**Tishk International University**
**Science Faculty**
**IT Department**


# Operating Systems

## Lecture 3: CPU Scheduling

### 3rd Grade - Fall Semester

**Instructor: Alaa Ghazi**

# Lecture 3: CPU Scheduling

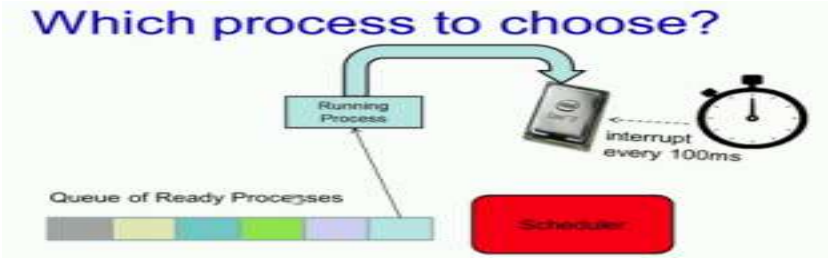
3.1 Basic Concepts

3.2 Scheduling Criteria

3.3 CPU Scheduling Algorithms

1. First-Come First-Serve Scheduling, FCFS
2. Shortest-Job-First Scheduling, SJF
3. Priority Scheduling
4. Round Robin Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback-Queue Scheduling

3.4 Multiple-Processor Scheduling



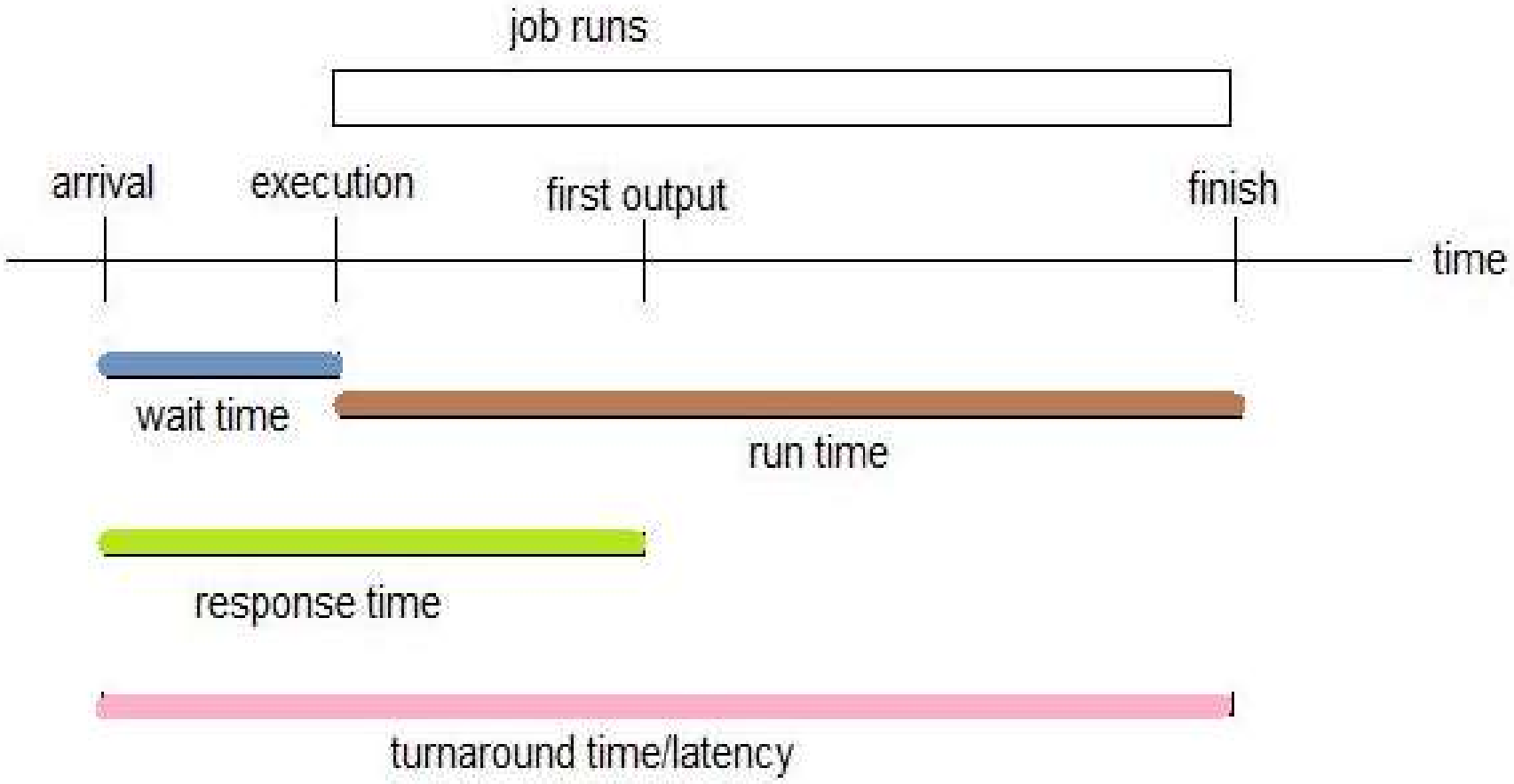
# 3.1 Basic Concepts

- CPU scheduling allows one process to use the CPU while the execution of other processes are on hold.
- Each process will pass into **cycles** of CPU execution and I/O wait, **CPU burst** followed by **I/O burst** and so on.
- CPU burst time is of main concern to the CPU scheduling.
- The important role of an OS is the act of managing and scheduling these activities to maximize the use of the resources and minimize wait and idle time.
- 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

# 3.2 Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – The number of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **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 Criteria – in Time Axis



# Scheduling Algorithm Optimization Criteria

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

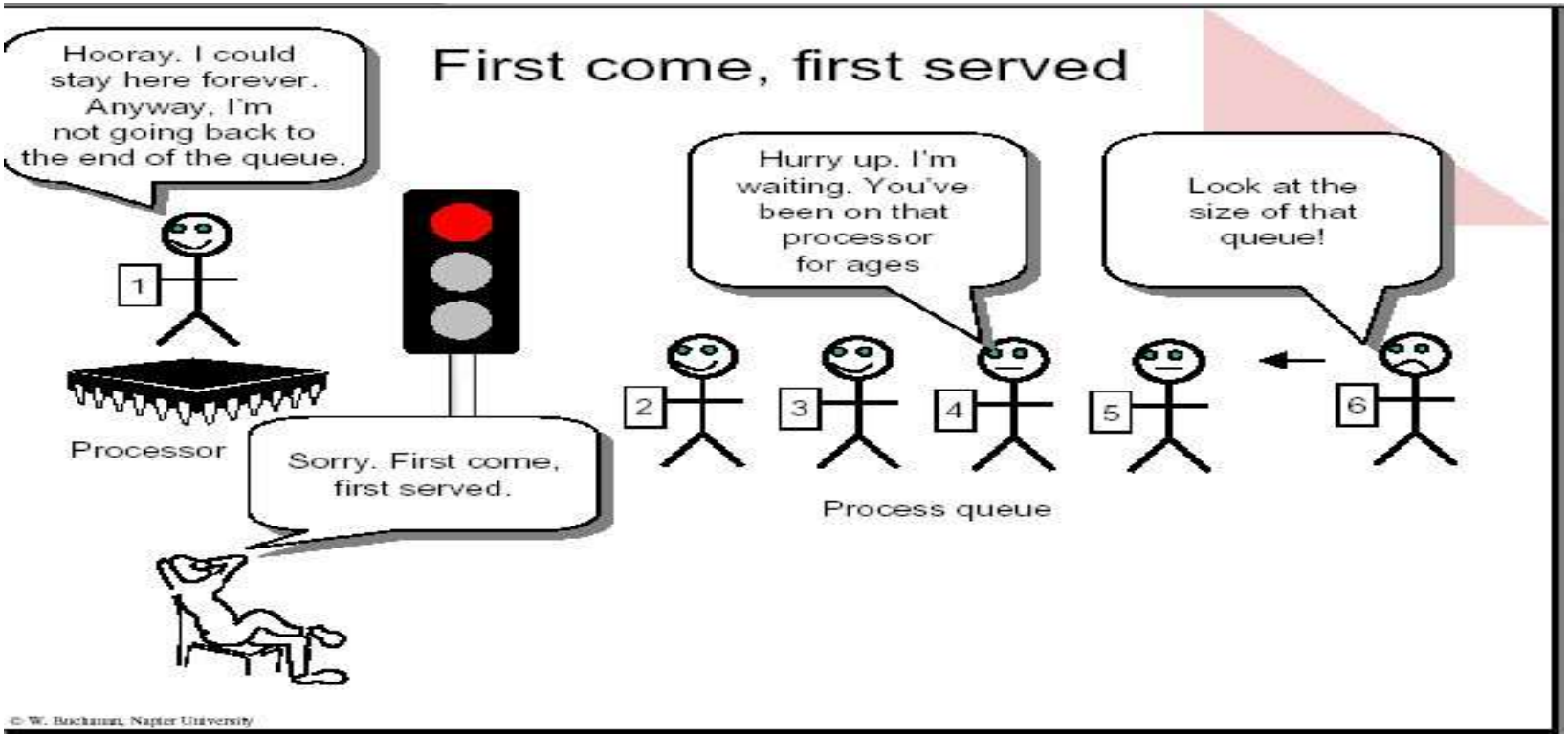
# 3.3 CPU Scheduling Algorithms

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. Algorithms user are:

1. First-Come First-Serve Scheduling, FCFS
2. Shortest-Job-First Scheduling, SJF
3. Priority Scheduling
4. Round Robin Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback-Queue Scheduling

# First- Come, First-Served (FCFS) Scheduling

- FCFS is very simple - like customers waiting in line at the bank or the post office.
- However, FCFS can yield some very long average wait times, particularly if the first process to get there takes a long time. For example, consider the following Example



# Example of FCFS

| 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

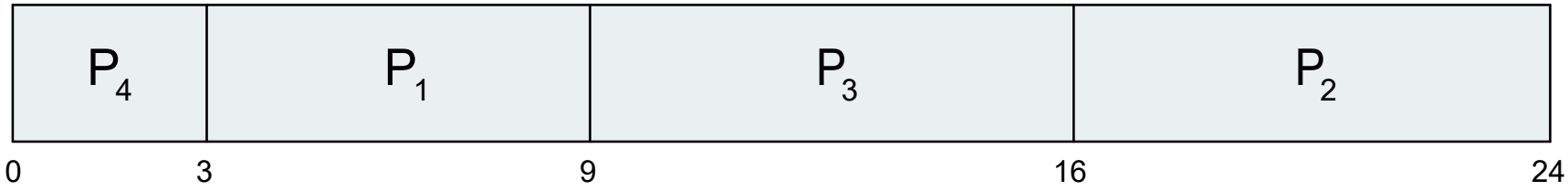
- Average waiting time: (0 + 24 + 27)/3 = 17

# Shortest-Job-First (SJF) Scheduling

- The idea behind the SJF algorithm is to pick the fastest little job that needs to be done, get it out of the way first, and then pick the next smallest fastest job to do next.
- Technically this algorithm picks a process based on the next shortest **CPU burst**, not the overall process 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

# Example of SJF

| Process | Burst Time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 8 |
| $P_3$ | 7 |
| $P_4$ | 3 |




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

# 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)
- Priorities can be assigned either internally or externally. Internal priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, and other factors available to the kernel. External priorities are assigned by users, based on the importance of the job.
- Priority scheduling can suffer from a major problem known as ***indefinite blocking***, or ***starvation***, in which a low-priority task can wait forever because there are always some other jobs around that have higher priority.
- 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

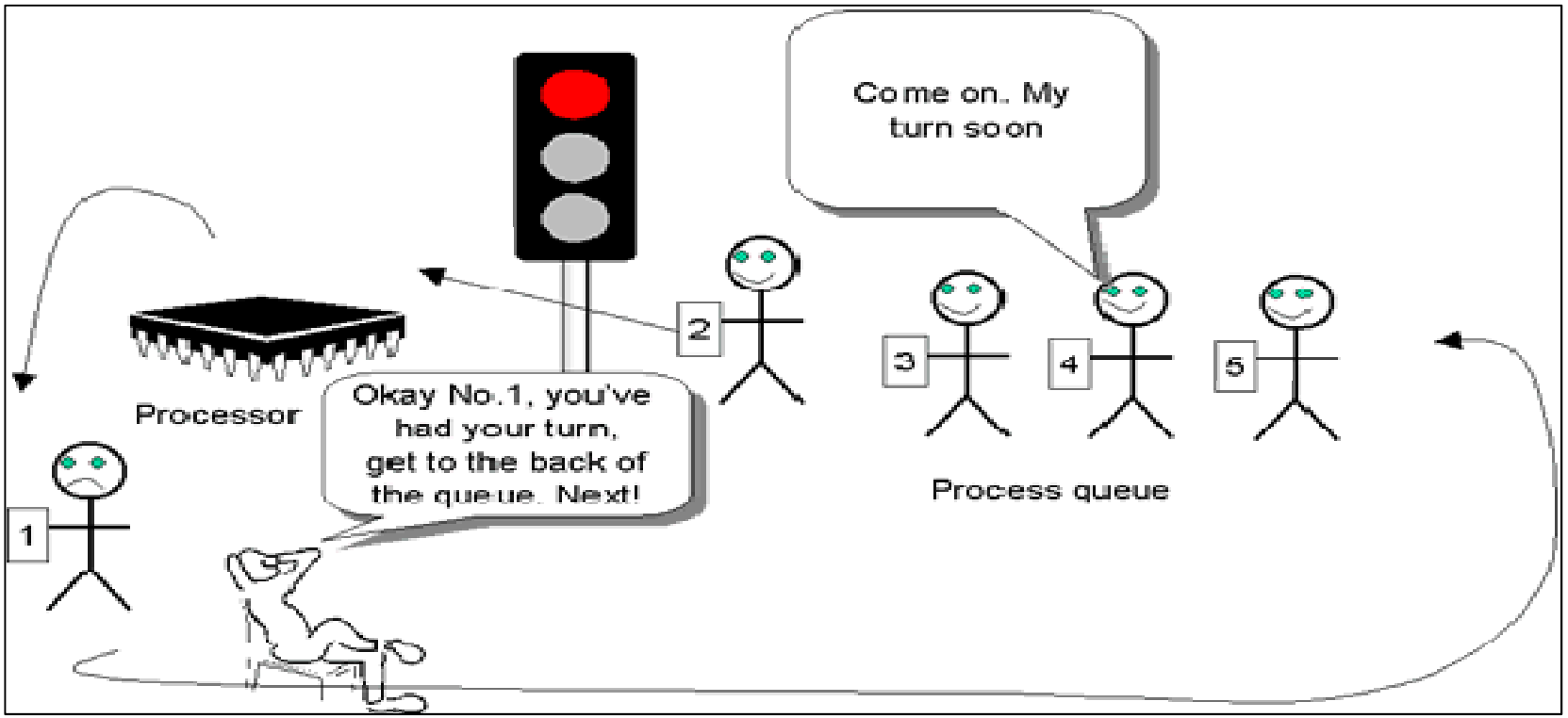

- Average waiting time = (0 + 1 + 6+ 16+ 18) /5 = 8.2 m sec

# Round Robin (RR)

- **Round robin scheduling** is similar to FCFS scheduling, except that CPU bursts are assigned with limits called ***time quantum***.
- When a process is given the CPU, a timer is set for a time quantum.
  - If the process finishes its burst before the time quantum timer expires, then it is swapped out of the CPU just like the normal FCFS algorithm.
  - If the timer goes off first, then the process is swapped out of the CPU and moved to the back end of the ready queue.

# Round Robin (RR)

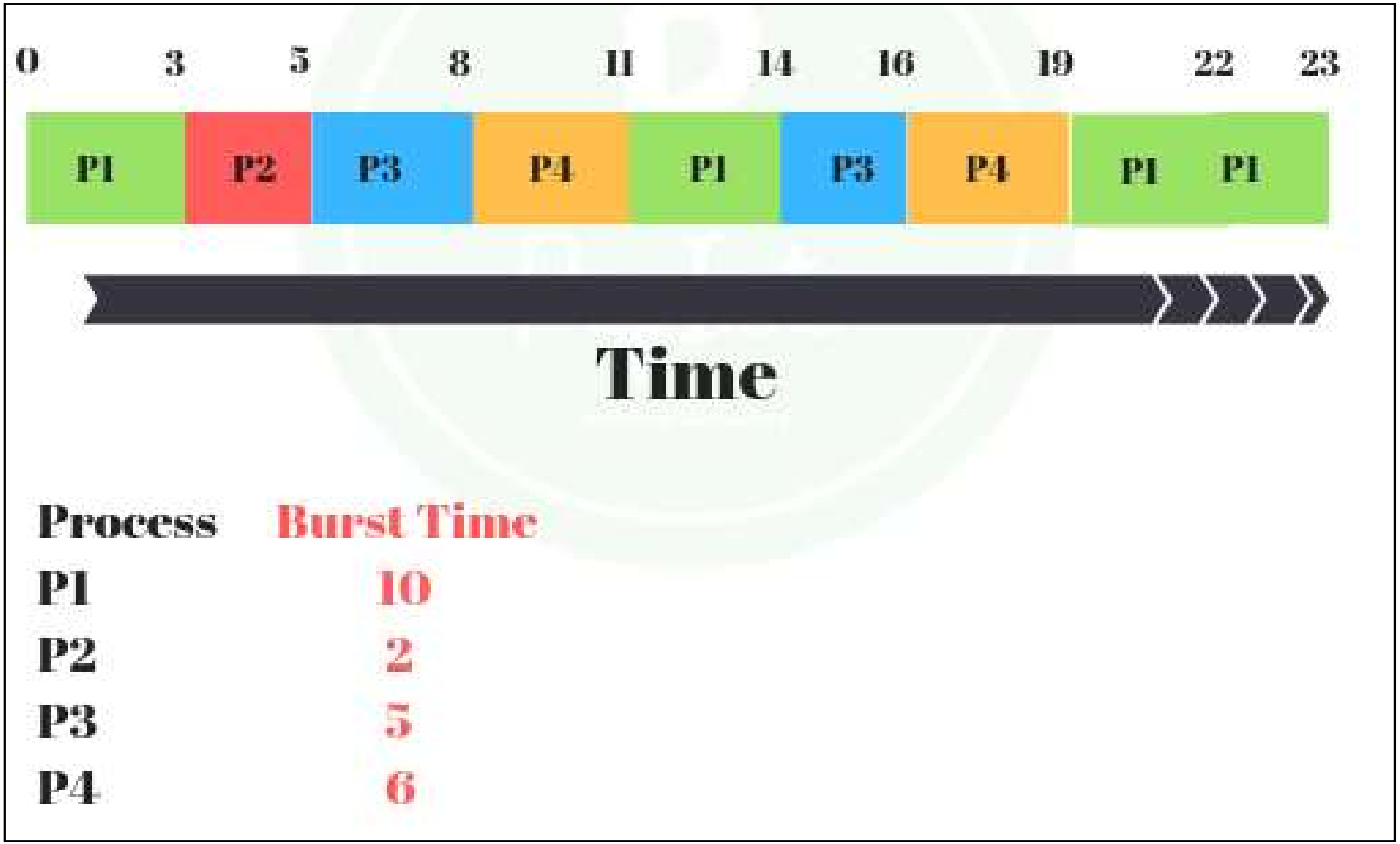
- The ready queue is maintained as a circular queue, so when all processes have had a turn, then the scheduler gives the first process another turn, and so on.
- RR scheduling can give the effect of all processes sharing the CPU equally.



# Advantages and Drawbacks of Round Robin (RR)

- The advantages of the round robin scheduling algorithm are:
  - The algorithm is fair because each process gets a fair chance to run on the CPU.
  - Better ***response than SJF***
- The drawbacks of the round robin scheduling algorithm are as follows;
  - there is an increase number of context switching that occurs with considerable over heads.
  - Typically, higher average turnaround than SJF.

# Example of RR with Time Quantum = 3 (not required in the exam)




| Process | Burst Time |
|---|---|
| P1 | 10 |
| P2 | 2 |
| P3 | 5 |
| P4 | 6 |

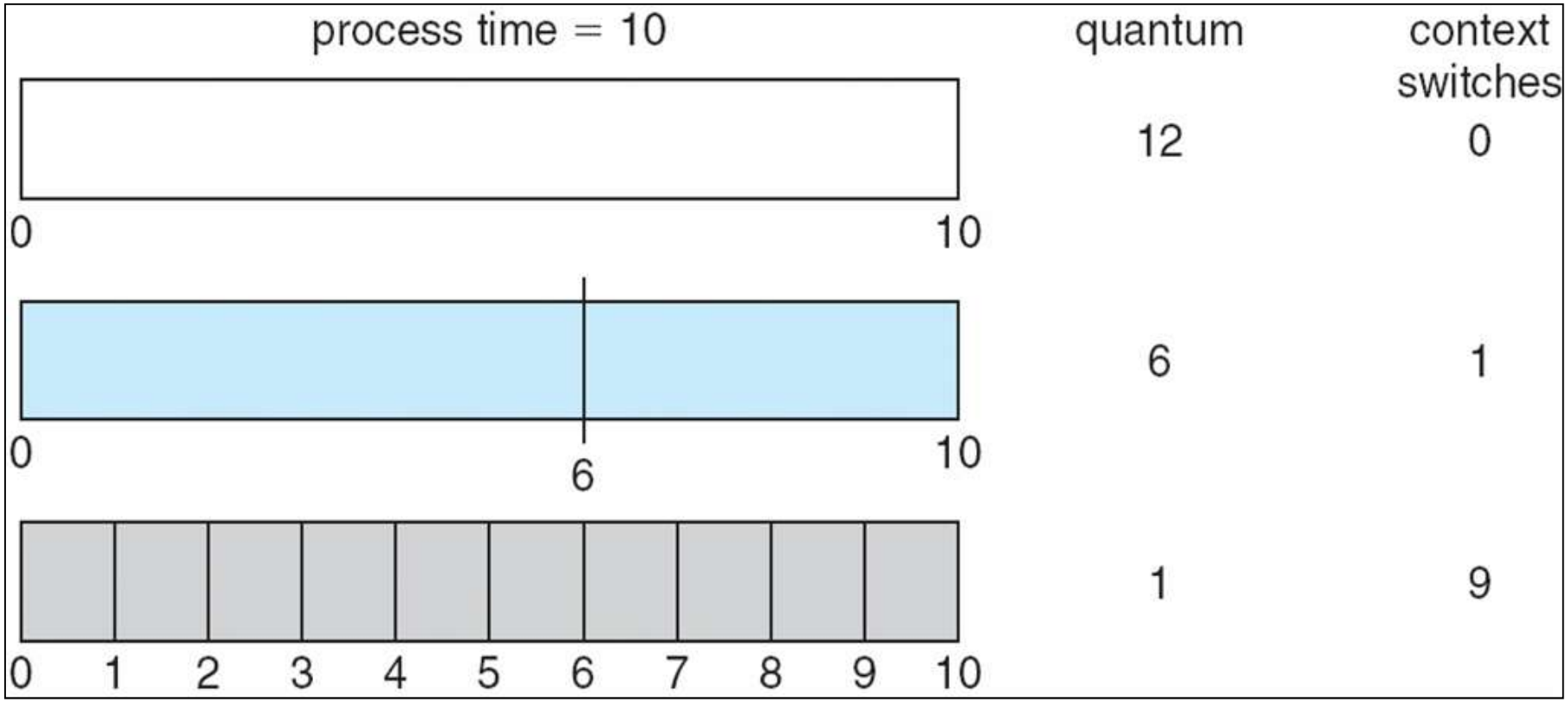
# RR Time Quantum Value

- A short quantum:
  - Good: because processes need not wait long before they are scheduled in.
  - Bad: because context switch overhead increase
- A long quantum:
  - Bad: because processes no longer appear to execute concurrently and that may degrade the system performance (same like FCFS)

- Time Quantum q should be large compared to context switch time but not larger than average process time

- q should be usually 10ms to 100ms, when context switch < 10 micro s

# Time Quantum and Context Switch Time Comparison Diagram (not required in the exam)



# Multilevel Queue

- When processes can be readily categorized, then multiple separate queues can be established, each implementing whatever scheduling algorithm is most appropriate for that type of job, and/or with different parametric adjustments.
- Note that under this algorithm jobs cannot switch from queue to queue - Once they are assigned a queue, that is their queue until they finish.
- Each queue has its own scheduling algorithm for example:
  - foreground – RR
  - background – FCFS

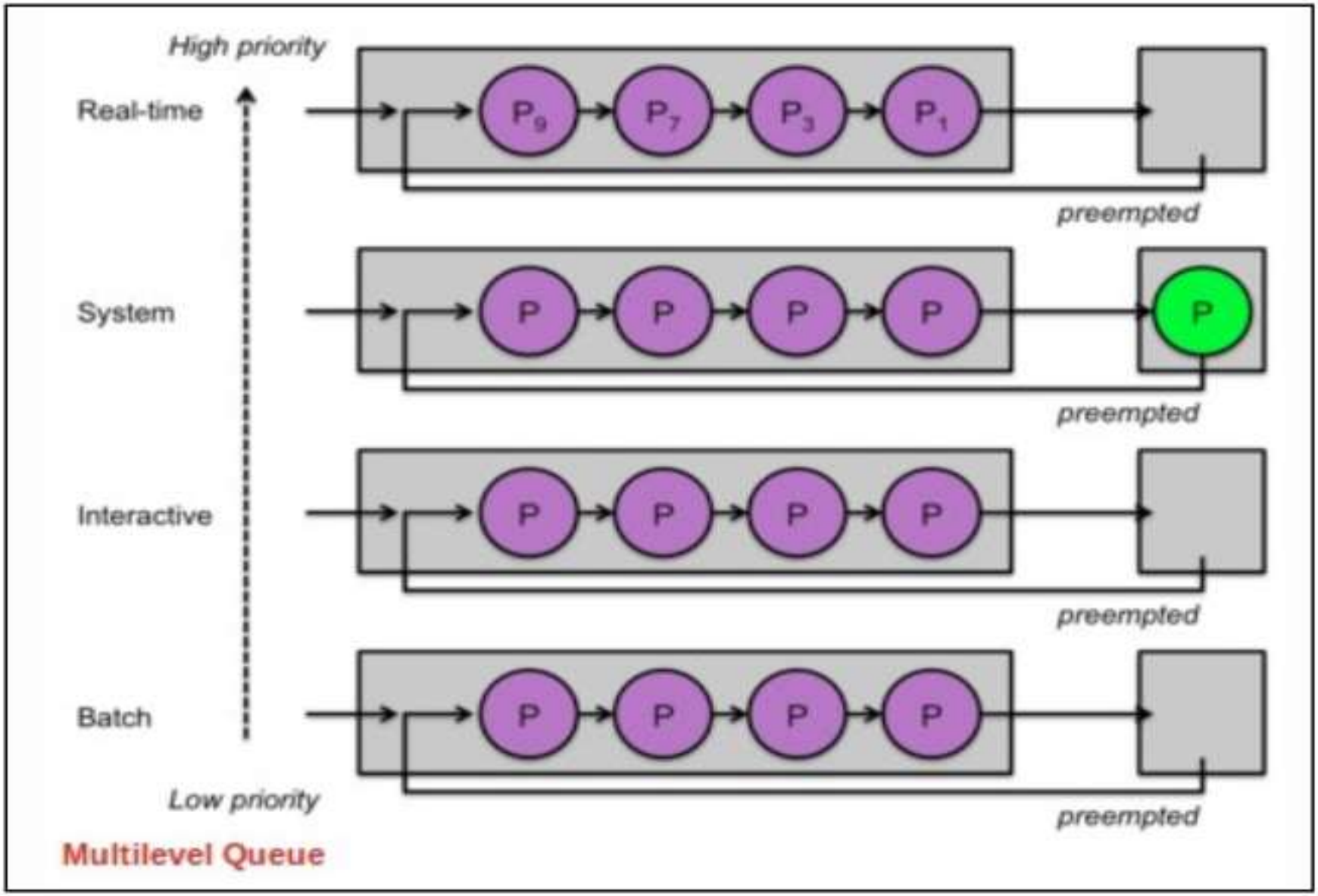
# Multilevel Queue

- 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



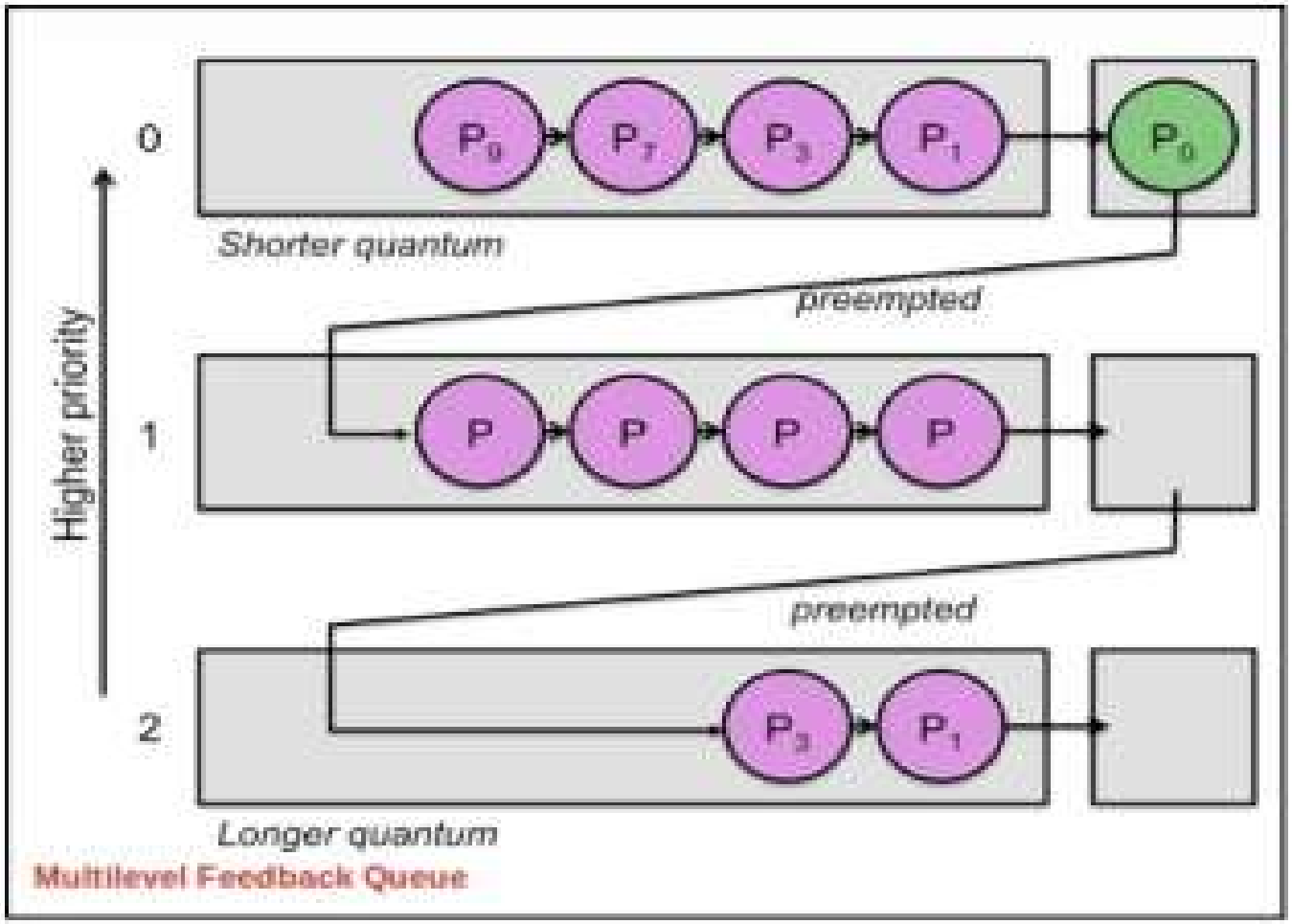
# Multilevel Feedback Queue

- Multilevel feedback queue scheduling is similar to the ordinary multilevel queue scheduling described above, but the **difference** is **jobs may be moved from one queue to another** for a variety of reasons:
  - If the characteristics of a job change between CPU-intensive and I/O intensive, then it switches a job from one queue to another.
  - Aging: a job that has waited for a long time can get bumped up into a higher priority queue.

# Multilevel Feedback Queue

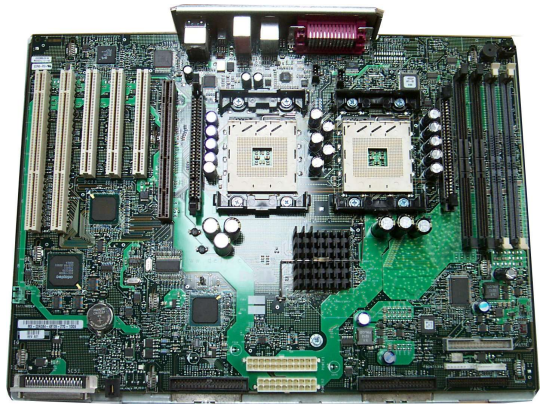
- Multilevel feedback queue scheduling is the most flexible, because it can be tuned for any situation. But it is also the most complex to implement because of all the adjustable parameters. Some of the **parameters** which define one of these systems include:
  - The number of queues.
  - The scheduling algorithm for each queue.
  - The methods used to transfer processes from one queue to another.
  - The method used to determine which queue a process enters initially.

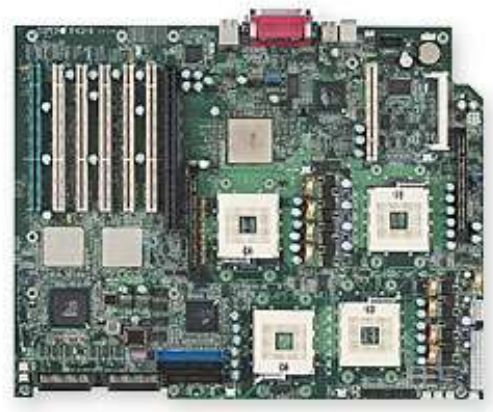
# Example of Multilevel Feedback Queue (not required in the exam)



# 3.4 Multiple-Processor Scheduling

- When multiple processors are available, then the scheduling gets more complicated, because now there is more than one CPU which must be kept busy and in effective use at all times.
- ***Load sharing*** revolves around balancing the load between multiple processors.
- Multi-processor systems may be **heterogeneous,** ( different kinds of CPUs ), or **homogenous,** ( all the same kind of CPU ).





# Approaches to Multiple-CPU Scheduling

- **Asymmetric multiprocessing** – here a single scheduler is running only on one CPU accesses the system data structures, and decides for every CPU.
- **Symmetric multiprocessing (SMP)** – the most common approach, when each CPU is self-scheduling, with two versions:
  - **Global Queue**: all processes in common ready queue, or
  - **Per CPU Queues**: each has its own private queue of ready processes

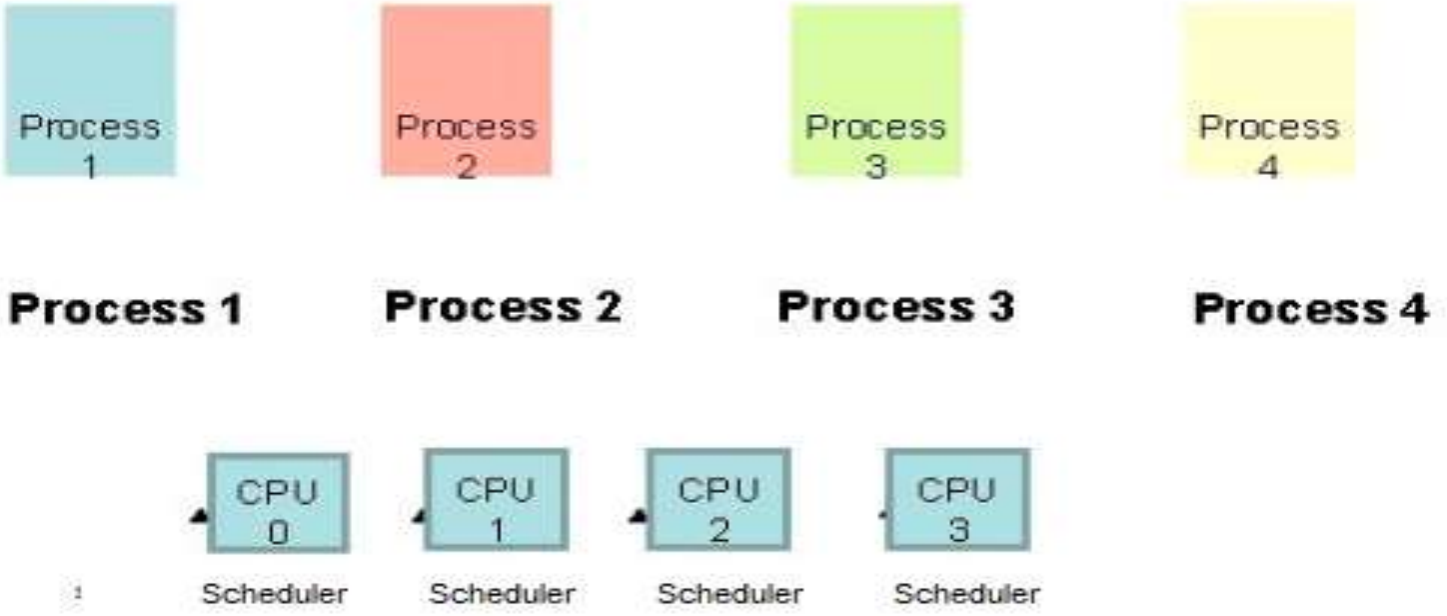
# MultiprocessorScheduling with a single scheduler


**Process 1** **Process 2** **Process 3** **Process 4**


**Single Scheduler**

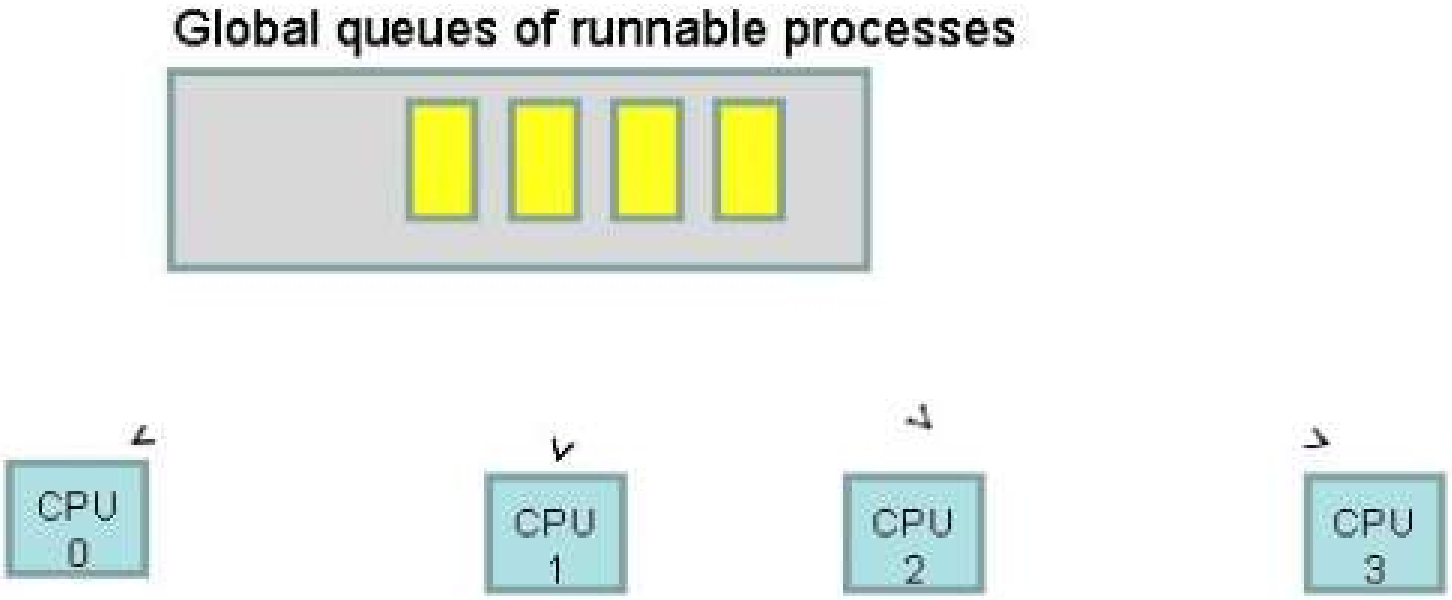
One processor decides for everyone

# MultiprocessorScheduling (Symmetical Scheduling)



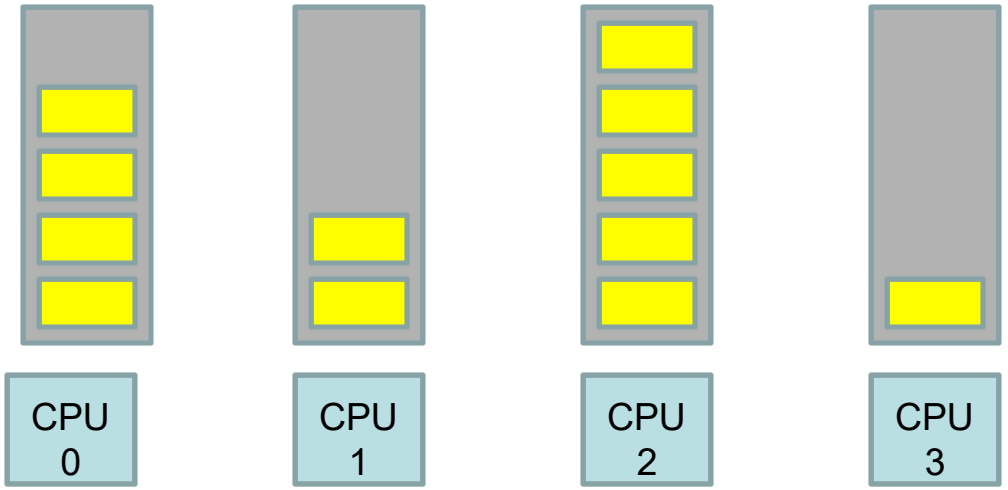

Each processor runs a scheduler independently to select the process to execute

# Symmetrical Scheduling (with global queues)



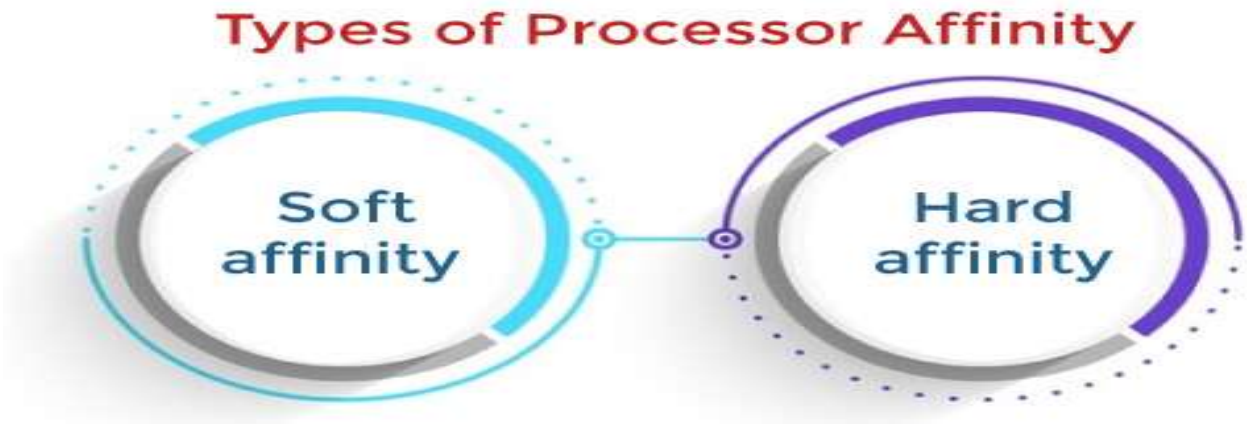
# Symmetrical Scheduling (with per CPU queues)

- Static partition of processes across CPUs



# Processor Affinity

- **Processor affinity** – the binding of a process or a thread to a **CPU**, so that the **process** or thread will execute only on the designated **CPU. There are two types:**
  - **soft affinity:** when the system attempts to keep processes on the same processor but makes no guarantees
  - **hard affinity:** in which a process specifies that it is not to be moved between processors.



# Multiple-Processor Scheduling – Load Balancing

- **Load balancing** attempts to keep workload evenly distributed, so that one CPU won't be sitting idle while another is overloaded. This is done using Process Migration.
- **Load Balancing Techniques:**
- **Push Migration** A special task periodically monitors load of all processors, and redistributes work when it finds an imbalance.
- **Pull Migration** Idle processors pull a waiting task from a busy processor