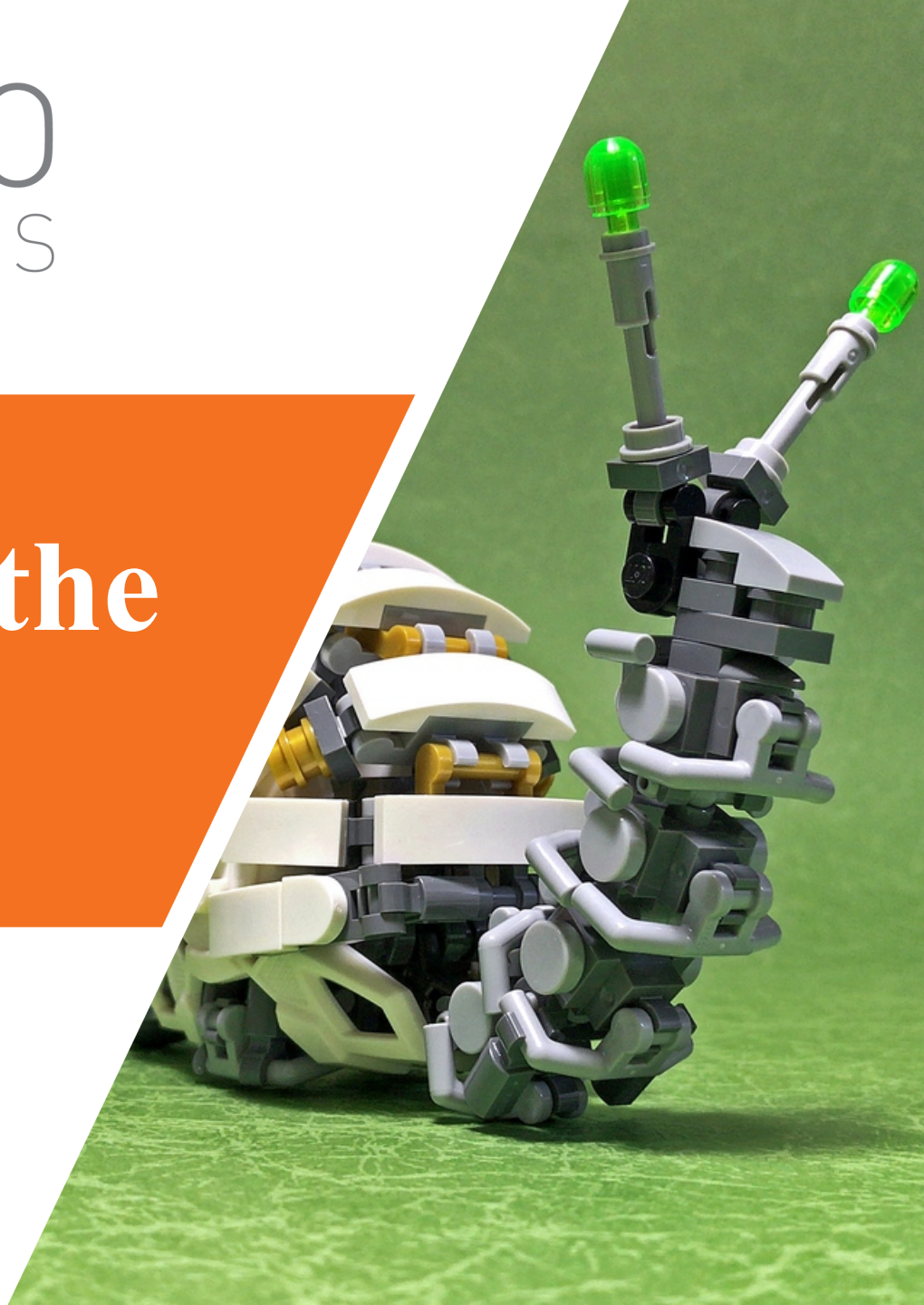
MANGO SOLUTIONS

The fast and the fuRious



Dr Jelena Ilic

Data Scientist

jilic@mango-solutions.com

Agenda

- Is R slow?
- Vectorise
- Matrices/Tables
- Allocate
- Subsetting
- What about loops?
- Reading data
- OO
- Passing by Reference

Is R slow?

- R runs in RAM
  - RAM is fast
- How much RAM do we need
  - https://csgillespie.github.io/efficientR/

A rough rule of thumb is that your RAM should be three times the size of your data set.

Is R slow?

- Core R is mostly written in C, C++ and Fortran
  - C, C++ and Fortran provide low-level access to memory
  - C, C++ and Fortran provide language constructs that map efficiently to machine instructions
  - C, C++ and Fortran **are fast**

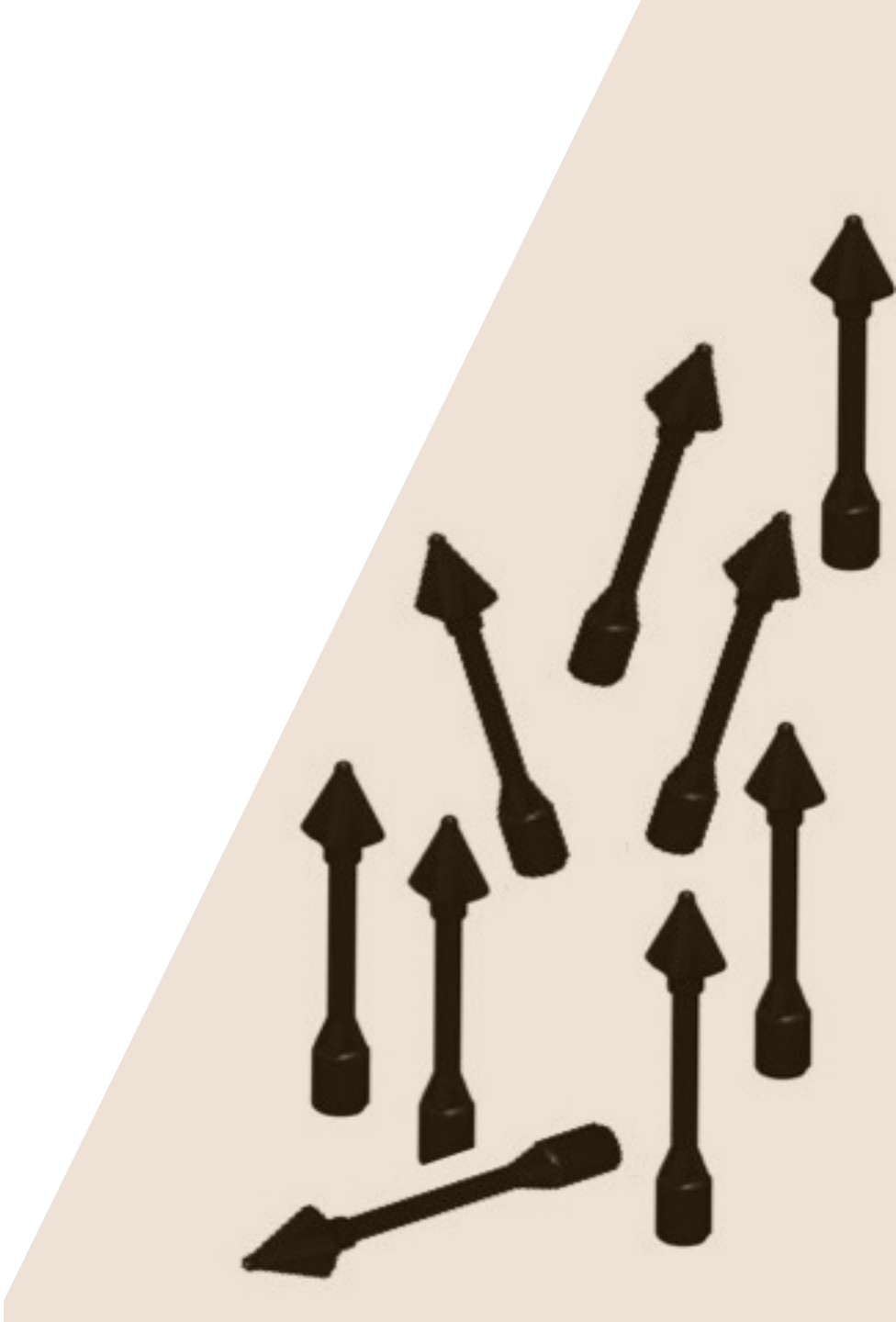
Is R slow?

- Most of the user-visible functions in R are written in R
- R is free, anybody can write a package

Is R slow?

- R lets users to solve problems in many ways
- In R “almost anything can be modified after it is created” (H. Wickham)
- Non-programmers can pick-up R quickly
- In other words, it is easy to write slow R code

Vectorise



Vectorise

- By its construction R is a high-level computer language
- It takes care of a lot of basic computer tasks for us:

R

```
> x <- 5
> y <- "test"
> z <- c(1,2,3,4,5)
```

C++

```
int x = 5;
string y = "test";
std::vector <int> z ={1,2,3,4,5};
```

- "Figuring out" is done for us!

Vectorise

- If we want to run a (compiled) function over all the values in a vector (object) we could:
  a) pass a whole vector (object) through the R function to the compiled code
  b) call the R function repeatedly for each value

- If we do the latter, R has to do the “figuring out” step for each element of the object

- Let’s look at the code

Matrices & Tables

Matrices & Tables

- Matrix is a memory efficient object
- Holds far less “metadata” than a table
- Use if the speed is the priority and we can transform the table into a matrix
- How: Reshape the data
  - tidyr (`gather, spread, separate`…)
- Let’s look at the code

Matrices & Tables

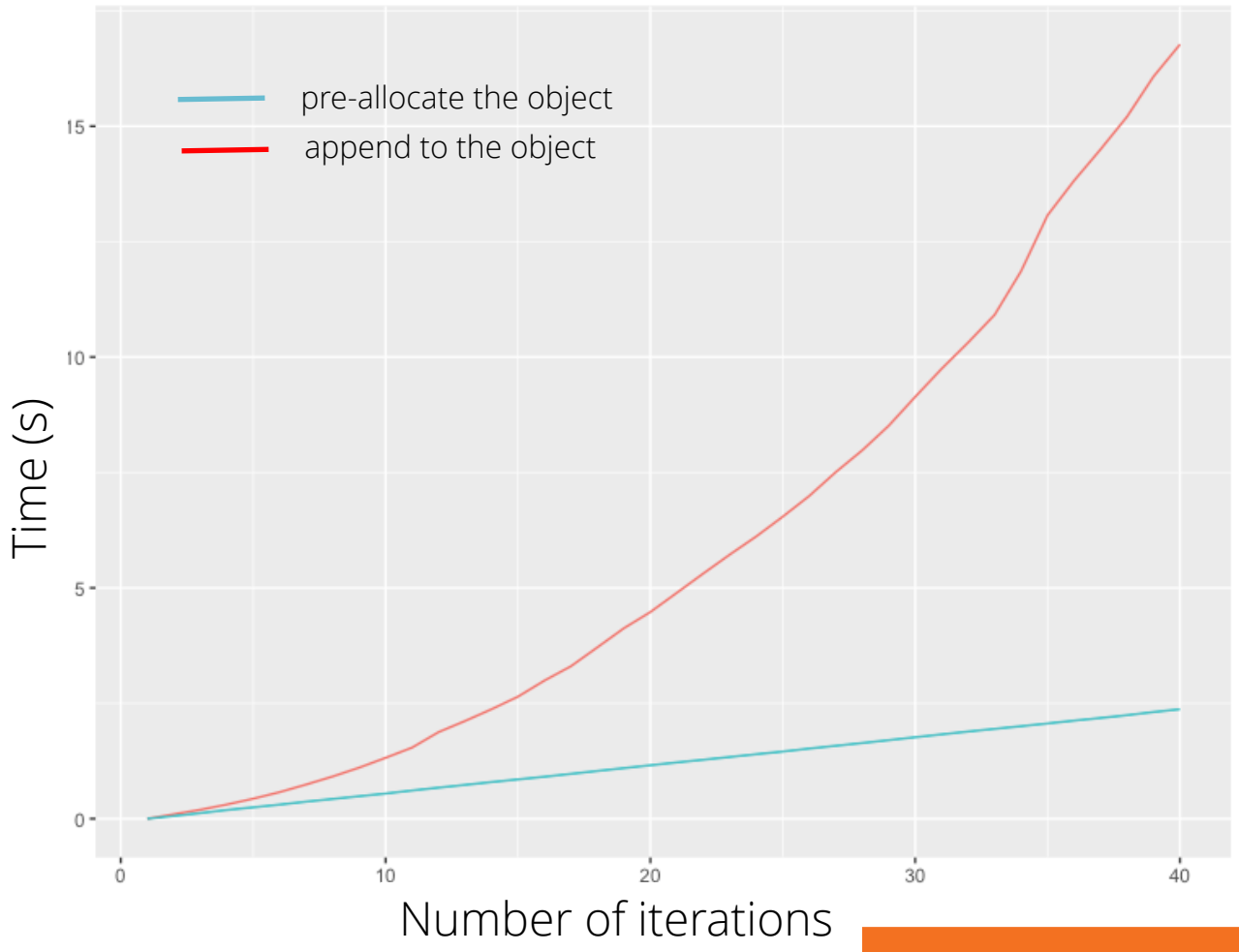
- If we have to deal with tables and the speed is important:
  - `data.table` package
- It is an enhancement to the data.frame class
- It allows indexing, merging and grouping data much faster than when using data frames
- Useful when we want to work with large data sets
- A bit odd syntax

Pre-Allocate

Pre-Allocate

- Memory allocation is (very) slow in R
- It is somewhat slow in all languages
- The cure is to think about the size of the object (number of rows, columns, elements…) in advance
- Let's look at the code

Pre-Allocate

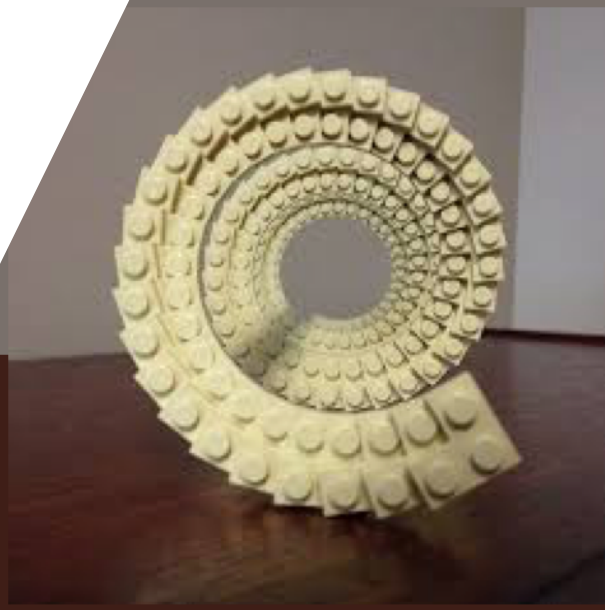


Subsetting

Subsetting

- Subsetting (filtering, selecting, extracting…) is a common task in data preparation process
- There are many convenience functions in R that make such operations easy to perform
- However, for code reuse, lower level functions are usually more efficient
- Let's look at the code

What about Loops?



What about Loops?

- Loops are not bad
- Some other things in combination with loops are
  - Not pre-allocating objects
  - Not initializing the loop
  - Nested loops
  - …
- Combining objects by using “`c`”, “`rbind`”, “`cbind`” or "`<-`" to a new position in an object we create a copy of the object containing both sets of data
- In loops this means we are creating many objects, then deleting them, then garbage collecting them many, many times…
- Let’s look at the code

Loops and Apply

- There is a misconception that **`apply`** functions are faster than loops
- They can be slightly faster
- Frequently can even be slower
- BUT, they are parallelizable, and so POTENTIALLY far faster
- With apply family the gain is in:
  - coding efficiency
  - computation time with the use of parallelization
- Annoying feature: input and output can be of different type

Loops and Apply

- apply functions
  - `apply`: arrays, matrices, data.frames
  - `lapply, sapply, vapply`: lists, data.frames, vectors
  - `tapply`: grouped operations (table apply)
  - `mapply`: multivariate version of sapply
  - `replicate`: similar to sapply

- Apply handles these memory allocation issues for you, but then you have to write the loop part as a function to pass to apply
- *At its heart, apply is just a for loop with extra convenience.*

- Let's look at the code

Reading data

Reading tables

- readr (read_csv,...)
- utils (read.table, ...)
- data.table (fread)

- By default, all read functions will try to deduce the type of data in the column(s)

- They do it somehow differently

- We could speed up the reading by setting column type parameter
- Or we could stick with `fread`

- Let's look at the code

A sprinkle of OO

OO R code

- Maintainability
  - Identifying the source of errors is easier (objects self-contained)
- Reusability
  - Objects contains both data and methods that act on data
  - Methods provide a predefined interface to data / functionality
- Scalability
  - Interface provides means for reusing the object in new software
  - Provides all info to replace object without affecting other code
  - Aging code can be replaced with faster algorithms and newer technology

OO R code

- Thinking OO helps with better understanding of the problem
- Organised code => faster code
- R6 classes
- https://cran.r-project.org/web/packages/R6/vignettes/Introduction.html

Passing by Reference

Passing by Reference

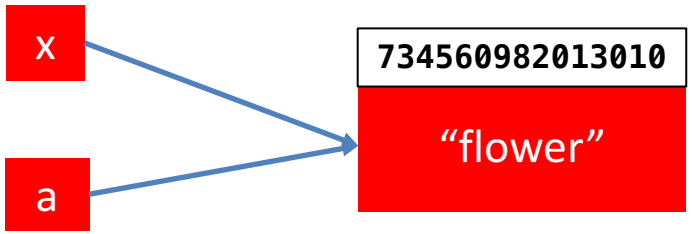
- R does not pass by reference
- Objects are copied
- And copied
- And copied
- ...
- Which is fine, easy to understand
- But costs memory and speed

Passing by Reference

some_function(x)

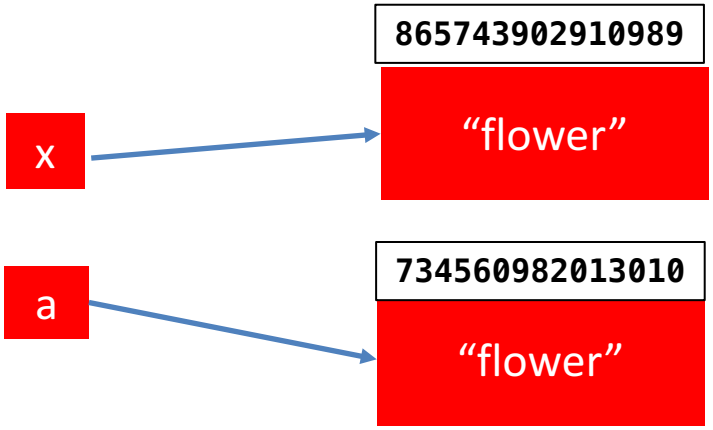
some_function(x = a)

By reference



x and **a** point to the same object

By value



x is a copy of **a**
x and **a** have different memory addresses

Passing by Reference - Environments

- There is a workaround: environments
- Environments are passed by reference
- Create an environment
- Sort out your data
- Put your data in it
- Write function that takes the environment as a parameter

Summary

- Vectorise as much as you can
- Use memory efficient objects wherever possible
- Have in mind that R copies the objects
- Loops are not (always) bad
- Pre-allocate
- Think OO and use environments
- Use low-level R functions