ARCOS Group

uc3m | Universidad Carlos III de Madrid

Lesson 5 (III)
Memory hierarchy

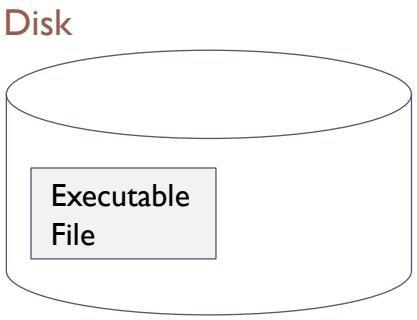
Computer Structure

Bachelor in Computer Science and Engineering

Contents

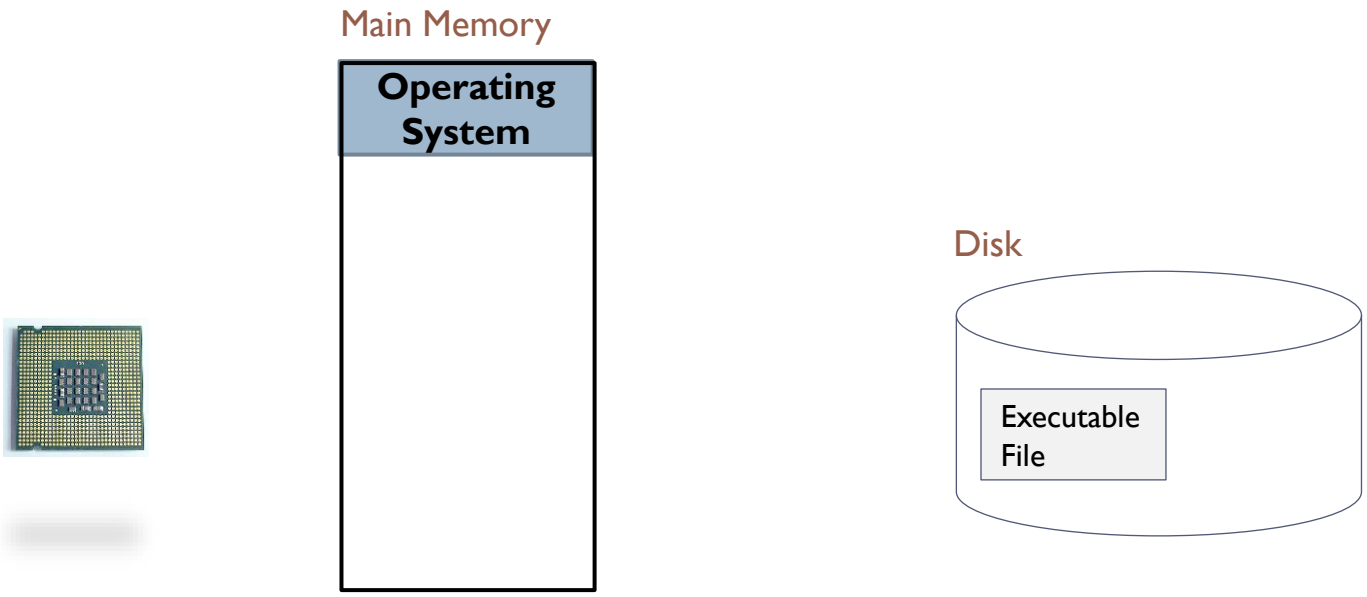
Program and process

- Program: A set of ordered data and instructions



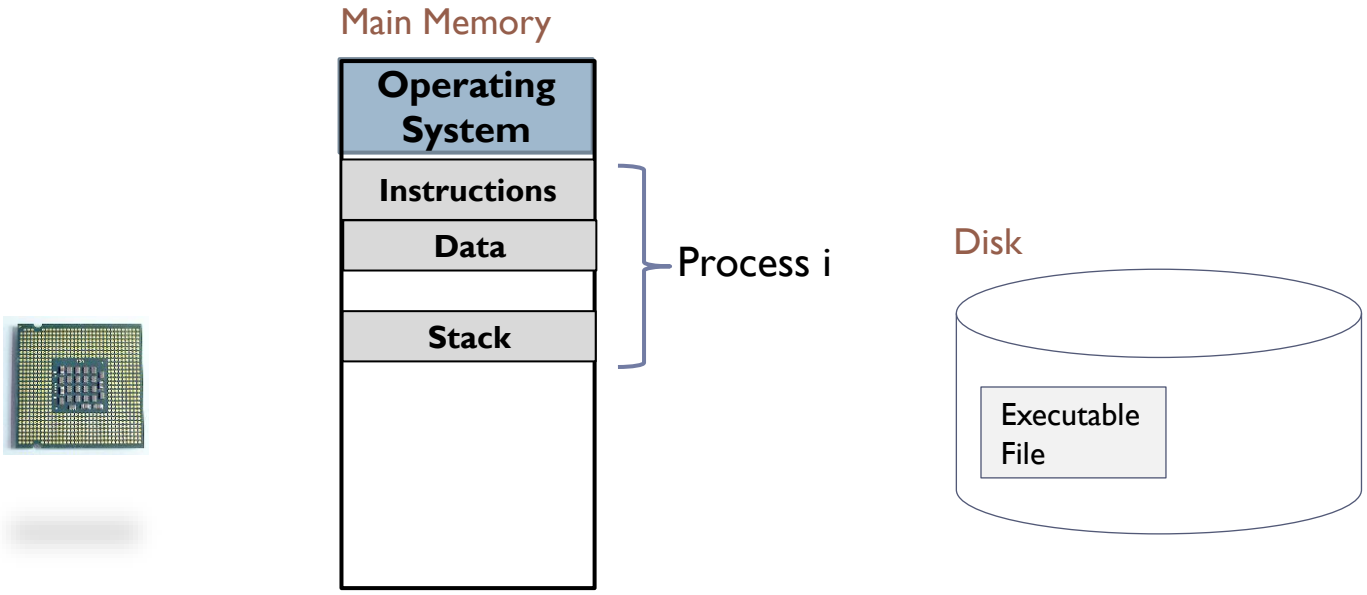
Program and process

- Program: A set of ordered data and instructions
  - Must be loaded in memory



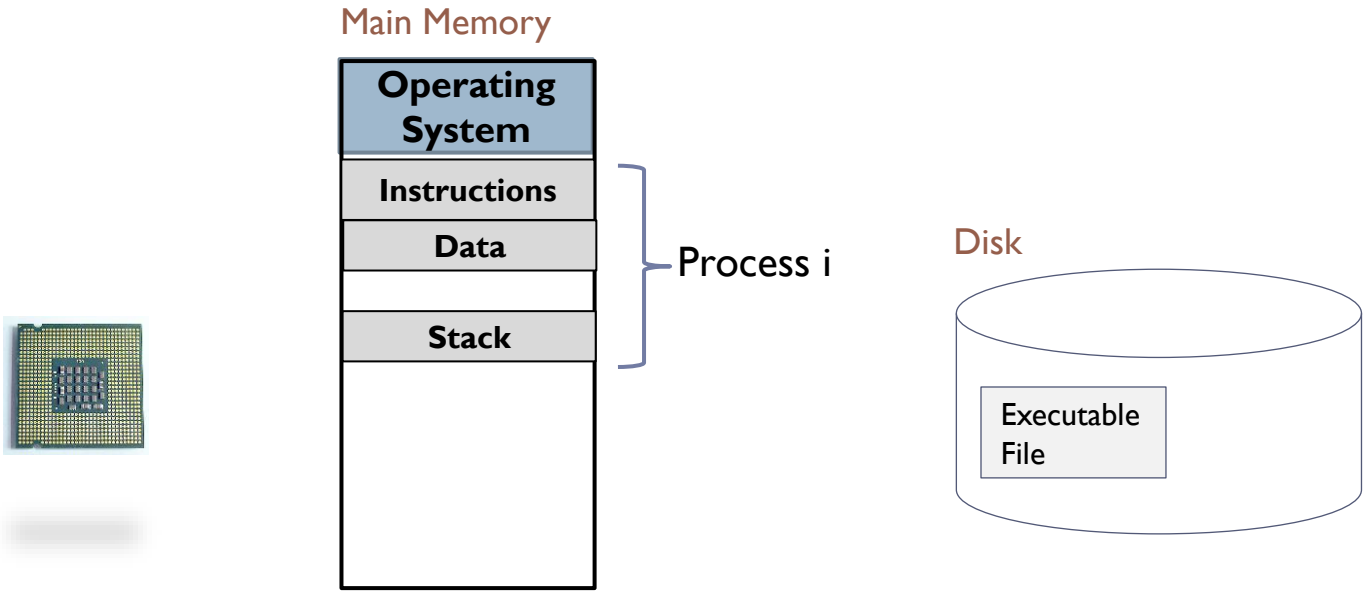
Program and process

- Process: program in execution
  - The same program can produce several processes



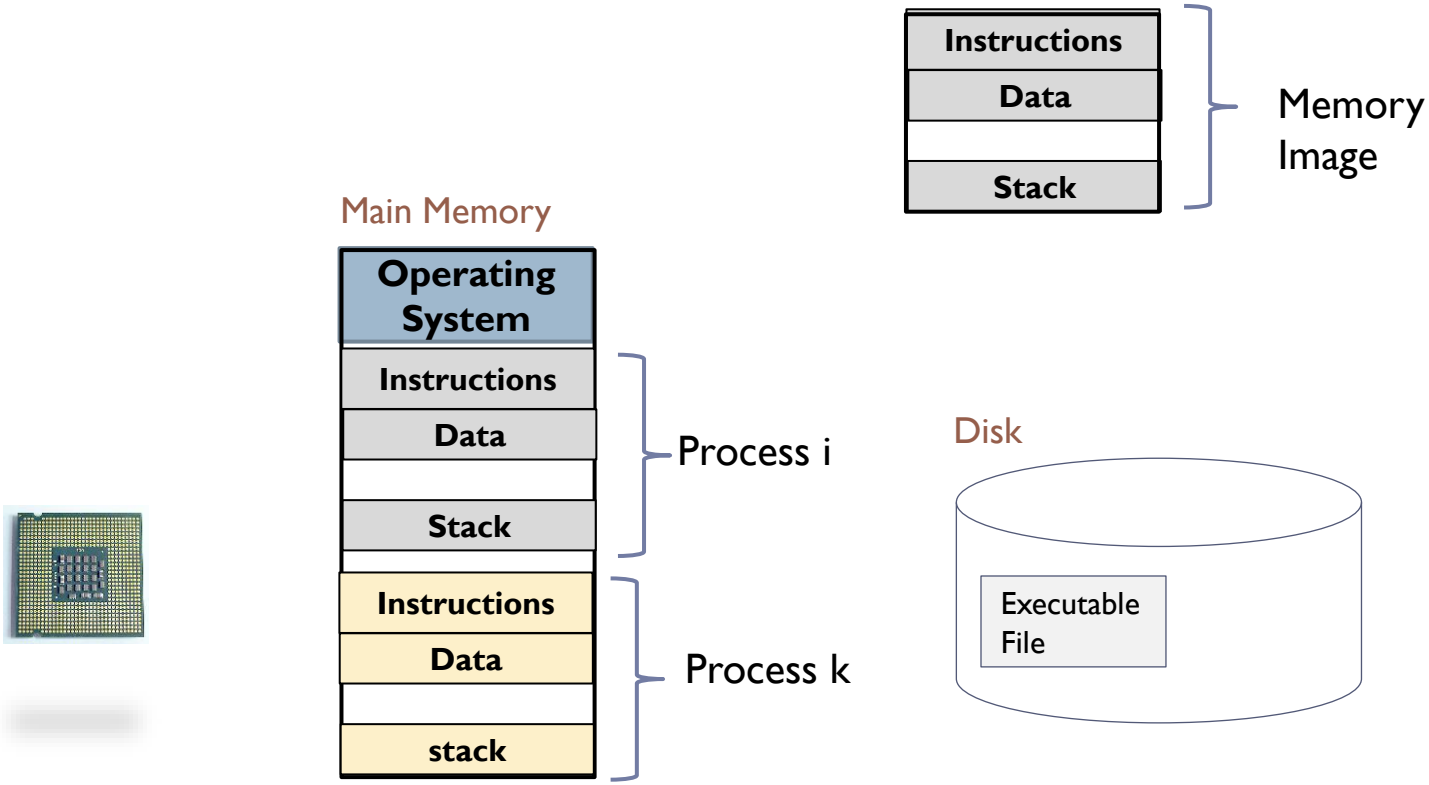
Program and process

- Program: A set of ordered data and instructions
  - Must be loaded in memory



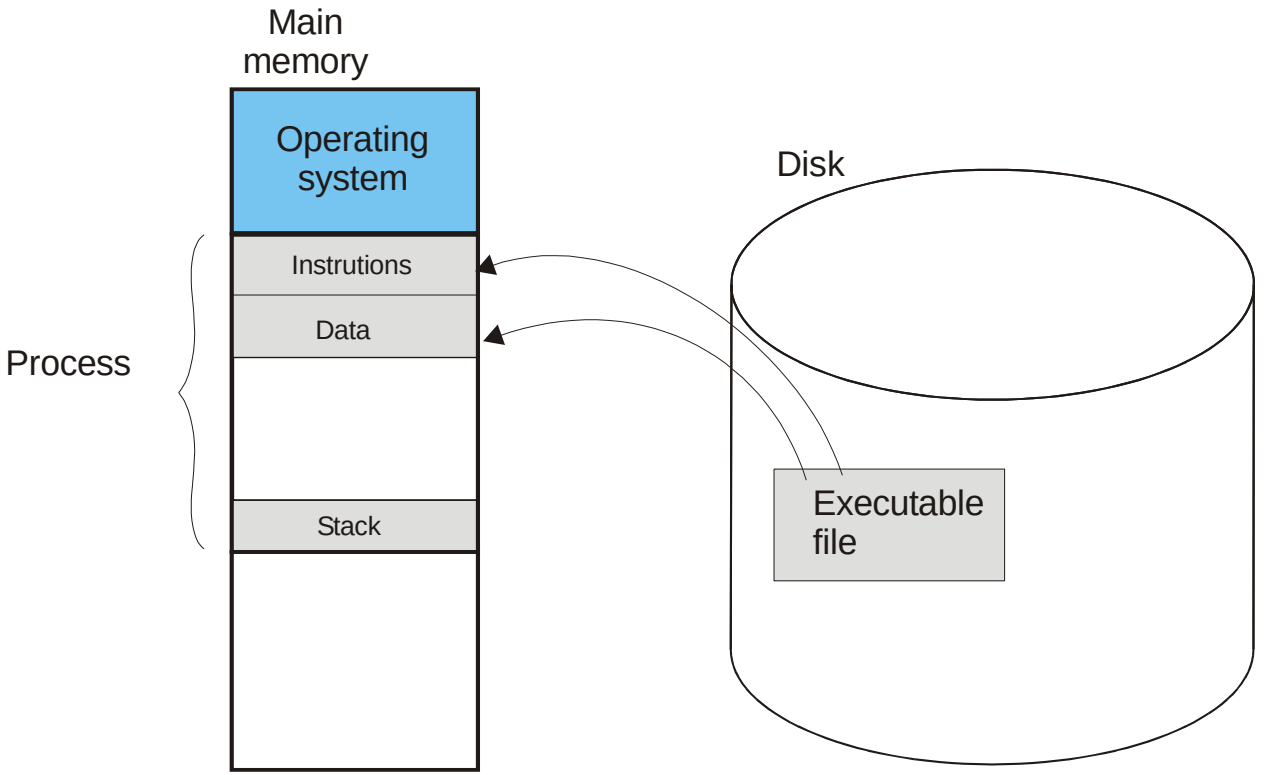
Memory Image of a Process

- Memory image consists of the **memory spaces** that a process is authorized to use.



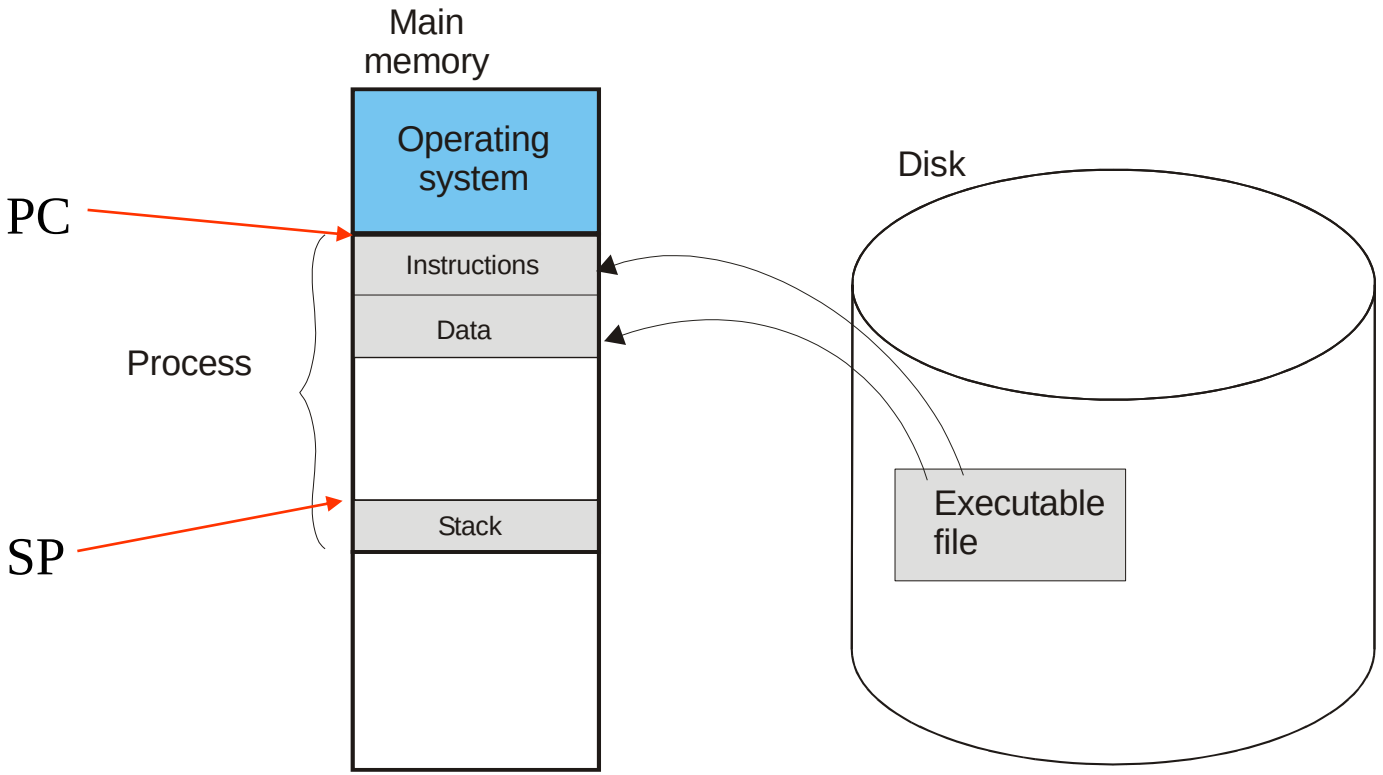
Systems without virtual memory

- In systems without virtual memory, the program is completely loaded in memory before the execution

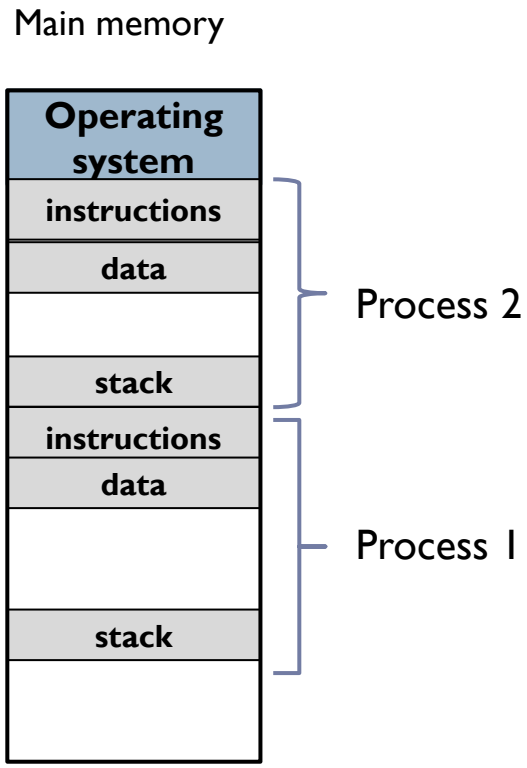


Systems without virtual memory

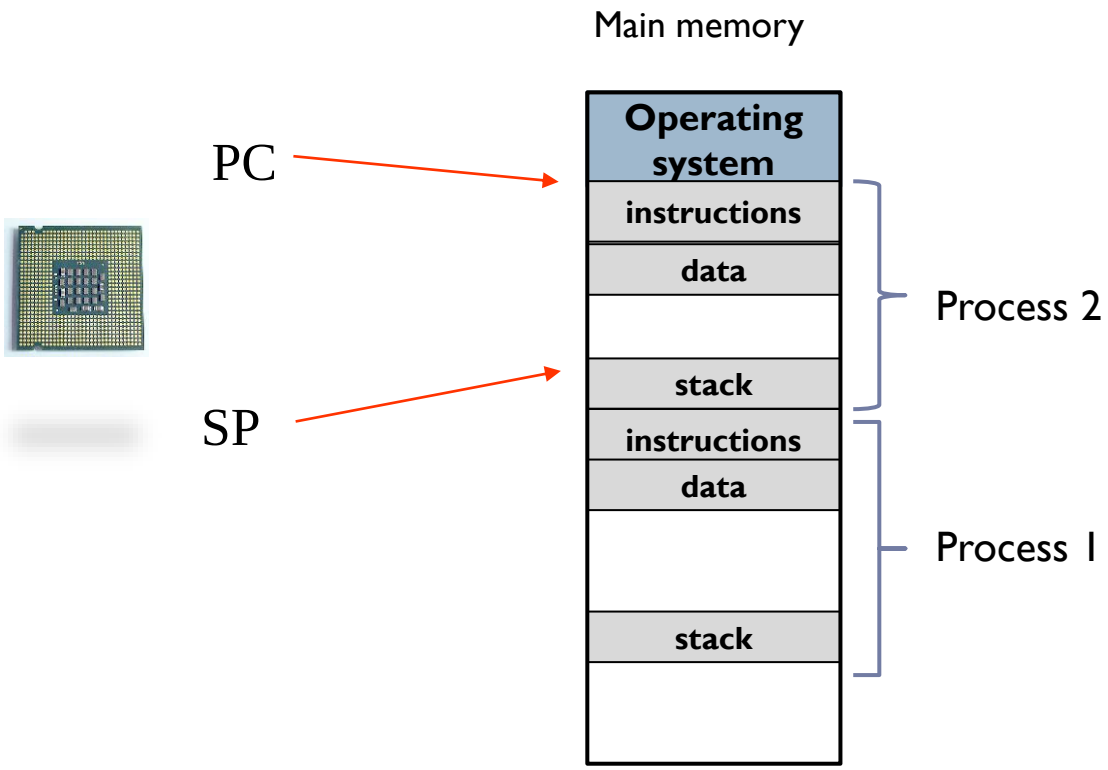
- Registers are initialized



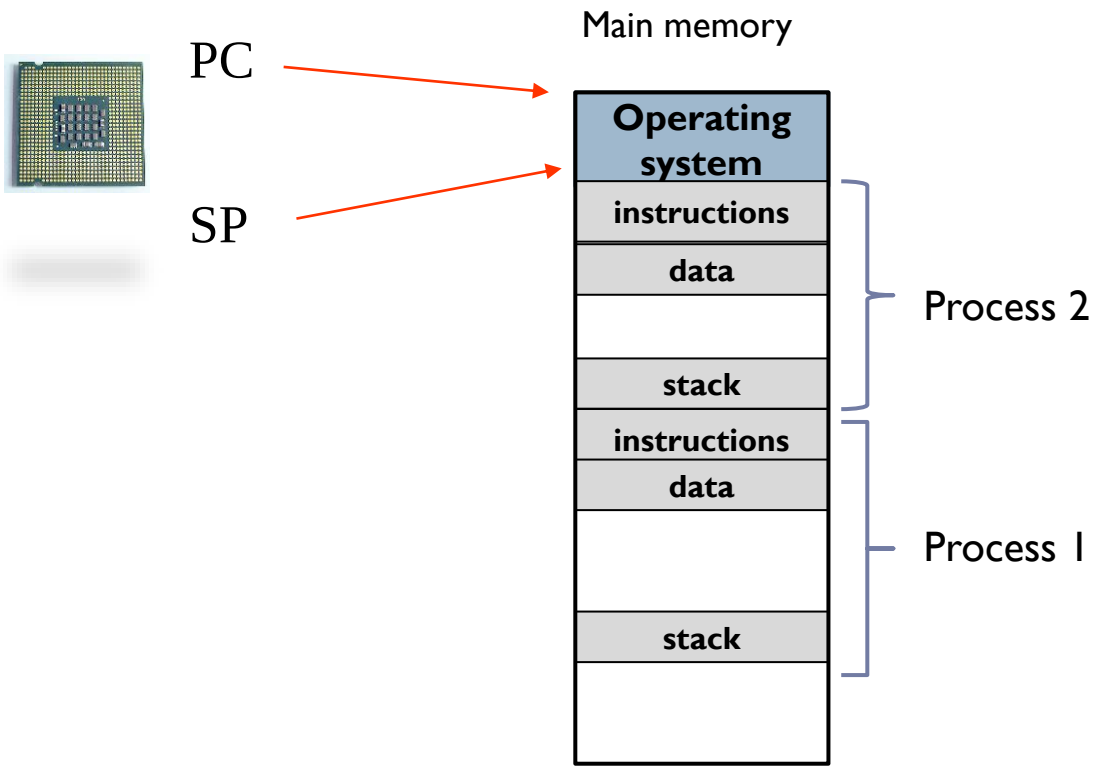
Multiple programs loaded in memory



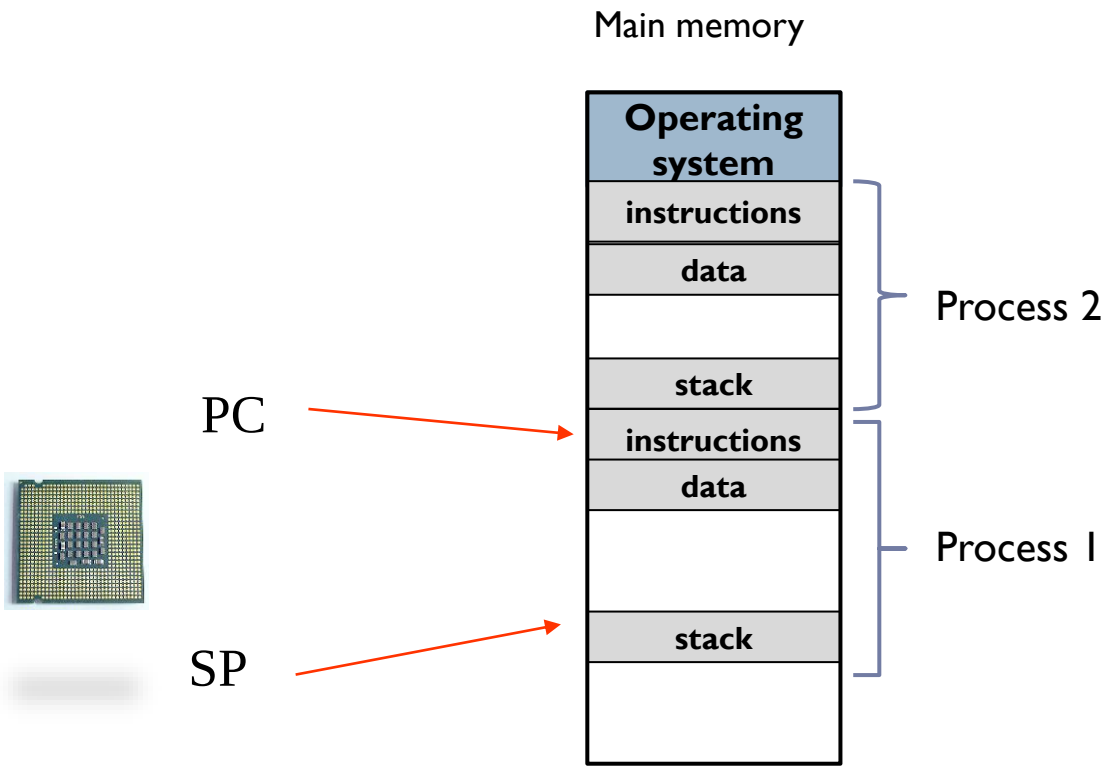
Multiple programs loaded in memory



Multiple programs loaded in memory

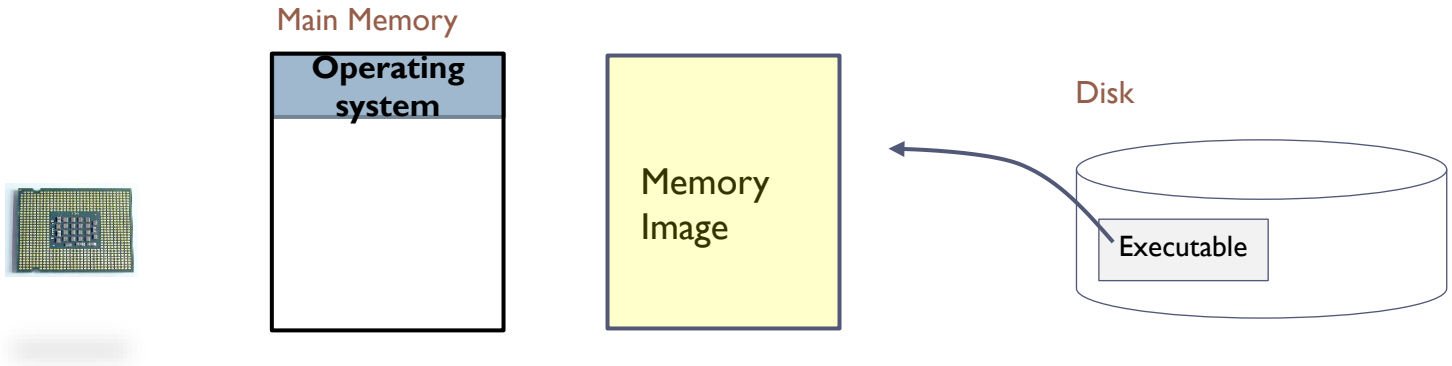


Multiple programs loaded in memory



Sistems without Virtual Memory

- On systems without virtual memory, the program is loaded completely into memory for execution.
- Main problems:
  - If the memory image of a process is larger than the main memory, its execution is not possible.
  - The large size of the memory image of a process may prevent the execution of other processes.



Hypothetical executable file

```
int v[1000];  // global
int i;
for (i=0; i < 1000; i++)
    v[i] = 0;
```

Hypothetical executable file

```
int v[1000];  // global
int i;
for (i=0; i < 1000; i++)
    v[i] = 0;
```

```
.data
        v: .space 4000
.text
main:   li   t0, 0
        li   t1, 0
        li   t2, 1000
bucle:  bge  t0, t2, fin
        sw   x0, v(t1)
        addi t0, t0, 1
        addi t1, t1, 4
        beq  x0, x0, bucle
fin:    …
```

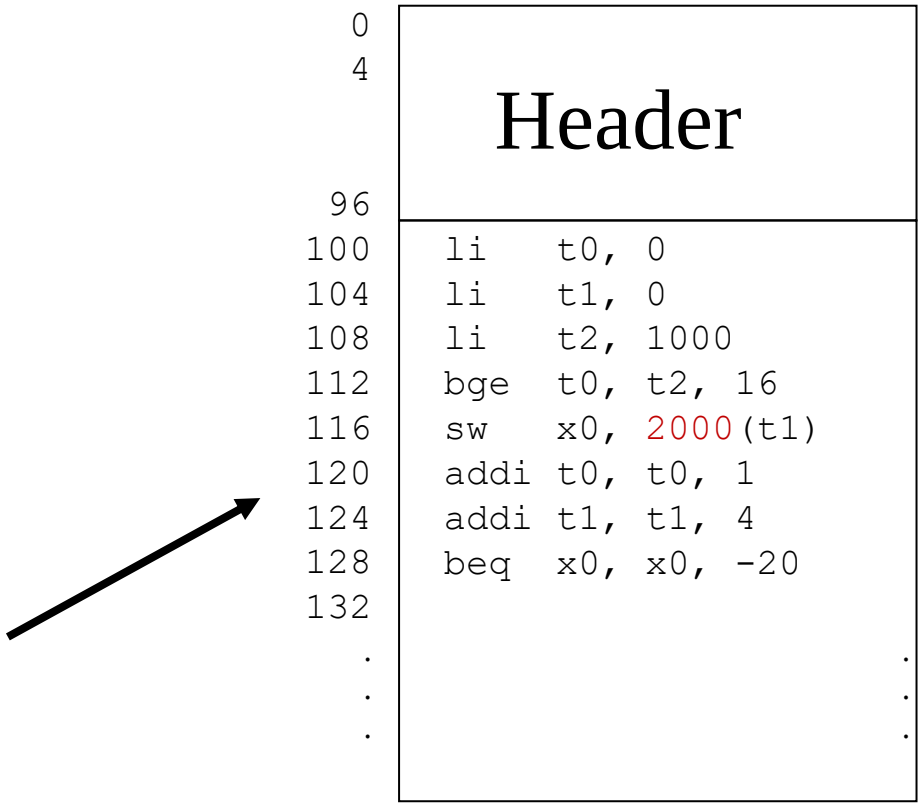
Hypothetical executable file

```
int v[1000];  // global
int i;
for (i=0; i < 1000; i++)
    v[i] = 0;
```

assembly

```
.data
        v: .space 4000
.text
main:   li   t0, 0
        li   t1, 0
        li   t2, 1000
bucle:  bge  t0, t2, fin
        sw   x0, v(t1)
        addi t0, t0, 1
        addi t1, t1, 4
        beq  x0, x0, bucle
fin:    …
```

executable

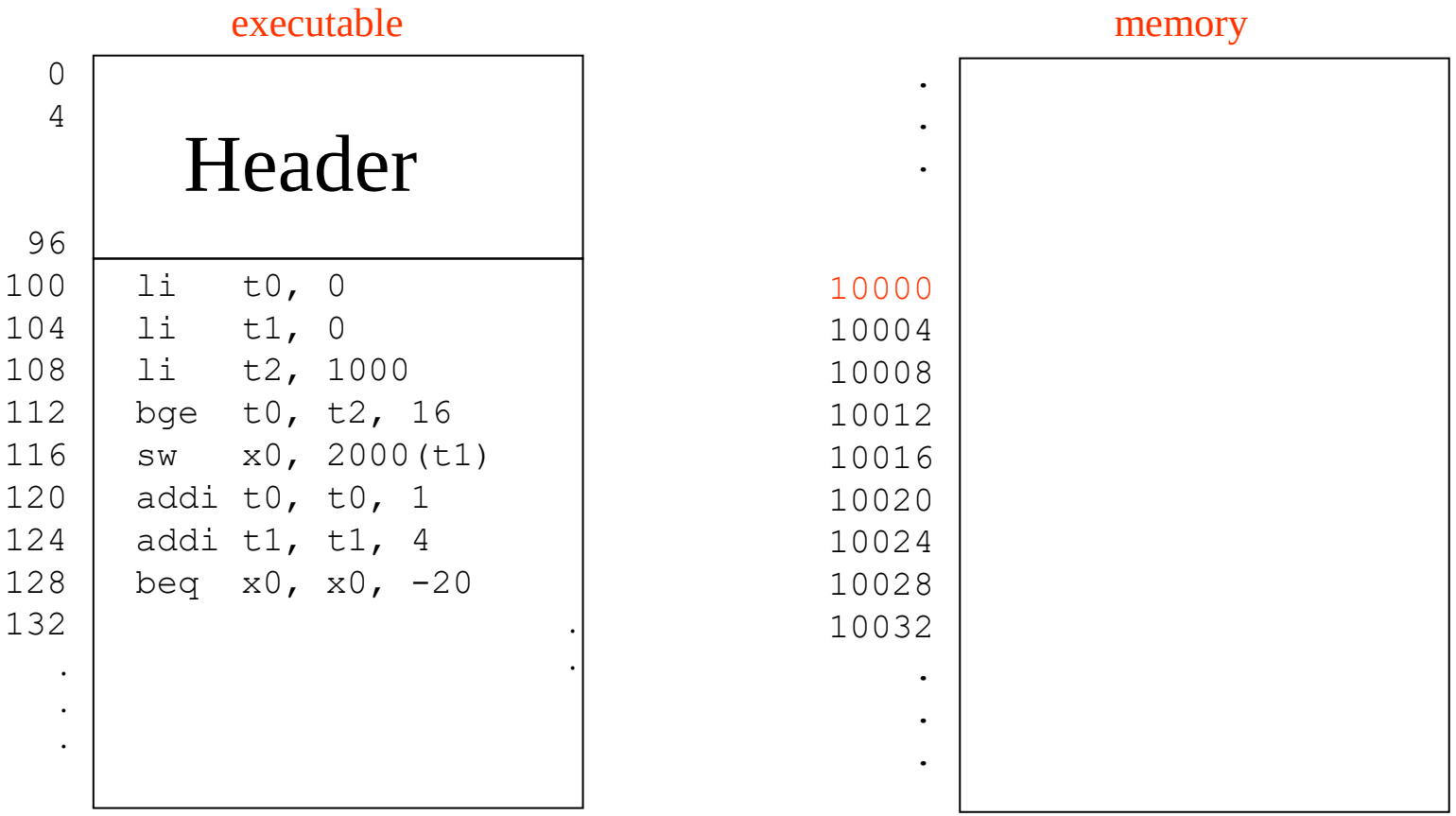


Address 2000 is assigned to v

Assumes that program starts in address 0

Loading the program in memory

- The Operating System reserves a contiguous free portion in memory for the entire process image

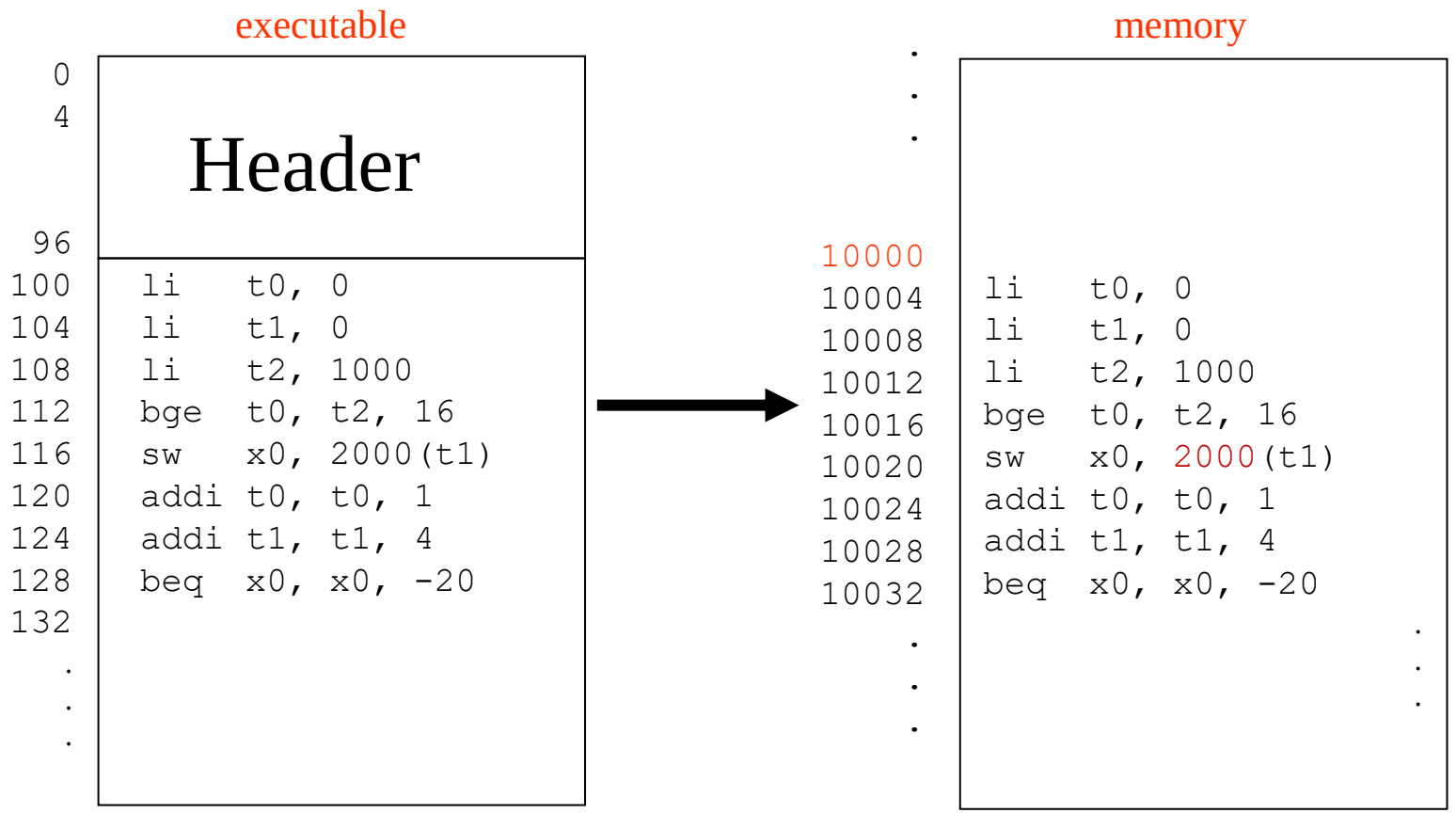


Loading the program in memory

- In the executable file the address 0 is considered as the init address
  - Logical address
- In memory, the init address is10000
  - Physical address
- Address translation is needed
  - From logical address to physical
- The array in memory is in:
  - The logical address 2000
  - The physical address 2000 + 10000
- This process is called relocation
  - Software relocation
  - Hardware relocation

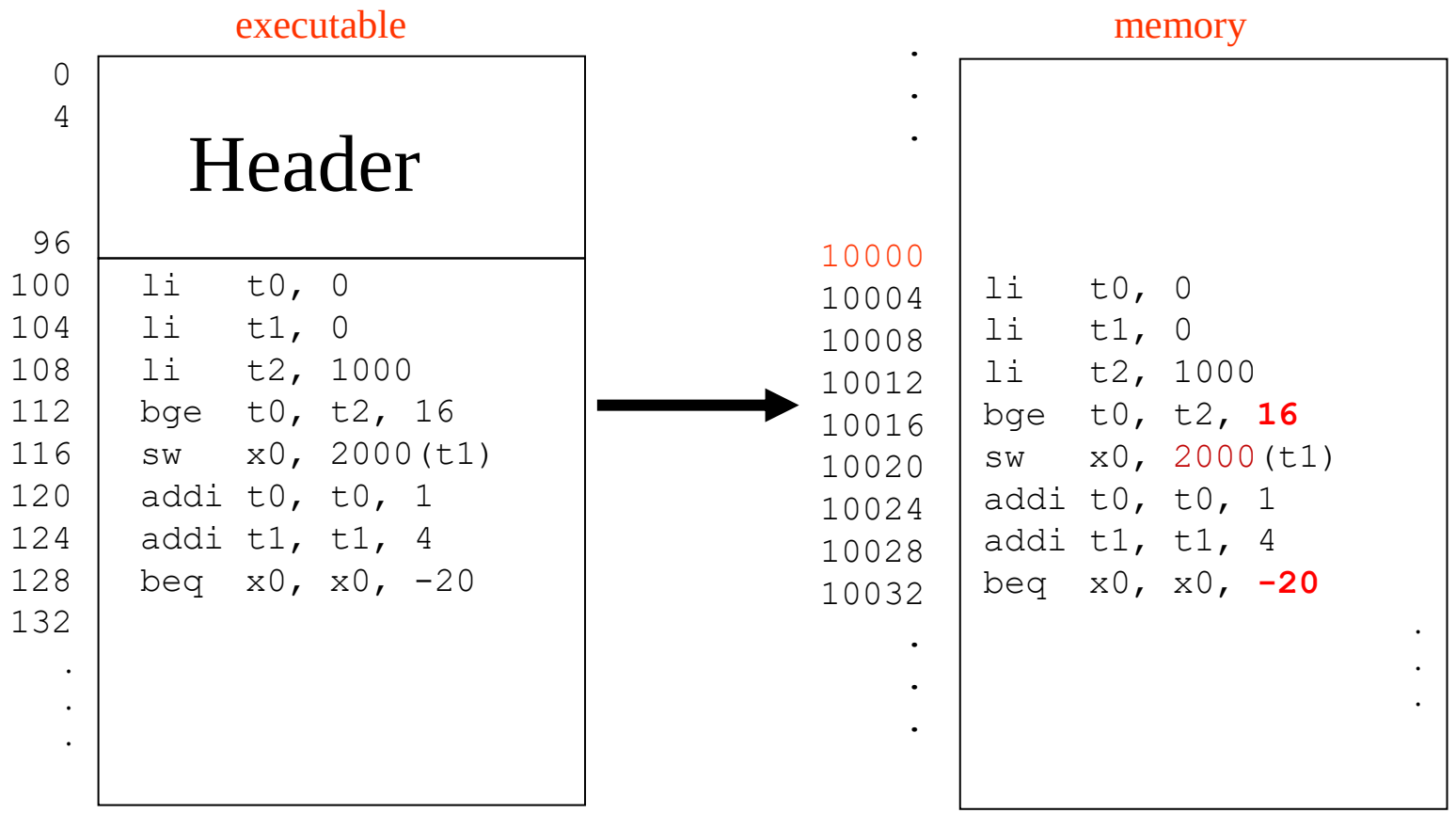
Software relocation

- Occurs in the loading process



Software relocation

- What happens with the instructions loaded in 10012 and 10028 addresses?



Problem with memory protection

- What happens if the program executes these instructions?

```
li t0, 8
sw t0, (x0)
```

Problem with memory protection

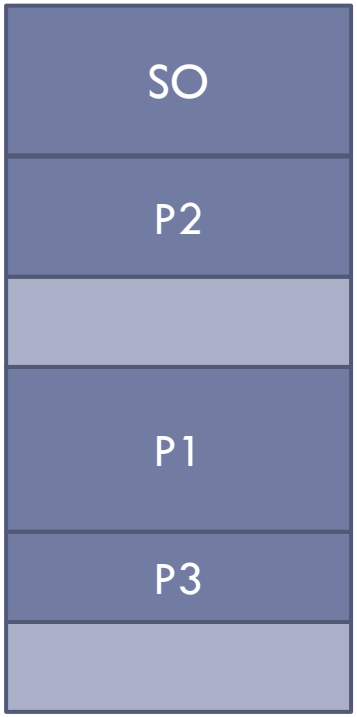
- What happens if the program executes these instructions?

```
li t0, 8
sw t0, (x0)
```

Illegal access to physical address 0 that is not assigned to the program

Multiprogramming

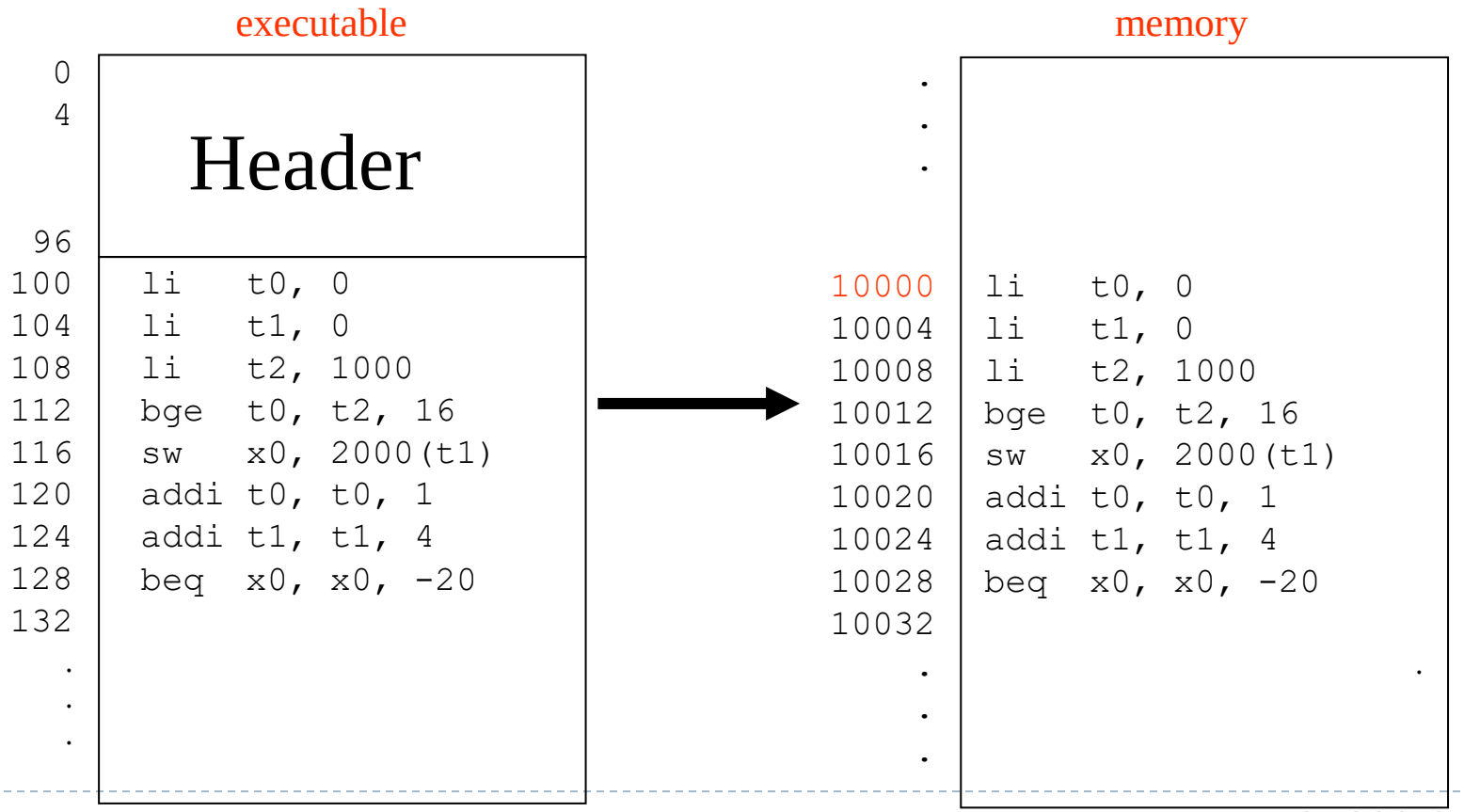
- A computer can store several programs in memory
- Each program needs an address space in memory



We need to ensure that a program does not access to the address space of other program

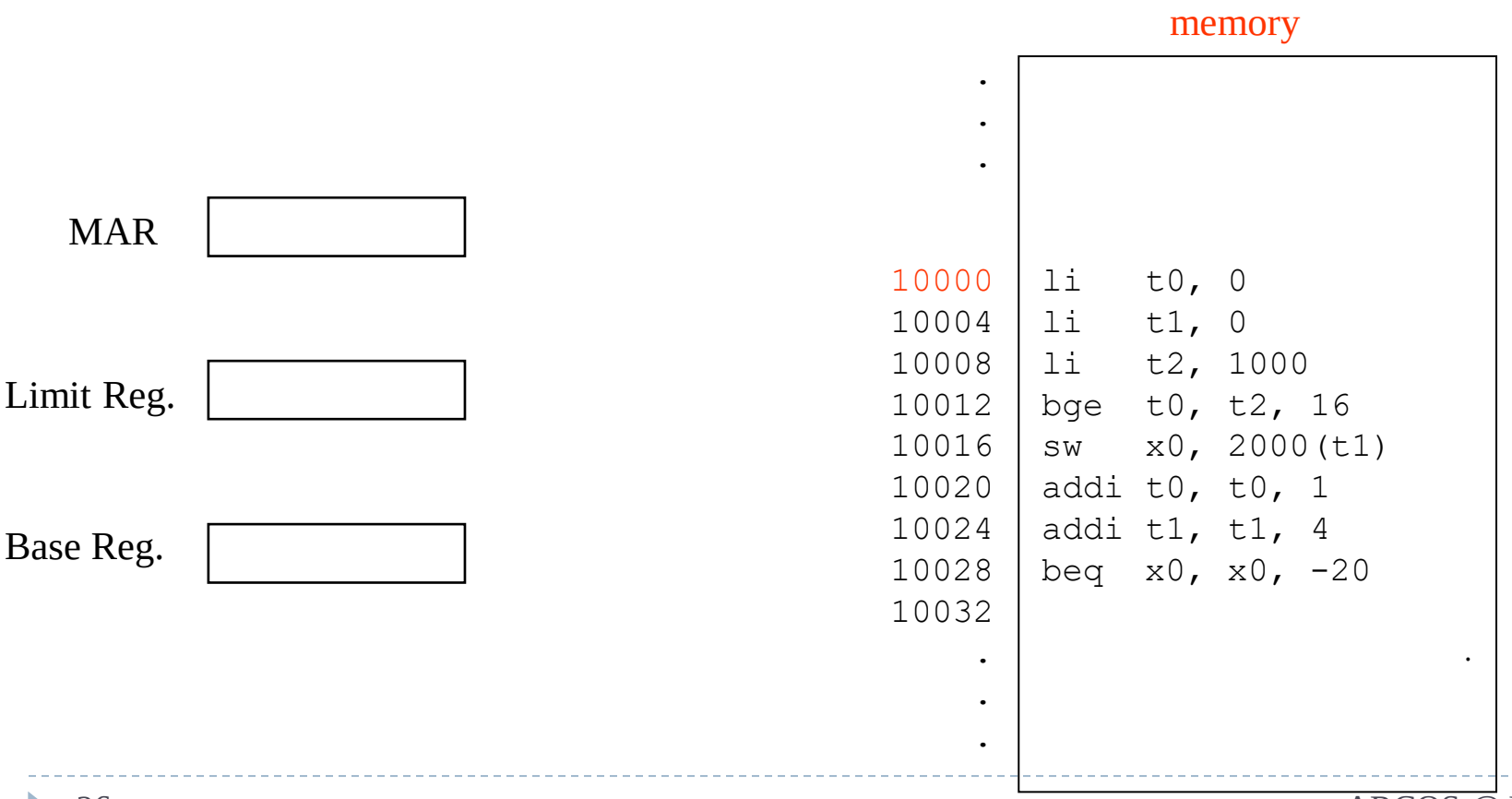
Hardware relocation

- The translation occurs in the execution
- Special HW is needed. Ensure protection



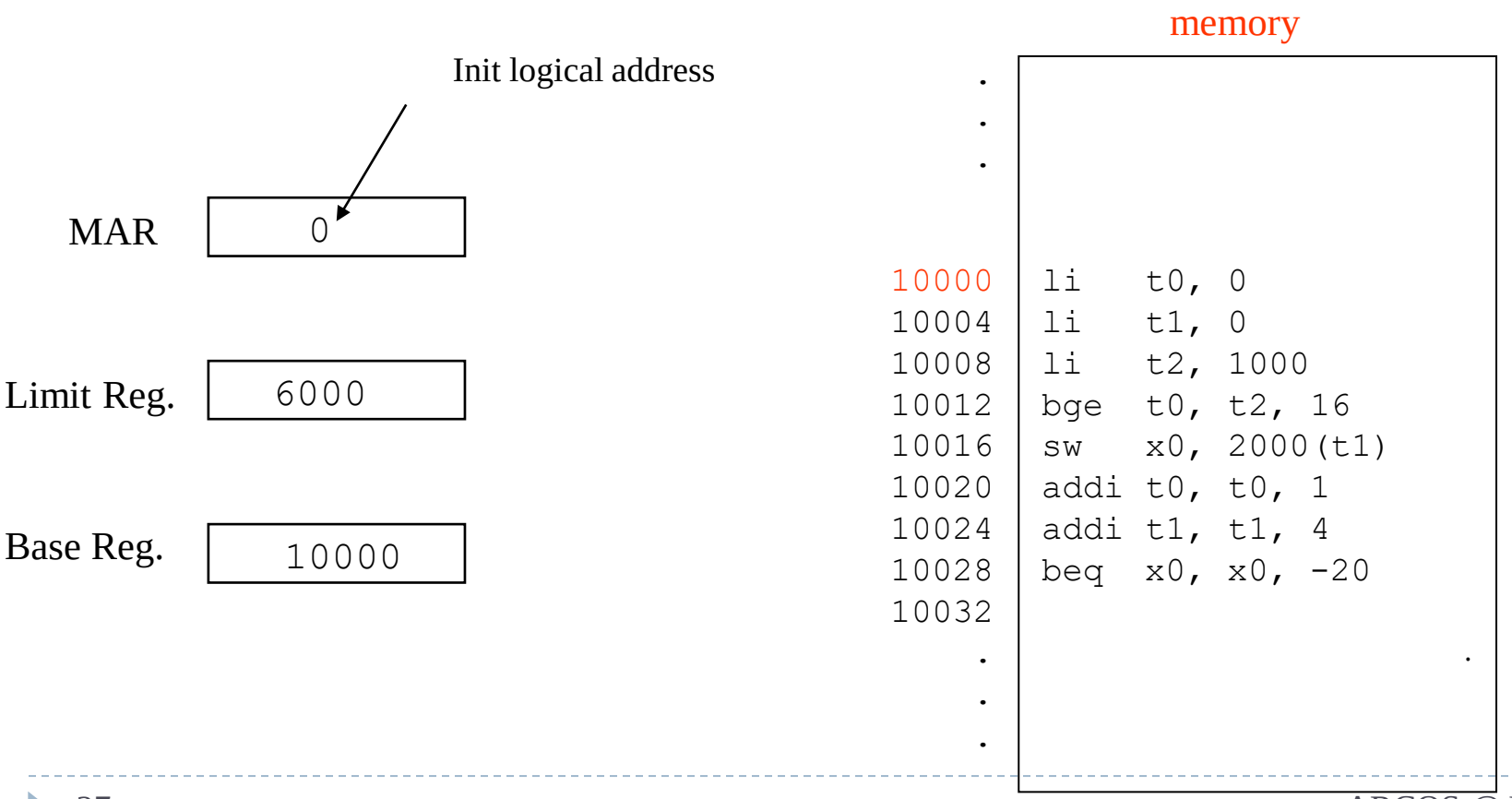
Example of hardware support

- Limit register: maximum logical address assigned to the program
- Base register: program init address in memory



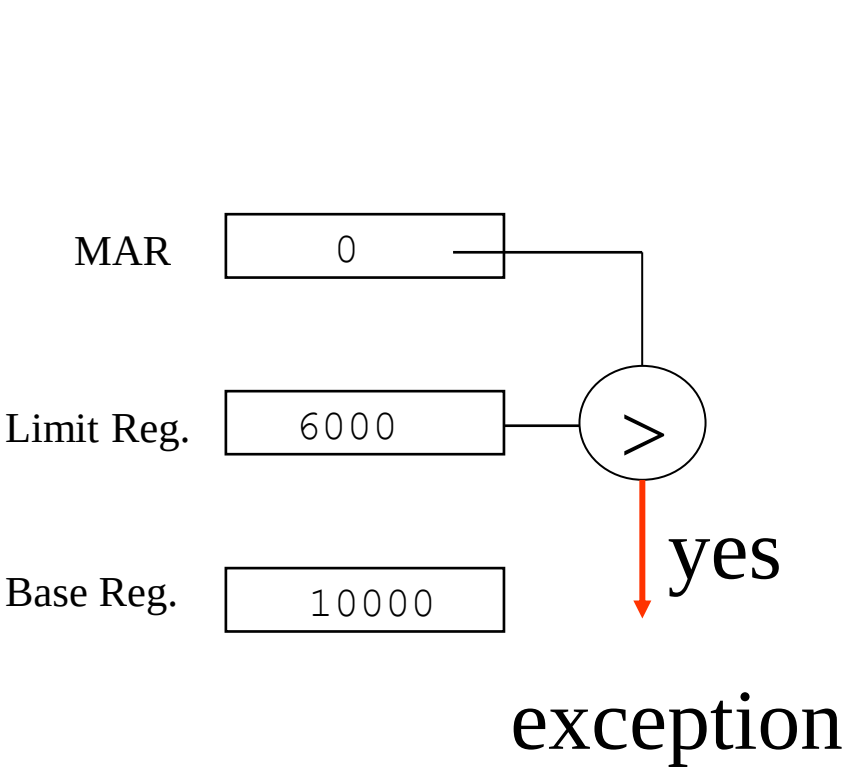
Example of hardware support

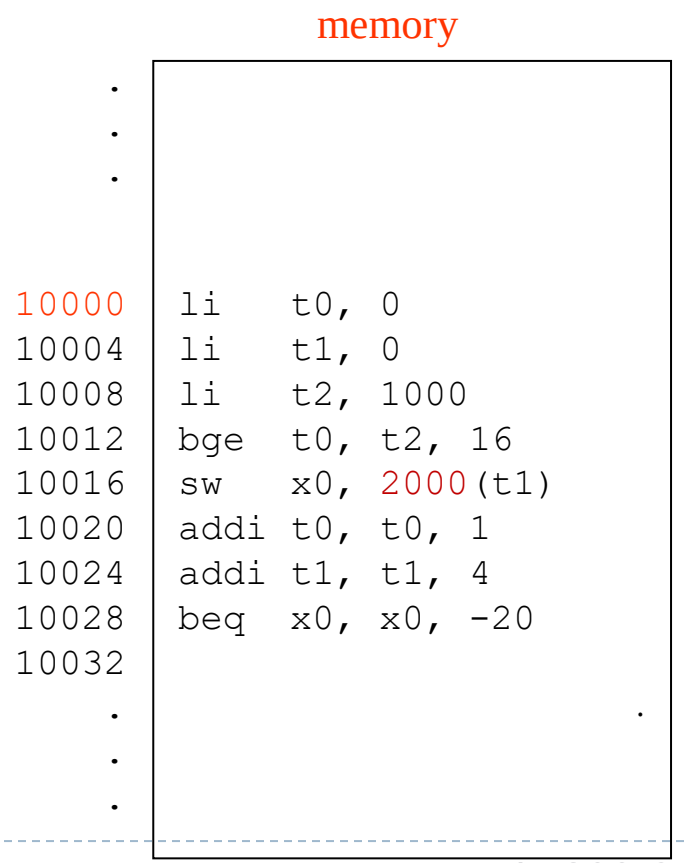
- Limit register: maximum logical address assigned to the program
- Base register: program init address in memory



Example of hardware support

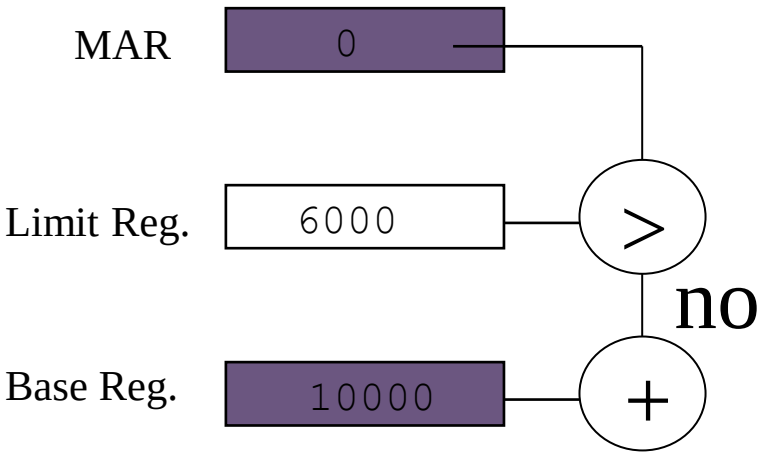
- Limit register: maximum logical address assigned to the program
- Base register: program init address in memory





Example of hardware support

- Limit register: maximum logical address assigned to the program
- Base register: program init address in memory

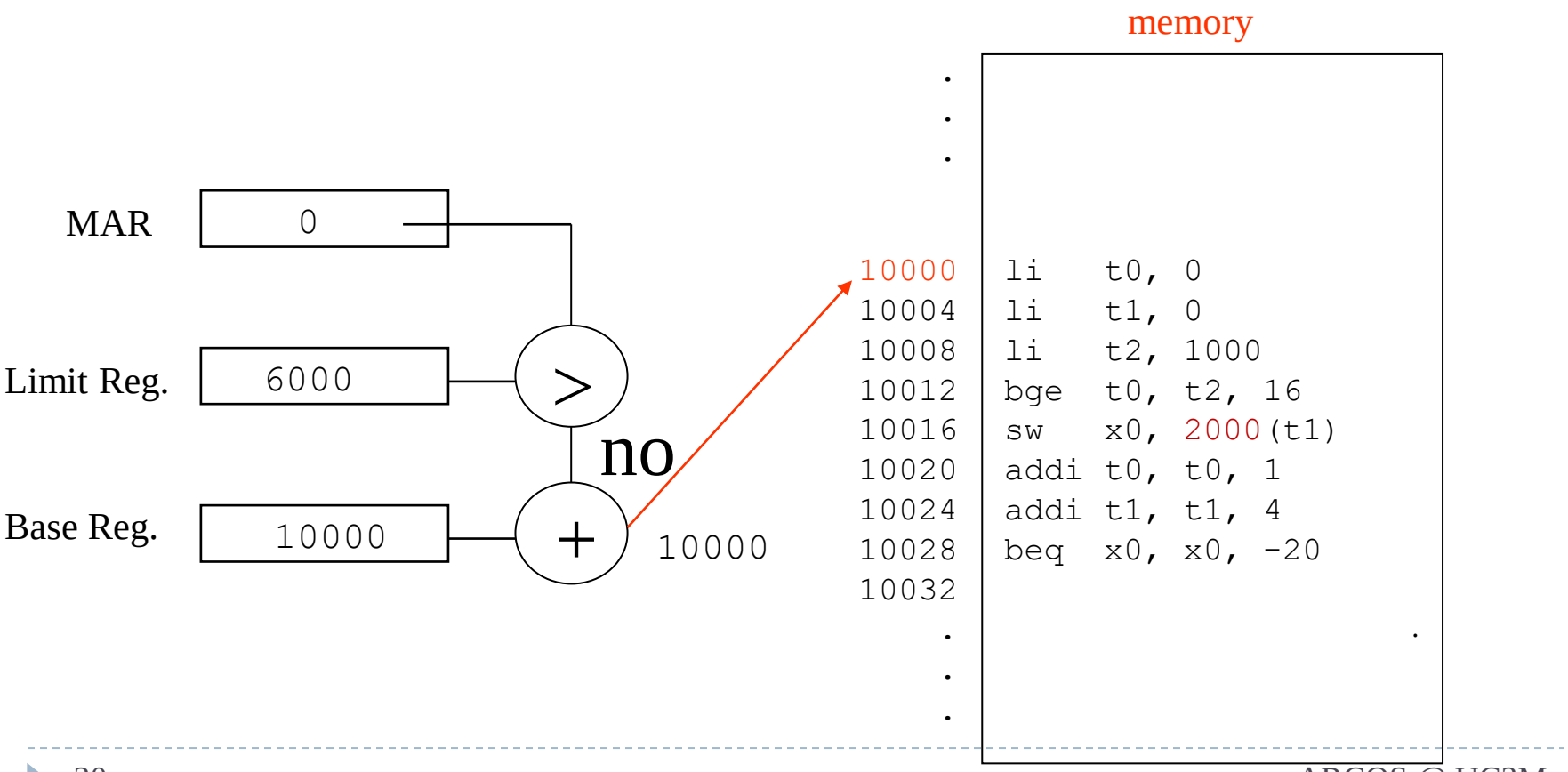


memory

```
10000  li   t0, 0
10004  li   t1, 0
10008  li   t2, 1000
10012  bge  t0, t2, 16
10016  sw   x0, 2000(t1)
10020  addi t0, t0, 1
10024  addi t1, t1, 4
10028  beq  x0, x0, -20
10032
```

Example of hardware support

- Limit register: maximum logical address assigned to the program
- Base register: program init address in memory

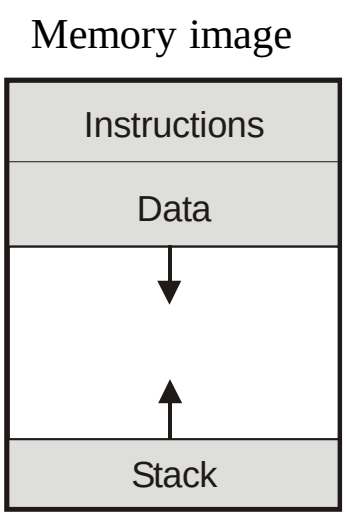


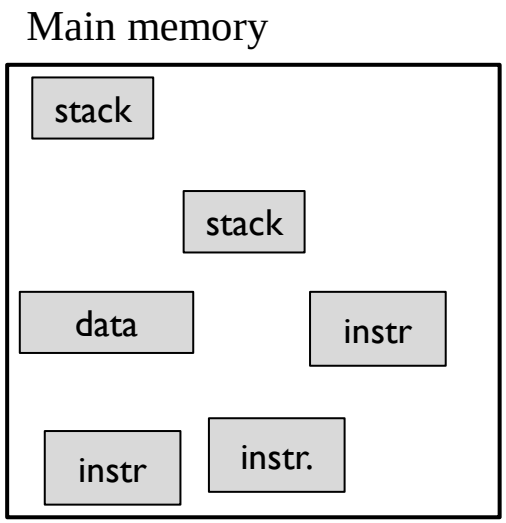
Systems without virtual memory
Main problems

- If the process image is bigger than the available memory, the process can not be executed
- In a 32-bit computer:
  - What is the theoretical maximum size of a program?
  - What if this size if the memory has 512 MB?
- The number of active programs is reduced

Virtual memory

- It is not needed to load the entire process in memory
- Only the program portions needed are loaded in memory
- Main advantages:
  - We can execute a program bigger than the main memory available
  - More programs can be active in memory

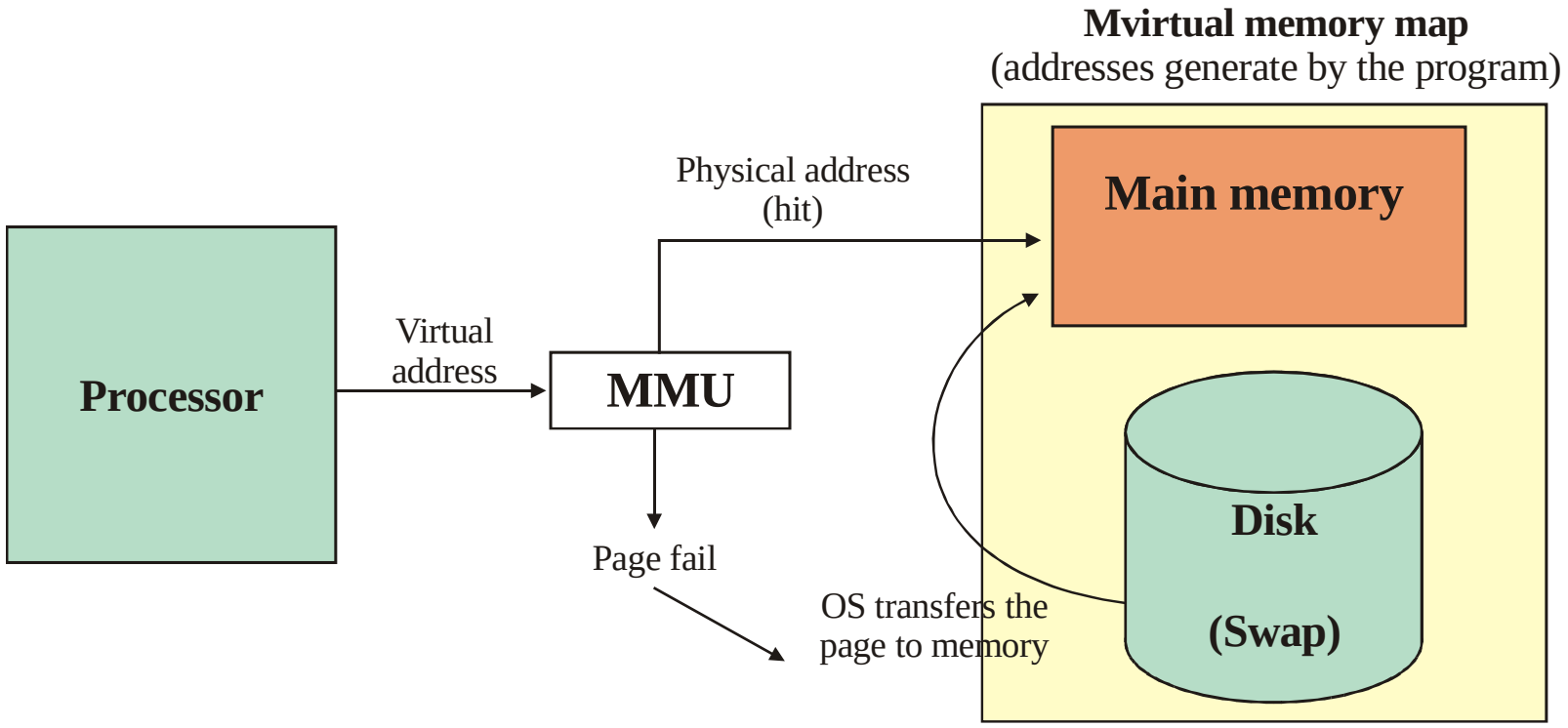




Main concepts on virtual memory

Virtual memory uses:

- Main memory: RAM
- Secondary memory: ssd, disk

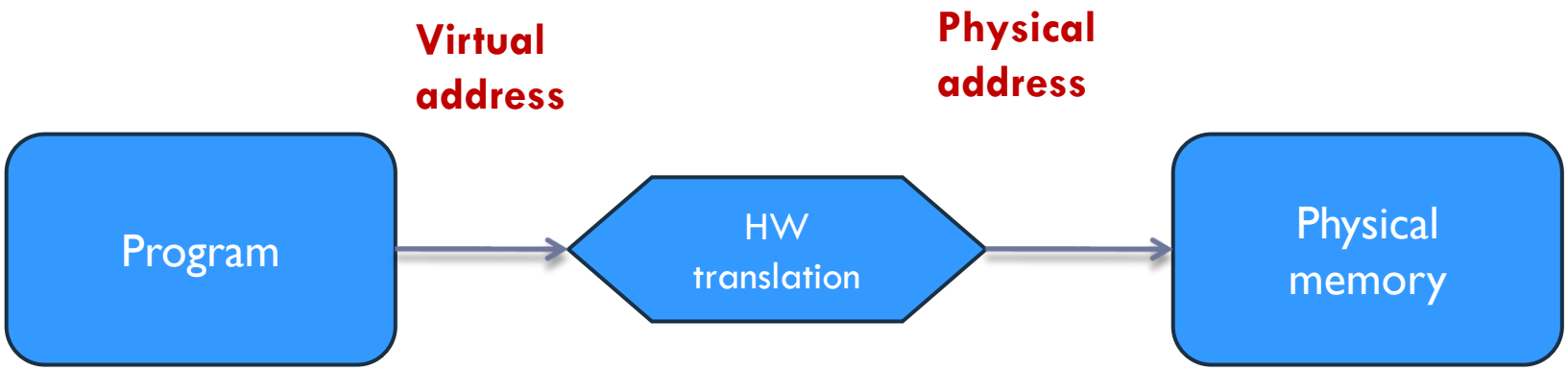


Pages virtual memory

- Processors generate virtual addresses
- The virtual address space is divided in equal size blocks called pages
- Main memory is divided in equal size blocks called page frames
- The part of the disk that supports the virtual memory is divided in equal size blocks called swap pages

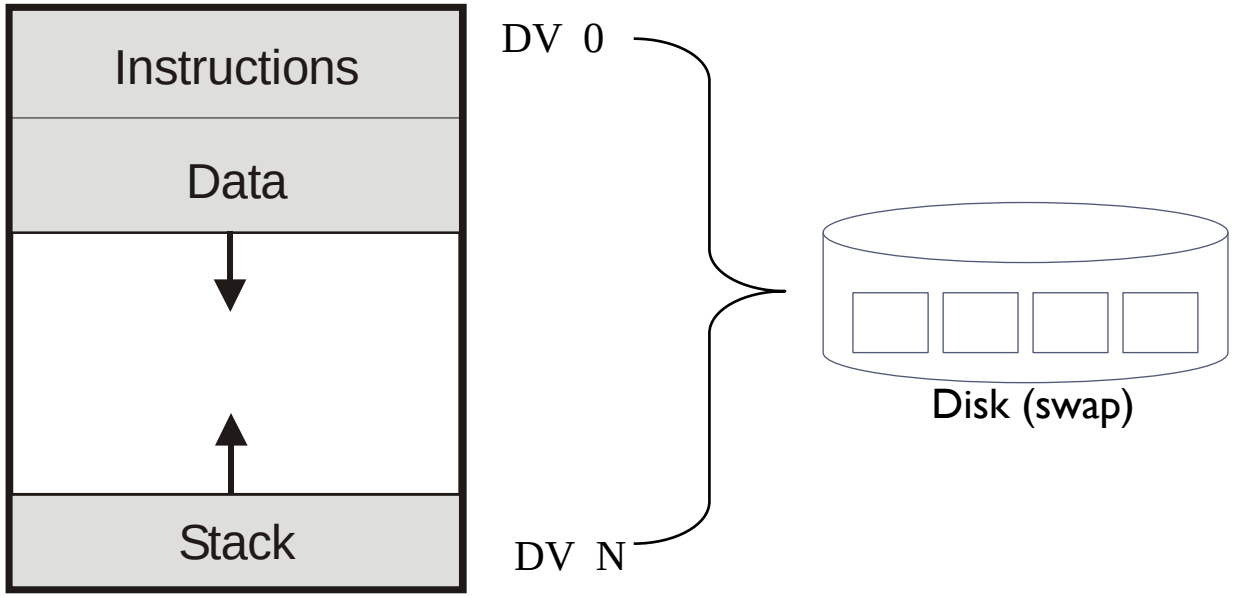
Physical address and virtual address

- Virtual address space:
  - Memory addresses that use the processor.
- Physical address space:
  - Main memory addresses

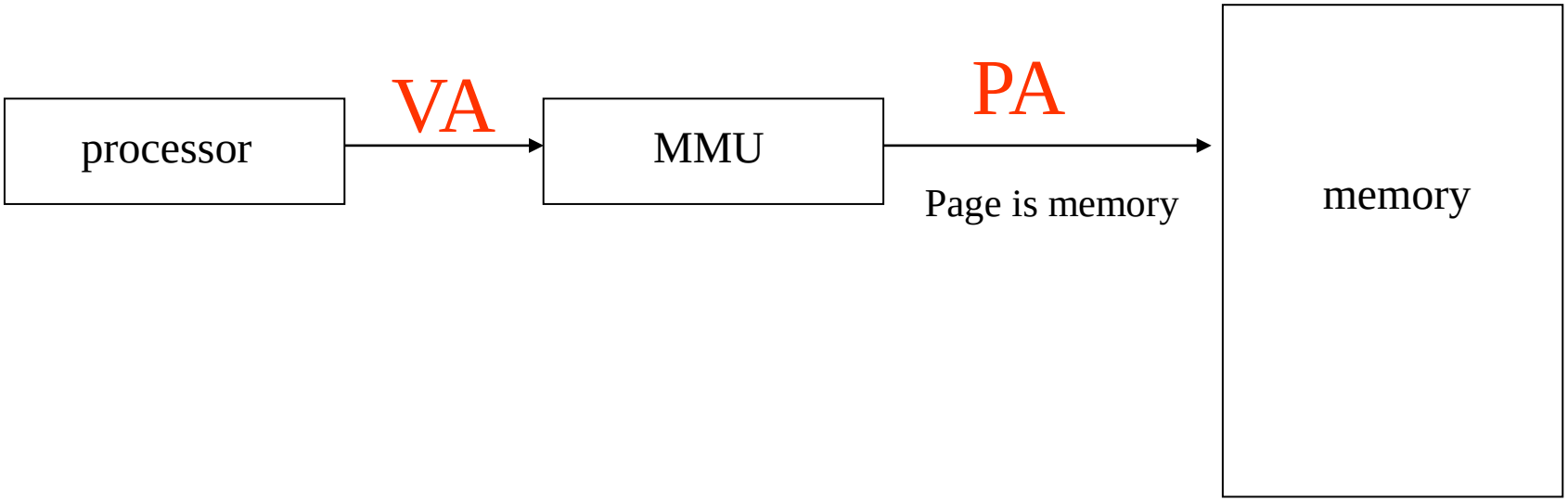


Paged virtual memory

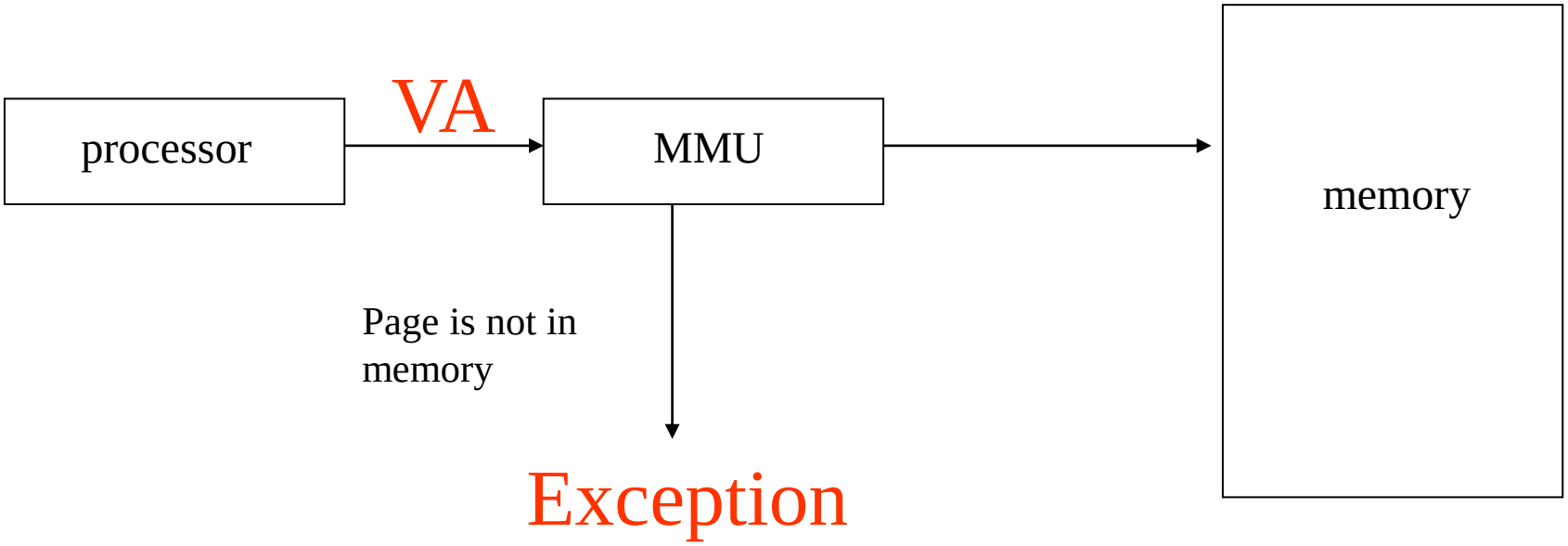
- The memory image of the programs are stored in disk



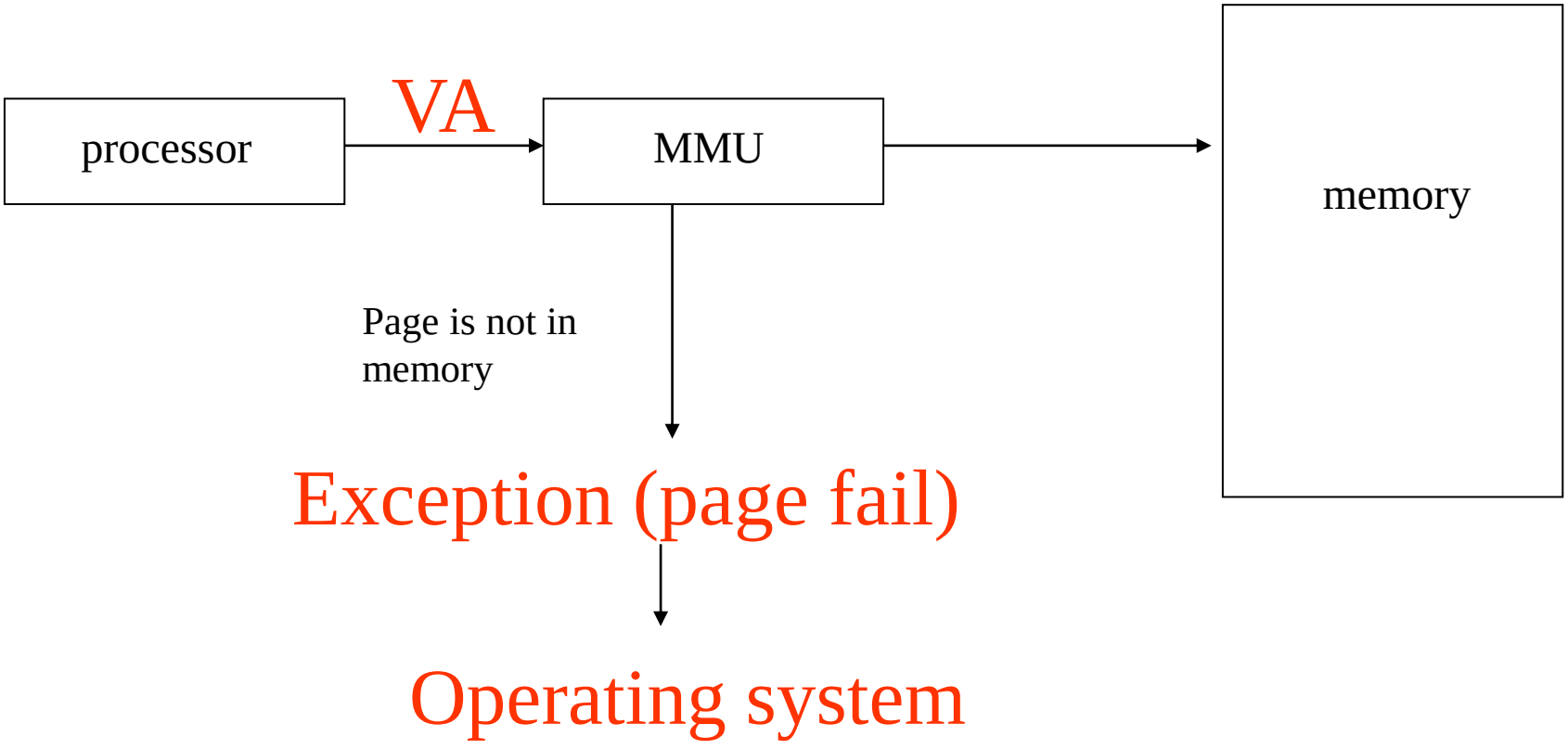
Address translation



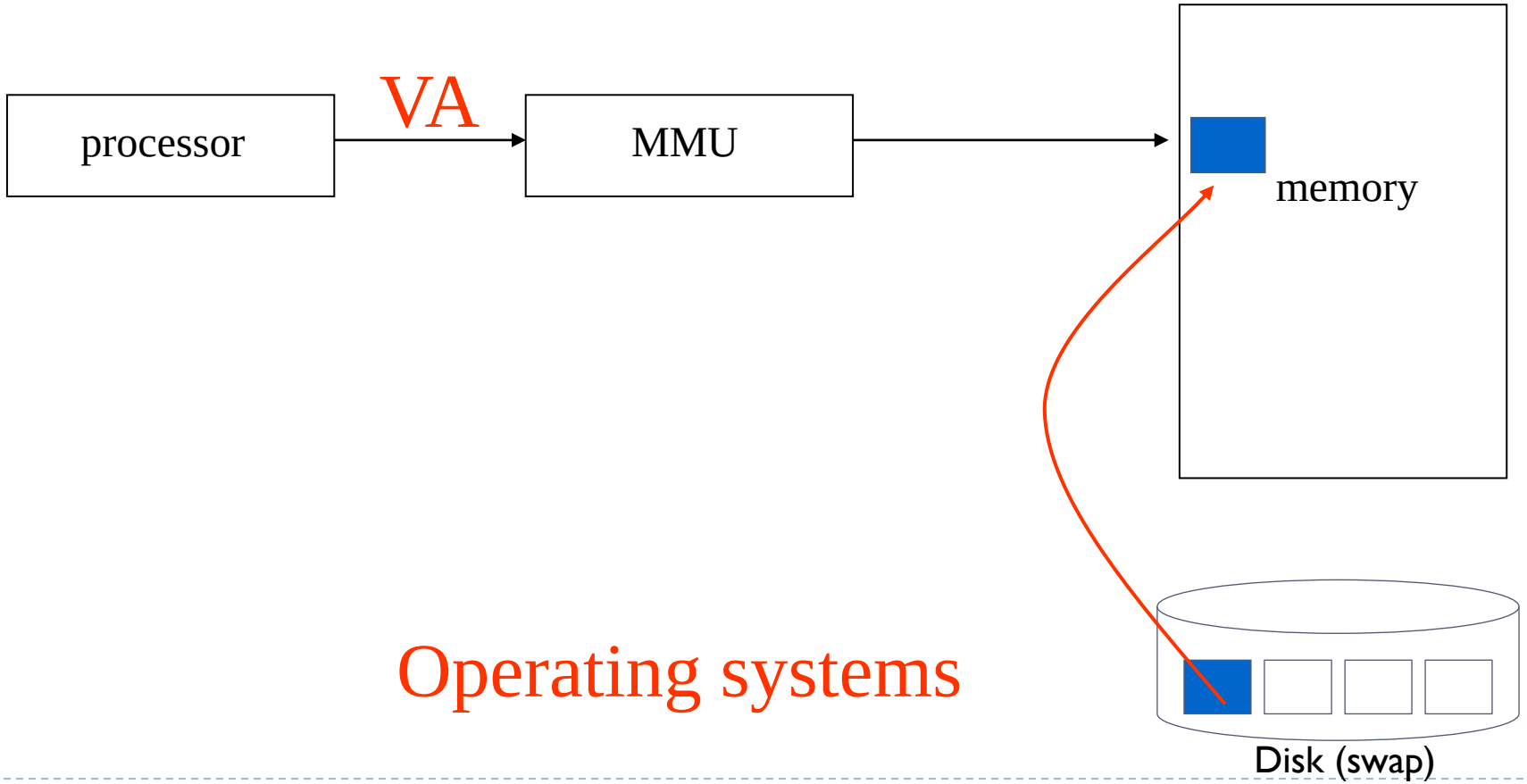
Address translation



Address translation

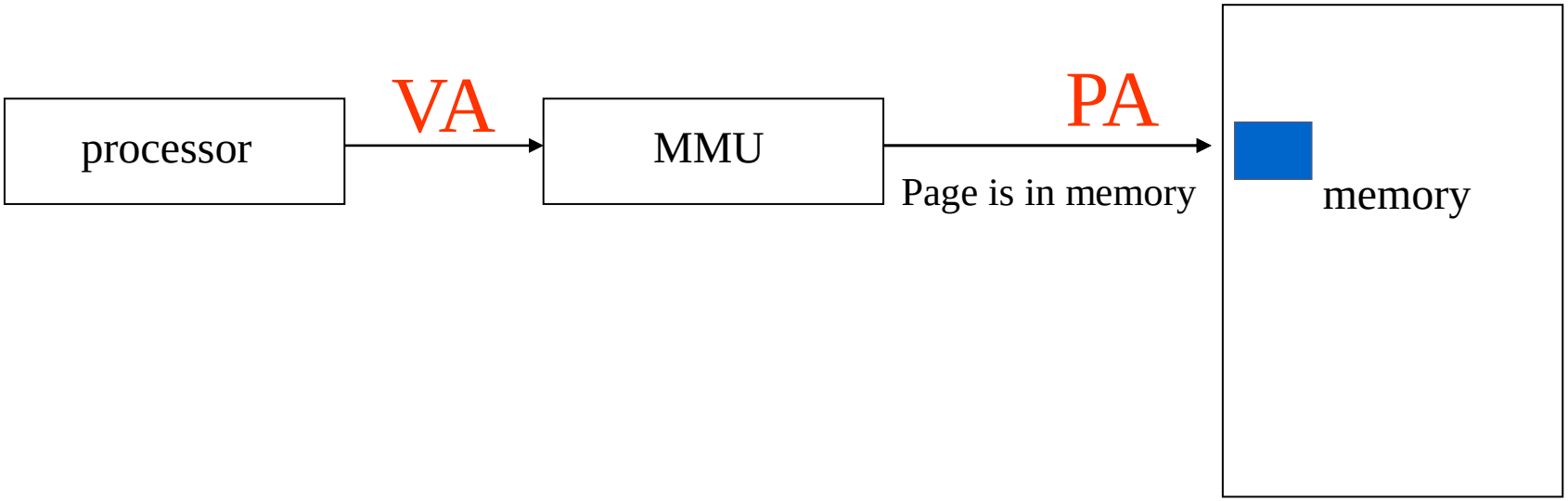


Address translation

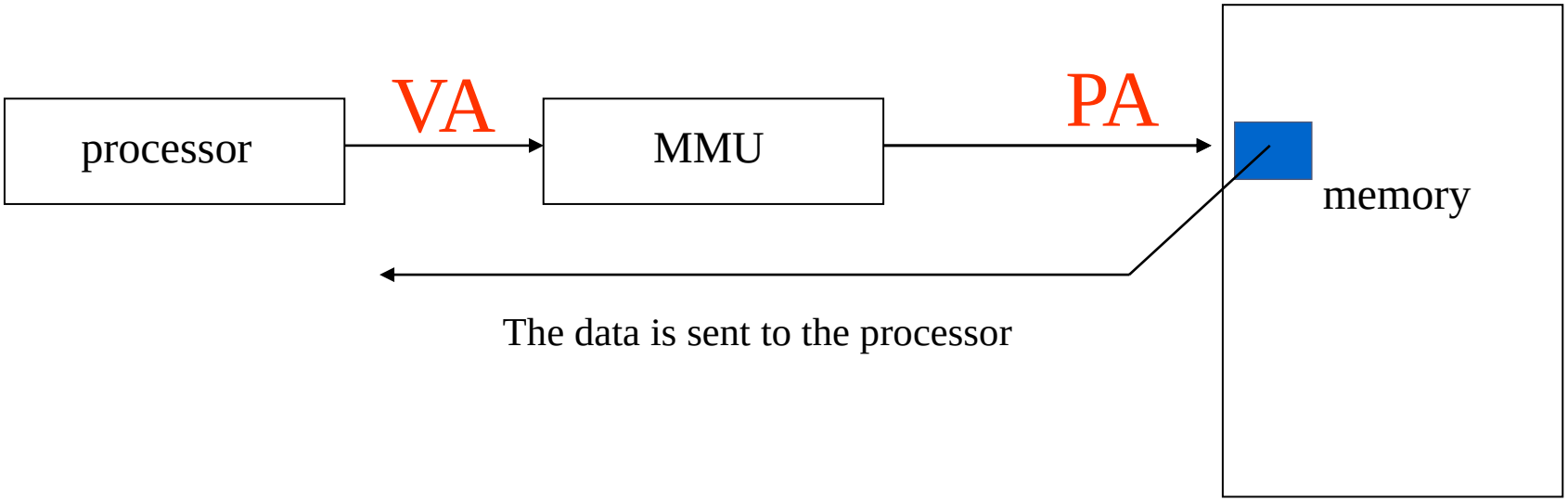


Operating systems

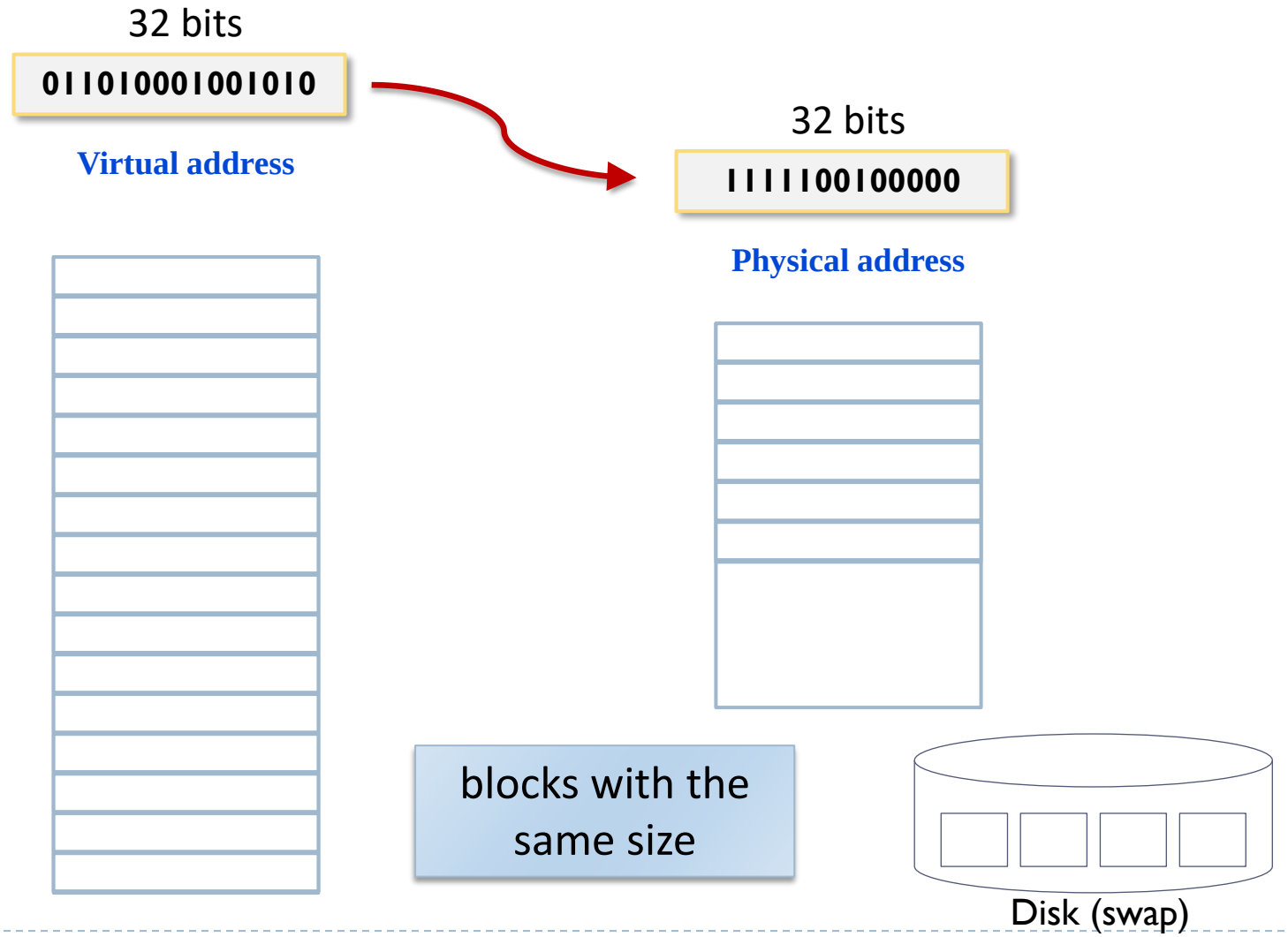
Address translation



Address translation

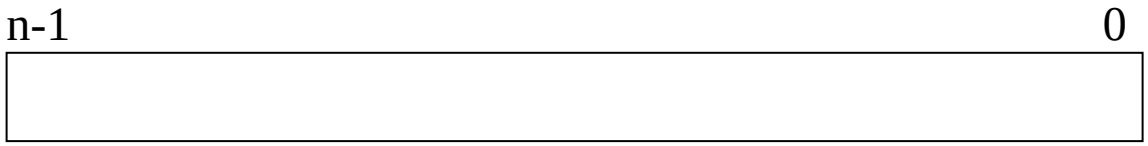


Paged virtual address



Structure of a virtual address

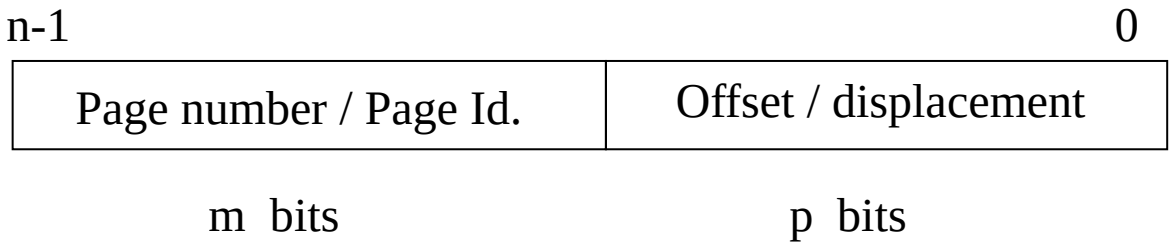
- A n bit computer has:
  - Addresses of n bits



  - Can address 2^n bytes

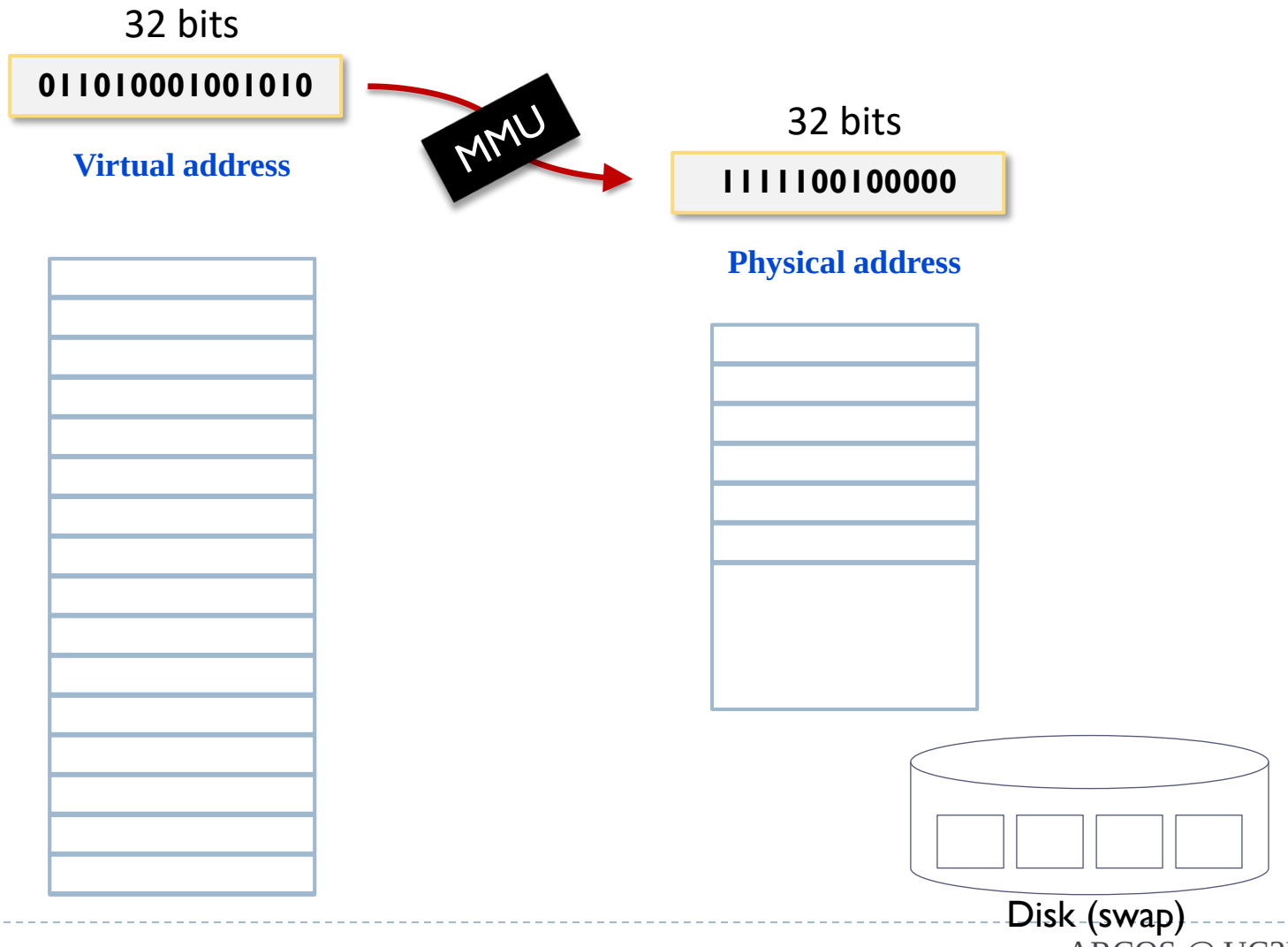
Structure of a virtual address

- Memory image consists of pages with the same size(4 KB, 8 KB)

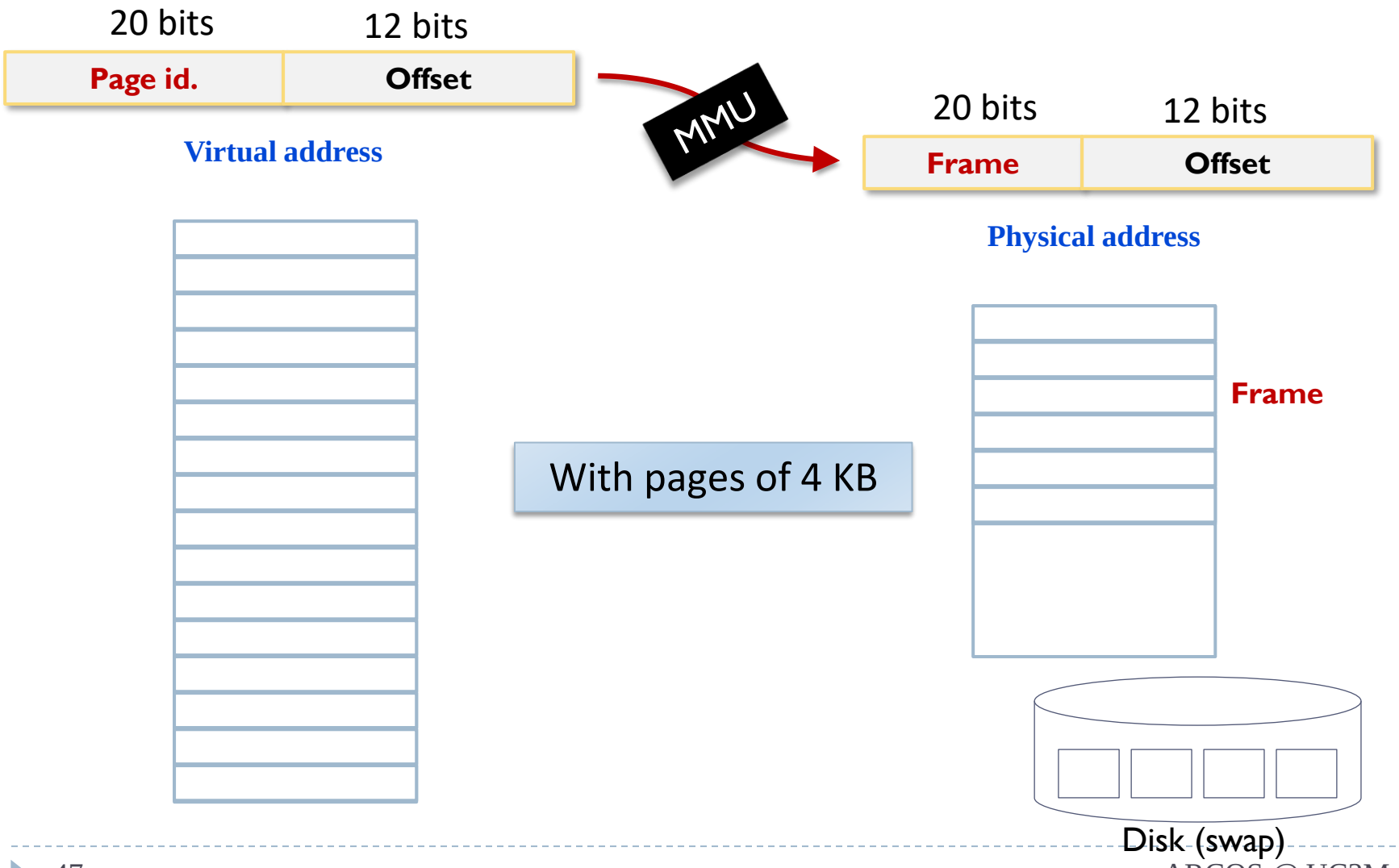


- n = m + p
- Addressable memory: 2^n bytes
- Page size: 2^p bytes
- Maximum number of pages: 2^m

Example


32 bits
0110100010010 10
Virtual address
MMU
32 bits
111110010000 0
Physical address
Disk (swap)

Example


20 bits
12 bits
Page id.
Offset
Virtual address
MMU
20 bits
12 bits
Frame
Offset
Physical address
With pages of 4 KB
Frame
Disk (swap)

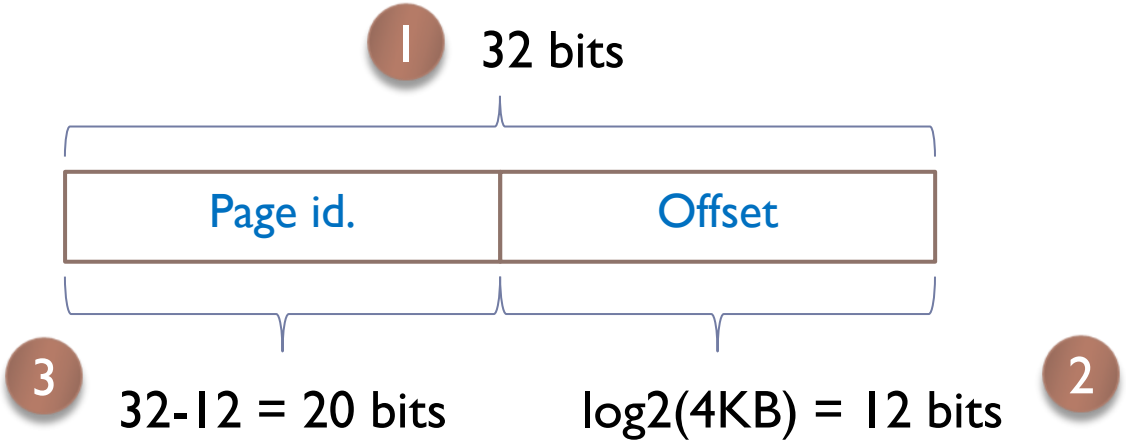
Exercise

- A 32 bit computer has a memory of 512 MB and pages of 4 KB

- Answer:
  a) Indicate the format of a virtual address and the number of page frames

Solution

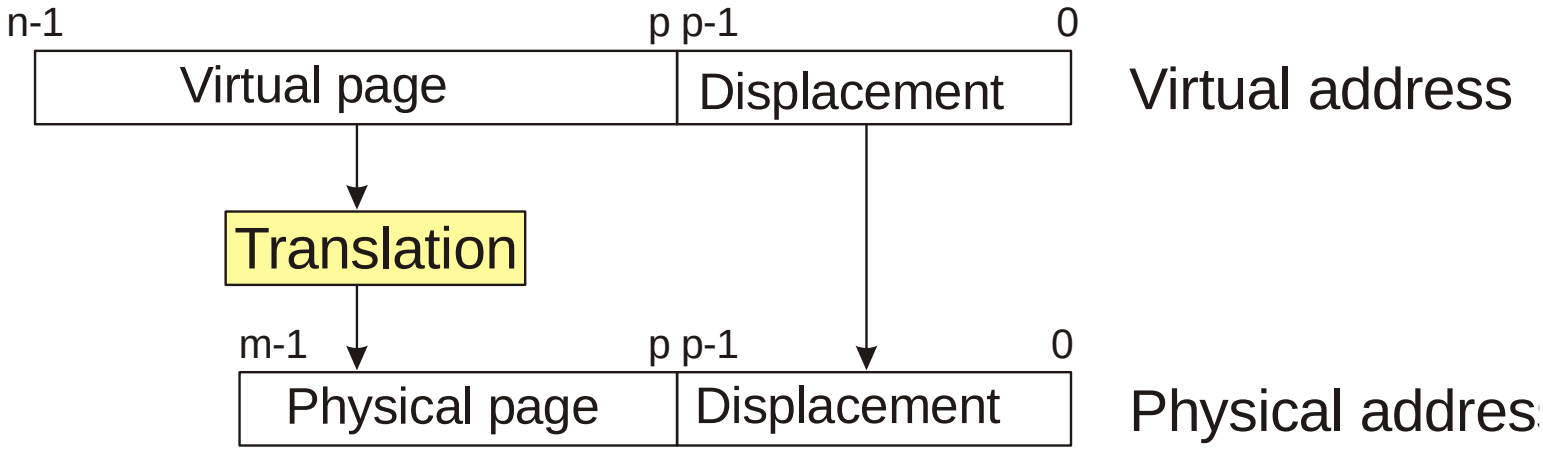
- Virtual address format:



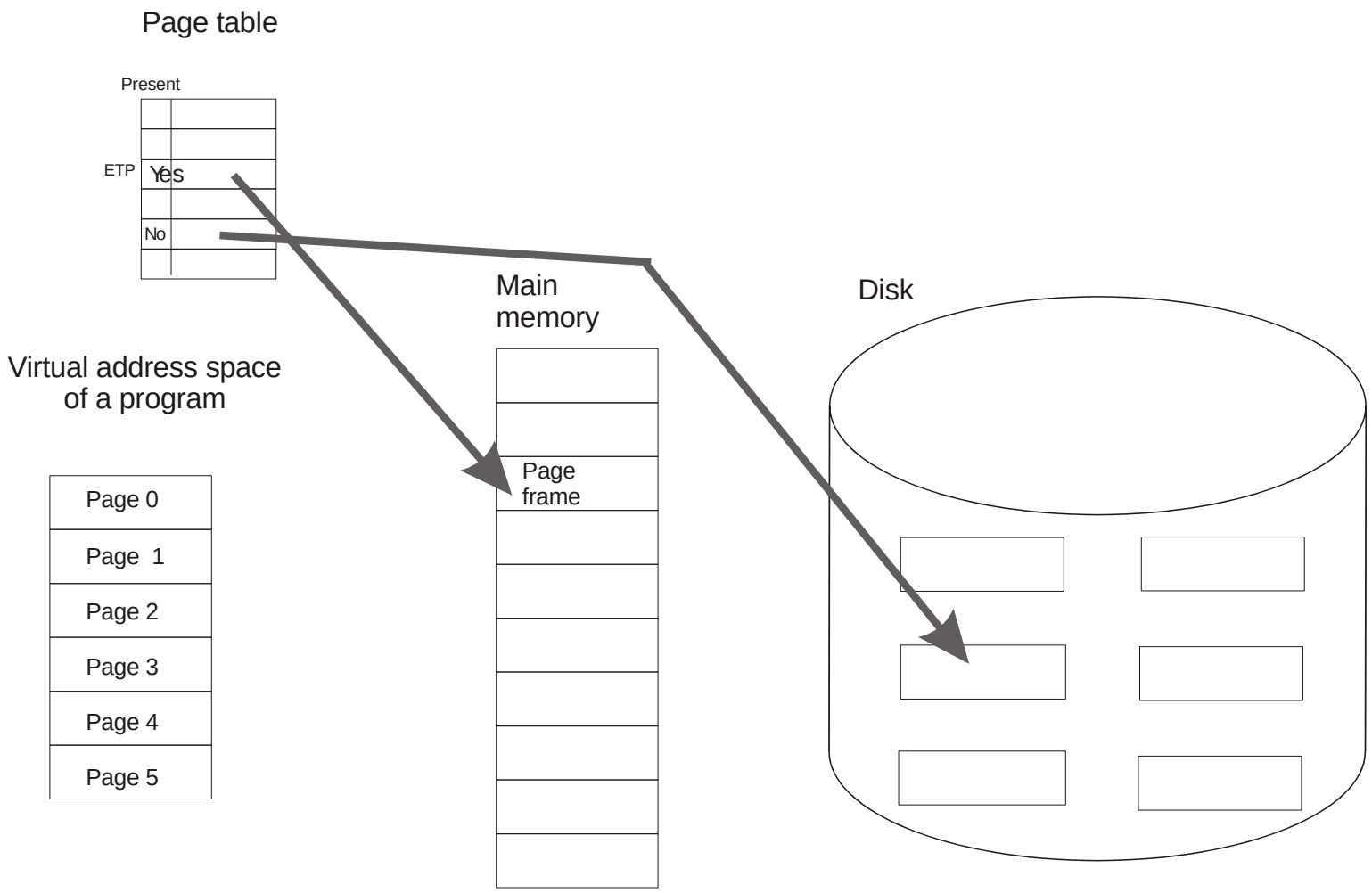
Number of page frames

$$\frac{\text{Main memory size}}{\text{Page size}} = \frac{512\ \text{MB}}{4\ \text{KB}} = \frac{512 * 2^{20}}{4 * 2^{10}} = 128 * 2^{10}$$

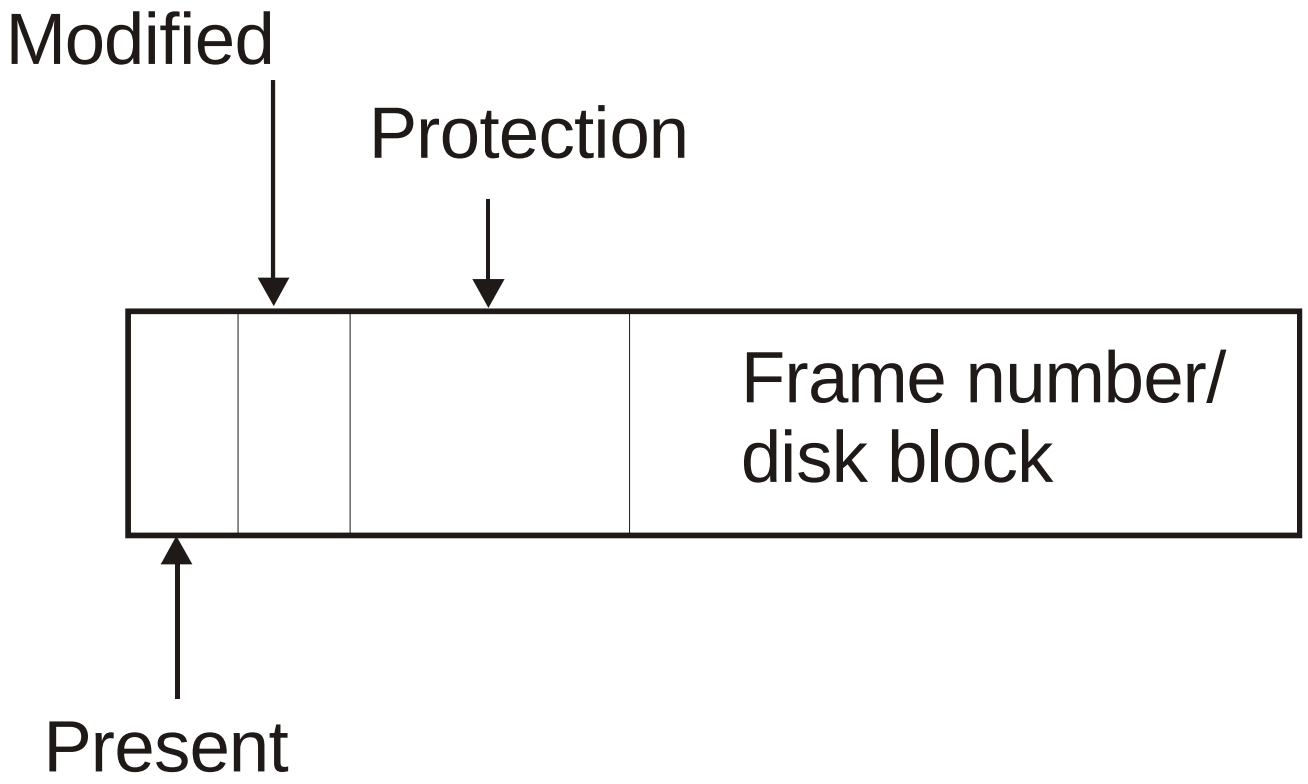
Address translation



Page table


Page table
Present
ETP
Yes
No
Virtual address space
of a program
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Main
memory
Page
frame
Disk

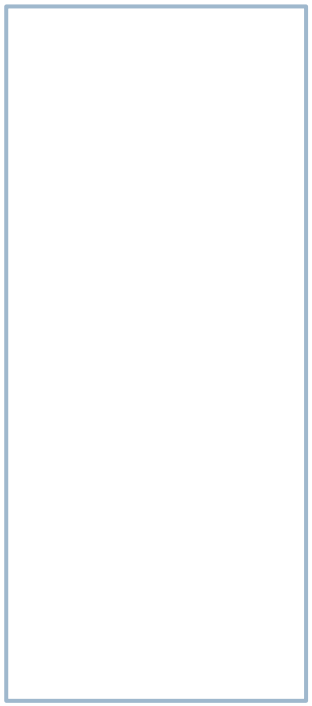
Page table entry



Page table structure

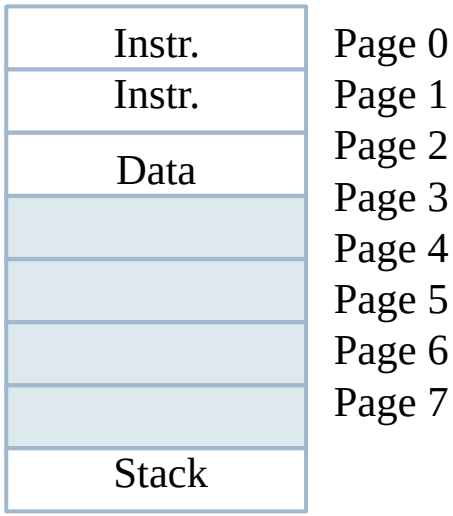
- Operating system creates the page table when a program is going to be executed
- The page table is accessed by the MMU in the translation process
- The page table is modified by the operating system when a page fail occurs

Example



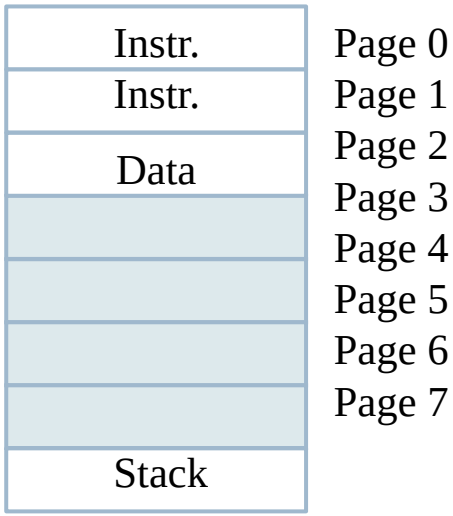
- Pages of 1 KB
- Process of 8 KB
  - Number of pages: 8
- Size of sections:
  - Instructions: 1.5 KB
  - Data: 1 KB
  - Stack: 0.2 KB

Example



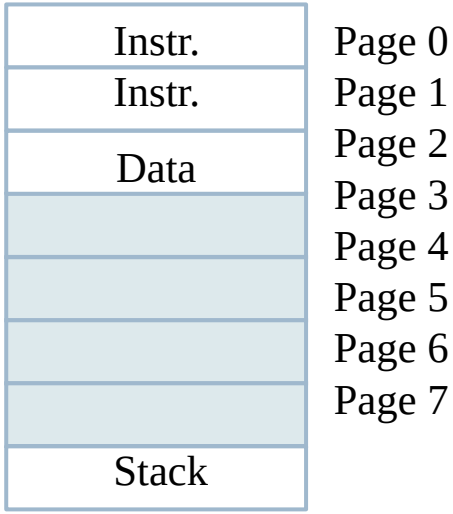
- Pages of 1 KB
- Process of 8 KB
  - Number of pages: 8
- Size of sections:
  - Instructions: 1.5 KB -> 2 pages
  - Data: 1 KB -> 1 page
  - Stack: 0.2 KB -> 1 page

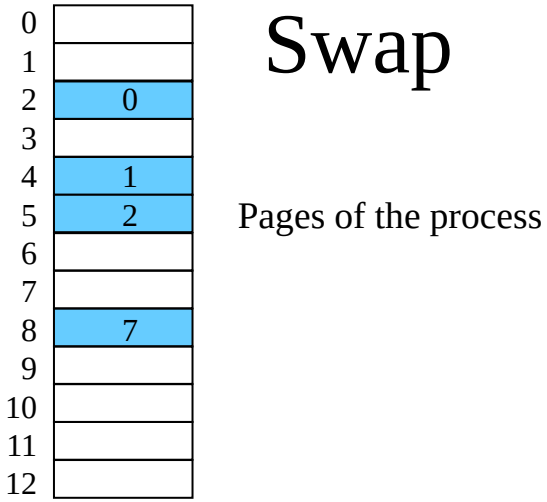
Example



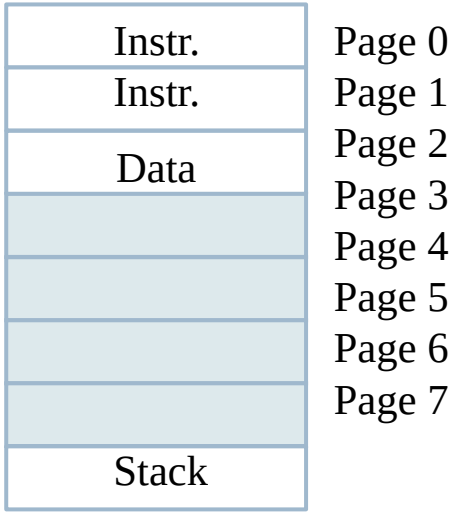
- Init virtual address (VA): 0
- Final virtual address: 8191
- Pags. 3, 4, 5 and 6 are not assigned to the program at the beginning

Example
Process image initially in disk



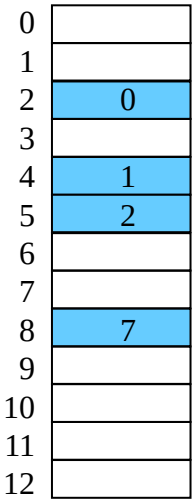


Example
OS creates the page table



	P	M	frame/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

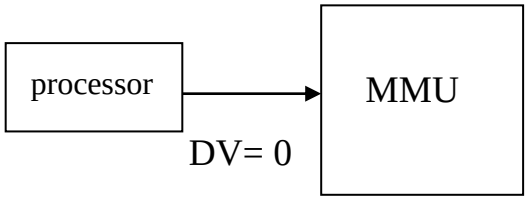
All pages in swap at the begining



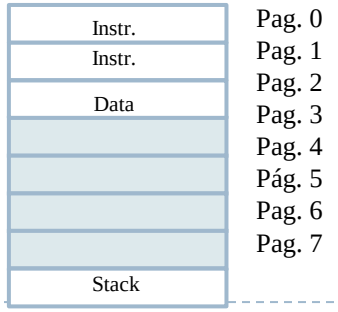
Swap

Pages of the process

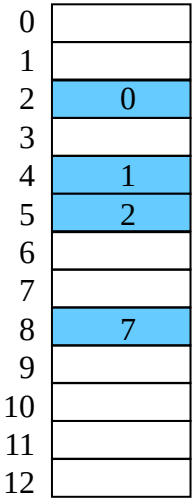
Example
Access to VA 0



	P	M	frame/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8



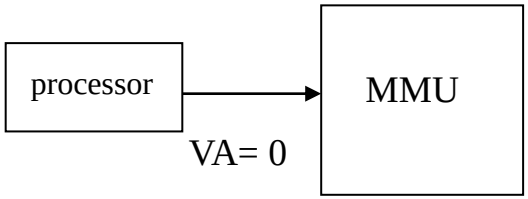
Pag. 0
Pag. 1
Pag. 2
Pag. 3
Pag. 4
Pág. 5
Pag. 6
Pag. 7



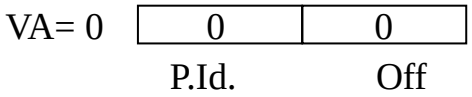
Swap

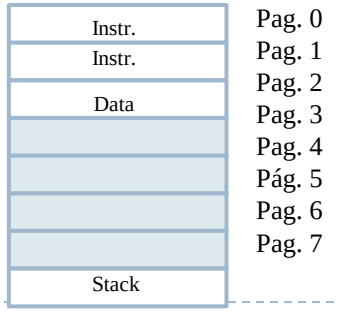
Pages of the process

Example
Access to VA 0

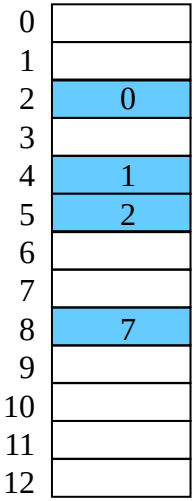


	P	M	frame/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8





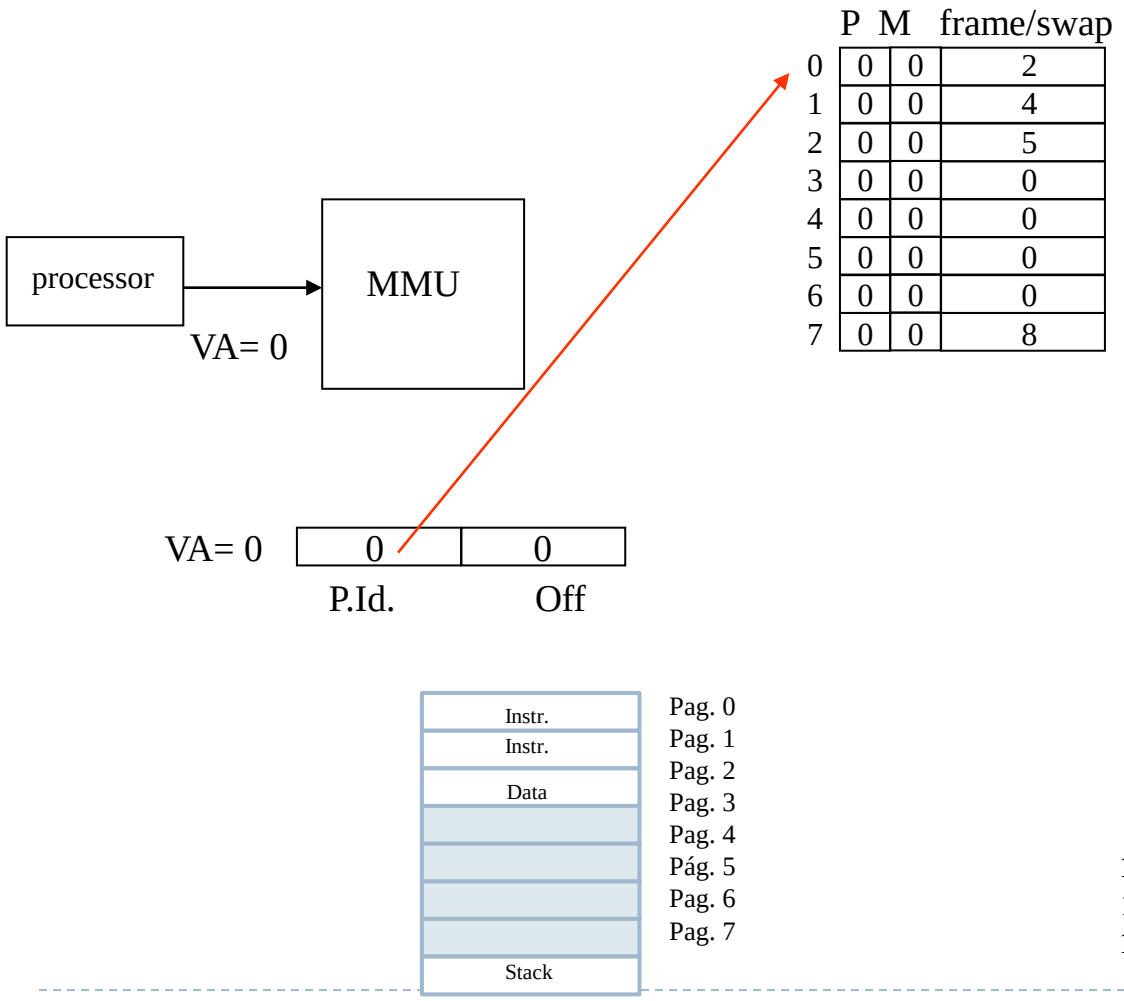
Pag. 0
Pag. 1
Pag. 2
Pag. 3
Pag. 4
Pág. 5
Pag. 6
Pag. 7



Swap

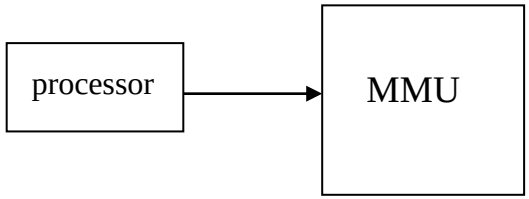
Pages of the process

Example
Access to VA 0

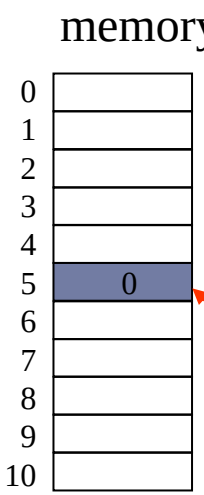


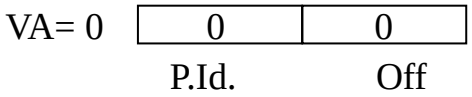
Page fault
Page 0 is not in memory

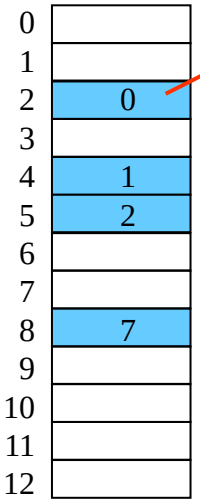
Example handling the page fault



	P	M	frame/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8





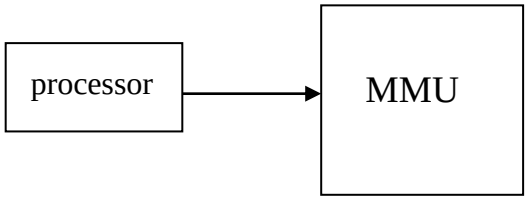


Swap

Pages of the process

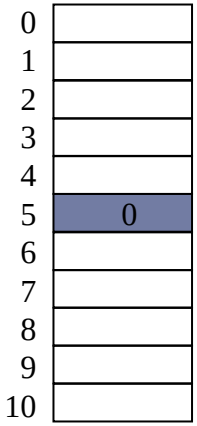
OS reserves a free page frame in memory (5) and copies the block 2 in the frame 5

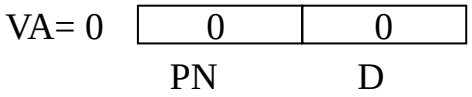
Example handling the page fault



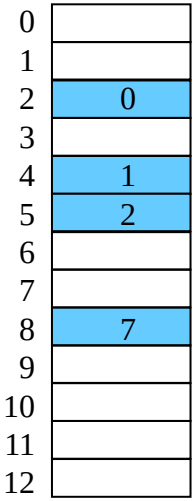
	P	M	frame/swap
0	1	0	5
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

Memory





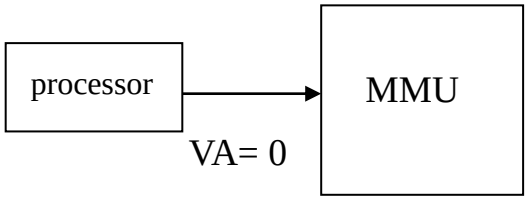
OS updates the page table



Swap

Pages of the process

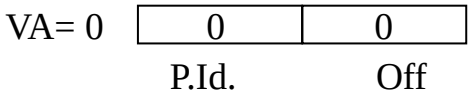
Example
Restoring the process



	P	M	frame/swap
0	1	0	5
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

Memory

0	
1	
2	
3	
4	
5	0
6	
7	
8	
9	
10	



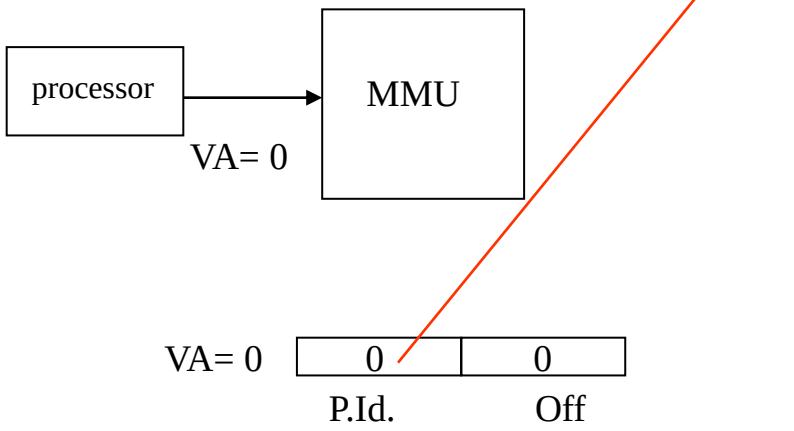
VA 0 is generated again

0	
1	
2	0
3	
4	1
5	2
6	
7	
8	7
9	
10	
11	
12	

Swap

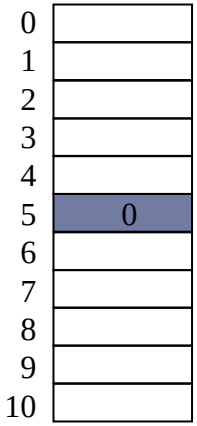
Pages of the process

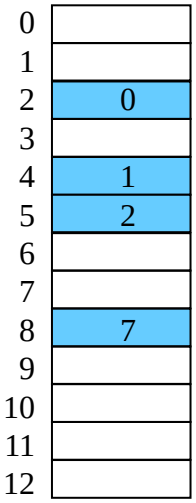
Example
Restoring the process



	P	M	frame/swap
0	1	0	5
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

Memory



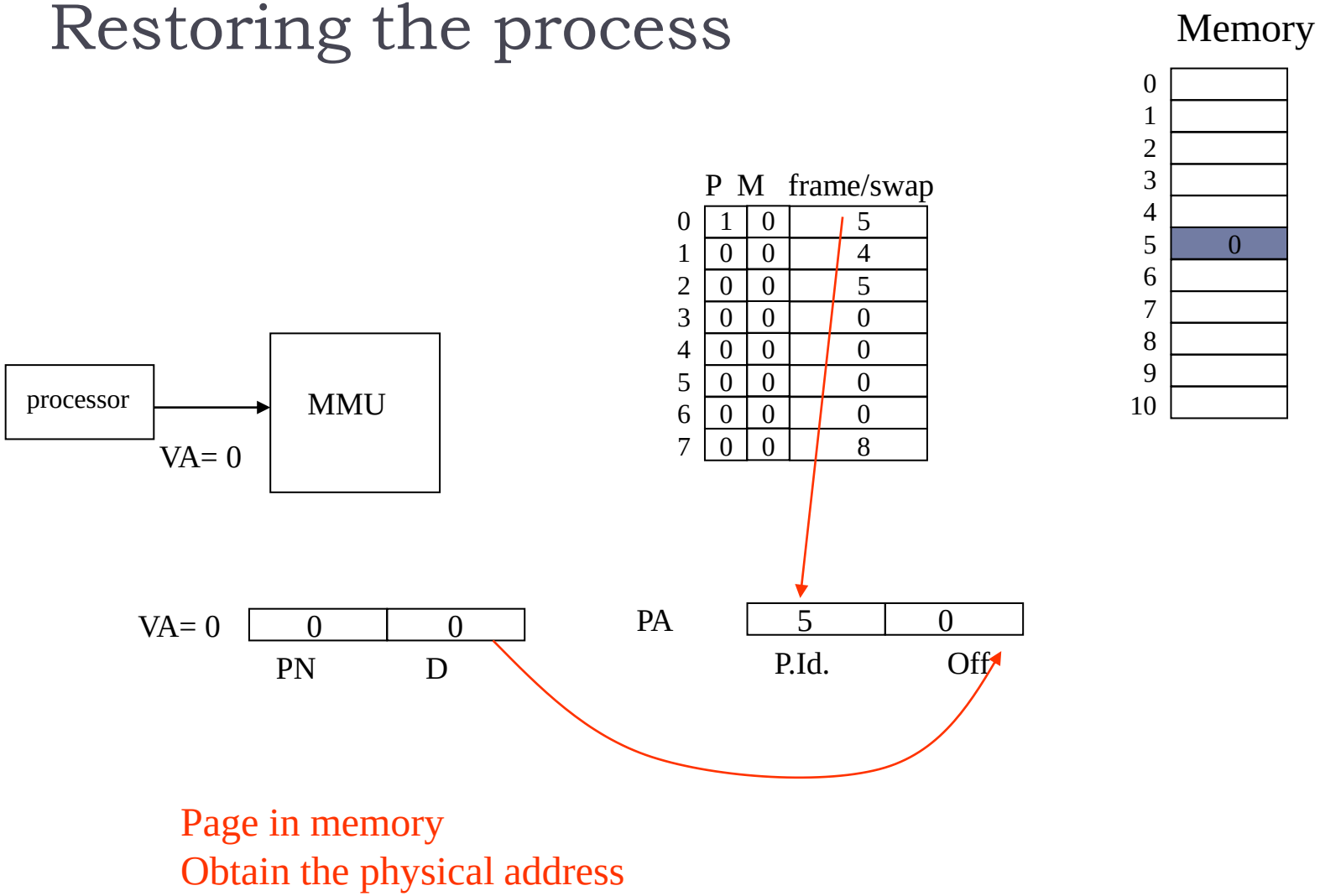


Swap

Pages of the process

VA 0 is generated again

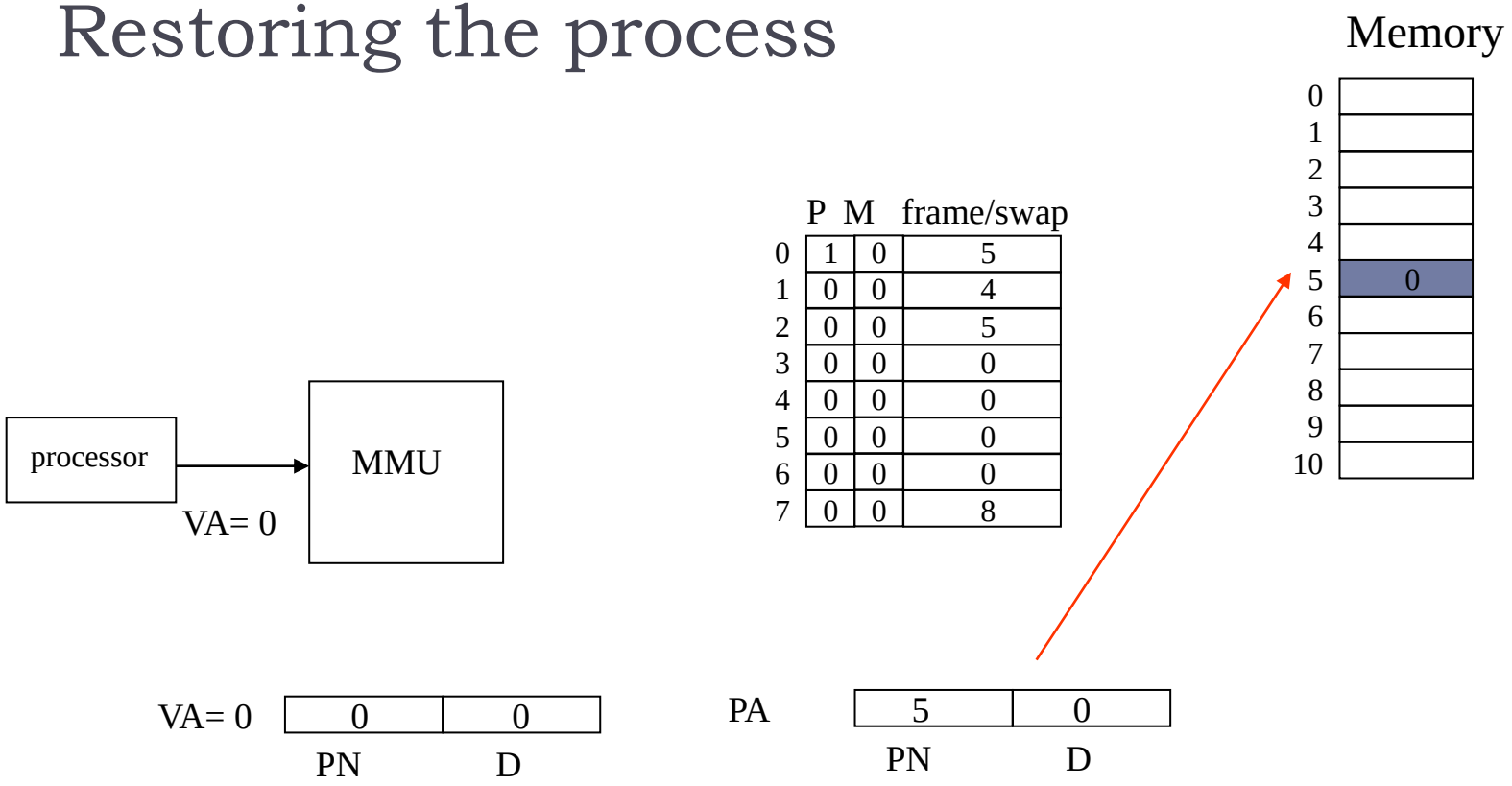
Example
Restoring the process



Page in memory

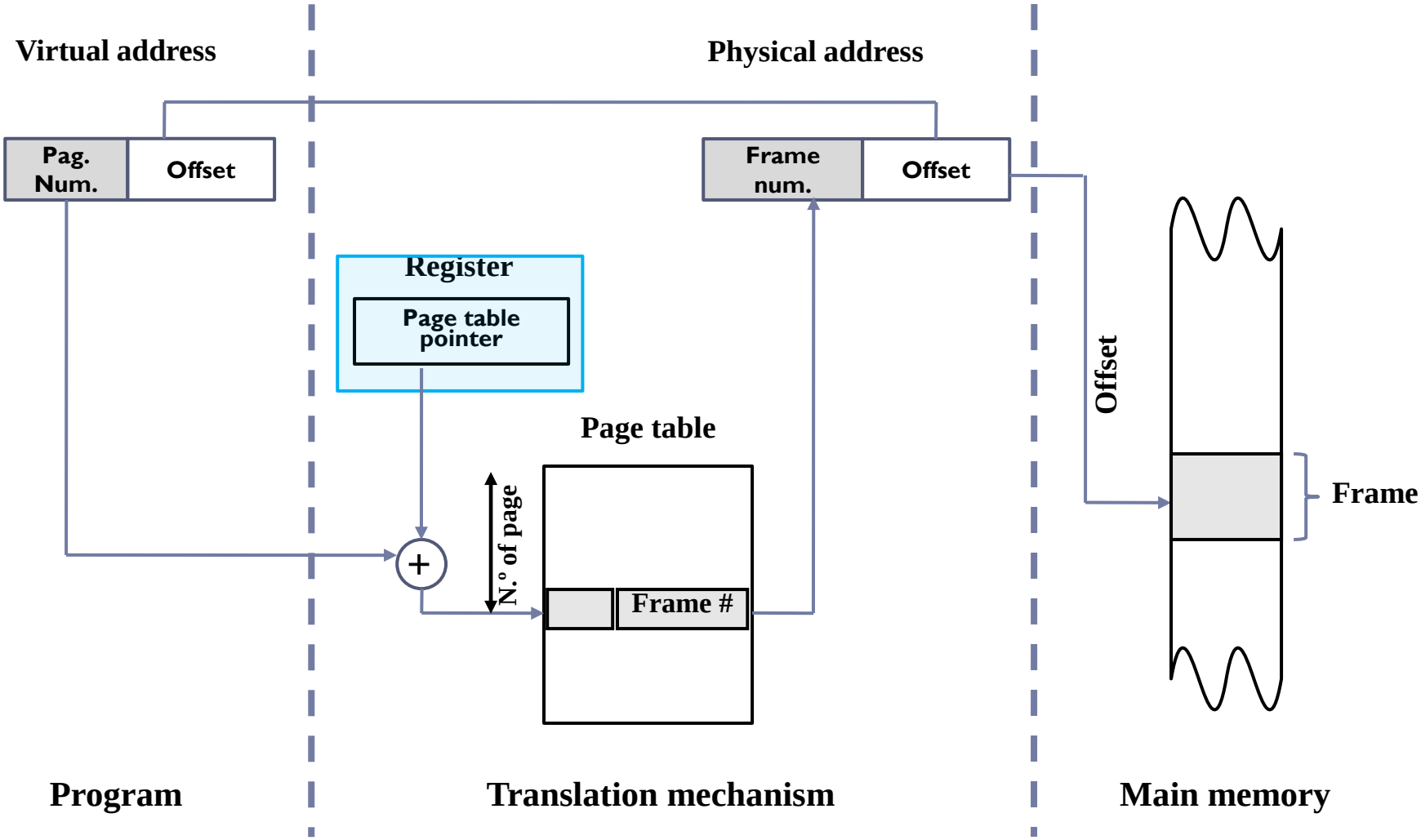
Obtain the physical address

Example
Restoring the process

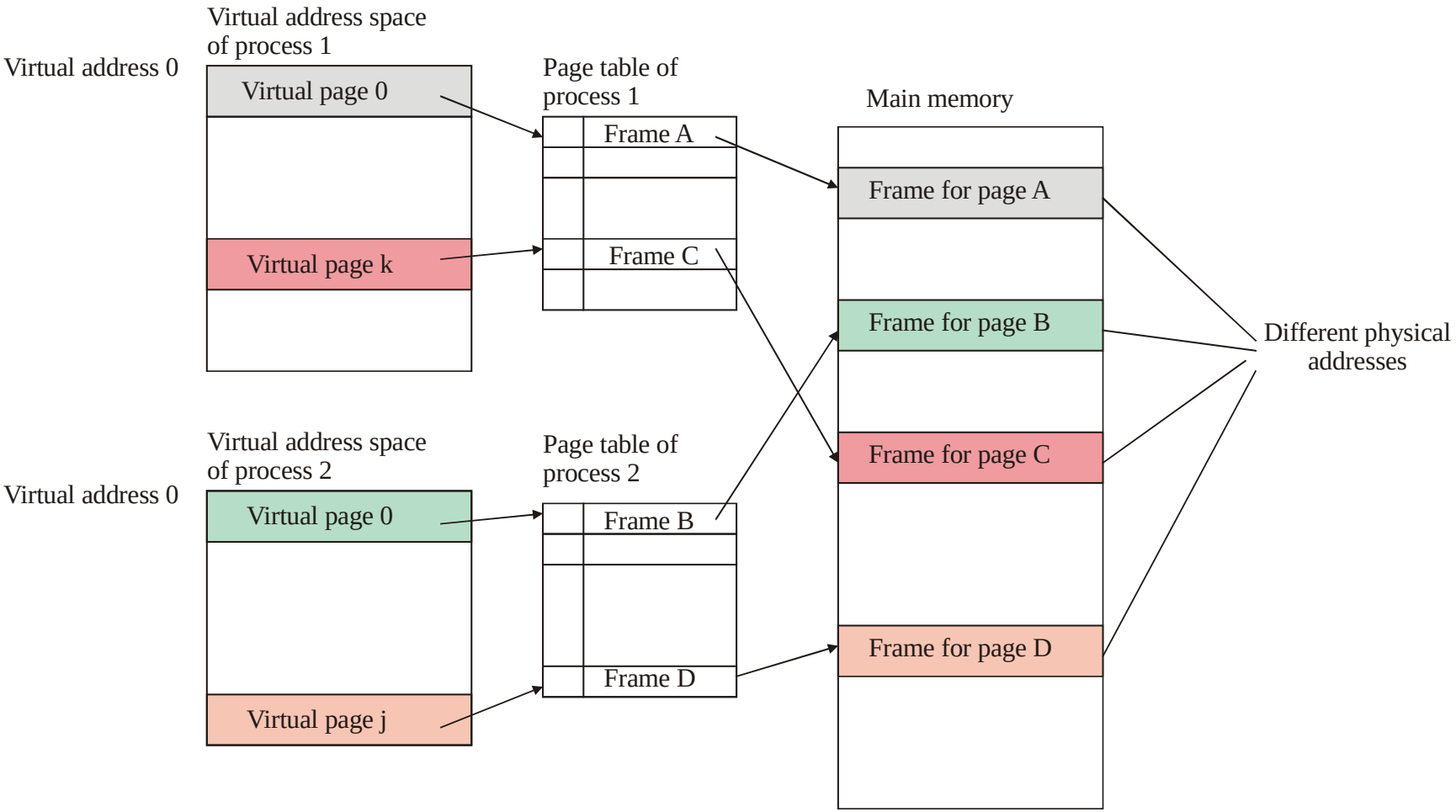


Access to memory

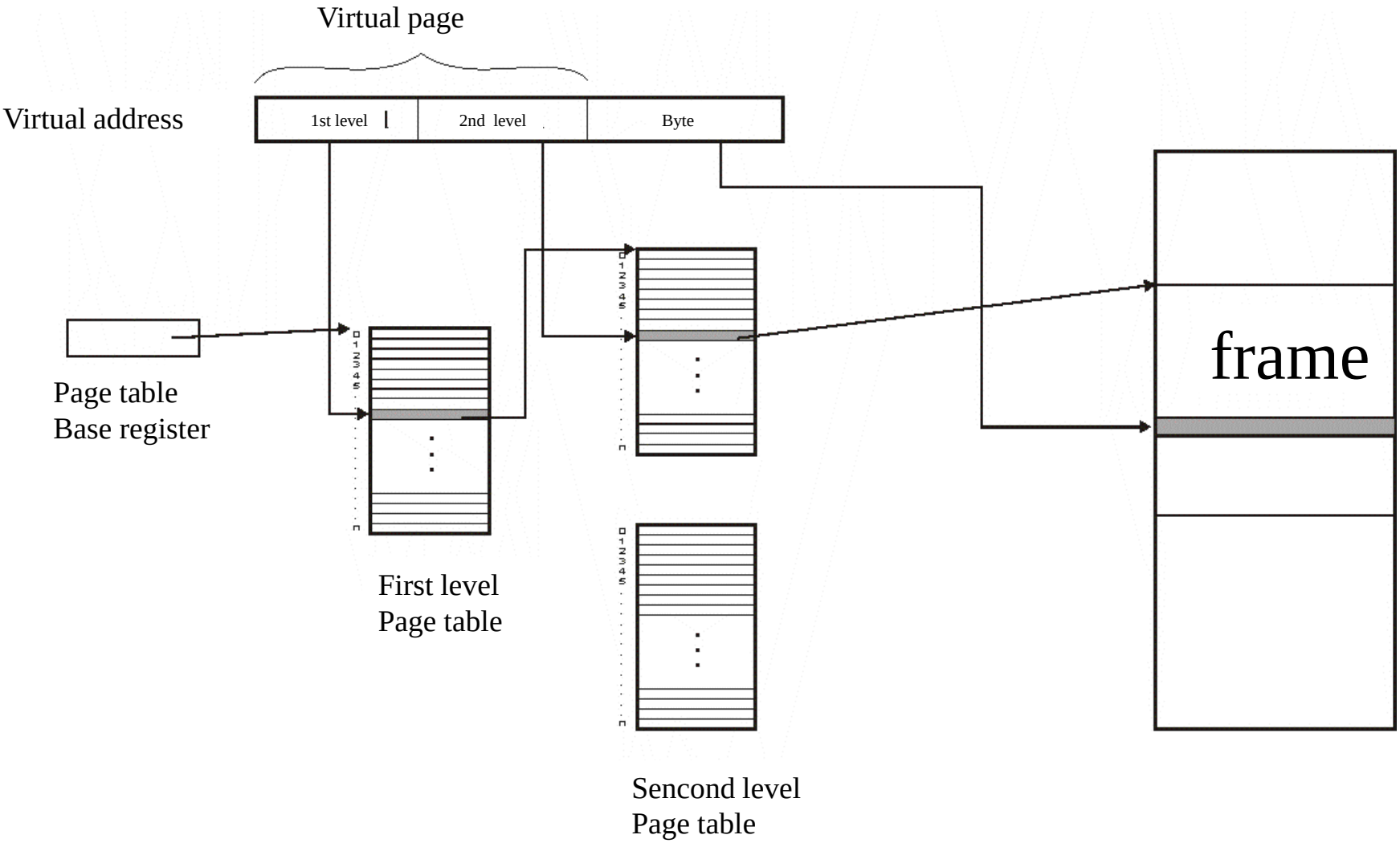
Translation



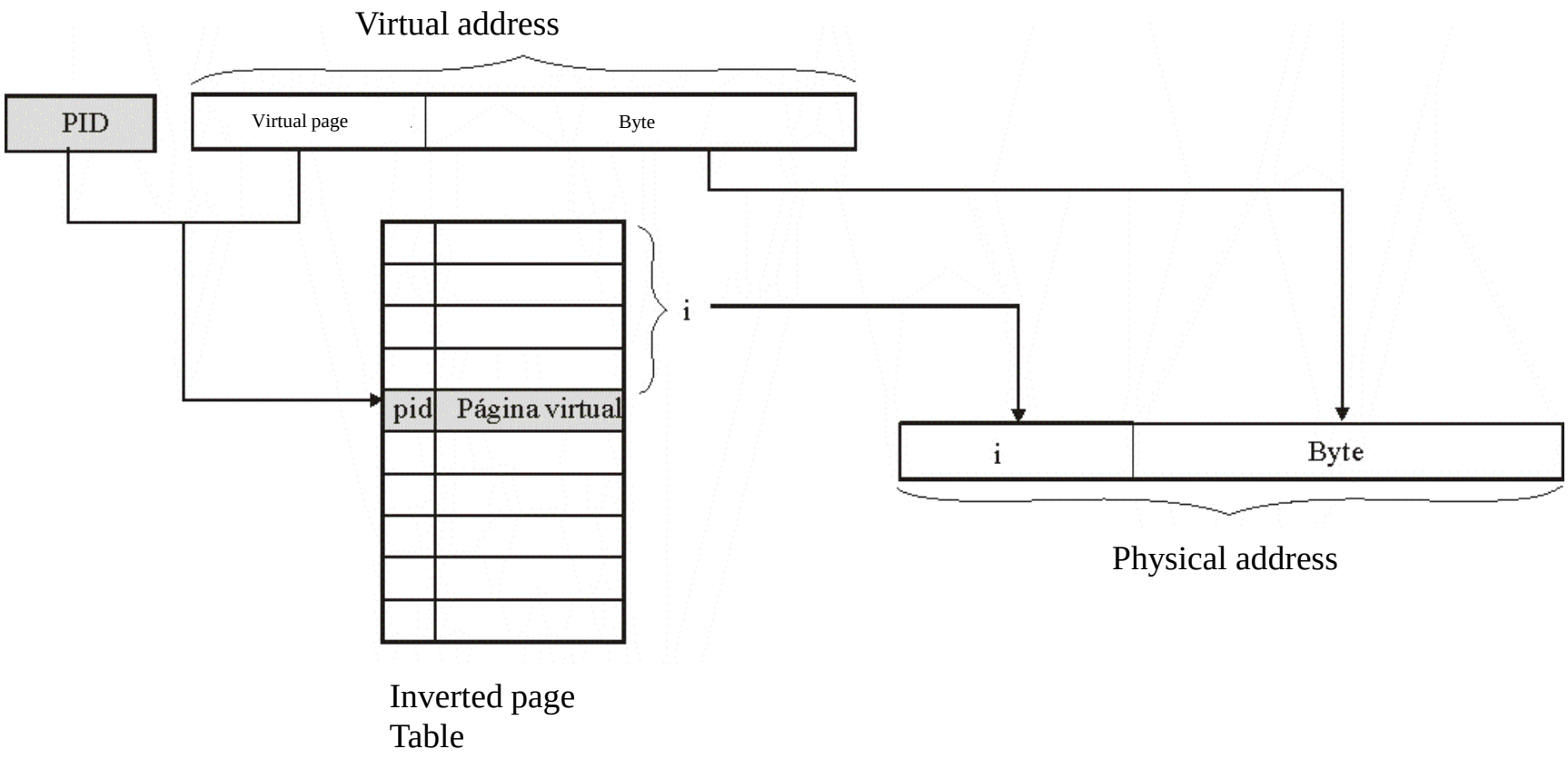
Memory protection



Two-level page table



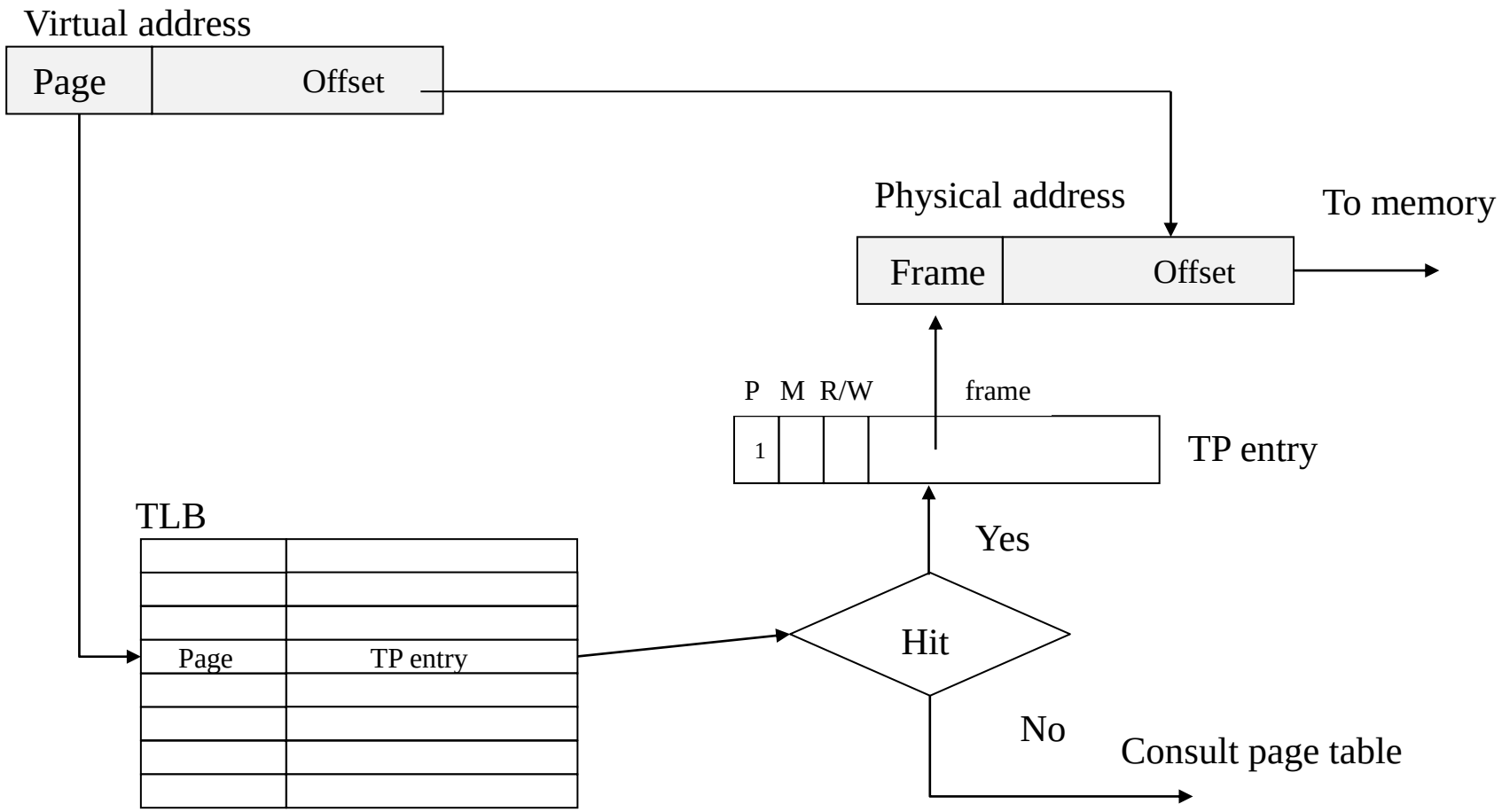
Inverted page table



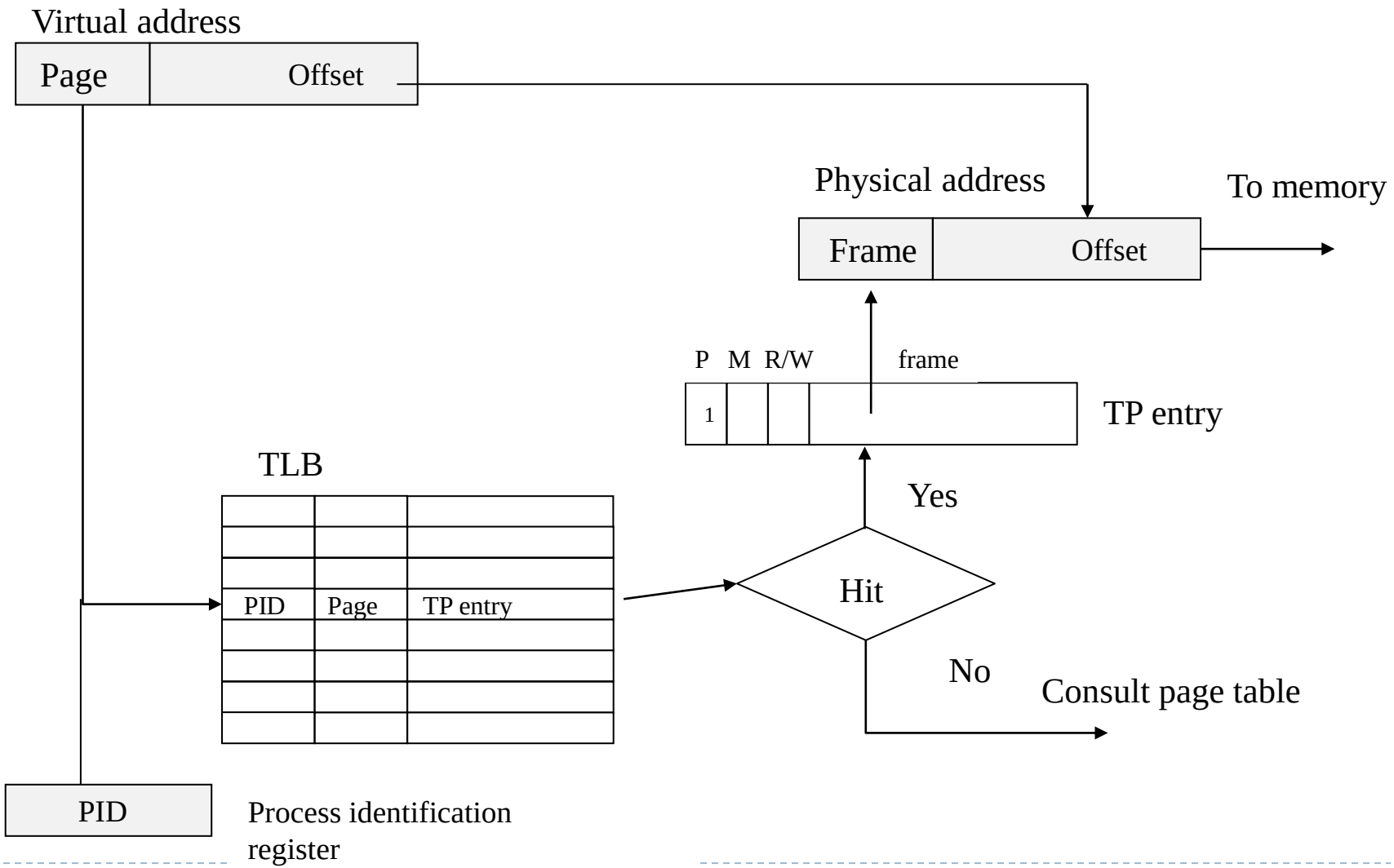
TLB (Translation Lookaside Buffer)

- With virtual memory, two memory accesses are needed for each memory reference:
  - One access to the page table
  - One access to the page in memory
- TLB is used to optimize the memory access:
  - Table with reduced access time located in the MMU
  - Each entry has the page number and the corresponding page table entry
  - In case of hit, the page table is not accessed
- Two types:
  - TLB with process identification
  - TLB without process identification

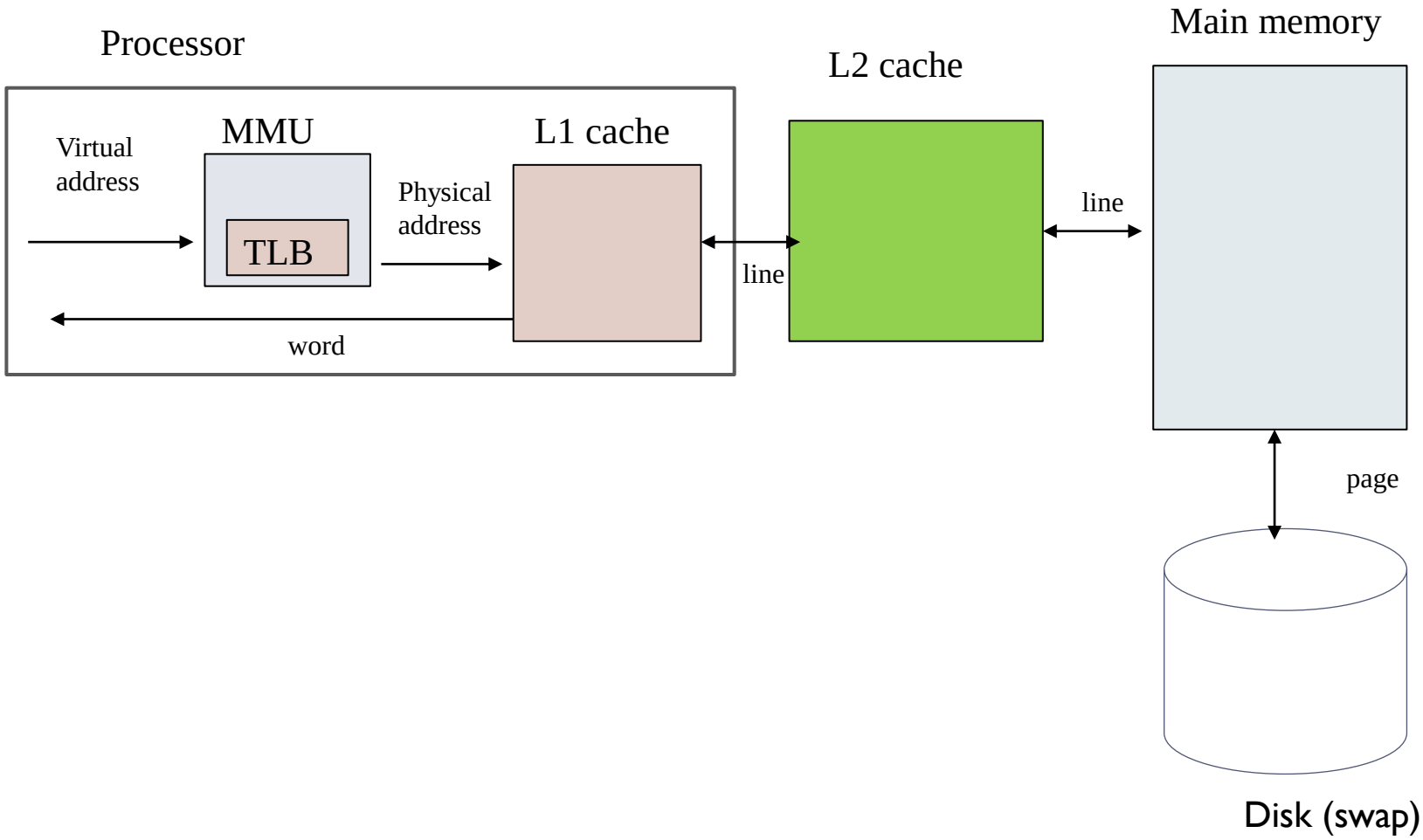
TLB without process identification



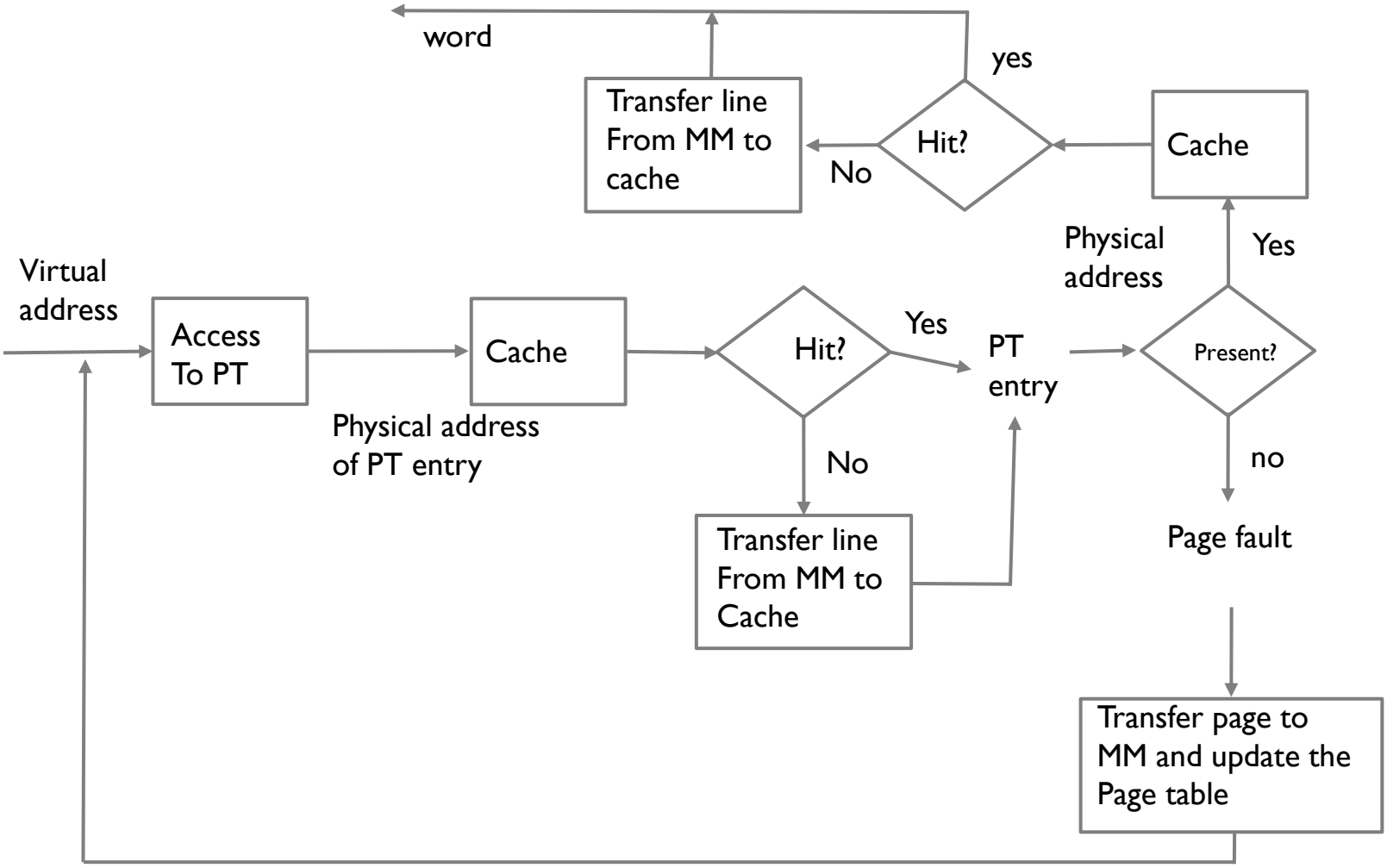
TLB without process identification



Virtual memory and cache memory



Read access with cache and virtual memory



ARCOS Group

uc3m | Universidad Carlos III de Madrid

Lesson 5 (III)
Memory hierarchy

Computer Structure

Bachelor in Computer Science and Engineering