UNIT - 2

Process

and

CPU Scheduling

By

Dr Capt R B Kallam

Topics to be covered

- Process Concept
- Process Scheduling
- Operation on Processes
- Cooperating Processes
- Interposes Communication
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling

Process Concept:

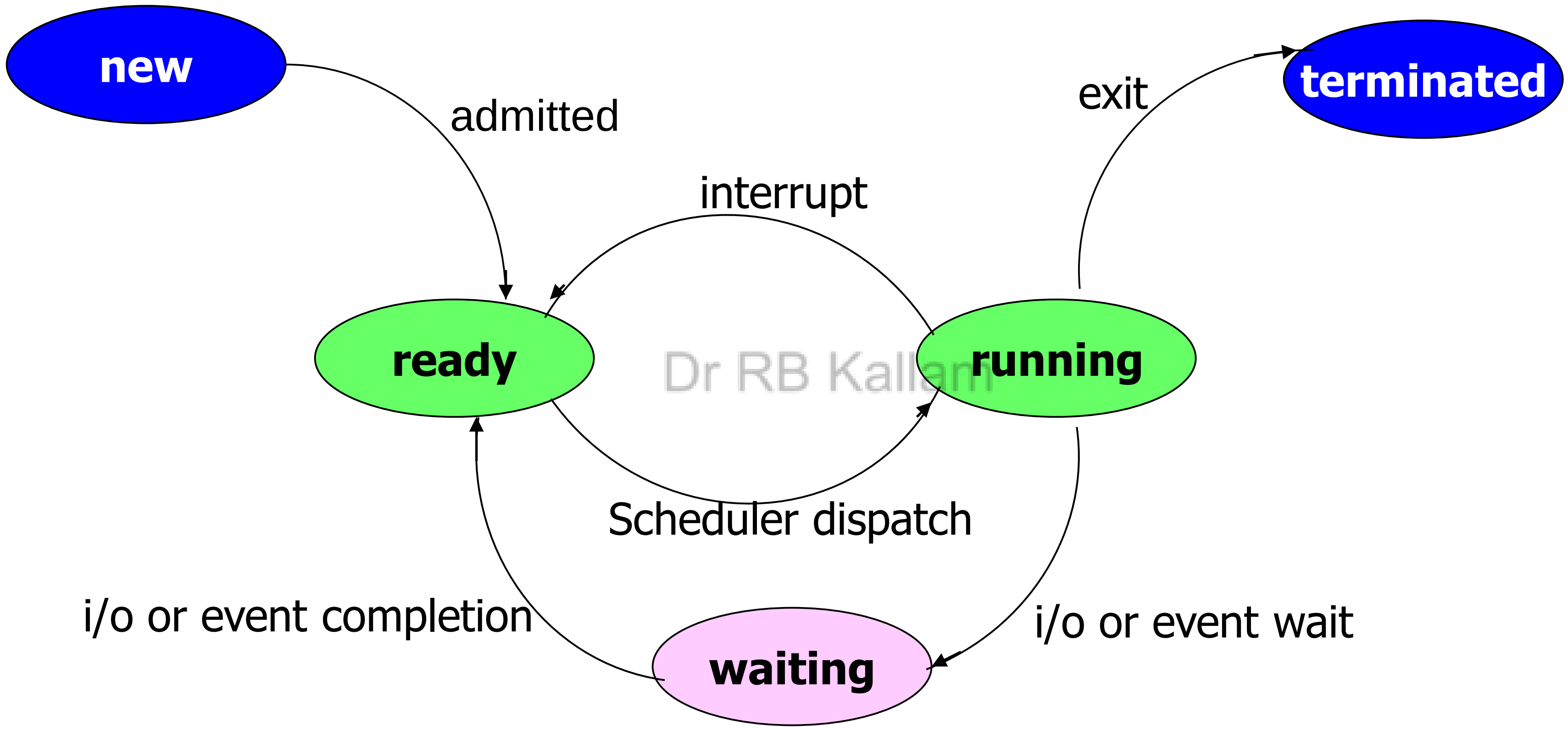
The process:

- The program currently under execution mode is called a process.
- The execution of a process must progress in a sequential fashion, based *Program counter*.
- At any time, at most one instruction is executed on behalf of the process.
- A program is a passive entity such as the contents of a file stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.

Process State:

- **The state of a process is defined by the current activity of that process.**
- ***Each process may be in one of the following states:***

- **New: The process is being created.**
- **Ready: The process is waiting to be assigned to a processor.**
- **Running: Instructions are being executed.**
- **Waiting: The process is waiting for some event to occur.**
- **Terminated: The process has finished execution.**



Process state transition diagram or 5 state diagram

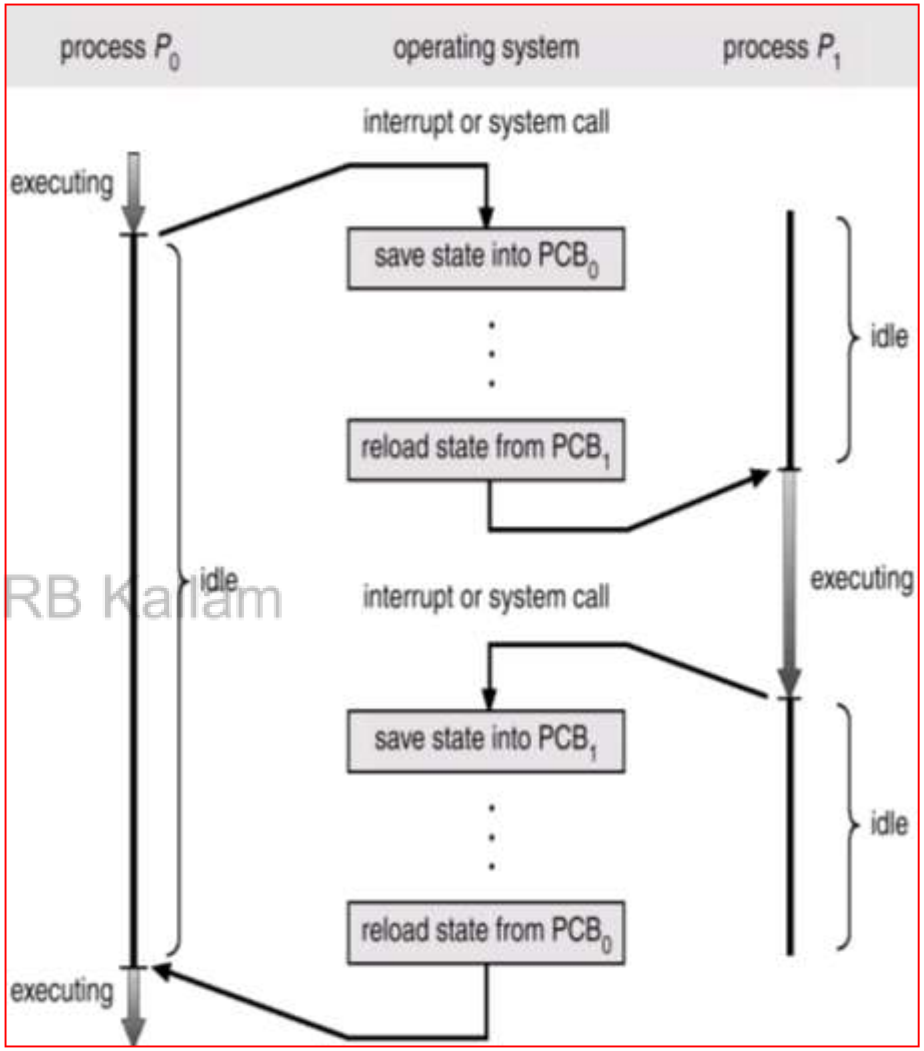
Process Control Block:

- Each process is represented in the OS by a Process control block (PCB) or task control block.
- It contains information associated with a specific process, including following:
  - **Pointer :**– It is a stack pointer which is required to be saved *when the process is switched from one state to another* to retain the current position of the process.
  - **Process state:-** It stores the respective state of the process
  - **Process Number**: A Unique number to identify the process
  - **Program counter:** It indicates the address of the next instruction to be executed for this process.
  - **CPU registers**: The registers vary in number and type, depending on the computer architecture. They include accumulators, instruction registers, stack pointers, Program counter, general-purpose registers, etc. *Process state information must be saved in registers when an interrupt occurs, to allow the process to be continued correctly afterward.*

- **CPU scheduling information: I**t maintains the scheduling information
- **Memory management information:** It includes the value of the base and limit registers, the page tables, or the segment tables etc
- **Accounting information:** This information includes the amount of CPU time used and usage of other resources by the process and so on.
- **I/O status information:** The information includes the list of I/O devices allocated to this process and so on.
- **List of Open files:** A process can deal with a number of files, so the CPU should maintain a list of files that are being opened by a process to make sure that no other process can open the file at the same time.

Pointer	Process state
Process number	
Program counter	
Registers	
Memory limits	
Accounting Information I/O Status Information List of open files	

Process Control Block (PCB)

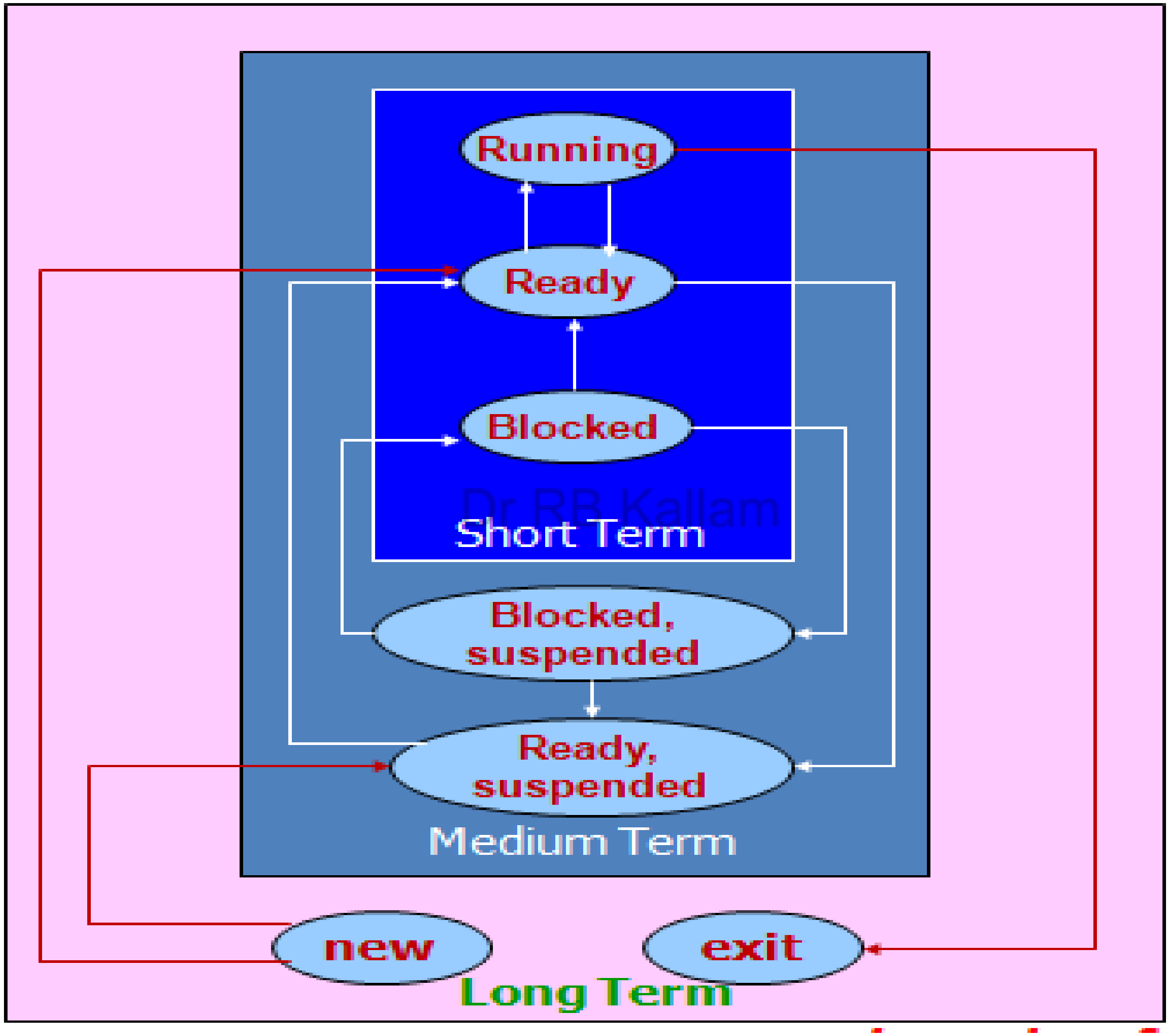


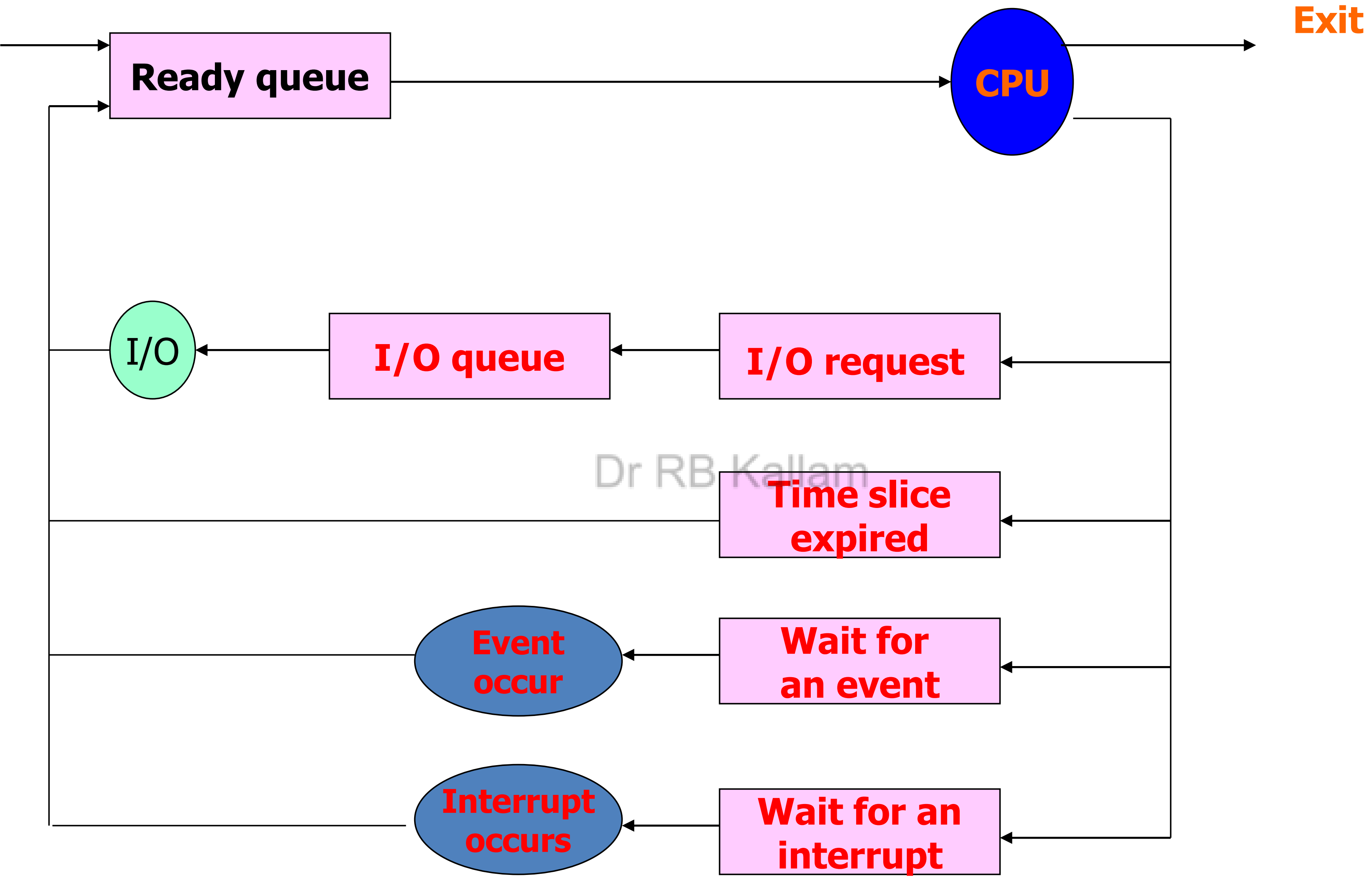
CPU Switch From Process to Process

Process Scheduling:

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- For a uni-processor system, there will never be more than one running process. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.
- **Types of scheduling:**
  - Long term scheduling: The decision to load the pool of processes into main memory for execution.
  - Medium term scheduling: The decision to add a process from the blocked or suspended states to ready state.
  - Short term scheduling: The decision as to which available processes will be executed by the processor
  - I/O Scheduling: The decision as to which processes pending I/O request shall be handled by an available I/O device.

Levels of scheduling

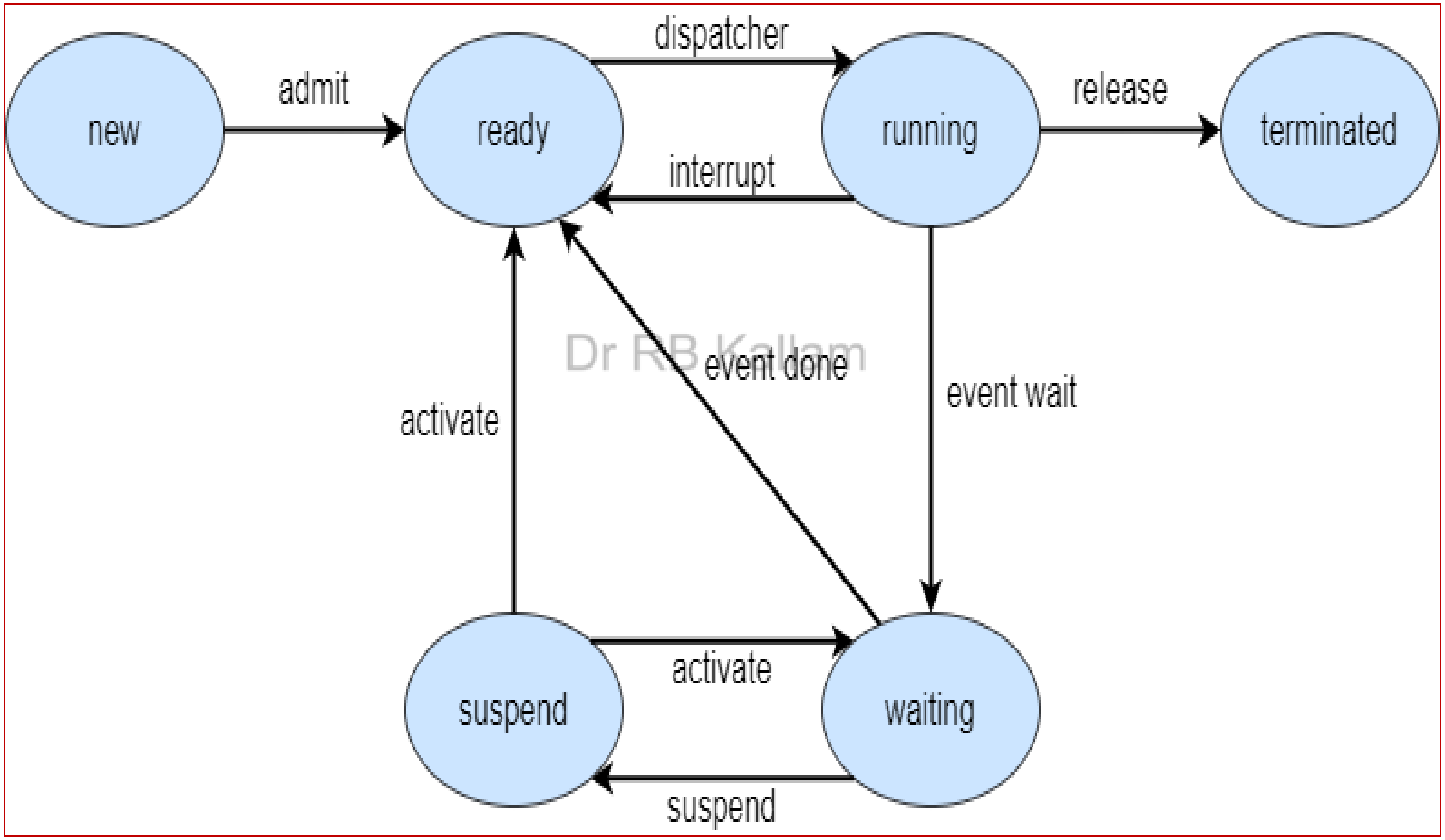




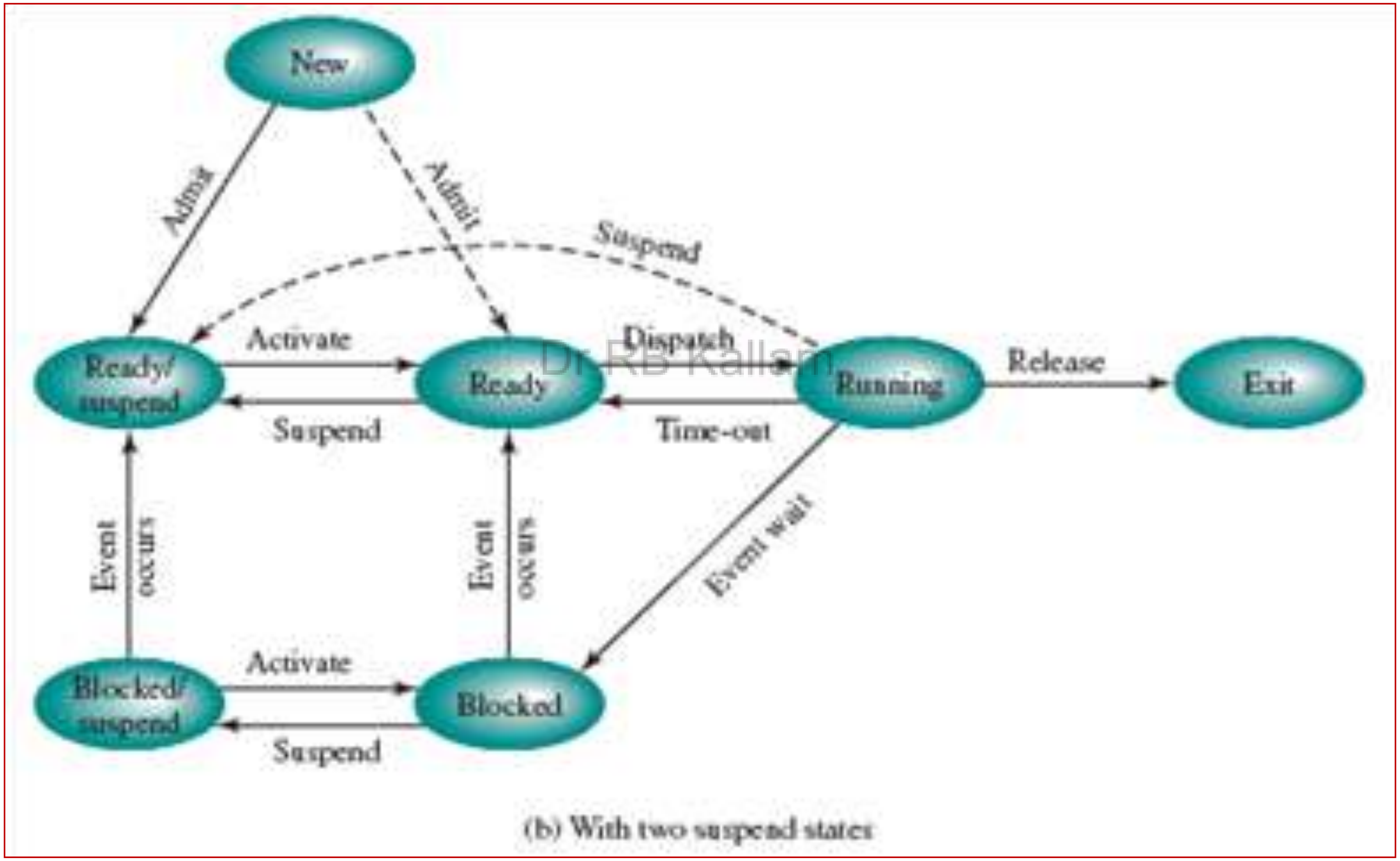
Queuing diagram representation of process scheduling

- Context Switching: Switching the CPU to another process requires performing a state save of the current process and state restore of a different process. This task is known as context switch.
- Process Switching: Switching the process from one state to another state due to several reasons.
- Saving the Context: When a CPU switches from one process to another, it saves the required values of the current process.
- Restoring the Context: When CPU retrieves a previously swapped out process it restores the state of the process.
- I/O Bounded Process: If a process mainly need to perform I/O operations then it is called I/O bounded program
- CPU Bounded Process: If a process mainly need to perform execution using CPU; then it is called CPU bounded program

Process state diagram with one suspended state- Six state diagram:



Process state transition diagram with two suspended states- Seven state model



Operation on Processes:

- Process creation:
  - A process may create several new processes, via *a create process system call,* during the cores of execution.
  - The creating process is called a parent process, where as the new processes are called the children of that process.
  - Each of these new processes may in turn create other processes, forming a tree of processes.
  - In Unix a new process is created by the *fork( ) system call with different process ID.*
  - The new process *consists of the copy of the address space of the original process.*
  - This mechanism allows the parent process to communicate easily with its child.

- The *exec( )* system call is invoked after a *fork( )*, by one of the two processes to replace the process memory space with a new program.
- The *exec( )* system call loads a binary file into memory and starts execution.
- The parent can then create more children; or, if it has nothing else to do while the child runs, it can issue a *wait ()* system call to move itself off the ready queue until the termination of the child.

- When process creates a new process, two possibilities exist in terms of execution:
  - The parent continue to execute concurrently with its children.
  - The parent waits until some or all of its children have terminated.
- There are also two possibilities in terms of the address space of the new process:
  - The child process is a duplicate of the parent process with different IDs.
  - The child process has a program loaded into it and have the same ID of the parent.

Reasons for Process Creation:

- New batch job: The OS is provided with a batch job control stream, usually on tape or disk. When a OS is prepared to take on new work, it will read the next sequence of job control command.
- Interactive logon: A user at terminal logs on to the system.
- Created by OS to provide a service: the OS can create a process to perform some function on behalf of a user program, without the user having to wait.
- Spawned by existing process: for the purpose of modularity or to exploit parallelism, a program can dictate the creation of number of processes.

- Process Termination:
  - A process terminates when it finishes executing its last statement and asks the *OS* to delete it by using *exit system call.*
  - All of the resources of the process are de -allocated by the Operating System when it terminates.
  - A parent process may also terminates its child process for various reasons:
    - The child has exceeded its usage of some of the resources it has been allocated.
    - The task assigned to the child is no longer required.
    - If the parent exit, and the OS does not allow a child to continue. *This phenomenon is referred to as cascading termination.*

Cooperating Processes:

The concurrent processes executing in the operating system may be either independent processes or cooperating processes.

The reasons for providing an environment that allows process cooperation are:

- Information Sharing: Since several users may be interested in the same piece of information, we must provide an environment to allow concurrent access to these types of resources.
- Computing speed up: If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. *Note that such a speedup can be achieved only if the computer has multiple processing elements (such as CPUs or I/O channels).*
- Modularity: We may want to construct the system in a modular fashion, dividing the system functions into separate processes for computation speedup or to use the resources in efficient manner.
- Convenience: Even an individual user may have many tasks to work on at one time. For instance, a user may be editing, printing, and compiling in parallel.

  Note: *For Concurrent execution processes need to communicate and synchronize their actions.*

Threads:

- *A thread, sometimes called a lightweight process (LWP), is a basic unit of CPU utilization, and consists of a program counter, a register set and a stack space.*
- Threads can share resources (code, data, OS) with its peer threads.
- The extensive sharing makes CPU switching among peer threads and the creation of threads inexpensive, compared with context switches among heavyweight processes.
- Sometimes thread switching does not need to call the OS, and to cause an interrupt to the kernel and hence it is very fast.

- Threads operate in many respects, in the same manner as processes.
- Threads can be in one of the states: ready, blocked, running or terminated.
- Threads can share the CPU, and only one thread at a time is active (running).
- A thread within a process executes sequentially, and each thread has its own stack and PC.
- Threads can create child threads, and can block waiting for system calls to complete.

Inter process Communication:

- Basic Structure:
  - One way to provide IPC is, it requires that the process should share common buffer space, and that the code for implementing the buffer be explicitly written by the application programmer.
  - Another is the OS should provide IPC
  - IPC is best provided by the means of message passing system.
  - An IPC provide at least the two operations:
    - *Send (message)*
    - *Receive (message)*

- Naming:
  - Processes that want to communicate with each other must have a way to refer to each other.
  - We have two methods:
    - **Direct communication:**
      - A process can directly communicate by using the following two operations:

» Send (destination, message)

» Receive(source, message)

A communication link in this scheme has the following properties:

- A link is established automatically between every pair of processes that want to communicate.
- A link is associated with exactly two processes.
- Between each pair of processes, there exists exactly one link

- **Indirect communication:**
  - With this messages are sent to and received from *mailboxes*.
  - The send and receive primitives are defined as follows:
    - » Send (A, message). Send a message to mailbox A
    - » Receive (A, message). Receive a message from mailbox A.

A communication link in this scheme has the following properties:

- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.
- Between each pair of communicating processes, there may be a number of different links, with each link corresponding to one mailbox.

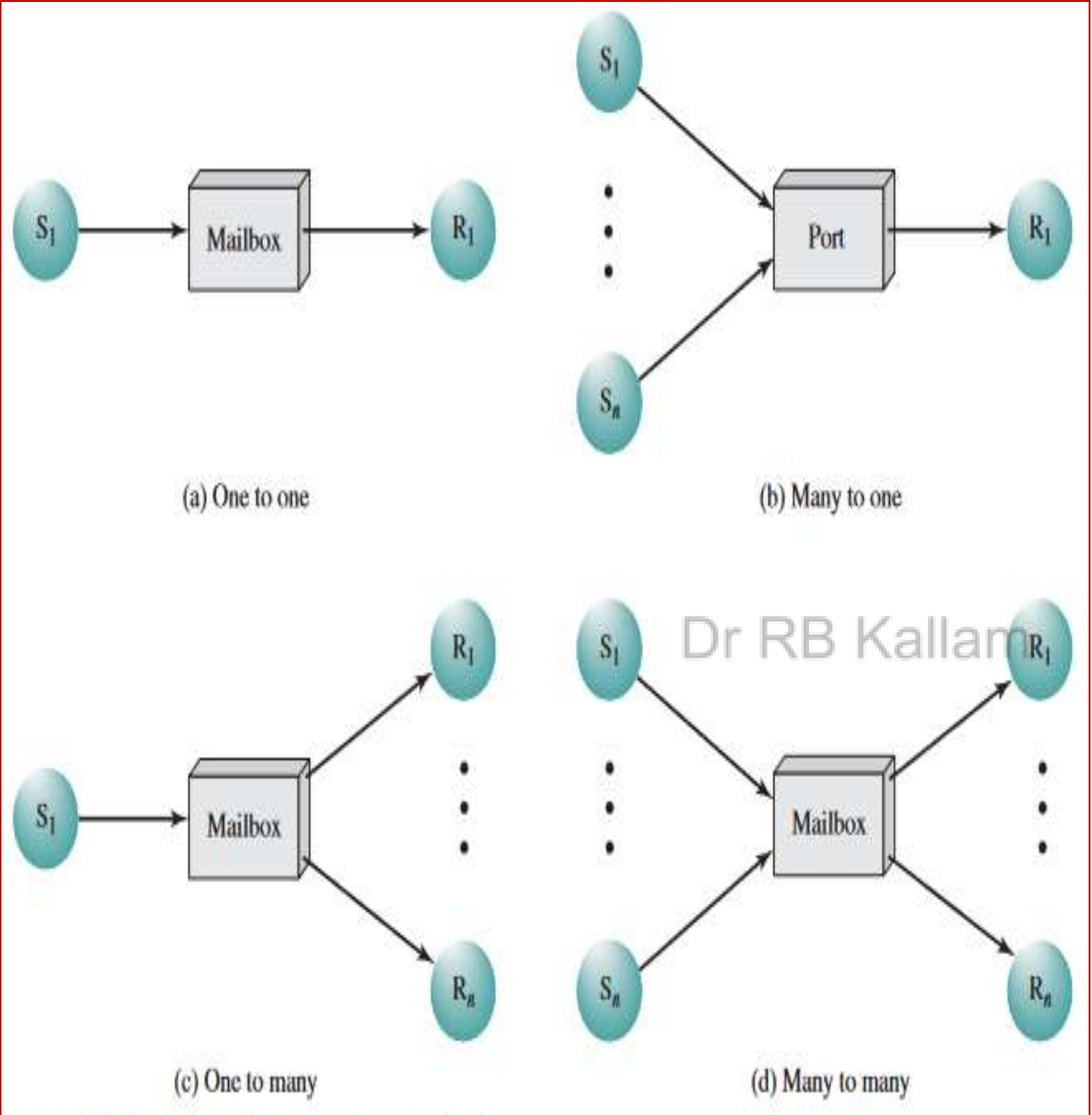


Figure 5.18 Indirect Process Communication

Source	Header
Destination	
Message Length	
Control Information	
Message Type	
Message Content	Body

Typical message format

Indirect process communication

Buffering:

- Buffering is used in direct and indirect communication. Messages exchanged by communicating processes reside in a temporary queue.
- Basically there are three ways that such a queue can be implemented:
  - **Zero capacity:** The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.
  - **Bounded capacity:** The queue has finite length n; thus, at most n messages can reside in it. If the link is full, the sender must block until space is available in the queue.
  - **Unbounded capacity:** The queues length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks..

CPU Scheduler:

- **Preemptive Scheduling:**
  - The currently running process may be interrupted and moved to the ready state by the Operating System.
  - The decision to preempt may be performed when a new process arrives, when an interrupt occurs or periodically on the basis of clock interrupt.

- **Non Preemptive Scheduling:**
  - Once a process is in the running state, it continue to execute until it terminates or blocks itself to wait for I/O or by requesting some OS services.

Scheduling Criteria:

- CPU Utilization: We want to keep the CPU as busy as possible. CPU utilization range from 0 to 100 percent.
- Throughput: It is number of processes that are completed execution per unit time.
- Turnaround Time (Tr): It is an interval of time between the process submission and its completion *(Finishing time-arrival time)*

- Waiting Time: It is the sum of the time spent by the process waiting in the ready queue *(turnaround time – burst time).*
- Response time: It is the time from the submission of a request until the first response is produced.
- Service Time (Ts): Total CPU time required by a process.
- Normalized Turnaround time(Tr/Ts): It is the ratio of the turnaround time (Tr) to service time (Ts).

Scheduling Algorithms:

- FCFS
- SJF or SPN
- Priority
- Round Robin
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

Optimization Criteria

In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms. The criteria include the following:

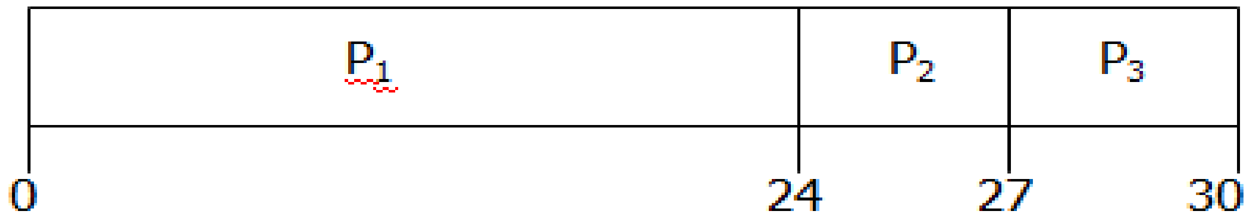
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

First-Come, First-Served (FCFS) Scheduling

- It is a non preemptive algorithm ,in this CPU is allocated to the processes based on FCFS.
- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
- Assume that the arrival time is '0' for all processes, calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and Average Wt.

Process	Burst Time
P_1	24
P_2	3
P_3	3

The Gantt Chart for the schedule is:

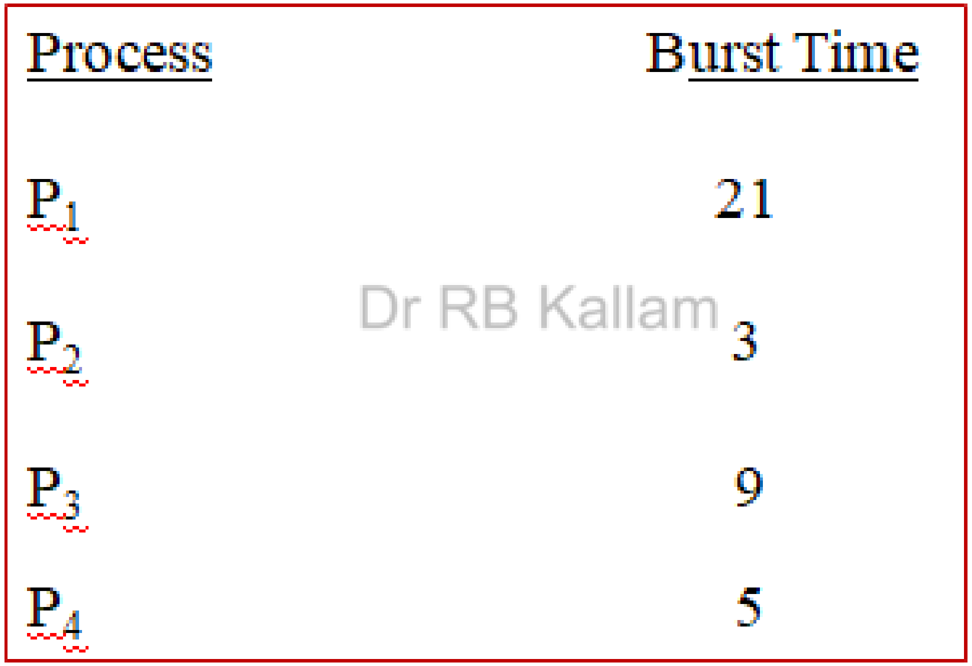


Process ID	Arrival Time (At)	Burst time (Bt)	Finishing Time (Ft)	Turnaround Time (Tr=Ft-At)	Waiting Time (Wt=Tr-Bt)
P1	0	24	24	24	0
P2	0	3	27	27	24
P3	0	3	30	30	27
				ATr=81/3=27	AWt=51/3=17

Note: ATr- Average Turnaround Time, AWt- Average Waiting Time

- Advantage: It is a simple scheduling algorithm among all.
- Disadvantage: The FCFS scheduling algorithm is non preemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O.

- Ex Q2: If the burst time for 4 processes are as given below, apply FCFS and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and Average Wt.



Process	Burst Time
P_1	21
P_2	3
P_3	9
P_4	5

Shortest-Job-First (SJF)or SPN Scheduling

- It is a *non preemptive policy*, in which the process with the shortest expected processing time is selected for next.

Disadvantage:

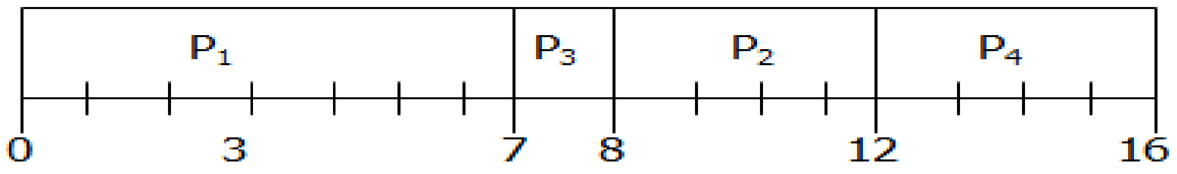
- It may leads to the starvation of longer processes.
- Starvation: A condition in which a process is indefinitely delayed because other processes are always given preferences.

Example of Non-Preemptive SJF

Q1: If the arrival time and burst time for 4 processes are as given below, apply SJF and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and Average Wt.

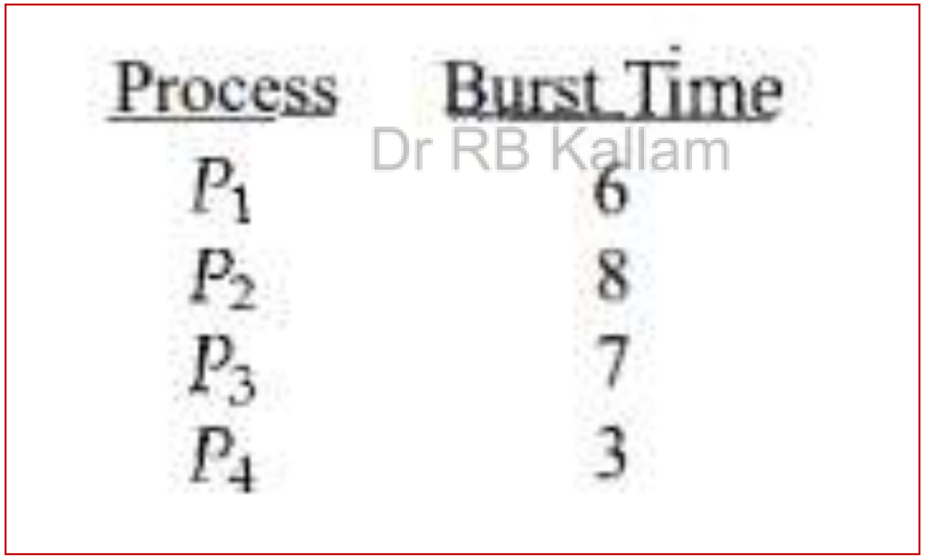
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

The Gantt Chart for the schedule is:



Process ID	Arrival Time (At)	Burst time (Bt)	Finishing Time (Ft)	Turnaround Time (Tr=Ft-At)	Waiting Time (Wt=Tr-Bt)
P1	0	7	7	7	0
P2	2	4	12	10	6
P3	4	1	8	4	3
P4	5	4	16	11	7
				ATr=32/4=8	AWt=16/4=4

Ex Q2: If the burst time for 4 processes are as given below, apply SJF and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and Average Wt.



Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Shortest-Remaining-Time-First (SRTF) or Shortest-Job-First (SJF) with Preemption

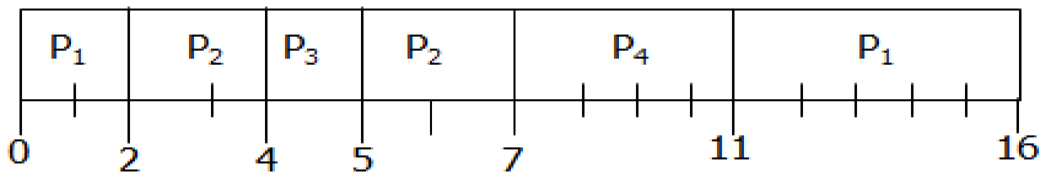
If a new process arrives with CPU burst length less than remaining time of current executing process, then the current process is preempted.

Example of Preemptive SJF/ SRTF:

Q: If the arrival time and burst time for 4 processes are as given below, apply SRTF and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and AverageWt.

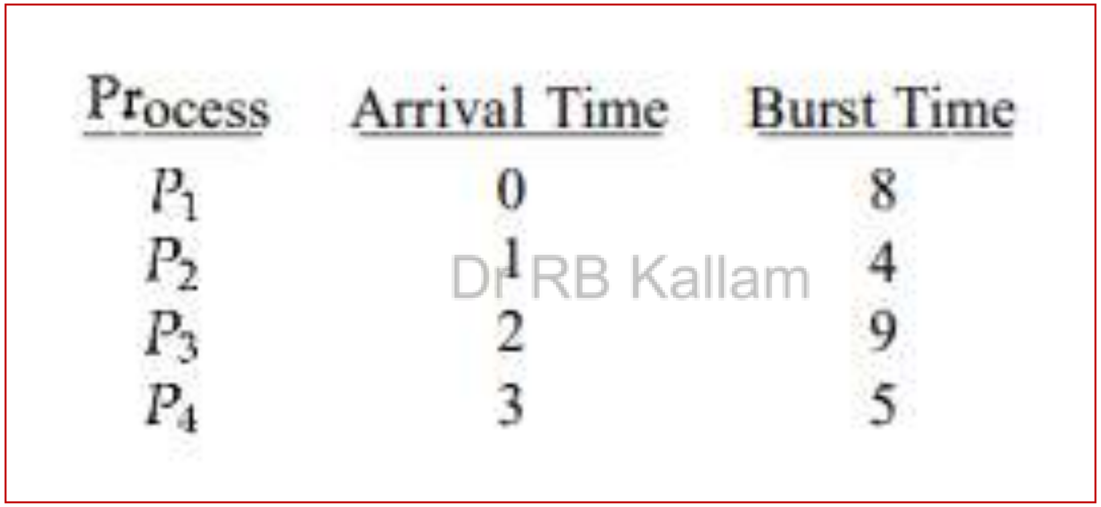
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

The Gantt Chart for the schedule is:



Process ID	Arrival Time (At)	Burst time (Bt)	Finishing Time (Ft)	Turnaround Time (Tr=Ft-At)	Waiting Time (Wt=Tr-Bt)
P1	0	7	16	16	9
P2	2	4	7	5	1
P3	4	1	5	1	0
P4	5	4	11	6	2
				ATr=28/4=7	AWt=12/4=3

Ex Q2: If the arrival time and burst time for 4 processes are as given below, apply SRTF and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and AverageWt.



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

Priority Scheduling

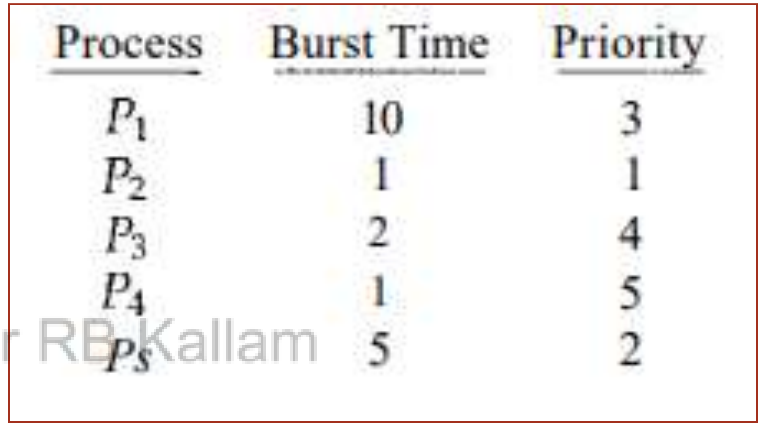
- It can be either Preemptive or non preemptive
- A priority number (integer) is associated with each process
- Non preemptive :The CPU is allocated to the process with the highest priority (smallest integer $\equiv$ highest priority).

Disadvantage:

Starvation – low priority processes may never execute if we continue to accept highest priority processes into the queue.

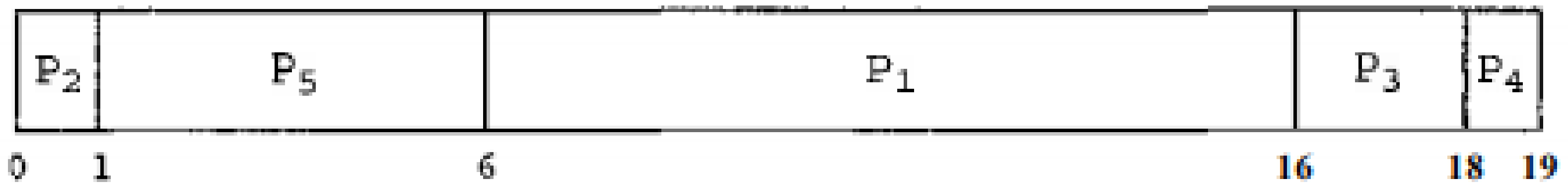
Example of Non Preemptive Priority scheduling:

Q: If the burst time and priority for 5 processes are as given below, apply Non Preemptive Priority scheduling and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and AverageWt.



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:



Process ID	Burst time (Bt)	Priority	Finishing Time (Ft)	Turnaround Time (Tr=Ft-At)	Waiting Time (Wt=Tr-Bt)
P1	10	3	16	16	6
P2	1	1	1	1	0
P3	2	4	18	18	16
P4	1	5	19	19	18
P5	5	2	6	6	1
				ATr=60/5=12	AWt=41/5=8.2

- Ex Q2: If the burst time and priority for 5 processes are as given below, apply Non Preemptive Priority scheduling and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and Average Wt.

Process ID	Burst Time	Priority
P1	21	2
P2	3	1
P3	9	4
P4	5	3
P5	7	5

Preemptive Priority Scheduling

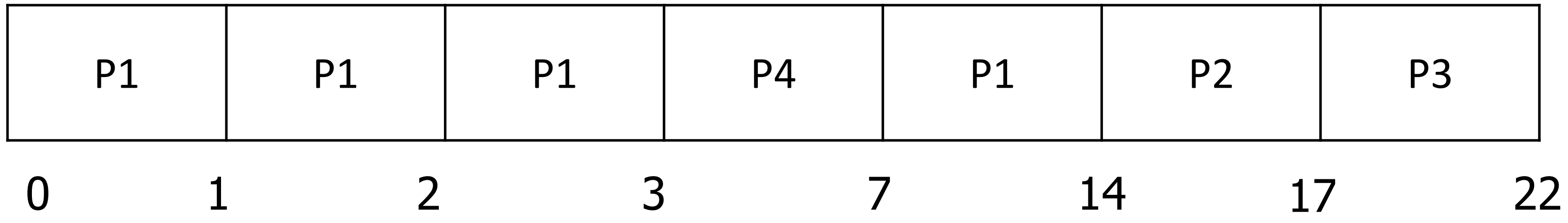
- Preemptive Priority Scheduling: a preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

Example for Preemptive Priority scheduling:

Q: If the arrival time, burst time and priority for 4 processes are as given below, apply Preemptive Priority scheduling and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and AverageWt.

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

The Gantt Chart for the schedule is:



Process ID	Arrival Time	Burst time (Bt)	Priority	Finishing Time (Ft)	Turnaround Time (Tr=Ft-At)	Waiting Time (Wt=Tr-Bt)
P1	0	10	2	14	14	4
P2	1	3	3	17	16	13
P3	2	5	4	22	20	15
P4	3	4	1	7	4	0
					ATr=54/4=13.5	Awt=32/4 =8

- Ex Q2: If the Arrival time, burst time and priority for 5 processes are as given below, apply Preemptive Priority scheduling and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and Average Wt.

Process ID	Arrival Time	Burst Time	Priority
P1	0	21	2
P2	2	3	1
P3	4	9	4
P4	6	5	3
P5	8	7	5

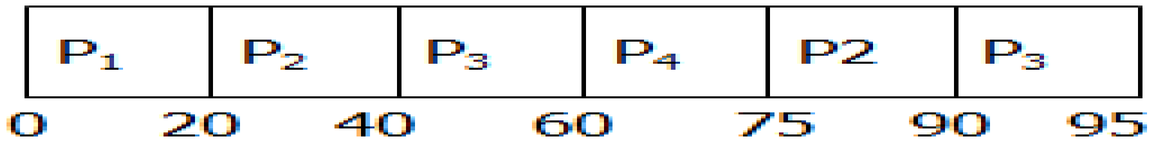
Round Robin (RR)

- It is a Preemptive scheduling algorithm.
- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- The ready queue is treated as a circular queue.
- The scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.
- It improves the response time, but may leads to the wastage of CPU time with context switching.

- Ex Q1: If the burst time for 4 processes are as given below, apply Round robin scheduling algorithm (with Quantum =20) and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and Average Wt.

Process	Burst Time
P_1	20
P_2	35
P_3	25
P_4	15

The Gantt Chart for the schedule is:



Process ID	Burst time (Bt)	Finishing Time (Ft)	Turnaround Time (Tr=Ft-At)	Waiting Time (Wt=Tr-Bt)
P1	20	20	20	0
P2	35	90	90	55
P3	25	95	95	70
P4	15	75	75	60
			ATr=280/4=70	AWt=185/4=46.25

- Ex Q2: If the burst time for 5 processes are as given below, apply Round robin scheduling algorithm (with Quantum =5) and calculate Turnaround time (Tr), Waiting time (Wt), Average Tr and Average Wt.

Process ID	Burst Time
P1	21
P2	3
P3	9
P4	5
P5	7

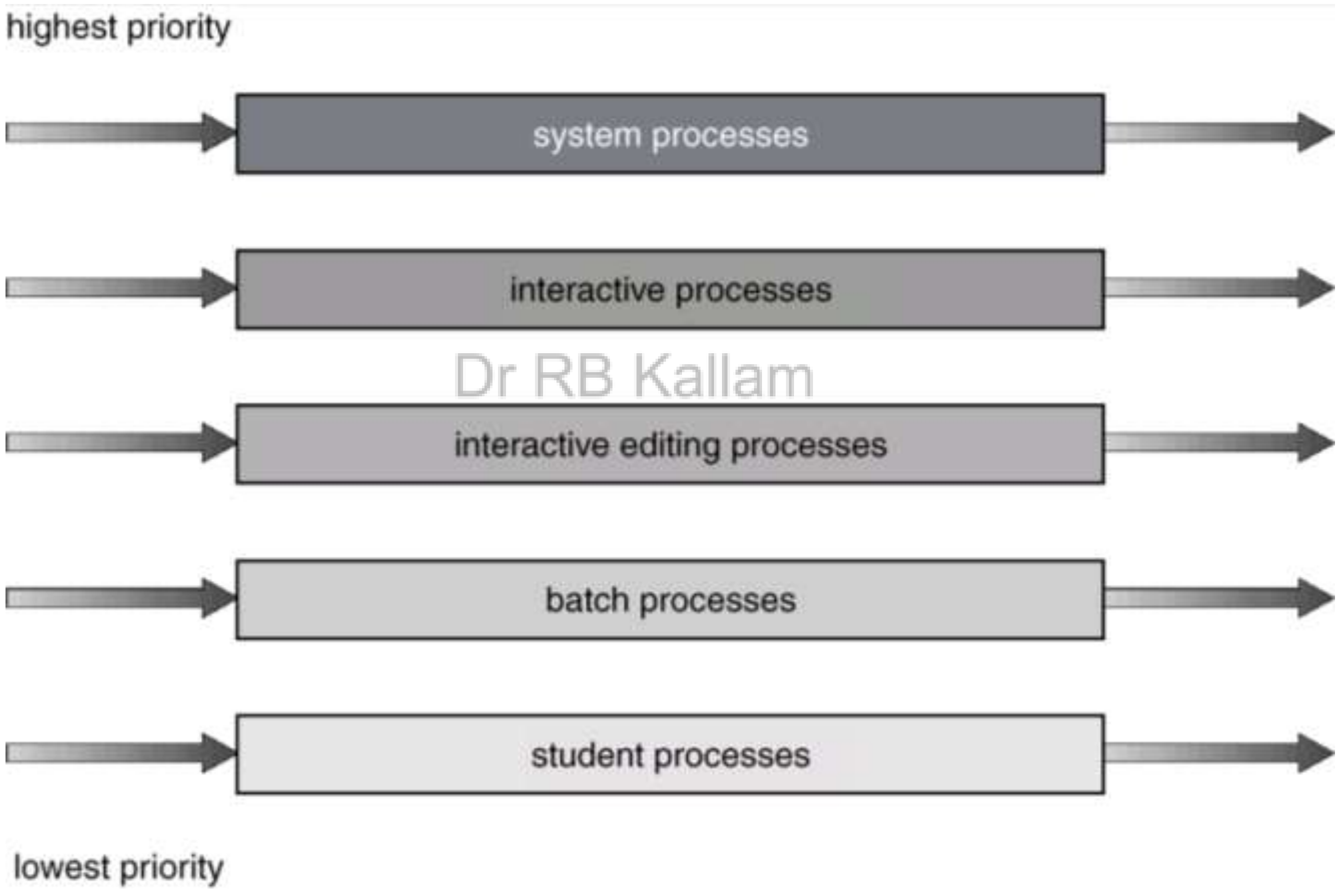
Ex Q3: Consider there are 5 processes and their arrival time and service times are given in the table. calculate the finishing time, Turnaround time, waiting time, average Tr and average Wt using FCFS, SPN, SRTF, RR algorithms.

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Multilevel Queue

- Ready queue is partitioned into separate queues based on the requirements.
- Each queue has its own scheduling algorithm,
  Ex: RR, FCFS, Priority, etc
- Scheduling must be done between the queues.

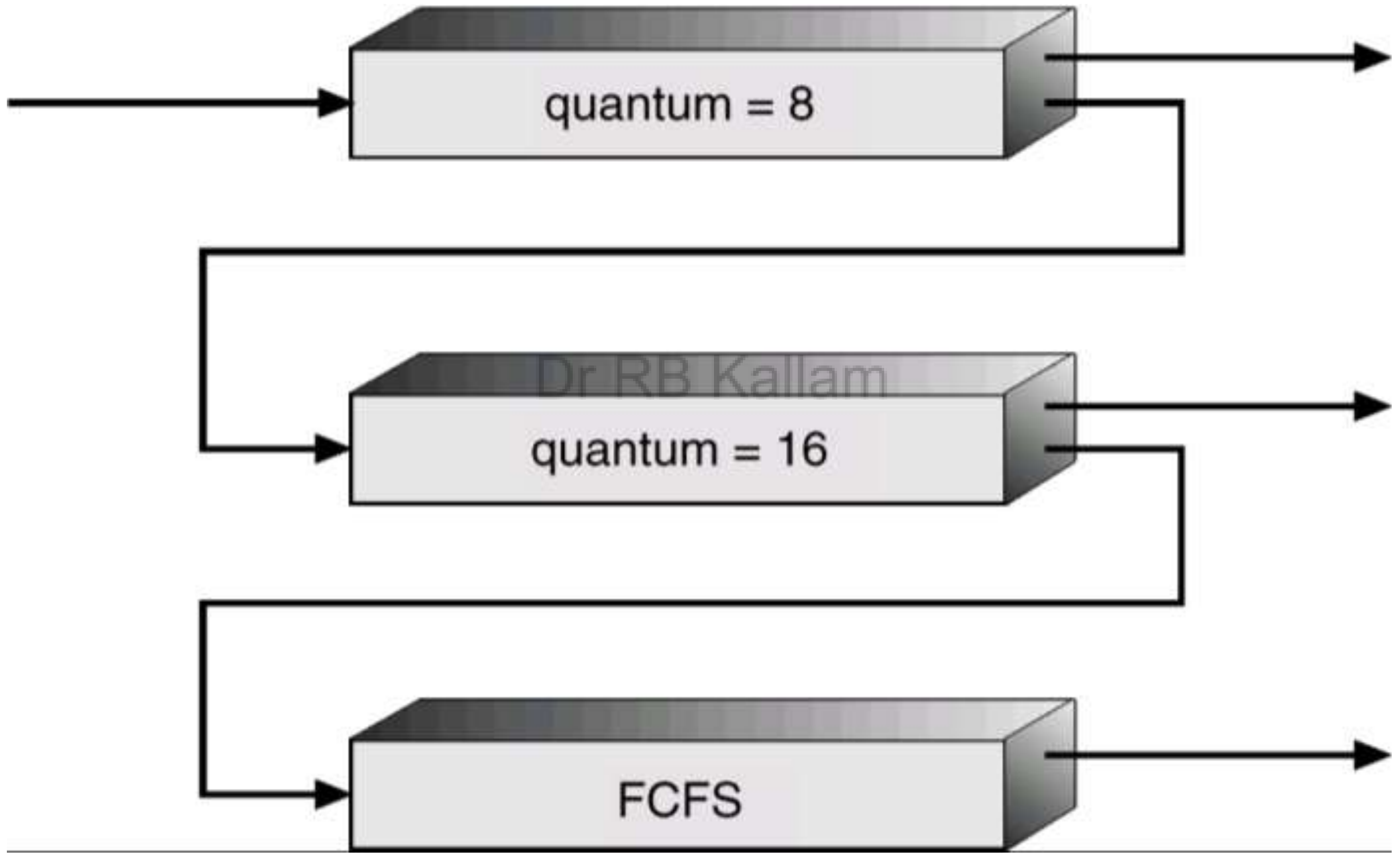
Multilevel Queue Scheduling



Multilevel Feedback Queue

- A process can move between the various queues
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process to higher priority queue.
  - method used to determine when to demote a process to lower priority queue.
  - method used to determine which queue a process will enter when that process needs service

Multilevel Feedback Queues



Multiple-Processor Scheduling

- CPU scheduling is more complex when multiple CPUs are available.
- A multiprocessor systems typically has two or more homogeneous processors
- *Major advantage is Load sharing* : In Symmetric multiprocessing systems, it is necessary to keep the workload balanced among all processors to fully utilize the available processors

There are Two types of load sharing:

- Push migration - a specific task periodically checks the load on each processor and rebalances the load if necessary .
- Pull migration - an idle processor can pull waiting tasks from a busy processor

Approaches to Multiple-Processor Scheduling

- *Asymmetric multiprocessing* – In this approach 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. It is simple because only one processor accesses the system data structures, reducing the need for data sharing.
- Symmetric Multiprocessing (SMP): In this each processor is self scheduling. all processes may be in common ready queue or each process may have its own private queue of ready processes.

Processor Affinity: It is a method of avoiding process migration from one processor to other or allowing process to run only in a fixed processor.

- Soft affinity - OS will attempt to keep a process running on the same processor, however there is no guarantee.
- Hard affinity - OS will force a process to continue running on the same processor.

Real-Time Scheduling

- Real time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced".

These are two types:

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time.
- *Soft real-time* computing – In this timelines are not very strict, a minor variation is tolerated.