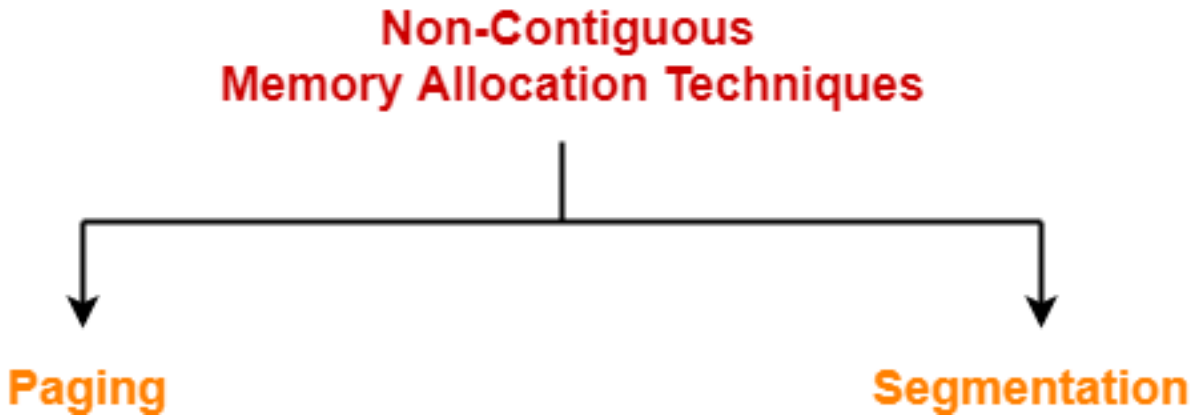
OPERATING SYSTEM: CSET209

CONTENT

- **Non-contiguous: Paging**
- **Translation Lookaside Buffer (TLB)**

NON-CONTIGUOUS MEMORY ALLOCATION-

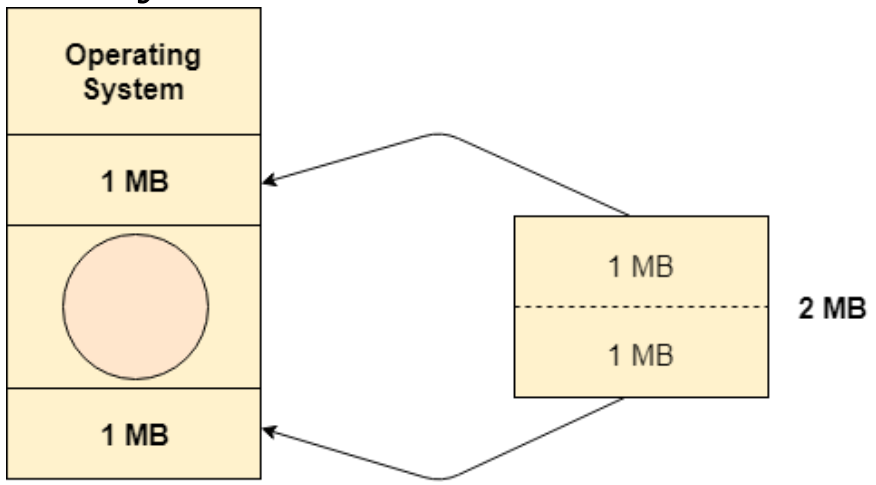
- Non-contiguous memory allocation is a memory allocation technique.
- It allows to store parts of a single process in a non-contiguous fashion.
- Thus, different parts of the same process can be stored at different places in the main memory..



NEED FOR PAGING

- The main disadvantage of Dynamic Partitioning is **External fragmentation**. Although, this can be removed by Compaction but as we have discussed earlier, the compaction makes the system inefficient.

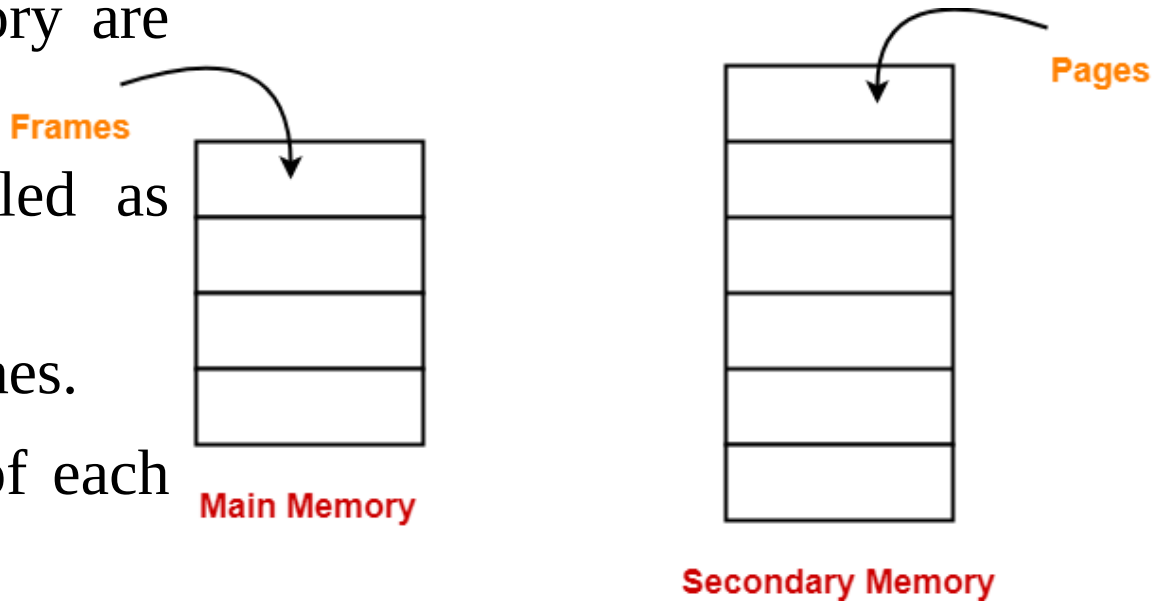
- We need to find out a mechanism which can load the processes in the partitions in a more optimal way. Let us discuss a dynamic and flexible mechanism called paging.



The process needs to be divided into two parts to get stored at two different places.

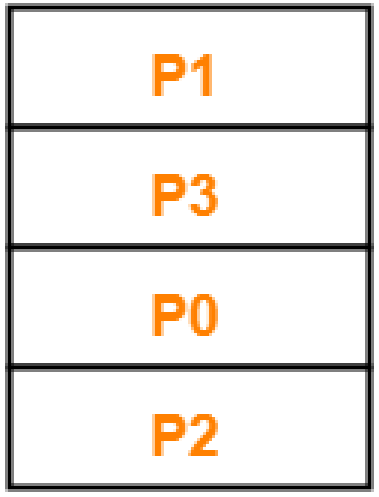
PAGING

- Paging is a fixed size partitioning scheme.
- In paging, secondary memory and main memory are divided into equal fixed size partitions.
- The partitions of secondary memory are called as pages.
- The partitions of main memory are called as frames.
- Each process is divided into parts where size of each part is same as page size.
- The size of the last part may be less than the page size.
- The pages of process are stored in the frames of main memory depending upon their availability.



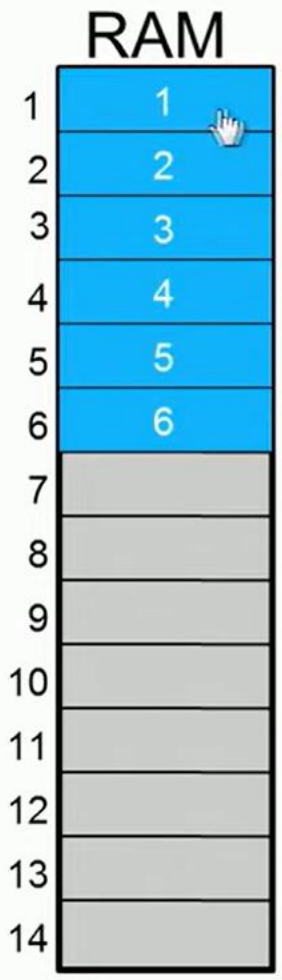
EXAMPLE

- Consider a process is divided into 4 pages P0, P1, P2 and P3.
- Depending upon the availability, these pages may be stored in the main memory frames in a non-contiguous fashion as shown-



Main Memory

Paging
(Page Table)

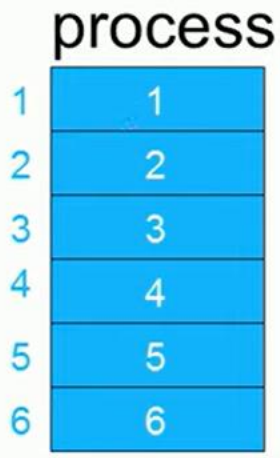


RAM is split into fixed size partitions called page frames.

Page frames typically 4KBs

Process split into blocks of equal size.

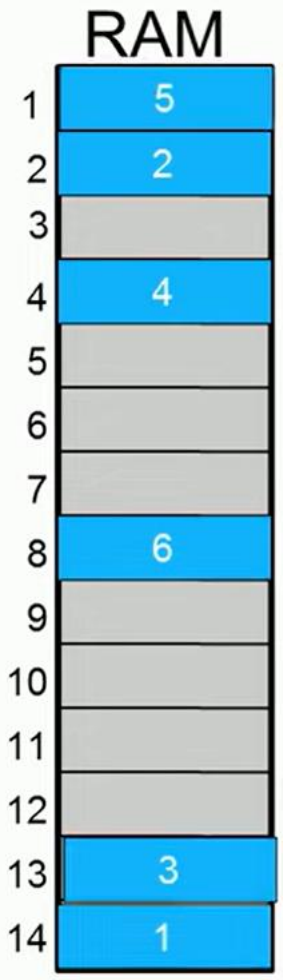
block size = page frame size

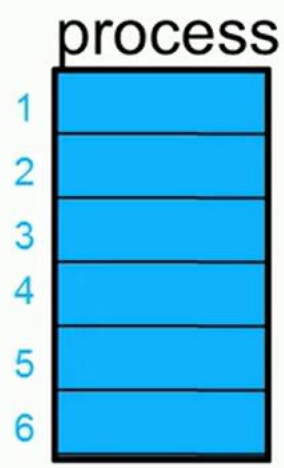


Per process page table (stored in RAM)

block	page frame
1	1
2	2
3	3
4	4
5	5
6	6

Paging





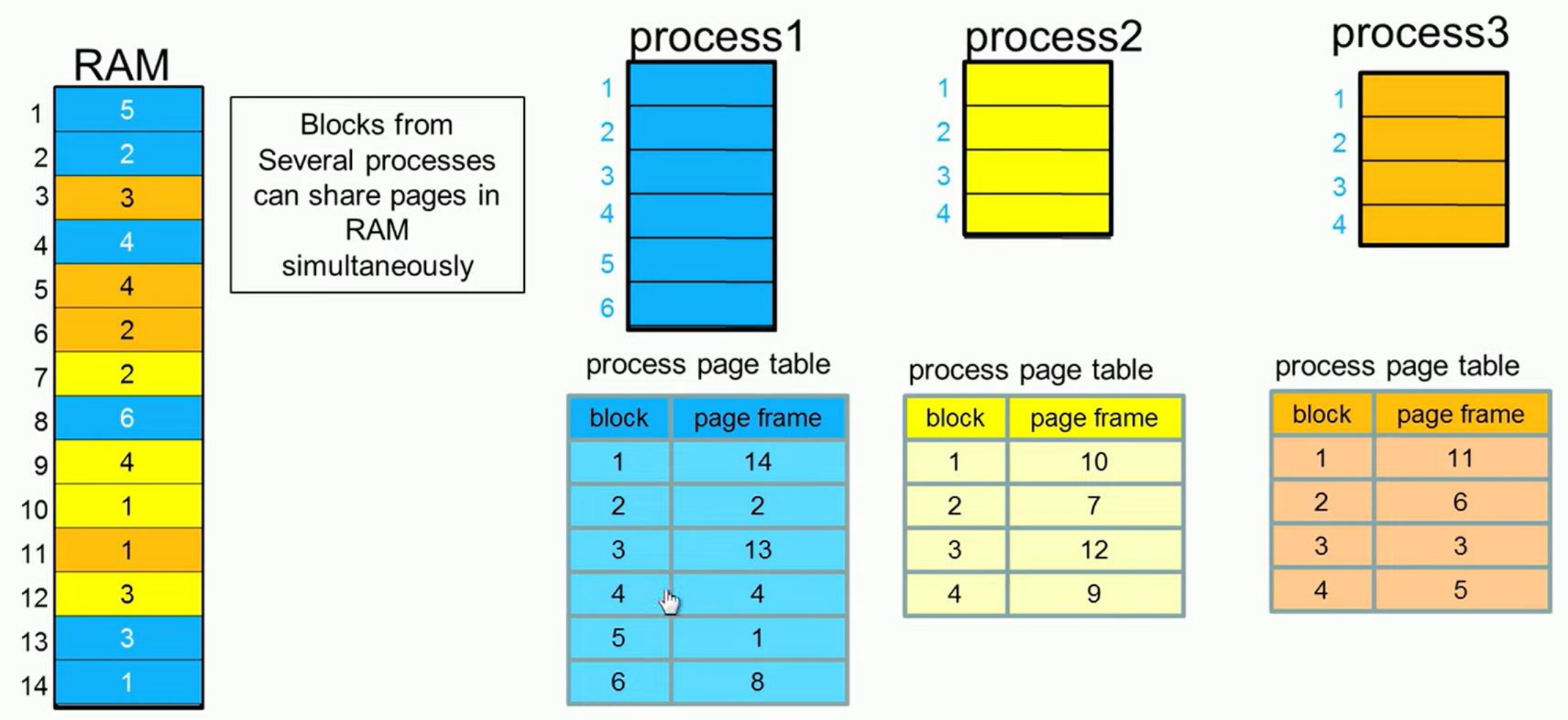
process page table

block	page frame
1	14
2	2
3	13
4	4
5	1
6	8

Because of the page table, blocks need not be in contiguous page frames

Every time a memory location is accessed, the processor looks into the page table to identify the corresponding page frame number.

Paging



process page table

block	page frame
1	14
2	2
3	13
4	4
5	1
6	8

process page table

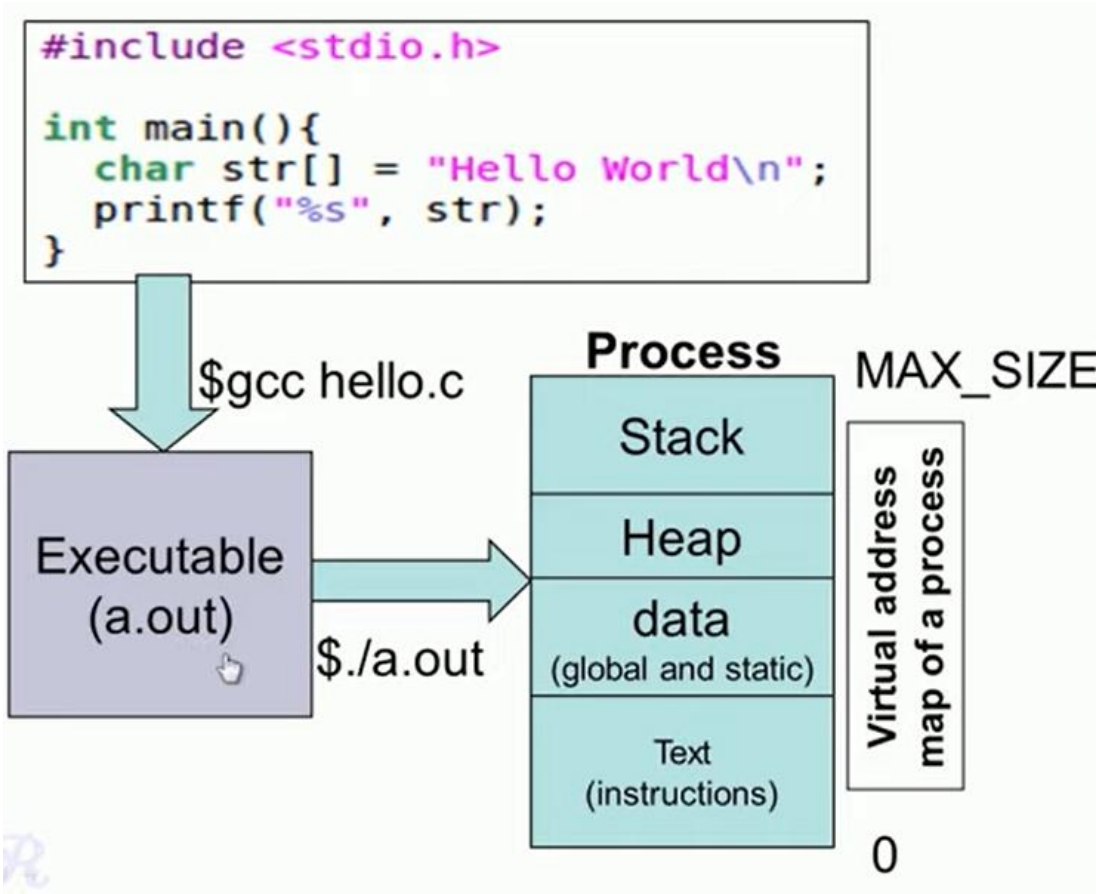
block	page frame
1	10
2	7
3	12
4	9

process page table

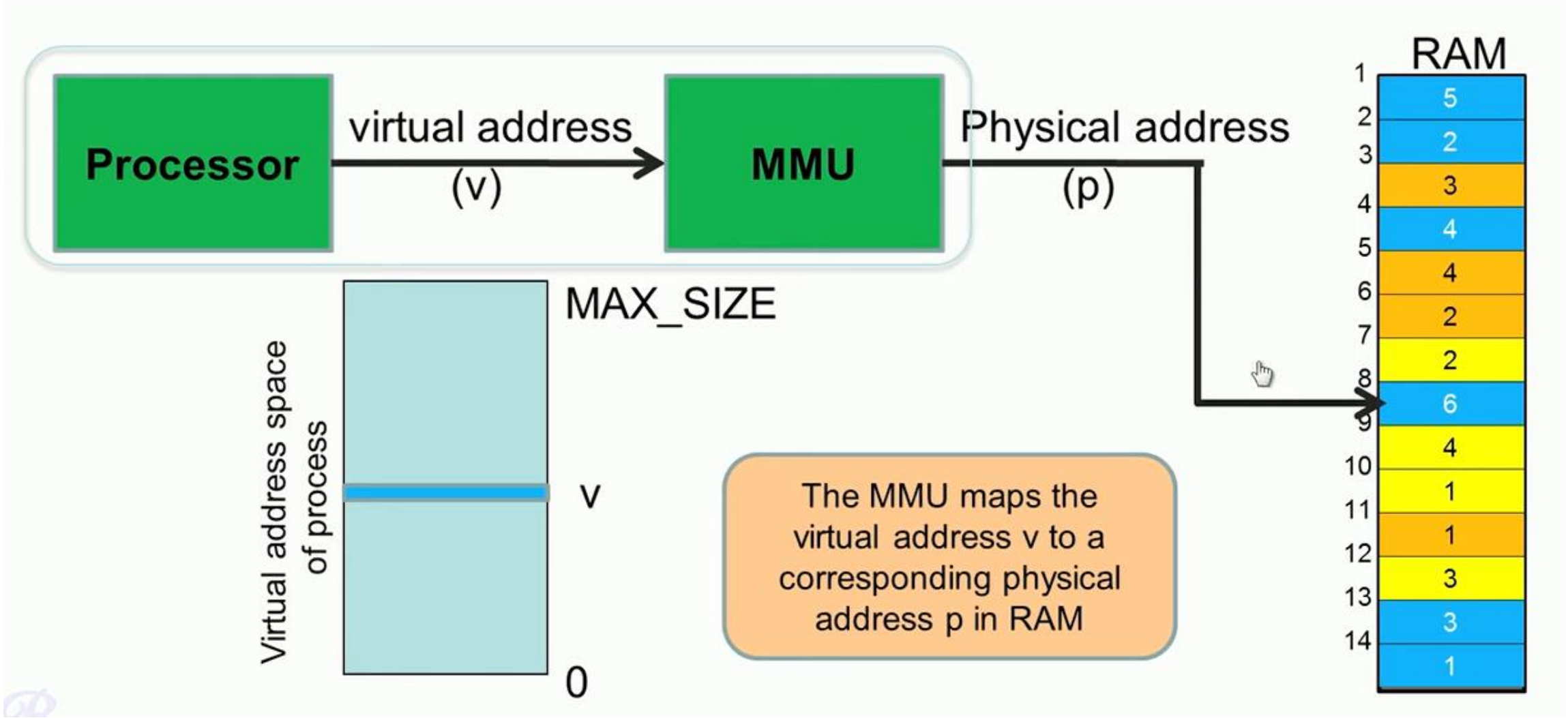
block	page frame
1	11
2	6
3	3
4	5

Virtual Address space of a process

- Contiguous memory addresses used to represent a process. A virtual address does not represent the actual physical location of an object in memory; instead, the *page table* for each process, is used to translate virtual addresses into their corresponding physical addresses.
- Each time a process references an address, the system (MMU) translates the virtual address to a physical address.
- If the process tries to print the address of str then the virtual address gets printed.

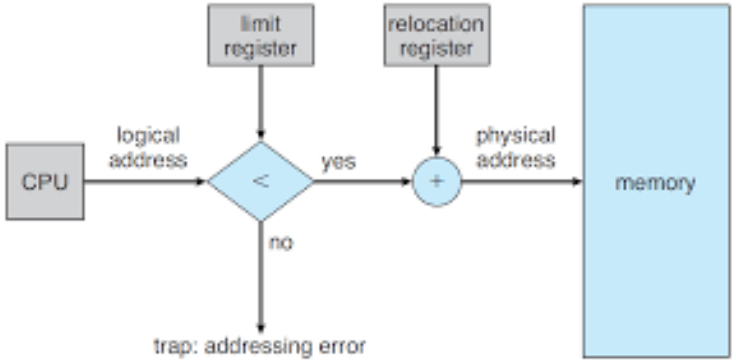


Translating Virtual (logical) address to Physical address (MMU)



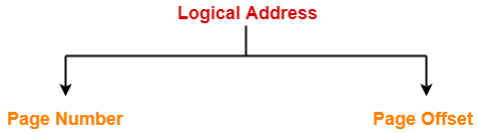
TRANSLATING LOGICAL ADDRESS INTO PHYSICAL ADDRESS

- CPU always generates a logical address.
- A physical address is needed to access the main memory



Following steps are followed to translate logical address into physical address-

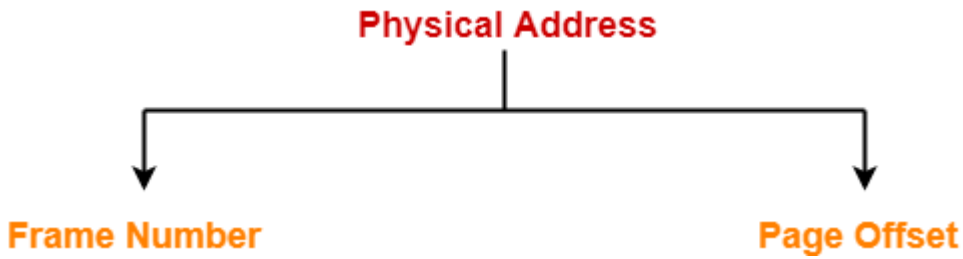
- Step-01:
  - CPU generates a logical address consisting of two parts-
    - Page Number
    - Page Offset



- Page Number specifies the specific page of the process from which CPU wants to read the data.
- Page Offset specifies the specific word on the page that CPU wants to read.

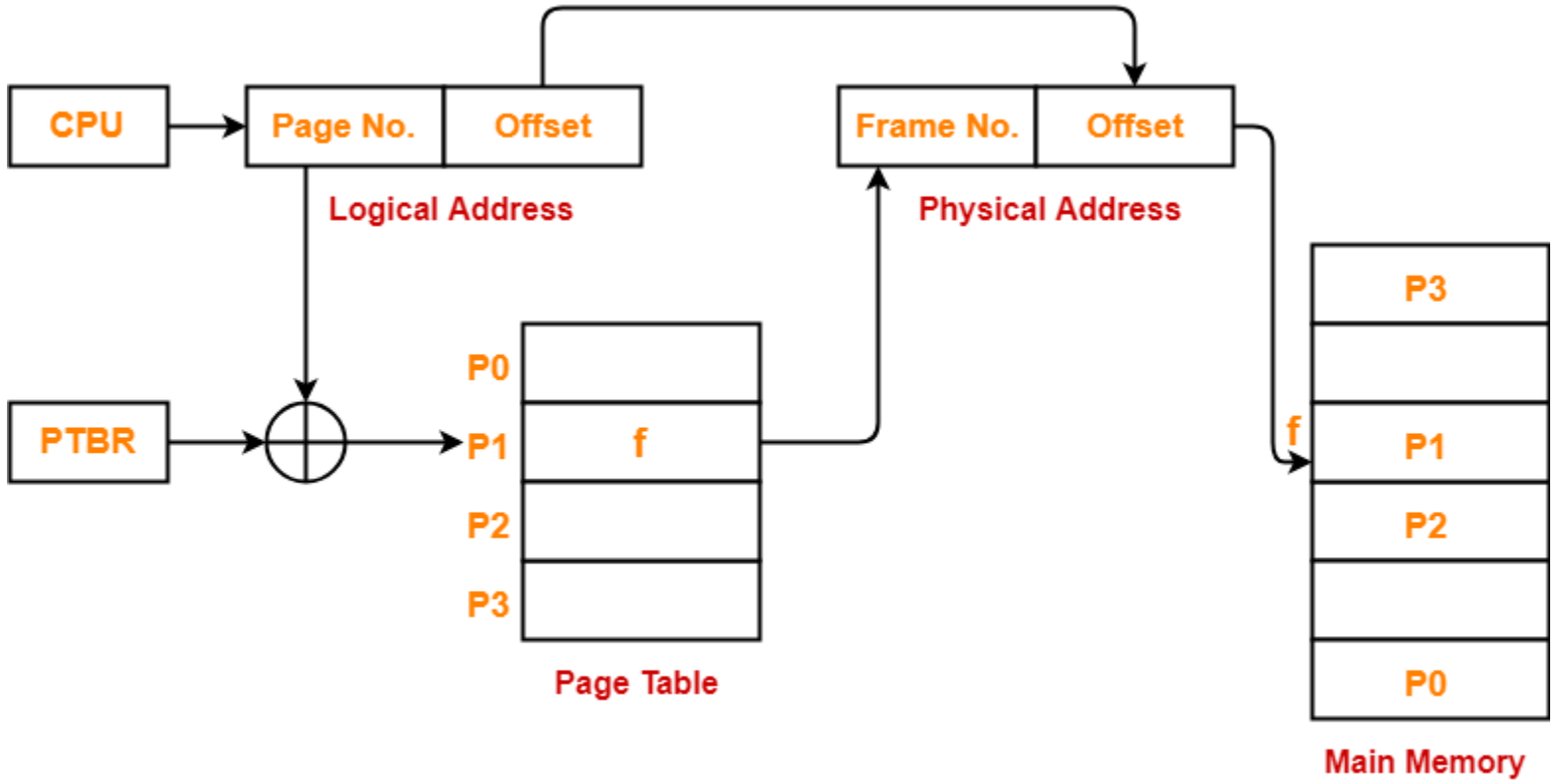
TRANSLATING LOGICAL ADDRESS INTO PHYSICAL ADDRESS

- **Step 02:**
  - For the page number generated by the CPU
  - Page Table provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.
- **Step 03:**
  - The frame number combined with the page offset forms the required physical address
  - Frame number specifies the specific frame where the required page is stored.
  - Page Offset specifies the specific word that has to be read from that page.



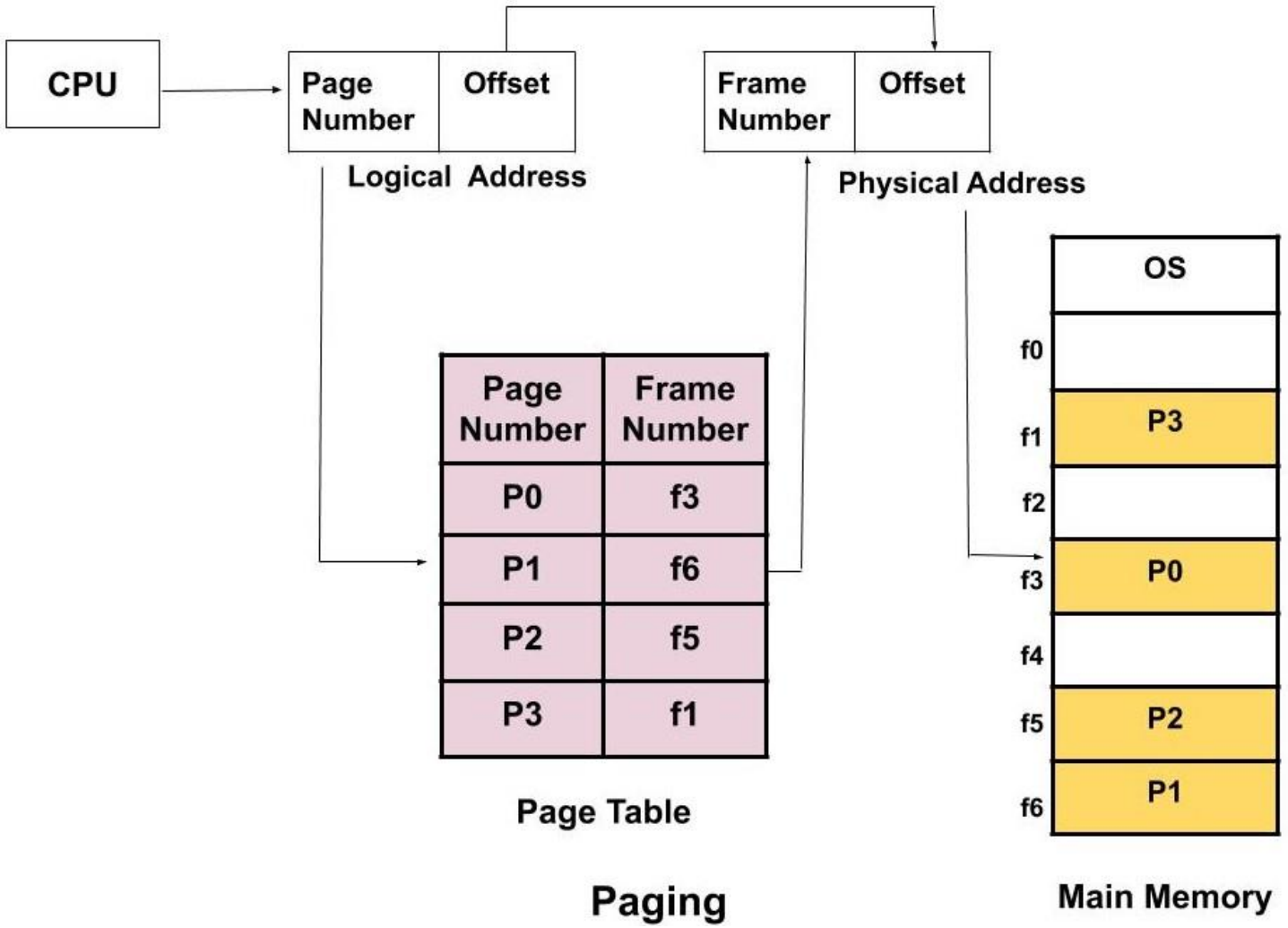
TRANSLATING LOGICAL ADDRESS INTO PHYSICAL ADDRESS

- The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

HOW IS THE TRANSLATION DONE?



TRANSLATING VIRTUAL (LOGICAL) ADDRESS TO PHYSICAL ADDRESS (MMU)

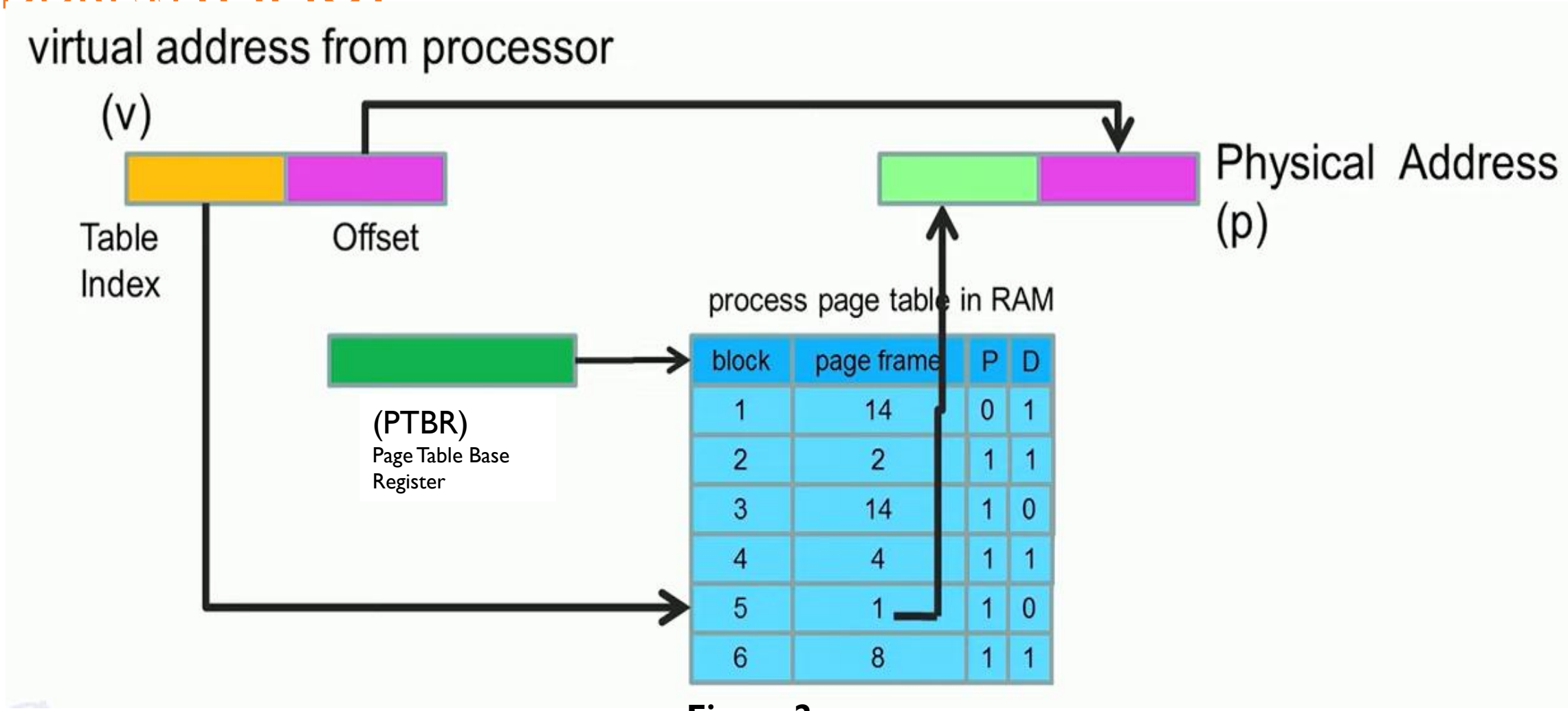
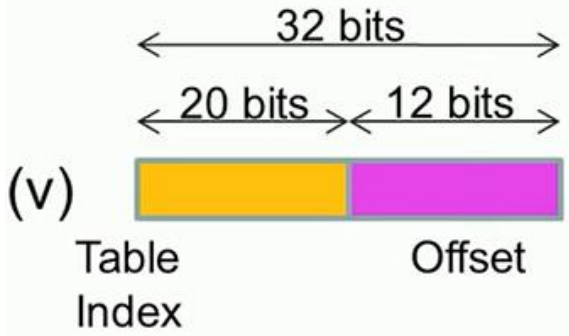


Figure 2

TRANSLATION EXAMPLE IN A 32-BIT SYSTEM

If each page frame is of size 4 KB, then
12 bits are required to address a page.
Thus offset is of 12 bits and index of 20 bits.



Number of entries in page table is 2^{20}

(around 4MB)

process page table in RAM

block	page frame	P	D
1	14	0	1
2	2	1	1
3	14	1	0
4	4	1	1
5	1	1	0
6	8	1	1

Note that the page table needs to be in 4MB contiguous memory.

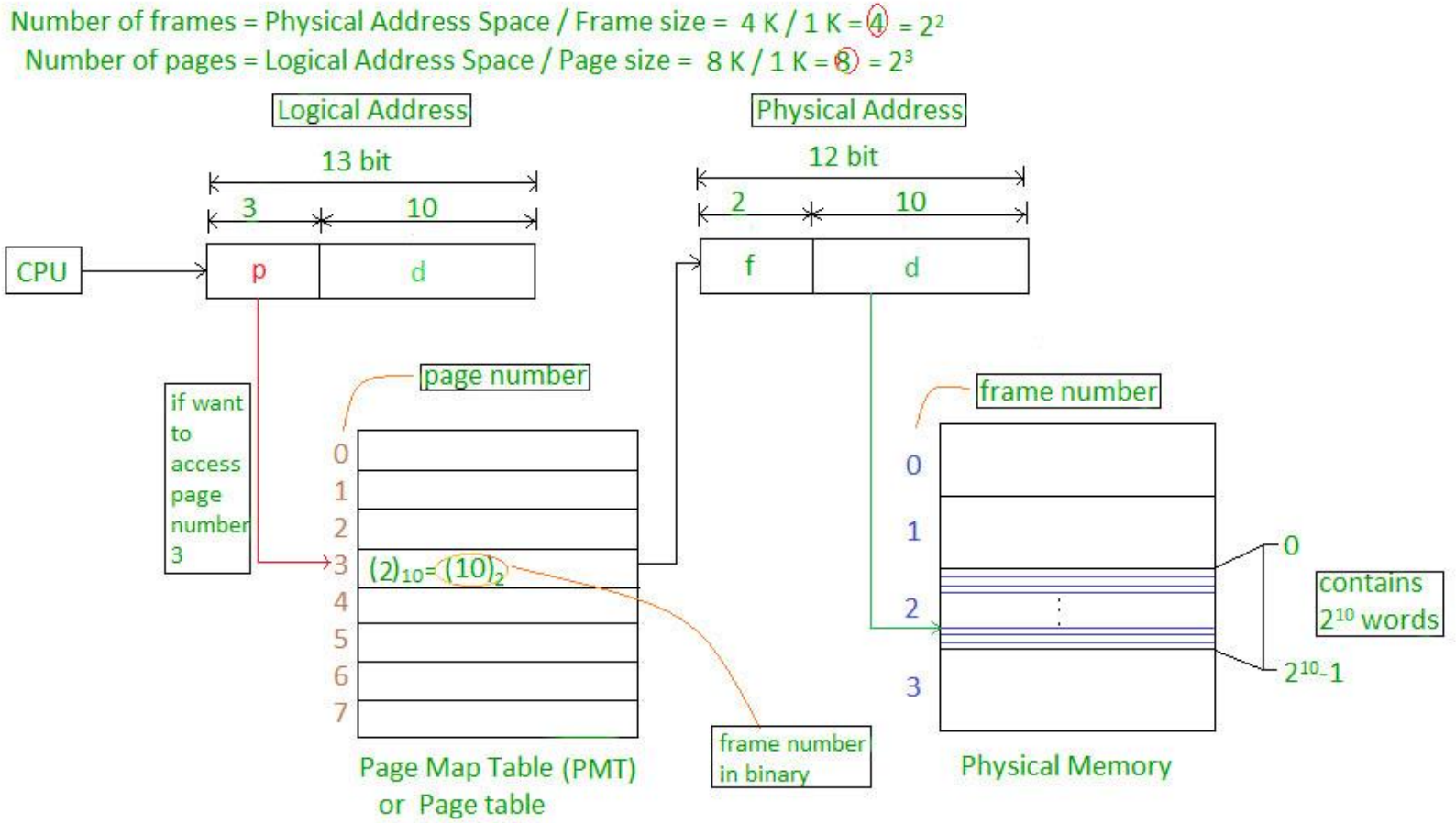
This is very large!

EXAMPLE:

- If Logical Address = 31 bits, then Logical Address Space = 2^31 words = 2 G words (1 G = 2^30)
- If Logical Address Space = 128 M words = 2^7 * 2^20 words, then Logical Address = $\log_2$ 2^27 = 27 bits
- If Physical Address = 22 bits, then Physical Address Space = 2^22 words = 4 M words (1 M = 2^20)
- If Physical Address Space = 16 M words = 2^4 * 2^20 words, then Physical Address = $\log_2 2^{24}$ = 24 bits

EXAMPLE:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)



QUESTION:

- Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?

Size of the page table = no. of entries * size of each entry

No. of bits required for offset (d) = 12

No. of bits required to index the page table (p) = 32 – 12 = **20**

Hence, No. of entries in page table = 2^{20}

Size of each entry in the page table (f) = 26-12 = **14**

Size of the page table = 2^{20} * 14 = 1.75 MB (2MB approx.)

TRANSLATING LOGICAL ADDRESS INTO PHYSICAL ADDRESS

- **Advantages:**
  - It allows to store parts of a single process in a non-contiguous fashion.
  - It solves the problem of external fragmentation.

- **Disadvantages:**
  - It suffers from internal fragmentation.
  - There is an overhead of maintaining a page table for each process.
  - The time taken to fetch the instruction increases since now two memory accesses are required.

TRANSLATION LOOKASIDE BUFFER | TLB | PAGING

- Page table due to its large size can't be stored in fast cache memory. Hence, it is stored in RAM.
- Problem: Two memory access are required to access a byte (word).
- Solution: Use of a special, small, fast-lookup hardware cache called Translation Lookaside Buffer (TLB).
- Translation Lookaside Buffer (TLB) is a solution that tries to reduce the effective access time.
- However, TLB must be kept small (typically between 32 and 1024 entries.
- Being a hardware, the access time of TLB is very less as compared to the main memory.

Structure

Page Number	Frame Number

Translation Lookaside Buffer

TRANSLATING LOGICAL ADDRESS INTO PHYSICAL ADDRESS

- In a paging scheme using TLB
- The logical address generated by the CPU is translated into the physical address using following steps-

Step 1: CPU generates a logical address consisting of two parts-

- **Page Number**
- **Page Offset**

Step 2:

- **TLB is checked to see if it contains an entry for the referenced page number.**
- **The referenced page number is compared with the TLB entries all at once.**

TRANSLATING LOGICAL ADDRESS INTO PHYSICAL ADDRESS

Case-01: If there is a TLB hit-

- **If TLB contains an entry for the referenced page number, a TLB hit occurs.**
- **In this case, TLB entry is used to get the corresponding frame number for the referenced page number.**

Case-02: If there is a TLB miss

- **If TLB does not contain an entry for the referenced page number, a TLB miss occurs.**
- **In this case, page table is used to get the corresponding frame number for the referenced page number.**
- **Then, TLB is updated with the page number and frame number for future references.**

Step-03:

- **After the frame number is obtained, it is combined with the page offset to generate the physical address.**
- **Then, physical address is used to read the required word from the main memory.**

TRANSLATING LOGICAL ADDRESS INTO PHYSICAL ADDRESS

Case-01: If there is a TLB hit-

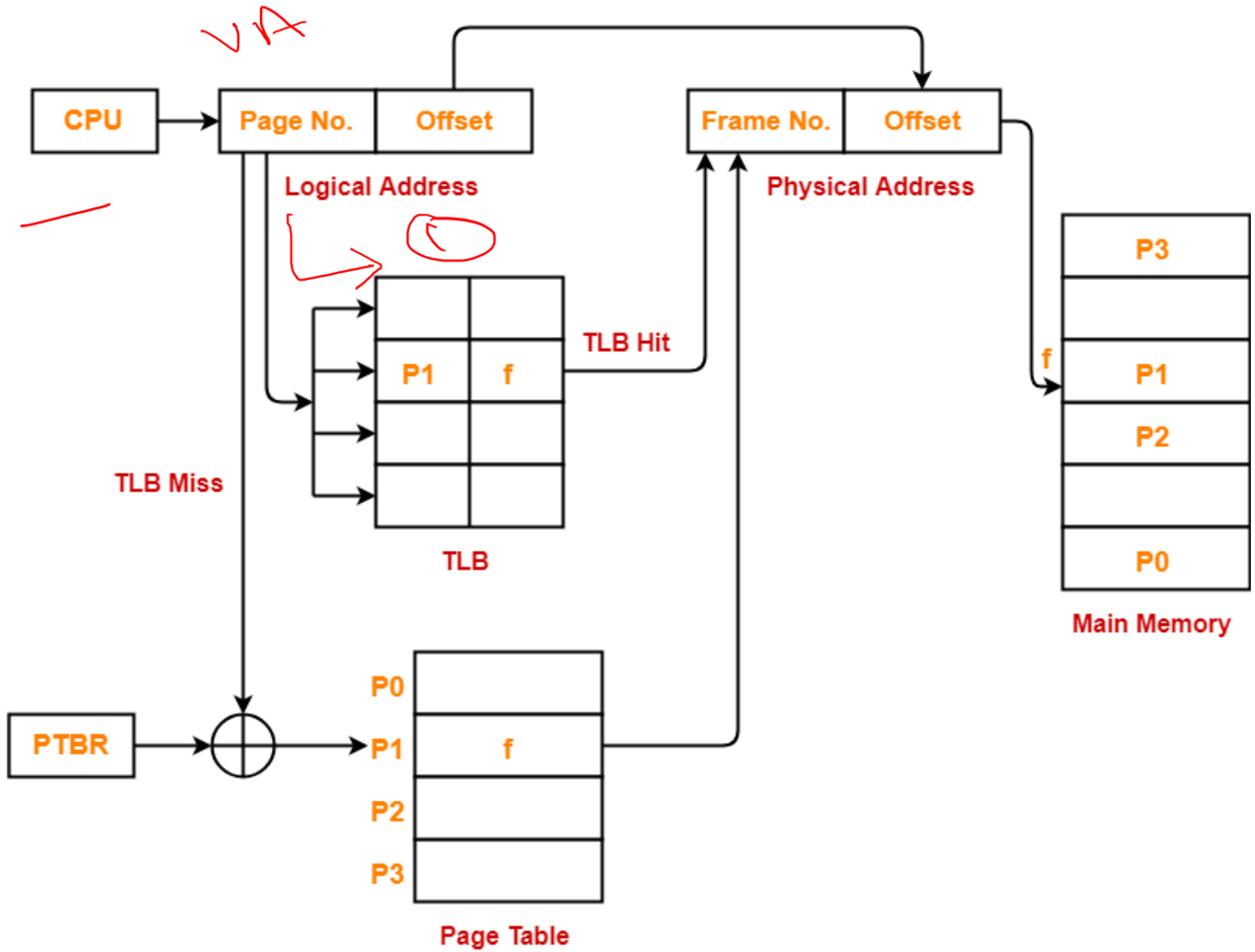
- **If TLB contains an entry for the referenced page number, a TLB hit occurs.**
- **In this case, TLB entry is used to get the corresponding frame number for the referenced page number.**

Case-02: If there is a TLB miss

- **If TLB does not contain an entry for the referenced page number, a TLB miss occurs.**
- **In this case, page table is used to get the corresponding frame number for the referenced page number.**
- **Then, TLB is updated with the page number and frame number for future references.**

Step-03:

- **After the frame number is obtained, it is combined with the page offset to generate the physical address.**
- **Then, physical address is used to read the required word from the main memory.**



Translating Logical Address into Physical Address

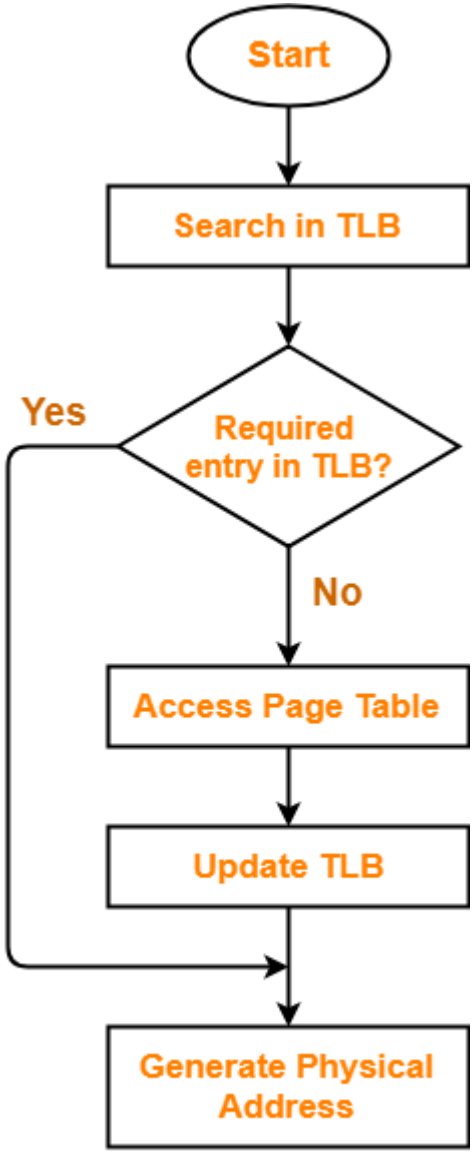
1 EAT

2 Hit

3 Miss

Δ TLB access time

memory access time



Flowchart

EFFECTIVE ACCESS TIME

Effective Access Time =

Hit ratio of TLB x { Access time of TLB + Access time of main memory }

+

Miss ratio of TLB x { Access time of TLB + 2 x Access time of main memory }

90 % hit ratio 0.9

10 % miss ratio 0.1

TLB

- **Advantages:**
  - TLB reduces the effective access time.
  - Only one memory access is required when TLB hit occurs..

- **Disadvantages:**
  - After some time of running the process, when TLB hits increases and process starts to run smoothly, a context switching occurs.
  - The entire content of the TLB is flushed.
  - Then, TLB is again updated with the currently running process.

PROBLEM-01:

- A paging scheme uses a Translation Lookaside buffer (TLB). A TLB access takes 10 ns and a main memory access takes 50 ns. What is the effective access time (in ns) if the TLB hit ratio is 90% and there is no page fault?

- 54
- 60
- 65
- 75

EAT = Hit * {TLB + memory}

+

miss * {TLB + 2 memory}

PROBLEM-01:

A paging scheme uses a Translation Lookaside buffer (TLB). A TLB access takes 10 ns and a main memory access takes 50 ns. What is the effective access time (in ns) if the TLB hit ratio is 90% and there is no page fault?

Given-
- TLB access time = 10 ns
- Main memory access time = 50 ns
- TLB Hit ratio = 90% = 0.9

Calculating TLB Miss Ratio-

TLB Miss ratio
= 1 – TLB Hit ratio
= 1 – 0.9
= 0.1

Calculating Effective Access Time-

Substituting values in the above formula, we get-
Effective Access Time
= 0.9 x { 10 ns + 50 ns } + 0.1 x { 10 ns + 2 x 50 ns }
= 0.9 x 60 ns + 0.1 x 110 ns
= 54 ns + 11 ns
= 65 ns
Thus, Option (C) is correct.

PROBLEM-02:

- A paging scheme uses a Translation Lookaside buffer (TLB). The effective memory access takes 160 ns and a main memory access takes 100 ns. What is the TLB access time (in ns) if the TLB hit ratio is 60% and there is no page fault?

- 54
- 60
- 20
- 75

PROBLEM-02:

- A paging scheme uses a Translation Lookaside buffer (TLB). The effective memory access takes 160 ns and a main memory access takes 100 ns. What is the TLB access time (in ns) if the TLB hit ratio is 60% and there is no page fault?
- Given-
- Effective access time = 160 ns
- Main memory access time = 100 ns
- TLB Hit ratio = 60% = 0.6

Calculating TLB Miss Ratio-

TLB Miss ratio
= 1 – TLB Hit ratio
= 1 – 0.6
= 0.4

Calculating TLB Access Time-

Let TLB access time = T ns.
Substituting values in the above formula, we get-
160 ns = 0.6 x { T + 100 ns } + 0.4 x { T + 2 x 100 ns }
160 = 0.6 x T + 60 + 0.4 x T + 80
160 = T + 140
T = 160 – 140
T = 20

THANKYOU