University of Tlemcen

Computer Science Department

1st Year Engineer

"Introduction to the operating systems 2"

Sessions 8, 9 and 10: Memory management (summary)

Academic year: 2023-2024

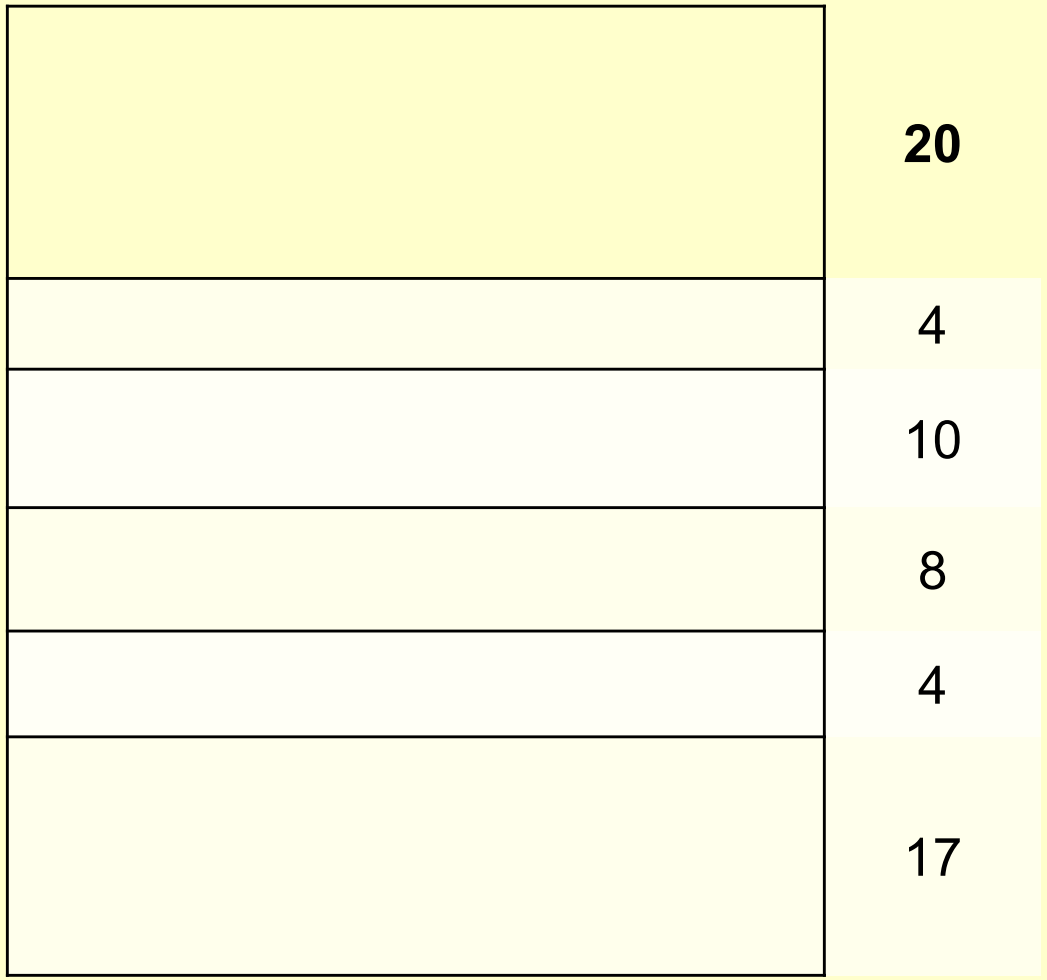
Memory management summary

Memory management system

- Memory is partitioned (divided into blocks called partitions)
  - *A partition contains one and only one process.*
- Memory management system
  - *Contiguous approach (the entire process is loaded)*
    - *Fixed partitioning of equal size*
      - *Choose a size for the partition that is larger than all the processes supported (impossible with current processes).*
    - *different size*
      - *Single queue*
      - *Multiple queues*
    - *Dynamic partitioning*
  - *Non-contiguous approach (Process is divided into several parts) the entire process is not necessarily loaded into memory*

- Contiguous approach
  - *Benefits*
    - *The process is loaded once and for all.*
    - *Addresses are easy to calculate because the process is loaded into a specific location in memory.*
  - *Disadvantages*
    - *The total free space in memory is greater than the size of the process and the process cannot be loaded into memory.*
    - *In the case of fixed partitioning of equal size*
      - *Problem of internal fragmentation when allocating a partition to a small process.*
    - *Planning space for increasing process size*
    - *It is virtually impossible to know the maximum size of a process in current systems.*

- Partitioning to different sizes (managed by two algorithms)
  - *single queue*
  - *multiple queues*



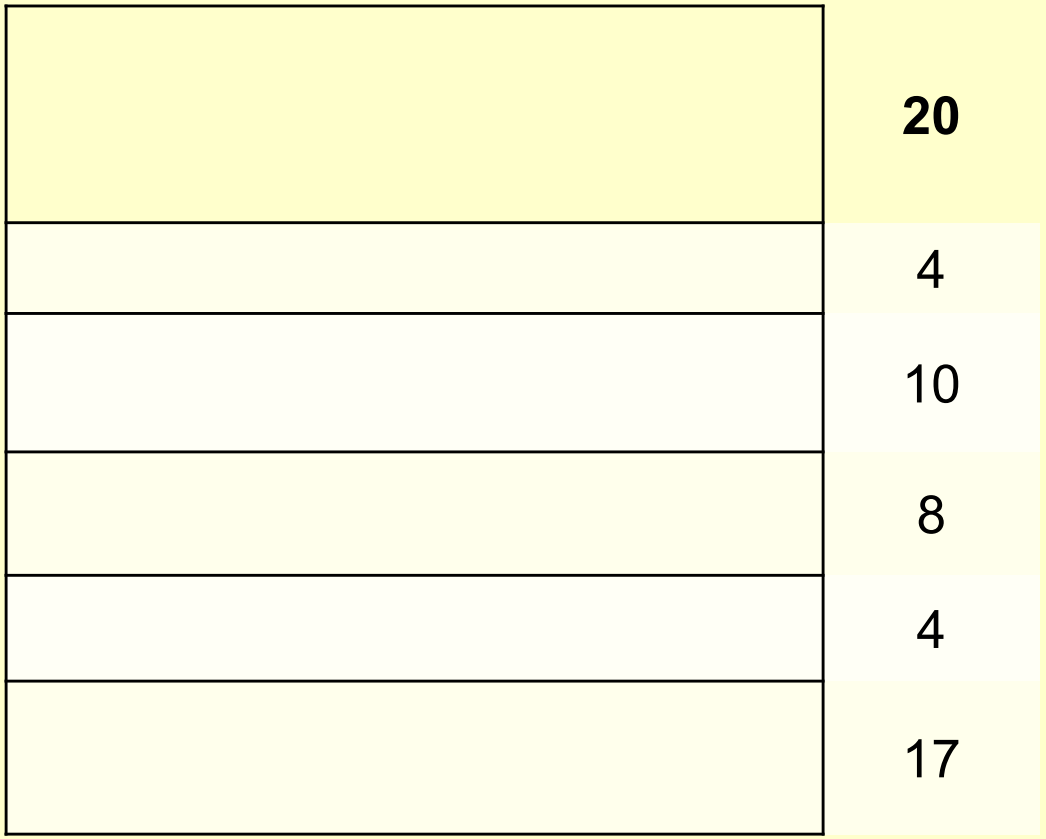
Waiting queue

$P_4(8)$	$P_3(4)$	$P_2(4)$	$P_1(4)$

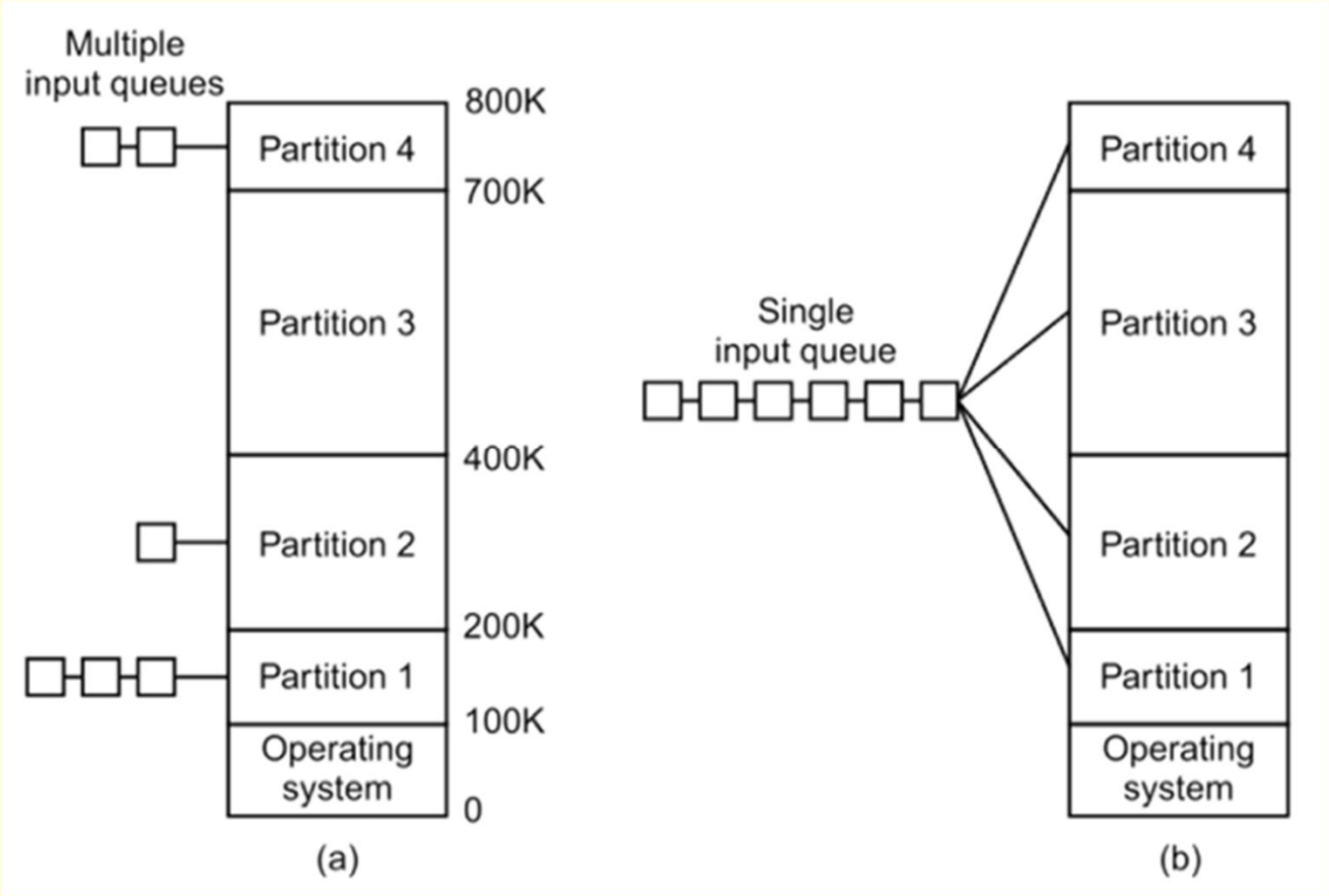
- with a single queue
  - *Partitions will be allocated to processes in order of arrival*

Queue

$P_4(8)$	$P_3(4)$	$P_2(4)$	$P_1(4)$

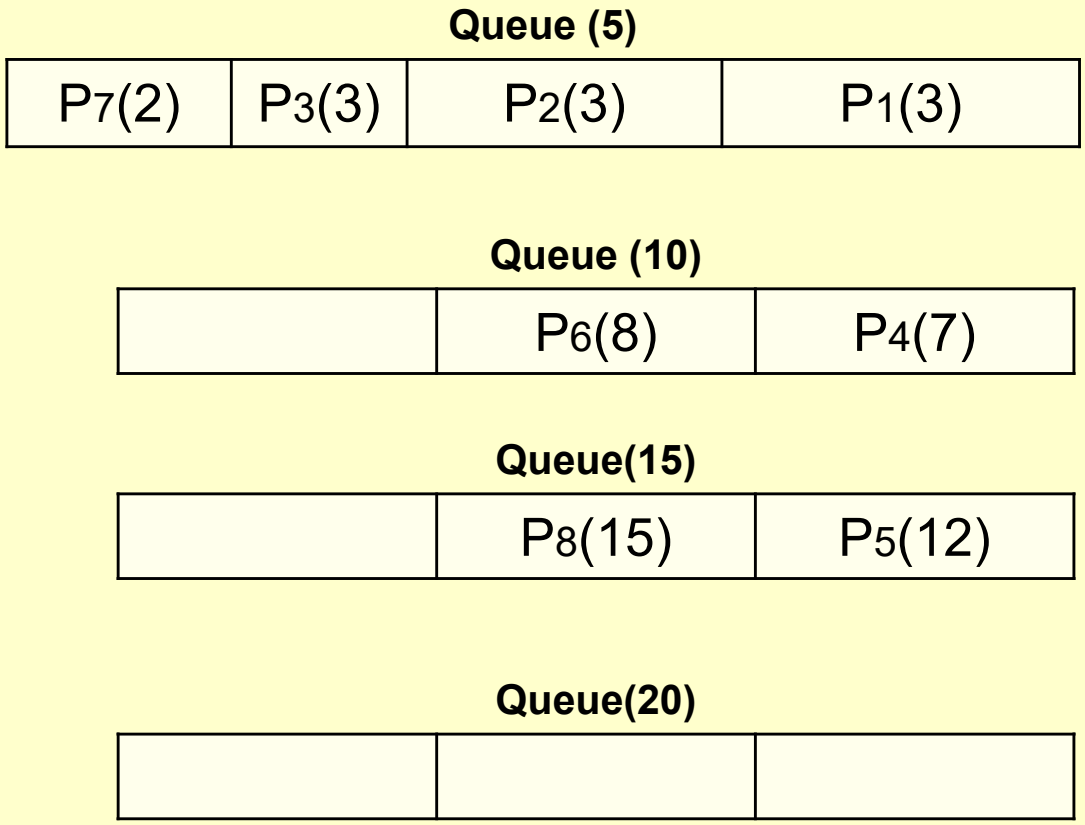


  - *The problem of internal fragmentation*
    - *Small processes can occupy large partitions*
    - *Running problems for large processes*


Multiple
input queues
Partition 4
Partition 3
Partition 2
Partition 1
Operating
system
800K
700K
400K
200K
100K
0
(a)
Single
input queue
Partition 4
Partition 3
Partition 2
Partition 1
Operating
system
(b)

- Suppose the long-term scheduler sorts processes by size (from largest to smallest)
  - *We could get into a blocking situation (blocking queue)*
    - *The process at the head of the queue is large (e.g. 15), followed by other small processes, and the maximum size of free partitions does not exceed 15.*
      - *Smaller processes in the queue will be penalized.*

- Several queues
  - *Each queue represents partitions of the same size*



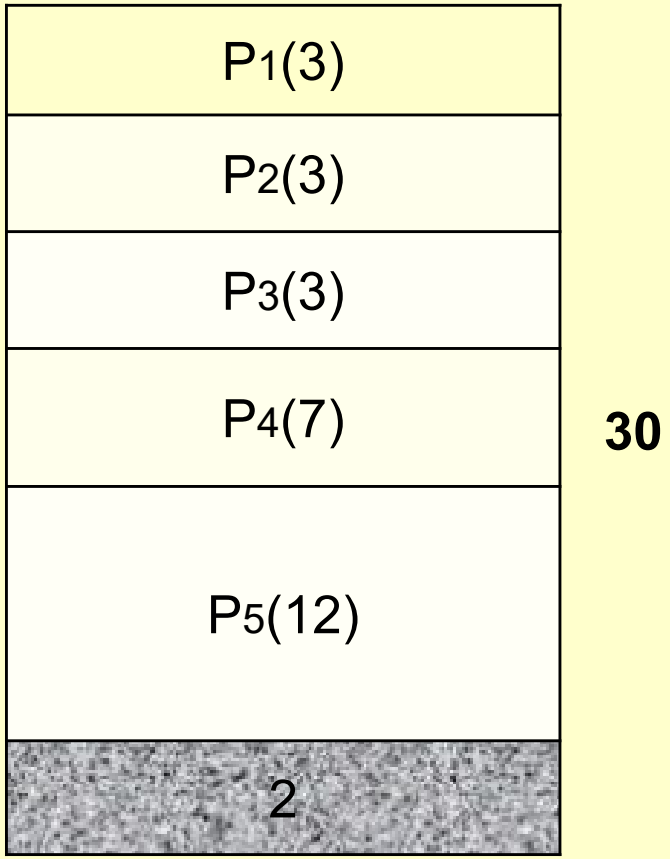
P1	**5**
P2	5
P4	10
P6	10
P5	15
	20
	20

- Several queues
  - *Reduced internal fragmentation*
  - *On the other hand, processes will be queued even if there are free partitions to contain them*
  - *Process dynamicity problem*
    - *If the process increases in size, there is a problem if its size is almost equal to that of the partition.*
      - *Changing partitions*
      - *Out of memory (not allowed to increase in size).*

- Dynamic partitioning
  - *Initially, the memory is not partitioned*

Queue

P8(15)	P7(2)	P6(8)	P5(12)	P4(7)	P3(3)	P2(3)	P1(3)



  - *Assuming that P_1 and P_3 have completed their execution we will not be able to load P_6*
  - *Suppose P_4 has finished executing*
    - *the free space becomes 10 we will be able to load P_6*

- Generation of external fragments (small partitions that cannot contain processes. A partition can contain several processes.
- Possibility of compacting (compressing) free partitions.
- Let's suppose we have a process whose size is 2 and we have two free partitions with the following sizes: 3 and 5. In which partition do we load the process in each case?
  - *Fixed partitioning*
  - *Dynamic partitioning*

Comparison of different memory allocation algorithms

- Worst fit
  - *Minimizes the number of external fragments.*
  - *But may prevent large processes from loading.*
- Best fit
  - *External fragmentation is large.*
  - *Large partitions can be exploited by large processes.*
- First fit
  - *The simplest algorithm to implement*
- The worst fit and best fit algo are not efficient.

Non-contiguous management

- Enables the process to be divided into a number of parts and placed in RAM as required.
- There are three types:
  - *Pagination*
    - *Divide the process into a set of elements of the same size (logical pages): page size*
    - *Divide RAM into frames (physical pages) that are the same size as the logical pages.*
    - *Load logical pages into RAM as they are used.*
    - *Manipulate page numbers instead of @.*
    - *Fixed equal size partitioning*
  - *Segmentation*
    - *Dynamic partitioning*
  - *Paged segmentation*

Pagination

- Correspondence between logical memory and physical memory
  - *In logical memory, the process is considered to be alone in the system.*
  - *Physical memory: @ allocated in RAM*

	Mémoire logique (Processus)
0	
N	

Mémoire physique (RAM)

- MMU transforms logical @ into physical @.

- Page table
  - *Maps logical pages to physical pages*

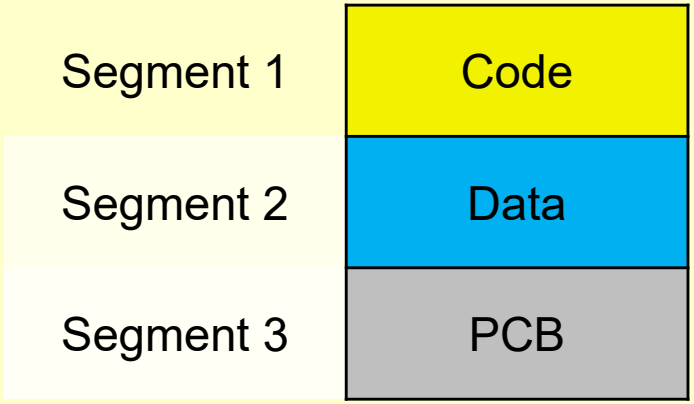
Logical page	Physical page	presence bit
0		
1		
2		
3		
4		

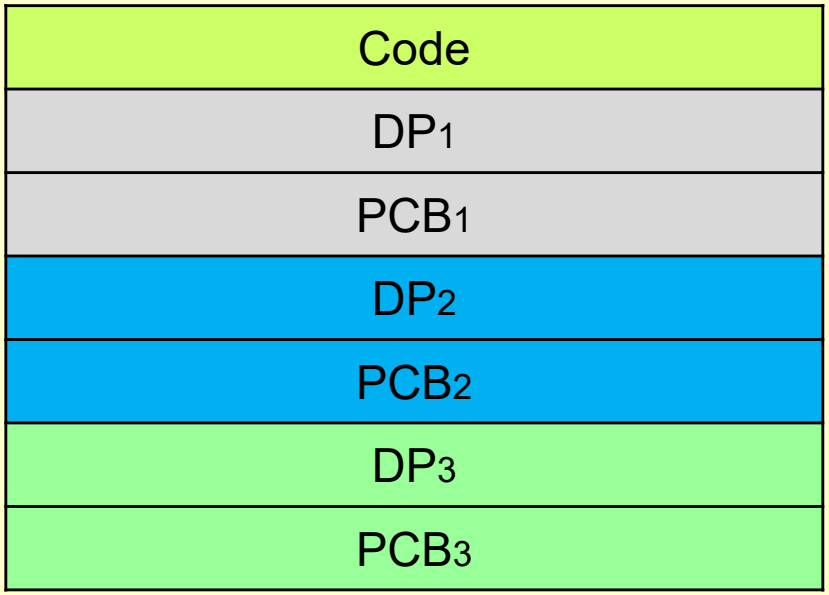
  - *Page fault problem (a page requested is not yet loaded into memory).*
    - *Run the placement algorithm and update the page table.*

- The content of the pages is not important, as you can have pages that contain both code and data.

Segmentation

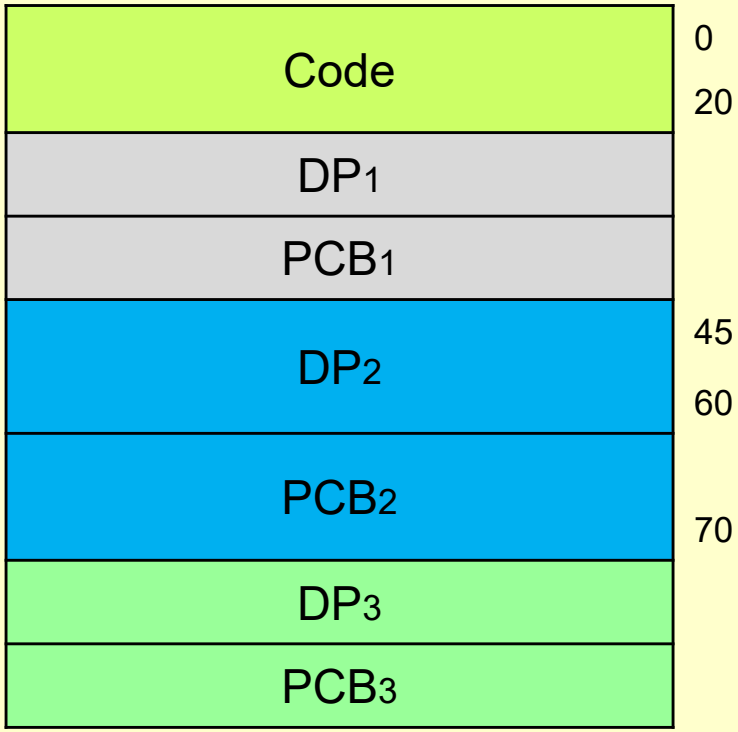
- The process is divided according to content, called segments (dynamic partitioning).
  - *For example, code, data and PCB (the segments)*
  - *Segments are loaded into RAM as required.*
  - *A segment is loaded in its entirety*
- Assume that a process is launched three times with different data:
  - *Three processes are created with the same code*
    - *There is therefore no need to load the code three times.*

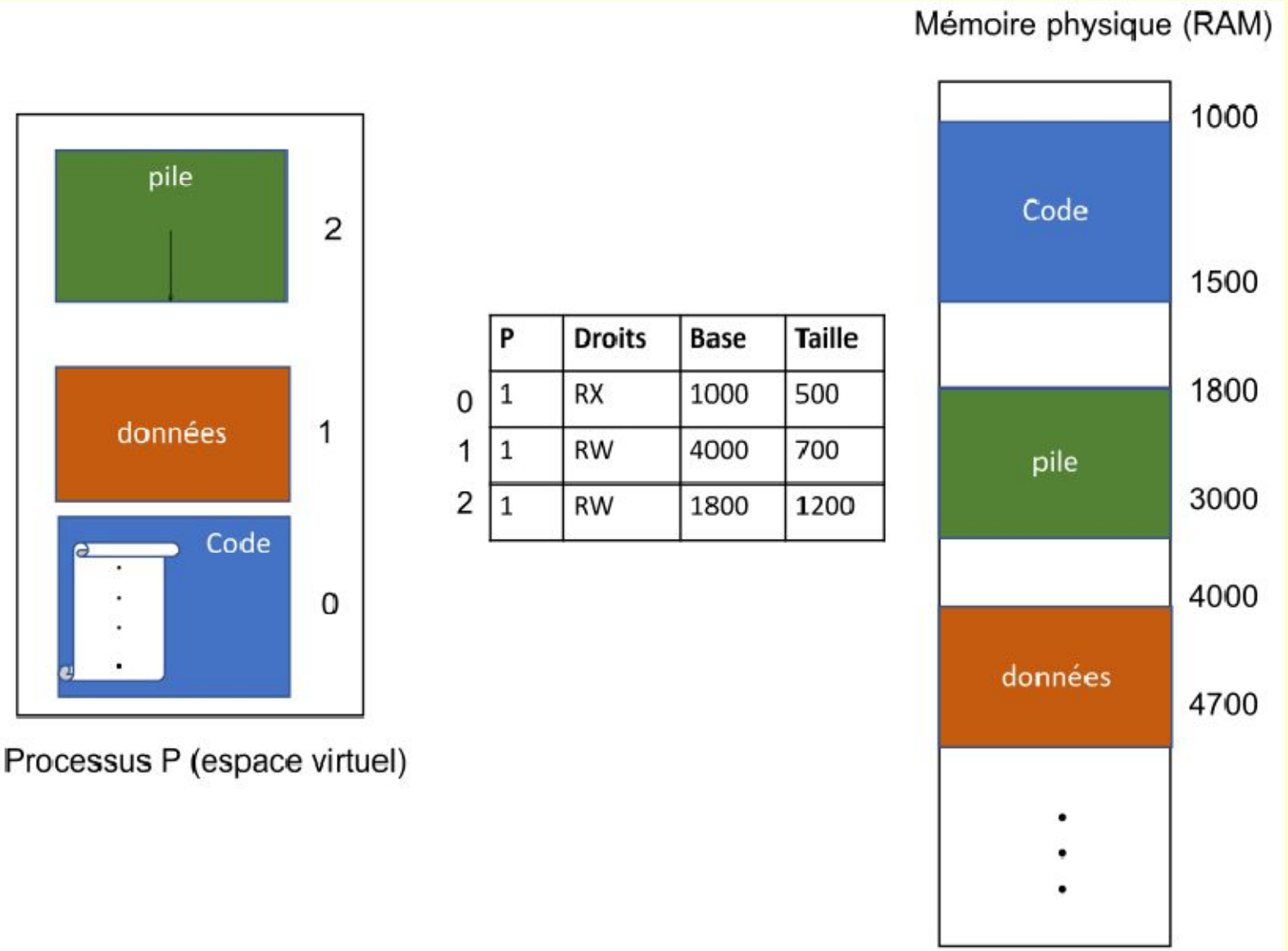




- Segment table
  - *Example: Process 2*

Segment	Limit (Size)	Base (@ start)	Presence bit
0	20	0	1
1	15	45	1
2	10	60	1



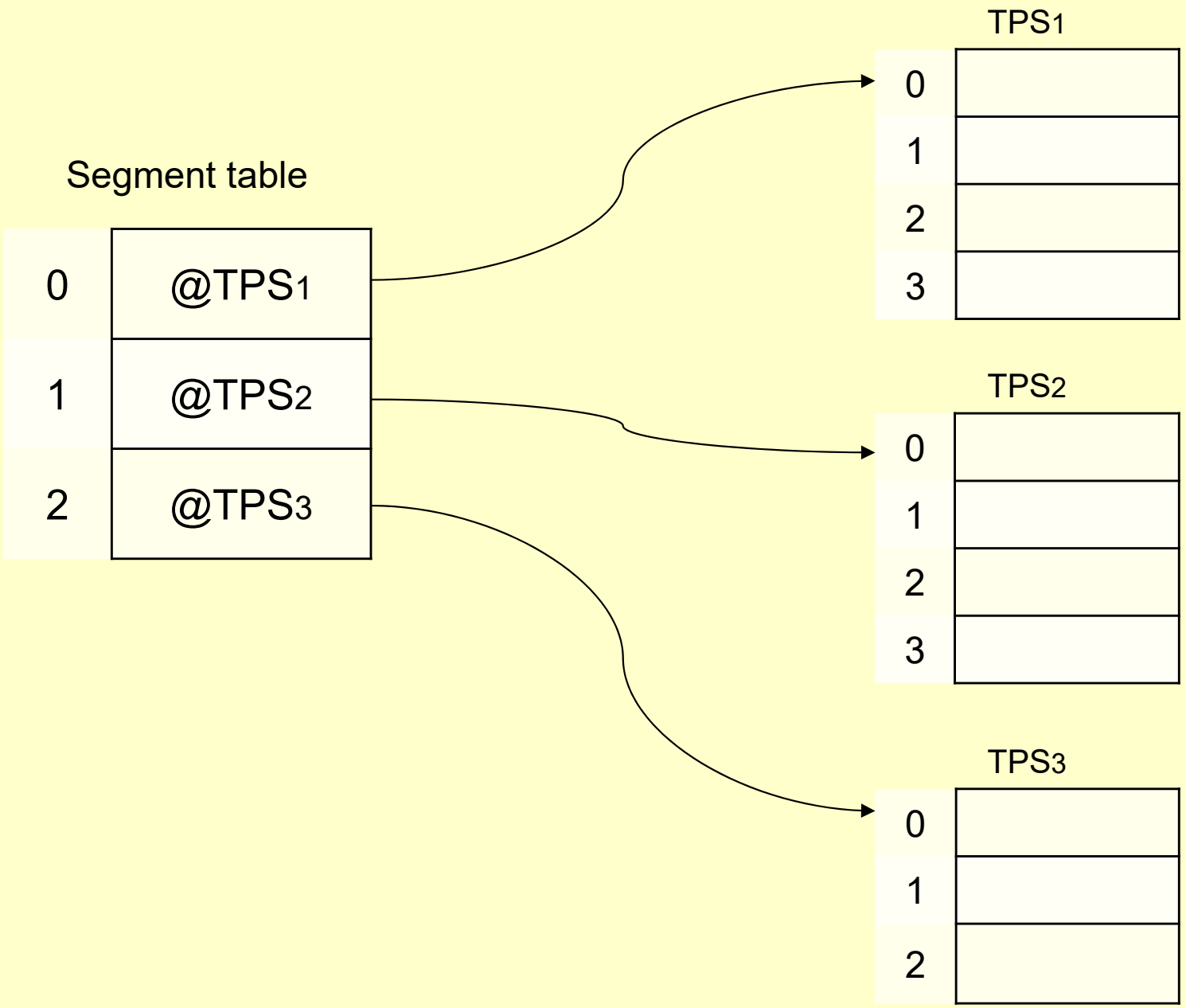


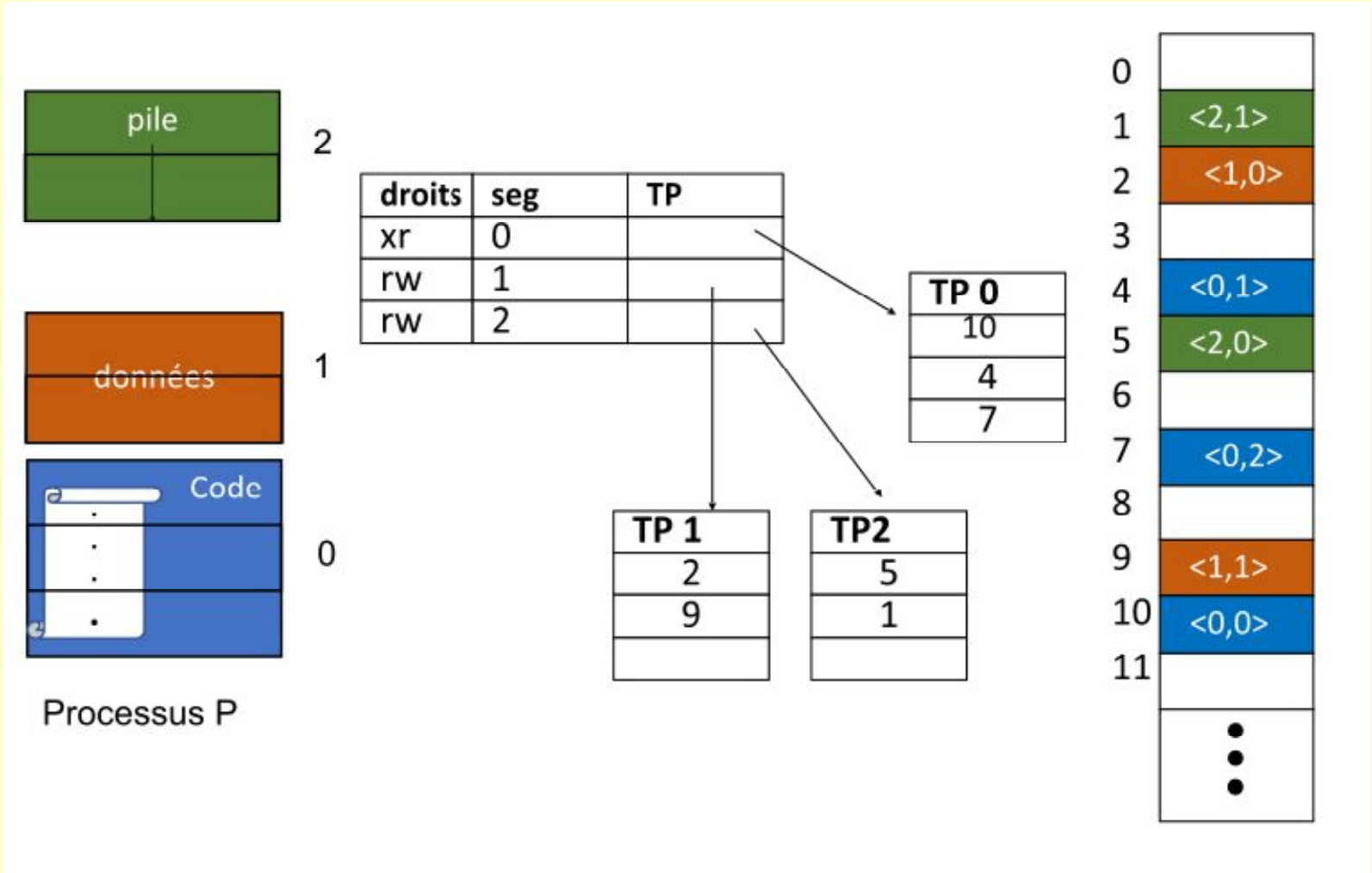
	P	Droits	Base	Taille
0	1	RX	1000	500
1	1	RW	4000	700
2	1	RW	1800	1200

- Segmentation limitation: contiguous allocation of segments in memory generates external fragmentation.
- This is in contrast to paging, which can even support processes larger than RAM.
- When the process increases in size, new pages simply need to be added.
- With pagination, internal fragmentation is only in the last page (less risk).
- Reuse of code in segmentation.

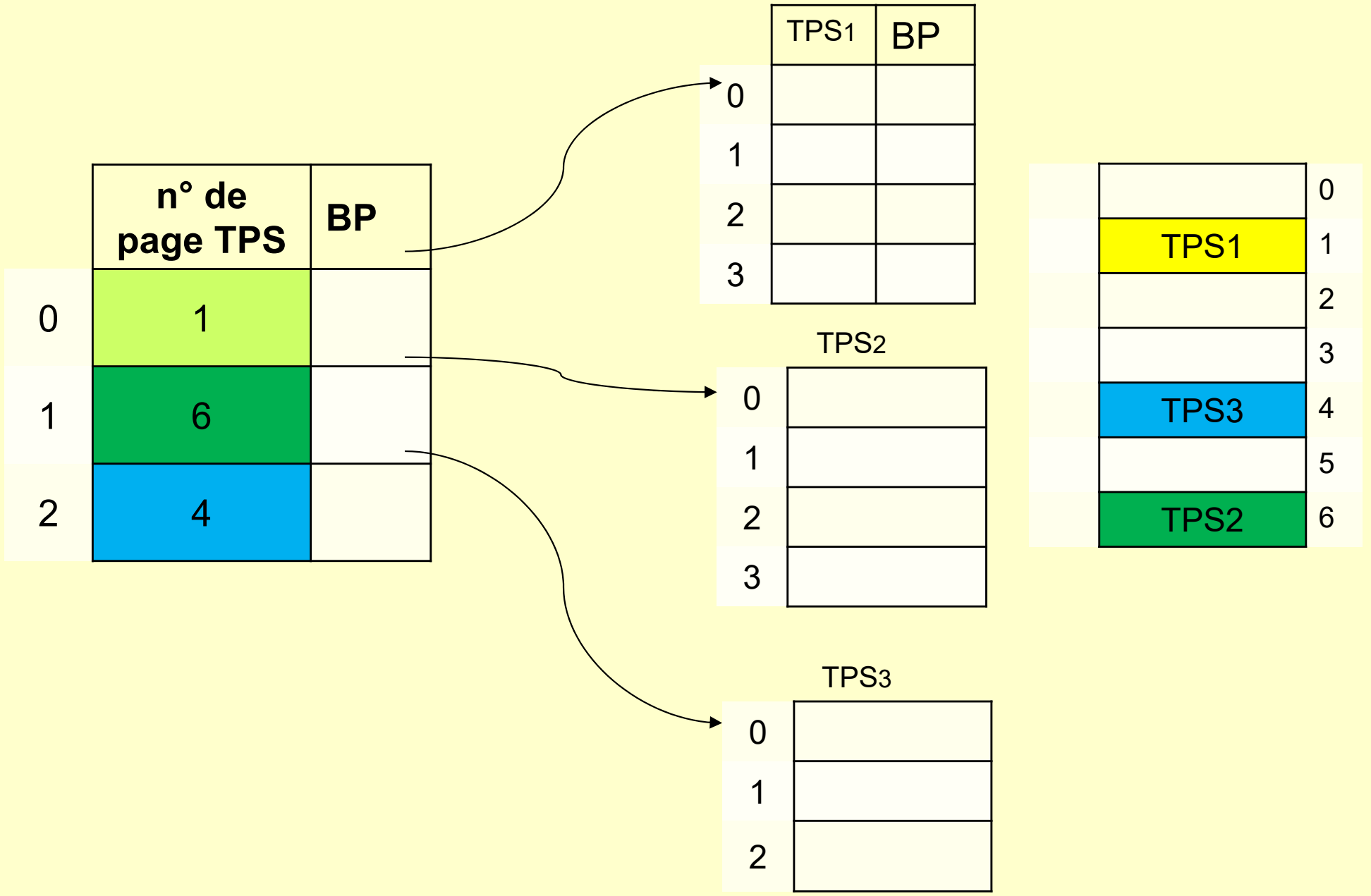
Paginated segmentation

- Pagination and segmentation : each have their drawbacks, and combining the advantages of both gives us a better solution:
  - *Divide the process into segments and then divide each segment into pages of fixed size.*
  - *Each page can then be loaded into memory.*
- Internal fragmentation averages ½ page per segment.
- One page table for each segment (three page tables for each process if the process uses three segments).

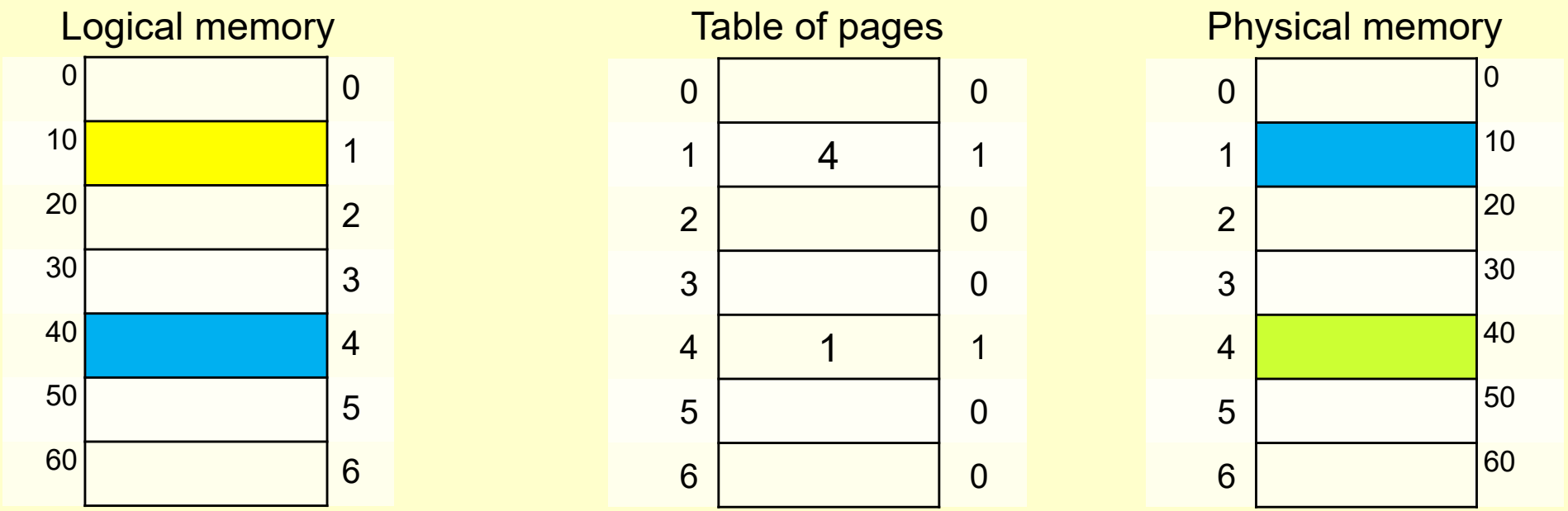

Segment table
0
@TPS1
1
@TPS2
2
@TPS3
TPS1
0
1
2
3
TPS2
0
1
2
3
TPS3
0
1
2


pile
données
Code
2
1
0
Processus P
droits
seg
TP
xr
0
rw
1
rw
2
TP 0
10
4
7
TP 1
2
9
TP2
5
1
0
1
2
3
4
5
6
7
8
9
10
11
<2,1>
<1,0>
<0,1>
<2,0>
<0,2>
<1,1>
<0,0>

Example


n° de page TPS
BP
0
1
2
1
6
4
TPS1
BP
0
1
2
3
TPS2
0
1
2
3
TPS3
0
1
2
0
TPS1
1
2
3
TPS3
4
5
TPS2
6

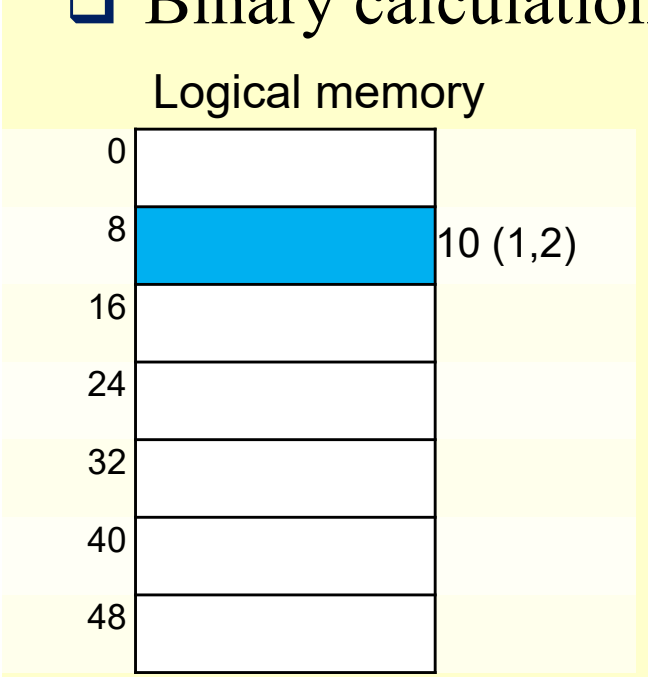
Pagination: Address calculation



- What is the physical @ associated with logical @ 17?
  - *We use the offset in the page.*
  - *logical @ (page, offset) → (1,7)*
  - *n° logical page (1) → n° physical page (f=4)*
  - *Physical @ =f∗size of frame+offset*

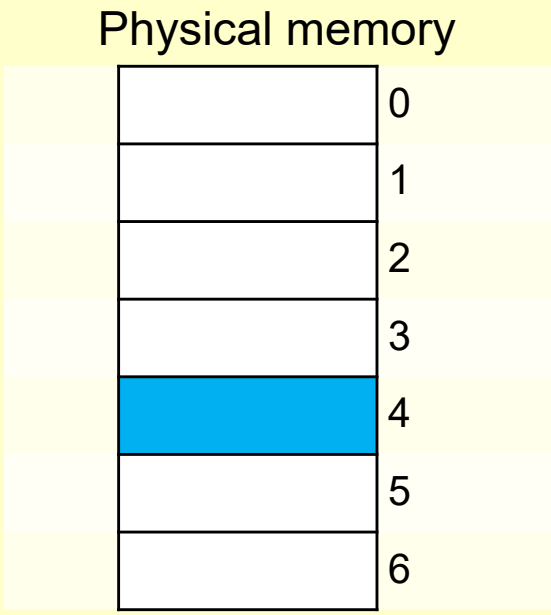
    =4∗10+7=47

❑ Binary calculation



$$Size\ of\ page = 8 = 2^3$$

$$10_{10} = 1010_2$$



❑ The size of @ depends on the architecture (example: 32 bits)

$$\underbrace{000000000000\ \ldots\ldots\ldots 1010}_{32\ bits}$$

❑ We need 3 bits to represent movement within a page.

$$\underbrace{000000000000\ \ldots\ldots\ldots 1}_{29\ bits\ for\ n°\ de\ page}\ \ \underbrace{010}_{3\ bits\ for\ offset}$$

- How to calculate physical @ ?

Page table		
0		
1	4	1
2		

$physical\ page\ (4) \rightarrow logical\ page\ (1)$

$physical\ @ :\ \underbrace{00000\ldots.100}_{29\ bits\ n°\ frame}\ \underbrace{010}_{offset}$

- **How do you make the transition from physical @physics to logical @ ?**

Exercise n° 1

- We assume that we have:
  - *16-bit address bus*
  - *Page size 4 kB = 4096 bytes (@ on 12 bits)*
  - *Page table in the following figure*
  - *Calculate the physical address corresponding to the logical address 8196 ?*
  - *16-bit address bus → physical @ on 16 bits*
  - *We work in decimal:*

Logical @=(n° of page, offset)

$n°\ of\ page = \text{div}(logical\ @, \text{page size}) = \text{div}(8196,4096) = 2$

$\text{offset} = \%(logical\ @, \text{page size}) = \%(8196,4096) = 4$

$2\ logical\ @ \rightarrow\ \ 6\ physical\ @$

$offset\ remains\ the\ same\ in\ both\ @$

$$Physical\ @\ = 6 * 4096 + 4 = 24\,580$$

15	**000**
14	000
13	000
12	000
11	111
10	000
9	101
8	000
7	000
6	000
5	011
4	100
3	000
2	110
1	001
0	010

Pagination: Exercise n° 2

- A computer with :
  - *A physical memory with a capacity of 128 MB,*
  - *32-bit architecture,*
  - *The size of a page is equal to 4 KB,*
  - *A virtual address indexes one byte.*

Give the maximum size of the page table where in this table each entry (frame reference + presence bit) ?

$$\text{Size of PT} = Number\ of\ pages * (Number\ of\ bits\ to\ encode\ n°\ of\ frame + 1\ bit)$$

Number of logical pages represents the number of pages that can contain a process

$$Number\ of\ logical\ pages = \frac{Max\ process\ size}{Page\ size} = \frac{2^{32}}{2^{12}} = 2^{20}$$

Number of frames= $\frac{Physical\ memory\ size}{Frame\ size} = \frac{2^{27}}{2^{12}} = 2^{15}$

Page table (PT)

	n° case	BP
0		1
1		

To represent these frames we need 15 $bits$

$$\text{Size of PT} = 2^{20} * (15\ bits + 1\ bit) = 2^{24}\ bits = 2\ Mo$$

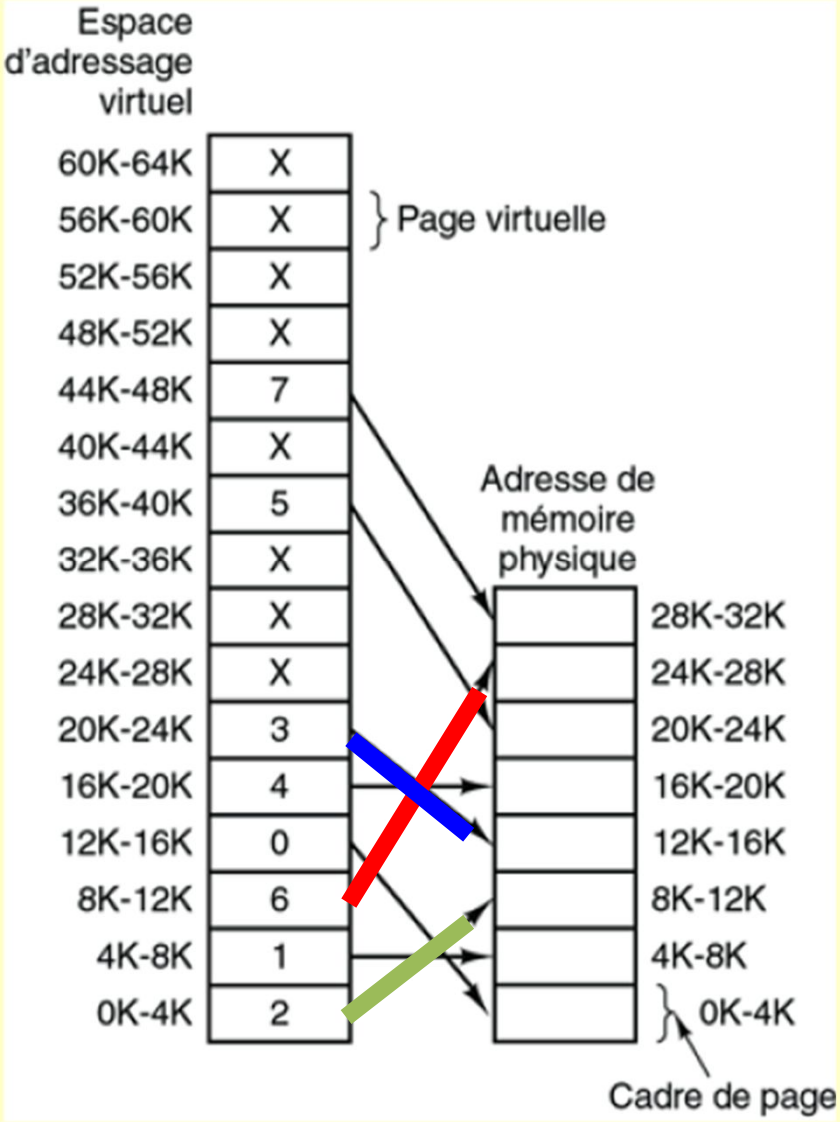
Page replacement algorithms

Page fault problem

- When the program tries to access address 0, for example, using the instruction:

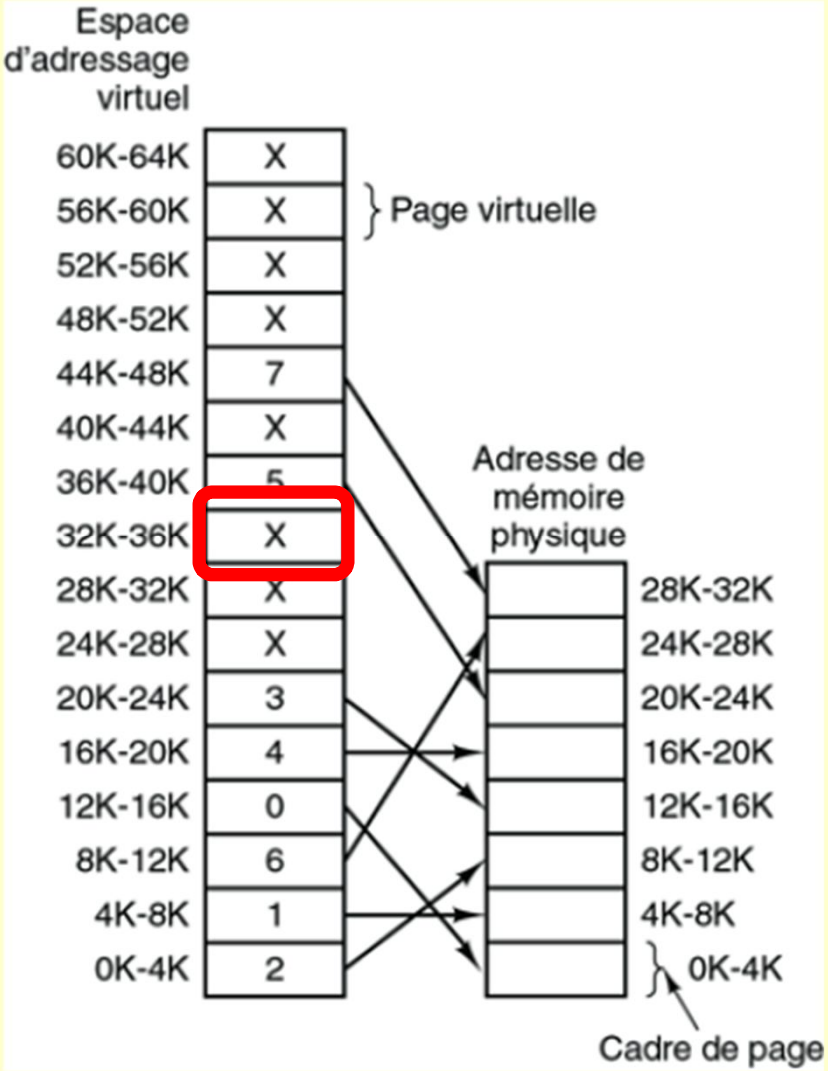
  MOV REG, [0]

- Virtual address 0 is sent to the MMU.
  - The MMU finds that this virtual address falls within page 0 (with values between 0 and 4095), which corresponds to the frame of page 2 (with values between 8192 and 12287).
  - It transforms the address into 8192 and presents it on the bus.

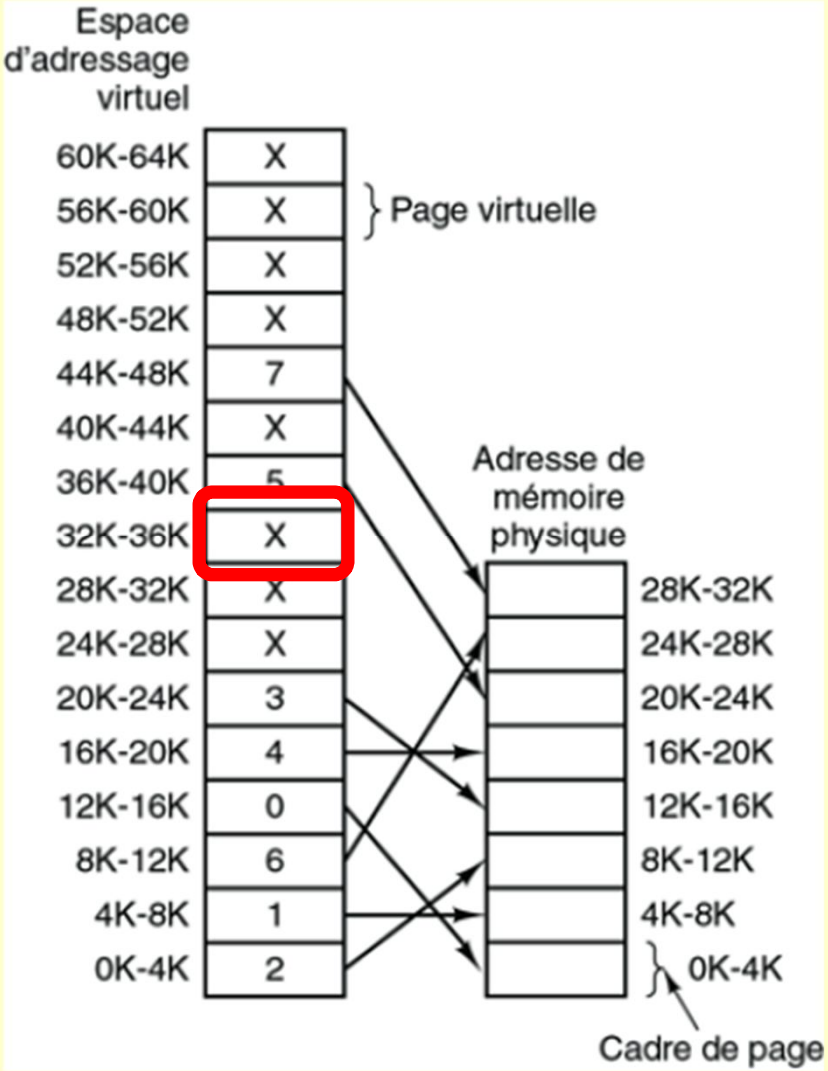


- This ability to map the 16 virtual pages onto the 8 page frames does not solve the problem posed by a virtual address space that is larger than the physical memory.
- Only 8 of the virtual pages are mapped to physical memory.
- A presence/absence bit keeps track of which pages are physically in memory.
- What happens if the program tries to call a page that is not present with the instruction

  MOV REG, [32780] ????



- *MOV REG,[32780] corresponds to byte 12 of virtual page 8 (which starts at 32768).*
- *The MMU notices that the page is missing and causes the CPU to perform a bypass, i.e. the processor is returned to the operating system.*
- *This rerouting, known as a page fault, is carried out in the following way:*
  - *The operating system selects a little-used page frame,*
  - *It writes its contents to disk;*
  - *It then transfers the page that has just been referenced to the freed page frame,*
  - *It modifies the correspondence and repeats the rerouted instruction.*



Page replacement algorithms

- When a page fault occurs, the S.E. must choose a page to remove from memory in order to make place for the page to be loaded.
- If the page to be removed was changed when it was in memory, it must be written back to disk to update the disk copy.
- If the page has not been changed, the disk copy is already up to date and does not need to be rewritten.
- The page to be read will simply overwrite the page to be removed.

- Although it is possible to randomly choose the page to be removed for each page fault, system performance will be better if a little-used page is chosen.
- If a frequently used page is removed, it will probably have to be reloaded quickly, resulting in additional overhead.
- The fundamental question: should a page belonging to the process requesting a new page be deleted from memory, or can a page belonging to another process be removed?

The optimal page replacement algorithm

- The best page replacement algorithm imaginable is easy to describe but almost impossible to implement.
- When a page fault is generated, several pages are in memory. One of these pages will be referenced by a very next instruction. Other pages may not be referenced until the tenth, hundredth or thousandth instruction that follows.
- Each page can be labelled with the number of instructions that will be executed before that page is referenced.

- The optimal page algorithm simply says that the page with the largest label will be removed.
  - *If a page is not used before 8 million instructions and another page before 6 million instructions,*
  - *Removing the first one will postpone the page fault until as late as possible.*
- The only problem with this algorithm is that it is infeasible.
  - *When the page fault occurs, the S.E. has no way of knowing when each of these pages will be referenced.*

The algorithm for replacing pages not recently used

- To enable the OS to collect useful statistics on the number of pages used and the number of unused pages, most computers with virtual memory have 2 status bits (R and M) associated with each page.
  - *The R bit is set every time the page is referenced (read or written),*
  - *The M bit is set to 1 when the page is rewritten (modified).*
  - *These bits must be updated each time the memory is referenced, so it is essential that this update is hardware.*

- When a process starts, the 2 page bits for all its pages are set to 0 by the S.E.
- Periodically (every clock interrupt), the R bit is cleared to distinguish pages that have not been recently referenced from those that have.
- When a page fault is generated, the S.E. examines all the pages and separates them into four categories based on the current values of R and M:
  - *Class 0: not referenced, not modified*
  - *Class 1: not referenced, modified*
  - *Class 2: referenced, unmodified*
  - *Class 3: referenced, modified*

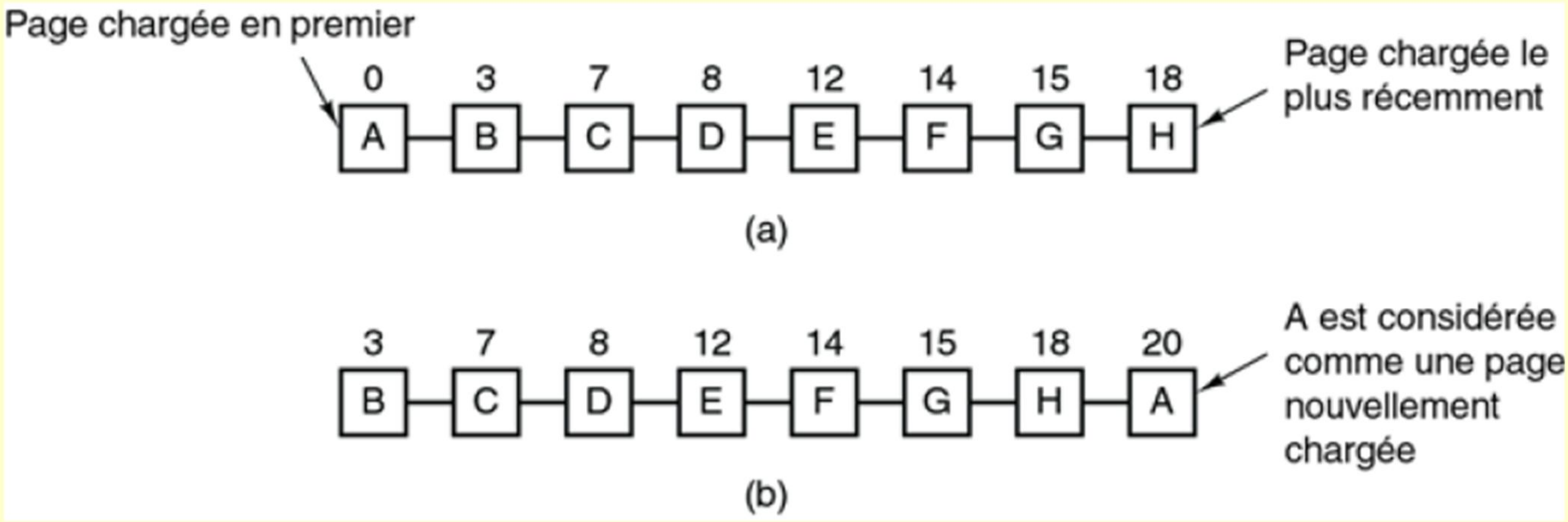
- Although the existence of class 1 pages seems impossible at first sight, they are generated when the R bit of a class 3 page is cleared by a clock interrupt.
  - *Clock interrupts do not clear the M bit because this information is needed to know whether or not the page has been written back to disk.*
  - *If the R bit is cleared but not the M bit, this results in class 1.*
- The Not Recently Used (NRU) algorithm removes a page at random from the lowest non-empty class.
  - *It is better to remove a modified page that has not been referenced in at least one clock top than a good page that is used a lot.*

The first-come, first-served page replacement algorithm (FCFS)

- The S.E maintains a FIFO list of all the pages currently in memory, with the oldest page at the top of the list and the most recent page at the bottom.
- In the event of a page fault, the page at the top of the list is removed and the new page added to the bottom of the list.
- Disadvantage
  - *Older pages may also be those that are most frequently used.*

The second-chance page replacement algorithm

- A simple modification can be made to the FIFO algorithm to avoid the common practice of deleting a page.
- This involves inspecting the R bit of the oldest page
  - *If it is 0, the page is both old and unused, so it is replaced immediately*
  - *If the R bit is set to 1, the bit is cleared, the page is placed at the end of the list of pages, and its load time is updated as if it had just arrived in memory.*
  - *The search then continues*



- Second chance operation:
  - *Pages sorted in order of arrival*
  - *If the R bit of the oldest page is 1, it is placed at the end of the list (as a new page) and its R bit is set to 0.*

The algorithm for replacing the least recently used page

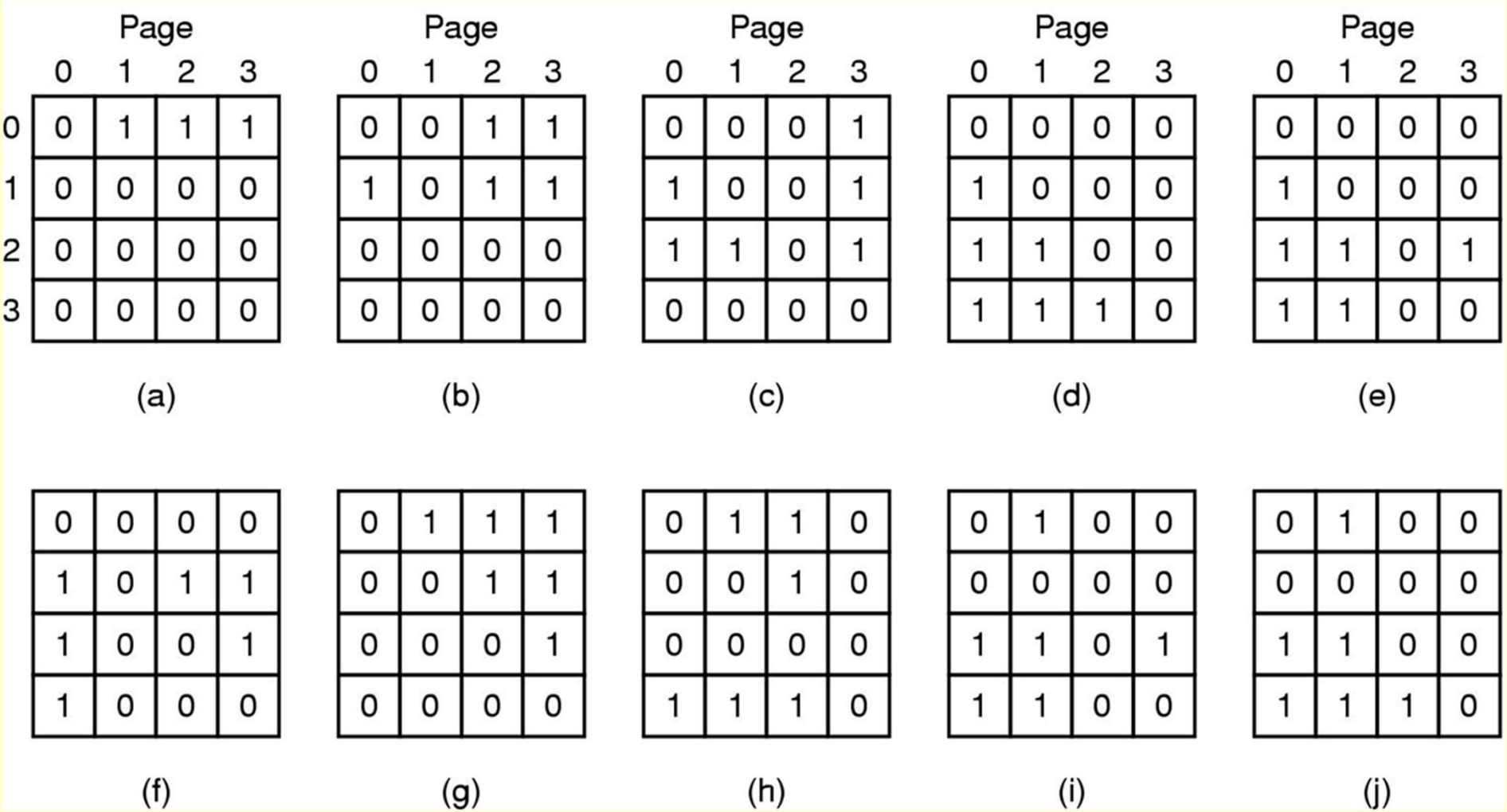
- To get as close as possible to the optimal algorithm, we need to make the following observation:
  - *the pages most referenced in the last instructions will probably be used a lot in the next instructions.*
  - *Conversely, pages that have not been used for a long time will not be requested for a long time.*
- The following algorithm can be deduced from this:
  - *when a page fault occurs, the page that has not been used for the longest time is removed.*

- This method is known as LRU (Least Recently Used) pagination.
- This LRU algorithm is feasible but expensive.
  - *Updating this list with each memory reference is a costly operation.*
  - *To implement it fully, it is necessary to manage a linked list of all the pages in memory, with the most recently used page at the top and the least recently used page at the bottom.*
  - *The difficulty is that this list has to be updated for each memory reference.*
  - *Searching for a page in the list, destroying it and moving it to the top of the list are time-consuming operations, even if they are carried out in hardware (assuming such hardware exists).*

- However, there are other ways of implementing this algorithm with particular hardware components.
- Let's look at the simplest of these. This algorithm requires the hardware to be equipped with a 64-bit counter, ‘c’, which increments automatically after each instruction.
  - *In addition, each entry in the page table must have a field large enough to contain this counter.*
- After each memory reference, the current value of ‘c’ is stored in the page table entry for the entry that has just been referenced.
- When a page fault occurs, the S.E. examines all the counters in the page table to find the smallest of them.
  - *This corresponds to the least recently used page.*

- Let's now look at a second hardware solution.
  - *For a machine with n page frames, the hardware must manage a matrix of n * n bits, initially all zero.*
- When a page k is referenced, the hardware first sets all the bits in row k to 1 and all the bits in column k to 0.
- At each point in time, the row with the smallest binary value indicates the least recently used page.

- Page reference : 0,1,2,3,2,1,0,3,2,3



Software simulation of the LRU algorithm

- Although the two previous LRU algorithms are feasible, few machines have the appropriate hardware.
- Instead, a software solution can be implemented:
  - *NFU: Not Frequently Used*
  - *The ageing algorithm*

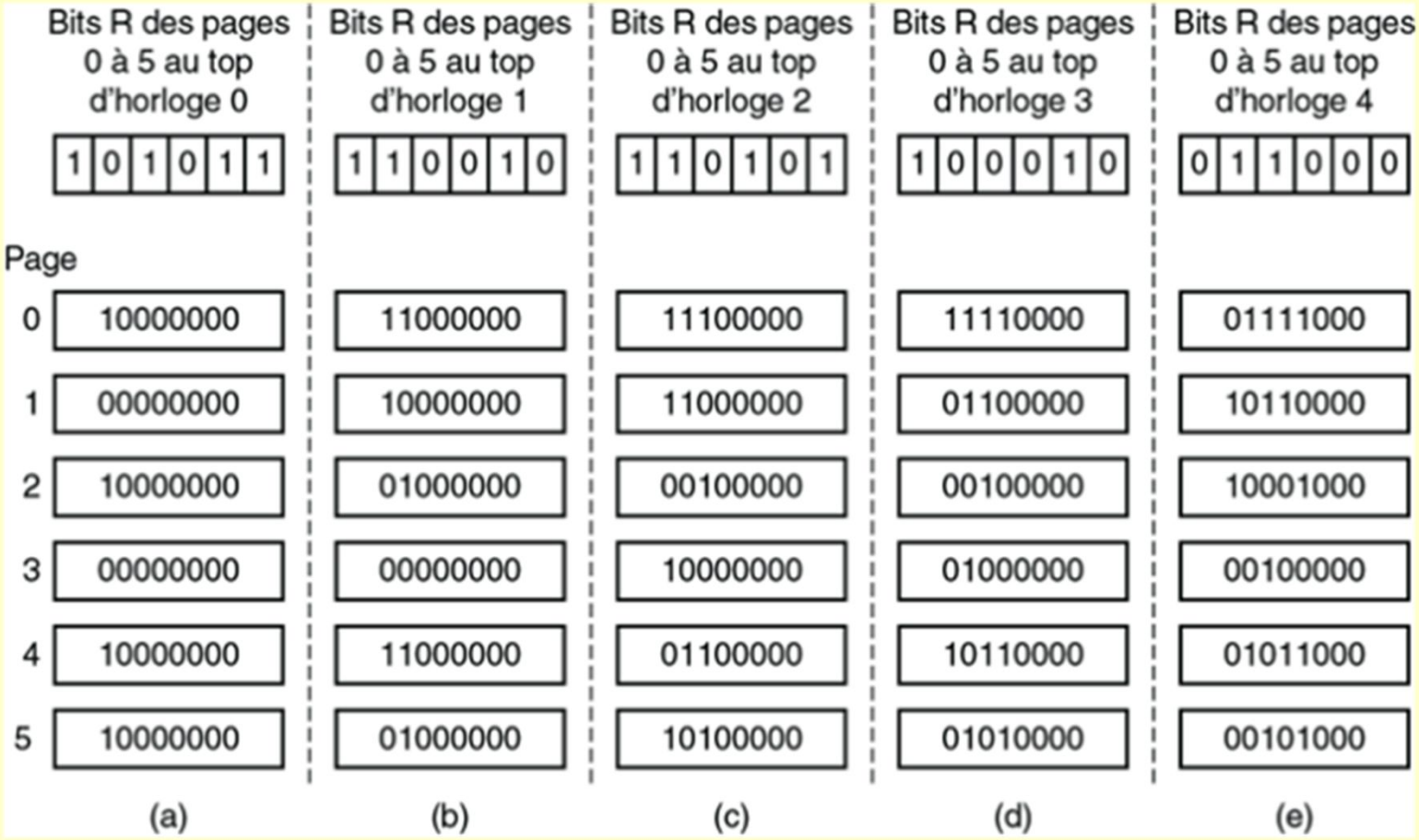
The algorithm "NFU"

- It requires a software counter per page, initially set to 0.
- Each time the clock is interrupted, the S.E. examines all the pages in memory:
  - *For each page, the R bit, value 0 or 1, is added to its counter.*
  - *The counter stores the different page references.*
  - *When a page fault occurs, the page with the lowest counter is replaced.*

- The main problem with NFU is that it doesn't forget anything.
  - *In a multi-pass compiler, the counters for pages that are heavily used in the first pass have high values in subsequent passes.*
    - *If the first pass has the longest execution time, the pages containing the code for subsequent passes always have a lower counter than the pages for the first pass.*
    - *The operating system will therefore tend to remove useful pages rather than unused pages.*

The ageing algorithm

- There is, however, a small modification that can be made to the NFU algorithm to enable it to behave like the LRU algorithm.
- This modification is carried out in two stages:
  - *The counters are shifted one bit to the right before being added to the R bit.*
  - *Then the R bit is added to the most significant bit (the left-hand bit) rather than the least significant bit.*
- Suppose that after the first clock signal, the R bits on pages 0 to 5 have the values 1,0,1,0,1,1
  - *Between clock times 0 and 1, pages 0,2,4,5 are referenced and their R bits are set to 1, while the others remain at 0.*



- *One software counter per page (set to 0),*
- *Each time the clock is interrupted, all the entries in the page table are examined.*
- *The counter is shifted to the right and the R bit is added to the left.*

- When a page fault occurs, the page with the smallest counter will be removed.
- Obviously, a page that has not been referenced for 4 clock beats has four 0s in its counter.
  - *It will therefore have a smaller value than the counter for a page that has not been referenced for 3 clock beats.*