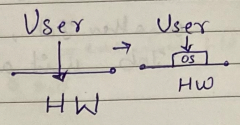
LECTURE 1 TO LECTURE 3 WRITTEN NOTES:-

Operating System

Lecture #1 : Introduction to OS.

→ OS is an interface b/w user and hardware.



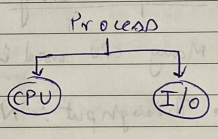
→ Resource allocator

→ Manager → memory, processes, files, security etc.

Main Goal of OS:

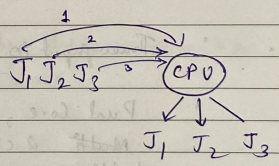
Primary → convinience (macOS)
Secondary → efficiency

Types of OS



1) Batch OS

2) Multiprogramming



CPU sits idle in Batch OS when J_1 goes to I/o
∴ there is starvation
less efficient.

In Multiprogramming, when J_1 would go to I/O, then J_2 will be processed given it only requires CPU.

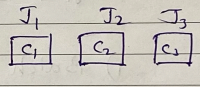
Some textbooks call this same as multiprogramming.

Multitasking : When OS chooses one Job, in then middles leaves it to complete another Job, then again does the same for either Job until all of them are completed.

From Preemption : If you could allow the processor to stop a process forcefully, and then start the other process.

→ If we allow multiprogramming with pre-emption then multitasking = Multiprogramming.

Multiprocessing :



- Many CPU used in computer/OS.
- Throughput : No. of Jobs done / unit time
- Throughput is large.
- Dual Core, Quad Core - These all are just 2 CPU, 4 CPU etc which means it is Multiprocessing

Real time OS: Job needs to be finished before deadline.

Example : Military applications, Rocket Launch

* Terminologies will be used before they are explained. ∴ keep going even if you don't understand, it will make sense later.

Camlin

Lecture #2 : Processes, PCB & Attributes.

Process Management

Example.C ⇒ Complier ⇒ a.out ↗ executable file.

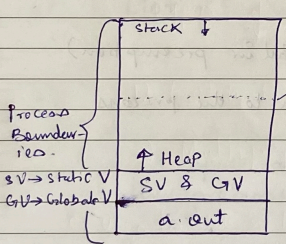
program.

↑ Interprets and converts to machine language

a.out run by OS → Main memory (RAM) → structure creation in RAM

↓ Stored in secondary memory (Harddisk)

↓ This data structure which is created is called a process.



Not exactly, but kind of boundary.

Stack → for function calls/Recursions

Heap → for dynamic allocation & work

- If our program/process tries to access memory outside process boundaries, we get segmentation fault.
- Heap grows upwards from bottom, stack grows downwards from top. By this, the space is automatically managed / allocated to stack/Heap if it requires.

Camlin

more memory, and the other space would as as a consequence be shrinked.

→ This whole data structure which was created by Operating system by the program/executable file stored in Harddisk is called a process.

" Soul is to Body as Program is to Process "

Attributes of a process:

1) Process ID: Unique number given to process in a computer.

2) Program counter: Contains next instruction that has to be executed.

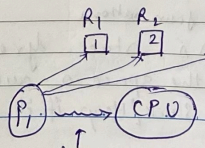
(will be helpful in preemption)

3) Priority: A number given to the process when it is created.

4) General purpose Registers:

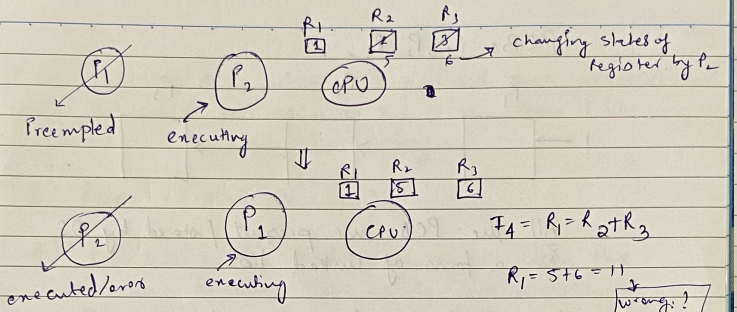
A Process doesn't only refer to the elements that are in the process, it can also refer to general purpose registers.

A process can store some numbers in registers



$I_4 : R_1 = R_2 + R_3$

Camlin



∴ What to stored in the General purpose registers should also be stored somewhere.

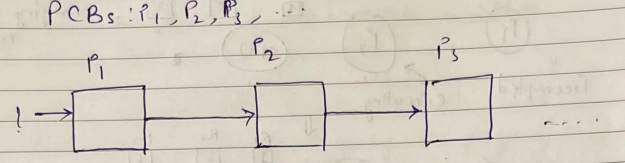
5) List of open files

6) List of open devices (Hardware) for protection (like printer, scanner)

7) Protection: Making sure a process does not have access to other process's work space and a process is not interfered by other processes.

8) Process state: Keeps what's the status of the process
Ex.: i) Ready to run ii) Running iii) Blocked iv) Waiting or more

→ All this information about a process is stored in Process Control Block (PCB).
For every process, we get 1 PCB.



All the PCB are present / stored by OS in a form of linked list

Info. about one Process can be known through 1 PCB but to the info. of all process, we should use the linked list.

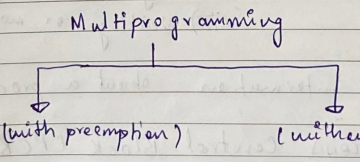
Lecture # 3: Process states and multiprogramming

→ PCB (Process control Block) is also called ~~Content~~ context. Everything about a process is called content.

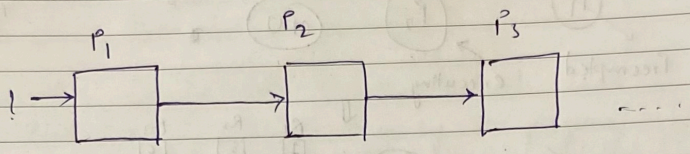
States of a process

1) New → When process is in secondary memory

2) Ready: Process in the main memory.



PCBs : $P_1, P_2, P_3, \ldots$



All the PCB are present / stored by OS in a form of linked list

Info. about one Process can be known through 1 PCB but to the info. of all process, we should use the linked list.

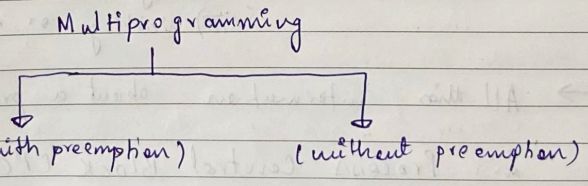
Lecture # 3 : Process states and multiprogramming.

→ PCB (Process control Block) is also called ~~content~~ context. Everything about a process is called content.

States of a process

1) New → When process is in secondary memory

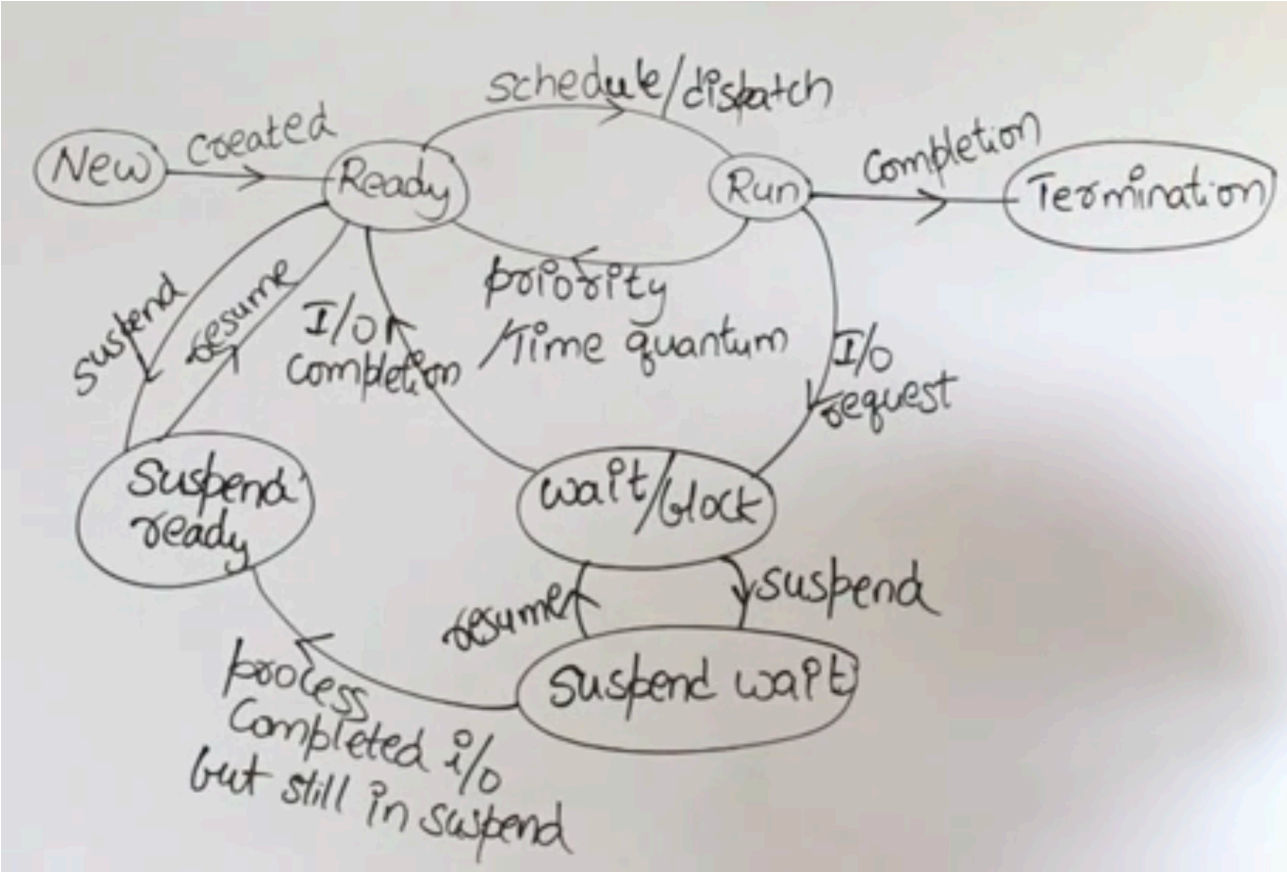
2) Ready : Process in the main memory.



Lecture #4

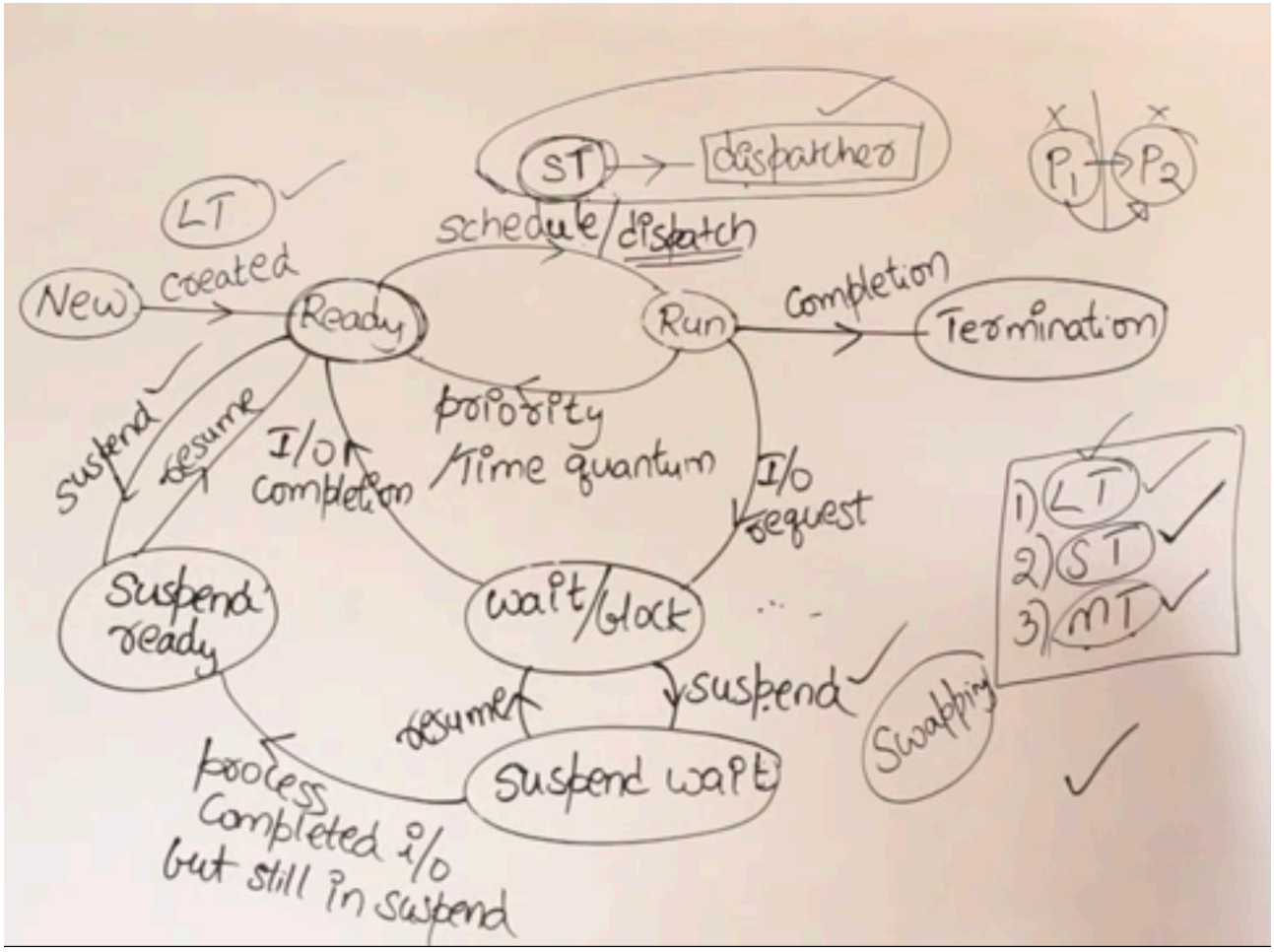
PROCESS STATE TRANSITION DIAGRAM AND VARIOUS SCHEDULERS

State diagram



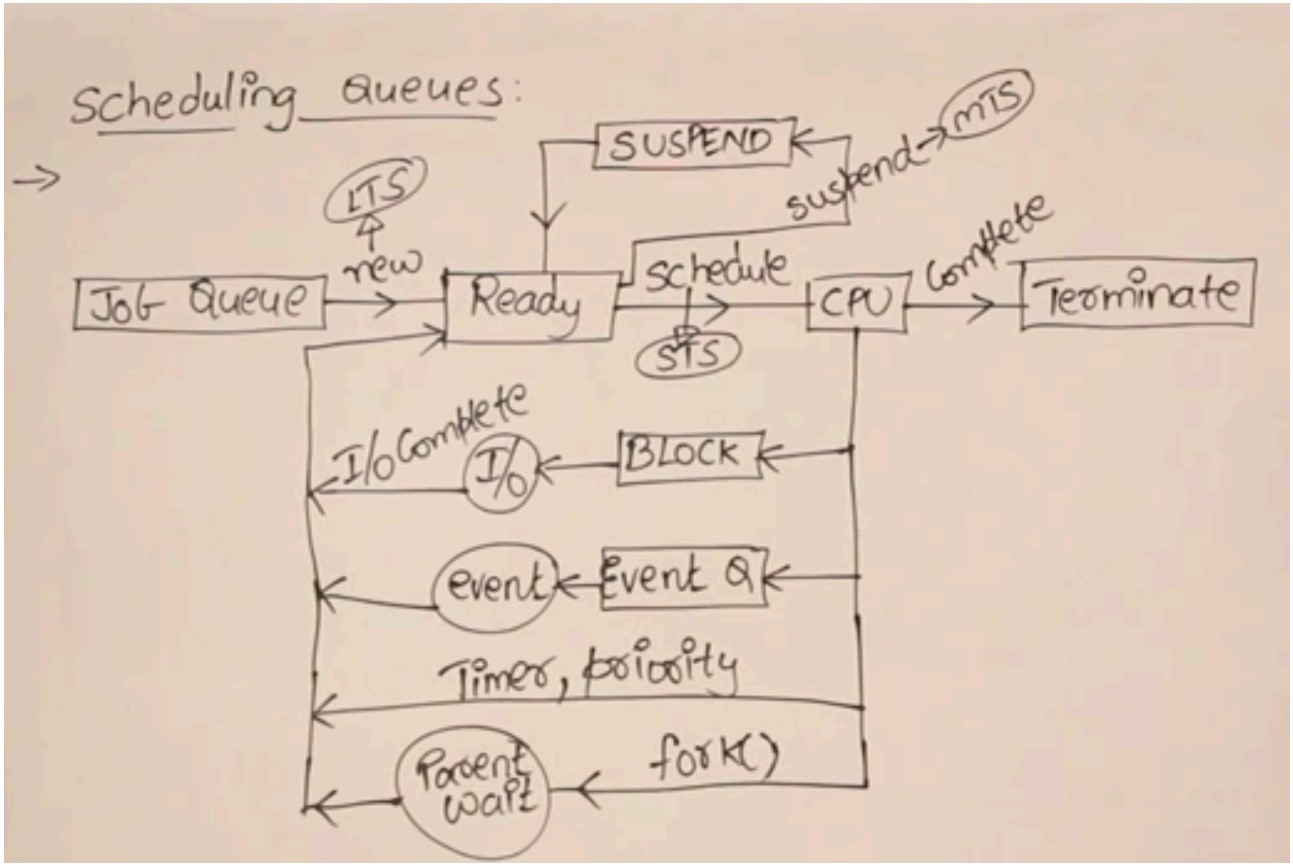
- How many processes are in ready state will depend on the resources that we have
- Smallest number of states a process can go through is 4
- This is New —> Ready —> Run —> Termination
- A process contains two times: CPU time and I/O time
- If a process has more CPU time, then it is called CPU bound process, and consequently, it is called I/O bound process if it has a lot of I/O to perform (input/output)
- Once a process asks for I/O, it is sent to wait/block state. We do not send it to ready state as we do not want process in ready state to ask for the I/O again. Simply put, we want our ready state process to be I/O free, so that we can efficiently schedule it.
- Once it's I/O is over, it is sent back to ready state to be executed by the CPU.
- Suppose if a very high priority process appeared, then we would need to stop the current process and execute that first.
- Therefore, a process may go from run to ready again due to appearance of a high priority process
- In other words, **priority will cause a process to get preempted**.
- Due to preemption, this entire thing can be said as multitasking or multiprogramming with preemption. Multiprogramming because there are more than one processes in ready state.
- Another thing which decides preemption is the **Time quantum.** If we want a process to be executed for a particular amount of time and then keep changing, then this will also cause preemption.
- These 5 state are basic states which every operating system will have:New, Ready, Run, Termination and wait/block.
- Depending our the requirements, when our resources are too scarce, will have to suspend some process for some time
- If we push out a process from ready state, then it will go to suspend ready state, and similarly, wait/block state processes will go to suspend wait state.

- Both suspend wait and suspend ready state processes are in secondary memory (i.e. hard rive)
- Since I/O can happen from second memory also, when the process is in suspend wait, it will keep performing its I/O, and when it's finished, it will go back to suspend wait, and then to ready, to be ready for execution by the CPU.
- We notice that we make a choice here, and the choice being the number of processes to be created, because once you create a processes, you have to live with it, and all the processes then have to be in either of the states of the diagram.
- So how many processes are to be created is a long term decision and there it taken by a scheduler
- There three types of scheduler - Long term scheduler, Short term scheduler, medium term scheduler
- Long term scheduler makes the decision of how many processes are to be made.
- Picking one of the processes from the ready state is a short term decision i.e. it won't last forever or for a long time ∴ short term scheduler makes this decision
- This is the reason Short term scheduler is also called as dispatcher
- When a process goes into either suspend wait or suspend ready, it is known that it will not last for a very long time in that state, and it is also known it may not be become ready shortly after it was suspended. ∴ medium term scheduler makes the decision of suspension.
- Degree of programming - Maximum no. of processes that can be in the ready state at once.
- LT scheduler will have significant impact on the performance of the CPU
- If LT scheduler picks only those processes which are I/O bound, then the performance of the CPU will be less as most of them will be in wait/block state
- If LT scheduler picks only those processes which are CPU bound, then it will havre more performance, but there will occur a starvation of I/O bound processes
- ∴ **there should be an ideal mix of both I/O bound and CPU bound processes given by the LT scheduler**
- There is a difference b/w ST scheduler and dispatcher - STS picks a process that needs to go from ready to run, and then calls dispatcher which actually moves the process from ready to run (or run to ready).



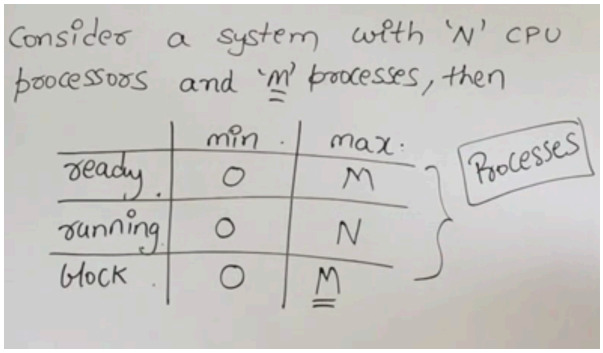
- This is called **context switching.**
- This context switching takes time - it depends om the size of the context.
  - This is also called context switching time
- So ST Scheduler should choose processes such that there are minimal context switches.
- MT scheduler is not that important - in the sense that it will move processes from suspend to ready or run or vice-versa depending on the resources available.
  - It is responsible for swapping - putting processes from main memory to secondary memory or vice-versa
- MT scheduler should also be optimised, i.e. swapping should be as less as possible, as it would take a lot of time.

Lecture #5: Process queues



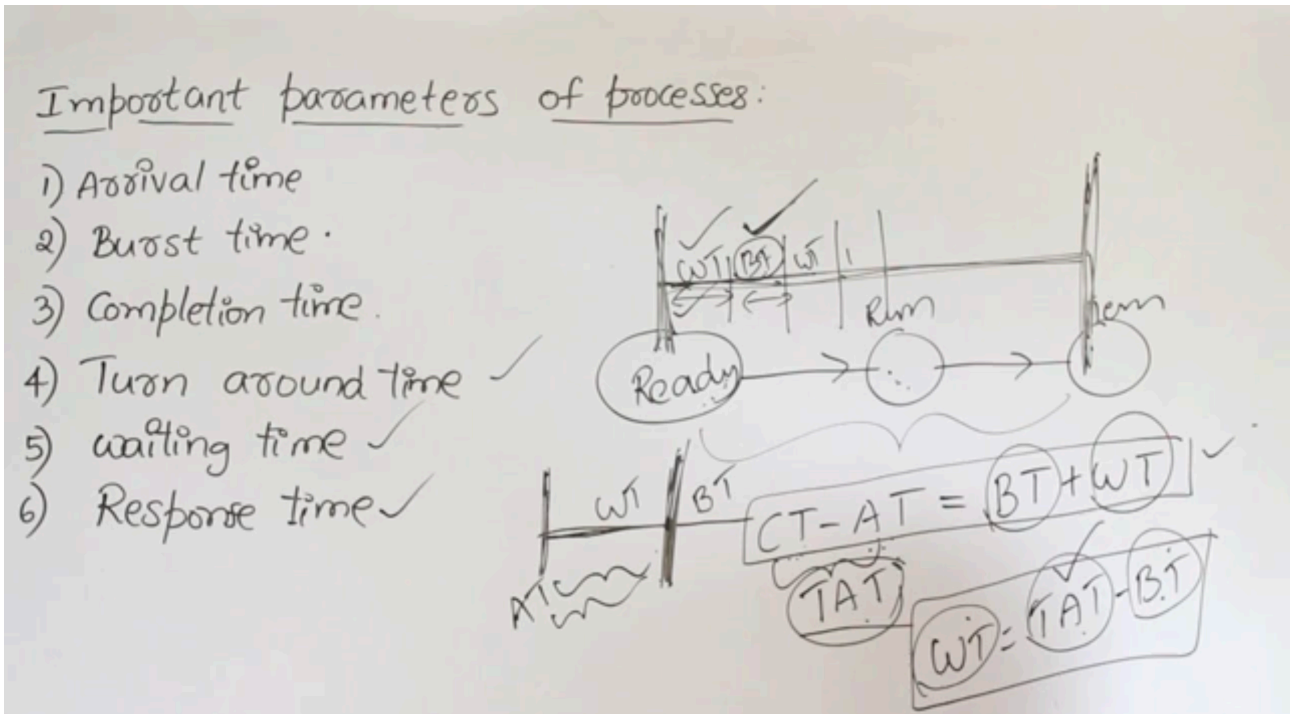
In the diagram above, the blocks are different kinds of queues implemented as linked list.

Lecture #6: Question on process states

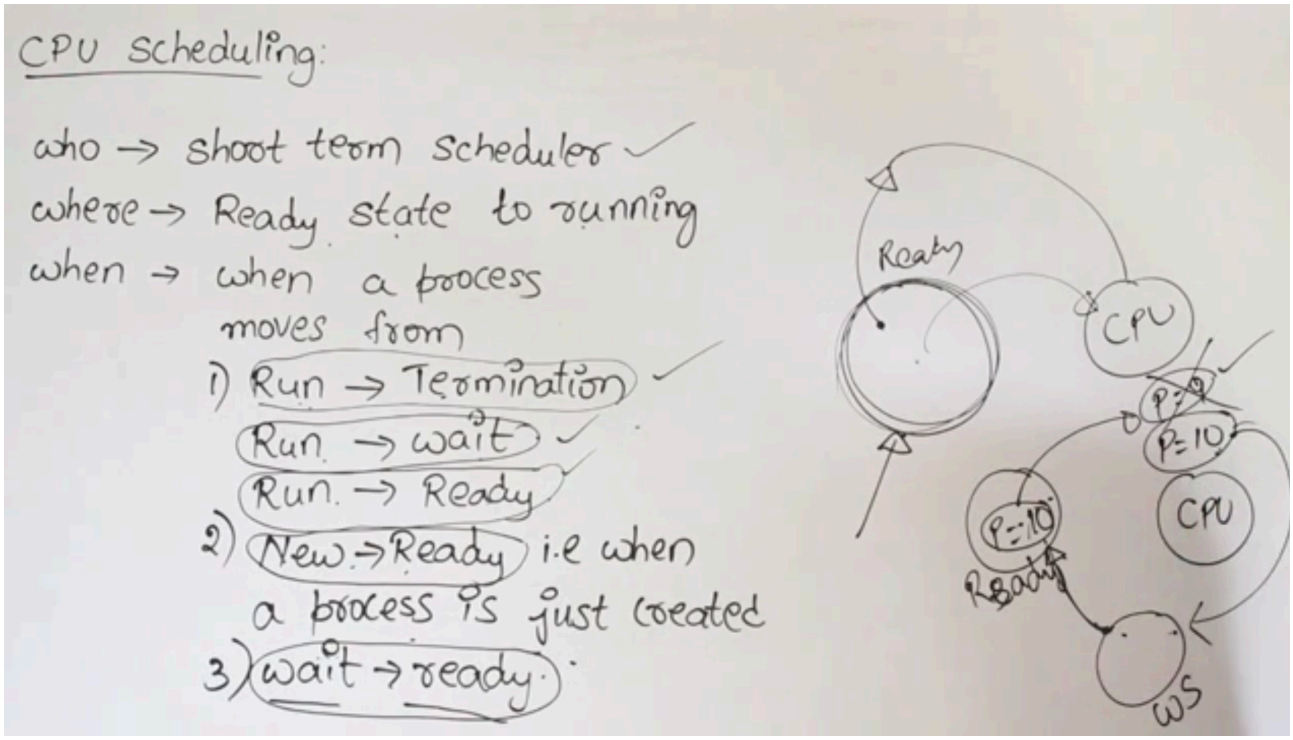


Lecture #7: Various times related to process

- Arrival time - the time at which the process arrives in the ready state
- Burst time - the amount of time that is required by a process to finish the CPU time
- Completion time - the time at which the process finishes
- Waiting time - the time for which the process sits idle in the ready stare before going to the run state
- ***Completion time (CT) - Arrival time (AT) = Burst time (BT) + Waiting time (WT)***
- CT - AT is also called turnaround time (TAT)
- ∴ ***WT = TAT - BT***
- Response Time (RT) = first time the process makes contact with the CPU - AT



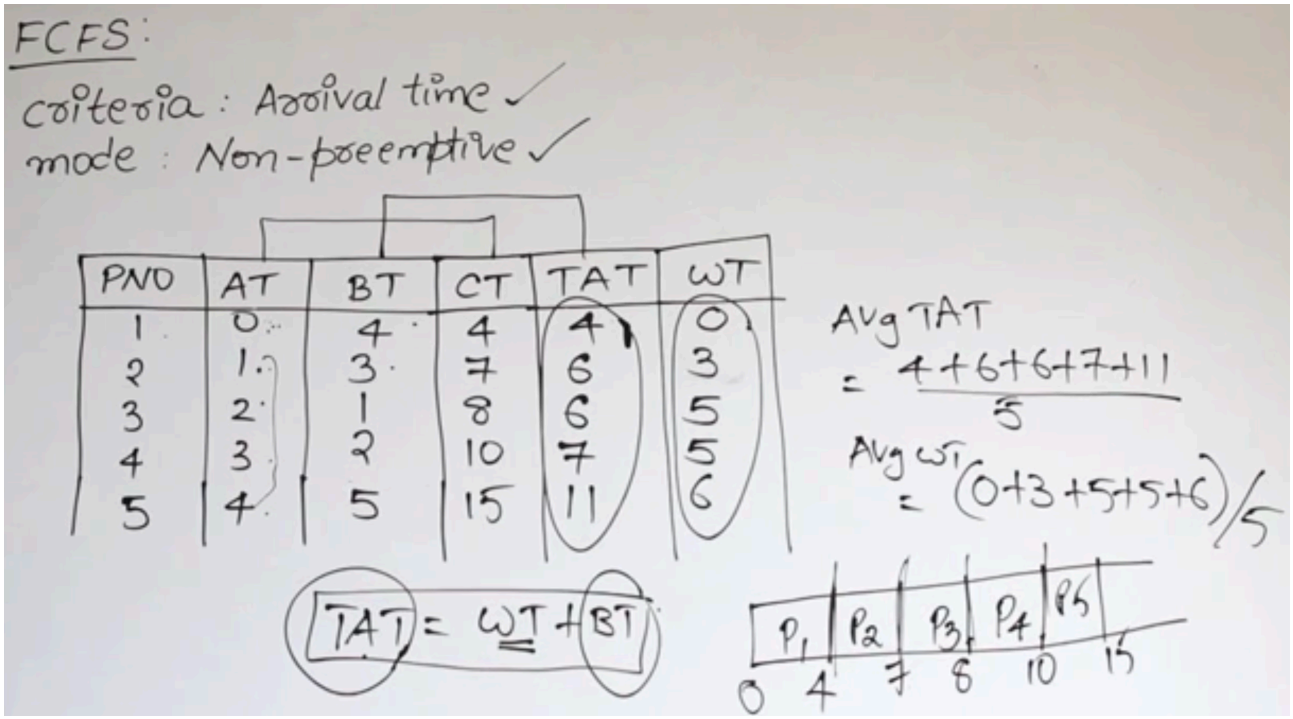
Lecture #8: CPU Scheduling

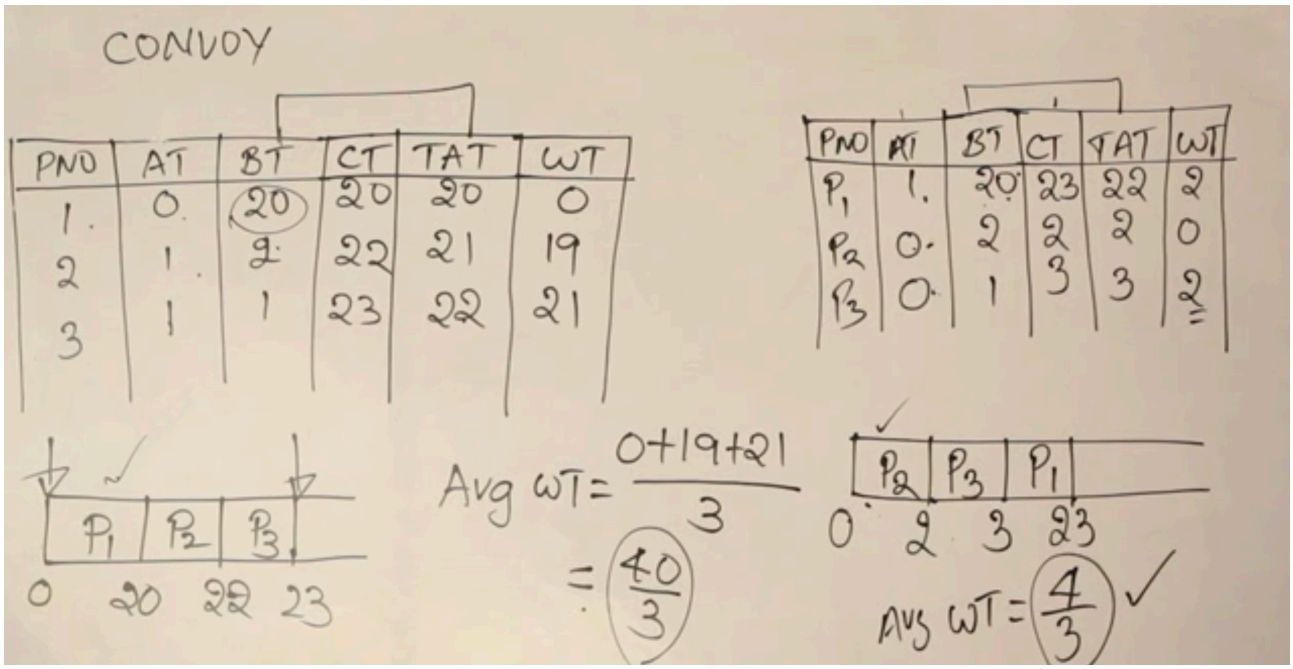


- When run to {termination, wait, ready} happens, we should definitely schedule a process
  - This because when a process goes from run to any other state, the CPU is idle and needs to be schedule so that it does not remain idle
- When {new, wait} to ready happens, scheduling need not be required as the priority of the process that is freshly queued into the ready queue may not be as much as the process which is already in the run state/ in the CPU. Though if the freshly queued process has the priority greater than the running process, preemption/scheduling should happen.

Lecture #9: Introduction to FCFS

- FCFS means First Come First Serve
- In this we assume that every process requires only CPU time and no I/O time
- ∴ no need to use block/wait state
- ∴ state diagram/path is simple: New —> Ready —>Run —>Terminate
- Arrival time is relative time
- ***When arrival time is same, the process with smaller Process ID should be given priority***
- Gantt chart is the chart prepared from diagrammatically representing the sequence of jobs with the time of each job when it is scheduled
- Arrival time is the priority here as it would be obvious by its name
- **There is no preemption in this FCFS**



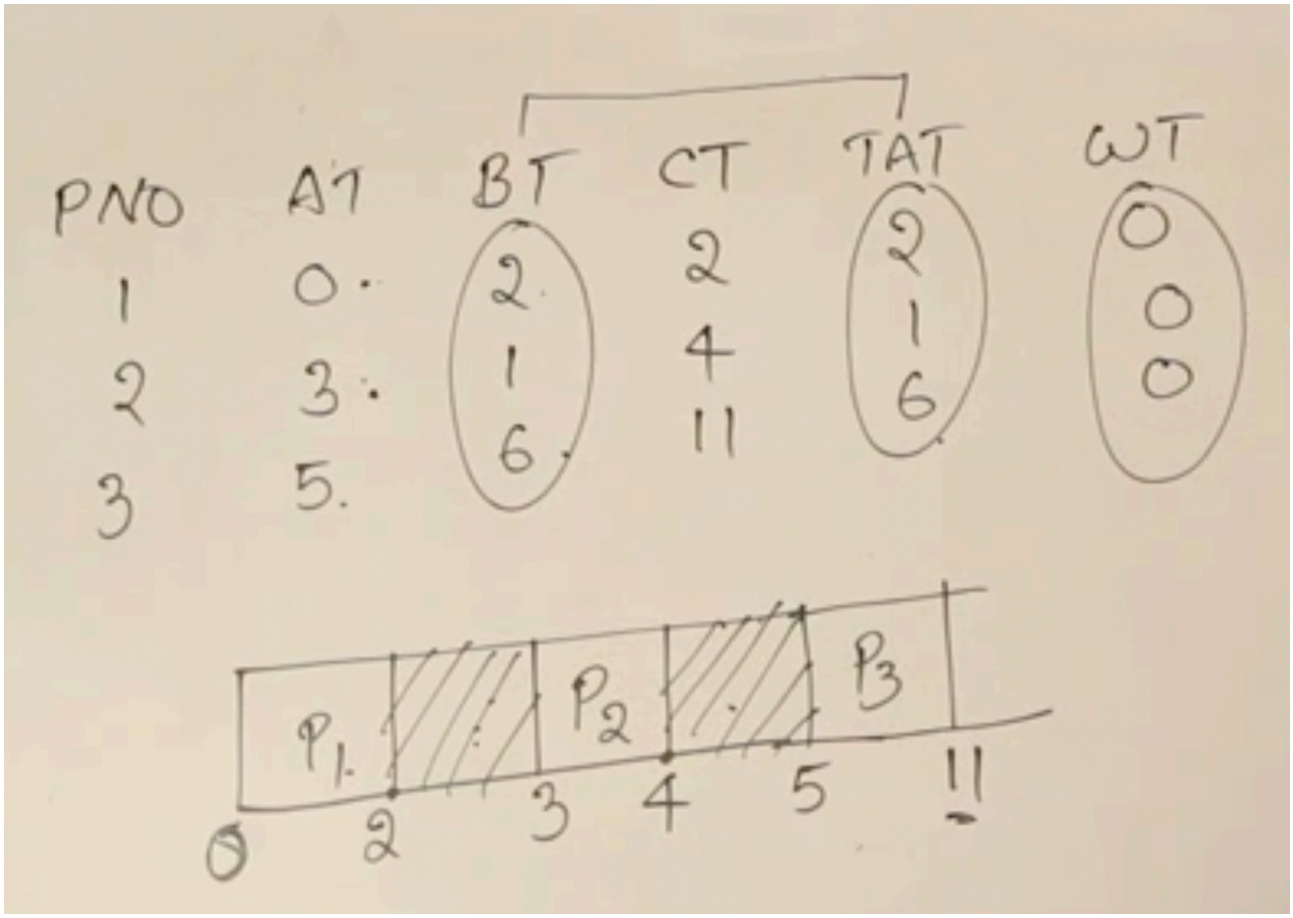


Lecture #10: Convoy Effect

- Convoy effect or starvation is the main disadvantage of FCFS
- It mainly happens when a process with large burst time arrives first
- To tackle this, we devise another scheduling algorithm - which will be called Shortest Job First (SJF)

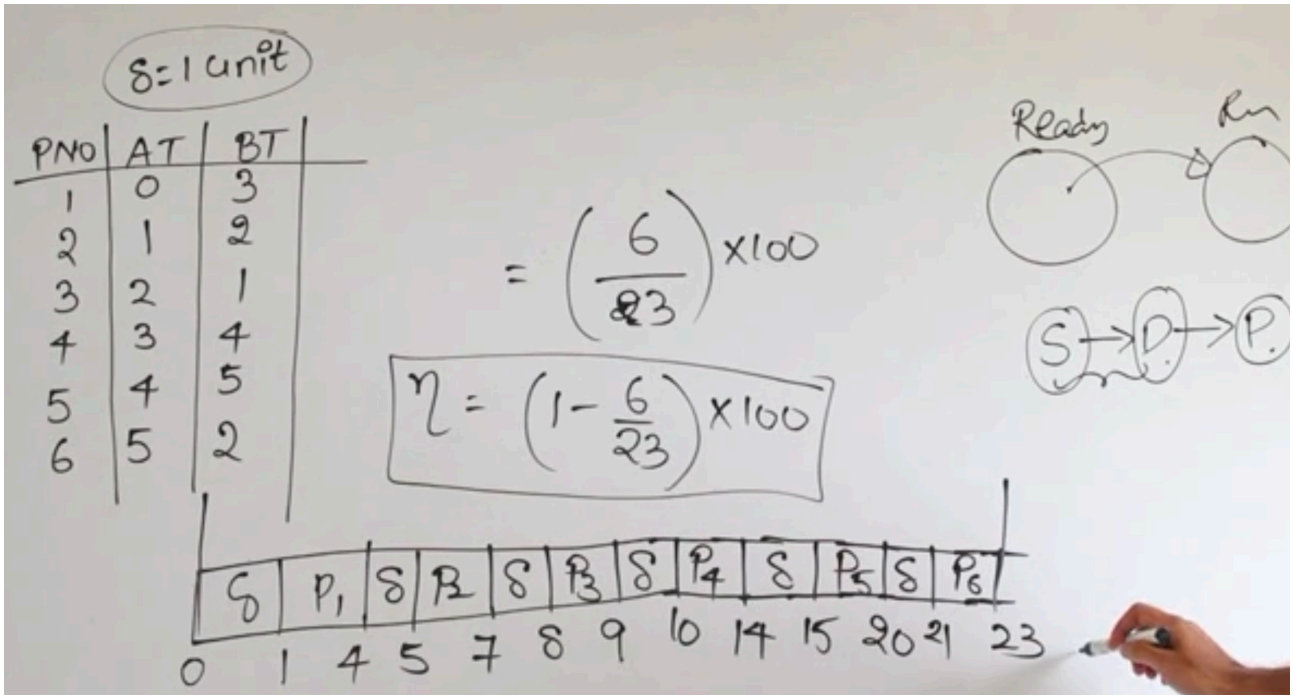
Lecture #11: FCFS example

- when drawing the gantt chart of FCFS, there might be gaps in the chart which signifies there are no processes in ready state when the CPU is idle
- ***The simplest data structure required to implement FCFS is queue***
- Time complexity of FCFS is $O(N)$



Lecture #12: FCFS with overhead

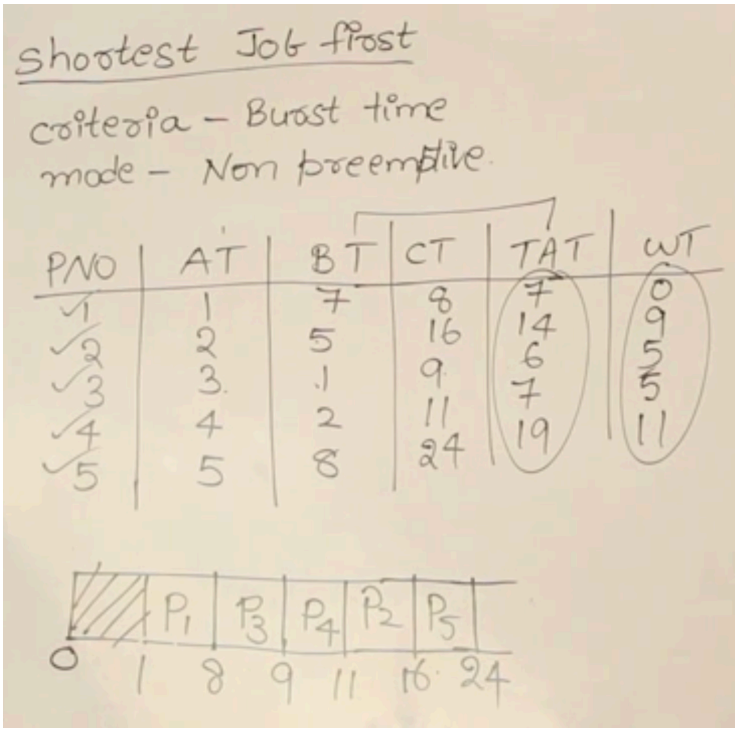
- Once process is over, scheduler will call the dispatcher, which will make the process available for running
- This process of invocation and scheduling is called the delay or overhead time δ
- Efficiency is defined as $\eta_e = (1 - \frac{\delta_{total}}{t}) \times 100$ where t is the total time to execute all the processes



Lecture #13: Shortest Job First

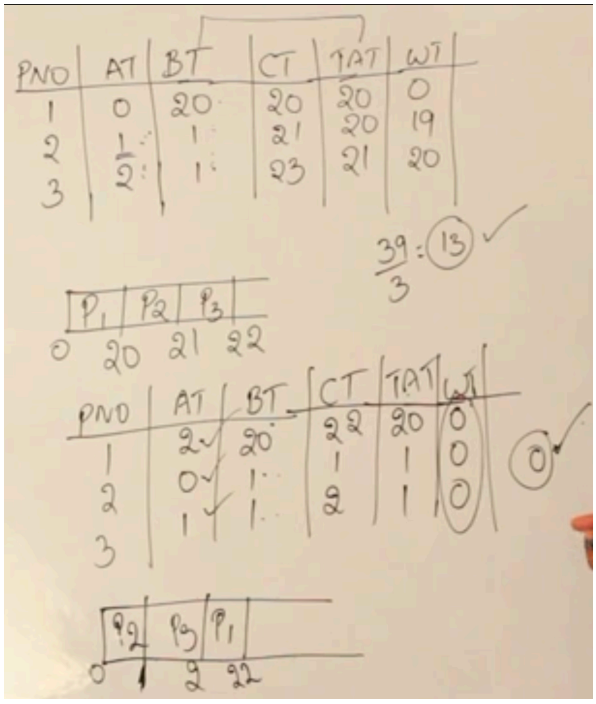
Question: Given any schedule or random combination of processes, with random possible arrival times, our SJF algorithm will always schedule the shortest job first. (T/F)
Answer: **False**
It will always schedule the shortest ***available*** job first.

- SJF does not guarantee that whatever be the AT, it will always schedule the shortest job first (or the job with the least BT)
- ∴ sometimes, even SJF will fall victim to the convoy effect
- The ideal data structure required for implementing SJF is **min-heap.**
- The time complexity for SJF is $O(N \log N)$ as each step requires exactly one deletion and one insertion both of which take $O(\log N)$ time (*EXTRACT-MIN* and *INSERT*)



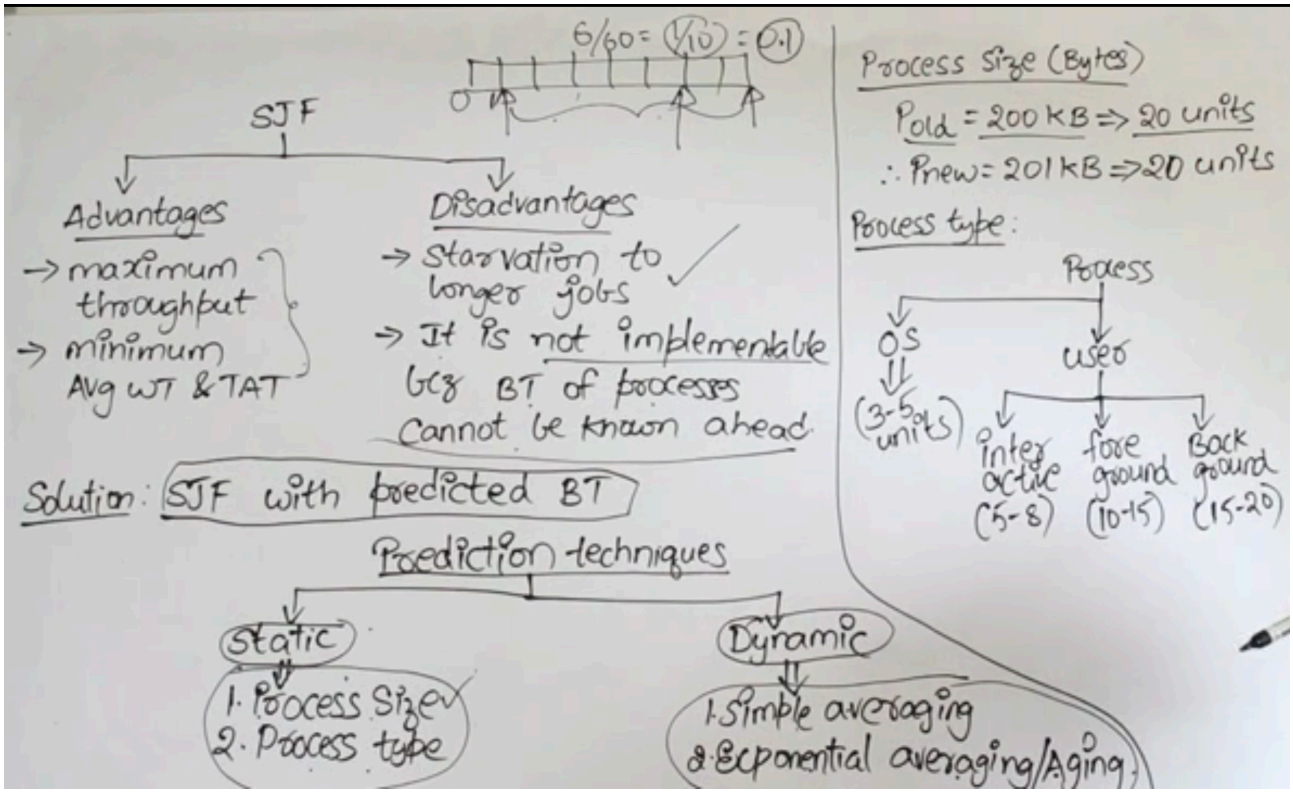
Lecture #14: Analysis of SJF

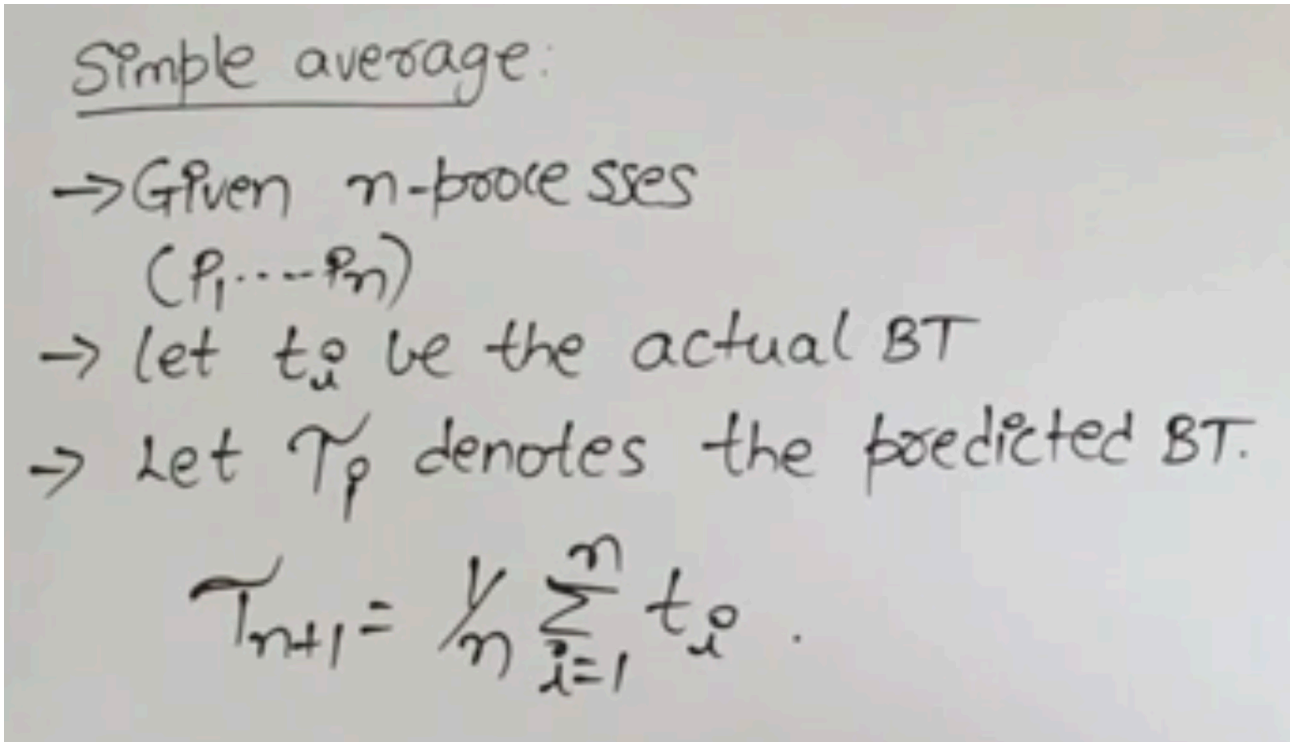
- though it is not mentioned in any textbook that SJF exhibits convoy effect, an example can easily be constructed to show the same

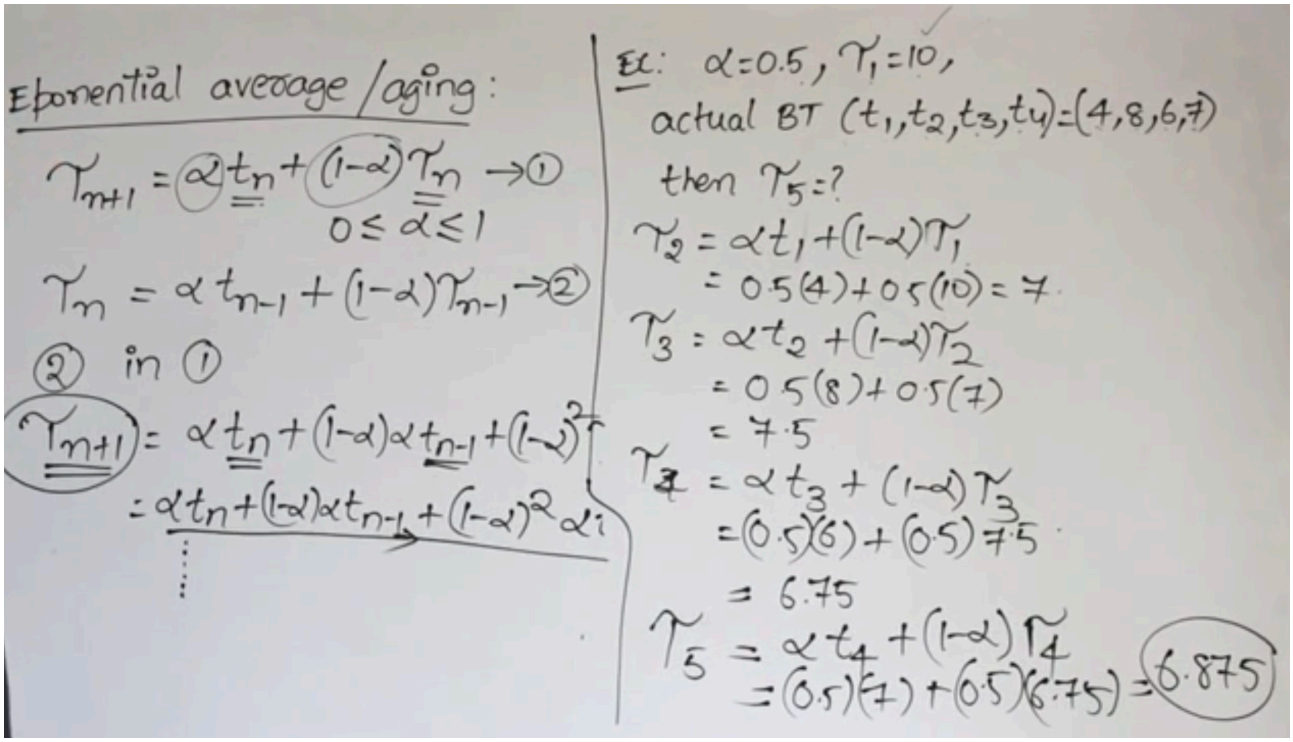


- When we have two processes with same BT, then the one with less AT will be picked up for the job
- SJF is not practical, in the sense that it is not very easy to know the BT of processes before they even arrive.
- FCFS in this sense is relatively more practical and easy to implement.
- ∴ we don't really implement SJF practically, and therefore is used for performance measuring
- We'll prove this later, but SJF usually gives the best throughput possible: number of executed processes per unit time

Lecture #16: SJF with prediction of BT



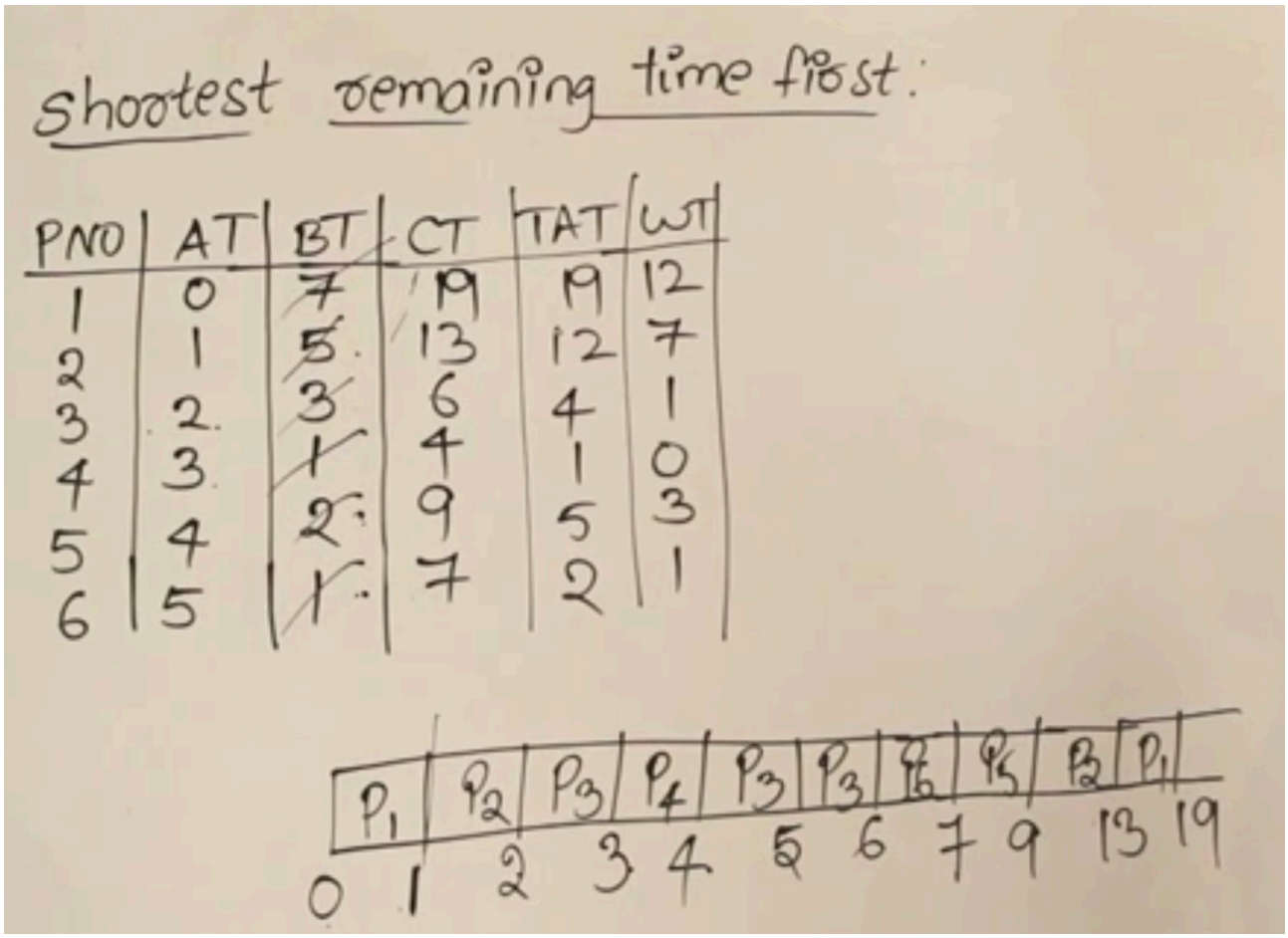




- α is called the smoothening factor
- If we want to give the previous processes' actual BT more importance then $\alpha > 0.5$ and if we think that our previous predictions will determine the next BT more accurately, then let $\alpha < 0.5$
- τ_i is the predicted BT of ith process

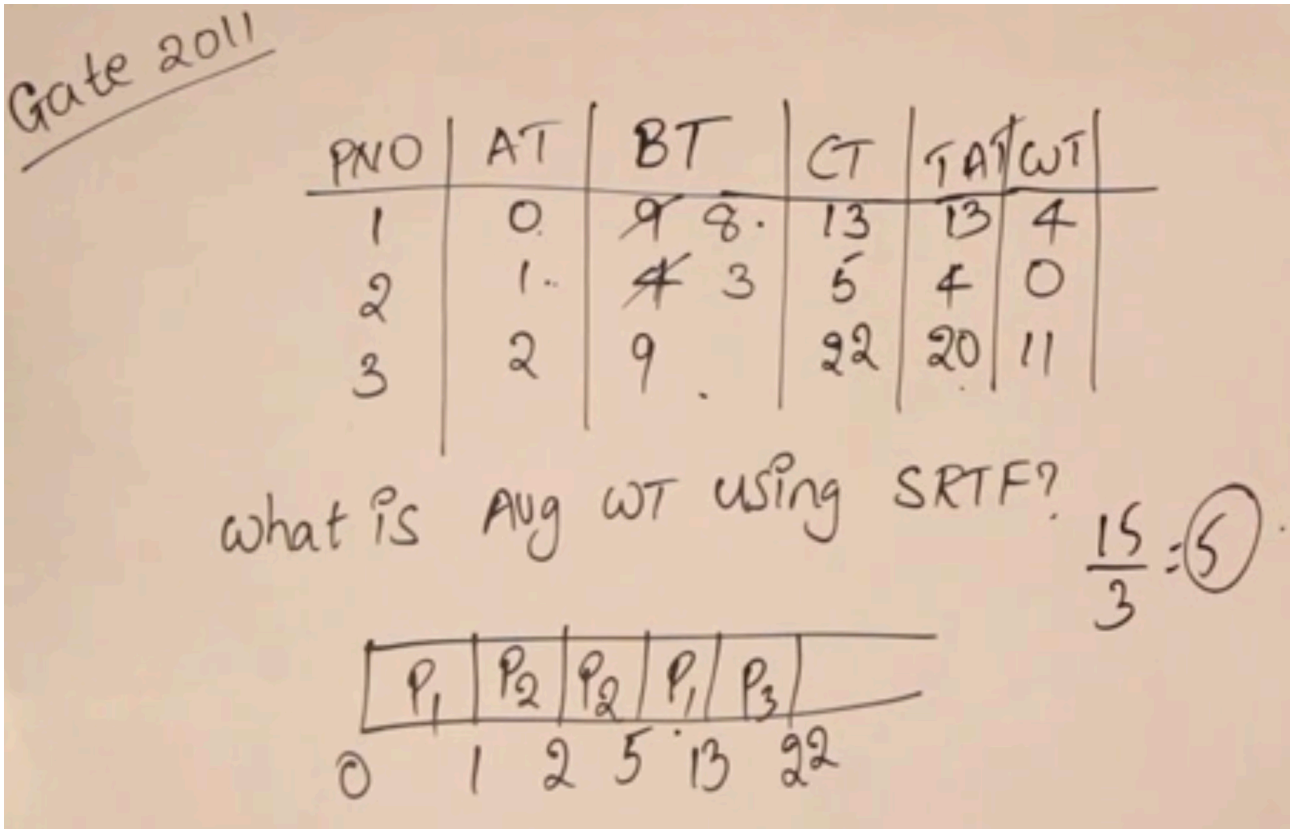
Lecture #17: Introduction to SRTF

- SRTF is just like SJF, but the mode here is preemptive
- SRTF means Shortest Remaining Time First
- As the name suggests, we'll constantly see what is the process with the shortest remaining time after each unit of BT. If the shortest remaining BT process happens to be a process which is not in run state, then we preempt the currently running process from run to ready and then switch to process with shortest remaining time
- Once all the processes become available, then this algorithm becomes a typical SJF algorithm i.e. we could implement it as a non-preemptive version. Now we need not check after each unit of BT which process has the shortest remaining time because all the processes have already arrived
- In SRTF there is no convoy effect or starvation
- Even this cannot be applied practically as we would never know the BT of each process beforehand
- Response time: first time the process makes a contact with the CPU - AT

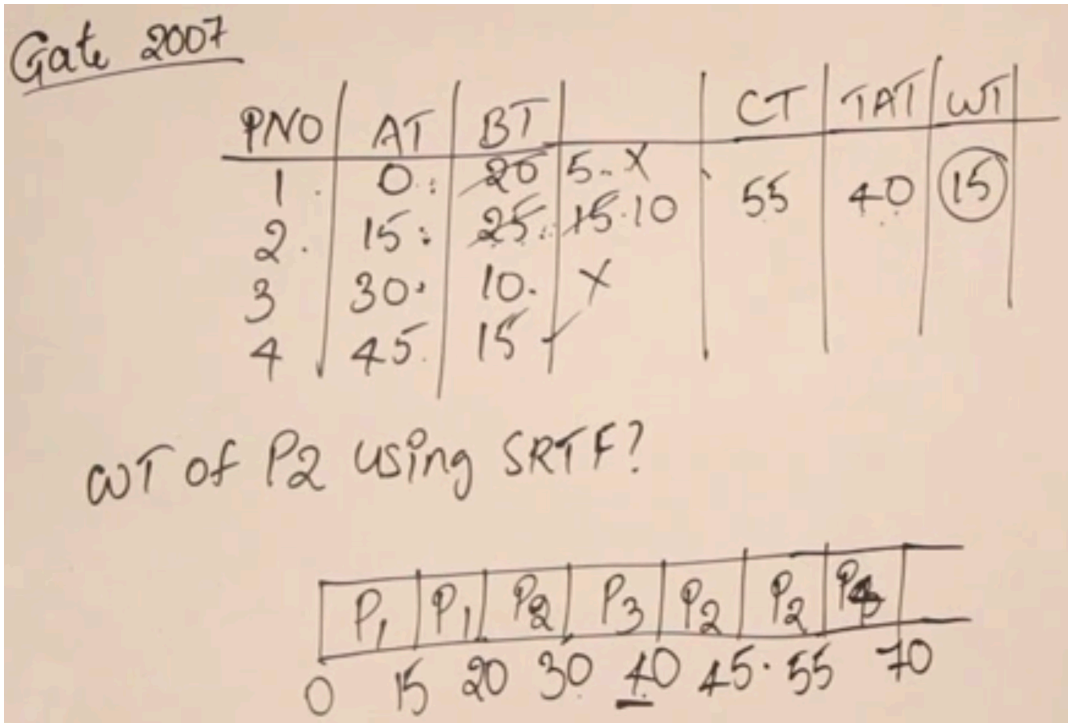


- here the response time for p_1, p_2, p_3, p_4, p_5, p_6 is $0, 0, 0, 0, 3, 1$ respectively

Lecture #18: Example on SRTF Gate 2011



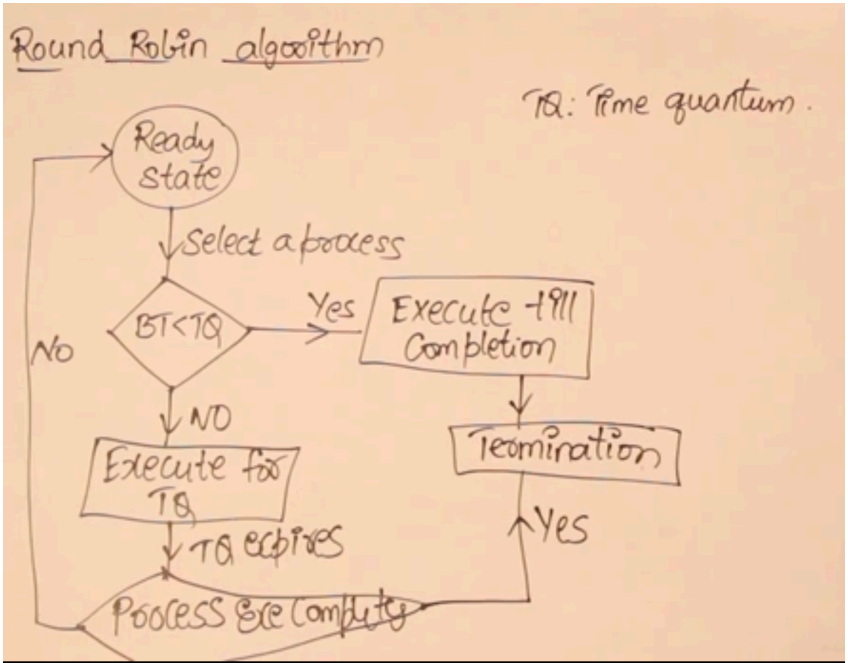
Lecture #19: Example on SRTF Gate 2007



- whenever we have long gaps between ATs then it makes sense to not stop the processes for every unit of BT
- Stopping and checking only when the next process or a new process has arrived is a wise idea

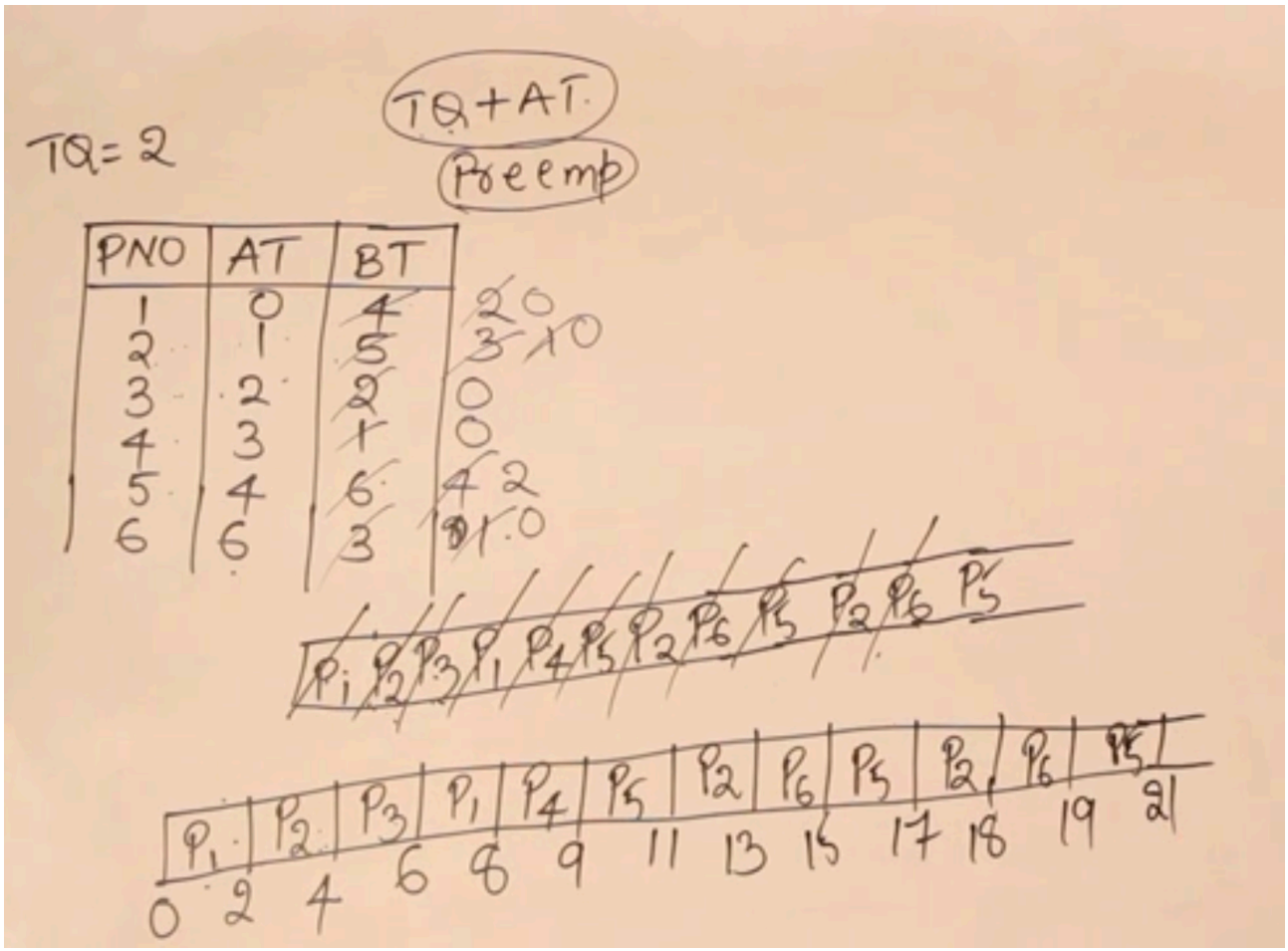
Lecture #20: Round Robin algorithm

- This is the most popular algorithm of all
- Most of the operating systems are using it today
- This is because it is practically implementable i.e. it does not depend of BT
- It does not require complex data structures like heap, using a queue we can implement it
- There is no starvation in RR
- It works by taking a particular process for execution and then execute it for a particular quantum of time, and then again preempts it and to choose another process to do the same
- ∴ no process will have to wait forever for execution or CPU i.e. they will keep on getting the chance after some regular amount of time

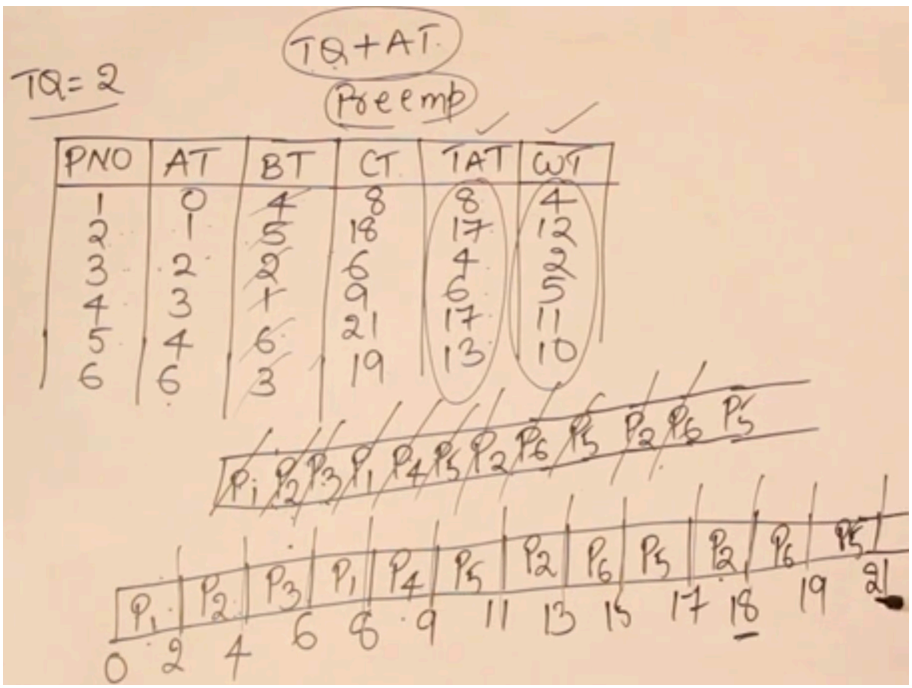


Lecture #21: Round Robin example 1

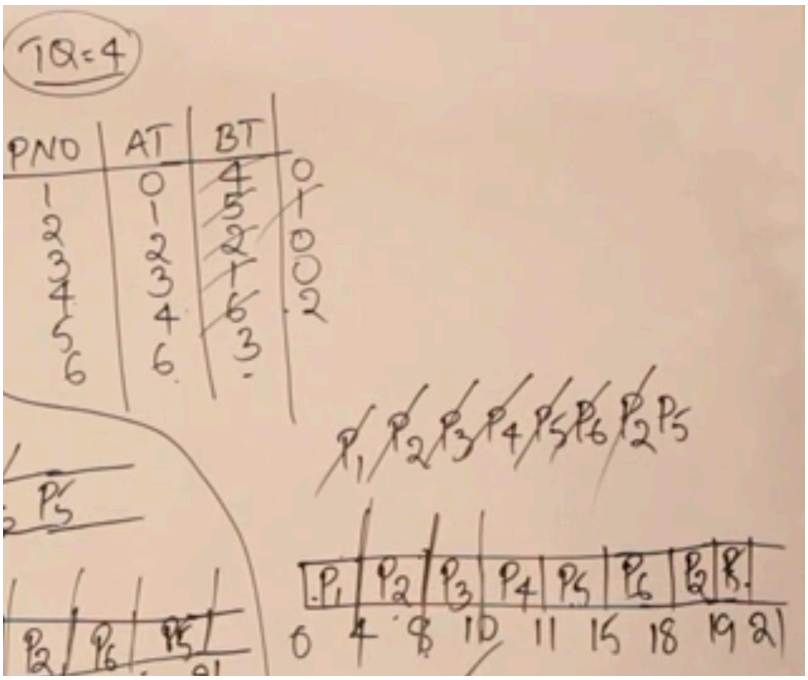
- Questions on Round robin are a bit difficult, in the sense that you have to be a bit attentive while solving questions on it.
- RR can be thought to be similar to FCFS in the sense that it schedules the processes according to AT
- RR —> FCFS+Time Quantum+Preemption



- to see the completion time of a process in the Gantt chart of RR, we will check the first occurrence of a process from the the end (This is equivalent to checking the last occurrence of a process from the start, but more easy)

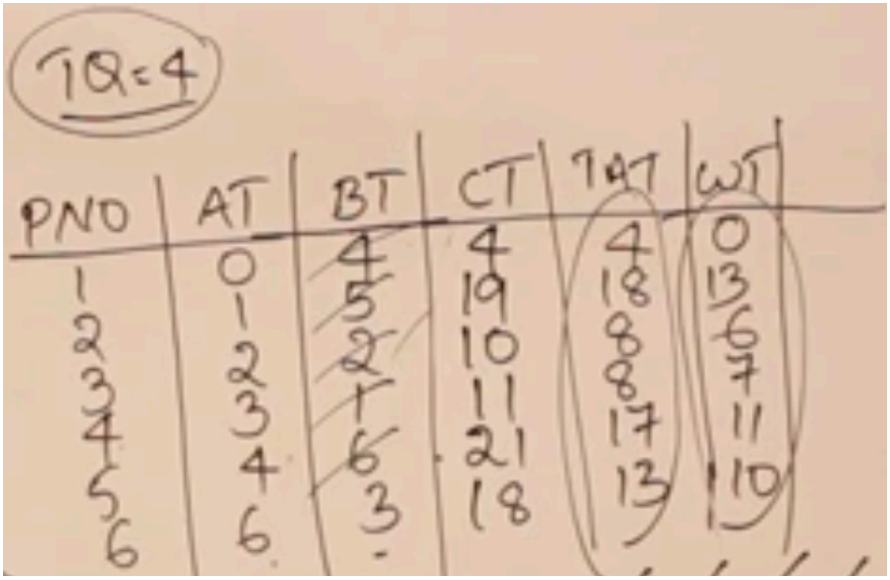


- The preemption of a process and the execution of a new process is called context switching
- For each time quantum (TQ), a context switching will happen
- The shorter the TQ, the more the context switching will happen
- And context switching takes time (context switching time), so TQ should be large
- But if TQ is large, then starvation (of some processes) might be large as they might then have to wait for a long time to get executed
  - Example: if we have 100 processes and TQ = 2 units, then some processes might have to wait for 2 • 100 units (the last one)
- We solve the same problem as before but with TQ = 4 this time



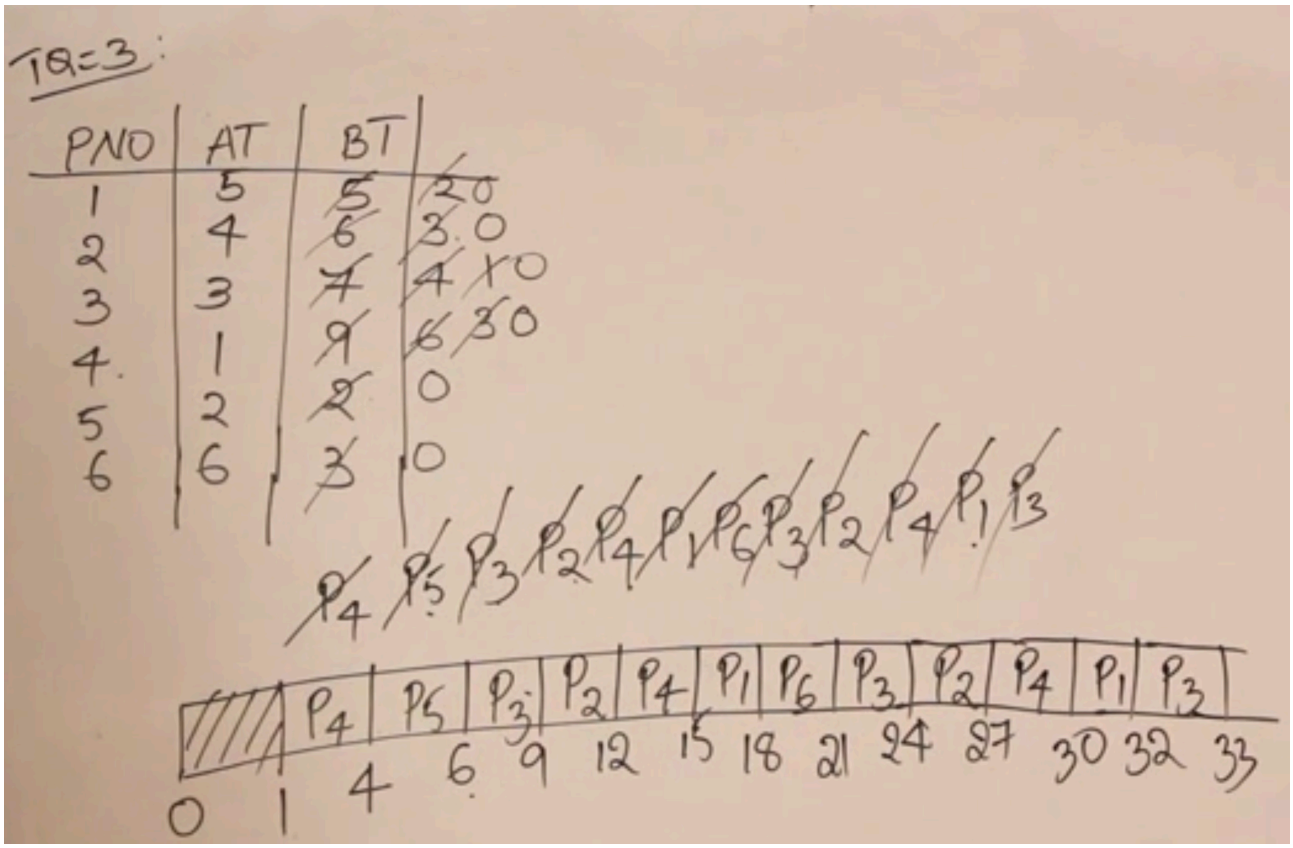
- From this we can clearly see that the number of context switches have decreased from before
- But we also see that processes have starved. As an example:
  - p_2 was scheduled at 13 units for the first time when TQ was 2, but it is now scheduled at 15 for the first time when TQ is 4 units

- p_4 was scheduled at 9 units for the first time when TQ was 2, but it is now scheduled at 11 for the first time when TQ is 4 units
- ∴ the TQ should not be too small or too large



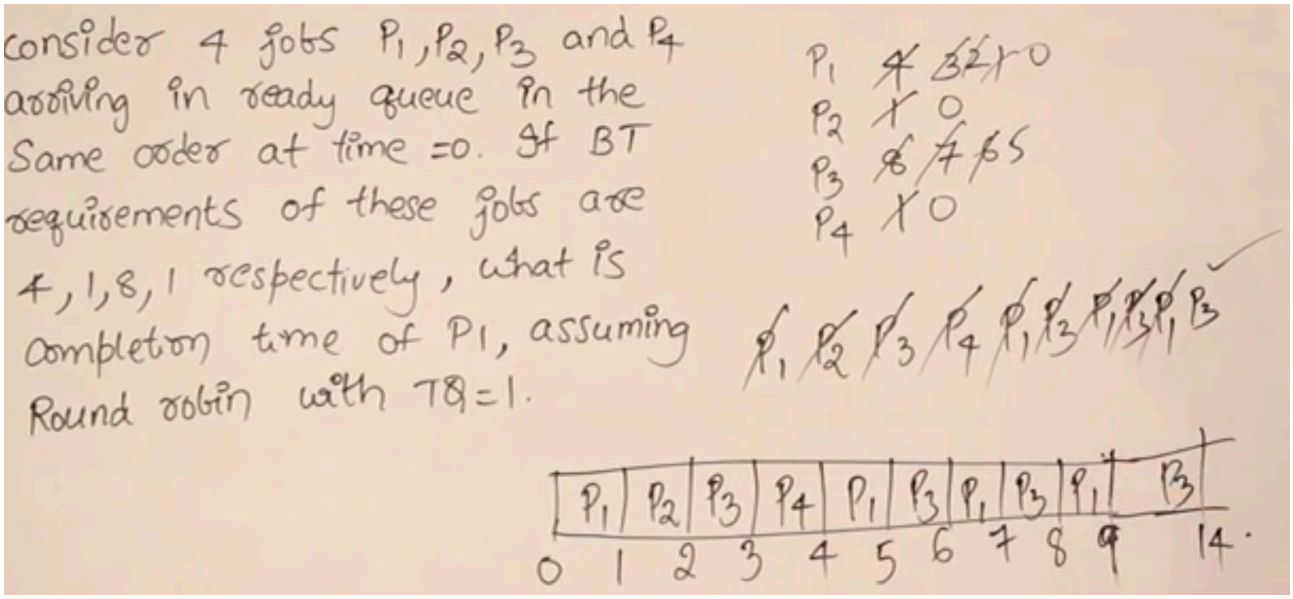
Lecture #22: Round Robin example 2

- Always remember that, after executing a process for a TQ, before doing anything else, you should first add the processes which have arrived in the meantime in the queue and then you should add the preempted process at the end of the queue (if its still not complete, that is)



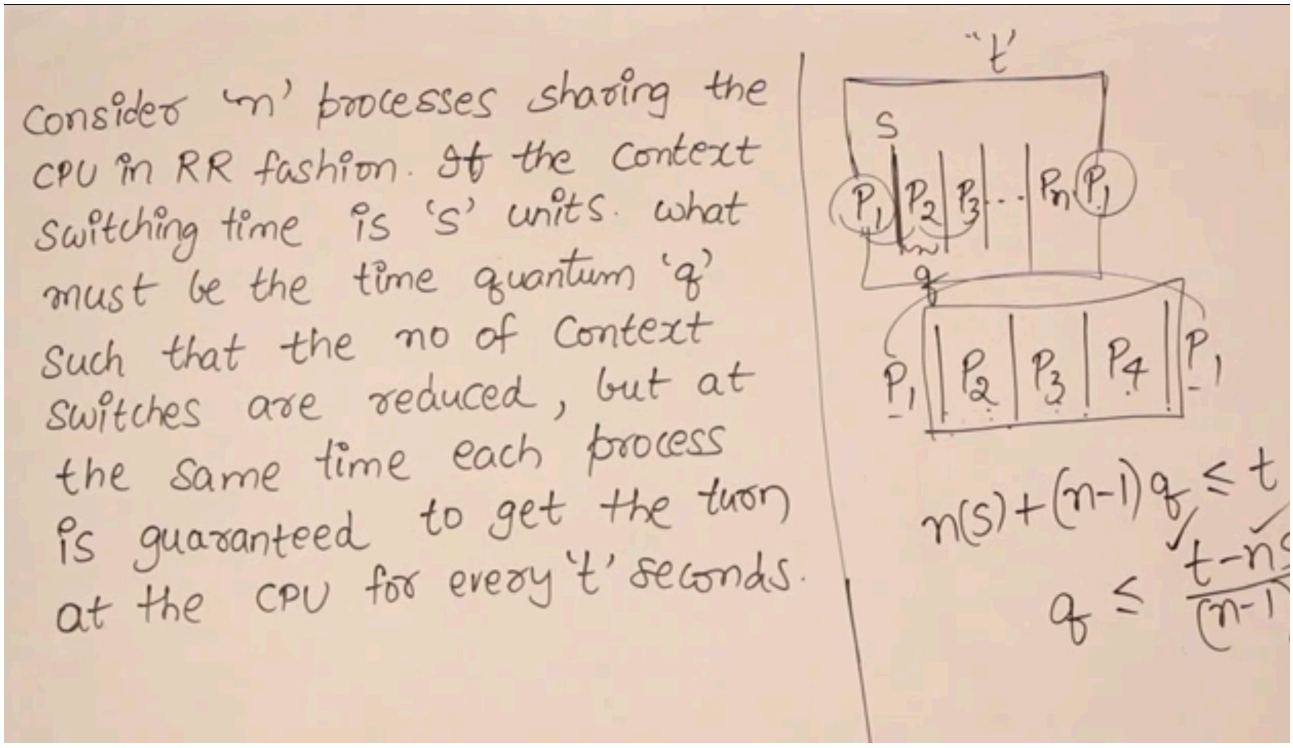
- Now, to calculate the response time (RT), we need to check the first occurrence of a given process from the start to find out when it was first scheduled, and then we can subtract it with AT to find out RT
- When TQ increases, the RT decreases and vice-versa
- $\therefore TQ \propto \frac{CST}{RT}$ where CST is context switching time
- $\therefore$ if we want our processes to be very interactive, then we should go with high TQ or consequently low RT
- When $TQ \to \infty, RR \to FCFS$

Lecture #23: Round Robin example 3

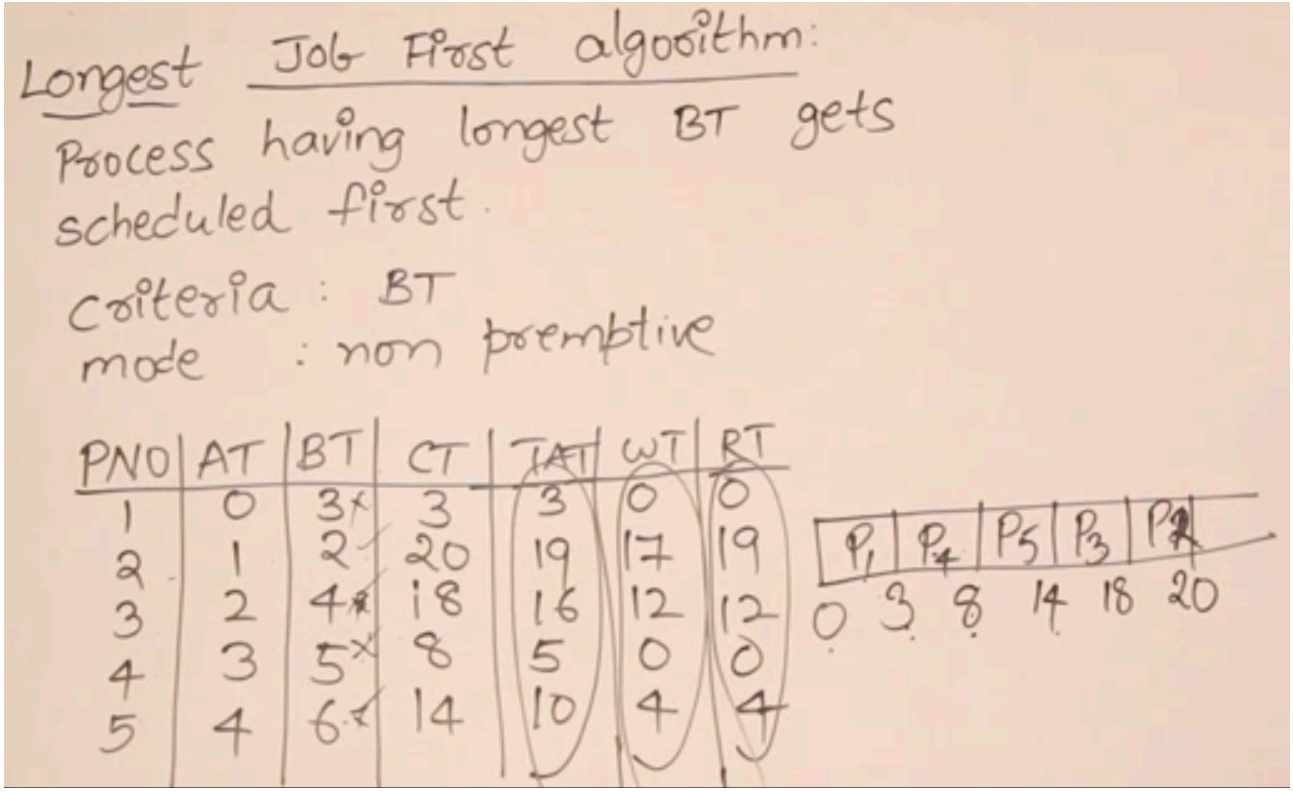


Lecture #24: Round Robin example 4

Answer: $q \leq \frac{t - nS}{n - 1}$



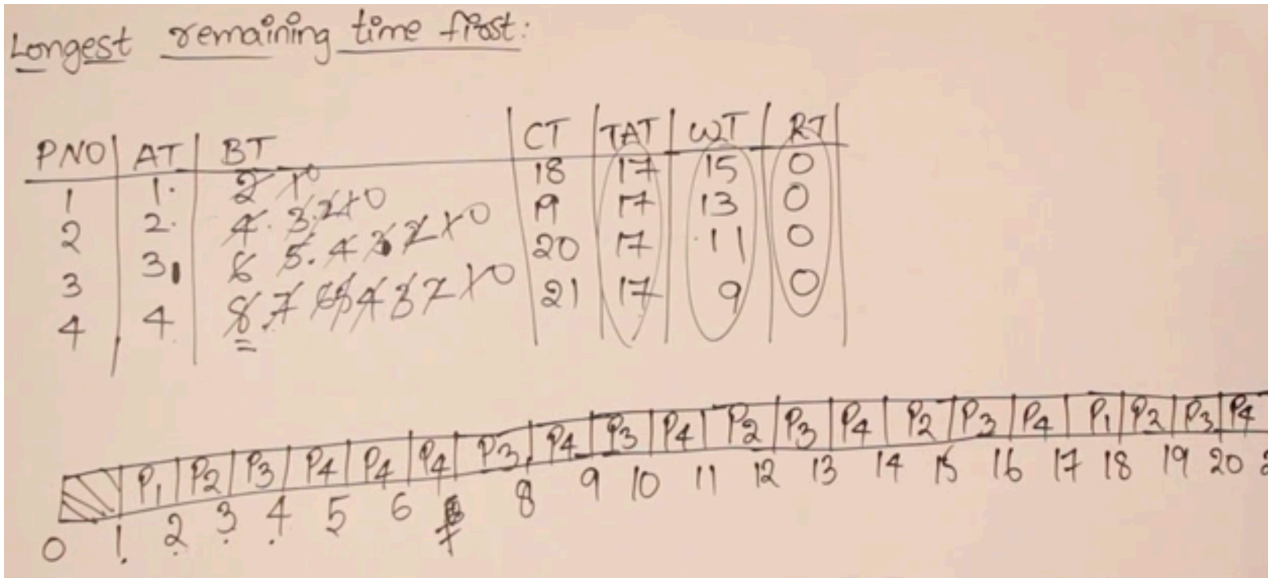
Lecture #25: Longest job first (LJF)



- If we change the BTs to their negative values, then we'll have LJF algorithm
- *in non preemptive modes of scheduling algorithms,* $RT = WT$ ***always***
- $\therefore RT$ of p_2 is 17, not 19

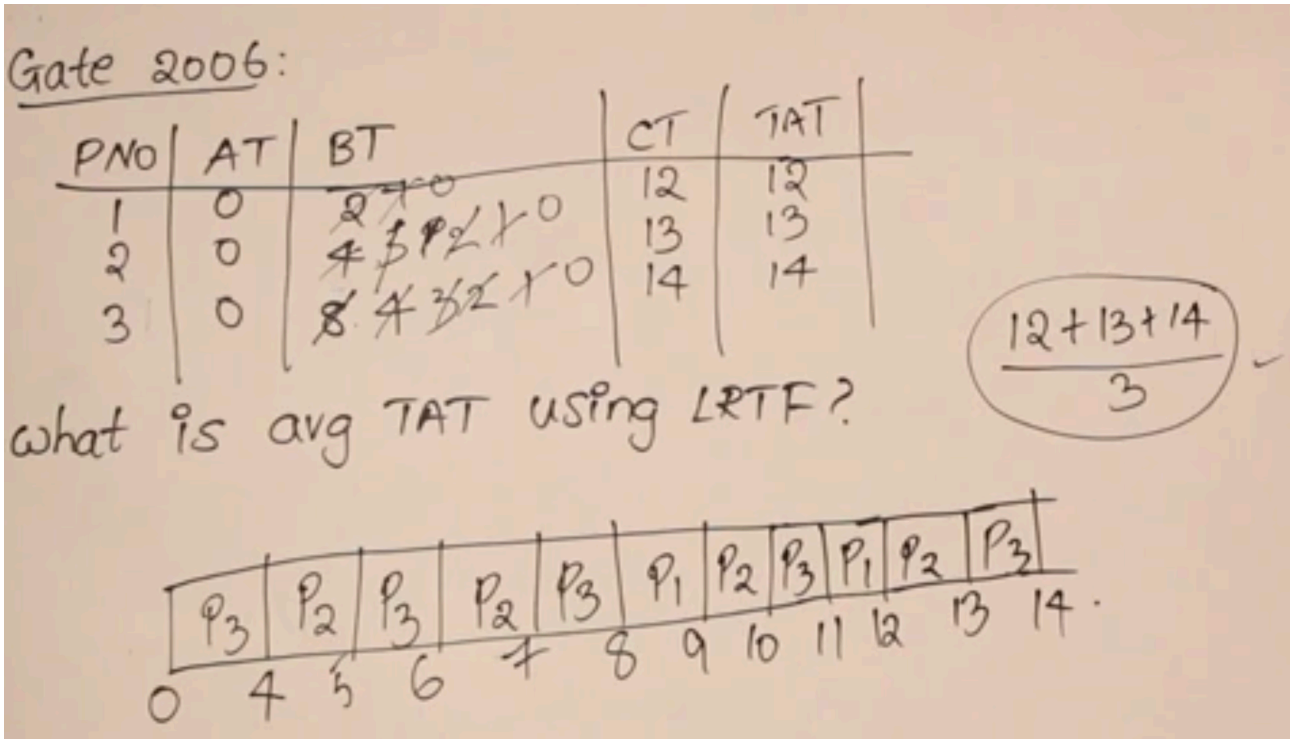
Lecture #26: Longest remaining time first

- Preemptive version of LJF
- Changing the BTs to their negative values, we'll get LRTF (except a few major changes)
- The only difference between LRTF and SRTF is that when that when we had all the processes in the ready queue of SRTF, we then converted it to SJF, but in LRTF we cannot convert it to LJF as when we execute each longest jobs, their BTs keep decreasing, which would eventually make the BT of the executing process lower than other processes' BTs. $\therefore$ we would have to context switch to the process with largest BT
- Whenever two processes have same BT, we'll choose the process with the smaller AT



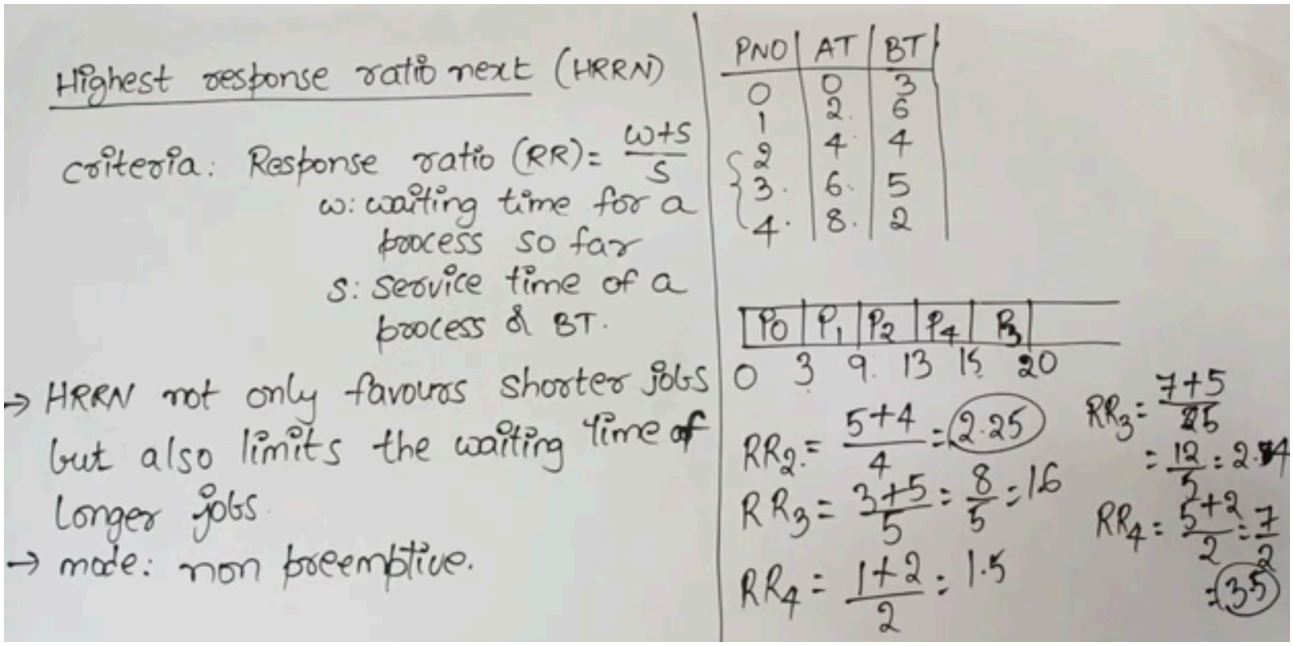
- here it so happened that whichever new process that arrived happened to have the longest BT
- ∴ all processes' RT is zero and TAT is same, but it need not be true for all examples
- In LRTF the completion time of all the processes' will always be consecutive in the order of their ATs

Lecture #27: LRTF Gate 2006 question

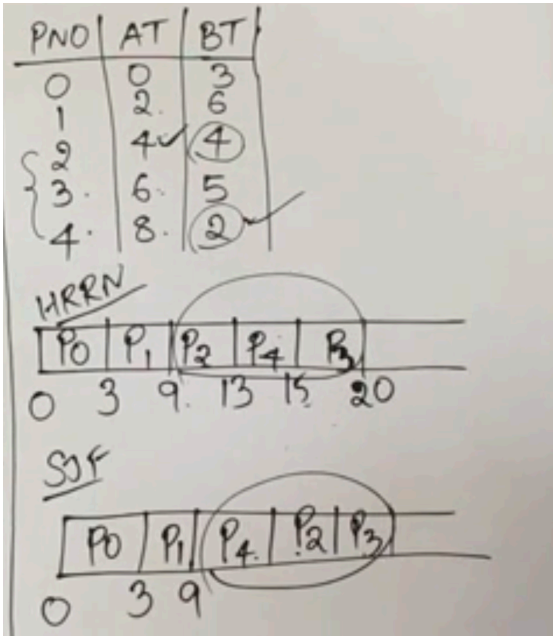


Lecture #28: HRRN

- Highest Response Ratio Next

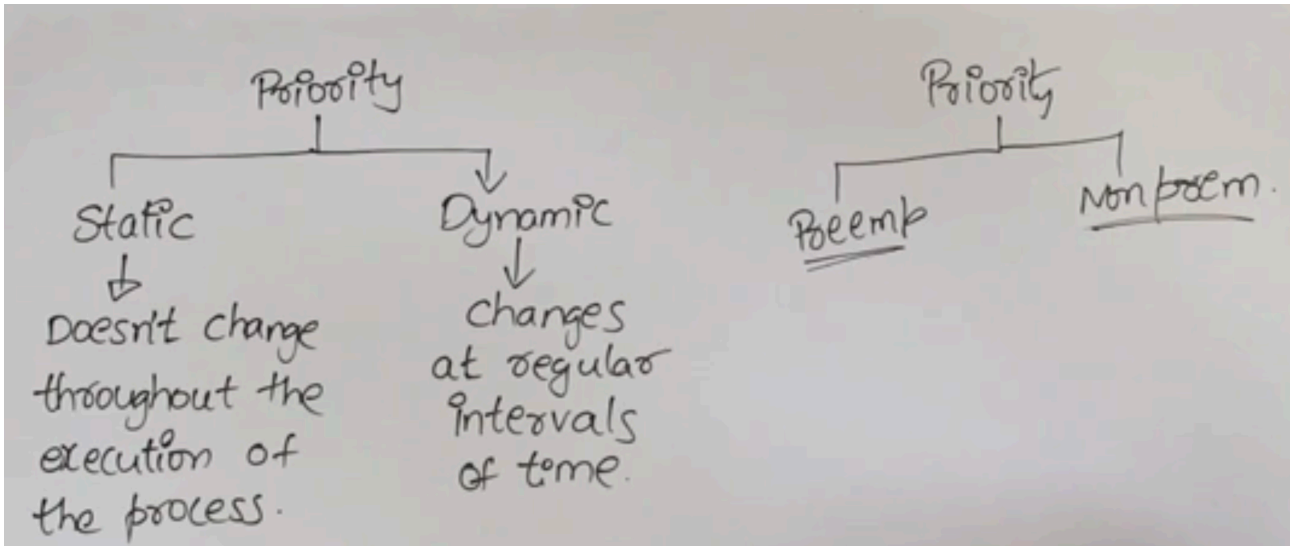


- Response Ratio (RR) $\rightarrow RR = \frac{WT + BT}{BT}$ where WT is ***waiting time so far*** and BT is burst time
- It differs from SJF as it not only favour shorter jobs but also favours jobs with higher WT
- Example of HRRN and SJF applied to the same problem

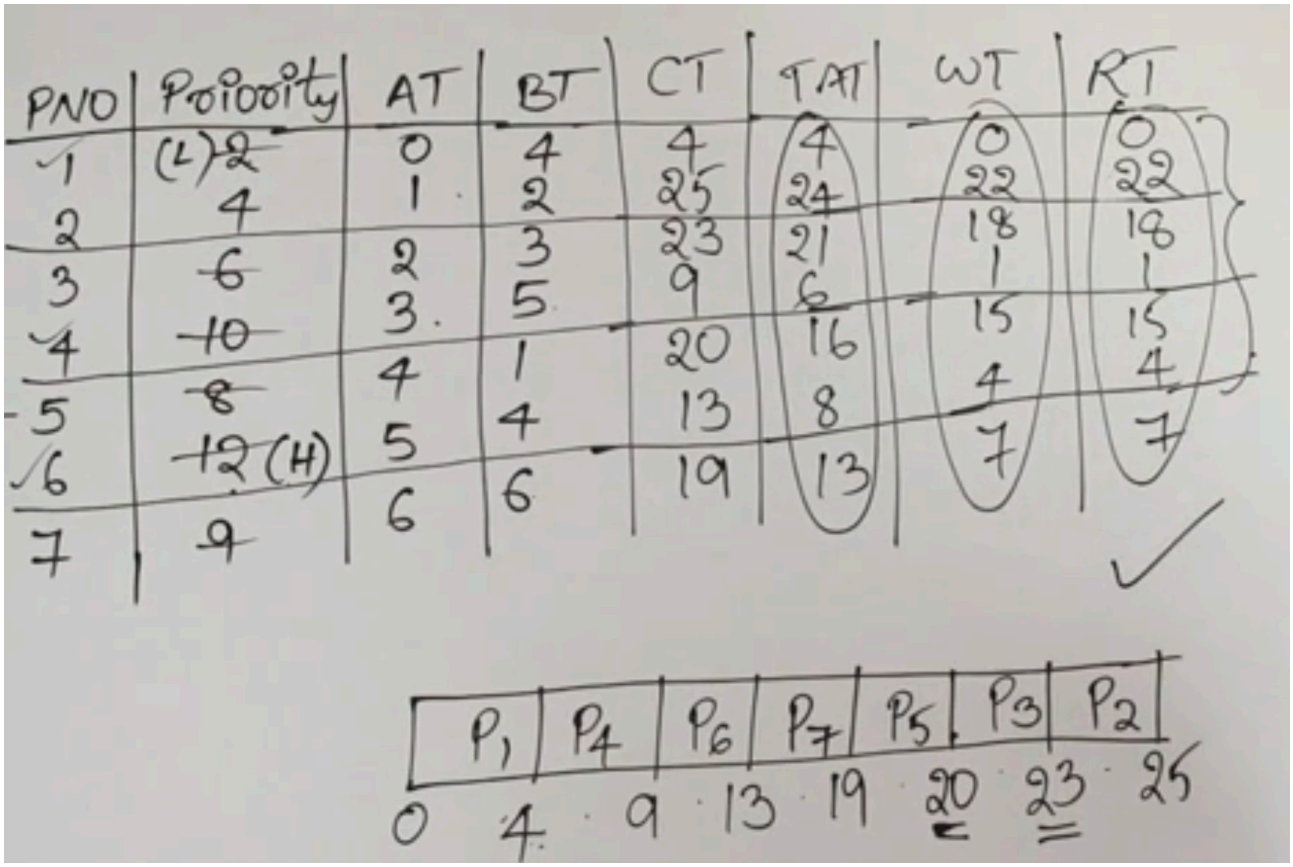


Lecture #29: Priority Scheduling

- Whenever a process is created by the scheduler and comes to the ready queue, it is assigned a priority
- While implementing, lower priority number might indicate higher priority and higher priority number might mean lower priority, though this may vary from implementation to implementation

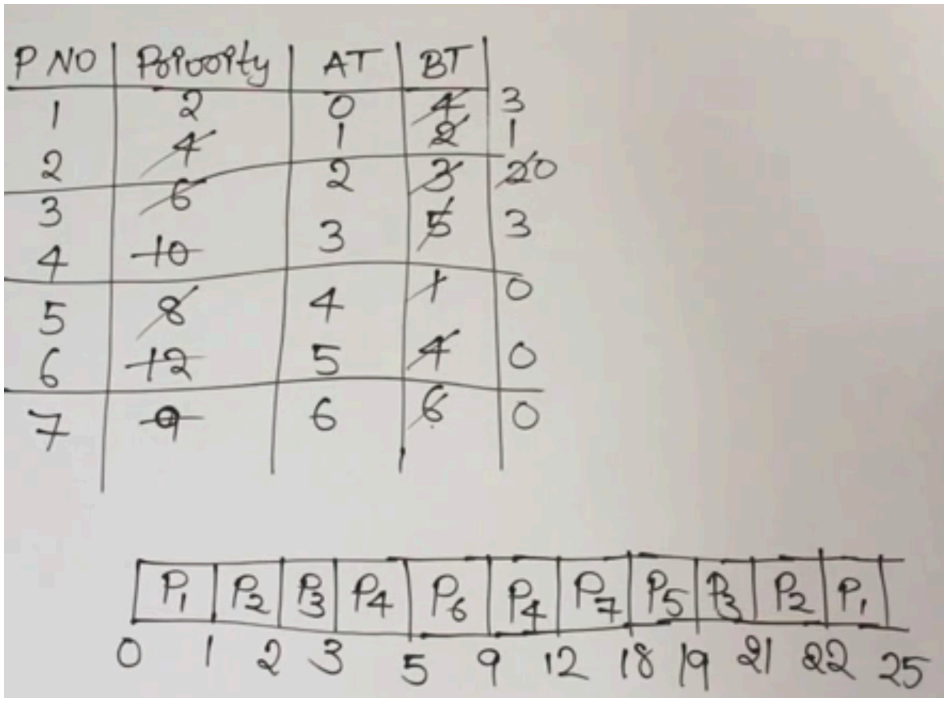


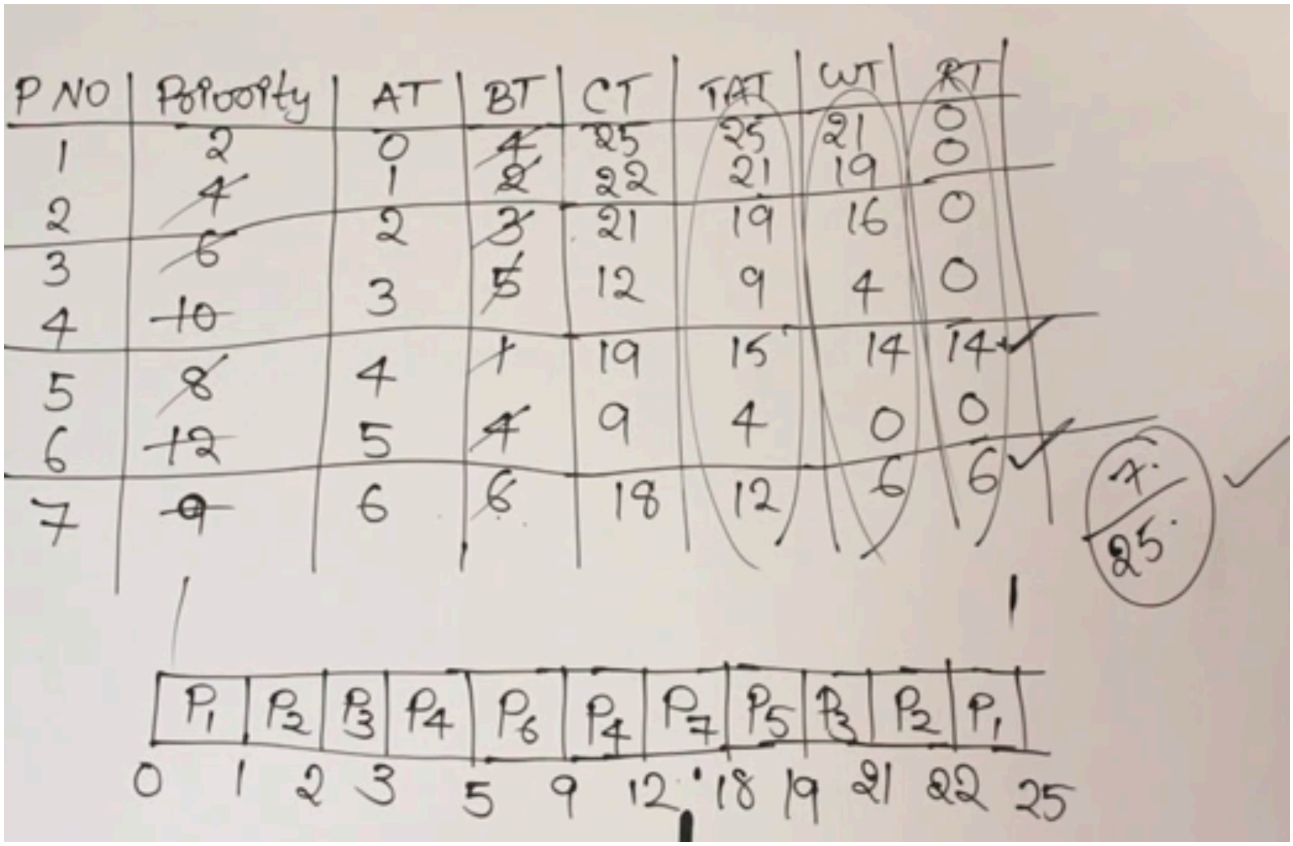
Lecture #30: Non-Preemptive priority scheduling



Lecture #31: Pre-emptive priority scheduling

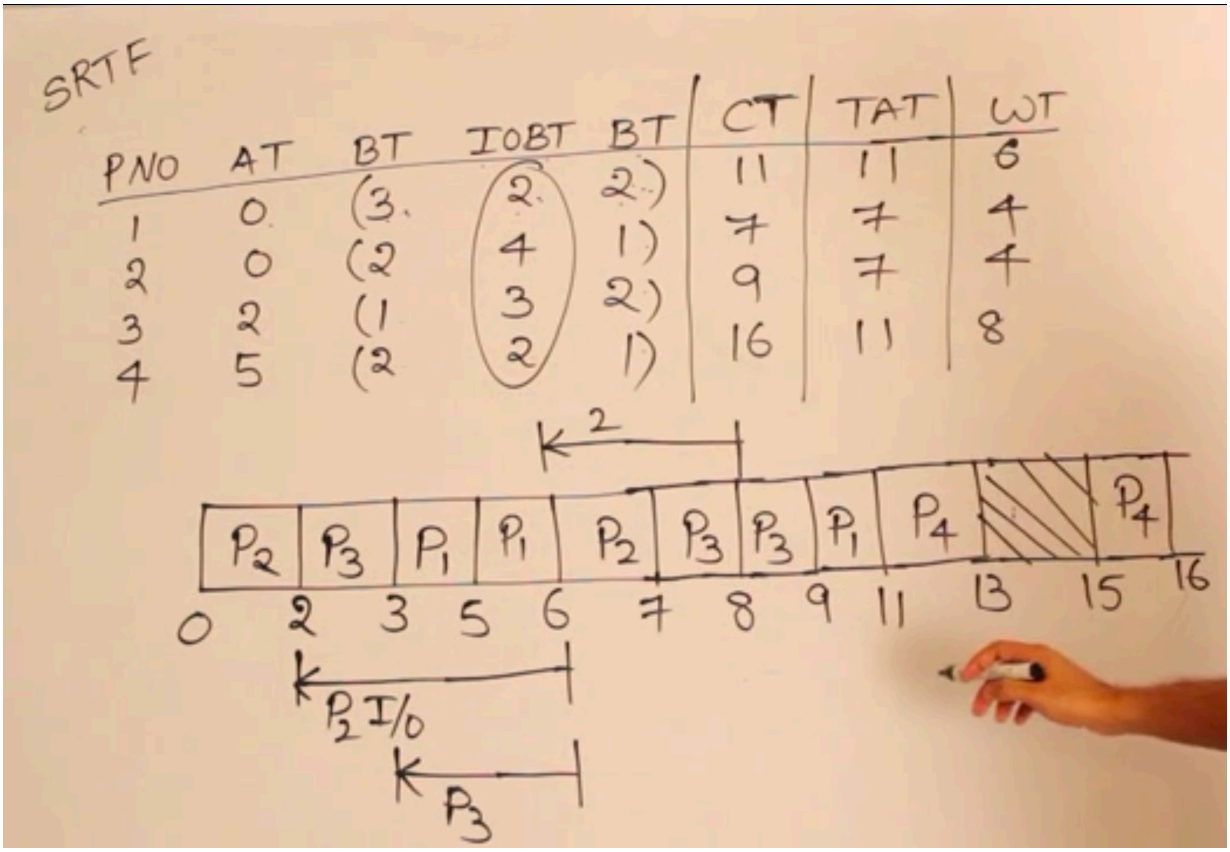
- in this, we will preempt as and when a new process with higher priority comes to the ready queue
- When all the processes are in the ready queue, then it becomes non-preemptive priority scheduling
- When all the processes have the same AT then the non-preemptive and preemptive priority scheduling will be the same



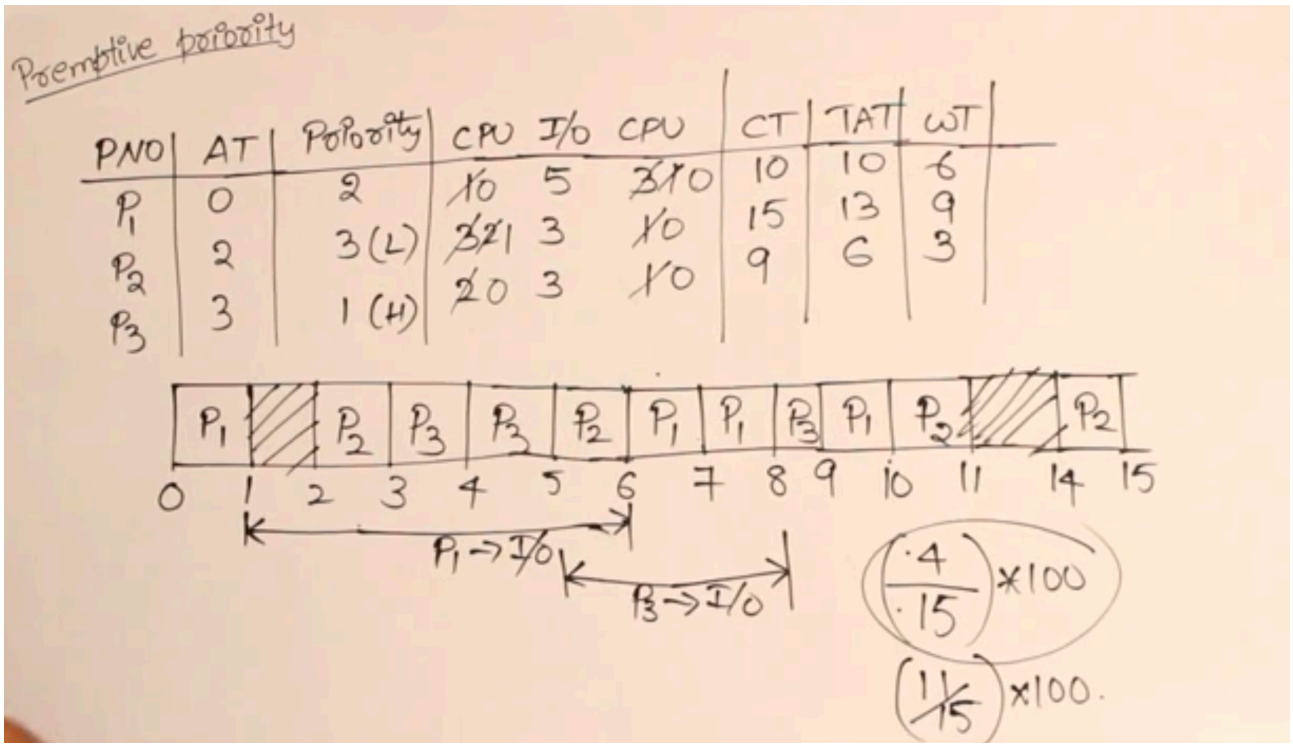


Lecture #32: SRTF with processes containing CPU and IO time example 1

- previously we had to move the processes from ready to run only as we assumed all the processes to be CPU bound
- Now, we will consider the case where processes can have I/O time, and therefore could switch between three states, namely - Ready, Run and Wait/Block states.

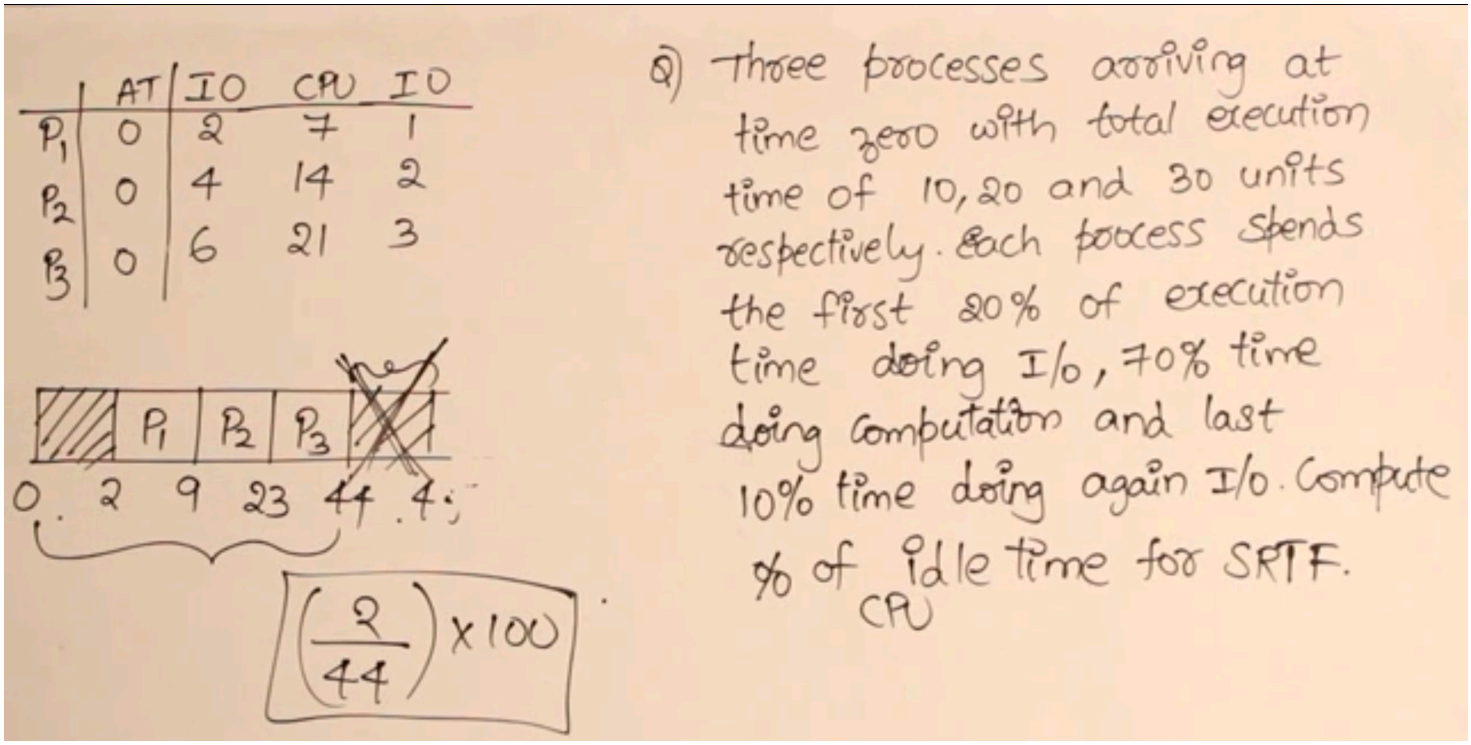


Lecture #33: Pre-emptive priority with processes contains CPU and IO time



- here, the CPU stays idle for 4 units of time, from 1 to 2 and from 11 to 14
- $\therefore$ the CPU inefficiency if x is the unit of time the CPU stays ideal is: $\frac{x}{TCT} \times 100$ where TCT is the total completion time
- $\therefore$ the CPU efficiency η_e is $(1 - \frac{x}{TCT}) \times 100$
- In this example, $\eta_e = (1 - \frac{4}{15}) \times 100 \Rightarrow 73.33\,\%$

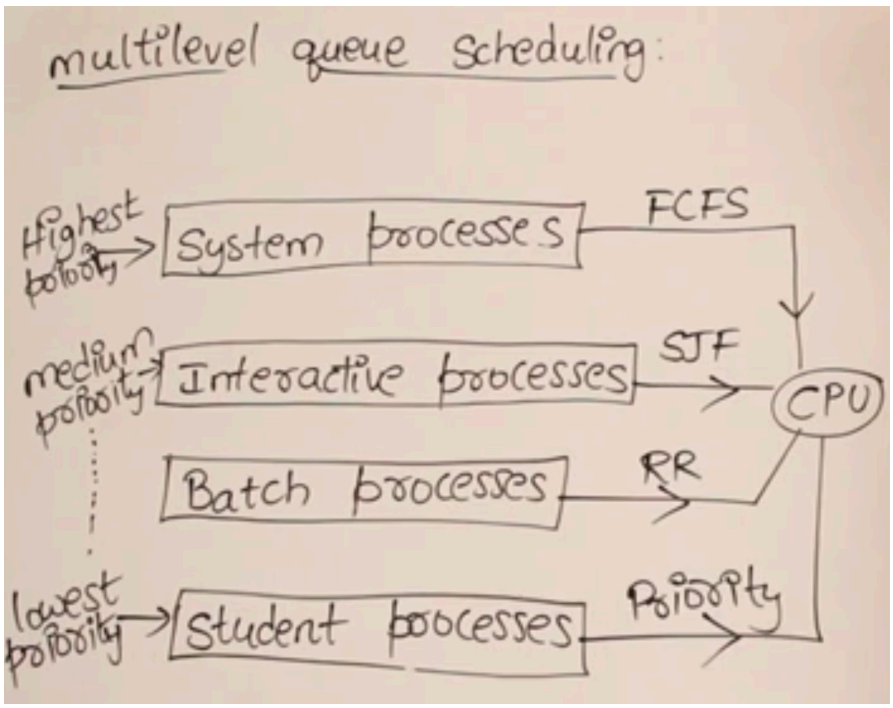
Lecture #34: SRTF with processes contains CPU and IO time example 2



- we would not consider the time 44 to 45 as idle time as at that time we could schedule some other process if we had any

Lecture #35: Multi level queues and multi level feedback queues

- we cannot put all the processes in a single queue and apply the same scheduling algorithm to the queue
- ∴ we have different levels of queue for different kinds of processes



- a disadvantage of this method is that there will occur starvation of lower level queues as the CPU can only take one process at a time
- One advantage is that we can apply different kinds of scheduling algorithm to different queues as shown in the diagram
- The solution to the disadvantage is that we can move the processes up a level if it is staying for too long in some queue
- This is called Multi-level Feedback Scheduling, and by implementing this we can avoid starvation of processes completely