SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution

Coimbatore-35

Department of Computer Science and Engineering

23CST206–OPERATING SYSTEMS AND VIRTUALIZATION

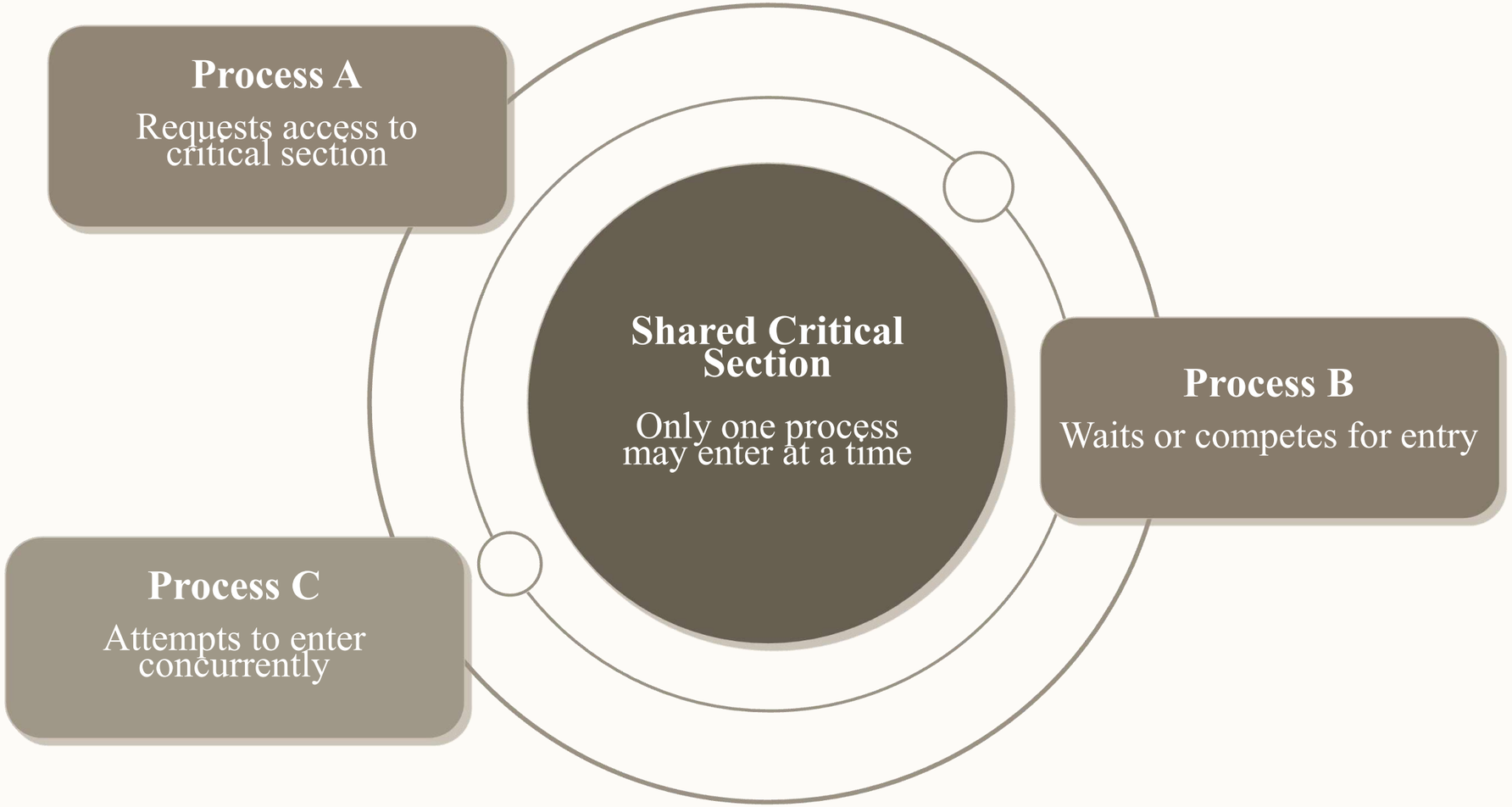
B.E- CSE /IV SEMESTER

UNIT – II PROCESS MANAGEMENT

Topic 6: The Critical-Section Problem

The Critical-Section Problem

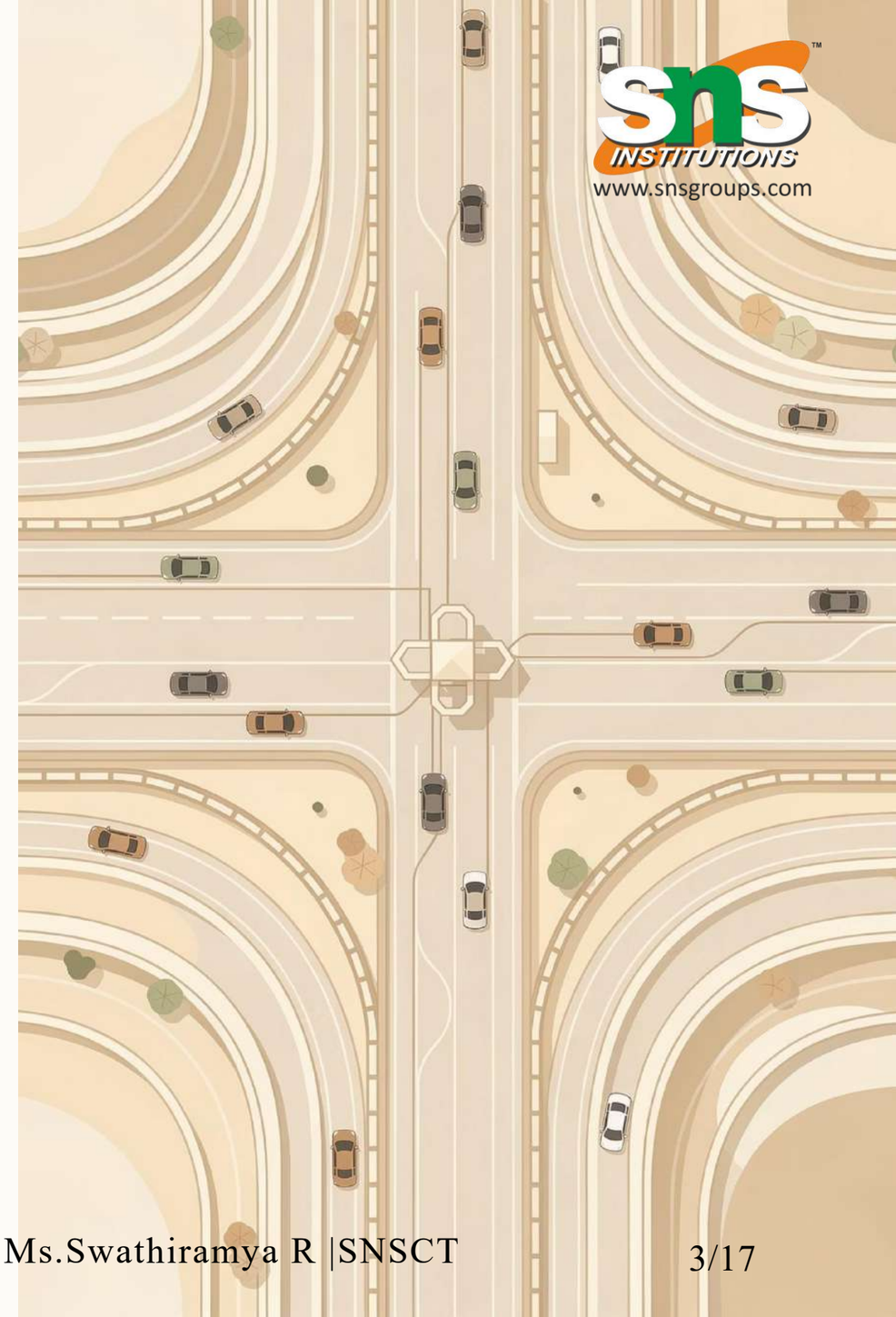
Multiple processes often need to access shared resources within a 'critical section'. The challenge is to ensure that only one process can enter this section at a time, preventing race conditions and maintaining data consistency.



The Critical-Section Problem

The Challenge

Ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section that accesses the same shared resource.



Structure of a Process

Each process is divided into four parts:

Entry Section

Code that requests permission to enter the critical section. The process requests permission to enter the critical section.

Critical Section

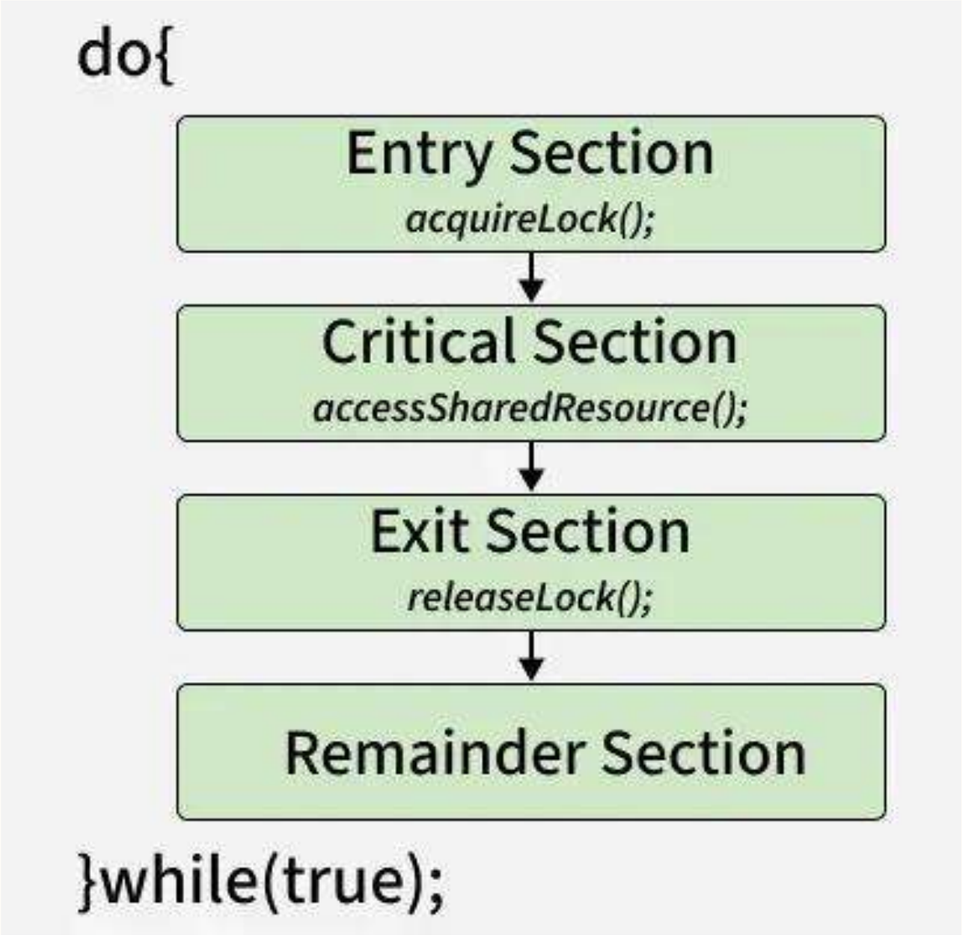
Code that accesses shared resources

Remainder Section

Remaining code of the process

Exit Section

Code that releases the shared resource



Requirements of Critical Section Solutions

A correct solution to the critical-section problem must satisfy three conditions:

1. Mutual Exclusion

Only one process can be in the critical section at a time.

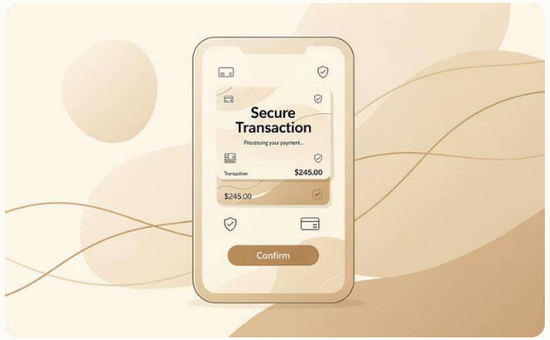
2. Progress

- If no process is in the critical section, and some processes want to enter, the choice of who enters next should not be postponed indefinitely.

3. Bounded Waiting

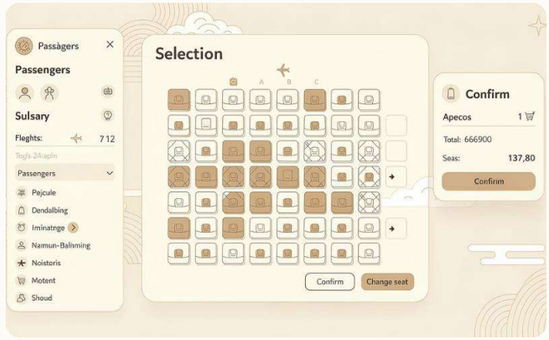
- There must be a limit on how long a process waits before it gets a chance to enter the critical section.
- Prevents starvation, where one process is repeatedly bypassed while others get to execute.

Examples of Critical Sections in Real-World Applications



Banking System

Critical Section: Updating an account balance during a deposit or withdrawal.



Ticket Booking System

Critical Section: Reserving the last available seat.

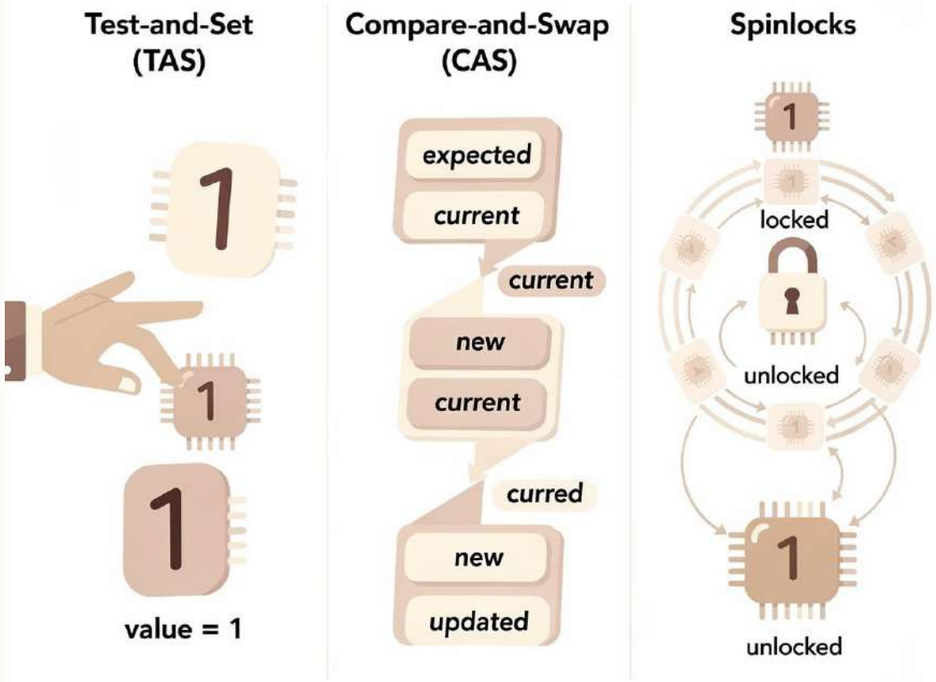
Print Spooler

Critical Section: Sending print jobs to the printer queue.

File Editing in Shared Documents

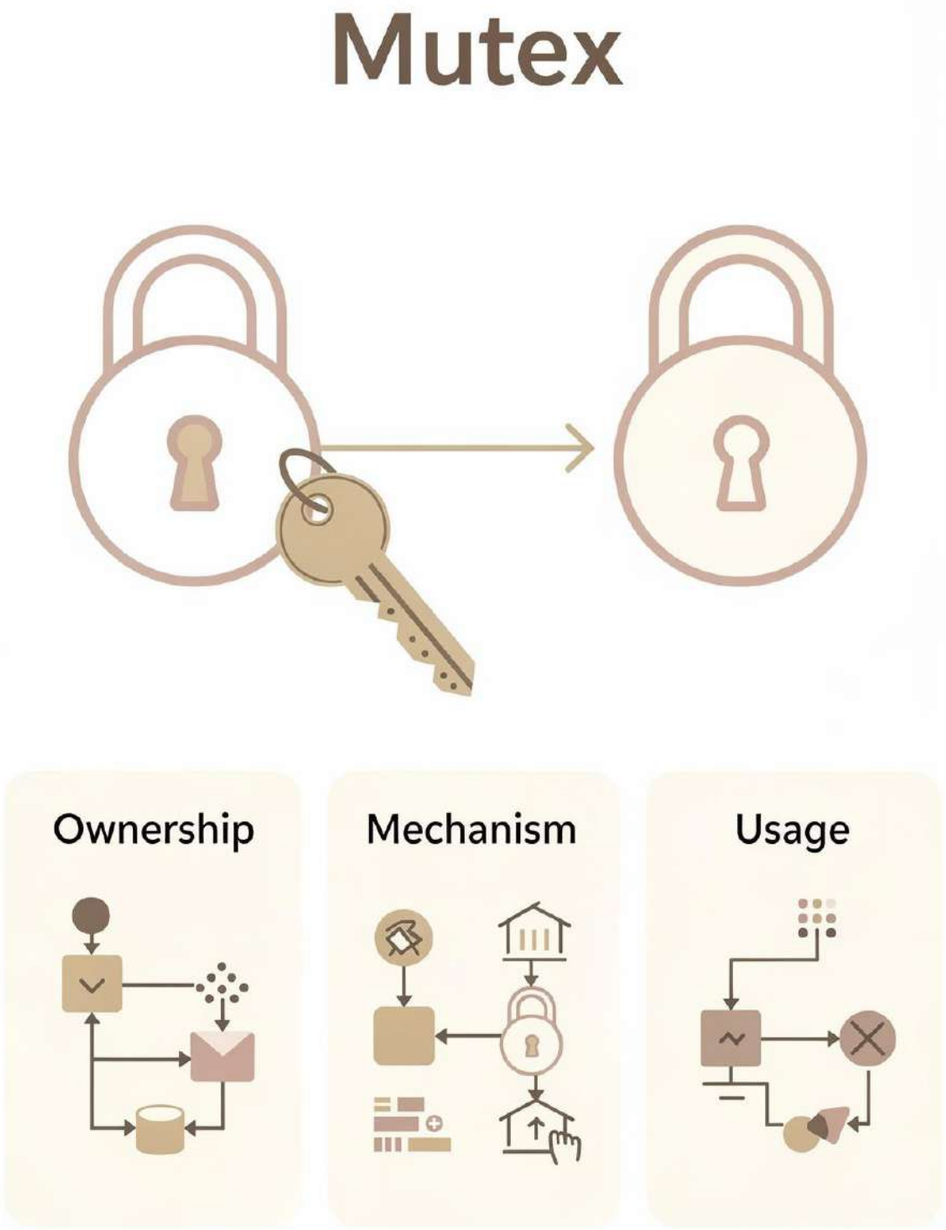
Critical Section: Saving or writing to the shared document.

Synchronization Hardware in Operating Systems



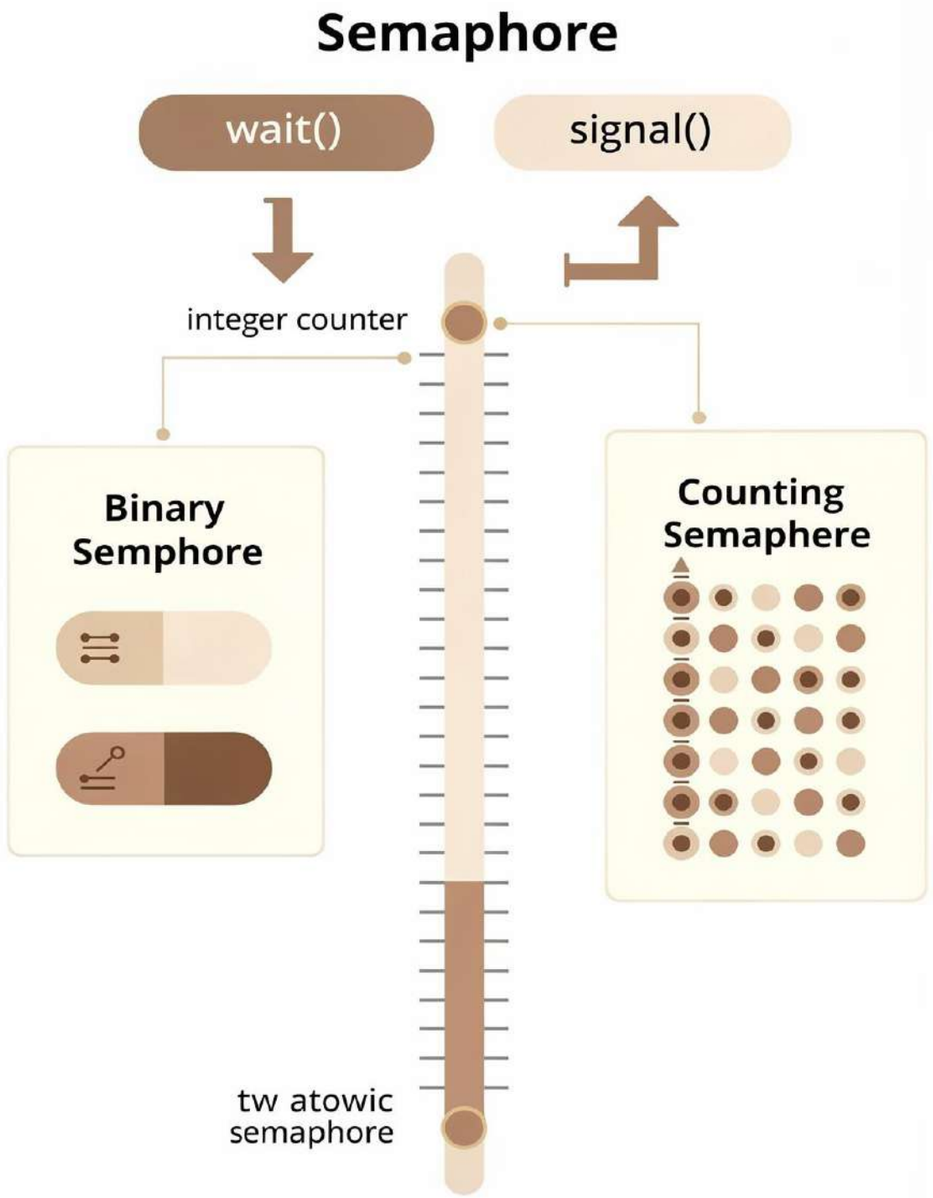
Synchronization Hardware

Modern CPUs provide specialized, atomic hardware instructions that simplify the solution to the critical section problem. These instructions are uninterruptible.



Mutex (Mutual Exclusion)

A Mutex is a higher-level software abstraction, often built on hardware primitives, specifically designed for exclusive access to a single resource.



Semaphore

A Semaphore is a signaling mechanism represented by an integer variable that manages access to multiple instances of a resource.

- Operations
- Binary Semaphore
- Counting Semaphore

Problem Statement

In modern operating systems, multiple processes or threads often execute concurrently and share common resources such as memory, files, or devices. Uncontrolled access to these shared resources may lead to race conditions, data inconsistency, deadlock, or starvation.

Safe Access

Ensure safe access to shared resources

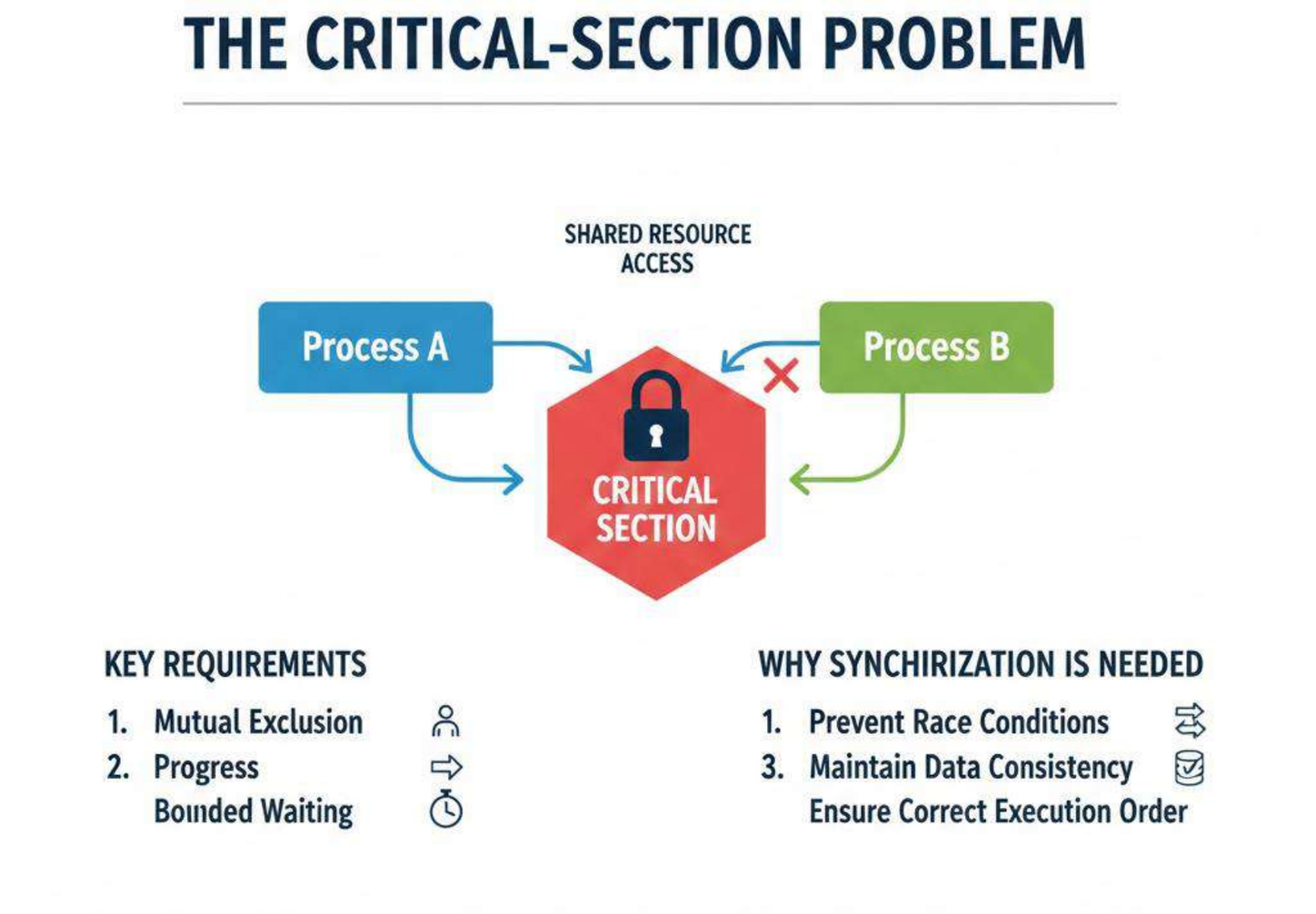
Fairness

Maintain fairness among processes

Efficiency

Optimize CPU utilization

DEFINE (Understand the Problem)



STEP 2

IDEATE (Generate Possible Solutions)

To solve the critical-section problem, several synchronization ideas were explored:

1

Idea 1: Software-only Solutions

- Peterson's Algorithm
- Bakery Algorithm

❌ Not efficient or scalable on multiprocessor systems

2

Idea 2: Synchronization Hardware

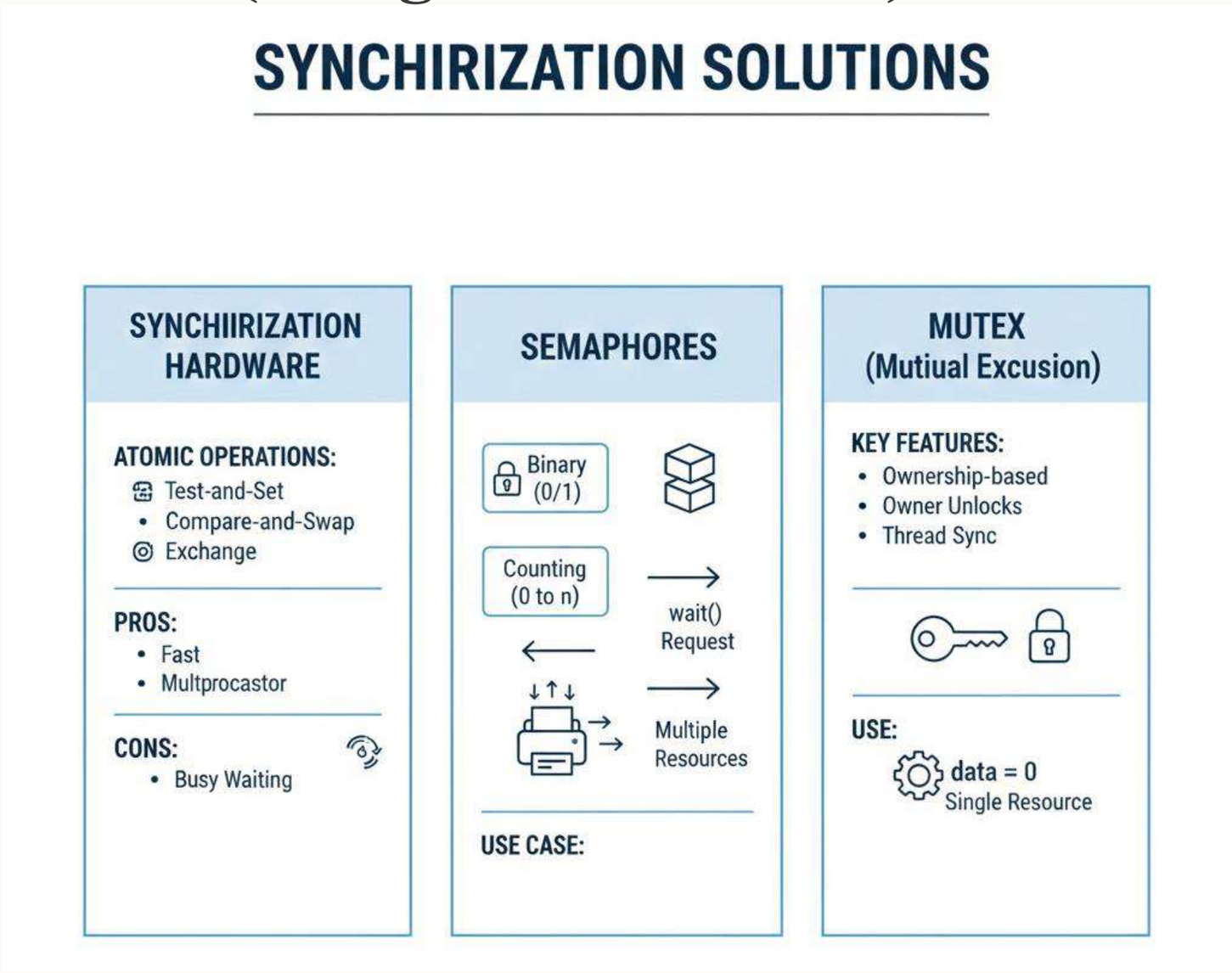
- Use atomic CPU instructions to control access
- Examples: Test-and-Set, Compare-and-Swap

3

Idea 3: High-Level Synchronization Tools

- Semaphores for controlled access
- Mutex locks for strict mutual exclusion

PROTOTYPE (Design the Solution)



TEST (Evaluate the Solution)

EVALUATE THE SOLUTION

CRITERIA

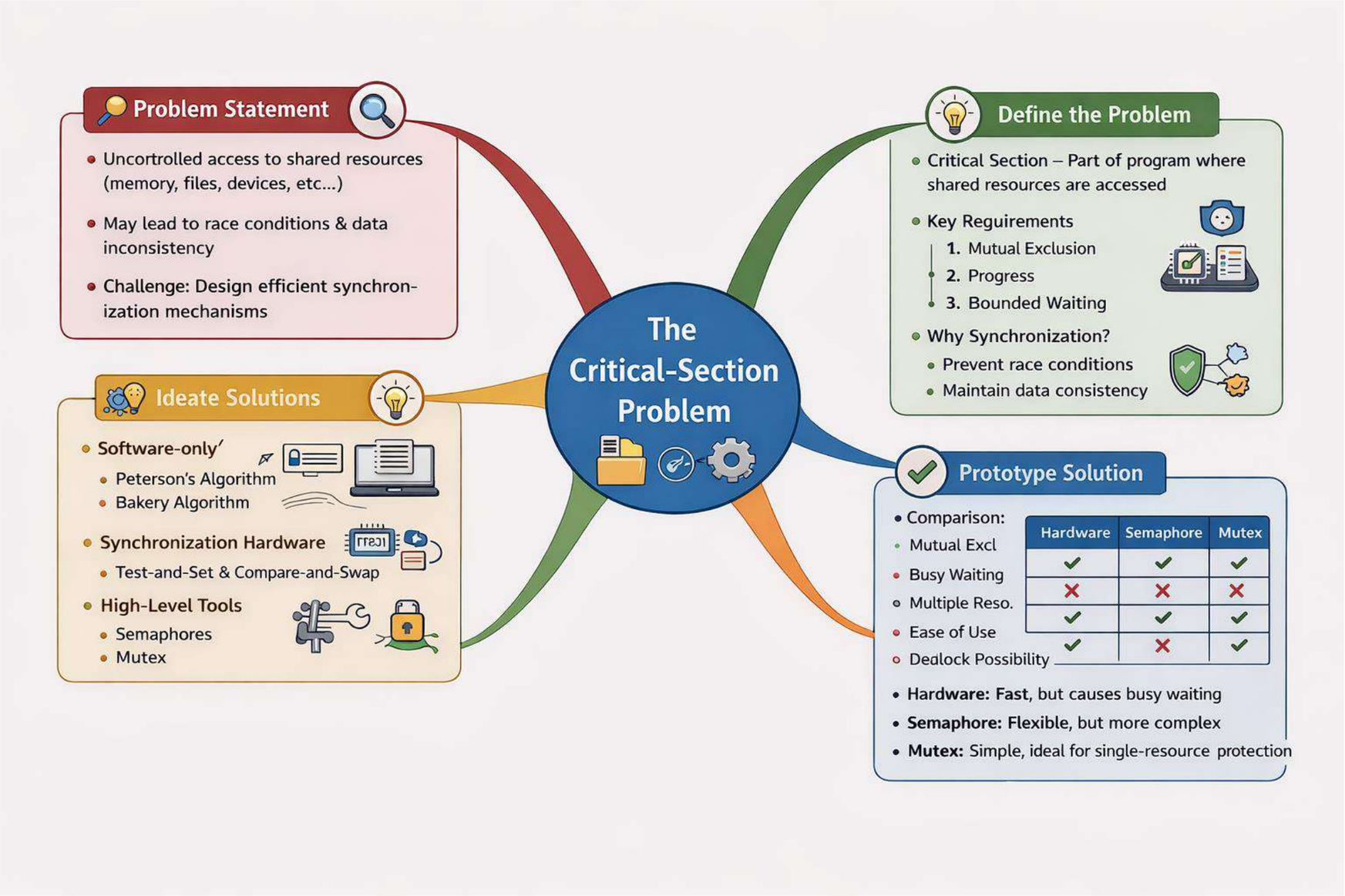
		HARDWARE	SEAMAPHORE	MUTEX
1	Mutual Exclusion	✓	✓	✓
2	Busy Waiting	✓	✗	✗
3	Multiple Resources	✓	✓	✓
4	Ease of Use			✓
5	Deadlock Possibility	Low	Medium	High

4 TEST
Evaluate the Solution

CONCLUSION

Hardware: Fast but Infficient (Busy Waiting)
Flexible but Complex

Mutex: Simple & Ideal for Single Resources


The Critical-Section Problem
Problem Statement
Uncontrolled access to shared resources (memory, files, devices, etc...)
May lead to race conditions & data inconsistency
Challenge: Design efficient synchronization mechanisms
Define the Problem
Critical Section – Part of program where shared resources are accessed
Key Requirements
1. Mutual Exclusion
2. Progress
3. Bounded Waiting
Why Synchronization?
Prevent race conditions
Maintain data consistency
Ideate Solutions
Software-only
Peterson's Algorithm
Bakery Algorithm
Synchronization Hardware
Test-and-Set & Compare-and-Swap
High-Level Tools
Semaphores
Mutex
Prototype Solution
Comparison:
Mutual Excl
Busy Waiting
Multiple Reso.
Ease of Use
Deadlock Possibility
Hardware
Semaphore
Mutex
Hardware: Fast, but causes busy waiting
Semaphore: Flexible, but more complex
Mutex: Simple, ideal for single-resource protection

🧠 Puzzle: The Shared Notebook Problem

Three students — A, B, and C — share a single laboratory notebook to record experiment results.

Rules:

1. Only one student can write in the notebook at a time.
2. A student must finish writing before another can start.
3. No student should wait forever if they request the notebook.

Task 1

Identify the critical section in this scenario.

Task 2

Suggest whether a semaphore or mutex is more suitable.


Thank
you