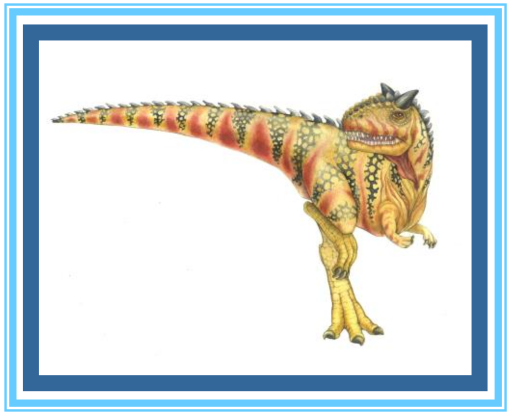
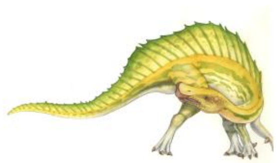
Module 2
Chapter 6: CPU Scheduling

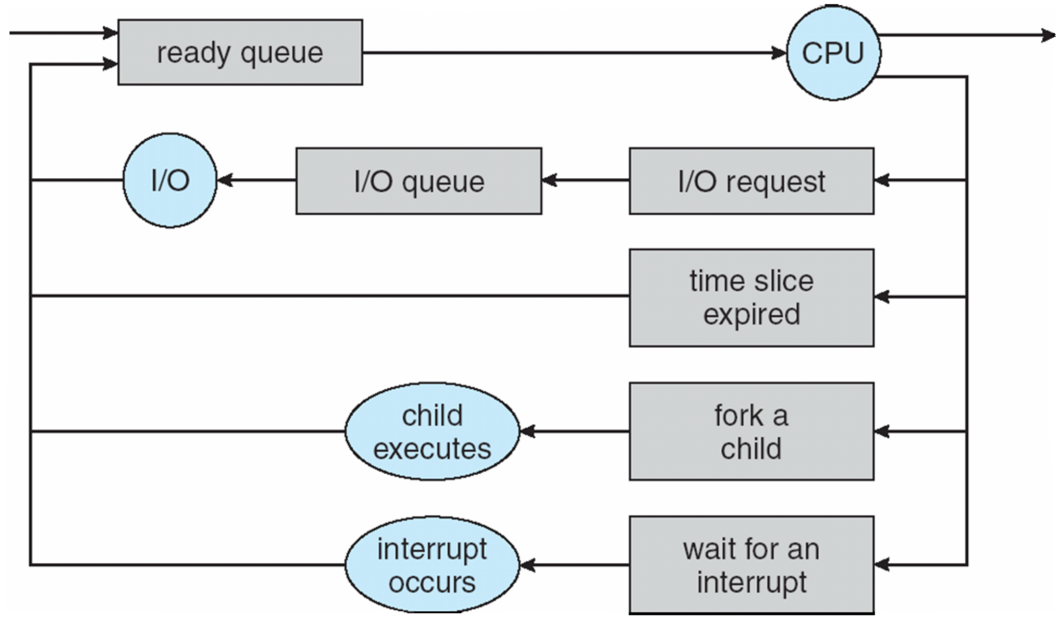


Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Algorithm Evaluation



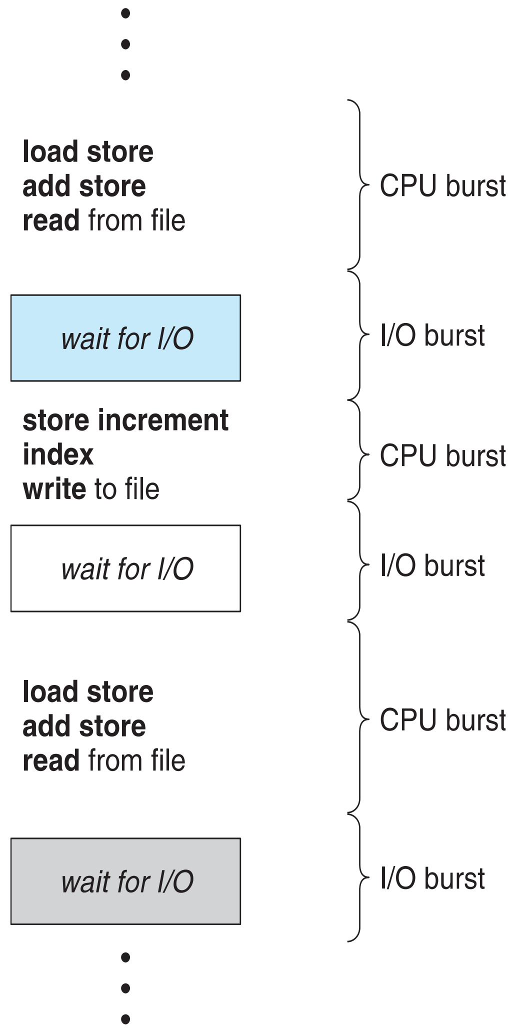
CPU Scheduling



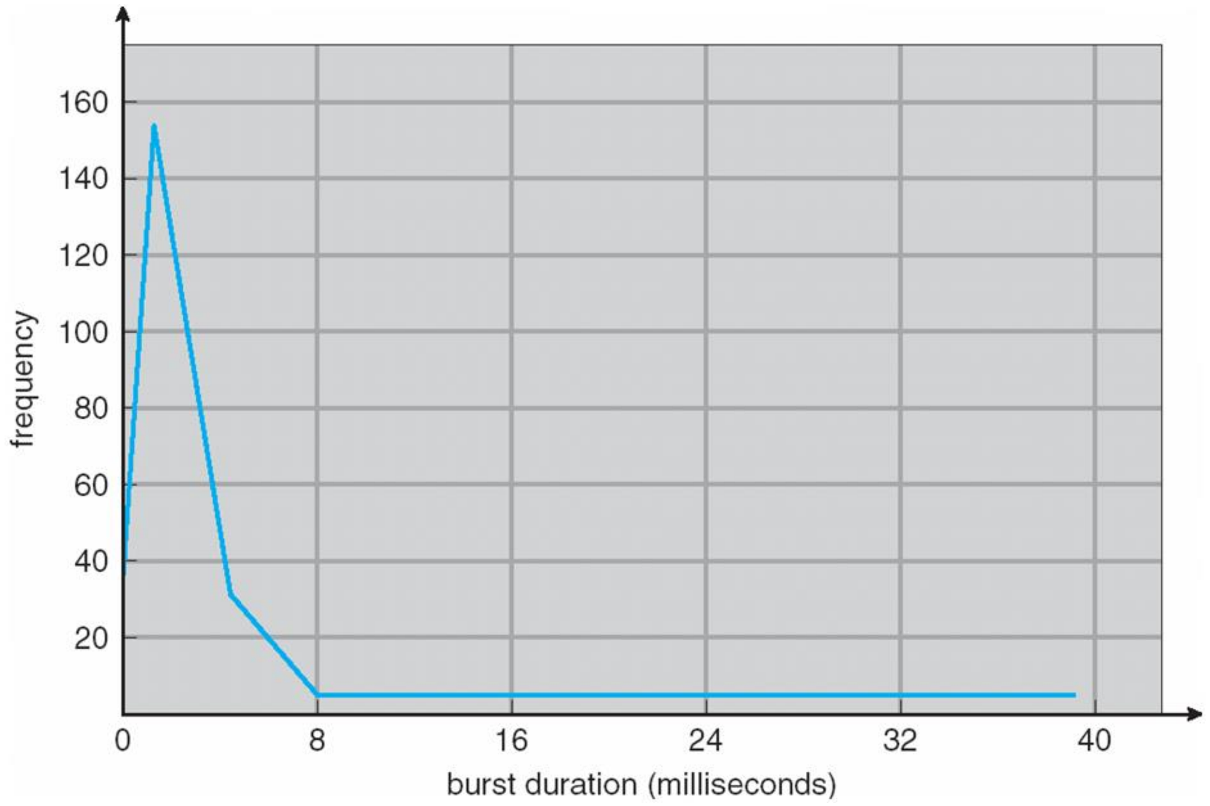
- How is the OS to decide which of several tasks to take off a queue?
- Scheduling: deciding which threads are given access to resources from moment to moment.

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern

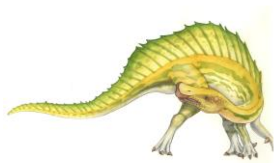


Histogram of CPU-burst Times



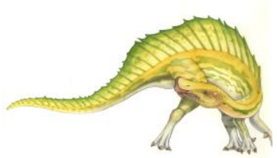
CPU Scheduling

- CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold.
- The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution.
- The selection process will be carried out by the CPU scheduler.
- It selects one of the processes in memory that are ready for execution.



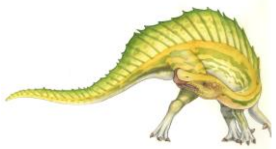
Types of CPU Scheduling

- **Preemptive Scheduling**
  - In Preemptive Scheduling, the tasks are mostly assigned with their priorities.
  - Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running.
  - The lower priority task holds for some time and resumes when the higher priority task finishes its execution.



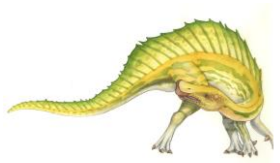
Types of CPU Scheduling

- **Non-Preemptive Scheduling**
  - In this type of scheduling method, the CPU has been allocated to a specific process.
  - The process that keeps the CPU busy will release the CPU either by switching context or terminating.
  - It doesn't need special hardware (for example, a timer) like preemptive scheduling



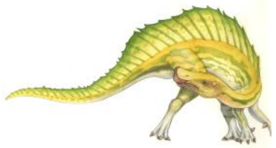
CPU Scheduler

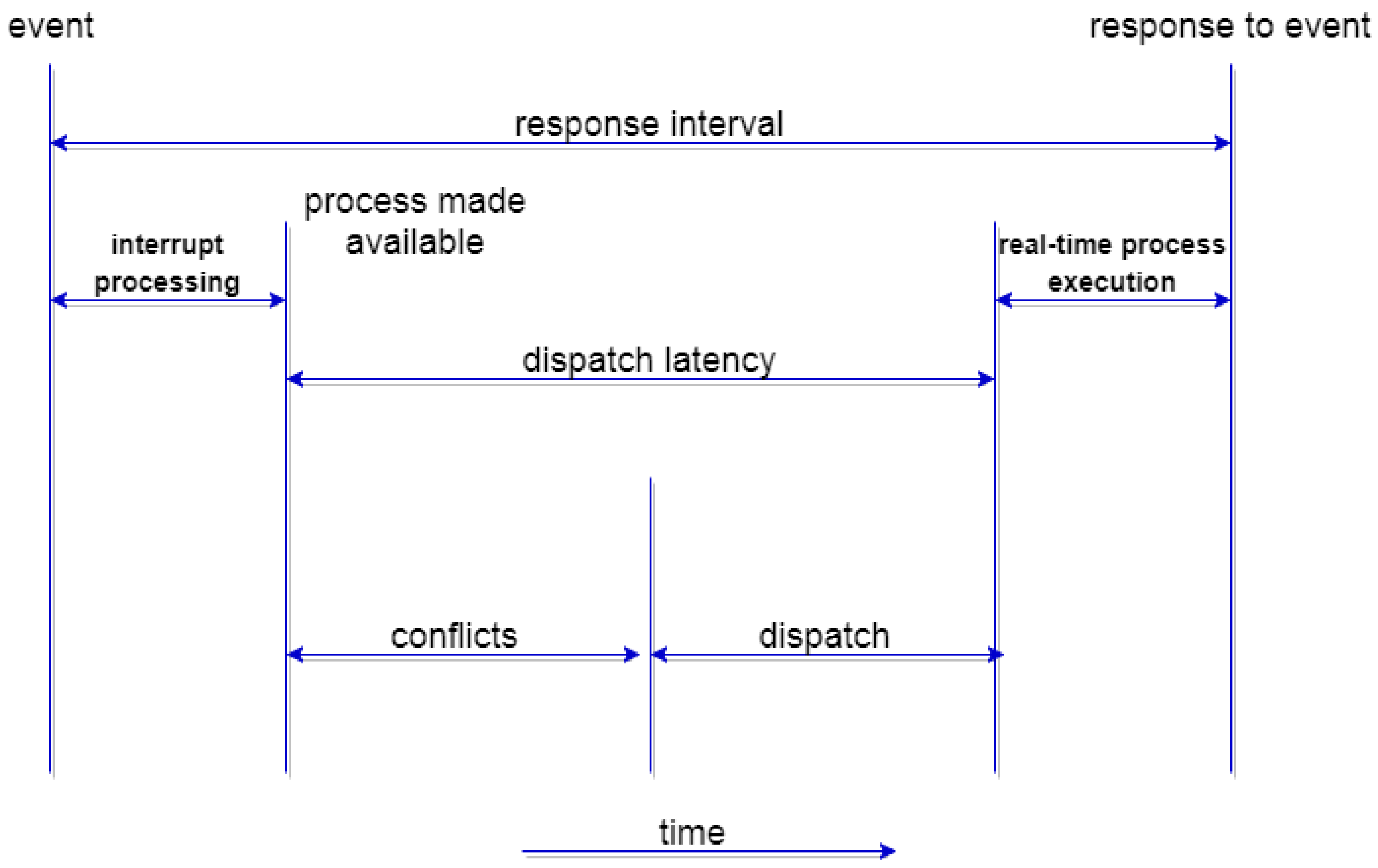
- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities



Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running



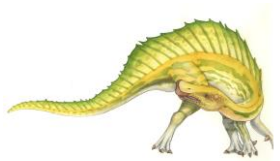

event
response to event
response interval
process made available
interrupt processing
real-time process execution
dispatch latency
conflicts
dispatch
time

Important CPU scheduling Terminologies

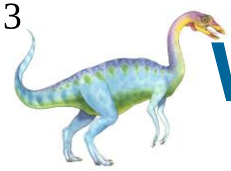
- **Burst Time/Execution Time:** It is the time required by the process to complete execution. It is also called running time.
- **Arrival Time:** when a process enters in a ready state
- **Finish Time:** when process complete and exit from a system
- **Multiprogramming:** A number of programs which can be present in memory at the same time.
- **Jobs:** It is a type of program without any kind of user interaction.
- **User:** It is a kind of program having user interaction.
- **Process:** It is the reference that is used for both job and user.
- **CPU/IO burst cycle:** Characterizes process execution, which alternates between CPU and I/O activity. CPU times are usually shorter than the time of I/O.

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

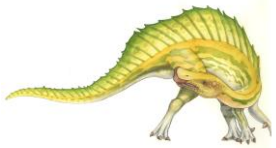


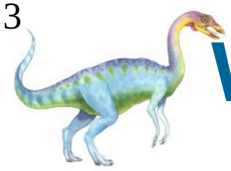
What is Important in a Scheduling Algorithm?



What is Important in a Scheduling Algorithm?

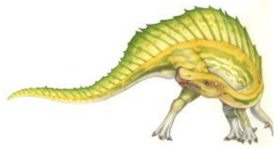
- Minimize Response Time
  - Elapsed time to do an operation (job)
  - Response time is what the user sees
    - Time to echo keystroke in editor
    - Time to compile a program
    - Real-time Tasks: Must meet deadlines imposed by World

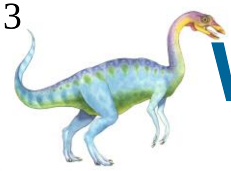




What is Important in a Scheduling Algorithm?

- Maximize Throughput
  - Jobs per second
  - Throughput related to response time, but not identical
    - Minimizing response time will lead to more context switching than if you maximized only throughput
  - Minimize overhead (context switch time) as well as efficient use of resources (CPU, disk, memory, etc.)

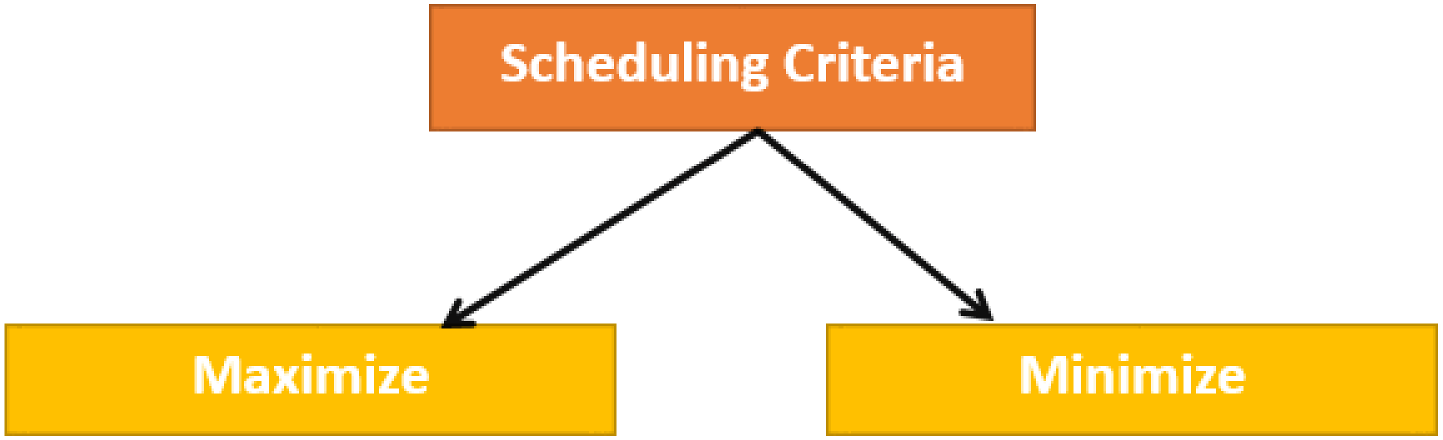




What is Important in a Scheduling Algorithm?

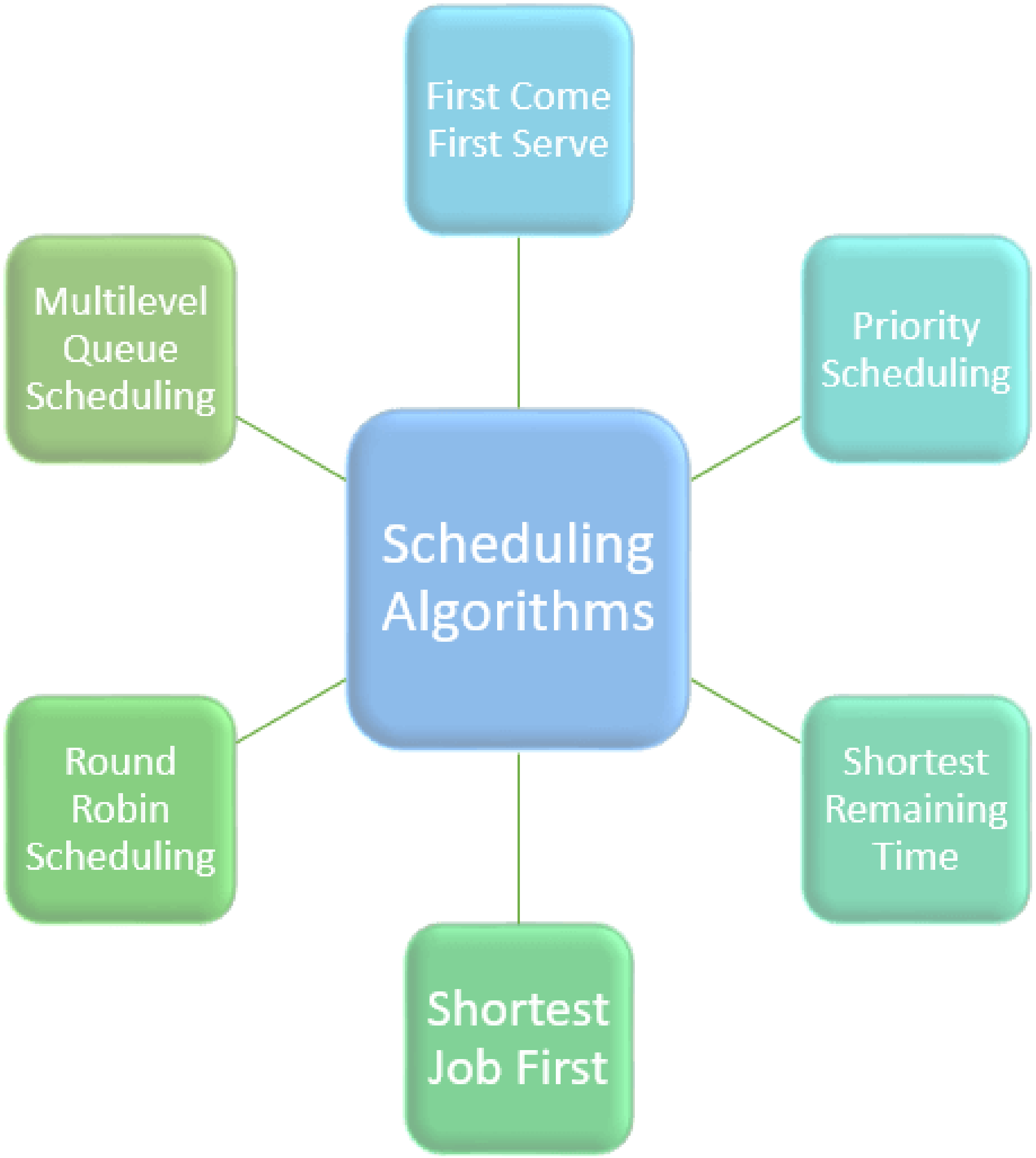
- Fairness
  - Share CPU among users in some equitable way
  - Not just minimizing average response time

Scheduling Algorithm Optimization Criteria



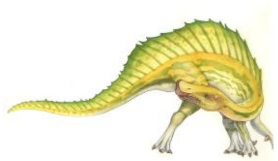
Maximize:
CPU Utilization
Throughput

Minimize:
Turnaround time
Waiting time
Response time


First Come First Serve
Priority Scheduling
Shortest Remaining Time
Shortest Job First
Round Robin Scheduling
Multilevel Queue Scheduling
Scheduling Algorithms

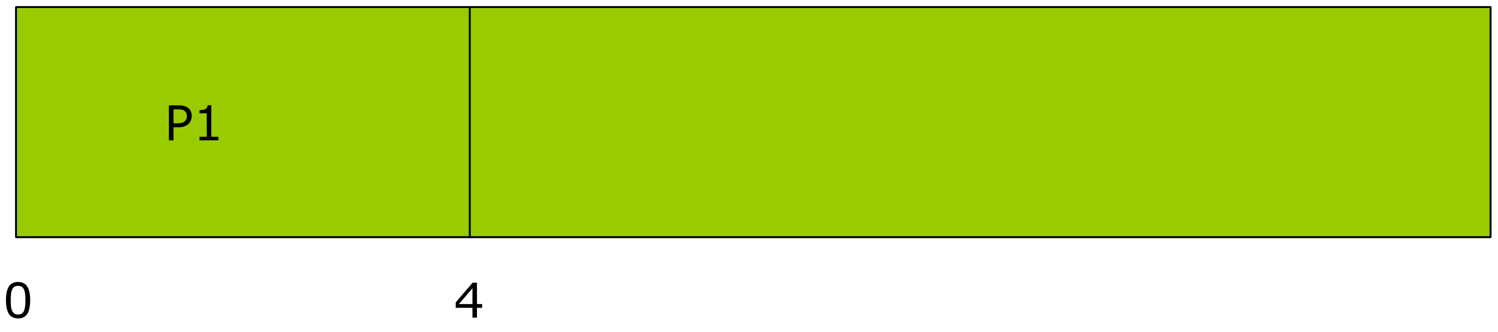
Gantt Chart

Process	Burst Time
P1	4
P2	3
P3	5



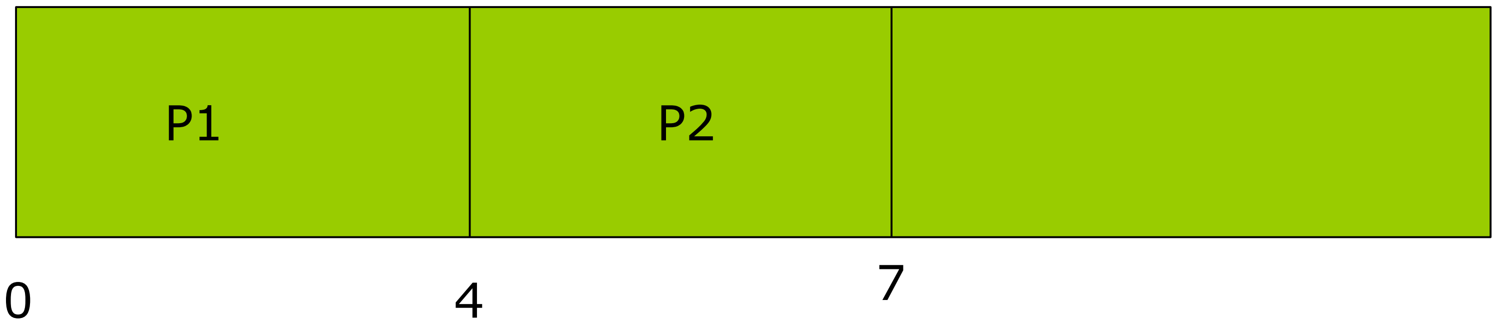
Gantt Chart

Process	Burst Time
P1	4
P2	3
P3	5



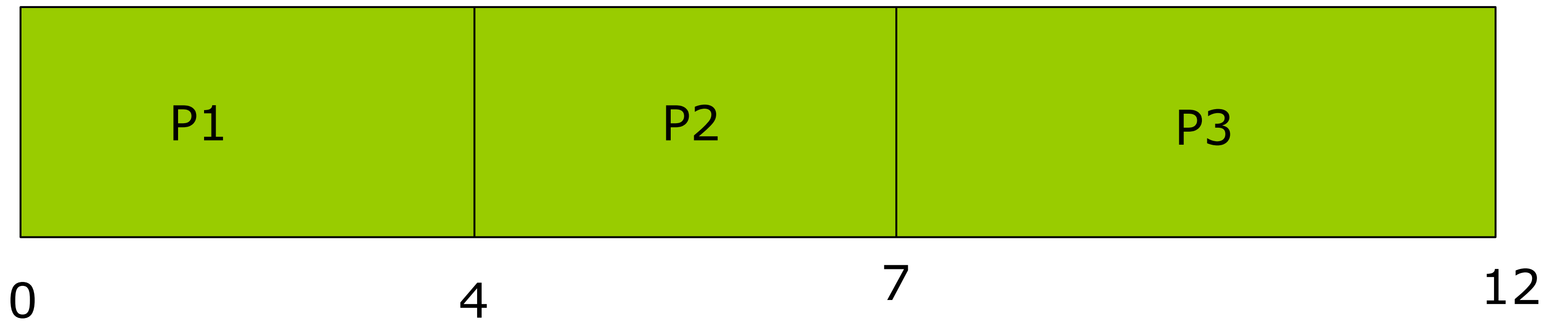
Gantt Chart

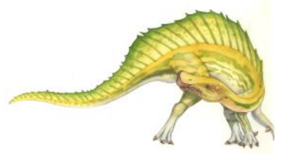
Process	Burst Time
P1	4
P2	3
P3	5



Gantt Chart

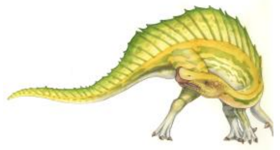
Process	Burst Time
P1	4
P2	3
P3	5





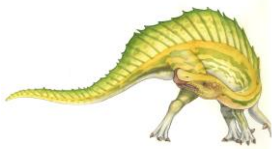
Simple formulae for calculating various times for given processes:

- **Completion Time**: Time taken for the execution to complete, starting from arrival time.
- **Turn Around Time**: Time taken to complete after arrival.
  - In simple words, it is the difference between the Completion time and the Arrival time.
- **Waiting Time**: Total time the process has to wait before it's execution begins.
  - It is the difference between the Turn Around time and the Burst time of the process.



First Come First Serve (FCFS)

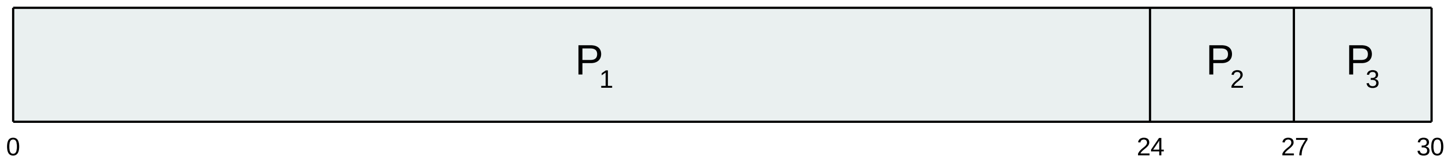
- Jobs are executed on first come, first serve basis.
- It is a non-preemptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.



First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



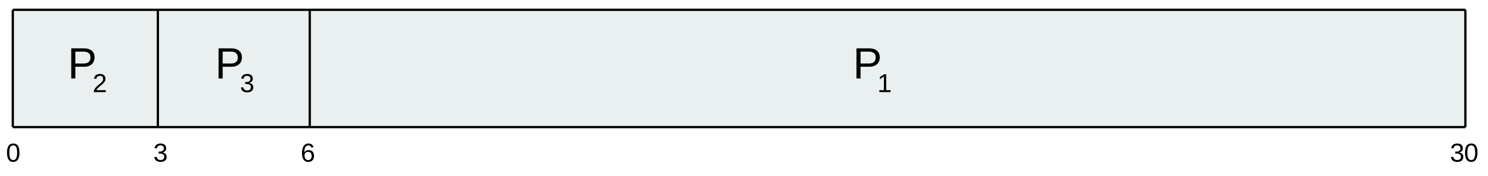
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: (0 + 24 + 27)/3 = 17
- Completion Time: $P_1 = 24$; $P_2 = 27$; $P_3 = 30$
- Average Completion Time: (24+27+30)/3 = 27

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

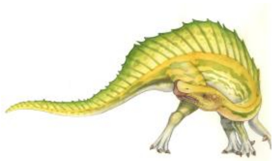
- The Gantt chart for the schedule is:

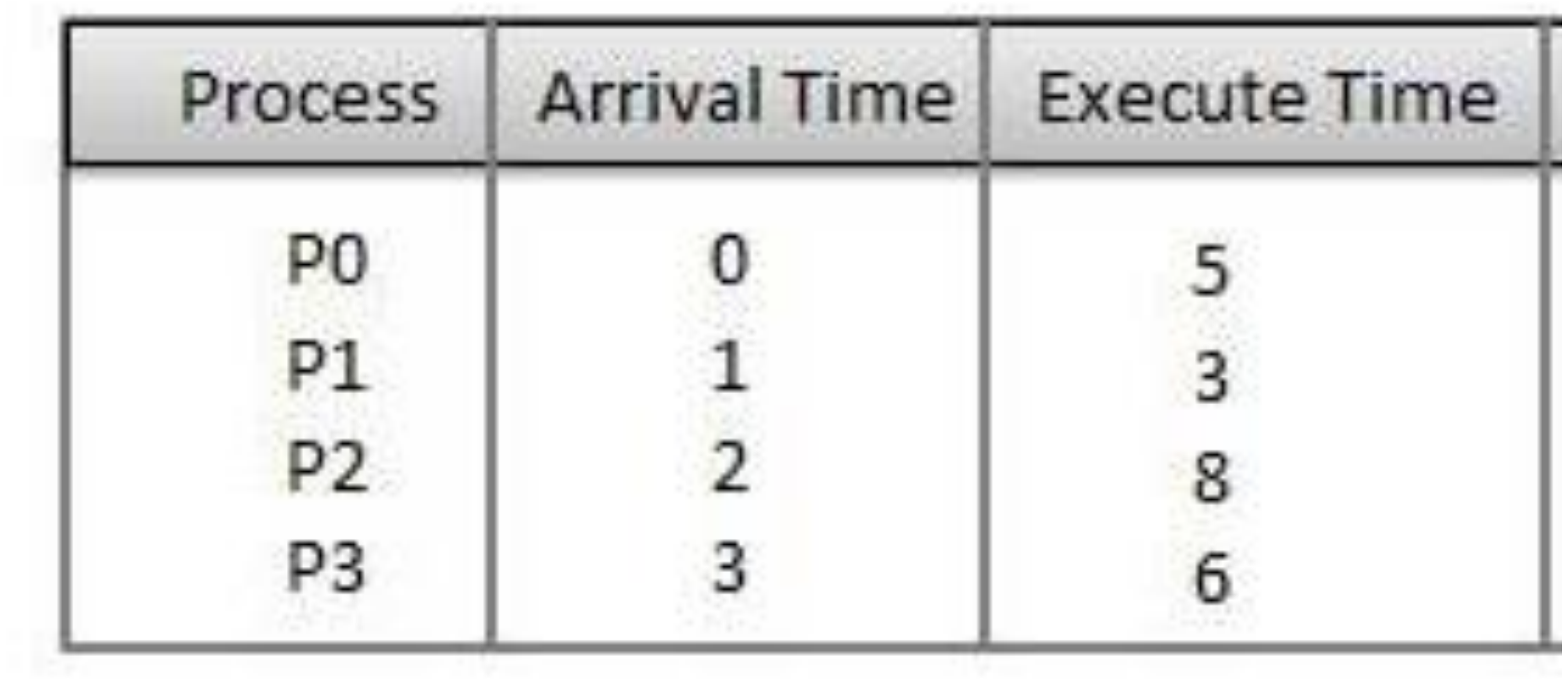


- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3 (compared to 17)
- Completion Time: P_1 = 30; P_2 = 3; $P_3 = 6$
- Average Completion Time: (30+3+6)/3 = 13 (compared to 27)
- Much better than previous case

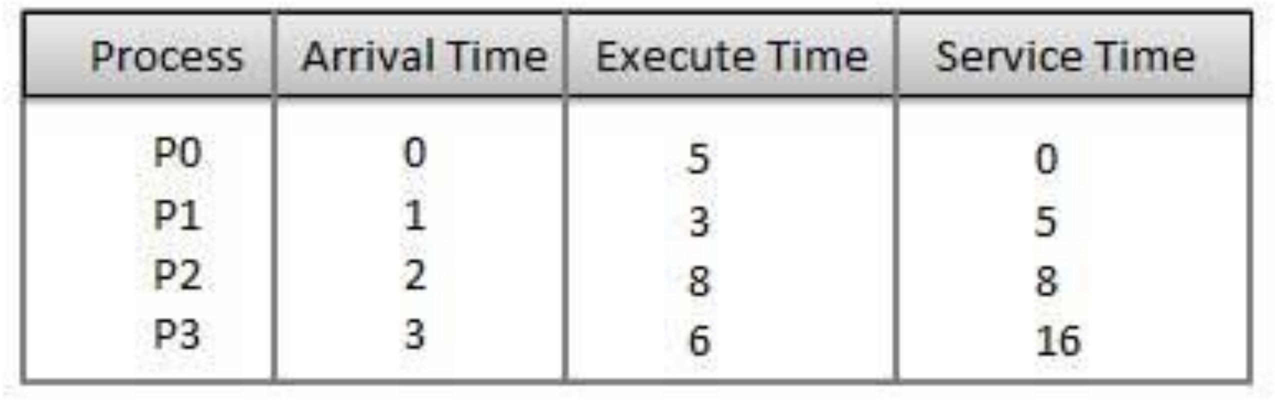
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

- Convoy Effect is a situation where many processes, who need to use a resource for short time are blocked by one process holding that resource for a long time.
- This essentially leads to poor utilization of resources and hence poor performance.

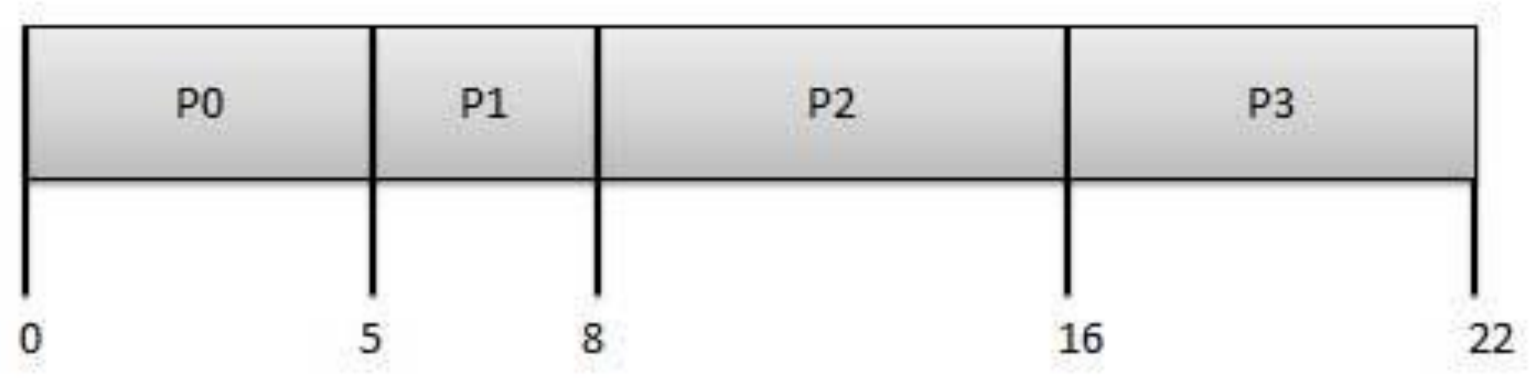




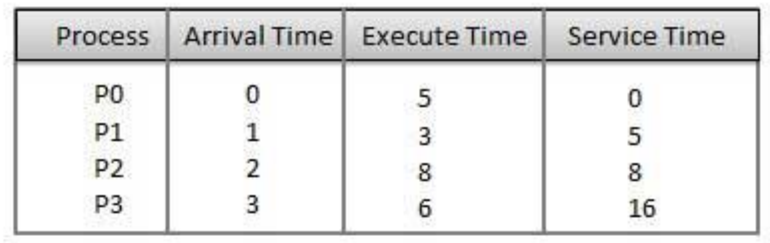
Process	Arrival Time	Execute Time
P0	0	5
P1	1	3
P2	2	8
P3	3	6



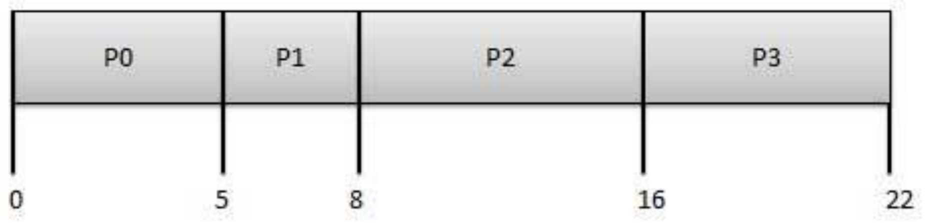
Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



- Turn around time = CT- AT
- = WT + BT



Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



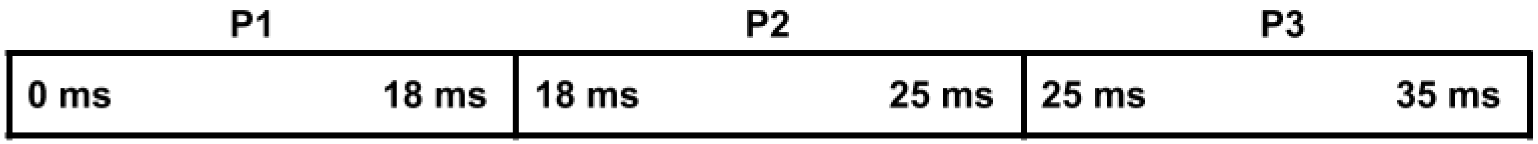
Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Process	Arrival time	Burst time
P1	0 ms	18 ms
P2	2 ms	7 ms
P3	2 ms	10 ms

Process	Arrival time	Burst time
P1	0 ms	18 ms
P2	2 ms	7 ms
P3	2 ms	10 ms

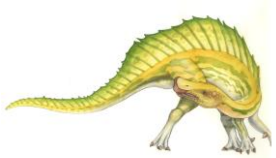
Gantt Chart



```
---------------------------------------------
| Process | Waiting Time | Turnaround Time |
 --------------------------------------------
| P1      | 0ms              | 18ms             |
| P2      | 16ms             | 23ms             |
| P3      | 23ms             | 33ms             |
---------------------------------------------

Total waiting time: (0 + 16 + 23) = 39ms
Average waiting time: (39/3) = 13ms

Total turnaround time: (18 + 23 + 33) = 74ms
Average turnaround time: (74/3) = 24.66ms
```

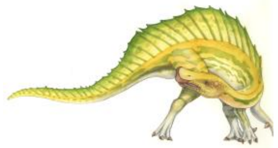


Pros and Cons of FCFS

- **Advantages of FCFS:**
  - It is the most simple scheduling algorithm and is easy to implement.
- **Disadvantages of FCFS:**
  - This algorithm is non-preemptive so you have to execute the process fully and after that other processes will be allowed to execute.
  - Throughput is not efficient.
  - FCFS suffers from the **Convey effect** i.e. if a process is having very high burst time and it is coming first, then it will be executed first irrespective of the fact that a process having very less time is there in the ready state

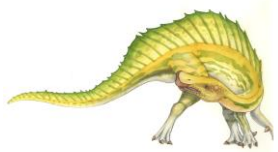
Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.



Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user



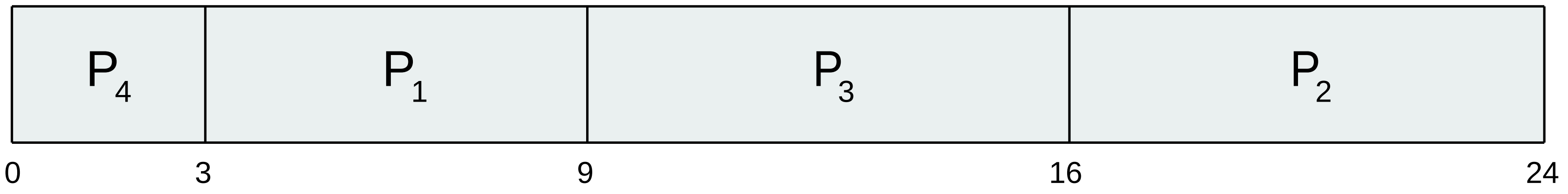
Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

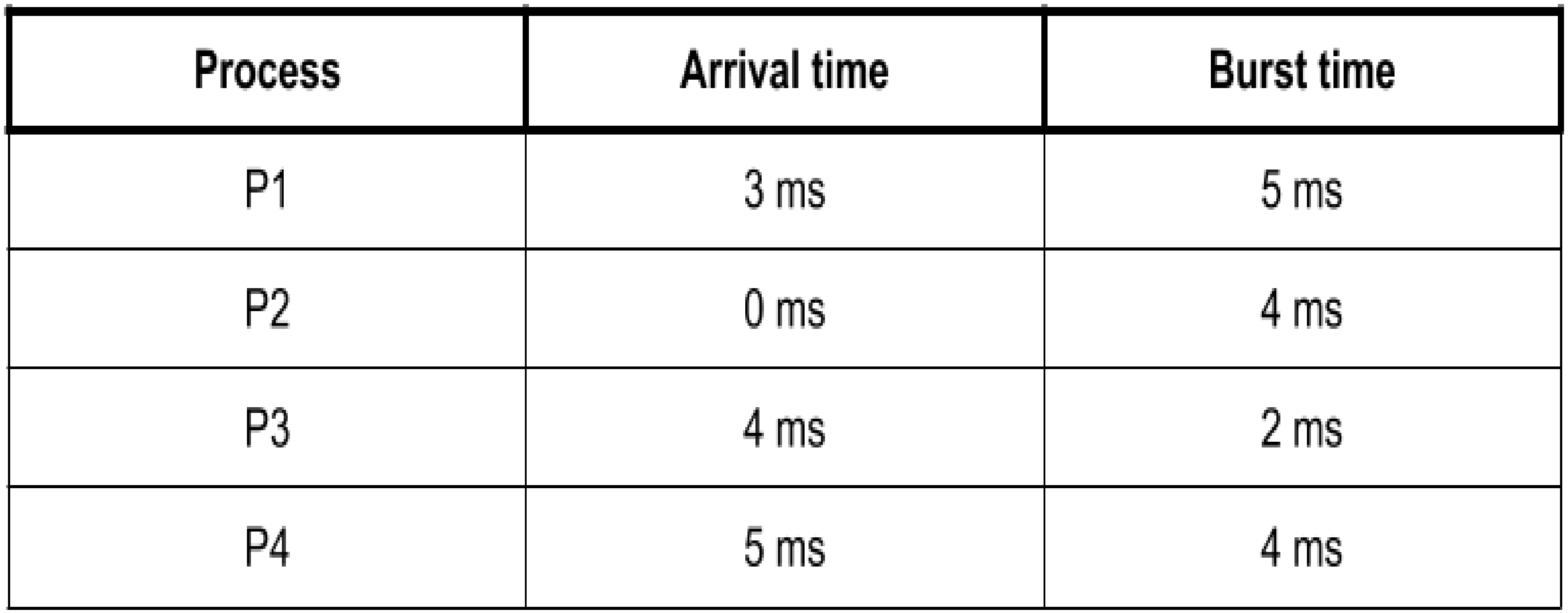
Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



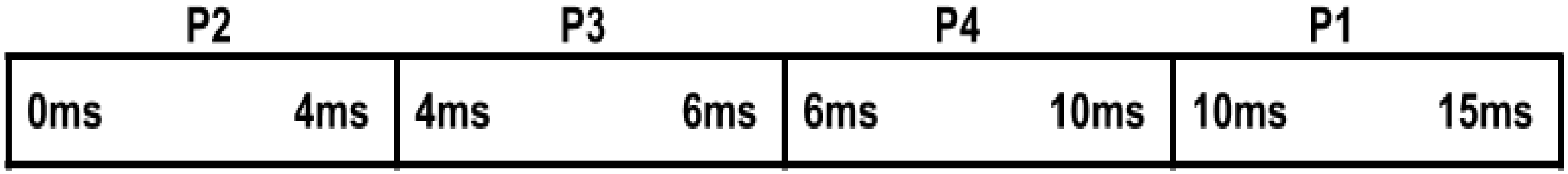
- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7



Process	Arrival time	Burst time
P1	3 ms	5 ms
P2	0 ms	4 ms
P3	4 ms	2 ms
P4	5 ms	4 ms

Process	Arrival time	Burst time
P1	3 ms	5 ms
P2	0 ms	4 ms
P3	4 ms	2 ms
P4	5 ms	4 ms

Gantt Chart



Process	Waiting Time	Turnaround Time
P1	7ms	12ms
P2	0ms	4ms
P3	0ms	2ms
P4	1ms	5ms

Total waiting time: (7 + 0 + 0 + 1) = 8ms

Average waiting time: (8/4) = 2ms

Total turnaround time: (12 + 4 + 2 + 5) = 23ms

Average turnaround time: (23/4) = 5.75ms

Practice Problem 1

Find the avg. waiting time and avg. turnaround time using: a) FCFS b) SJF

Process	Burst Time	Arrival Time
P1	2	2
P2	6	5
P3	4	0
P4	7	0
P5	4	7

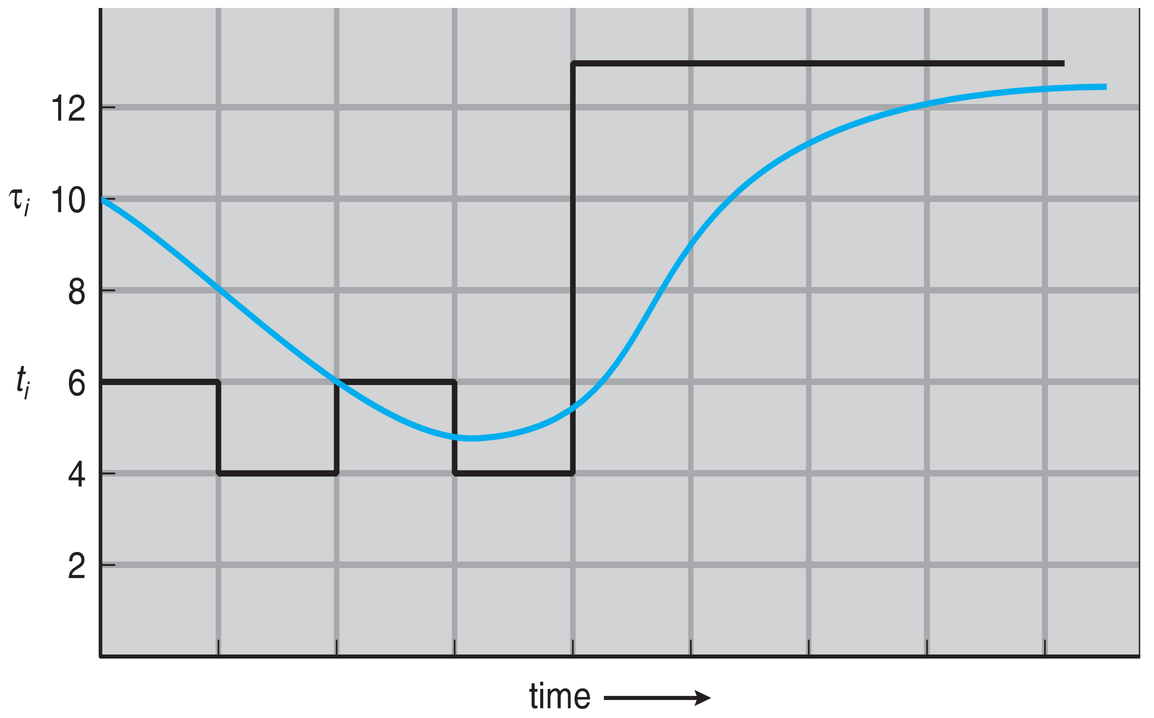
Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$

- Commonly, α set to ½

Prediction of the Length of the Next CPU Burst



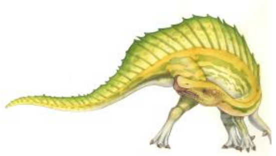
CPU burst (t_i)		6	4	6	4	13	13	13 . . .
"guess" (τ_i)	10	8	6	6	5	9	11	12 . . .

Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} &= \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \dots \\ &\quad + (1 - \alpha)^j \alpha\, t_{n-j} + \dots \\ &\quad + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor



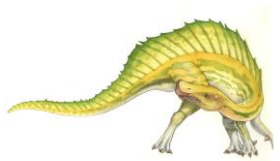
Pros and Cons of SJF

- **Advantages of SJF (non-preemptive):**
  - Short processes will be executed first.
- **Disadvantages of SJF (non-preemptive):**
  - It may lead to starvation if only short burst time processes are coming in the ready state

Preemptive version called **shortest-remaining-time-first**

Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.



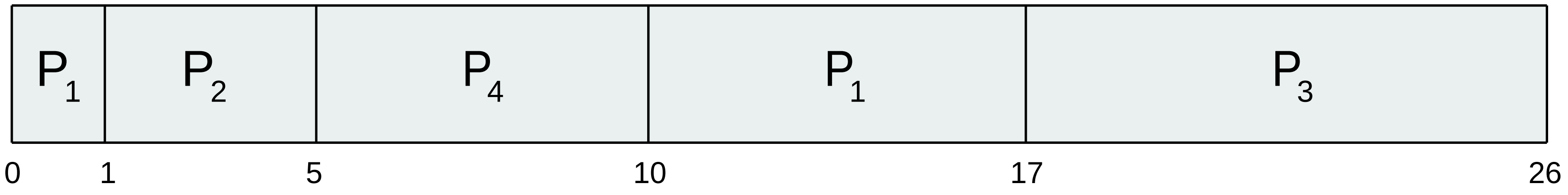
Example of Shortest-remaining-time-first

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

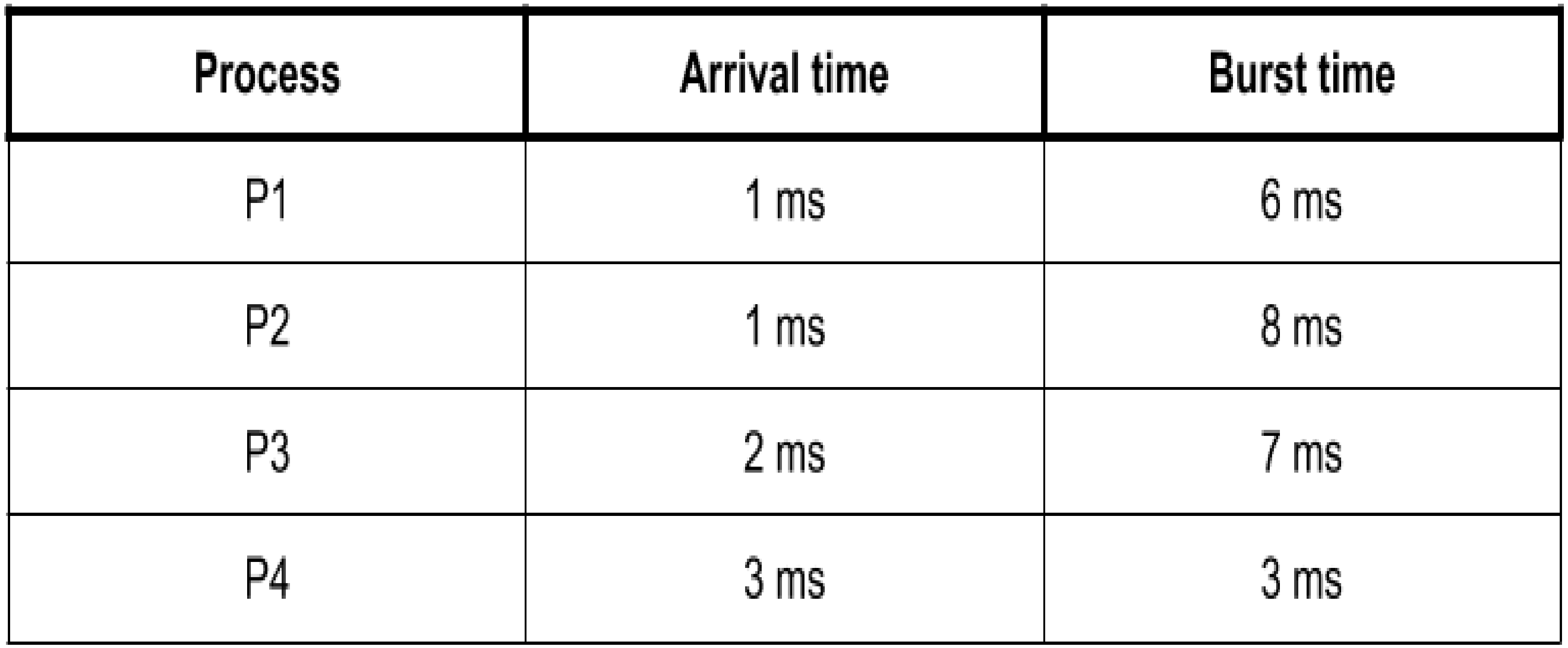
Example of Shortest-remaining-time-first

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



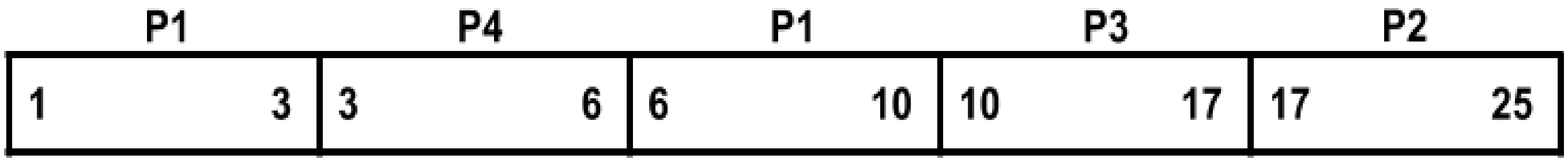
- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec



Process	Arrival time	Burst time
P1	1 ms	6 ms
P2	1 ms	8 ms
P3	2 ms	7 ms
P4	3 ms	3 ms

Process	Arrival time	Burst time
P1	1 ms	6 ms
P2	1 ms	8 ms
P3	2 ms	7 ms
P4	3 ms	3 ms

Gantt Chart



Process	Waiting Time	Turnaround Time
P1	3ms	9ms
P2	16ms	24ms
P3	8ms	15ms
P4	0ms	3ms

Total waiting time: (3 + 16 + 8 + 0) = 27ms

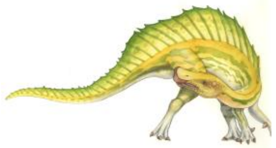
Average waiting time: (27/4) = 6.75ms

Total turnaround time: (9 + 24 + 15 + 3) = 51ms

Average turnaround time: (51/4) = 12.75ms

SRTF Pros and Cons

- **Advantages of SJF (preemptive):**
  - Short processes will be executed first.
  - Optimal (average response time)
- **Disadvantages of SJF (preemptive):**
  - Hard to predict future
  - Unfair, even though we minimized average response time
  - It may result in starvation if short processes keep on coming.



Priority Scheduling

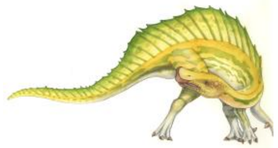
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - **Preemptive Priority Scheduling**: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stopped and the incoming new process with higher priority gets the CPU for its execution.
  - **Non-Preemptive Priority Scheduling**: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

Priority Scheduling

- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
  - A process is considered blocked when it is ready to run but has to wait for the CPU as some other process is running currently.
  - But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution.
  - In 1973, when the IBM 7904 machine was shut down at MIT, a low-priority process was found which was submitted in 1967 and had not yet been run.

Priority Scheduling

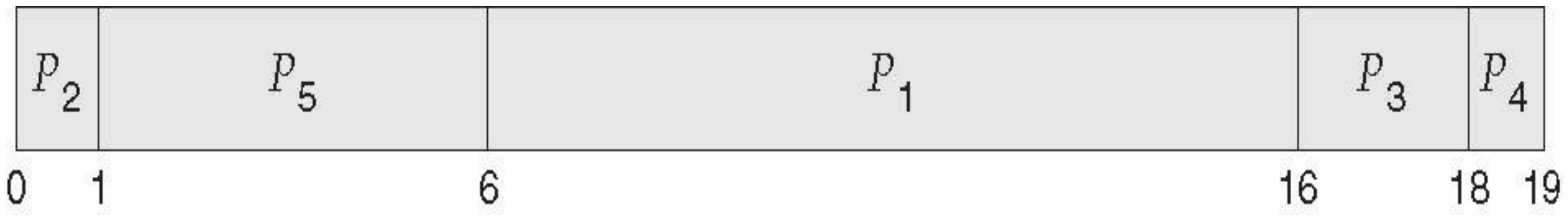
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process
- For example, if we decide the aging factor to be **0.5** for each day of waiting, then if a process with priority **20**(which is comparatively low priority) comes in the ready queue. After one day of waiting, its priority is increased to **19.5** and so on.



Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Process	Arrival time	Burst time	Priority
P1	0 ms	5 ms	1
P2	1 ms	3 ms	2
P3	2 ms	8 ms	1
P4	3 ms	6 ms	3

NOTE: In this example, we are taking higer priority number as higher priority.

Gantt Chart

P1	P4	P2	P3
0ms 5ms	5ms 11ms	11ms 14ms	14ms 22ms

Process	Waiting Time	Turnaround Time
P1	0ms	5ms
P2	10ms	13ms
P3	12ms	20ms
P4	2ms	8ms

Total waiting time: (0 + 10 + 12 + 2) = 24ms

Average waiting time: (24/4) = 6ms

Total turnaround time: (5 + 13 + 20 + 8) = 46ms

Average turnaround time: (46/4) = 11.5ms

- **Advantages of priority scheduling (non-preemptive):**
  - Higher priority processes like system processes are executed first.
- **Disadvantages of priority scheduling (non-preemptive):**
  - It can lead to starvation if only higher priority process comes into the ready state.
  - If the priorities of two or more processes are the same, then we have to use some other scheduling algorithm.

Round Robin

- A fixed time is allotted to each process, called a **quantum**, for execution.
- Once a process is executed for the given time period that process is preempted and another process executes for the given time period.
- Context switching is used to save states of preempted processes.
- This algorithm is simple and easy to implement and the most important is thing is this algorithm is starvation-free as all processes get a fair share of CPU.
- It is important to note here that the length of time quantum is generally from 10 to 100 milliseconds in length.

Round Robin (RR)

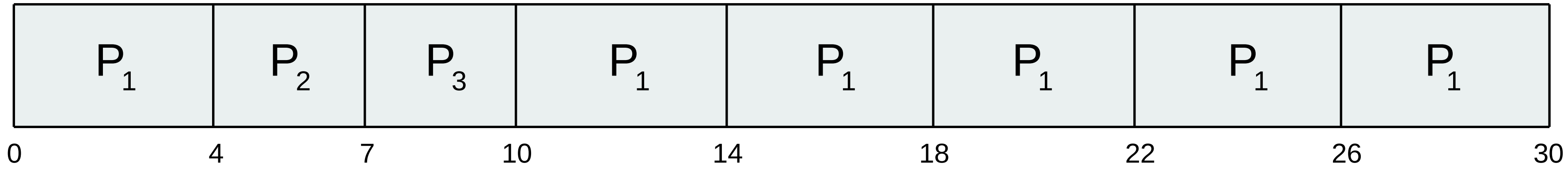
- Each process gets a small unit of CPU time (**time quantum** *q*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high

- Some important characteristics of the Round Robin(RR) Algorithm are as follows:
  - Round Robin Scheduling algorithm resides under the category of Preemptive Algorithms.
  - This algorithm is one of the oldest, easiest, and fairest algorithm.
  - This Algorithm is a real-time algorithm because it responds to the event within a specific time limit.
  - In this algorithm, the time slice should be the minimum that is assigned to a specific task that needs to be processed. Though it may vary for different operating systems.
  - This is a hybrid model and is clock-driven in nature.
  - This is a widely used scheduling method in the traditional operating system.

Example of RR with Time Quantum = 4

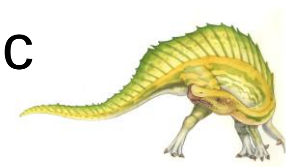
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

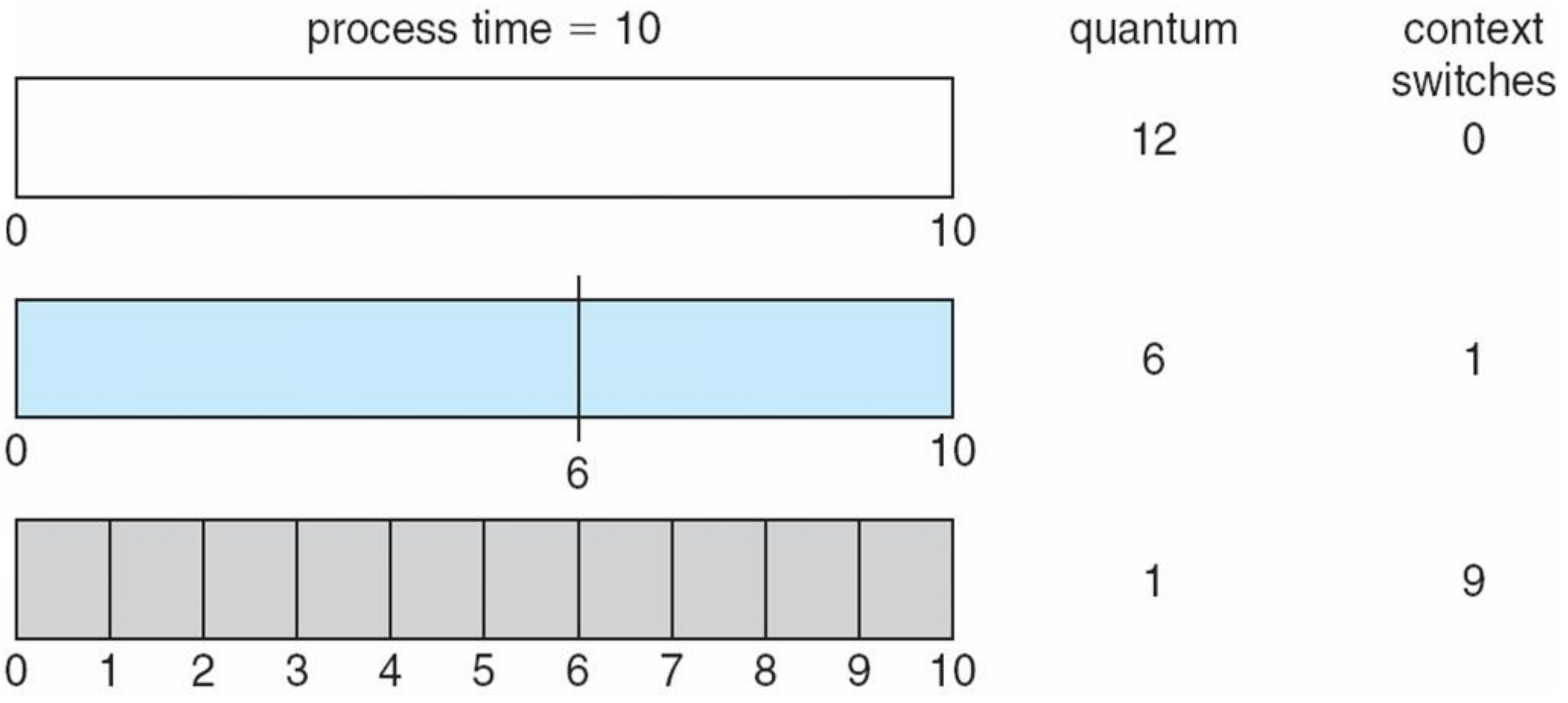


Typically, higher average turnaround than SJF, but better ***response***

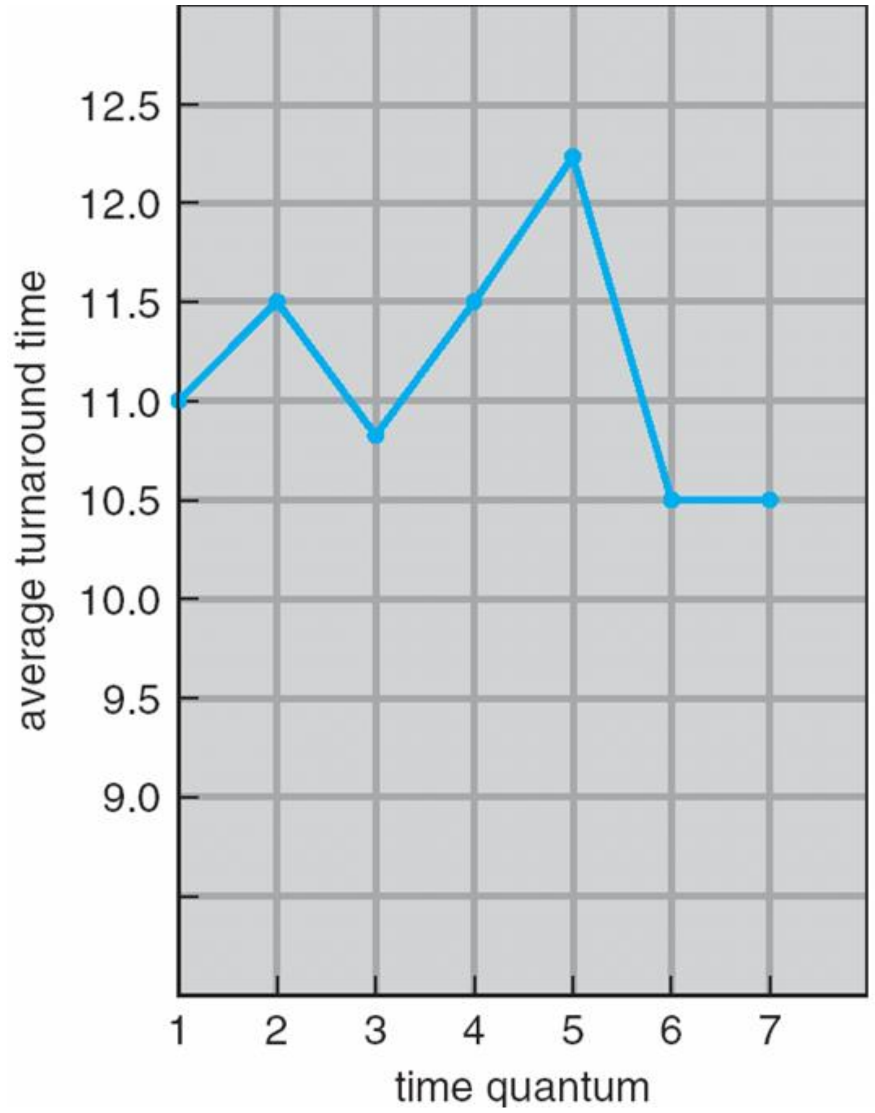
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec



Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

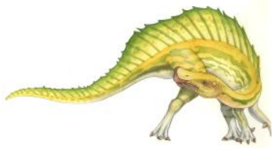
Time quantum : 2 ms

Process	Arrival time	Burst time
P1	0 ms	10 ms
P2	0 ms	5 ms
P3	0 ms	8 ms

Gantt Chart

P1	P2	P3	P1	P2	P3
1 2	2 4	4 6	6 8	8 10	10 12

P1	P2	P3	P1	P3	P1
12 14	14 15	15 17	17 19	19 21	21 23

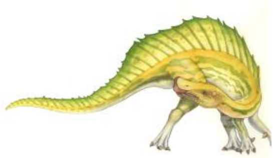


Process	Waiting Time	Turnaround Time
P1	13ms	23ms
P2	10ms	15ms
P3	13ms	21ms

```
Total waiting time: (13 + 10 + 13)= 36ms
Average waiting time: (36/3) = 12ms
Total turnaround time:(23 + 15 + 21)= 59ms
Average turnaround time: (59/3) = 19.66ms
```

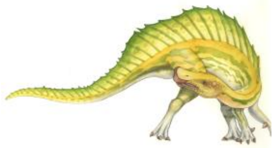
Advantages of round-robin:

- In terms of average response time, this algorithm gives the best performance.
- All the jobs get a fair allocation of CPU.
- There are no issues of starvation or convoy effect.
- Deals with all processes without any priority.
- Cyclic in nature.
- Newly created process is added to the end of the ready queue.
- A round-robin scheduler generally employs time-sharing which means providing each job a time slot or quantum.
- Each process gets a chance to reschedule after a particular quantum time.



Disadvantages of Round Robin Scheduling Algorithm

- This algorithm spends more time on context switches.
- For small quantum, it is time-consuming scheduling.
- This algorithm offers a larger waiting time and response time.
- In this, there is low throughput.
- If time quantum is less for scheduling then its Gantt chart seems to be too big.



Some points to remember

- With the decreasing value of time quantum
  - The number of context switches increases.
  - The Response Time decreases
  - Chances of starvation decrease in this case.
- With the increasing value of time quantum
  - The number of context switch decreases
  - The Response Time increases
  - Chances of starvation increases in this case.

Some points to remember

- If the value of time quantum is increasing then Round Robin Scheduling tends to become FCFS Scheduling.
- In this case, when the value of time quantum tends to infinity then the Round Robin Scheduling becomes FCFS Scheduling.
- Thus the performance of Round Robin scheduling mainly depends on the value of the time quantum.
- And the value of the time quantum should be such that it is neither too big nor too small.

Summary -FCFS

- FCFS_algorithm doesn't include any complex logic, it just puts the process requests in a queue and executes it one by one.
- Hence, FCFS is pretty simple and easy to implement.
- Eventually, every process will get a chance to run, so starvation doesn't occur.
- **Disadvantages:**
- There is no option for pre-emption of a process. If a process is started, then CPU executes the process until it ends.
- Because there is no pre-emption, if a process executes for a long time, the processes in the back of the queue will have to wait for a long time before they get a chance to be executed.

Summary SJF

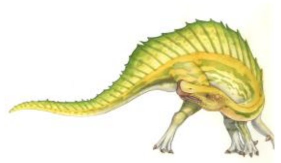
- According to the definition, short processes are executed first and then followed by longer processes.
- The throughput is increased because more processes can be executed in less amount of time.
- **Disadvantages:**
- The time taken by a process must be known by the CPU beforehand, which is not possible.
- Longer processes will have more waiting time, eventually they'll suffer starvation

Summary – Round Robin

- Each process is served by the CPU for a fixed time quantum, so all processes are given the same priority.
- Starvation doesn't occur because for each round robin cycle, every process is given a fixed time to execute. No process is left behind.
- **Disadvantages:**
- The throughput in RR largely depends on the choice of the length of the time quantum. If time quantum is longer than needed, it tends to exhibit the same behavior as FCFS.
- If time quantum is shorter than needed, the number of times that CPU switches from one process to another process, increases. This leads to decrease in CPU efficiency.

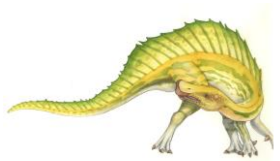
Summary –priority based

- The priority of a process can be selected based on memory requirement, time requirement or user preference.
  - For example, a high end game will have better graphics, that means the process which updates the screen in a game will have higher priority so as to achieve better graphics performance.
- **Disadvantages:**
- A second scheduling algorithm is required to schedule the processes which have same priority.
- In preemptive priority scheduling, a higher priority process can execute ahead of an already executing lower priority process. If lower priority process keeps waiting for higher priority processes, starvation occurs.



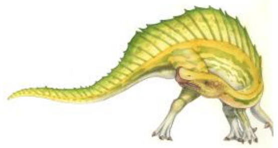
Usage of Scheduling Algorithms in Different Situations

- Situation 1:
- The incoming processes are short and there is no need for the processes to execute in a specific order.
- In this case, FCFS works best when compared to SJF and RR because the processes are short which means that no process will wait for a longer time.
- When each process is executed one by one, every process will be executed eventually.



Usage of Scheduling Algorithms in Different Situations

- Situation 2:
- The processes are a mix of long and short processes and the task will only be completed if all the processes are executed successfully in a given time.
- Round Robin scheduling works efficiently here because it does not cause starvation and also gives equal time quantum for each process.



Usage of Scheduling Algorithms in Different Situations

- Situation 3:
- The processes are a mix of user based and kernel based processes.
- Priority based scheduling works efficiently in this case because generally kernel based processes have higher priority when compared to user based processes.
- For example, the scheduler itself is a kernel based process, it should run first so that it can schedule other processes.