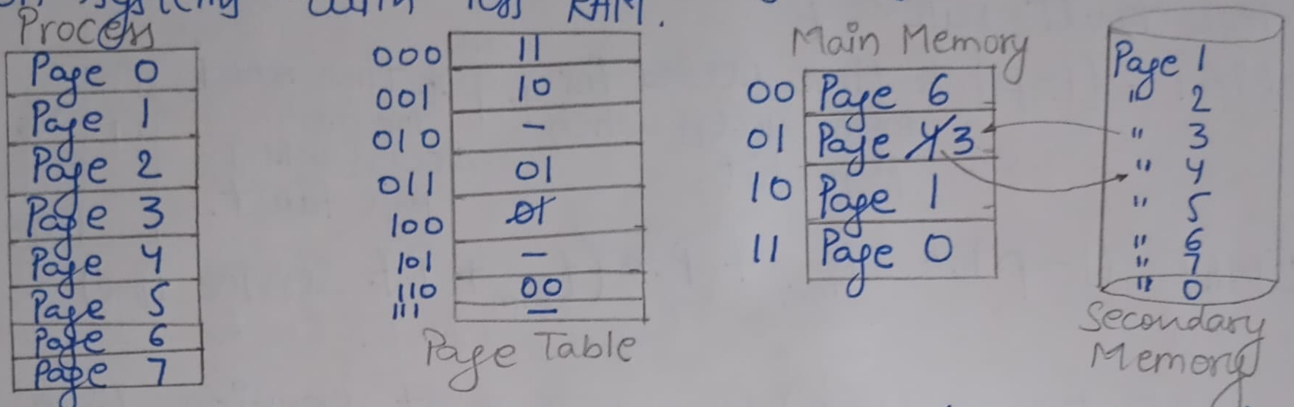
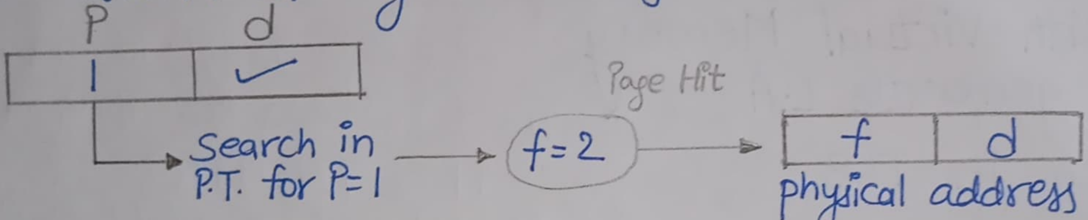
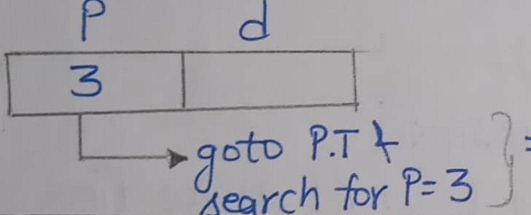
CH 06 Virtual Memory

*Virtual Memory: It is a memory managment technique used by operating system to give the appearance of a large, continuous block of memory to applications, even if the physical memory (RAM) is limited. It allows the system to compensate for physical memory shortages, enabling larger applications to run on systems with less RAM.



Explanation: CPU generates logical address (Virtual Address)





= Page fault = OS does page fault Service

Bring faulted page from secondary memory to main memory. (A page is replaced if needed & update page table) After service the instruction which raised page fault will restart.

*Demand Paging: When a process arrives, a few initial pages are brought to main memory and rest are brought on demand.

*Pure demand Paging: When a process arrives, none of its pages are brought to main memory and pages are brought into main memory on demand.

*Page Fault: when demanded pages is not available in physical memory.

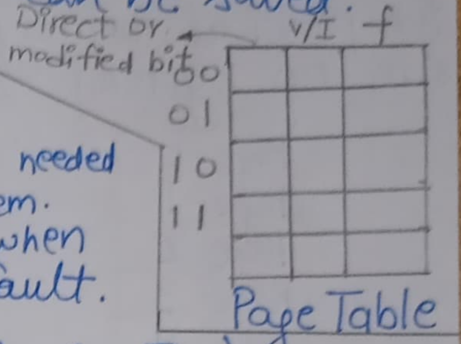
Page fault service time: Time needed to service a page fault, after page fault is raised.

Page fault rate: fraction of time page fault occurs.

* How to Check Page Hit/Fault using valid/invalid bit → 0 invalid, → 1 valid

* Saving Page Swap Time : Direct or modified bit
→ 0 only read done in page
→ 1 write done in page (Page has become dirty)

Only dirty pages are copied back to secondary memory when replaced, hence write time of non-dirty replaced pages can be saved.



* Effective Memory Access Time

Page fault rate = p

EMAT = (1-p) * time needed for mem. access when page hit + p * time needed for mem. access when page fault.

$$EMAT = (1-p) * 2t_{mm} + P * (t_{mm} + P.f. \text{ Service Time})$$

$2t_{mm}$ → one for P.T, → One for Content

t_{mm} → for P.T. Check

$$= t_{mm} + (1-p)t_{mm} + p * pt \text{ service time}$$

* TLB with Virtual Memory

P = Page fault in mm
(1 − P) = page hit in mm
H = TLB hit
(1 − H) = TLB fault

CPU generates L.A
⇓
search in TLB

TLB Hit → Physical Address (PA) ⇓ access content from mm $= t_{TLB} + t_{mm}$

TLB Miss → Search P.T

- Page Hit → P.A → access content from mm $= t_{TLB} + 2t_{mm}$
- Page fault → Page fault Service $= t_{TLB} + t_{mm} + P.f \text{ service time}$

Effective Memory Access Time with TLB :

$$EMAT = H(t_{TLB} + t_{mm}) + (1-H)\left[(1-p)*(t_{TLB} + t_{mm} + t_{mm})\right.$$
$$\left. + p*(t_{TLB} + t_{mm} + p.f \text{ service time})\right]$$
$$= t_{TLB} + t_{mm} + (1-H)\left[(1-p)*t_{mm} + p * p.f \text{ servicetime}\right]$$

* Dirty Bit Included (Without TLB)

P.T search with L.A in mm

Page Hit → P.A ⇓ Content accessed $= t_{mm} + t_{mm}$

Page fault → Replaced page dirty or not

- non-dirty → P.f service without copying replaced page to disk $= t_{mm} +$ P.f. service time for non-dirty page
- dirty → P.f. service with replaced page copied to disk $= t_{mm} +$ P.f. service time for dirty page.

$$EMAT = (1-p)(2t_{mm}) + p[(1-M)(t_{mm} + \text{P.f.S.t for non-dirty page}) + M(t_{mm} + \text{P.f.S.t for dirty pages})]$$

M = % of replaced page which are dirty/modified.

* Dirty Bit with TLB

CPU generates L.A

↓

Search in TLB

- Physical Address (PA) ⇓ access content from mm $= t_{TLB} + t_{mm}$
- Same as Dirty Bit without TLB

$$EMAT = H \times (t_{TLB} + t_{mm}) + (1-H)[t_{TLB} + (1-p) \times 2t_{mm} + p[(1-M)(t_{mm} + \text{P.f.S.T for non-dirty}) + M(t_{mm} + \text{P.f.S.t for dirty})]]$$

* P.f service time when replaced page is not dirty:

= page transfer time from disk to mm + t_{mm} → for updating page table

* P.f.S.T for dirty page:

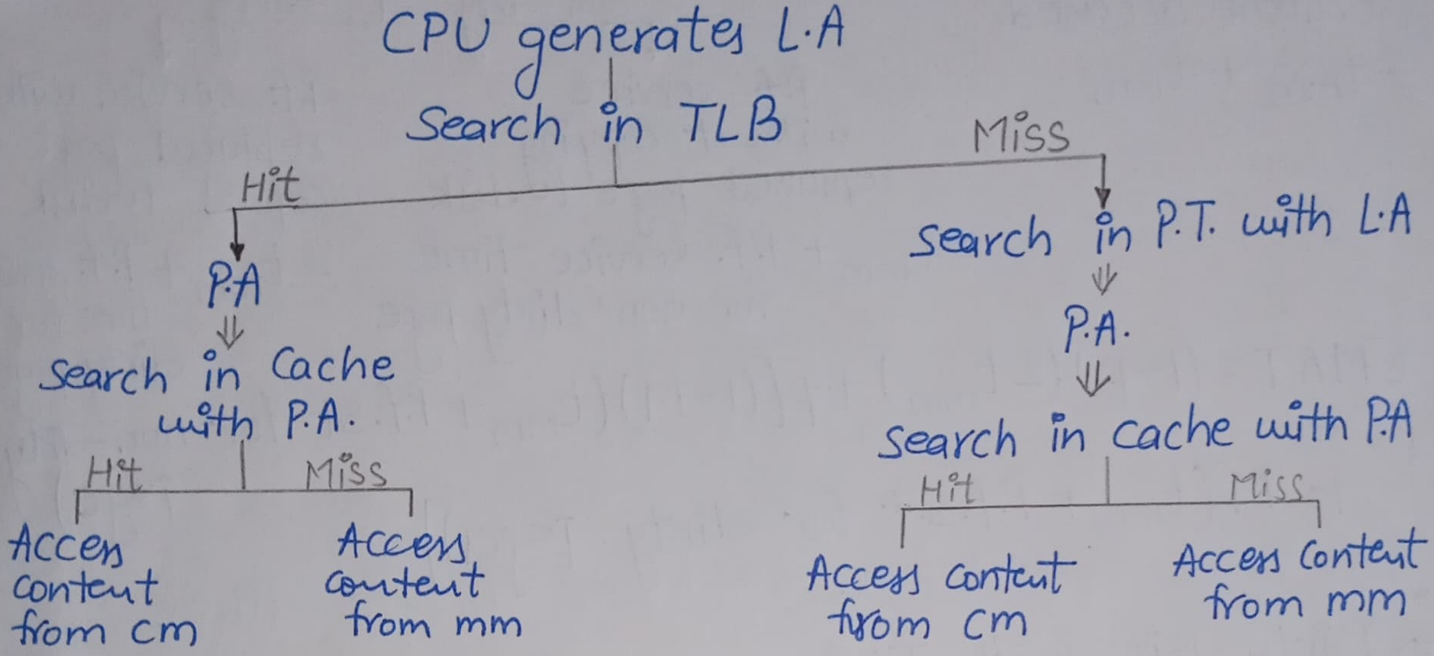
= Page transfer time from mm to disk for dirty page + Page transfer time from disk to mem for faulted page + t_{mm}

* Cache with TLB: Physically address cache => cache is accessed using P.A only.

Here, H_{cm} = cache hit ratio

t_{cm} = cache access time

Content access time with PA = $H_{cm} \times t_{cm} + (1-H_{cm})(t_{cm} + t_{mm})$



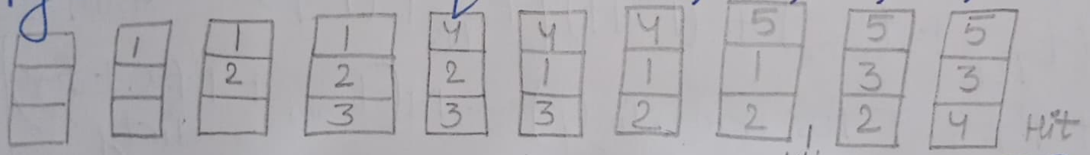
EMAT = $H_{TLB}[t_{TLB}$ + Content access time$]$ +
$(1-H_{TLB})[t_{TLB} + t_{mm}$ + content access time$]$

* Page Replacement Policies

1. First in First Out (FIFO): Replace the page which comes first in mm among all mm pages.

Assume: Number of frames = 3 (All empty initially)

Page reference sequence = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



	1	1	1	4	4	4	5	5	5
		2	2	2	1	1	1	3	3
			3	3	3	2	2	2	4

* Belady's Anomaly: It occurs only in FIFO. For some page reference sequences, increasing number of frames may increase number of page faults.

Advantages
1. Simple and easy to implement.
2. Low Overhead
To check which page to replace

Disadvantages
High no. of page faults
1. Poor Performance.
2. Suffers from Belady's Anomaly

2. Optimal Policy: Replace the page which is not going to be used for longest period of time.

	1	1	1	1			1			3	3	
		2	2	2	Hit	Hit	2	Hit	Hit	2	4	Hit
			3	4			5			5	5	

Advantages: Easy to Implement, Highly efficient (min. no. of page faults)

Disadvantages: Requires future knowledge (which is not practical), Time-Consuming.

3. Least Recently Used (LRU)

Replace the page which has not been used since longest period of time.

1	1	1	4	4	4	5			3	3	3
	2	2	2	1	1	1	Hit	Hit	1	4	4
		3	3	3	2	2			2	2	5

No. of page fault = 10

Advantages: Efficient, Doesn't suffer from Belady's Anomaly.

Disadvantages: Complex Implementation, Expensive, Requires hardware support.

4. Last In First Out (LIFO)

Replace the page which comes last in mm.

1	1	1	1			1			1	1	1
	2	2	2			2			2	2	2
		3	4	Hit	Hit	5	Hit	Hit	3	4	5

5. Most Recently Used (MRU)

Replace the page which has been refered most recently.

1	1	1	1			1		2	3		
	2	2	2			5		5	5	Hit	Hit
		3	4	Hit	Hit	4	Hit	4	4		

6. Counting Algorithms

Counting Algorithms looks at the number of accurances of a particular page and use this as the criterion for replacement.

Such Counting algorithms includes:

(i) LFU (Least Frequently Used)

(ii) MFU (Most Frequently Used)

ⓘ Least Frequently Used (LFU)

Replace the page which has been refered minimum number of times.

*Tie breaker=FIF

Assume: No. of frames = 3 (All empty initially)

Page reference sequence: 1 2 0 3 0 4 2 3 0 3 2

1	1	1	3		3	2	2			
	2	2	2		4	4	3	Hit	Hit	Hit
		0	0	Hit	0	0	0			

No. of Page Fault = 7

Page	Frequency
1	1
2	~~1~~ 2
3	~~1~~ 2
4	1
0	~~1~~ 2

(ii) Most Frequently Used (MFU)

Replace the page which has been refered maximum number of the times. → Tie break = FIFO

1	1	1	3		3			3		2
	2	2	2	Hit	2	Hit	Hit	0	Hit	0
		0	0		4			4		4

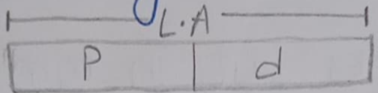
Page	Frequency
0	~~1~~ ~~2~~ 3
1	1
2	~~1~~ ~~2~~ 3
3	~~1~~ ~~2~~ 3
4	1

* Making Page Reference Sequence

Logical address or Virtual address } ⇒ Page Number

Option 1 : L.A (V.A) given in Binary/Hexadecimal

convert

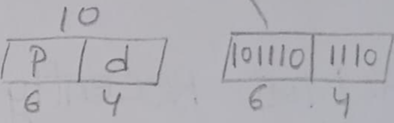


example : L.A = 10 bits

Page Size = 16 bytes = 2^4 B = 4 bits

L.A = 1011101110 ⇒ P = $(101110)_2$

= $(46)_{10}$



Option 2:

L.A (V.A) given in decimal

$$P_{no.} = \left[\frac{L.A}{Page\ Size}\right]$$

→ Integer value रख लो

d = L.A % Page Size

example : Page Size = 4B

0/4 = 0

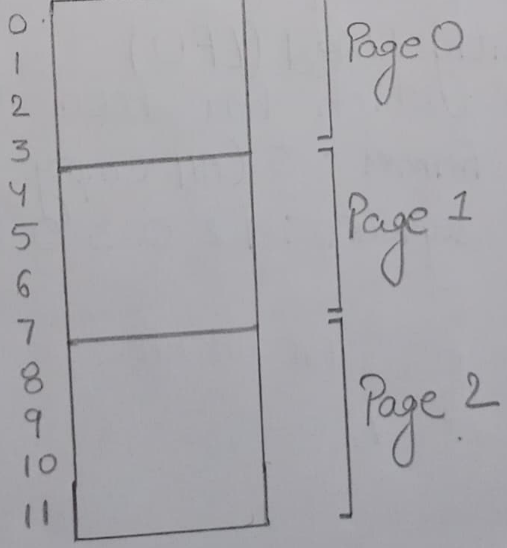
0%4 = 0

1/4 = 0.25 = 0^p

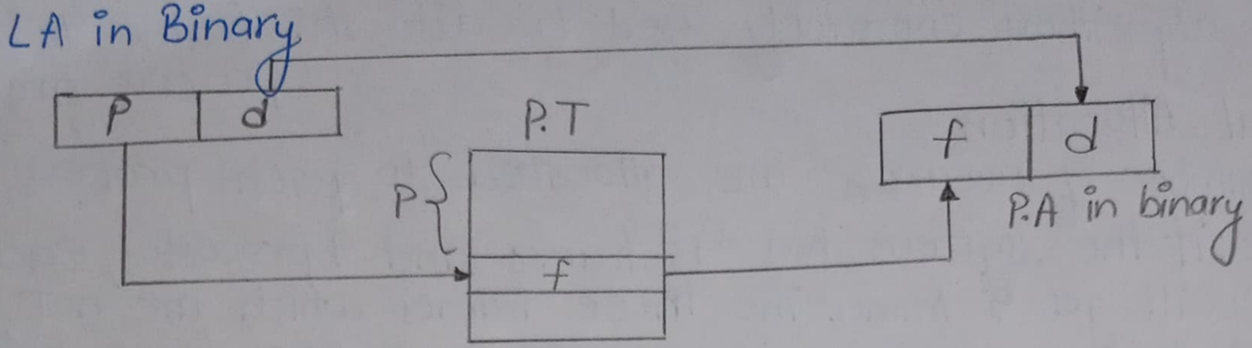
1%4 = 1^d

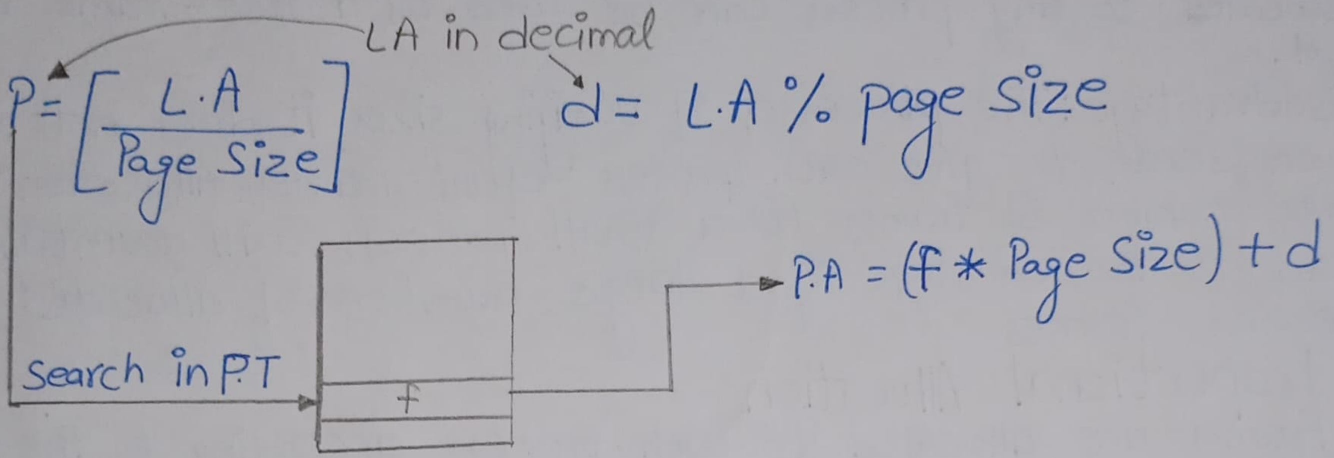
10/4 = 2^p

10%4 = 2



* L.A to PA translation in Decimal Numbers

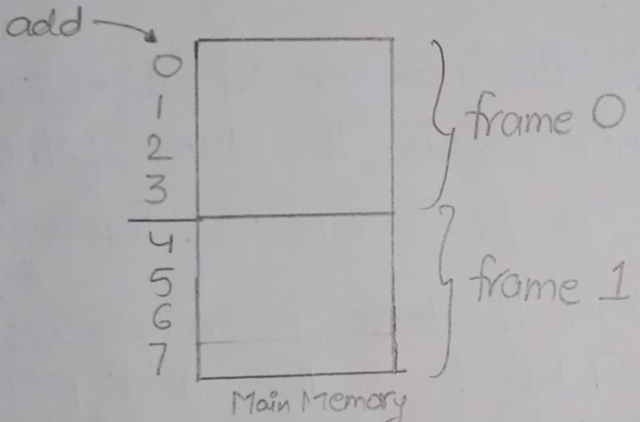




$$P = \left[\frac{L.A}{\text{Page Size}}\right]$$

$$d = L.A \% \text{ page size}$$

$$P.A = (f * \text{Page Size}) + d$$



Page Size = 4 bytes

Page Size * frame + d → we get the starting address of frame and then we add offset to it.

* Frame Allocation : How many frames to be allocated to one process.

There are various constraints to the strategies for the allocation of frames:

1. You cannot allocate more than the total number of available frames.

2. At least minimum number of frames should be allocated to each process. This constraint is supported by two reasons. The first reason is, as less number of frames are allocated, there is an increase in the page fault ratio, decreasing the performance of the execution of the process. Secondly, a sufficient number of frames available to hold all the different pages that a single instruction in a process might need to access.

* Frame allocation algorithms

The two algorithms commonly used to allocate frames to a process are:

1. Equal Allocation

Equal number of resources are allocated to each process.

Example: if the system has 48 frames and 9 processes, each process will get 5 frames. The three frames which are not allocated to any process can be used as a free-frame buffer pool.

Disadvantage: The processes of varying sizes, it does not make much sense to give each process equal frames. Allocation of a large number of frames to a small process will eventually lead to the wastage of a large number of allocated unused frames.

2. Proportional Allocation

Frames are allocated to each process according to the process size.

For a process Pi of size Si, the number of allocated frames is $a_i = \left(\frac{S_i}{S}\right) * m$, where S is the sum of the size of all the processes and m is the number of frames in the system.

Example: In system will 62 frames, if there is a process of 10 KB and another process of 127 KB, then the first process will be allocated $\left(\frac{10}{137}\right) * 62 = 4$ frames and the other process will get $(127/137) * 62 = 57$ frames.

Advantage: All the processes share the available frames according to their needs, rather than equally.

* Global vs Local Allocation

The number of frames allocated to a process can also dynamically change depending on whether you have used global replacement or local replacement for replacing pages in case of a page fault.

1. Local Replacement

When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from its own set of allocated frames only. The number of frames belonging to the process will not change. This allows processes to control their own page fault rate.

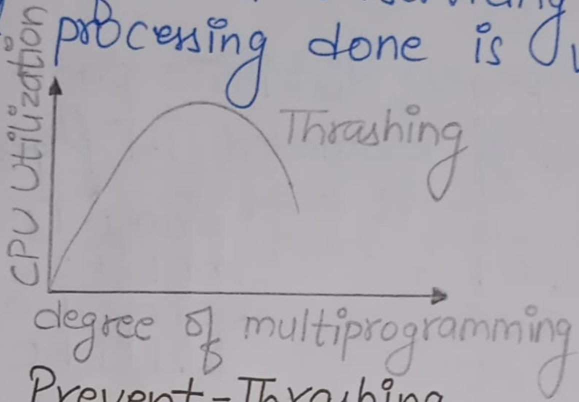
2. Global Replacement

When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from the set of all frames, even if that frame is currently allocated to some other process; that is, one process can take a frame from another. Global allocation makes for more efficient use of frames and their better throughput.

* Thrashing

when CPU spends more time on

It is a condition or a situation when the system is spending a major portion of its time servicing the page faults, but the actual processing done is very negligible.

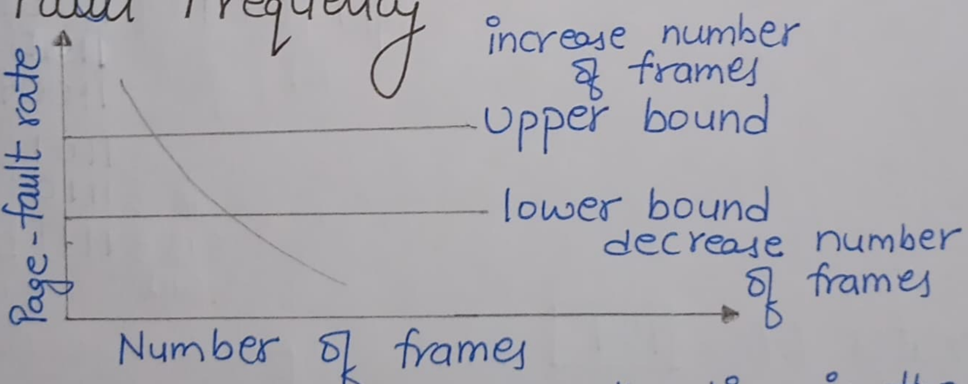


Technique to Prevent-Thrashing

1. Working-Set Model

If we allocate enough frames to a process to run process smoothly then there will not be many page faults. Hence no thrashing.

2. Page Fault Frequency



The problem associated with thrashing is the high page fault rate, and thus, the concept here is to control the page fault rate.

If the page fault rate is too high, it indicates that the process has too few frames allocated to it. On the contrary, a low page fault rate indicates that the process has too many frames.

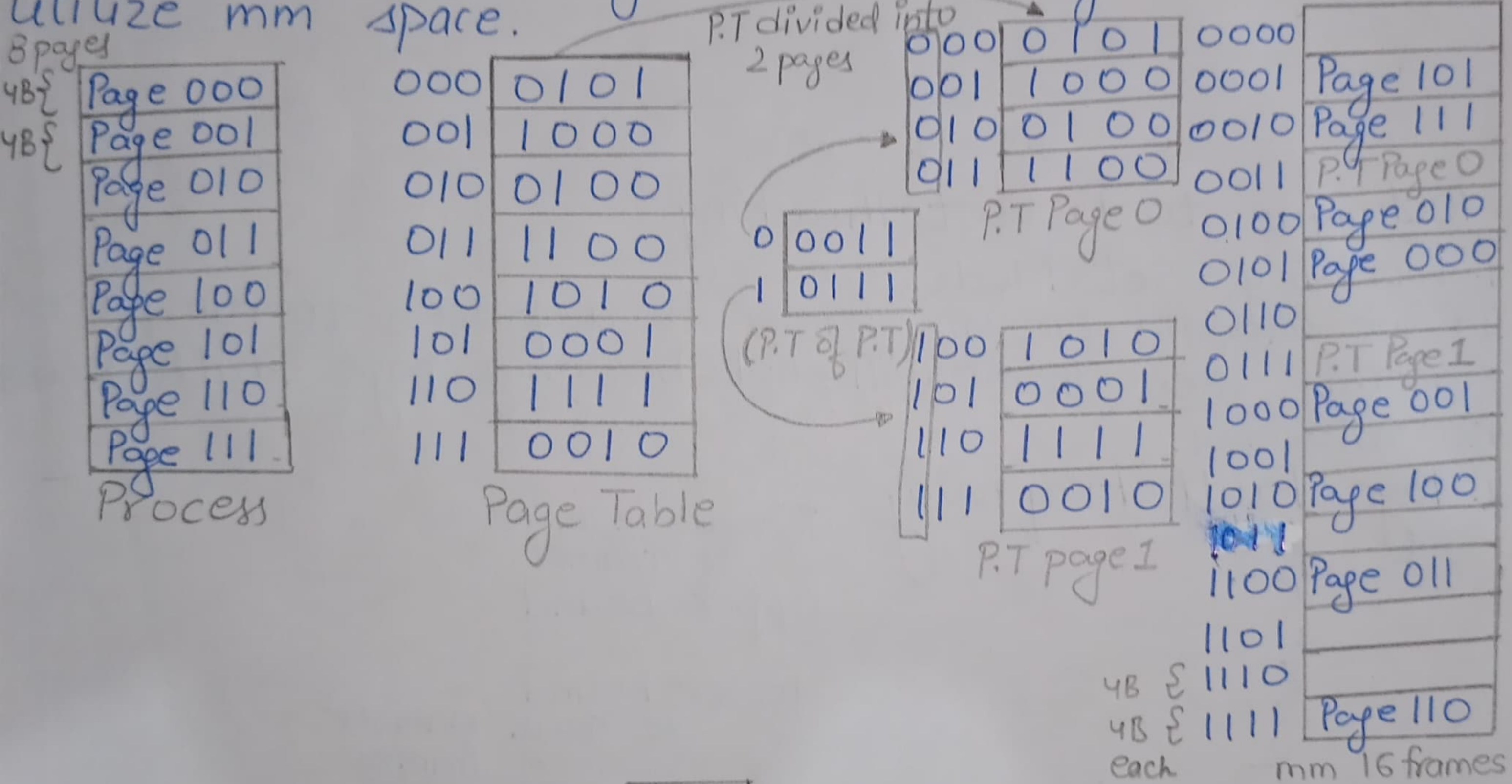
If the page fault rate falls below the lower limit, frames can be removed from the process. Similarly, if the page faults rate exceeds the upper limit, more frames can be allocated to the process.

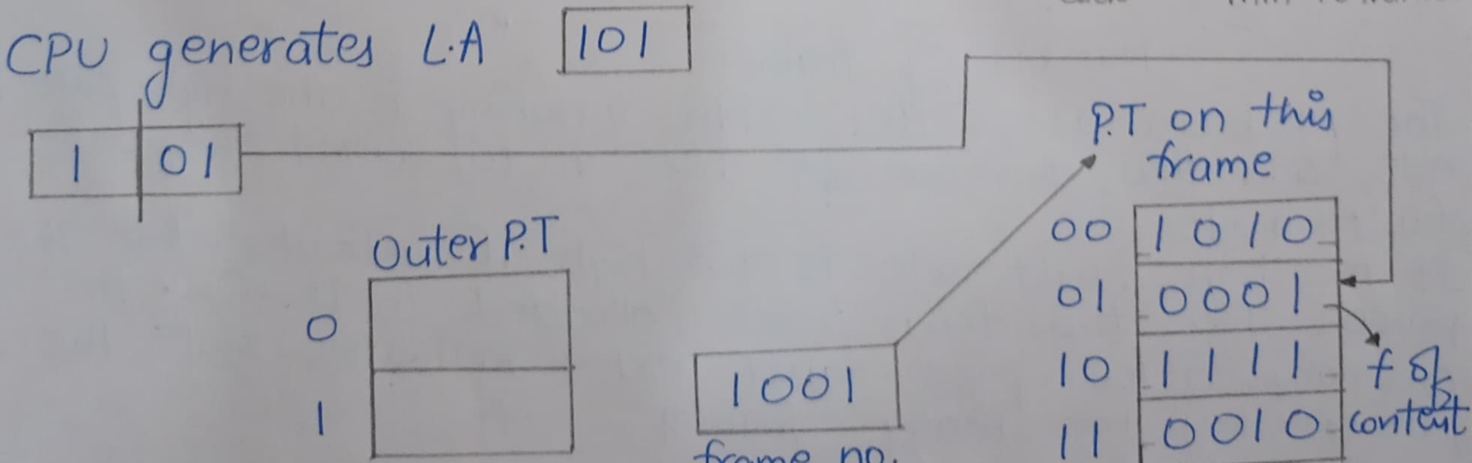
If the page fault rate is high with no free frames, some of the processes can be suspended and allocated to them can be reallocated to other processes. The suspended processes can restart later.

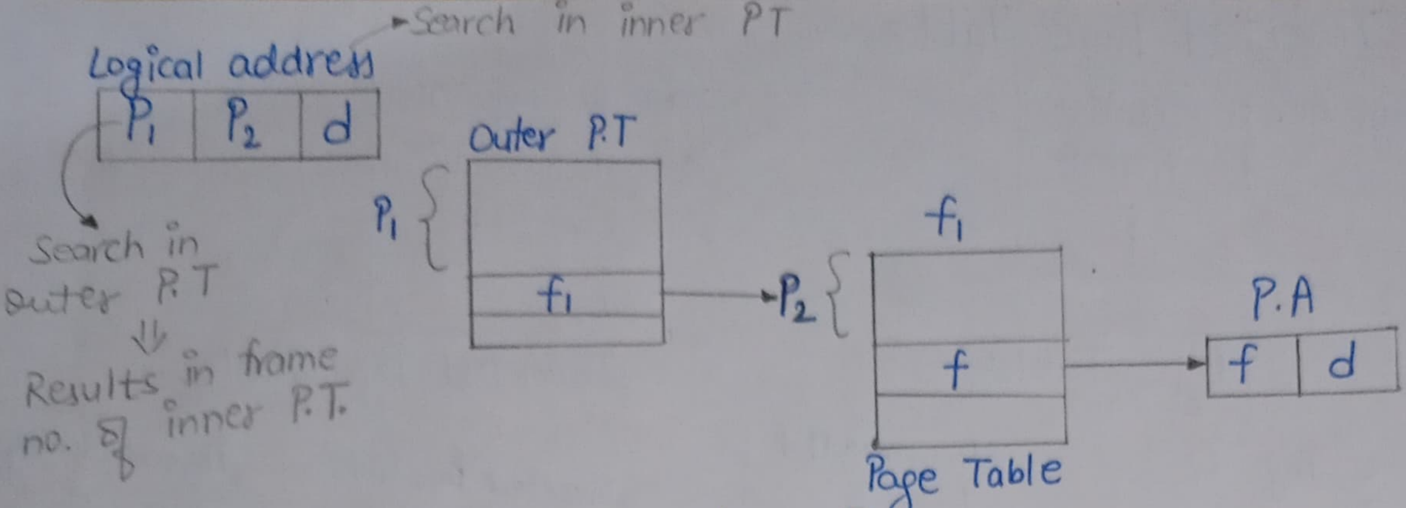
* When P.T size is very large, then to store entire P.T into mm together becomes difficult.

Sol^n ⇓

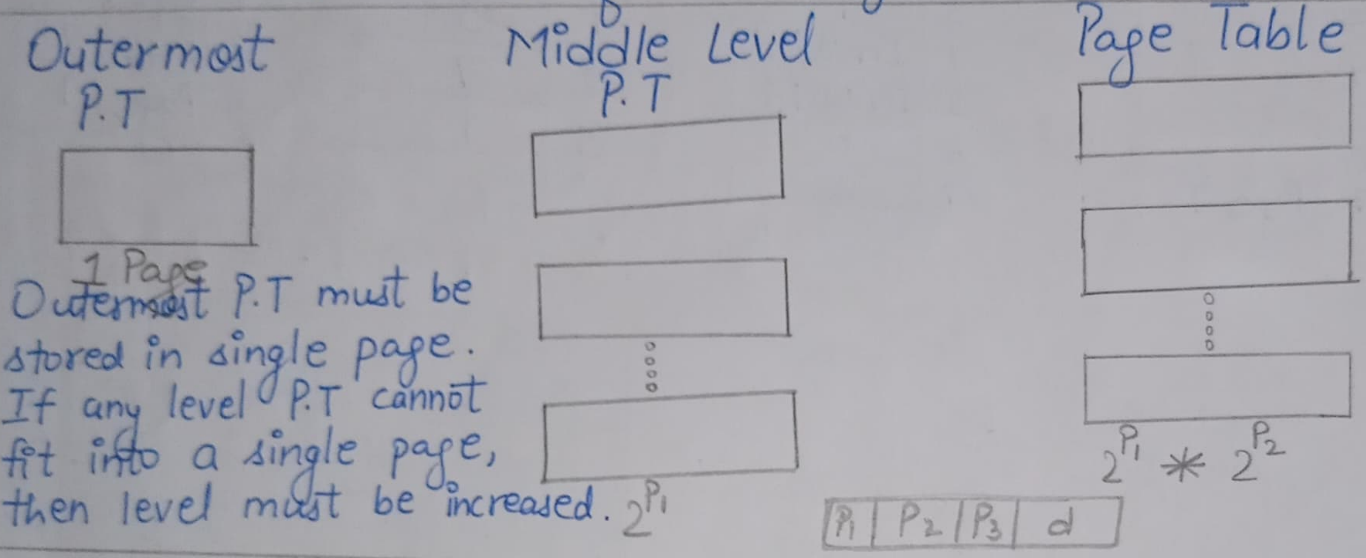
Divide P.T also into pages; distribute page table pages also on frames and keep only demanded P.T pages into mm to utilize mm space.







PTBR stores address of outer Page Table.



Outermost P.T must be stored in single page. If any level P.T cannot fit into a single page, then level must be increased.

* EMAT with Multilevel Paging

1. Single Level Paging : EMAT = $2 * t_{mm}$ (1 for P.T, 1 for Content)
2. 2-Level Paging : EMAT = $3 * t_{mm}$ (2 for P.T, 1 for Content)
3. 3-level Paging : EMAT = $4 * t_{mm}$

2-level Paging with TLB:

$$EMAT = H(t_{TLB} + t_{mm}) + (1-H)(t_{TLB} + \underbrace{3 * t_{mm}}_{\text{for P.T + Content}})$$

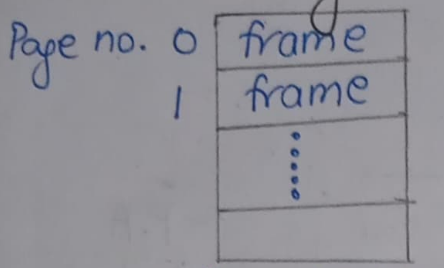
* Problem in Virtual Memory

Page Table is too large and so many page table entries are invalid.

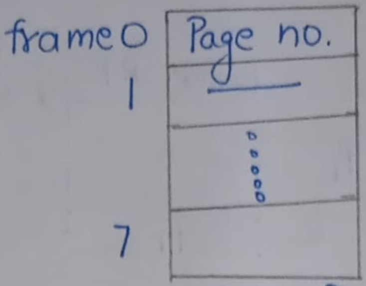
⇓

Hence a lot of space of mm wasted for keeping invalid Page Table entries.

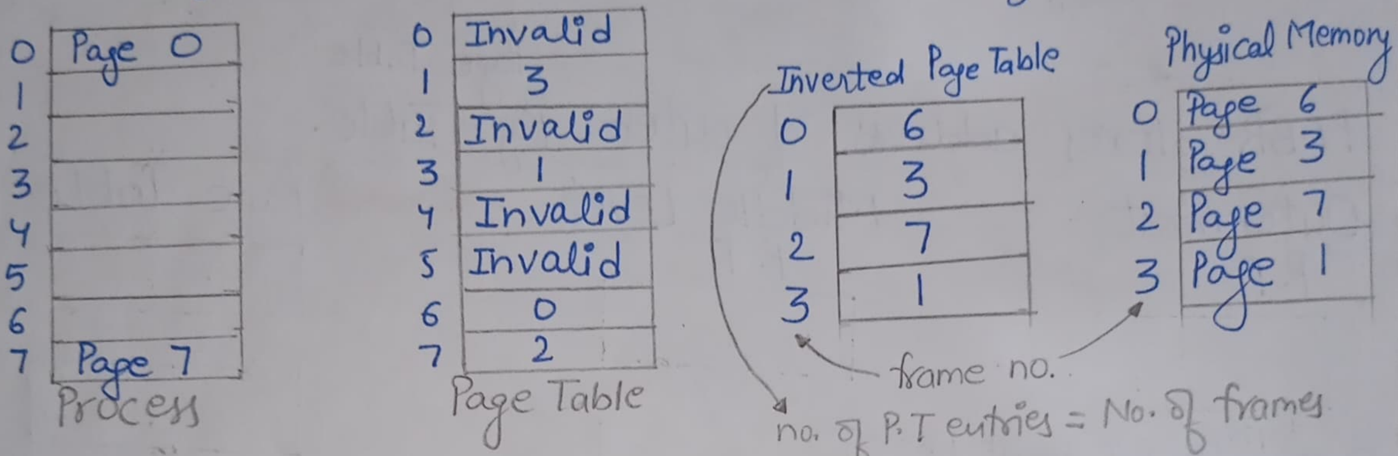
* Inverted Page Table

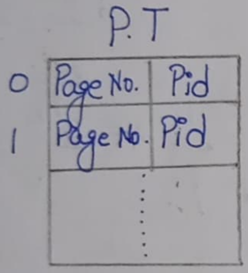


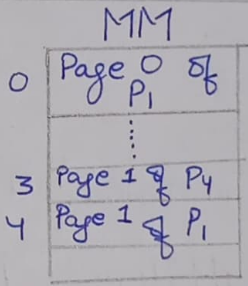
Page Table



Inverted Page Table



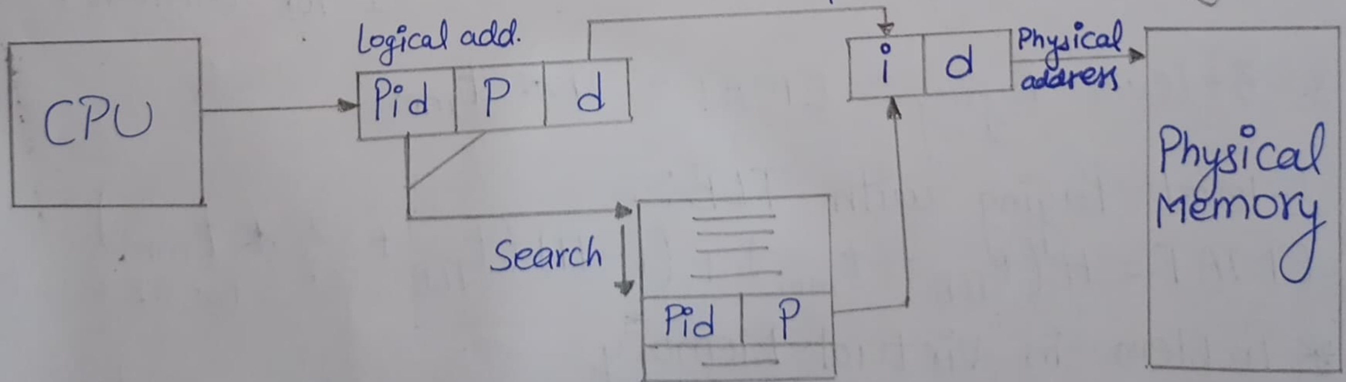




System maintain single inverted P.T for all processes.

Each entry in the page table contains the following fields:

1. Page Number 2. Process id 3. Control bits → V/I, Replacement bits, dirty bit
4. Chained pointer



Page Table

*Advantages & disadvantages

Reduced memory space.

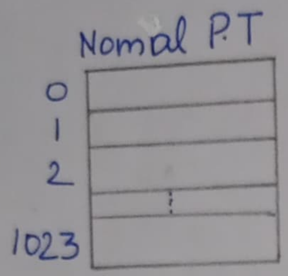
Longer Lookup time

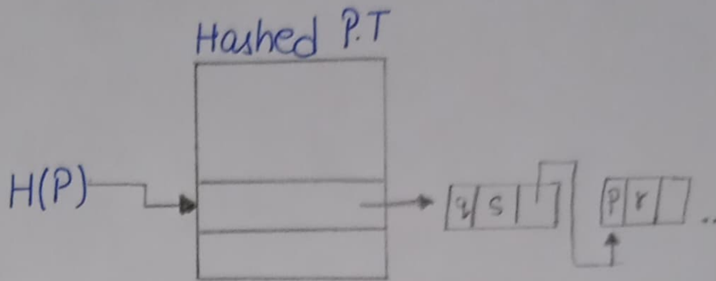
Difficult shared memory implementation.

* Note: Internal fragmentation will increase with frame size.

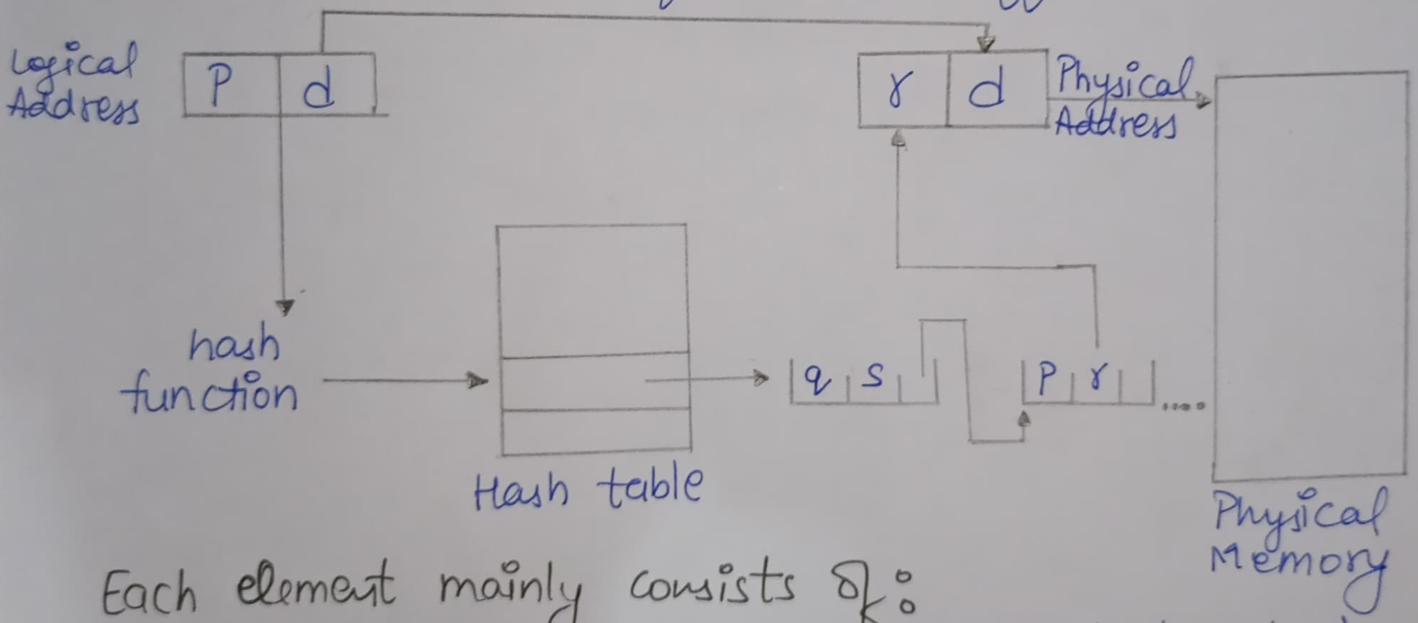
* Hashed Page Table

Hashed Page Tables are a type of data structure used by OS to efficiently manage memory mapping between virtual and physical memory addresses.





The virtual page number in the virtual address is hashed into the hash table. Each entry in the hash table has a Linked List of elements hashing to the same location (to avoid collisions as we can get the same value of a hash function for different page numbers). The hash value is the virtual page number. The virtual page number is all the bits not part of the page offset.



Each element mainly consists of:

1. The virtual page number (which is the hash value).
2. Value of the mapped page frame.
3. A pointer to the next element in the Linked List.

* Impact of Page Size on Page Table Size

When Page Size increase, Page Table Size decrease.

$$\text{Page Table Size} \propto \frac{1}{\text{Page Size}}$$ [Process Size is constant]