DCS216 Operating Systems

Lecture 16
Scheduling (1)

Apr 22nd, 2024

Instructor: Xiaoxi Zhang

Sun Yat-sen University

Announcements

Written Assignment #1

- Hard Deadline: Fri 2024-04-26 22:00
  - Start Early, Finish Early
  - Don't wait until the last HOUR.

题目名称	题目类型	开始时间	截止时间	提交进度
第一次理论作业-填空题计算题	报告题	04-11 08:00	04-26 22:00	25%
第一次理论作业-选择题	选择题	04-11 08:00	04-26 22:00	43%

Content

Content

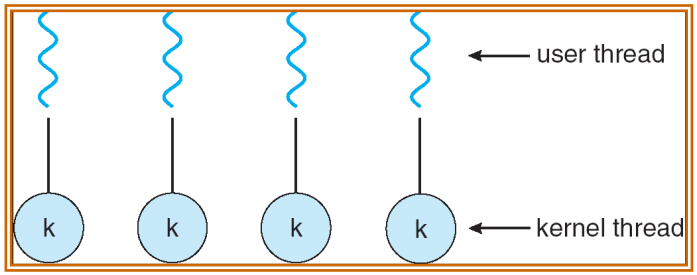
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
  - First-Come, First-Served Scheduling (**FCFS**)
  - Round-Robin Scheduling (**RR**)
  - Shortest-Job-First Scheduling (**SJF**)
    - **SRTF** $\Rightarrow$ Preempted SJF
  - Priority Scheduling
  - Multilevel Queue Scheduling
  - Multilevel Feedback Queue Scheduling (**MFQS**)

Content

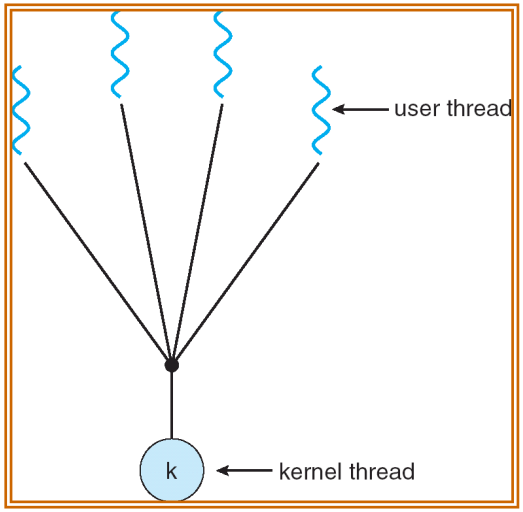
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
  - First-Come, First-Served Scheduling (**FCFS**) 先到先服务调度
  - Round-Robin Scheduling (**RR**) 轮转调度
  - Shortest-Job-First Scheduling (**SJF**) 最短作业优先调度
    - **SRTF** (最短剩余时间优先调度) ⇒ Preempted SJF
  - Priority Scheduling 优先级调度
  - Multilevel Queue Scheduling 多级队列调度
  - Multilevel Feedback Queue Scheduling (**MFQS**) 多级反馈队列调度

Recall: Threading Models

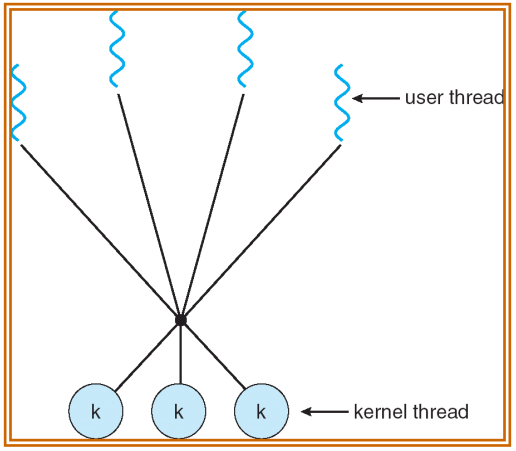
Almost all current implementations



Simple One-to-One Threading Model



Many-to-One



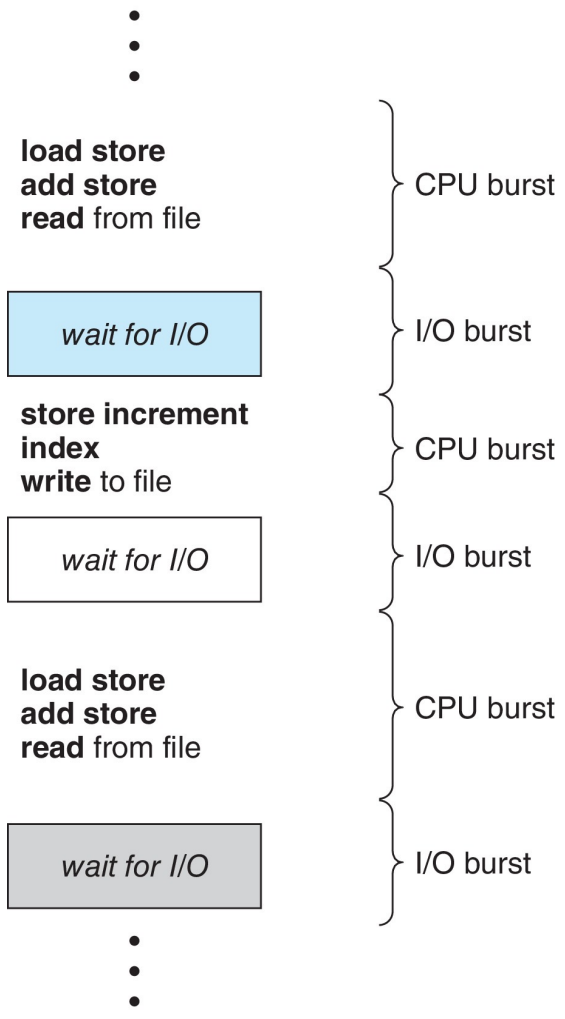
Many-to-Many

Processes vs. Threads

- **Process** is the unit of **resource** management.
  - **进程是资源管理的基本单位**
- **Thread** is the unit of **concurrency** and **scheduling**.
  - **线程是并发和调度的基本单位**
- On modern OSes, it is the kernel-level threads (**NOT** processes) that are in fact being scheduled by the OS.
- However, the terms "**process scheduling**" and "**thread scheduling**" are often used ***interchangeably***.
- In this class, we use "**process scheduling**" when discussing general scheduling concepts; and use "**thread scheduling**" to refer to thread-specific ideas.

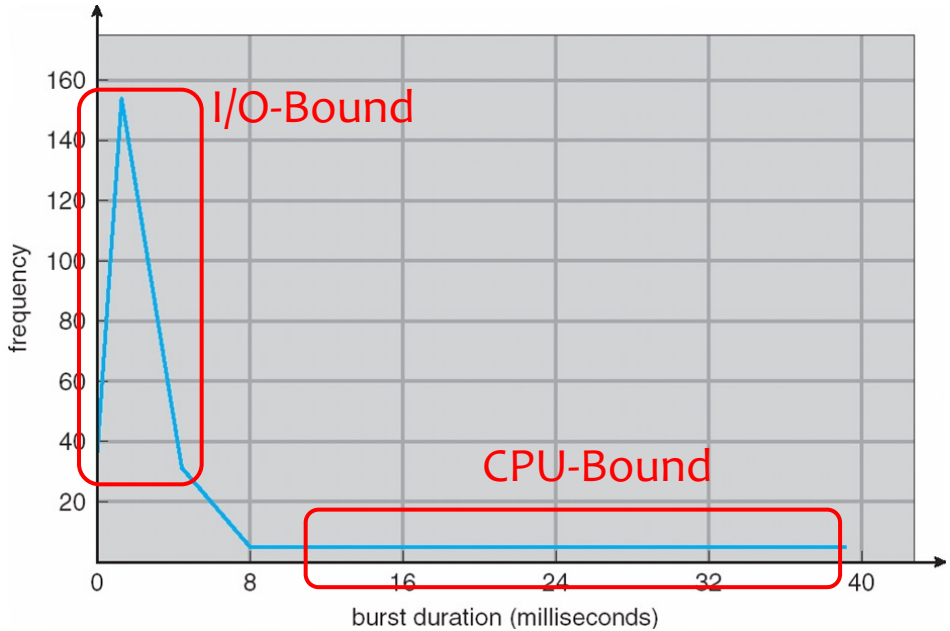
CPU Bursts

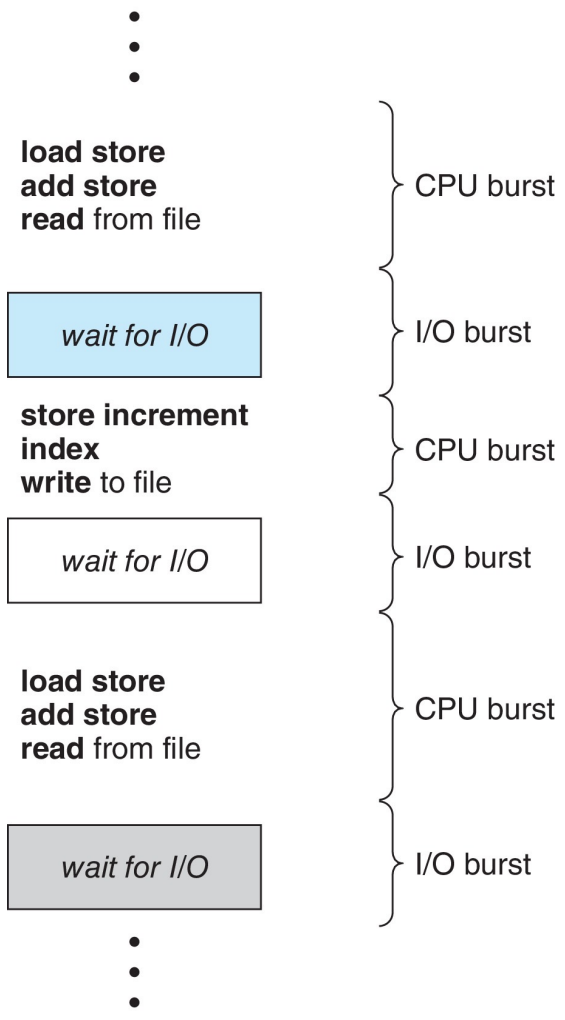
- Execution model: processes alternate between bursts of **CPU** and **I/O**.
  - Process execution begins with a **CPU Burst** (CPU突发)
  - ...followed by (a longer) **I/O Burst** (I/O突发)...
  - ...followed by another **CPU Burst,**
  - ...then another **I/O Burst**...
  - ...usually **terminates** with a final **CPU Burst**.



CPU Bursts

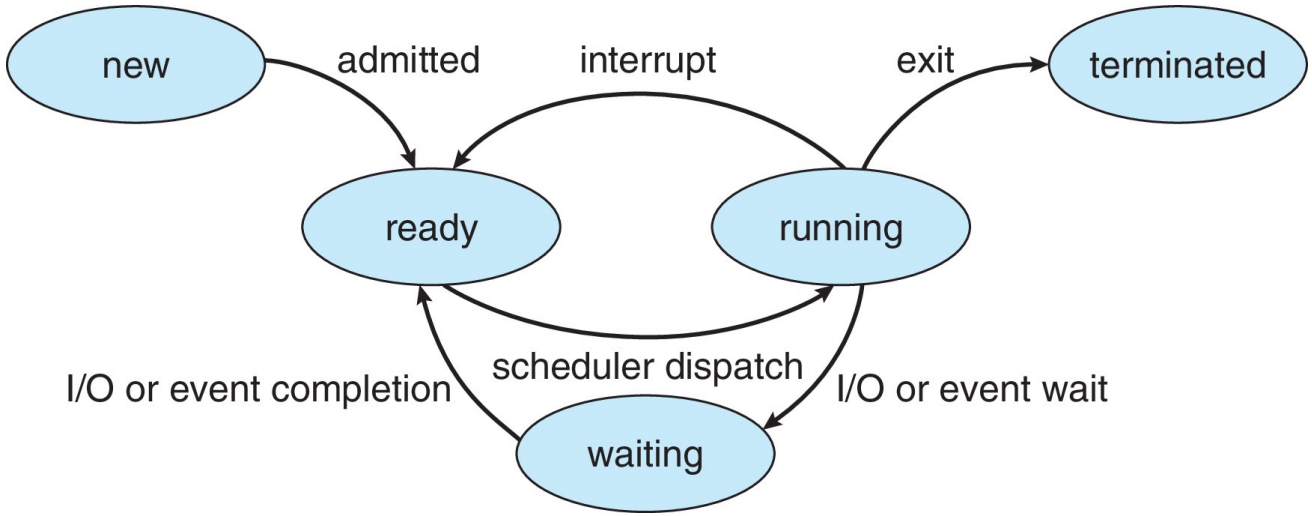
- Execution model: processes alternate between bursts of **CPU** and **I/O**.
  - Process execution begins with a **CPU Burst** (CPU突发)
  - ...followed by (a longer) **I/O Burst** (I/O突发)...
  - ...followed by another **CPU Burst,**
  - ...then another **I/O Burst**...
  - ...usually **terminates** with a final **CPU Burst.**
- Histogram of CPU Burst time





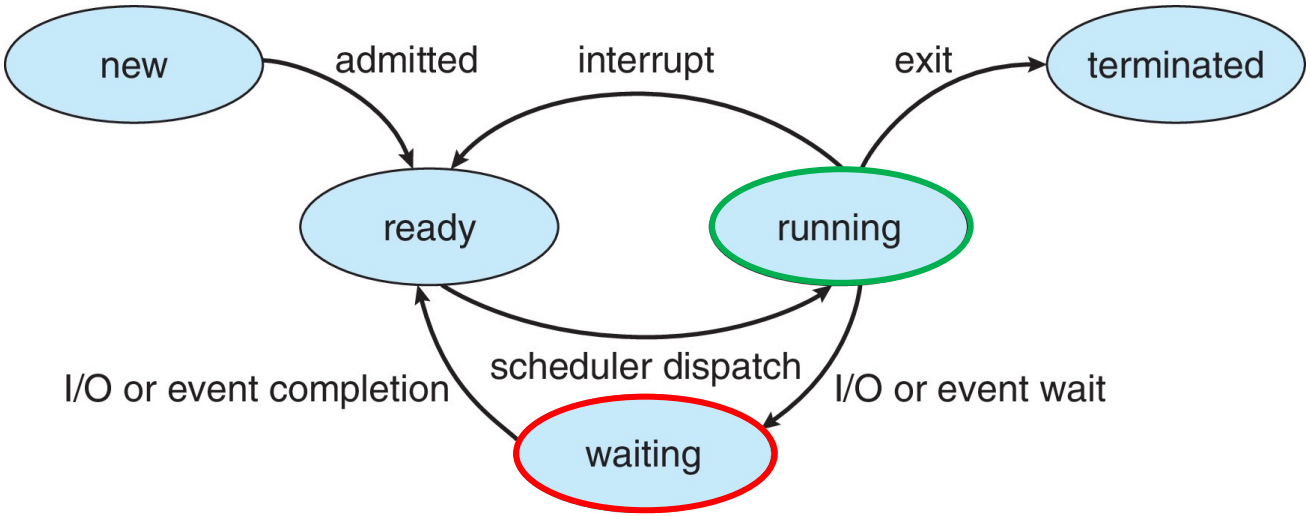
CPU Scheduler

- The CPU Scheduler selects among the processes in the **ready** queue, and allocates a CPU core to that **selected process.**
  - 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 state
  4. Terminates (Switch from RUNNING to TERMINATED state)



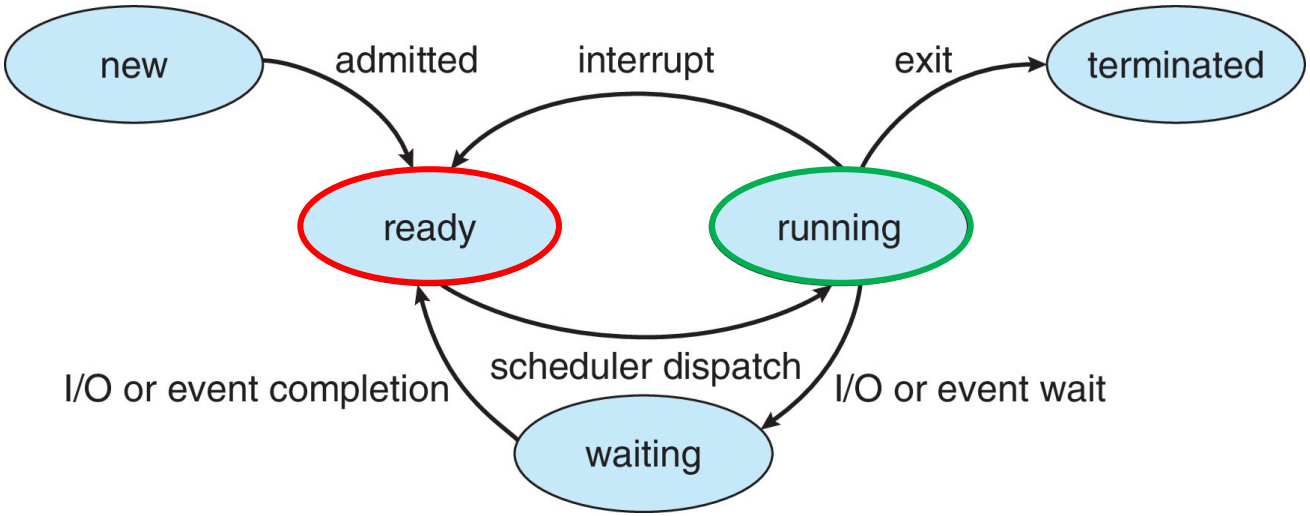
CPU Scheduler

- The CPU Scheduler selects among the processes in the **ready** queue, and allocates a CPU core to that **selected process.**
  - 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 state
  4. Terminates (Switch from RUNNING to TERMINATED state)



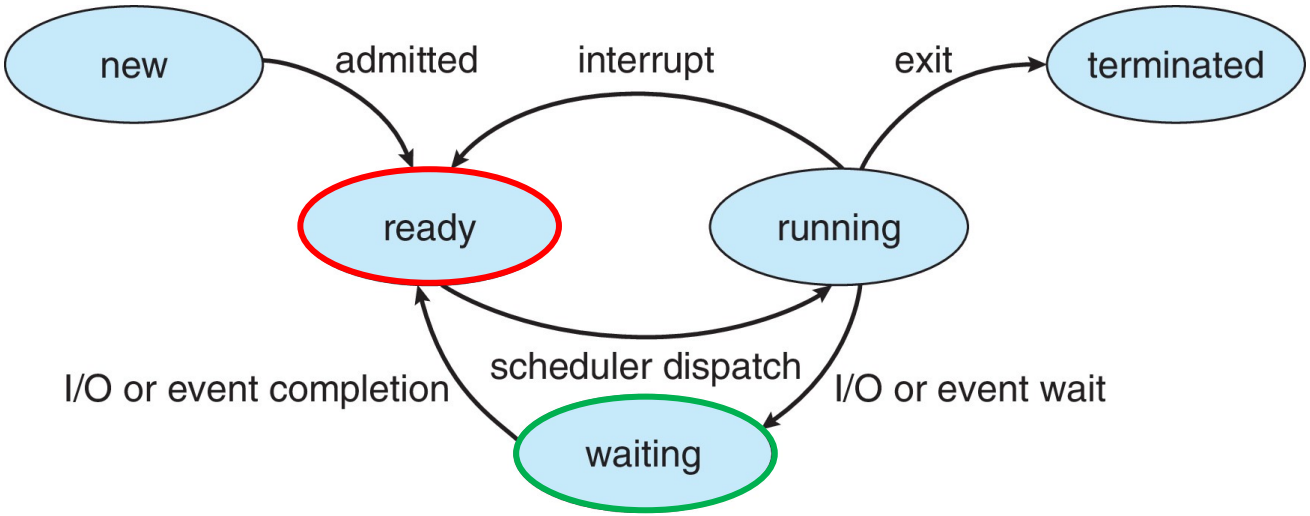
CPU Scheduler

- The CPU Scheduler selects among the processes in the **ready** queue, and allocates a CPU core to that **selected process.**
  - 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 state
  4. Terminates (Switch from RUNNING to TERMINATED state)



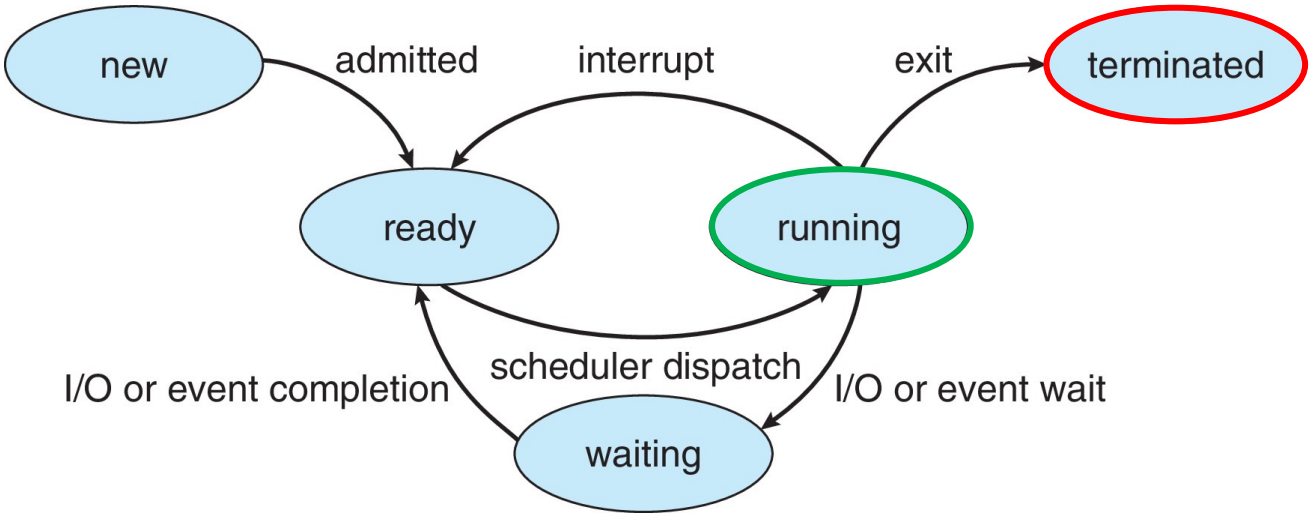
CPU Scheduler

- The CPU Scheduler selects among the processes in the **ready** queue, and allocates a CPU core to that **selected process.**
  - 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** state
  4. Terminates (Switch from RUNNING to TERMINATED state)



CPU Scheduler

- The CPU Scheduler selects among the processes in the **ready** queue, and allocates a CPU core to that **selected process.**
  - 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 state
  4. Terminates (Switch from **RUNNING** to **TERMINATED** state)



CPU Scheduler

- The CPU Scheduler selects among the processes in the **ready** queue, and allocates a CPU core to that **selected process**.
  - 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 state
  4. Terminates (Switch from **RUNNING** to **TERMINATED** state)
- For situation #1 and #4, there is no choice in terms of scheduling.
  - A new process (if exists in ready queue) must be selected for execution

CPU Scheduler

- The CPU Scheduler selects among the processes in the **ready** queue, and allocates a CPU core to that **selected process**.
  - 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** state
  4. Terminates (Switch from RUNNING to TERMINATED state)
- For situation #1 and #4, there is no choice in terms of scheduling.
  - A new process (if exists in ready queue) must be selected for execution
- For situation #2 and #3, however, there is a **choice**...

- **Preemptive and Nonpreemptive Scheduling**
  - When scheduling takes place **ONLY** under situation #1 and #4, the scheduling scheme is **nonpreemptive** (or cooperative).
  - Otherwise, the scheduling scheme is **preemptive**.

Preemptive and Nonpreemptive Scheduling

- When scheduling takes place **ONLY** under situation #1 and #4, the scheduling scheme is **nonpreemptive** (or cooperative).
- Otherwise, the scheduling scheme is **preemptive**.
- Under nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by **terminating** or by voluntarily **switching** to the **waiting** state.
- Virtually all modern OSes (incl. Linux, MacOS, Windows) use **preemptive** scheduling algorithms.
- **Preemptive** scheduling can result in race conditions
  - when data are shared among several processes
  - or when data are shared among multiple threads in a single process

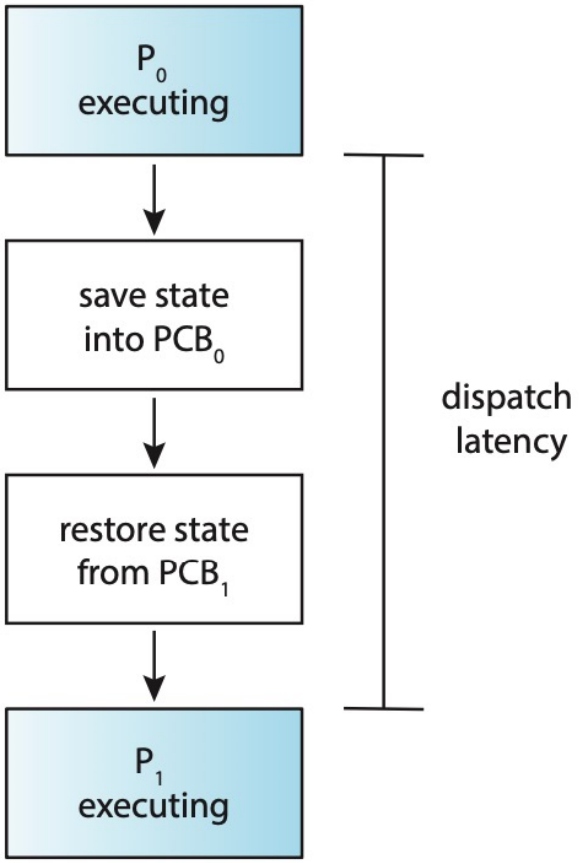
- **Dispatcher**
  - Dispatcher module (分派程序) gives control of the CPU to the process selected by the scheduler.
  - The function of dispatcher involves:
    - Switching context from one process to another
    - Switching to user mode
    - Jumping to the proper location in the user program to resume

Dispatcher

- Dispatcher module (分派程序) gives control of the CPU to the process selected by the scheduler.
- The function of dispatcher involves:
  - Switching context from one process to another **(Context Switch)**
  - Switching to user mode **(Mode Switch)**
  - Jumping to the proper location in the user program to resume

Dispatcher

- Dispatcher module (分派程序) gives control of the CPU to the process selected by the scheduler.
- The function of dispatcher involves:
  - Switching context from one process to another **(Context Switch)**
  - Switching to user mode **(Mode Switch)**
  - Jumping to the proper location in the user program to resume
- **Dispatch Latency** (分派延迟)
  - The dispatcher should be as fast as possible, since it is invoked during every process switch.
  - The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**.



Scheduling Criteria

- **Scheduling Criteria**
  - CPU Utilization (CPU利用率)
  - Throughput (吞吐量)
  - Turnaround Time (周转时间)
  - Waiting Time (等待时间)
  - Response Time (响应时间)

Scheduling Criteria

- **Scheduling Criteria**
  - **CPU Utilization**
    - Keep the CPU as busy as possible. Ranges from (0% to 100%)
  - **Throughput**
    - The number of processes that complete their execution per time unit
  - **Turnaround Time**
    - The amount of time to execute a particular process
    - Turnaround Time = Time spent in:
      - **waiting** in the ready queue
      - **executing** on the CPU
      - doing **I/O**
  - **Waiting Time**
    - The amount of time a process has been **waiting in the ready queue**.
  - **Response Time**
    - The amount of time it takes from request submission until the first response is produced.
    - A good metric for interactive systems.

Scheduling Criteria

- **Scheduling Algorithm Optimization Criteria**
  - CPU Utilization (CPU利用率)
    - **Max**imize
  - Throughput (吞吐量)
    - **Max**imize
  - Turnaround Time (周转时间)
    - **Min**imize
  - Waiting Time (等待时间)
    - **Min**imize
  - Response Time (响应时间)
    - **Min**imize

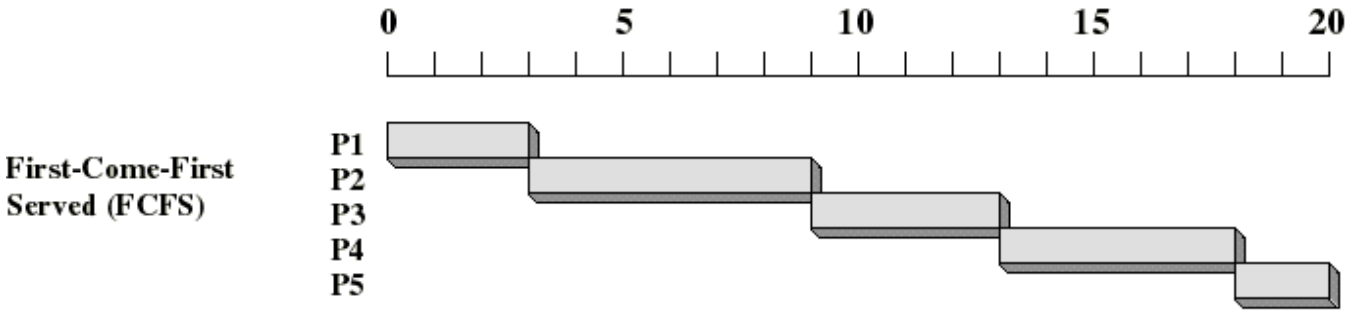
Scheduling Criteria

- **Scheduling Goals for Different Systems**
  - Batch Systems
    - Throughput
    - Turnaround Time
    - CPU Utilization
  - Interactive Systems
    - Response Time
    - Proportionality (平衡性)
  - Real-Time Systems
    - Meeting Deadlines
    - Predictability
  - User-Oriented
    - Response Time
    - Turnaround Time
  - System-Oriented
    - CPU Utilization; Throughput; Fairness...

First-Come, First-Served Scheduling (FCFS)

- **Selection Function:** the process that has been waiting the longest in the ready queue – hence First-Come, First-Served (FCFS).
- **Implementation:** FIFO ready queue
- **Decision Mode:** **Non-Preemptive**
- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	3
P_2	2	6
P_3	4	4
P_4	6	5
P_5	8	2

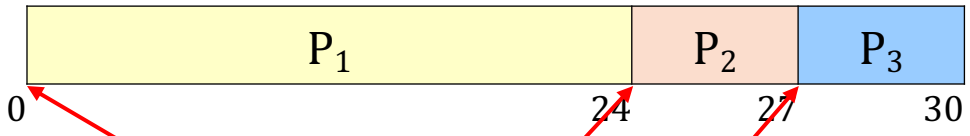


First-Come, First-Served Scheduling (FCFS)

- A simpler FCFS example:

Process	CPU 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 (甘特图) for the schedule is:



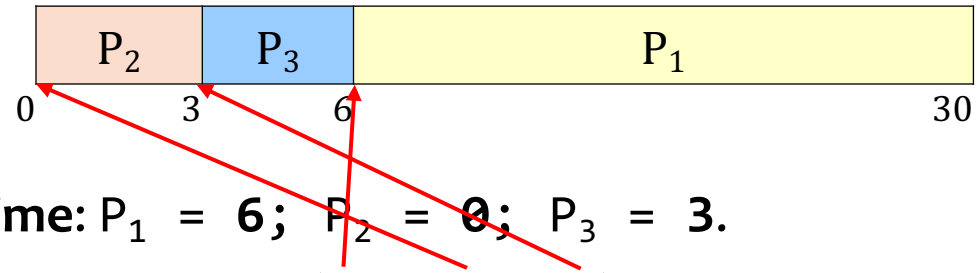
- **Waiting Time:** P_1 = **0**; P_2 = **24**; P_3 = **27**.
- **Average Waiting Time:** (**0** + **24** + **27**)/3 = 17.
- Convoy effect (护航效应): short process **stuck** behind long process
  - Consider one CPU-bound and many I/O-bound processes

First-Come, First-Served Scheduling (FCFS)

- A simpler FCFS example:

Process	CPU 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 (甘特图) for the schedule is:



- **Waiting Time:** P_1 = **6**; P_2 = **0**; P_3 = **3**.
- **Average Waiting Time:** (**6** + **0** + **3**)/3 = 3.
- Much better than previous case.

Scheduling Algorithms

First-Come, First-Served Scheduling (FCFS)

- FCFS **Pros:**
  - Simple
- FCFS **Cons:**
  - A process that does not perform any I/O will monopolize the CPU.
  - All other processes wait for the (one long) process to release the CPU (**Convoy** Effect).
  - Favors CPU-bound processes:
    - I/O-bound processes have to wait until CPU-bound processes complete
    - I/O devices may have to wait even when their I/O are completed (poor device utilization)
  - We could have kept the I/O devices busy by giving a bit more priority to I/O bound processes.

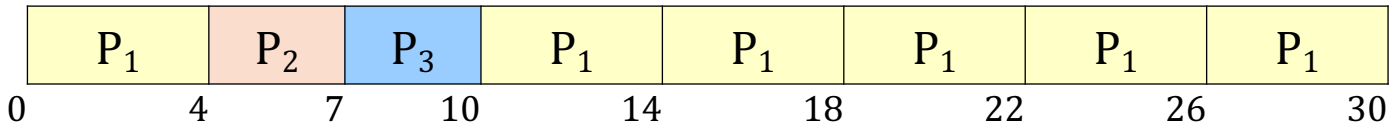
- **Round Robin Scheduling (RR)**
  - FCFS Scheme: Potentially bad for short jobs.
    - Depends on submission order
  - **Round Robin**: FCFS + Preemption!
    - Each process gets a small unit of CPU time (time quantum), usually 10 ~ 100 milliseconds
    - After quantum expires, the process is preempted and appended to the end of the ready queue.

Round Robin Scheduling (RR)

- Example of RR with time quantum ($q = 4$).

Process	CPU Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive at the same time. The Gantt Chart for the schedule is:



- **Waiting Time:** P_1 = (10-4) = 6; P_2 = 4; P_3 = 7
- **Average Waiting time:** (6 + 4 + 7) / 3 = 5.66
- Typically, RR ⇒ worse Average Waiting Time & better Response Time

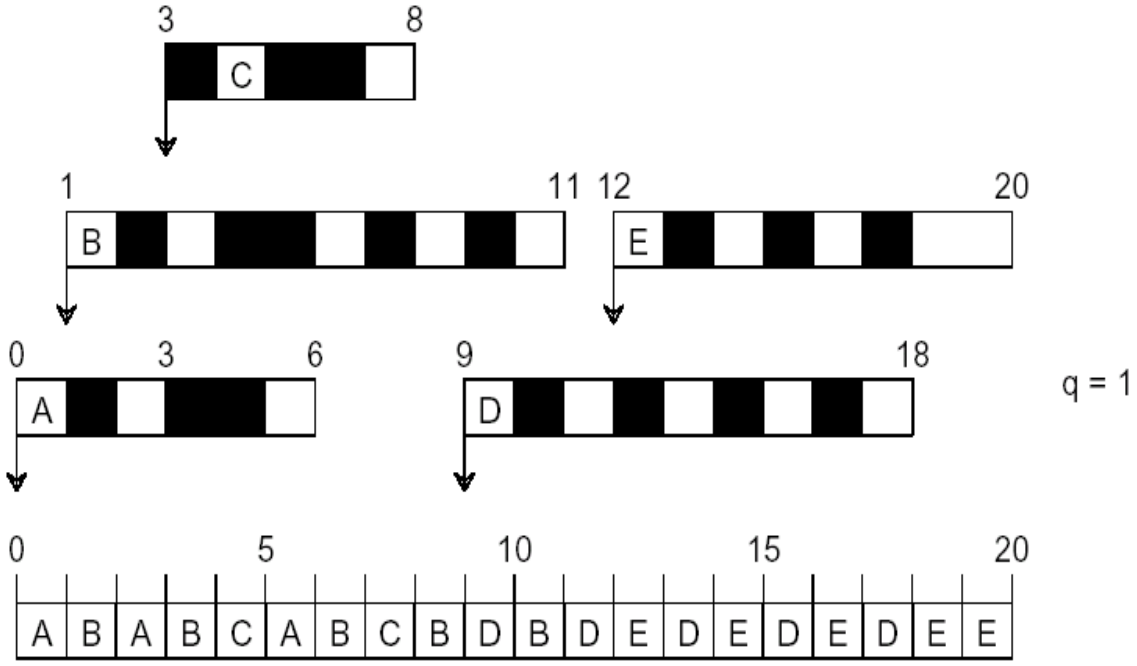
Round Robin Scheduling (RR)

- FCFS Scheme: Potentially bad for short jobs.
  - Depends on submission order
- **Round Robin**: FCFS + Preemption!
  - Each process gets a small unit of CPU time (time quantum), usually 10 ~ 100 milliseconds
  - After quantum expires, the process is preempted and appended to the end of the ready queue.
  - n processes in ready queue and time quantum is $q \Rightarrow$
    - Each process gets $1/n$ of the CPU time
      - In chunks of at most q time units
    - No process waits more than $(n-1)q$ time units until its next time quantum.
      - For example, with 5 processes and a time quantum of 20ms, each process will get up to 20ms every 100ms.

Round Robin Scheduling (RR)

- Example of RR with Time Quantum ($q = 1$)

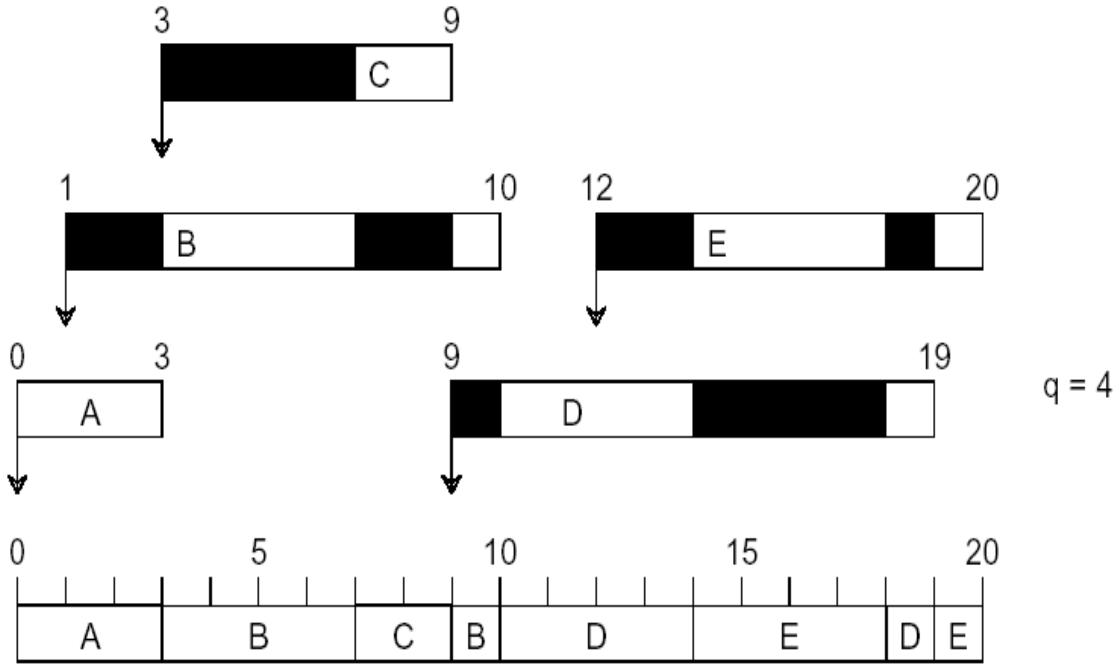
Process	Arrival Time	CPU Burst Time
A	0	3
B	1	5
C	3	2
D	9	5
E	12	5



Round Robin Scheduling (RR)

- Example of RR with Time Quantum ($q = 4$)

Process	Arrival Time	CPU Burst Time
A	0	3
B	1	5
C	3	2
D	9	5
E	12	5

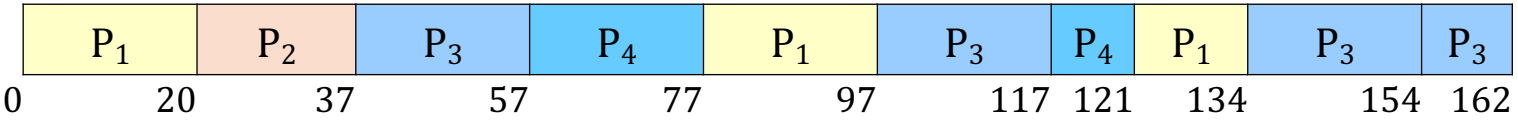


Round Robin Scheduling (RR)

- Example of RR with Time Quantum ($q = 20$). Suppose that all processes arrive at the same time.

Process	CPU Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

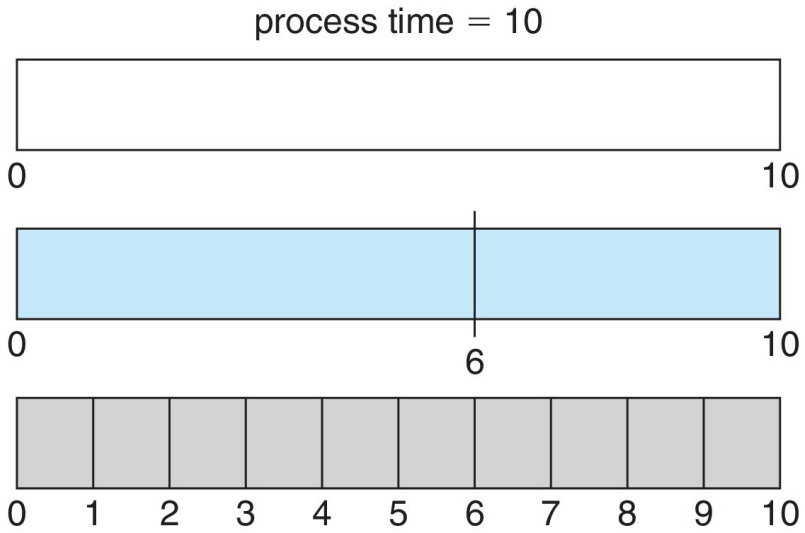
- The scheduling Gantt chart:



- If there are n processes in the ready queue, and the time quantum is q, then no process waits more than $(n-1)\times q$ time units.

Round Robin Scheduling (RR)

- Example: Smaller time quantum increases context switches.
- Suppose we have only one process of 10 time units.



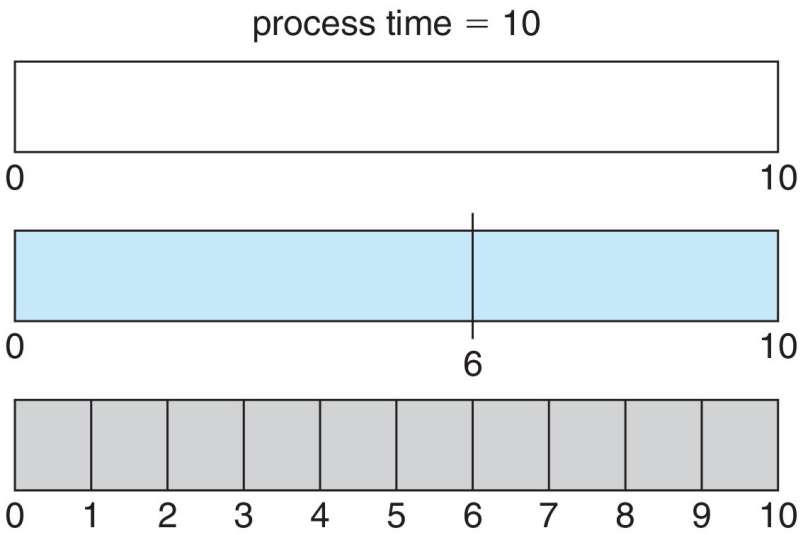
quantum	context switches
12	0
6	1
1	9

- If the context-switch time is 10% of the time quantum, then about 10% of the CPU time will be spent (wasted) in context switching.

Round Robin Scheduling (RR)

Performance

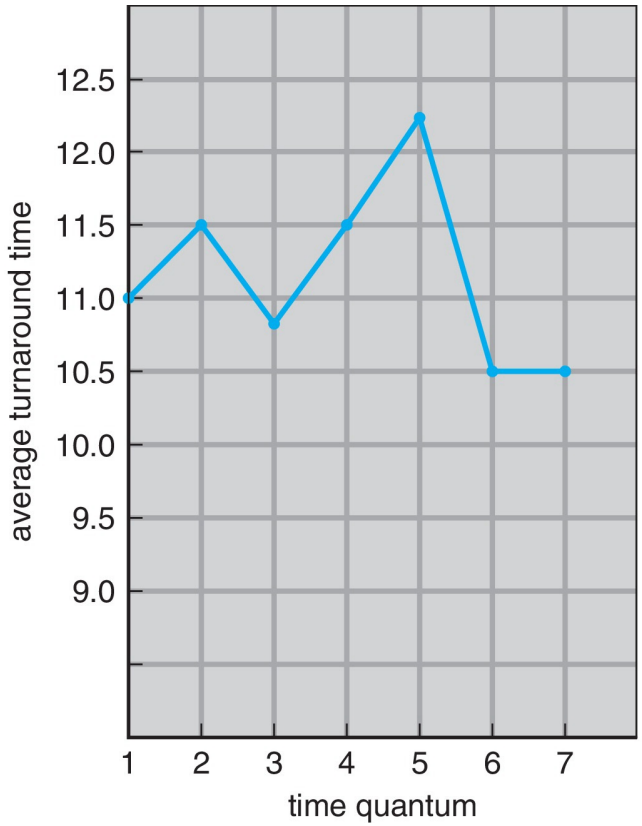
- depends **heavily** on the size of the time quantum q
- q large $\Rightarrow$ FCFS (FIFO)
- q small $\Rightarrow$ Interleaved
- q must be larger than context switch, otherwise overhead is too high.
  - q usually ranges from 10ms to 100ms
  - context switch overhead ~ 10µs ($< 0.1\%$ of q)



quantum	context switches
12	0
6	1
1	9

Round Robin Scheduling (RR)

- Turnaround time varies with the time quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

- The average turnaround time of a set of processes does not necessarily improve as the time quantum size increases.

Round Robin Scheduling (RR)

- Turnaround time varies with the time quantum
  - In general, the average turnaround time can be improved if most processes finish their next CPU bursts in a single time quantum.
    - For example, given 3 processes of 10 time units each, and a time quantum of 1 time unit, the average turnaround time is 29. If the time quantum is 10, however, the average turnaround time drops to 20.
  - In addition, if context-switch time is added in, the average turnaround time increases even more for a smaller time quantum, since more context switches are required.

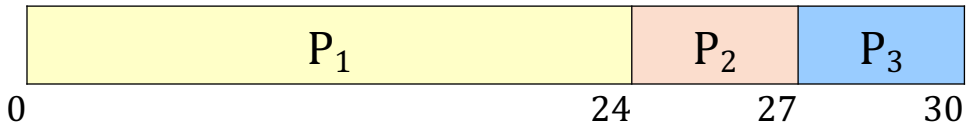
First-Come, First-Served Scheduling (FCFS)

- A simpler FCFS example:

Process	CPU Burst Time
P_1	24
P_2	3
P_3	3

Notice that the CPU Burst Time info is not used in **FCFS** scheduling decision at all!

- Suppose that the processes arrive in the order: <P_1, P_2, P_3>. The Gantt Chart (甘特图) for the schedule is:



- **Waiting Time:** P_1 = **0**; P_2 = **24**; P_3 = **27**.
- **Average Waiting Time:** (**0** + **24** + **27**)/3 = 17.
- Convoy effect (护航效应): short process **stuck** behind long process
  - Consider one CPU-bound and many I/O-bound processes

- **Shortest-Job-First Scheduling (SJF)**
  - **Selection Function:** the process with the **shortest** next CPU burst time in the ready queue – hence Shortest-Job-First (SJF).
  - Associate with each process the length of its next CPU burst
    - Use these lengths to schedule the process with the shortest time.
  - **Decision Mode:** **Non-Preemptive** or **Preemptive**
    - **Preemptive** SJF is also called **Shortest-Remaining-Time-First** (**SRTF**)

Shortest-Job-First Scheduling (SJF)

- An SJF Example: (suppose all processes arrive at the same time)

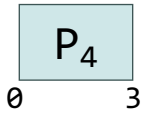
Process	CPU Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Shortest-Job-First Scheduling (SJF)

- An SJF Example: (suppose all processes arrive at the same time)

Process	CPU Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF Gantt Chart:

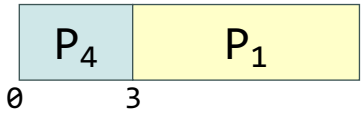


Shortest-Job-First Scheduling (SJF)

- An SJF Example: (suppose all processes arrive at the same time)

Process	CPU Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF Gantt Chart:

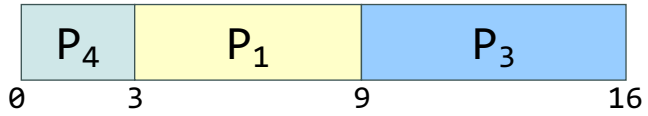


Shortest-Job-First Scheduling (SJF)

- An SJF Example: (suppose all processes arrive at the same time)

Process	CPU Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF Gantt Chart:

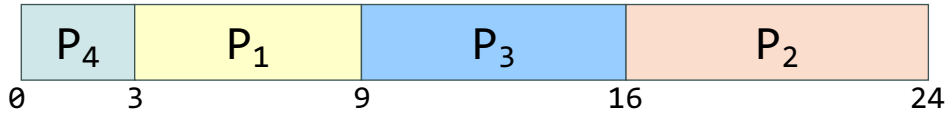


Shortest-Job-First Scheduling (SJF)

- An SJF Example: (suppose all processes arrive at the same time)

Process	CPU Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF Gantt Chart:



- Average Waiting Time:

```
(3 + 16 + 9 + 0) / 4 = 7
```

- Average Turnaround Time:

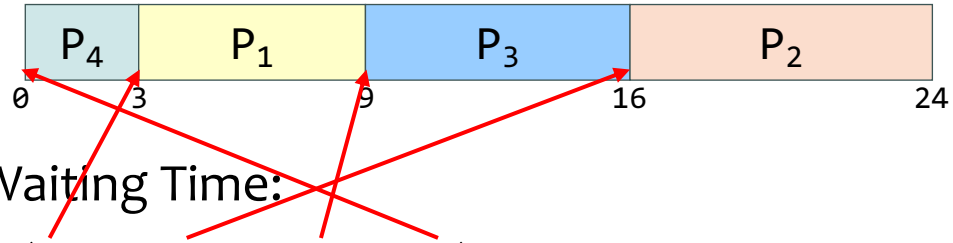
```
(9 + 24 + 16 + 3) / 4 = 13
```

Shortest-Job-First Scheduling (SJF)

- An SJF Example: (suppose all processes arrive at the same time)

Process	CPU Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF Gantt Chart:



- Average Waiting Time:

$$(3 + 16 + 9 + 0) / 4 = 7$$

- Average Turnaround Time:

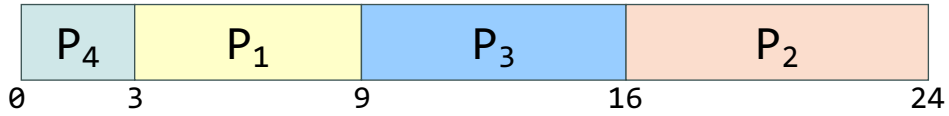
$$(9 + 24 + 16 + 3) / 4 = 13$$

Shortest-Job-First Scheduling (SJF)

- An SJF Example: (suppose all processes arrive at the same time)

Process	CPU Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF Gantt Chart:



- Average Waiting Time:

```
(3 + 16 + 9 + 0) / 4 = 7
```

- Average Turnaround Time:

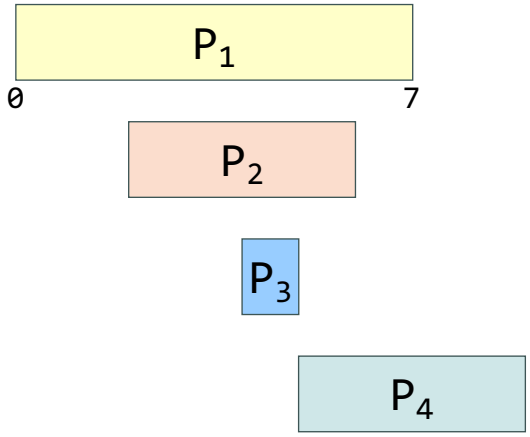
```
(9 + 24 + 16 + 3) / 4 = 13
```

Shortest-Job-First Scheduling (SJF)

- Another SJF Example: (processes with different arrival times)

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

- SJF Gantt Chart:

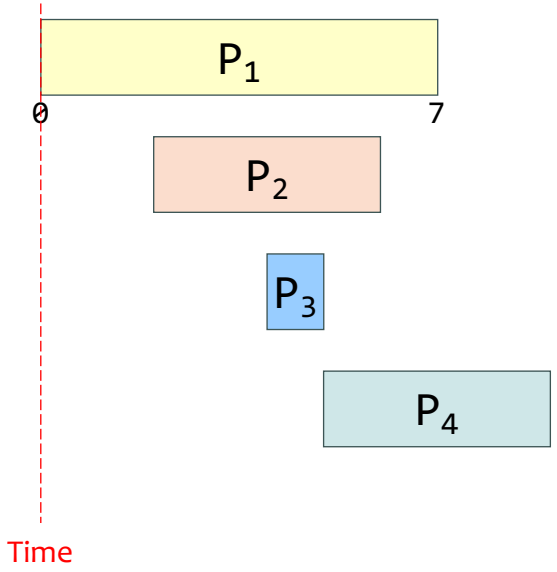


Shortest-Job-First Scheduling (SJF)

- Another SJF Example: (processes with different arrival times)

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

- SJF Gantt Chart:

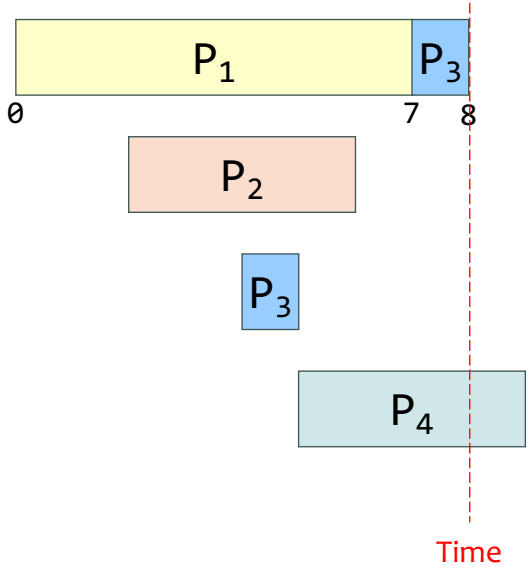


Shortest-Job-First Scheduling (SJF)

- Another SJF Example: (processes with different arrival times)

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

- SJF Gantt Chart:

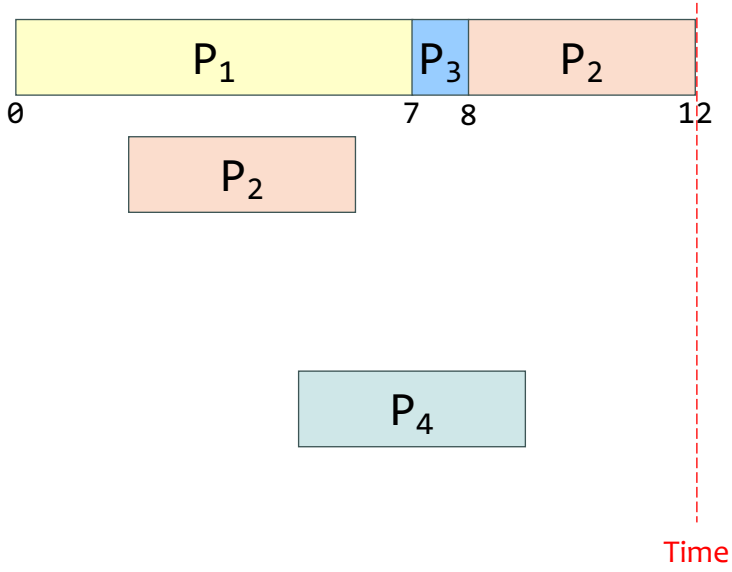


Shortest-Job-First Scheduling (SJF)

- Another SJF Example: (processes with different arrival times)

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

- SJF Gantt Chart:

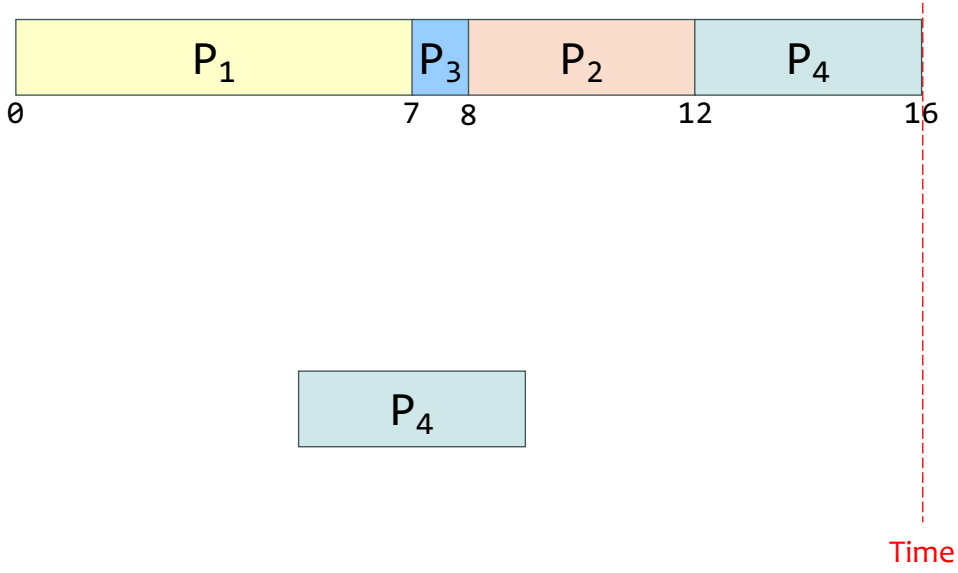


Shortest-Job-First Scheduling (SJF)

- Another SJF Example: (processes with different arrival times)

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

- SJF Gantt Chart:

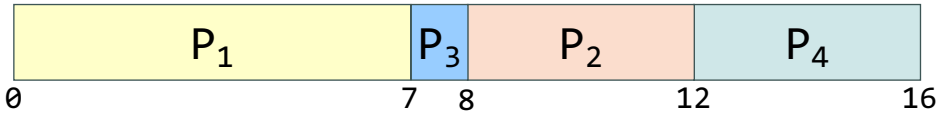


Shortest-Job-First Scheduling (SJF)

- Another SJF Example: (processes with different arrival times)

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

- SJF Gantt Chart:

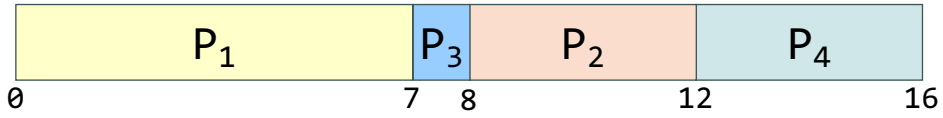


Shortest-Job-First Scheduling (SJF)

- Another SJF Example: (processes with different arrival times)

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

- SJF Gantt Chart:



- Average Waiting Time:
  - `[(0-0)+(8-2)+(7-4)+(12-5)] / 4 = 4`
- Average Turnaround Time:
  - `[(7-0)+(12-2)+(8-4)+(16-5)] / 4 = 8`

- **Shortest-Job-First Scheduling (SJF)**
  - **Selection Function:** the process with the **shortest** next CPU burst time in the ready queue – hence Shortest-Job-First (SJF).
  - Associate with each process the length of its next CPU burst
    - Use these lengths to schedule the process with the shortest time.
  - **Decision Mode:** **Non-Preemptive** or **Preemptive**
    - **Preemptive** SJF is also called **Shortest-Remaining-Time-First** (**SRTF**)
  - SJF is **optimal**:
    - It gives the **minimum** average waiting time for a given set of processes.
    - Moving a short process before a long process **decreases** the waiting time of the short process more than it **increases** the waiting time of the long process.
    - Consequently, the average waiting time decreases. In other words, it is always better to move the short process in front of a long one.

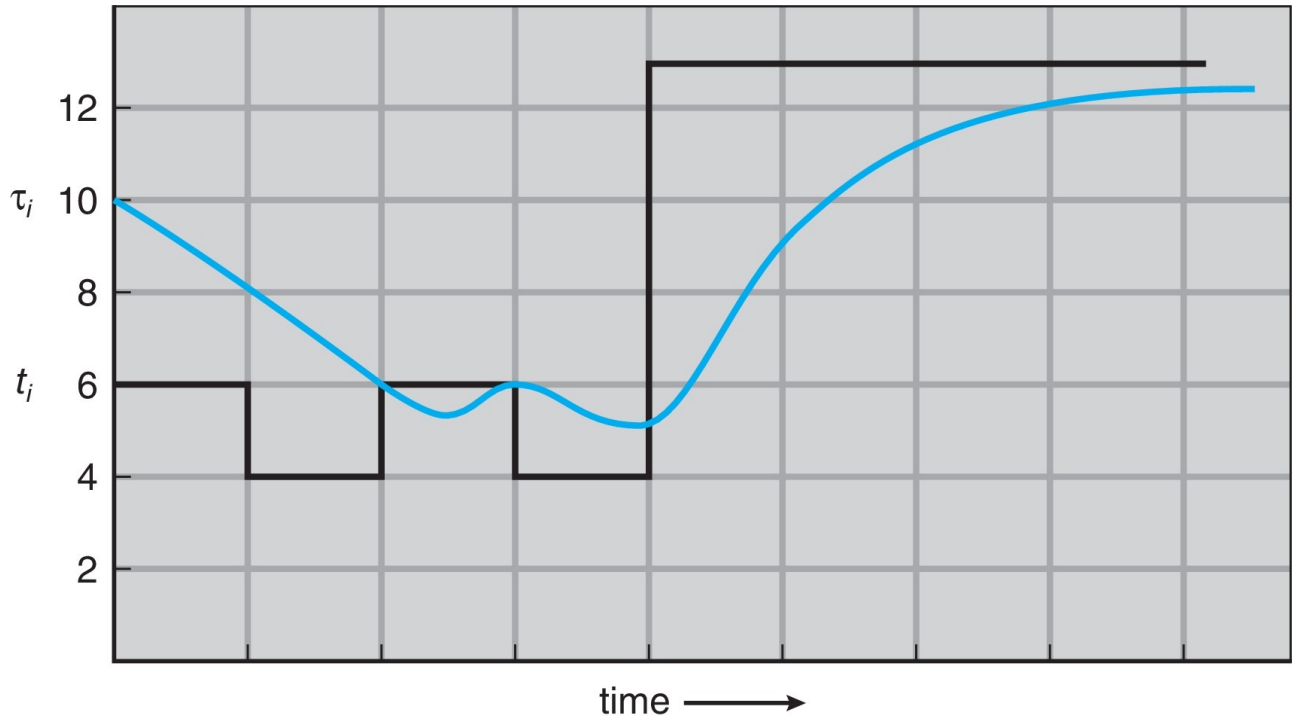
- **Shortest-Job-First Scheduling (SJF)**
  - Although SJF is optimal, it is not practical to implement at the level of the CPU scheduling.
  - In reality, impossible to get the exact **Length of the Next CPU Burst.**
  - One approach is to approximate SJF scheduling: we estimate the length of the next CPU burst: should be similar to previous ones
    - Then pick process with the **shortest** predicted next CPU burst.
  - Can be done by using the length of previous CPU bursts, using exponential averaging
    - t_n = actual length of n^{th} CPU burst
    - τ_{n+1} = predicted value for the next CPU burst
    - for α, $0 \leq \alpha \leq 1$,
    - Define: $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$

Shortest-Job-First Scheduling (SJF)

- Prediction of the length of the next CPU burst: exponential averaging
  - t_n = actual length of n^{th} CPU burst
  - τ_{n+1} = predicted value for the next CPU burst
  - for α, $0 \leq \alpha \leq 1$,
  - Define: $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$
- How to set α?
  - If $\alpha = 0$
    - $\tau_{n+1} = \tau_n$
    - Recent history has no effect (current conditions assumed to be transient)
  - If $\alpha = 1$
    - $\tau_{n+1} = t_n$
    - Only the most recent CPU burst matters (history is assumed to be old and irrelevant)
  - More commonly, $\alpha = 0.5$

Shortest-Job-First Scheduling (SJF)

- 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	...

- **Shortest-Job-First Scheduling (SJF)**
  - Prediction of the length of the next CPU burst
  - If we expand the formula $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$, we get:

$$\begin{aligned}\tau_{n+1} = \alpha t_n &+ (1-\alpha)^1 \tau_{n-1} + \cdots \\ &+ (1-\alpha)^2 \tau_{n-2} + \cdots \\ &+ (1-\alpha)^j \tau_{n-j} + \cdots \\ &+ (1-\alpha)^{n+1} \tau_0\end{aligned}$$

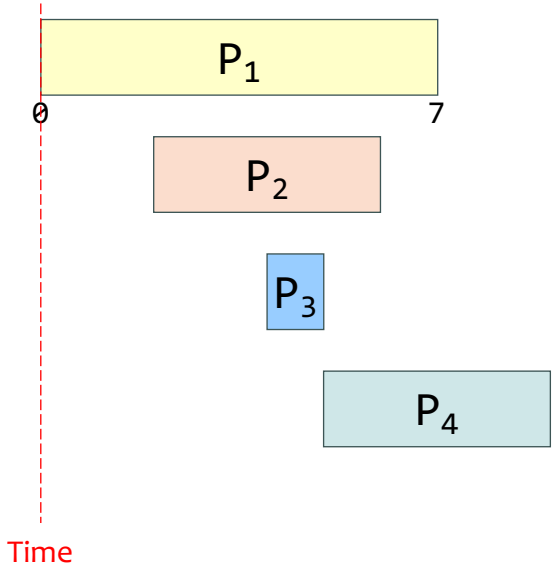
  - Since both α and $(1-\alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor; term weights are decreasing exponentially.

Shortest-Job-First Scheduling (SJF)

- SJF Scheduling can be either **Preemptive** or **Non-Preemptive**.
  - **Preemptive** SJF $\Rightarrow$ **SRTF**
- The difference lies in the **choice** to be made when a new process arrives at the ready queue while a previous process is still executing.
- If the next CPU burst of the newly arrived process is shorter than what is left of the currently executing process, SRTF will preempt the current process, while SJF will allow it to finish completion.

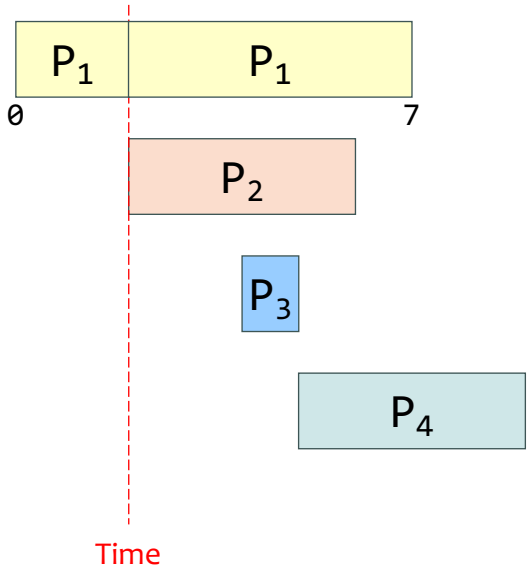
Shortest-Job-First Scheduling (SJF)

- SJF Scheduling can be either **Preemptive** or **Non-Preemptive**.
  - **Preemptive** SJF ⇒ **SRTF**
- The difference lies in the **choice** to be made when a new process arrives at the ready queue while a previous process is still executing.
- If the next CPU burst of the newly arrived process is shorter than what is left of the currently executing process, SRTF will preempt the current process, while SJF will allow it to finish completion.



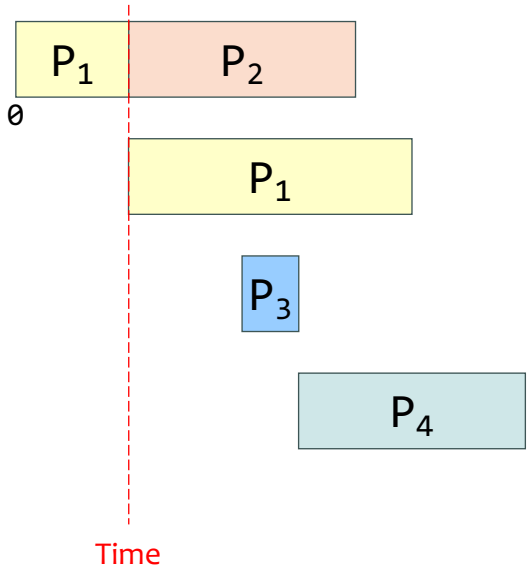
Shortest-Job-First Scheduling (SJF)

- SJF Scheduling can be either **Preemptive** or **Non-Preemptive**.
  - **Preemptive** SJF ⇒ **SRTF**
- The difference lies in the **choice** to be made when a new process arrives at the ready queue while a previous process is still executing.
- If the next CPU burst of the newly arrived process is shorter than what is left of the currently executing process, SRTF will preempt the current process, while SJF will allow it to finish completion.



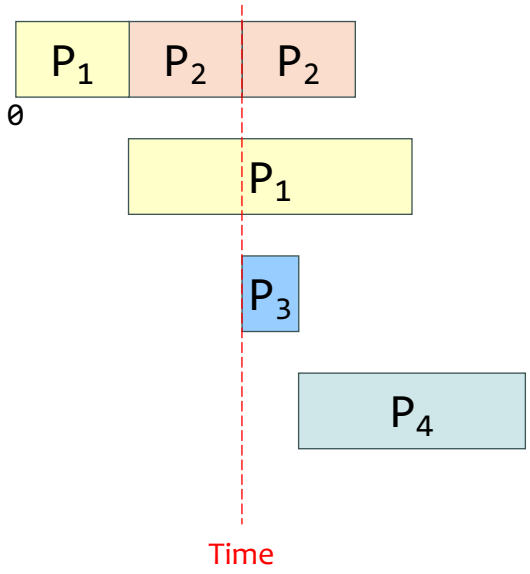
Shortest-Job-First Scheduling (SJF)

- SJF Scheduling can be either **Preemptive** or **Non-Preemptive**.
  - **Preemptive** SJF ⇒ **SRTF**
- The difference lies in the **choice** to be made when a new process arrives at the ready queue while a previous process is still executing.
- If the next CPU burst of the newly arrived process is shorter than what is left of the currently executing process, SRTF will preempt the current process, while SJF will allow it to finish completion.



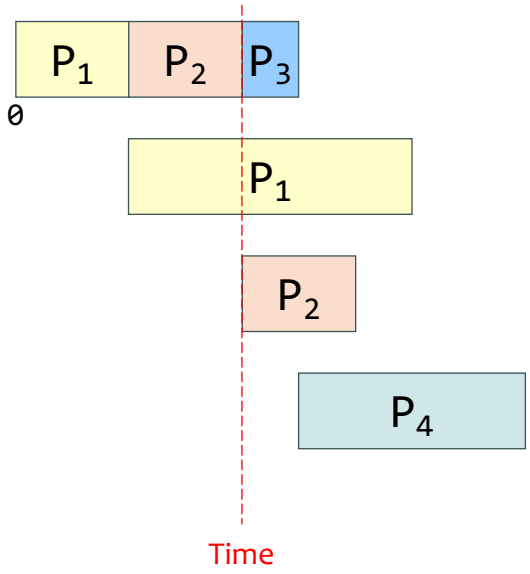
Shortest-Job-First Scheduling (SJF)

- SJF Scheduling can be either **Preemptive** or **Non-Preemptive**.
  - **Preemptive** SJF ⇒ **SRTF**
- The difference lies in the **choice** to be made when a new process arrives at the ready queue while a previous process is still executing.
- If the next CPU burst of the newly arrived process is shorter than what is left of the currently executing process, SRTF will preempt the current process, while SJF will allow it to finish completion.



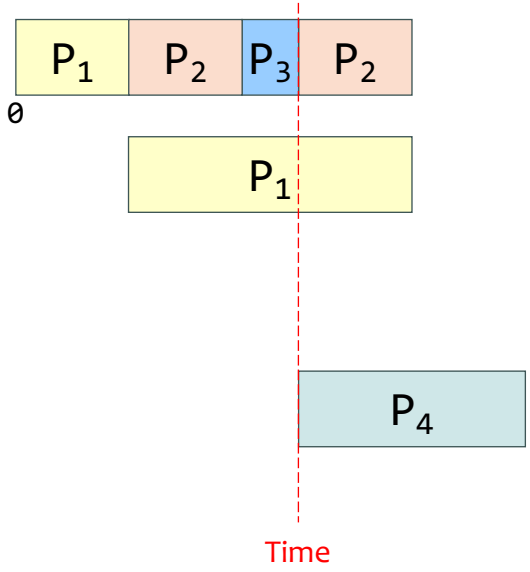
Shortest-Job-First Scheduling (SJF)

- SJF Scheduling can be either **Preemptive** or **Non-Preemptive**.
  - **Preemptive** SJF ⇒ **SRTF**
- The difference lies in the **choice** to be made when a new process arrives at the ready queue while a previous process is still executing.
- If the next CPU burst of the newly arrived process is shorter than what is left of the currently executing process, SRTF will preempt the current process, while SJF will allow it to finish completion.



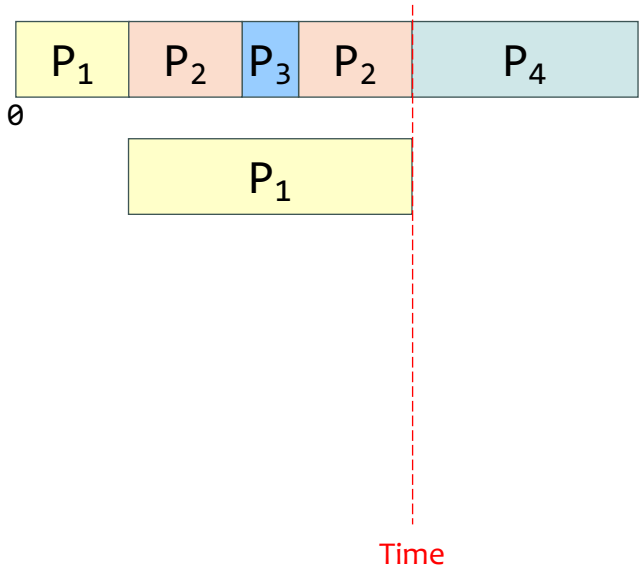
Shortest-Job-First Scheduling (SJF)

- SJF Scheduling can be either **Preemptive** or **Non-Preemptive**.
  - **Preemptive** SJF ⇒ **SRTF**
- The difference lies in the **choice** to be made when a new process arrives at the ready queue while a previous process is still executing.
- If the next CPU burst of the newly arrived process is shorter than what is left of the currently executing process, SRTF will preempt the current process, while SJF will allow it to finish completion.



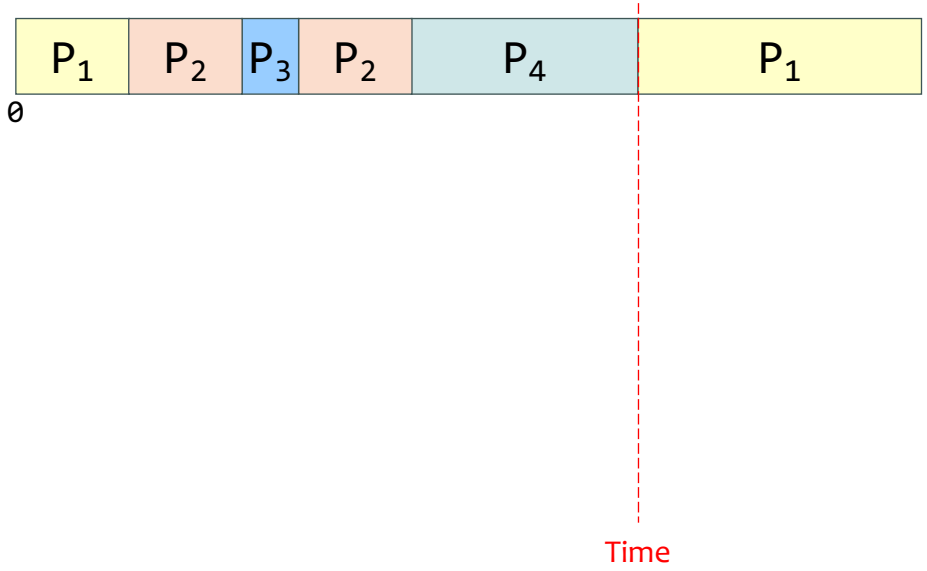
Shortest-Job-First Scheduling (SJF)

- SJF Scheduling can be either **Preemptive** or **Non-Preemptive**.
  - **Preemptive** SJF ⇒ **SRTF**
- The difference lies in the **choice** to be made when a new process arrives at the ready queue while a previous process is still executing.
- If the next CPU burst of the newly arrived process is shorter than what is left of the currently executing process, SRTF will preempt the current process, while SJF will allow it to finish completion.



Shortest-Job-First Scheduling (SJF)

- SJF Scheduling can be either **Preemptive** or **Non-Preemptive**.
  - **Preemptive** SJF ⇒ **SRTF**
- The difference lies in the **choice** to be made when a new process arrives at the ready queue while a previous process is still executing.
- If the next CPU burst of the newly arrived process is shorter than what is left of the currently executing process, SRTF will preempt the current process, while SJF will allow it to finish completion.

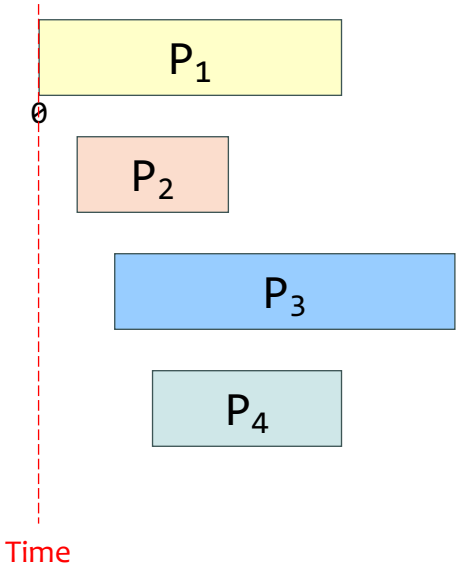


SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- **SJF** Scheduling Gantt chart

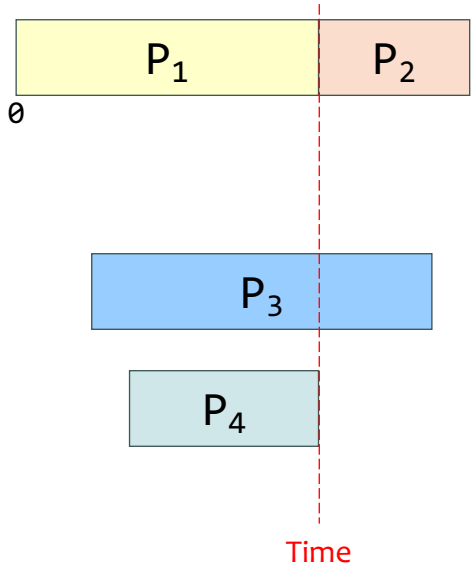


SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- **SJF** Scheduling Gantt chart

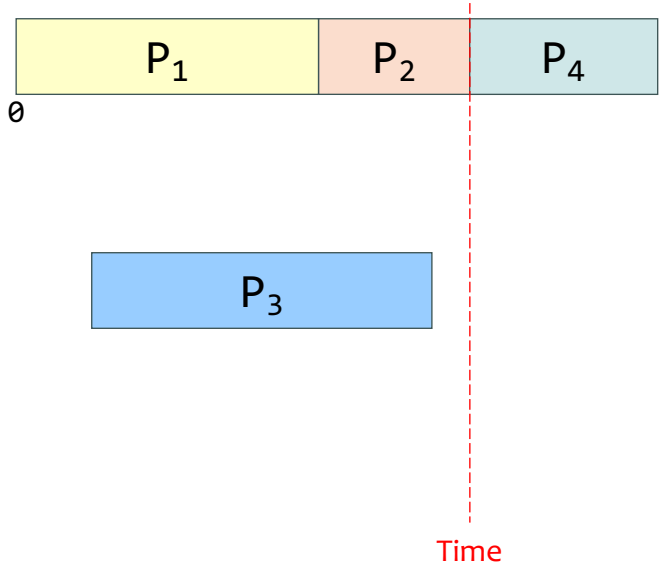


SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- **SJF** Scheduling Gantt chart

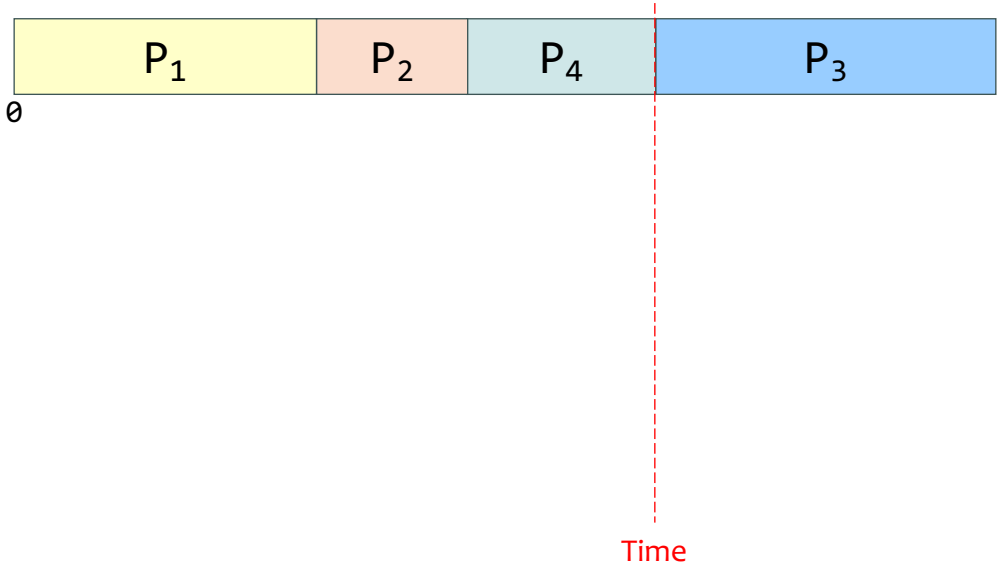


SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- **SJF** Scheduling Gantt chart

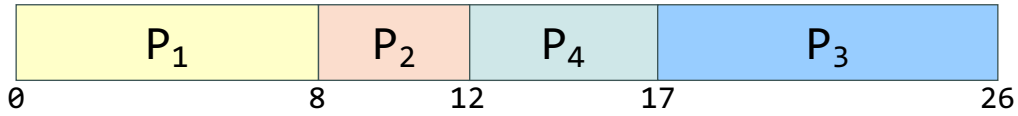


SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- **SJF** Scheduling Gantt chart



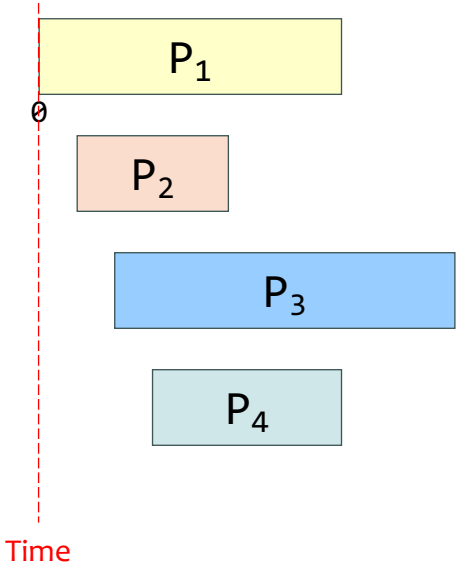
- Average Waiting Time:
  - `[(0-0) + (8-1) + (17-2) + (12-3)] / 4 = 7.75`
- Average Turnaround Time:
  - `[(8-0) + (12-1) + (26-2) + (17-3)] / 4 = 14.25`

SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- SRTF Scheduling Gantt chart

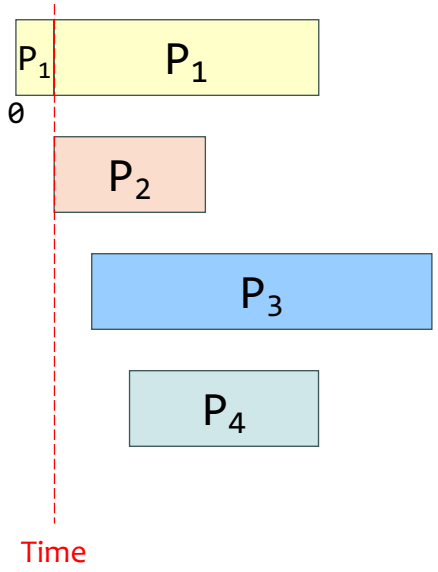


SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- SRTF Scheduling Gantt chart



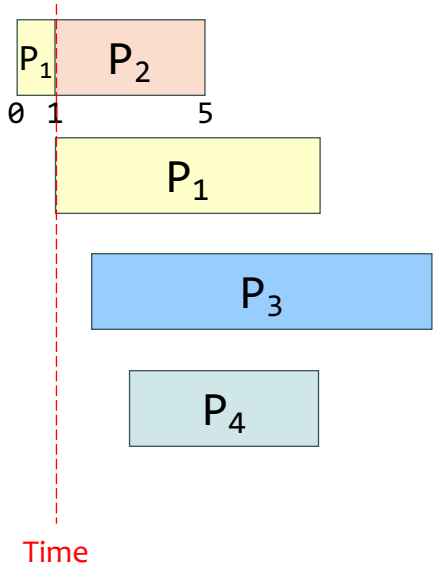
Scheduling Algorithms

SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- SRTF Scheduling Gantt chart

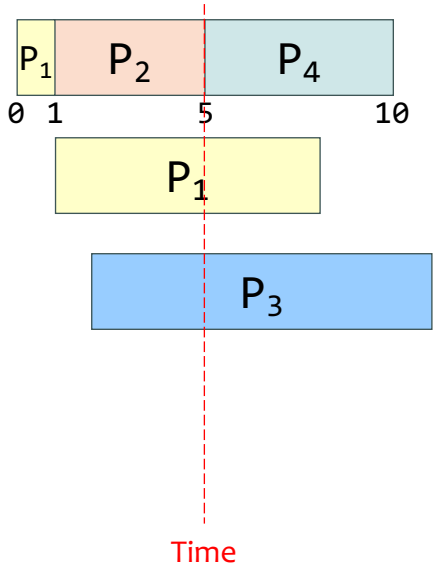


SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- SRTF Scheduling Gantt chart

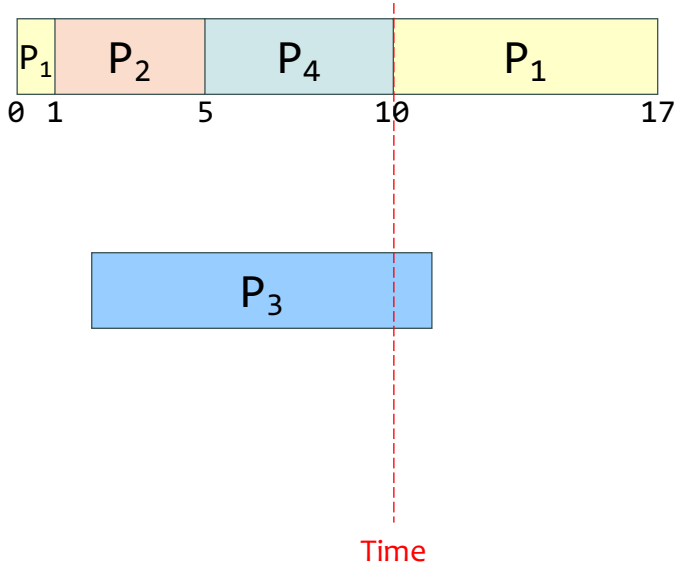


SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- SRTF Scheduling Gantt chart

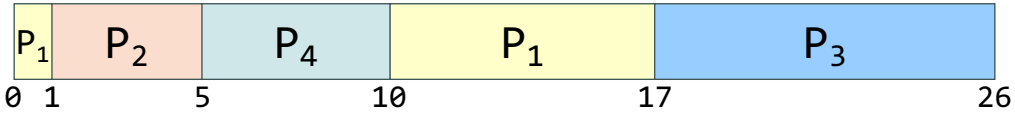


SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- SRTF Scheduling Gantt chart



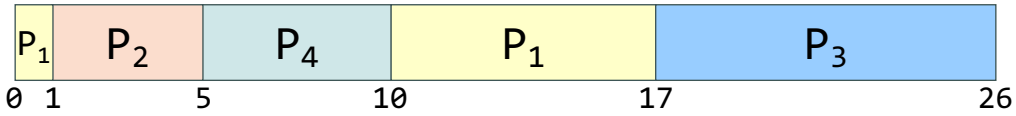
Timeline	Arrival	Remaining Time				Scheduling	Ending
		P_1	P_2	P_3	P_4		
0	P_1	8	-	-	-	P_1	
1	P_2	7	4	-	-	P_2	
2	P_3	7	3	9	-	P_2	
3	P_4	7	2	9	5	P_2	
5		7	-	9	5	P_4	P_2
10		7	-	9	-	P_1	P_4
17		-	-	9	-	P_3	P_1
26		-	-	-	-	-	P_3

SJF (Non-Preemptive) vs SRTF (Preemptive)

- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- SRTF Scheduling Gantt chart



- Average Waiting Time:
  - `[(10-1) + (1-1) + (17-2) + (5-3)] / 4 = 6.5`
- Average Turnaround Time:
  - `[(17-0) + (5-1) + (26-2) + (10-3)] / 4 = 13`

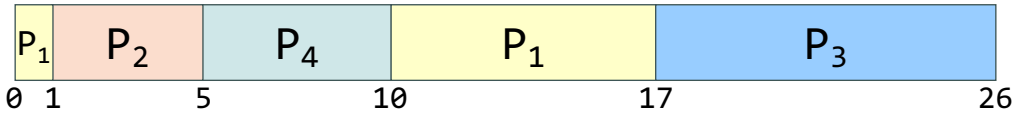
Scheduling Algorithms

SJF (Non-Preemptive) vs SRTF (Preemptive)

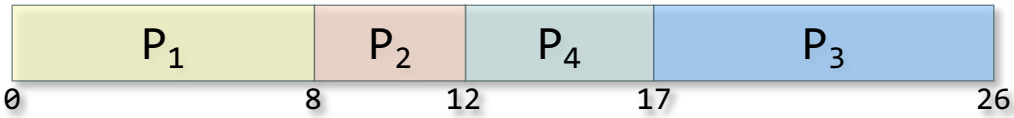
- Example:

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- SRTF Scheduling Gantt chart



- Average Waiting Time:
  - `[(10-1) + (1-1) + (17-2) + (5-3)] / 4 = 6.5`



- Average Turnaround Time:
  - `[(17-0) + (5-1) + (26-2) + (10-3)] / 4 = 13`

`[(0-0)+(8-1)+(17-2)+(12-3)]/4 = 7.75`

`[(8-0)+(12-1)+(26-2)+(17-3)]/4 = 14.25`

Priority Scheduling

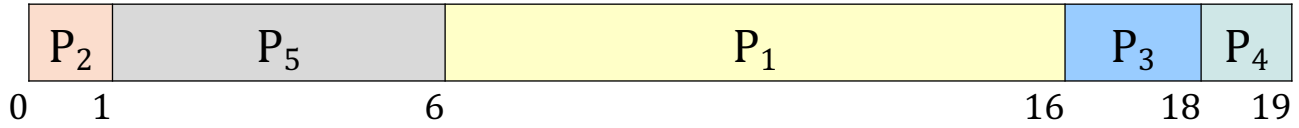
- A priority number is associated with each process.
- The CPU is allocated to the process with the highest priority
  - **Convention:** smaller integer value ⇒ higher priority
- **SJF** is a special case of the general **Priority Scheduling** algorithm.
  - **priority** == the **predicted** next CPU burst time
- Priorities can be defined either internally or externally:
  - Internally defined priorities use some measurable quantities to compute the priority of a process
    - E.g., time limits, memory requirements, the number of open files, the ratio of average I/O bursts to average CPU bursts, etc.
  - Externally defined priorities are set by criteria outside the OS, such as the importance of the process, the type and amount of funds being paid for computer use, the department sponsoring the work, and other, often political, factors.

Priority Scheduling

- Example of Priority Scheduling. Assume all processes arrive at time 0.

Process	CPU 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:



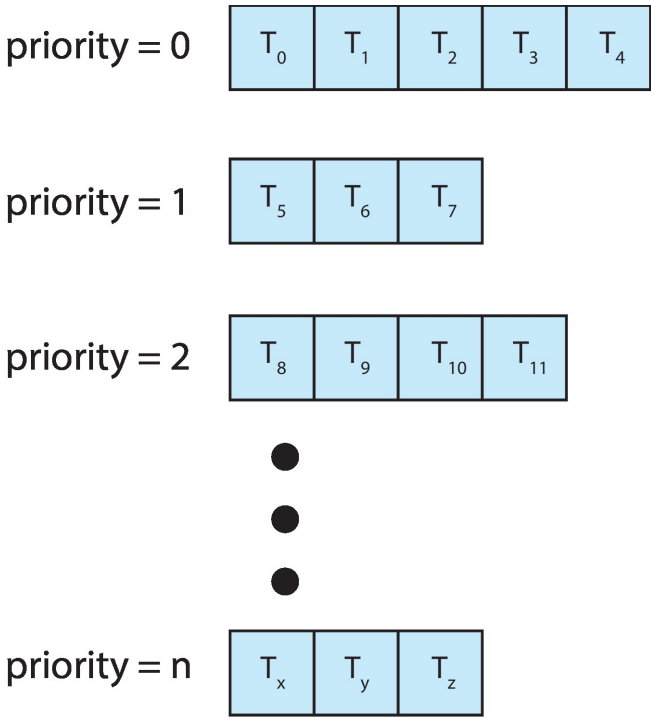
- Average Waiting Time:
  - (6 + 0 + 16 + 18 + 1) / 5 = 8.2

Priority Scheduling

- Priority scheduling can be either **Preemptive** or **Non-Preemptive**.
  - When a process arrives at the ready queue, its priority is compared with the priority of the currently running process. Preemption would occur if the priority of the new process is higher than the currently running process.
  - In comparison, a Non-Preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue, and continue executing the currently running process til completion.
- A major problem with Priority Scheduling: **Starvation**
  - Also known as indefinite blocking.
  - Low-priority processes might wait indefinitely until other high-priority processes are completed.
  - **Solution:**
    - **Aging**: gradually **increasing** the **priority** of processes that wait in the system for a long time.

Priority Scheduling + Round-Robin

- Run the process with the highest priority. Processes with the **same priority** run Round-Robin.
  - All processes in a single queue
  - Time to determine the highest-priority process: $O(n)$
    - assume no advanced data structures being used.
  - Can we do better than $O(n)$?
    - Multilevel Queue Scheduling
    - Time complexity *can* be $O(P)$
      - $P \Rightarrow$ number of priority levels
      - much better than $O(n)$



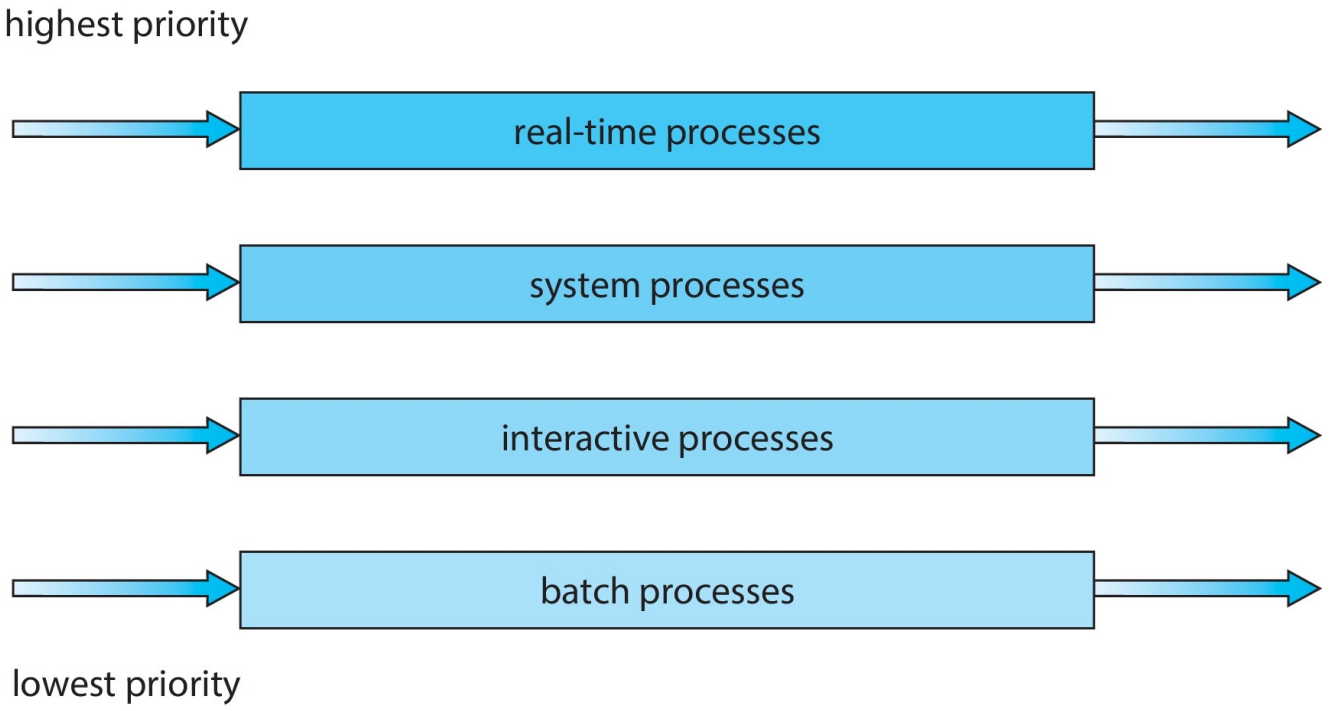
Scheduling Algorithms

Multilevel Queue Scheduling

- Multiple queues, each with different proirity
  - Schedule the process in the highest-priority queue first
- Multilevel queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine which queue a process will enter when that process needs service
  - Scheduling among the queues

Multilevel Queue Scheduling

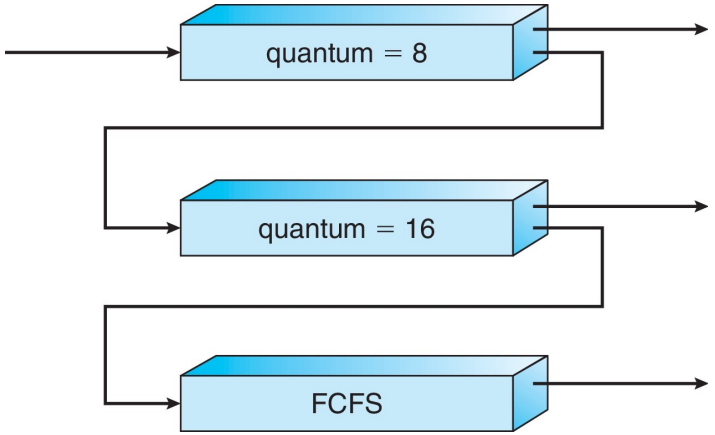
- Prioritization based upon process type



- **Multilevel Feedback Queue Scheduling**
  - Multilevel Queue Scheduling Upgraded
    - a process can move between various queues
  - 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
  - Aging can be implemented using multilevel feedback queue

Multilevel Feedback Queue Scheduling Example

- Three queues:
  - Q_0 – RR with time quantum 8ms
  - Q_1 – RR with time quantum 16ms
  - Q_2 – FCFS



- Scheduling
  - A new process enters Q_0 which is served by **RR**
    - When it gains CPU, the process receives 8ms
    - If it does not finish in 8ms, the process is moved to Q_1
  - At Q_1, job is again served in **RR** and receives additional 16ms
    - If it still does not complete, it is preempted and moved to Q_2

Thank you!