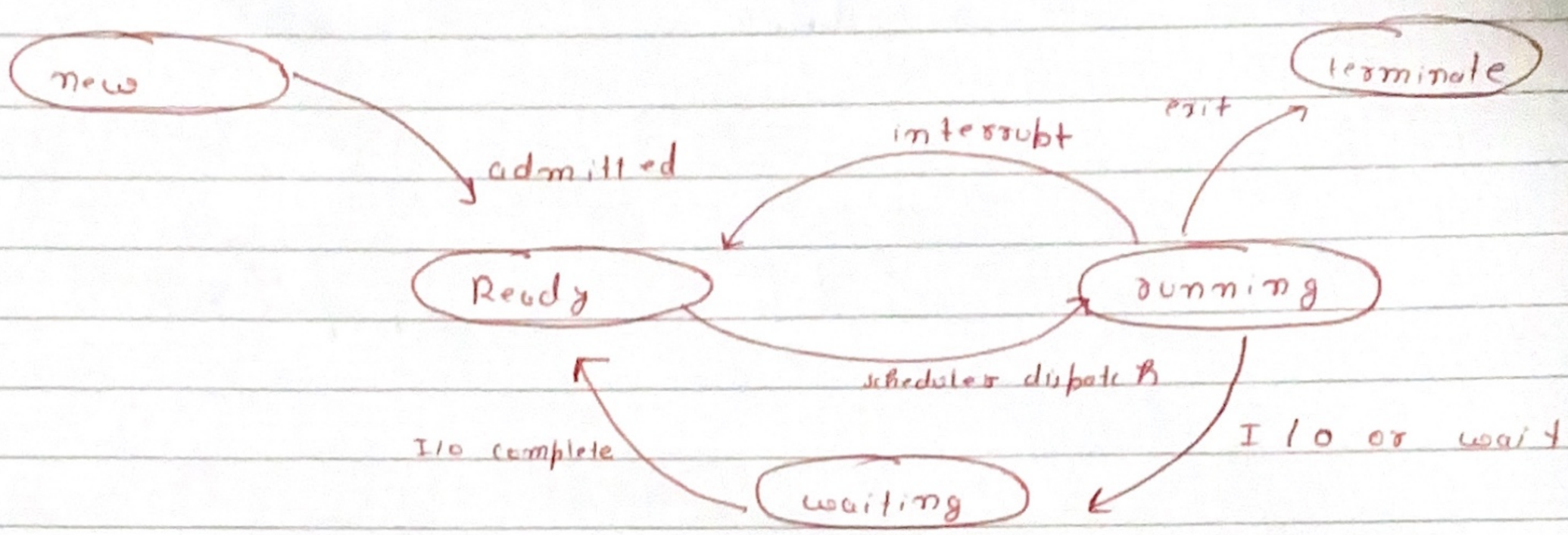
Diagram of Process State



PCB

Process State
Process Number
Program Counter
registers
memory limits
list of open files

Schedulers

Short Term Scheduler => ready to execute (fast)

Long Term Scheduler => Job pool area to ready

Medium Term Schedules → koi process preempt hogay toh uska PCB ka state store karna then again resume karna

Context Switch

C.P.U switches to another process System must save the state of the old process loaded the saved state for the new process via context switch.

I/O System Call

① int creat (char * filename, mode_t mode);

② int open (char * path, int flags, [int mode]);

③ close ( ) → pin to disable

④ read (int fd, void * buf, size_t cnt);

⑤ write (int fd, void * buf, size_t cnt);

dup ( ) → You know

Jo pin sabse last ki disable hoti hai usme updation karta hai.

New Copy of file descriptor

File descriptor

No. that uniquely identifies an open file in a computer OS, It describes data resource & how that resource may be accessed.

CLASSMATE
Date :
Page :

When a process makes a successful request to open a file the kernel returns a file descriptor which points to an entry in Kernel global file table.

Process Synchronization.

Why => Home ye toh pata hai ki har ek instruction ke baad preemption aa sakta hai toh naamlo koi ek program kisi hardware mein kuch change kar raha tha but just uske reflect hone se pehle preemption aa gaya aur uss hardware ko access karne lega = ?? (PROBLEM)

Critical Section => Ye vo part hai code ka jo ki access karta hai vo resource ko jisko ek baar mein ek hi process use kar sakta hai

3 key Requirements.

- Mutual Exclusion => Agar koi Process P_i is executing its critical section koi doosri problem allowed nahi hai ki vo apna critical section chala sake.

- Progress => Agr koi process hai jisne apna critical section khatam karliya hai & then kuch aur process hai jinhe vo section execute karna hai then they will Progress of critical section hote rahne chahiye

- Bounded Waiting => koi aisi process nahi honi chahiye jo wait Kartii hee rahe ki mein ab execute karoon Pur ussey mauka hee na mile. Aisa Nahi hona chahiye.

Section

Entry ⟶ Kya mein andar aa sakta hoon

Critical ⟶ Mein apna kaam karoo Akele mein

Exit ⟶ Mera kaam hogaya. Dooosra Koi AA sakta hoon

Remainder ⟶ Code jo ki critical section se related nahi hai

Peterson Solution

Two variable ⟶ flag [ ]
turn

flag ⟶ kaunsa process jaana chahta hai vo dekho

turn ⟶ kya uski baari hai

```
do {
    flag[i] = true;
    turn = j;
    while (turn == j and flag[j]);
        critical section
    flag[i] = false;  —— Exit
```

kyu Pj jaana chahta hai aur uski turn hai tab wait karo ⟵

While peterson Soln is a great way its not used oftenly kyonki No. of process jab badh jaati hai tab Complexity increase ho jaati hai;

Semaphore

Semaphore => Ab Ye eik aisa variable hai jo ki processes ke among share hota hai jisey synchronization hoti rahe.

2 operations => wait() and signal()

```
wait (S) {
    while (S <= 0);
    S--;
}

Signal (S) {
    S++;
}
```

→ Ye bee 2 functions hai

Semaphore
- → Counting Semaphore
- → Binary Semaphore

```
Semaphore mutex;
do {
    wait (mutex);
        // Critical Section
    signal (mutex);
        // Remainder Section
} while (TRUE);
```

Problem

Maanlo koi eb Process me P_1, wait logaya resource R1. pe aur P_2 bhi usko use karna chahti thee but Nahi kar paa rahi ; but P_2 is still in ready Queue vo CPU ki cycles waste kar rahi hai. ⟶ Effective Nahi hai;

Semaphore Sol with no busy waiting — :

Har eb Semaphore ke saath eb waiting Queue hoga jiska No aayega vo uthke ready ~~Queue~~ state mein jaeye. jo wait karega vo eb waiting Queue mein baitha rahe

- Operations => block => place the process invoking operation on appropriate waiting Queue.

  wakeup => remove one of process in waiting Queue and place it in ready Queue.

Deadlock

Eh aisi condition jahan be 2 ya jyada process aapas mein hee infinite waiting mein chale jaaye

wait (S); ⟶ (1)
wait (Q); ⟶ wait

PPL

wait (Q); ⟶ (2)
wait (S); ⟶ wait

Starvation => Kahin aisa na hojaye ke kisi process ko vo resource Kabhi mile hee nahi vo bhookhi baithe rahe

Priority Inversion => Scheduling problem when lower priority problem resource ko lock karde. Jabki uski jarooorat higher priority vaale

Synchronization Hardware

Uniprocessor => Agar koi Hardware be run kar raha hai toh vo execute karta rahega without breemption

Generally too inefficient on multiprocessor Systems ⟶ scalable Nahi hai

Test & Set Instruction

```
boolean TestAndSet (boolean *target)
{
    boolean rv = *target;
    *target = TRUE;
    return rv;
}
```

Agar false return - ⌣
true return - busy.

pehle kya tha 'rv' mein store Ab usey true kiya Agar pehle false hoga Yani resource Abhi free hai return false process use karega & target = TRUE;

Bounded Buffer Problem

Producer mein value produce karega ek buffer mein. Then consumer consume karega buffer ka size hai N →

buffer is shared blw process

↓

mutex

Producer

```
do {
    wait (empty);
    wait (mutex);
        // add item to buffer
    Signal (mutex);
    Signal (full);
} while(True)
```

Consumer

```
do {
    wait (full);
    wait (mutex)
        // remove
    signal (mutex)
    signal (Empty)
} while (True);
```

Dissey ek samay pe ek hee buffer mein dool ya nikal sakte

ek jagah bhardi

ek jagah khali

Empty => kitna khali hai;

full => kitna bhara hai;

Readers- Writers Problem

Readers ⟶ kitne bhi read kar sakte hai

Writers ⟶ ek samay pe sirf ek hee write kar sakta hai

Writer Process

```
do {
        wait (wrt);
            // Writing is performed
        signal (wrt);
} while (True)
```

Readers Process

```
do {
    wait (mutex);
    readcount ++;
    if (readcount == 1)
        wait (wrt);          ] ⟶ Agar pehla reader hai toh check karo koi in writers likh toh nahi raha.
    Signal(mutex)
        // Read karo
    wait (mutex);
    readcount --;
    if (read count == 0);
        signal(wrt);         ] ⟶ Readers khatam toh writer ko signal dedo
    Signal (mutex);
} while (True);
```

Dining - Philosphers

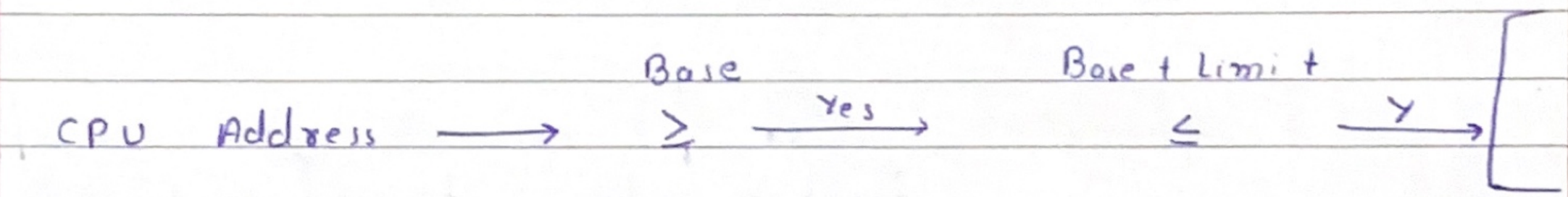
```
do {
    wait ( chopstick [i] );
    wait ( chopstick [(i+1) % 5] );
    // Eat
    signal ( chopstick [i] );
    signal ( chopstick [(i+1) % 5] );
} while (TRUE);
```

Race Condition => You Knows.

Memory Management

Base & Limit Registers => Base & Limit Register protects the processes from kahin vo doosri process ka code or data access na karle.

Jab bhi koi CPU Address Generate hota hai toh vo pehle check hota hai ki vo bada ho Base se & small ho base + limit se.



Address binding of Instructions

- Compile Time
- Load Time
- Execution Time.

Straight to Important topics.

Logical Address => CPU Generated
Physical Address => Physical Memory Generated

Memory Management Unit → CPU Generate karta hai virtual Address. usko Physical memory mein map karta hai vo.

User program's deals with logical addresses.

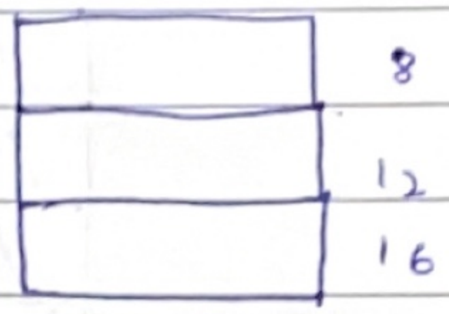
Swapping => A process can be swapped temporarily out of memory out to a backing store, and then brought back into memory for continued execution.

Process ki physical space ki jo demand hoti hai toh ek baar mein saari demand satisfy nahi ho sakti.

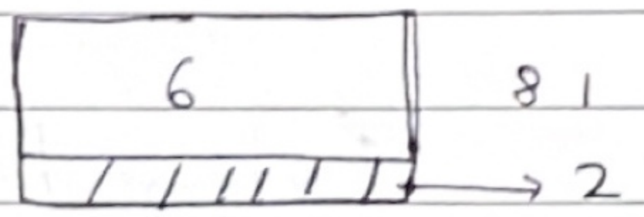
lets see ki Jo Main Memory mein kaise divide hoti hai

Contiugous

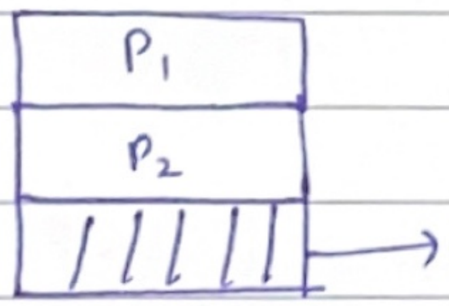
Fixed Partition



- Partions are fixed → Easy to Implement
- > Internal Fragmentation =>



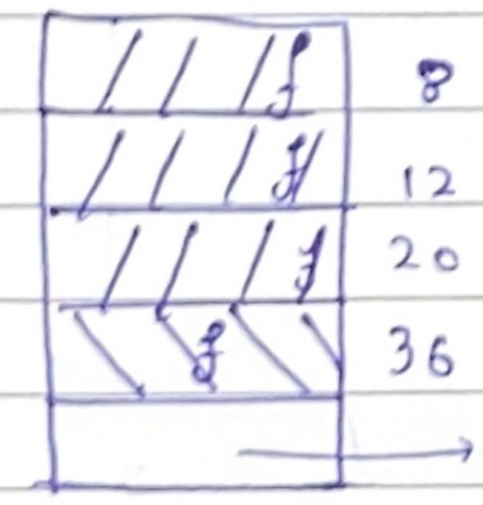
- > External Fragmentation => Total unused space



CLASSMATE
Date :
Page :

Variable Partioning

Isme kya hota hai process aati hai har unke demand ki according unhe space allocate hoti hai



: External Fragmentation - ✓
Internal Fragmentation - X

Dynamic Storage Allocation

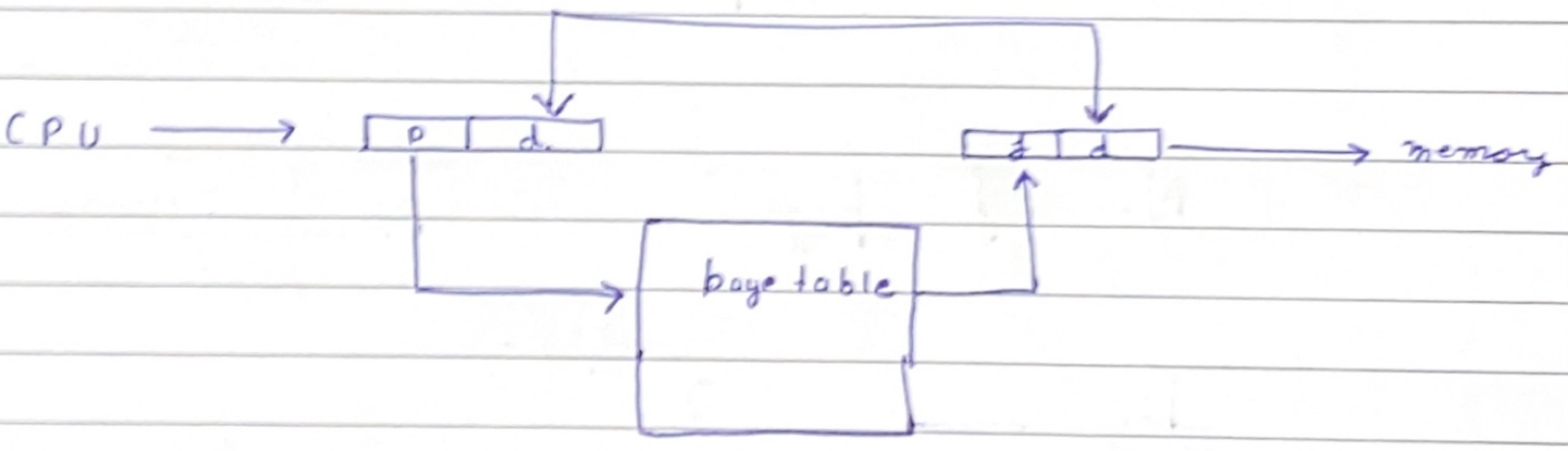
First Fit — Allocate the first hole that is big Enough

Best Fit — Allocate Smallest that it can have

Worst — Allocate largest hole.

Non Contigous

Paging



Logical Address => p + d
pageno offset

Numericals

Page Size = Frame Size

Logical Physical Address bits = N

Physical / logical Memory Space = 2^N

2^{10} = 1 KB 2^{20} = 1 MB

2^{30} = 1 GB

No° of pages = $\frac{\text{logical Memory Space}}{\text{page size}}$

Page Size = 2^{offset}

Size of page table = No° of Entries x Bits req to Address frames.

Ab Ye Saare pages, page table Ye sab store hai main memory mein toh kuch pages Jo frequently aate hain unko hum eh TLB mein daalde hain kyonki Search time kum lagta hai

Effective Access Time

α Hit ratio = (TLB mein milne ka chance)

ε a ns => Time to Search TLB
b ns = Time to Search in memory.

→ Agar TLB Mila + Memory page ka frame dhoondo.

→ Nahi mila toh pehle pagetable memory mein dhoondo then uss page ko dhoondo

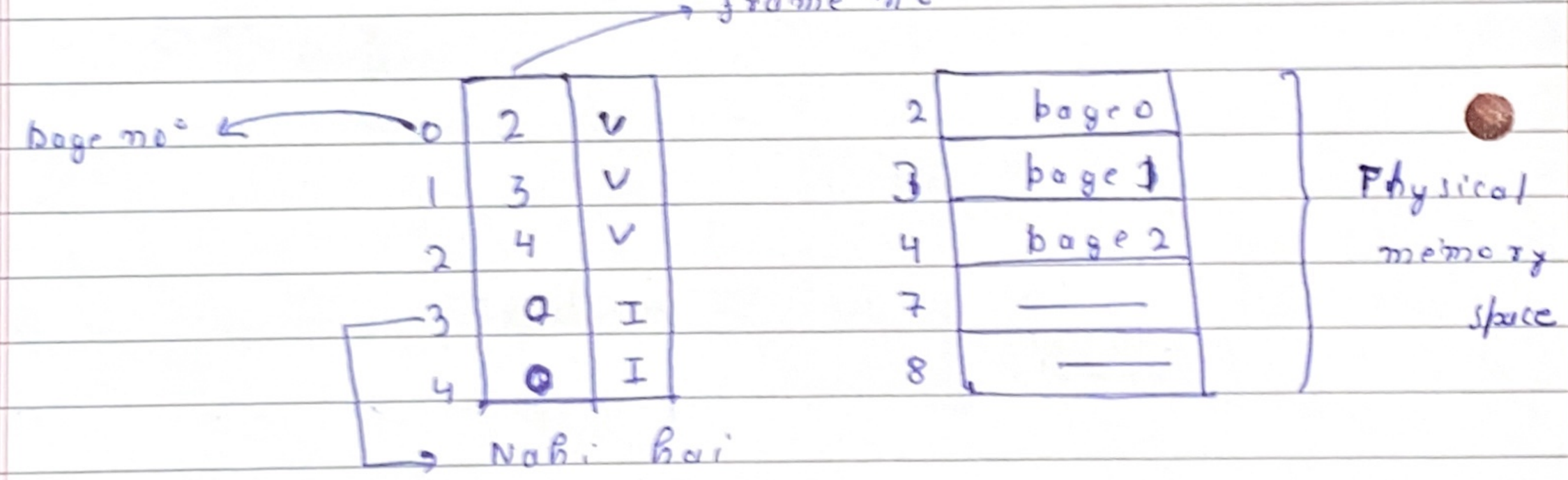
$$EAT = \alpha (a + b) + (1 - \alpha)(a + 2b)$$

Memory Protection

Har ek process ki apni page table hoti hai aur ab frames mein hum kuch bits assign karte hai jisse pata chale ki vo read-only hai sirf. Aur read-write access ho.

Valid - Invalid => ki jo CPU Generated page hai vo Exists karta hai ki nahi

Page Table



Shared Pages

Suppose ek page hai jo ki sirf readonly hai toh lo process ke liye uski lo copy banegi Ye hai wastage.

toh aise pages share ho jaate hai

page no° → ed1 3 ← frame no°
ed2 4
data 6

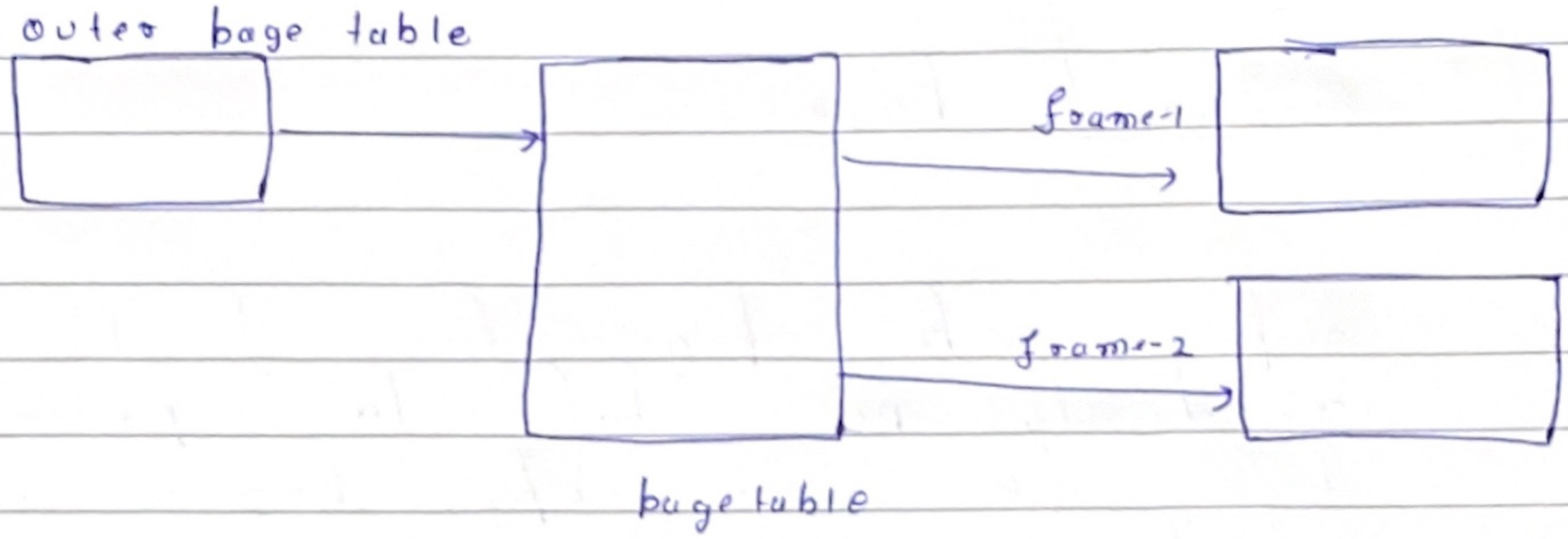
P1

P2
ed 1 3
ed 2 4
data 2

Multilevel paging

See pagetable => Size must be equal to frame Size because Store hona toh frame mein hee hai.

But aisa nahi hai toh.



outer page table batayegi ki page range jo hai 0 - 100 vo kahan rak°

Aise Leveling hoti hai

2 - Level

Ques = Physical Memory Space = 8 GB
Page Size = 8 KB
logical bit = 46 - bit
page tabel Entry Size = 4B