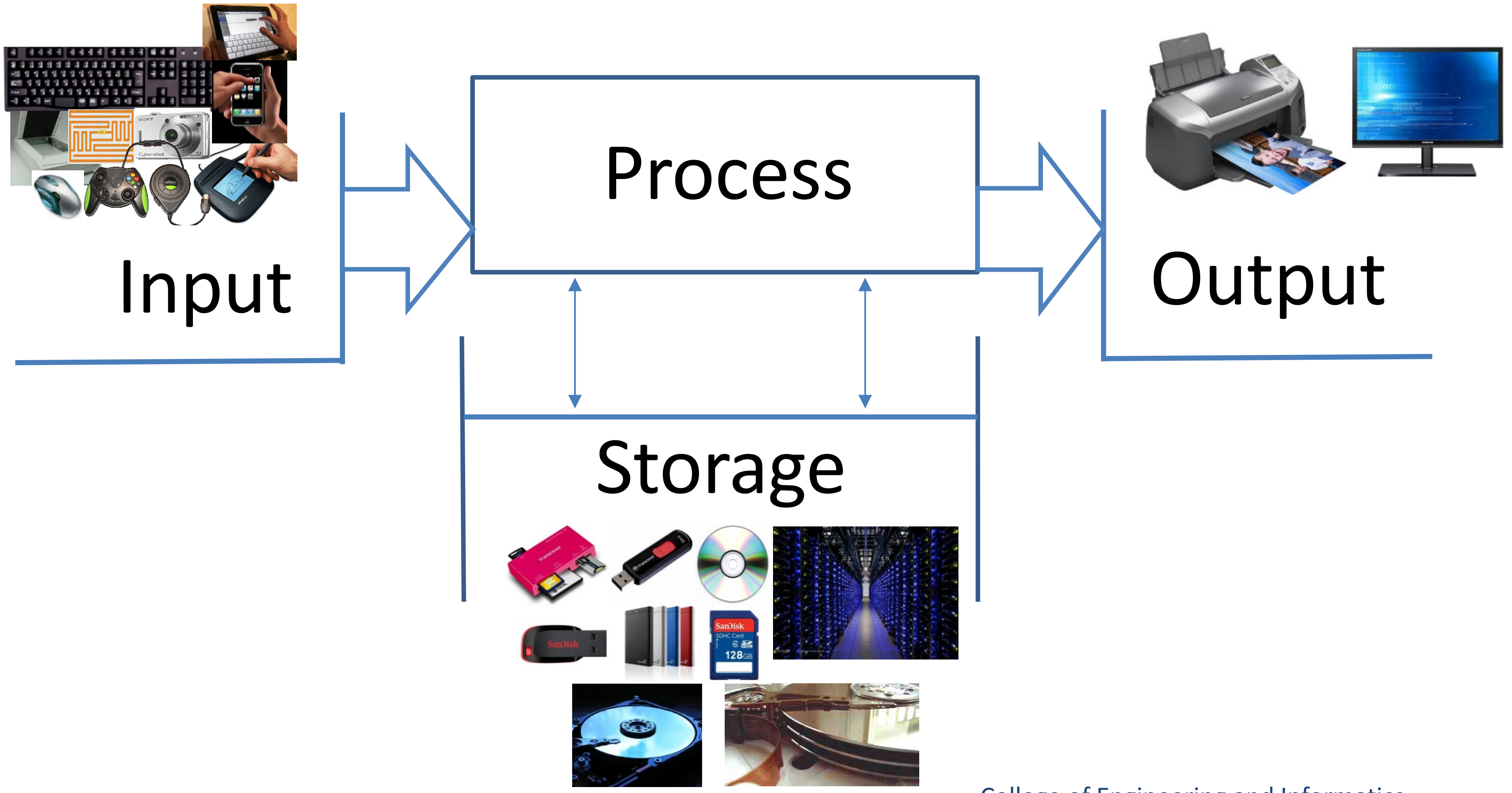
# CT1100: Computer Systems

# Computer Hardware: An Overview

Dr. Jim Duggan,
School of Computer Science
National University of Ireland Galway.

# Computer Hardware



Process

Input

Output

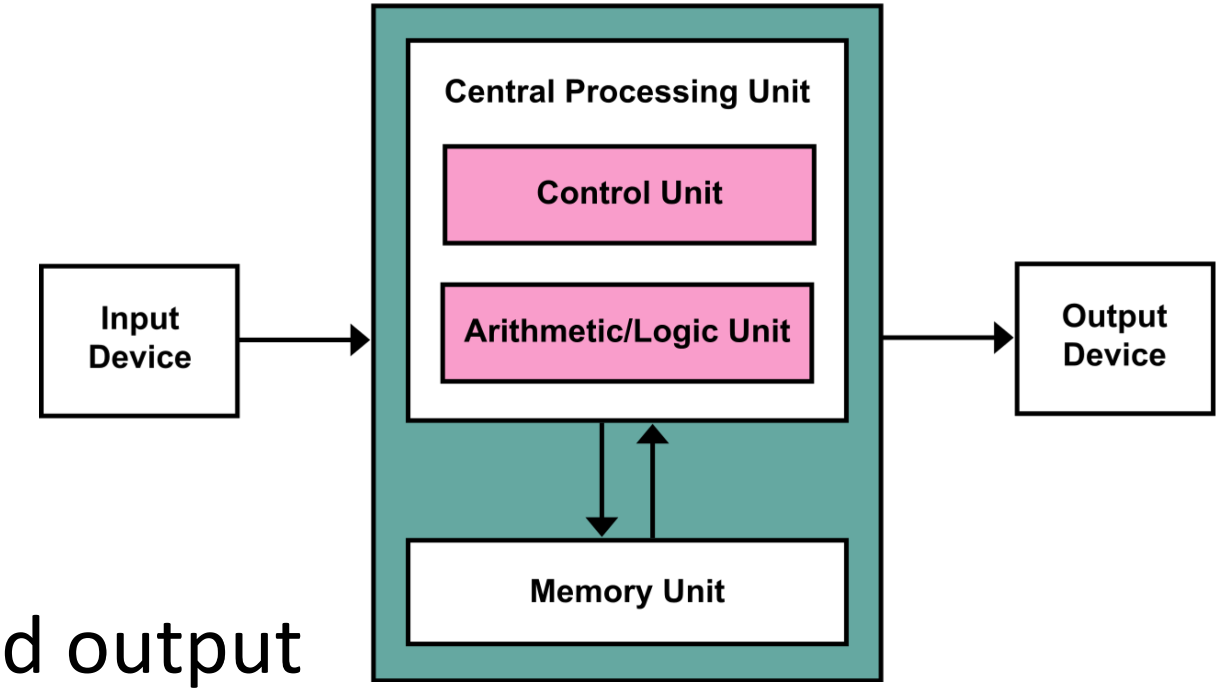
Storage

# Computer Architecture

- Computers process the data we input in order to achieve a required output
  - Input data is recognised by the computer as a series of binary numbers (electronic signals)
  - The chip circuitry manipulates these data following the instructions coded in programs
- The computer memory is used to store both data and the program
  - From there, the processor can easily extract the instructions and execute them
  - The program is stored as a series of bit patterns and can be easily altered

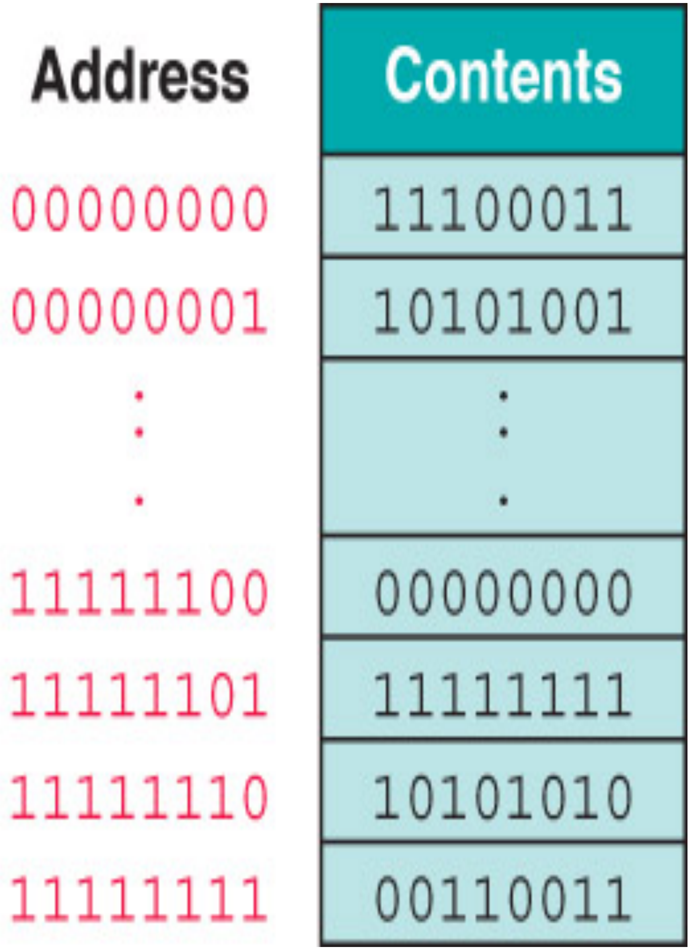
# The Processing Cycle

- The CPU
  - ALU
  - Control unit
  - Bus
- Computer memory
- Processing input and output
- Running a computer program
  - Fetch-decode-execute cycle



# Memory

- Memory is a collection of cells, each with a unique physical address; both addresses and contents are in binary
- Contents of a cell may be a value (e.g. number, text, image) or a program instruction



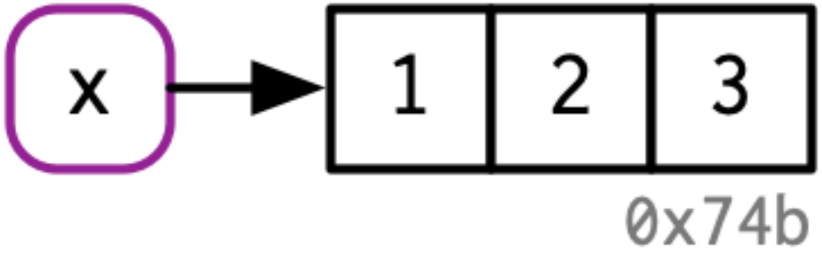
| Address | Contents |
|---|---|
| 00000000 | 11100011 |
| 00000001 | 10101001 |
| ⋮ | ⋮ |
| 11111100 | 00000000 |
| 11111101 | 11111111 |
| 11111110 | 10101010 |
| 11111111 | 00110011 |

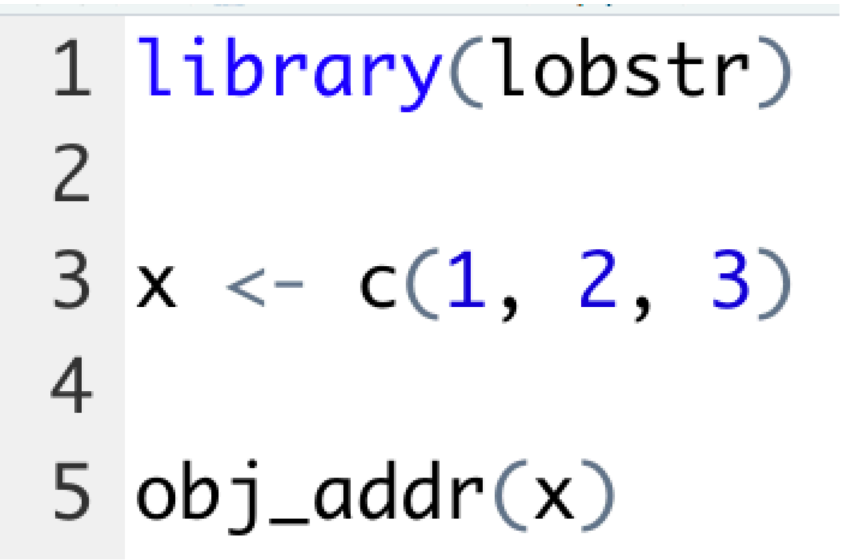
# Memory

- RAM: Random Access Memory (Main memory)
  - Primary storage area
  - On-line, fast, small, directly accessible, volatile, expensive
  - Accessed by address for read and write
- ROM: Read Only Memory
  - Written during manufacture: can only be read by user
- PROM and EPROM
- Extended memory
  - Use of secondary storage as virtual memory

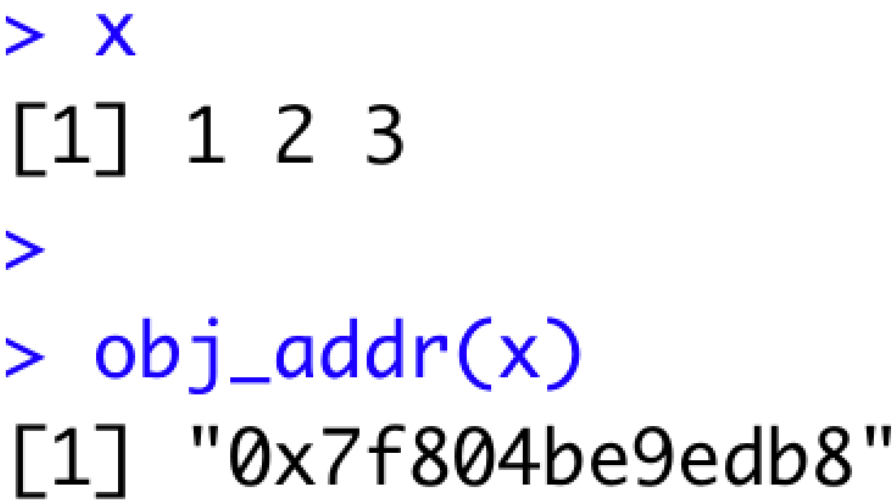
# In R

x <- c(1, 2, 3)



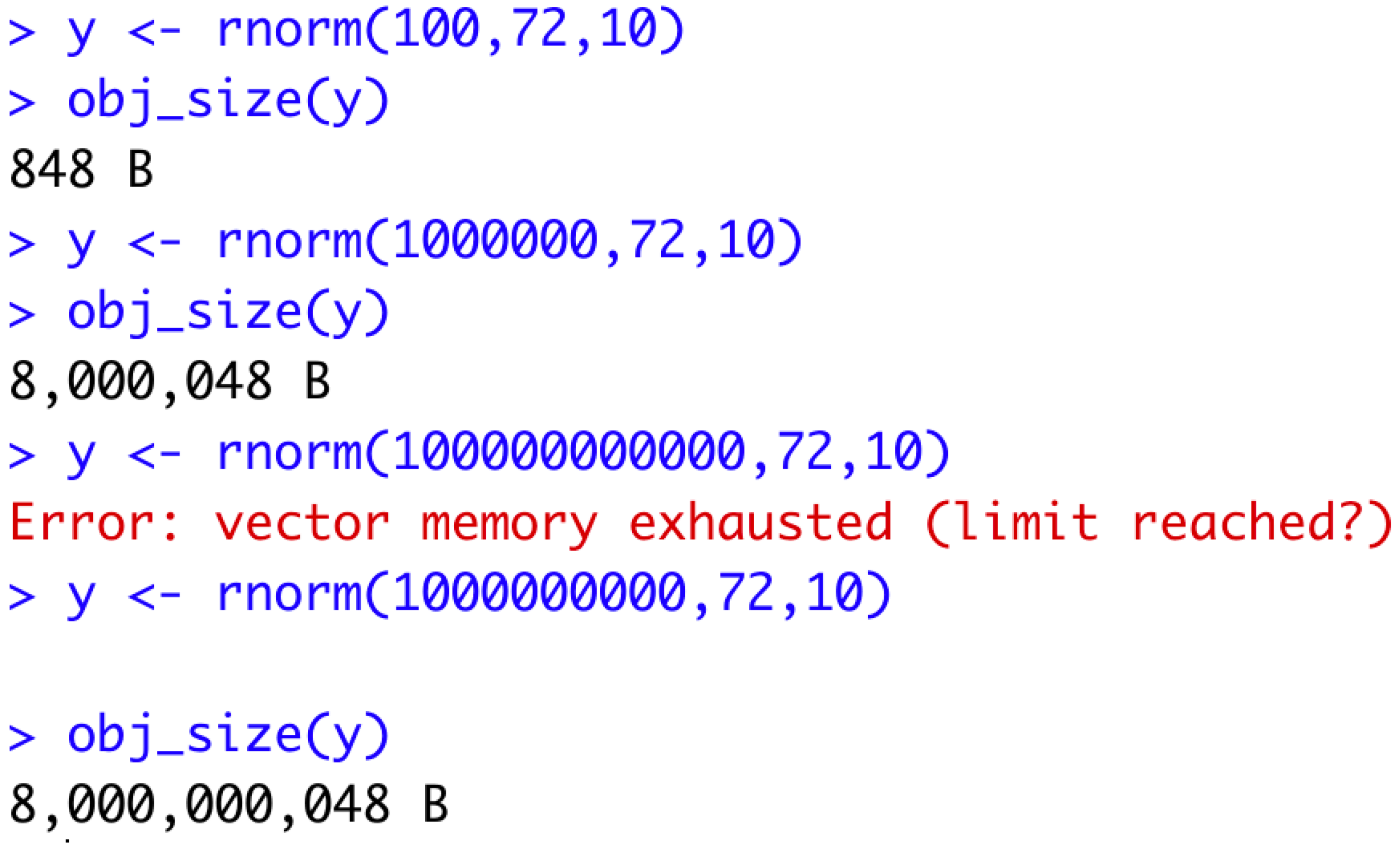


```
library(lobstr)

x <- c(1, 2, 3)

obj_addr(x)
```



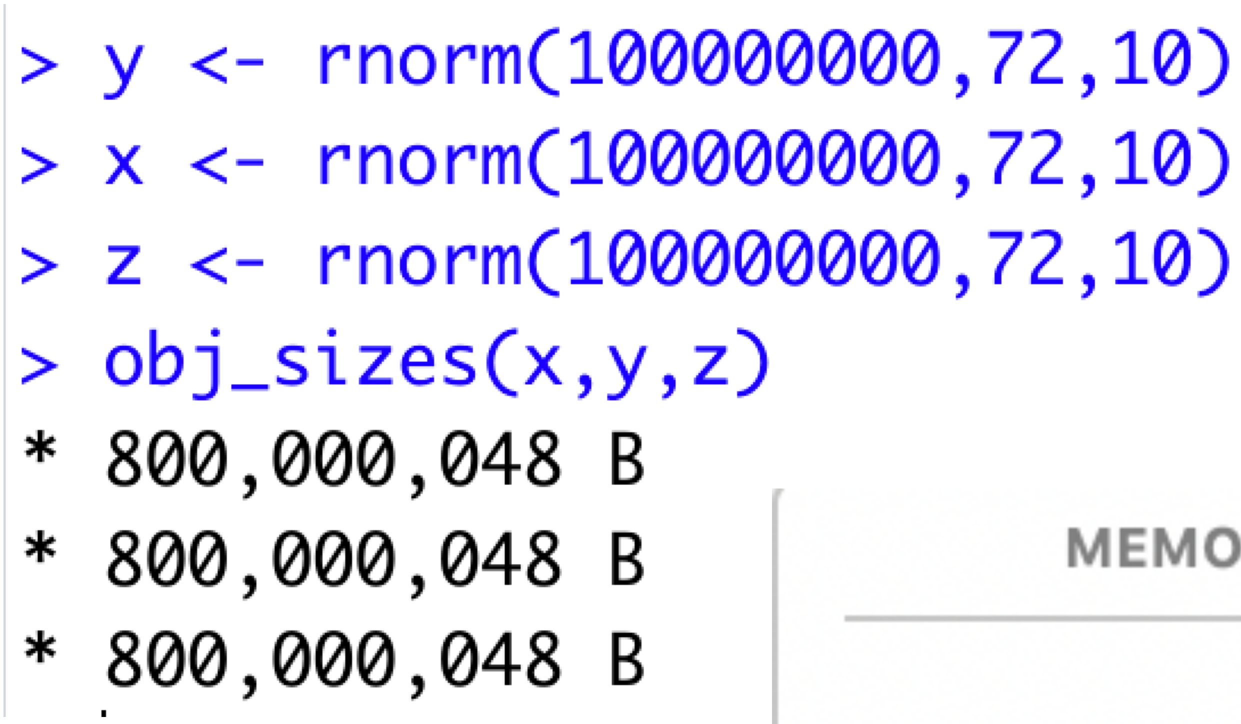
```
> x
[1] 1 2 3
>
> obj_addr(x)
[1] "0x7f804be9edb8"
```

# Size of objects in memory

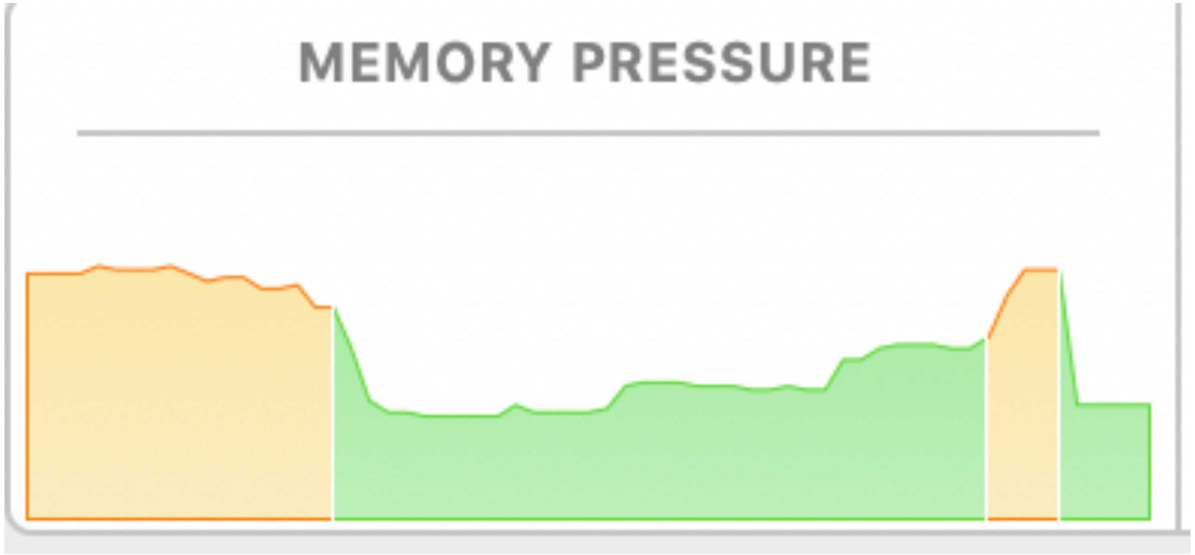


```
> y <- rnorm(100,72,10)
> obj_size(y)
848 B
> y <- rnorm(1000000,72,10)
> obj_size(y)
8,000,048 B
> y <- rnorm(100000000000,72,10)
Error: vector memory exhausted (limit reached?)
> y <- rnorm(1000000000,72,10)

> obj_size(y)
8,000,000,048 B
```

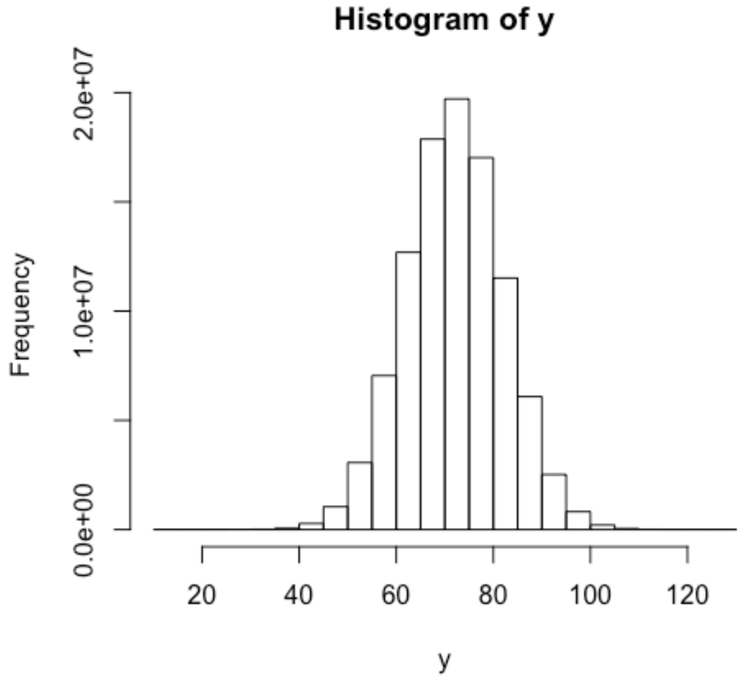
# The impact on System Memory

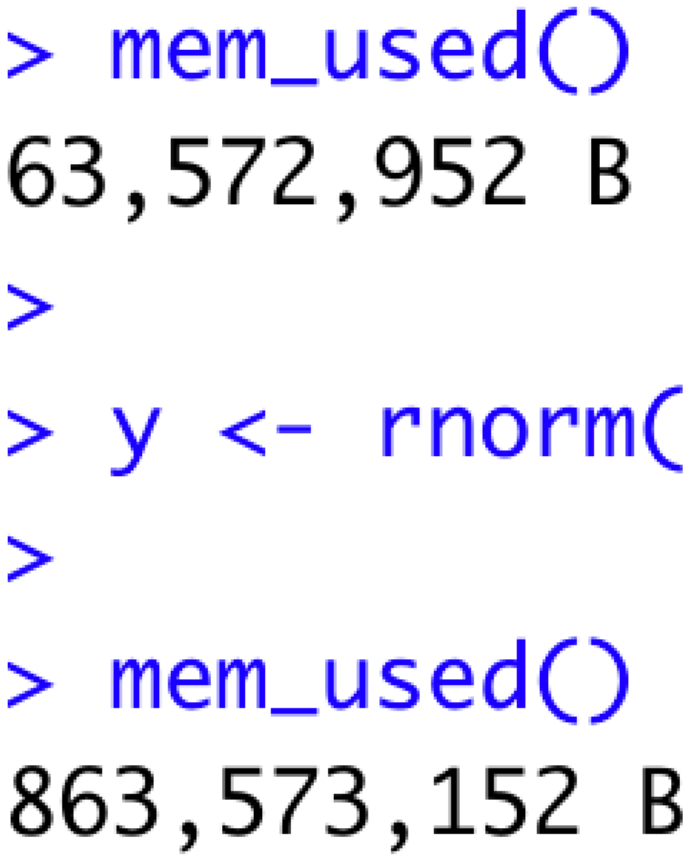


```
> y <- rnorm(100000000,72,10)
> x <- rnorm(100000000,72,10)
> z <- rnorm(100000000,72,10)
> obj_sizes(x,y,z)
* 800,000,048 B
* 800,000,048 B
* 800,000,048 B
```



# Memory Used

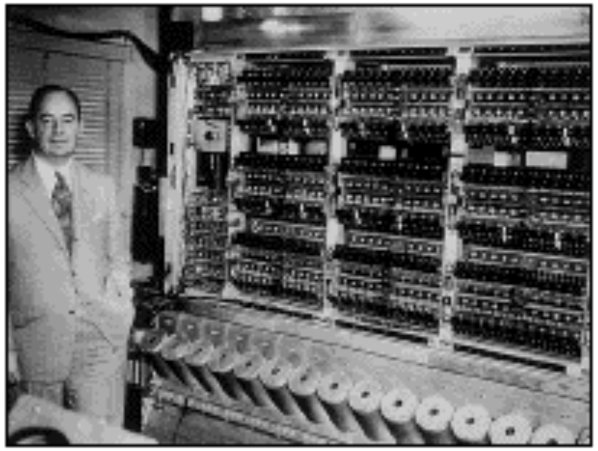




```
> mem_used()
63,572,952 B
>
> y <- rnorm(100000000,72,10)
>
> mem_used()
863,573,152 B
```

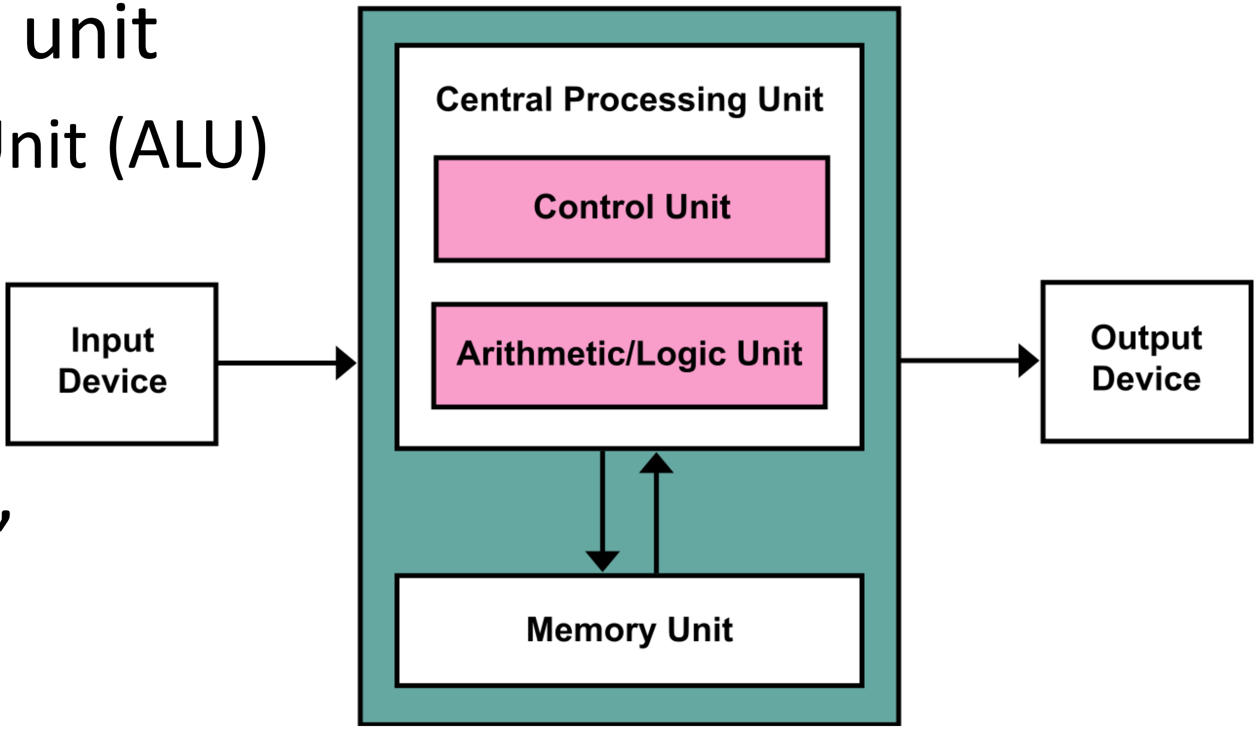
# John von Neumann 1903 - 1957

- 1946 Stored program concept (von Neumann architecture)
  - Program stored in same memory as data
  - Instructions comprise opcode and operand(s)
  - Instructions executed sequentially by Central Processing Unit (CPU)
- Worked on EDVAC, an early computer

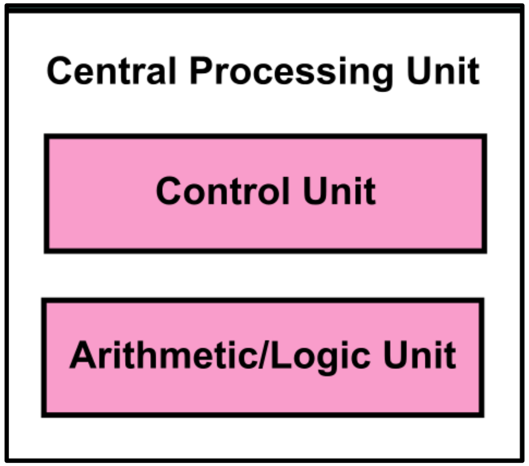


# von Neumann architecture

- Input and output devices
- Central processing unit
  - Arithmetic Logic Unit (ALU)
  - Control Unit
- Memory Unit
- Bus: data, address, control

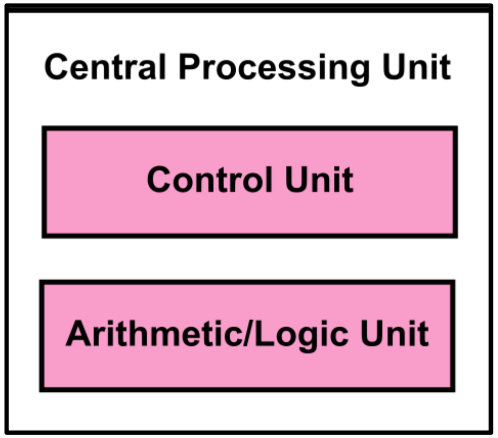


# Central Processing Unit (CPU)



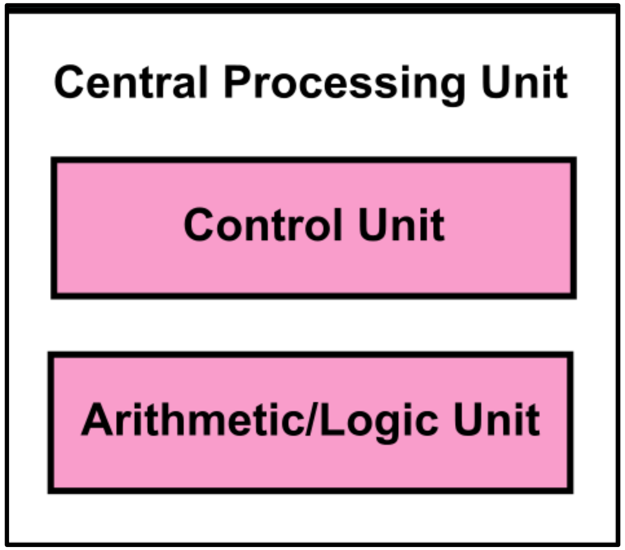
- Manages the *fetching* and *decoding* of programming instructions  in sequence
- Causes the processing by the ALU of the arithmetic and logic operations required by the instruction
- Sends out command signals to control the operation of the other elements of the system

# CPU: Arithmetic Logic Unit



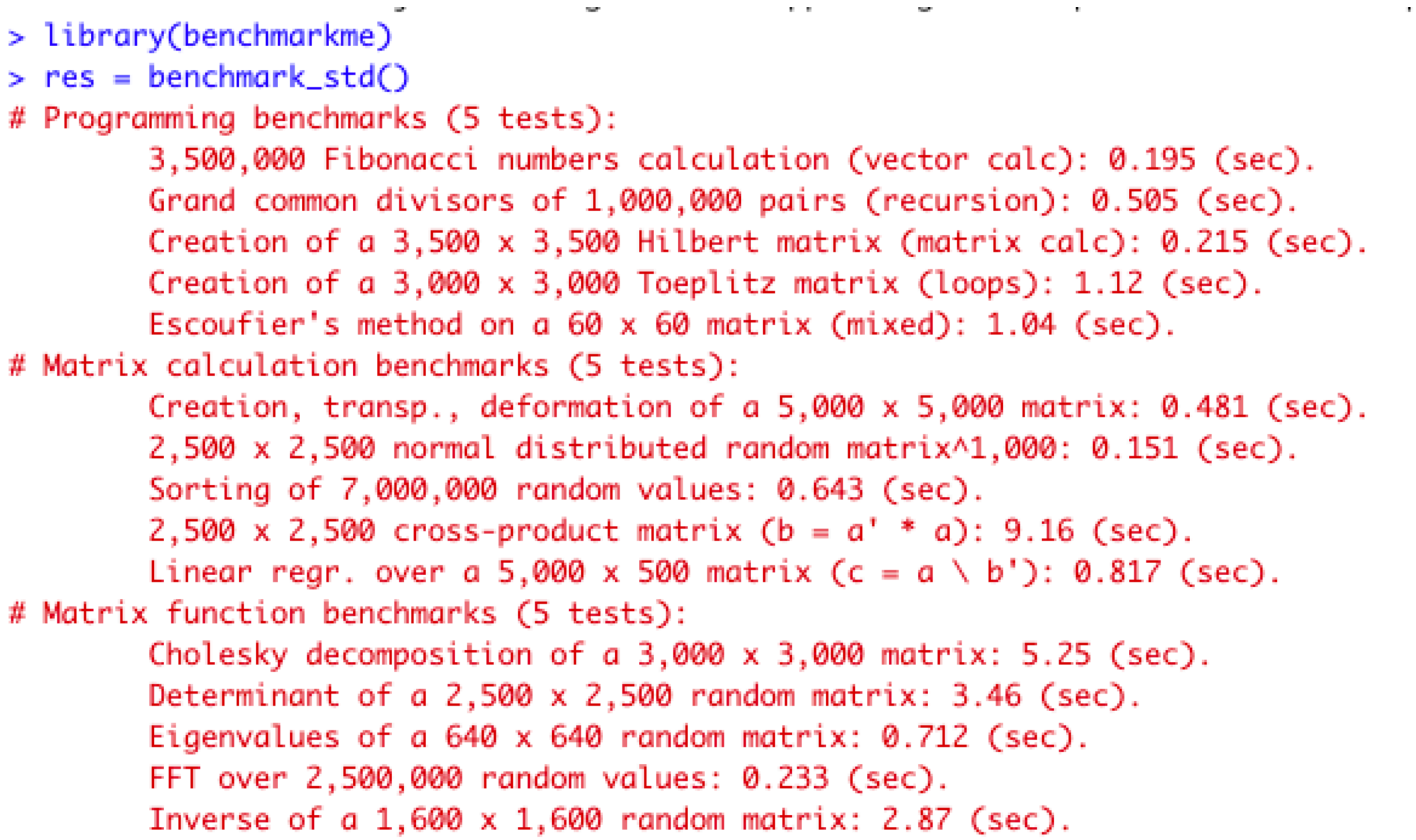
- Carries out arithmetic and performs logic operations
- Has available limited number of internal *registers*
  - General purpose registers for temporary storage while doing operations (*accumulators*)
  - Special purpose registers: program counter, instruction register, processor status etc.
- Is directed by the Control unit

# CPU: Control Unit



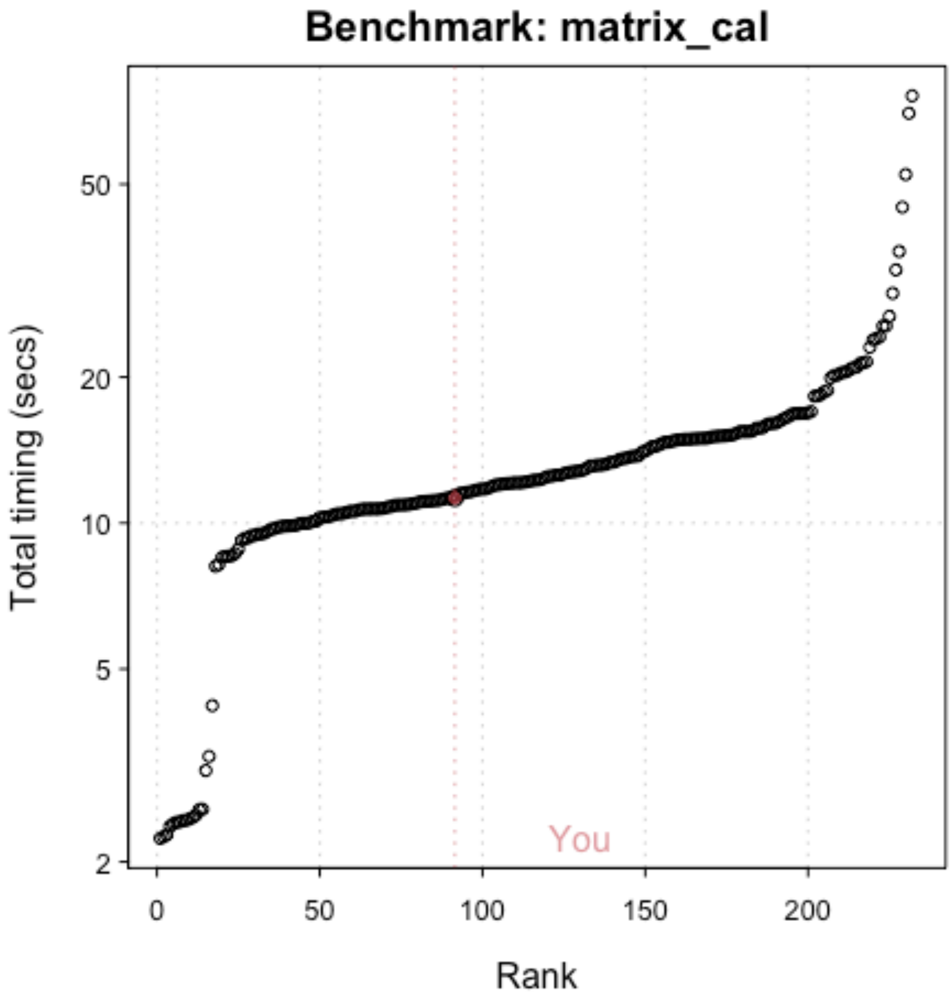
- Controls all activity in the microprocessor
  - Communicates with input and output devices to initiate transfer of instructions and data from memory
  - Fetches instructions from memory, interprets them, and decodes them to produce microinstructions
  - Dictates the correct sequence of operations
- Does not process or store data, just directs

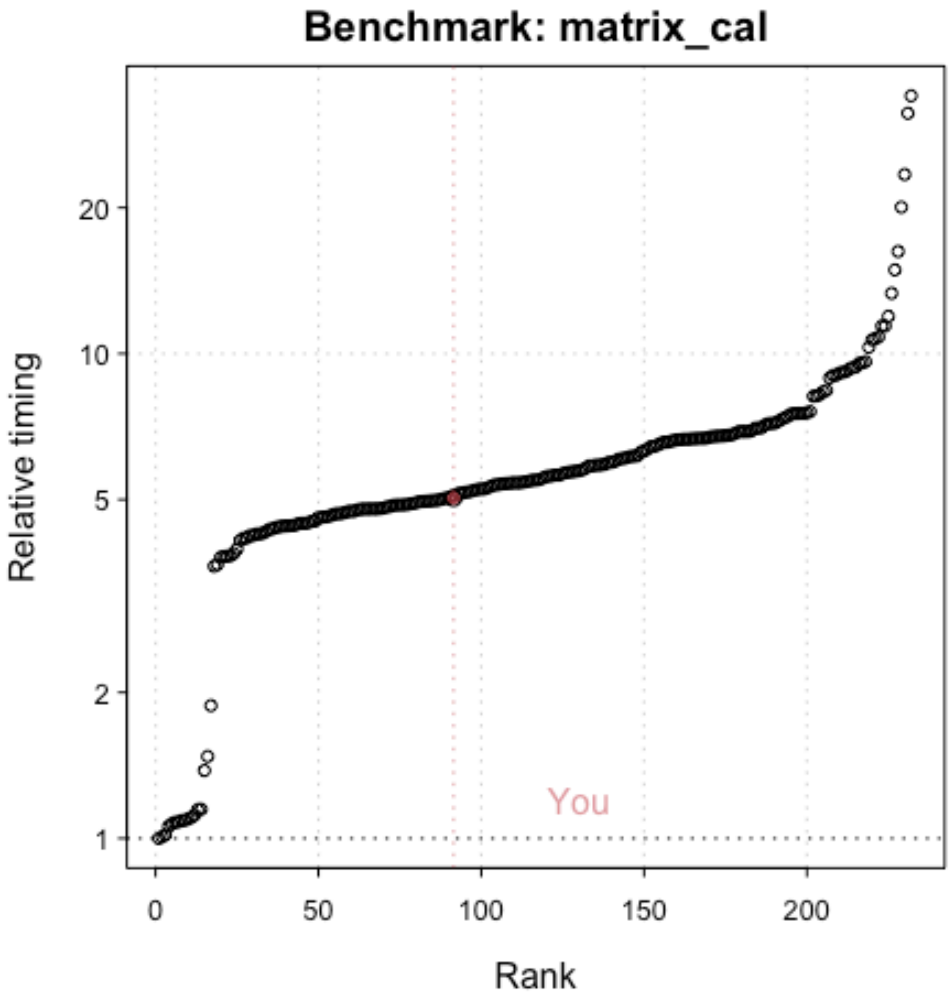
# Benchmarks - Laptop



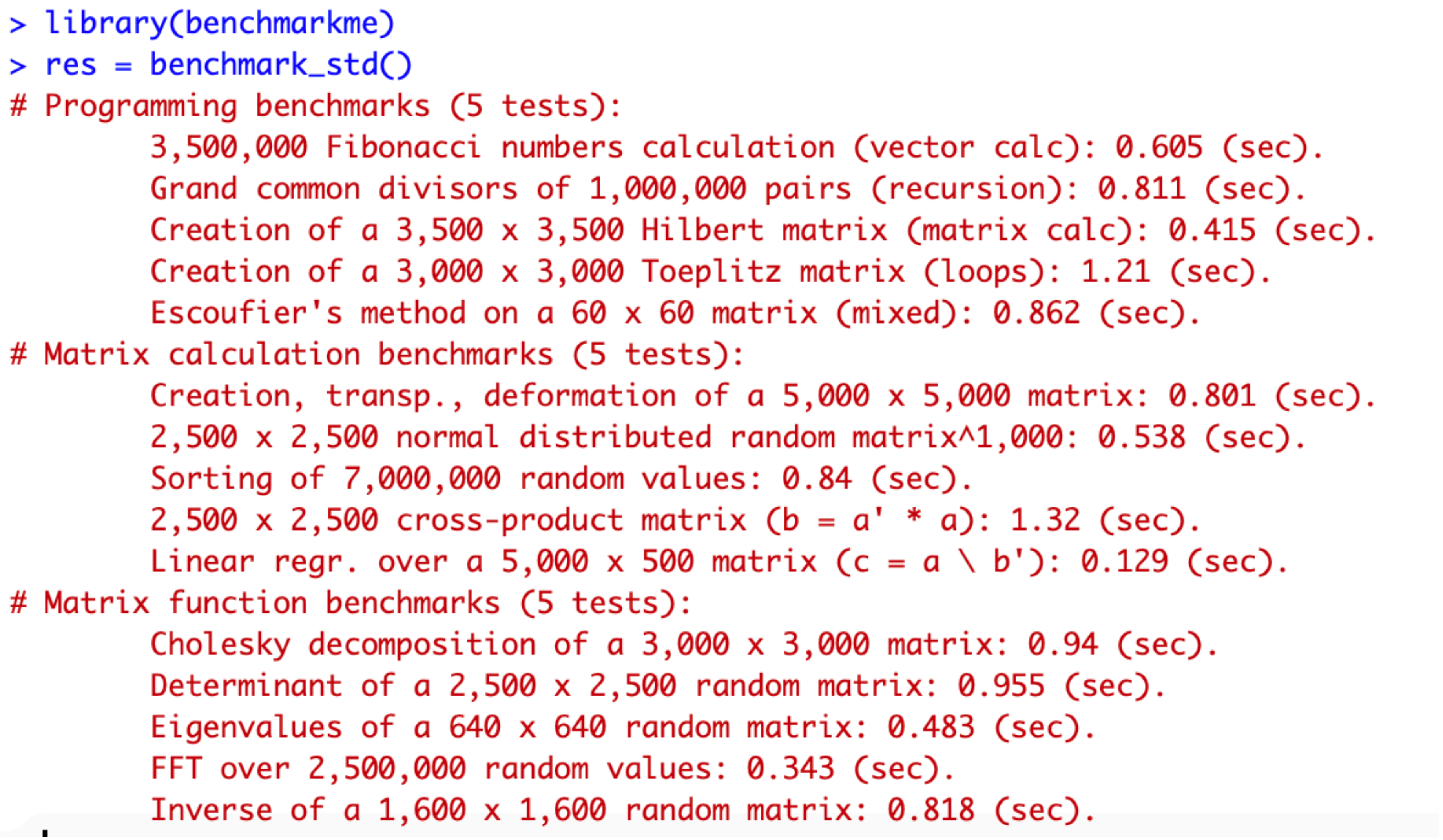
```
> library(benchmarkme)
> res = benchmark_std()
# Programming benchmarks (5 tests):
        3,500,000 Fibonacci numbers calculation (vector calc): 0.195 (sec).
        Grand common divisors of 1,000,000 pairs (recursion): 0.505 (sec).
        Creation of a 3,500 x 3,500 Hilbert matrix (matrix calc): 0.215 (sec).
        Creation of a 3,000 x 3,000 Toeplitz matrix (loops): 1.12 (sec).
        Escoufier's method on a 60 x 60 matrix (mixed): 1.04 (sec).
# Matrix calculation benchmarks (5 tests):
        Creation, transp., deformation of a 5,000 x 5,000 matrix: 0.481 (sec).
        2,500 x 2,500 normal distributed random matrix^1,000: 0.151 (sec).
        Sorting of 7,000,000 random values: 0.643 (sec).
        2,500 x 2,500 cross-product matrix (b = a' * a): 9.16 (sec).
        Linear regr. over a 5,000 x 500 matrix (c = a \ b'): 0.817 (sec).
# Matrix function benchmarks (5 tests):
        Cholesky decomposition of a 3,000 x 3,000 matrix: 5.25 (sec).
        Determinant of a 2,500 x 2,500 random matrix: 3.46 (sec).
        Eigenvalues of a 640 x 640 random matrix: 0.712 (sec).
        FFT over 2,500,000 random values: 0.233 (sec).
        Inverse of a 1,600 x 1,600 random matrix: 2.87 (sec).
```

# Comparisons - Laptop



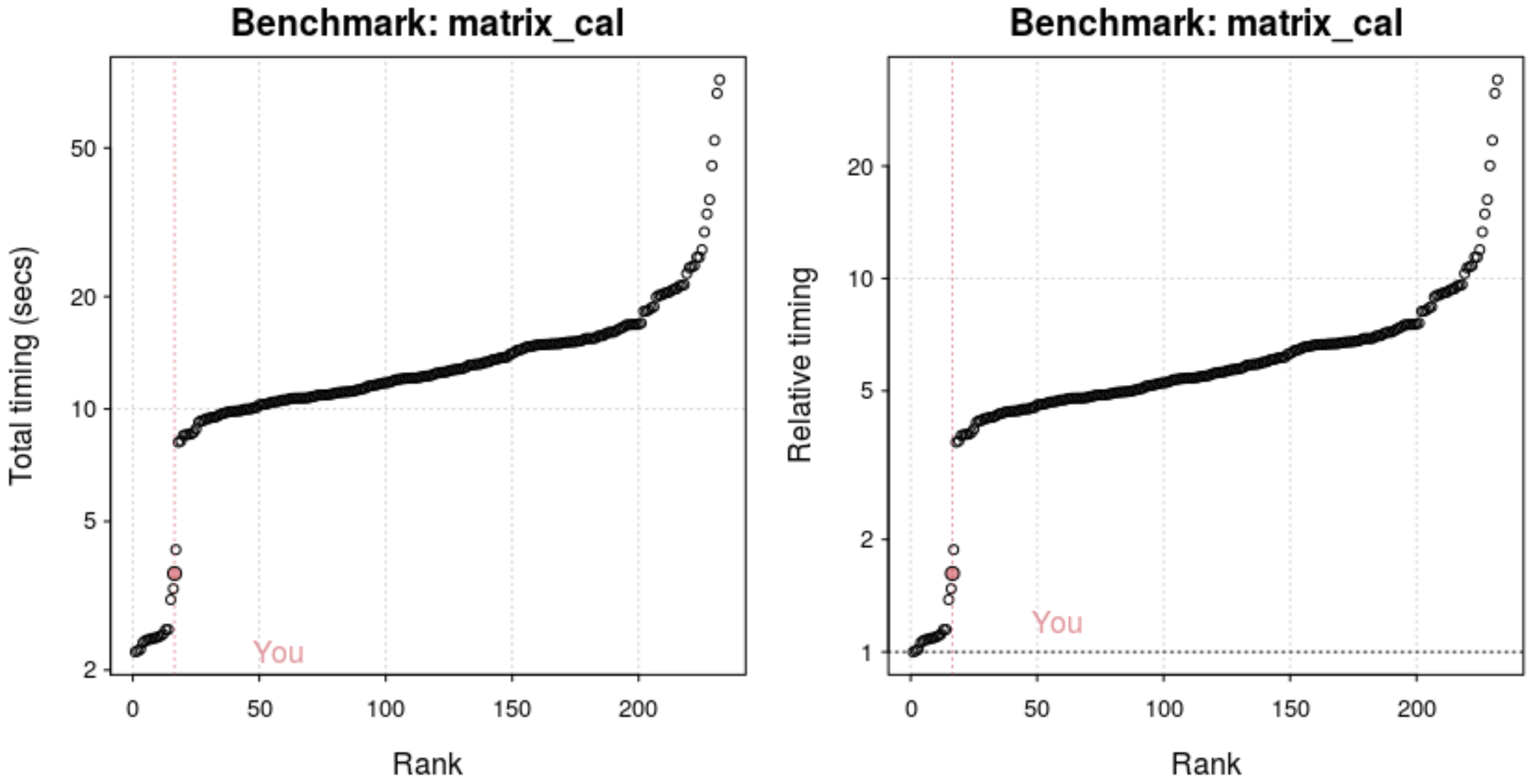


# Benchmarks – Rstudio Cloud



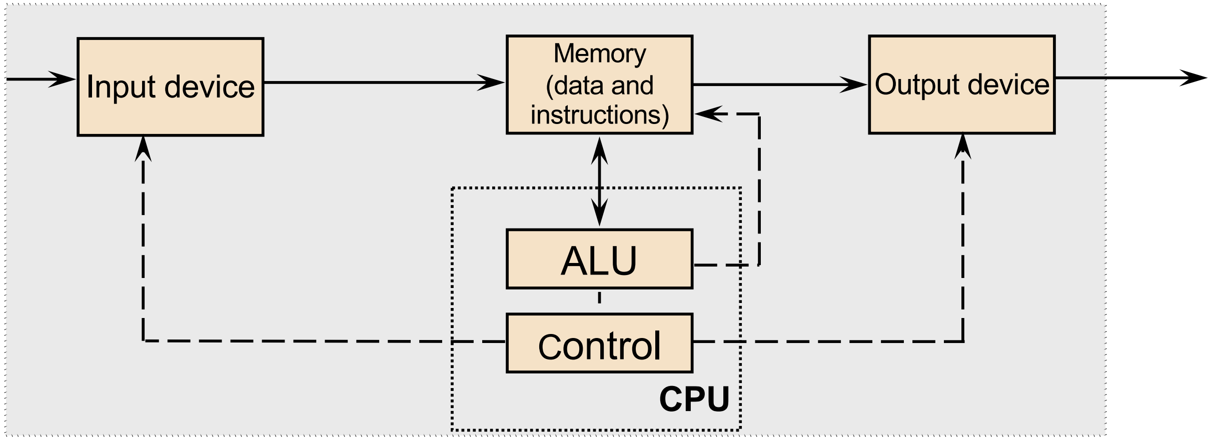
```
> library(benchmarkme)
> res = benchmark_std()
# Programming benchmarks (5 tests):
        3,500,000 Fibonacci numbers calculation (vector calc): 0.605 (sec).
        Grand common divisors of 1,000,000 pairs (recursion): 0.811 (sec).
        Creation of a 3,500 x 3,500 Hilbert matrix (matrix calc): 0.415 (sec).
        Creation of a 3,000 x 3,000 Toeplitz matrix (loops): 1.21 (sec).
        Escoufier's method on a 60 x 60 matrix (mixed): 0.862 (sec).
# Matrix calculation benchmarks (5 tests):
        Creation, transp., deformation of a 5,000 x 5,000 matrix: 0.801 (sec).
        2,500 x 2,500 normal distributed random matrix^1,000: 0.538 (sec).
        Sorting of 7,000,000 random values: 0.84 (sec).
        2,500 x 2,500 cross-product matrix (b = a' * a): 1.32 (sec).
        Linear regr. over a 5,000 x 500 matrix (c = a \ b'): 0.129 (sec).
# Matrix function benchmarks (5 tests):
        Cholesky decomposition of a 3,000 x 3,000 matrix: 0.94 (sec).
        Determinant of a 2,500 x 2,500 random matrix: 0.955 (sec).
        Eigenvalues of a 640 x 640 random matrix: 0.483 (sec).
        FFT over 2,500,000 random values: 0.343 (sec).
        Inverse of a 1,600 x 1,600 random matrix: 0.818 (sec).
```

# Comparisons - RStudio



# Bus

- The bus is a common route way along which the bits travel inside the machine
- Can have 8,16,32 or 64 bit bus (the number of bits travelling in parallel)
- Data bits travelling on the bus are available to any component attached to bus
- Three separate bus pathways: data, address and control

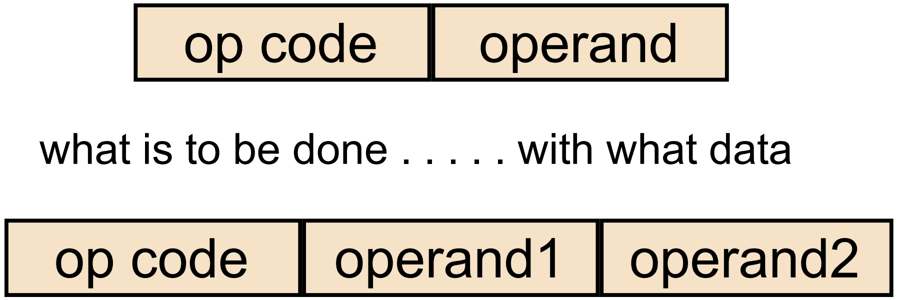


# Computer Bus

- Data Bus
  - Bus for moving data between different units
  - Bi-directional: direction governed by additional signal lines
- Address Bus
  - An address identifies memory locations
  - This is a unidirectional  bus carrying addresses between microprocessor to memory and I-O
- Control Bus
  - Used to send control signals which control access to and use of data and address buses

# Running a Computer Program

- Program instructions are stored in memory in the form:



| op code | operand |
|---|---|

what is to be done . . . . . with what data

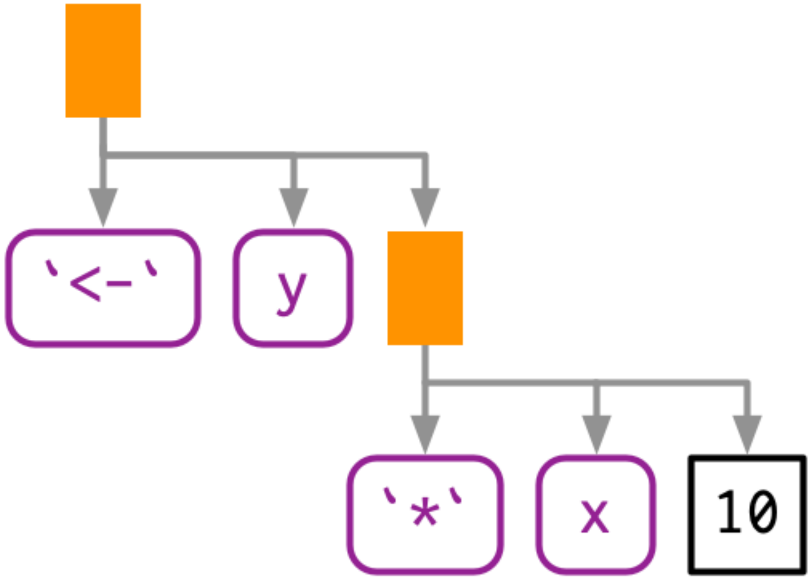
| op code | operand1 | operand2 |
|---|---|---|

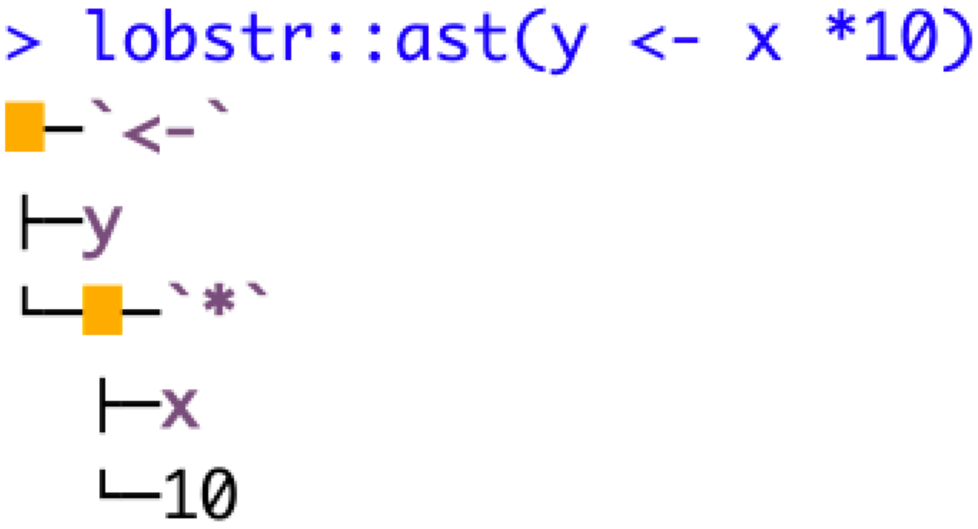
- Processing comprises:
  - Fetch the next instruction from memory
  - Determine the type of instruction just fetched (decode)
  - If the instruction uses data in memory, determine what they are and fetch them
  - Execute the instruction
  - Store the results in the proper place (in memory)

# Fetch-Decode-Execute

- Fetch: the address of the next instruction in the program is held in the *program counter register*
  - This address is sent on the address bus to memory with read signal
  - The value is put on the data bus and then stored in the *instruction register*
  - The program counter register is updated to point to the next instruction
- Decode: the instruction to be executed is decoded into low-level machine microinstructions
- Execute: the decoded instructions are carried out in the ALU

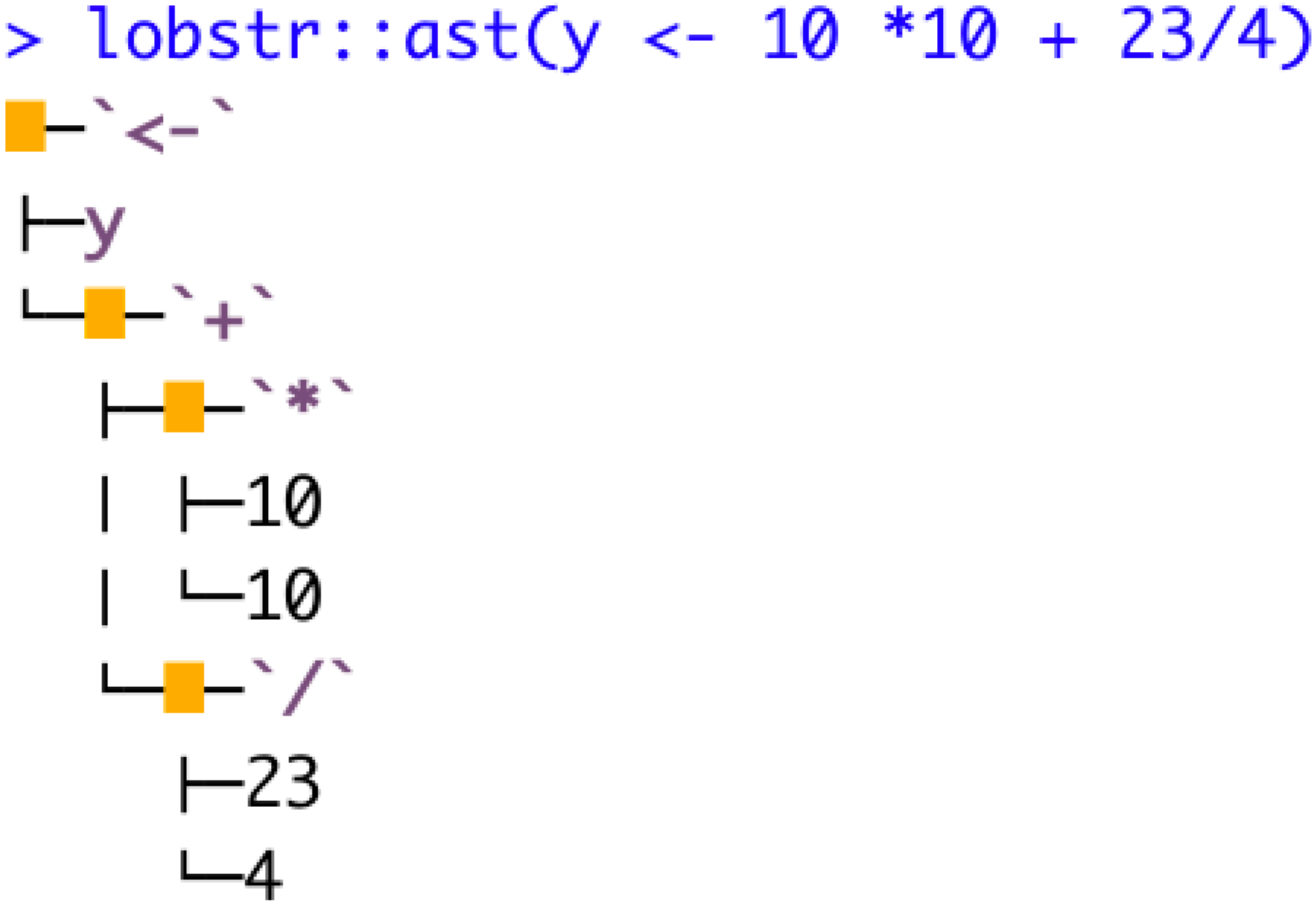
# Computation Idea





```
> lobstr::ast(y <- x *10)
■─`<-`
├─y
└─■─`*`
  ├─x
  └─10
```

# More complicated equation

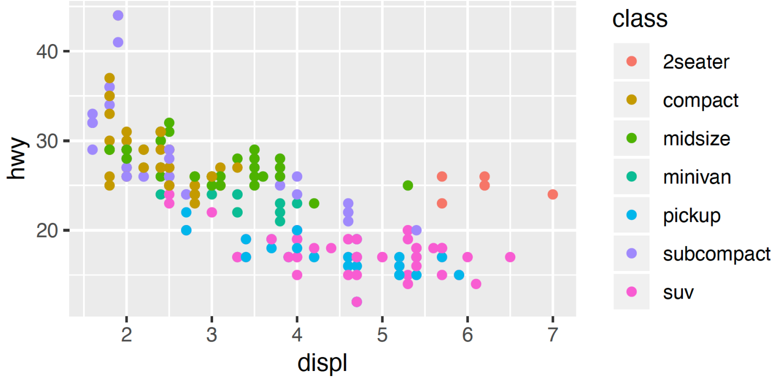


```
> lobstr::ast(y <- 10 *10 + 23/4)
█─`<-`
├─y
└─█─`+`
  ├─█─`*`
  │ ├─10
  │ └─10
  └─█─`/`
    ├─23
    └─4
```

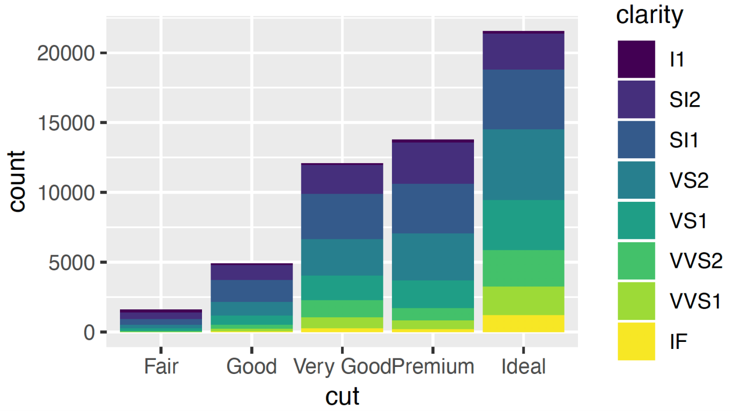
# Course Summary

| Lecture(s) | Topic |
|---|---|
| 1 | Course Introduction |
| 2 | The Processing Cycle and Binary Data |
| 3 | Data in R with Atomic Vectors |
| 4 | The CRAN Library and Calling Functions in R |
| 5 | Tidy Data and Data Frames |
| 6-7 | **ggplot2** - A Grammar of Graphics |
| 8-10 | **dplyr** - A Grammar of Data Manipulation |
| 11-12 | Introduction to Hardware |

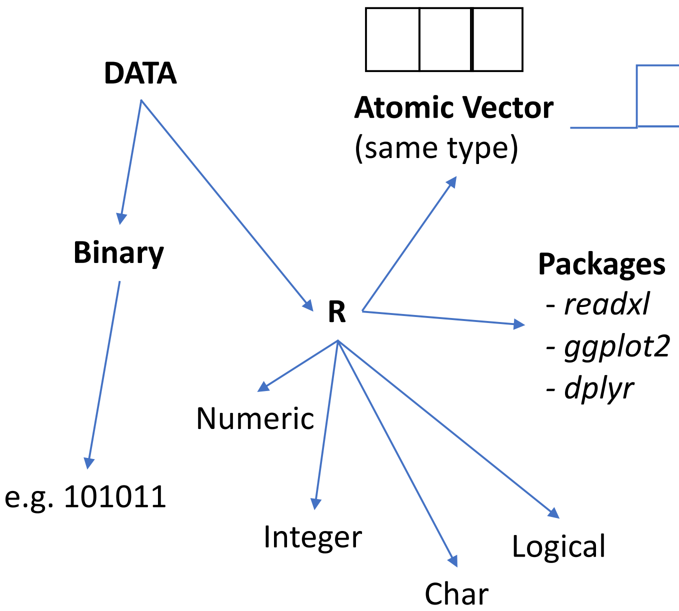
```
ggplot(data=d)+
  geom_point(aes(x=displ,y=hwy,colour=class))
```

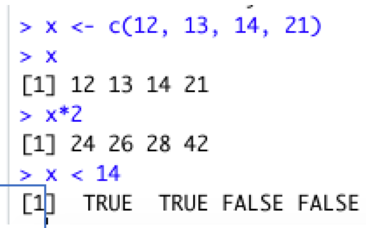


```
ggplot(data=diamonds)+
  geom_bar(aes(x=cut,fill=clarity))
```

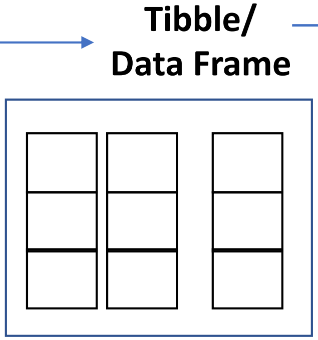


# R Summary





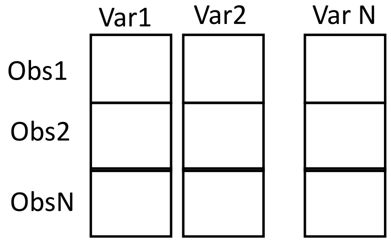
```
> x <- c(12, 13, 14, 21)
> x
[1] 12 13 14 21
> x*2
[1] 24 26 28 42
> x < 14
[1]  TRUE  TRUE FALSE FALSE
```



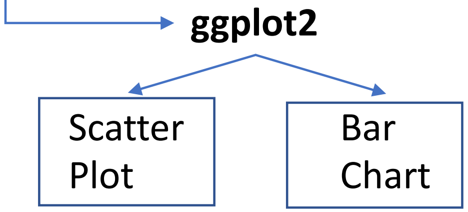
**Tidy Data**

*Row – Observation*

*Column – Variable*



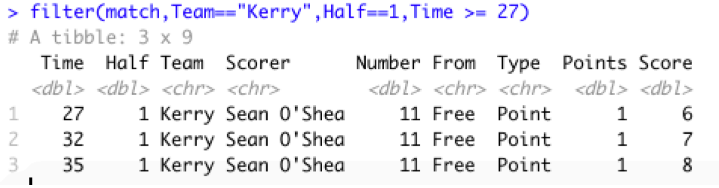
*Example → ggplot2::mpg*



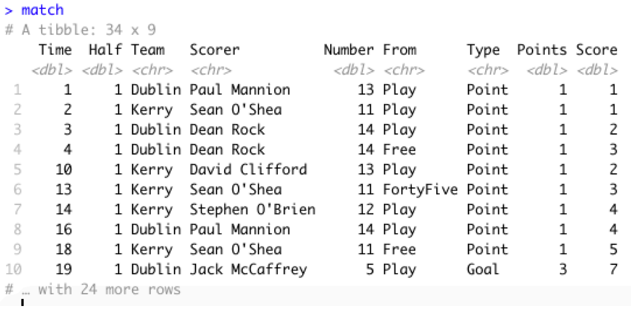
**dplyr**

"5 Verbs"

- **filter(),** arrange()
- select(), mutate()
- summarise()



```
> filter(match,Team=="Kerry",Half==1,Time >= 27)
# A tibble: 3 x 9
   Time  Half Team  Scorer       Number From  Type  Points Score
  <dbl> <dbl> <chr> <chr>         <dbl> <chr> <chr>  <dbl> <dbl>
1    27     1 Kerry Sean O'Shea      11 Free  Point      1     6
2    32     1 Kerry Sean O'Shea      11 Free  Point      1     7
3    35     1 Kerry Sean O'Shea      11 Free  Point      1     8
```



```
> match
# A tibble: 34 x 9
    Time  Half Team   Scorer          Number From       Type  Points Score
   <dbl> <dbl> <chr>  <chr>            <dbl> <chr>      <chr>  <dbl> <dbl>
 1     1     1 Dublin Paul Mannion        13 Play       Point      1     1
 2     2     1 Kerry  Sean O'Shea         11 Play       Point      1     1
 3     3     1 Dublin Dean Rock           14 Play       Point      1     2
 4     4     1 Dublin Dean Rock           14 Free       Point      1     3
 5    10     1 Kerry  David Clifford      13 Play       Point      1     2
 6    13     1 Kerry  Sean O'Shea         11 FortyFive Point      1     3
 7    14     1 Kerry  Stephen O'Brien     12 Play       Point      1     4
 8    16     1 Dublin Paul Mannion        14 Play       Point      1     4
 9    18     1 Kerry  Sean O'Shea         11 Free       Point      1     5
10    19     1 Dublin Jack McCaffrey       5 Play       Goal       3     7
# … with 24 more rows
```