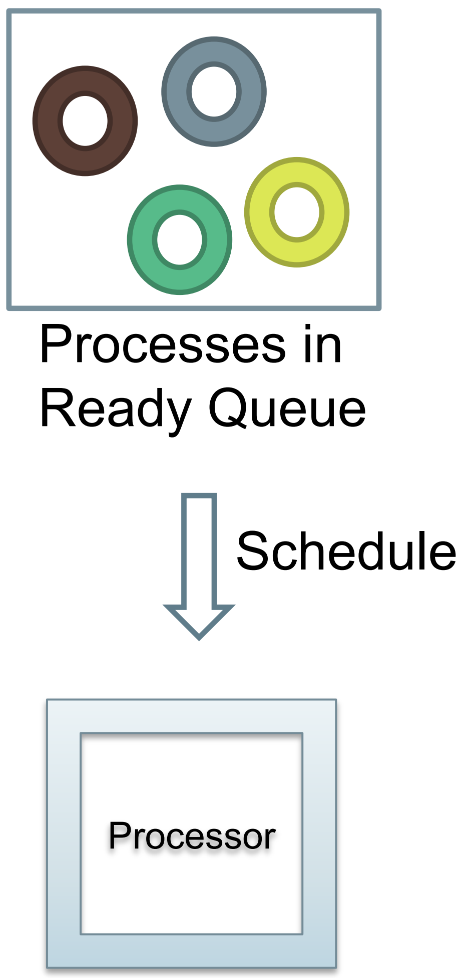
OPERATING SYSTEMS

CPU Scheduling

Basic Concepts

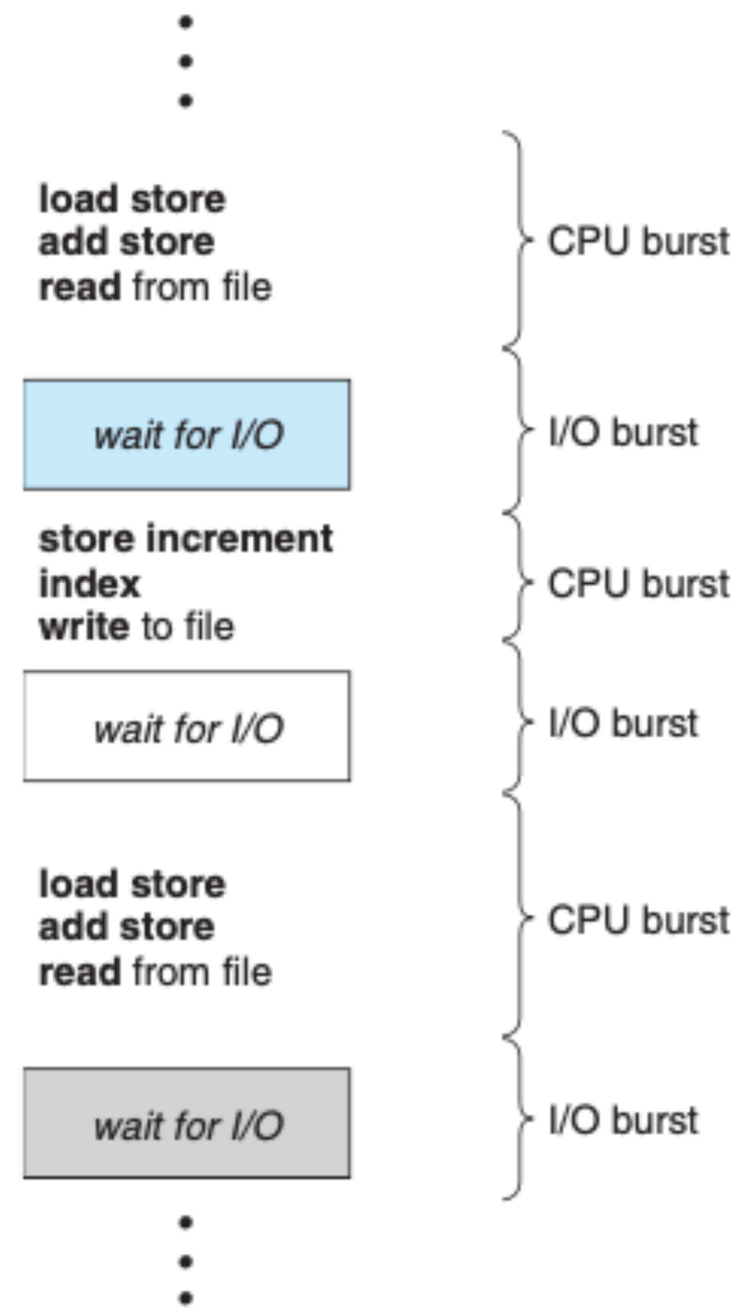
- Maximum CPU utilization obtained with multiprogramming

- Continuous Cycle :
    - one process has to wait (I/O)
    - Operating system takes the CPU away
    - Give CPU to another process
    - This pattern continues



Basic Concepts

- CPU–I/O Burst Cycle : Process execution consists of a *cycle* of CPU execution and I/O wait



CPU Scheduler

- **Selects from among the processes in ready queue, and allocates the CPU to one of them**
  - FIFO queue
  - Priority queue
  - Tree
  - Unordered linked-list
- **CPU scheduling decisions may take place when a process:**
  1. Switches from running to waiting state (I/O request)
  2. Switches from running to ready state (e.g. when interrupt occurs)
  3. Switches from waiting to ready (e.g. at completion of I/O)
  4. Terminates
- Scheduling under 1 and 4 is ***nonpreemptive***
- All other scheduling is ***preemptive***
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible

- **Throughput** – # of processes that complete their execution per time unit

- **Turnaround time**
  – amount of time to execute a particular process
  -- the interval from the time of submission of a process to the time of the completion.
  -- sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, doing I/O

- **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

OPERATING SYSTEMS

CPU Scheduling Algorithms - First Come First Serve (FCFS)

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:

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:

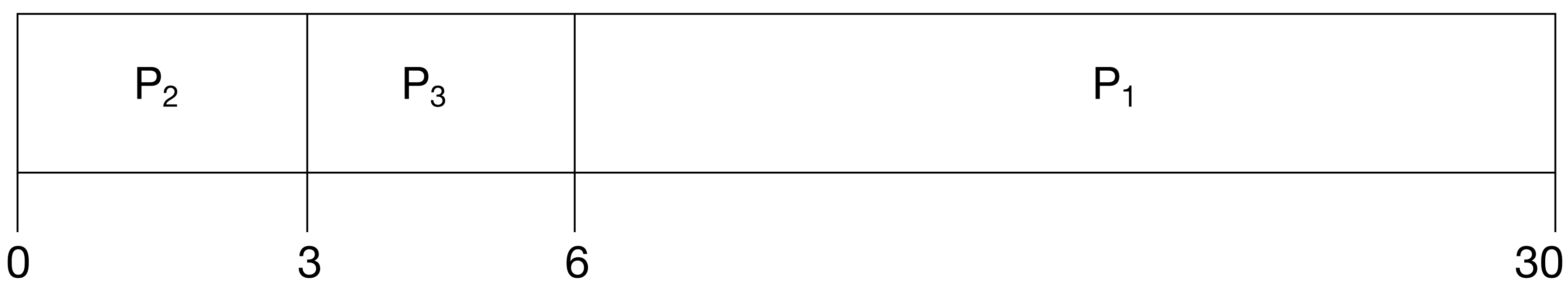
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17
- Turnaround time P_1 = 24; P_2 = 27; P_3 = 30

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

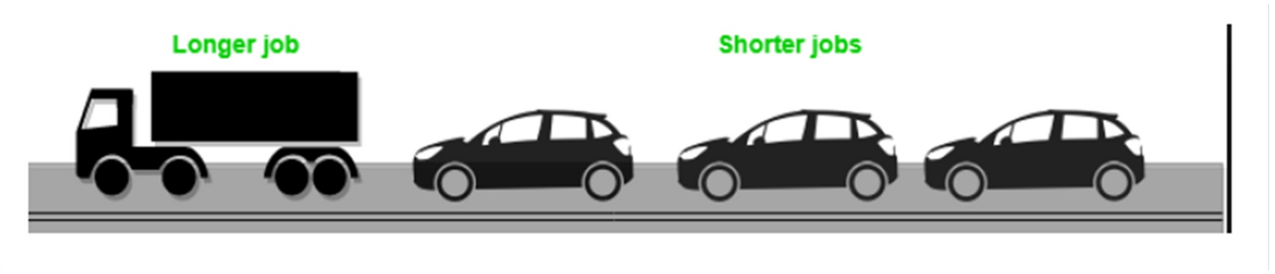
P_2 , P_3 , P_1

- 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



OPERATING SYSTEMS

CPU Scheduling Algorithms - Shortest Job First (SJF)

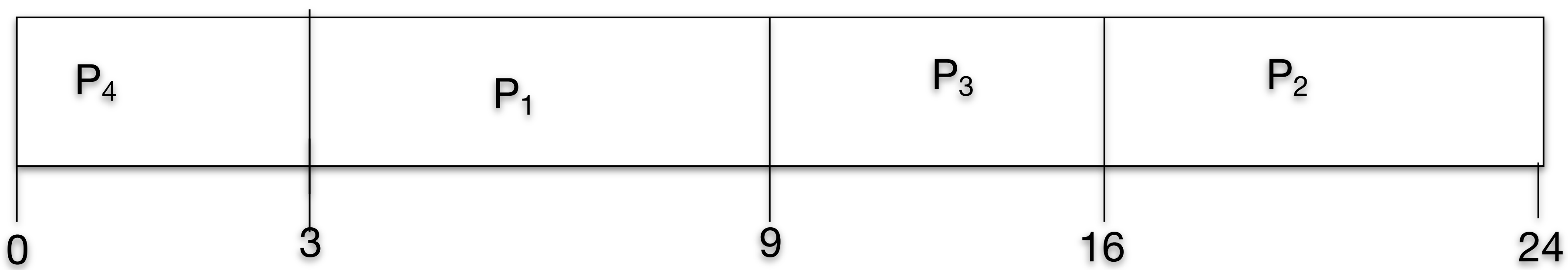
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - **Non-preemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst
  - **preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart

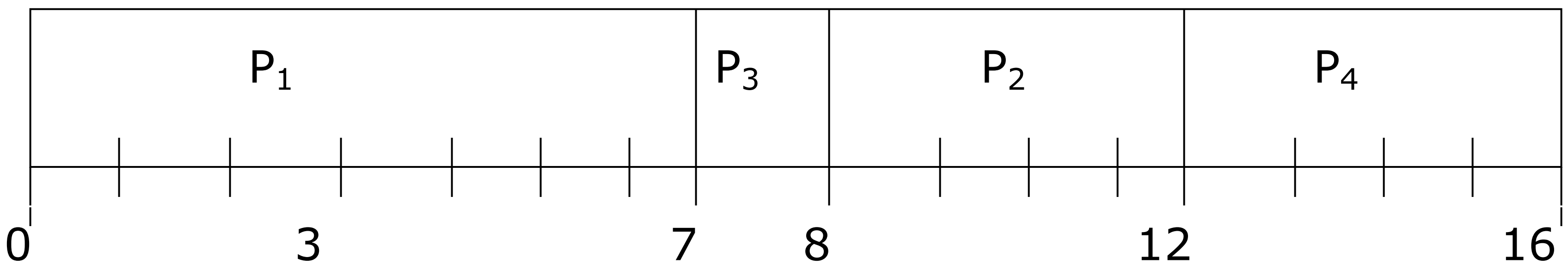


Waiting time for $P_1 = 3$; $P_2 = 16$; $P_3 = 9$; $P4 = 0$
Average waiting time: $(3 + 16 + 9 + 0)/4 = 7$ ms

Example of Non-Preemptive SJF

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

- SJF (non-preemptive)



Waiting time for $P_1 = 0$; $P_2 = (8 - 2) = 6$; $P_3 = (7 - 4) = 3$; $P_4 = (12 - 5) = 7$

Average waiting time: $(0 + 6 + 3 + 7)/4 = 4$ ms

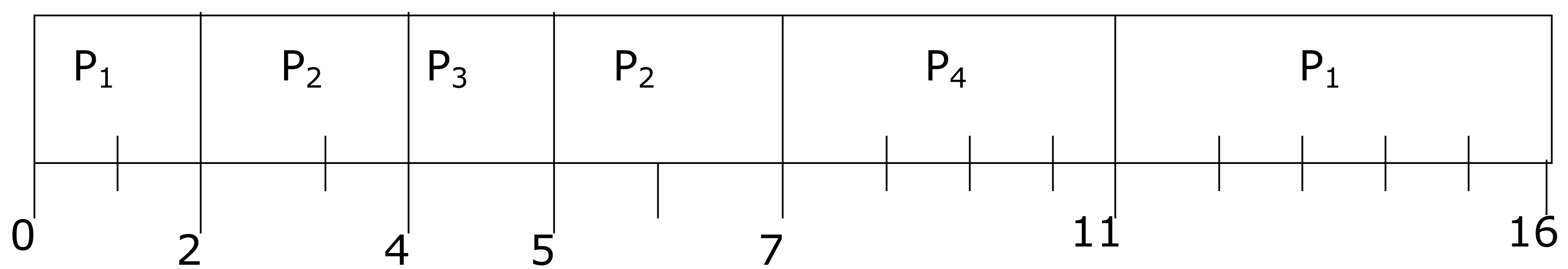
Average turnaround time: $= (7 + (12 - 2) + (8 - 4) + (16 - 5)) / 4 =$

$= (7 + 10 + 4 + 11)/4 = 8$ms

Example of Preemptive SJF

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

- SJF (preemptive)



Waiting time for $P_1 = 0 + (11 - 2) = 9$; $P_2 = (2 - 2) + (5 - 4) = 1$;
$P_3 = (4 - 4) = 0$; $P_4 = (7 - 5) = 2$

Average waiting time: $(9 + 1 + 0 + 2)/4 = 3$ *ms*

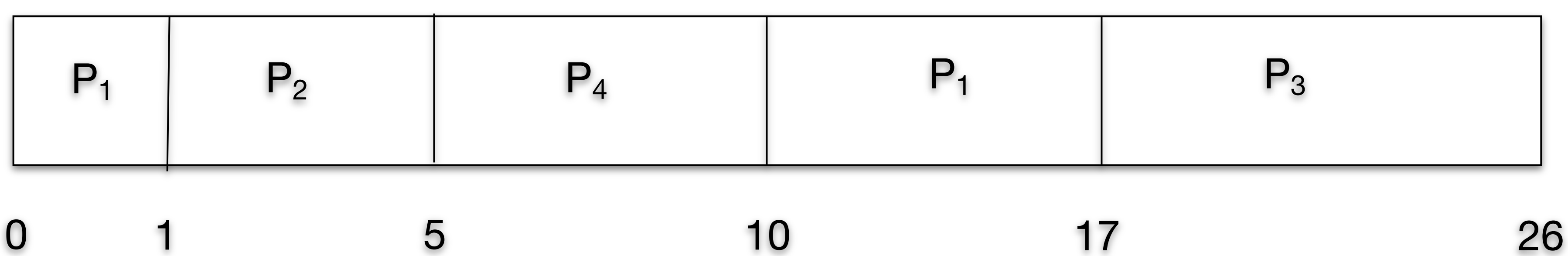
Average turnaround time: $= ((16 - 0) + (7 - 2) + (5 - 4) + (11 - 5)) / 4 =$
$= (16 + 5 + 1 + 6)/4 = 7$*ms*

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
P1	0	8
P2	1	4
P3	2	9
P4	3	5

- *Preemptive* SJF Gantt Chart



Waiting time for $P_1 = 0 + (10 - 1) = 9$; $P_2 = (1 - 1) = 0$;
$P_3 = (17 - 2) = 15$; $P_4 = (5 - 3) = 2$

Average waiting time: $(9 + 0 + 15 + 2)/4 = 6.5$ *ms*

Average turnaround time: $= ((17 - 0) + (5 - 1) + (26 - 2) + (10 - 3)) / 4 =$
$= (17 + 4 + 24 + 7)/4 = 13$*ms*

OPERATING SYSTEMS

CPU Scheduling Algorithms - Priority Scheduling

Priority Scheduling

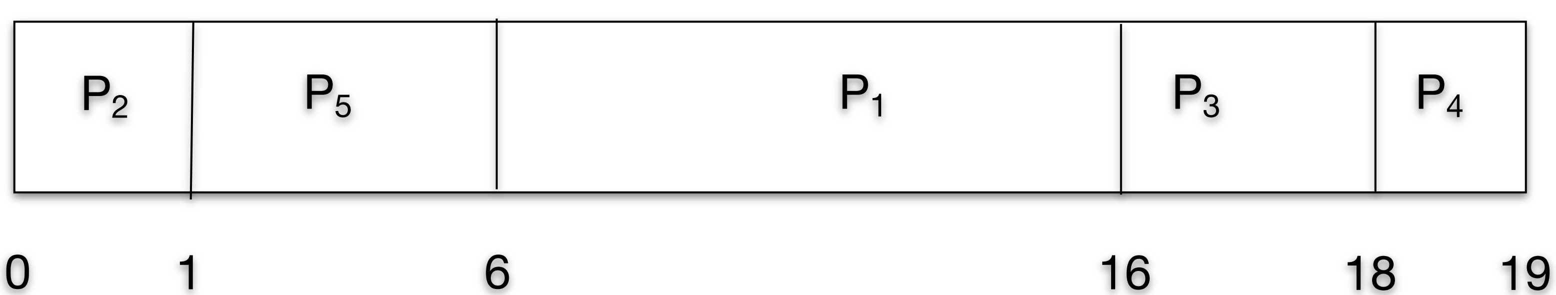
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
    - Preemptive
    - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Priority can be defined either internally or externally.
    - Factors for internal priority assignment:
        - Time limit, memory requirements, the number or open files etc.
    - Factors for external priority assignment:
        - Importance of the process, the type and amount of funds being paid for computer use, department sponsoring works etc.

Example of Priority Scheduling

Non Preemptive:

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: $(6 + 0 + 16 + 18 + 1)/5 = 8.2$
Average turnaround time: = (16 + 1 + 18 + 19 + 6)/5 = 12

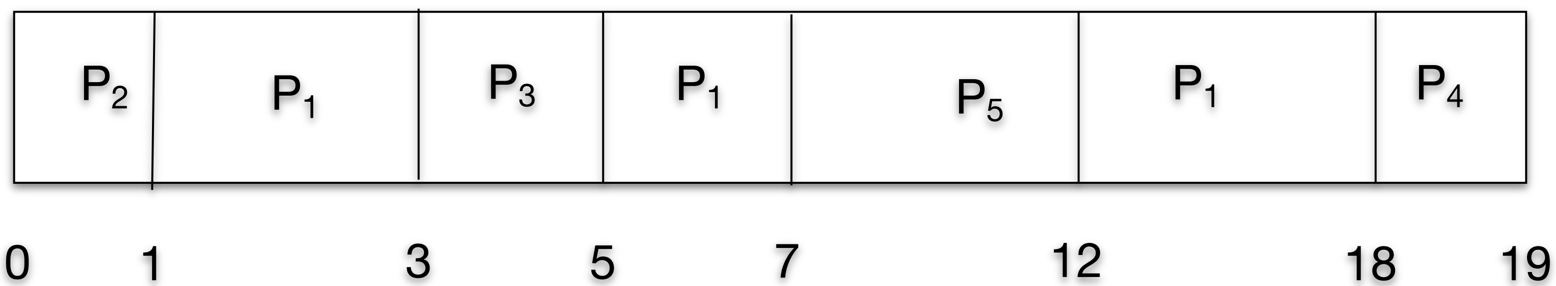
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

Preemptive:

Process	Arrival Time	Burst Time	Priority
P_1	0	10	4
P_2	0	1	1
P_3	3	2	3
P_4	5	1	5
P_5	7	5	2

- Priority scheduling Gantt Chart



Average waiting time: *(8 + 0 + 0 + 13 + 0)/5 = 4.2*
Average turnaround time: = (18 + 1 + 2 + 14 + 5)/5 = 8

- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

OPERATING SYSTEMS

CPU Scheduling Algorithms - Round Robin (RR)

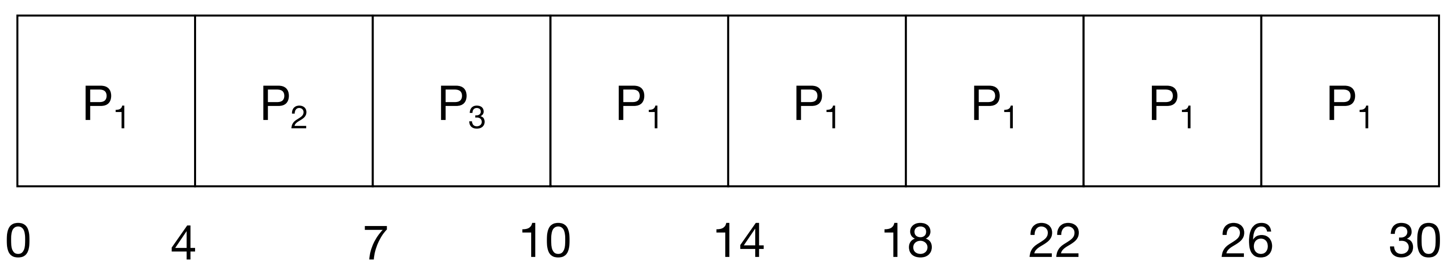
Round Robin (RR)

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

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

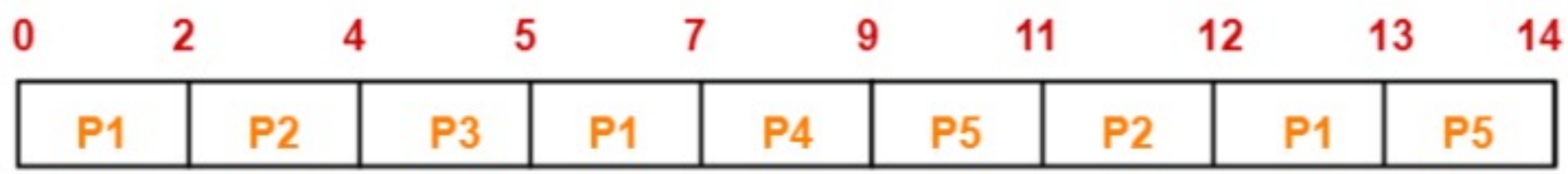
- The Gantt chart is:



- Average waiting time is 17 / 3 = 5.66 milisecond
- Typically, higher average turnaround than SJF, but better *response*
- quantum should be large compared to context switch time
- Quantum usually 10ms to 100ms, context switch < 10 usec

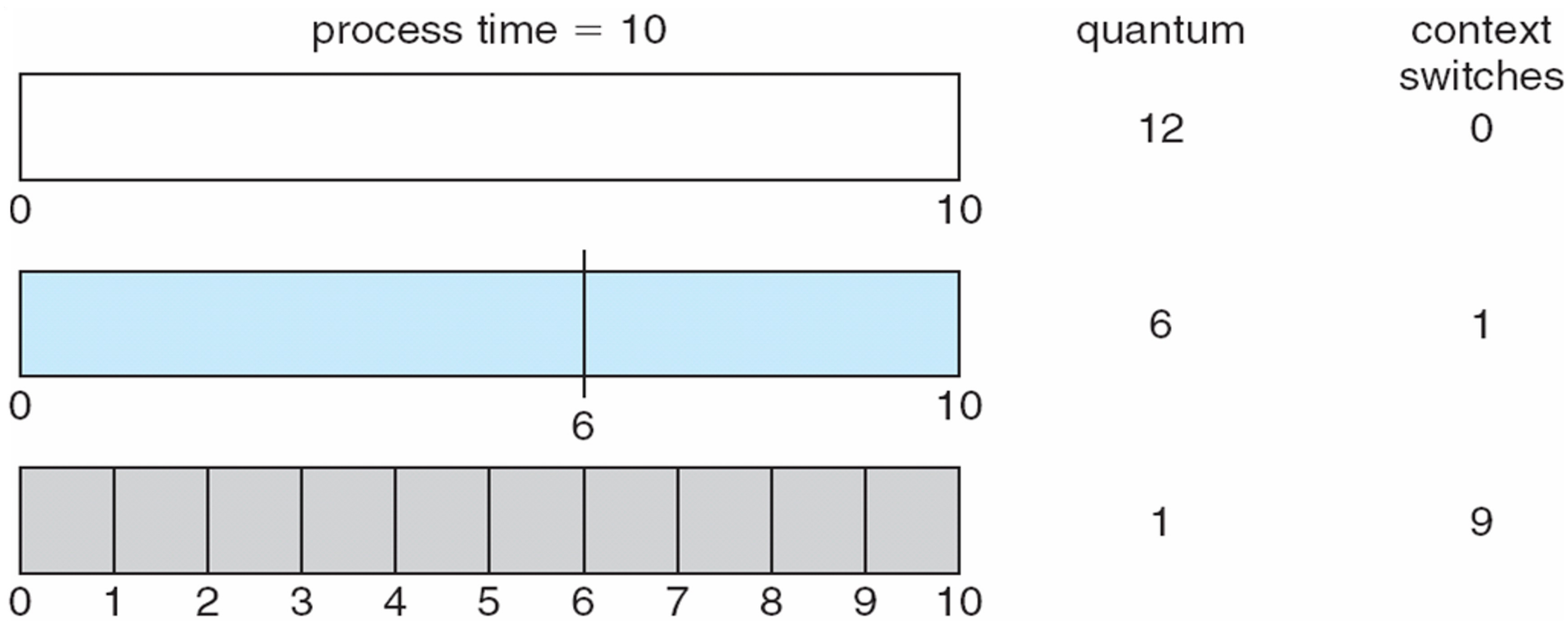
Example of RR with Time Quantum = 2

Process	Arrival Time	Burst Time
P1	0	5
P2	*1*	3
P3	2	1
P4	3	2
P5	*4*	3

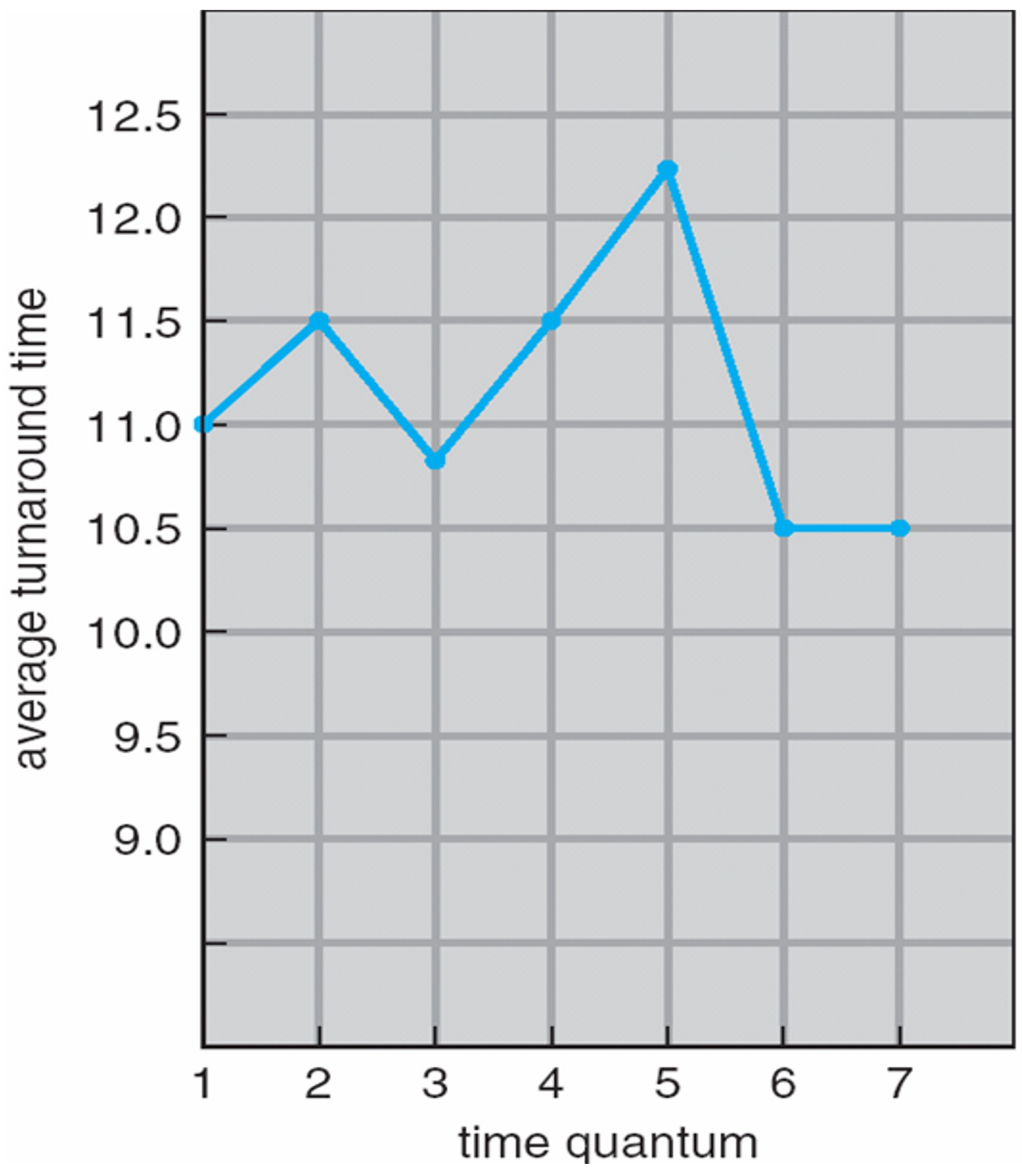


- Average waiting time is (8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8
- Average waiting time is (13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

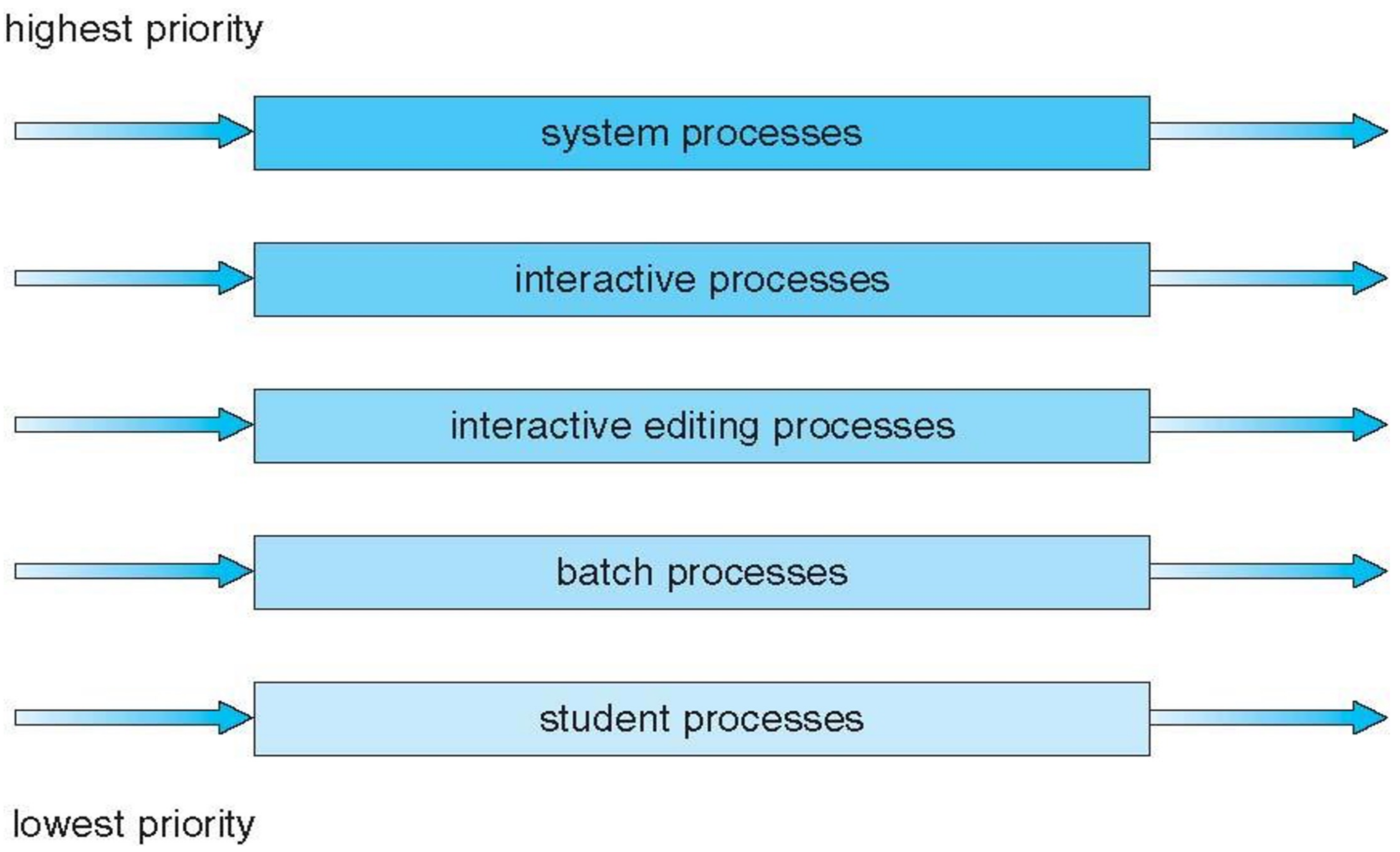
80% of CPU bursts should be shorter than quantum

OPERATING SYSTEMS

CPU Scheduling - Multilevel Queue, Multilevel Feedback Queue

Multilevel Queue

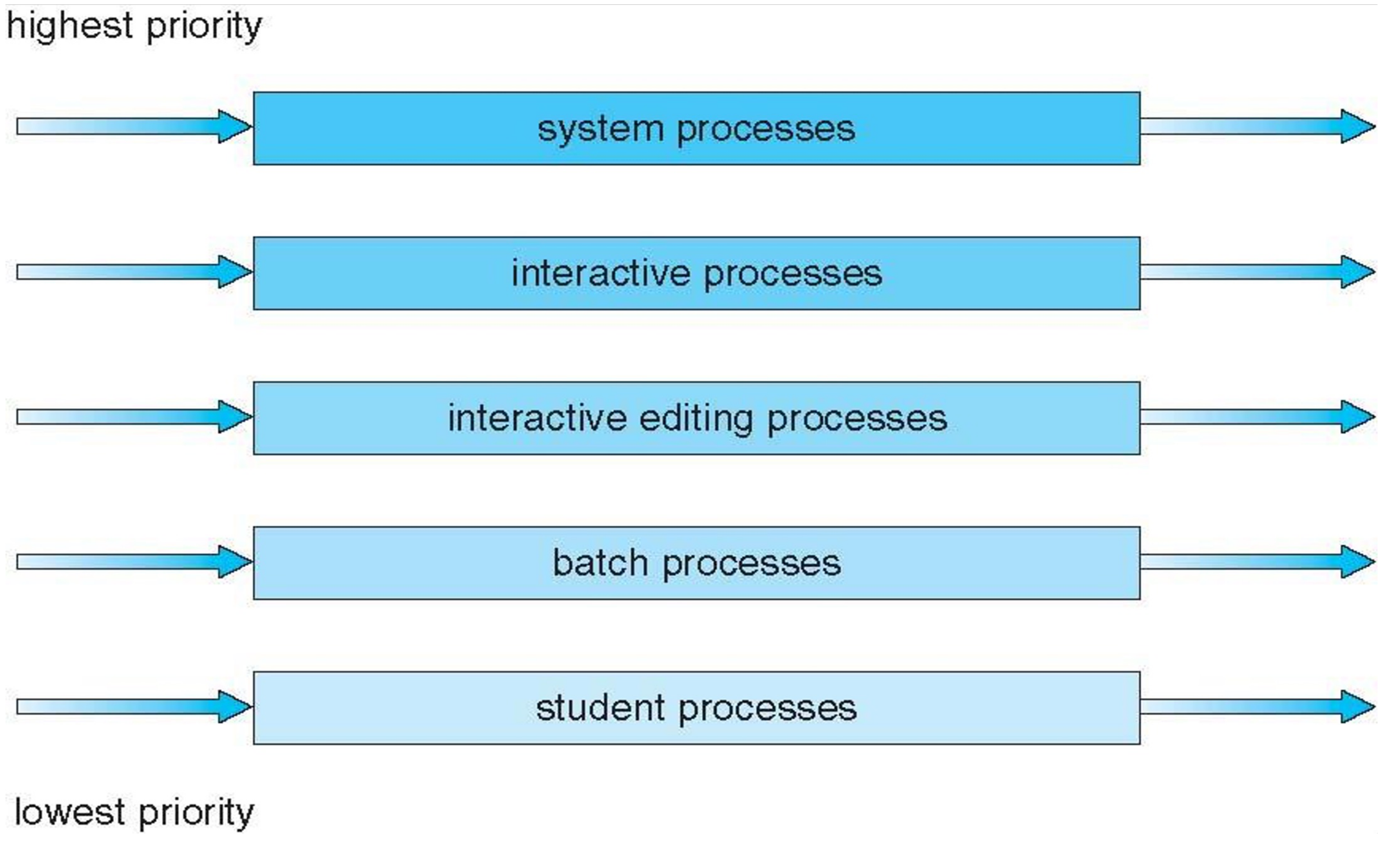
- Another class of scheduling algorithm needs- in which processes are classified into different groups, e.g.:
    - foreground (interactive) processes
    - background (batch) processes
- They have different response time requirements-so different scheduling needs.
- Foreground processes may have priority over background processes.
- A multilevel queue-scheduling algorithm partitions the ready queue into several separate queues



Multilevel Queue

- Each queue has its own scheduling algorithm:
  - Foreground queue scheduled by – RR algorithm
  - Background queue scheduled by – FCFS algorithm

- Scheduling must be done between the queues:
  - Fixed priority preemptive 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., foreground queue can be given 80% of the CPU time for RR-scheduling among its processes, while 20% to background in FCFS manner.



Multilevel Feedback Queue scheduling

- Multilevel Feedback Queue scheduling, allows a process to move between queues.
- If a process uses too much CPU time, it will be moved to a lower priority queue.
- Similarly, a process that waits too long in a lower-priority queue may me moved to a higher-priority queue.

- 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: (can see the figure in next slide)
  - 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 for RR
    - 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 RR and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2