Lecture 11

Recap Question

For each of the following thread state transitions, say whether the transition is legal *and* how the transition occurs or why it cannot.

i) Change from thread state BLOCKED to thread state RUNNING

ii) Change from thread state RUNNING to thread state BLOCKED

iii) Change from thread state RUNNABLE to thread state BLOCKED

(Ready)

https://www.cs.cornell.edu/courses/cs4410/2008fa/oldexams/midterm-solutions.pdf

Recap Question

For each of the following thread state transitions, say whether the transition is legal *and* how the transition occurs or why it cannot.

i) Change from thread state BLOCKED to thread state RUNNING ✗

ii) Change from thread state RUNNING to thread state BLOCKED ✓

iii) Change from thread state RUNNABLE to thread state BLOCKED ✗
(Ready)

https://www.cs.cornell.edu/courses/cs4410/2008fa/oldexams/midterm-solutions.pdf

Answer

For each of the following thread state transitions, say whether the transition is legal *and* how the transition occurs or why it cannot.

i) Change from thread state BLOCKED to thread state RUNNING

Illegal. The scheduler selects threads to run from the list of ready (or runnable) threads. A blocked thread must first be placed in the ready queue before it can be selected to run.

ii) Change from thread state RUNNING to thread state BLOCKED

Legal. A running thread can become blocked when it requests a resource that is not immediately available (disk I/O, lock, etc).

iii) Change from thread state RUNNABLE to thread state BLOCKED

Illegal. A thread can only transition to BLOCKED from RUNNING. It cannot execute any statements when still in a queue (i.e. the ready queue).

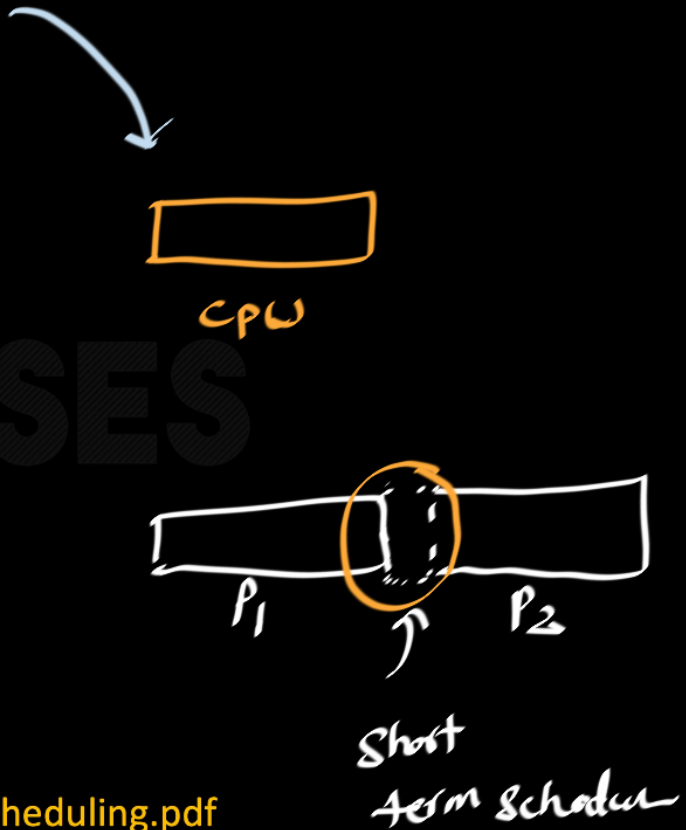
First-Come First-Serve (FCFS)

- By far the simplest CPU-scheduling algorithm

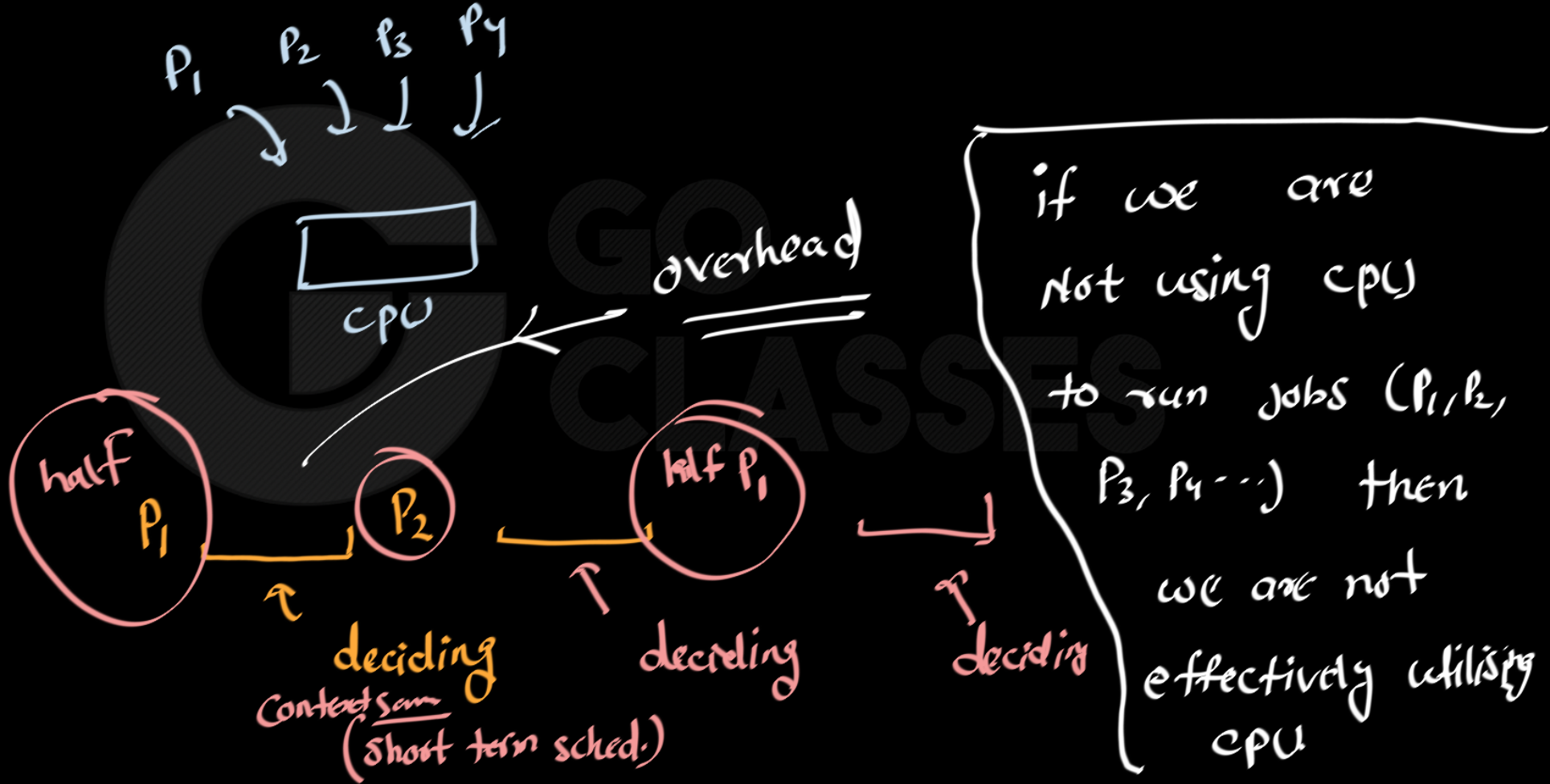
Process	Burst Time
P_1	24
P_2	3
P_3	3

FCFS vs Objective Functions

- Maximize CPU Utilization: Excellent
- Maximize Throughput: Highly dependent on first submitted job(s) duration
- Minimize Turnaround Time: Highly dependent on first submitted job(s) duration
- Minimize Waiting Time: Highly dependent on first submitted job(s) duration
- Minimize Response Time: Highly dependent on first submitted job(s) duration



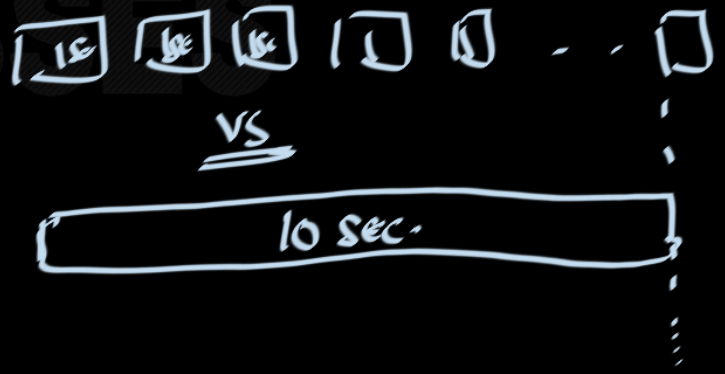
http://courses.ics.hawaii.edu/ReviewICS332/morea/060_Scheduling/ics332_scheduling.pdf


P1 P2 P3 P4
CPU
overhead
half P1
P2
half P1
deciding
Context Sam
(short term sched.)
deciding
deciding
if we are not using CPU to run jobs (P1, P2, P3, P4 ...) then we are not effectively utilising CPU

FCFS vs Objective Functions

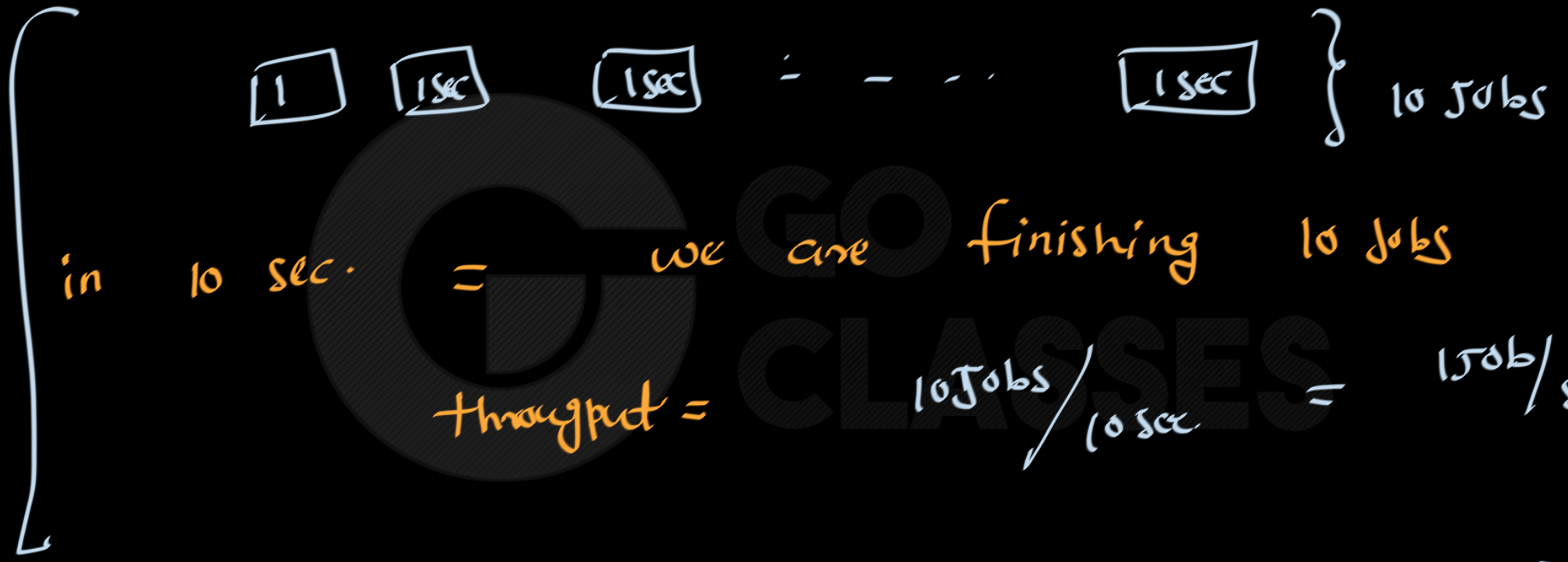
more clear in RR scheduling

- Maximize CPU Utilization: Excellent
- Maximize Throughput: Highly dependent on first submitted job(s) duration
- Minimize Turnaround Time: Highly dependent on first submitted job(s) duration
- Minimize Waiting Time: Highly dependent on first submitted job(s) duration
- Minimize Response Time: Highly dependent on first submitted job(s) duration



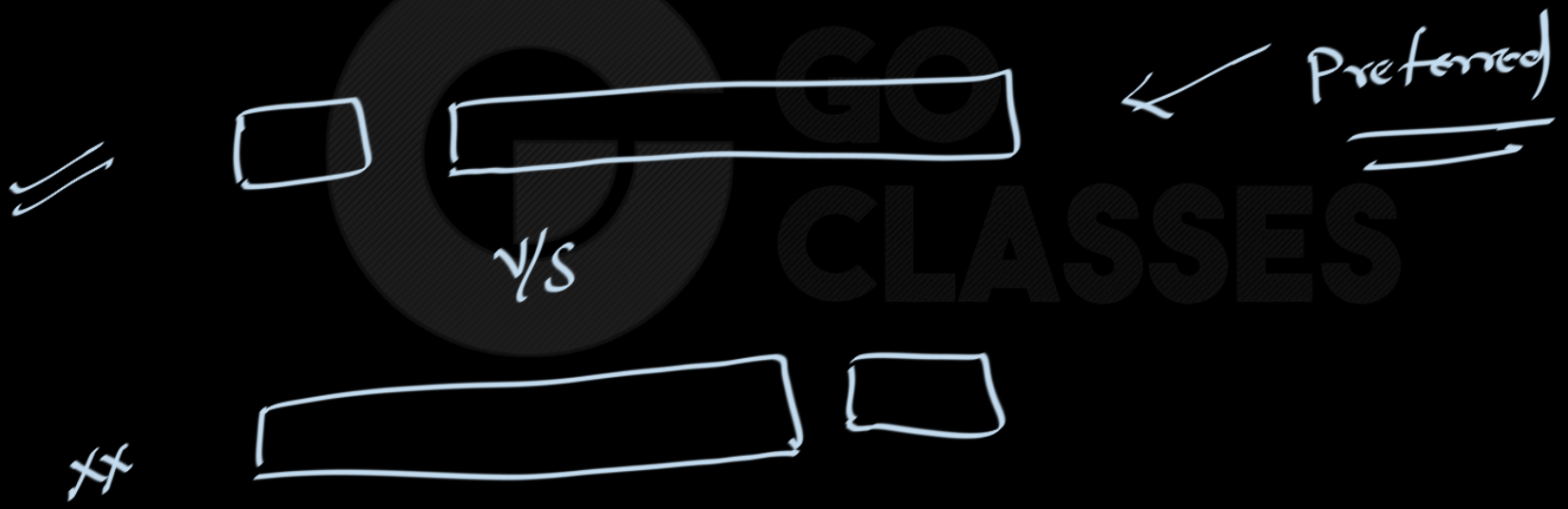
http://courses.ics.hawaii.edu/ReviewICS332/morea/060_Scheduling/ics332_scheduling.pdf

GO CLASSES



throughput = 1Job/10sec = 1/10 Jobs

Shortest Job First (SJF)



Shortest-Job-First Scheduling (SJF)

- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.

- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart

- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Non-preemptive SJF: Example

Process	Duration	Arrival Time
P1	6	0
P2	8	0
P3	7	0
P4	3	0

Do it yourself

P4 waiting time: 0
P1 waiting time: 3
P3 waiting time: 9
P2 waiting time: 16

The total time is: 24
The average waiting time (AWT):
(0+3+9+16)/4 = 7

Comparing to FCFS

Process	Duration	Arrival Time
P1	6	0
P2	8	0
P3	7	0
P4	3	0

Do it yourself

P1 waiting time: 0
P2 waiting time: 6
P3 waiting time: 14
P4 waiting time: 21

The total time is the same.
The average waiting time (AWT):
(0+6+14+21)/4 = 10.25
(compared to 7)

Predicting length of the next CPU burst

Let T_n be the length of the n^{th} CPU burst, and let S_{n+1} be our predicted value for the next CPU burst.

We define,

$$S_{n+1} = \alpha\,(T_n) + (1 - \alpha) * S_n$$

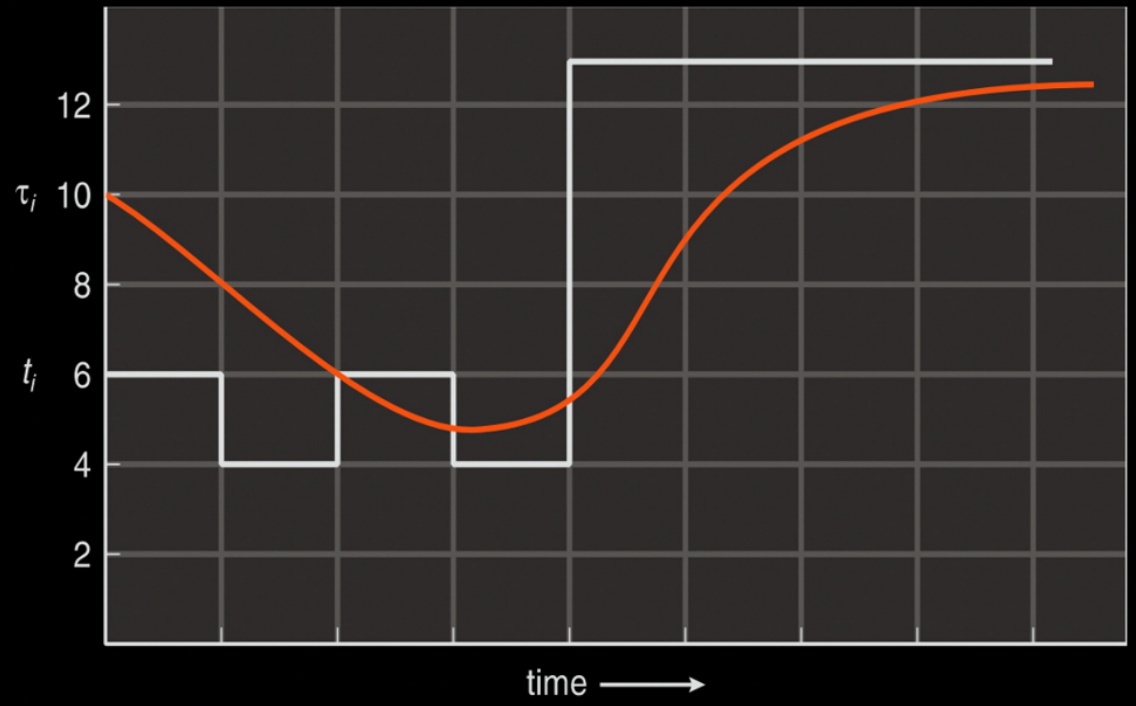
The value of T_n contains our most recent information, while S_n stores the past history.

- if α = 0, then S_{n+1} = S_n, and recent history has no effect.
- if α = 1, then S_{n+1} = T_n, and only the most recent CPU burst matters (history is assumed to be old and irrelevant).
- More commonly, α = 1/2, so recent history and past history are equally weighted.

To understand the behavior of the exponential average, we can expand the formula for S_{n+1} by substituting for S_n to find

$$S_{n+1} = \alpha T_n + (1-\alpha)\alpha T_{n-1} + \cdots + (1-\alpha)^j \alpha T_{n-j} + \cdots + (1-\alpha)^{n+1} S_0$$

Typically, α is less than 1 . As a result, $(1-\alpha)$ is also less than 1 , and each successive term has less weight than its predecessor.



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Question

we initially had jobs with actual run times 8, 7, 4, and 16. ← Arrival time = 0

Assuming no new jobs entered the queue, we'd schedule then in increasing order of their times, namely 4, 7, 8, 16 (since that's what SJF means--shortest job first).

Calculate the exponential averaging S_4 with $S_1 = 10, \alpha = 0.5$ and the algorithm is SJF .

initial guess

$S_n = \frac{1}{2} S_{n-1} + \frac{1}{2} t_{n-1}$

1. 9
2. 8
3. 7.5
4. None

$S_2 =$

Question

we initially had jobs with actual run times 8, 7, 4, and 16. ← Arrival time = 0

Assuming no new jobs entered the queue, we'd schedule then in increasing order of their times, namely 4, 7, 8, 16 (since that's what SJF means--shortest job first). ← real

Calculate the exponential averaging S_4 with $S_1 = 10, \alpha = 0.5$ and the algorithm is SJF . ← initial guess

1. 9
2. 8
3. 7.5
4. None

$S_n = \frac{1}{2} S_{n-1} + \frac{1}{2} t_{n-1}$

$S_2 = \frac{1}{2} 10 + \frac{1}{2} 4 = 7$ $S_3 =$

Answer

$$S_2 = \frac{1}{2}(4 + 10) = 7 \qquad \text{after the first job has run}$$

$$S_3 = \frac{1}{2}(7 + 7) = 7 \qquad \text{after the second job has run}$$

$$S_4 = \frac{1}{2}(8 + 7) = 7.5 \qquad \text{after the third job has run}$$

so the fourth job would be given a slice of 7.5, which is choice (3).

Optional Read from Galvin

The SJF algorithm can be either preemptive or nonpreemptive. The choice arises when a new process arrives at the ready queue while a previous process is still executing. The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process. A preemptive SJF algorithm will preempt the currently executing process, whereas a nonpreemptive SJF algorithm will allow the currently running process to finish its CPU burst. Preemptive SJF scheduling is sometimes called **shortest-remaining-time-first** scheduling.

ROUND ROBIN

GO CLASSES

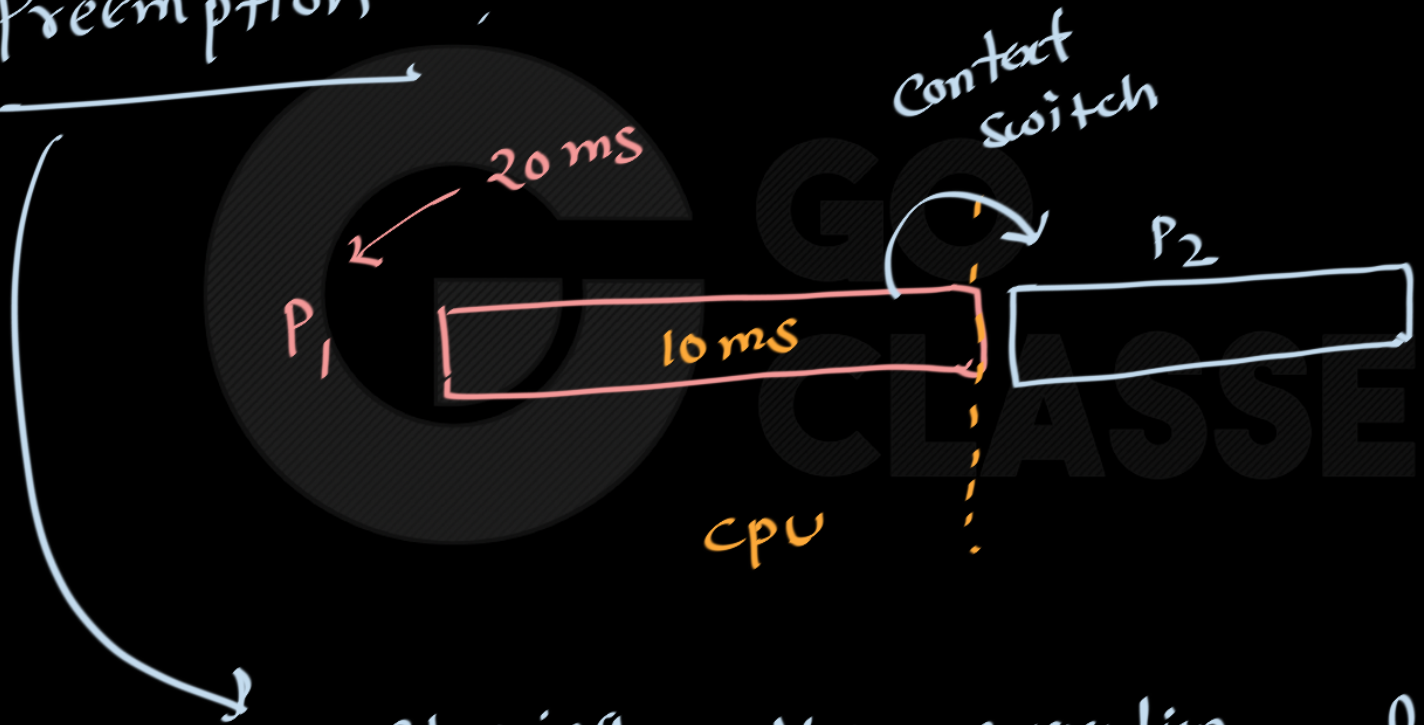
{ FCFS, SJF } ← Non preemptive scheduling algorithms

10ms 10ms

Video game , chrome

GO CLASSES

Preemption :



Stopping the execution of one process and running another process.

GO CLASSES

for preemption, we need to have context switch.

if we keep on switching between video game and chrome browser

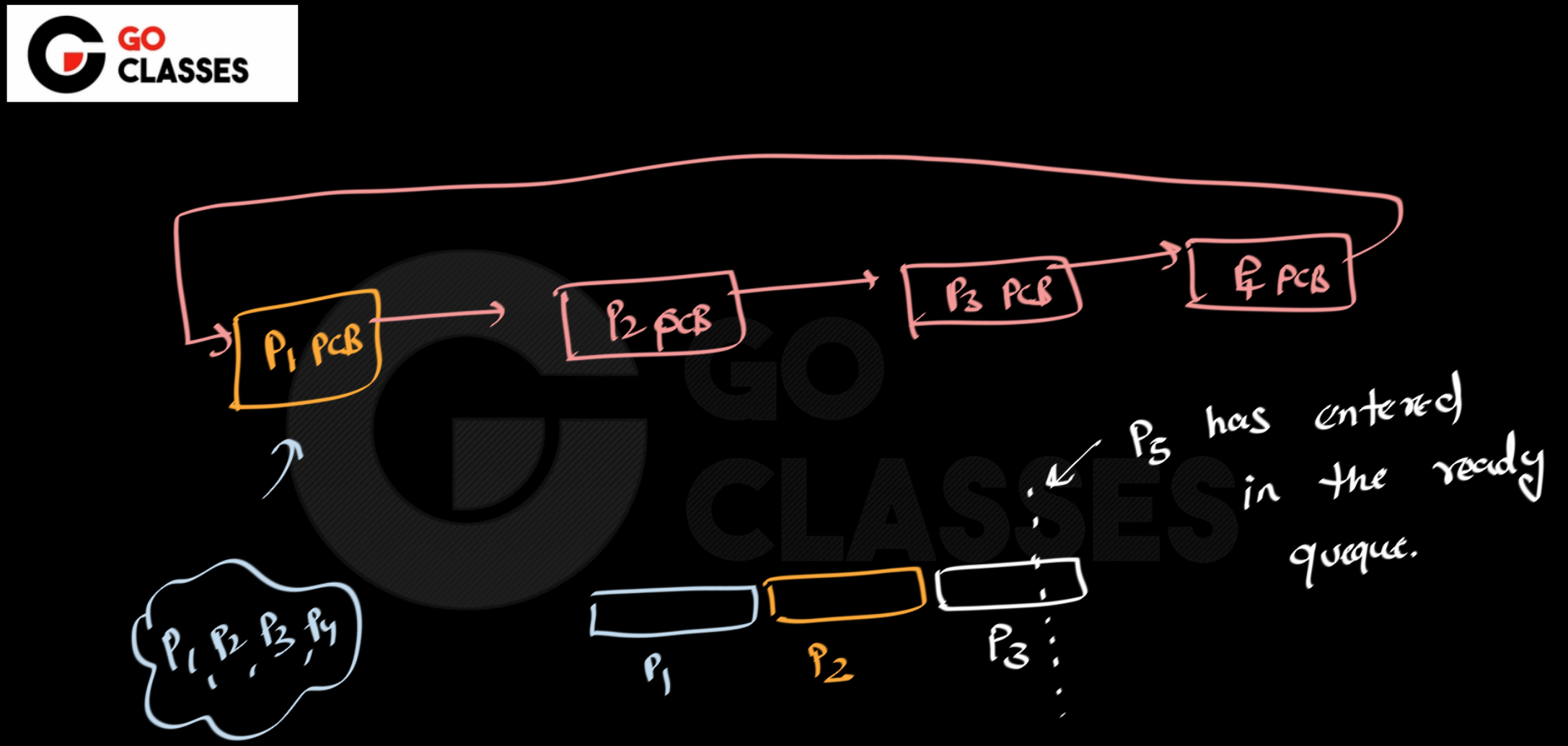
it will provide an illusion to the user that both are running simultaneously

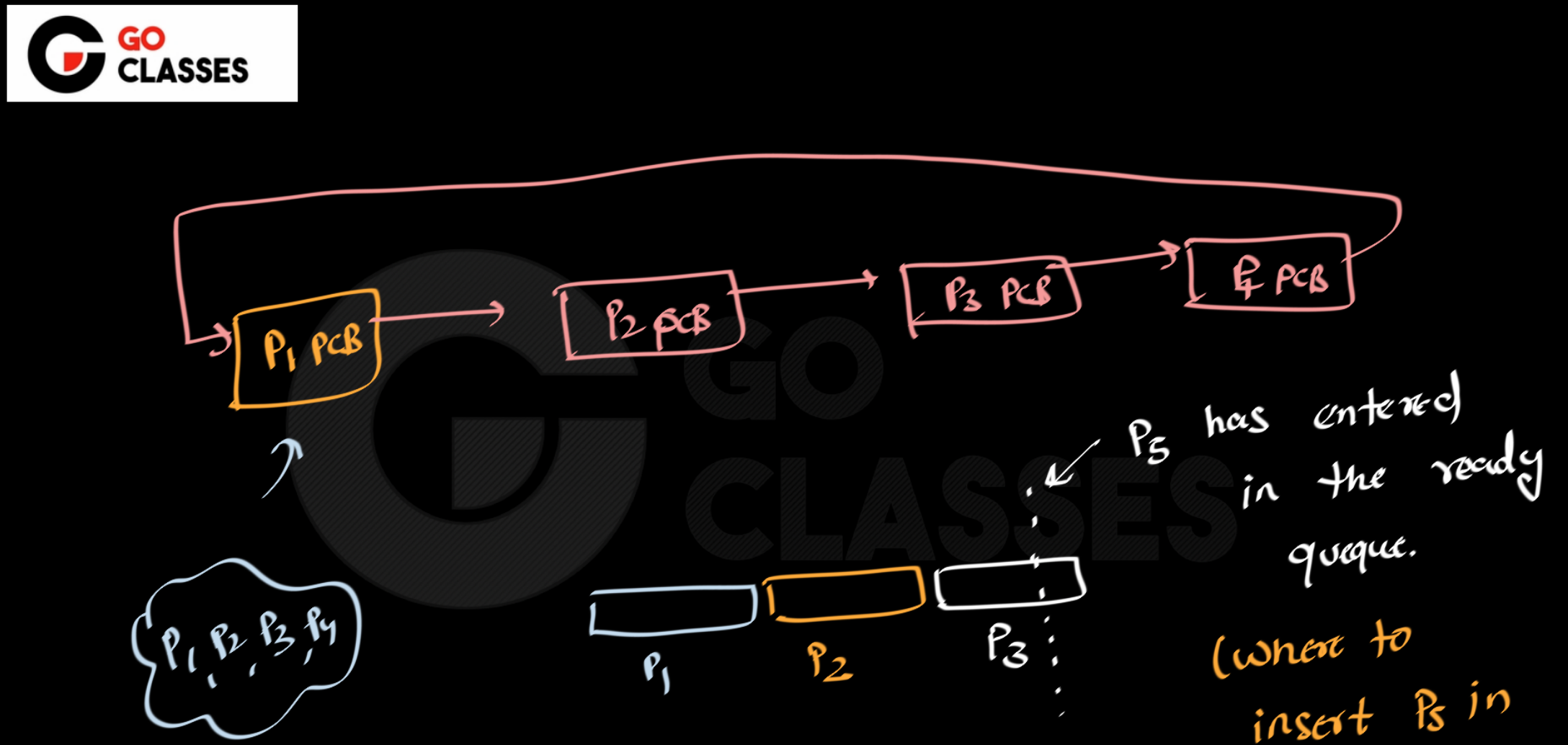
(in some cases it is desirable to switch in between of processes)

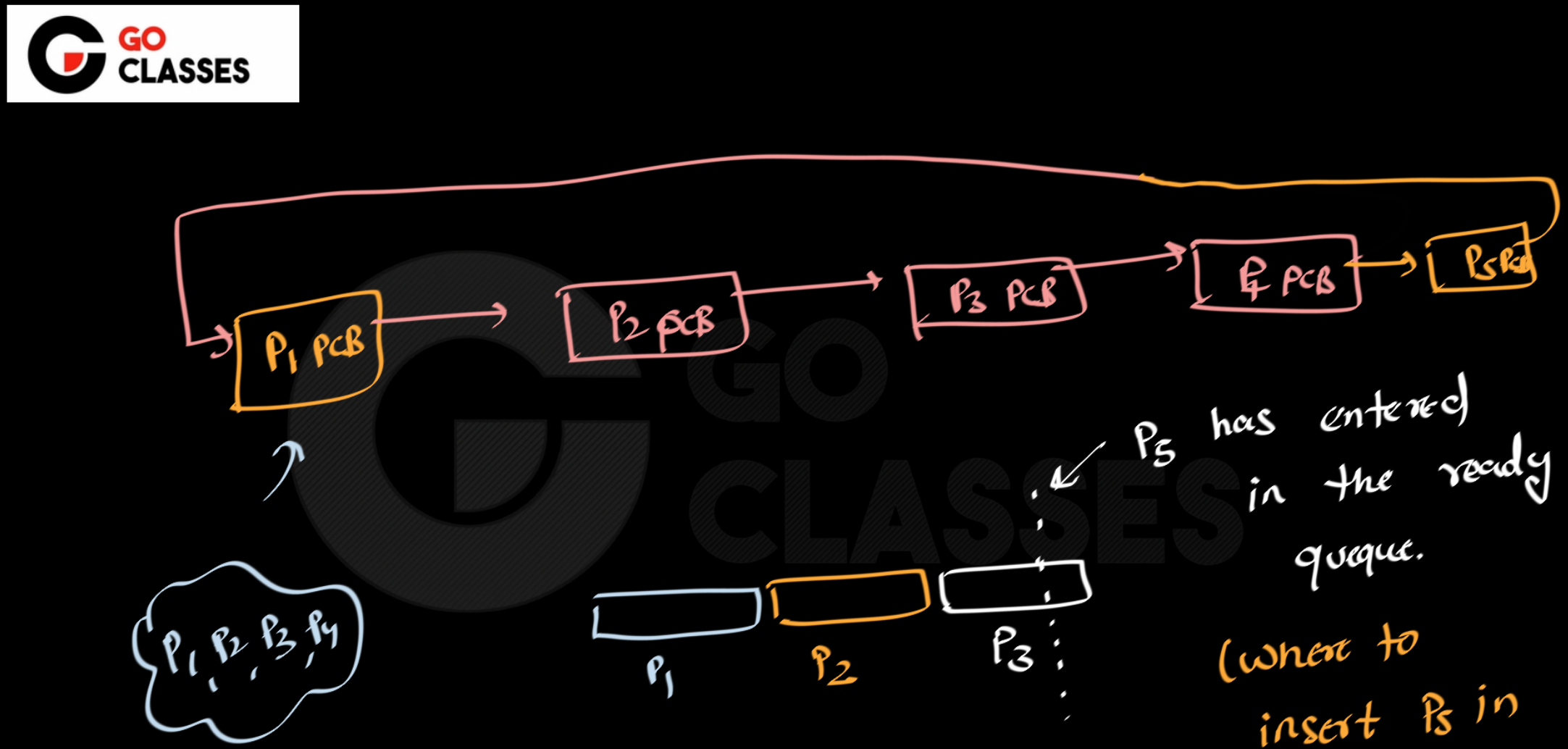
GO CLASSES

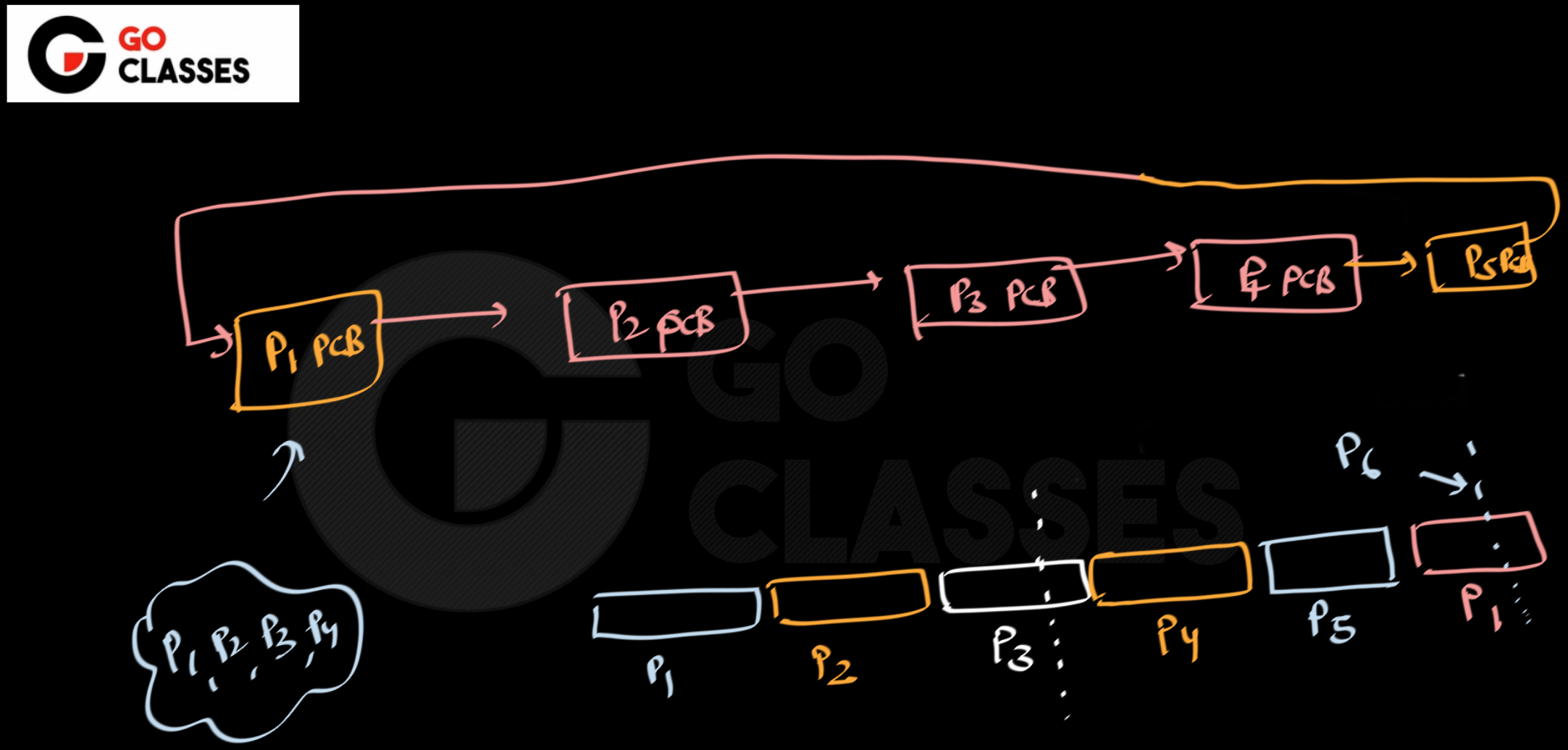
Ready queue

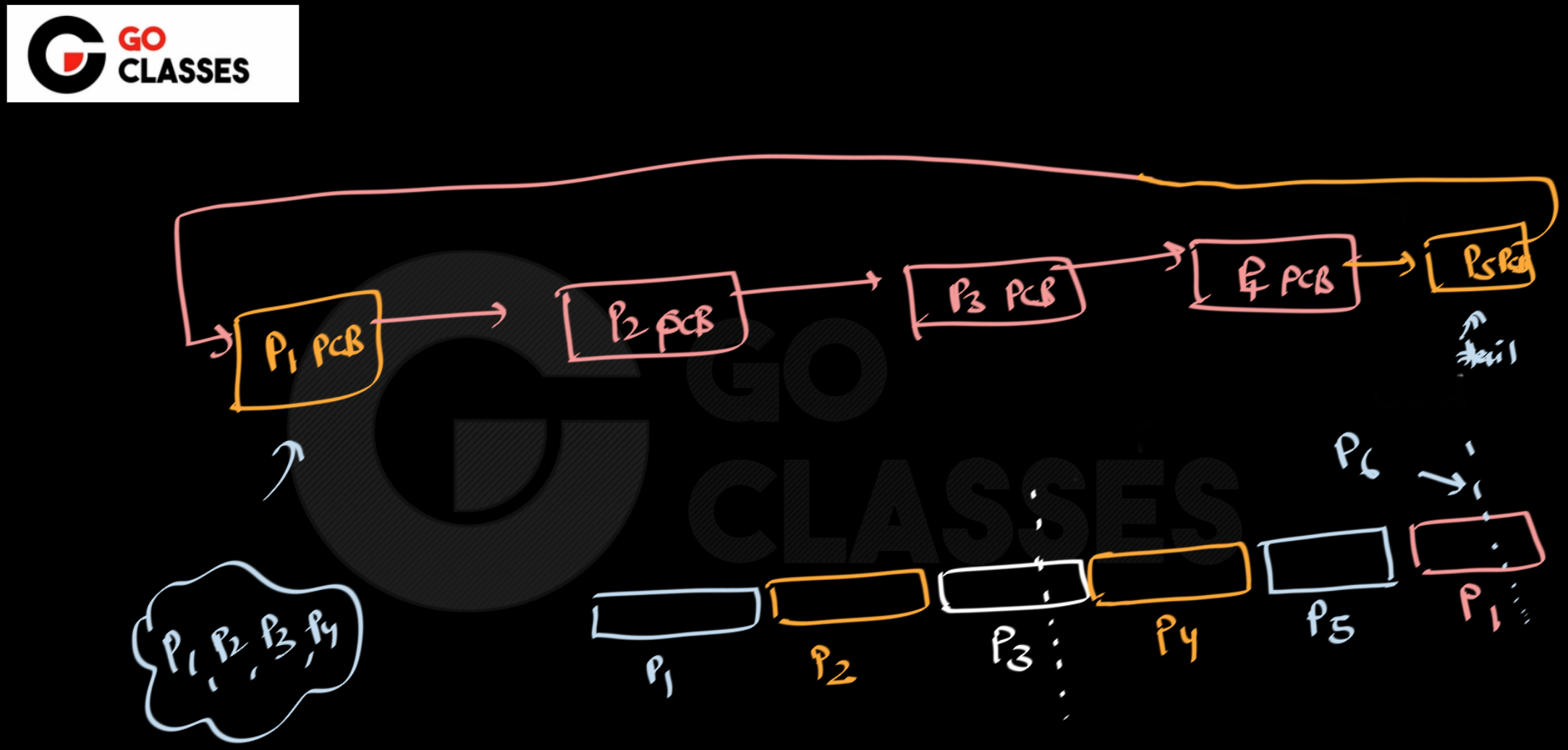
what data structure we can use for processes?

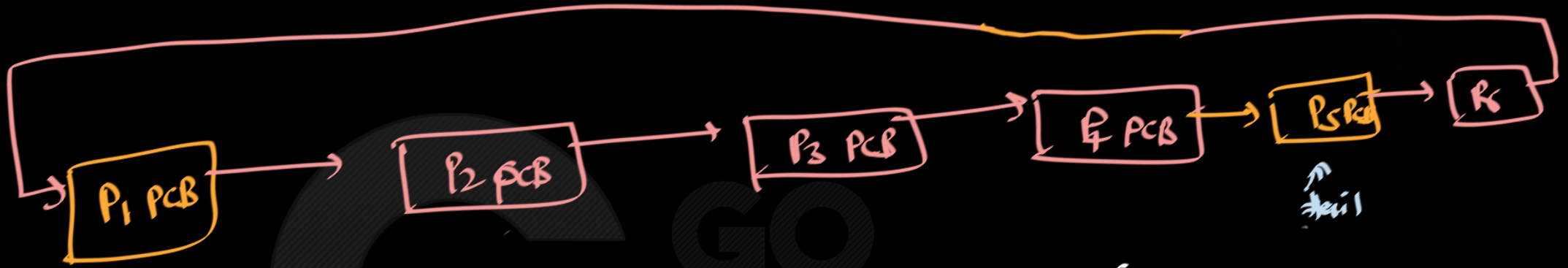

P1 PCB
P2 PCB
P3 PCB
P4 PCB
P1, P2, P3, P4
P1
P2
P3
P3 has entered in the ready queque.


P1 PCB
P2 PCB
P3 PCB
P4 PCB
P1, P2, P3, P4
P1
P2
P3
P3 has entered in the ready queque.
(where to insert P3 in circular LL)

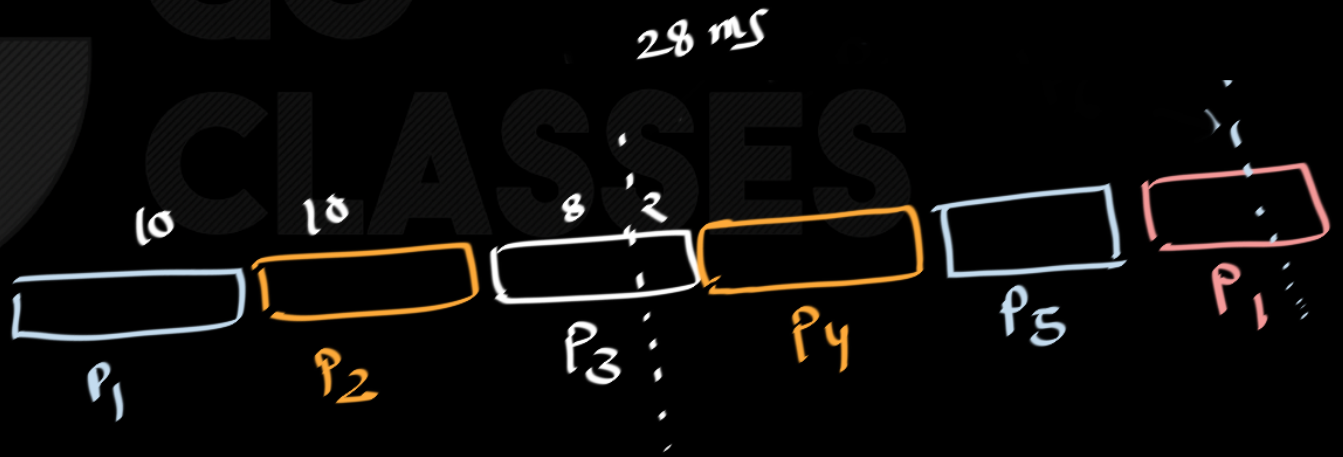

P1 PCB
P2 PCB
P3 PCB
P4 PCB
P5 PCB
P1, P2, P3, P4
P1
P2
P3
P5 has entered in the ready queque.
(where to insert P5 in Circular LL)


GO CLASSES
P1 PCB
P2 PCB
P3 PCB
P4 PCB
P5 PCB
P1, P2, P3, P4
P1
P2
P3
P4
P5
P6
P1


GO CLASSES
P1 PCB
P2 PCB
P3 PCB
P4 PCB
P5 PCB
tail
P1, P2, P3, P4
P1
P2
P3
P4
P5
P6
P1


P1 PCB
P2 PCB
P3 PCB
P4 PCB
P5 PCB
P1
head

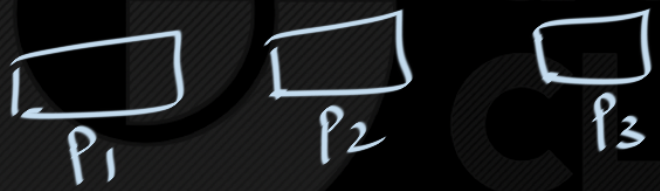

P1, P2, P3, P4


28 ms
10
10
8
2
P1
P2
P3
P4
P5
P1

GO CLASSES

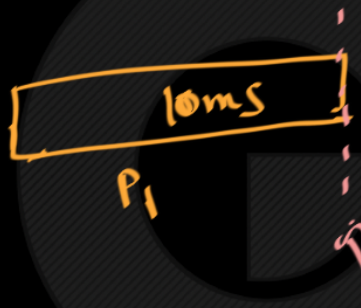
- whatever initial processes are there in the ready queque make a circular list out of that.
- Start execution of processes
- if some process is comming in between the park it at the end of circular list.

Round Robin scheduling



follows "quanta" ≈ this timer is maintained by hardware.

time quanta = 10 ms



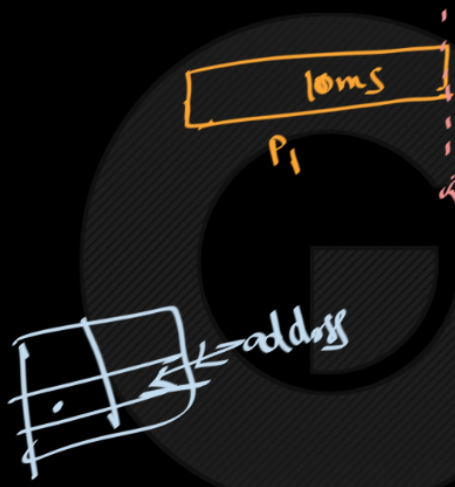
hard ware timer will be triggred

⇒ this is one interrupt

⇒ Interrupt service routine will be checked

⇒ this in turn will lead to short term scheduler code

time quanta = 10 ms



hard ware timer will be triggred

⇒ this is one interrupt

⇒ Interrupt service routine will be checked

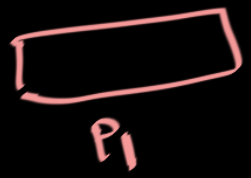
⇒ this in turn will lead to short term scheduler code

→ generally very very fast because decision about new process to be scheduled is very easy.

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

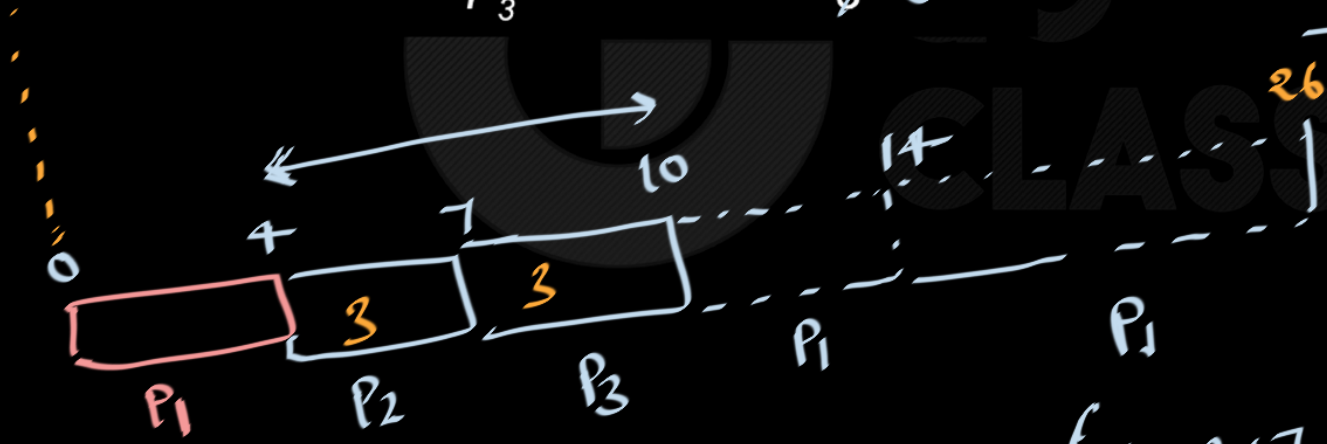
Arrival time = 0



Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24 ~~20~~ ~~16~~
P_2	~~3~~ 0
P_3	~~3~~ 0

Arrival time = 0



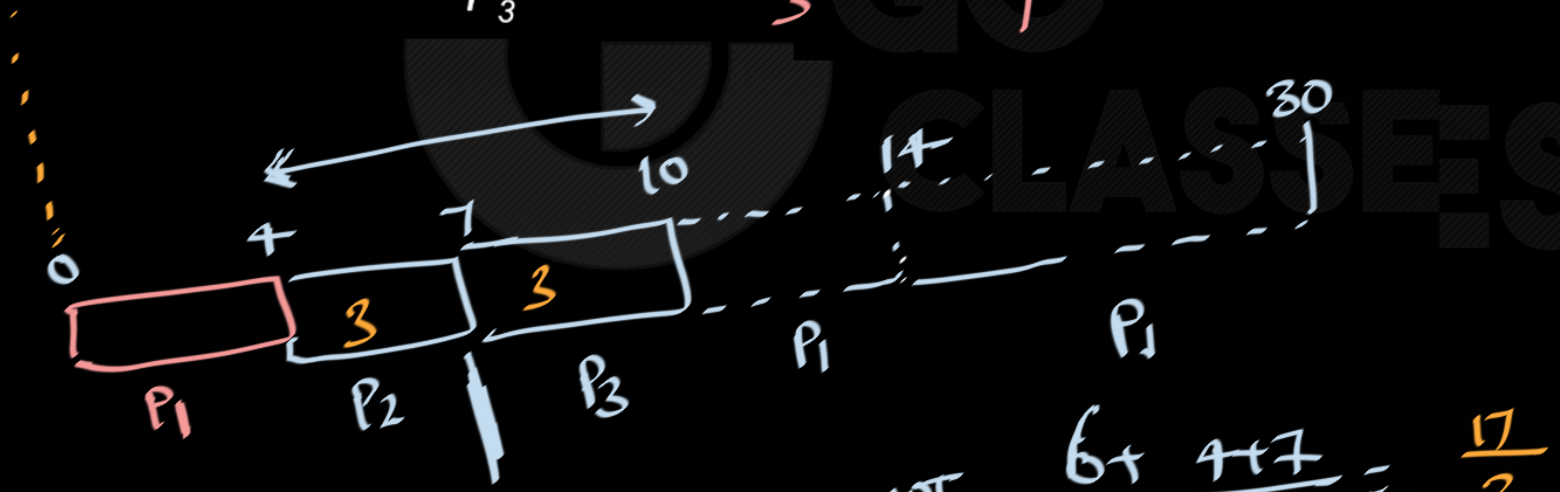
Avg. TAT = 26

↳ CT − AT

$$\text{Avg. WT} = \frac{6 + 4 + 7}{3} = \frac{17}{3}$$

Example of RR with Time Quantum = 4

Process	Burst Time	AT (given)	CT	TAT (CT − AT)	WT
P_1	24	0	30	30 − 0 = 30	6
P_2	3	0	7	7 − 0 = 7	4
P_3	3	0	10	10 − 0 = 10	7

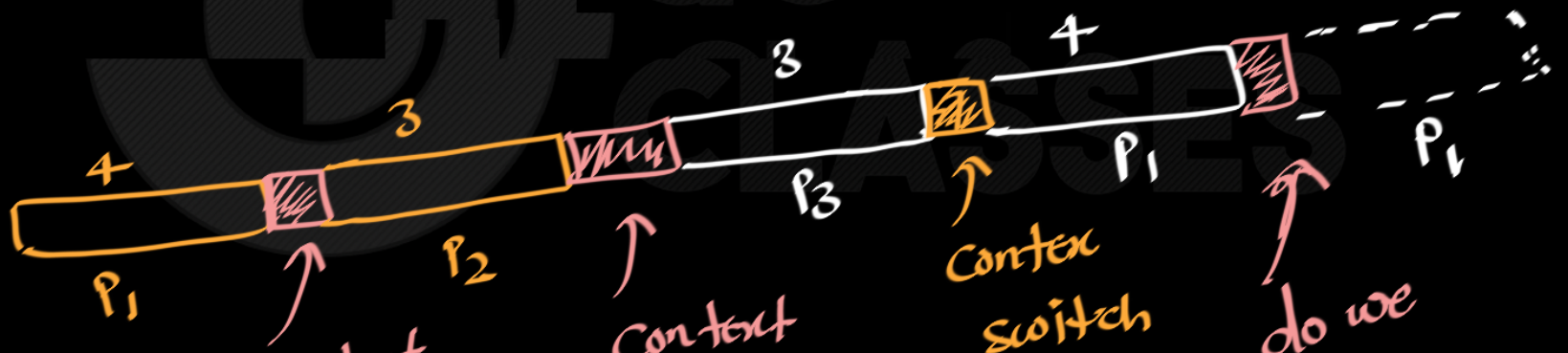


total time Process locs in system

$$\text{Avg. WT} = \frac{6 + 4 + 7}{3} = \frac{17}{3}$$

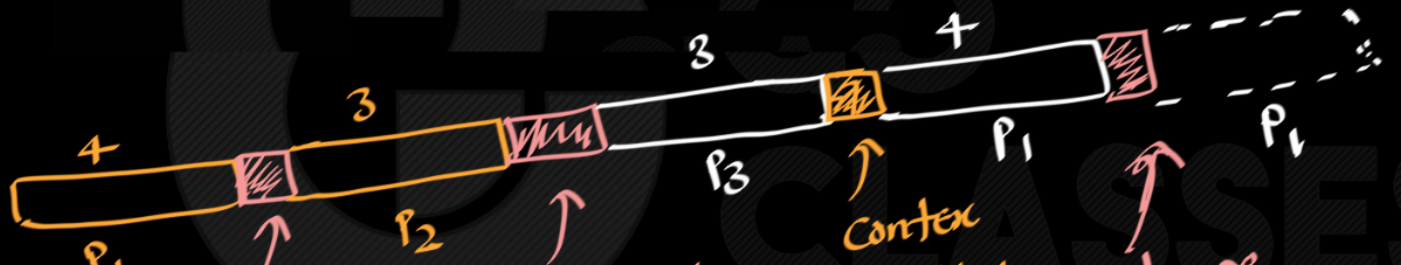
Example of RR with Time Quantum = 4

Process	Burst Time	AT (given)	CT	TAT (CT − AT)	WT
P_1	24	0	30	30 − 0 = 30	6
P_2	3	0	7	7 − 0 = 7	4
P_3	3	0	10	10 − 0 = 10	7



Example of RR with Time Quantum = 4

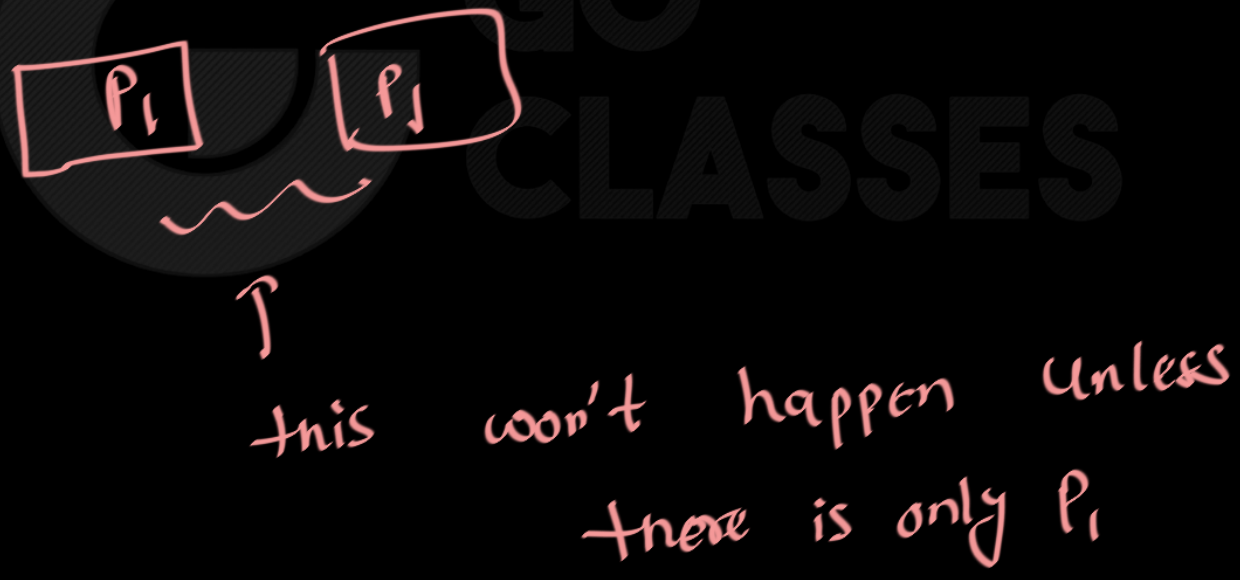
Process	Burst Time	AT (given)	CT	TAT (CT − AT)	WT
P_1	24	0	30	30 − 0 = 30	6
P_2	3	0	7	7 − 0 = 7	4
P_3	3	0	10	10 − 0 = 10	7



Should depend on implementation

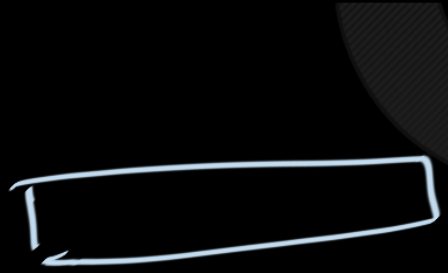
{ If Ready queue does not have any process then we can bypass (skip) the Context Switch. }

- In the RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row (unless it is the only runnable process).



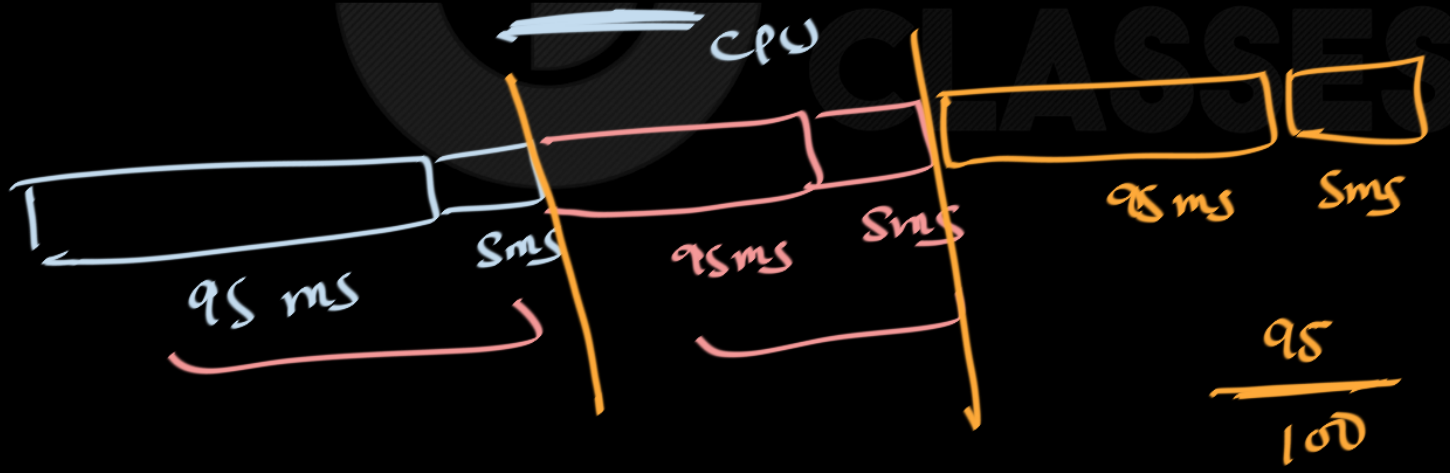
Question

A system uses a Round Robing scheduling algorithm with a time slice of 95 millisecond; the context switch time is 5 milliseconds. What is the average processor utilization?



Question

A system uses a Round Robing scheduling algorithm with a time slice of 95 millisecond; the context switch time is 5 milliseconds. What is the average processor utilization?



Answer

A system uses a Round Robing scheduling algorithm with a time slice of 95 millisecond; the context switch time is 5 milliseconds. What is the average processor utilization?

$$U = \frac{UsefulTime}{TotalTime} = \frac{95}{95+5} = 95\%.$$

x100

↳ cpu is working effectively

time quantum = δ ms

Context switch time = γ ms

$$\text{CPU utilization} = \frac{\text{useful work time}}{\text{total time}} = \frac{\delta}{\delta+\gamma}$$

time quantum = δ ms

Context switch time = γ ms

$$\text{CPU utilization} = \frac{\text{useful work time}}{\text{total time}} = \frac{\delta}{\delta+\gamma}$$

$$\frac{\delta}{\delta+\gamma} \rightsquigarrow 1 \quad \left(\text{will happen when } \gamma \rightsquigarrow 0\right)$$

time quantum = δ ms

context switch time = γ ms

$$\text{cpu utilization} = \frac{\text{useful work time}}{\text{total time}} = \frac{\delta}{\delta+\gamma}$$

Suppose, $\gamma = \delta$

cpu utilization = 50%.

$$\frac{\delta}{\delta+\gamma} = \frac{\delta}{2\delta} = \frac{1}{2}$$

time quantum = δ ms

Context switch time = γ ms

$$\text{CPU utilization} = \frac{\text{useful work time}}{\text{total time}} = \frac{\delta}{\delta+\gamma}$$

$$\frac{\delta}{\delta+\gamma} \rightsquigarrow 1 \quad \left(\text{will happen when } \delta >> \gamma\right)$$

γ we can almost ignore γ in term $\delta+\gamma$

$10000 + 0.001 \approx 10000$

this is very less compare to first number (10000) then we can ignore this

GO CLASSES

$q = 10000$ ms

as good as fcfs

for this process there will be the preemption.

- q large $\Rightarrow$ FIFO
- q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

No mathematical explanation,

RR advantages and disadvantages

- Advantages
  - Low response time, good interactivity
  - Fair allocation of CPU across processes
  - Low average waiting time when job lengths vary widely
- Disadvantages
  - Poor average waiting time when jobs have similar lengths
    - Average waiting time is even worse than FCFS!
  - Performance depends on length of time slice
    - Too high ➔ degenerate to FCFS
    - Too low ➔ too many context switches, costly

http://www.cs.columbia.edu/~jae/4118-LAST/L15-scheduling-1.pdf

Round-Robin Scheduling

- RR Scheduling is preemptive and designed for time-sharing
- It defines a time quantum
  - A fixed interval of time (10-100ms)
- Unless a process is the only READY process, it never runs for longer than a time quantum before giving control to another ready process
  - It may run for less than the time quantum if its CPU burst is smaller than the time quantum
- Ready Queue is a FIFO
  - Whenever a process changes its state to READY it is placed at the end of the FIFO
- Scheduling:
  - Pick the first process from the ready queue
  - Set a timer to interrupt the process after 1 quantum
  - Dispatch the process

http://courses.ics.hawaii.edu/ReviewICS332/morea/060_Scheduling/ics332_scheduling.pdf

Picking the Right Quantum

quantum	context switches
12	0
6	1
1	9

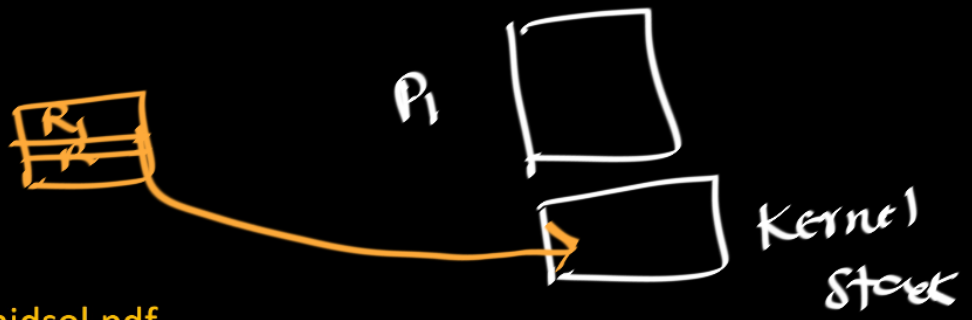
- Trade-off:
  - Short quantum: great response/interactivity but high overhead
    - Hopefully not too high if the dispatcher is fast enough
  - Long quantum: poor response/interactivity, but low overhead
    - With a very long time quantum, RR Scheduling becomes FCFS Scheduling
- If context-switching time is 10% of time quantum, then the CPU spends >10% of its time doing context switches
- In practice, %CPU time spent on switching is very low
  - time quantum: 10ms to 100ms
  - context-switching time: 10 μs

Question

Consider a system with two processes, P_1 and P_2. Process P_1 is running and process P_2 is ready to run. The operating system uses preemptive round robin scheduling. Consider the situation in which P_1's scheduling quantum is about to expire, and P_2 will be selected to run next.

List the sequence of events that will occur in this situation, in the order in which they will occur. Create your list by choosing from the events shown below, using the event numbers to identify events. For example, a valid answer might be: "7,5,4,8,5,2". If an event occurs more than once, you may include it more than once in your list.

1. P_1's user-level state is saved into a trap frame
2. P_2's user-level state is saved into a trap frame
3. a timer interrupt occurs
4. the operating system's scheduler runs
5. there is a context switch from thread T_1 (from process P_1) to thread T_2 (from process P_2).
6. P_1's user-level state is restored from a trap frame
7. P_2's user-level state is restored from a trap frame
8. a "system call" instruction is executed
9. thread T_2 (for process P_2) is created
10. thread T_1 (from process P_1 is destroyed



https://student.cs.uwaterloo.ca/~cs350/common/old-exams/W08midsol.pdf

Question

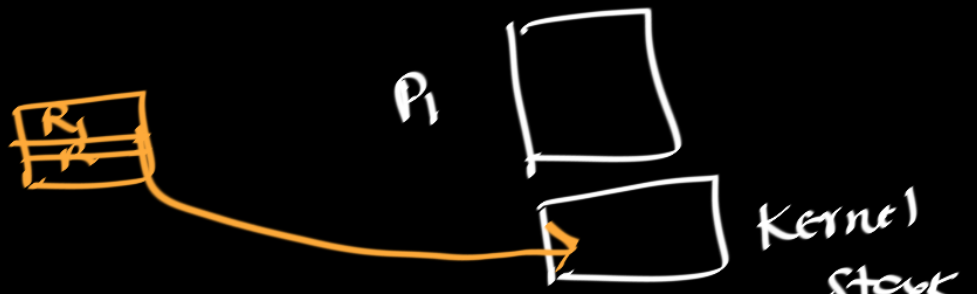
H.W.

Consider a system with two processes, P_1 and P_2. Process P_1 is running and process P_2 is ready to run. The operating system uses preemptive round robin scheduling. Consider the situation in which P_1's scheduling quantum is about to expire, and P_2 will be selected to run next.

List the sequence of events that will occur in this situation, in the order in which they will occur. Create your list by choosing from the events shown below, using the event numbers to identify events. For example, a valid answer might be: "7,5,4,8,5,2". If an event occurs more than once, you may include it more than once in your list.

1. P_1's user-level state is saved into a trap frame
2. P_2's user-level state is saved into a trap frame
3. a timer interrupt occurs
4. the operating system's scheduler runs
5. there is a context switch from thread T_1 (from process P_1) to thread T_2 (from process P_2).
6. P_1's user-level state is restored from a trap frame
7. P_2's user-level state is restored from a trap frame
8. a "system call" instruction is executed
9. thread T_2 (for process P_2) is created
10. thread T_1 (from process P_1 is destroyed

whatever we put in Kernel Stack



https://student.cs.uwaterloo.ca/~cs350/common/old-exams/W08midsol.pdf

Write your answer here:

3,1,4,5,7

Question	True/False

1 If all jobs have identical run lengths, a RR scheduler (with a time-slice much shorter than the jobs' run lengths) provides better average turnaround time than FIFO.

2 The longer the time slice, the more a RR scheduler gives similar results to a FIFO scheduler.

https://pages.cs.wisc.edu/~dusseau/Classes/CS537/Fall2016/Exams/f16-e2-answers.pdf

4) If all jobs have identical run lengths, a RR scheduler (with a time-slice much shorter than the jobs' run lengths) provides better average turnaround time than FIFO.

False – with RR, identical jobs will all finish at nearly the same time (at the very end of the workload time), which has very poor average turnaround time.

5) The longer the time slice, the more a RR scheduler gives similar results to a FIFO scheduler.

True – In the extreme, when the time slice is >= the length of the job, RR degenerates to FIFO (or FCFS).

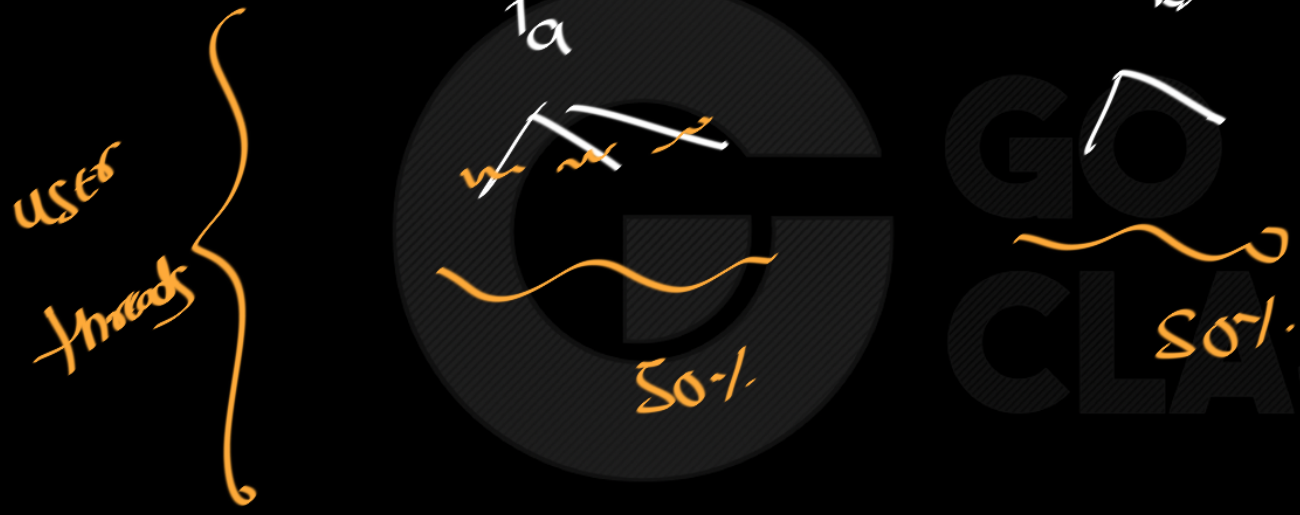
https://pages.cs.wisc.edu/~dusseau/Classes/CS537/Fall2016/Exams/f16-e2-answers.pdf

Question

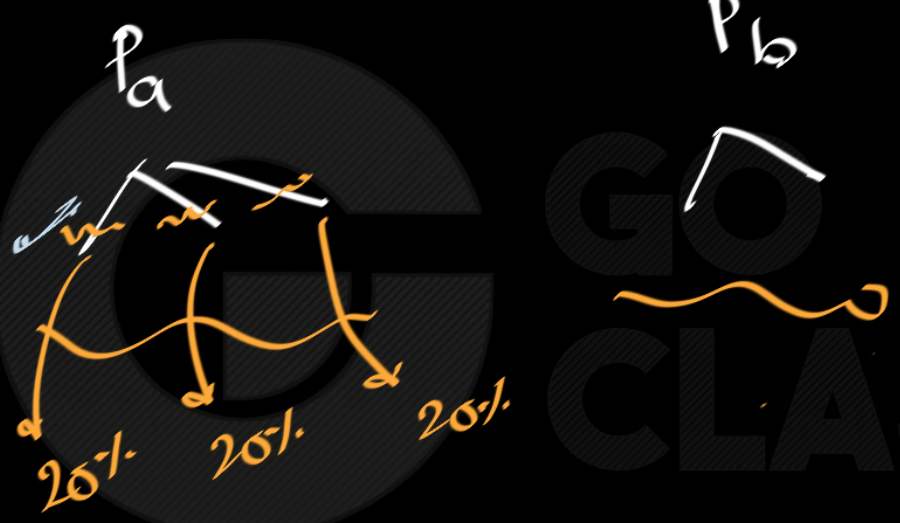
Suppose that two processes, P_a and P_b, are running in a uniprocessor system. P_a has three threads. P_b has two threads. All threads in both processes are CPU-intensive, i.e., they never block for I/O. The operating system uses simple round-robin scheduling.

a. Suppose that all of the threads are user-level threads, and that user-level threads are implemented using a single kernel thread per process. What percentage of the processor's time will be spent running P_a's threads? For full credit, briefly justify your answer.

b. Suppose instead that all of the threads are kernel threads. What percentage of the processor's time will be spent running P_a's threads? For full credit, briefly justify your answer.


User
threads
P_a
50%
P_b
50%

Kernel Threads



total how much time Pa threads are taking?

60% of CPU

Suppose that two processes, P_a and P_b, are running in a uniprocessor system. P_a has three threads. P_b has two threads. All threads in both processes are CPU-intensive, i.e., they never block for I/O. The operating system uses simple round-robin scheduling.

a. Suppose that all of the threads are user-level threads, and that user-level threads are implemented using a single kernel thread per process. What percentage of the processor's time will be spent running P_a's threads? For full credit, briefly justify your answer.

50% of the processor's time will be spent running P_a's threads.
Each process has a single kernel thread. Threads never block, so they will always consume their entire scheduling quantum. Once a thread has consumed its quantum, it will be preempted in to allow the thread from the other process to run.

b. Suppose instead that all of the threads are kernel threads. What percentage of the processor's time will be spent running P_a's threads? For full credit, briefly justify your answer.

60% of the process's time will be spent running P_a's threads.
In this case there are five kernel threads, each of which will receive 20% of the processor's time. Three of these are associated with process P_a.

ROUND-ROBIN VS. STF/FIFO

- Round-Robin preferable for
  - Interactive applications
  - User needs quick responses from system
- FIFO/STF preferable for Batch applications
  - User submits jobs, goes away, comes back to get result

Shortest-remaining-time-first (SRTF)