

# UNIT-2

## CHAPTER-3

### PROCESS SCHEDULING

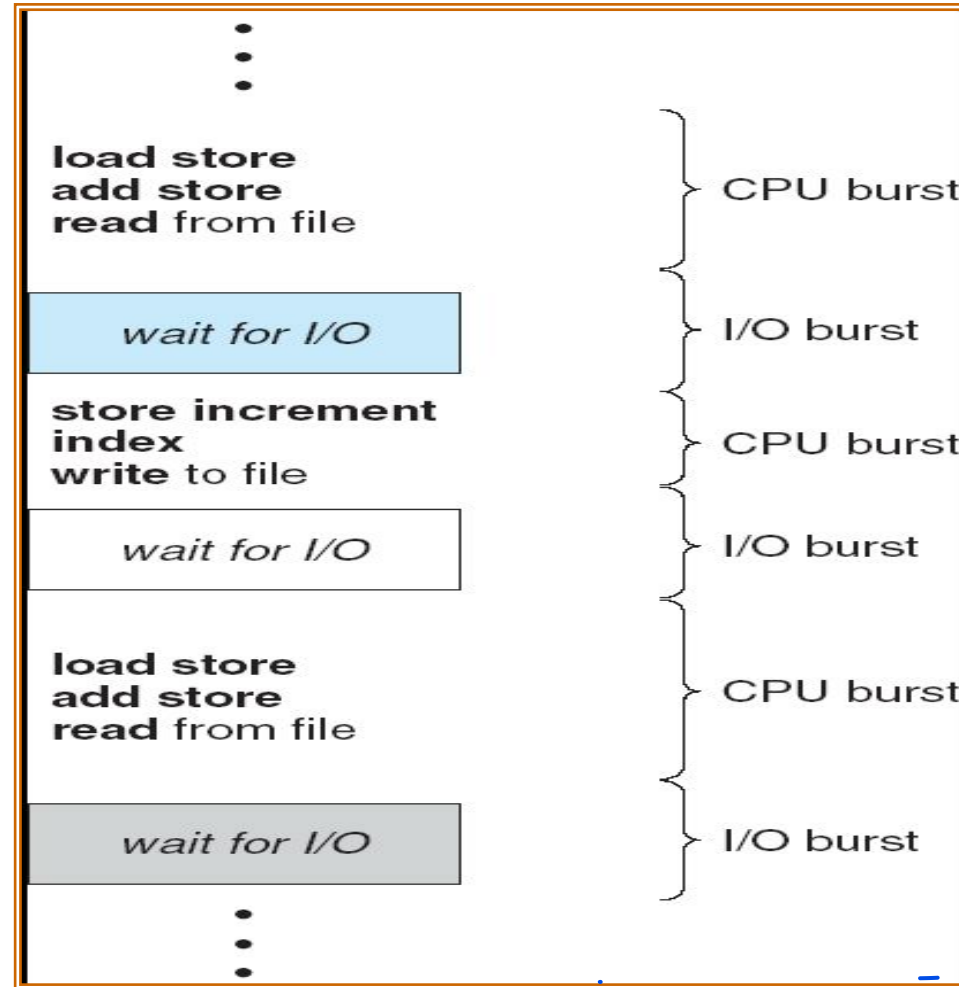
# Basic Concepts

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- With multiprogramming, Several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process.
- This pattern continues. Every time one process has to wait, another process can take over use of the CPU.
- Scheduling of this kind is a fundamental operating-system function.
- The CPU is, one of the primary computer resources. Thus, its scheduling is central to operating-system design.

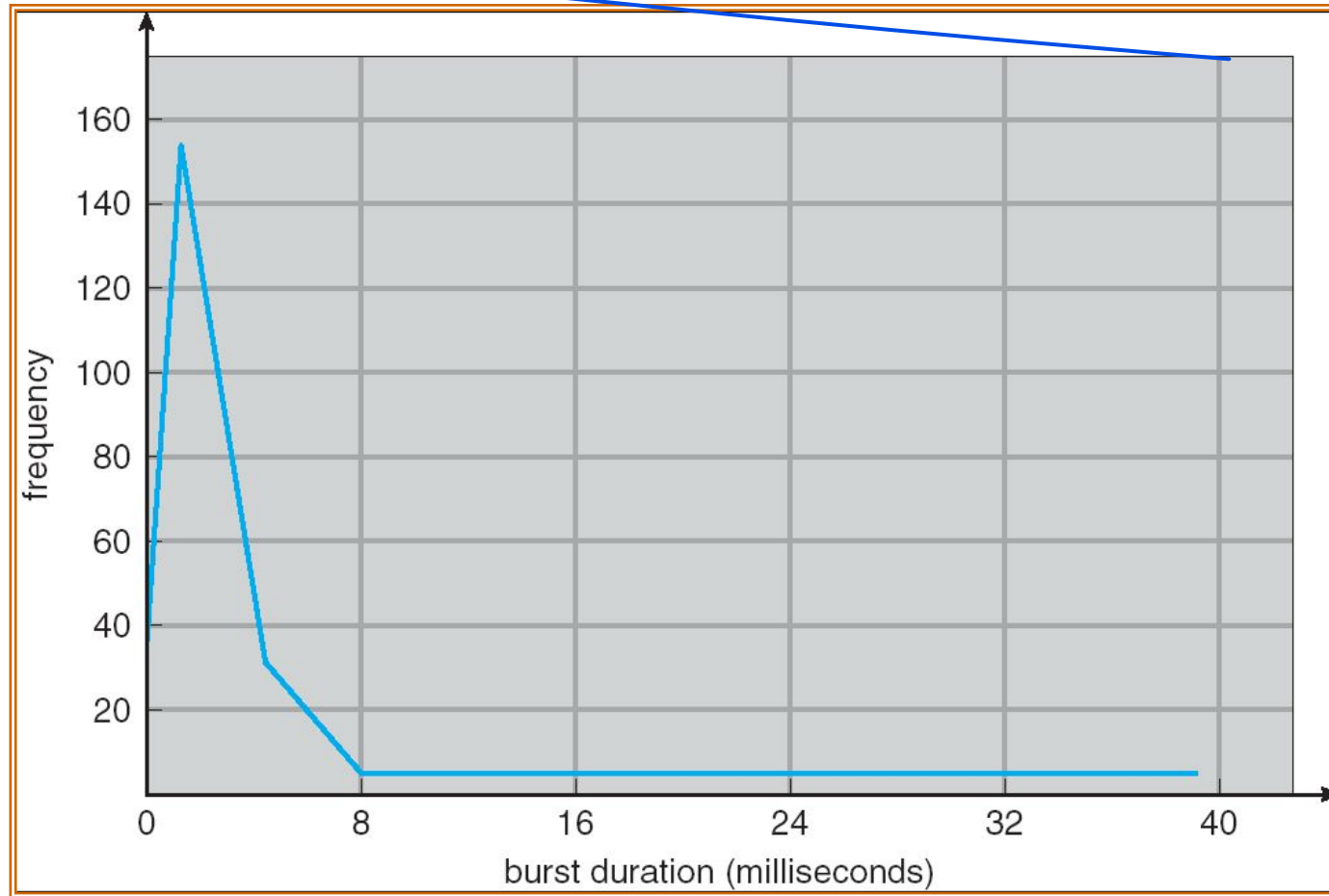
# CPU-I/O Burst Cycle

- Process execution consists of a cycle of CPU execution and I/O wait
- Processes alternate between these two states.
- Process execution begins with a CPU burst. That is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution as shown in the figure

# Alternating Sequence of CPU And I/O Bursts



# Histogram of CPU-burst Times



- The curve is generally characterized as exponential or hyperexponential, with a large number of short CPU bursts and a small number of long CPU bursts.
- An I/O-bound program typically has many short CPU bursts.
- A CPU-bound program might have a few long CPU bursts.
- This distribution can be important in the selection of an appropriate CPU scheduling algorithm.

# CPU Scheduler

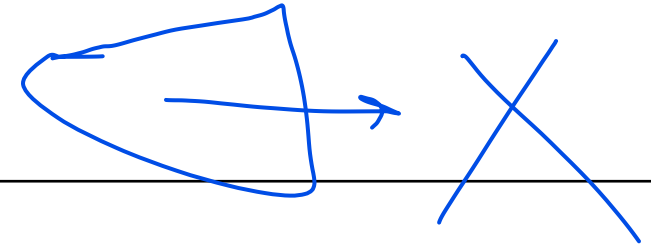
- Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed.
- The selection process is carried out by the short-term scheduler (or CPU scheduler).
- The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.
- The Ready queue can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list.
- 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.)

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

1. When a process switches from the running state to the waiting state.
2. When a process switches from the running state to the ready state.
3. When a process switches from the waiting state to the ready state.
4. When a process terminates.



# Two types of scheduling

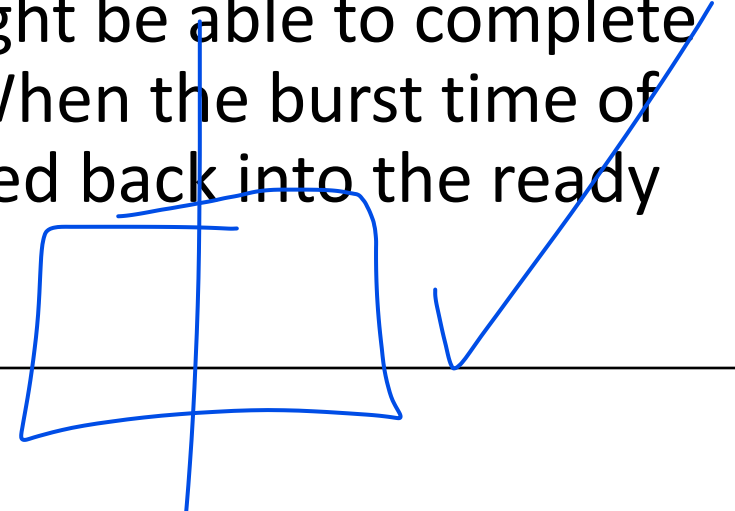


- **Nonpreemptive or cooperative**

Under nonpreemptive 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**

Is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance.



# 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.
- Switching to user mode.
- Jumping to the proper location in the user program to restart that program.
- The dispatcher should be as fast as possible, since it is invoked during every process switch.
- The time taken by the dispatcher to stop one process and start another running is known as the **dispatch latency**.

# Scheduling Criteria

- Different CPU scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another.
- Criteria suggested for comparing CPU scheduling algorithms:
  - **CPU utilization**-CPU should be kept as busy as possible.
  - **Throughput**-number of processes that are completed per time unit, called *throughput*.
  - **Turnaround time**-The interval from the time of submission of a process to the time of completion is the *turnaround time*.
  - **Waiting time**-*Waiting time* is the sum of the periods spent waiting in the ready queue.
  - **Response time**- The time from the submission of a request until the first response is produced.
- It is desirable to **maximize CPU utilization and throughput** and to **minimize turnaround time, waiting time, and response time**.

# 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 scheduling algorithms can be used to decide which of the following processes in the ready queue is to be allocated the CPU:
  - First-Come, First-Served Scheduling(FCFS)
  - Shortest-Job-First Scheduling(SJF/SRTF)
  - Priority Scheduling
  - Round-Robin Scheduling(RR)
  - Multilevel Queue Scheduling
  - Multilevel Feedback-Queue Scheduling

# First-Come, First-Served Scheduling(FCFS)

- This is a Simplest , Non-Preemptive scheduling algorithm.
- With this scheme, the process that requests the CPU first is allocated the CPU first.
- Implemented through FIFO queue
- When a process enters the ready queue, its PCB is linked onto the tail of the queue.
- When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.

## Example to illustrate First-Come, First-Served (FCFS) Scheduling

Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds.

Process	Burst Time
P1	24
P2	3
P3	3

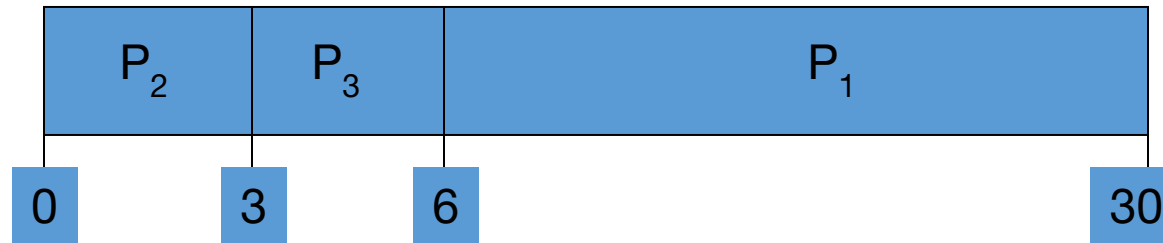
If the processes arrive in the order P1, P2, P3, and are served in FCFS Order, we get the result shown in the following Gantt chart:



Waiting time for P1 = 0; P2 = 24; P3 = 27

Average waiting time = waiting time of all processes / number of processes =  $(0 + 24 + 27) / 3 = 17$

- If the processes arrive in the order  $P_2$ ,  $P_3$ ,  $P_1$ , however, the results will be as shown in the following Gantt chart:



- The average waiting time is now  $(6 + 0 + 3)/3 = 3$  milliseconds.
- This is better than the previous case.
- Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the process's CPU burst times vary greatly.

### Drawback of FCFS:

Leads to convoy effect. (Larger CPU burst process preventing the smaller CPU burst Processes from execution)

- This effect results in lower CPU and device utilization.



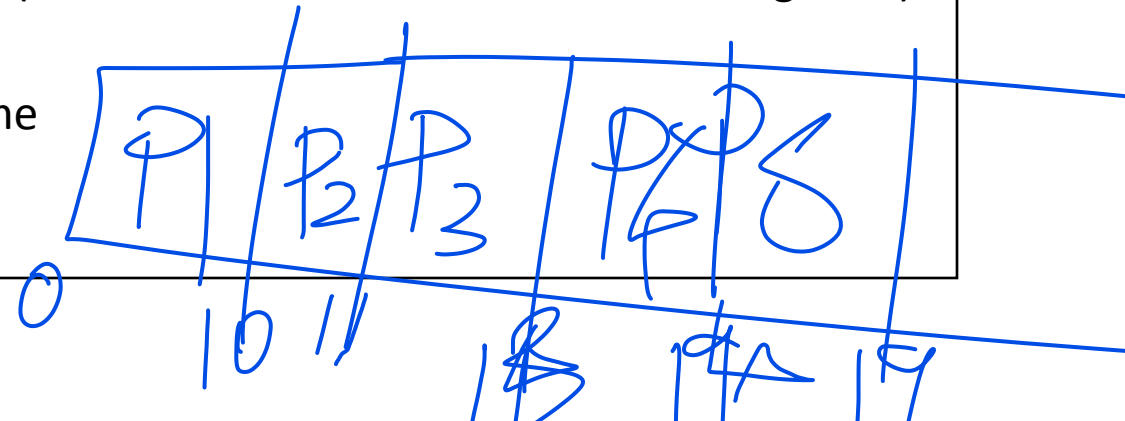
# Problems on FCFS scheduling algorithm

## Problem-1

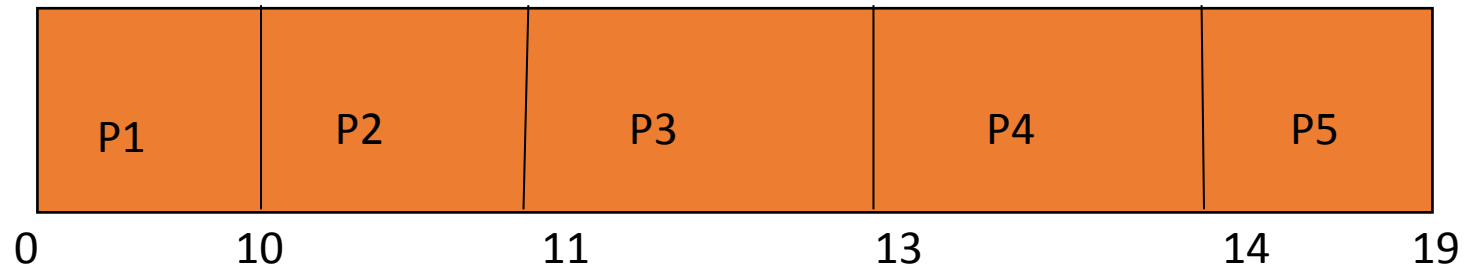
Consider the following set of processes with the length of the CPU burst and arrival time is given in milliseconds:

Processes	Arrival time	Burst time
P1	0	10
P2	0	1
P3	3	2
P4	5	1
P5	10	5

- (i) Draw a Gantt chart to illustrate the execution of these processes using FCFS scheduling.
- (ii) Calculate the turn around time for each of these processes. (turnaround time = burst time + waiting time)
- (iv) Calculate waiting time for each of these processes.
- (v) Calculate average waiting time and average turn around time



## Gantt chart to illustrate the execution of processes using FCFS.



To compute waiting time:

Waiting time = time at which process scheduled for execution - Arrival time

PROCESS	WAITING TIME(ms)
P1	0
P2	10
P3	8
P4	8
P5	4

**To compute average waiting time:**

Average waiting time = (sum of waiting time of all processes / total number of processes)  
$$= (0 + 10 + 8 + 8 + 4) / 5$$
$$= 6\text{ms}$$

To compute turnaround time:

PROCESS	TURN AROUND TIME (Waiting time + Burst time)
P1	0+10=10
P2	10+1=11
P3	8+2=10
P4	8+1=9
P5	4+5=9

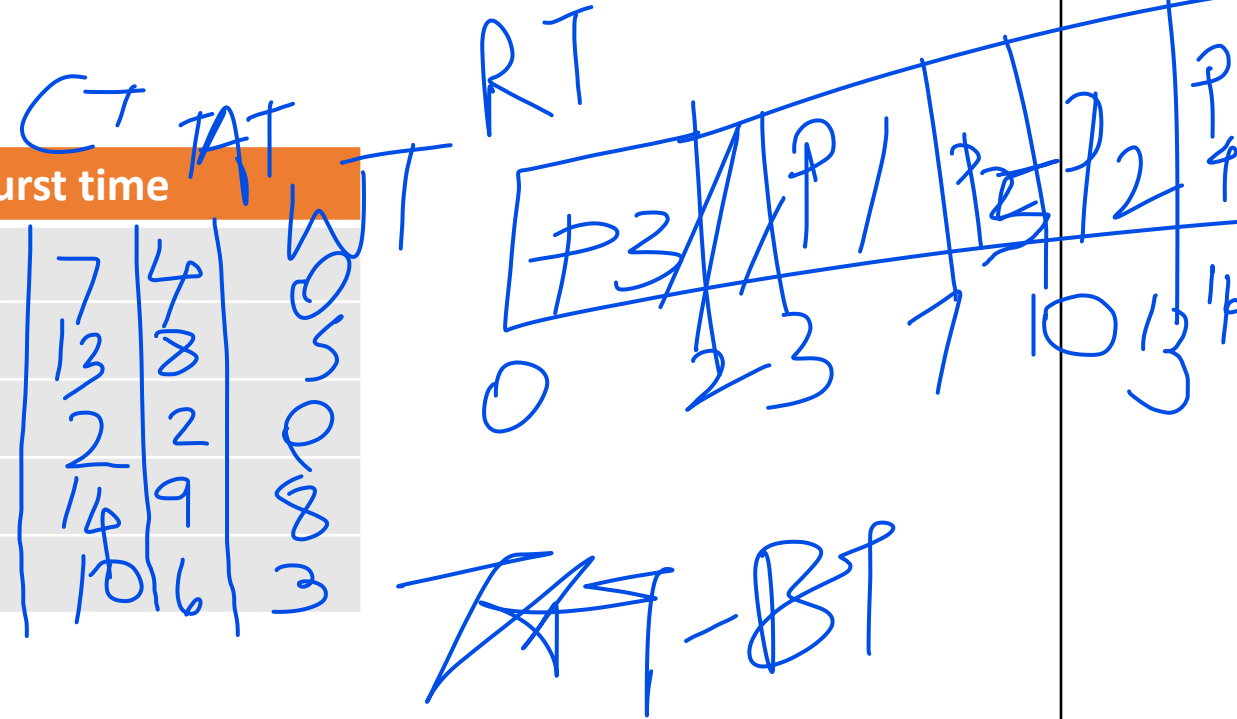
Average turn around time= (sum of turn around time of each process/total number of processes)

$$\begin{aligned}\text{Average turn around time} &= (10+11+10+9+9)/5 \\ &= 9.8\text{ms}\end{aligned}$$

## Problem-2

Consider the set of 5 processes whose arrival time and burst time are given below:

Process ID	Arrival time	Burst time
P1	3	4
P2	5	3
P3	0	2
P4	5	1
P5	4	3



If the CPU scheduling policy is FCFS, calculate the average waiting time and average turn around time.

CT

Solution

Gantt chart:



Gantt Chart

(Here, black box represents the idle time of CPU.)

Turn Around time = Burst time + waiting time

Or

{Turn Around time = Exit time – Arrival time}

Waiting time=arrival time - time at which process scheduled for execution.

or

{Waiting time = Turn Around time – Burst time}

Note: Any one of the above formula can be used to compute the waiting time and turn around time

Process Id	Exit time	Turn Around time	Waiting time
P1	7	$7 - 3 = 4$	$4 - 4 = 0$
P2	13	$13 - 5 = 8$	$8 - 3 = 5$
P3	2	$2 - 0 = 2$	$2 - 2 = 0$
P4	14	$14 - 5 = 9$	$9 - 1 = 8$
P5	10	$10 - 4 = 6$	$6 - 3 = 3$

Average Turn Around time =  $(4 + 8 + 2 + 9 + 6) / 5 = 29 / 5 = 5.8$  unit

Average waiting time =  $(0 + 5 + 0 + 8 + 3) / 5 = 16 / 5 = 3.2$  unit

Consider the set of 3 processes whose arrival time and burst time are given below

Process Id	Arrival time	Burst time
P1	0	2
P2	3	1
P3	5	6

If the CPU scheduling policy is FCFS, calculate the average waiting time and average turn around time.

Solution:

Gantt chart



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	2	$2 - 0 = 2$	$2 - 2 = 0$
P2	4	$4 - 3 = 1$	$1 - 1 = 0$
P3	11	$11 - 5 = 6$	$6 - 6 = 0$

we know that,

Turn Around time = Exit time – Arrival time

Waiting time = Turn Around time – Burst time

Average Turn Around time =  $(2 + 1 + 6) / 3 = 9 / 3 = 3$  unit

Average waiting time =  $(0 + 0 + 0) / 3 = 0 / 3 = 0$  unit



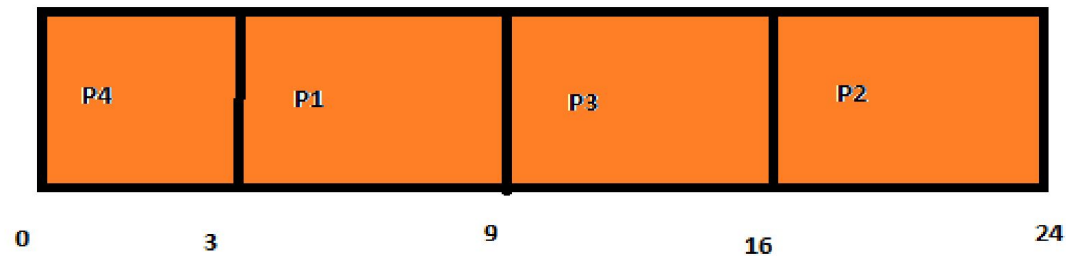
# Shortest-Job-First Scheduling

- This is also referred to as shortest next CPU burst algorithm.
- This algorithm associates with each process the length of the process's next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

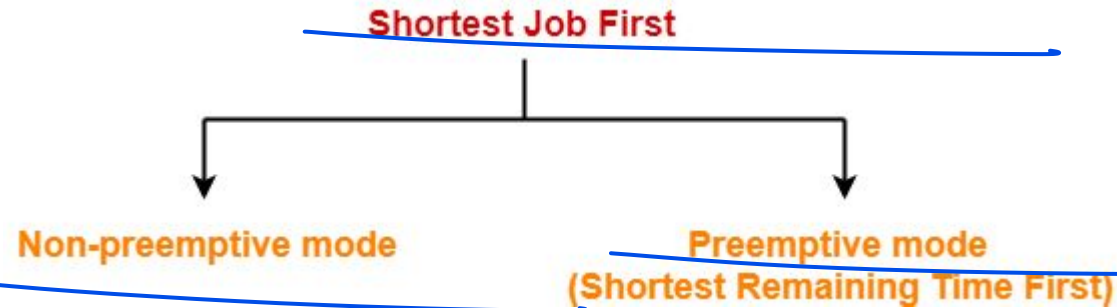
Example: Consider the following set of processes, with the length of the CPU burst given in milliseconds

Process	Burst time
P1	6
P2	8
P3	7
P4	3

- Using SJF scheduling, we would schedule these processes according to the following Gantt chart



- The waiting time is 3 milliseconds for process  $P_1$ , 16 milliseconds for process  $P_2$ , 9 milliseconds for process  $P_3$ , and 0 milliseconds for process  $P_4$ .
- The average waiting time is  $(3 + 16 + 9 + 0)/4 = 7$  milliseconds.
- The SJF algorithm can be either preemptive or nonpreemptive.
- The choice arises when a new process arrives at the ready queue while a previous process is still executing.
- The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process.
- A preemptive SJF algorithm will preempt the currently executing process, whereas a nonpreemptive SJF algorithm will allow the currently running process to finish its CPU burst.
- Preemptive SJF scheduling is sometimes called shortest-remaining-time-first scheduling (SRTF).



## **Advantages**

SRTF is optimal and guarantees the minimum average waiting time

## **Disadvantages**

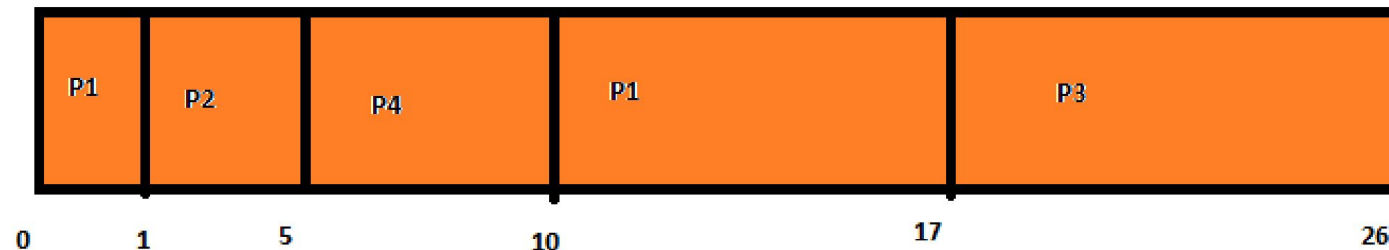
- It can not be implemented practically since burst time of the processes can not be known in advance.
- It leads to starvation for processes with larger burst time

## Example to illustrate ~~Preemptive SJF(or SRTF)~~

Consider the following four processes, with the length of the CPU burst given in milliseconds

Process	Arrival time	Burst time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

If the processes arrive at the ready queue at the times shown and need the indicated burst times, then the resulting preemptive SJF schedule is as depicted in the following Gantt chart:



- Process P1 is started at time 0, since it is the only process in the queue.
- Process P2 arrives at time 1. The remaining time for process *P1* (7 milliseconds) is larger than the time required by process *P2* (4 milliseconds), so process *P1* is preempted, and process *P2* is scheduled.
- The average waiting time for this example is

$$((10 - 1) + (1 - 1) + (17 - 2) + (5 - 3))/4 = 26/4 = 6.5 \text{ milliseconds.}$$

- Nonpreemptive SJF scheduling would result in an average waiting time of 7.75 milliseconds.

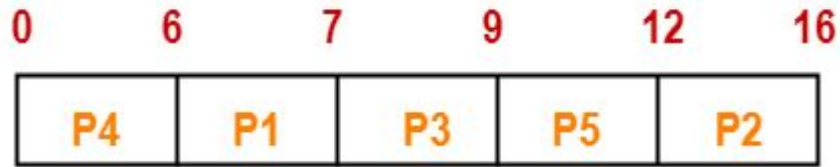
## Example problems on SJF

**Problem-1:** Consider the set of 5 processes whose arrival time and burst time are given below

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turn around time

# Gantt chart



**Gantt Chart**

we know that,

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	7	$7 - 3 = 4$	$4 - 1 = 3$
P2	16	$16 - 1 = 15$	$15 - 4 = 11$
P3	9	$9 - 4 = 5$	$5 - 2 = 3$
P4	6	$6 - 0 = 6$	$6 - 6 = 0$
P5	12	$12 - 2 = 10$	$10 - 3 = 7$



Average Turn Around time =  $(4 + 15 + 5 + 6 + 10) / 5 = 40 / 5 = 8$  unit

Average waiting time =  $(3 + 11 + 3 + 0 + 7) / 5 = 24 / 5 = 4.8$  unit

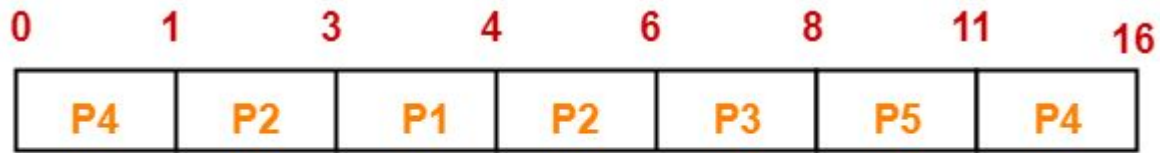
# Problems on preemptive SJF/SRTF

**Problem-1:** Consider the set of 5 processes whose arrival time and burst time are given below:

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF preemptive, calculate the average waiting time and average turn around time.

## Gantt Chart-



**Gantt Chart**

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	4	$4 - 3 = 1$	$1 - 1 = 0$
P2	6	$6 - 1 = 5$	$5 - 4 = 1$
P3	8	$8 - 4 = 4$	$4 - 2 = 2$
P4	16	$16 - 0 = 16$	$16 - 6 = 10$
P5	11	$11 - 2 = 9$	$9 - 3 = 6$

- Average Turn Around time =  $(1 + 5 + 4 + 16 + 9) / 5 = 35 / 5 = 7$  unit
- Average waiting time =  $(0 + 1 + 2 + 10 + 6) / 5 = 19 / 5 = 3.8$  unit

Example: Consider the following set of processes, with the length of the CPU burst given in milliseconds:

PROCESS	BURST TIME
P1	10
P2	1
P3	2
P4	1
P5	5

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

- Draw Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, SRTF (Preemptive SJF).
- What is the turnaround time of each process for each of the scheduling algorithms in part a?
- What is the waiting time of each process for each of the scheduling algorithms in part a?
- Which of the algorithms in part a results in the minimum average waiting time (over all processes)?

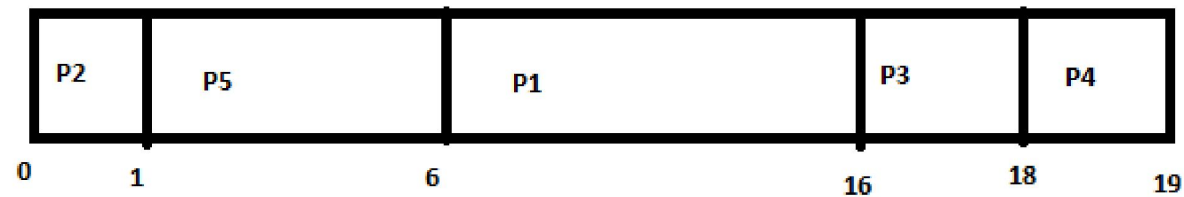
# Priority Scheduling

- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- Equal-priority processes are scheduled in FCFS order.

Consider the following set of processes, assumed to have arrived at time 0, in the order P1, P2, .. P5, with the length of the CPU burst given in milliseconds.

process	Burst time	priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Using priority scheduling, we would schedule these processes according to the following Gantt chart



The average wa

Priorities can be defined either **internally or externally**.

- Internally defined priorities use some measurable quantity or quantities to compute the priority of a process.
- For example, time limits, memory requirements, the number of open files, and the ratio of average I/O burst to average CPU burst have been used in computing priorities.
- External priorities are set by criteria outside the operating system, such as the importance of the process, the type and amount of funds being paid for computer use etc.
- Priority scheduling can be either **preemptive or non-preemptive**.
- When a process arrives at the ready queue, its priority is compared with the priority of the currently running process. A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.



- A nonpreemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.
- A major problem with priority scheduling algorithms is **indefinite blocking, or starvation**.

(Priority scheduling algorithm can leave some low priority processes waiting indefinitely. In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU).

- A solution to the problem of indefinite blockage of low-priority processes is **aging**.
- **Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.**

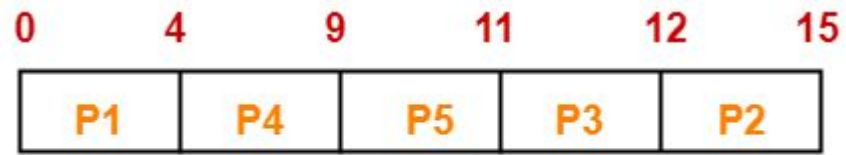
## Example Problem

**Problem-1:** Consider the set of 5 processes whose arrival time and burst time are given below

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

# Gantt chart



Gantt Chart

we know that,

Turn Around time = Exit time – Arrival time

Waiting time = Turn Around time – Burst time

Average Turn Around time =  $(4 + 14 + 10 + 6 + 7) / 5 = 41 / 5 = 8.2$  unit

Average waiting time =  $(0 + 11 + 9 + 1 + 5) / 5 = 26 / 5 = 5.2$  unit

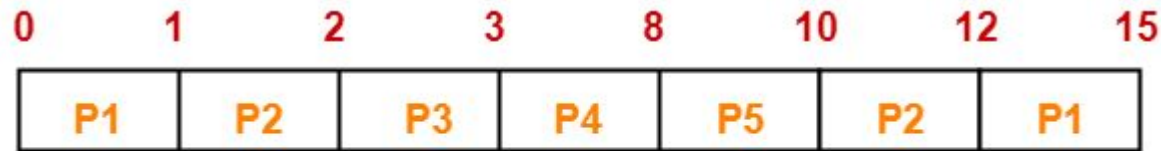
Process Id	Exit time	Turn Around time	Waiting time
P1	4	$4 - 0 = 4$	$4 - 4 = 0$
P2	15	$15 - 1 = 14$	$14 - 3 = 11$
P3	12	$12 - 2 = 10$	$10 - 1 = 9$
P4	9	$9 - 3 = 6$	$6 - 5 = 1$
P5	11	$11 - 4 = 7$	$7 - 2 = 5$

Consider the set of 5 processes whose arrival time and burst time are given below

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is ~~priority preemptive~~, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

# Gantt chart



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	15	$15 - 0 = 15$	$15 - 4 = 11$
P2	12	$12 - 1 = 11$	$11 - 3 = 8$
P3	3	$3 - 2 = 1$	$1 - 1 = 0$
P4	8	$8 - 3 = 5$	$5 - 5 = 0$
P5	10	$10 - 4 = 6$	$6 - 2 = 4$

Average Turn Around time =  $(15 + 11 + 1 + 5 + 6) / 5 = 38 / 5 = 7.6$  unit

Average waiting time =  $(11 + 8 + 0 + 0 + 4) / 5 = 23 / 5 = 4.6$  unit

# Round-Robin Scheduling

- The round-robin (RR) scheduling algorithm is designed especially for timesharing systems.
- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time, called a time quantum or time slice, is defined.  
(A time quantum is generally from 10 to 100 milliseconds).
- The ready queue is treated as a circular queue.
- The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.
- To implement RR scheduling, we keep the ready queue as a FIFO queue of Processes.

- New processes are added to the tail of the ready queue.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.
- One of two things will then happen.
- The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, 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 operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

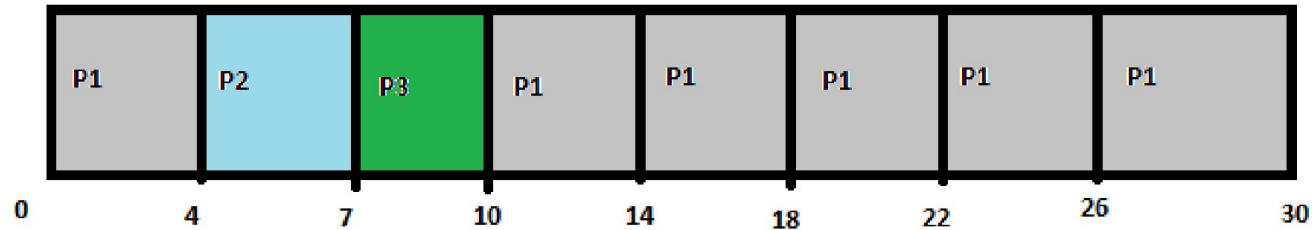
Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst time
P1	24
P2	3
P3	3

- If we use a time quantum of 4 milliseconds, then process P1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process P2 .
- Since process P2 does not need 4 milliseconds, it quits before its time quantum expires.
- The CPU is then given to the next process, process P3.
- Once each process has received 1 time quantum, the CPU is returned to process P1 for an additional time quantum.



## Gantt chart



- The average waiting time is  $17/3 = 5.66$  milliseconds.
- In the RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row (unless it is the only runnable process).
- If a process's CPU burst exceeds 1 time quantum, that process is *preempted* and is put back in the ready queue.
- The RR scheduling algorithm is thus **preemptive**.

- If there are 11 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.
- Each process must wait no longer than  $(n - 1) \times q$  time units until its next time quantum.
- For example, with five processes and a time quantum of 20 milliseconds, each process will get up to 20 milliseconds every 100 milliseconds.

# Problems on Round robin scheduling

**Problem-1:** Consider the set of 5 processes whose arrival time and burst time are given below

Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turn around time.

# Gantt chart

READY QUEUE

P5, P1, P2, P5, P4, P1, P3, P2, P1



Gantt Chart

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	13	$13 - 0 = 13$	$13 - 5 = 8$
P2	12	$12 - 1 = 11$	$11 - 3 = 8$
P3	5	$5 - 2 = 3$	$3 - 1 = 2$
P4	9	$9 - 3 = 6$	$6 - 2 = 4$
P5	14	$14 - 4 = 10$	$10 - 3 = 7$

**To compute Average turn around time and Average waiting time:**

Average Turn Around time =  $(13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6$  unit

Average waiting time =  $(8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8$  unit

Problem-2: Consider the set of 6 processes whose arrival time and burst time are given below

Process Id	Arrival time	Burst time
P1	0	4
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 2, calculate the average waiting time and average turn around time.

Solution:

P5, P6, P2, P5, P6, P2, P5, P4, P1, P3, P2, P1

READY QUEUE

Gantt chart



Gantt Chart

Continued..

we know that,

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	8	$8 - 0 = 8$	$8 - 4 = 4$
P2	18	$18 - 1 = 17$	$17 - 5 = 12$
P3	6	$6 - 2 = 4$	$4 - 2 = 2$
P4	9	$9 - 3 = 6$	$6 - 1 = 5$
P5	21	$21 - 4 = 17$	$17 - 6 = 11$
P6	19	$19 - 6 = 13$	$13 - 3 = 10$

Continued...



To compute Average waiting time and turn around time

$$\begin{aligned}\text{Average Turn Around time} &= (8 + 17 + 4 + 6 + 17 + 13) / 6 \\ &= 65 / 6 = 10.84 \text{ unit}\end{aligned}$$

$$\begin{aligned}\text{Average waiting time} &= (4 + 12 + 2 + 5 + 11 + 10) / 6 \\ &= 44 / 6 = 7.33 \text{ unit}\end{aligned}$$

**Problem-3:** Four jobs to be executed on a single processor system arrive at time 0 in the order A, B, C, D. Their burst CPU time requirements are 4, 1, 8, 1 time units respectively. The completion time of A under round robin scheduling with time slice of one time unit is

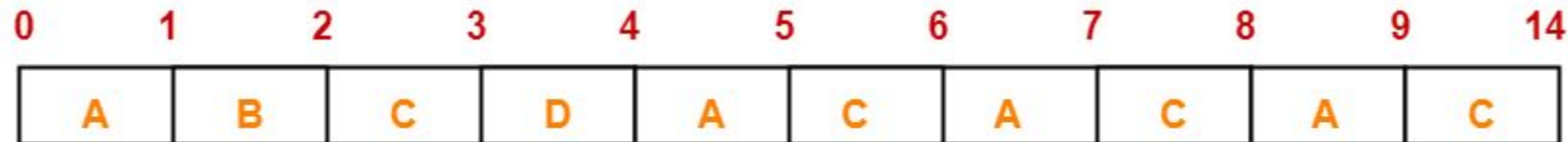
- A. 10
- B. 4
- C. 8
- D. 9

## Solution:

Process Id	Arrival time	Burst time
A	0	4
B	0	1
C	0	8
D	0	1

C, A, C, A, C, A, D, C, B, A

ready queue



Gantt Chart

Therefore, completion time of process A = 9 unit.

## Practice problems

1. Consider three process, all arriving at time zero, with total execution time of 10, 20 and 30 units respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of does the CPU remain idle?

- a. 0%
- b. 10.6%
- c. 30.0%
- d. 89.4%

Continued...

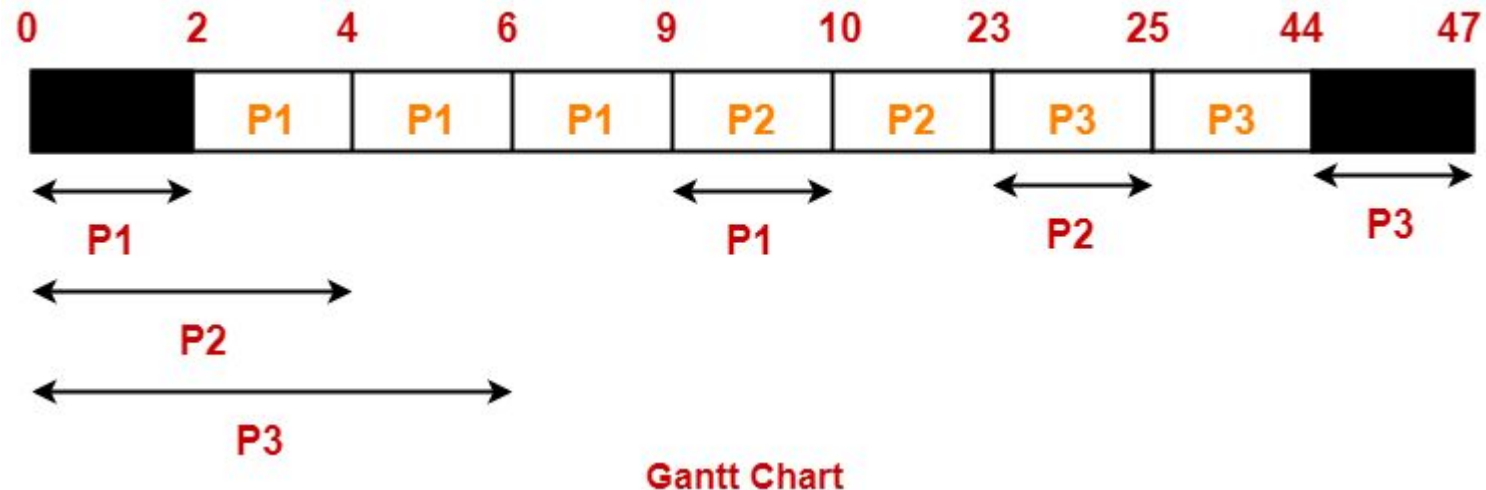
Solution:

According to question, we have

	Total burst time	I/o burst	CPU burst	I/O Burst
Process P1	10	2	1	1
Process P2	20	4	14	2
Process P3	30	6	21	3

The scheduling algorithm used is Shortest Remaining Time First

Gantt chart



Percentage of time CPU remains idle

$$= (5 / 47) \times 100$$

$$= 10.638\%$$

Thus, Option (B) is correct.

**Problem-2:** Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

Continued.....

- a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: PCPS, SIP, nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1).
- b. What is the turnaround time of each process for each of the scheduling algorithms in part a?
- c. What is the waiting time of each process for each of the scheduling algorithms in part a?
- d. Which of the algorithms in part a results in the minimum average waiting time (over all processes)?



# Multilevel Queue scheduling

- Created for situations in which processes are easily classified into different groups.
- Example, a common division is made between foreground (interactive) processes and background (batch) processes.
- These two types of processes have different response-time requirements and so may have different scheduling needs. In addition, foreground processes may have priority (externally defined) over background processes.
- A multilevel queue scheduling algorithm partitions the ready queue into several separate queues as shown in the figure

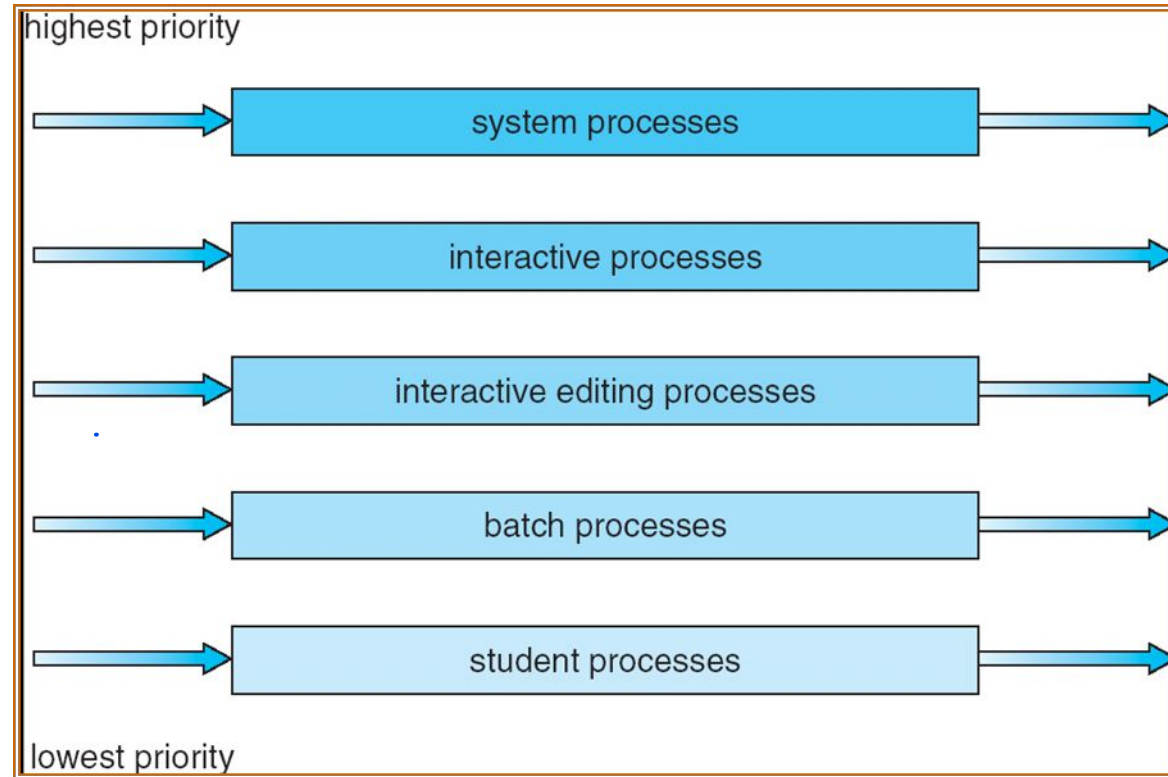



Figure: Multilevel Queue scheduling

- The processes are permanently assigned to one queue, based on some property of the process, such as memory size, process priority, or process type.
- Each queue has its own scheduling algorithm. For example, separate queues might be used for foreground and background processes. The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.
- In addition, 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.

Figure represents an example of a multilevel queue scheduling algorithm with five queues, listed below in order of priority:

1. System processes
  2. Interactive processes
  3. Interactive editing processes
  4. Batch processes
  5. Student processes
- 

- Each queue has absolute priority over lower-priority queues.
- No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process would be preempted.
- Another possibility is to time-slice among the queues. Here, each queue gets a certain portion of the CPU time, which it can then schedule among its various processes

# Multilevel Feedback-Queue Scheduling

- Major drawback of multilevel queue scheduling algorithm is it is **inflexible**. (i.e. Processes are permanently assigned to a queue when they enter the system.).
- The multilevel feedback queue scheduling algorithm, allows a **process to move between queues**.
- The idea is to separate processes according to the characteristics of their CPU bursts.
- 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.
- In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

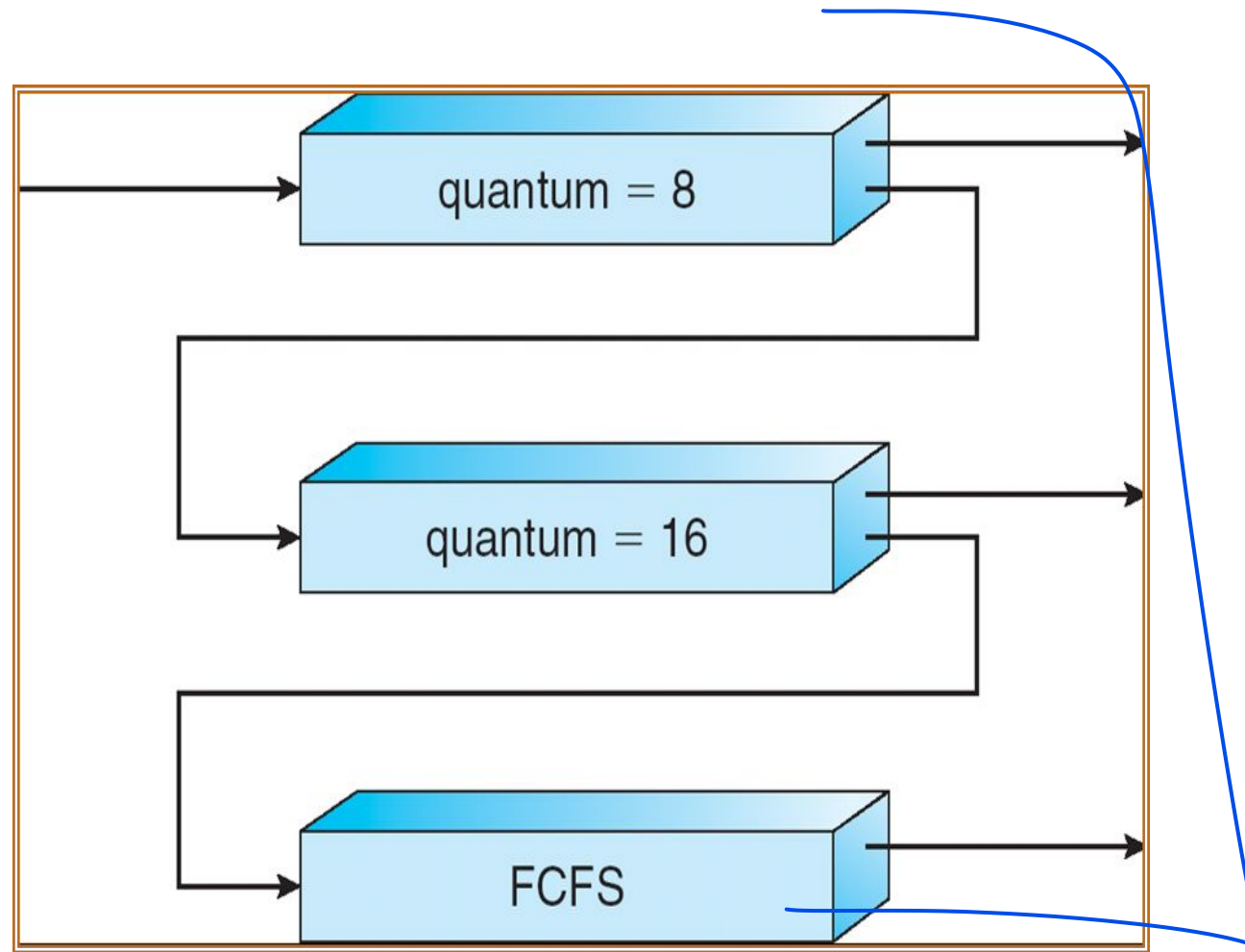


Figure: Multilevel feedback queues

- Figure represents a multilevel feedback-queue 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.
- A process entering the ready queue is put in queue 0. A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1.
- If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2. Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.
- The multilevel feedback queue scheduling algorithm shown in figure ,gives highest priority to any process with a CPU burst of 8 milliseconds or less. Such a process will quickly get the CPU, finish its CPU burst, and go off to its next I/O burst. Processes that need more than 8 but less than 24 milliseconds are also served quickly, although with lower priority than shorter processes. Long processes automatically sink to queue 2 and are served in FCFS order with any CPU cycles left over from queues 0 and 1.

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

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



# Multiple-Processor Scheduling

So far ,we Focused on the problems of scheduling the CPU in a system with a single processor.

If multiple CPUs are available, load sharing becomes possible; however, the scheduling problem becomes correspondingly more complex.

## Approaches to Multiple-Processor Scheduling

1. Asymmetric multiprocessing
2. Symmetric multiprocessing (SMP)

## Asymmetric multiprocessing

This approach to CPU scheduling in a multiprocessor system has all scheduling decisions, I/O processing, and other system activities handled by a single processor-the master server. The other processors execute only user code.

Asymmetric multiprocessing is simple because only one processor accesses the system data structures, reducing the need for data sharing.

## Symmetric multiprocessing (SMP)

In this approach each processor is self-scheduling. All processes may be in a common ready queue, or each processor may have its own private queue of ready processes. scheduling proceeds by having the scheduler for each processor examine the ready queue and select a process to execute.

# Issues concerning SMP systems.

## 1. Processor Affinity

- The data most recently accessed by the process populates the cache for the processor; and as a result, successive memory accesses by the process are often satisfied in cache memory.
- If the process migrates to another processor, The contents of cache memory must be invalidated for the processor being migrated from, and the cache for the processor being migrated to must be re-populated.
- Because of the high cost of invalidating and re-populating caches, most SMP systems try to avoid migration of processes from one processor to another and instead attempt to keep a process running on the same processor. This is known as **processor affinity**, meaning that a process has an affinity for the processor on which it is currently running.

## Two Forms of Processor Affinity

- **Soft affinity**

When an operating system has a policy of attempting to keep a process running on the same processor-but not guaranteeing that it will do so is called soft affinity.

- **Hard affinity**

OS provide system calls that support hard affinity, thereby allowing a process to specify that it is not to migrate to other processors.

## 2. Load Balancing

Load balancing attempts to keep the workload evenly distributed across all processors in an SMP system.

There are two general approaches to load balancing:

1. **push migration**: In case of push migration, a specific task periodically checks the load on each processor and-if it finds an imbalance-evenly distributes the load by moving (or pushing) processes from overloaded to idle or less-busy processors.
2. **pull migration**: Pull migration occurs when an idle processor pulls a waiting task from a busy processor.

# Algorithm Evaluation

- There are several challenges involved in selecting a scheduling algorithm for a particular system.
- The first problem is defining a criteria to be used in selecting an algorithm. Criteria are often defined in terms of CPU utilization, response time, or throughput. To select an algorithm, we must first define the relative importance of these measures.
- Once the selection criteria have been defined, we want to evaluate the algorithms under consideration.

# Evaluation methods

- **Deterministic Modelling**

This method takes a particular predetermined workload and defines the performance of each algorithm for that workload.

Example:

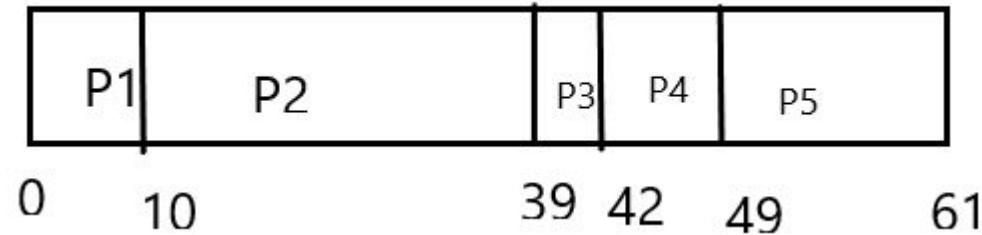
Consider the following workload, All five processes arrive at time 0, in the order given, with the length of the CPU burst given in milliseconds:

PROCESS	Burst time
P1	10
P2	29
P3	3
P4	7
P5	12

**Consider the FCFS, SJF, and RR (quantum = 10 milliseconds) scheduling algorithms for this set of processes. Which algorithm would give the minimum average waiting time?**

Solution:

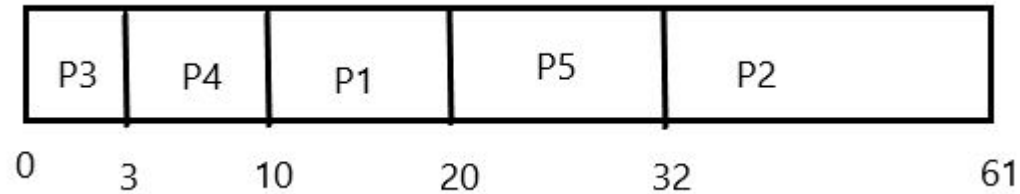
For the FCFS algorithm, we would execute the processes as



The waiting time is 0 milliseconds for process P1, 10 milliseconds for process P2, 39 milliseconds for process P3, 42 milliseconds for process P4, and 49 milliseconds for process P5. Thus, the average waiting time is  $(0 + 10 + 39 + 42 + 49)/5 = 28$  milliseconds.

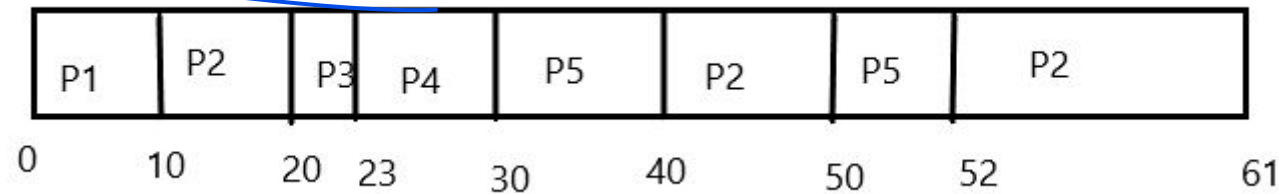


With nonpreemptive SJF scheduling, we execute the processes as



The waiting time is 10 milliseconds for process P1, 32 milliseconds for process P2, 0 milliseconds for process P3, 3 milliseconds for process P4, and 20 milliseconds for process P5. Thus, the average waiting time is  $(10 + 32 + 0 + 3 + 20)/5 = 13$  milliseconds.

With the RR algorithm, we execute the processes as



The waiting time is 0 milliseconds for process P1, 32 milliseconds for process P2 , 20 milliseconds for process P3 , 23 milliseconds for process P4 , and 40 milliseconds for process P5. Thus, the average waiting time is  $(0 + 32 + 20 + 23 + 40)/5 = 23$  milliseconds.

We see that, in this case, the average waiting time obtained with the SJF policy is less than half that obtained with FCFS scheduling; the RR algorithm gives us an intermediate value.

## Queueing Models

- On many systems, the processes that are run vary from day to day, so there is no static set of processes (or times) to use for deterministic modeling.
- however, the distribution of CPU and I/O bursts can be determined. These distributions can be measured and then approximated or simply estimated.
- let  $n$  be the average queue length (excluding the process being serviced), let  $W$  be the average waiting time in the queue, and  $\lambda$  be the average arrival rate for new processes in the queue (such as three processes per second). We expect that during the time  $W$  that a process waits,  $\lambda \times W$  new processes will arrive in the queue. If the system is in a steady state, then the number of processes leaving the queue must be equal to the number of processes that arrive. Thus,  $n = \lambda \times W$ . This equation, known as **Little's formula**.

For example, if we know that 7 processes arrive every second (on average), and that there are normally 14 processes in the queue, then we can compute the average waiting time per process as 2 seconds using little's formula.

### **Limitation of queueing models**

The classes of algorithms and distributions that can be handled are fairly limited.

### **Simulations**

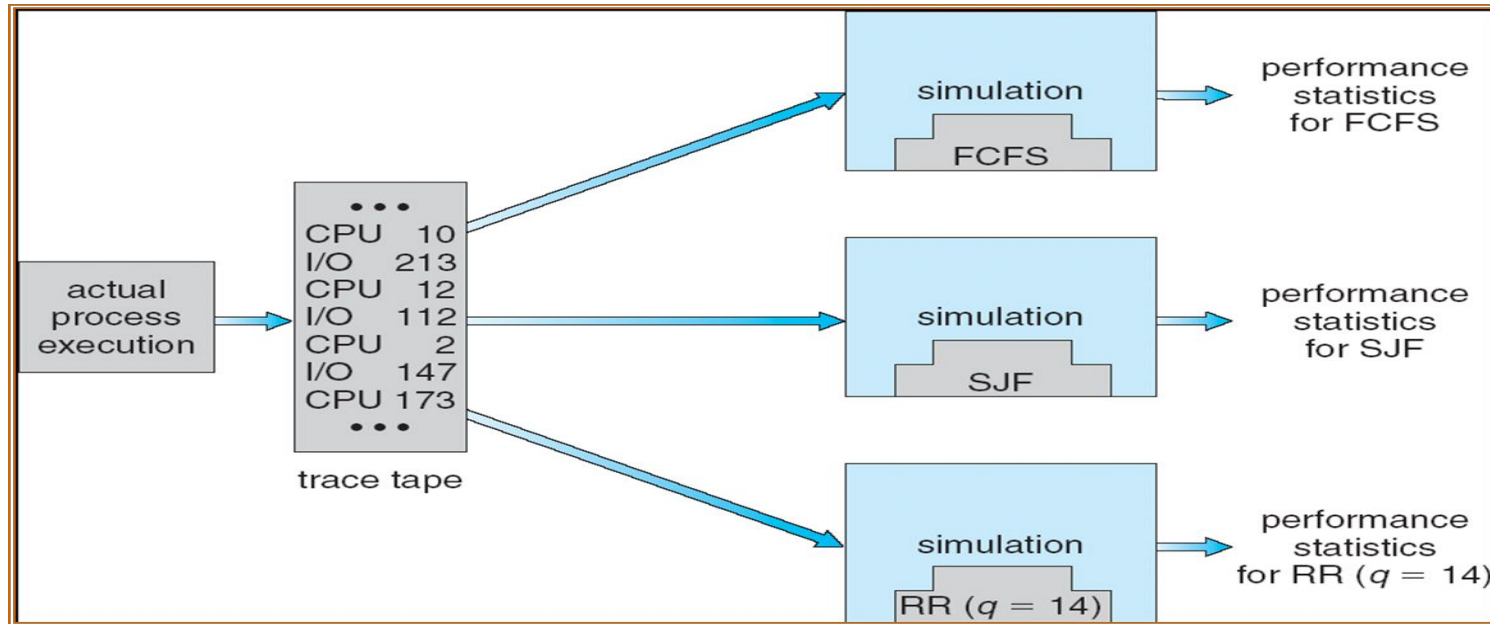
To get a more accurate evaluation of scheduling algorithms, we can use simulations.

Running simulations involves programming a model of the computer system.

Software data structures represent the major components of the system.

The simulator has a variable representing a clock; as this variable's value is increased, the simulator modifies the system state to reflect the activities of the devices, the processes, and the schedule

As the simulation executes, statistics that indicate algorithm performance are gathered and printed.



- The data to drive the simulation can be generated in several ways such as random-number generator, trace tapes etc.

### Drawback

- Simulations can be expensive, often requiring hours of computer time

**Thank you**