Data Handling: Import, Cleaning and Visualisation

Lecture 3: A Brief Introduction to Data and Data Processing

Dr. Aurélien Sallin

2024-03-10

Recap

Goals of last course and exercise session

- Basic programming concepts.
- Functions and functionals.
- Working installation of R and first coding exercises.

Basic programming concepts

- Values, variables
- Vectors
- Matrices
- Loops
- Logical statements
- Control statements
- Functions
- Functionals

Three tutorials

- Compute the mean with your own function -> no solution posted
- Evolution in action: fast and slow sloths -> solution in the first exercise session
- Append and lists -> solution in the second exercise session

Warm-up

Warm-up

Consider the following vector. Which answers are true?

```
some_numbers <- c(30, 50, 60)
```

Possible answers

1. `sum(some_numbers[c(2,3)]) == 80`
2. `some_numbers[some_numbers %in% c(30)] == 30`
3. `some_numbers * 5` will do `30*5 + 50*5 + 60*5`

Warm-up

We create the following matrix

```r
mymatrix <- matrix(c(1,2,3,11,12,13,1,10),
                   nrow = 2,
                   ncol = 4,
                   byrow = FALSE)
```

Which of the answers are correct?

Answer 1 (recap lecture 2)

```r
max(mymatrix[,2]) == 3
```

Answer 2 (recap lecture 2)

```r
apply(mymatrix, MARGIN = 2, min)
```

returns a vector of size 4.

Warm-up

We create the following matrix

```
mymatrix <- matrix(c(1,2,3,11,12,13,1,10),
                   nrow = 2,
                   ncol = 4,
                   byrow = FALSE)
```

Which of the answers are correct?

Answer 3 (🌶)

```
m <- apply(mymatrix, MARGIN = 2, min)
```

can be coded as

```
i <- 1
m <- c()

while(i < (ncol(mymatrix) + 1)){
  m <- append(m, min(mymatrix[,i]))
  i <- i + 1
}
```

Warm-up

We create the following matrix

```
mymatrix <- matrix(c(1,2,3,11,12,13,1,10),
                   nrow = 2,
                   ncol = 4,
                   byrow = FALSE)
```

Which of the answers are correct?

Answer 4 (🌶)

In this code:

```
i <- 1
m <- c()

while(i < (ncol(mymatrix) + 1)){
  m <- append(m, min(mymatrix[,i]))
  i <- i + 1
}
```

we actually don't need to create an empty vector m. We just need to save the output of the loop as an object, i.e.:

```
i <- 1

m <- while(i < (ncol(mymatrix) + 1)){
append(m, min(mymatrix[,i]))
i <- i + 1
}
```

Warm-up

What is `total_sum`?

```r
numbers <- 1:4
total_sum <- 0
n <- length(numbers)

# start loop
for (i in 1:n) {

  if(i %% 2 == 0){
     total_sum <- total_sum + numbers[i]
  } else {
    total_sum <- total_sum + 2*numbers[i]
  }

}
```

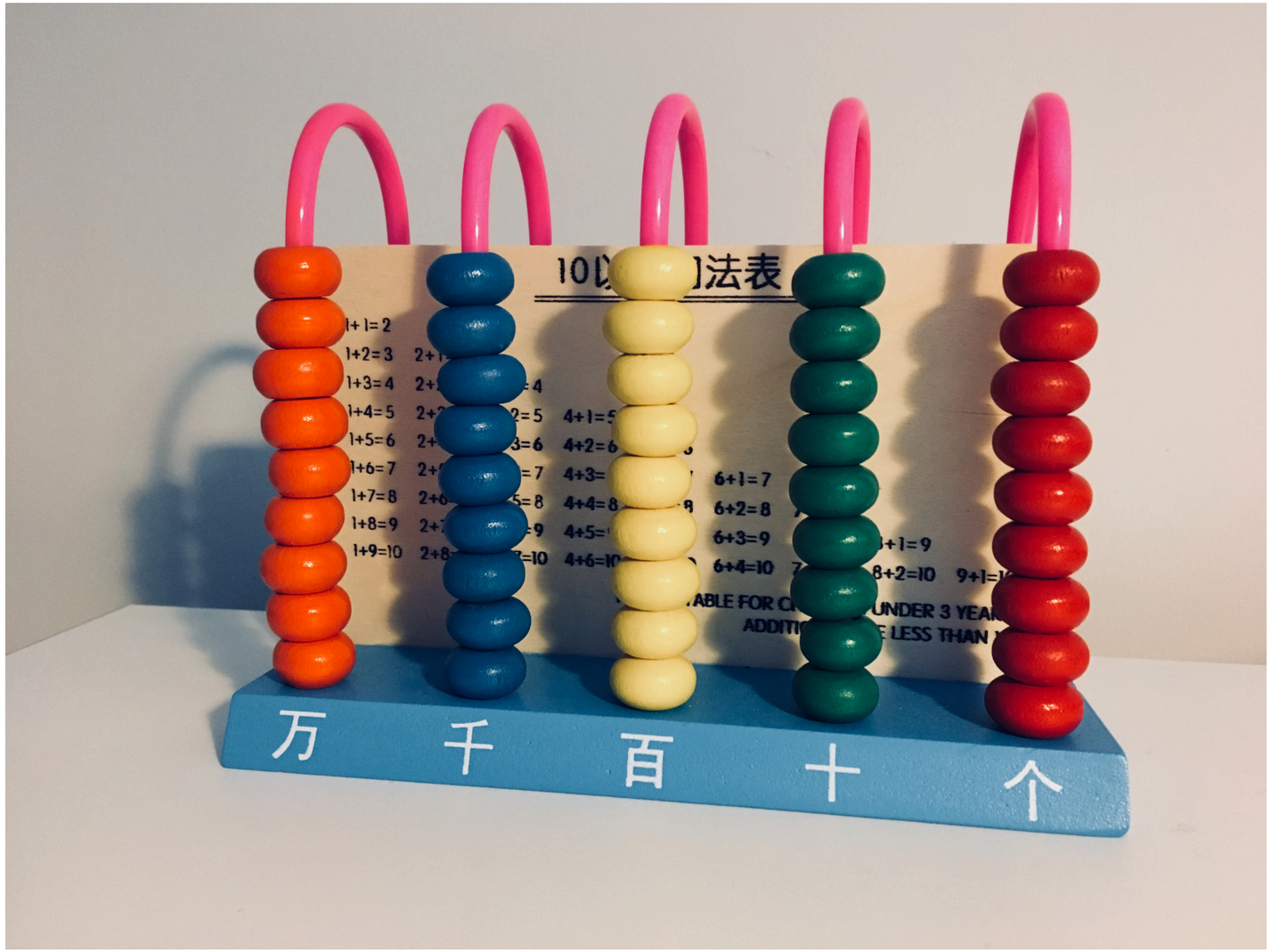
Don’t forget…

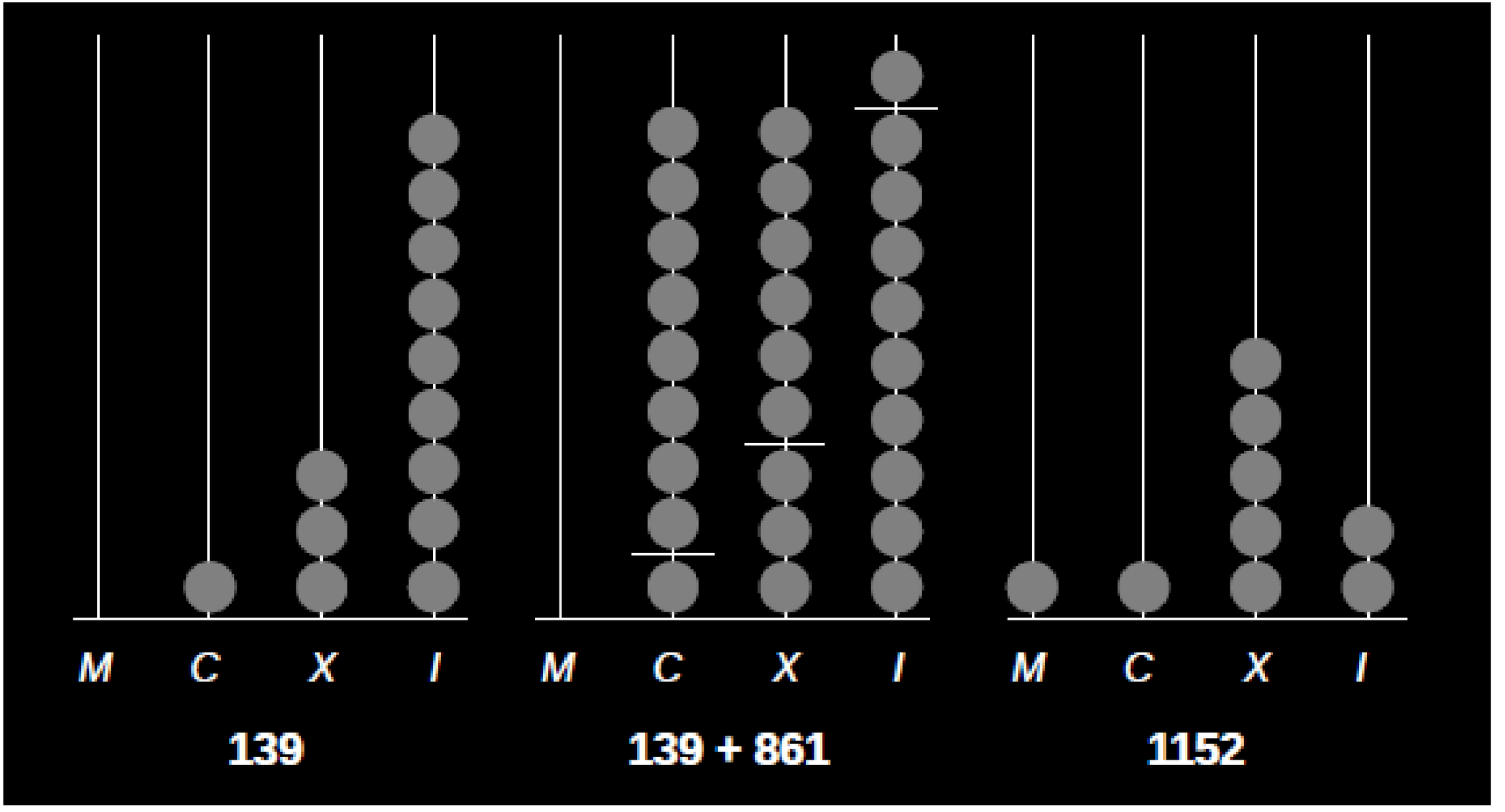
Data Processing

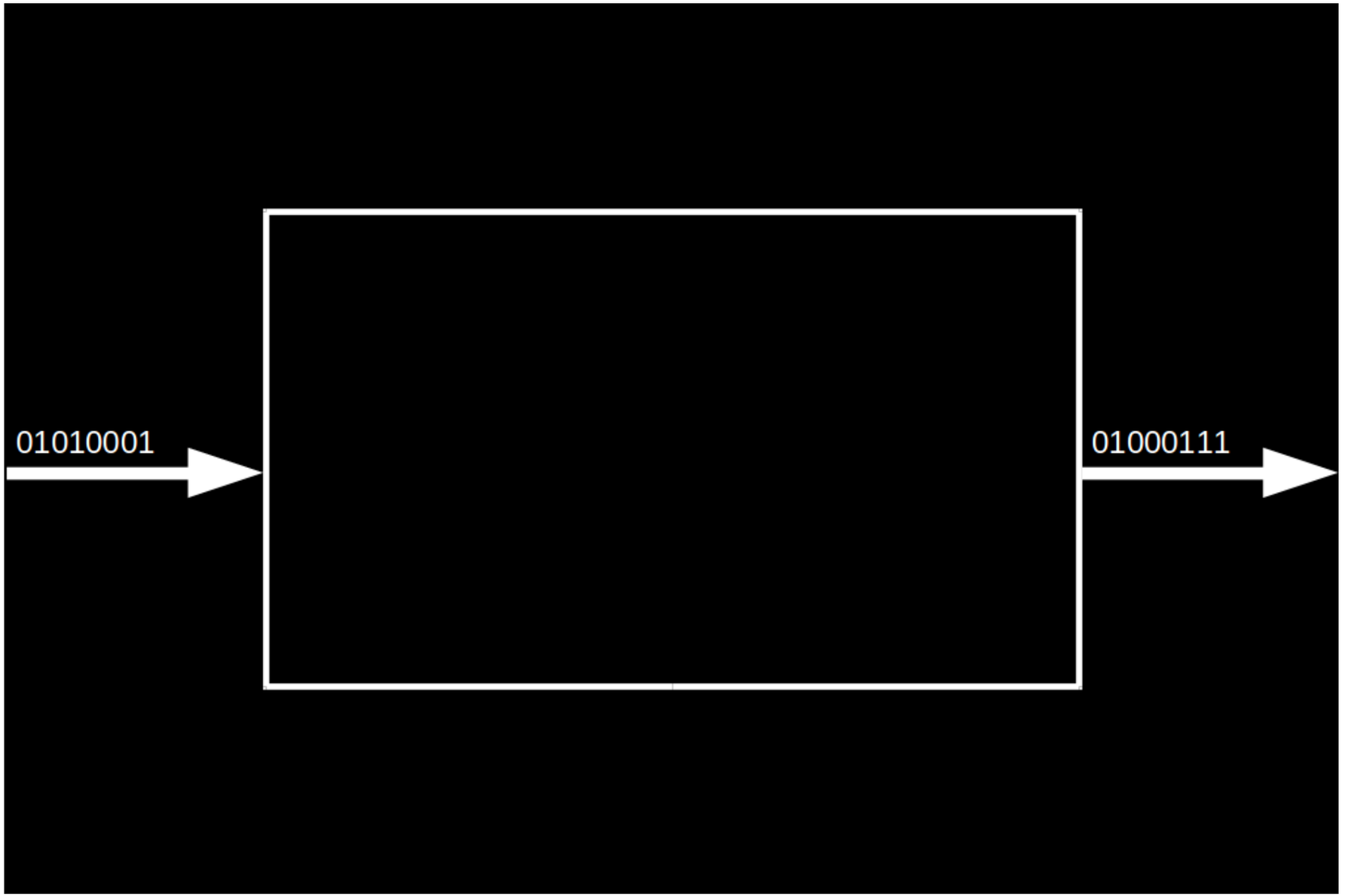
Goals for today

1. Have a basic understanding of data processing;
2. Understand binary and hexadecimal systems;
3. Understand the importance of encoding for data projects.

Numeral systems as representations



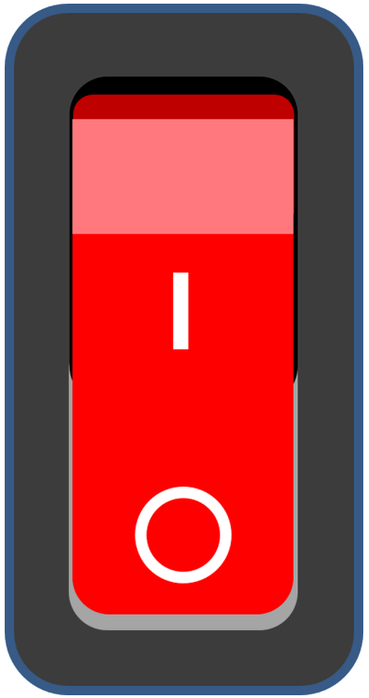

M C X I
M C X I
M C X I
139
139 + 861
1152


01010001
01000111

The binary sytem

Microprocessors can only represent two signs (states):

- ‘Off’ = `0`
- ‘On’ = `1`



The binary sytem

The binary counting frame

- Only two signs: `0`, `1`.
- Base 2.
- Columns: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, and so forth.

The binary counting frame

What is the decimal number *139* in the binary counting frame?

The binary counting frame

What is the decimal number *139* in the binary counting frame?

- Solution:

$$(1 \times 2^7) + (1 \times 2^3) + (1 \times 2^1) + (1 \times 2^0) = 139.$$

The binary counting frame

What is the decimal number *139* in the binary counting frame?

- Solution:

$$(1 \times 2^7) + (1 \times 2^3) + (1 \times 2^1) + (1 \times 2^0) = 139.$$

- More precisely:

$$(1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (1 \times 2^3)$$
$$+ (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 139.$$

- That is, the number `139` in the decimal system corresponds to `10001011` in the binary system.

Conversion between binary and decimal

Number	128	64	32	16	8	4	2	1

Conversion between binary and decimal

Number	128	64	32	16	8	4	2	1
0 =	0	0	0	0	0	0	0	0
1 =	0	0	0	0	0	0	0	1
2 =	0	0	0	0	0	0	1	0
3 =	0	0	0	0	0	0	1	1
…								
139 =	1	0	0	0	1	0	1	1

The binary counting frame

- Sufficient to represent all *natural* numbers in the decimal system.

The binary counting frame

- Sufficient to represent all *natural* numbers in the decimal system.
- Representing fractions is tricky
  - e.g. $1/3 = 0.333..$ actually constitutes an infinite sequence of 0s and 1s.
  - Solution: ‘floating point numbers’ (not 100% accurate)

Floating point numbers

```r
# Subtracting two nearly identical floating-point nu
x <- 0.3 - 0.2
y <- 0.1

# Check if they are equal
result <- x == y
```

```r
print(x)
```

```
[1] 0.1
```

```r
print(y)
```

```
[1] 0.1
```

Floating point numbers

A function to convert a decimal number to binary:

▶ Show the function

```
decimal_to_binary(0.1, precision = 100)
```

```
[1] ".0001100110011001100110011001100110011001100110011001100110011001101"
```

Floating point numbers

```r
# Subtracting two nearly identical floating-point nu
x <- 0.3 - 0.2
y <- 0.1

# Check if they are equal
result <- x == y
```

```r
print(result)
```

```
[1] FALSE
```

Floating point numbers

```
# prints a more precise value of x
print(format(x, digits = 20))
```

```
[1] "0.099999999999999977796"
```

```
# prints a more precise value of y
print(format(y, digits = 20))
```

```
[1] "0.10000000000000000555"
```

...now...

```
tolerance <- 1e-9
equal <- abs(x - y) < tolerance
print(equal)
```

```
[1] TRUE
```

Decimal numbers in a computer

If computers only understand `0` and `1`, how can they express decimal numbers like *139*?

Decimal numbers in a computer

If computers only understand `0` and `1`, how can they express decimal numbers like *139*?

- *Standards* define how symbols, colors, etc are shown on the screen.
- Facilitates interaction with a computer (our keyboards do not only consist of a `0`/`1` switch).

What time is it?

The hexadecimal system

- Binary numbers can become quite long rather quickly.
- Computer Science: refer to binary numbers with the *hexadecimal* system.

The hexadecimal system

- *16 symbols*:
  - `0`-`9` (used like in the decimal system)…
  - and `A`-`F` (for the numbers 10 to 15).

The hexadecimal system

- *16 symbols*:
  - `0`-`9` (used like in the decimal system)…
  - and `A`-`F` (for the numbers 10 to 15).
- *16 symbols >>> base 16*: each digit represents an increasing power of 16 (16^0, 16^1, etc.).

The hexadecimal system

What is the decimal number 139 expressed in the hexadecimal system?

The hexadecimal system

What is the decimal number 139 expressed in the hexadecimal system?

- Solution:

$$(8 \times 16^1) + (11 \times 16^0) = 139.$$

- More precisely:

$$(8 \times 16^1) + (B \times 16^0) = 8B = 139.$$

- Hence: `10001011` (in binary) = `8B` (in hexadecimal) = `139` in decimal.

The hexadecimal system

Advantages (when working with binary numbers)

1. Shorter than raw binary representation
2. Much easier to translate back and forth between binary and hexadecimal than binary and decimal.

WHY?

😆

Character Encoding

Computers and text

How can a computer understand text if it only understands `0`s and `1`s?

A modified version of South Korean Dubeolsik (two-set type) for old hangul letters. (Illustration by Yes0song 2010, Creative Commons Attribution-Share Alike 3.0 Unported)

Computers and text

How can a computer understand text if it only understands `0`s and `1`s?

- *Standards* define how `0`s and `1`s correspond to specific letters/characters of different human languages.
- These standards are usually called *character encodings*.
- Coded character sets that map unique numbers (in the end in binary coded values) to each character in the set.

Computers and text

How can a computer understand text if it only understands `0`s and `1`s?

- *Standards* define how `0`s and `1`s correspond to specific letters/characters of different human languages.
- These standards are usually called *character encodings*.
- Coded character sets that map unique numbers (in the end in binary coded values) to each character in the set.
- For example, ASCII (American Standard Code for Information Interchange), now superseded by **utf-8** (Unicode).

ASCII logo. (public domain).

ASCII Table

Binary	Hexadecimal	Decimal	Character
0011 1111	3F	63	?
0100 0001	41	65	A
0110 0010	62	98	b

Character encodings: why should we care?

- In practice, Data Science means handling digital data of all formats and shapes.
  - Diverse sources.
  - Different standards.
  - Different languages (Japanese vs English).
  - *read*/*store* data.
- At the lowest level, this means understanding/handling encodings.

Computer Code and Text-Files

Putting the pieces together…

Two core themes of this course:

1. How can *data* be *stored* digitally and be *read* by/imported to a computer?
2. How can we give instructions to a computer by writing *computer code*?

Putting the pieces together…

Two core themes of this course:

1. How can *data* be *stored* digitally and be *read* by/imported to a computer?
2. How can we give instructions to a computer by writing *computer code*?

In both of these domains we mainly work with one simple type of document: *text files*.

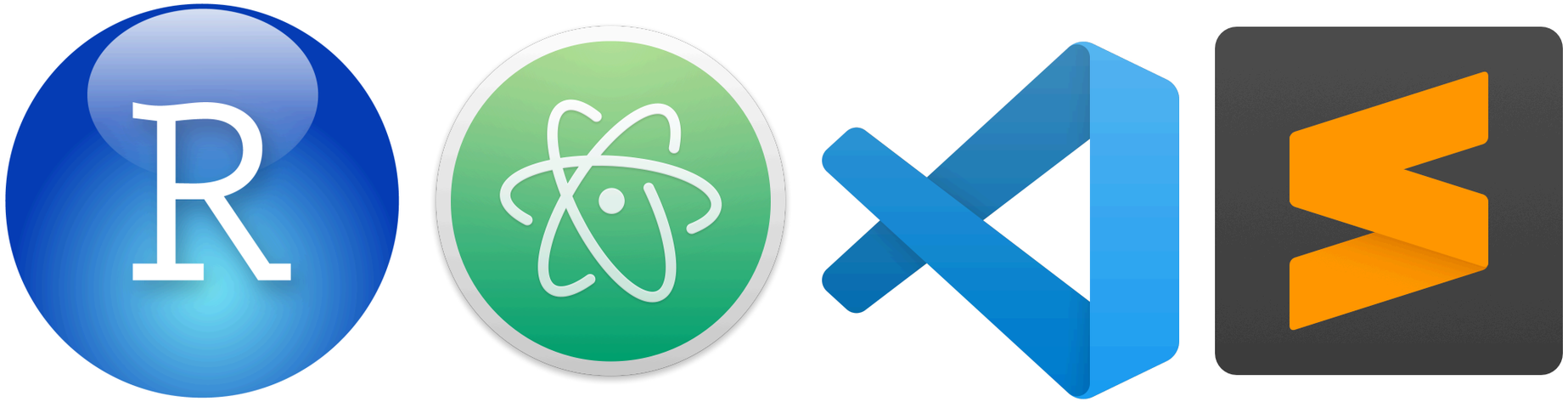
Text-files

- A *collection of characters* stored in a designated part of the computer memory/hard drive.
- An easy-to-read representation of the underlying information (`0`s and `1`s)!

Text-files

- A *collection of characters* stored in a designated part of the computer memory/hard drive.
- An easy to read representation of the underlying information (`0`s and `1`s)!
- Common device to store data:
  - Structured data (tables)
  - Semi-structured data (websites)
  - Unstructured data (plain text)
- Typical device to store computer code.

Text-editors: RStudio, Atom, VsCode



Install RStudio from here!

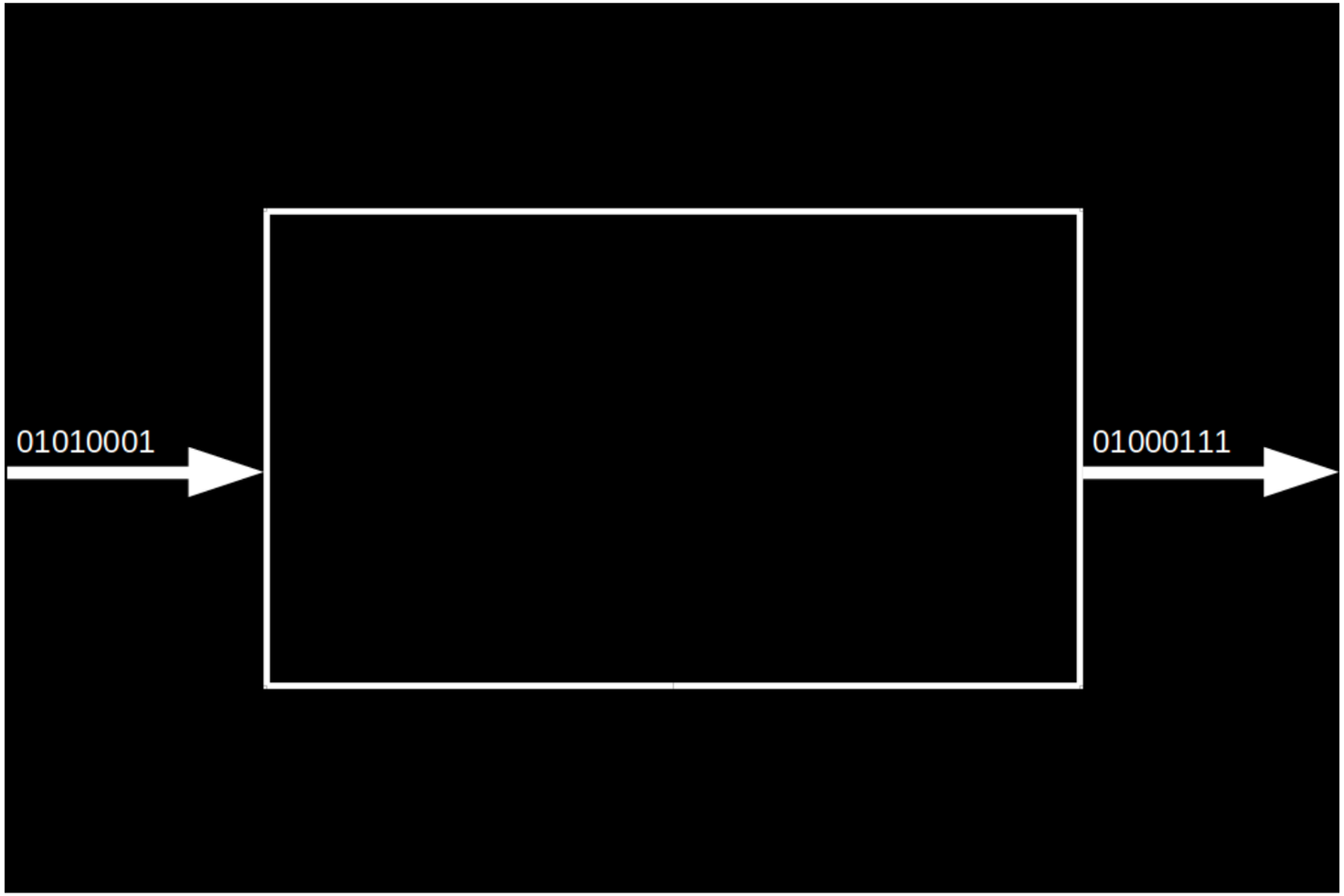
Install Atom from here!

Install VScode from here!

Install Sublime text from here!

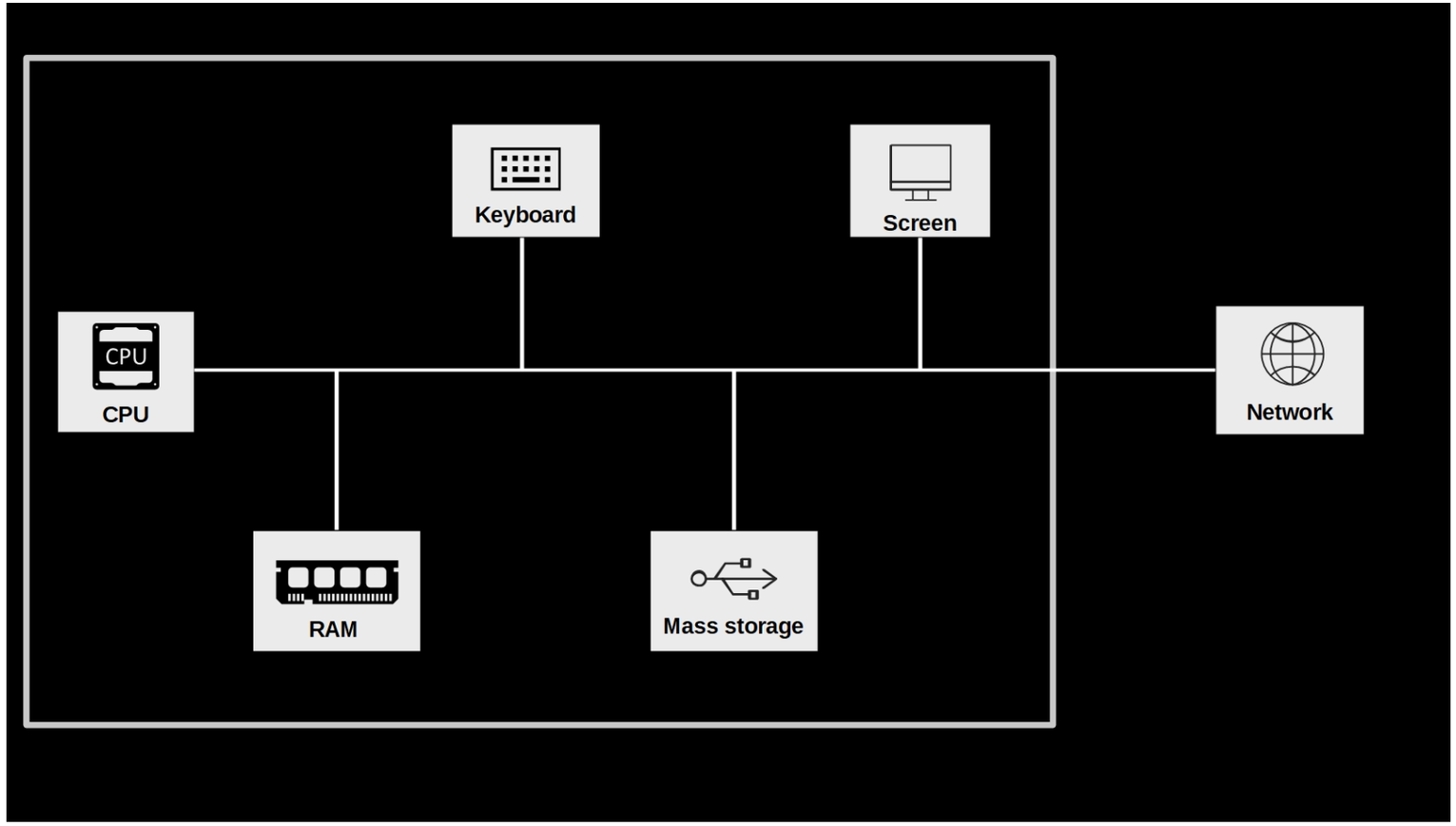
Data Processing Basics

The black box of data processing

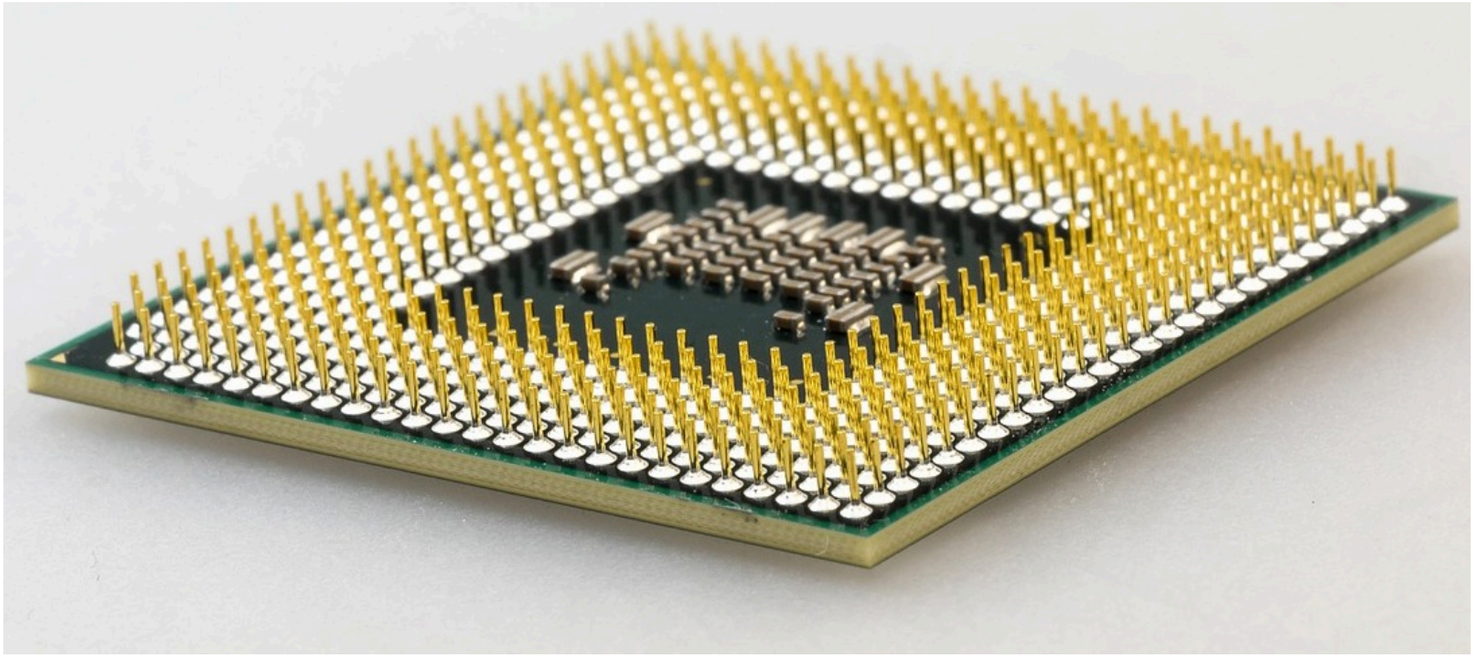


The ‘blackbox’ of data processing.

Components of a standard computing environment

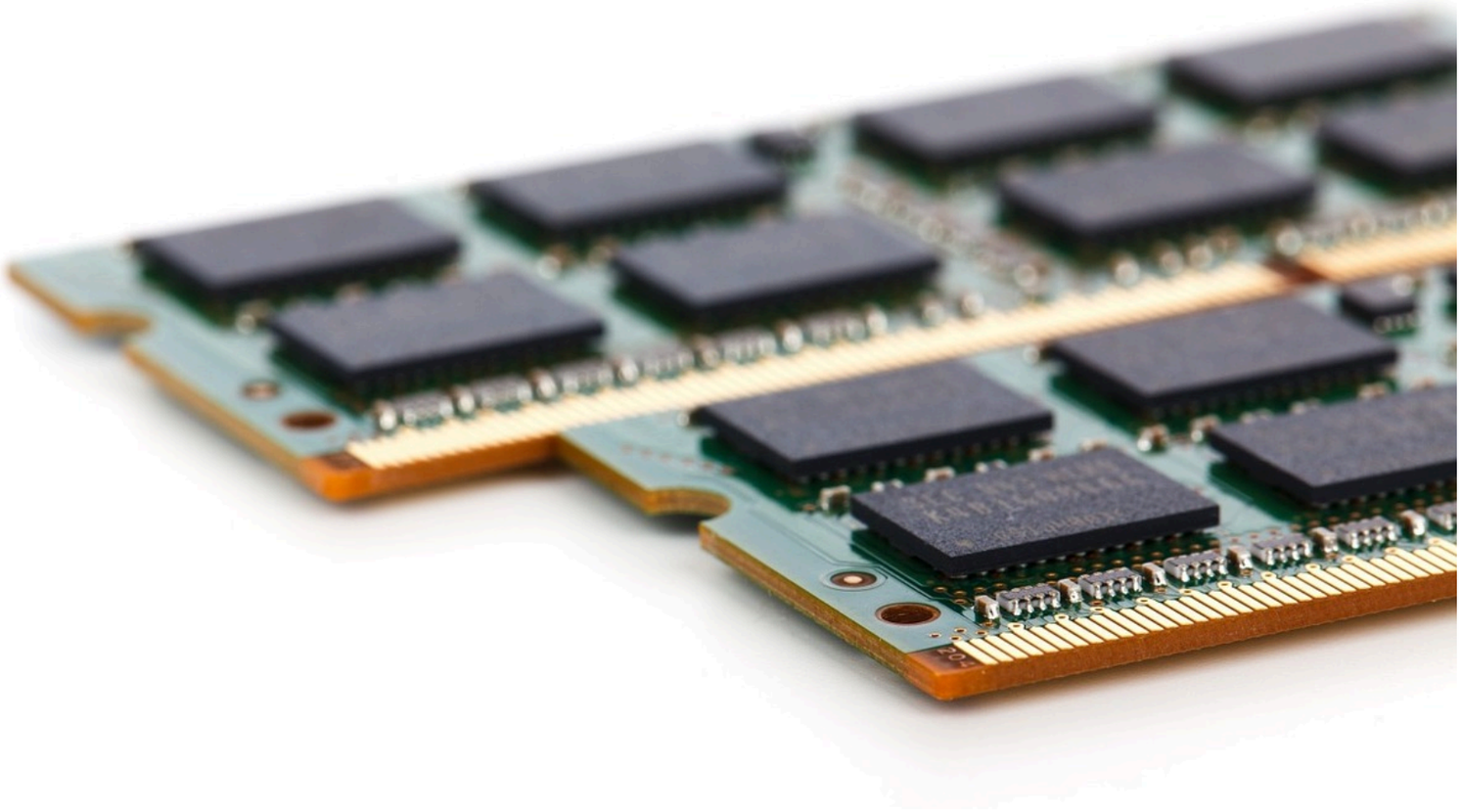


Central Processing Unit



- R runs on one *CPU core* by default.
- All modern CPUs have multiple cores.
- *Advanced*: explore parallelization with `plyr`, `doParallel()` and `future`

Random Access Memory



Random Access Memory

- Try to create a matrix with $10^8 \times 10^8$ elements:

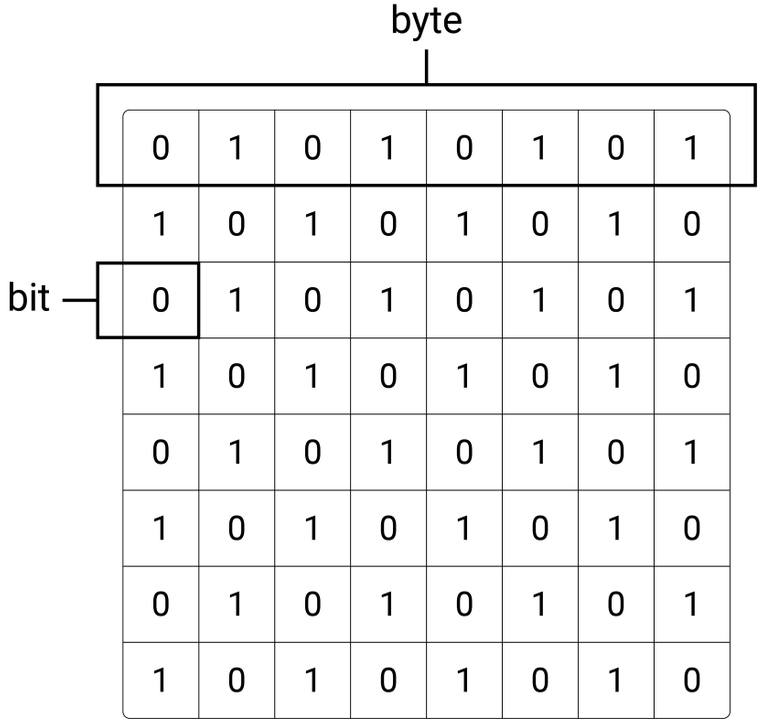
```
large_matrix <- matrix(1, nrow=1e8, ncol=1e8)
```

```
Error in matrix(1, nrow = 1e+08, ncol = 1e+08): vector is too large
```

- 💣💥💣💥💣💥
- Assuming each number is stored using 8 bytes, this matrix would require 8×10^16 bytes of RAM.

Units of information/data storage

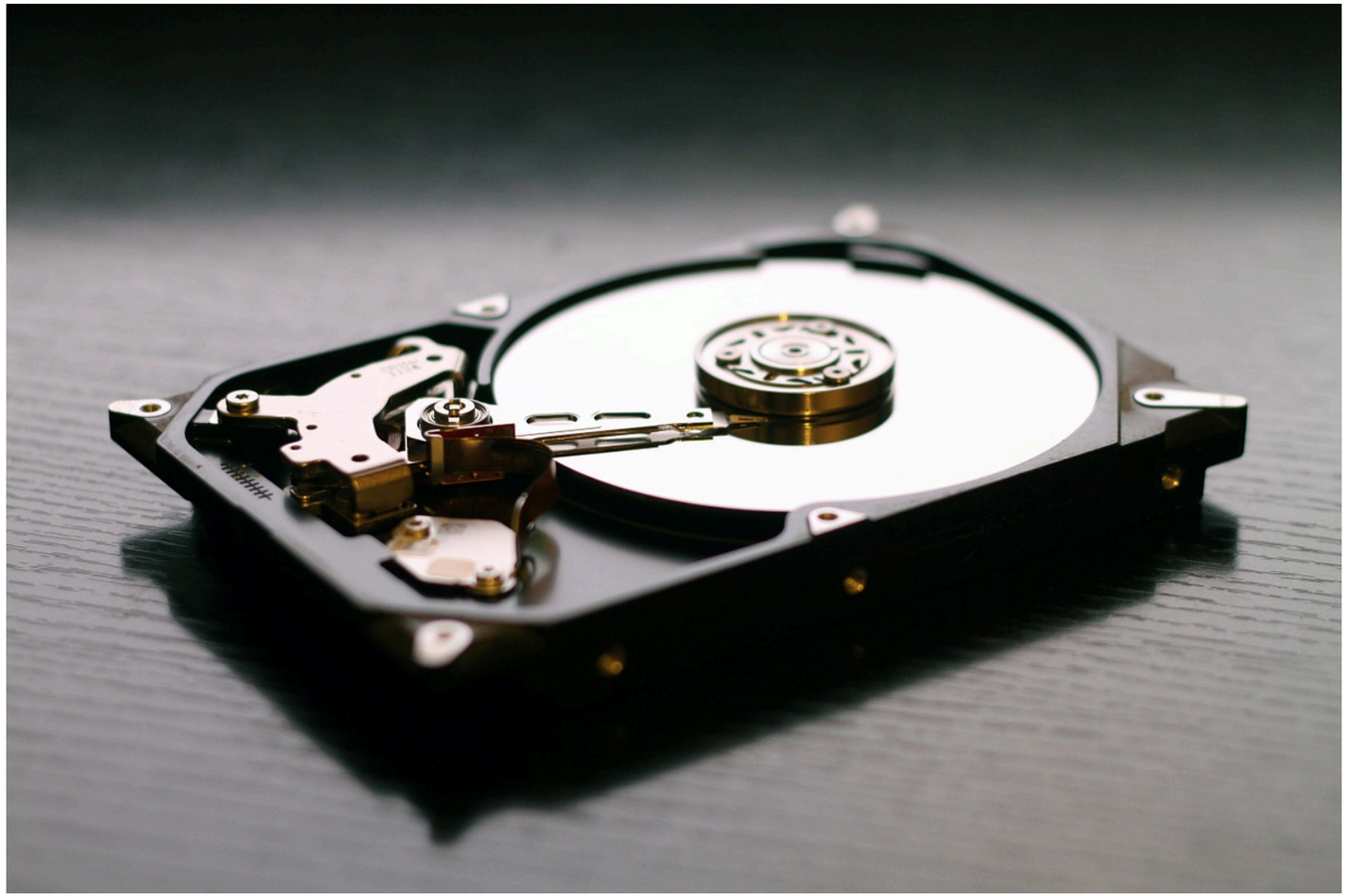
- *bit* (abbrev ‘b’): smallest unit of information in computing: `0` or `1`.
- To store this number, we require a capacity of 8 bits or one *byte* (1 byte = 8 bits; abbrev. ‘B’).
- Historically, one byte encoded a single character of text (ASCII). 4 bytes (32 bits) are called a *word*.



Bigger units for storage capacity

- 1 kilobyte (KB) = 1000^1 bytes
- 1 megabyte (MB) = 1000^2 bytes
- 1 gigabyte (GB) = 1000^3 bytes

Mass storage: hard drive



Network: Internet, cloud, etc.

Putting the pieces together…

Recall the initial example (survey) of this course.

1. Access a website (over the Internet), use keyboard to enter data into a website (a Google sheet in that case).
2. R program accesses the data of the Google sheet (again over the Internet), downloads the data, and loads it into RAM.
3. Data processing: produce output (in the form of statistics/plots), output on screen.

5468616E6B7320616E642073656520796F75206E6578742077656B21

Q&A