

# Memory and I/O

# Statistical Computing & Programming

Shawn Santo

# Supplementary materials

Full video lecture available in Zoom Cloud Recordings

Additional resources

- [Chapter 2](#), Advanced R by Wickham, H.
- [vroom vignette](#)

# Memory basics

# Names and values

In R, a name has a value. It is not the value that has a name.

For example, in

```
x <- c(-3, 4, 1)
```

the object named `x` is a reference to vector `c(-3, 4, 1)`.



We can see where this lives in memory with

```
library(lobstr)
lobstr::obj_addr(x)
```

```
#> [1] "0x7f95dd9f4f08"
```

and its size with

```
lobstr::obj_size(x)
```

```
#> 80 B
```

# Copy-on-modify: atomic vectors

Understanding when R creates a copy of an object will allow you to write faster code. This is also important to keep in mind when working with very large vectors.

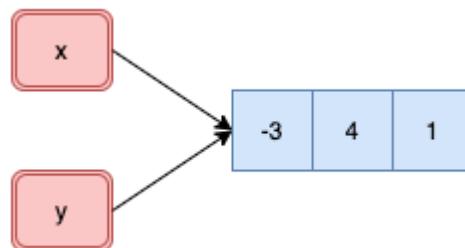
```
x <- c(-3, 4, 1)  
y <- x
```

```
obj_addr(x)
```

```
#> [1] "0x7f95e12bd818"
```

```
obj_addr(y)
```

```
#> [1] "0x7f95e12bd818"
```



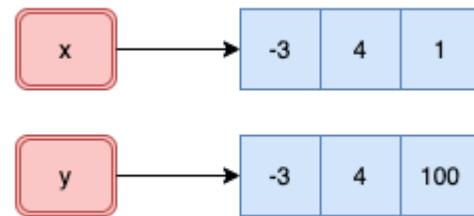
```
y[3] <- 100
```

```
obj_addr(x)
```

```
#> [1] "0x7f95e12bd818"
```

```
obj_addr(y)
```

```
#> [1] "0x7f95ddc5ec78"
```



```
x <- c(0, 1, 9)  
y <- x  
  
obj_addr(x)
```

```
#> [1] "0x7f95e1367928"
```

```
obj_addr(y)
```

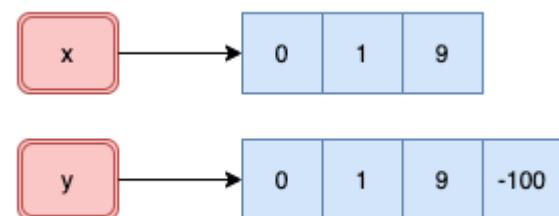
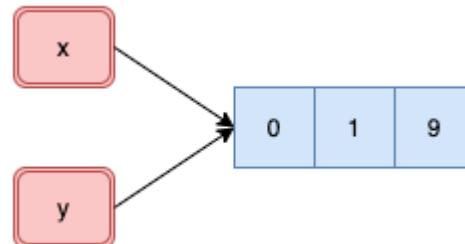
```
#> [1] "0x7f95e1367928"
```

```
y[4] <- -100  
obj_addr(x)
```

```
#> [1] "0x7f95e1367928"
```

```
obj_addr(y)
```

```
#> [1] "0x7f95df538838"
```



Even though only one component changed in the atomic vector y, R created a new object as seen by the new address in memory.

# Memory tracking

Function `tracemem()` marks an object so that a message is printed whenever the internal code copies the object. Let's see when `x` gets copied.

```
x <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
tracemem(x)
```

```
#> [1] "<0x7f95df23efa8>"
```

```
y <- x
```

```
y[1] <- 0
```

```
#> tracemem[0x7f95df23efa8 -> 0x7f95e12fef28]: eval eval withVisible withCallingH
```

```
x
```

```
#> [1] 0 1 1 2 3 5 8 13 21 34
```

```
y
```

```
#> [1] 0 1 1 2 3 5 8 13 21 34
```

```
c(obj_addr(x), obj_addr(y))
```

```
#> [1] "0x7f95df23efa8" "0x7f95e12fef28"
```

```
x[1] <- 0
```

```
lobstr::ref(x)
```

```
#> [1:0x7f95df23efa8] <dbl>
```

```
lobstr::ref(y)
```

```
#> [1:0x7f95e12fef28] <dbl>
```

```
untracemem(x)
```

# Copy-on-modify: lists

```
x <- list(a = 1, b = 2, c = 3)
obj_addr(x)
```

```
#> [1] "0x7f95e51d6528"
```

```
y <- x
```

```
c(obj_addr(x), obj_addr(y))
```

```
#> [1] "0x7f95e51d6528" "0x7f95e51d6528"
```

```
ref(x, y)
```

```
#> ┌─ [1:0x7f95e51d6528] <named list>
#>   ├─ a = [2:0x7f95e51565e0] <dbl>
#>   ├─ b = [3:0x7f95e51565a8] <dbl>
#>   └─ c = [4:0x7f95e5156570] <dbl>
#>
#> [1:0x7f95e51d6528]
```

```
y$c <- 4
```

```
ref(x, y)
```

```
#> [1:0x7f95e51d6528] <named list>
#>   a = [2:0x7f95e51565e0] <dbl>
#>   b = [3:0x7f95e51565a8] <dbl>
#>   c = [4:0x7f95e5156570] <dbl>
#>
#> [5:0x7f95e44709d8] <named list>
#>   a = [2:0x7f95e51565e0]
#>   b = [3:0x7f95e51565a8]
#>   c = [6:0x7f95e44343c8] <dbl>
```

```
x <- list(a = 1, b = 2, c = 3)
y <- x
```

```
c(obj_addr(x), obj_addr(y))
```

```
#> [1] "0x7f95e60143e8" "0x7f95e60143e8"
```

```
y$d <- 9
ref(x, y)
```

```
#> ┌─ [1:0x7f95e60143e8] <named list>
#>   ├─ a = [2:0x7f95e47bafe0] <dbl>
#>   ├─ b = [3:0x7f95e47bafa8] <dbl>
#>   └─ c = [4:0x7f95e47baf70] <dbl>
#>
#> ┌─ [5:0x7f95e5918de8] <named list>
#>   ├─ a = [2:0x7f95e47bafe0]
#>   ├─ b = [3:0x7f95e47bafa8]
#>   ├─ c = [4:0x7f95e47baf70]
#>   └─ d = [6:0x7f95e58b39a8] <dbl>
```

R creates a shallow copy. Shared components exist with elements a, b, and c.

# Copy-on-modify: data frames

```
library(tidyverse)
x <- tibble(a = 1:3, b = 9:7)
```

```
ref(x)
```

```
#> ─ [1:0x7f95e6aca5c8] <tibble[,2]>
#>   a = [2:0x7f95e429f838] <int>
#>   b = [3:0x7f95e42b86a8] <int>
```

```
y <- x %>%
  mutate(b = b ^ 2)
```

```
ref(x, y)
```

```
#> ─ [1:0x7f95e6aca5c8] <tibble[,2]>
#>   a = [2:0x7f95e429f838] <int>
#>   b = [3:0x7f95e42b86a8] <int>
#>
#> ─ [4:0x7f95e815af88] <tibble[,2]>
#>   a = [2:0x7f95e429f838]
#>   b = [5:0x7f95e8113a88] <dbl>
```

```
z <- x  
ref(x, z)
```

```
#> [1:0x7f95e6aca5c8] <tibble[,2]>  
#> | a = [2:0x7f95e429f838] <int>  
#> | b = [3:0x7f95e42b86a8] <int>  
#>  
#> [1:0x7f95e6aca5c8]
```

```
z <- x %>%  
  add_row(a = -1, b = -1)
```

```
ref(x, z)
```

```
#> [1:0x7f95e6aca5c8] <tibble[,2]>  
#> | a = [2:0x7f95e429f838] <int>  
#> | b = [3:0x7f95e42b86a8] <int>  
#>  
#> [4:0x7f95e7d48848] <tibble[,2]>  
#> | a = [5:0x7f95e7d4adb8] <dbl>  
#> | b = [6:0x7f95e7d4ad68] <dbl>
```

If you modify a column, only that column needs to be copied in memory. However, if you modify a row, the entire data frame is copied in memory.

# Exercise

Can you diagnose what is going on below? Why are two copies being made?

```
x <- c(1L, 2L, 3L)
tracemem(x)
```

```
y <- x
```

```
y[3] <- 4L
untracemem(x)
```

```
tracemem[0x7fb729b374c8 -> 0x7fb729b373c8] :
tracemem[0x7fb729b373c8 -> 0x7fb72320e9a8] :
```

# Object size

Object sizes can sometimes be deceiving.

```
x <- rnorm(1e6)
y <- 1:1e6
z <- seq(1, 1e6, by = 1)
s <- (1:1e6) / 2
```

```
c(obj_size(x), obj_size(y), obj_size(z), obj_size(s))
```

```
#> * 8,000,048 B
#> *          680 B
#> * 8,000,048 B
#> * 8,000,048 B
```

```
c(obj_size(c(1L)), obj_size(c(1.0)))
```

```
#> * 56 B  
#> * 56 B
```

```
c(obj_size(c(1L, 2L)), obj_size(as.numeric(c(1.0, 2.0))))
```

```
#> * 56 B  
#> * 64 B
```

```
c(obj_size(c(1L, 2L, 3L)), obj_size(as.numeric(c(1.0, 2.0, 3.0))))
```

```
#> * 64 B  
#> * 80 B
```

```
c(obj_size(integer(10000)), obj_size(numeric(10000)))
```

```
#> * 40,048 B  
#> * 80,048 B
```

There is overhead with creating vectors in R. Take a look at `?Memory` if you want to dig deeper as to the overhead cost.

# Exercise

Starting from 0 we can see that

```
lobstr::obj_size(integer(0))
```

```
#> 48 B
```

```
lobstr::obj_size(numeric(0))
```

```
#> 48 B
```

are both 48 bytes. Based on the results on the next slide can you deduce how R handles these numeric data in memory?

```
diff(sapply(0:100, function(x) lobstr::obj_size(integer(x))))
```

```
#> [1] 8 0 8 0 16 0 0 0 16 0 0 0 16 0 0 0 64 0 0 0 0 0 0 0 0 0  
#> [26] 0 0 0 0 0 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8  
#> [51] 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8  
#> [76] 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8
```

```
c(obj_size(integer(20)), obj_size(integer(22)))
```

```
#> * 176 B  
#> * 176 B
```

```
diff(sapply(0:100, function(x) lobstr::obj_size(numeric(x))))
```

```
#> [1] 8 8 16 0 16 0 16 0 64 0 0 0 0 0 0 0 8 8 8 8 8 8 8 8 8 8  
#> [26] 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
#> [51] 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
#> [76] 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
```

```
c(obj_size(numeric(10)), obj_size(numeric(14)))
```

```
#> * 176 B  
#> * 176 B
```

# I/O medium data

# Getting medium data into R

Dimensions: 3,185,906 x 9

```
url <- "http://www2.stat.duke.edu/~sms185/data/bike/cbs_2015.csv"
```

```
system.time({x <- read.csv(url)})
```

user	system	elapsed
29.739	1.085	37.321

```
system.time({x <- readr::read_csv(url)})
```

```
Parsed with column specification:  
cols(  
  Duration = col_double(),  
  `Start date` = col_datetime(format = ""),  
  `End date` = col_datetime(format = ""),  
  `Start station number` = col_double(),  
  `Start station` = col_character(),  
  `End station number` = col_double(),  
  `End station` = col_character(),  
  `Bike number` = col_character(),  
  `Member type` = col_character()  
)  
=====| 100% 369 MB  
     user   system elapsed  
12.773  1.727 22.327
```

```
system.time({x <- data.table::fread(url)})
```

```
trying URL 'http://www2.stat.duke.edu/~sms185/data/bike/cbs_2015.csv'
Content type 'text/csv' length 387899567 bytes (369.9 MB)
=====
downloaded 369.9 MB

  user  system elapsed
 7.363   2.009 19.942
```

```
system.time({x <- vroom::vroom(url)})
```

```
Observations: 3,185,906
Variables: 9
chr [4]: Start station, End station, Bike number, Member type
dbl [3]: Duration, Start station number, End station number
dttm [2]: Start date, End date

Call `spec()` for a copy-pastable column specification
Specify the column types with `col_types` to quiet this message

  user  system elapsed
 5.873   2.361 18.606
```

# Getting bigger data into R

Dimensions: 10,277,677 x 9

```
url <- "http://www2.stat.duke.edu/~sms185/data/bike/full.csv"
```

```
system.time({x <- read.csv(url)})
```

user	system	elapsed
119.472	5.037	139.214

```
system.time({x <- readr::read_csv(url)})
```

Parsed with column specification:

```
cols(  
  Duration = col_double(),  
  `Start date` = col_datetime(format = ""),  
  `End date` = col_datetime(format = ""),  
  `Start station number` = col_double(),  
  `Start station` = col_character(),  
  `End station number` = col_double(),  
  `End station` = col_character(),  
  `Bike number` = col_character(),  
  `Member type` = col_character()  
)  
=====| 100% 1191 MB  
     user   system elapsed  
46.845  7.607  87.425
```

```
system.time({x <- data.table::fread(url)})
```

```
trying URL 'http://www2.stat.duke.edu/~sms185/data/bike/full.csv'
Content type 'text/csv' length 1249306730 bytes (1191.4 MB)
=====
downloaded 1191.4 MB

|-----|
|=====|
  user   system elapsed
33.402    7.249   79.806
```

```
system.time({x <- vroom::vroom(url)})
```

```
Observations: 10,277,677
Variables: 9
chr [4]: Start station, End station, Bike number, Member type
dbl [3]: Duration, Start station number, End station number
dttm [2]: Start date, End date

Call `spec()` for a copy-pastable column specification
Specify the column types with `col_types` to quiet this message
  user   system elapsed
18.837    6.731   57.203
```

# Summary

Function	Elapsed Time (s)
<code>vroom::vroom()</code>	~57
<code>data.table::fread()</code>	~80
<code>readr::read_csv()</code>	~87
<code>read.csv()</code>	~139

Observations: 10,277,677

Variables: 9

# Going forward

# Big data strategies

1. Avoid unnecessary copies of large objects
2. Downsample - you can't exceed  $2^{31} - 1$  rows, columns, or components
  - Downsample to visualize and use summary statistics
  - Downsample to wrangle and understand
  - Downsample to model
3. Get more RAM - this is not easy or even sometimes an option
4. Parallelize - this is not always an option
  - Execute a chunk and pull strategy

# References

1. Read and Write Rectangular Text Data Quickly. (2021). <https://vroom.r-lib.org/>
2. Wickham, H. (2021). Advanced R. <https://adv-r.hadley.nz/>