CS 452 Operating System Concepts

CPU Scheduling

Dr. Denton Bobeldyk

Scheduling Algorithms

- First-Come First Served
- Shortest Job First
- Priority Scheduling
- Round Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

Scheduling Algorithms

- **First-Come First Served**
- Shortest Job First
- Priority Scheduling
- Round Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

First Come First Serve

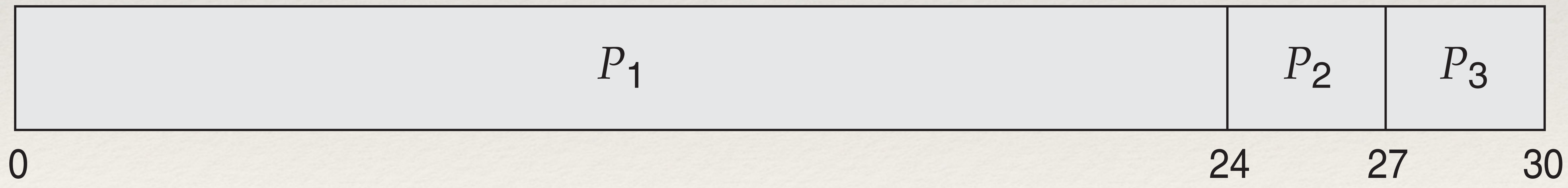
- First process to request the CPU, gets it 😁
- Managed with a FIFO queue (First In First Out)

Evaluating FCFS

Process	Burst Time
P1	24
P2	3
P3	3

Evaluating FCFS

Process	Burst Time
P1	24
P2	3
P3	3



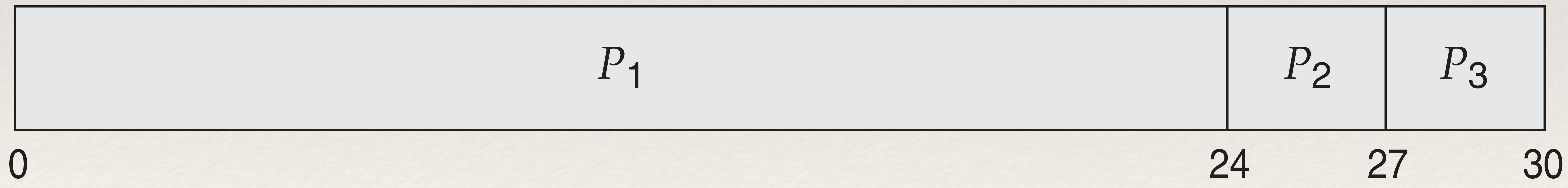
Use of Gant Chart to help visualize!

Scheduling Algorithm Metric

- Need a metric to compare CPU scheduling algorithms
  - Calculate the average wait time for each process

Evaluating FCFS

Process	Burst Time
P1	24
P2	3
P3	3

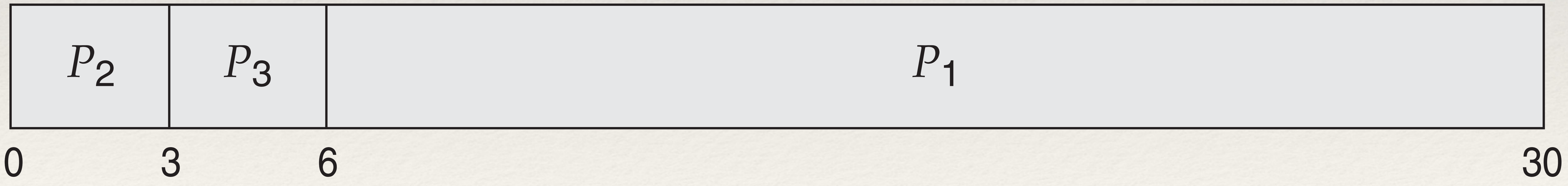


Average Wait Time = (0 + 24 +27) / 3 = 17 milliseconds

Evaluating FCFS

What if the processes arrive in a different order?

Process	Burst Time
P2	3
P3	3
P1	24

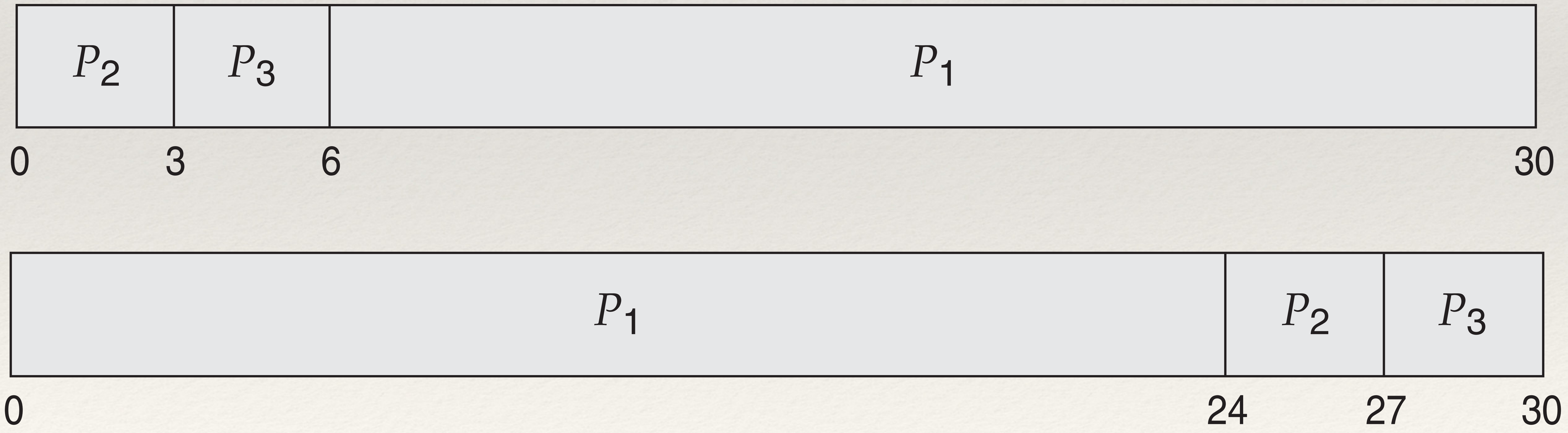


Average Wait Time = (0 + 3 +6) / 3 = 3 milliseconds

FCFS Thoughts

FCFS average wait time is heavily dependent on:

1. The burst time of the processes arriving
2. The order in which they arrive



Evaluating FCFS - Individual Exercise

Process	Burst Time
P1	12
P2	1
P3	4

What is the average wait time?

Evaluating FCFS - Individual Exercise

Process	Burst Time
P1	12
P2	1
P3	4

Average Wait Time = (0 + 12 +13) / 3 = 8.33 milliseconds

FCFS Scheduling Algorithm

1. Non preemptive
2. Convoy Effect
    - All processes need to wait for one “big” process to complete

Scheduling Algorithms

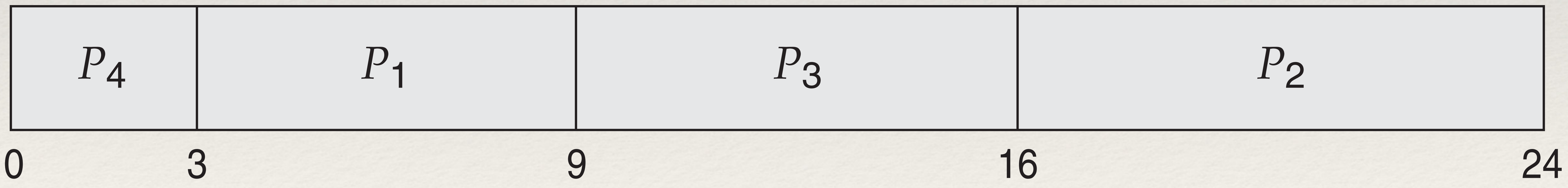
- First-Come First Served
- **Shortest Job First**
- Priority Scheduling
- Round Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

Shortest Job First (SJF)

- When the CPU is available, it is assigned the processes that has the smallest next CPU burst

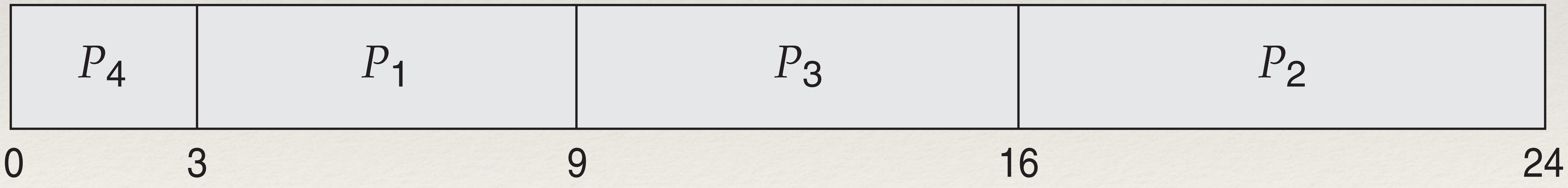
Shortest Job First (SJF)

Process	Burst Time
P1	6
P2	8
P3	7
P4	3



Shortest Job First (SJF)

Process	Burst Time
P1	6
P2	8
P3	7
P4	3



Average Wait Time = (0 + 3 + 9 + 16) / 4 = 7 milliseconds

SJF vs FCFS

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

SJF: Average Wait Time = (0 + 3 + 9 + 16) / 4 = 7 milliseconds
FCFS: Average Wait Time = (0 + 6 + 14 + 21) / 4 = 10.25 milliseconds

SJF Complications

- What is the burst time for each of the processes arriving at the CPU???

SJF Complications - Group Exercise

- What is the burst time for each of the processes arriving at the CPU???

How can the burst time be determined?

Calculating Burst Time

- Predicted as an **exponential average** of the measured length of previous CPU bursts
    - Let τ_{n+1} be the predicted value of the next burst
    - Let t_n = length of nth CPU burst
    - Then for α, $0 \le \alpha \le 1$
        - α determines the weight of recent and past history cpu burst times
    - $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$

Calculating Burst Time

- $\tau_{n+1} = \alpha \ t_n + (1 - \alpha)\tau_n$
  - t_n contains the most recent information
  - τ_n stores the past history
- α controls the relative weight of recent and past history
- If $\alpha = 0$, then recent history has no effect
- If $\alpha = 1$, then only the most recent CPU burst is relevant
- More commonly $\alpha = 1/2$
- Initial τ (τ_0) can be defined as a constant or as an overall system average

Calculating Burst Time

$$\alpha = 0$$

$$\tau_{n+1} = 0 \ t_n + (1 - 0)\tau_n$$

$$\tau_{n+1} = \tau_n$$

Calculating Burst Time

$$\alpha = 1$$

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

$$\tau_{n+1} = t_n$$

Calculating Burst Time

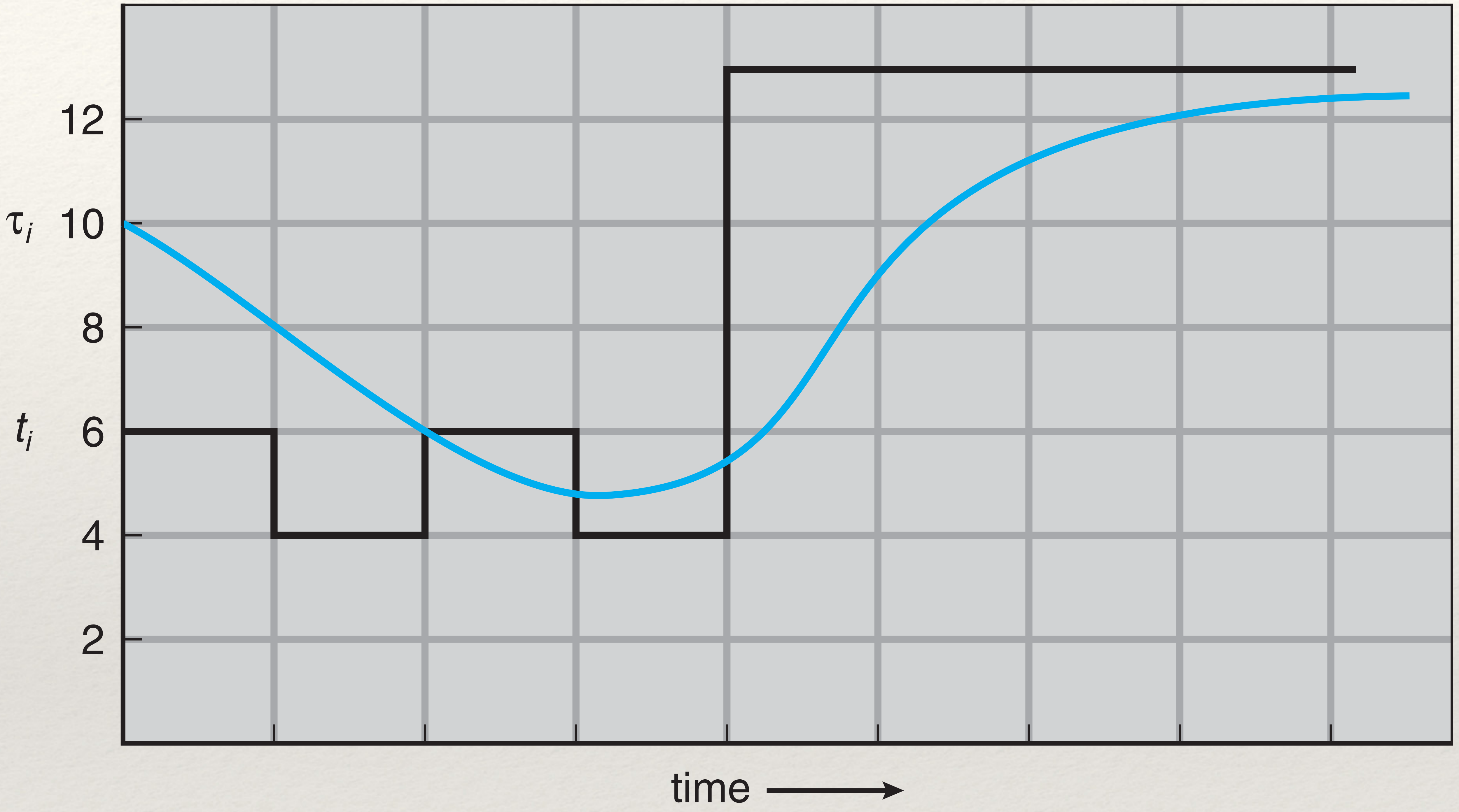
$$\tau_{n+1} = \alpha \; t_n + (1 - \alpha)\alpha t_{n-1} + \ldots + (1 - \alpha)^j \alpha t_{n-j} + \ldots + (1 - \alpha)^{n+1}\tau_0$$

t_n contains the most recent information

τ_n stores the past history

Calculating Burst Time

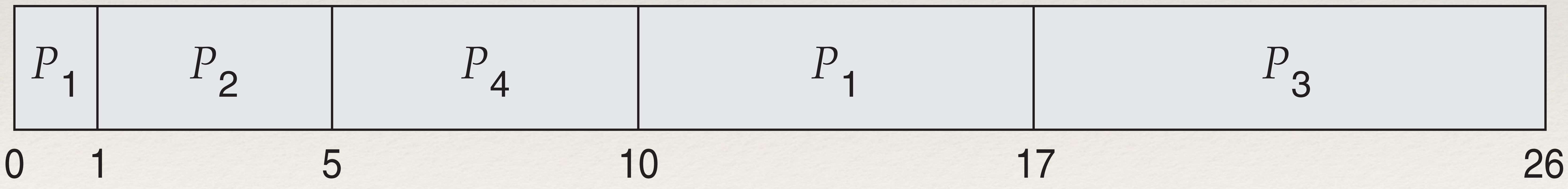
$\tau_0 = 10, \alpha = 1/2$



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)

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



Average Wait Time =

SJF with Preemption

Wait Time Per Job = Start Time - Arrival Time

Wait Time for a preempted job:

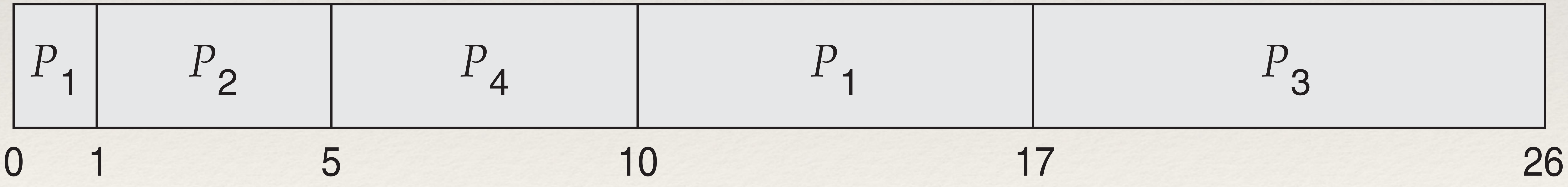
(Time of first execution - arrival time) +

$$\sum_{2}^{n}(\text{startTime\_i - endTime\_i-1}) \,)$$

Where n is the number of times the process was preempted + 1

Shortest Job First (SJF)

Process	Arrival Time	Burst Time	Wait Time
P1	0	8	(0-0) + (10-1) = 9
P2	1	4	1-1 = 0
P3	2	9	17-2 = 15
P4	3	5	5 -3 = 2



Average Wait Time = 9 + 0 + 15 + 2 = 26/4 = 6.5 milliseconds

SJF Algorithm

- Preemptive or nonpreemptive
- Preemptive SJF scheduling is sometimes called shortest-remaining-time-first

Scheduling Algorithms

- First-Come First Served
- Shortest Job First
- **Priority Scheduling**
- Round Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

Priority Scheduling

- A priority is associated with each process
- The CPU is allocated to the process with the highest priority

Priority Scheduling - Number Representation

Which do we use?

- The lower the number, the higher the priority.
- The higher the number, the higher the priority.

Priority Scheduling - Number Representation

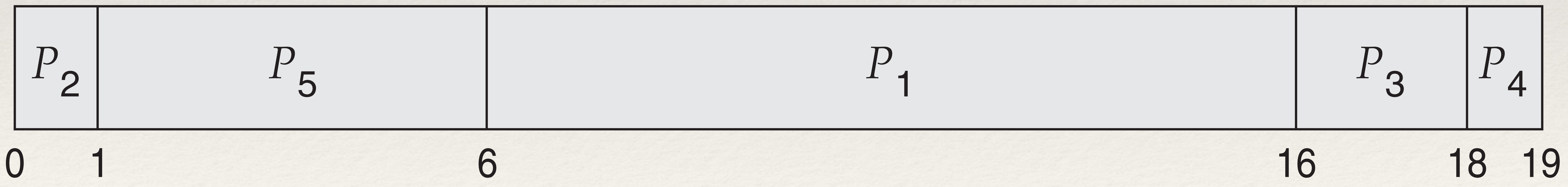
- **The lower the number, the higher the priority. (Textbook uses this)**
- The higher the number, the lower the priority.

Priority Scheduling

- SJF is an example of priority scheduling
  - The larger the CPU burst size, the lower the priority
  - The smaller the CPU burst size, the higher the priority

Priority Scheduling

Process	Arrival Time	Burst Time	Priority	Wait Time
P1	0	10	3	6
P2	0	1	1	0
P3	0	2	4	16
P4	0	1	5	18
P5	0	5	2	1



Average Wait Time = 6 + 0 + 16 + 18 + 1 = 41/5 = 8.2 milliseconds

Priority Scheduling - Problem

- Indefinite Blocking - Starvation can occur

Priority Scheduling - Problem

- Indefinite Blocking - Starvation can occur
- How can this be solved?

Priority Scheduling - Problem

- Indefinite Blocking - Starvation
  - Can add an ‘age’ to each process
    - The longer a process waits, the better their priority becomes

Scheduling Algorithms

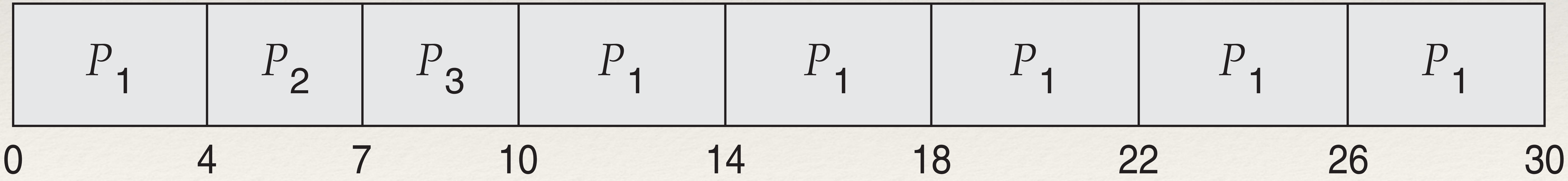
- First-Come First Served
- Shortest Job First
- Priority Scheduling
- **Round Robin Scheduling**
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

Round Robin Scheduling

- Ready Queue is FIFO
- Time Slice (Quantum) is defined
  - Generally from 10-100ms in length
- CPU scheduler allocates CPU to each process for one time slice
- New processes are added to the tail of the queue

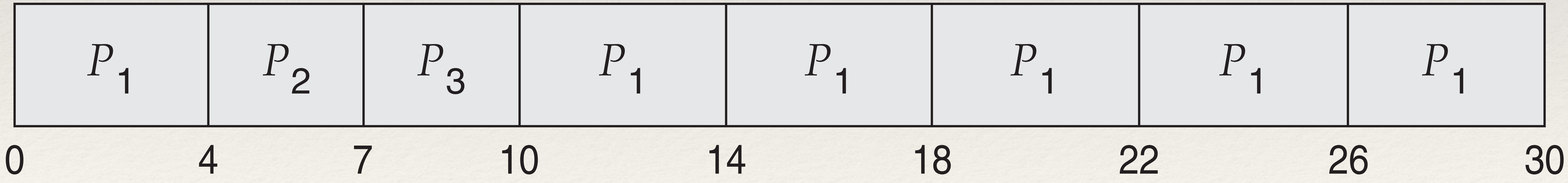
Round Robin Scheduling

Process	Arrival Time	Burst Time
P1	0	24
P2	0	3
P3	0	3



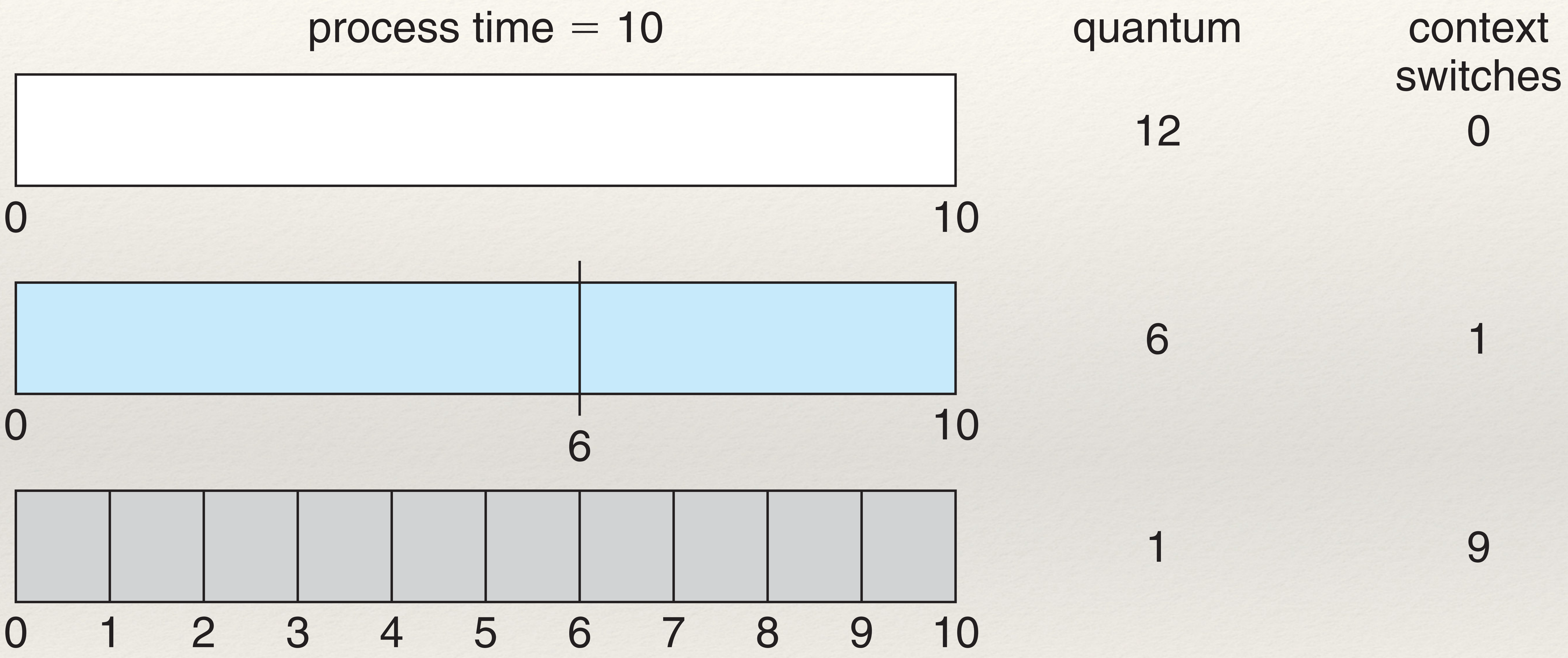
Round Robin Scheduling

Process	Arrival Time	Burst Time
P1	0	24
P2	0	3
P3	0	3



Average Wait Time = (10-4) + 4 + 7 = 17/3 = 5.66 milliseconds

Round Robin Scheduling



Round Robin Scheduling

- If the time quantum is really high - same as FCFS
- If the time quantum is really small - a lot of overhead doing process switching
- Rule of Thumb: 80 percent of the CPU bursts should be less than the time slice/quantum (no context switching for those processes)

Scheduling Algorithms

- First-Come First Served
- Shortest Job First
- Priority Scheduling
- Round Robin Scheduling
- **Multilevel Queue Scheduling**
- Multilevel Feedback Queue Scheduling

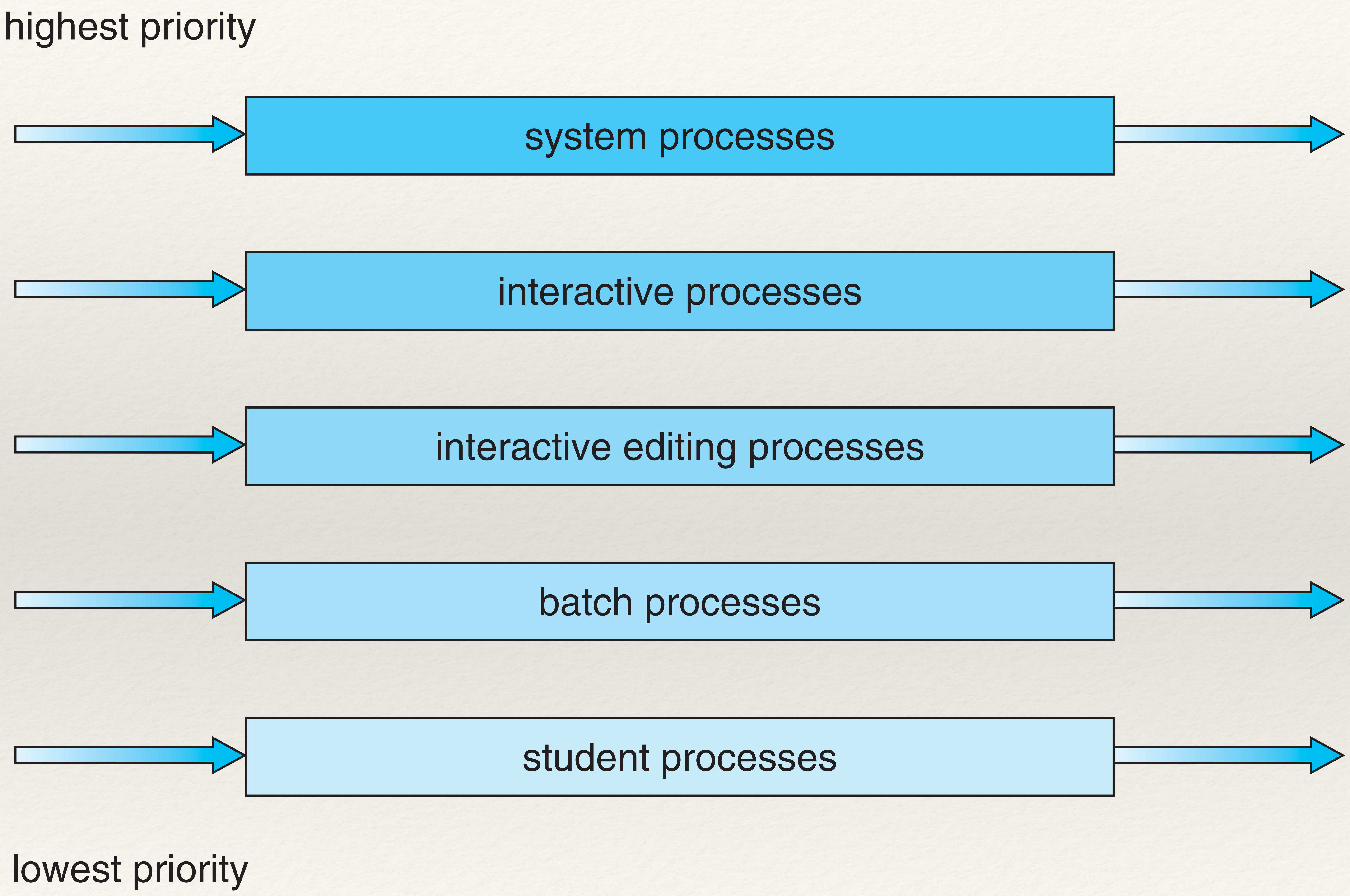
Multi-level Queue Scheduling

- Ready Queue is partitioned into separate queues:
  - Foreground (interactive)
  - Background (batch)
- Each Queue has its own scheduling algorithm
  - Foreground - round robin
  - Background - FCFS

Multi-level Queue Scheduling

- CPU Scheduling must occur between the queues
  - Fixed priority scheduling
    - Serve all foreground, then background
  - Time slice
    - 80% to foreground using RR
    - 20% to background using FCFS

Multilevel queue scheduling



Scheduling Algorithms

- First-Come First Served
- Shortest Job First
- Priority Scheduling
- Round Robin Scheduling
- Multilevel Queue Scheduling
- **Multilevel Feedback Queue Scheduling**

Multi-level Feedback Scheduling

- Allows processes to move between queues
- Separate processes by their CPU burst size
  - If it uses too much time, move to a lower priority queue

Multi-level Feedback Scheduling

- Schedule all processes in queue 0
- Once queue 0 is empty, process those in queue 1
- Once queue 0 and 1 are empty, process those in queue 2

Multi-level Feedback Scheduling

- Defined by:
  - The number of queues
  - The scheduling algorithm for each queue
  - Method used to determine initial queue
  - Method used to determine promotion/demotion to a different queue

Multiple-Processor Scheduling

- Multiple CPUs available
- Load share over each of the CPUs
- Scheduling can be more complex
- Define **homogenous** as processors that are identical in terms of their functionality

Multi-Processor Scheduling

- Asymmetric multiprocessing
- Symmetric multiprocessing

Multi-Processor Scheduling

- **Asymmetric multiprocessing**
    - Master server (one processor)
        - Scheduling decisions, I/O processing, System activities
    - Other processors execute user only code
        - Reduces need for sharing data across processors
- Symmetric multiprocessing

Multi-Processor Scheduling

- Asymmetric multiprocessing
- Symmetric multiprocessing
  - Each processor examines the ready queue and selects a process to execute
  - Windows/Linux/Mac OS X use this type of scheduling

Symmetric multiprocessing (SMP)

- Processor Affinity
- Load Balancing

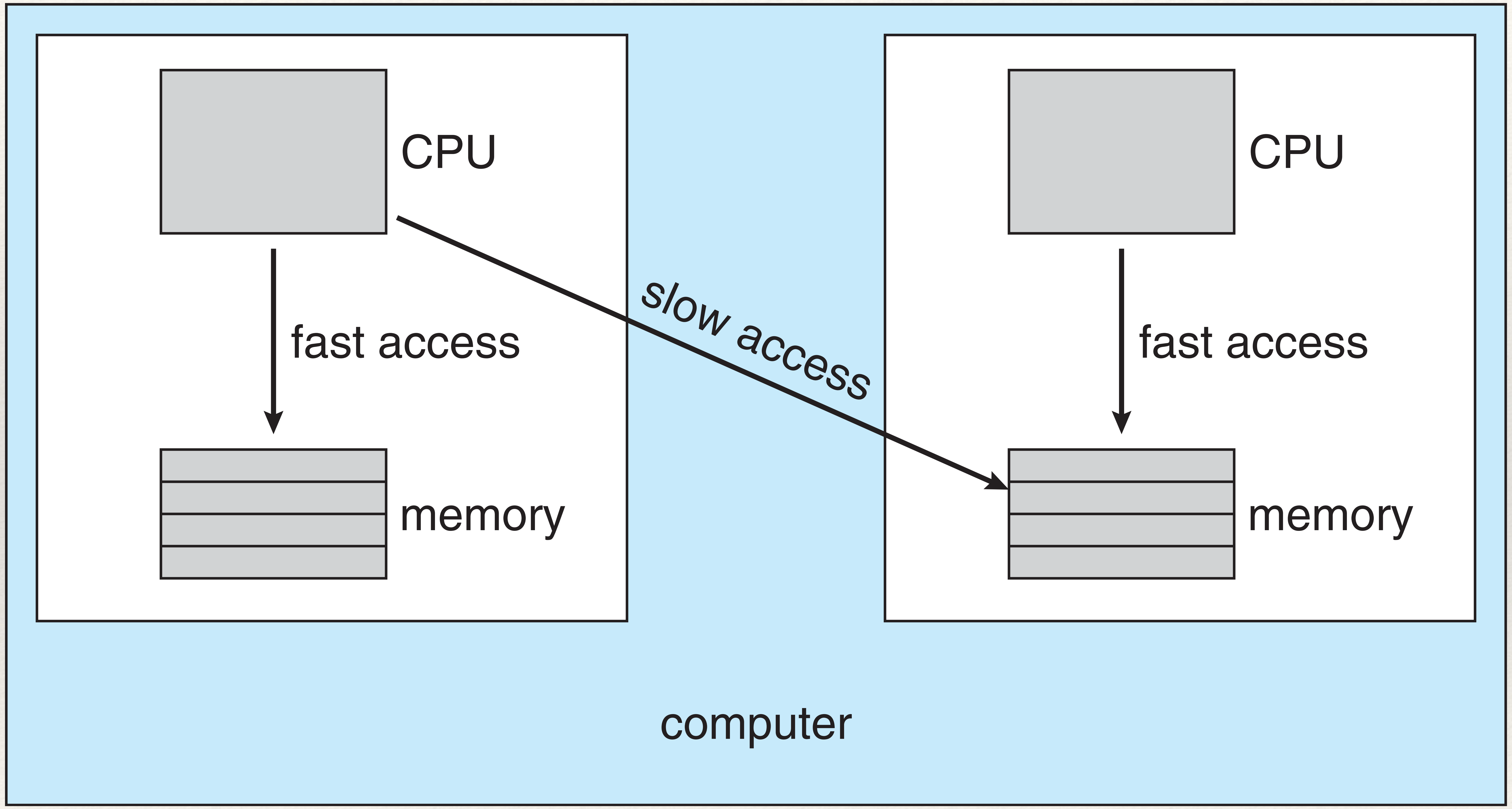
Processor Affinity

- Each processor can have cache memory
- If a process runs on a processor, the cache memory will be populated with data most recently accessed for that process
- If the process switches to another processor, the data cache from the original processor must be invalidated and any new data must be loaded into the cache for the new processor
- It is advantageous for a process to stay on the same processor, aka Processor Affinity

Processor Affinity

- Soft Affinity - attempt to keep on same processor, but no guarantee
- Hard Affinity - a process can specify a subset of processors on which it may run
- Linux provides the 'sched_setaffinity()' system call which supports hard affinity. Default is soft affinity

Processor Affinity



A visual of a CPU accessing slower access memory

Load Balancing

- Attempt to keep workload even across all processors
- Push Migration - A specific task periodically checks the load on each processor and evenly distributes the load by moving (or pushing) processes from overloaded processors to idle or less busy processors
- Pull Migration - An idle processor pulls a waiting task from a busy processor

Load Balancing

- Linux scheduler implements both techniques
  - Push & Pull

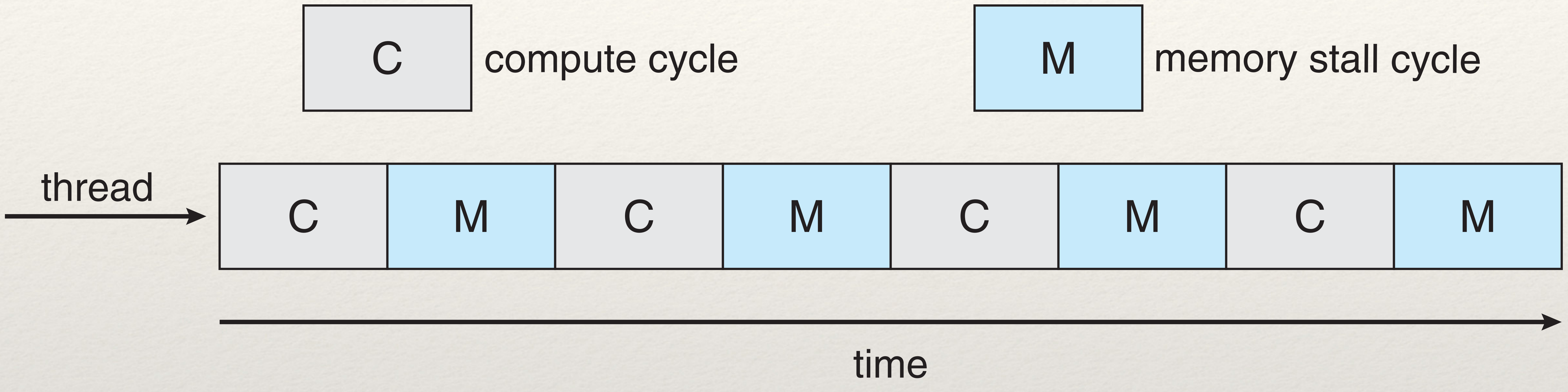
Multicore Processor

- Multiple processor cores on the same chip
- OS treats them as individual processor cores
- Faster and consume less power
- Complicates scheduling issues

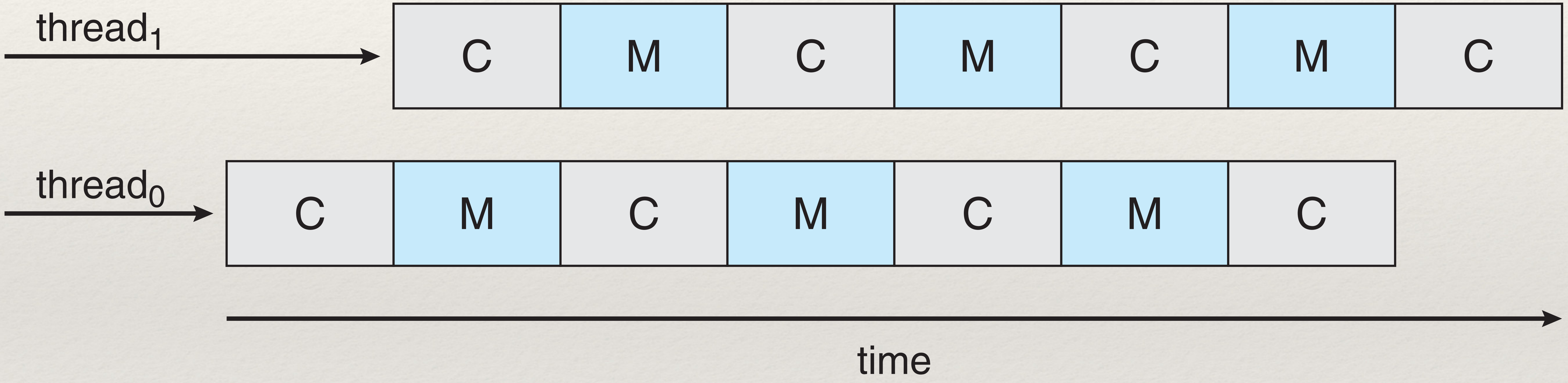
Multicore Processor

- Processors may spend a significant amount of time waiting for data to become available, this is called **memory stall**
- Multithreaded processor core: 2 or more hardware threads are assigned to a core
- Operating system treats each hardware thread as a logical processor

Multicore Processor



Multicore Processor

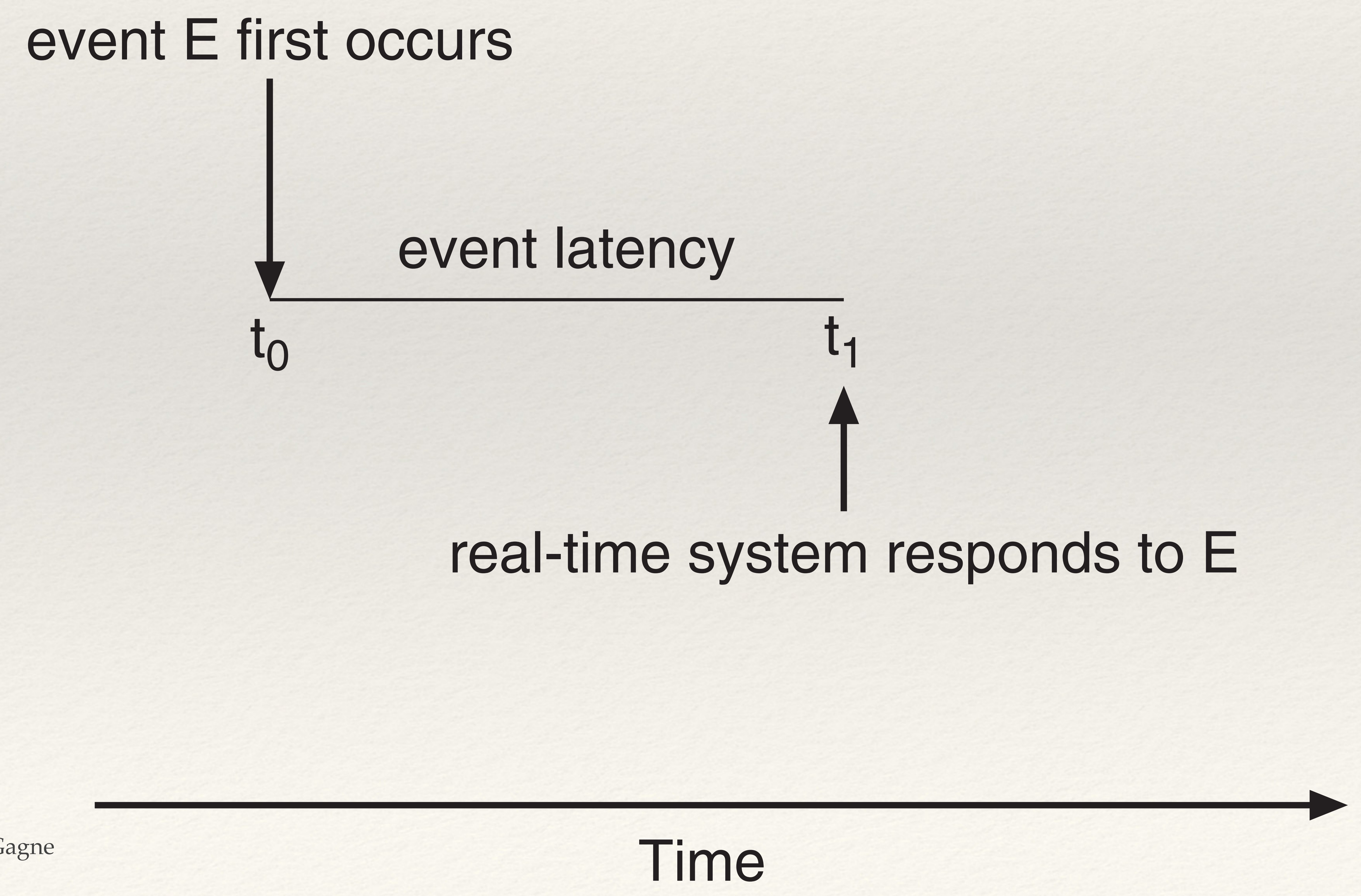


Real Time CPU Scheduling

- Soft real-time systems - no guarantee for critical real time process to be scheduled
- Hard real-time systems - stricter requirements, task must be serviced by it's deadline

Event Latency

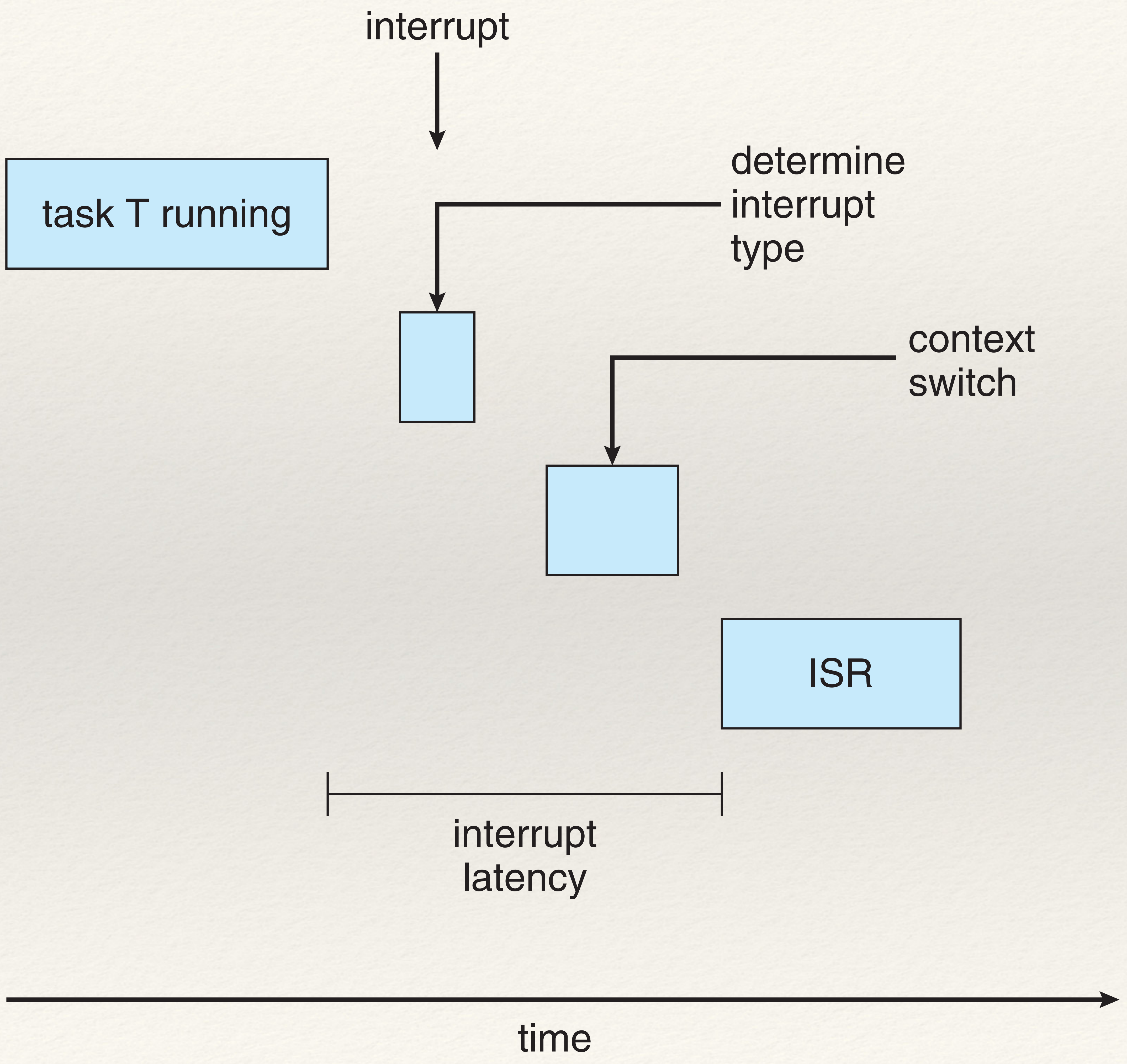
- Event Latency - amount of time that elapses from when an event occurs to when it's serviced



Types of Latency

- Interrupt Latency
- Dispatch Latency

Interrupt Latency

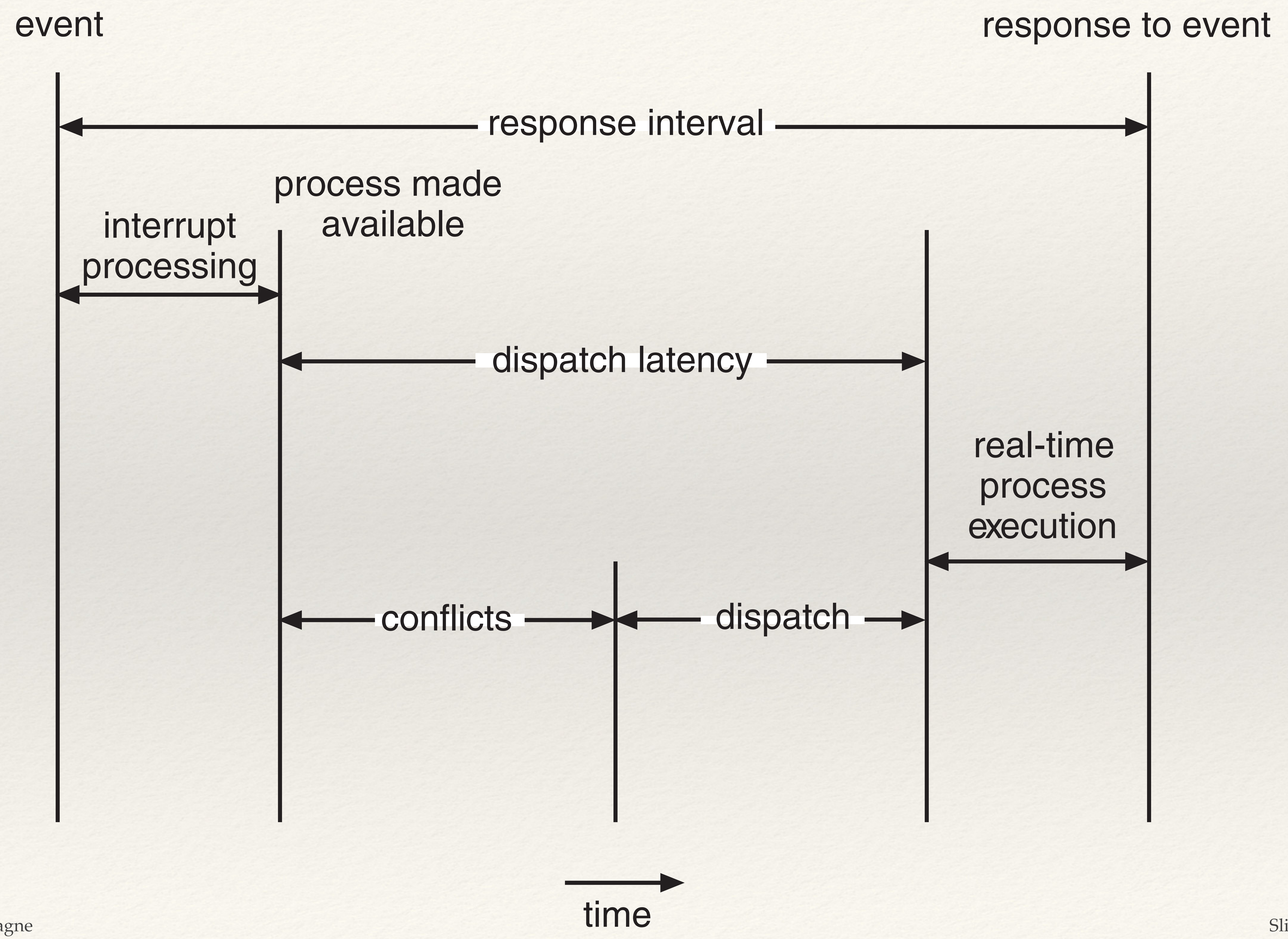


Interrupt Latency

- When an interrupt occurs
  - The OS must first complete the instruction it is executing and determine the type of interrupt that occurred.
  - Save the state of the current process
  - Service the interrupt using the Interrupt service routine (ISR)

Interrupt Latency is the time it takes to do the above tasks

Dispatch Latency



Dispatch Latency

- Scheduling dispatcher to stop one process and start another
- Conflict phase
  - Preemption of any process running in the kernel
  - Release of low priority processes of resources needed by a high-priority process

Priority based scheduling

Periodic processes require the CPU at certain intervals

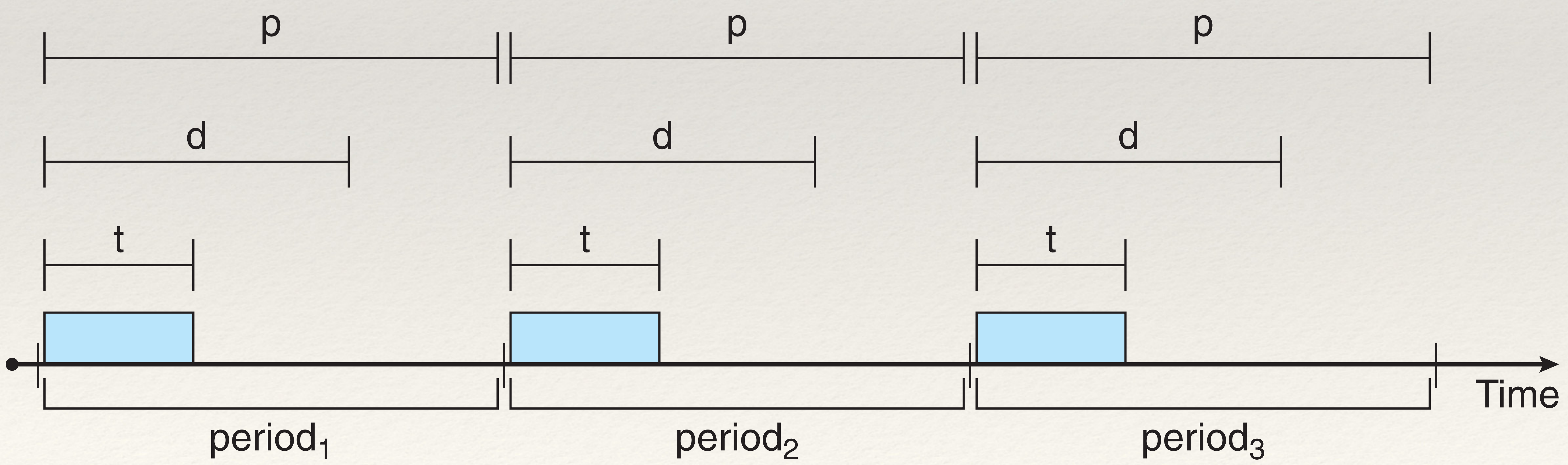
The scheduler can guarantee the requirements or reject them (Admission-control algorithm)

p = period

d = deadline

t = fixed processing time

The **rate** of a period task is 1/p



Rate-Monotonic Scheduling

- Used to schedule periodic tasks
- Assign a task a priority based on it's period
  - The shorter the period, the higher priority
  - The longer the period, the lower priority
- Assume each CPU burst is the same every period

Rate-Monotonic Example

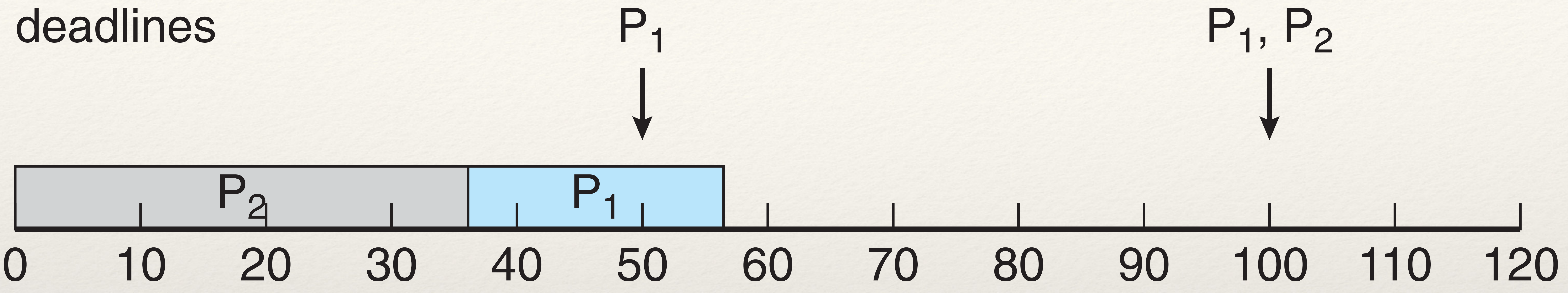
- Process One: $p_1 = 50,\ t_1 = 20\ d_1 = 50$
- Process Two: $p_2 = 100,\ t_2 = 35\ d_2 = 100$

Rate-Monotonic Example

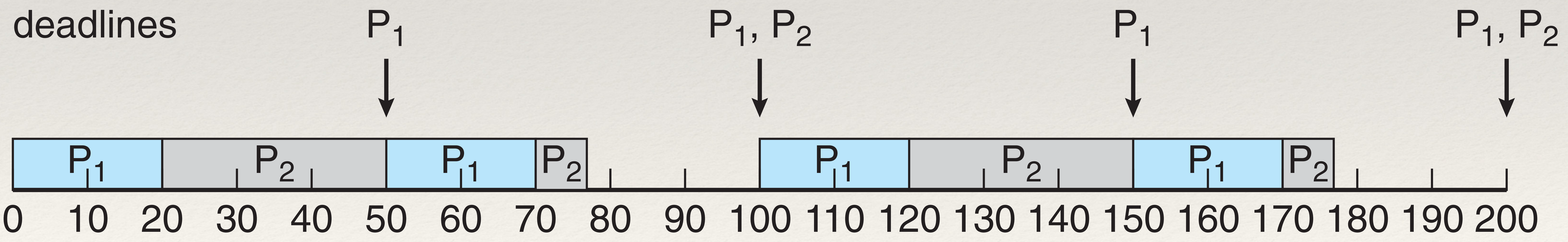
- Process One: $p_1 = 50,\ t_1 = 20\ d_1 = 50$
  - Calculate CPU utilization $20/50 = 0.40$
- Process Two: $p_2 = 100,\ t_2 = 35\ d_2 = 100$
  - Calculate CPU utilization $35/100 = .35$

- If the sum > 1, not possible to schedule them on the same CPU.

Rate Monotonic Example



P2 greater priority than P1



P1 greater priority than P2

Earliest-Deadline-First Scheduling

- Schedule priorities dynamically
- Earlier the deadline, the higher the priority

Proportional Share Scheduling

- Scheduler allocates T shares among all applications
- Each application can receive N shares of time
  - This ensures that each application will have N / T of the total processor time
- Admission-control policy admits requests or denies request based on the number of shares that are available