SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution

Coimbatore-35

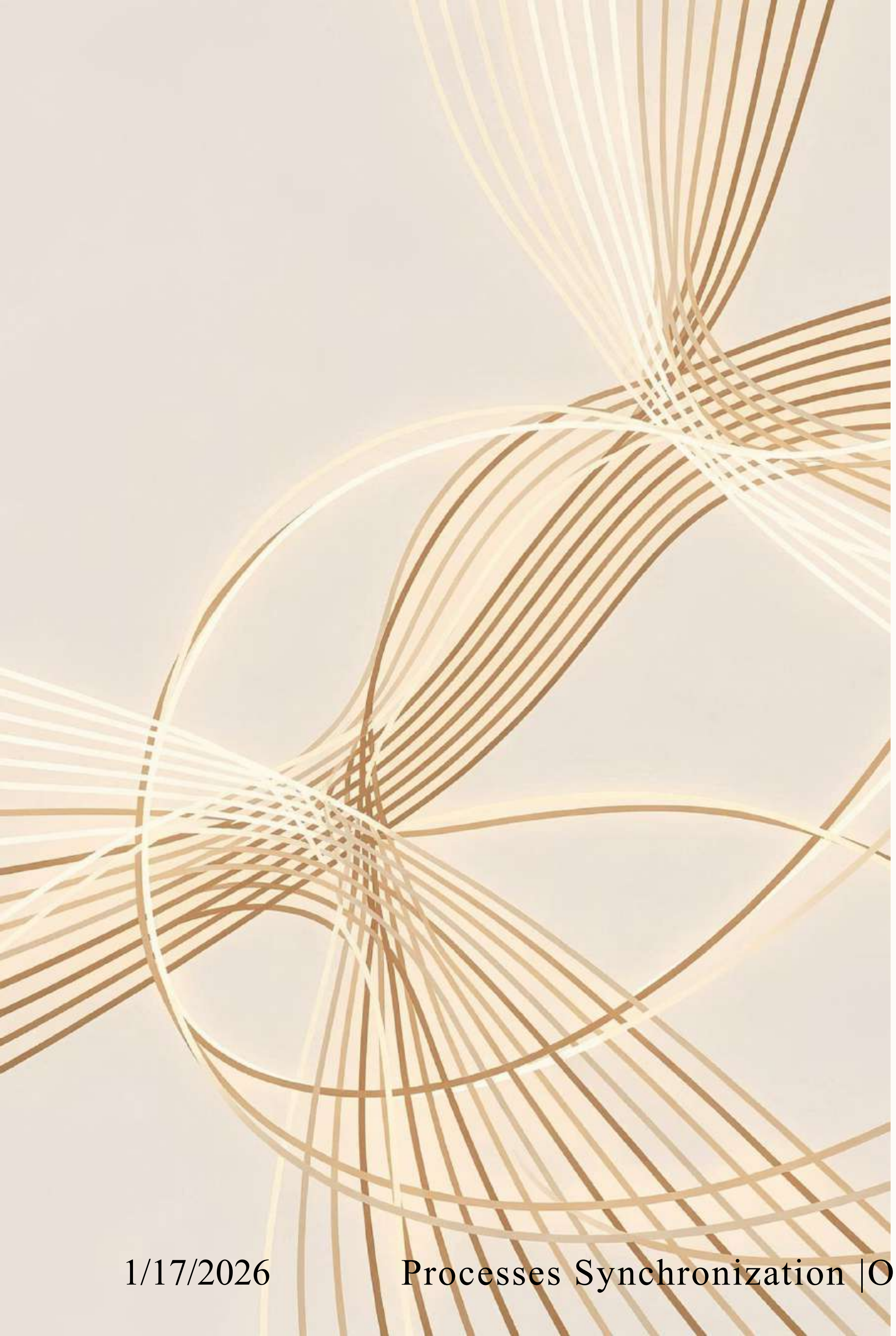
Department of Computer Science and Engineering

23CST206–OPERATING SYSTEMS AND VIRTUALIZATION

B.E- CSE /IV SEMESTER

UNIT – II PROCESS MANAGEMENT

Topic 5:Processes Synchronization

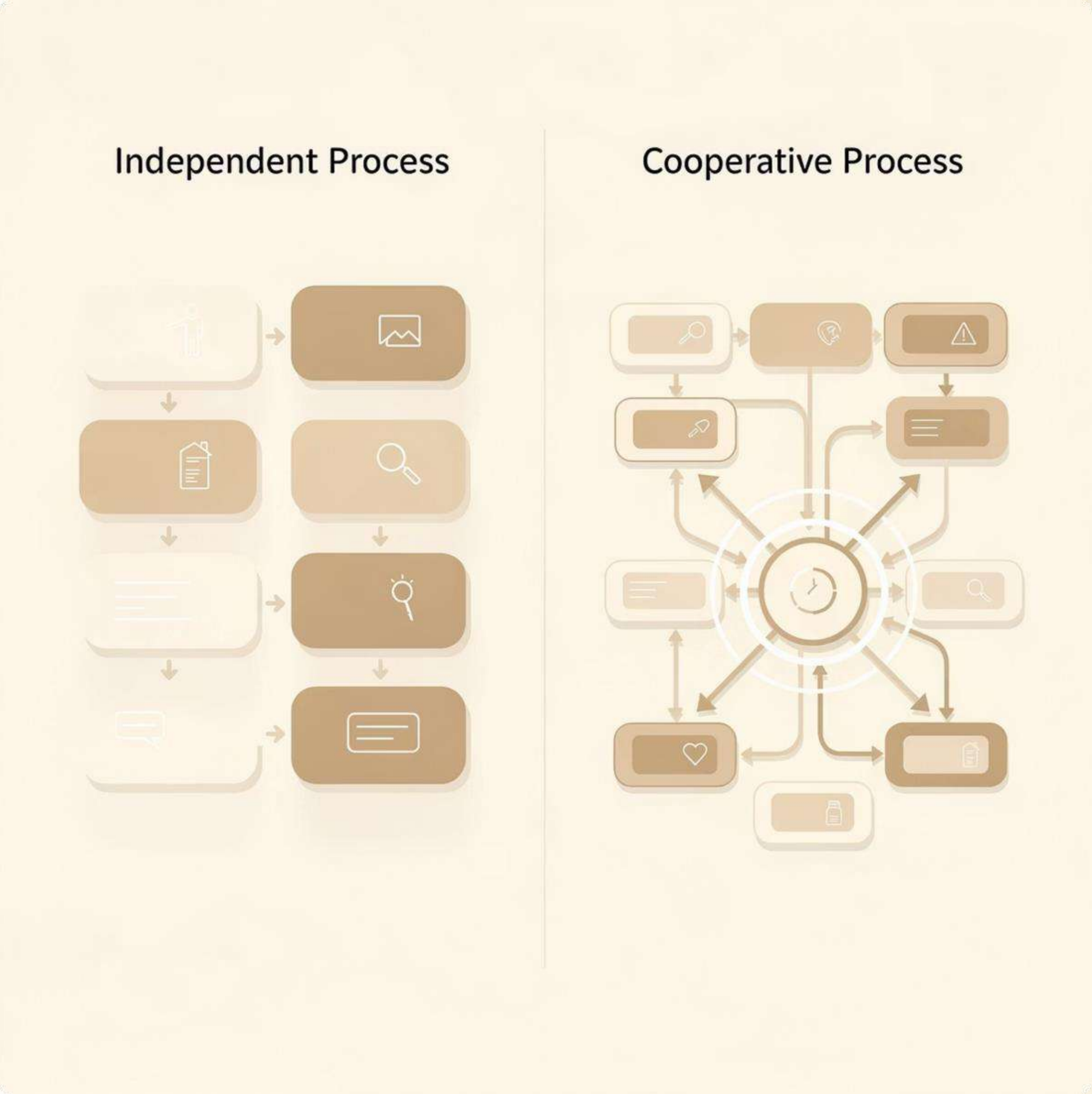


Process Synchronization

Managing concurrent process execution in modern operating systems to ensure safe, consistent access to shared resources

CHAPTER 1

Understanding Process Types



Independent Process

Executes without affecting other processes

- No shared resources
- Isolated execution
- No coordination needed

Cooperative Process

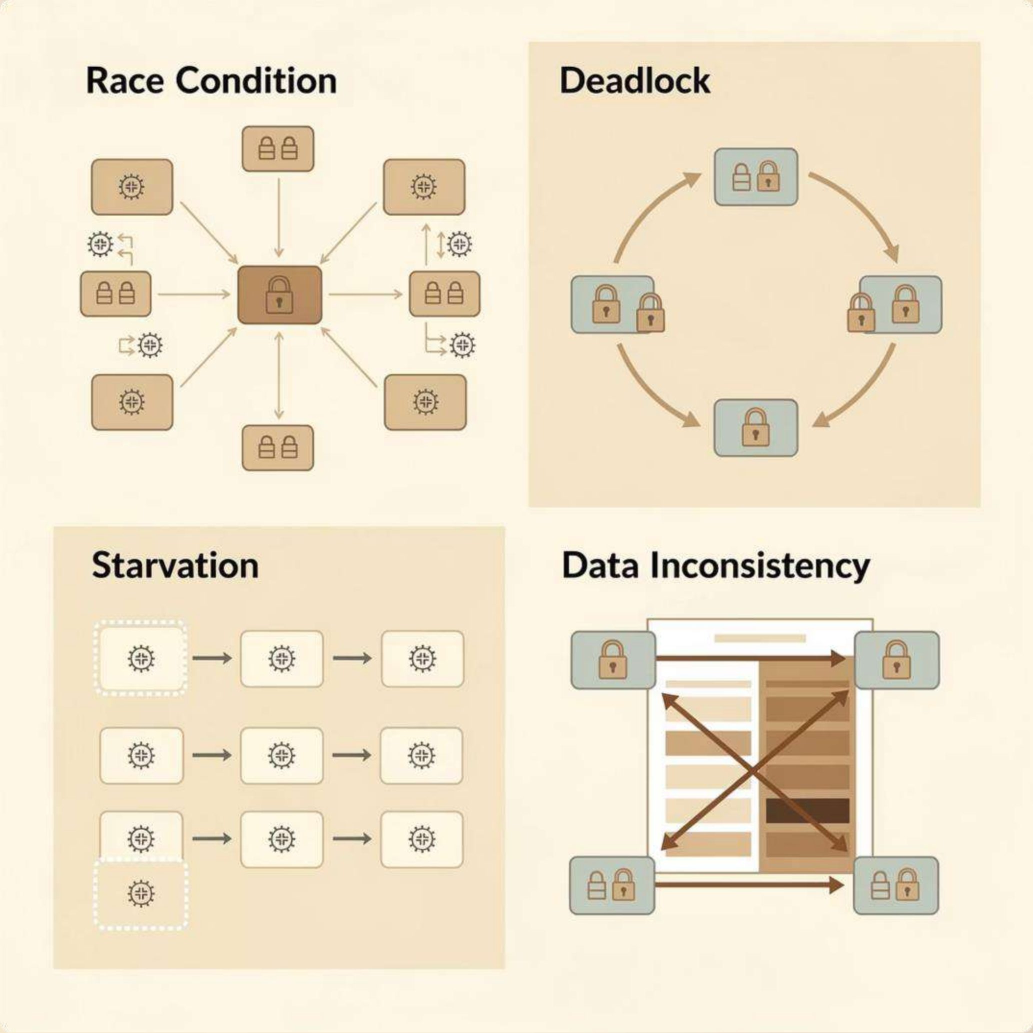
Execution impacts other processes

- Shares resources
- Requires coordination
- Needs synchronization

⚠ CHALLENGES

Concurrent Execution Problems

Multiple processes sharing resources face critical coordination challenges:



CRITICAL SECTION

The Critical Section Problem

> A critical section is code where shared resources are accessed. Only one process should execute this section at any time.

Mutual Exclusion

Only one process in critical section at a time

Progress

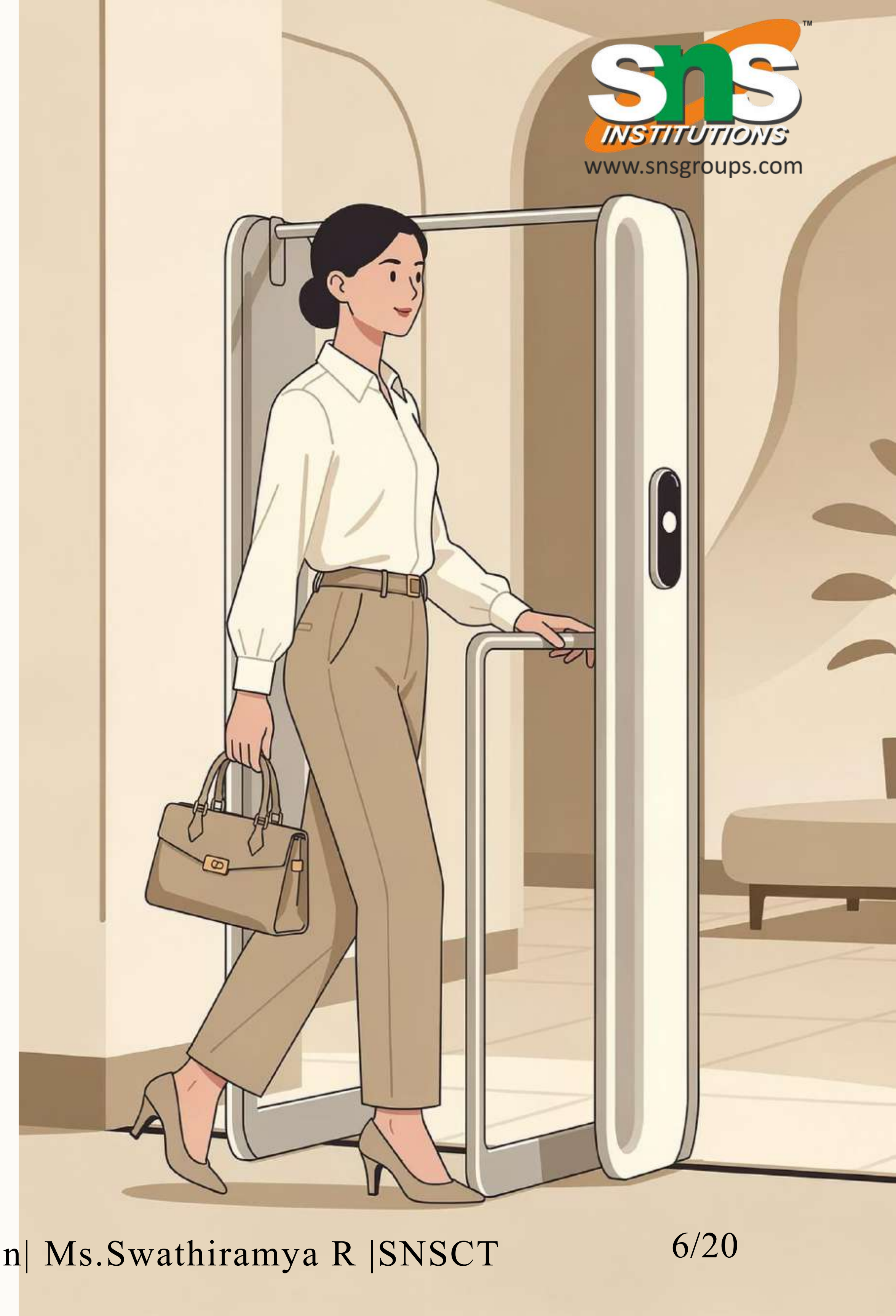
Process selection occurs without unnecessary delay

Bounded Waiting

Every process gets fair chance to enter

CHAPTER 2

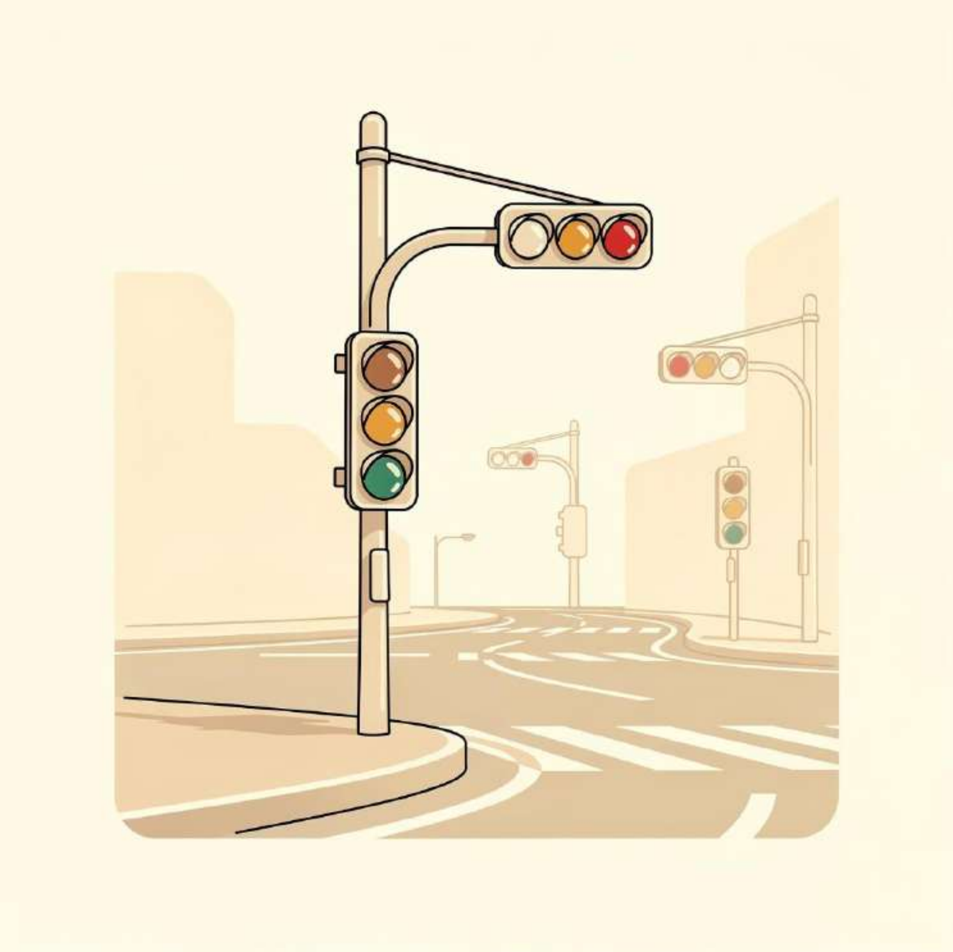
Synchronization Mechanisms



Mutual Exclusion

Ensures only one process or thread accesses a shared resource at a time, preventing conflicts and maintaining data consistency across concurrent operations.

Semaphores



Counter-Based Resource Control

Semaphores use an integer counter to manage access to shared resources.

- **wait()** operation decreases counter
- **signal()** operation increases counter
- Blocks processes when counter reaches zero
- Allows access based on availability

Monitors

Shared Data

Encapsulates resources and variables

Operations

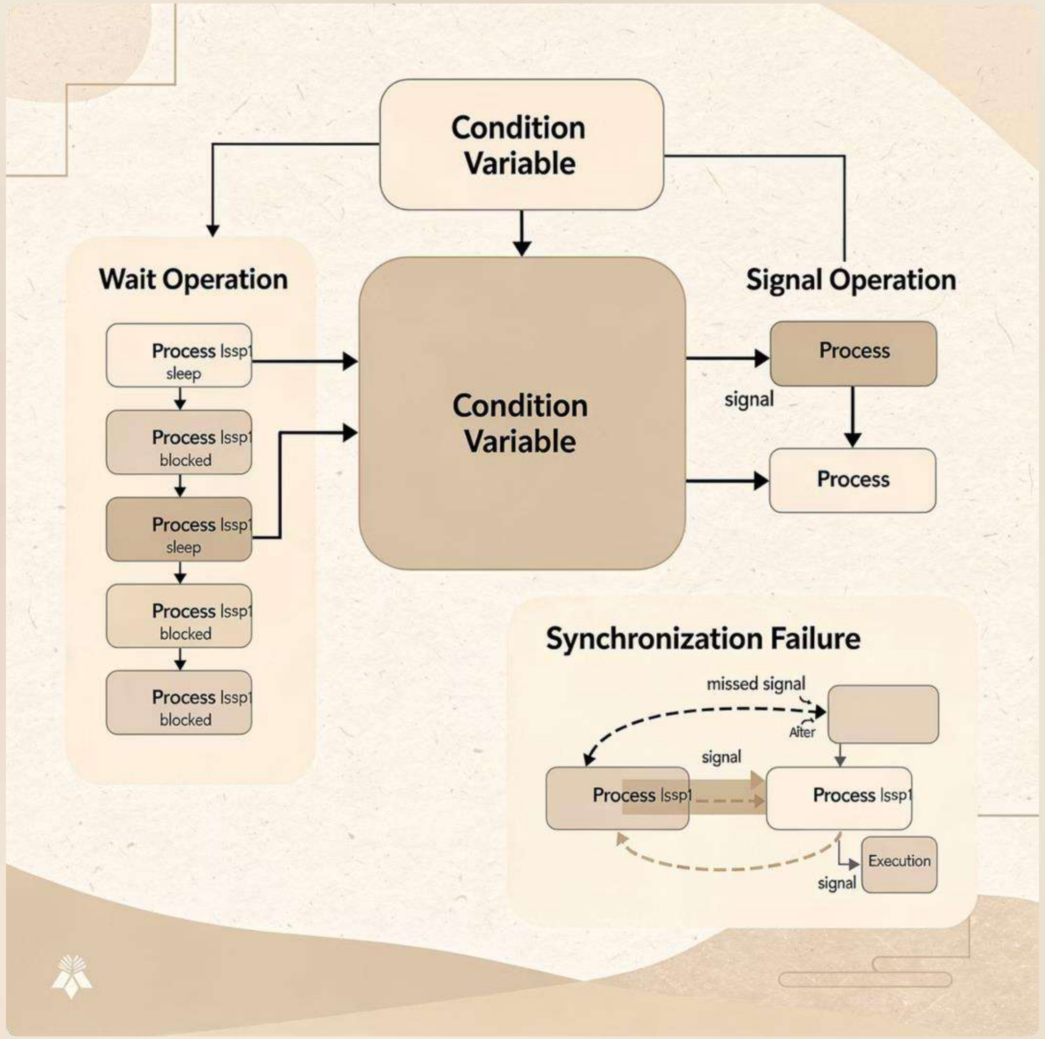
Procedures to access data

Automatic Protection

One process executes at a time

High-level synchronization construct combining data and operations, automatically enforcing mutual exclusion without manual locking.

Condition Variables



Wait & Signal Pattern

Processes wait until specific conditions become true, then continue execution after receiving signals.

- Blocks until condition met
- Releases lock while waiting
- Wakes up on signal notification

Synchronization Failures

- Race Conditions → unpredictable results
- Deadlocks → system freeze
- Starvation → unfair scheduling
- Priority Inversion → scheduling chaos

REAL-WORLD EXAMPLE

Online Ticket Booking System

Two users simultaneously attempt to book the same seat:

❌ Without Synchronization

Both users read "Seat A5 available"

Both confirm booking simultaneously

Result: Double booking, data corruption, angry customers

✅ With Synchronization

First user locks seat A5

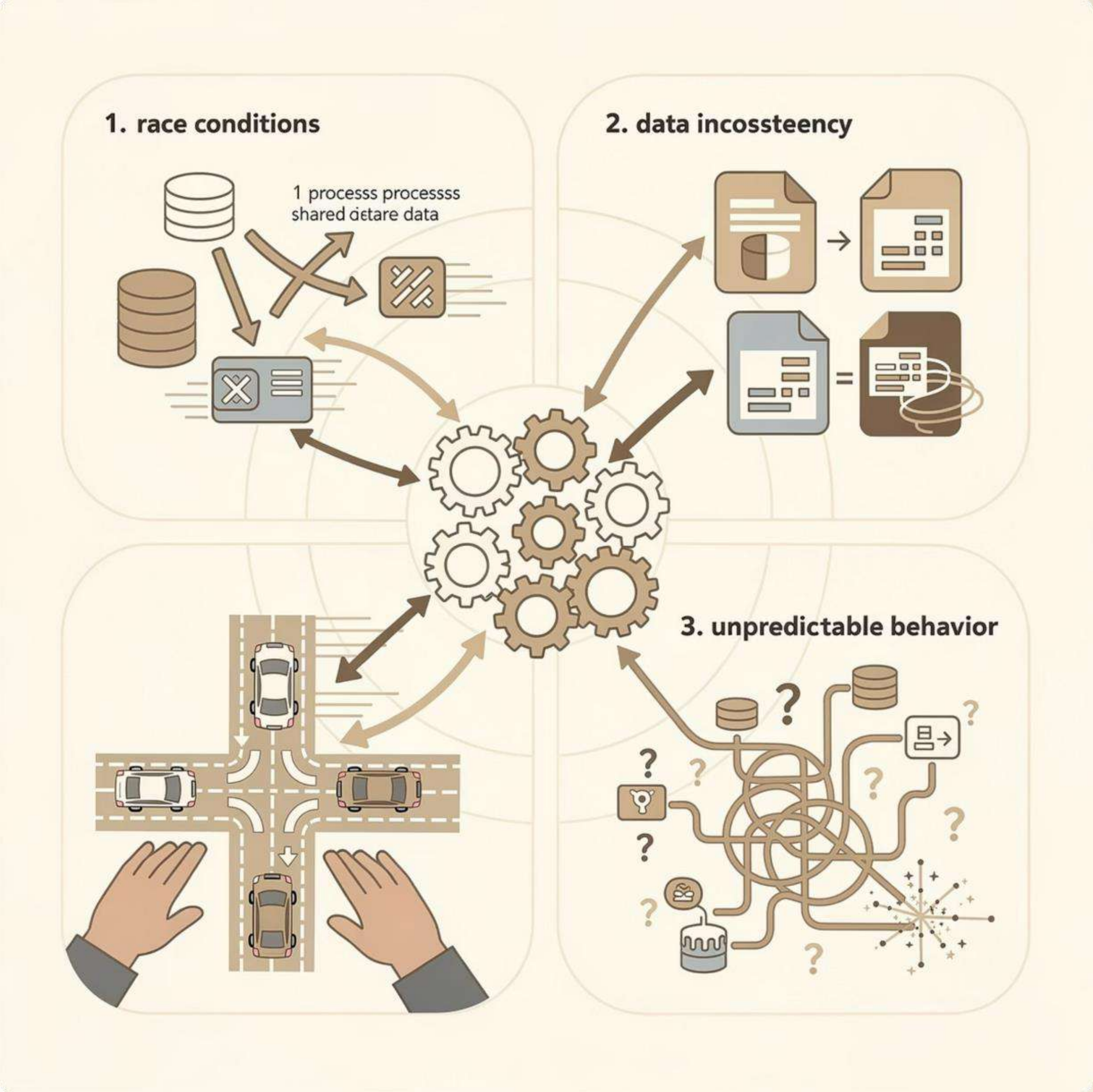
Second user sees seat unavailable

Result: Seat allocated to one user only, system integrity maintained

PROBLEM STATEMENT

The Core Challenge

> How can an operating system allow multiple processes to run concurrently while ensuring safe, consistent, and conflict-free access to shared resources?



Race conditions cause data corruption

Unsynchronized access leads to unpredictable state changes

Inconsistent data threatens system reliability

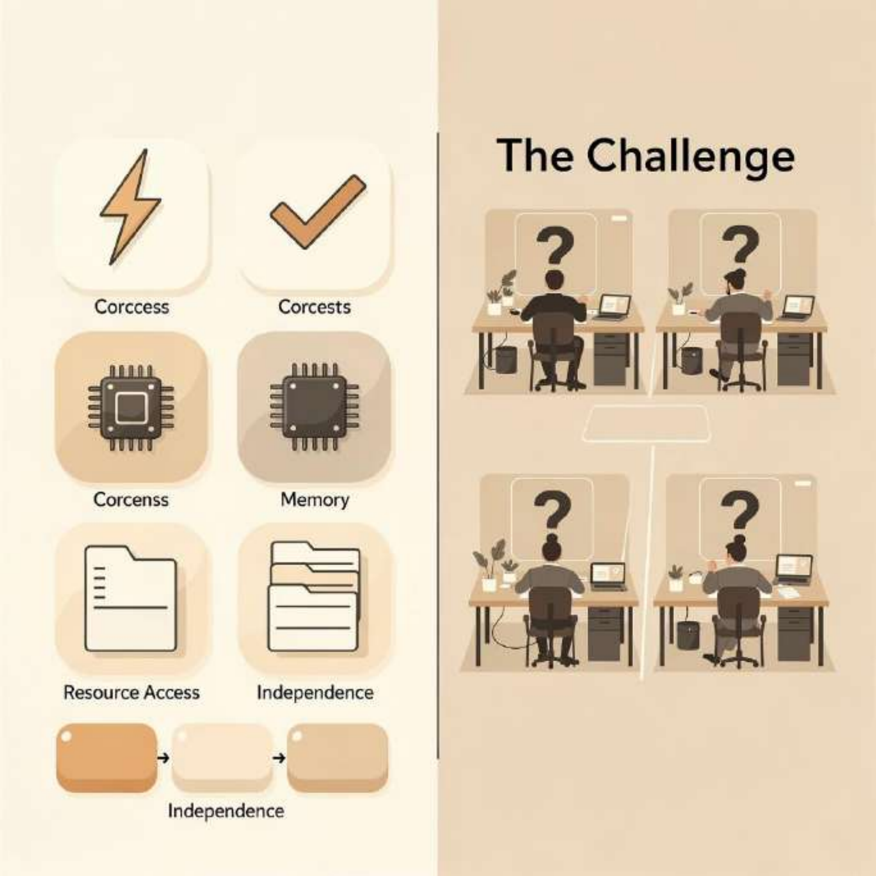
Shared resources modified without coordination produce incorrect results

Unpredictable behavior breaks applications

Non-deterministic execution makes debugging nearly impossible

DESIGN THINKING STEP 1

Empathy — Understanding Processes



What Processes Want

- Fast, efficient execution
- Correct, reliable results
- Access to CPU, memory, files
- Independence from other processes

The Challenge

Processes operate independently without awareness of each other's actions or timing.

DESIGN THINKING STEP 3

Define — Core Issues Identified

Multiple simultaneous entries

Processes enter critical sections at the same time

Incorrect data modification

Shared data gets corrupted without coordination

No execution control

Missing mechanism to manage access order

01

Mutual Exclusion

02

Progress

03

Bounded Waiting

DESIGN THINKING STEP 4

Ideate — Generating Solutions

The OS introduces synchronization mechanisms to control critical section access:

Mutex Locks
0
1
Semaphores
03
wait(|)
signal(|)
Monitors
Spinlocks

DESIGN THINKING STEP 5

Prototype — Implementation

Operating systems implement synchronization tools as control gates:

1 Mutex

Binary lock — only one process enters critical section

2 Semaphore

Counter-based mechanism with wait() and signal() operations

3 Monitor

High-level abstraction automatically enforcing mutual exclusion

Test — Verification Results

✅ Success Metrics

- Single process in critical section
- Data consistency maintained
- Race conditions eliminated
- Predictable system behavior

Real-Time Test: Online Banking

Scenario	Without Sync	With Sync
Two users withdraw simultaneously from same account	Balance corrupted ❌	Correct balance ✅

The Shared Notebook Puzzle

> Two students, A and B, write marks in the same notebook:

A reads current total: **50**

B reads current total: **50**

A adds 10, writes **60**

B adds 20, writes **70**

80

Expected Total
Correct result: 50 + 10 + 20

70

Actual Total
Incorrect result in notebook

❓ Questions:

1. What OS problem is illustrated here?
2. Which synchronization concept can prevent this issue?

Hint: Think critical section and race conditions

Solving the Shared Notebook Puzzle

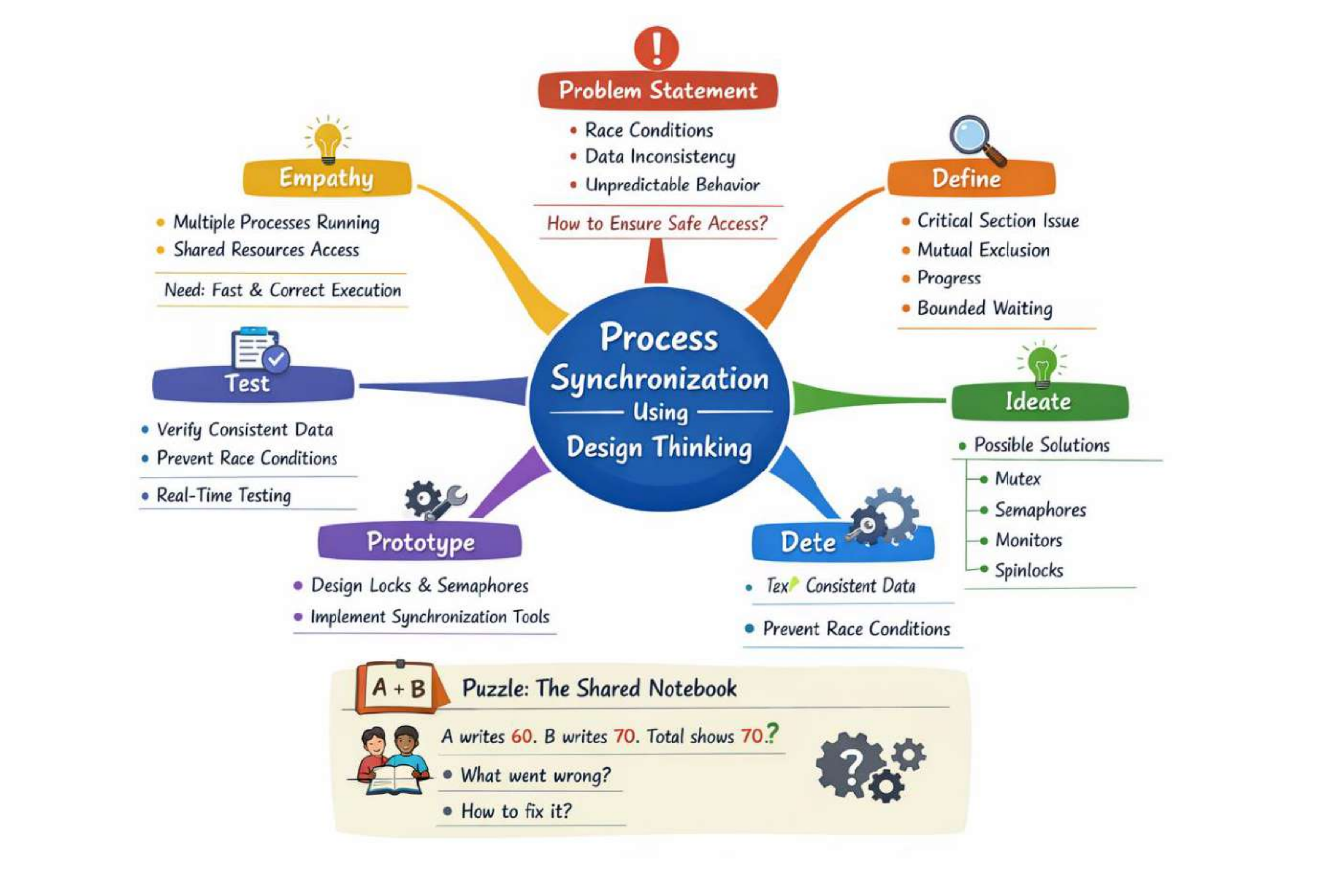
The OS Problem: Race Condition

This highlights a **race condition** within a **critical section**. Both students (processes) read the total simultaneously, causing one update to overwrite the other and resulting in an incorrect final value.

The Synchronization Solution: Mutual Exclusion

Prevent this with **mutual exclusion** using **Semaphores** or **Monitors**. This ensures only one process accesses the shared notebook at a time, guaranteeing data consistency.

- Student A acquires a lock on the notebook.
- A reads 50, adds 10, writes 60, then releases the lock.
- Student B waits for the lock, then acquires it.
- B reads 60, adds 20, writes 80, then releases the lock.
- The final, correct total is 80.



Process Synchronization Using Design Thinking

Problem Statement

- Race Conditions
- Data Inconsistency
- Unpredictable Behavior

How to Ensure Safe Access?

Empathy

- Multiple Processes Running
- Shared Resources Access

Need: Fast & Correct Execution

Define

- Critical Section Issue
- Mutual Exclusion
- Progress
- Bounded Waiting

Ideate

- Possible Solutions
  - Mutex
  - Semaphores
  - Monitors
  - Spinlocks

Dete

- Tex Consistent Data
- Prevent Race Conditions

Prototype

- Design Locks & Semaphores
- Implement Synchronization Tools

Test

- Verify Consistent Data
- Prevent Race Conditions
- Real-Time Testing

A + B Puzzle: The Shared Notebook

A writes 60. B writes 70. Total shows 70?

- What went wrong?
- How to fix it?

Thank You