



université  
PARIS-SACLAY



# COMPUTER ARCHITECTURE AND SOFTWARE EXECUTION PROCESS

## MEMORY MANAGEMENT

🎓 Bachelor in Artificial Intelligence, Data and Management Sciences

🏛️ CentraleSupélec and ESSEC Business School - 2024/2025



**Idir AIT SADOUNE** 🌐

[idir.aitsadoune@centralesupelec.fr](mailto:idir.aitsadoune@centralesupelec.fr) ✉️

# OUTLINE

- The main memory
- Memory allocation strategies
- The Virtual memory

[Back to the begin](#) - [Back to the outline](#)

# OUTLINE

- The main memory
- Memory allocation strategies
- The Virtual memory

[Back to the begin](#) - [Back to the outline](#)

# MEMORY FOR WHOM AND WHY?

# MEMORY FOR WHOM AND WHY?

For the operating system

# MEMORY FOR WHOM AND WHY?

## For the operating system

- When a machine is started, the **OS** is the **first program** loaded into memory.

# MEMORY FOR WHOM AND WHY?

## For the operating system

- When a machine is started, the **OS** is the **first program** loaded into memory.
- The **OS** needs a **memory space** for :
  - ➡ the **code** of its **Core**
  - ➡ the **interruptions** table
  - ➡ the **processes** table
  - ➡ data structures (**PCBs** and others)
  - ➡ ...

# MEMORY FOR WHOM AND WHY?

For processes (running programs)



# MEMORY FOR WHOM AND WHY?

## For processes (running programs)

- When a process is created, the OS creates a PCB and allocates memory for the process.

# MEMORY FOR WHOM AND WHY?

## For processes (running programs)

- When a process is created, the OS creates a PCB and allocates memory for the process.
- For security reasons, each process must use a separate memory area (an address space).

# MEMORY FOR WHOM AND WHY?

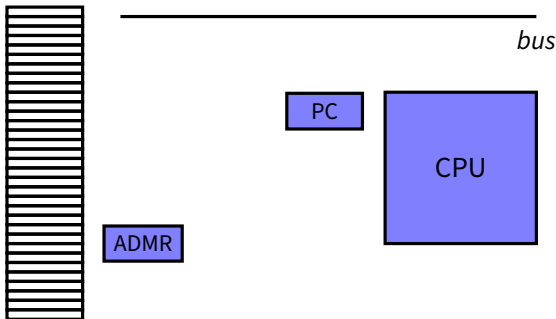
## For processes (running programs)

- When a process is created, the **OS creates a PCB** and **allocates memory** for the process.
- For **security** reasons, each **process** must use **a separate memory area (an address space)**.
  - ➡ which mechanism for **allocating** this space?
  - ➡ how to ensure the **protection** of this area?
  - ➡ how to ensure the **transparency** of this space?

# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

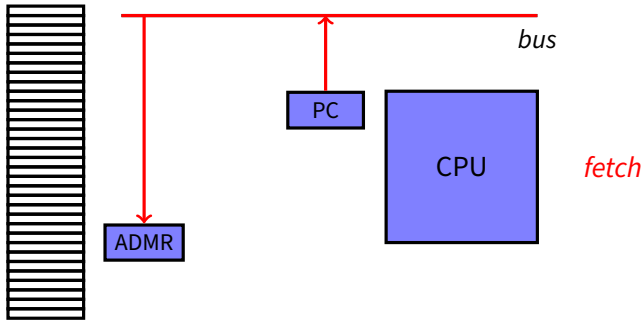
- instructions → **PC** register on the processor
- data → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

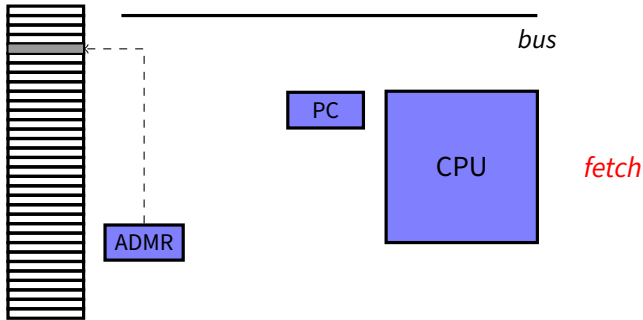
- **instructions** → **PC** register on the processor
- data → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

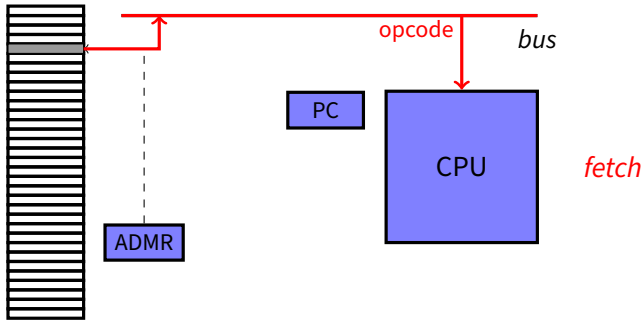
- **instructions** → **PC** register on the processor
- data → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

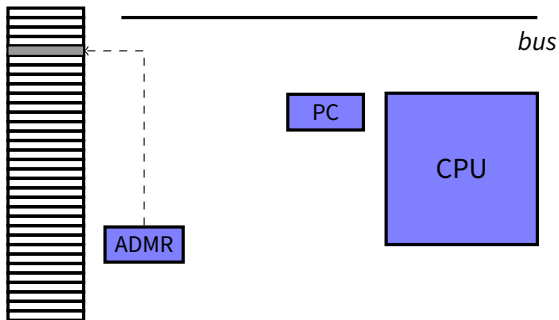
- **instructions** → **PC** register on the processor
- data → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

- instructions → **PC** register on the processor
- data → **ADMR** register on the processor

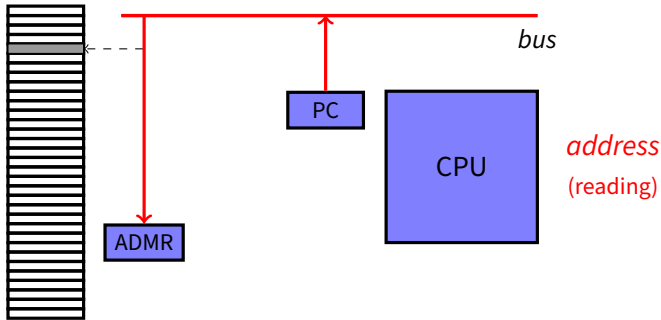




# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

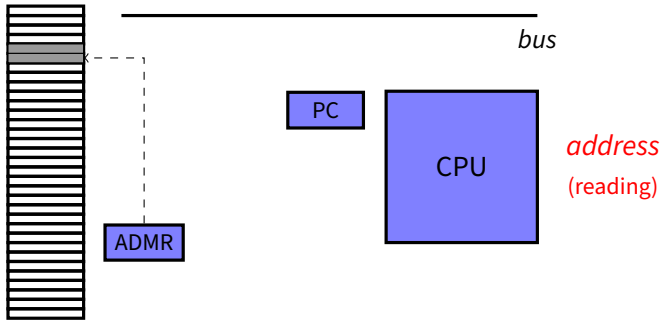
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

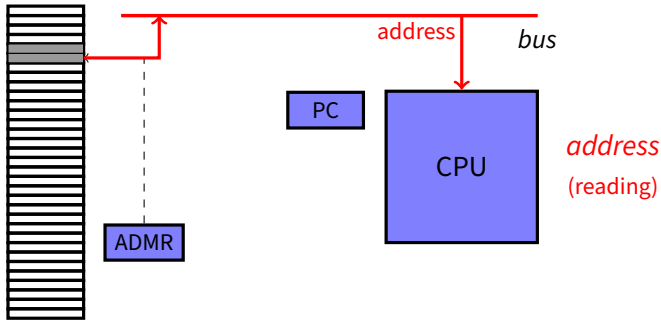
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

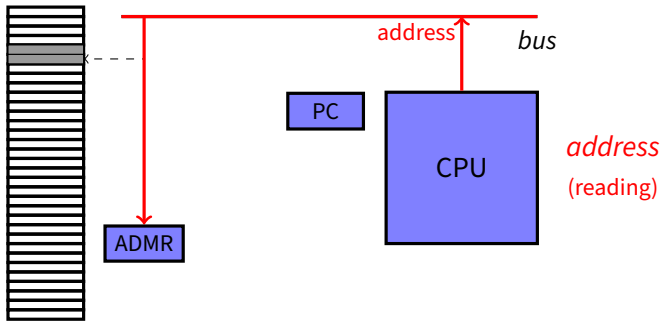
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

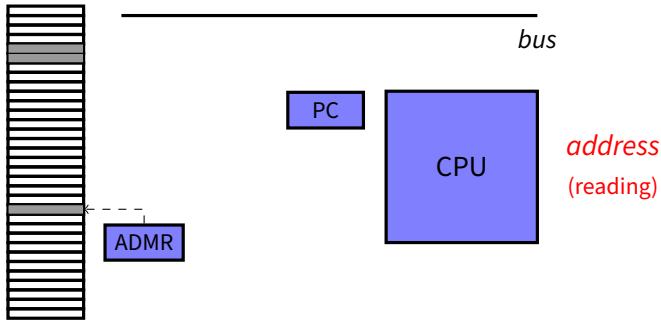
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

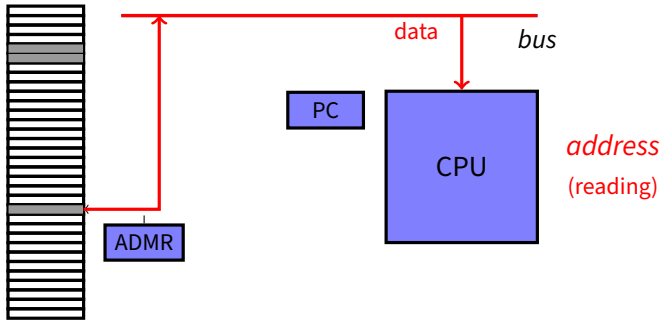
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

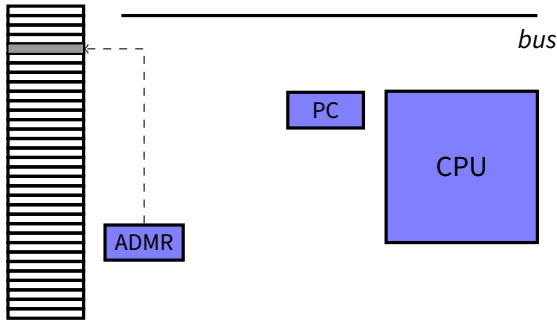
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

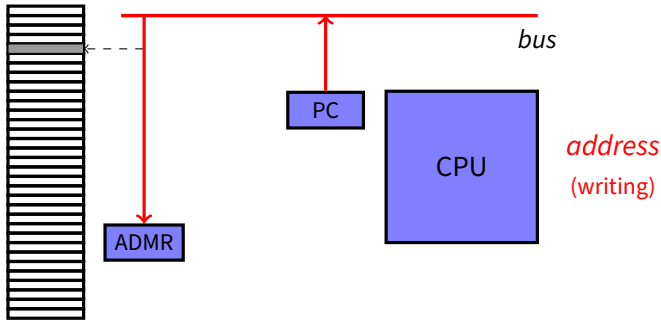
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor

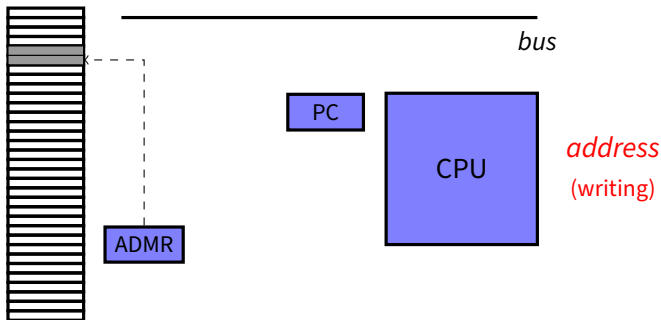




# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

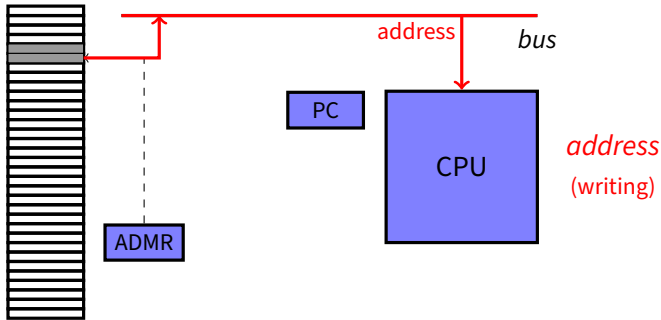
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

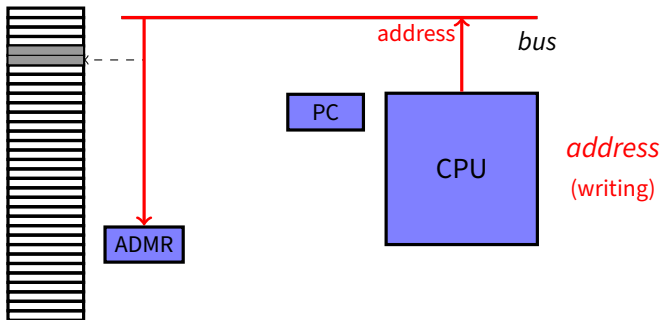
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

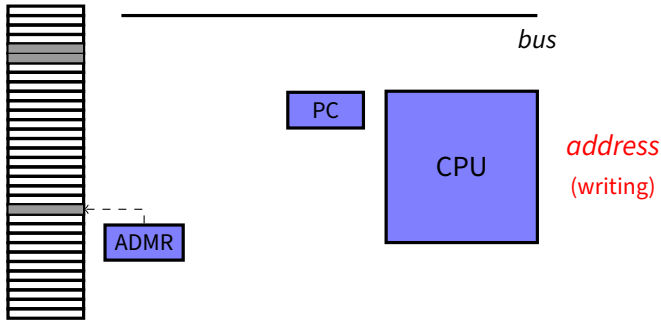
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

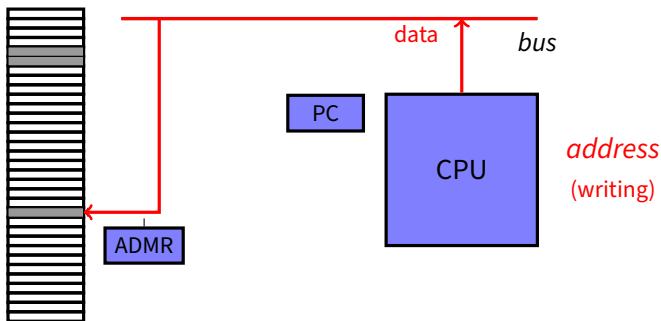
- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# STORAGE SPACE

Ordered set of **memory words** indexed by their **address** (**box number**) and containing :

- instructions → **PC** register on the processor
- **data** → **ADMR** register on the processor



# MEMORY STRUCTURE

# MEMORY STRUCTURE

- Each **memory word** is associated with an **address**
  - ➡ the box number

# MEMORY STRUCTURE

- Each **memory word** is associated with an **address**
  - ➡ the box number
- This **address** is obtained from **an instruction**
  - ➡ the address is in **binary** with  $n$  bits



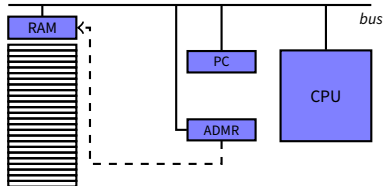
# MEMORY STRUCTURE

- Each **memory word** is associated with an **address**
  - ➡ the box number
- This **address** is obtained from **an instruction**
  - ➡ the address is in **binary** with  $n$  bits
- So there are  $2^n$  **different addresses**
  - ➡  $2^n$  boxes from 0000 ... 0 to 1111 ... 1

# MEMORY STRUCTURE

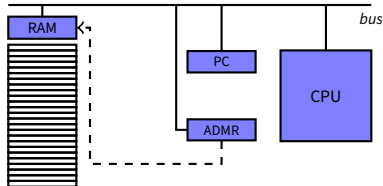
- Each **memory word** is associated with an **address**
  - ➡ the box number
- This **address** is obtained from **an instruction**
  - ➡ the address is in **binary** with  $n$  bits
- So there are  $2^n$  **different addresses**
  - ➡  $2^n$  boxes from 0000 ... 0 to 1111 ... 1
- **Example** : 32 bits  $\rightarrow 2^{32} \approx 4$  GB

# HOW MEMORY WORKS



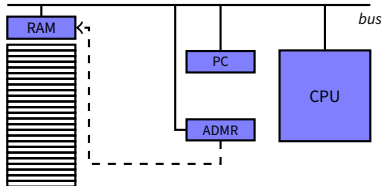
# HOW MEMORY WORKS

- The **CPU** will fetch **instructions** from memory **by using their address** (*fetch*);



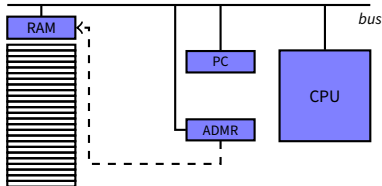
# HOW MEMORY WORKS

- The **CPU** will fetch **instructions** from memory **by using their address** (*fetch*);
- The **CPU** will retrieve **data** from variables in memory **by using their address**;



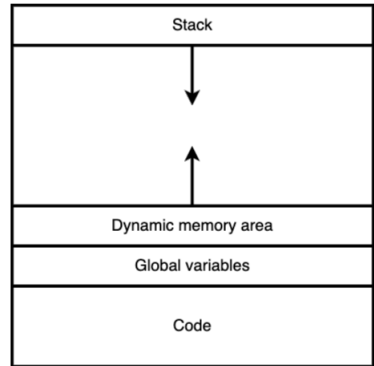
# HOW MEMORY WORKS

- The **CPU** will fetch **instructions** from memory **by using their address** (*fetch*);
- The **CPU** will retrieve **data** from variables in memory **by using their address**;
- The **CPU** writes on **variables at a given address** in memory.



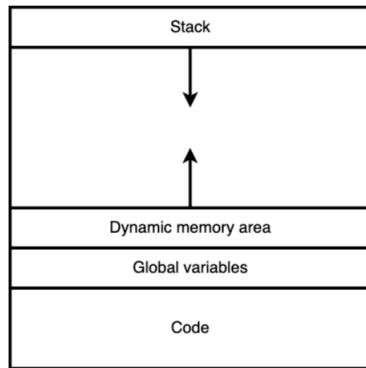
# ADDRESS SPACE USAGE

- What does a process's memory space contain?



# ADDRESS SPACE USAGE

- What does a process's memory space contain?
  - Code (known size)
  - Global variables (known size)
  - Stack (unknown size)
  - Dynamic memory area (unknown size)
  - ...





# WHERE DO THE PROGRAMS COME FROM?

# WHERE DO THE PROGRAMS COME FROM ?

A **program** (code + data) is **loaded** from the **disk** to the **memory** ...



# WHERE DO THE PROGRAMS COME FROM ?

A **program** (code + data) is **loaded** from the **disk** to the **memory** ... it is placed at a given location in the memory

# WHERE DO THE PROGRAMS COME FROM ?

A **program** (code + data) is **loaded** from the **disk** to the **memory** ... it is placed at a given location in the memory

## Question

what are the addresses of the variables in memory?

# PROGRAM VS PROCESS

**symbolic** addresses vs **memory** addresses



# PROGRAM VS PROCESS

**symbolic** addresses vs **memory** addresses

```
1 int a = 3;  
2 a = a + 2;
```



# PROGRAM VS PROCESS

**symbolic** addresses vs **memory** addresses

```
1 int a = 3;  
2 a = a + 2;
```

```
1 @a: memval 3  
2     mov eax, a  
3     mov ebx, 2  
4     add ecx, eax, ebx  
5     mov a, ecx
```

# PROGRAM VS PROCESS

**symbolic** addresses vs **memory** addresses

```
1 int a = 3;  
2 a = a + 2;
```

```
1 @a: memval 3  
2     mov eax, a  
3     mov ebx, 2  
4     add ecx, eax, ebx  
5     mov a, ecx
```

```
1 2B50: mov eax, 2B1E  
2 2B52: mov ebx, #0002  
3 2B54: add ecx, eax, ebx  
4 2B55: mov 2B1E, ecx  
5 ...  
6 2B1E: 0003
```



# PROGRAM VS PROCESS

**symbolic** addresses vs **memory** addresses

```
1 int a = 3;  
2 a = a + 2;
```

```
1 @a: memval 3  
2     mov eax, a  
3     mov ebx, 2  
4     add ecx, eax, ebx  
5     mov a, ecx
```

```
1 2B50: mov eax, 2B1E  
2 2B52: mov ebx, #0002  
3 2B54: add ecx, eax, ebx  
4 2B55: mov 2B1E, ecx  
5 ...  
6 2B1E: 0003
```

## Link editing

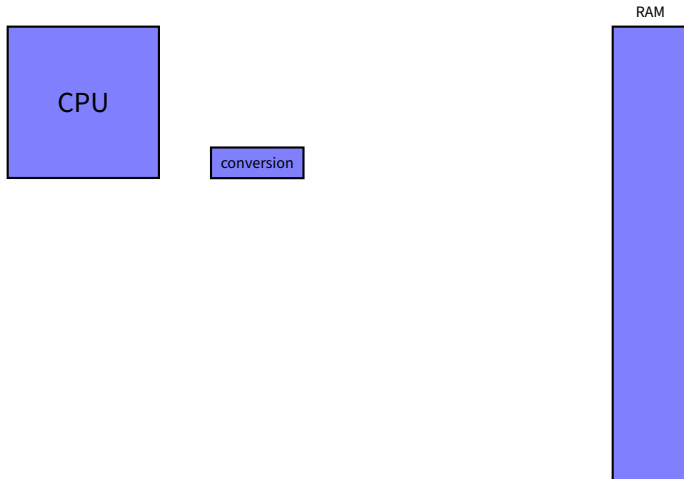
When creating processes, the **OS instantiates** the program.

➡ transform **names** of variables into **addresses**.



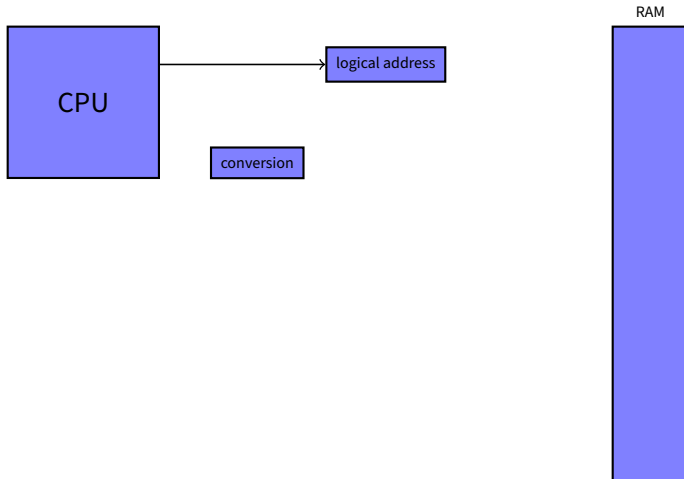
# ADDRESS RESOLUTION

## MEMORY MANAGEMENT UNIT - MMU



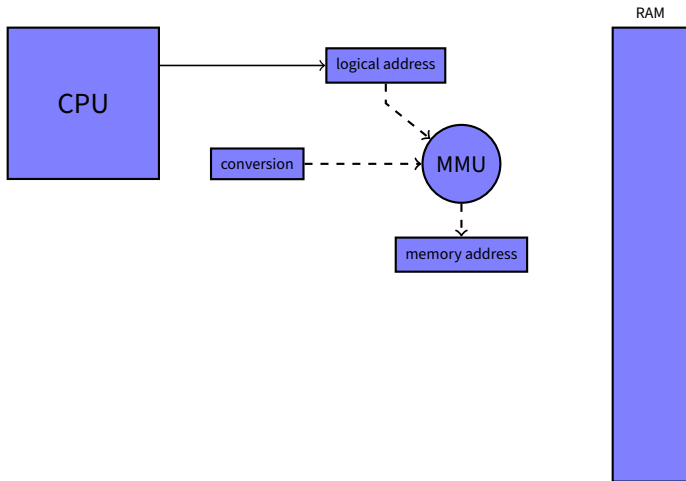
# ADDRESS RESOLUTION

## MEMORY MANAGEMENT UNIT - MMU



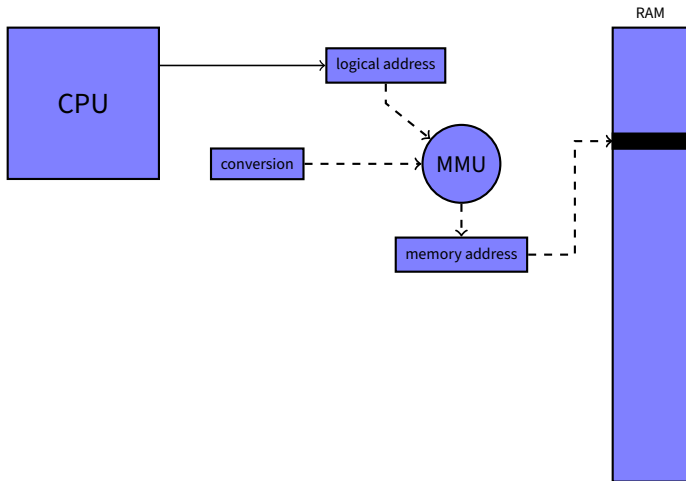
# ADDRESS RESOLUTION

## MEMORY MANAGEMENT UNIT - MMU



# ADDRESS RESOLUTION

## MEMORY MANAGEMENT UNIT - MMU



# OUTLINE

- The main memory
- Memory allocation strategies
- The Virtual memory

[Back to the begin](#) - [Back to the outline](#)

# MEMORY ALLOCATION STRATEGIES

# MEMORY ALLOCATION STRATEGIES

- A strategy for **allocating and freeing memory** must be chosen based on the needs of the processes.



# MEMORY ALLOCATION STRATEGIES

- A strategy for **allocating and freeing memory** must be chosen based on the needs of the processes.
- Two possible strategies :

# MEMORY ALLOCATION STRATEGIES

- A strategy for **allocating and freeing memory** must be chosen based on the needs of the processes.
- Two possible strategies :
  1. **Contiguous allocation** of memory slots (**by partition**)

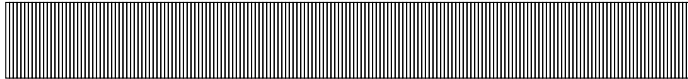
# MEMORY ALLOCATION STRATEGIES

- A strategy for **allocating and freeing memory** must be chosen based on the needs of the processes.
- Two possible strategies :
  1. **Contiguous allocation** of memory slots (**by partition**)
  2. **Non-contiguous allocation** (**by pagination**)

# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

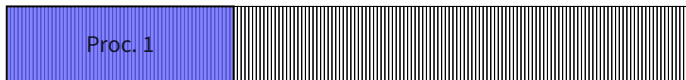
RAM



# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

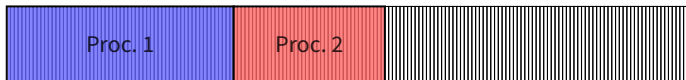
RAM



# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

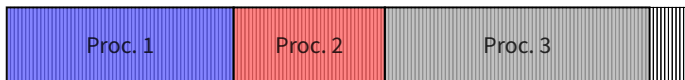
RAM



# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

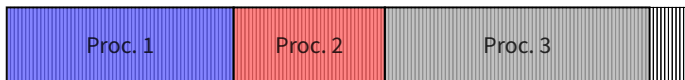
RAM



# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

RAM

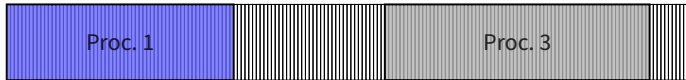




# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

RAM

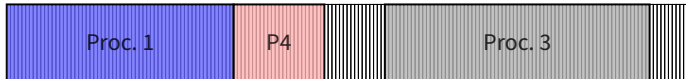


✗ holes appear → **fragmentation**

# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

RAM

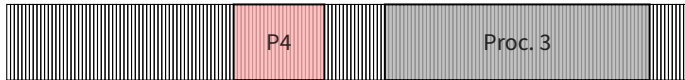


✗ holes appear → **fragmentation**

# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

RAM

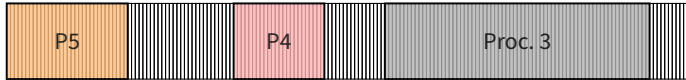


✗ holes appear → **fragmentation**

# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

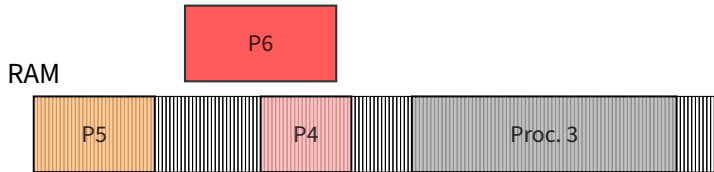
RAM



✗ holes appear → fragmentation

# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

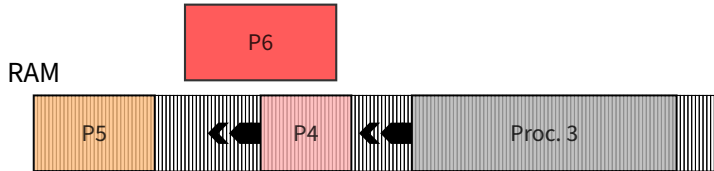


✗ holes appear → **fragmentation**

✗ bigs processes cannot fit in

# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a single, non-decomposable block.



- ✗ holes appear → fragmentation
- ✗ bigs processes cannot fit in → defragment

# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.



- ✗ holes appear → **fragmentation**
- ✗ big processes cannot fit in → **defragment**

# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

RAM



- ✗ holes appear → **fragmentation**
- ✗ bigs processes cannot fit in → **defragment**



# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

RAM



- ✗ holes appear → **fragmentation**
- ✗ big processes cannot fit in → **defragment**
- ➡ **reduce fragmentation**

# MULTIPROGRAMMING WITH PARTITIONS

Process constitutes a **single, non-decomposable block**.

RAM



- ✗ holes appear → **fragmentation**
- ✗ big processes cannot fit in → **defragment**
- ➡ reduce **fragmentation**
- ➡ reduce **defragmentation** operations

# MULTIPROGRAMMING WITH PARTITIONS



# MULTIPROGRAMMING WITH PARTITIONS

- **Pros**
  - ✓ Material simplicity
  - ✓ Transparency for programs
  - ✓ Checking the validity of addresses

# MULTIPROGRAMMING WITH PARTITIONS

- **Pros**

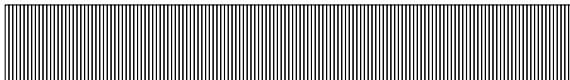
- ✓ Material simplicity
- ✓ Transparency for programs
- ✓ Checking the validity of addresses

- **Cons**

- ✗ Fragmentation
- ✗ Fixed size of memory spaces

# MULTIPROGRAMMING WITH PAGING

RAM



Process :

code, libraries  
environment, heap, stack...

# MULTIPROGRAMMING WITH PAGING

- Divide the **physical memory** (Main memory) into **blocks** (**frames**).

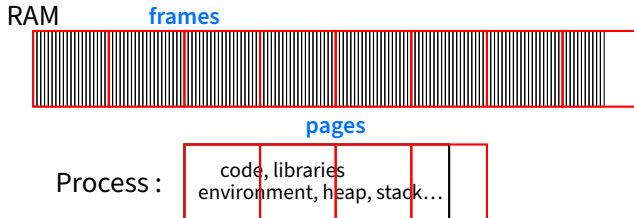


Process :

code, libraries  
environment, heap, stack...

# MULTIPROGRAMMING WITH PAGING

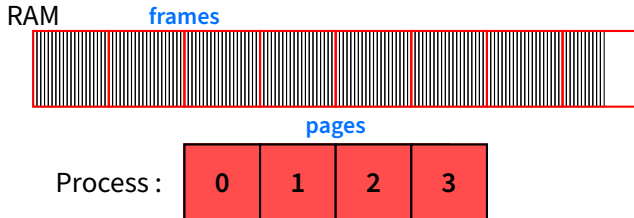
- Divide the **physical memory** (Main memory) into **blocks** (**frames**).
- Divide the **address space** used by a process (**logical space**) into **pages**.





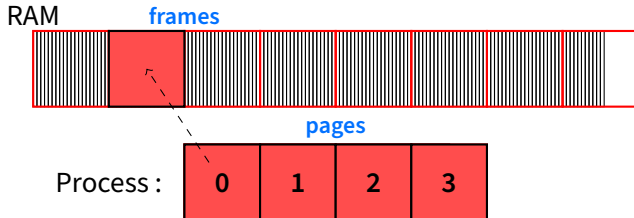
# MULTIPROGRAMMING WITH PAGING

- Divide the **physical memory** (Main memory) into **blocks** (**frames**).
- Divide the **address space** used by a process (**logical space**) into **pages**.
  - each **page** has the **same size** as a **frame**



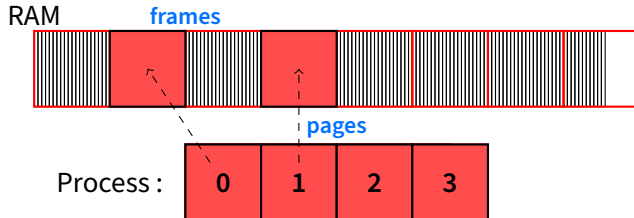
# MULTIPROGRAMMING WITH PAGING

- Divide the **physical memory** (Main memory) into **blocks (frames)**.
- Divide the **address space** used by a process (**logical space**) into **pages**.
  - each **page** has the **same size** as a **frame**
- Place the pages in the frames



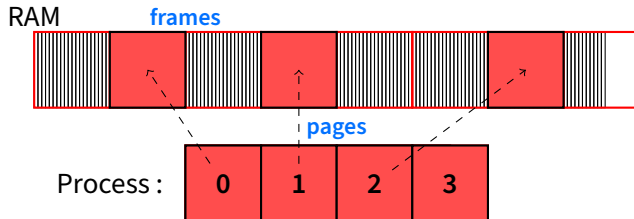
# MULTIPROGRAMMING WITH PAGING

- Divide the **physical memory** (Main memory) into **blocks (frames)**.
- Divide the **address space** used by a process (**logical space**) into **pages**.
  - each **page** has the **same size** as a **frame**
- Place the pages in the frames



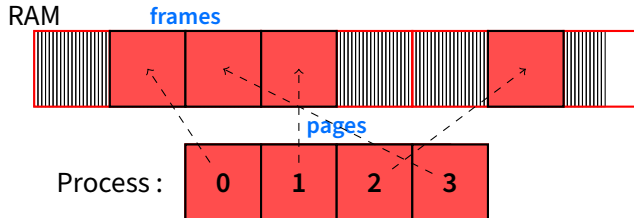
# MULTIPROGRAMMING WITH PAGING

- Divide the **physical memory** (Main memory) into **blocks (frames)**.
- Divide the **address space** used by a process (**logical space**) into **pages**.
  - each **page** has the **same size** as a **frame**
- Place the pages in the frames



# MULTIPROGRAMMING WITH PAGING

- Divide the **physical memory** (Main memory) into **blocks (frames)**.
- Divide the **address space** used by a process (**logical space**) into **pages**.
  - each **page** has the **same size** as a **frame**
- Place the pages in the frames



# MULTIPROGRAMMING WITH PAGING

- Memory allocation :
  - ➡ a process is in *disjoint* areas
  - ➡ no need to defragment

# MULTIPROGRAMMING WITH PAGING

- **Memory allocation :**
  - ➡ a process is in *disjoint* areas
  - ➡ no need to defragment
- **Adaptation :**
  - ➡ need more memory → *add* pages
  - ➡ no need to re-allocate it entirely

# MULTIPROGRAMMING WITH PAGING

- **Memory allocation :**
  - ➡ a process is in *disjoint* areas
  - ➡ no need to defragment
- **Adaptation :**
  - ➡ need more memory → *add* pages
  - ➡ no need to re-allocate it entirely
- **Virtual memory :** load only the pages that the process needs.



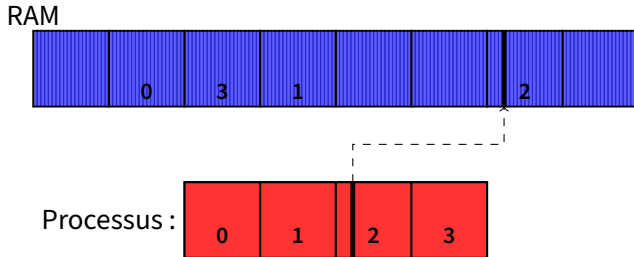
# MULTIPROGRAMMING WITH PAGING ADDRESSING

# MULTIPROGRAMMING WITH PAGING ADDRESSING

- Determine the **physical** address from the **logical** address

# MULTIPROGRAMMING WITH PAGING ADDRESSING

- Determine the **physical** address from the **logical** address



# MULTIPROGRAMMING WITH PAGING ADDRESSING

- Determine the **physical** address from the **logical** address
- **Logical address**
  - Page number ( $n$  bits) + offset ( $m$  bits)

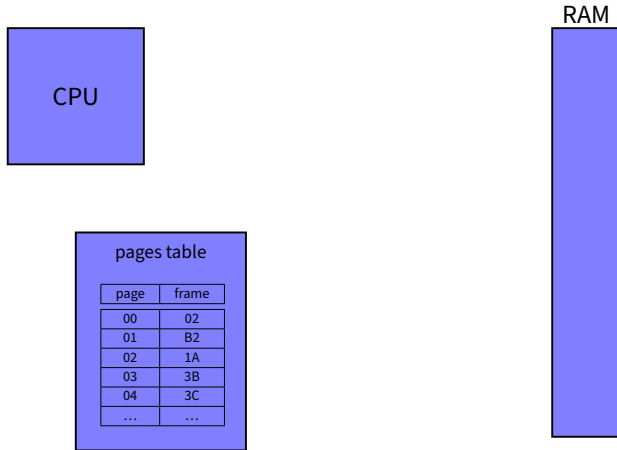
# MULTIPROGRAMMING WITH PAGING ADDRESSING

- Determine the **physical** address from the **logical** address
- **Logical address**
  - Page number ( $n$  bits) + offset ( $m$  bits)
- Each process maintains a list :
  - page number  $\rightarrow$  frame number

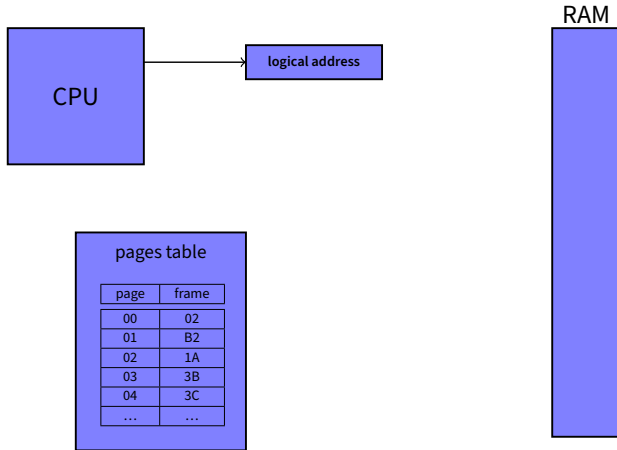
# MULTIPROGRAMMING WITH PAGING ADDRESSING

- Determine the **physical** address from the **logical** address
- **Logical address**
  - Page number ( $n$  bits) + offset ( $m$  bits)
- Each process maintains a list :
  - page number  $\rightarrow$  frame number
  - this is the **pages table**

# MULTIPROGRAMMING WITH PAGING

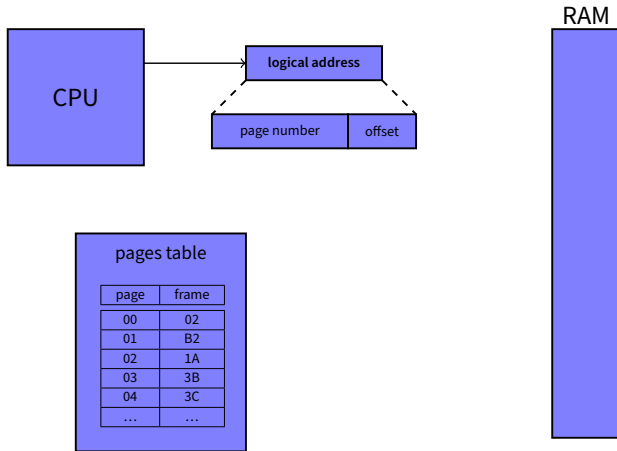


# MULTIPROGRAMMING WITH PAGING

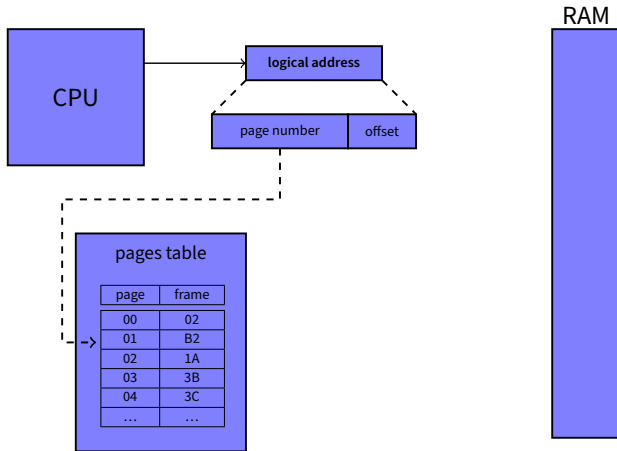




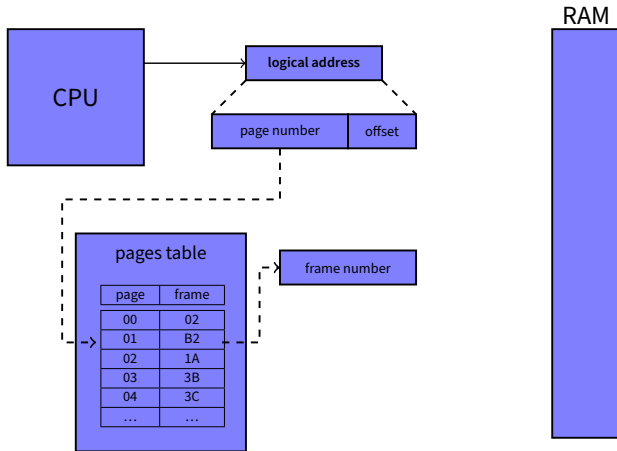
# MULTIPROGRAMMING WITH PAGING



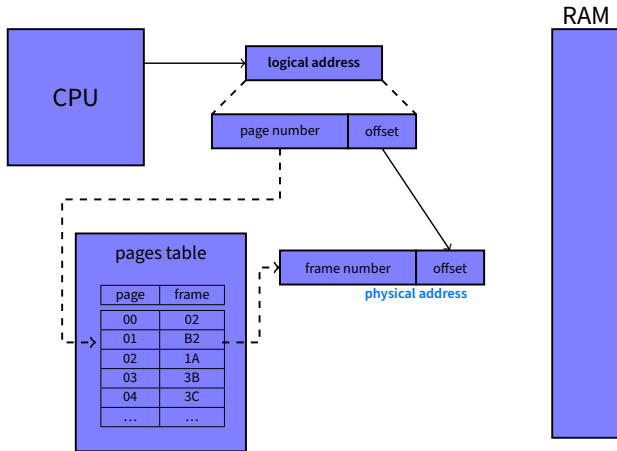
# MULTIPROGRAMMING WITH PAGING



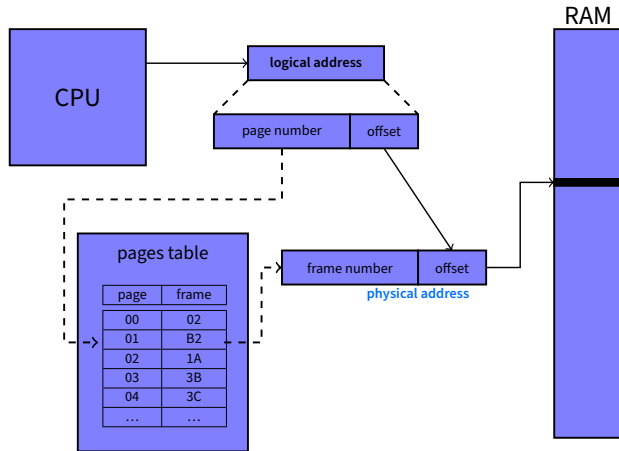
# MULTIPROGRAMMING WITH PAGING



# MULTIPROGRAMMING WITH PAGING



# MULTIPROGRAMMING WITH PAGING



# OUTLINE

- The main memory
- Memory allocation strategies
- The Virtual memory

[Back to the begin](#) - [Back to the outline](#)

# MEMORY STATE

# MEMORY STATE

- Number and size of processes :



# MEMORY STATE

- Number and size of processes :
  - between 200 and 500 parallel processes on a CPU

# MEMORY STATE

- Number and size of processes :
  - between 200 and 500 parallel processes on a CPU
  - big process : Eclipse = 250 MB, game data  $\geq$  1 GB

# MEMORY STATE

- Number and size of processes :
  - between 200 and 500 parallel processes on a CPU
  - big process : Eclipse = 250 MB, game data  $\geq 1$  GB
  - ✗ sum of process sizes  $\geq$  RAM capacity

# MEMORY STATE

- Number and size of processes :
  - between 200 and 500 parallel processes on a CPU
  - big process : Eclipse = 250 MB, game data  $\geq 1$  GB
  - ✗ sum of process sizes  $\geq$  RAM capacity
- Unused parts of code

# MEMORY STATE

- Number and size of processes :
  - between 200 and 500 parallel processes on a CPU
  - big process : Eclipse = 250 MB, game data  $\geq 1$  GB
  - ✗ sum of process sizes  $\geq$  RAM capacity
- Unused parts of code
  - error handling  $\rightarrow$  rarely used

# MEMORY STATE

- Number and size of processes :
  - between 200 and 500 parallel processes on a CPU
  - big process : Eclipse = 250 MB, game data  $\geq 1$  GB
  - ✗ sum of process sizes  $\geq$  RAM capacity
- Unused parts of code
  - error handling  $\rightarrow$  rarely used
  - data (table, game, ...)  $\rightarrow$  not all at once

# MEMORY STATE

- Number and size of processes :
  - between 200 and 500 parallel processes on a CPU
  - big process : Eclipse = 250 MB, game data  $\geq 1$  GB
  - ✗ sum of process sizes  $\geq$  RAM capacity
- Unused parts of code
  - error handling  $\rightarrow$  rarely used
  - data (table, game, ...)  $\rightarrow$  not all at once
  - library  $\rightarrow$  not all at once

# MEMORY STATE

- Number and size of processes :
  - between 200 and 500 parallel processes on a CPU
  - big process : Eclipse = 250 MB, game data  $\geq 1$  GB
  - ✗ sum of process sizes  $\geq$  RAM capacity
- Unused parts of code
  - error handling  $\rightarrow$  rarely used
  - data (table, game, ...)  $\rightarrow$  not all at once
  - library  $\rightarrow$  not all at once

▶ load only useful pages!



# MEMORY STATE

- Number and size of processes :
  - between 200 and 500 parallel processes on a CPU
  - big process : Eclipse = 250 MB, game data  $\geq 1$  GB
  - ✗ sum of process sizes  $\geq$  RAM capacity
- Unused parts of code
  - error handling  $\rightarrow$  rarely used
  - data (table, game, ...)  $\rightarrow$  not all at once
  - library  $\rightarrow$  not all at once

➡ load only useful pages!

➡ leave the rest on disk

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

- *Extension* of **pagination** mechanisms

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

- *Extension* of **pagination** mechanisms
  - a process organized into pages can represent **spaces not present in RAM** (eg disk) → **virtual memory**

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

- *Extension* of **pagination** mechanisms
  - a process organized into pages can represent **spaces not present in RAM** (eg disk) → **virtual memory**
  - pages are either in **RAM** or in **auxiliary memory** (**swap**) → RAM is a cache

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

- *Extension* of **pagination** mechanisms
  - a process organized into pages can represent **spaces not present in RAM** (eg disk) → **virtual memory**
  - pages are either in **RAM** or in **auxiliary memory** (**swap**) → RAM is a cache
- Each row of **the page table** contains :

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

- *Extension* of **pagination** mechanisms
  - a process organized into pages can represent **spaces not present in RAM** (eg disk) → **virtual memory**
  - pages are either in **RAM** or in **auxiliary memory** (**swap**) → RAM is a cache
- Each row of **the page table** contains :
  - a **validity bit** that indicates whether the page is in RAM

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

- *Extension* of **pagination** mechanisms
  - a process organized into pages can represent **spaces not present in RAM** (eg disk) → **virtual memory**
  - pages are either in **RAM** or in **auxiliary memory** (**swap**) → RAM is a cache
- Each row of **the page table** contains :
  - a **validity bit** that indicates whether the page is in RAM
  - the corresponding address in RAM (**frame number**)

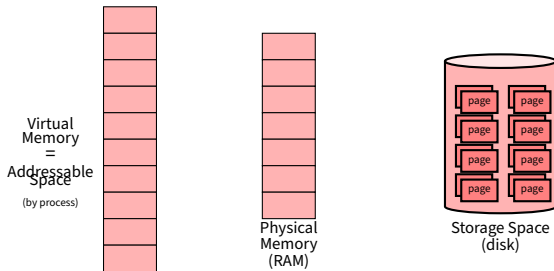


# MULTIPROGRAMMING WITH VIRTUAL MEMORY

- *Extension* of **pagination** mechanisms
  - a process organized into pages can represent **spaces not present in RAM** (eg disk) → **virtual memory**
  - pages are either in **RAM** or in **auxiliary memory** (**swap**) → RAM is a cache
- Each row of **the page table** contains :
  - a **validity bit** that indicates whether the page is in RAM
  - the corresponding address in RAM (**frame number**)
  - otherwise information to find it on **disk**

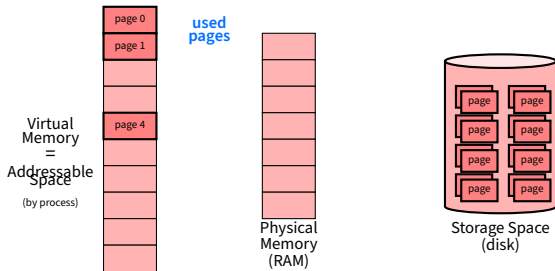
# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**



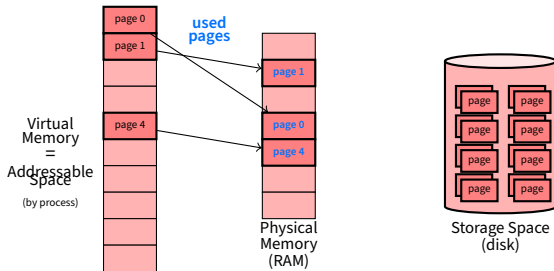
# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**



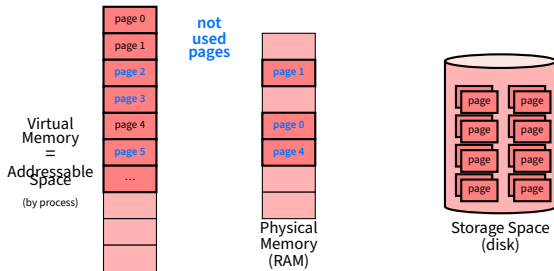
# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**



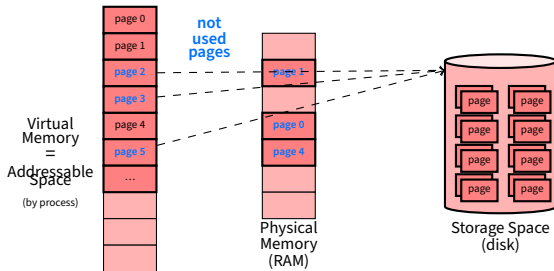
# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**



# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**



# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**

- Access to a page not present in RAM :



# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**

- Access to a page not present in RAM :
  - ➡ raising a CPU **exception** → **Page-Default**

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**

- Access to a page not present in RAM :
  - ➡ raising a CPU **exception** → **Page-Default**
  - ➡ *current process blocked* and **loading** the page into RAM

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**

- Access to a page not present in RAM :
  - ➡ raising a CPU **exception** → **Page-Default**
  - ➡ *current process blocked* and **loading** the page into RAM
- **Benefits**

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**

- Access to a page not present in RAM :
  - ➡ raising a CPU **exception** → **Page-Default**
  - ➡ *current process blocked* and **loading** the page into RAM
- **Benefits**
  - ✓ hide RAM size

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**

- Access to a page not present in RAM :
  - ➡ raising a CPU **exception** → **Page-Default**
  - ➡ *current process blocked* and **loading** the page into RAM
- **Benefits**
  - ✓ hide RAM size
  - ✓ possibility to put more processes in parallel

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**

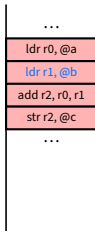
- Access to a page not present in RAM :
  - ➡ raising a CPU **exception** → **Page-Default**
  - ➡ *current process blocked* and **loading** the page into RAM
- **Benefits**
  - ✓ hide RAM size
  - ✓ possibility to put more processes in parallel
  - ✓ assign multiple virtual addresses to a physical address

# MULTIPROGRAMMING WITH VIRTUAL MEMORY

Each process can **address** more space than it actually has in **physical memory**

- Access to a page not present in RAM :
  - ➡ raising a CPU **exception** → **Page-Default**
  - ➡ *current process blocked* and **loading** the page into RAM
- **Benefits**
  - ✓ hide RAM size
  - ✓ possibility to put more processes in parallel
  - ✓ assign multiple virtual addresses to a physical address
  - ✓ pagination on demand

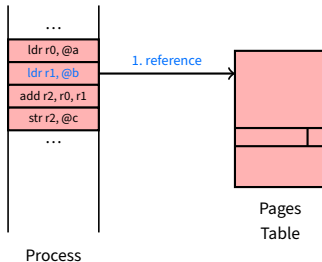
# PAGINATION ON DEMAND



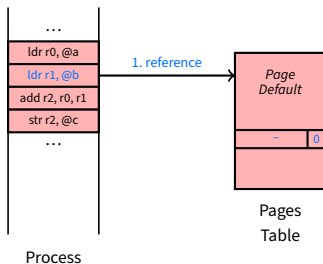
Process



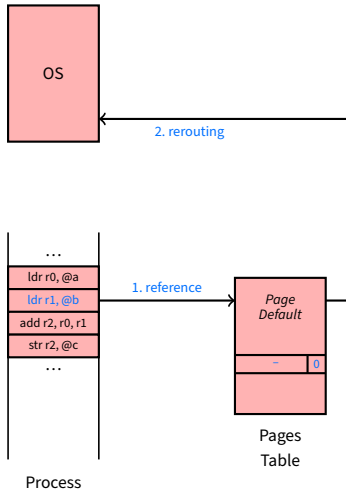
# PAGINATION ON DEMAND



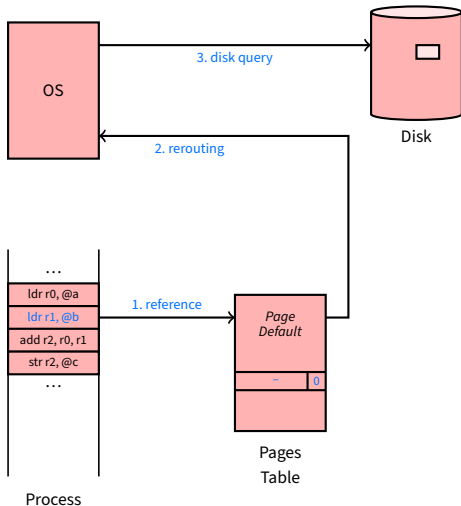
# PAGINATION ON DEMAND



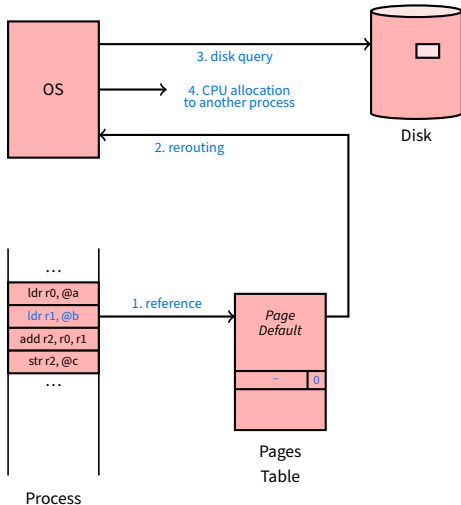
# PAGINATION ON DEMAND



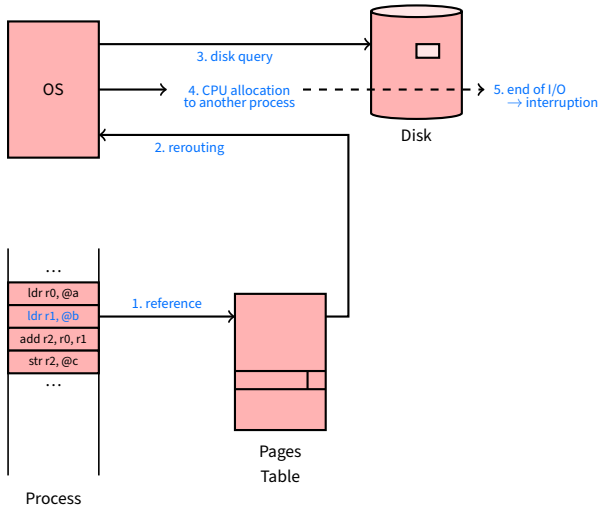
# PAGINATION ON DEMAND



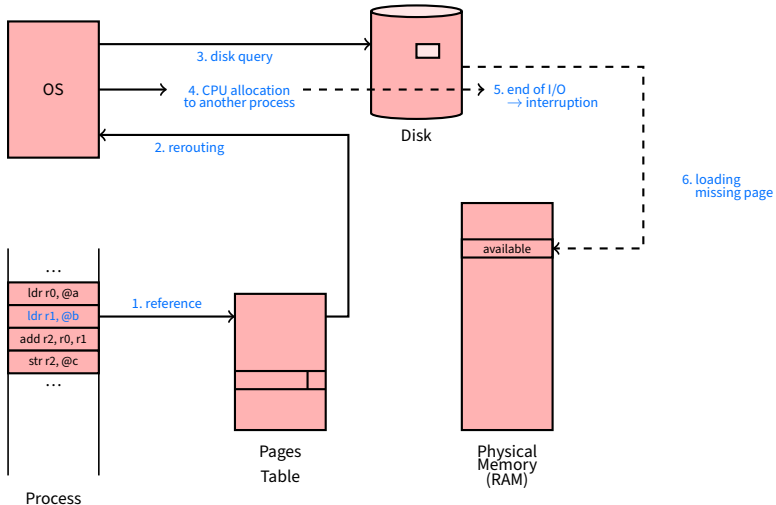
# PAGINATION ON DEMAND



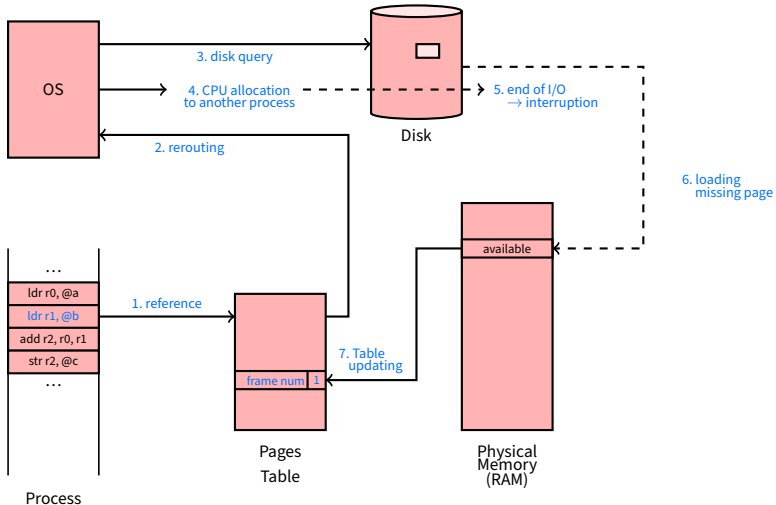
# PAGINATION ON DEMAND



# PAGINATION ON DEMAND

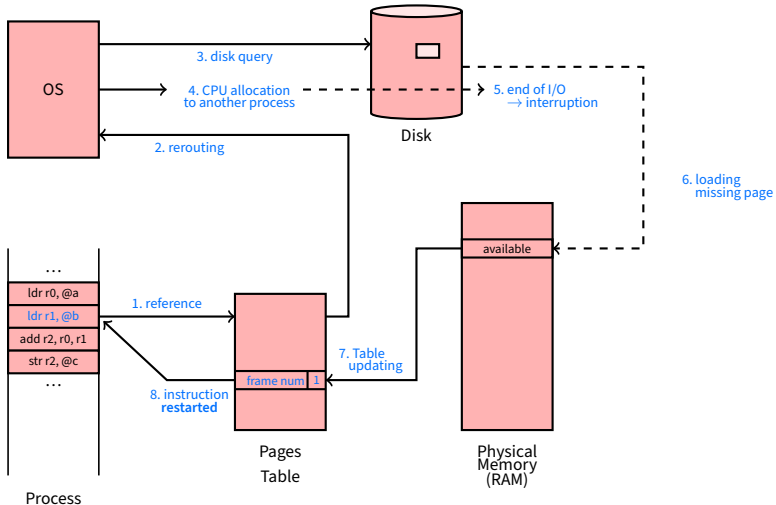


# PAGINATION ON DEMAND





# PAGINATION ON DEMAND



# THANK YOU

[Back to the begin](#) - [Back to the outline](#)