Lecture 24: Profiling and microbenchmarking

An order of operations for programming

1. Make it run
2. Make it right
3. Make it fast

When speed matters

- You are working with very large data
- You are running a process (a simulation, a data analysis, etc.) many times
- A piece of code will be called many times (e.g., choosing a split in a decision tree)

Goals

- Learn how to identify bottlenecks in code
- Learn approaches for more efficient code in R
- Time permitting: learn how to use C++ to make code faster

Example: timing code

Suppose we want to compute the mean of each column of a data frame:

```
n <- 100000
cols <- 150
data_mat <- matrix(rnorm(n * cols, mean = 5), ncol = cols)
data <- as.data.frame(data_mat)

means <- rep(NA, cols)
for(i in 1:cols){
  means[i] <- mean(data[,i])
}
```

Example: timing code

Suppose we want to compute the mean of each column of a data frame:

```
n <- 100000
cols <- 150
data_mat <- matrix(rnorm(n * cols, mean = 5), ncol = cols)
data <- as.data.frame(data_mat)

system.time({
  means <- rep(NA, cols)
  for(i in 1:cols){
    means[i] <- mean(data[,i])
  }
})
```

```
   user  system elapsed 
  1.923   0.014   1.939 
```

timing code

(in seconds)

Alternatives

```
means <- rep(NA, cols)
for(i in 1:cols){
  means[i] <- mean(data[,i])
}
```

What are the alternatives to this for-loop approach?

- Is there a built-in function?

  colMeans : function for computing column means of matrices & data frames

- Is there another way to iterate?

  apply : apply a function to the margins of a matrix or data frame

  apply(data, 2, mean)

Alternatives

```r
# Option 1: for loop
for_loop_means <- function(data){
  cols <- ncol(data)
  means <- rep(NA, cols)
  for(i in 1:cols){
    means[i] <- mean(data[,i])
  }
  return(means)
}
means <- for_loop_means(data)

# Option 2: apply
means <- apply(data, 2, mean)

# Option 3: colMeans
means <- colMeans(data)
```

Comparing performance

Microbenchmarking: Evaluating the performance of a small piece of code

useful for benchmarking

```
bench::mark(
  means <- for_loop_means(data),
  means <- apply(data, 2, mean),
  means <- colMeans(data),
  check = F
)
```

```
# A tibble: 3 × 6
  expression                          min   median `itr/sec` mem_alloc
`gc/sec`
  <bch:expr>                     <bch:tm> <bch:tm>     <dbl> <bch:byt>
<dbl>
1 means <- for_loop_means(data)    1.92s    1.92s     0.522    1.85KB
0
2 means <- apply(data, 2, mean)    2.01s    2.01s     0.497  400.57MB
0.993
3 means <- colMeans(data)        450.29ms 462.51ms     2.16   114.45MB
1.08
```

Similar performance

least memory used

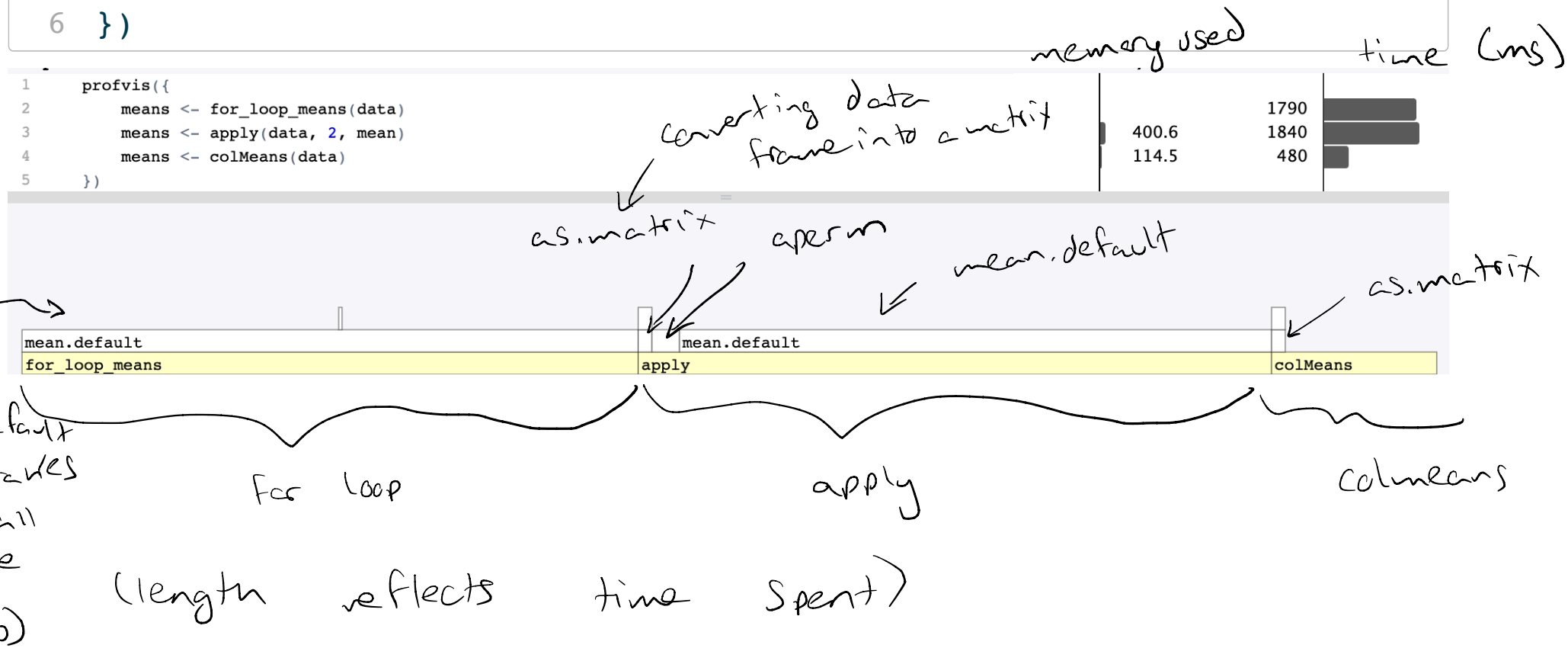
most memory used

about 4 times faster

time it took for the code to run once

Profiling

```
1 library(profvis)
2 profvis({
3   means <- for_loop_means(data)
4   means <- apply(data, 2, mean)
5   means <- colMeans(data)
6 })
```



Space for efficiency increases?

```
1 colMeans
```

```
function (x, na.rm = FALSE, dims = 1L)
{
    if (is.data.frame(x))
        x <- as.matrix(x)
    if (!is.array(x) || length(dn <- dim(x)) < 2L)
        stop("'x' must be an array of at least two dimensions")
    if (dims < 1L || dims > length(dn) - 1L)
        stop("invalid 'dims'")
    n <- prod(dn[id <- seq_len(dims)])
    dn <- dn[-id]
    z <- if (is.complex(x))
        .Internal(colMeans(Re(x), n, prod(dn), na.rm)) + (0+1i) *
            .Internal(colMeans(Im(x), n, prod(dn), na.rm))
    else .Internal(colMeans(x, n, prod(dn), na.rm))
```

checking inputs / doing transformations

Increase efficiency by avoiding extraneous steps

```
1  n <- 100000
2  cols <- 150
3  data_mat <- matrix(rnorm(n * cols, mean = 5), ncol = cols)
4  data <- as.data.frame(data_mat)
5
6  bench::mark(
7    means <- colMeans(data_mat),
8    means <- colMeans(data),
9    check = F
10 )
```

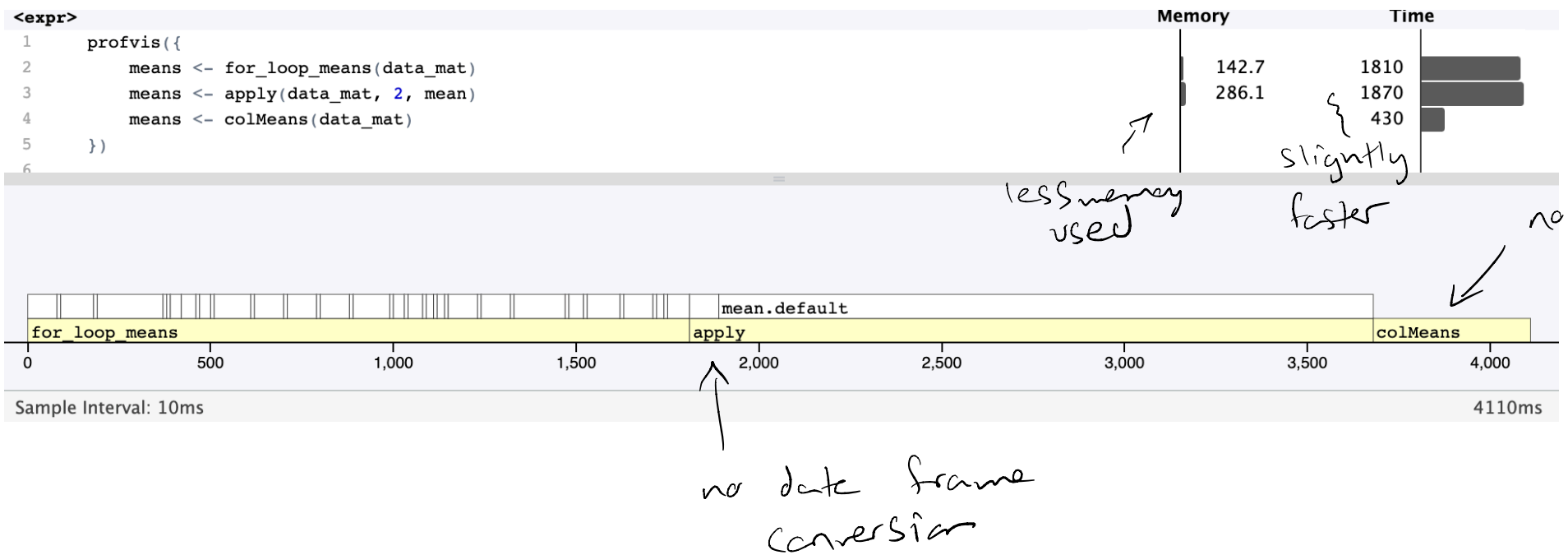
```
# A tibble: 2 × 6
  expression                      min   median `itr/sec` mem_alloc
`gc/sec`
  <bch:expr>                  <bch:tm> <bch:tm>     <dbl> <bch:byt>
<dbl>
1 means <- colMeans(data_mat)    434ms    436ms      2.29    1.22KB
0
2 means <- colMeans(data)        450ms    450ms      2.22  114.45MB
2.22
```

slightly faster

less memory used

Profiling

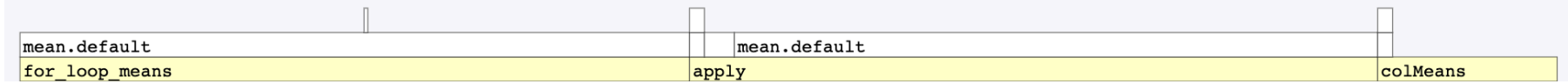
```
1 profvis({
2   means <- for_loop_means(data_mat)
3   means <- apply(data_mat, 2, mean)
4   means <- colMeans(data_mat)
5 })
```



Profiling

```
1  profvis({
2      means <- for_loop_means(data)
3      means <- apply(data, 2, mean)
4      means <- colMeans(data)
5  })
```

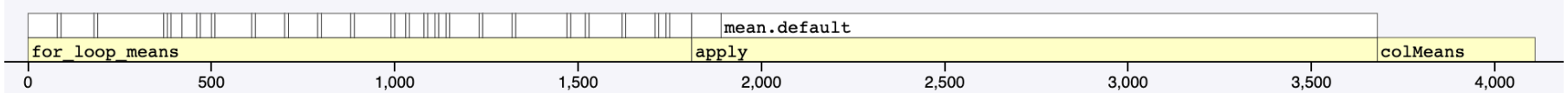
400.6	1790
114.5	1840
	480



<expr>

```
1  profvis({
2      means <- for_loop_means(data_mat)
3      means <- apply(data_mat, 2, mean)
4      means <- colMeans(data_mat)
5  })
```

Memory	Time
142.7	1810
286.1	1870
	430



Sample Interval: 10ms

4110ms

Class activity

https://sta279-s24.github.io/class_activities/ca_lecture_24.html