Operating Systems
CS 415

Lecture 8:
Scheduling

Suyash Gupta
Assistant Professor
Distopia Labs and ONRG
Dept. of Computer Science
(E) suyash@uoregon.edu
(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Announcements

- **Suyash Gupta**
  - Office Hours: Thursday, 10-11am, Deschutes 334

- **Nihal Balivada (TA)**
  - Office Hours: Wednesday/ Friday, 11-12pm, Deschutes 335

- **Ranjitha Rani (TA)**
  - Office Hours: Monday /Tuesday, 1-2pm, Deschutes 229

Assignment 2 is out!

- **Deadline → May 12, 2026 at 11:59pm PST**
- Please start working and talk to us if you are stuck.

- **Final → June 10, 2026 at 12:30pm PST, STB 145**
  - Closed book, no cheat sheets, no discussions.

Last Class

- Threads (Chapter 4)
- Next, we move to Chapter 5!

Who gets what?

- In a multiprocessing system, OS shares resources among running processes.
- Which process gets access to which resources?

Resource Types?

Resource Types?

- **Memory:**
  - Allocate everyone a subset of available memory
  - Physical resources are limited.
  - Virtual memory tries to make this appear infinite.
- **I/O:**
  - Allocate which I/O and how much time can be spent.
- **CPU:**
  - Allocate time slot to use.
  - A time slot to run instructions.
- We will focus on CPU resource allocation for now.

Short Term Scheduling

Controls how the CPU is assigned to processes

How do we know OS is doing Optimal Scheduling?

Scheduling Metrics

- **Utilization/efficiency**
  - Keep the CPU busy 100% of the time with useful work.
- **Throughput**
  - Maximize the number of jobs processed per hour.
- **Latency**
  - From the time of submission to the time of completion.

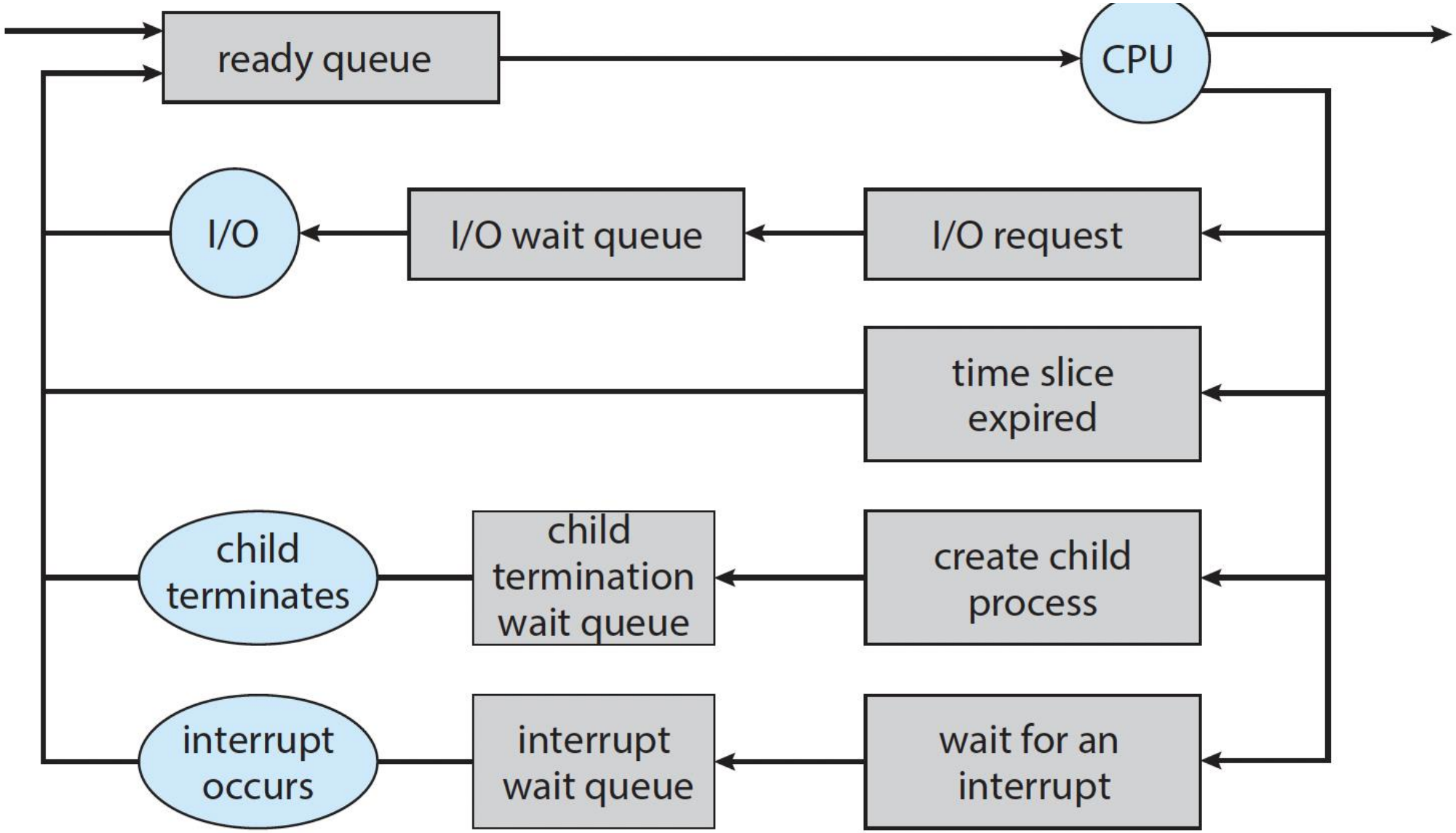
Scheduling Metrics

- **Waiting time**
  - Sum of time spent (in ready queue) waiting to be scheduled on the CPU.
- **Response time**
  - Time from submission until the first response is produced (mainly for interactive jobs).
- **Fairness**
  - Make sure each process gets a fair share of the CPU.

Desired Scheduling

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

Process Scheduling Occurs When?



Dispatcher vs. Scheduler

Dispatcher vs. Scheduler

- **Scheduler** → Decides which process or thread should run next.

- **Dispatcher** →
  - Performs the actual context switch to run the process chosen by the scheduler.
  - Dispatcher module gives control of the CPU to the process selected by the short-term scheduler.

- **Dispatch latency** → Time it takes for the dispatcher to stop one process and start another running.
  - Context switch time

What if a process is running for too long?

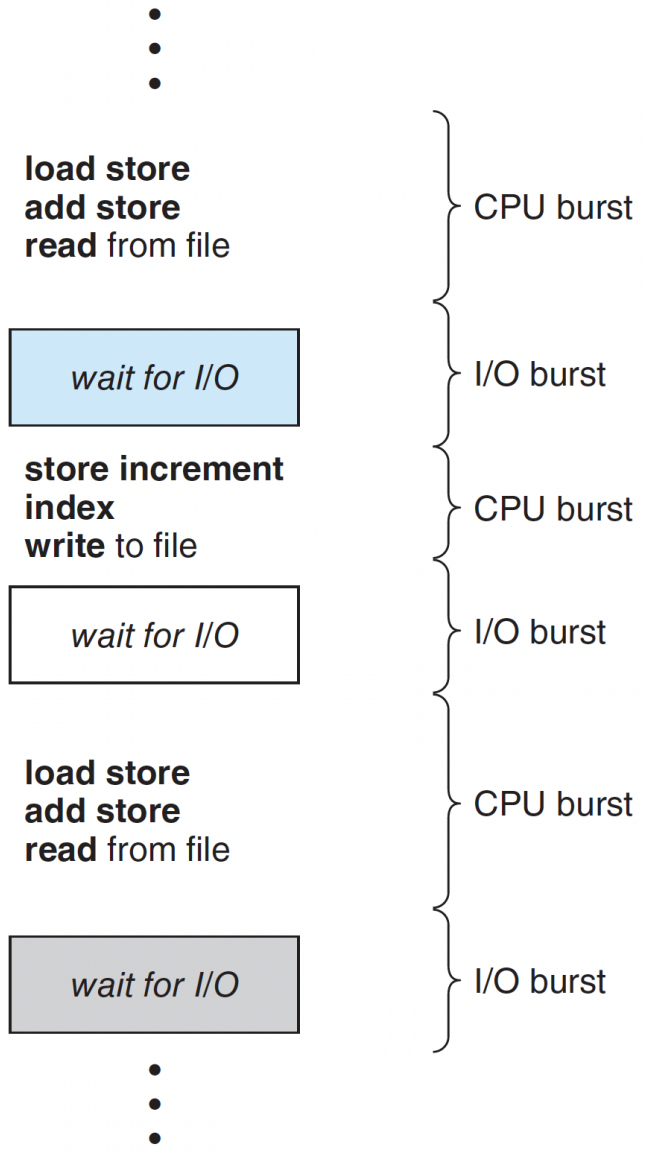
Preemption

- **Preemptive Scheduler**
  - Permits rescheduling a process that is actively running (i.e., preempt its execution).

- **Non-Preemptive Scheduler**
  - Does not support Preemption!

Process Behavior

- **CPU burst** → run instructions.
- **I/O burst** → initiates and waits for I/O.
- Scheduling is aided by knowing the length of these bursts.



Scheduling Algorithms

First Come First Served (FCFS)

First Come First Served (FCFS)

- Serve the jobs in the order they arrive
- Non-preemptive
  - Process runs until it has to wait or terminates
  - OS can NOT stop the process and put it in ready queue
- Simple and easy to implement
  - When a process is ready, add it to tail of ready queue, and serve the ready queue in FCFS order.
- Very fair
  - No process is starved out
  - Service order is immune to job size (does not depend)
  - It depends only on time of arrival

First Come First Served (FCFS)

Assume the processes arrive at the same time (e.g., time 0) but in the order: P1, P2, P3.

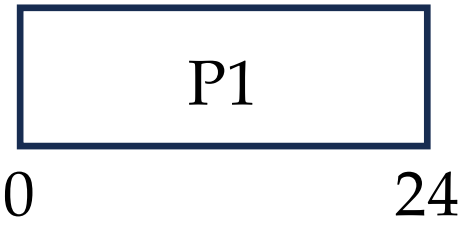
Process	Burst Time
P1	24
P2	3
P3	3

First Come First Served (FCFS)

Assume the processes arrive at the same time (e.g., time 0) but in the order: P1, P2, P3.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for the Schedule

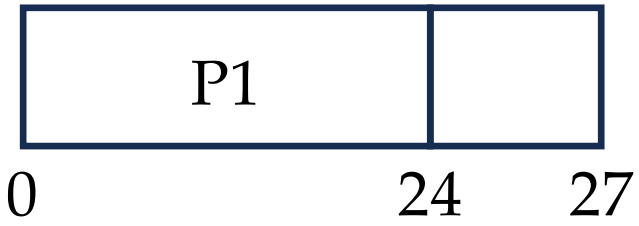


First Come First Served (FCFS)

Assume the processes arrive at the same time (e.g., time 0) but in the order: P1, P2, P3.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for the Schedule

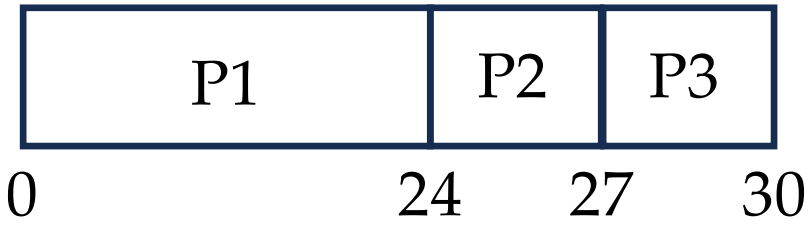


First Come First Served (FCFS)

Assume the processes arrive at the same time (e.g., time 0) but in the order: P1, P2, P3.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for the Schedule



Waiting time for P1 = 0; P2 = 24; P3 = 27
Average waiting time: (0 + 24 + 27)/3 = 17

How to reduce waiting time in FCFS?

- Convoy effect!
  - Short processes gets placed behind long process in the scheduling order.

How to reduce waiting time in FCFS?

Suppose processes arrive in a different order:
P2, P3, P1 (again at time 0).

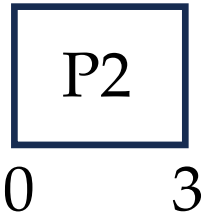
Process	Burst Time
P2	3
P3	3
P1	24

How to reduce waiting time in FCFS?

Suppose processes arrive in a different order:
P2, P3, P1 (again at time 0).

Process	Burst Time
P2	3
P3	3
P1	24

Gantt Chart for the Schedule

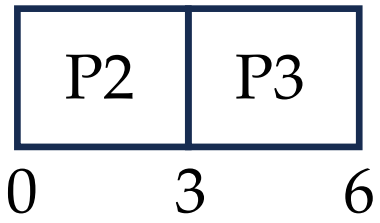


How to reduce waiting time in FCFS?

Suppose processes arrive in a different order:
P2, P3, P1 (again at time 0).

Process	Burst Time
P2	3
P3	3
P1	24

Gantt Chart for the Schedule

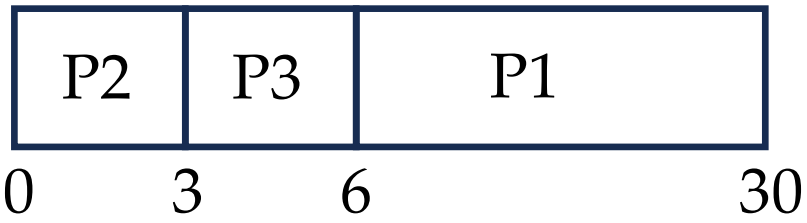


How to reduce waiting time in FCFS?

Suppose processes arrive in a different order:
P2, P3, P1 (again at time 0).

Process	Burst Time
P2	3
P3	3
P1	24

Gantt Chart for the Schedule



Waiting time for P1 = 6; P2 = 0; P3 = 3
Average waiting time: (6 + 0 + 3)/3 = 3

Any other Scheduling Algorithm?

Shortest Job First (SJF)

- Suppose we know length of next CPU burst
  - Then use these lengths to schedule the process.
  - Process with the shortest next CPU burst time goes first.

- Non-preemptive → cannot be preempted until it completes its CPU burst.

- SJF is optimal with respect to waiting time
  - Gives minimum average waiting time for a set of processes.

Shortest Job First (SJF)

Shortest Job First (SJF)

Suppose we have processes P1, P2, P3 arriving at time 0.

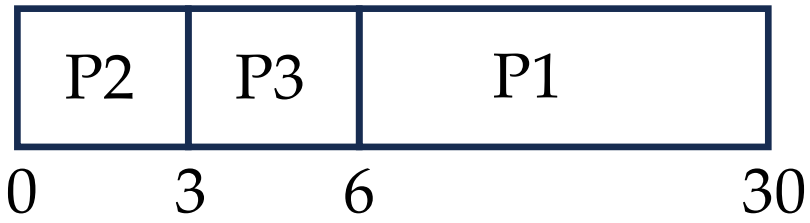
Process	Burst Time
P1	24
P2	3
P3	3

Shortest Job First (SJF)

Suppose we have processes P1, P2, P3 arriving at time 0.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for the Schedule



Waiting time for P1 = 6; P2 = 0; P3 = 3
Average waiting time: (6 + 0 + 3)/3 = 3

Shortest Job First (SJF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

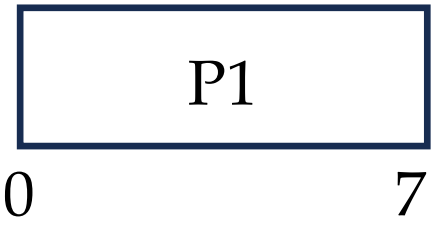
Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Shortest Job First (SJF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule

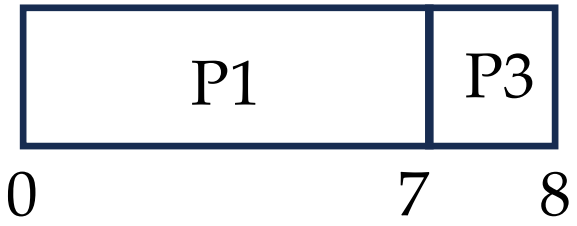


Shortest Job First (SJF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule

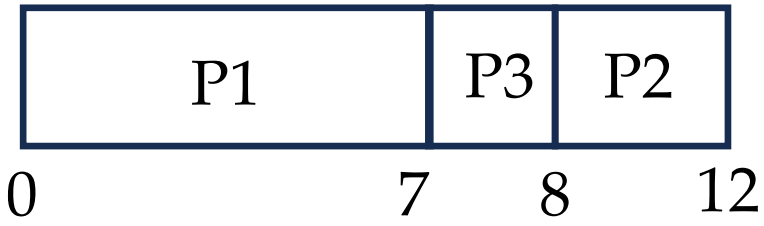


Shortest Job First (SJF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule

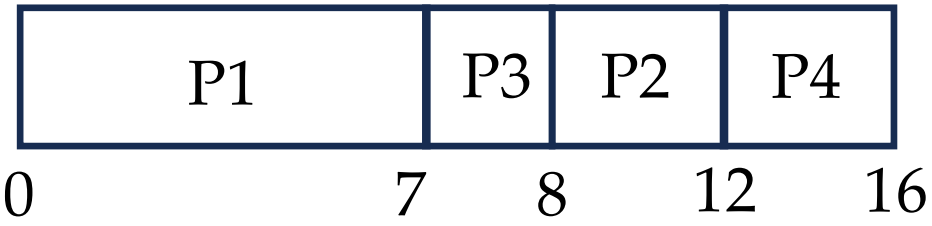


Shortest Job First (SJF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule



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

Any other Scheduling Algorithm?

Shortest Remaining Time First (SRTF)

- How about we allow SJF scheduler to be preemptive.
- Scheduler makes a decision at any time using preemption to stop the currently running process and context switch to another process.
- Shortest Remaining Time First.

Shortest Remaining Time First (SRTF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

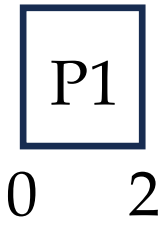
Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Shortest Remaining Time First (SRTF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule

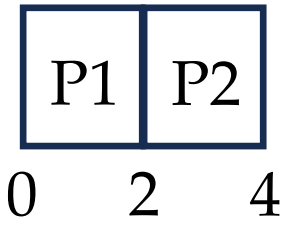


Shortest Remaining Time First (SRTF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule

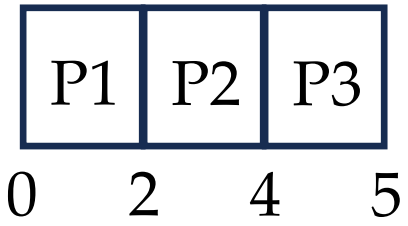


Shortest Remaining Time First (SRTF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule

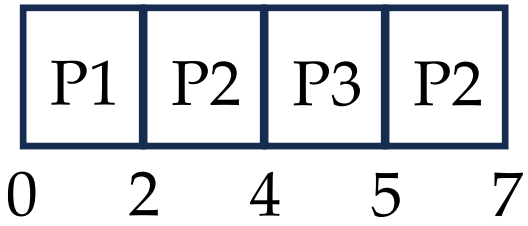


Shortest Remaining Time First (SRTF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule

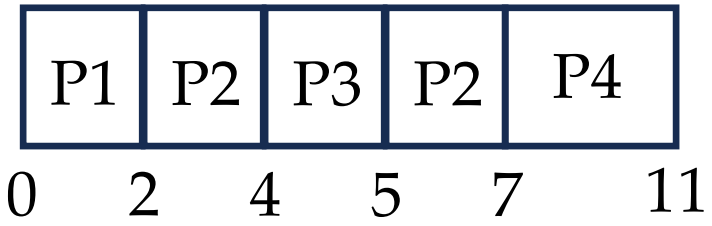


Shortest Remaining Time First (SRTF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule

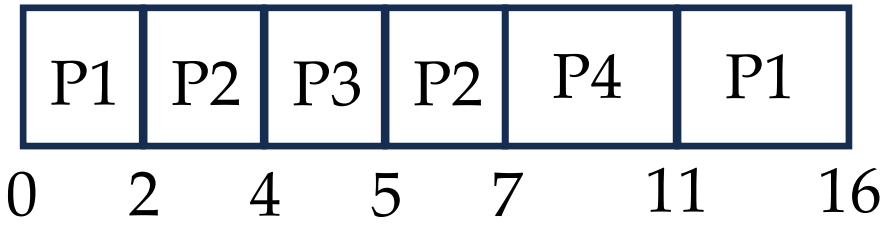


Shortest Remaining Time First (SRTF)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for the Schedule



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

What if processes have priorities?

Priority Scheduling

- Each process is given a **priority value**.

- Always schedule the process with highest priority.
  - Preemptive
  - Non-preemptive

Priority Scheduling (non-preemptive)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

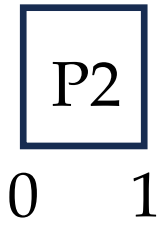
Process	Arrival Time	Burst Time	Priority
P1	0	10	3
P2	0	1	1
P3	0	2	4
P4	0	1	5
P5	0	5	2

Priority Scheduling (non-preemptive)

Suppose we have processes P1, P2, P3, P4, P5.

Process	Arrival Time	Burst Time	Priority
P1	0	10	3
P2	0	1	1
P3	0	2	4
P4	0	1	5
P5	0	5	2

Gantt Chart for the Schedule

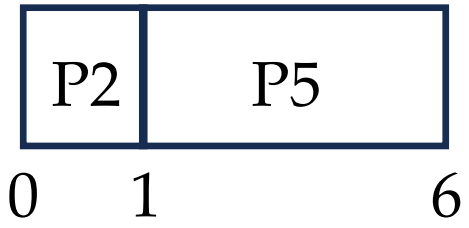


Priority Scheduling (non-preemptive)

Suppose we have processes P1, P2, P3, P4, P5.

Process	Arrival Time	Burst Time	Priority
P1	0	10	3
P2	0	1	1
P3	0	2	4
P4	0	1	5
P5	0	5	2

Gantt Chart for the Schedule

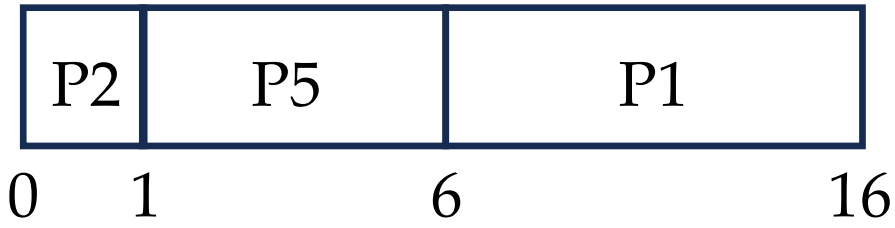


Priority Scheduling (non-preemptive)

Suppose we have processes P1, P2, P3, P4, P5.

Process	Arrival Time	Burst Time	Priority
P1	0	10	3
P2	0	1	1
P3	0	2	4
P4	0	1	5
P5	0	5	2

Gantt Chart for the Schedule

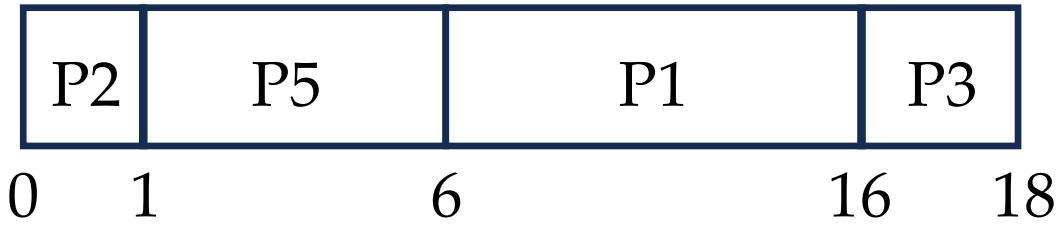


Priority Scheduling (non-preemptive)

Suppose we have processes P1, P2, P3, P4, P5.

Process	Arrival Time	Burst Time	Priority
P1	0	10	3
P2	0	1	1
P3	0	2	4
P4	0	1	5
P5	0	5	2

Gantt Chart for the Schedule

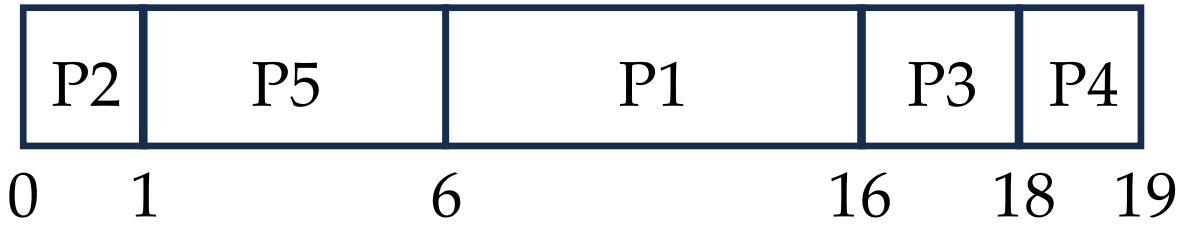


Priority Scheduling (non-preemptive)

Suppose we have processes P1, P2, P3, P4, P5.

Process	Arrival Time	Burst Time	Priority
P1	0	10	3
P2	0	1	1
P3	0	2	4
P4	0	1	5
P5	0	5	2

Gantt Chart for the Schedule



Average waiting time = 8.2

Priority Scheduling (Preemptive)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

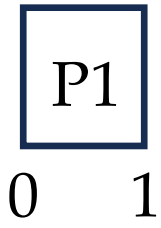
Process	Arrival Time	Burst Time	Priority
P1	0	8	2
P2	1	4	1
P3	2	9	3
P4	3	5	2

Priority Scheduling (Preemptive)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time	Priority
P1	0	8	2
P2	1	4	1
P3	2	9	3
P4	3	5	2

Gantt Chart for the Schedule

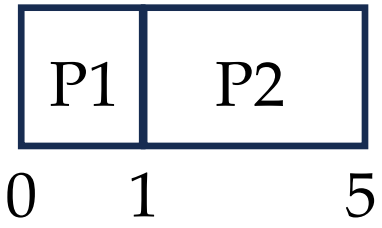


Priority Scheduling (Preemptive)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time	Priority
P1	0	8	2
P2	1	4	1
P3	2	9	3
P4	3	5	2

Gantt Chart for the Schedule

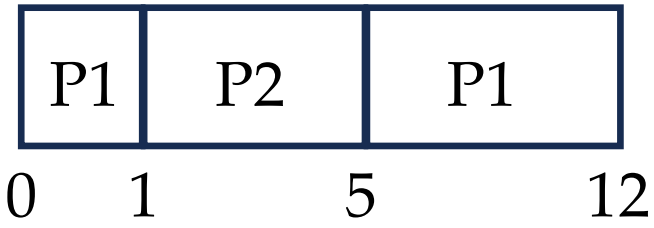


Priority Scheduling (Preemptive)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time	Priority
P1	0	8	2
P2	1	4	1
P3	2	9	3
P4	3	5	2

Gantt Chart for the Schedule

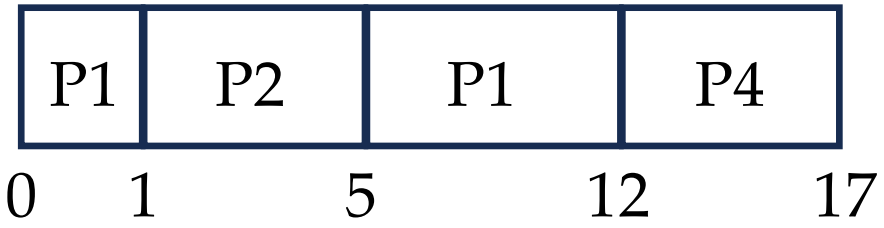


Priority Scheduling (Preemptive)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time	Priority
P1	0	8	2
P2	1	4	1
P3	2	9	3
P4	3	5	2

Gantt Chart for the Schedule

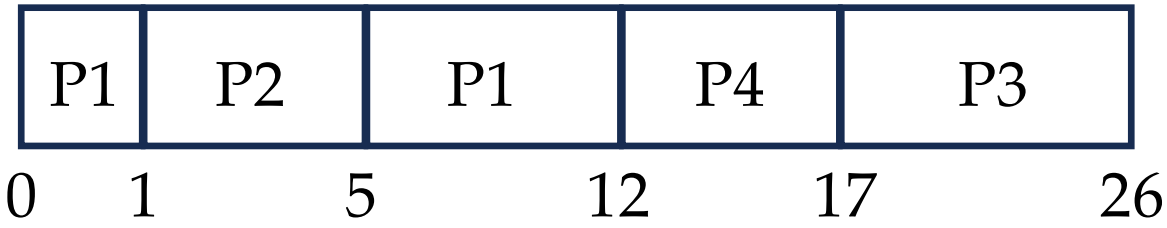


Priority Scheduling (Preemptive)

Suppose we have processes P1, P2, P3, P4 arriving at time different times.

Process	Arrival Time	Burst Time	Priority
P1	0	8	2
P2	1	4	1
P3	2	9	3
P4	3	5	2

Gantt Chart for the Schedule



Limitations of Priority Scheduling

Limitations of Priority Scheduling

- **Starvation**
  - Low priority processes may never execute.

- **Aging** → Addresses Process starvation.
  - Increase priority of processes as as time progresses.

Any other Scheduling Algorithm?

Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum).
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.

- **Algorithm**
  - Consider **n** processes in the ready queue.
  - Consider time quantum is **q**.
  - Then each process works for **q** time before it is preempted.

Round Robin (RR)

Assume the processes arrive at the same time (e.g., time 0) but in the order: P1, P2, P3.

Process	Burst Time
P1	24
P2	3
P3	3

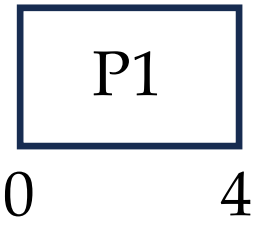
Time Quantum
q=4

Round Robin (RR)

Assume the processes arrive at the same time (e.g., time 0) but in the order: P1, P2, P3.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for the Schedule



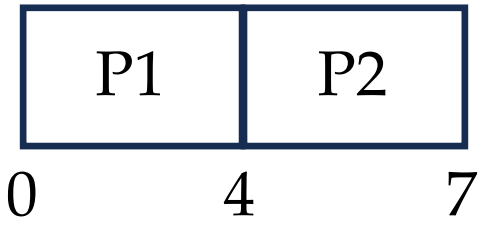
Time Quantum
q=4

Round Robin (RR)

Assume the processes arrive at the same time (e.g., time 0) but in the order: P1, P2, P3.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for the Schedule



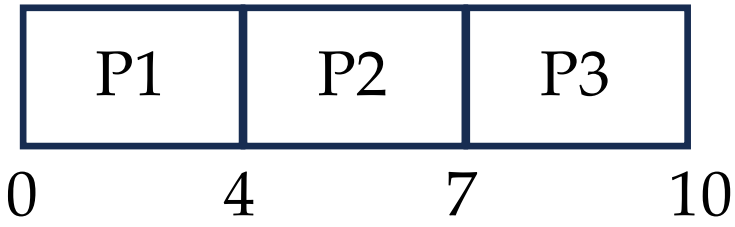
Time Quantum
q=4

Round Robin (RR)

Assume the processes arrive at the same time (e.g., time 0) but in the order: P1, P2, P3.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for the Schedule



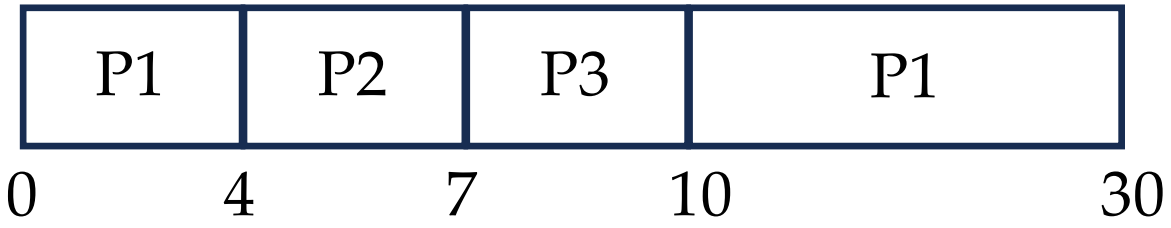
Time Quantum
q=4

Round Robin (RR)

Assume the processes arrive at the same time (e.g., time 0) but in the order: P1, P2, P3.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for the Schedule



Time Quantum
q=4

Limitations of Round Robin

Limitations of Round Robin

- Need to determine Time quantum **q** correctly.

- Smaller time quantums allows the CPU to be frequently shared between the processes.
  - Increases time wasted in context switch.

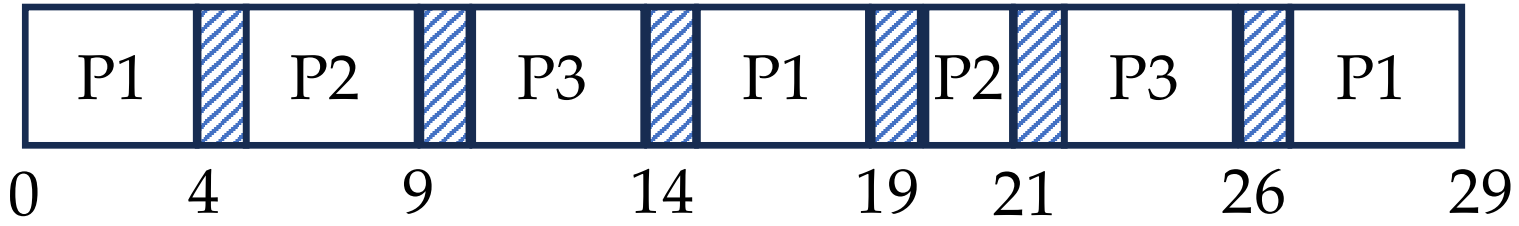
- Larger time quantums will give more preference to processes with larger burst times.
  - Smaller processes need to wait longer before they can be scheduled.

Limitations of Round Robin

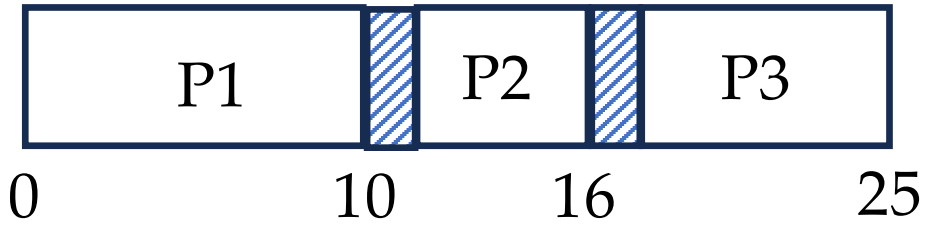
Process	Burst Time
P1	10
P2	5
P3	8

- Assume the processes arrive at the same time in order: P1, P2, P3.
- Context switch costs 1 unit of time

q = 4



q = 10

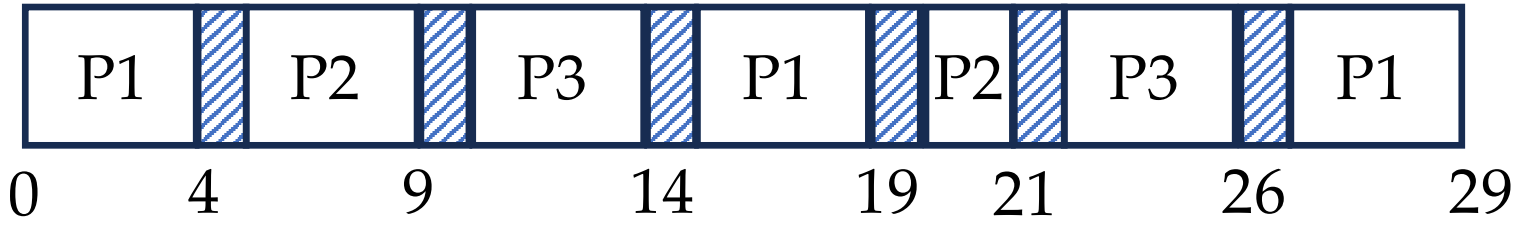


Limitations of Round Robin

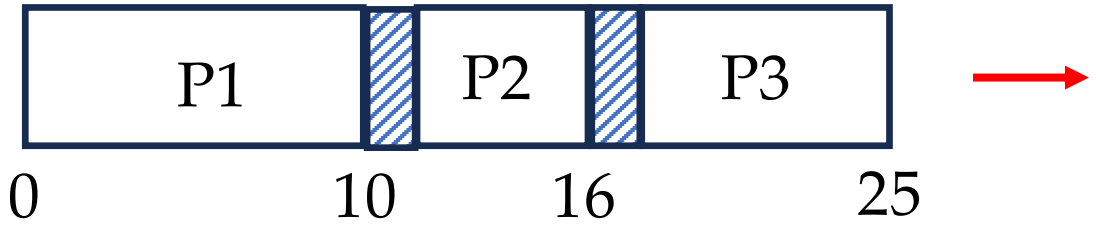
Process	Burst Time
P1	10
P2	5
P3	8

- Assume the processes arrive at the same time in order: P1, P2, P3.
- Context switch costs 1 unit of time

q = 4



q = 10

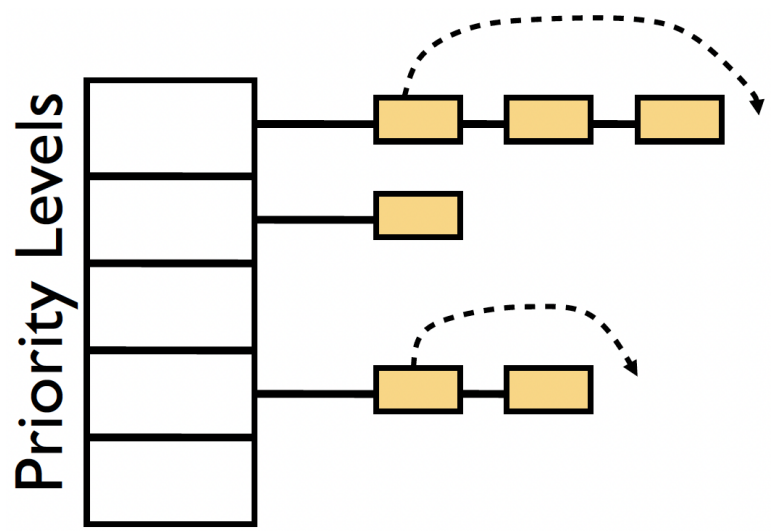


This looks like FCFS!

Round Robin with Priority?

Round Robin with Priority

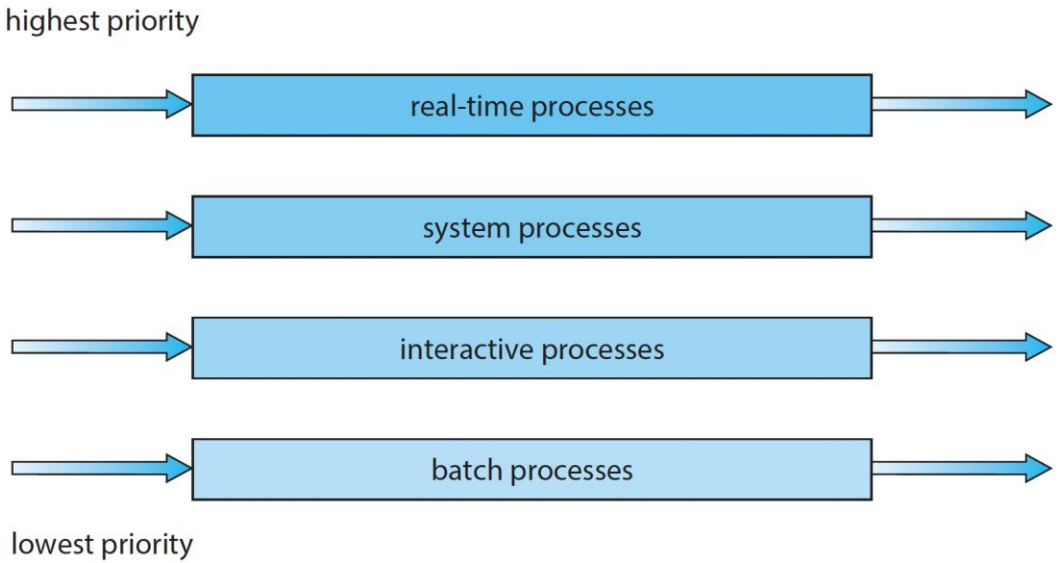
- Have a set of ready queues.
  - One queue for each priority level.
- Always service the queue at the highest priority level.
- Round-robin within each queue.



- **Limitations:**
  - With fixed priorities, processes lower in the priority level can starve.
  - Need to employ aging to increase process priority.

Multilevel Queuing

- Ready queue is partitioned into separate queues.
- Each queue has its own scheduling algorithm:
  - Foreground – RR
  - Background – FCFS
- Scheduling must be done between the queues.
- Fixed priority scheduling
  - Possibility of starvation.
- Time slice
  - Each queue gets a certain amount of CPU time.



Thread Scheduling

Thread Scheduling

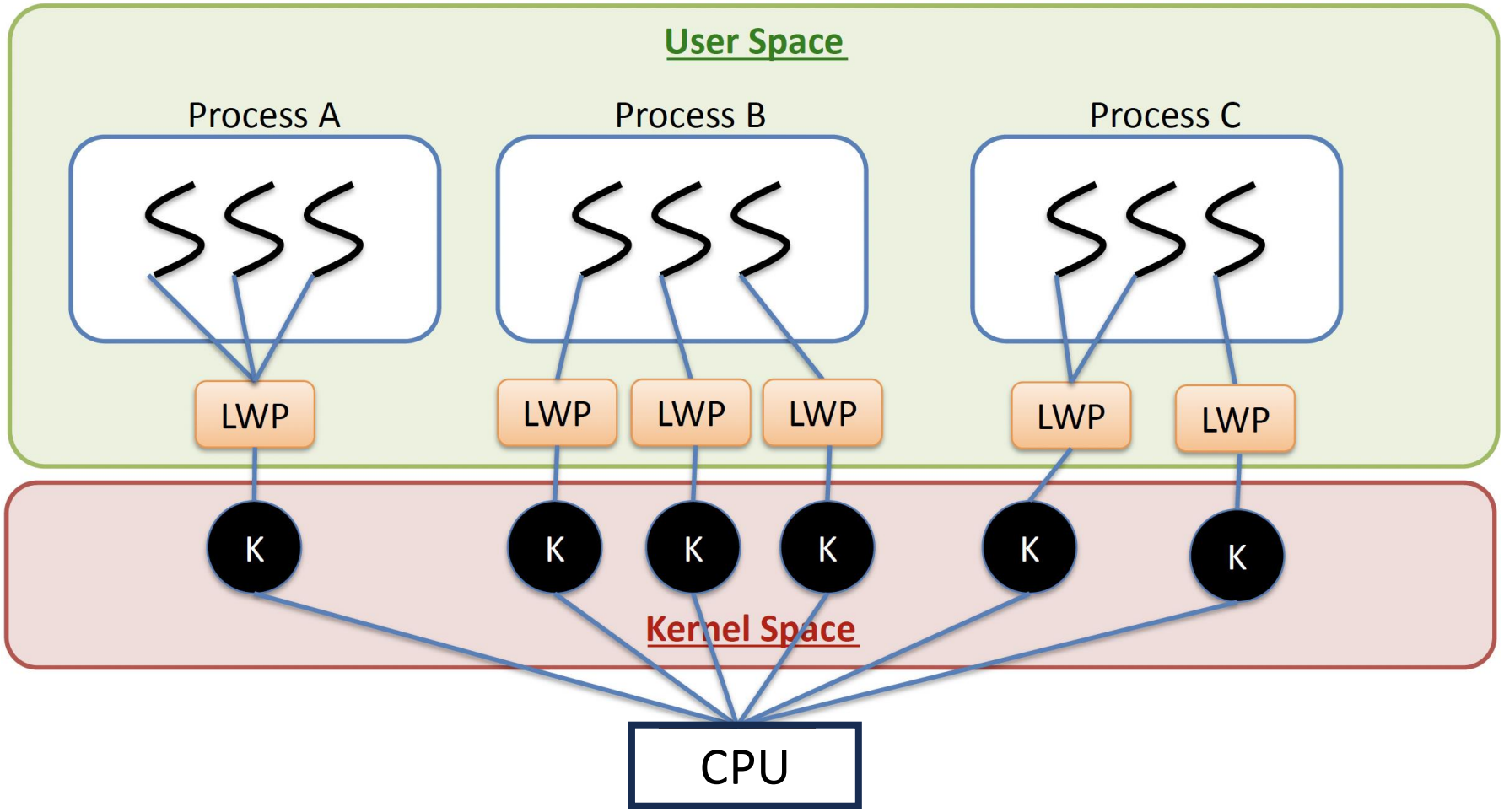
- Scheduling User Threads and Kernel Threads.
  - To run on a CPU, user-level threads must be mapped to an associated kernel-level thread.

- User Thread scheduling managed by Thread libraries.
  - OS is unaware of this!
  - **Process Contention Scope (PCS)** → Competition for being assigned a kernel thread takes place among threads belonging to the same process.

- Kernel Thread scheduling managed by OS.
  - **System-Contention Scope (SCS)** → Which kernel-level thread to schedule onto a CPU.

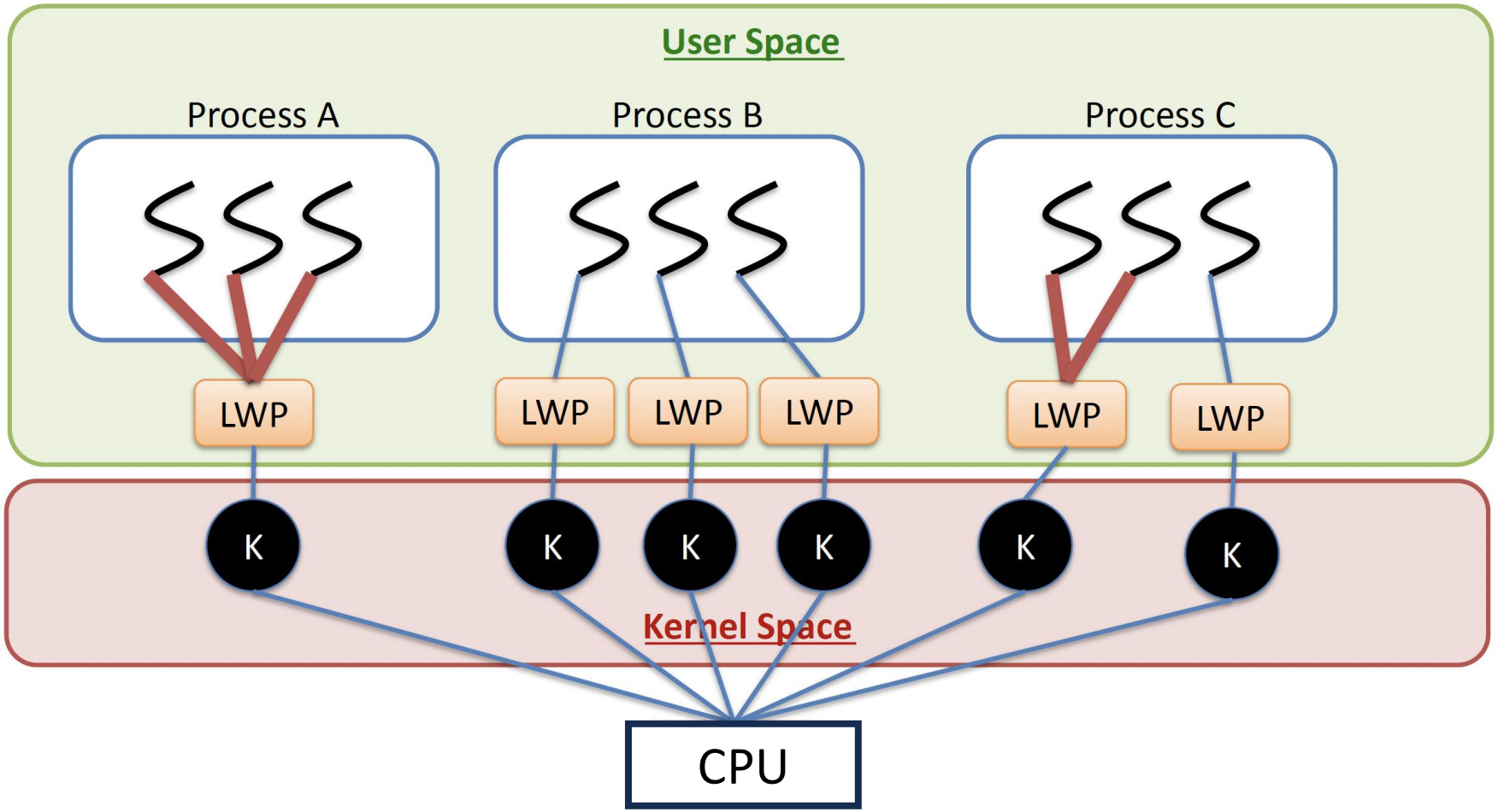
Thread Scheduling

Light-weight process (LWP): Interface between user threads and kernel threads.



Thread Scheduling

Process Contention Scope



Thread Scheduling

System-Contention Scope