SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution

Coimbatore-35

www.snsgroups.com

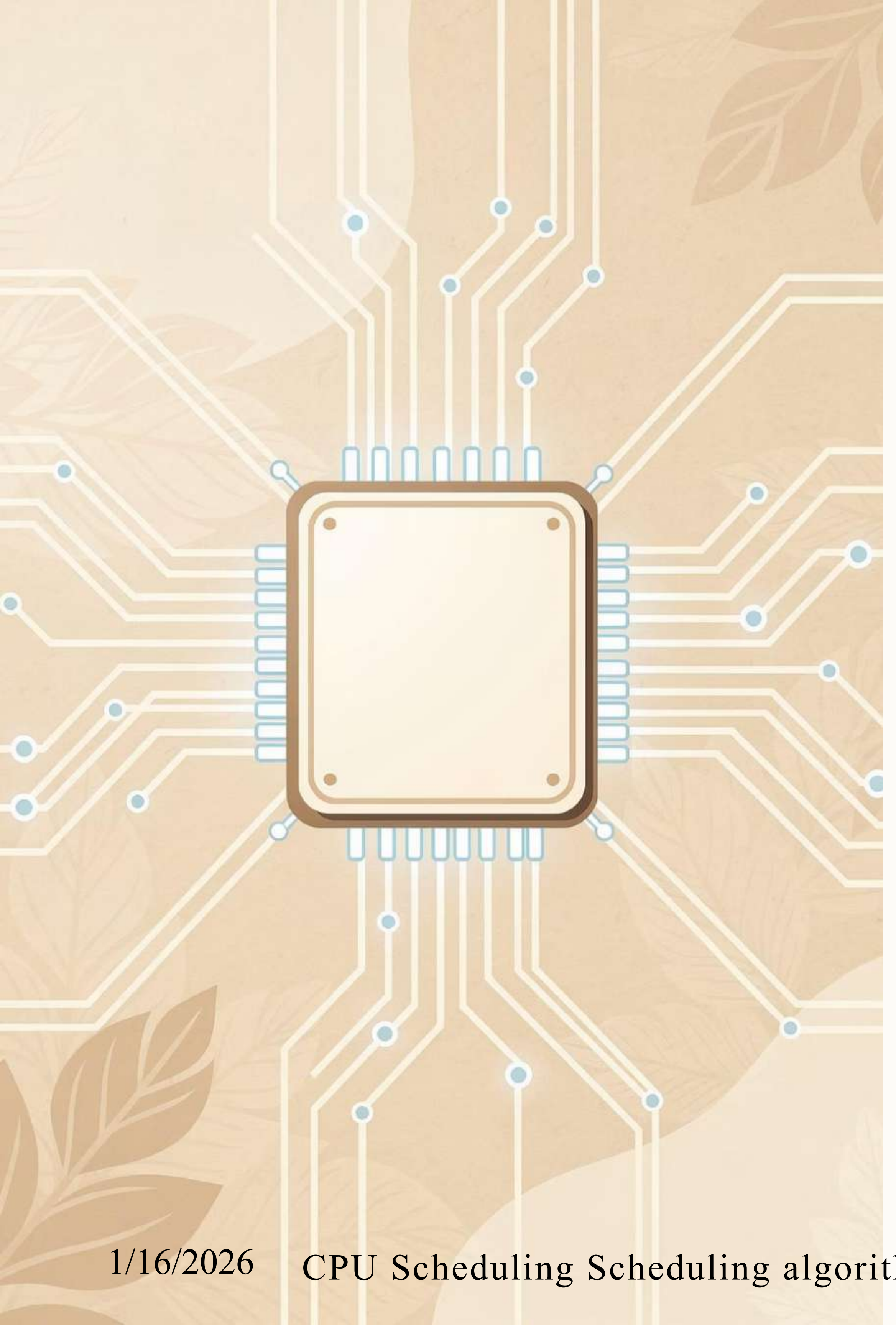
Department of Computer Science and Engineering

23CST206–OPERATING SYSTEMS AND VIRTUALIZATION

B.E- CSE /IV SEMESTER

UNIT – II PROCESS MANAGEMENT

Topic 3:CPU Scheduling - Scheduling criteria - Scheduling algorithms

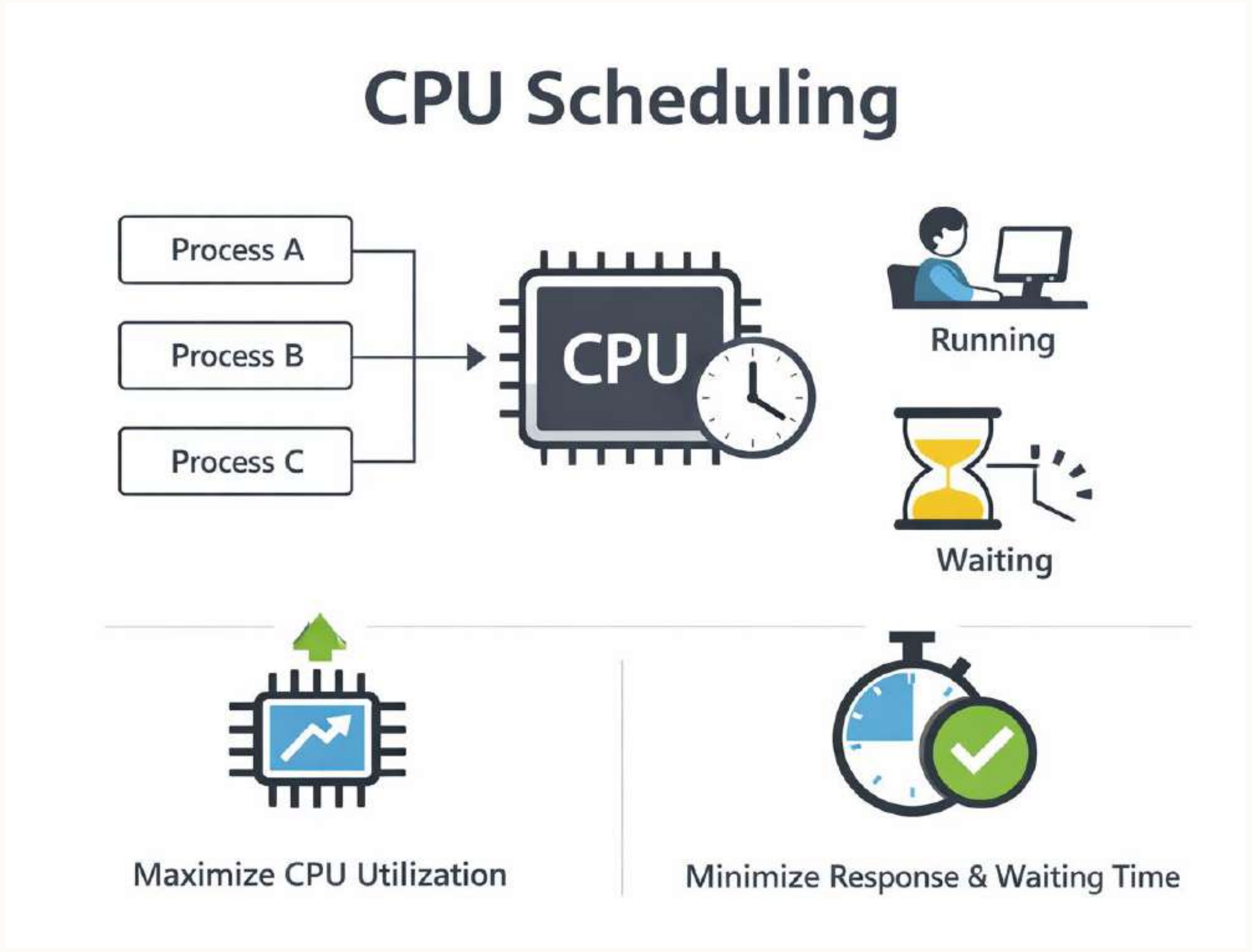


CPU Scheduling

OPERATING SYSTEMS CORE CONCEPTS

Understanding how operating systems manage process execution through scheduling criteria and algorithms

What is CPU Scheduling?



- CPU scheduling is a process used by the operating system to decide which task or process gets to use the CPU at a particular time.

- This is important because a CPU can only handle one task at a time, but there are usually many tasks that need to be processed.

Why CPU Scheduling is Needed

Single Resource

Only one process can use the CPU at a time

Multiple Demands

Multiple processes compete for CPU access

System Performance

To improve overall system efficiency

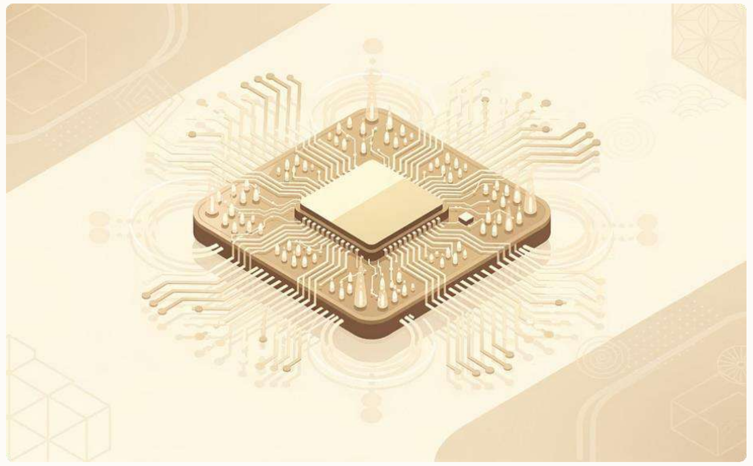
Quick Response

To provide fast response to users

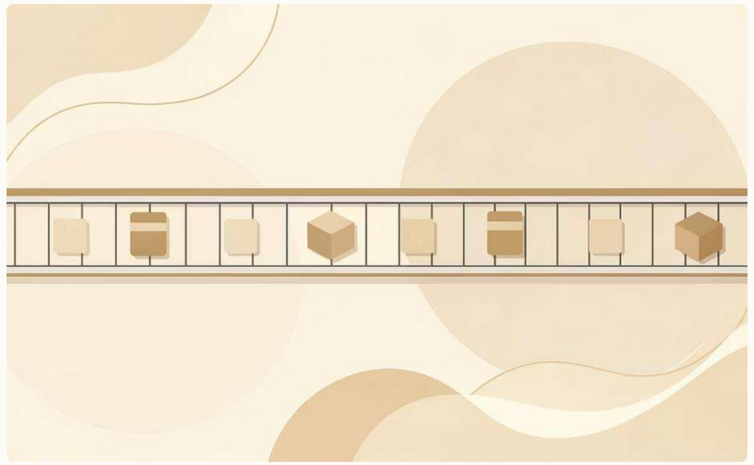
Resource Optimization

To avoid starvation and CPU idle time

CPU Scheduling Criteria



CPU Utilization
Keep the CPU as busy as possible



Throughput
Number of processes completed per unit time

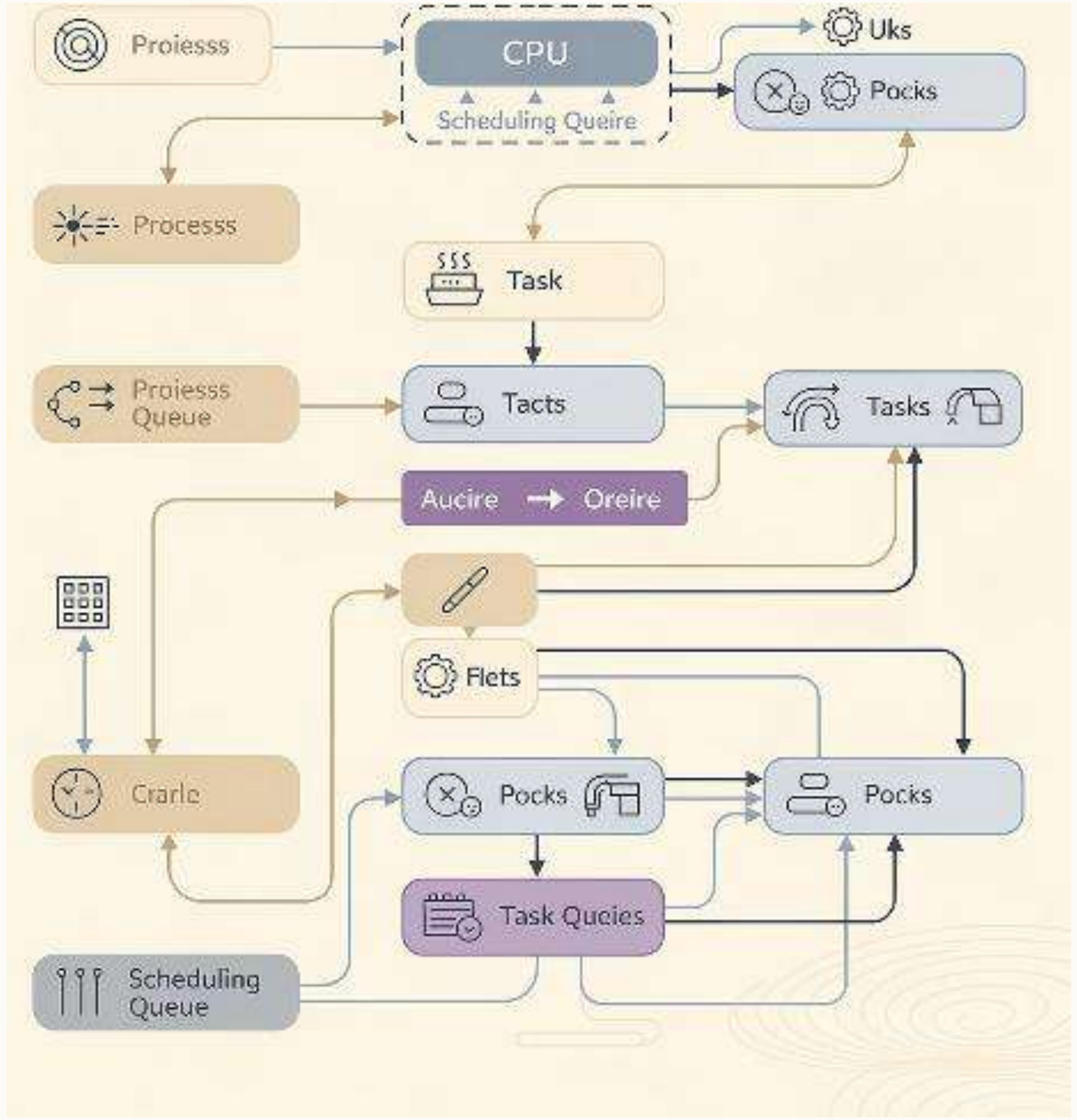
Turnaround Time
Total time taken from process submission to completion

Waiting Time

Total time a process spends in the ready queue

Response Time
Time from process submission to first response (important in interactive systems)

Fairness
Ensure all processes get a chance to run

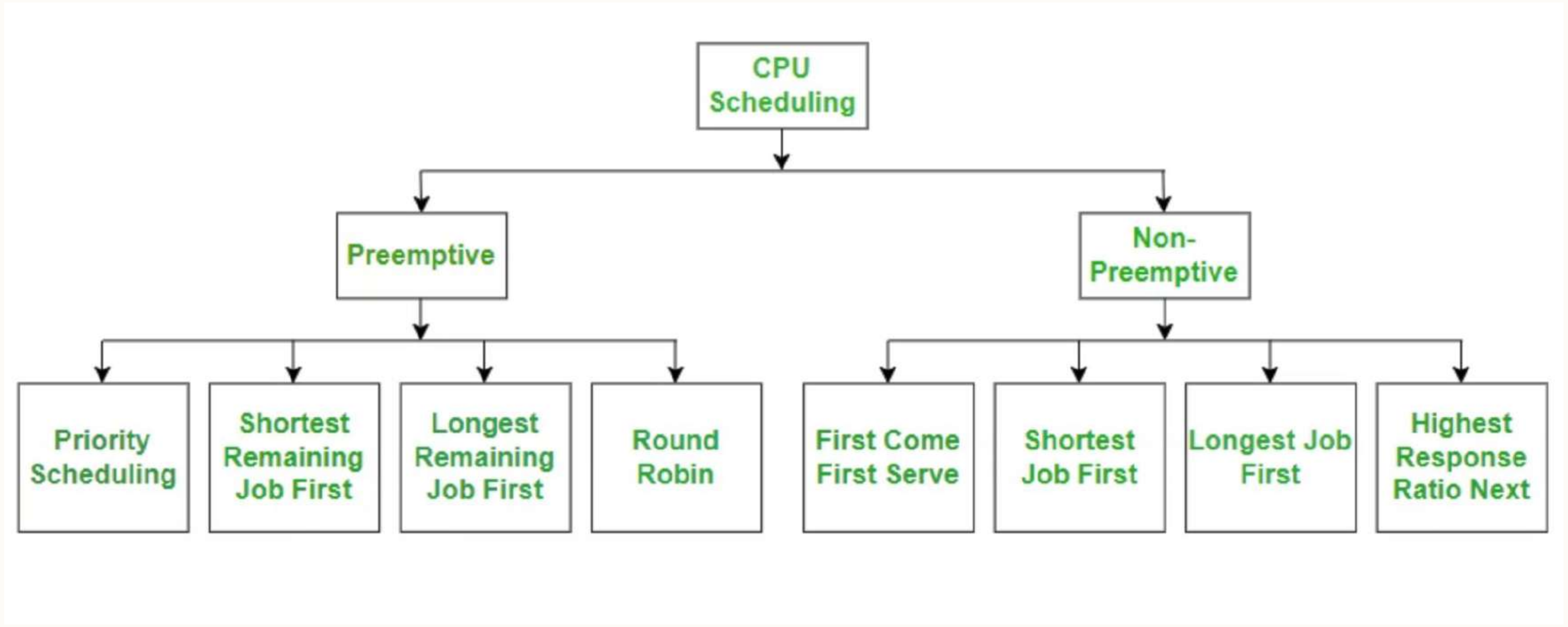


The Need for CPU Scheduling Algorithms

- CPU scheduling is the process of deciding which process will own the CPU to use while another process is suspended.
- The main function of CPU scheduling is to ensure that whenever the CPU remains idle, the OS has at least selected one of the processes available in the ready-to-use line.

Types of CPU Scheduling Methods

There are two fundamental approaches to CPU scheduling, each with distinct characteristics and use cases.



CPU Scheduling Algorithms Overview

Modern operating systems employ various scheduling algorithms, each designed to optimize different aspects of system performance. Let's explore the major algorithms used in practice.

FCFS

Processes served in arrival order.

SJF

Select shortest job next to reduce wait.

SRTF

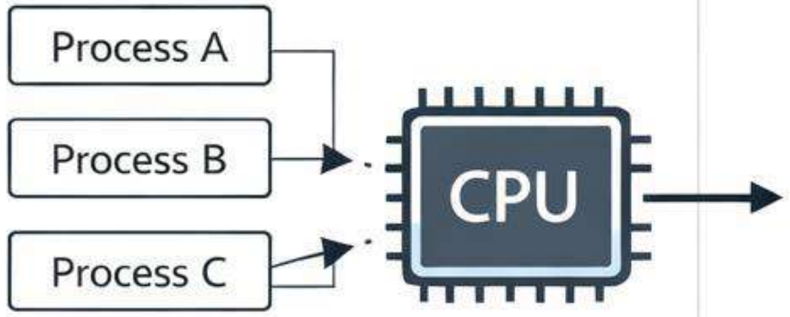
Preemptive SJF, switches on shorter arrivals.

Round Robin

Time-sliced fairness across ready processes.

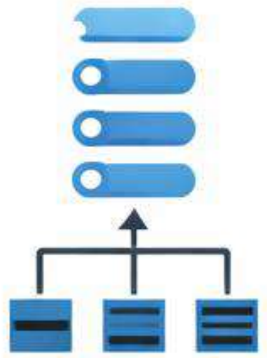
FCFS – First Come, First Serve

How It Works



Scheduled in order of arrival

Features



- Non-preemptive
- Uses FIFO queue

Advantages

- Simple to implement
- Fair order of arrival

Disadvantages

- Convoy effect
- Not suitable for real-time

Example

Printing documents in order of arrival

ALGORITHM 2

SJF – Shortest Job First

How it works

The process with the shortest burst time is scheduled first. Non-preemptive SJF waits until the running process finishes.

Features

Non-preemptive scheduling. Requires knowledge of burst time (can be estimated).

Advantages

Minimizes average waiting time and turnaround time. Efficient for batch systems.

Disadvantages

- Long processes may starve if short jobs keep arriving
- Difficult to estimate CPU burst times accurately

Example: Running quick maintenance tasks before longer system updates.

SRTF – Shortest Remaining Time First

Preemptive version of SJF. If a new process arrives with shorter remaining CPU time than the current running process, CPU is reassigned.

Features

- **Preemptive:** ensures shortest jobs finish first
- Efficient in minimizing average waiting time

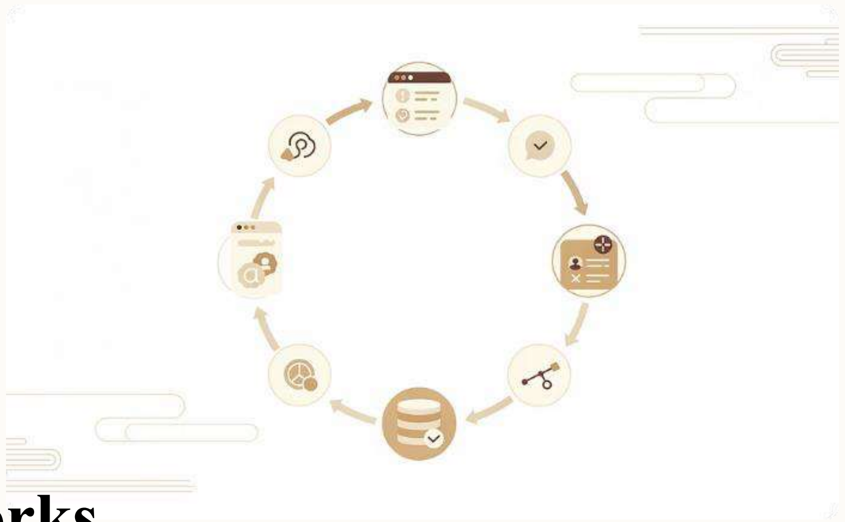
Advantages

- Reduces average waiting time more than SJF
- Provides better response for short processes

Disadvantages

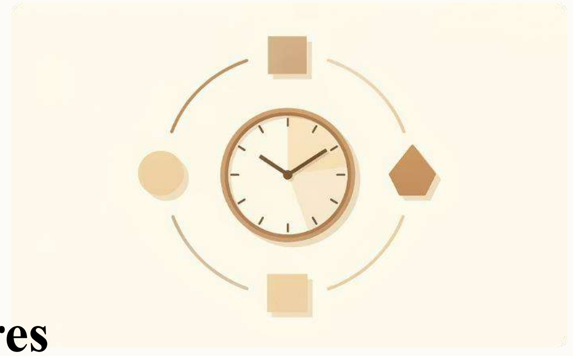
- High context switching overhead
- Risk of starvation for long processes

Round Robin (RR)



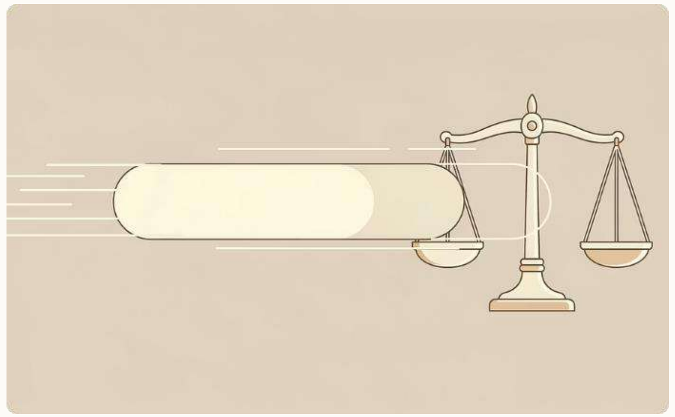
How it works

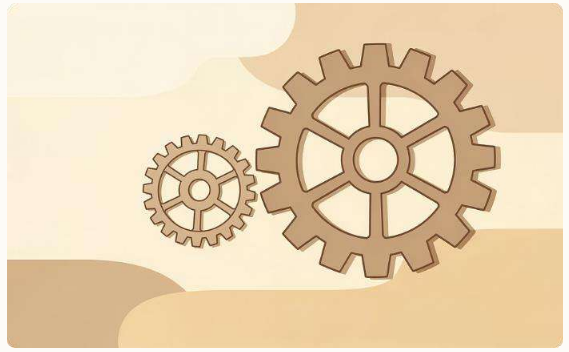
Each process gets CPU for a fixed time slice (quantum). If the process does not finish in the time quantum, it goes to the back of the ready queue.



Features

- Preemptive, time-sharing scheduling
- Each process gets fair share of CPU time





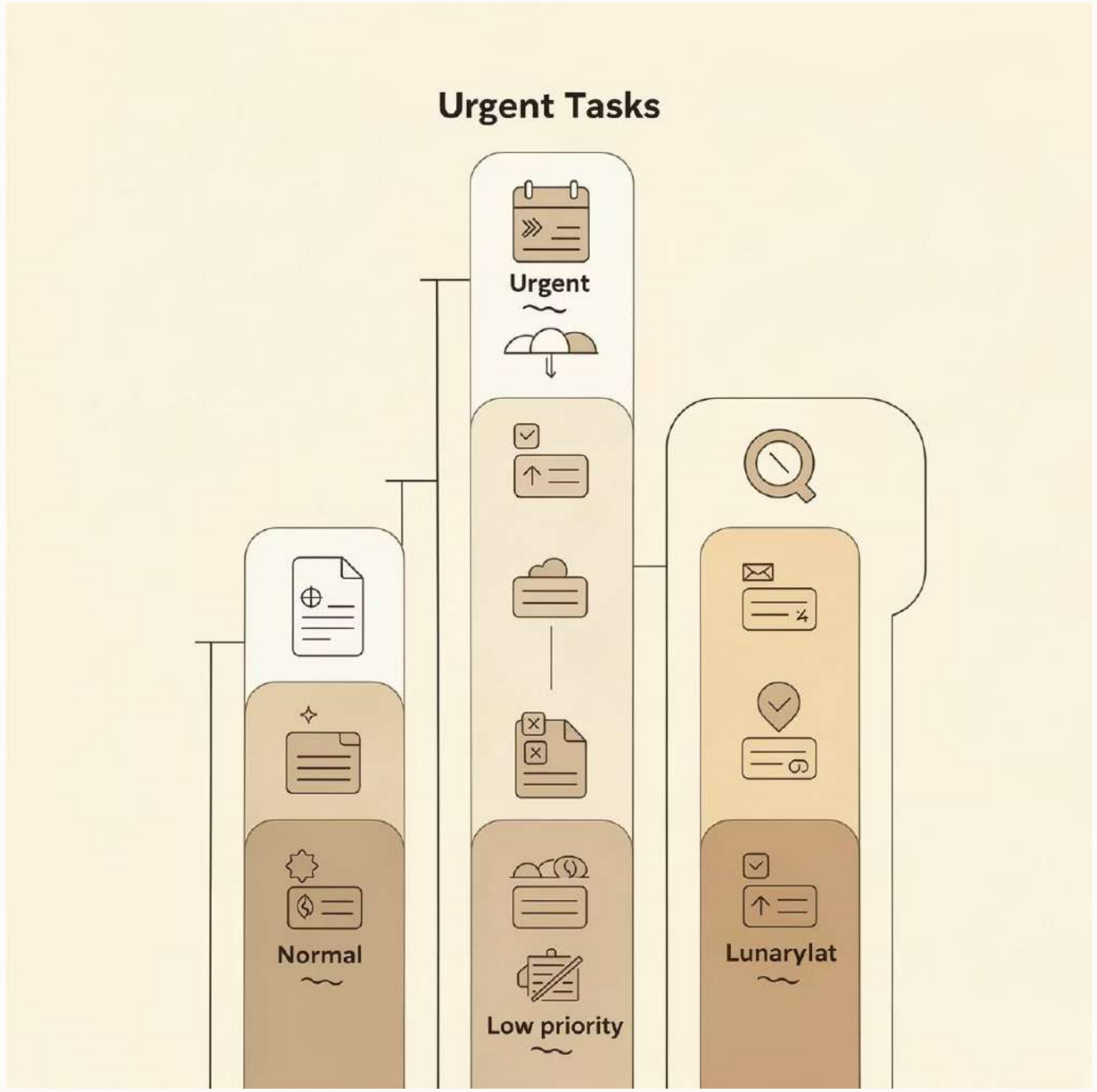
Disadvantages

Performance depends on time quantum size:

- **Too small** → frequent context switches (overhead)
- **Too large** → behaves like FCFS

Advantages

- Fair and responsive, suitable for interactive systems



Priority Scheduling

- Each process is assigned a priority.
- CPU is allocated to the process with highest priority first.
- Preemptive version can interrupt running process if a higher priority process arrives.

Features

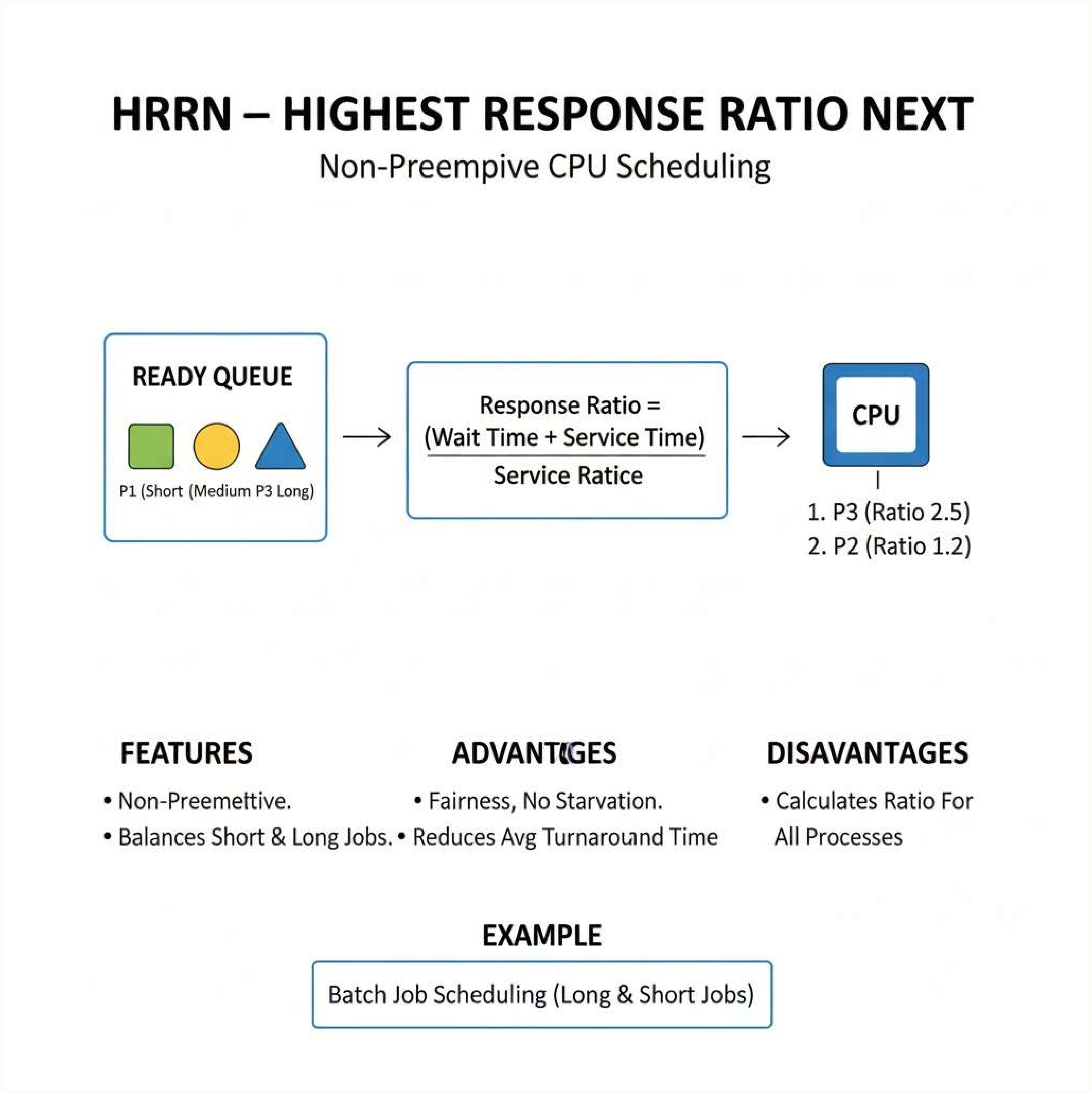
Can be preemptive or non-preemptive. Priority can be static or dynamic.

Advantages

Critical or important tasks get CPU quickly. Useful for real-time systems.

Disadvantages

Starvation of low-priority processes (solution: aging).



HRRN – HIGHEST RESPONSE RATIO NEXT

Non-Preempive CPU Scheduling

FEATURES

- Non-Preemeltive.
- Balances Short & Long Jobs.

ADVANTCGES

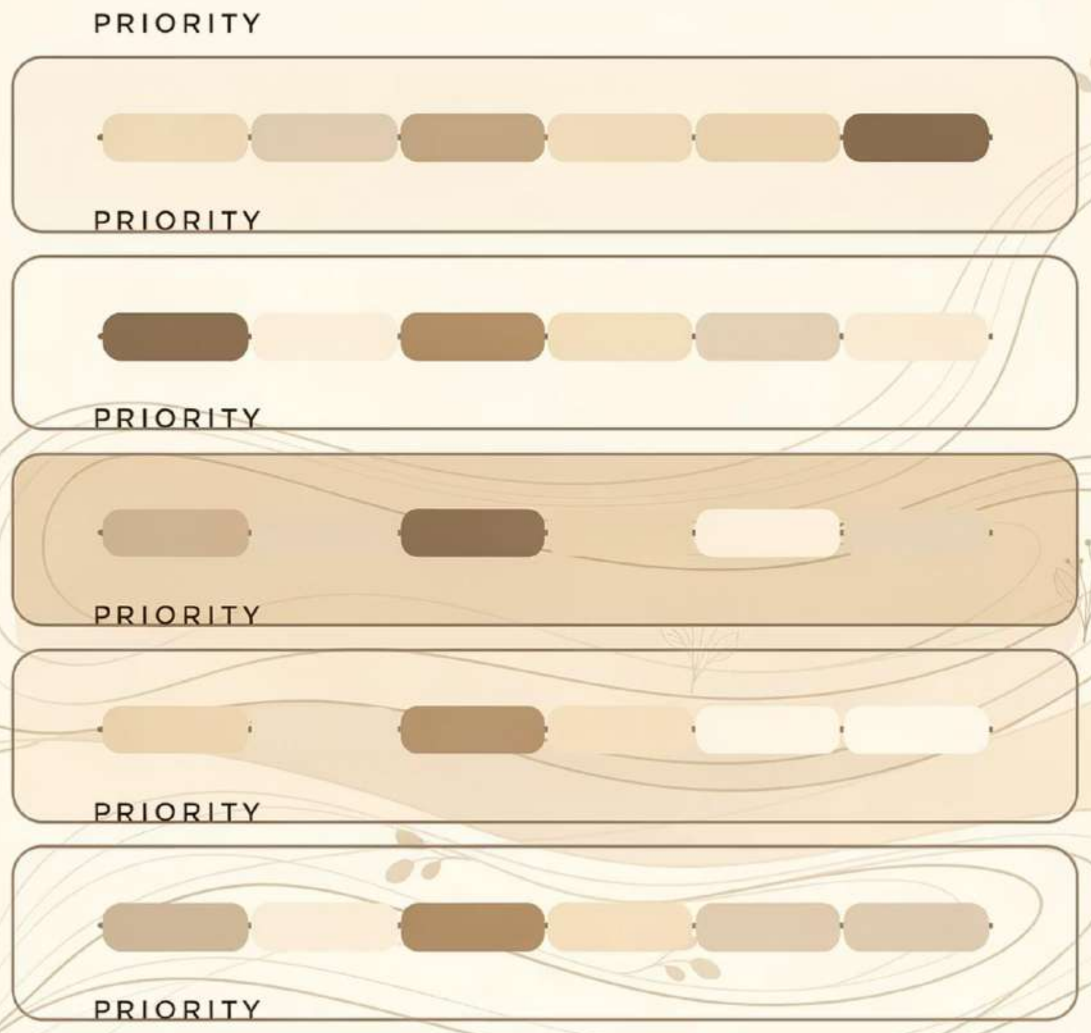
- Fairness, No Starvation.
- Reduces Avg Turnaround Time

DISAVANTAGES

- Calculates Ratio For All Processes

EXAMPLE

Batch Job Scheduling (Long & Short Jobs)



Multiple Queue Scheduling

How it works

- Ready queue divided into multiple queues based on priority, process type, or resource requirements.
- Each queue can have different scheduling algorithm. CPU allocated based on queue priority.

Features

Good for differentiating interactive and batch processes.

Advantages

- Organized scheduling, improves efficiency
- Can treat important tasks separately

Disadvantages

Fixed priorities may lead to starvation of lower-priority queues.

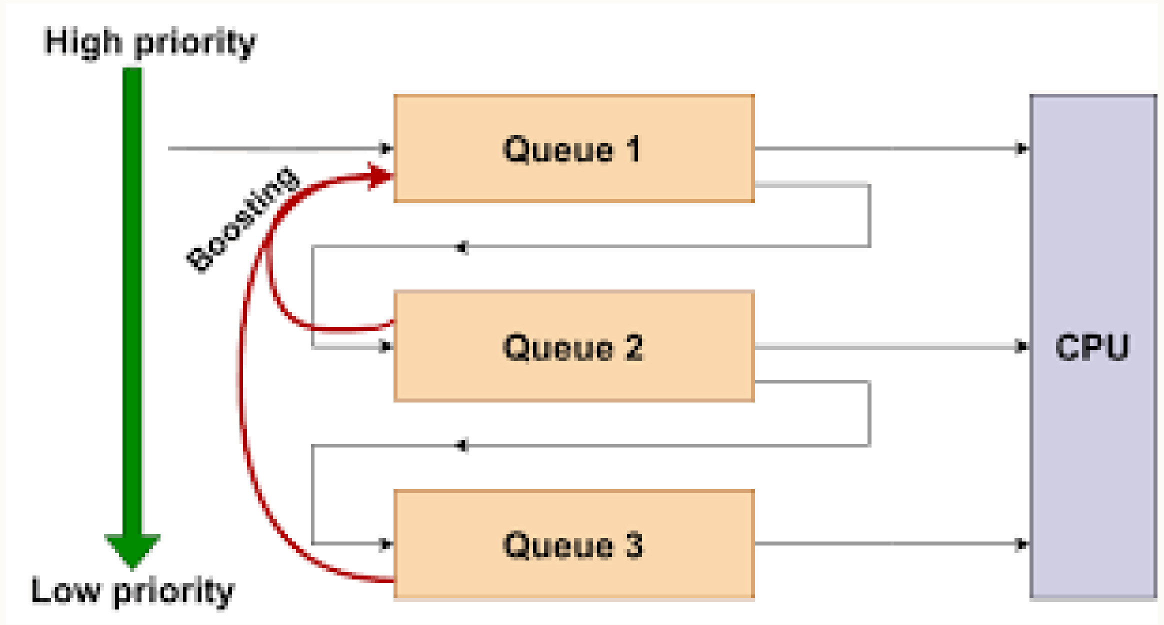
Multilevel Feedback Queue Scheduling

Similar to multiple queue scheduling but processes can move between queues based on their CPU burst behavior. Processes using CPU heavily are moved to lower-priority queues. I/O-bound or interactive processes stay in higher-priority queues.

Features
Adaptive, dynamic scheduling

Benefits
Prevents starvation and favors short, interactive processes

Implementation
Complex to implement and maintain

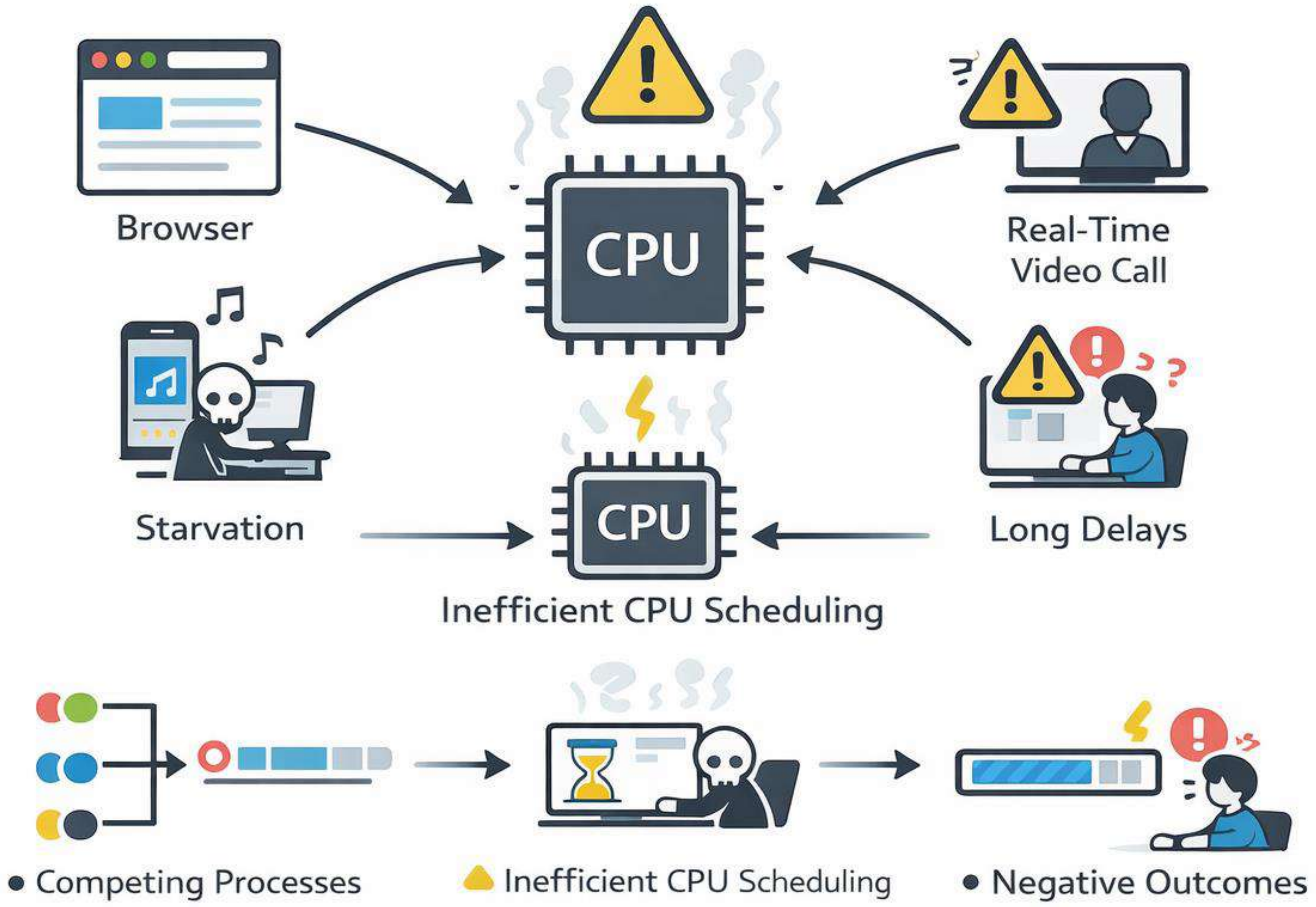


Advantages

- Flexible, fair, prevents starvation
- Suitable for modern time-sharing systems

Example: Modern OS like Windows and Linux use feedback scheduling for process management.

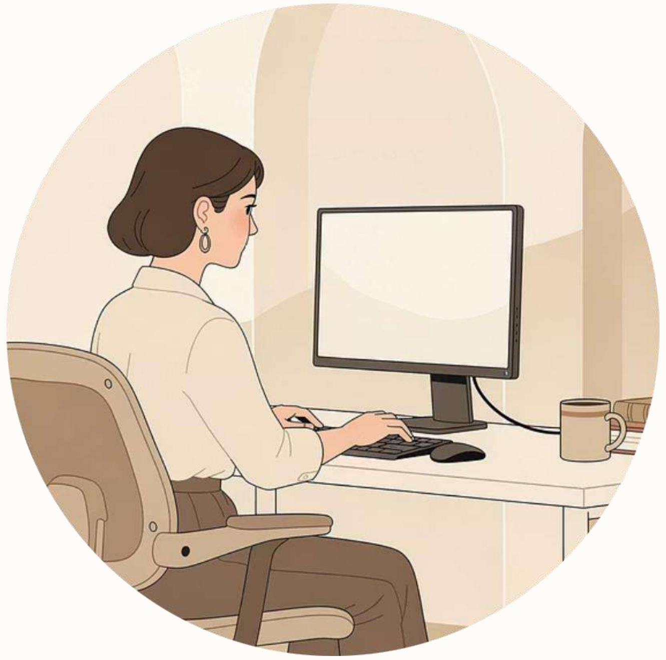
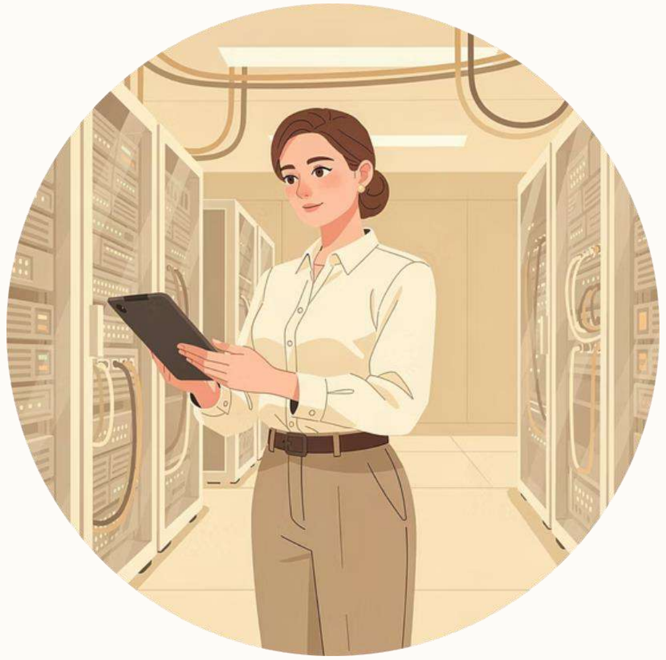
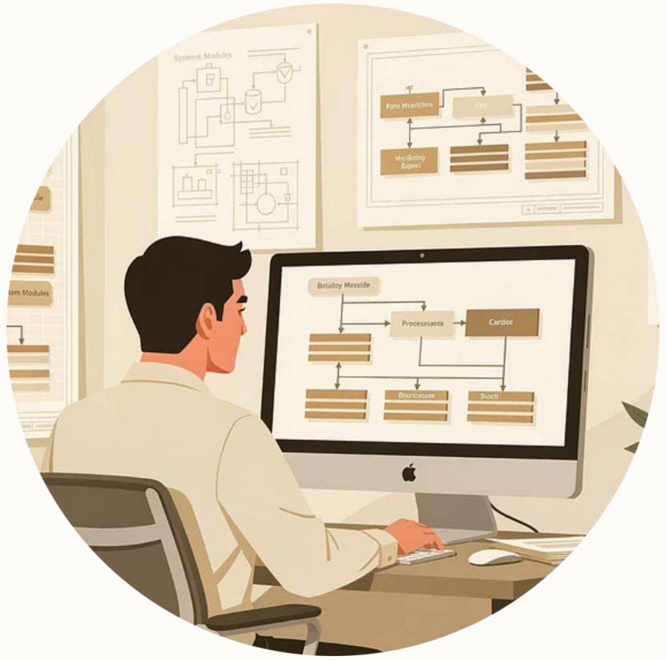
Problem Statement



DESIGN THINKING

Empathize: Understanding Stakeholders

Effective CPU scheduling must address the needs of diverse stakeholders, each with unique requirements and expectations.

End Users

- Want fast, responsive applications
- Do not want freezes, delays, or crashes

Programmers / Developers

- Want predictable CPU access for their programs
- Require fair execution of processes for testing and performance

System Administrators

- Need maximum CPU utilization
- Want to prevent starvation and ensure priority tasks run on time

OS Designers

- Must balance performance, fairness, and responsiveness
- Must handle both batch and interactive processes efficiently

DESIGN THINKING

Define: Core Problem Identification

The operating system must make critical decisions to ensure optimal process management and system performance.

Process Selection

Which process gets the CPU next

Time Allocation

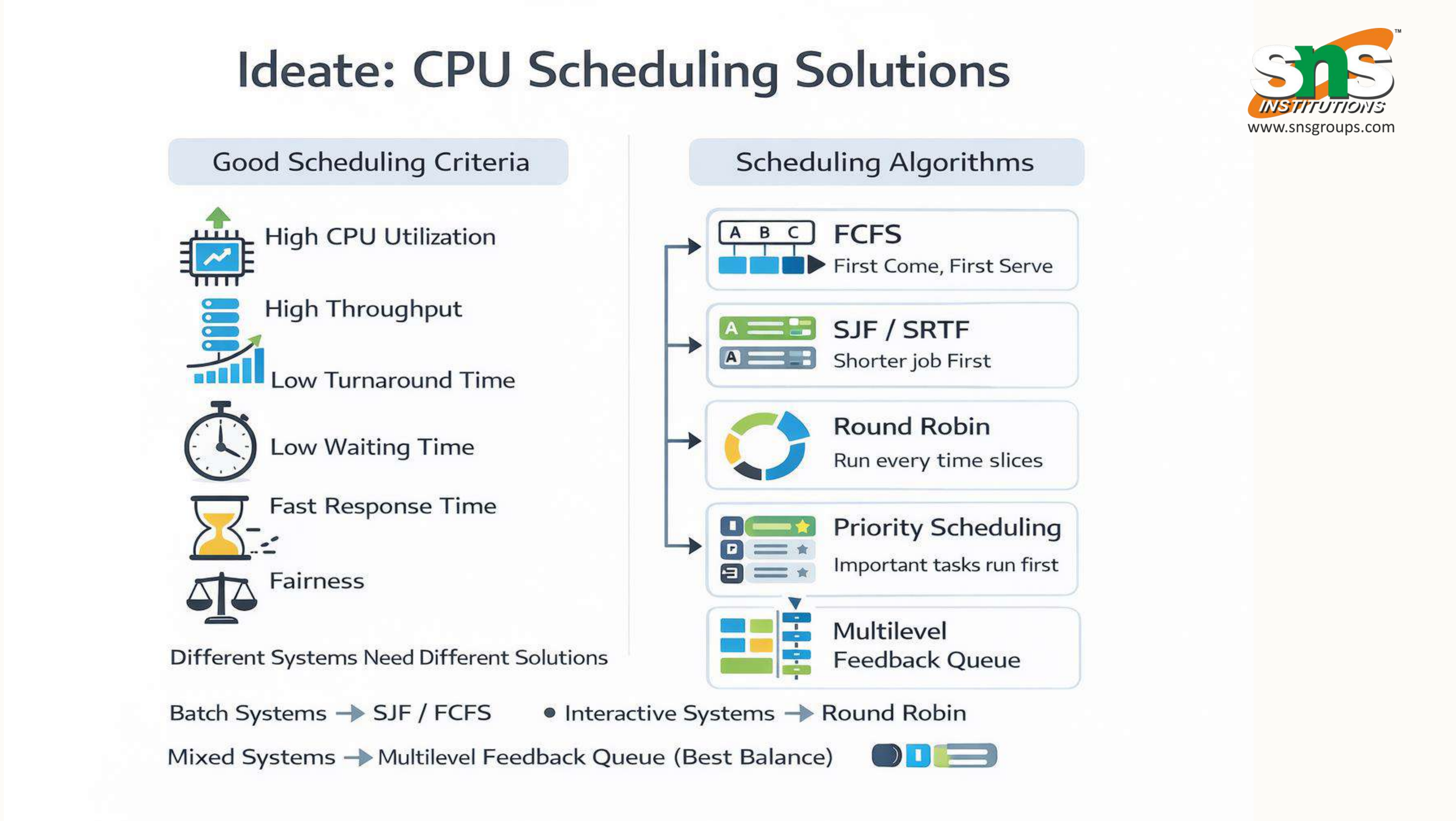
How long it runs (time slice or quantum)

Fairness Assurance

How to prevent starvation and ensure fairness

Efficiency Measurement

How to measure efficiency using scheduling criteria



Ideate: CPU Scheduling Solutions

www.snsgroups.com

Good Scheduling Criteria

- High CPU Utilization
- High Throughput
- Low Turnaround Time
- Low Waiting Time
- Fast Response Time
- Fairness

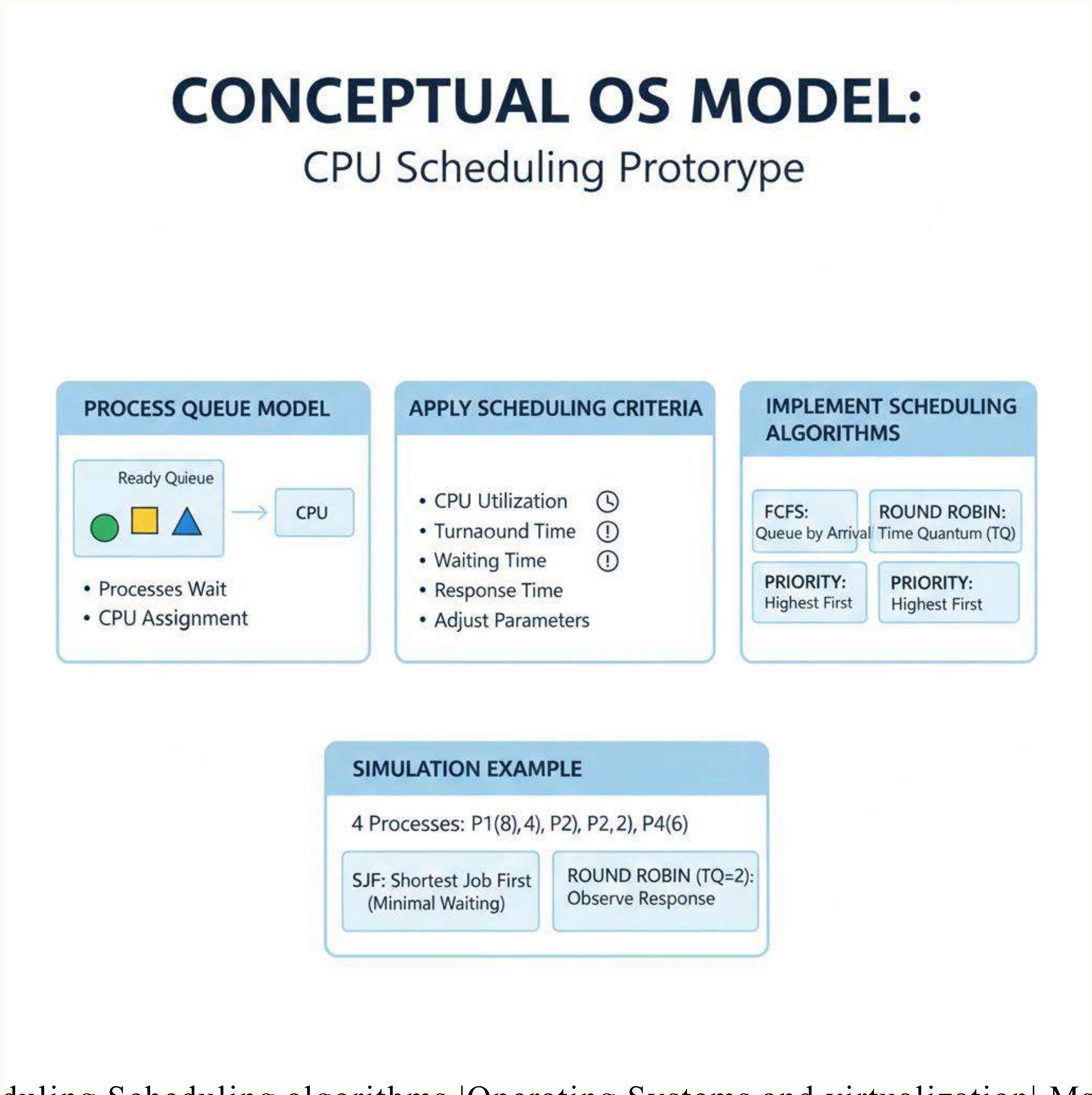
Different Systems Need Different Solutions

Scheduling Algorithms

- **FCFS** First Come, First Serve
- **SJF / SRTF** Shorter job First
- **Round Robin** Run every time slices
- **Priority Scheduling** Important tasks run first
- **Multilevel Feedback Queue**

Batch Systems → SJF / FCFS ● Interactive Systems → Round Robin

Mixed Systems → Multilevel Feedback Queue (Best Balance)



CONCEPTUAL OS MODEL:

CPU Scheduling Protorype

PROCESS QUEUE MODEL

- Processes Wait
- CPU Assignment

APPLY SCHEDULING CRITERIA

- CPU Utilization
- Turnaound Time
- Waiting Time
- Response Time
- Adjust Parameters

IMPLEMENT SCHEDULING ALGORITHMS

FCFS: Queue by Arrival

ROUND ROBIN: Time Quantum (TQ)

PRIORITY: Highest First

PRIORITY: Highest First

SIMULATION EXAMPLE

4 Processes: P1(8),4), P2), P2,2), P4(6)

SJF: Shortest Job First (Minimal Waiting)

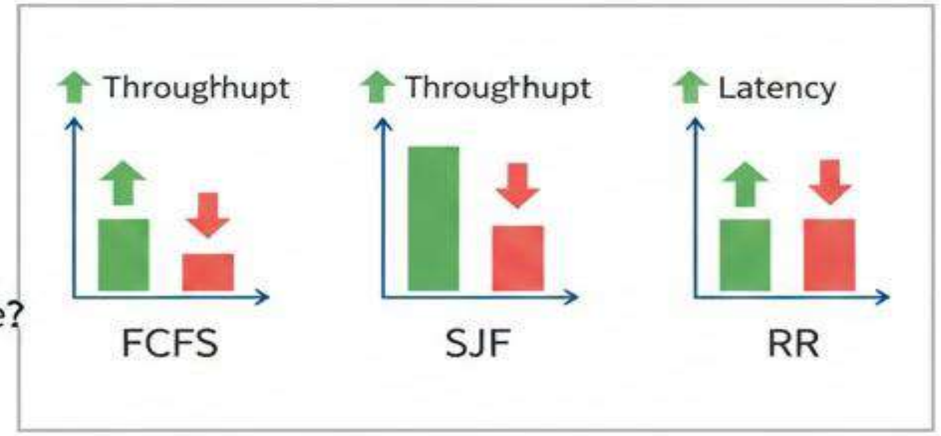
ROUND ROBIN (TQ=2): Observe Response

OPERATING SYSTEM CPU SCHEDULING

TEST & VALIDATION

- Check Performance Metrics
- CPU Utilization High?
- Turnaound/Waiting Times
  Turnaroiand Times Minimized?
  Fast Response for Interactive Processe?
- Fairness? No Starvation?

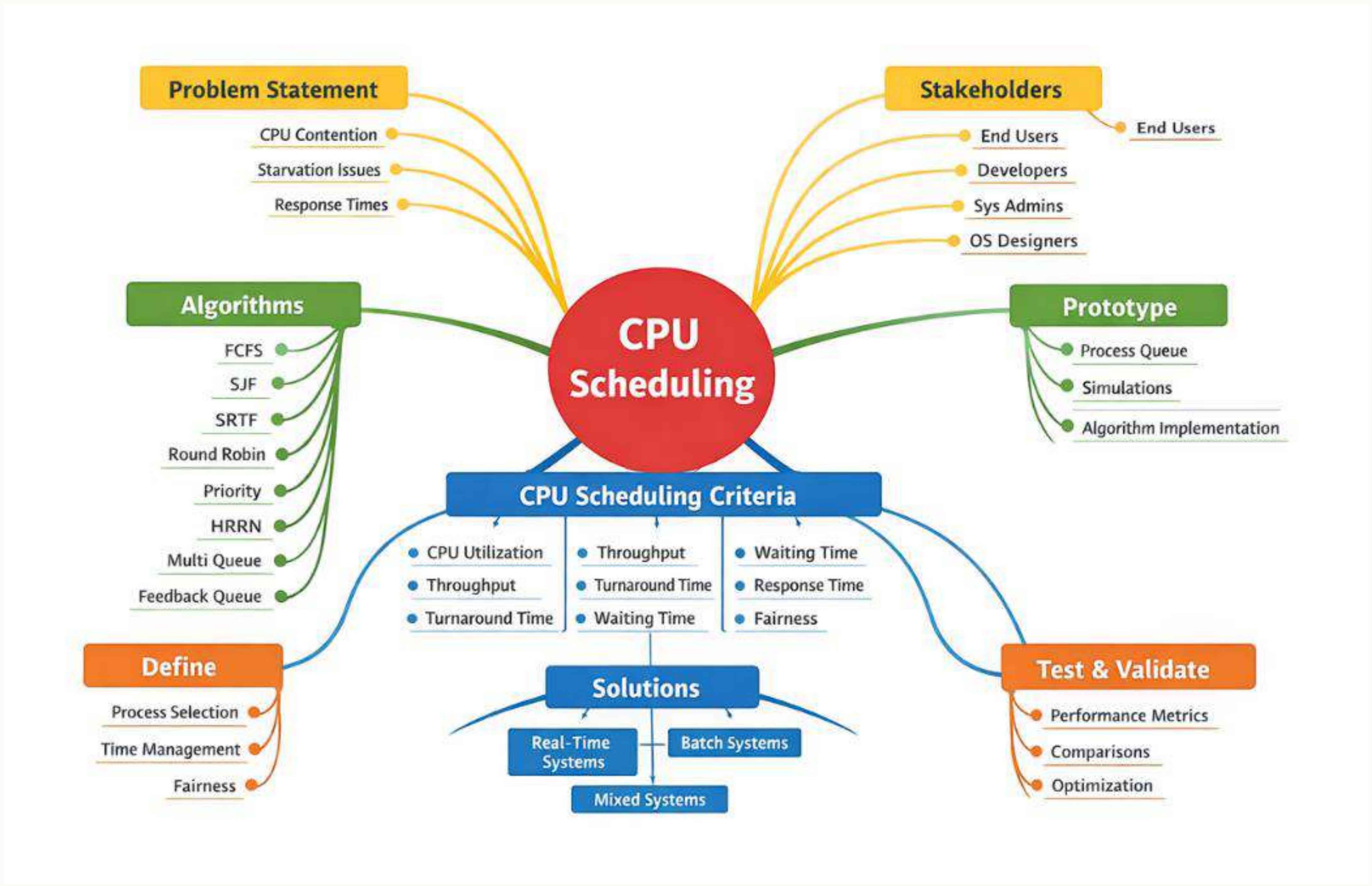
ALGORTHM COMPARISON



VALIDATION SCENARIO

- 10 Processes (Varying CPU/Priority)
- Compare FCFS, SJF, RR
- Evaluate Algorithm Performance

KEY INSIGHT:
No single algorithm is best.
Multilevel Fedback Queue:
Flexible, Fair, Efficient.


CPU Scheduling
Problem Statement
CPU Contention
Starvation Issues
Response Times
Stakeholders
End Users
End Users
Developers
Sys Admins
OS Designers
Algorithms
FCFS
SJF
SRTF
Round Robin
Priority
HRRN
Multi Queue
Feedback Queue
Prototype
Process Queue
Simulations
Algorithm Implementation
CPU Scheduling Criteria
CPU Utilization
Throughput
Turnaround Time
Throughput
Turnaround Time
Waiting Time
Waiting Time
Response Time
Fairness
Define
Process Selection
Time Management
Fairness
Solutions
Real-Time Systems
Batch Systems
Mixed Systems
Test & Validate
Performance Metrics
Comparisons
Optimization

THE OS JUGGLER

P2
(5 units)
P3
8 units)
P1
(10 units)
PU
P4
6 units)
P5
2 units)

THE CHALLENGE:

1. Minimize Total Waiting Time
2. Ensure Fairness (No Starvation)

YOUR TASK:
Decide the Juggling Order!
Which CPU Scheduling Strategy?

FCFS SJF Priority Round Robbin

Solution:

Goal: Minimize total waiting time → shortest jobs first.

Given execution times:

- **P1 = 10, P2 = 5, P3 = 8, P4 = 6, P5 = 2**

Step 1: Sort processes by execution time (Shortest Job First – SJF):

- **P5 (2), P2 (5), P4 (6), P3 (8), P1 (10)**

Step 2: Order of execution:

- **CPU executes: P5 → P2 → P4 → P3 → P1**

Step 3: Waiting time calculation (optional):

- **P5 waits 0**
- **P2 waits 2 (P5's time)**
- **P4 waits 2 + 5 = 7**
- **P3 waits 7 + 6 = 13**
- **P1 waits 13 + 8 = 21**

Result:

- **Total waiting time = 0 + 2 + 7 + 13 + 21 = 43 units**
- **This is the minimum total waiting time compared to any other order.**

Thank you!