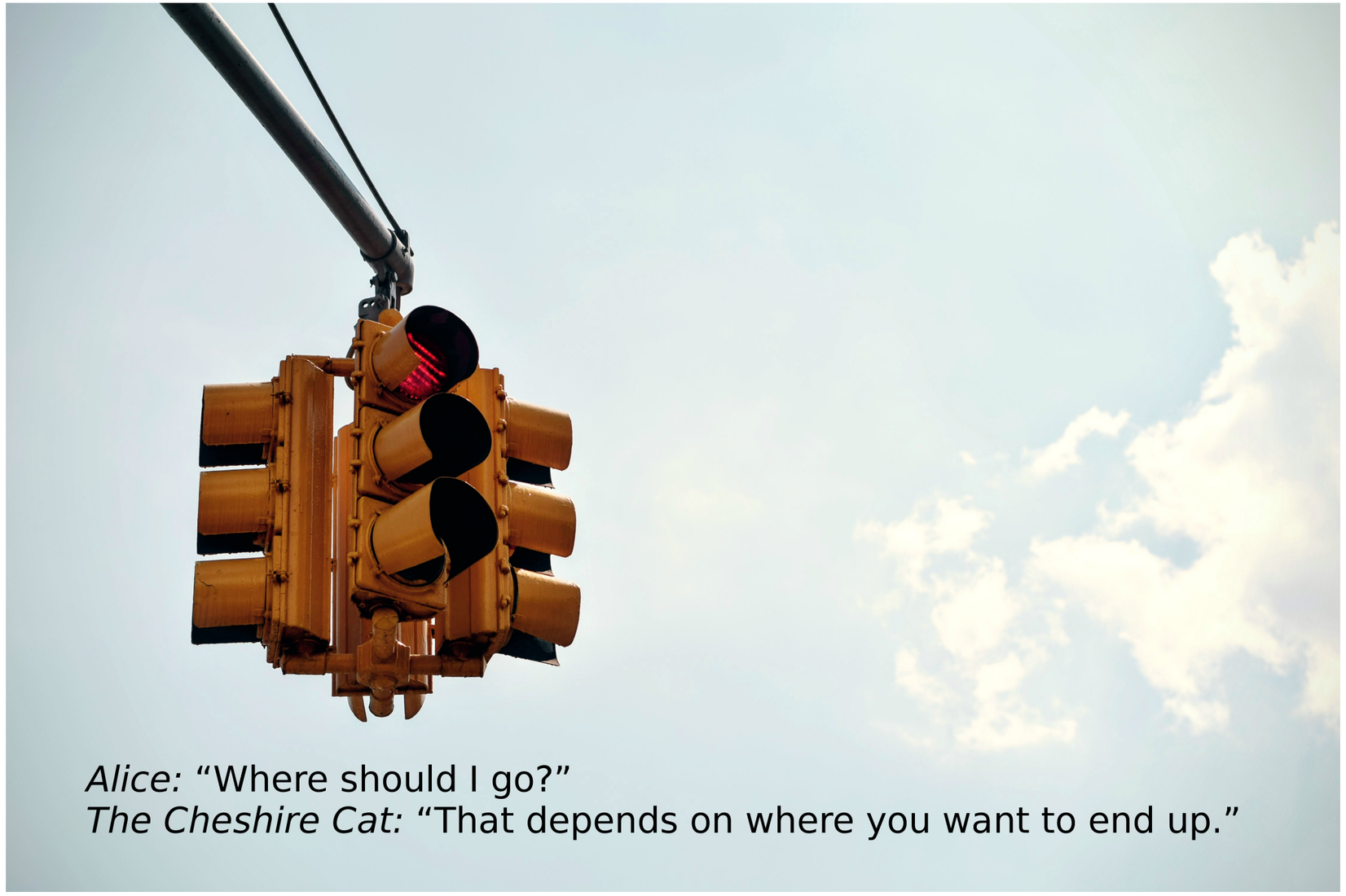
How do I find the bottleneck?

WRITING EFFICIENT R CODE

Colin Gillespie
Jumping Rivers & Newcastle University


Alice: “Where should I go?”
The Cheshire Cat: “That depends on where you want to end up.”

Code profiling

The general idea is to:

- Run the code
- Every few milliseconds, record what is being currently executed
- `Rprof()` comes with R and does exactly this
  - Tricky to use
- Use **profvis** instead

IMDB data set

- From the **ggplot2movies** package

```
data(movies, package = "ggplot2movies")
dim(movies)
```

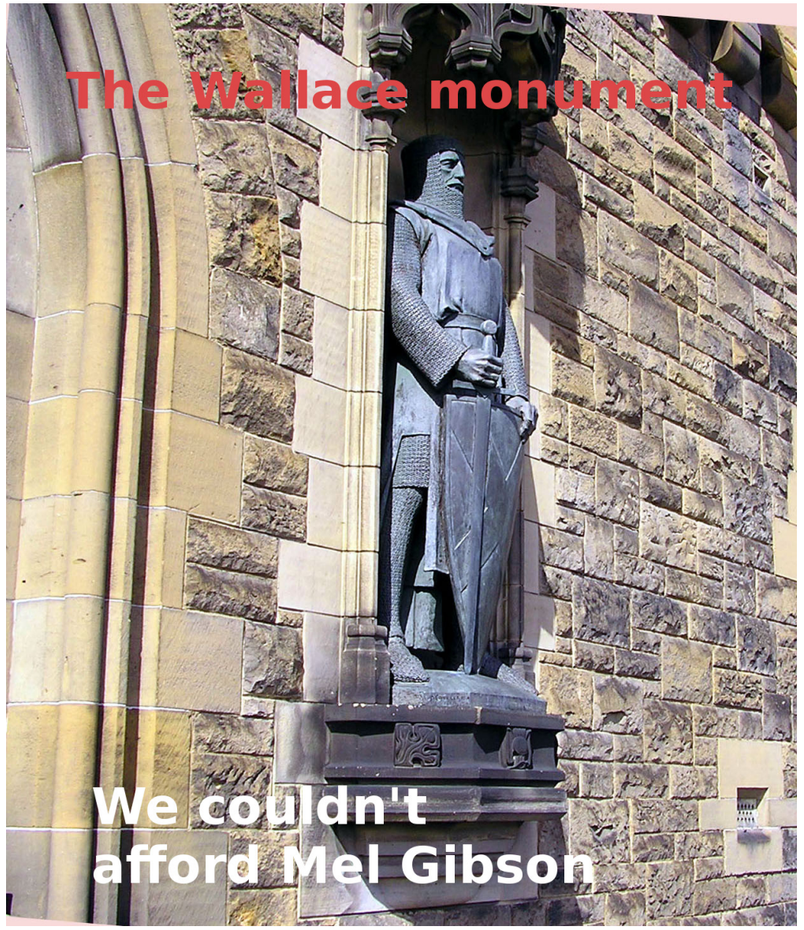
```
58788    24
```

- Data frame: around 60,000 rows and 24 columns
- Each row corresponds to a particular movie

Braveheart

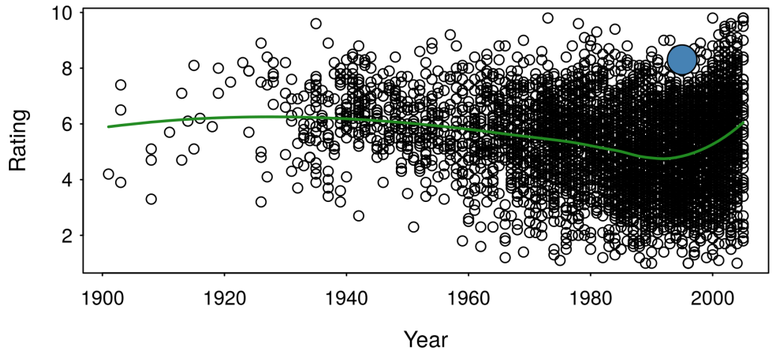
```
braveheart = movies[7288,]
```

Year	Length	Rating
1995	177	8.3



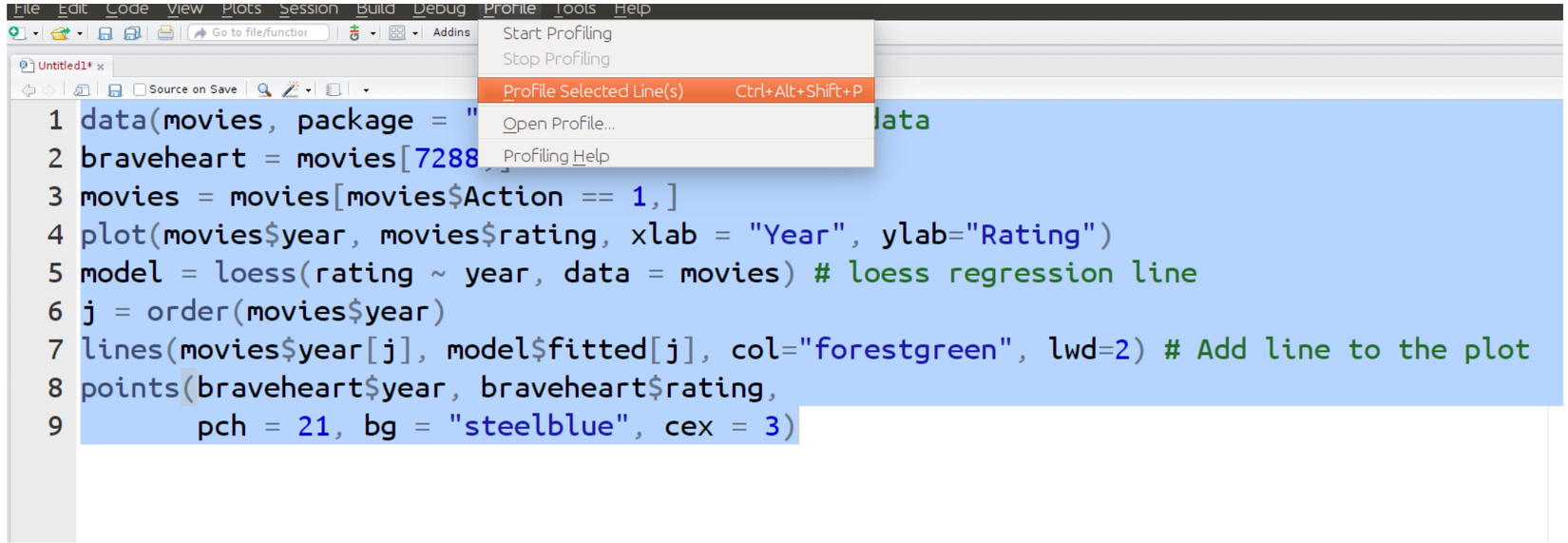
Example: Braveheart

```
# Load data
data(movies,
     package = "ggplot2movies")
braveheart <- movies[7288,]
movies <- movies[movies$Action==1,]
plot(movies$year, movies$rating,
     xlab = "Year", ylab = "Rating")
# local regression line
model <- loess(rating ~ year,
               data = movies)
j <- order(movies$year)
lines(movies$year[j],
      model$fitted[j],
      col = "forestgreen")
points(braveheart$year,
       braveheart$rating,
       pch = 21,
       bg = "steelblue")
```



Profvis

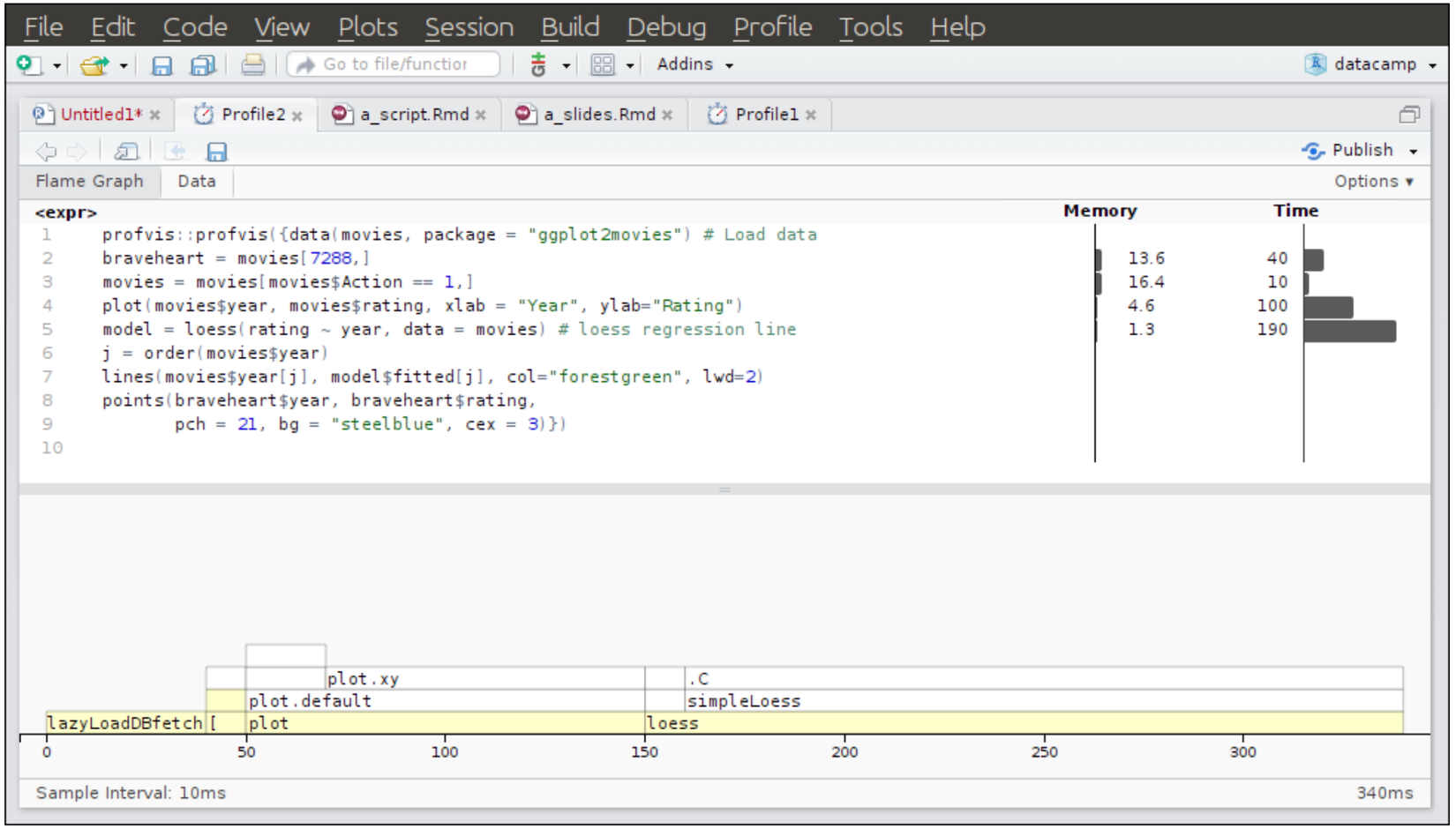
- RStudio has integrated support for profiling with profvis
  - Highlight the code you want to profile
  - `Profile -> Profile Selected lines`

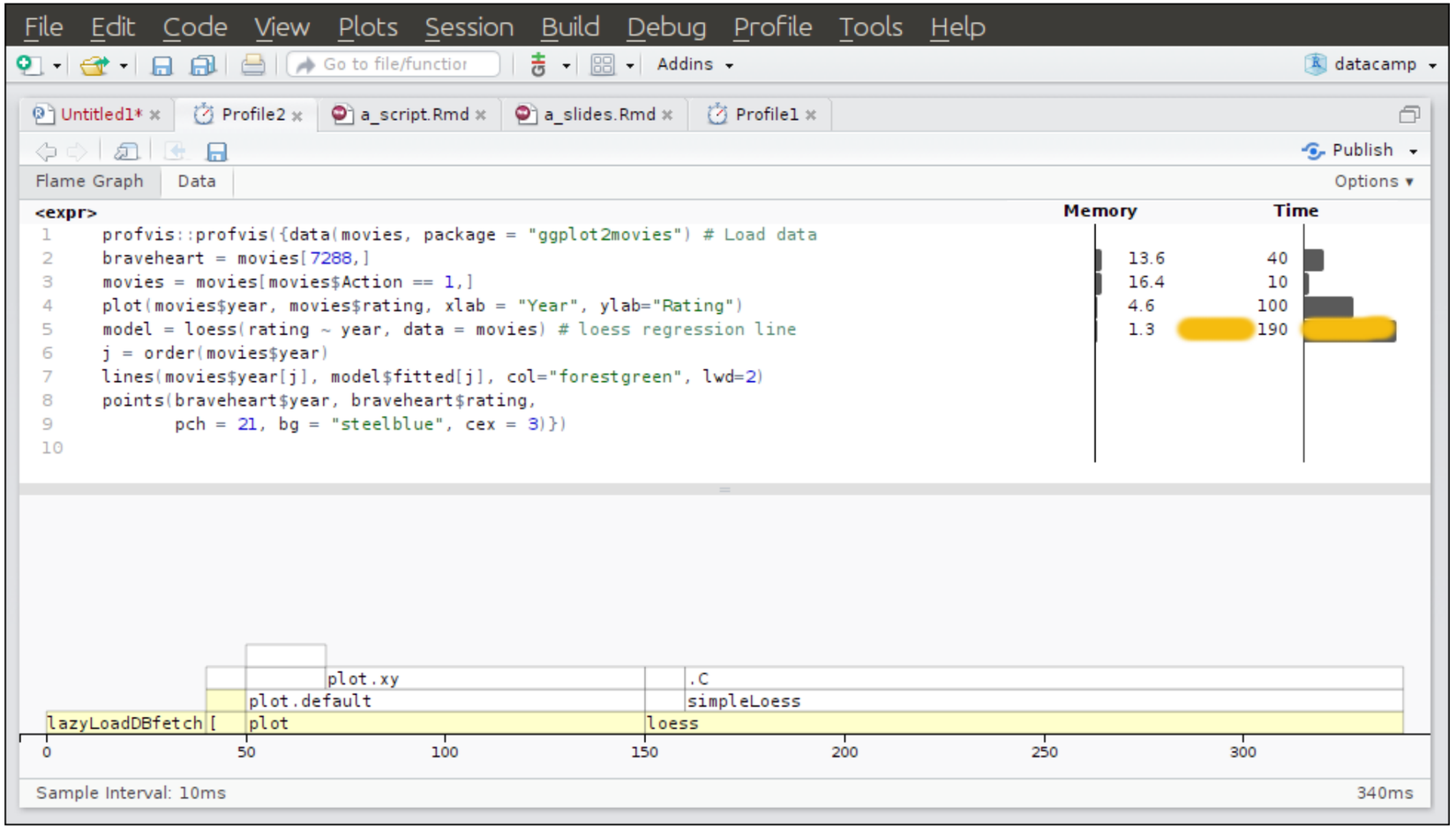


Command line

```
library("profvis")
profvis({
data(movies, package = "ggplot2movies") # Load data
braveheart <- movies[7288,]
movies <- movies[movies$Action == 1,]
plot(movies$year, movies$rating, xlab = "Year", ylab="Rating")
model <- loess(rating ~ year, data = movies) # loess regression line
j <- order(movies$year)
lines(movies$year[j], model$fitted[j], col="forestgreen", lwd=2)
points(braveheart$year, braveheart$rating,
     pch = 21, bg = "steelblue", cex = 3)
})
```

Which line do you think will be the slowest?


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/functior
Addins
datacamp
Untitled1*
Profile2
a_script.Rmd
a_slides.Rmd
Profile1
Publish
Flame Graph
Data
Options
<expr>
Memory
Time
1 profvis::profvis({data(movies, package = "ggplot2movies") # Load data
2 braveheart = movies[7288,]
3 movies = movies[movies$Action == 1,]
4 plot(movies$year, movies$rating, xlab = "Year", ylab="Rating")
5 model = loess(rating ~ year, data = movies) # loess regression line
6 j = order(movies$year)
7 lines(movies$year[j], model$fitted[j], col="forestgreen", lwd=2)
8 points(braveheart$year, braveheart$rating,
9 pch = 21, bg = "steelblue", cex = 3)})
10
13.6
40
16.4
10
4.6
100
1.3
190
plot.xy
.C
plot.default
simpleLoess
lazyLoadDBfetch
[
plot
loess
0
50
100
150
200
250
300
Sample Interval: 10ms
340ms


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/functior
Addins
datacamp
Untitled1*
Profile2
a_script.Rmd
a_slides.Rmd
Profile1
Publish
Flame Graph
Data
Options
<expr>
Memory
Time
1 profvis::profvis({data(movies, package = "ggplot2movies") # Load data
2 braveheart = movies[7288,]
3 movies = movies[movies$Action == 1,]
4 plot(movies$year, movies$rating, xlab = "Year", ylab="Rating")
5 model = loess(rating ~ year, data = movies) # loess regression line
6 j = order(movies$year)
7 lines(movies$year[j], model$fitted[j], col="forestgreen", lwd=2)
8 points(braveheart$year, braveheart$rating,
9 pch = 21, bg = "steelblue", cex = 3)})
10
13.6
40
16.4
10
4.6
100
1.3
190
plot.xy
.C
plot.default
simpleLoess
lazyLoadDBfetch
[
plot
loess
0
50
100
150
200
250
300
Sample Interval: 10ms
340ms

Let's practice!

WRITING EFFICIENT R CODE

Profvis

WRITING EFFICIENT R CODE

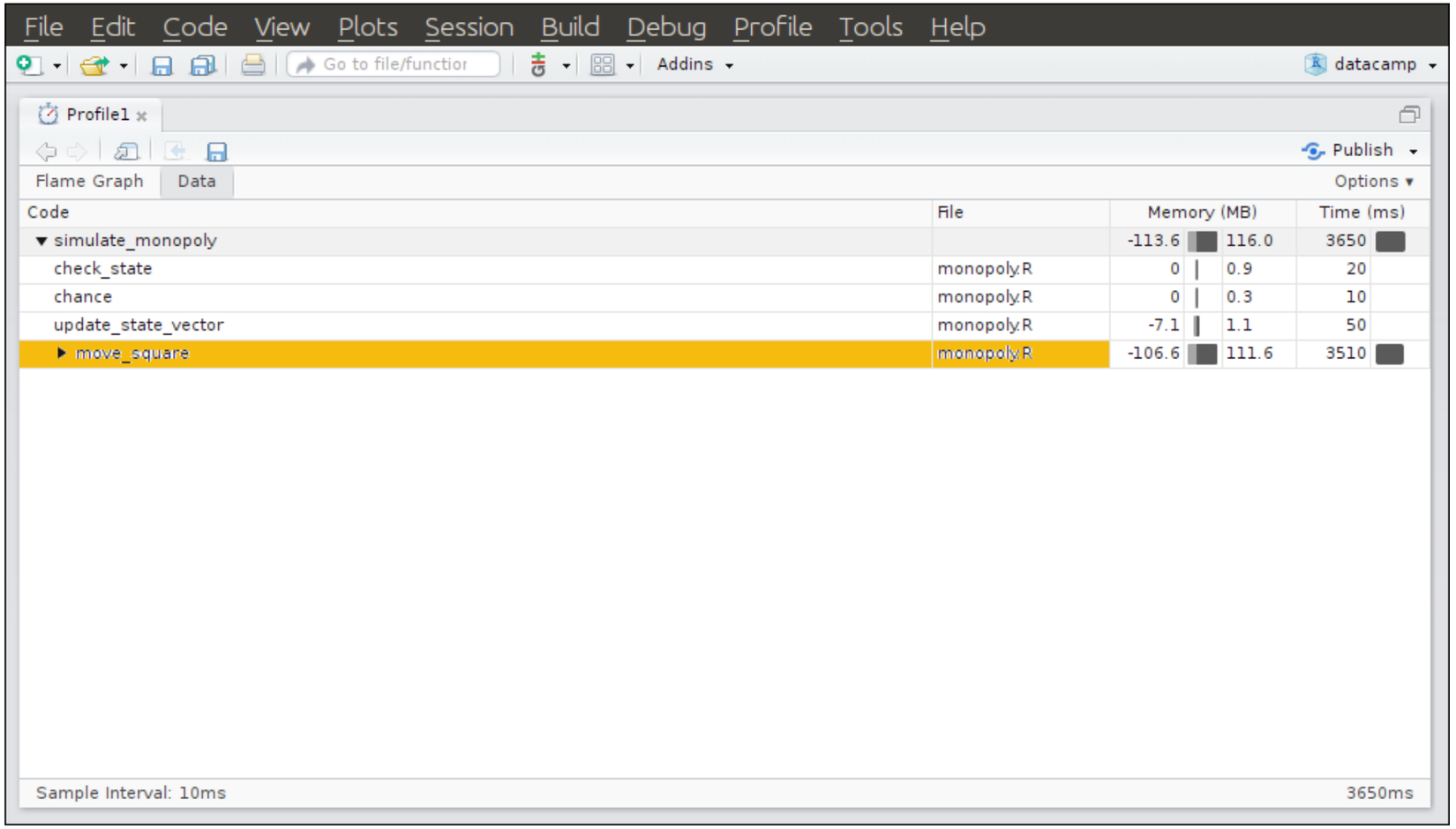
Colin Gillespie
Jumping Rivers & Newcastle University

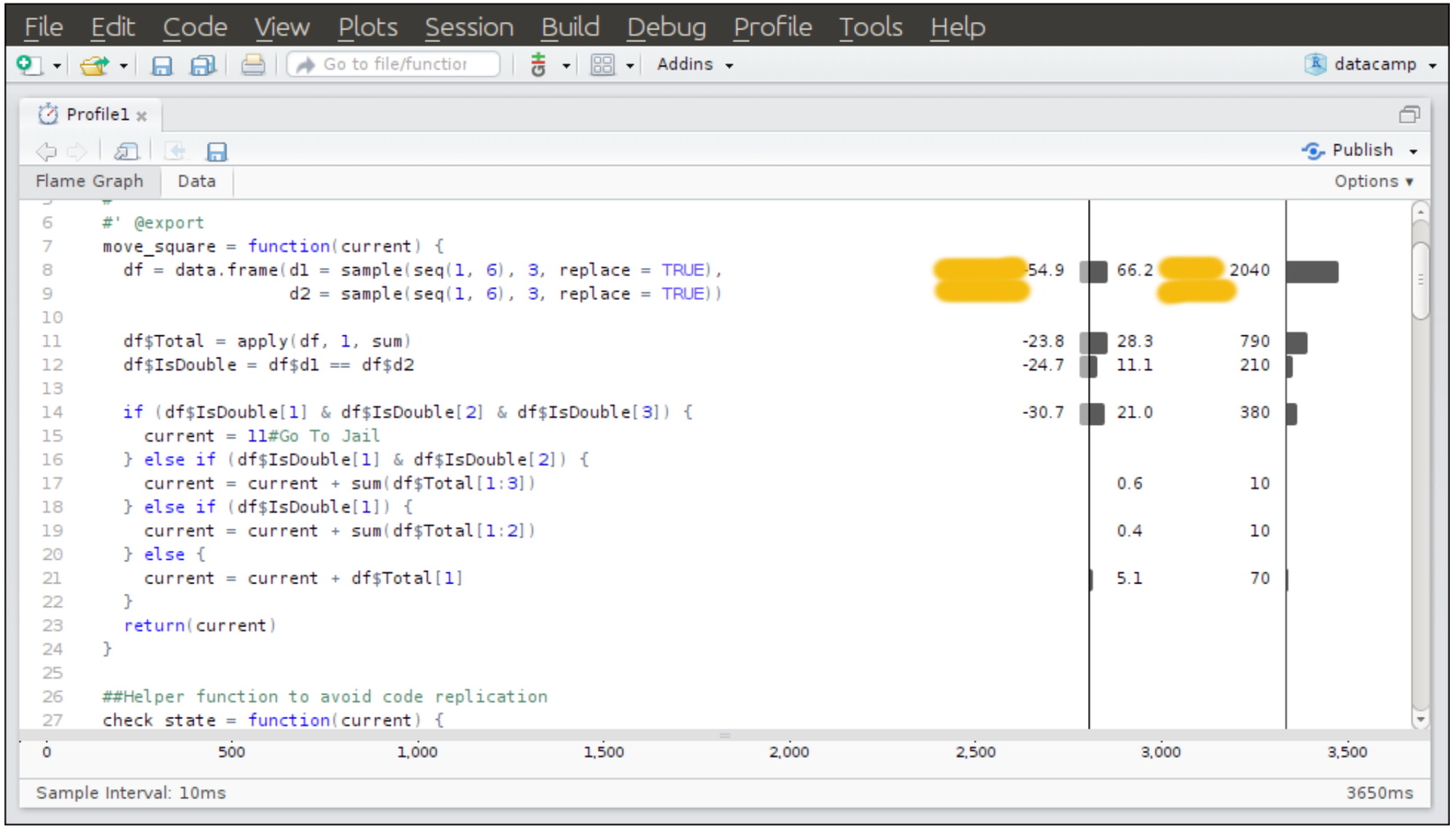
Monopoly

- 40 squares
  - 28 properties (22 streets + 4 stations + 2 utilities)
    - Players take turns moving by rolling dice
  - Buying properties
  - Charging other players
    - Sent to jail: three consecutive doubles in a single turn

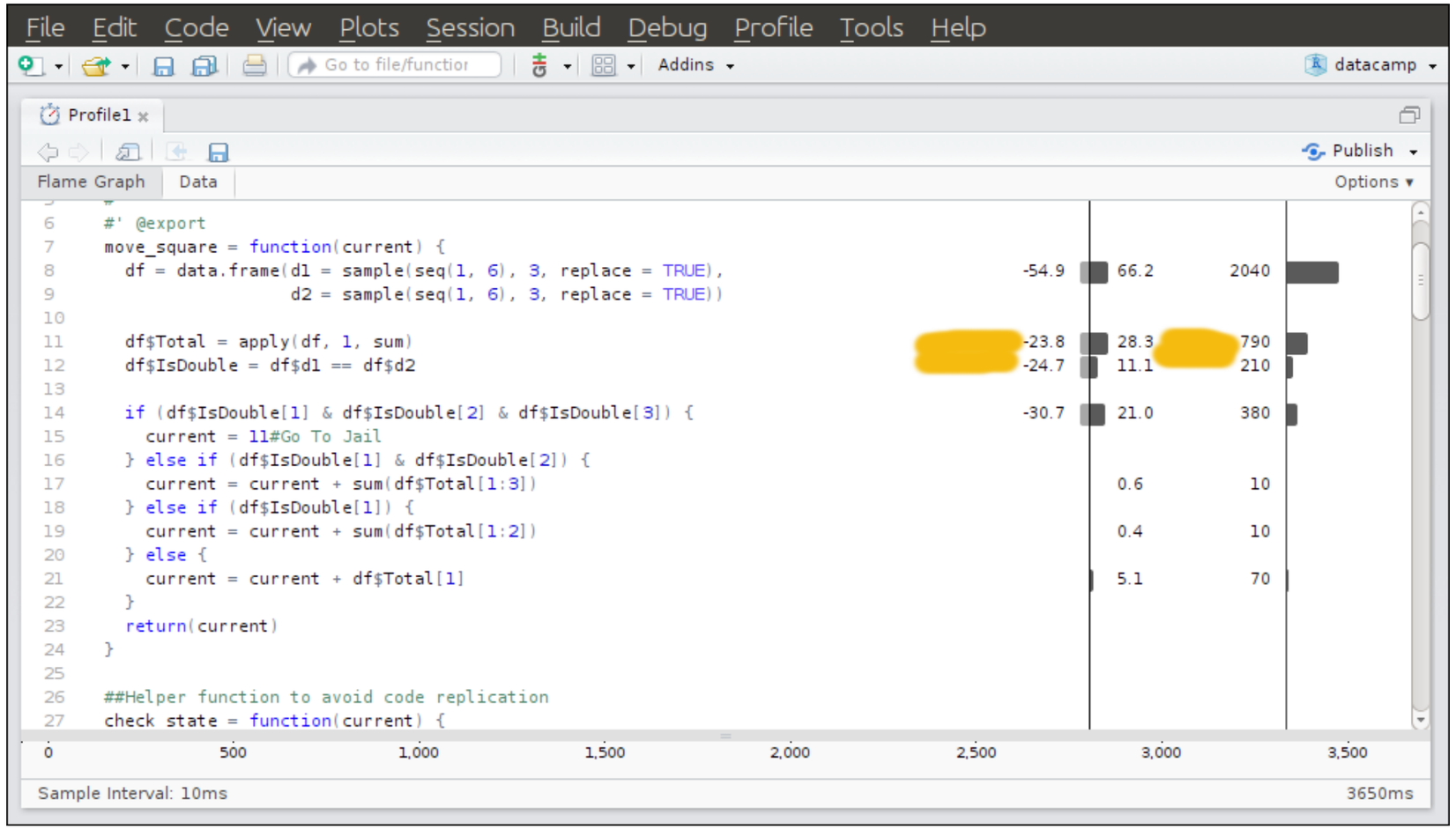
Monopoly Code

- Around 100 lines of code
  - Simplified game
    - Reject the capitalist system: no money
    - No friends, only 1 player
  - `simulate_monopoly(no_of_rolls)`


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/functior
Addins
datacamp
Profile1
Publish
Flame Graph
Data
Options
Code
File
Memory (MB)
Time (ms)
simulate_monopoly
-113.6
116.0
3650
check_state
monopoly.R
0
0.9
20
chance
monopoly.R
0
0.3
10
update_state_vector
monopoly.R
-7.1
1.1
50
move_square
monopoly.R
-106.6
111.6
3510
Sample Interval: 10ms
3650ms


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/functior
Addins
datacamp
Profile1
Publish
Flame Graph
Data
Options
#' @export
move_square = function(current) {
df = data.frame(d1 = sample(seq(1, 6), 3, replace = TRUE),
d2 = sample(seq(1, 6), 3, replace = TRUE))
df$Total = apply(df, 1, sum)
df$IsDouble = df$d1 == df$d2
if (df$IsDouble[1] & df$IsDouble[2] & df$IsDouble[3]) {
current = 11#Go To Jail
} else if (df$IsDouble[1] & df$IsDouble[2]) {
current = current + sum(df$Total[1:3])
} else if (df$IsDouble[1]) {
current = current + sum(df$Total[1:2])
} else {
current = current + df$Total[1]
}
return(current)
}
##Helper function to avoid code replication
check state = function(current) {
54.9
66.2
2040
-23.8
28.3
790
-24.7
11.1
210
-30.7
21.0
380
0.6
10
0.4
10
5.1
70
0
500
1,000
1,500
2,000
2,500
3,000
3,500
Sample Interval: 10ms
3650ms

Monopoly profvis



How would you optimize this code?

Let's practice!

WRITING EFFICIENT R CODE

Monopoly recap

WRITING EFFICIENT R CODE

Colin Gillespie
Jumping Rivers & Newcastle University

Data frames vs. matrices

```
# Original
rolls <- data.frame(d1 = sample(1:6, 3, replace = TRUE),
                    d2 = sample(1:6, 3, replace = TRUE))
```

```
# Updated
rolls <- matrix(sample(1:6, 6, replace = TRUE), ncol = 2)
```

- Total Monopoly simulation time: 2 seconds to 0.5 seconds
- Creating a data frame is slower than a matrix
- In the Monopoly simulation, we created 10,000 data frames

apply vs. rowSums

```
# Original
total <- apply(df, 1, sum)
```

```
# Updated
total <- rowSums(df)
```

- 0.5 seconds to 0.16 seconds - 3 fold speed up

& vs. &&

```
# Original
is_double[1] & is_double[2] & is_double[3]
```

```
# Updated
is_double[1] && is_double[2] && is_double[3]
```

- Limited speed-up
- 0.16 seconds to 0.15 seconds

Overview

Method	Time (secs)	Speed-up
Original	2.00	1.0
Matrix	0.50	4.0
Matrix + rowSums	0.20	10.0
Matrix + rowSums + &&	0.19	10.5

Let's practice!

WRITING EFFICIENT R CODE