



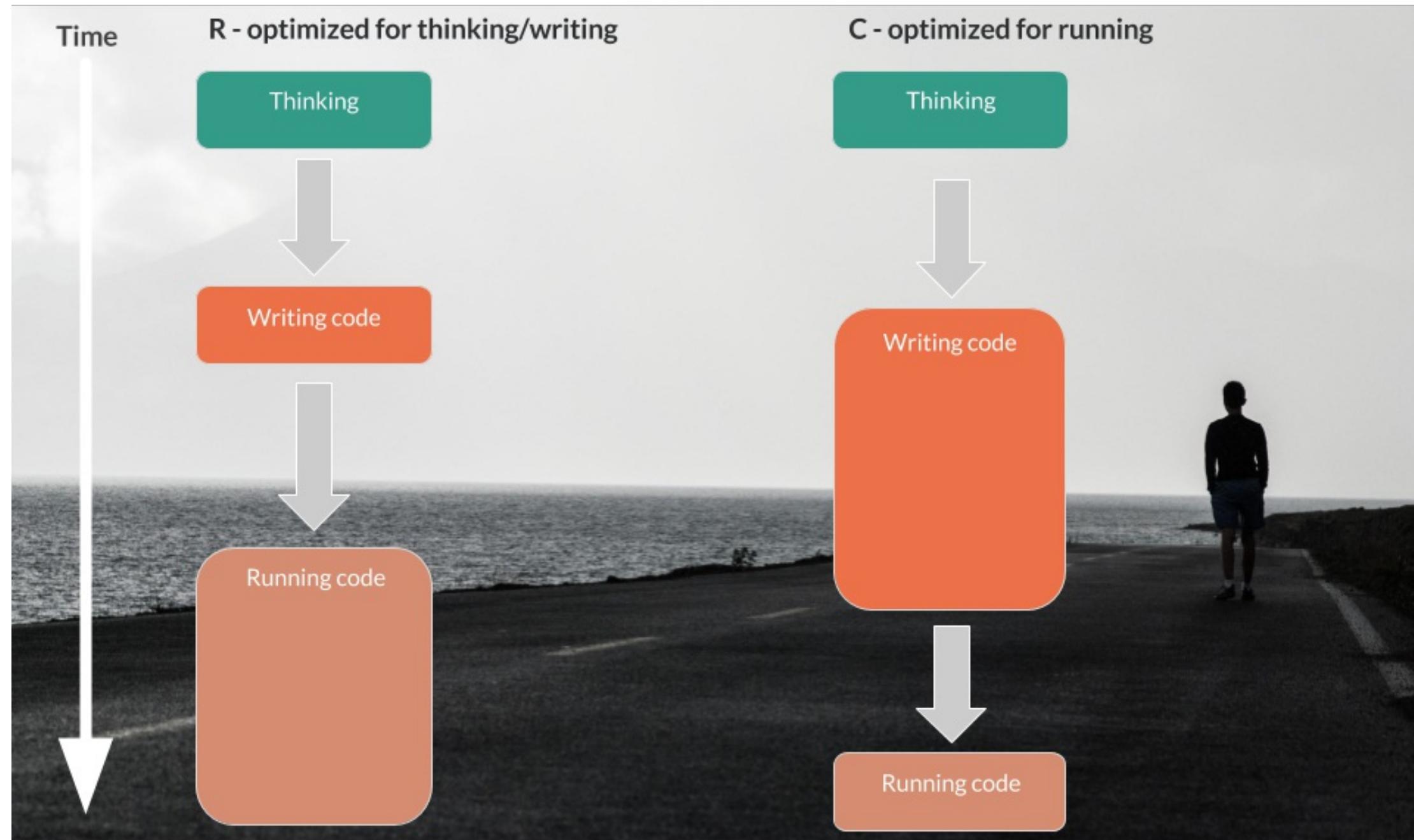
WRITING EFFICIENT R CODE

**Welcome, Bienvenue,
Willkommen,**

Colin Gillespie

Jumping Rivers & Newcastle University





A typical R workflow

```
data_set <- read.csv("dataset.csv") # Load  
plot(data_set$x, data_set$y) # Plot  
lm(y ~ x, data = data_set) # Model
```

When to optimize

premature optimization is the root of all evil

Popularised by **Donald Knuth**

R version

- **v2.0** Lazy loading; fast loading of data with minimal expense of system memory.
- **v2.13** Speeding up functions with the byte compiler
- **v3.0** Support for large vectors
- Main releases every *April*
 - e.g. 3.0, 3.1, 3.2
- Smaller bug fixes throughout the year
 - e.g. 3.3.0, 3.3.1, 3.3.2



WRITING EFFICIENT R CODE

Let's practice!



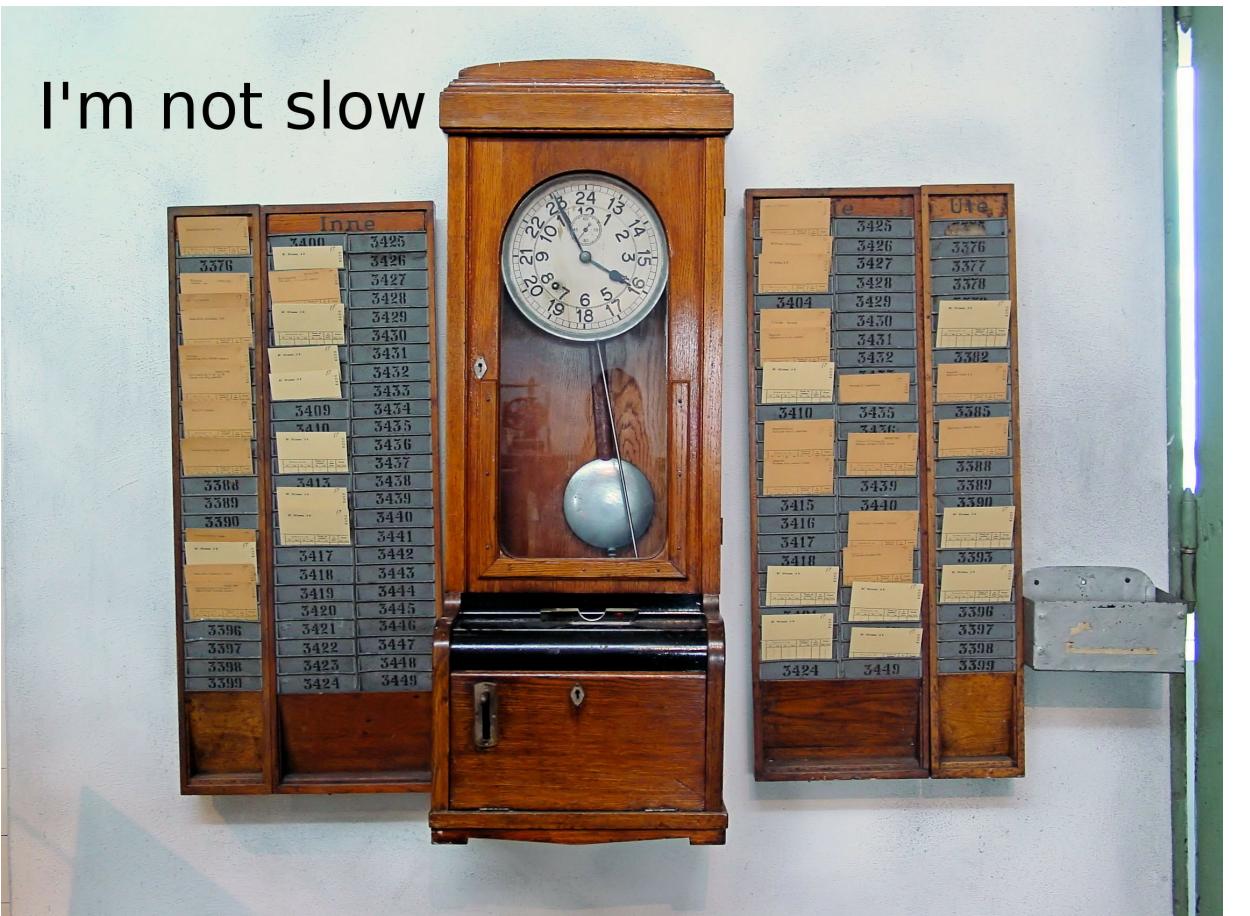
WRITING EFFICIENT R CODE

My code is slow!

Colin Gillespie

Jumping Rivers & Newcastle University

Is my code really slow?



- 1 second?
- 1 minute?
- 1 hour?

Is my code really slow?

Benchmarking: time how long each solution takes, then select the fastest

Benchmarking

1. We construct a function around the feature we wish to benchmark
2. We time the function under different scenarios, e.g. data set

Example: Sequence of numbers

\$\$ 1, 2, 3, \dots, n \$\$

OPTION 1

```
1:n
```

OPTION 2

```
seq(1, n)
```

OPTION 3

```
seq(1, n, by = 1)
```

Function wrapping

```
colon <- function(n) 1:n  
colon(5)  
# [1] 1 2 3 4 5
```

```
seq_default <- function(n) seq(1, n)  
seq_by <- function(n) seq(1, n, by = 1)
```

Timing with system.time()

```
system.time(colon(1e8))
#   user  system elapsed
# 0.032  0.028  0.060
```

```
system.time(seq_default(1e8))
#   user  system elapsed
# 0.060  0.028  0.086
system.time(seq_by(1e8))
#   user  system elapsed
# 1.088  0.520  1.600
```

- **user** time is the CPU time charged for the execution of user instructions.
- **system** time is the CPU time charged for execution by the system on behalf of the calling process.
- **elapsed** time is approximately the sum of user and system, this is the number we typically care about

Storing the result

The trouble with

```
system.time(colon(1e8))
```

is we haven't stored the result. We need to rerun the code to store the result

```
res <- colon(1e8)
```

The `<-` operator performs both:

- Argument passing
- Object assignment

```
system.time(res <- colon(1e8))
```

The `=` operator performs **one** of:

- Argument passing
- object assignment

```
## Raises an error  
system.time(res = colon(1e8))
```

Relative time

Method	Absolute time (secs)	Relative time
colon(n)	0.062	\$0.060/0.060 = 1.00\$
seq_default(n)	0.086	\$0.086/0.060 = 1.40\$
seq_by(n)	1.607	\$1.60/0.060 = 26.7\$

Microbenchmark package

- Compares functions
- Each function is run multiple times

```
library("microbenchmark")

n = 1e8
microbenchmark(colon(n),
               seq_default(n),
               seq_by(n),
               times = 10) # Run each function 10 times
# Unit: milliseconds
#      expr   min    lq    mean   median    uq    max   neval cld
# colon(n)  59 130  220     202 341  391    10     a
# seq_default(n) 94 204  290     337 348  383    10     a
# seq_by(n) 1945 2044 2260    2275 2359 2787    10     b
```



WRITING EFFICIENT R CODE

Let's practice!



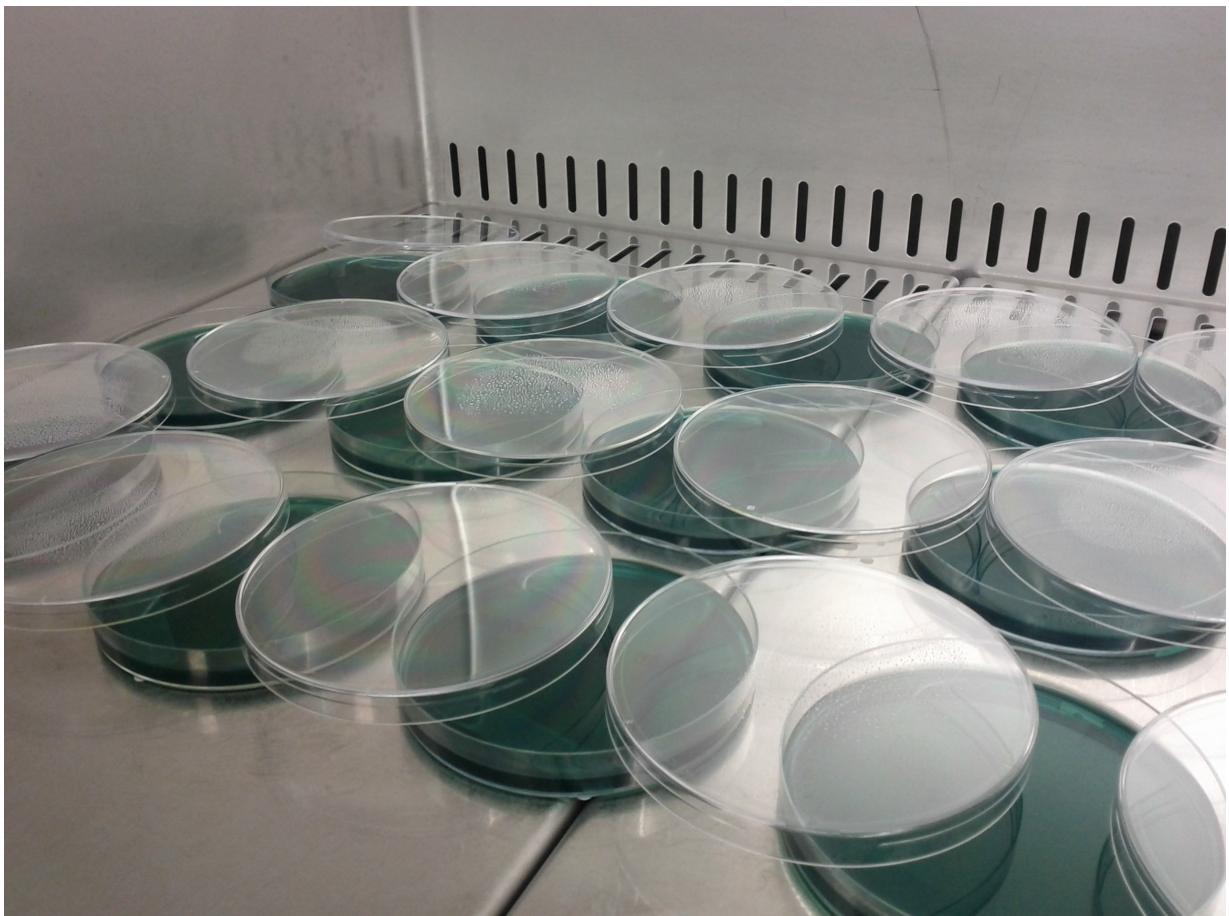
WRITING EFFICIENT R CODE

How good is your machine?

Colin Gillespie

Jumping Rivers & Newcastle University

Experiments!

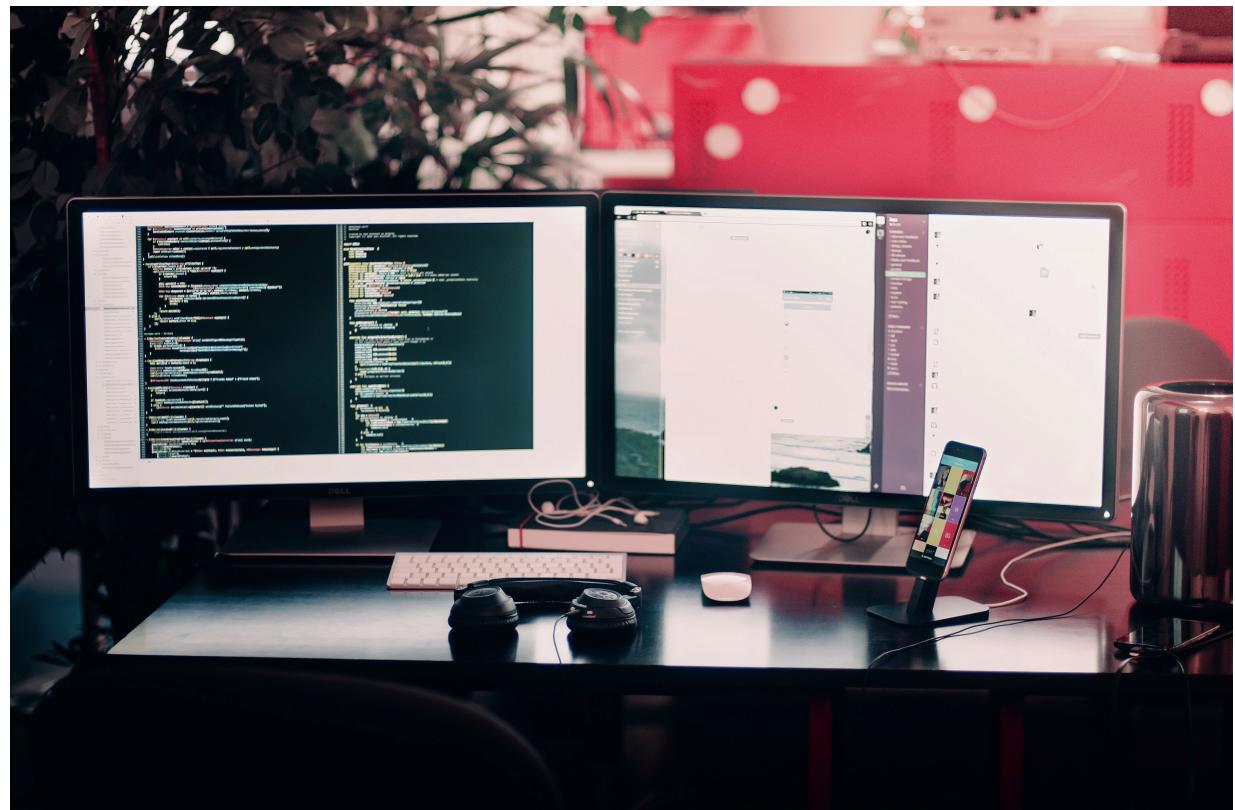


Cost of experiment:

- Experimental equipment
- Researcher time

Not cheap!

To buy, or not to buy...



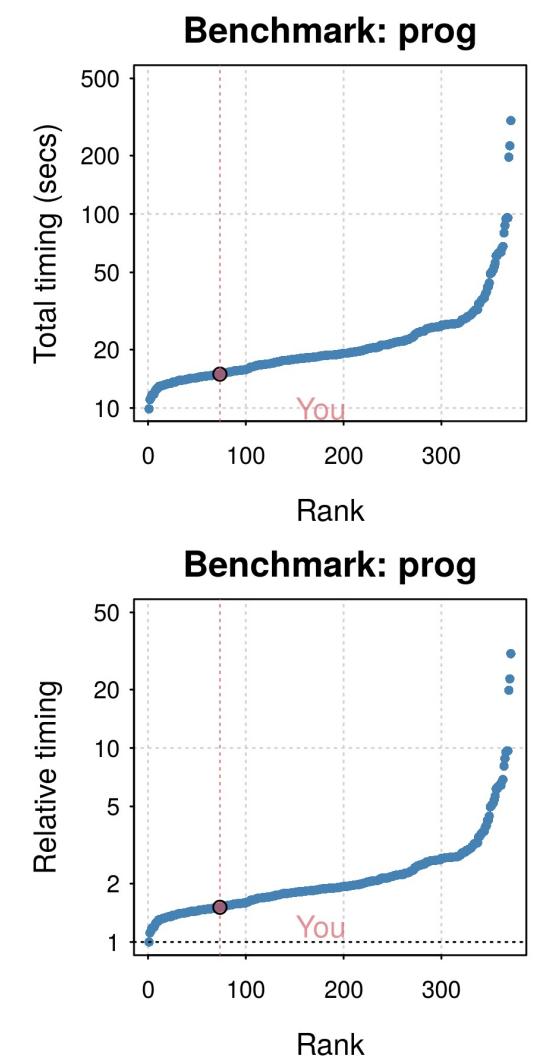
To buy, or not to buy...

- Analysis takes twenty minutes on your current machine
- Ten minutes to run on a new machine
- Your time is charged at \$100 per hour
- Run sixty analyses to pay back the cost of a \$1000 machine

The benchmarkme package

```
install.packages("benchmarkme")  
  
library("benchmarkme")  
  
## Run each benchmark 3 times  
res <- benchmark_std(runs = 3)  
  
plot(res)  
  
upload_results(res)
```

*My machine is ranked
75th out 400 machines*





WRITING EFFICIENT R CODE

Let's practice!