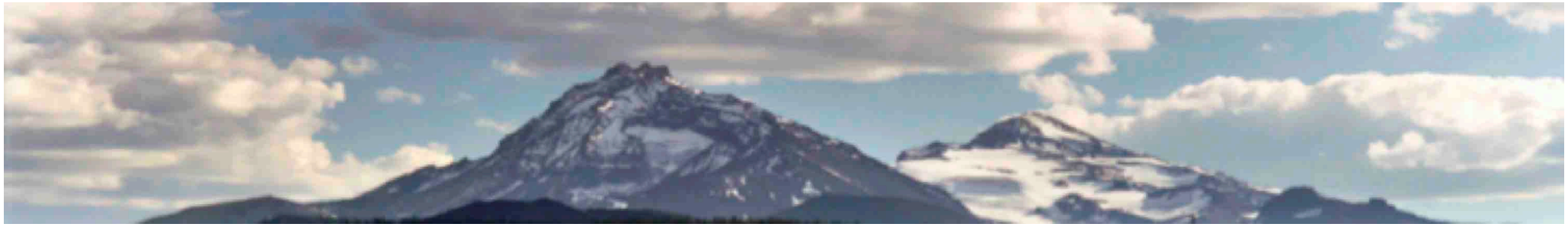


CS 415
Operating Systems
Scheduling

Prof. Jieyang Chen
Department of Computer and Information Science
Fall 2025

UNIVERSITY OF OREGON

Logistics

- Now in Week 5
  - Last topics before midterm
- Project 1 … grading this week
- Project 2 … posted last week
- Read Chapter 5

Outline

- Basic scheduling concepts and criteria
- Scheduling algorithms
- Thread scheduling
- Multiple processor scheduling
- Real-Time CPU scheduling
- Algorithm evaluation

Resource Allocation

- In a multiprogramming / multiprocessing system, OS shares resources among running processes
  - There are different types of OS resources
- Which process gets access to which resources and why?

Resource Types

- *Memory*: Allocate portion of finite resource
  - Physical resources are limited
  - Virtual memory tries to make this appear infinite
- *I/O*: Allocate portion of finite resource and time spent with the resource
  - Store information on disk
  - A time slot to store that information
- *CPU*: Allocate time slot with resource
  - A time slot to run instructions
- We will focus on CPU resource allocation for now

Types of CPU Scheduling

- CPU resource allocation is known as CPU *scheduling*
- *Long-term scheduling*
  - Decides to allow a job to enter the system
- *Medium-term scheduling*
  - Controls the degree of multiprogramming
  - Temporarily swap out processes from memory
- *Short-term scheduling*
  - Controls how the CPU is assigned to processes

← **We focus on this**

CPU Scheduling Goals

- Single process goals
  - GUI (graphical user interface) request
    - click on the mouse (*responsiveness*)
  - Scientific computation
    - long-running, but want to complete ASAP (*time to solution*)
- System goals
  - Get as many tasks done as quickly as possible
    - *throughput* objective
  - Minimize waiting time for processes
    - *response time* objective
  - Get full utilization from the CPU
    - *utilization* objective

Scheduling Criteria (Metrics) - Summary

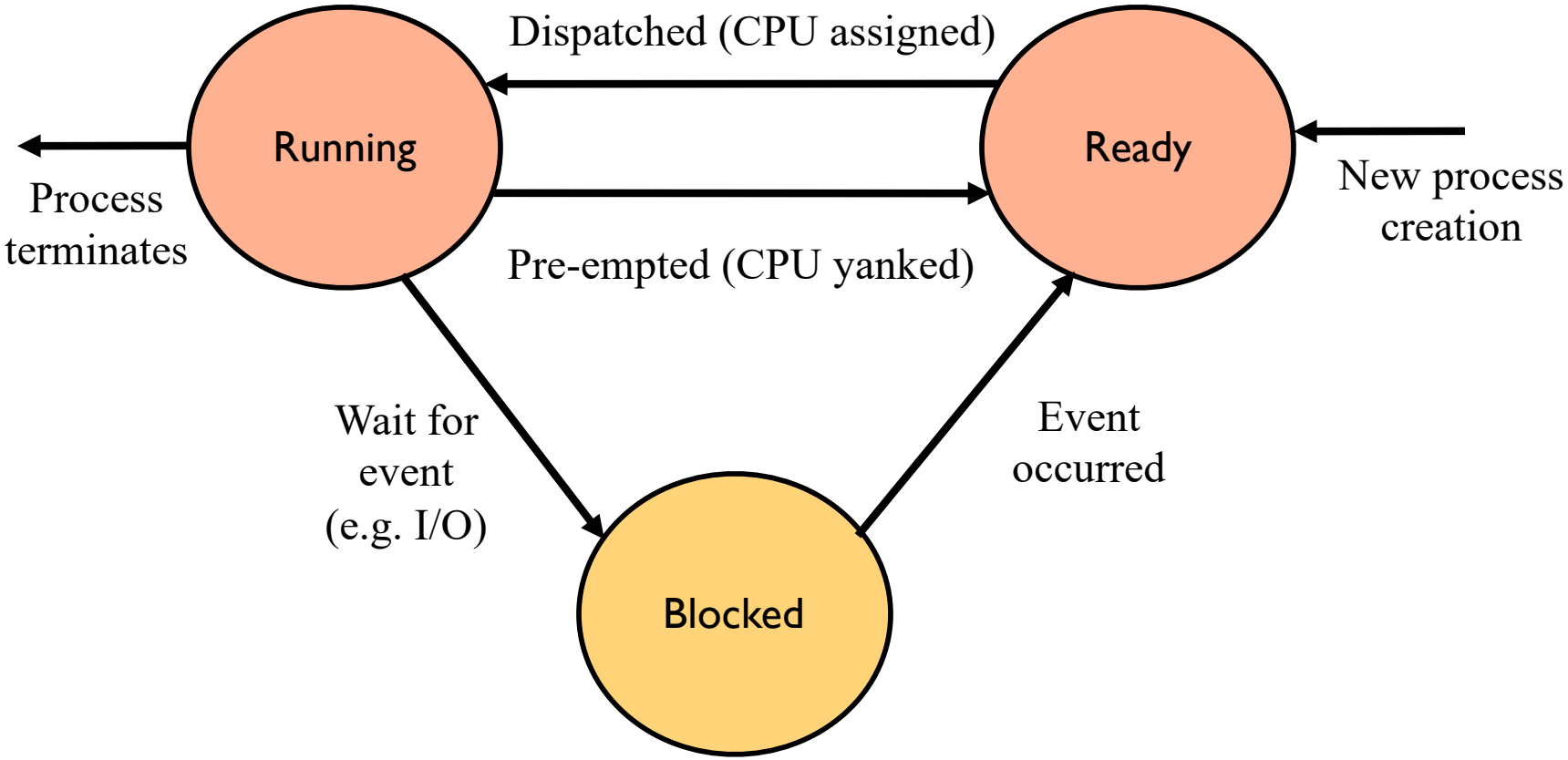
- *Utilization/efficiency*
  - Keep the CPU busy 100% of the time with useful work
- *Throughput*
  - Maximize the number of jobs processed per hour.
- *Turnaround time (latency)*
  - From the time of submission to the time of completion.
- *Waiting time*
  - Sum of time spent (in ready queue) waiting 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

Scheduling Algorithm Optimization Criteria

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

Process Scheduling

- Process transition diagram
- Think about the OS perspective



When does scheduling occur?

- CPU scheduling decisions may take place when:
  1. running → waiting
  2. running → ready
  3. waiting → ready
  4. running → terminates
- Example: Process voluntarily gives up (yields) the CPU
  - What would happen?
  - Who will be the next to run?
- Essentially, scheduling can occur whenever the OS regains control
  - How exactly can this happen?

Scheduling Problem

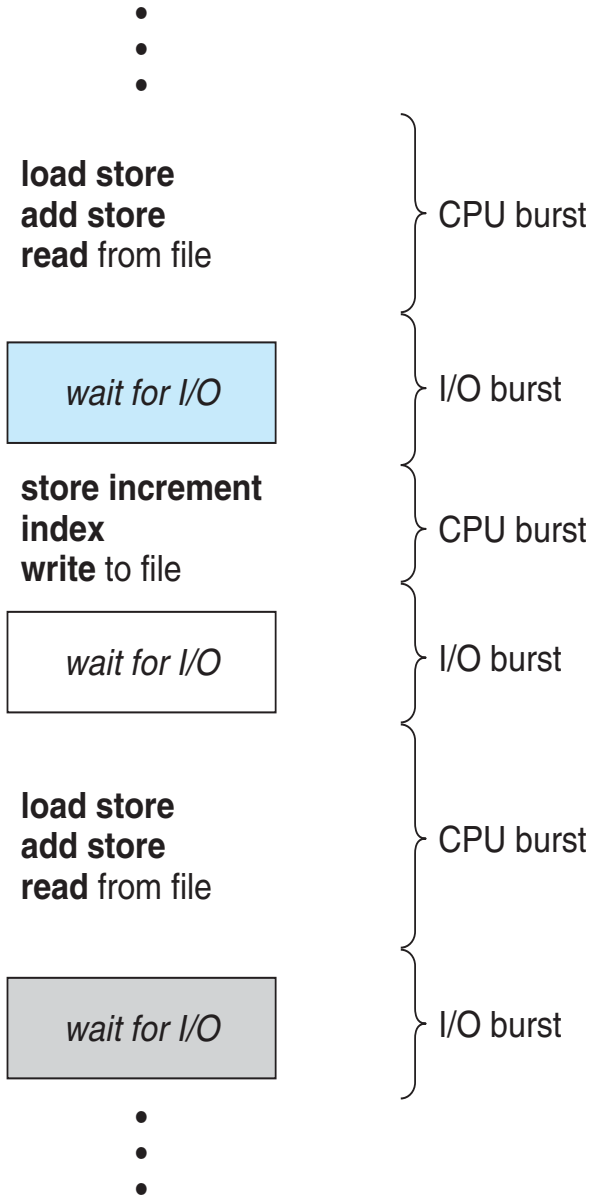
- Always want to choose the "best" process to run
  - How to define "best"?
  - Does the "best" always stay best?
  - What if the "best" has become worse?
  - What if a "better" one enters the system?
  - How to achieve a scheduling goal?

Preemption

- Can we reschedule a process that is actively running (i.e., preempt its execution)?
  - Yes → a *preemptive* scheduler
  - No → a *non-preemptive* scheduler
- Suppose a process becomes ready
  - A new process is created or it is no longer waiting
- Suppose that another process is currently running
- However, it may be "better" to schedule the process we just put on the ready queue
  - Again, what does "better" mean?
  - So, we have to preempt the running process … how?
- In what ways could the new process be better?

Basic Concepts – CPU-I/O Bursts

- *CPU–I/O burst cycle*
  - **CPU burst:** run instructions
  - **I/O burst:** initiates and waits for I/O
- Scheduling is aided by knowing the length of these bursts
  - Hmm, do we know this? … more later



Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the *short-term* scheduler

Scheduler: decides which process or thread should run next
Dispatcher: performs the actual context switch to run the process chosen by the scheduler

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

Scheduling Algorithms

- Some may seem intuitively better than others
- But a lot has to do with the type of offered *workload* to the processor
- Best scheduling comes with best context of the tasks to be completed
  - Knowing something about the workload behavior is important
- Non-preemptive and preemptive algorithm
  - What are the benefits and drawbacks?

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*

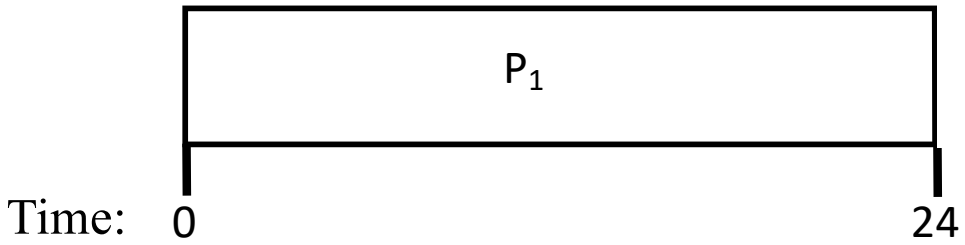
FCFS (Non-preemptive)

Process	Burst Time
P_1	24
P_2	3
P_3	3

Burst time here represents the job's entire execution time

Assume no I/O occurs

- Suppose that the processes arrive in the order: P_1, P_2, P_3
  - Assume processes arrive at the same time (e.g., time 0)
- The *Gantt chart* for the schedule is:



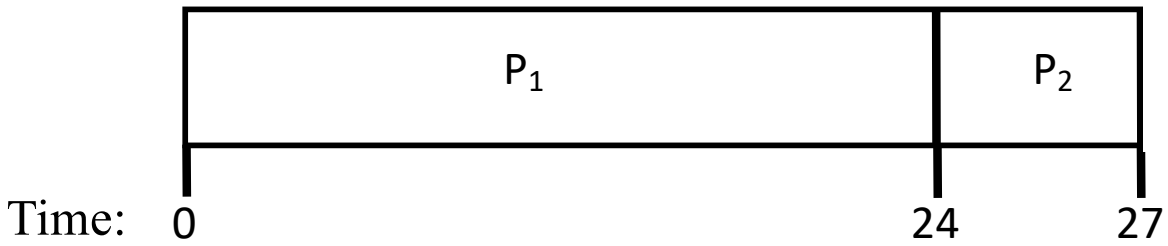
FCFS (Non-preemptive)

Process	Burst Time
P_1	24
P_2	3
P_3	3

Burst time here represents the job's entire execution time

Assume no I/O occurs

- Suppose that the processes arrive in the order: P_1, P_2, P_3
  - Assume processes arrive at the same time (e.g., time 0)
- The *Gantt chart* for the schedule is:



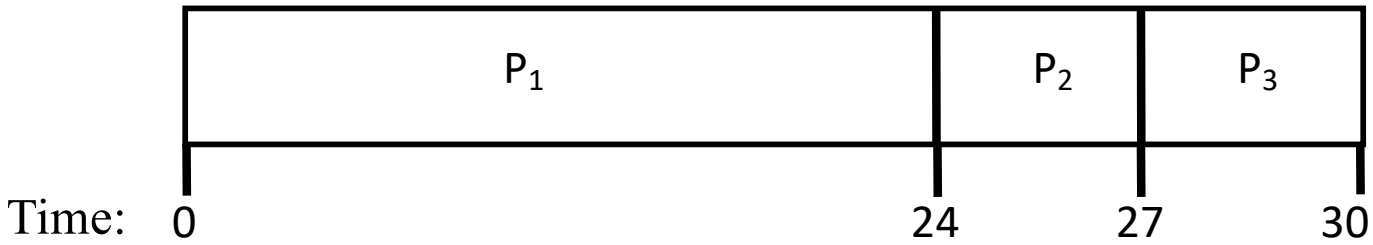
FCFS (Non-preemptive)

Process	Burst Time
P_1	24
P_2	3
P_3	3

Burst time here represents the job's entire execution time

Assume no I/O occurs

- Suppose that the processes arrive in the order: P_1, P_2, P_3
  - Assume processes arrive at the same time (e.g., time 0)
- The *Gantt chart* for the schedule is:



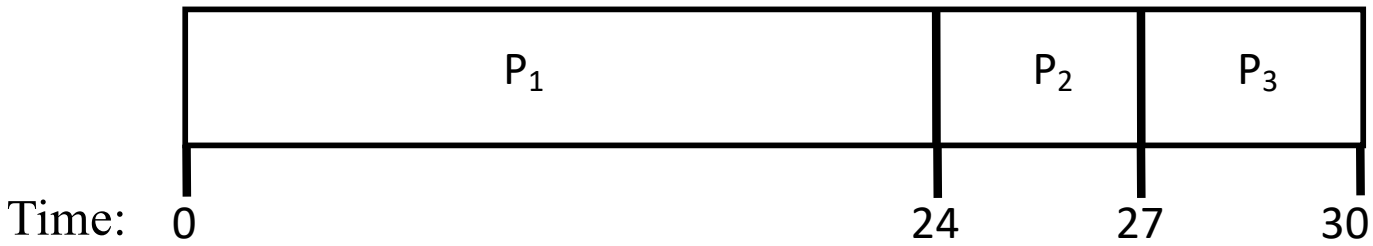
FCFS (Non-preemptive)

Process	Burst Time
P_1	24
P_2	3
P_3	3

Burst time here represents the job's entire execution time

Assume no I/O occurs

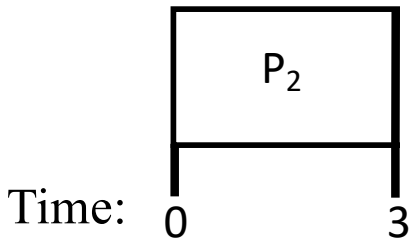
- Suppose that the processes arrive in the order: P_1, P_2, P_3
  - Assume processes arrive at the same time (e.g., time 0)
- The *Gantt chart* for the schedule is:



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

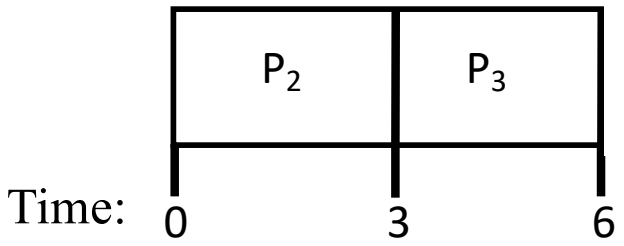
How to Reduce Waiting Time

- Suppose processes arrive in a different order: P_2, P_3, P_1
  - Again, assume that they arrive at the same time (e.g., time 0)
- The Gantt chart for the schedule is:



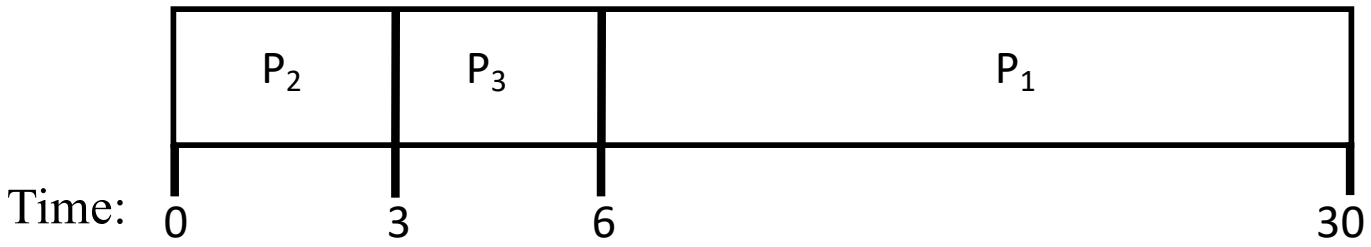
How to Reduce Waiting Time

- Suppose processes arrive in a different order: P_2, P_3, P_1
  - Again, assume that they arrive at the same time (e.g., time 0)
- The Gantt chart for the schedule is:



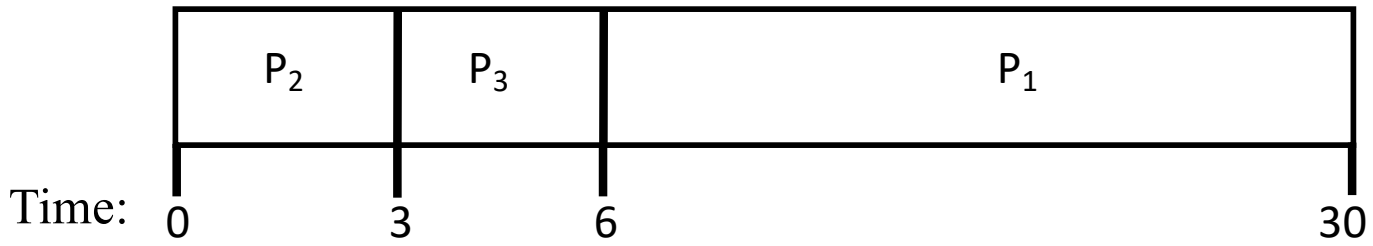
How to Reduce Waiting Time

- Suppose processes arrive in a different order: P_2, P_3, P_1
  - Again, assume that they arrive at the same time (e.g., time 0)
- The Gantt chart for the schedule is:



How to Reduce Waiting Time

- Suppose processes arrive in a different order: P_2, P_3, P_1
  - Again, assume that they arrive at the same time (e.g., time 0)
- 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 … Why?
- *Convoy effect*: short processes gets placed behind long process in the scheduling order
  - think about impact on the overall waiting time
  - earlier jobs have heavier impact

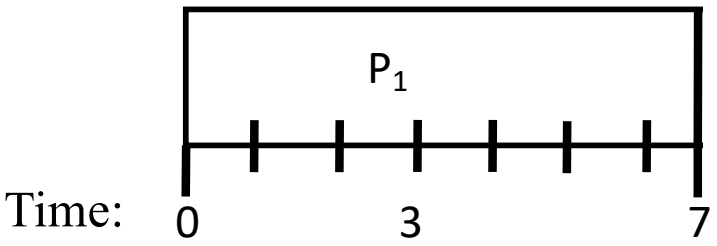
Shortest-Job-First (SJF)

- Suppose we know length of *next* CPU burst
  - Do we know this for a process? (think about this …)
- Then use these lengths to schedule the process
  - Process with the shortest next CPU burst time goes first
- Two schemes:
  - *Non-preemptive* –cannot be preempted until it completes its CPU burst
  - *Preemptive* – if a newly arrived shorter process can preempt
    - Also known as the *Shortest-Remaining-Time-First (SRTF)*
- SJF is optimal with respect to waiting time
  - Gives minimum average waiting time for a set of processes
  - So we should always use it, right? (think about drawbacks?)

SJF (Non-Preemptive)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

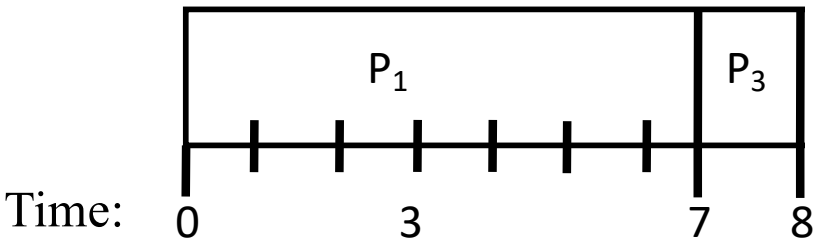


Scheduler makes a decision only when the next job is to be scheduled

SJF (Non-Preemptive)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

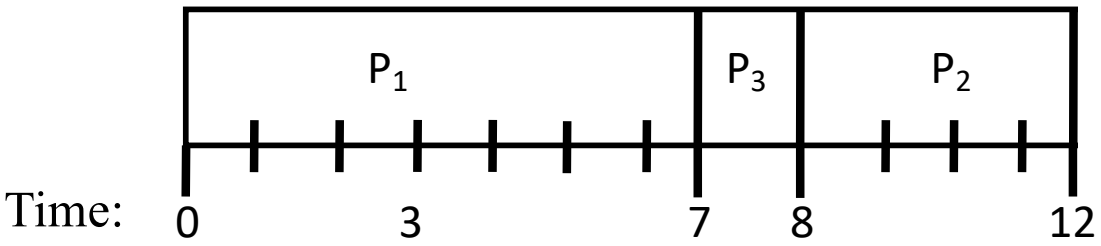


Scheduler makes a decision only when the next job is to be scheduled

SJF (Non-Preemptive)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



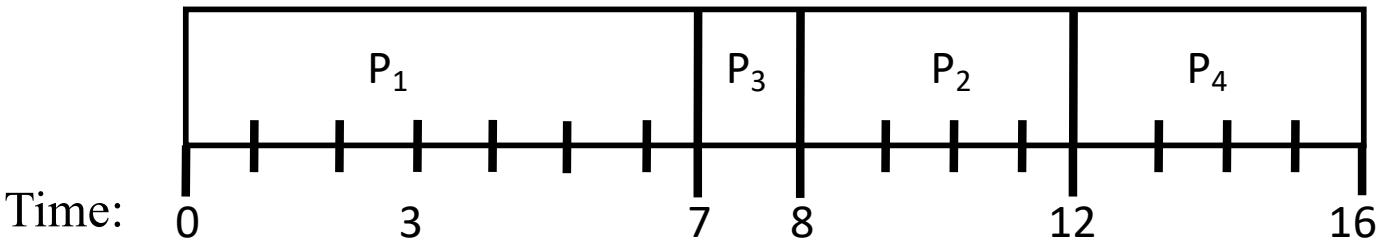
Scheduler makes a decision only when the next job is to be scheduled

SJF (Non-Preemptive)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

Scheduler makes a decision only when the next job is to be scheduled

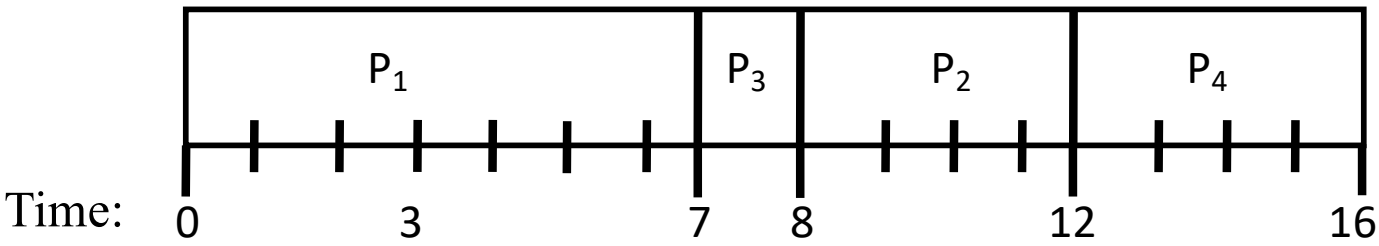


SJF (Non-Preemptive)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

Scheduler makes a decision only when the next job is to be scheduled



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

SJF (Preemptive) (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

Scheduler makes a decision at any time using preemption to stop the currently running process and context switch to another process

- SJF (preemptive) (Shortest Remaining Time First)

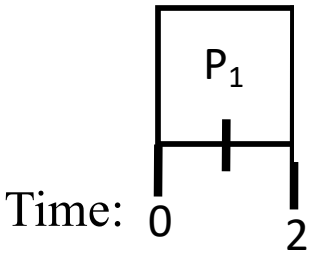
Time: 0

SJF (Preemptive) (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	5
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

Scheduler makes a decision at any time using preemption to stop the currently running process and context switch to another process

- SJF (preemptive) (Shortest Remaining Time First)

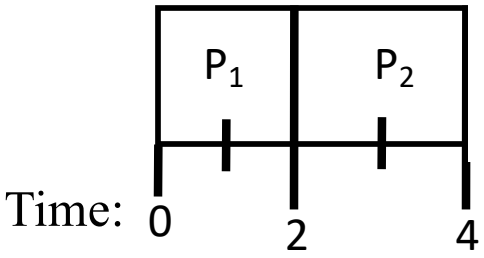


SJF (Preemptive) (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	5
P_2	2.0	2
P_3	4.0	1
P_4	5.0	4

Scheduler makes a decision at any time using preemption to stop the currently running process and context switch to another process

- SJF (preemptive) (Shortest Remaining Time First)

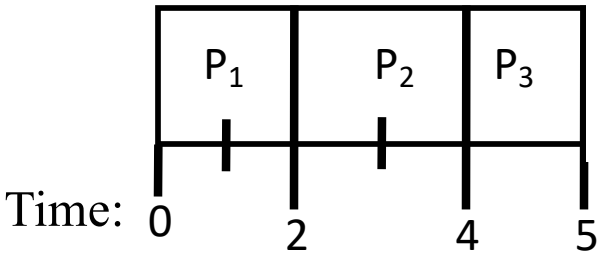


SJF (Preemptive) (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	5
P_2	2.0	2
P_3	4.0	0
P_4	5.0	4

Scheduler makes a decision at any time using preemption to stop the currently running process and context switch to another process

- SJF (preemptive) (Shortest Remaining Time First)

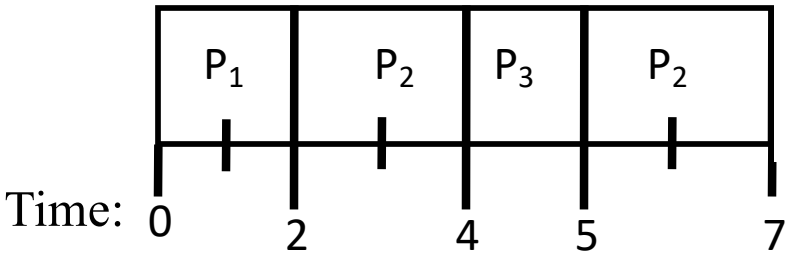


SJF (Preemptive) (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	5
P_2	2.0	0
P_3	4.0	0
P_4	5.0	4

Scheduler makes a decision at any time using preemption to stop the currently running process and context switch to another process

- SJF (preemptive) (Shortest Remaining Time First)

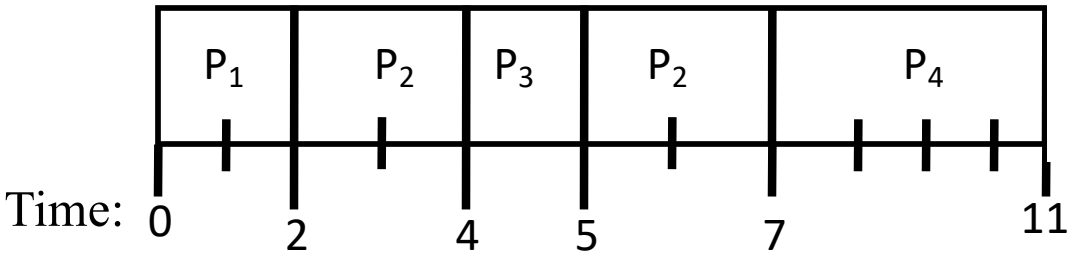


SJF (Preemptive) (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	5
P_2	2.0	0
P_3	4.0	0
P_4	5.0	0

Scheduler makes a decision at any time using preemption to stop the currently running process and context switch to another process

- SJF (preemptive) (Shortest Remaining Time First)

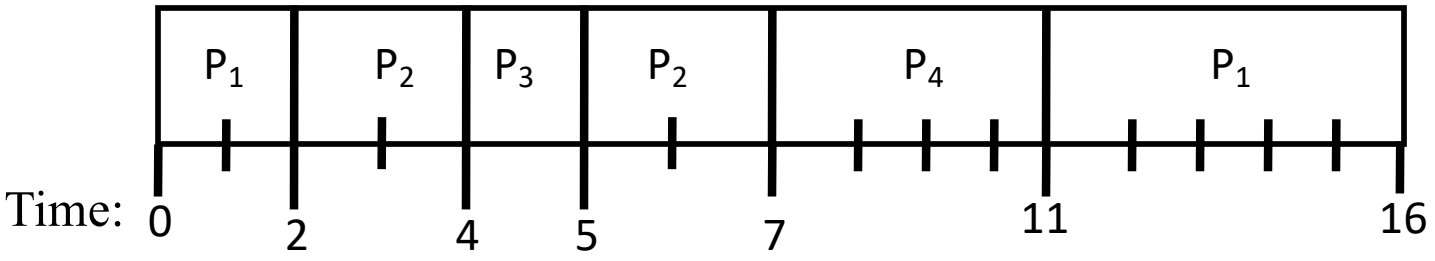


SJF (Preemptive) (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	0
P_2	2.0	0
P_3	4.0	0
P_4	5.0	0

Scheduler makes a decision at any time using preemption to stop the currently running process and context switch to another process

- SJF (preemptive) (Shortest Remaining Time First)

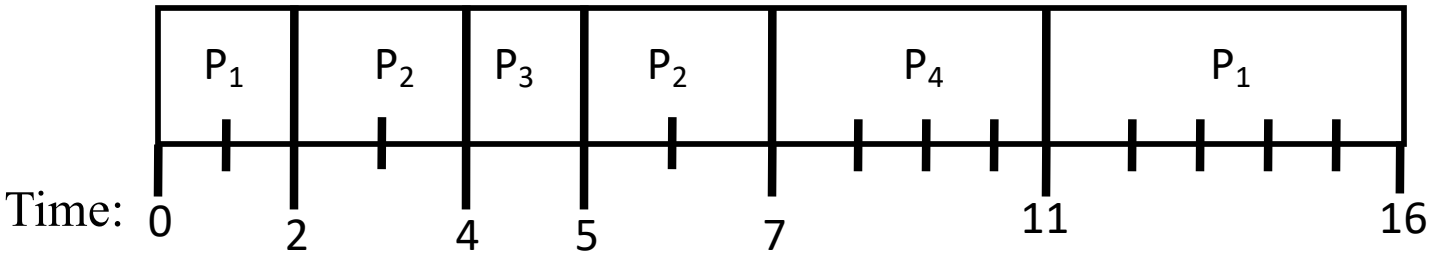


SJF (Preemptive) (SRTF)

Process	Arrival Time	Burst Time
P_1	0.0	0
P_2	2.0	0
P_3	4.0	0
P_4	5.0	0

Scheduler makes a decision at any time using preemption to stop the currently running process and context switch to another process

- SJF (preemptive) (Shortest Remaining Time First)



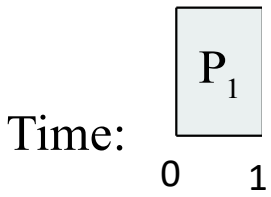
- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Example of Shortest-Remaining-Time-First

- Now consider varying arrival times and preemption

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

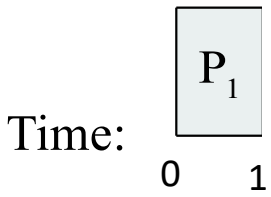


Example of Shortest-Remaining-Time-First

- Now consider varying arrival times and preemption

Process	Arrival Time	Burst Time
P_1	0	7
P_2	1	4
P_3	2	9
P_4	3	5

- Preemptive SJF Gantt Chart

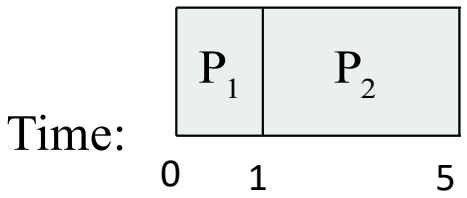


Example of Shortest-Remaining-Time-First

- Now consider varying arrival times and preemption

Process	Arrival Time	Burst Time
P_1	0	7
P_2	1	0
P_3	2	9
P_4	3	5

- Preemptive SJF Gantt Chart

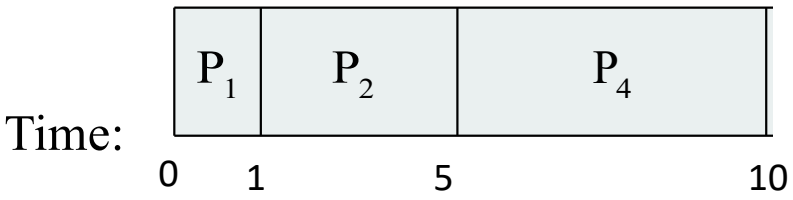


Example of Shortest-Remaining-Time-First

- Now consider varying arrival times and preemption

Process	Arrival Time	Burst Time
P_1	0	7
P_2	1	0
P_3	2	9
P_4	3	0

- Preemptive SJF Gantt Chart

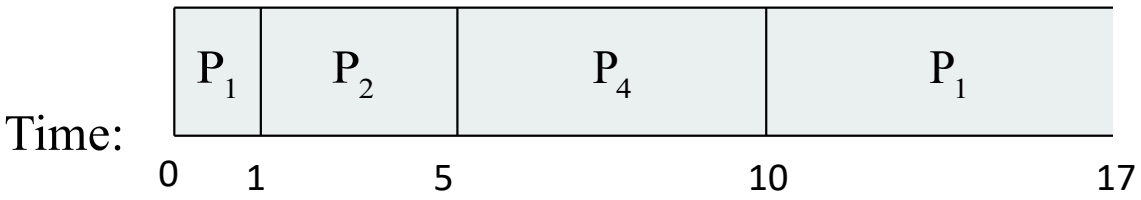


Example of Shortest-Remaining-Time-First

- Now consider varying arrival times and preemption

Process	Arrival Time	Burst Time
P_1	0	0
P_2	1	0
P_3	2	9
P_4	3	0

- Preemptive SJF Gantt Chart

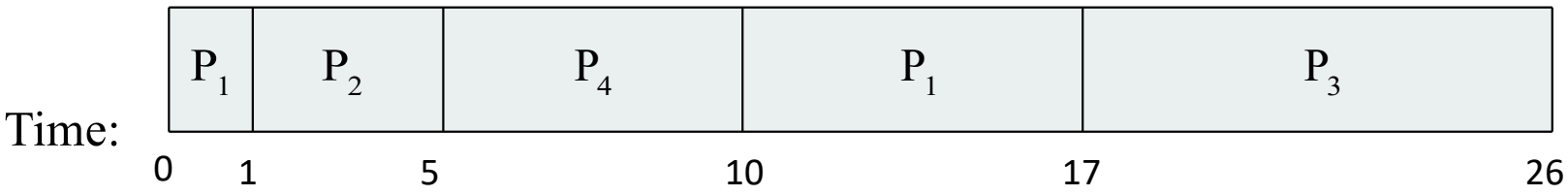


Example of Shortest-Remaining-Time-First

- Now consider varying arrival times and preemption

Process	Arrival Time	Burst Time
P_1	0	0
P_2	1	0
P_3	2	0
P_4	3	0

- Preemptive SJF Gantt Chart

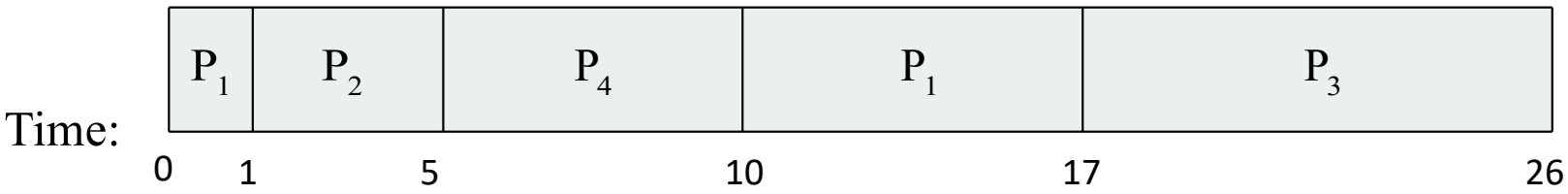


Example of Shortest-Remaining-Time-First

- Now consider varying arrival times and preemption

Process	Arrival Time	Burst Time
P_1	0	0
P_2	1	0
P_3	2	0
P_4	3	0

- Preemptive SJF Gantt Chart



- Average waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4
  = 26/4 = 6.5

Scheduling Algorithms

- First-come, First-serve (FCFS)
  - Non-preemptive
  - Does not account for waiting time (or much else)
    - convoy problem
- Shortest Job First (SJF)
  - May be preemptive (SRTF)
  - Optimal for minimizing waiting time

How to know the burst length?

How to about real-time systems?

Priority Scheduling

- Each process is given a certain priority "value"
- Always schedule the process with highest priority
  - Preemptive
  - Non-preemptive
- Problems can occur
  - Low priority processes may never execute
  - Process *starvation*
- Use *aging* to address starvation
  - Process increase priority as as time progresses
- Can formulate different scheduling policies with appropriate priority function

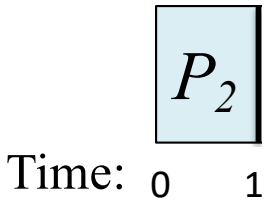
Example of Priority Scheduling (Non-Preemptive)

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Again, burst time here represents the job's entire execution time

Assume no I/O occurs

- Assume all processes arrive at the same time
- Priority scheduling Gantt Chart (non-preemptive)



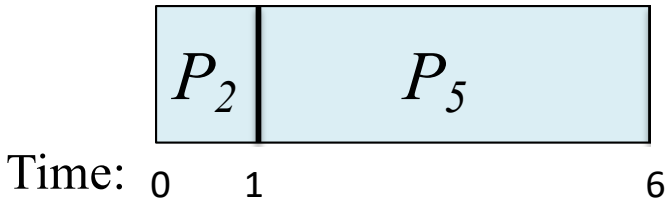
Example of Priority Scheduling (Non-Preemptive)

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Again, burst time here represents the job's entire execution time

Assume no I/O occurs

- Assume all processes arrive at the same time
- Priority scheduling Gantt Chart (non-preemptive)



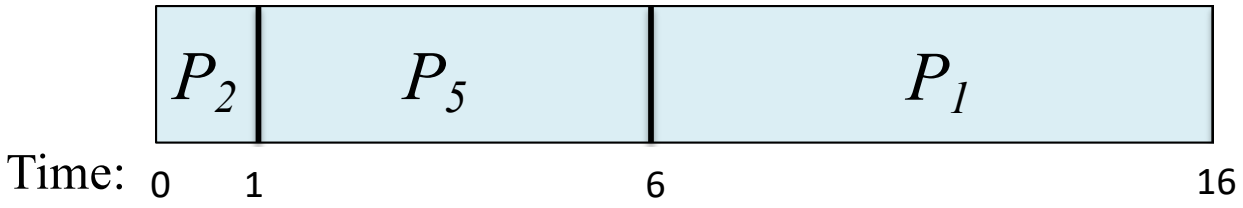
Example of Priority Scheduling (Non-Preemptive)

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Again, burst time here represents the job's entire execution time

Assume no I/O occurs

- Assume all processes arrive at the same time
- Priority scheduling Gantt Chart (non-preemptive)



Example of Priority Scheduling (Non-Preemptive)

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Again, burst time here represents the job's entire execution time

Assume no I/O occurs

- Assume all processes arrive at the same time
- Priority scheduling Gantt Chart (non-preemptive)

Example of Priority Scheduling (Non-Preemptive)

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Again, burst time here represents the job's entire execution time

Assume no I/O occurs

- Assume all processes arrive at the same time
- Priority scheduling Gantt Chart (non-preemptive)

Example of Priority Scheduling (Non-Preemptive)

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Again, burst time here represents the job's entire execution time

Assume no I/O occurs

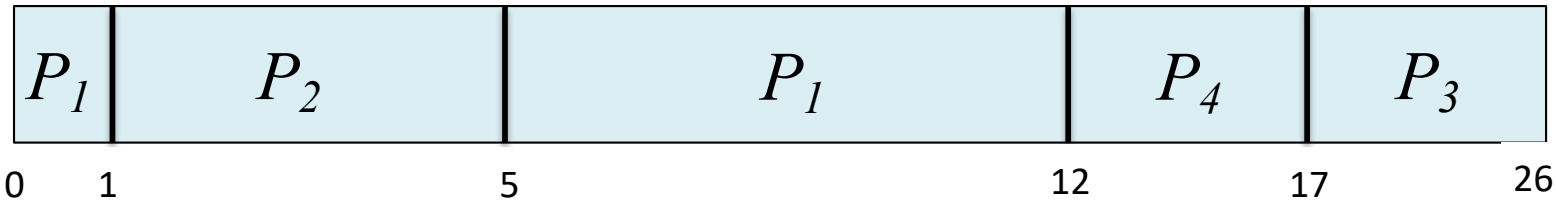
- Assume all processes arrive at the same time
- Priority scheduling Gantt Chart (non-preemptive)

- Average waiting time = 8.2

Example of Priority Scheduling (Preemptive)

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

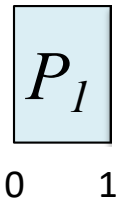
- Priority scheduling Gantt Chart (non-preemptive)



Example of Priority Scheduling (Preemptive)

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

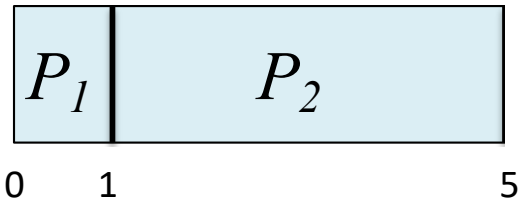
- Priority scheduling Gantt Chart (preemptive)



Example of Priority Scheduling (Preemptive)

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

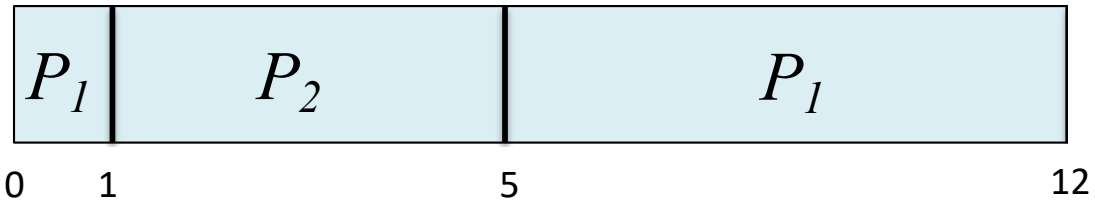
- Priority scheduling Gantt Chart (preemptive)



Example of Priority Scheduling (Preemptive)

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

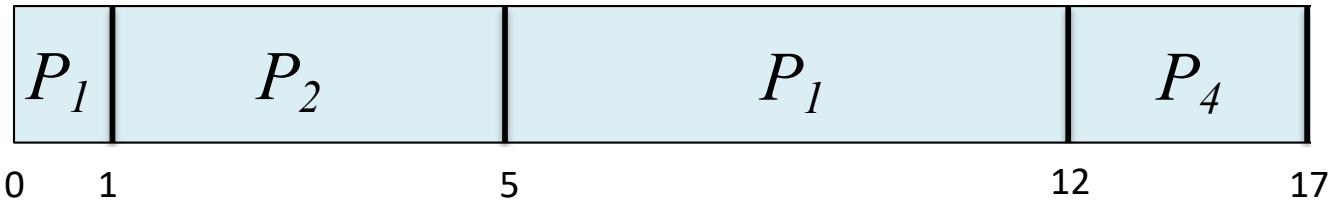
- Priority scheduling Gantt Chart (preemptive)



Example of Priority Scheduling (Preemptive)

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

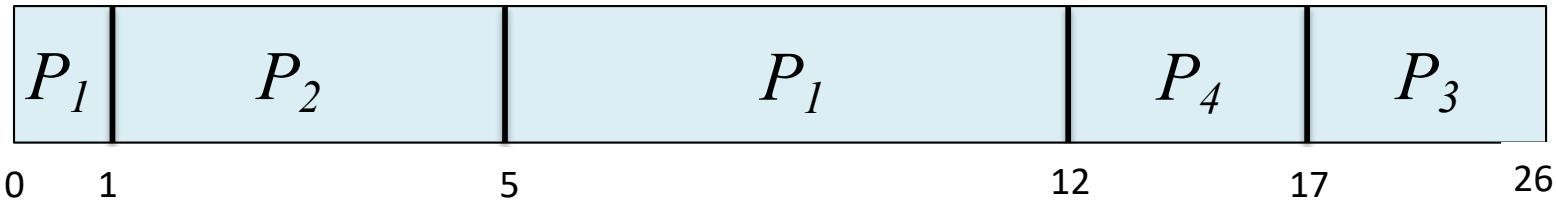
- Priority scheduling Gantt Chart (preemptive)



Example of Priority Scheduling (Preemptive)

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

- Priority scheduling Gantt Chart (preemptive)



Priorities

- Note that FCFS and SJF are specialized versions of *Priority Scheduling*
  - Assigning priorities to the processes in a certain way
- What would the priority function be for FCFS?
- What would the priority function be for SJF?

Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum)
  - Usually 10-100 milliseconds
  - After this time has elapsed, the process is *preempted* and added to the end of the ready queue
- Approach
  - Consider *n* processes in the ready queue
  - Consider time quantum is *q*
  - Then each process gets approximatly *1/n* of the CPU time
  - In chunks of at most *q* time units at once.

What hardware can facilitate Round Robin?

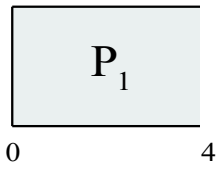
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

Assume all processes arrive at the same time in the order P_1, P_2, P_3

Still assume no I/O occurs

- The Gantt chart is:



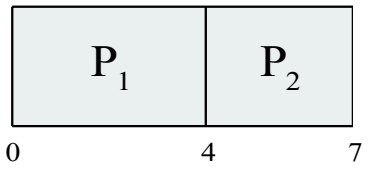
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

Assume all processes arrive at the same time in the order P_1, P_2, P_3

Still assume no I/O occurs

- The Gantt chart is:



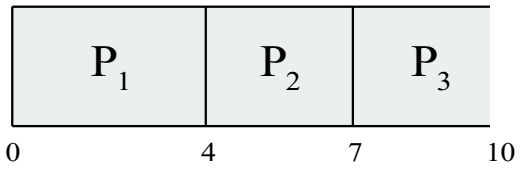
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

Assume all processes arrive at the same time in the order P_1, P_2, P_3

Still assume no I/O occurs

- The Gantt chart is:



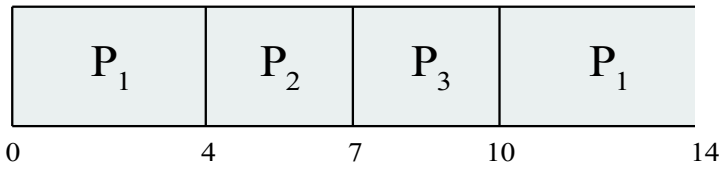
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

Assume all processes arrive at the same time in the order P_1, P_2, P_3

Still assume no I/O occurs

- The Gantt chart is:



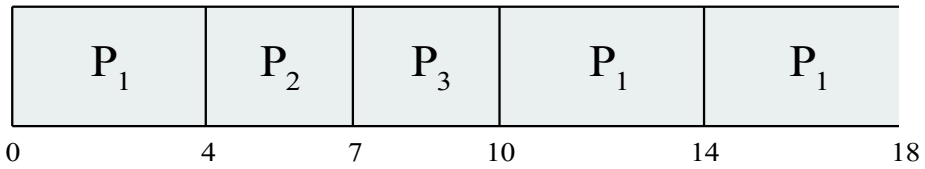
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

Assume all processes arrive at the same time in the order P_1, P_2, P_3

Still assume no I/O occurs

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better response times
- *q* should be large compared to context switch time
  - Usually *q* is between 10ms to 100ms
  - Context switch < 10 usec

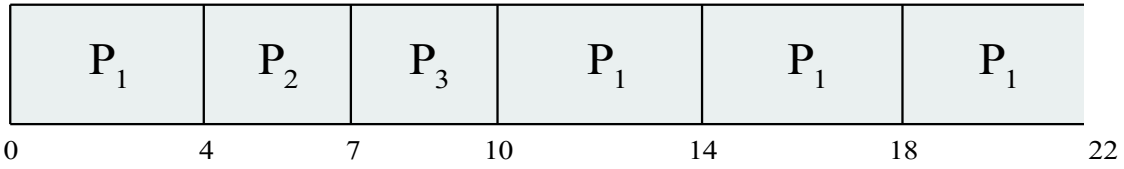
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

Assume all processes arrive at the same time in the order P_1, P_2, P_3

Still assume no I/O occurs

- The Gantt chart is:



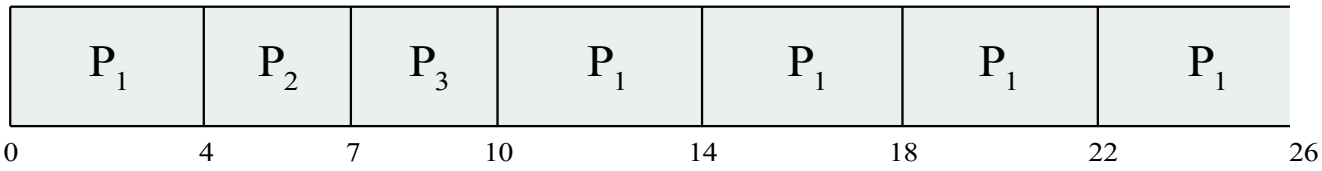
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

Assume all processes arrive at the same time in the order P_1, P_2, P_3

Still assume no I/O occurs

- The Gantt chart is:



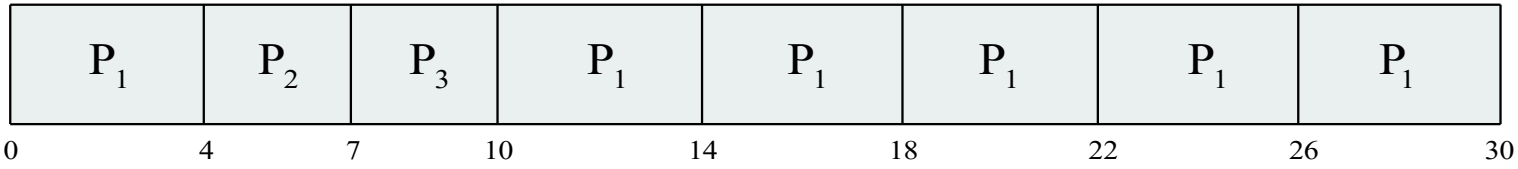
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

Assume all processes arrive at the same time in the order P_1, P_2, P_3

Still assume no I/O occurs

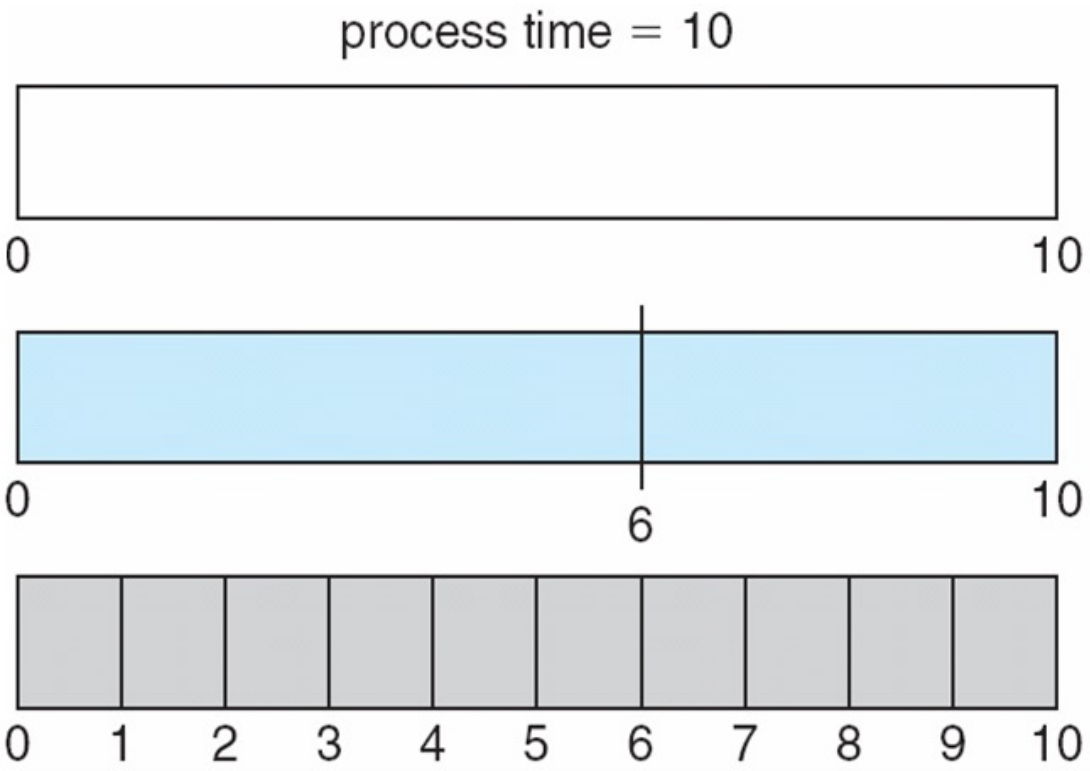
- The Gantt chart is:



RR Time Quantum

- Round robin allows the CPU to be *frequently shared* between the processes
  - All processes have the illusion that they are always running (at *1/n*-th the CPU speed)
  - What does it mean to the users?
- **Smaller** time quantums make this illusion more realistic, but there are problems
  - Issue with turnaround time?
- **Larger** time quantums will give more preference to processes with larger burst times
  - Issue with responsiveness?

Time Quantum and Context Switch Time



quantum	context switches
12	0
6	1
1	9

Note: Context switches are not free!

- Saving/restoring registers
- Switching address spaces
- Indirect costs (e.g., cache pollution)

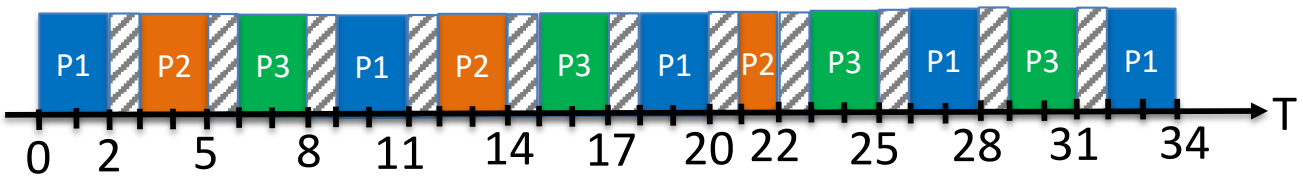
What is the consequence of more context switches?

RR Time Quantum

Process	Burst Time
P1	10
P2	*5*
P3	*8*

Assumption
- all processes arrive at the same time in the order P_1, P_2, P_3
- context switch costs 1 unit of time

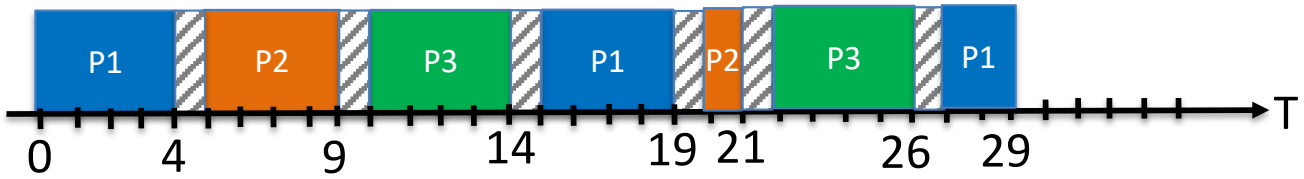
Quantum = 2



# of context switch? 11

Turnaround time? P1 P2 P3 34 22 31 Avg.= 29

Quantum = 4

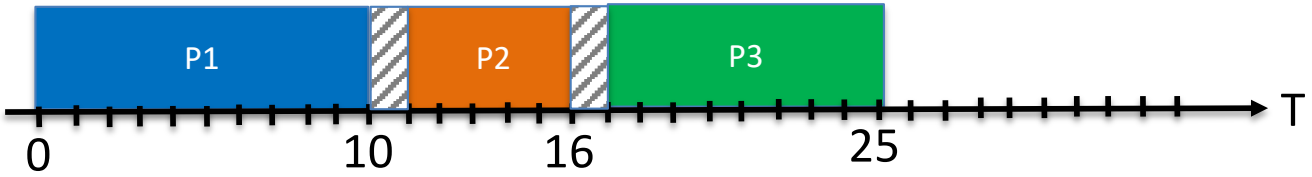


# of context switch? 6

Turnaround time? P1 P2 P3 29 21 26 Avg.= 25.33

Quantum = 10

What has this become?



# of context switch? 2

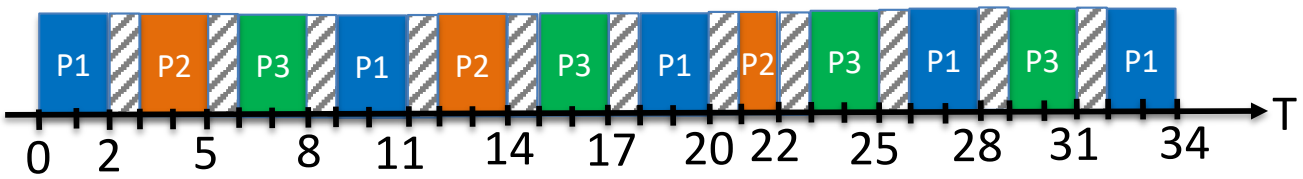
Turnaround time? P1 P2 P3 10 16 25 Avg.= 17

RR Time Quantum

Process	Burst Time
P1	10
P2	*5*
P3	*8*

Assumption
- all processes arrive at the same time in the order P_1, P_2, P_3
- context switch costs 1 unit of time

Quantum = 2



Turnaround time?

P1 P2 P3

34 22 31

Avg.= 29

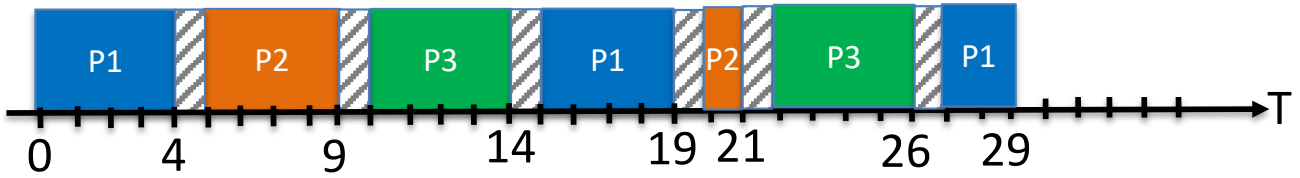
Response time?

P1 P2 P3

0 3 6

Avg.= 3

Quantum = 4



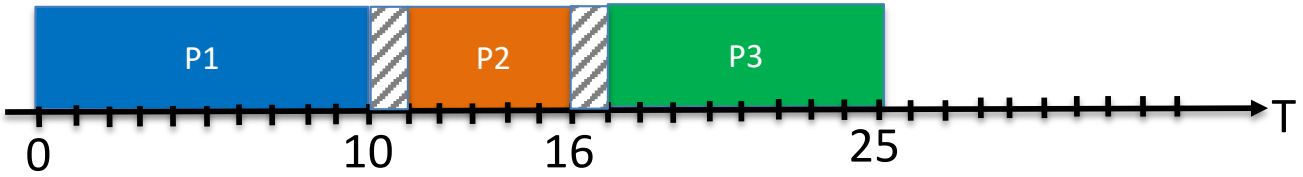
29 21 26

Avg.= 25.33

0 5 10

Avg.= 5

Quantum = 10



10 16 25

Avg.= 17

0 11 17

Avg.= 9.33

Comparing All

Criteria	FCFS	SJF	SRTF	Round Robin	Priority
Preemptive	No	No	Yes	No	Either

Comparing All

Criteria	FCFS	SJF	SRTF	Round Robin	Priority
Preemptive	No	No	Yes	No	Either
Response Time	Slow for short jobs	Fast for short jobs	Very fast for short jobs	Good for all jobs (small quantum)	Depends on priority

Comparing All

Criteria	FCFS	SJF	SRTF	Round Robin	Priority
Preemptive	No	No	Yes	No	Either
Response Time	Slow for short jobs	Fast for short jobs	Very fast for short jobs	Good for all jobs (small quantum)	Depends on priority
Job Completion Throughput	Moderate	High	High	Good (large quantum)	Depends

Comparing All

Criteria	FCFS	SJF	SRTF	Round Robin	Priority
Preemptive	No	No	Yes	No	Either
Response Time	Slow for short jobs	Fast for short jobs	Very fast for short jobs	Good for all jobs (small quantum)	Depends on priority
Job Completion Throughput	Moderate	High	High	Good (large quantum)	Depends
Context Switch Overhead	Low	Low	High	High	Depends on priority

Comparing All

Criteria	FCFS	SJF	SRTF	Round Robin	Priority
Preemptive	No	No	Yes	No	Either
Response Time	Slow for short jobs	Fast for short jobs	Very fast for short jobs	Good for all jobs (small quantum)	Depends on priority
Job Completion Throughput	Moderate	High	High	Good (large quantum)	Depends
Context Switch Overhead	Low	Low	High	High	Depends on priority
Starvation	None	Yes (long jobs may wait)	Yes (long jobs may wait)	None	Yes (low-priority may wait)

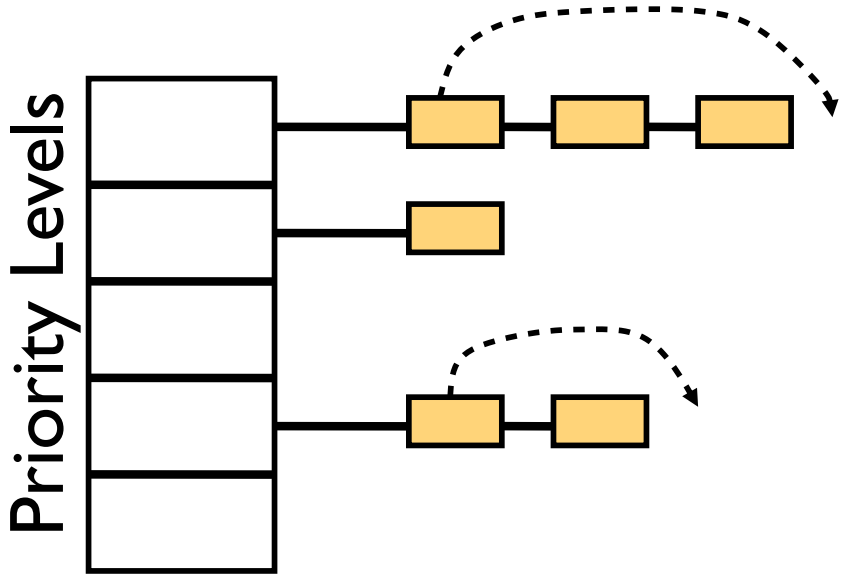
Comparing All

No single best approach? Can we mix them?

Criteria	FCFS	SJF	SRTF	Round Robin	Priority
Preemptive	No	No	Yes	No	Either
Response Time	Slow for short jobs	Fast for short jobs	Very fast for short jobs	Good for all jobs (small quantum)	Depends on priority
Job Completion Throughput	Moderate	High	High	Good (large quantum)	Depends
Context Switch Overhead	Low	Low	High	High	Depends on priority
Starvation	None	Yes (long jobs may wait)	Yes (long jobs may wait)	None	Yes (low-priority may wait)
Use Case	Batch systems (e.g., printing queues, HPC systems)	Batch systems (optimized for waiting time)	Real-time systems (aggressively optimized for waiting time)	Interactive systems (GUI)	Real-time systems (e.g., OS kernel, medical devices, industrial control)

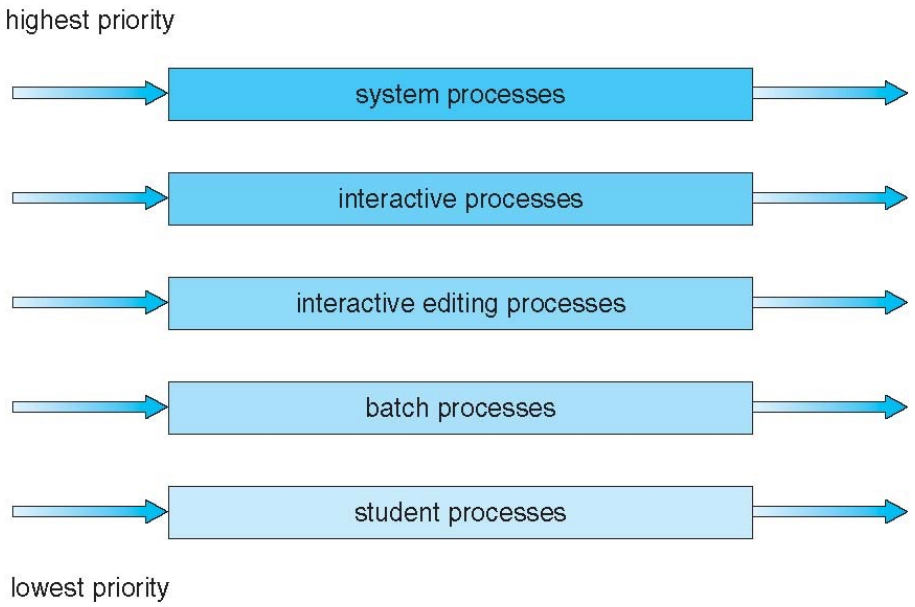
Round Robin with Priority

- Have a ready queue for each priority level
- Always service the queue at the highest priority level
- Round-robin within each queue
- Problems?
  - With fixed priorities, processes lower in the priority level can get *starved out!*
  - In general, you employ a mechanism to "age" the priority of processes



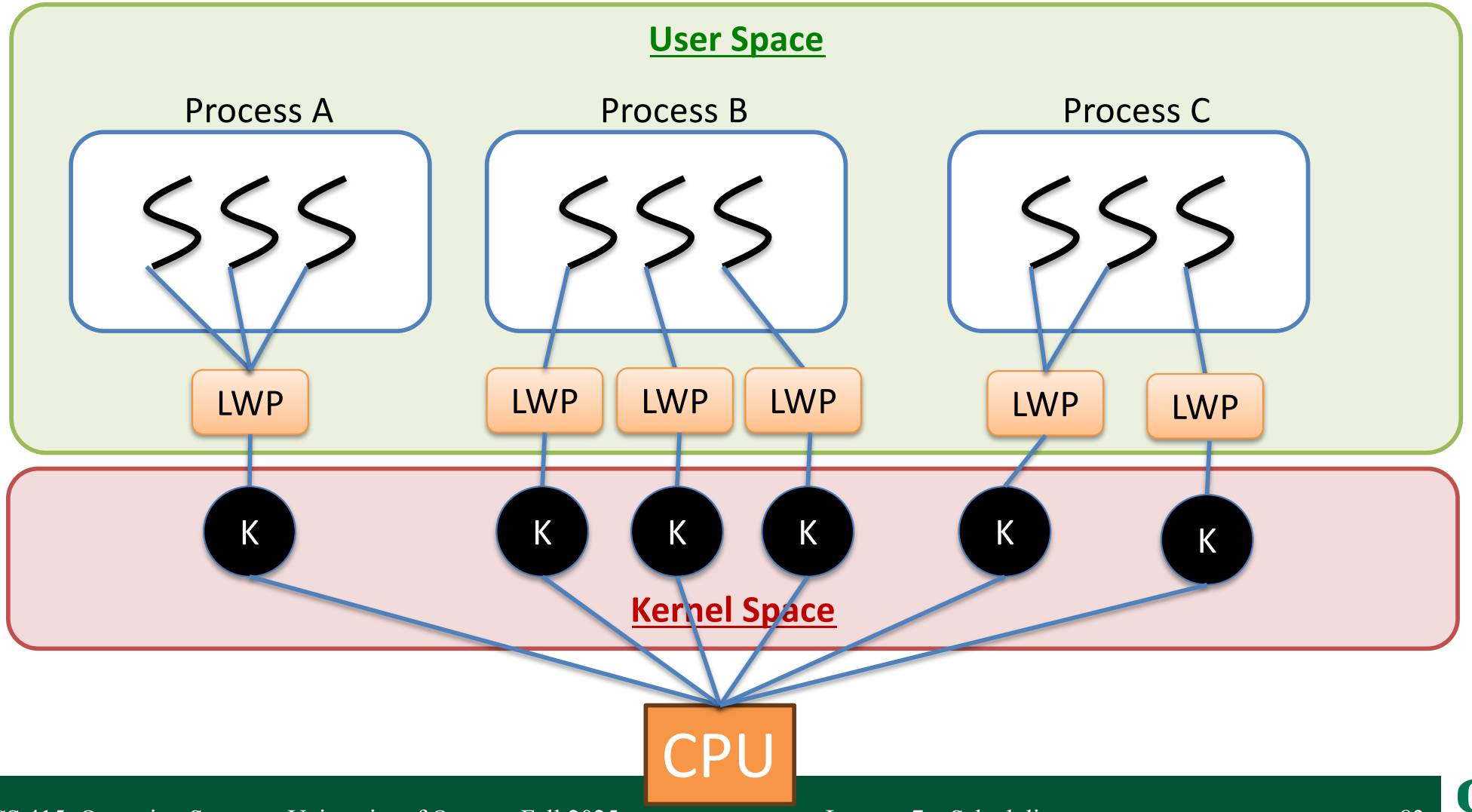
Multilevel Queue

- Ready queue is partitioned into separate queues:
  - Foreground (*interactive*)
  - Background (*batch*)
- Each queue has its own scheduling algorithm:
  - Foreground – RR
  - Background – FCFS
- Scheduling must be done between the queues. Why?
  - Fixed priority scheduling
    - possibility of starvation
  - Time slice
    - each queue gets a certain amount of CPU time
    - which it can schedule amongst its processes



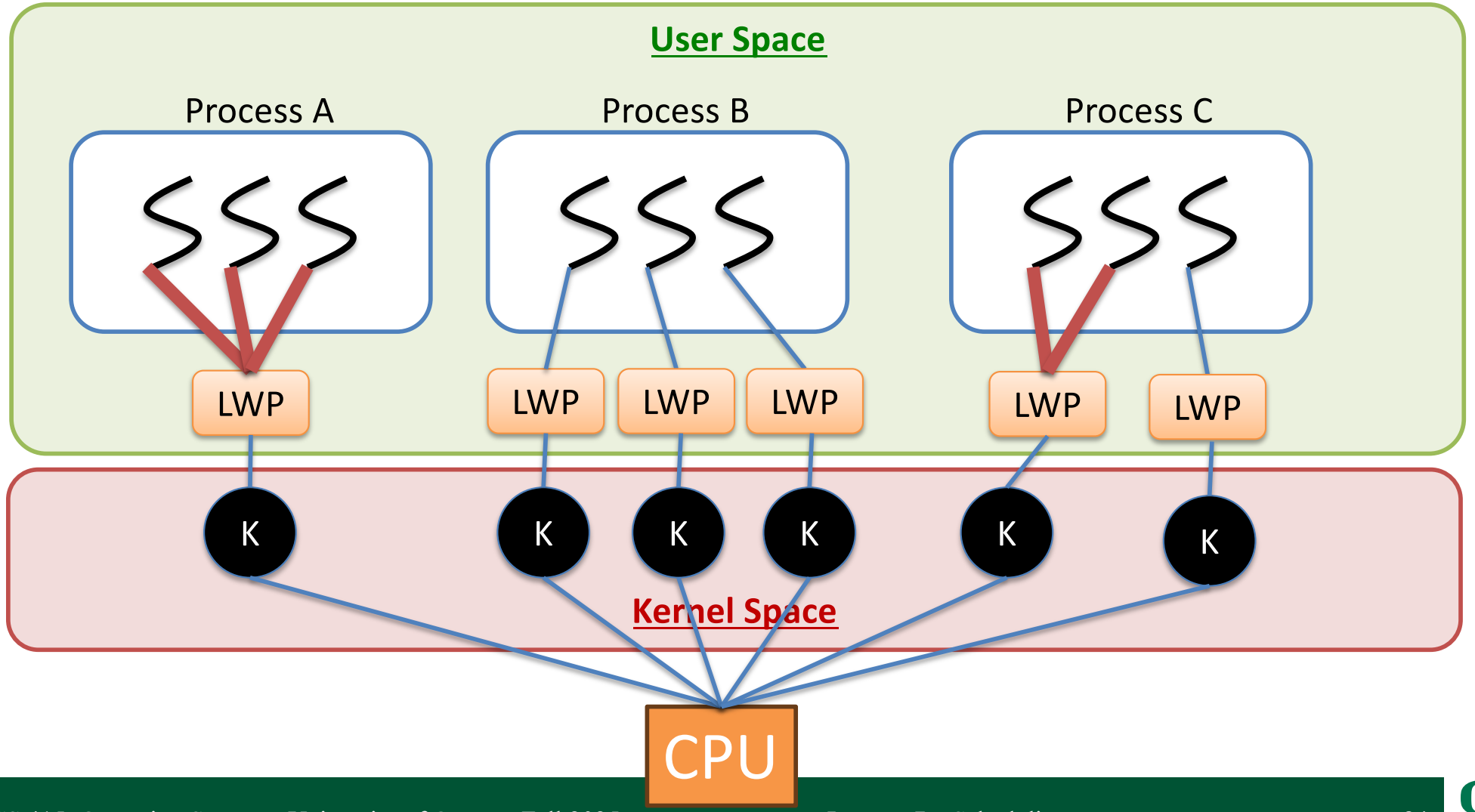
Thread Scheduling

Light-weight process (LWP): Interface between user threads and kernel threads *process-contention scope* (PCS) since scheduling



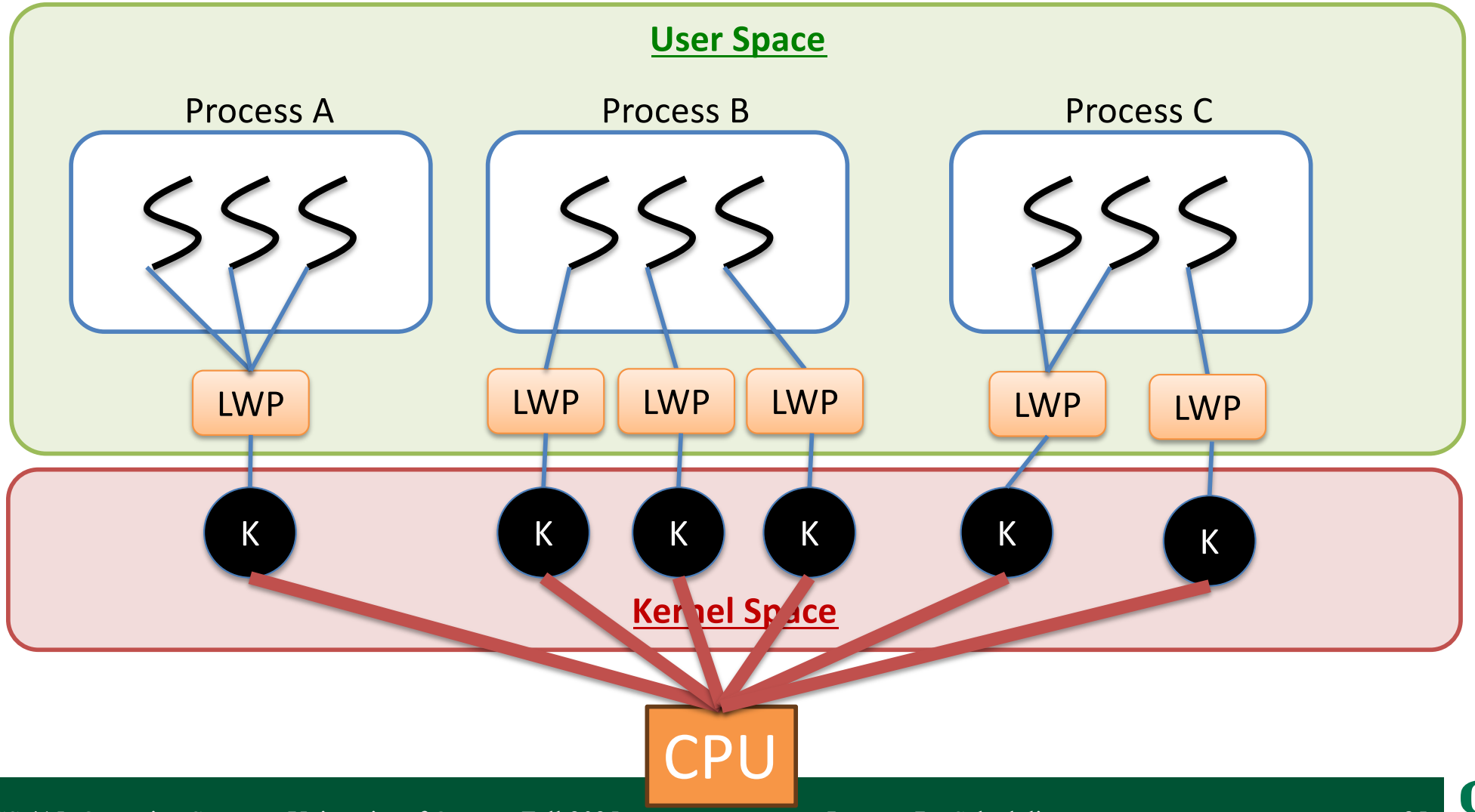
Thread Scheduling

- *Process-contention scope* (PCS) scheduling
  - The thread library decides which thread is assigned to LWP



Thread Scheduling

- *System-contention scope* (CCS) scheduling
  - OS/kernel decides which kernel thread is assigned to CPU core



Thread Scheduling Summary

- If threads are supported, it is threads that are scheduled, not processes
- N:1 and N:M models, thread library schedules user threads to run on LWP
  - Light-weight processes (LWP)
  - Known as *process-contention scope* (PCS) since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel threads are scheduled onto available CPUs in *system-contention scope* (SCS)
  - Compete among all threads in system

Does 1:1 thread model needs PCS scheduling?

Summary

- CPU Scheduling
  - Algorithms
  - Combination of algorithms
    - Multi-level Feedback Queues
  - Thread Scheduling

Next Class

- Concurrency and synchronization!