Process Management And Scheduling

Prepared By : Prof. Pooja Makawana
Assistant Professor
Department of I.T
Dharmsinh Desai University

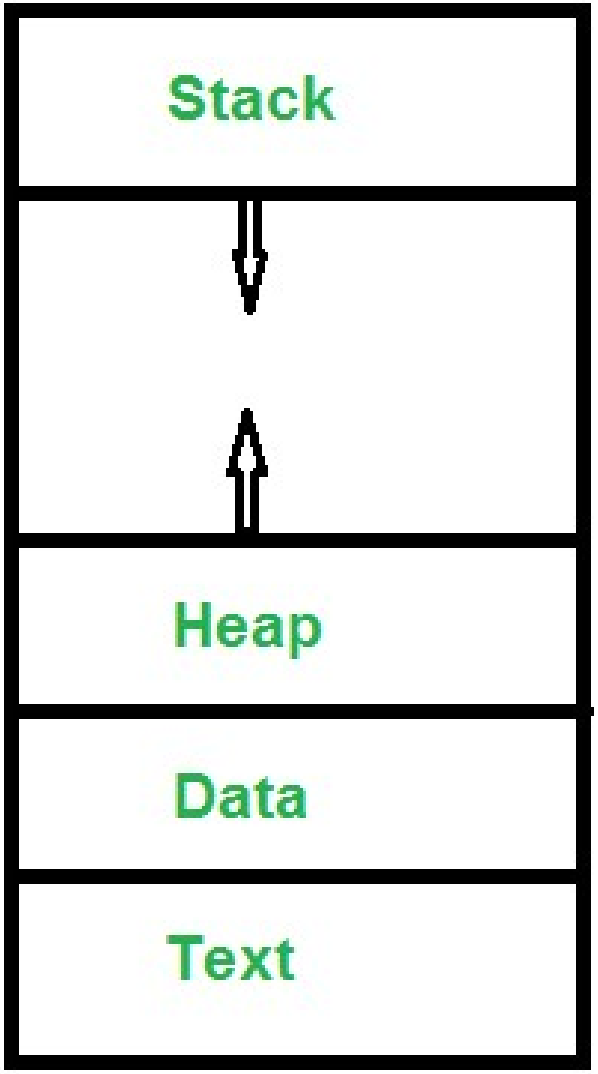
Outline

- Process Concept
- Process States
- Process Control Block
- Process Scheduling
- Schedulers
- Context switching
- Scheduling Criteria

process

- A process is sequential program in execution.
- A process defines the fundamental unit of computation for the computer.
- Components of process are :
  - **Object Program**: code to be executed
  - **Data**: Used for executing the program.
  - **Resources**: While Executing program it may require resources
  - **Status of the process execution**: Used for verifying the status of the process execution

What does a process look like in Memory?



Program to Process

- We write a program in e.g., Java.
- A compiler turns that program into an instruction list.
- The CPU interprets the instruction list (which is more a graph of basic blocks).

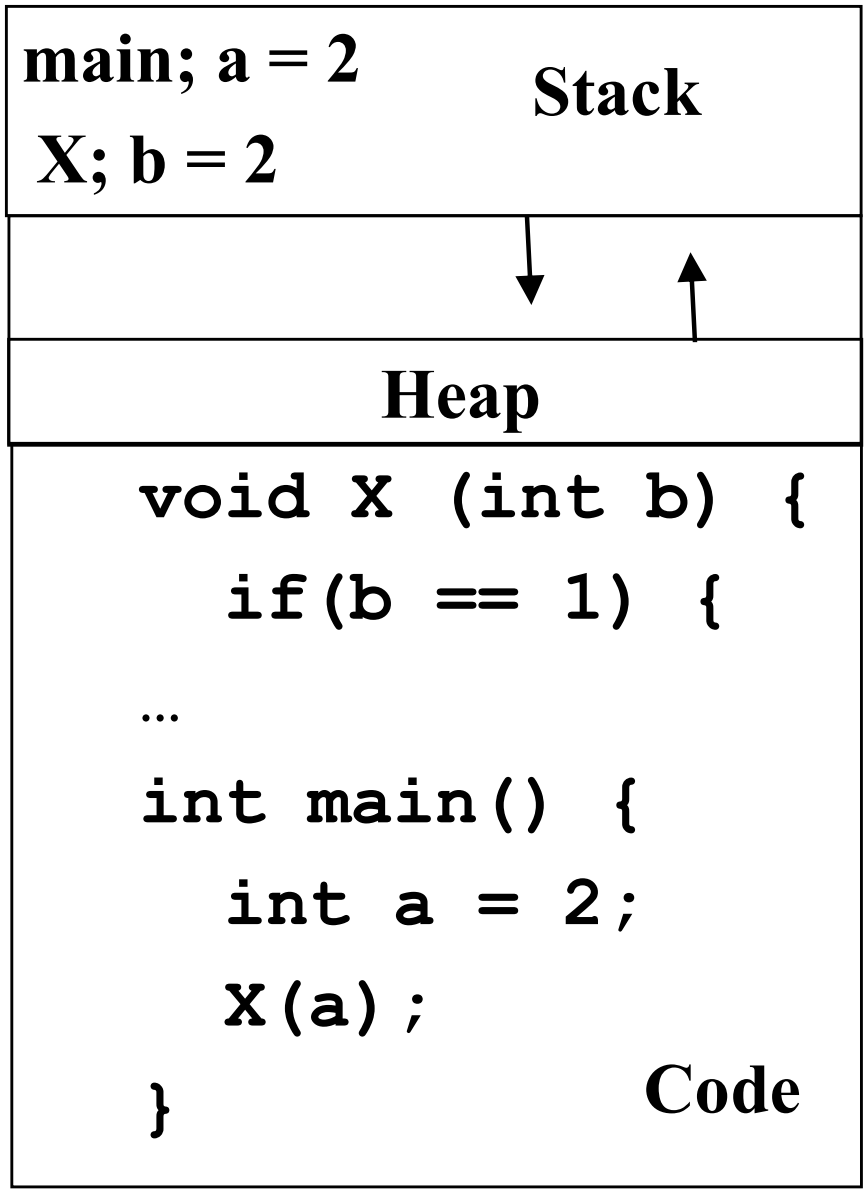
```
void X (int b) {
  if(b == 1) {
…
int main() {
  int a = 2;
  X(a);
}
```

Process in Memory

- Program to process.
  - What you wrote

```
void X (int b) {
  if(b == 1) {
…
int main() {
  int a = 2;
  X(a);
}
```

  - What must the OS track for a process?
  - What is in memory.



Program v/s process

Program	Process
Set of instruction/code.	When program is going for execution is called process.
Passive entity.	Active entity.
Reside in secondary memory.	Reside in main memory.
A program exists at single place in space and continues to exist.	Process exists in a limited span of time.
Consumes no resources.	Consumes required resources.
A program does not perform the action by itself.	Two or more processes could be executing the same program, each using their own data and resources.

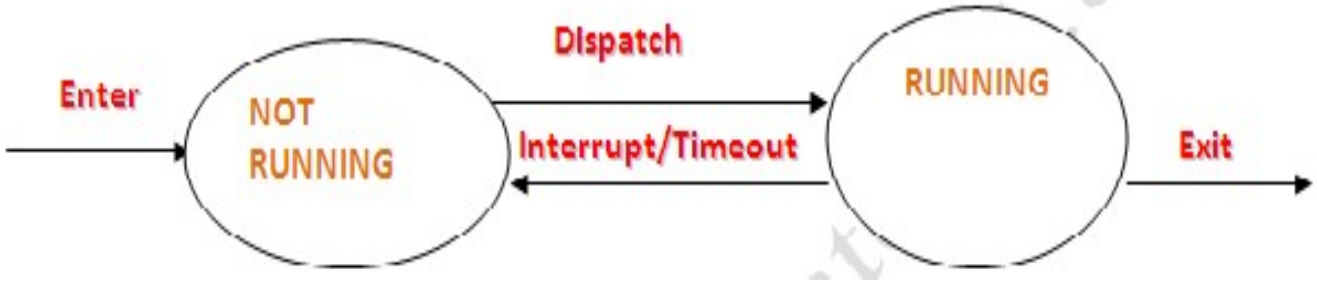
Process relationship

- **One to One**
  - A single execution of a sequential program

- **Many to One**
  - Many simultaneous executions of program
  - Execution of a concurrent program

Process State

- When process executes, it changes state.
- Process state is defined as the **current activity** of the process.
- **TWO STATE PROCESS MODEL :-**



- when O.S. is create the process and enter into the main memory for execution this time its state will be **NOT RUNNING** and these processes are waiting for the execution.
- When process is going for execution its state is called **RUNNING.**

5 state of Process

1. New
2. Ready
3. Running
4. Waiting
5. Terminated

Process States

- As a process executes, it changes *state*
- **New**:
  - A process that has just created by the O.S., but not submit into the Queue for execution.

- **Ready**:
  - The process is waiting to be assigned to a processor
  - When process in main memory and ready for the execution but execution is not start. This state of process is called **Ready State.**
  - During the execution if any interrupt generate by system or process is timeout than it will send back to Ready state.

Process State:

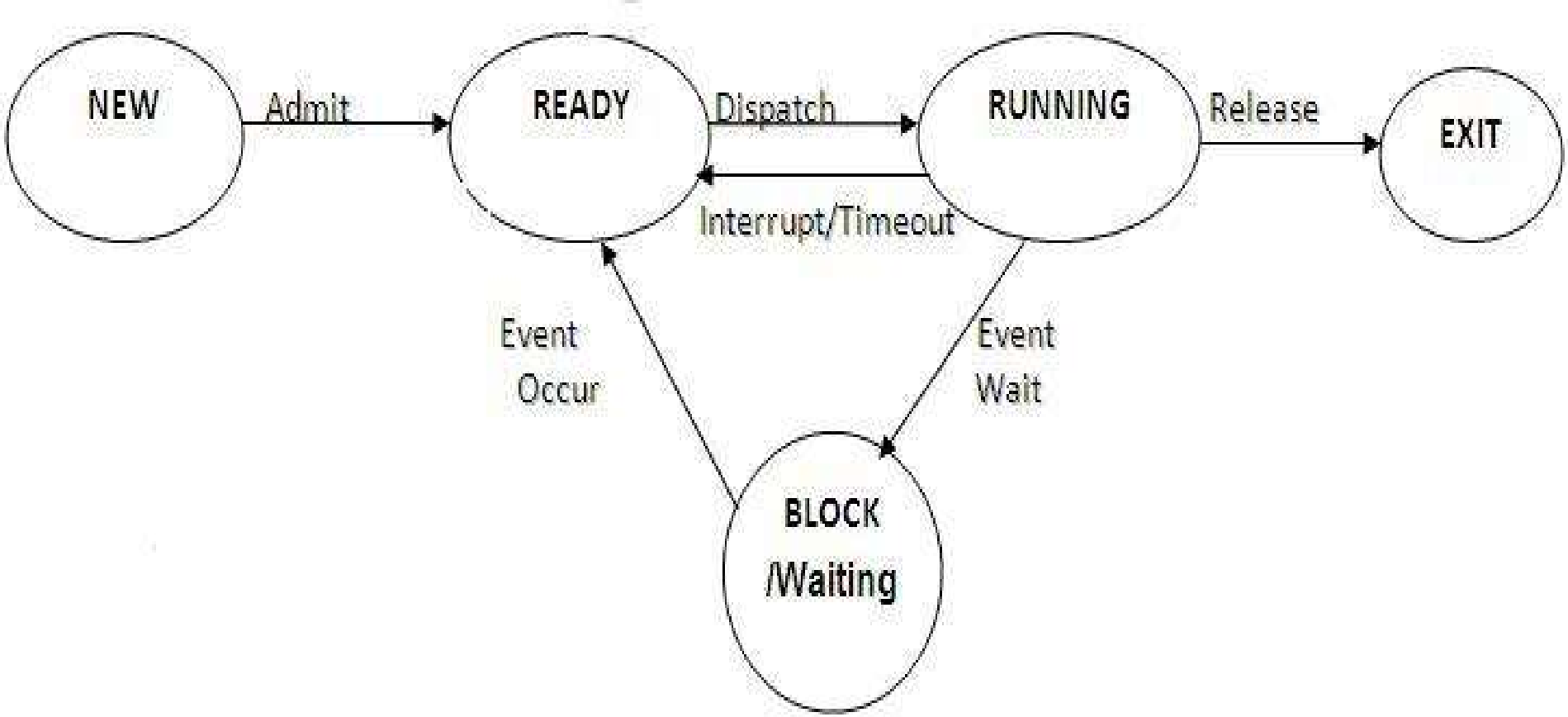
- **Running**:
  - When process is currently executing by processor then its state is called Running state.
  - Dispatcher will provide process to processor for execution.
  - At a time only one process can be executed

- **Block/Waiting**:
  - The process is waiting for some event to occur
  - when Process is executing and during execution if any event or I/O operation is in waiting for completion of process than process cannot execute until event occurs.
  - This process will declare as Block Process. This state is called **Block/Waiting state**.
  - When Event will occur than process will come into Ready state.

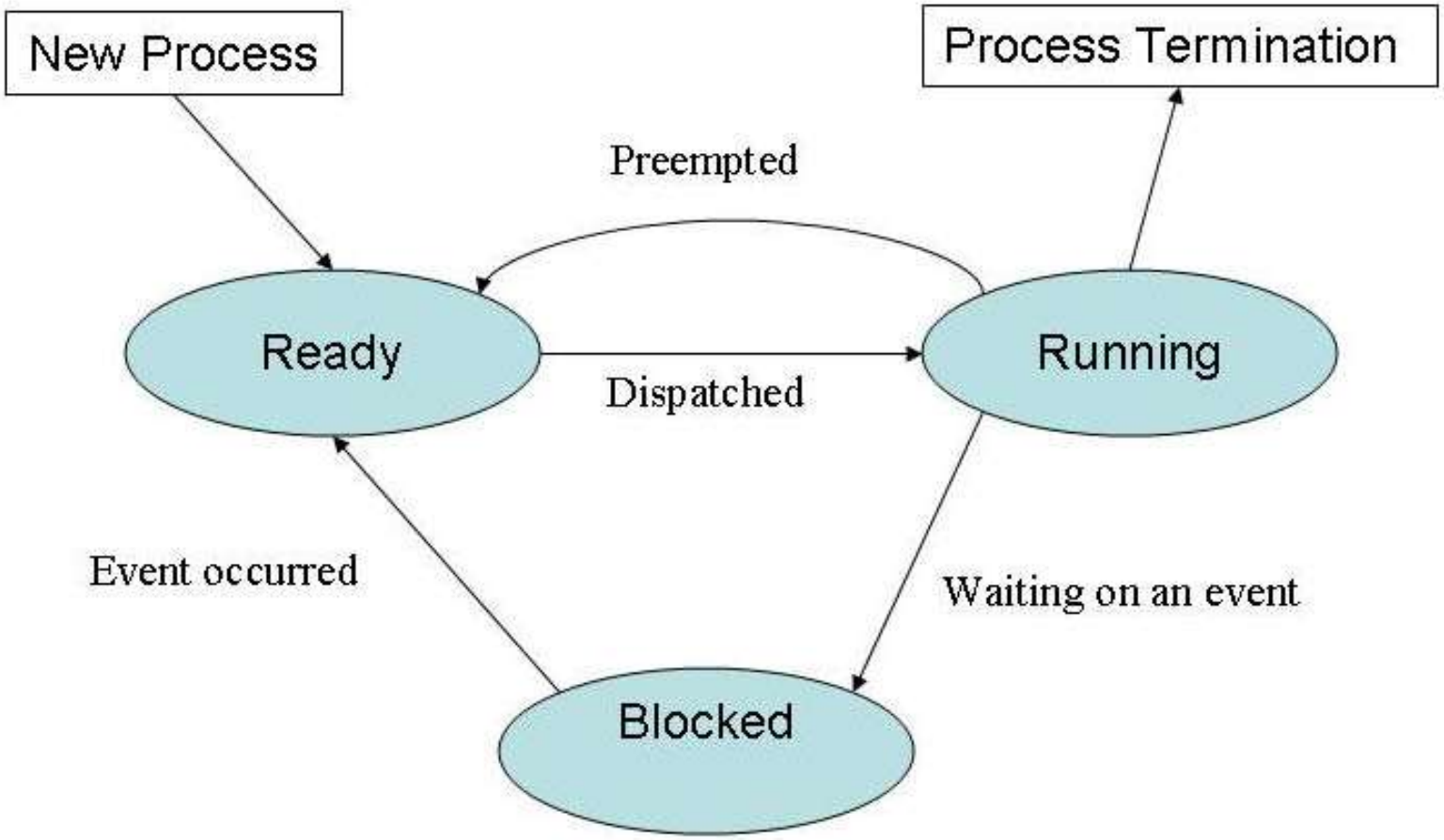
- **Terminated**:
  - The process has finished execution
  - process will release by processor

5 State process state model

It depicts the different states and the transitions between states that a process makes during execution.



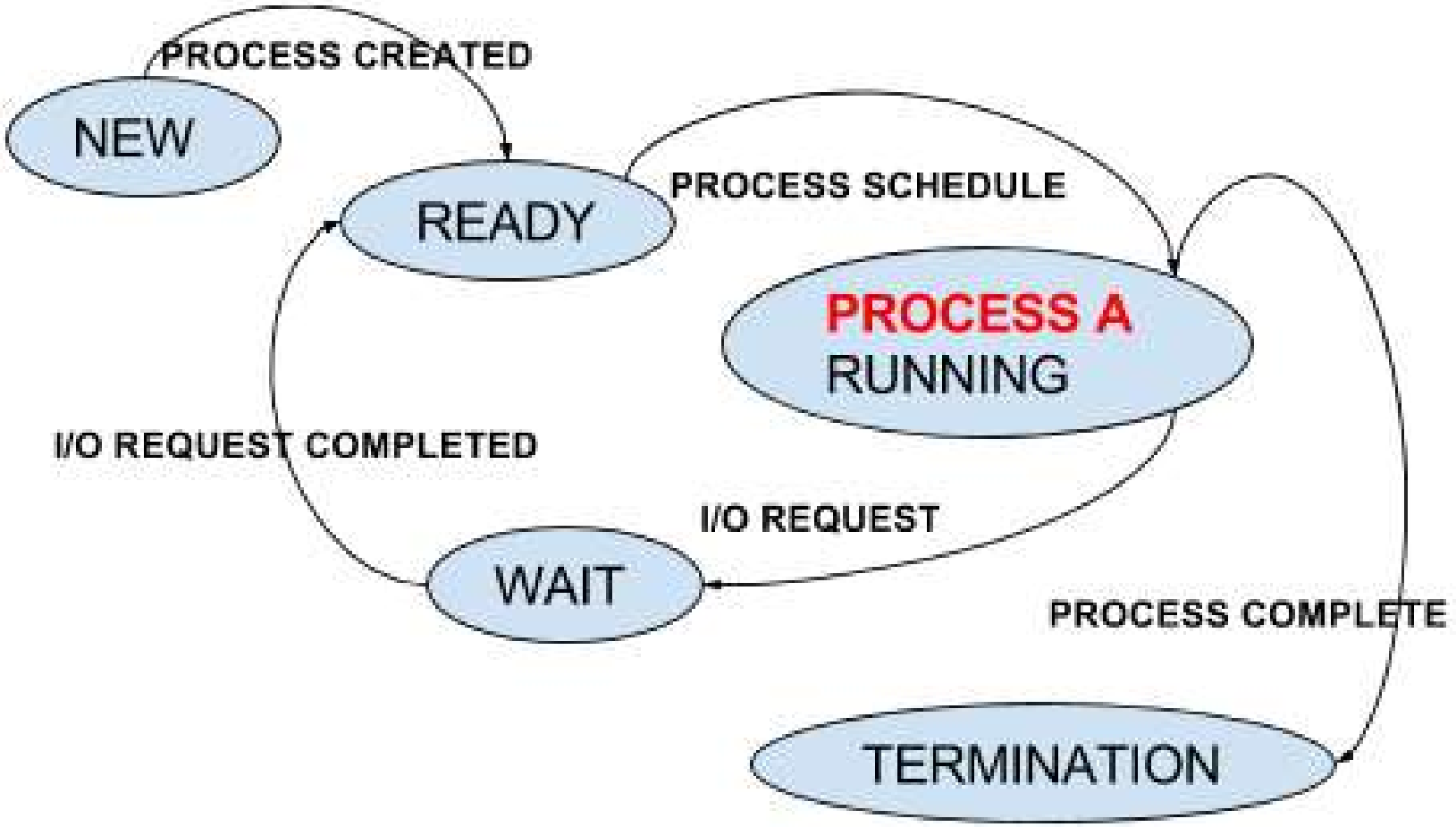
CPU preemption process state diagram



CPU Preemption Reason

1. Higher priority process initiate
2. Time quantum expire

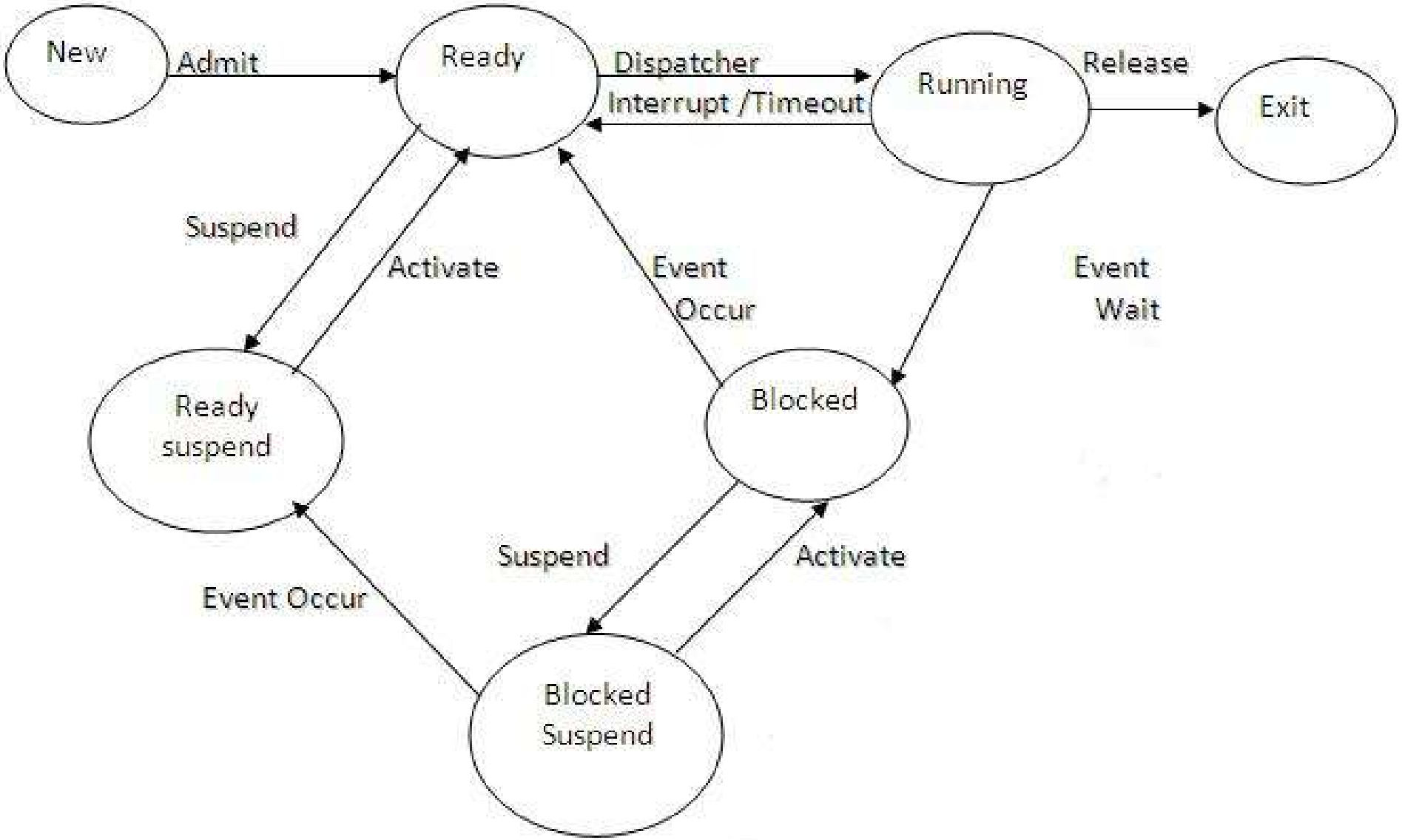
CPU Non-preemption process state diagram



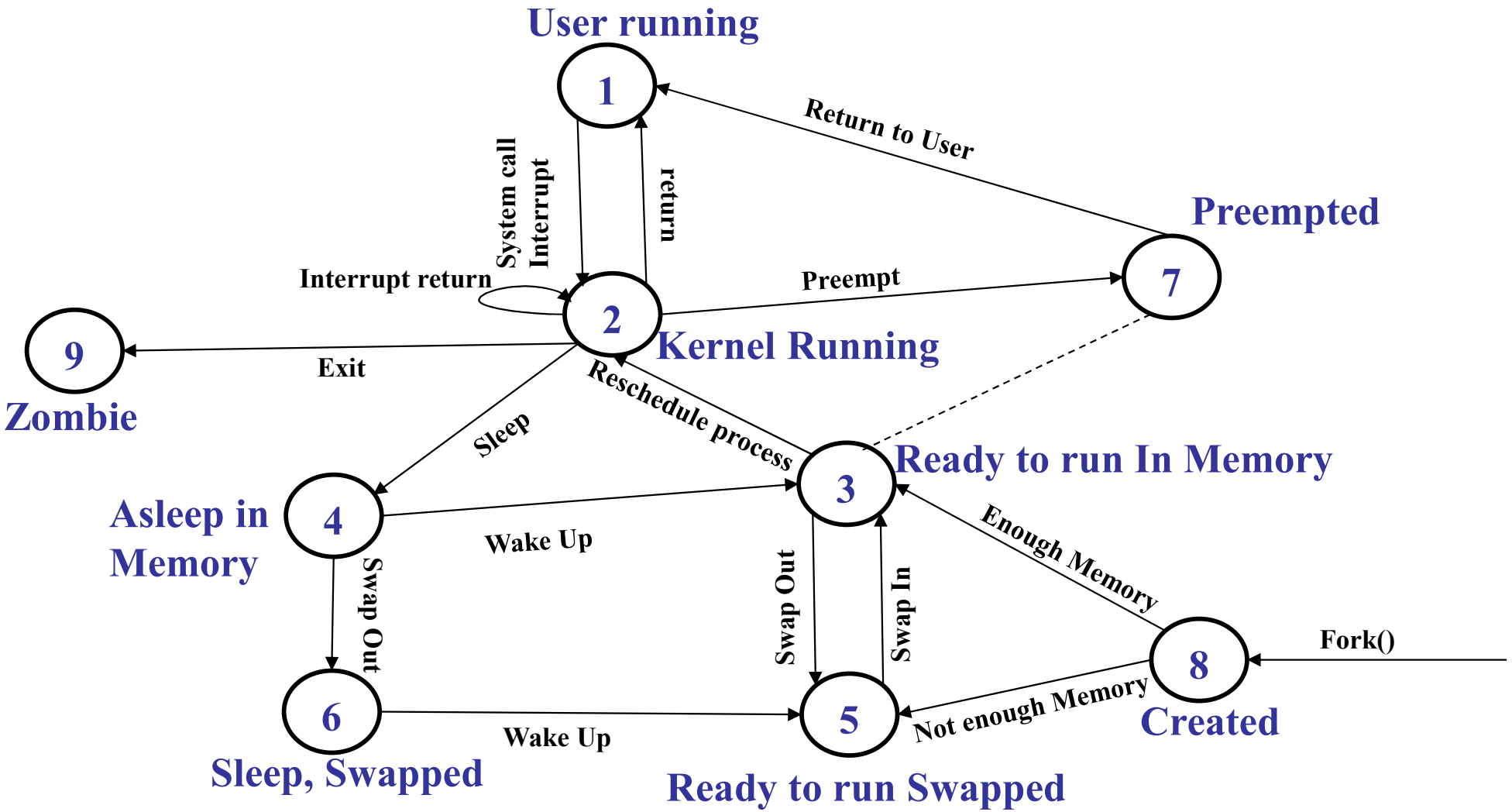
7 State process state model

- **Ready Suspended**
  - when Ready queue will be full, some process will be suspended temporarily , and store into the Ready suspended Queue.
  - And its state is called **ready suspended**.
  - When Ready queue will be free to store process than process from Ready suspend queue will store back into the Ready Queue.
- **Block/Wait suspended**
  - when block queue will be full, some process will be suspended temporarily, and store into the block suspended Queue.
  - Its state is called **block suspended.**
  - When block queue will be free to store process than process from block suspend queue store back into the block Queue.

7 State process state model



9 state process state diagram for UNIX



Special Cases

1. Blocked Suspend -----→ Block/Waiting
2. Running----------→ Ready Suspend
3. Ready----------→ Ready Suspend

Process Control Block (PCB)

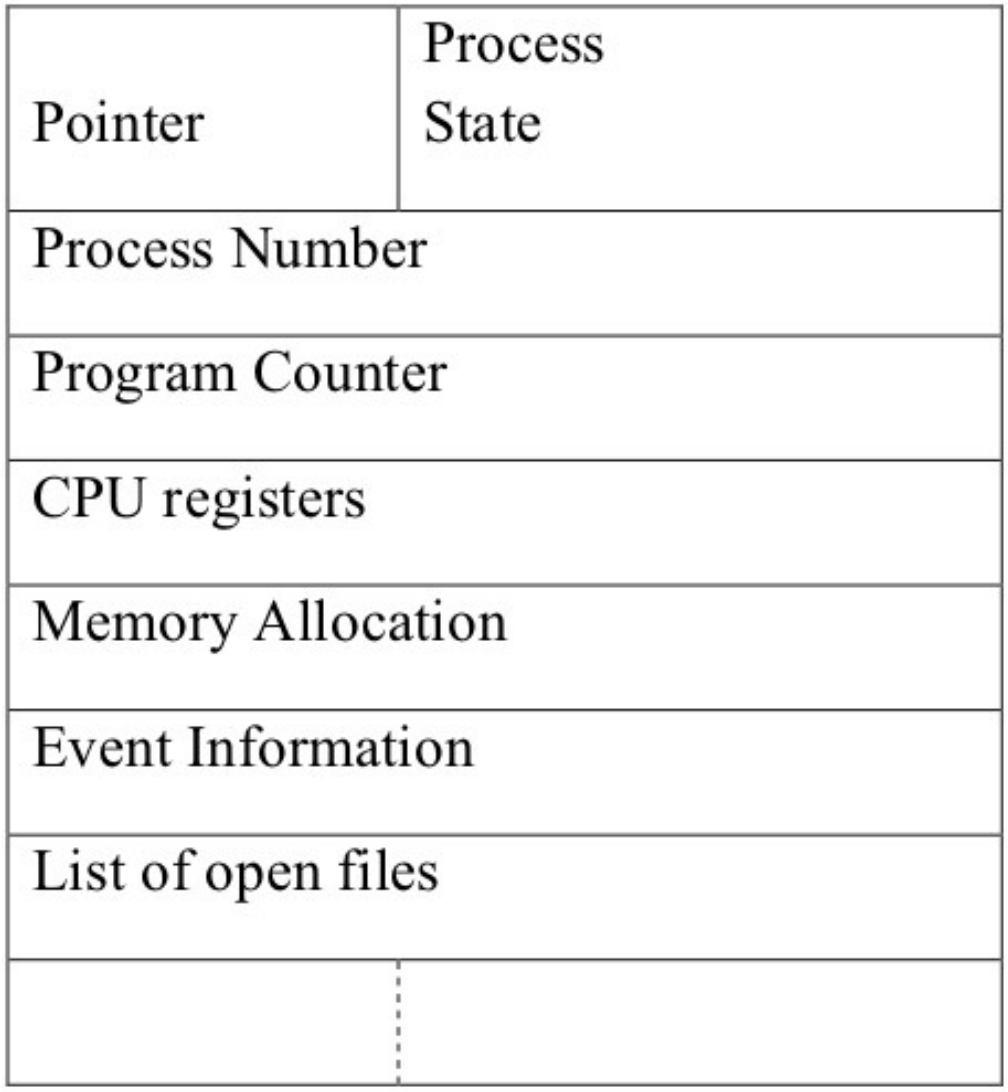
- Operating system provide the management of all the processes of system like
  - process creation, process termination, process execution and process scheduling etc.

- OS maintain the table to store information about process is called process control block (PCB).

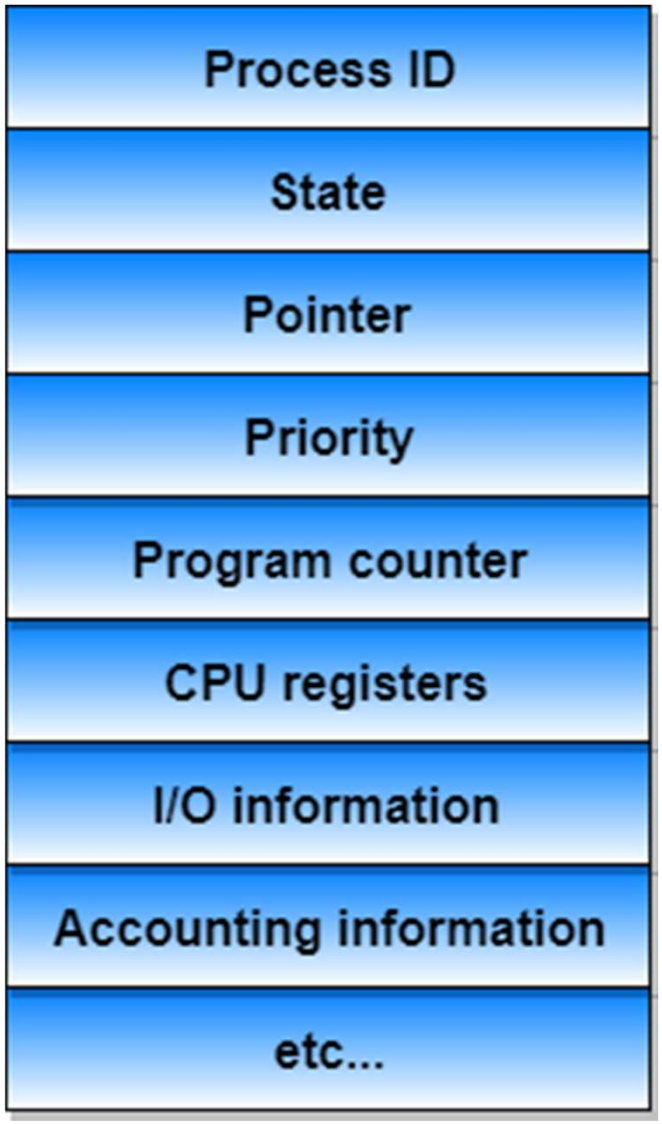
- It is also called data structure of process.(kernel data structure)

- **TCB(Task Control Block) or Process Descriptor or Process Object**

Process Control Block (PCB)



PCB


Process ID
State
Pointer
Priority
Program counter
CPU registers
I/O information
Accounting information
etc...

Process Control Block (PCB)

Information associated with each process

- **Pointer**
  - Pointer points to another process control block. Pointer is used for maintaining the scheduling list. (Info. Of processes are maintained as a **link list)**

- **Process state**
  - Process state may be new, ready, running, waiting and so on.

- **Program counter**
  - It indicates the address of the next instruction to be executed for this process.

Process Control Block (PCB)

- **CPU registers**
  - It indicates general purpose register, stack pointers, index registers and accumulators etc.
  - Number of register and type of register totally depends upon the computer architecture.
  - On an interrupt this state information along with program counter is saved to allow a process to continue correctly later.

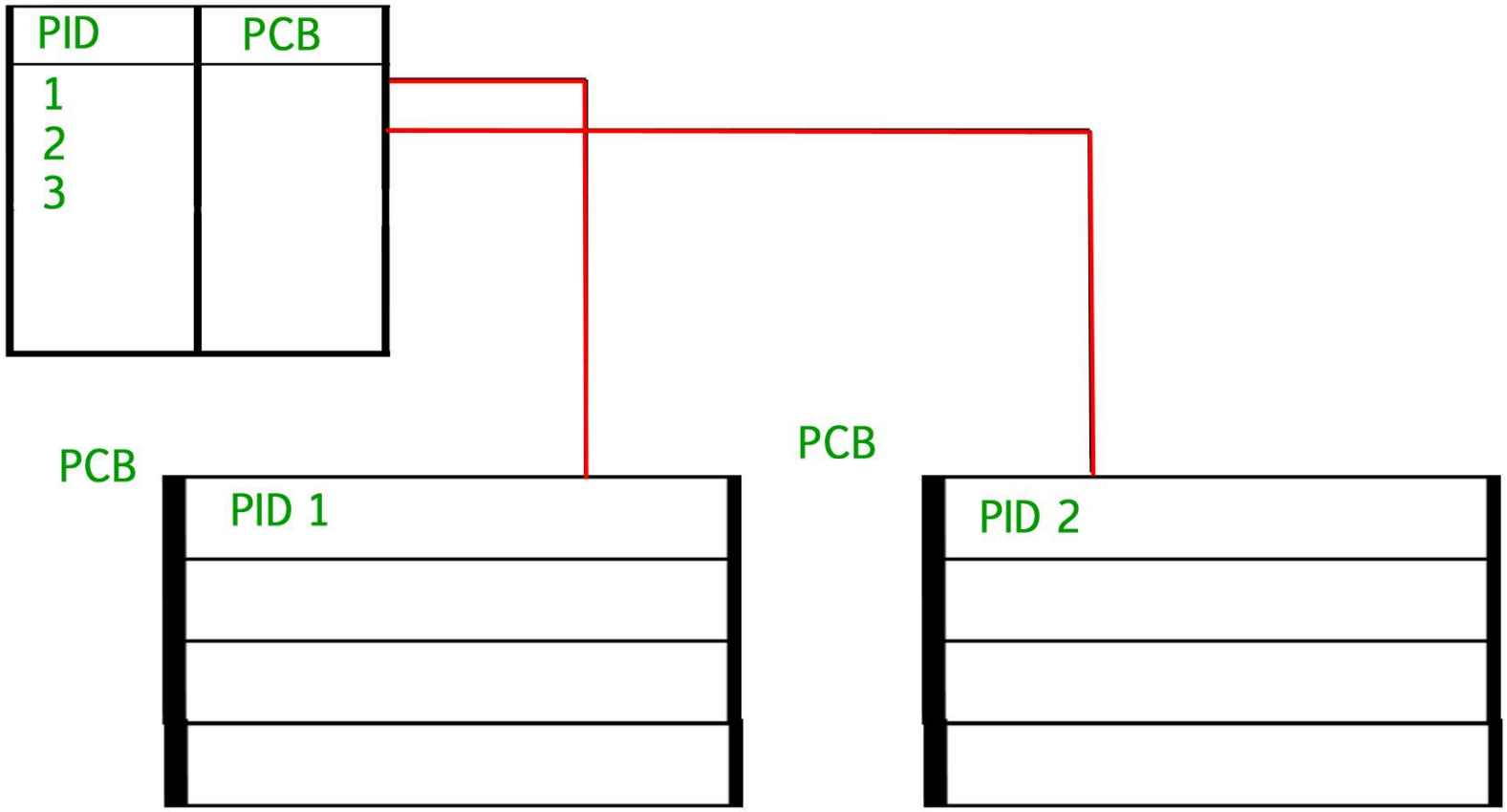
- **Memory Management Information :**
  - This information may include the value of base and limit register, page table/segment table.
  - This information is useful for deallocating the memory when the process terminates.

Process Control Block (PCB)

- **Accounting Information :**
  - This information includes the amount of CPU and real time used, time limits, job or process numbers, account numbers etc.

- Process control block also includes
  - The information about CPU scheduling,
  - I/O resource management,
  - File management information,
  - Priority and so on.

PCB



Process table and process control block

Process Scheduling

- In multiprogramming, more than one process and only one processor
  - therefore only one process can go for execution at a time And rest of the process has to wait for execution.
- OS is responsible for the scheduling of processes. And controlling the execution of process.

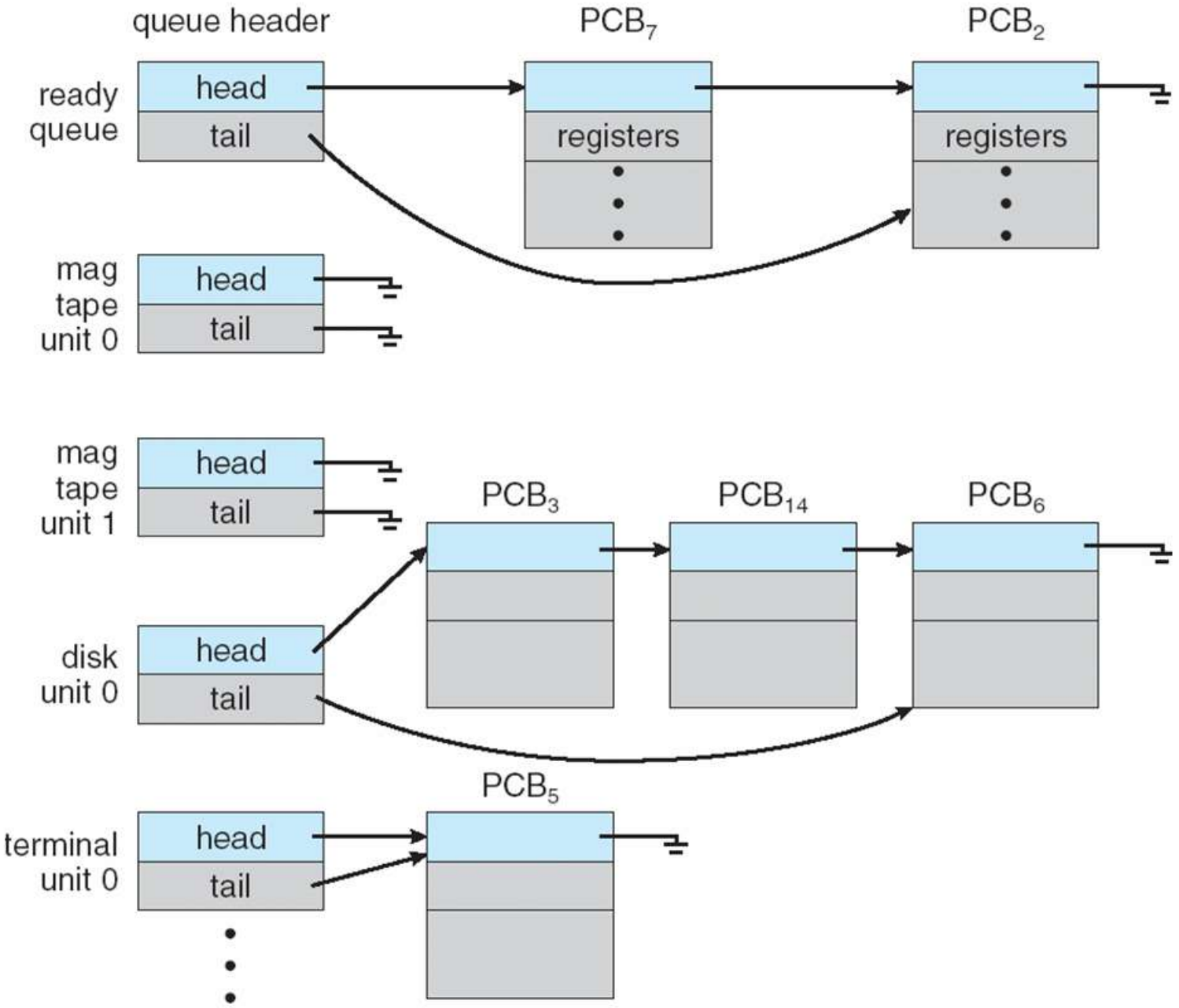
- Maximize CPU use, quickly switch processes onto CPU for time sharing

- **Process scheduler** selects among available processes for next execution on CPU

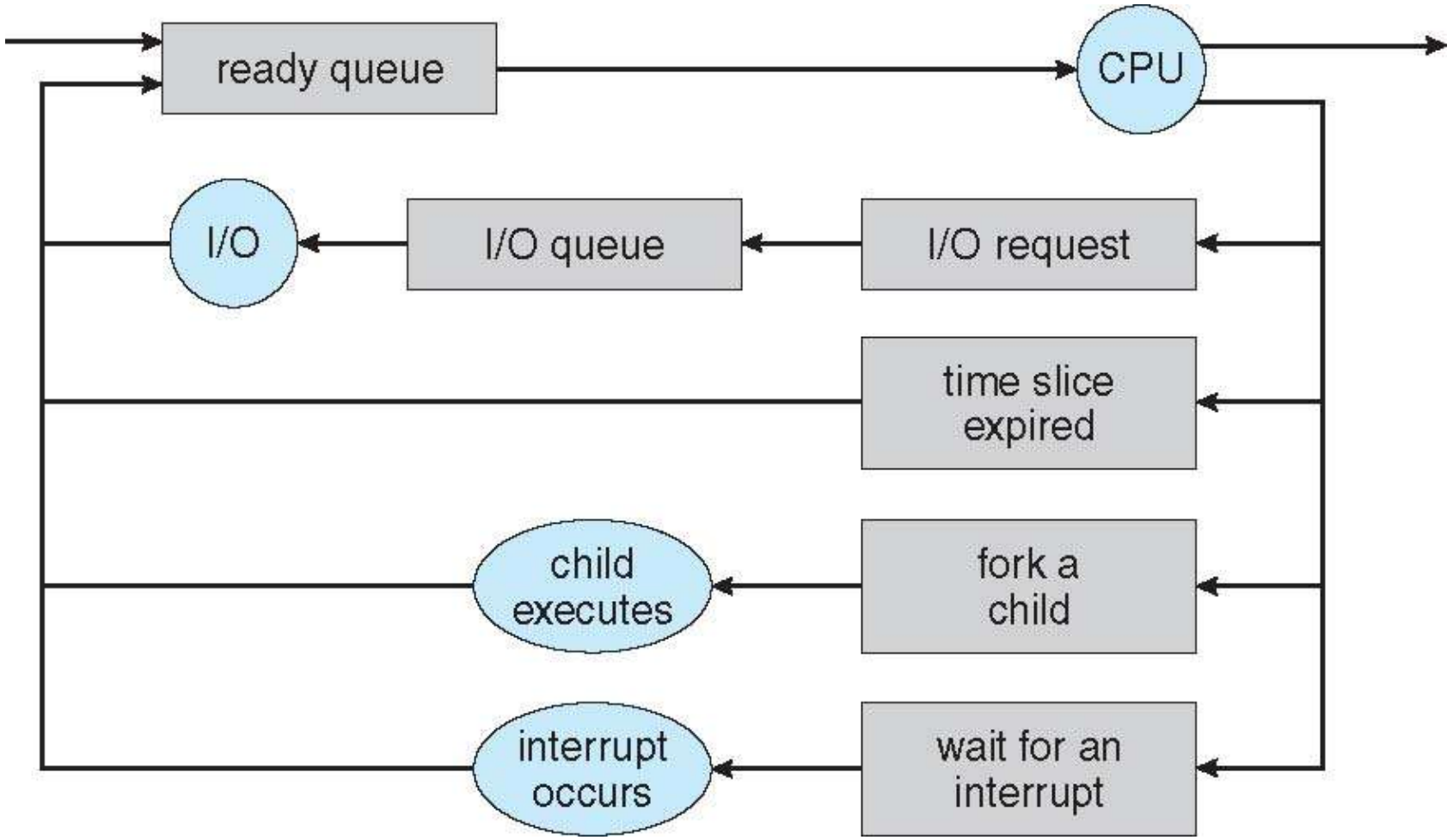
Process Scheduling

- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
    - Each device has its own queue.
  - Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



Queuing Diagram

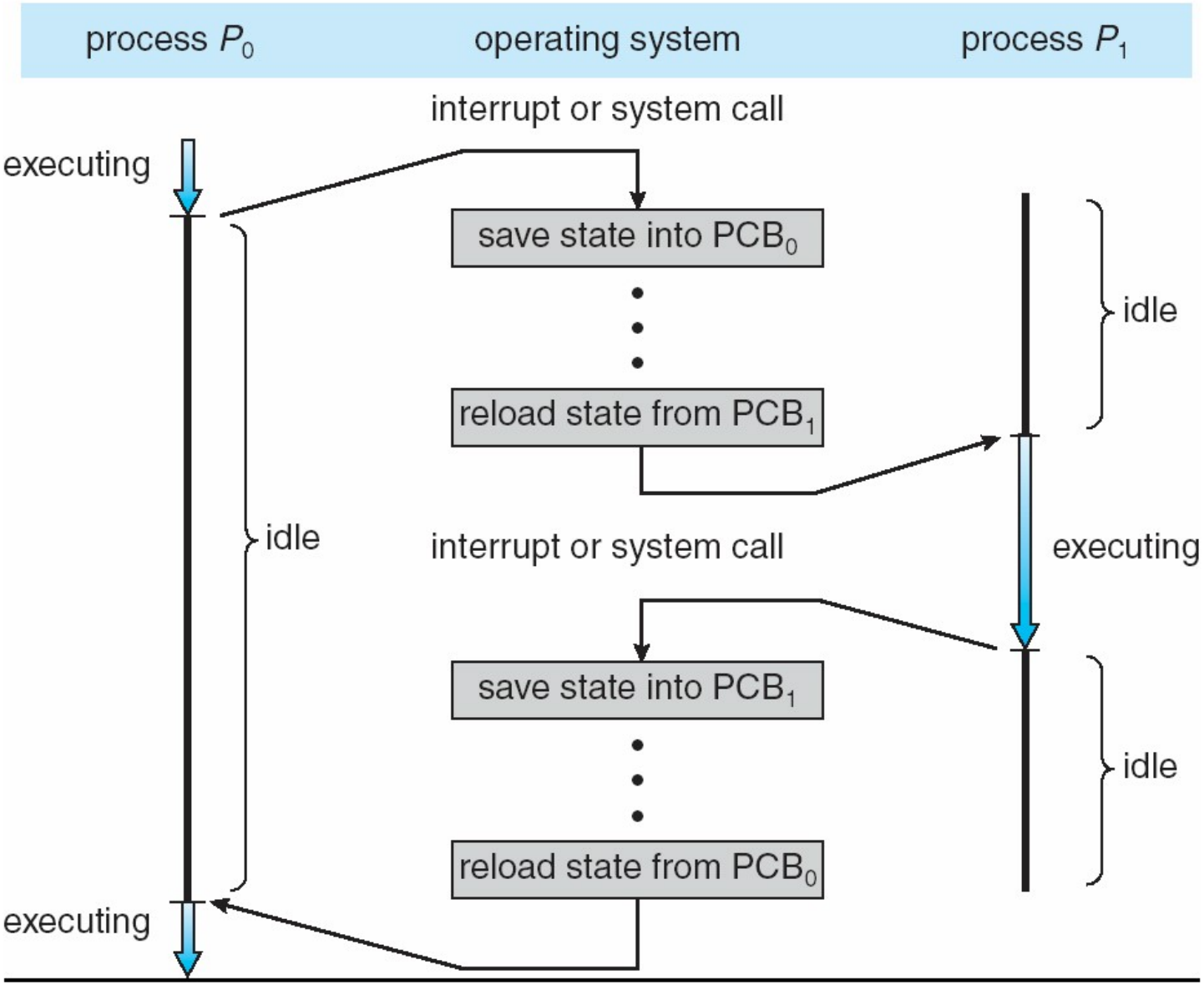
Process scheduling using queuing diagram

- A new process **is put in** the **ready queue** initially.
- It **waits in ready queue** till selected for execution.
- While executing the process, one of the several events could occur.
  - The process could issue an **I/O request** and then place in an I/O queue.
  - The process could **create new sub process** and waits for its termination.
  - The process could be removed forcibly from the CPU, as a result **of interrupt** and put back in the ready queue.

Context Switch

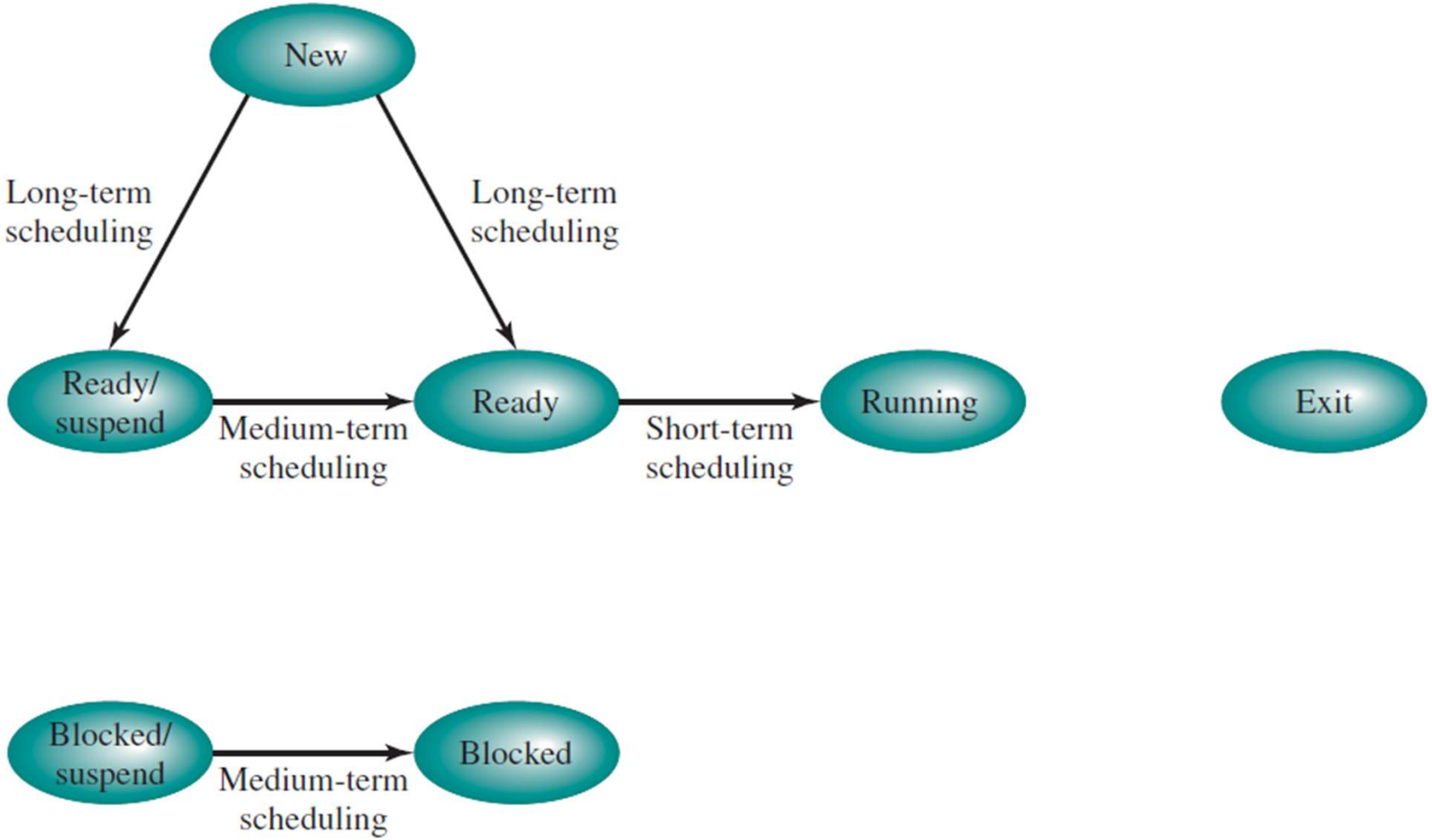
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**.
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB → longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU → multiple contexts loaded at once

CPU Switch From Process to



Schedulers

- **Responsible for selecting a process for scheduling/operating with the resources**
- **Three types of Schedulers :**
  - **Long-term scheduler** (or **job** scheduler) – selects which processes should be brought into the ready queue
  - **Short-term scheduler** (or **CPU** scheduler) – selects which process should be executed next and allocates CPU
    - Sometimes the only scheduler in a system
  - **Medium-term scheduler**
    - Intermediate level of scheduling



Scheduling Process and State Transitions

Long-term scheduler (or Job scheduler)

- It selects processes from the disk and loads them into memory for execution.
- LTS control the degree of multiprogramming (no. of processes in memory)
- It invoked only when a process leaves the system
  - So as to keep the degree of multiprogramming is stable,
  - Average rate of process creation = Average departure rate of process leaving the system.
  - So Execution frequency is less in comparison of others

Long-term scheduler

- In time sharing system LTS is not used
  - because in this type of system, process directly put into the ready queue.

- **LTS** needs to select good mix of **CPU bound and I/O bound processes** so as to avoid the ready queue being empty with no work for STS or **I/O queue** to be empty and devices not being utilized.

- **CPU Bound process**
  - It generates minimal or very few I/O requests and does more computation.
- **I/O Bound process**
  - It does more of I/O than computation.

Short-term scheduler

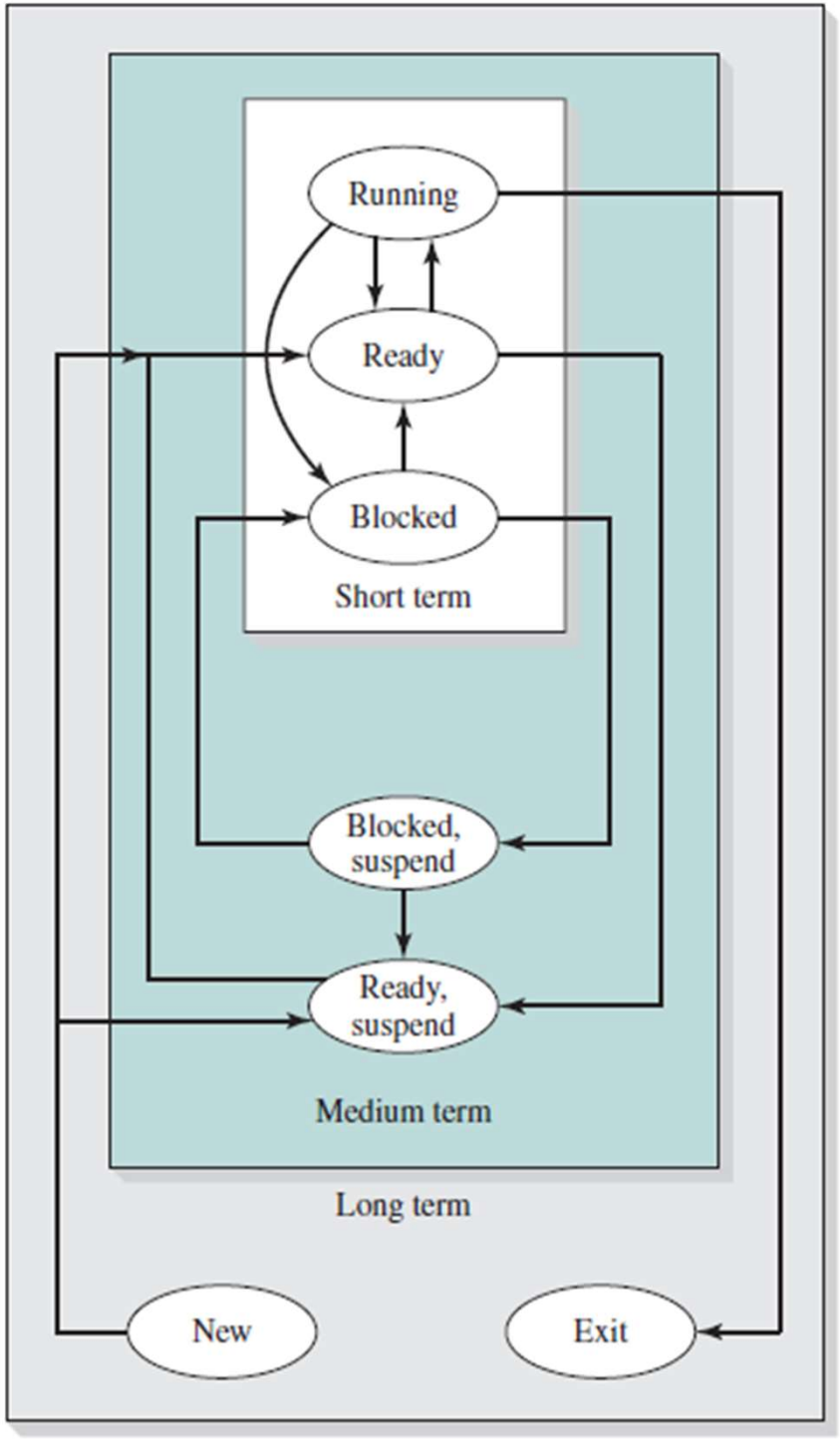
- Short term scheduler select the process from the ready queue, which are ready for the execution, and allocate the processor for execution.
- STS is faster then LTS.
- STS is also called CPU scheduler or dispatcher.
- It is invoked each time the CPU requires a new process for execution.

Medium-term schedulers

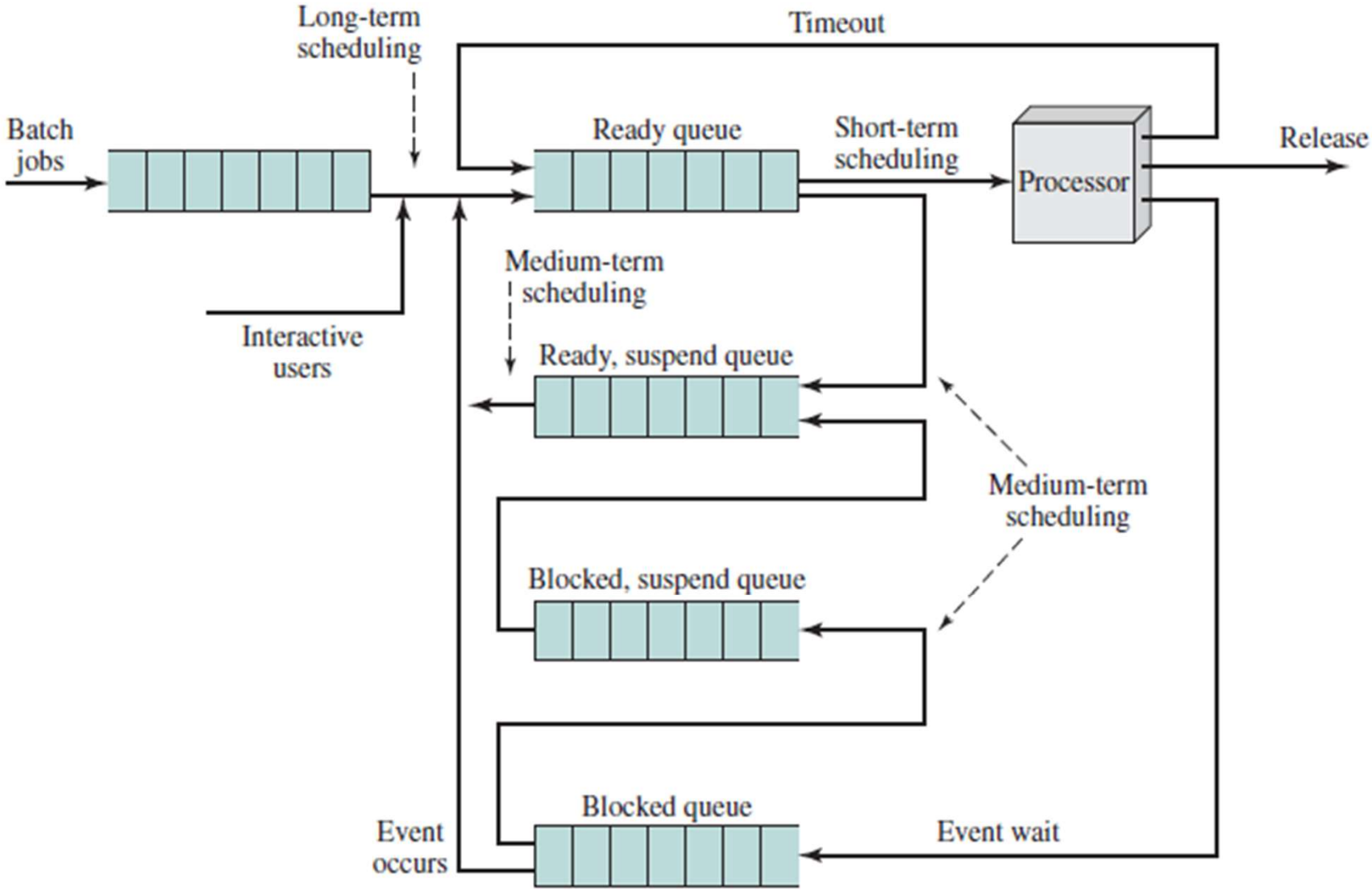
- Medium term scheduler provides the **swapping.**
-
- Ready Q and Block Q are loaded into the main memory. So some time **to reduce the degree of multiprogramming**. Process are suspended from the queue, and move to the secondary memory for temporary time, this is called **swap out.**

- when again these processes are loaded into main memory from secondary memory for execution is called **swap in.**

- These swap in and swap out is determine by the MTS.
- Used to decrease the load of the CPU.
- Used to increase process **Response time.**
- Swapping may be necessary to **improve the process mix**.

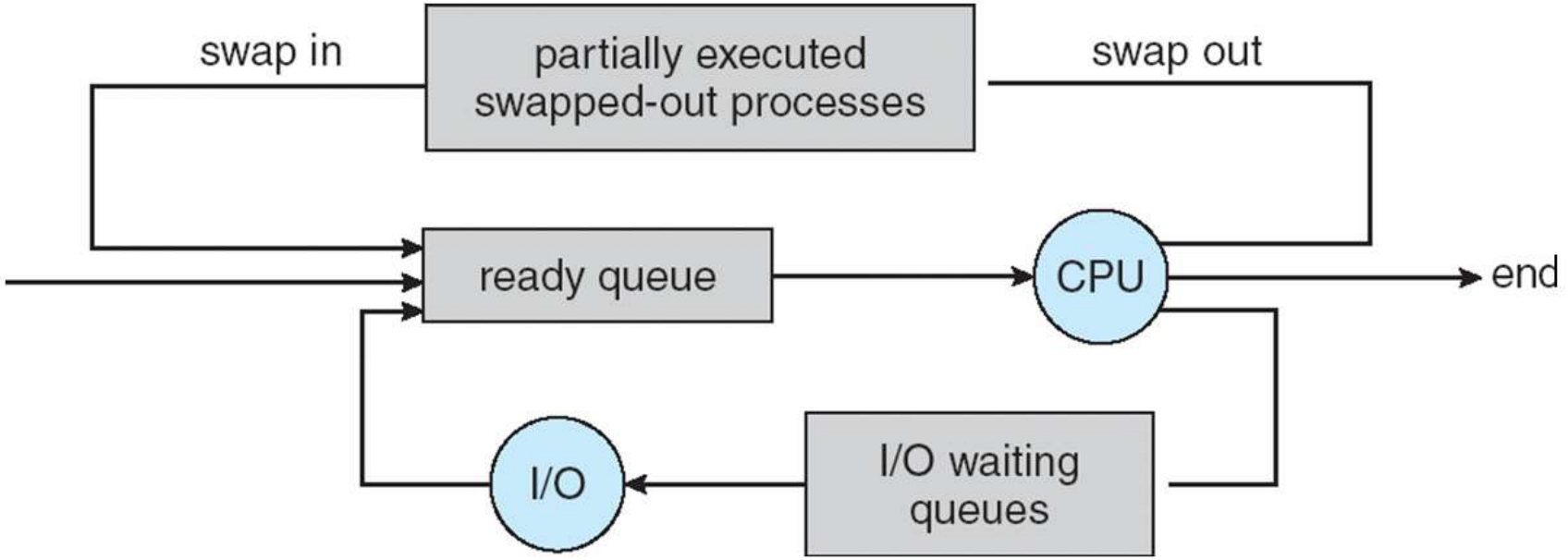


Levels of scheduling

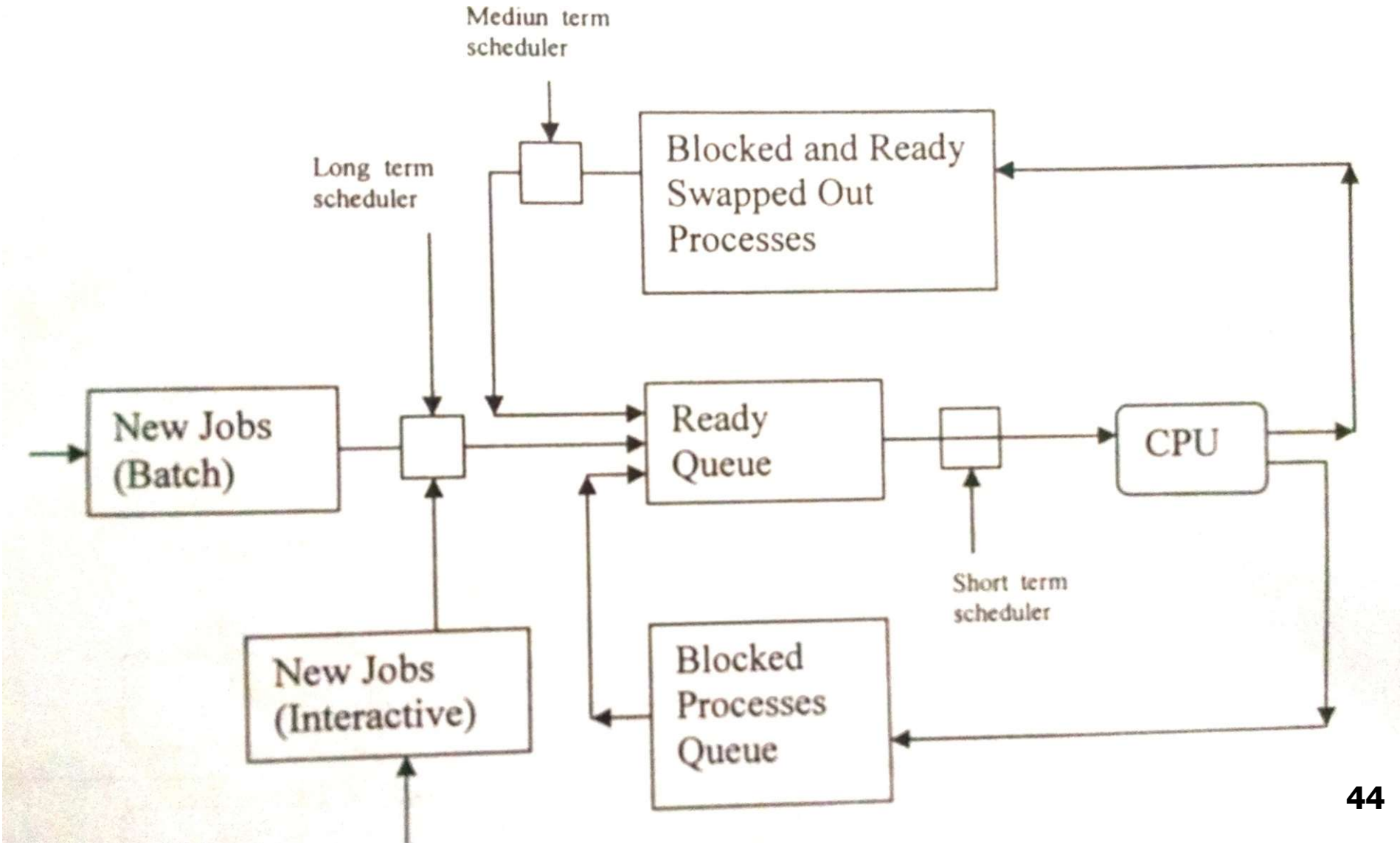


Queuing Diagram for Scheduling

Medium Term Scheduler



Schedulers


Mediun term scheduler
Long term scheduler
Blocked and Ready Swapped Out Processes
New Jobs (Batch)
Ready Queue
CPU
Short term scheduler
New Jobs (Interactive)
Blocked Processes Queue

Dispatcher

- It is a component of CPU scheduling function

- It gives control of the CPU to the process selected by STS.
- Its functions include:
  - Switching the context
  - Switching to User mode
  - Jumping to the proper location in the user program to restart the selected program.

- It is invoked during every process switch.
- The time taken by the dispatcher to stop one process and start another process running is called **dispatch latency**

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – (work done) number of processes that complete their execution per time unit (it may be 1 process/hour or 10 process/second)
- **Turnaround time** – Amount of time to execute a particular process (waiting time to get into memory + waiting time in ready queue + time executing on CPU + time taken for I/O)
- **Waiting time** – amount of time a process has been waiting in the ready queue (sum of all waiting time)
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithm Optimization Criteria

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

Scheduling Algorithms

We'll discuss few major scheduling algorithms here which are following :

1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Shortest Remaining Time First(SRTF)
4. Priority Scheduling ***(Non Preemptive)***
5. Round Robin(RR) Scheduling
6. Multilevel Queue Scheduling
7. Multilevel Feedback Queue Scheduling

Different time with respect to a process

Arrival Time: Time at which the process arrives in the ready queue.

Completion Time: Time at which process completes its execution.

Burst Time: Time required by a process for CPU execution.

Turn Around Time: Time Difference between completion time and arrival time.

Turn Around Time = Completion Time - Arrival Time

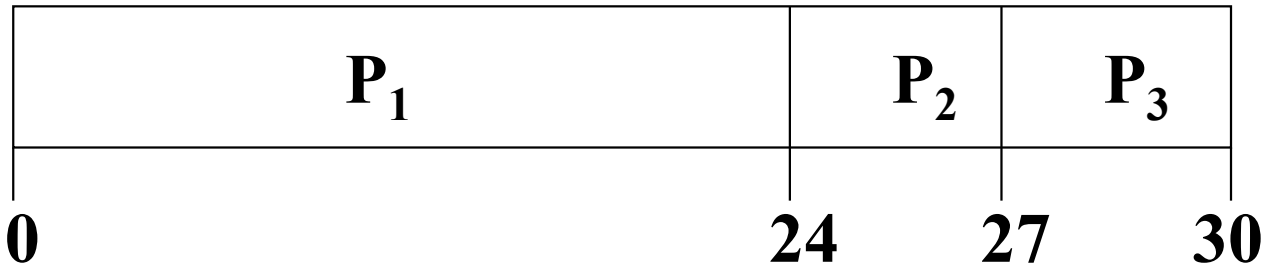
Waiting Time(W.T): Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time - Burst Time

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time/CPU Time
P_1	24
P_2	3
P_3	3

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



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

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6; P_2 = 0, P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

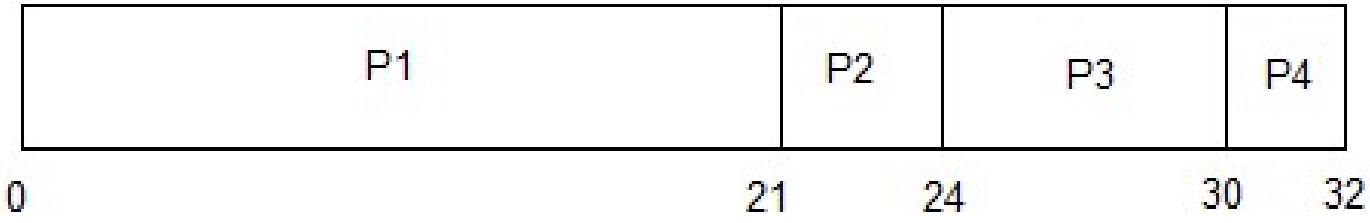
Example-01 What is the average waiting time for the four processes? (FCFS)

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

Solution

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms

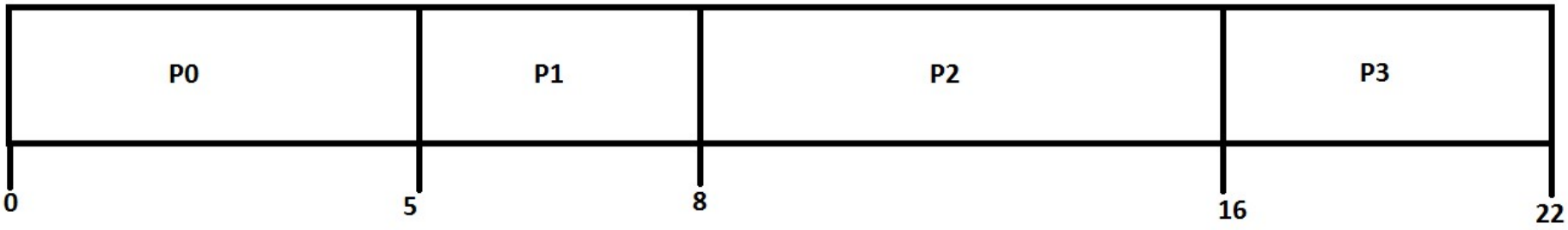


This is the GANTT chart for the above processes

Example-02 What is the average waiting time for the four processes? (FCFS)

Processes	Burst time	Arrival time
P0	5	0
P1	3	1
P2	8	2
P3	6	3

Solution



P0 ----→ 0 - 0 = 0

P1----→ 5 - 1 = 4

P2 ---→ 8 - 2 = 6

P3 ---→ 16 - 3 = 13

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

Example-03 What is the average turn around time for the four processes? (FCFS)

Process	Arrival time	CPU time
P1	0	8
P2	2	2
P3	6	1
P4	8	6
P5	12	4

Avg. turn around time = (8+8+5+9+9)/5 = 7.8

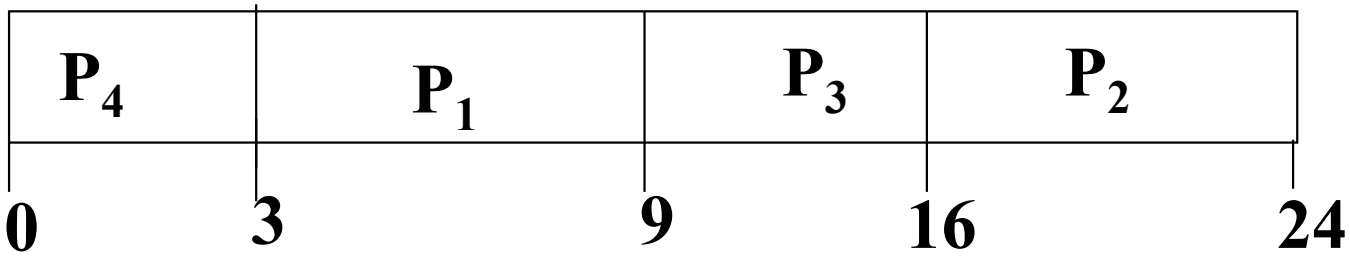
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time

- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Example of SJF (arrival time = 0) Find AVG Waiting Time=?

Process	Arival	Burst Time
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3



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

Example-01 What is the average waiting time for the four processes? (SJF)

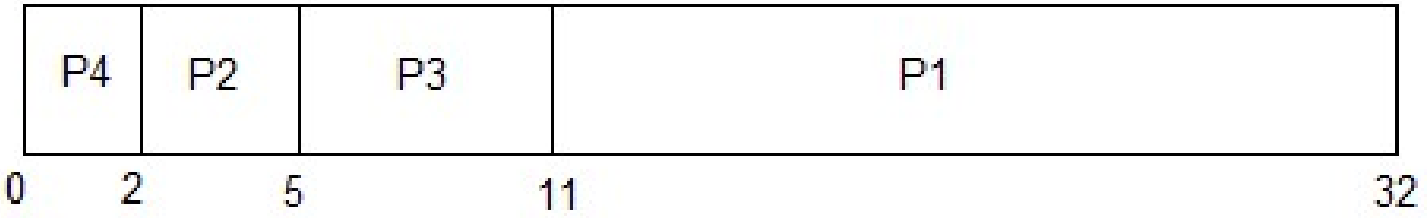
PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

Example-1

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



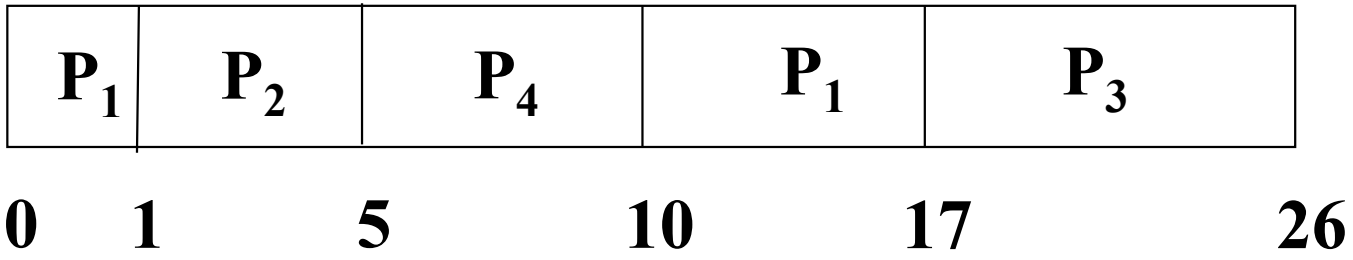
Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

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

- *Preemptive* SJF Gantt Chart



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

Example-1 SRTF

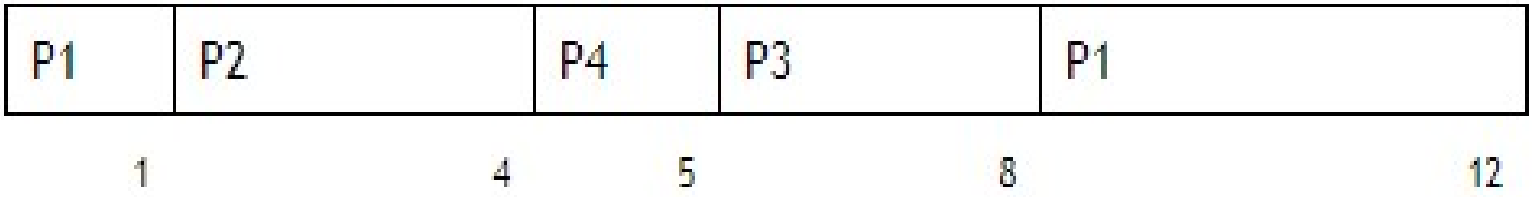
Process	Arrival-Time	Burst-Time
P1	0	5
P2	1	3
P3	2	3
P4	4	1

What is the average **turnaround time** for these processes with the preemptive shortest remaining processing time first (SRTF) algorithm?

(a) 5.50 (b) 5.75 (c) 6.00 (d) 6.25

solution

Execution chart is shown below:



Calculate the Turn Around Time (TAT) for each process as shown in the table below.

TAT = Completion Time - Arrival Time

Pro	AT	BT	CT	TAT(CT-AT)
P1	0	5	12	12
P2	1	3	4	3
P3	2	3	8	6
P4	4	1	5	1

Example-02

An operating system uses Shortest Remaining Time first (SRTF) process scheduling algorithm. Consider the arrival times and execution times for the following processes:

Process	Execution time	Arrival time
P1	20	0
P2	25	15
P3	10	30
P4	15	45

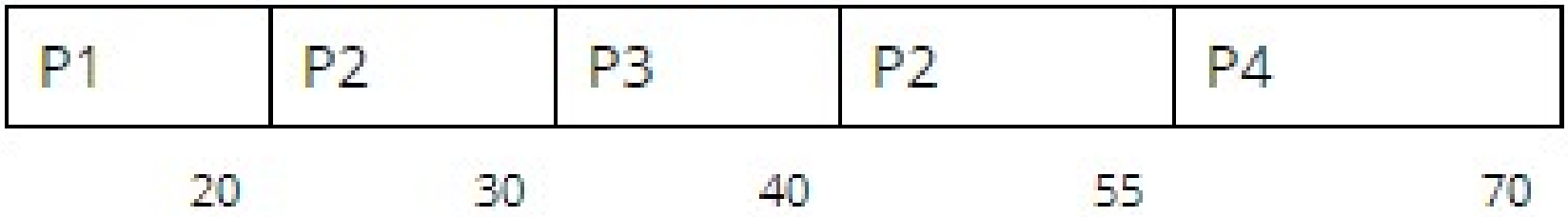
What is the total waiting time for process **P2**?

(a) 5 (b) 15 (c) 40 (d) 55

What is the average Turnaround Time & Waiting Time?

Solution

Execution chart is shown below:



Waiting Time = Completion Time - Arrival Time - Execution Time

Waiting Time = 55 - 15 - 25 = 15

Avg. Turnaround time= 23.75
Avg. Waiting time= 6.25

Example-03

Consider the following table of arrival time and burst time for three processes P0, P1 and P2.

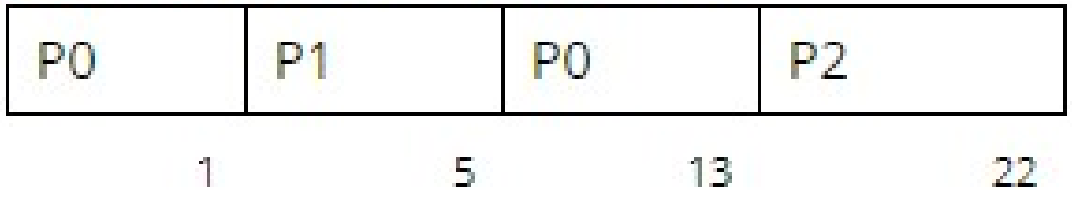
Process	Arrival time	Burst Time
P0	0 ms	9 ms
P1	1 ms	4 ms
P2	2 ms	9 ms

The **pre-emptive shortest job first scheduling algorithm** is used. Scheduling is carried out only at arrival or completion of processes. What is the average **waiting time** for the three processes?

(a) 5.0 ms (b) 4.33 ms (c) 6.33 ms (d) 7.33 ms

Solution

Execution chart is shown below:



Waiting Time = Completion Time - Arrival Time - Execution Time

Pro	AT	BT	CT	WT
P0	0	9	13	4
P1	1	4	5	0
P2	2	9	22	11

Average Waiting Time = (4+0+11)/3 = 5ms

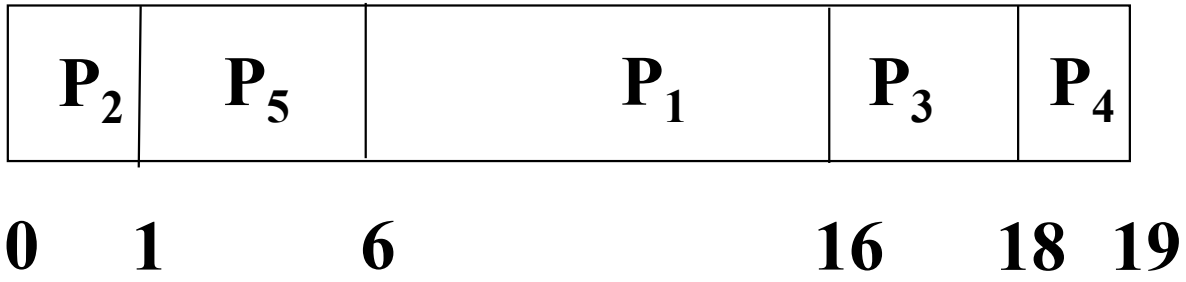
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non preemptive

Example-1 **Non-Preemptive** Priority Scheduling

ProcessAarri	Burst TimeT	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

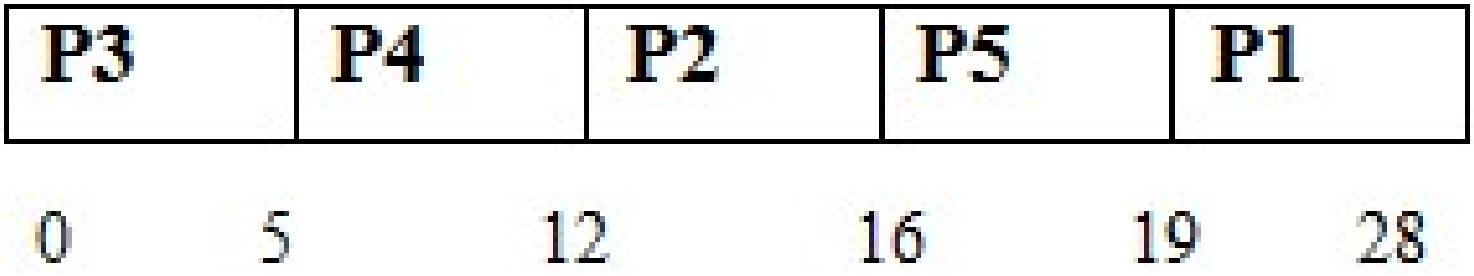
Example-2 **Non-Preemptive** Priority Scheduling

Process	Burst Time (mills.)	Priority
P1	9	5
P2	4	3
P3	5	1
P4	7	2
P5	3	4
Total	**28**	

Find

- **T.W.T** = *Total Waiting Time*
- **A.W.T** = *Average Waiting Time*
- **T.T.T** = *Total Turnaround Time*
- **A.T.T** = *Average Turnaround Time*

Solution



T.W.T= 19 + 12 + 0 + 5 + 16 =**52 mills**
A.W.T = 52 / 5 = **10.4 mills**
T.T.T = 28 + 16 + 5 + 12 + 19 = **80 mills.**
A.T.T =80 / 5 = **16 mills.**

Example-3 **Priority** with **Preemption**

Process	Burst Time (mills.)	Priority	Arrival Time (mills)
P1	9	5	0
P2	4	3	1
P3	5	1	2
P4	7	2	3
P5	3	4	4
Total	28		

Find

1. AVG. Waiting Time
2. AVG. Turnaround Time

Solution

Mills.	P1	P2	P3	P4	P5
0.ms	9				
1.ms	8	4			
2.ms	8	3	5		
3.ms	8	3	4	7	
4.ms	8	3	3	7	3
7.ms	8	3	0	7	3
14.ms	8	3	×	0	3
17.ms	8	0	×	×	3
20.ms	8	×	×	×	0
28.ms	0	×	×	×	×

Average Waiting Time

P1 = 20 - 1 - 0 = 19
P2 = 14 - 1 - 1 = 12
P3 = 4 - 2 - 2 = 0
P4 = 7 - 0 - 3 = 4
P5 = 17 - 0 - 4 = 13

Total 48 Mills.

Average Waiting Time = Total Waiting Time / No.of Process

48 / 5 (No.of Process)

= 9.6 Mills.

Average Turnaround Time

P1 = 28 - 0 = 28
P2 = 17 - 1 = 16
P3 = 7 - 2 = 5
P4 = 14 - 3 = 11
P5 = 20 - 4 = 16

Total 76 Mills.
Average Turnaround Time = Total Turnaround Time / No.of Process

76 / 5

=15.2 Mills.

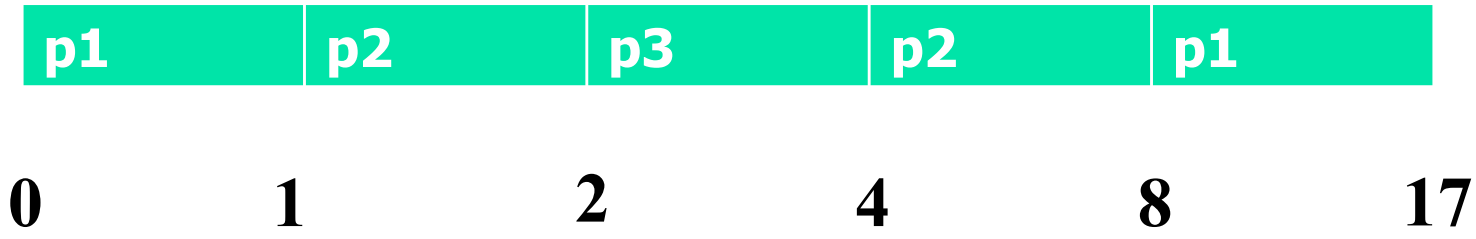
Ex-03

process	CPU Burst Time	Priority	Arrival
p1	10	3	0
p2	5	2	1
p3	2	1	2

Find Average Waiting Time?

Let us assume that 1 is the highest priority whereas 3 is the least priority.

Solution



- Waiting time for P1= 7
- Waiting time for P2= 2
- Waiting time for P3 = 0
- Average Waiting Time = (7+2+0) / 3 = 3 Millisecond

Limitation

- indefinite blocking or ***starvation***

- In priority scheduling some low priority process may keep waiting indefinitely for CPU. In a heavily loaded system continuous arrival of higher priority processes can prevent low priority process from getting the CPU.

Solution

- One solution to the problem of *starvation* is ***aging***
- **Aging** is a technique of gradually increasing the priority of process that waits in the system for a longer period of time.
- Eg:- If priority range from 127(low) – 0(high) we could decremented the priority of waiting process by every 15 minutes.

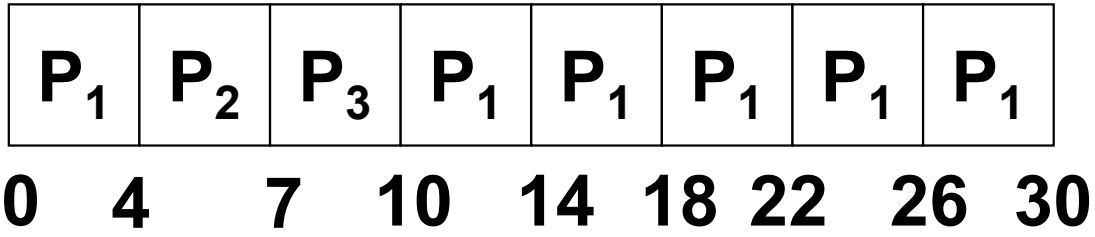
Round Robin (RR)

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

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Draw Gantt chart



- Typically, higher average turnaround than SJF, but better *response*
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

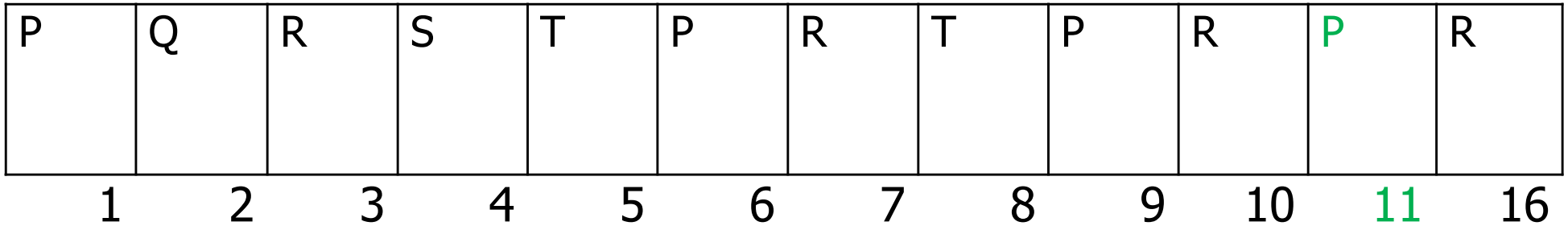
Ex-01

Job-Id	CPU-BurstTime
p	4
q	1
r	8
s	1
t	2

The jobs are assumed to have arrived at time 0 and in the order p, q, r, s, t. Calculate the departure time (completion time) for job **p** if scheduling is **round robin** with time slice **1**.

(a) 4 (b) 10 (c) 11 (d) 12

Solution



Ex-02

Process	Arrival time	Service time (Burst time)
P1	0	4
P2	3	6
P3	5	3
P4	8	2

Time quantum=2

Draw Gantt Chart

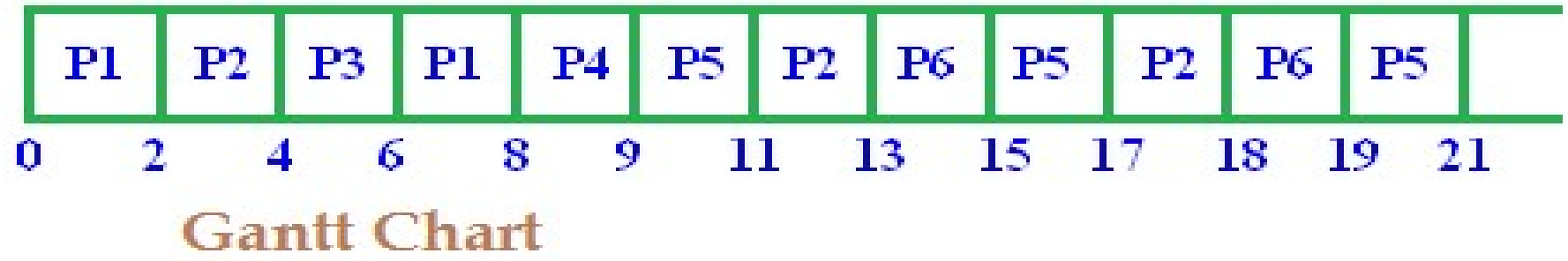
Ex-03

P. No.	A.T.	B.T.
P1	0	4
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

- Calculate the **Turnaround time** and **waiting time** of the processes on the basis of **round robin scheduling** algorithm. Assume **Time Quantum is set to 2 units**. Also calculate the **average waiting time and turn around time**.

Solution

Ready State : P1 P2 P3 P1 P4 P5 P2 P6 P5 P2 P6 P5



Gantt Chart

P No.	AT	BT	CT	TAT	WT
1	0	4	8	8	4
2	1	5	18	17	12
3	2	2	6	4	2
4	3	1	9	6	5
5	4	6	21	17	11
6	6	3	19	13	10
				67/6=10.8	46/6=7.3

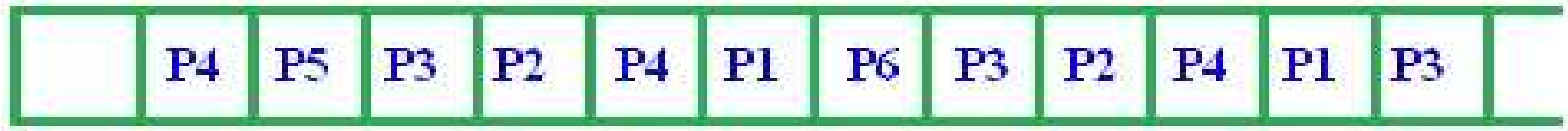
Ex-04

P No.	AT	BT
P1	5	5
P2	4	6
P3	3	7
P4	1	9
P5	2	2
P6	6	3

Calculate the Turnaround time and waiting time of the processes on the basis of round robin scheduling algorithm. Assume Time Quantum is set to 3 units. Also calculate the average waiting time and turn around time.

Solution

Ready State : P4 P5 P3 P2 P4 P1 P6 P3

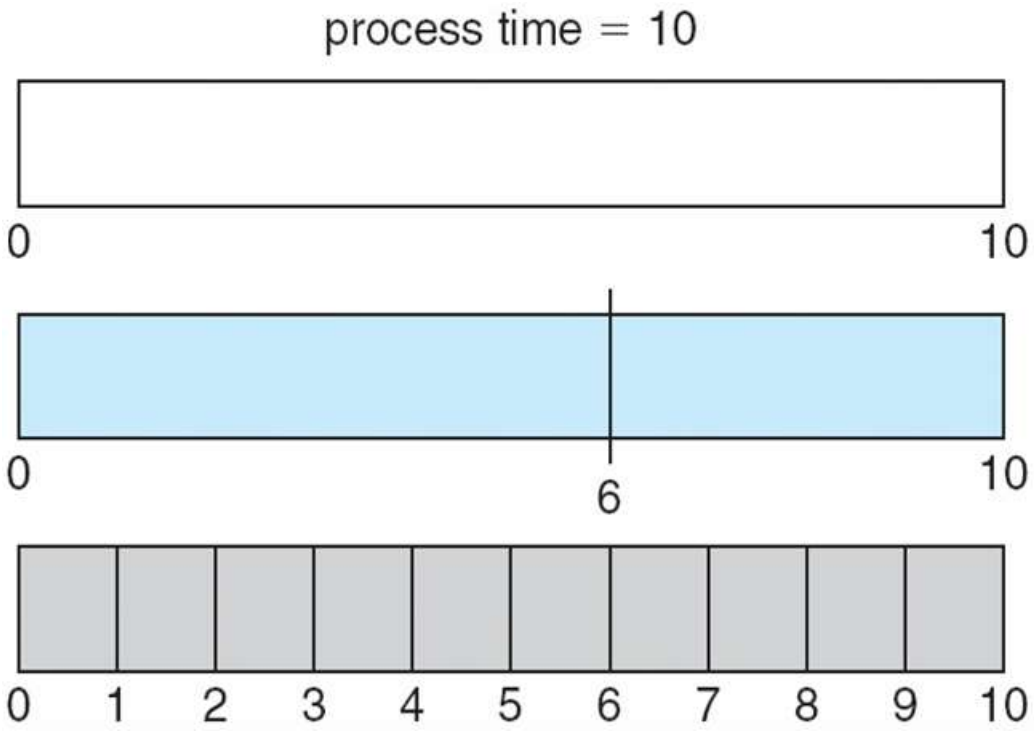


0 1 4 6 9 12 15 18 21 24 27 30 32 33

Gantt Chart

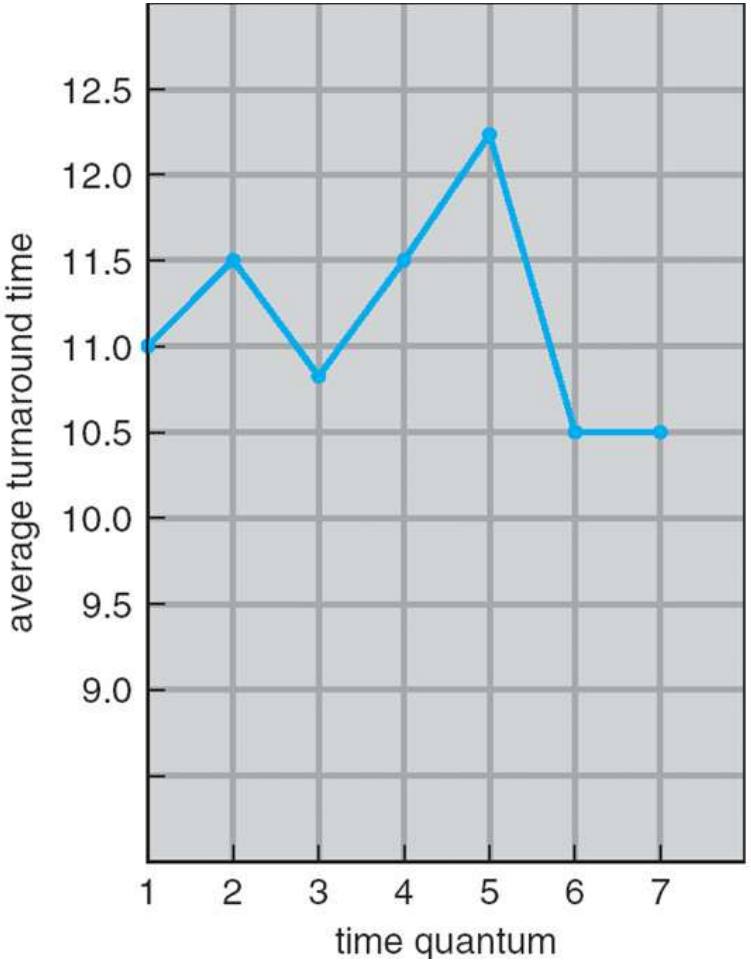
P No.	AT	BT	CT	TAT	WT
1	5	5	31	26	21
2	4	6	29	25	19
3	3	7	32	29	22
4	1	9	26	25	16
5	2	2	5	3	1
6	6	3	20	14	11
				128/6=21.3	92/6=15.3

Time Quantum and Context Switch Time



quantum	context switches
12	0
6	1
1	9

Turnaround Time Varies With The Time Quantum



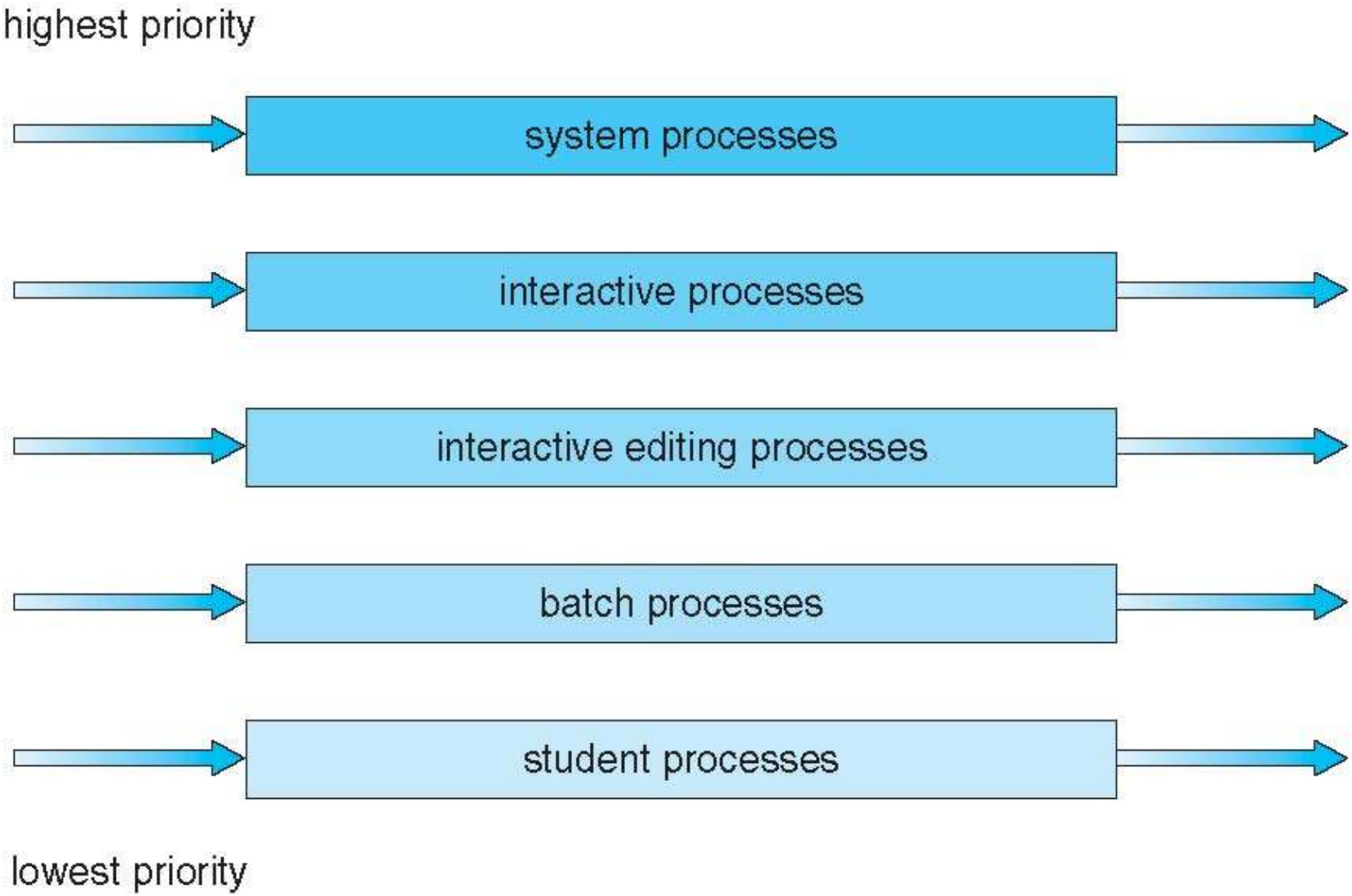
process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - 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

Multilevel Queue Scheduling

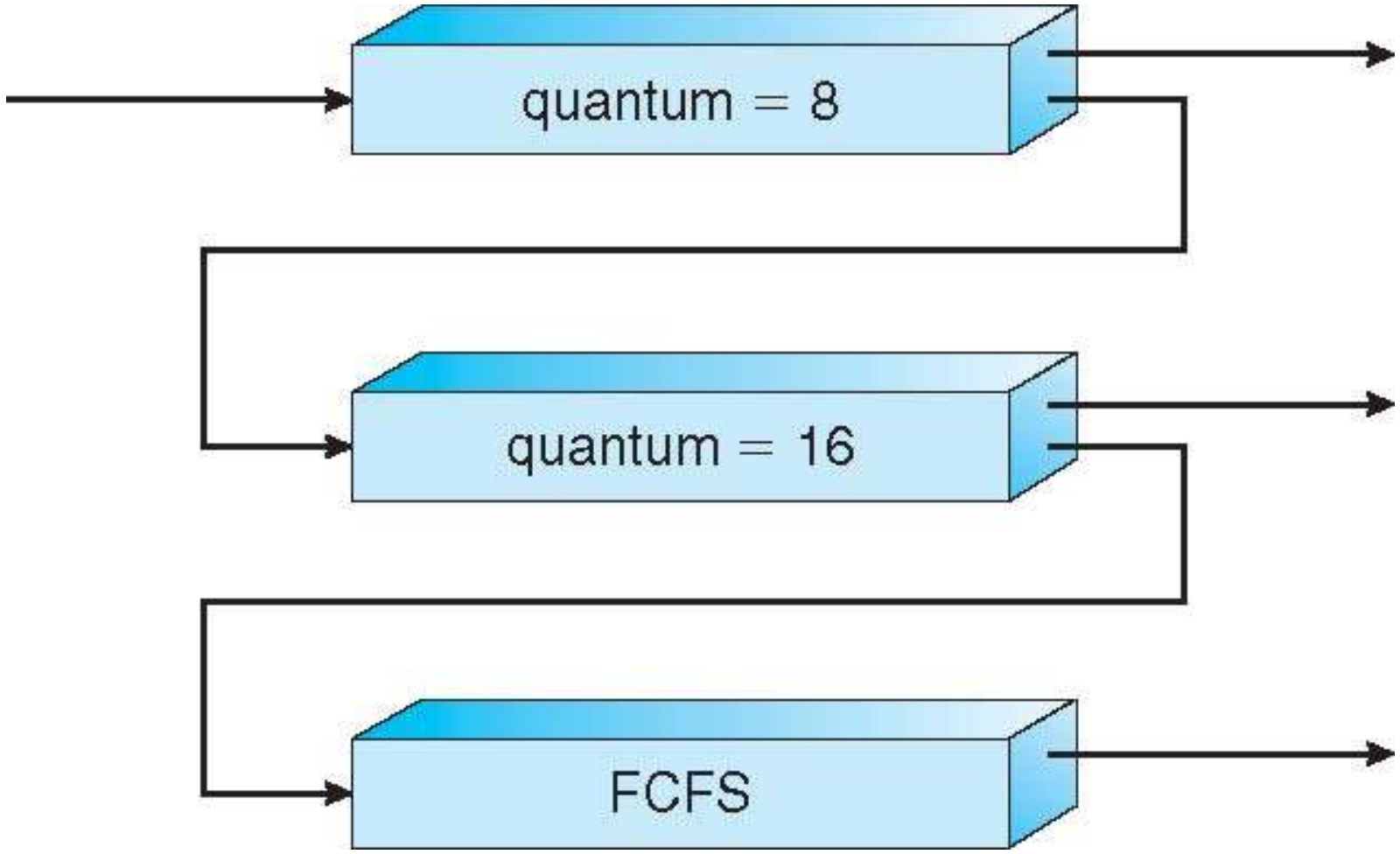


Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS

- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2

Multilevel Feedback Queues



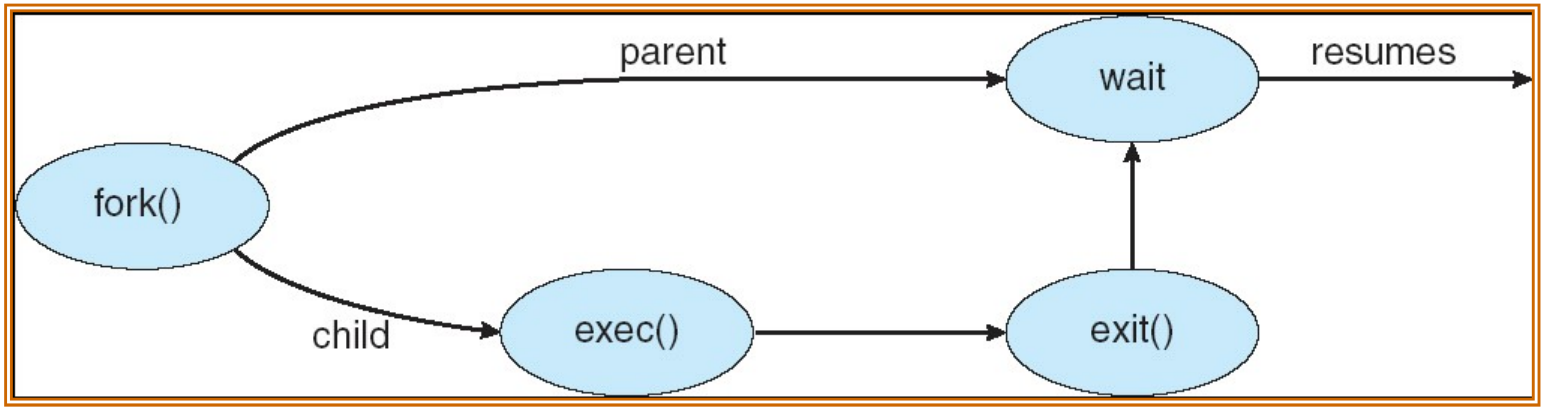
Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program

Process Creation



C Program Forking Separate Process

```
int main()
{
Pid_t  pid;
        /* fork another process */
        pid = fork();
        if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                exit(-1);
        }
        else if (pid == 0) { /* child process */
                execlp("/bin/ls", "ls", NULL);
        }
        else { /* parent process */
                /* parent will wait for the child to complete */
                wait (NULL);
                printf ("Child Complete");
                exit(0);
        }
}
```

Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*

Thank You