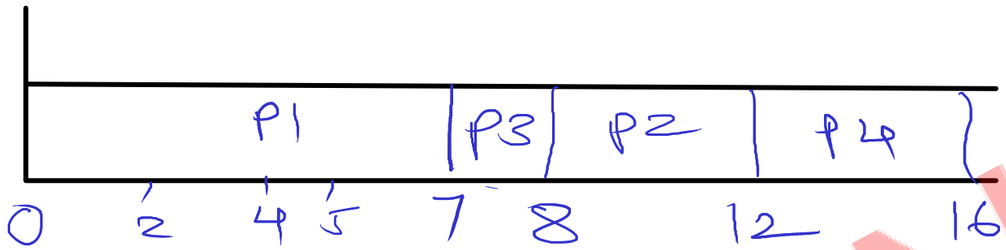
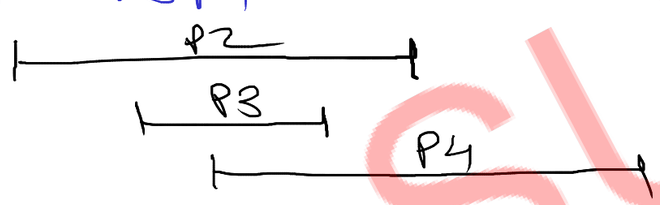
SJF (Shortest Job First)

(Non-Preemptive)

Process	Arrival	CPU Burst	WT	RT	TAT
P1	0	7	0	0	7
P2	2	4	6	6	10
P3	4	1	3	3	4
P4	5	4	7	7	11



P1 P2 P3 P4

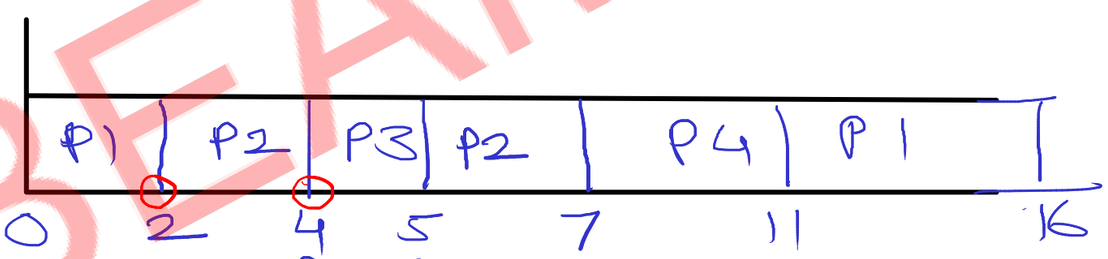


(Preemptive)

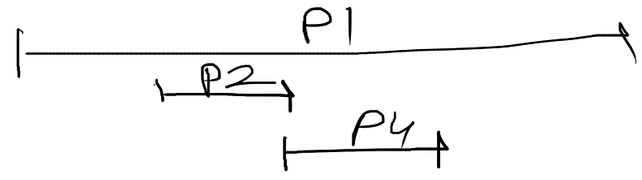
(Shortest Remaing Time First)

Process	Arrival	CPU Burst
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Remain time	WT	RT	TAT
5	9	0	16
2	1	0	5
0	0	0	1
4	2	2	6



P1 P2 P3 P4



Starvation

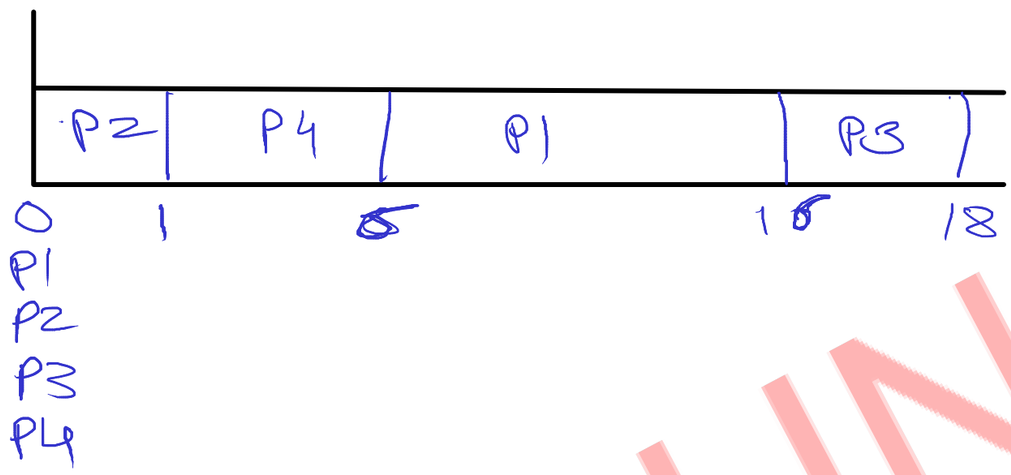
- due to longer CPU burst, process will not get enough CPU time to execute

- there is no solution for starvation

Priority

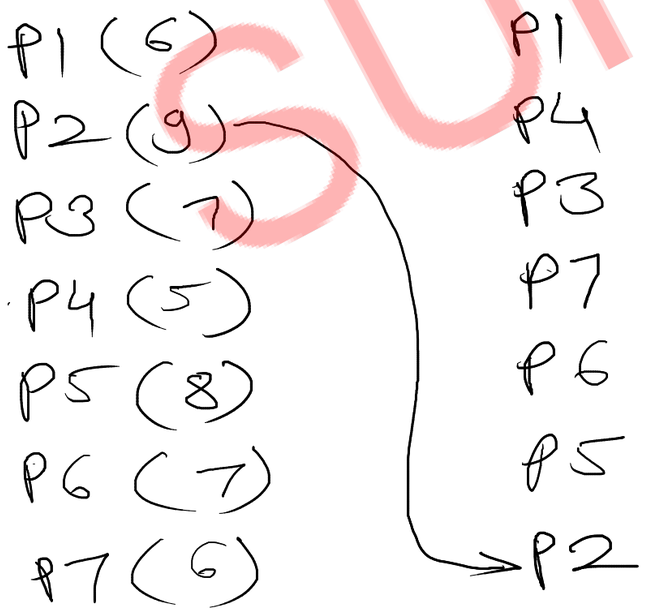
(Non-Preemptive)

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	WT	RT	TAT
P1	0	10	3	6	6	16
P2	1	1	1	0	0	1
P3	3	2	4	13	13	15
P4	0	5	2	1	0	6



Starvation

- due to less priority of process, it is not getting enough time to execute on CPU

Aging

- increase the priority of process (whose priority is less) gradually

RR (Round Robin)

(pure preemptive)

- CPU time is divided into time slices (Time Quantum)
- for every time quantam one process is scheduled

Time quantum = 20

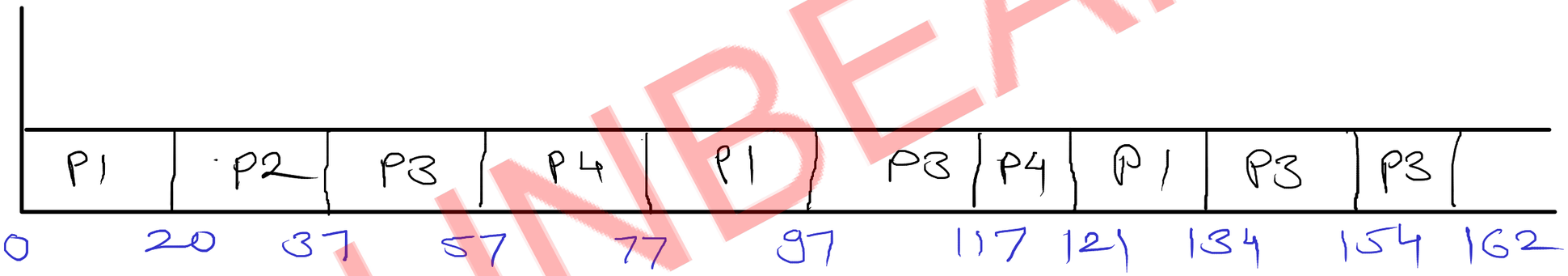
Process	CPU Burst
P1	53
P2	17
P3	68
P4	24

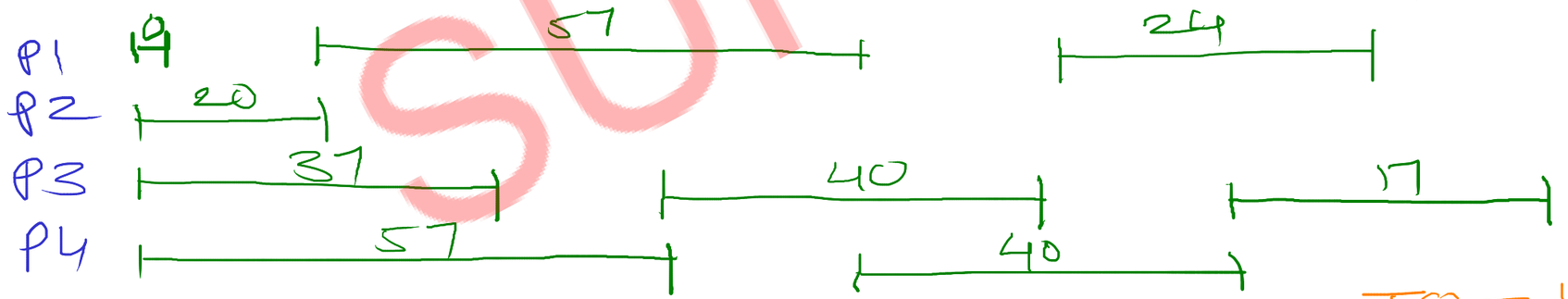
Remain time
P1: 33, 13, 0 ✗
P2: 0 ✗
P3: 48, 28, 8 ✗
P4: 4, 0 ✗

Waiting time
P1: 0 + 57 + 24
P2: 20
P3: 37 + 40 + 17
P4: 57 + 40

Response time
P1: 0
P2: 20
P3: 37
P4: 57

TAT
P1: 134 − 0 = 134
P2: 37 − 0 = 37
P3: 162 − 0 = 162
P4: 121 − 0 = 121





first waiting time is always response time

TQ = 100
↳ behave like FCFS

TQ = 4
↳ CPU overhead will increase

Fair Share

- CPU time is divided into time slices (epoch)
- some share of each epoch is given to the processes which are in ready queue.
- share is given to the process on the basis of their priority
- priority of every process is decided by its nice value
- nice values range ---> -20 to +19 (40 values)
  - * -20 - highest priority
  - * +19 - lowest priority

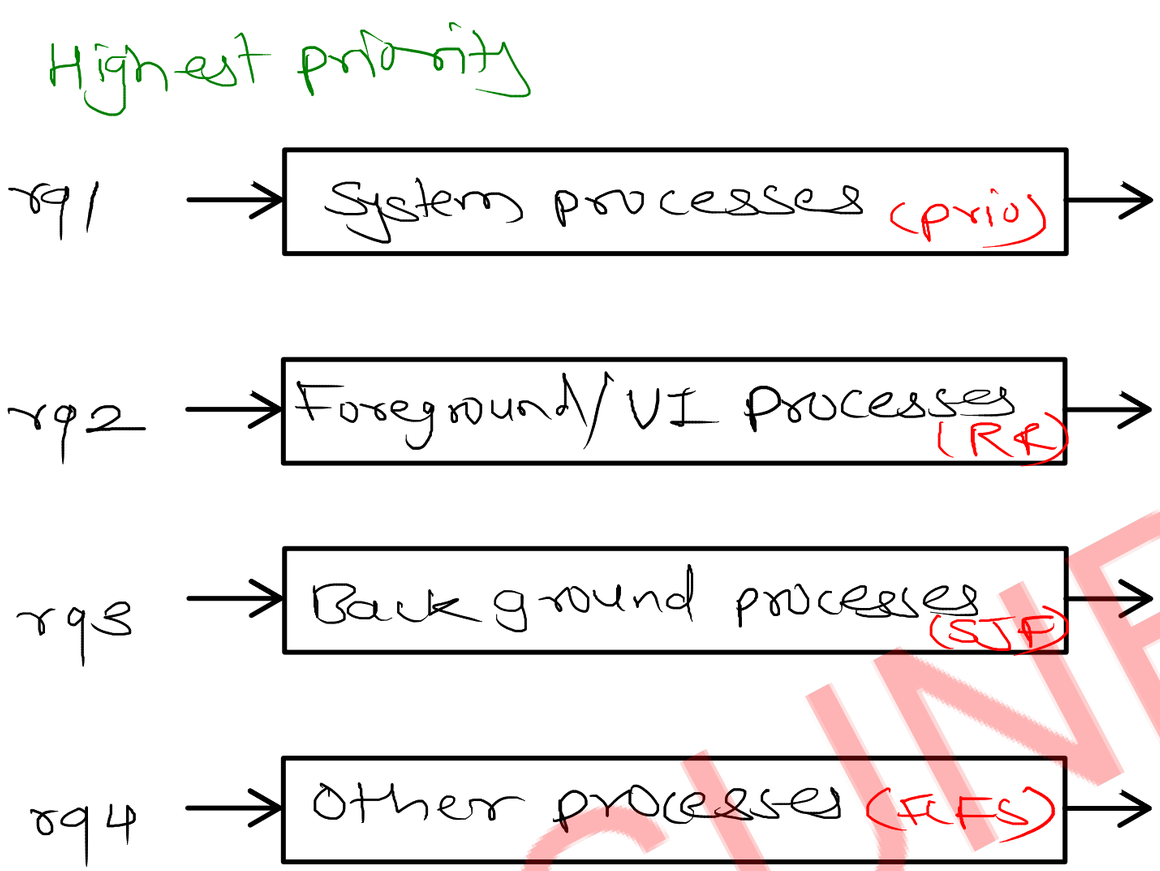
Process	Nice Value
P1	10
P2	10
P3	10
P4	10

Epoch - 100

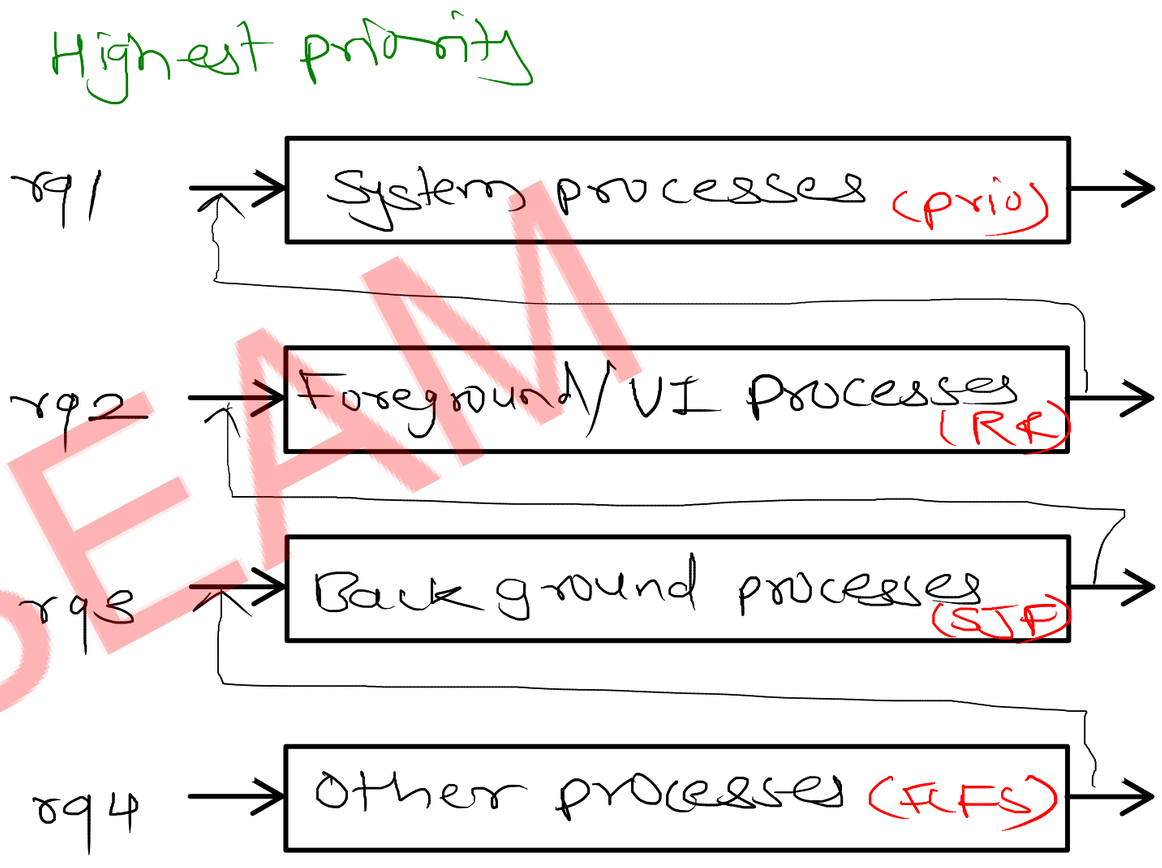
Process	Nice Value
P1	5
P2	5
P3	10
P4	10

1. ps -e -o pid,ppid,ni,cmd - to see the nice value
2. nice -n 10 ./demo01.out - to assign nice value to the process (decrease prio)
3. sudo nice -n -10 ./demo01.out - to assign nice value to the process (increase prio)
4. renice -n 10 -p <pid> - to change nice value of process at runtime

Multi level Ready Queue



Multi level Feedback Ready Queue



- fixed priority preemptive scheduling

Latencies

Interrupt latency - **time from arrival of interrupt to start of ISR execution**

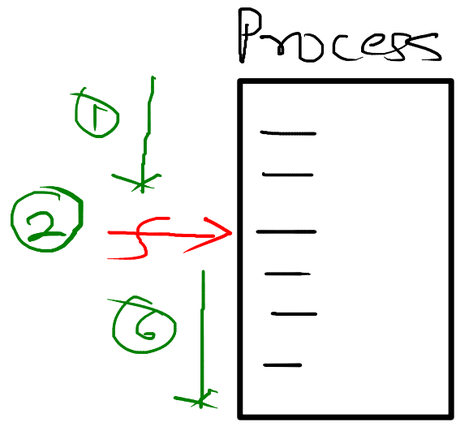
Scheduler latency - **time from completion of ISR to start of scheduler**

Kernel Response Time (Scheduling Latency)

- time to start next process / task

= Interrupt latency + ISR duration + Scheduler latency + Scheduler duration

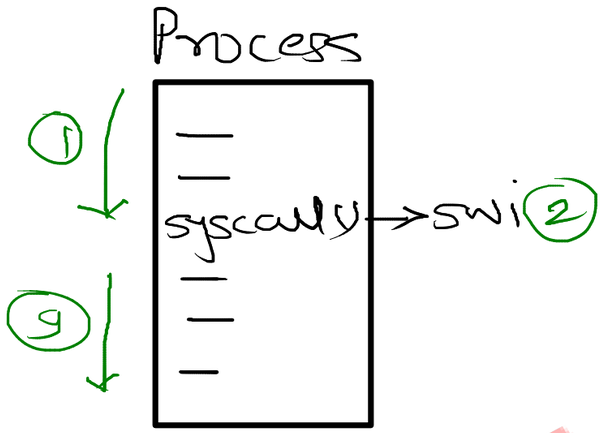
User Space Preemption

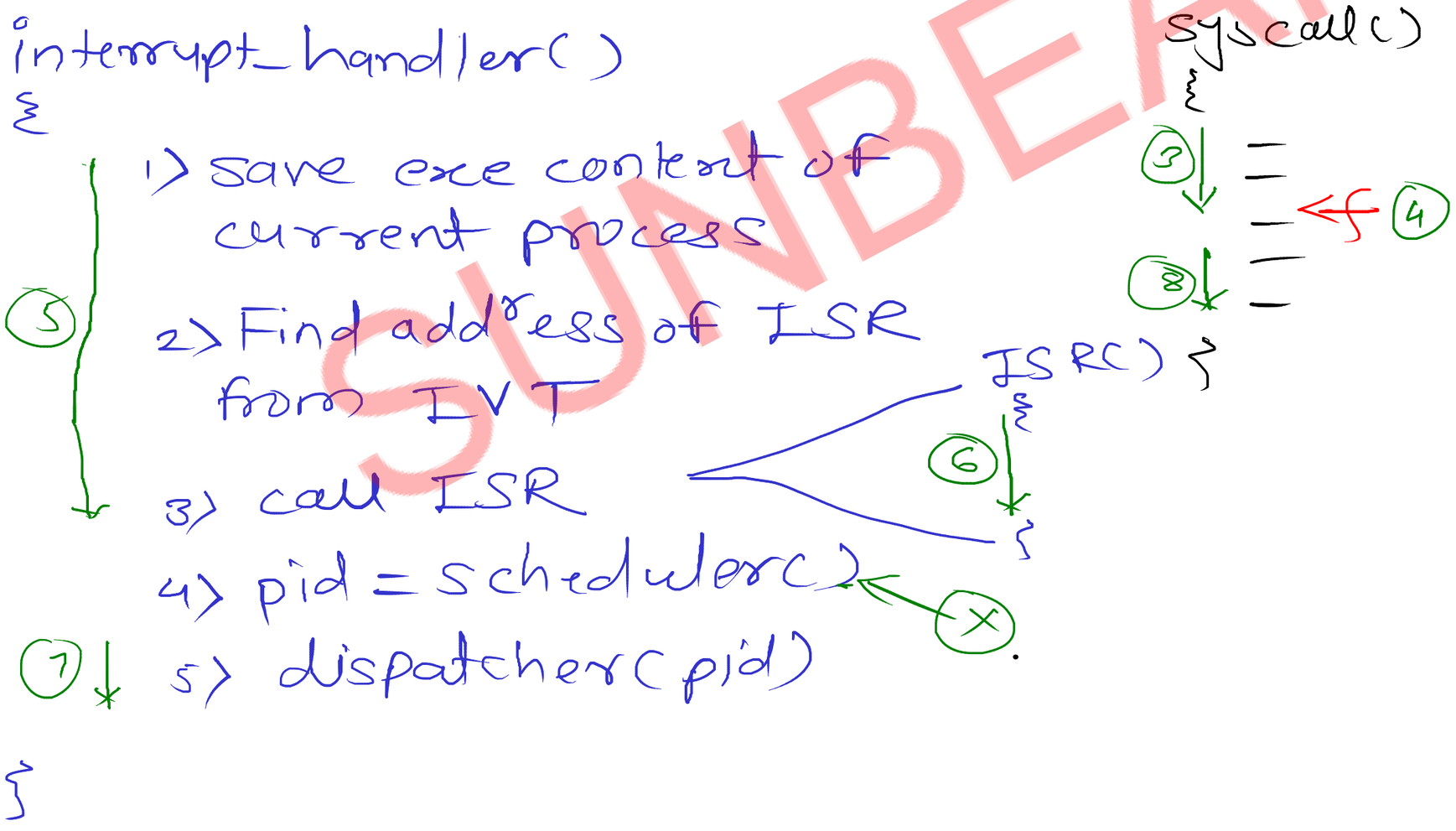



```
interrupt_handler()
{
    1> save exe context of
       current process
    2> Find address of ISR
       from IVT
    3> call ISR
    4> pid = scheduler()
    5> dispatcher(pid)
}
```

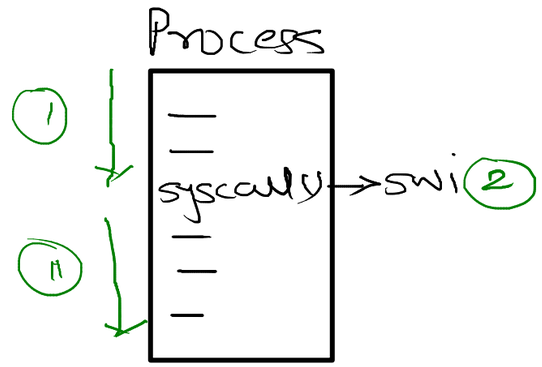
ISR()
{
}

Kernel Space Preemption



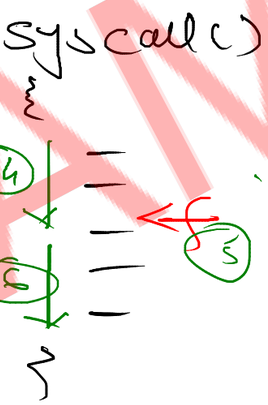


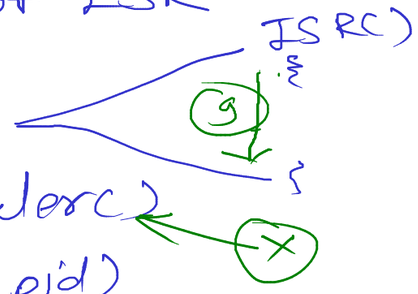
User Space Preemption



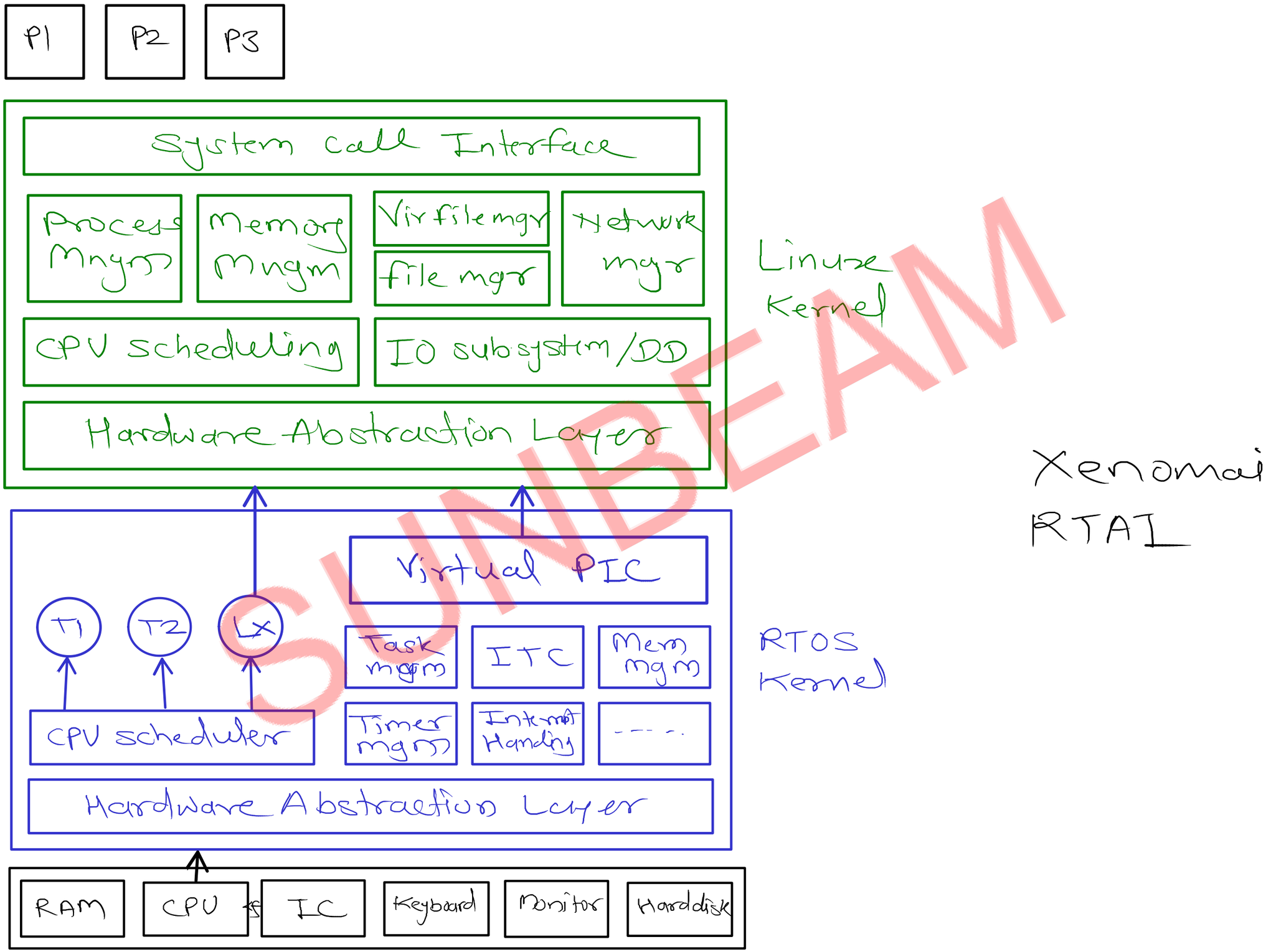
swi_handler()
{
3 - get system call number;
- find actual system implemention
disable_intr()
- syscall()
7 enable_intr()
- pid = scheduler();
- dispatcher();
}

interrupt_handler()
{
8 1> save exe context of current process
2> Find address of ISR from IVT
3> call ISR
4> pid = scheduler()
5> dispatcher(pid)
}

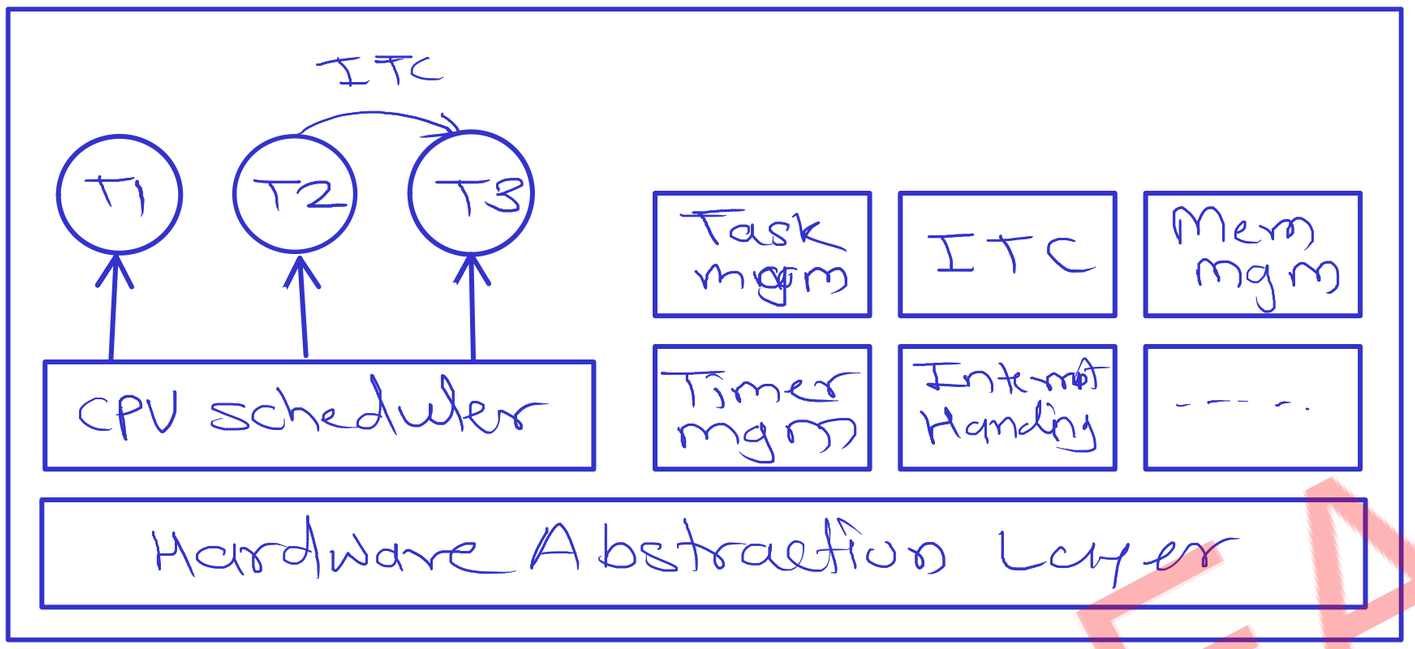




Dual Kernel Approach
(Linux Based RTOS)



Embedded RTOS



Embedded RTOS Kernel

FreeRTOS
VxWorks
QNX
uCOS