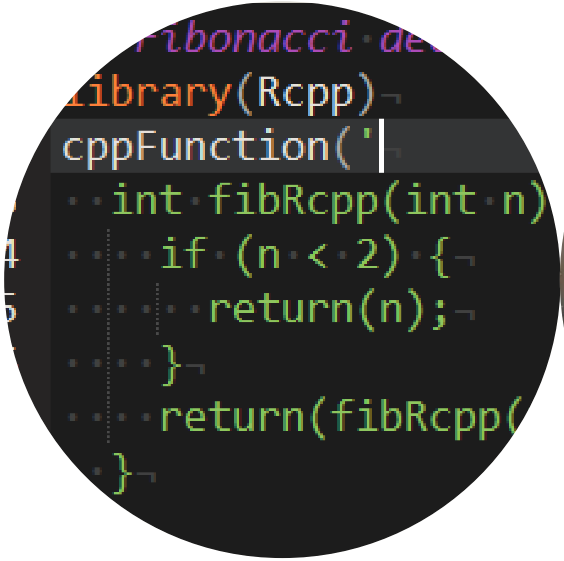
# Calling C++ code from R using Rcpp

Johannes Kruisselbrink

Biometris, Wageningen University & Research

January 16, 2018



```
Fibonacci de
library(Rcpp)
cppFunction('
  int fibRcpp(int n)
    if (n < 2) {
      return(n);
    }
    return(fibRcpp(
  }
```

# Contents

- About Rcpp
- Basics and some simple examples
- Use of Rcpp the R package Kohonen

# About Rcpp?

What is Rcpp?

- An R package that allows you to include C++ code in R

Why Rcpp?

- To make your R code faster
- (… or to include already existing C/C++ code)

Made by…

- Dirk Eddelbuettel and Romain Francois

# Our first example: Fibonacci in R

```r
fibR <- function(n) {
  if (n < 2) return(n)
  return(fibR(n-1) + fibR(n-2))
}
```

```r
sapply(1:10, fibR)
# [1]  1  1  2  3  5  8 13 21 34 55
```

Source: http://dirk.eddelbuettel.com/papers/rcpp_sydney-rug_jul2013.pdf

# Our first example: Fibonacci in Rcpp

```
library(Rcpp)
cppFunction('
  int fibRcpp(int n) {
    if (n < 2) {
      return(n);
    }
    return(fibRcpp(n-1) + fibRcpp(n-2));
  }
')
```

```
sapply(1:10, fibRcpp)
# [1]  1  1  2  3  5  8 13 21 34 55
```

Source: http://dirk.eddelbuettel.com/papers/rcpp_sydney-rug_jul2013.pdf

# Fibonacci in R/Rcpp – timings (1)

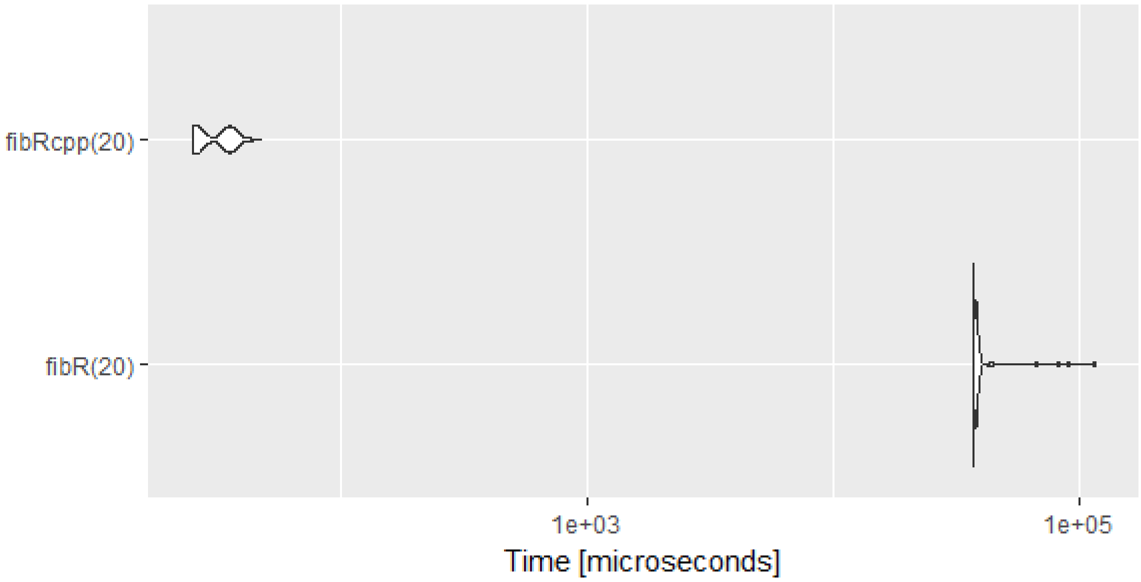
```
library(microbenchmark)
timings <- microbenchmark(
  fibR(20),
  fibRcpp(20)
)
```

```
print(timings)
# Unit: microseconds
#         expr       min       lq        mean    median         uq       max neval cld
#     fibR(20) 37680.979 38879.05 40855.88913 39454.136 41137.8440 70167.926   100   b
#  fibRcpp(20)    24.997    25.60    31.45472    29.214    37.0445    41.562   100  a
```

Source: [http://dirk.eddelbuettel.com/papers/rcpp_sydney-rug_jul2013.pdf](http://dirk.eddelbuettel.com/papers/rcpp_sydney-rug_jul2013.pdf)

# Fibonacci in R/Rcpp – timings (2)

```
library(ggplot2)
autoplot(timings)
```



Source: http://dirk.eddelbuettel.com/papers/rcpp_sydney-rug_jul2013.pdf

# Ways to call Rcpp

- **sourceCpp:** imports all functions marked with the Rcpp::export attribute of the specified C++ file or source code

- **cppFunction:** imports a single R function from the provided C++ source code

- **evalCpp:** evaluates a C++ expression

# Example sourceCpp

A file containing the C++ code; use Rcpp::export to specify the functions that are to be imported in R.

File: DoubleMeRcpp.cpp

```
#include <Rcpp.h>
// [[Rcpp::export]]
int doubleMeRcpp(int x) {
  return 2 * x;
}
```

```
sourceCpp("DoubleMeRcpp.cpp")
doubleMeRcpp(5)
# 10
```

# Example cppFunction

One function declaration that is assumed to be the C++ function that is to be imported in R.

```
cppFunction('
  int doubleMeRcpp(int x) {
    return 2 * x;
  }
')
doubleMeRcpp(5)
# 10
```

# Example evalCpp

No function arguments here, just evaluate.

```
evalCpp("5 * 5")
# [1] 25
```

# Rcpp data types

Rcpp provides C++ wrappers for R data structures:

- IntegerVector, NumericVector, LogicalVector, CharacterVector
- List, DataFrame
- Named, Dimension
- IntegerMatrix, NumericMatrix
- Function
- Environment

# Example NumericVector - CumSum

```
cppFunction('
 NumericVector cumSumRcpp(NumericVector x) {
    double acc = 0;
    NumericVector res(x.size());
    for(int i = 0; i < x.size(); i++){
      acc += x[i];
      res[i] = acc;
    }
    return res;
  }
')
```

```
cumSumRcpp(1:10)
# [1]  1  3  6 10 15 21 28 36 45 55
```

# Example NumericMatrix - RowMeans

```
cppFunction('
  NumericVector rowMeansRcpp(NumericMatrix& x) {
    int nRows = x.nrow();
    int nCols = x.ncol();
    NumericVector out(nRows);
    for (int i = 0; i < nRows; i++) {
      double sum = 0;
      for (int j = 0; j < nCols; j++) {
        sum += x(i, j);
      }
      out[i] = sum / nCols;
    }
    return out;
  }
')
```

```
set.seed(1)
rowMeansRcpp(matrix(runif(100), ncol=20))
# [1] 0.5301390 0.5045979 0.5150565 0.5092252 0.5302167
```

# Example List

```
cppFunction('
  List rcppSquare(const NumericVector &x) {
    NumericVector vec(x.size());
    for (int i = 0; i < x.size(); i++) {
      vec[i] = x[i] * x[i];
    }
    return List::create(
      Named("original") = x,
      Named("result") = vec
    );
  }
')
```

```
rcppSquare(1:10)
# $original
#  [1]  1  2  3  4  5  6  7  8  9 10
#
# $result
#  [1]   1   4   9  16  25  36  49  64  81 100
```

# Calling R functions from C++

```
cppFunction('
  double callFunctionRcpp(NumericVector x, Function f) {
    double result = as<double>(f(x));
    return result;
  }
')
```

```
callFunctionRcpp(1:10, sum)
# [1] 55
```

```
callFunction(1:10, mean)
# [1] 5.5
```

# Printing to R console – Rcout and Rcerr

```
cppFunction('
  void helloWorld() {
    Rcout << "Hello World!" << std::endl;
  }
')
```

```
helloWorld()
# Hello World!
```

```
cppFunction('
  void helloError() {
    Rcerr << "Hello error!" << std::endl;
  }
')
```

```
helloError()
# Hello error!
```

# Rcpp’s (syntactic) sugar

- A collection of C++ routines that mirror frequently used R functions
    - +, -, *, /
    - <, >, <=, >=, ==, !=
    - any, all
    - is_na, is_true, is_false
    - seq, seq_along, seq_len
    - ifelse
    - log, sin, cos, . . .
    - distribution functions dnorm, qnorm, . . .
    - lapply, sapply
    - …

# Example Rcpp sugar - ifelse

```
cppFunction('
  NumericVector rcppIfElse(NumericVector x, NumericVector y) {
    return ifelse(x < y, x * x, -(y * y));
  }
')
```

```
rcppIfElse(1:10, rep(5,10))
# [1]   1   4   9  16 -25 -25 -25 -25 -25 -25
```

# Rcpp and STL

- C++ standard library (STL) provides a set of generic algorithms and data structures

- Rcpp knows how to convert many of STL data structures to their R equivalents

- Rcpp integrates well with the STL algorithms

# Example Rcpp and STL

```
cppFunction('
  std::vector<double> logRcpp(std::vector<double> x) {
    std::transform(x.begin(), x.end(), x.begin(), ::log);
    return x;
  }
')
```

```
logRcpp(1:5)
# [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

# .C versus .Call versus Rcpp

## C

- Simplest interface
- Useful if you have standard C code
- Does not work directly on the R objects
- Objects are copied from R and passed as a pointer
- No way of creating new objects in C and pass them back to R

## .Call

- Allows you to create objects and pass them back to R
- Faster than .C because it does not copy objects from R
- Works with R objects in the form of S expressions (SEXPs)
- Requires proper bookkeeping to work with the R objects
- Steep learning curve

## Rcpp

- Still uses the .Call interface under the hood
- Rcpp generates extern "C" and .Call wrappers for C++ functions
- Much simpler C/C++ code
- Easier to learn

# Example .C vs. .Call versus Rcpp

```
void doubleMeC(int* x) {
  *x = *x + *x;
}
```

```
#include <R.h>
#include <Rdefines.h>
SEXP doubleMeCall(SEXP x) {
  SEXP result;
  PROTECT(result = NEW_INTEGER(1));
  INTEGER(result)[0] = INTEGER(x)[0] * 2;
  UNPROTECT(1);
}
```

```
#include <Rcpp.h>
// [[Rcpp::export]]
int doubleMeRcpp(int x) {
  return 2 * x;
}
```

# Rcpp – Use verbose to see generated code

```
cppFunction('
  int doubleMeRcpp(int x) {
    return 2 * x;
  }
', verbose=T)
```

```
Generated extern "C" functions
--------------------------------------------------------


#include <Rcpp.h>
// doubleMeRcpp
int doubleMeRcpp(int x);
RcppExport SEXP sourceCpp_3_doubleMeRcpp(SEXP xSEXP) {
BEGIN_RCPP
    Rcpp::RObject rcpp_result_gen;
    Rcpp::RNGScope rcpp_rngScope_gen;
    Rcpp::traits::input_parameter< int >::type x(xSEXP);
    rcpp_result_gen = Rcpp::wrap(doubleMeRcpp(x));
    return rcpp_result_gen;
END_RCPP
}
```

# Rcpp in Kohonen v3 package

- Kohonen: R package containing several self-organizing map implementations
- Kohonen v2 used C code for the training and mapping functions and .C to call it
- Objectives:
    - Improve memory usage
    - Improve calculation efficiency and add parallelization
    - Allow for user defined distance functions

- Wehrens, Lutgarde, and Buydens. “Self- and Super-organizing Maps in R: The kohonen Package“. Journal of Statistical Software, Volume 21, Issue 5 (8 October 2007)
- Wehrens and Kruisselbrink. “Flexible Self-Organising Maps in kohonen v3.0”. Journal of Statistical Software (to appear).

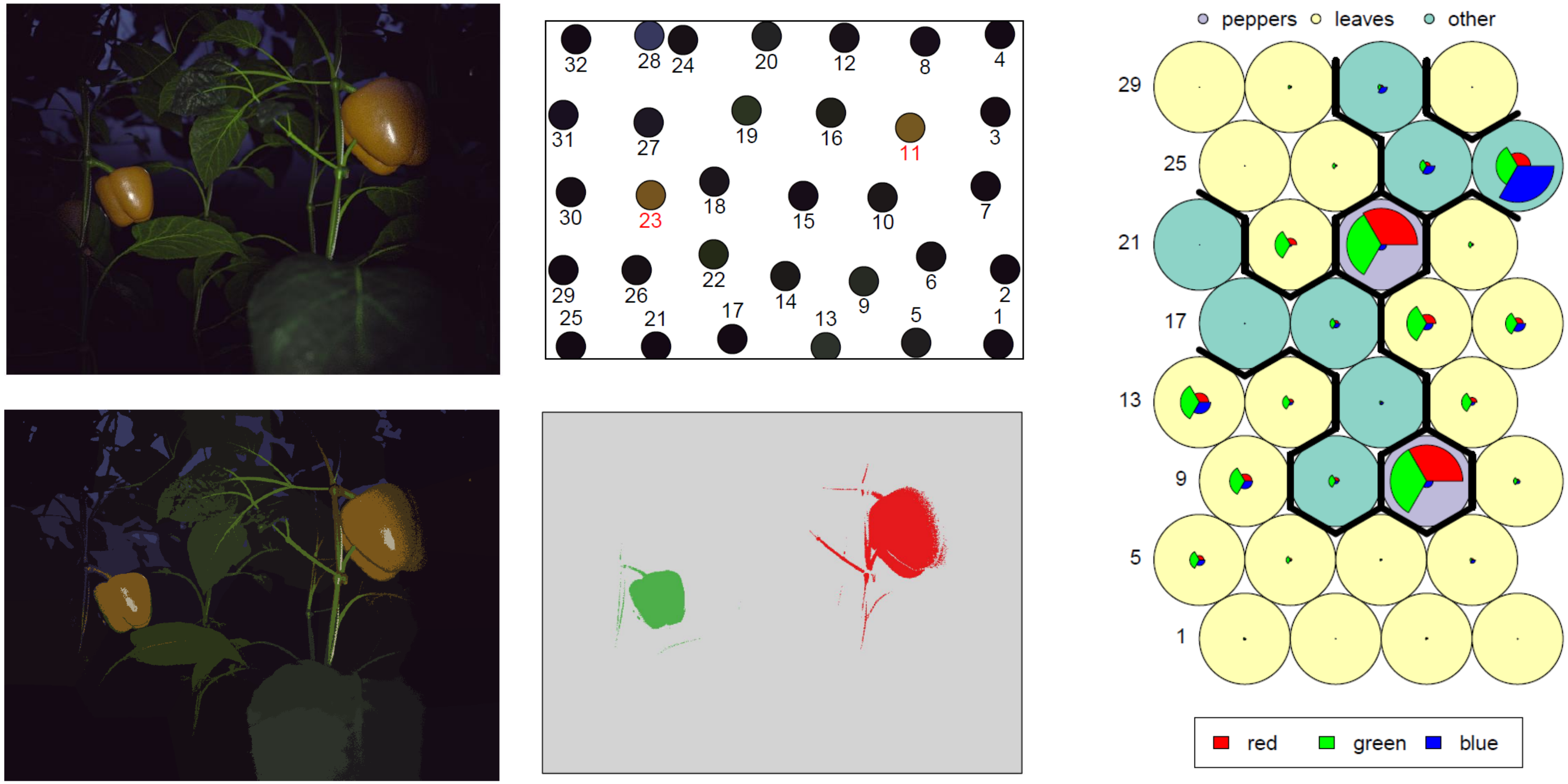
# Self-Organizing Maps (SOMs)

- Unsupervised learning technique that projects multi-dimensional data onto a two-dimensional map in which topological properties are maintained
- SOM: a two-dimensional map of nodes (units) organized in some grid in which each unit is associated with a codebook vector that resides in the domain of the data samples
- Suitable for clustering / identification of groups of data points with similar characteristics

# Training and mapping

- Training:
  - Start: assign a random codebook vector to each unit
  - Loop: expose the SOM to the data points; for each data point, update the best-matching node and its neighbors by pulling them in the direction of the data point
- Mapping:
  - New data points are mapped to the unit of which the codebook vector is most similar to that data point

# SOMs for pepper image segmentation



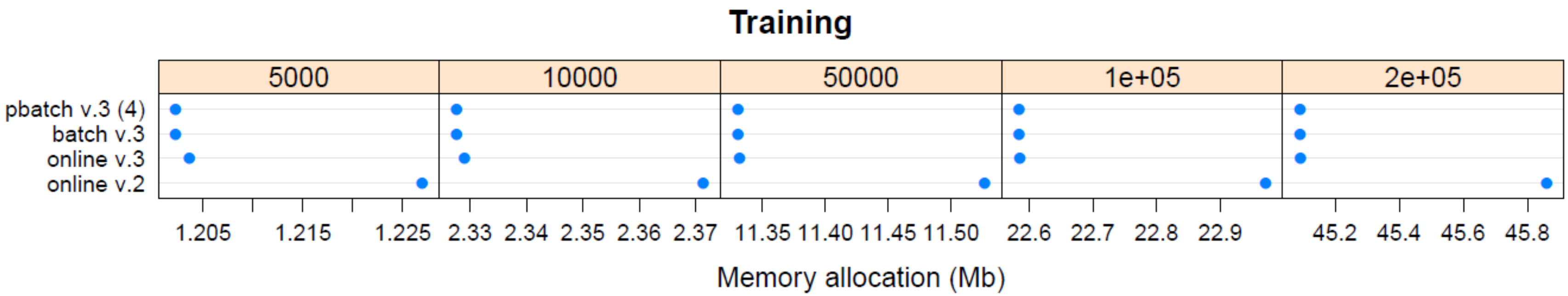
SOM trained on pixel position and RGB value

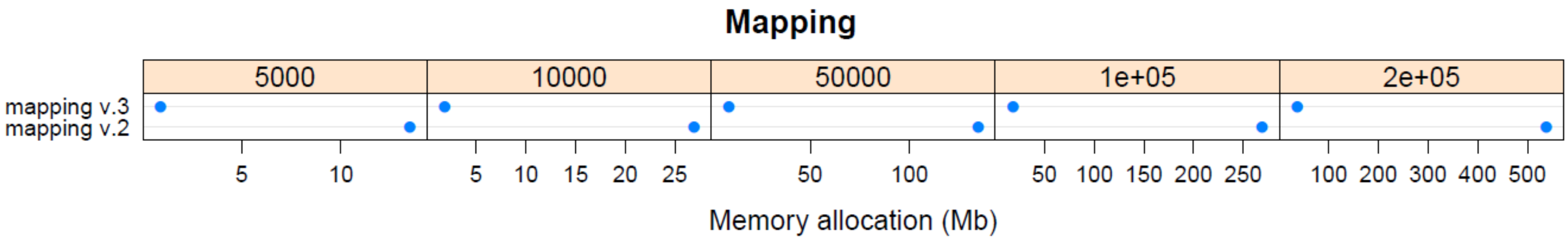
# Memory improvement

- Issue:
  - Kohonen v2 uses the .C interface
  - With .C, the R objects are copied before being passed to the C code, and copied again to an R list object when the compiled code returns
- Solution:
  - Switch to .Call or Rcpp which act on the R objects directly
  - … we choose Rcpp

# Memory improvement

Memory profiling results of different versions of the training and mapping functions on varying dataset sizes.





# Calculation speed improvement

- Issue:
  - Can we speed-up Kohonen v2?
- Answer:
  - Flip data matrices in order to get the elements of the data and codebook vectors contiguous in memory
  - Omit NA checks in distance/similarity calculations when not neede
  - Implement batch training algorithm and a parallel version of this training algorithm

# R stores matrices in column major order

```
cppFunction('
  NumericMatrix iterateMatrix(int nRow, int nCol) {
    NumericMatrix out(nRow, nCol);
    int xsize = nRow * nCol;
    for (int i = 0; i < xsize; i++) {
      out[i] = i;
    }
    return out;
  }
')
iterateMatrix(4,3)
```

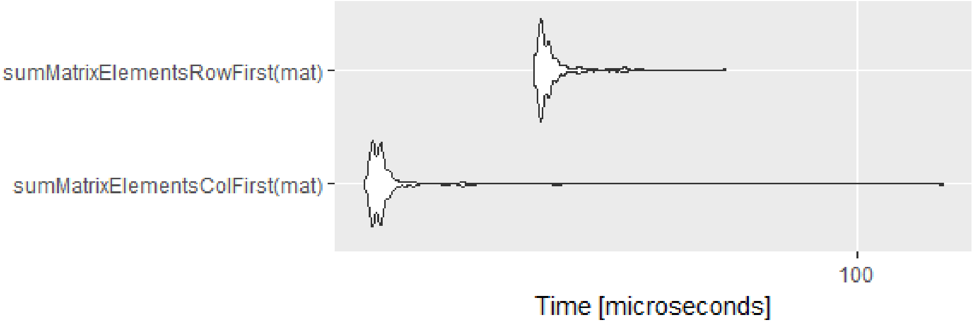
```
iterateMatrix(4,3)
#      [,1] [,2] [,3]
# [1,]    0    4    8
# [2,]    1    5    9
# [3,]    2    6   10
# [4,]    3    7   11
```

# Row first versus column first

```
mat <- matrix(runif(100000), ncol=500)

library(microbenchmark)
timings <- microbenchmark(
  sumMatrixElementsColFirst(mat),
  sumMatrixElementsRowFirst(mat)
)
```

```
print(timings)
# Unit: microseconds
#                            expr    min     lq     mean median     uq      max neval cld
#  sumMatrixElementsColFirst(mat) 80.112 80.414 93.60805 80.715 81.016 1358.898   100   a
#  sumMatrixElementsRowFirst(mat) 86.437 86.738 87.80750 87.039 87.341  128.601   100   a
```



# BTW – alternative to previous examples

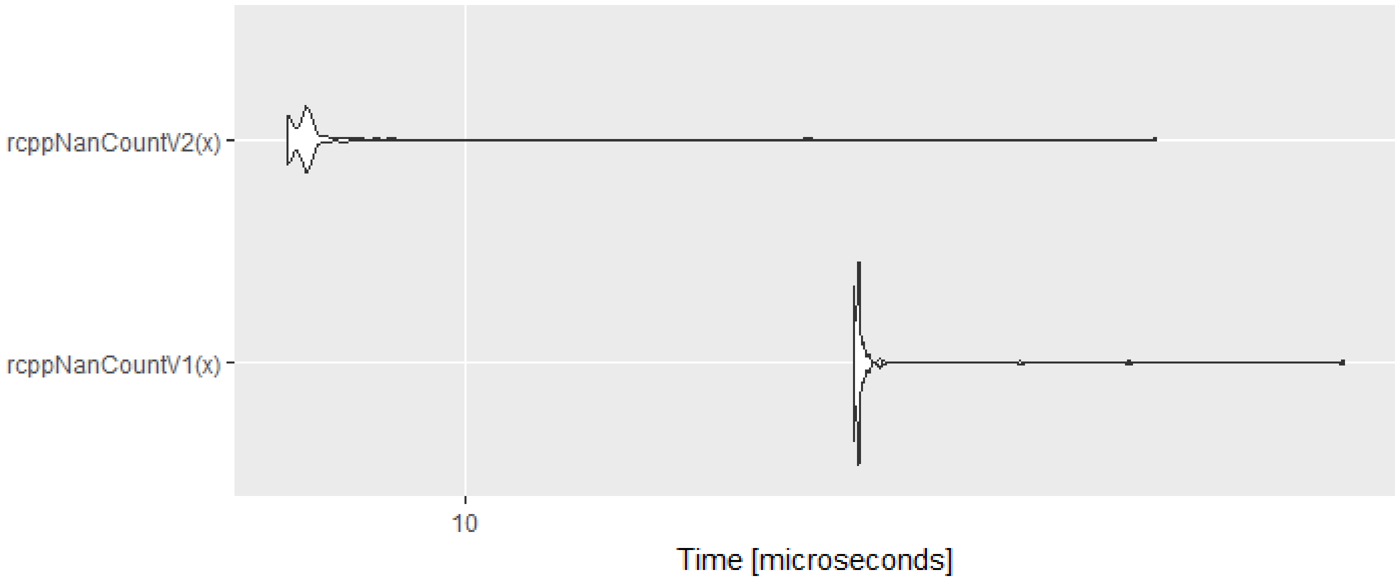
```
cppFunction('
  double sumMatrixElementsFlat(NumericMatrix& x) {
    double sum = 0;
    int xsize = x.nrow() * x.ncol();
    for (int i = 0; i < xsize; i++) {
      sum += x[i];
    }
    return sum;
  }
')
```

# Check for NaN - A small difference …

```
cppFunction('
  int rcppNanCountV1(NumericVector x) {
    int count = 0;
    int size = x.size();
    for (int i = 0; i < size; i++) {
      if (ISNAN(x[i])) {
        count++;
      }
    }
    return count;
  }
')
```

```
cppFunction('
  int rcppNanCountV2(NumericVector x) {
    int count = 0;
    int size = x.size();
    for (int i = 0; i < size; i++) {
      if (std::isnan(x[i])) {
        count++;
      }
    }
    return count;
  }
')
```

# Check for NaN - A small difference …



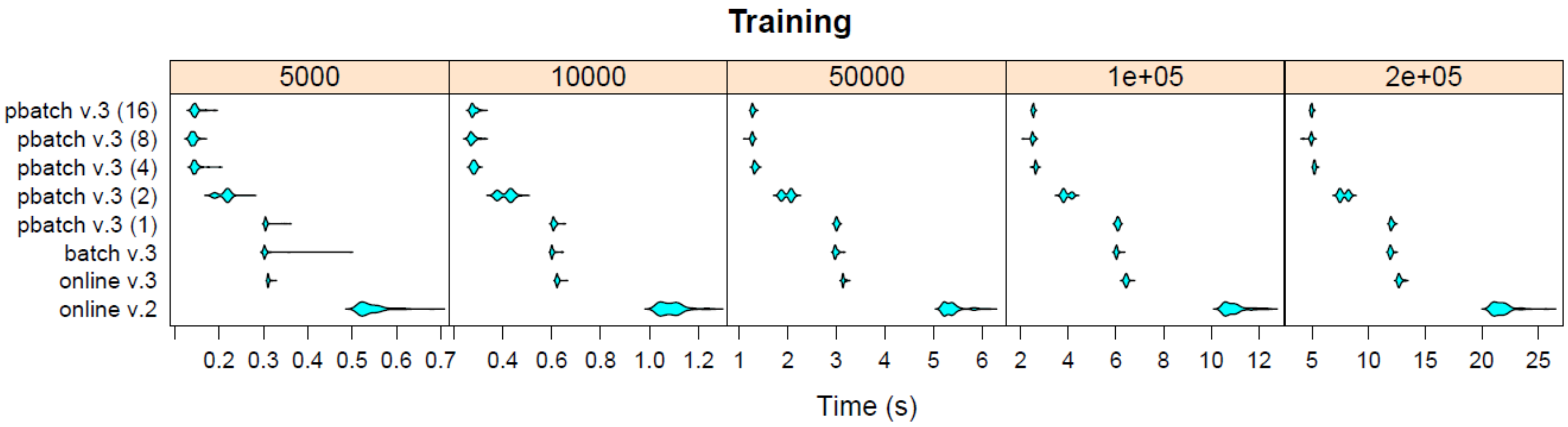
```
# Unit: microseconds
#               expr    min     lq     mean median     uq    max neval cld
#  rcppNanCountV1(x) 24.395 24.396 25.51866 24.696 24.697 75.595   100   b
#  rcppNanCountV2(x)  6.626  6.626  7.45723  6.927  6.927 49.091   100  a
```

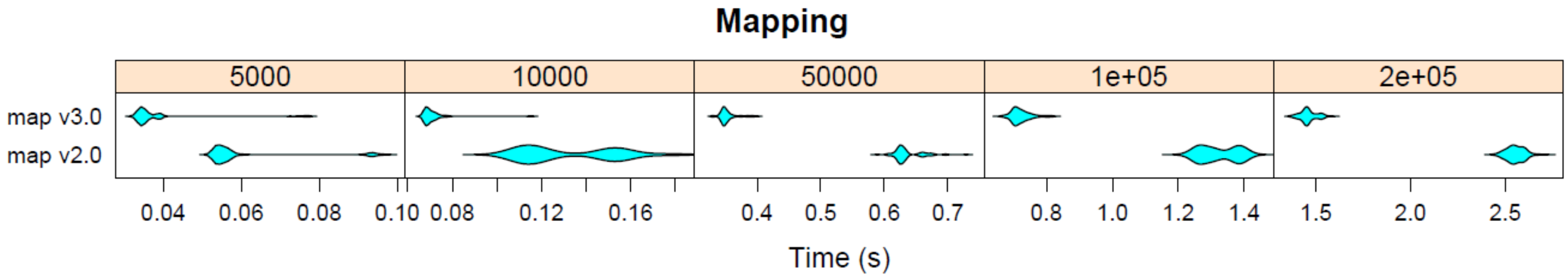
# Batch implementation and parallelisation

- Batch training algorithm:
  - Datasets are split up in subsets that are trained separately
  - Training results of subsets are merged after each epoch
- Use Rcpp + OpenMP for parallelization (alternatively, RcppParallel could be used for parallelization)

# Calculation speed improvement

Timing results of different versions of the training and mapping on varying dataset sizes.





# User-defined distance functions

- Issue:
  - Default distance function is Euclidean distance
  - Can we enable user to specify custom distance/dissimilarity functions (as C++ code)?
- Solution:
  - Encapsulate function definitions in Rcpp XPtrs
  - Inspiration: http://gallery.rcpp.org/articles/passing-cpp-function-pointers/

# Example defining and using XPtrs

```
code <- '
  #include <Rcpp.h>
  typedef double (*FuncionPtr)(double);

  double myCustomFunction(double value) {
    return value * value;
  }

  // [[Rcpp::export]]
  Rcpp::XPtr<FuncionPtr> myFunctionPtr() {
    return Rcpp::XPtr<FuncionPtr>(new FuncionPtr(&myCustomFunction));
  }

  // [[Rcpp::export]]
  double evaluate(SEXP xpsexp, double x) {
    Rcpp::XPtr<FuncionPtr> distanceFunctionXPtr(xpsexp);
    FuncionPtr fun = *distanceFunctionXPtr;
    return fun(x);
  }
'
sourceCpp(code = code)
```

```
evaluate(myFunctionPtr(), 6)
# [1] 36
```

# Bray-Curtis dissimilarity

```
BCcode <- '
  #include <Rcpp.h>
  typedef double (*DistanceFunctionPtr)(double *, double *, int, int);

  double brayCurtisDissim(double *data, double *codes, int n, int nNA) {
    if (nNA > 0) return NA_REAL;
    double num = 0.0, denom = 0.0;
    for (int i = 0; i < n; i++) {
        num += std::abs(data[i] - codes[i]);
        denom += data[i] + codes[i];
    }
    return num/denom;
  }

  // [[Rcpp::export]]
  Rcpp::XPtr<DistanceFunctionPtr> BrayCurtis() {
    return Rcpp::XPtr<DistanceFunctionPtr>(new DistanceFunctionPtr(&brayCurtisDissim));
  }
'
```

# Kudos to the Rcpp development team

- Very active development
- 1000+ CRAN packages use Rcpp
- Good package documentation
- Many tutorials available on the web
- Growing collection of questions/answers on stackoverflow
- Good support from the rcpp-devel mailing list

# Some good sources

- Rcpp quick reference guide: http://dirk.eddelbuettel.com/code/rcpp/Rcpp-quickref.pdf
- Rcpp tutorial by Dirk Eddelbuettel: http://dirk.eddelbuettel.com/papers/rcpp_workshop_introduction_user2012.pdf
- Guide by Hadley Wickham: http://adv-r.had.co.nz/Rcpp.html
- Rcpp gallery: http://gallery.rcpp.org/

# Thank you

```
bonacci decl
brary(Rcpp)
ppFunction('
  int fibRcpp(int n) {
    if (n < 2) {
      return(n);
    }
    return(fibRcpp(n-
}
```