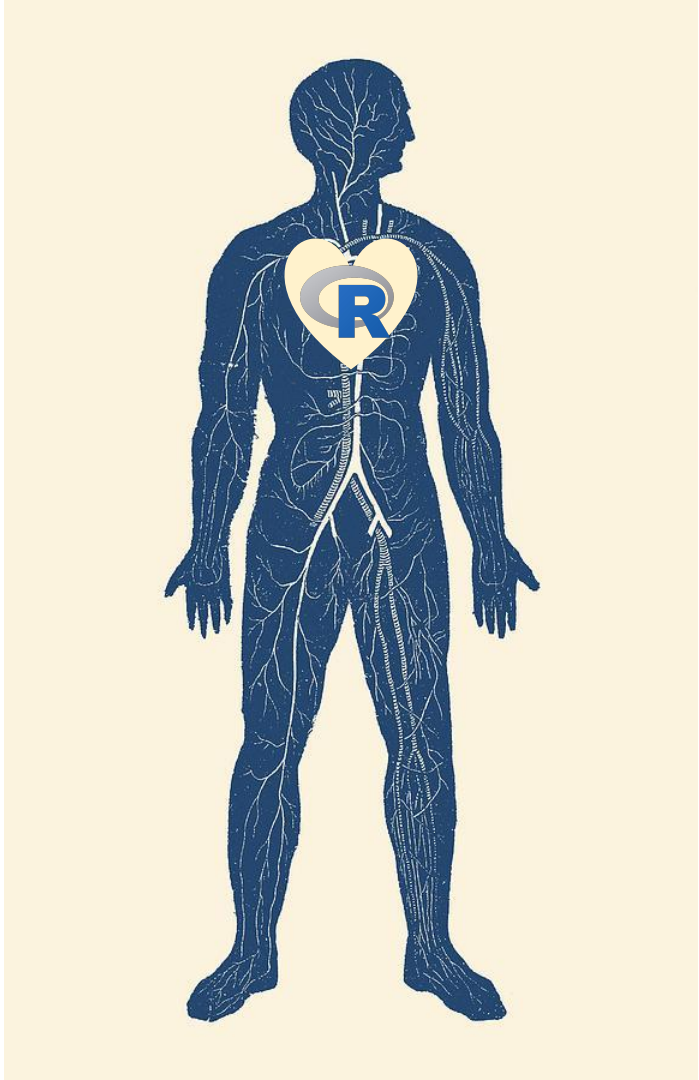
Dissecting a Function

Profiling and Benchmarking Your Way to Faster Code

Adam Austin

just_add_data

ataustin



Why are we here?

- We write R code. Sometimes it’s slow.
- In R, we are free to achieve our objectives in many different ways
- That freedom comes at a cost

Computational awareness

"To understand computations in R, two slogans are helpful:

- Everything that exists is an object.
- Everything that happens is a function call."

-- John Chambers

stuff uses memory

stuff takes processing power

Don’t worry, Hadley will save us

“Before you can make your code faster, you first need to figure out what’s making it slow. **This sounds easy, but it’s not.** Even experienced programmers have a hard time identifying bottlenecks in their code.”

-- Hadley Wickham (Advanced R)

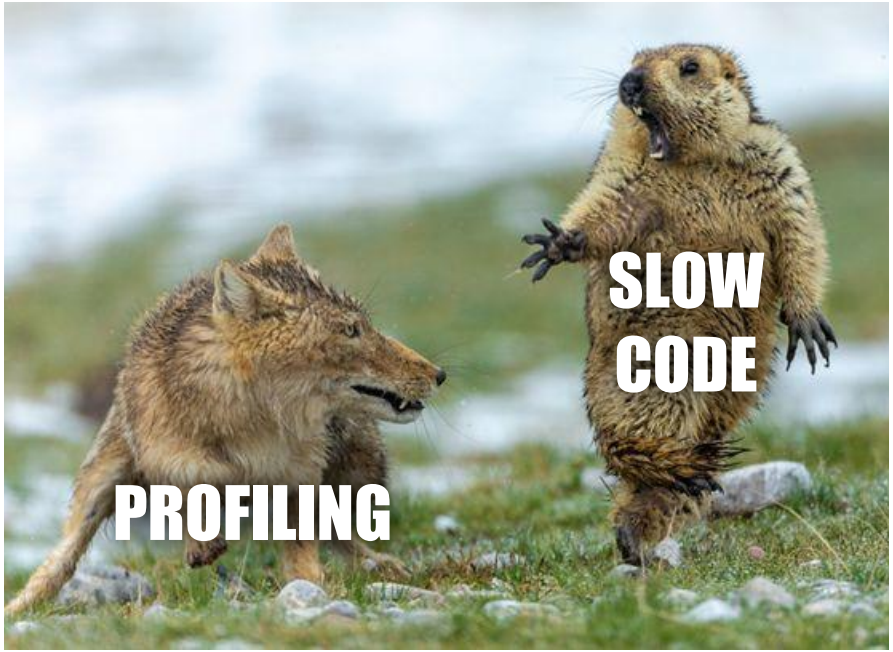
How do we see the computational cost of our code?

Profiling: the empiricist's guide to faster code

DEFINITION

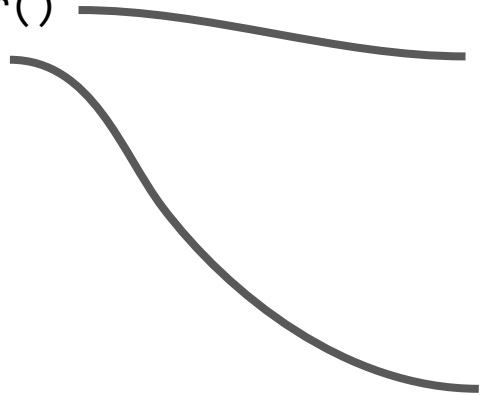
A **profile** is is R's estimate of the resources consumed during the execution of code.

To build a profile, R inspects the *call stack* many times per second and logs the current function and the memory allocation.



The call stack: a record of function calls

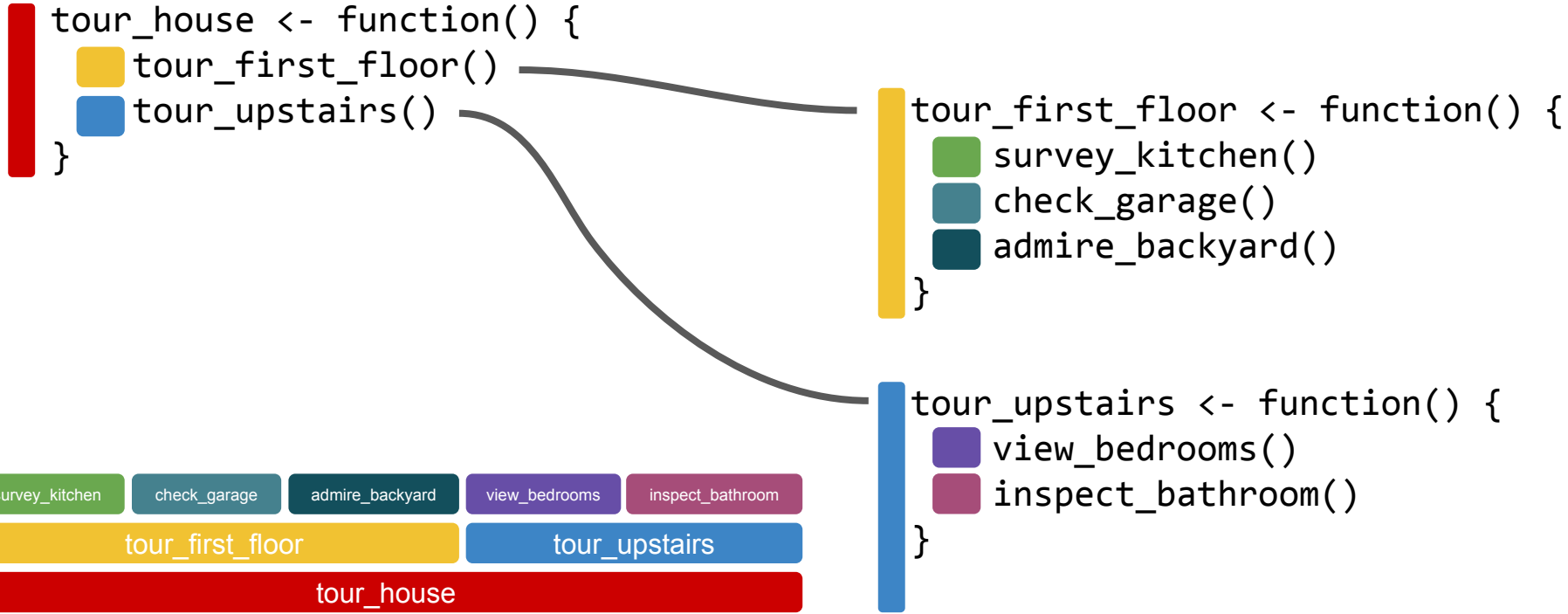
```
tour_house <- function() {
    tour_first_floor()
    tour_upstairs()
}
```

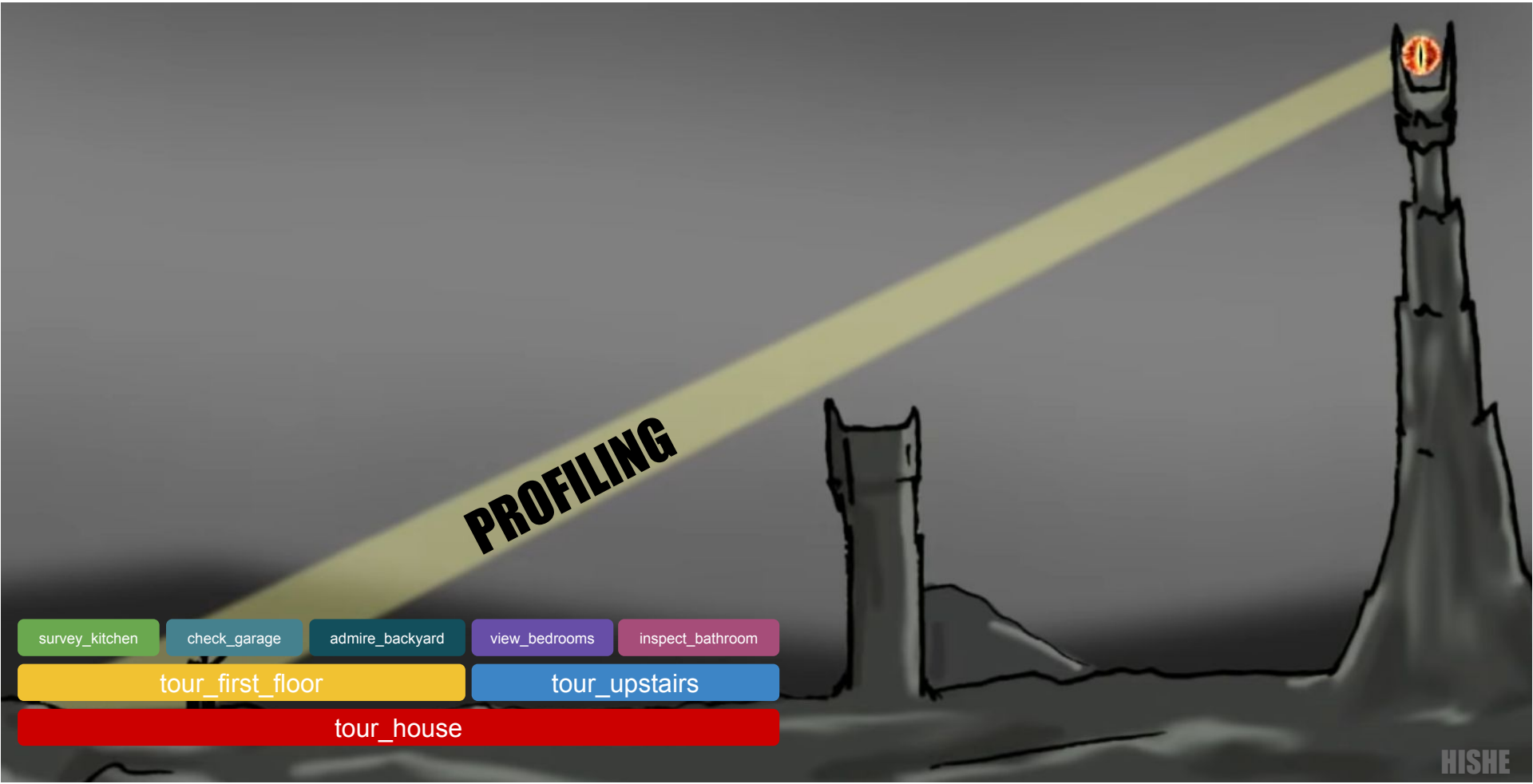


```
tour_first_floor <- function() {
    survey_kitchen()
    check_garage()
    admire_backyard()
}
```

```
tour_upstairs <- function() {
    view_bedrooms()
    inspect_bathroom()
}
```

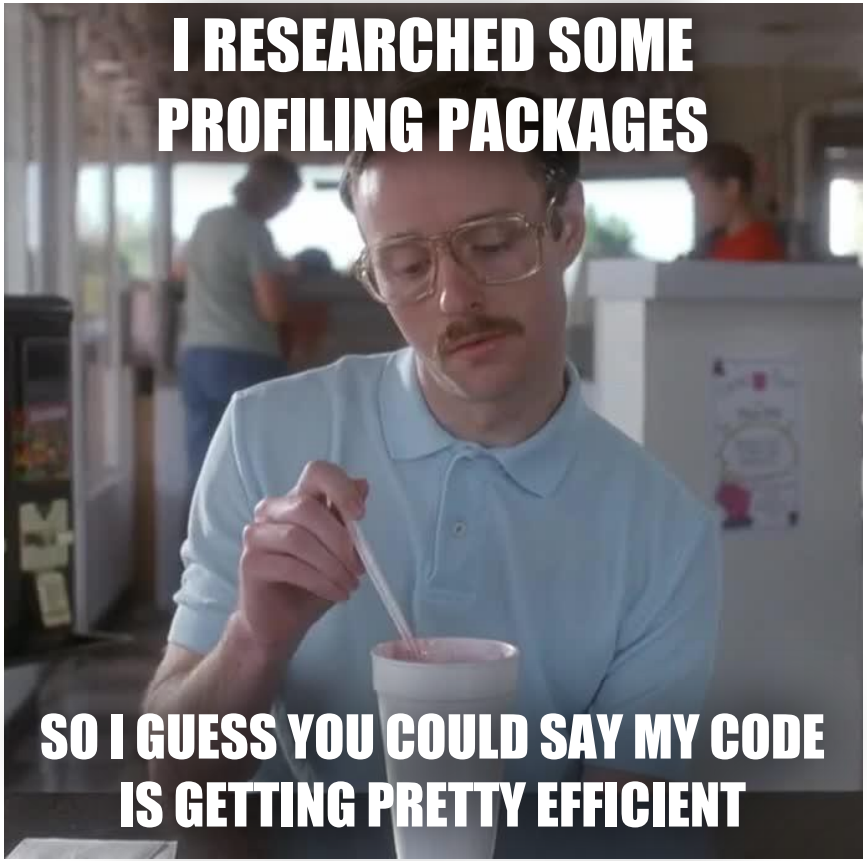
The call stack: a record of function calls




PROFILING
survey_kitchen
check_garage
admire_backyard
view_bedrooms
inspect_bathroom
tour_first_floor
tour_upstairs
tour_house
HISHE

Profiling packages: the lineup (all on CRAN)

- `utils` (yeah, `utils`!)
  - Ships with R!
  - Rprof, Rprofmem, summaryRprof
- `proftools`
  - Maintained by Luke Tierney
  - Helpful visualizations
- `profvis`
  - Maintained by Winston Chang
  - Built on top of utils functions
  - RStudio integrated support



A closer look at profvis

```
profvis::profvis(
    expr =
   ,interval =
   ,[...other stuff...]
)
```

the code to profile

time interval (in seconds) at which to sample the call stack

output options, memory management, etc.

A closer look at `profvis`

```
profvis::profvis(
    expr = too_long_function()
   ,interval = 0.01       # the default


)
```

`profvis` in action

Speed bumps 101: extraneous `for` loop

Instead of...

```
input <- 1:100
total <- 0

for(i in 1:length(input)) {
  total <- total + input[i]
}
```

try...

```
sum(input)
```

because...

Built-in vectorized functions call C code under the hood and run their loops blazingly fast.

Some amazing magical loops:

```
sum
cumsum
prod
cumprod
pmin
pmax
cummin
cummax
colSums
rowSums
colMeans
rowMeans
which.min
which.max
diff
```

Speed bumps 201: memory not allocated

Instead of...

```
input  <- 1:100
output <- some_func(input[1])

for(i in 2:length(input)) {
  output[i] <- some_func(input[i])
}
```

try...

```
vapply(input, some_func, numeric(1))
```

because...

The *apply family pre-allocates memory so R does not repeatedly make new copies of the vector. (You can also do this yourself before a `for` loop).

The *apply family is running out of letters:

```
apply
eapply
lapply
mapply
rapply
sapply
tapply
vapply
```

With special guests:

```
replicate
simplify2array
```

Speed bumps 301: too much subsetting

Instead of...

```
for(col in names(df)) {
  data[[col]] <- as.integer(df[[col]])
}
```

try...

```
df[] <- lapply(df, as.integer)
```

because...

Empty subsetting with assignment will preserve that object's class.

To selectively replace values in a vector, use subsetting conditions such as:

```
is.na
is.nan
is.infinite
==, !=, <, >
```

For example:

```
x[is.na(x)] <- 0

x[x < 0] <- NA
```

Speed bumps 401: sneaky functions

Instead of...

```
x <- sapply(y, package::function)
```

try...

```
library(package)
x <- sapply(y, function)
```

because...

Everything that happens in R is a function call… like, *everything*.

Some common sneaky functions:

```
<-
$, $<-
[, [<-
[[, [[<-
:
::, :::
(
{
&, |
+, -, *, /, ^
==, !=, <, >
```

Do you even need to optimize?

Is the speedup in the future worth the time investment now?

Is your code being called repeatedly, or only once?

Will your function be used regularly, or only rarely?

Are you optimizing globally, or only one small part of a larger process?

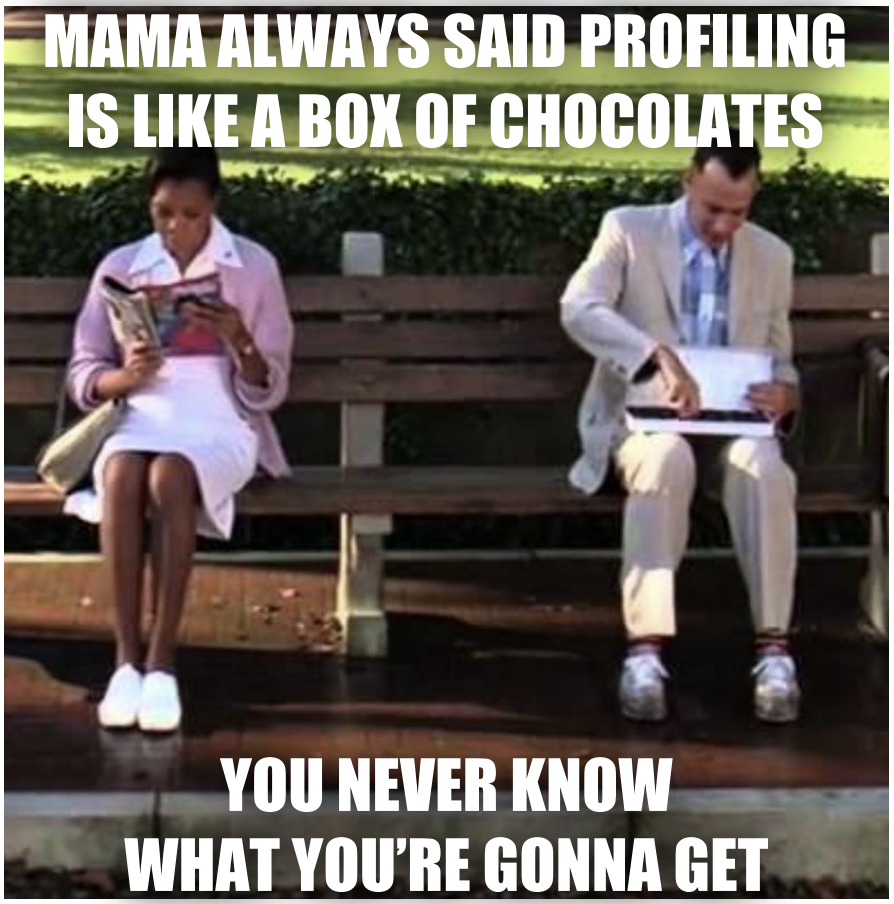
Limitations of profiling

A profile is not deterministic.

Anonymous functions can’t be distinguished from each other.

Lazy evaluation might make the call stack hard to interpret.

Standard tools won’t help you profile compiled (C/C++) code.



You've found the problem. Now what?

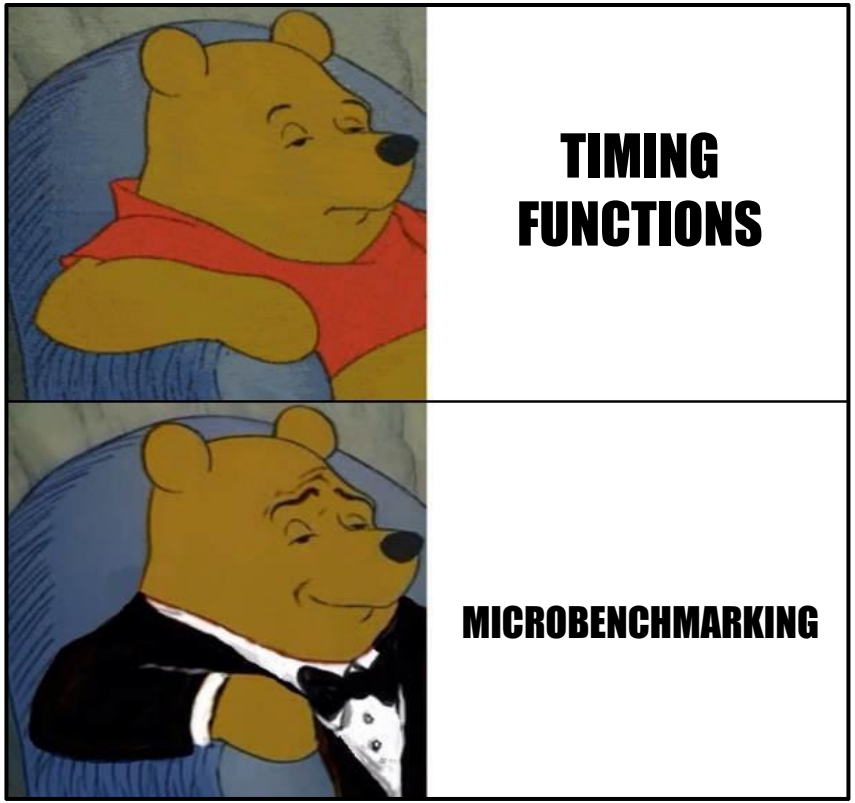
- **Profiling** finds bottlenecks in a sequence of steps in your function
- **Benchmarking** compares alternative approaches to an individual step

Incremental improvements with (micro)benchmarking

DEFINITION

A **(micro)benchmark** is a performance comparison of different functions that accomplish the same thing.

Benchmarks provide the timing of small, specific, and relatively short pieces of code.



Benchmarking packages: the lineup (all on CRAN)

- `base`
  - Sort of. You can use `system.time`
- `rbenchmark`
- `microbenchmark`
- `tictoc`
- `bench`

A closer look at bench (from Jim Hester)

```
bench::mark(
    ... =
   ,min_time =
   ,[...other stuff...]
)
```

comma-separated expressions to benchmark

minimum number of seconds to run each expression

iteration counts, memory management, etc.

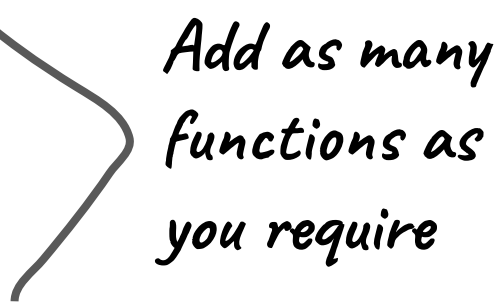
A closer look at bench

```
bench::mark(
    ... =


  ,min_time =
)
```

A closer look at bench

```
bench::mark(
    function1(x)
   ,function2(x)
   ,min_time = 0.5          # the default
)
```

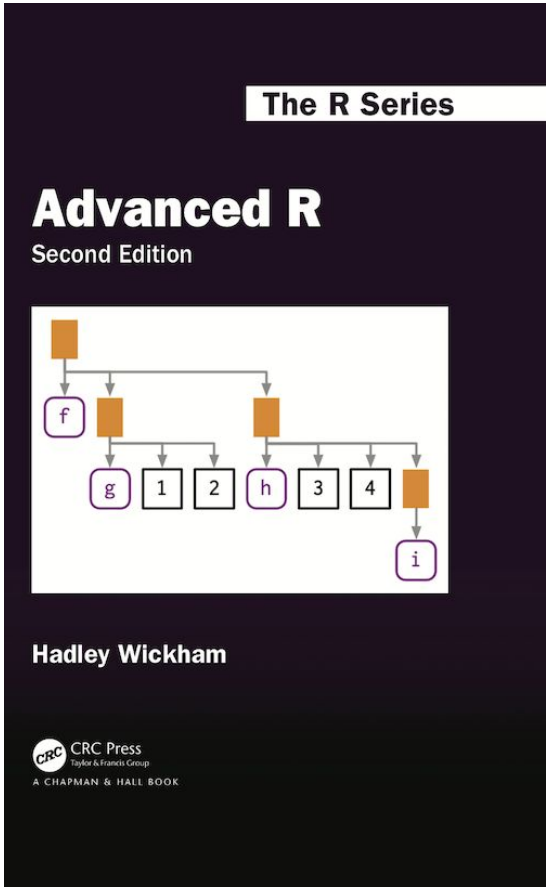


A closer look at bench

```
bench::mark(
    sqrt(1:1000)
  ,(1:1000)^0.5
  ,min_time = 0.5          # the default
)
```

bench in action

Further reading

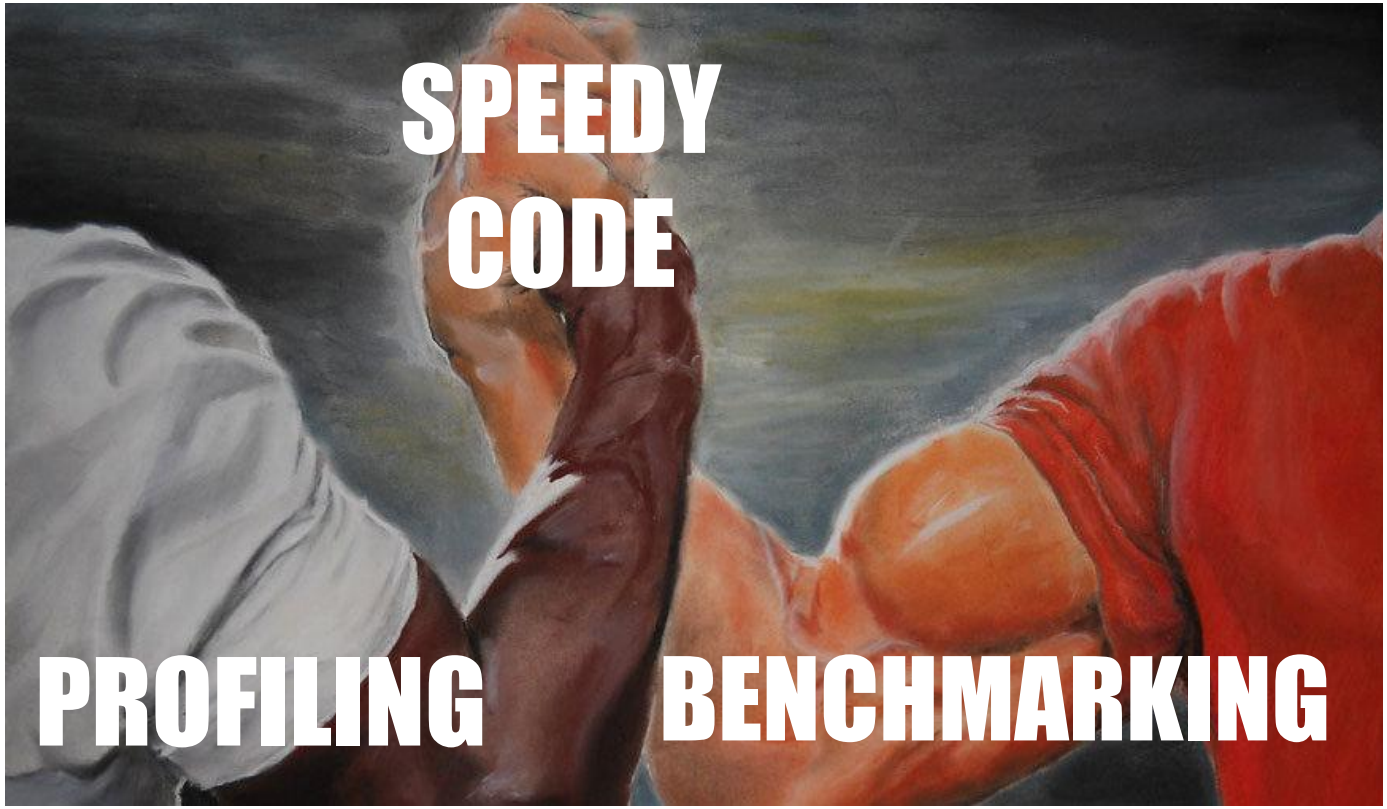


Advanced R, 2nd Edition

https://adv-r.hadley.nz/

Chapter 23: Measuring Performance

Chapter 24: Improving Performance



Thanks for listening!

Adam Austin
just_add_data
ataustin