

# Lecture 22: 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:

```
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 means <- rep(NA, cols)
7 for(i in 1:cols){
8   means[i] <- mean(data[,i])
9 }
```

# Example: timing code

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

```
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  system.time({
7    means <- rep(NA, cols)
8    for(i in 1:cols){
9      means[i] <- mean(data[,i])
10   }
11 })
```

user	system	elapsed
1.930	0.017	1.960

# Alternatives

```
1 means <- rep(NA, cols)
2 for(i in 1:cols){
3   means[i] <- mean(data[,i])
4 }
```

What are the alternatives to this for-loop approach?

# Alternatives

```
1 # Option 1: for loop
2 for_loop_means <- function(data){
3   cols <- ncol(data)
4   means <- rep(NA, cols)
5   for(i in 1:cols){
6     means[i] <- mean(data[,i])
7   }
8   return(means)
9 }
10 means <- for_loop_means(data)
11
12 # Option 2: apply
13 means <- apply(data, 2, mean)
14
15 # Option 3: colMeans
16 means <- colMeans(data)
```



# Comparing performance

**Microbenchmarking:** Evaluating the performance of a small piece of code

```
1 bench::mark(  
2   means <- for_loop_means(data),  
3   means <- apply(data, 2, mean),  
4   means <- colMeans(data),  
5   check = F  
6 )
```

# A tibble: 3 × 6

expression	min	median	`itr/sec`	mem_alloc
<code>&lt;bch:expr&gt;</code>	<code>&lt;bch:tm&gt;</code>	<code>&lt;bch:tm&gt;</code>	<code>&lt;dbl&gt;</code>	<code>&lt;bch:byt&gt;</code>
<code>&lt;dbl&gt;</code>				
1 means <- for_loop_means(data)	1.93s	1.93s	0.519	1.85KB
0				
2 means <- apply(data, 2, mean)	2.03s	2.03s	0.493	400.57MB
0.987				
3 means <- colMeans(data)	461.4ms	469.34ms	2.13	114.45MB
1.07				

# Profiling

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

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

	1790
400.6	1840
114.5	480

mean.default  
for\_loop\_means

mean.default  
apply

colMeans

# 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))
}
```

# Increase efficiency by avoiding extraneous steps

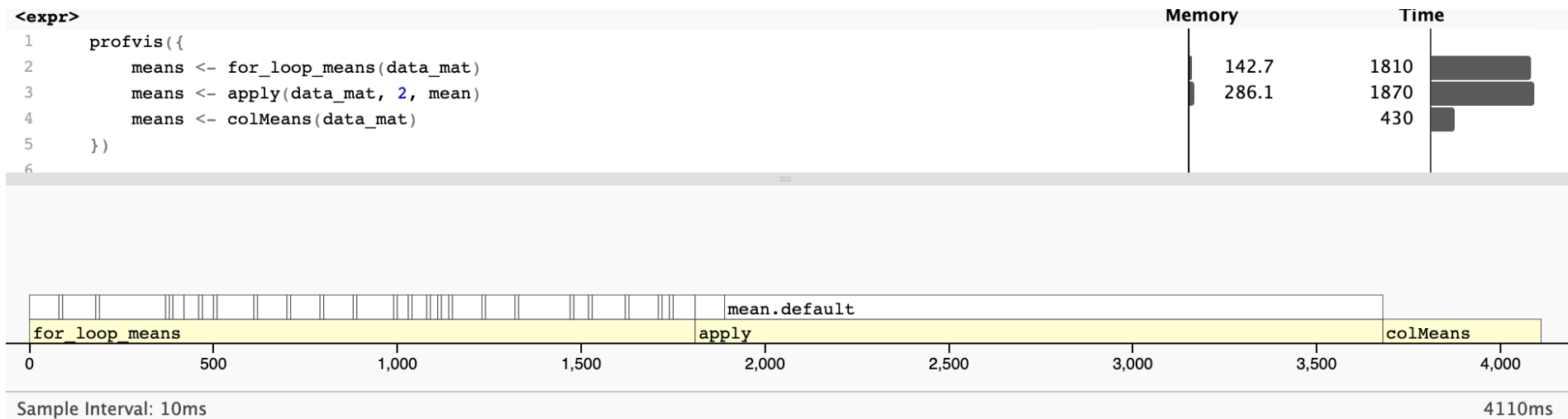
```
1 n <- 100000
2 cols <- 150
3 data_mat <- matrix(rnorm(n * cols, mean = 5), ncol = cols)
4 data <- as.data.frame(data_mat)
5
6 bench::mark(
7   means <- colMeans(data_mat),
8   means <- colMeans(data),
9   check = F
10 )
```

# A tibble: 2 × 6

expression	min	median	`itr/sec`	mem_alloc
<code>`gc/sec`</code>				
<code>&lt;bch:expr&gt;</code>	<code>&lt;bch:tm&gt;</code>	<code>&lt;bch:tm&gt;</code>	<code>&lt;dbl&gt;</code>	<code>&lt;bch:byt&gt;</code>
<code>&lt;dbl&gt;</code>				
1 means <- colMeans(data_mat)	437ms	437ms	2.29	1.22KB
0				
2 means <- colMeans(data)	453ms	453ms	2.21	114.45MB
2.21				

# Profiling

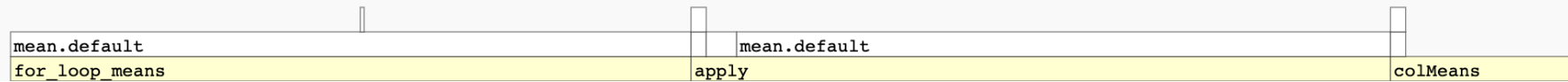
```
1 profvis({  
2   means <- for_loop_means(data_mat)  
3   means <- apply(data_mat, 2, mean)  
4   means <- colMeans(data_mat)  
5 })
```



# Profiling

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

	1790	
400.6	1840	
114.5	480	



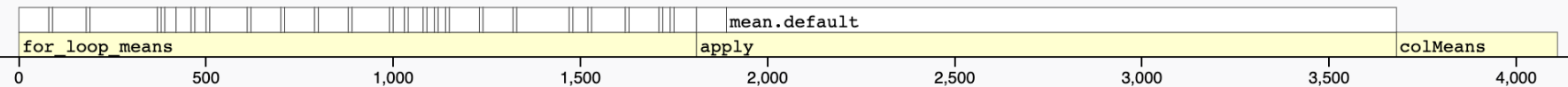
<expr>

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

Memory

Time

142.7	1810	
286.1	1870	
	430	



Sample Interval: 10ms

4110ms

# Class activity

[https://sta279-f23.github.io/class\\_activities/ca\\_lecture\\_24.html](https://sta279-f23.github.io/class_activities/ca_lecture_24.html)

