→ End user

1. Interface.
2. Controll prog.
3. Resource Allocation / manager.

IDE | Chrome | media player. | shell | Hello.c ⇒ Appln s/w

End user.

4. Bootable storage device (has contain OS setup)
   CD / DVD / PD / HDD

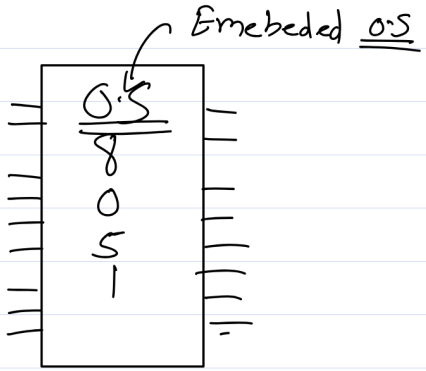
O.S. ⇒ Core OS + utility s/w + Appln s/w

Kernel

(control pannel, task mgr) (Notepad, paint, MS office, Calendor...)

CPU | RAM | HDD | Key-board (primary) i/p | monitor (primary) o/p | etc. ⇒ Hardware Resources

* Functions of Core os / Kernel



1. Process management
2. CPU Scheduling
3. Memory Mgm
4. File & I/O Mgm
5. Hardware Abstraction.

} Core function of os

6. Networking
7. Security and protection
8. User interface.

} Extra utility fun of os.

* Process Management

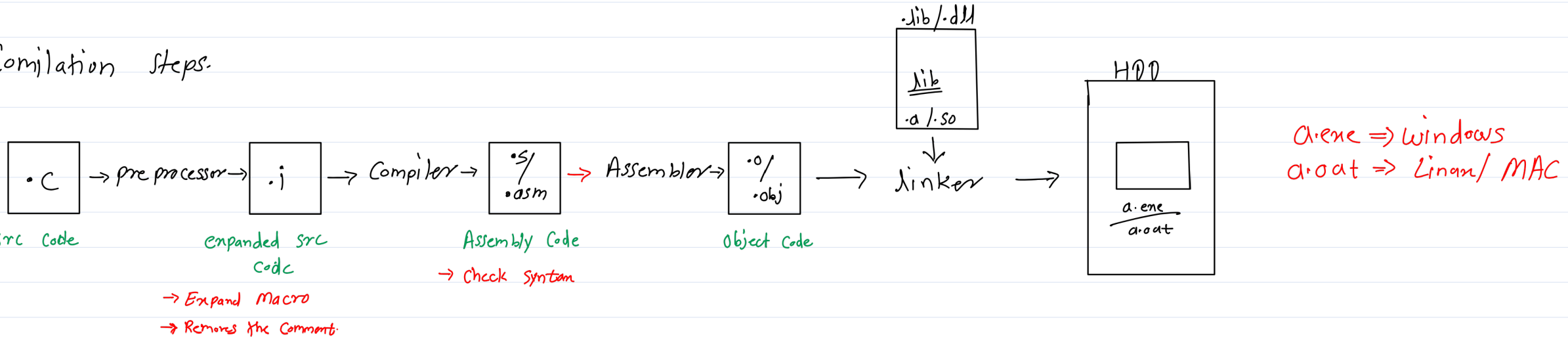
→ Process

→ Program : set of instructions.

Source Code — executable code

.c → .exe (Compiler)

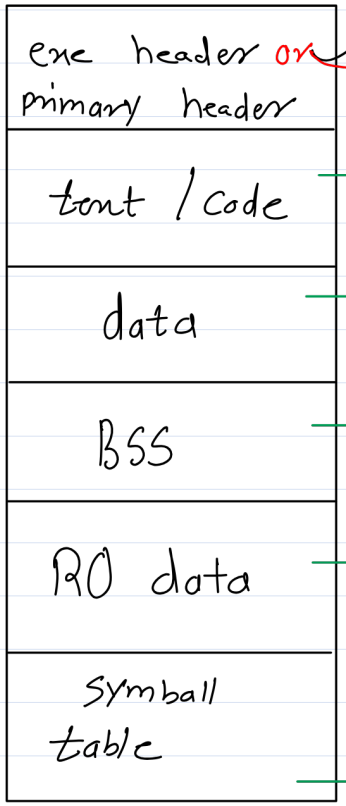
* Compilation steps.



a.exe ⇒ windows
a.out ⇒ Linux / MAC

(Section binary)

•out



→ Entry point function address (main)

→ Information about the Remaing section.

→ Magic Number → uniq Number of file format
→ Identity of file format.

→ machine code

→ - initialised global & static variable

→ (block started by symball)
- uninitialised global & static variable

→ " Constant string literals, 0, 1, 2, 'A', 'B' ...

→ - The information Related of Variable & Fanction.
- name, address, section, size, flags
- These into use for debugging purpose.

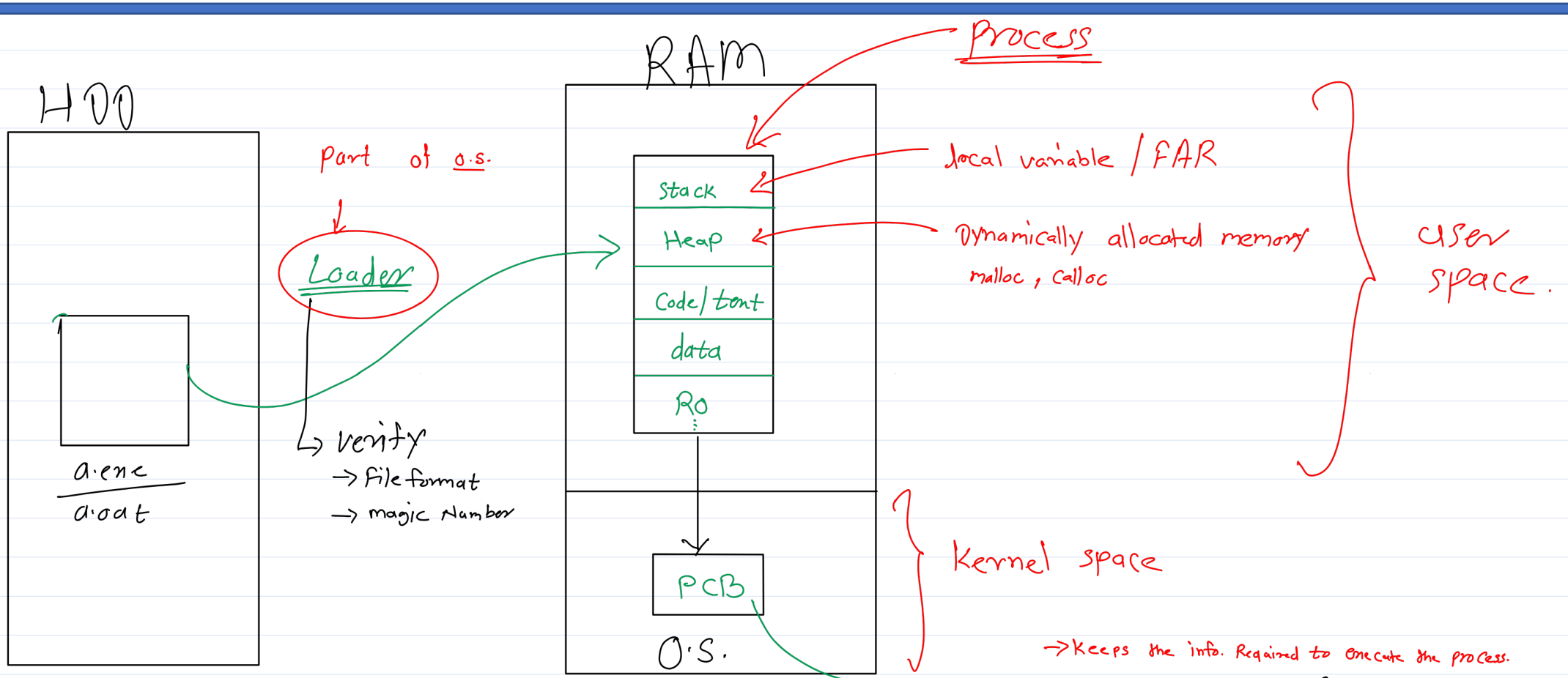
Char *p = "Sunbeam" ⇒ RO

*p = 'x' ; // Error
puts(p)

Binary file has magic Number
(It present on starting 2 or 4 bytes of of code file)

a.exe ⇒ MZ
a.out ⇒ ?ELF
• bmp ⇒
• pdf ⇒
• text → X

Portable Executable (PE)
Executable linking format (ELF)

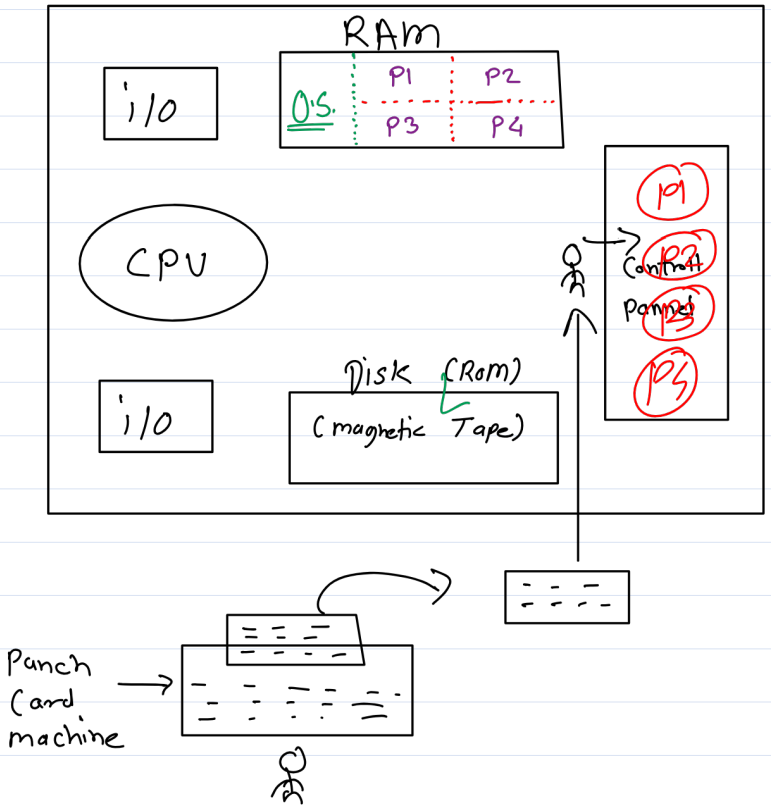

HDD
a.exe
a.out
Part of o.s.
Loader
verify
→ file format
→ magic Number
RAM
Process
Stack
Heap
Code/text
data
RO
local variable / FAR
Dynamically allocated memory
malloc, calloc
user space.
PCB
O.S.
Kernel space
→ Keeps the info. Required to execute the process.
PCB ⇒ Process Control block, task_descripter
→ Pid
→ Sched. info (state, Priority, algo)
→ files info, mem info., IPC info.
→ Execution Context, exit status, PC
./a.exe → Ran

* OS Evolution.

```
}
sf (no3)
sf (no2)        } I/o instr
add = no1 + No2
sab = no1 - No2
mal = no1 * no2 } CPU instr
pf (add)
pf (sab)        } I/o instr
pf (mal)
}
```

Mainframe Computer

(old computer)



1. Resident monitor.
2. Batch System
3. Multi-Programing.
   - loading multiple program in main memory.
   - Here on RAM load mixed program
     - ↳ CPU bound + I/o bound
   - better utlization of CPU
   - degree of M.P.
     - ⇒ num of programs that can be kept in RAM.

Program = CPU instr + I/o instr

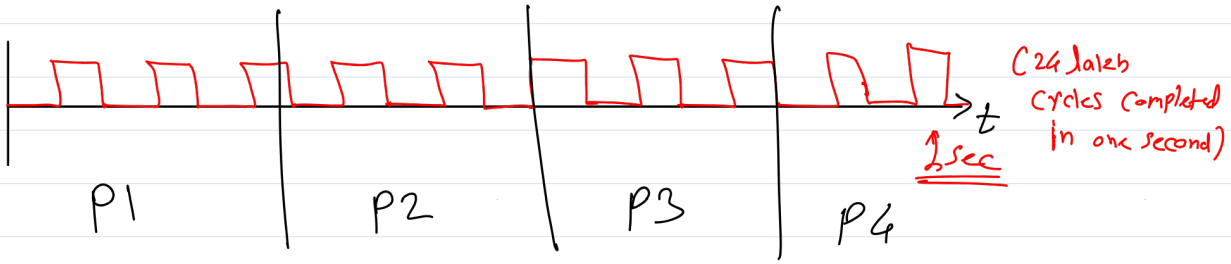
(CPU burst Time) + (I/o burst time)

④ Time sharing / Multi tasking O.S.

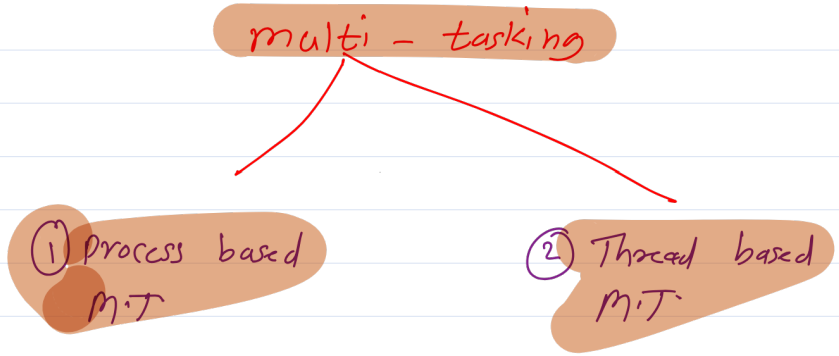
⇒ Sharing CPU time among multiple tasks present in main memory & Ready for execution.

(2.40GHz)

* freq : Num of cycles completed in one sec.

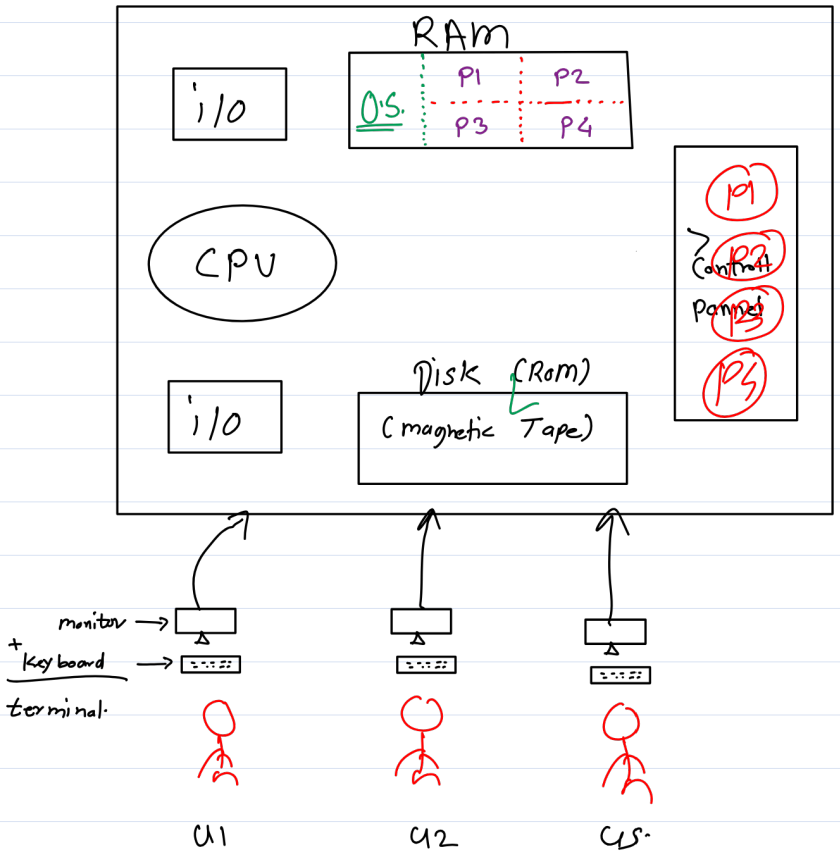


⇒ Response time < 1 sec

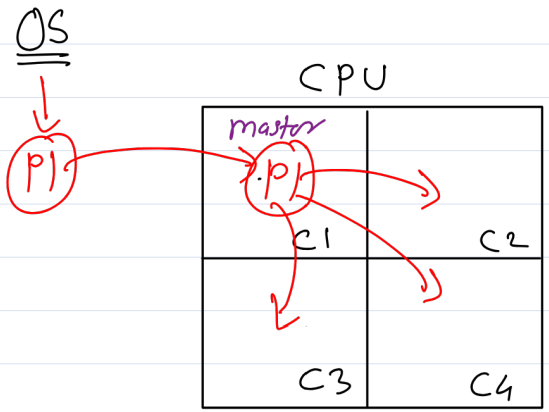


* Multi - user System.

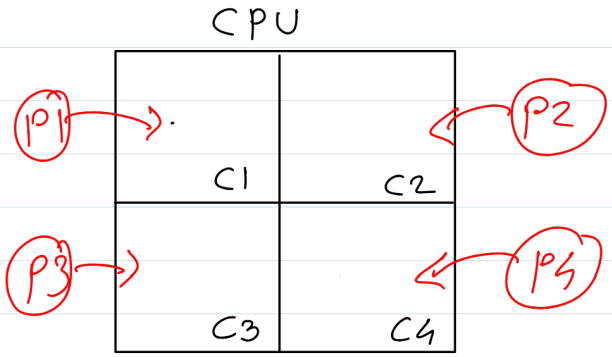
puter)



* Multi-Core / Multi - Processor.



Asymmetic M.P.

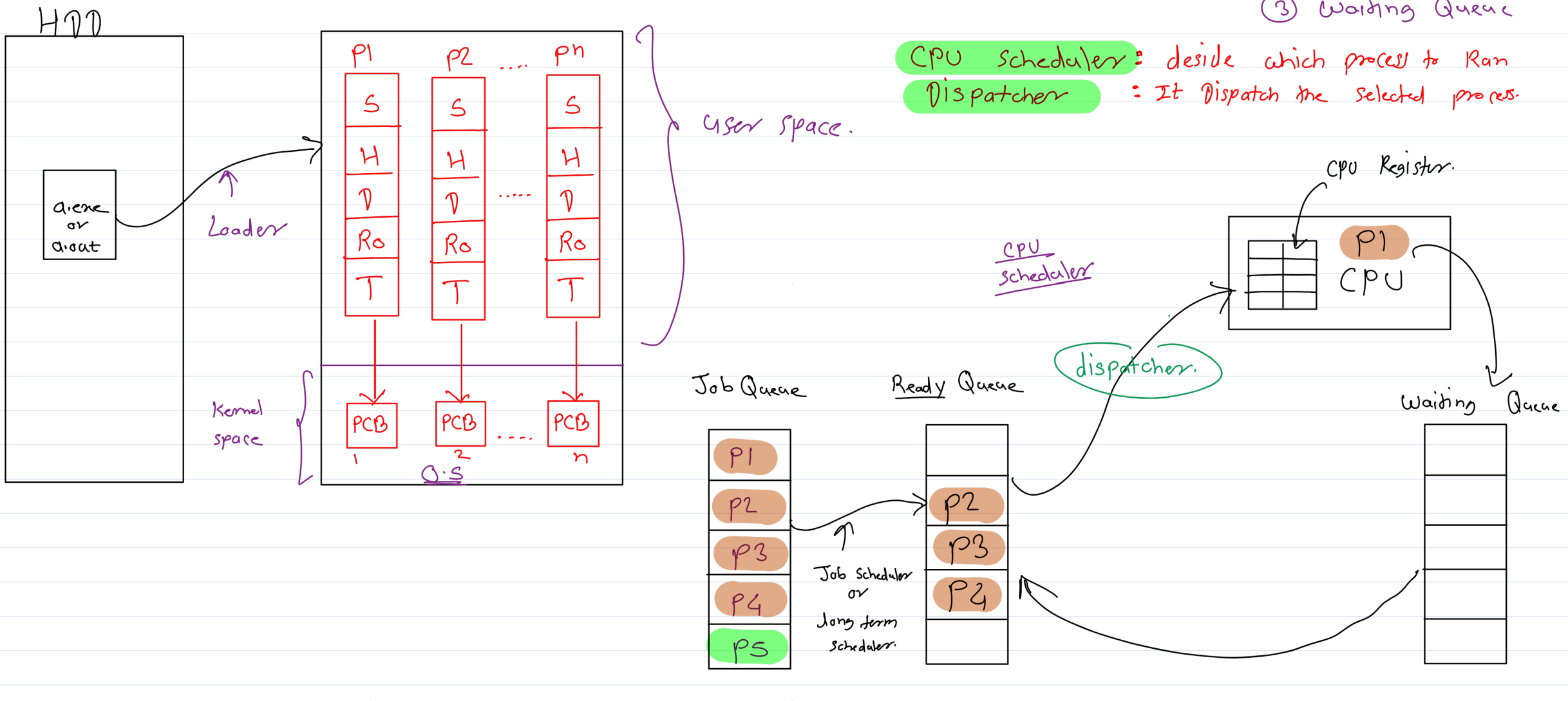


Symmetic M.P.

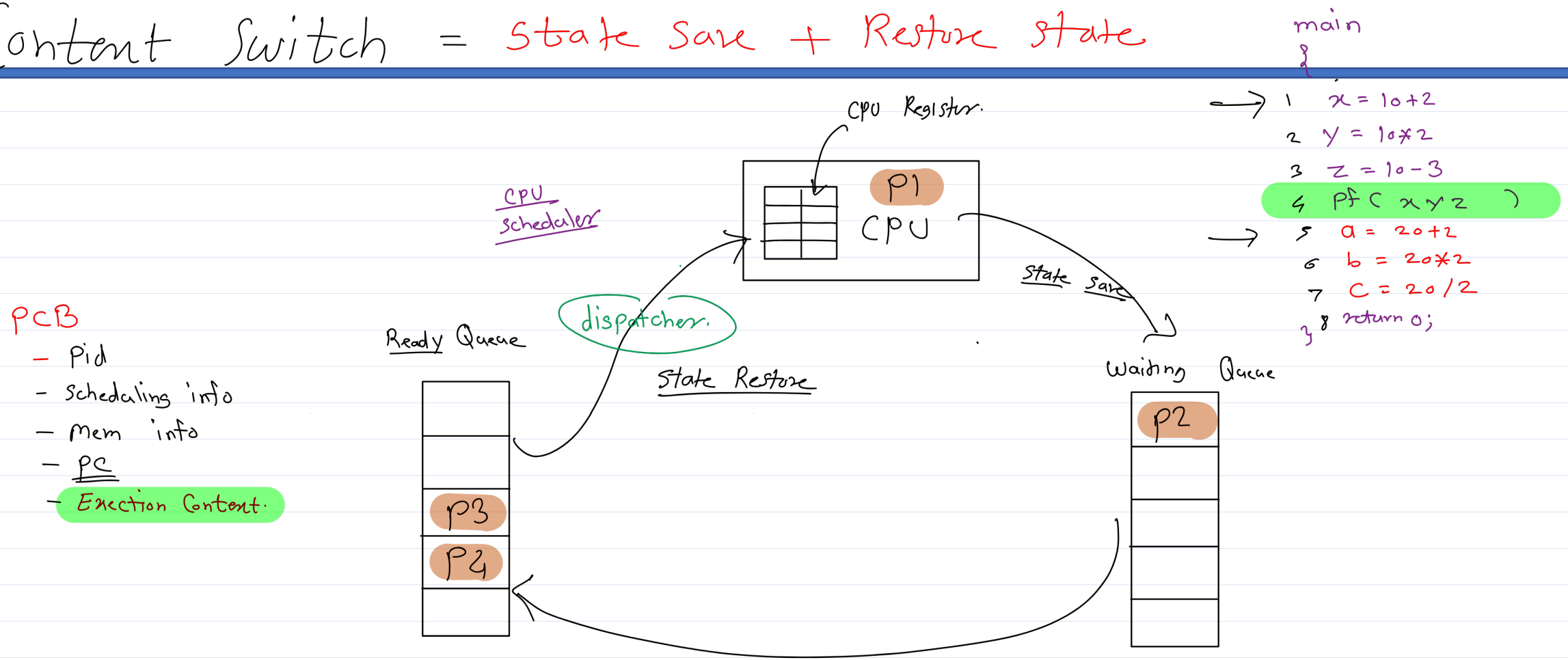
* Process life Cycle

Kernel data structe.
1. Job Queue
2. Ready Queue
3. Waiting Queue

CPU scheduler : deside which process to Ran
Dispatcher : It Dispatch the selected process.

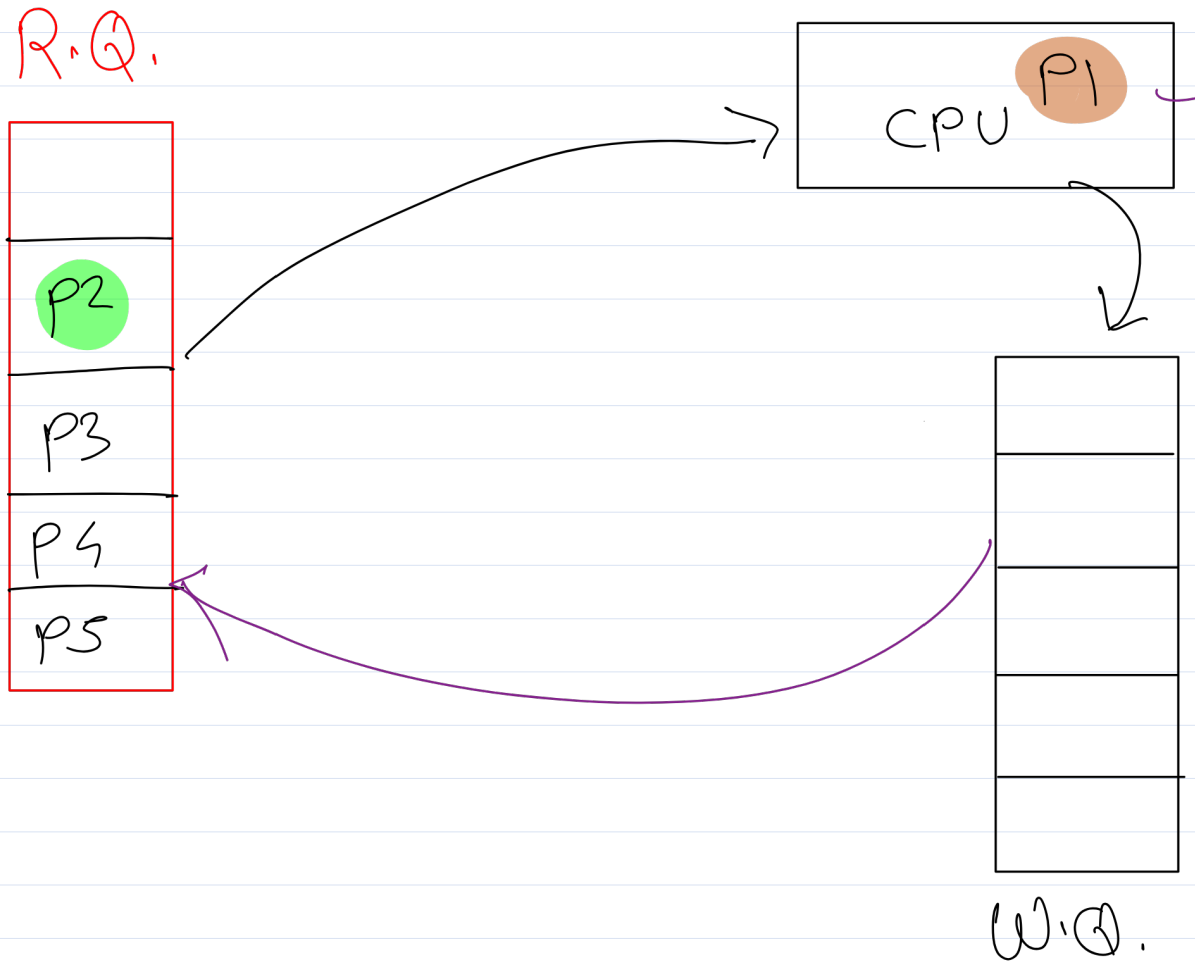


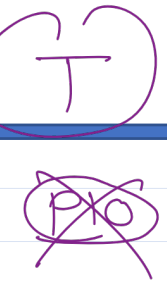
* Content Switch = State Save + Restore State



PCB
- Pid
- Scheduling info
- Mem info
- PC
- Exection Content

(7) CPU Schedular





1. Running → Terminated
2. Running → waiting state.
3. Running → Ready State
4. Waiting → Ready.

* FCFS.

Convoy effect : If bigger process arrive first, Avg W.T increase.

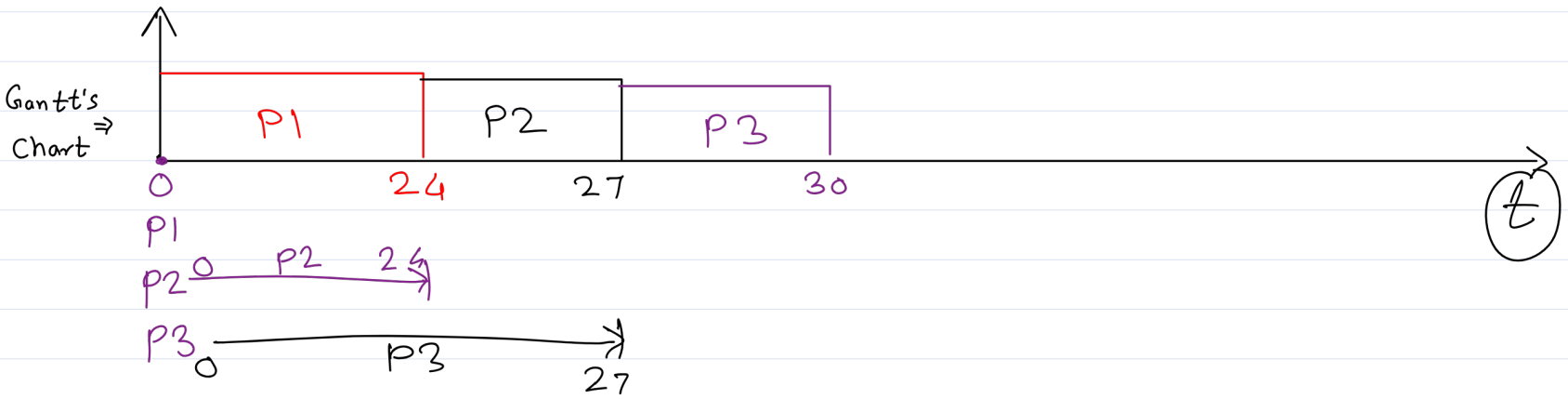
➢FCFS Scheduling

Process	Arrival Time	CPU Burst	Wait Time	Turn Around Time
P1	0	24	0	24
P2	0	3	24	27
P3	0	3	27	30

Avg W.T. = Sum of all process W.T. / Num of process

= (0 + 24 + 27) / 3

= 51 / 3 => 17



Avg. T.A.T = Sum of all process T.A.T / Num of process

=> (24 + 27 + 30) / 3

=> 81 / 3

=> 27

* FCFS.

➢FCFS Scheduling

No Convoy effect.

Completion time

Process	Arrival Time	CPU Burst	Wait Time	Turn Around Time
P1	0	3	0	3
P2	0	3	3	6
P3	0	24	6	30

$$\text{Avg. W.T.} \Rightarrow \frac{0+3+6}{3}$$

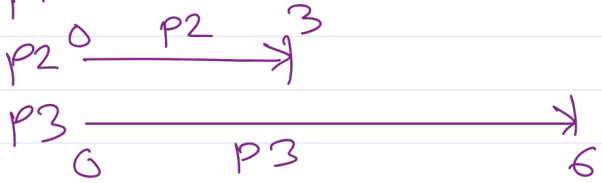
$$= \frac{9}{3}$$

$$= 3$$

$$\text{Avg. T.A.T} = \frac{3+6+30}{3}$$

$$= \frac{39}{3}$$

$$\Rightarrow 13$$

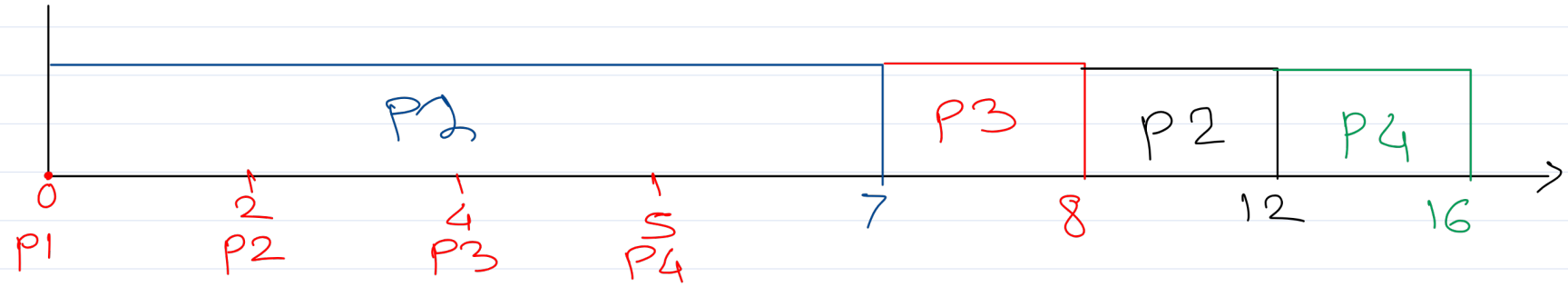


SJF → SNTF (Non-Preemptive) X
SRTF (Preemptive)

SJF – gives min Avg wait time.

➢ SJF/SNTF Scheduling (Non-Preemptive)

Process	Arrival Time	CPU Burst	Wait Time	Turn Around Time
P1	0	7	0	7
P2	2	4	6	10
P3	4	1	3	4
P4	5	4	7	11



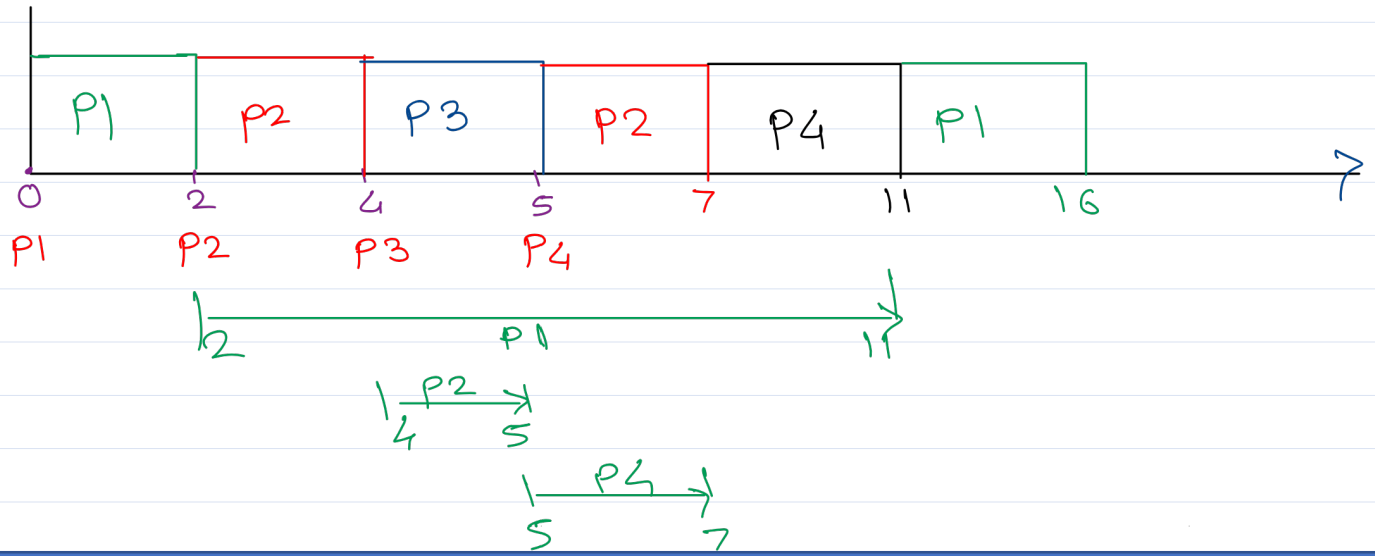
SJF

➢ SRTF Scheduling (Preemptive)

Process	Arrival Time	CPU Burst	Remaining Time	Wait Time	Turn Around Time
P1	0	7	5 X	9	16
P2	2	4	2	1	5
P3	4	1	X	0	1
P4	5	4	X	2	6

Avg. WT. =>

Avg. T.A.T =



* Starvation

A.T	Process	C.B.T
0	P1	100
0	P2	20 X
0	P3	10 X
30	P4	20 X
50	P5	10

blocked (Starvation)

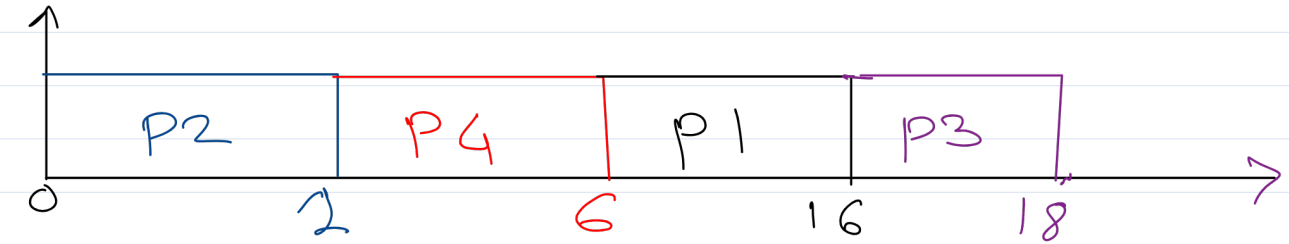
min → high priority

max → low priority

➢ Priority Scheduling

Process	Arrival Time	CPU Burst	Priority	Wait Time
P1	0	10	3	6
P2	0	1	1 – high	0
P3	0	2	4 – low	16
P4	0	5	2	1

T.A.T
16
1
18
6

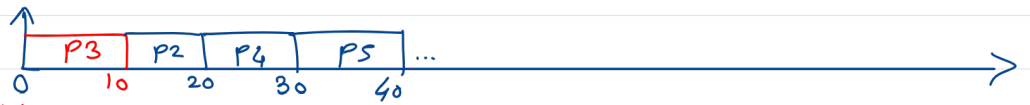


P1
P2
P3
P4

* Starvation → Soln → Ageing

A.T	Process	C.B.T	Priority
0	P1	10	20 → 19 → 18.
0	P2	10	5 x
0	P3	10	1 x
20	P4	10	7 x
30	P5	10	10

blocked (Starvation)



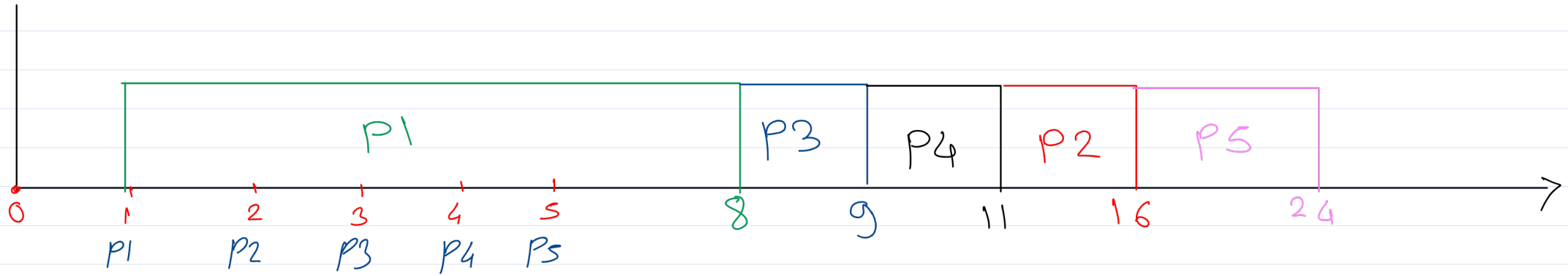
P1
P2
P3

SJF → non-preemptive (SHTF)

Process	A.T.	C.B.T	W.T.	T.A.T (W.T + C.B.T)
P1	1	7	0	7
P2	2	5	9	14
P3	3	1	5	6
P4	4	2	5	7
P5	5	8	11	19

Avg. T.A.T. $\Rightarrow \frac{7+14+6+7+19}{5}$

$\Rightarrow \frac{53}{5} \Rightarrow 10.6$



* Kernel data structure.
  - Job Queue
  - Ready Queue
  - Waiting Queue.

* Process life cycle / Process state
  1. New
  2. Ready
  3. Running
  4. Waiting
  5. terminating

(*) Ran → terminated
(2) Ran → waiting

(3) Ran → Ready.
(4) waiting → Ready.

* CPU Scheduling Algo.

  * CPU Scheduling Criteria
    - CPU utilization
    - Throughput
    - Waiting time
    - Response time
    - Turn around time.

(1) FCFS
  ↳ non-preemptive
  → Convoy effect

(2) SJF
  → min^m Response time (Avg)
  → Starvation.

(3) Priority
  ↳ starvation Soln → Ageing.

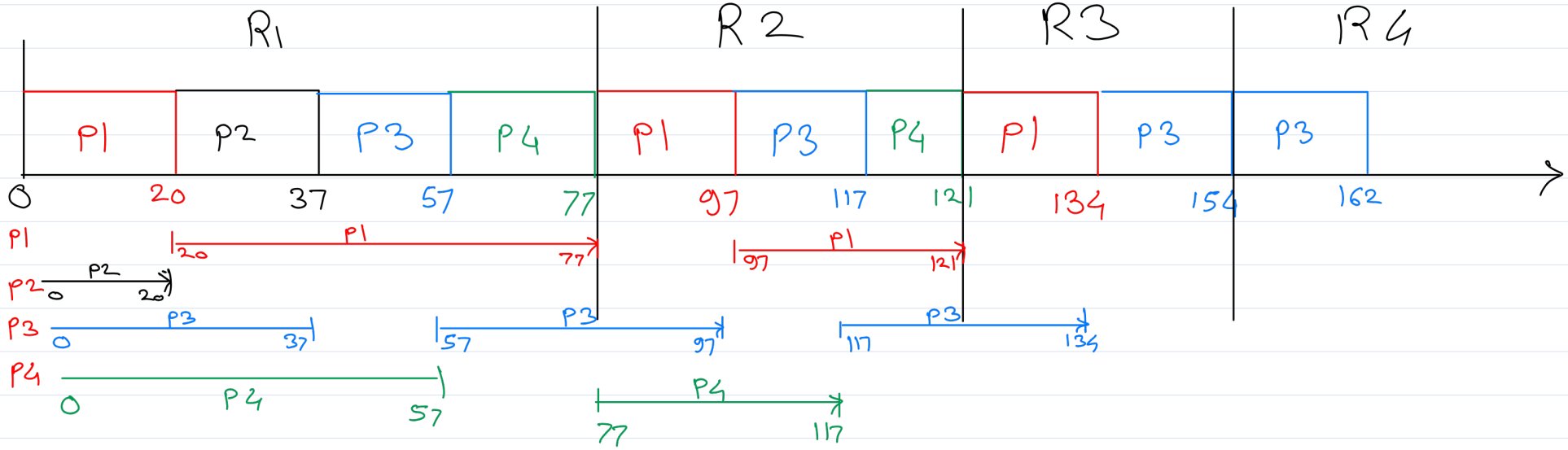
(4) Round Robin

Given data: T.Q ⇒ 20

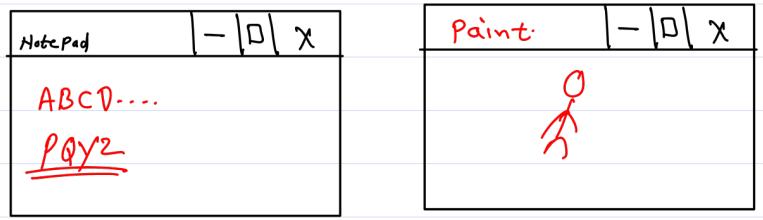
➢ Round Robin Scheduling (preemptive)

Process	A.T	CPU Burst	Remaining Time	Wait Time	Response Time
P1	0	53	33, 13, X	57+24 ⇒ 81	0
P2	0	17	X X X	20	20
P3	0	68	48, 28, 8, X	37+40+17 ⇒ 94	37
P4	0	24	4 X X	57+40 ⇒ 97	57

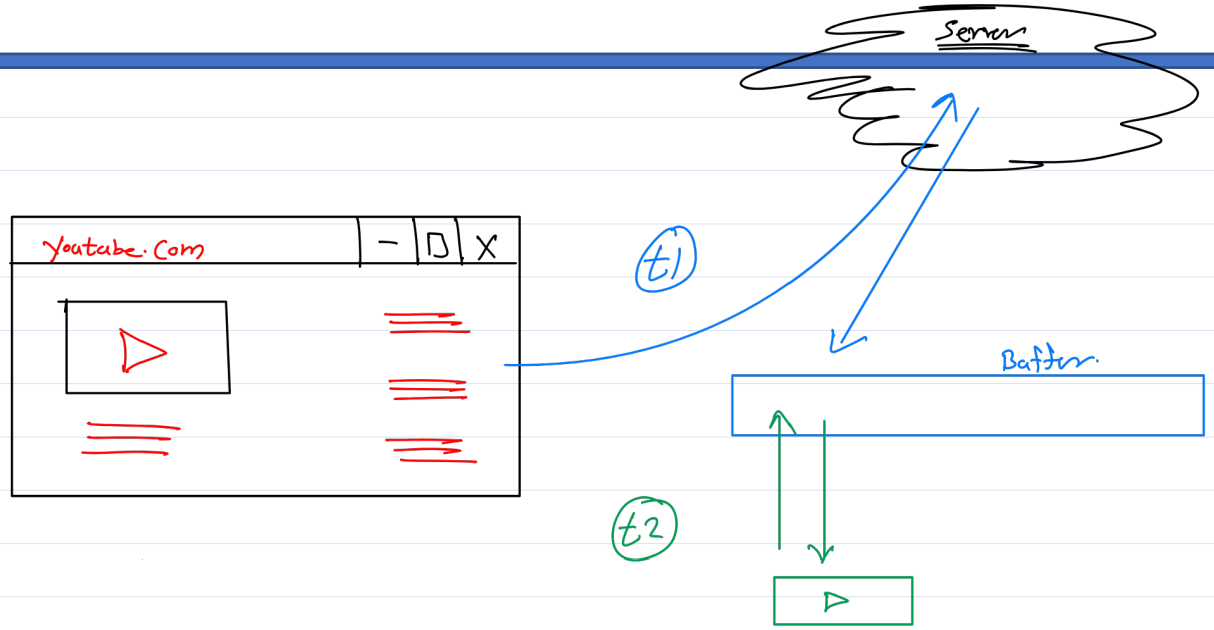
T.A.T.
134
37
162
121



*IPC



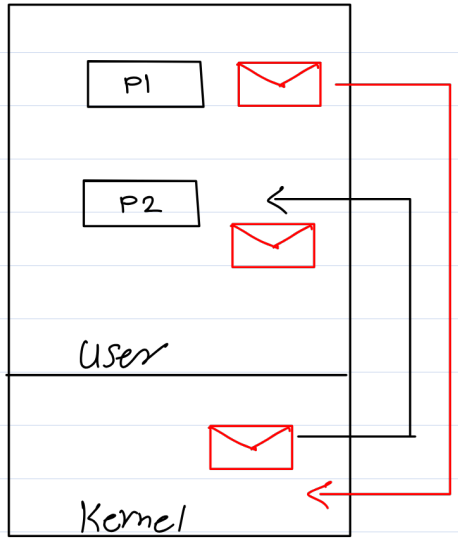
* Independent process.



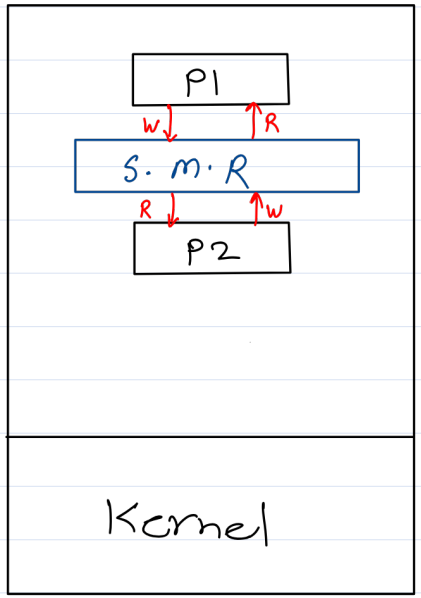
* Co-operative process.

* IPC

* Message Passing

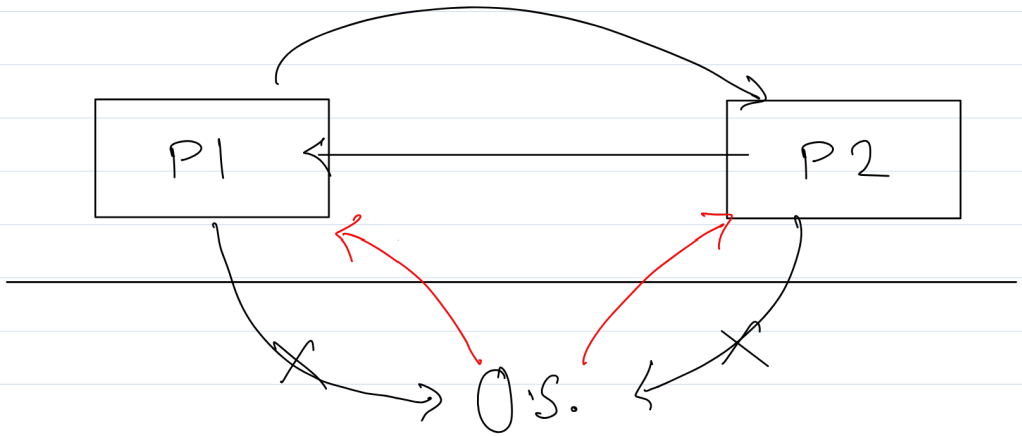


* Shared memory.

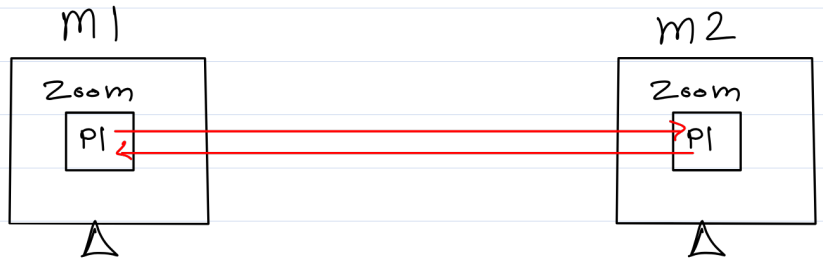


1. pipe
2. Message Queue
3. signal
4. Socket

(2) Signal.



(3) Socket.

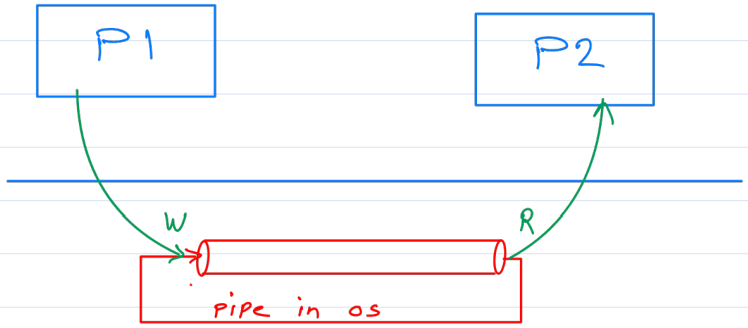


IPC

① Pipe

- stream based
- unidirectional

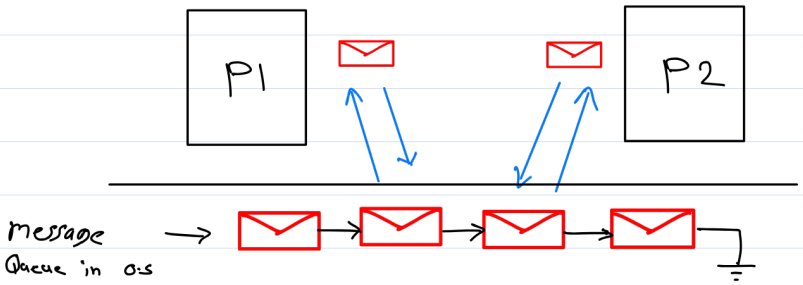
types
① Named pipe
② unnamed pipe



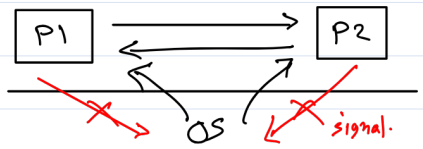
→ pipe is circular Buffer.

② message Queue.

- Bidirectional

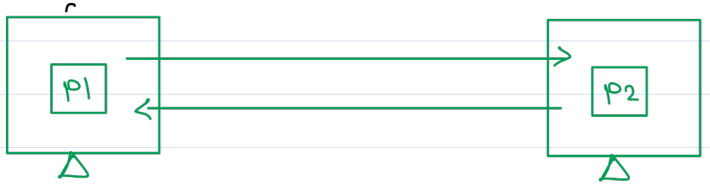


③ signal



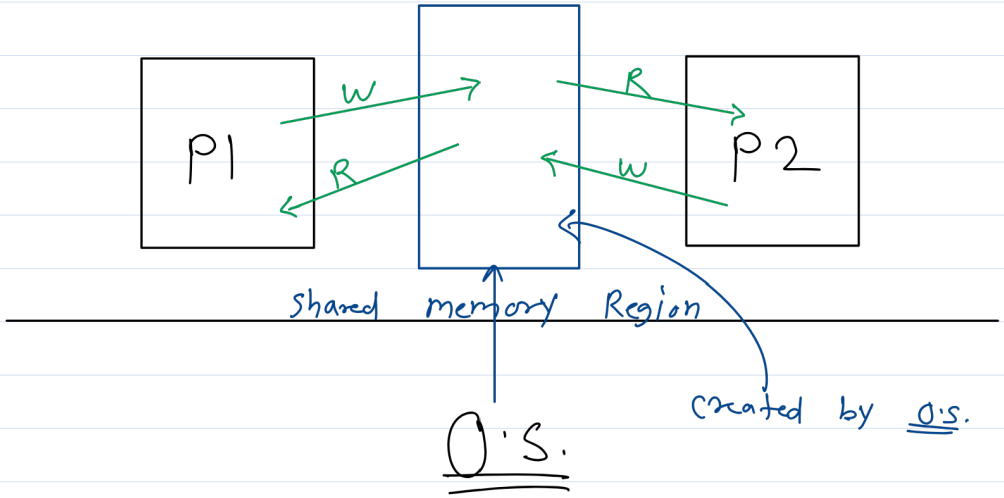
- Predefined signal
- 64 signal in Unix o.s.
- unidirectional.

④ Sockets.

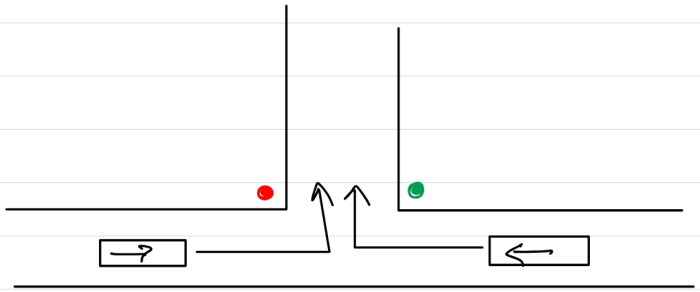


- Socket is comm^n endpoint
- Bidirectional
- use to comm^n diff or same machine.

(4) Shared memory:

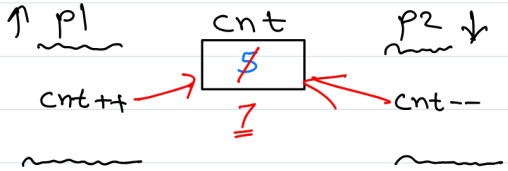


- It is fastest IPC mechanism.



Race Cond^n: If two processes try to access same Resource of the same time.

- due to Race Condition data inconsistency happened.
  (Peterson problem)



Sync mechanism: ensure that only one process will Accesses the Resource at a time and thus data inconsistency avoided.

① Classic / Counting Semaphore.

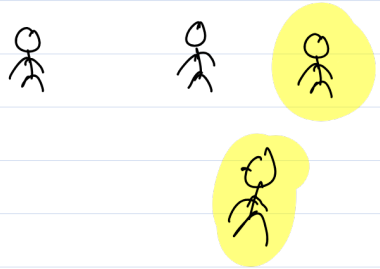
(A) Semaphore

- It is Counter.

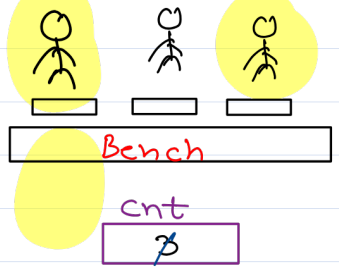
* Decr op / wait op / P op
  - Decrement Semaphore Count
  - if cnt < 0, True → block the current process.
  - Before accessing Resource.

* Inc op / signal op / V op
  - Increment Sema Count
  - If one/more process are, then blocked wak-up one of the process.
  - After the Releasing Resource.

(↑) V oprn



P oprn ↓

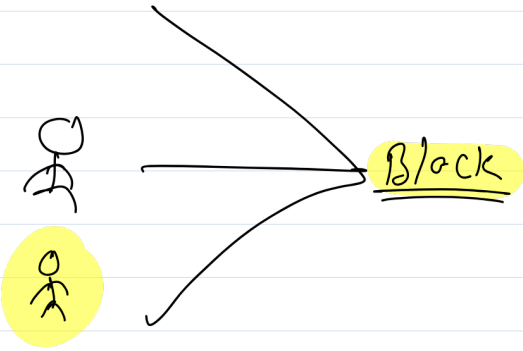


2 < 0 → F
1 < 0 → F
0 < 0 → F

-1 < 0 → T

-2 < 0 → T

-3 < 0 → T



↓

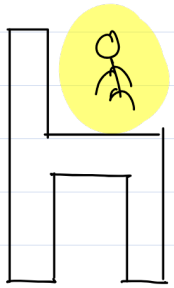
-2

-1

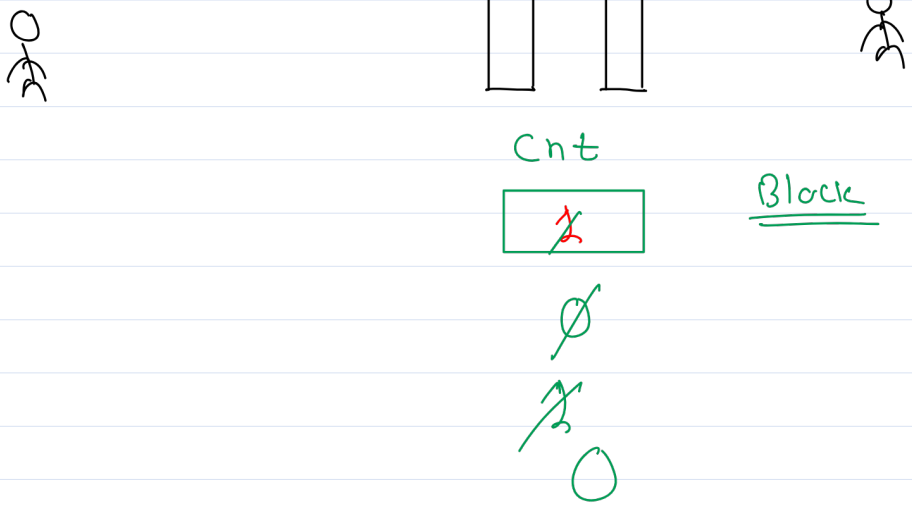
2 Binary Semaphore

0 & 1

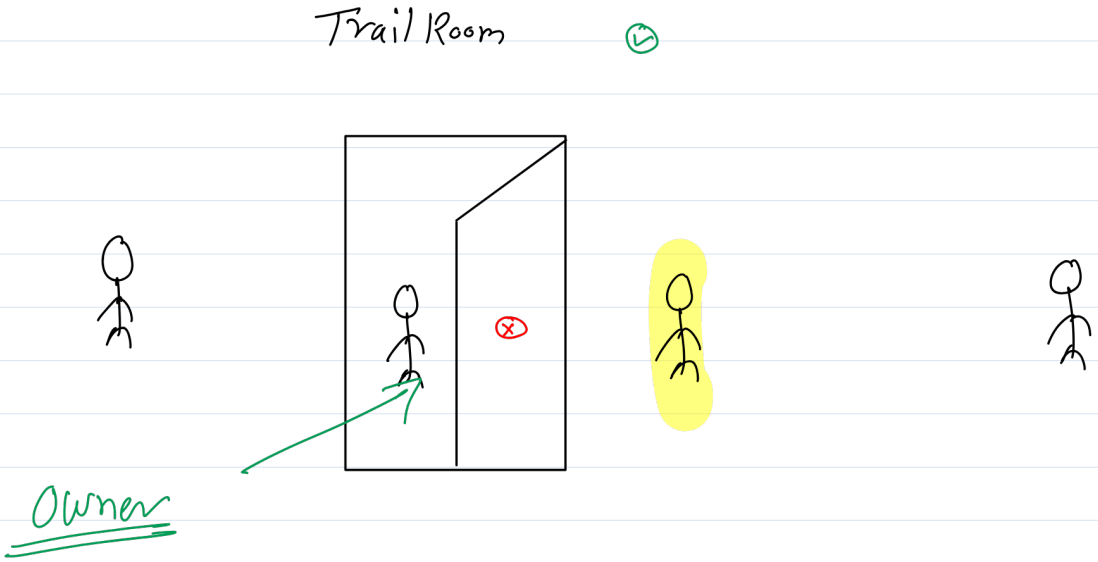
V (↑)



P (↓)



2 Mutex.



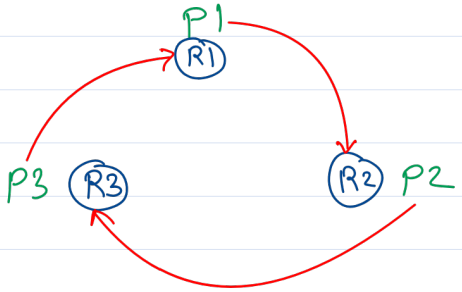
P1 (↑) lock()

P2 (↓)

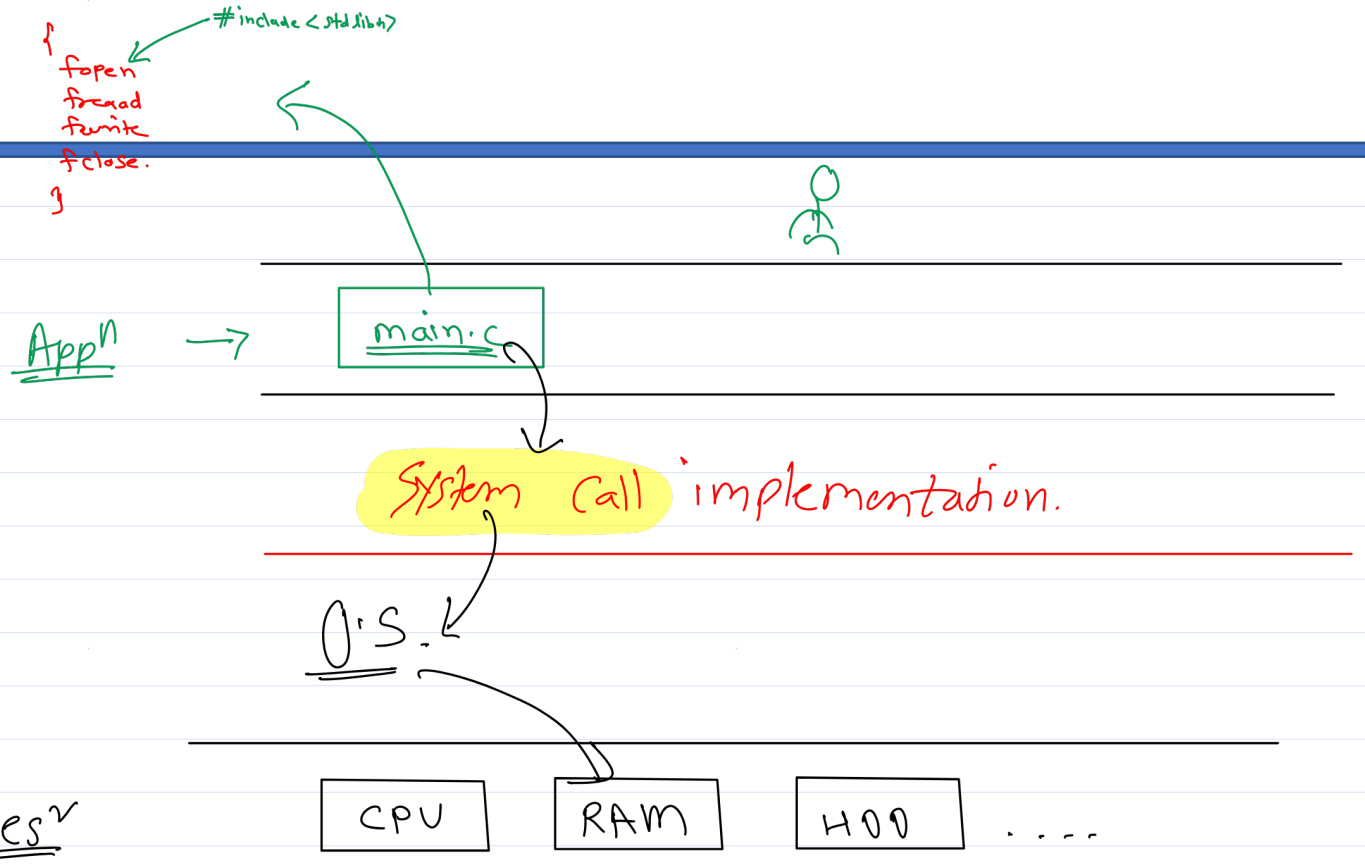
unlock()

* Deadlock.

1. Mutual enclusion
2. No-Preemption
3. Hold & wait
4. Circular wait.



System Call


#include <stdlib.h>
fopen
fread
fwrite
fclose.
App^n
main.c
System Call implementation.
O.S.
Res^r
CPU
RAM
HDD
....