ARCOS Group

uc3m | Universidad Carlos III de Madrid

L5: Memory hierarchy (3)

Computer Structure

Bachelor in Computer Science and Engineering
Bachelor in Applied Mathematics and Computing
Dual Bachelor in Computer Science and Engineering and Business Administration

Contents

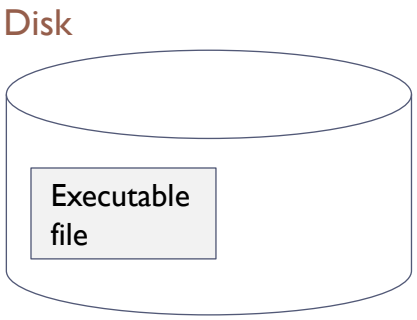
Cache and virtual memory

advance

Cache	Virtual memory
Accelerate access (+ faster access)	Increase addressable space (+ space)
Transfer by blocks or lines	Transfer per page
Blocks: 32-64 bytes	Pages: 4-8 KiB
Translation: mapping algorithm	Translation: Fully associative
Immediate or deferred writing	Deferred writing

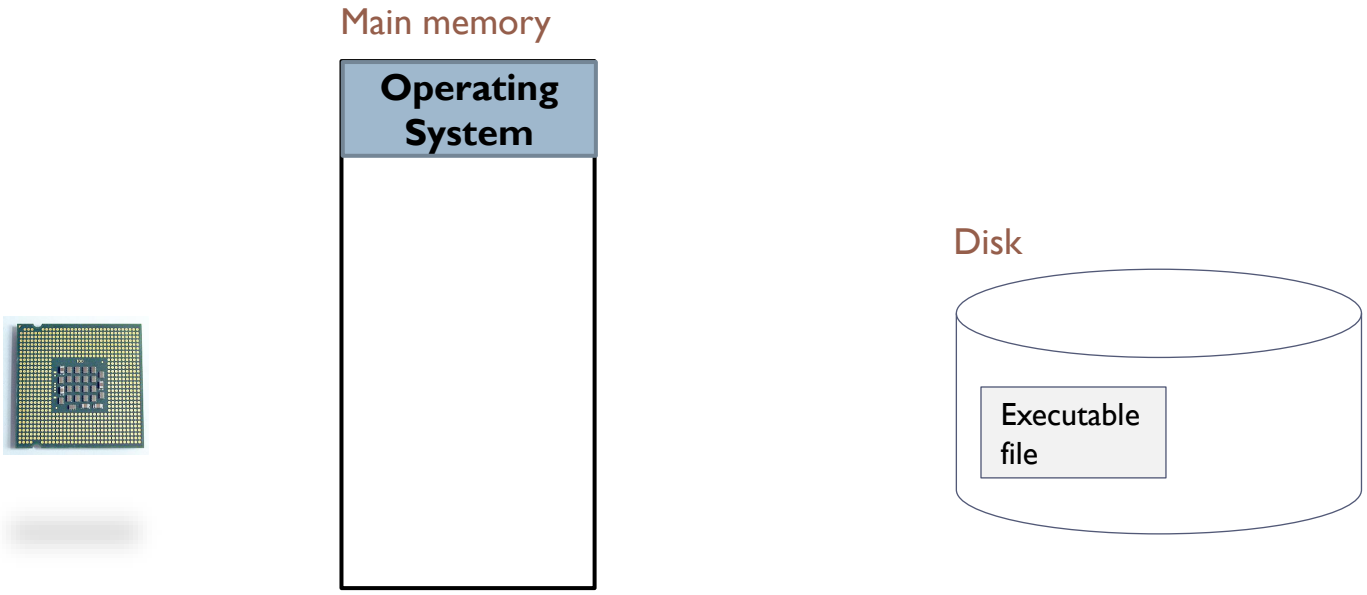
Program and process

- Program: a set of data and instructions arranged in order to perform a specific task or job.



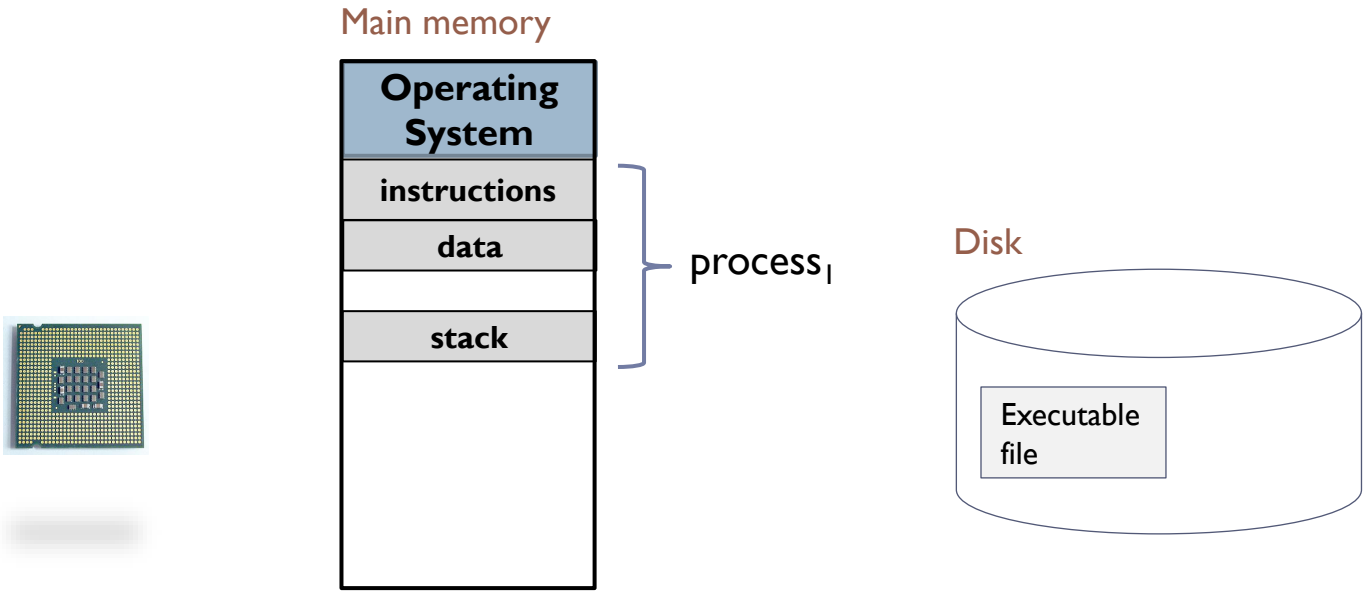
Program and process

- Program: a set of data and instructions arranged in order to perform a specific task or job.
  - For its execution, it must be loaded in memory.



Program and process

- Process: running program.



Program and process

- Process: running program.
  - It is possible to run the same program several times (resulting in several processes)

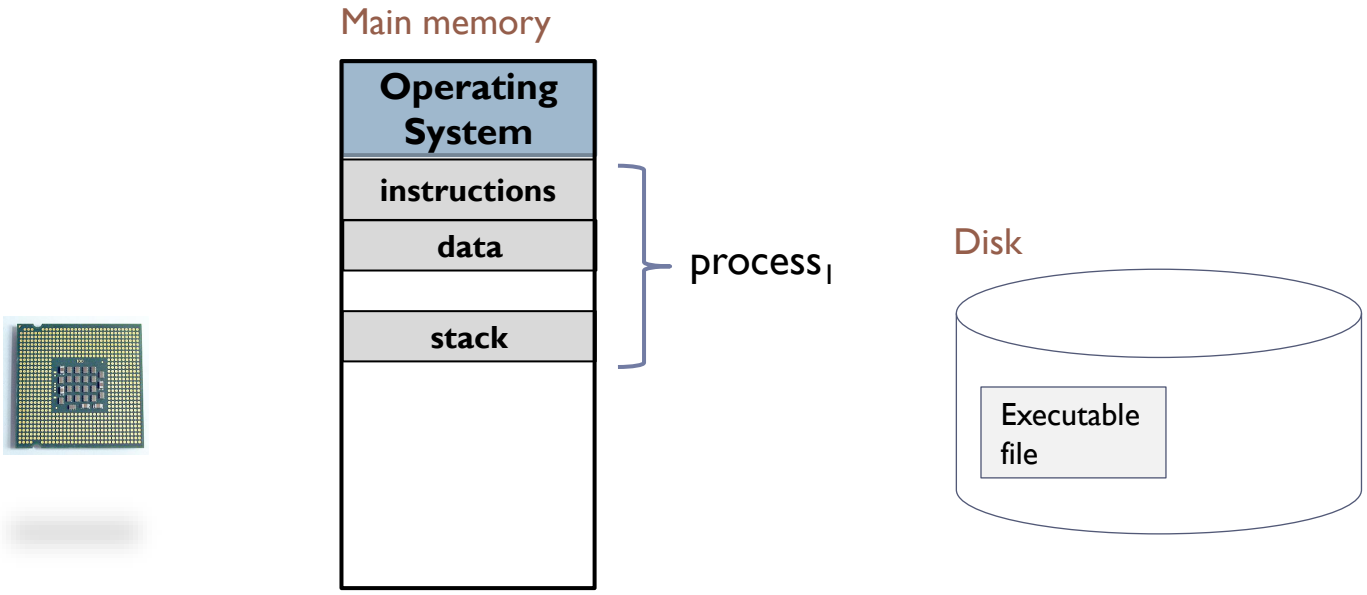
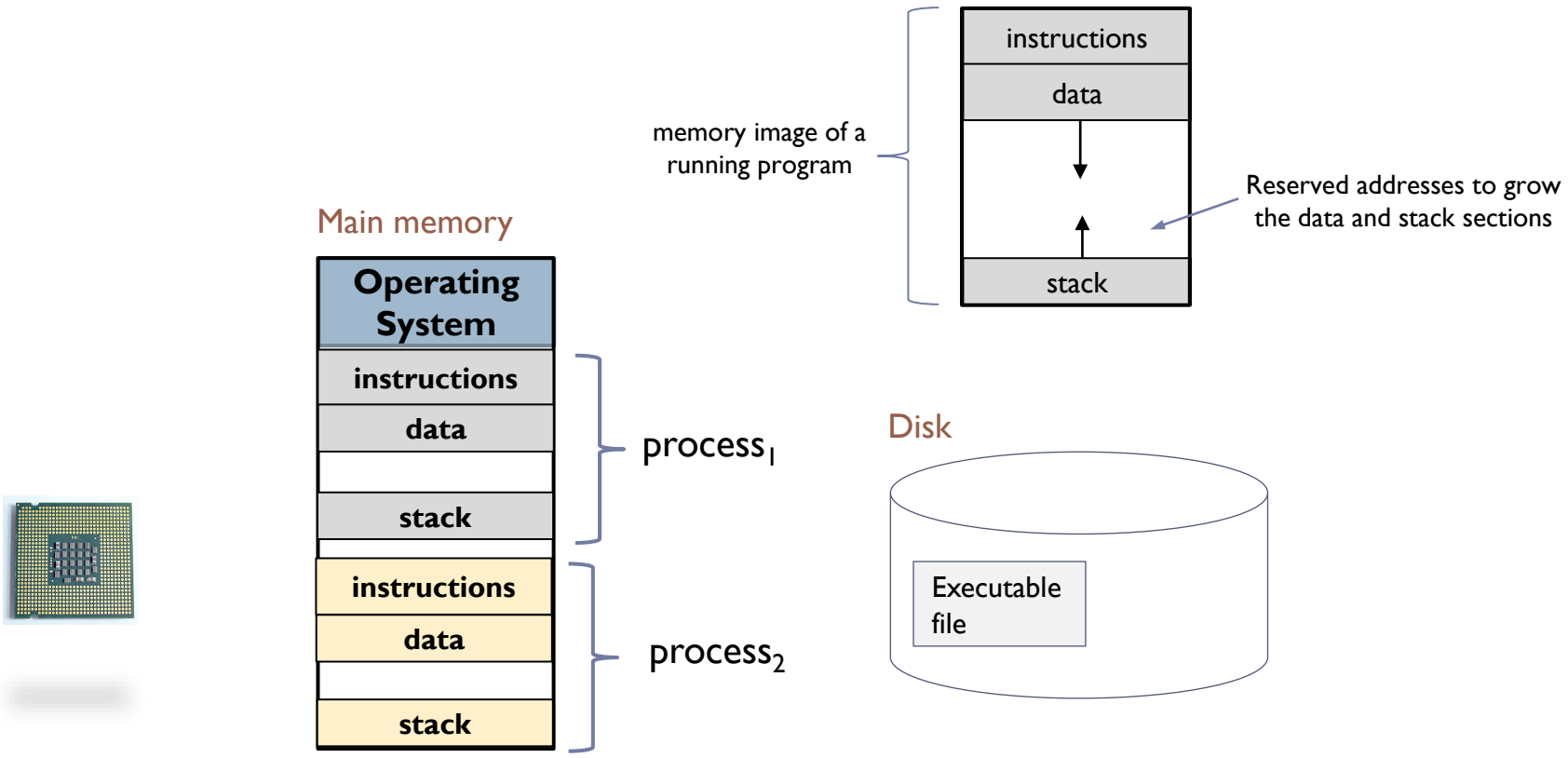


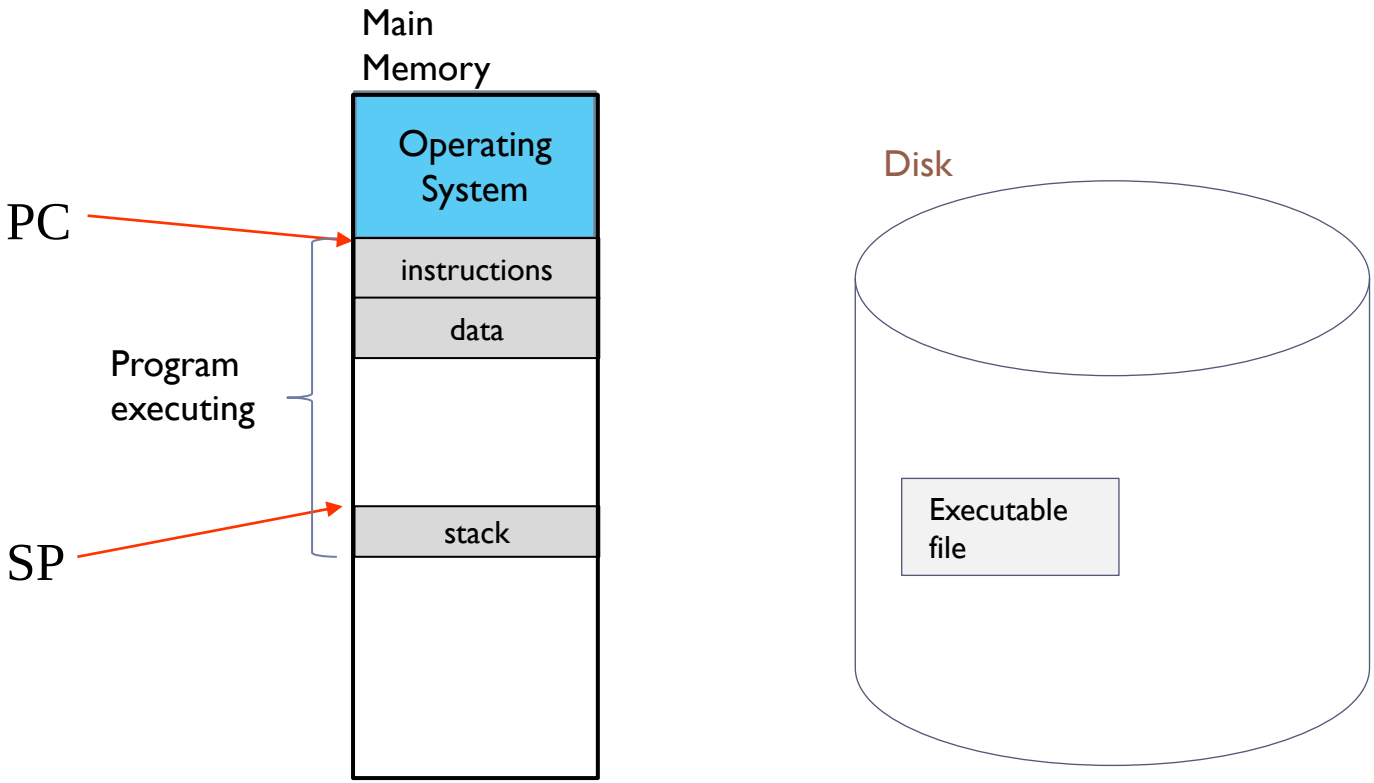
Image of a process

- Memory image: set of memory addresses assigned to the program being executed (and content)



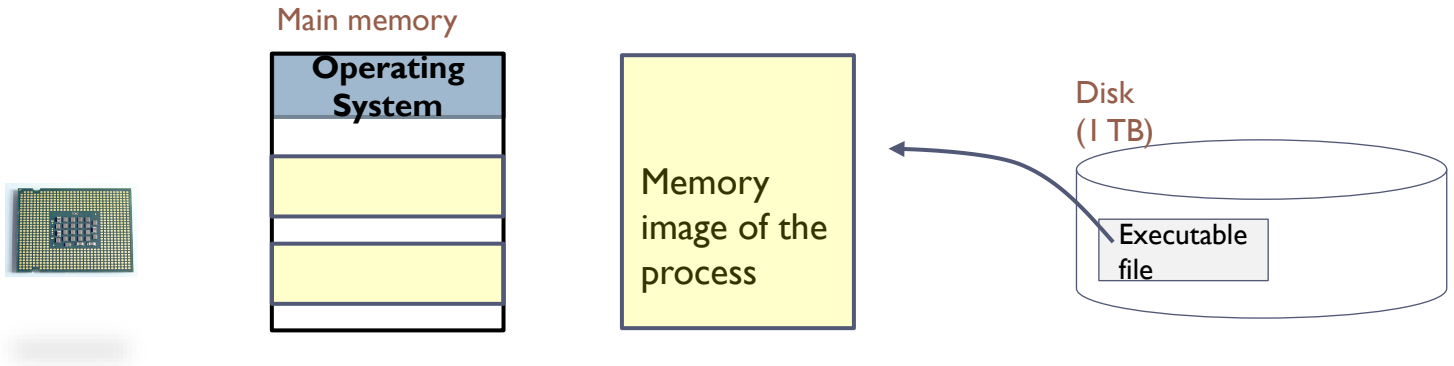
Systems **without** virtual memory

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



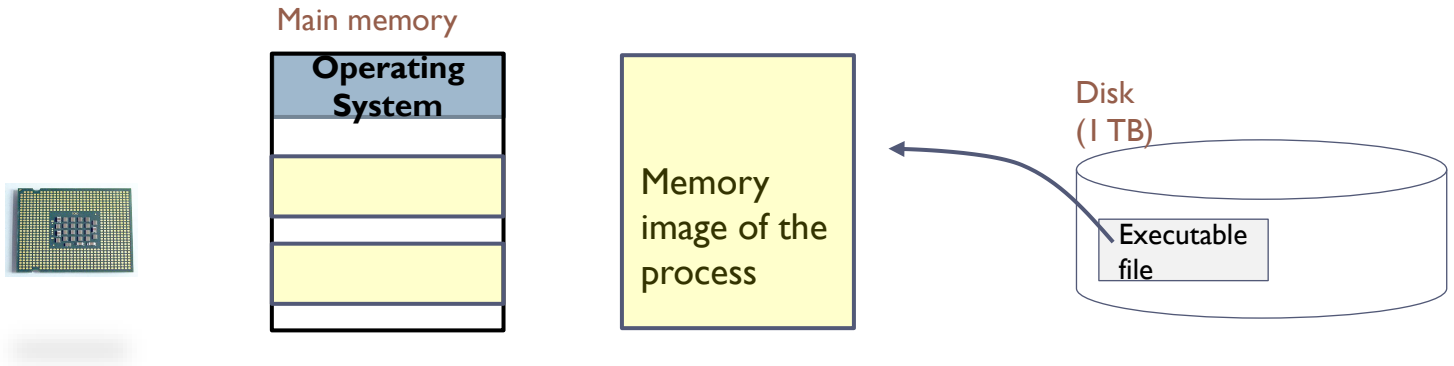
Systems **without** virtual memory

- In systems without virtual memory, the program is completely loaded in memory before the execution.
- Main issues (1/2):
  - **Relocation**: image must be able to be uploaded to any assigned location.
  - **Protection**: prevent access outside the allocated space.



Systems **without** virtual memory

- In systems without virtual memory, the program is completely loaded in memory before the execution.
- Main issues (2/2):
  - 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
        j    bucle
fin:    …
```

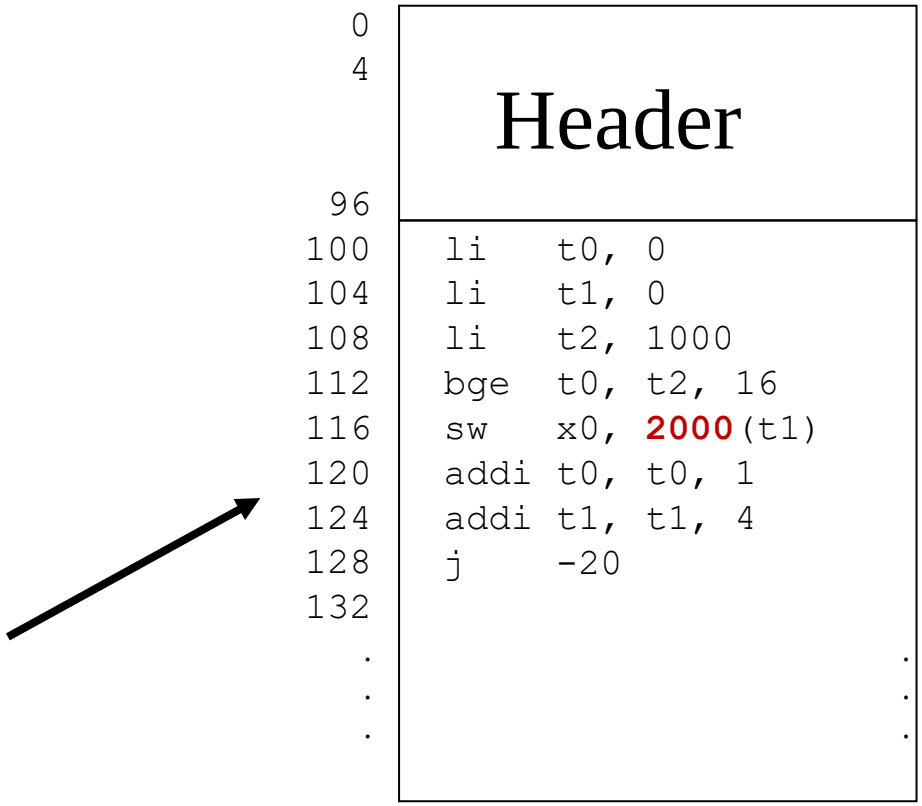
Hypothetical executable file

```
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
        j    bucle
fin:    …
```

executable

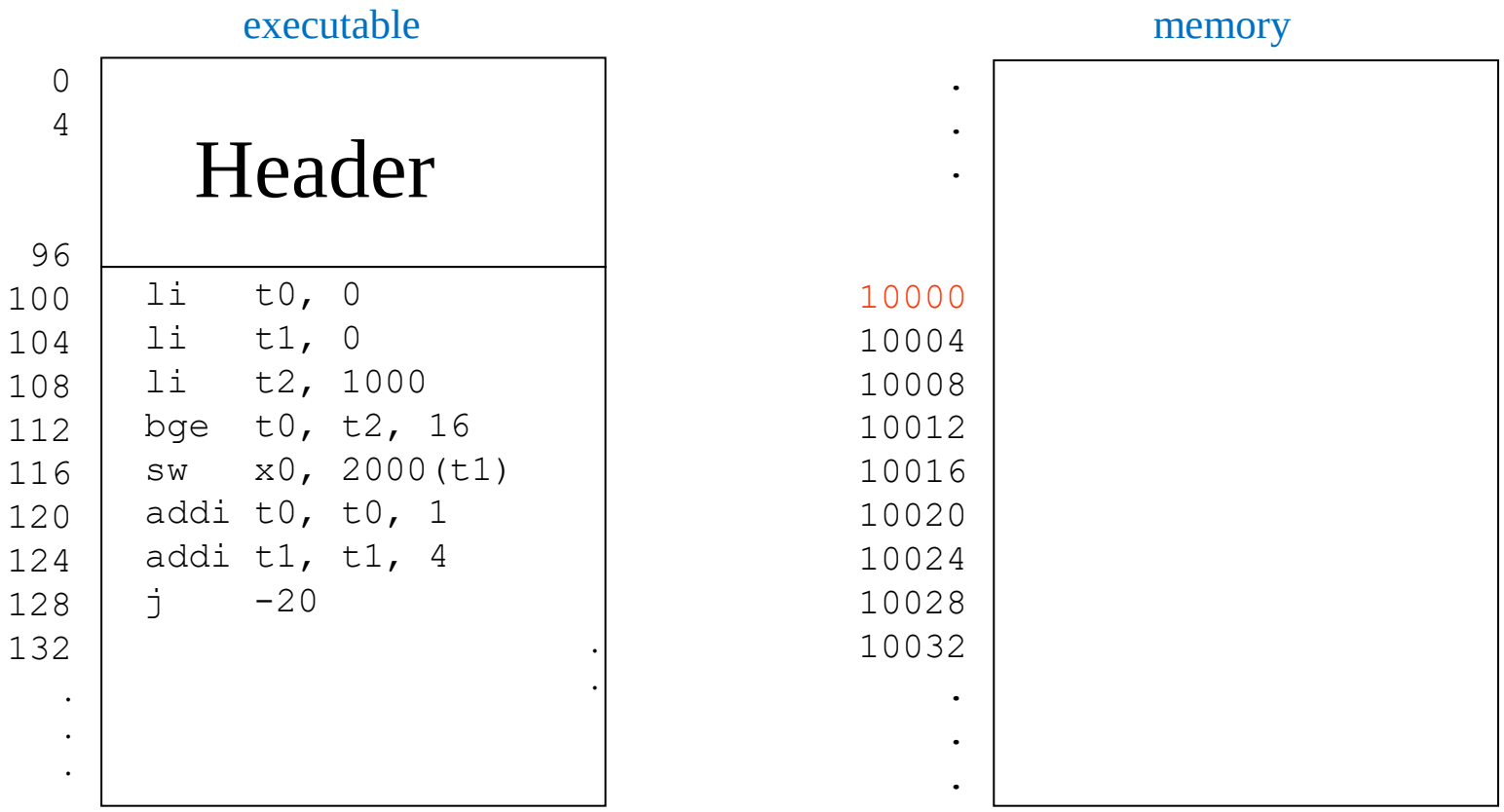


Address 2000 is assigned to **v**

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.

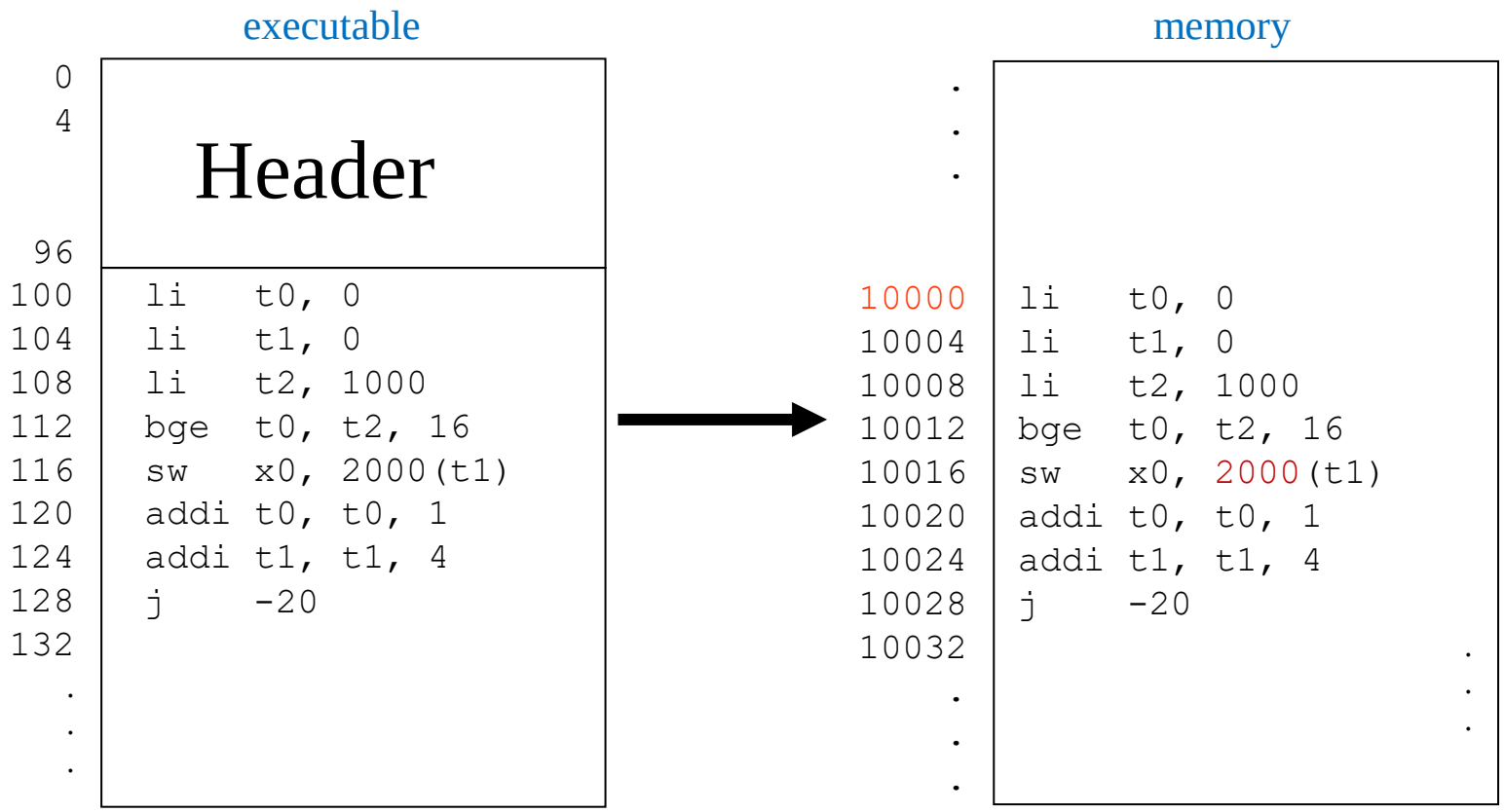


Loading the program in memory

- In the executable file the address 0 is considered as the initial address
  - Logical address
- In memory, the initial 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

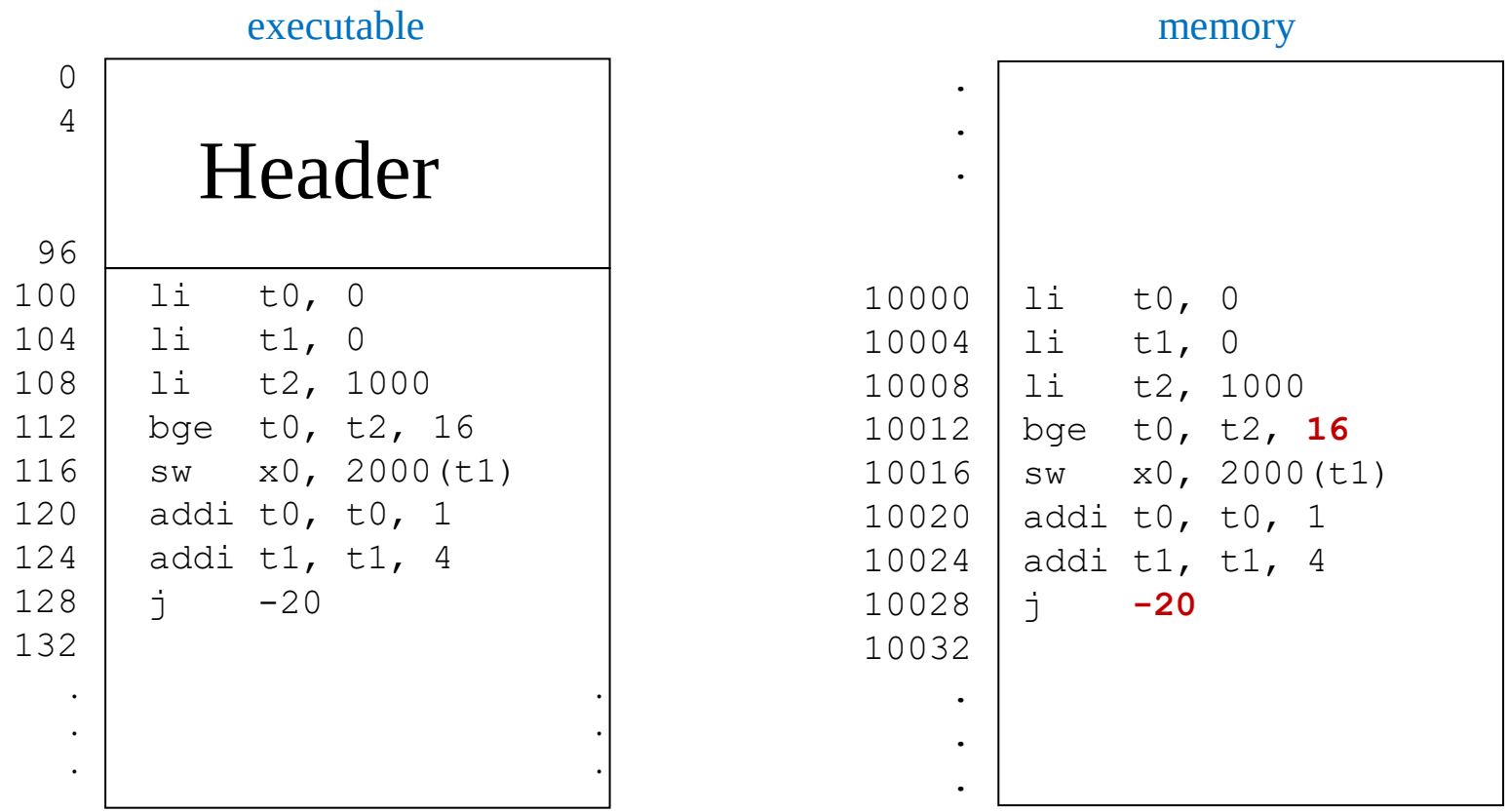
Software relocation

- Occurs in the loading process



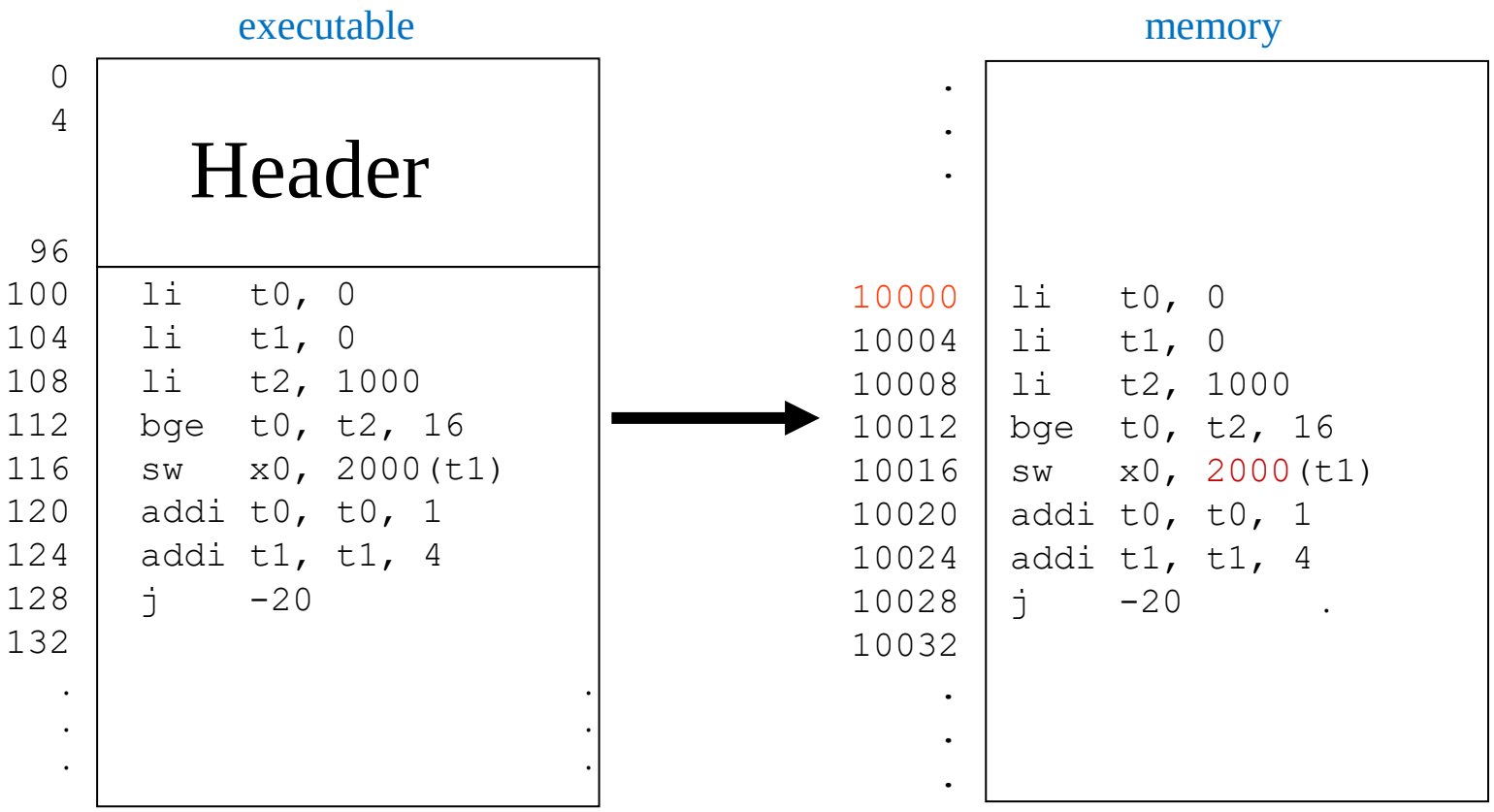
Software relocation

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



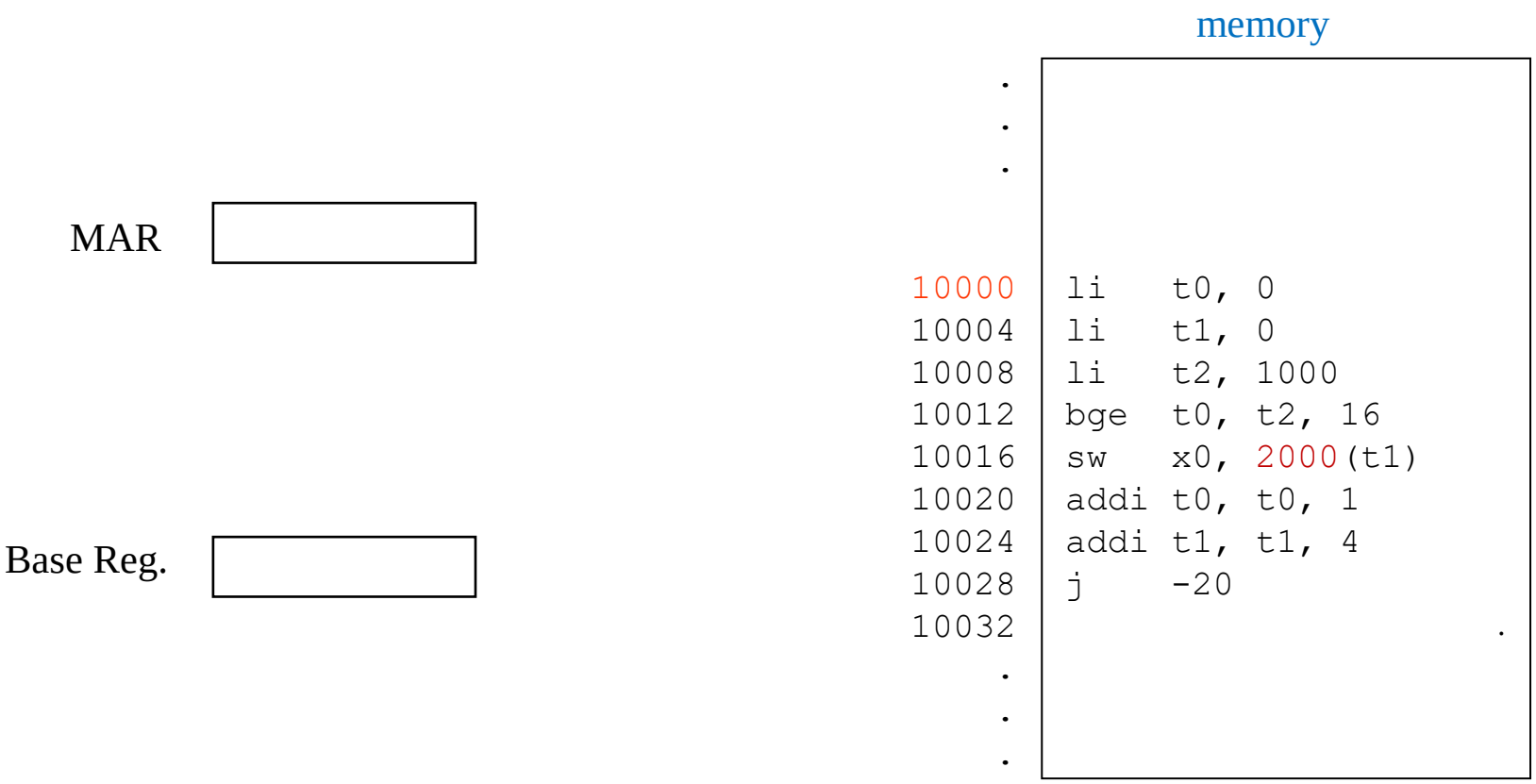
Hardware relocation

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



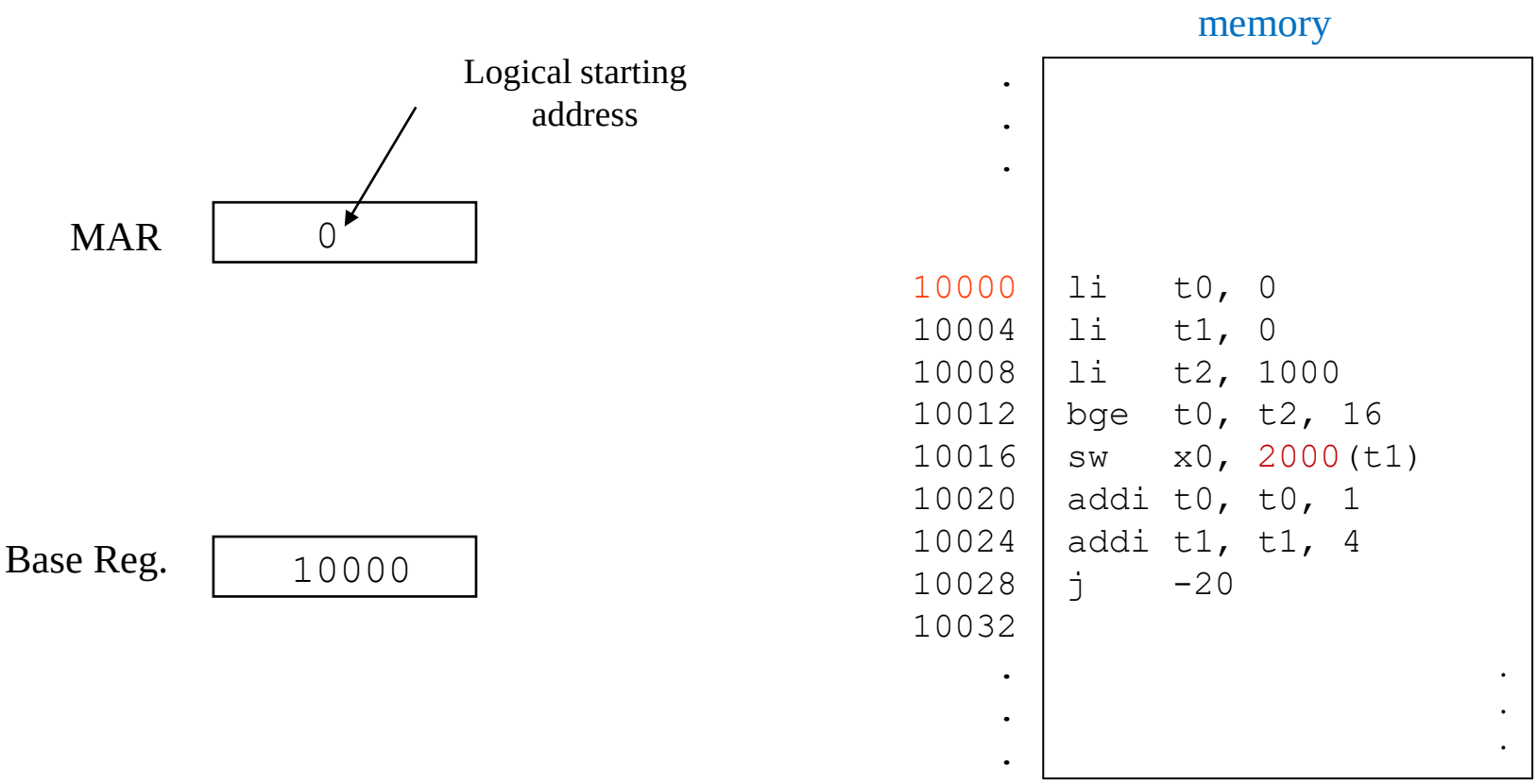
Example of hardware support

- Base register: program start address in memory



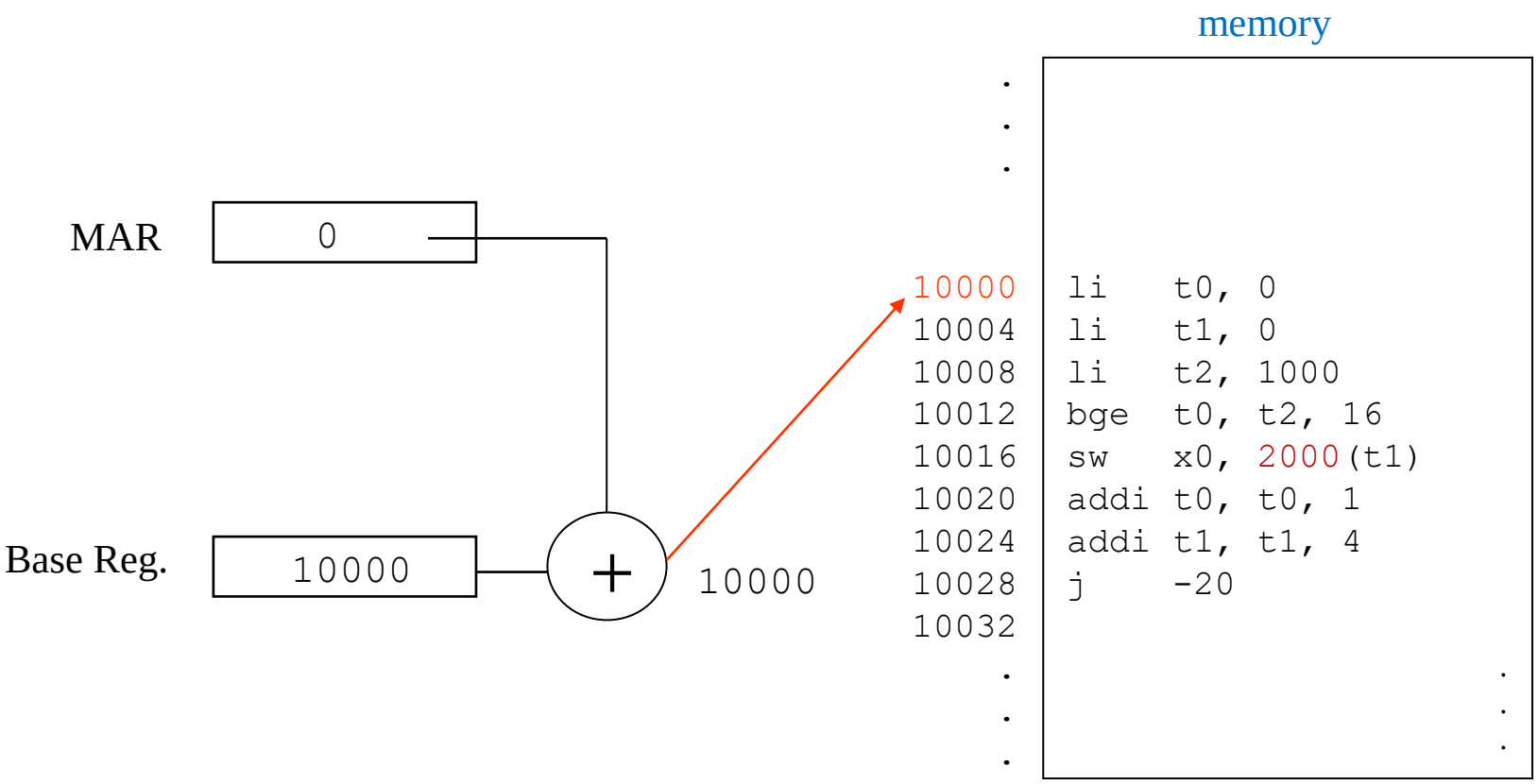
Example of hardware support

- Base register: program start address in memory

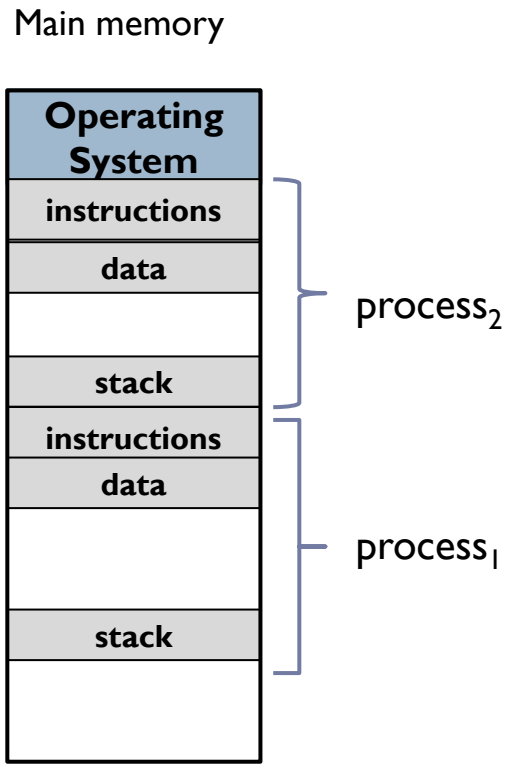


Example of hardware support

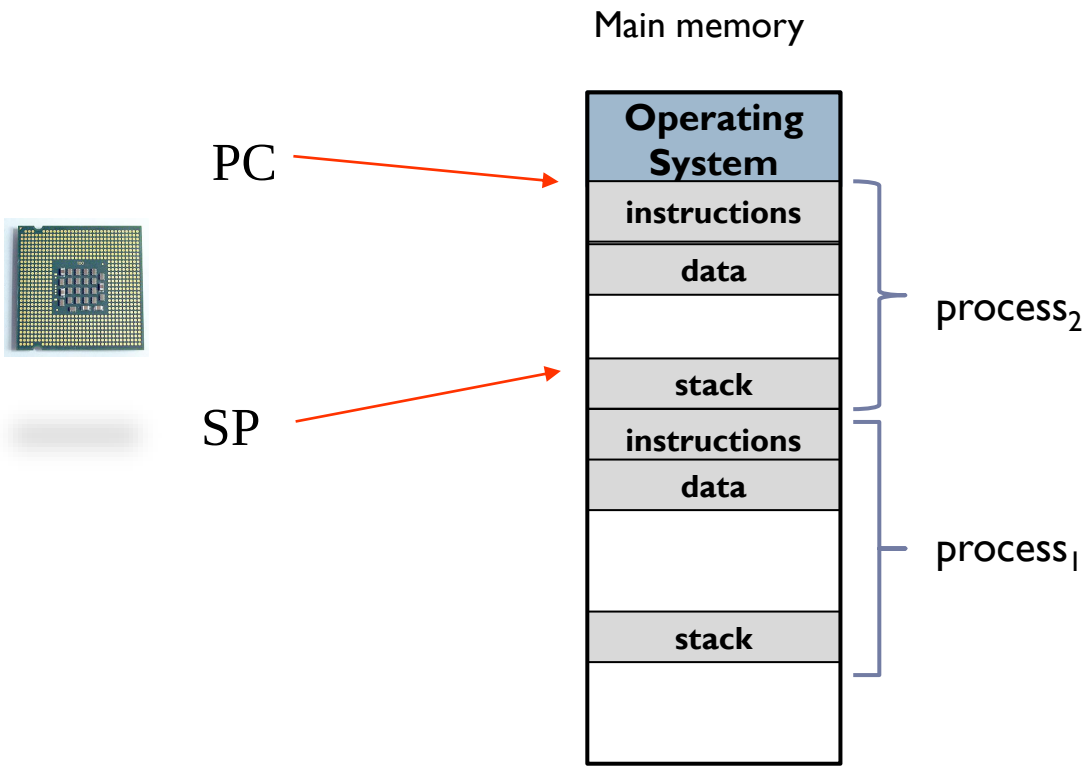
- Base register: program start address in memory



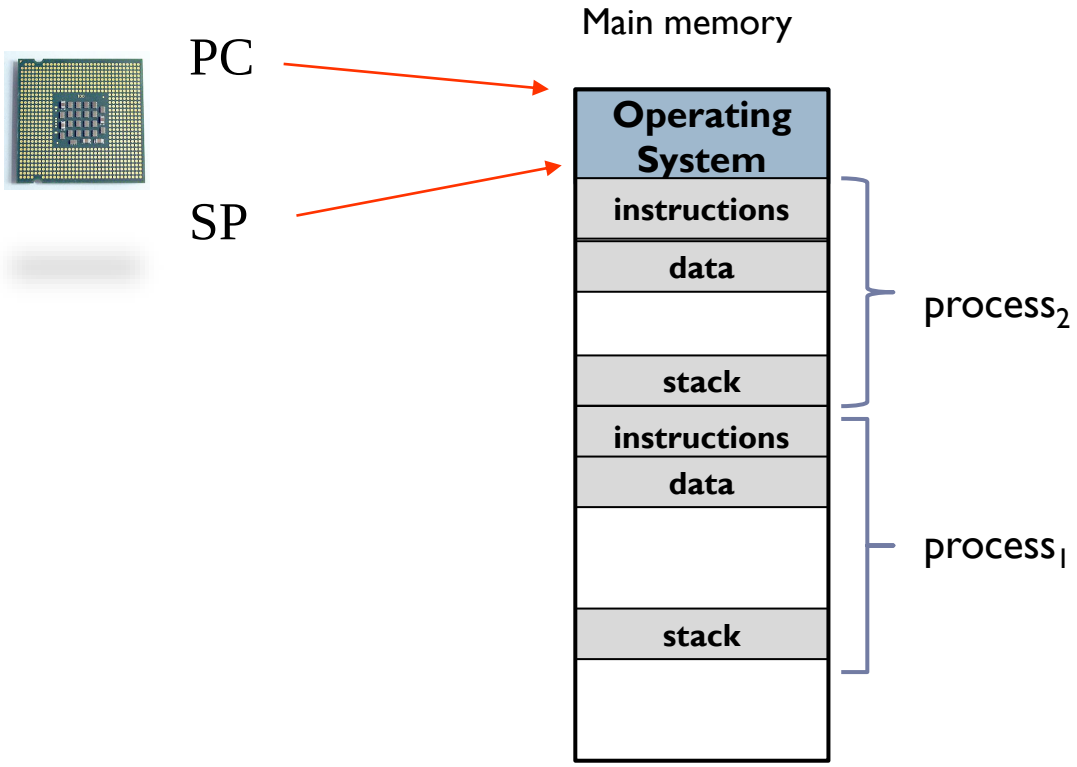
Multiple programs loaded in memory



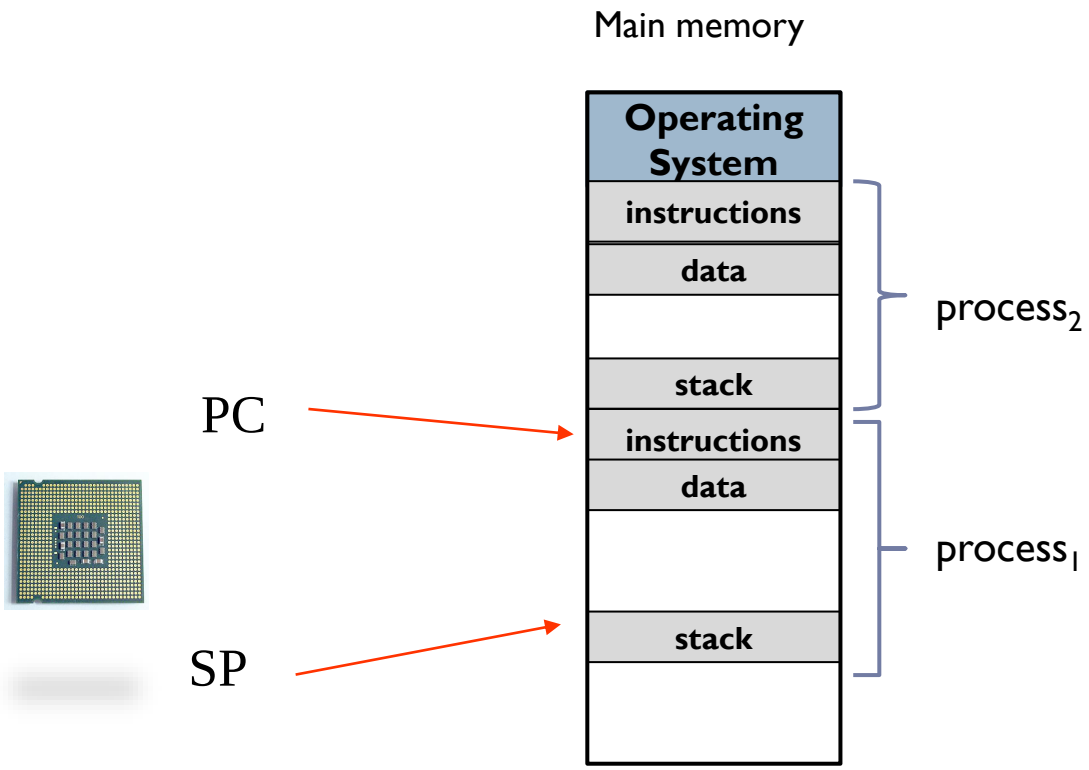
Multiple programs loaded in memory



Multiple programs loaded in memory

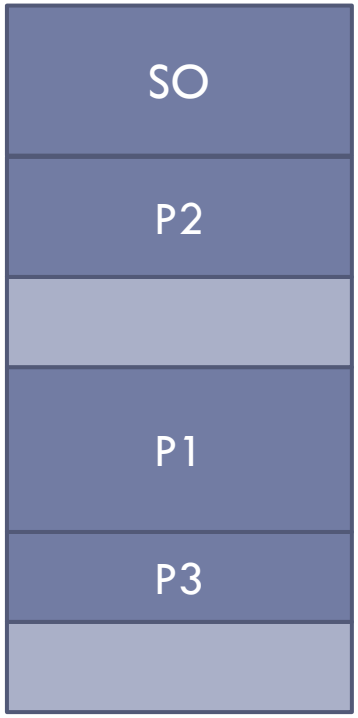


Multiple programs loaded in memory



Multiprogramming: memory protection

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



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

Problem with memory protection

- What happens if the program executes these instructions?

```
li t0, 8
sw t0, 0(x0)
```

Problem with memory protection

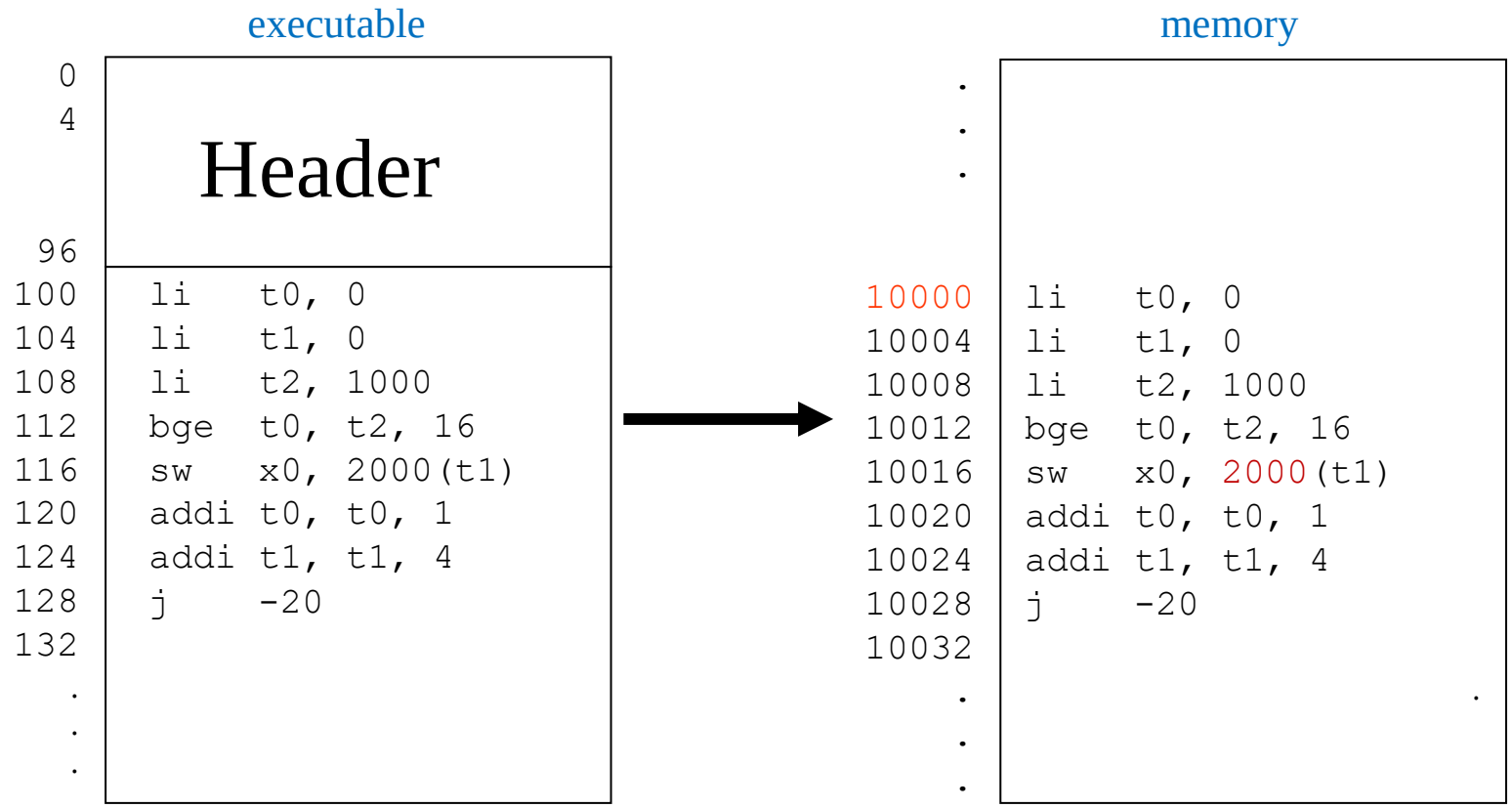
- What happens if the program executes these instructions?

```
li t0, 8
sw t0, 0(x0)
```

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

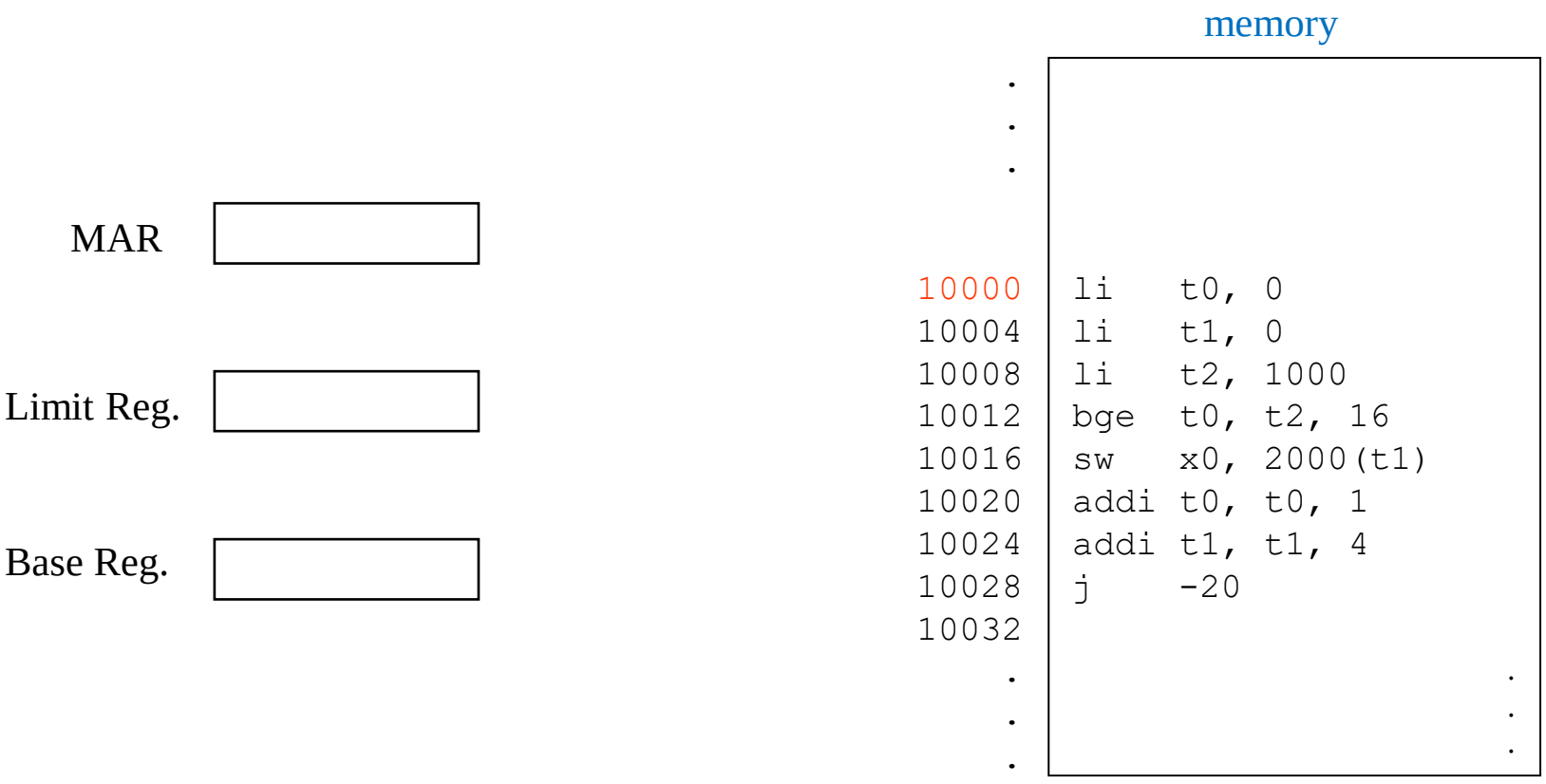
Hardware relocation (with limit register)

- Translation **and testing** is performed during execution.
- Special hardware is needed. Ensure protection.



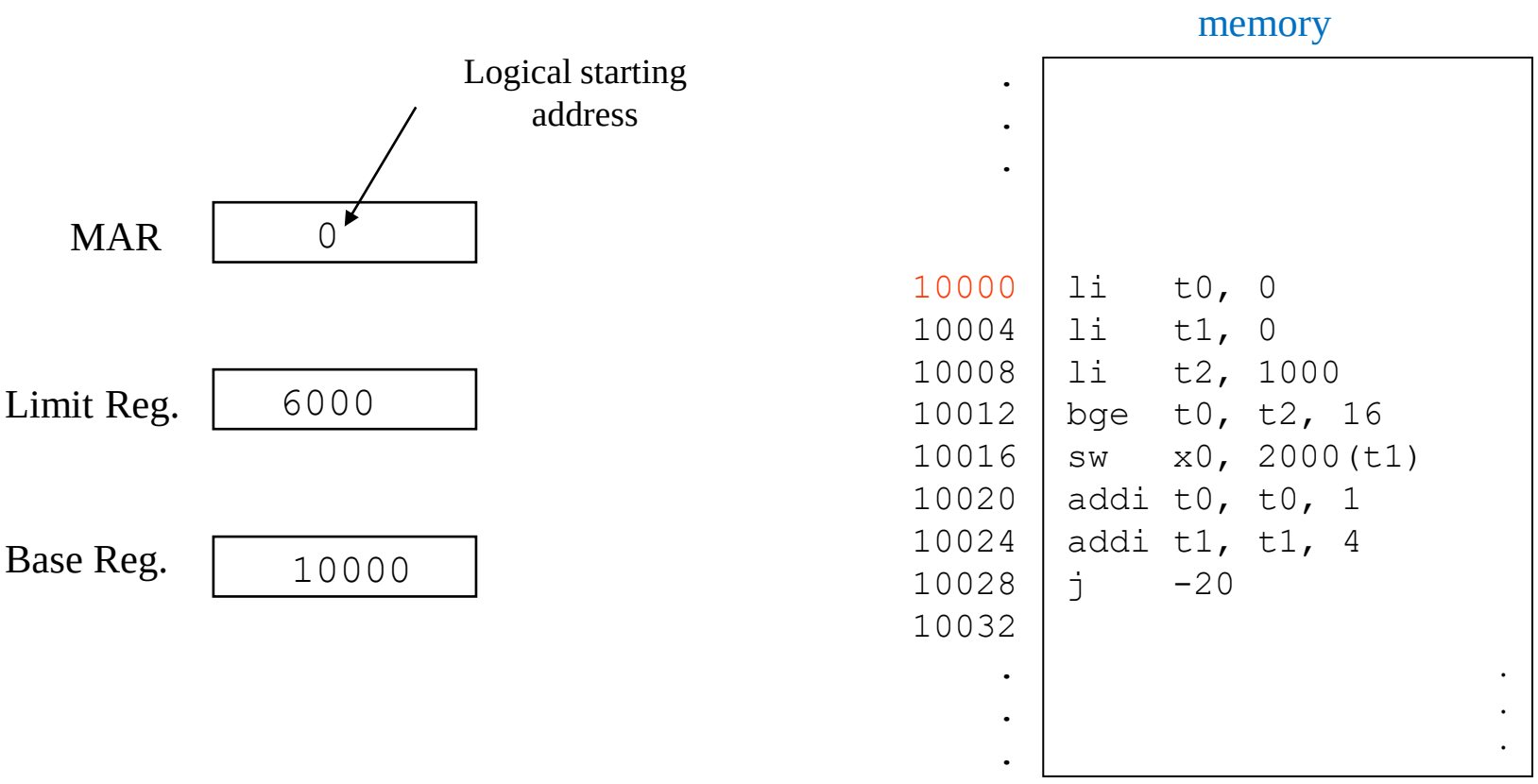
Example of hardware support

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



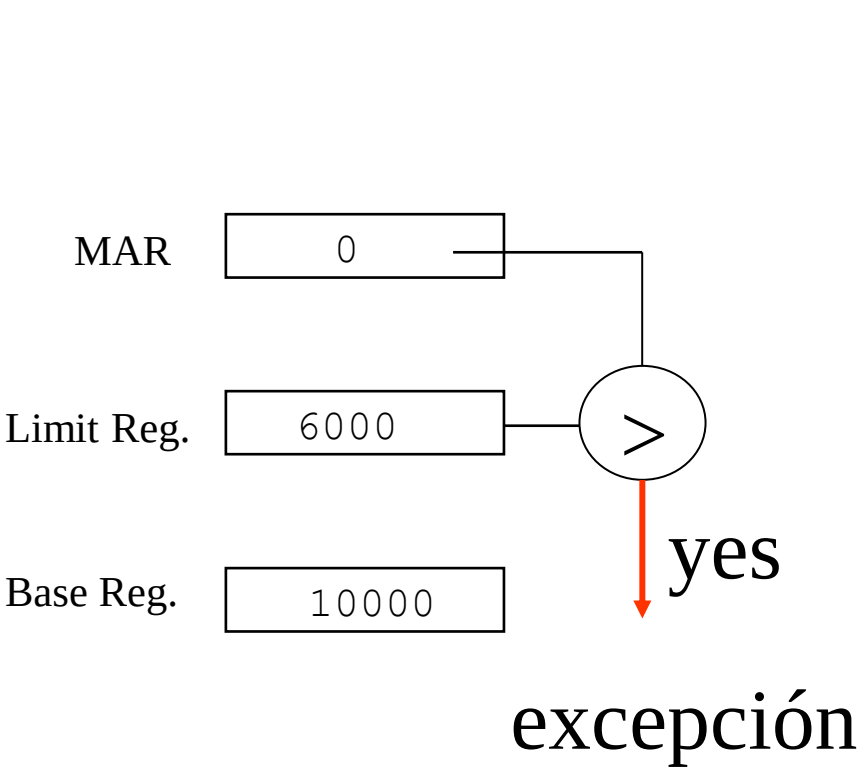
Example of hardware support

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



Example of hardware support

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

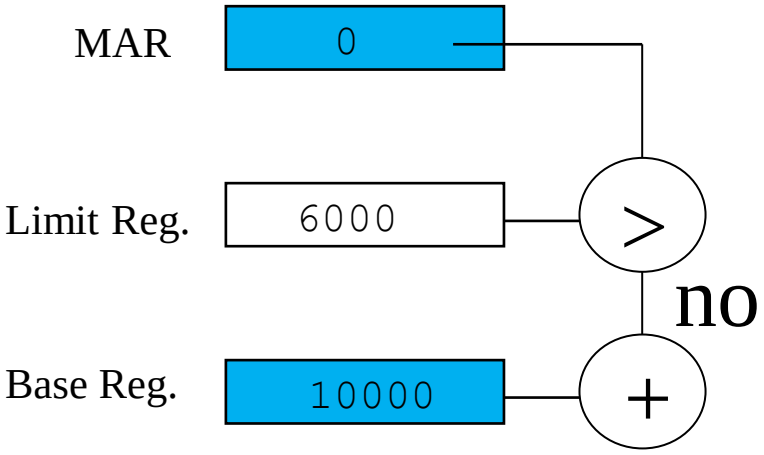


memory

```
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  j    -20
10032
```

Example of hardware support

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

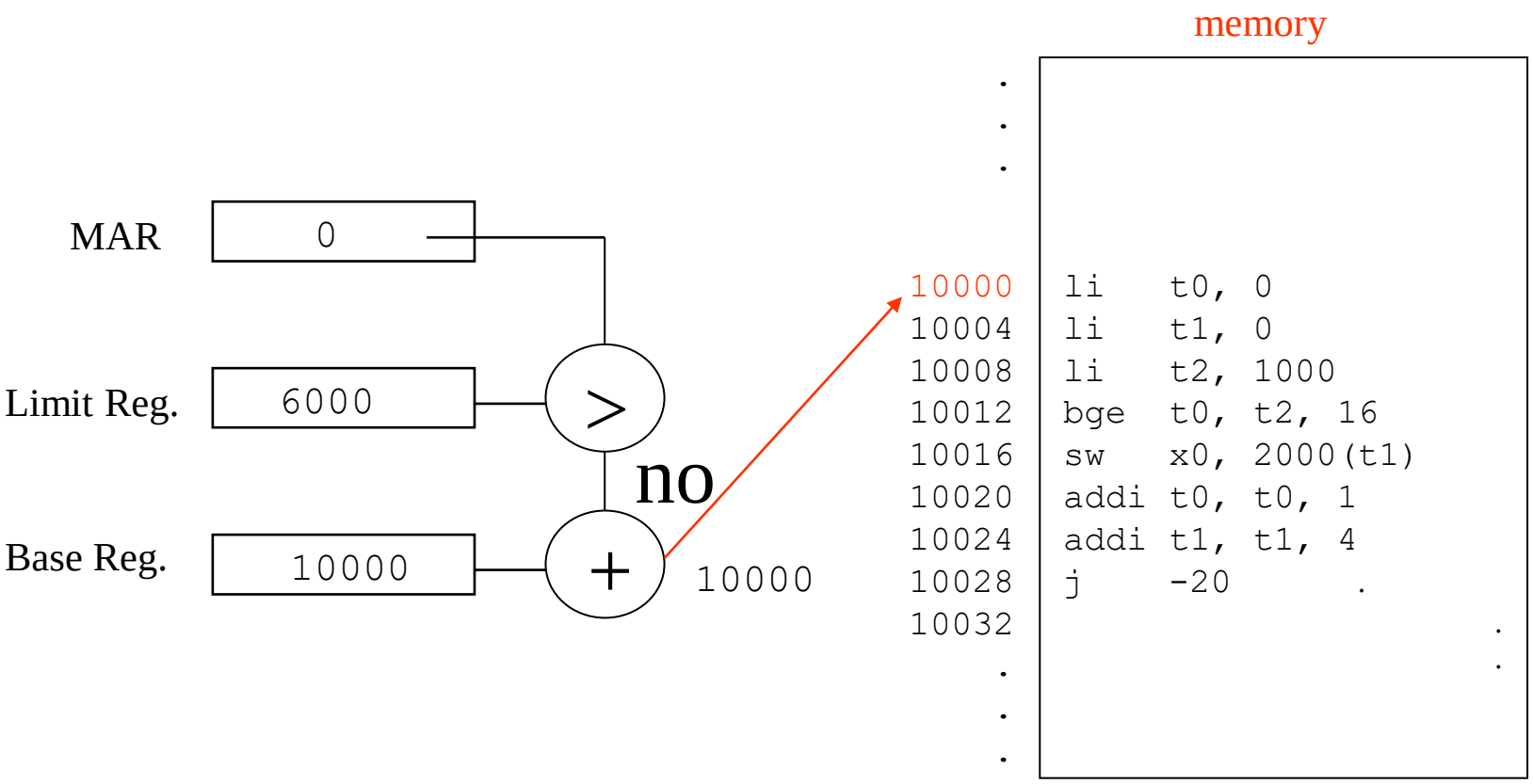


memory

```
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  j    -20
10032
```

Example of hardware support

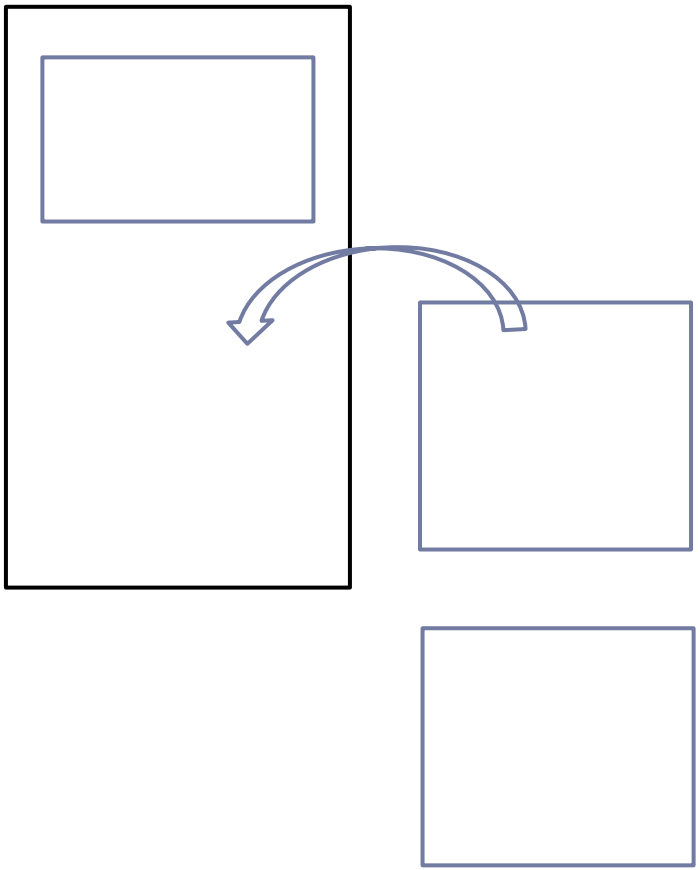
- Limit register: maximum logical address assigned to the program
- Base register: program initial address in memory



Systems without virtual memory
Main problems (summary)

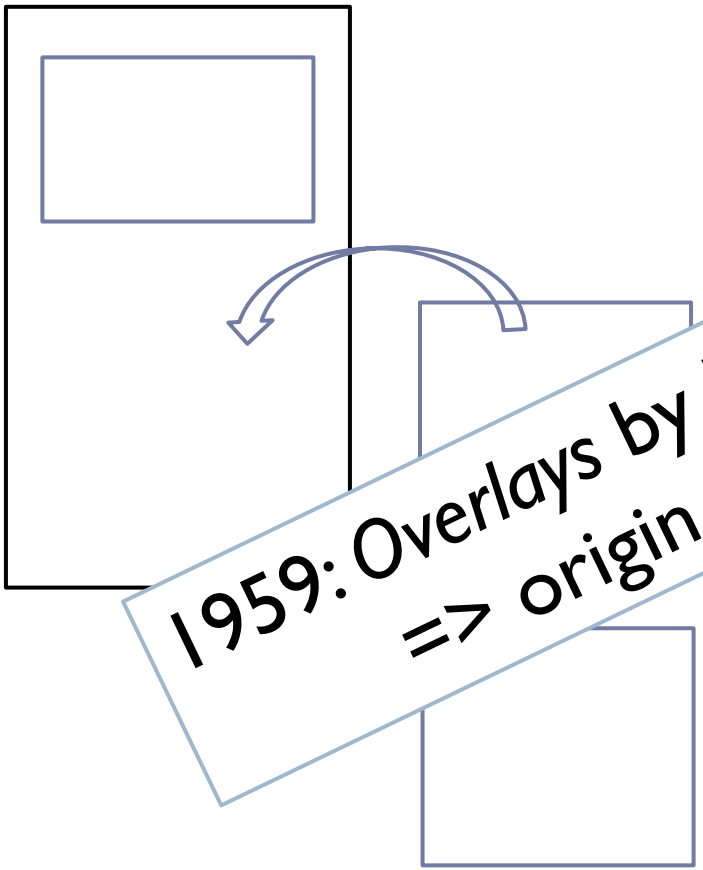
- **Relocation and protection**.
- If the **process image** is **bigger** than the **available memory**, the **process can not be executed**.
- The **number** of **active programs** in memory is **limited**.
  - In a 32-bit computer:
    - What is the theoretical maximum size of a program?
    - What if this size if the memory has 512 MB?
    - If each program occupies 100 MiB, how many can I run?

Overlays



- In the years 1950-85 the IBM Mainframe-PC had little memory and no virtual memory.
- Using overlays was a popular technique for loading part of the program when it was used, and unloading to make room when it was not needed.

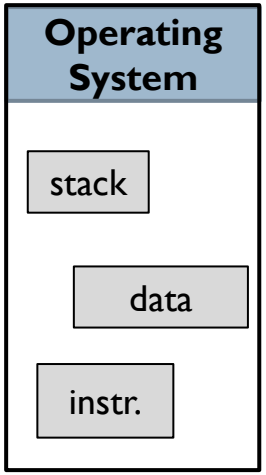
Overlays



1959: Overlays by hardware?
=> origin of virtual memory

- In the years 1950-85 the IBM Mainfr… PC had little… no …
- …sing overlays was a popular technique for loading part of the program when it was used, and unloading to make room when it was not needed.

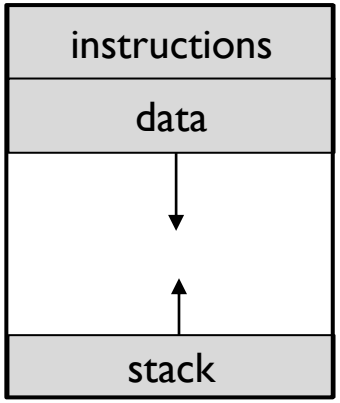
Systems **with** virtual memory



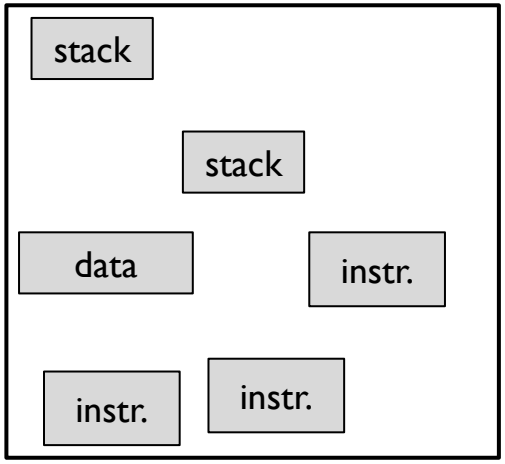
- Programs are partially loaded into main memory for execution:
  - When a part of it is needed, it is loaded in main memory.
  - When it is not needed, it is moved to secondary memory (e.g., SSD, hard disk, etc.)
- Main advantages:
  - A program bigger than the main memory available can be execute.
  - More programs can be executed at the same time.
  - Each program has its own memory space.

Systems **with** virtual memory

Process memory image



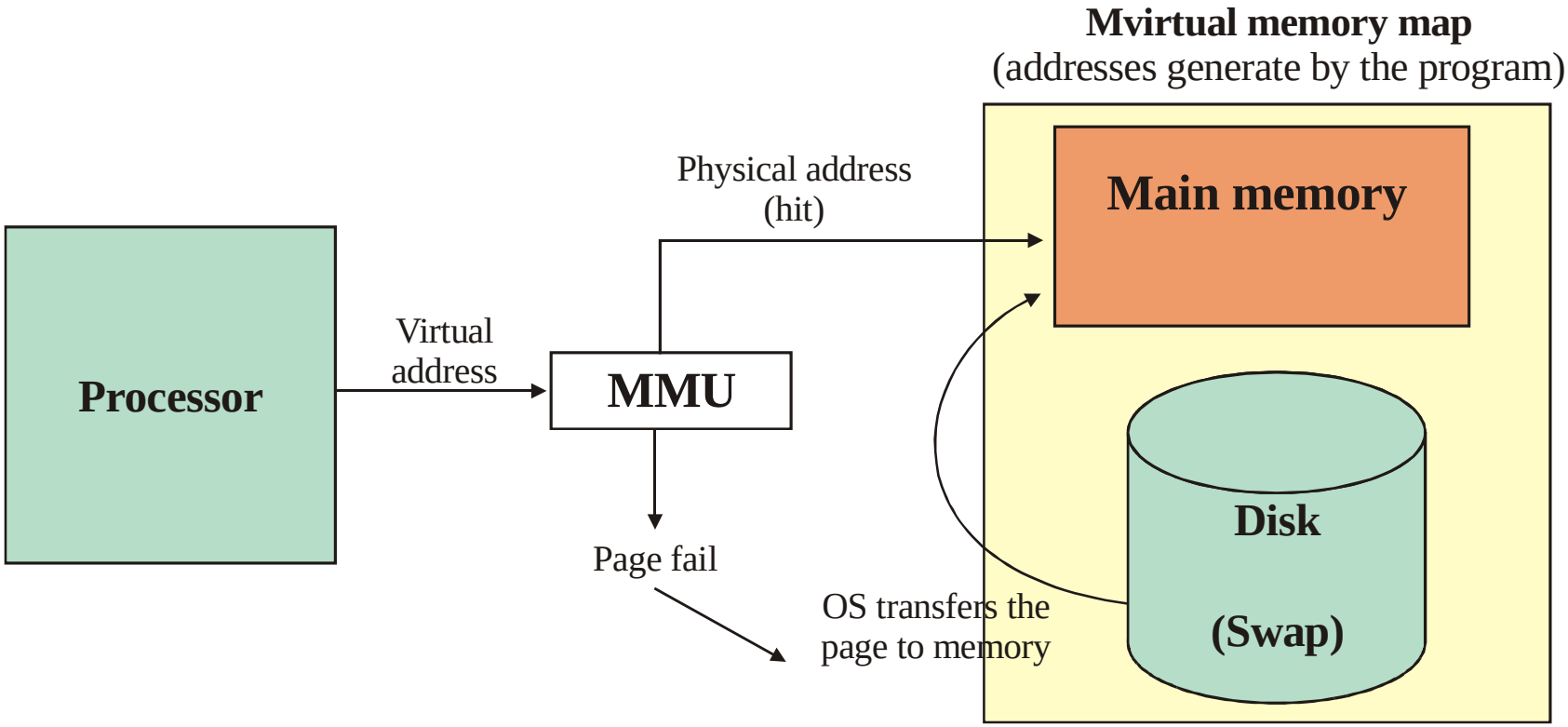
Main memory



Main concepts on virtual memory

Virtual memory uses:

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



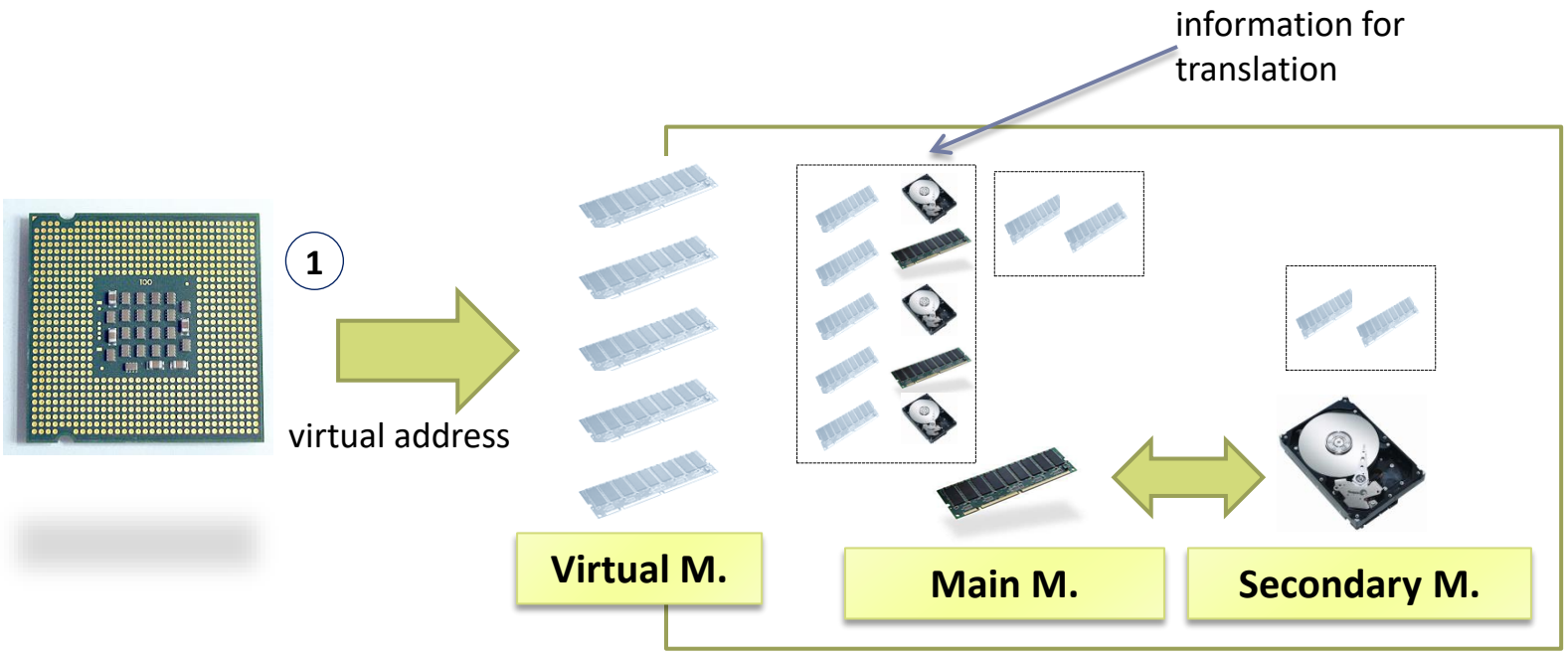
Different models of virtual memory

- Virtual memory **paged**
- Virtual memory **segmented**
- Virtual memory with **paged segmentation**

Paged virtual memory

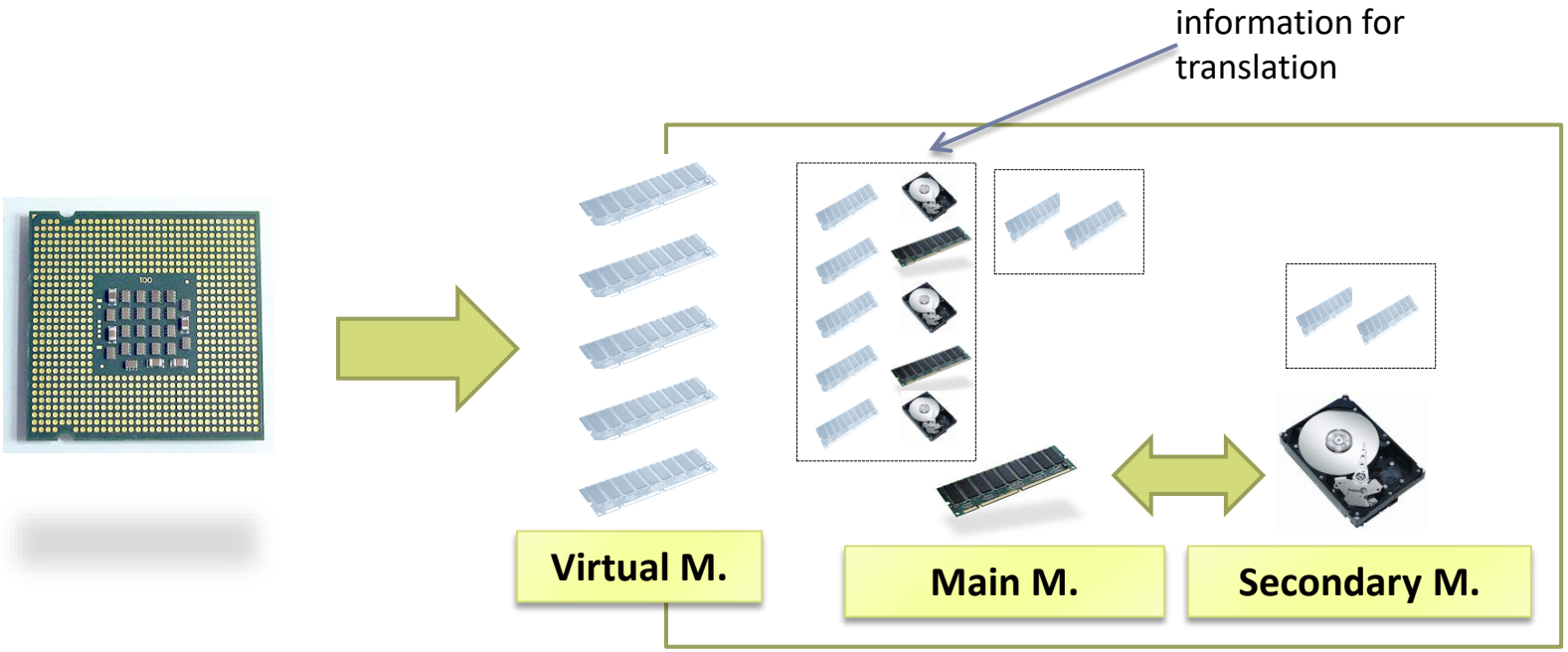
- The addresses generated by the processor are virtual addresses.
- The virtual address space is divided into chunks of equal size called pages.
- The main memory is divided into chunks of equal size to the pages called page frames.
- The disk area that supports the virtual memory is divided into equal-sized chunks called swap pages or swap pages.

Virtual address space

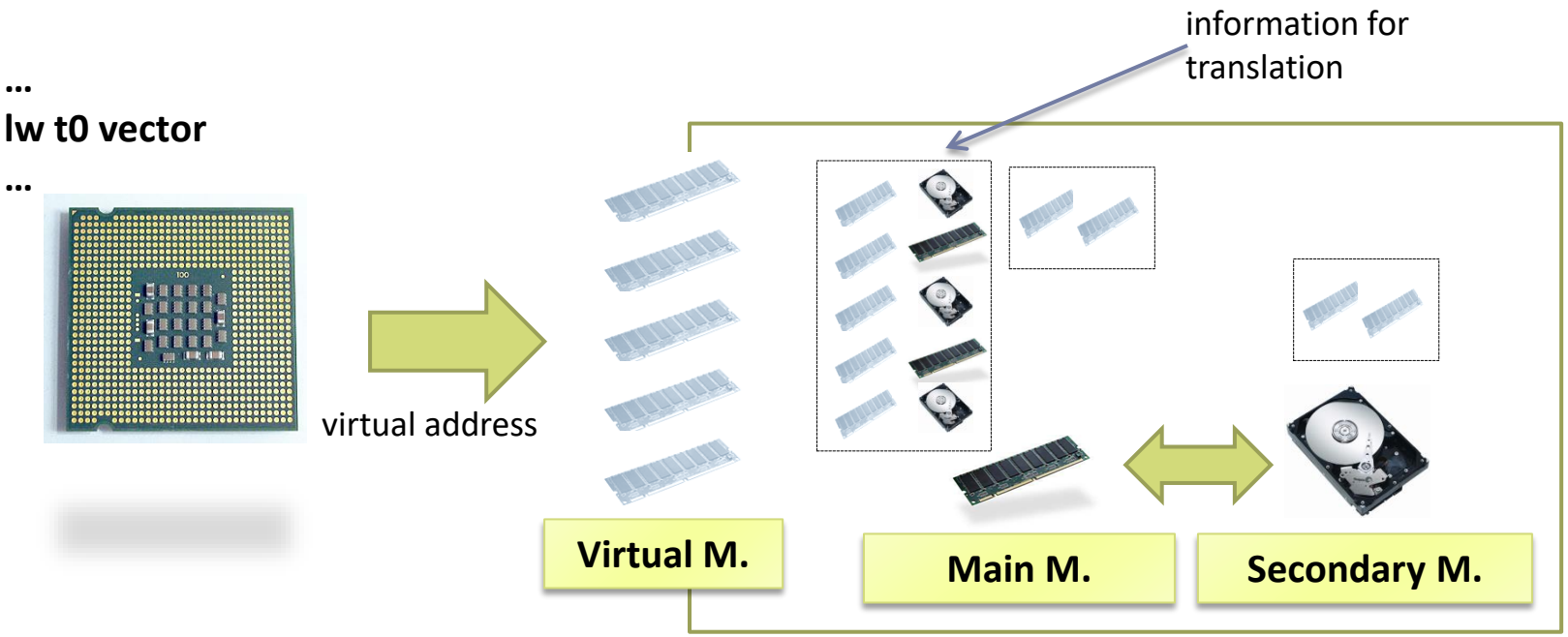


- Virtual address space (Virtual M.)
  - The programs manage a virtual space that resides at MP+disk
  - MMU: Memory Management Unit

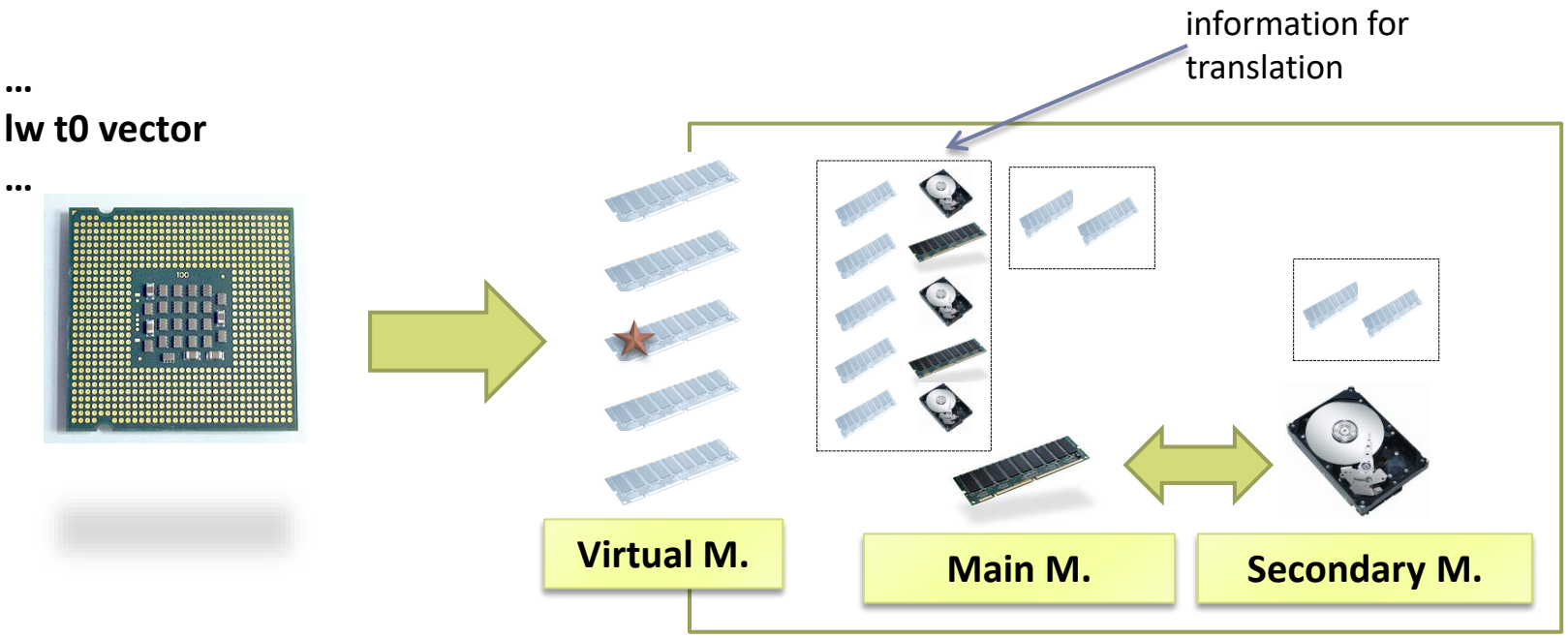
Fundamentals of virtual memory



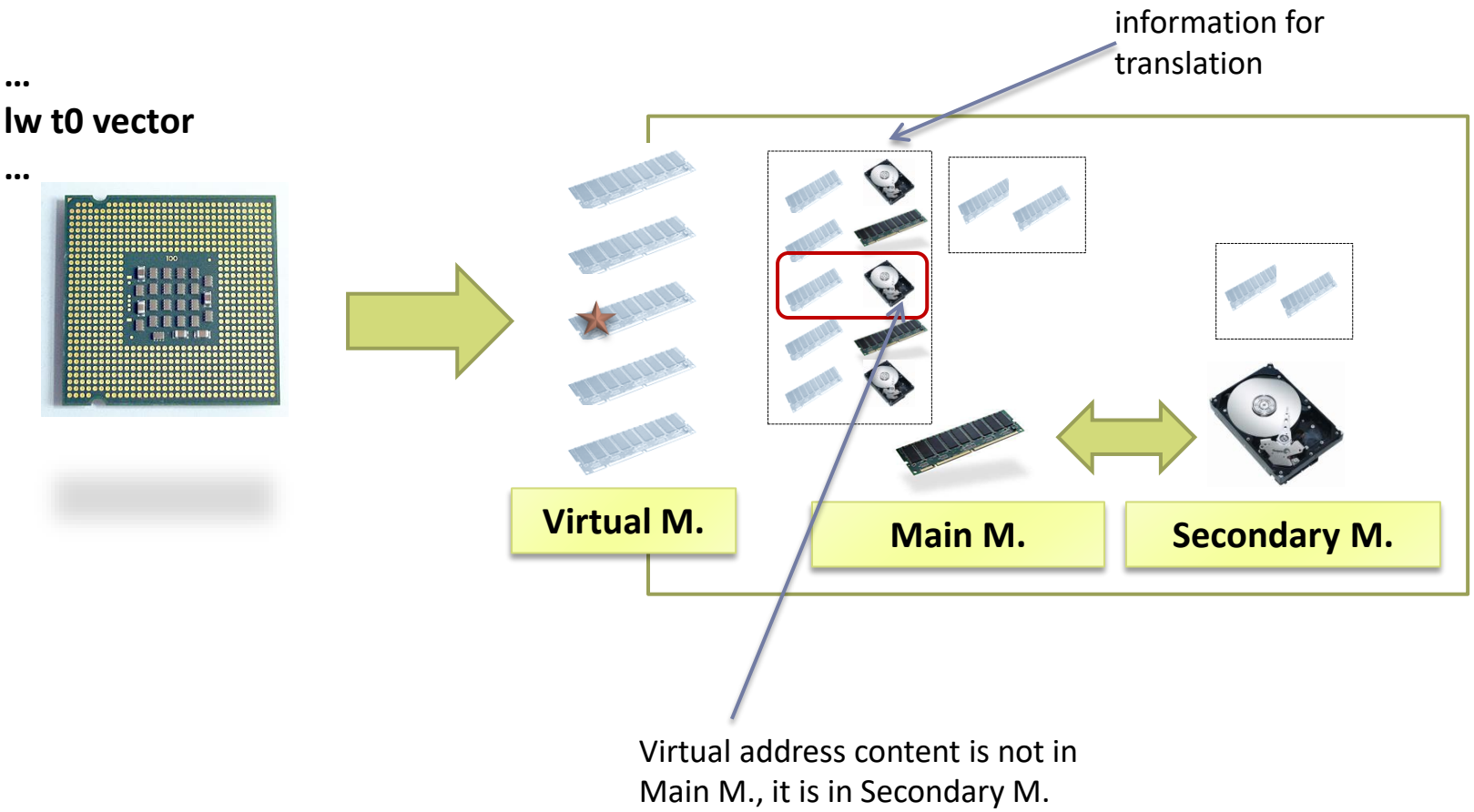
Fundamentals of virtual memory



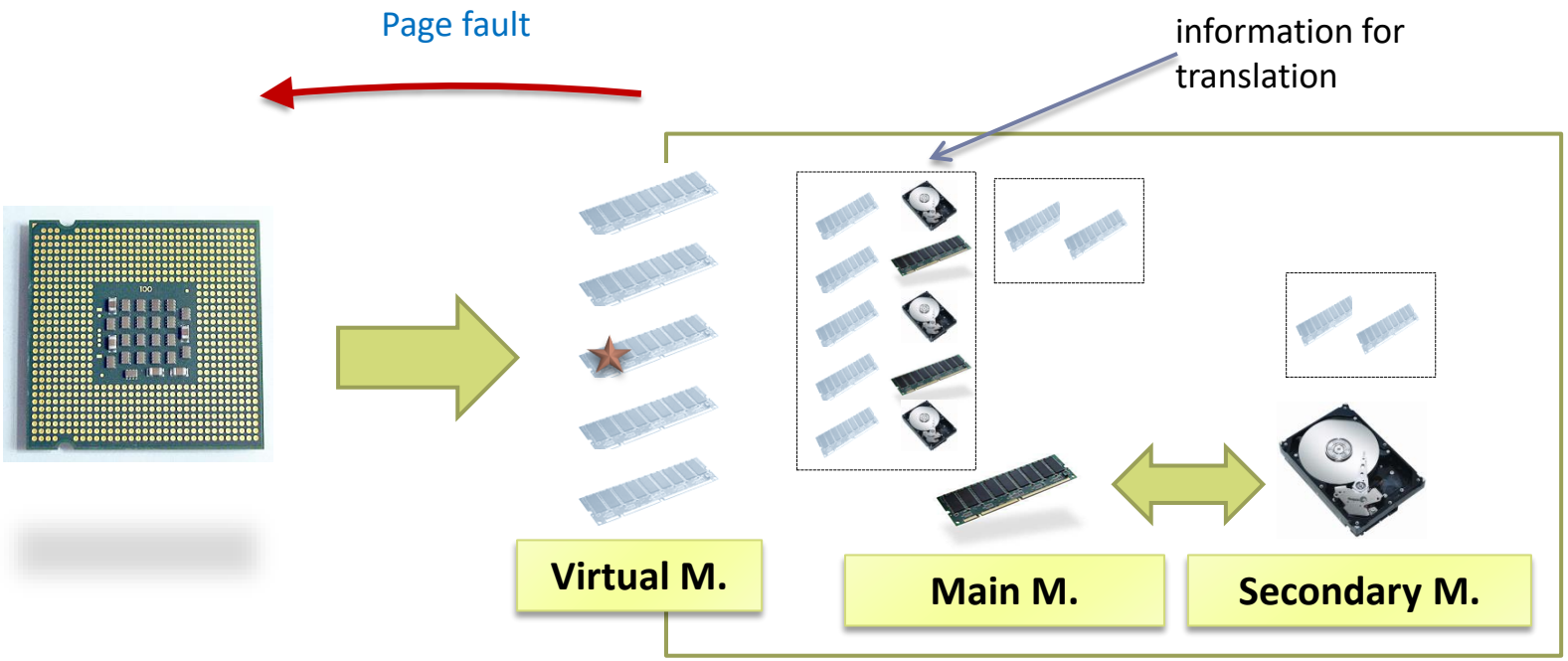
Fundamentals of virtual memory



Fundamentals of virtual memory

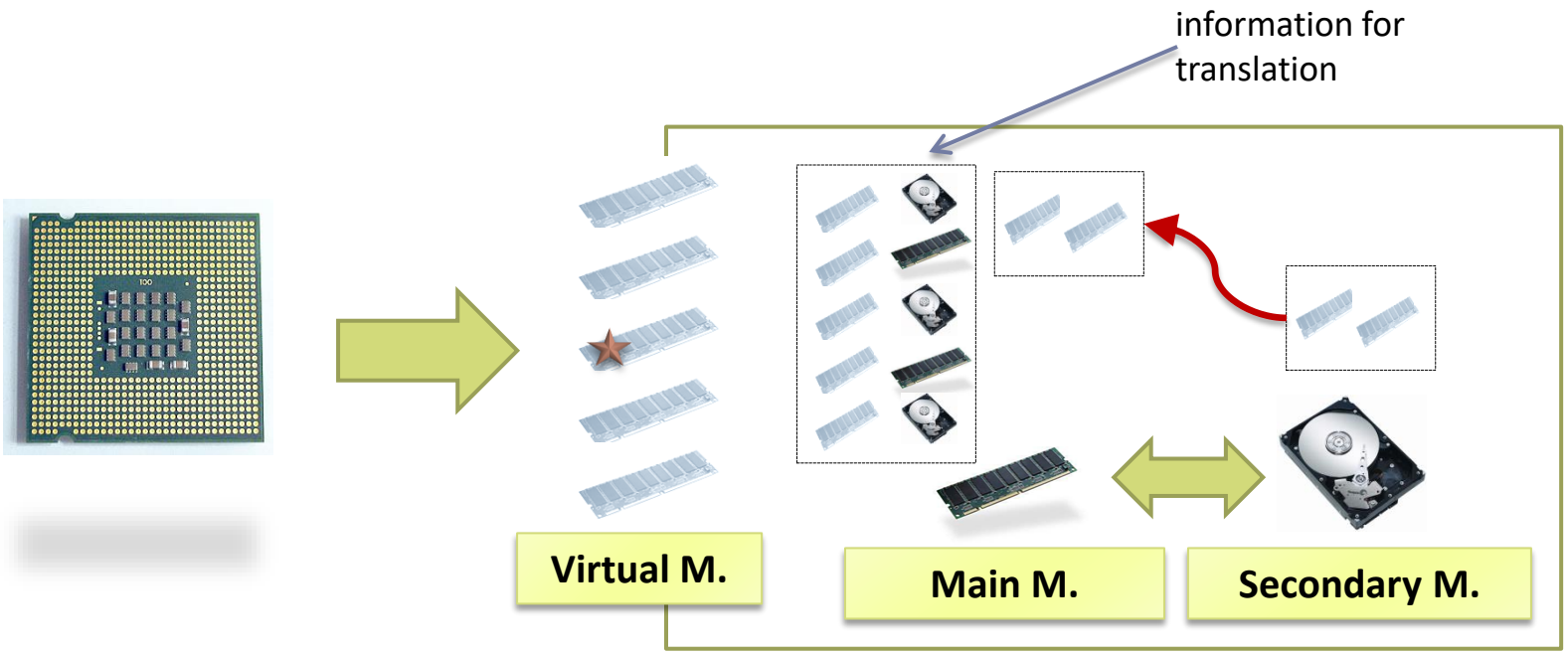


Fundamentals of virtual memory



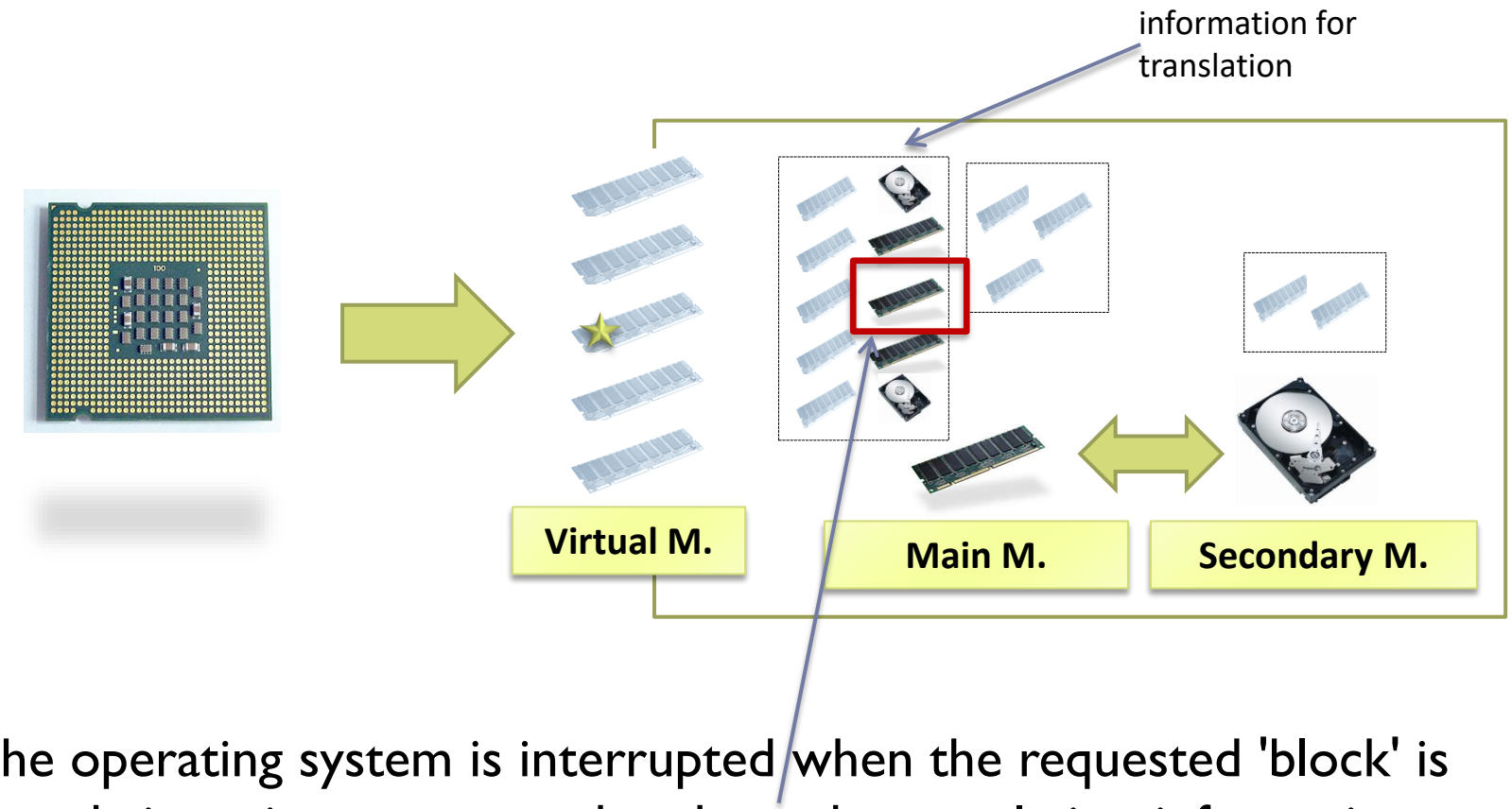
- Page fault is an exception that causes the processor to execute the associated processing routine (the process that generated the page fault is suspended and it cannot continue its execution).
- It is implemented in the operating system.

Fundamentals of virtual memory



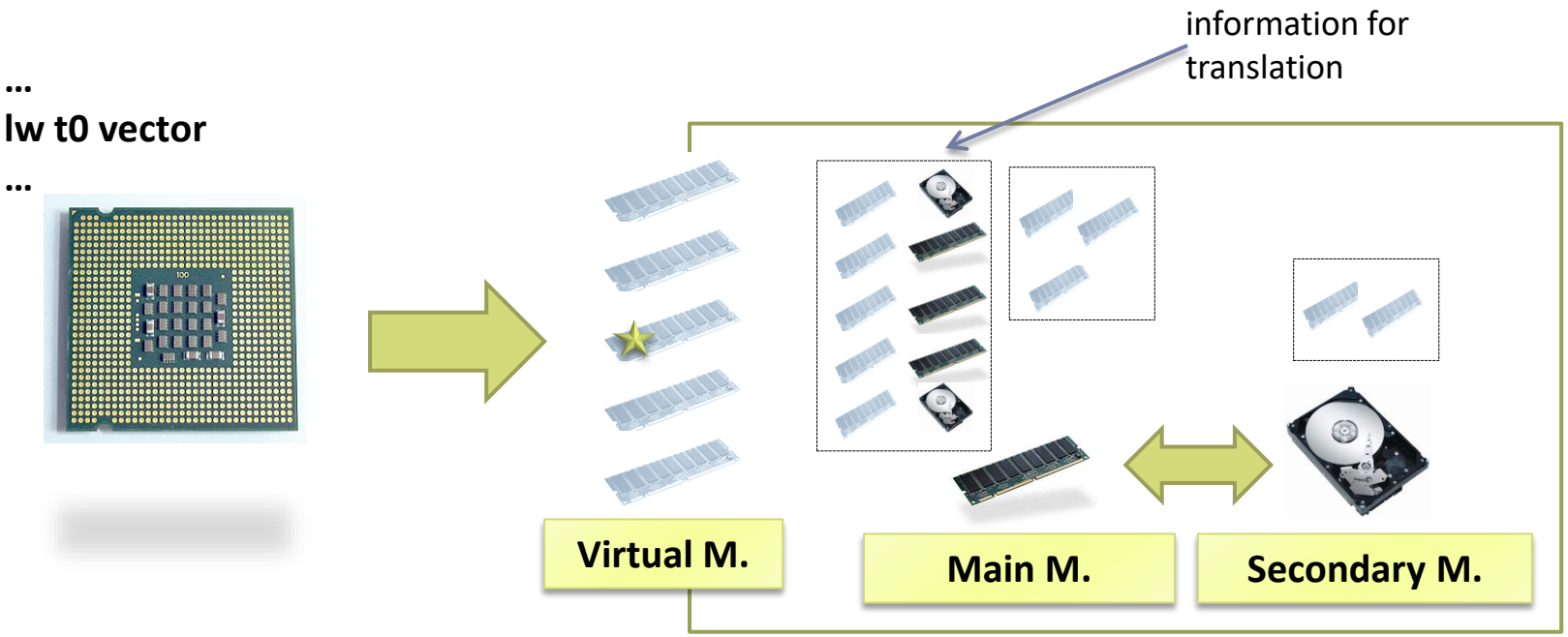
- The operating system asks to transfer the requested 'block' to main memory (the operating system sets another process to execute)

Fundamentals of virtual memory



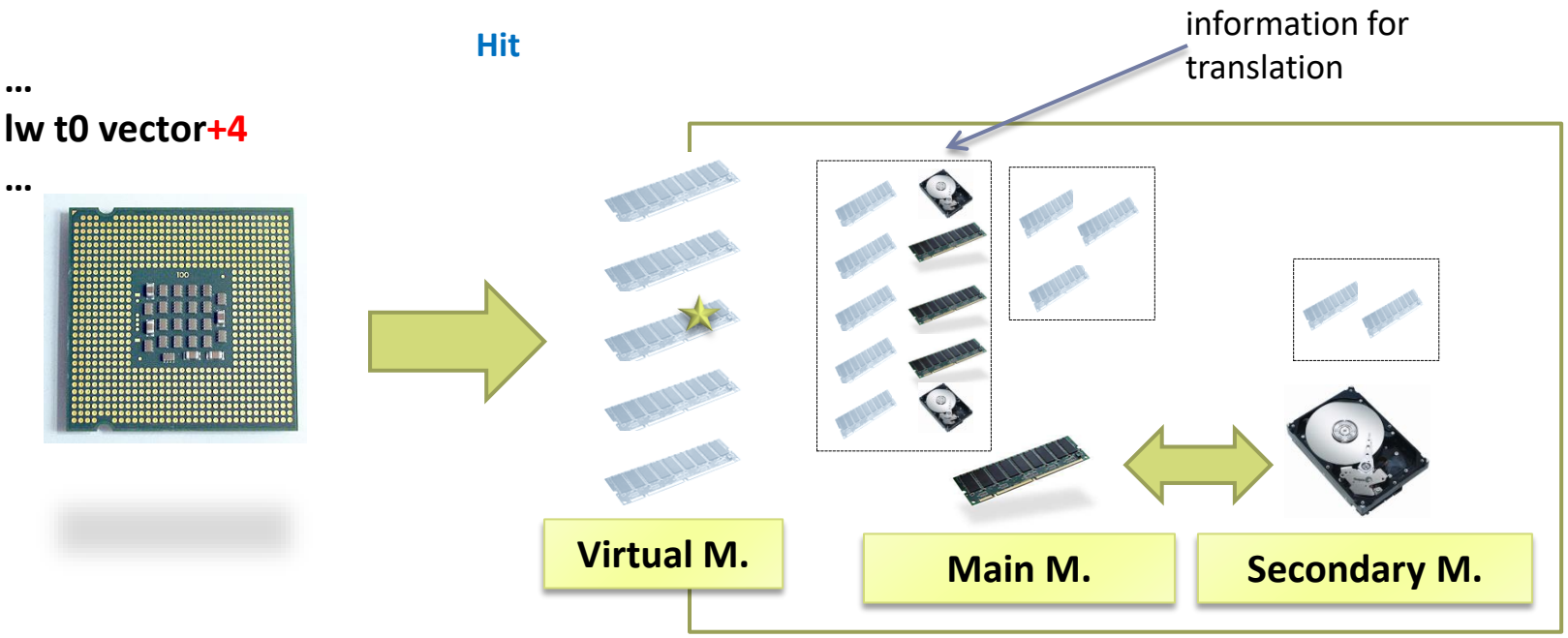
- The operating system is interrupted when the requested 'block' is already in main memory and updates the translation information

Fundamentals of virtual memory

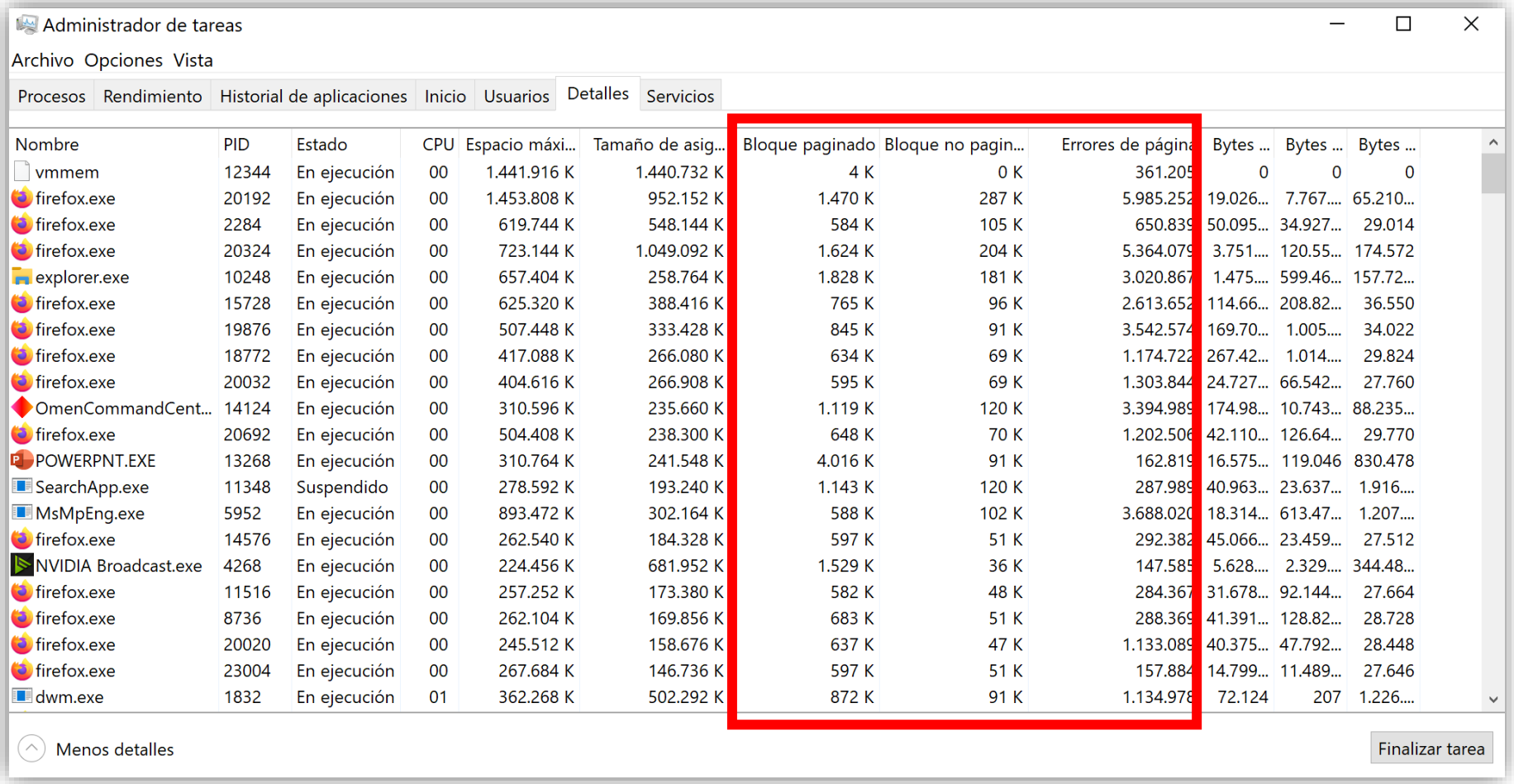


- The execution of the process that caused the failure is resumed and the execution of the instruction that caused the failure is resumed.

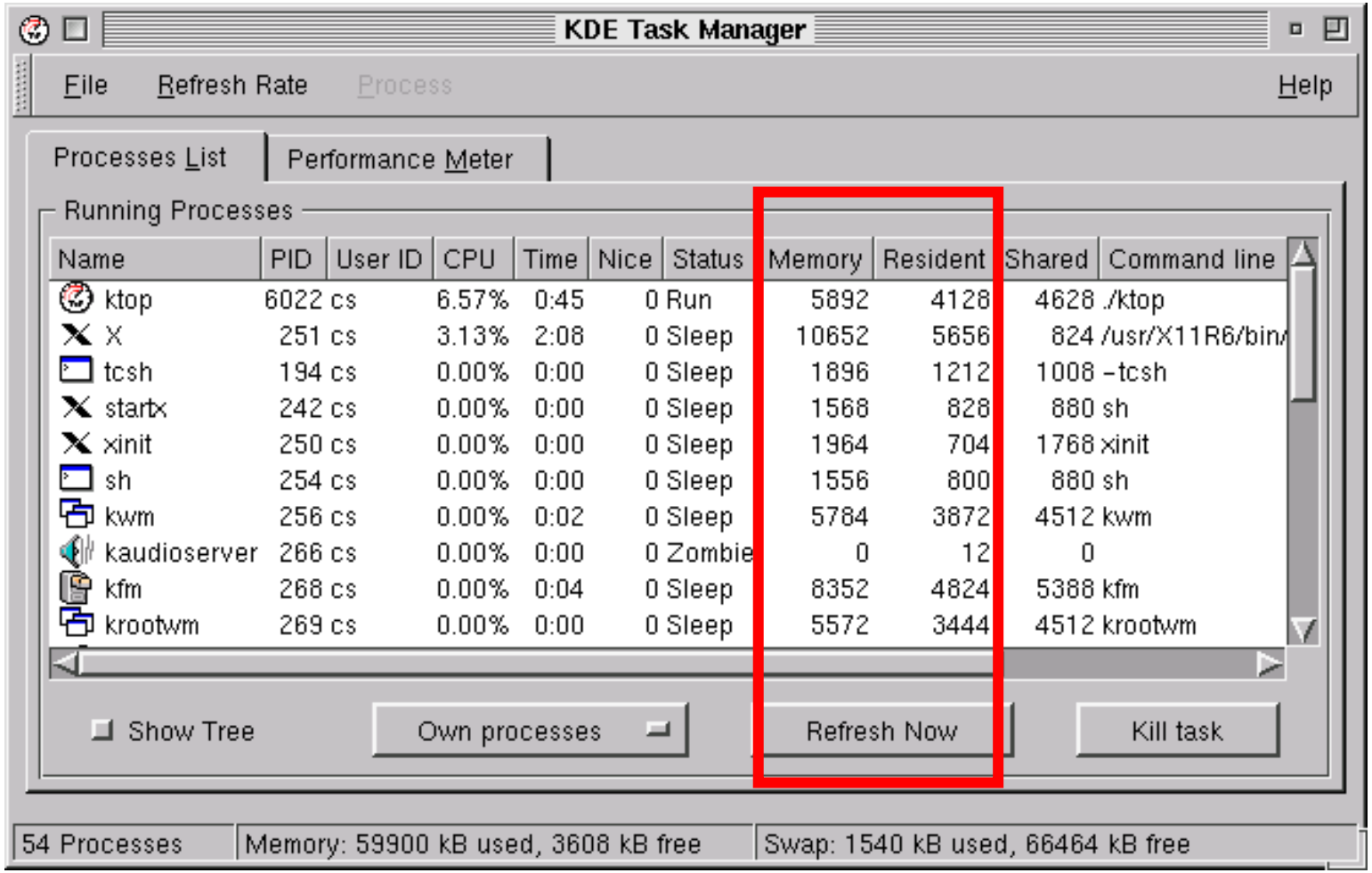
Fundamentals of virtual memory



Virtual memory: windows



Virtual memory: linux

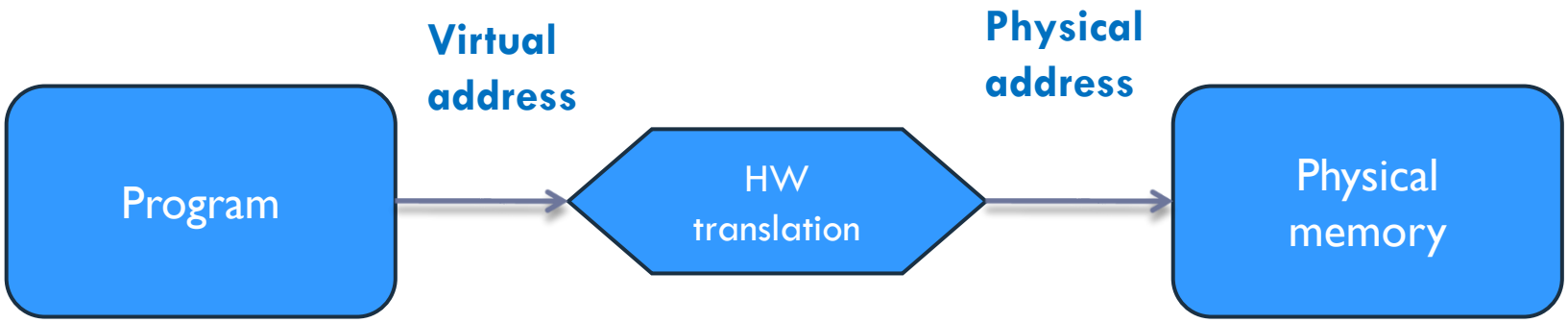


Paged virtual memory
summary

- The addresses generated by the processor are virtual addresses.
- The virtual address space is divided into chunks of equal size called pages.
- The main memory is divided into chunks of equal size to the pages called page frames.
- The disk area that supports the virtual memory is divided into equal-sized chunks called swap pages or swap pages.

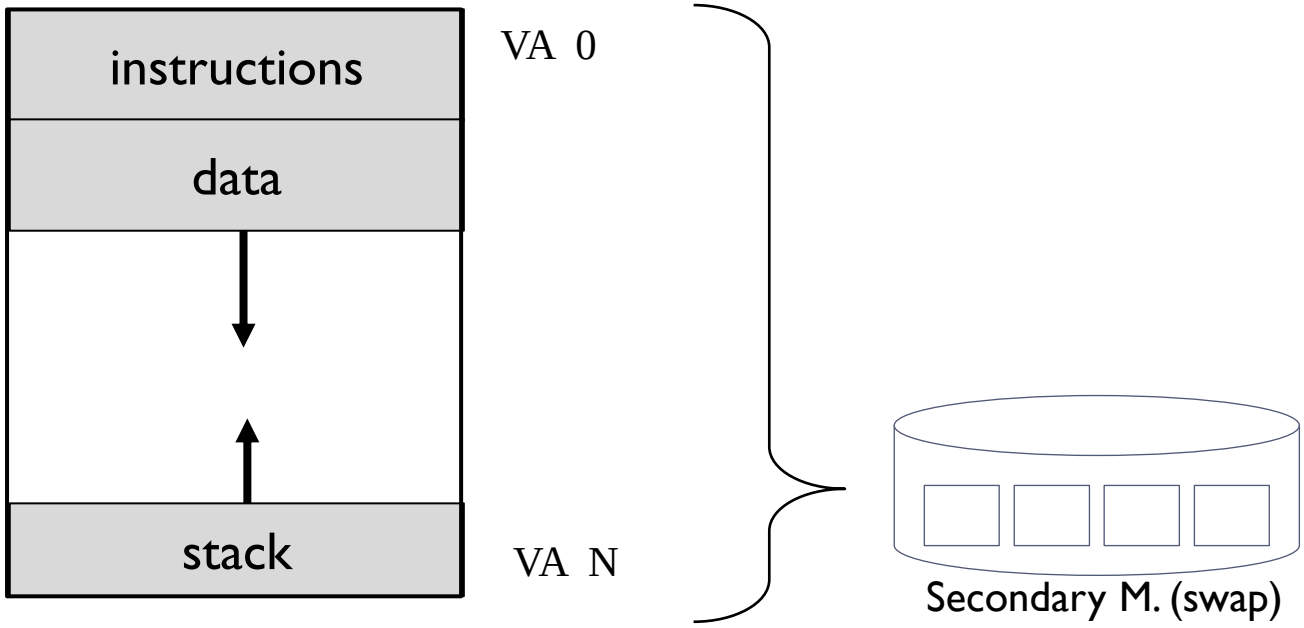
Physical address and virtual address **Translation**

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

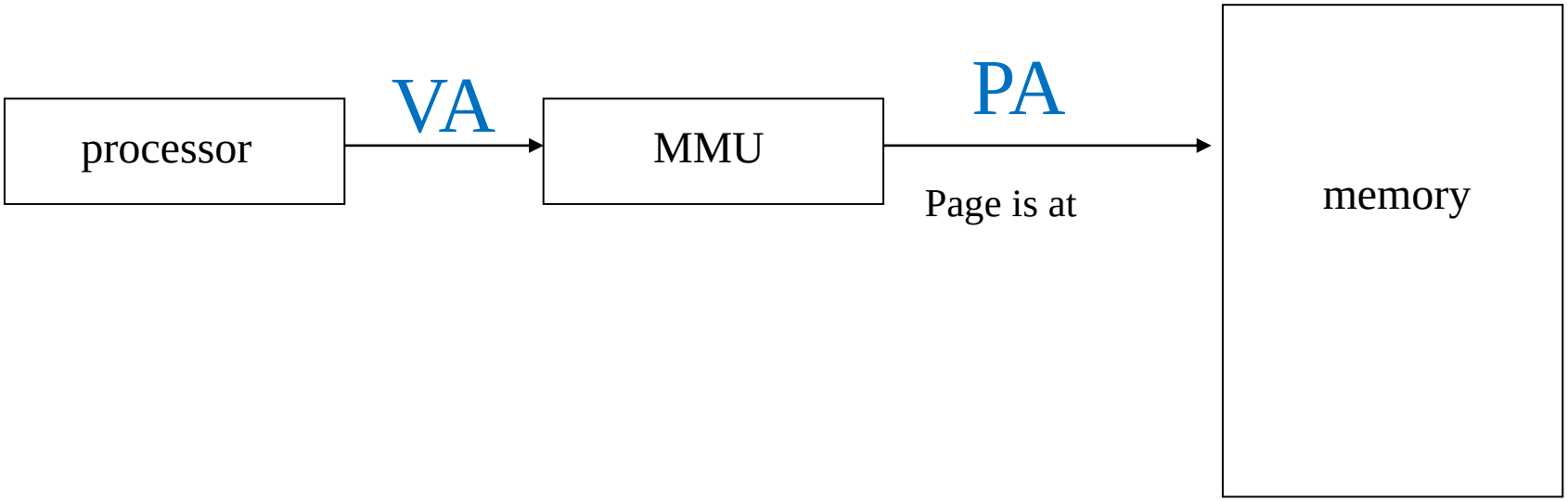


Paged virtual memory

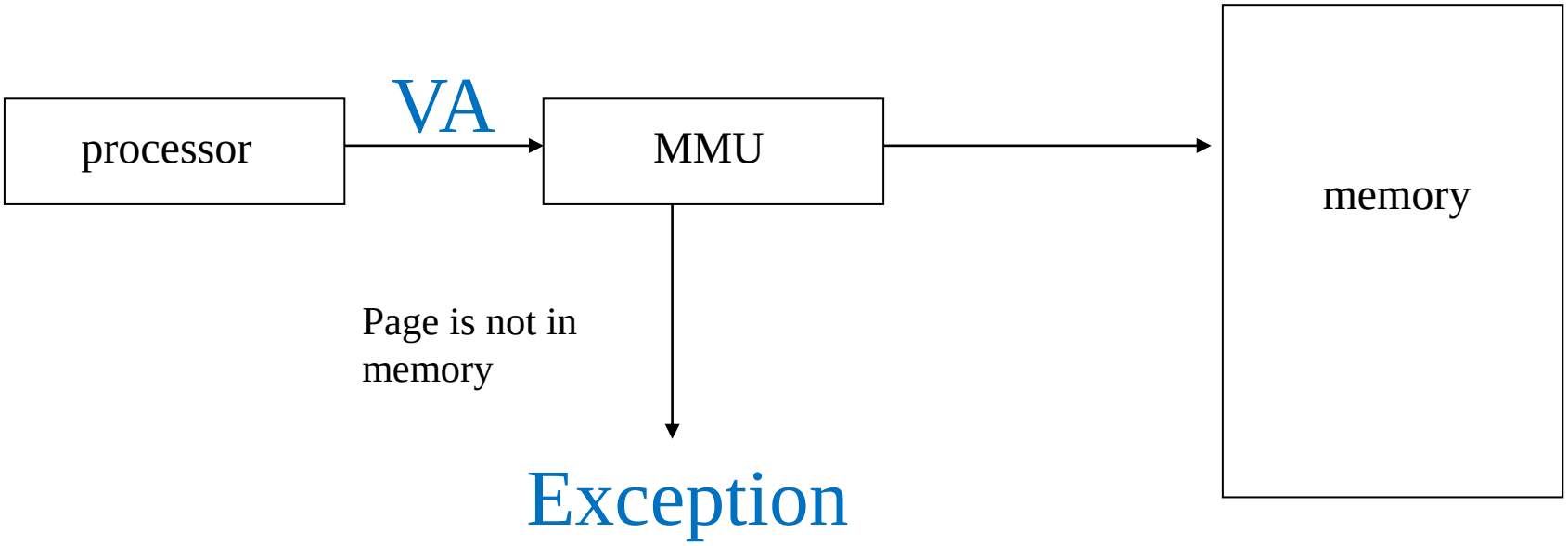
- The memory image of the programs are stored in disk



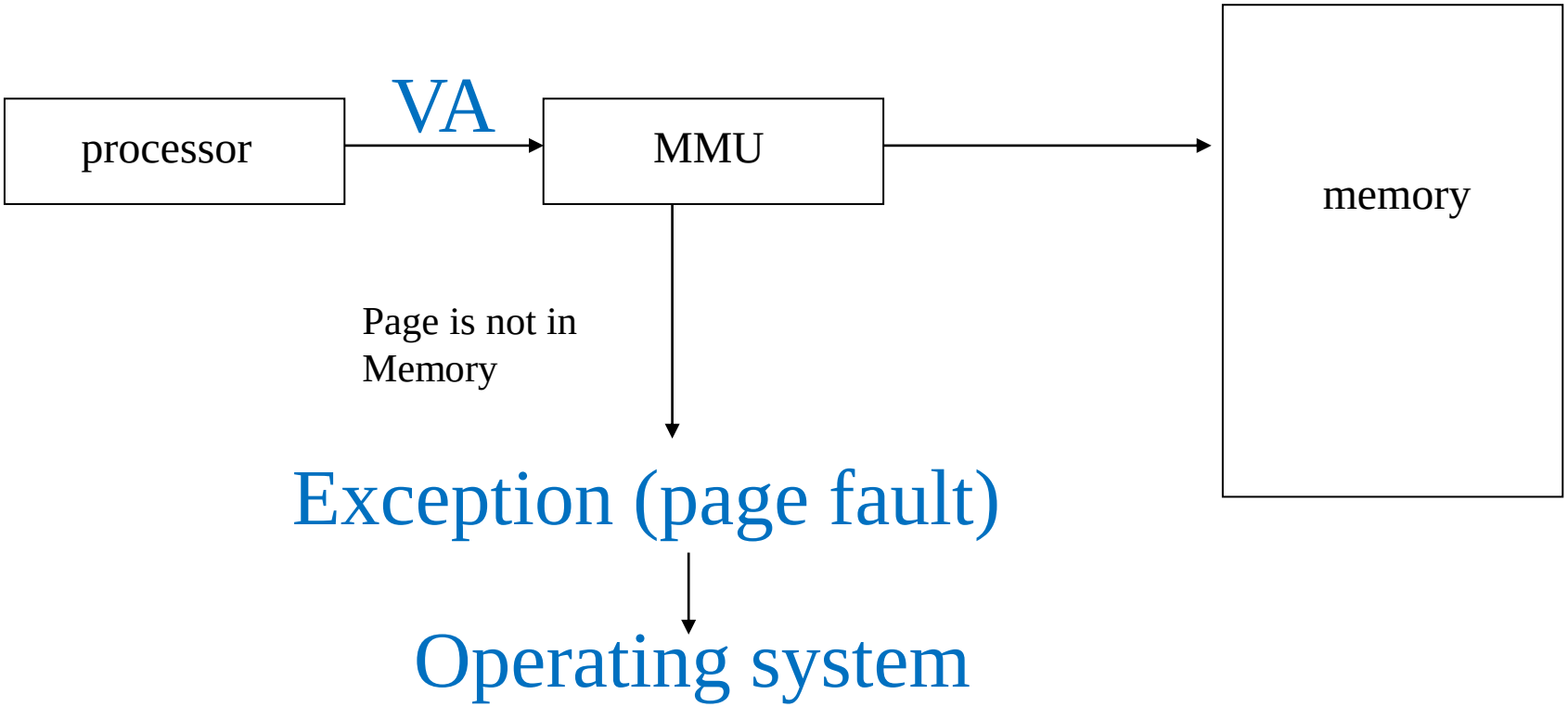
Address translation



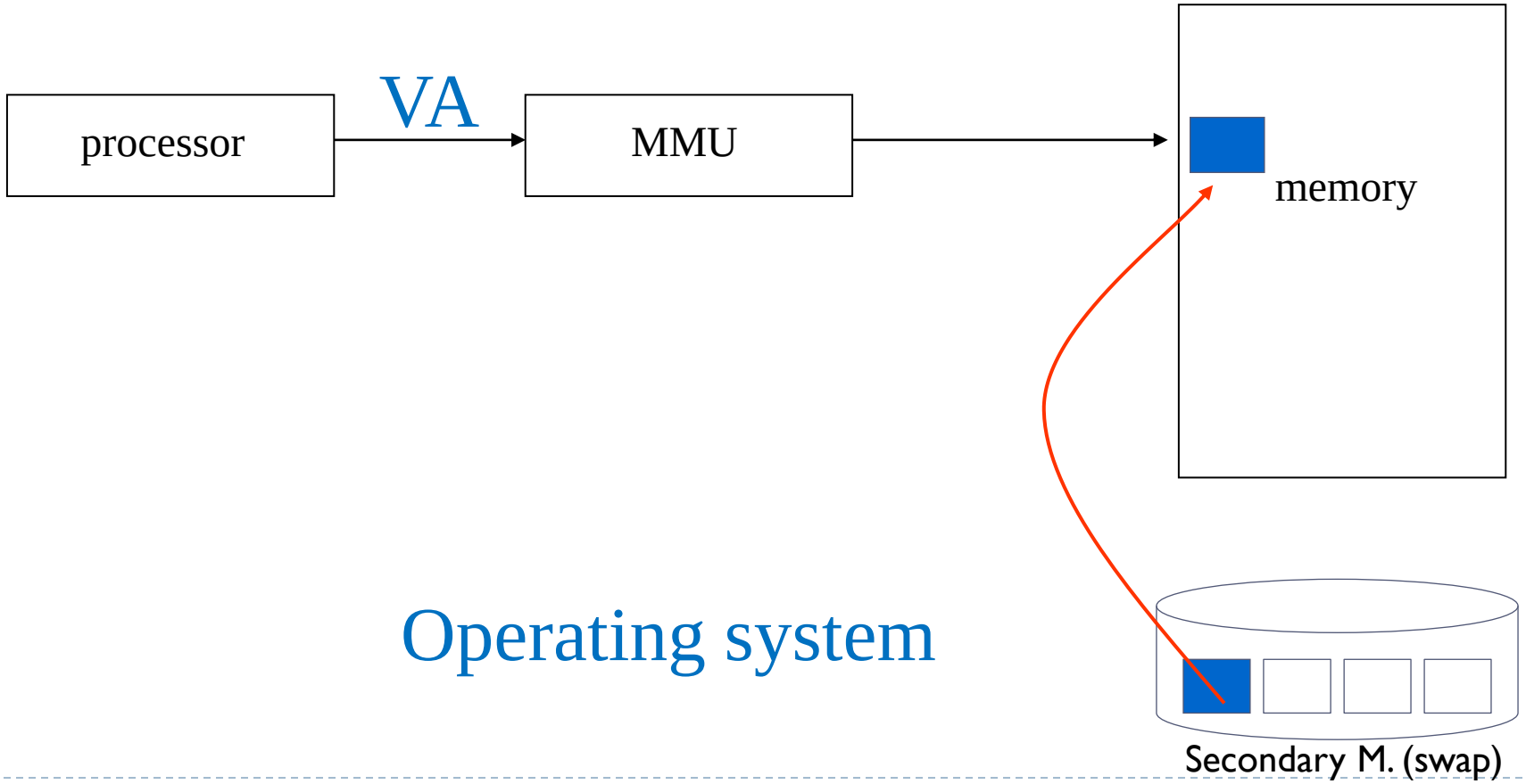
Address translation



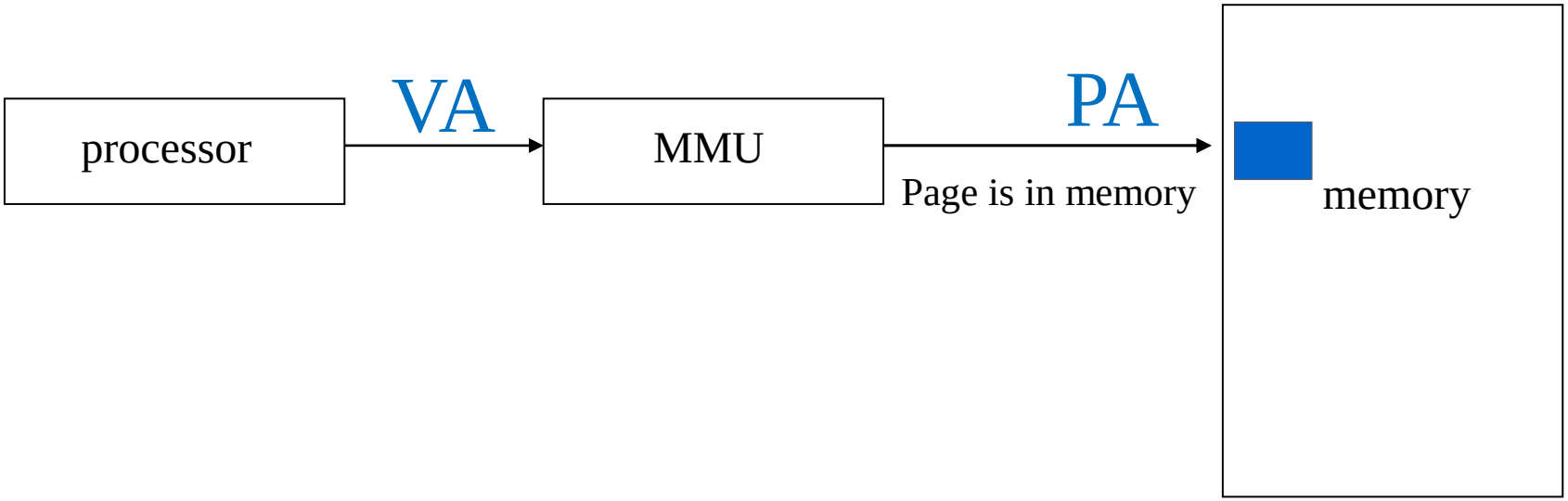
Address translation



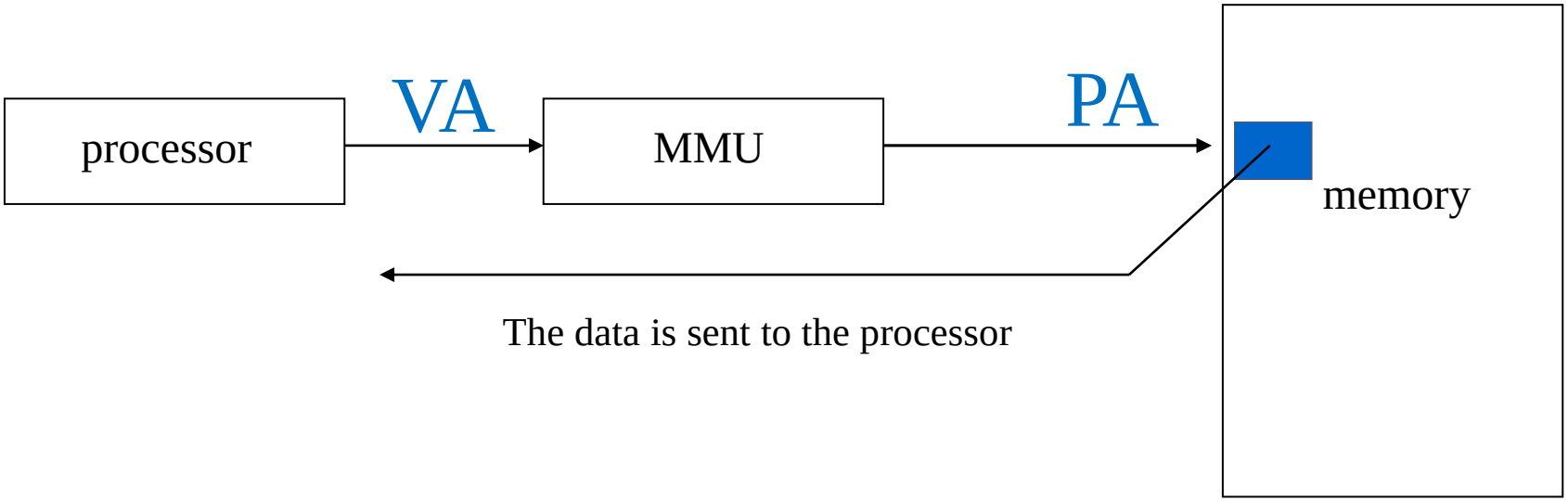
Address translation



Address translation



Address translation



Example

32 bits

01101000100101O

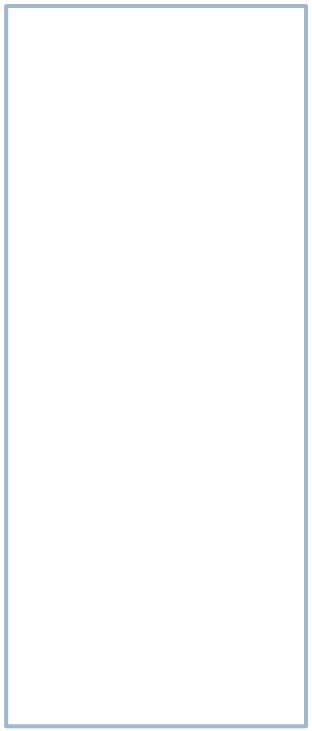
virtual address

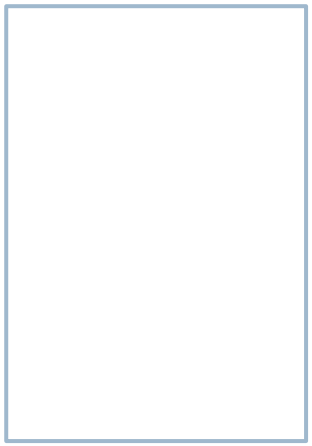
MMU

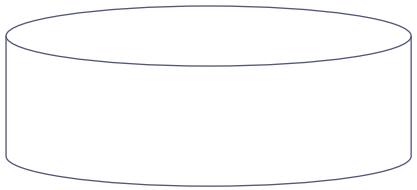
22 bits

1111100100000

Physical address/MM

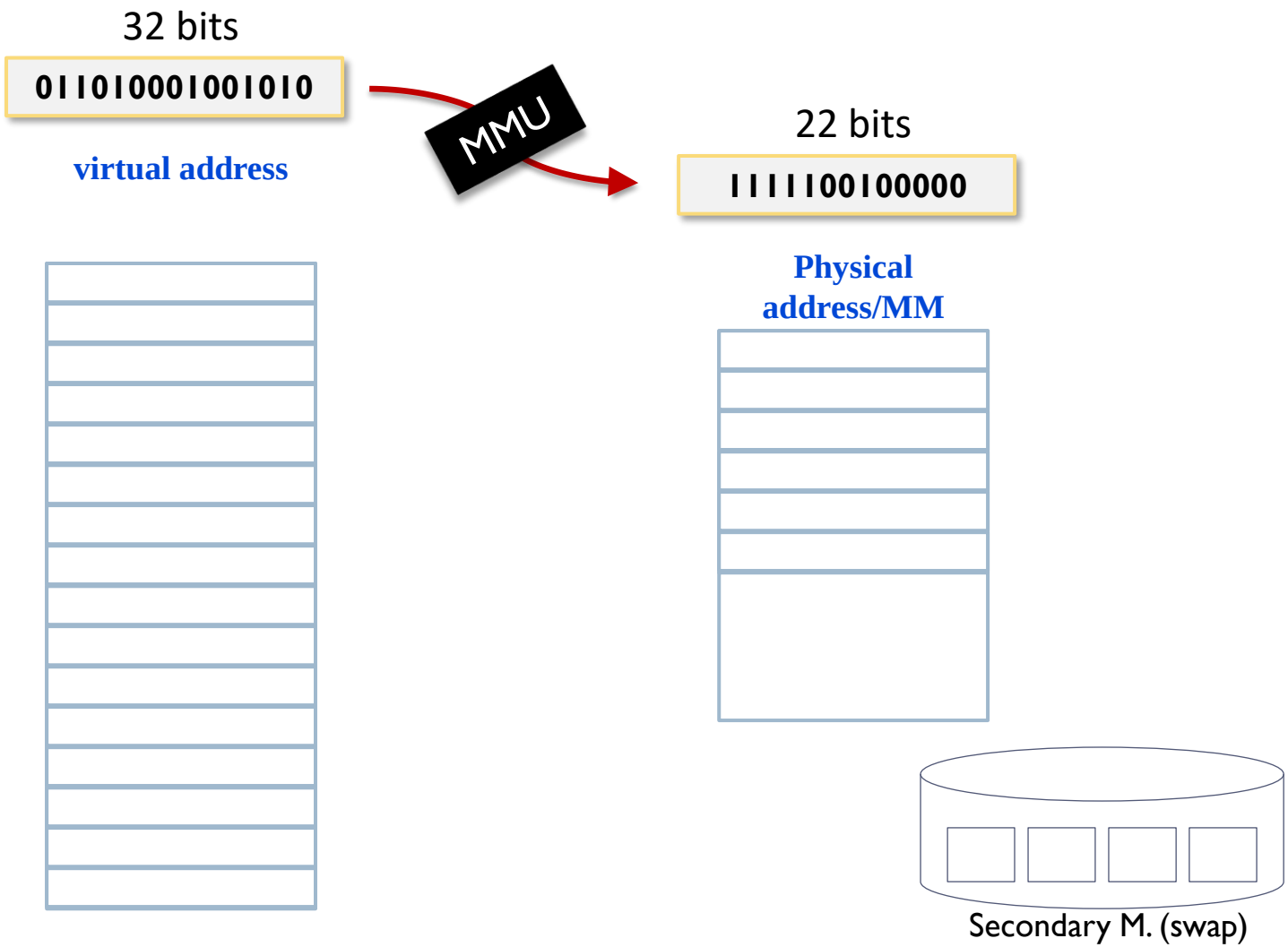






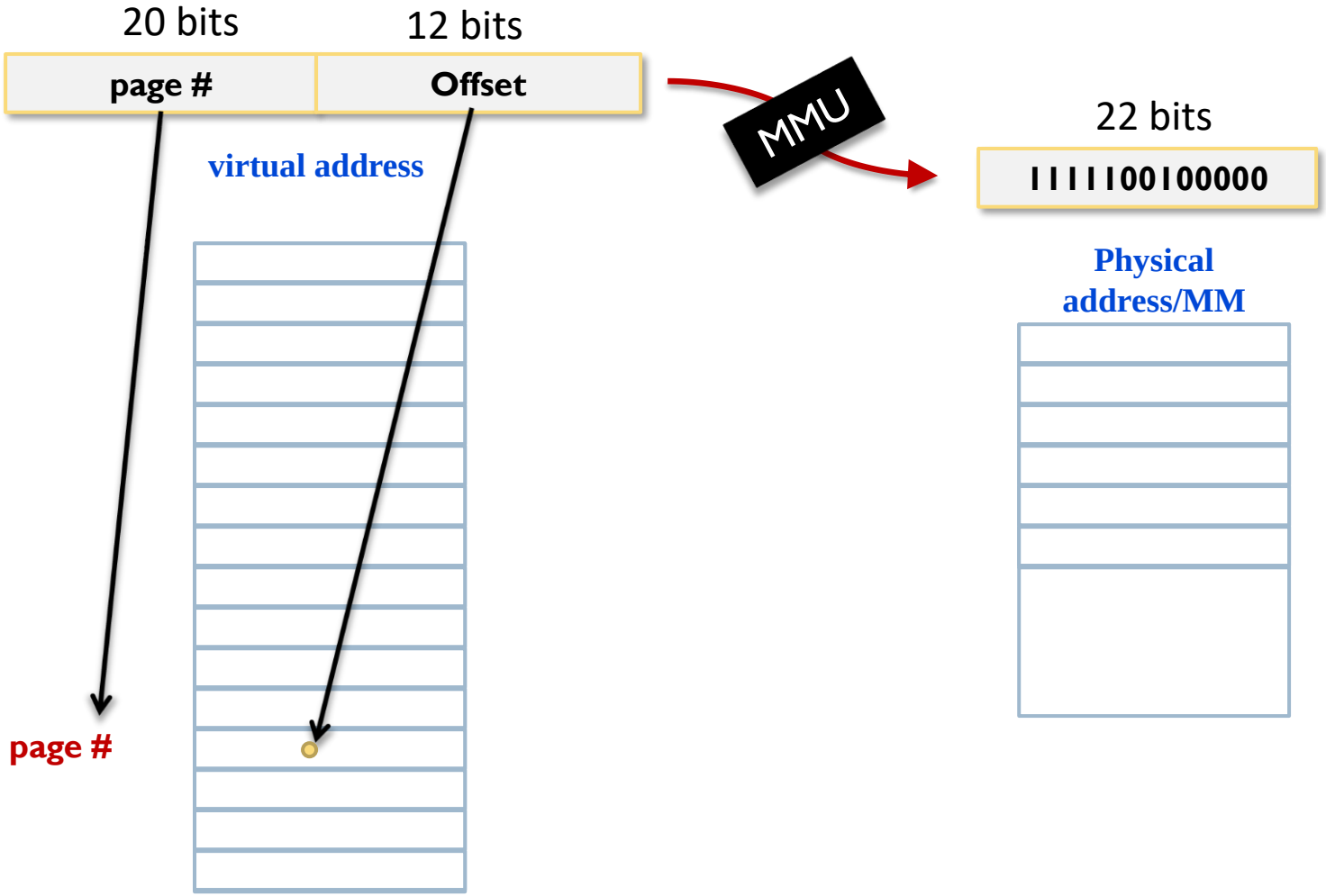
Secondary M. (swap)

Example



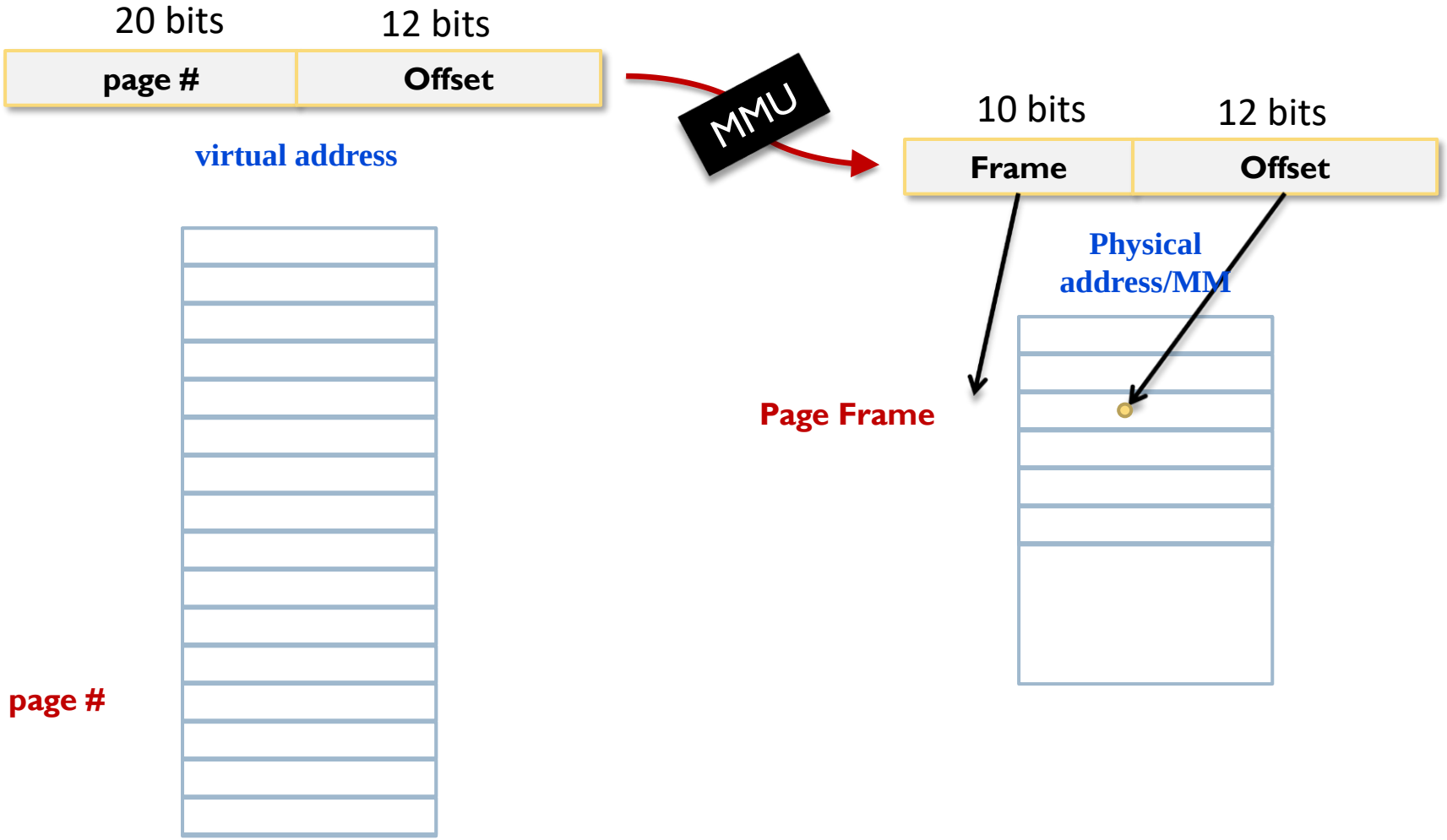
blocks with the same size -> pages

Example



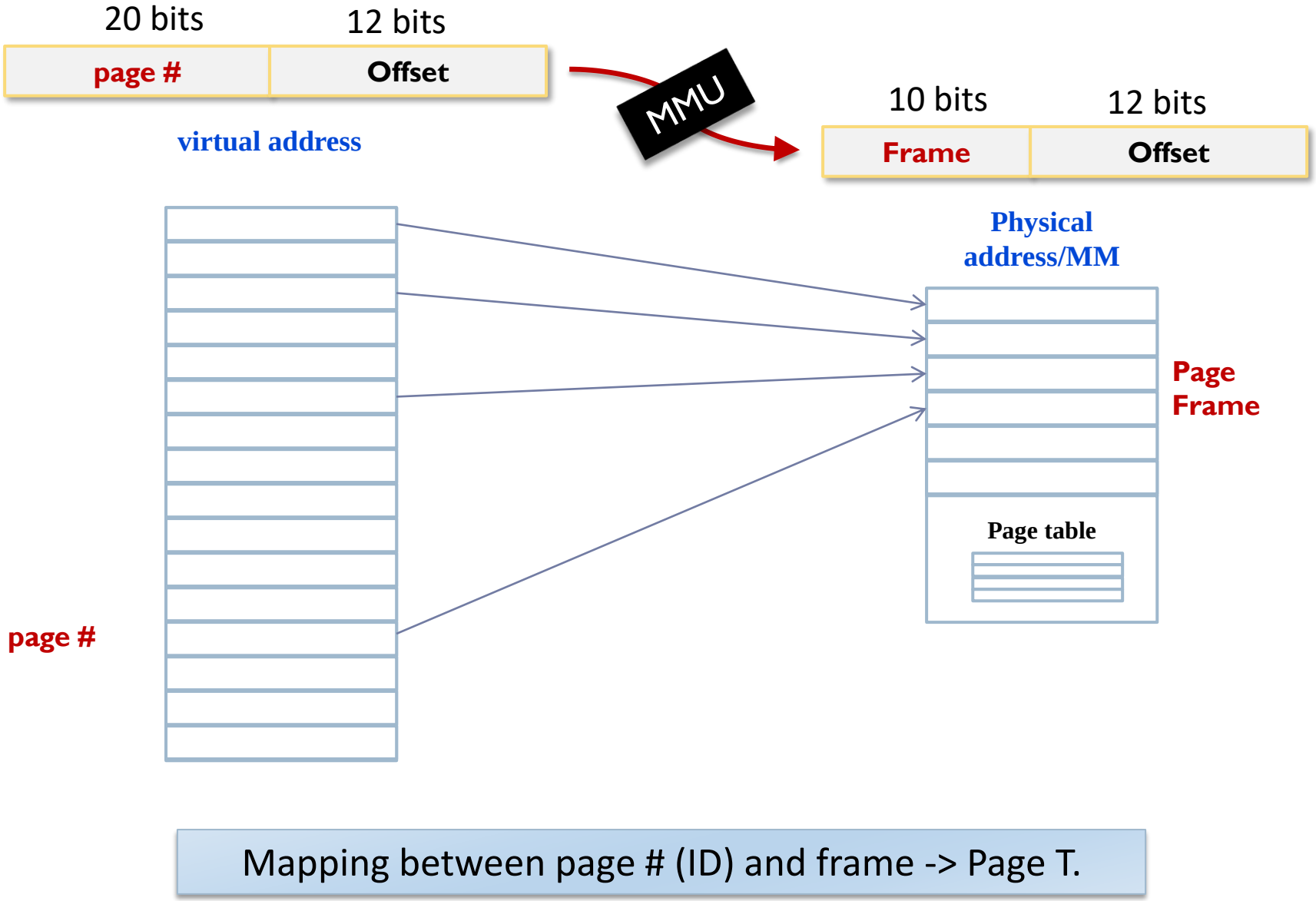
blocks with the same size -> pages

Example



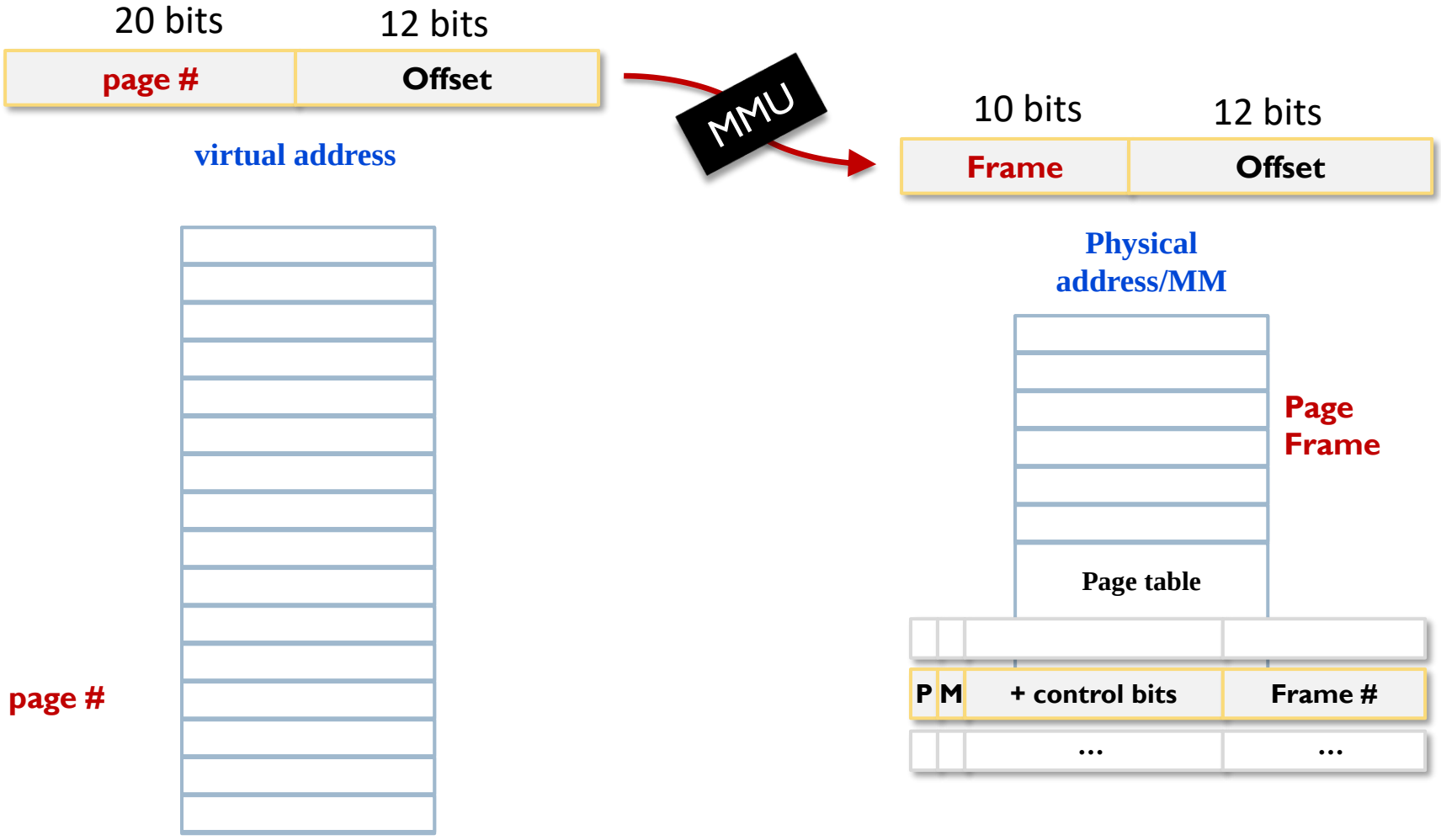
blocks with the same size -> pages

Example



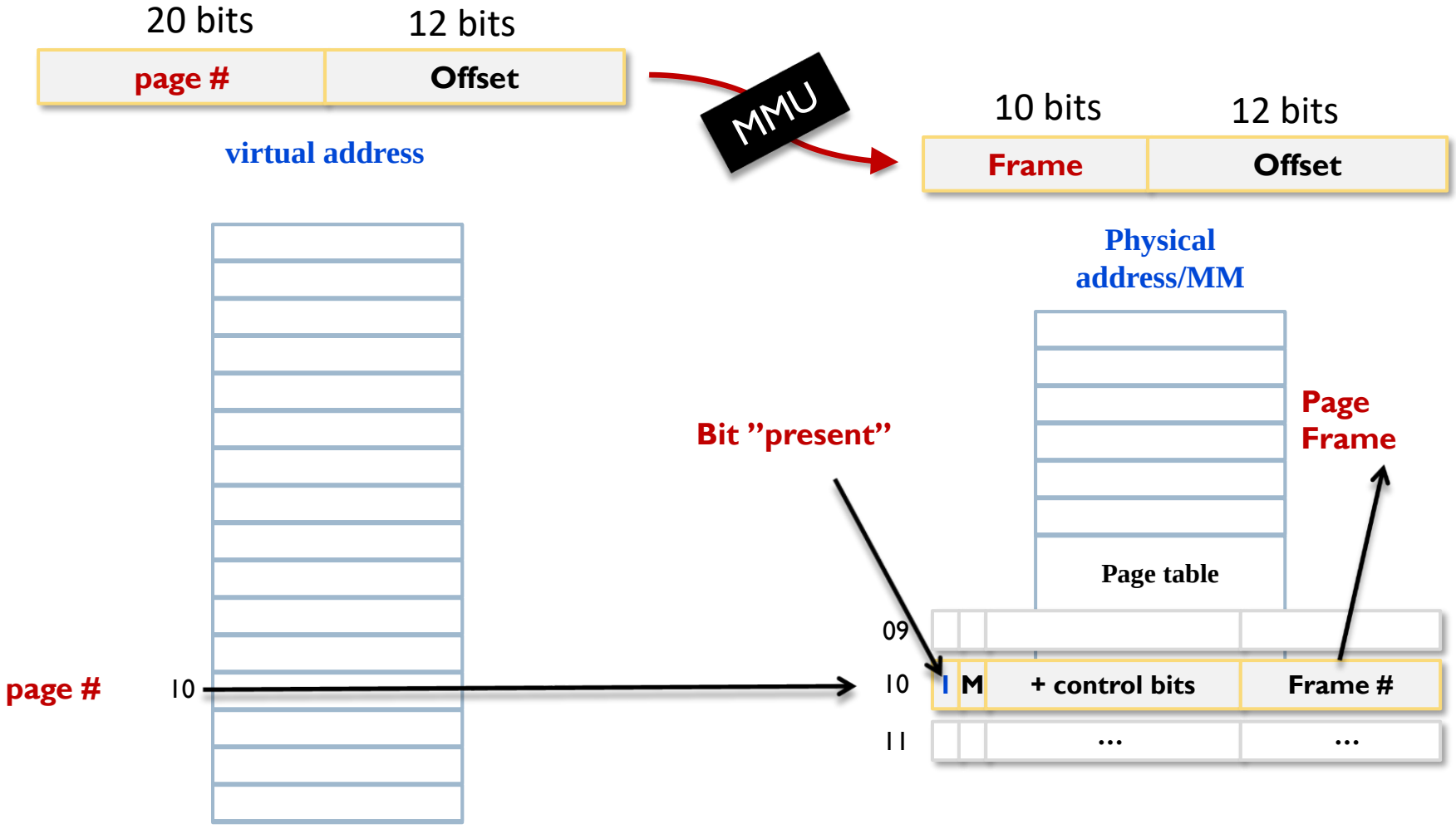
Mapping between page # (ID) and frame -> Page T.

Example

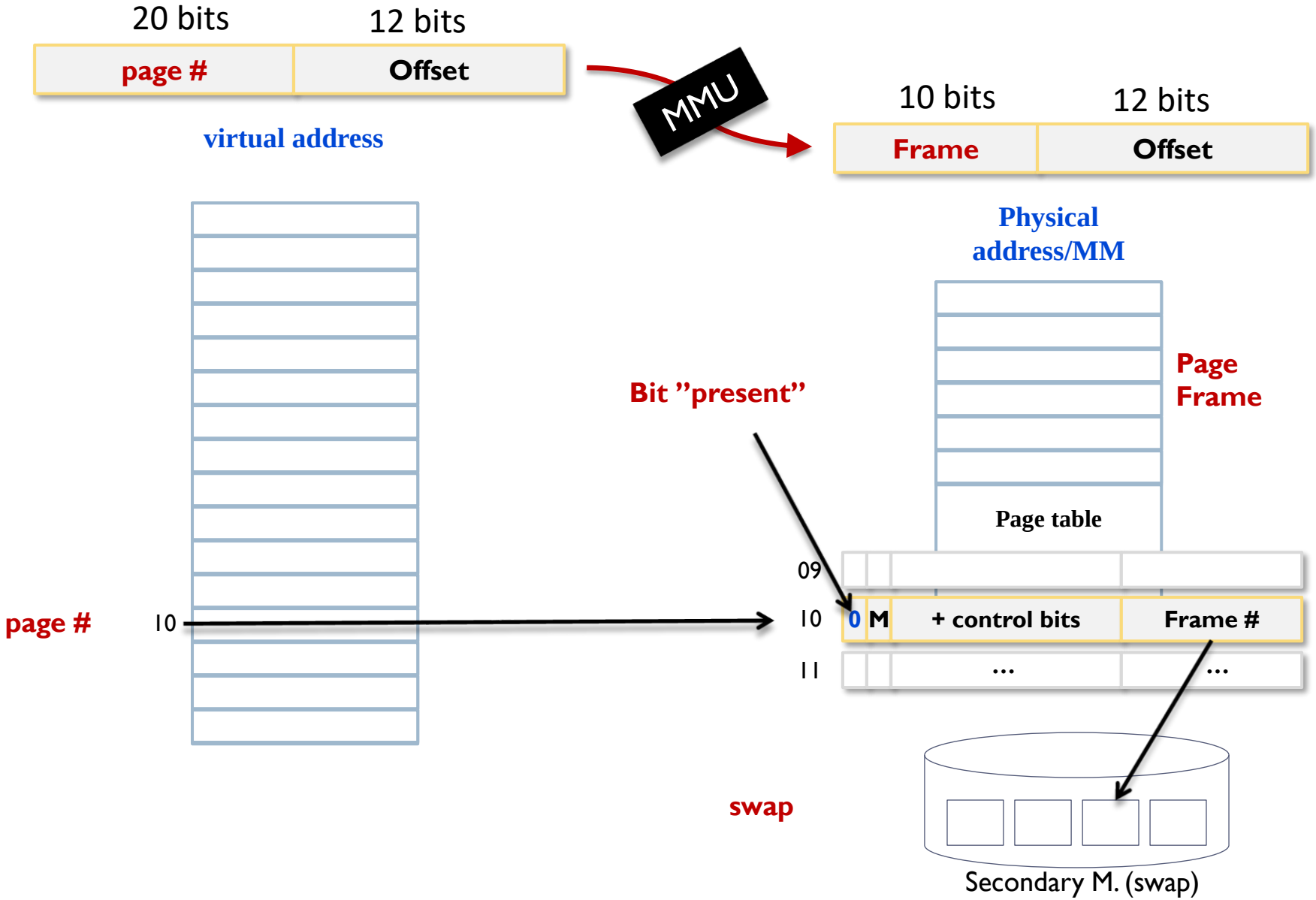


Mapping between page # (page ID) and frame -> Page Table

Example

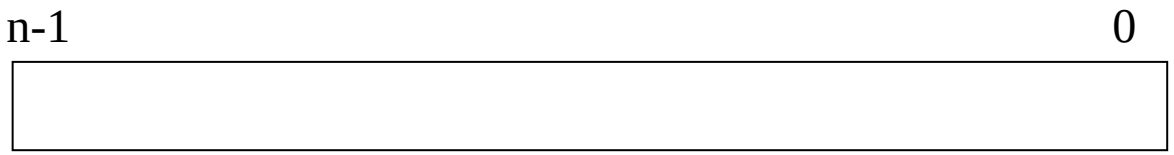

20 bits
12 bits
page #
Offset
virtual address
MMU
10 bits
12 bits
Frame
Offset
Physical address/MM
Bit "present"
Page Frame
Page table
page #
10
09
10
11
I
M
+ control bits
Frame #
...
...

Example


20 bits
12 bits
page #
Offset
virtual address
MMU
10 bits
12 bits
Frame
Offset
Physical address/MM
Page Frame
Bit "present"
Page table
page #
10
09
10
11
0
M
+ control bits
Frame #
...
...
swap
Secondary M. (swap)

Structure of a virtual address

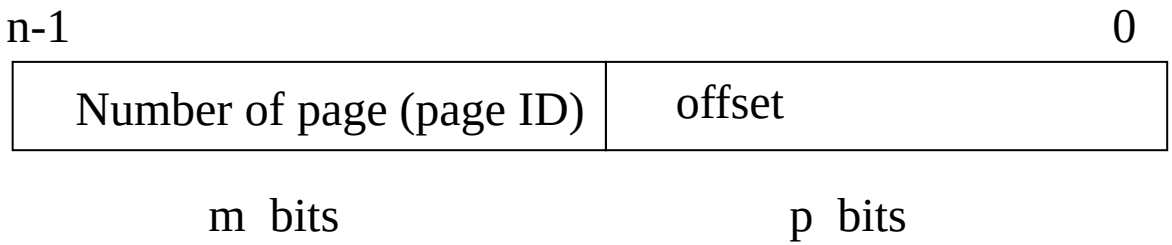
- A n-bits computer has:
  - Address of n bits



  - Can address up to 2^n bytes

Structure of a virtual address

- The memory image is composed of pages of equal size (2^P bytes)



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

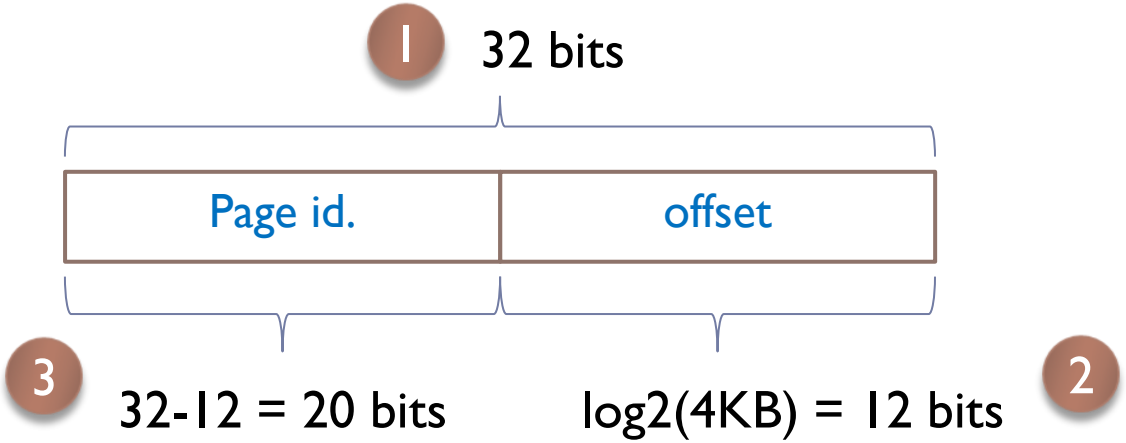
Exercise

We work with 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.

Exercise (solution)

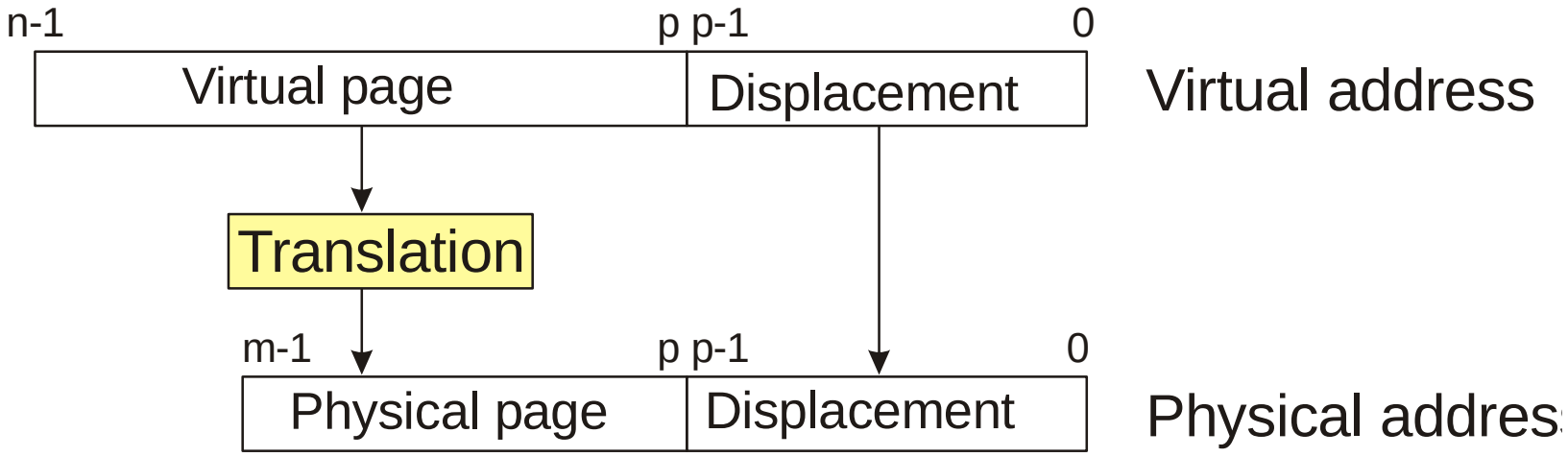
- Virtual address format:



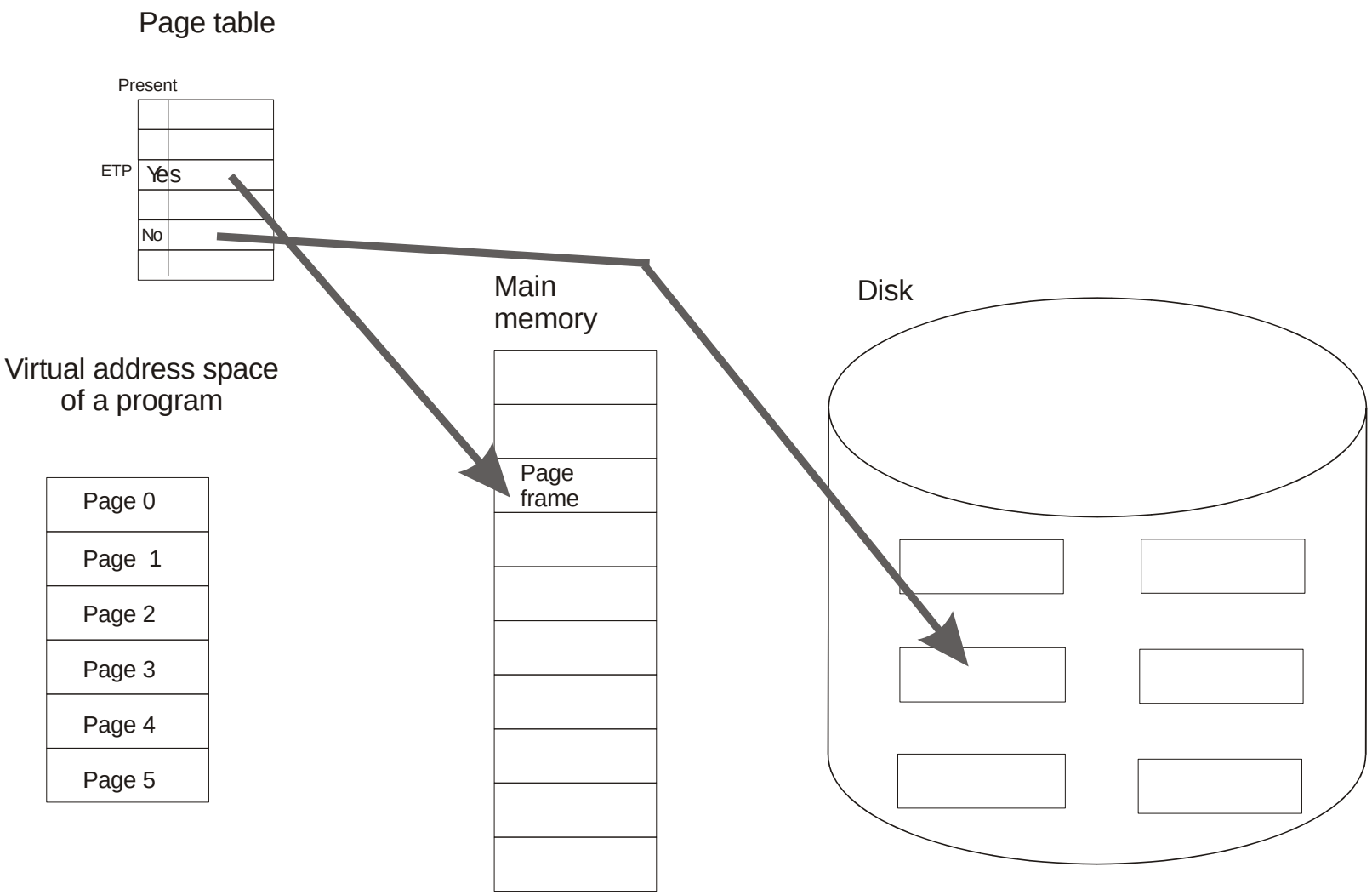
- Number of page frames:

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

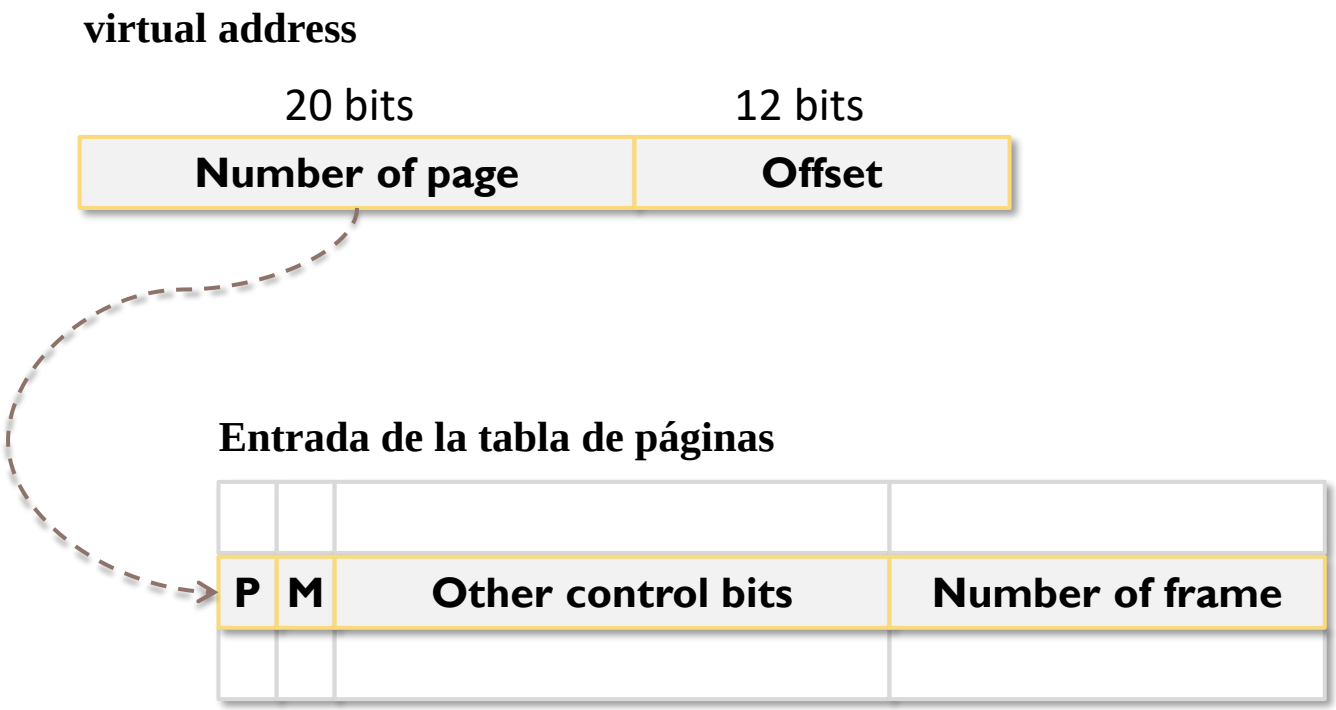
Paged Virtual memory
Page tables



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 entries (typical format)

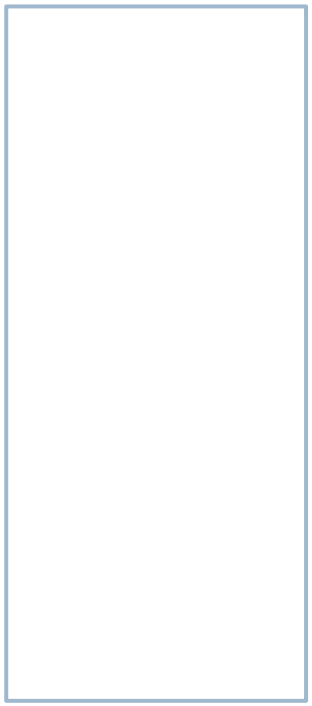


- P bit: indicates whether the page is present in M.M.
- M bit: indicates whether the page has been modified in M.M.
- Other bits: protection (read, write, execute, etc.), mgmr. (cow, etc.)

Page table structure

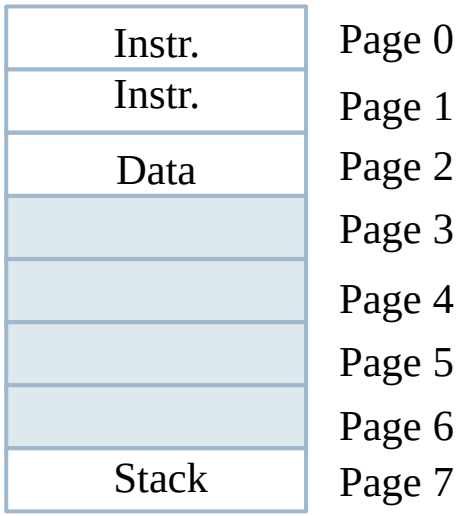
- Operating system creates the page table when a program is going to be executed
- The page table is accessed by the MMU in the translation process
- The page table is modified by the operating system when a page fail occurs

Example



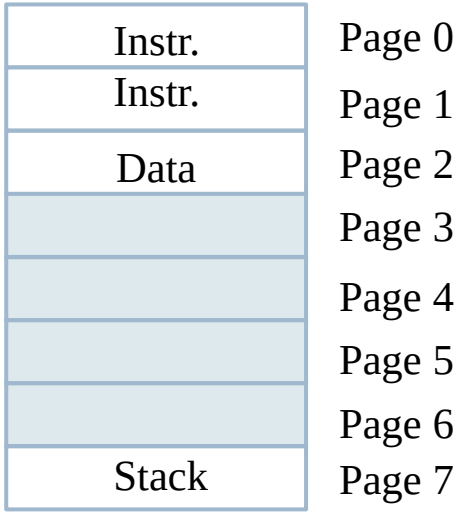
- Pages of 1 KB
- Process of 8 KB
  - Number of pages: 8
- Size of sections:
  - Instructions: 1.5 KB
  - Data: 1 KB
  - Stack: 0.2 KB

Example



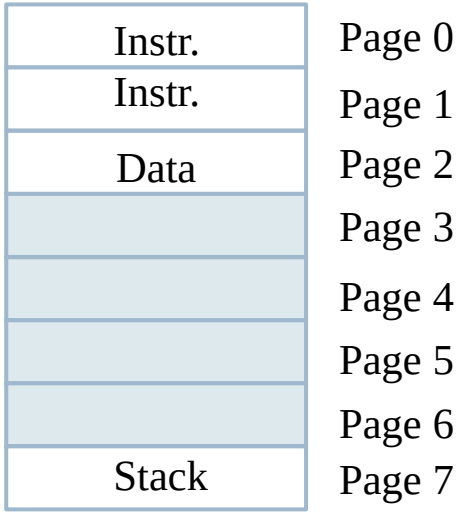
- Pages of 1 KB
- Process of 8 KB
  - Number of pages: 8
- Size of sections:
  - Instructions: 1.5 KB -> 2 pages
  - Data: 1 KB -> 1 page
  - Stack: 0.2 KB -> 1 page

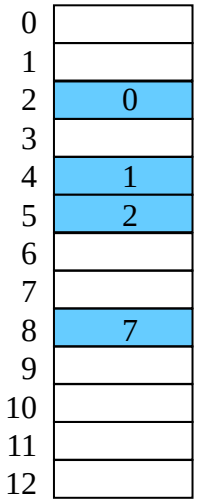
Example



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

Example
Process image initially in disk

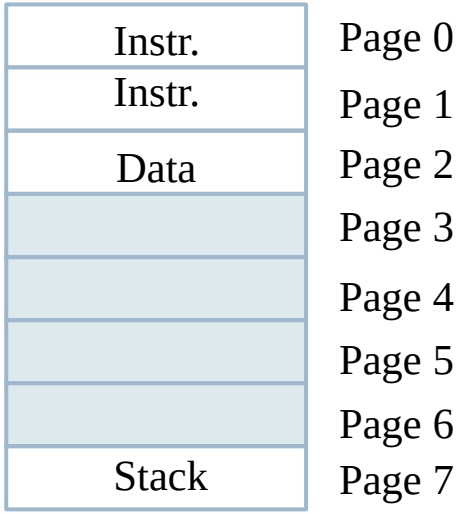




Swap

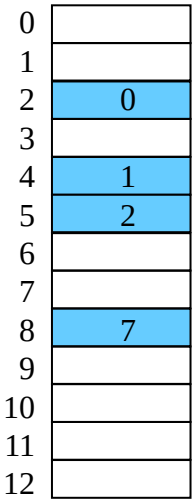
Pages of the process

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

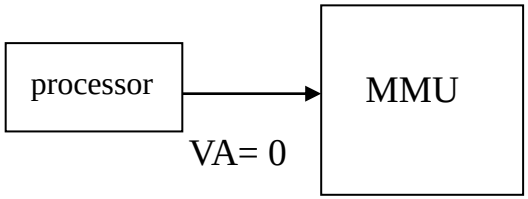
All pages in swap at the begining



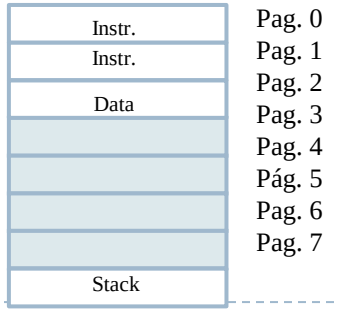
Swap

Pages of the process

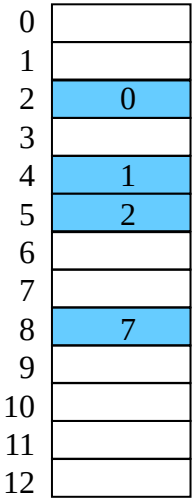
Example
Access to VA 0



	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



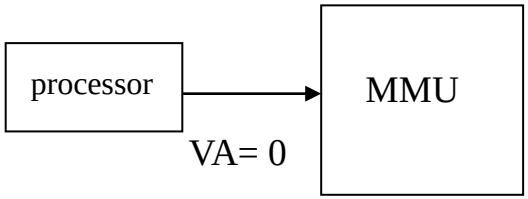
Pag. 0
Pag. 1
Pag. 2
Pag. 3
Pag. 4
Pág. 5
Pag. 6
Pag. 7



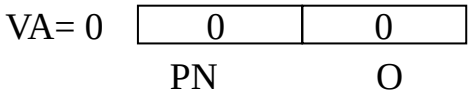
Swap

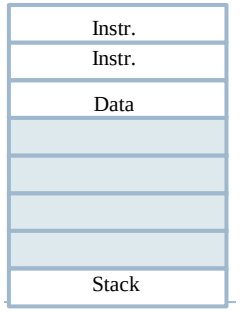
Pages of the process

Example
Access to VA 0

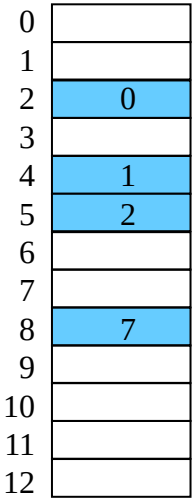


	P	M	frame/swap
0	0	0	2
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8





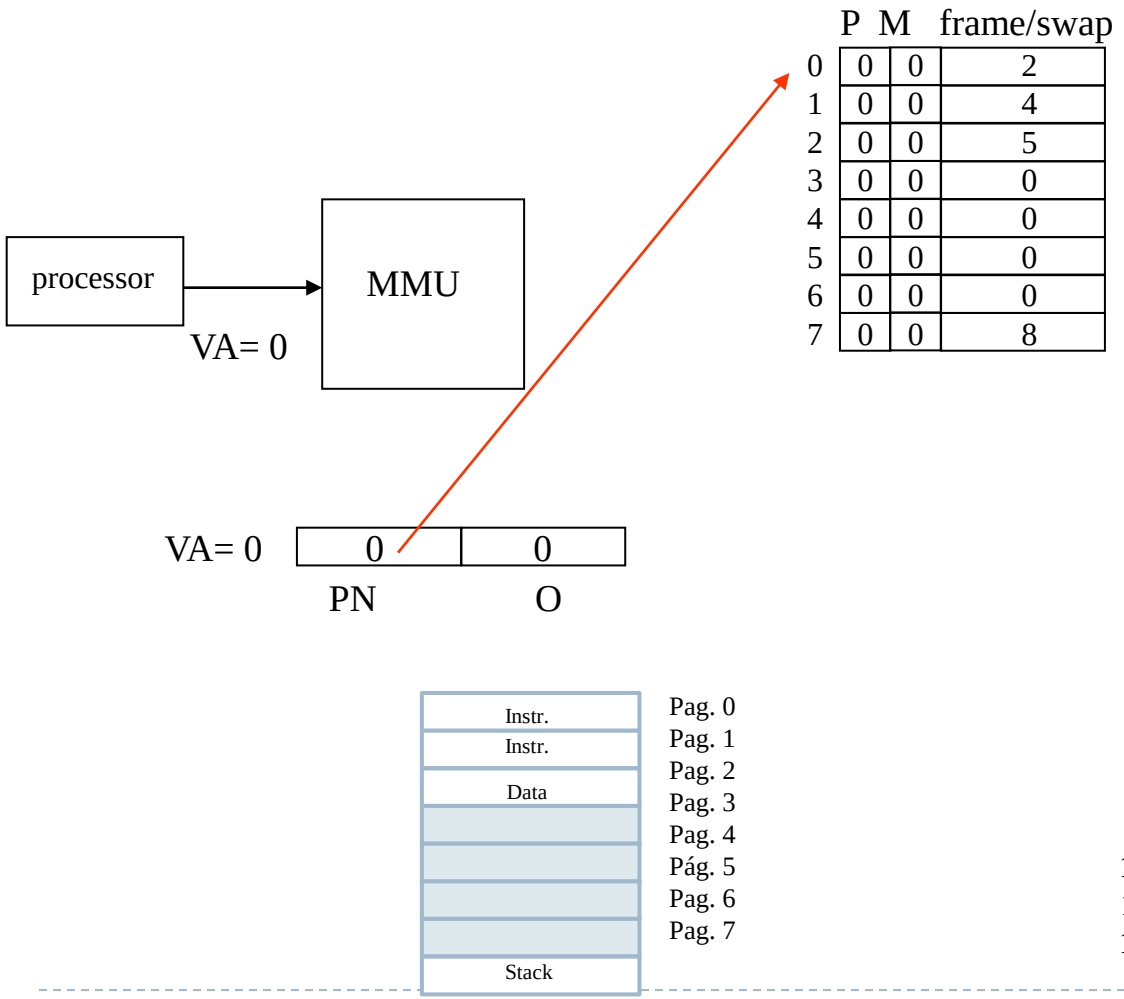
Pag. 0
Pag. 1
Pag. 2
Pag. 3
Pag. 4
Pág. 5
Pag. 6
Pag. 7



Swap

Pages of the process

Example
Access to VA 0

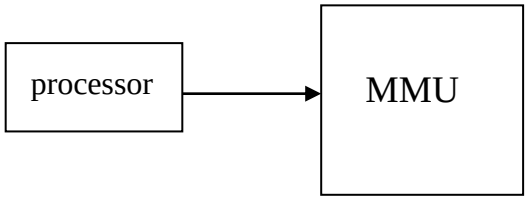


Page fault
Page 0 is not in memory

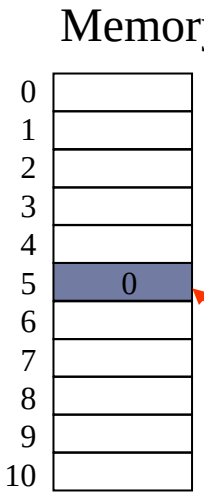
Swap

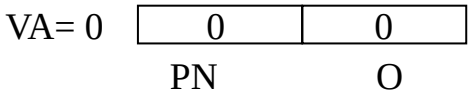
Pages of the process

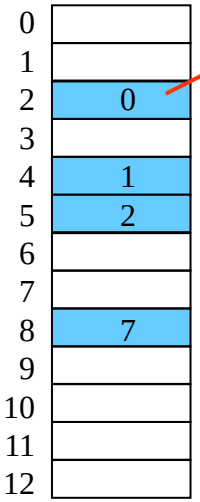
Example handling the page fault



	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





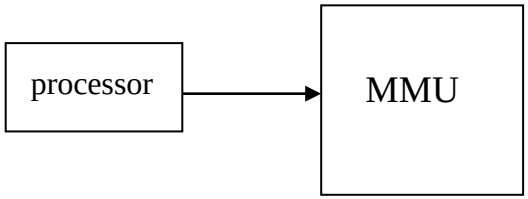


Swap

Pages of the process

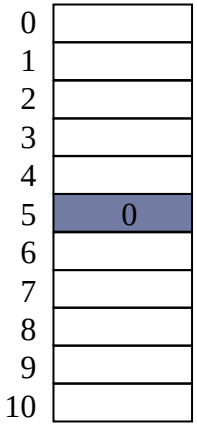
The O.S. reserves a free page frame in memory (5) and copies the block 2 in the frame 5

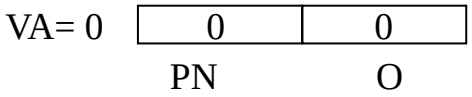
Example handling the page fault



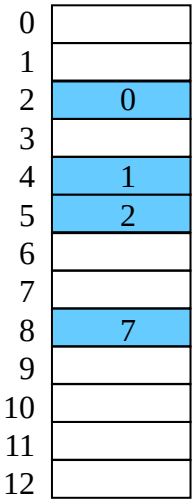
	P	M	frame/swap
0	1	0	5
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

Memory





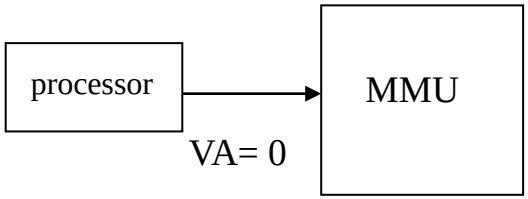
The O.S. updates the page table



Swap

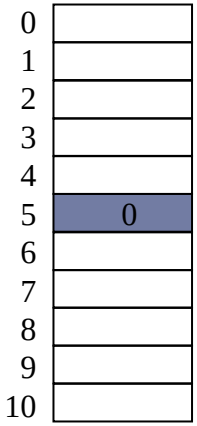
Pages of the process

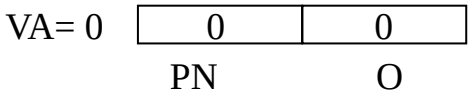
Example
Resuming the process



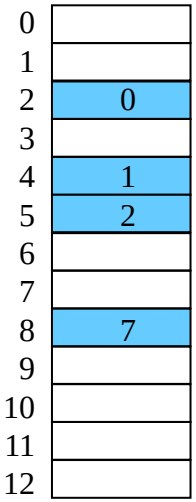
	P	M	frame/swap
0	1	0	5
1	0	0	4
2	0	0	5
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	8

Memory





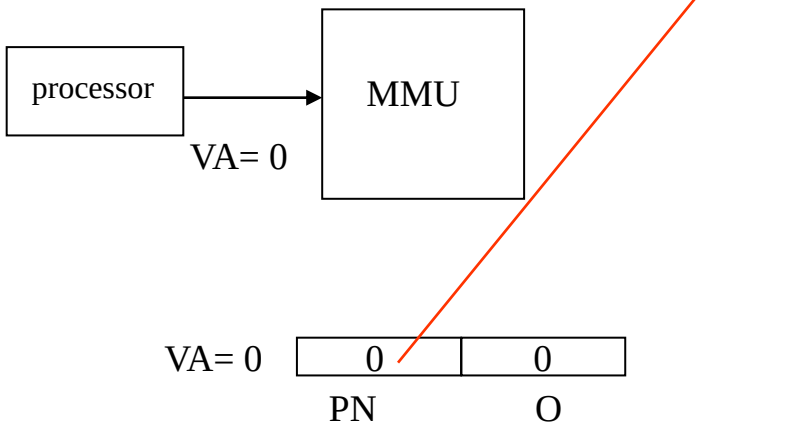
VA 0 is generated again



Swap

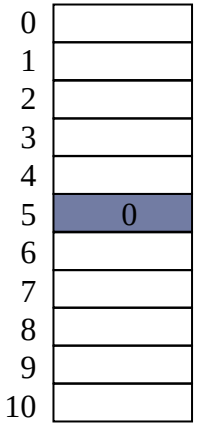
Pages of the process

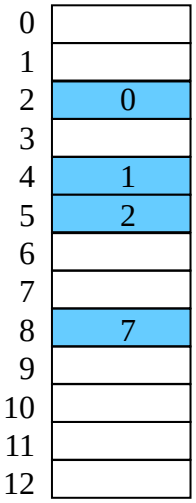
Example
Resuming the process



	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



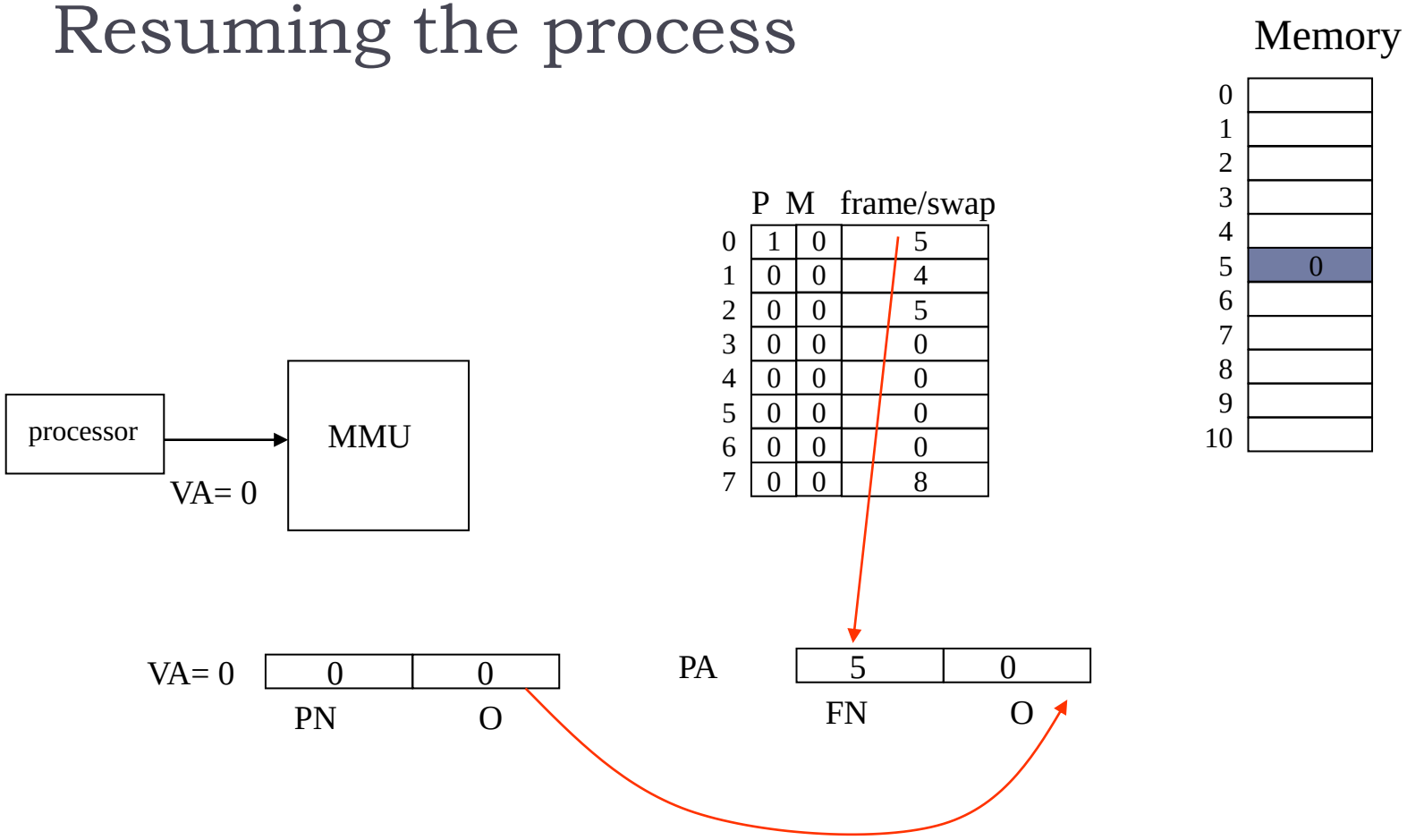


Swap

Pages of the process

VA 0 is generated again

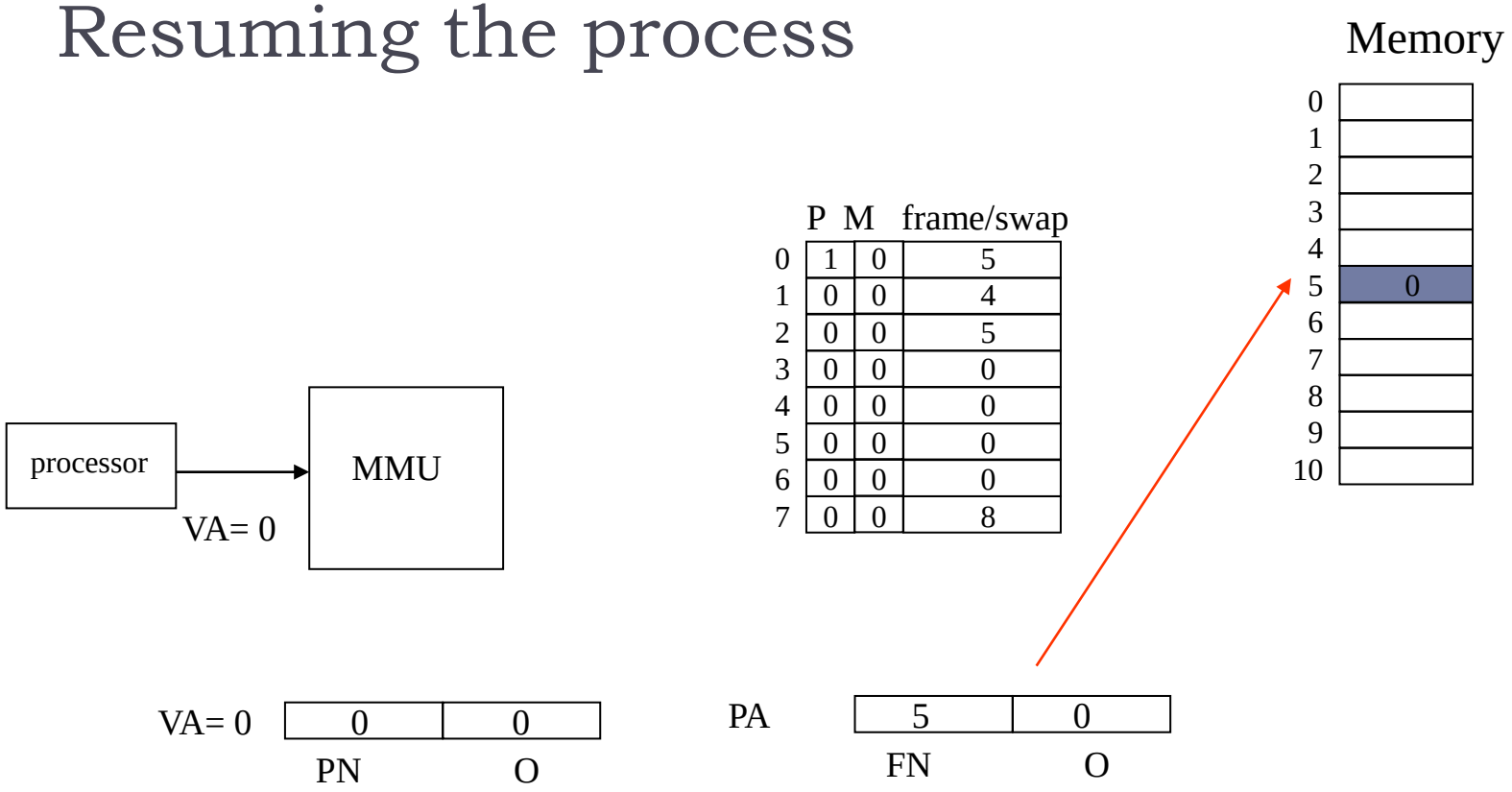
Example
Resuming the process



Page in memory

Obtain the physical address

Example
Resuming the process



Access to memory

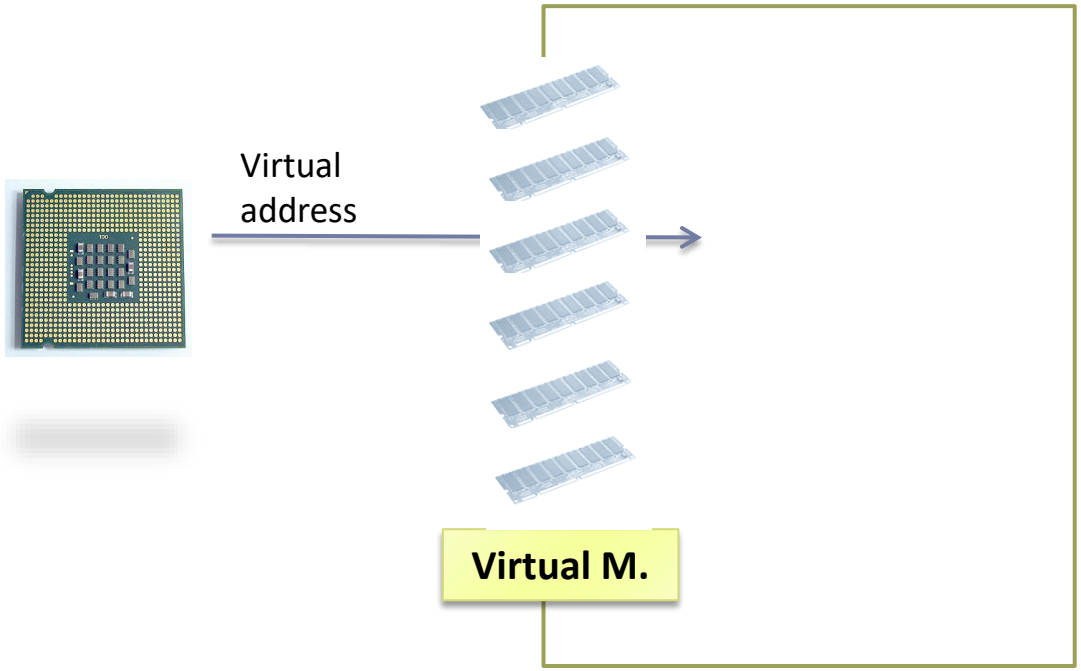
Exercise

A computer that addresses memory by byte uses 32-bit virtual addresses.
Each entry in the page table requires 32 bits, and the system uses 4 KB pages.

- Answer:
  a) What is the addressable memory space for a running program?
  b) What is the maximum page table size on this computer?

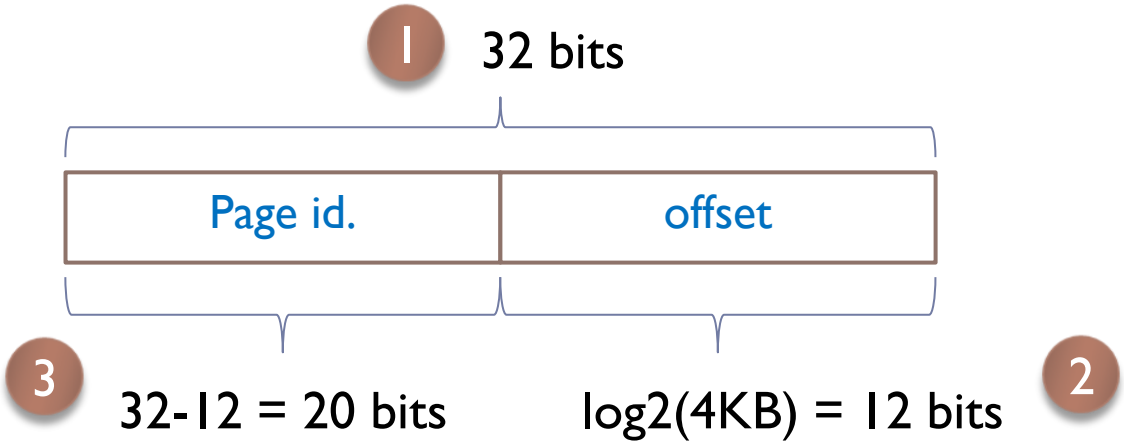
Exercise (solution)

- The memory space addressable by a running program is determined by the number of bits of the virtual address:
  - 2^{32} = 4 GiB



Exercise (solution)

- The size of the page table will depend on the maximum number of page frames and the size of each table entry:
  - 2^{20} * 4 bytes (32 bits) = 4 MB



4 If there is as much main memory as virtual memory, the page frame identifiers will also be 20 bits long.

Exercise

Let be a computer with 32-bit virtual addresses and 4 KB pages.
In this computer is executed a program whose page table is:

P	M	Perm.	Frame/ Block
0	0	R	1036
1	0	R	4097
0	0	W	3000
0	0	W	7190
0	0	W	3200
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	W	2400
0	0	W	3000

- Please answer:
  a) Size occupied by the program memory image.
  b) If the first virtual address of the program is 0x00000000, enter the last virtual address of the program.
  c) Given the following virtual addresses, indicate whether they generate page fault or not:
     - 0x00001000
     - 0x0000101C
     - 0x00004000

Exercise (solution)

P	M	Perm.	Frame/ Block
0	0	R	1036
1	0	R	4097
0	0	W	3000
0	0	W	7190
0	0	W	3200
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	W	2400
0	0	W	3000

- The size of the program's memory image will depend on the total number of pages assigned to it and the size of the page:
  - 7 * 4 KB = 28 KB

Exercise (solution)

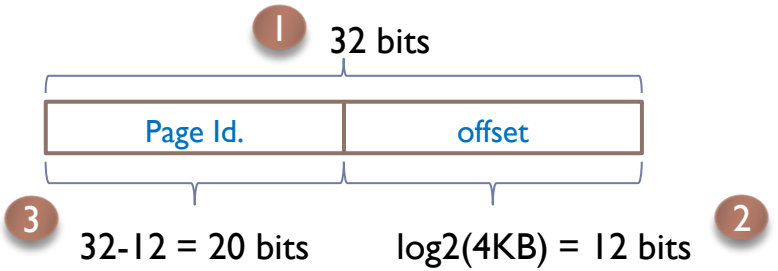
P	M	Perm.	Frame/ Block
0	0	R	1036
1	0	R	4097
0	0	W	3000
0	0	W	7190
0	0	W	3200
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	W	2400
0	0	W	3000

- If the total size of the program is 28 KB and the first virtual address is 0x00000000,
  the last address will be :
  - 28 * 1024 - 1

Exercise (solution)

P	M	Perm.	Frame/ Block
0	0	R	1036
1	**0**	**R**	**4097**
0	0	W	3000
0	0	W	7190
0	**0**	**W**	**3200**
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	W	2400
0	0	W	3000

- The first thing to do is to know the format of the virtual address:

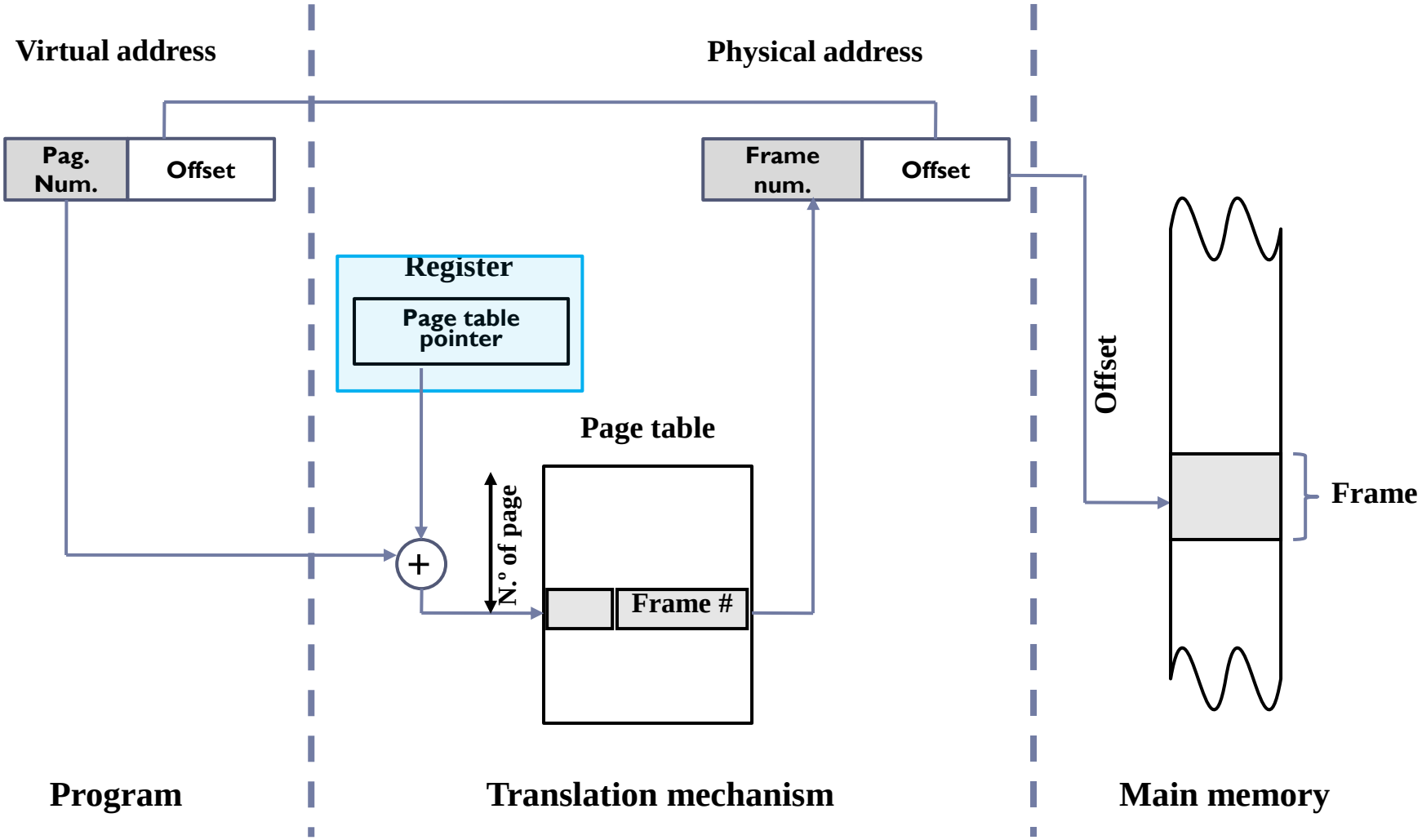


- For each virtual address, extract the page identifier, search the Page table for its entry, and see if the present bit (P) is set to 1:
  - 0x**0000**1000 -> no
  - 0x**0000**101C -> no
  - 0x**0000**4000 -> yes

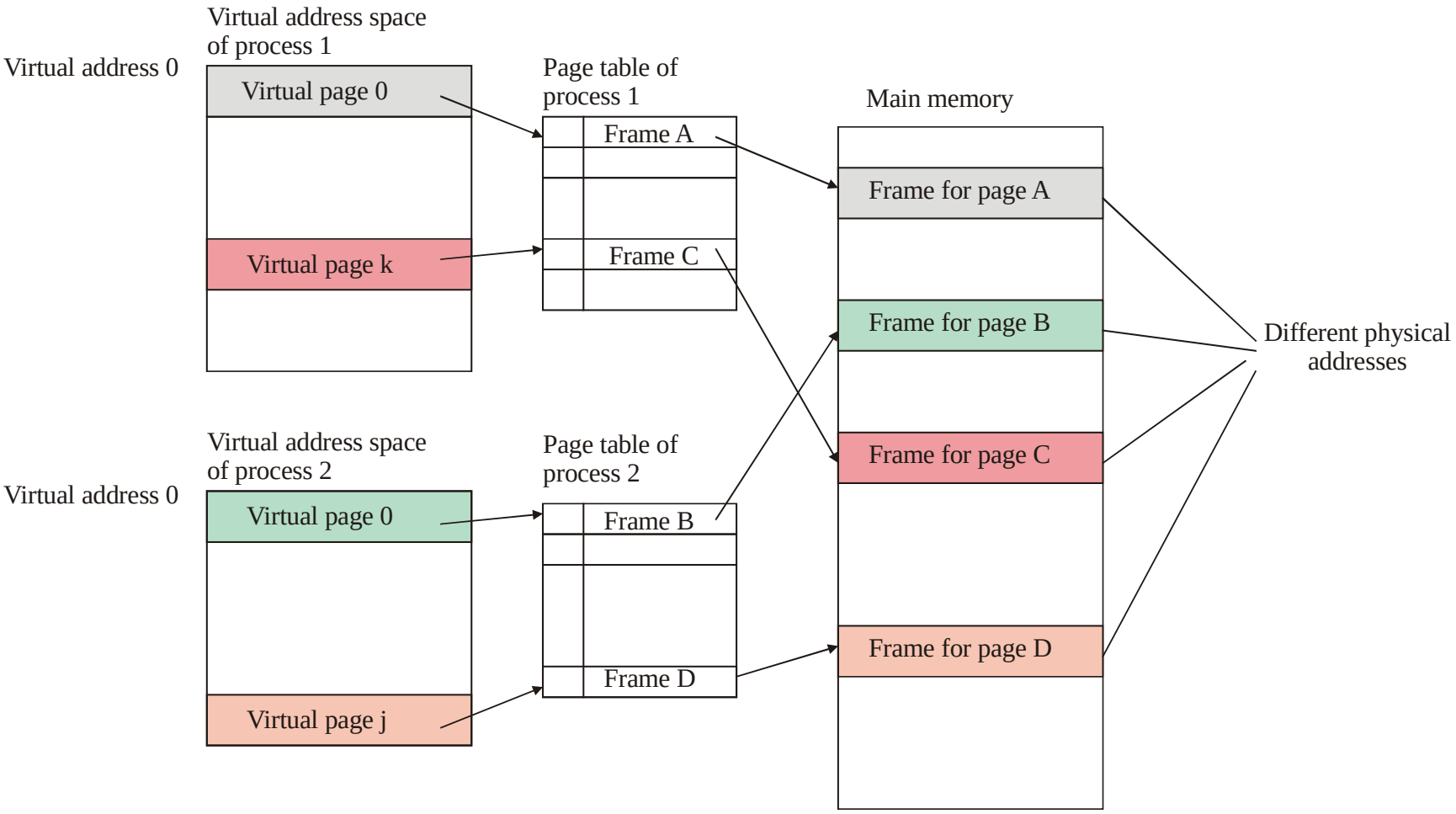
Page table management

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

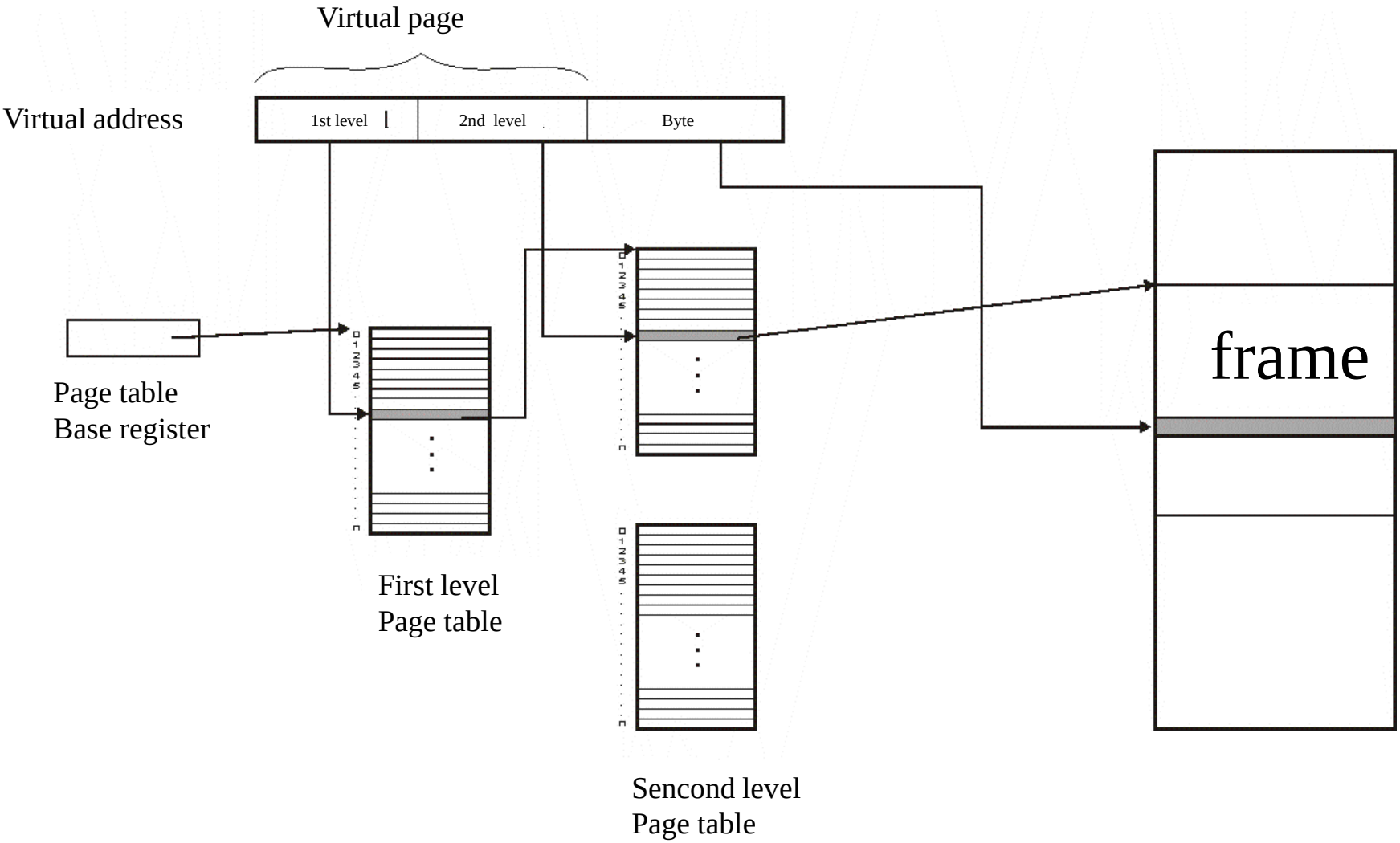
Translation



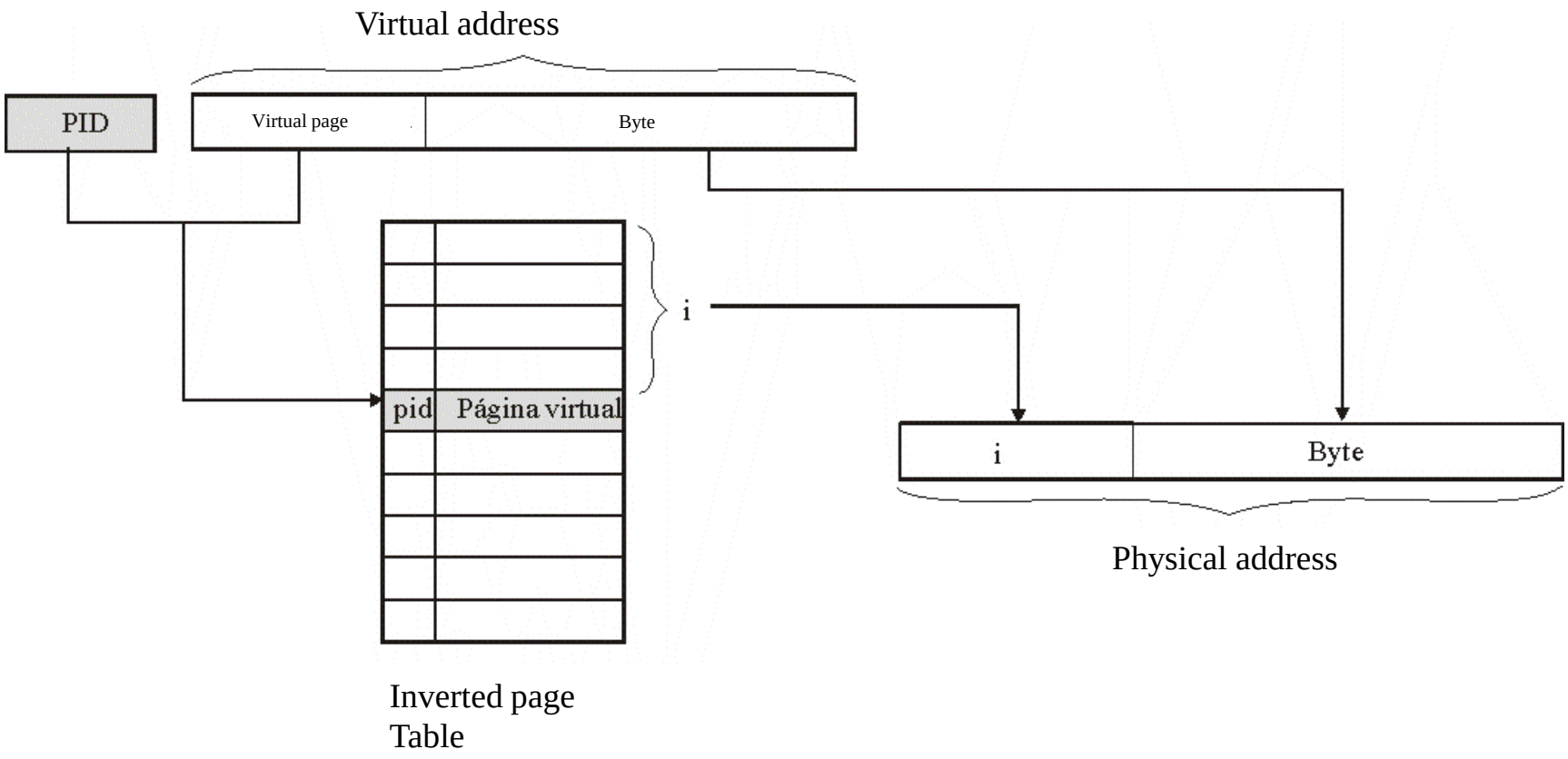
Memory protection



Two-level page table



Inverted page table

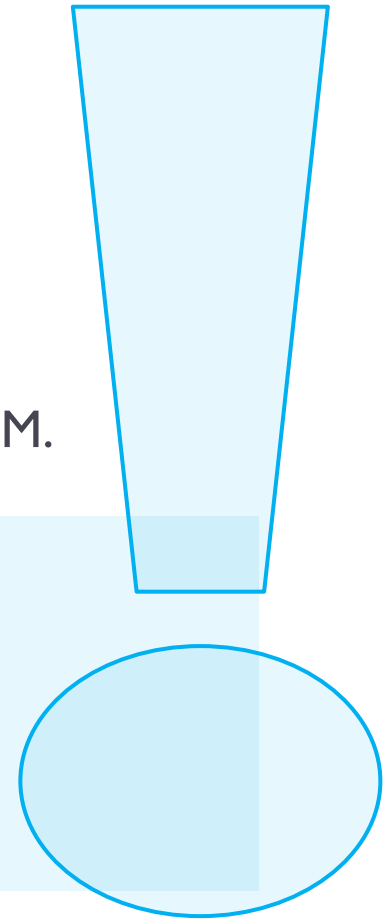


Page movement

- Initially:
  - Non-resident page is marked absent
  - The address of the swap block containing it is saved
- Secondary M. to Main M. (on demand):
  - Access to non-resident page: Page failure
  - O.S. reads page from Secondary M. and takes it to Main M.
- Main M. to Secondary M. (by expulsion):
  - There is no space in Main M. to bring in page
  - A resident page is replaced (stealing)
  - O.S. writes replaced page to Secondary M. (if bit M=1)

Page movement

- Initially:
  - Non-resident page is marked absent
  - The address of the swap block containing it is saved
- Secondary M. to Main M. (on demand):
  - Access to non-resident page: Page failure
  - O.S. reads page from Secondary M. and takes it to Main M.
- Main M. to Secondary M. (by expulsion):
  - There is no space in Main M. to bring in page
  - A resident page is replaced (stealing)
  - O.S. writes replaced page to Secondary M. (if bit M=1)



Replacement policies

- Which page is to be replaced (operating system)
- The page to be replaced must be the one that has the least chance of being referenced in the near future.
- Most policies attempt to predict future behavior on the basis of past behavior.
- Example of policies: **LRU, FIFO, etc.**

Non-replacement policies

- Frame locking (pinned pages):
  - When a frame is locked, the page loaded in that frame cannot be replaced.
- Examples of when a frame is pinned:
  - Most of the operating system kernel.
  - Control structures.
  - I/O buffers.
- **Pinning is achieved by associating a lock bit to each frame.**

Translation cache
TLB (*Translation Lookaside Buffer*)

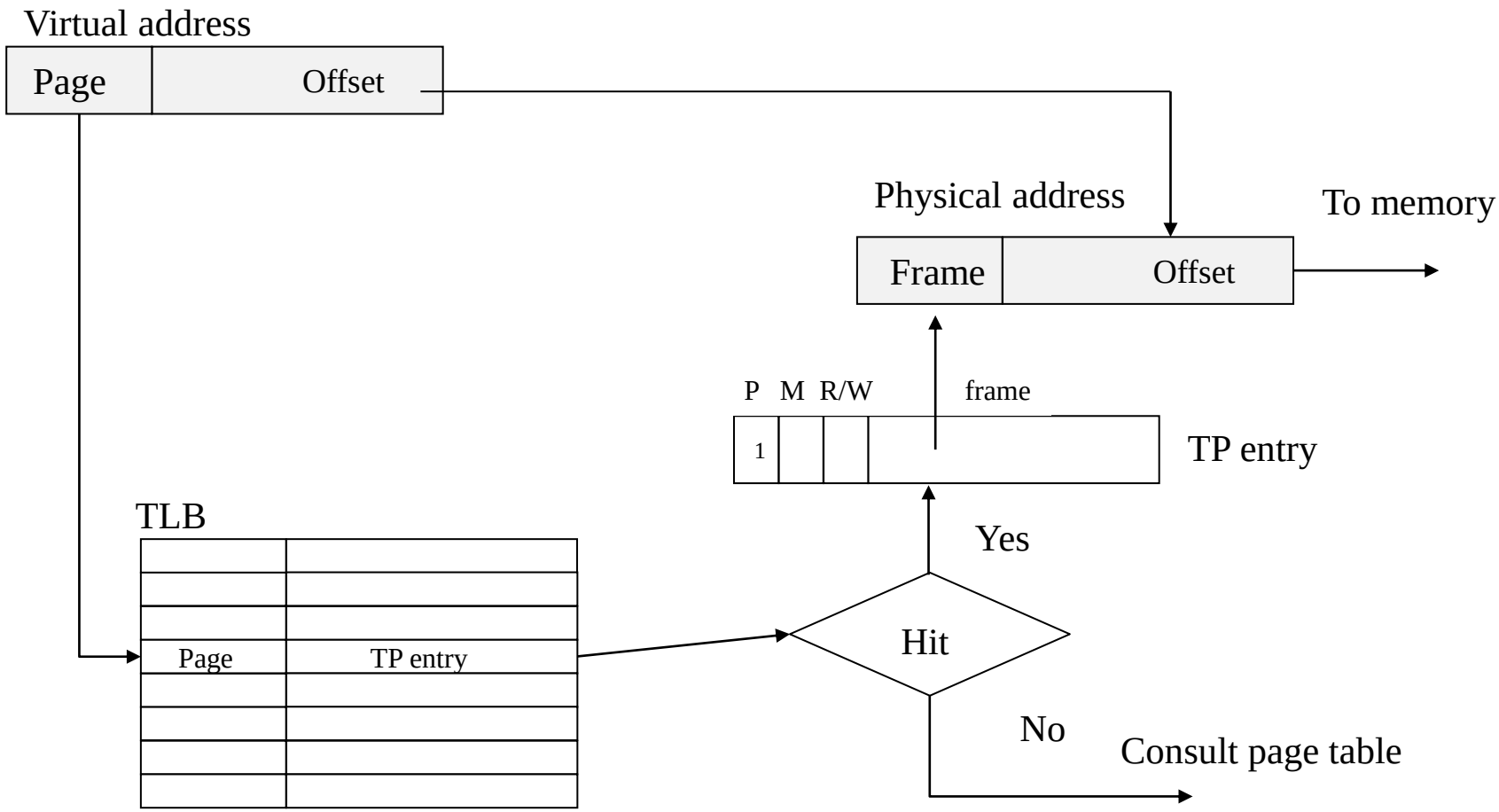
- Virtual memory based on page tables:
  - Problem: memory access overhead (2 access)
    - One to the page table that resides in MM
    - Another to the page containing the data
  - Solution: TLB.

- TLB: forward translation buffer:
  - Associative cache memory that stores the most recently used page table entries.
  - Allows to speed up the frame search process.

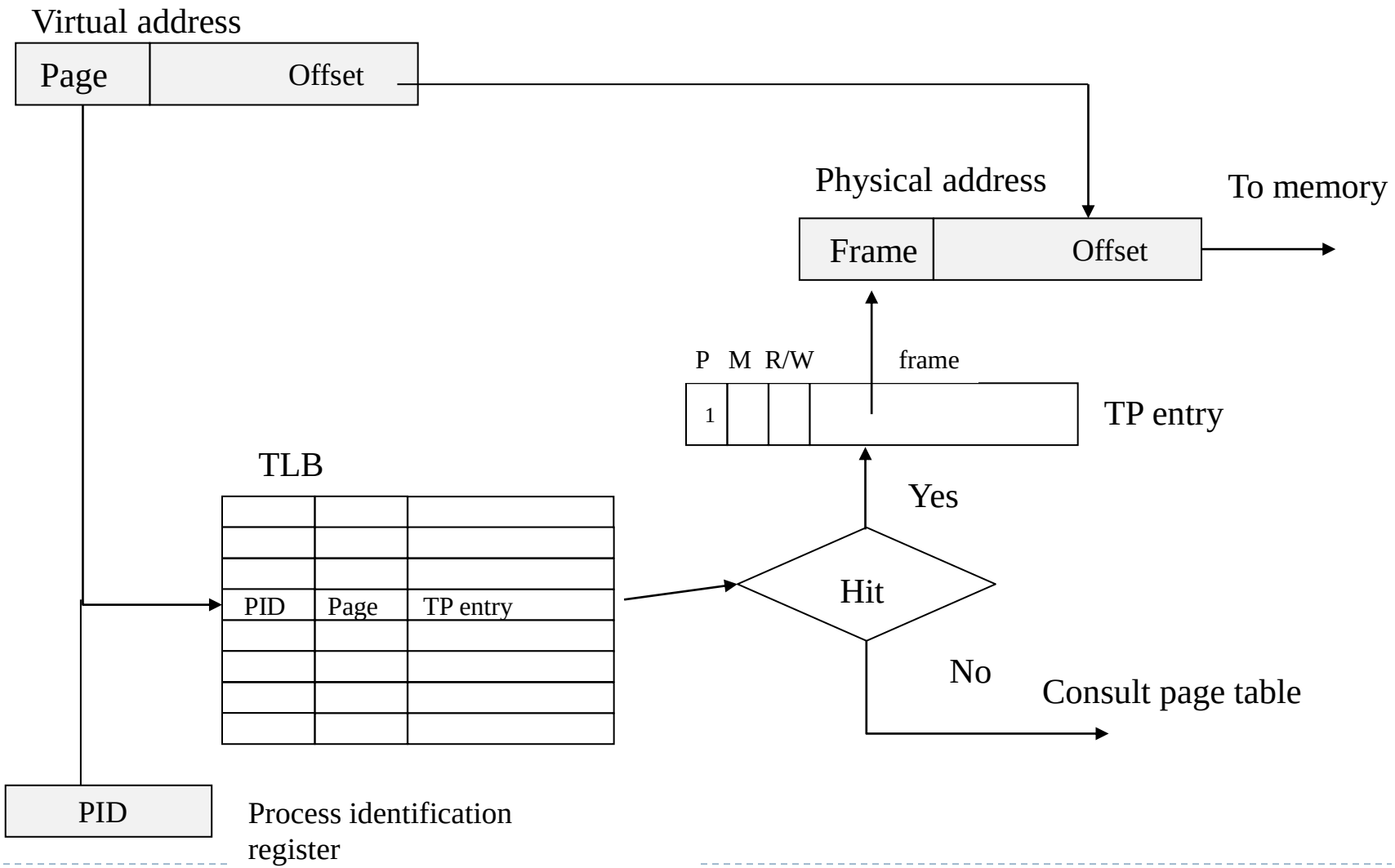
TLB (Translation Lookaside Buffer)

- 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



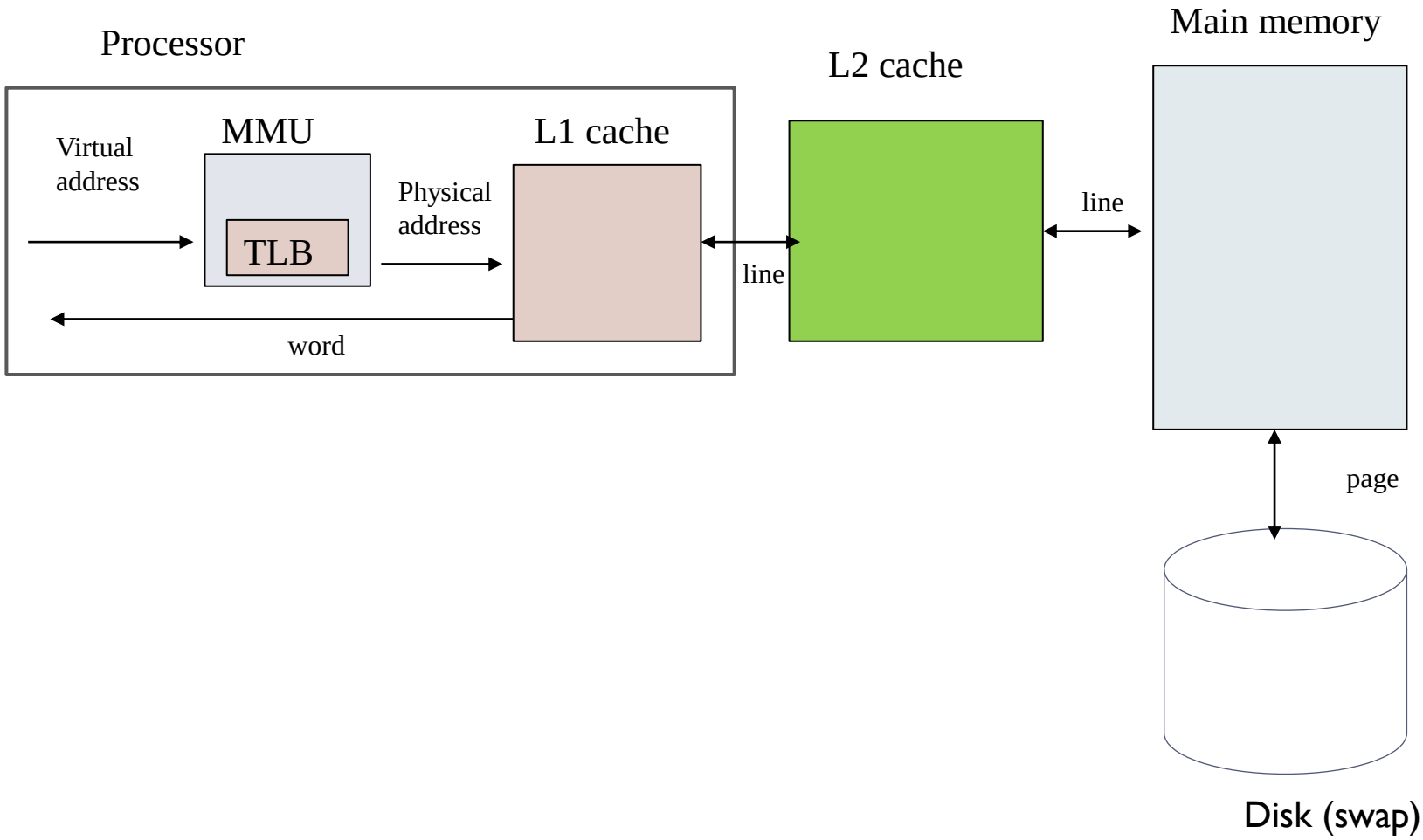
TLB witht process identification



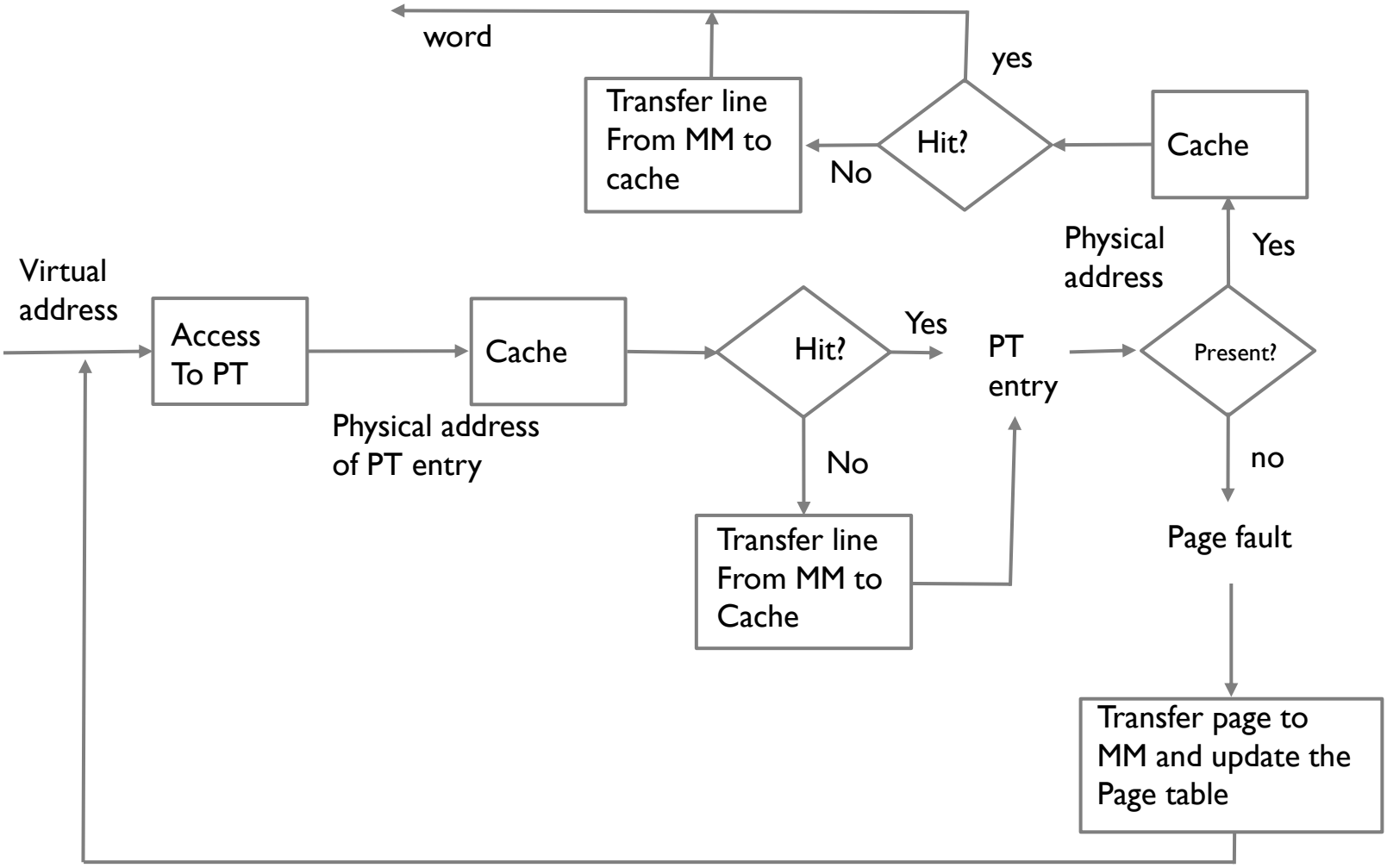
Cache and virtual memory

Cache	Virtual memory
Accelerate access	Increase addressable space
Transfer by blocks or lines	Transfer per page
Blocks: 32-64 bytes	Pages: 4-8 KiB
Translation: mapping algorithm	Translation: Fully associative
Immediate or deferred writing	Deferred writing

Virtual memory and cache memory



Read access with cache and virtual memory



ARCOS Group

uc3m | Universidad Carlos III de Madrid

L5: Memory hierarchy (3)

Computer Structure

Bachelor in Computer Science and Engineering
Bachelor in Applied Mathematics and Computing
Dual Bachelor in Computer Science and Engineering and Business Administration