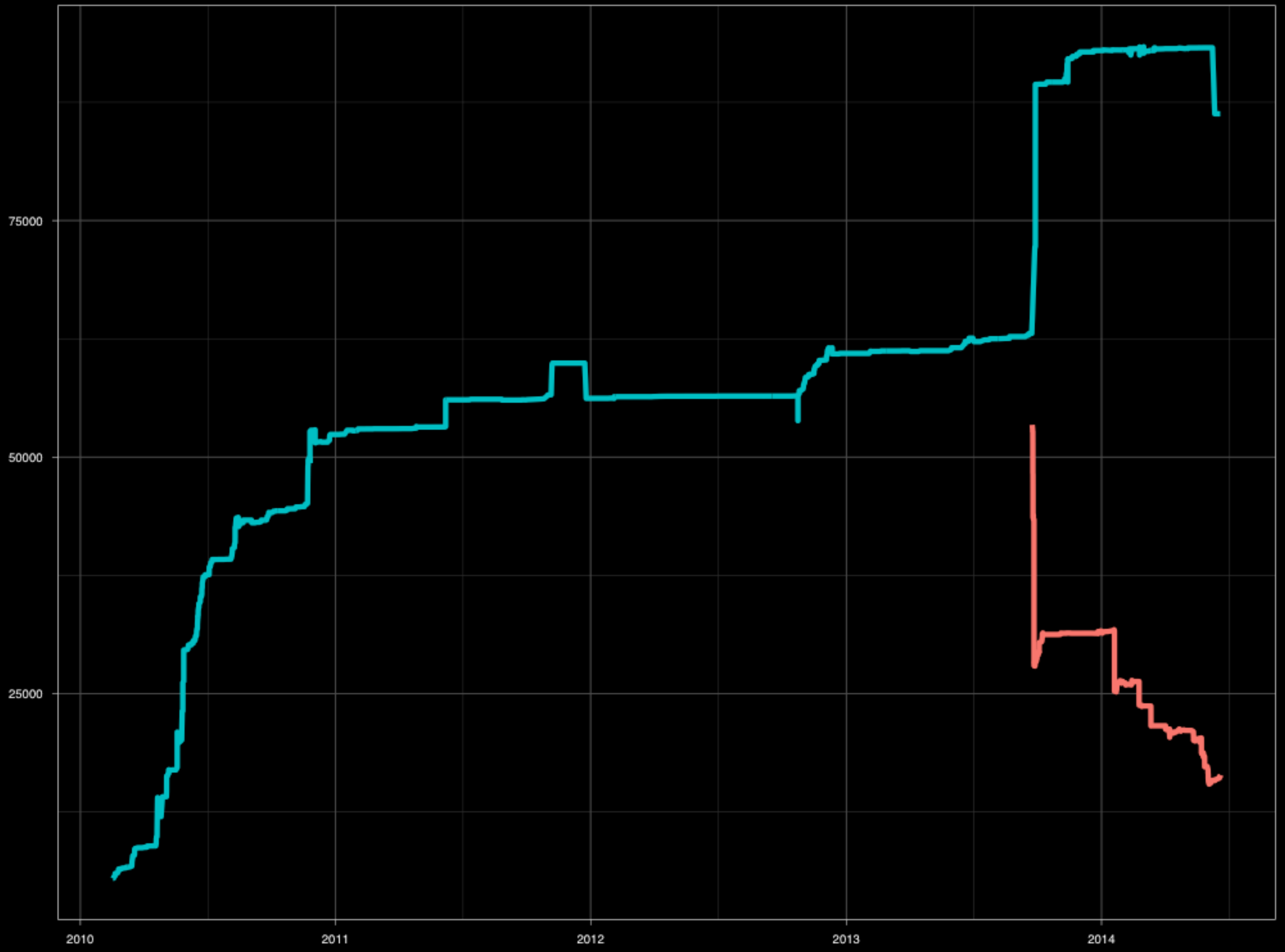
romain@r-enthusiasts.com @romain_francois

I  R


I
Rcpp

Rcpp11

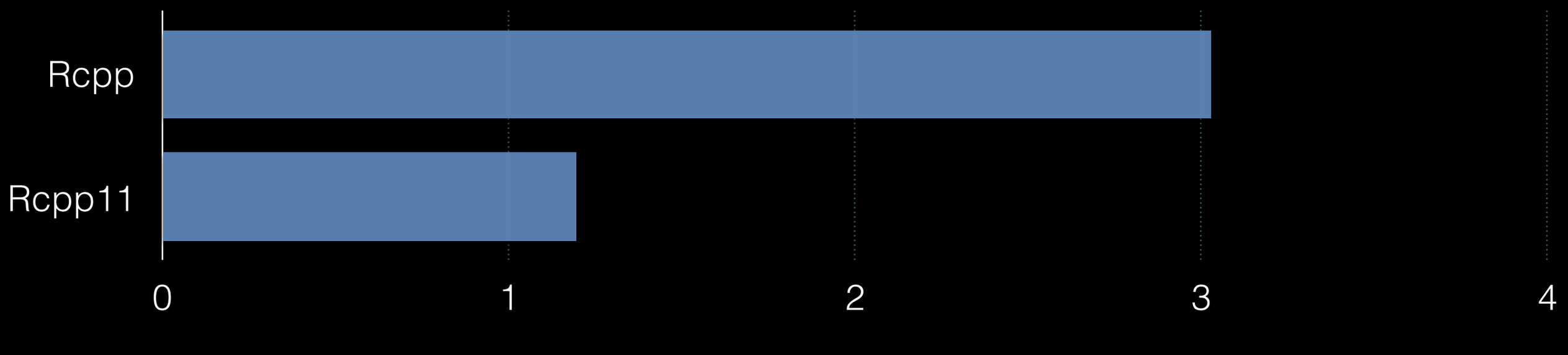
- C++ = C++11
- Smaller code base (than Rcpp)
- Header only


75000
50000
25000
2010
2011
2012
2013
2014

Less code -> faster compiles

```
#include <Rcpp.h>

// [[Rcpp::export]]
int foo(){
    return 2 ;
}
```



Interface simplification

NumericVector constructors

```
Vector() {
Vector( const Vector& other){
Vector( SEXP x ) {

Vector( const GenericProxy<Proxy>& proxy ){
explicit Vector( const no_init& obj) {
Vector( const int& size, const stored_type& u ) {
Vector( const std::string& st ){
Vector( const char* st ) {
Vector( const int& siz, stored_type (*gen)(void) ) {
Vector( const int& size ) {
Vector( const Dimension& dims) {
Vector( const Dimension& dims, const U& u) {
Vector( const VectorBase<RTYPE,NA,VEC>& other ) {
Vector( const int& size, const U& u) {
Vector( const sugar::SingleLogicalResult<NA,T>& obj ) {
Vector( const int& siz, stored_type (*gen)(U1), const U1& u1) {
Vector( const int& siz, stored_type (*gen)(U1,U2), const U1& u1, const U2& u2) {
Vector( const int& siz, stored_type (*gen)(U1,U2,U3), const U1& u1, const U2& u2, const U3& u3) {
Vector( InputIterator first, InputIterator last){
Vector( InputIterator first, InputIterator last, int n) {
Vector( InputIterator first, InputIterator last, Func func) {
Vector( InputIterator first, InputIterator last, Func func, int n){
Vector( std::initializer_list<init_type> list ) {
```

23 constructors in Rcpp

NumericVector constructors

```
Vector(){
Vector( const Vector& other){
Vector( Vector&& other){

explicit Vector(int n) {
explicit Vector(R_xlen_t n) {

Vector( R_xlen_t n, value_type x ) {
Vector( std::initializer_list<value_type> list ){
Vector( std::initializer_list<traits::named_object<value_type>> list ){

Vector( const SugarVectorExpression<eT, Expr>& other ) {
Vector( const LazyVector<RT,Expr>& other ) {
Vector( const GenericProxy<Proxy>& proxy ){
```

11 constructors in Rcpp

NumericVector constructors

~~Vector(const int& siz, stored_type (*gen)(U1), const U1& u1)~~ -> replicate

```
double fun( int power ){
    return pow(2.0, power) ;
}

// Rcpp
NumericVector x(10, fun, 10.0) ;


// Rcpp11
NumericVector x = replicate(10, fun, 10.0) ;
```

NumericVector constructors

~~Vector(InputIterator first, InputIterator last)~~ -> import

```
std::vector<double> v = /* ... */ ;

// Rcpp
NumericVector x(v.begin(), v.end()) ;
```

```
// Rcpp11
NumericVector x = import(begin(v), end(v)) ;
NumericVector x = import(v) ;
```

Some examples

NumericVector ctors

```
// C++ uniform initialization
NumericVector x = {1.0, 2.0, 3.0} ;

// init with given size
NumericVector y( 2 ) ; // data is not initialized
NumericVector y( 2, 3.0 ) ; // initialize all to 3.0

// fuse several vectors (similar to c in R)
NumericVector z = fuse(x, y) ;
NumericVector z = fuse(x, y, 3.0) ;
```

create

```
// old school NumericVector::create (Rcpp style).
NumericVector x = NumericVector::create(1.0, 2.0, 3.0) ;
NumericVector x = NumericVector::create(
   _["a"] = 1.0, _["b"] = 2.0, _["c"] = 3.0
) ;
```

```
// create as a free function. Rcpp11 style
NumericVector x = create(1.0, 2.0, 3.0) ;
NumericVector x = create(
   _["a"] = 1.0, _["b"] = 2.0, _["c"] = 3.0
) ;
```

sapply + lambda

sapply + lambda

```r
x <- 1:3
sapply( x, function(d){
  exp(d*2) ;
})
```

```
#include <Rcpp11>

// [[export]]
NumericVector foo( NumericVector x ){
    return sapply(x, [](double d){
        return exp(d * 2) ;
    });
}
```

Rcpp11 release cycle

- Available from CRAN:

```
> install.packages( "Rcpp11" )
```

- Evolves quickly. Get it from github

```
> install_github( "Rcpp11/Rcpp11" )
```

- Next Release: 3.1.1 (same time as R 3.1.1)

attributes companion package

- Standalone impl of Rcpp attributes
  `[[Rcpp::export]]`, `sourceCpp`, `compileAttributes`

- Only available from github:

```
> install_github( "Rcpp11/attributes" )
```

Use Rcpp11 in your package

- Write your .cpp files

```
#include <Rcpp11>

// [[export]]
void foo( … ) { … } // your code
```

- Express dep on C++11 and Rcpp11 headers

```
SystemRequirements: C++11
LinkingTo: Rcpp11
```

- Generate the boiler plate

```
library(attributes)
compileAttributes( "mypkg" )
```

Header only

- No runtime dependency
- Snapshot the code base
- Better control for package authors

error handling

C level try/catch

Internal C level try catch

- Problem: evaluate an R call that might error (long jump)

- Rcpp solution: wrap the call in R's `tryCatch`

- Rcpp11 ~~hack~~ solution:
  - Leverage `R_ToplevelExec` / mess with the context
  - New problem. `R_ToplevelExec` expectations

Internal C level try catch

```
SEXP res ;
try_catch( [&](){
   // some unsafe C code that might JUMP
   res = Rf_eval(expr, env) ;
}) ;
return res ;
```

Internal C level try catch

```
/* R_ToplevelExec - call fun(data) within a top level context to
   insure that this functin cannot be left by a LONGJMP.  R errors in
   the call to fun will result in a jump to top level. The return
   value is TRUE if fun returns normally, FALSE if it results in a
   jump to top level. */
```

```
Rboolean R_ToplevelExec(void (*fun)(void *), void *data)
```

pointer to a function
taking data as void*

Data

Internal C level try catch

The function templated by the real function to call

Data, also parameterise by Fun

```
template <typename Fun>
void try_catch( Fun fun ) {
    typedef std::pair<Fun*, SEXP&> Pair ;

    SEXP return_value ;

    Pair pair = std::make_pair(&fun, std::ref(return_value)) ;

    bool ok = R_ToplevelExec( &internal::try_catch_helper<Fun>, &pair ) ;

    if( !ok ){

      SEXP condition = VECTOR_ELT(return_value,0) ;

      if( Rf_isNull(condition) ){
          stop("eval error : %s", R_curErrorBuf()) ;
      } else {
          Shield<SEXP> msg = Rf_eval( Rf_lang2( Rf_install( "conditionMessage"),  condition ), R_GlobalEnv )
;
          stop("eval error : %s", CHAR(STRING_ELT(msg, 0)) ) ;
      }
    }
}
```

Internal C level try catch

```
template <typename Fun>
void try_catch_helper( void* data ){
    typedef std::pair<Fun*, SEXP&> Pair ;
    Pair* pair = reinterpret_cast<Pair*>(data) ;

    RCNTXT* ctx    = (RCNTXT*) R_GlobalContext ;
    ctx->callflag  = CTXT_FUNCTION ;

    // first call to .addCondHands to add a handler
    SEXP args = pairlist(
        Rf_mkString("error"),
        Rf_allocVector(VECSXP,1),
        ctx->cloenv,
        ctx->cloenv,
        R_FalseValue
    ) ;

    SEXP symb = Rf_install(".addCondHands") ;
    SEXP ifun = INTERNAL( symb ) ;
    PRIMFUN(ifun)(symb, ifun, args, R_GlobalEnv );

    // call it a second time to get the current R_HandlerStack
    CAR(args) = R_NilValue ;
    SEXP value = PRIMFUN(ifun)(symb, ifun, args, R_GlobalEnv ) ;
    pair->second = VECTOR_ELT(CAR(value),4) ;

    Fun& fun = *reinterpret_cast<Fun*>(pair->first) ;
    fun() ;

}
```

the actual function passed to R_ToplevelExec

hack the contexts

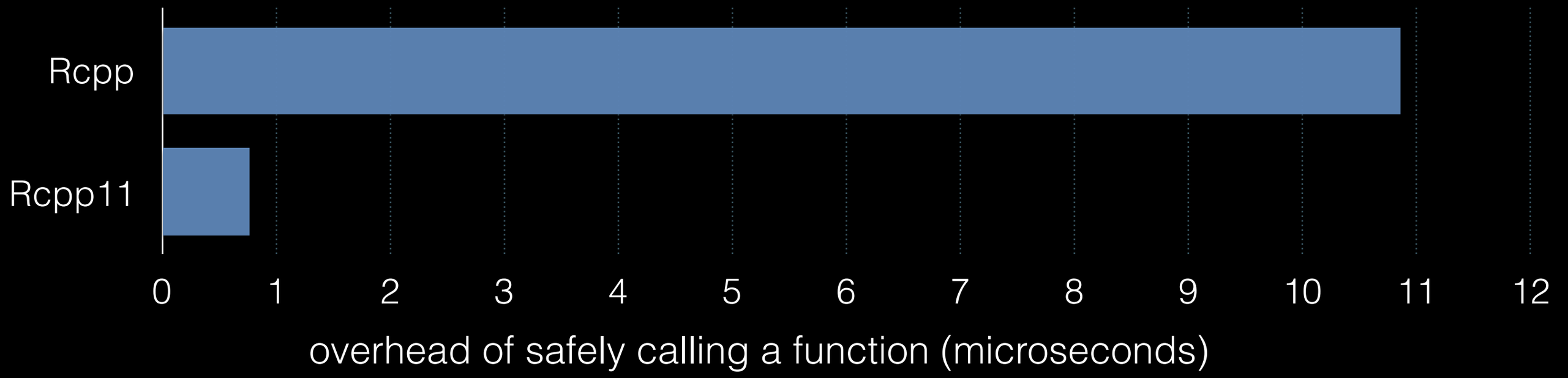
Finally calling the real function

Internal C level try catch

Nice syntax :

```
SEXP res ;
try_catch( [&](){
    // some unsafe C code that might JUMP
    res = Rf_eval(expr, env) ;
}) ;
return res ;
```

Better performance than Rcpp's solution



Internal C level try catch

- Problem: evaluate an R call that might error
- Rcpp solution: wrap the call in R's tryCatch
- Rcpp11 hack: leverage R_ToplevelExec / mess with the context
- Question: Can you help ? Please.

R_PreserveObject

R_ReleaseObject

R_PreserveObject / R_ReleaseObject

```
/* This code keeps a list of objects which are not assigned to variables
   but which are required to persist across garbage collections.  The
   objects are registered with R_PreserveObject and deregistered with
   R_ReleaseObject. */

void R_PreserveObject(SEXP object)
{
    R_PreciousList = CONS(object, R_PreciousList);
}

static SEXP RecursiveRelease(SEXP object, SEXP list)
{
    if (!isNull(list)) {
    if (object == CAR(list))
        return CDR(list);
    else
        CDR(list) = RecursiveRelease(object, CDR(list));
    }
    return list;
}

void R_ReleaseObject(SEXP object)
{
    R_PreciousList =  RecursiveRelease(object, R_PreciousList);
}
```

Questions ?

Romain François

romain@r-enthusiasts.com @romain_francois