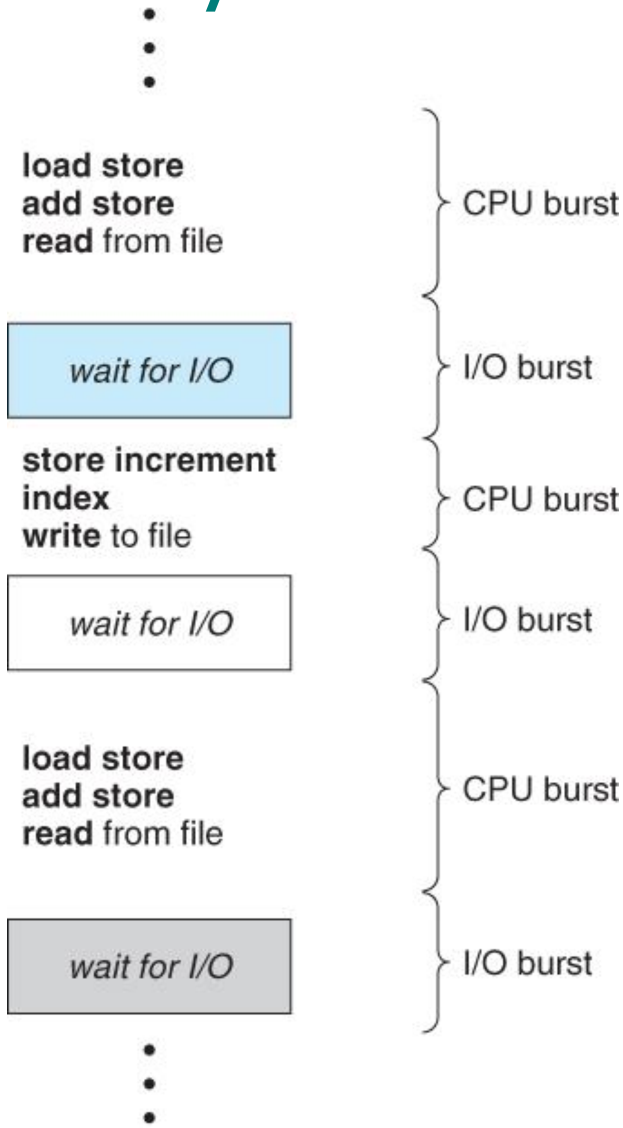
CPU Scheduling

Dr. Manmath N. Sahoo
Dept. of CSE, NIT Rourkela

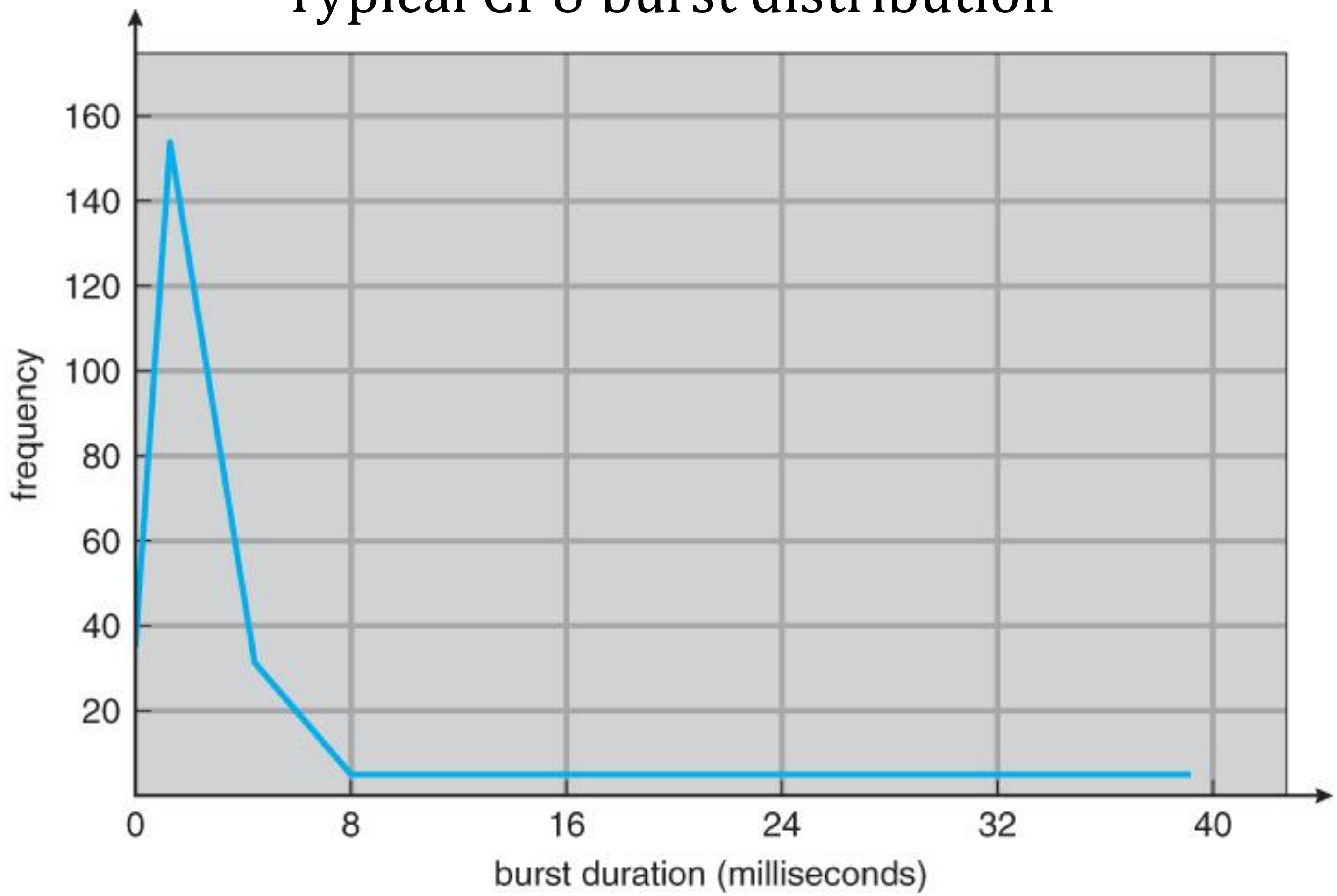
CPU-I/O Burst Cycle



- CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait.
- CPU burst is length of time process needs to use CPU before it next makes a I/O request.
- I/O burst is the length of time process spends waiting for I/O to complete.

Histogram of CPU-burst Times

Typical CPU burst distribution



CPU Scheduler

- Allocates CPU to one of processes that are ready to execute (in ready queue)
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state (e.g. I/O request)
  2. Terminates
  3. Switches from waiting to ready (e.g. I/O completion)
  4. Switches from running to ready state (e.g. priority based interruption)
- Scheduling only when 1 and 2 happen is **nonpreemptive** - process keeps CPU until it voluntarily releases it
- Scheduling also when 3 & 4 happen is **preemptive** - CPU can be taken away from process by OS

Dispatcher

- **Dispatcher** gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode (if CPU was executing the previous process in kernel mode and preempted)
  - jumping to the proper location in the user program to restart that program (i.e., last action is to set program counter)
- **Dispatch latency** – time it takes for the dispatcher to switch between processes and start new one running

Scheduling Criteria

- **CPU utilization:** i.e. CPU usage – want to maximize
- **Throughput:** number of processes that complete their execution per time unit – want to maximize
- **Turnaround time:** amount of time to execute a particular process – want to minimize
- **Waiting time:** amount of time a process has been waiting in the ready queue – want to minimize
- **Response time:** amount of time it takes from when a job was submitted until it initiates its first response (output) – want to minimize

First-Come, First-Served Scheduling

- Schedule: order of arrival of process in ready queue

Example:

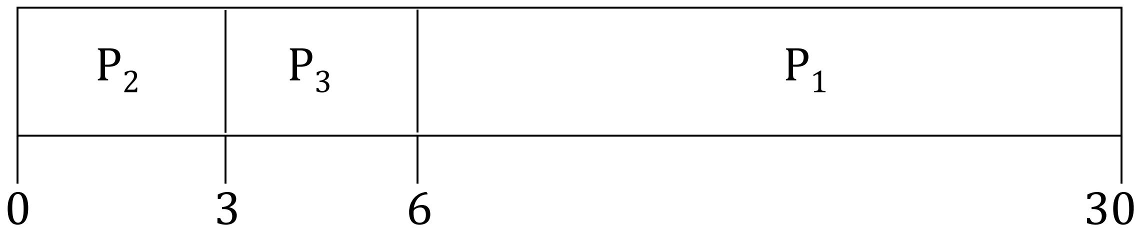
Process	Burst Time
P_1	24
P_2	*3*
P_3	*3*

- 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

- 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: $(6 + 0 + 3)/3 = 3$

FCFS Scheduling

+ Simple to implement by FIFO queue.
+ Code is easy to write and understand
- Average waiting time is not minimal
- Not suitable for time-sharing systems
- Convoy effect: short I/O intensive processes behind long CPU intensive process

▶ NOTE: FCFS Scheduling is non-preemptive

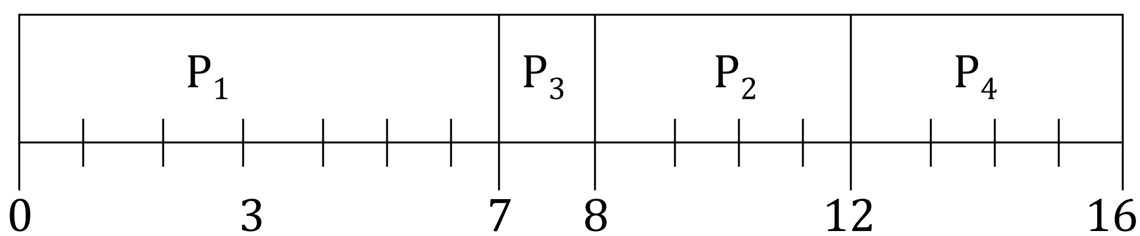
Shortest-Job-First (SJF) Scheduling

- Schedule: order of the CPU burst of the processes.
- Two schemes:
  - Non-Preemptive SJF / Shortest Job Next (SJN)
    - once CPU given to the process it cannot be preempted until completes its CPU burst.
  - Preemptive SJF / Shortest Remaining Time First (SRTF)
    - if a new process arrives with CPU burst length less than remaining time of current executing process, preempt.

Non-Preemptive SJF (SJN)

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Gantt Chart:

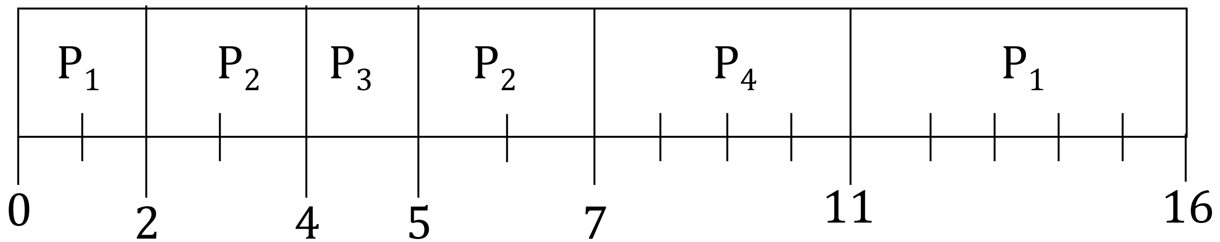


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Preemptive SJF (SRTF)

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Gantt Chart:



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Example

Process	Arrival Time	Burst Time
P1	0	16
P2	2	8
P3	4	12
P4	6	2
P5	8	4

- Draw Gantt chat and find the average waiting time in each of the following cases:
  - SRTF Scheduling
  - SJN Scheduling
  - FCFS Scheduling

Answer to Example

Process	Arrival Time	Burst Time
P1	0	16
P2	2	8
P3	4	12
P4	6	2
P5	8	4

- SRTF

P1	P2	P2	P4	P2	P5	P3	P1

0 2 4 6 8 12 16 28 42

- Average waiting time = (26+6+12+0+0)/5

  = 8.8

Answer to Example

Process	Arrival Time	Burst Time
P1	0	16
P2	2	8
P3	4	12
P4	6	2
P5	8	4

- SJN

P1	P4	P5	P2	P3

0 16 18 22 30 42

- Average waiting time = (0+20+26+10+10)/5

  = 13.2

Answer to Example

Process	Arrival Time	Burst Time
P1	0	16
P2	2	8
P3	4	12
P4	6	2
P5	8	4

- FCFS

P1	P2	P3	P4	P5

0 16 24 36 38 42

- Average waiting time = (0+14+20+30+30)/5

  = 18.8

Answer to Example

Scheduling Algorithm	Average Waiting Time
FCFS	18.8
SJN	13.2
SRTF	8.8

Shortest Job First

+ SRTF is Optimal: Minimum Average waiting time.
- Longer processes may be starved.
- Difficult to know the CPU burst of processes before their execution

Determining Length of Next CPU Burst

- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging (decaying average).

Determining Length of Next CPU Burst

1. $t_n =$ actual lenght of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define: $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$

Determining Length of Next CPU Burst

- $\alpha = 0$, $\tau_{n+1} = \tau_n$
  - last CPU burst does not count - only longer term history
- $\alpha = 1$, $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts.
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j\, \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1}\, \tau_0$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

Round Robin (RR) Scheduling

- Each process gets a **time quantum**, 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.
- No process waits more than $(n\text{-}1)q$ time units.

Example: RR with q= 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

- Typically, higher average turnaround than SJF, but better *response*.

Round Robin (RR) Scheduling

- FCFS + Preemption = RR
- Performance
  - q large $\Rightarrow$ behaves similar to FCFS
  - q small $\Rightarrow$ more context switch overhead

Modified RR Scheduling

- Based on variable time quantum
- A preempted process, on its next turn will use an increased time quantum
- RR Scheduling
  - P_1: 300ms, q=20ms, time per context switch=2ms
  - #of preemptions due to this process = 14
  - Preemption overhead = 28ms

Modified RR Scheduling

- Modified RR Scheduling – q is doubled in next turn.
  - P_1: 300ms, q=20ms (initial)
  - #of preemptions due to this process = 3. (20+40+80+160=300)
  - Preemption overhead = 6ms

Priority Scheduling

- A priority number (integer) is associated with each process and the CPU is allocated to the process with the highest priority
- Some schemes use smallest integer ≡ highest priority.
- SJF is effectively priority scheduling where priority is defined on next CPU burst time.
- FCFS is effectively priority scheduling where priority is defined on arrival time.

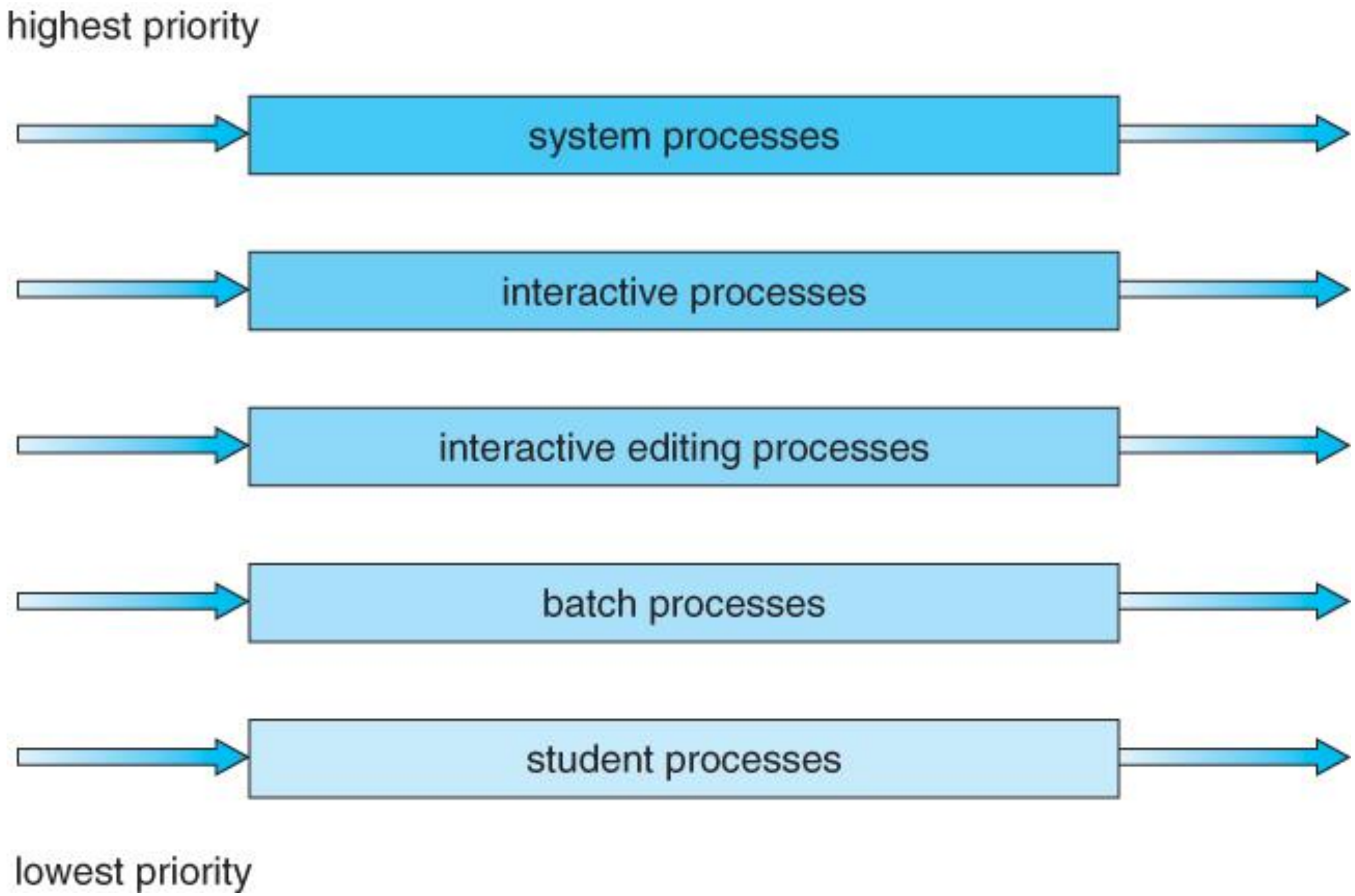
Priority Scheduling

- Problem: **Starvation** – low priority processes may never execute.
- Solution: **Aging** – as time progresses increase the priority of the process.
  - e.g. in every 10ms increase the priority of process by 1

Multi level Queue Scheduling

- Ready queue is partitioned into separate queues.
- Each queue holds a particular category of processes.
- Each queue can implement whatever scheduling algorithm is most appropriate for that type of processes.

Multi level Queue Scheduling



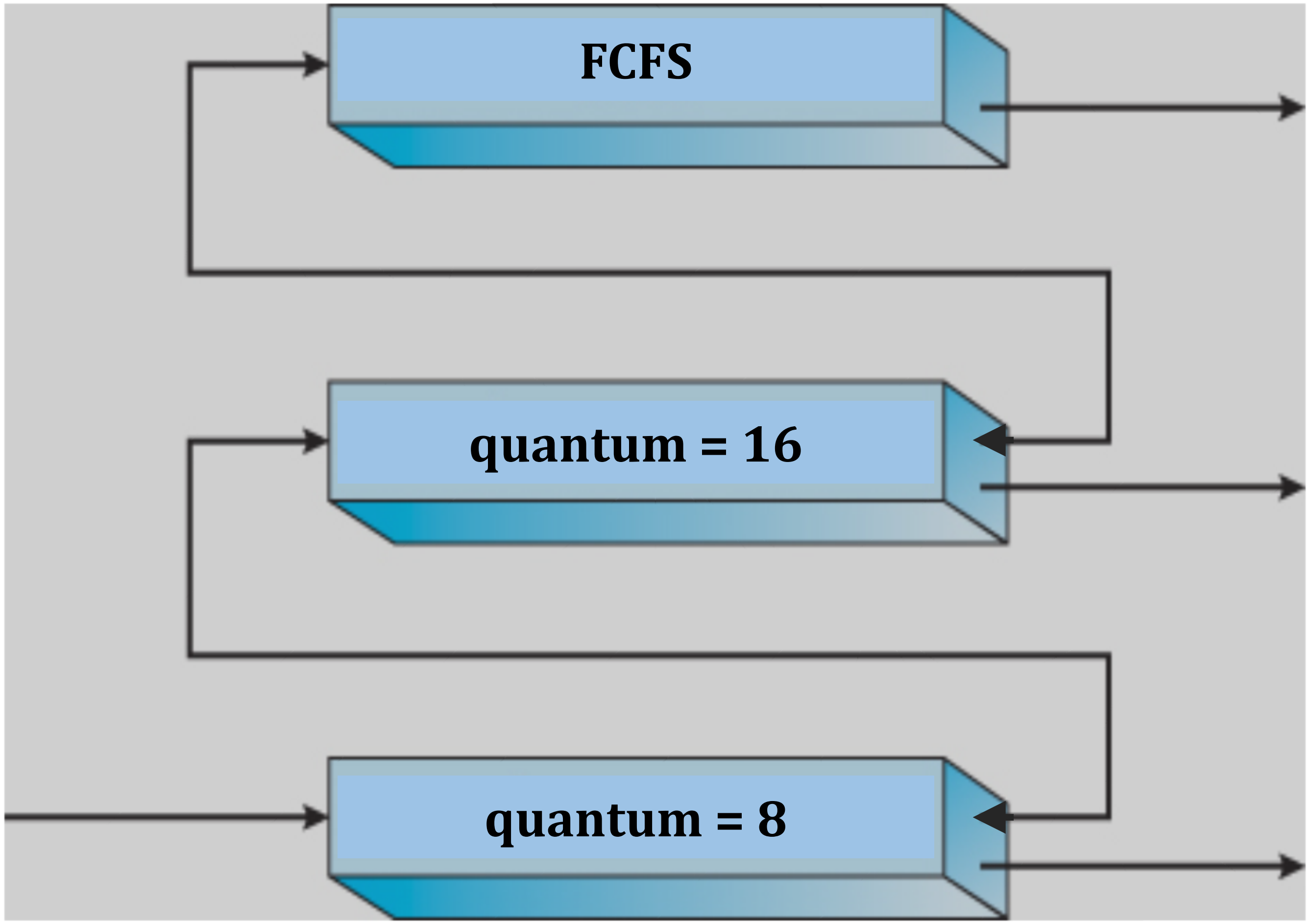
Multilevel Queue Scheduling

- Scheduling must be done between the queues.
  - Priority scheduling; But possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes;

Multilevel Feedback Queue Scheduling

- 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 initially

Multilevel Feedback Queue Scheduling



Multi Processor Scheduling

- Separate queue for each processor
- Common ready queue

Multi Processor Scheduling:
Separate queues

- 4 Processors: C1, C2, C3, C4.
  - P1 goes to C1
  - P2 goes to C2
  - P3 goes to C3
  - P4 goes to C4
  - P5 goes to C1
  - P6 goes to C2 …
- Load balancing w.r.t size of the processes???
  - C1 may be overloaded, while C3 is idle

Multi Processor Scheduling:
Common ready queue

- **Symmetric Approach:**
  - Each processor examines the common ready queue and selects a process to execute.
  - Several instances of STS will be running on different processors, whenever necessary
  - Mutual exclusive access to the common queue???
- **Asymmetric Approach**
  - One processor acts as scheduler (Master) for other processor (Slaves)