Let,

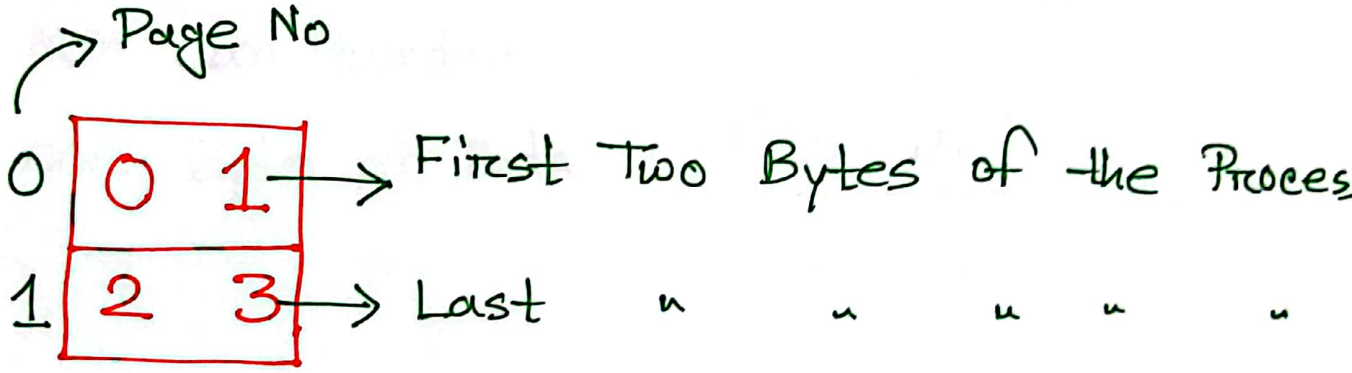
Process Size = 4 Byte

Page Size = 2 Byte

Number of Process Pages Required

For this Process = 4/2

= 2



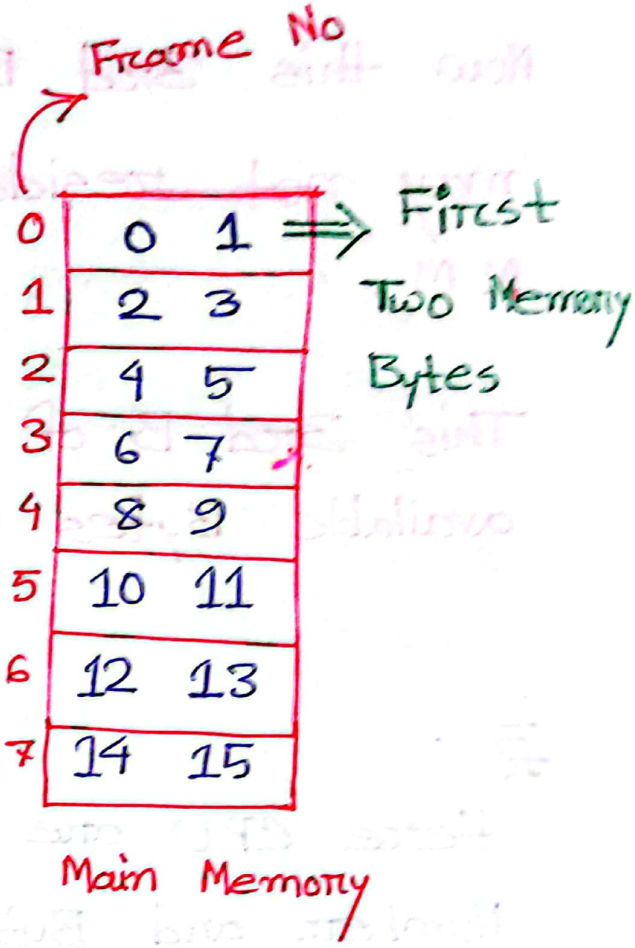
Let, Main Memory Size = 16 Byte

Frame Size = Page Size = 2 Byte

Num of Frames = 16/2 = 8

- We Can Store 16 Bytes of Data in this M.M

- Each Byte of M.M can contain one Byte of Data



Now CPU does not know about any info related to Paging.

Lets, CPU are executing this P1 Process. CPU need each instruction / each byte for executing the Process.

Let's, CPU wants Byte 3 of Process P1

Now this 3rd Byte of Process 1 may not reside in 3rd Byte of M.M.

This 3rd B. of Process 1 can be in any available Bytes of M.M

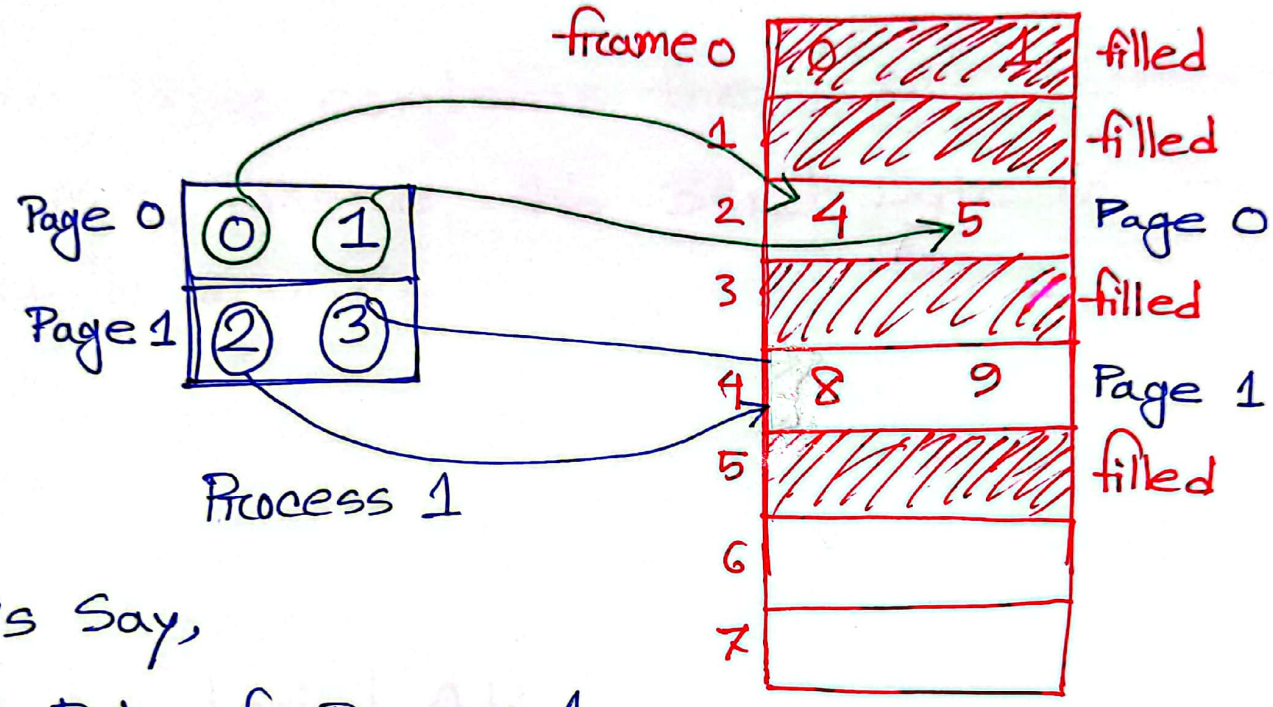
Here CPU are generating address Page Number and Byte Number of Process 1

Page 1	Byte 3

But this address is not actual memory address of Byte 3 of Process 1.

Byte 3 actually stores in 9th Byte of Main Memory.

MMU converts CPU generated logical address into actual Physical Address of M.M



Let's Say,

0th Byte of Process 1

will be stored at 4th Memory Byte of M.M.

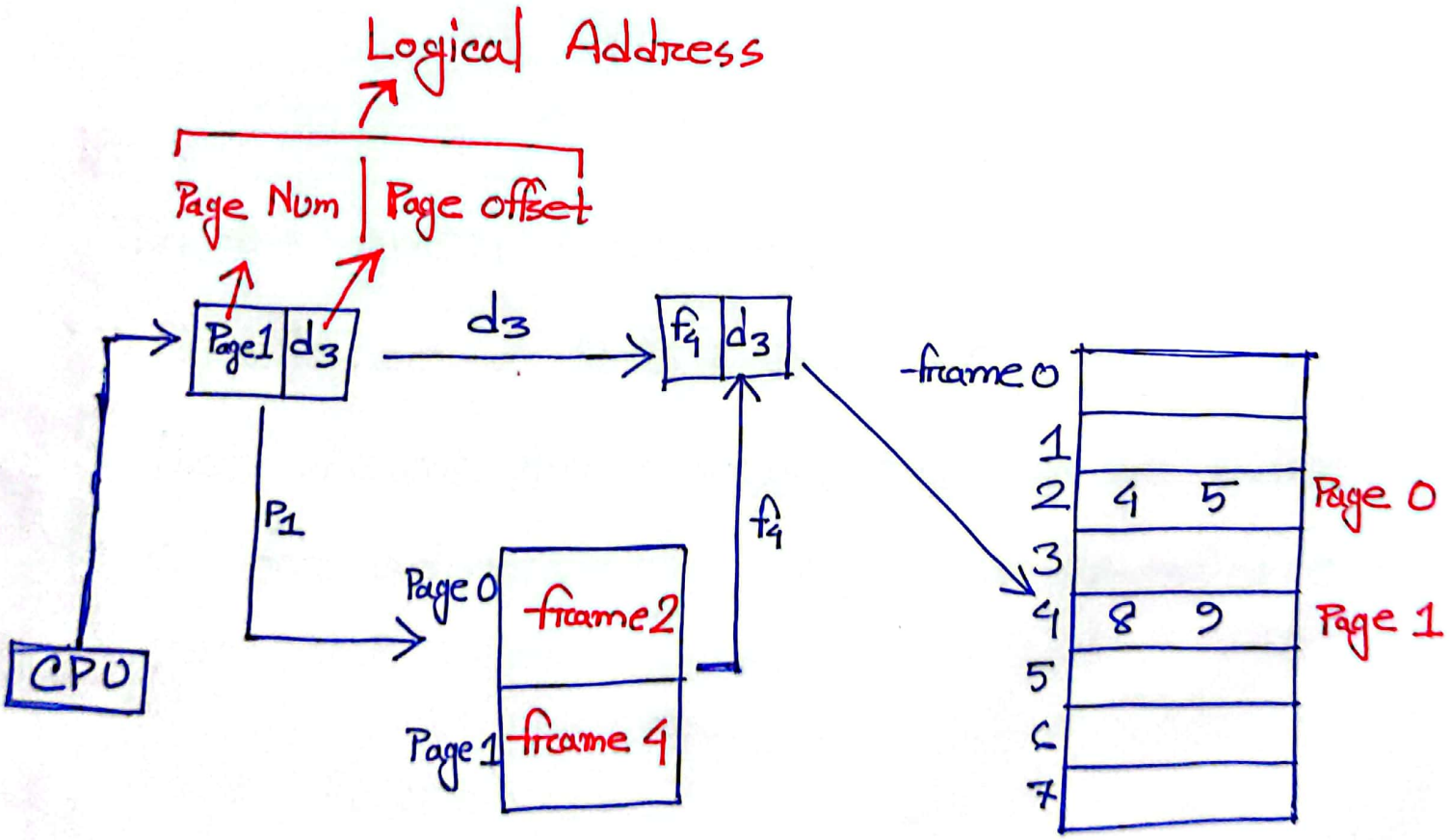
1th Byte in 5th Byte of M.M

:

:

MMU uses Page Table to perform this conversion.

Page Table contains the frame number of M.M where this 3rd Byte of Process1 actually stores.



From 1 bit, we get combination = $2 = 2^1$

From 2 " , ⟶ $4 = 2^2$

□ → either 0 or 1 → so 2 combinations

one bit

so using n bits we can address 2^n number of memory location.

In the given example;

Page offset

Page Size = 2 Bytes

That means each page contain 2 bytes:

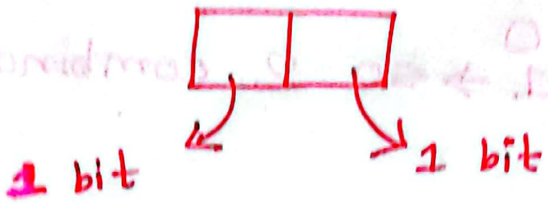
so for addressing these 2 bytes of each page we need 1 bit

Page Number

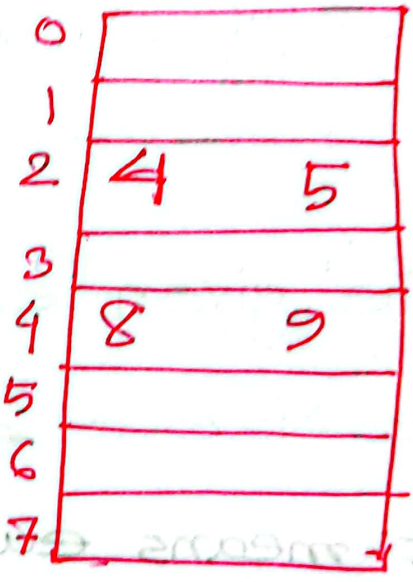
Page Number = 2

so here, we again need 1 bit to address 2 Pages

Logical Address:



Page 0	0	1
Page 1	2	3

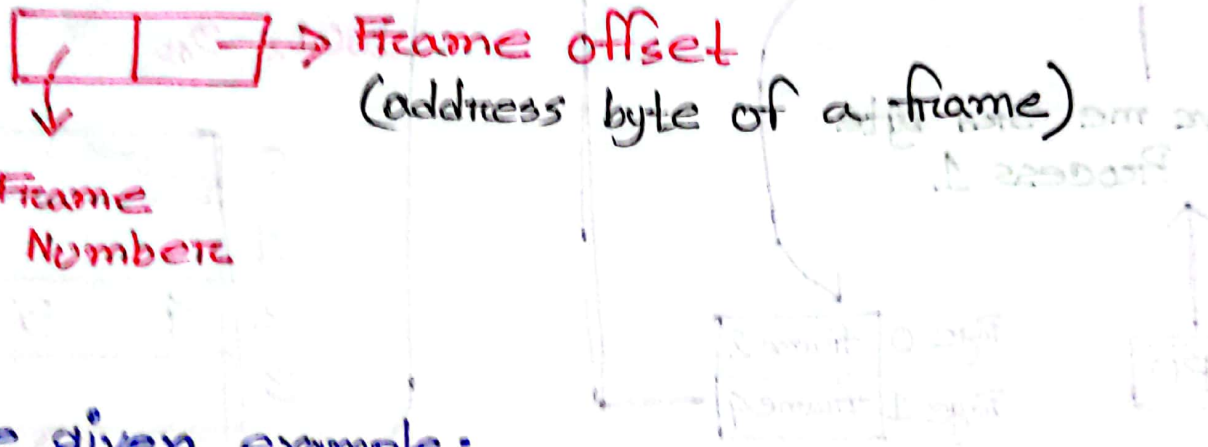


1	1

→ indicates second byte where page = 1

Each page contains 2 bytes. We use 0 to address first byte and 1 to address second byte of each process

Number of bits required to represent physical address depend upon the size of the Main Memory.



⇒ In the given example:

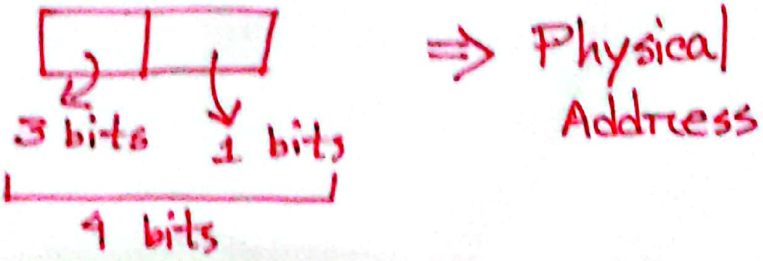
Frame Size = Page Size = 2

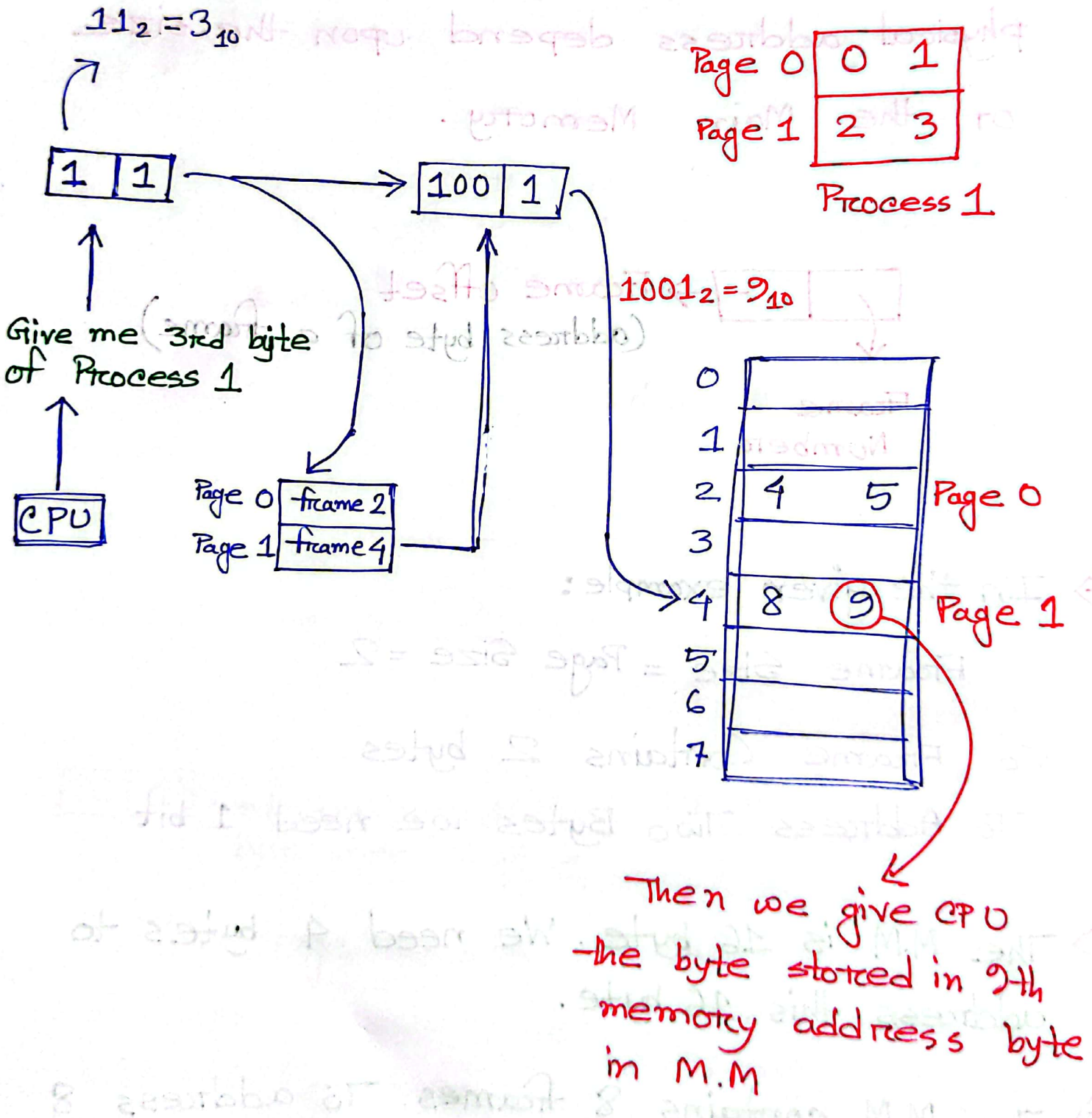
So Frame Contains 2 bytes.

To Address Two Bytes we need 1 bit

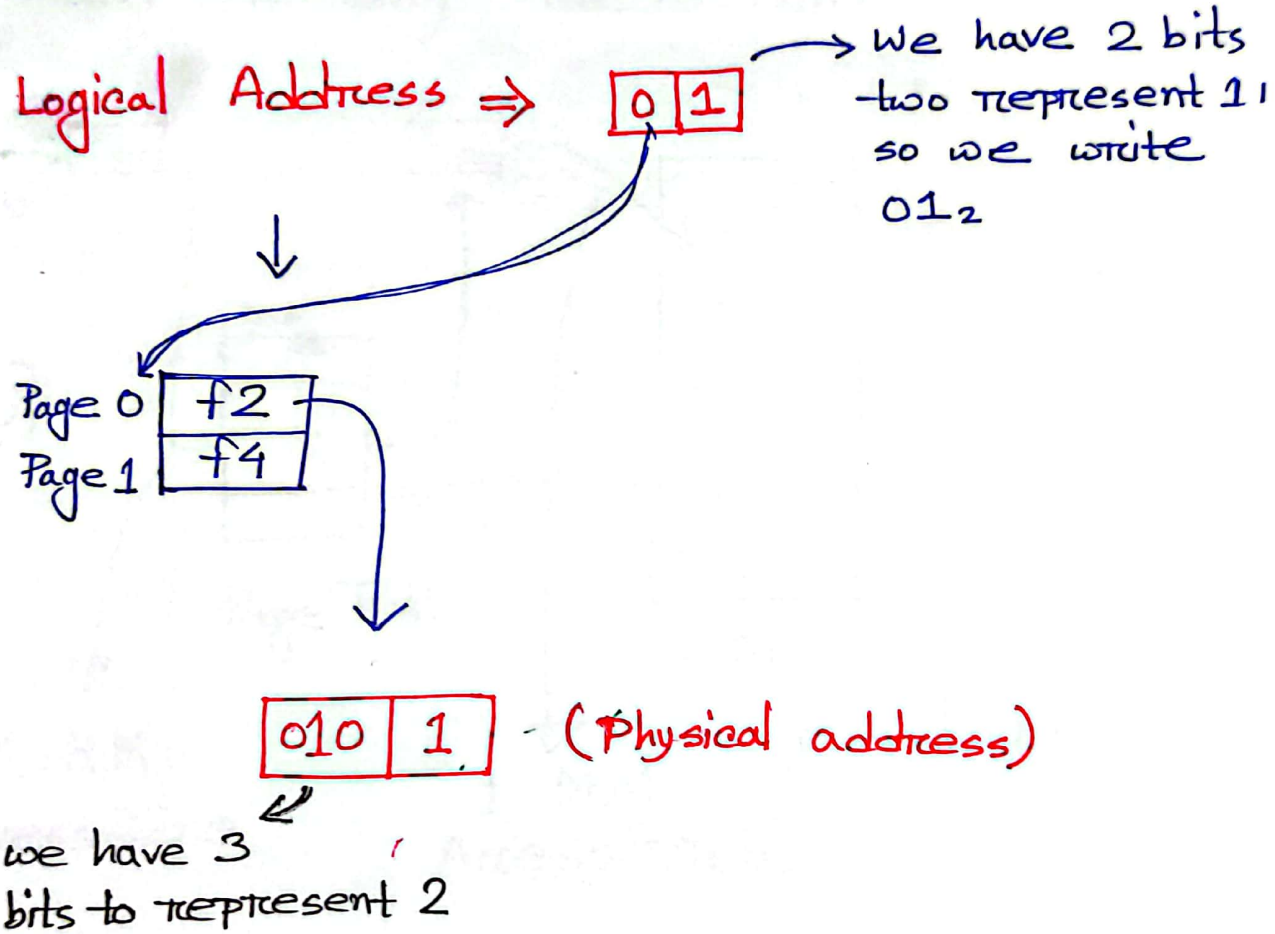
⇒ The M.M is 16 byte. We need 4 bytes to address this 16 byte.

⇒ The M.M contains 8 frames. To address 8 frames we need 3 bits.



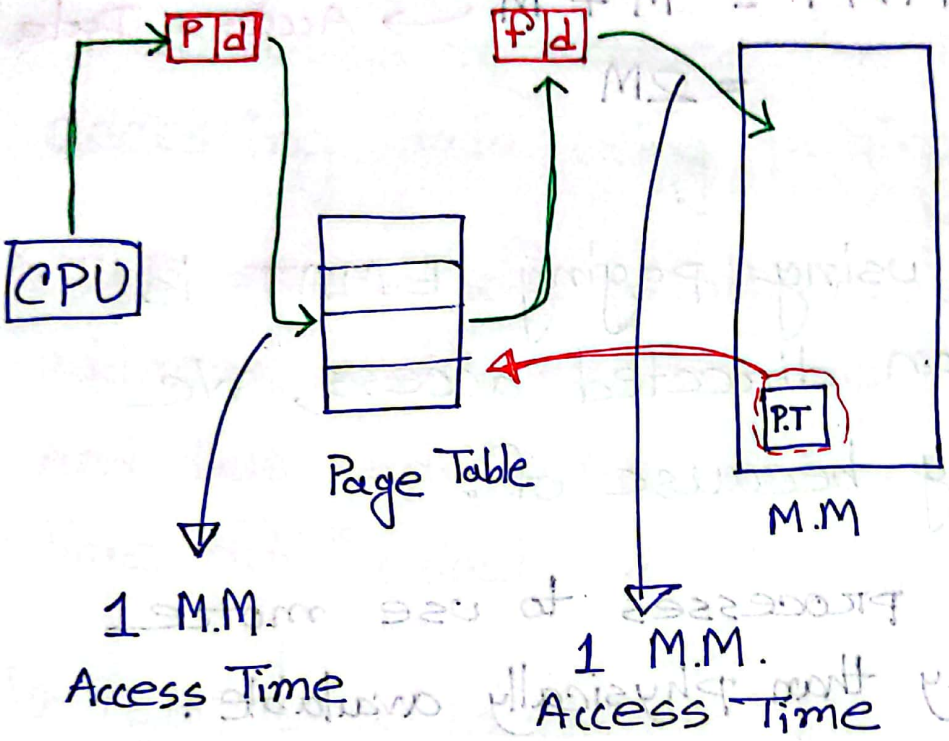

$11_2 = 3_{10}$
1 1
100 1
Page 0 0 1
Page 1 2 3
Process 1
$1001_2 = 9_{10}$
Give me 3rd byte of Process 1
CPU
Page 0 frame 2
Page 1 frame 4
0
1
2 4 5
3
4 8 9
5
6
7
Page 0
Page 1
Then we give CPU the byte stored in 9th memory address byte in M.M

田 if CPU wants 1-th byte



$0101_2 = 5_{10}$ ⇒ so we pick byte from 5th byte of M.M.

Let, Main Memory Access Time = M



EMAT (Effective Memory Access Time) ⇒ average time it takes for the CPU to access data from the main memory.

If we map entire address-space directly to the physically possible memory addresses in M.M., then EMAT = M

When we use paging, then

$$EMAT = M + M \rightarrow \text{Access Data}$$
$$\rightarrow \text{L.A to P.A}$$
$$= 2M$$

Though by using paging, EMAT is higher than directed access, we use paging because of:

* enables processes to use more memory than physically available.
* memory isolation & protection.
* demand paging.

TLB (Translation Look Aside Buffer)

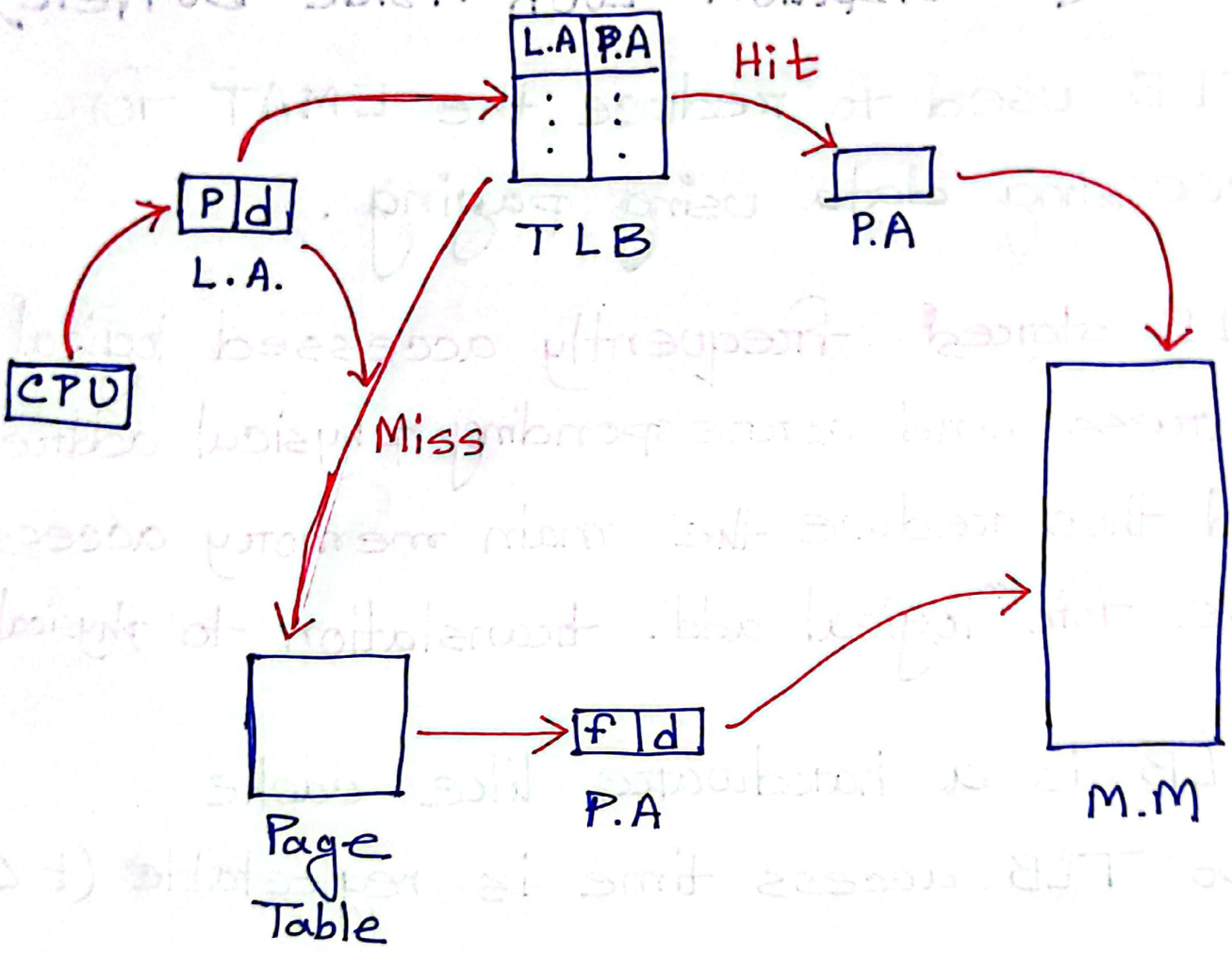
↳ TLB used to reduce the EMAT for accessing data using paging.

↳ TLB stores frequently accessed logical address and corresponding physical address. And thus reduce the main memory access time for "logical add. translation to physical A"

↳ TLB is a hardware like cache

↳ So TLB access time is neglectable ($t \ll M$)

- TLB stores the entries of currently running process. When context switching happens, all TLB entries will be invalid.



* Hit means CPU find corresponding physical address of requested logical address. Miss means, no found → go to Page Table.

- Hit Rate $= \dfrac{\text{Number of TLB Hit}}{\text{Total Number of Memory Access Requests}}$

- Hit Rate refers to the percentage of memory access requests for which the required translated P.A is found.

- $EMAT_{TLB} =$

$$\underbrace{(\text{Hit Rate}).(t+M)}_{\text{Incase of Hit}} + \underbrace{(1-\text{Hit Rate}).(t+2M)}_{\text{Incase of Miss}}$$

- $M = 100\,ns$, Hit Rate $= 90\%$, $t = 20ns$

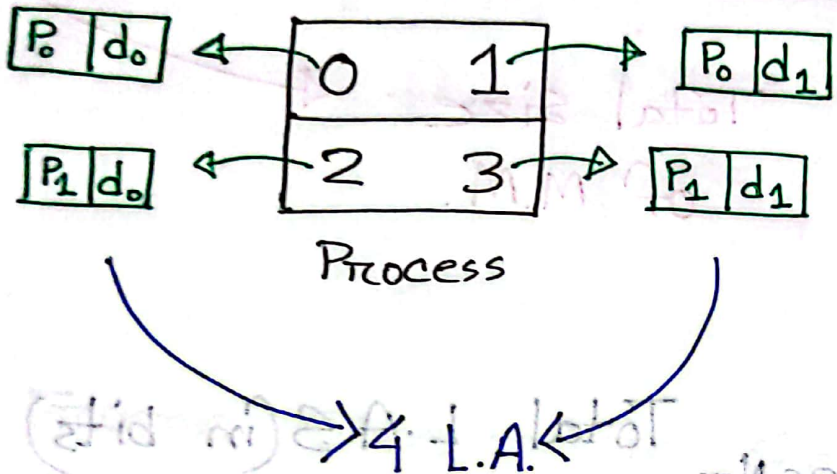
  $EMAT_{TLB} = ?$

$\Rightarrow$ $EMAT_{TLB} = 0.90 * (20+100) + (1-0.9)(20+200)$

$= 130\ ns$

In Previous Example :

Process Size = 4 Byte



4 L.A. for a Process Size of 4 Byte.

P	d
0	0
0	1
1	0
1	1

⇒ Logical Address Space for this Process

Logical Address Space = Total Number of L.A

$= 2^{L.A}$

here, L.A is of 2 bits. So L.A.S $= 2^2 = 4$

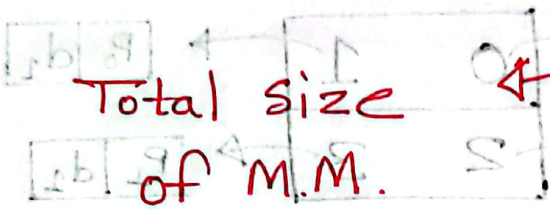
(*) L.A is in bits. L.A.S (2^{LA}) is Bytes

P.A.S = $2^{(P.A)}$ → in bits

→ in Byte

Here, P.A is of 4 bits. So P.A.S = 2^4

= 16



Num of Pages, $N = \frac{\text{Total L.A.S (in bits)}}{\text{Page Size (in bits)}}$

Bits required to Address N number of pages, $P = \lceil \log_2 N \rceil$

Bits required to Address each Byte of each page, Page offset (d) = $\lceil \log_2 (\text{Page Size}) \rceil$

L.A ⇒ | P | d |

- Frame Size = Page Size
- Number of frames = $\frac{\text{Physical A. Space}}{\text{Page Size}}$
- #Bits to Address M Frames, f = $\lceil \log_2 M \rceil$
- Physical Address = | f | d |

Page Table Size = N * e

- N = Number of entries in Page Table
- e = entry size

Logical Address = 32 bits

Physical Address = 27 bits

Page Size = 4 KB

Page Table Entry Size = 3B

What is the Page Table Size?

⇒

L.A.S = 2^{32} B

P.A.S = 2^{27} B

$$N = \frac{2^{32} \text{ Bytes}}{4 \text{ KB}}$$

$$= \frac{2^{32} \text{ Bytes}}{2^2 \times 2^{10} \text{ Bytes}}$$

$$= \frac{2^{32}}{2^{12}} \text{ Bytes}$$

$$= 2^{32-12} = 2^{20} \text{ Bytes}$$

$$= 1 \text{ MB}$$

1 byte = 2^0 bytes

1 KB = 2^{10} bytes

1 MB = 2^{20} bytes

1 GB = 2^{30} bytes

1 B = 8 = 2^3 bits

~~Page Table Size = 3 MB~~

Page Table Size = 1MB * 3 B

$= 1 * 2^{20}$ B * 3 B

$= 3 * 2^{20}$ B

= 3 MB

⇐

* e may not be given.

In that case,

Page Table Size = N = Num of Pages.

Logical Address = 64 bits

Page Size = 16 KB

Physical Memory Size = 8 GB

① What is the Page Table Size / Num of Entries in a Single-level Page Table?

② Page Number & Page offset bit numbers?

⇒

① Page Table Size, $N = \frac{L.A.S}{\text{Page Size}}$

$L.A.S = 2^{L.A}$

$= 2^{64}$

Page Size = 16 KB $= 2^4 \times 2^{10}$ B

$= 16 \times 2^{10}$ B / Page

$= 4 \times 4 \times 2^{10}$ B $= 2^4 \times 2^{10}$ B/Page

$= 2^{14}$ B / Page

So, Num of Entries for Single-level P.T $= \frac{2^{64}}{2^{14}}$ page

$= 2^{50}$ pages

So, Num of Pages $= 2^{50}$

② Num of Bits to represent Pages =

$$\lceil \log_2 50 \rceil = 50 \text{ bits}$$

Num of Bits required for Page offset =

$$\lceil \log_2 16\,KB \rceil$$

$$= \log_2 2^{14}$$

$$= 14 \text{ bits}$$

Inverted Paging:

⇒ in single-level paging, if 10 processes are running in CPU currently, then 10 different page-table will be needed. Each page table for each page.

⇒ Currently running processes' page table must be in Main Memory.

⇒ So, we have to use Main Memory frames for storing many page tables.

⇒ Storing multiple page table (one PT for one Process) can lead to increased in Memory Usages. Thus, memory overflow can happen.

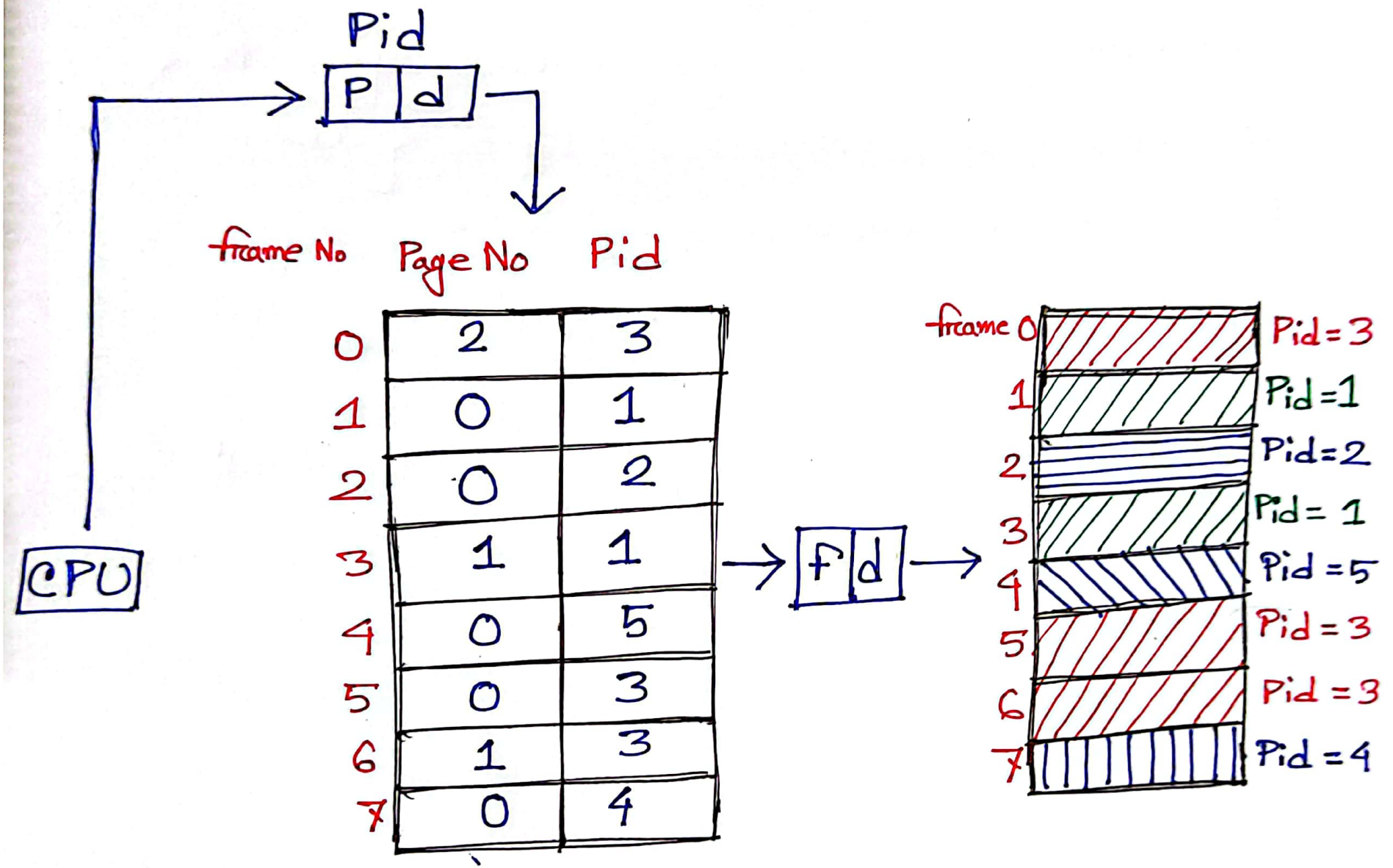
⇒ In inverted paging, a single global page-table will be shared by all processes. Here, we do not use unique page table for each process.

- Number of Entries in Inverted Page Table = Num of Frames in M.M

- If we want to find frame number of logical address generated by CPU, then we have to perform linear search for finding corresponding process's page number.

- Linear Seach is inefficient in case of time complexity.

- So, inverted page table can reduce memory usage but also will lead to higher time complexity.


Pid
P
d
CPU
frame No
Page No
Pid
0 2 3
1 0 1
2 0 2
3 1 1
4 0 5
5 0 3
6 1 3
7 0 4
f
d
frame 0
1
2
3
4
5
6
7
Pid=3
Pid=1
Pid=2
Pid= 1
Pid =5
Pid =3
Pid =3
Pid =4

P.A.S = 256 MB

L.A.S = 4 GB

Frame Size = 4 KB

Here, frame size = page size = $4KB = 2^{12} B$

offset bit number = $\lceil \log_2 2^{12} \rceil$

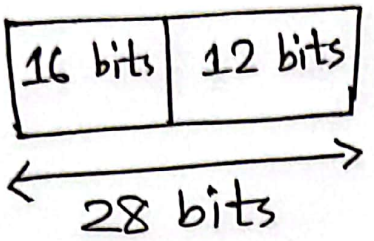
= 12 bits

Num of frames in 256MB M.M = $\frac{256\ MB}{4\ KB}$

$= \frac{2^8 \times 2^{20}}{2^2 \times 2^{10}} = 2^{28-12}$

$= 2^{16}$

Num of bits required to address 2^{16} frames = $\lceil \log_2 2^{16} \rceil = 16$ bits



$\Rightarrow$ Physical Address

(11)

$$\text{Num of Pages} = \frac{4\,GB}{4\,KB}$$

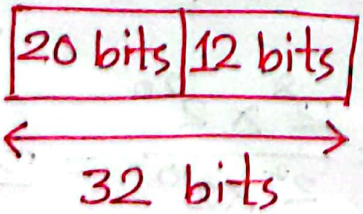
$$= \frac{4 \times 2^{30}}{4 \times 2^{10}} = \frac{2^{32}}{2^{12}}$$

$$= 2^{20}$$

Bits required to address 2^{20} pages $= \lceil \log_2 20 \rceil$

$= 20$ bits

offset bit number $= 12$ bits



$\Rightarrow$ Logical Address