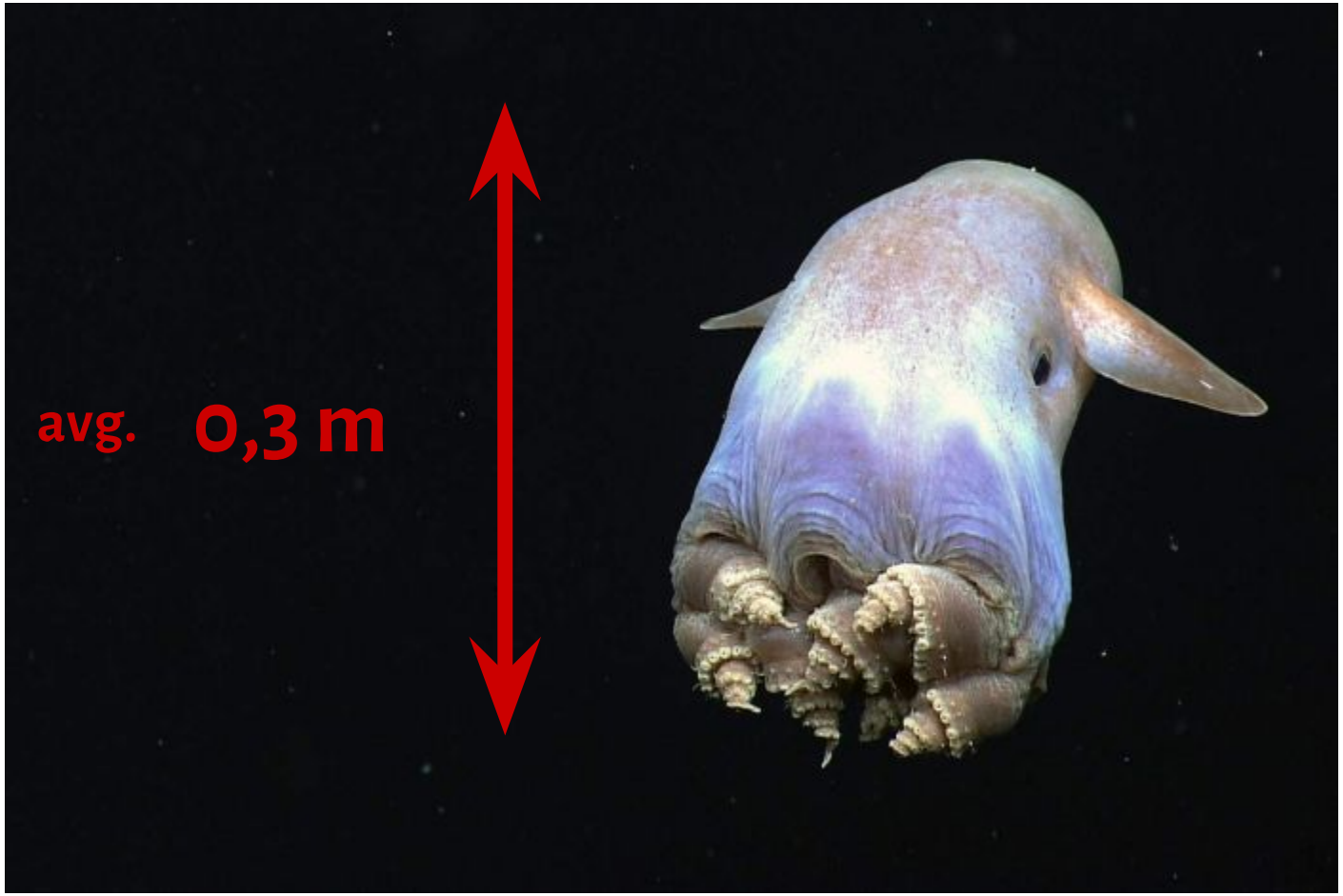
# Operating Systems

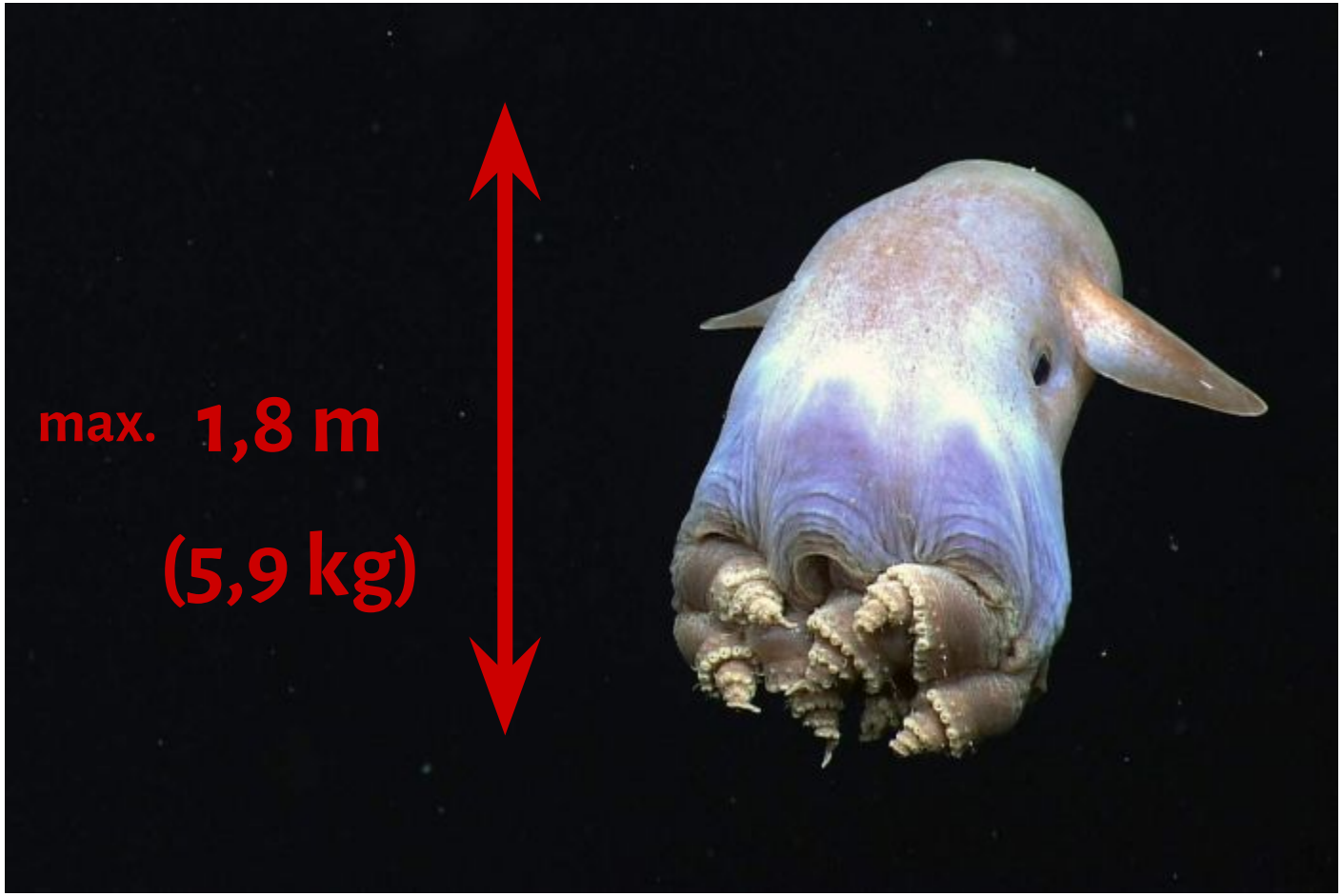
2018

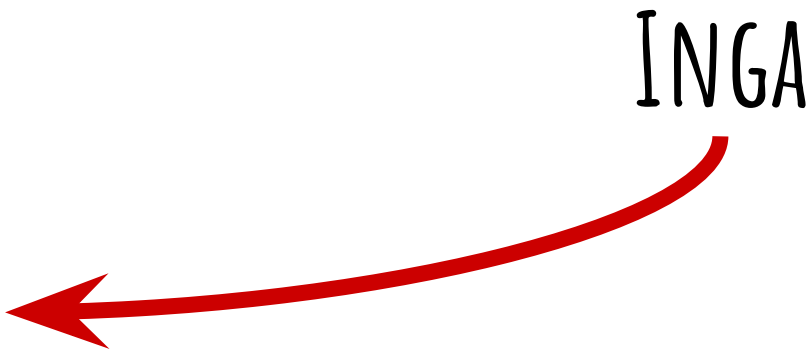
# *Grimpoteuthis* (dumbo octopus)

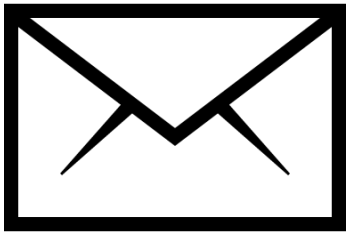
# *Grimpoteuthis* (dumbo octopus)



# *Grimpoteuthis* (dumbo octopus)




Inga



TO:

**inga.rub@mimuw.edu.pl**

SUBJECT: **[SO 2018] ...**

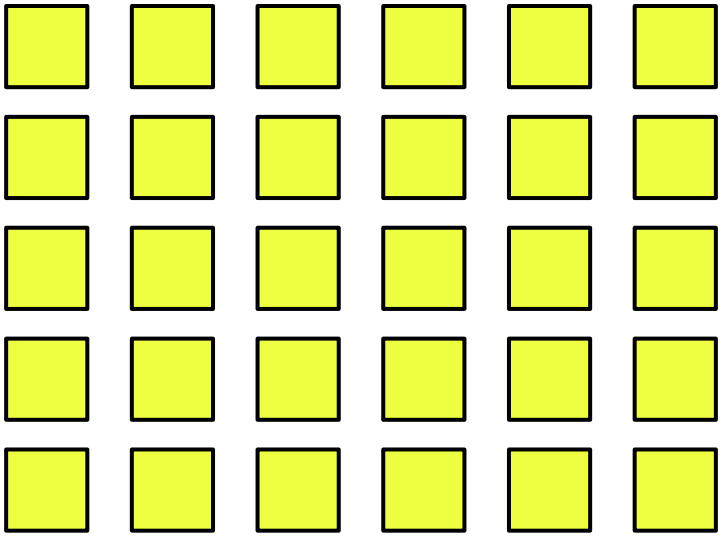
INGA

## The schedule:

- ★ Assembly
- ★ MINIX, qemu, workflow
- ★ System calls
- ★ Inter-process communication
- ★ Process scheduling
- ★ Filesystem
- ★ Drivers
- ★ Virtual memory

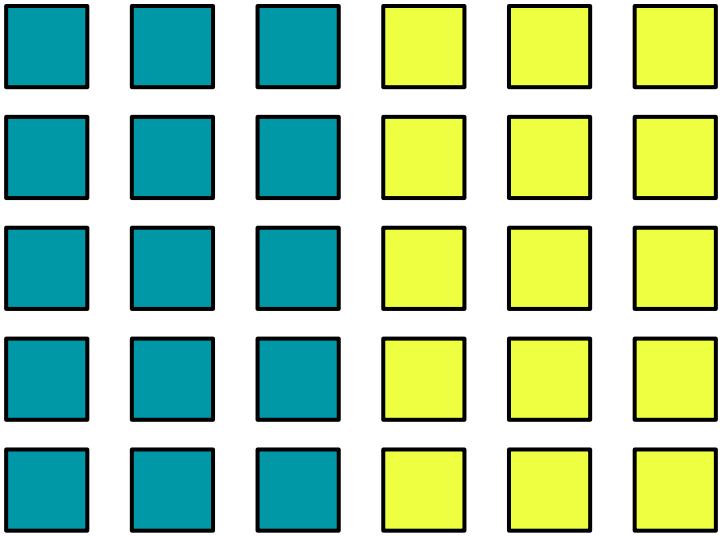
## Entertainment:

6 small assignments (each for **5** points)



## Entertainment:

6 small assignments (each for **5** points)

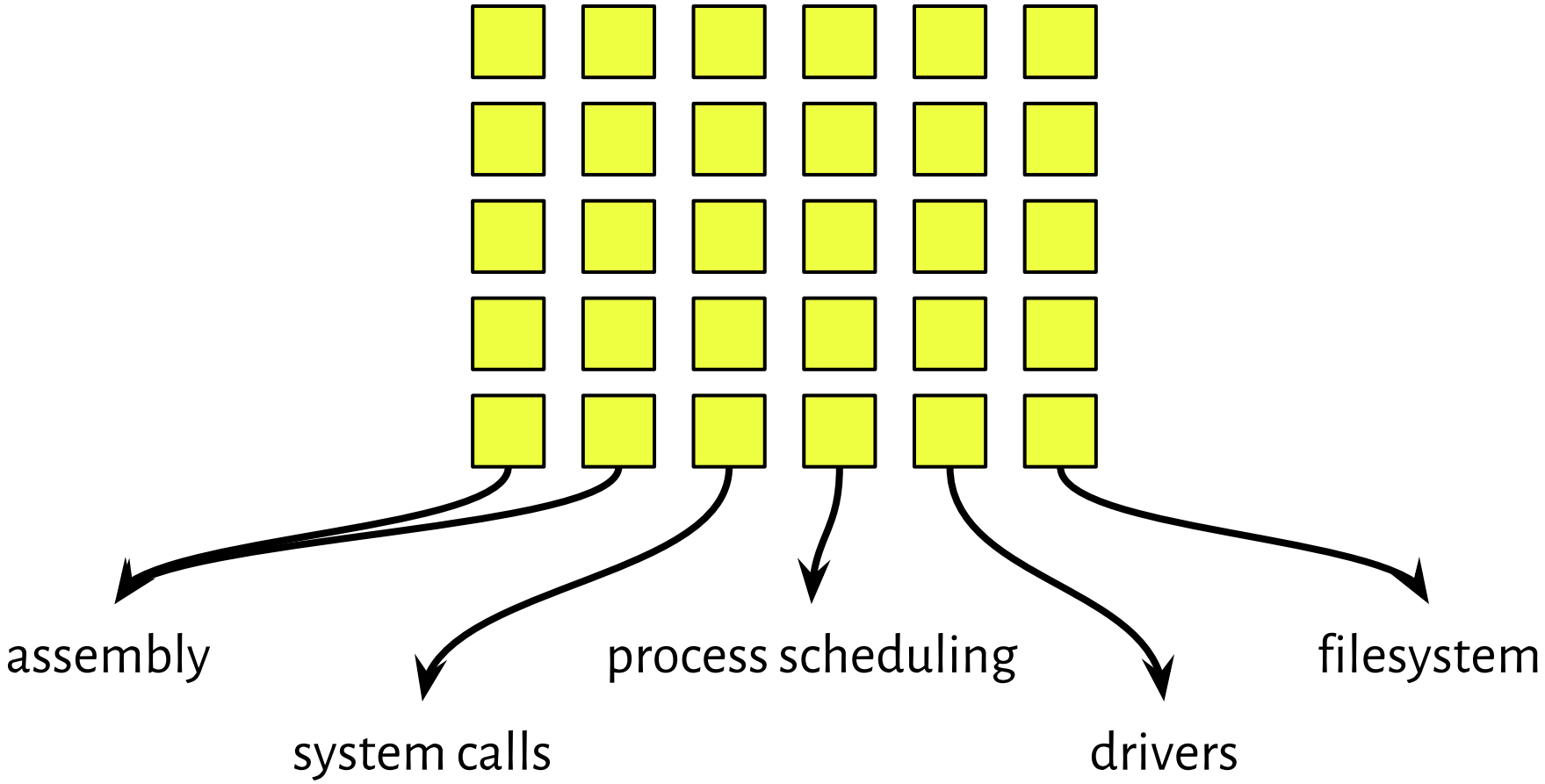


## The requirement:

at least **15** points out of 30

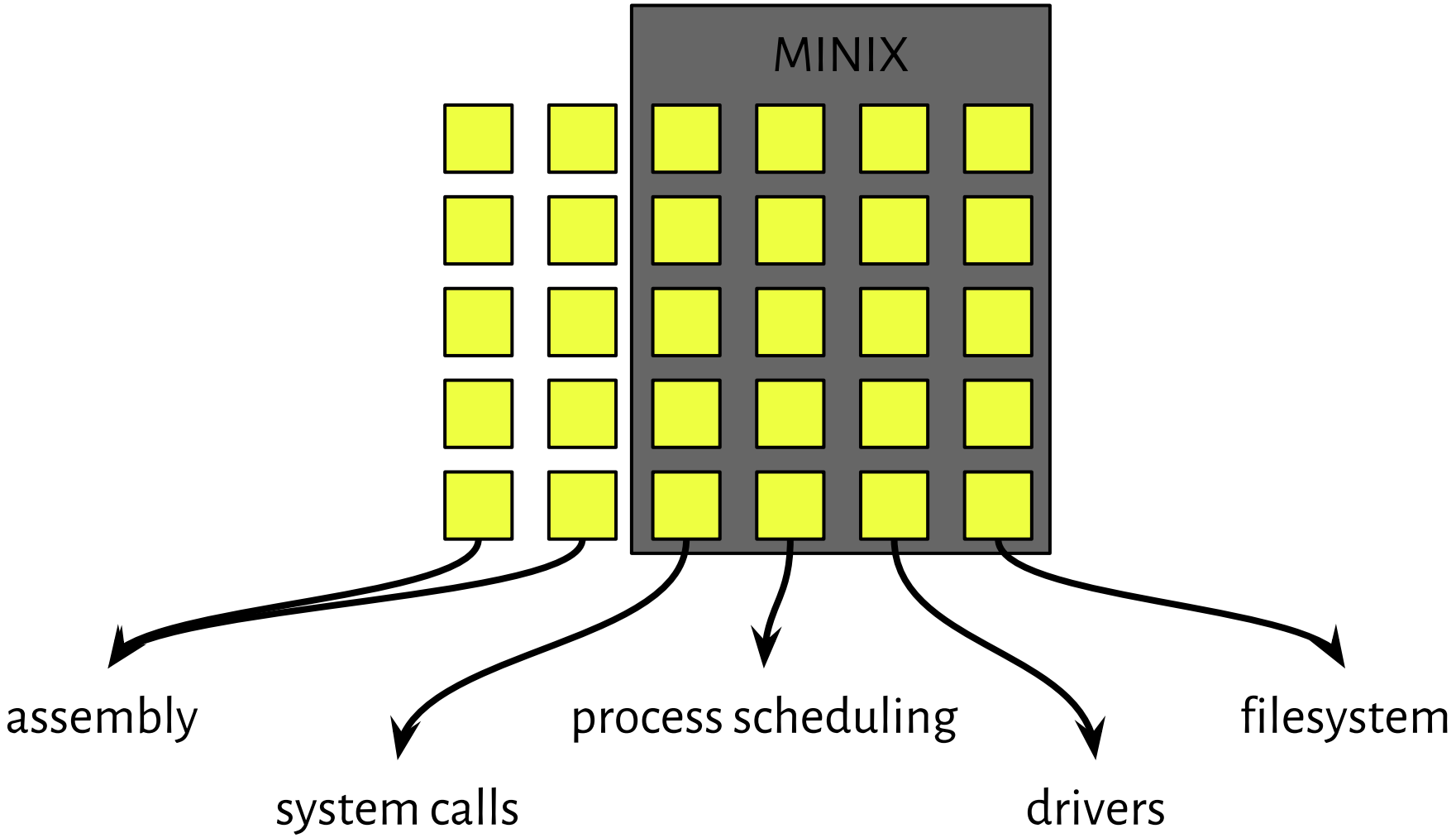
## Entertainment:

6 small assignments (each for **5** points)



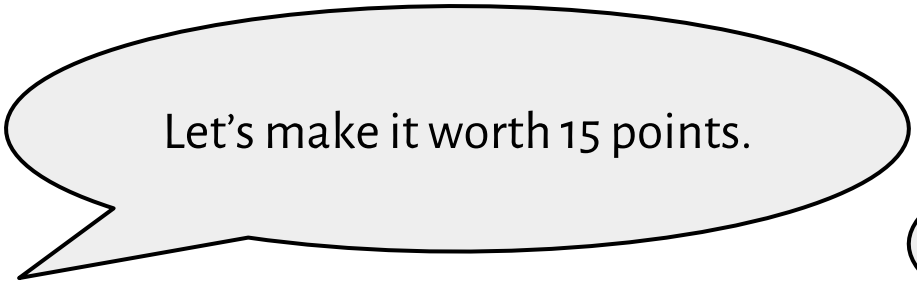
## Entertainment:

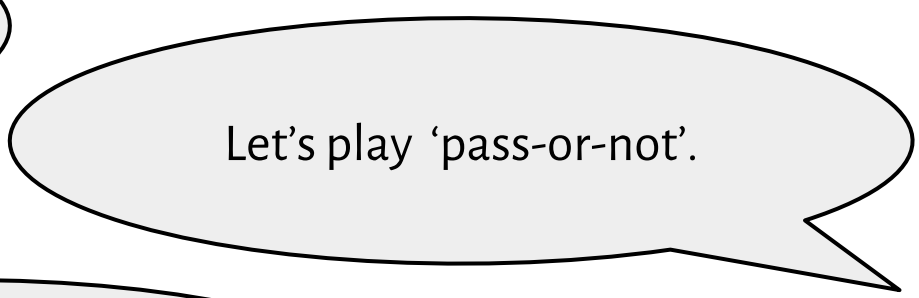
6 small assignments (each for **5** points)

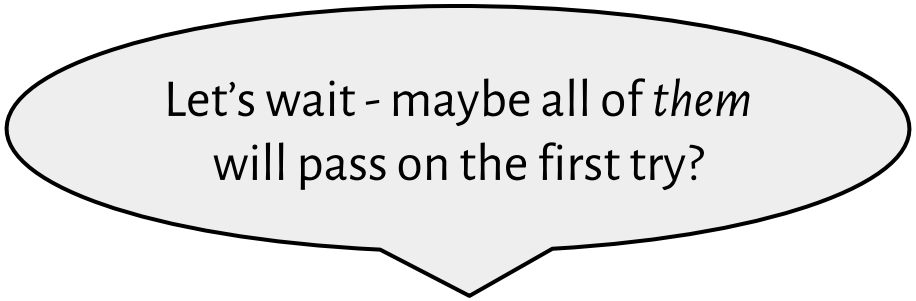


## What if I fail?

there will be a 2nd-chance assignment



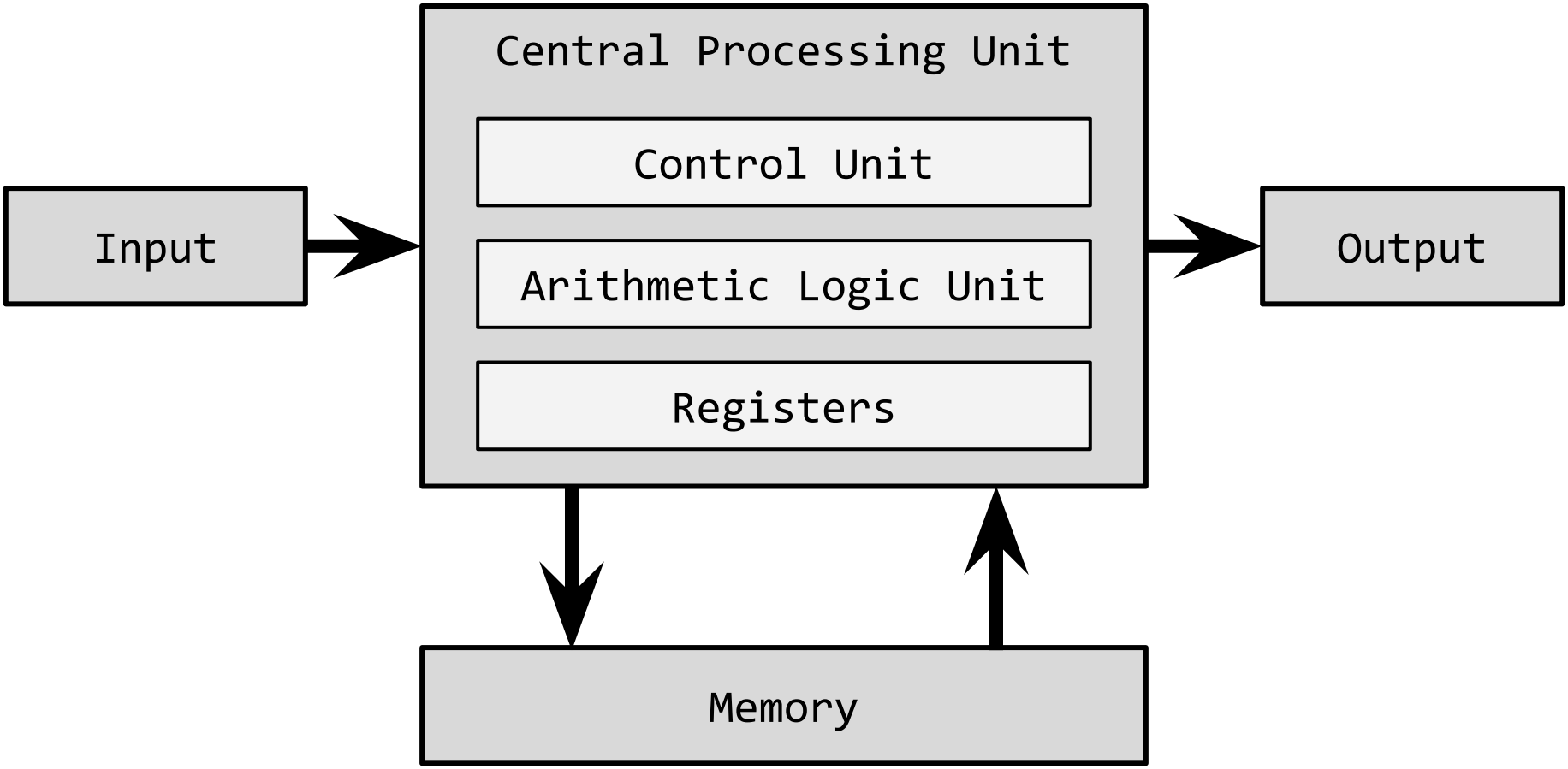

?

http://www.mimuw.edu.pl/~inga/SO2018

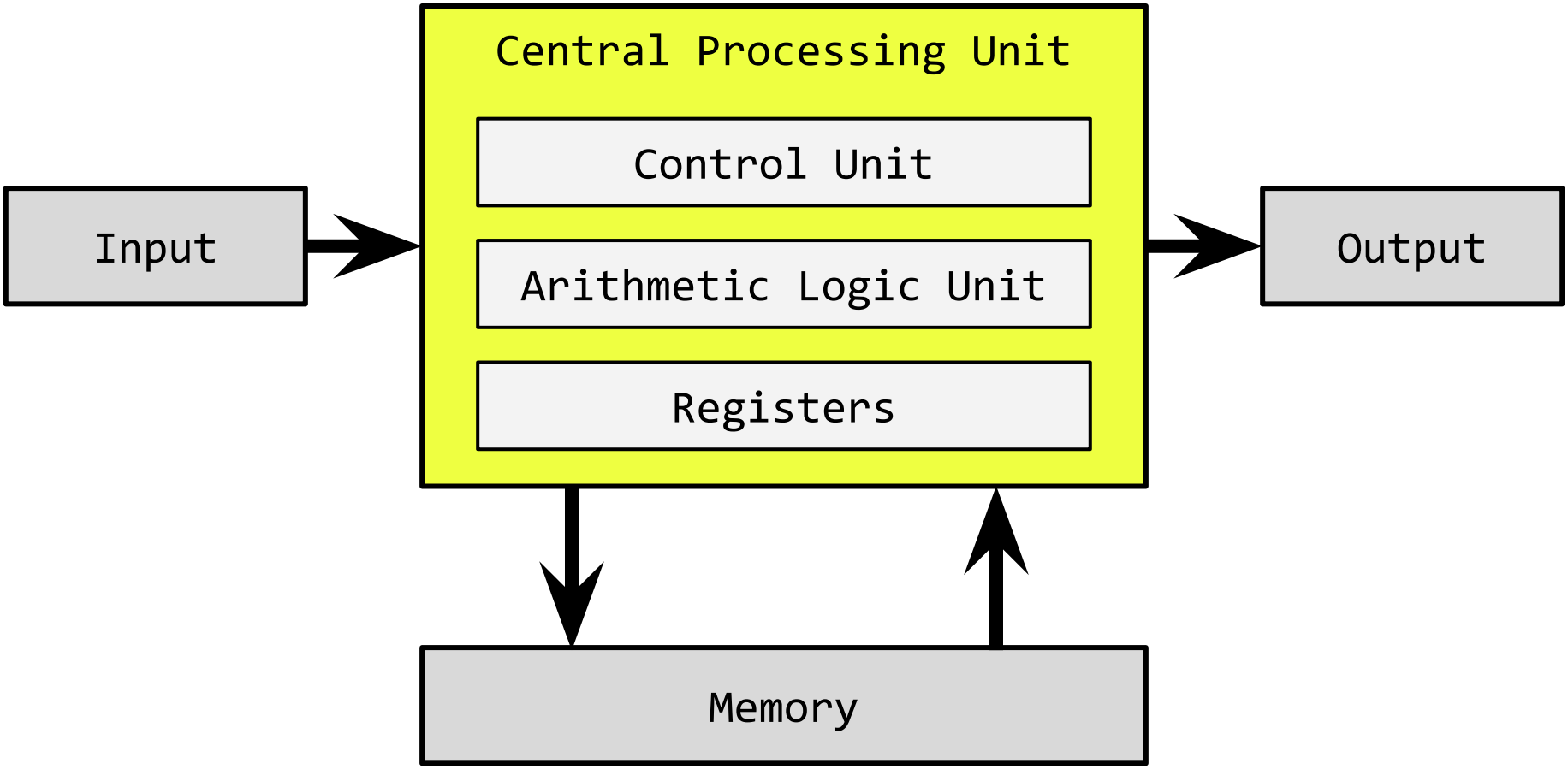
/home/students/inf/PUBLIC/SO**/scenariusze**

/home/students/inf/PUBLIC/SO**/zadania**

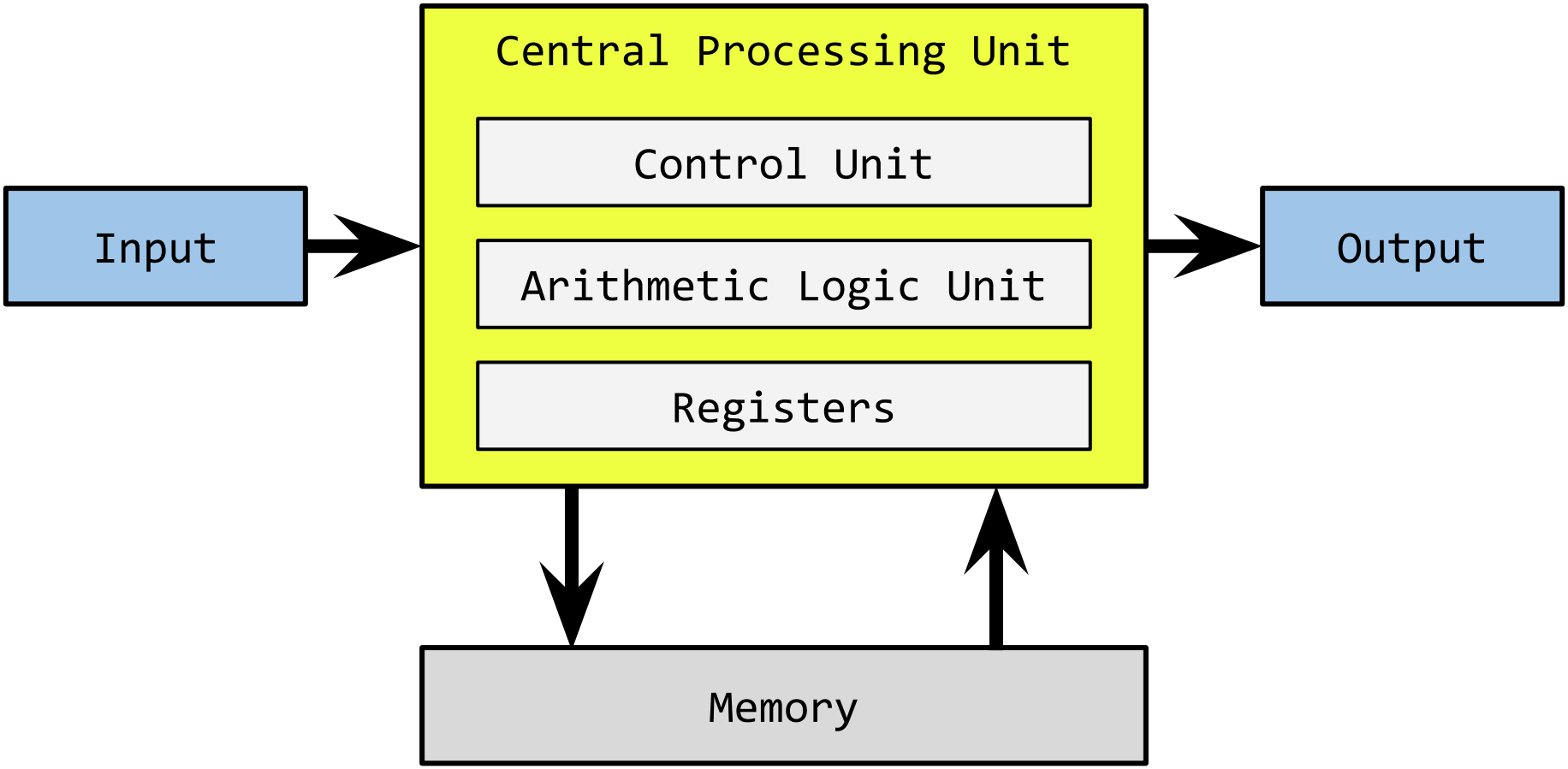
# Computer Architecture



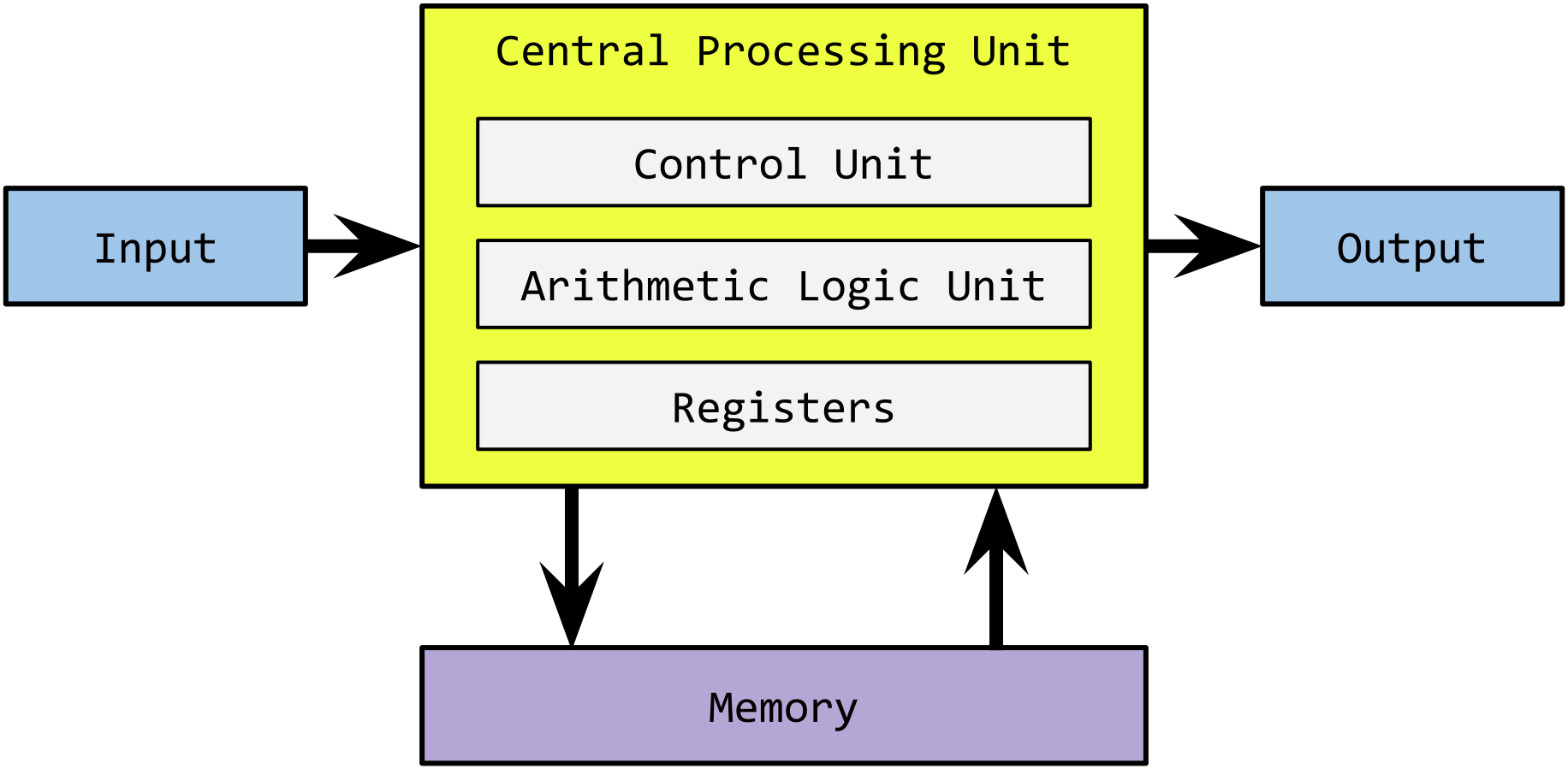
# Computer Architecture



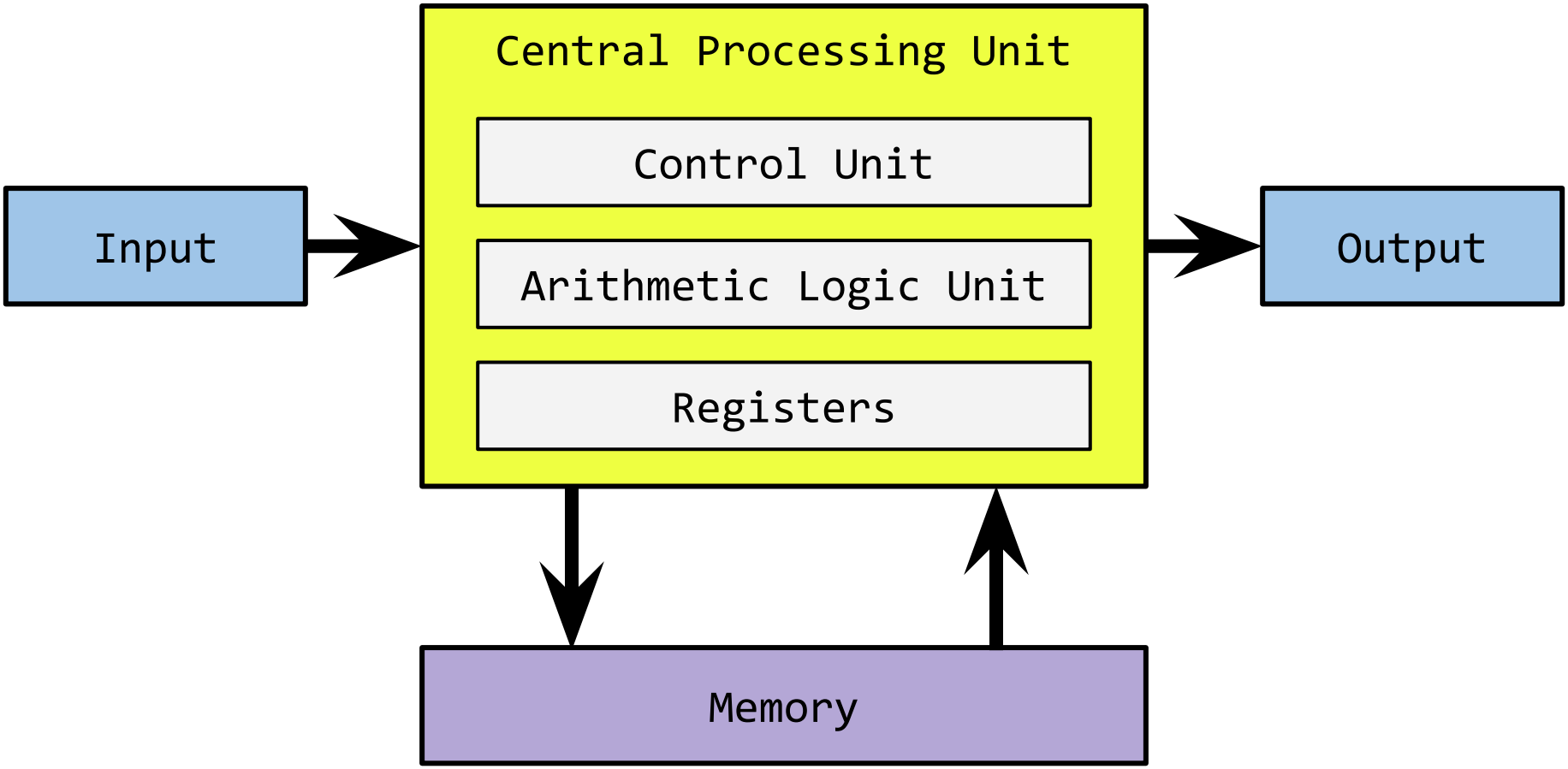
# Computer Architecture



# Computer Architecture



# Computer Architecture




The Princeton Architecture

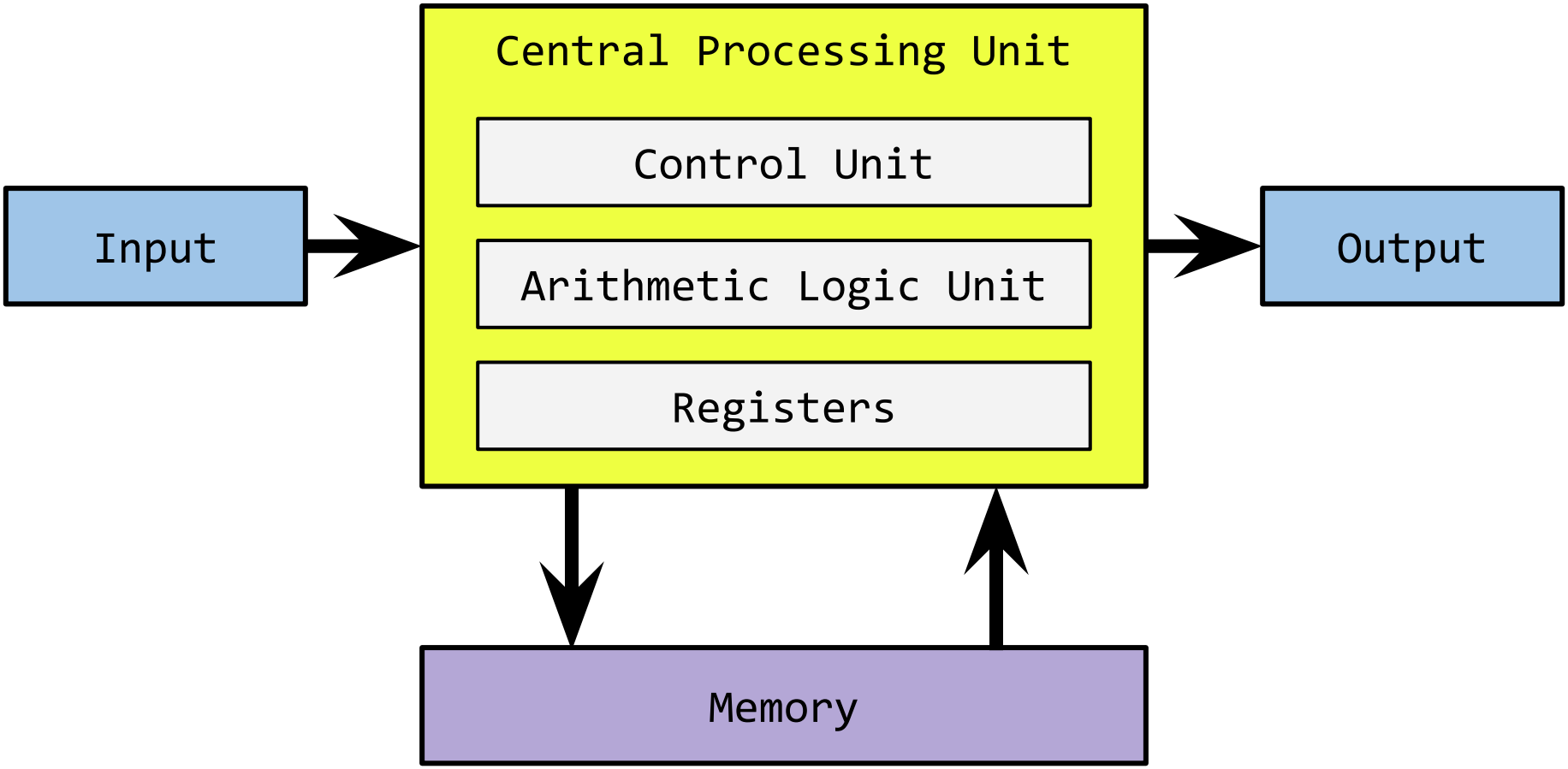
# The Princeton Architecture

Alan Turing


Presper Eckert & John William Mauchly

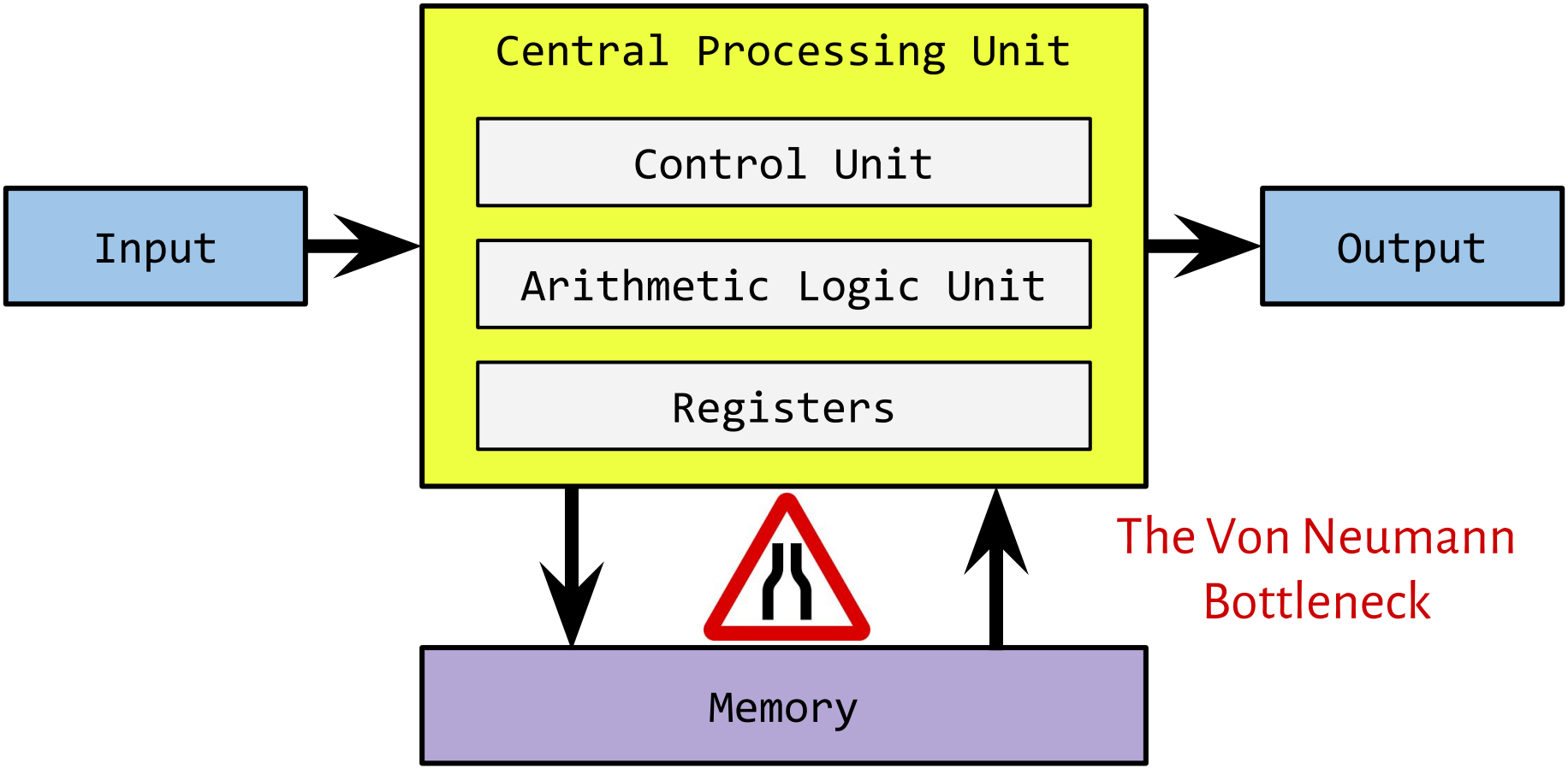
John von Neumann

# Computer Architecture



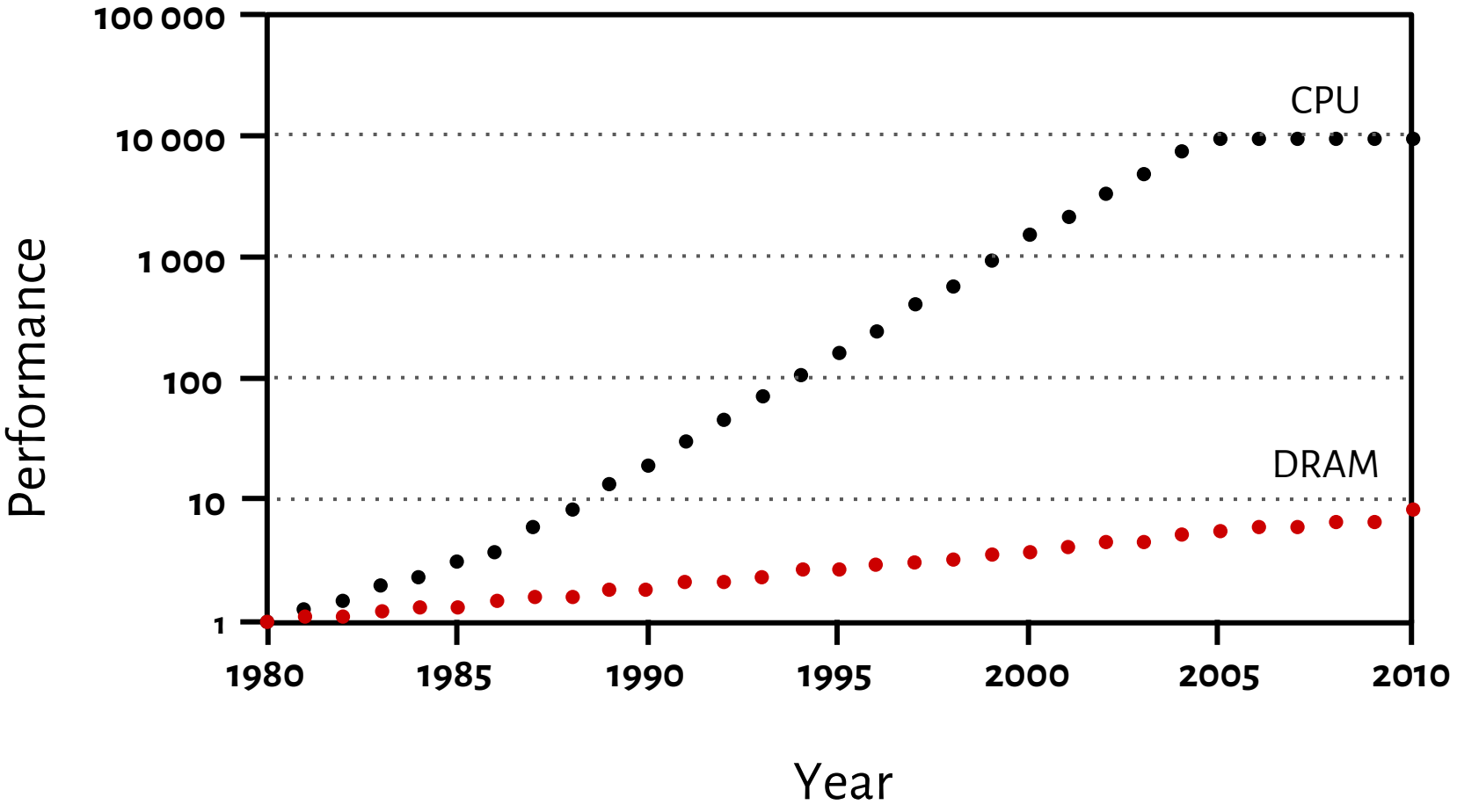

The Von Neumann Architecture

# Computer Architecture



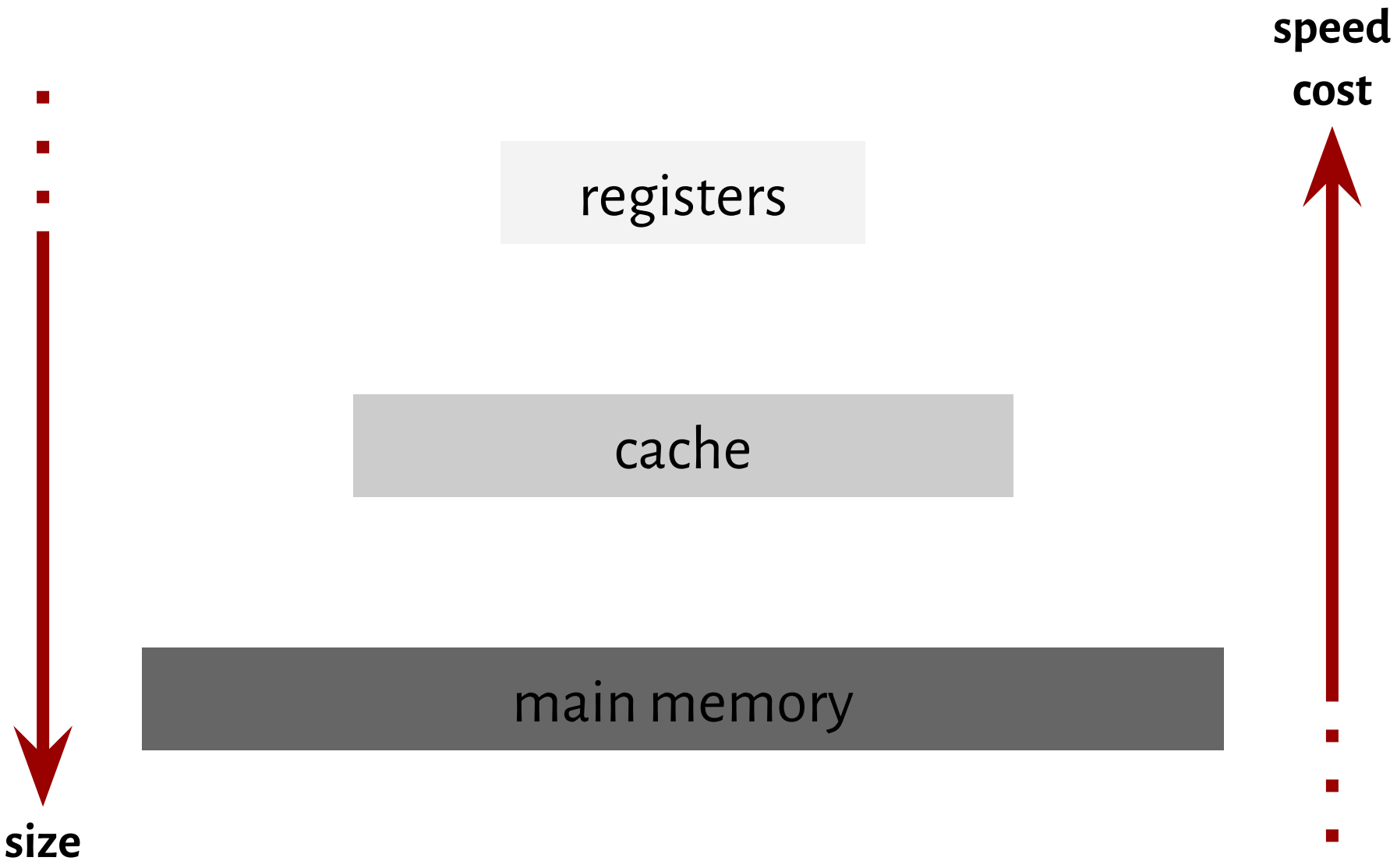

The Von Neumann Architecture

# Memory-access bottleneck

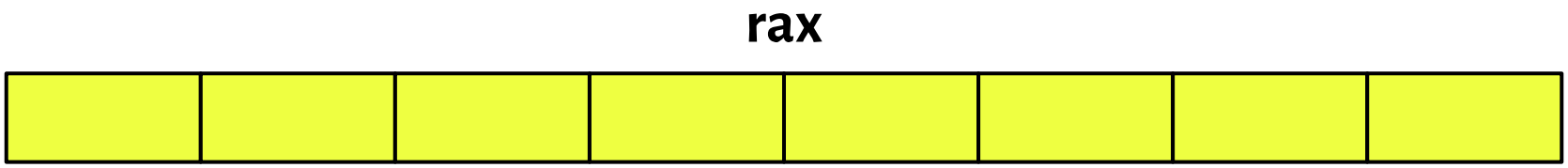



J. L. Hennessy, Stanford University and D. A. Patterson, University of California, Berkeley: *Memory Hierarchy Design*, 2012

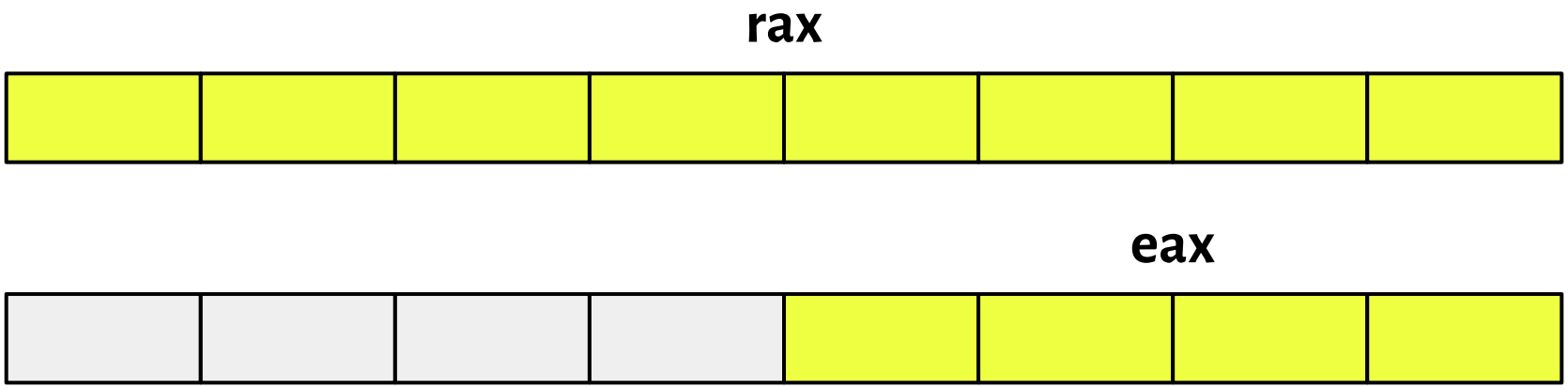
# Memory hierarchy



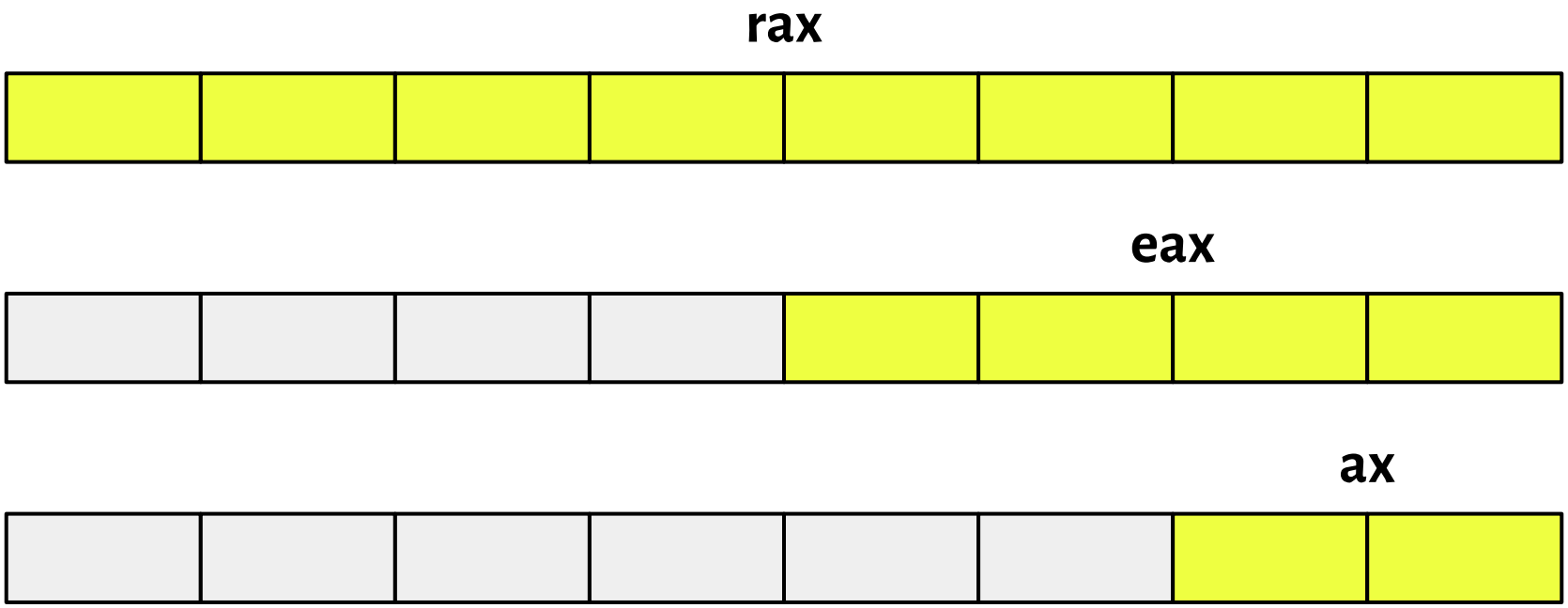
# General Purpose Registers



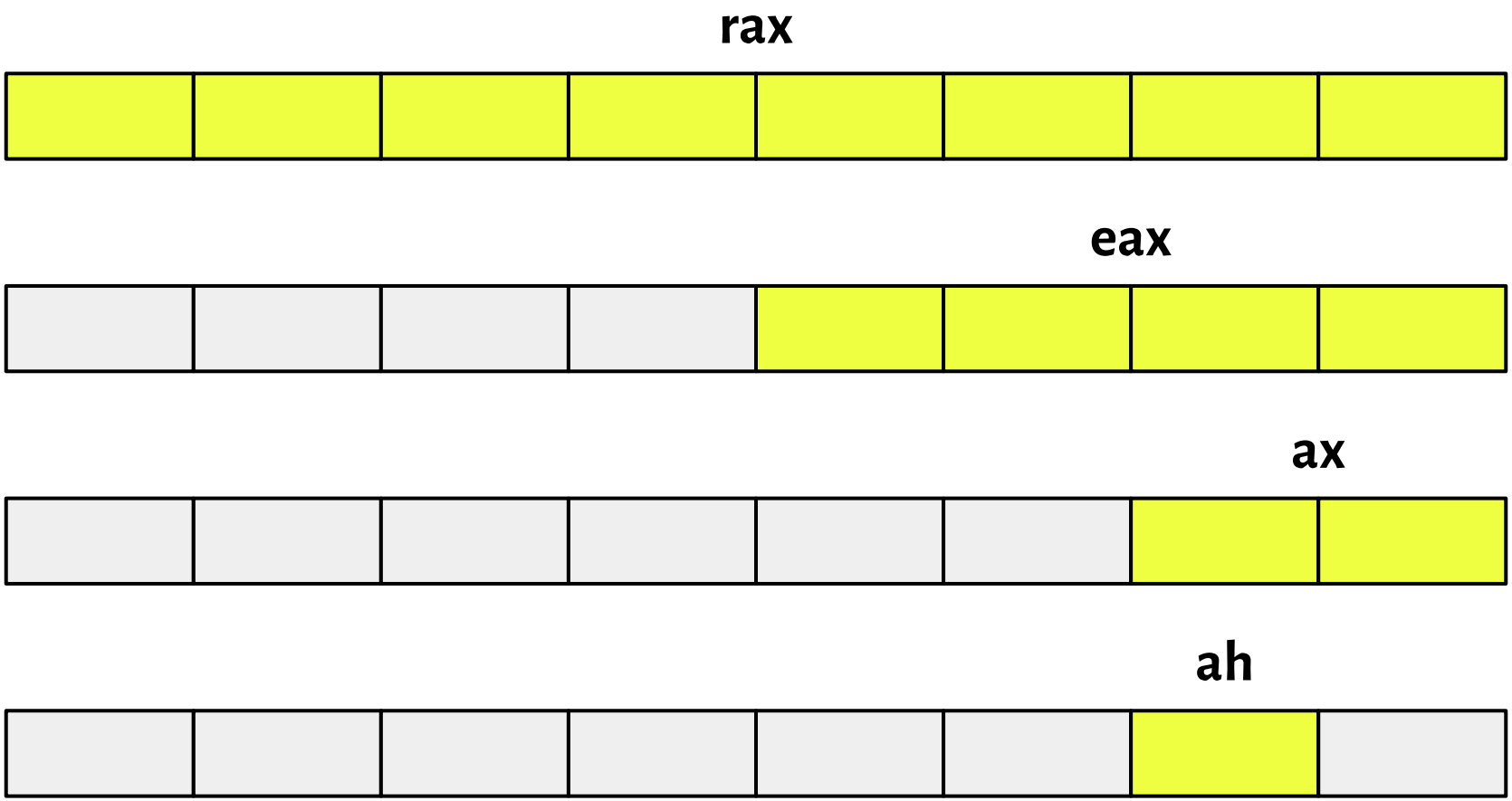
# General Purpose Registers



# General Purpose Registers

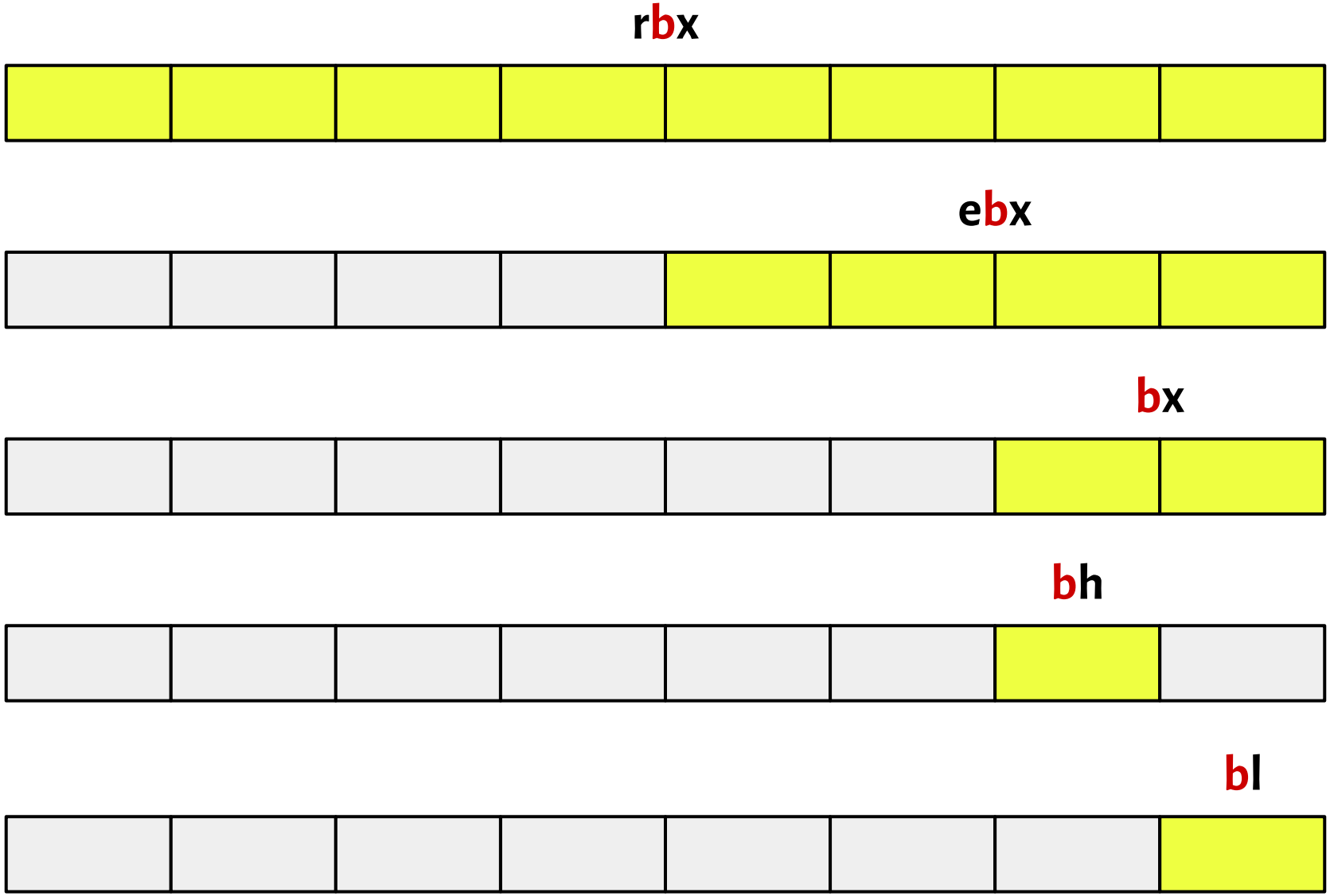


# General Purpose Registers



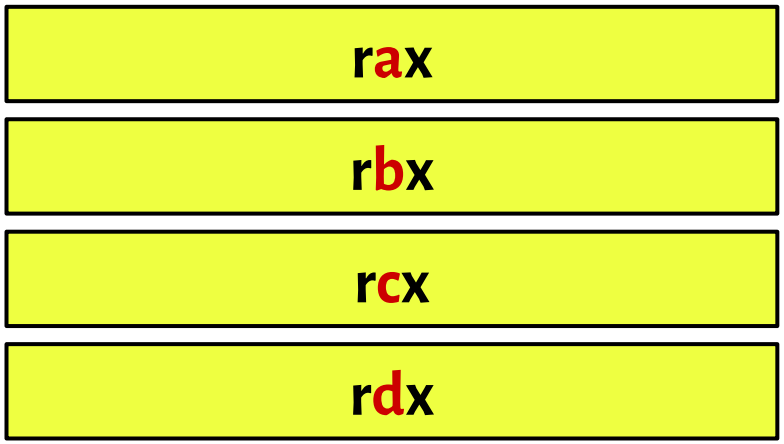
# General Purpose Registers

# General Purpose Registers



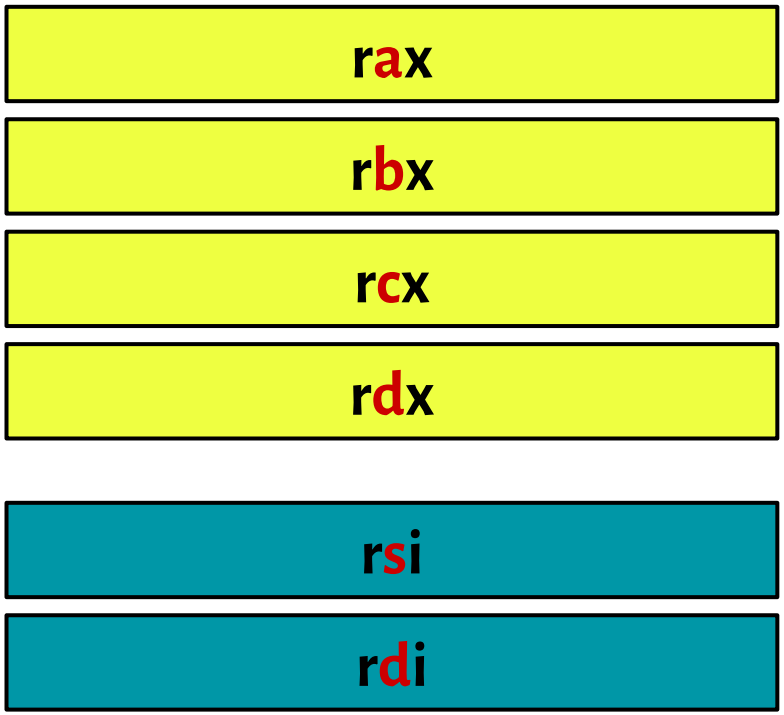
# General Purpose Registers

In **x86-64** architecture: 16 General Purpose Registers



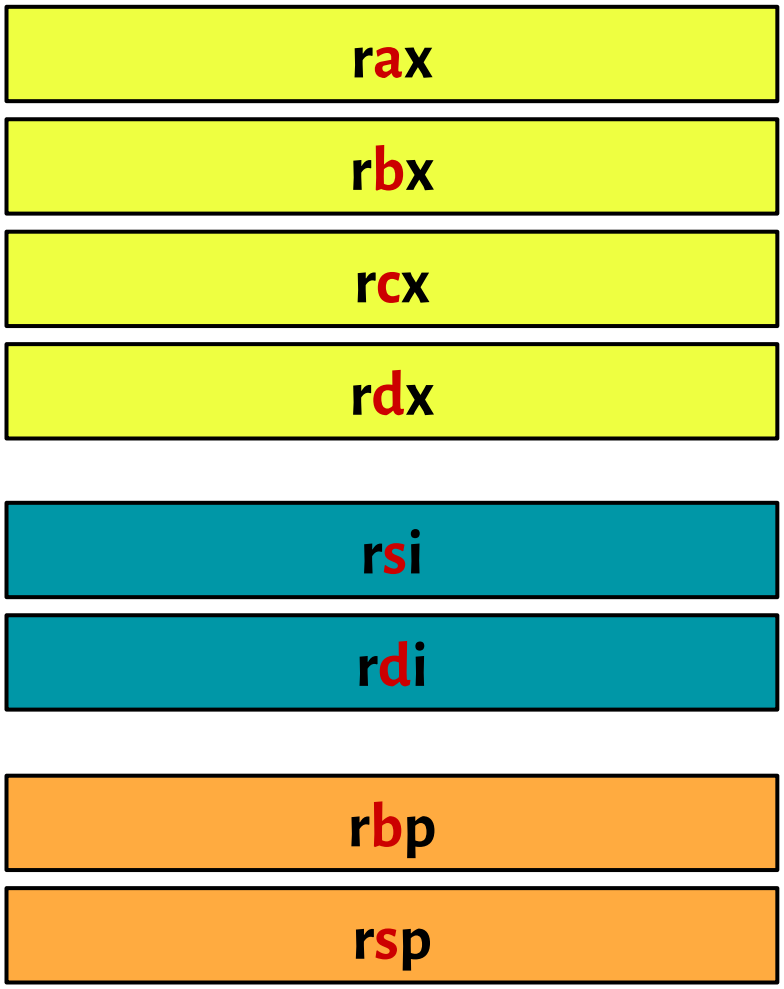
# General Purpose Registers

In **x86-64** architecture: 16 General Purpose Registers



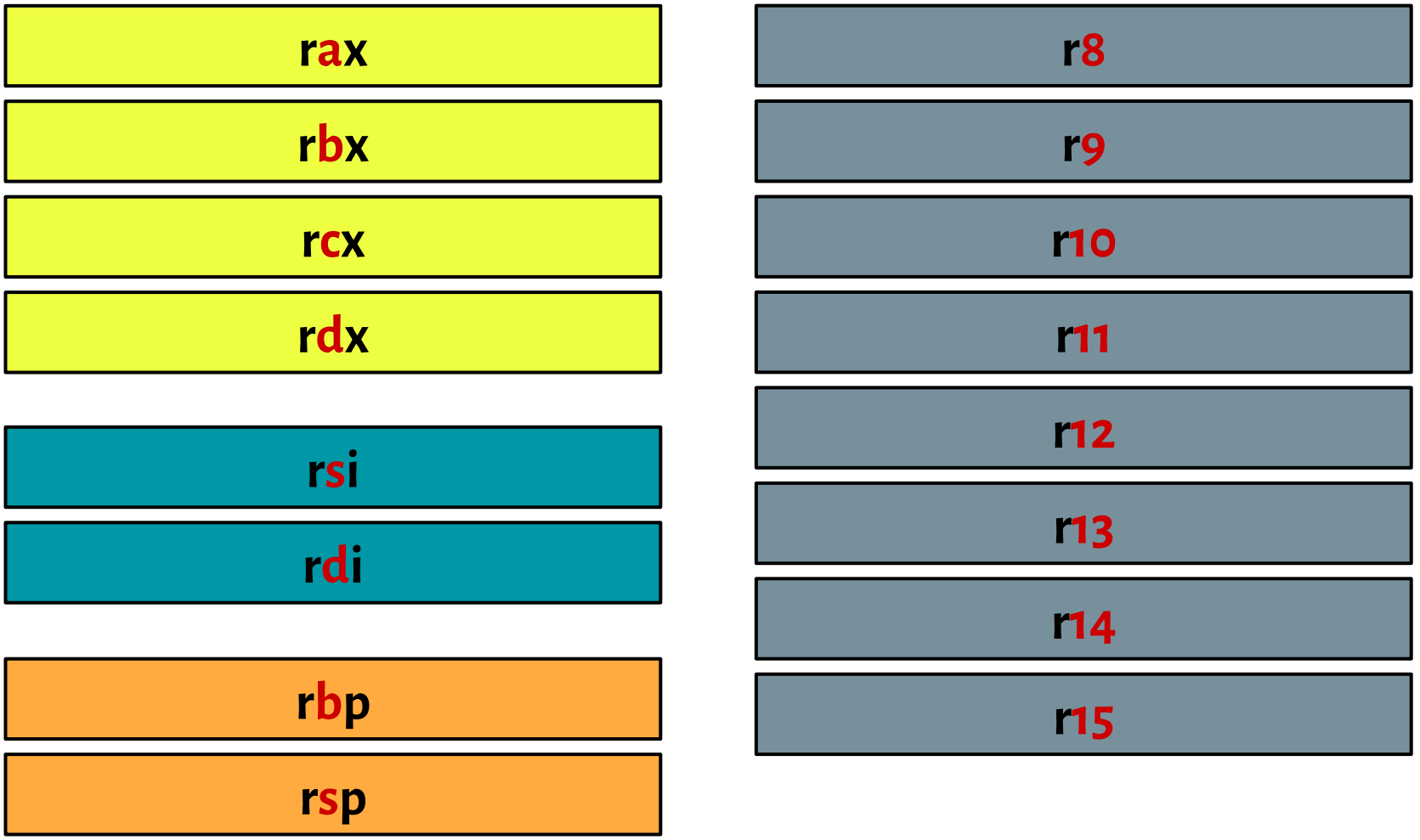
# General Purpose Registers

In **x86-64** architecture: 16 General Purpose Registers



# General Purpose Registers

In **x86-64** architecture: 16 General Purpose Registers



# Other Registers

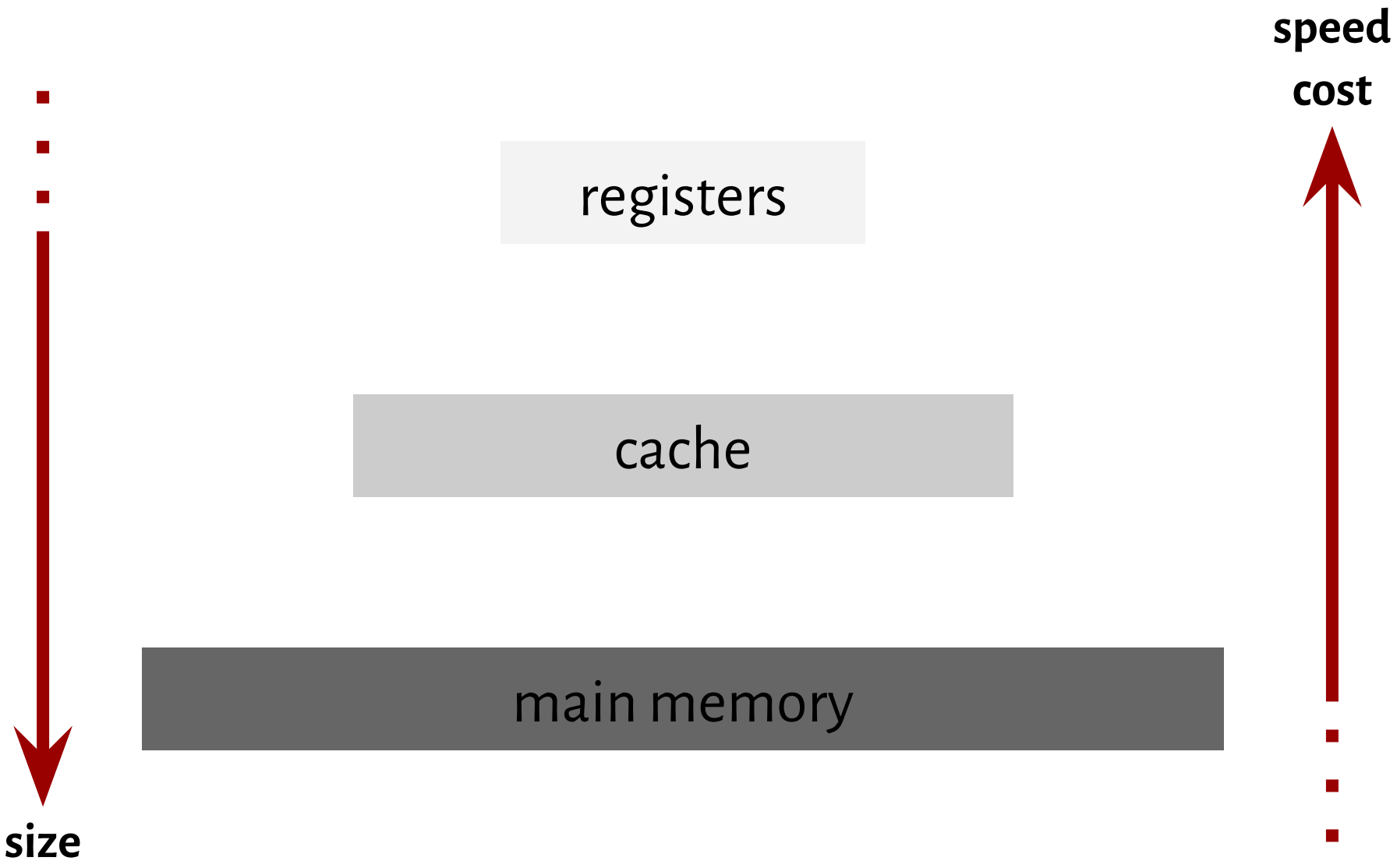
Instruction Pointer Register (rip)

Flag Registers (rFlags)

XMM registers

and other other…

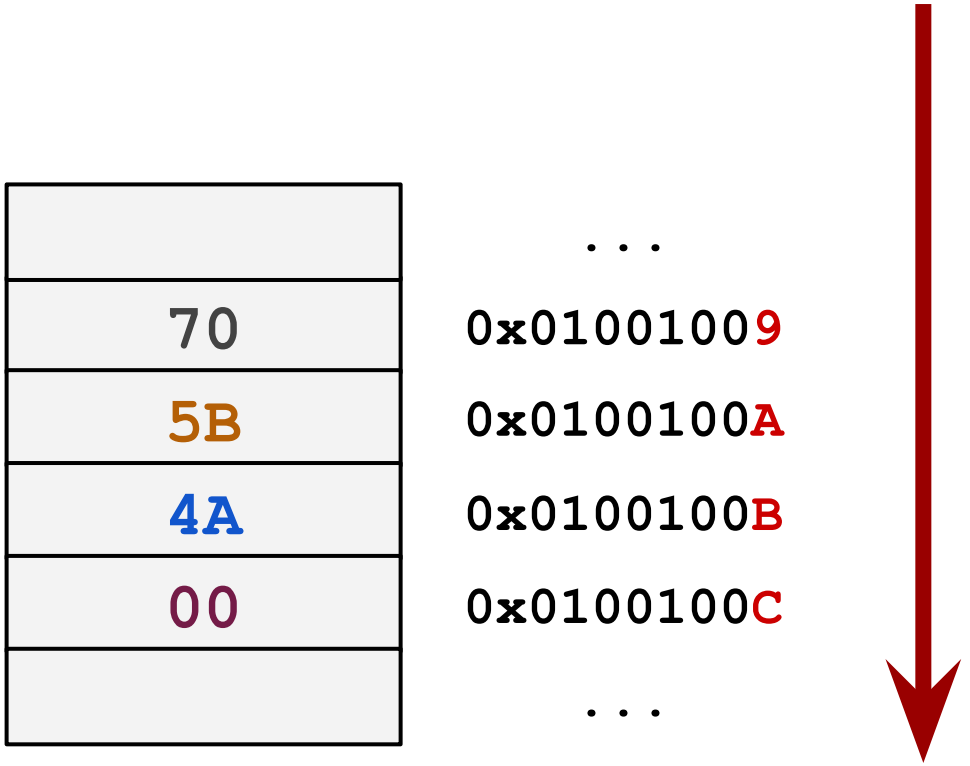
# Memory hierarchy



# Main Memory

# Main Memory

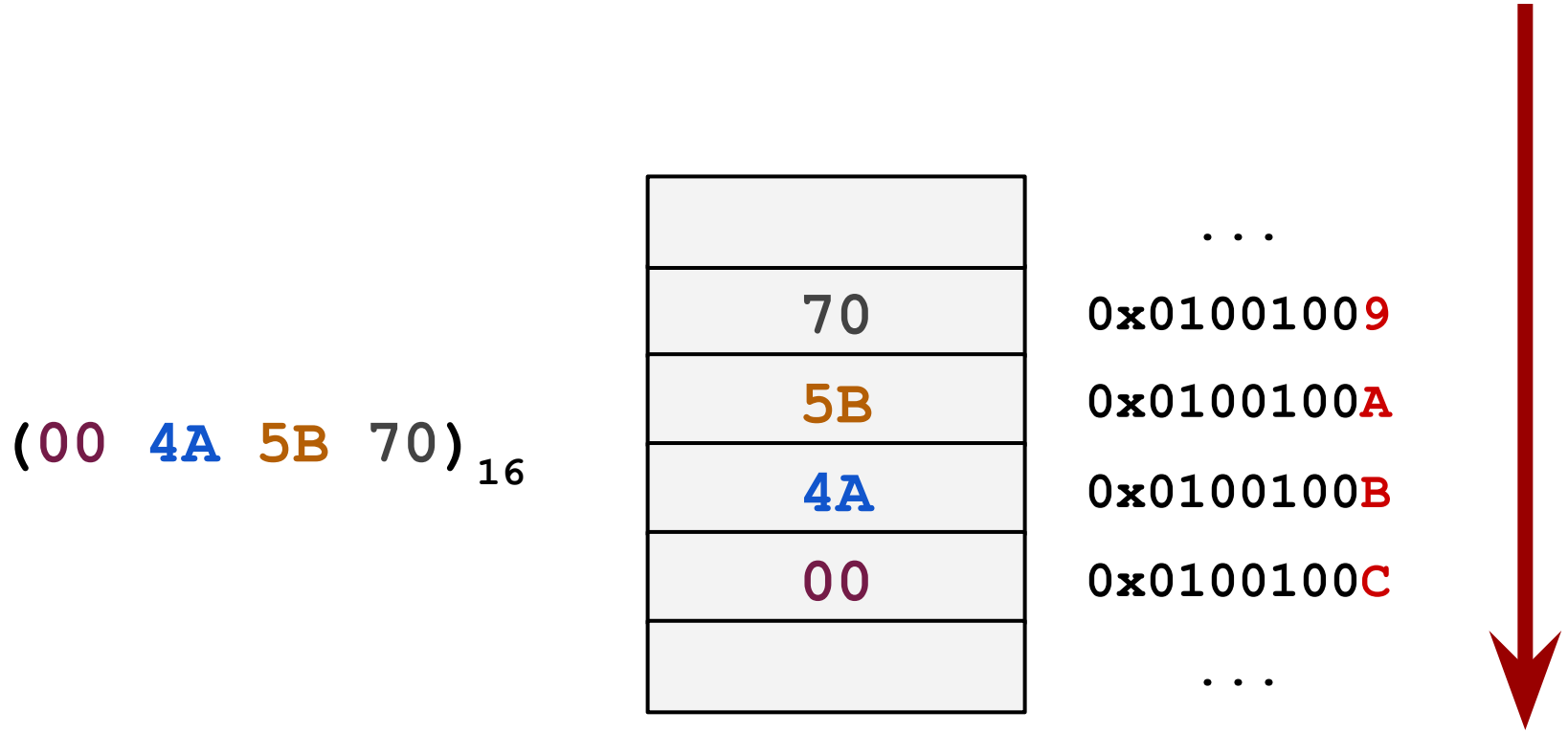
each memory address specifies a corresponding **byte**



# Main Memory

each memory address specifies a corresponding **byte**

little-endian: little end goes first



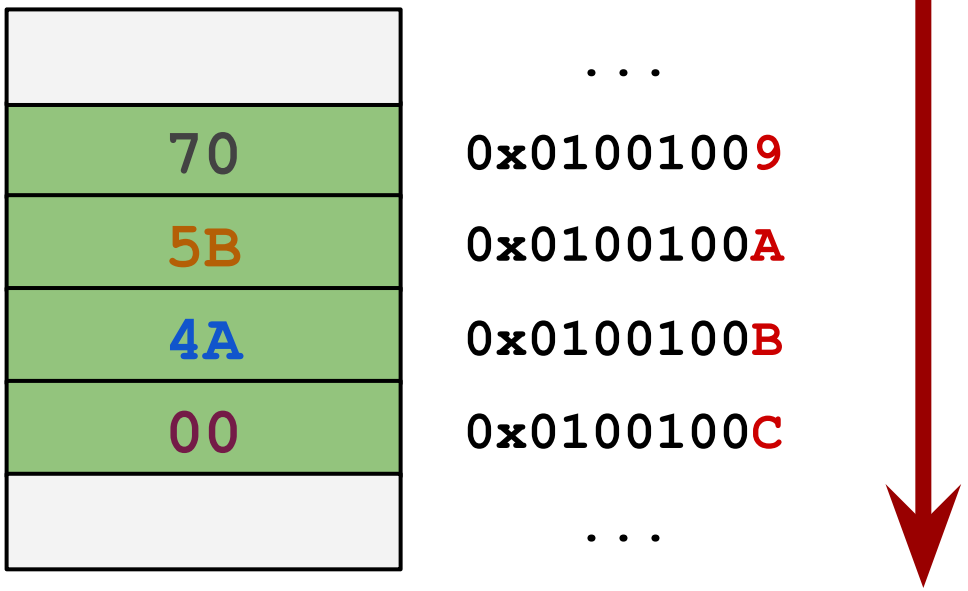
# Main Memory

each memory address specifies a corresponding **byte**

little-endian: little end goes first

a computer accesses memory by a single **memory word** at a time

$(00\ 4A\ 5B\ 70)_{16}$



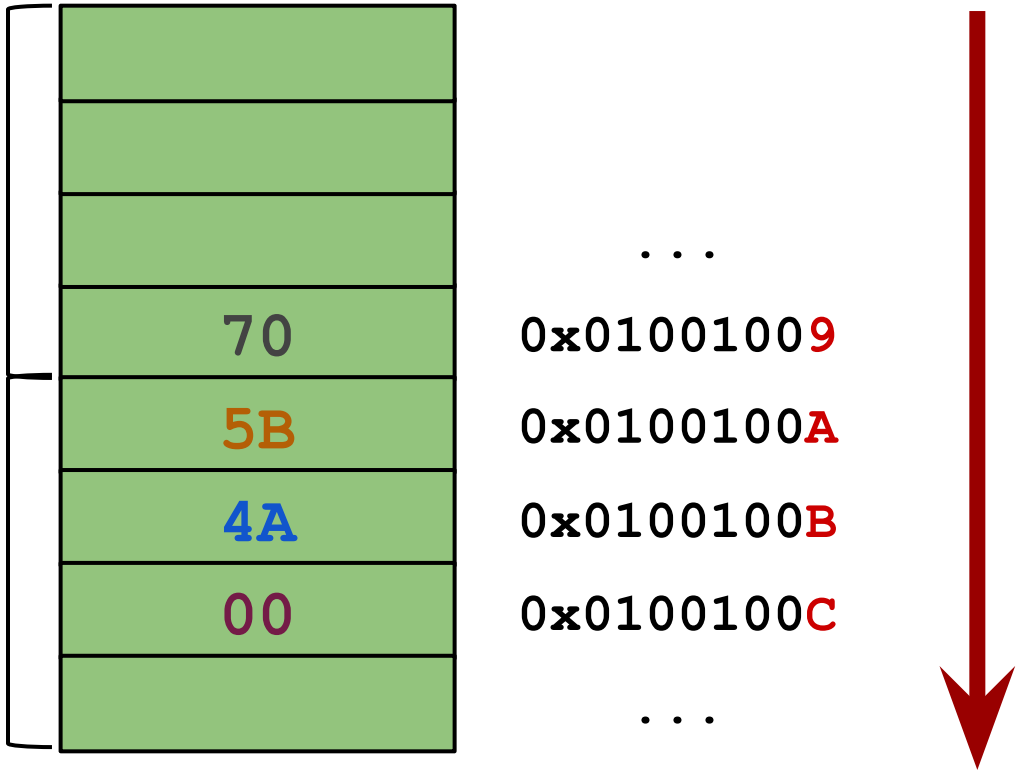
# Main Memory

each memory address specifies a corresponding **byte**

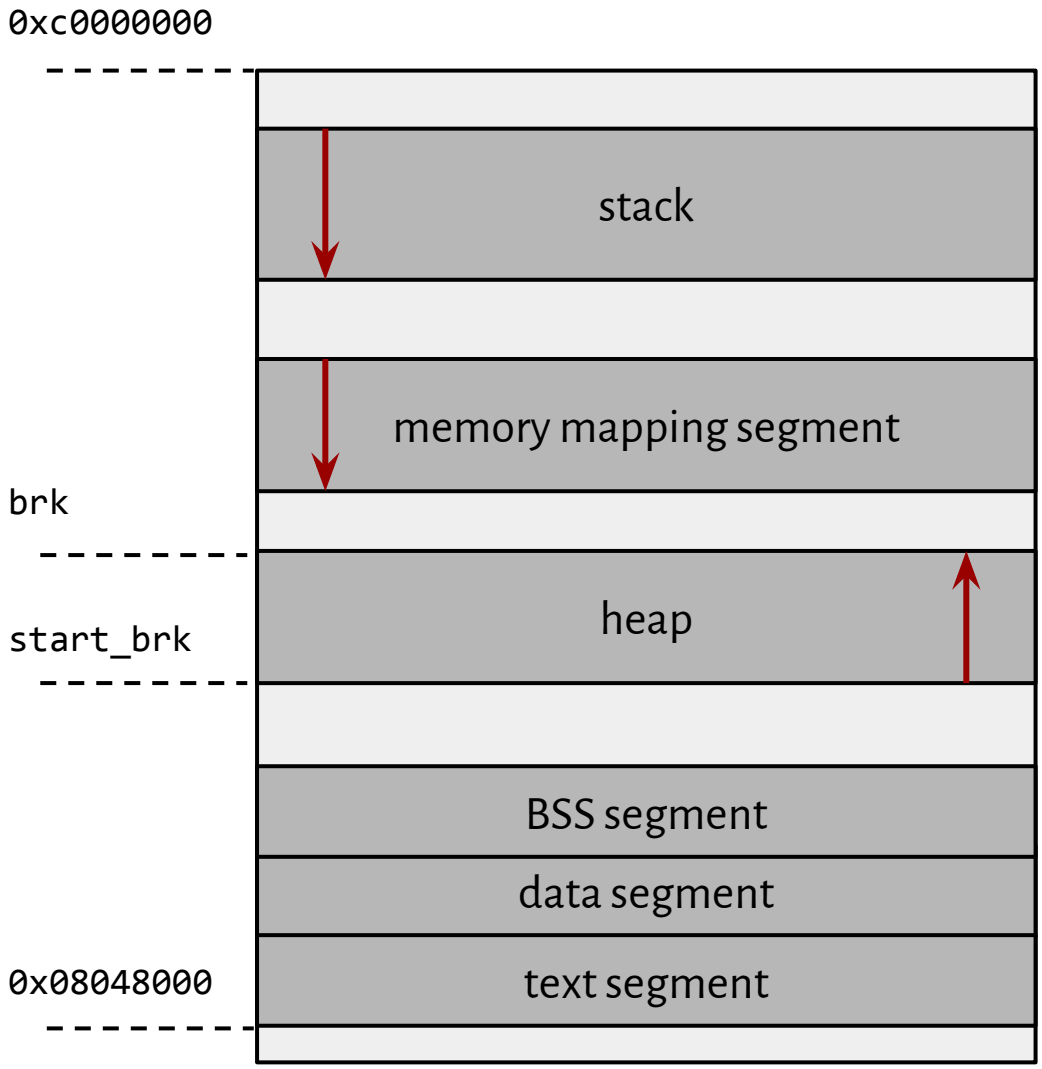
little-endian: little end goes first

a computer accesses memory by a single **memory word** at a time

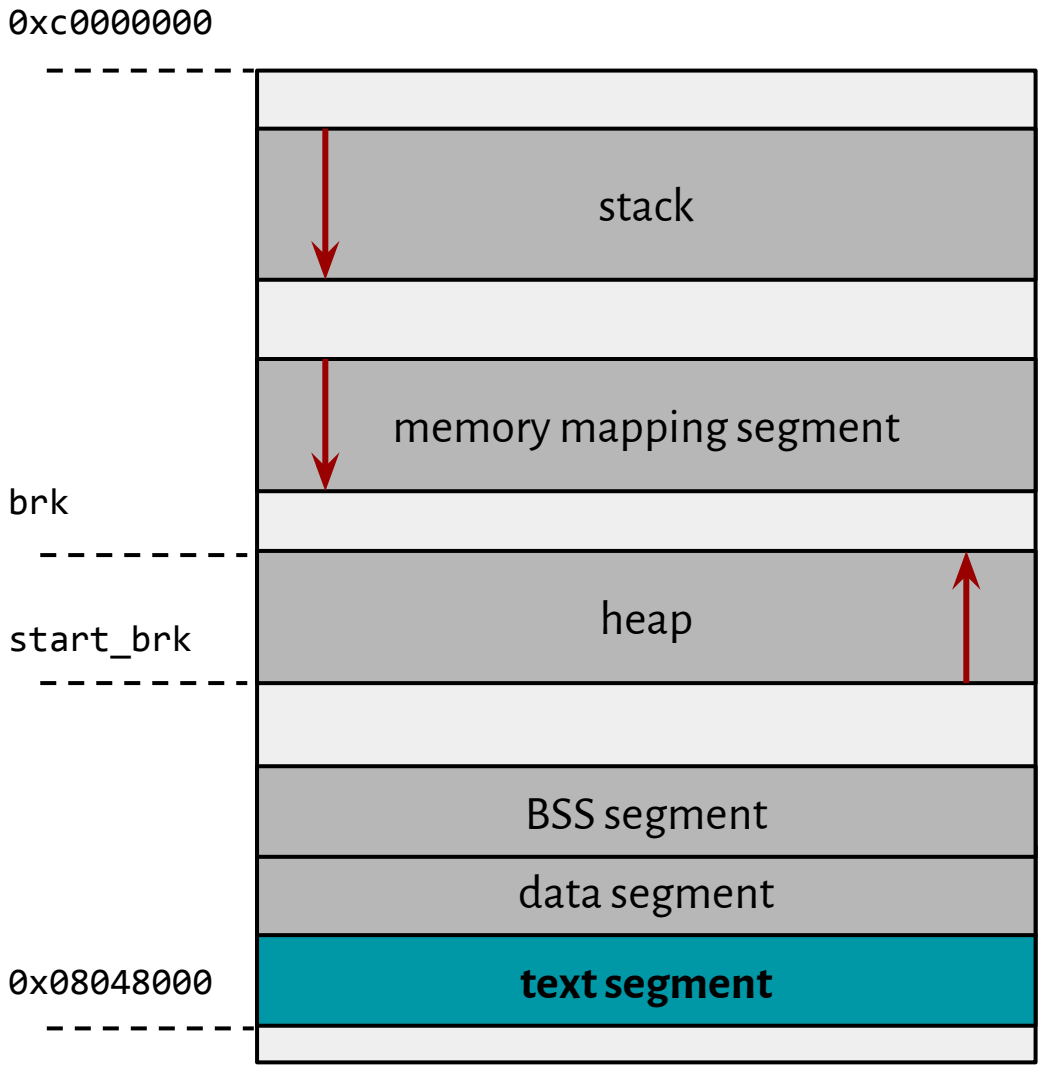
$(00\ 4A\ 5B\ 70)_{16}$



# Program Memory Layout



# Program Memory Layout

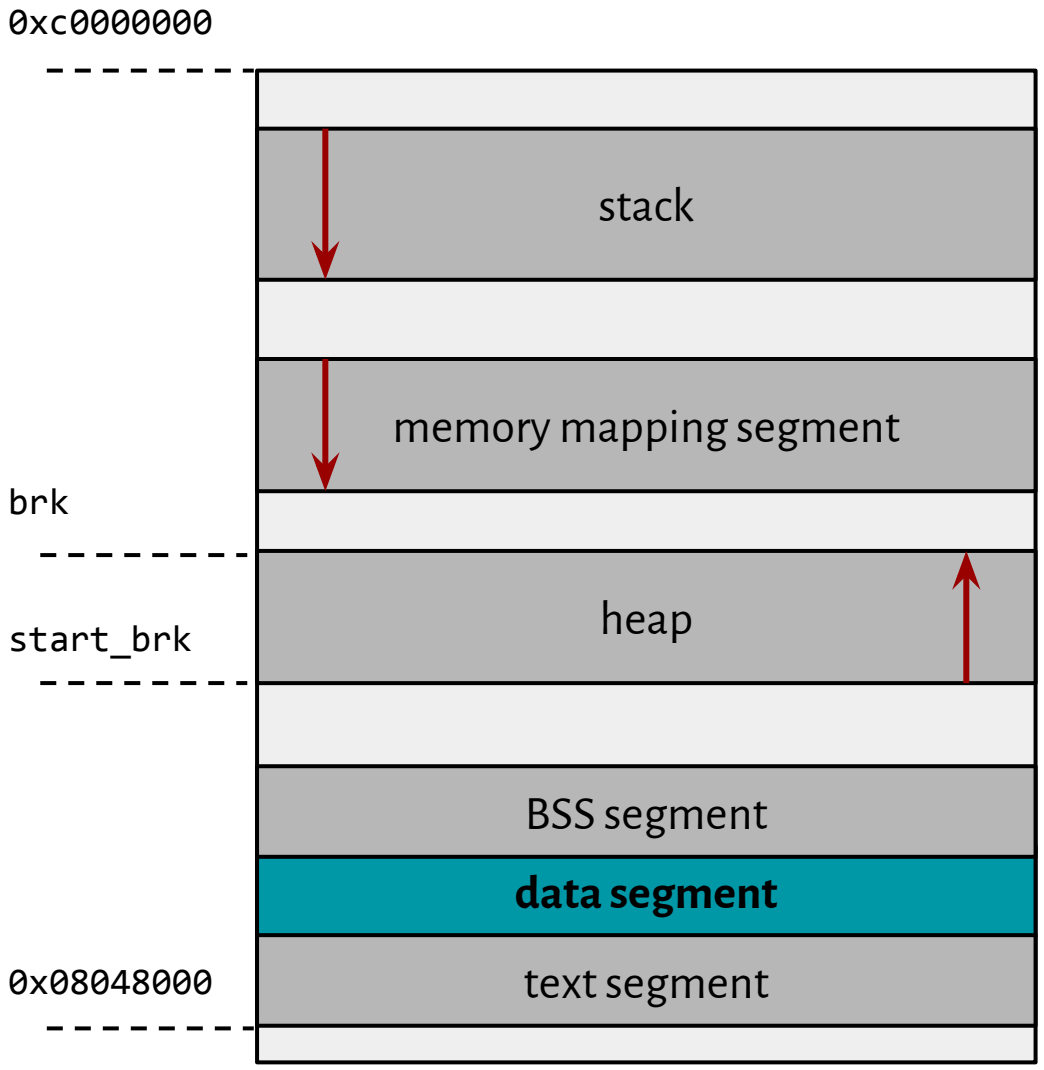



**text segment** (code segment)

corresponds to a part of an object file and contains executable instructions

- ★ read-only
- ★ fixed size

# Program Memory Layout

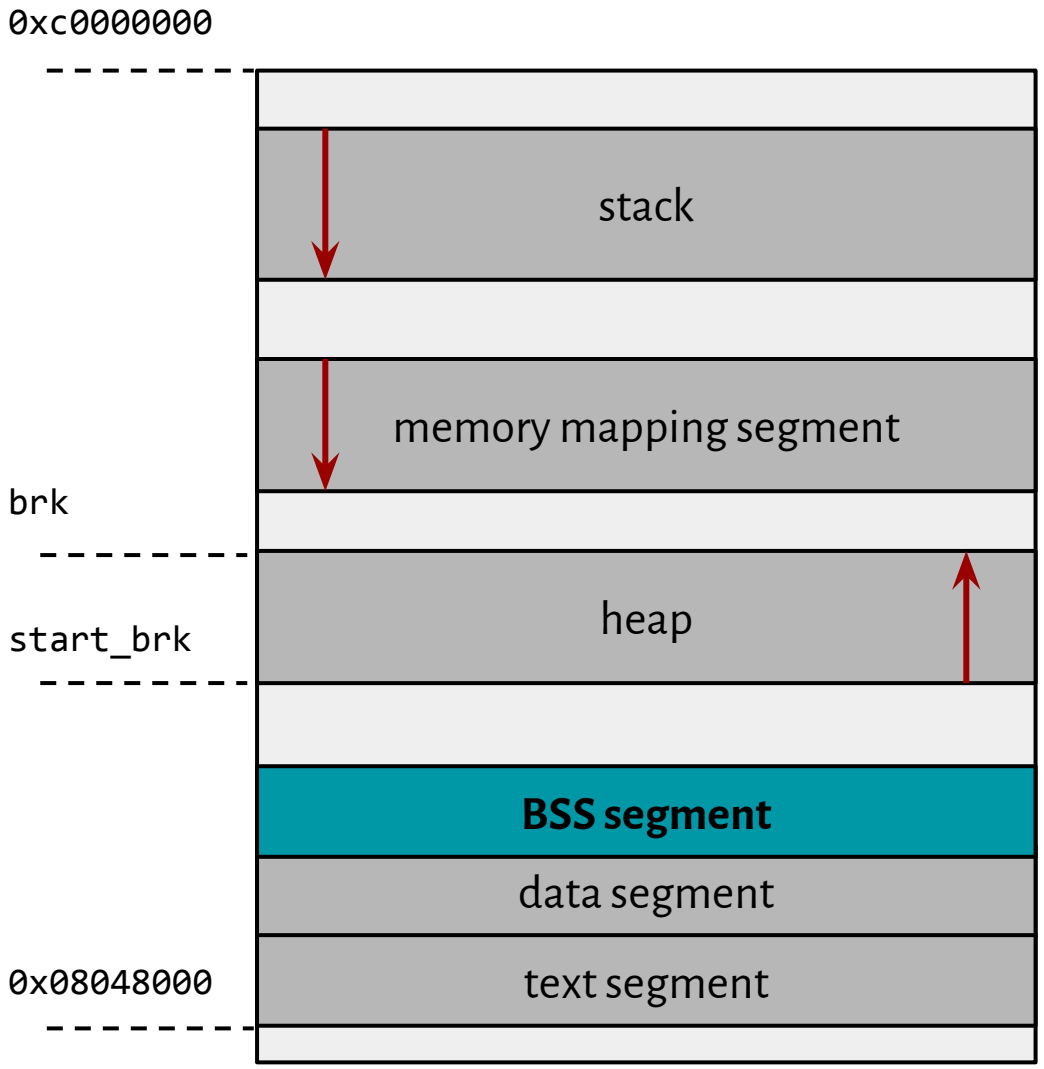



**data segment**

contains any global or static variables which have a predefined value and can be modified

- ★ read-write
- ★ fixed size

# Program Memory Layout

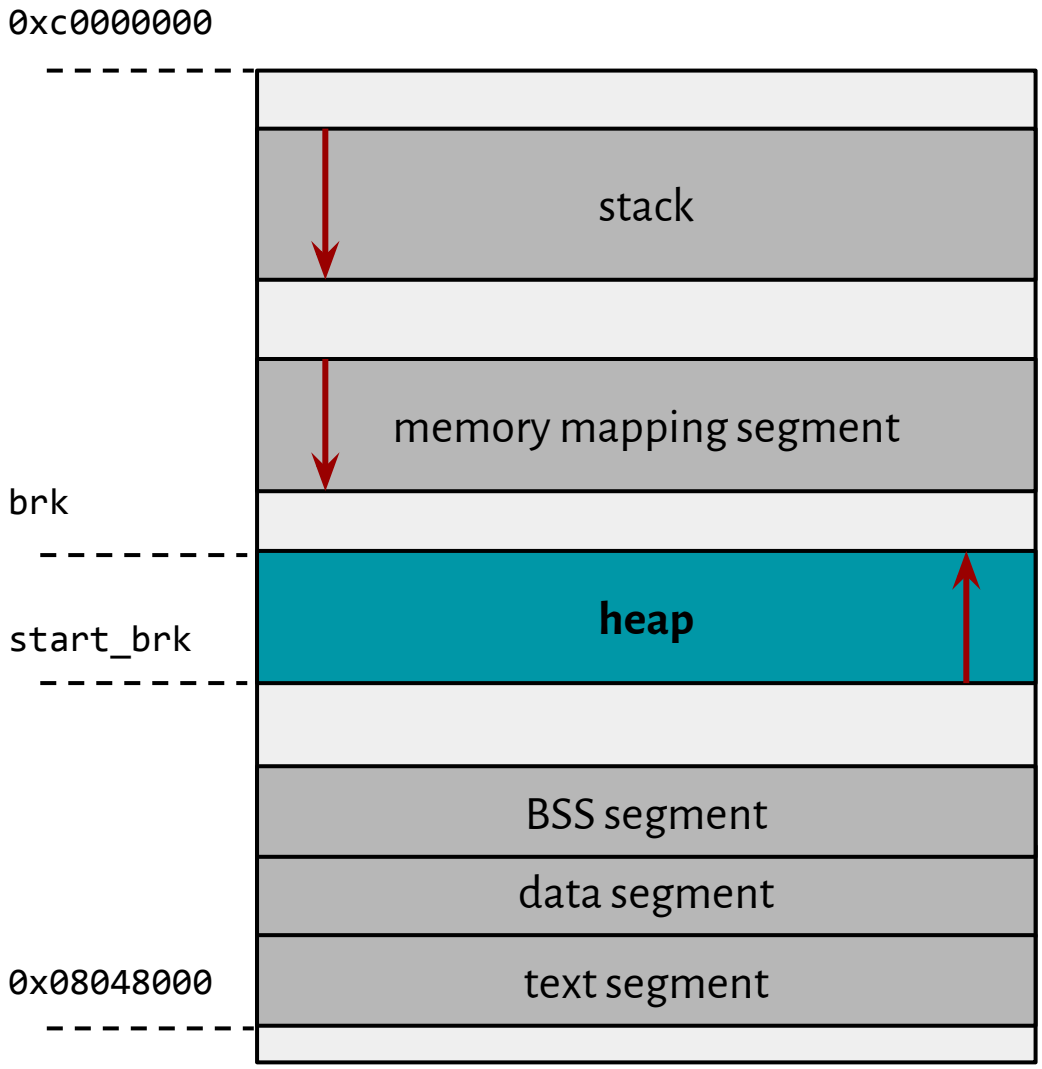



**BSS segment**

memory initialized with zeroes that represent uninitialized static variables

- ★ read-write
- ★ fixed size

# Program Memory Layout

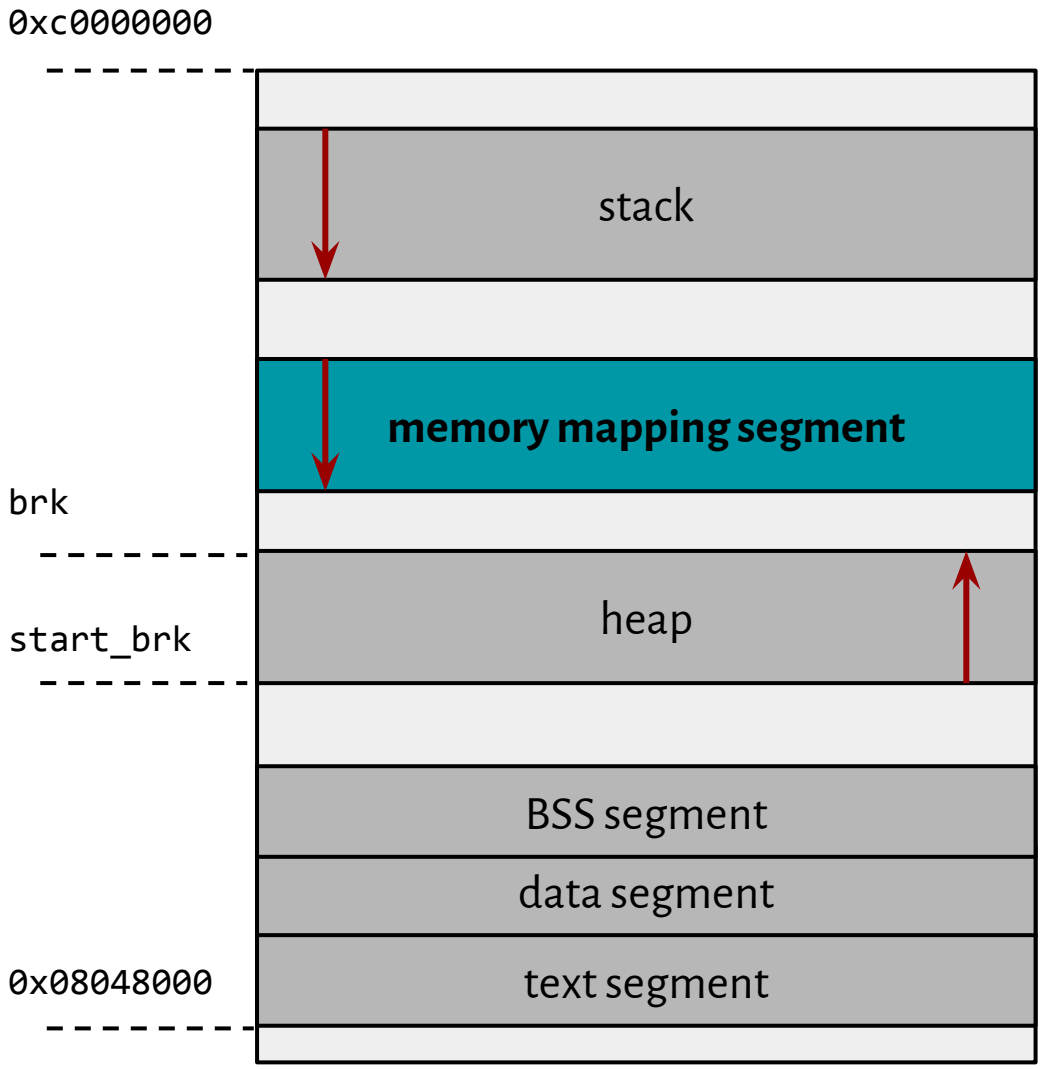



**heap**

contains the dynamically allocated memory

- ★ read-write
- ★ dynamic size

# Program Memory Layout

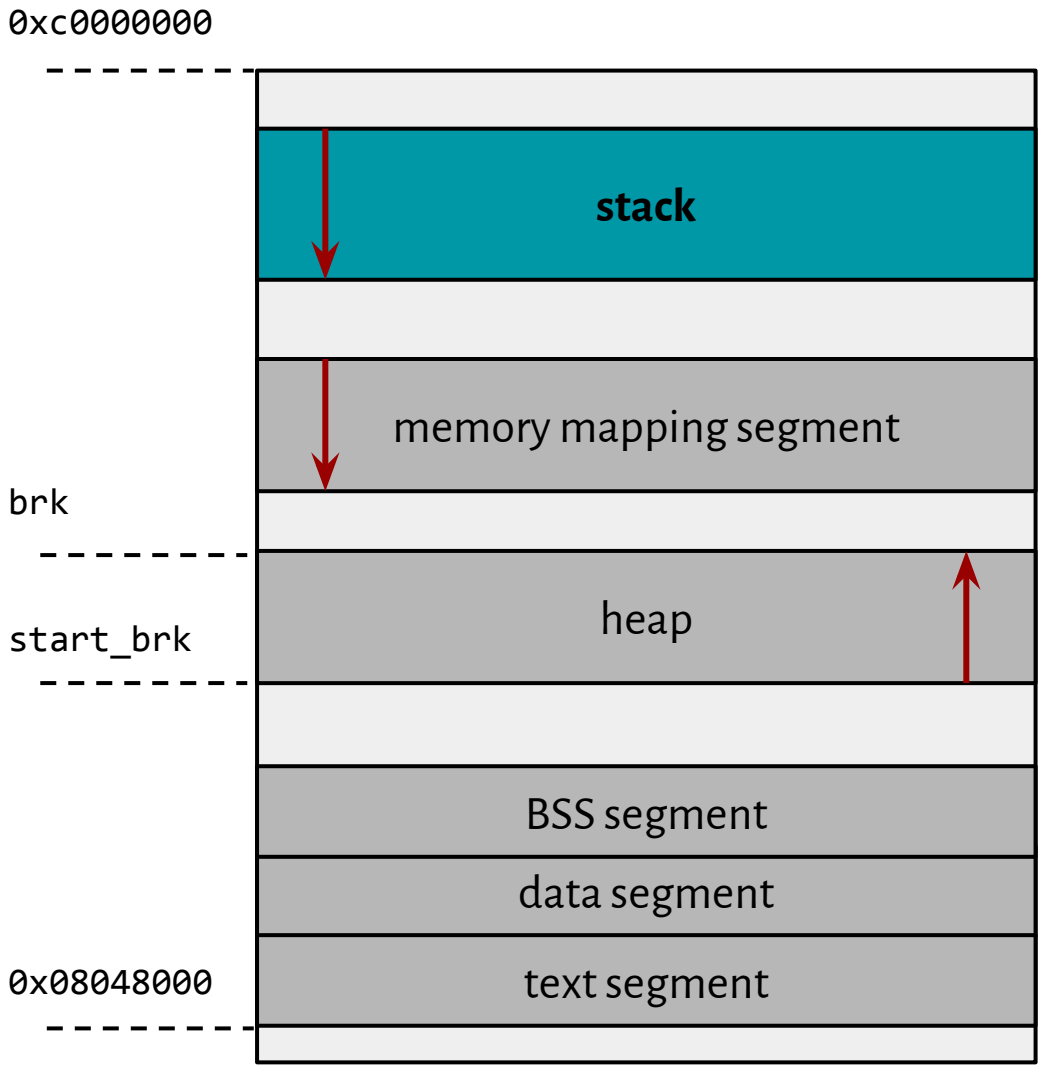



**mmap segment**

a direct byte-for-byte correlation with some portion of a file or file-like resource

- ★ read-write
- ★ dynamic size

# Program Memory Layout

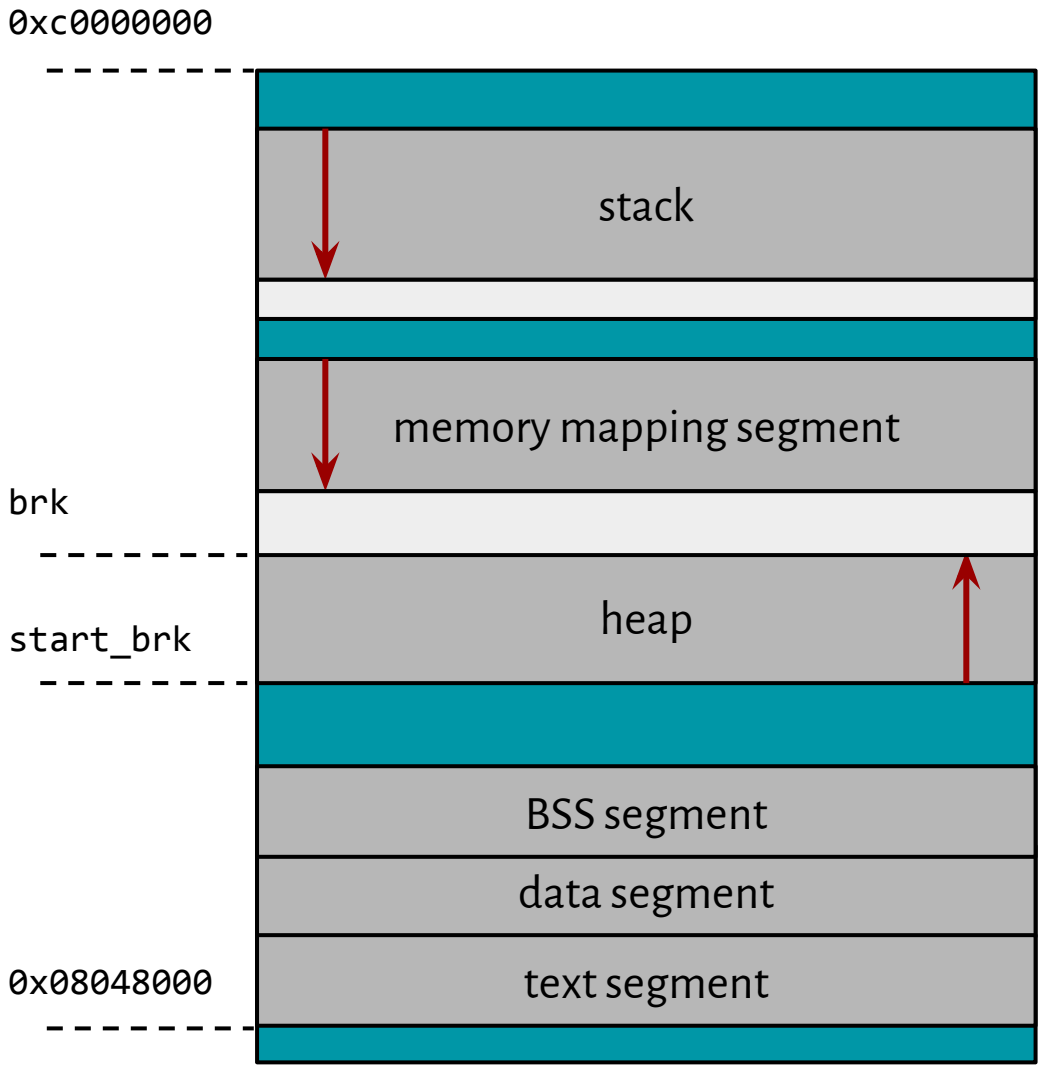



**stack**

stores local variables and function parameters

- ★ read-write
- ★ dynamic size

# Program Memory Layout




**offsets**

present because of safety reasons

★ any access triggers a segfault

★ dynamic size (random)

# Assembly language

★ one-to-one correspondence:

assembly instruction  machine code instruction

# Assembly language

★ one-to-one correspondence:

assembly instruction ⟷ machine code instruction

★ specific to a particular computer architecture

# Assembly language

★ one-to-one correspondence:

assembly instruction ⟷ machine code instruction

★ specific to a particular computer architecture

★ x86 assembly language has two main syntax branches

```
mov $5, %eax
```

AT&T

```
mov eax, 5
```

Intel

# Assembly - basic instructions

```
registers_and_memory:
        mov ebp, esp
        mov rax, qword [mloc]
        mov [mloc], ecx
registers_and_stack:
        pop eax
        push ebx
```

# Assembly - basic instructions

```
arithmetic_operations:
        add eax, ebx
        add rax, qword [mloc]
        or <dest>, <src>
```

https://c9x.me/x86/html/file_module_x86_id_5.html

# Assembly - basic instructions

## Add

| Opcode | Mnemonic | Description |
|---|---|---|
| 04 ib | ADD AL, imm8 | Add imm8 to AL |
| 05 iw | ADD AX, imm16 | Add imm16 to AX |
| 05 id | ADD EAX, imm32 | Add imm32 to EAX |
| 80 /0 ib | ADD r/m8, imm8 | Add imm8 to r/m8 |
| 81 /0 iw | ADD r/m16, imm16 | Add imm16 to r/m16 |
| 81 /0 id | ADD r/m32, imm32 | Add imm32 to r/m32 |
| 83 /0 ib | ADD r/m16, imm8 | Add sign-extended imm8 to r/m16 |
| 83 /0 ib | ADD r/m32, imm8 | Add sign-extended imm8 to r/m32 |
| 00 /r | ADD r/m8, r8 | Add r8 to r/m8 |
| 01 /r | ADD r/m16, r16 | Add r16 to r/m16 |
| 01 /r | ADD r/m32, r32 | Add r32 to r/m32 |
| 02 /r | ADD r8, r/m8 | Add r/m8 to r8 |
| 03 /r | ADD r16, r/m16 | Add r/m16 to r16 |
| 03 /r | ADD r32, r/m32 | Add r/m32 to r32 |

**Description**

Adds the first operand (destination operand) and the second operand (source operand) and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADD instruction performs integer addition. It evaluates the result for both signed and unsigned integer operands and sets the OF and CF flags to indicate a carry (overflow) in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

https://c9x.me/x86/html/file_module_x86_id_5.html

# Assembly - basic instructions

## Add

| Opcode | Mnemonic |
|---|---|
| 04 ib | ADD AL, imm8 |
| 05 iw | ADD AX, imm16 |
| 05 id | ADD EAX, imm32 |
| 80 /0 ib | ADD r/m8, imm8 |
| 81 /0 iw | ADD r/m16, imm16 |
| 81 /0 id | ADD r/m32, imm32 |
| 83 /0 ib | ADD r/m16, imm8 |
| 83 /0 ib | ADD r/m32, imm8 |
| 00 /r | ADD r/m8, r8 |
| 01 /r | ADD r/m16, r16 |
| 01 /r | ADD r/m32, r32 |
| 02 /r | ADD r8, r/m8 |
| 03 /r | ADD r16, r/m16 |
| 03 /r | ADD r32, r/m32 |

| Mnemonic |
|---|
| ADD AL, imm8 |
| ADD AX, imm16 |
| ADD EAX, imm32 |
| ADD r/m8, imm8 |
| ADD r/m16, imm16 |
| ADD r/m32, imm32 |
| ADD r/m16, imm8 |
| ADD r/m32, imm8 |
| ADD r/m8, r8 |
| ADD r/m16, r16 |
| ADD r/m32, r32 |
| ADD r8, r/m8 |
| ADD r16, r/m16 |
| ADD r32, r/m32 |

**Description**

Adds the first operand (destination operand) and the second operand
operand can be a register or a memory location; the source operand ca
cannot be used in one instruction.) When an immediate value is used a

The ADD instruction performs integer addition. It evaluates the result fo
a carry (overflow) in the signed or unsigned result, respectively. The SF

This instruction can be used with a LOCK prefix to allow the instruction

perand. The destination
r, two memory operands
n operand format.

and CF flags to indicate

https://c9x.me/x86/html/file_module_x86_id_5.html

# Assembly - basic instructions

```
arithmetic_operations:
        add eax, ebx
        add rax, qword [mloc]
        or <dest>, <src>
registers_and_stack:
        jmp eax
        cmp rcx, 0
        jne arithmetic_operations
```

# Assembler


```
$ nasm -f elf program.asm
```


```
$ ld -o program program.o
```

# Zadanie 1

Dany jest zbiór *N* zawierający liczby całkowite od **0** do **255**.

# Zadanie 1

Dany jest zbiór *N* zawierający liczby całkowite od **0** do **255**.

Ciąg elementów zbioru *N* jest **poprawny**, jeśli

dla pewnego podzbioru $M \subset N$, $0 \notin M$, dany ciąg jest postaci

(<permutacja *M*>, 0, <permutacja *M*>, 0, ... , <permutacja *M*>, 0)

# Zadanie 1

Dany jest zbiór *N* zawierający liczby całkowite od **0** do **255**.

Ciąg elementów zbioru *N* jest **poprawny**, jeśli

dla pewnego podzbioru $M \subset N$, $0 \notin M$, dany ciąg jest postaci

(<permutacja *M*>, 0, <permutacja *M*>, 0, … , <permutacja *M*>, 0)

Przykład poprawnego ciągu:

(1, 2, 3, 0, 3, 2, 1, 0, 2, 1, 3, 0)

Przykład niepoprawnych ciągów:

(); (1); (1, 1, 0); (1, 2, 0, 1, 0); (1, 2, 0, 1, 3, 0)

# Zadanie 1

**Input:**

argument - ścieżka do pliku, który zawiera ciąg liczb (rozmiaru 1 bajtu)

**Output:**

1 jeśli wystąpił błąd lub ciąg jest niepoprawny, 0 w przeciwnym razie

# Zadanie 1

**Input:**

argument - ścieżka do pliku, który zawiera ciąg liczb (rozmiaru 1 bajtu)

**Output:**

1 jeśli wystąpił błąd lub ciąg jest niepoprawny, 0 w przeciwnym razie

★ Jak sprawdzić argumentów podanych do programu?

★ Jak odpowiednio zakończyć program?

# Zadanie 1

**Input:**

argument - ścieżka do pliku, który zawiera ciąg liczb (rozmiaru 1 bajtu)

**Output:**

1 jeśli wystąpił błąd lub ciąg jest niepoprawny, 0 w przeciwnym razie

★ Jak sprawdzić argumentów podanych do programu?

Ich liczba jest na stosie, a sam argument znajduje się w rejestrze (ABI).

★ Jak odpowiednio zakończyć program?

```
mov eax, 60          ; numer wywołania systemowego exit
mov edi, 0
syscall
```

# Zadanie 1

**Put your solution into the repository before:**

**12 March 2018, 8 p.m.**

**But first, make sure you can access your repository account:**

```
$ svn checkout https://svn.mimuw.edu.pl/repos/SO/studenci/ab123456
$ cd ab123456
$ mkdir zadanie1
$ svn add zadanie1
$ svn commit -m "zalozenie katalogu na rozwiazanie pierwszego zadania"
```