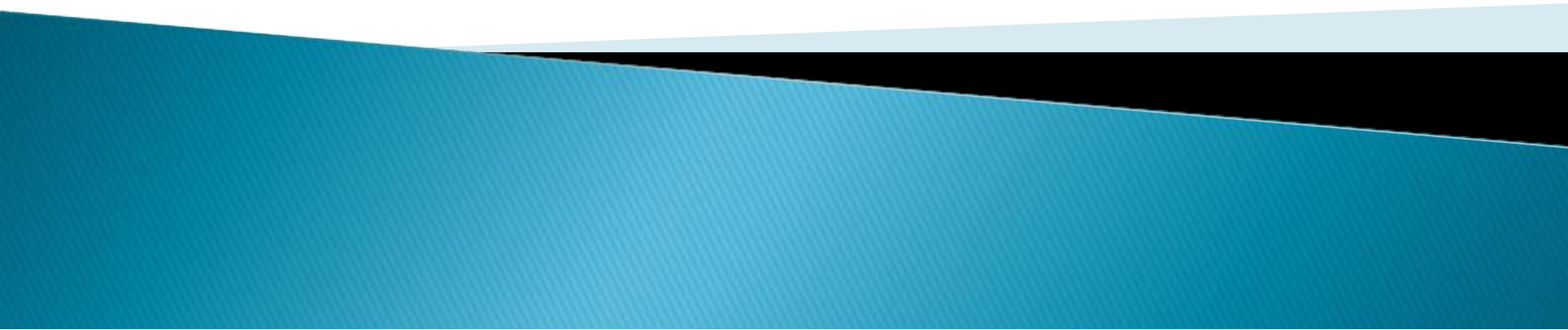


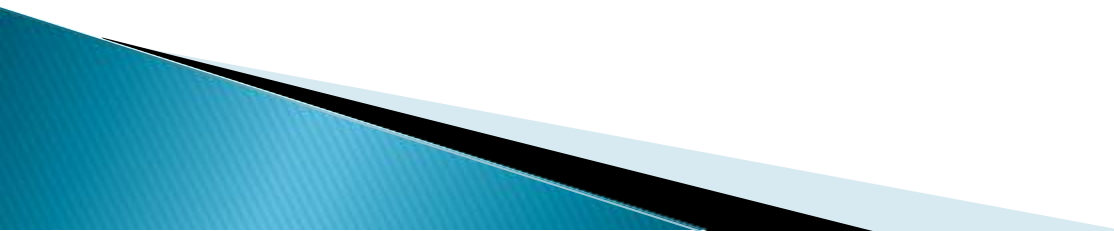
Process Management

Tasneem Mirza



Process Concept

Difference between a process and a program

- ▶ A program is a passive entity, such as the contents of a file stored on disk
 - ▶ The term process refers to program code that has been loaded into a computer's memory so that it can be executed by the *central processing unit* (CPU).
 - ▶ In a multiprogramming system, there will be a number of processes in memory waiting to be executed on the CPU.
 - ▶ The process management function of the operating system must ensure that each process gets a fair share of the CPU's time.
 - ▶ The operating system must then determine when the CPU can be made available to the process, and for how long. Once a process controls the CPU, its instructions will be executed until its allotted time expires, or until it terminates, or until it requests an input or output operation.
- 

Process States

- ▶ Process execution is a series of bursts
CPU burst, I/O burst, CPU burst, I/O burst, etc.

For ex:

Start	-----	CPU cycle
Read A,B	-----	IO cycle
C=A+B	}	CPU cycle
D=(A*B)-C		
E=A-B		
F=D/E		
Write A,B,C,D,E,F	-----	IO Cycle
Stop	-----	CPU cycle

- ▶ As a process executes , it changes state .
- ▶ Each process may be in one of the following states:
- ▶ **New** : a new process has not yet been loaded into main memory, although its **process control block (PCB)** has been created.
- ▶ **Ready** :The process is in the memory and waiting to be assigned to a processor.
- ▶ **Running** : The process that is currently being executed. Assume a computer with a single processor, so at most one process at a time can be in this state.
- ▶ **Waiting** : The process is waiting for some event like I/O.
- ▶ **Terminated**: The process finished execution

**Note: Only one process can be running (single processor system)
many processes may be in ready and waiting**



Process State transition Diagram

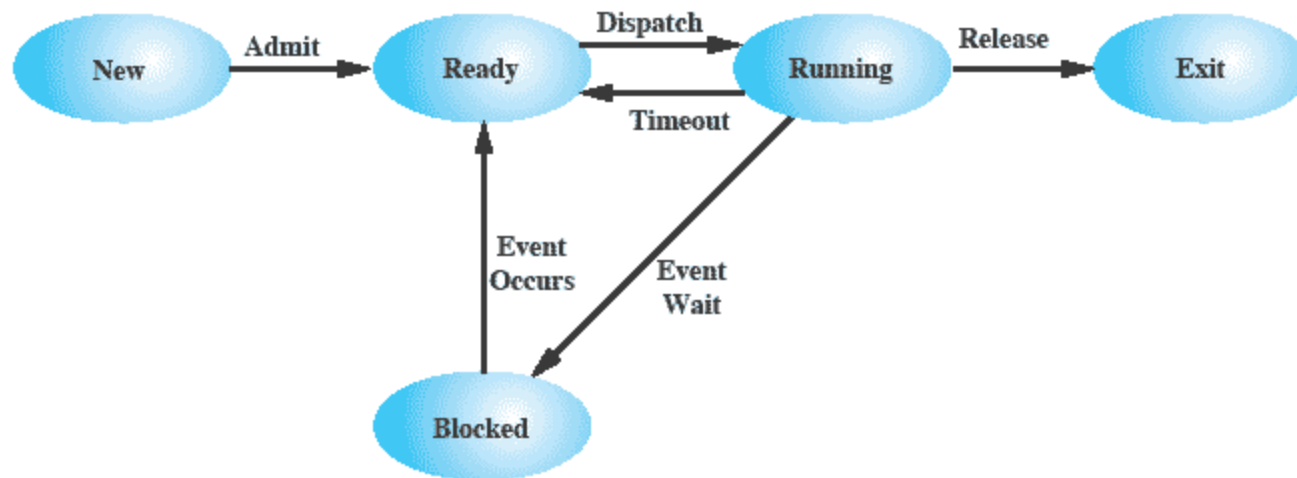
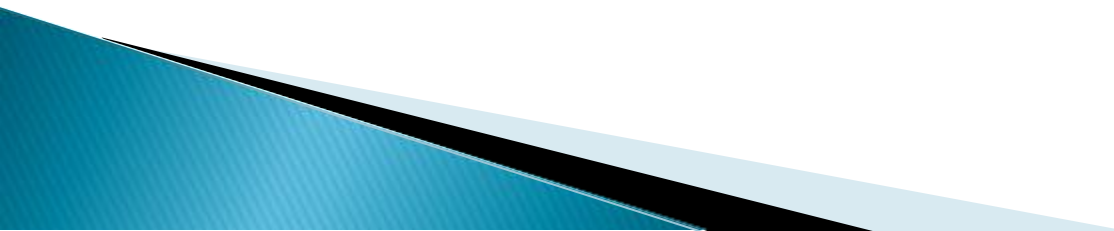


Figure 3.6 Five-State Process Model

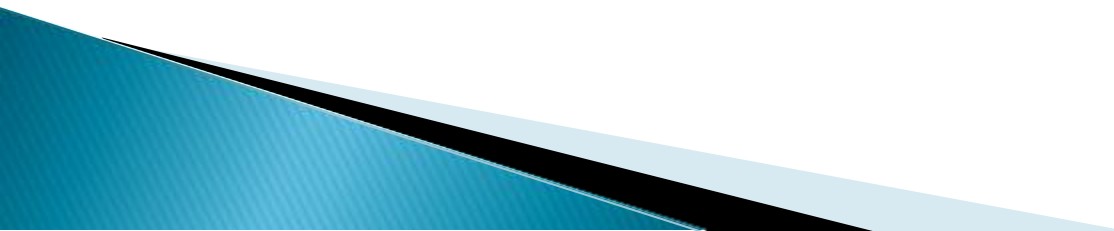
Process Queues

OS keeps track of processes with a set of queues:

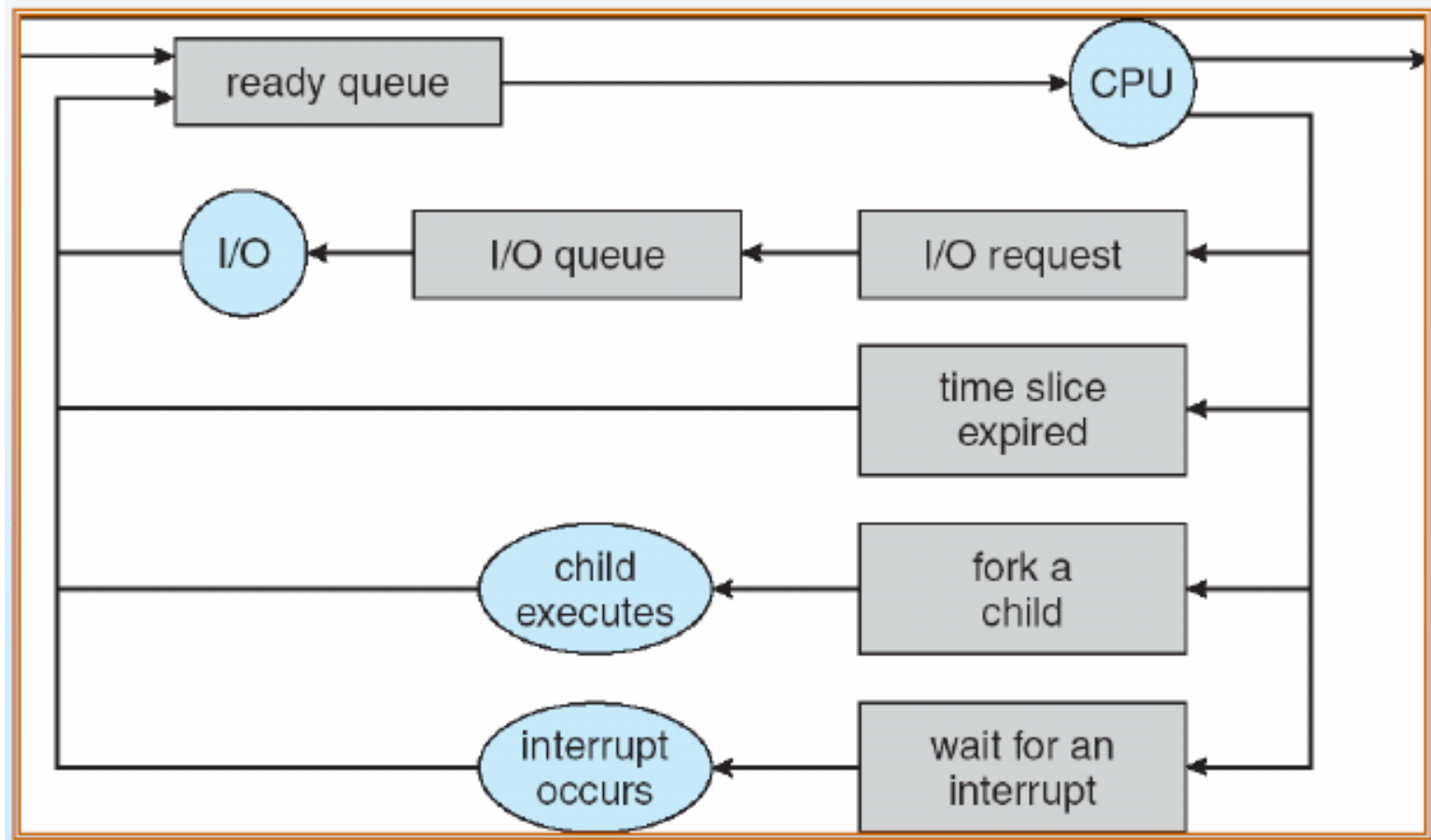
- ▶ Job queue – set of all processes in the system
 - ▶ Ready queue – set of all processes residing in main memory, ready and waiting to execute
 - ▶ Device queues – set of processes waiting for an I/O device
 - ▶ Processes migrate between the various queues as they execute
- 

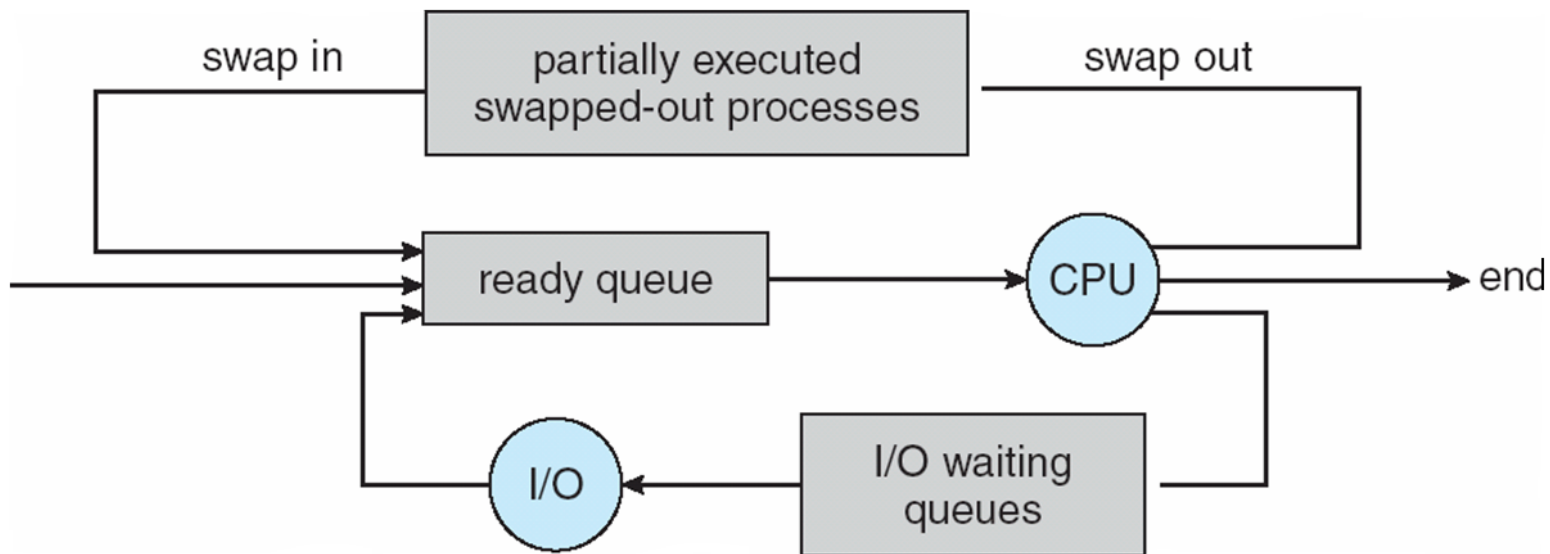
- ▶ The OS must select , for scheduling purposes processes from these queues in some fashion.
- ▶ The selection process is carried out by the appropriate scheduler.
- ▶ The processes are spooled to a mass storage device like a disk.
- ▶ The long term scheduler(LTS) or job scheduler, selects from this pool and loads them into memory for execution.
- ▶ Hence LTS controls the degree of multiprogramming.
- ▶ The short term scheduler (STS) or the CPU scheduler , selects from among the processes that are ready to execute , and allocates CPU to one of them.
- ▶ Difference between LTS and STS is frequency of execution. The STS must select a new process for the CPU frequently.
Hence STS executes more frequently than LTS.

- ▶ Processes can be described as either:
 - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts (e.g., surfing the web, copying large files)
 - **CPU-bound process** – spends more time doing computations; few very long CPU bursts (e.g., processes doing intensive mathematical computation)
- ▶ It is important for the long-term scheduler to select a good process mix of I/O-bound and CPU-bound processes
 - What will happen to the ready queue and I/O queue:
 - If all processes are I/O bound?
Then ready queue will be almost empty.
 - If all processes are CPU bound?
Then I/O queue will be almost empty

- ▶ Another component involved in the CPU scheduling function (Function of assigning the CPU to one of the processes in the ready queue) is the dispatcher.
 - ▶ The dispatcher is the module that actually gives control of the CPU to the process selected by the STS.
 - ▶ This function involves loading the registers of the process, jumping to the proper location in the program to restart it.
-
- ▶ Another scheduler involved is the mid term scheduler(MTS)
 - ▶ The MTS removes processes from the main memory and thus reduces the degree of multiprogramming. At some time later, the process can be reintroduced into the memory and its execution can be continued from where it left it off.
- 

Scheduling queues

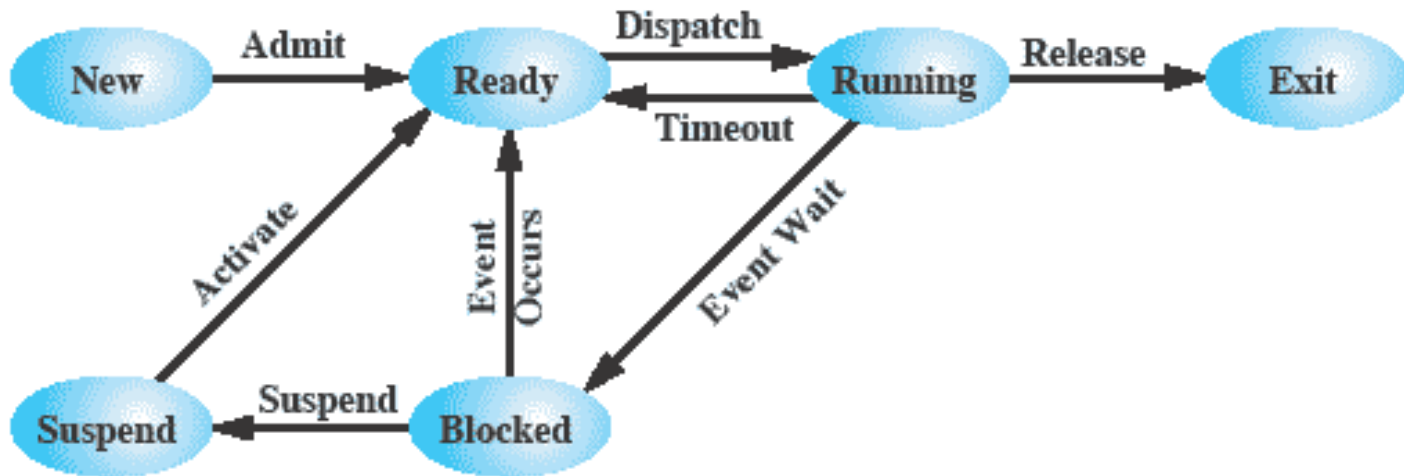




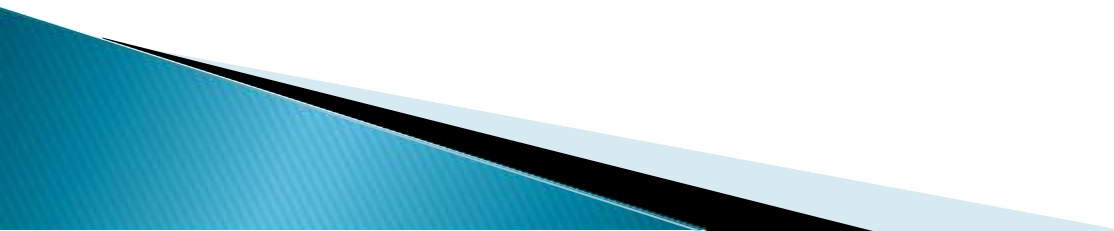
Addition of medium term scheduler to the queuing diagram

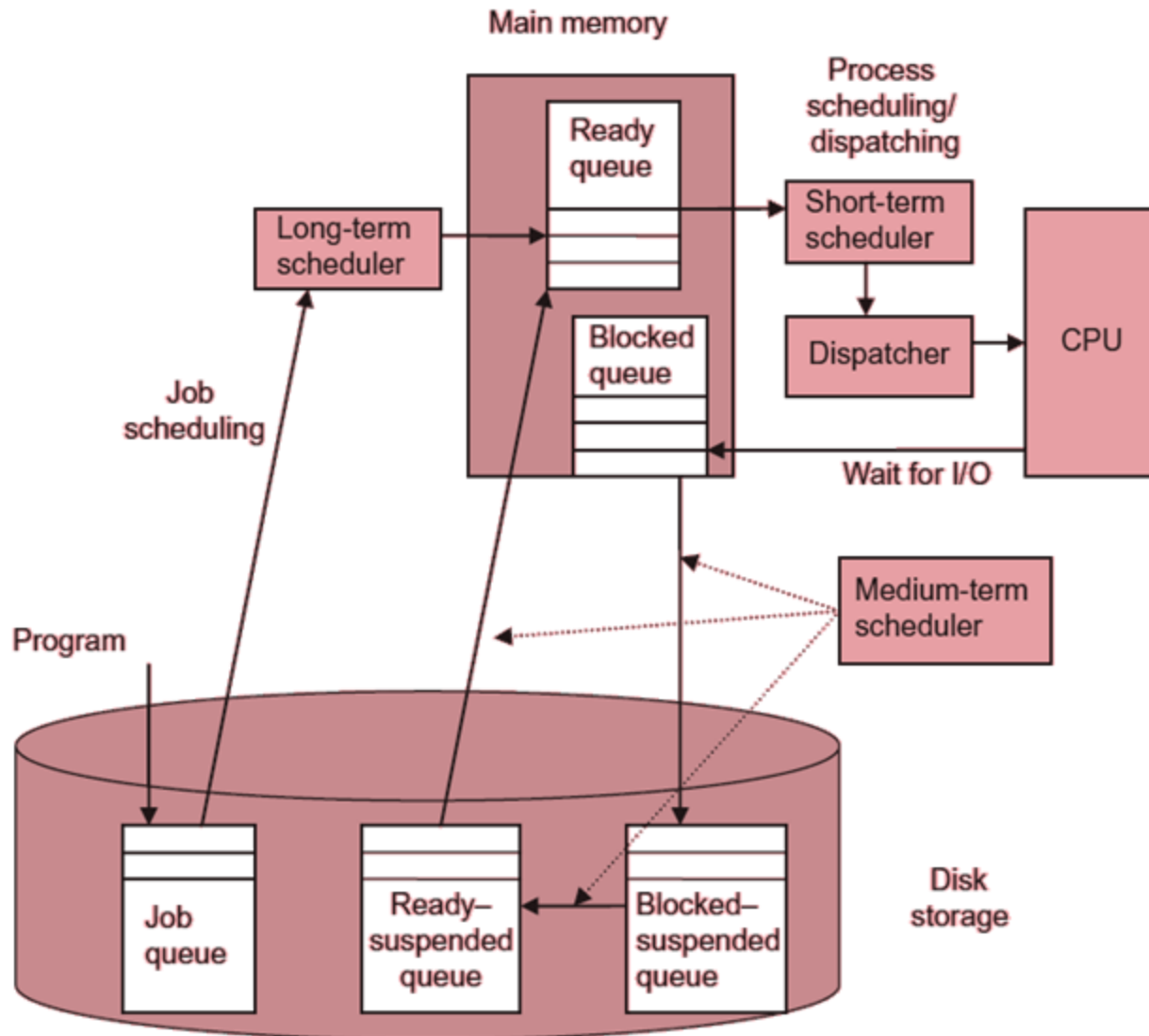
Medium Term Scheduling

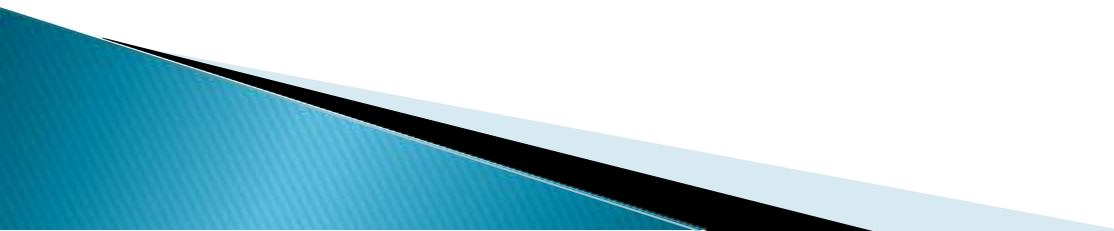
- Possibility-All processes in memory need I/O ie. All processes are blocked and waiting for an event (IO) and no process is under execution.
- Requirement – Bring in some process from the disk which is ready for execution.
- If no space in memory – Free the space by swapping out a blocked process.
- The swapped out processes are known as the suspended processes.



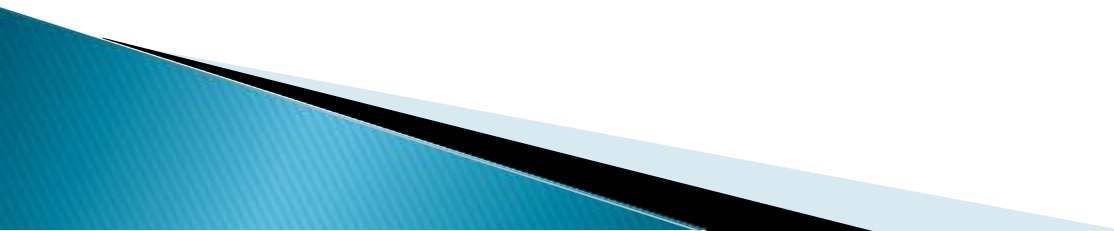
(a) With One Suspend State

- ▶ There another queue on the disk called the **blocked suspend queue** for all the processes which are suspended in the blocked state.
 - ▶ The task of swapping the process from blocked queue to blocked suspend queue is performed by the medium term scheduler.
 - ▶ When there is a signal for the completion of an IO , for which the process is blocked and presently in the suspend queue, the state of the process is changed to ready-suspend and moved to the ready-suspend queue (which is another queue on the disk). This task of moving a process from blocked-suspend queue to the ready suspend queue is also performed by the medium term scheduler.
- 



- ▶ Whenever the suspended process is swapped out on the disk, there are two choices for bringing in a process, which are ready for execution.
 - ▶ First : Pick up a process from the ready suspend queue and send them to the ready queue.
 - ▶ Second : A new process from the job queue can be send to the ready queue.
 - ▶ Second choice increases the load on the system(by increasing the list of unfinished jobs).
 - ▶ Hence generally the first option is preferred.
- 

A Seven-State Model

- ▶ Ready: The process is in main memory and available for execution.
 - ▶ Blocked: The process is in main memory and awaiting an event I/O event.
 - ▶ Blocked suspend: The process is in secondary memory and awaiting an event.
 - ▶ Ready suspend: The process is in secondary memory but is available for execution as soon as it is loaded into main memory.
- 

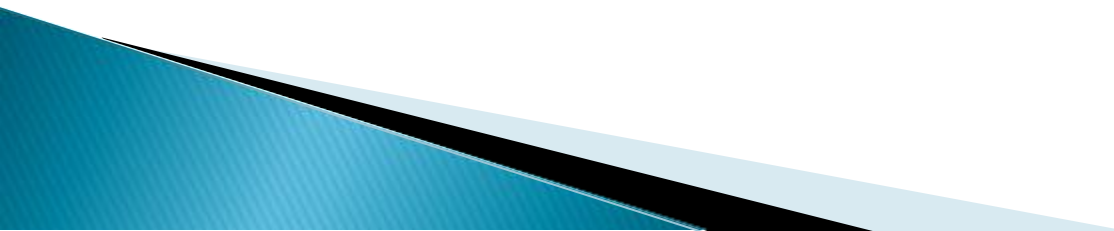
A Seven-State Model

- ▶ Blocked > Blocked suspend
If there are no ready processes, then at least one blocked process is swapped out to make room for another process that is not blocked.

A Seven-State Model

- ▶ Blocked suspend > Ready suspend


A process in the Blocked suspend state is moved to the Ready suspend state when the event for which it has been waiting occurs.



A Seven-State Model

- ▶ Ready suspend > Ready

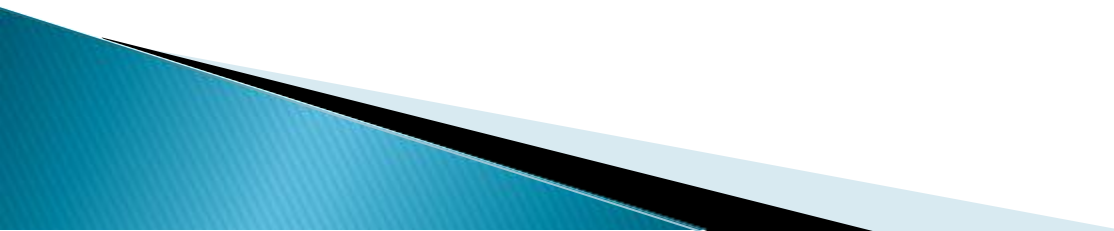
When there are no ready processes in main memory, the operating system will need to bring one in to continue execution.

- ▶ It might be the case that a process in the ready suspend state has higher priority than any of the processes in the Ready state. In that case, the operating system designer may decide that it is more important to get in the higher-priority process than to minimize swapping.
- 

A Seven-State Model

- ▶ Ready > Ready suspend

It may be necessary to suspend a ready process if that is the only way to free a sufficiently large block of main memory.



A Seven-State Model

- ▶ New > Ready suspend and New > Ready
When a new process is created, it can either be added to the Ready queue or the Ready, suspend queue.
- ▶ There would always be insufficient room in main memory for a new process; hence the use of the New > Ready suspend transition.

A Seven-State Model

- ▶ Blocked suspend > Blocked

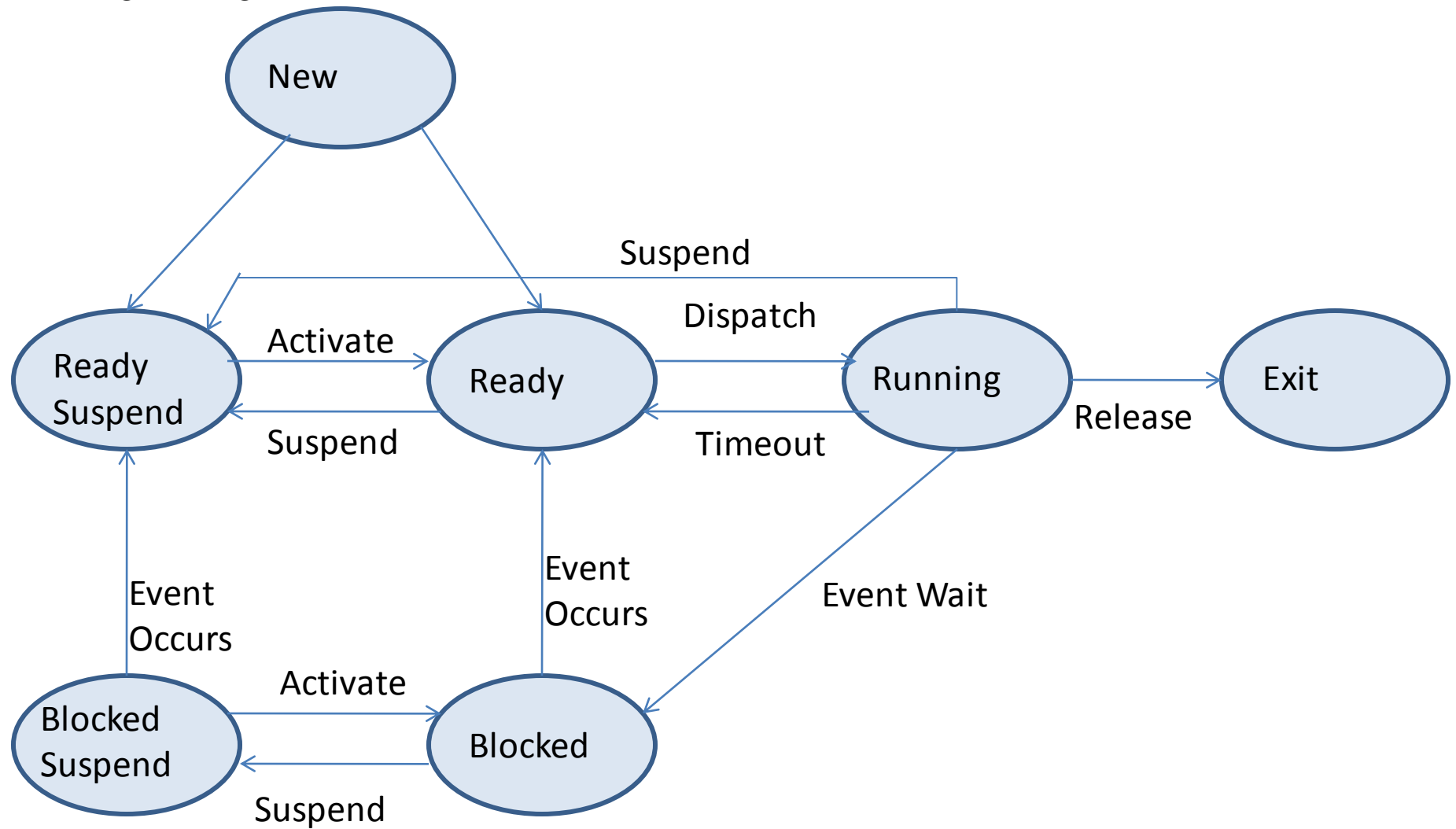
A process terminates, freeing some main memory. There is a process in the Blocked suspend queue with a higher priority than any of the processes in the Ready suspend queue and the operating system has reason to believe that the blocking event for that process will occur soon.

A Seven-State Model

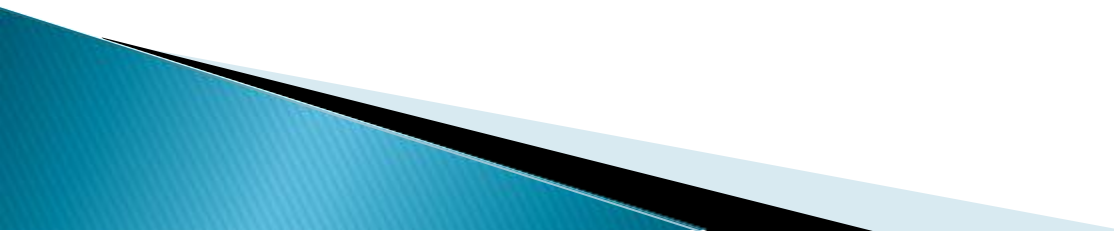
- ▶ Running > Ready suspend

If the operating system is preempting the process because a higher-priority process on the Blocked suspend queue has just become unblocked, the operating system could move the running process directly to the Ready suspend queue and free some main memory.

7 STATE PST



Context Switch

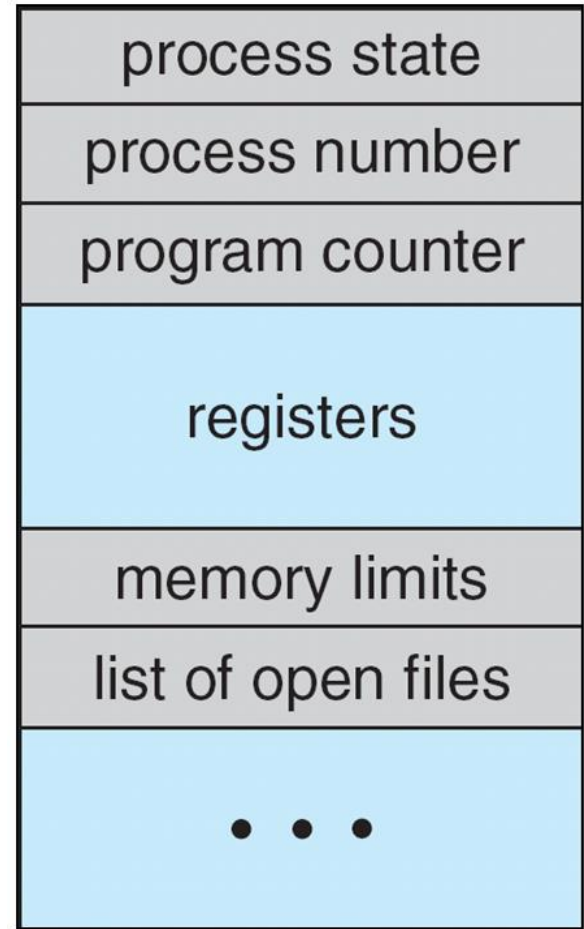
- ▶ When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
 - ▶ **Context** of a process represented in the Process Control Block(PCB)
 - ▶ Context-switch time is overhead; the system does no useful work while switching
- 

Process Control Block (PCB)

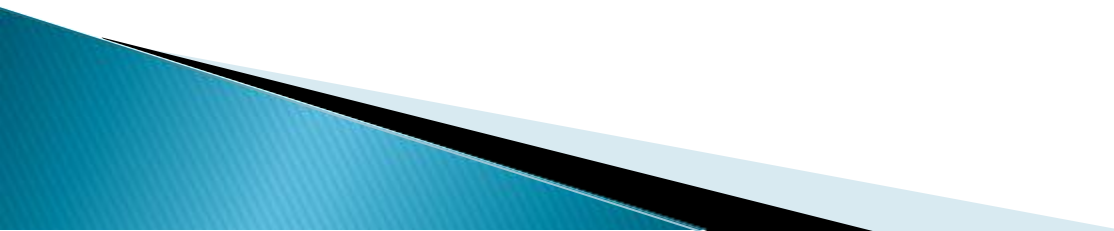
Information associated with each process
is stored in the PCB

(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute.
- CPU registers – contents of all process-centric registers.
- CPU scheduling information- priorities, the amount of time the process has been waiting and the amount of the process executed the last time it was running.
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start,
- I/O status information – I/O devices allocated to process.



Cpu scheduling / Process Scheduling

- ▶ How does CPU manage the execution of simultaneously ready processes?
i.e. When the CPU becomes idle , the OS must select one of the processes in the ready queue to be executed.
 - ▶ The selection process is carried out by the STS (CPU scheduler).
 - ▶ The scheduler selects from among the processes that are ready to execute, and allocates the CPU to one of them.
 - ▶ **Which process to select ?**
 - ▶ **When to select ?**
- 

- ▶ **Non –preemptive**

Process runs until voluntarily relinquishes the CPU

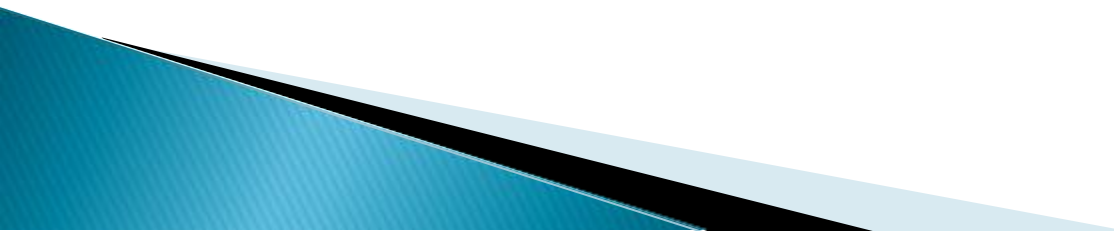
- process blocks on an event (e.g., I/O)

- process terminates

- ▶ **Preemptive**

The scheduler actively interrupts and deschedules an executing process and schedules another process to run on the CPU.

Scheduling criteria

- ▶ Process/CPU scheduling algorithms decides which process to execute from a list of processes in the ready queue.
 - ▶ There a number of algorithms for CPU scheduling. All these algorithms have certain properties
 - ▶ To compare different algorithms there are certain criteria
 - ▶ These criteria can be can be used to compare different algorithms to find out the best algorithm.
- 

Scheduling criteria

1. **CPU utilization** : The CPU should be kept as busy as possible. CPU utilization ranges from 0% to 100 %. Typically it ranges from 40 % (for a lightly loaded system) to 90 % (for a heavily loaded system)
2. **Throughput** : If the CPU is busy executing processes, then work is being done.

One measure of work is the number of processes completed per time unit, called throughput.

Varies from 1 process per hour to 10 processes per second.

3. **Turnaround time**: How long it takes to execute a process ?
Measured from the time of interval of submission of a process to the time of completion.
4. **Waiting time** : The amount of time that a process spends in the ready queue. Sum of the periods spent waiting in the ready queue.
5. **Response time** : Time from the submission of a request until the first response is produced. NOT THE TIME IT TAKES TO OUTPUT THE RESPONSE

- ▶ Ideally the Scheduling algorithm should :
Maximize : CPU utilization , Throughput.
Minimize : Turnaround time, Waiting time, Response time

Type of Scheduling Algorithms

- ▶ Non-preemptive Scheduling
 - ▶ Preemptive Scheduling
- 

Basic Scheduling Algorithm

- ▶ Deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.

1. FCFS – First–Come, First–Served

- CPU is allocated to the process that requested it first.
- Non–preemptive
- Ready queue is a FIFO queue
- Jobs arriving are placed at the end of queue
- Dispatcher selects first job in queue and this job runs to completion of CPU burst

First-Come, First-Served (FCFS) Scheduling

<u>Process</u>	<u>Burst Time</u>
P_1	24 ms
P_2	3 ms
P_3	3 ms

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3 , all at the same time. The Gantt Chart for the schedule is:



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

First-Come, First-Served (FCFS) Scheduling

- ▶ Compute Average turnaround time



Compute each process's turnaround time

$$T(p_1) = 24\text{ms}$$

$$T(p_2) = 3\text{ms} + 24\text{ms} = 27\text{ms}$$

$$T(p_3) = 3\text{ms} + 27\text{ms} = 30\text{ms}$$

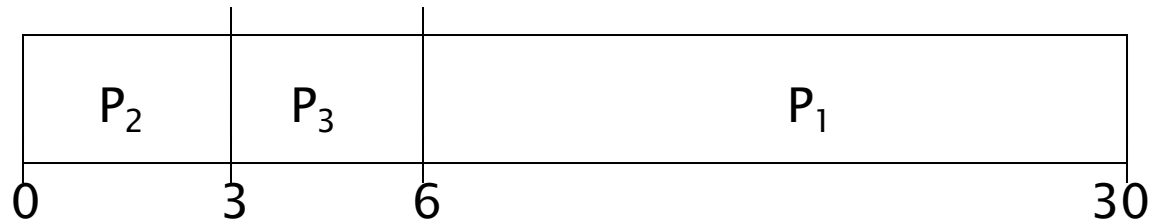
$$\text{Average turnaround time} = (24 + 27 + 30)/3 = 81 / 3 = 27\text{ms}$$

FCFS Scheduling

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:** All small processes have to wait for one big process to get off the cpu.
- ▶ **Alternative :** Allow shorter processes to go first

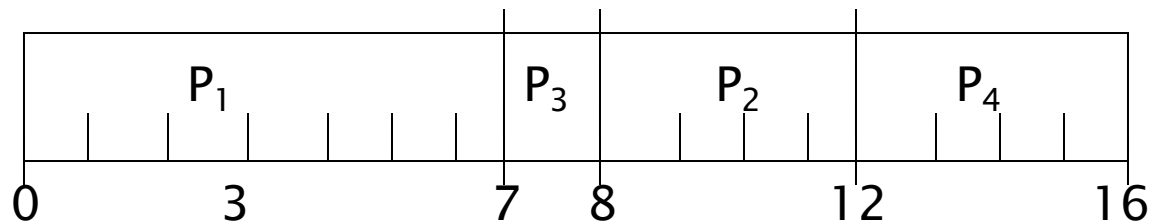
2. Shortest-Job-First (SJF) Scheduling

- ▶ Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- ▶ Two schemes:
 - **Non-preemptive** – Once the CPU is given to the process it cannot be preempted until it completes its CPU burst. i.e once the CPU has been allocated to a process, it can keep the cpu until it wants to release it.
 - **Preemptive** – If a new process arrives with CPU burst length less than remaining time of currently executing process then preempt it. This scheme is know as the **Shortest-Remaining-Time-First (SRTF)**.

Example of Non-Preemptive SJF

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- ▶ SJF (non-preemptive)

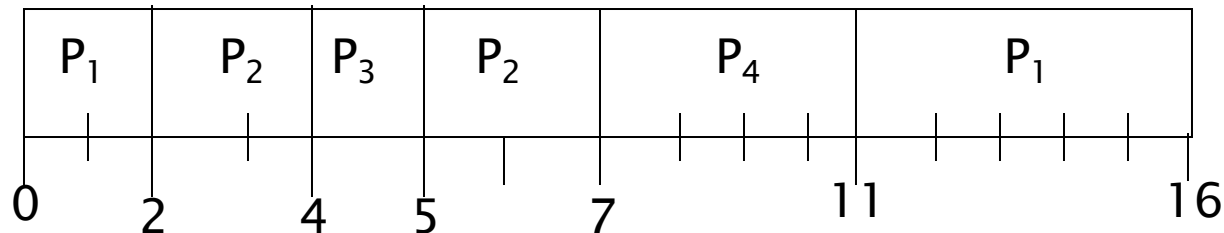


- ▶ Average waiting time = $(0 + (8-2) + (7-4) + (12-5))/4 = 4\text{ms}$
- ▶ Average turnaround time = $((7-0)+(12-2) + (8-4)+(16-5)) = 8\text{ ms}$

Example of Preemptive SJF(SRT)

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P ₁	0.0	7
P ₂	2.0	4
P ₃	4.0	1
P ₄	5.0	4

▶ SJF (preemptive)

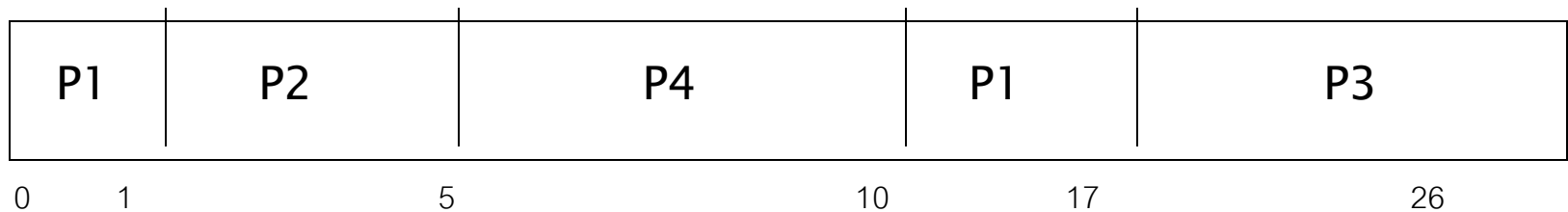


- ▶ Average waiting time = $((11-2) + 1 + 0 + 2)/4 = 3\text{ms}$
- ▶ Average turnaround time = $(16 + (7-2) + (5-4) + (11-5))/4 = 7\text{ms}$

Example

- Shortest-Remaining-Time-First Scheduling(SRTF)

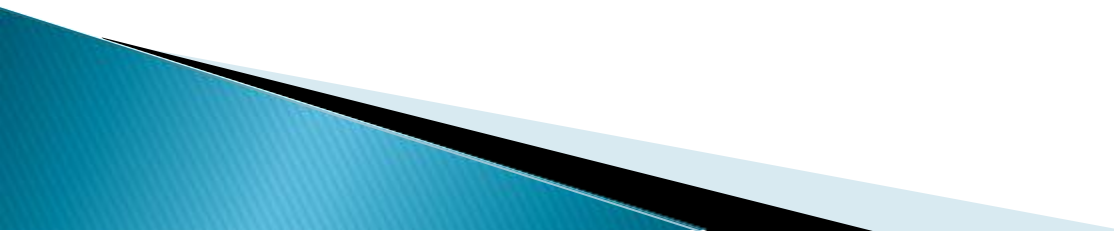
Process	Arrival time	Burst Time
P1	0	8 ms
P2	1	4 ms
P3	2	9 ms
P4	3	5 ms



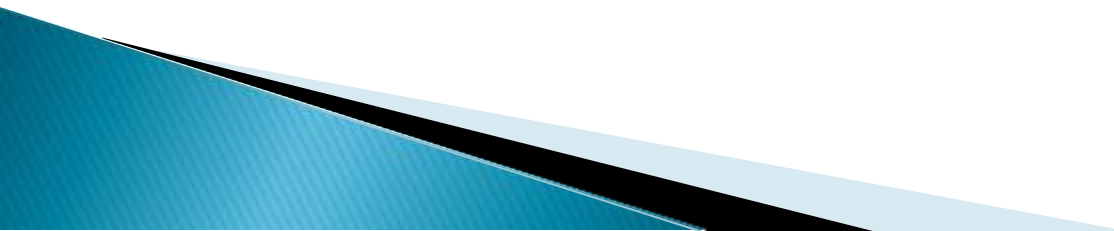
$$\text{Average WT} = ((10-1)+(1-1)+(17-2)+(5-2))/4 = 6.5 \text{ ms}$$

$$\text{Average TT} = ((17-0)+(5-1)+(26-2)+(10-3))/4 = 13 \text{ ms}$$

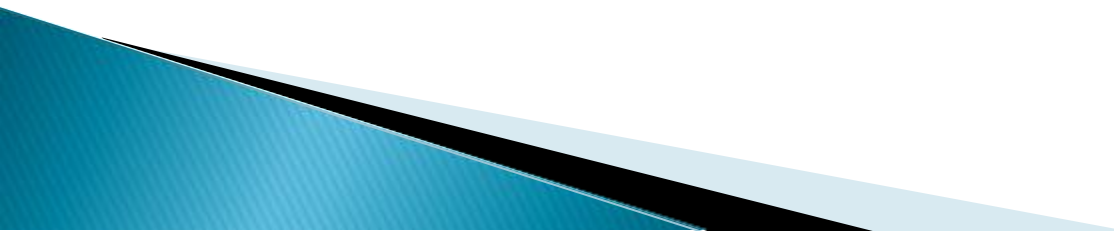
3. Priority algorithm

- ▶ A priority number (integer) is associated with each process.
 - ▶ CPU is allocated to the process with the highest priority (smallest integer \equiv highest priority).
 - ▶ SJF is a priority algorithm with shorter CPU burst having more priority.
 - ▶ **Drawback** : Can leave some low priority processes waiting indefinitely for the CPU. This is called starvation.
 - ▶ **Solution** to the problem of indefinite blockage of low priority jobs is **aging**.
- 

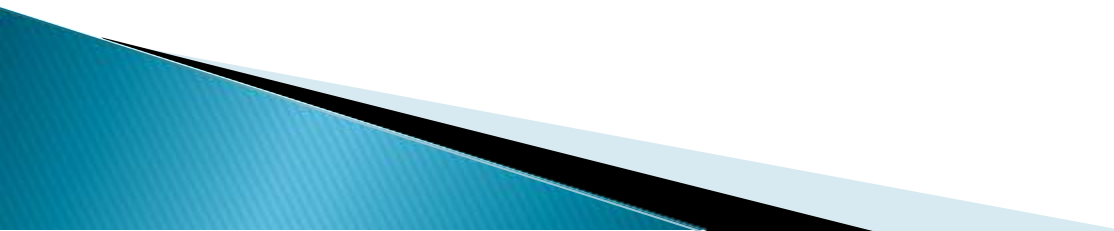
Priority algorithm

- ▶ Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.
 - ▶ For example, if priorities range from 127 (low) to 0 (high), we could increase the priority of a waiting process by 1 every 15 minutes.
 - ▶ Eventually, even a process with an initial priority of 127 would have the highest priority in the system and would be executed.
- 

Priority algorithm

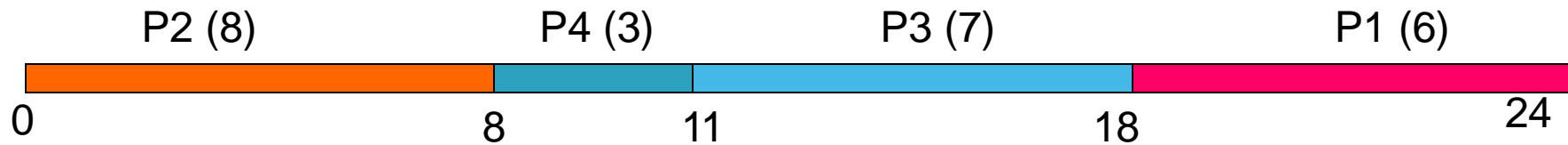
- ▶ Priorities can be defined either internally or externally.
 - ▶ Internally defined priorities use some measurable quantity or quantities to compute the priority of a process. For example, memory requirements of each process.
 - ▶ External priorities are set by criteria outside the OS, such as the importance of the process, the type of process (system/user) etc.
- 

Priority algorithm

- ▶ Priority scheduling can be either pre-emptive or non preemptive.
 - ▶ When a process arrives at the ready queue, its priority is compared with the priority of the currently running process.
 - ▶ A **pre-emptive 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 **non preemptive priority scheduling** algorithm will simply continue with the current process
- 

Priority Scheduling: Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0



P2 waiting time: 0
P4 waiting time: 8
P3 waiting time: 11
P1 waiting time: 18

The average waiting time (AWT):
 $(0+8+11+18)/4 = 9.25\text{ms}$

Priority algorithm

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

P2	P5	P1	P3	P4
0	1	6	16	18 19

Average wait time = $(6+0+16+18+1)/5 = 8.2$ ms

Turnaround time = $(16 + 1 + 18 + 19 + 6)/5 = 12$ ms

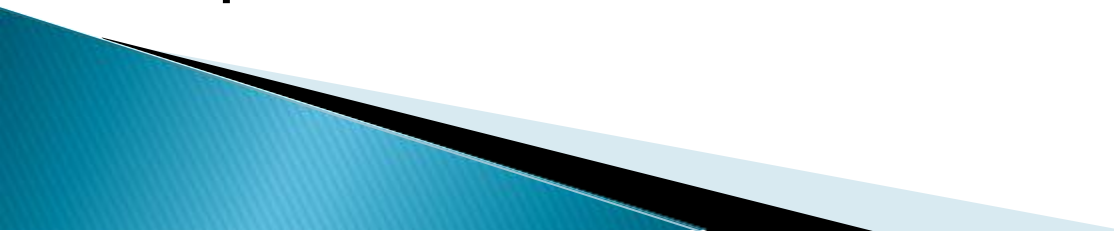
	Arrival Time	Burst Time	Priority
P1	0	5	3
P2	2	6	1
P3	3	3	2

| p1 | p2 | p3 | p1 |
0 2 8 11 14

Average waiting time $= (9 + 0 + 5) / 3 = 4.66$ ms

Average turn around time $= (14 + 6 + 8) / 3 = 9.33$ ms

4. Round Robin Algorithm

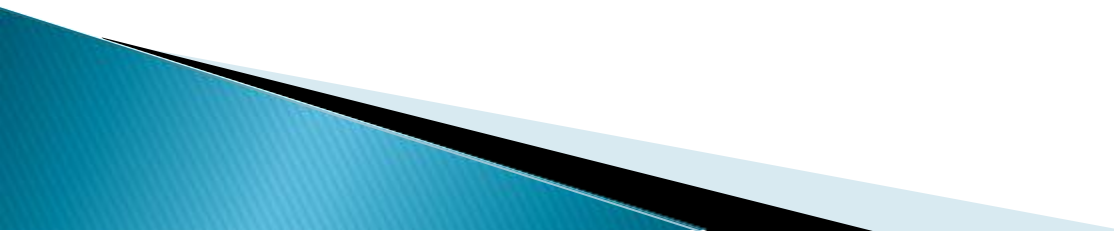
- ▶ The round-robin (RR) scheduling algorithm is designed especially for time-sharing systems.
 - ▶ It is similar to FCFS scheduling, but pre-emption is added to switch between processes.
 - ▶ A small unit of time, called a time quantum or time slice, is defined.
 - ▶ A time quantum is generally from 10 to 100 milliseconds.
 - ▶ The ready queue is treated as a circular queue.
- 

Round Robin Algorithm

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.
- ▶ 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 OS.
 - 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.

Round Robin Algorithm

- ▶ The performance of the RR algorithm depends heavily on the size of the time quantum. If the time quantum is extremely large, the RR policy is the same as the FCFS policy.
 - ▶ If the time quantum is extremely small the RR approach is called processor sharing and (in theory) creates the appearance that each of the n users has its own processor running at $1/n$ th the speed of the real processor.
- 

- Shorter response time
- Fair sharing of CPU

Example of RR with Time Quantum = 20

<u>Process</u>	<u>Burst Time</u>
----------------	-------------------

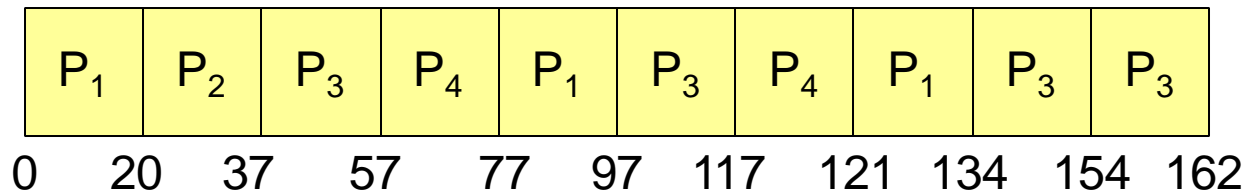
P_1	53
-------	----

P_2	17
-------	----

P_3	68
-------	----

P_4	24
-------	----

- ▶ The Gantt chart is:



- ▶ Average turn-around time = $(134 + 37 + 162 + 121) / 4 = 113.5$ ms

<u>Process</u>	<u>Burst Time</u>	<u>Wait Time</u>
P_1	53	$57 + 24 = 81$
P_2	17	20
P_3	68	$37 + 40 + 17 = 94$
P_4	24	$57 + 40 = 97$

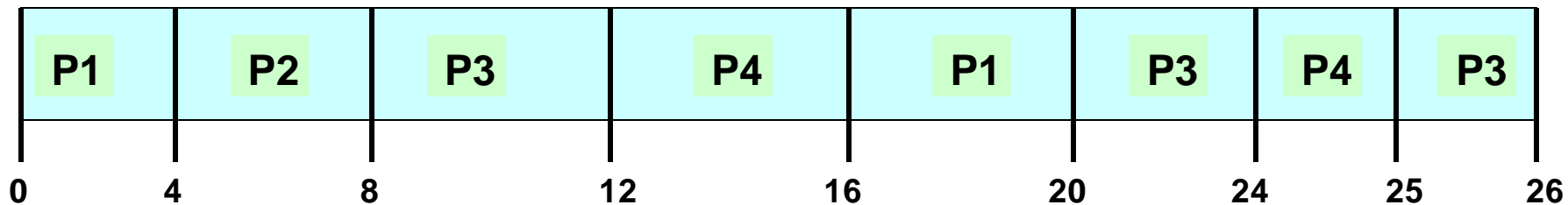
- ▶ Average wait time = $(81 + 20 + 94 + 97) / 4 = 73$ ms

Round robin

▶ EXAMPLE DATA:

	Process	Arrival Time	CPU Time
○	1	0	8
○	2	1	4
○	3	2	9
○	4	3	5

- Round Robin, quantum = 4



$$\text{Average TAT} = ((20-0) + (8-1) + (26-2) + (25-3))/4 = 73/4 = 18.25\text{ms}$$

$$\text{Average WT} = (16-4) + (4-1) + (8+(20-12)+(25-24)-2) + (12+(24-16)-3) = 11.75\text{ms}$$

Examples

1. FCFS

Process	Arrival Time	Exec. Time
P1	0	5
P2	2	4
P3	3	7
P4	5	6



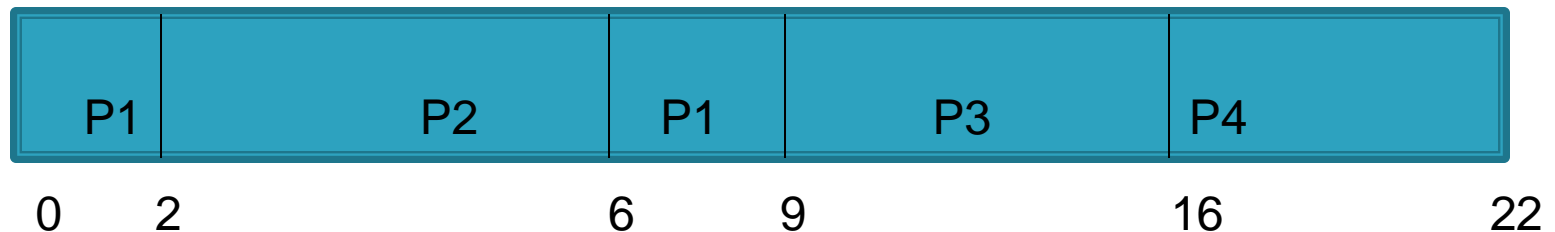
Avg WT = $(0 + (5-2) + (9-3) + (16-5))/4 = 5$ time units

Avg TAT = $((5-0) + (9-2) + (16-3) + (22-5))/4 = 10.5$ time units

Examples

2. Priority preemptive

Process	Arrival Time	Exec. Time	Priority
P1	0	5	2
P2	2	4	1
P3	3	7	3
P4	5	6	4



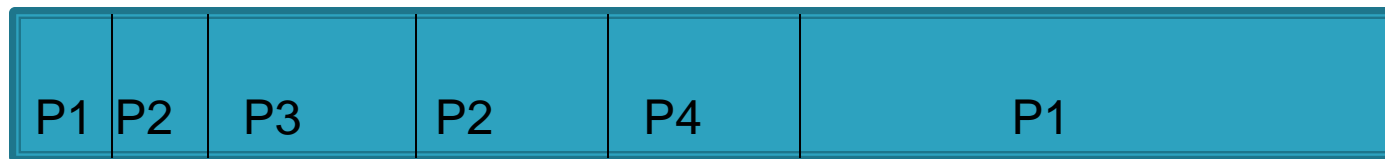
Avg WT = $((0+4) + 0 + (9-3) + (16-5)) / 4 = 5.25$ time units

Avg TAT = $((9-0) + (6-2) + (16-3) + (22-5)) / 4 = 10.75$ time units

Examples

3. SRTF

Process	Arrival Time	Exec. Time
P1	0	9
P2	1	5
P3	2	3
P4	3	4



$$\text{Avg WT} = \frac{(12 + 3 + 0 + 6)}{4} = 5.25 \text{ time units}$$

$$\text{Avg TAT} = \frac{(21 + 8 + 3 + 10)}{4} = 10.5 \text{ time units}$$

Multilevel Queue Scheduling

- ▶ General class of algorithms involving *multiple* ready queues
- ▶ Appropriate for situations where processes are easily classified into different groups (e.g., foreground and background)
- ▶ Processes permanently assigned to one ready queue depending on some property of process.
- ▶ Each queue has its own scheduling algorithm
 - foreground – RR
 - background – FCFS
- ▶ Scheduling as well between the queues—often a **priority preemptive scheduling**. For example, foreground queue could have absolute priority over background queue. (New foreground jobs displace running background jobs ; serve all from foreground then from background).

Multilevel Feedback Queue

Three queues:

- ▶ $Q0$ – RR with time quantum 8 milliseconds
- ▶ $Q1$ – RR time quantum 16 milliseconds
- ▶ $Q2$ – FCFS

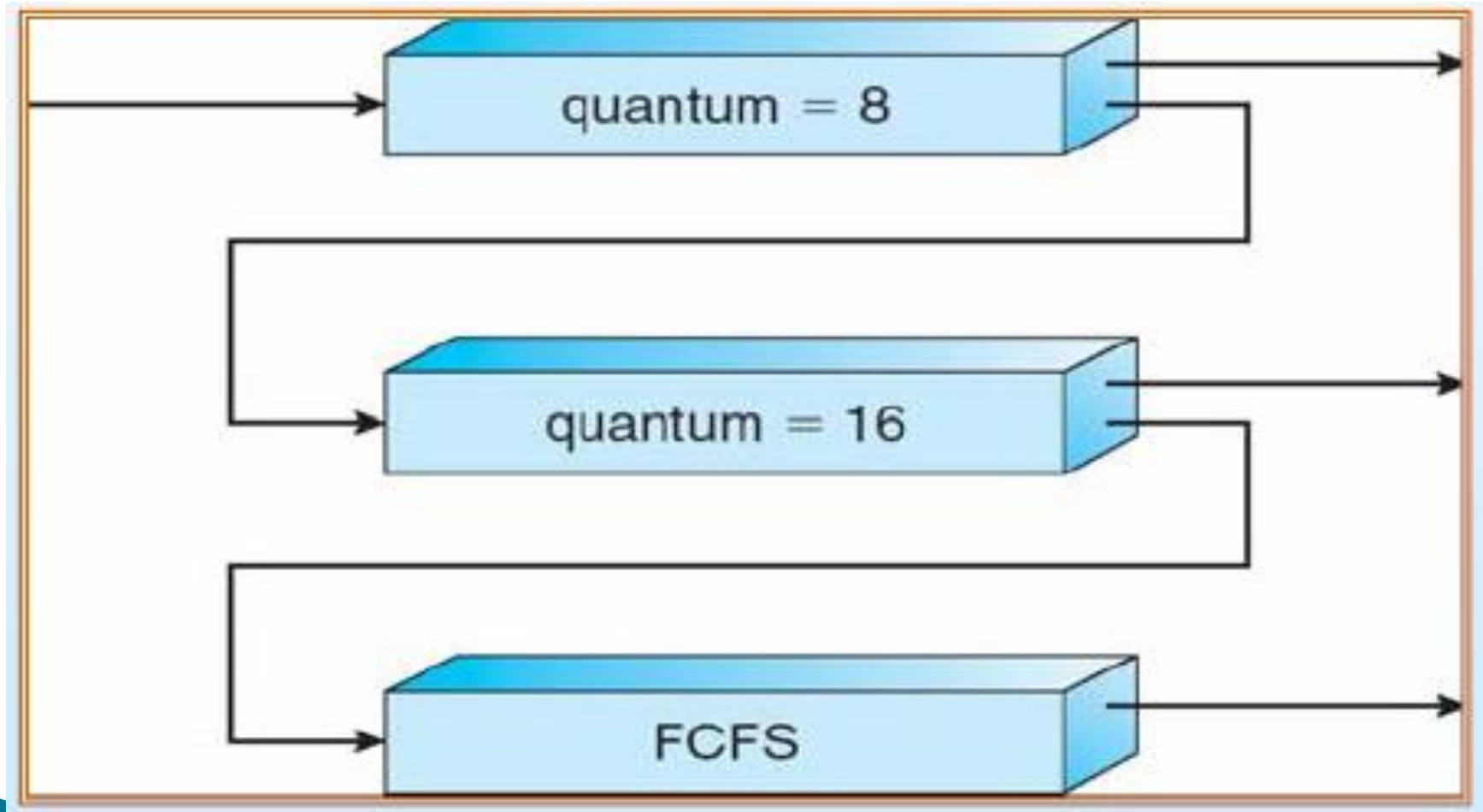
Scheduling

A new job enters queue $Q0$ 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 $Q1$.

At $Q1$ job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue $Q2$.

Long jobs automatically sink to $Q2$ and are served FCFS

Multilevel Feedback Queue



- ▶ For example, Windows NT/XP/Vista uses a multilevel feedback queue, a combination of fixed priority preemptive scheduling, round-robin, and first in first out.