CS3701 - Operating Systems

Chapter 4 - CPU Scheduling

Department of Computer Science

College of Computer Engineering and Science

Prince Sattam bin Abdulaziz University

AY - 2025/2026

Prepared by Dr. Mahdi Khemakhem, Reviewed by Dr. Essra Aldessouky

Updated on January 22, 2026

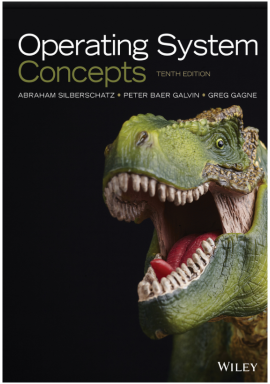
Outline

Textbook Reference

Required Reading

Refer to Chapter 5 of the textbook:

- **Title:** *Operating System Concepts*
- **Authors:** Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne
- **Publisher:** John Wiley & Sons
- **Edition:** 10^{th} Edition (2018)



Outline

CPU Scheduling: Motivation

Why do we need CPU scheduling?

- **Multiprogramming**: Keep CPU busy at all times
- Multiple processes competing for CPU time
- Goal: **Maximize CPU utilization** and **improve system throughput**
- Without scheduling: CPU would be idle during I/O operations

Impact of CPU Scheduling

Consider a system with:

- Process P1 needs 10 ms of CPU time
- Process P2 needs 2 ms of CPU time

Different scheduling orders produce different waiting times:

- **P1 then P2**: Average WT $= (0 + 10)/2 = 5$ ms
- **P2 then P1**: Average WT $= (0 + 2)/2 = 1$ ms

Choice of scheduling algorithm significantly impacts performance!

CPU-I/O Burst Cycle

Process Execution Pattern

Process execution consists of a **cycle of CPU execution** and **I/O wait**:

- **CPU burst**: Process is executing instructions on CPU
- **I/O burst**: Process is waiting for I/O operation to complete
- Process alternates between these two states, it never uses CPU continuously - it pauses for I/O

Typical Pattern

- Starts with **CPU burst**
- Followed by **I/O burst**
- Ends with **CPU burst**

Duration varies:

- **CPU-bound**: Long CPU bursts
- **I/O-bound**: Short CPU bursts, frequent I/O

CPU → I/O → CPU → I/O → CPU

CPU bursts alternate with **I/O bursts**

CPU Scheduler

CPU Scheduler (Short-term Scheduler)

The CPU scheduler **selects** a process from the ready queue and **allocates** the CPU to it.

CPU scheduling decisions occur when a process:

1. Switches from **running** to **waiting** state (e.g., I/O request)
2. Switches from **running** to **ready** state (e.g., interrupt, time slice expired)
3. Switches from **waiting** to **ready** state (e.g., I/O completion)
4. **Terminates**

Scheduling Types

Based on when scheduling occurs:

- **Nonpreemptive (cooperative)**: Scheduling only at 1 and 4
- **Preemptive**: Scheduling at other points 2 and 3

Dispatcher

Dispatcher

The **dispatcher** is the module that gives control of the CPU to the process selected by the short-term scheduler.

Dispatcher functions:

- **Context switching**: Save state of current process, load state of new process
- **Switching to user mode**

Dispatch Latency

Dispatch latency: Time it takes to stop one process and start another

- Should be **minimized** as it's pure overhead
- Typical values: **1-10 microseconds**
- Includes time for: (i) Saving context of current process, (ii) Updating process control block (PCB), (iii) Moving processes between queues, and (iv) Loading context of new process

Lower dispatch latency → Better system responsiveness

CPU Scheduler vs Dispatcher vs Context Switch

Key Differences

- **CPU Scheduler**:
  - **Decision-making** component: *Which* process should run next?
  - Selects process from ready queue based on scheduling algorithm
  - Focus: **Process selection policy**
- **Dispatcher**:
  - **Execution** component: Carries out scheduler's decision
  - Performs the actual *mechanism* of switching CPU control
  - Focus: **Process switching mechanism**
- **Context Switch**:
  - **Core operation** performed by dispatcher
  - Saves state of old process and loads state of new process
  - Focus: **Saving/restoring process state**

Scheduler decides, Dispatcher acts, Context switch is the action

Scheduling Criteria

Different CPU scheduling algorithms have different properties. How do we compare them?

Optimization Criteria

Maximize:

- **CPU Utilization**: Keep CPU as busy as possible (ideally 40%-90%)
- **Throughput**: Number of processes completed per time unit

Minimize:

- **Turnaround Time**: Time from submission to completion
- **Waiting Time**: Total time spent in ready queue
- **Response Time**: Time from submission until first entry into CPU

Scheduling Criteria - Trade-offs

Criteria Trade-offs

- **Interactive systems**: Minimize response time (user experience)
- **Batch systems**: Maximize throughput and utilization
- **Real-time systems**: Meet deadlines (predictability)

No single algorithm is best for all scenarios!

Performance Metrics - Detailed

Key Performance Metrics

For each process, we calculate:

1. **Waiting Time (WT):**
   - Total time spent in **ready queue**
   - Formula: $\text{WT} = \text{TAT} - \text{Burst Time}$
2. **Turnaround Time (TAT):**
   - Time interval from **submission** to **completion**
   - Formula: $\text{TAT} = \text{Completion Time} - \text{Arrival Time}$
   - TAT can be calculated as: $\text{TAT} = \text{WT} + \text{Burst Time}$
3. **Response Time (RT):**
   - Time from submission to **first response**
   - Formula: $\text{RT} = \text{First CPU Time} - \text{Arrival Time}$
   - Important for *interactive systems*
   - *RT and WT are identical for nonpreemptive scheduling!*

Performance Metrics - Example

Example Calculation

Process arrives at $t = 2$, first gets CPU at $t = 6$, runs for 5 ms, completes at $t = 11$:

- **TAT** $= 11 - 2 = 9$ ms
- **WT** $= 9 - 5 = 4$ ms or $(6 - 2) = 4$ ms
- **RT** $= 6 - 2 = 4$ ms

Key Takeaways

Key Takeaways

- **CPU scheduling** is essential for multiprogramming systems to **maximize CPU utilization** and **throughput**.
- Processes alternate between **CPU bursts** and **I/O bursts**, influencing **scheduling decisions**.
- The **CPU scheduler** selects processes from the **ready queue** based on **specific algorithms**.
- Scheduling can be **nonpreemptive** or **preemptive**, affecting **responsiveness** and **complexity**.
- Key performance metrics include **turnaround time**, **waiting time**, and **response time**, which help **evaluate scheduling algorithms**.

Outline

FCFS Scheduling

First-Come, First-Served (FCFS)

Simplest CPU scheduling algorithm. Process that requests CPU **first** is allocated CPU **first**.

Characteristics:

- **Nonpreemptive**: Once CPU allocated, process runs until completion or I/O
- Implemented using **FIFO queue**
- Easy to understand and implement

FCFS - Example 1

FCFS Example with 3 Processes

Given processes arriving at time 0:

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

Gantt Chart (FCFS order: P1, P2, P3):

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

CPU	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P2	P2	P2	P3	P3	P3
Ready Queue	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P3	P3	P3			
	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3	P3						

Calculations:

- **Waiting Time**: P1 $= 0$, P2 $= 24$, P3 $= 27 \rightarrow$ Avg $= (0 + 24 + 27)/3 =$ **17**
- **Response Time**: **FCFS is nonpreemptive, so RT = WT** $\rightarrow$ Avg $=$ **17**
- **Turnaround Time**: P1 $= 24$, P2 $= 27$, P3 $= 30 \rightarrow$ Avg $= (24 + 27 + 30)/3 =$ **27**

FCFS - Example 2 (Convoy Effect)

FCFS with Different Order

Same processes, but arriving in order: P2, P3, P1

Gantt Chart:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

CPU	P2	P2	P2	P3	P3	P3	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P1
Ready Queue	P3	P3	P3	P1	P1	P1																								
	P1	P1	P1																											

Calculations:

- **Waiting Time = Waiting Time**: P1 $= 6$, P2 $= 0$, P3 $= 3 \rightarrow$ Avg $= (6 + 0 + 3)/3 = \mathbf{3}$
- **Turnaround Time**: P1 $= 30$, P2 $= 3$, P3 $= 6 \rightarrow$ Avg $= (30 + 3 + 6)/3 = \mathbf{13}$

Notice: Average waiting time dropped from 17 to 3! This demonstrates the *convoy effect* - **order matters in FCFS**.

FCFS - Pros and Cons

FCFS Pros and Cons

Advantages:

- **Simple and easy to implement**
- **Fair:** Processes served in order of arrival
- **Minimal overhead:** No complex data structures needed

Disadvantages:

- **Convoy effect**: Short processes wait behind long ones
- **Poor average waiting time**: Can be high in certain scenarios
- **Nonpreemptive**: Cannot respond quickly to interactive processes
- **Not suitable** for time-sharing systems
- **Difficult to predict** waiting time for processes

SJF Scheduling

Shortest Job First (SJF)

SJF scheduling selects the process with the *smallest CPU burst time*.

It can be:

- **Nonpreemptive**: Once CPU given, process runs to completion
- **Preemptive** (**SRTF - Shortest Remaining Time First**): If new process arrives with shorter burst than remaining time of current process, **preempt** current process

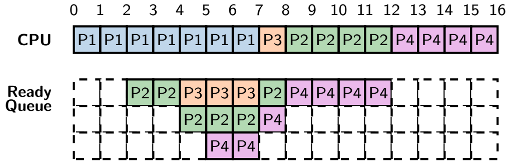
SJF - Example

SJF Nonpreemptive Example

Given the following processes info:

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

Gantt Chart:



Calculations:

- **WT = RT**: P1=0, P2=6, P3=3, P4=7 → Avg $= (0 + 6 + 3 + 7)/4 =$ **4**
- **TAT**: P1=7, P2=10, P3=4, P4=11 → Avg $= (7 + 10 + 4 + 11)/4 =$ **8**

→ **Tie-breaker**: If two processes have same burst time, **use for example FCFS to decide**.

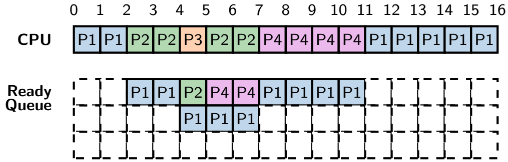
SJF Preemptive (SRTF) - Example

SJF Preemptive (SRTF) - Example

Given the following system snapshot:

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

Gantt Chart:



Calculations:

- **WT**: P1=9, P2=1, P3=0, P4=2 $\rightarrow$ Avg $= (9+1+0+2)/4 =$ **3**
- **RT**: P1=0, P2=0, P3=0, P4=2 $\rightarrow$ Avg $= (0+0+0+2)/4 =$ **0.5**
- **TAT**: P1=16, P2=5, P3=1, P4=6 $\rightarrow$ Avg $= (16+5+1+6)/4 =$ **7**

$\rightarrow$ **WT and TAT are approximately the same as non-preemptive SJF, but RT improved significantly!**

$\rightarrow$ **Notice how preemption improved response time!**

SJF - Pros and Cons

SJF Pros and Cons

Advantages:

- **Optimal** for minimizing average waiting time
- **Reduces convoy effect** compared to FCFS
- **Better turnaround time** for short processes

Disadvantages:

- **Difficult to predict** CPU burst time
- May cause **starvation** of long processes
- **Requires knowledge of future burst times**
- **Preemptive version** adds complexity and overhead

SJF - Predicting CPU Burst Time

The Challenge

Problem: SJF requires knowing CPU burst times in advance, but we cannot know the future!

Solution: Use *exponential averaging* to **predict** next CPU burst based on **past behavior**.

Exponential Averaging Formula

Let τ_{n+1} be the predicted burst time for the $(n+1)$-th CPU burst:

$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$ where:

- t_n = actual length of n^{th} CPU burst
- τ_n = predicted value for n^{th} CPU burst
- α = weighting factor, $0 \leq \alpha \leq 1$

Exponential Averaging - Example

Prediction Example with $\alpha = 0.5$

Given: Initial prediction $\tau_1 = 10$, $\alpha = 0.5$

Burst	Predicted τ_n	Calculation	Actual t_n
1	10.00	(Random guess initial prediction)	6.00
2	8.00	$0.5 \times 6.00 + 0.5 \times 10.00 = 8.00$	4.00
3	6.00	$0.5 \times 4.00 + 0.5 \times 8.00 = 6.00$	6.00
4	6.00	$0.5 \times 6.00 + 0.5 \times 6.00 = 6.00$	4.00
5	5.00	$0.5 \times 4.00 + 0.5 \times 6.00 = 5.00$	4.00
6	4.50	$0.5 \times 4.00 + 0.5 \times 5.00 = 4.50$	4.00
7	4.25	$0.5 \times 4.00 + 0.5 \times 4.50 = 4.25$	4.00
8	4.13	$0.5 \times 4.00 + 0.5 \times 4.25 = 4.13$	8.00
9	6.06	$0.5 \times 8.00 + 0.5 \times 4.13 = 6.06$	8.00
10	7.03	$0.5 \times 8.00 + 0.5 \times 6.06 = 7.03$	8.00

Observations:

- Prediction **adapts** to actual burst times
- With $\alpha = 0.5$, prediction is *average* of last burst and previous prediction
- See visualization in Figure next slide

Exponential Averaging - Visualization

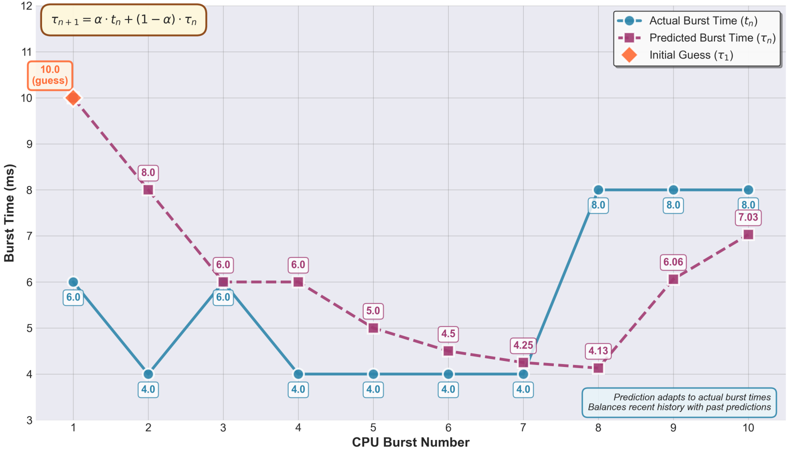


Figure 1: Exponential Averaging: Actual vs Predicted CPU Burst Times ($\alpha = 0.5$ and $\tau_1 = 10$)

Exponential Averaging - α Impact

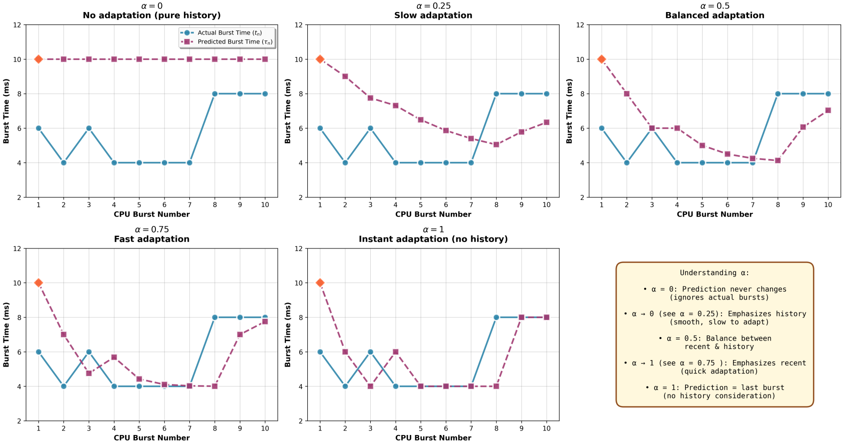


Figure 2: Exponential Averaging: Impact of Different α Values on Prediction

SJF - Burst Prediction Challenges and Modern Approaches

Challenges in Burst Prediction

Issues with predicting CPU burst times:

- **Initial prediction**: No prior data for first burst
- **Dynamic behavior**: Process behavior may change over time
- **I/O-bound vs CPU-bound**: Different process types have different burst patterns
- **Choosing α**: Balancing responsiveness vs stability in predictions
- **Overhead**: Maintaining historical data and calculations

Modern Approaches

Modern techniques for burst time prediction:

- **Machine Learning**: Use historical data to train models predicting burst times (e.g., used by modern OS schedulers) → **more accurate than simple averaging**
- **Statistical Analysis**: Advanced statistical methods to analyze process behavior
- **Hybrid Approaches**: Combine exponential averaging with ML/statistics for improved accuracy

Round Robin Scheduling

Round Robin (RR)

Round Robin is designed for *time-sharing systems*. It is similar to FCFS but with **preemption** to switch between processes.

Key concept:

- Define a **time quantum** (or time slice), typically 10-100 ms
- Ready queue is treated as **circular queue**
- Each process gets CPU for at most one time quantum
- If burst not complete, process goes to end of ready queue

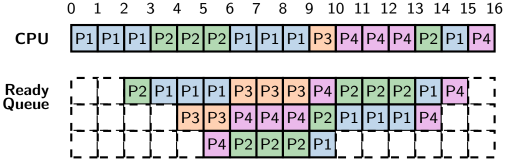
Round Robin - Example 1

Round Robin - Example with Quantum = 3

Given the following system snapshot:

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

Gantt Chart:



Calculations:

- **WT**: P1=8, P2=8, P3=5, P4=7 → Avg $= (8 + 8 + 5 + 7)/4 =$ **7**
- **RT**: P1=0, P2=1, P3=5, P4=5 → Avg $= (0 + 1 + 5 + 5)/4 =$ **2.75**
- **TAT**: P1=15, P2=12, P3=6, P4=11 → Avg $= (15 + 12 + 6 + 11)/4 =$ **11**

→ *To improve the average waiting time, when a process leaves the CPU and returns to the ready queue at the same moment another process arrives, the newly arriving process is placed ahead of the returning process in the queue. (This is known as the "new process advantage" rule.)*

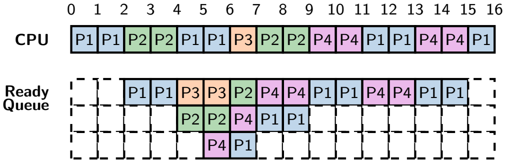
Round Robin - Example 2

Round Robin - Example with Quantum = 2

Given the following system snapshot:

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

Gantt Chart:



Calculations:

- **WT**: P1=9, P2=3, P3=2, P4=6 → Avg $= (9 + 3 + 2 + 6)/4 =$ **5**
- **RT**: P1=0, P2=0, P3=2, P4=4 → Avg $= (0 + 0 + 2 + 4)/4 =$ **1.5**
- **TAT**: P1=16, P2=7, P3=3, P4=10 → Avg $= (16 + 7 + 3 + 10)/4 =$ **9**

→ *Notice that process P3 takes advantage of the "new process advantage" rule to reduce its waiting time!*

Round Robin - Time Quantum Impact

Effect of Time Quantum Size

Time quantum size significantly affects RR performance:

- **Too large**: Degenerates to **FCFS**
  - If **quantum** $\geq$ **max burst time**, **no preemption** occurs
  - **Poor response time**
- **Too small**: Excessive **context switching overhead**
  - **Context switch time** becomes **significant fraction** of quantum
  - **CPU utilization decreases**
  - **Thrashing** can occur (i.e., spending more time context switching than executing processes)
- **Rule of thumb**: **Time quantum** should be **large** compared to **context switch time** $\rightarrow$ Typically: quantum = **10-100 ms**, context switch = **1-10 μs**

Round Robin - Pros and Cons

RR Pros and Cons

Advantages:

- **Fairness**: Each process gets equal CPU time
- **Good response time** for interactive processes
- **Simple to implement** with circular queue

Disadvantages:

- **Higher average waiting time** compared to SJF
- **Context switching overhead** if quantum is too small
- **Not optimal** for minimizing turnaround time
- **Performance sensitive** to time quantum size

Priority Scheduling

Priority Scheduling

Each process is assigned a **priority number**. CPU is allocated to the process with *highest priority*. → **Ties are broken using FCFS, RR, or other methods.**

Priority assignment:

- Can be **internally defined**: time limits, memory requirements, file access
- Can be **externally defined**: importance of process, department, payment
- **Lower number = higher priority** (typical convention)

Priority Scheduling Types

- **Nonpreemptive**: Higher priority process **waits** until current process completes
- **Preemptive**: Arriving process with higher priority **preempts** running process

Priority Scheduling - Nonpreemptive Example

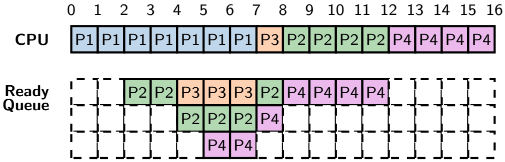
Priority Scheduling (Nonpreemptive)

Given the following system snapshot:

Process	Arrival	Burst	Priority
P1	0	7	3
P2	2	4	2
P3	4	1	1
P4	5	4	4

Lower number = Higher priority

Gantt Chart:



Explanation: P1 runs to completion. At t=7, P3 (pri=1) has highest priority among waiting processes [P2(pri=2), P3(pri=1), P4(pri=4)], so P3 runs next. Then P2 (pri=2) runs, finally P4 (pri=4).

Calculations:

- **WT = RT**: P1=0, P2=6, P3=3, P4=7 $\rightarrow$ Avg $= (0+6+3+7)/4 =$ **4**
- **TAT**: P1=7, P2=10, P3=4, P4=11 $\rightarrow$ Avg $= (7+10+4+11)/4 =$ **8**

Priority Scheduling - Preemptive Example

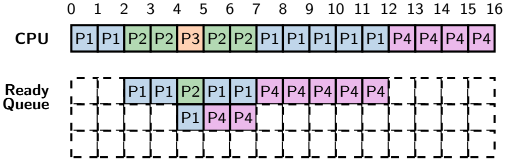
Priority Scheduling (Preemptive)

Given the following system snapshot:

Process	Arrival	Burst	Priority
P1	0	7	3
P2	2	4	2
P3	4	1	1
P4	5	4	4

Lower number = Higher priority

Gantt Chart:



Explanation: At t=2, P2 (pri=2) preempts P1 (pri=3). At t=4, P3 (pri=1) preempts P2. At t=5, P2 resumes (higher priority than P1 and P4). At t=7, P1 resumes.

Calculations:

- **WT**: P1=5, P2=1, P3=0, P4=7 → Avg $= (5+1+0+7)/4 =$ **3.25**
- **RT**: P1=0, P2=0, P3=0, P4=7 → Avg $= (0+0+0+7)/4 =$ **1.75**
- **TAT**: P1=12, P2=5, P3=1, P4=11 → Avg $= (12+5+1+11)/4 =$ **7.25**

Priority Scheduling - Pros and Cons

Priority Scheduling Pros and Cons

Advantages:

- **Flexibility**: Can assign **priorities** based on various **criteria** (importance, deadlines, resource needs)
- **Real-world applicability**: Reflects **actual system requirements** where processes have **different importance**
- **Control**: System or user can **influence scheduling decisions**
- **Efficient for critical tasks**: **High-priority processes** get **immediate attention**

Disadvantages:

- **Starvation**: **Low-priority processes** may **never execute** → solution: *Aging* - see next slide
- **Complexity**: More **complex to implement** and manage compared to simpler algorithms
- **Priority inversion**: **Lower-priority process** holds a **resource** needed by a **higher-priority process**, causing **delays**
- **Subjectivity**: Priority assignment can be **arbitrary** and may not reflect **true process importance**

Aging - Solution to Starvation

Aging Technique

Aging is a technique used to prevent *starvation* in priority scheduling by **gradually increasing the priority** of processes that wait in the system for a long time.

How it works:

- Every fixed time interval, **decrease the priority number** (increase priority) of waiting processes (e.g., reduce priority value by 1 every 5 time units)
- Ensures *all processes get CPU time* (no indefinite blocking)

Key benefits:

- **Prevents starvation**: All processes eventually get executed
- **Fair**: Older processes gain priority over time
- **Automatic**: No manual intervention required
- **Adjustable**: Time interval can be tuned based on system needs

Key Takeaways - Scheduling Algorithms

Key Takeaways

- **FCFS** is simple but suffers from convoy effect; average waiting time depends heavily on arrival order.
  - **Convoy effect**: Short processes wait behind long ones, increasing average waiting time.
- **SJF** minimizes average waiting time but requires predicting CPU burst times using exponential averaging.
  - **Optimality**: SJF provides the lowest average waiting time for a given set of processes.
  - **Exponential averaging**: Uses past CPU bursts to estimate future burst times, balancing recent and historical data.
  - **Nonpreemptive SJF** can lead to longer response times for interactive processes.

Key Takeaways - Scheduling Algorithms (Contd.)

Key Takeaways

- **SRTF** (preemptive SJF) provides even better average waiting time but increases context switching.
  - **Preemption** allows new short processes to run immediately, reducing waiting time.
  - **Higher context switching overhead** due to more frequent preemptions.
  - **Better response time** for interactive processes compared to nonpreemptive SJF.
  - **Starvation risk** for long processes if many short processes arrive.
- **Round Robin** provides good response time for time-sharing systems; quantum size is critical.
  - **Time quantum** should be large compared to context switch time to avoid overhead.
  - **Smaller quantum** improves response time but increases context switching.
  - **Larger quantum** reduces context switching but can lead to poor response time.

Key Takeaways - Scheduling Algorithms (Contd.)

Key Takeaways

- **Priority Scheduling** allows flexible process management but can lead to starvation.
  - **Aging** can be used to prevent starvation by gradually increasing the priority of waiting processes.
  - **Preemptive priority scheduling** allows high-priority processes to run immediately, improving responsiveness.
  - **Nonpreemptive priority scheduling** is simpler but may lead to longer wait times for high-priority processes arriving later.
  - **Priority inversion** can occur when lower-priority processes hold resources needed by higher-priority ones; solutions include priority inheritance.

Outline

Comparing All Algorithms - Dataset

Common Dataset for Comprehensive Comparison

We will compare all scheduling algorithms using:

Process	Arrival	Burst	Priority
P1	0	7	3
P2	2	4	2
P3	4	1	1
P4	5	4	4

Algorithms to compare:

- **FCFS** - First Come First Served
- **SJF** - Shortest Job First (non-preemptive)
- **SRTF** - Shortest Remaining Time First (preemptive SJF)
- **RR (q=2)** - Round Robin with **quantum = 2**
- **RR (q=3)** - Round Robin with **quantum = 3**
- **Priority (NP)** - Priority Scheduling (non-preemptive)
- **Priority (P)** - Priority Scheduling (preemptive)

Results - Complete Comparison Table

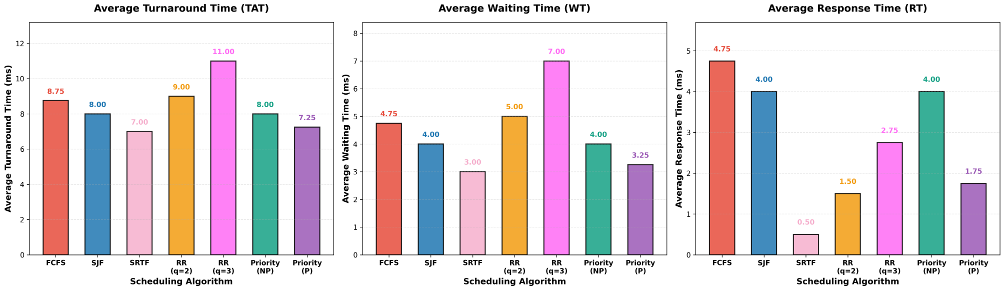
Algorithm	Turnaround Time (TAT)					Waiting Time (WT)					Response Time (RT)				
	P1	P2	P3	P4	Avg	P1	P2	P3	P4	Avg	P1	P2	P3	P4	Avg
FCFS	7	9	8	11	**8.75**	0	5	7	7	**4.75**	0	5	7	7	**4.75**
SJF	7	10	4	11	**8.00**	0	6	3	7	**4.00**	0	6	3	7	**4.00**
SRTF	16	5	1	6	**7.00**	9	1	0	2	**3.00**	0	0	0	2	**0.50**
RR (q=2)	16	7	3	10	**9.00**	9	3	2	6	**5.00**	0	0	2	4	**1.50**
RR (q=3)	15	12	6	11	**11.00**	8	8	5	7	**7.00**	0	1	5	5	**2.75**
Priority (NP)	7	10	4	11	**8.00**	0	6	3	7	**4.00**	0	6	3	7	**4.00**
Priority (P)	12	5	1	11	**7.25**	5	1	0	7	**3.25**	0	0	0	7	**1.75**

Key Observations:

- **Best performers**: SRTF (TAT=7.0, WT=3.0, RT=0.5), Priority (P) (TAT=7.25, WT=3.25, RT=1.75)
- **Worst performers**: RR (q=3) for TAT/WT (11.0, 7.0), FCFS for RT (4.75)
- **Quantum impact**: RR (q=2) outperforms RR (q=3) by 18% in TAT, 29% in WT, 45% in RT
- **Identical behavior**: SJF $\equiv$ Priority (NP) due to burst times matching priority order
- **Preemption advantage**: Preemptive algorithms consistently achieve better RT

$\Rightarrow$ **Performance depends on workload characteristics and system objectives.**

Visual Comparison - Performance Metrics



Performance Analysis:

- **TAT/WT ranking**: SRTF (best) < Priority (P) < SJF < FCFS < RR (q=2) < RR (q=3) (worst)
- **RT ranking**: SRTF (0.5) < RR (q=2) (1.5) < Priority (P) (1.75) < RR (q=3) (2.75) < SJF/FCFS (4+)
- **Trade-offs**: Preemptive algorithms excel in RT but may increase overhead; RR provides fairness at efficiency cost
- **Quantum selection critical**: Smaller quantum (q=2) significantly improves all metrics vs larger (q=3)

Algorithm Performance Analysis

Algorithm	Strengths	Weaknesses
FCFS	**Simple**, no starvation	**Convoy effect**, poor average performance
SJF	**Good TAT/WT**, no preemption	**Starvation possible**, poor RT, requires burst prediction
SRTF	**Best overall**, excellent RT	**High overhead**, requires burst prediction
RR (q=2)	**Better RT** (1.5 ms), fair scheduling	**Higher TAT/WT** than optimal, more context switches
RR (q=3)	**Fair scheduling**, no starvation	**Worst TAT/WT** (11.0, 7.0), poor efficiency
Priority (NP)	**Flexible**, good for critical tasks	**Same as SJF here**, starvation, poor RT
Priority (P)	**Second best overall**, very good RT	**Complex**, starvation without aging

When to use each algorithm:

- **FCFS**: Simple embedded devices, print queues (ease of implementation), e.g., IoT devices, printers
- **SJF/SRTF**: Batch processing, background jobs (optimize TAT/WT), e.g., scientific computing, data analysis
- **RR**: Time-sharing OS (Unix, Windows), web servers (fair CPU sharing)
- **Priority**: Real-time systems, mission-critical tasks (guarantee deadlines), e.g., medical devices, telecommunications

Key Takeaways - Algorithm Comparison

Key Takeaways

- **No one-size-fits-all**: Scheduling algorithm choice depends on workload characteristics and system goals (TAT, WT, RT).
- **Preemptive algorithms** (SRTF, Priority P) generally outperform non-preemptive ones in responsiveness.
- **Round Robin** offers fairness but requires careful quantum selection to balance responsiveness and efficiency.
- **Priority scheduling** provides flexibility but needs mechanisms (like aging) to prevent starvation.
- **Understanding trade-offs** is crucial for selecting the right algorithm for specific applications and environments.
- **Performance metrics** (TAT, WT, RT) should guide algorithm selection based on application needs.

Outline

Exercise 1: Exponential Averaging

CPU Burst Time Prediction

Given: Initial prediction $\tau_1 = 10$ ms, $\alpha = 0.5$

Actual CPU burst times: $t_1 = 6, t_2 = 8, t_3 = 12, t_4 = 4$

Formula: $\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$

Tasks:

1. Calculate predicted burst times $\tau_2, \tau_3, \tau_4, \tau_5$
2. Compare predicted vs actual values in a table
3. What would happen if $\alpha = 0$? What if $\alpha = 1$?

Solution: Exercise 1

Step-by-step calculations:

$$\tau_2 = 0.5 \cdot 6 + 0.5 \cdot 10 = 3 + 5 = 8$$
$$\tau_3 = 0.5 \cdot 8 + 0.5 \cdot 8 = 4 + 4 = 8$$
$$\tau_4 = 0.5 \cdot 12 + 0.5 \cdot 8 = 6 + 4 = 10$$
$$\tau_5 = 0.5 \cdot 4 + 0.5 \cdot 10 = 2 + 5 = 7$$

n	Actual t_n	Predicted τ_n	Error $\lvert t_n - \tau_n \rvert$
1	6	10	4
2	8	8	0
3	12	8	4
4	4	10	6

Alpha analysis:

- $\alpha = 0$: Prediction never changes ($\tau_{n+1} = \tau_n$) - ignores recent history
- $\alpha = 1$: Prediction = last burst ($\tau_{n+1} = t_n$) - ignores past history
- $\alpha = 0.5$: **Balanced** - weights recent and past equally

Exercise 2: Scheduling Algorithm Comparison

Compare FCFS, SJF, and RR

Given processes:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	2

Tasks:

1. Calculate average TAT, WT, and RT for **FCFS**
2. Calculate average TAT, WT, and RT for **SJF (non-preemptive)**
3. Calculate average TAT, WT, and RT for **RR with quantum = 3**
4. Which algorithm provides the best response time? Why?
5. Which algorithm provides the best waiting time (Turnaround Time)? Why?

Solution: Exercise 2 - FCFS

FCFS Gantt Chart:

0 8 12 14

P1	P1	P1	P1	P1	P1	P1	P1	P2	P2	P2	P2	P3	P3

Calculations:

Process	CT	TAT	WT	RT
P1	8	8-0=8	8-8=0	0-0=0
P2	12	12-1=11	11-4=7	8-1=7
P3	14	14-2=12	12-2=10	12-2=10
Average		**10.33**	**5.67**	**5.67**

Solution: Exercise 2 - SJF

SJF (Non-Preemptive) Gantt Chart:

0								8		10			14
P1	P1	P1	P1	P1	P1	P1	P1	P3	P3	P2	P2	P2	P2

Calculations:

Process	CT	TAT	WT	RT
P1	8	8-0=8	8-8=0	0-0=0
P2	14	14-1=13	13-4=9	10-1=9
P3	10	10-2=8	8-2=6	8-2=6
Average		**9.67**	**5.00**	**5.00**

Note: P3 (burst=2) executes before P2 (burst=4) after P1 completes

Solution: Exercise 2 - RR (q=3)

RR with Quantum = 3 Gantt Chart:

0 3 6 8 11 12 14

P1	P1	P1	P2	P2	P2	P3	P3	P1	P1	P1	P2	P1	P1

Calculations:

Process	CT	TAT	WT	RT
P1	14	14-0=14	14-8=6	0-0=0
P2	12	12-1=11	11-4=7	3-1=2
P3	8	8-2=6	6-2=4	6-2=4
Average		**10.33**	**5.67**	**2.00**

Conclusion:

- **Best RT**: RR (q=3) with 2.00 ms due to frequent context switching
- **Best WT/TAT**: SJF with 5.00 ms WT and 9.67 ms TAT due to optimal scheduling order

Exercise 3: Round Robin Quantum Analysis

Theoretical Analysis of Time Quantum

Context: Consider n processes with burst times $b_1, b_2, \ldots, b_n$ scheduled on a single CPU using Round Robin with quantum Q_1 (algorithm A1) and quantum Q_2 (algorithm A2).

Notation:

- R_1, R_2: Average response times using A1 and A2
- W_1, W_2: Average waiting times using A1 and A2

Questions:

1. Using $b_1, b_2, \ldots, b_n$, give the expression for the **total burst time** of all n processes.
2. For $Q_1 = 3$ and $Q_2 = 6$, compare R_1 and R_2. Which one is larger? Justify your answer.
3. Using $n, b_1, b_2, \ldots, b_n, W_1$, give the expression for the **average turnaround time** when using A1.

Solution: Exercise 3

Question 1: Total Burst Time

The total burst time of all n processes is:

$$B_{total} = \sum_{i=1}^{n} b_i = b_1 + b_2 + \cdots + b_n$$

Question 2: Response Time Inequality

For $Q_1 = 3$ and $Q_2 = 6$ (smaller quantum vs larger quantum):

$$R_1 \leq R_2$$

Explanation: Smaller quantum ($Q_1 = 3$) provides faster response time because processes get the CPU sooner and more frequently. With $Q_2 = 6$, processes wait longer for their first CPU access.

Question 3: Average Turnaround Time

Using the relationship: $TAT = WT + BT$
Average turnaround time using A1:

$$TAT_{avg} = W_1 + \frac{1}{n}\sum_{i=1}^{n} b_i = W_1 + \frac{B_{total}}{n}$$

where $\frac{1}{n}\sum_{i=1}^{n} b_i$ is the average burst time.

End of Chapter 4