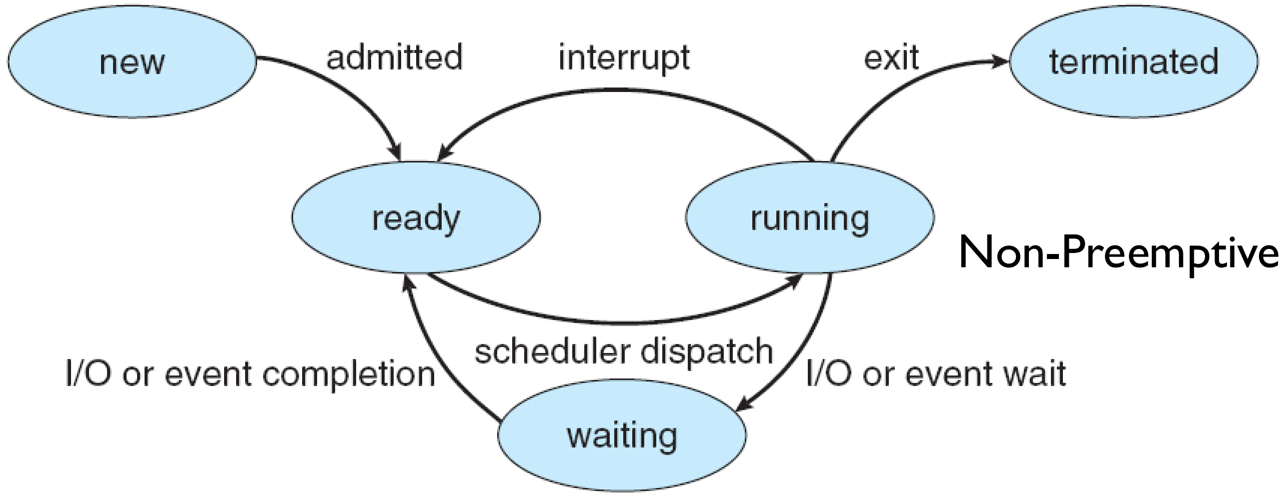
OPERATING SYSTEMS INTERNALS

CSCI 509

CHAPTER 5: CPU SCHEDULING

PREEMPTIVE VS NON-PREEMPTIVE

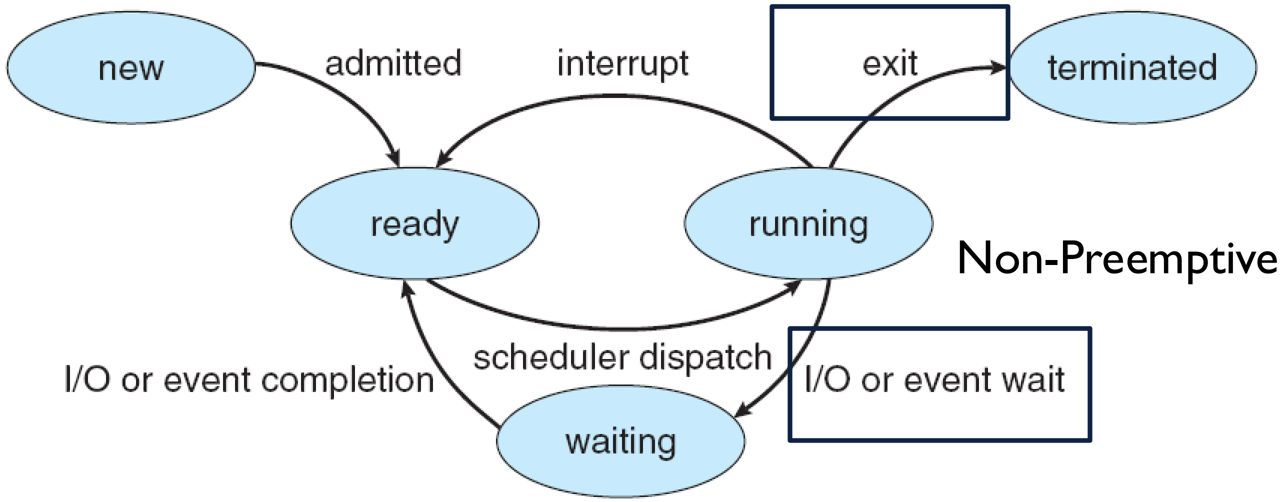
- Preemptive: Thread/Process gets switched out immediately at scheduler request.



Non-Preemptive

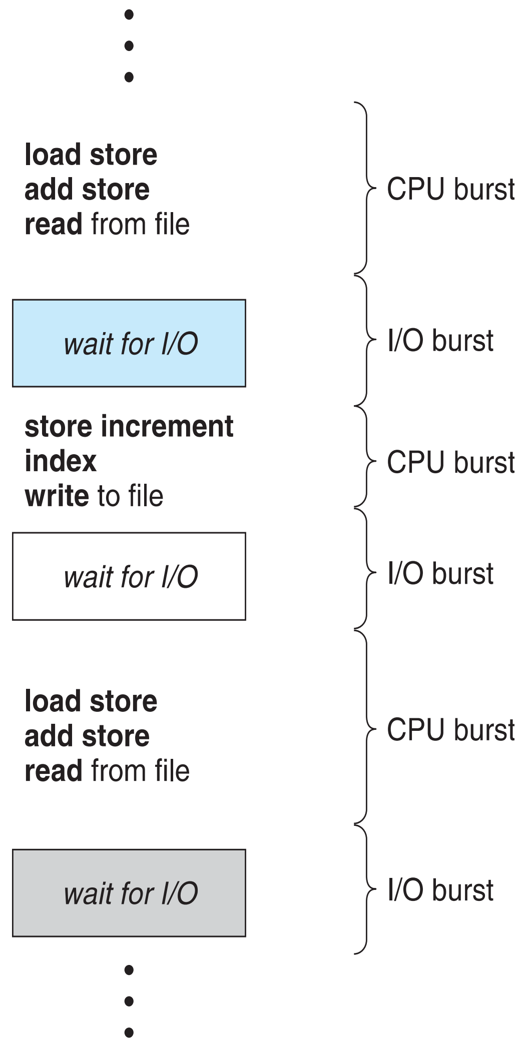
PREEMPTIVE VS NON-PREEMPTIVE

- Preemptive: Thread/Process gets switched out immediately at scheduler request.
- Non-preemptive: Thread/Process either decides to switch or is "asked" to switch out.



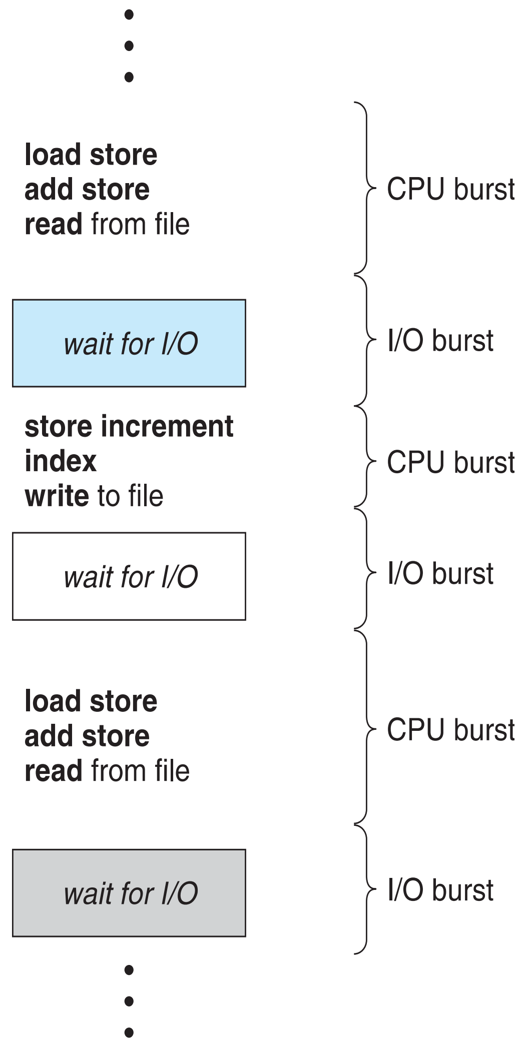
CPU SCHEDULING: I/O BURST VS CPU BURST

- Typical Thread Execution:
- **CPU burst** followed by **I/O burst**



CPU SCHEDULING: I/O BURST VS CPU BURST

- Typical Thread Execution:
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern.
- Schedulers, in general, want to avoid interrupting CPU bursts. Why?



CPU SCHEDULING CRITERIA

- How to evaluate a Scheduling algorithm?
- How to choose between one algorithm or the other?
- What's the criteria?

CPU SCHEDULING CRITERIA

- How to evaluate a Scheduling algorithm?
- How to choose between one algorithm or the other?
- What's the criteria?

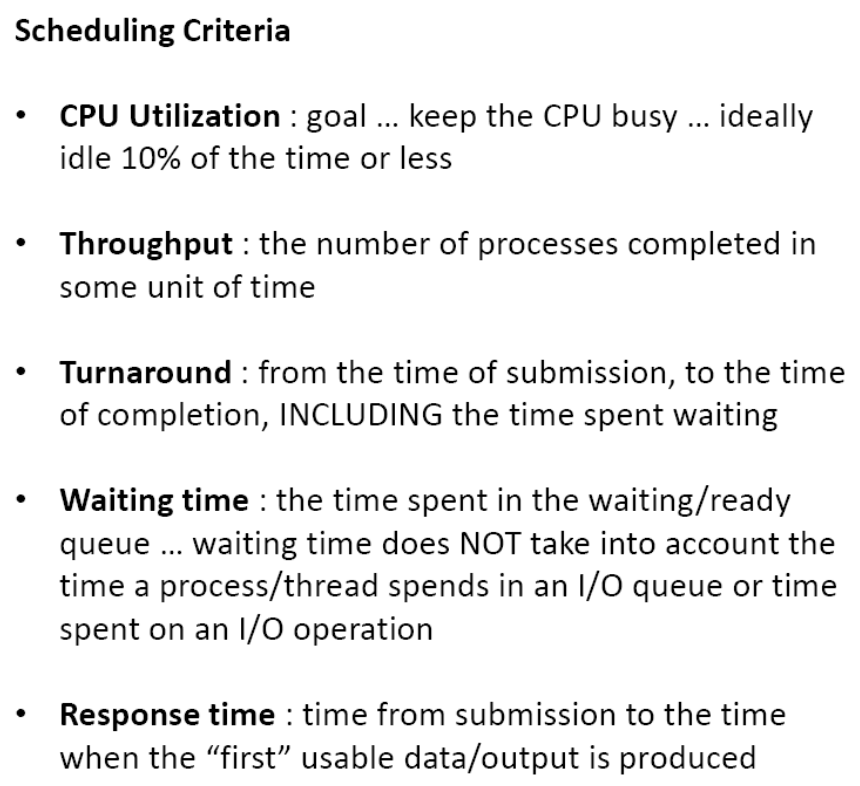
- Worksheet Task: List few scheduling performance criteria.

CPU SCHEDULING CRITERIA

How to evaluate a Scheduling algorithm? How to choose between one algorithm or the other? What's the criteria?

CPU SCHEDULING CRITERIA

How to evaluate a Scheduling algorithm? How to choose between one algorithm or the other? What's the criteria?



Scheduling Criteria

- **CPU Utilization** : goal … keep the CPU busy … ideally idle 10% of the time or less
- **Throughput** : the number of processes completed in some unit of time
- **Turnaround** : from the time of submission, to the time of completion, INCLUDING the time spent waiting
- **Waiting time** : the time spent in the waiting/ready queue … waiting time does NOT take into account the time a process/thread spends in an I/O queue or time spent on an I/O operation
- **Response time** : time from submission to the time when the "first" usable data/output is produced

CPU SCHEDULING CRITERIA

How to evaluate a Scheduling algorithm? How to choose between one algorithm or the other? What's the criteria?

Scheduling Criteria

- **CPU Utilization** : goal … keep the CPU busy … ideally idle 10% of the time or less
- **Throughput** : the number of processes completed in some unit of time
- **Turnaround** : from the time of submission, to the time of completion, INCLUDING the time spent waiting
- **Waiting time** : the time spent in the waiting/ready queue … waiting time does NOT take into account the time a process/thread spends in an I/O queue or time spent on an I/O operation
- **Response time** : time from submission to the time when the "first" usable data/output is produced

Q: Which of these do we want to minimize?

Q: Which of these do we want to maximize?

CPU SCHEDULING CRITERIA

Scheduling Criteria

- **CPU Utilization** : goal … keep the CPU busy … ideally idle 10% of the time or less ↑
- **Throughput** : the number of processes completed in some unit of time
- **Turnaround** : from the time of submission, to the time of completion, INCLUDING the time spent waiting
- **Waiting time** : the time spent in the waiting/ready queue … waiting time does NOT take into account the time a process/thread spends in an I/O queue or time spent on an I/O operation
- **Response time** : time from submission to the time when the “first” usable data/output is produced

Q: Which of these do we want to minimize?

Q: Which of these do we want to maximize?

CPU SCHEDULING CRITERIA

Scheduling Criteria

- **CPU Utilization** : goal … keep the CPU busy … ideally idle 10% of the time or less ↑
- **Throughput** : the number of processes completed in some unit of time ↑
- **Turnaround** : from the time of submission, to the time of completion, INCLUDING the time spent waiting
- **Waiting time** : the time spent in the waiting/ready queue … waiting time does NOT take into account the time a process/thread spends in an I/O queue or time spent on an I/O operation
- **Response time** : time from submission to the time when the "first" usable data/output is produced

Q: Which of these do we want to minimize?

Q: Which of these do we want to maximize?

CPU SCHEDULING CRITERIA

Scheduling Criteria

- **CPU Utilization** : goal … keep the CPU busy … ideally idle 10% of the time or less ↑
- **Throughput** : the number of processes completed in some unit of time ↑
- **Turnaround** : from the time of submission, to the time of completion, INCLUDING the time spent waiting ↓
- **Waiting time** : the time spent in the waiting/ready queue … waiting time does NOT take into account the time a process/thread spends in an I/O queue or time spent on an I/O operation
- **Response time** : time from submission to the time when the "first" usable data/output is produced

Q: Which of these do we want to minimize?

Q: Which of these do we want to maximize?

CPU SCHEDULING CRITERIA

Scheduling Criteria

- **CPU Utilization** : goal … keep the CPU busy … ideally idle 10% of the time or less ↑
- **Throughput** : the number of processes completed in some unit of time ↑
- **Turnaround** : from the time of submission, to the time of completion, INCLUDING the time spent waiting ↓
- **Waiting time** : the time spent in the waiting/ready queue … waiting time does NOT take into account the time a process/thread spends in an I/O queue or time spent on an I/O operation ↓
- **Response time** : time from submission to the time when the "first" usable data/output is produced

Q: Which of these do we want to minimize?

Q: Which of these do we want to maximize?

CPU SCHEDULING CRITERIA

Scheduling Criteria

- **CPU Utilization** : goal … keep the CPU busy … ideally idle 10% of the time or less ↑
- **Throughput** : the number of processes completed in some unit of time ↑
- **Turnaround** : from the time of submission, to the time of completion, INCLUDING the time spent waiting ↓
- **Waiting time** : the time spent in the waiting/ready queue … waiting time does NOT take into account the time a process/thread spends in an I/O queue or time spent on an I/O operation ↓
- **Response time** : time from submission to the time when the “first” usable data/output is produced ↓

Q: Which of these do we want to minimize?

Q: Which of these do we want to maximize?

FIFO : First In First Out == FCFS : First Come First Served

(queue implementation)   (behavior description)

FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling

FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Once a process starts a burst, it won’t be interrupted.

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling

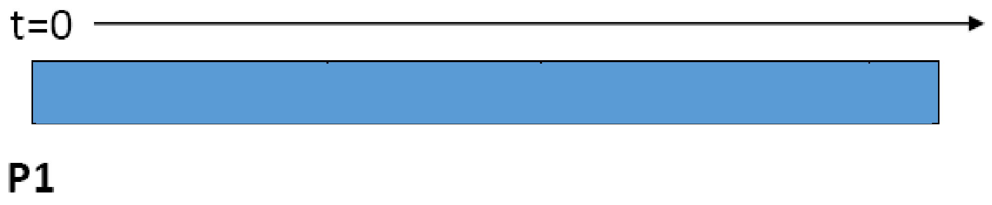
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	24 ms
P2	16 ms
P3	32 ms
P4	4 ms

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



At what time is P2 allowed to "begin"?

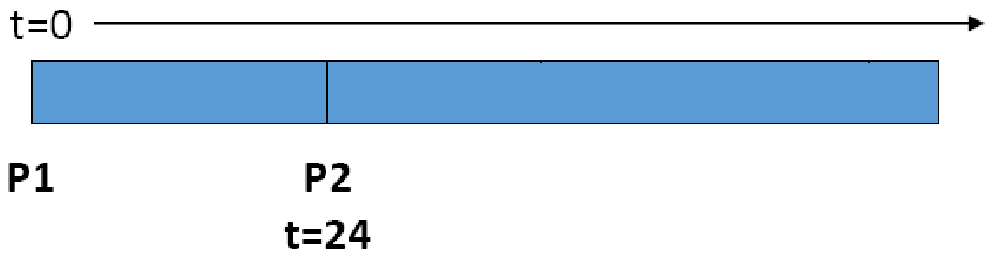
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



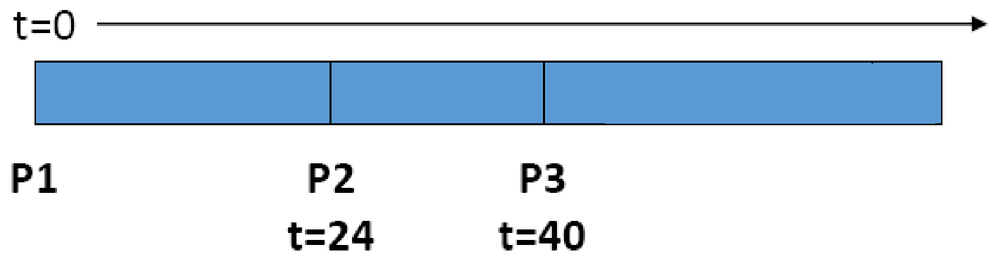
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



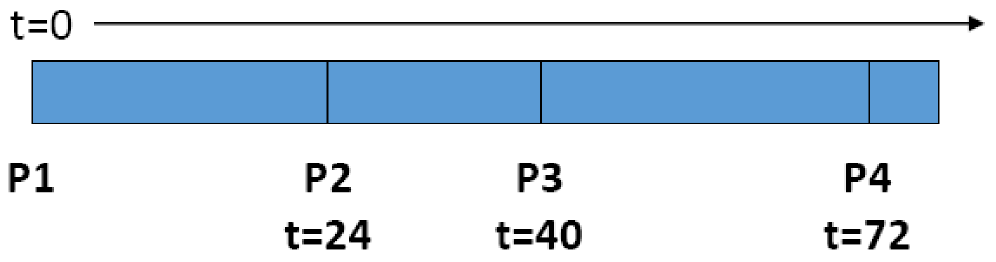
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



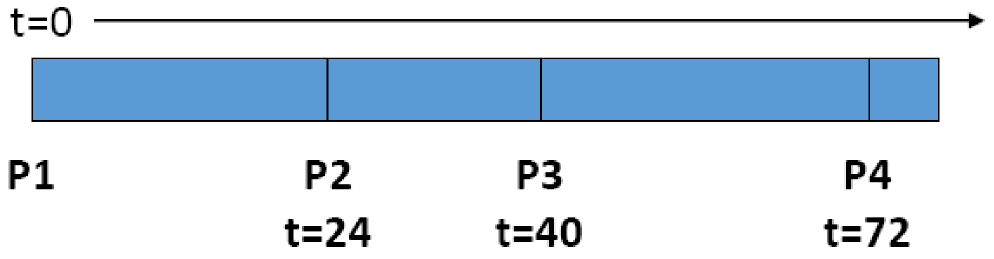
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation)   (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



Worksheet Q2

Q: What are the advantages of this approach?

Q: What are the disadvantages of this approach?

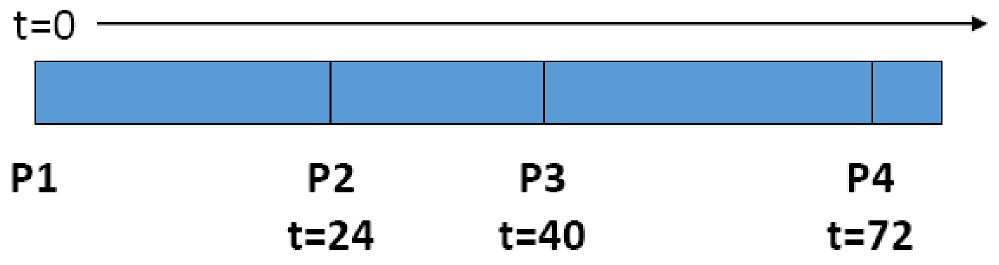
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



Q: What are the advantages of this approach?

Easy to implement
No scheduling "calculation" required

Q: What are the disadvantages of this approach?

In a time sharing system, each process might not get a share of the CPU at some regular interval $\longrightarrow$ long wait times

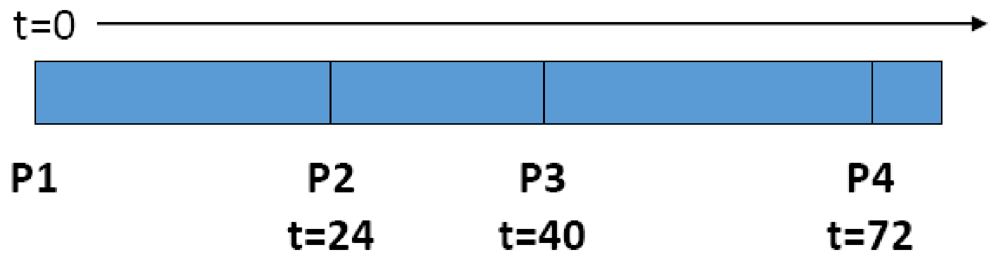
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



Q: What are the advantages of this approach?

Q: What are the disadvantages of this approach?

To "see" the disadvantage of this approach, what if P1 had a burst time of 300ms. Should it still "go" first?

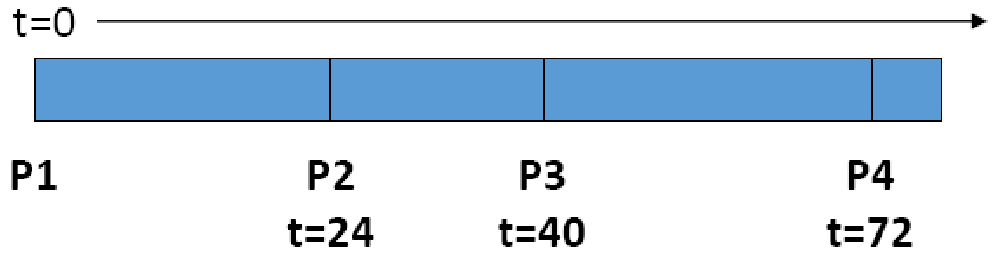
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



Q: To quantitatively assess the "goodness" of FIFO/FCFS with nonpreemptive scheduling, what metric (CPU Utilization, Throughput, Turnaround, Waiting time, Response time) should we calculate?

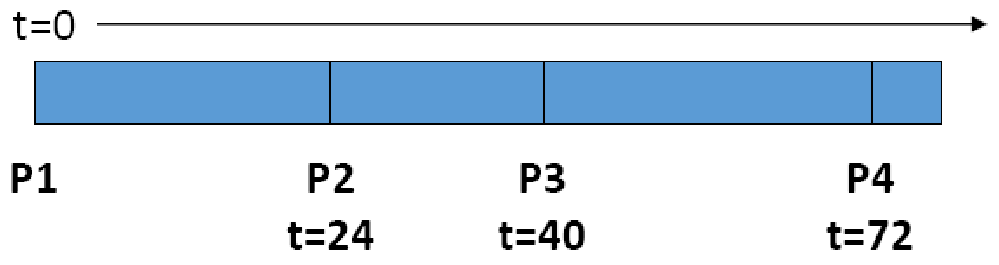
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



Q: To quantitatively assess the "goodness" of FIFO/FCFS with nonpreemptive scheduling, what metric (CPU Utilization, Throughput, Turnaround, Waiting time, Response time) should we calculate?

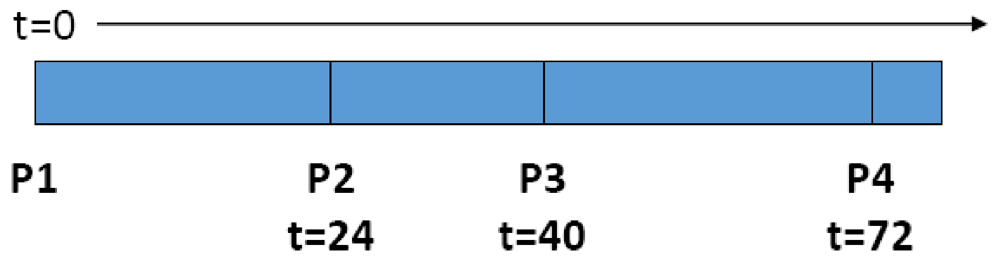
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



Wait times : P1 : 0 ms

P2 : 24 ms

P3 : 40 ms

P4 : 72 ms

Average : 34 ms

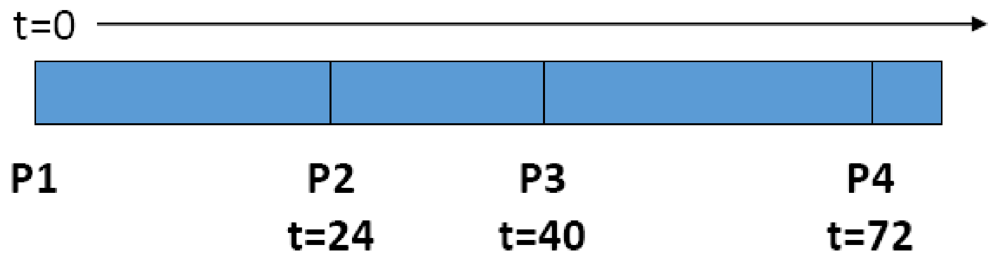
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



Wait times :

P1 : 0 ms
P2 : 24 ms
P3 : 40 ms
P4 : 72 ms

Average : 34 ms

Q: Can we do better?

Q: Is a mere reordering of start times sufficient to reduce average wait time?

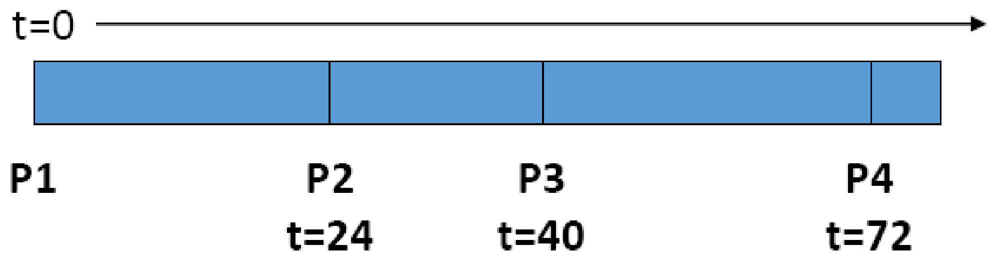
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



Wait times : P1 : 0 ms

P2 : 24 ms

P3 : 40 ms

P4 : 72 ms

Average : 34 ms

Q: How does swapping the order of P1 and P2 alter the average wait time for the 4 processes?

🟥 A	🟩 B
🟨 C	🟦 D

A : reduce
B : increase
C : no change

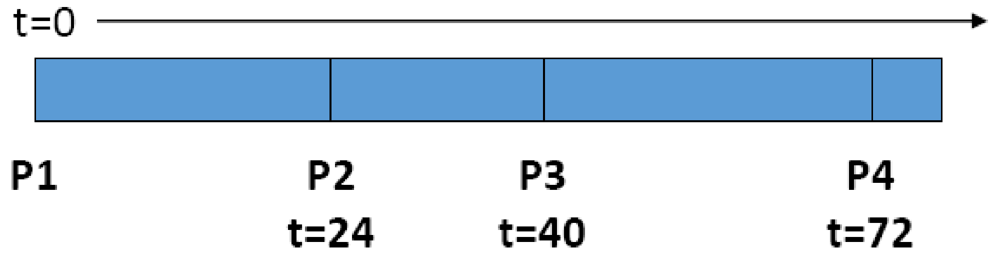
FIFO : First In First Out == FCFS : First Come First Served

(queue implementation) (behavior description)

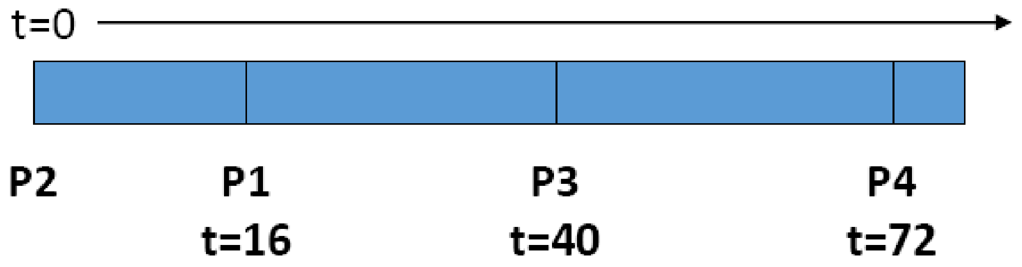
Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If P1 < P2 < P3 < P4 and assuming nonpreemptive scheduling



If P2 < P1 < P3 < P4 and assuming nonpreemptive scheduling



Average Wait times

P1 < P2 < P3 < P4 : 34 ms

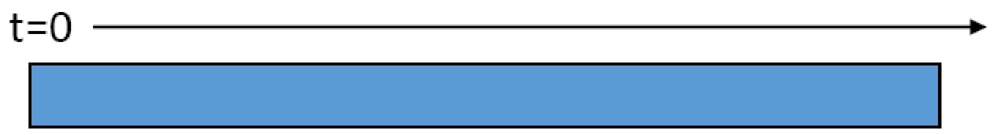
P2 < P1 < P3 < P4 : 32 ms

Shortest Job First (SJF)

Assume 4 processes and their required burst times

P1	24 ms
P2	16 ms
P3	32 ms
P4	4 ms

If all 4 processes are requesting access to the CPU … *



* That also means that the ready queue is no longer a "queue" in the classical sense …

Shortest Job First (SJF)

Assume 4 processes and their required burst times

P1	24 ms
P2	16 ms
P3	32 ms
P4	4 ms

If all 4 processes are requesting access to the CPU … *

What would be the most suitable data structure for an SJF scheduling algorithm?

Shortest Job First (SJF)

Assume 4 processes and their required burst times

P1	**24 ms**
P2	**16 ms**
P3	**32 ms**
P4	**4 ms**

If all 4 processes are requesting access to the CPU ... *

	insert	deleteMin	remove	findMin
ordered array	O(n)	O(1)	O(n)	O(1)
ordered list	O(n)	O(1)	O(1)	O(1)
unordered array	O(1)	O(n)	O(1)	O(n)
unordered list	O(1)	O(n)	O(1)	O(n)
binary heap	O(log n)	O(log n)	O(log n)	O(1)

Shortest Job First (SJF)

Assume 4 processes and their required burst times

P1	24 ms
P2	16 ms
P3	32 ms
P4	4 ms

If all 4 processes are requesting access to the CPU … *

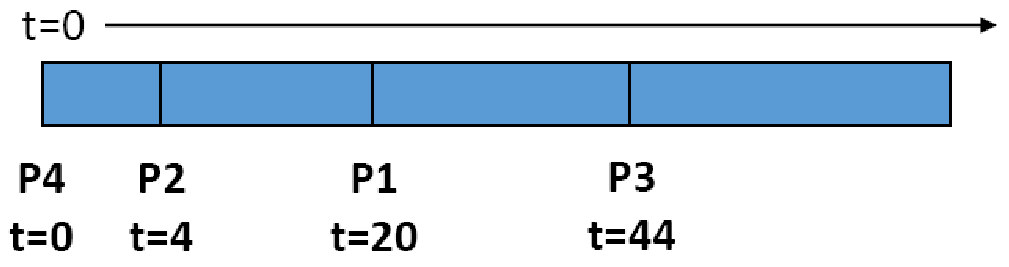
	insert	deleteMin	remove	findMin
ordered array	O(n)	O(1)	O(n)	O(1)
ordered list	O(n)	O(1)	O(1)	O(1)
unordered array	O(1)	O(n)	O(1)	O(n)
unordered list	O(1)	O(n)	O(1)	O(n)
binary heap	O(log n)	O(log n)	O(log n)	O(1)

Shortest Job First (SJF)

Assume 4 processes and their required burst times

P1	24 ms
P2	16 ms
P3	32 ms
P4	4 ms

If all 4 processes are requesting access to the CPU ...



Worksheet Q3

Q: What are the advantages of this approach?
Q: What are the disadvantages of this approach?

(versus FCFS)

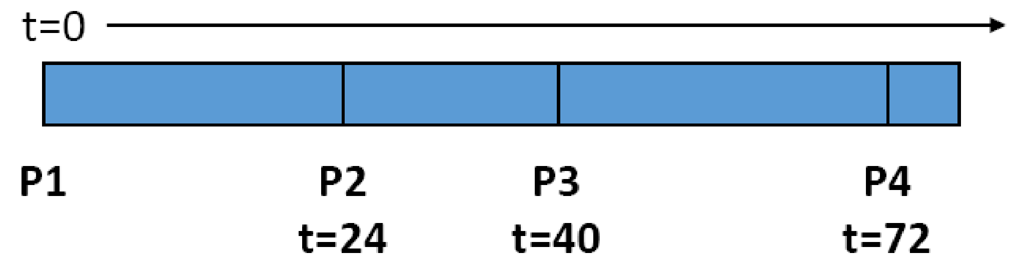
Assume 4 processes and their required burst times

P1	24 ms
P2	16 ms
P3	32 ms
P4	4 ms

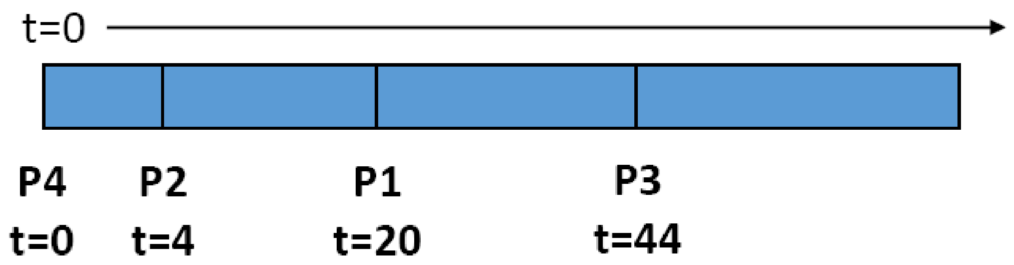
Average wait times

FCFS : 34ms
SJF : 17ms

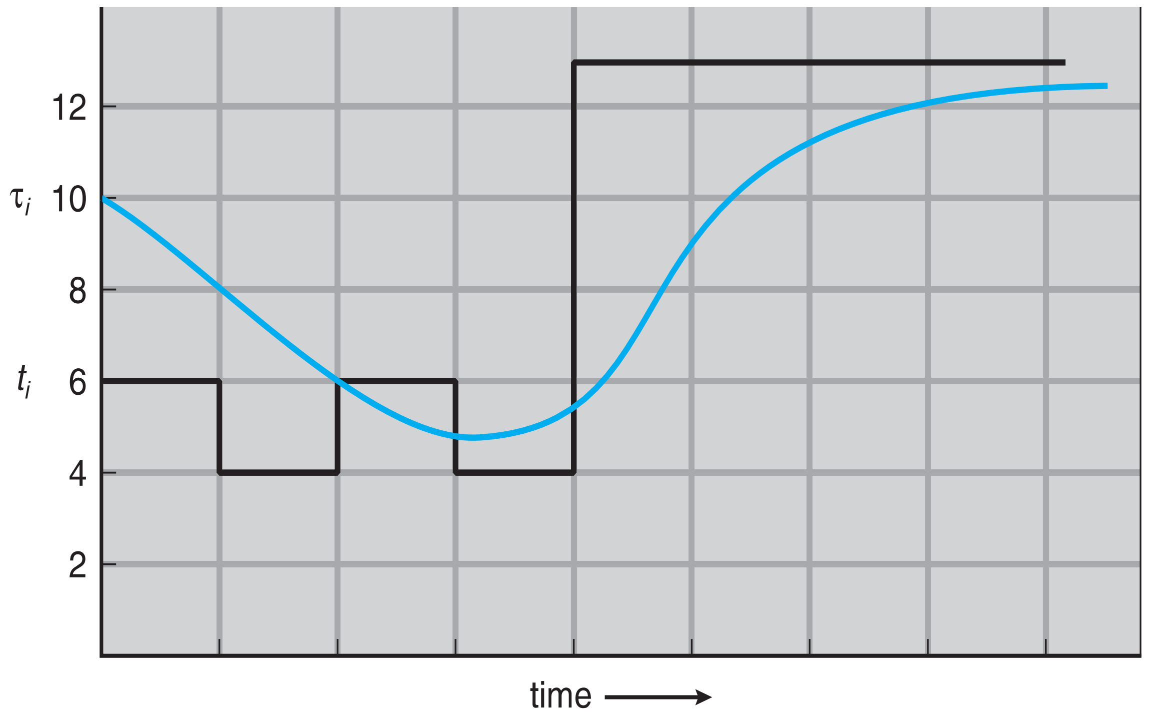
First Come First Serve P1 < P2 < P3 < P4



Shortest Job First



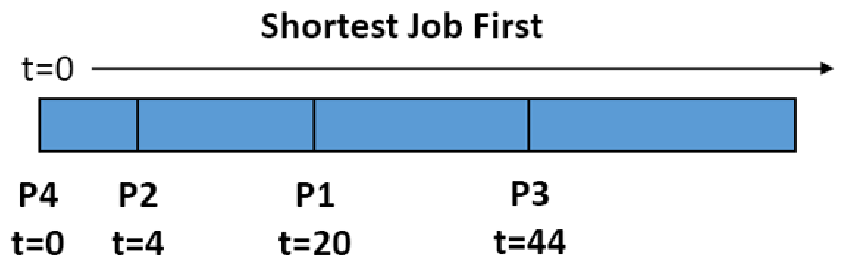
ESTIMATING PROCESS BURST DURATION



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 (SJF): ESTIMATING PROCESS BURST DURATION

- Can only estimate the length – should be similar to the previous on
  - Then pick process with shortest predicted next CPU burst



- Can be done by using the length of previous CPU bursts, using exponential averaging

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

- Commonly, α set to ½

IN CLASS EXERCISE

Assume 5 processes and their required burst times and their arrival times

P11	4 ms	t=0
P37	5 ms	t=5
P7	2 ms	t=2
P16	4 ms	t=3
P25	3 ms	t=7

Task : If SJF nonpreemptive CPU scheduling is used, then what is the average wait time for the 5 processes, and at what time does each process finish?

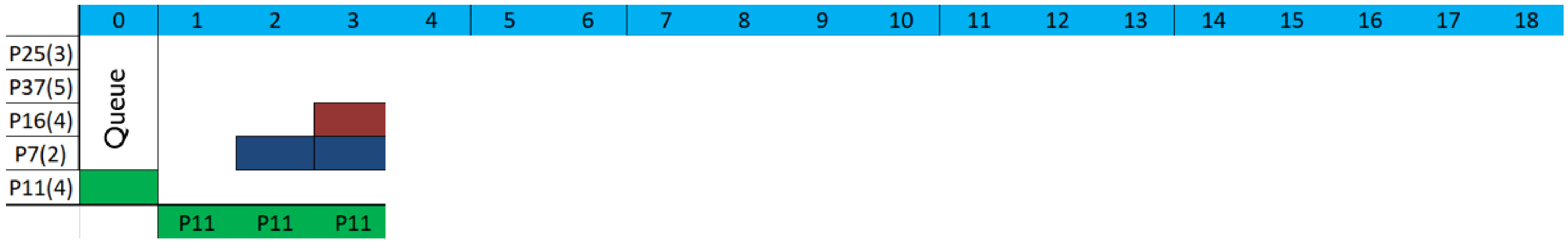
IN CLASS EXERCISE

Task : If SJF nonpreemptive CPU scheduling is used, then what is the average wait time for the 5 processes, and at what time does each process finish?

Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		

Average Wait Time	

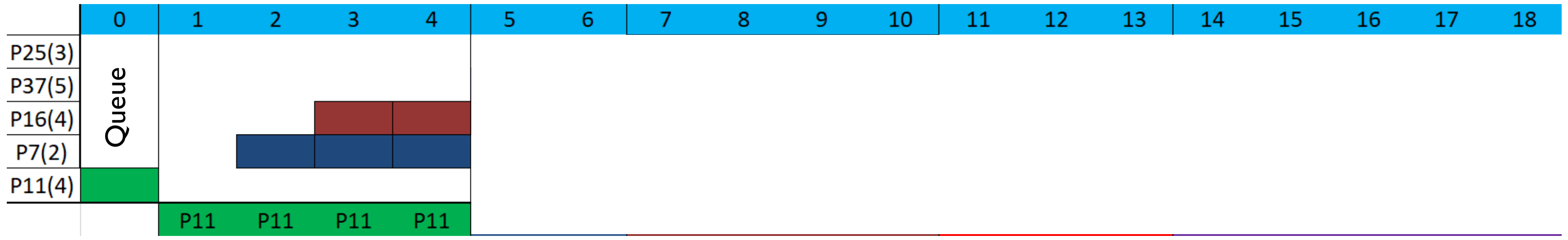
IN CLASS EXERCISE



Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		

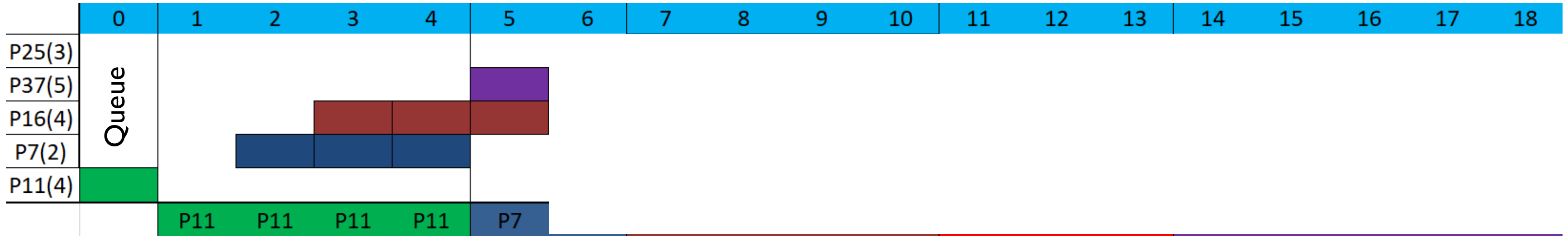
Average Wait Time	

IN CLASS EXERCISE



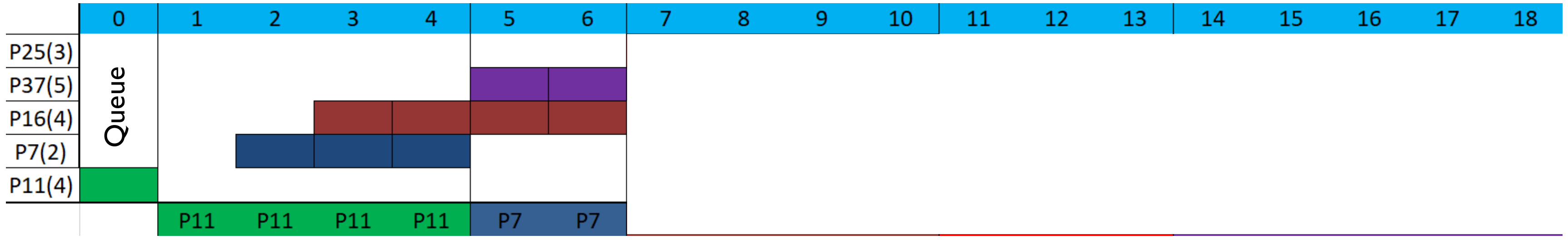
Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



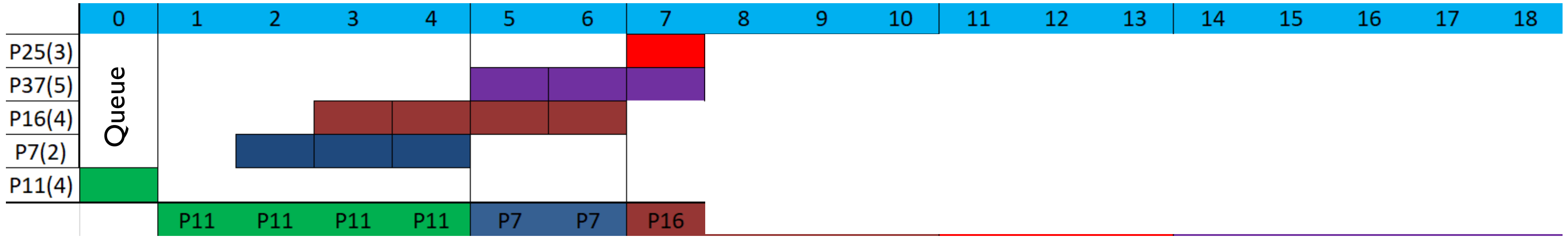
Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



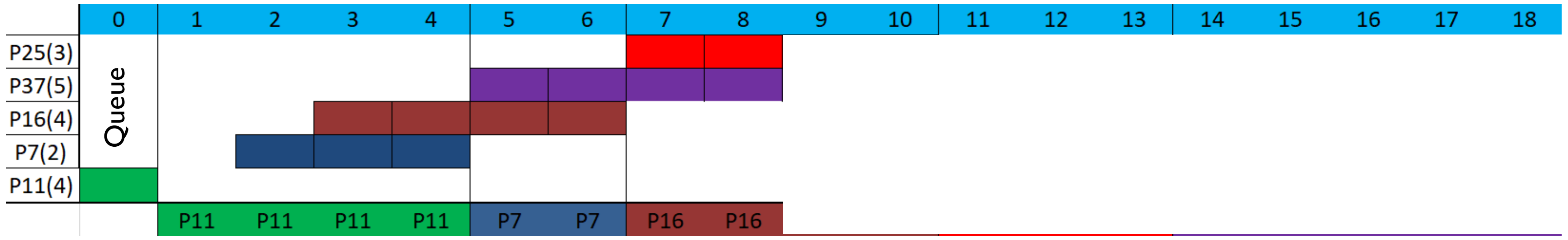
Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



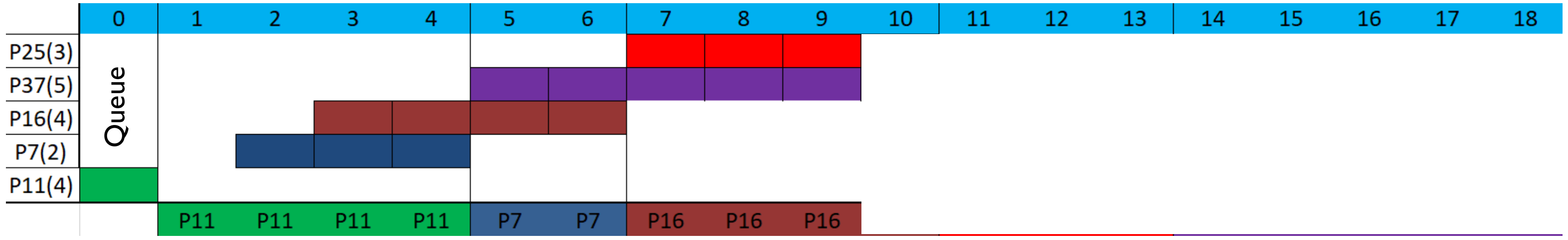
Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



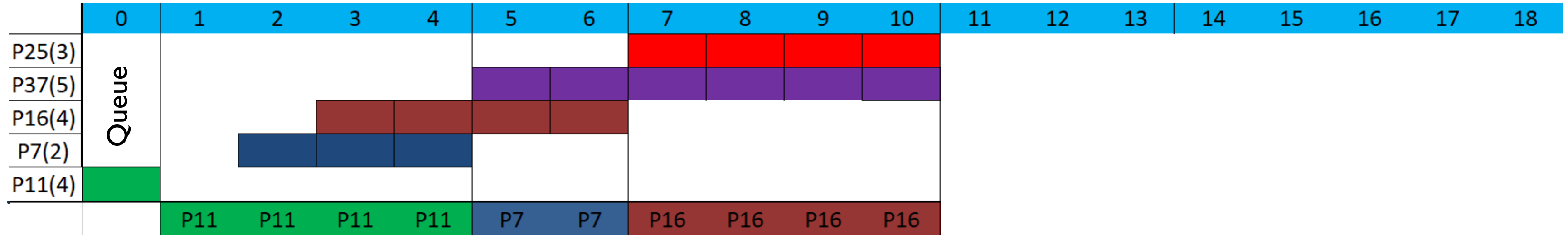
Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



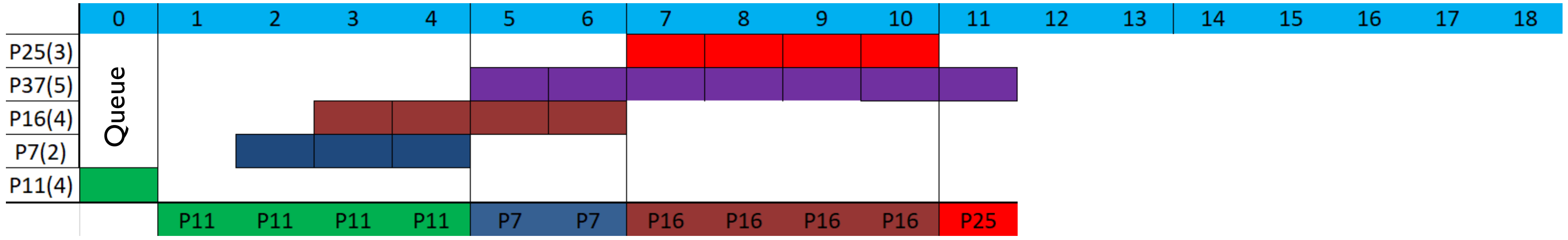
Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



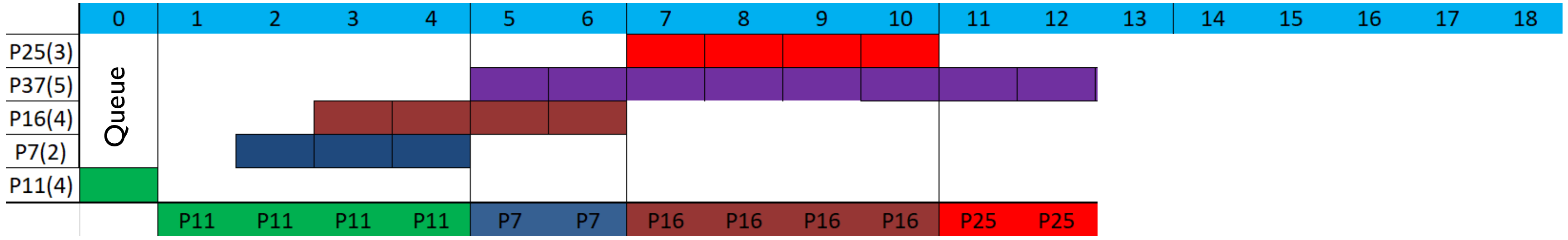
Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



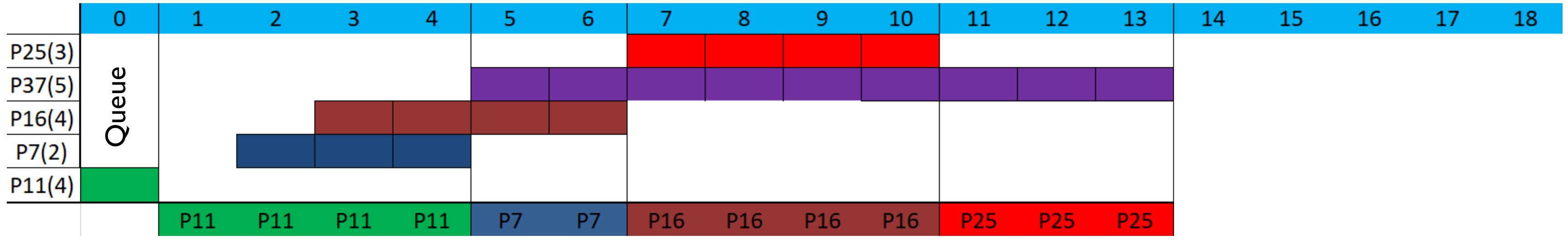
Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



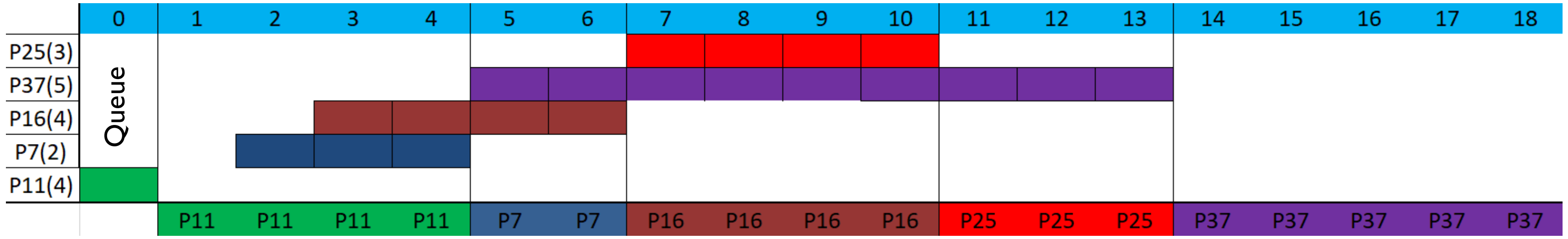
Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms		
P37	5ms	t=5ms		
P7	2ms	t=2ms		
P16	4ms	t=3ms		
P25	3ms	t=7ms		
	Average Wait Time			

IN CLASS EXERCISE



Process	Burst Duration	Arrival Time	Finish Time	Wait Time
P11	4ms	t=0ms	4	0
P37	5ms	t=5ms	18	9
P7	2ms	t=2ms	6	3
P16	4ms	t=3ms	10	4
P25	3ms	t=7ms	13	4
	Average Wait Time		4 ms	

PRIORITY SCHEDULING

- Shortest Time First is a special case of priority scheduling:
  - Process priority is the inverse of the process burst time.
- Priority can be established based on other factors.

PRIORITY SCHEDULING

Up until now, all scheduling schemes have assumed a nonpreemptive approach. A more typical approach is for a CPU/OS to employ a preemptive priority scheduling algorithm.

Q: If a preemptive scheduling algorithm is used, how does that affect the order/re-order of Process dispatch?

- Preemptive: CPU can stop the current burst.
- Non-Preemptive: CPU let the process finish its burst.

PRIORITY SCHEDULING

Q: Priorities based on what factor/criteria?

PRIORITY SCHEDULING

Q: Priorities based on what factor/criteria?

Internally defined priority : based on some measurable (computable) quantity … that the OS can reason about / calculate

PRIORITY SCHEDULING

Q: Priorities based on what factor/criteria?

Internally defined priority : based on some measurable (computable) quantity ... that the OS can reason about / calculate

Externally defined priority : based on criteria outside of the OS

Priority Scheduling

Assume 5 processes, their required burst times, and priorities*

P1	24 ms	1
P2	16 ms	3
P3	2 ms	5
P4	4 ms	2
P5	3 ms	4

*Low numbers refer to high priorities

If P1-P5 at t=0 are all waiting to be dispatched, and using a nonpreemptive scheduling scheme ..

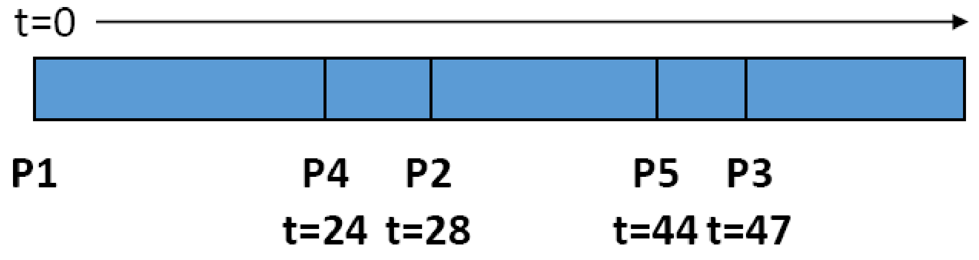
Priority Scheduling

Assume 5 processes, their required burst times, and priorities*

P1	24 ms	1
P2	16 ms	3
P3	2 ms	5
P4	4 ms	2
P5	3 ms	4

*Low numbers refer to high priorities

If P1-P5 at t=0 are all waiting to be dispatched, and using a nonpreemptive scheduling scheme ..



PRIORITY SCHEDULING

- Preemptive: CPU can stop the current burst.
- Non-Preemptive: CPU let the process finish its burst.

Up until now, all scheduling schemes have assumed a nonpreemptive approach. A more typical approach is for a CPU/OS to employ a preemptive priority scheduling algorithm.

Q: If a preemptive scheduling algorithm is used, how does that affect the order/re-order of Process dispatch?

PRIORITY SCHEDULING

- Preemptive: CPU can stop the current burst.
- Non-Preemptive: CPU let the process finish its burst.

Up until now, all scheduling schemes have assumed a nonpreemptive approach. A more typical approach is for a CPU/OS to employ a preemptive priority scheduling algorithm.

Q: If a preemptive scheduling algorithm is used, how does that affect the order/re-order of Process dispatch?

Processes will be interrupted before they finish their burst.

PREEMPTIVE PRIORITY SCHEDULING EXERCISE

- Assume 5 processes and their required burst times, arrival times, and priorities shown in the table below.
- Q: If ***preemptive*** *(processes can be removed before completion)* priority scheduling is used, then at what time does each process finish? What is the total time of each process? How many context switch(es) does each process experience?

Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Queue	P1																	
	P2	3																
	P3																	
	P4																	
	P5																	
CPU																		

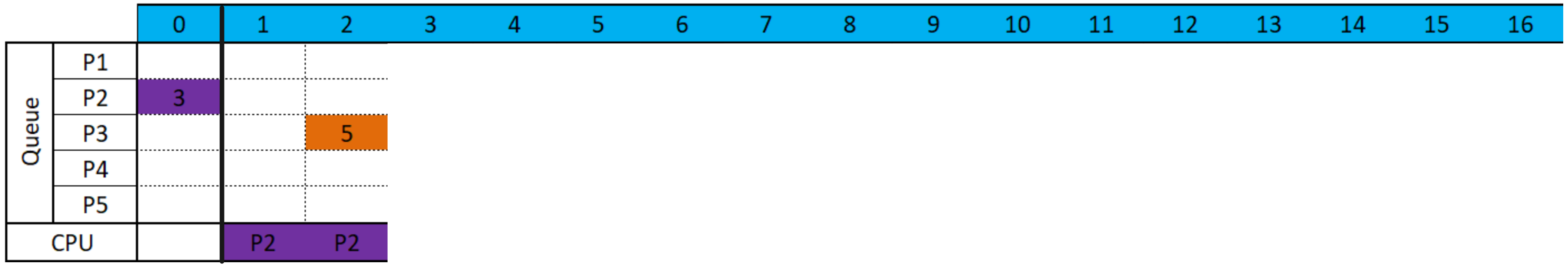
Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Queue	P1																	
	P2	3																
	P3																	
	P4																	
	P5																	
CPU			P2															

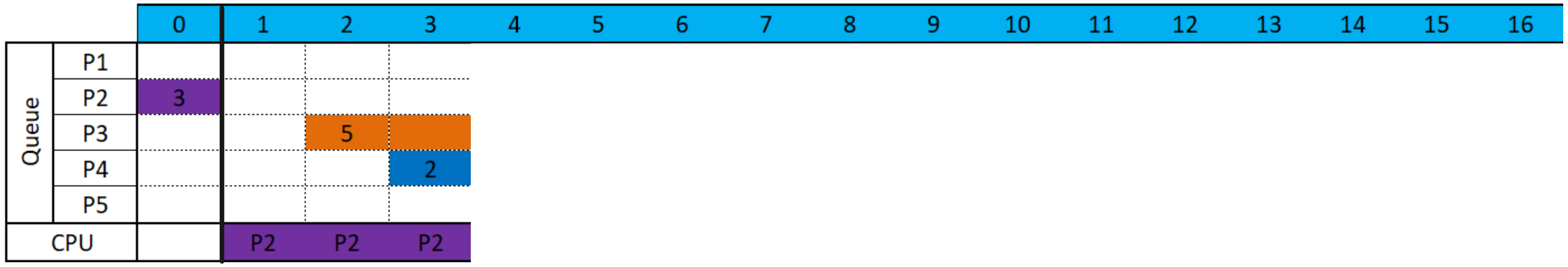
Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE



Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

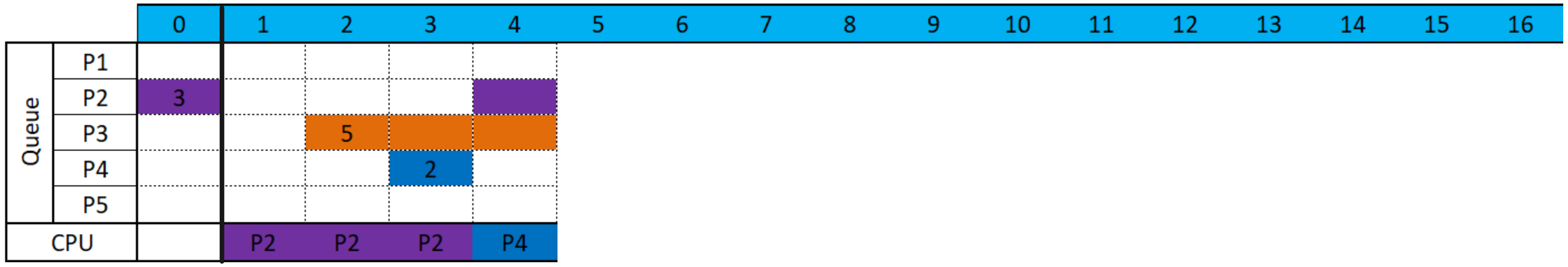
PREEMPTIVE PRIORITY SCHEDULING EXERCISE



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Queue	P1																	
	P2	3																
	P3			5	🟧													
	P4				2													
	P5																	
CPU			P2	P2	P2													

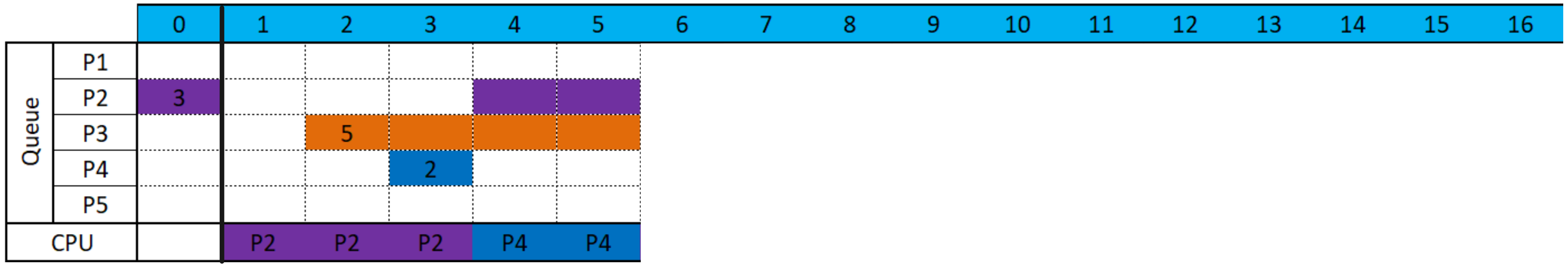
Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE



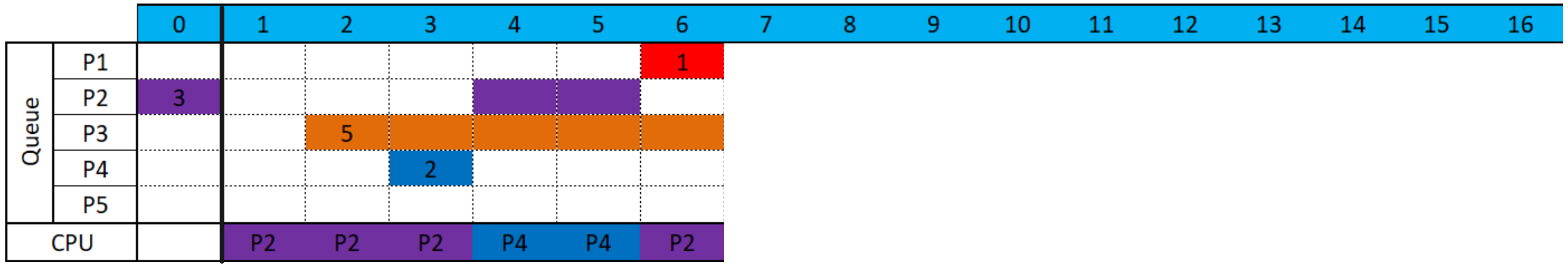
Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE



Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

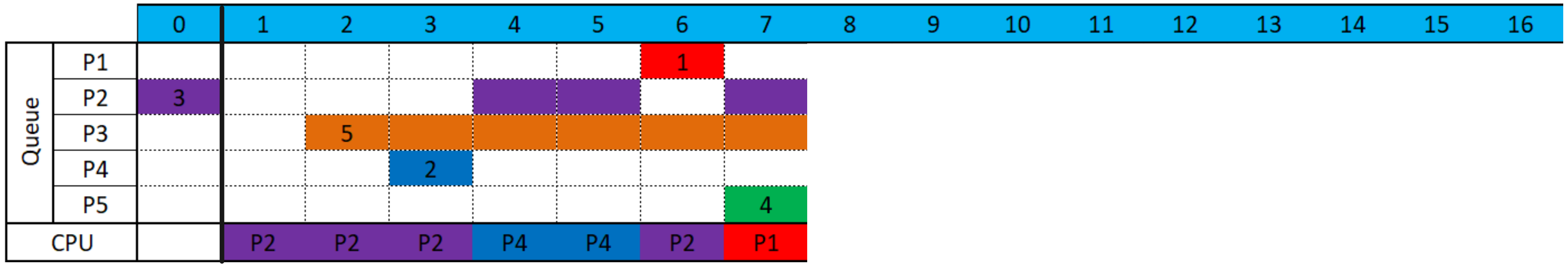
PREEMPTIVE PRIORITY SCHEDULING EXERCISE



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Queue	P1							1										
	P2	3				🟪	🟪											
	P3			5	🟧	🟧	🟧	🟧										
	P4				2													
	P5																	
CPU			P2	P2	P2	P4	P4	P2										

Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

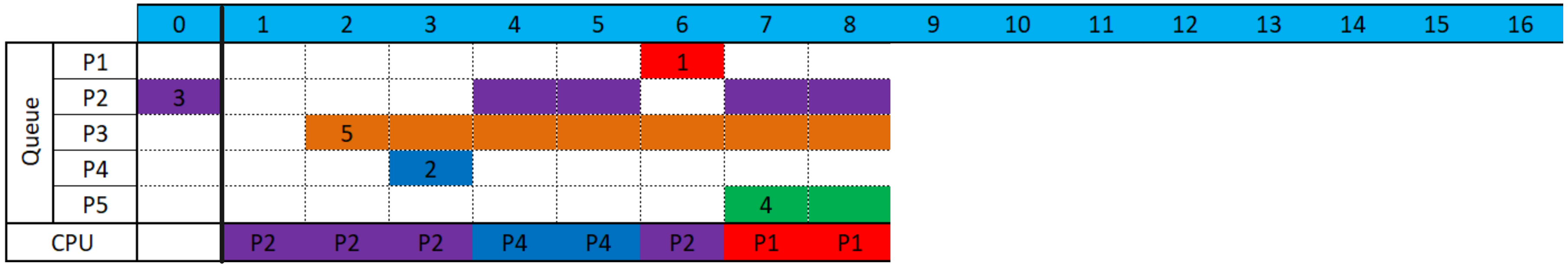
PREEMPTIVE PRIORITY SCHEDULING EXERCISE



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Queue	P1							1										
	P2	3				▨	▨		▨									
	P3			5	▨	▨	▨	▨	▨									
	P4				2													
	P5								4									
CPU			P2	P2	P2	P4	P4	P2	P1									

Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

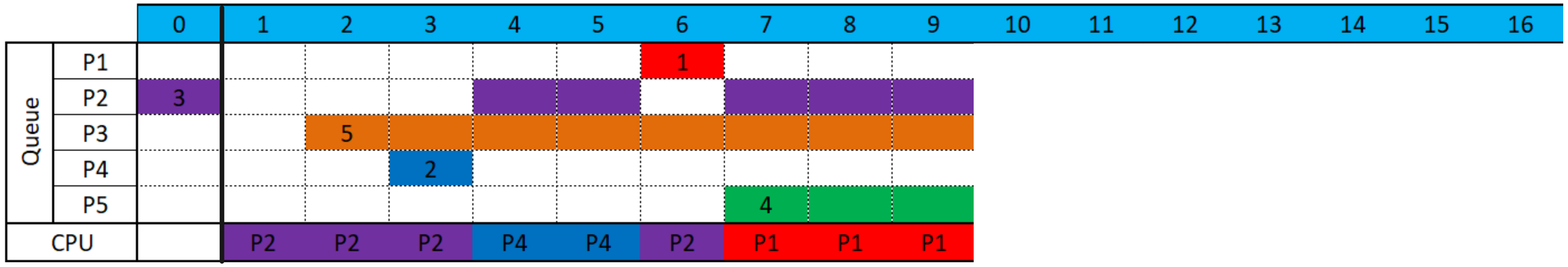
PREEMPTIVE PRIORITY SCHEDULING EXERCISE



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Queue	P1							1										
	P2	3				🟪	🟪		🟪	🟪								
	P3			5	🟧	🟧	🟧	🟧	🟧	🟧								
	P4				2													
	P5								4	🟩								
CPU			P2	P2	P2	P4	P4	P2	P1	P1								

Process	Burst Time	Priority	Arrival	Completio n Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Queue	P1							1										
	P2	3				🟪	🟪		🟪	🟪	🟪							
	P3			5	🟧	🟧	🟧	🟧	🟧	🟧	🟧							
	P4				2													
	P5								4	🟩	🟩							
CPU			P2	P2	P2	P4	P4	P2	P1	P1	P1							

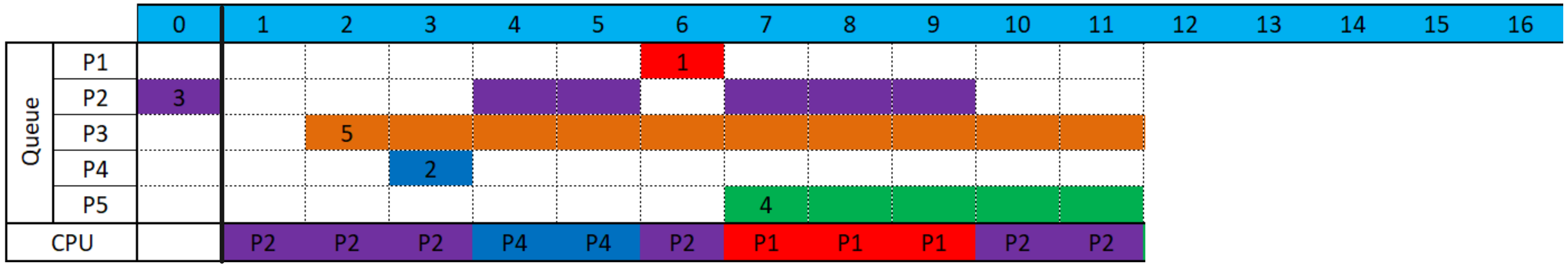
Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6			
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Queue	P1							🟥 1										
	P2	🟪 3				🟪	🟪		🟪	🟪	🟪							
	P3			🟧 5	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧						
	P4				🟦 2													
	P5								🟩 4	🟩	🟩	🟩						
CPU			P2	P2	P2	P4	P4	P2	P1	P1	P1	P2						

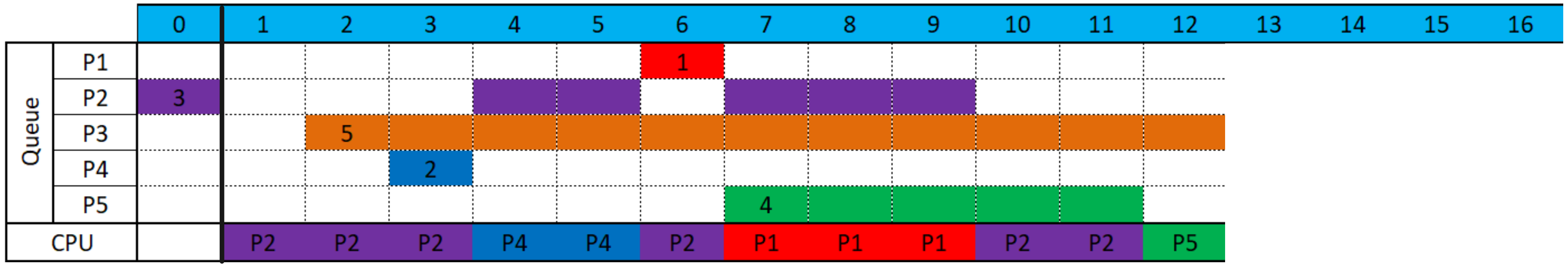
Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6	9		
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE



Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6	9		
P2	6	3	t=0			
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

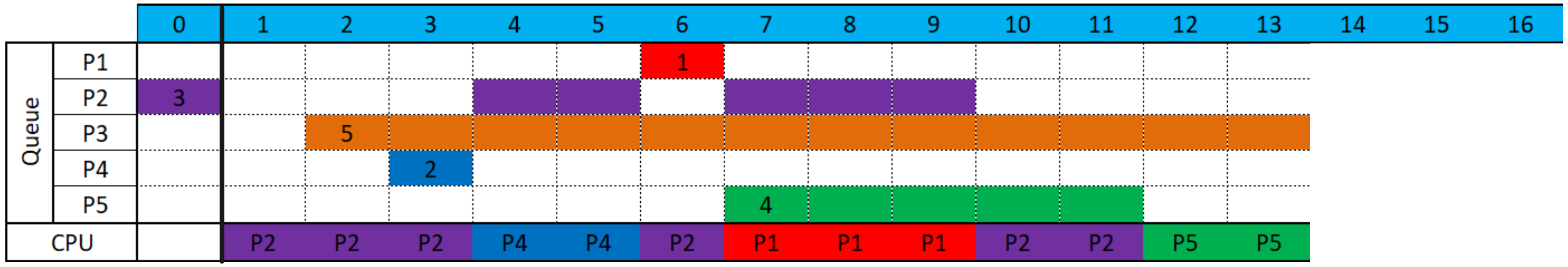
PREEMPTIVE PRIORITY SCHEDULING EXERCISE



Queue	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P1							1										
P2	3				🟪	🟪		🟪	🟪	🟪							
P3			5	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧				
P4				2													
P5								4	🟩	🟩	🟩	🟩					
CPU		P2	P2	P2	P4	P4	P2	P1	P1	P1	P2	P2	P5				

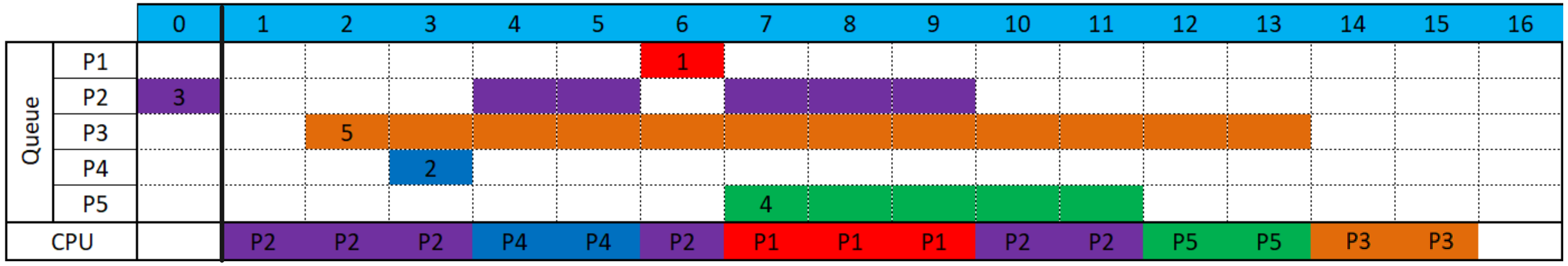
Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6	9		
P2	6	3	t=0	11		
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE



Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6	9		
P2	6	3	t=0	11		
P3	2	5	t=2			
P4	2	2	t=3			
P5	2	4	t=7			

PREEMPTIVE PRIORITY SCHEDULING EXERCISE



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Queue	P1							1										
	P2	3				🟪	🟪		🟪	🟪	🟪							
	P3			5	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧			
	P4				2													
	P5								4	🟩	🟩	🟩	🟩					
CPU			P2	P2	P2	P4	P4	P2	P1	P1	P1	P2	P2	P5	P5	P3	P3	

Process	Burst Time	Priority	Arrival	Completion Time	Wait time	# of Context Switch(es)
P1	3	1	t=6	9	1	1
P2	6	3	t=0	11	5	3
P3	2	5	t=2	15	12	1
P4	2	2	t=3	5	1	1
P5	2	4	t=7	13	5	1

ROUND ROBIN SCHEDULING

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

ROUND ROBIN SCHEDULING

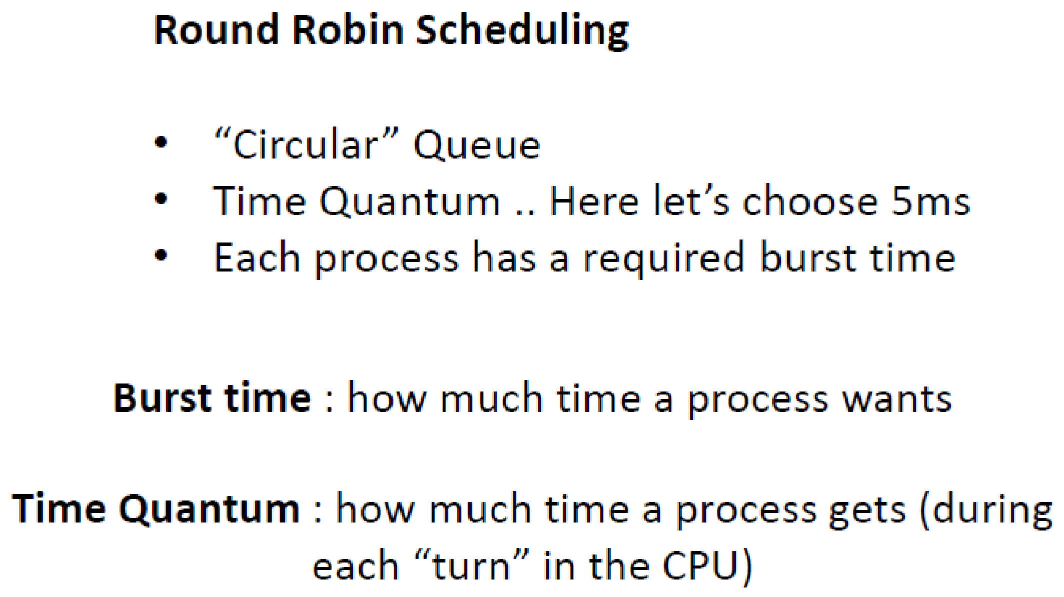
Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Burst time : how much time a process wants

Time Quantum : how much time a process gets (during each "turn" in the CPU)

ROUND ROBIN SCHEDULING



Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Burst time : how much time a process wants

Time Quantum : how much time a process gets (during each "turn" in the CPU)

No process can utilize more than the time quantum in the CPU. It has to yield to others.

ROUND ROBIN SCHEDULING

Turnaround time

P3

P32

P7

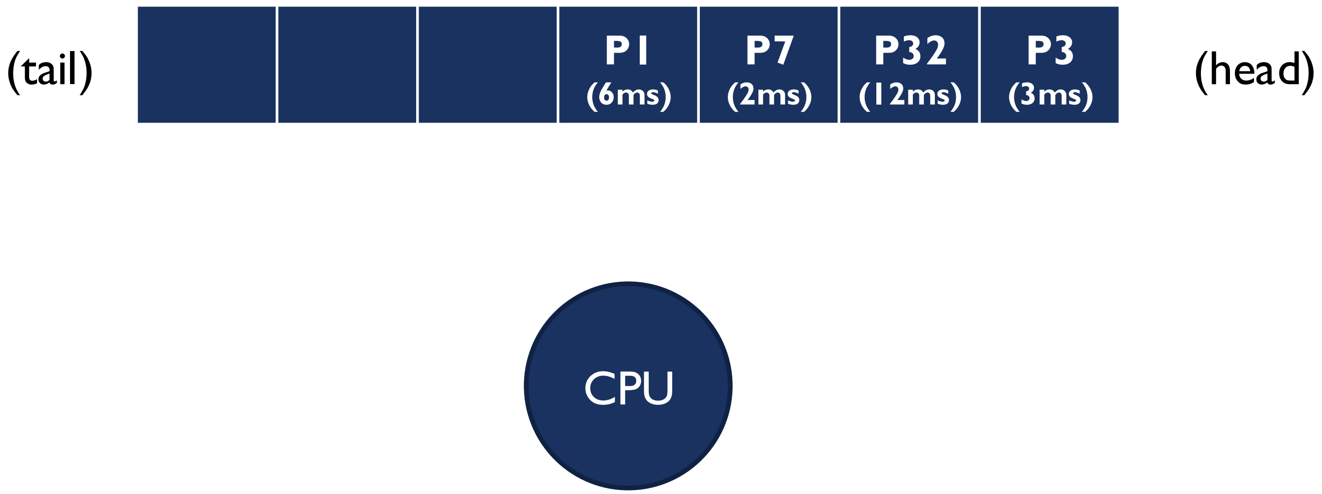
P1

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 0

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

P3

P32

P7

P1

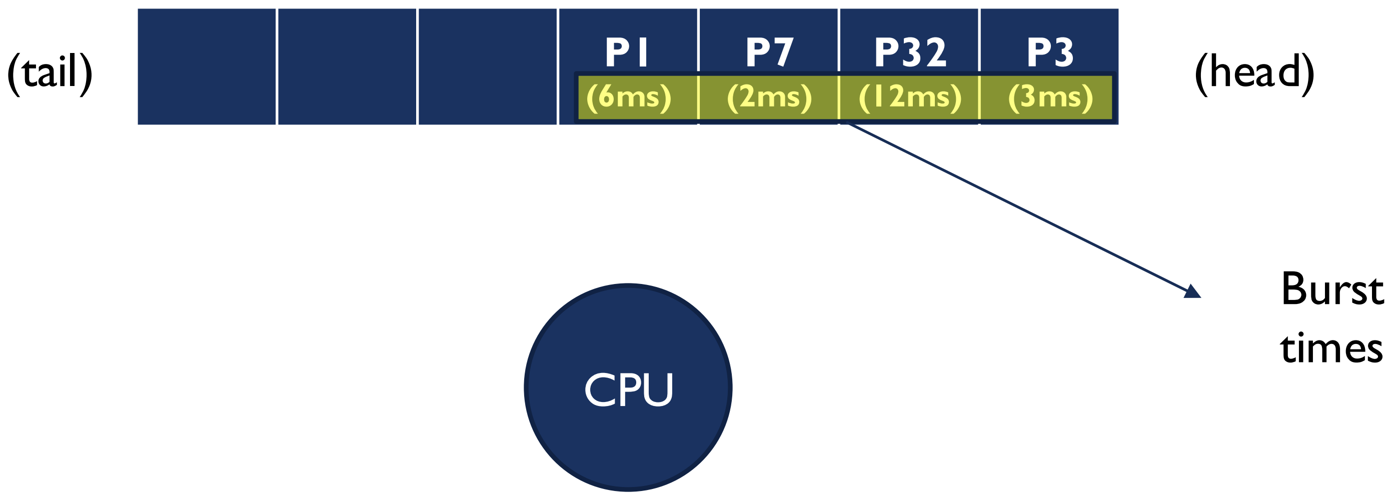
Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 0

Quantum

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

P3

P32

P7

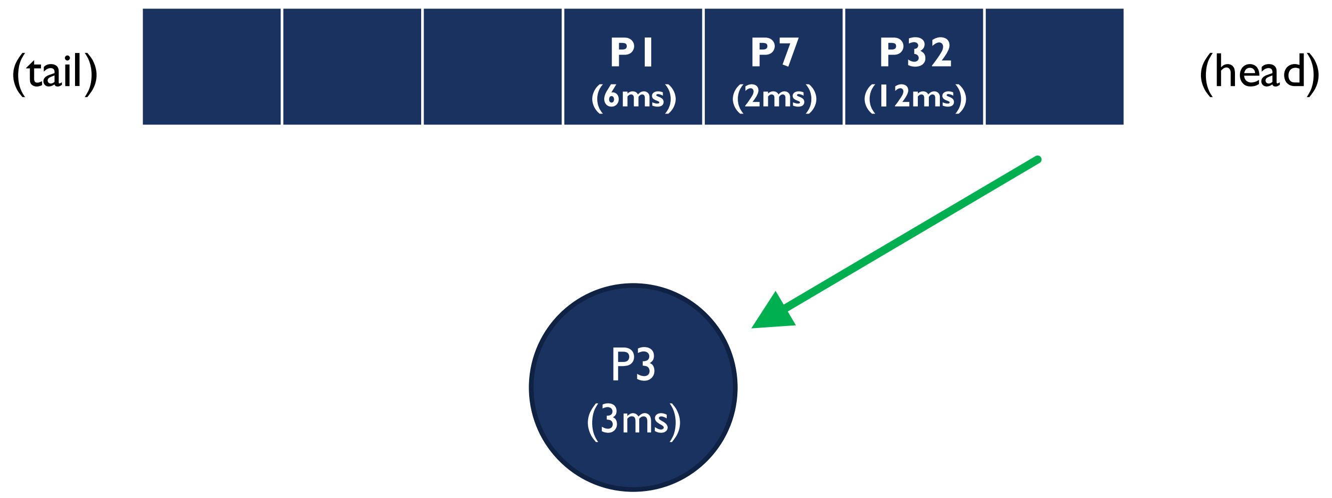
P1

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

FIFO Ready "circular" Queue

Clock tick: 0



ROUND ROBIN SCHEDULING

Turnaround time

P3

P32

P7

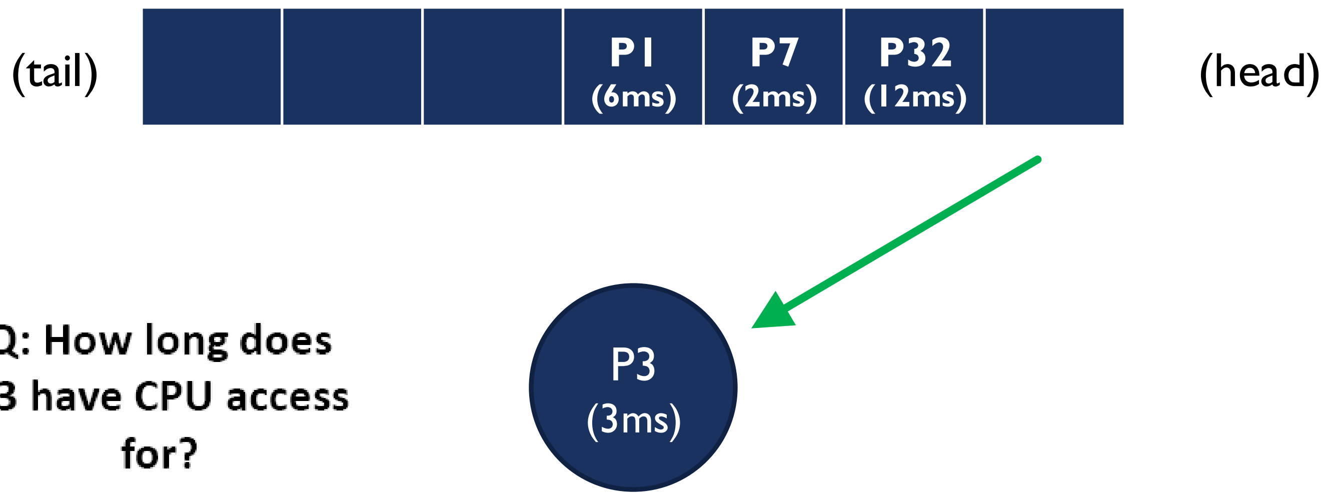
P1

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 0

FIFO Ready "circular" Queue



Q: How long does P3 have CPU access for?

ROUND ROBIN SCHEDULING

Turnaround time

P3

P32

P7

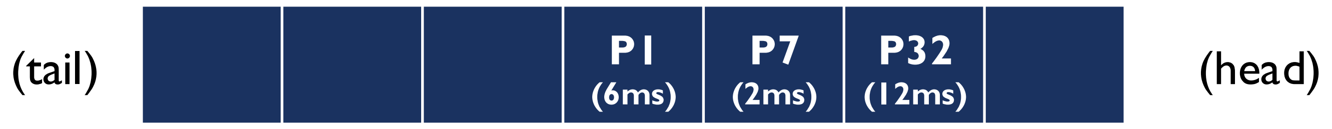
P1

Round Robin Scheduling

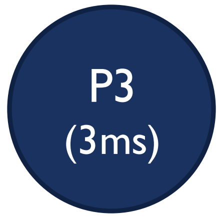
- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 0

FIFO Ready "circular" Queue



Q: What is the turnaround time for P3?



ROUND ROBIN SCHEDULING

Turnaround time

P3

P32

P7

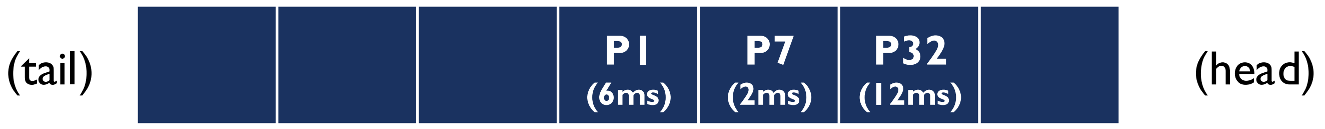
P1

Round Robin Scheduling

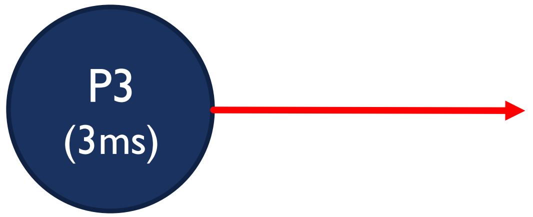
- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 3

FIFO Ready "circular" Queue



Q: What is the turnaround time for P3?



ROUND ROBIN SCHEDULING

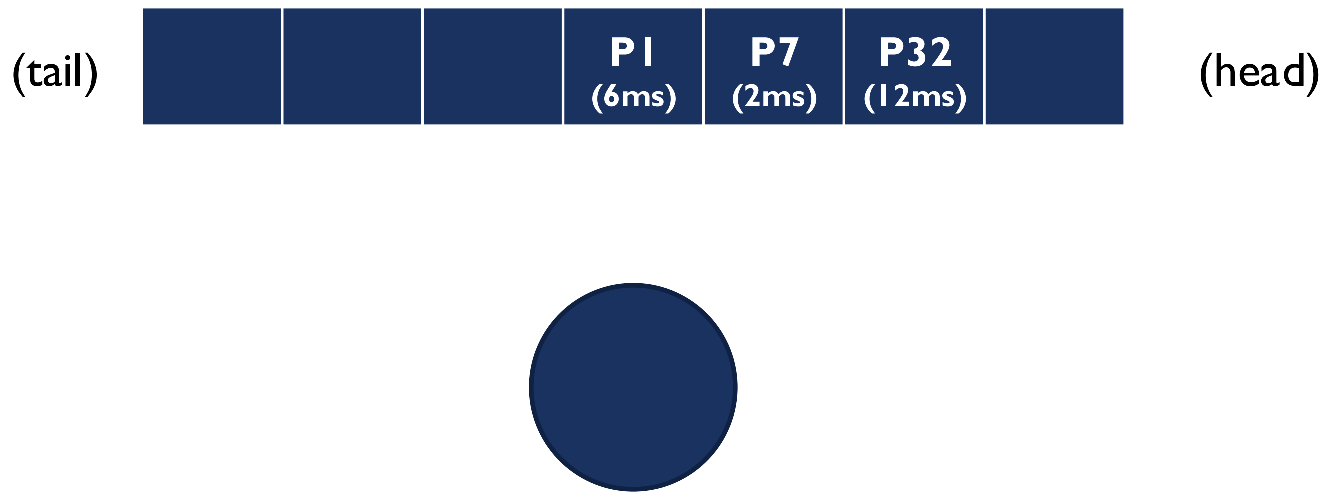
Turnaround time

P3	3ms
P32	
P7	
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 3



ROUND ROBIN SCHEDULING

Turnaround time

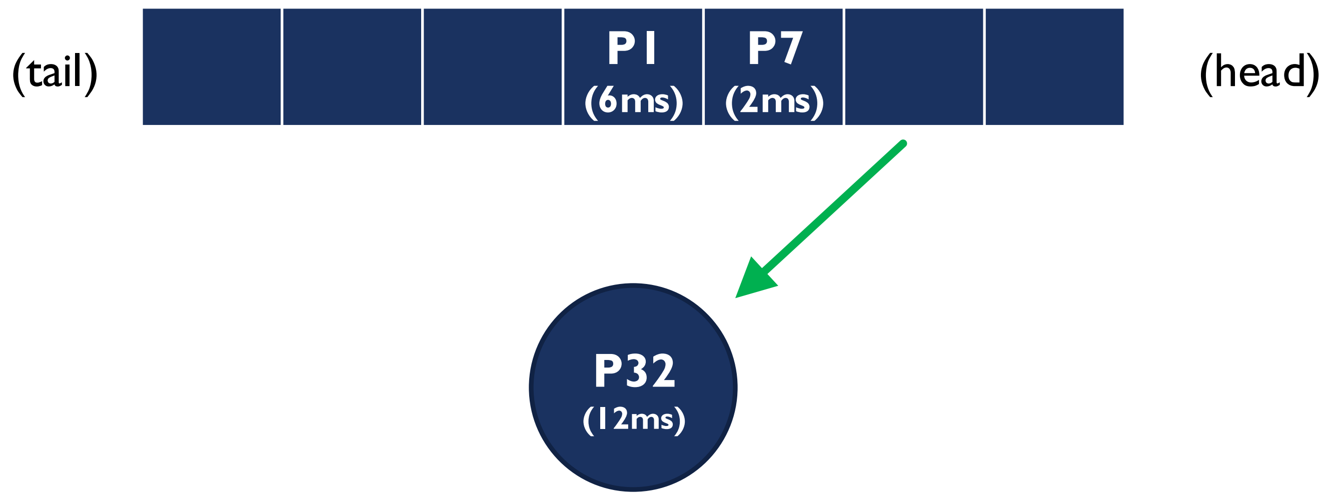
P3	3ms
P32	
P7	
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 3

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

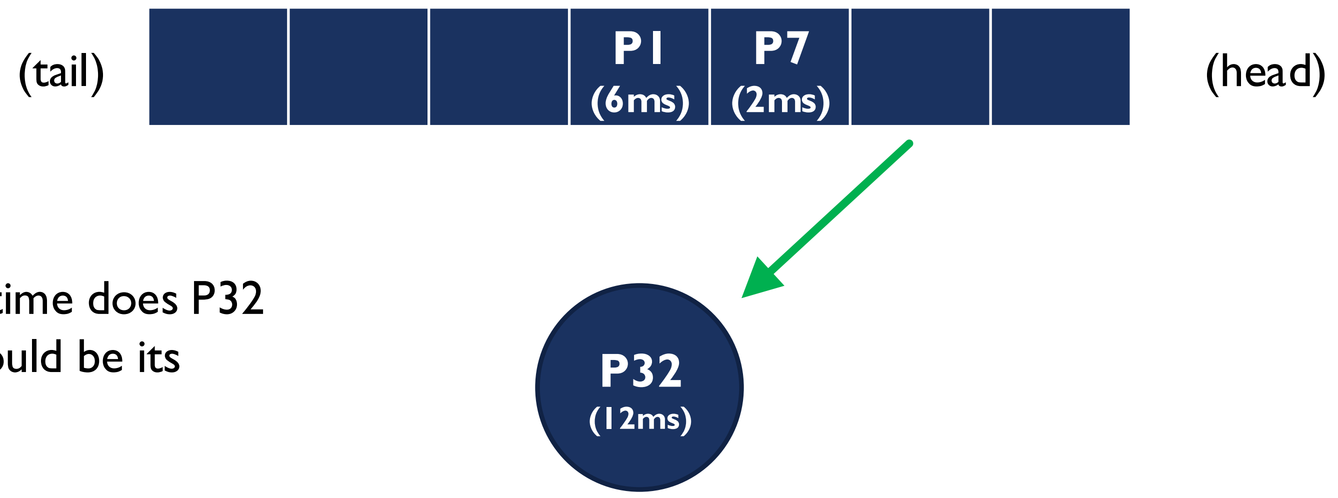
P3	3ms
P32	
P7	
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 3

FIFO Ready "circular" Queue



Worksheet Q1: At what time does P32 Leave the CPU? What would be its remaining CPU time?

ROUND ROBIN SCHEDULING

Turnaround time

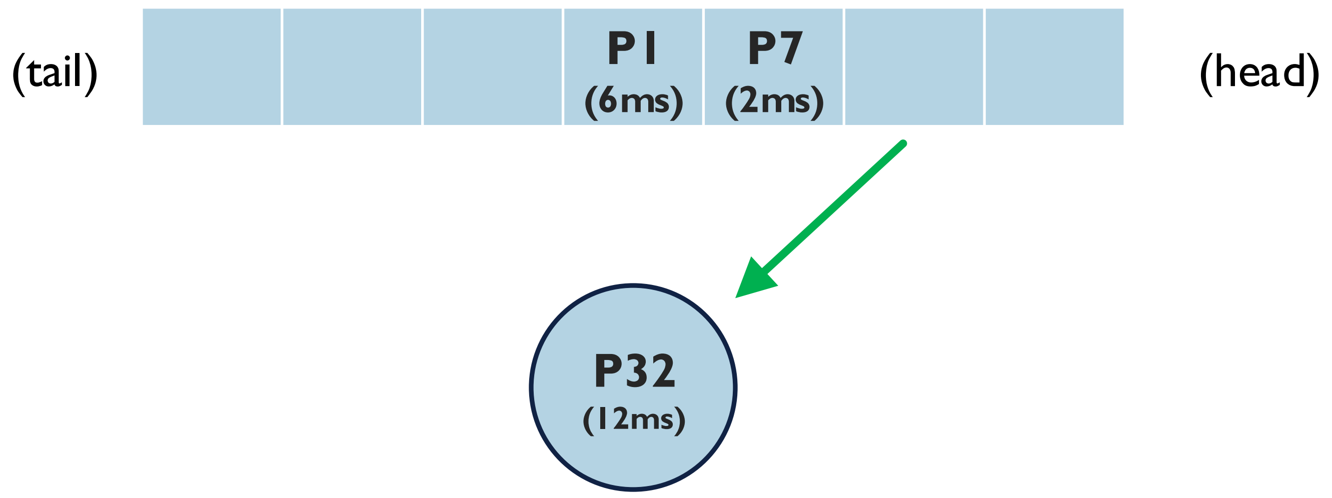
P3	3ms
P32	
P7	
P1	

Round Robin Scheduling

Clock tick: 3

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

FIFO Ready "circular" Queue



Worksheet Q1: At what time does P32 Leave the CPU? What would be its remaining CPU time?

ROUND ROBIN SCHEDULING

Turnaround time

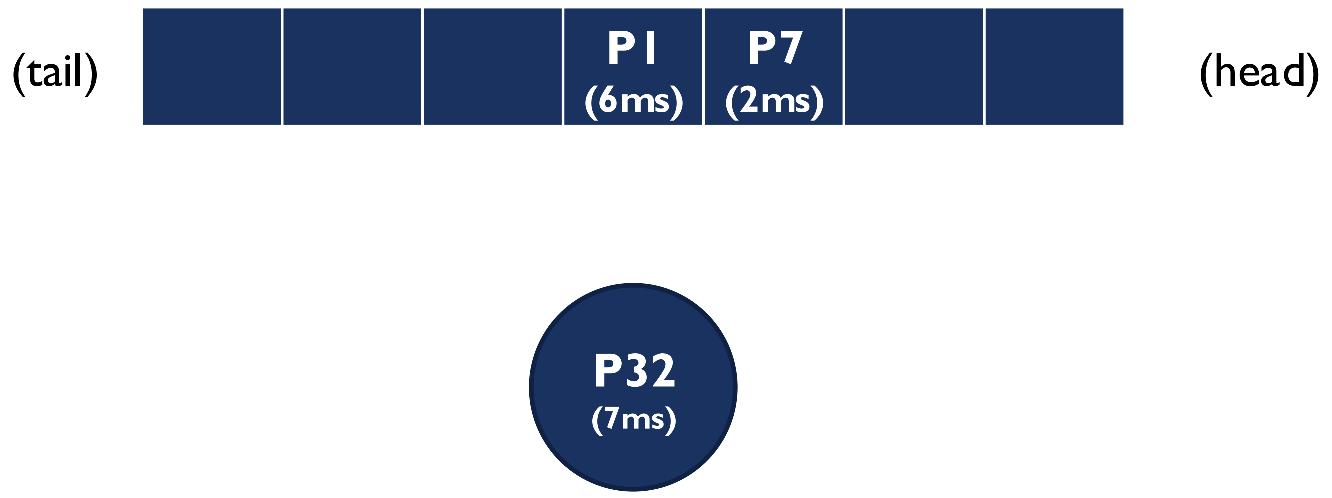
P3	3ms
P32	
P7	
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 8

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

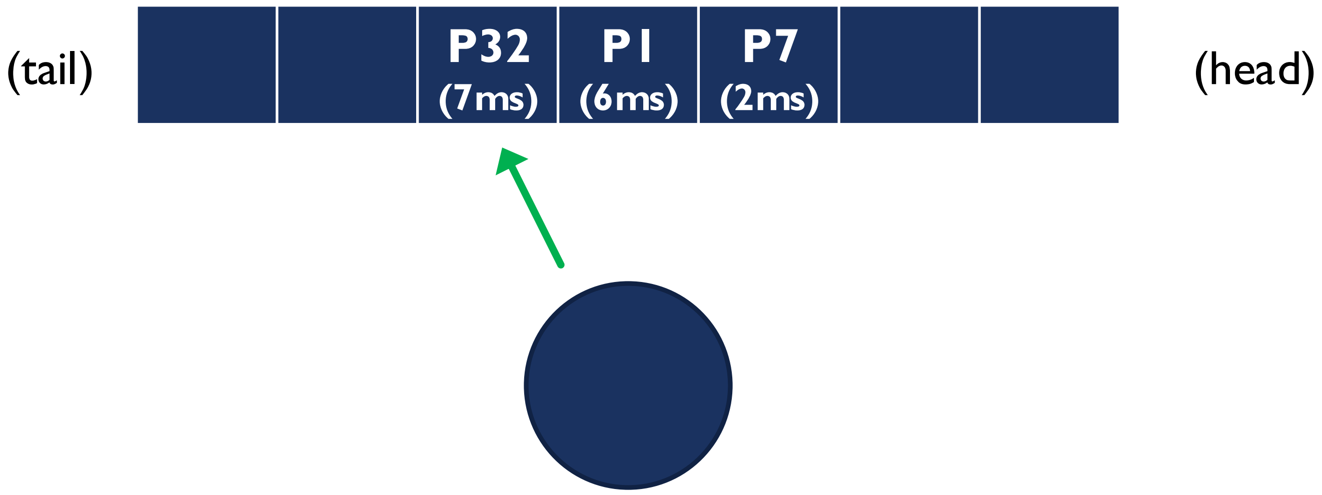
P3	3ms
P32	
P7	
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 8

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

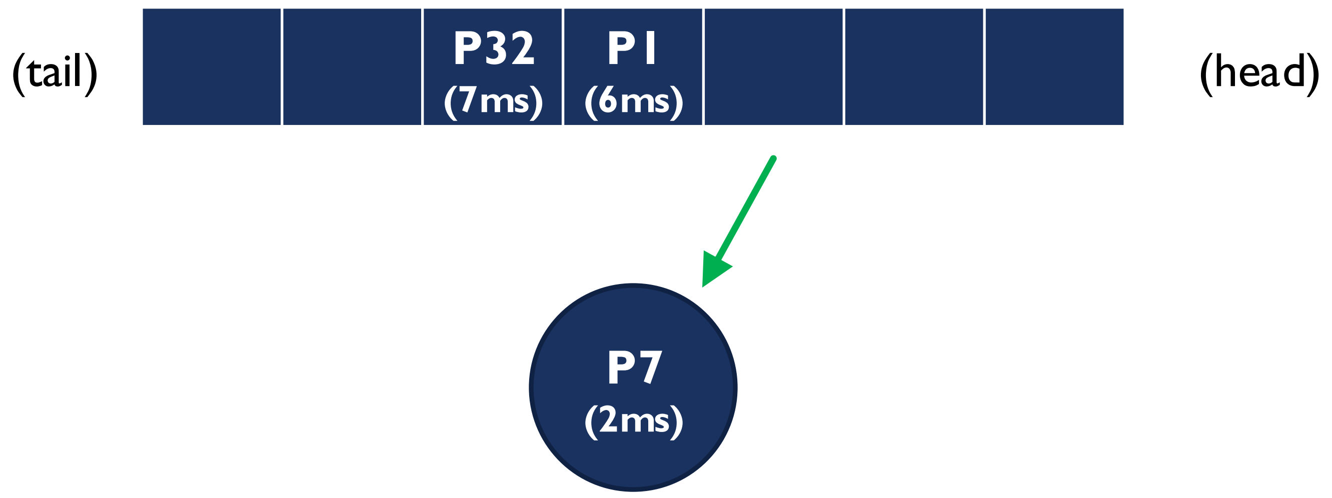
P3	3ms
P32	
P7	
P1	

Round Robin Scheduling

Clock tick: 8

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

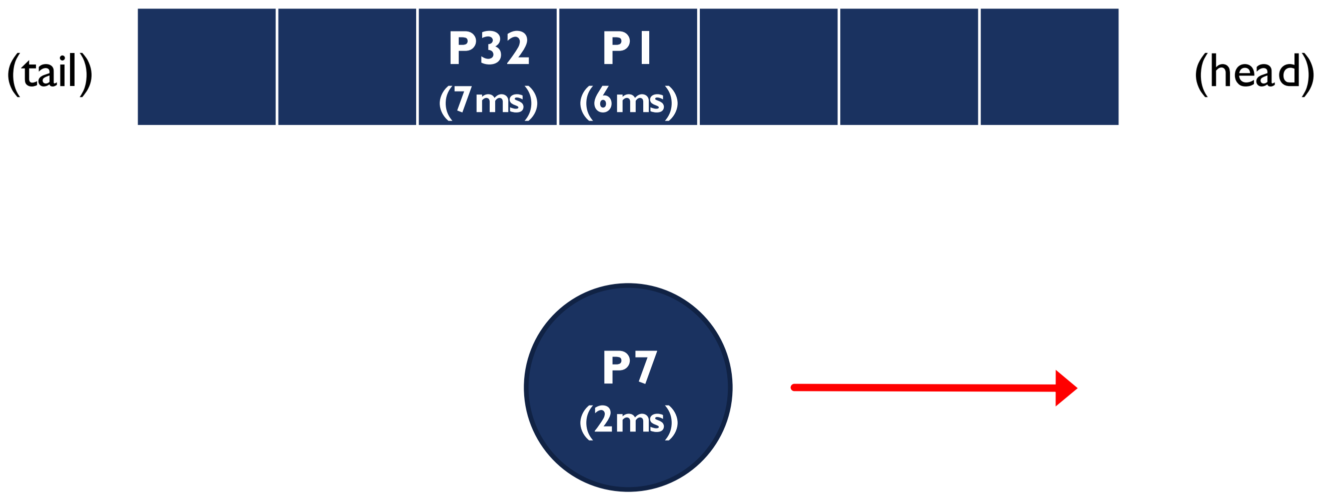
P3	3ms
P32	
P7	10ms
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
10

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

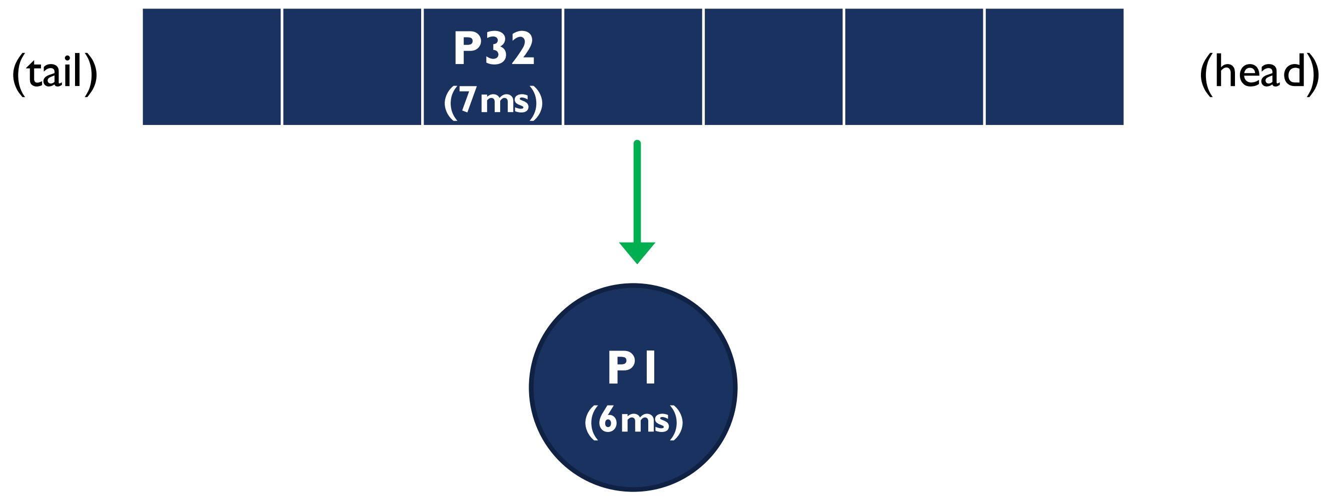
P3	3ms
P32	
P7	10ms
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
10

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

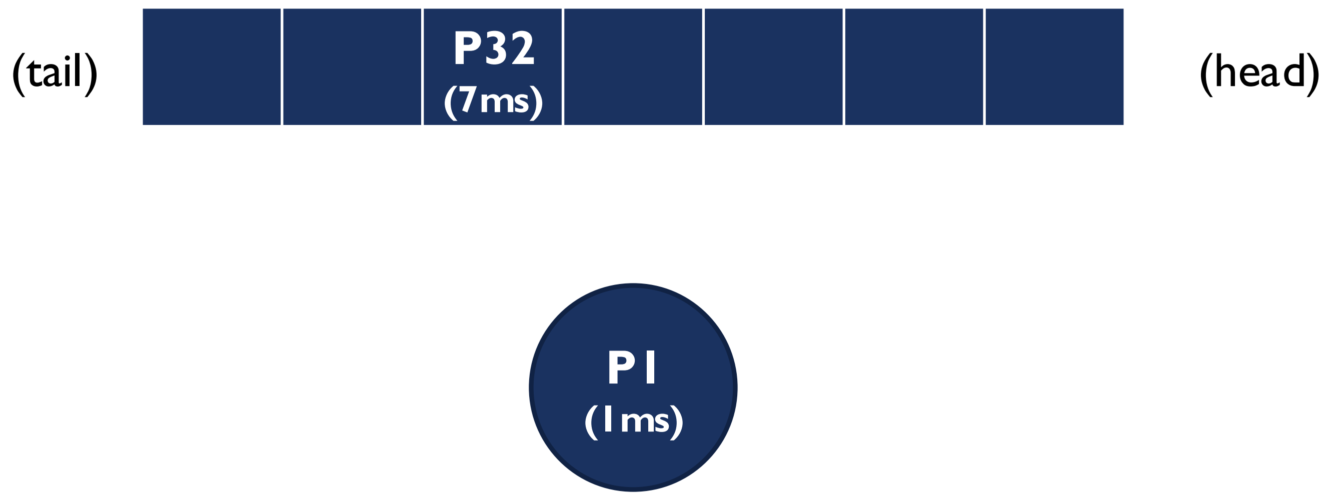
P3	3ms
P32	
P7	10ms
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
15

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

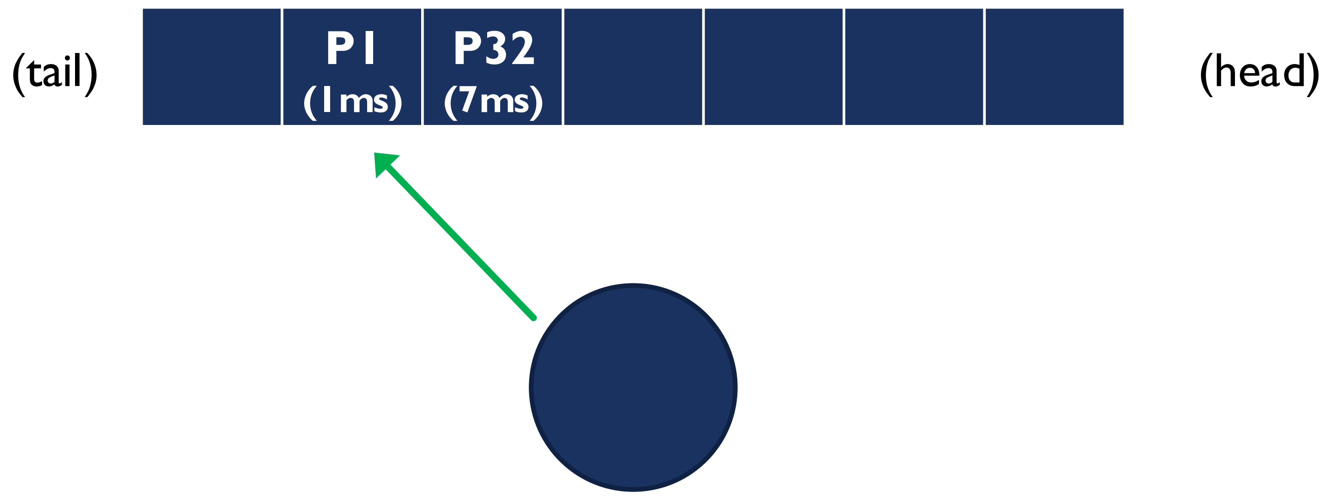
P3	3ms
P32	
P7	10ms
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
15

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

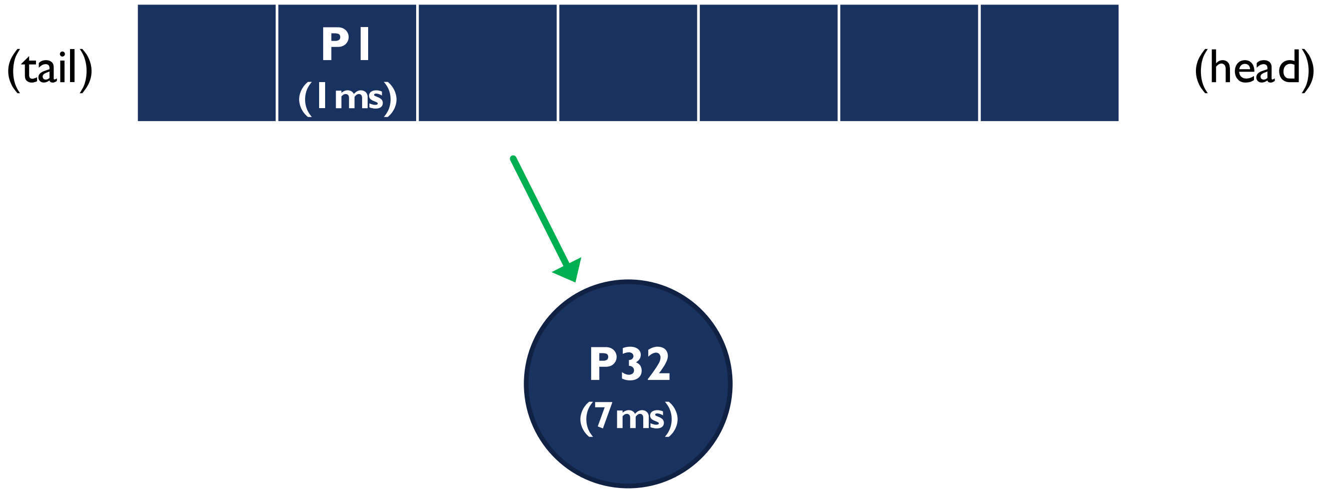
P3	3ms
P32	
P7	10ms
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
15

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

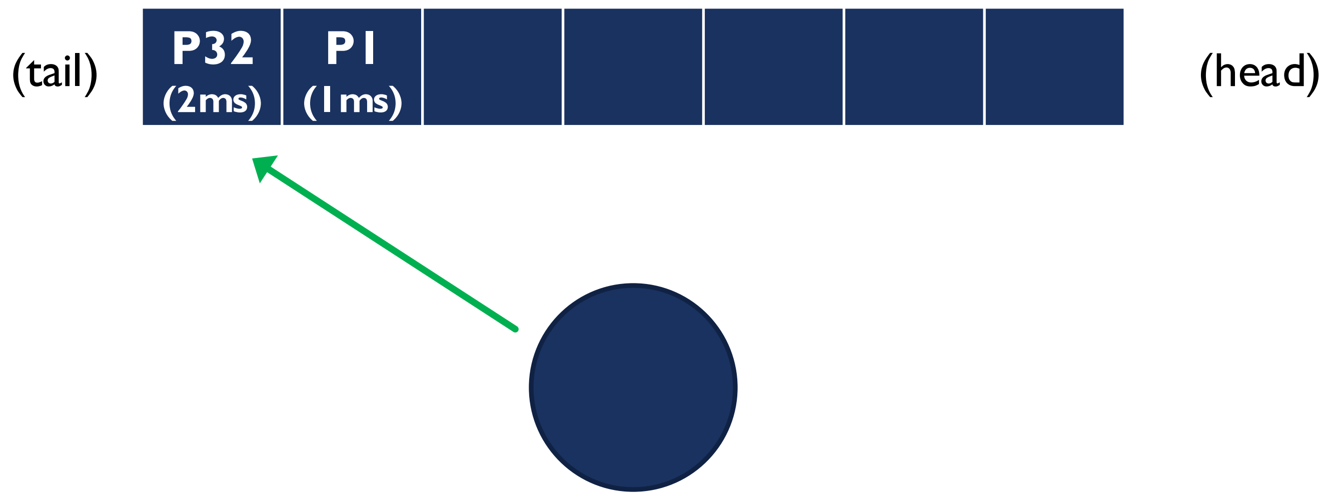
P3	3ms
P32	
P7	10ms
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
20

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

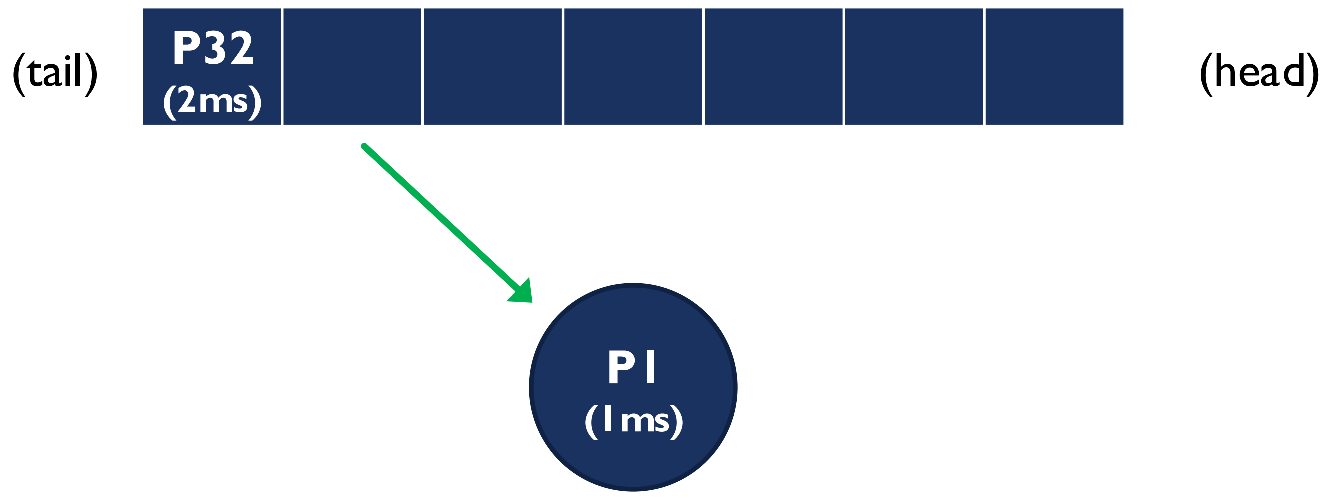
P3	3ms
P32	
P7	10ms
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
20

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

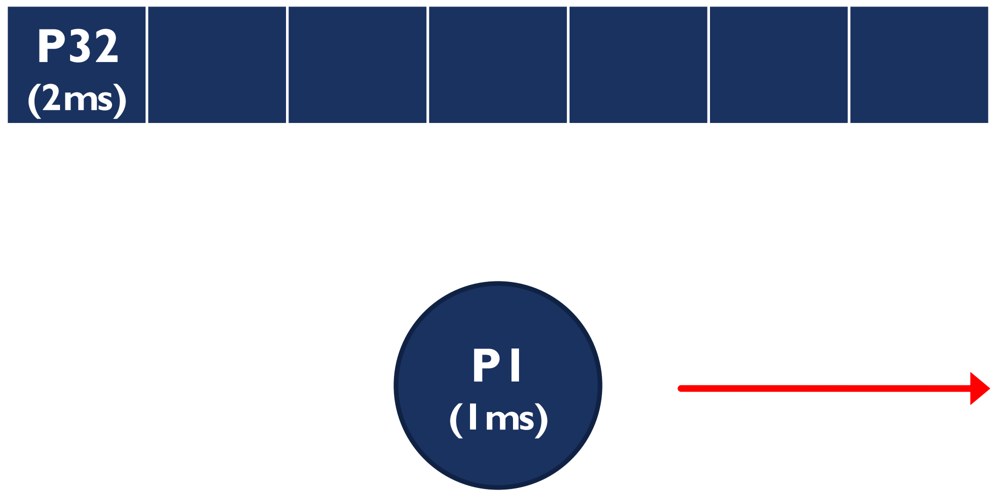
P3	3ms
P32	
P7	10ms
P1	

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
21

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

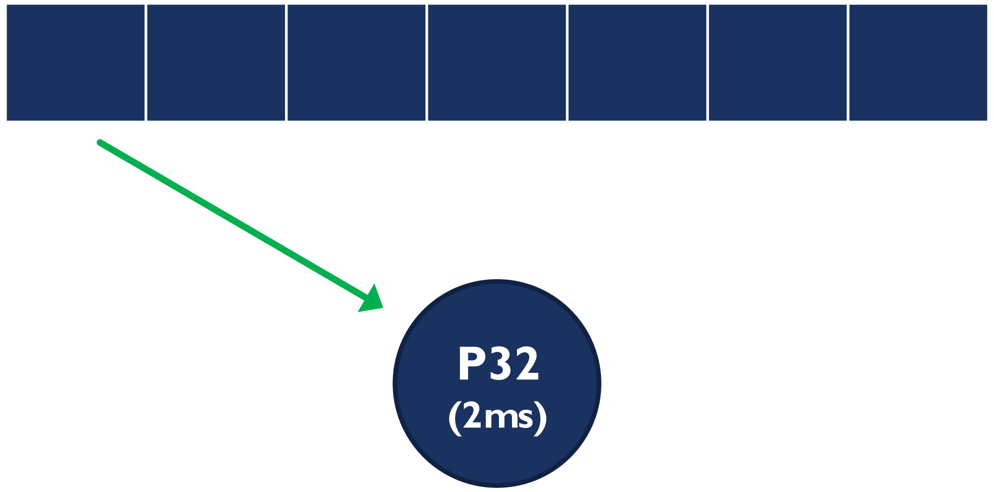
P3	3ms
P32	
P7	10ms
P1	21ms

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
21

FIFO Ready "circular" Queue



ROUND ROBIN SCHEDULING

Turnaround time

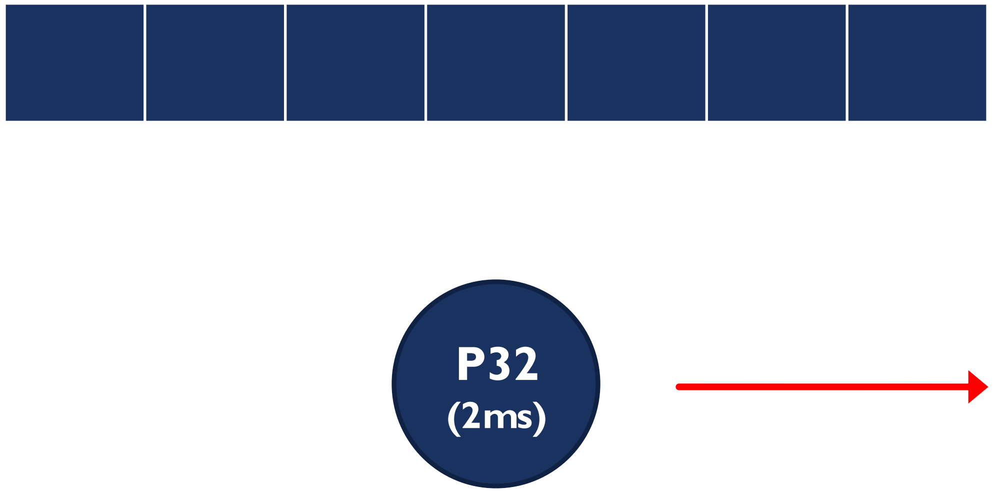
P3	3ms
P32	23ms
P7	10ms
P1	21ms

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick:
23

FIFO Ready "circular" Queue



Q2:What is the disadvantage of Round Robin Scheduling? How to mitigate it?

ROUND ROBIN SCHEDULING

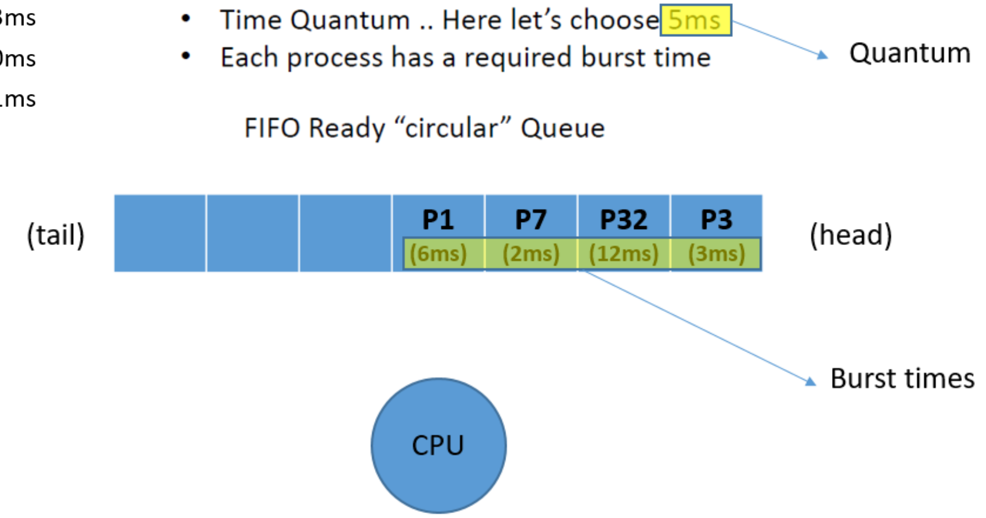
Turnaround time

P3	3ms
P32	23ms
P7	10ms
P1	21ms

Round Robin Scheduling

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time

Clock tick: 0



ROUND ROBIN SCHEDULING

- The time quantum should be large compared with the context switch time
- However, if the time quantum is too large, RR scheduling degenerates to an FCFS policy.
- A rule of thumb is that 80 percent of the CPU bursts should be shorter than the time quantum.

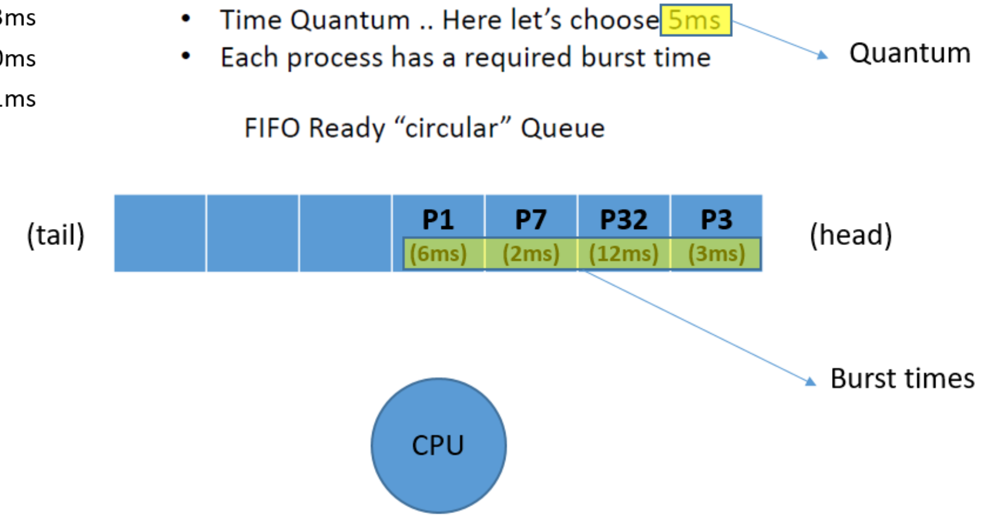
Turnaround time

P3	3ms
P32	23ms
P7	10ms
P1	21ms

Round Robin Scheduling

Clock tick: 0

- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time



ROUND ROBIN SCHEDULING

- The time quantum should be large compared with the context switch time
- However, if the time quantum is too large, RR scheduling degenerates to an FCFS policy.
- A rule of thumb is that 80 percent of the CPU bursts should be shorter than the time quantum.

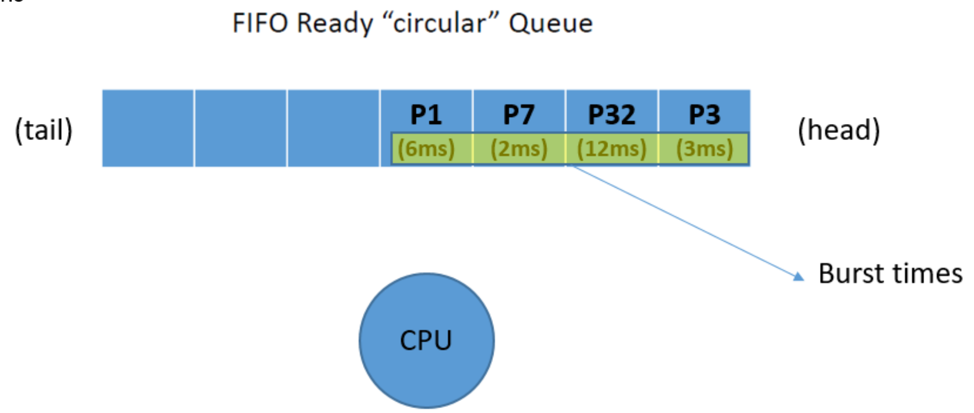
Turnaround time

P3	3ms
P32	23ms
P7	10ms
P1	21ms

Round Robin Scheduling

Clock tick: 0

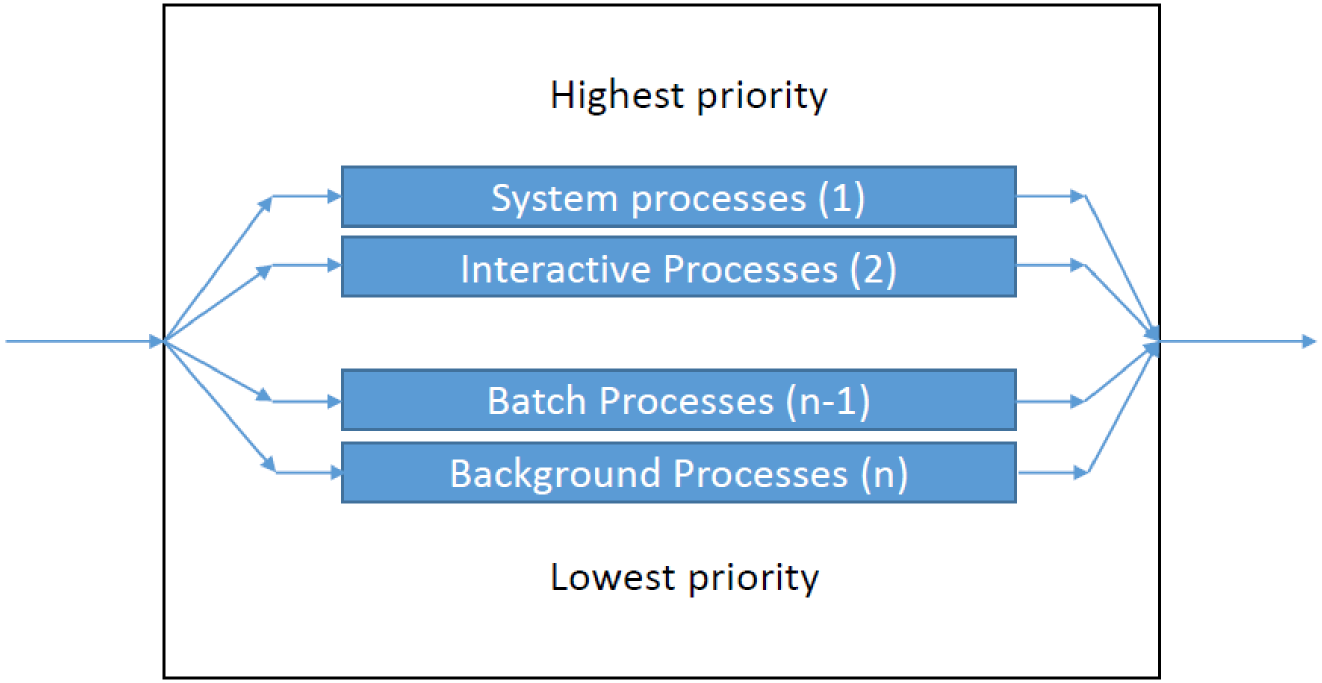
- "Circular" Queue
- Time Quantum .. Here let's choose 5ms
- Each process has a required burst time



Round Robin is simply trying to cut off "long bursts" while avoiding cutting off any "regular burst" processes.

MULTILEVEL QUEUE

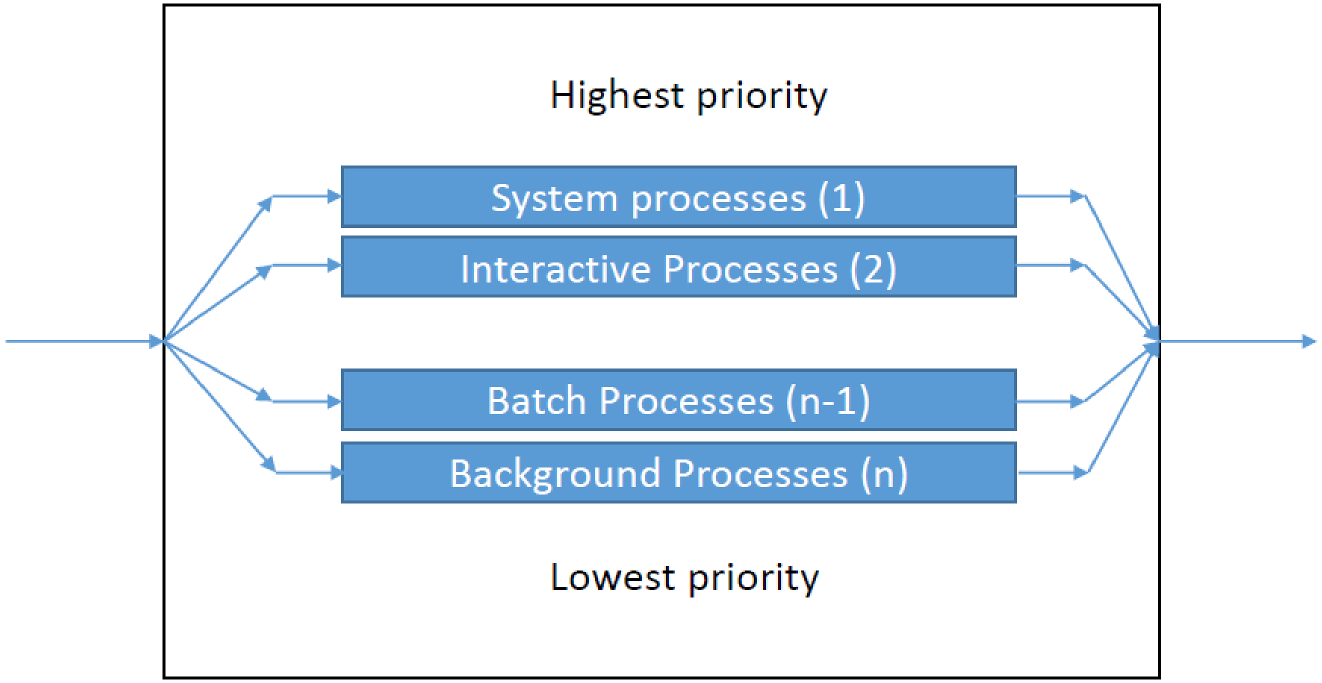
Multilevel Queue The ready queue is partitioned into several separate queues



MULTILEVEL QUEUE

Multilevel Queue The ready queue is partitioned into several separate queues

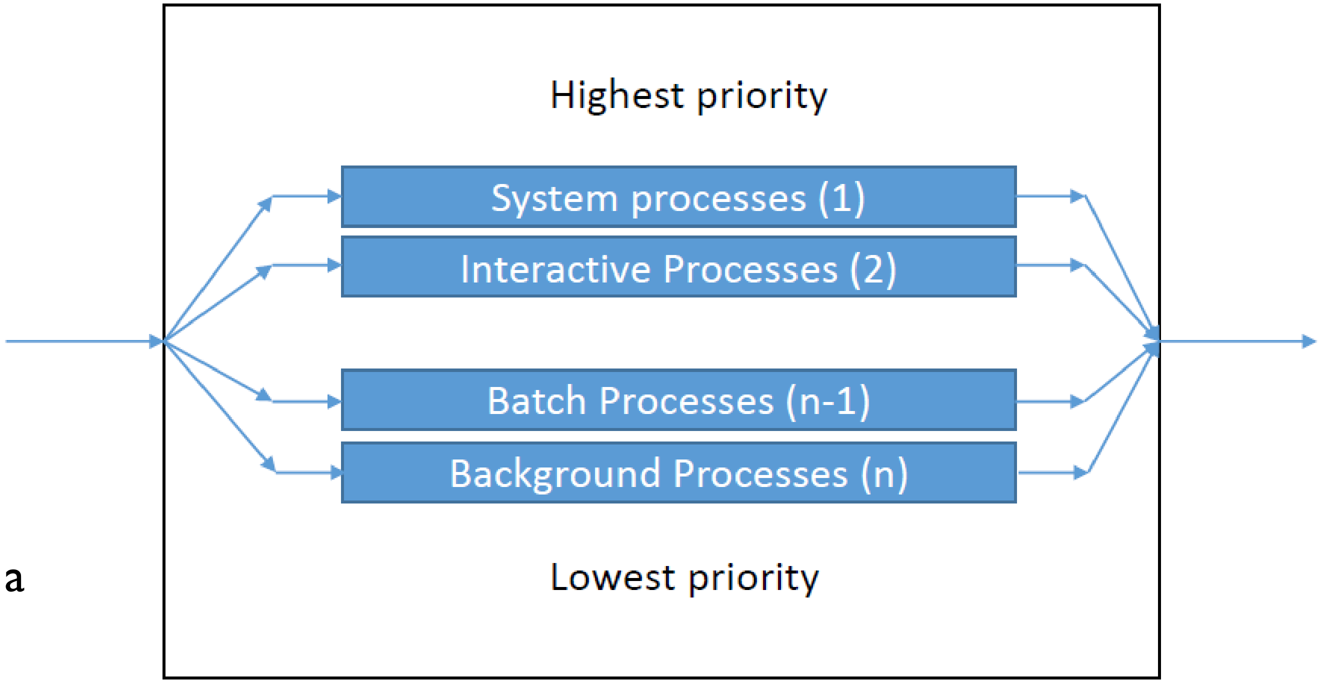
Creating different queues for different priorities.



MULTILEVEL QUEUE

Multilevel Queue The ready queue is partitioned into several separate queues

Creating different queues for different priorities.

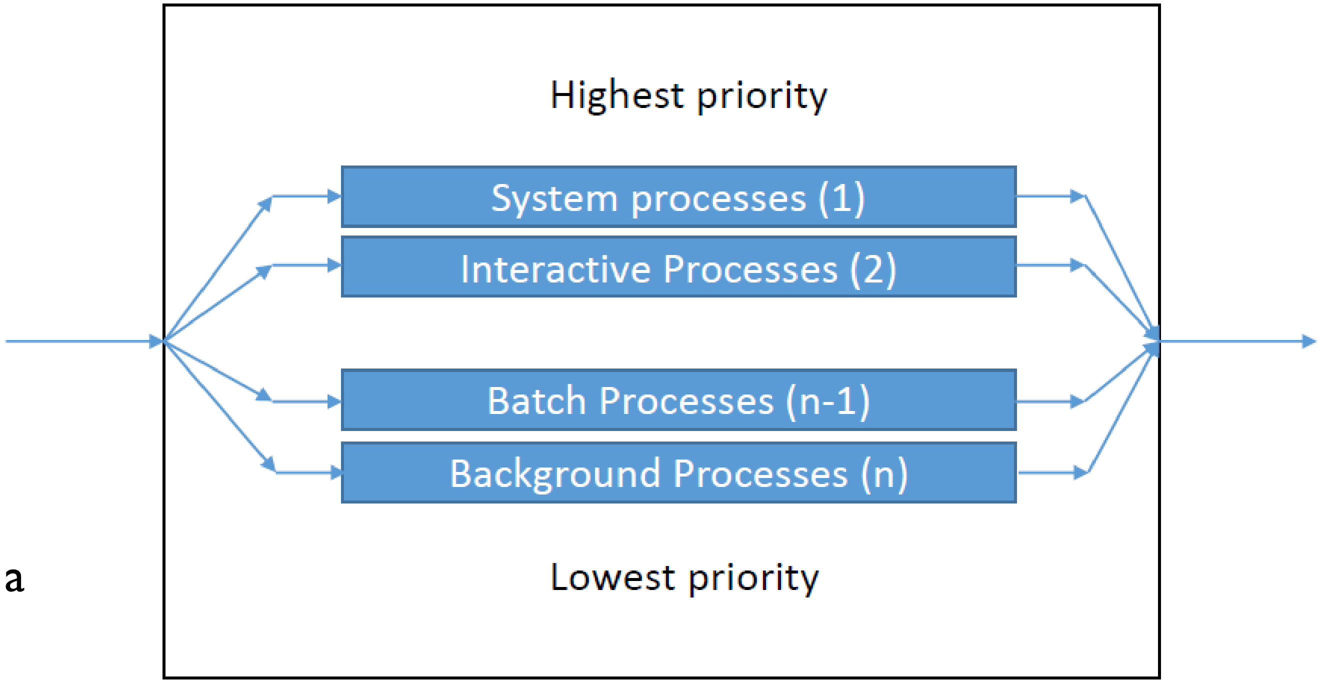


Q3:What is the advantage of this approach over having a single queue for all priorities?

MULTILEVEL QUEUE

Multilevel Queue The ready queue is partitioned into several separate queues

Creating different queues for different priorities.



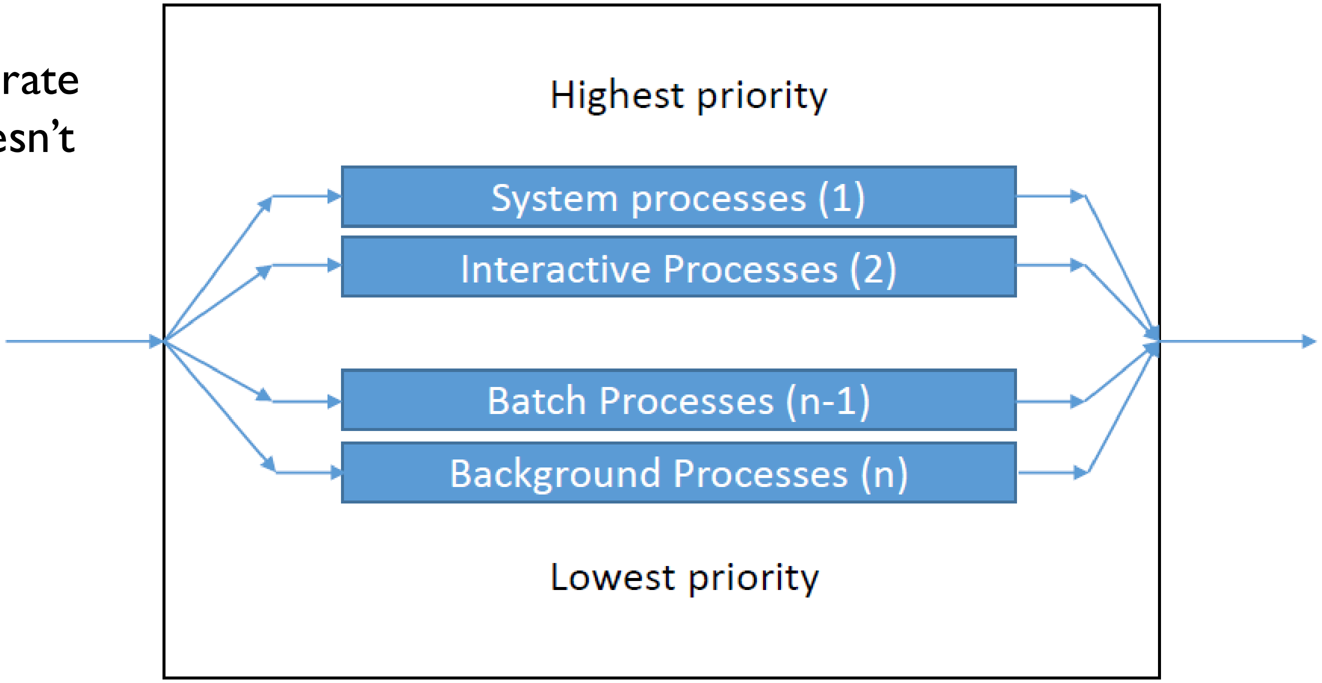
Q3:What is the advantage of this approach over having a single queue for all priorities?

No need for sorting!

MULTILEVEL QUEUE

Multilevel Queue The ready queue is partitioned into several separate queues

We can do more elaborate scheduling. Priority doesn't have to be absolute.

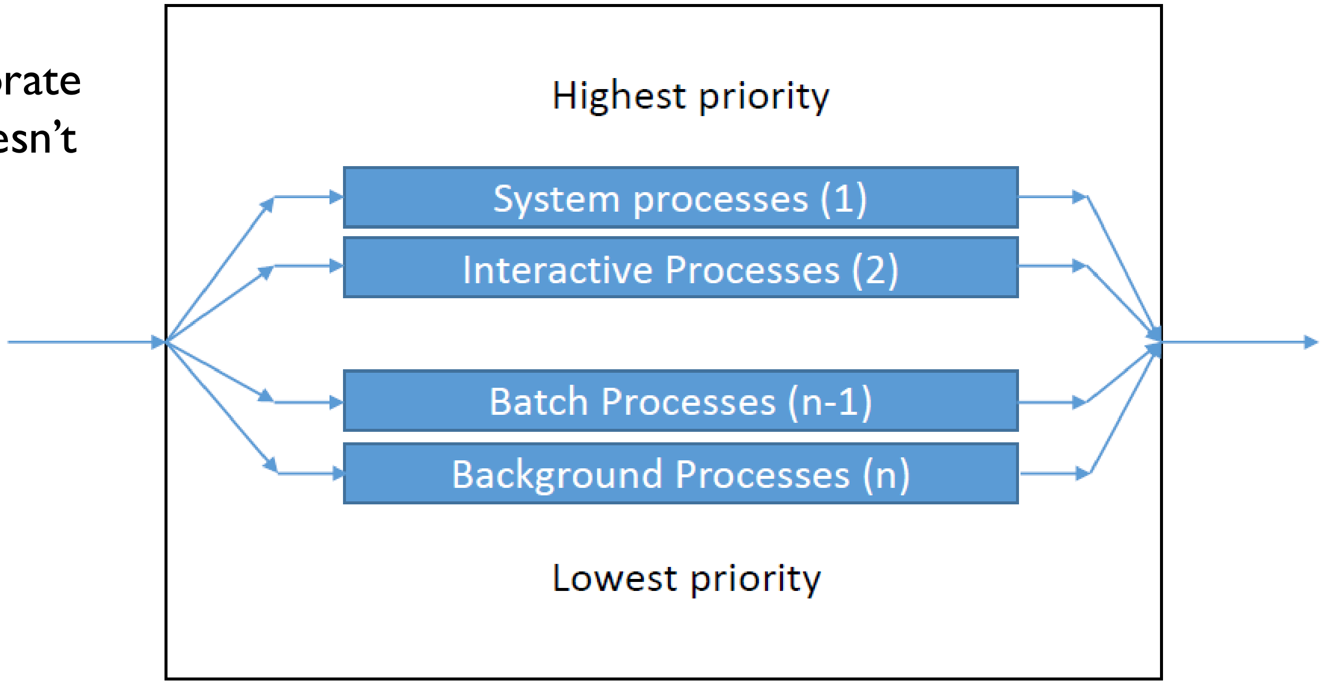


MULTILEVEL QUEUE

Multilevel Queue The ready queue is partitioned into several separate queues

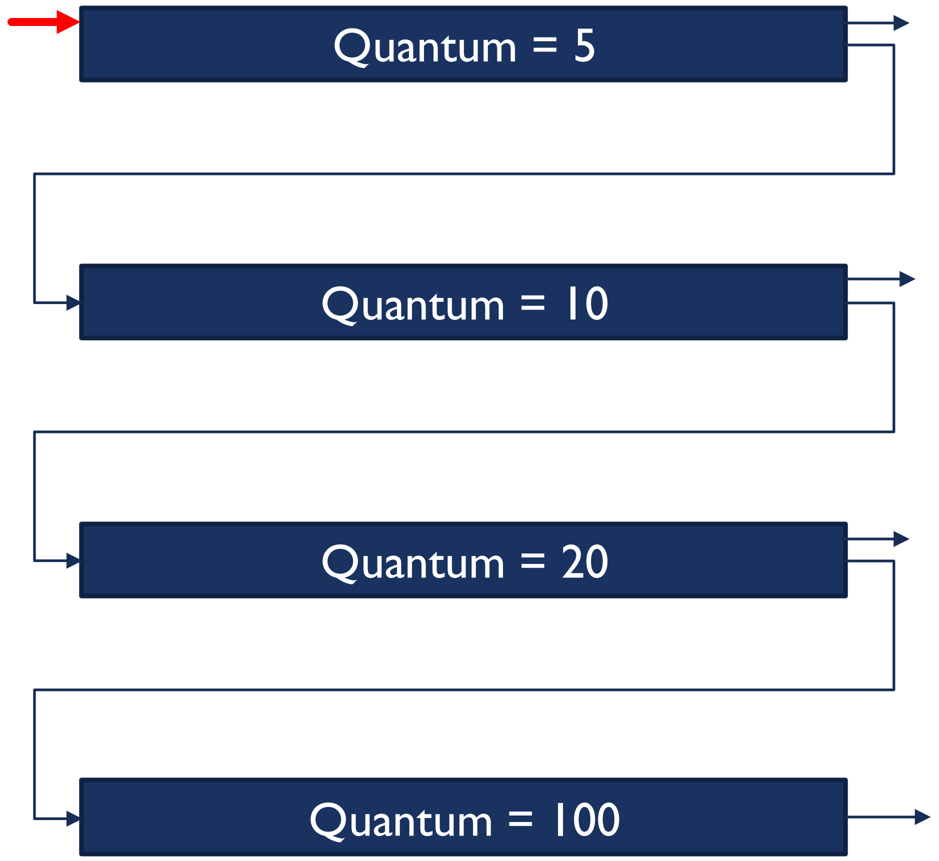
We can do more elaborate scheduling. Priority doesn't have to be absolute.

Highest priority queue might have 70% of CPU time, while the others 30%



MULTILEVEL QUEUE WITH FEEDBACK

Processes can migrate from one queue to another.

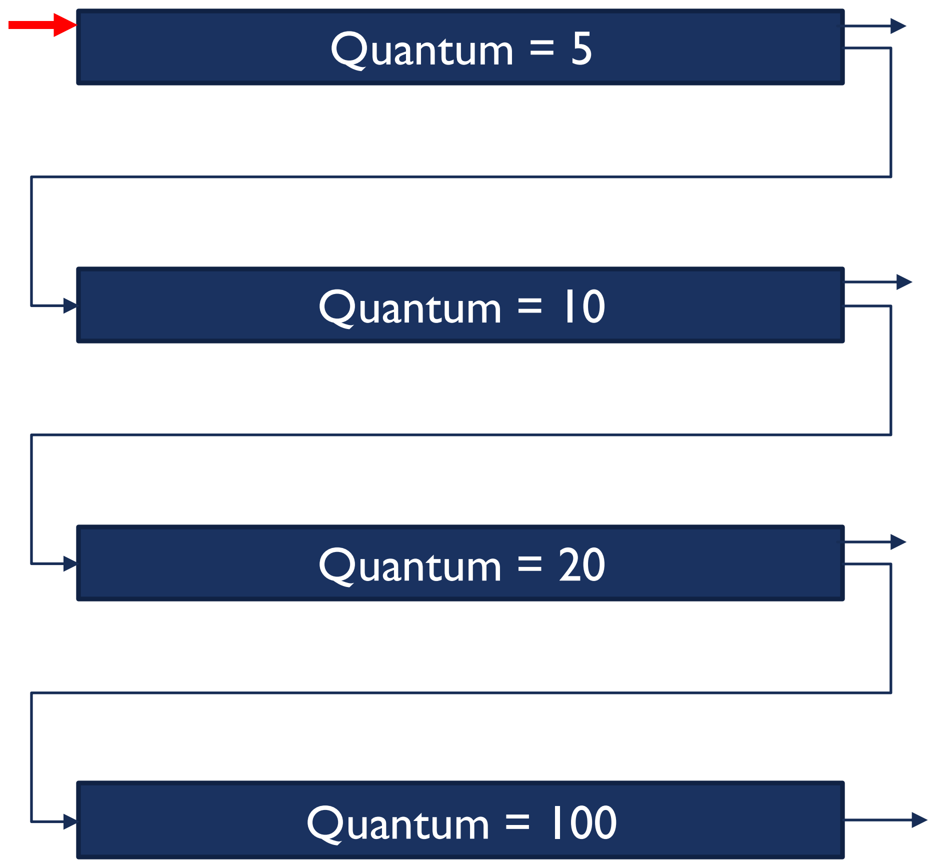


MULTILEVEL QUEUE WITH FEEDBACK

Multilevel Feedback Queue

Possible design

- If a dispatched process from the quantum=5 queue does not finish, a context switch occurs, and process is placed at tail end of quantum=10 queue.

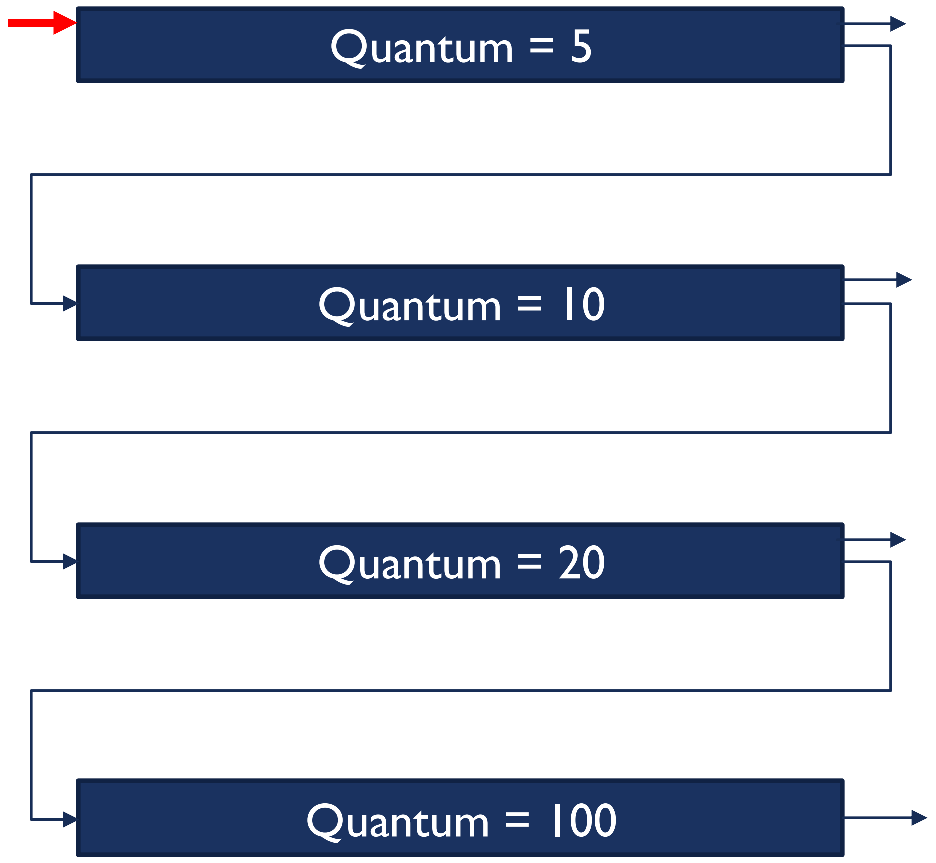


MULTILEVEL QUEUE WITH FEEDBACK

Multilevel Feedback Queue

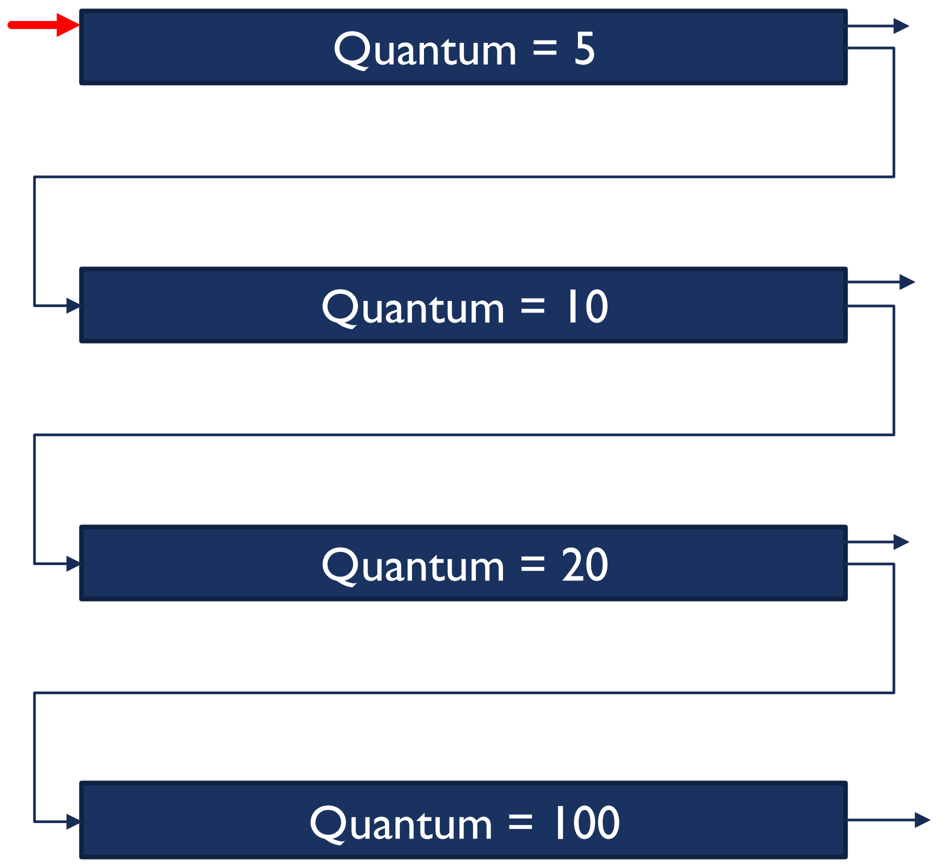
Possible design

- If a dispatched process from the quantum=5 queue does not finish, a context switch occurs, and process is placed at tail end of quantum=10 queue.
- If a process from the quantum=10 queue does not finish, a context switch occurs, and process is placed at the tail end of the quantum=20 queue, etc. etc.



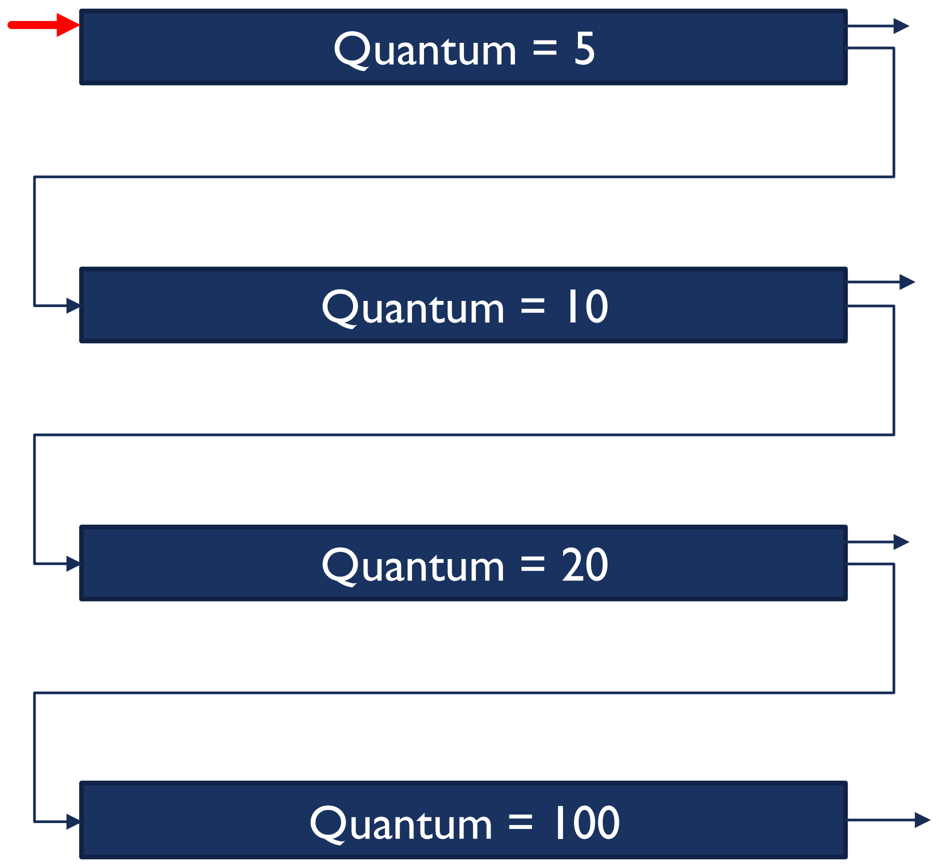
MULTILEVEL QUEUE WITH AGING

- Can a low priority process be starved?



MULTILEVEL QUEUE WITH AGING

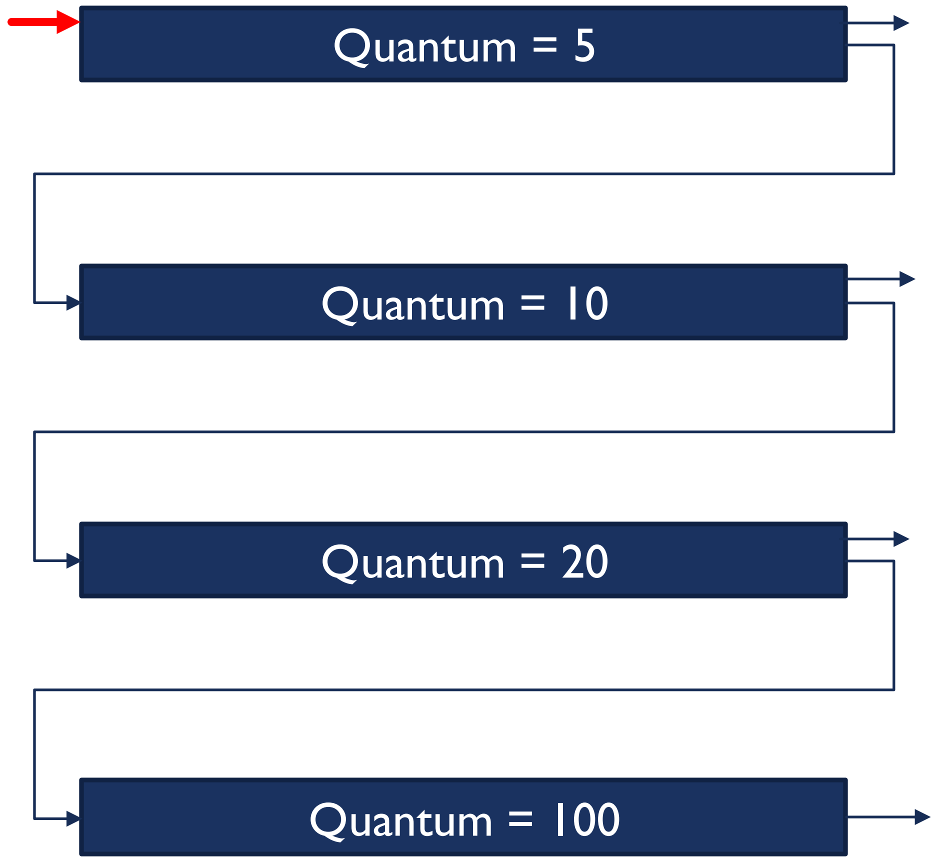
- Can a low priority process be starved?
- If the processor is always busy with high priority processes, low priority ones can become starved.



MULTILEVEL QUEUE WITH AGING

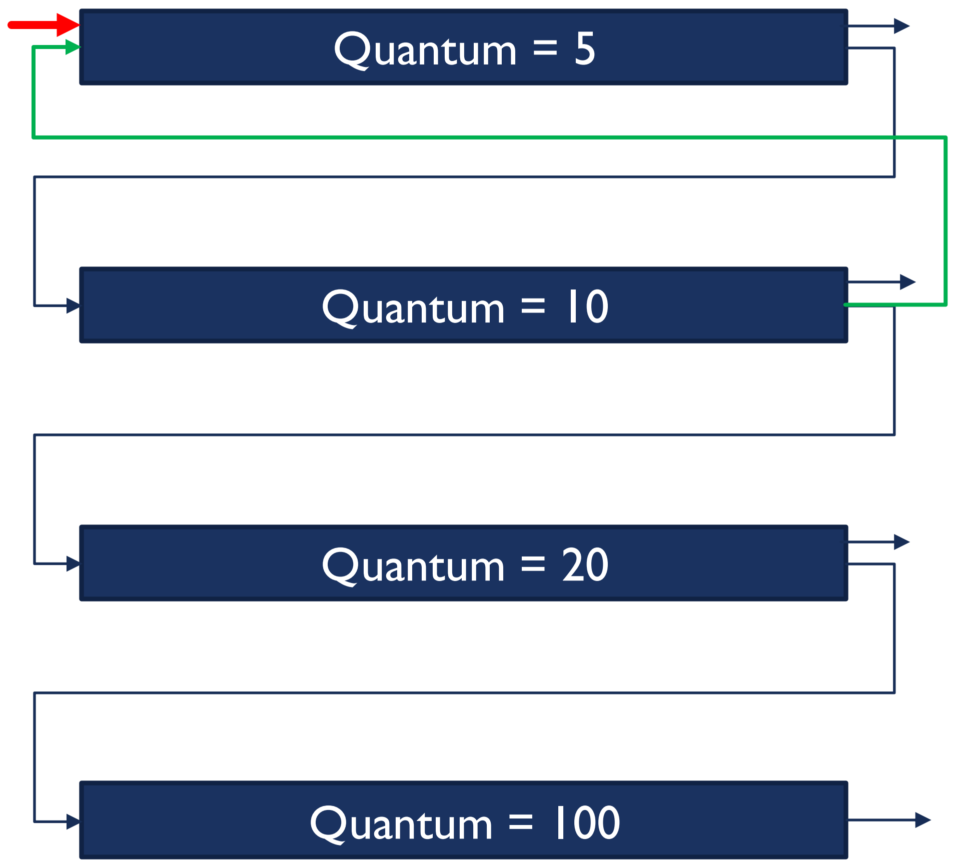
- Can a low priority process be starved?
- If the processor is always busy with high priority processes, low priority ones can become starved.

Q4: How would you fix the starvation problem?



MULTILEVEL QUEUE WITH AGING

- Can a low priority process be starved?
- If the processor is always busy with high priority processes, low priority ones can become starved.
- Aging can solve this.
- At predefined intervals, all processes in queue have their priority increased.
- After execution, a process priority goes back to its default value.



REAL-TIME SCHEDULING

- Time sensitive application:
  - Transportation: smart cars, jets, trains …
  - Manufacturing pipelines
  - Playing audio/video

REAL-TIME SCHEDULING

- Time sensitive application:
  - Transportation: smart cars, jets, trains …
  - Manufacturing pipelines
  - Playing audio/video
- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time

REAL-TIME SCHEDULING

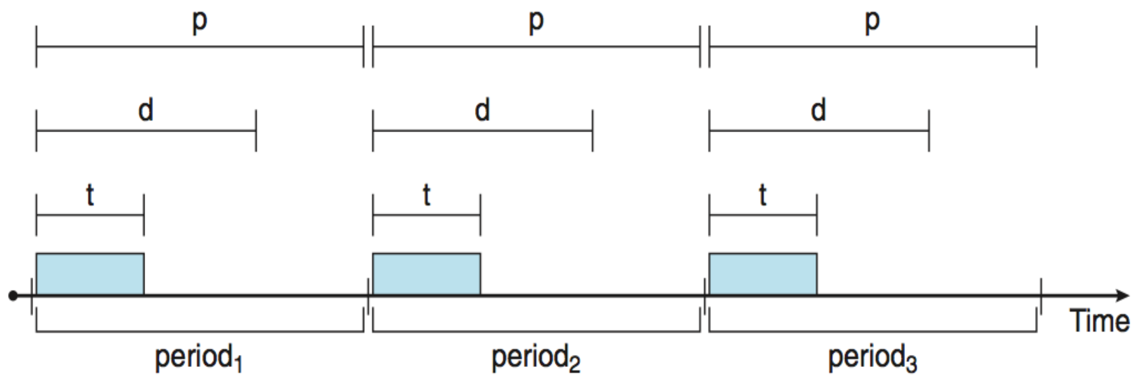
- Time sensitive application:
  - Transportation: smart cars, jets, trains …
  - Manufacturing pipelines
  - Playing audio/video
- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For **hard** real-time must also provide ability to meet deadlines. If it can't, the OS would refuse to run the program.

REAL-TIME SCHEDULING

- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - processing time t
  - deadline d
  - period p
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is $1/p$

REAL-TIME SCHEDULING

- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - processing time t
  - deadline d
  - period p
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is $1/p$

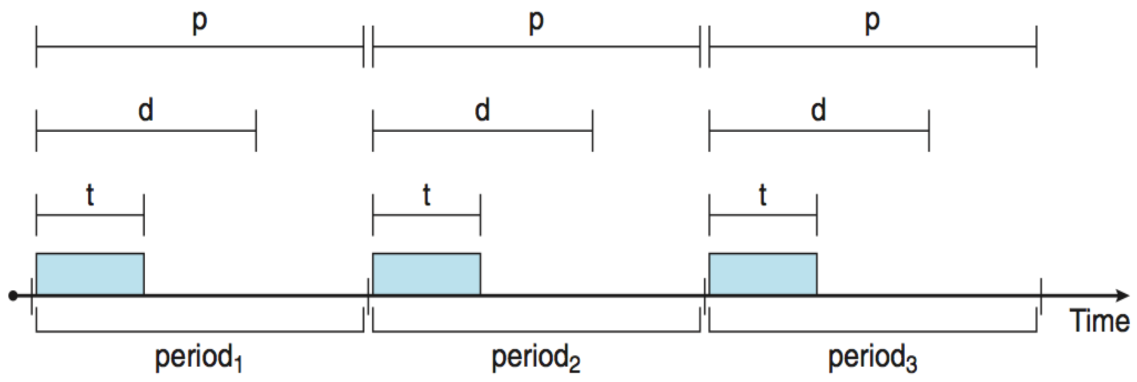


REAL-TIME SCHEDULING

- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - processing time t
  - deadline d
  - period p
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is $1/p$

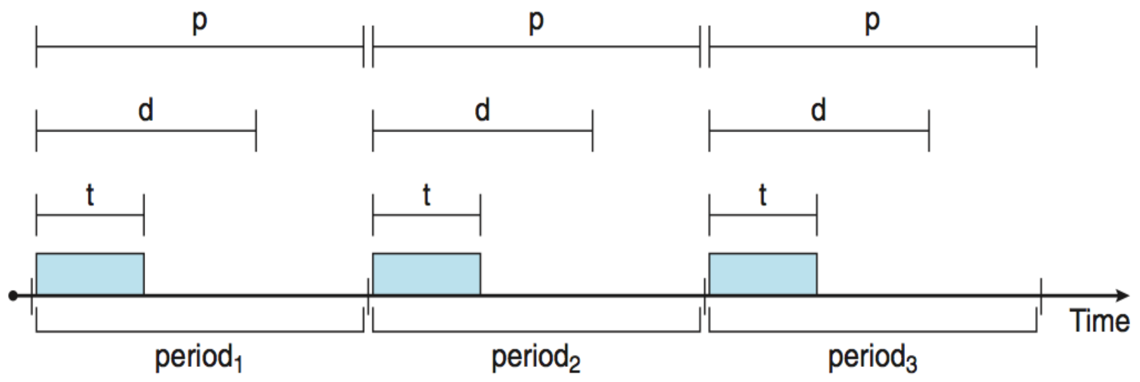
$\longrightarrow$ If p < t … what does that mean?

Processor is not fast enough!



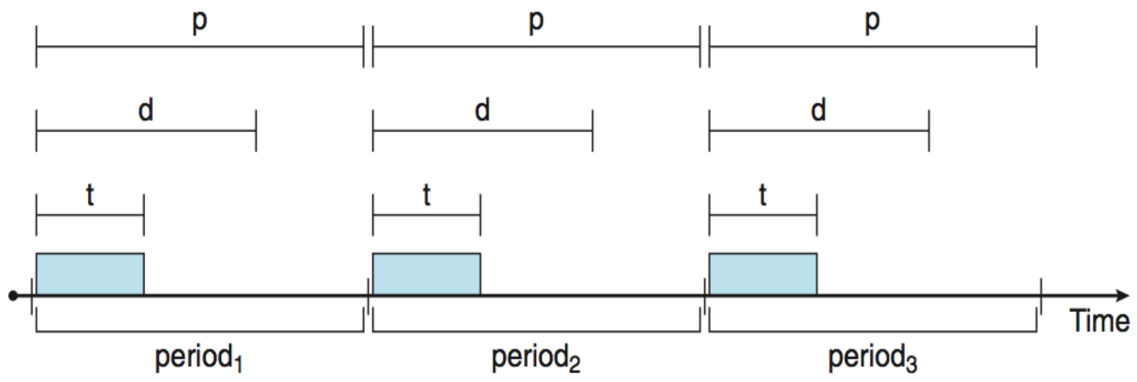
REAL-TIME SCHEDULING

- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d*, period *p*
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is $1/p$



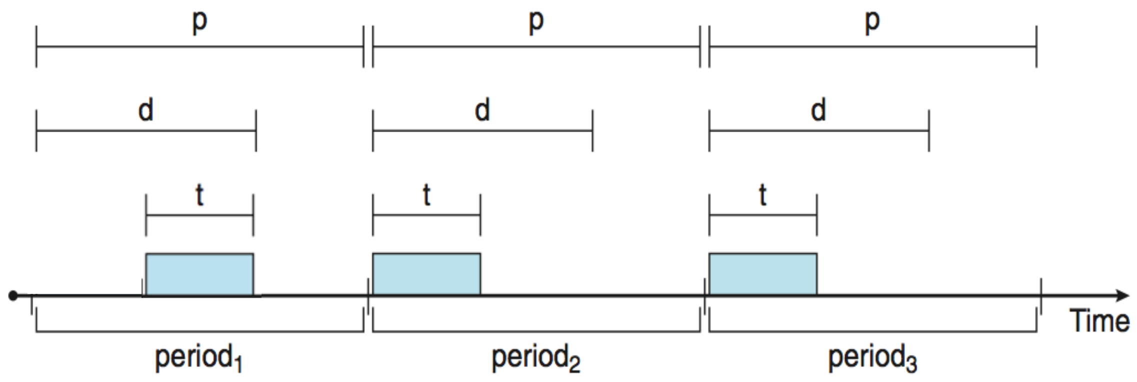
REAL-TIME SCHEDULING

- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d*, period *p*
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is 1/*p*



REAL-TIME SCHEDULING

- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d*, period *p*
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is 1/*p*



REAL-TIME SCHEDULING

Periodic : occurring at a constant interval, or period (p)
Processing time : time needed to execute (burst time, t)
Deadline : time by which a process must finish

All units are ms

Q: if a requesting process does not "satisfy" the $0 \le t \le d \le p$ **requirement, the CPU scheduler {might/should/should not} reject the process**

🟥 A 🟩 B
🟨 C

A : might
B : should
C : should not

REAL-TIME SCHEDULING

Periodic : occurring at a constant interval, or period (p)
Processing time : time needed to execute (burst time, t)
Deadline : time by which a process must finish

All units are ms

Q: if a requesting process does not "satisfy" the $0 \le t \le d \le p$ **requirement, the CPU scheduler {might/should/should not} reject the process**

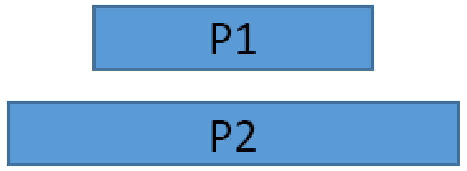
🟥 A	🟩 B
🟨 C	

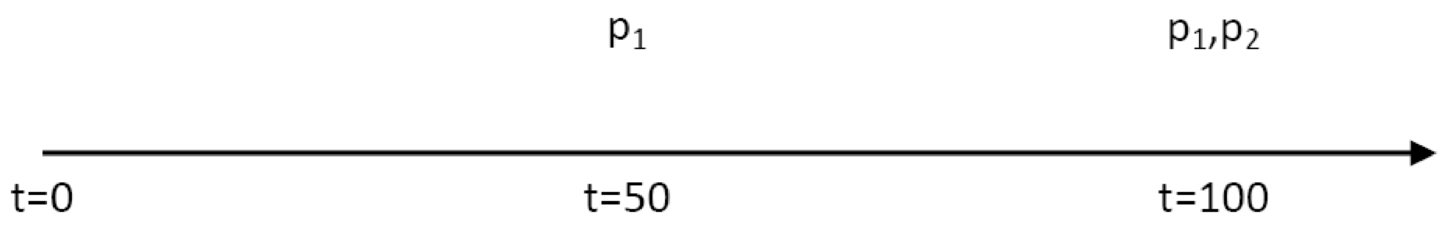
A : might
B : should
C : should not

The processor should **reject** the process as it can never satisfy its runtime requirement.

REAL-TIME SCHEDULING

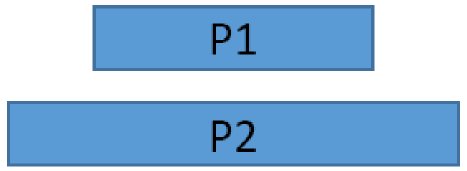
Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

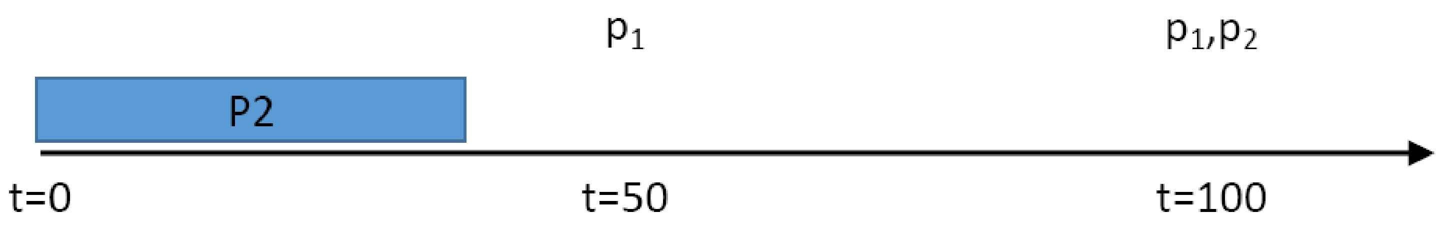




REAL-TIME SCHEDULING

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

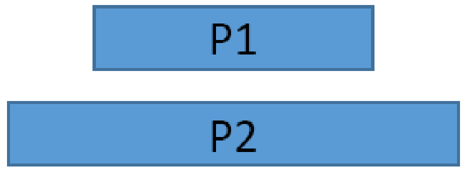


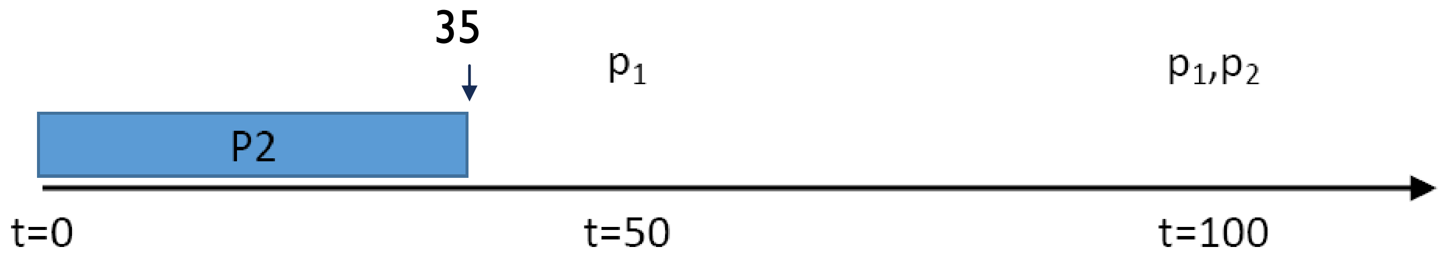


If P2 is scheduled before P1, what is the timeline?

REAL-TIME SCHEDULING

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

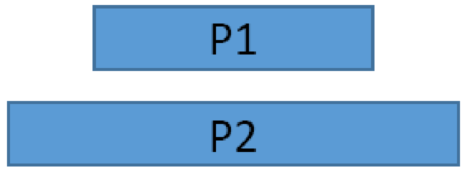


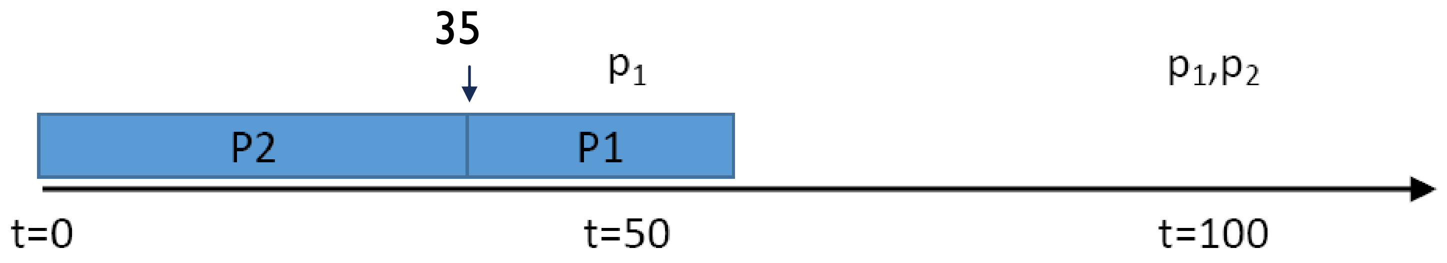


If P2 is scheduled before P1, what is the timeline?

REAL-TIME SCHEDULING

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

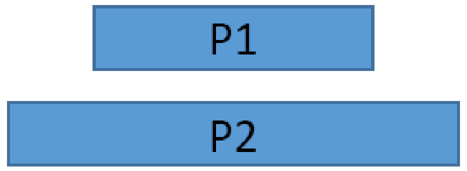


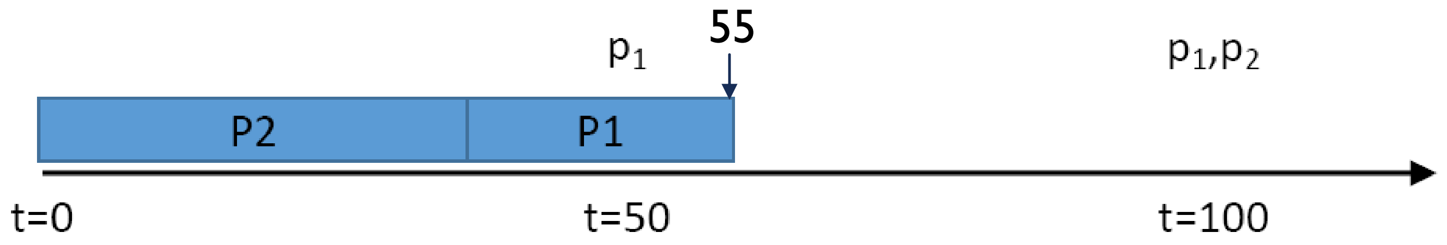


If P2 is scheduled before P1, what is the timeline?

REAL-TIME SCHEDULING

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period



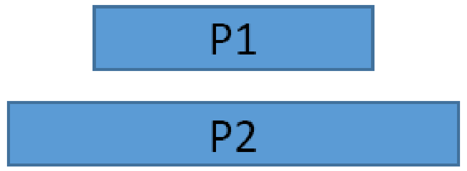


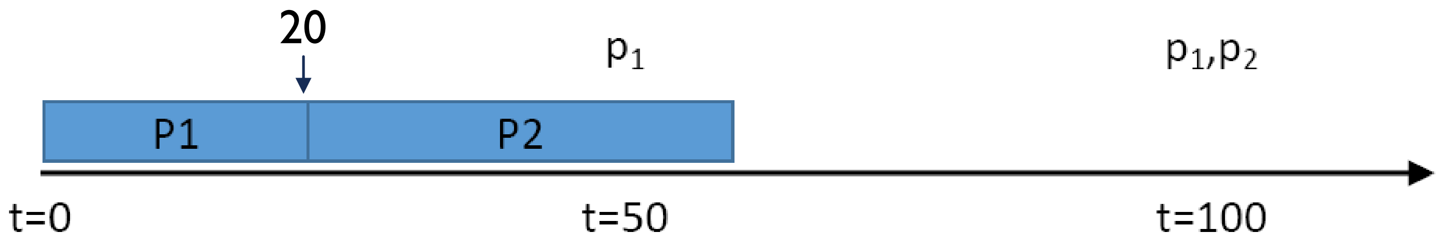
If P2 is scheduled before P1, what is the timeline?

P1 misses its deadline!

REAL-TIME SCHEDULING

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

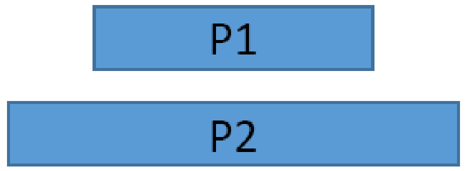


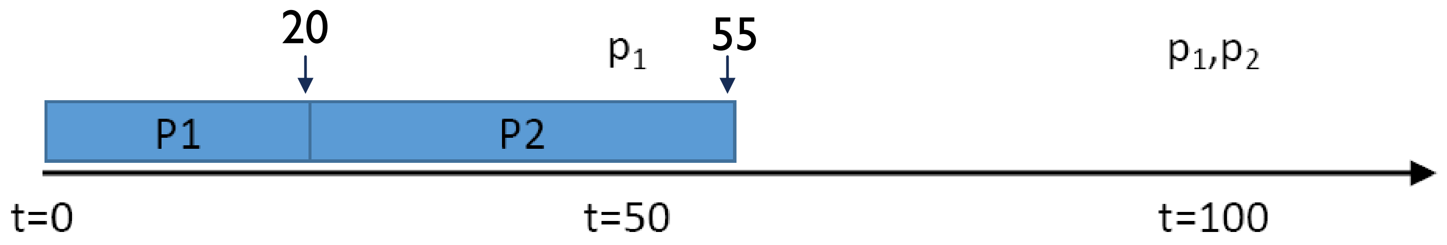


What if P1 goes first?

REAL-TIME SCHEDULING

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

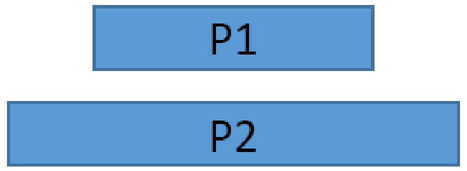


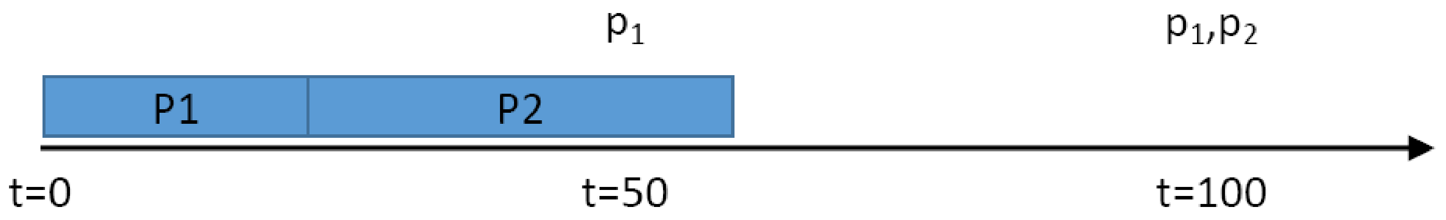


What if P1 goes first?

REAL-TIME SCHEDULING

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

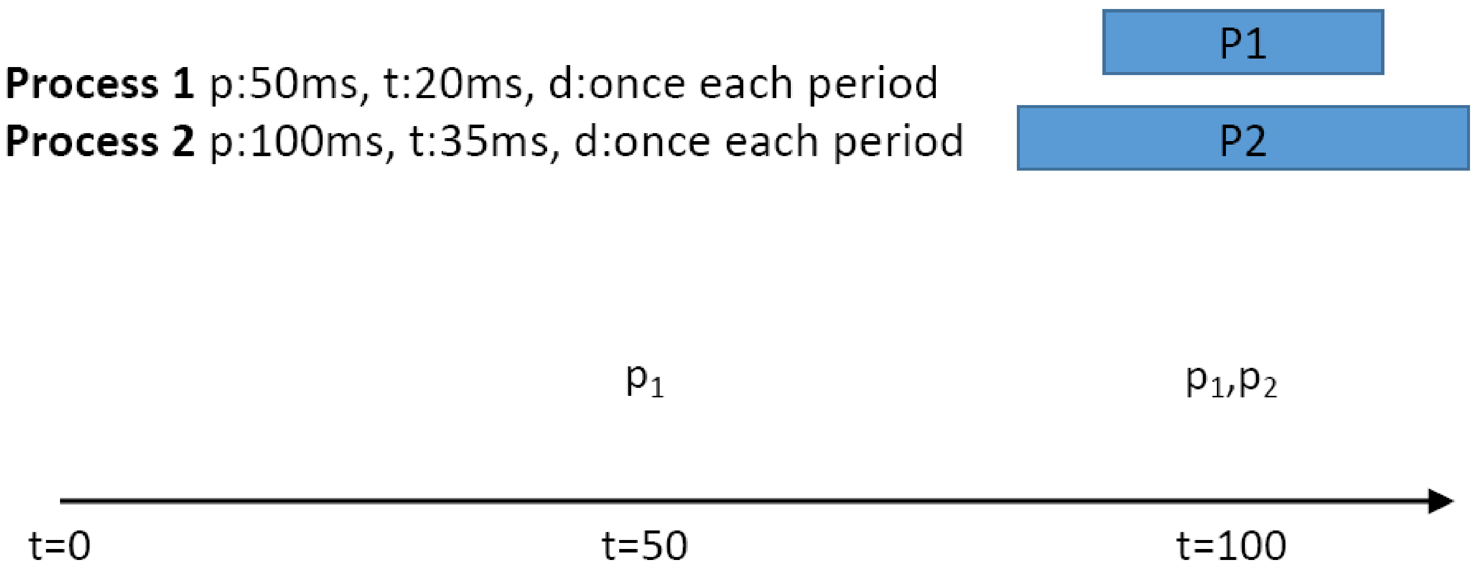




What if P1 goes first?
Both processes can meet their deadline.

REAL-TIME SCHEDULING: RATE MONOTONIC SCHEDULING (RMS)

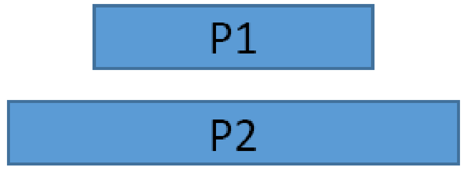
Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

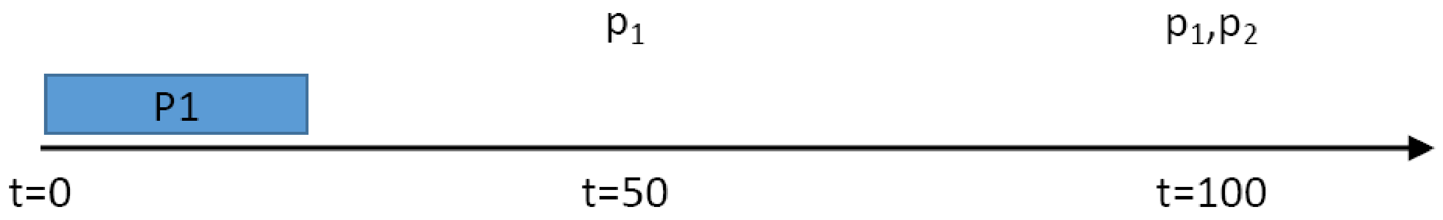


RMS: Assigns higher priority to shorter period with preemptive scheduling.

REAL-TIME SCHEDULING: RATE MONOTONIC SCHEDULING (RMS)

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

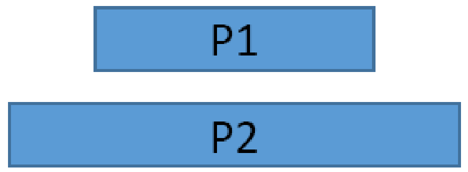


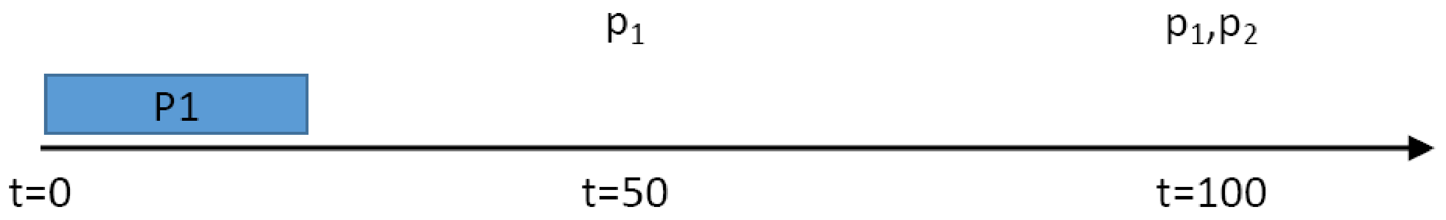


Priority of P1 : 1/50
Priority of P2 : 1/100

REAL-TIME SCHEDULING: RATE MONOTONIC SCHEDULING (RMS)

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period



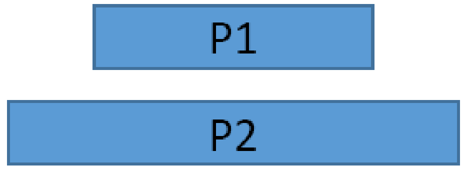


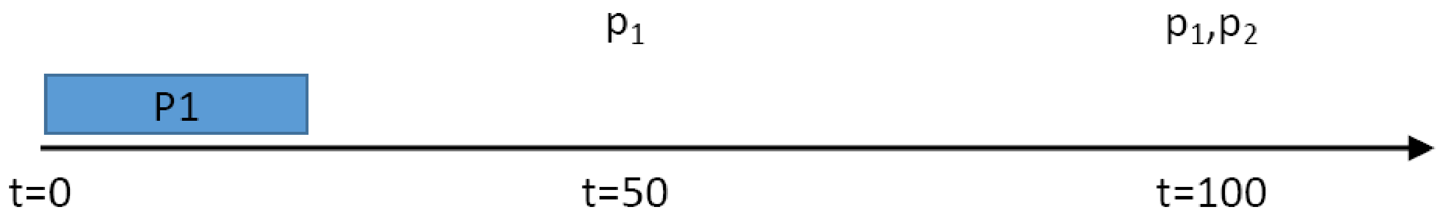
Priority of P1 : 1/50
Priority of P2 : 1/100

- **P1 will have higher priority**
- **Priority is static**

REAL-TIME SCHEDULING: RATE MONOTONIC SCHEDULING (RMS)

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period





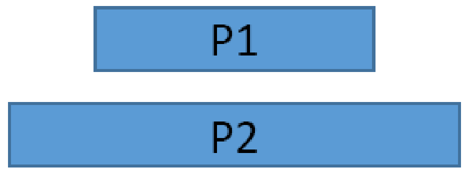
Priority of P1 : 1/50
Priority of P2 : 1/100

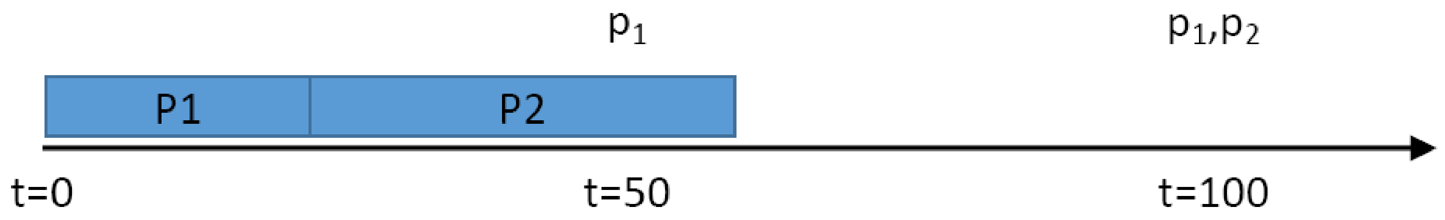
- **P1 will have higher priority**
- **Priority is static**

- **Scheduling is preemptive: lower priority process will be removed as soon as higher priority ones are available**

REAL-TIME SCHEDULING: RATE MONOTONIC SCHEDULING (RMS)

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period



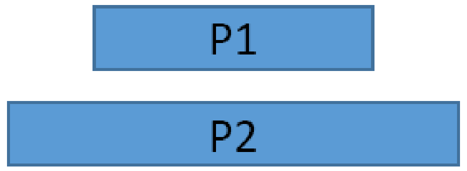


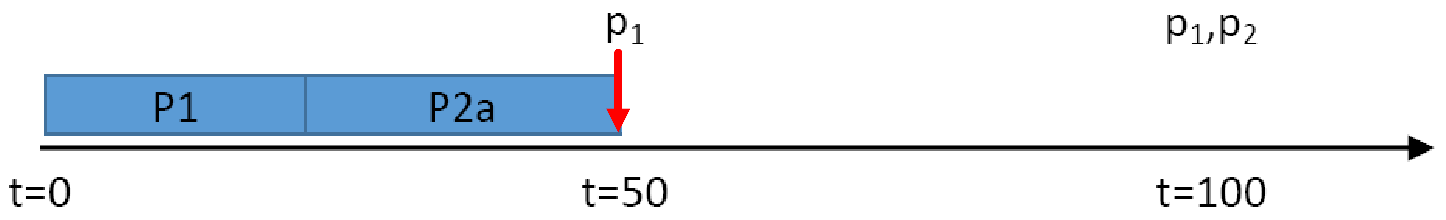
Priority of P1 : 1/50
Priority of P2 : 1/100

Scheduling is preemptive: lower priority process will be removed as soon as higher priority ones are available

REAL-TIME SCHEDULING: RATE MONOTONIC SCHEDULING (RMS)

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period



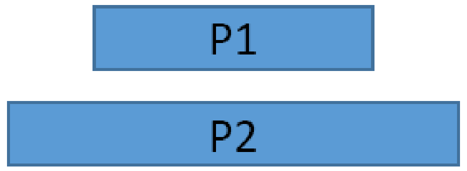


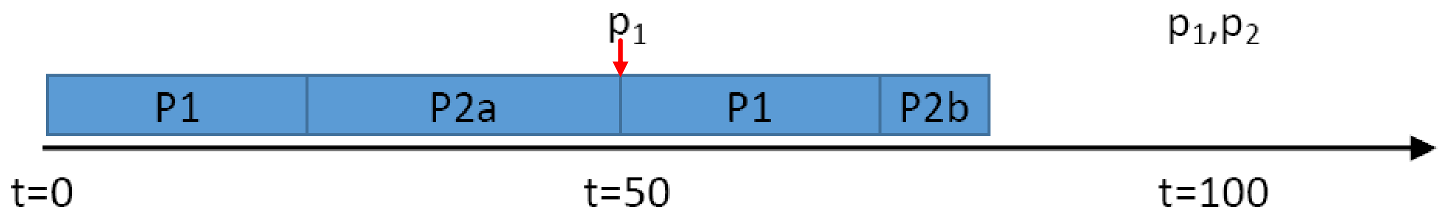
Priority of P1 : 1/50
Priority of P2 : 1/100

Scheduling is preemptive: lower priority process will be removed as soon as higher priority ones are available

REAL-TIME SCHEDULING: RATE MONOTONIC SCHEDULING (RMS)

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period



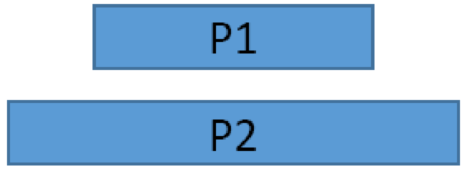


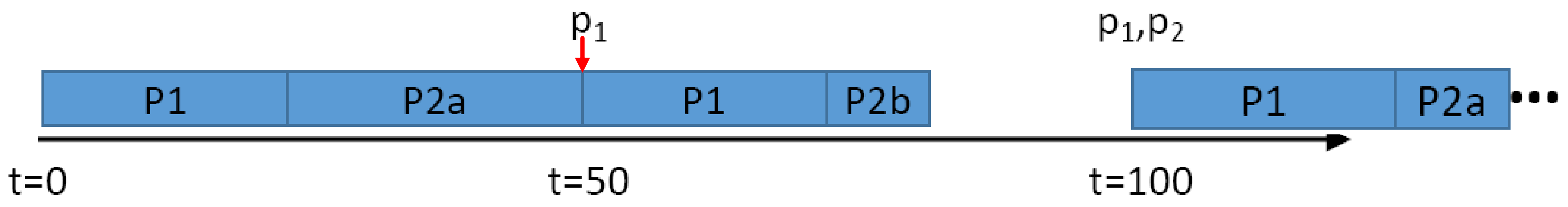
Priority of P1 : 1/50
Priority of P2 : 1/100

Scheduling is preemptive: lower priority process will be removed as soon as higher priority ones are available

REAL-TIME SCHEDULING

Process 1 p:50ms, t:20ms, d:once each period
Process 2 p:100ms, t:35ms, d:once each period

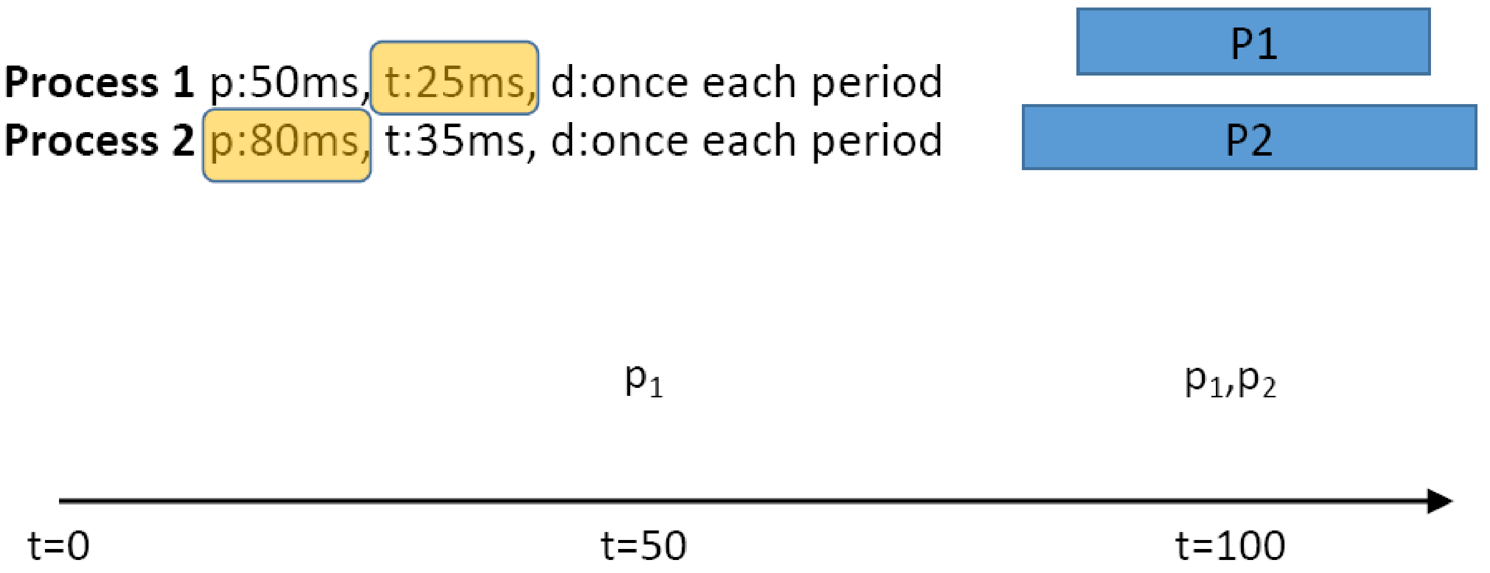




Cycle will repeat

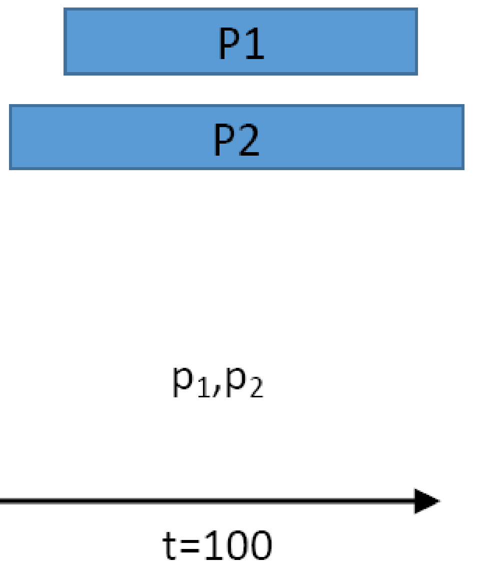
REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period



REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

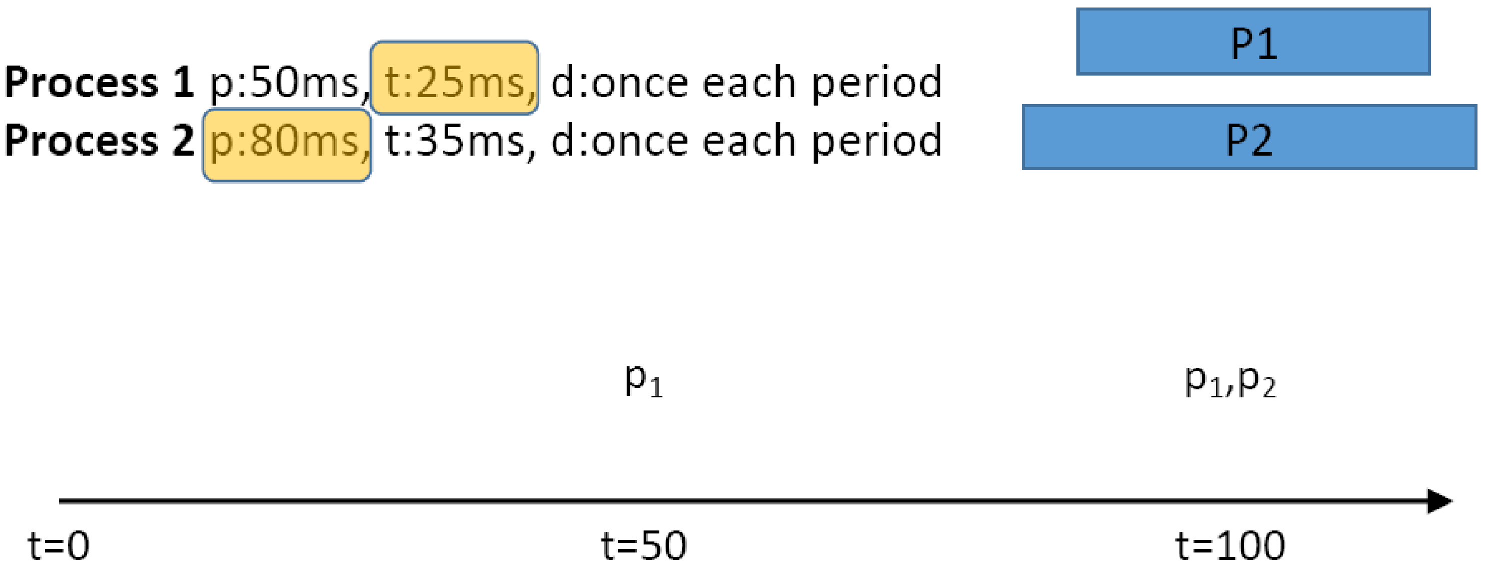


Q: Do we have enough CPU time for both?

P1: 25/50 = 0.5
P2: 35/80 = 0.4375
Total = 0.94375 < 1

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

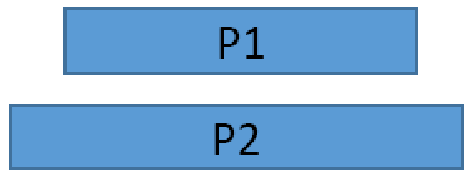


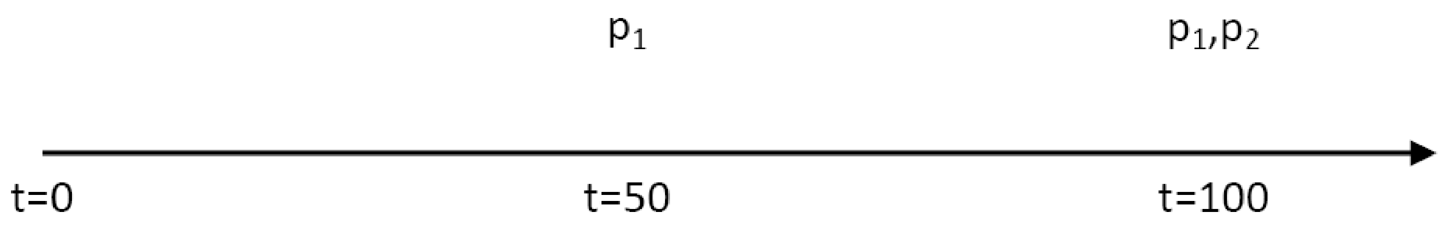
Q: Do we have enough CPU time for both?
Yes, we should have.

P1: 25/50 = 0.5
P2: 35/80 = 0.4375
Total = 0.94375 < 1

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

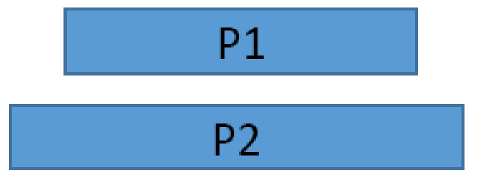


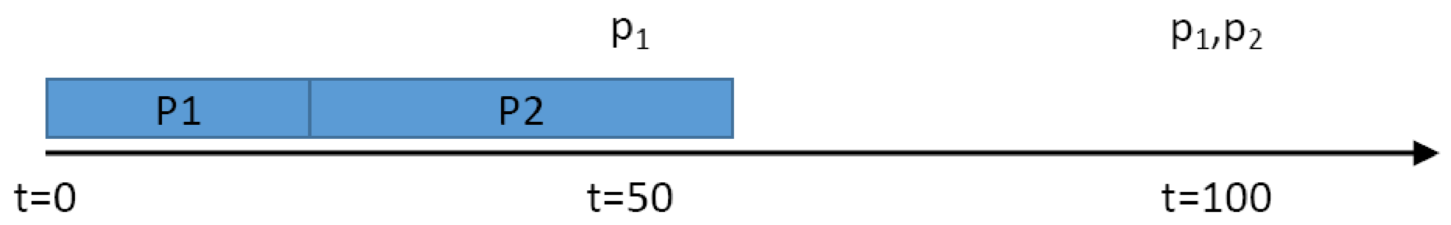


P1 Priority = 1/50
P2 Priority = 1/80

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

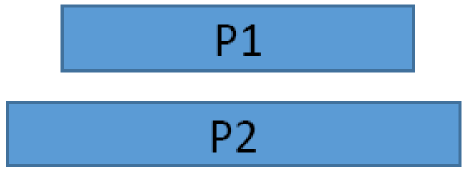


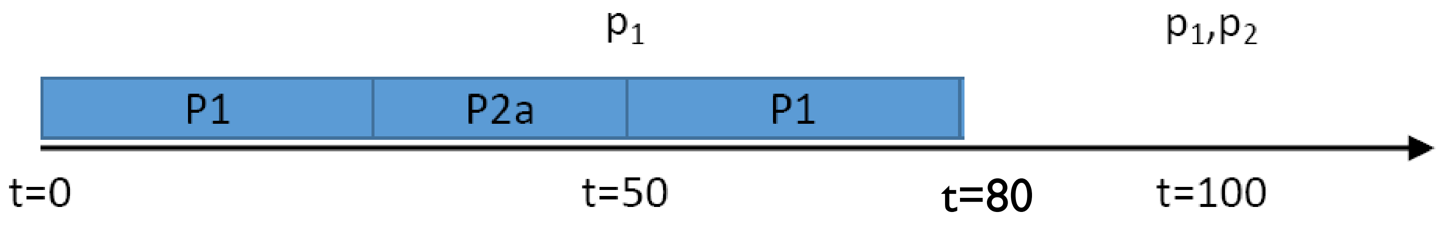


P1 Priority = 1/50
P2 Priority = 1/80

REAL-TIME SCHEDULING

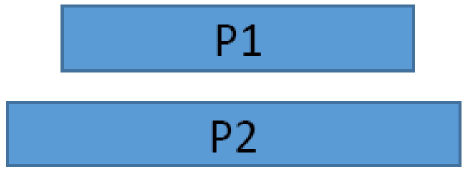
Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

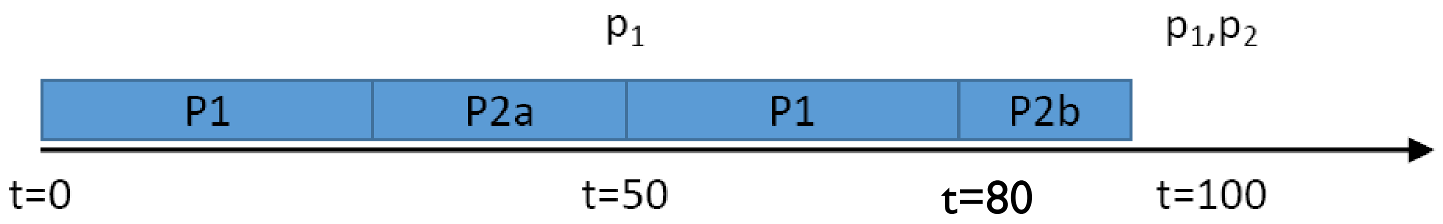




REAL-TIME SCHEDULING

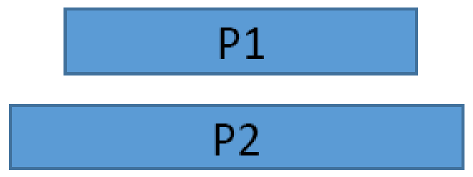
Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

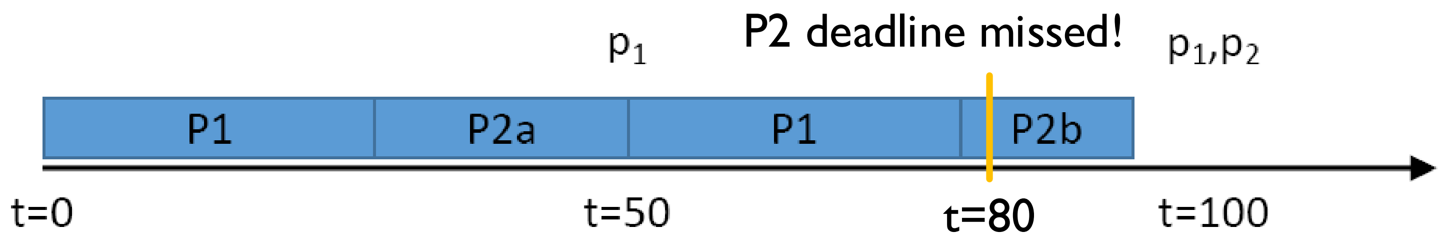




REAL-TIME SCHEDULING

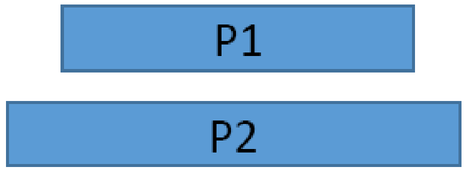
Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

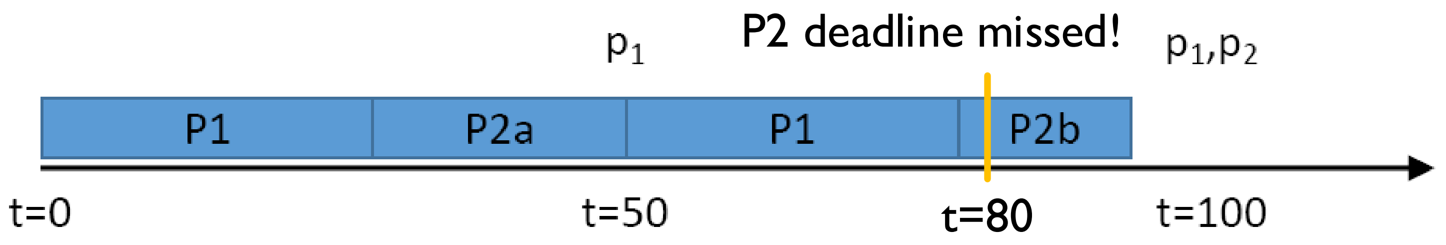




REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

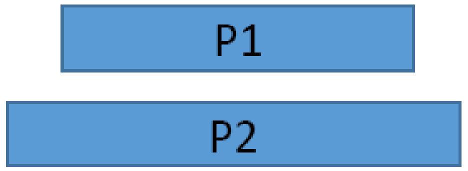


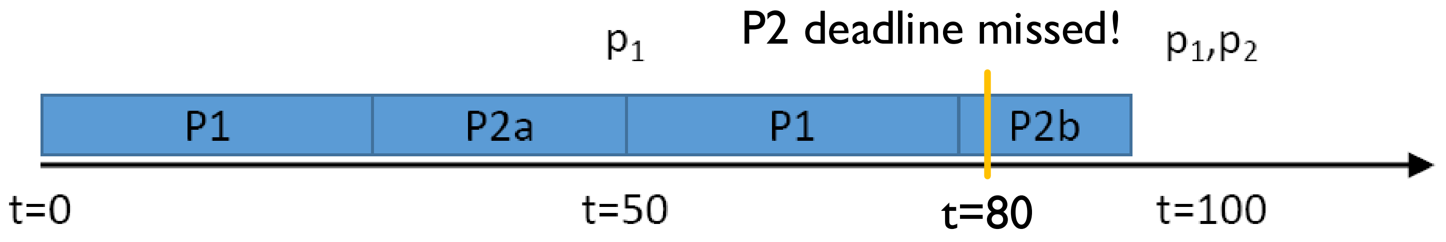


For RMS CPU utilization is actually capped at $\boldsymbol{N(2^{1/N} - 1)}$
where N is the number of processes.

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period



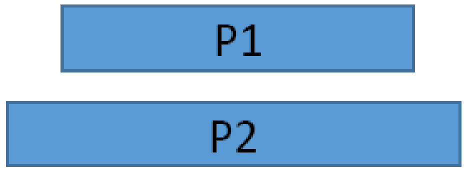


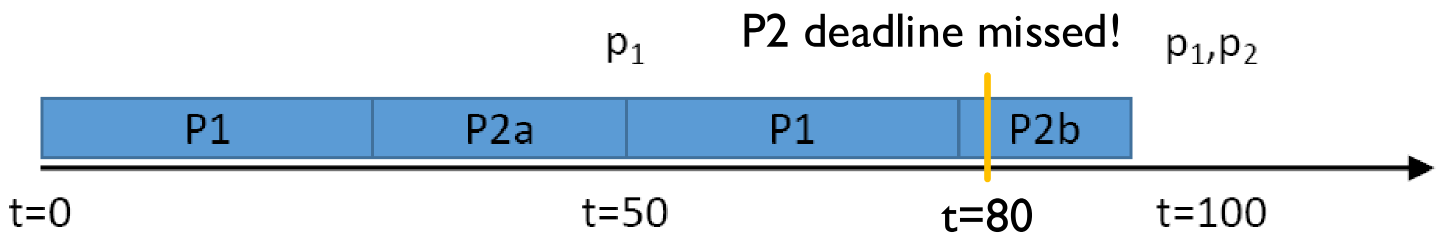
For RMS CPU utilization is actually capped at $\boldsymbol{N(2^{1/N} - 1)}$
where N is the number of processes.

CPU Cap: $N(2^{1/N} - 1) = 2(2^{1/2} - 1) = 0.83$

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period



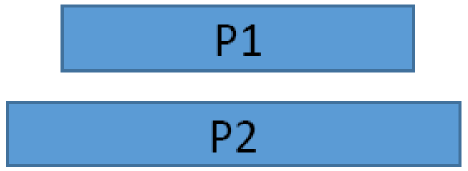


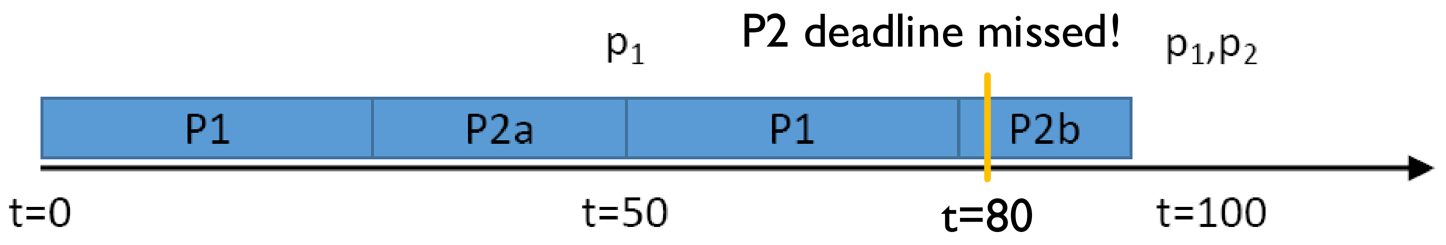
For RMS CPU utilization is actually capped at $\boldsymbol{N(2^{1/N} - 1)}$
where N is the number of processes.

CPU Cap: $N(2^{1/N} - 1) = 2(2^{1/2} - 1) = 0.83$
CPU Utilization required: 0.94

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period



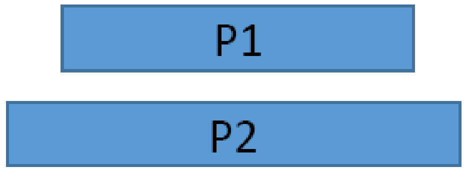


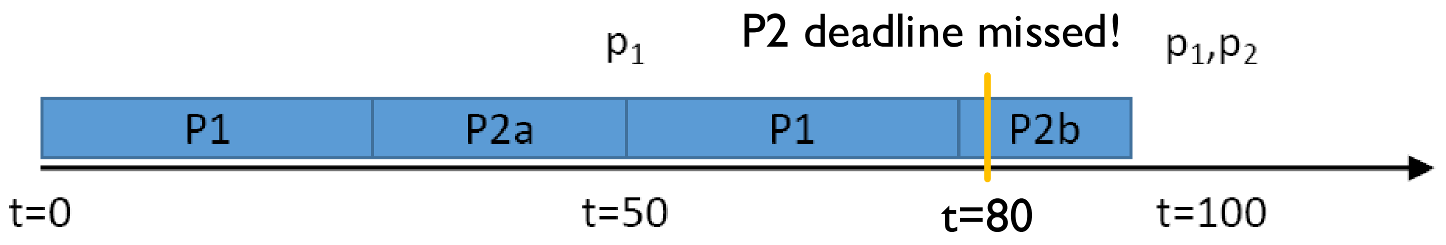
For RMS CPU utilization is actually capped at $\boldsymbol{N(2^{1/N} - 1)}$
where N is the number of processes.

CPU Cap: $N(2^{1/N} - 1) = 2(2^{1/2} - 1) = 0.83$
CPU Utilization required: 0.94 = 35/80+25/50

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

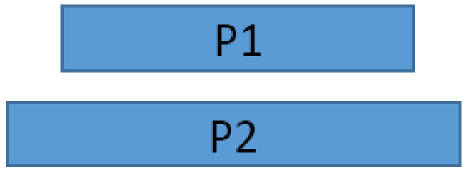




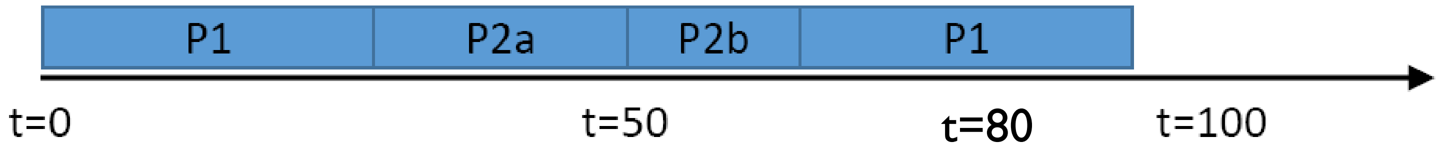
Q: How to fix this?

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

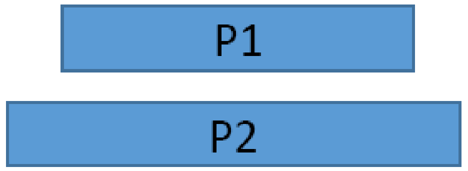


P1 lets P2 complete

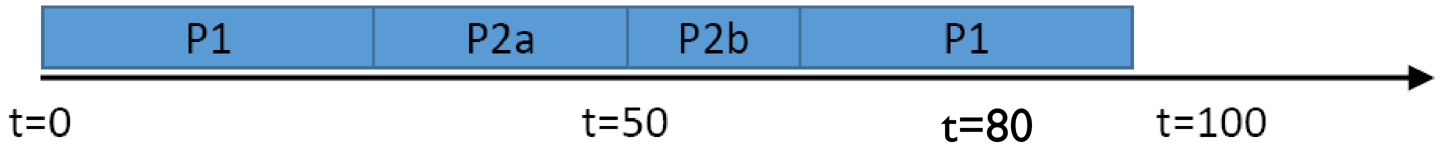


REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period



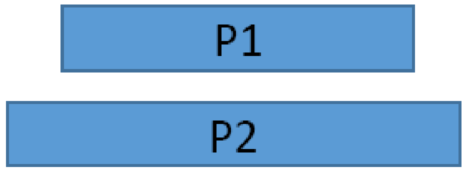
P1 lets P2 complete



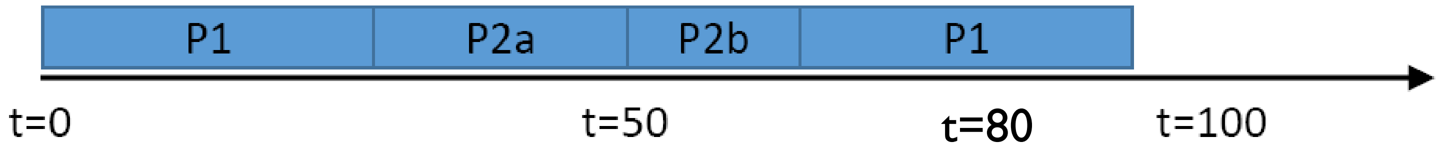
- This works but this requires dynamic priority.
- First period: priority was to P1, second period it was to P2.

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period



P1 lets P2 complete

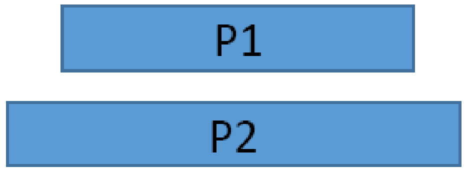


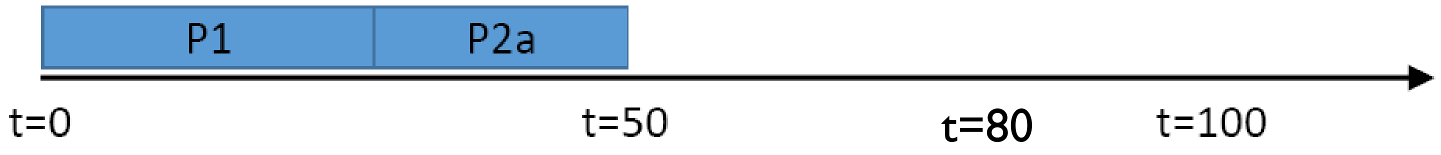
- This works but this requires dynamic priority.
- First period: priority was to P1, second period it was to P2.

Need a new algorithm with dynamic priority.

REAL-TIME SCHEDULING: EARLIEST DEADLINE FIRST

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period

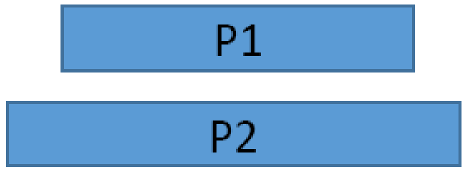


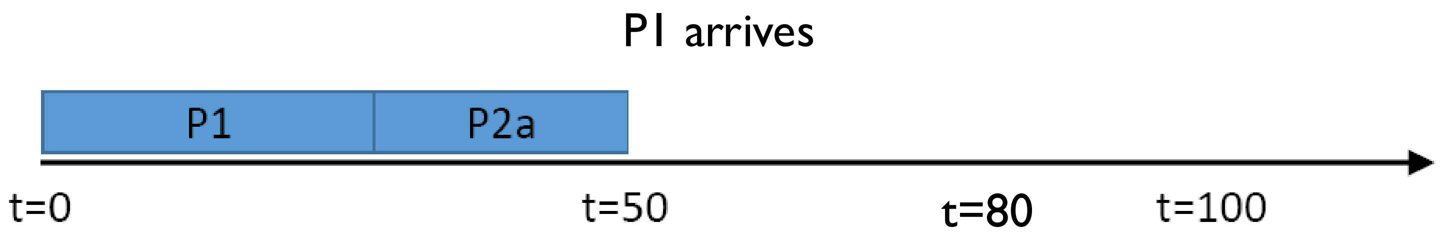


Earliest Deadline First Algorithm:
At t=0, earliest deadline is P1's so priority goes to P1

REAL-TIME SCHEDULING: EARLIEST DEADLINE FIRST

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period



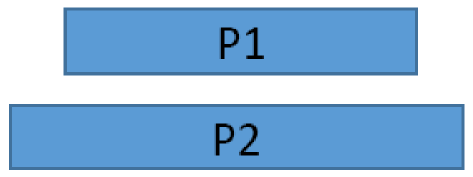


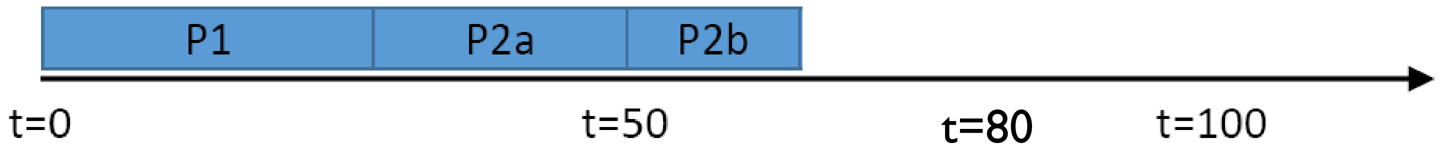
At t=50:

- Deadline For P1: 100ms
- Deadline for P2: 80ms
- Earliest Deadline First: P2 keeps executing

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period



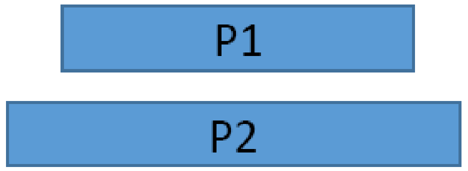


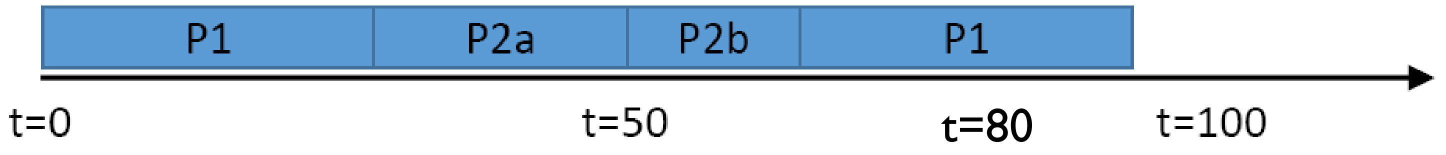
At t=50:

- Deadline For P1: 100ms
- Deadline for P2: 80ms
- Earliest Deadline First: P2 keeps executing

REAL-TIME SCHEDULING

Process 1 p:50ms, t:25ms, d:once each period
Process 2 p:80ms, t:35ms, d:once each period



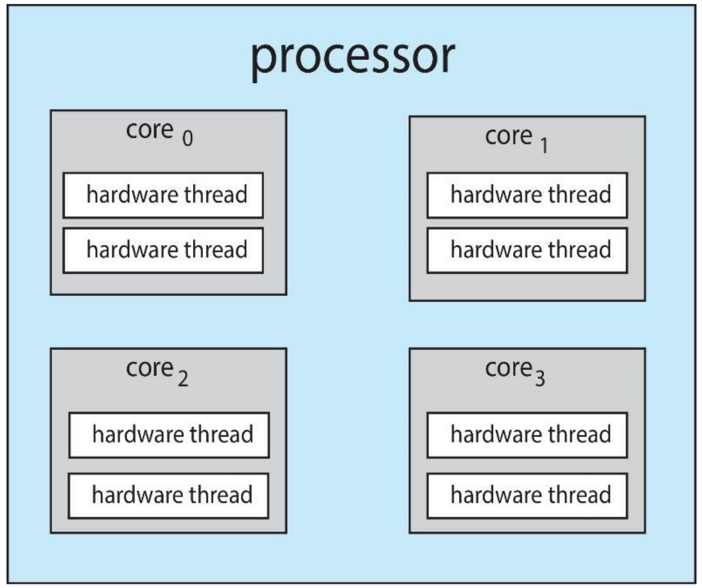


At t=50:

- Deadline For P1: 100ms
- Deadline for P2: 80ms
- Earliest Deadline First: P2 keeps executing
- P1 follows

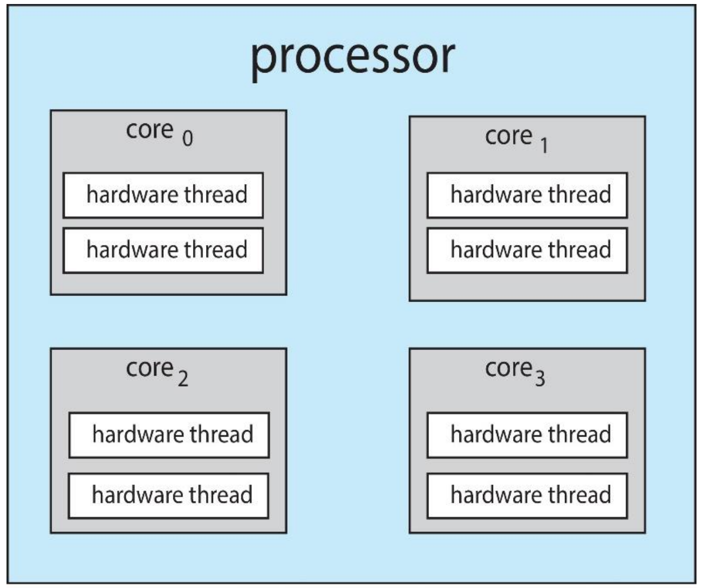
MULTITHREADED MULTICORE SYSTEM

- CPU scheduling more complex when multiple CPUs are available



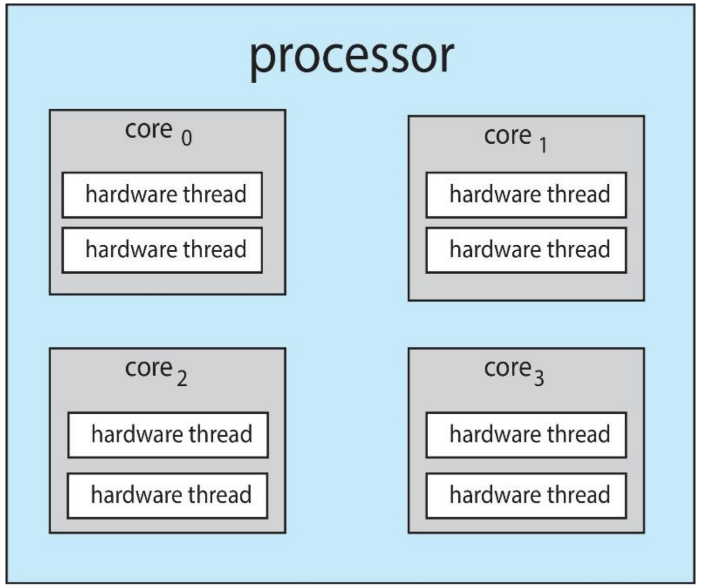
MULTITHREADED MULTICORE SYSTEM

- **Homogeneous processors** within a multiprocessor: all processors/cores are the same.
- **Heterogeneous processors** within a multiprocessor: more than one core/processor model.



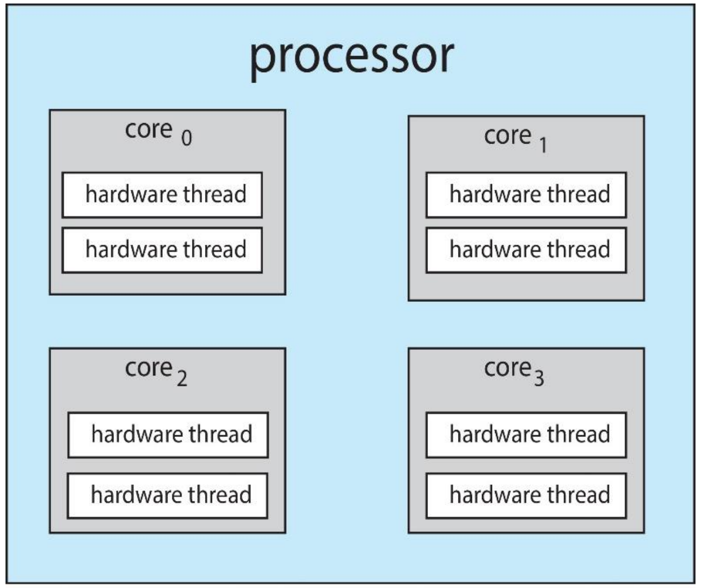
MULTITHREADED MULTICORE SYSTEM

- **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing
- **Symmetric multiprocessing** (**SMP**) – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
  - Currently, most common



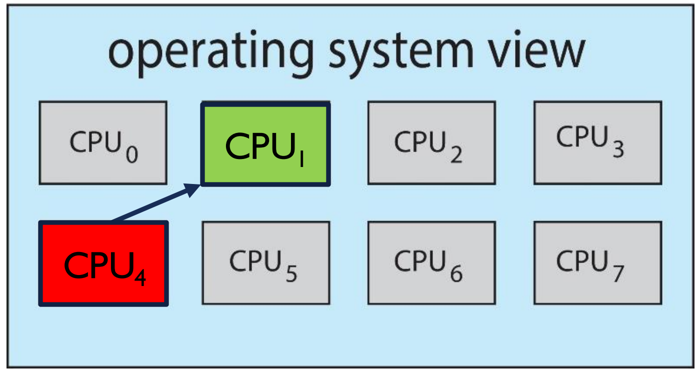
PROCESSOR AFFINITY

- **Processor affinity** – process has affinity for processor on which it is currently running
  - **soft affinity**
  - **hard affinity**
- **Soft Affinity**– process will resist switching to another core. CPU has to be highly overloaded for it to migrate.
- **Hard affinity**– process will never switch to another core.



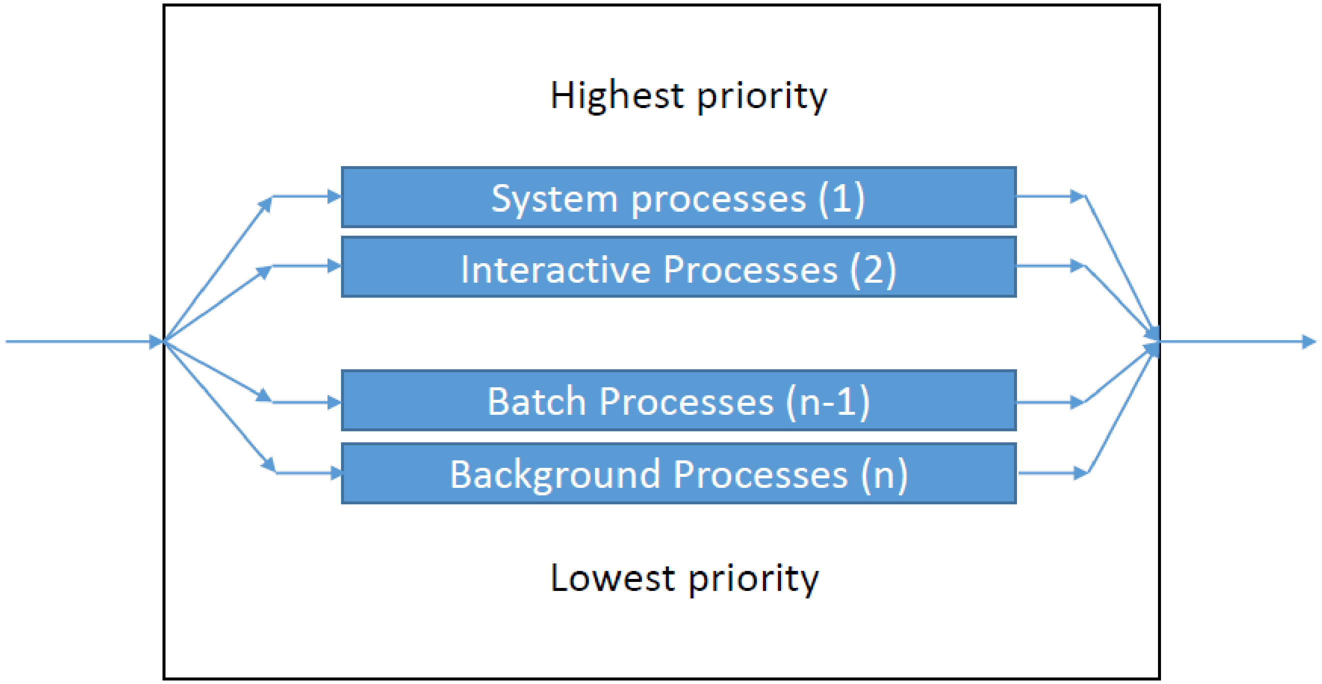
LOAD BALANCING

- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task from busy processor



WINDOWS SCHEDULING

- Windows uses priority-based preemptive scheduling
- Highest-priority thread runs next
- Thread runs until (1) blocks, (2) uses time slice, (3) preempted by higher-priority thread
- Real-time threads can preempt non-real-time
- 32-level priority scheme
- **Variable class** is 1-15, **real-time class** is 16-31
- Queue for each priority
- If no run-able thread, runs **idle thread**



LINUX SCHEDULER

Completely Fair Scheduler (CFS)

- **Scheduling classes**
  - Each has specific priority
  - Scheduler picks highest priority task in highest scheduling class

LINUX SCHEDULER

Completely Fair Scheduler (CFS)

- **Scheduling classes**
  - Each has specific priority
  - Scheduler picks highest priority task in highest scheduling class
  - Rather than quantum based on fixed time allotments, based on proportion of CPU time
  - Quantum is longer when the processor have less load.

LINUX SCHEDULER

Completely Fair Scheduler (CFS)

- Quantum calculated based on **nice value** from -20 to +19
  - Lower value is higher priority
- CFS scheduler maintains per task **virtual run time** in variable **`vruntime`**
  - Associated with decay factor based on priority of task – lower nice value is higher decay rate
  - Normal default priority yields virtual run time = actual run time

LINUX SCHEDULER

Completely Fair Scheduler (CFS)

- Quantum calculated based on **nice value** from -20 to +19
  - Lower value is higher priority
- CFS scheduler maintains per task **virtual run time** in variable **`vruntime`**
  - Associated with decay factor based on priority of task – lower nice value is higher decay rate
  - Normal default priority yields virtual run time = actual run time
- Virtual runtime does not affect quantum duration. It simply keeps track of how often a task was scheduled.
- To decide next task to run, scheduler picks task with lowest virtual run time

LINUX SCHEDULER

Completely Fair Scheduler (CFS)

- Quantum calculated based on **nice value** from -20 to +19
  - Lower value is higher priority
- CFS scheduler maintains per task **virtual run time** in variable **`vruntime`**
  - Associated with decay factor based on priority of task – lower nice value is higher decay rate
  - Normal default priority yields virtual run time = actual run time
- Virtual runtime does not affect quantum duration. It simply keeps track of how often a task was scheduled.
- To decide next task to run, scheduler picks task with lowest virtual run time

Aging

DEADLOCKS

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource

DEADLOCKS

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes

DEADLOCKS

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task

DEADLOCKS

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

FINAL EXAM

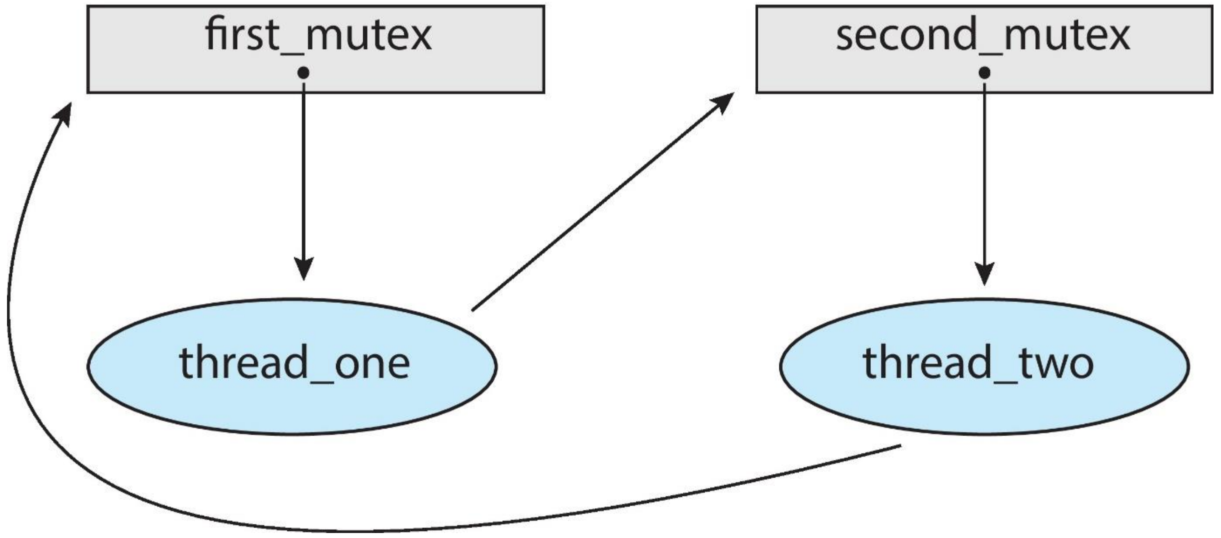
- Comprehensive
- 2 Hours
- Tuesday, March 14th from 1:00 to 3:00 pm on Canvas (online exam)
- Two pages of notes (single sheet).

FINAL EXAM

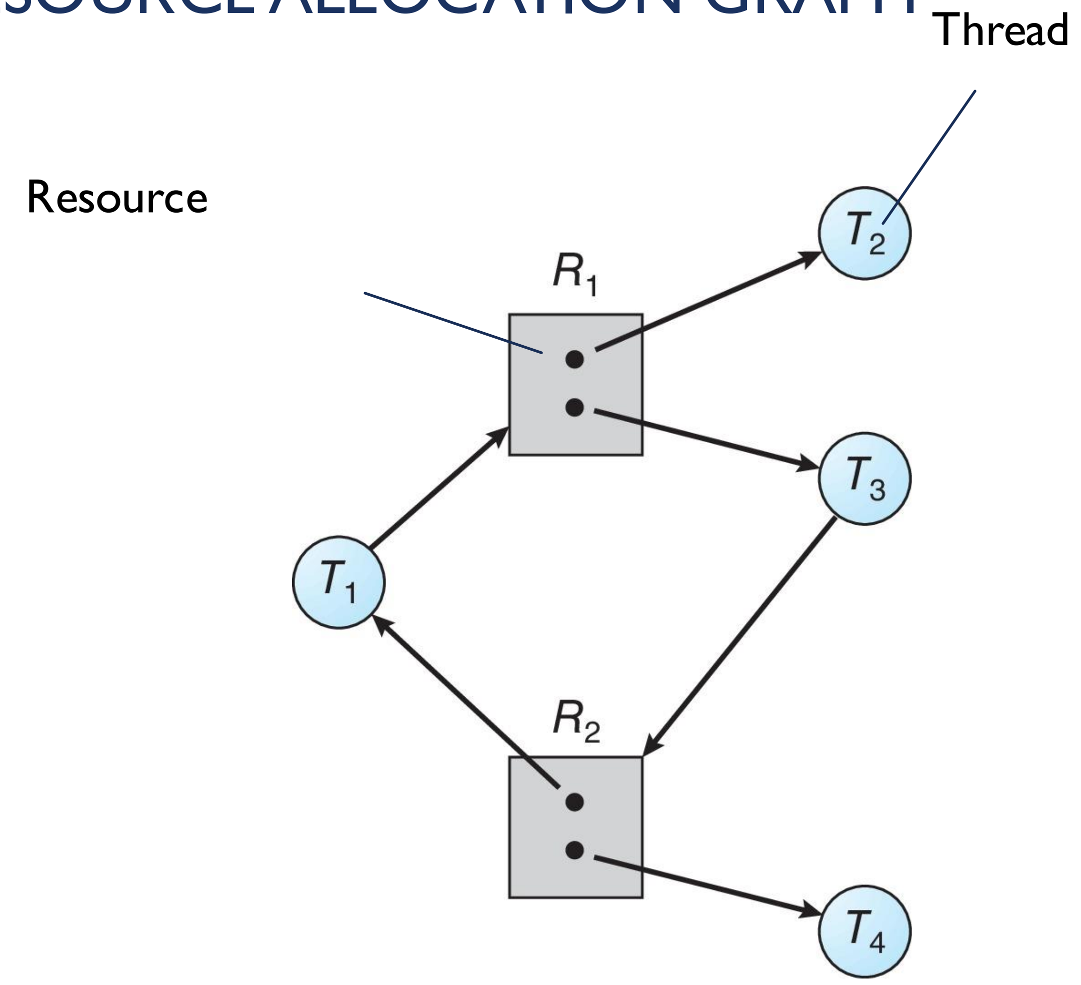
- Multiple Choice
- True/False
- Essay questions.
- Questions will include a wide variety of topics.
- Questions guaranteed to be there:
  - FS and their implementation
  - Segmentation and TLB/Paging question: similar to worksheet
  - Scheduling (be ready for real-time or non-real time scheduling question)
- Review quizzes and worksheets: some questions will be very similar.
- This is not a comprehensive list of questions!

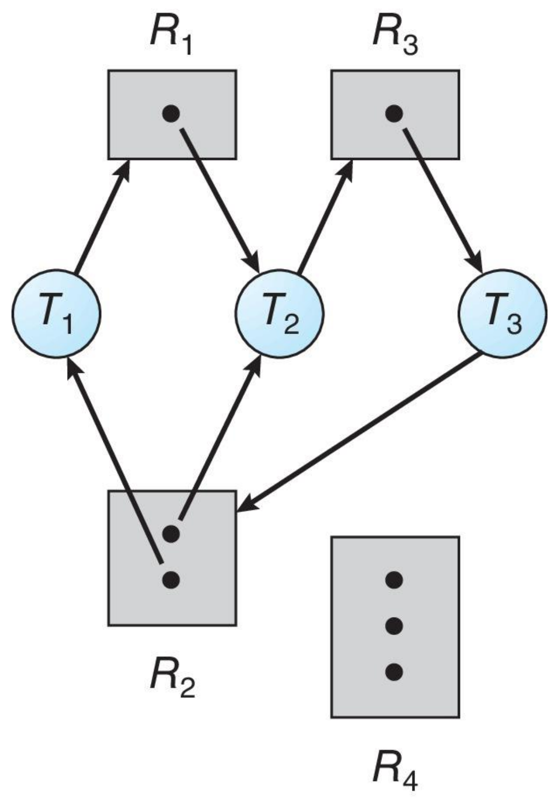
DEADLOCKS IN MULTITHREADED APPLICATION

- Mutual Exclusion can cause deadlocks when not well planned.
- Can be illustrated with a **resource allocation graph**:

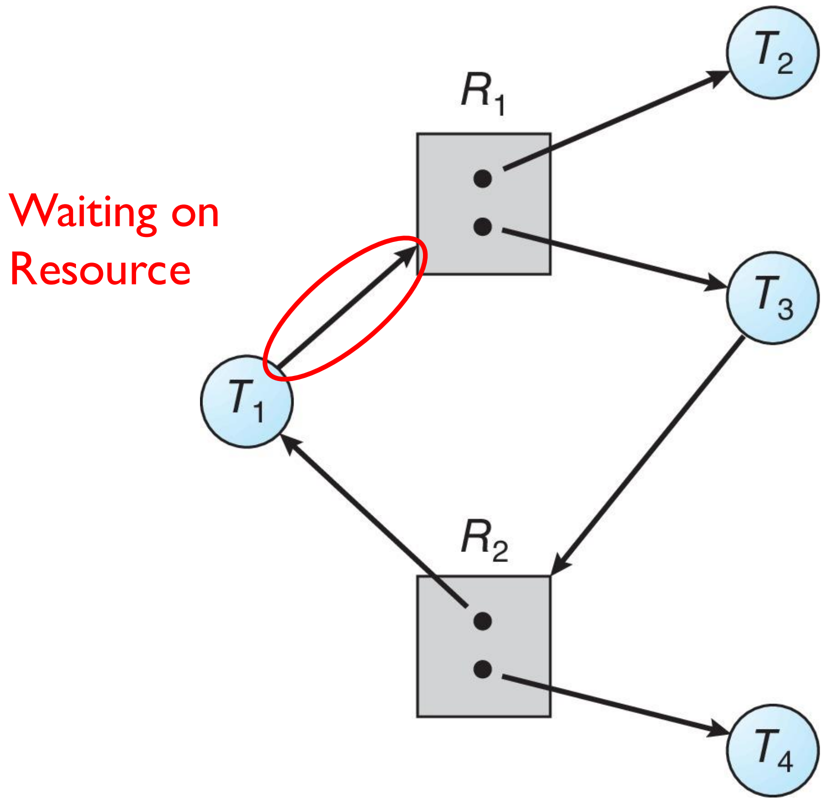


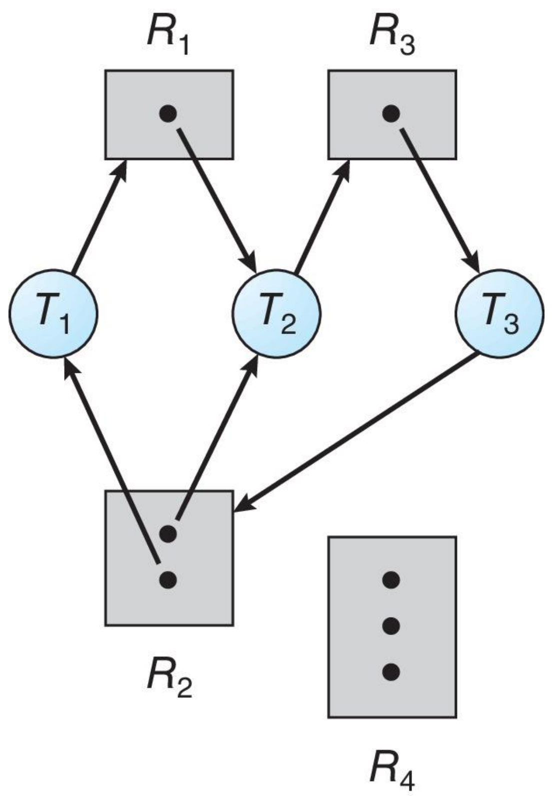
RESOURCE ALLOCATION GRAPH



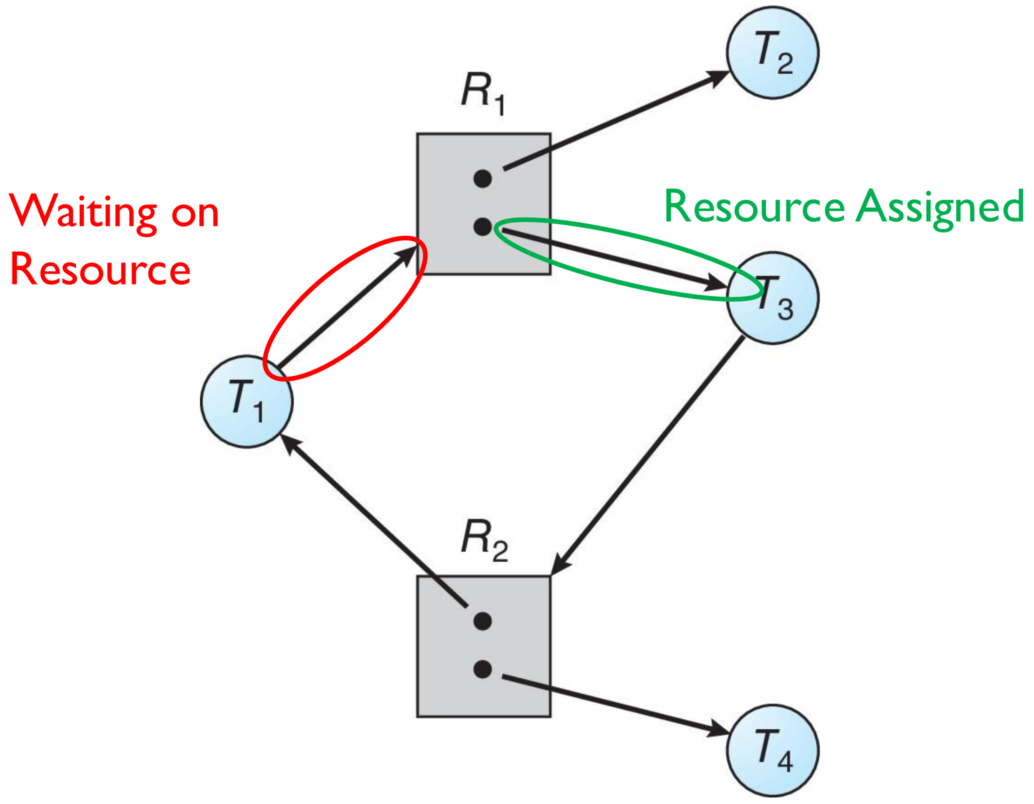


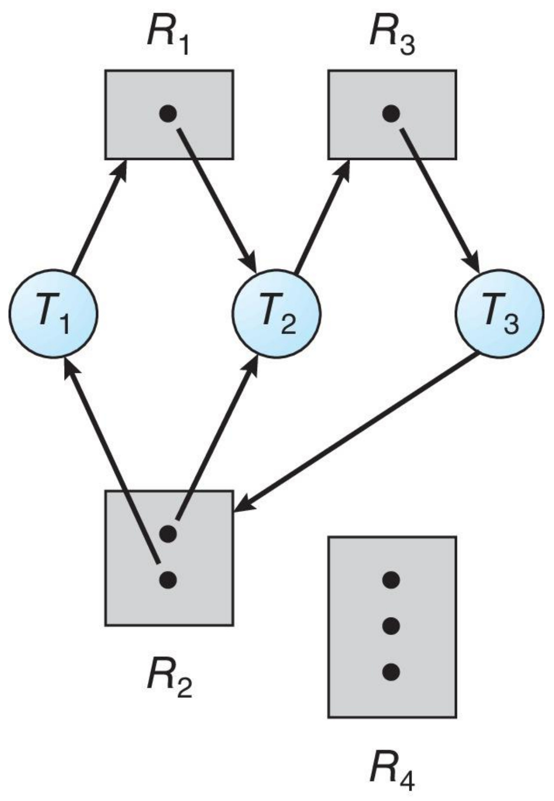
RESOURCE ALLOCATION GRAPH



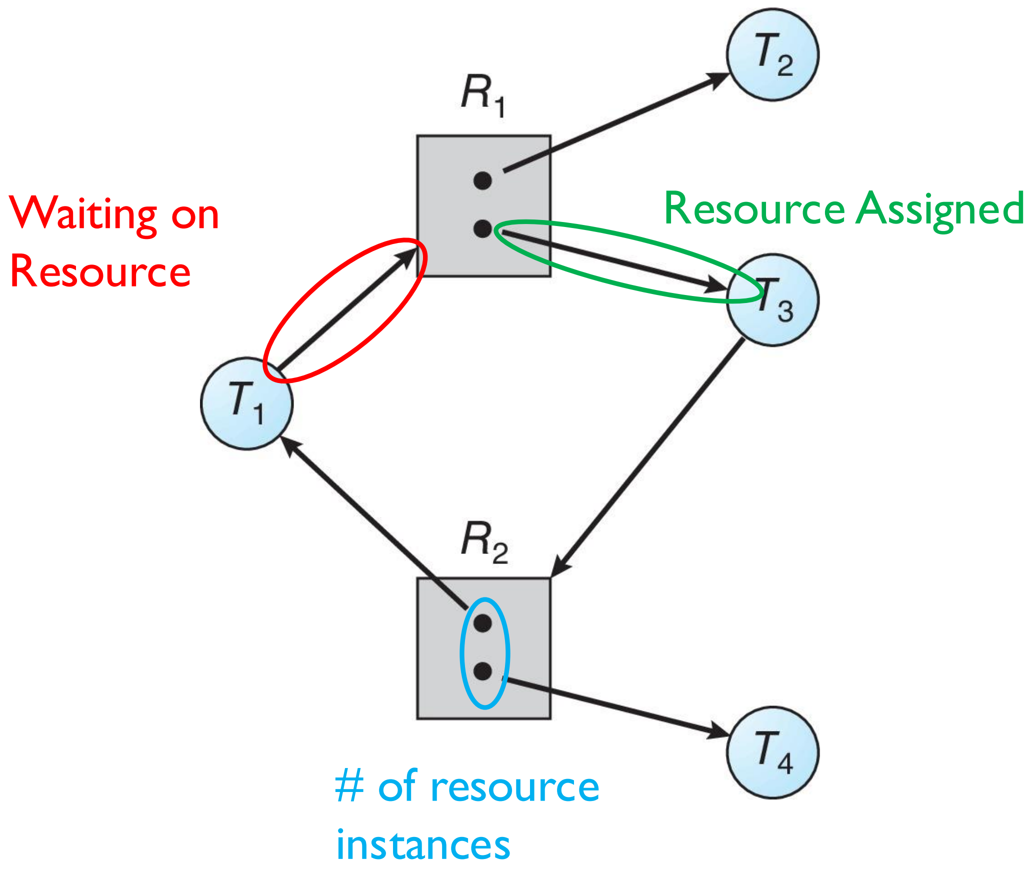


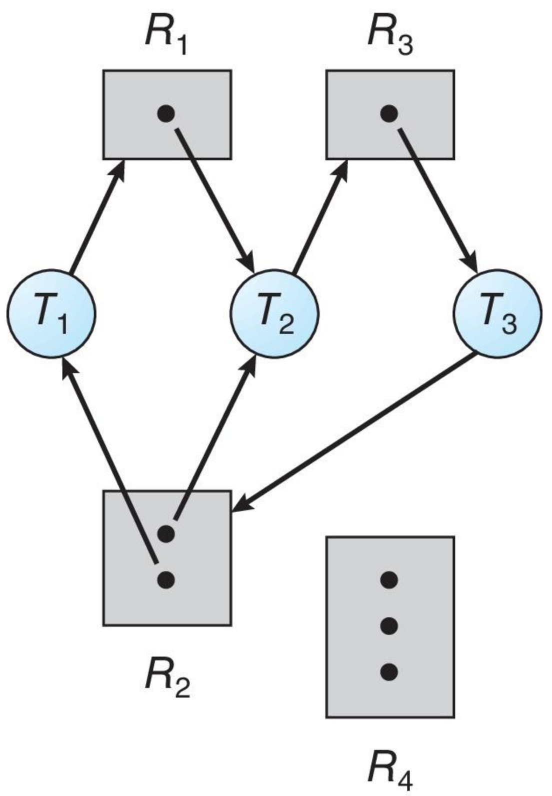
RESOURCE ALLOCATION GRAPH



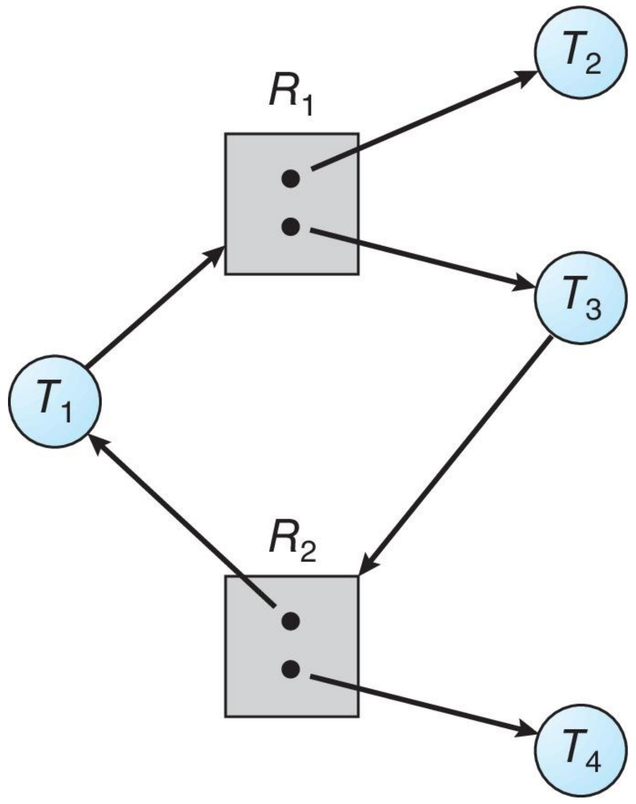


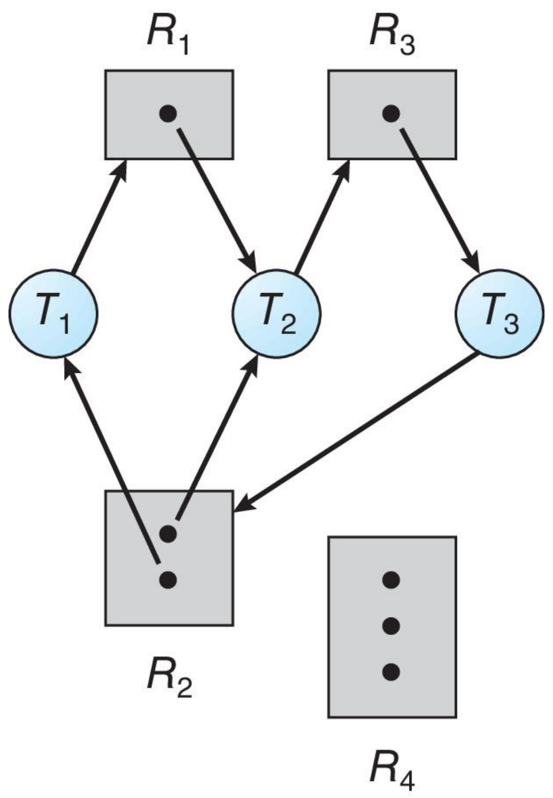
RESOURCE ALLOCATION GRAPH



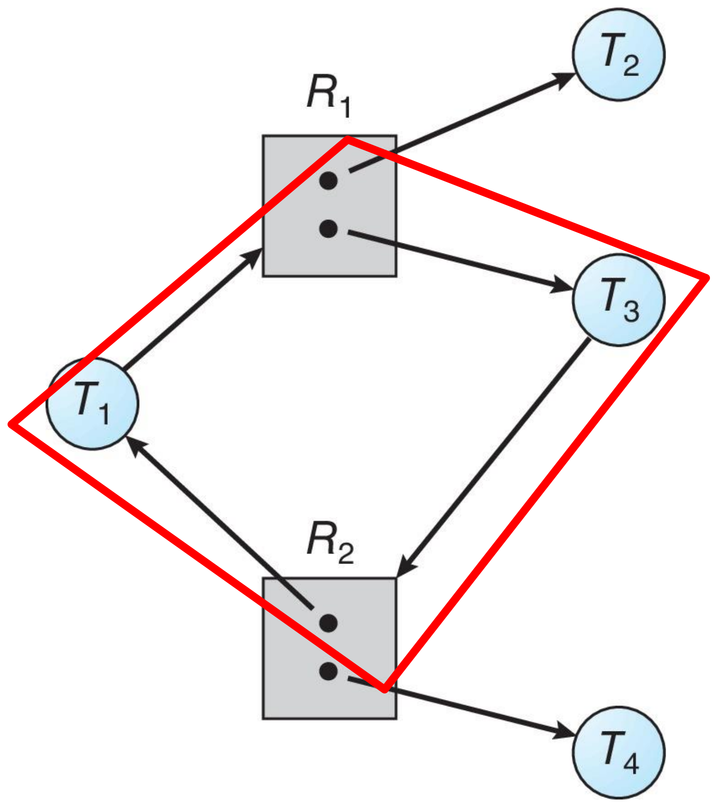


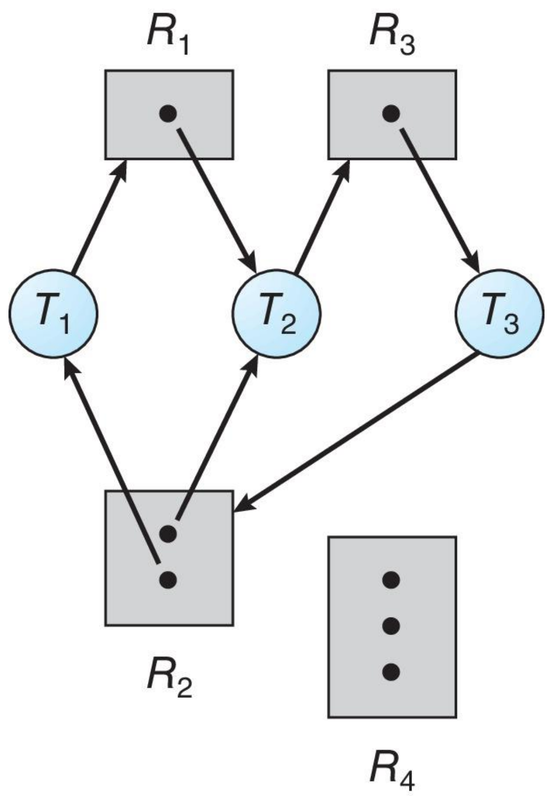
RESOURCE ALLOCATION GRAPH: CLOSED LOOP



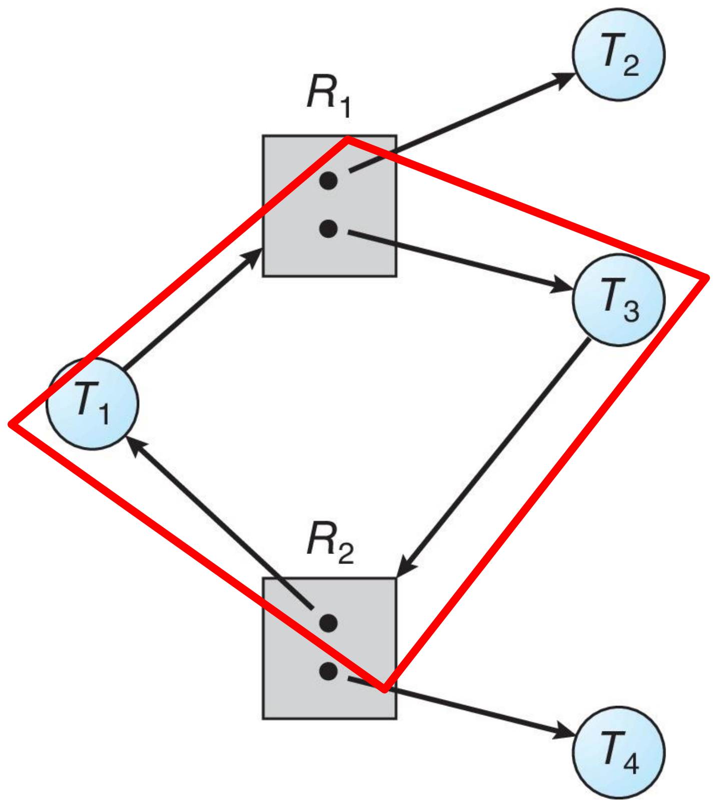


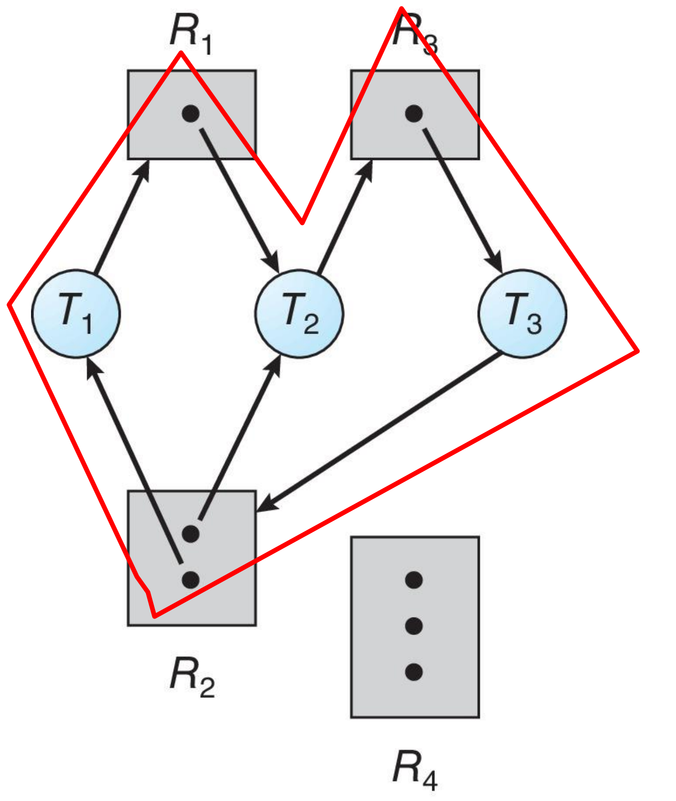
RESOURCE ALLOCATION GRAPH: CLOSED LOOP



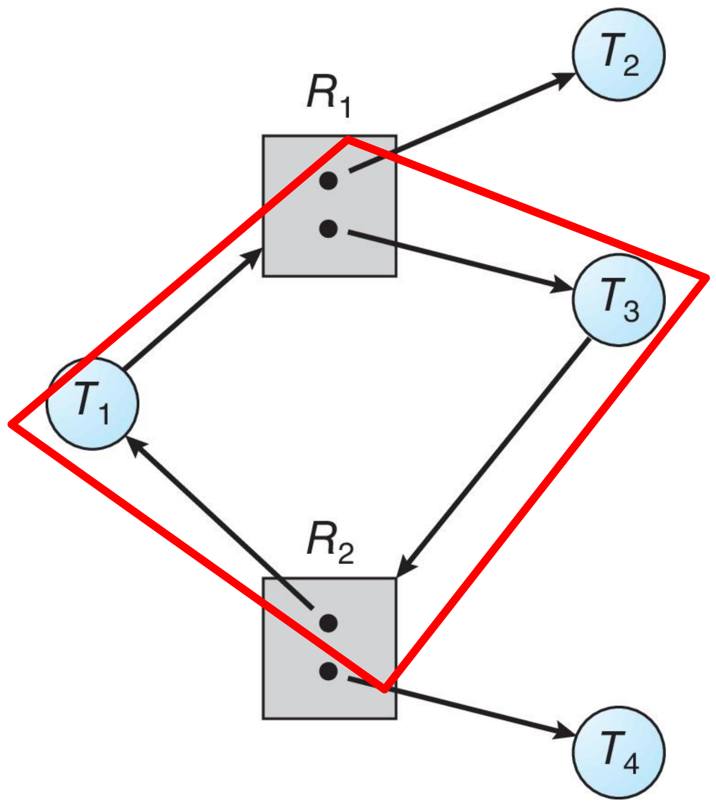


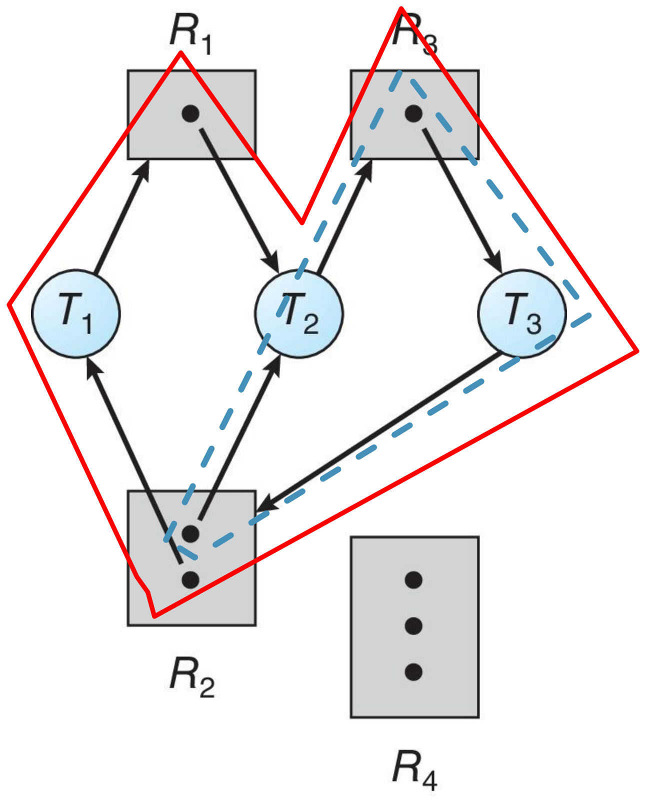
RESOURCE ALLOCATION GRAPH: CLOSED LOOP



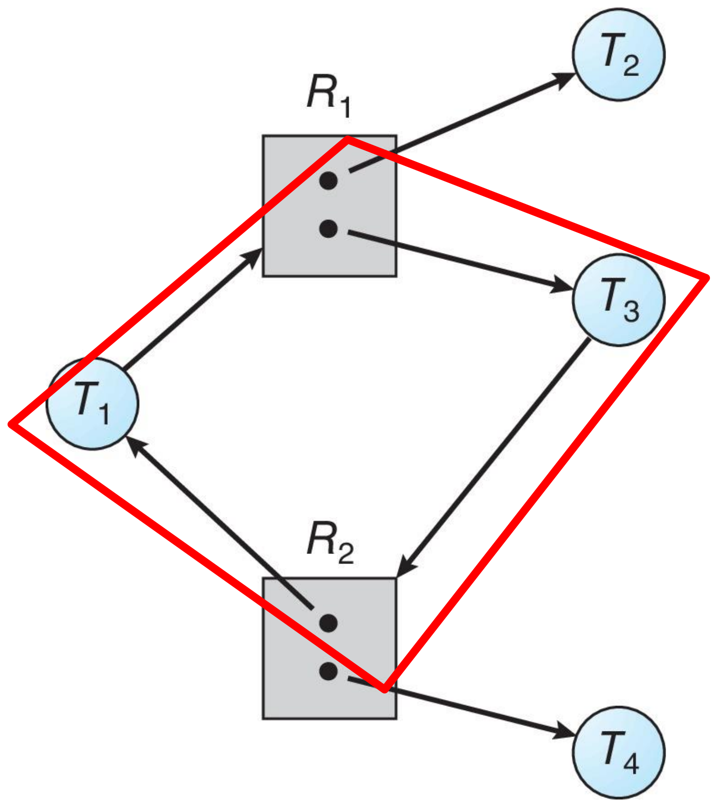


RESOURCE ALLOCATION GRAPH: CLOSED LOOP

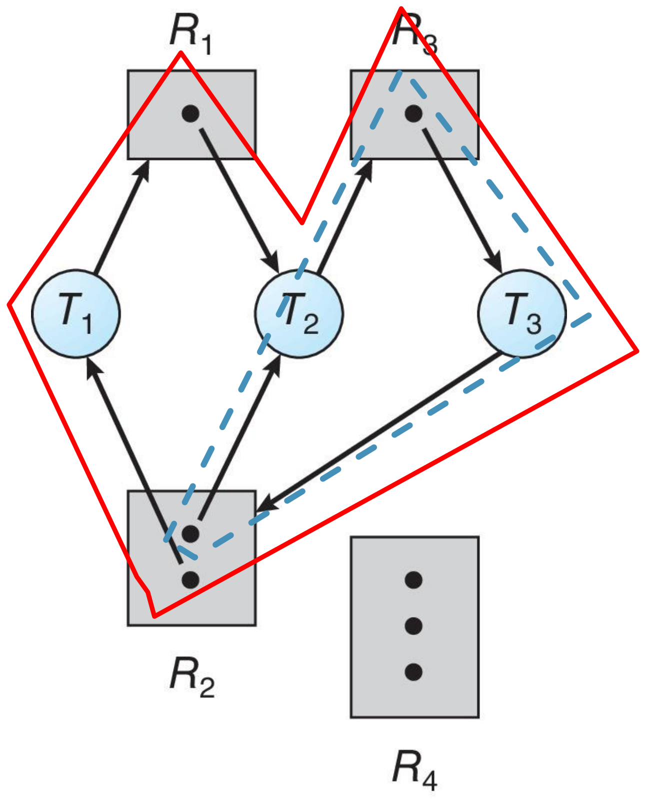




RESOURCE ALLOCATION GRAPH: CLOSED LOOP



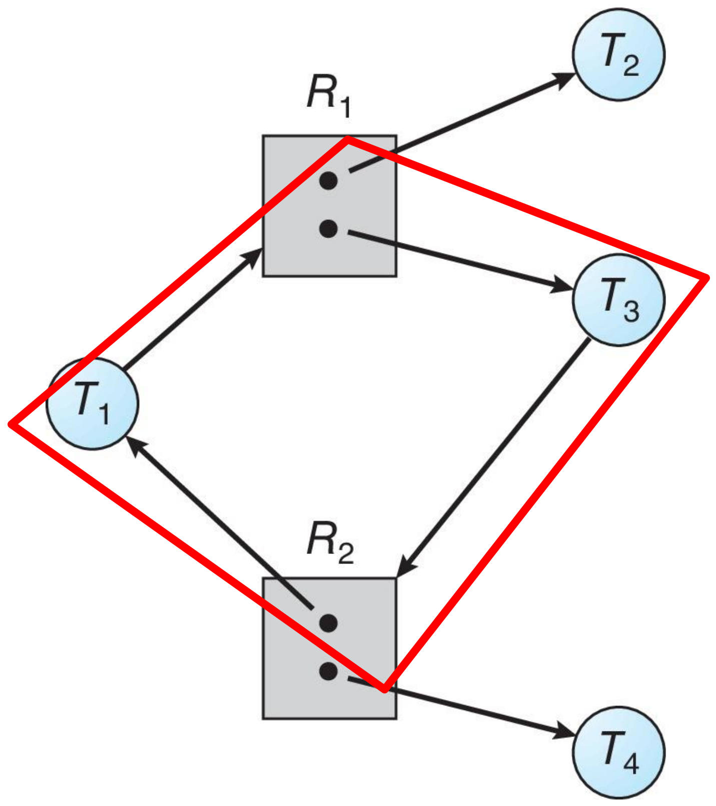
A



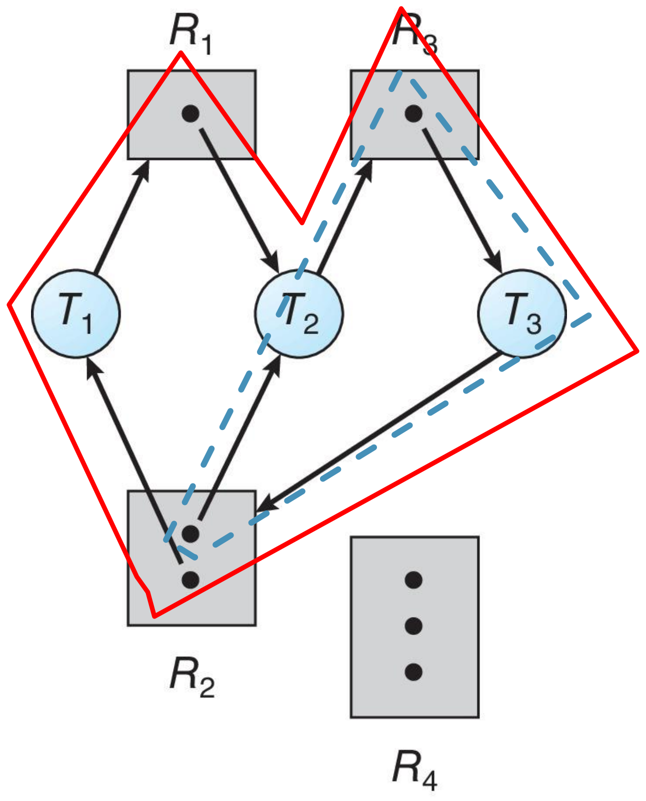
B

RESOURCE ALLOCATION GRAPH: CLOSED L

A	B	A: Deadlock only at A
		B: Deadlock only at B
C	D	C: Both Deadlocked
		D: None

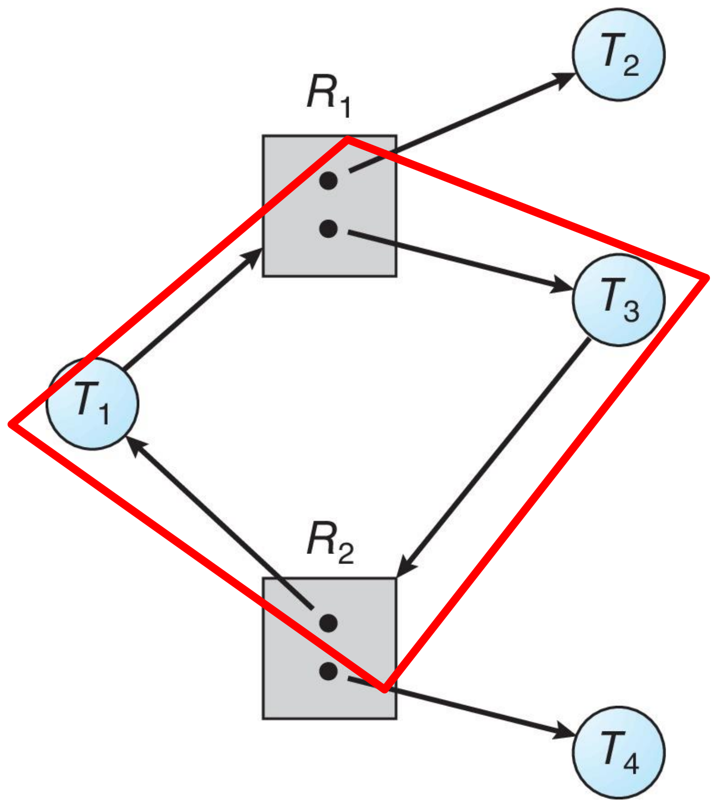


A

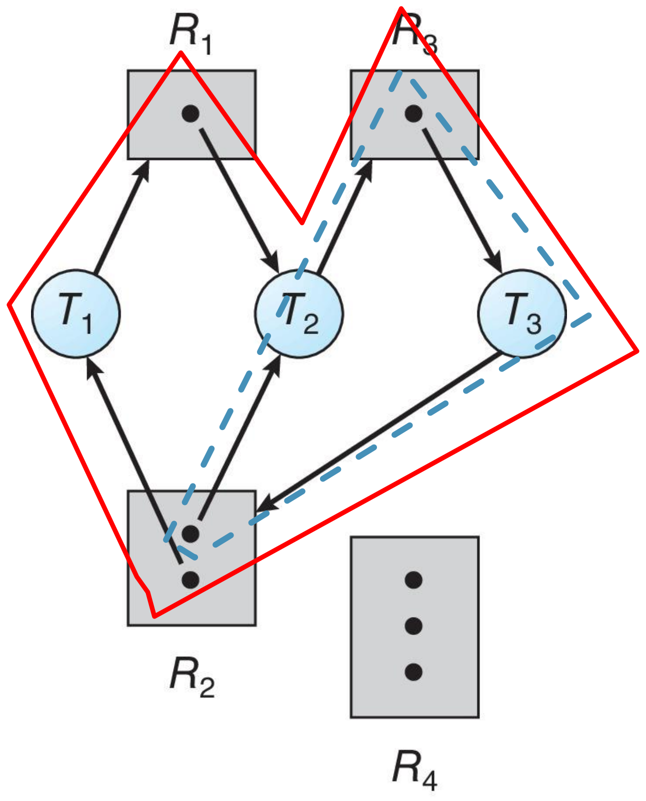


B

RESOURCE ALLOCATION GRAPH: CLOSED LOOP

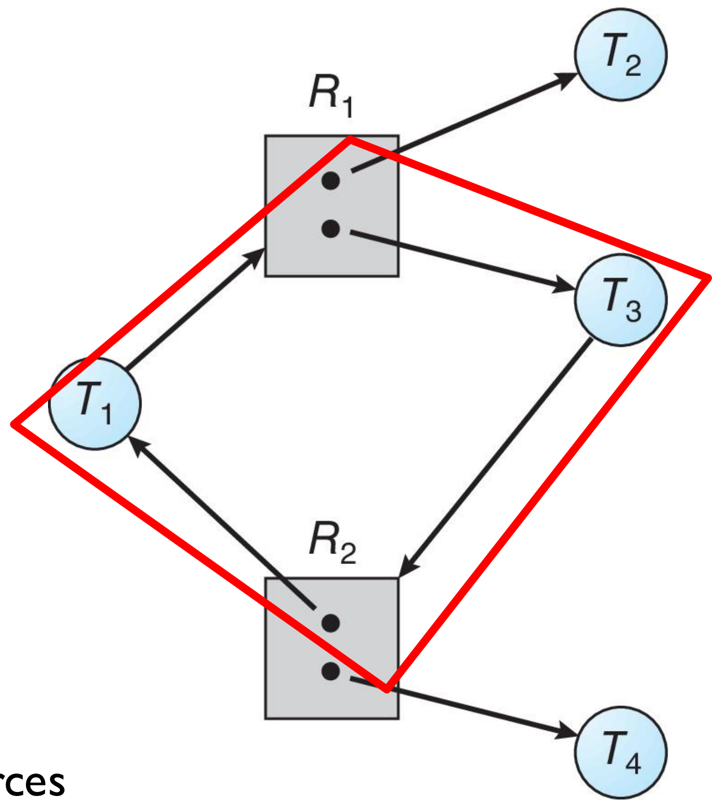


A: No Deadlock



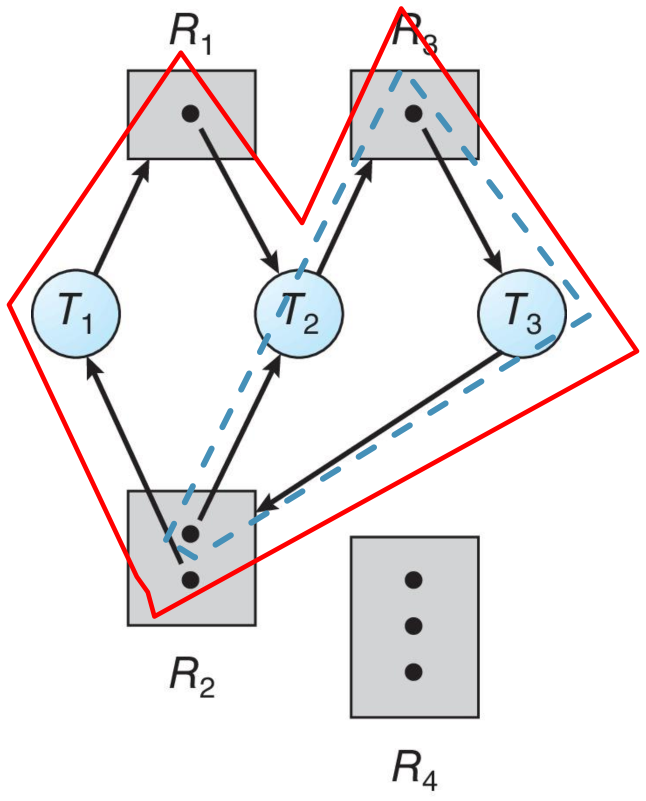
B: Deadlocked

RESOURCE ALLOCATION GRAPH: CLOSED LOOP



Some resources are assigned to threads outside the loop.

A: No Deadlock



B: Deadlocked

All resources instances are part of the loop threads.

HANDLING DEADLOCKS

Three approaches for deadlocks:

- Prevent/Avoid Deadlocks
- Detect and Recover
- Ignore Deadlocks

HANDLING DEADLOCKS

Three approaches for deadlocks:

- Prevent/Avoid Deadlocks
- Detect and Recover
- Ignore Deadlocks

- Operating Systems ignore deadlocks!

HANDLING DEADLOCKS

Three approaches for deadlocks:

- Prevent/Avoid Deadlocks
- Detect and Recover
- Ignore Deadlocks

- Operating Systems ignore deadlocks!
- As a programmer, you have to make sure to use mutual exclusion properly.