CSE321: Operating Systems

Topic: Main Memory

Prepared by:
Saad Bin Sohan

BRAC University

Email: sohan.academics@gmail.com
GitHub: https://github.com/saad-bin-sohan

Main Memory

how does OS handle the execution of multiple processes?

ans: The processes must be first loaded into the main memory. Then CPU executes them. Cz CPU can't access anything other than main memory (RAM). In all this, main memory needs to be managed.

- Memory unit sees either

  i) memory addresses and read their data

  or

  ii) memory addresses, and a data to write

- Register can access CPU really fast (1 or less CPU cycle. But main memory taskes many

cycles causing a 'stall'.

- Every process has some memory allocated for them which no other process can access. And it cannot access the addresses that are not allocated for it.

To ensure such protection, each process is assigned a pair of-

allocated memory range
- i) Base Register (starting one)
- ii) Limit Register (ending one)

i.e: A process gets memory allocation of 6-12 registers in physical address.

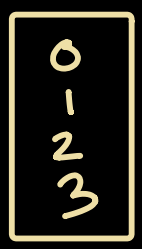
6 → Base Address (lower threshold)

12 → Limit Address (higher threshold)

Address Space

Logical Address → Address generated by the CPU (not the actual raw physical address)
(~virtual address)

eg: For P_2 → so P_2 main memory ৎ



→ logical address lets CPU map a process to the actual physical address

→ the allocation of memory must be contageous (continuous) only. no splitting

Physical Address

→ RAM ল actual physical memory

- Multiple process এর logical address (local address) same হতে পারে। But physical address must be different. Since physical addresses are different, no issue in execution

- Complie time ল and RAM ল (program) load এর time ল

Physical address = Logical Address

But once a process is loaded into the RAM

and execution time is,

Physical address = Logical Address

Memory Management Unit (MMU)

- কোনো program এর execution request আসে with logical address.

eg: execute line 2-3 on logical address but since CPU can only access RAM, the logical address must be mapped to physical address for CPU's execution.

And this mapping is done using "Memory-Management Unit (MMU)."

→ MMU is a hardware device that does the mapping.

how the mapping is done?

ans: it simply adds the base register ("relocation register") to the count value of logical address assigned for the process.

eg: A process is allocated 12 addresses (0-11) in logical address and its physical address is 23 to 34 (12টা). And LA এর 2nd line execution এর request আসলো।

so

$$\underbrace{2}_{LA} + \underbrace{23}_{BA} = \text{Mapped address}$$

// user program never sees the physical address

Dynamic Loading

- How to run a program with a size larger than the main memory's size (RAM)?

=> Using dynamic allocation.

The program is divided into many tasks called "Routine". And a routine does not execute until it is called. It executes only when it's called.

Dynamic Linking

- Static linking:

  Every program has 2 parts:

  i) header, where all the libraries are imported

  ii) body part with logical reasoning

Static linking → the entire program is compiled and executed.

Program $\xrightarrow{\text{compile}}$ executable file → execution

But dynamic linking 2-

when a 'Routine's execution is requested it only imports the library that specific routine requires. And it does so using a tiny code called 'stub'.

how 'Stub' works:

→ when RAM is allocated, firstly the stub code is placed there and executes. After stub's execution it replaces itself with the 'Routine.'

Contiguous Allocation

Main memory is divided into 2 parts:

i) RAM এর lowest most bit গুলোতে OS থাকে only

ii) Highest most address গুলোতে user program থাকে।

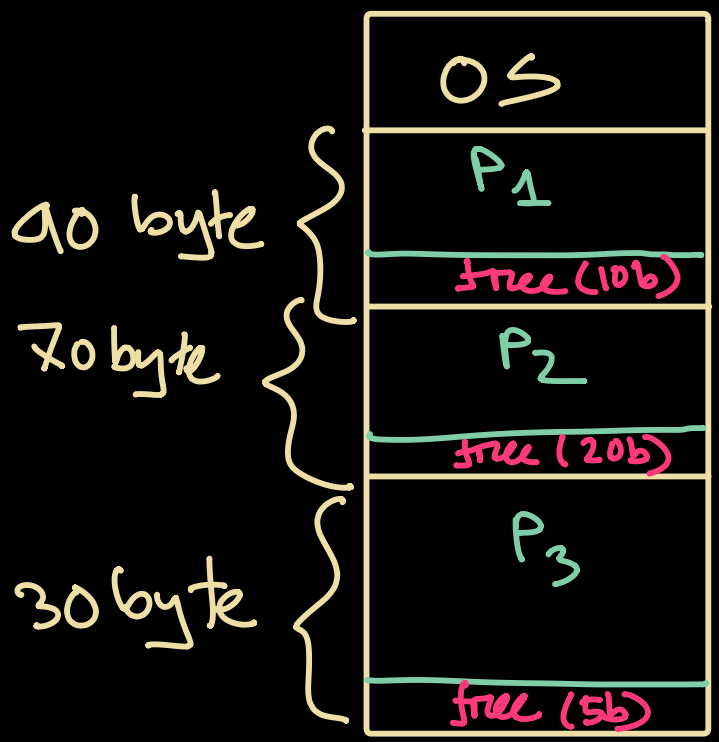
And all the process must be allocated continuous memory with no splits in between.

- Kernel code এর changing size এর জন্য allocated memory increase হয়ে allowed.

Multiple-partition Allocation

→ a mechanism for allocating RAM to multiple processes

Fixed size partition:

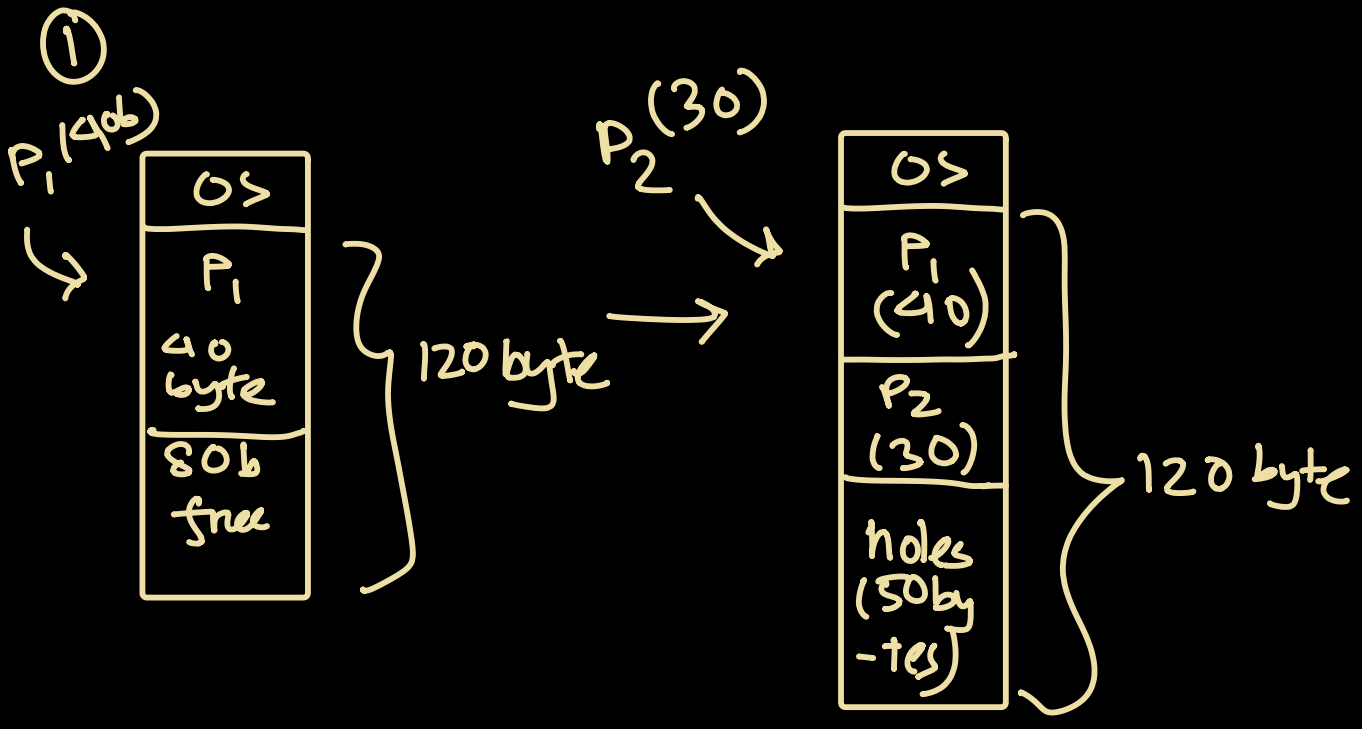


processes allowed at a time in fixed-size-partition = # partition

∴ P_4 can't be allocated until free

Variable-partition:

→ partitions are created as per the memory allocation a process needs



- দুইটা hole এর মাঝে as long as there is a process, they won't merge.
- continuous holes merge.

Dynamic-Memory Allocation

→ instead of setting aside a fixed amount of memory before the program runs,

dynamic-memory allocation is like getting memory for a program while it's running only when it needs the memory and releasing the memory when done.

3 methods of dynamic-storage allocation

i) First-fit

ii) Best-fit

iii) worst fit

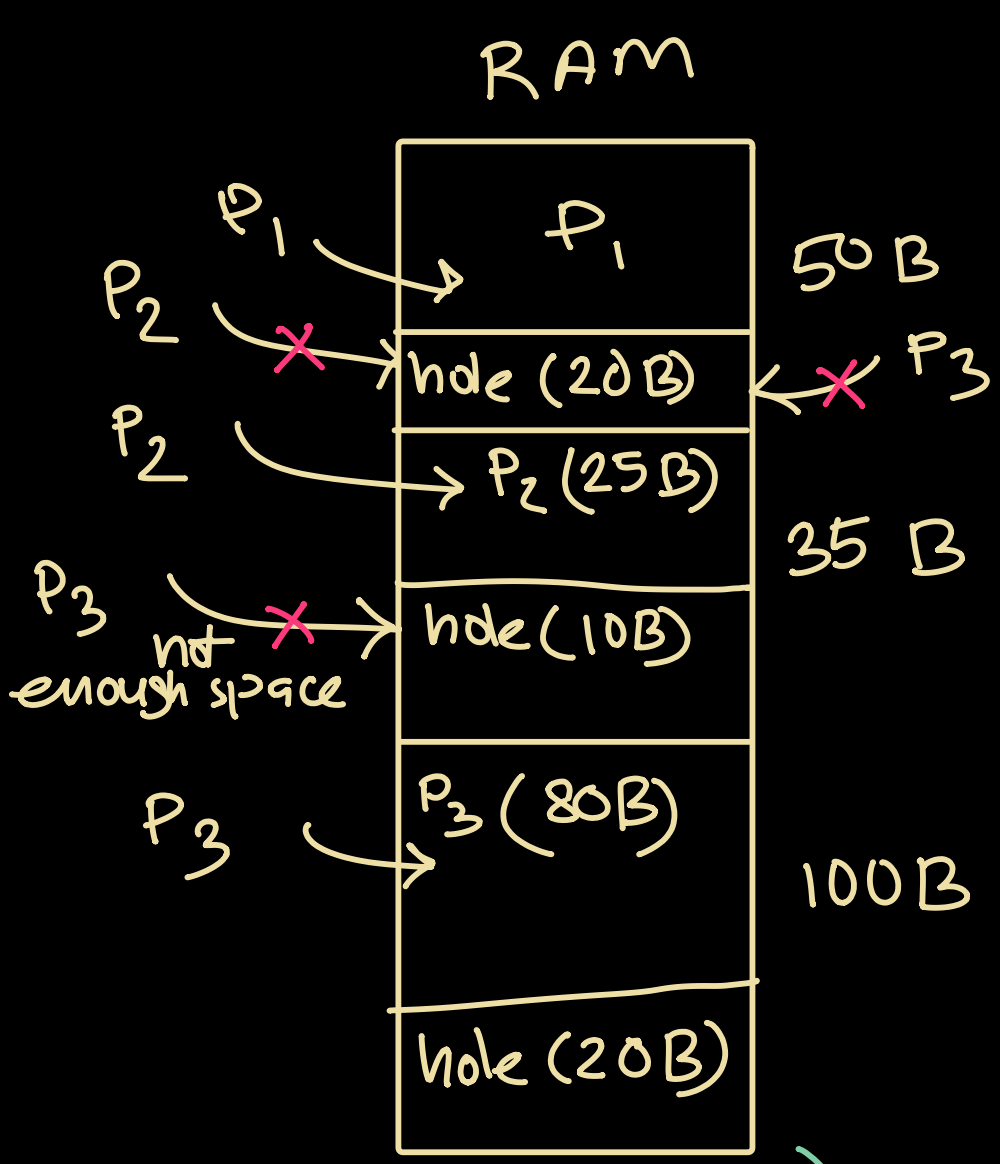
First-fit allocation:

→ allocate the first hole that is big enough

eg: P_1 (30 B)

P_2 (25 B)

P_3 (80 B)



now P_4 (40 B)

total free space 50B. still can't allocate P_4 (2 not enough contiguous memory of 40B.

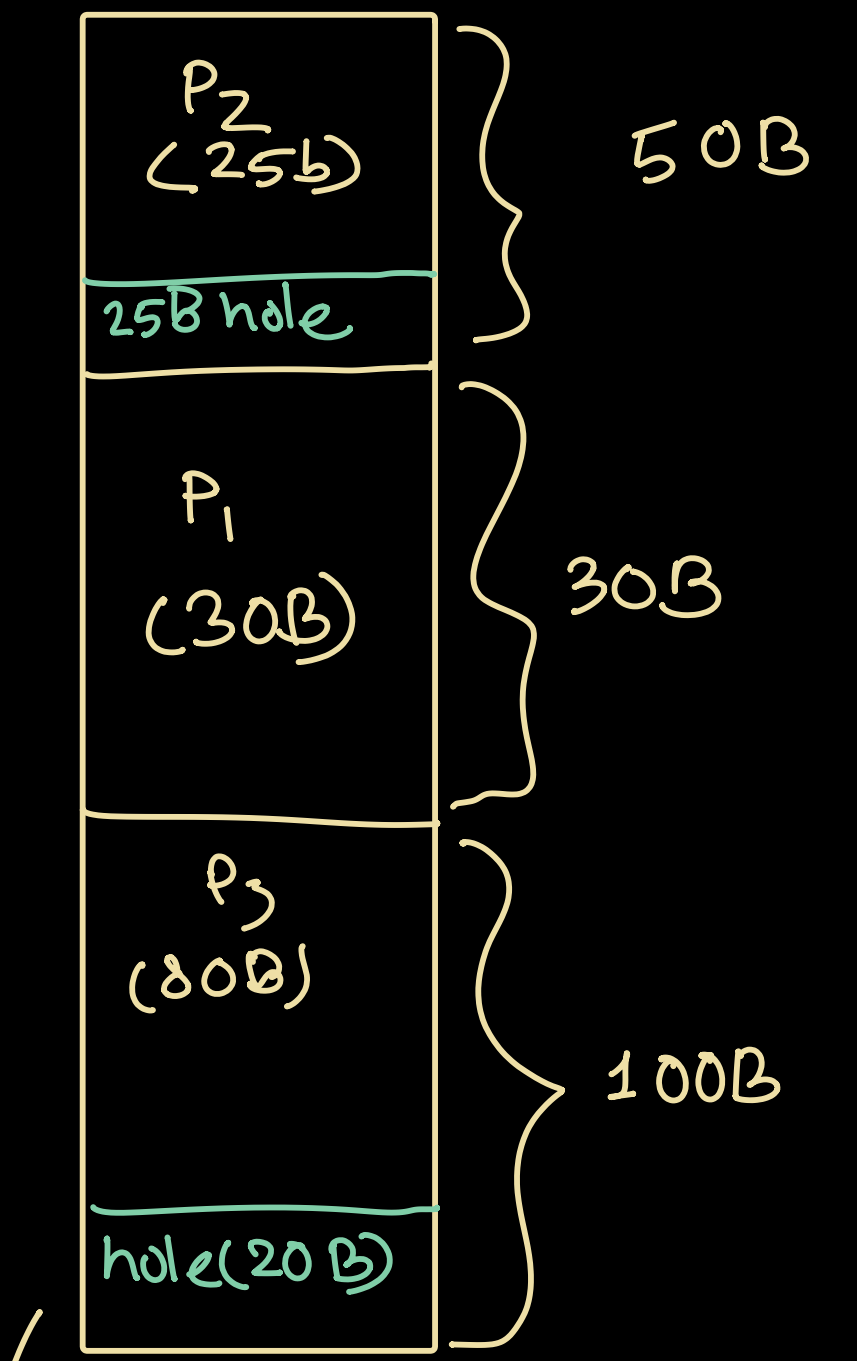
Best-fit algorithm:

→ allocate the smallest hole that is big enough to hold the program, must search entire list (unless the memories are already ordered by size)

i.e: P_1 (30B)

P_2 (25B)

P_3 (80B)



total hole = 45 byte

holes are scattered

Worst fit Algorithm:

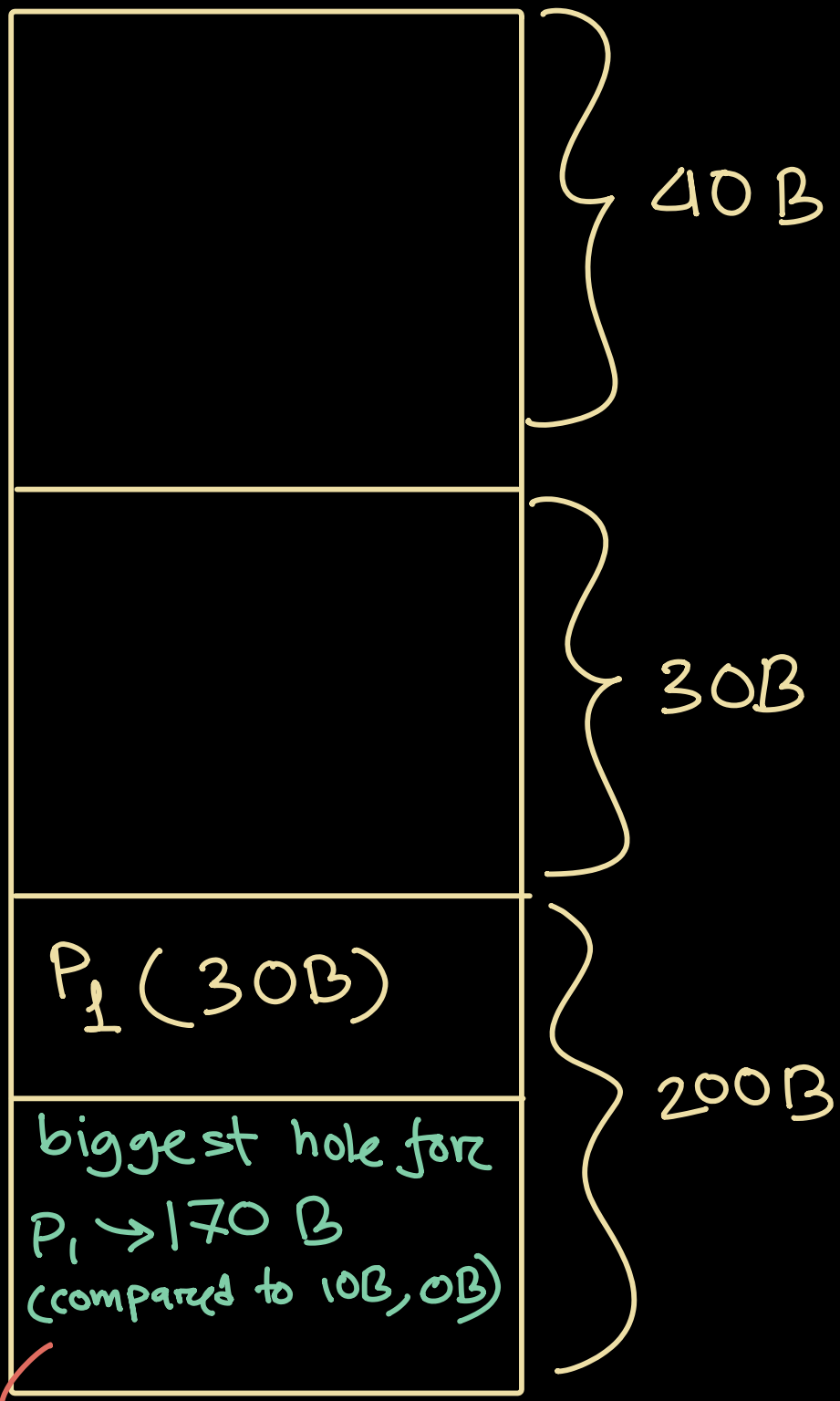
→ Allocate the largest hole to a program in the entire remaining free memory. (must search the entire list)

→ most efficient algorithm

eg: P_1 (30 B)

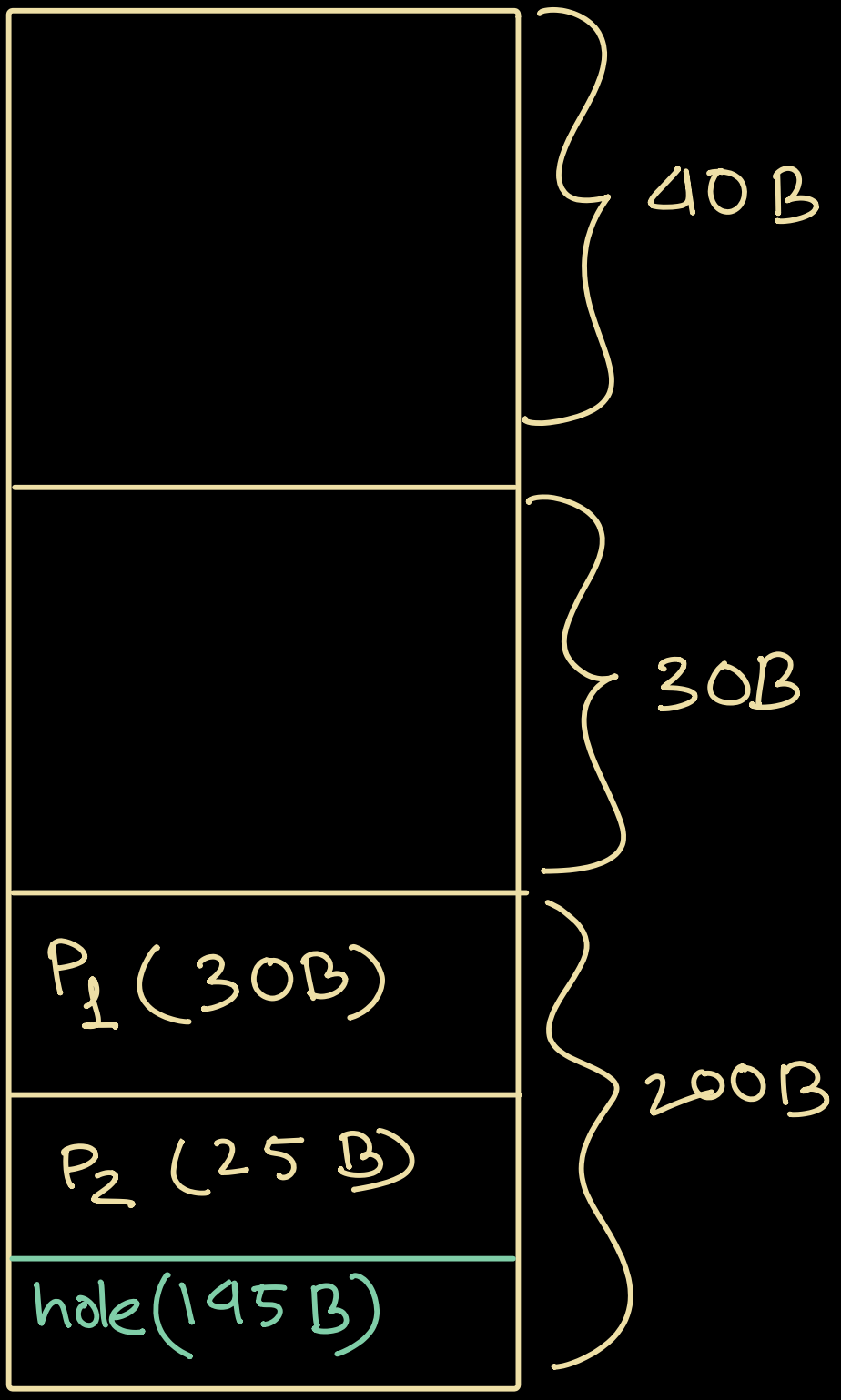
P_2 (25 B)

P_3 (80 B)



→ largest hole for P_2 is the hole previously caused due to P_1

so now,



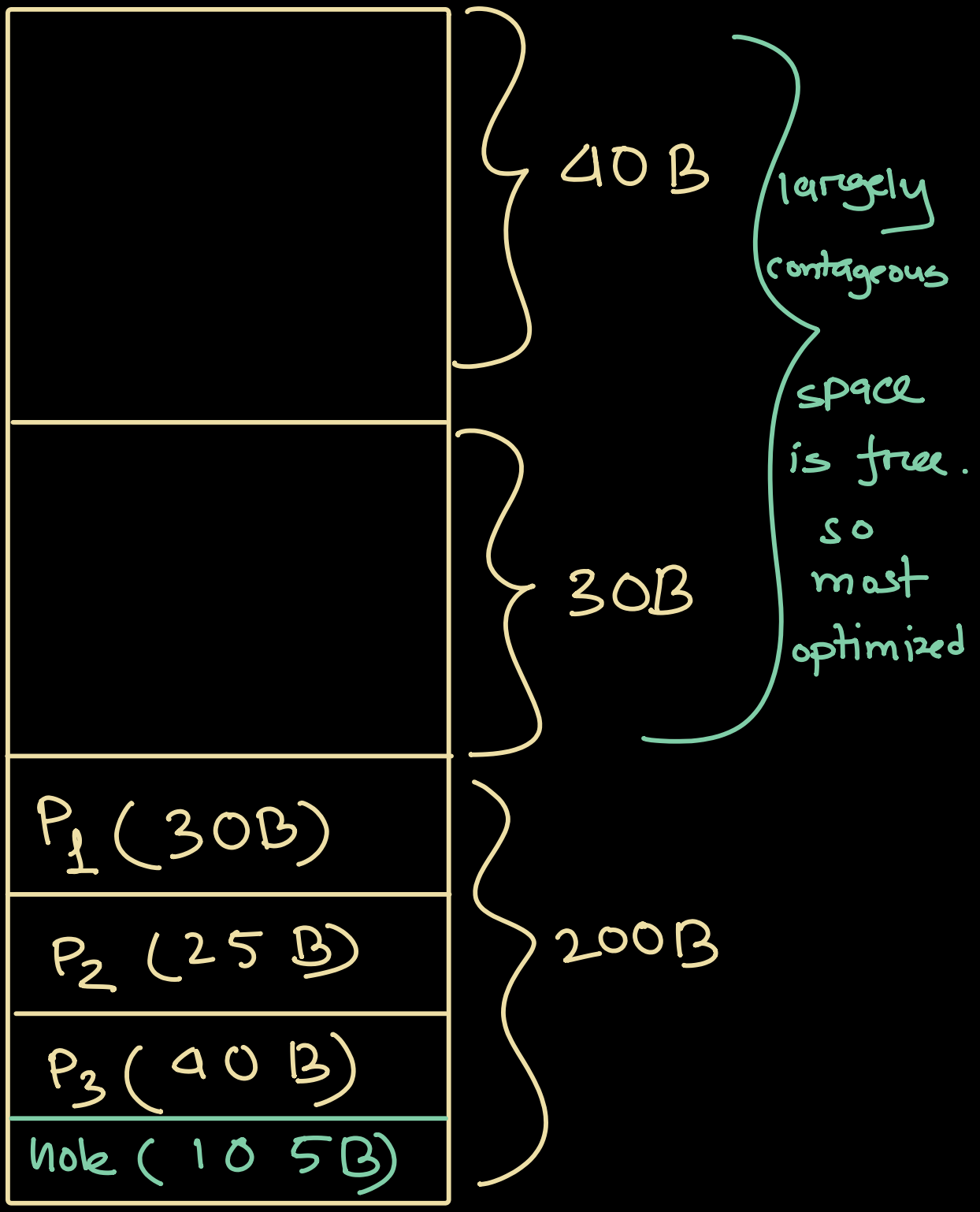
next for process 3,

hole

3 free spaces: 40B, 30B, 195B

biggest hole would be if P_3 placed at the hole

so now


40 B
30B
largely contageous space is free. so most optimized
P1 (30B)
P2 (25 B)
P3 (40 B)
Hole (10 5B)
200B

Fragmentation

→ the remaining scattered holes

→ remaining free space contigeous না হলে larger program কে place করা যাবে না

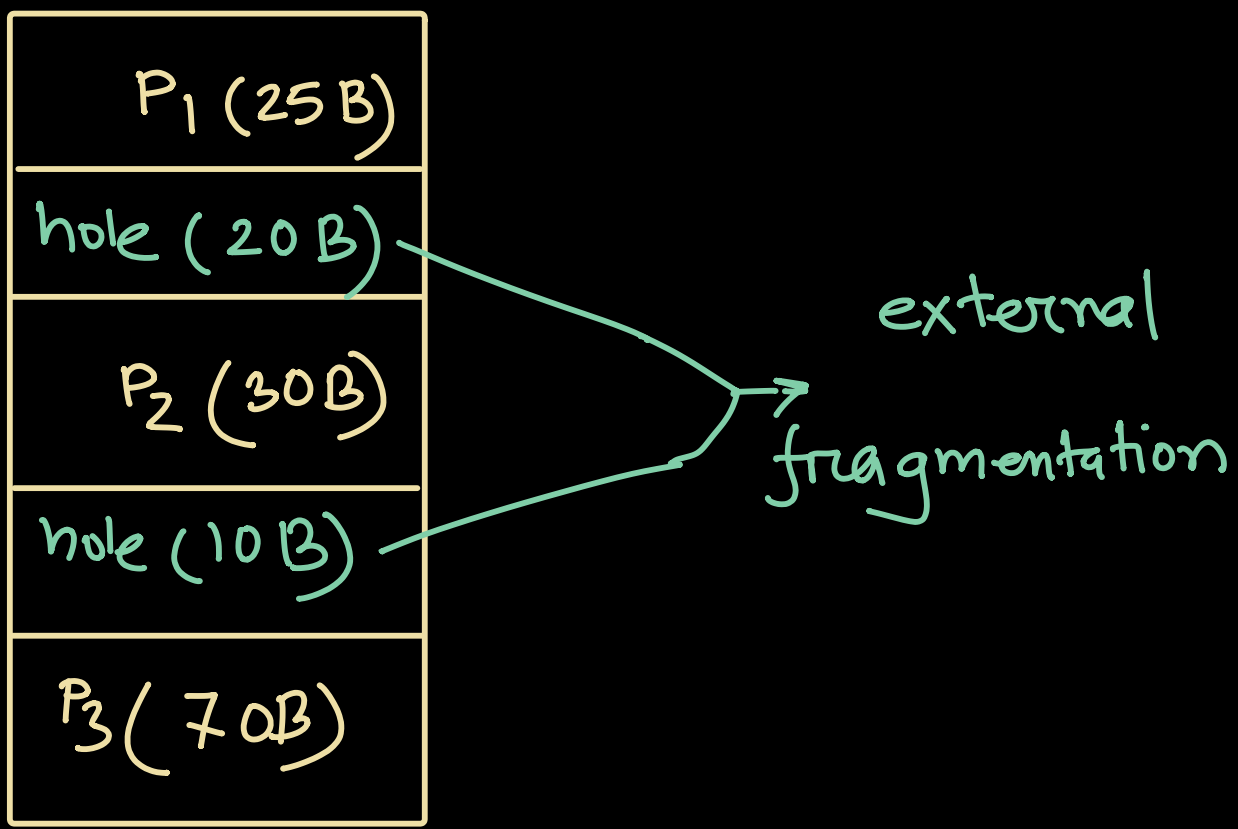
2 types:

i) External Fragmentation

ii) Internal Fragmentation

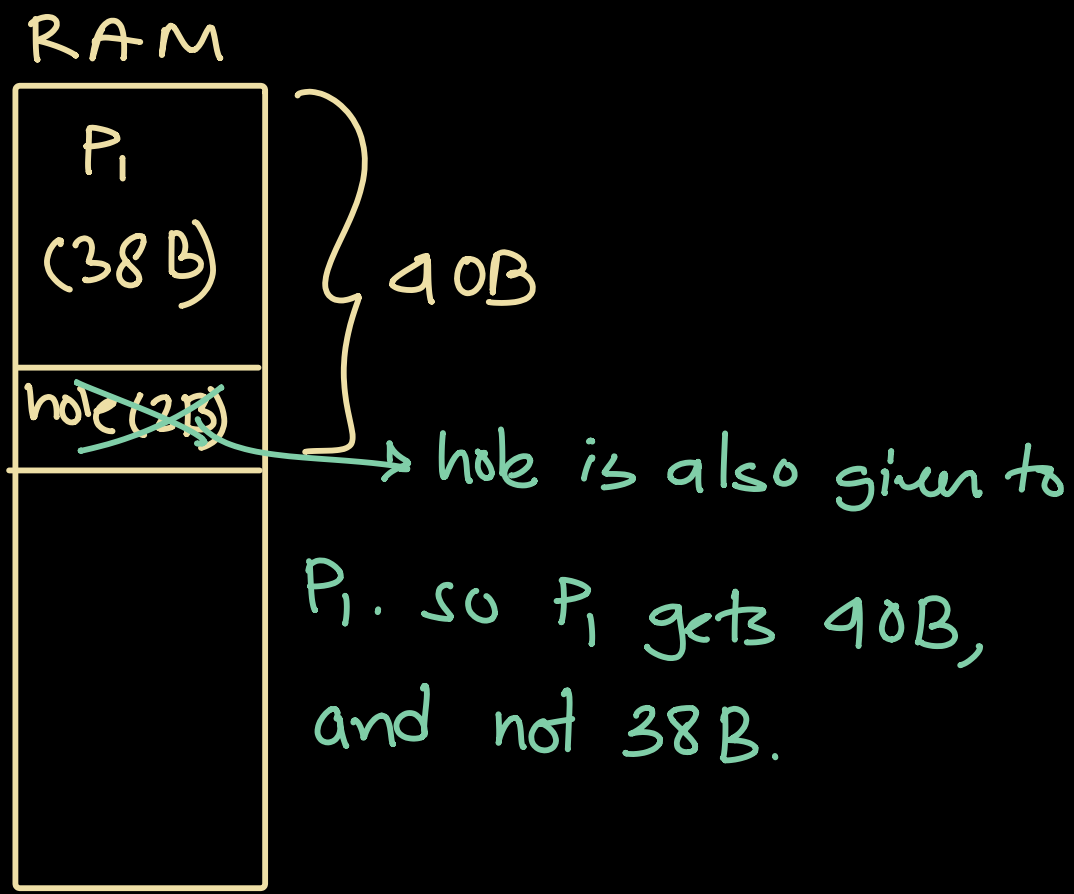
External Fragmentation

→ enough total free memory available but they are scattered



Internal Fragmentation

→ a program is located more memory than it actually needs (cz the hole is too small to deal with the overhead caused by the hole)



→ no solution for Internal Fragmentation

→ to solve External Fragmentation:

i) Compaction: All process's memory is brought together. All holes are brought together.

ii) Paging

Paging

→ main memory and process কে same size memory chunk এ divide করা

→ avoids external Fragmentation and varying size of memory chunks

Paging Method:

step- i) divide physical memory into fixed size-blocks (called frames) in a way যে-

যেটা memory allocate হবে and

each block এর size 2 এর power

(2^n) হওয়ার (btn 512 bytes and 16 Mbytes)

ii) keep track of all free pages

ie: 12 byte process size, 20 byte RAM

size হবে,

RAM → 4 byte এর 5 টা chunks

process → 4 byte এর 3 টা chunks

☆ RAM and process এর each chunk

এর size must be same

→ Process এর chunks are called "pages".

→ so to run program, ram must have

3 free frame

iii) set up a "page table" that translates logical address to physical address

page table:

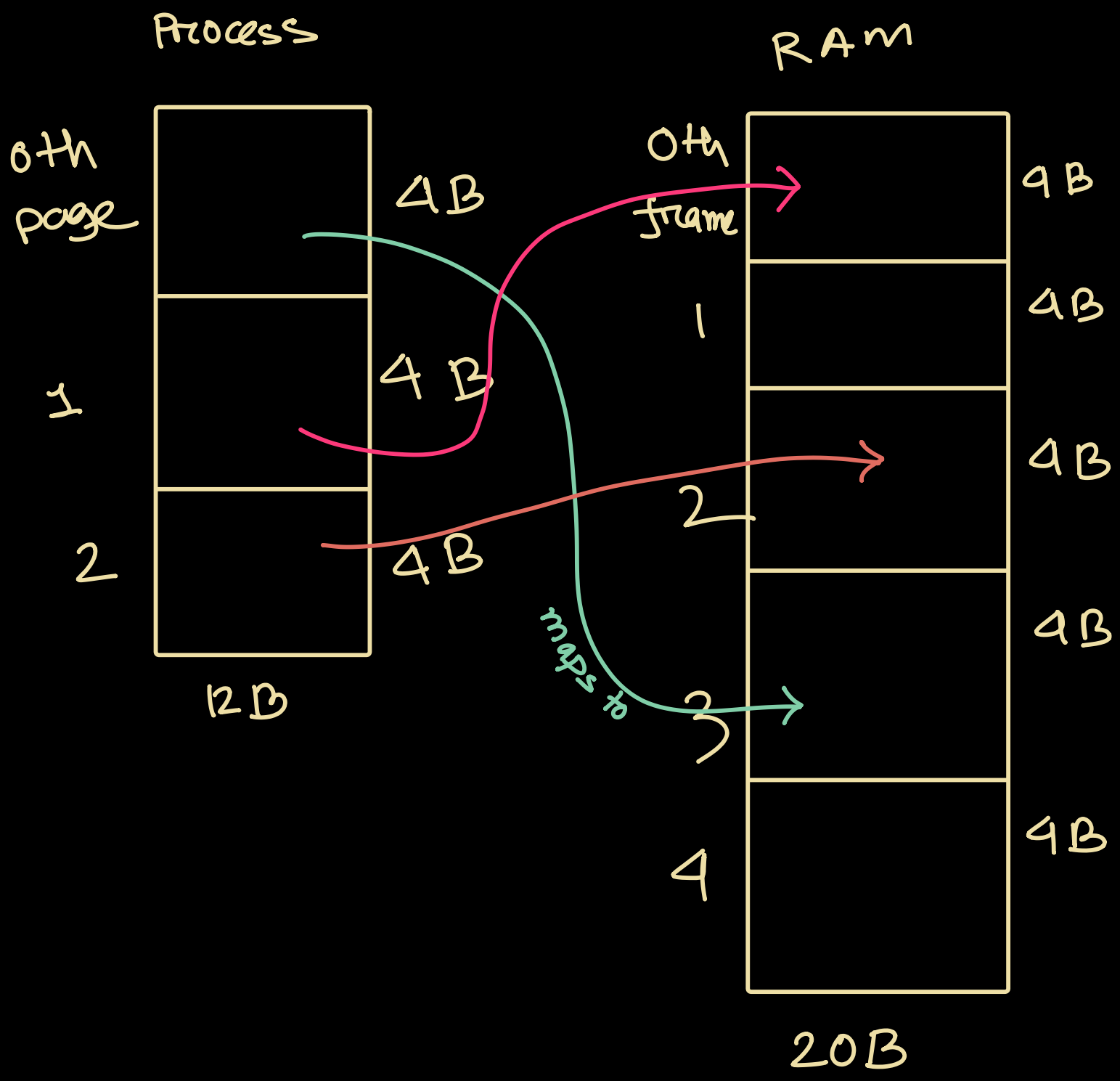
p	f
0	3
1	0
2	2

p → page number

f → frame number

free frame → 1, 9

i.e:


Process
RAM
0th page
1
2
4B
4B
4B
12B
0th frame
1
2
3
4
4B
4B
4B
4B
4B
20B
maps to

Address Translation Scheme

→ frame numbers, page number exist in binary.

→ the method of converting a logical address to physical address

→ প্রতি page এ logical address থাকে

Logical address has 2 parts:

i) Page number (কোন page এর সাথে connected, সেই page এর base address) → p

→ index to a page table

ii) Page offset (d)

→ physical memory তে কতটুকু জায়গা নিচ্ছে

→ page number এর সাথে combine করে physical memory address কে define করা হয়

i.e: 16 byte এর memory

- Logical address space 2^m byte এর হয়। so, $16 = 2^m$

  $\Rightarrow m = 4$

- Page size 2^n byte এর হয়।

i.e: page size 4 byte এর হলে,

$4 = 2^n$

$\Rightarrow n = 2$

16 byte কে represent করতে binary তে

a bit প্রয়োজন। — — — —

Page number, p	page offset, d
m-n	n

so, logical address space 16, page size 4 byte:

let's say we want to find the page number and page offset of a specific logical address (eg: 0110).

left side এর m-n bits represent করে page number. Remaining part is page offset.

prev example এ m = 4, n = 2.

m - n = 2

0 1 | 1 0

(m-n) bits = 2 → decimal এ 1. page number = 1.

0 1 | 1 0

→ decimal ல 2.

∴ page offset = 2

1st page ன் 2nd address ஐ 0110.

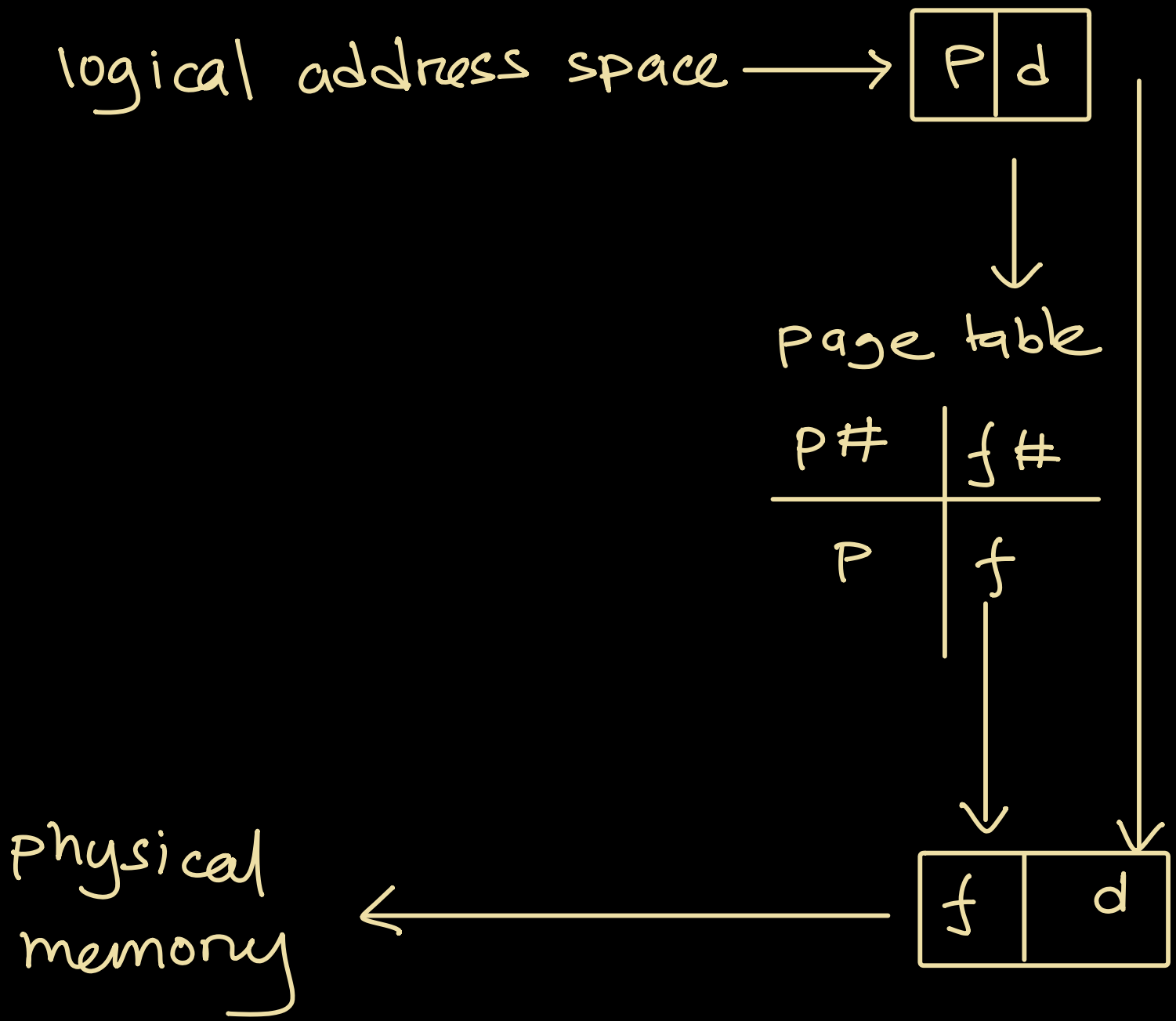
i.e: 10 11

m = 4, n = 2, m - n = 2

10 | 11

page number = 2 page offset = 3

Paging Hardware



Paging Model of logical and Physical Memory:

32/4 = 8 frames

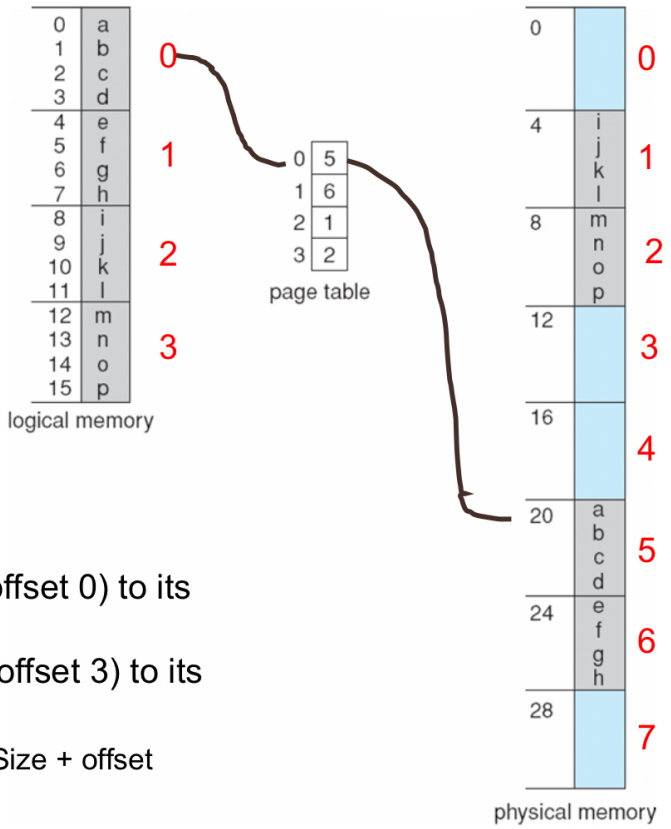
Paging Example

m is used to determine the number of logical addresses, 2m = 16

n indicates the offset within each logical address

n=2 and m=4

Physical memory → 32-byte memory and 4-byte pages



- Map logical address 0 (page 0, offset 0) to its corresponding physical address
- Map Logical address 3 (page 0, offset 3) to its corresponding physical address

frameNumber * frameSize + offset

Size of page:

→ Internal fragmentation measure করা হয় using 'size of a page' and size of a process'.

Size of Page

Calculating internal fragmentation

Page size = 2,048 bytes

Process size = 72,766 bytes

35 pages + 1,086 bytes

Internal fragmentation of 2,048 - 1,086 = 962 bytes

Worst case fragmentation = 1 frame – 1 byte

On average fragmentation = 1 / 2 frame size

So small frame sizes desirable?

But each page table entry takes memory to track

Page sizes growing over time

Solaris supports two page sizes – 8 KB and 4 MB

Process view and physical memory now very different

By implementation process can only access its own memory

* At all times, a system keeps track of the free frames it has. Before, after allocation. Always

Page Table Implementation

→ Page table is kept only in main memory

→ 2 registers are used to maintain the page table:

i) Page-table base register (PTBR)
points to the page table

ii) Page-table length register (PTLR)
stores the size of page table

In Paging approach, each instruction requires two memory accesses –

i) one for reading the page table which tells us data কোন frame এর read/write করা হবে

ii) another for accessing that data itself.

☐ The two memory access inefficiency is solved using 2 types of registers

i) associative memory

or

11) translation look aside buffer

Associative Memory

→ its a register that keeps a copy of the page table and uses the copy for all works

→ since accessing register is much faster than RAM, issue solved.

→ incase the copy doesnt have mention of a process that the original page table has, that process's info is simply duplicated in the copy

Paging Hardware with TLB

→ Just like 'associative memory', this approach also stores a copy of page table

→ TLB is a cache register

→ after generating page number from logical address, TLB replaces the page number with the frame number using its copy table and combines it with page offset (TLB hit)

→ memory was accessed once after lookup

→ TLB এর copy table এ info না পাওয়া গেলে ("TLB Hit") main memory তে একবার access করে original page table থেকে এ info access করে।

Effective Access time

- Associative lookup = ε unit time

  (can be < 10% of memory access time)

- Hit ratio, α = percentage of times that a page number is found in associative register.

math:

$\alpha = 80\%$, $\varepsilon = 20$ ns for TLB search,
100ns for memory access

EAT = ?

solve:

$$EAT = \alpha * (20 + 100) + (1 - \alpha) * (2 * 100)$$

$$= .8\,(120) + (.2)\,200 \quad = 96 + 40$$

$$= 136 \text{ ns}$$

Q) $\alpha = 99\%$, $\epsilon = 20$ ns for TLB search

$\epsilon = 100$ ns for memory access

ans:

$$EAT = \alpha * (20 + 120) + (1-\alpha) * (2*100)$$

$$= 0.99 (120) + (1 - 0.99) * (2 \times 100)$$

$$= 120.8$$

Maths

Logical address ସ୍ତ corresponding

physical address = (frame number * page size) + page offset

Maths

In a system there are 2 processes- P1 (16 bytes), P2 (8 bytes) with page size of 4 bytes. Size of the main memory is 32 bytes. Page tables of processes are given below.

PMT of p1

Page #	Frame #
0	3
1	7
2	10
3	5

PMT of p2

Page #	Frame #
0	4
1	0

Find corresponding physical addresses of the following logical addresses:

a. Address 10111 of p1

b. Address 00101 of p1

c. Address 11011 of p2

d. Address 01010 of p2

e. Address 1011 of p1

f. Address 1111 of p2

Logical address space = $2^m = 32$

$m = 5$

Page size, $2^n = 4$

$n = 2$

$m - n = 5 - 2 = 3$

a) 10 | 11

5

5 page number ২য় process, P_1 ২য়

page table ২ নাই so invalid address

b) 00 | 01

1 1

∴ physical address = (7 × 9) + 1

= 29

c) 11 0 | 11

6

invalid

d) 01010

2

invalid

e) 01011

2 3

physical address = (10 × 9) + 3

= 93

f) 01111

3

invalid

memory address 2 এর power এ না থাকলে,

i.e: 36 bytes

step: nearest lower 2 এর power বের করে m calculate করে add one.

$36 \Rightarrow 32 = 2^5$

$m = 5 + 1 = 6$

Assume that, page size = 8 bytes and Physical Memory = 64 bytes. If the CPU generates logical addresses 16, 6, 77, 21, 14, 3, 19 and 5 respectively then how can the users' view of memory be mapped into physical memory?

Logical Memory

Page #	Data
P0	l
P1	p
P2	c
P3	s
P4	t
P5	c

Page Table

Page #	Frame #
P0	4
P1	2
P2	10
P3	7
P4	9
P5	0

Main memory

solve:

page size: 2^n = 8 bytes

$n = 3$

physical memory, 2^m = 64 bytes

$\Rightarrow m = 6$

$$m - n = 6 - 3$$

$$= 3$$

pages or frames required,

$$= \frac{\text{RAM size}}{\text{page size}} = \frac{64}{8} = 8$$

so main memory gets divided into 8 frames

now lets find out physical address for each logical address.

logical address 16:

$$(16)_{10} = (10000)_2$$

m = 6 so 6 bit এর address হবে হবে।

m - n = 3

∴ physical address = (10 * 8) + 0

= 80

=> invalid address কেন main memory তে 63 পর্যন্ত আছে।

6:

$(6)_{10} = (110)_2$

000 | 110

0 | 6

∴ physical address = $(4 \times 8) + 6$

$= 38$

77:

$(77)_{10} = \underbrace{(1001101)_2}$

7 bit দিয়ে m=6, so not possible

21:

$(21)_{10} = (1\ 0\ 1\ 0\ 1)_2$

0 1 0	1 0 1
2	5

phy. add = $(10 * 8) + 5$

$= 85$

invalid

14:

$(14)_{10} = (1\ 1\ 1\ 0)_2$

0 0 1	1 1 0
1	6

phy. addr = $(2 \times 8) + 6$

$= 22$

3:

$(3)_{10} = (11)_2$

0 0 0 0 | 1 1

0 | 3

Ph. adr = $(4 \times 8) + 3$

$= 35$

19 :

$(19)_{10} = (1\ 0\ 0\ 1\ 1)_2$

0 10 | 0 1 1

2 | 3

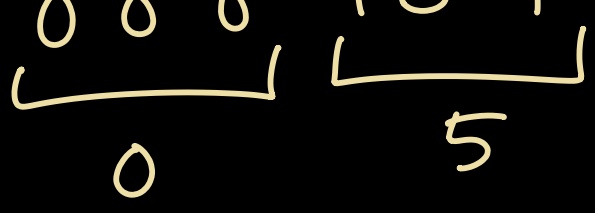
phy adr = $(10 \times 8) + 3$

$= 83$

invalid

5:

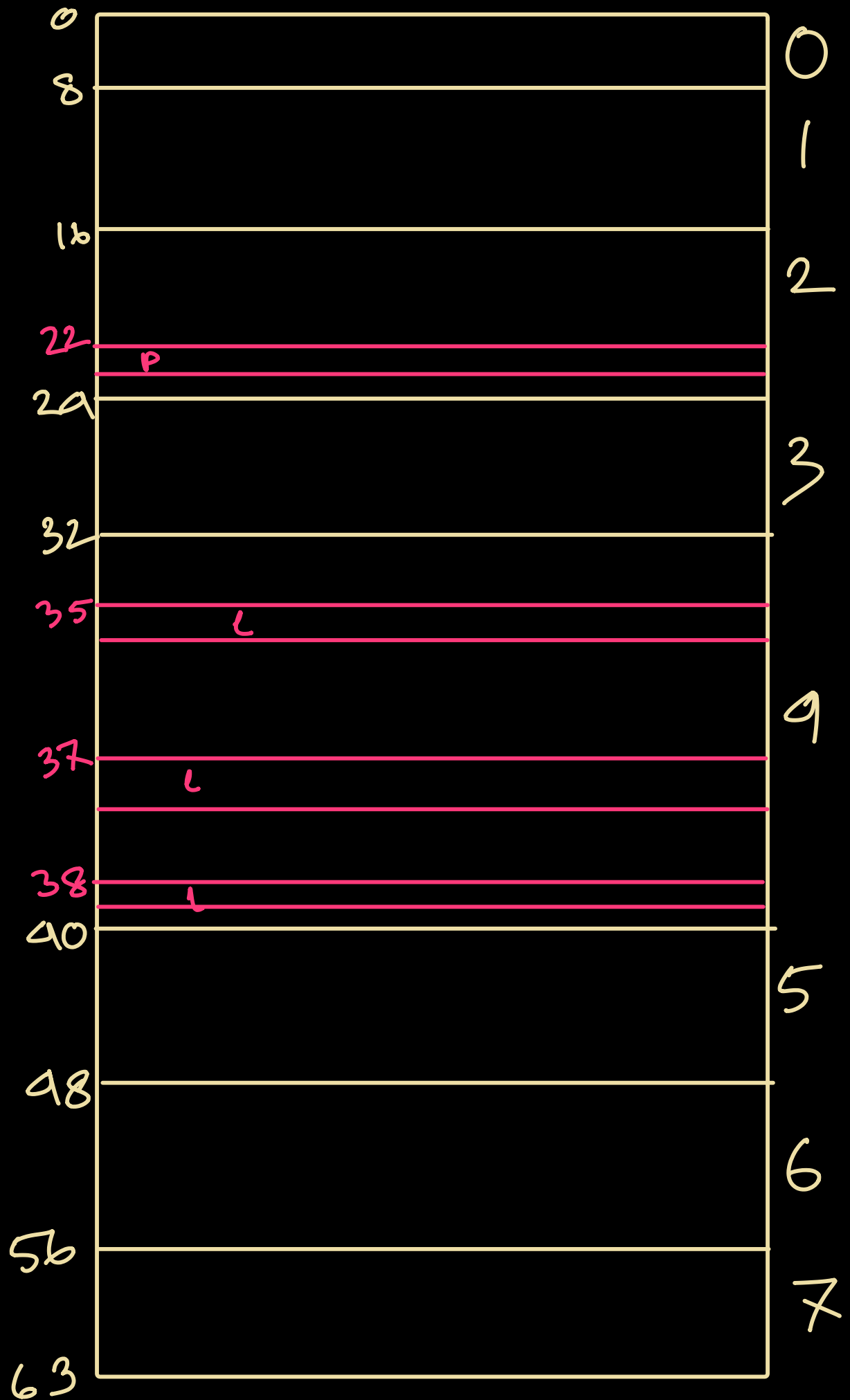
$(5)_{10} = (\quad 101)_2$



phy adr = $(4 \times 8) + 5$

$= 37$

user's view of main memory:


0
8
16
22
P
29
32
35
L
37
L
38
L
40
48
56
63
0
1
2
3
9
5
6
7

3 types of math

i) Effective Access Time

ii) Multiple process এর logical address থাকবে। page size, main memory size থাকবে। physical address বের করা।

iii) Process এর data, page table (+ page size, memory size) থাকবে। user's view of main memory বের করা

iv) internal fragmentation (4?)