SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution

Coimbatore-35

Department of Computer Science and Engineering

23CST206–OPERATING SYSTEMS AND VIRTUALIZATION

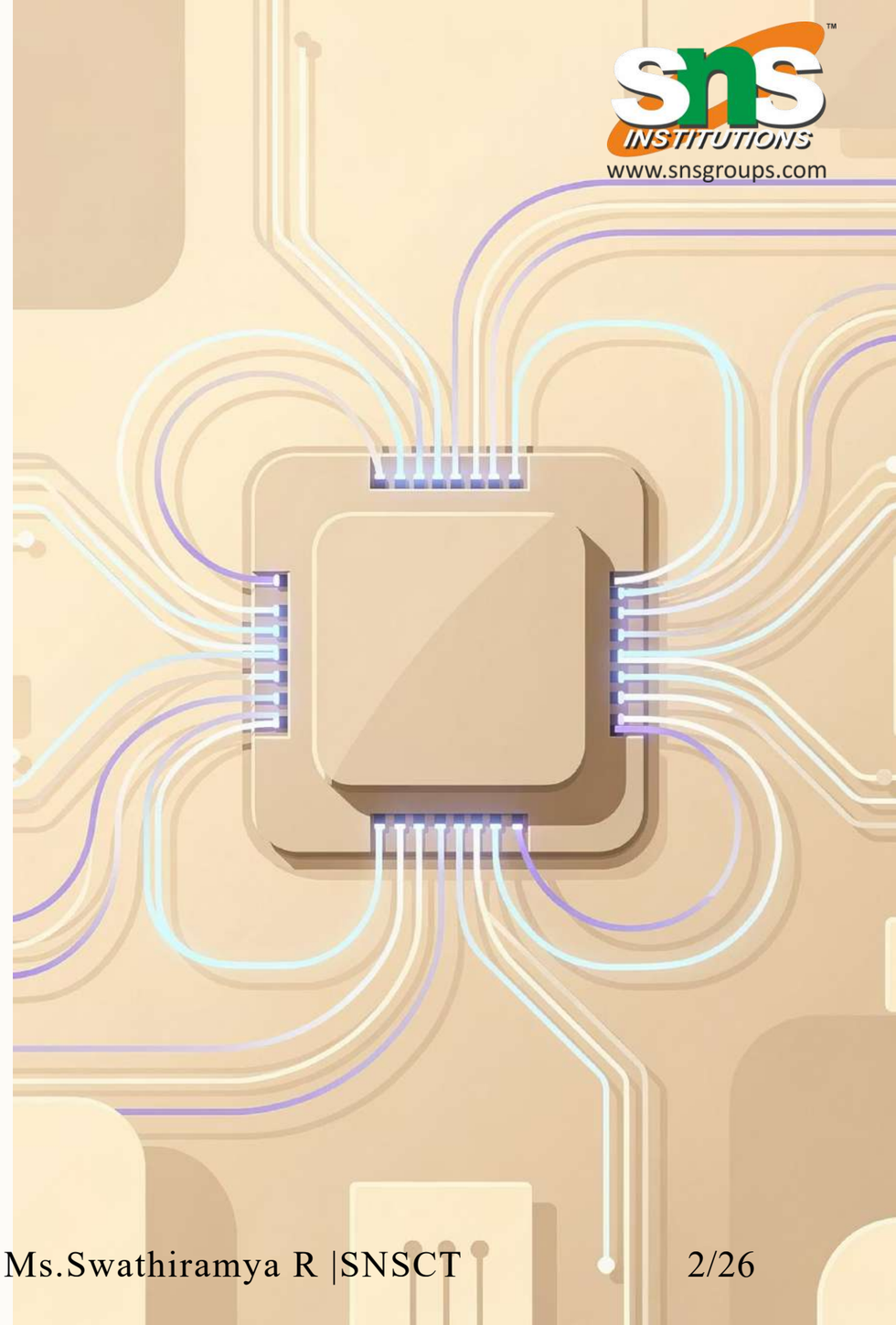
B.E- CSE /IV SEMESTER

UNIT – II PROCESS MANAGEMENT

Topic 1:Process Scheduling - Inter-process Communication

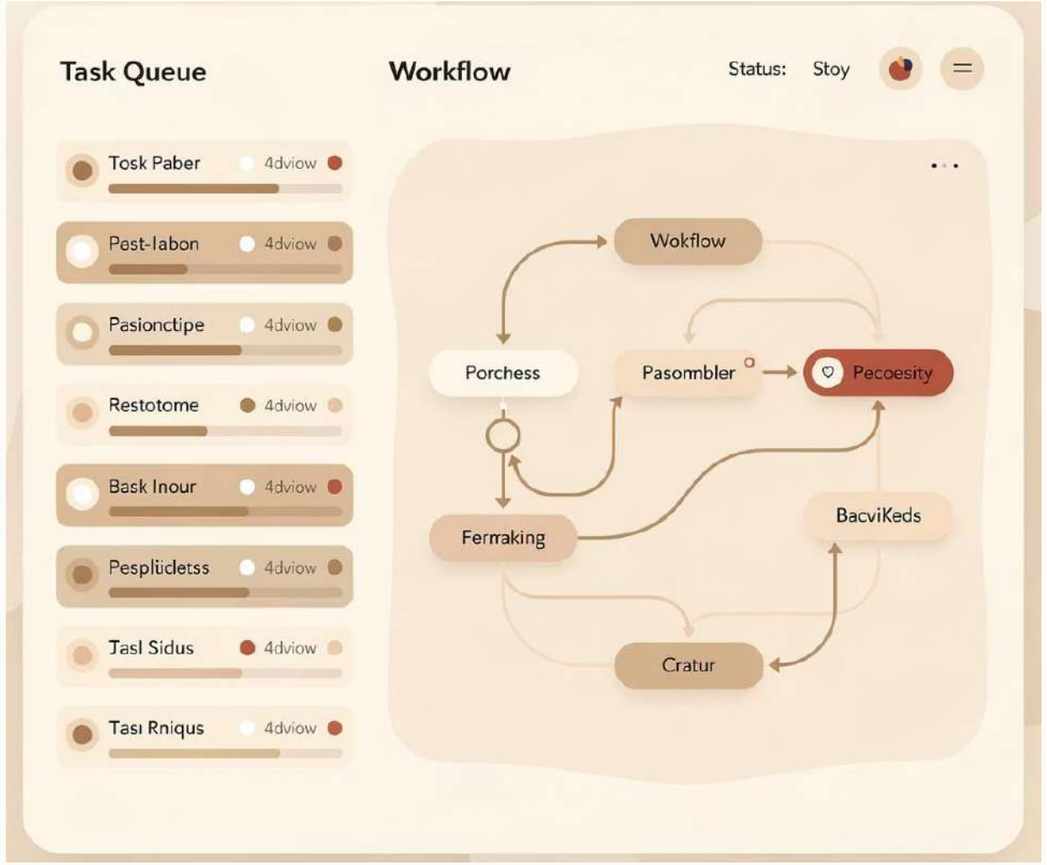
Process Scheduling & Inter-Process Communication

Exploring the core mechanisms that enable operating systems to manage multiple processes efficiently and facilitate communication between them.



Process Scheduling

- Process scheduling is a core function of the operating system that decides which process runs next on the CPU.
- Since multiple processes may be ready simultaneously, the OS uses scheduling algorithms to manage CPU allocation efficiently.
- The process manager handles removal of the running process from the CPU and selection of another process based on a particular strategy.



Objectives of Process Scheduling

Maximize CPU Utilization

Keep the processor busy and productive at all times.

Minimize Wait Times

Reduce waiting time and turnaround time for all processes.

Ensure Fairness

Provide equitable CPU access among all processes.

Prevent Starvation

Ensure low-priority processes eventually get CPU time.

Responsive Behavior

Maintain system responsiveness in real-time applications.

Categories of Scheduling

Non-Preemptive

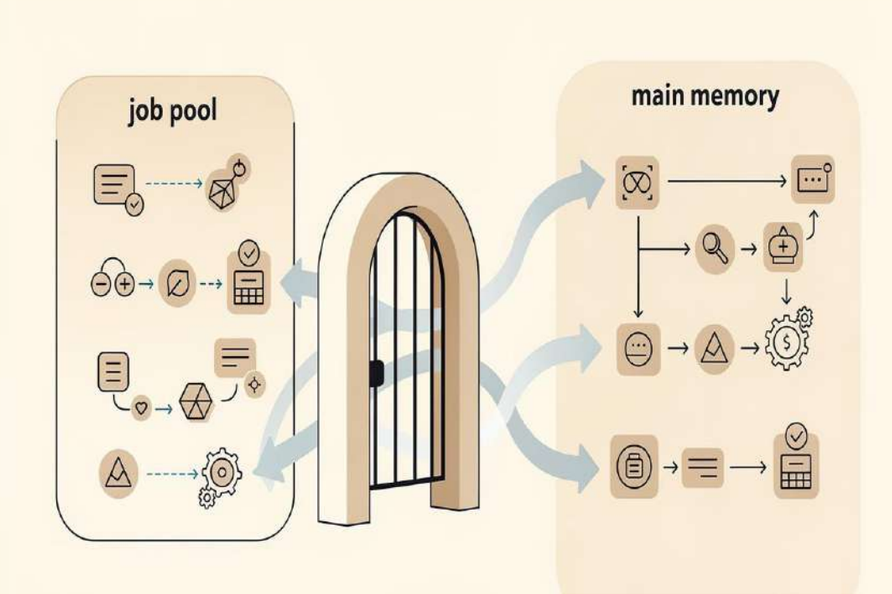
A process's resource cannot be taken before it finishes running. Resources switch only when a running process finishes and transitions to a waiting state.

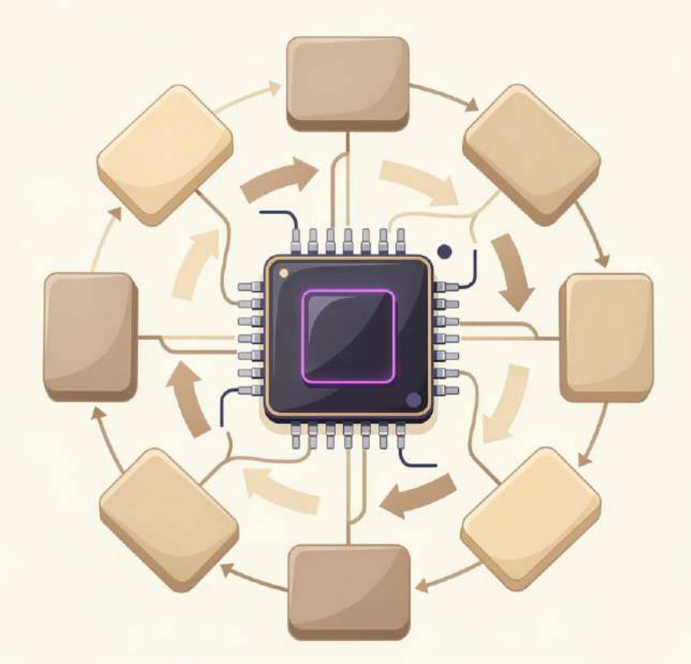
Preemptive

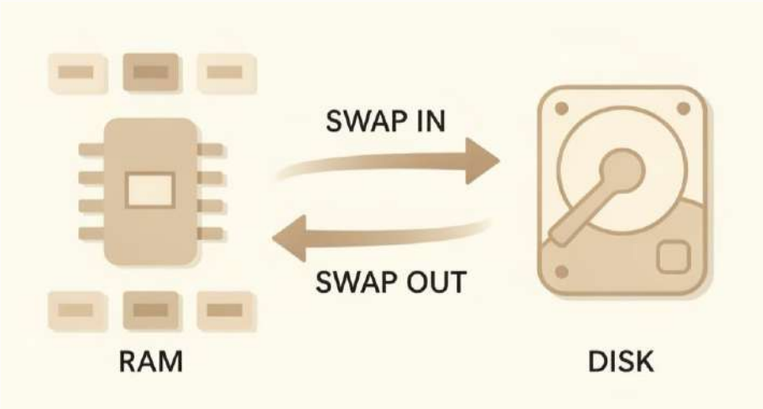
The OS can switch a process from running to ready state. The CPU may give other processes priority and substitute the currently active process for a higher priority one.

Process Scheduler

Process Schedulers are fundamental components of operating systems responsible for deciding the order in which processes are executed by the CPU.



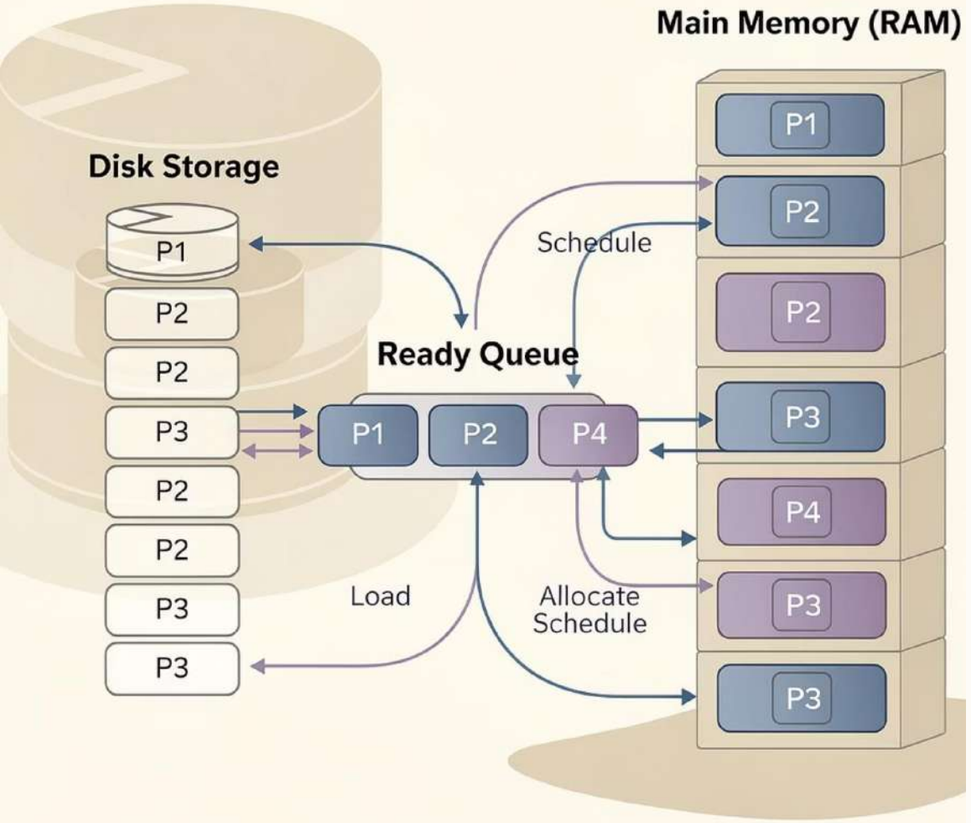




Long-Term Scheduler (Job Scheduler)

Short-Term Scheduler (CPU Scheduler)

Medium-Term Scheduler



TYPE 1

Long-Term Scheduler

Job Scheduler

The Long-Term Scheduler loads processes from disk into main memory so they can begin execution. When admitted, a process moves into the Ready state.

- Moves processes from Job Queue → Ready Queue
- Controls degree of multiprogramming
- Prevents CPU or I/O devices from remaining idle
- May not exist in modern time-sharing systems like Windows

TYPE 2

Short-Term Scheduler

CPU Scheduler

Core Function

Selects a process from the ready queue and assigns the CPU to it.

Speed Requirement

Must be fast because it executes frequently, often every few milliseconds.

Key Goals

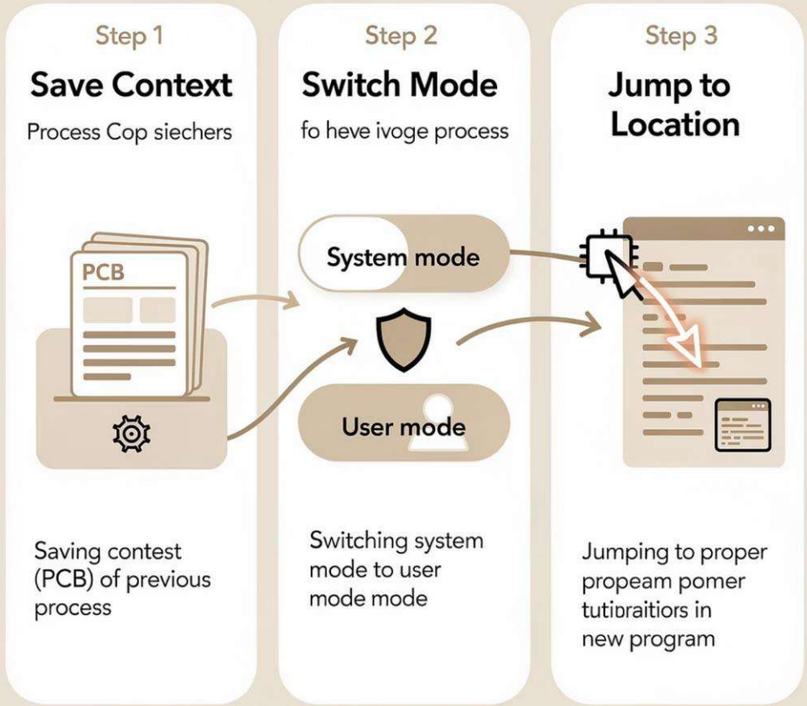
Ensures no process suffers from starvation and maximizes CPU utilization by keeping the processor busy.

Implementation

Uses various CPU scheduling algorithms to decide process order. Example: Real-time OS.

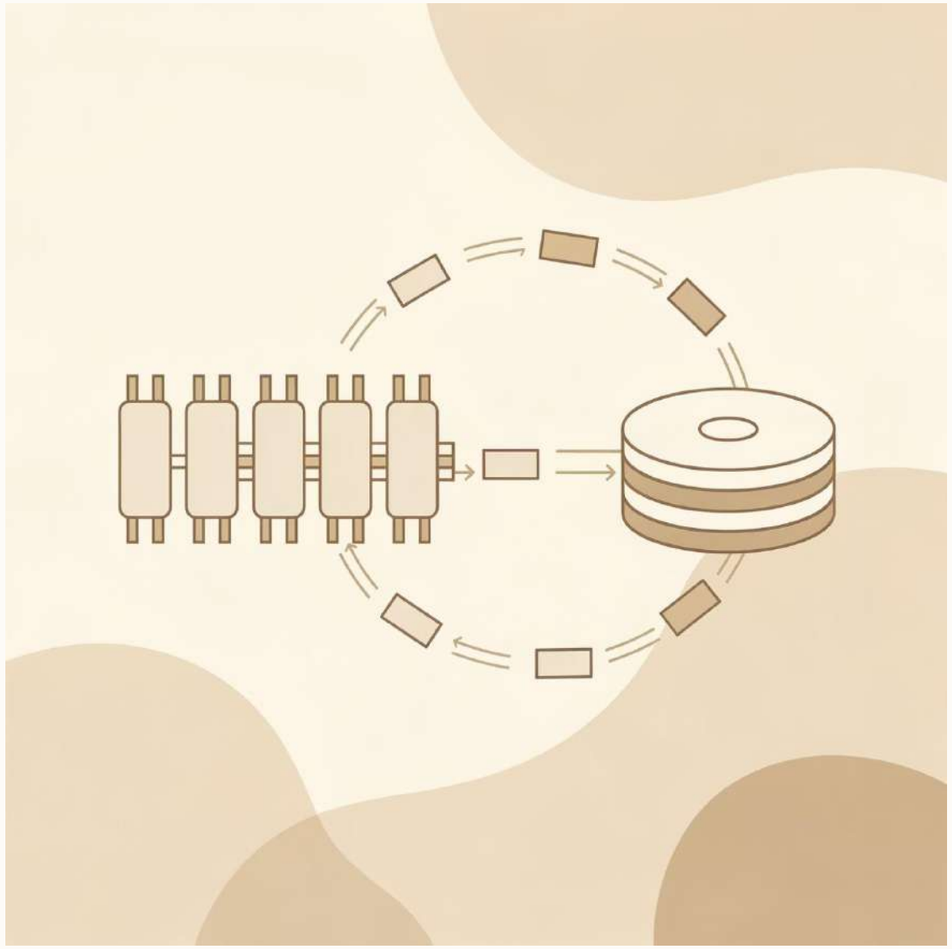
The Dispatcher

- The dispatcher is responsible for loading the process selected by the Short-term scheduler onto the CPU (Ready to Running State).
- Context switching is done by the dispatcher only.
- The Dispatcher is a special program that takes over once the short-term scheduler selects a process. It transfers control of the CPU to the chosen process.



TYPE 3

Medium-Term Scheduler



The Medium-Term Scheduler manages swapping, which temporarily moves processes between main memory and disk.

- Swaps processes out of memory when waiting (e.g., blocked for I/O)
- Reduces the degree of multiprogramming
- Frees memory for other active processes
- Swaps processes back when ready to continue execution
- Operates faster than long-term but slower than short-term scheduler

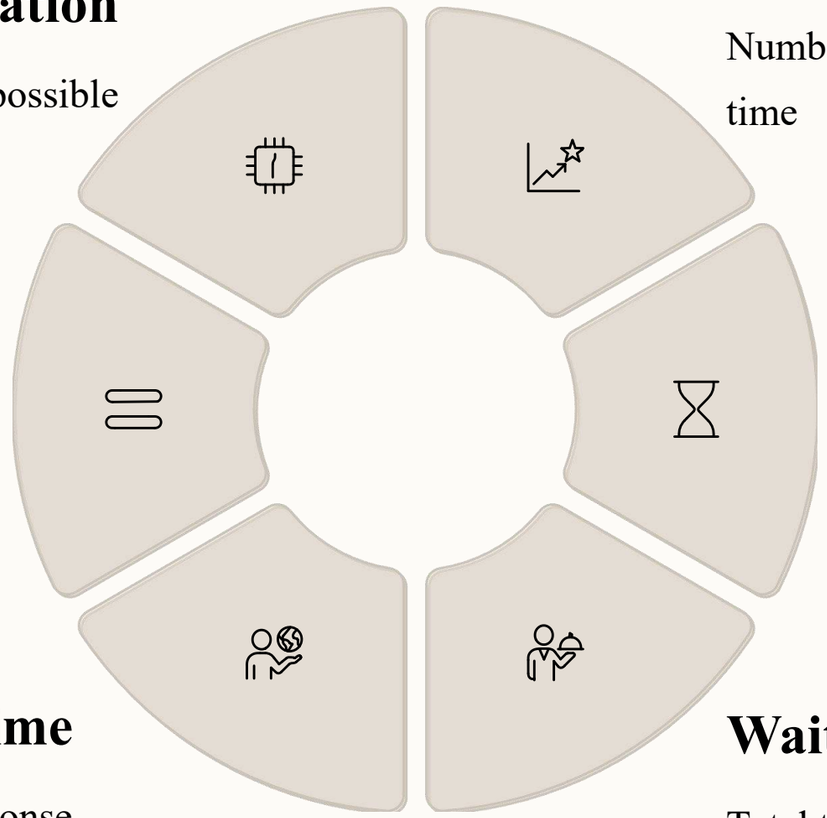
Scheduling Criteria

CPU Utilization

Keep CPU as busy as possible

Throughput

Number of processes completed per unit time



Fairness

Ensure all processes get a chance

Turnaround Time

Total time from submission to completion

Response Time

Time from submission to first response

Waiting Time

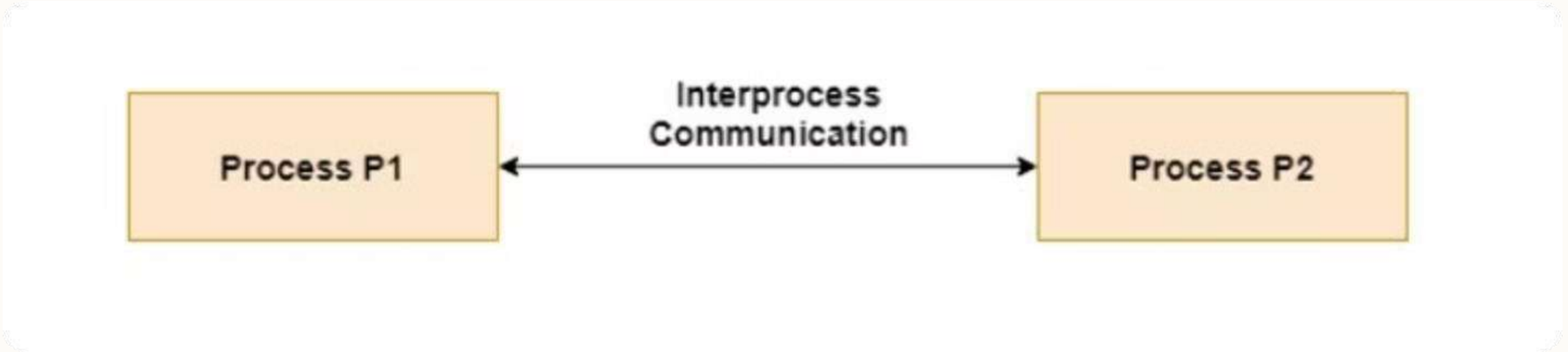
Total time in ready queue

Inter-Process Communication (IPC)

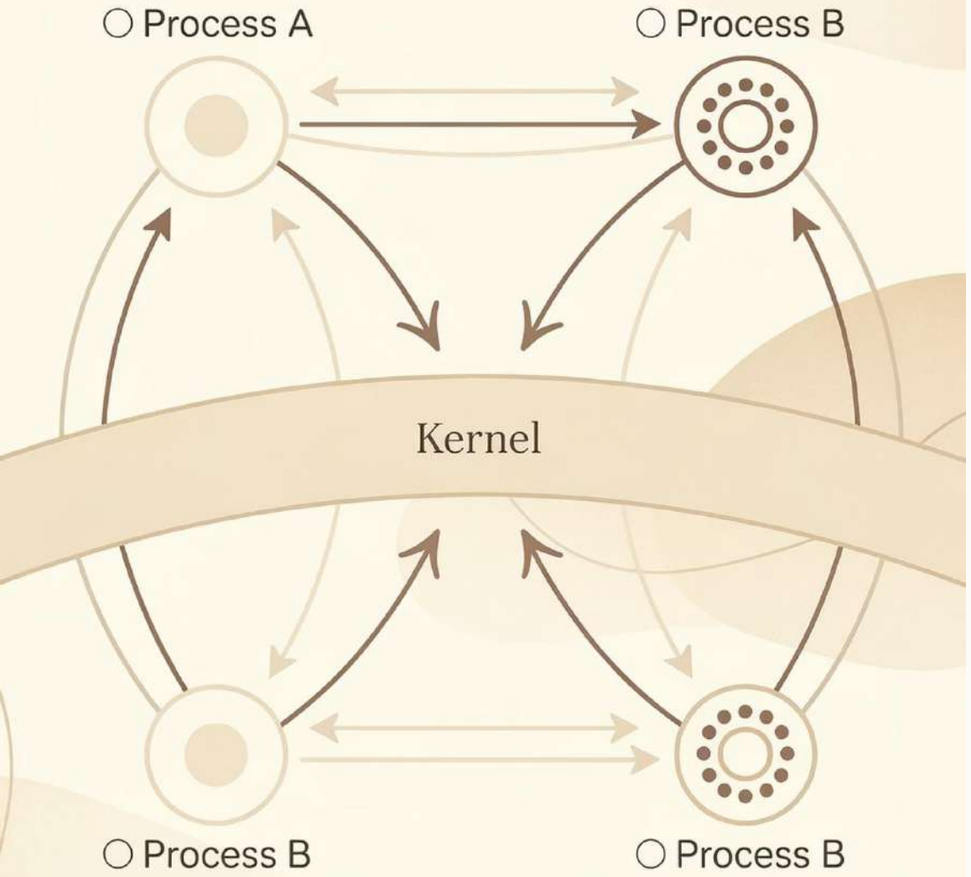
- When processes execute concurrently, they may need to exchange data or synchronize actions. This is called Inter-Process Communication (IPC).

- IPC is a mechanism which allows the exchange of data between processes, enabling resource and data sharing without interference.

- This communication involves a process letting another process know that some event has occurred or there has been data transfer between two or more processes.



IPC Fundamentals



Why IPC is Needed

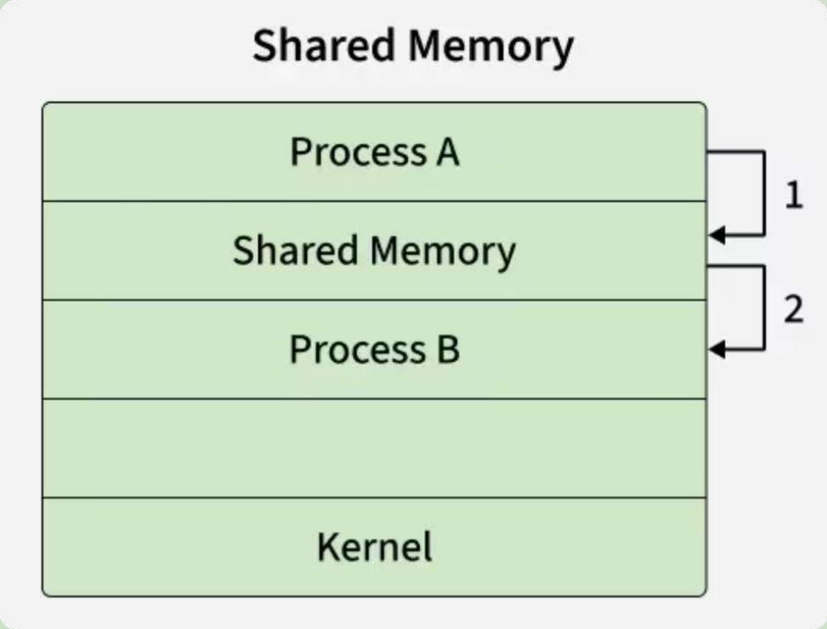
- Share data between processes
- Synchronize process actions (avoid conflicts)
- Cooperating processes need communication for efficiency

Types of Processes

Independent Processes: Do not share data or resources; no IPC needed.

Cooperating Processes: Share data, files, or resources; require IPC.

Shared Memory



A region of memory shared between processes. Communication using shared memory requires processes to share some variable and completely depends on how the programmer implements it.

1

Step 1: Write

Process A writes data into the shared memory region allocated by the kernel.

2

Step 2: Read

Process B directly reads this data from the same shared memory region.

Message Passing

Processes communicate via sending and receiving messages through the kernel.

Communication Types

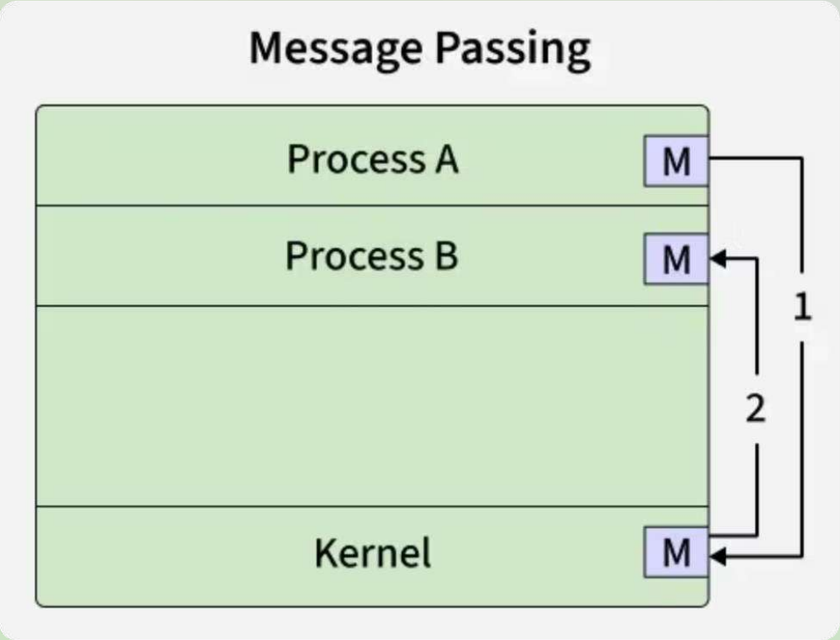
Direct Communication:
Processes explicitly name each other (send/receive).

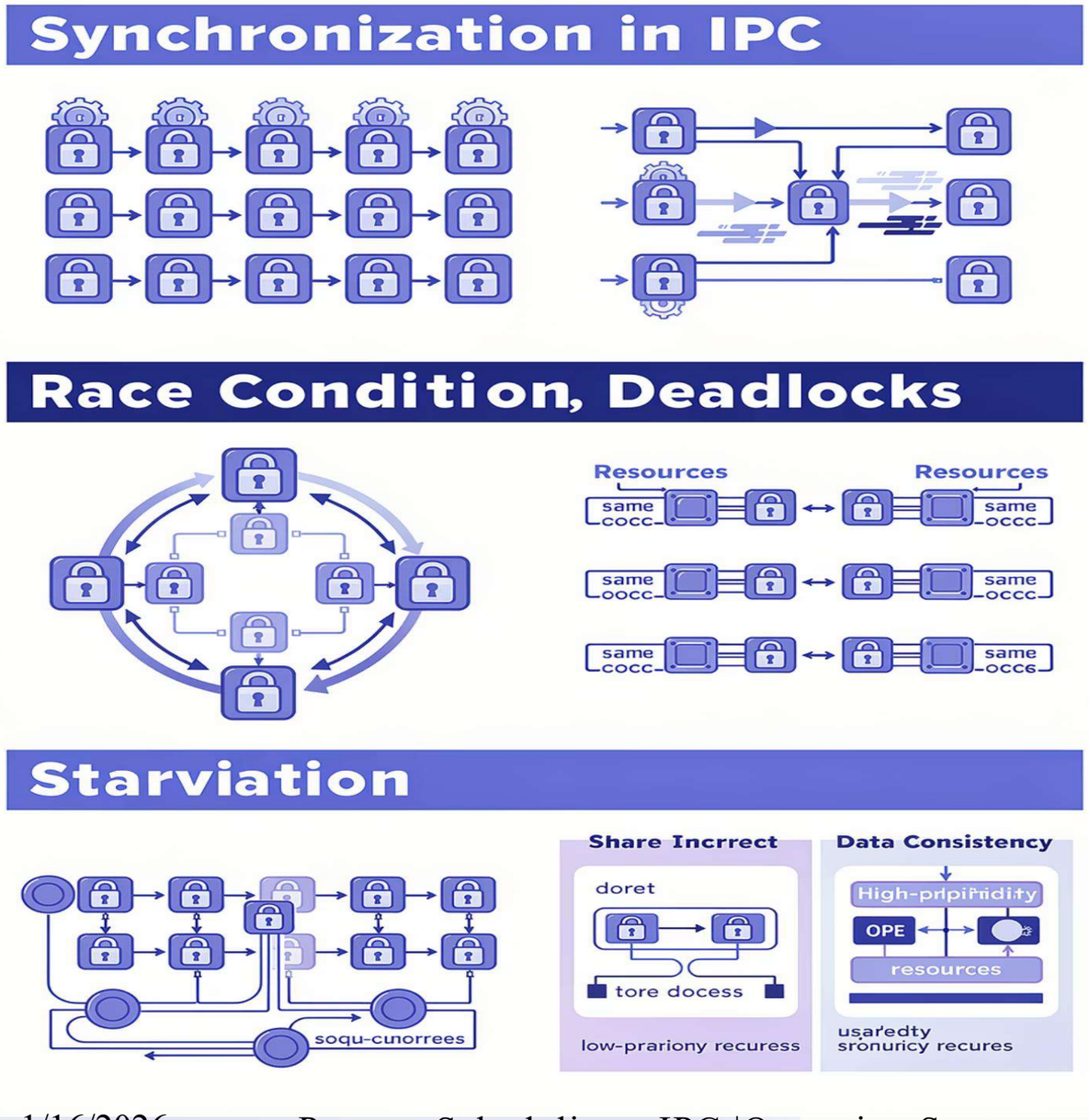
Indirect Communication:
Processes use mailboxes/ports to send and receive messages.

Process Flow

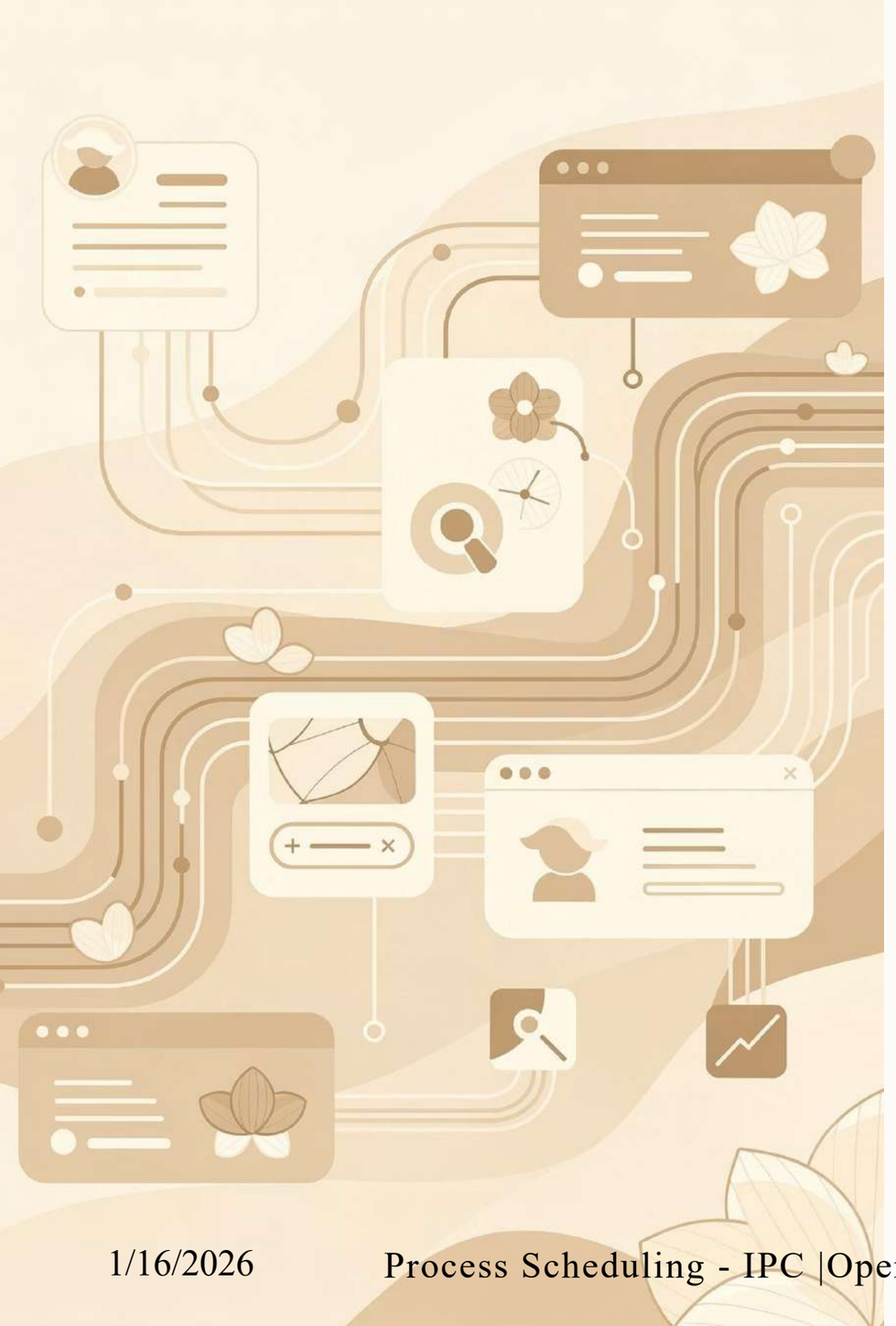
1. Process A sends a message to the kernel
2. The kernel delivers the message to Process B

Can be synchronous (blocking) or asynchronous (non-blocking).



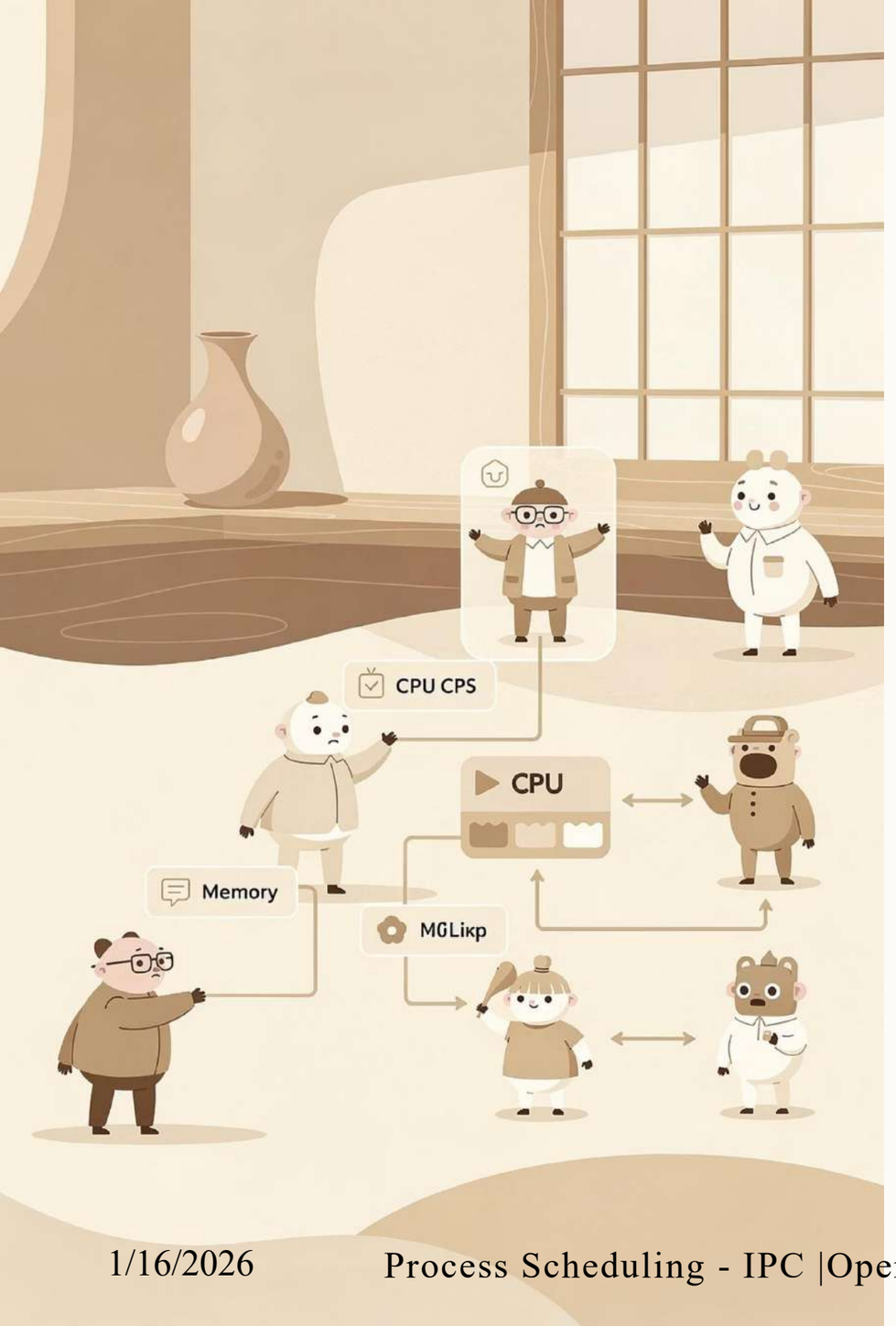


- **Synchronization & Key Issues**
  - **Synchronization in IPC**
  - **Race Conditions**
  - **Deadlocks**
  - **Starvation**
  - **Data Consistency**



Design Thinking Approach to Process Scheduling & Inter-Process Communication

Exploring how operating systems orchestrate concurrent processes and enable safe communication through systematic design thinking methodology.



PROBLEM STATEMENT

The Concurrent Process Challenge

Operating systems run many tasks at once, all needing the same limited resources like CPU and memory. Without smart management, this leads to big problems:

- Some tasks hog the CPU.
- Others get no resources.
- Shared data can get corrupted.
- The system slows down or crashes.

Core Challenge: How can an operating system manage these tasks and their communication fairly, efficiently, and reliably?

Empathize

Understand Stakeholders

End Users

- Smooth Multitasking
- Fast Response
- No Freezes or Delays

End Users

Application Developers

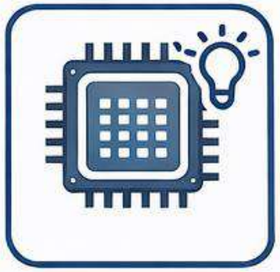
- Predictable Execution
- Safe Data Sharing
- OS Synchronization

Application Developers

System Administrators

- Efficient CPU Utilization
- Controlled Resource Sharing
- Stable & Secure Systems

System Administrators

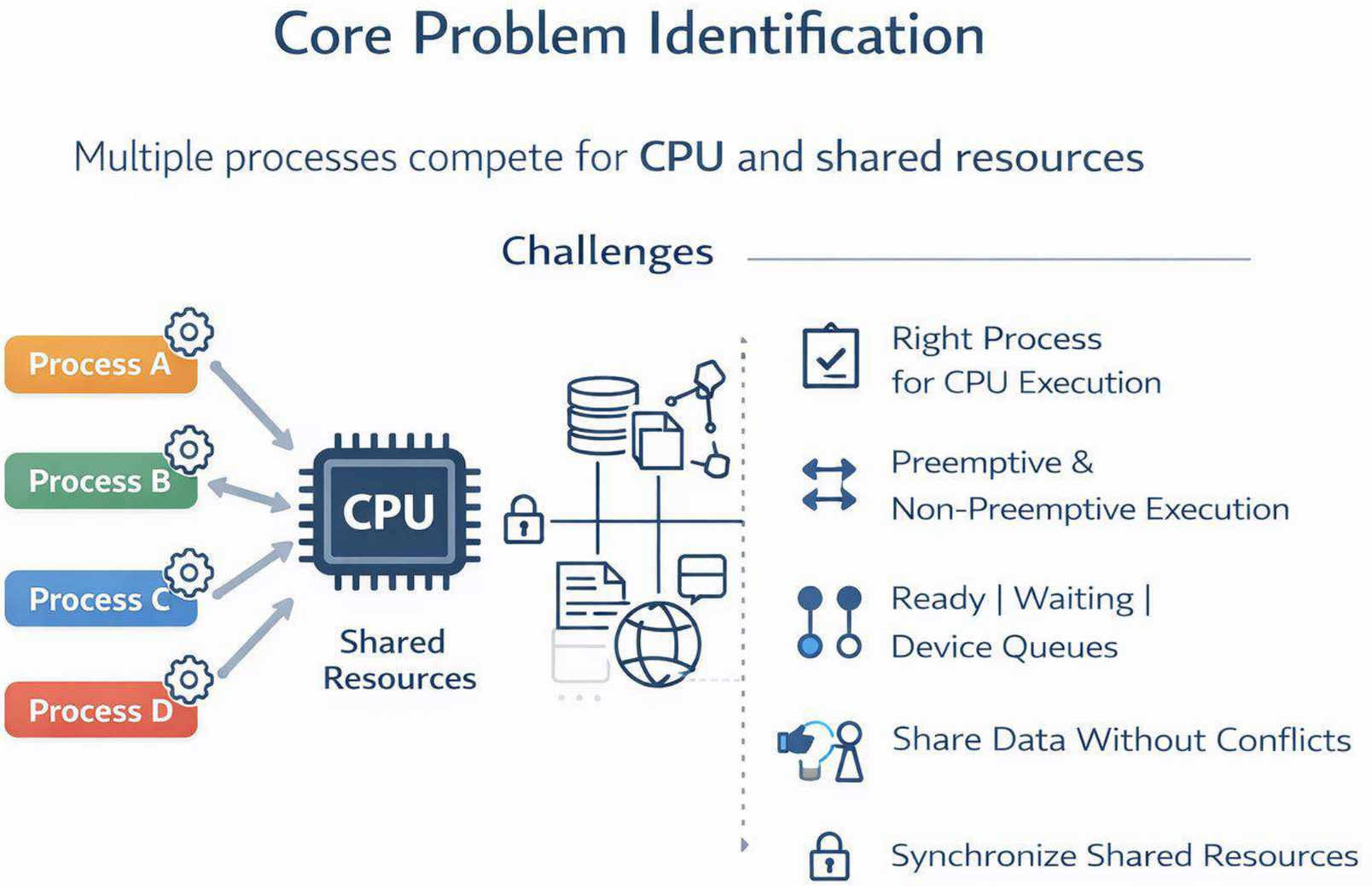


Operating System Designers

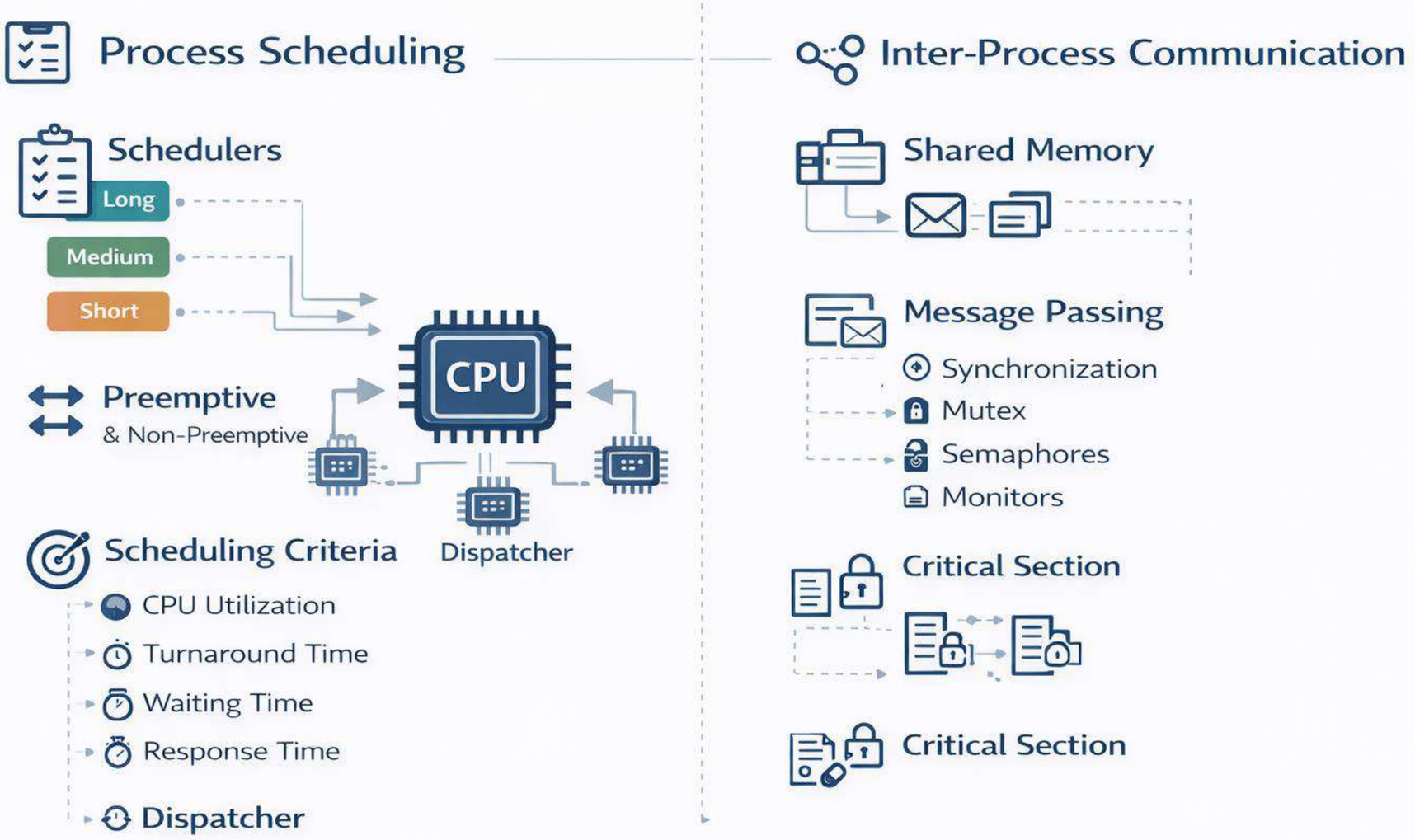
- Fairness & Performance
- Efficient Scheduling
- Prevent Issues & Deadlocks

Operating System Designers

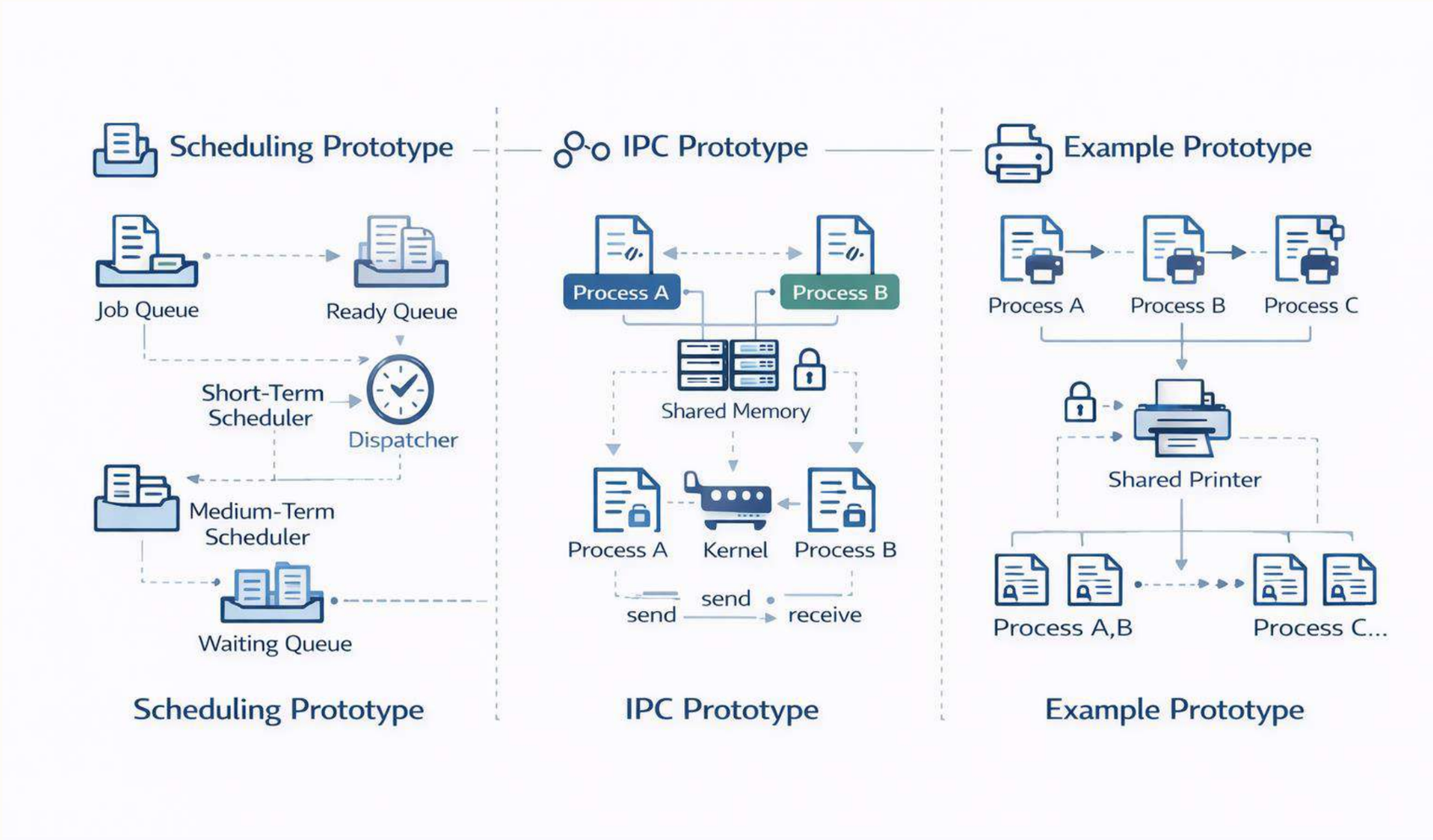
Define (Core Problem Identification)



Ideate (Possible Solutions)



Prototype (Conceptual OS Model)



Test: Validation and Evaluation

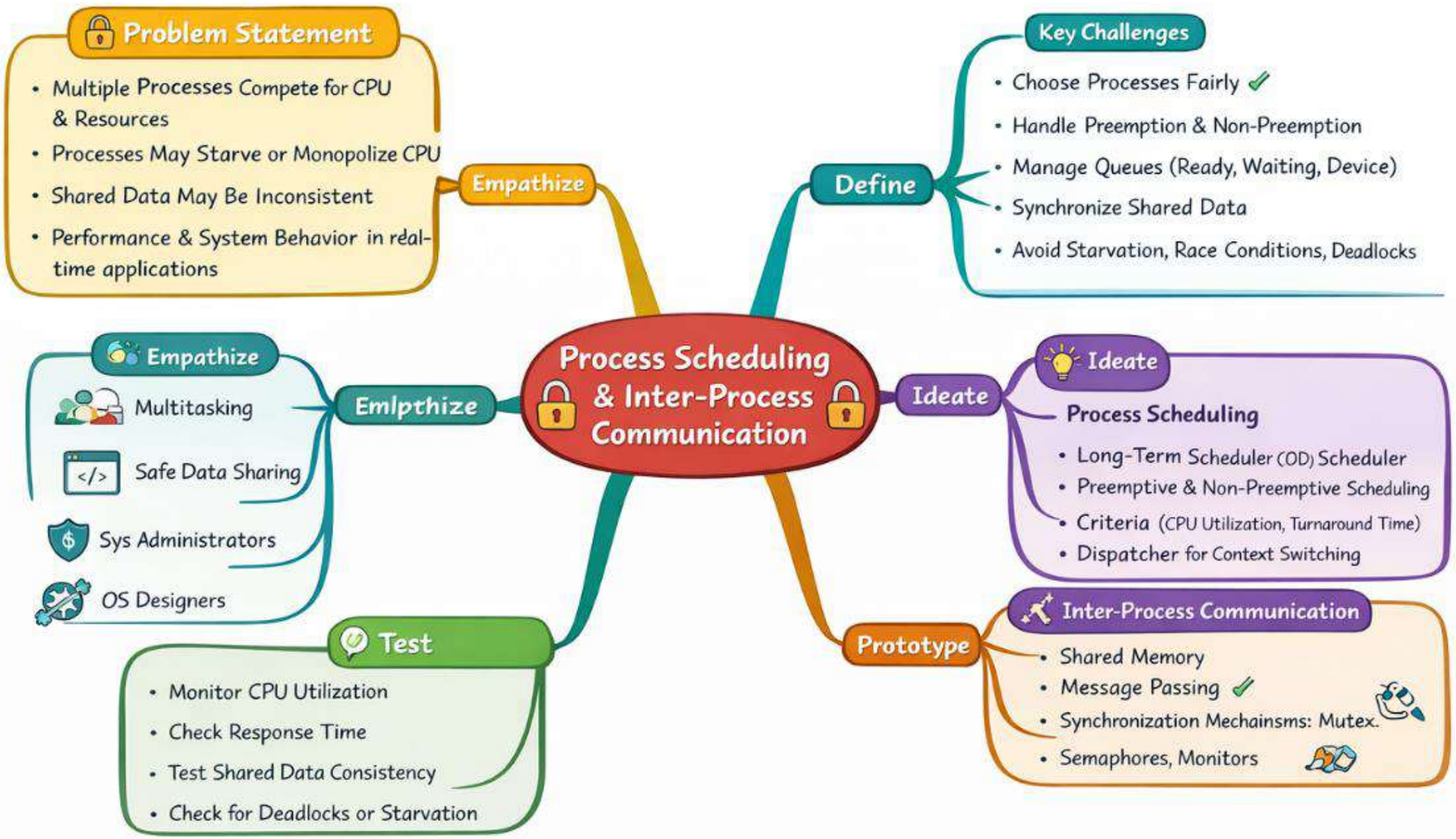
The system undergoes rigorous testing to validate scheduling efficiency and IPC correctness.

Scheduling Tests

- Is CPU always busy?
- Do all processes get fair CPU time?
- Is response time acceptable?
- Are low-priority processes starving?

IPC Tests

- Is shared data consistent?
- Are race conditions avoided?
- Are deadlocks prevented?
- Does communication occur correctly?


Process Scheduling & Inter-Process Communication
Empathize
Problem Statement
Multiple Processes Compete for CPU & Resources
Processes May Starve or Monopolize CPU
Shared Data May Be Inconsistent
Performance & System Behavior in real-time applications
Define
Key Challenges
Choose Processes Fairly
Handle Preemption & Non-Preemption
Manage Queues (Ready, Waiting, Device)
Synchronize Shared Data
Avoid Starvation, Race Conditions, Deadlocks
Emlpthize
Empathize
Multitasking
Safe Data Sharing
Sys Administrators
OS Designers
Ideate
Ideate
Process Scheduling
Long-Term Scheduler (OD) Scheduler
Preemptive & Non-Preemptive Scheduling
Criteria (CPU Utilization, Turnaround Time)
Dispatcher for Context Switching
Test
Monitor CPU Utilization
Check Response Time
Test Shared Data Consistency
Check for Deadlocks or Starvation
Prototype
Inter-Process Communication
Shared Memory
Message Passing
Synchronization Mechainsms: Mutex.
Semaphores, Monitors

The Confused Computer Lab Puzzle

In a computer lab, three processes are running simultaneously on a single CPU with one shared printer. Let's examine what happens:

Step 1

1

The CPU starts executing Process A – a student typing code

2

Step 2

Suddenly, Process B needs immediate CPU time to keep music playing smoothly

Step 3

3

While this is happening, Process C sends data to the printer

4

Step 4

Process A and Process C both try to update a shared log file simultaneously

Result: Success

Music plays smoothly, printing completes, shared log remains uncorrupted, and all processes finish successfully

Question: How did the operating system manage these competing processes without crashes or data corruption?

SOLUTION

Puzzle Solution Explained

Preemptive Scheduling

The CPU switched from Process A to Process B using preemptive scheduling to maintain system responsiveness and ensure smooth music playback.

Resource Protection

Printing occurred one job at a time because the printer was a shared resource protected by synchronization mechanisms.

Mutual Exclusion

The shared log file was protected using IPC with mutual exclusion, preventing data corruption and race conditions.