Sunbeam Institute of Information Technology
Pune and Karad

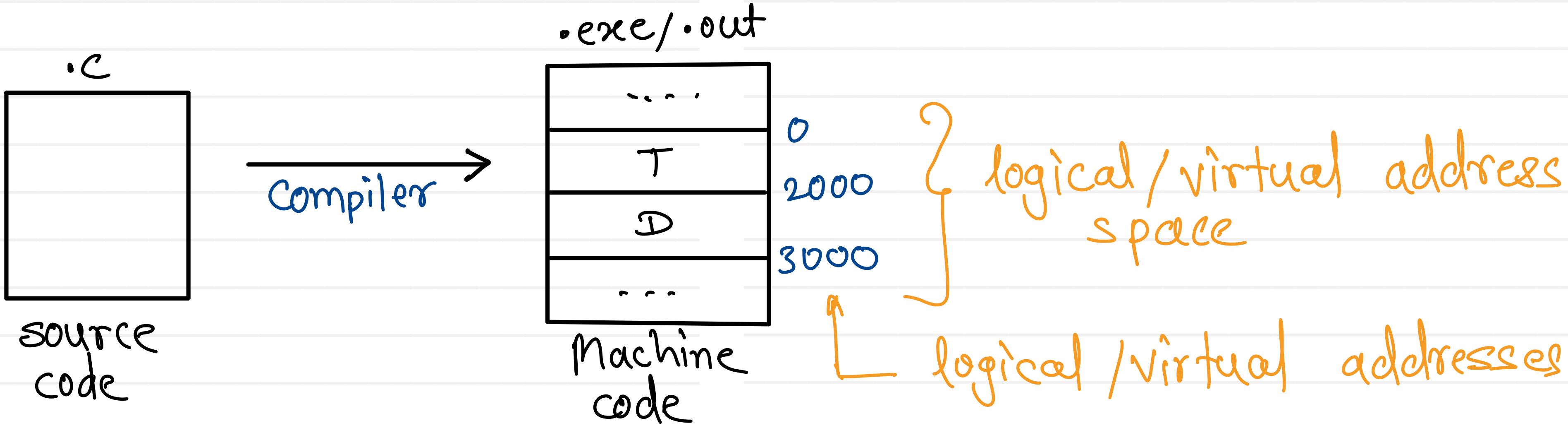
Module - Concepts of Operating System

Trainer - Devendra Dhande

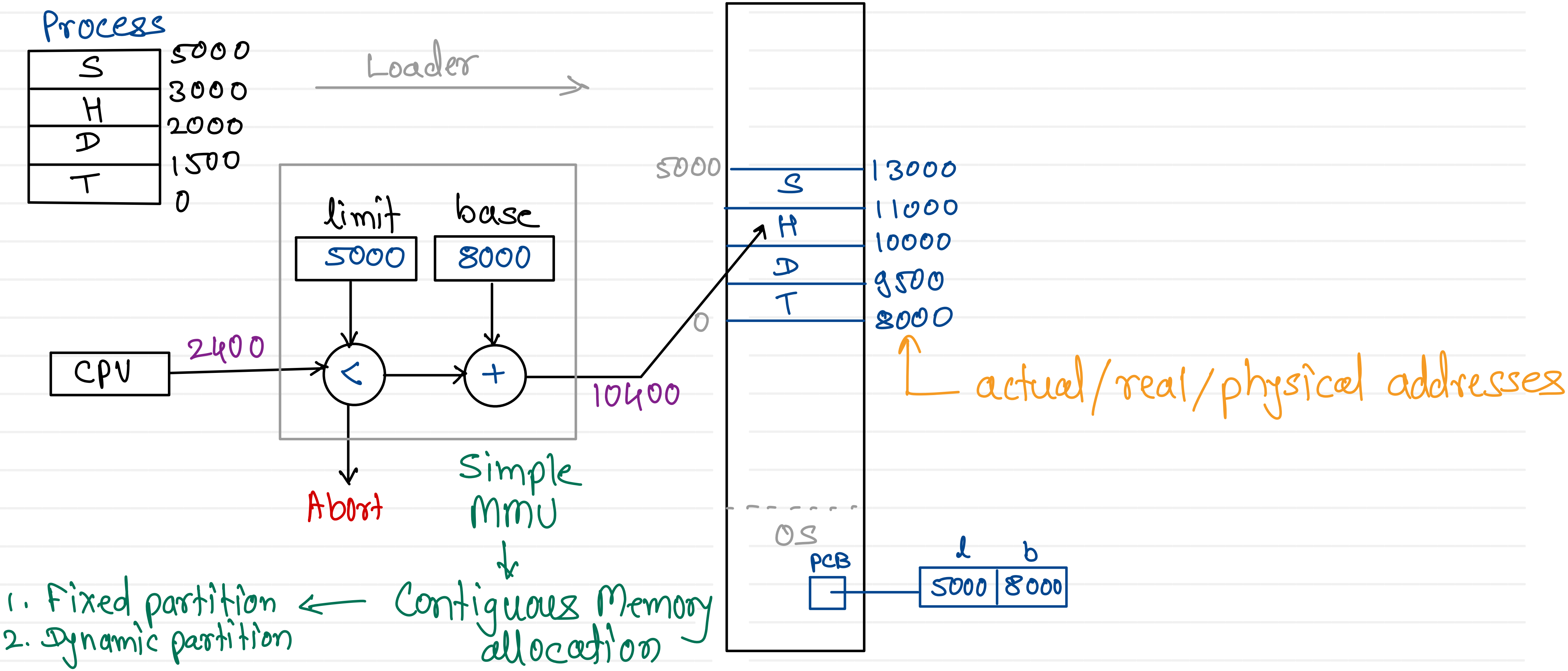
Email – devendra.dhande@sunbeaminfo.com

Memory Management

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

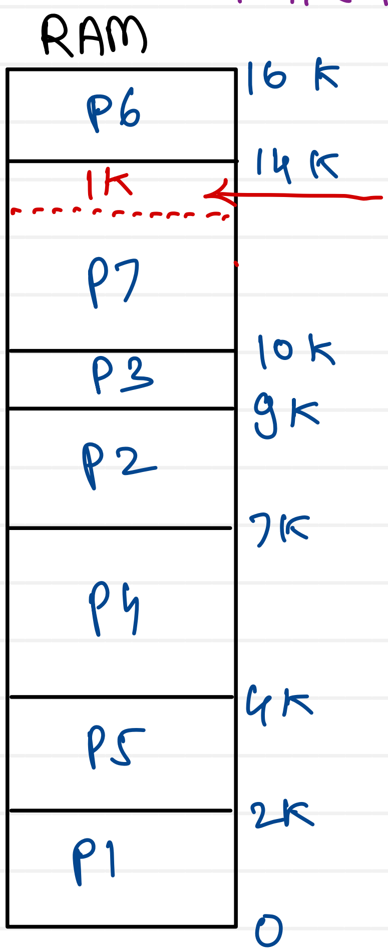


Simple MMU


Process
S
H
D
T
5000
3000
2000
1500
0
Loader
limit
base
5000
8000
CPU
2400
<
+
10400
Abort
Simple MMU
Contiguous Memory allocation
1. Fixed partition
2. Dynamic partition
5000
0
S
H
D
T
13000
11000
10000
9500
8000
actual/real/physical addresses
OS
PCB
l
b
5000
8000

Contiguous memory allocation

Fixed partition



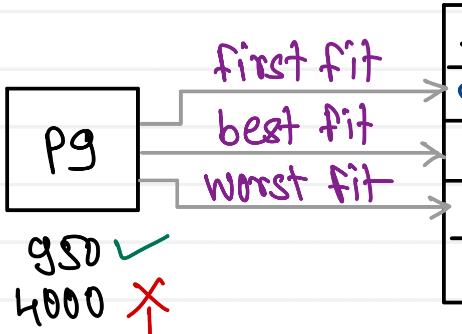
internal fragmentation

if process is not utilizing whole allocated partition, then some part will be wasted.

limitations :

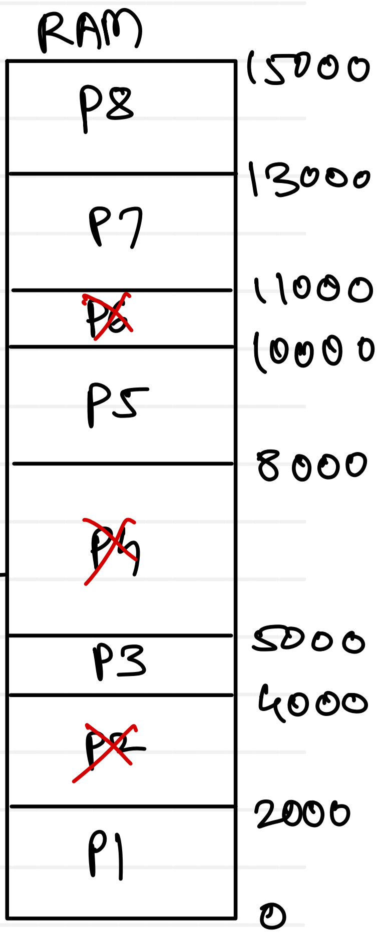
1. Max no. of processes are no. of partitions
2. max process size is equal to max partition size

Dynamic Partition



Free slot Table

limit	base
2000	2000
1000	10000
3000	5000



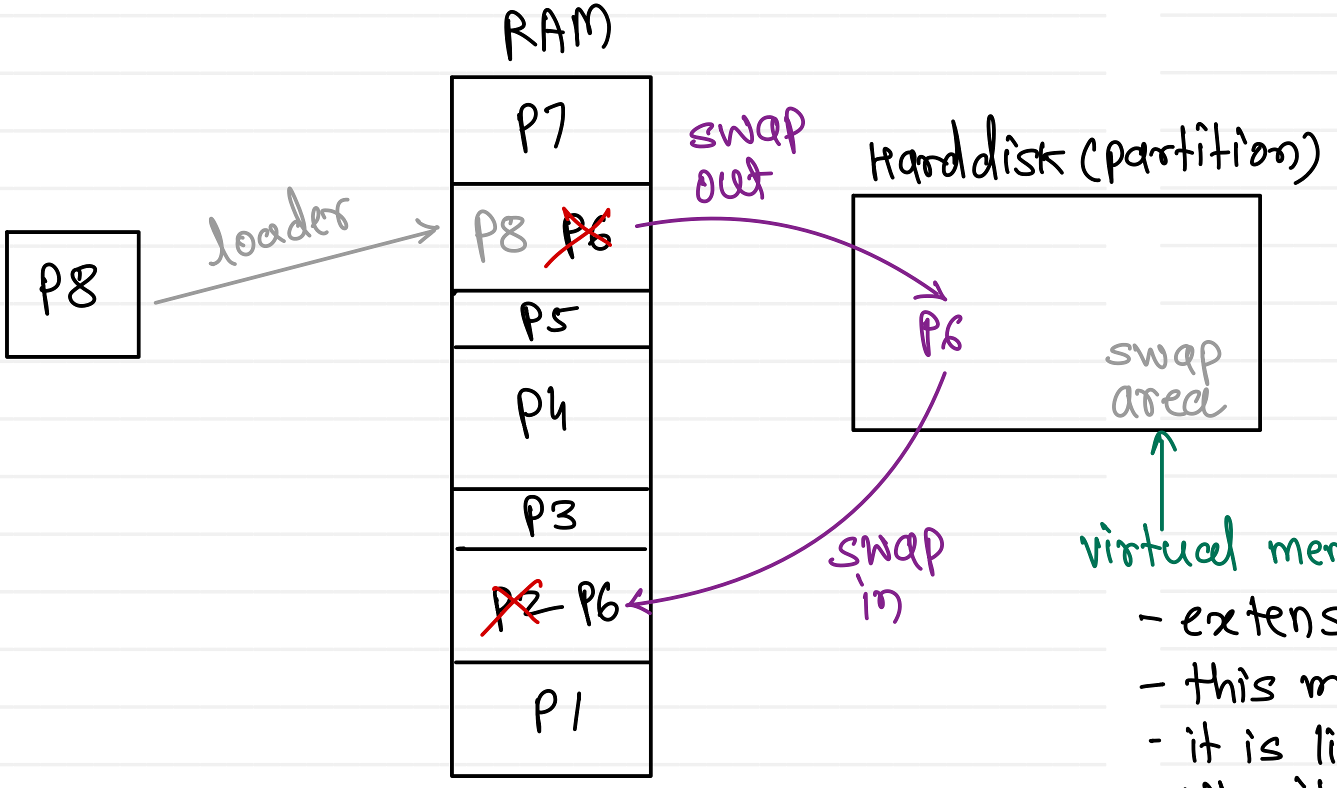
external fragmentation

if large contiguous free space is not available, we can not load process inside RAM

Compaction:

moving processes inside RAM to create large free space

Virtual memory



swap area

- partition → Linux
- file → Windows (pagefile.sys)

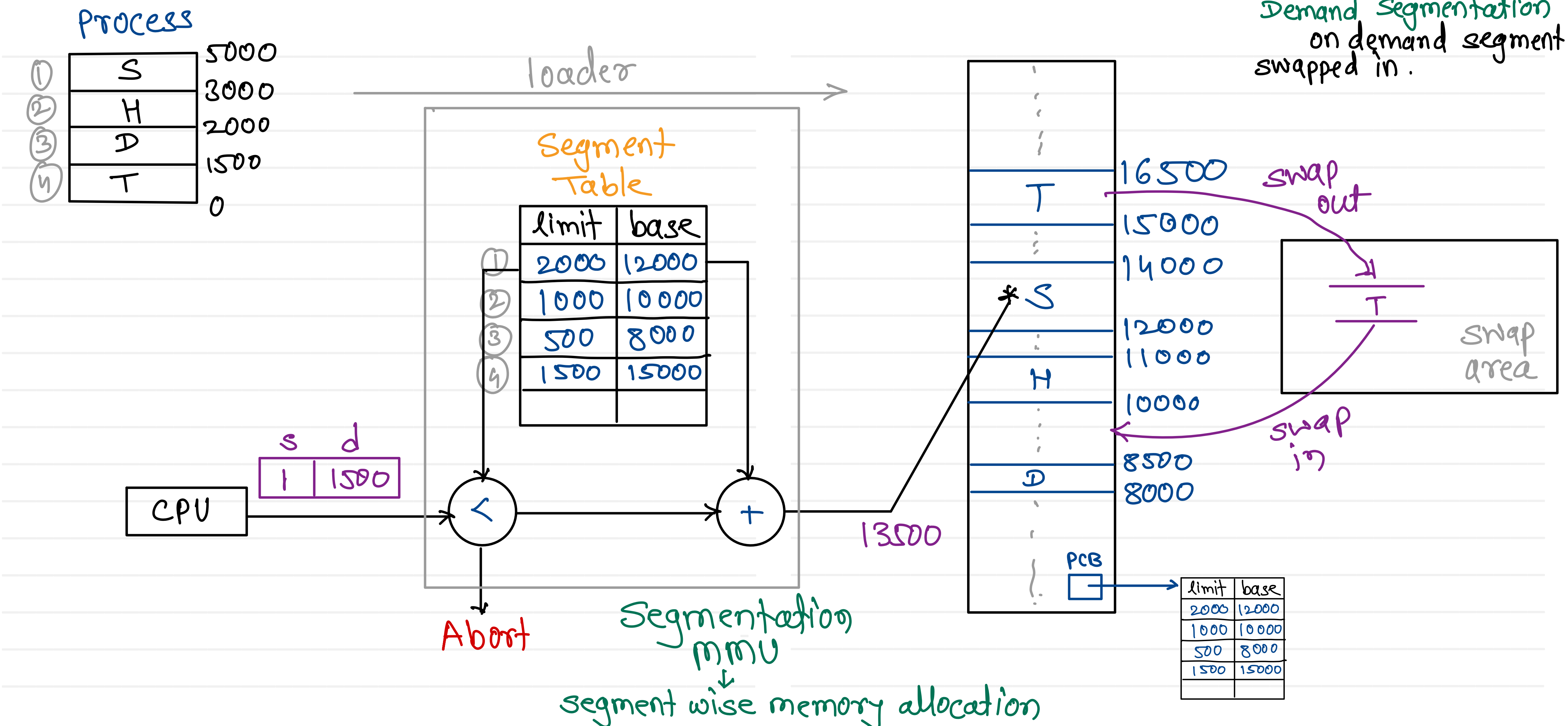
size = 2 * RAM size

virtual memory

- extension to RAM (memory)
- this memory is formatted like RAM
- it is like RAM but not RAM thats why it is called as virtual memory.
- processes are only kept into this area (inactive)
- processes never executes inside virtual memory.

Segmentation MMU

Demand Segmentation
on demand segment is swapped in.



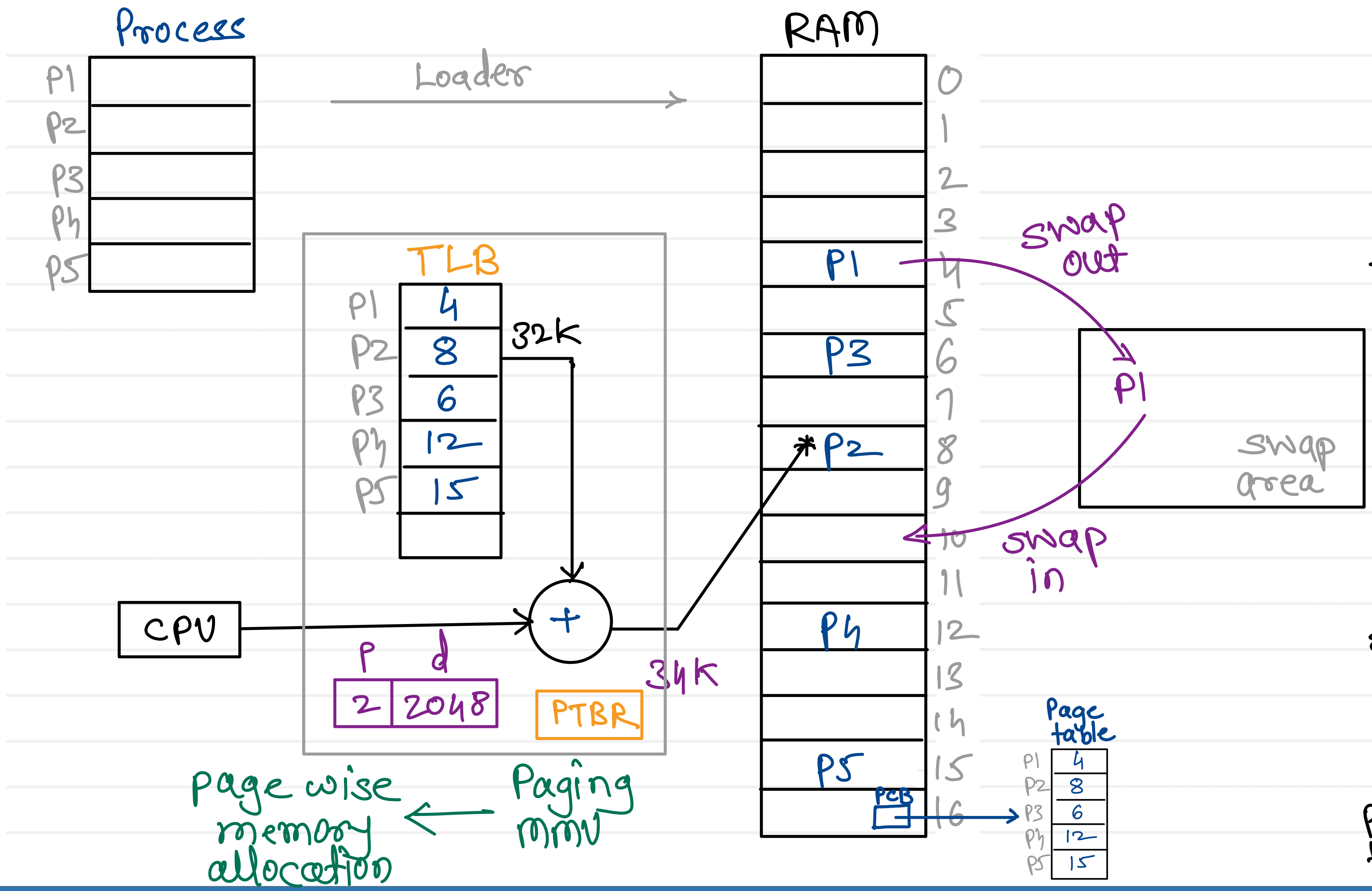
Segment Table

	limit	base
①	2000	12000
②	1000	10000
③	500	8000
④	1500	15000

limit	base
2000	12000
1000	10000
500	8000
1500	15000

Segmentation MMU
↓
segment wise memory allocation

Paging MMU



- RAM is divided into fixed equal size partitions

  size = 4kb (4096 bytes)

  "frame"/"physical page"

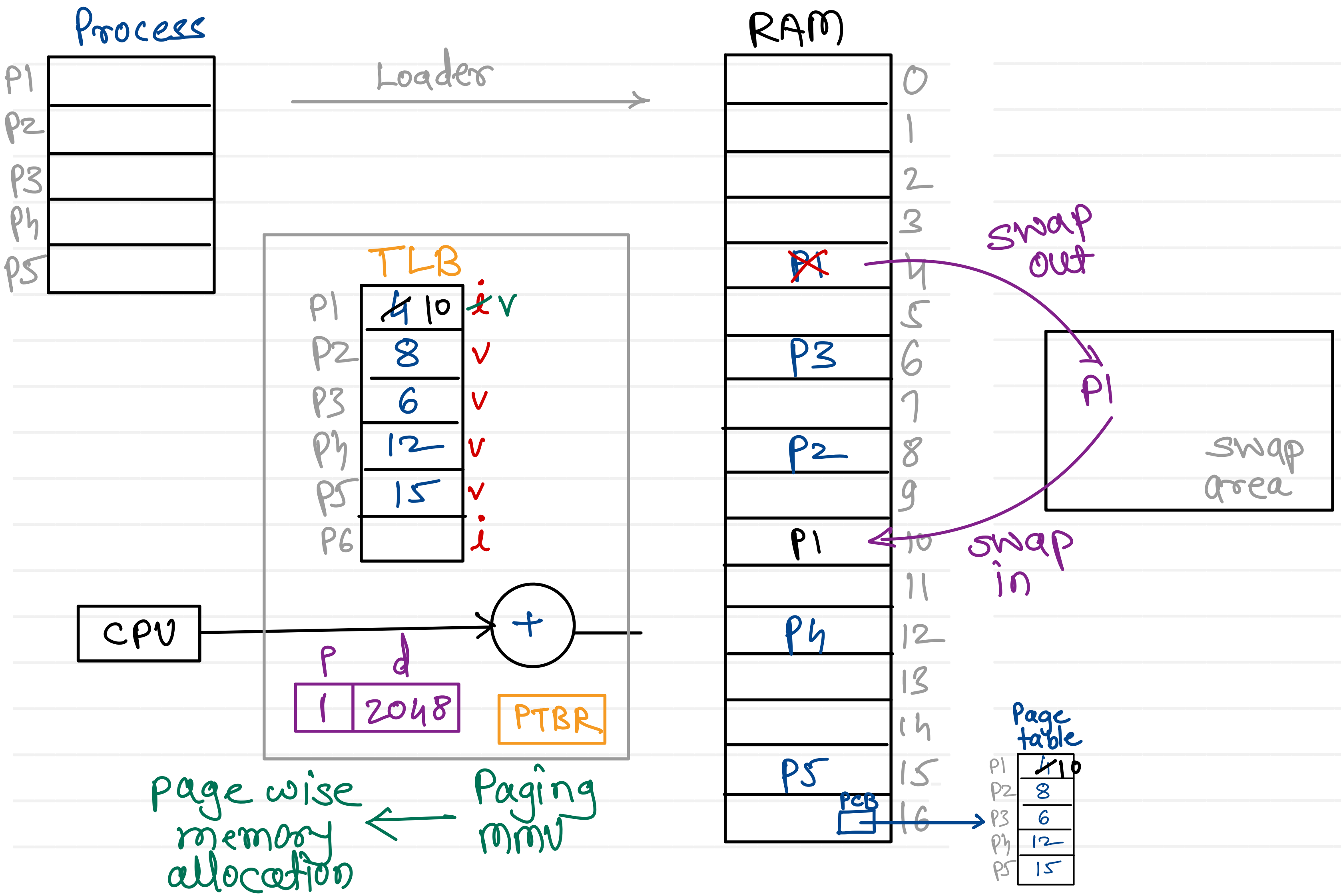
- Process is also divided into partitions of size equal to frame size.

  "page"/"logical page"

Demand paging: on demand page is swapped in

dirty bit: is used to decide, whether page to be copied or not at the time of swapping.

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

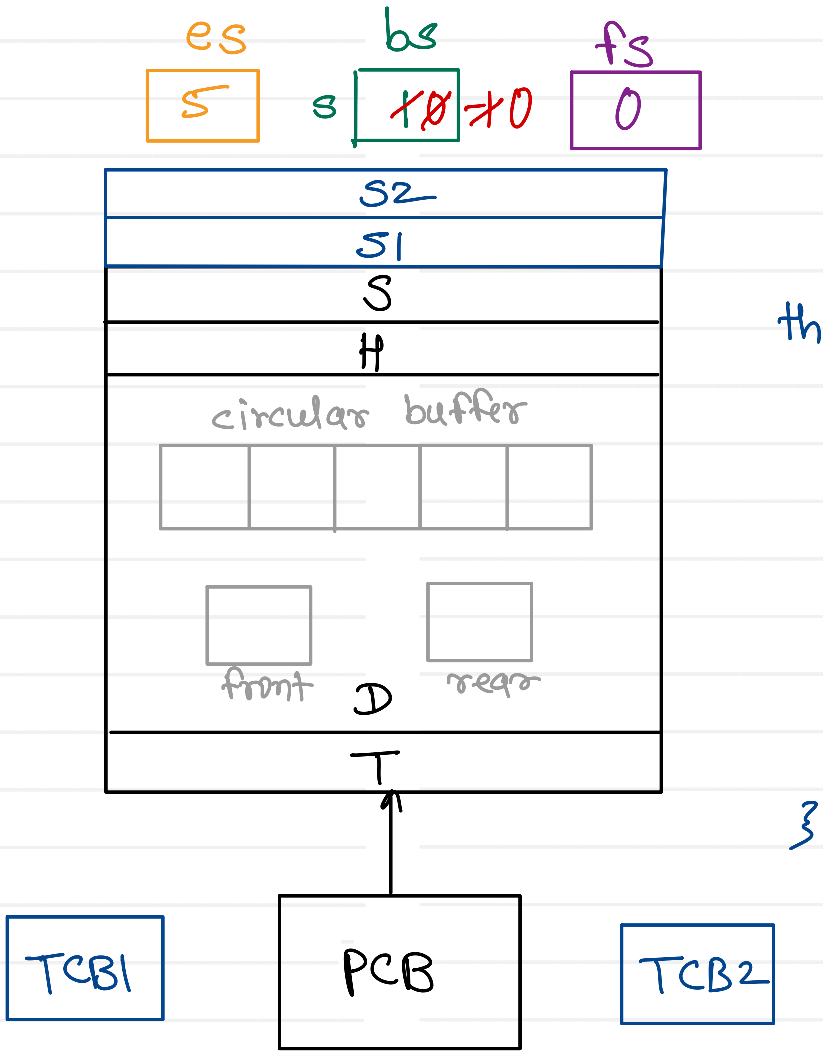
solution: increase the size

Producer - Consumer

Producer

```
thread1( ) {

    while(1) {
        P(s)
        P(es)
        buf[rear] = x;
        V(s)
        V(fs)
    }
}
```



Consumer

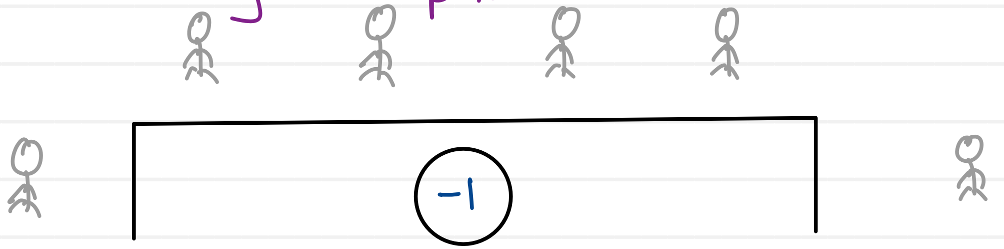
```
thread ( ) {

    while(1) {
        P(s)
        P(fs)
        x = buf[front];
        V(s)
        V(es)
    }
}
```

Semaphore

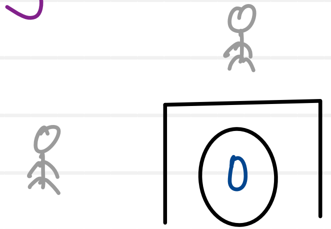
- semaphore is internally a counter
- Operations :
  1. Dec / wait / P() :
     a. dec count
     b. if count < 0, then block the current process
  2. Inc / post / V() :
     a. inc count
     b. if someone is blocked on this semaphore, wake up one

a. Counting semaphore



- counting no. of resources available
- counting no. of processes waiting

b. Binary semaphore



- only one should use resourse at a time.

Mutex

Mutex = Mutual Exclusion

↳ one at a time

- lock/unlock operations are performed on mutex

- process who locks the mutex becomes owner of the mutex

- only owner can unlock the mutex

Deadlock

- 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

SUNBEAM

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com