

Chapter 2: CPU Scheduling

Chapter 6: CPU Scheduling

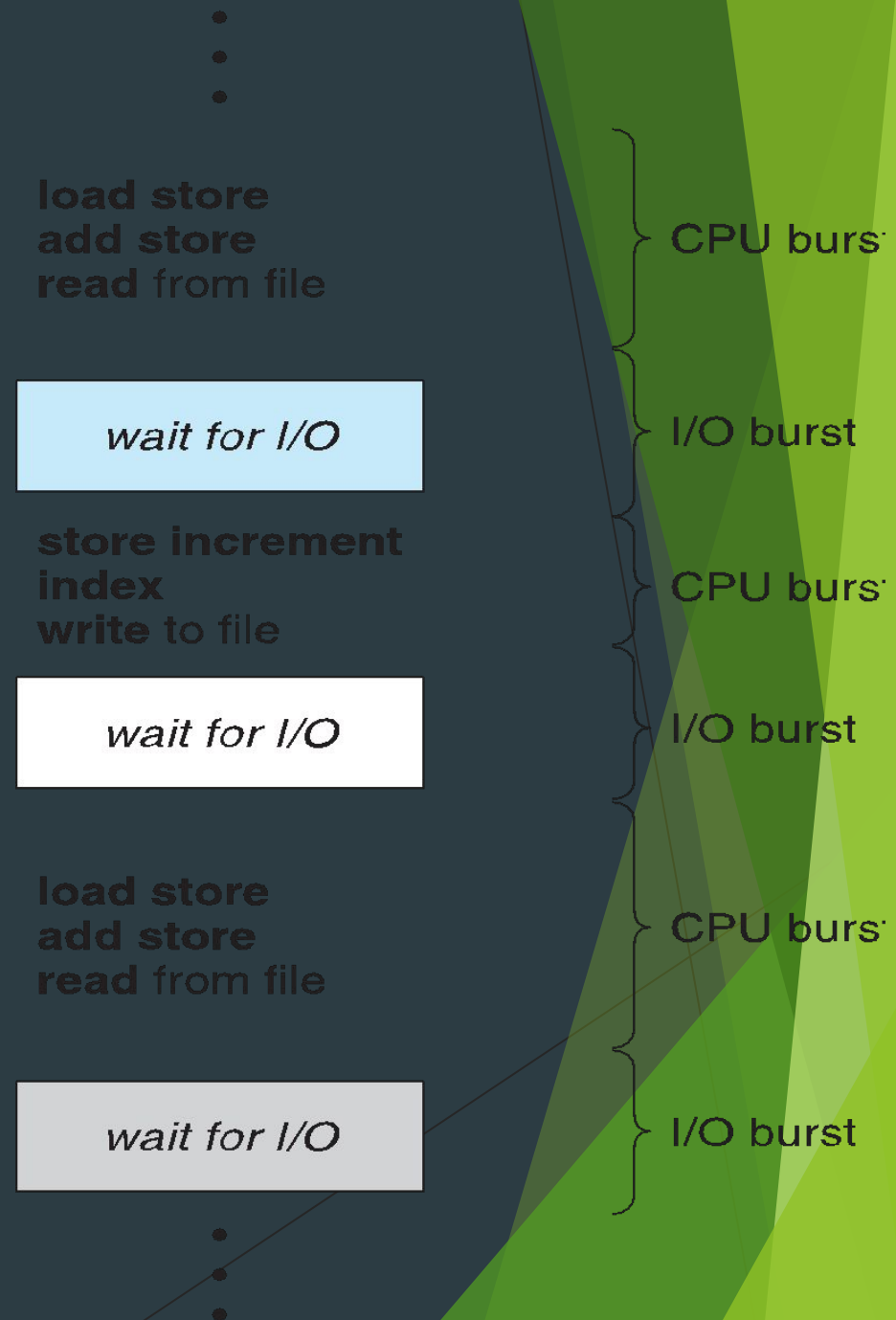
- ▶ Basic Concepts
- ▶ Scheduling Criteria
- ▶ Scheduling Algorithms

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 Scheduler

- **Short-term scheduler** selects from among the processes in the 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 **non preemptive**
- All other scheduling is **preemptive**
 - Consider access to shared data
 - Consider preemption while in kernel mode
 - Consider interrupts occurring during crucial OS activities

Dispatcher

- ▶ 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** - the time it takes for the dispatcher to stop one process and start another running

Scheduling Criteria

1. *CPU utilization* - keep the CPU as busy as possible
2. *Throughput* - no. of processes that complete their execution per time unit
3. *Turnaround time* - the amount of time to execute a particular process
4. *Waiting time* - the amount of time a process has been waiting in the ready queue
5. *Response time* - the amount of time it takes from when a request was submitted until the first response is produced, not output (for a time-sharing environment)

Scheduling Algorithm Optimization Criteria

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

First- Come, First-Served (FCFS) Scheduling

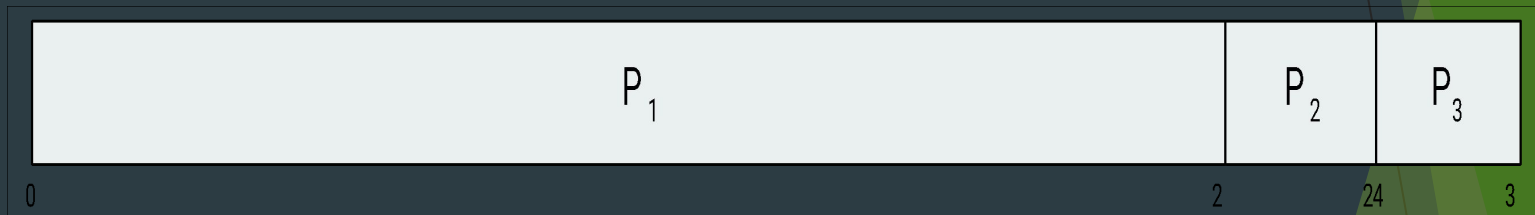
<u>Process</u>	<u>Burst Time</u>
----------------	-------------------

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:



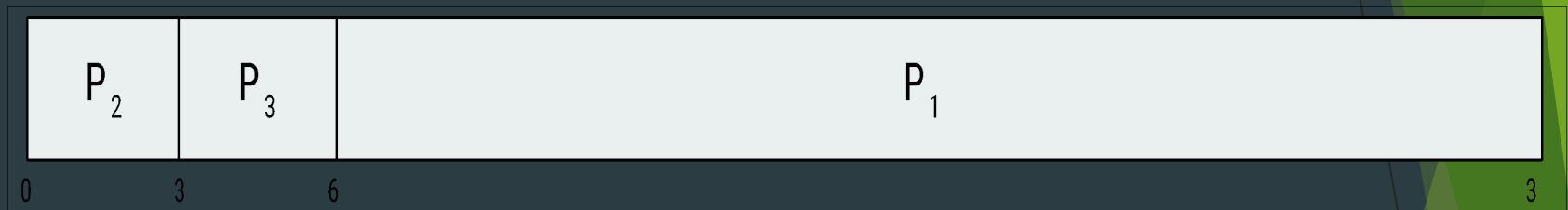
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

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

Characteristics of FCFS method

- ▶ It supports non-preemptive and pre-emptive scheduling algorithms.
- ▶ Jobs are always executed on a first-come, first-serve basis.
- ▶ It is easy to implement and use.
- ▶ This method is poor in performance, and the general wait time is quite high.

Advantages of FCFS

- The simplest form of a CPU scheduling algorithm
- Easy to program
- First come first served

Disadvantages of FCFS

- It is a Non-Preemptive CPU scheduling algorithm, so after the process has been allocated to the CPU, it will never release the CPU until it finishes executing.
- The Average Waiting Time is high.
- Short processes that are at the back of the queue have to wait for the long process at the front to finish.
- Not an ideal technique for time-sharing systems.
- Because of its simplicity, FCFS is not very efficient.

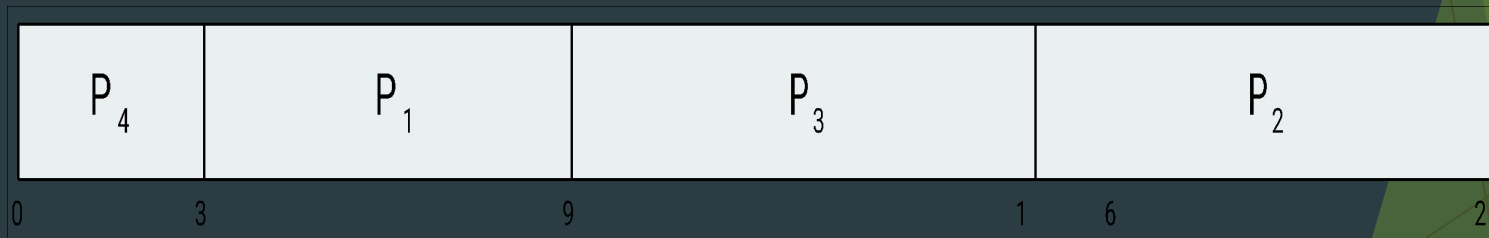
Shortest-Job-First (SJF) Scheduling

- ▶ Associate with each process the length of its next CPU burst
 - ▶ Use these lengths to schedule the process in 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

	<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1		0.0	6
P_2		2.0	8
P_3		4.0	7
P_4		5.0	3

- ▶ SJF scheduling chart



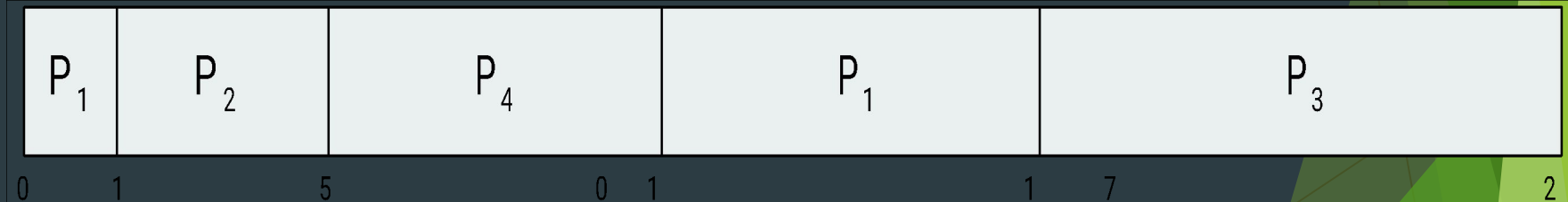
- ▶ Average waiting time = $(3 + 16 + 9 + 0) / 4 = 7$

Example of Shortest-remaining-time-first(Preemptive)

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

	<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
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

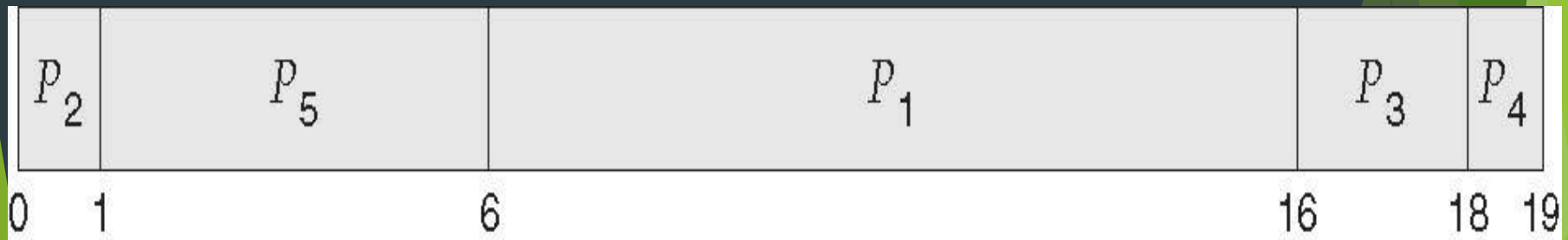
Priority Scheduling

- ▶ A priority number (integer) is associated with each process
- ▶ The CPU is allocated to the process with the highest priority (smallest integer \equiv highest priority)
 - ▶ Preemptive
 - ▶ Nonpreemptive
- ▶ SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- ▶ Problem \equiv **Starvation** - low priority processes may never execute
- ▶ Solution \equiv **Aging** - as time progresses increase the priority of the process

Example of Priority Scheduling

	<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
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 = 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 \Rightarrow FIFO
 - ▶ q small $\Rightarrow q$ must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time slice = 4

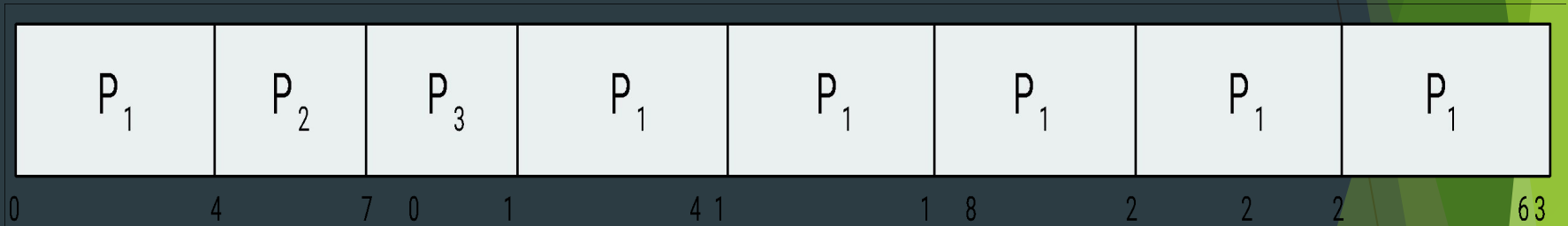
Process Burst Time

P_1 24

P_2 3

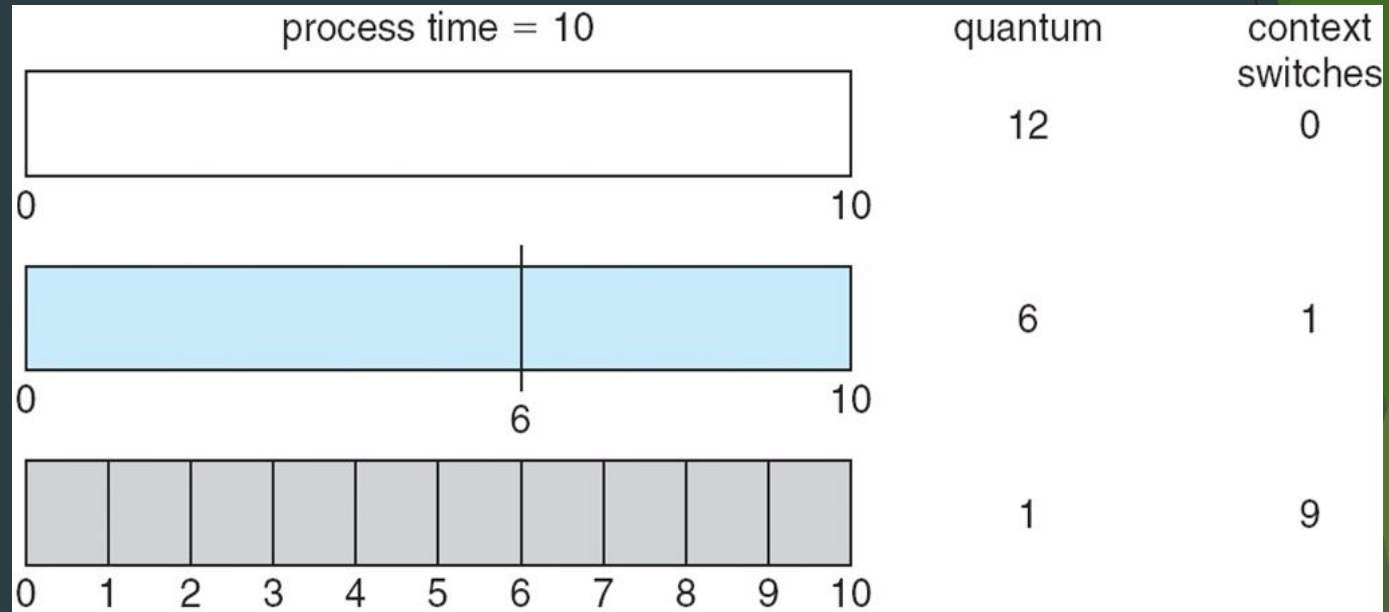
P_3 3

- ▶ 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

Time Quantum and Context Switch Time



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

