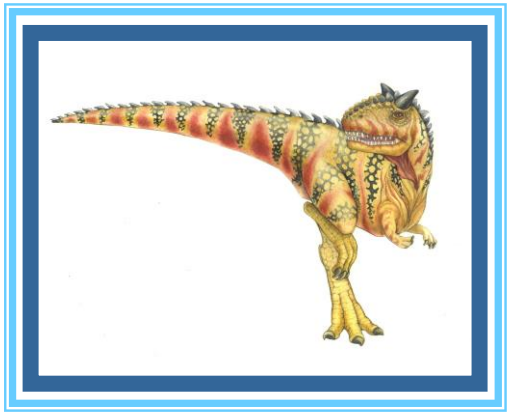
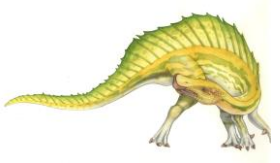
Chapter 5: CPU Scheduling



Silberschatz, Galvin and Gagne ©2018

Chapter 5: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation



Objectives

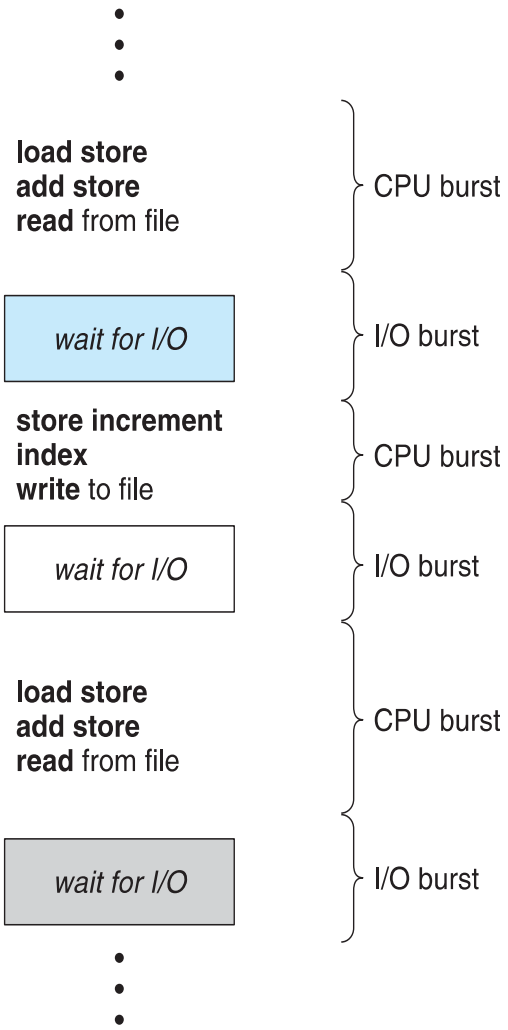
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

Basic Concepts

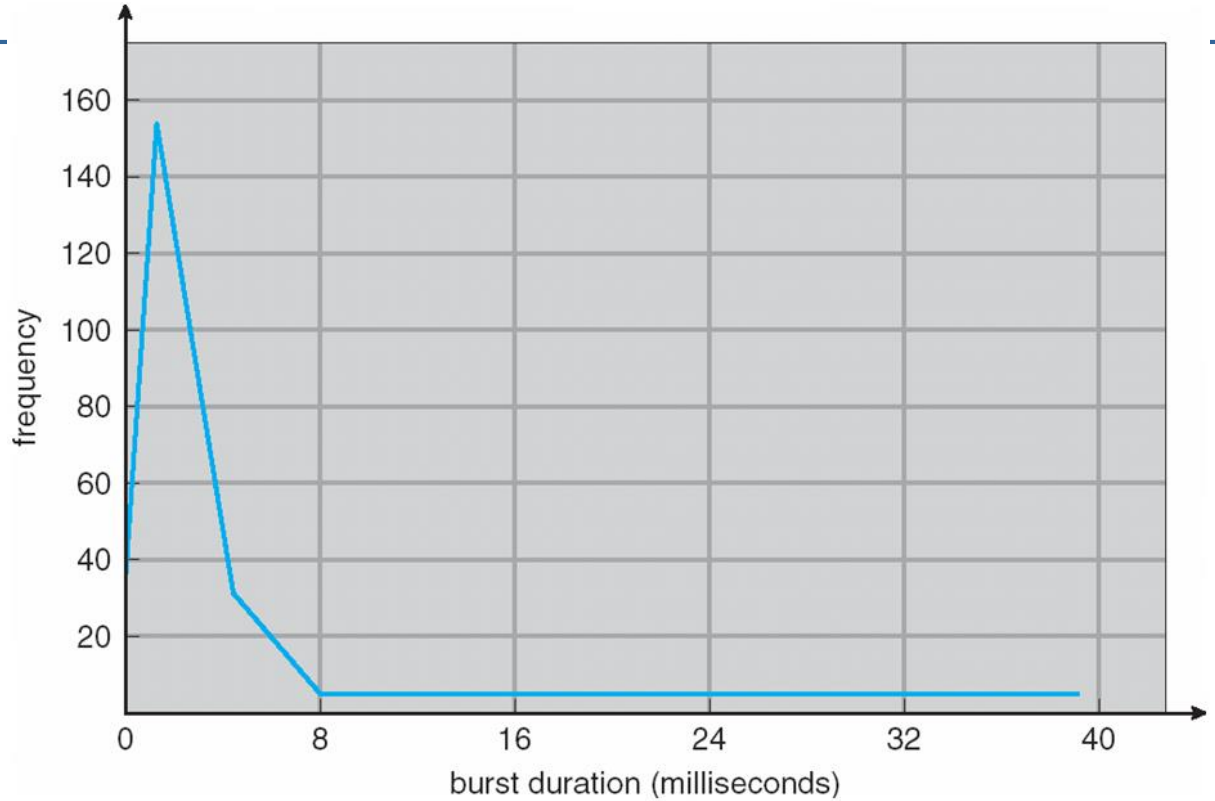
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern

CPU Burst Time: The time a process spends on the CPU executing instructions before it needs to either wait for I/O or is preempted by the scheduler.

I/O Burst Time: The time a process spends waiting for I/O operations to complete, during which it does not use the CPU.

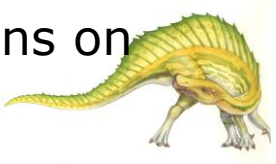


Histogram of CPU-burst Times



CPU Burst Distribution is of Main Concern

- The distribution of CPU bursts is important for **scheduling** decisions.
- Most processes tend to have short CPU bursts (they do a little computation, then wait for I/O)
- A few processes have long CPU bursts.
- Operating systems use this information to make better decisions on scheduling processes.



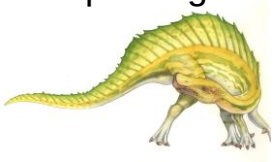
CPU Scheduling Policies

Categories of scheduling policies:

- Non-Preemptive -- no interruptions are allowed. A process completes execution of its CPU burst
- Preemptive – a process can be interrupted before the process completes its CPU burst

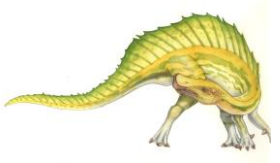
CPU Scheduler

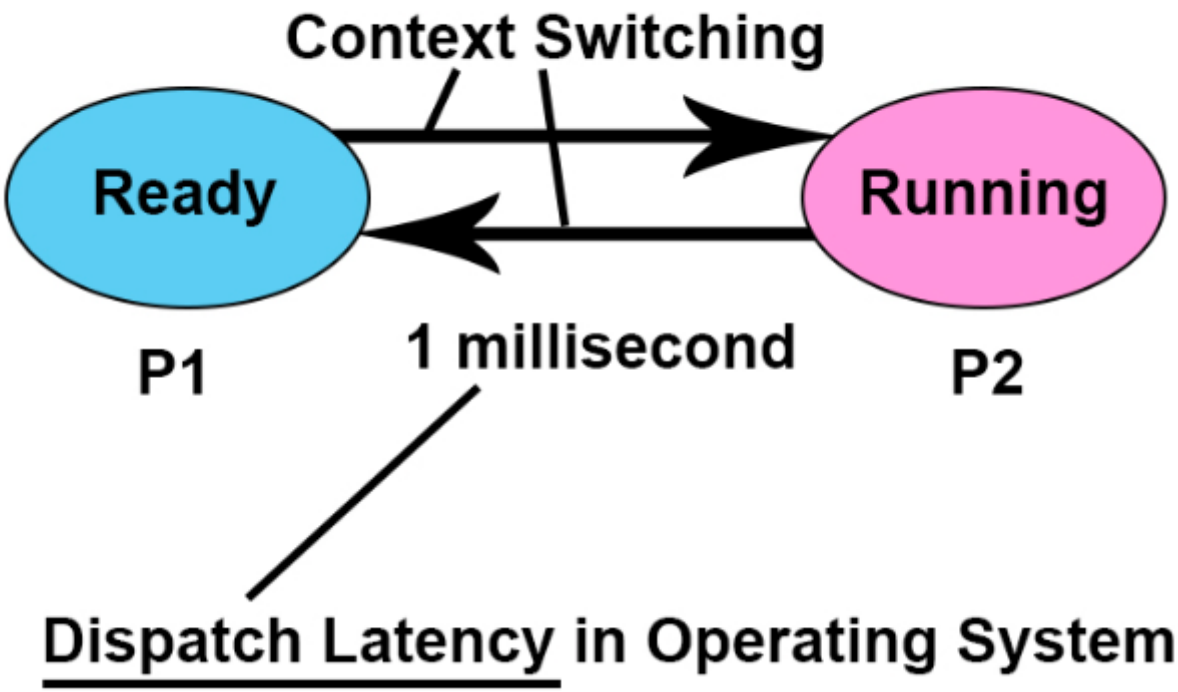
- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive:** simpler but less responsive
- All other scheduling is **preemptive:** improves responsiveness but comes with **challenges:**
  - Consider access to shared data
    - Inconsistent states or race conditions may occur.
  - Consider preemption while in kernel mode
    - Kernel operations might involve sensitive tasks like managing device controllers or memory
  - Consider interrupts occurring during crucial OS activities
    - Can allow interrupts to occur at any point, including during critical OS operations like updating the process control block (PCB).



Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
    - When the CPU switches from one process to another, it has to save the current state of the running process and load the state of the next process.
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
    - Setting the **program counter** to the proper instruction in the process's memory.
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running




Context Switching
Ready
Running
P1
1 millisecond
P2
Dispatch Latency in Operating System

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
  - In a time-sharing system, there are 10 processes. If the CPU spends 90% of the time **executing** these processes and 10% of the time **idle**, the CPU utilization is 90%.
  - Low utilization: not efficiently using the available resources, leading to potential slowdowns or wasted capacity.
- **Throughput** – # of processes that complete their execution per time unit
  - algorithm allows 5 processes to finish in 10 seconds.
  - The **throughput** is 0.5 processes per second.
- **Turnaround time** – amount of time to execute a particular process
  - **Completion Time** - **Arrival Time**
- **Waiting time** – amount of time a process has been waiting in the ready queue
  - **Turnaround Time** - **Burst 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)

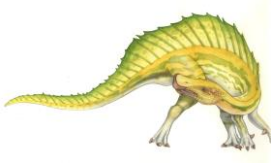
Example

Process	Arrival Time (ms)	Burst Time (ms)	Start Time (ms)	Completion Time (ms)	Turnaround Time (ms)	Waiting Time (ms)	Response Time (ms)
P1	0	10	0	10	10	0	0
P2	2	5	10	15	13	8	8
P3	4	8	15	23	19	11	11

$$\text{CPU Utilization} = \frac{23}{23} \times 100 = 100\%$$

$$\text{Throughput} = \frac{3}{23} = 0.1304 \,\text{processes per millisecond}$$

- Turnaround Time = Completion Time−Arrival Time
- Waiting Time = Turnaround Time−Burst Time
- Response Time = First Response Time−Arrival Time



Scheduling Algorithm Optimization Criteria

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

First- Come, First-Served (FCFS) Scheduling

Process	Burst 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

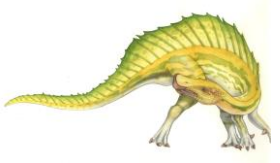
FCFS Scheduling (Cont.)

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



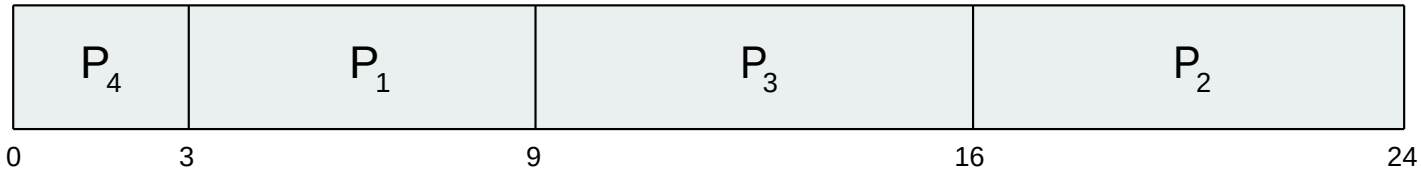
Shortest-Job-First (SJF) Scheduling

- 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

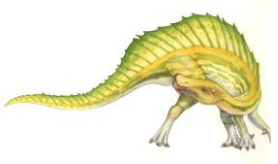
Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



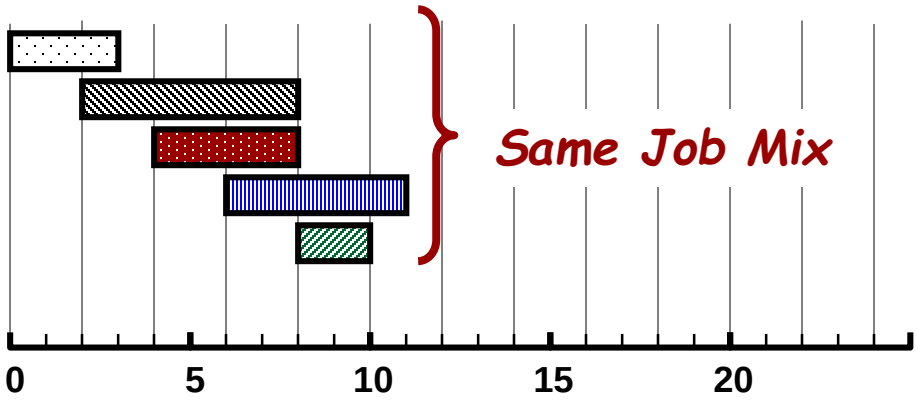
- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

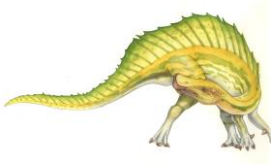


Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

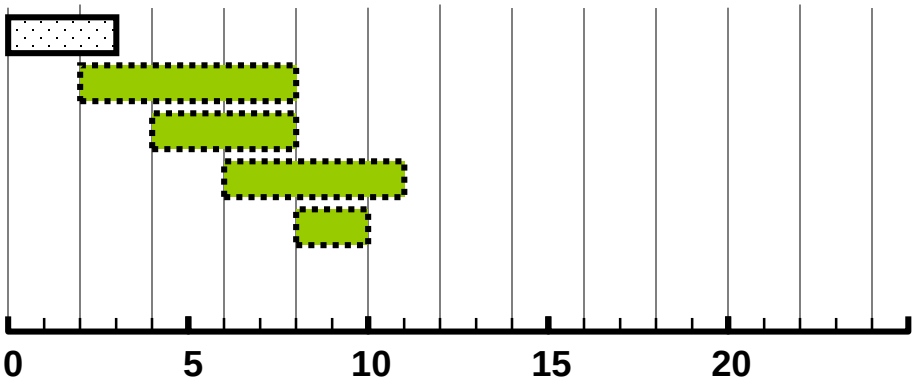




Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

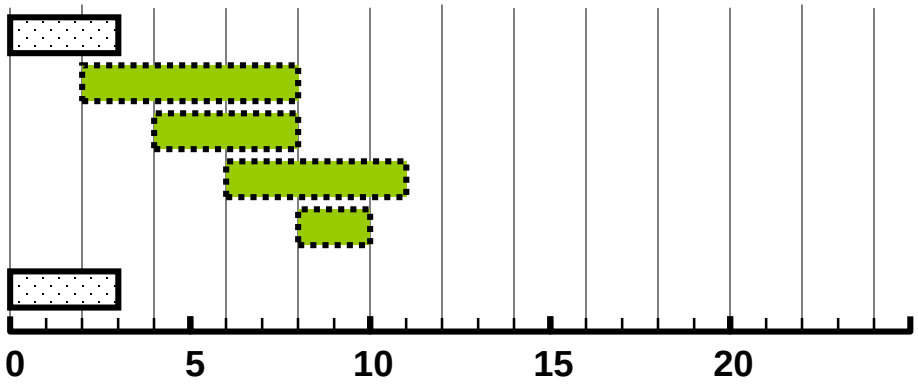


18

Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

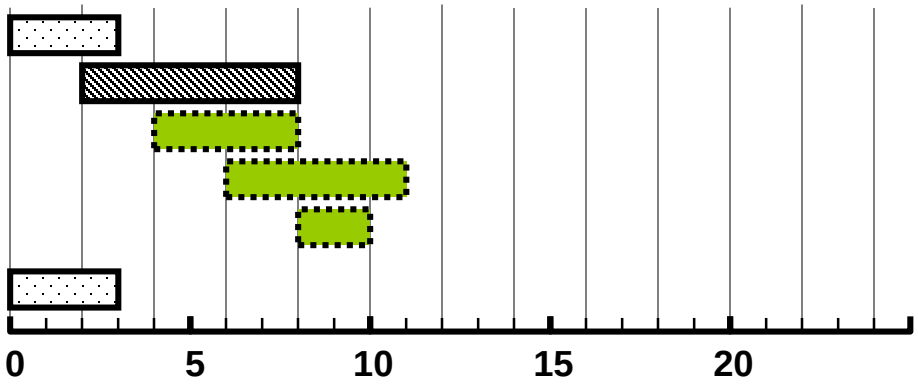


19

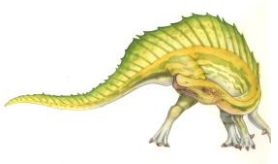
Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



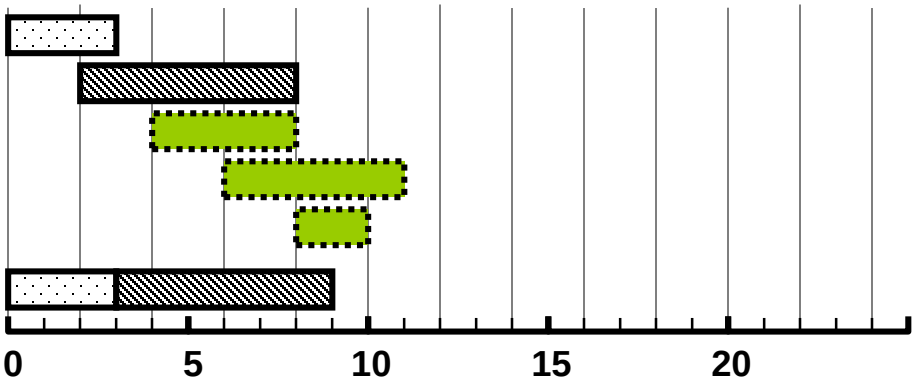
20



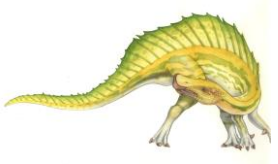
Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



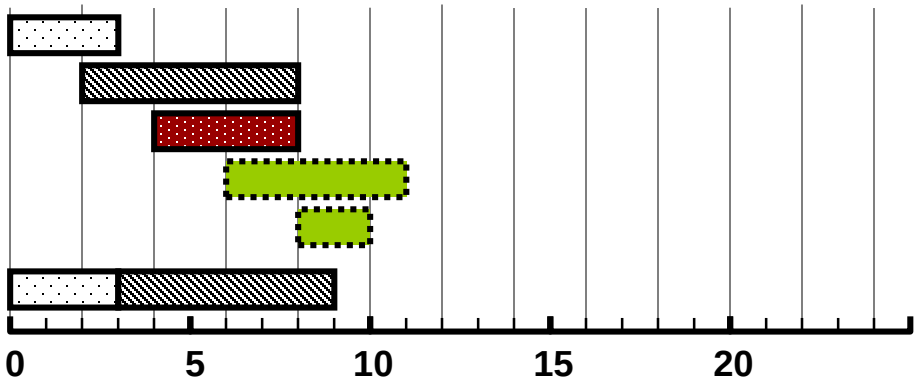
21



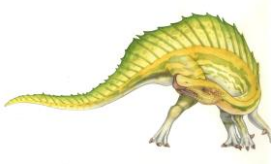
Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



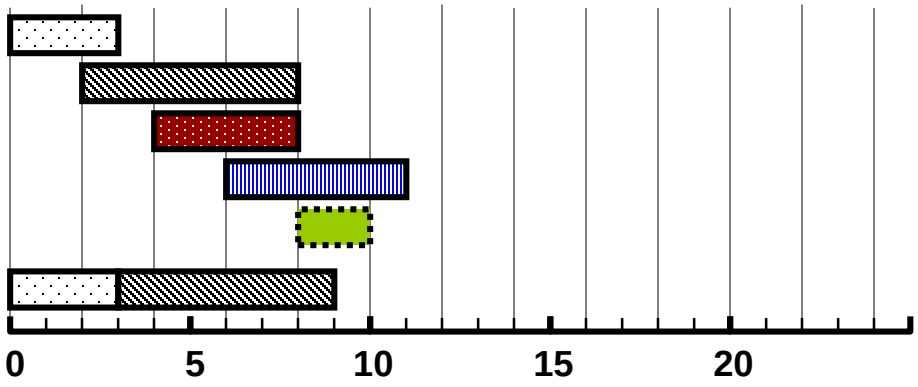
22



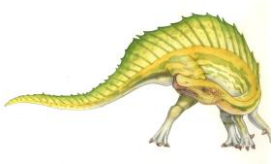
Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



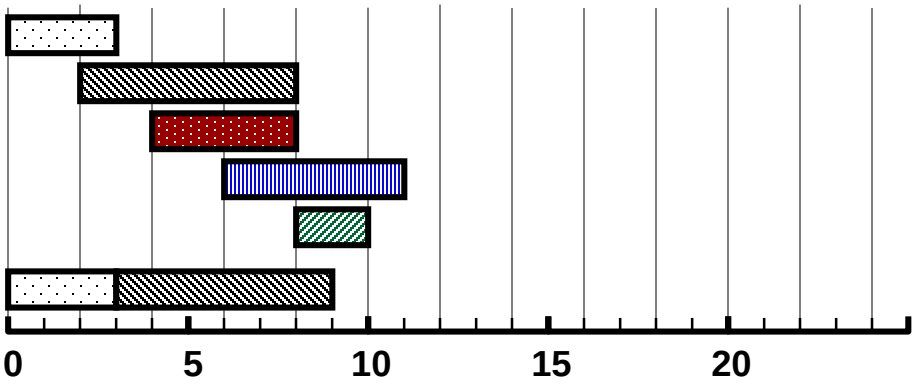
23



Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

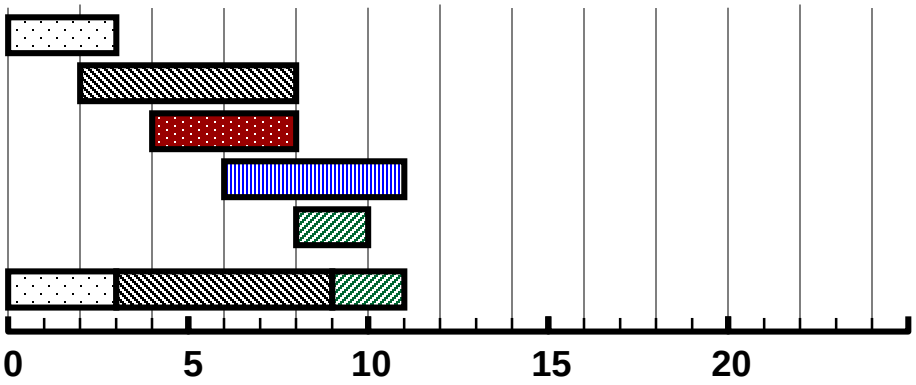


24

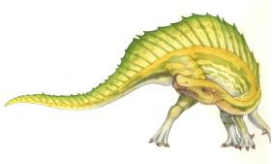
Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



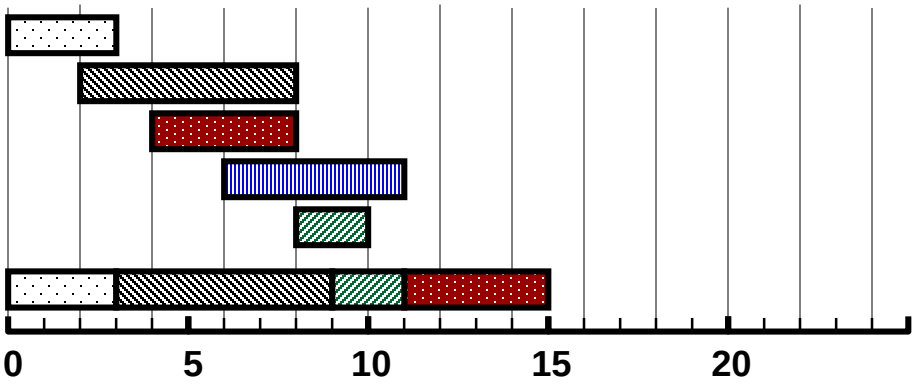
25



Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

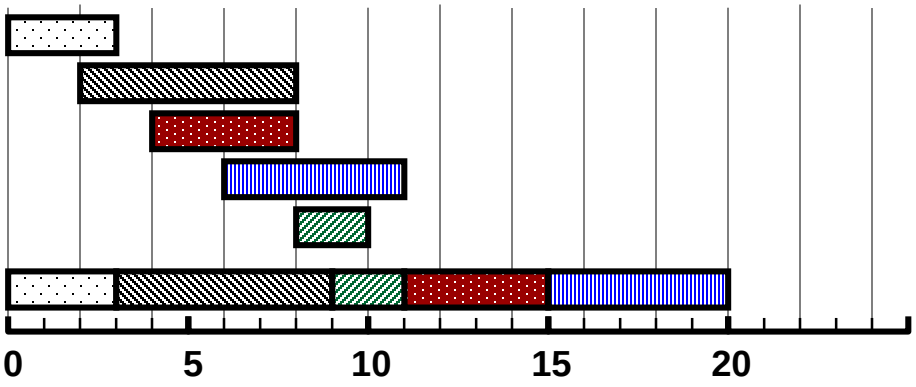


26

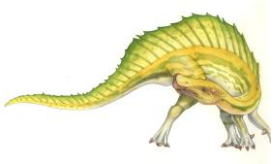
Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



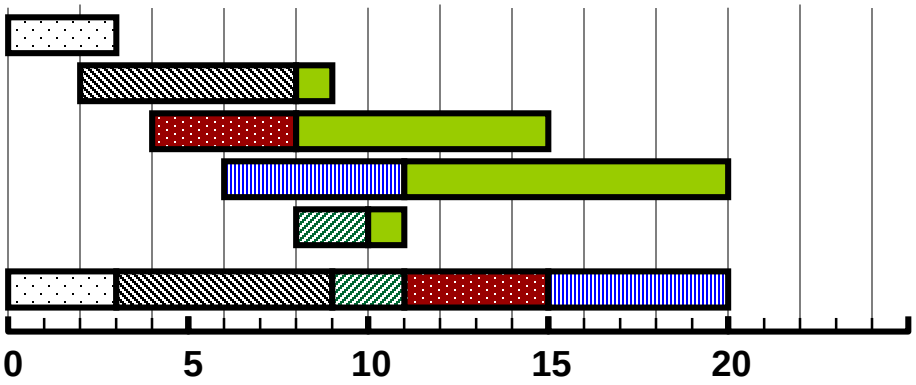
27



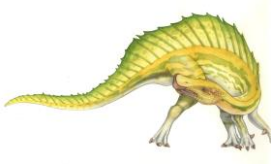
Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



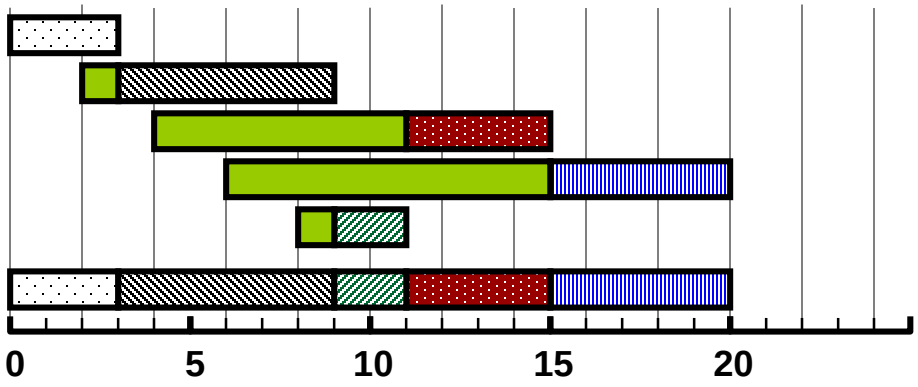
28



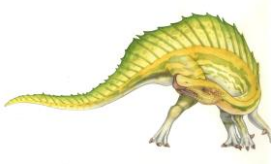
Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



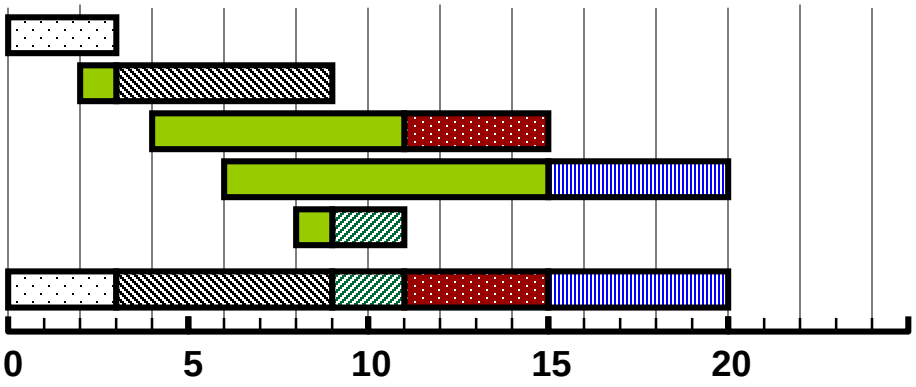
29



Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3	0	
2	2	6	1	
3	4	4	7	
4	6	5	9	
5	8	2	1	

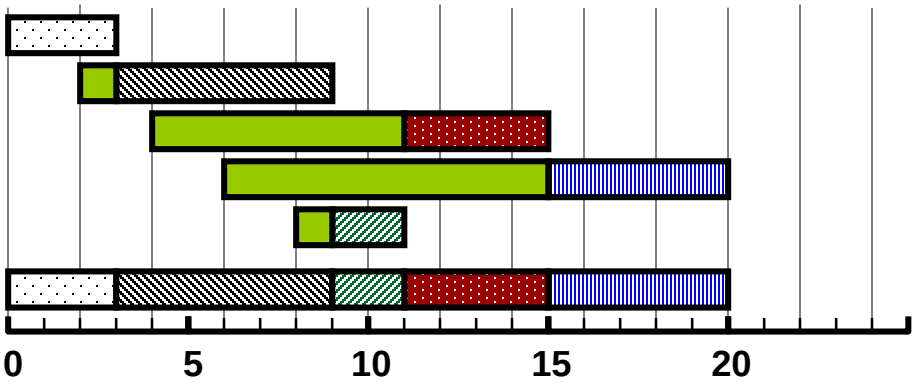


30

Shortest Job First

- Select the job with the shortest (expected) running time
- Non-Preemptive

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3	0	3
2	2	6	1	7
3	4	4	7	11
4	6	5	9	14
5	8	2	1	3



Determining Length of Next CPU Burst

- **Exponential averaging**: is a method for predicting future values based on **past** data, with more weight given to the most recent data.

$$\tau_{n+1} = \alpha \cdot T_n + (1 - \alpha) \cdot \tau_n$$

Where:

- τ_{n+1}: The estimated value for the **next CPU burst**.
- T_n: The actual value of the **most recent CPU burst**.
- τ_n: The **previously estimated CPU burst time**.
- α: A constant between 0 and 1, representing the weight given to the most recent burst time versus past estimates.
- α determines the balance between the recent actual burst (T_n) and the historical estimate (τ_n)
  - If α is close to **1**, the next prediction will heavily rely on the most recent CPU burst time.
  - If α is close to **0**, the estimate will give more weight to past values and change less with new data.

Determining Length of Next CPU Burst

1. **Initial Burst Time (τ_0):**
   - Can be a fixed default value, a guess based on process type, or user-specified in real-time systems.
2. **Observed Burst Time (T_n):**
   - Measured by the operating system using the difference between the time a process starts and stops using the CPU.
3. **Refinement:**
   - After each observed burst, the OS refines its estimate of the next CPU burst using methods like exponential averaging:

$$\tau_{n+1} = \alpha \cdot T_n + (1 - \alpha) \cdot \tau_n$$

   - This improves future scheduling decisions.

Example:

Let's say you want to estimate the **next CPU burst time** for a process.

- **Initial estimate** (τ_0) = 10 ms (this is the first guess).
- **Actual burst times** for the process:
  - First burst (T_1) = 12 ms.
  - Second burst (T_2) = 9 ms.
  - Third burst (T_3) = 11 ms.

Let's set $\alpha = 0.5$ to give equal weight to both the actual recent burst and the previous estimate.

1. **After the first burst:**

$$\tau_1 = 0.5 \cdot 12 + 0.5 \cdot 10 = 6 + 5 = 11\,\text{ms}$$

   The new estimate for the next burst is 11 ms.

2. **After the second burst:**

$$\tau_2 = 0.5 \cdot 9 + 0.5 \cdot 11 = 4.5 + 5.5 = 10\,\text{ms}$$

   The new estimate for the next burst is 10 ms.

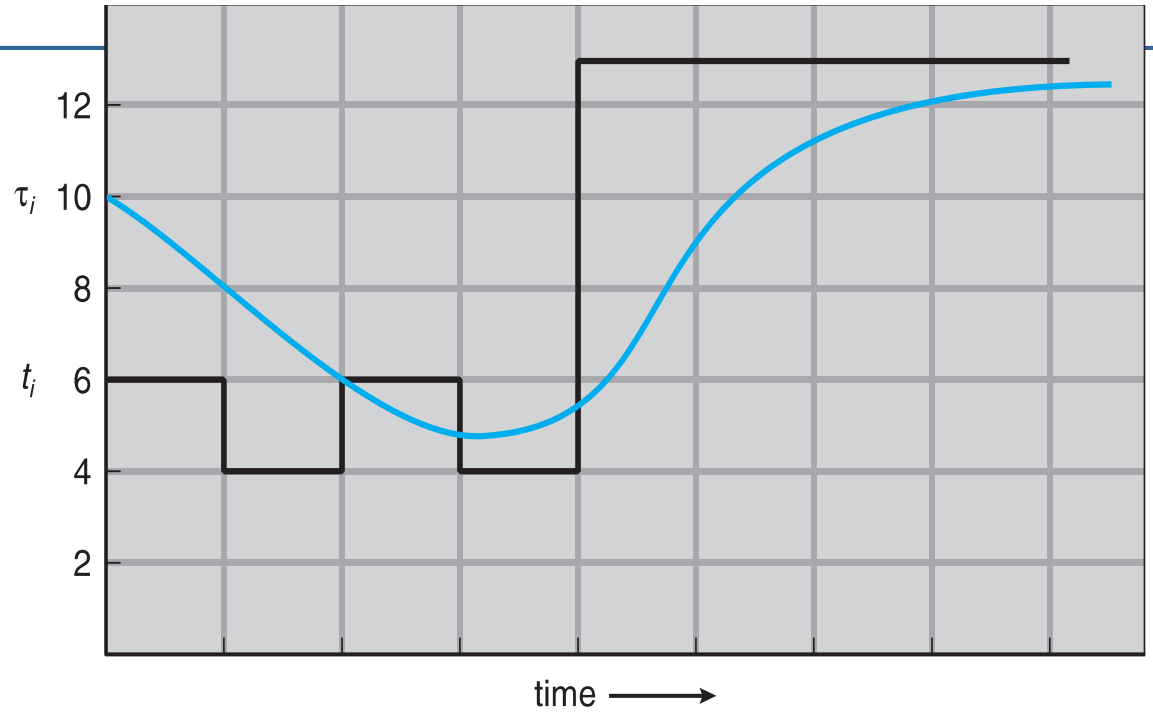
3. **After the third burst:**

$$\tau_3 = 0.5 \cdot 11 + 0.5 \cdot 10 = 5.5 + 5 = 10.5\,\text{ms}$$

   The new estimate for the next burst is 10.5 ms.

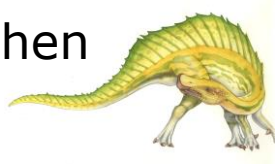
Gagne ©2018

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

- The blue line (predicted burst) lags behind the black line (actual burst).
- Gradually adjusts after each burst, improving the estimate for the upcoming CPU usage.
- Allow the system scheduler to make more efficient decisions when allocating CPU time.

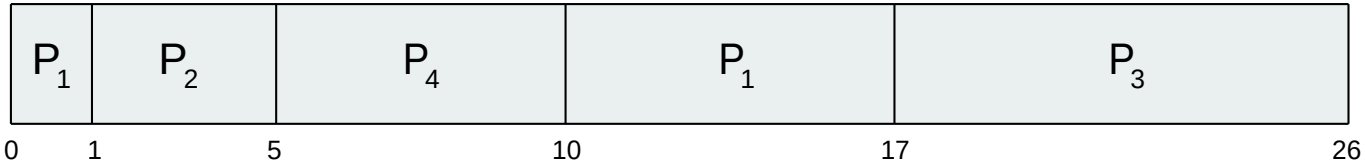


Example of Shortest-remaining-time-first

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

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

- *Preemptive* SJF Gantt Chart



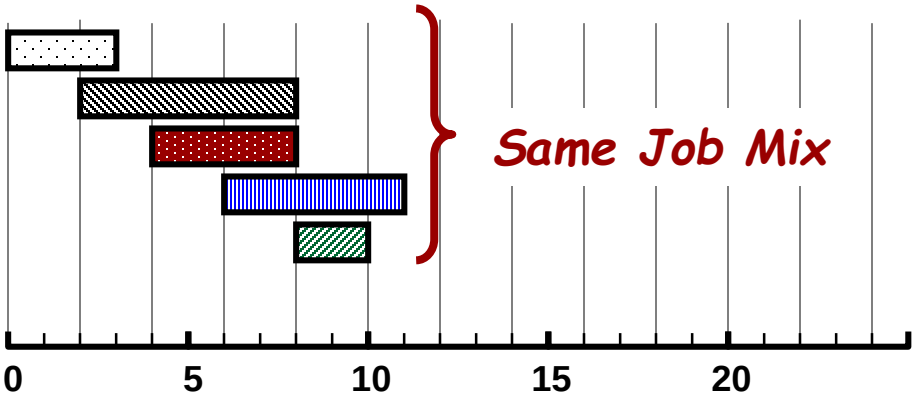
- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

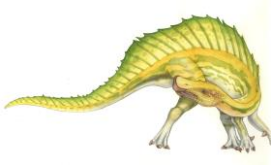
- Dynamically selects the process with the shortest remaining time, preempting the currently running process if a new process arrives with a shorter burst time.
- It can minimize waiting time but may lead to **starvation** if short jobs keep arriving, causing longer jobs to be delayed indefinitely.

Shortest Remaining Time

- Preemptive version of SJF

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

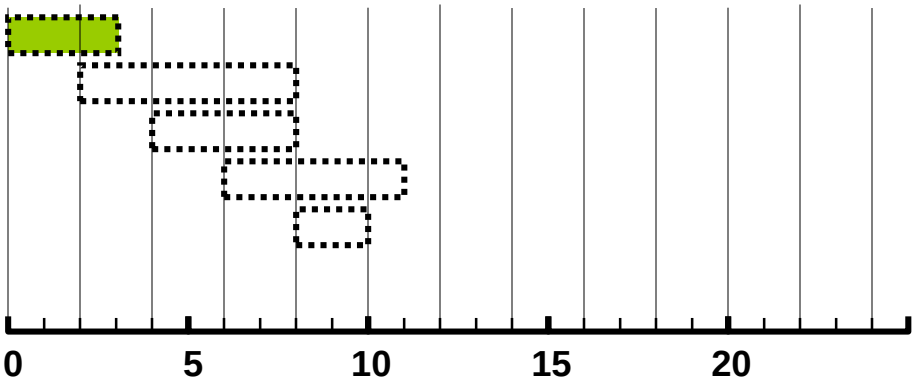




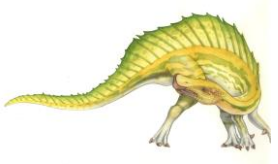
Shortest Remaining Time

- Preemptive version of SJF

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

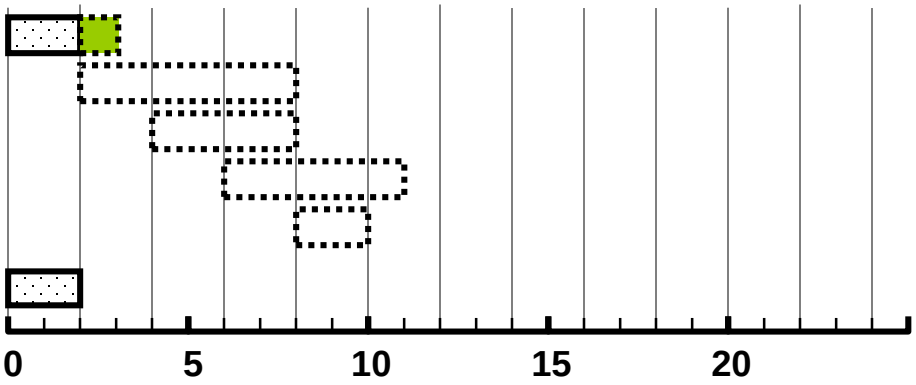


38

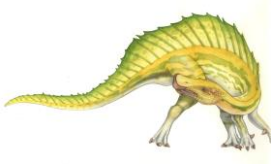


Shortest Remaining Time

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

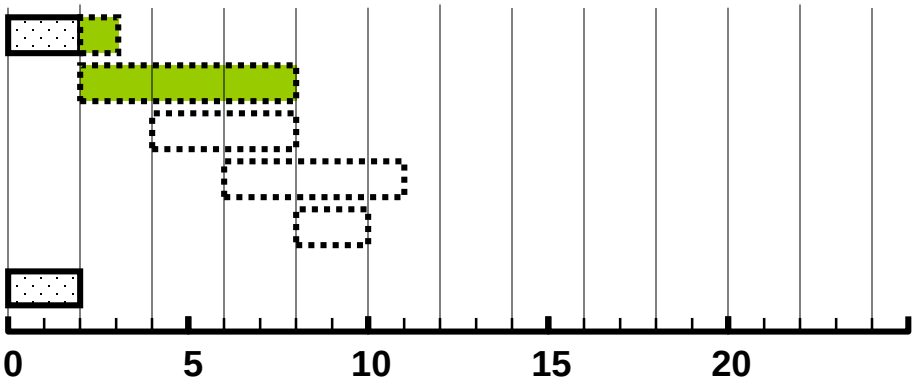


39

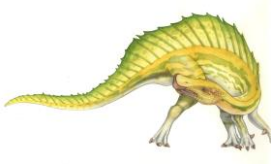


Shortest Remaining Time

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

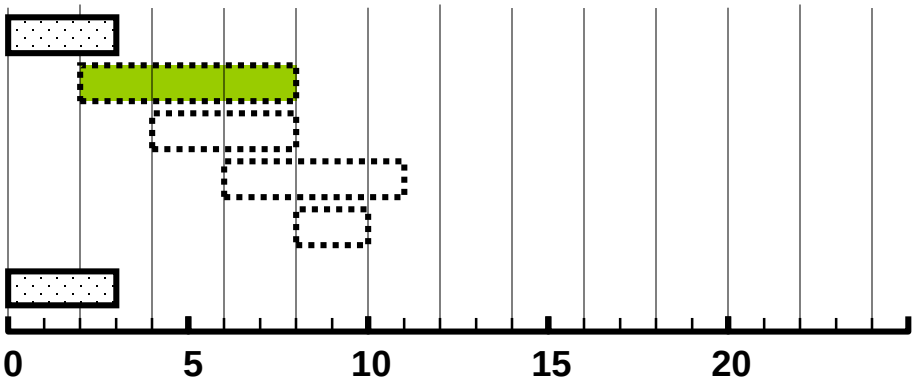


40

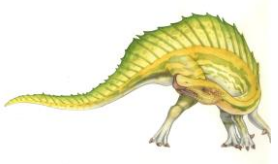


Shortest Remaining Time

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

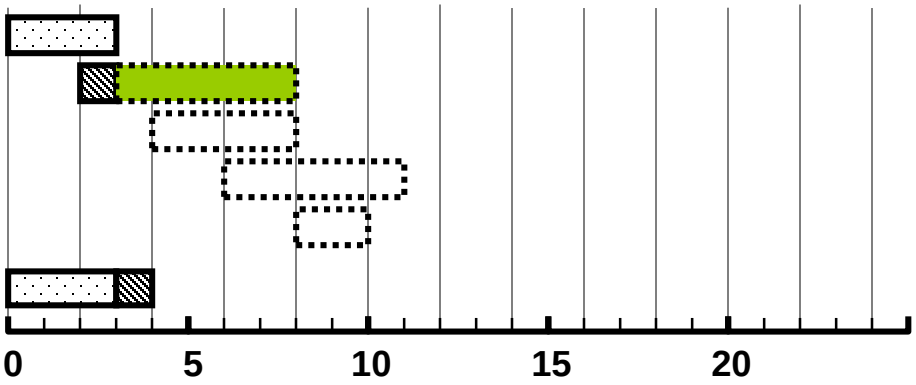


41



Shortest Remaining Time

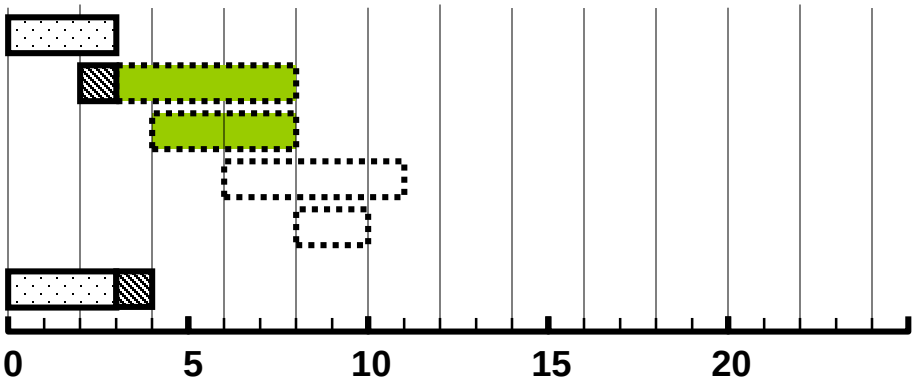
Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



42

Shortest Remaining Time

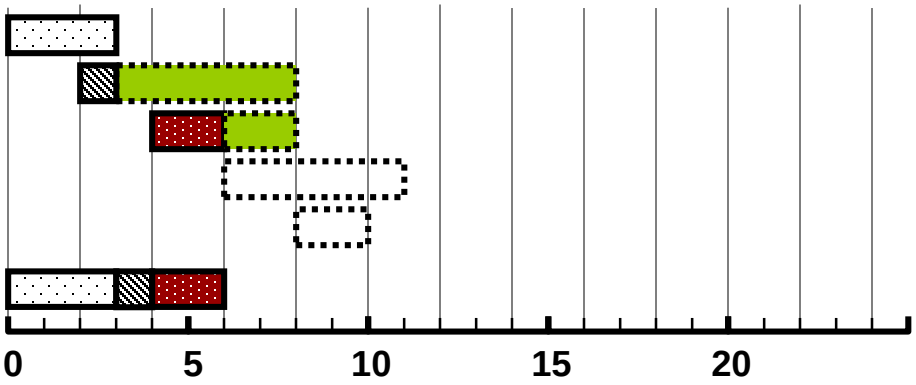
Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



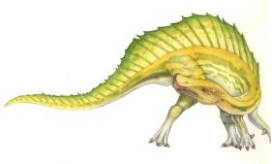
43

Shortest Remaining Time

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

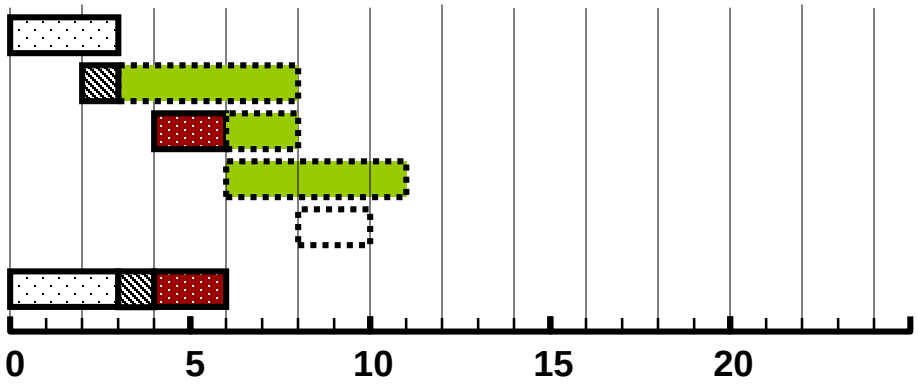


44



Shortest Remaining Time

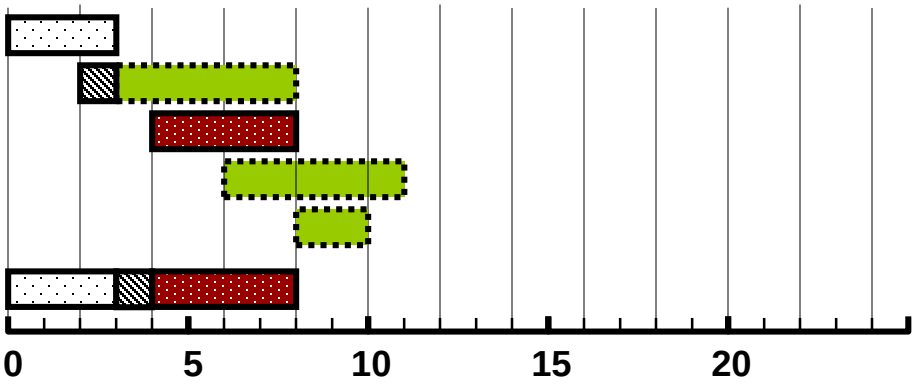
Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



45

Shortest Remaining Time

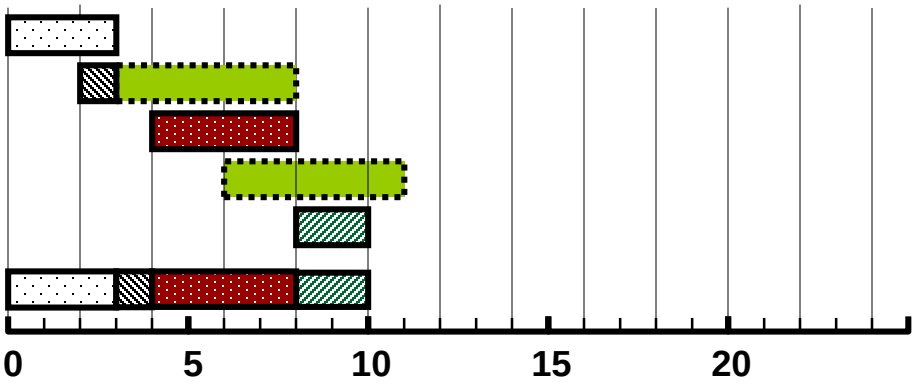
Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



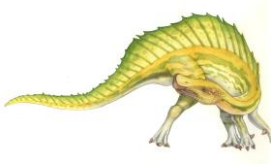
46

Shortest Remaining Time

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

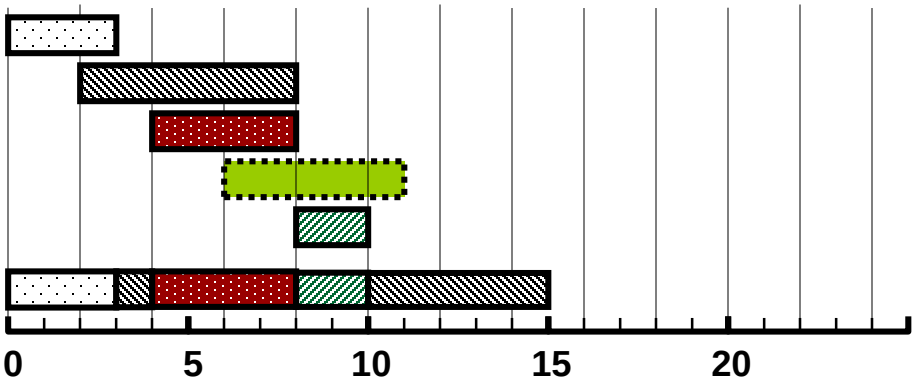


47



Shortest Remaining Time

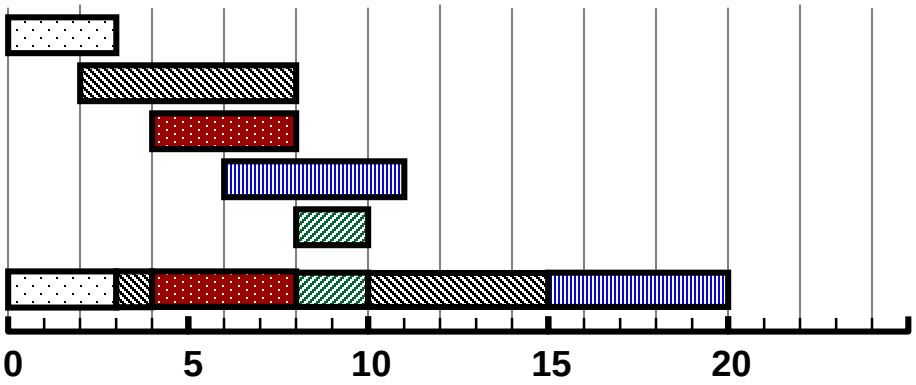
Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		



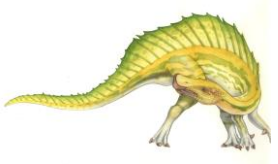
48

Shortest Remaining Time

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

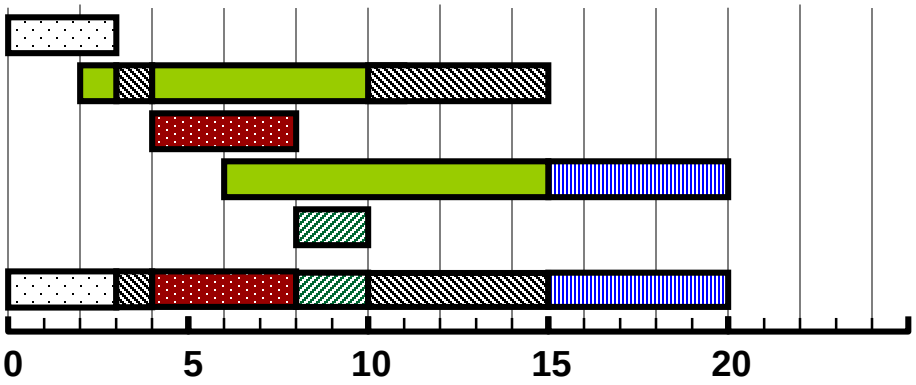


49

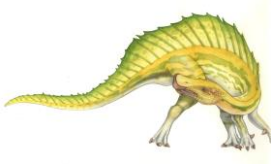


Shortest Remaining Time

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3		
2	2	6		
3	4	4		
4	6	5		
5	8	2		

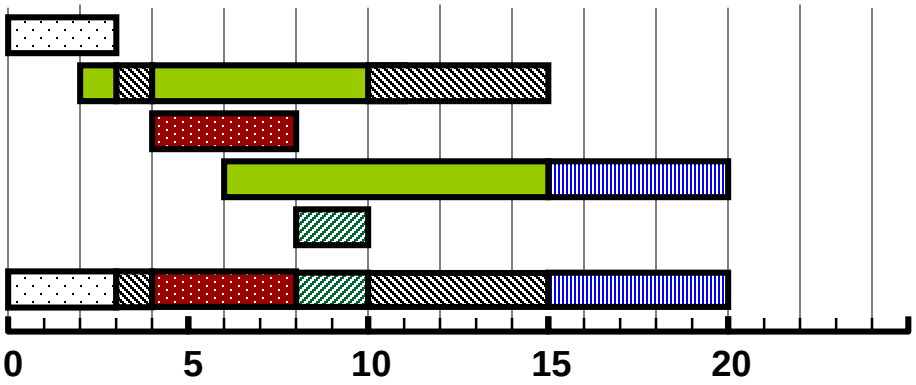


50



Shortest Remaining Time

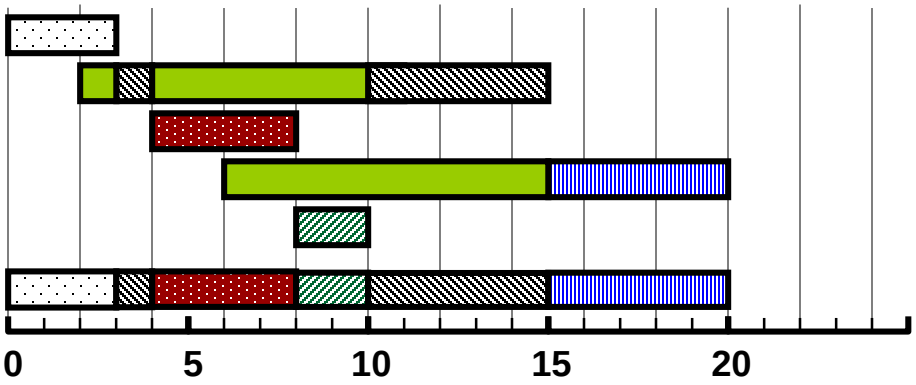
Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3	0	
2	2	6	7	
3	4	4	0	
4	6	5	9	
5	8	2	0	



51

Shortest Remaining Time

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3	0	3
2	2	6	7	13
3	4	4	0	4
4	6	5	9	14
5	8	2	0	2



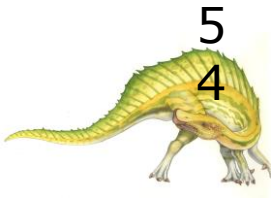
52

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
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process
  - Suppose P4 has a low priority of 4. After waiting for a certain period (e.g., 10 ms), its priority might increase to 3, and after another 10 ms, it might increase to 2, making it eligible for CPU allocation.

Priority Scheduling

- Processor Manager priority assignment methods
  - Memory requirements
    - Jobs requiring large amounts of memory
    - Allocated lower priorities (vice versa)
  - Number and type of peripheral devices
    - Jobs requiring many peripheral devices
    - Allocated lower priorities (vice versa)
- Processor Manager priority assignment methods (continued)
  - Total CPU time
    - Jobs having a long CPU cycle
    - Given lower priorities (vice versa)
  - Amount of time already spent in the system (aging)
    - Total time elapsed since job accepted for processing
    - Increase priority if job in system unusually long time



Memory Requirements and Priority Assignment

Job	Memory Required	Burst Time	Arrival Time	Priority
Job1	50 MB	10 ms	0 ms	High
Job2	300 MB	15 ms	1 ms	Medium
Job3	1000 MB	20 ms	2 ms	Low

Peripheral Devices and Priority Assignment

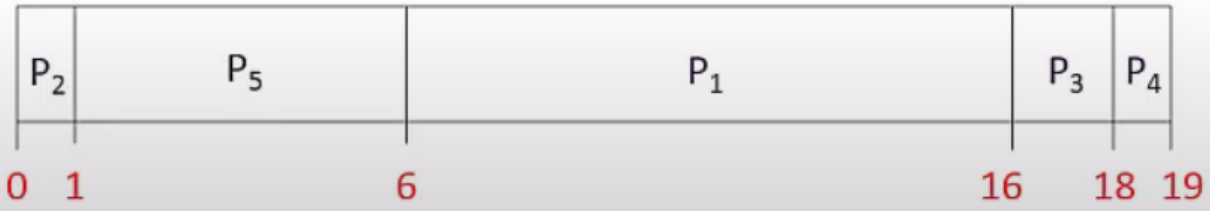
Job	Peripheral Devices Required	Burst Time	Arrival Time	Priority
Job1	Printer	5 ms	0 ms	High
Job2	Printer, Disk Drive, I/O Port	10 ms	1 ms	Medium
Job3	Printer, Disk Drive, Scanner	15 ms	2 ms	Low

Combined Memory and Peripheral Devices Priority Assignment

Priority scheduling example

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

assuming processes all arrived at time 0, *Gantt Chart* for the schedule is :



- average waiting time : (6 + 0 + 16 + 18 + 1)/5 = 8.2
- average turnaround time : (16 + 1 + 18 + 19 + 6)/5 = 12

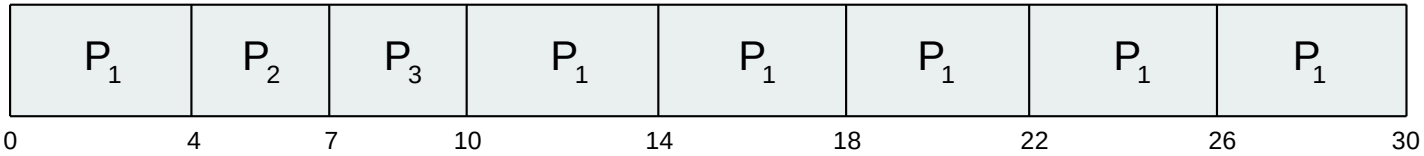
Round Robin (RR)

- 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
  - Information saved in PCB
- Performance
  - *q* large ⇒ FIFO
    - CPU would execute each process fully without switching in most cases. This leads to low context switching overhead, but high waiting times for shorter jobs.
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high
    - q should be large enough to **balance** execution time and context-switching overhead to optimize system performance.

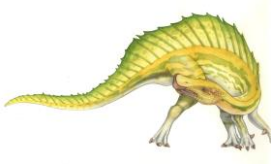
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



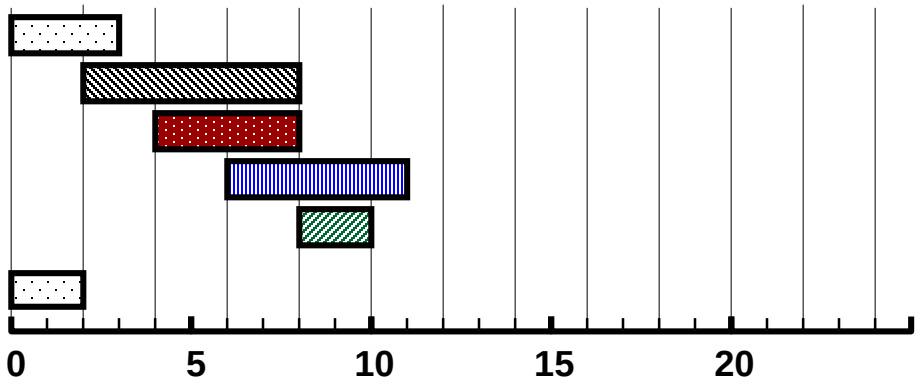
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec



Round-Robin Scheduling Quantum = 1

-

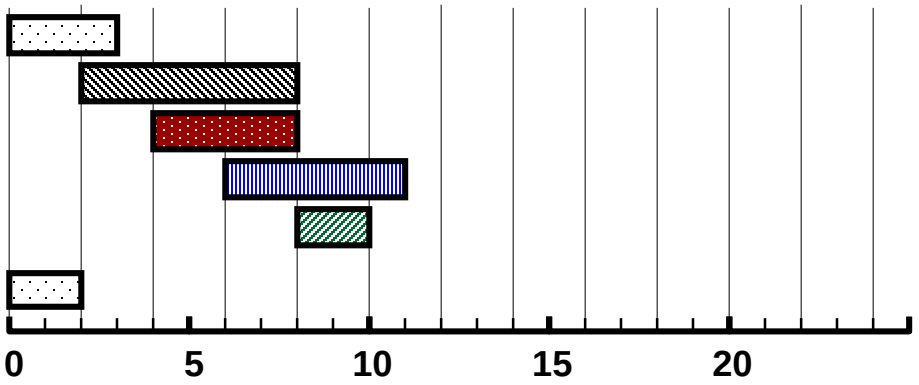
Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

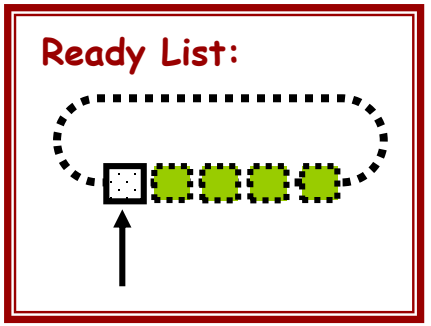


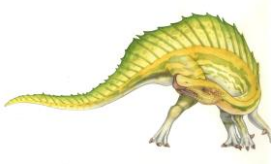
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



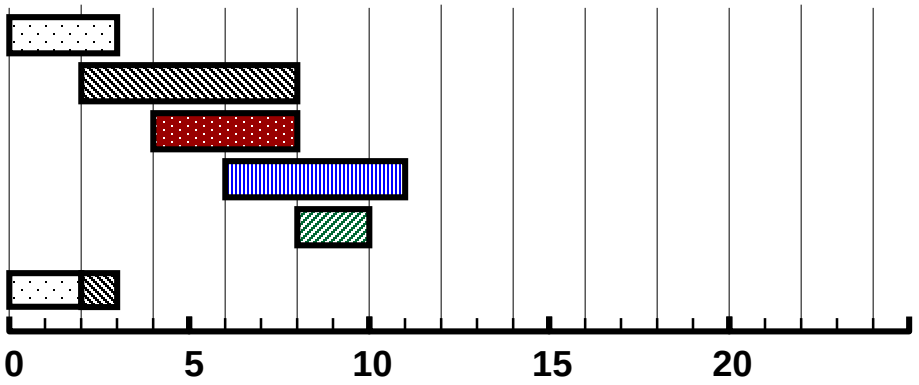


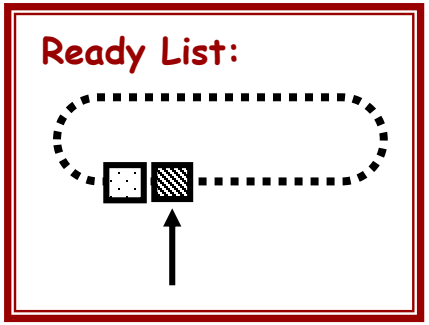


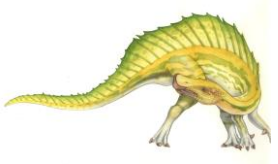
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



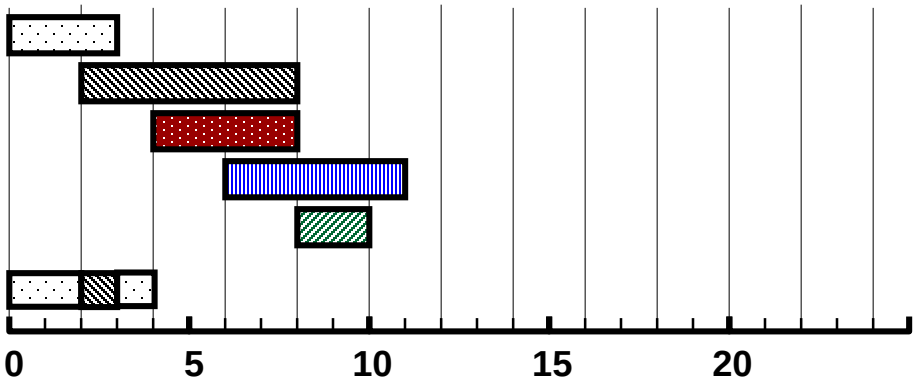


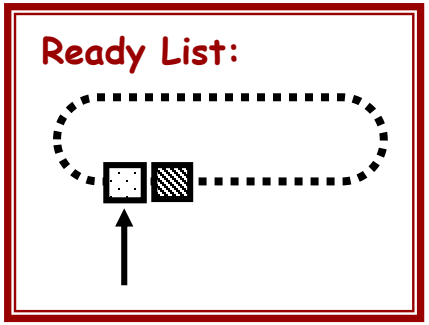


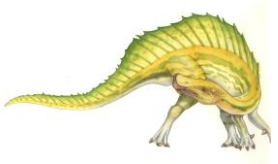
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



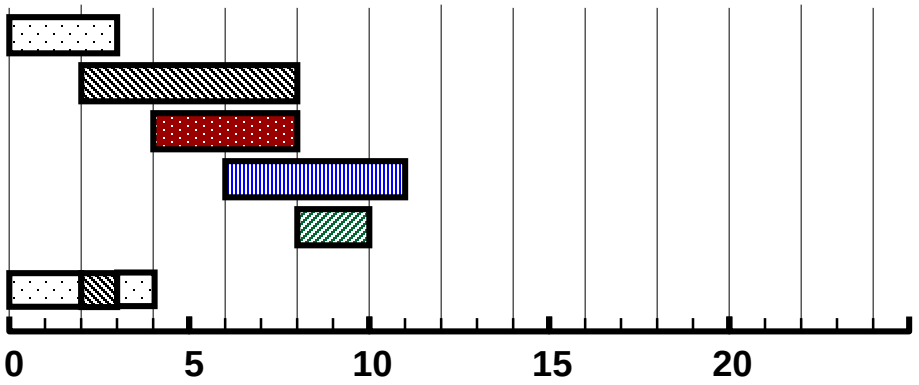


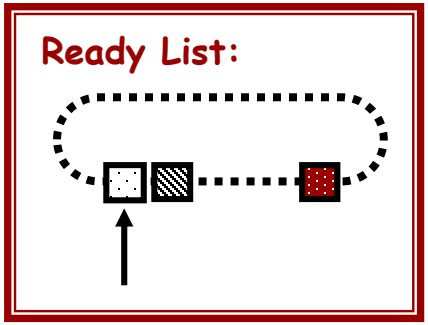


Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



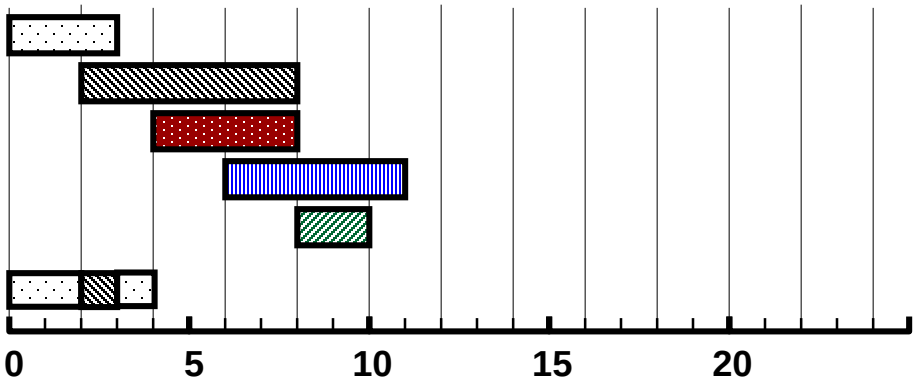


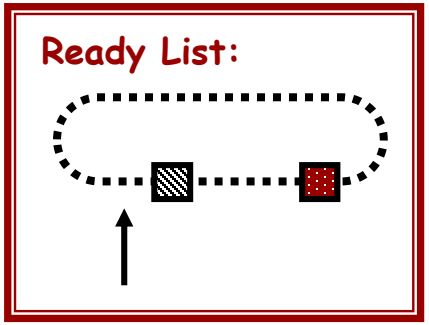
63

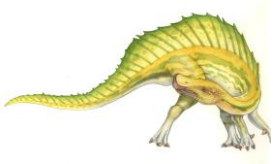
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



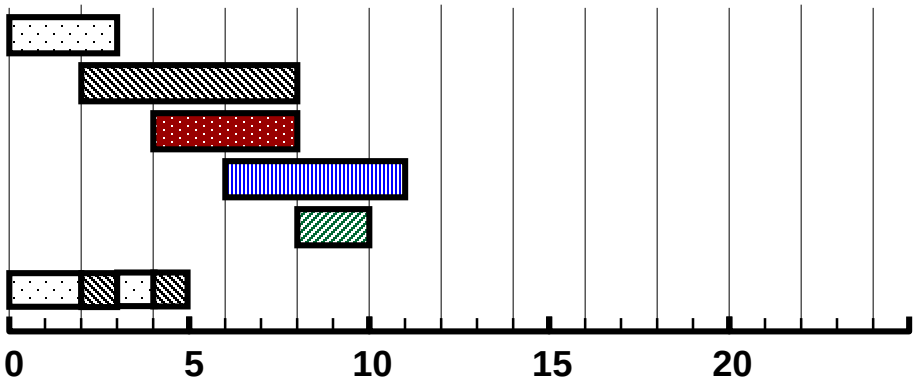


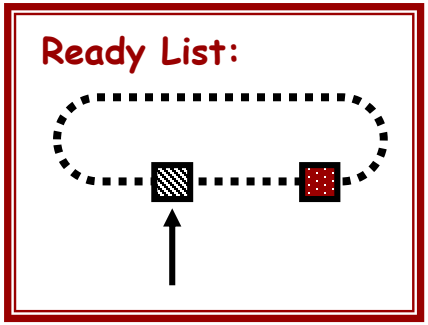


Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

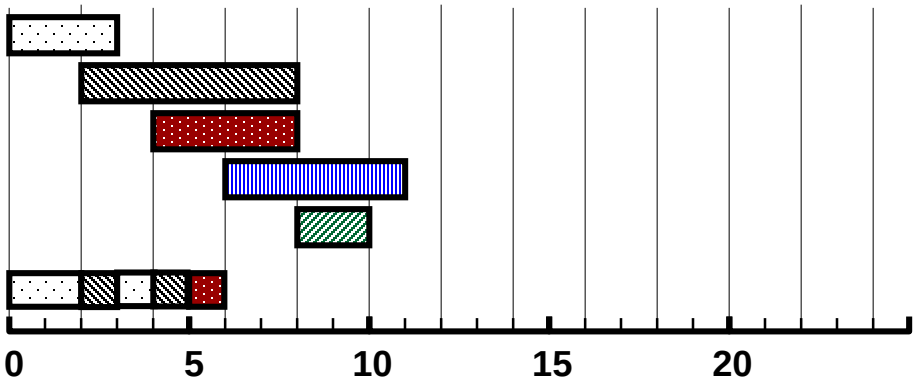


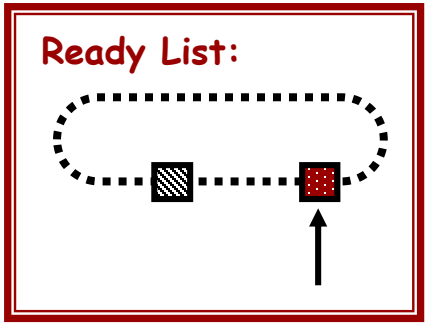


Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



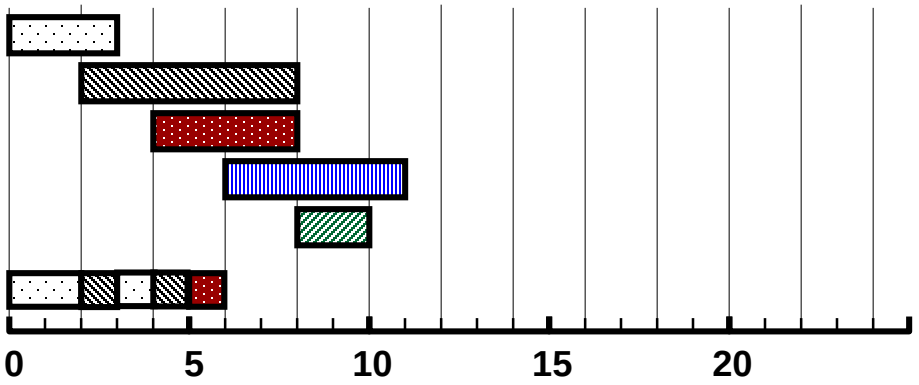


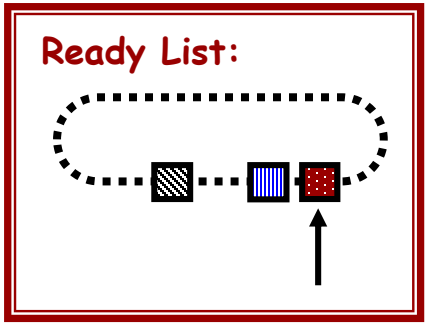
66

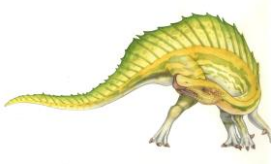
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



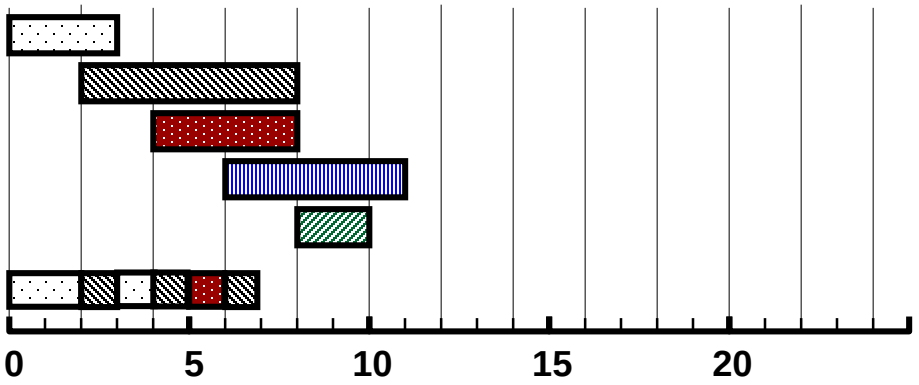


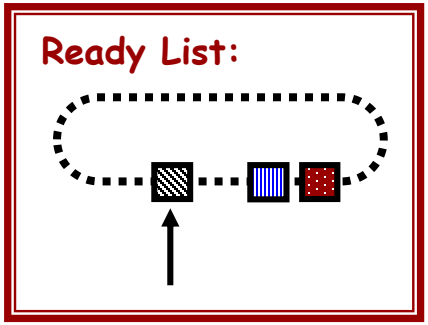


Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

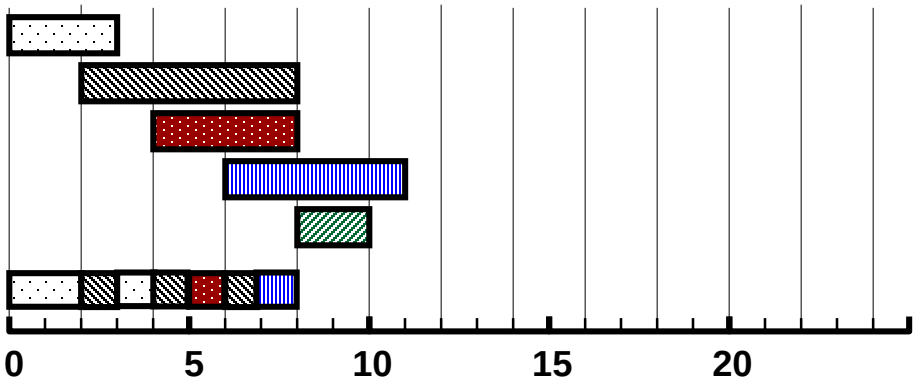


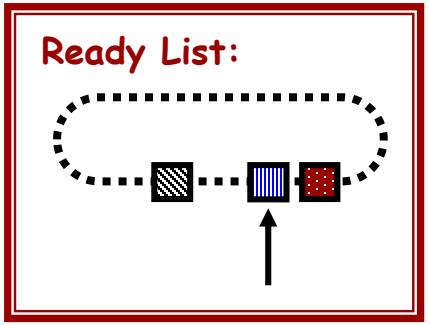


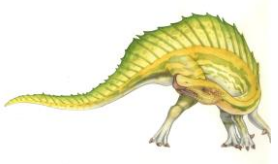
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



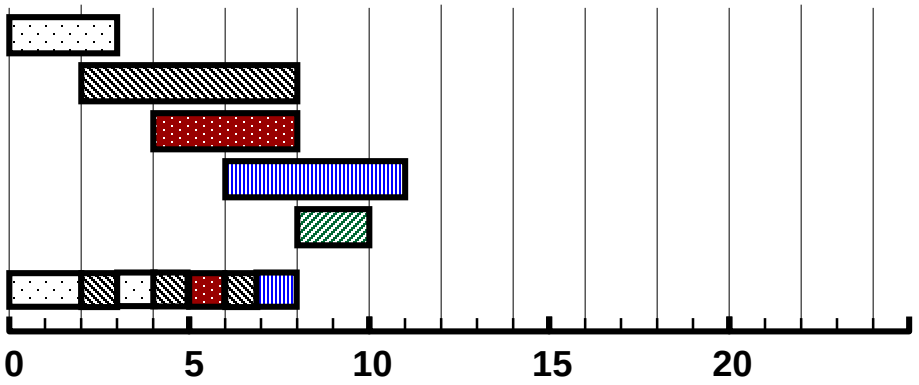


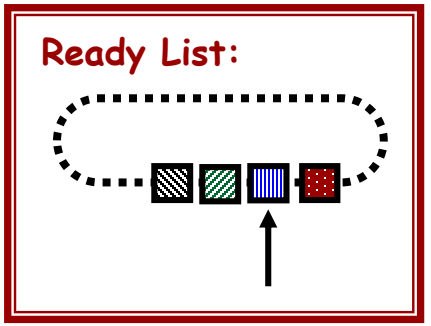


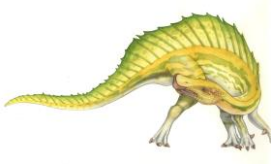
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



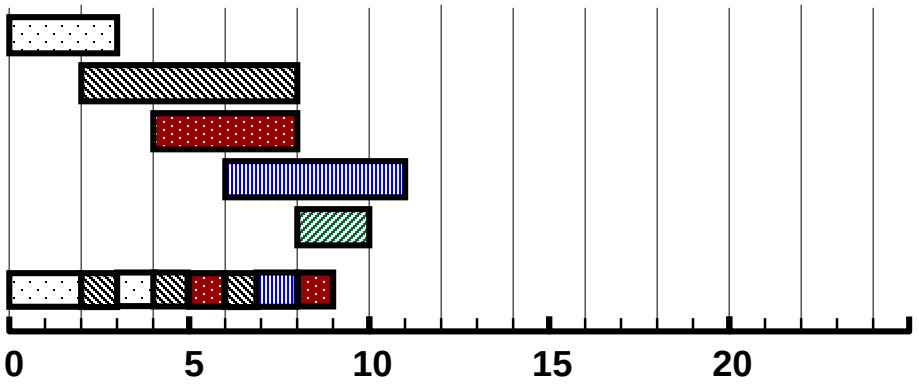


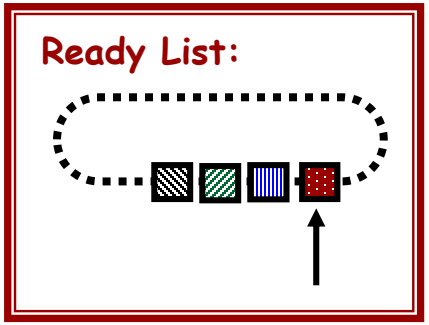


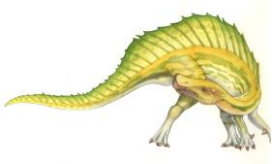
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



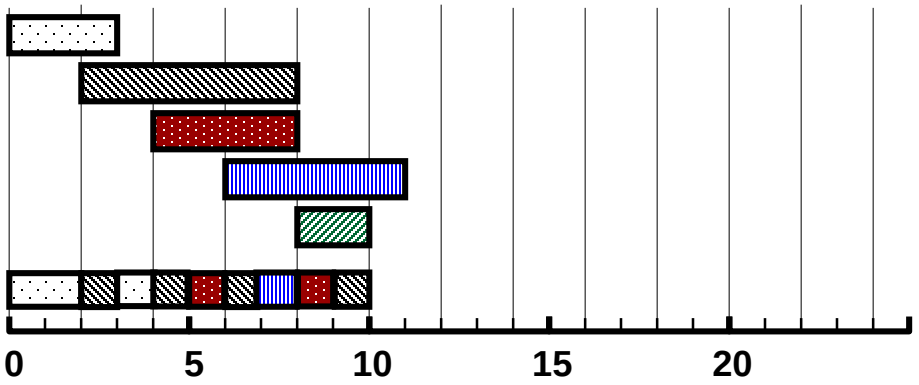


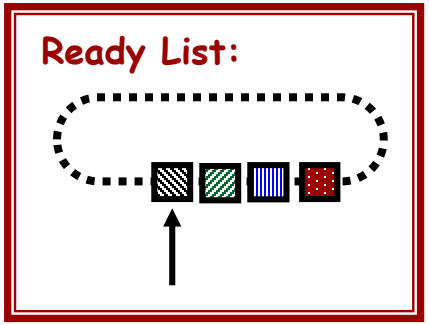


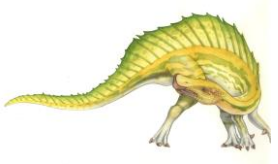
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



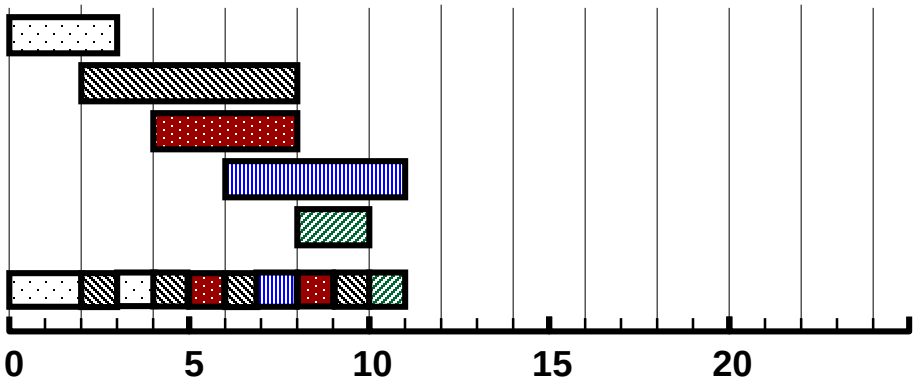


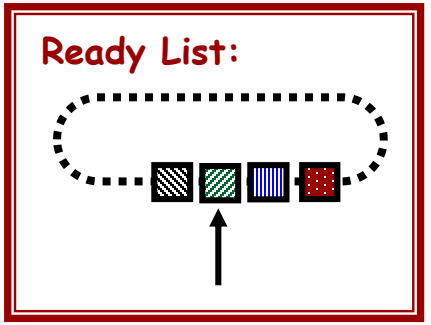


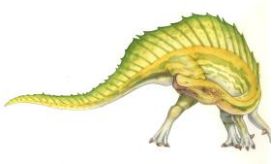
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



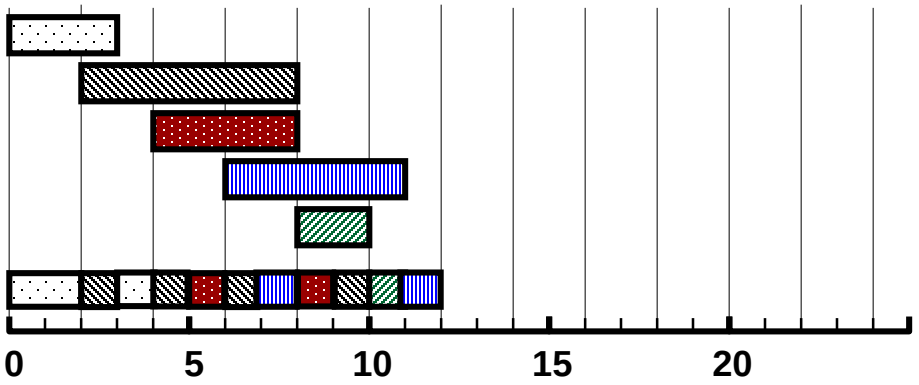


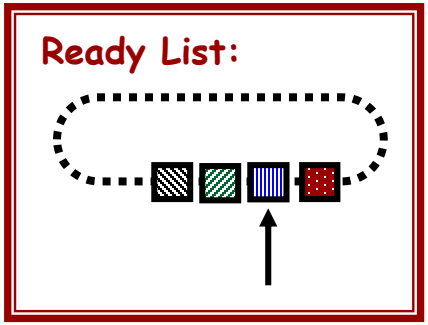


Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

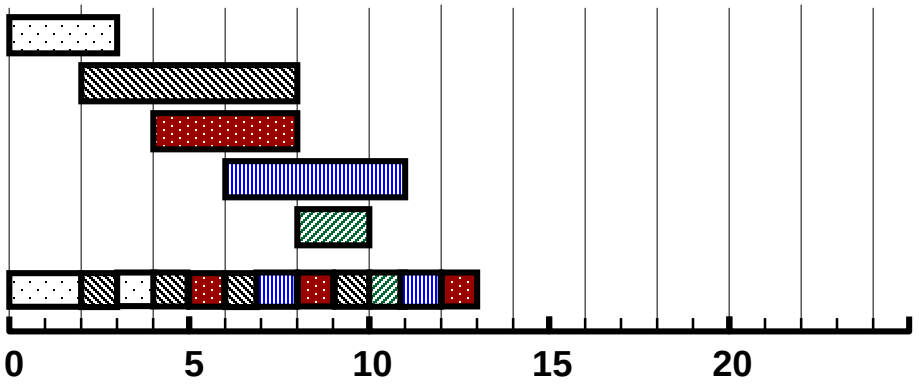


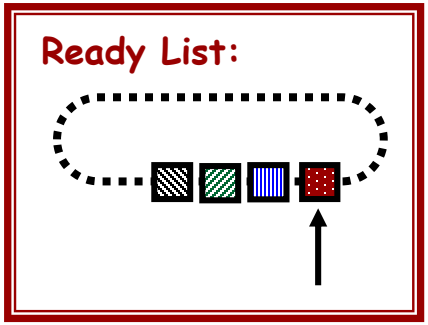


Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



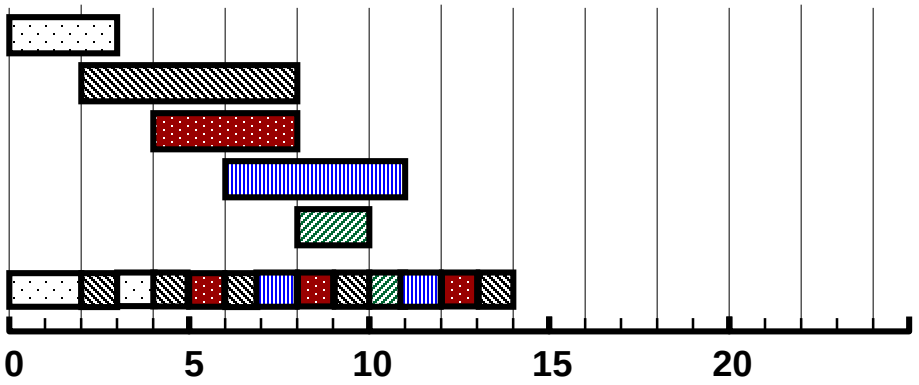


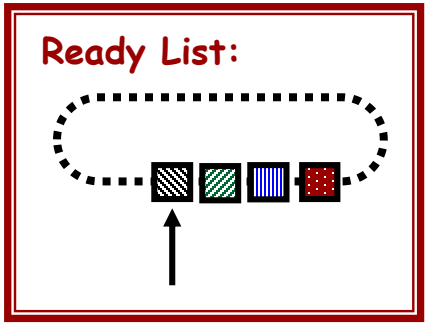
75

Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

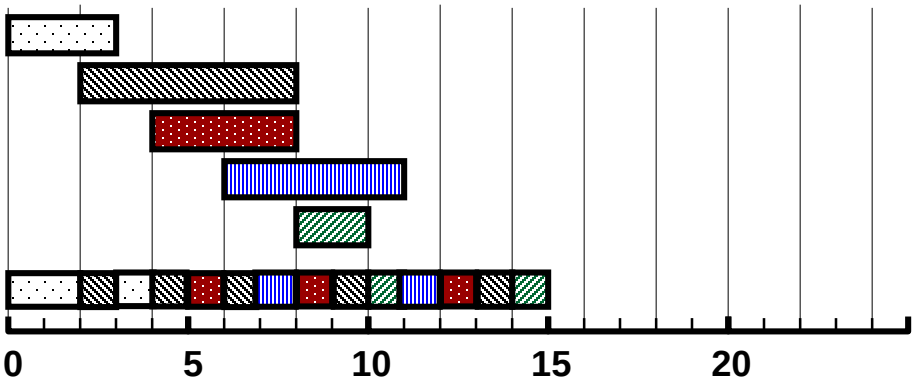


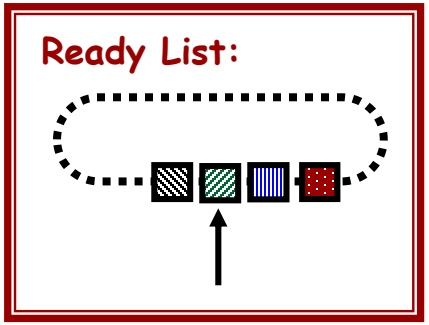


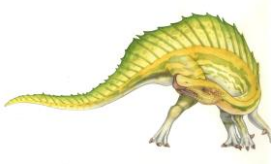
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



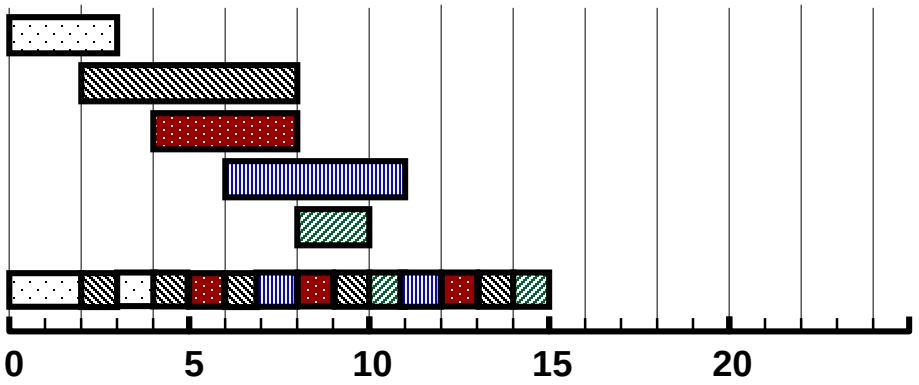


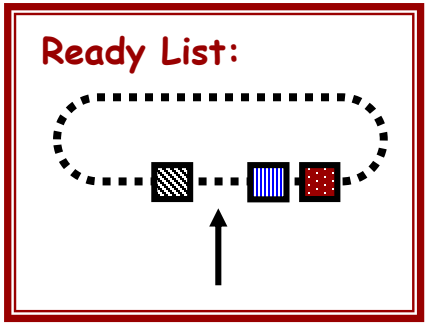


Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

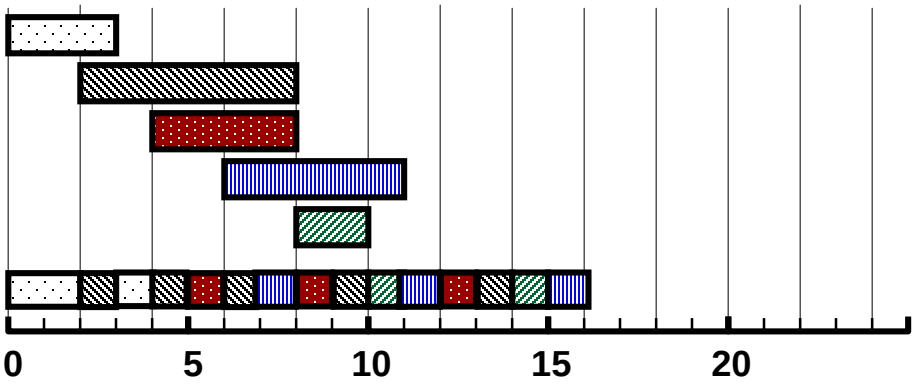


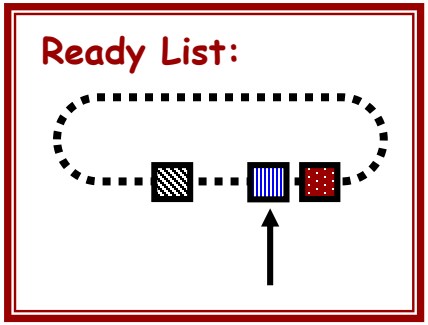


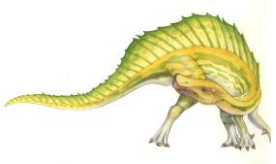
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



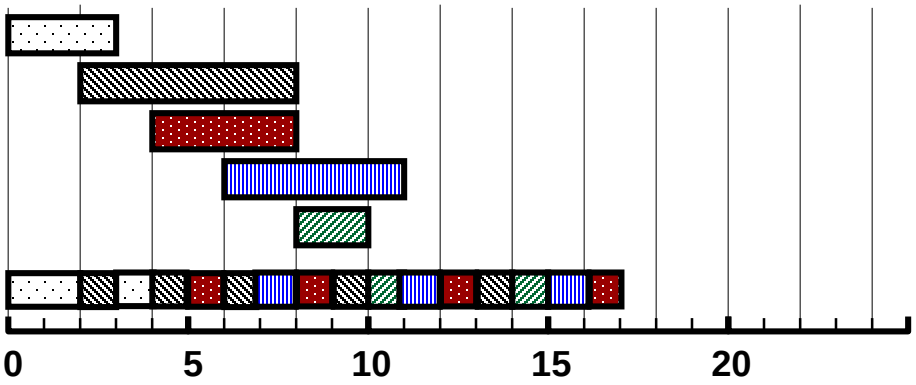


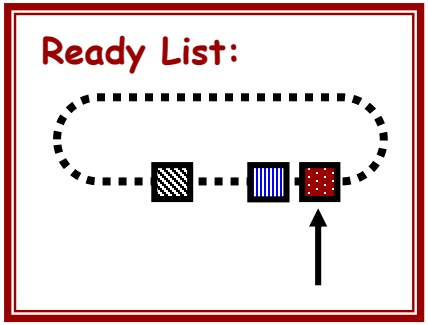


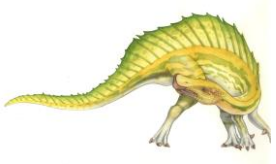
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



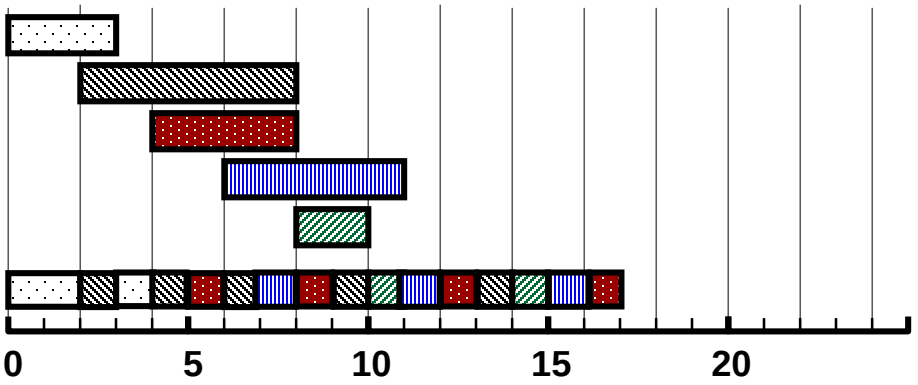


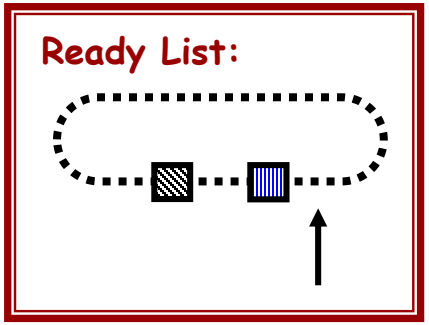


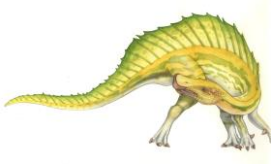
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



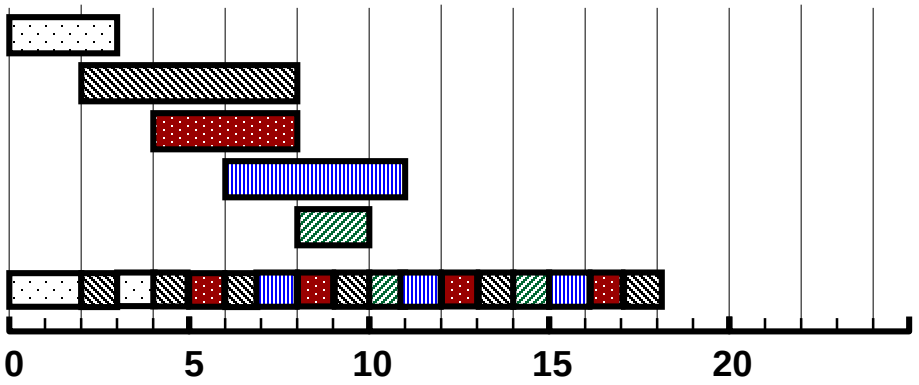


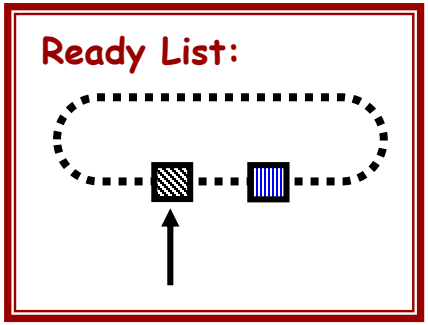


Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

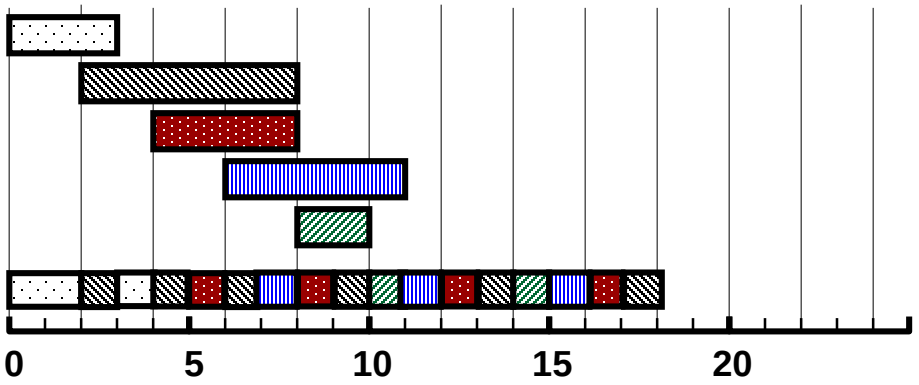


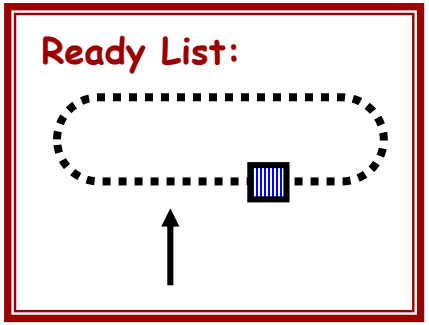


Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

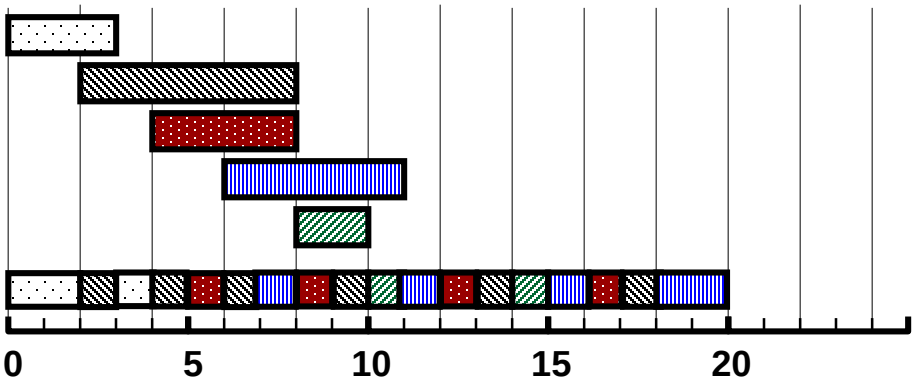


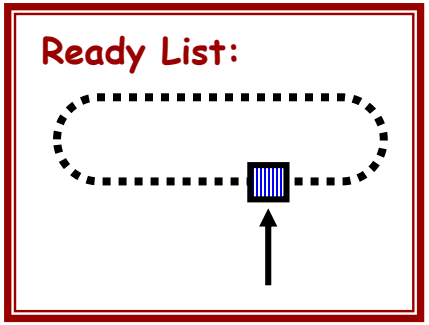


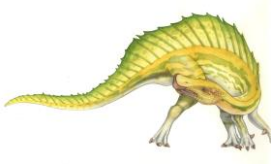
Round-Robin Scheduling

-

Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



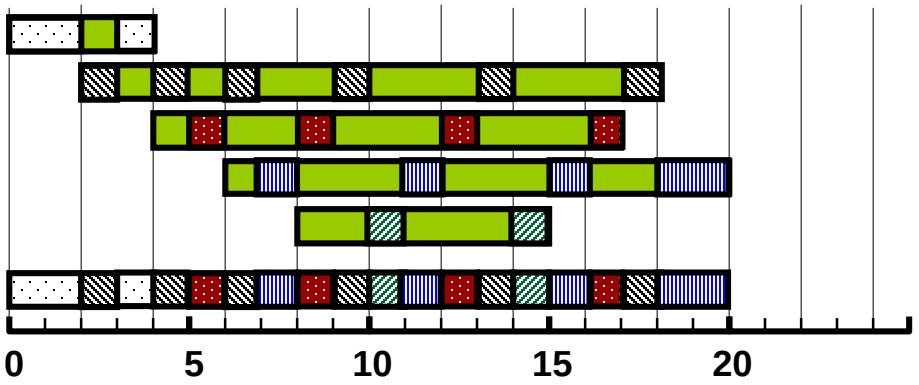




Round-Robin Scheduling

-

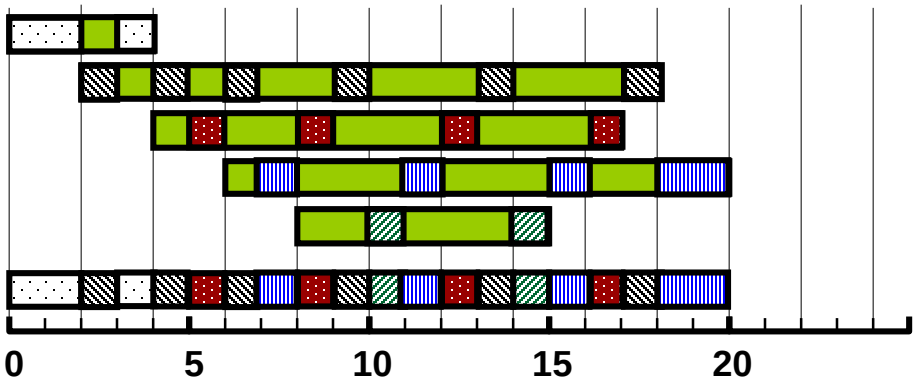
Process	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Round-Robin Scheduling

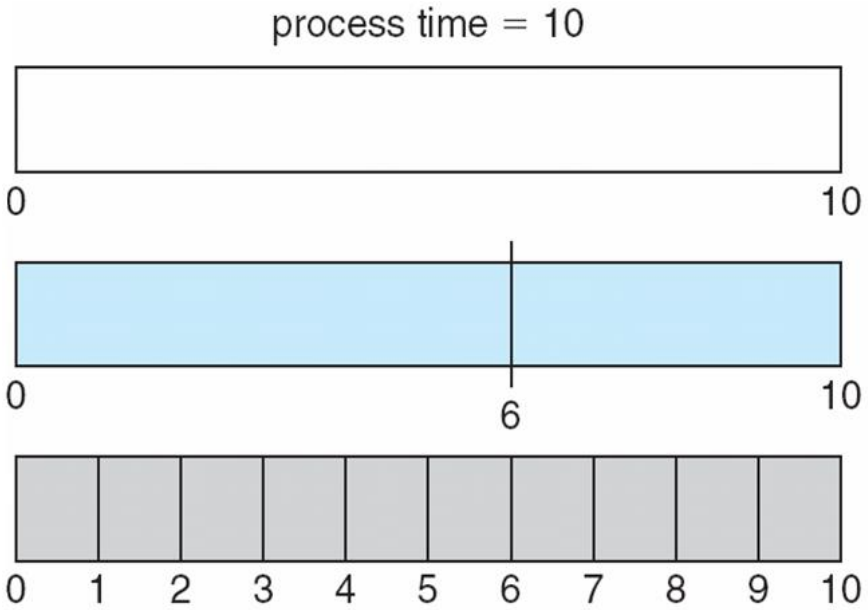
-

Process	Arrival Time	Processing Time	Delay	Turnaround Time
1	0	3	1	4
2	2	6	10	16
3	4	4	9	13
4	6	5	9	14
5	8	2	5	7



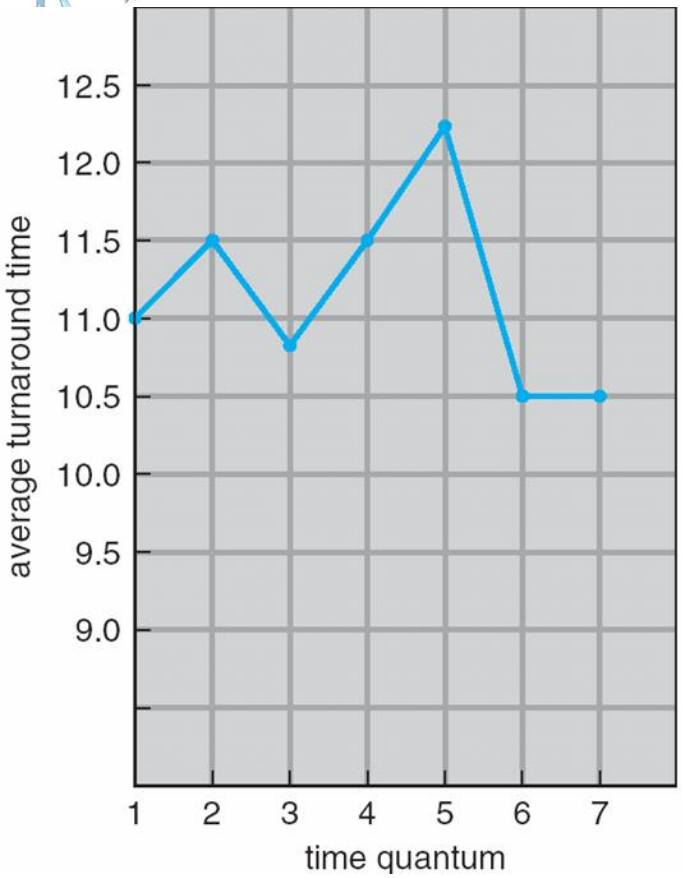
86

Time Quantum and Context Switch Time



quantum	context switches
12	0
6	1
1	9

Turnaround Time Varies With The Time Quantum



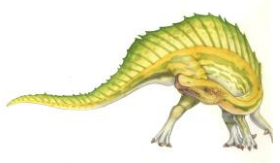
process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

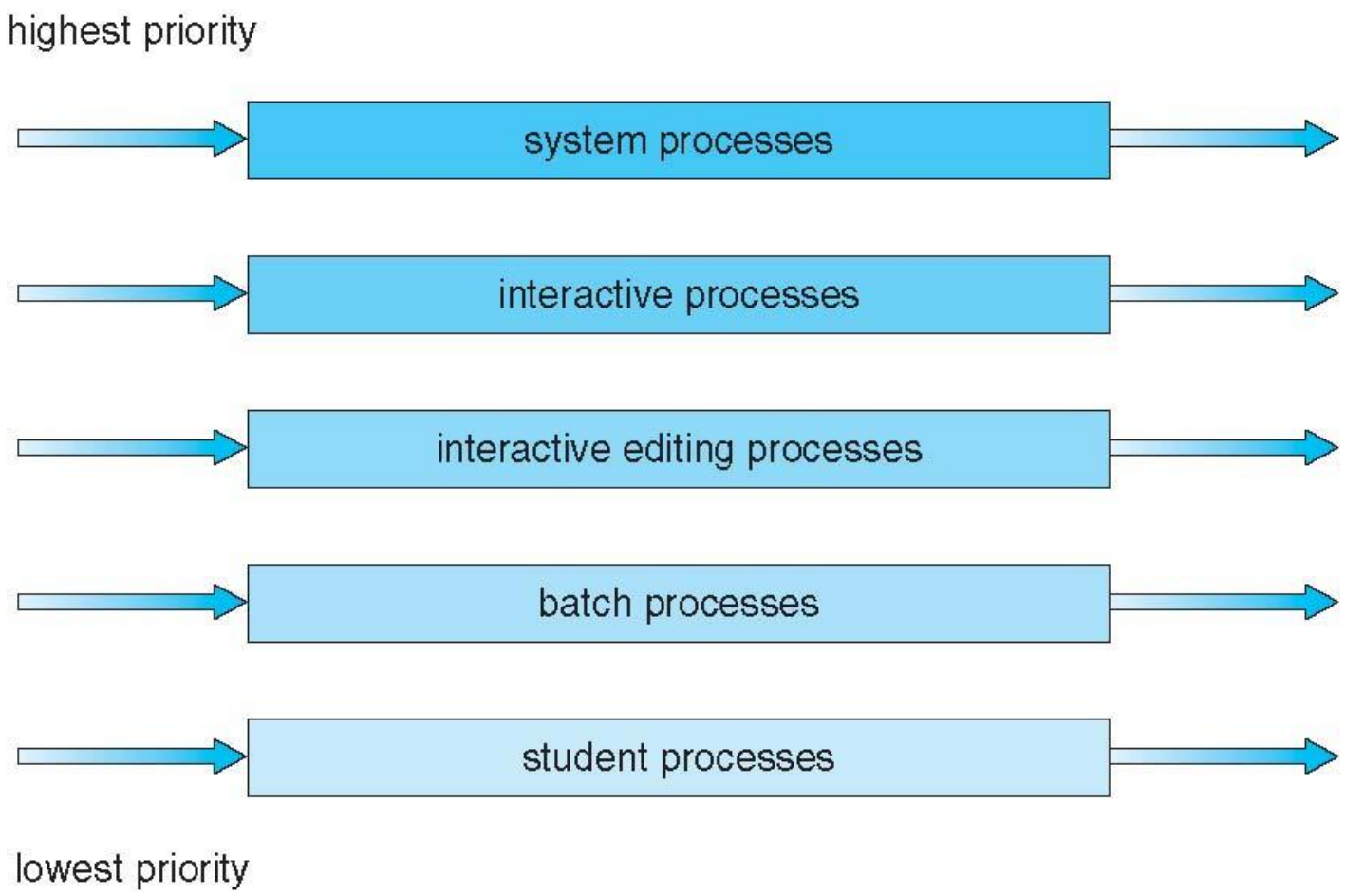
- At very small time quantum values, leading to more context switching overhead.
- When the time quantum becomes large enough, the scheduling behaves more like First-Come, First-Served (FCFS)
- Importance of choosing an appropriate time quantum in Round Robin scheduling to optimize system performance.

Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - **foreground** (interactive)
    - Typically need more immediate attention and quick responses.
  - **background** (batch)
    - Usually involve longer, non-interactive tasks that do not require immediate response.
- Process **permanently** in a given queue: once a process is assigned to a specific queue, it cannot move to another 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

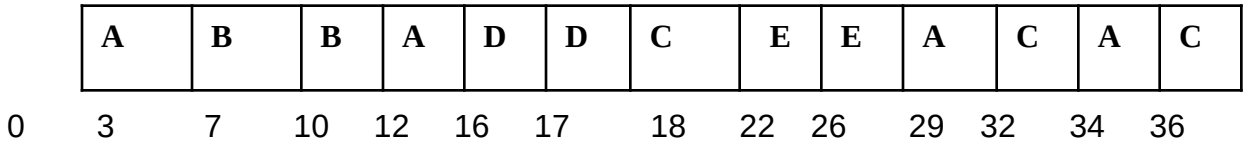


Multilevel Queue Scheduling

Process	Arrival time	CPU burst time (ms)	Priority
A	0	10	2
B	3	7	1
C	4	6	2
D	12	5	1
E	18	8	1

Assume that Quantum=4 ms for Queue 1 and Quantum =3 ms for Queue 2. Assume that Queue 1 has higher priority when compared to Queue 2.

If higher priorty process comes in while a lower priority process is running, lower process is interrupted regardless of q

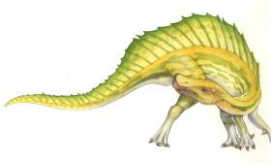


W (A) = 24ms **W (B) = 0ms** **W (C) = 26ms** **W (D) = 0ms**
W (E) = 0ms **W (F) = 10ms**

Multilevel Feedback Queue

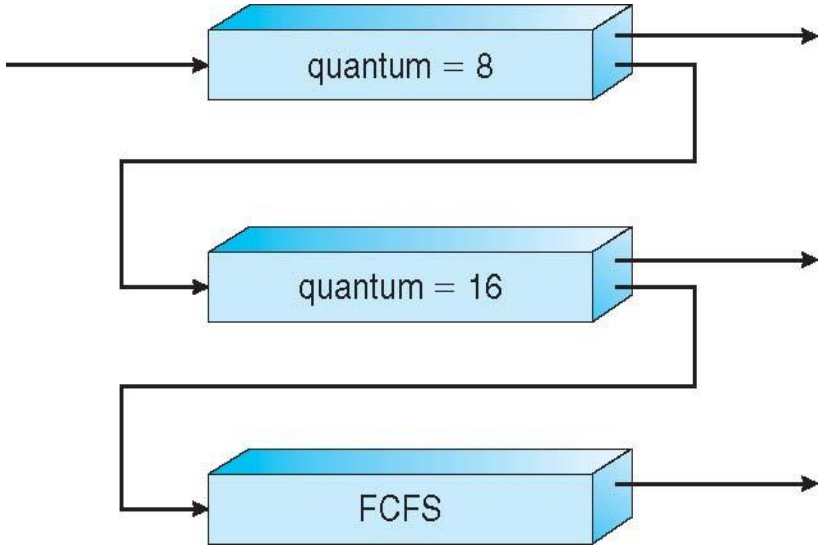
- A process **can move between the various queues**; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

no interrupts

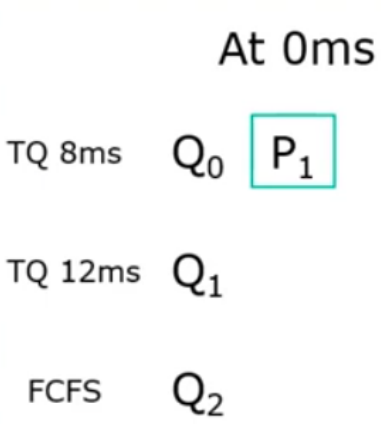


Example of Multilevel Feedback Queue

- 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



Example of Multilevel Feedback Queue



Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15

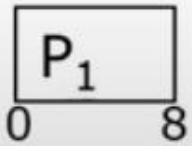
At 8ms

TQ 8ms Q_0 P_2

TQ 12ms Q_1

FCFS Q_2

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

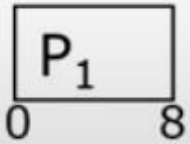
At 8ms

TQ 8ms Q_0 P_2

TQ 12ms Q_1 $P_{1(-4)}$

FCFS Q_2

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

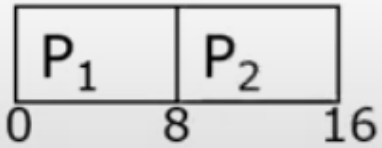
At 16ms

TQ 8ms Q_0

TQ 12ms Q_1 $P_{1(-4)}$

FCFS Q_2

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

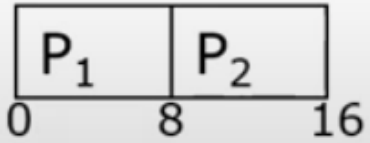
At 16ms

TQ 8ms Q_0

TQ 12ms Q_1 $P_{2(-37)}$ $P_{1(-4)}$

FCFS Q_2

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

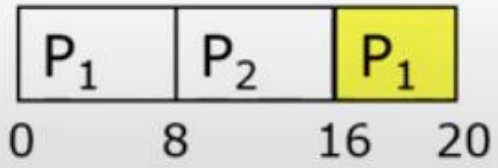
At 20ms

TQ 8ms Q_0

TQ 12ms Q_1 $P_{2(-37)}$

FCFS Q_2

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

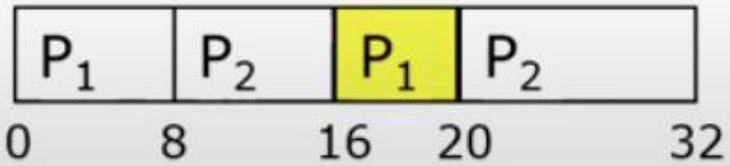
At 32ms

TQ 8ms Q_0 P_4 P_3

TQ 12ms Q_1

FCFS Q_2

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

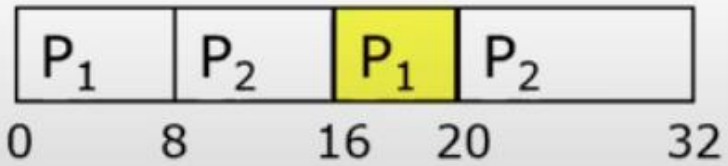
At 32ms

TQ 8ms Q_0 P_4 P_3

TQ 12ms Q_1

FCFS Q_2 $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

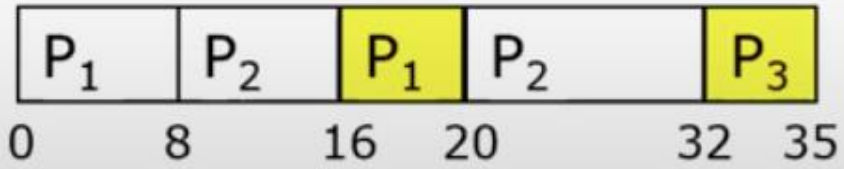
At 35ms

TQ 8ms Q_0 P_5 P_4

TQ 12ms Q_1

FCFS Q_2 $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

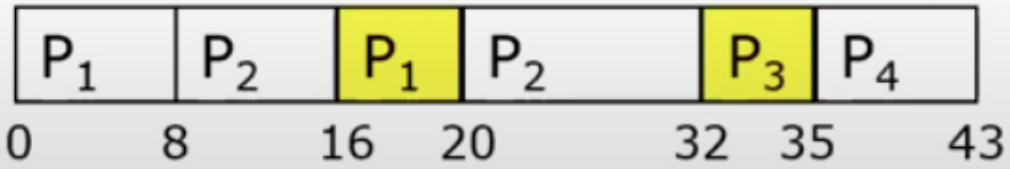
At 43ms

TQ 8ms Q_0 P_6 P_5

TQ 12ms Q_1

FCFS Q_2 $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

At 51ms

TQ 8ms Q_0 P_6

TQ 12ms Q_1 $P_{4\ (-14)}$

FCFS Q_2 $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15

Gantt Chart

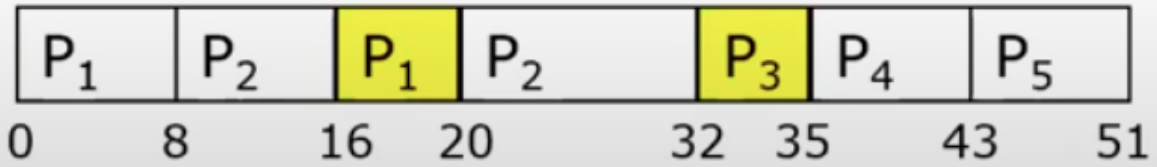
At 51ms

TQ 8ms Q_0 P_6

TQ 12ms Q_1 $P_{5\ (-24)}$ $P_{4\ (-14)}$

FCFS Q_2 $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

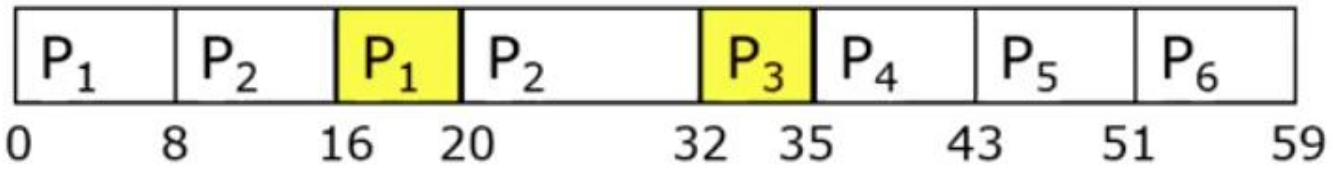
At 59ms

TQ 8ms Q_0

TQ 12ms Q_1 $P_{5\ (-24)}$ $P_{4\ (-14)}$

FCFS Q_2 $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

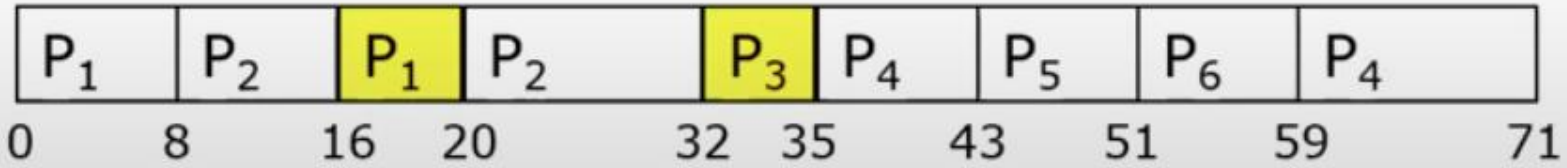
At 71ms

TQ 8ms Q_0

TQ 12ms Q_1 $P_{6\ (-7)}$ $P_{5\ (-24)}$

FCFS Q_2 $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

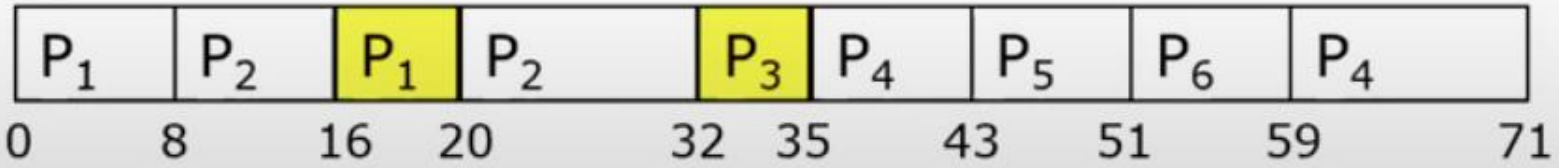
At 71ms

TQ 8ms Q_0

TQ 12ms Q_1 $P_{6\ (-7)}$ $P_{5\ (-24)}$

FCFS Q_2 $P_{4\ (-2)}$ $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

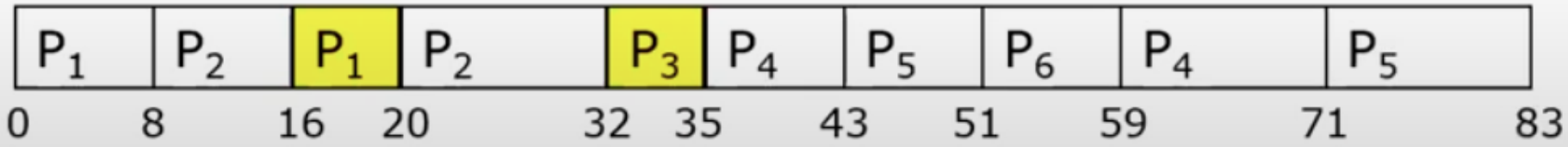
At 83ms

TQ 8ms Q_0

TQ 12ms Q_1 $P_{6\ (-7)}$

FCFS Q_2 $P_{4\ (-2)}$ $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15



Gantt Chart

At 90ms

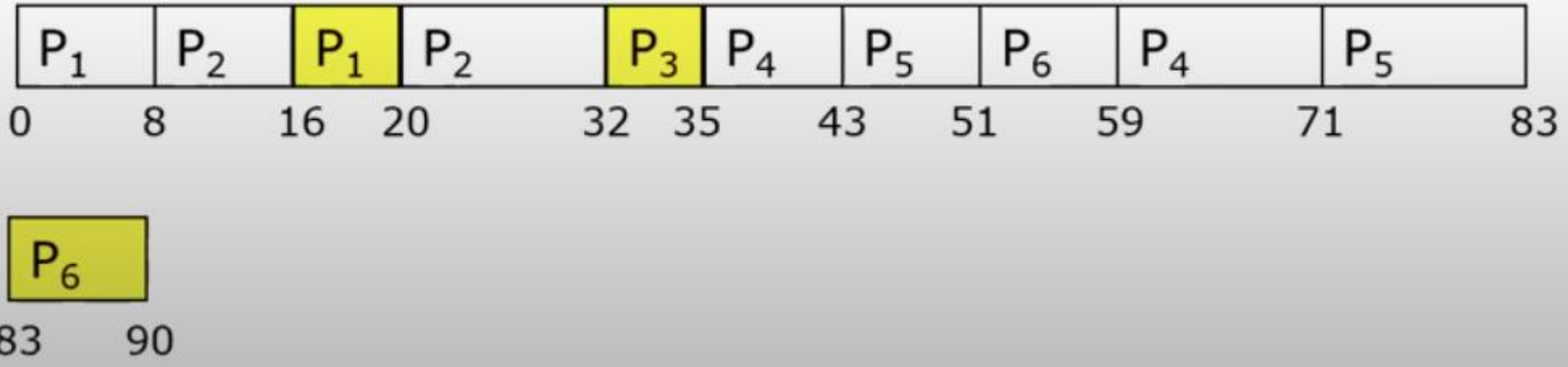
TQ 8ms Q_0

TQ 12ms Q_1

FCFS Q_2 $P_{5\ (-12)}$ $P_{4\ (-2)}$ $P_{2(-25)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15

Gantt Chart



At 117ms

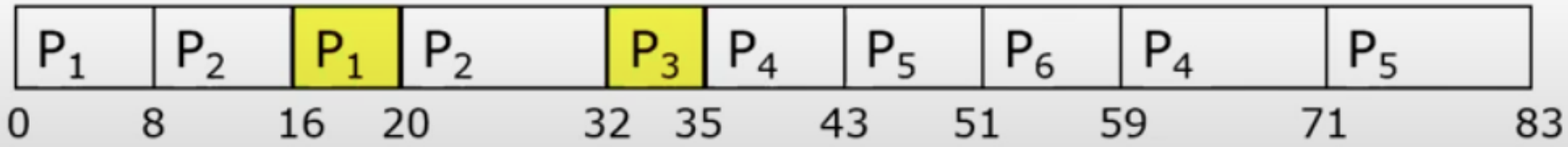
TQ 8ms Q_0

TQ 12ms Q_1

FCFS Q_2 $P_{5\ (-12)}$

Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15

Gantt Chart



At 129ms

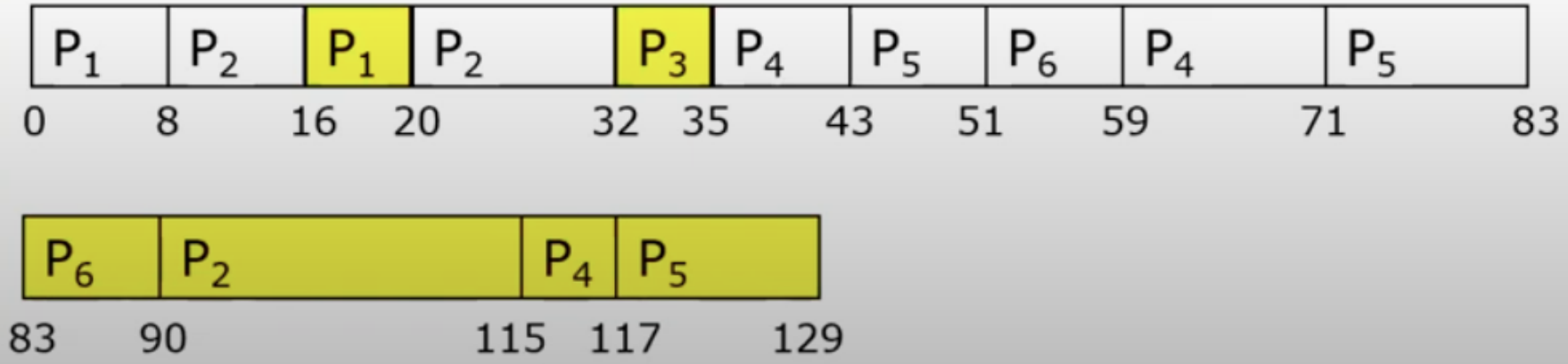
TQ 8ms Q_0

TQ 12ms Q_1

FCFS Q_2

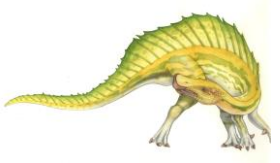
Process	Arrival Time	Burst Time
P1	0	12
P2	5	45
P3	24	3
P4	30	22
P5	33	32
P6	40	15

Gantt Chart



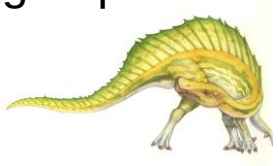
Thread Scheduling

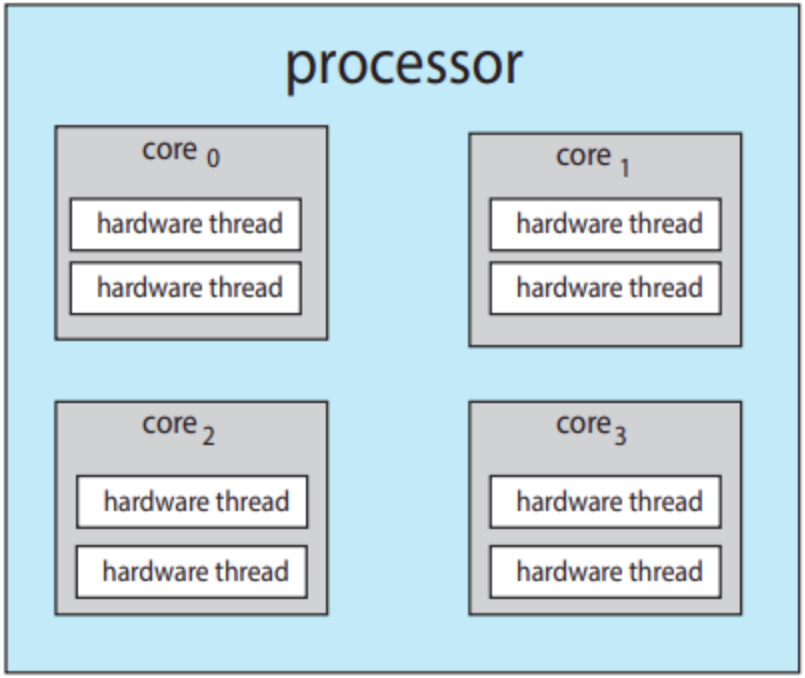
- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Known as **process-contention scope (PCS)** since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope (SCS)** – competition among all threads in system

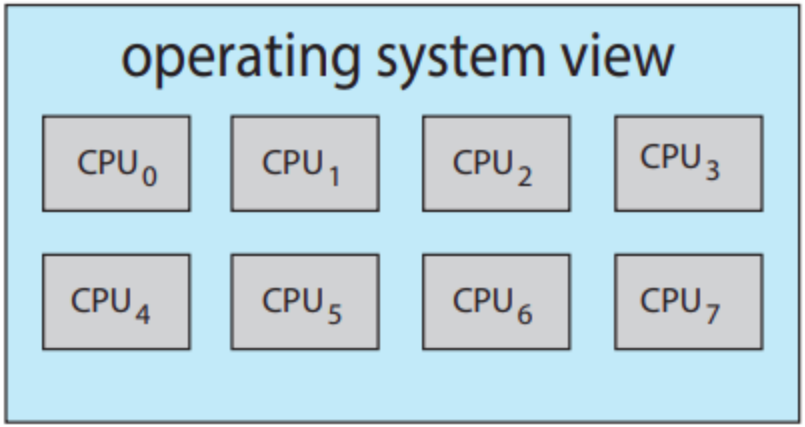


Multiple-Processor Scheduling

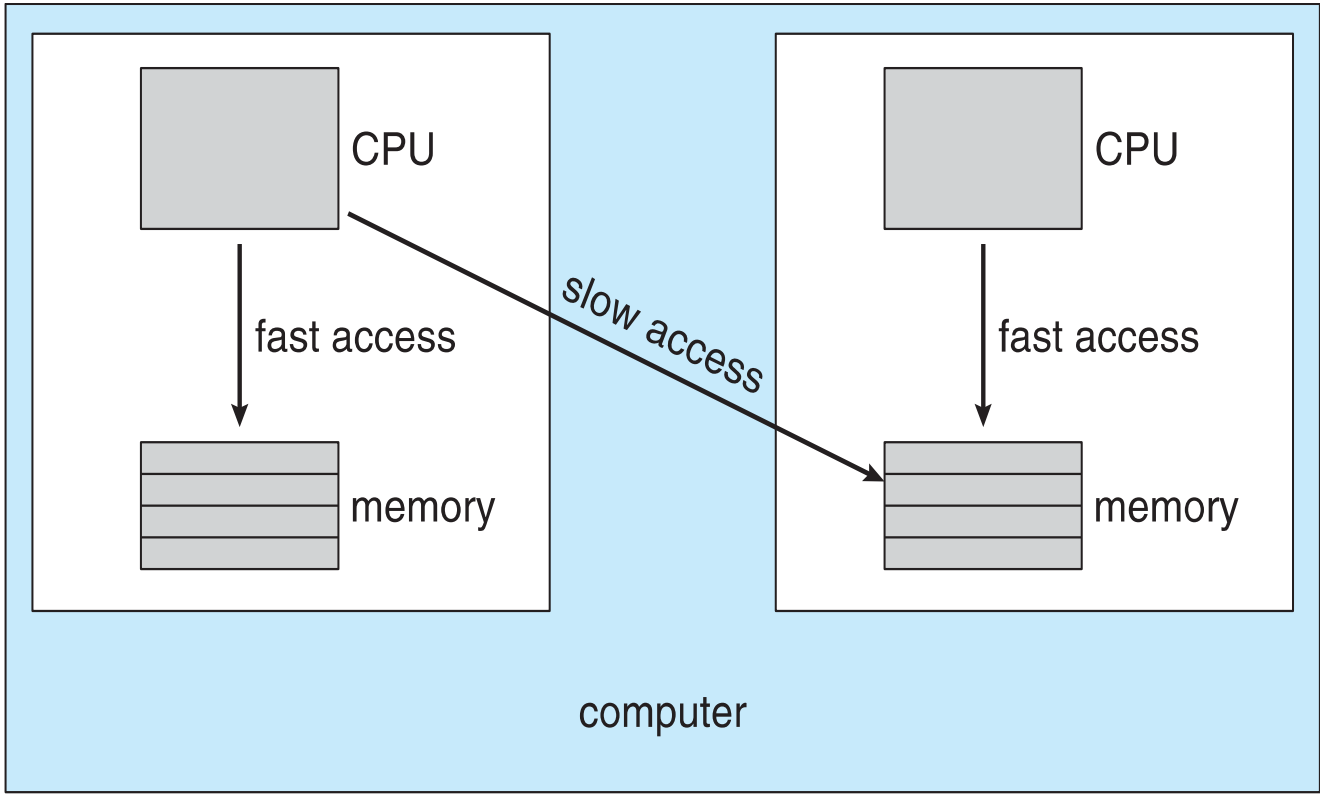
- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous processors** within a multiprocessor
- **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
  - Currently, most common
- **Processor affinity** – process has affinity for processor on which it is currently running
  - **soft affinity: OS** tries to keep a process running on the same processor, but it **does not enforce** this strictly.
  - **hard affinity**
  - Variations including **processor sets:** load balancing within the specified set of processors, while still restricting the process to a defined group of CPUs




processor
core 0
hardware thread
hardware thread
core 1
hardware thread
hardware thread
core 2
hardware thread
hardware thread
core 3
hardware thread
hardware thread


operating system view
CPU 0
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6
CPU 7

NUMA (Non-Uniform Memory Access) and CPU Scheduling



Note that memory-placement algorithms can also consider affinity

Multiple-Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task from busy processor

CPU	Current Load (Processes)
CPU 0	4 processes
CPU 1	1 process
CPU 2	2 processes
CPU 3	**0 processes**

Push Migration

CPU	Current Load (Processes)
CPU 0	**2 processes**
CPU 1	1 process
CPU 2	2 processes
CPU 3	**2 processes**

CPU	Current Load (Processes)
CPU 0	2 processes
CPU 1	**0 process**
CPU 2	2 processes
CPU 3	2 processes

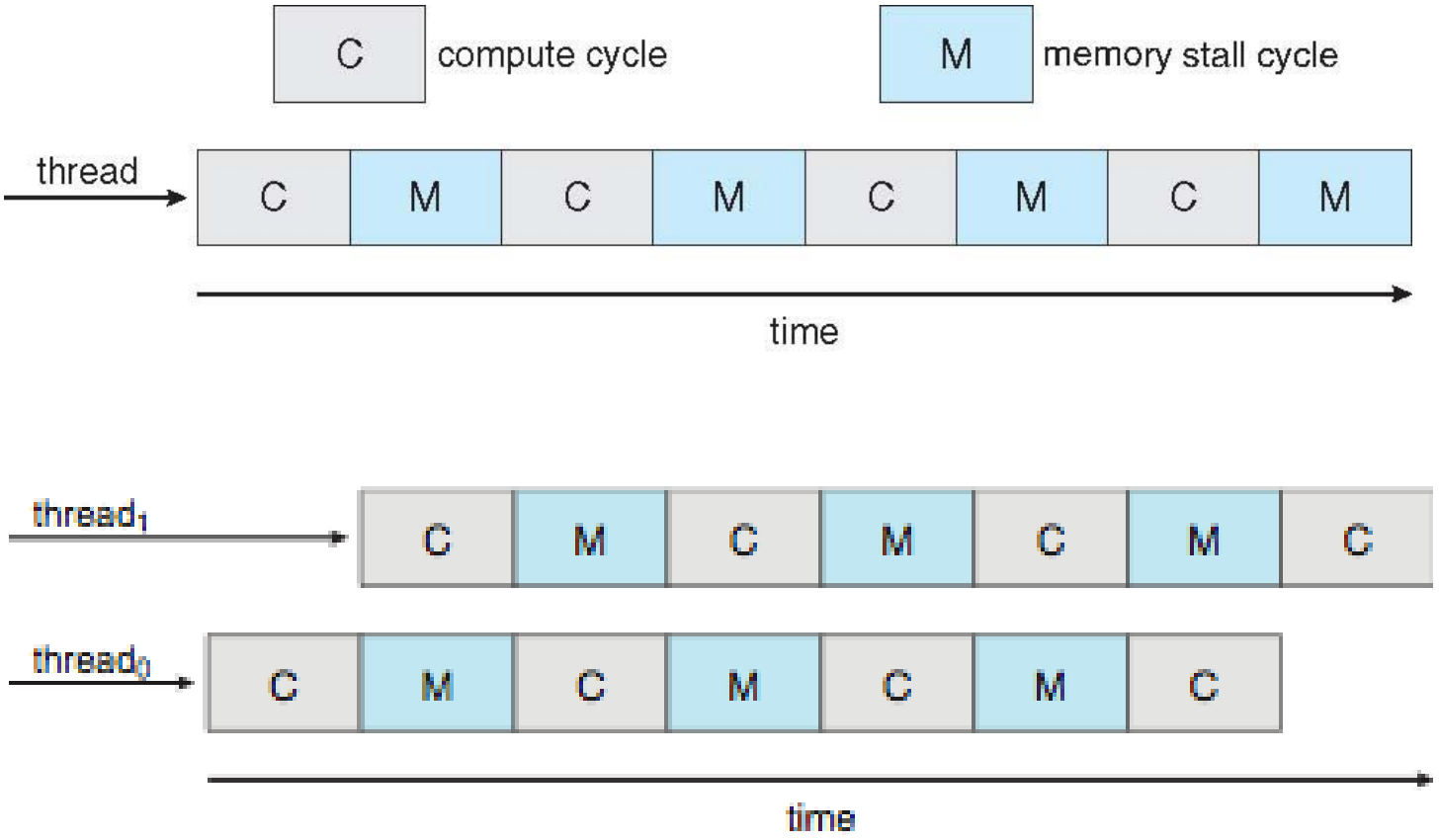
Pull Migration

CPU	Current Load (Processes)
CPU 0	**1 processes**
CPU 1	**1 process**
CPU 2	2 processes
CPU 3	2 processes

Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

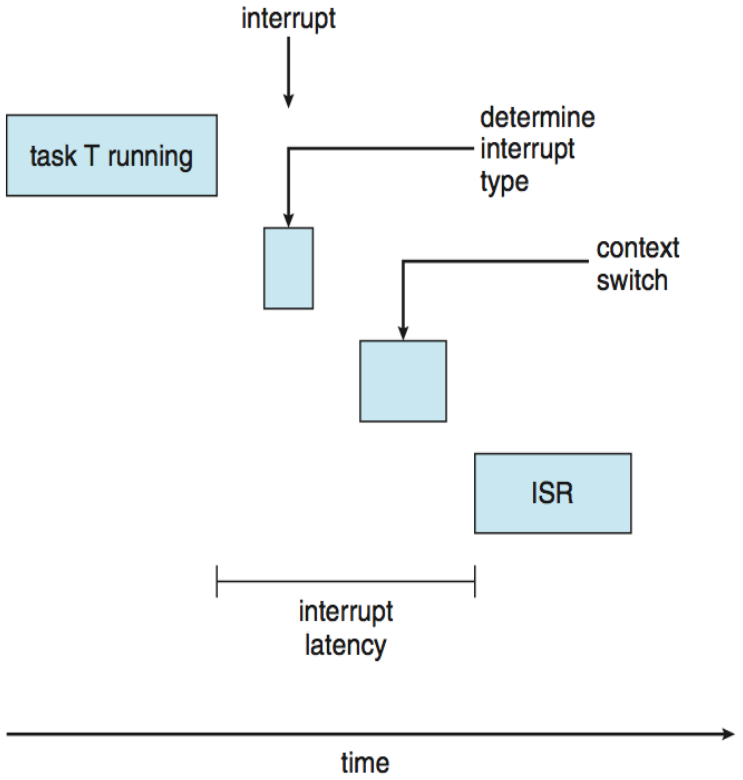
Multithreaded Multicore System



Real-Time CPU Scheduling

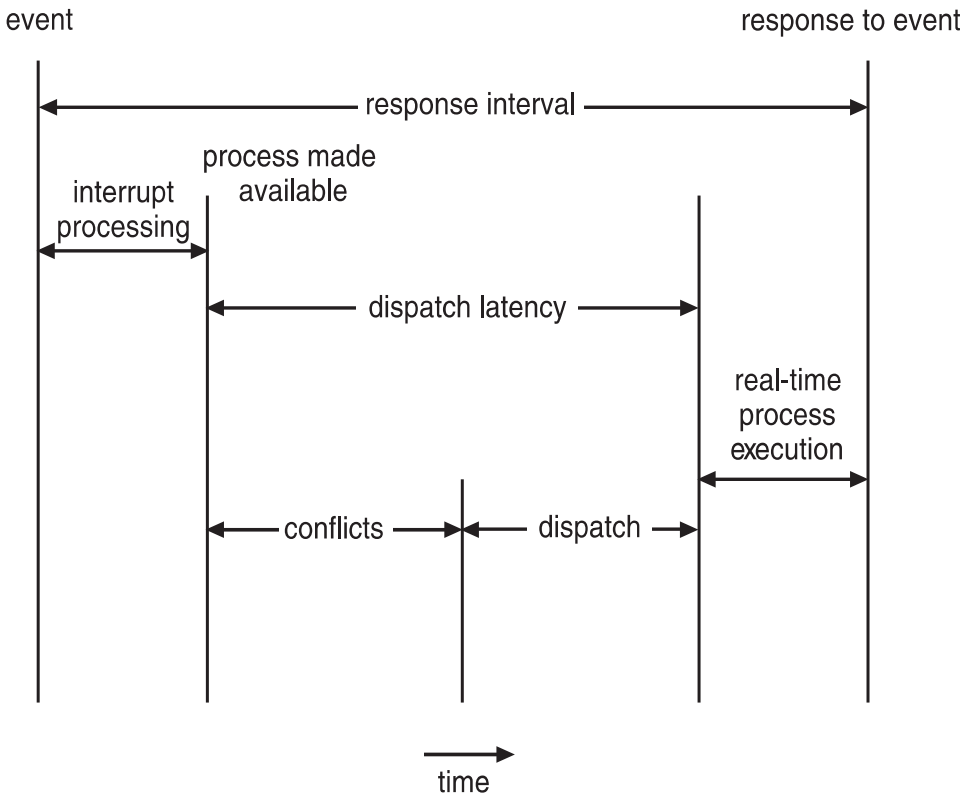
- Can present obvious challenges
- **Soft real-time systems** – no guarantee as to when critical real-time process will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline
- Two types of latencies affect performance
  1. Interrupt latency – time from arrival of interrupt to start of routine that services interrupt
  2. Dispatch latency – time for schedule to take current process off CPU and switch to another

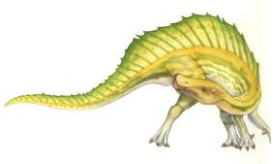
interrupt service routine (ISR)



Real-Time CPU Scheduling (Cont.)

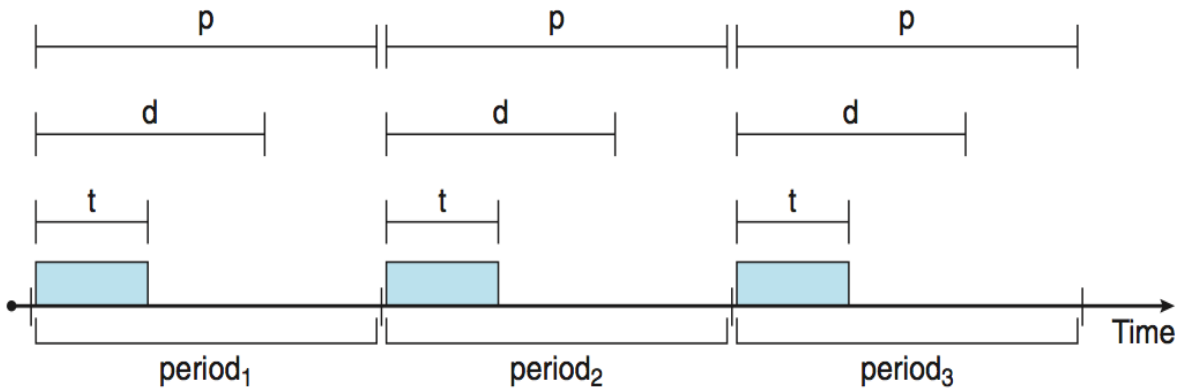
- Conflict phase of dispatch latency:
  1. Preemption of any process running in kernel mode, higher privileges and may be executing critical system functions
  2. Release by low-priority process of resources needed by high-priority processes

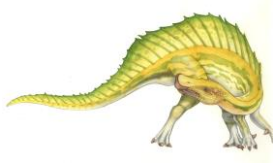




Priority-based Scheduling

- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d,* period *p*
  - $0 \le t \le d \le p$
  - **Rate** of periodic task is $1/p$



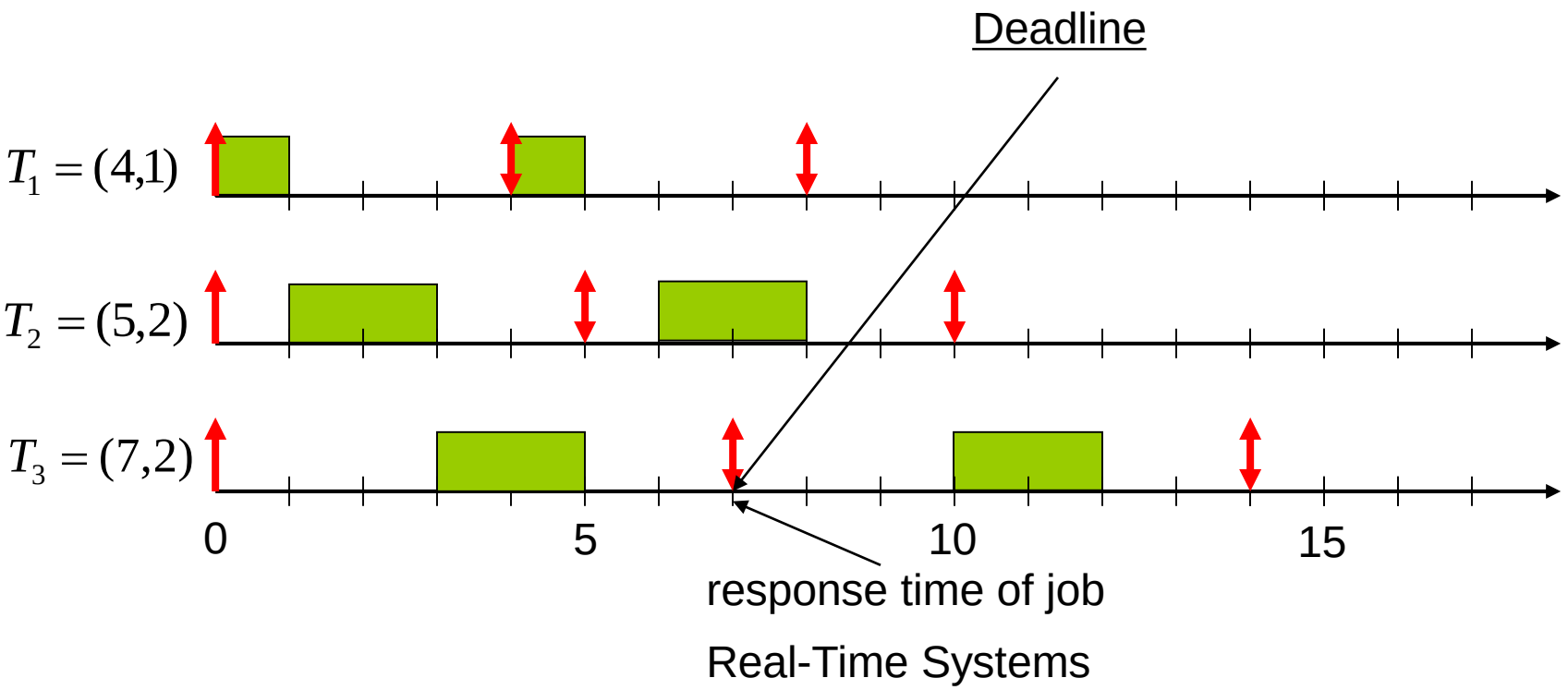


Rate Montonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2.

Example: Rate Monotonic Scheduling

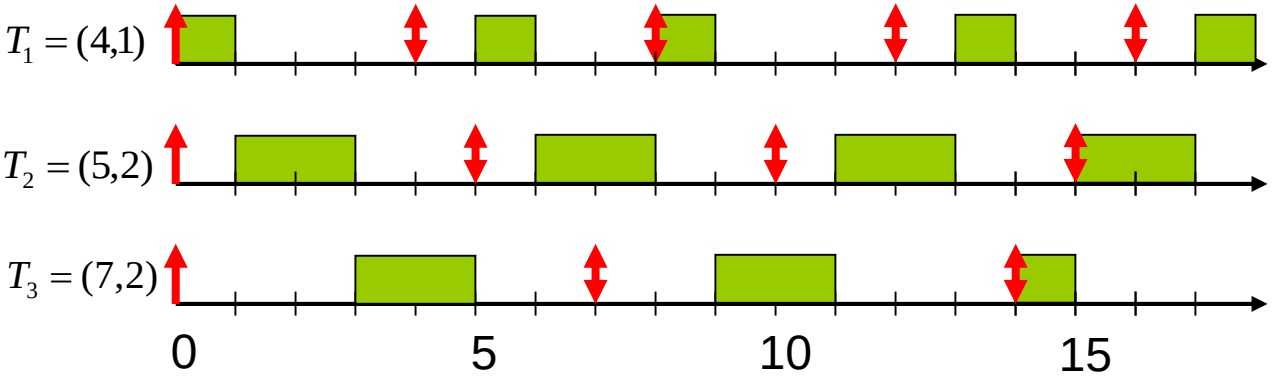
$$T_i = (P_i, C_i) \quad P_i = \text{period} \quad C_i = \text{processing time}$$

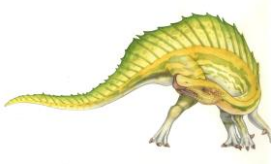


EDF Scheduling: Principle

$$T_i = (P_i, C_i) \quad P_i = \text{period} \quad C_i = \text{processing time}$$

- Preemptive priority-based dynamic scheduling, the earlier the deadline, the higher the priority; the later the deadline, the lower the priority
- Each task is assigned a (current) priority based on how close the absolute deadline is.
- The scheduler always schedules the active task with the closest absolute deadline.
- **All instances of a periodic task have the same relative deadline, which is equal to the period**



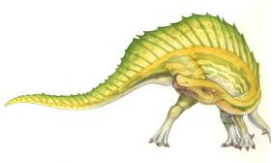


Proportional Share Scheduling

- *T* shares are allocated among all processes in the system
- An application receives *N* shares where *N* < *T*
- This ensures each application will receive ***N*** / *T* of the total processor time
- Provides a **fair** and **proportional** distribution of CPU time based on the relative importance of processes or applications, making it ideal for environments where controlled resource sharing is crucial.
- Example:
  1. **Total Shares (T):**
     - Suppose the system has **100 total shares (T = 100)**. These shares represent the total available CPU time that can be distributed among all processes.
  2. **Process A Receives N Shares:**
     - Suppose **Process A** is allocated **20 shares (N = 20)**.
  3. **CPU Time for Process A:**
     - The share of CPU time **Process A** will receive is proportional to the **N / T** ratio:

$$\text{CPU Time for Process A} = \frac{N}{T} = \frac{20}{100} = 0.2$$

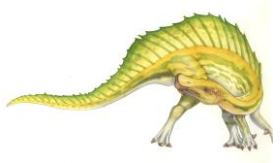
     - **Process A** will receive **20% of the total CPU time.**



Operating System Examples

- Linux scheduling
- Windows scheduling
- Solaris scheduling
- **Linux's Completely Fair Scheduler (CFS)** and **Windows Scheduling**

Feature	Linux - CFS	Windows Scheduling
Scheduling Type	Fair-share based on virtual runtime (vruntime)	Priority-based with 32 levels
Time Slice	Dynamic, based on how much CPU a task has used	Variable, based on task priority
Priority Scheme	No fixed priorities (uses nice values to adjust CPU time)	Fixed 32 priority levels (0-15 real-time, 16-31 variable)
Preemption	Based on vruntime (lower vruntime tasks preempt others)	Preemptive, based on priority



Algorithm Evaluation

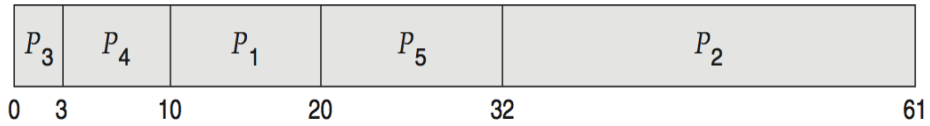
- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

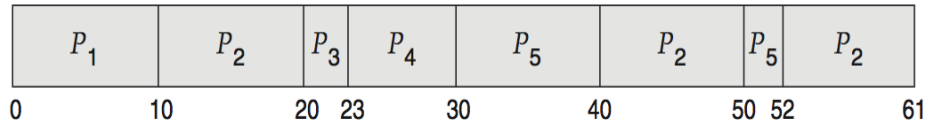
Deterministic Evaluation

- For each algorithm, calculate minimum average waiting time
- Simple and fast, but requires exact numbers for input, applies only to those inputs
  - FCS is 28ms:

  - Non-preemptive SFJ is 13ms:



  - RR is 23ms:



Queueing Models

- Describes the arrival of processes, and CPU and I/O bursts probabilistically, Commonly exponential, and described by mean:
  - Let's assume that processes arrive at the CPU on average **every 5 seconds**. **mean inter-arrival time** = **5 seconds**, in reality, some processes may arrive after 1 second, while others may arrive after 10 seconds. The distribution of these arrival times follows an exponential pattern around the mean.
  - Similarly, if the **mean CPU burst time** is 3 seconds, the CPU will spend an average of 3 seconds on each process, but some CPU bursts may last 1 second, and others may last 5 seconds.
- Computer system described as network of servers, each with queue of waiting processes:
  - Knowing arrival rates and service rates
  - Computes utilization, average queue length, average wait time, etc

Little's Formula

- Widely used theorem in queueing theory
- n = average queue length
- W = average waiting time in queue
- λ = average arrival rate into queue
- Little's law – in steady state, processes leaving queue must equal processes arriving, thus:

  $n = \lambda \times W$

  - Valid for any scheduling algorithm and arrival distribution
- For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

1. **Arrival Rate (λ):**
   - On average, **2 processes** arrive per second.
2. **Average Time in the System (W):**
   - Each process spends an average of **3 seconds** in the system (waiting in the queue and being processed by the CPU).
3. **Applying Little's Formula:**

$$L = \lambda \times W$$

$$L = 2 \text{ processes/second} \times 3 \text{ seconds}$$

$$L = 6 \text{ processes}$$

This means that, on average, there are **6 processes** in the system at any given time (either waiting in the queue or being processed by the CPU).

```
from collections import deque
# Process class to store the process information
class Process:
    def __init__(self, pid, arrival_time, burst_time):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.remaining_time = burst_time
        self.completion_time = 0

# FCFS scheduling algorithm
def fcfs_scheduling(processes):
    current_time = 0
    waiting_time = 0
    turnaround_time = 0
    print("\nFCFS Scheduling:")
    for process in processes:
        if current_time < process.arrival_time:
            current_time = process.arrival_time
        process.completion_time = current_time + process.burst_time
        waiting_time += current_time - process.arrival_time
        turnaround_time += process.completion_time - process.arrival_time
        current_time += process.burst_time
        print(f"Process {process.pid}: Start = {current_time - process.burst_time},\
              End = {process.completion_time}")

    avg_waiting_time = waiting_time / len(processes)
    avg_turnaround_time = turnaround_time / len(processes)
    print(f"Average Waiting Time: {avg_waiting_time:.2f}")
    print(f"Average Turnaround Time: {avg_turnaround_time:.2f}")
```

```
# Round Robin scheduling algorithm
def round_robin_scheduling(processes, quantum):
    current_time = 0
    waiting_time = 0
    turnaround_time = 0
    queue = deque(processes)  # Queue for round robin scheduling

    print("\nRound Robin Scheduling:")
    while queue:
        process = queue.popleft()
        if current_time < process.arrival_time:
            current_time = process.arrival_time
        if process.remaining_time > quantum:
            print(f"Process {process.pid}: Running from {current_time} to {current_time + quantum}")
            process.remaining_time -= quantum
            current_time += quantum
            queue.append(process)  # Process goes back to the queue if not finished
        else:
            print(f"Process {process.pid}: Running from {current_time} to {current_time + process.remain
            current_time += process.remaining_time
            process.completion_time = current_time
            waiting_time += process.completion_time - process.arrival_time - process.burst_time
            turnaround_time += process.completion_time - process.arrival_time
            process.remaining_time = 0

    avg_waiting_time = waiting_time / len(processes)
    avg_turnaround_time = turnaround_time / len(processes)
    print(f"Average Waiting Time: {avg_waiting_time:.2f}")
    print(f"Average Turnaround Time: {avg_turnaround_time:.2f}")
```

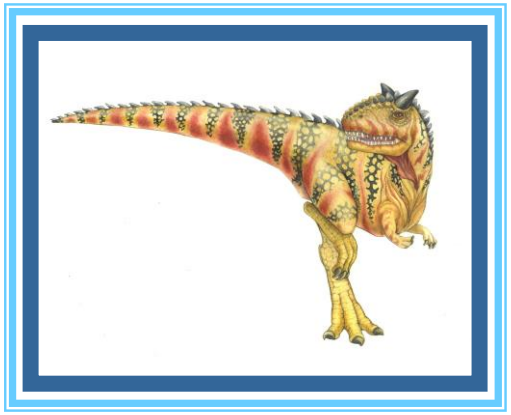
```
# Example Processes
process_list = [
    Process(1, 0, 10),   # Process 1: arrival_time=0, burst_time=10
    Process(2, 2, 5),    # Process 2: arrival_time=2, burst_time=5
    Process(3, 4, 8),    # Process 3: arrival_time=4, burst_time=8
    Process(4, 6, 6),    # Process 4: arrival_time=6, burst_time=6
]

# Run FCFS and Round Robin with a quantum of 4
fcfs_scheduling(process_list)
round_robin_scheduling(process_list, quantum=4)
```

```
FCFS Scheduling:
Process 1: Start = 0, End = 10
Process 2: Start = 10, End = 15
Process 3: Start = 15, End = 23
Process 4: Start = 23, End = 29
Average Waiting Time: 9.00
Average Turnaround Time: 16.25

Round Robin Scheduling:
Process 1: Running from 0 to 4
Process 2: Running from 4 to 8
Process 3: Running from 8 to 12
Process 4: Running from 12 to 16
Process 1: Running from 16 to 20
Process 2: Running from 20 to 21
Process 3: Running from 21 to 25
Process 4: Running from 25 to 27
Process 1: Running from 27 to 29
Average Waiting Time: 15.25
Average Turnaround Time: 22.50
```

End of Chapter 5



Silberschatz, Galvin and Gagne ©2018