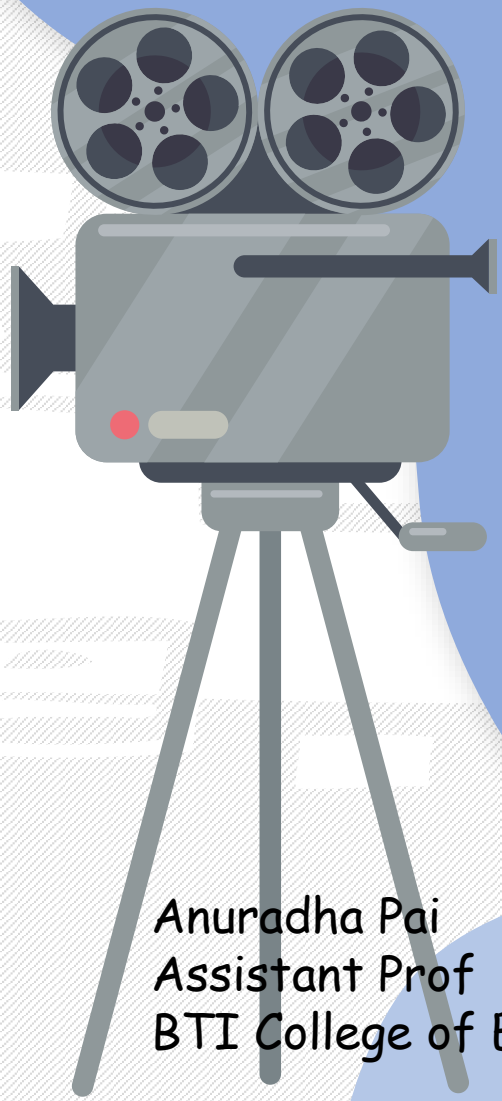
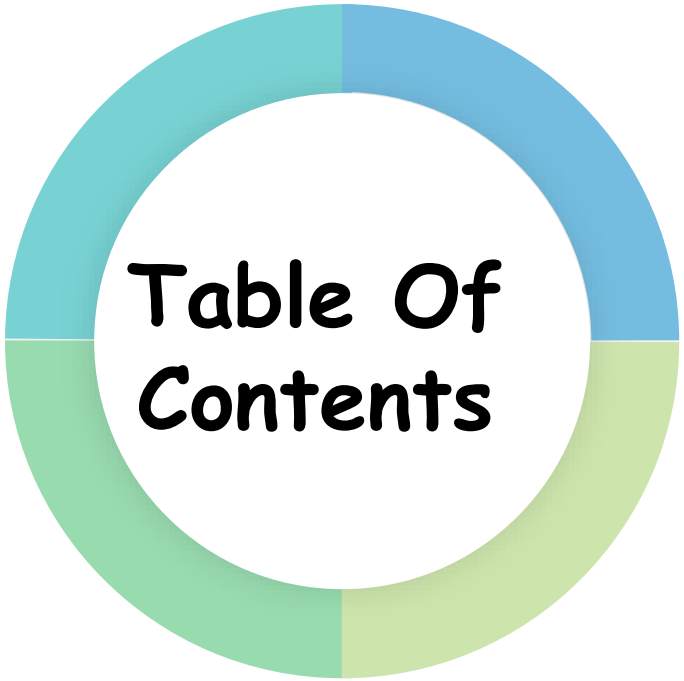
# OPERATING SYSTEMS

## Module-2 Part3: Process Scheduling



Anuradha Pai
Assistant Prof
BTI College of Engineering



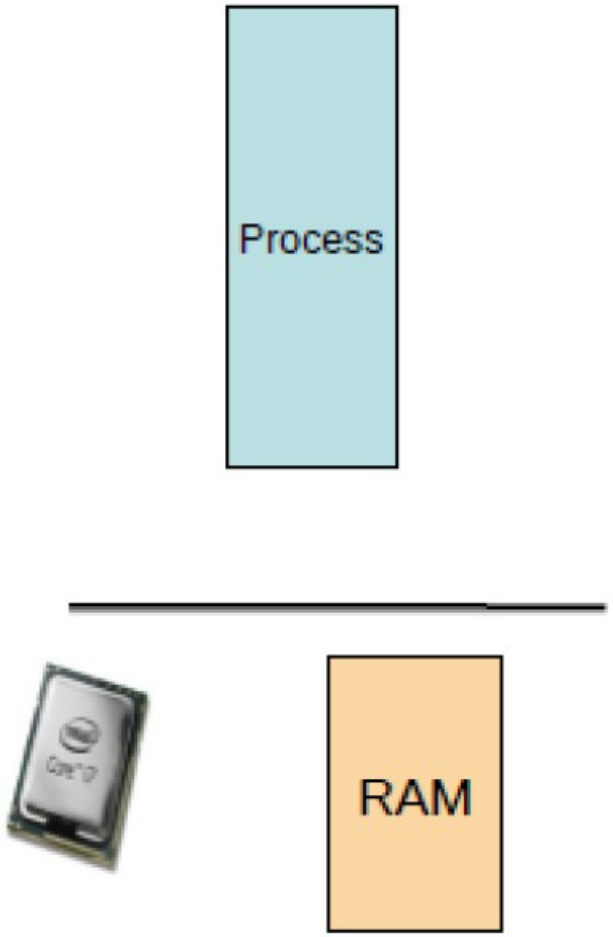

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling

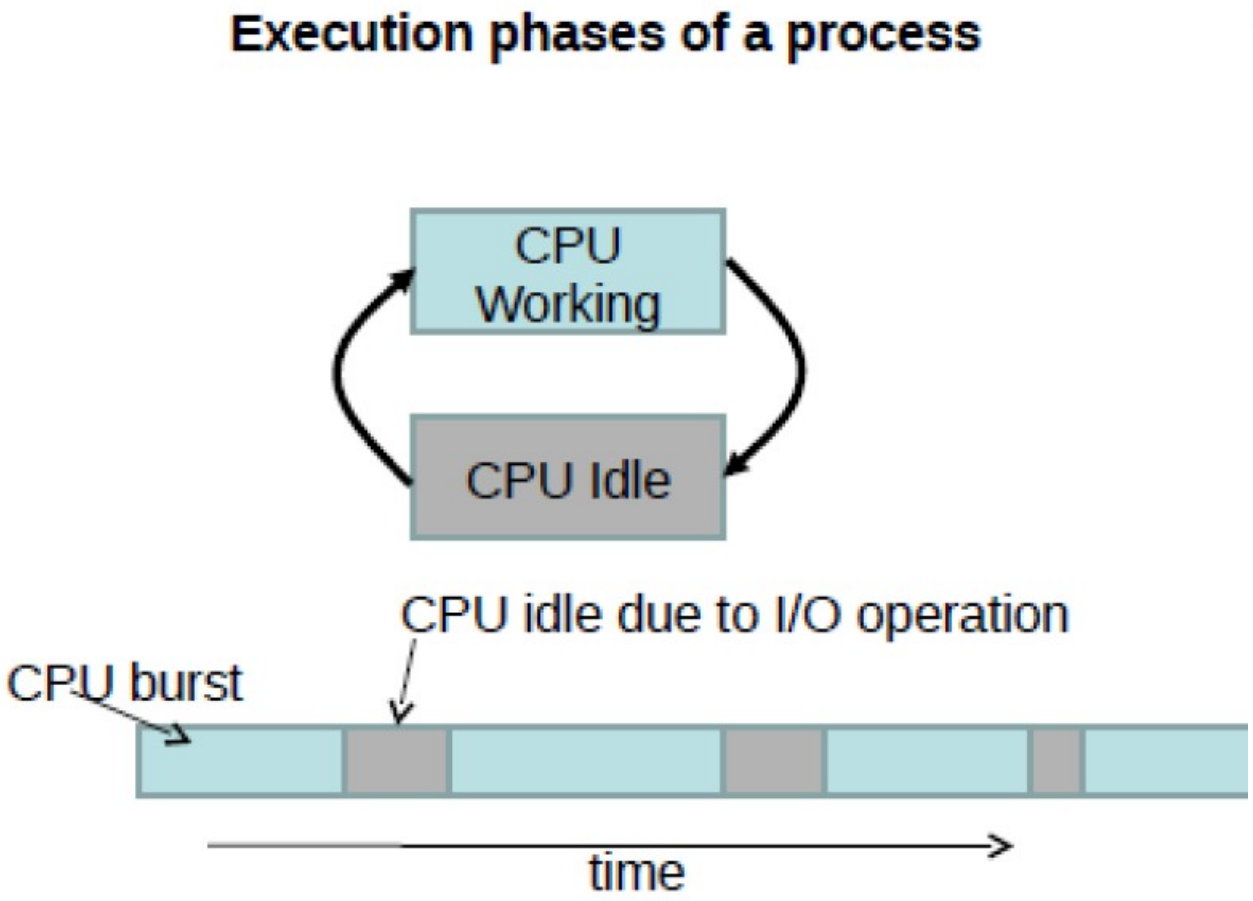
# 1. Basic Concepts

The idea of multi-programming

- Keep several processes in memory
- Every time one process has to wait, another process takes over the use of the CPU

# 1.1 Execution Phase of Process



# 1.2 Types of Processes

Process execution consists of a cycle of CPU execution (CPU burst) and I/O wait (I/O burst)

- I/O bound
  - Has small bursts of CPU activity and then waits for I/O
  - eg. Word processor
  - Affects user interaction (we want these processes to have highest priority)


- CPU bound
  - Hardly any I/O, mostly CPU activity (eg. gcc, scientific modeling, 3D rendering, etc)
    - Useful to have long CPU bursts
  - Could do with lower priorities

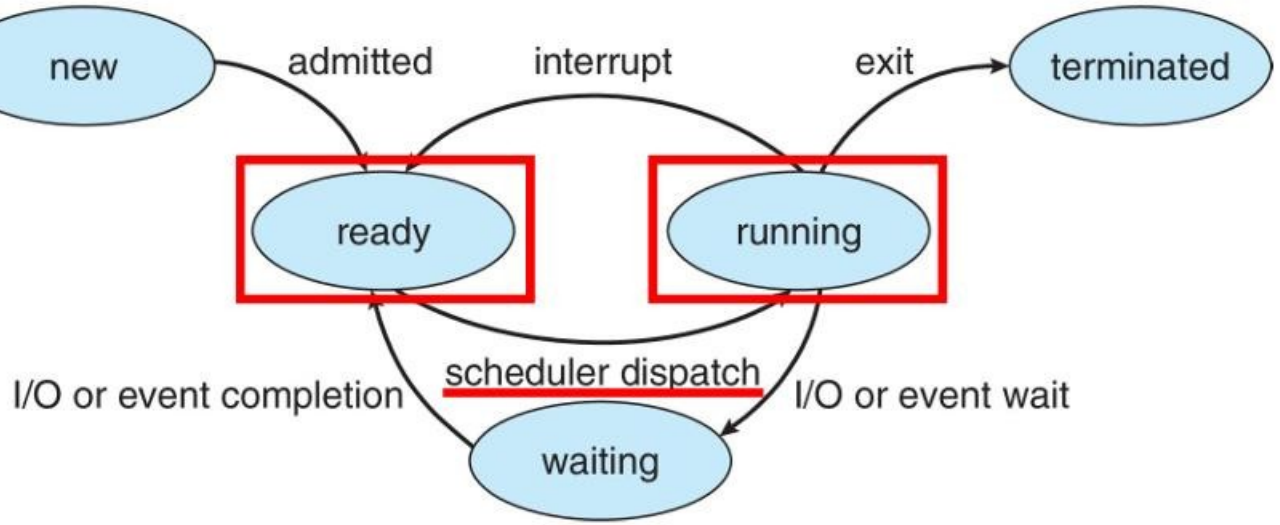
# 1.3 CPU Scheduler

- Whenever the CPU becomes idle, the operating system must select one of th processes from the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.
- A **ready queue** can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list. All the processes in the ready queue are lined up waiting for a chance to run on the CPU. The records in the queues are generally process control blocks (PCBs) of the processes.

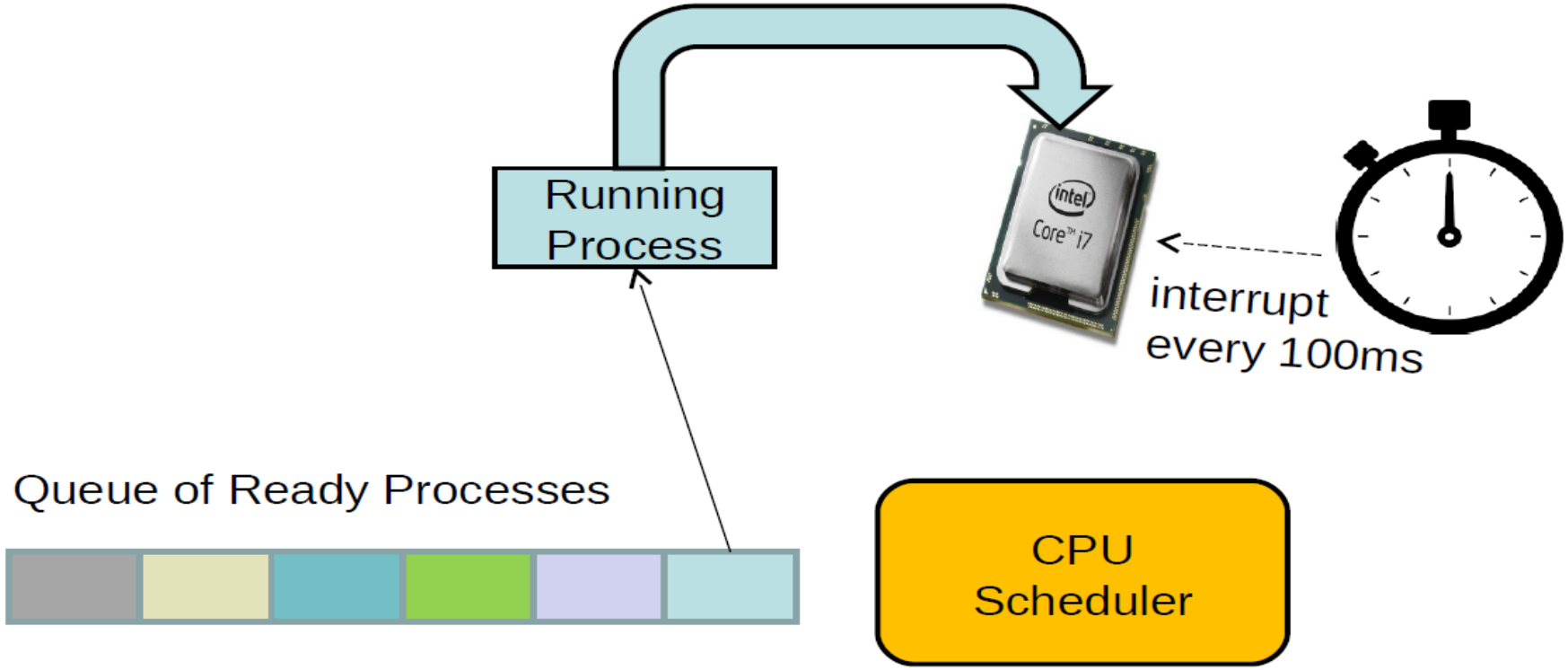
# 1.3 CPU Scheduler

Selects process from ready queue to execute

- Allocate a CPU for the selected process



# 1.3 CPU Scheduler




Scheduler triggered to run when timer interrupt occurs or when running process is blocked on I/O

Scheduler picks another process from the ready queue

Performs a context switch

# 1.3.1 Preemptive vs Non-preemptive Scheduling

CPU-scheduling decisions may take place under the following four circumstances:

- 1. When a process switches from the **running state to the waiting state** (for example, as the result of an I/O request or an invocation of wait for the termination of one of the child processes)
- 2. When a process switches from the **running state to the ready state** (for example, when an interrupt occurs)
- 3 When a process switches from the **waiting state to the ready state** (for example, at completion of I/O)
- 4 When a process **terminates**

When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is non preemptive or cooperative; otherwise, it is preemptive

# 1.3.1 Preemptive vs Non-preemptive Scheduling

- **Non - Preemptive Scheduling** once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
- **Preemptive Scheduling** The process under execution, may be released from the CPU, in the middle of execution due to some inconsistent state of the process.

# 1.3.1 Preemptive Scheduling Issues

Inconsistent state of shared data

- Require process synchronization
- Incur a cost associated with access to the shared data

Example

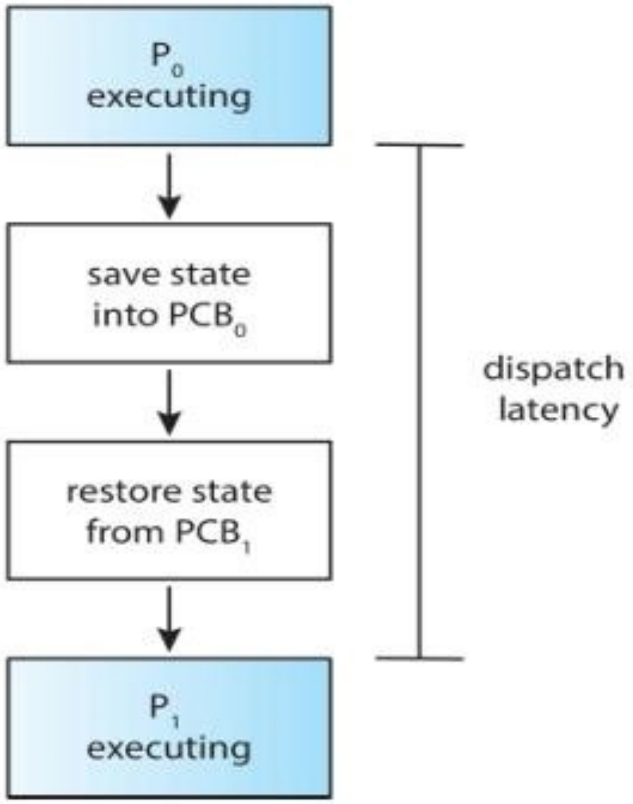
- Two processes share data
- While one process is updating the data, it is preempted so the second process can run
- The second process then tries to read the data
- Inconsistent state happens!

# 1.4 Dispatcher

Another component involved in the CPU-scheduling function is the dispatcher. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves the following:

- Switching context
- Jumping to the proper location in the in the selected program

The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency** (Context Switch Time).



# 2. Scheduling Criteria

Different CPU-scheduling algorithms

- Have different properties and
- May favour one class of processes over another.

Criteria to compare CPU-scheduling algorithms:

**1) CPU Utilization**

- We must keep the CPU as busy as possible.
- In a real system, it ranges from 40% to 90%.

# 2. Scheduling Criteria

**2) Throughput**

- Number of processes completed per time unit.
- For long processes, throughput may be 1 process per hour;
- For short transactions, throughput might be 10 processes per second.

**3) Turnaround Time**

- The interval from the time of submission of a process to the time of completion.
- Turnaround time is the sum of the periods spent
  - waiting to get into memory
  - waiting in the ready-queue
  - executing on the CPU and
  - doing I/O.

# 2. Scheduling Criteria

**4) Waiting Time**

- The amount of time that a process spends waiting in the ready-queue.

**5) Response Time**

- The time from the submission of a request until the first response is produced.
- The time is generally limited by the speed of the output device.

# 2. Scheduling Criteria

- Decides which process should run next.
- Aims,
  - Minimize waiting time
    - Process should not wait long in the ready queue
  - Maximize CPU utilization
    - CPU should not be idle
  - Maximize throughput
    - Complete as many processes as possible per unit time
  - Minimize response time
    - CPU should respond immediately
  - Fairness
    - Give each process a fair share of CPU

# 3. Scheduling Algorithms

- CPU scheduling deals with the problem of deciding which of the processes in the ready-queue is to be allocated the CPU.
- Following are some scheduling algorithms:

1. First Come First Served(FCFS) Scheduling
2. Round Robin Scheduling
3. Shortest Job First(SJF)  Scheduling
4. Priority scheduling
5. Multilevel Queue scheduling and
6. Multilevel Feedback Queue scheduling

# 3. Time Factors associated with a Process

- **Arrival Time(AT):** Time at which the process arrives in the ready queue.
- **Completion Time(CT):** Time at which process completes its execution.
- **Burst Time(BT):** Time required by a process for CPU execution.
- **Turn Around Time(TAT):** Time Difference between completion time and arrival time.

  Turn Around Time = Completion Time - Arrival Time.
- **Waiting Time (WT):** Time Difference between turnaround time and burst time.

  Waiting Time = Turn Around Time - Burst Time.
- **Response Time:** The time from the submission of a process execution request (Ready Queue) until the first response is produced.

# 3.1 First Come First Serve(FCFS)

- First job that requests the CPU gets the CPU
- **Non preemptive**
  - Process continues till the burst cycle ends

Procedure:

1) When a process enters the ready-queue, its PCB is linked onto the tail of the queue.

2) When the CPU is free, the CPU is allocated to the process at the queue‘s head.

3) The running process is then removed from the ready queue.

# 3.1.1 FCFS Example1

**Q1:** Consider the following set of processes arriving in the order: P1,P2,P3,P4, with length of CPU burst time given in milliseconds

- What is the waiting time for each process?
- Calculate the average waiting time and turn around time by drawing the Gantt Chart using FCFS

| Process | Burst Time |
|---|---|
| P1 | 7 |
| P2 | 4 |
| P3 | 2 |
| P4 | 5 |

# 3.1.1 FCFS Example1

Ans:


Waiting Time for

P1 = 0ms

P2 = 7ms

P3 = 11ms

P4 = 13ms

Average Turn around time

= (7+11+13+18)/4 = 12.25ms

Average waiting time

= (0+7+11+13)/4 = 7.75ms

Average Response Time is same as average wait time

# 3.1.2 FCFS Example2

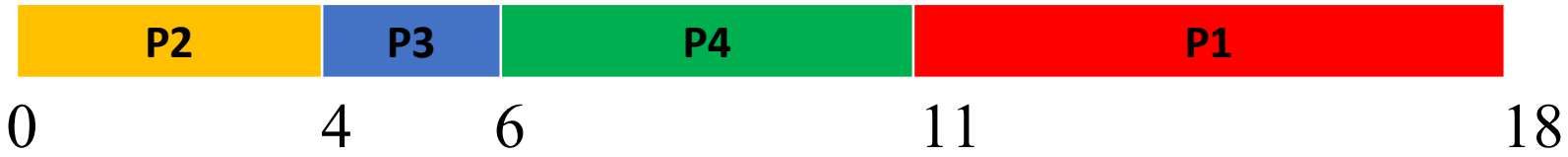
**Q2:** Consider the following set of processes arriving in the order: P2, P3,P4,P1 with length of CPU burst time given in milliseconds

- What is the waiting time for each process?
- Calculate the average waiting time and turn around time by drawing the Gantt Chart using FCFS

| Process | Burst Time |
|---|---|
| P1 | 7 |
| P2 | 4 |
| P3 | 2 |
| P4 | 5 |

# 3.1.2 FCFS Example2

Ans:




Waiting Time for

P1 = 11ms

P2 = 0ms

P3 = 4ms

P4 = 6ms

Average Turn around time

= (18+4+6+11)/4 = 9.75ms

Average waiting time

= (11+0+4+6)/4 = 5.25ms

Average Response Time is same as average wait time

# 3.1.3 FCFS Pros and Cons

- **Advantages**
  - Code is simple to write & understand.
- **Disadvantages**
  - Waiting time depends on arrival order
  - **Convoy effect**: Short processes are stuck waiting for long process to complete
  - Not suitable for time sharing systems
  - Non Preemptive

# 3.1.4 FCFS Example3(HW)

**Q3:** Calculate average waiting and turnaround times by drawing the Gantt chart using FCFS

| Processes | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 9 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |

# 3.1.4 FCFS Example3

**Ans:**


Waiting Time for

P1 = 0ms

P2 = 8ms

P3 = 11ms

P4 = 19ms

Average Turn around time

= ((9-0)+(13-1)+(22-2)+(27-3))/4 = 16.25ms

Average waiting time

= ((9-9)+(12-4)+(20-9)+(24-5))/4 = 9.5ms

**Average Response Time?**

# 3.2 SJF Scheduling

- The CPU is assigned to the process that has the smallest next CPU burst.
- If two processes have the same length CPU burst, FCFS scheduling is used to break the tie.
- For long-term scheduling in a batch system, we can use the process time limit specified by the user, as the length(CPU burst)
- SJF can't be implemented at the level of short-term scheduling, because there is no way to know the length of the next CPU burst.

# 3.2 SJF Scheduling

SJF algorithm may be either

1. **Non preemptive SJF**

The current process is allowed to finish its CPU burst.

2. **Preemptive SJF(SRTF)**

If the new process has a shorter next CPU burst than what is left of the executing process, that process is preempted.

It is also known as **Shortest-Remaining-Time-First** scheduling (SRTF).

**Advantage:**

Minimizes average wait time

and average response time

**Disadvantage:**

- Not practical : difficult to predict burst time
    - Learning to predict future
- May starve long jobs

# 3.2.1 SJF(without preemption) Example1

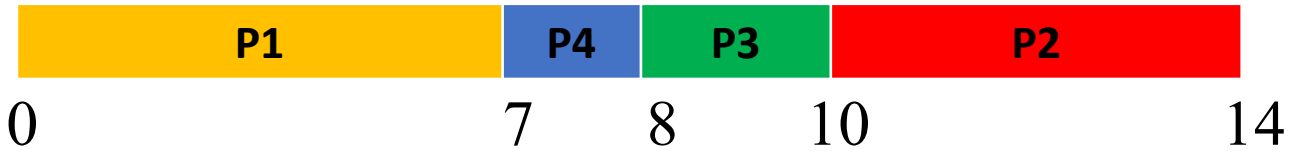
**Q4:** Consider the following set of processes with length of CPU burst time given in milliseconds

- What is the waiting time for each process?
- Calculate the average waiting time and turn around time by drawing the Gantt Chart using SJF

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 2 |
| P4 | 7 | 1 |

# 3.2.1 SJF(without preemption) Example1

Ans:




Waiting Time (=CT – AT – BT)

P1 = 0ms

P2 = (14-2-4)ms = 8ms

P3 = (10-4-2)ms = 4ms

P4 = 0ms

Average Turn around time

= (7+12+6+1)/4 = 6.5ms

Average waiting time

= (0+8+4+0)/4 = 3ms

# 3.2.2 SJF(without pre-emption) Example

**Q5:** Processes arrived in P1,P2,P3,P4, P5 order at time 0

| Process | burst time | priority |
|---|---|---|
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

- Draw Gantt Chart to show execution using SJF. Calculate turn around time for each of the scheduling algorithm.
- What is the waiting time of each process?

**Q6:** Consider the set of processes wi Arrival Time, CPU burst time shown below. Write Gantt chart and solve the average waiting time and average turn around time using SJF scheduling

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 10 |
| P2 | 1 | 1 |
| P3 | 2 | 2 |
| P4 | 3 | 1 |
| P5 | 4 | 5 |

# 3.2.3 SJF Example with ready queue

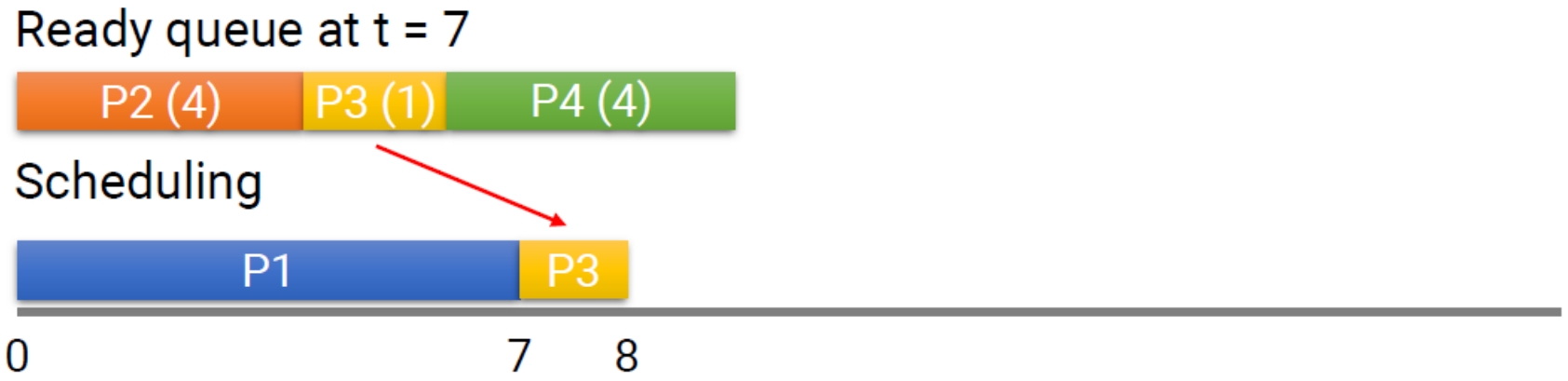
SJF Example showing the ready queue during different time interval.

# Non-Preemptive SJF Example

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

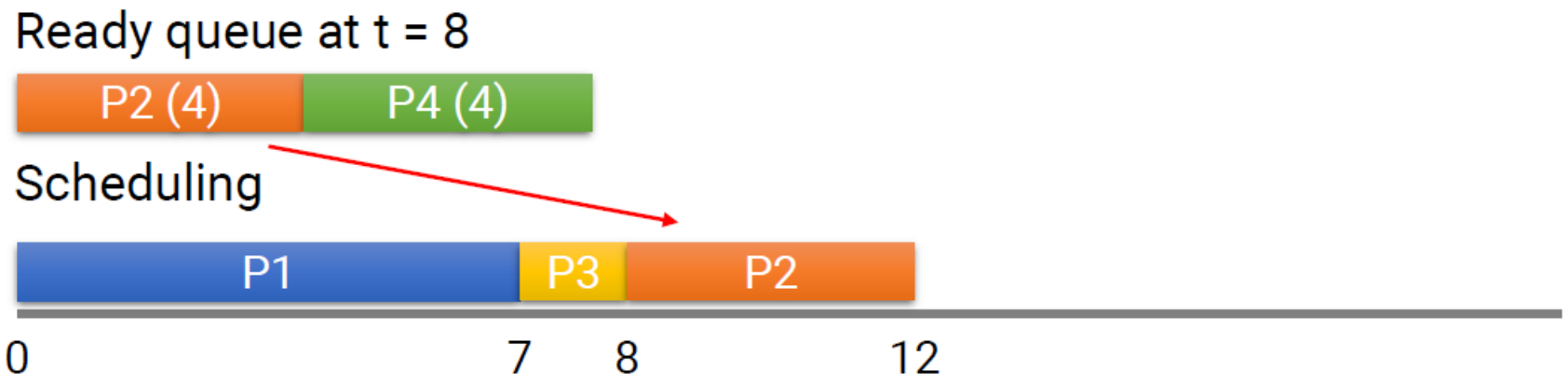
# Non-Preemptive SJF Example (cont.)

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |



# Non-Preemptive SJF Example (cont.)

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

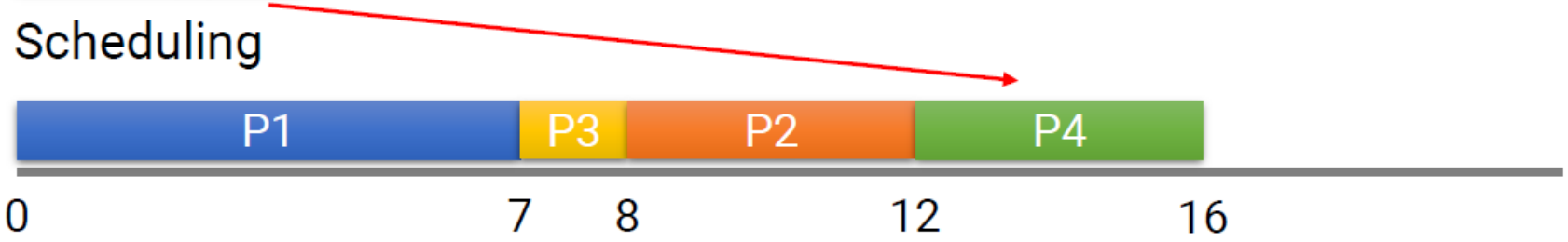


# Non-Preemptive SJF Example (cont.)

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

Ready queue at t = 12


Scheduling




- **Wait time = completion time - arrival time - run time**
- **AWT** = [(7-0-7) + (12-2-4) + (8-4-1) + (16-5-4)] / 4 = 4
- **Response time**: P1 = 0, P2 = 6, P3 = 3, P4 = 7

# 3.2.4 SRTF(preemptive SJF) Scheduling

- If a new process arrives with a shorter burst time than *remaining of current process* then schedule new process
- Further reduces average waiting time and average response time
- Not practical

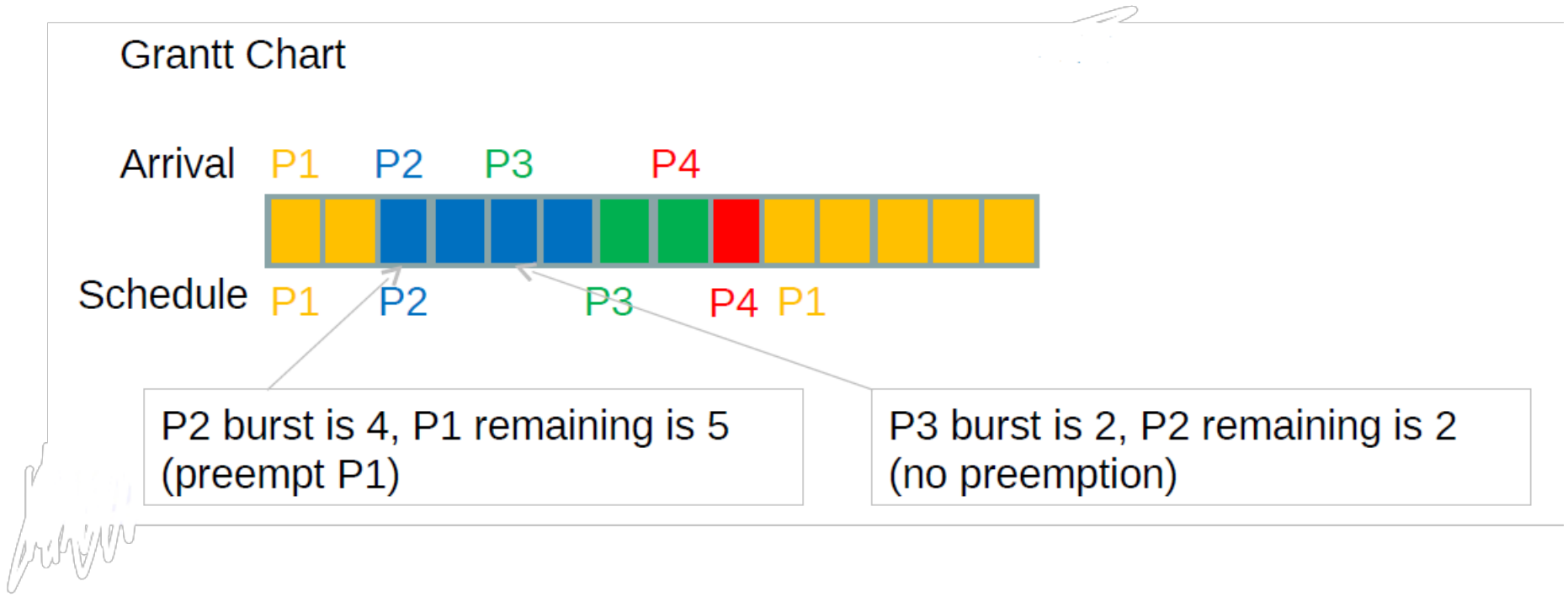
# 3.2.4 SRTF(preemptive SJF) Example

**Q7:** Consider the following set of processes with length of CPU burst time given in milliseconds

- What is the waiting time for each process?
- Calculate the average waiting time and turn around time by drawing the Gantt Chart using SRTF

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 2 |
| P4 | 7 | 1 |

# 3.2.4 SRTF Example



# 3.2.4 SRTF Example

Arrival P1 P2 P3 P4


Schedule 0 2 6 8 9 14

Waiting Time (=CT – AT – BT)

P1 = (14-0-7) = 7

P2 = (6-2-4) = 0

P3 = (8-4-2) = 2

P4 = (9-7-1) = 1

Average Turn around time

= (14+4+4+2)/4 = 6ms

Average waiting time

= (7+0+2+1)/4 = 2.5ms

# 3.2.5 SRTF Example with ready queue

SRTF Example showing the ready queue during different time interval.

# Preemptive SJF Example

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

# Preemptive SJF Example

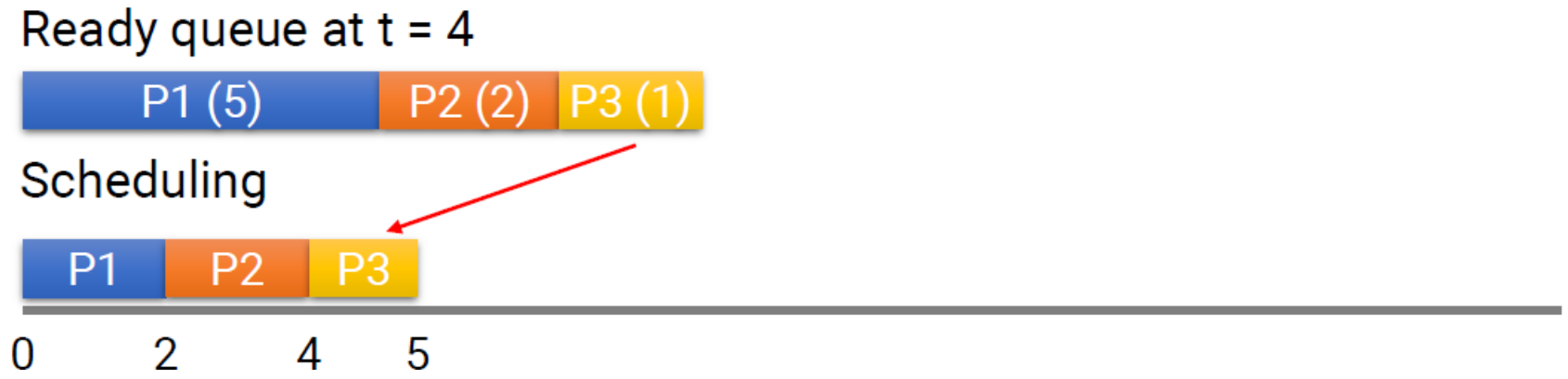
| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

Ready queue at t = 2


Scheduling

# Preemptive SJF Example

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |



# Preemptive SJF Example

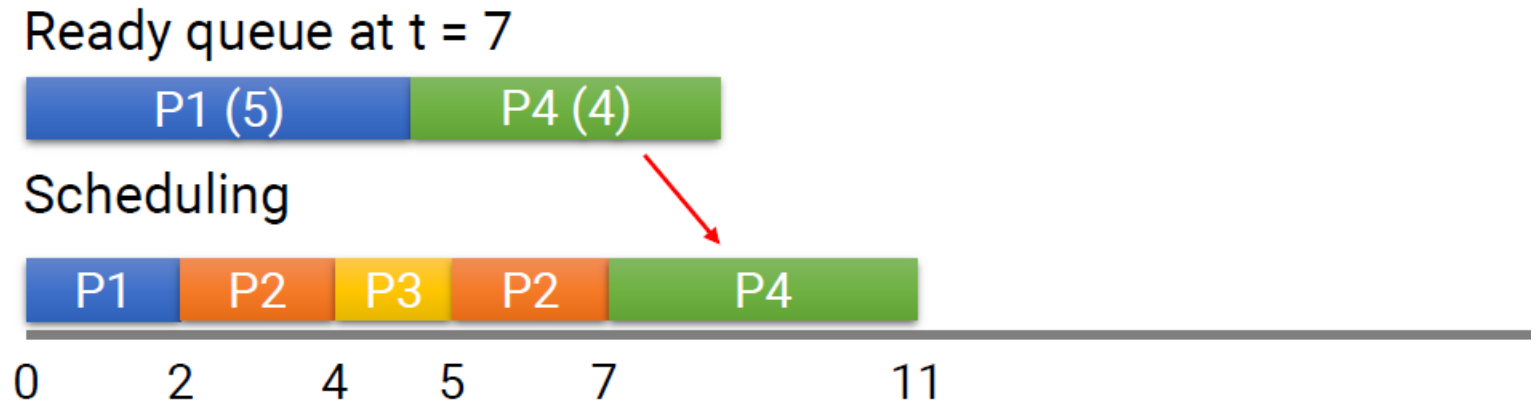
| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

Ready queue at t = 5


Scheduling

# Preemptive SJF Example

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

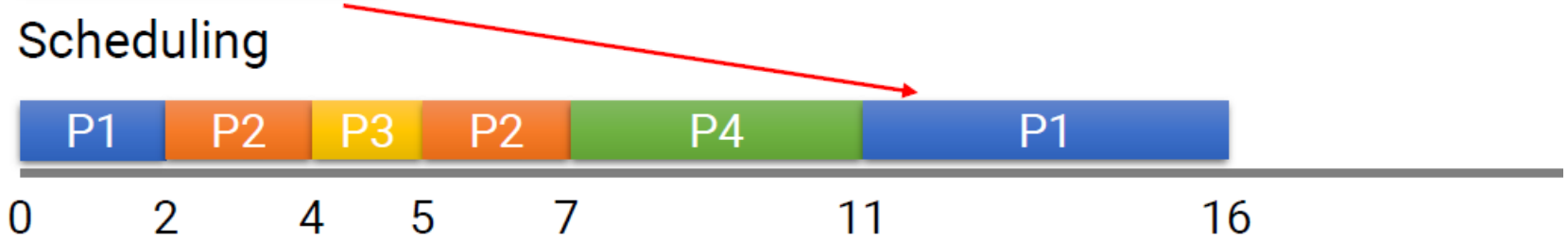


# Preemptive SJF Example

| Process | Arrival Time | Burst Time |
|---|---|---|
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

Ready queue at t = 11


Scheduling




- **Wait time = completion time - arrival time - run time**
- **AWT** = [(16-0-7) + (7-2-4) + (5-4-1) + (11-5-4)] / 4 = 3
- **Response time**: P1 = 0, P2 = 0, P3 = 0, P4 = 2

# 3.3 Priority Based Scheduling

- A priority is associated with each process.
- The CPU is allocated to the process with the highest priority.
- Equal-priority processes are scheduled in FCFS order.
- Priorities can be defined either internally or externally.
    1. Internally-defined priorities.
        - Use some measurable quantity to compute the priority of a process.
        - For example: time limits, memory requirements, no. of open files.
    2. Externally-defined priorities.
        - Set by criteria that are external to the OS
        - For example:
            - importance of the process
            - political factors

# 3.3 Priority Based Scheduling

Priority scheduling can be either preemptive or nonpreemptive.

**1. Preemptive**

The CPU is preempted if the priority of the newly arrived process is higher than the priority of the currently running process.

**2. Non Preemptive**

The new process is put at the head of the ready-queue

**Advantage:**

Higher priority processes can be executed first.

**Disadvantage:**

**Starvation:** Indefinite blocking, where low-priority processes are left waiting indefinitely for CPU.

Solution: **Aging** is a technique of increasing priority of processes that wait in system for a long time.

Example: Consider the following set of processes, assumed to have arrived at time 0, in the order PI, P2, ..., P5, with the length of the CPU-burst time given in milliseconds.

# 3.3.1 Priority based Scheduling Example

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

The Gantt chart for the schedule is as follows:

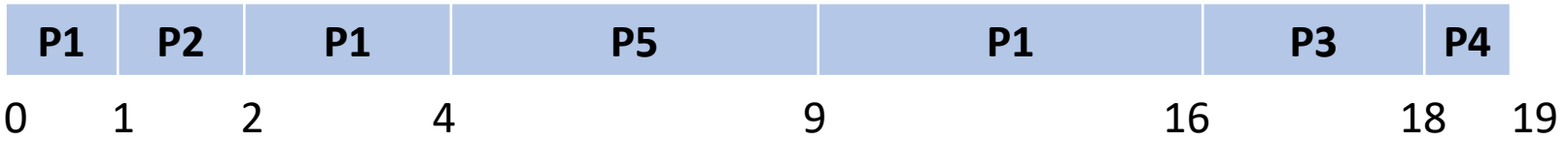

The average waiting time is 8.2 milliseconds.

# 3.3.2 Priority based Scheduling Example (Preemptive)

| Process | Arrival Time | Burst Time | Priority |
|---|---|---|---|
| P1 | 0 | 10 | 3 |
| P2 | 1 | 1 | 1 |
| P3 | 2 | 2 | 4 |
| P4 | 3 | 1 | 5 |
| P5 | 4 | 5 | 2 |

Average Turnaround time
= ((16-0)+(2-1)+(18-2)+(19-3)+(9-4))/5= 10.8ms

Average Waiting Time
= ((16-10)+(1-1)+(16-2)+(16-1)+(5-5))/5 = 7ms



# 3.4 Round Robin Scheduling

- Designed especially for timesharing systems.
- It is similar to FCFS scheduling, but with preemption.
- Each process gets a small unit of CPU time (**time quantum**, TQ), usually 10 ~ 100 ms
  - Context switch time usually < 10 microseconds
- After TQ elapsed, process is **pre-empted** and added to the end of the ready queue
- The ready-queue is implemented as a **FIFO circular queue.**
- CPU scheduler
  1. Picks the first process from the ready-queue.
  2. Sets a timer to interrupt after 1 time quantum and
  3. Dispatches the process.

# 3.4 Round Robin Scheduling

One of two things will then happen.

1. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily.
2. If the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the OS. The process will be put at the tail of the ready-queue.

# 3.4.1 RR Scheduling: Time Quantum & Context Switch Time

In software, we need to consider the effect of context switching on the performance of RR scheduling

1) Larger the time quantum for a specific process time, less time is spend on context switching.

2) The smaller the time quantum, more overhead is added for the purpose of context-switching.

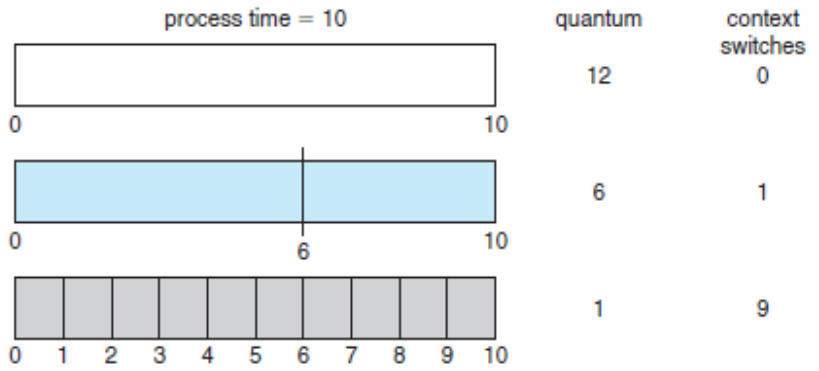
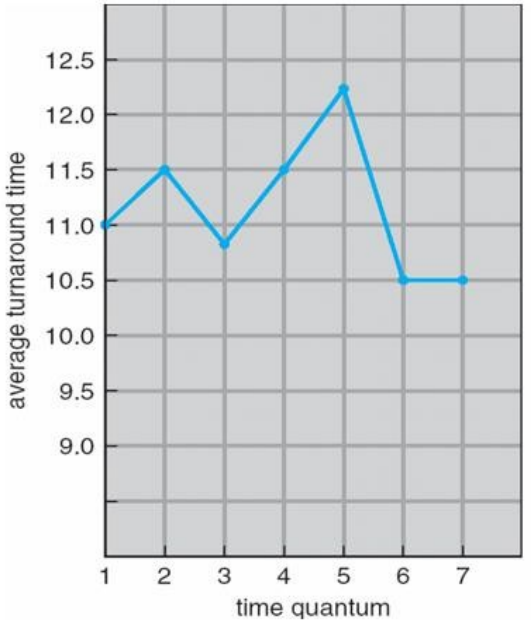



*Fig: How a smaller time quantum increases context switches*

# 3.4.2 RR Scheduling: Turn around time varies with the Time Quantum

- The performance of algorithm depends heavily on the size of the time quantum
  - If time quantum=very large, RR policy is the same as the FCFS policy.
  - If time quantum=very small, RR approach appears to the users as though each of n processes has its own processor running at l/n the speed of the real processor.
- The average turnaround time of a set of processes does not necessarily improve as the time-quantum size increases. In general, the average turnaround time can be improved if most processes finish their next CPU burst in a single 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

*Fig: How turn around time varies with the time quantum*

# 3.4.3 Round Robin(RR) Scheduling

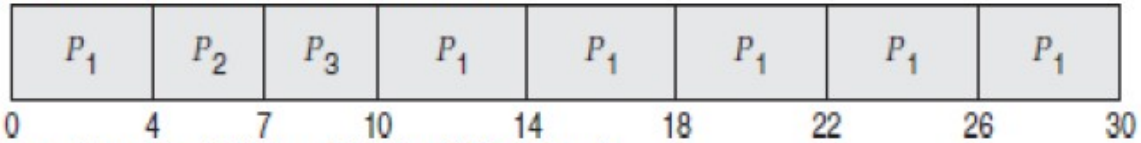
- Advantages
  - Fair (Each process gets a fair chance to run on the CPU)
  - Low average wait time, when burst times vary
  - Faster response time
- Disadvantages
  - Increased context switching
  - Context switches are overheads!!!
  - High average wait time, when burst times have equal lengths

# 3.4.4 RR Scheduling Example1

Calculate average waiting time and turnaround times by drawing the Gantt Chart using RR(TQ = 4ms)

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

The Gantt chart for the schedule is as follows:




The average waiting time is 17/3 = 5.66 milliseconds.

# 3.5 Multilevel Queue Scheduling

- Useful for situations in which processes are easily classified into different groups.
  - For example, a common division is made between
    - foreground (or interactive) processes and
    - background (or batch) processes.
- The ready-queue is partitioned into several separate queues. For example, separate queues might be used for foreground and background processes.
- The processes are permanently assigned to one queue based on some property like

  → memory size

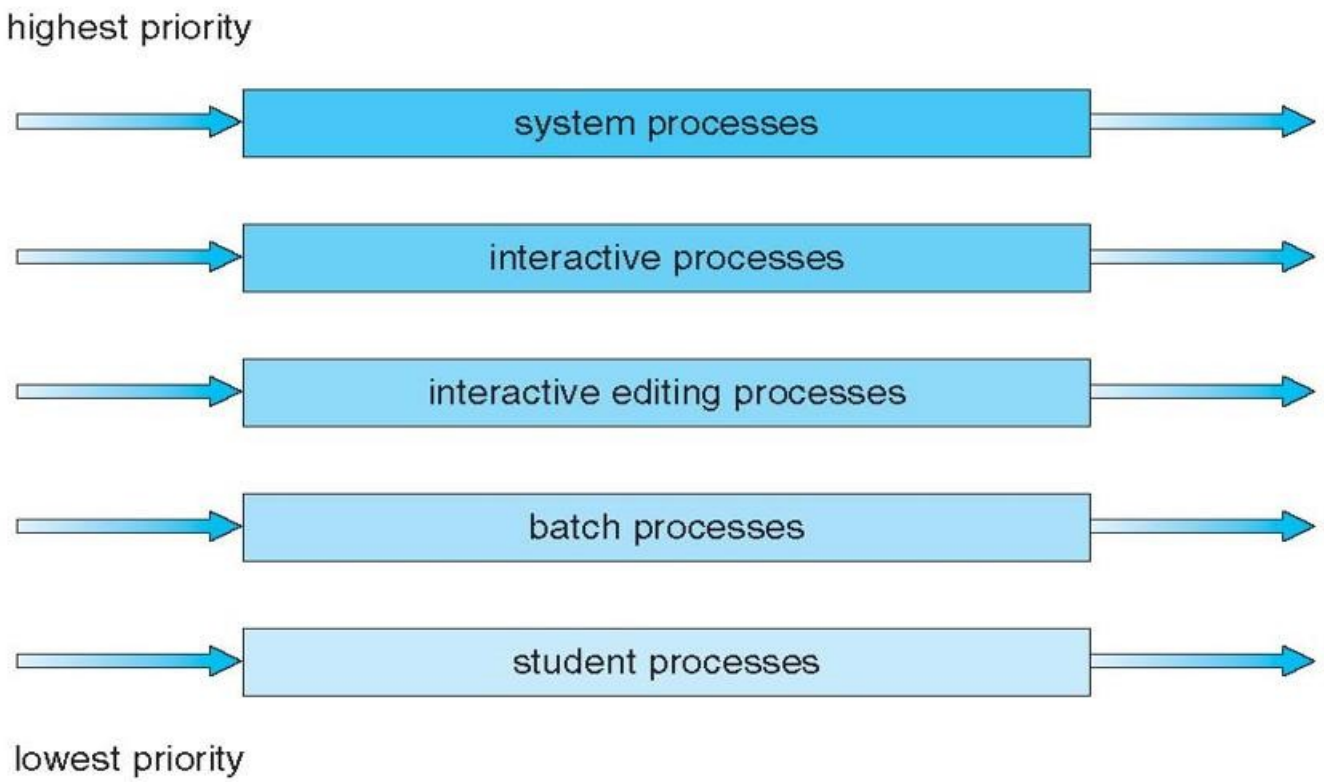
  → process priority or

  → process type.
- Each queue has its own scheduling algorithm.

# 3.5 Multilevel Queue Scheduling

- There must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling.
  - For example, the foreground queue may have absolute priority over the background queue.
- **Time slice**: each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR 20% to background in FCFS

# 3.5 Multilevel Queue Scheduling



# 3.6 Multilevel Feedback Queue Scheduling

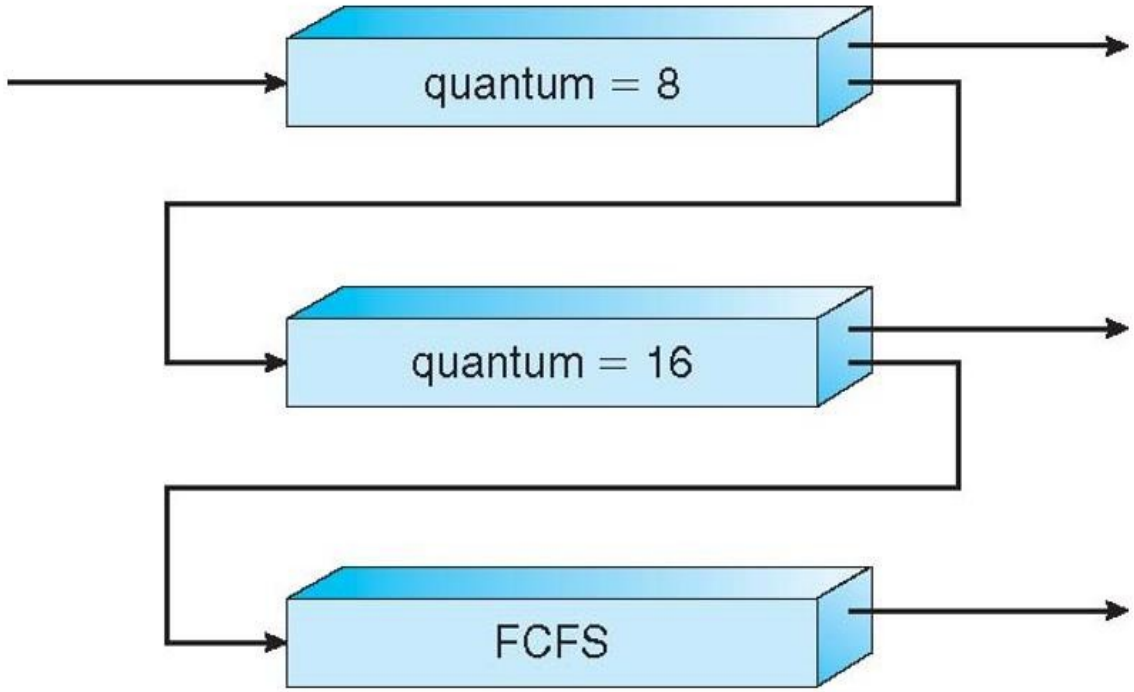
- A process may move between queues
- The basic idea:
  - Separate processes according to the features of their CPU bursts.
  - For example
    1. If a process uses too much CPU time, it will be moved to a lower-priority queue.
       - This scheme leaves I/O-bound and interactive processes in the higher-priority queues.
    2. If a process waits too long in a lower-priority queue, it may be moved to a higher priority queue
       - This form of aging prevents starvation.

# 3.6 Multilevel Feedback Queue Scheduling

In general, a multilevel feedback queue scheduler is defined by the following parameters:

1) The number of queues.
2) The scheduling algorithm for each queue.
3) The method used to determine when to upgrade a process to a higher priority queue.
4) The method used to determine when to demote a process to a lower priority queue.
5) The method used to determine which queue a process will enter when that process needs service.

# 3.6 Multilevel Feedback Queue Scheduling




- Consider a multilevel feedback queu scheduler with three queues, numbered from 0 to 2.
- The scheduler first executes all processes in queue 0. Only when queue 0 is empty will it execute processes in queue 1. Similarly, processes in queue 2 will only be executed if queues 0 and 1 are empty.
- A process that arrives for queue 1 will preempt a process in queue 2. A process in queue 1 will in turn be preempted by a process arriving for queue 0.

# CPU Scheduling Question1

consider following process with cpu burst time given in milliseconds.

| Process | burst time | priority |
|---|---|---|
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

Processes are arrived in P1,P2,P3,P4,P5 order of all at time 0.

1)Draw gantt charts to show execution using FCFS,SJF,non preemptive priority(smaller number implies higher priority.) and round robin(quantum = 1 ) scheduling.

2)Also calculate turnaround time for each scheduling algorithm.

3)What is the waiting time of each process for each one of the above scheduling algorithm.

# CPU Scheduling Question2

For the following set of processes find the average waiting time and average turnaround time using gantt chart for 1)SJF 2)Priority scheduling(Smaller number implies higher priority)

L6

| Process | burst time | priority |
|---|---|---|
| P1 | 5 | 5 |
| P2 | 3 | 4 |
| P3 | 8 | 3 |
| P4 | 2 | 1 |
| P5 | 1 | 2 |

# CPU Scheduling Question3

4 batch jobs P1,P2,P3,P4 arrive at computer centre at most the same order .The estimated running times are 10,8,2,4 ms.
The priorities are 3,4,1,2. Time quantum is 2 ms.
Draw gantt chart and compute average waiting time and average turnaround time to following algorithm

1)FCFS
2)SJF(non preemptive )
3)priority
4)Round robin

**Real leaders are ordinary people with extraordinary determination.**

**Good Luck☺**