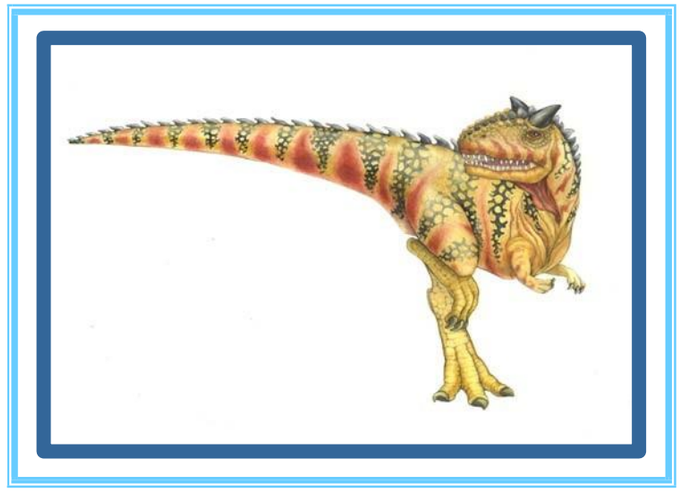
CSE-309
Operating Systems

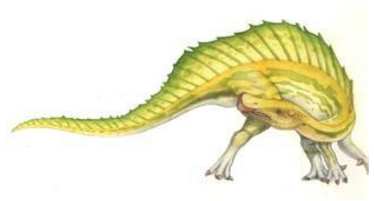
Mohammad Shariful Islam
Lecturer, Department of CSE
Contact no: 01747612143
Email: sharifulruhan@gmail.com

Chapter 5:
CPU Scheduling



Chapter 5: CPU Scheduling

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

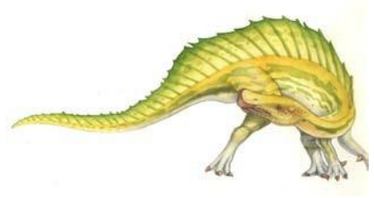


Objectives

- 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

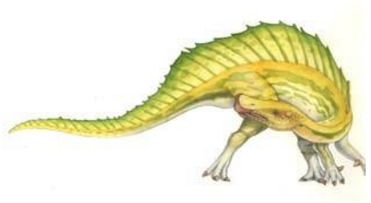
Basic Concepts

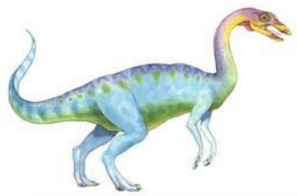
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait
- **CPU burst** distribution



CPU Scheduler

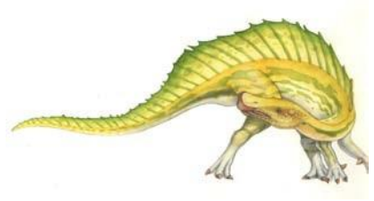
- Selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**
  - Windows 3.x used **nonpreemptive**
  - From Windows 95 introduced **preemptive**





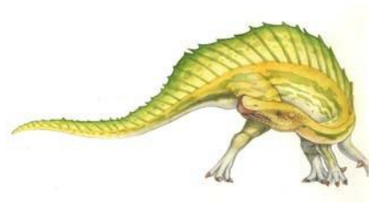
Dispatcher

- SCHEDULER : “When more than one process is runnable, the operating system must decide which one is first. The part of the operating system concerned with this decision is called the scheduler, and algorithm it uses is called the scheduling algorithm.”
- **Short term scheduler also known as dispatcher. Dispatcher is responsible for loading the selected process by Short Term scheduler on the CPU (Ready to Running State).**
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running



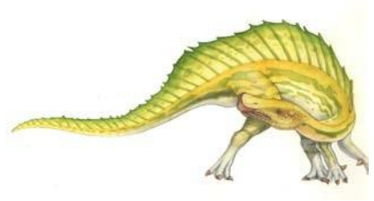
Scheduling Criteria

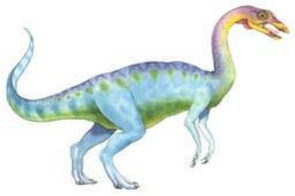
- **CPU utilization** – keep the CPU as busy as possible (should be 40-90% in real system)
- **Throughput** – # of processes that complete their execution per time unit (may be in seconds, milliseconds, or hours)
- **Turnaround time** – amount of time to execute a particular process; i.e., waiting time + execution time
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)



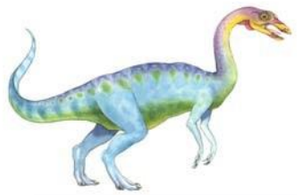
Scheduling Algorithm Optimization Criteria

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





Scheduling Algorithms



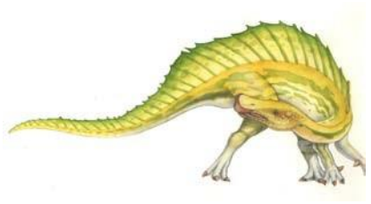
First-Come, First-Served (FCFS) Scheduling

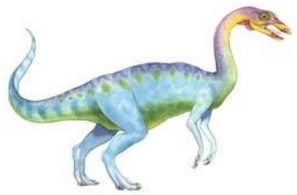
Process	Burst Time (ms)
P_1	24
P_2	3
P_3	3

- 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
- Turnaround time: P_1 = 24; P_2 = 27; P_3 = 30
- Average Turnaround time: (24 + 27+30)/3 = 27





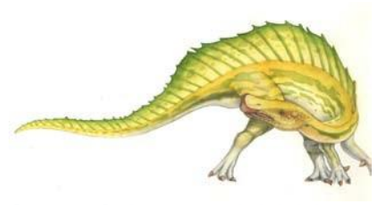
FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

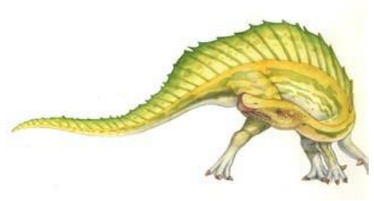
- The Gantt chart for the schedule is:

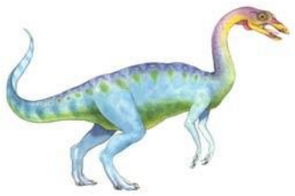
- Waiting time for $P_1 = 6; P_2 = 0, P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes



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

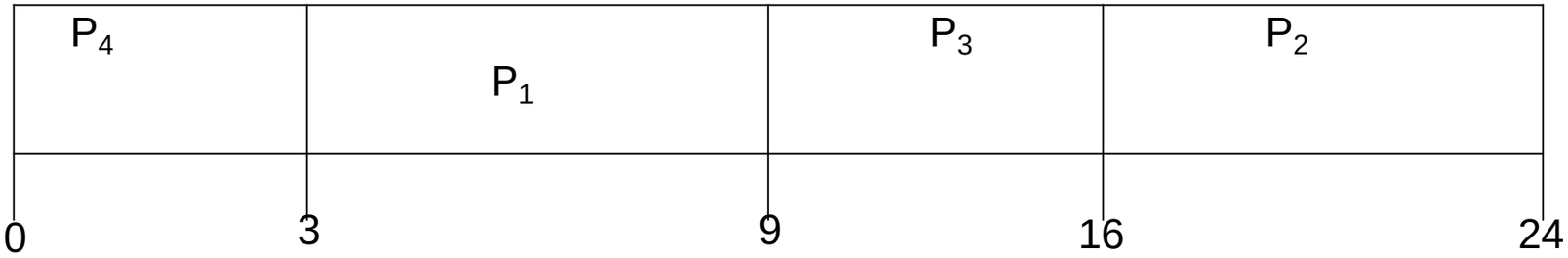




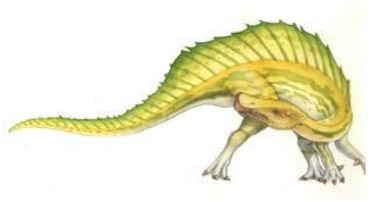
Example of SJF

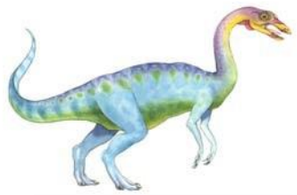
Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Waiting time for P_1 = 3; P_2 = 16; P_3 = 9; P_4 = 0;
- Average waiting time: (3 + 16 + 9+0)/4 = 7 milliseconds
- Turnaround time: P_1 = 9; P_2 = 24; P_3 = 16; P_4 = 3;
- Average Turnaround time: (9 + 24+16+3)/4 = 13
- Now do it for FCFS and then compare with SJF





Example of Shortest-remaining-time-first

- **SRTF**, Which Stands for **Shortest Remaining Time First.**
- can also be called as the preemptive version of the SJF scheduling algorithm.
- The process which has the least processing time remaining is executed first.
- As it is a preemptive type of schedule, it is claimed to be better than SJF scheduling Algorithm.

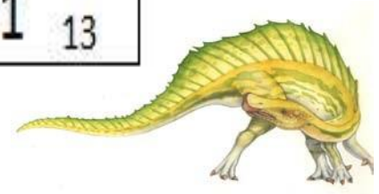
➢ Let's understand this with the help of an example.

➢ Suppose we have the following 3 processes with process ID's **P1**, **P2**, and **P3** and they arrive into the CPU in the following manner:

Process ID	Arrival Time (milliseconds)	Burst Time (milliseconds)
P1	0	8
P2	1	2
P3	4	3

➢ **Gantt Chart:**

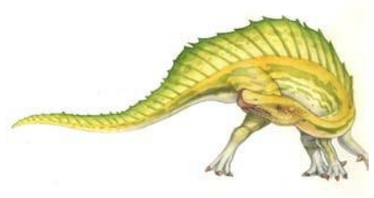
0 P1	1 P2	2 P2	3 P1	4 P3	5 P3	6 P3	7 P1	8 P1	9 P1	10 P1	11 P1	12 P1 13

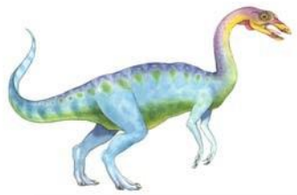


Example of Shortest-remaining-time-first

Explanation:

- At the 0th unit of the CPU, we have only process **P1**, so it gets executed for the 1-time unit.
- At the 1st unit of the CPU, the Process **P2** also arrives. Now, the **P1** needs 7 more units more to be executed, and **P2** needs only 2 units. So, **P2** is executed by preempting **P1**.
- **P2** gets completed at time unit 3, and unit now no new process has arrived. So, after the completion of **P2**, again **P1** is sent for execution.
- Now, **P1** has been executed for one unit only, and we have an arrival of new process **P3** at time unit 4. Now, the **P1** needs 6-time units more and **P3** needs only 3-time units. So, **P3** is executed by preempting **P1**.
- **P3** gets completed at time unit 7, and after that, we have the arrival of no other process. So again, **P1** is sent for execution, and it gets completed at 13th unit.



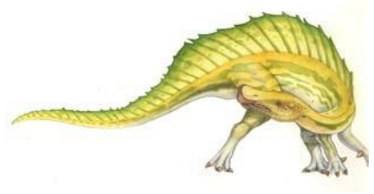


Example of Shortest-remaining-time-first

P ID	Arrival Time	Burst Time	Completion time (milliseconds)	Turn Around Time (milliseconds)	Waiting Time (milliseconds)
P1	0	8	13	13	5
P2	1	2	3	2	0
P3	4	3	7	3	0

- Total Turn Around Time = 13 + 2 + 3
- = 18 milliseconds
- Average Turn Around Time= Total Turn Around Time / Total No. of Processes
- = 18 / 3
- = 6 milliseconds

- Total Waiting Time = 5 + 0 + 0
- = 5 milliseconds
- Average Waiting Time = Total Waiting Time / Total No. of Processes
- = 5 / 3
- = 1.67 milliseconds

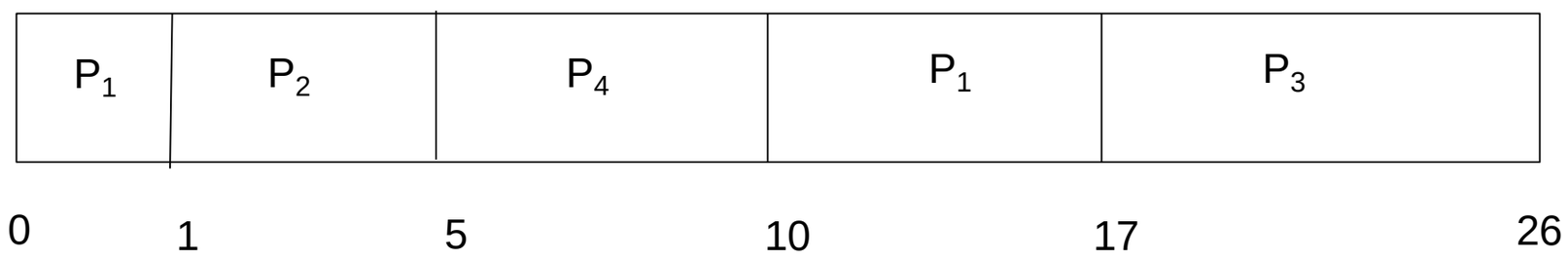


Another Example of Shortest-remaining-time-first

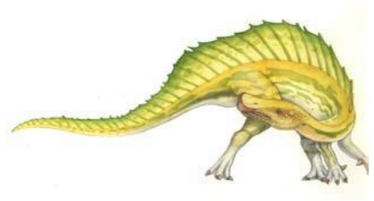
- Again consider this example

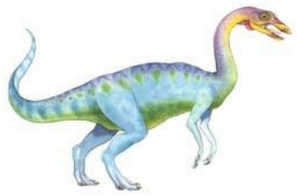
Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



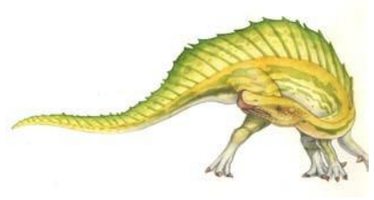
- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

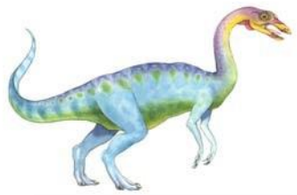




Priority Scheduling

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

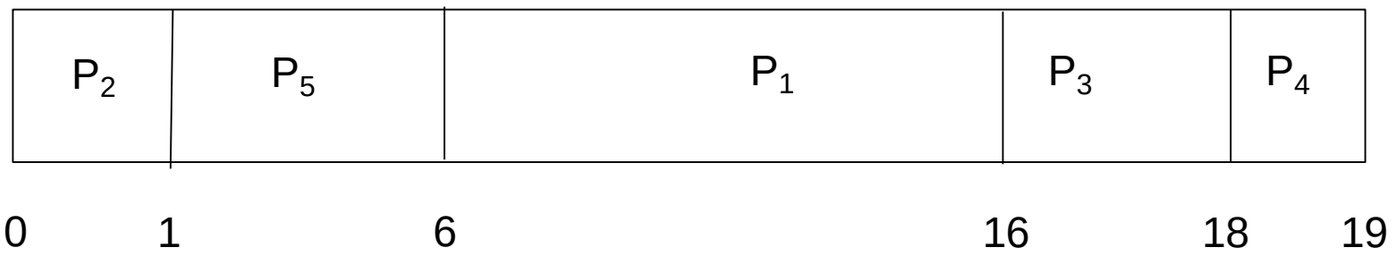




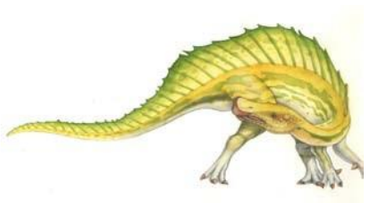
Example of Priority Scheduling

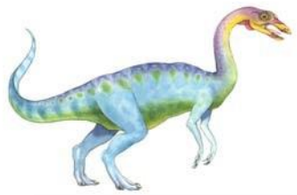
Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



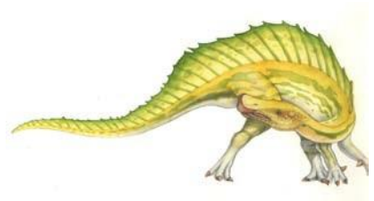
- Average waiting time = (0+1+6+16+18)/5=8.2 msec

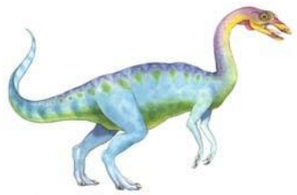




Round Robin (RR)

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

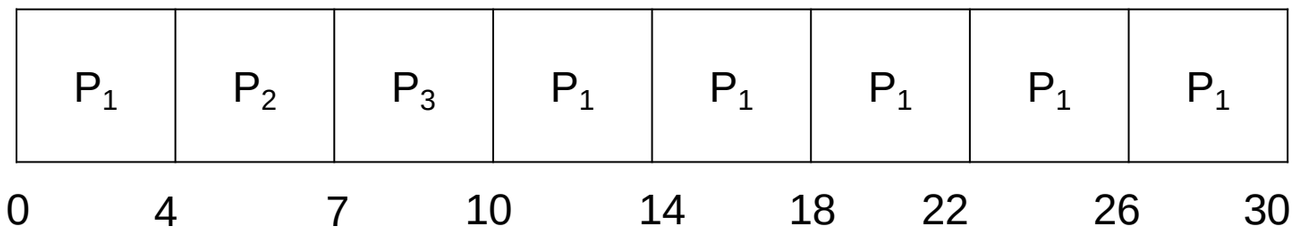




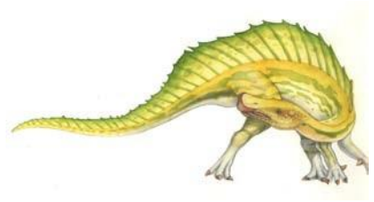
Example of RR with Time Quantum = 4

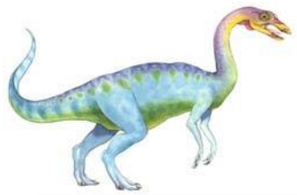
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



- Average waiting time? (6+4+7)/3=5.66 milliseconds.
- P_1 =(10-4)=6 ; P_2 = 4; P_3 = 7;
- 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



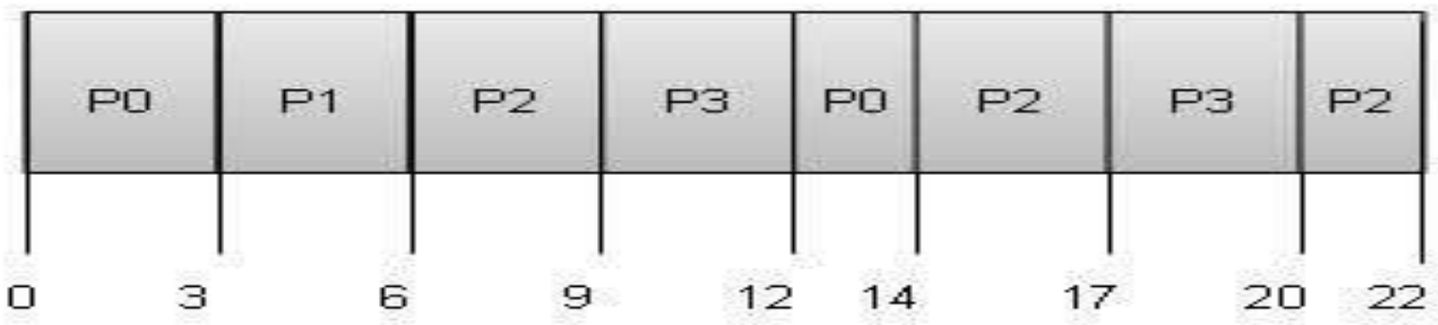


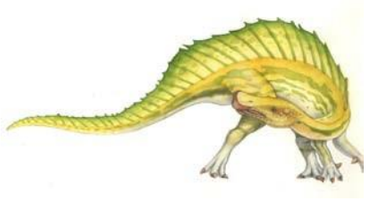
Example of RR with Time Quantum = 3

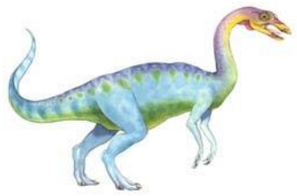
- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Process	Arrival Time	Execute Time	Service Time
P_0	0	5	0
P_1	1	3	5
P_2	2	8	8
P_3	3	6	16

Quantum = 3





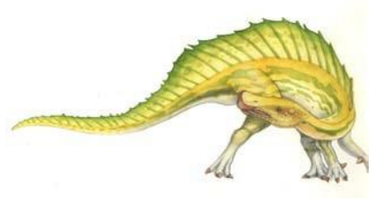


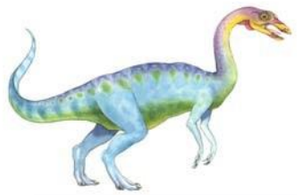
Example of RR with Time Quantum = 3

- **Wait time** of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

- Average Wait Time: (9+2+12+11) / 4 = 8.5
-

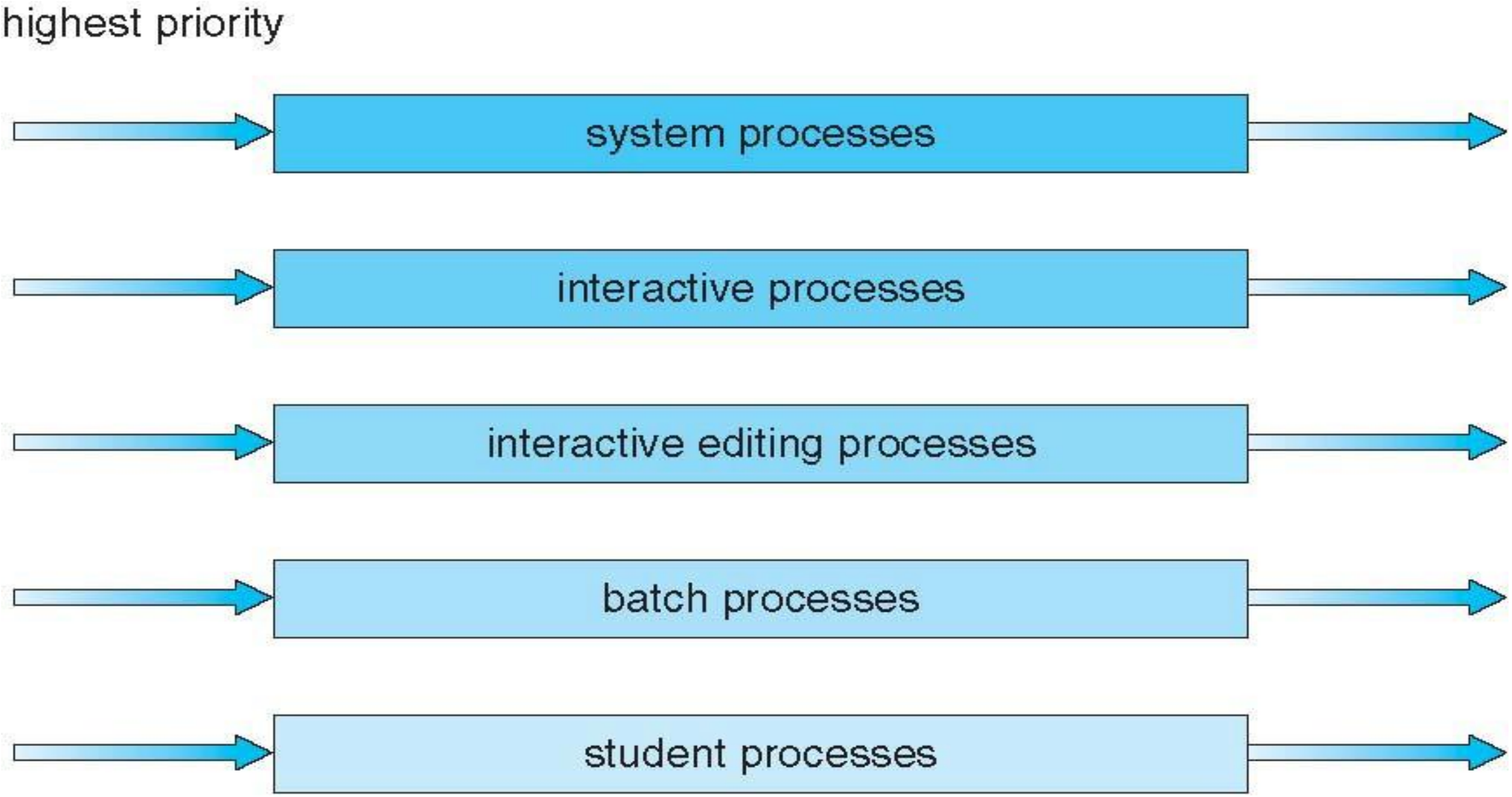




Multilevel Queue

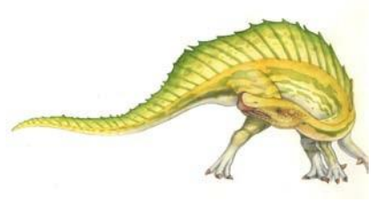
- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Queue Scheduling



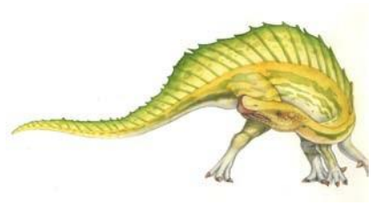
Multilevel Feedback Queue

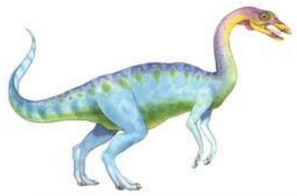
- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine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



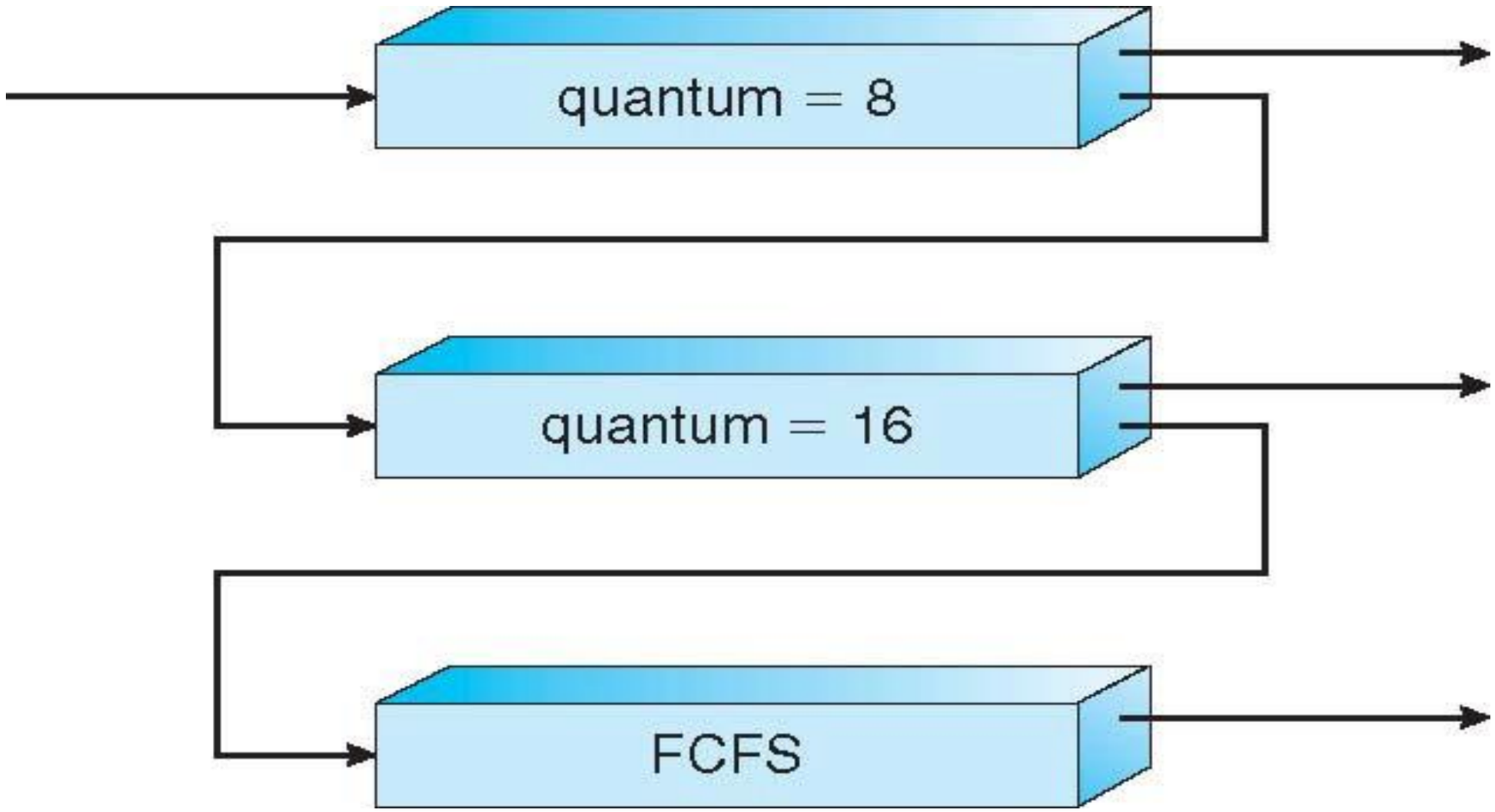
Example of Multilevel Feedback Queue

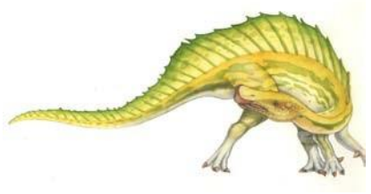
- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue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





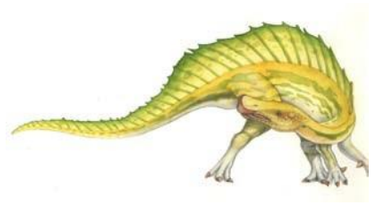
Multilevel Feedback Queues

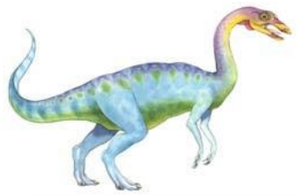




Windows Scheduling

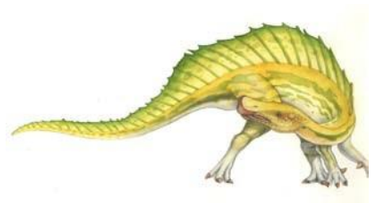
- Windows uses priority-based preemptive scheduling
- Highest-priority thread runs next
- *Dispatcher* is scheduler
- Thread runs until (1) blocks, (2) uses time slice, (3) preempted by higher-priority thread
- Real-time threads can preempt non-real-time
- 32-level priority scheme
- **Variable class** is 1-15, **real-time class** is 16-31
- Priority 0 is memory-management thread
- Queue for each priority
- If no run-able thread, runs **idle thread**





Linux Scheduling

- Constant order *O*(1) scheduling time
- Preemptive, priority based
- Two priority ranges: time-sharing and real-time
- **Real-time** range from 0 to 99 and **nice** value from 100 to 140
- Map into global priority with numerically lower values indicating higher priority
- Higher priority gets larger q
- Task run-able as long as time left in time slice (**active**)
- If no time left (**expired**), not run-able until all other tasks use their slices
- All run-able tasks tracked in per-CPU **runqueue** data structure
  - Two priority arrays (active, expired)
  - Tasks indexed by priority
  - When no more active, arrays are exchanged



Exercise 1/ Assignment 1

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

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

Now draw the Gantt chart and find the average waiting time, turnaround time for the following scheduling algorithms:

- **FCFS**
- **SJF**
- **Priority**
- **RR (time quantum time=2)**

End of Chapter 5