Computer Systems
Process Scheduling

Lecture Outline

- Basic Concepts of CPU Scheduling
- Scheduling Criteria
- Scheduling Algorithms
  - FCFS – First Come First Serve
  - SJF – Shortest Job First
  - SRTF – Shortest Remaining Time First
  - Priority and Round-Robin Scheduling

Basic Concepts

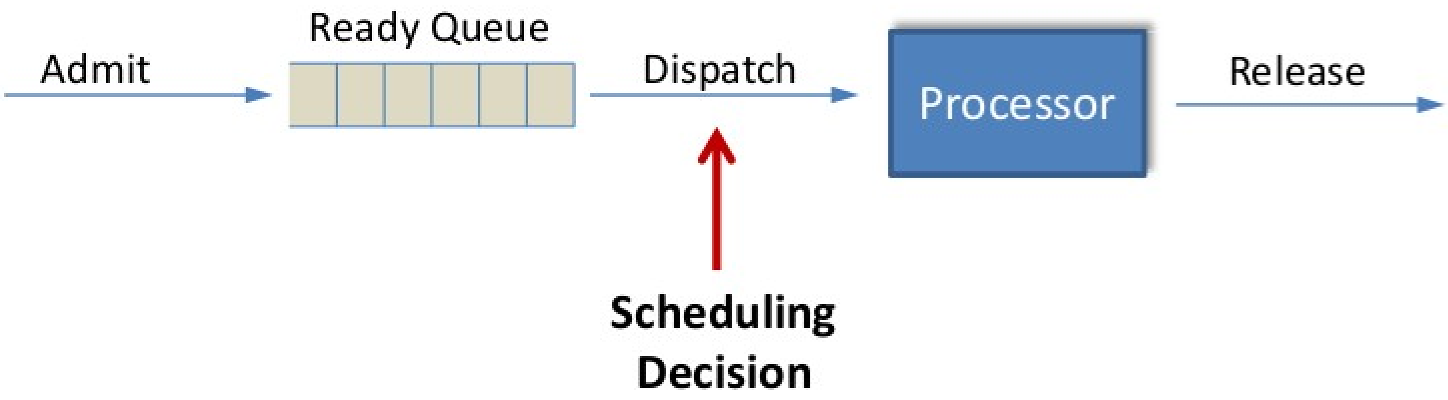
- **Single processor** system = **single process** can run at a time
- Others must wait until the CPU is free and can be rescheduled
- Objective of multi-programming is to have some process running at all times

Every OS has to have a scheduler.

  - **Maximize CPU utilization**
- This is a fundamental OS function

Scheduling

- CPU Scheduling
  - Decides which process to execute next – is the activity of **selecting the next process** in the Ready queue for execution on a processor



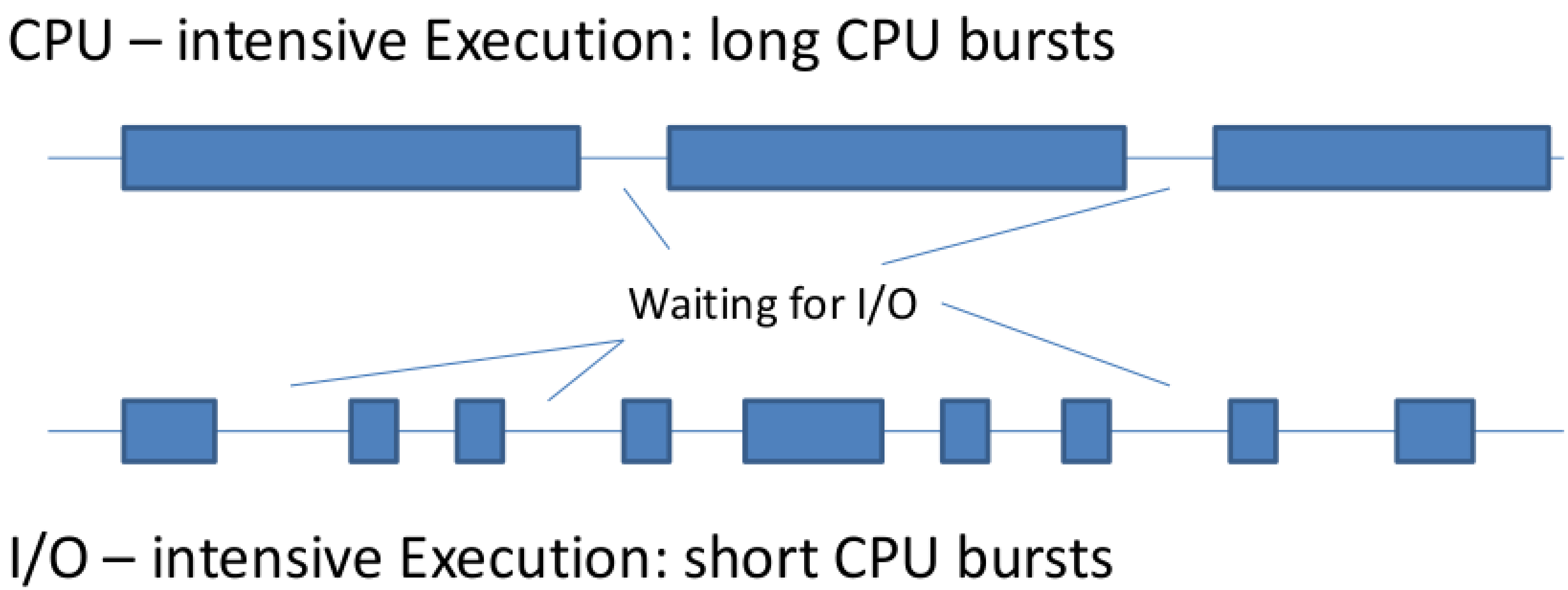
When does Scheduling happen?

- A scheduling decision takes place when an event occurs that interrupts the execution of a process
- Possible events
  - Clock Interrupts (i.e. process **running** → **ready** state)
  - I/O Interrupts (i.e. process **waiting** → **ready** state)
  - Operating System Calls (read, write, process **ready** → **waiting**)
  - Signals (e.g. Semaphores)

A lock

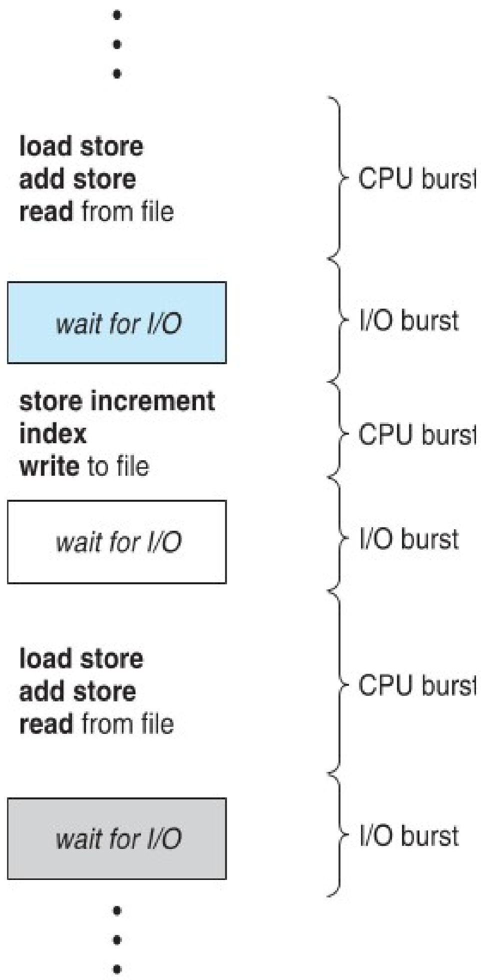
Characteristics of Process Execution

- Batch Processing: **CPU Bound**
- Interactive Systems: **I/O Bound**



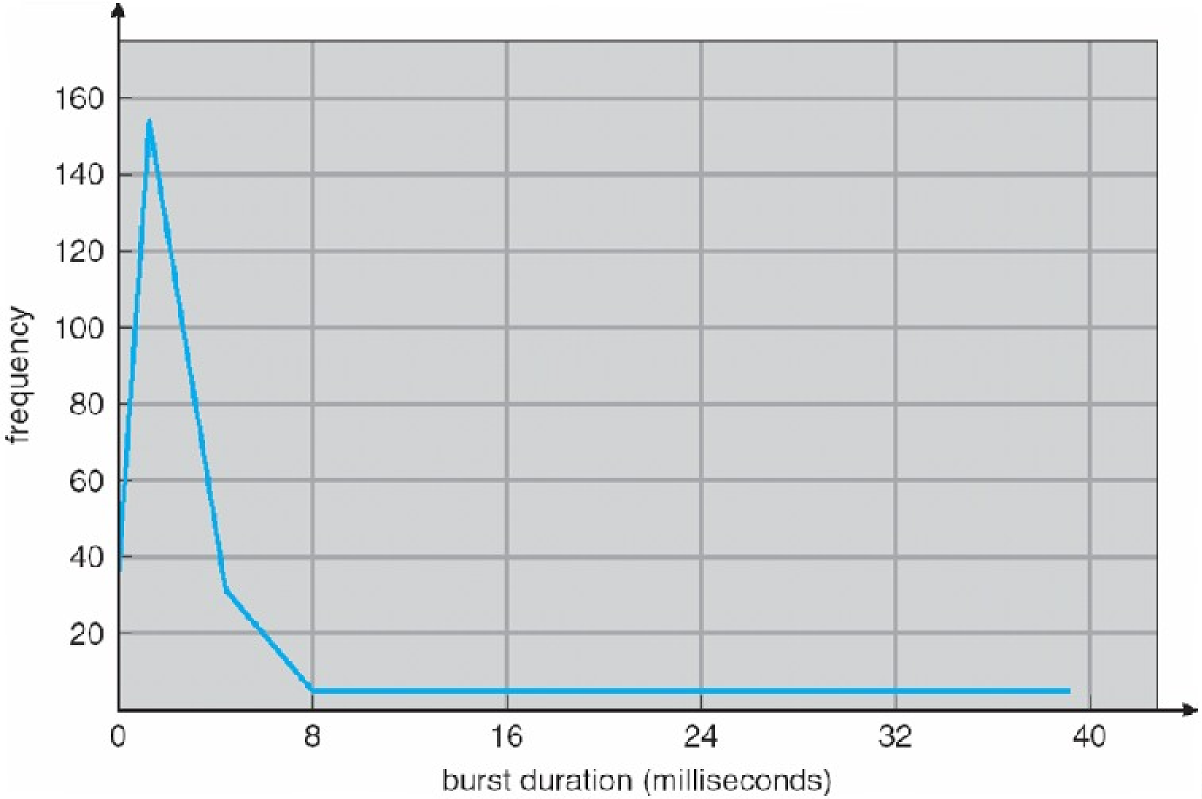
CPU-I/O Burst Cycle

- Process execution consists of **a cycle** of CPU execution and I/O wait
- Processes **alternate** between these two states
- Process execution begins with a **CPU burst,** this is followed by an **I/O burst.** This pattern repeats!
- Scheduler can schedule another process during I/O burst.
- Eventually **final CPU burst** ends with a system request to terminate execution



Histogram of CPU Bursts

- This histogram shows that most of the processes have many very short CPU bursts e.g. Interactive systems



CPU Scheduler

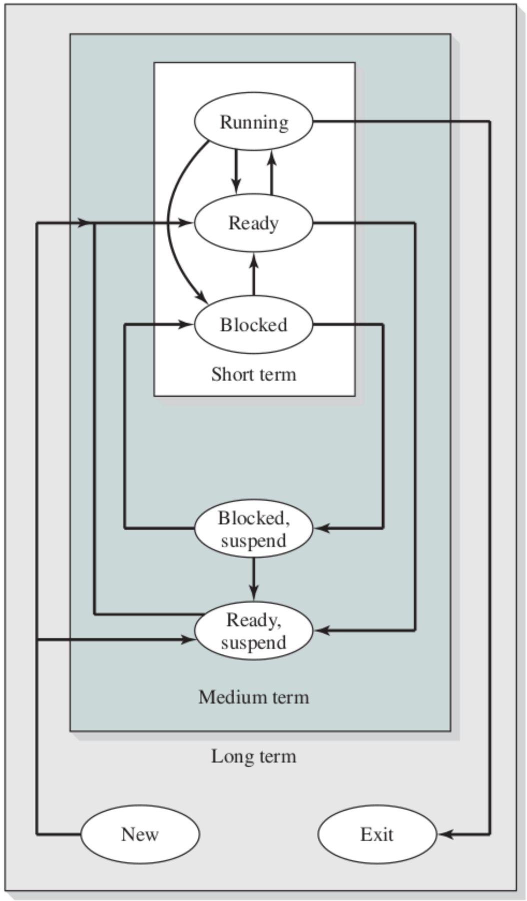
- Recall: whenever the CPU becomes idle, the OS must select one of the processes in the ready queue to be executed next
  - This is carried out by the **short-term scheduler** from the processes in the ready queue
- The queue is not necessarily first-in, first-out (FIFO)
  - It may be a priority queue, tree or an unordered linked list
- Conceptually, all processes are waiting for a chance to run on the CPU

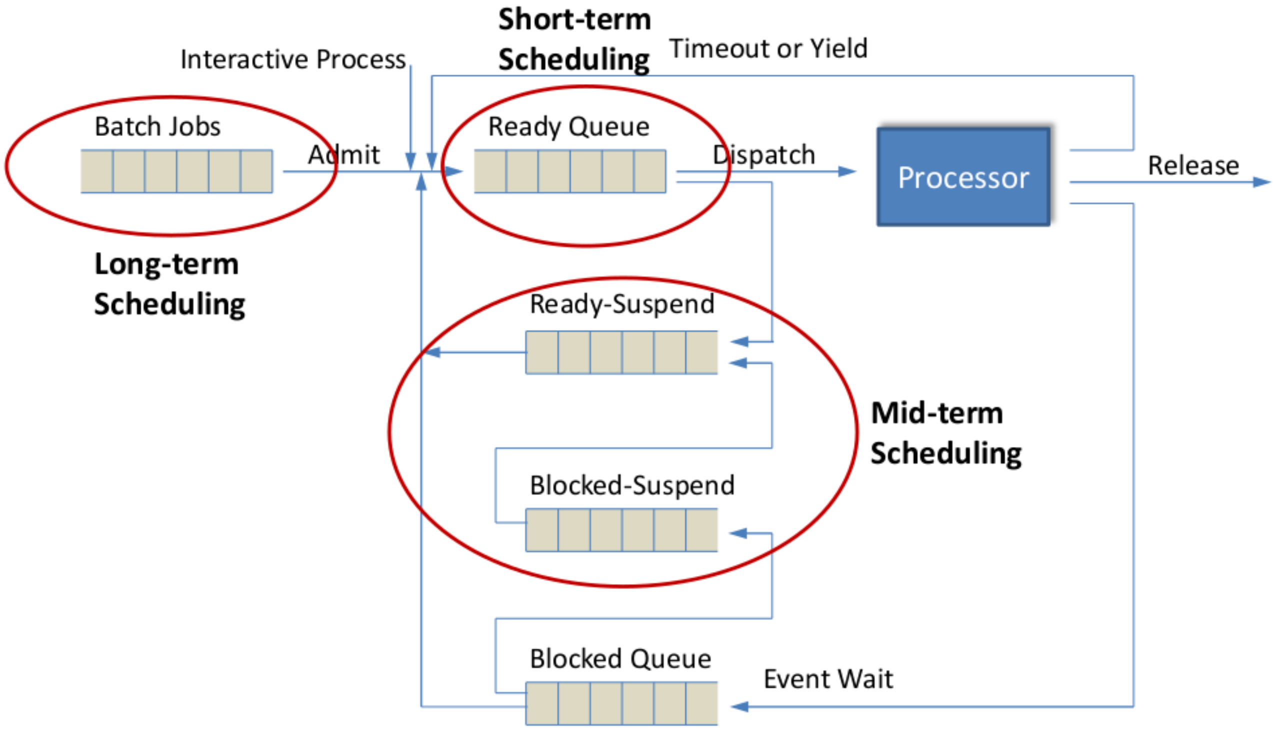
Dispatcher

- The **dispatcher** is the module that 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
- Dispatcher should be as fast as possible
  - It is invoked during every process switch
  - Time taken by dispatcher to stop one process and start another one is known as **dispatch latency.**

调度员
If dispatcher is slow, this very reduce the speed of a computer significantly

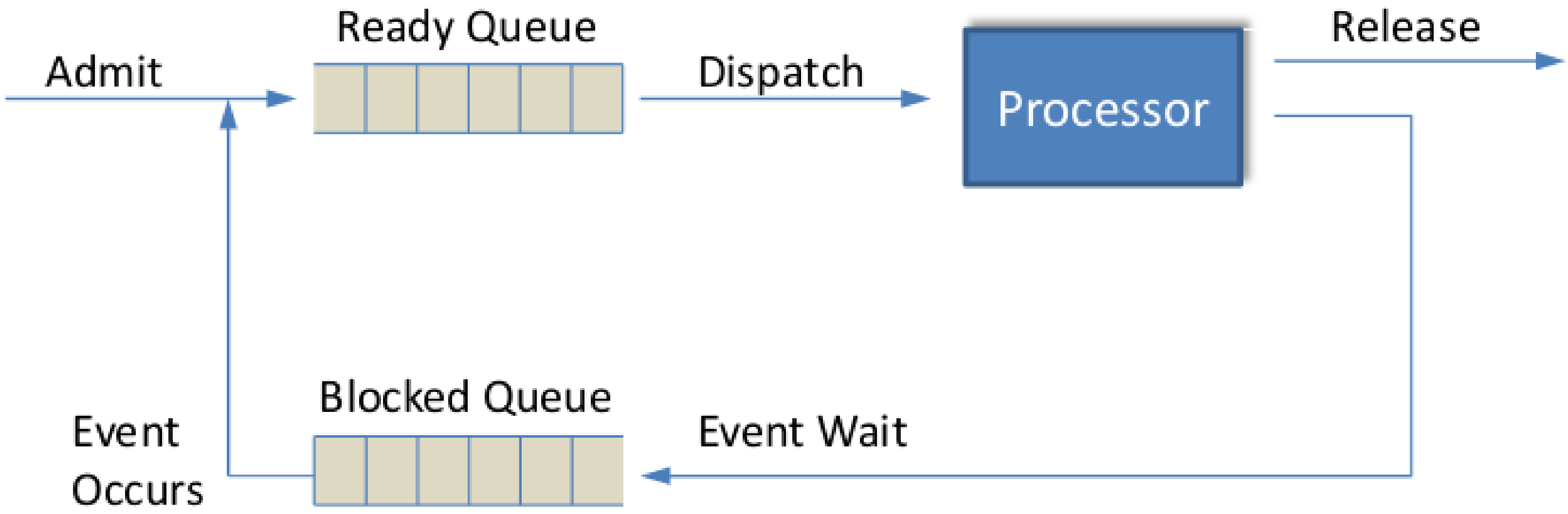
Levels of Scheduling





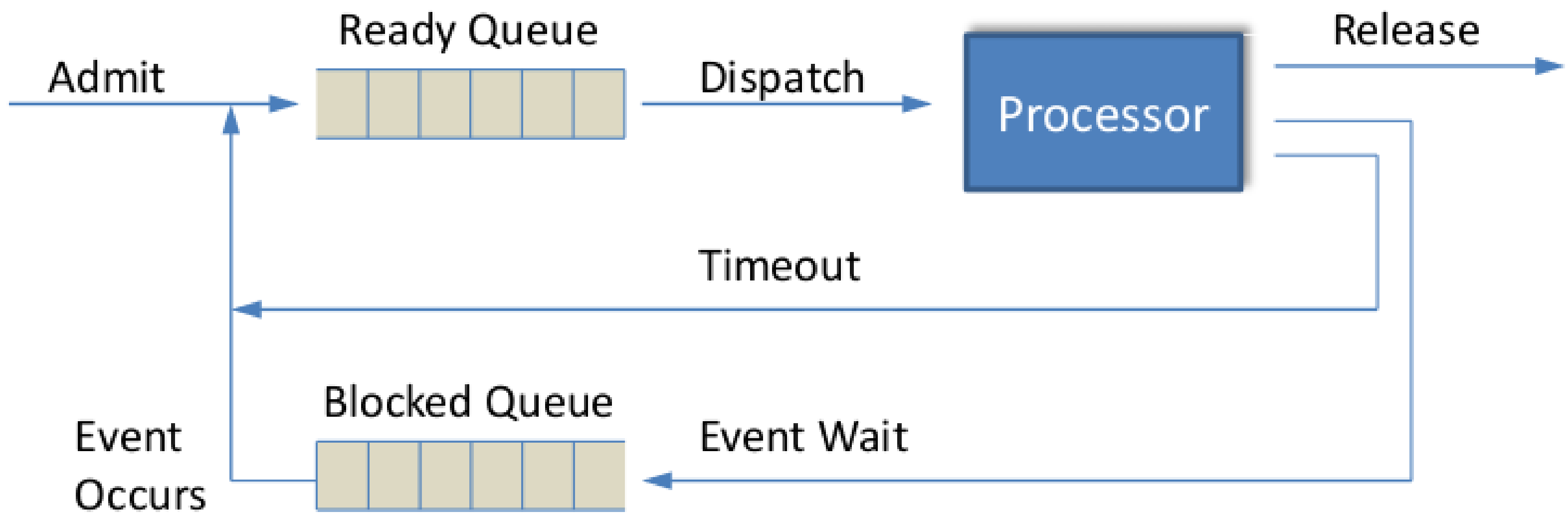
Non-preemptive Scheduling

- Once a process is scheduled, it continues to execute on the CPU, until:
  - it is finished (terminates)
  - it releases the CPU voluntarily (cooperative scheduling)
  - It blocks due to an event such as I/O interrupts, waits for another process etc.



Preemptive Scheduling

- The operating system interrupts (intervenes) processes
  - A scheduled process executes, until its time slice is used up
  - Clock interrupt returns control of CPU to scheduler (time slice expired)
    - Current process is suspended and placed in the Ready queue
    - New process is selected from Ready queue and executed on CPU
  - When a process with higher priority becomes ready



Scheduling Criteria

- Different CPU-scheduling algorithms have different properties. One may favour one class of processes over another.
- Many criteria suggested, including
  - **CPU Utilization** - keep that CPU as busy as possible. 0-100%, but realistically, 40-90%
  - **Throughput** - Number of processes that complete their execution per time unit
  - **Turnaround time** - Total amount of time to execute one process to its completion.
  - **Waiting time** - Total amount of time a process has been waiting in the Ready queue
  - **Response time** - The time it takes from when a request was submitted until the first response is produced by the operating system (latency).
  - Others include meeting deadlines, predictability, fairness, balance and enforcing priorities.

Optimization Criteria

- **Maximize (operating system concerns)**
  - CPU Utilization
  - Throughput
- **Minimize (user concerns)**
  - Turnaround time
  - Waiting time
  - Response time

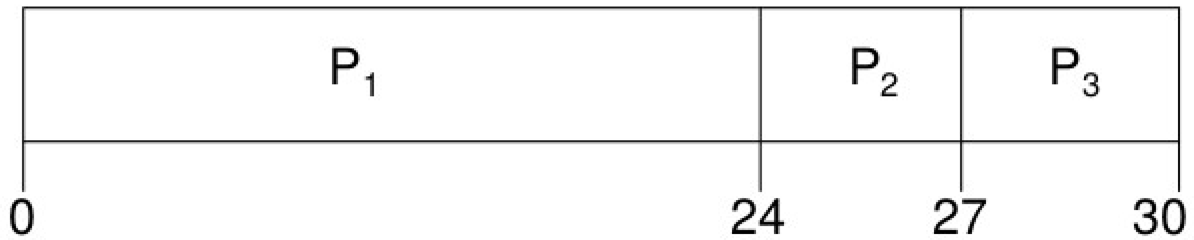
First Come, First Served Scheduling (FCFS)

- Simplest CPU-scheduling algorithm
- The process that requests the CPU first is allocated the CPU first
- Managed with a FIFO queue, process enters the ready queue
- PCB linked onto tail of queue
- When CPU free, it is allocated to the process at the head of the queue
- Running process then removed from queue

FCFS Example

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose the processes arrive in the order: P1 , P2 , P3. Gantt Chart as follows:



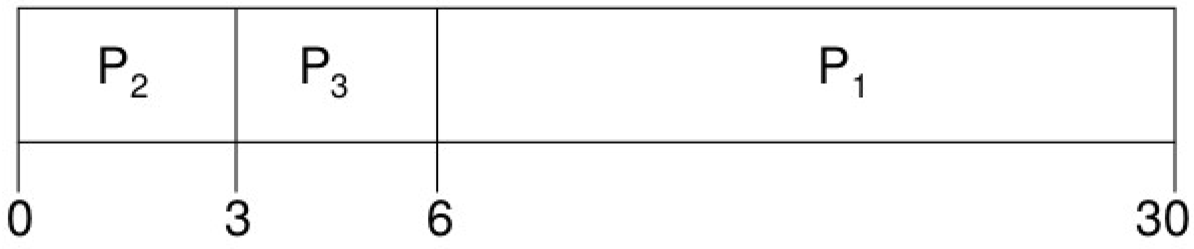
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$ms

Non-preemptive Scheduling

FCFS Example - 2

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose now that the processes arrive in the order: P_2, P_3, P_1. Gantt Chart for schedule:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$ms
- Better, but shows that waiting time under an FCFS policy may vary substantially

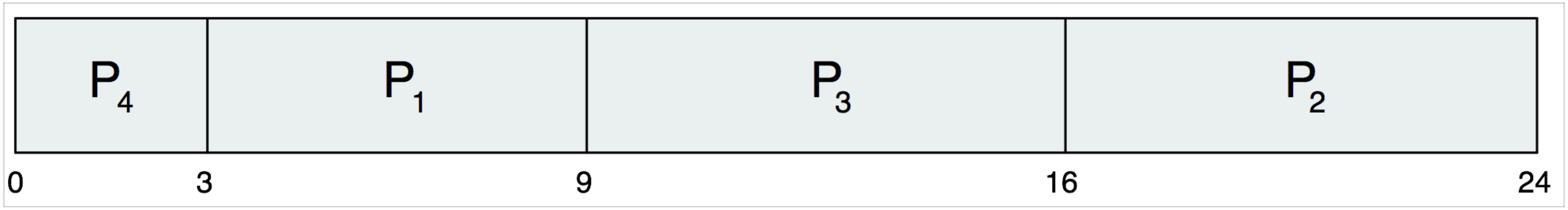
Shortest-Job First (SJF) Scheduling

- Associate with each process the length of the process's next CPU burst
- When the CPU is available it is assigned to the process that has the smallest next CPU burst
  - If next CPU bursts of two processes are the same, FCFS is used to break the tie
- Better name **shortest-next-CPU-burst** algorithm - scheduling depends on the length of the next CPU burst of a process, rather than total length

SJF Example

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

SJF scheduling chart



- Average waiting time: (3 + 16 + 9 + 0)/4 = 7ms
- In comparison, FCFS would be 10.25ms

Optimality of SJF

- Provably optimal - it gives the minimum average waiting time for a given set of processes
- Moving a short process before a long one decreases the waiting time of the short process more than it increases the waiting time of the long process - hence average time decreases

Difficulty of SJF

- Determining the length of the next CPU request is difficult!
- Could ask the user (if they know)?
- For this reason, SJF is difficult to implement at the level of short-term CPU scheduling
- There is no way to know the length of the next CPU burst
- Approximation is a possible solution

Determining the Length of the next CPU burst

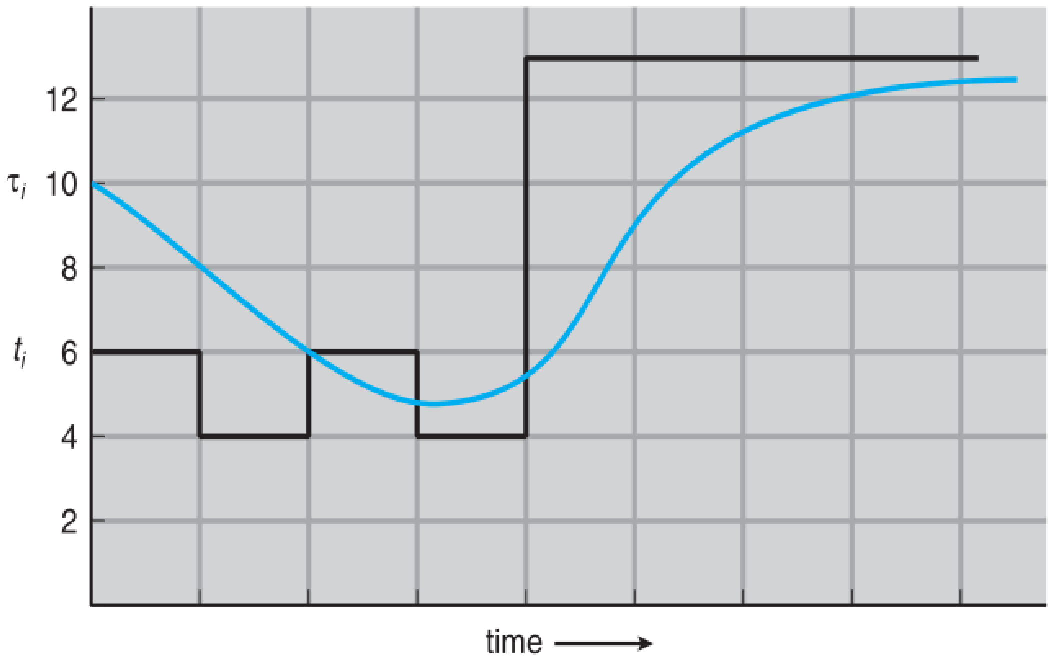
- Exponential averaging of the measured lengths of previous CPU bursts:

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :

$$\tau_{n+1} = \alpha \, t_n + (1 - \alpha) \tau_n$$

Commonly, **α** set to 0.5

Prediction of the Length of the next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$$

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

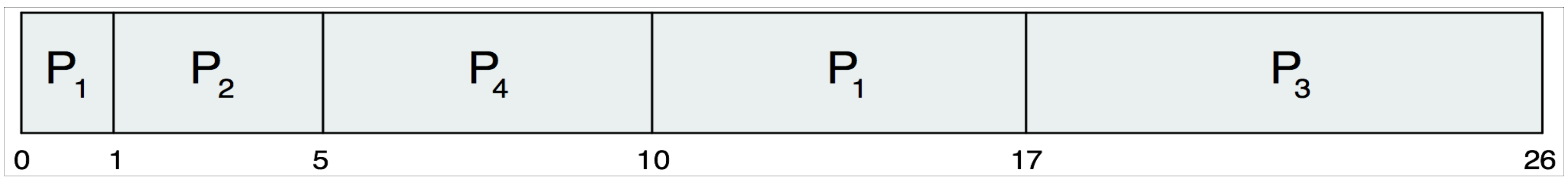
Shortest Remaining Time First Scheduling

- SJF is either preemptive or nonpreemptive
- Choice arises when a new process arrives at the ready queue while a previous process still executing
- **Next CPU burst** of the newly arrived process may be shorter than what is left of the currently executing process
- Preemptive SJF will preempt the currently executing process
- Nonpreemptive will allow the currently running process to finish its CPU burst
- Preemptive SJF scheduling called **Shortest-Remaining-Time-First (SRTF)** scheduling

SRTF Scheduling – 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 = 6.5ms

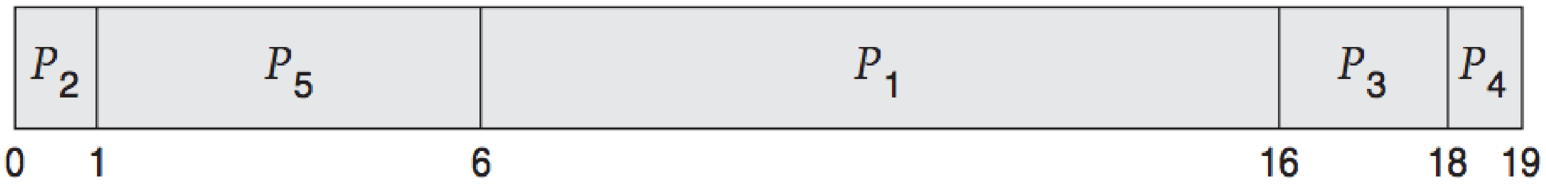
Priority Scheduling

- SJF special case of general **priority-scheduling** algorithm
- Priority associated with each process
- CPU allocated to process with highest priority
  - Equal priority scheduled using FCFS

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

- Low vs. high priority
- Priorities indicated by fixed range of numbers
- **Low numbers = high priority**



Priority Scheduling

- Priorities can be defined internally or externally
- **Internal criteria** = measurable quantity such as time limits, memory requirements, number of open files
- **External criteria** = Outside OS such as process importance, type and amount of funds paid for computation, department sponsoring the work, *political factors*

Priority Scheduling – Preemption

Preemptive vs. Non-preemptive

- Preemptive will preempt the CPU if the priority of the newly arrived process in the ready queue is higher than the priority of the currently running process

- Non-preemptive priority scheduling algorithm will put the new process at the head of the ready queue

Priority Scheduling – Problems

- Indefinite **blocking** or **starvation**
- A process ready to run but waiting for CPU is blocked
- Low priority processes can be blocked indefinitely
- A heavily loaded computer system with a steady stream of high priority processes can prevent a low-priority process from ever getting the CPU
- Two things will probably happen:
  - Process will **eventually run** (at 2 A.M Sunday)
  - Computer will **eventually crash** and lose these unfinished, low priority processes
- **Solution:** importance increases with age (**aging**)
  - Increase importance by a point every 15 minutes

Round-Robin (RR) Scheduling

- Specially designed for **time-sharing systems**
- Similar to FCFS, but with preemption to enable the switching of processes
- Small unit of time used: **time quantum** or **time slice**
  - 10 to 100 milliseconds in length
- Ready queue is circular
- CPU goes around the ready queue, allocating CPU time to each process for a time interval or up to 1 time quantum

Round-Robin (RR) Scheduling

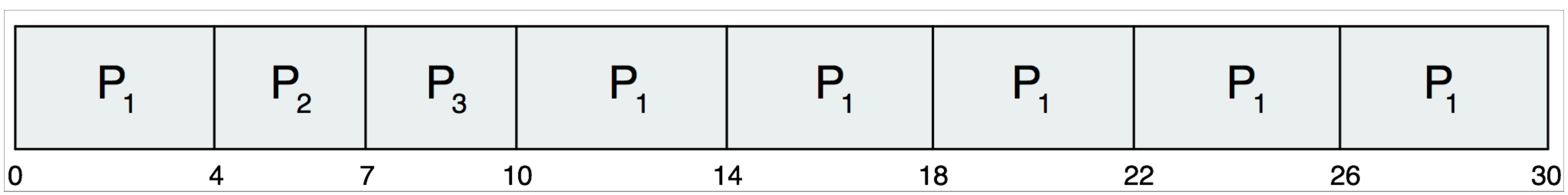
- Implementation treats ready queue as a FIFO queue of processes
- CPU scheduler **picks first process** from ready queue, **sets a timer** to interrupt after 1 time quantum, and dispatches the process
- Two things could happen:
  - Process may have CPU burst of less than 1 quantum, so process will release CPU **voluntarily**
  - If process CPU burst greater than 1 time quantum, timer goes off and causes an **interrupt to the OS**, resulting in a context switch and process goes to the tail of ready queue

Round-Robin (RR) Scheduling – Example

- Time quantum = 4 milliseconds

Process	Burst Time
P_1	24
P_2	3
P_3	3

Resulting RR schedule as follows:



Average waiting time = ((10 - 4) + 4 + 7)/3

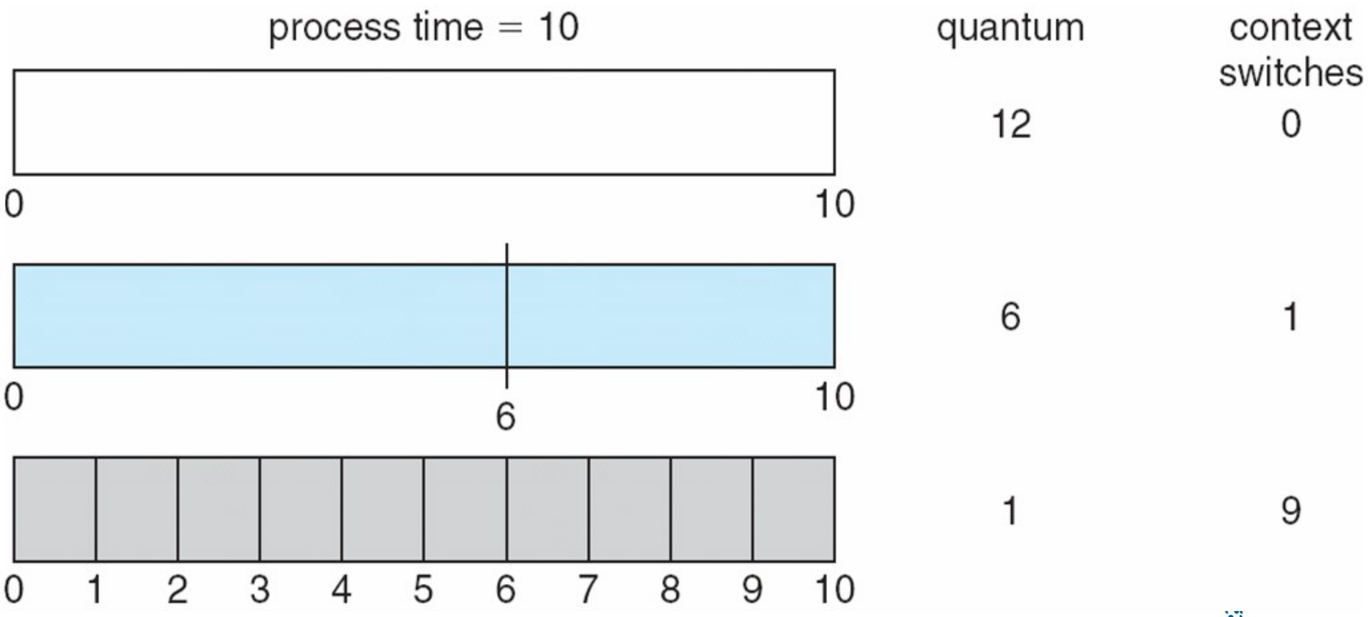
= 17/3 = 5.66ms

Round-Robin (RR) Scheduling

- If there are n processes in the ready queue and the time quantum is **q**, each process gets **1/n** of the CPU time in chunks of at most **q** time units
- Each process must wait no longer than **(n - 1) x q time units** until its next time quantum
- Example:
  - 5 processes
  - q = 20 milliseconds
  - Each process gets up to 20 milliseconds every 100 milliseconds

RR Scheduling Performance

- Depends heavily on **size of time quantum**
  - If extremely large, RR is same as FCFS policy
  - If extremely small, RR can result in a large number of context switches



RR – Time Quantum vs. Context Switch Time

- Ideally, time quantum should be large with respect to the context-switch time
- If context-switch is approximately 10% of time quantum, then ~10% of CPU time will be spent context-switching
- In reality, most modern systems have time quanta ranging from **10 - 100 milliseconds**
  - In comparison, context-switch is typically less than **10 microseconds**

Summary

- Basics of CPU Scheduling
  - CPU-I/O Burst Cycle
  - Preemption
  - Dispatcher
- Scheduling Criteria
- Scheduling Algorithms
  - FCFS, SJF, SRTF, Priority, Round-Robin

References / Links

- Chapter # 6: **CPU Scheduling**, Operating System Concepts (9th edition) by Silberschatz, Galvin & Gagne