Combining R and C++: Using the Rcpp package

Thijs Janzen

```cpp

#include <Rcpp.h>

// [[Rcpp::export]]
double calc_square(double input) {
    double output = input * input;
    return output;
}

```

Who am I?

PhD University of Groningen

Community Assembly
Diversification
Approximate Bayesian Computation

Post Doc
Max Planck Institute for Evolutionary Biology
Plön, Germany

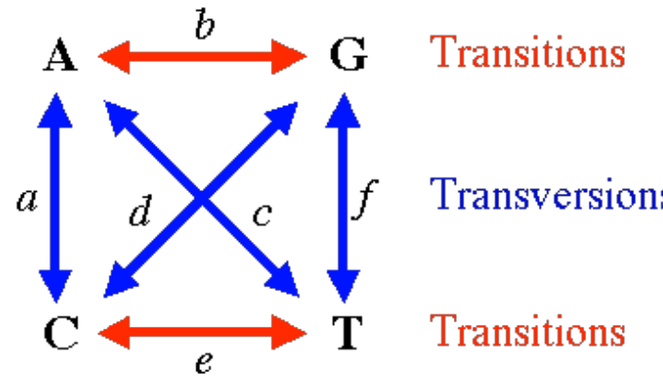
Theory of junctions
Missegregation

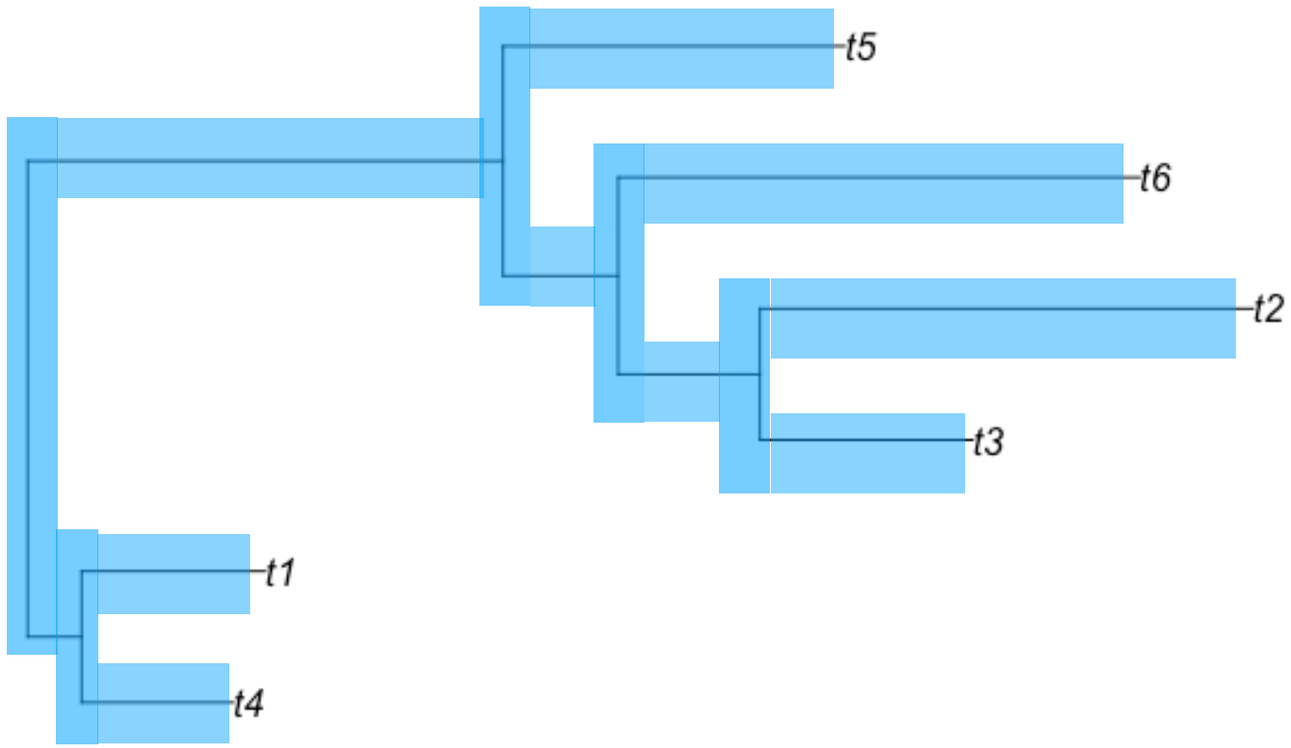
Post Doc
Carl von Ossietzky University
Oldenburg, Germany

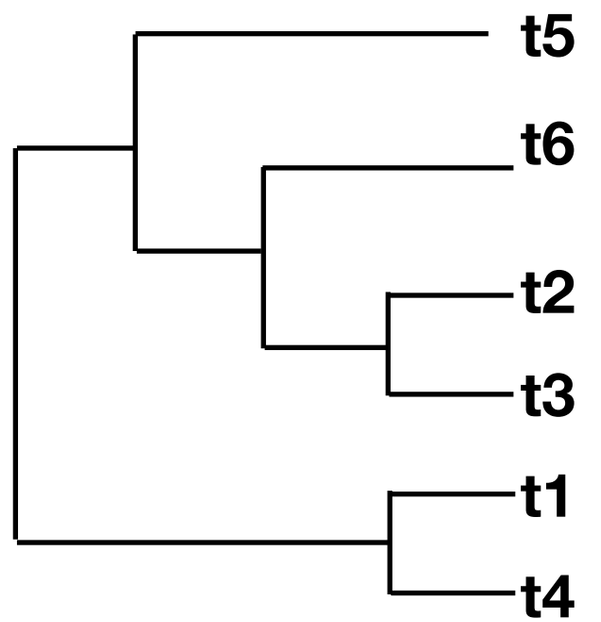
Macrogenomic patterns of diversification
Theory of junctions
Genetic basis of sex determination

Plans

- Substitution model variation

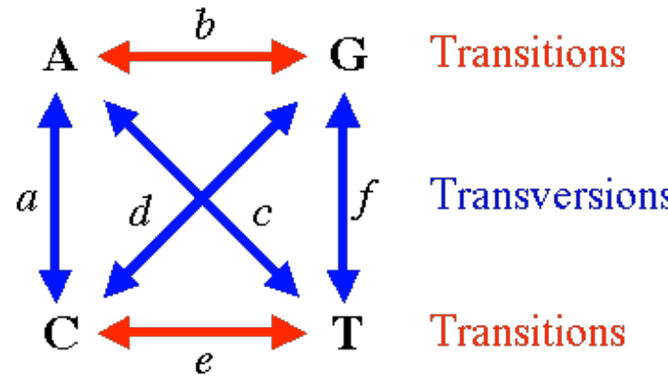


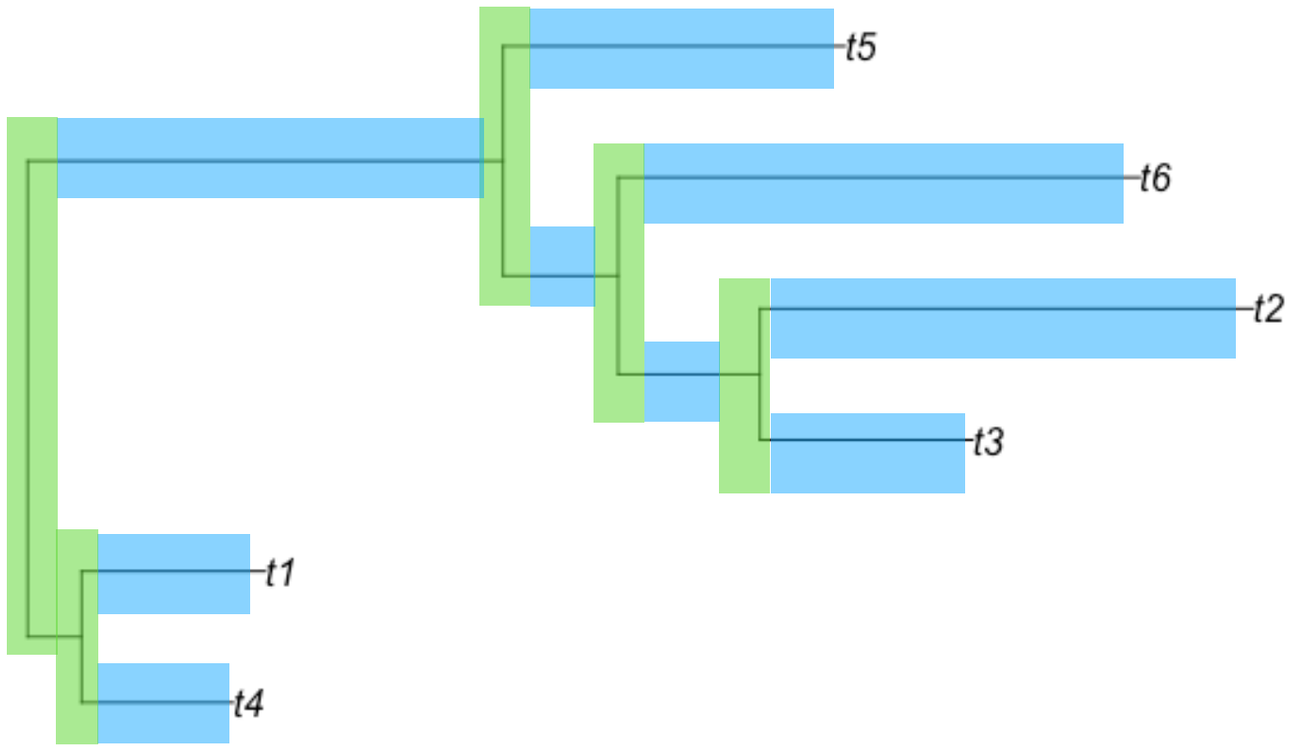




Plans

- Substitution model variation





?

Combining R and C++: Using the Rcpp package

R and C++

- C++
  - Any-level language
  - Compiled language —> Fast
  - Strict typesystem

- R
  - Easily installable packages
  - Built-in visualisation (plotting etc)
  - Powerful data analysis tools
  - Easy to apply across common platforms

Integrating R and C++

- The best of both worlds:
  - Speed of C++
  - Cross compatibility of R
  - Visualisation options of R
- In fact, R *is written in C++*!

- And usage of C code within R has been around for ages:

Two functions are available for establishing condition handlers from within C code:

```
#include <Rinternals.h>

SEXP R_tryCatchError(SEXP (*fun)(void *data), void *data,
                     SEXP (*hndlr)(SEXP cond, void *hdata), void *hdata);

SEXP R_tryCatch(SEXP (*fun)(void *data), void *data,
                SEXP,
                SEXP (*hndlr)(SEXP cond, void *hdata), void *hdata,
                void (*clean)(void *cdata), void *cdata);
```

`R_tryCatchError` establishes an exiting handler for conditions inheriting form class `error`.

`R_tryCatch` can be used to establish a handler for other conditions and to register a cleanup action. The conditions be passed as `fun` or `clean` if condition handling or cleanup are not needed.

These are currently implemented using the R-level `tryCatch` mechanism so are subject to some overhead.

The function `R_UnwindProtect` can be used to ensure that a cleanup action takes place on ordinary return as well a

```
SEXP R_UnwindProtect(SEXP (*fun)(void *data), void *data,
                     void (*clean)(void *data, Rboolean jump), void *cdata,
                     SEXP cont);
```

The Rcpp package

- Made easy using the R package ‘Rcpp’
  - Provides R functions to compile and link C++ code
  - Provides C++ handles to existing R functions
  - Automates communication between R and C++
  - Does all the ‘dirty’ work

Example

C++ code (“R_demonstration.cpp”)

```cpp

#include <Rcpp.h>

// [[Rcpp::export]]
double calc_square(double input) {
    double output = input * input;
    return output;
}

```

R code

Source on Save

```r
library(Rcpp)
sourceCpp("R_demonstration.cpp")
calc_square(2)

```

Passing objects

- Standard datatypes (int, float, double)
    - No need to specify, Rcpp automagically makes this work
- Vectors
    - Rcpp::NumericVector
        - Vector of floating numbers
    - Rcpp::IntegerVector
        - Vector of integers
    - Rcpp::LogicalVector
        - Vector of booleans
- Matrices
    - Rcpp::NumericMatrix / Rcpp::IntegerMatrix / Rcpp::LogicalMatrix

Rcpp::NumericVector

C++ code

```
// [[Rcpp::export]]
Rcpp::NumericVector calc_squares(Rcpp::NumericVector v) {
    Rcpp::NumericVector output(v.size());
    for(int i = 0; i < v.size(); ++i) {
        output[i] = v[i] * v[i];
    }
    return(output);
}
```

R code

```
library(Rcpp)
sourceCpp("R_demonstration.cpp")
calc_square(2)

vx <- 1:10
calc_squares(vx)
```

Rcpp::NumericVector

- Indexing starts at **0**

- New options to create vectors:

```
Rcpp::NumericVector v(3)    // v <- rep(0, 3)
Rcpp::NumericVector v(3,1)  // v <- rep(1, 3)
```

- Additional ways accessing vector:

```
int a = v[0]; // no range checking
int b = v(0); // range check: throws error if index is outside the vector
```

- Both C++ and R member functions available:

  - push_back, .size(), .begin(), .end() but also:
    sum, max, min, .length()

Rcpp::NumericMatrix

C++ code

```
using namespace Rcpp;

// [[Rcpp::export]]
NumericMatrix calc_squares_matrix(NumericMatrix m) {
    NumericMatrix output(m.nrow(), m.ncol());
    for(int i = 0; i < m.nrow(); ++i) {
        output(i, _) = calc_squares( m(i, _) );
    }
    return(output);
}
```

The '_' operator indicates usage
of all values in that index (e.g. using the entire column)

R code

```
input <- matrix(1:4, ncol=2, nrow=2)
input
calc_squares_matrix(input)
```

Rcpp allows usage of R functions within C++

- Random number generation
  - Rcpp::runif(min, max) or Rcpp::runif(N)
  - Rcpp::rpois(lambda) or Rcpp::rpois(N, lambda)
  - Rcpp::rgamma(shape) or Rcpp::rgamma(N, shape)
  - Rcpp::rnorm(mean, sd) or Rcpp::rnorm(N, mean, sd)

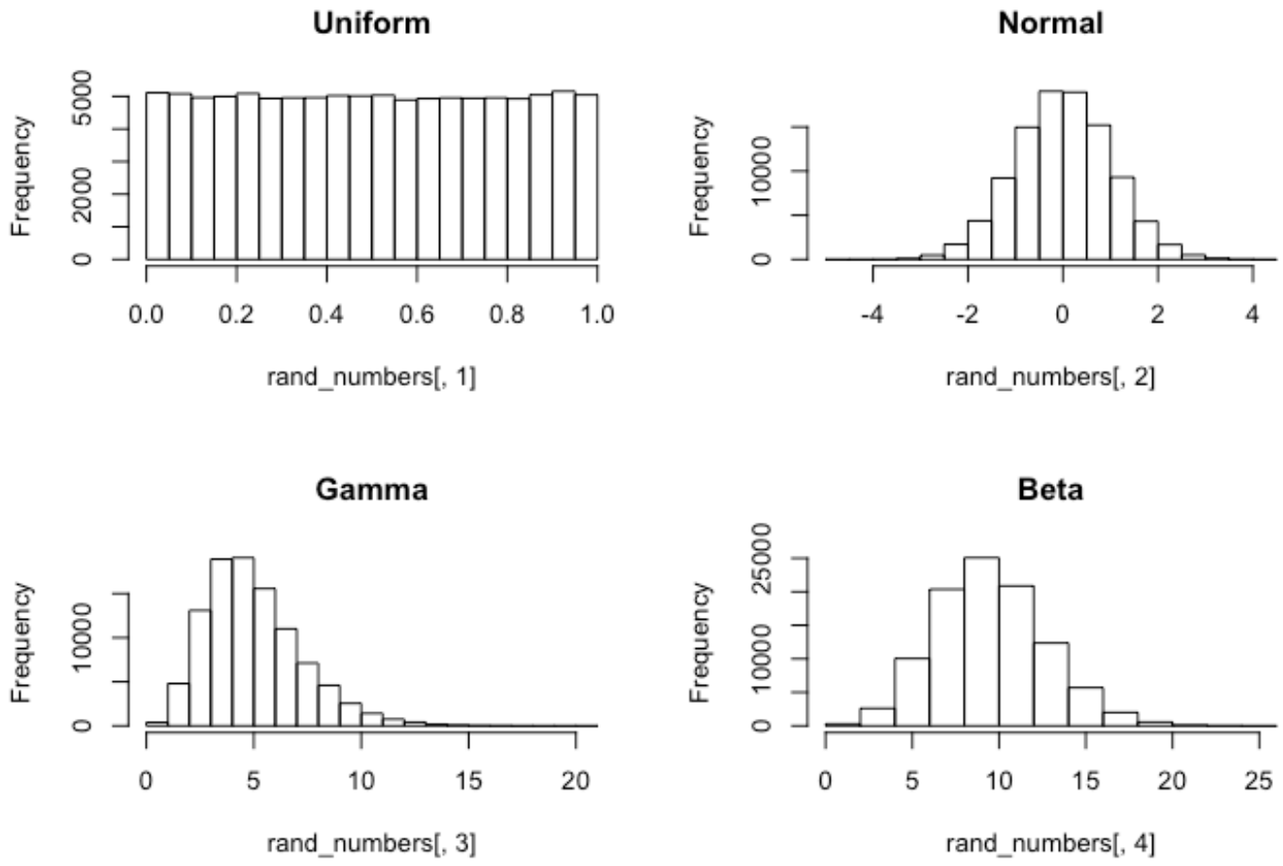
Random number generation in Rcpp

C++ code

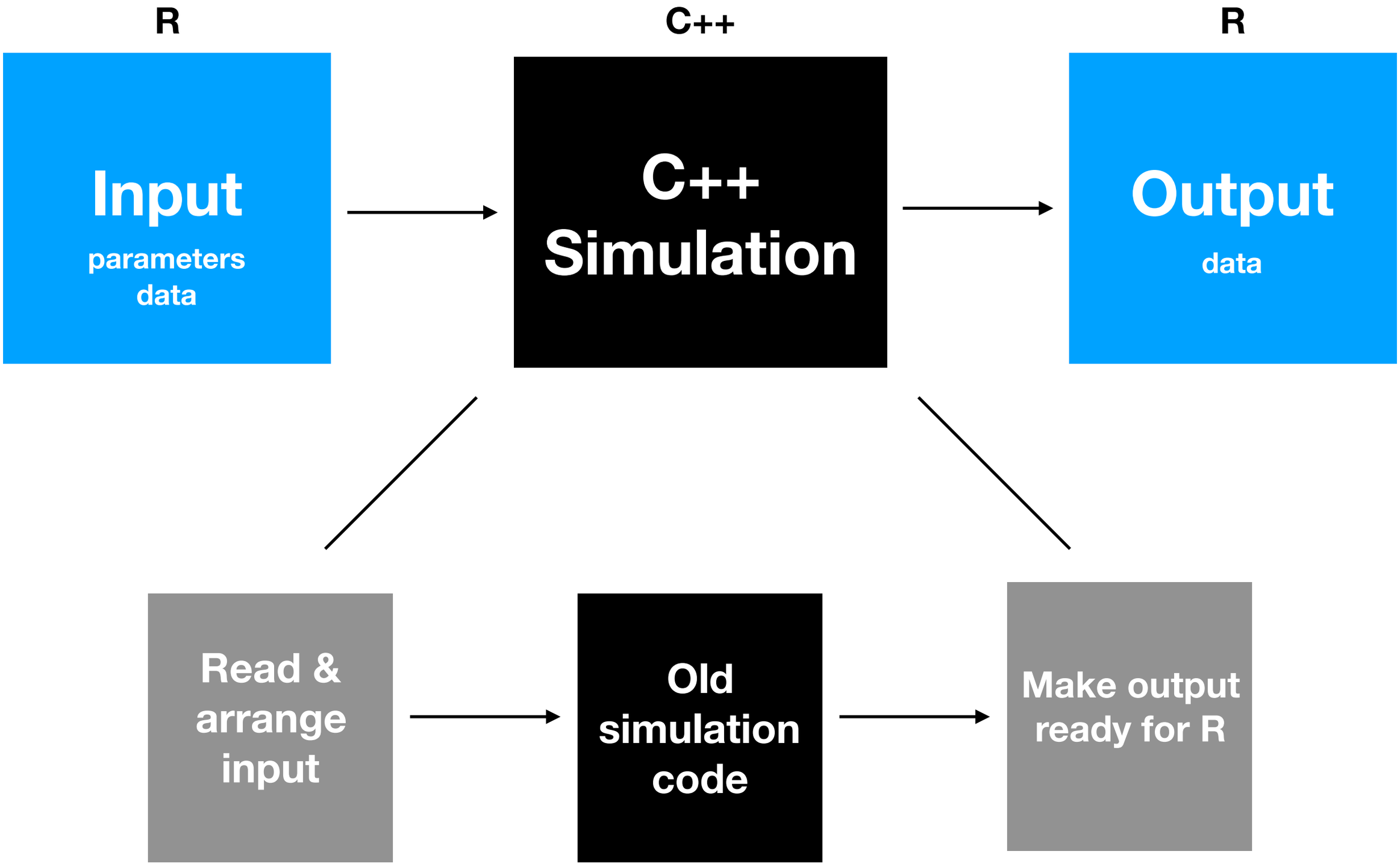
```
// [[Rcpp::export]]
// example from: http://gallery.rcpp.org/articles/random-number-generation/
NumericMatrix rngCpp(const int N) {
    NumericMatrix X(N, 4);
    X(_, 0) = runif(N); // N values in [0,1]
    X(_, 1) = rnorm(N); // N values with mu = 0, sd = 1
    X(_, 2) = rgamma(N, 5); // N values with shape = 5
    X(_, 3) = rpois(N, 10); //N values with lambda = 10
    return X;
}
```

R code

```
rand_numbers <- rngCpp(1e5)
par(mfrow=c(2,2))
hist(rand_numbers[,1], main = "Uniform")
hist(rand_numbers[,2], main = "Normal")
hist(rand_numbers[,3], main = "Gamma")
hist(rand_numbers[,4], main = "Poisson")
```



How to make your C++ code usable in R



Example simulation

- A-sexual population, non-overlapping generations
- Individuals have:
  - ecological trait
  - fitness (higher fitness if trait is closer to 0)
- Probability of reproduction proportional to fitness
- Small probability of mutation

Making the most out of Rcpp

- Use Rcpp within a package
  - multiple c++ files
  - easier use of C++11
- Use Armadillo library
  - improved matrix- and vector operations
- Allow the user to interrupt operation (helps in case of crashes)
  - Rcpp::checkUserInterrupt()

Simulations using Rcpp

https://github.com/thijsjanzen/junctions

junctions

Individual based simulations of hybridizing populations, where the accumulation of junctions is tracked. Furthermore, mathematical equations are provided to verify simulation outcomes. Both simulations and mathematical equations are based on Janzen et al. (2018)

references

Janzen, T. , Nolte, A. W. and Traulsen, A. (2018), The breakdown of genomic ancestry blocks in hybrid lineages given a finite number of recombination sites. Evolution, 72: 735-750. https://doi.org/10.1111/evo.13436

Updates

Version 1.2: Added support for estimating the expected number of junctions for arbitrarily distributed markers.

Version 1.1: Updated random number generation for picking recombination sites. Previous implementation was limited to 6 digit precision, current precision is at least double that, minimizing the probability of recombination occur twice in the same location for an infinite chromosome.

https://github.com/thijsjanzen/GenomeAdmixR

GenomeAdmixR

Genetic admixture simulation

What is GenomeAdmixR?

A package under construction to simulate genetic admixture in relation

Demonstration GenomeAdmixR

Thijs Janzen gave a presentation demonstrating GenomeAdmixR (then of Groningen, Groningen, The Netherlands. You can watch his presenta

Mathematical calculations using Rcpp

https://github.com/thijsjanzen/GUILDS

GUILDS

The GUILDS package combines a range of sampling formulas for the unified neutral moc biodiversity. Alongside the sampling formulas, it includes methods to perform maximum sampling formulas, methods to generate data given the neutral model, and methods to e abundance distribution. Sampling formulas included in the GUILDS package are the Etie 2005), the guild sampling formula, where guilds are assumed to differ in dispersal ability sampling formula conditioned on guild size (Janzen et al. 2015).