CS & IT ENGINEERING

Operating System

Inverted Paging, Monitors and Shared Pages

Lecture no:01

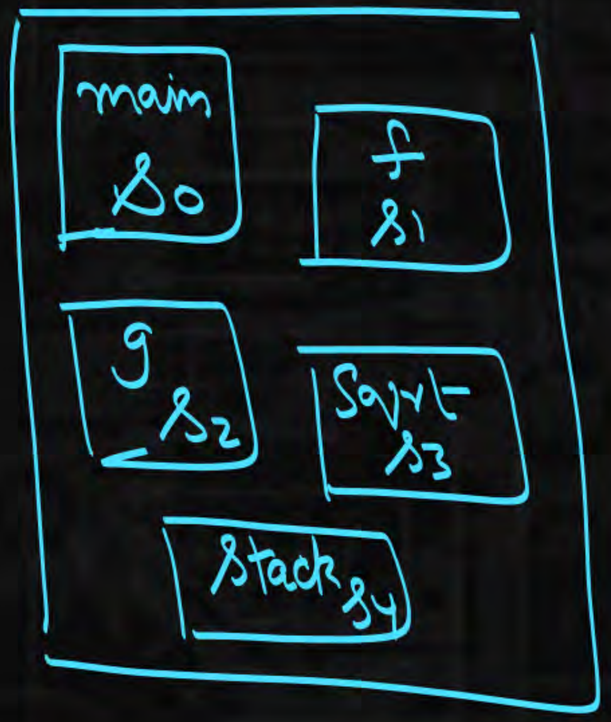
By- Dr. Khaleel Khan sir

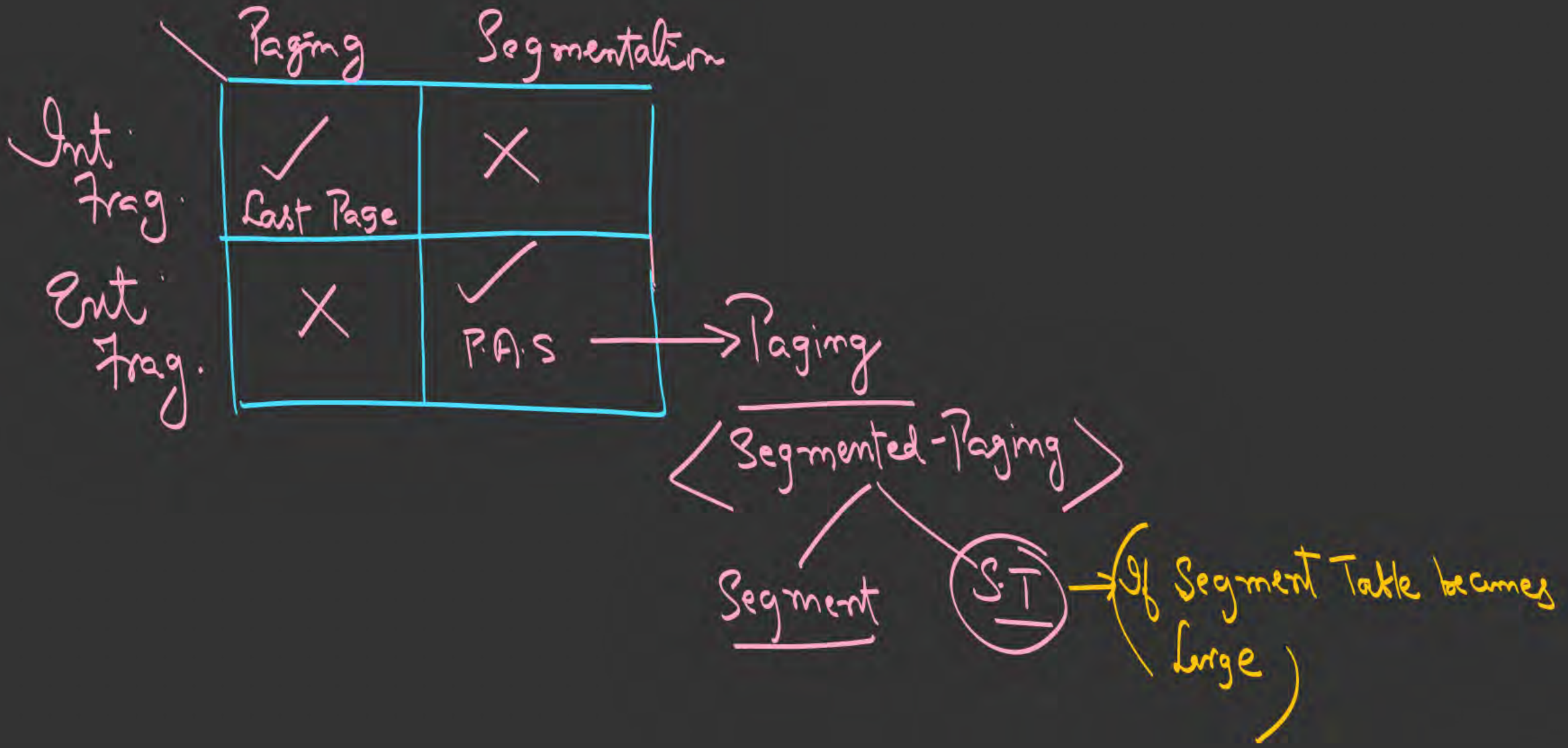
Inverted Paging, Monitors and Shared Pages

Segmented Paging Architecture

Segmentation

↳ to guarantee (Preserve) user's view of Mem. alloc. to Programs;




Paging
Segmentation
Int. Frag.
✓
Last Page
×
Ext. Frag.
×
✓
P.A.S
Paging
Segmented-Paging
Segment
S.T
If Segment Table becomes Large

Segmented - Paging :

LA = 34 bits

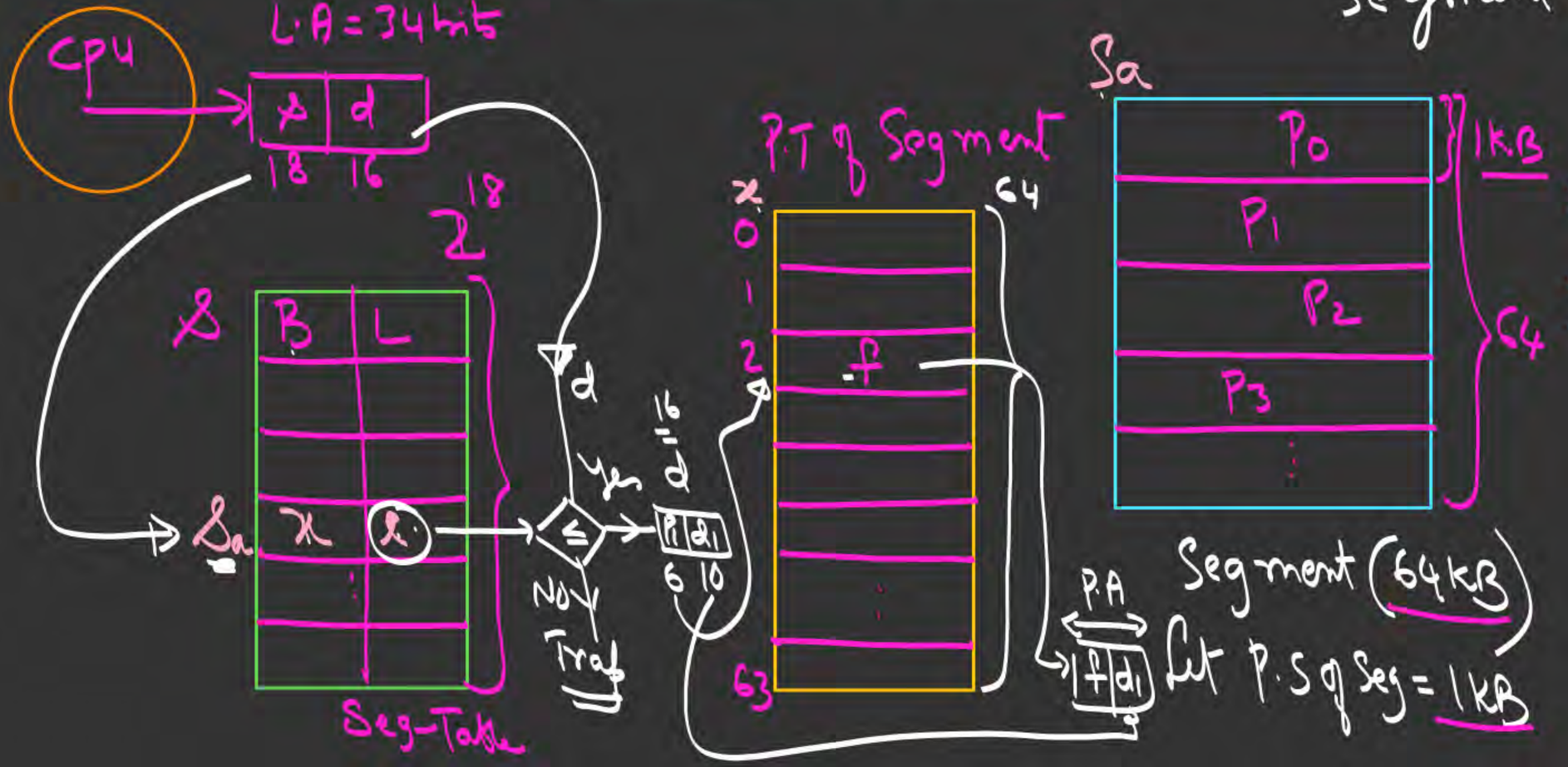
<s, d> = <18, 16>

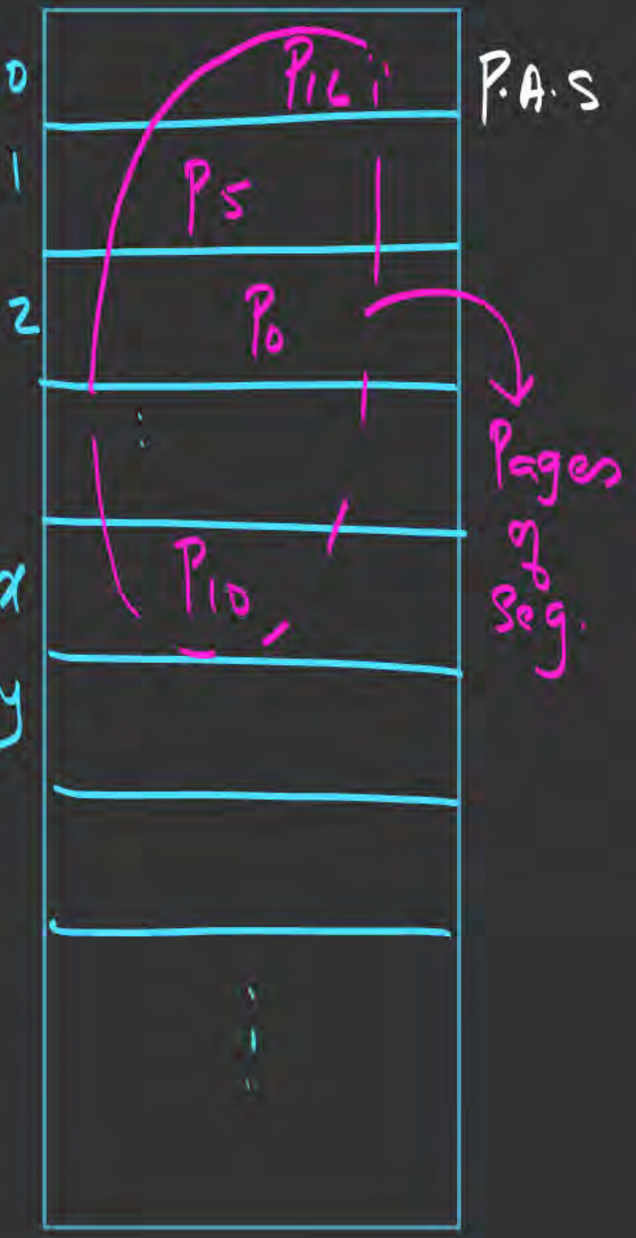
No. of Segs = 2^{18}

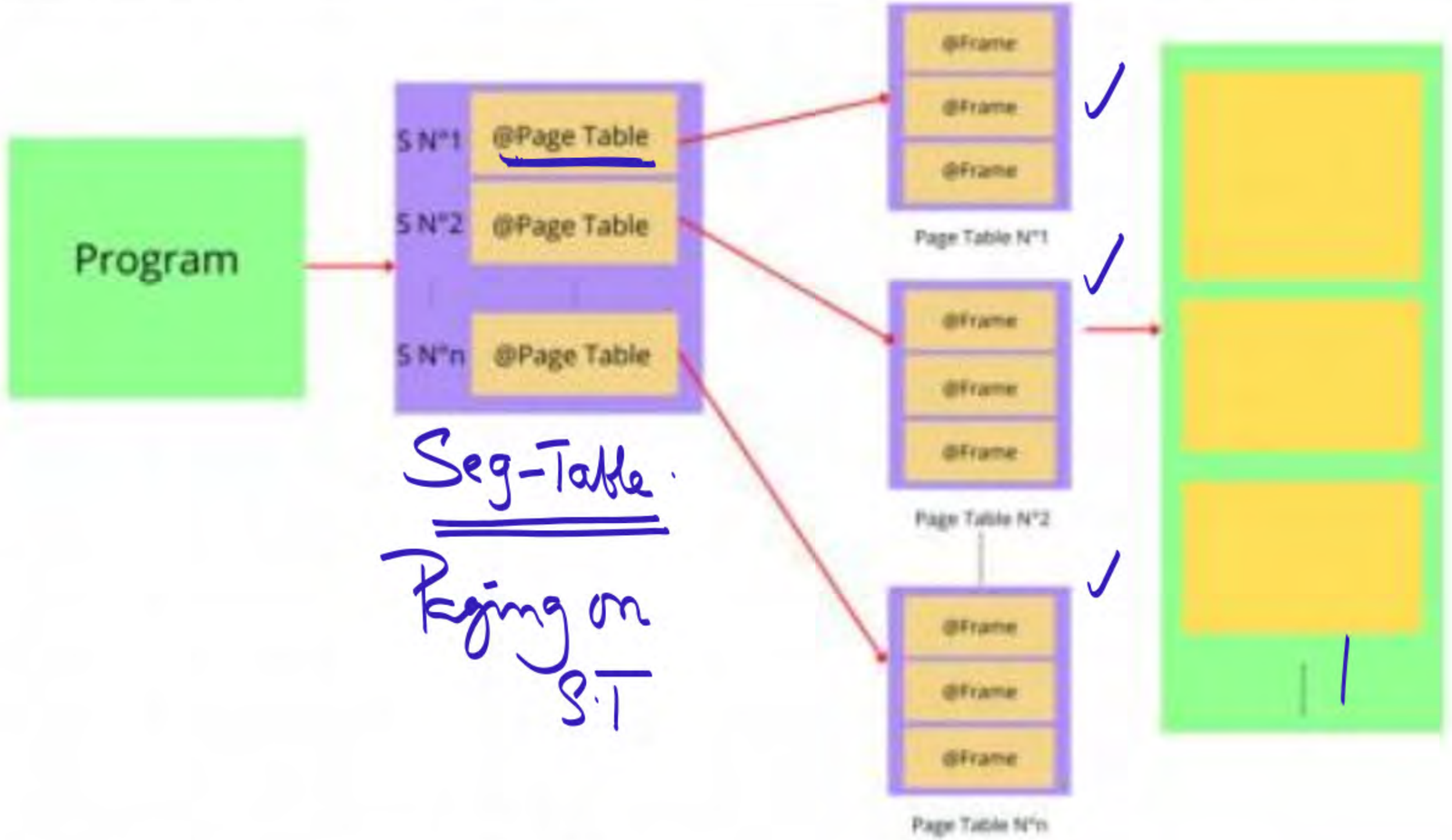
$\Rightarrow$ Max. Seg Size = 2^{16} = 64KB

B = Address of PT of the Seg.

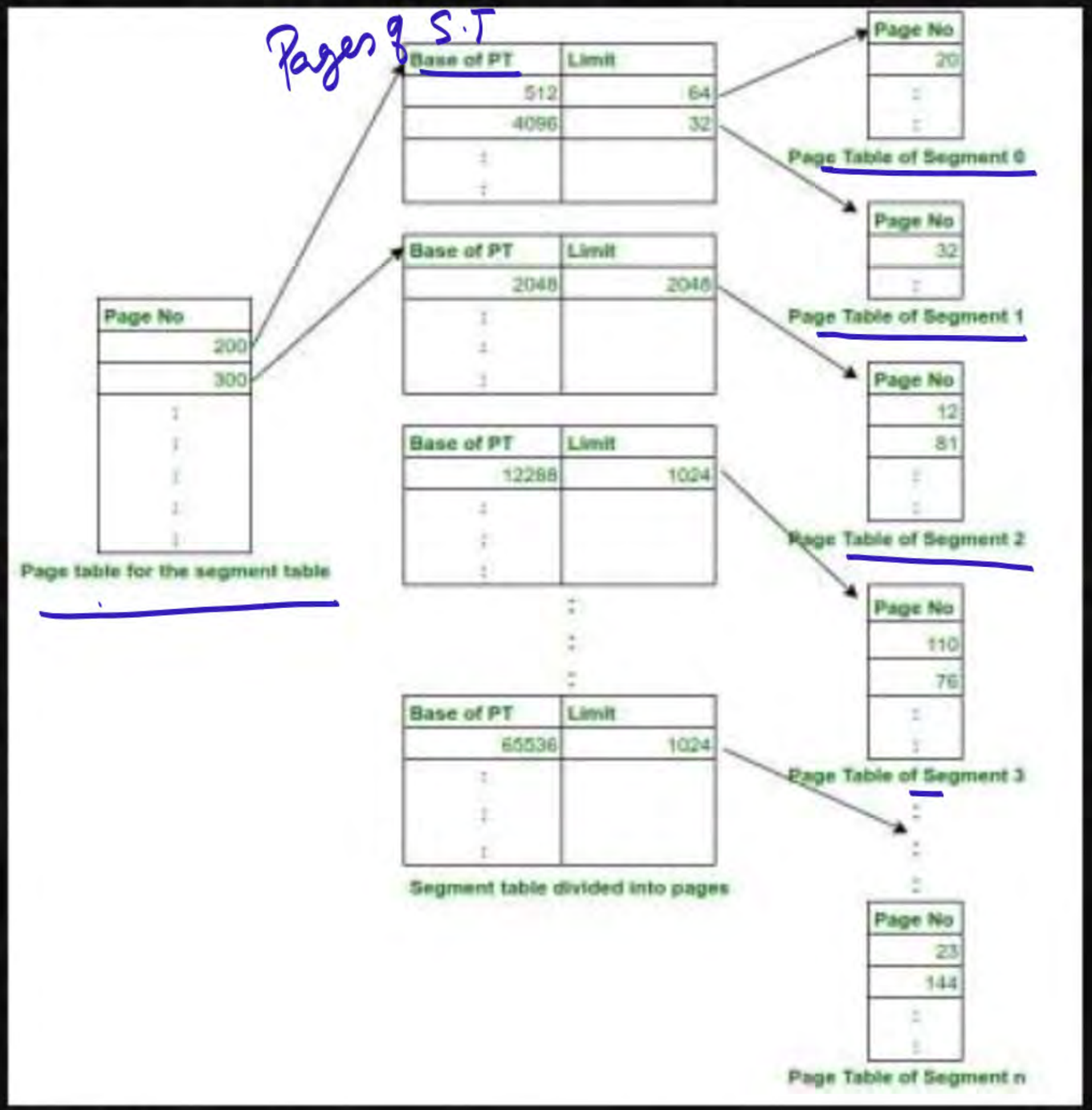
Apply Paging on Segment

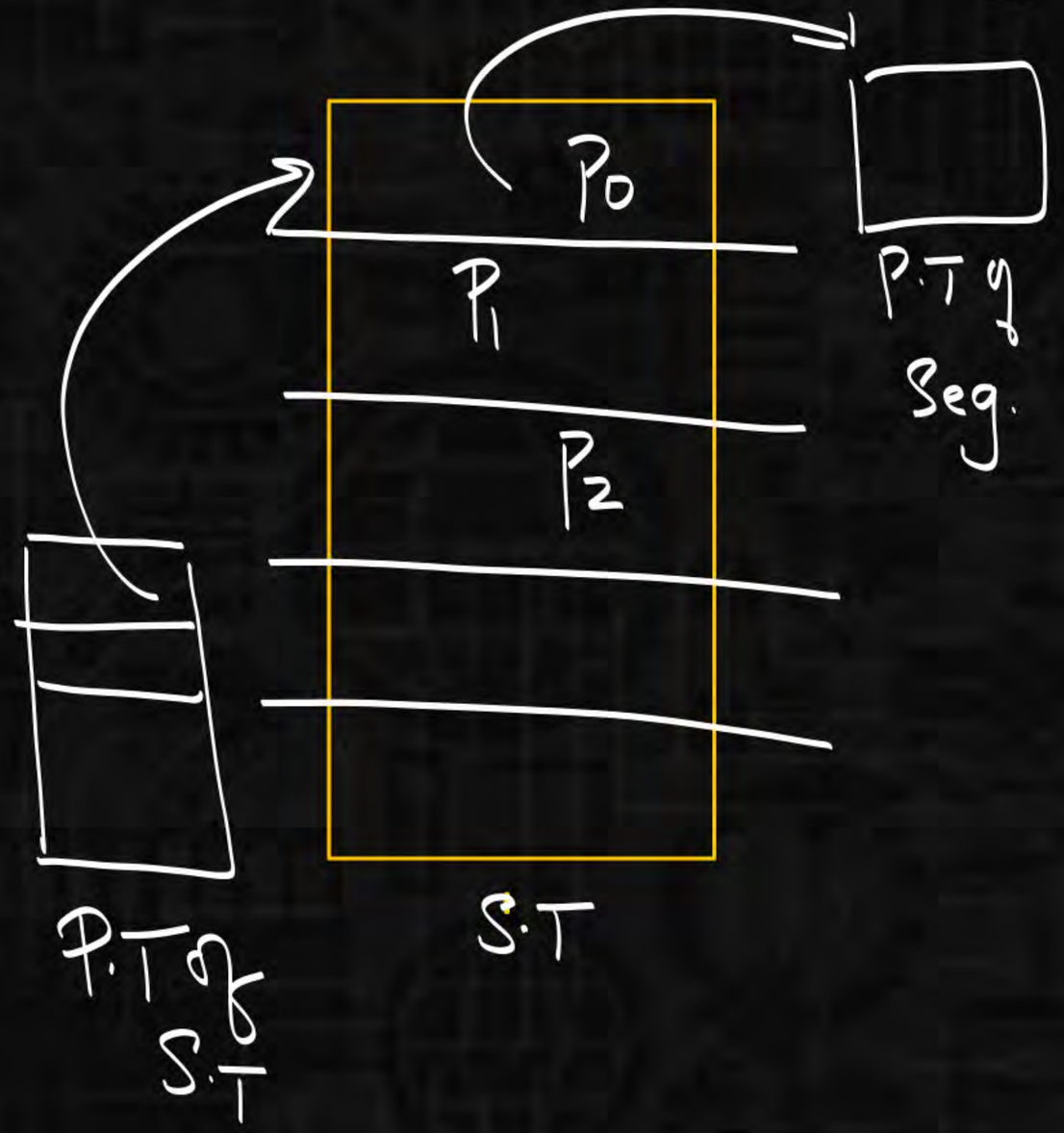






Virtual Address : Segment Number + Page Number + Offset


Pages & S.T
Base of PT
Limit
512
64
4096
32
Page No
20
Page Table of Segment 0
Page No
32
Page Table of Segment 1
Base of PT
Limit
2048
2048
Page No
12
81
Page Table of Segment 2
Base of PT
Limit
12288
1024
Page No
110
76
Page Table of Segment 3
Base of PT
Limit
65536
1024
Page No
23
144
Page Table of Segment n
Page No
200
300
Page table for the segment table
Segment table divided into pages


P0
P1
P2
P.T of Seg.
P.T of S.T
S.T

Q.1 Consider a System using Segmented-Paging Architecture with V.A.S = P.A.S = 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 By V.A.S = 2^{16} By

$\binom{2^{14-x}}{2} \quad 2^x$

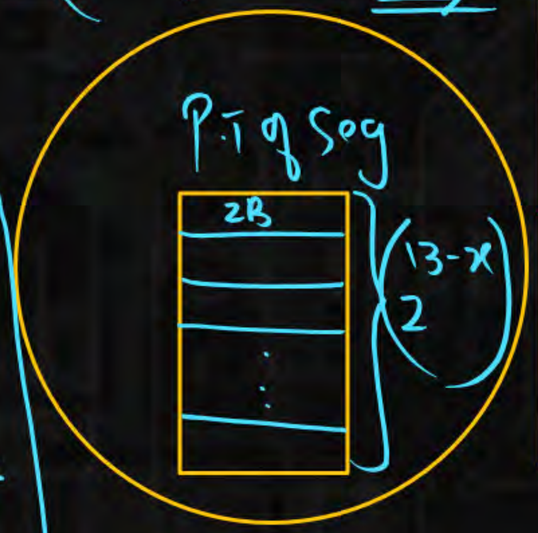
$\therefore 2^{14-x} = 2^x$

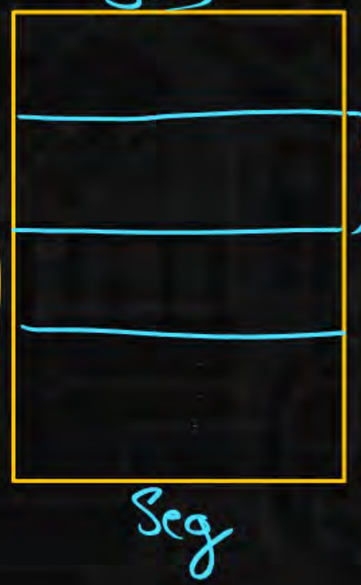
$\therefore 14 = 2x$

$\therefore x = 7$

P.S = 2^7 = 128B

P.T.S = $\binom{2^{13-x}}{2}$ 2B = $\binom{2^{14-x}}{2}$ B





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

$N = \frac{2^{13}}{2^x} = \binom{2^{13-x}}{2}$

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.

Calculate

(a) Size of Virtual Address Space

(b) Address Translation Space Overhead in Bytes.

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

e_1 e_2

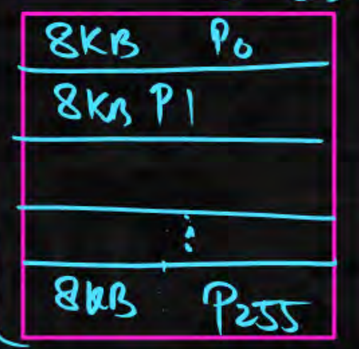
S.T + P.T

S.T Size = 2K * 4B = 8KB ✓

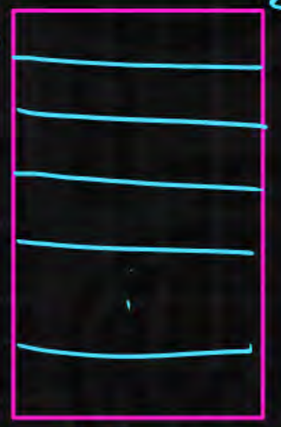
P.T Size = 256 * 2B = 512B

2K Seg's ∴ Seg Size = 256 * 8KB = $2^8 \times 2^{13} = 2^{21}$ = 2MB

V.A.S = No. of Seg's * Seg. Size = 2K * 2MB = 4GB ✓

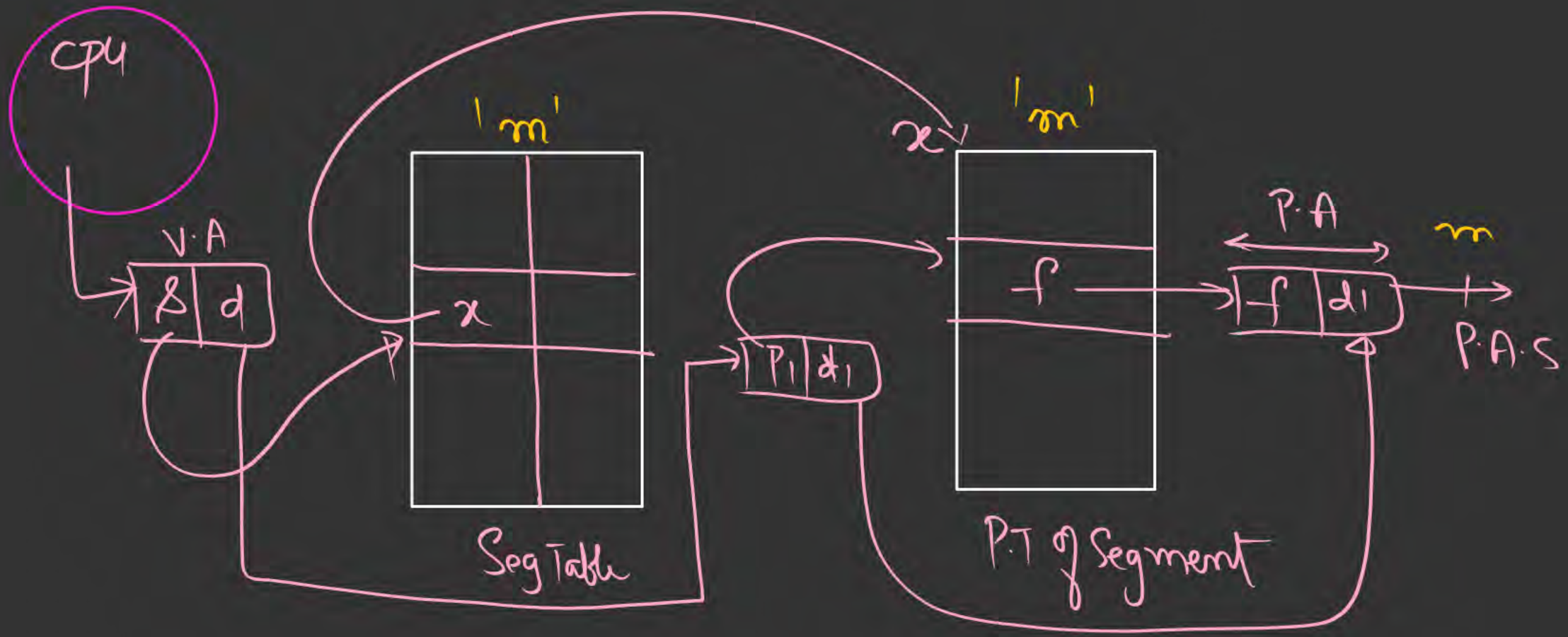


∴ 256 Pages in Seg.



$M.M.A.T = 'm'$

$EMAT = 3m$



Q.3 Which one of the following statements is FALSE?

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

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.

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

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.

Shared Pages

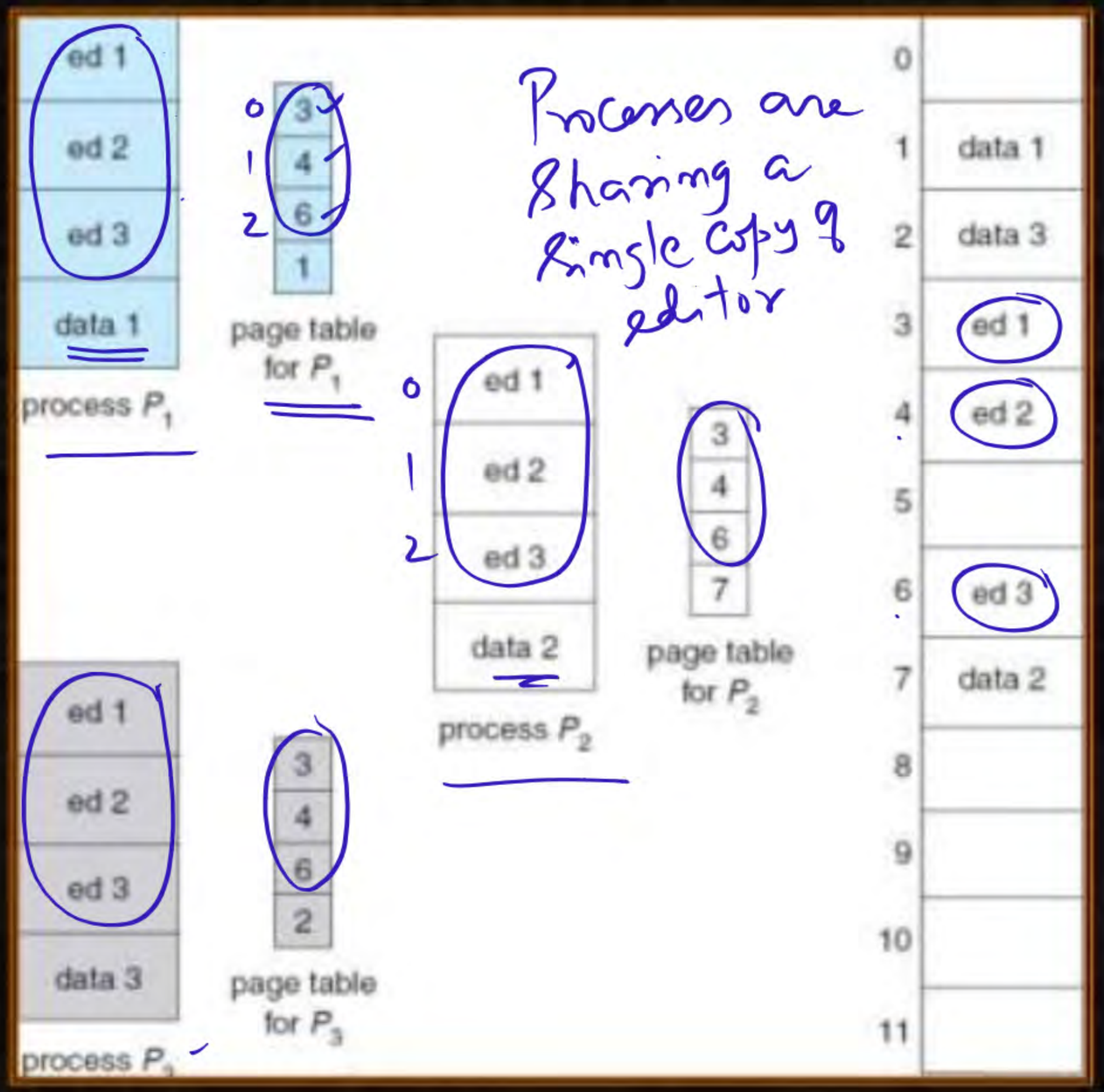
Many a times in a Multiprogrammed environment applications need to access common Routines / Programs like Editors & other System Programs

→ Shared by different 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



Monitors

Synchronization

- Inconsistency
- data Loss
- deadlocks

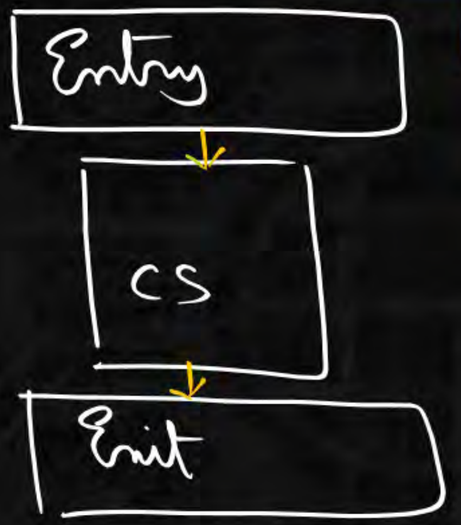
Nec Conditions:

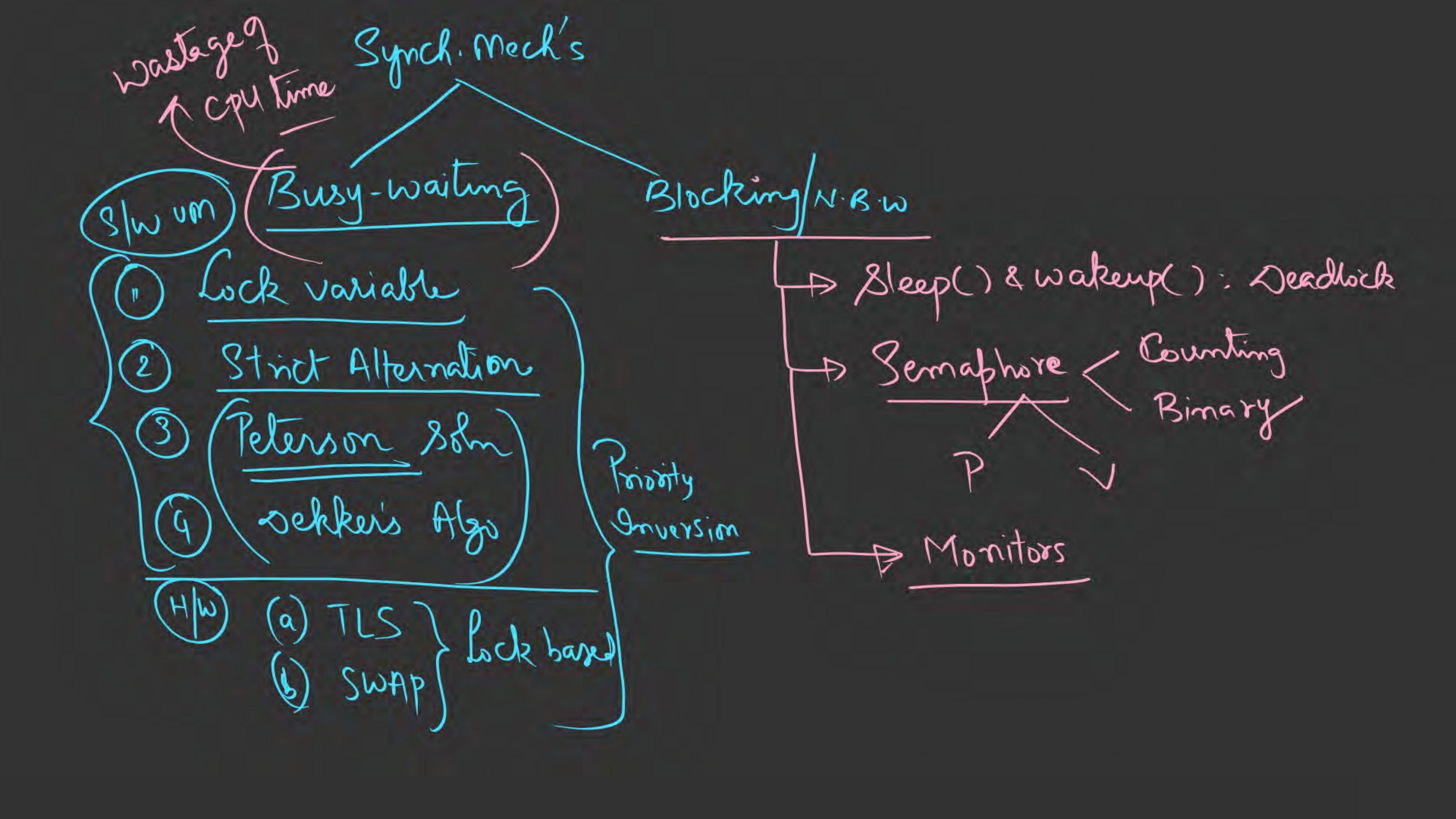
- Critical-Sec (Shared resources)
- Race Conditions
- Pre Emption

CS-Problem

Requirements

- Mut Exclusion
- Progress
- Boundedwait




Wastage of cpu time
Synch. Mech's
Busy-waiting
Blocking/N.B.W
S/w
Lock variable
Strict Alternation
Peterson Soln
Dekker's Algo
H/W
(a) TLS
(b) SWAP
Lock based
Priority Inversion
Sleep() & wakeup() : Deadlock
Semaphore
Counting
Binary
P
V
Monitors

Monitors:

↳ (B. Hansen)

: is a collection of Procedures, variables and data structures that are all grouped together in a special kind of Module | Package, <class = Abstraction + Encapsulation>

→ Procedures that run outside the monitor cannot access Monitor's internal variables and data structures, however they can activate only Member functions. <Private>

→ Monitor include special Condition variables, that support 'P' & 'V' opns;

→ Monitors have an important Property that only one Process will be active in the Monitor at any time;

```
monitor monitor name
{
   /* shared variable declarations */

   function P1 ( . . . ) {
      . . .
   }

   function P2 ( . . . ) {
      . . .
   }

         .
         .
         .

   function Pn ( . . . ) {
      . . .
   }

   initialization_code ( . . . ) {
      . . .
   }
}
```

Figure 5.15 Syntax of a monitor.

Constructor

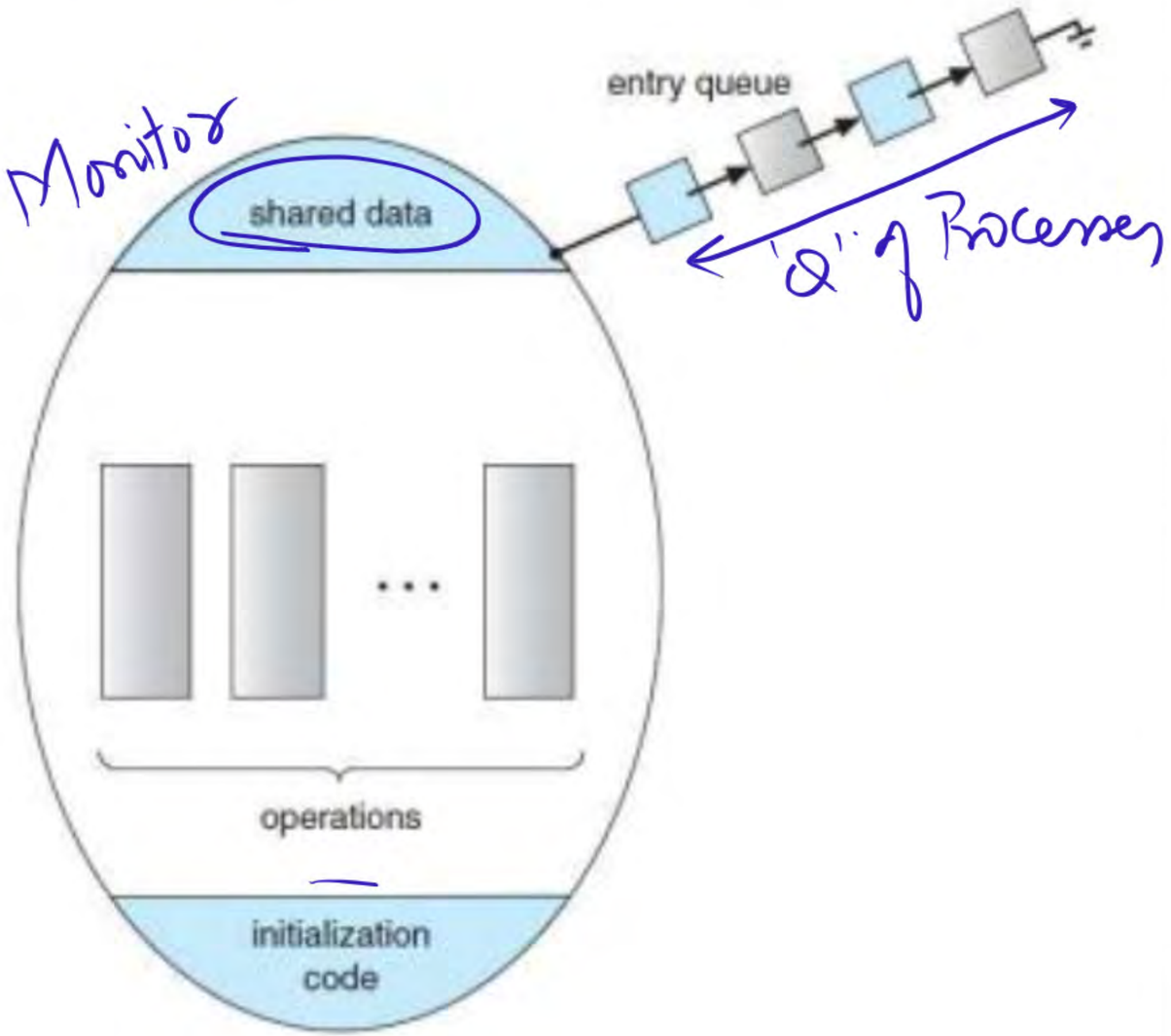


Figure 5.16 Schematic view of a monitor.

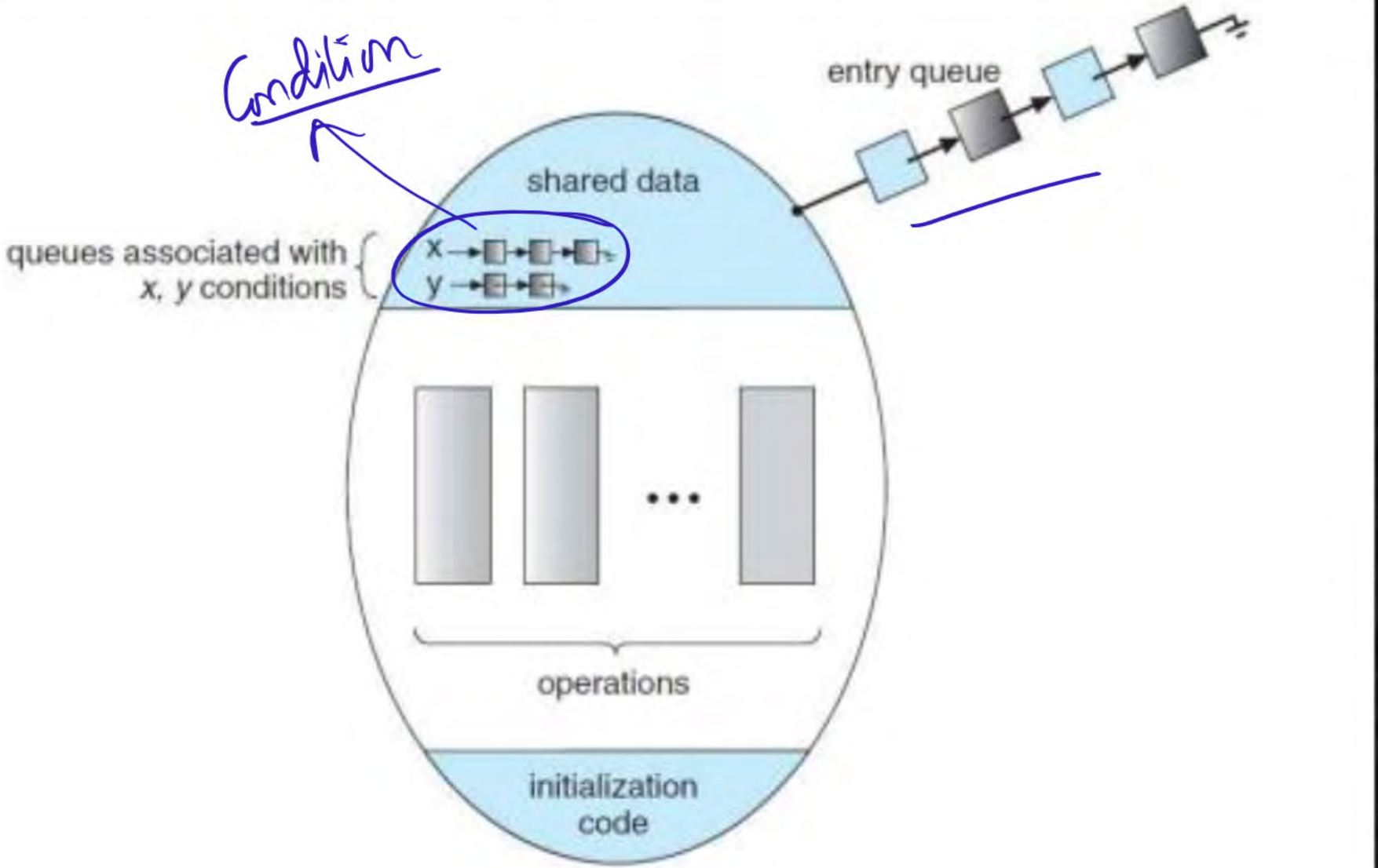


Figure 5.17 Monitor with condition variables.

```
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);
    test((i + 1) % 5);
  }

  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;
  }
}
```

Figure 5.18 A monitor solution to the dining-philosopher problem.

```
enum {THINKING, HUNGRY, EATING} state[5];
```

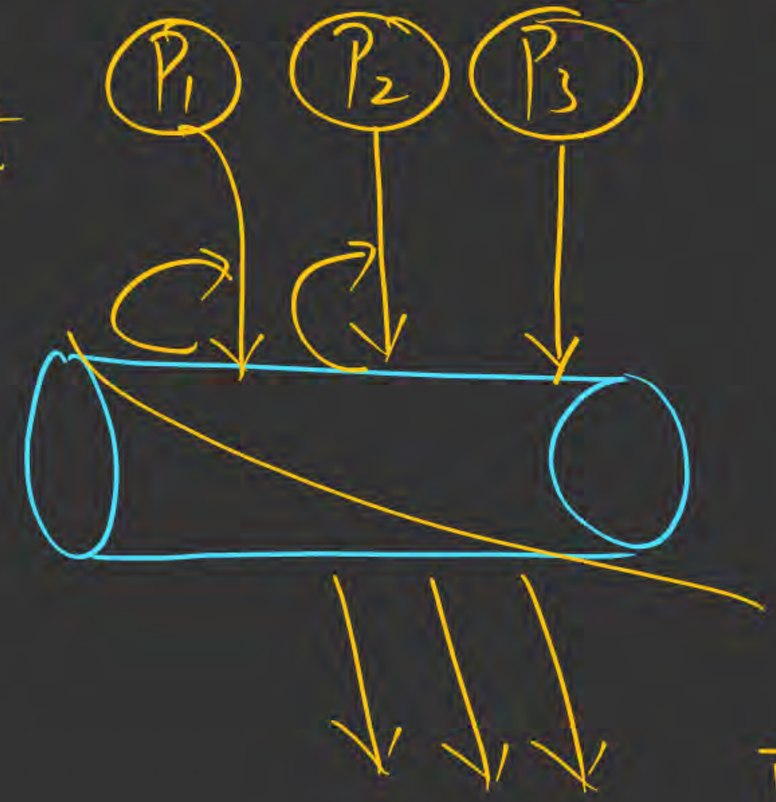
```
condition self[5];
```

Implementation of Dining Philosopher Problem using Monitors

Semaphore ~ Monitors

(Regular Revision)

Barrier Synchronization



Test Series
↳ (Time Mgmnt
↳ Syllabus Coverage)

40%
↳ Practice (PYQ's + ISRO + BARC + TIFR)
+ Exercises

(Recorded videos for unsolved (H/W) Questions of the Handout)

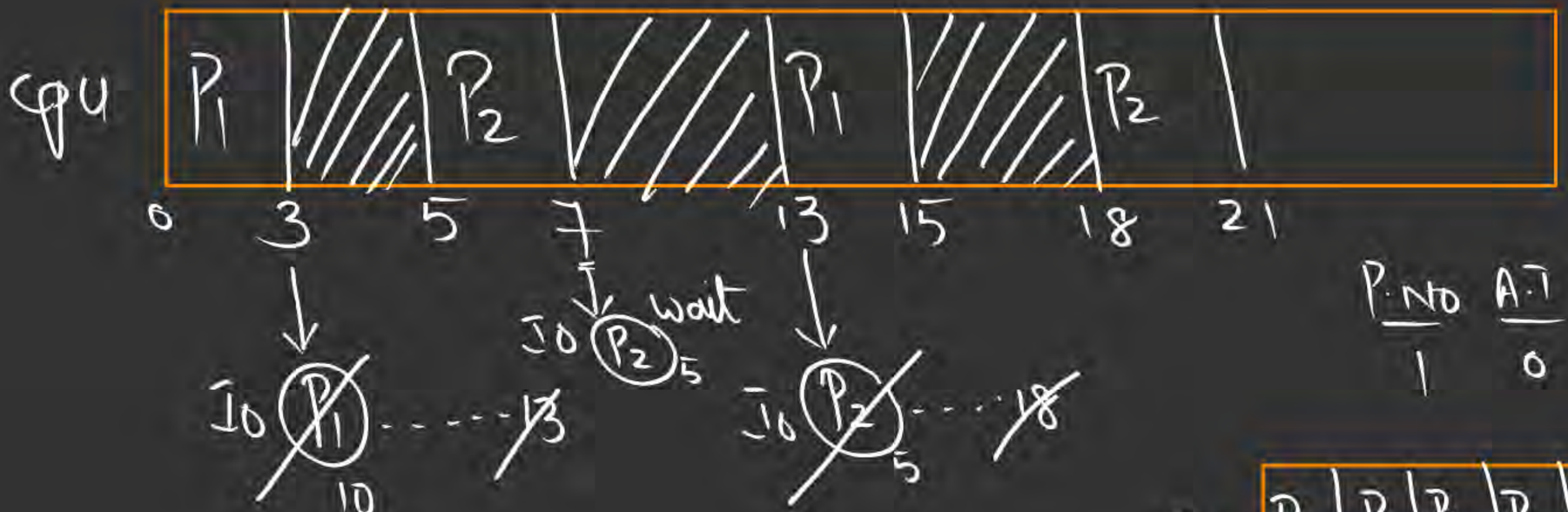
Platform
60% + 40%

FCFS with non-Conc IO

P.NO	A.T	<BT ; IOBT ; BT>
1	0	<3 ; 10 ; 2>
2	5	<2 ; 5 ; 3>

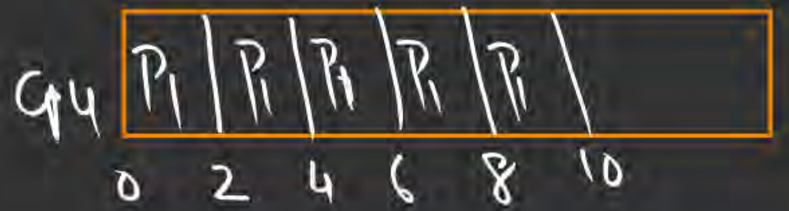
$TAT(P_1) = 15 - 0 = 15$

$TAT(P_2) = 21 - 5 = 16$



IO = 2

P.NO	A.T	BT
1	0	10




THANK YOU
GW
SOLDIERS !