Sunbeam Institute of Information Technology
Pune and Karad

Module - Concepts of Operating System

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

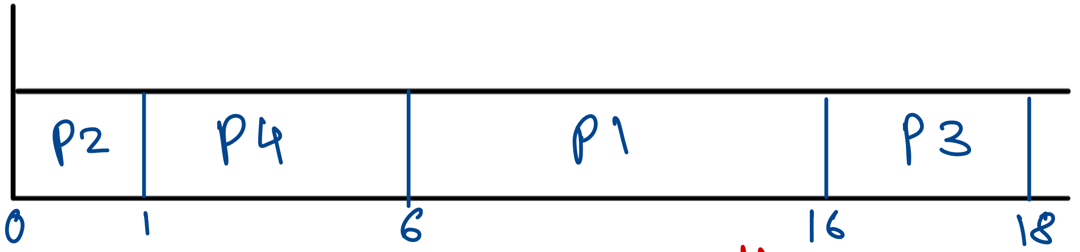
Priority

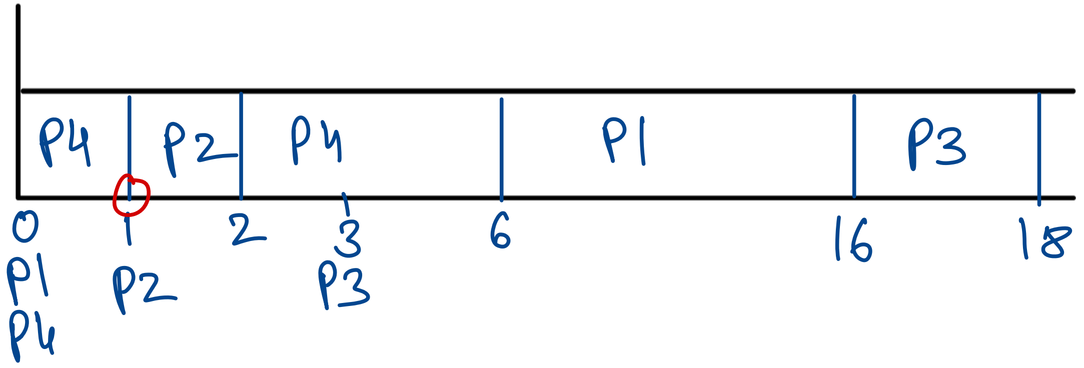
(Non preemptive)

Process	Arrival	CPU Burst	Priority	WT	RT	TAT
P1	0	10	3	6	6	16
P2	0	1	1 (H)	0	0	1
P3	0	2	4 (L)	16	16	18
P4	0	5	2	1	1	6

(Preemptive)

Process	Arrival	CPU Burst	Priority
P1	0	10	3
P2	1	1	1 (H)
P3	3	2	4 (L)
P4	0	5	2





P1 (6)
P2 (8)
P3 (6)
P4 (5)
P5 (7)
P6 (7)

P1
P4
P3
P5
P6
P2

starvation:
due to low priority, process is not getting enough time to execute on CPU

Aging:
priority of process is increased gradually.

RR (Round Robin) (pre emptive)

Process	CPU Burst
P1	53
P2	17
P3	68
P4	24

33, 13 ✗

✗

48, 28, 8

4 ✗

$\underline{0}$ + 57 + 24

$\underline{20}$

$\underline{37}$ + 40 + 17

$\underline{57}$ + 40

Time Quantum: CPU time slice

TQ = 20

TQ = 100

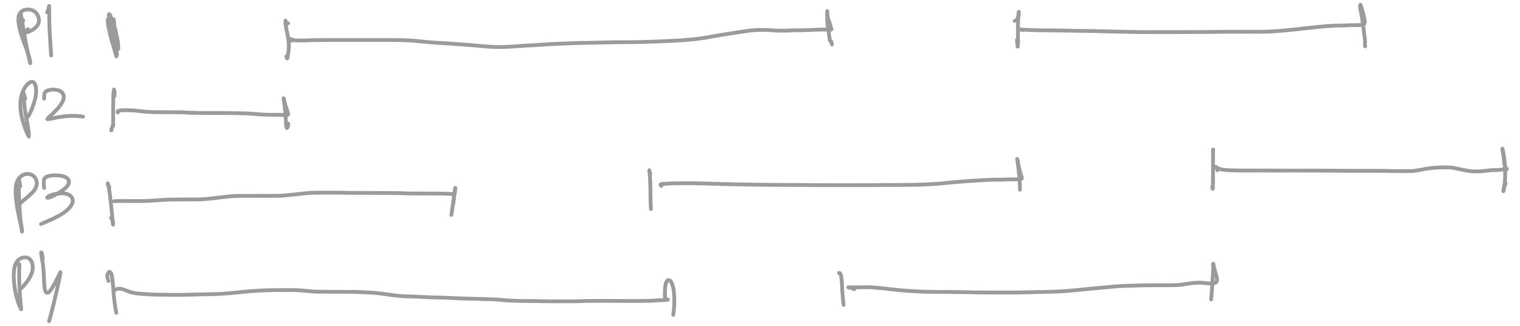
↳ behave like FCFS

TQ = 4

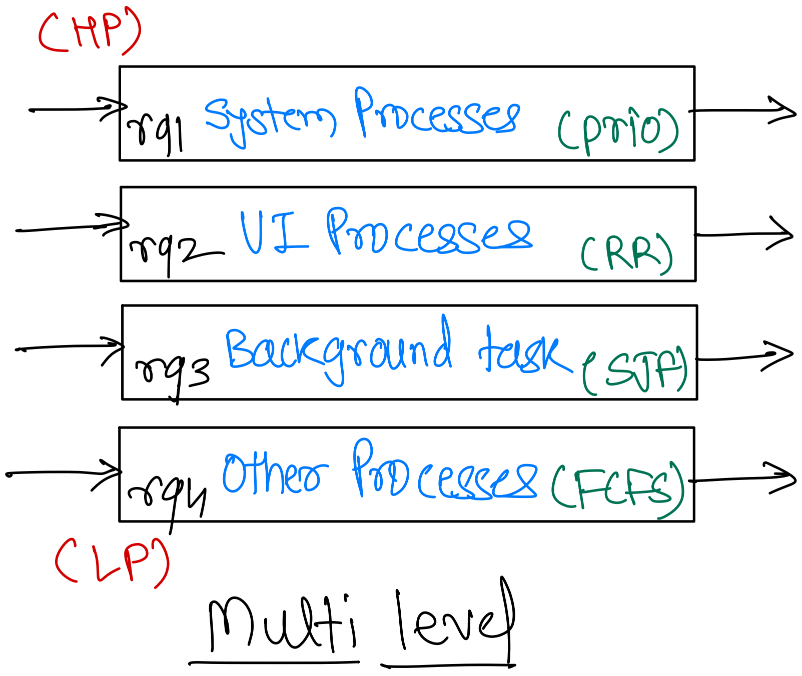
↳ CPU overhead will increase

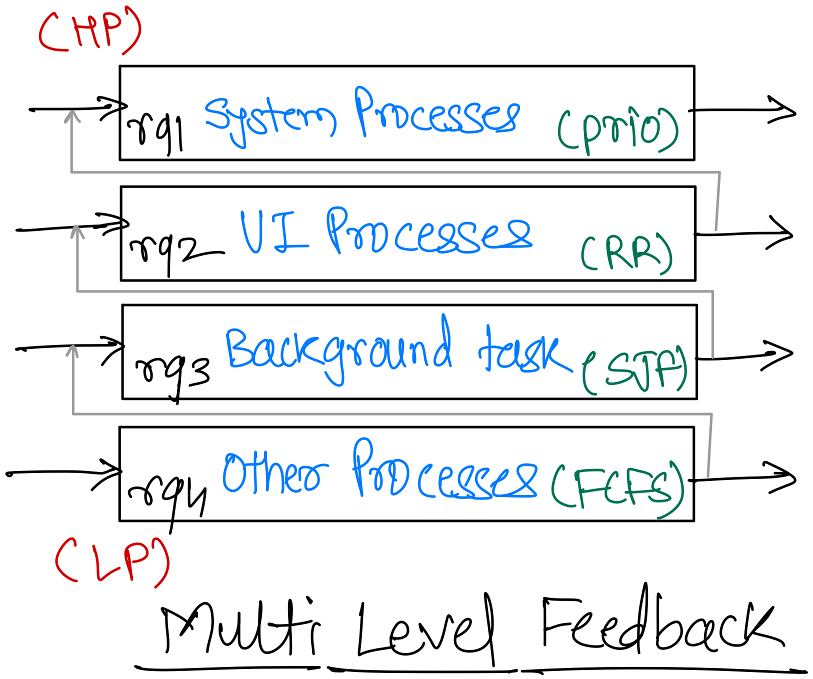
P1	P2	P3	P4	P1	P3	P4	P1	P3	P3

0 20 37 57 77 97 117 121 134 154 162



Multi Level Ready Queue





fork( ) — to create a child process

- child process is created by duplicating parent process (calling process)

Orphan process

- process who's parent terminates before it

init/systemd - process (pid=1)

defunct / zombie process

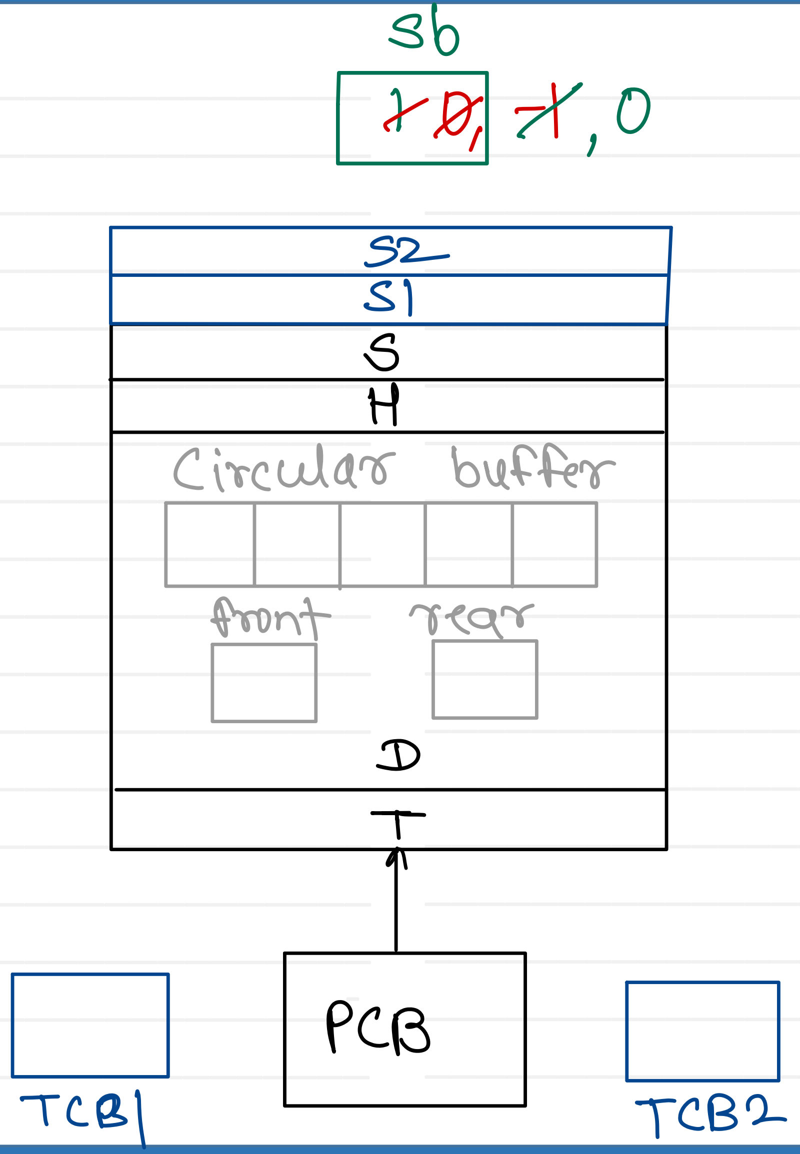
- process who terminates before its parent.

wait( ) / waitpid( )

exec — execl, execv, execvp, execlp, execve

- load program from harddisk to RAM in newly created process.

Producer - Consumer




```
Thread1( ) {
    while(1) {
        P(sb)
        buf[rear] = x;
        V(sb)
    }
}
```

```
Thread2( ) {
    while(1) {
        P(sb)
        D = buf[front];
        V(sb)
    }
}
```

Semaphore

- internally it is a counter

1. Dec / wait( ) / P( )

before accessing the resource

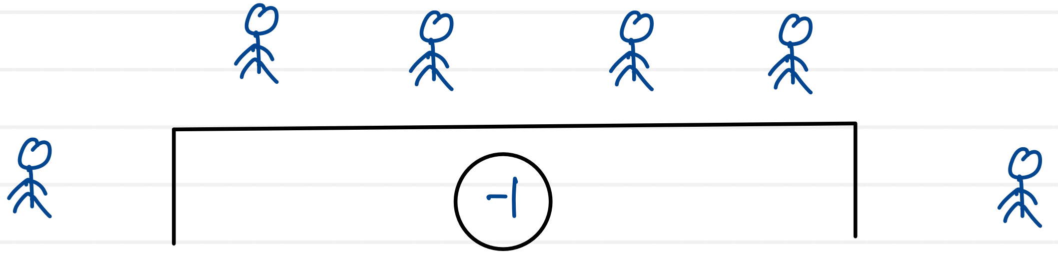
   - decrement counter
   - if counter <0 then block the current process.

2. Inc / post( ) / V( )

after releasing the resource

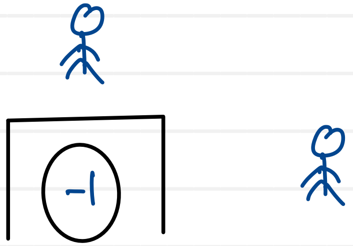
   - increment counter
   - if someone is blocked on this semaphore wakeup it.

1. Counting Semaphore



- count no. of available resources
- count no. of processes waiting for it.

2. Binary semaphore



- only one will use resoure at a time

Mutex

Mutex = Mutual Exclusion
(one at a time)

- mutex is same like binary semaphore
- operations – lock / unlock
- process who locks the mutex becomes owner of mutex.
- only owner can unlock the mutex.

- infinite waiting for a resource
- deadlock occurs only when below four conditions hold true at a time
  1. Mutual Exclusion
  2. No preemption
  3. Hold & wait
  4. Circular wait

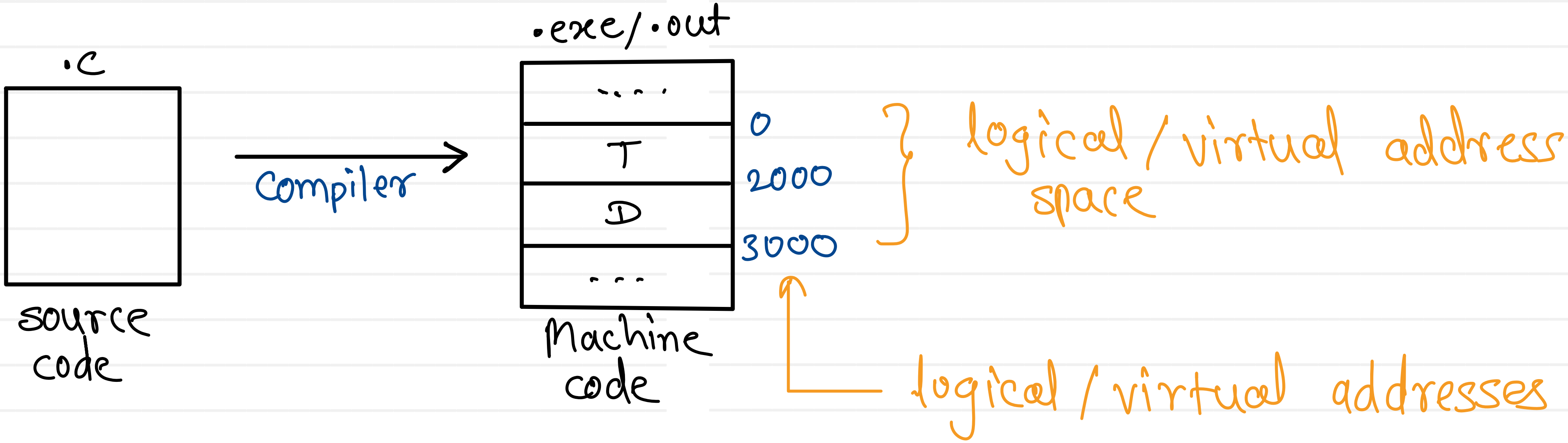
Prevention :
while implementing OS, it is always ensured that 1/4 condition will hold false.

Avoidance :
1. Banker's algorithm
2. Resourse allocation graph
3. Safe state algorithm

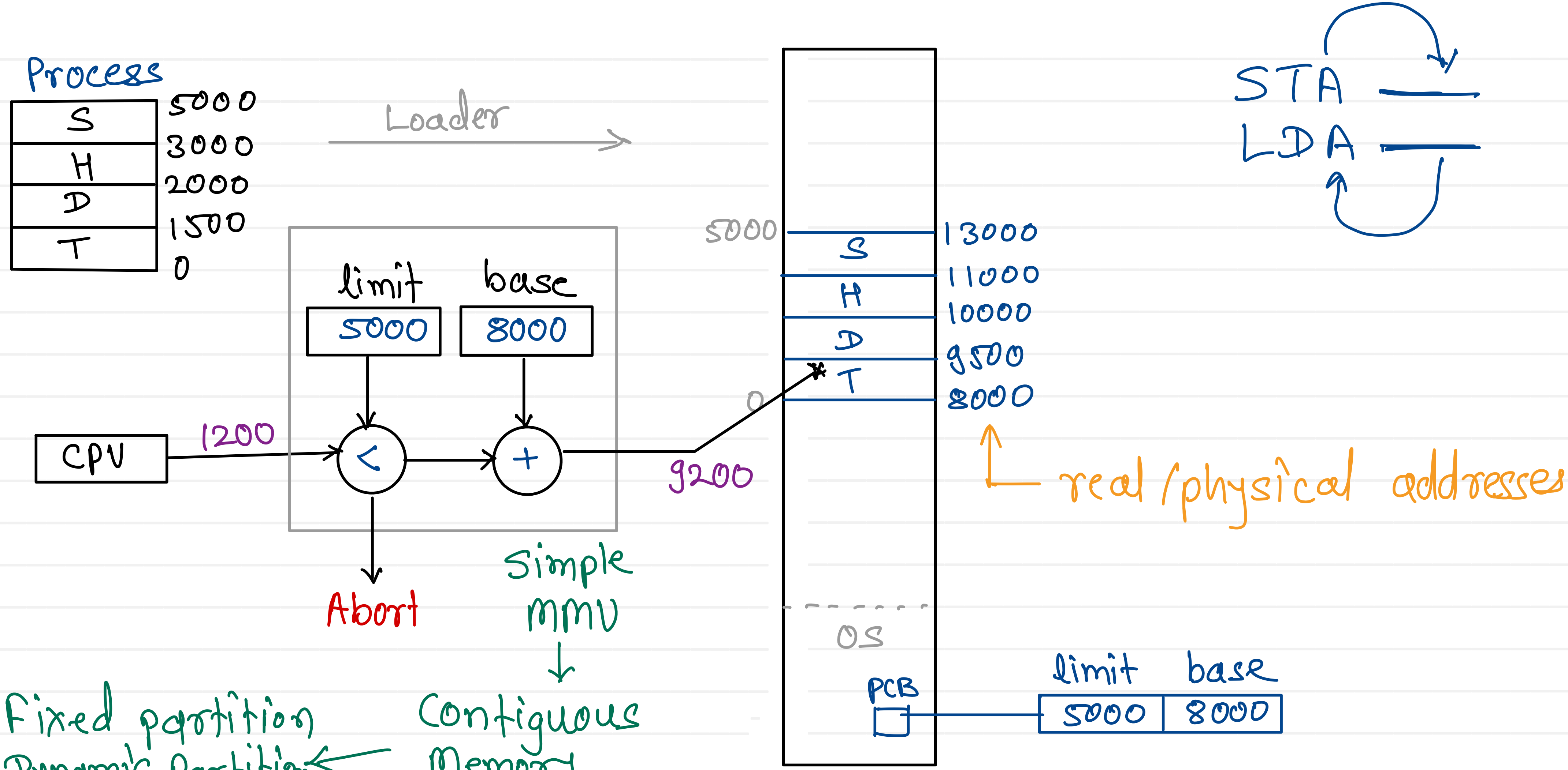
Recover :
1. resource preemption
2. forceful termination of process

Memory Management

- compiler always assigns logical / virtual / imaginary addresses to the functions are variables



Simple MMU

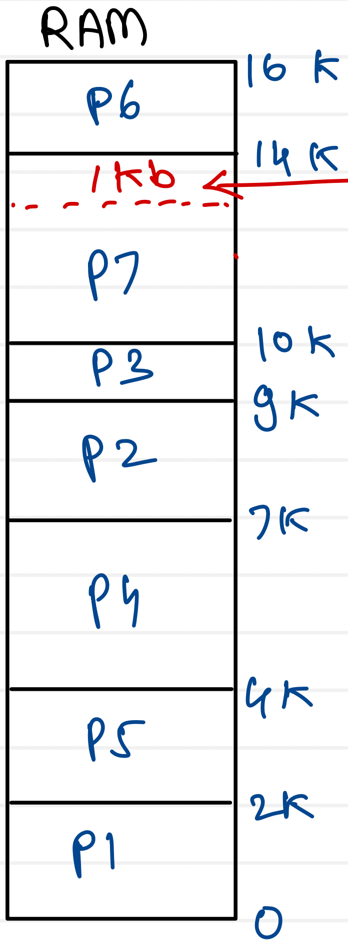


Simple MMU
↓
Contiguous Memory allocation

1. Fixed partition
2. Dynamic Partition

Contiguous memory allocation

Fixed partition



internal fragmentation

if process is not utilizing whole allocated partition, some part of that memory will be wasted.

limitations :

1. Max process size is size of maximum partition
2. Max no. of processes are no. of partitions

Dynamic Partition

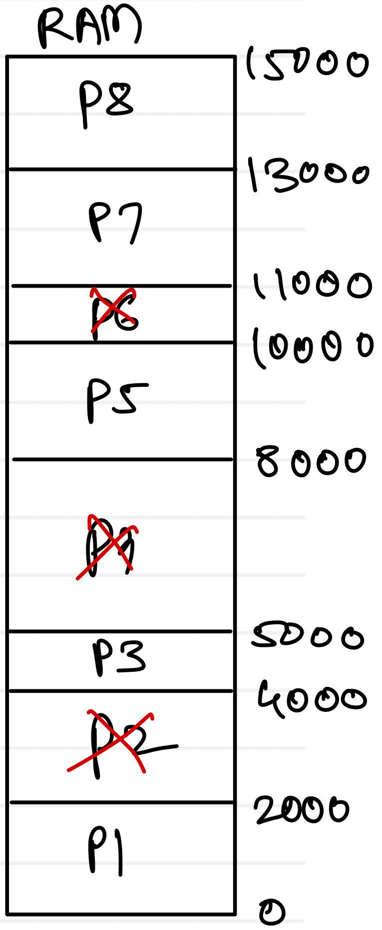
Free slot table

P9 — first fit / best fit / Worst fit

limit	base
2000	2000
1000	10000
3000	5000

950 ✓

4000 ✗



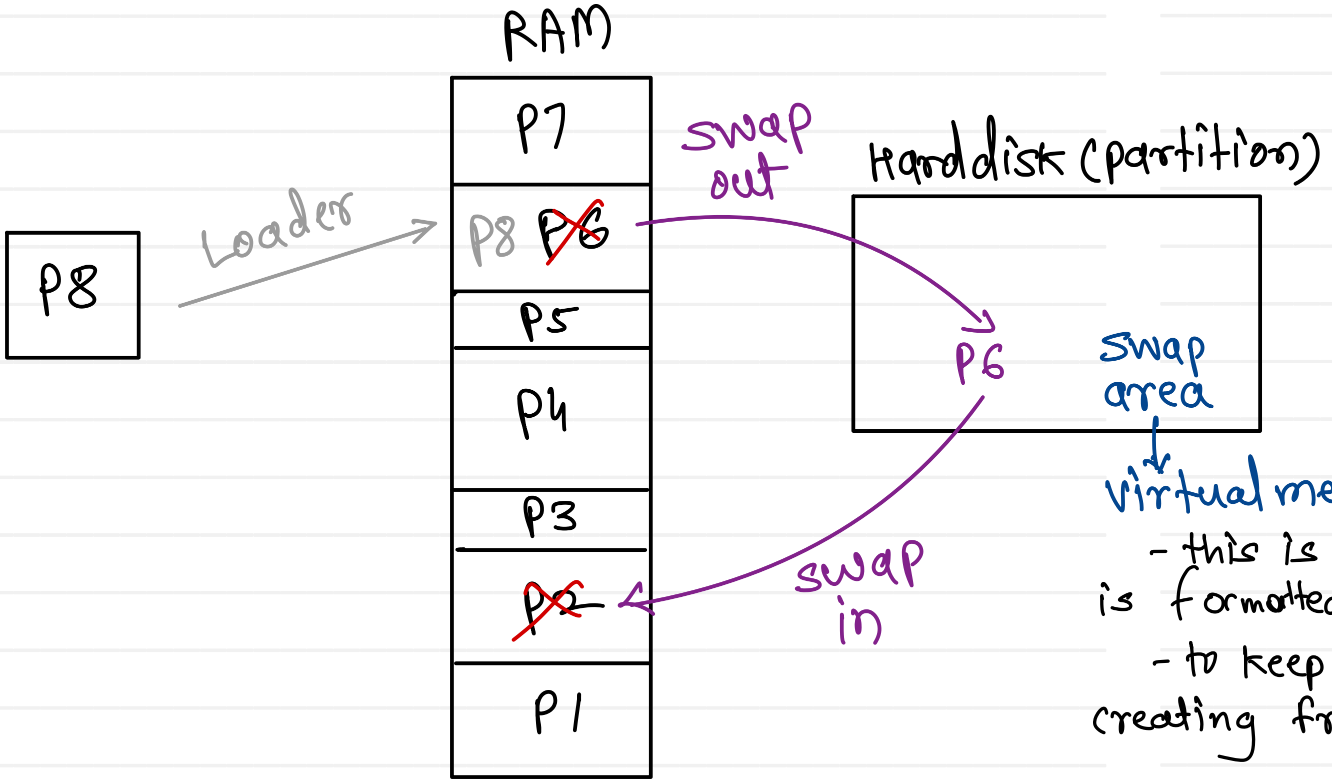
external fragmentation

due to unavailable large contiguous free space, we can not load process inside RAM

compaction :

moving processes inside RAM to create contiguous free space.

Virtual memory

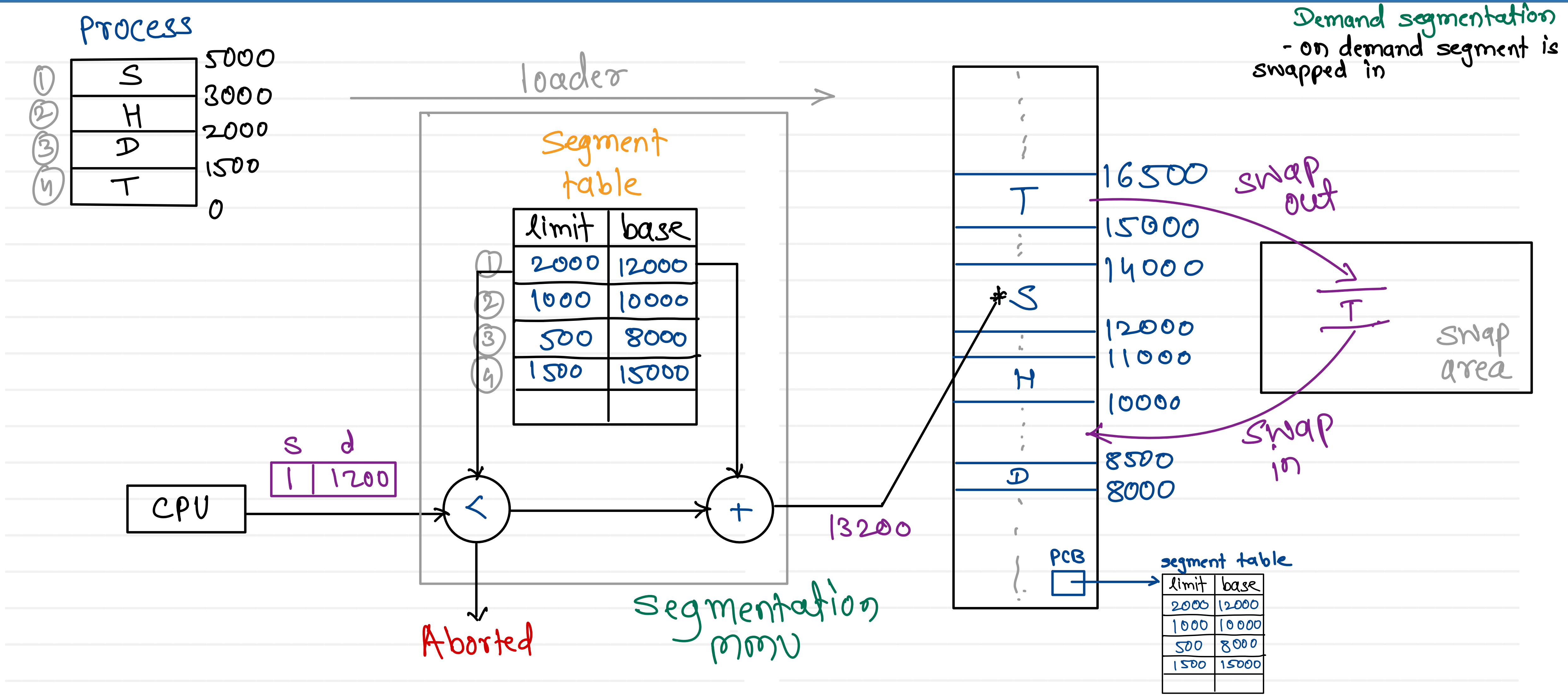


↓
virtual memory
- this is not actual RAM it is formatted like RAM.
- to keep inactive processes for creating free space inside RAM

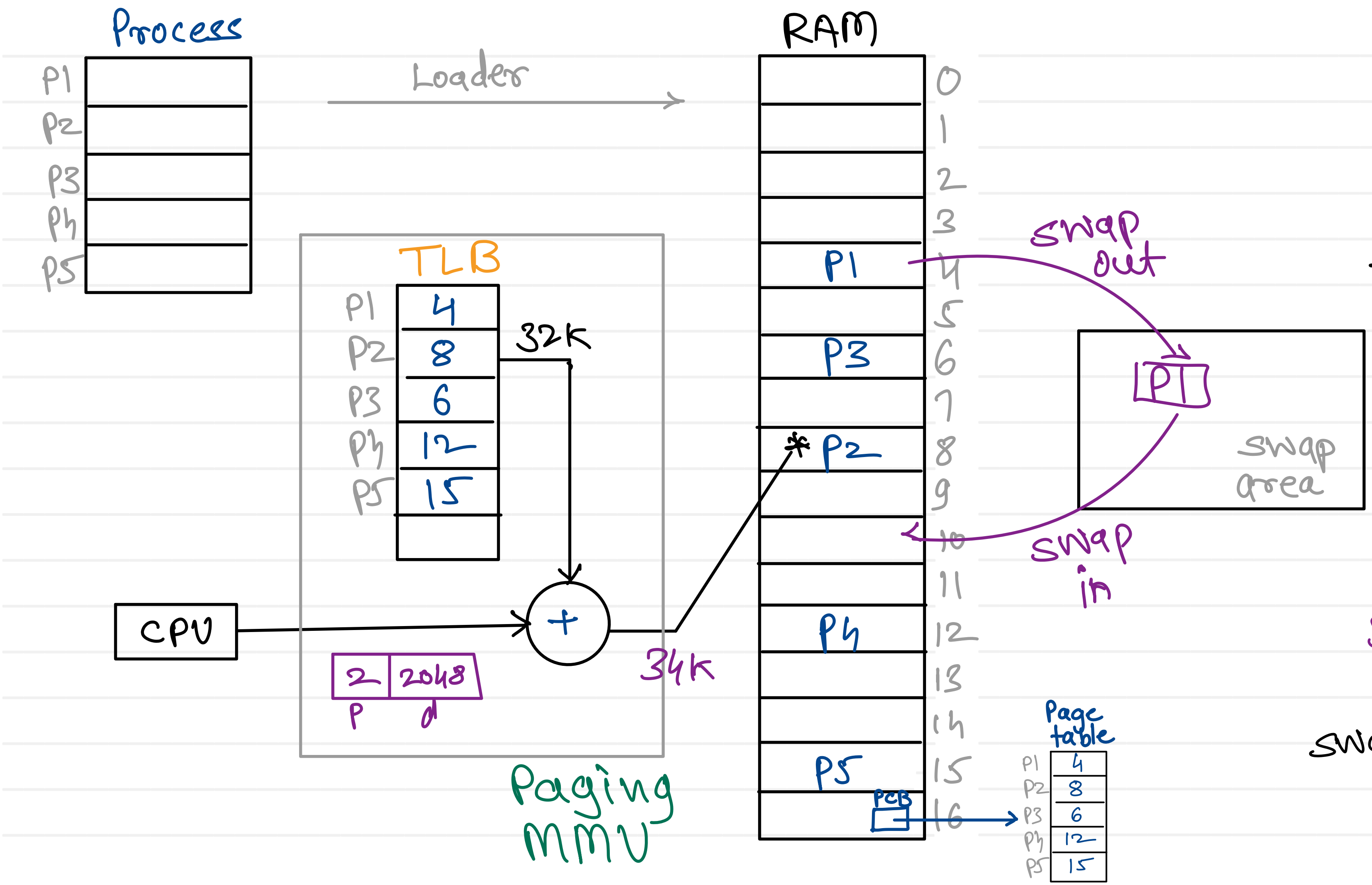
swap area
- partition → linux
- file → windows (pagefile.sys)

size = 2 * RAM size

Segmentation MMU



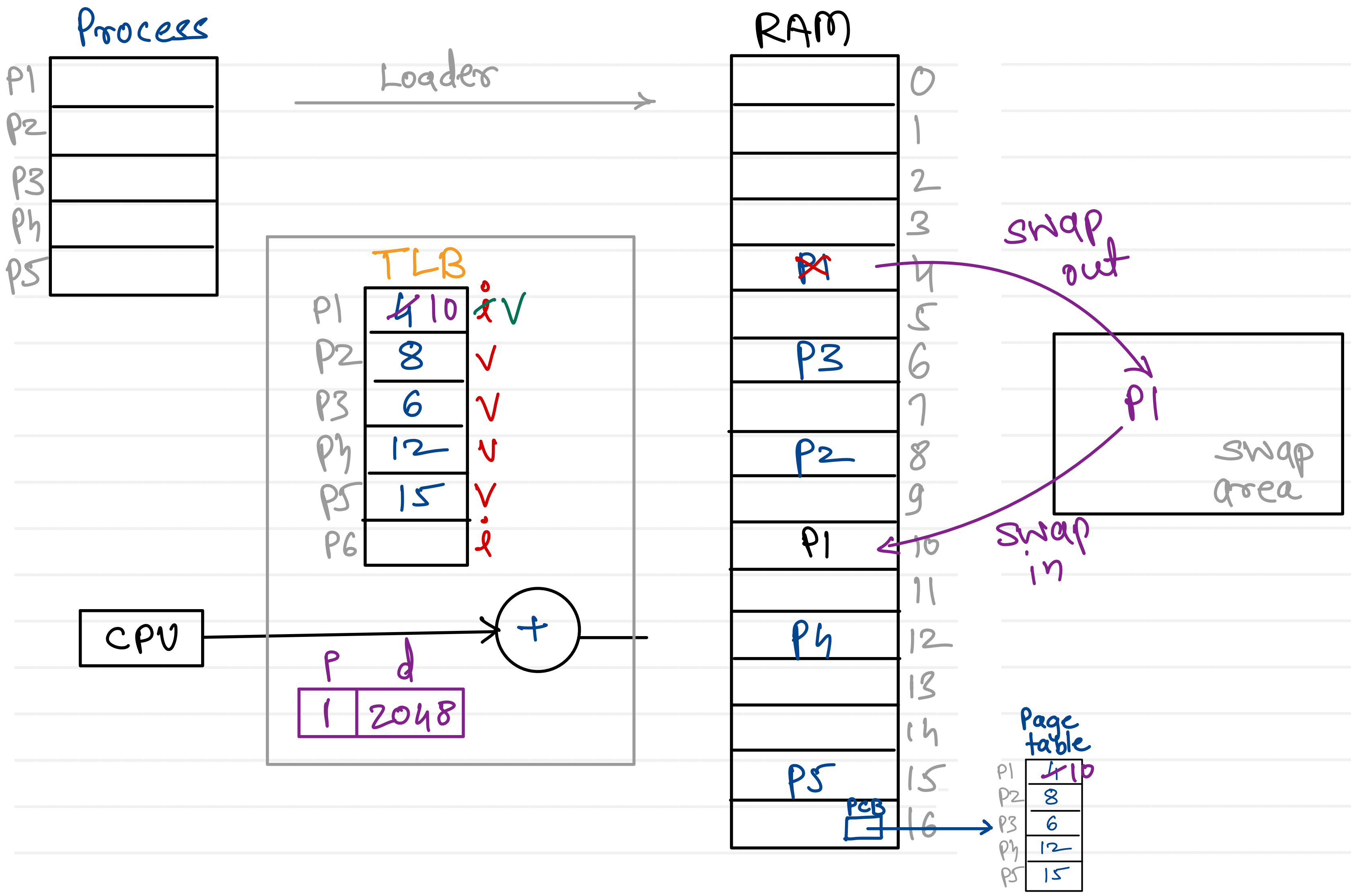
Paging MMU



- RAM is divided into fixed equal size partitions
  size = 4kb (4096 bytes)
  "frame"/"physical page"
- Process is also divided into partitions of size equal to frame size.
  "page"/"logical page"

Demand paging:
On demand page is swapped in memory

Page fault



Page fault :
Whenever CPU request for the address of some invalid entry of page table, this fault is generated.
on every page fault, page fault handler of OS is called

pagefault_handler() {
1. validate the address
2. check for read/write perm
3. find free frame in RAM
4. swap in page into free frame
5. update mapping in page table and TLB.
6. re execute the command for which page was occured
}

Thrashing: frequent swap in & swap out of pages
solution: increase the size

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com