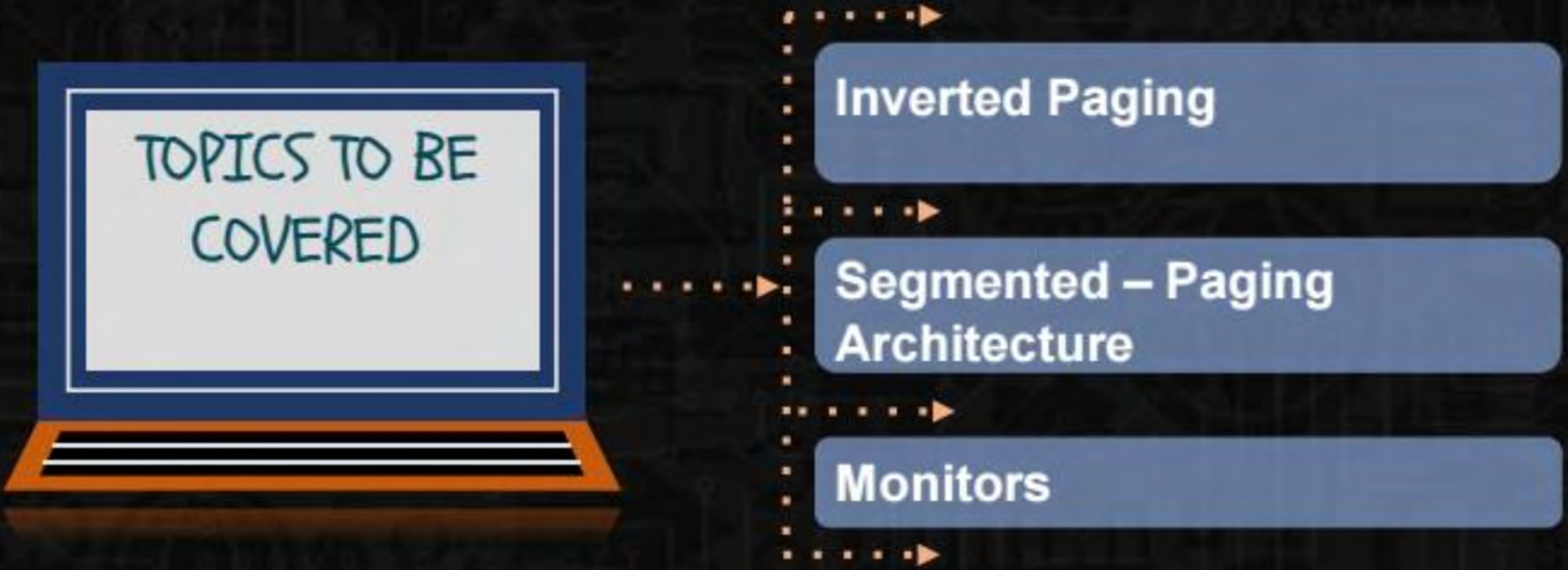
CS & IT ENGINEERING

Operating Systems

System Calls and Threads

Lecture No. 02

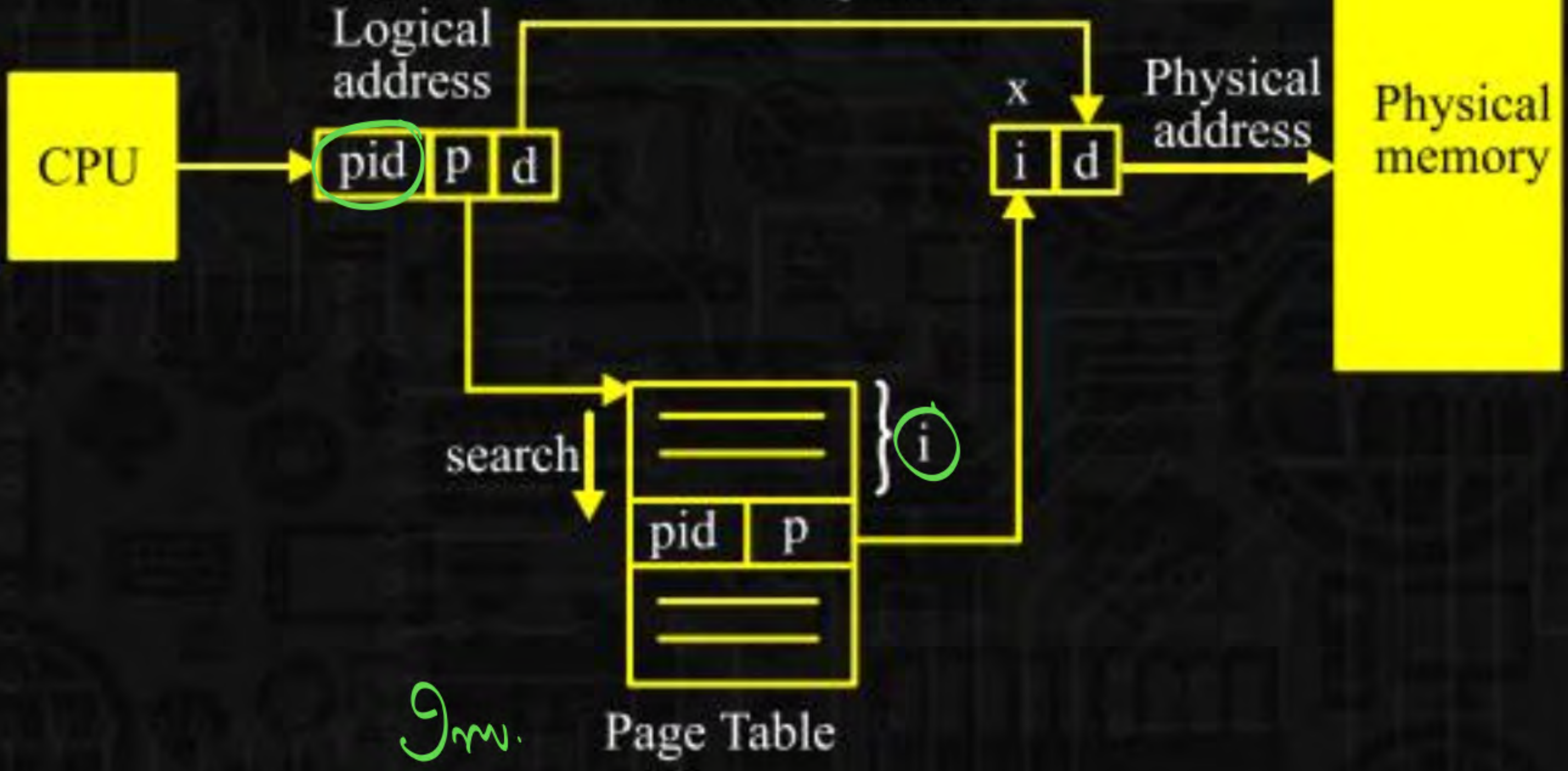
By- Dr. Khaleel Khan Sir



TOPICS TO BE COVERED

- Inverted Paging
- Segmented – Paging Architecture
- Monitors

Inverted Page Table



Imp.

Q. Which one of the following statements is FALSE?

PYQ

A. The TLB performs an associative search in parallel on all its valid entries using page number of incoming virtual address. T

B. If the virtual address of a word given by CPU has a TLB hit, but the subsequent search for the word results in a cache miss, then the word will always be present in the main memory. T

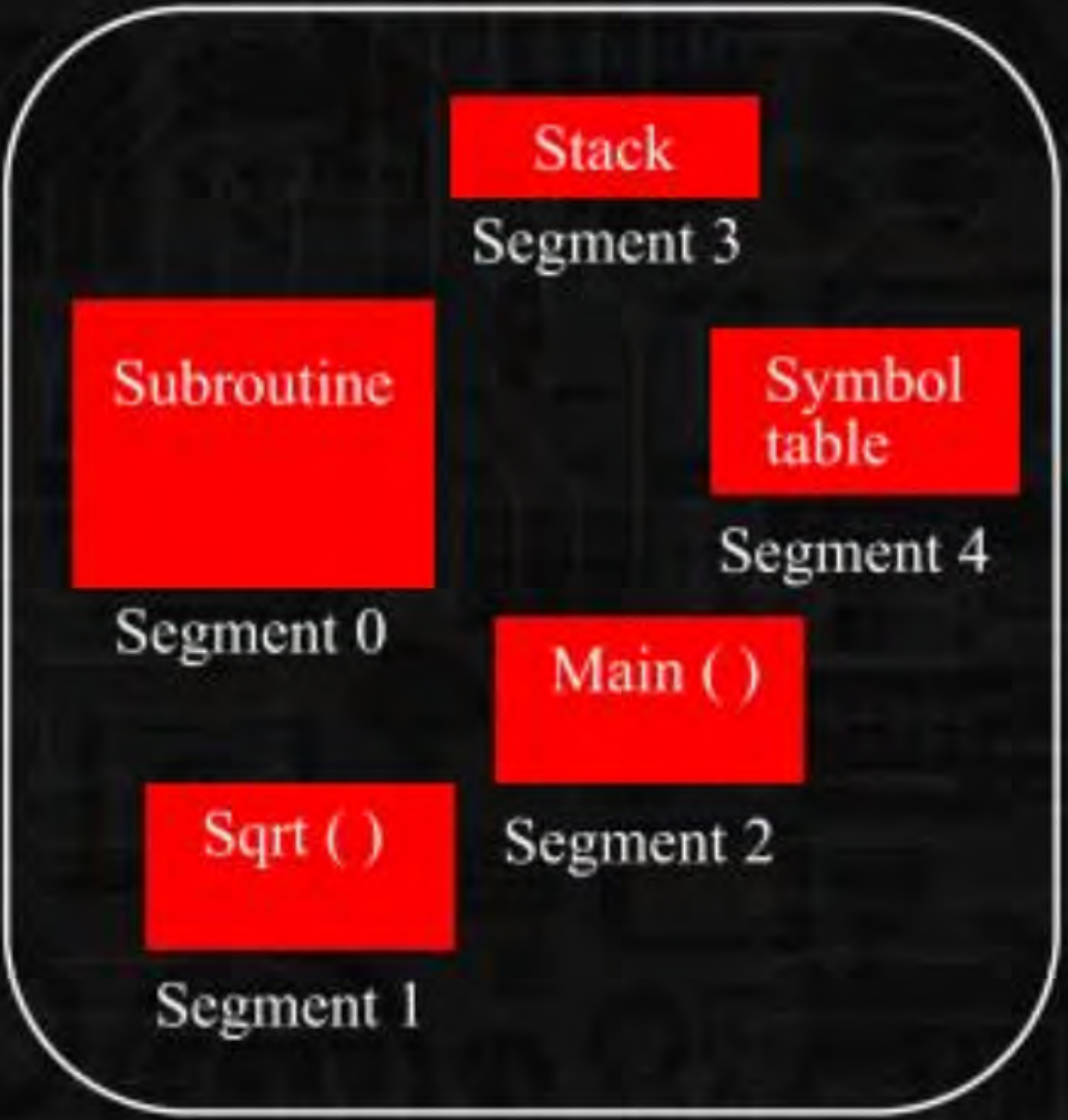
[✓ C.] The memory access time using a given inverted page table is always same for all incoming virtual addresses. F

D. In a system that uses hashed page tables, if two distinct virtual addresses V1 and V2 map to the same value while hashing, then the memory access time of these addresses will not be the same. T

Segmented Paging Architecture

→ overcome the Problem of External frag. (PAS)

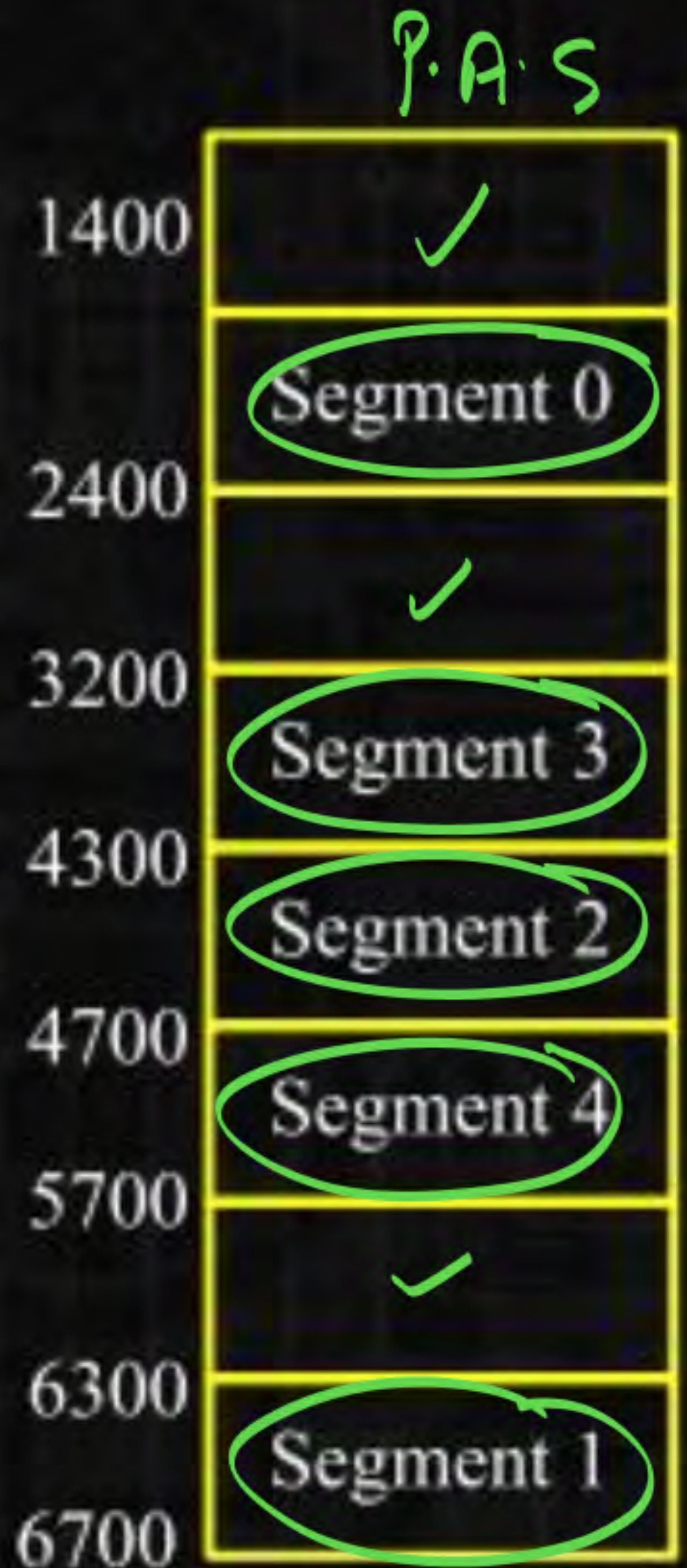
→ Assoc. a Smaller Transl. Table with Process, by applying paging on Seg-Table;



Example of Segmentation

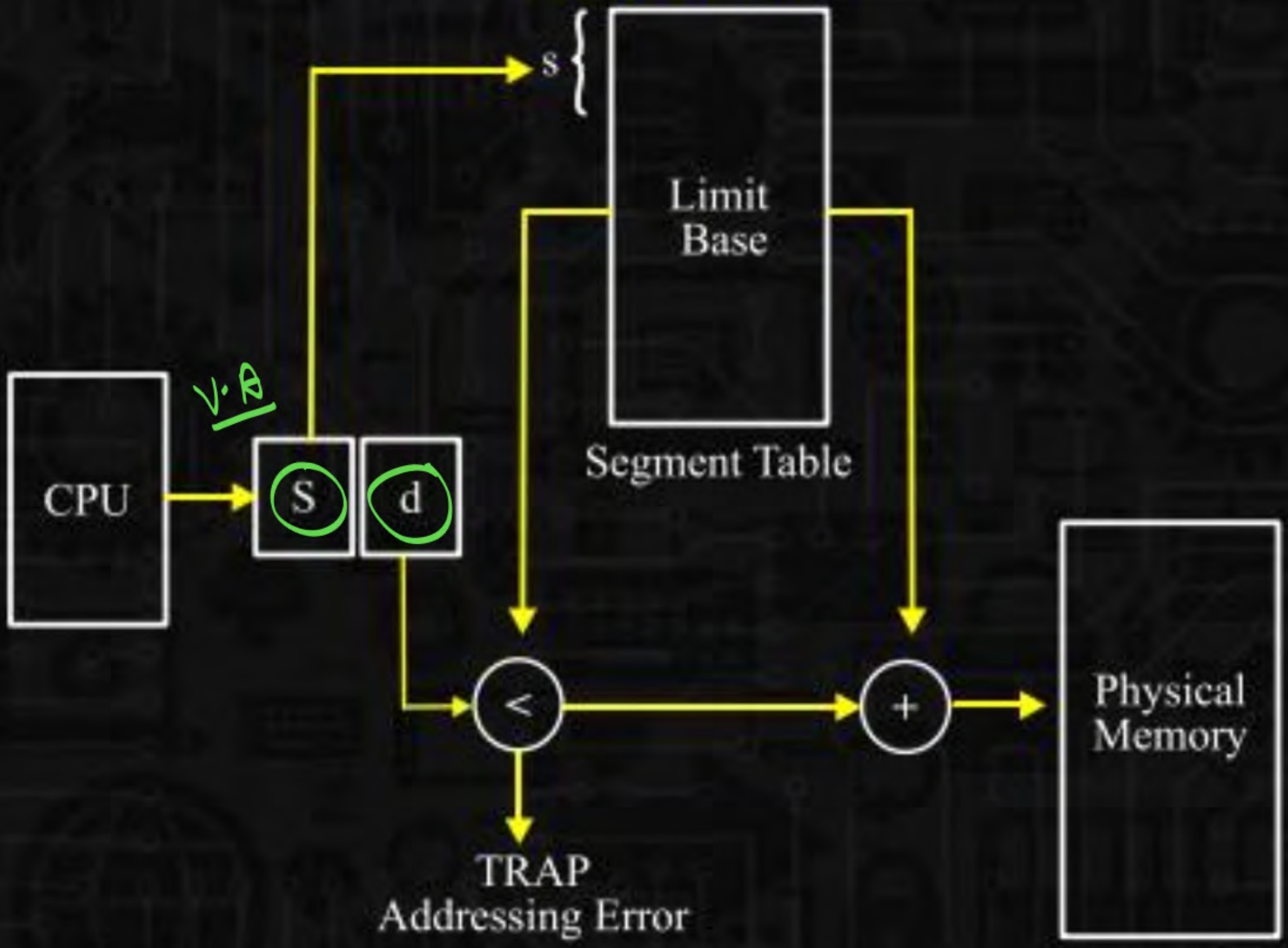
	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segment Table



Physical Memory

Segmentation



∴ Paging on Segment

L.A: 34 bit

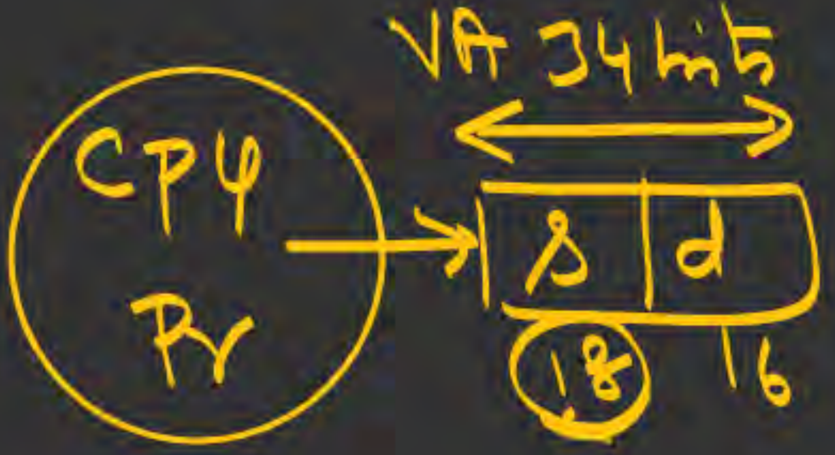
$\langle s, d \rangle = \langle 18, 16 \rangle$

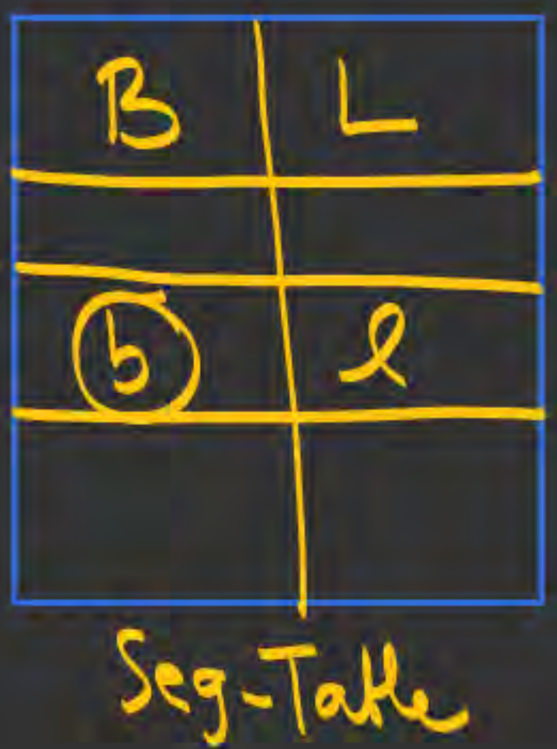
Max·Seg-Size $= 2^{16}$

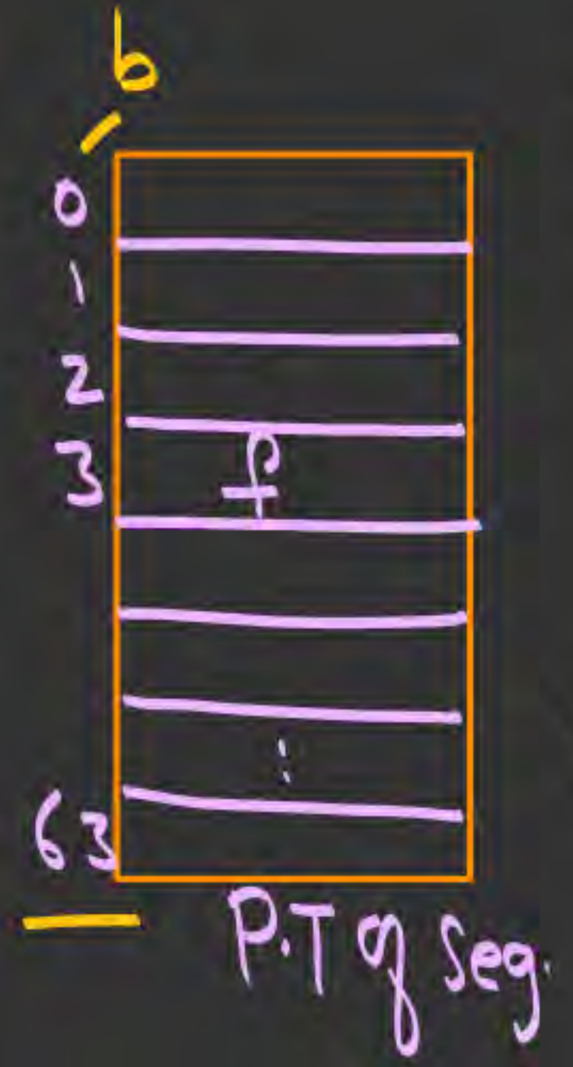
$= 64KB$

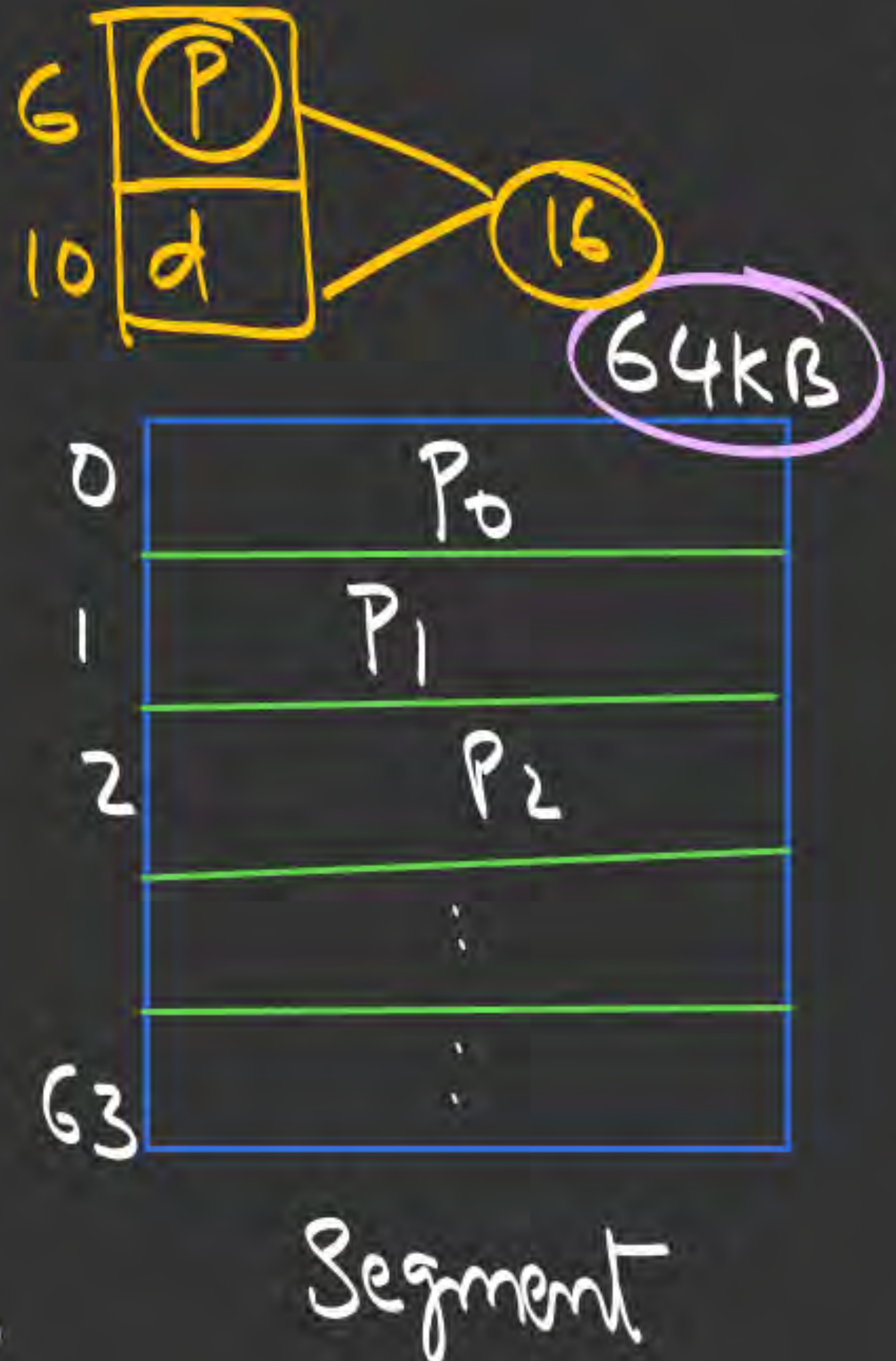
B = Addrs. of P.T of Seg.

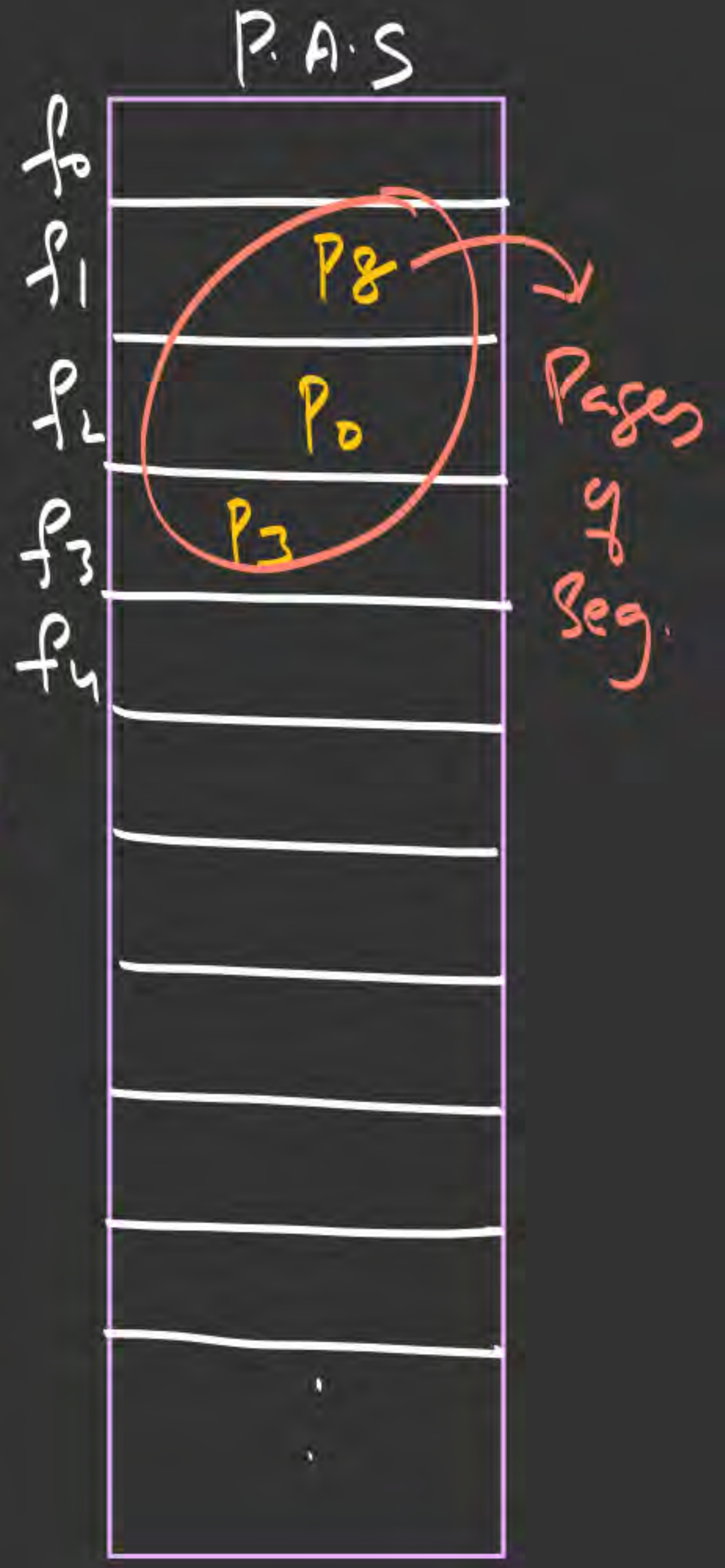
Let P.S of Segment = 1KB

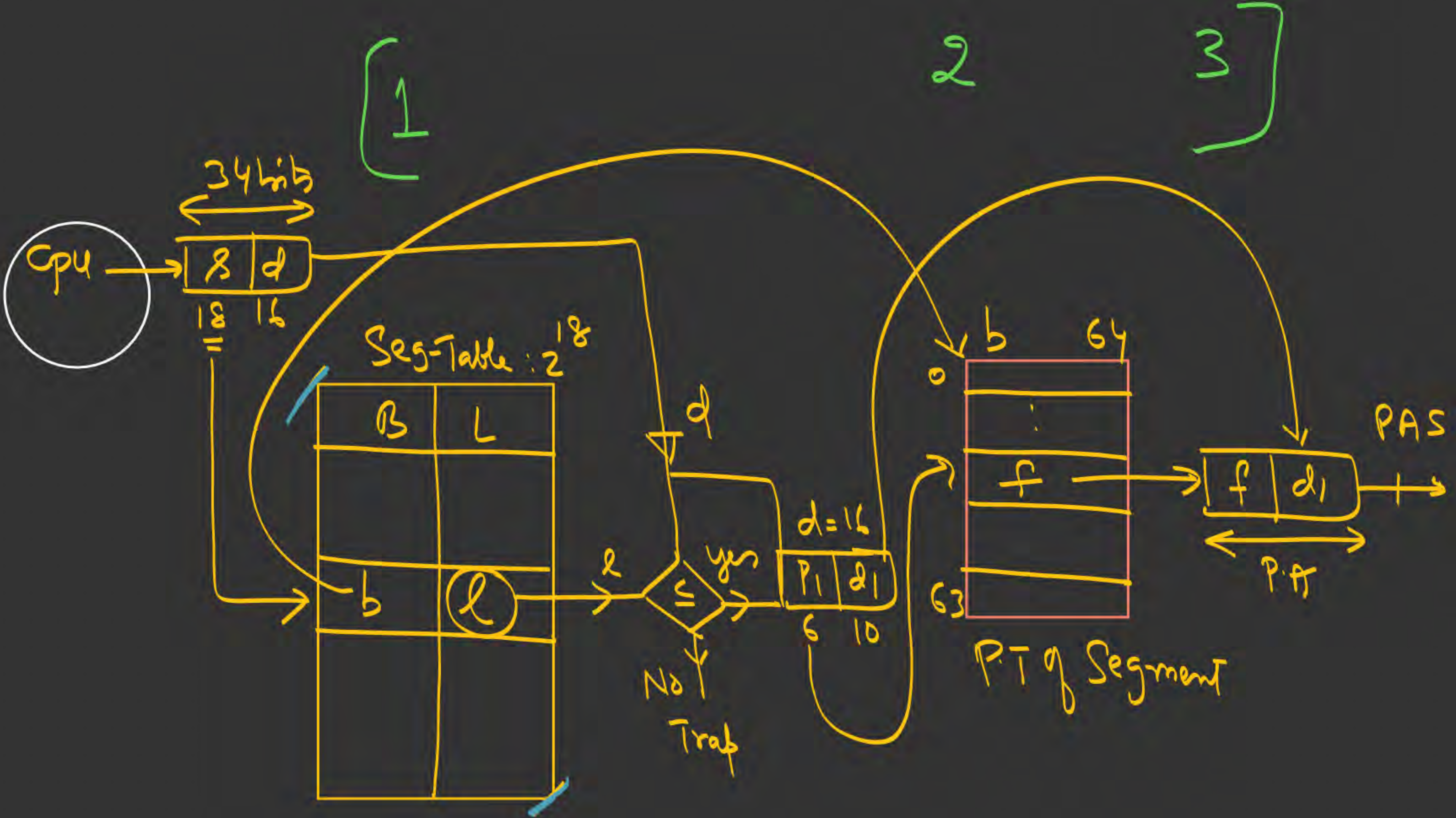


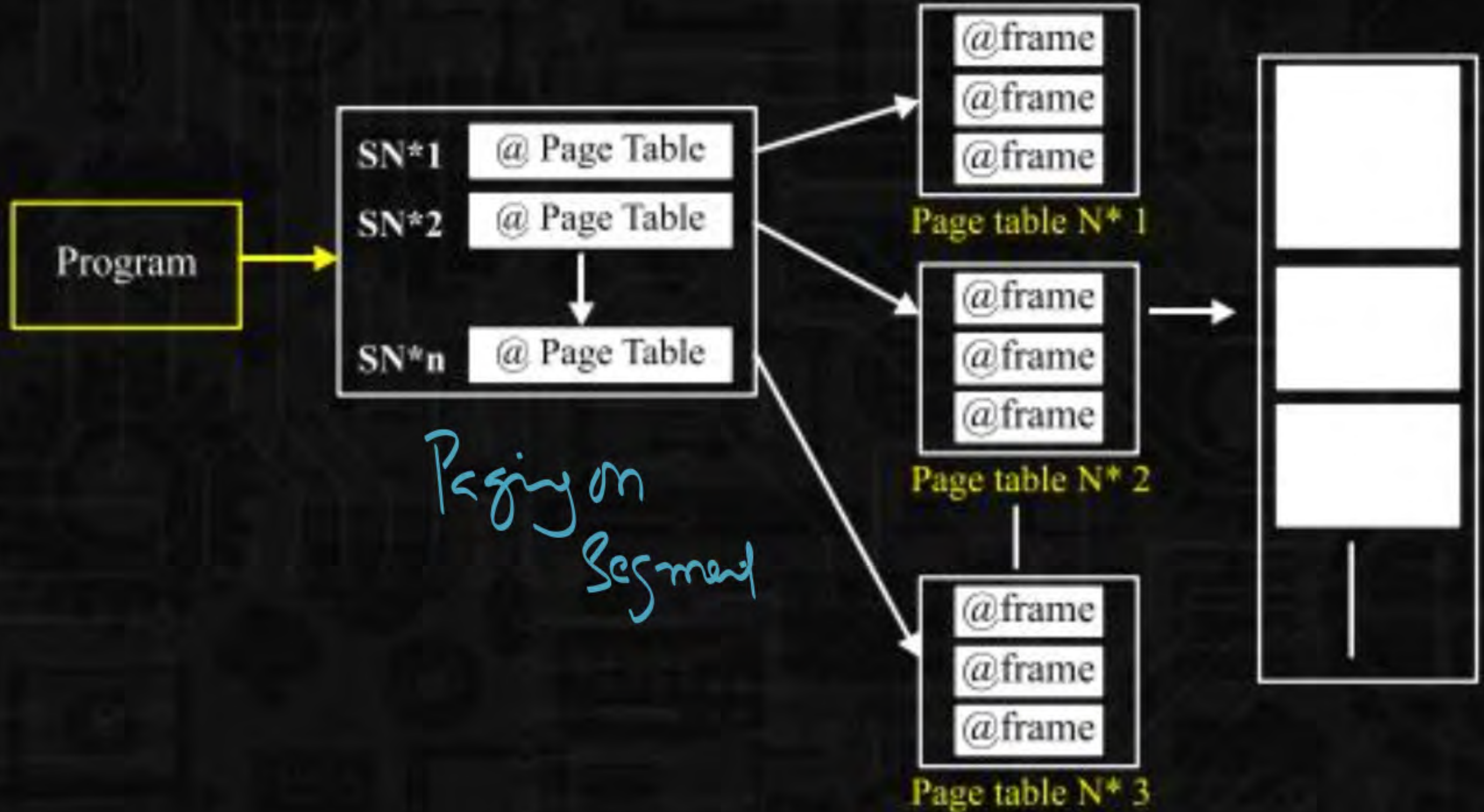








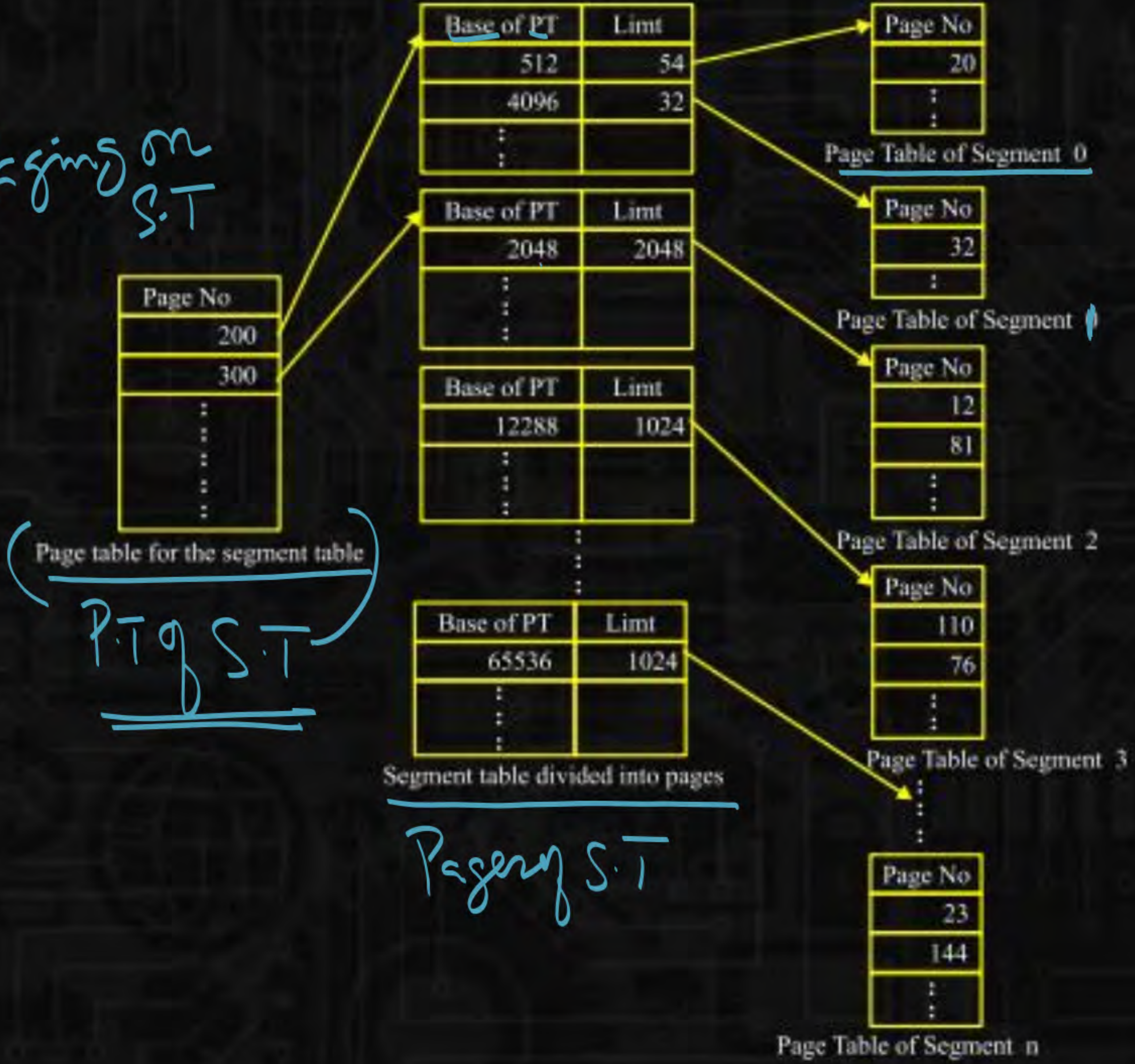

[1
2
3]
34 bits
Cpu
s
d
18
16
Seg-Table : 2^{18}
B
L
b
l
d
l
≤
yes
No
Trap
d=16
P1
d1
6
10
0
b
64
f
63
P.T of Segment
f
d1
PAS
P.A



Paging on Segment

Program divided into Segments → Segment Table with page Table Address → Page Table with frames addreses → Main Memory

Virtual Address: Segment Number + Page Number + Offset



Paging on S.T

Page No
200
300
⋮

Page table for the segment table

P.T of S.T

Base of PT	Limt
512	54
4096	32
⋮	

Base of PT	Limt
2048	2048
⋮	

Base of PT	Limt
12288	1024
⋮	

⋮

Base of PT	Limt
65536	1024
⋮	

Segment table divided into pages

Paging S.T

Page No
20
⋮

Page Table of Segment 0

Page No
32
⋮

Page Table of Segment 1

Page No
12
81
⋮

Page Table of Segment 2

Page No
110
76
⋮

Page Table of Segment 3

⋮

Page No
23
144
⋮

Page Table of Segment n

Q. 1

Consider a System using Segmented-Paging Architecture with V.A.S = P.A.S = 216 2^{16} Bytes. The VAS is divided into 8 equal sized nonoverlapping Segments. Paging is applied on Segment, with a Page size being a power of 2 in Bytes. The Page Table Entry size of the Page Tables of the Segment is 16 bits. What must be the Page Size of the Segment such that the Page Table of the Segment exactly fits in one frame of Memory.

Let $P.S = 2^x B$

$$P.T.S = \left(\frac{2^{13}}{2^x}\right) \cdot 2B$$

$$= 2^{13-x} \cdot 2^1 B$$

$$= 2^{14-x} B$$

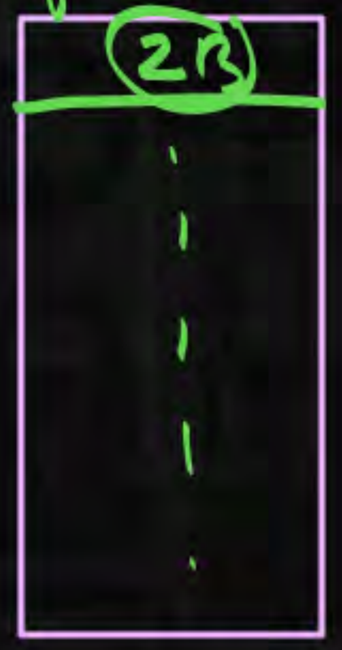
$$2^{14-x} = 2^x$$

$$14 - x = x$$

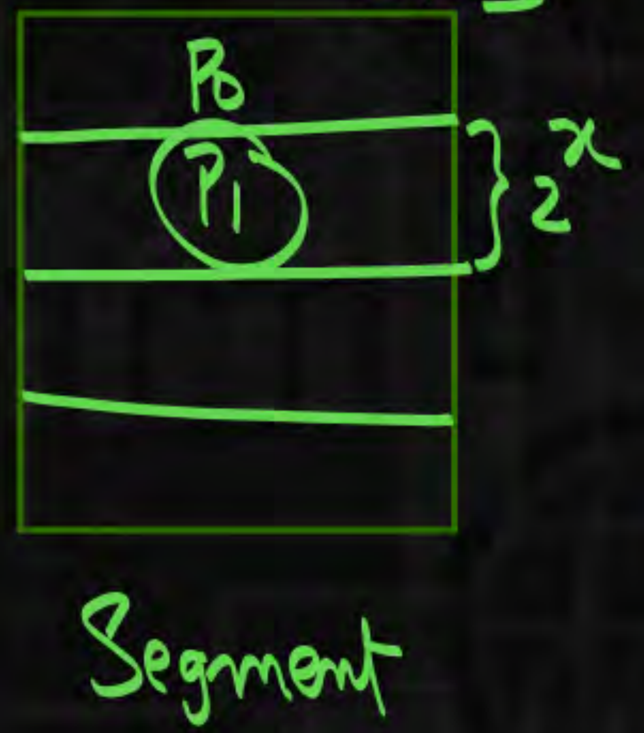
$$\therefore x = 7$$

$$\therefore P.S = 2^7 = 128B \checkmark$$

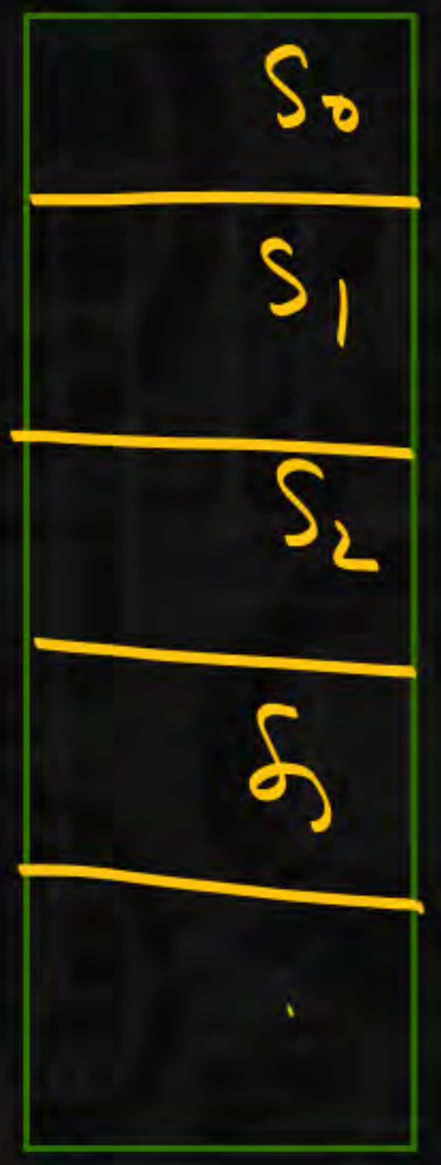
P.T of Seg.



8KB



VAS = 2^{16}



$$\frac{2^{16}}{8} = 2^{13}$$

Q. 2

Consider a System using Segmented-Paging Architecture. Paging is applied on Segment. The System maintains a 256 entry Page-Table per Segment. The Page Size of Segment is 8Kbytes. The Virtual Address Space supports 2K Segments. Page Table Entry Size is 16 bits while the Segment Table entry is 32 bits in size.

$\boxed{VAS = 2K * 2MB = 4GB}$

Calculate

✓ (a) Size of Virtual Address Space

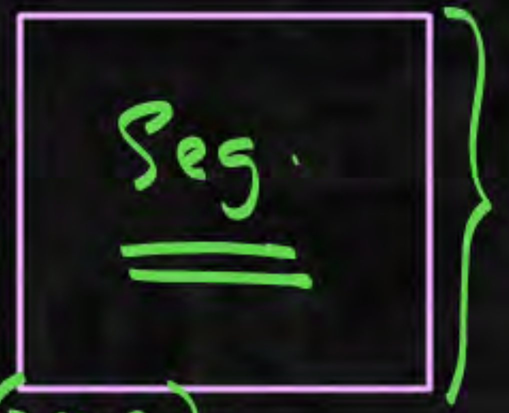
S.T.S + P.T.S ← (b) Address Translation Space Overhead in Bytes.

(c) The number of levels of Memory accesses required for Address Translation. (3)

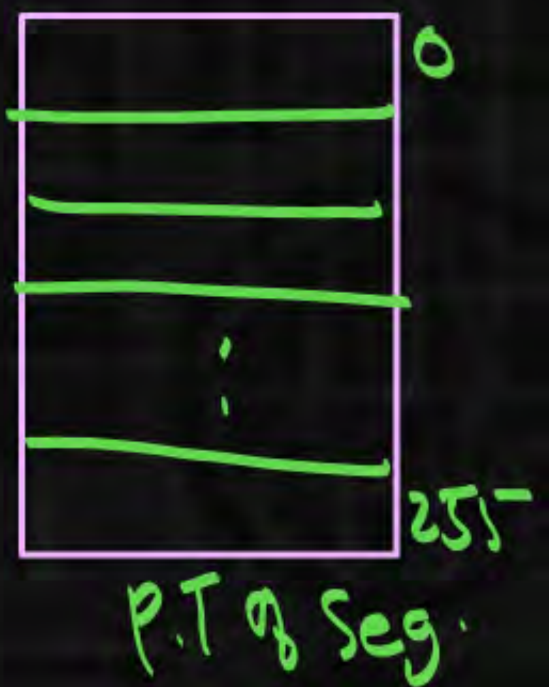
$256 \times 8KB$

$2^8 \times 2^3 \times 2^{10}$

$= 2^{21} = 2MB$



$$\left.\begin{aligned} S.T.S &= 2K * 4B = 8KB \\ P.T.S &= 256 * 2B = 512B \end{aligned}\right\} \boxed{8.5\,KB}$$



⇒ There are 256 pages in Segment

Shared Pages

→ Optimize Memory

: Compilers, Editors can be shared by multiple applications;

Shared Pages

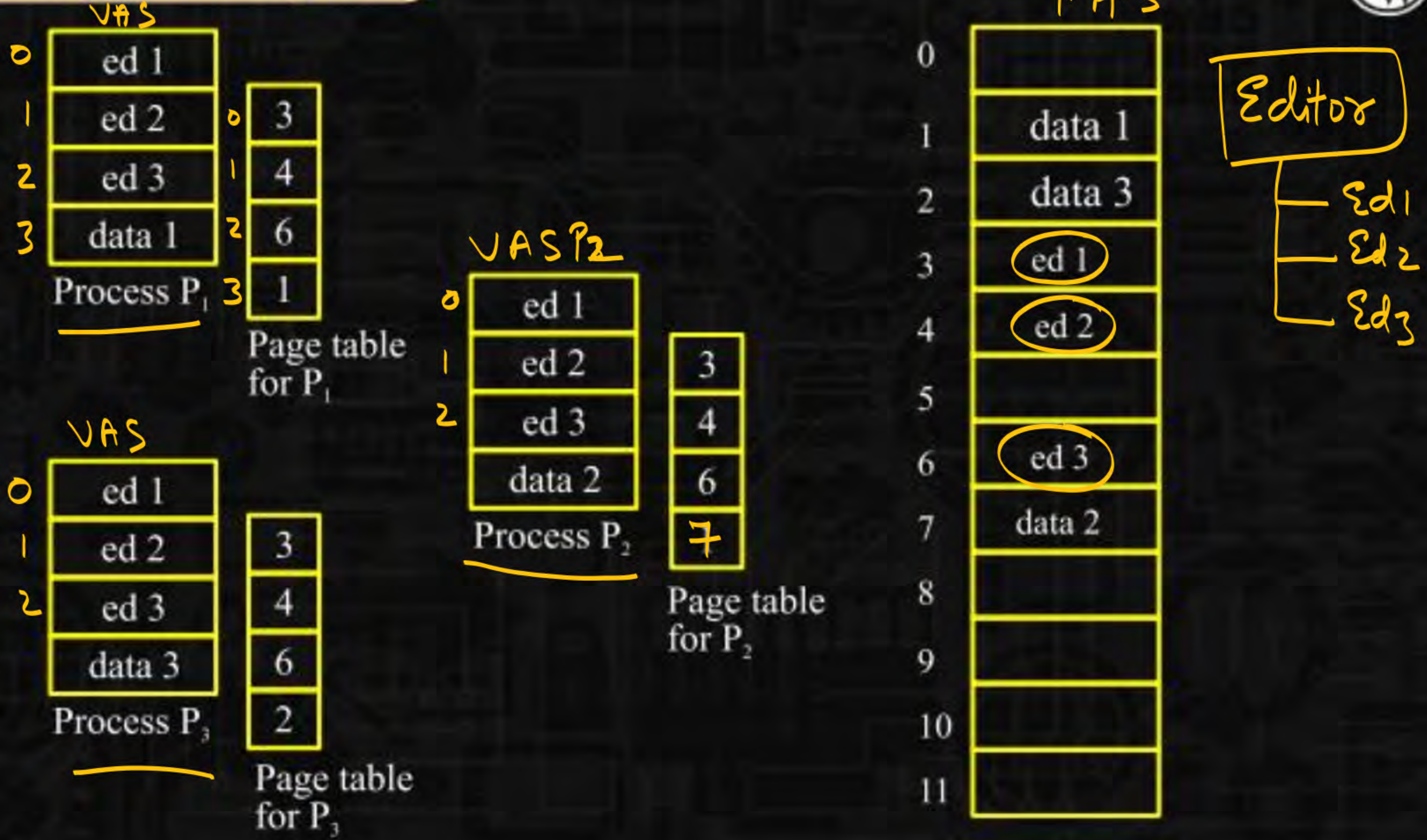
Shared code

- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
- Shared code must appear in same location in the logical address space of all processes

Private code and data

- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



VAS

	Process P1
0	ed 1
1	ed 2
2	ed 3
3	data 1

	Page table for P1
0	3
1	4
2	6
3	1

VAS

	Process P3
0	ed 1
1	ed 2
2	ed 3
	data 3

Page table for P3
3
4
6
2

VAS P2

	Process P2
0	ed 1
1	ed 2
2	ed 3
	data 2

Page table for P2
3
4
6
7

P.A.S

0	
1	data 1
2	data 3
3	ed 1
4	ed 2
5	
6	ed 3
7	data 2
8	
9	
10	
11	

Monitors

<Synchronization Mechanism]

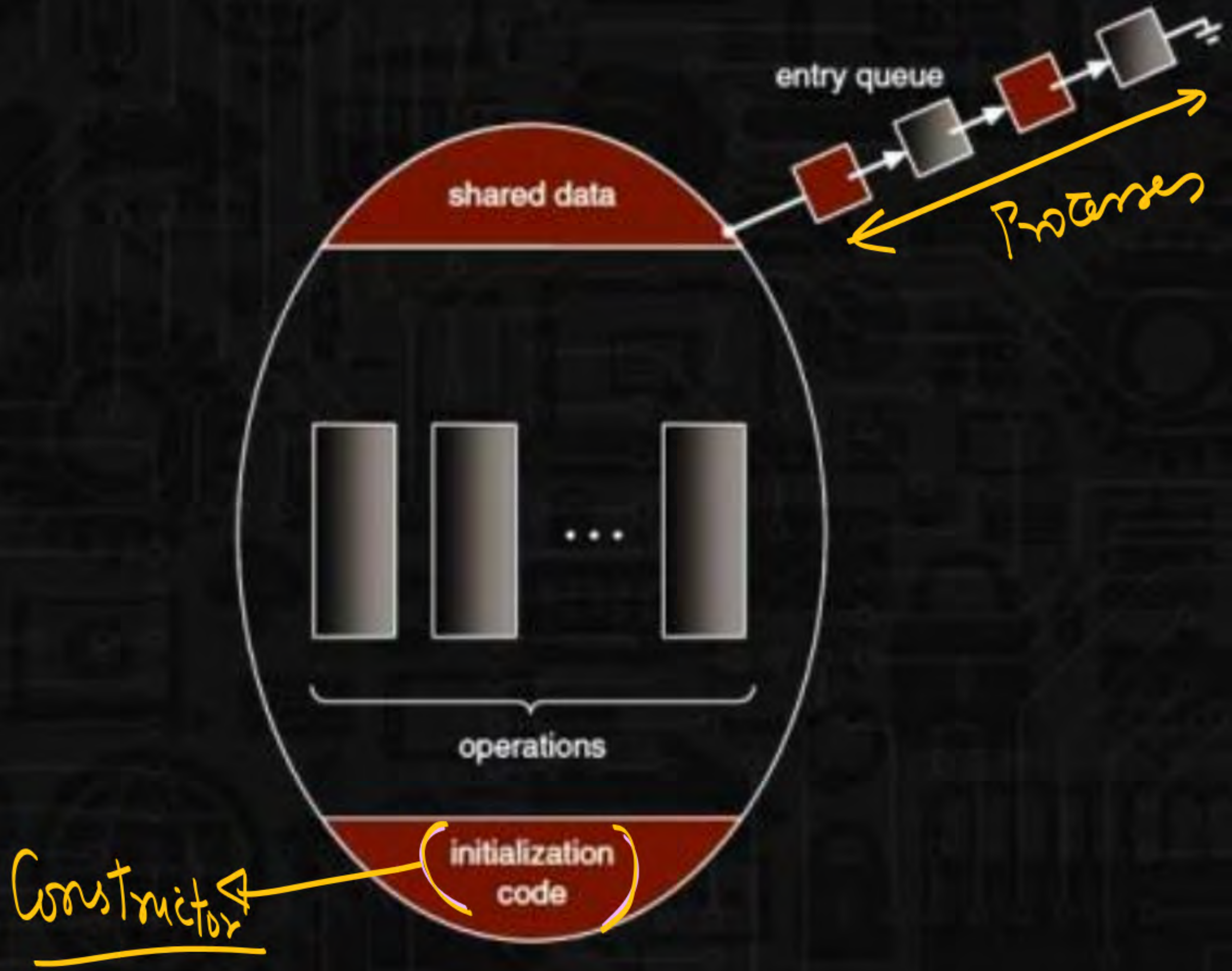
class

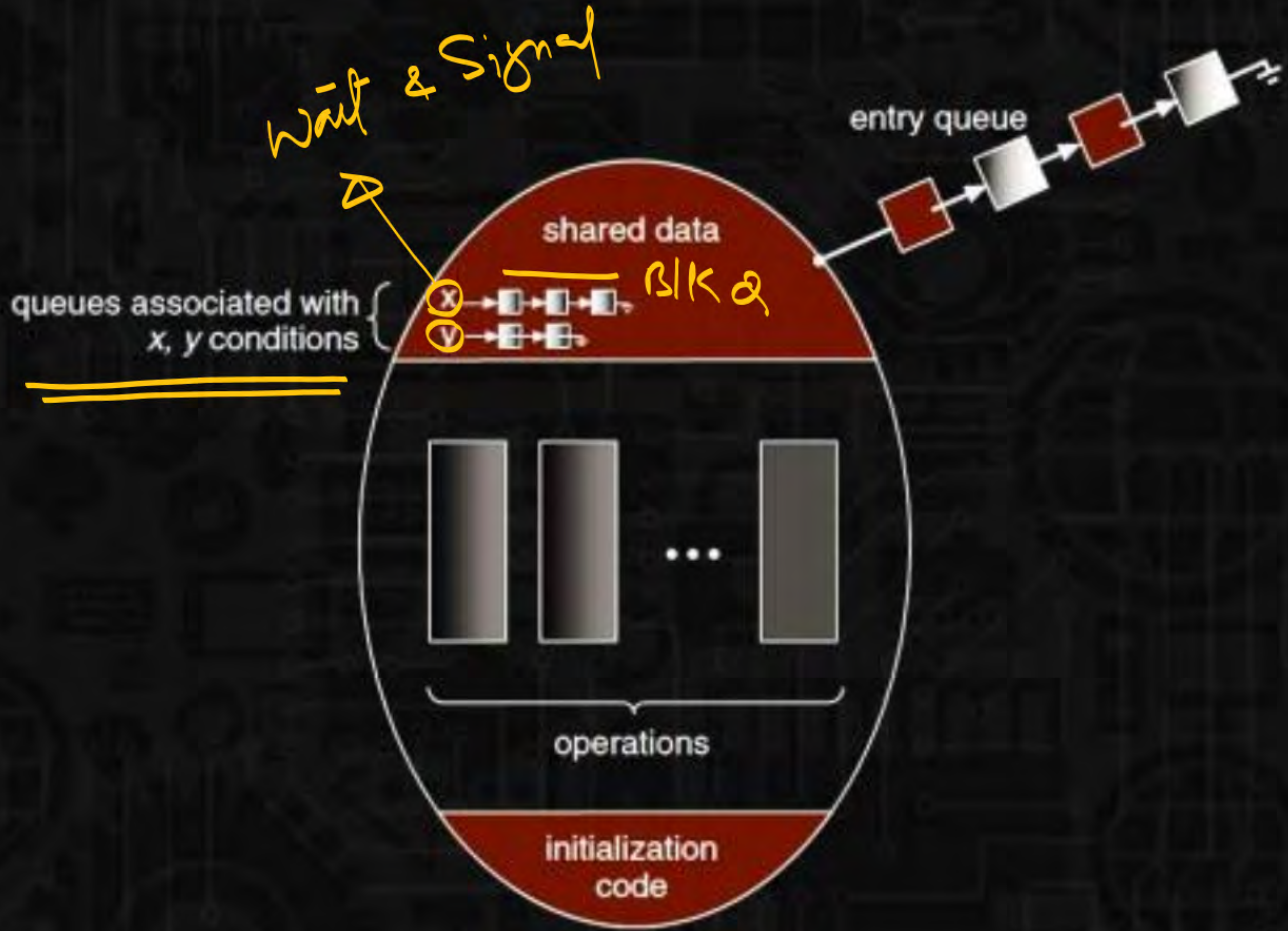
→ is a Collection of Procedures, variables and other Data Structures that are all grouped together in a Special Kind of Module/Package;

→ Procedures that run outside the Monitor Cannot access, Monitors internal variables and data structures. They Can only activate Monitor Member fn's;

→ Monitor also include Special condition variables with wait & Signal op'n's;

→ only one Process Can be active in the Monitor at any time;


entry queue
Processes
shared data
...
operations
initialization code
Constructor



entry queue

shared data

queues associated with x, y conditions

x

y

operations

initialization code

Wait & Signal

Blk Q

```
monitor monitor_name
{
   /* shared variable declarations */
   function Pl (• • •) {
   } • • •
   function P2 (• • •) {
   } ···
               •
               •
               •
   function Pn (• • • ) {
   • • •
   }
   initialization_code (• • •){
   }• • •
}
```

class KRK
{

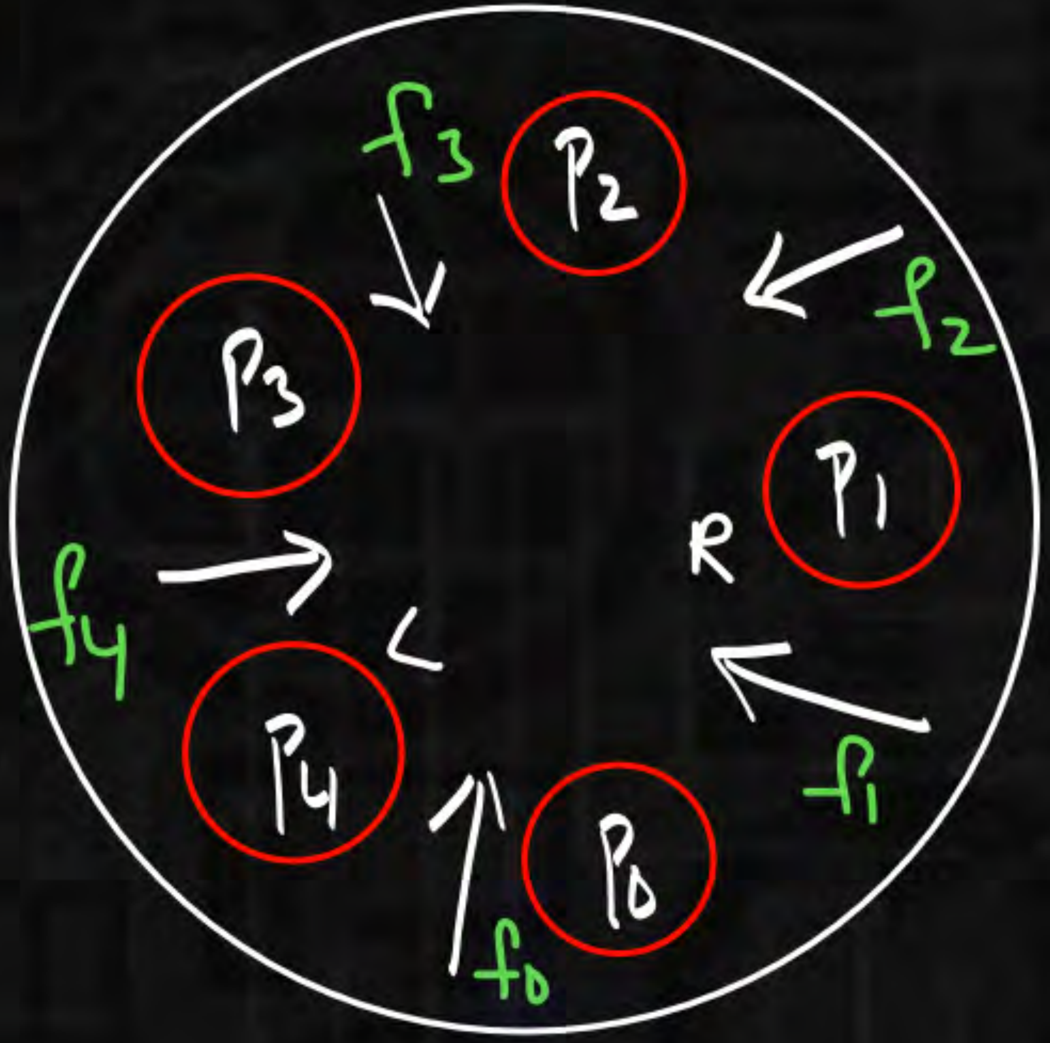
}

Dining Philosophers Problem : MONITORS

```
monitor DiningPhilosophers
{
        enum {THINKING, HUNGRY, EATING} state[5];
        condition self [5];
        void pickup(int i)
        {
                state[i] = HUNGRY;
                test(i);
                if (state[i] != EATING)
                self[i].wait();
        }
        void putdown(int i) {
                state[i] = THINKING;
                test((i + 4) % 5); L
                test((i + 1) % 5); R
        }
        void test(int i) {
                if ((state[(i + 4)% 5]!= EATING) &&
                (state[i] == HUNGRY) &&
                (state[(i + 1) % 5]!= EATING)) {
                        state[i] = EATING;
                        self[i].signal();
                }
        }
  initialization_code() {
  for(int i = 0; i < 5; i++)
  state[i] = THINKING;
  }
}
```

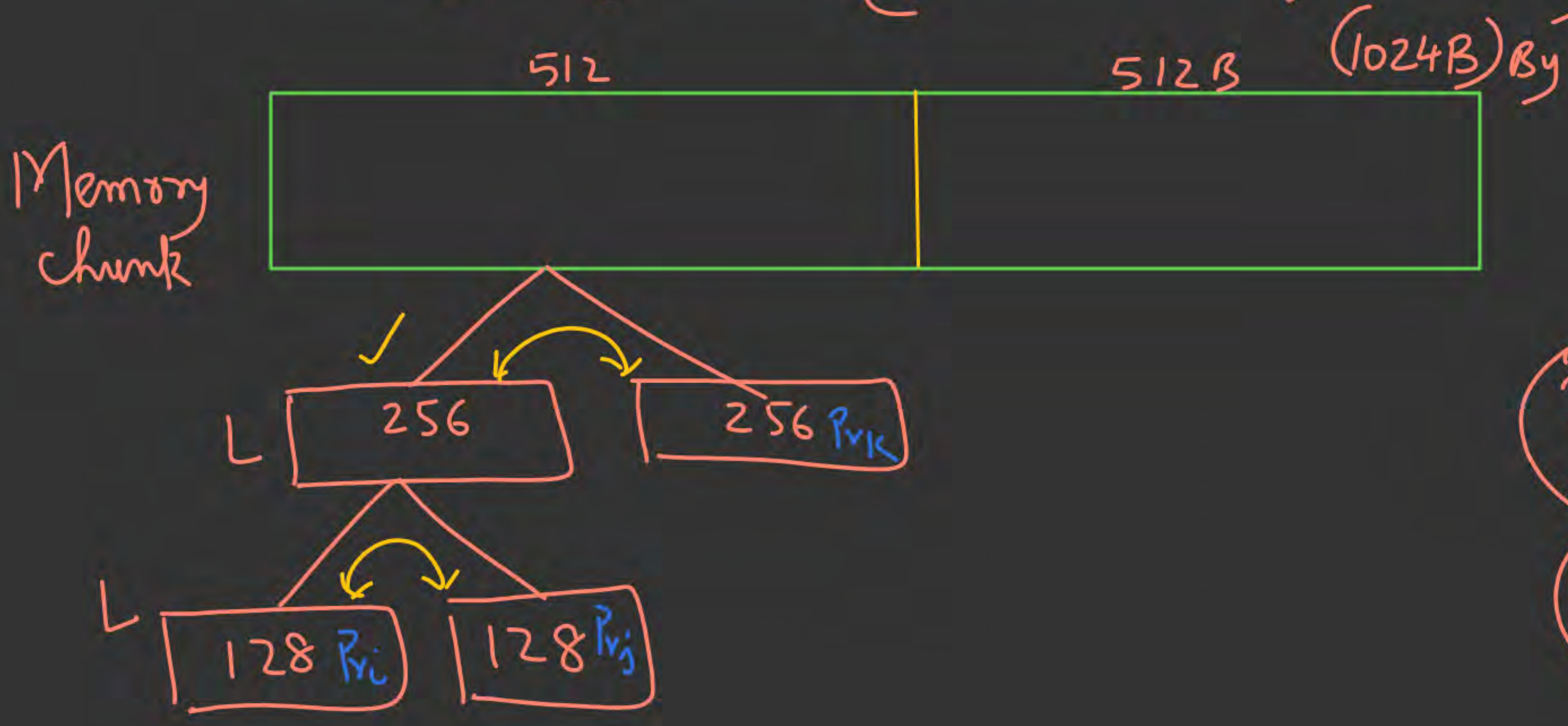
Deadlock free Soln

Enum{THINKING, HUNGRY, EATING}state[5];
Condition self[5];



Buddy System (variation of M.V.T)

(Power of 2 Allocation)



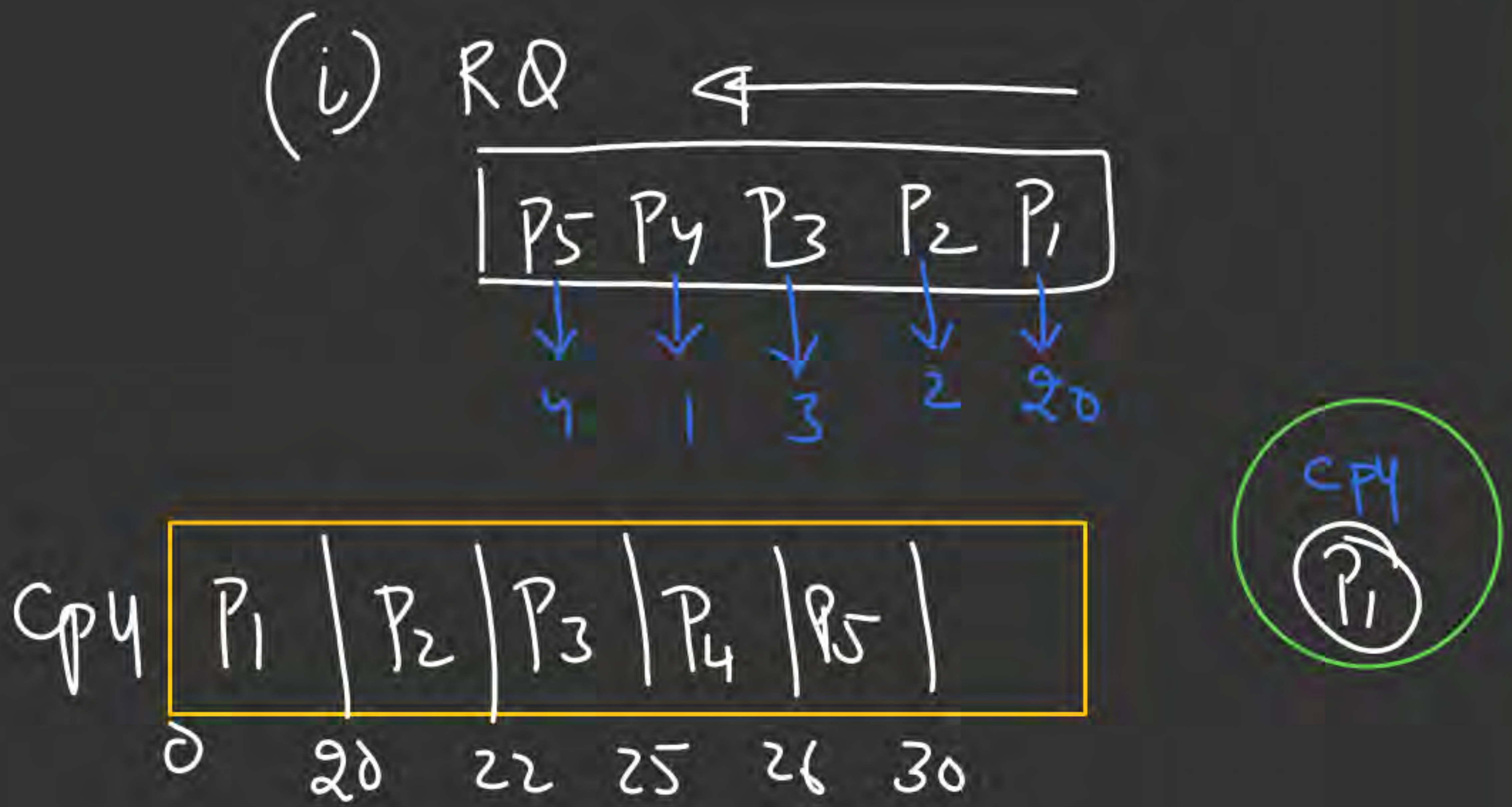
Pri : 100B

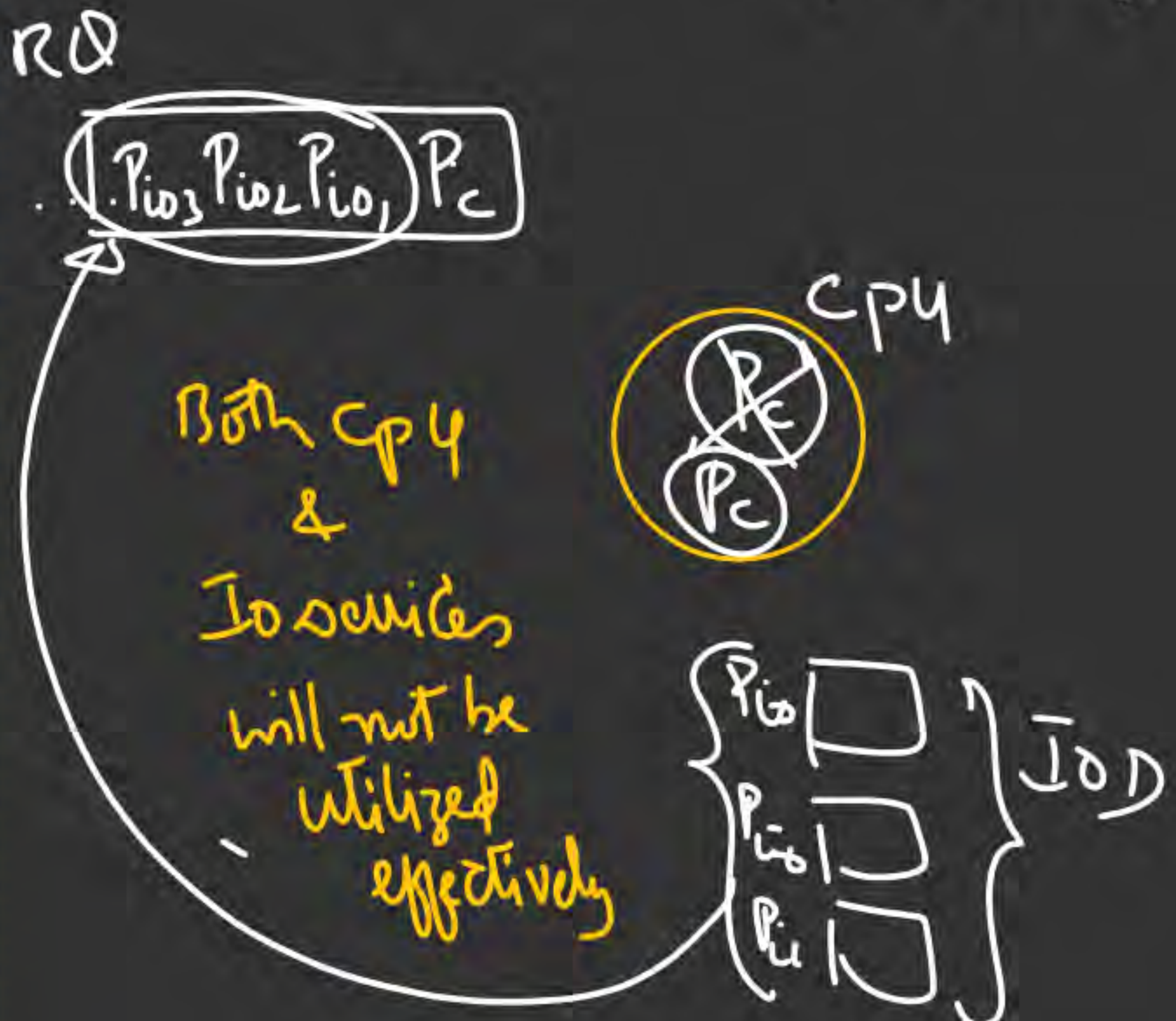
(Smallest Buddy that can accommodate 100B)

Convoy effect:

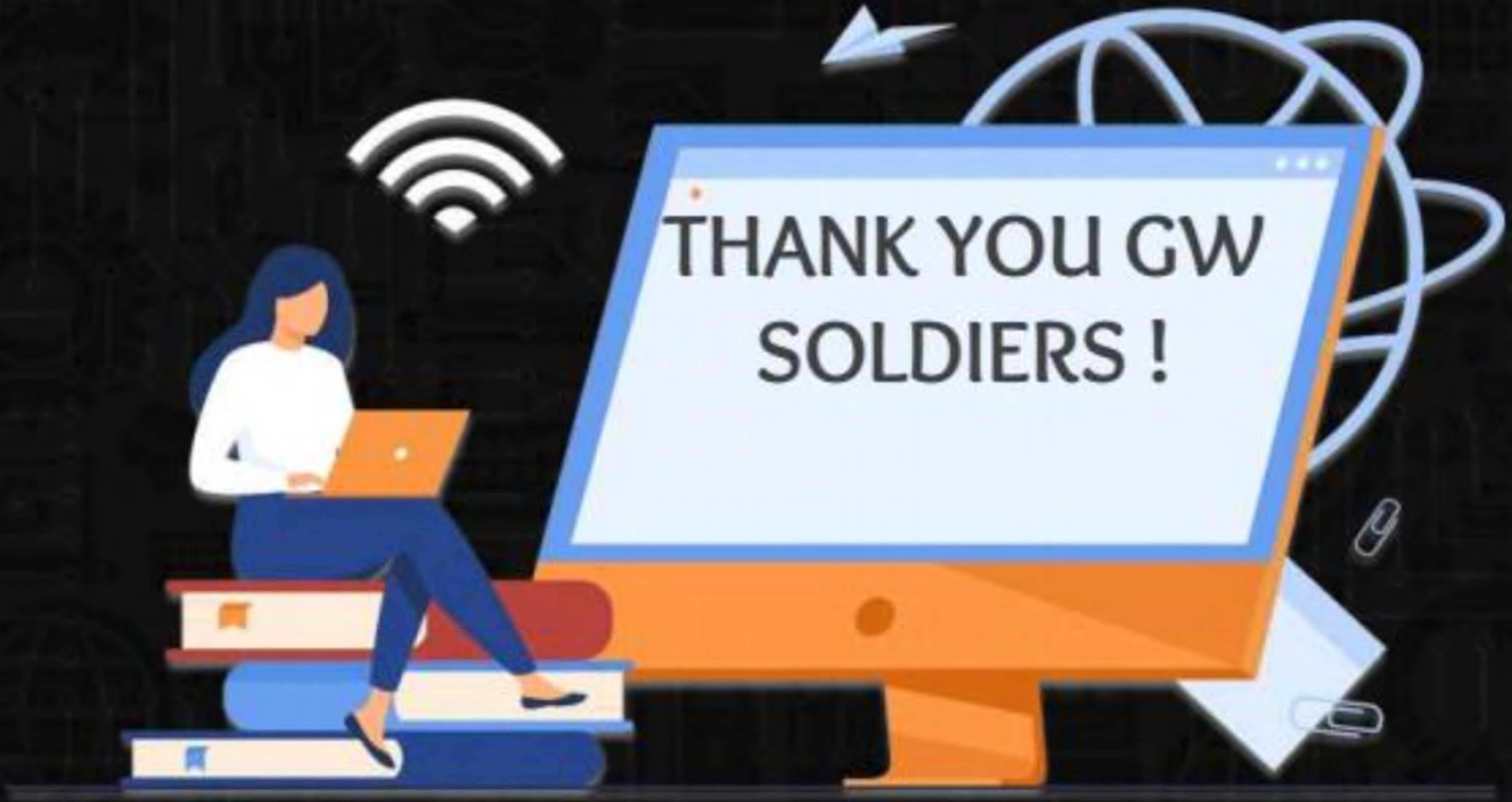
↳ FCFS Scheduling:

When a Long Process (CPU Bound) is currently occupied on CPU & thereby causing a waiting for other Processes





Both CPU & Io services will not be utilized effectively



THANK YOU GW SOLDIERS !