The Sleeping TA Problem

A Classic Concurrency Challenge

The Challenge: The Rules of the Office

The TA

Naps when idle, helps when woken, and checks the hallway chairs after finishing with a student.

The Students

Work, then seek help. Wait in a chair if busy. Leave if all chairs are full and try again later. 😤

The Chairs

Only 3 available. This is the critical resource students compete for just to be able to wait.

The Solution: A "Rules Engine"

1. The "Waiting Chairs"

Analogy: 3 tokens on a hook.
Tool: `SemaphoreSlim(3)`

2. The "TA's Office Door"

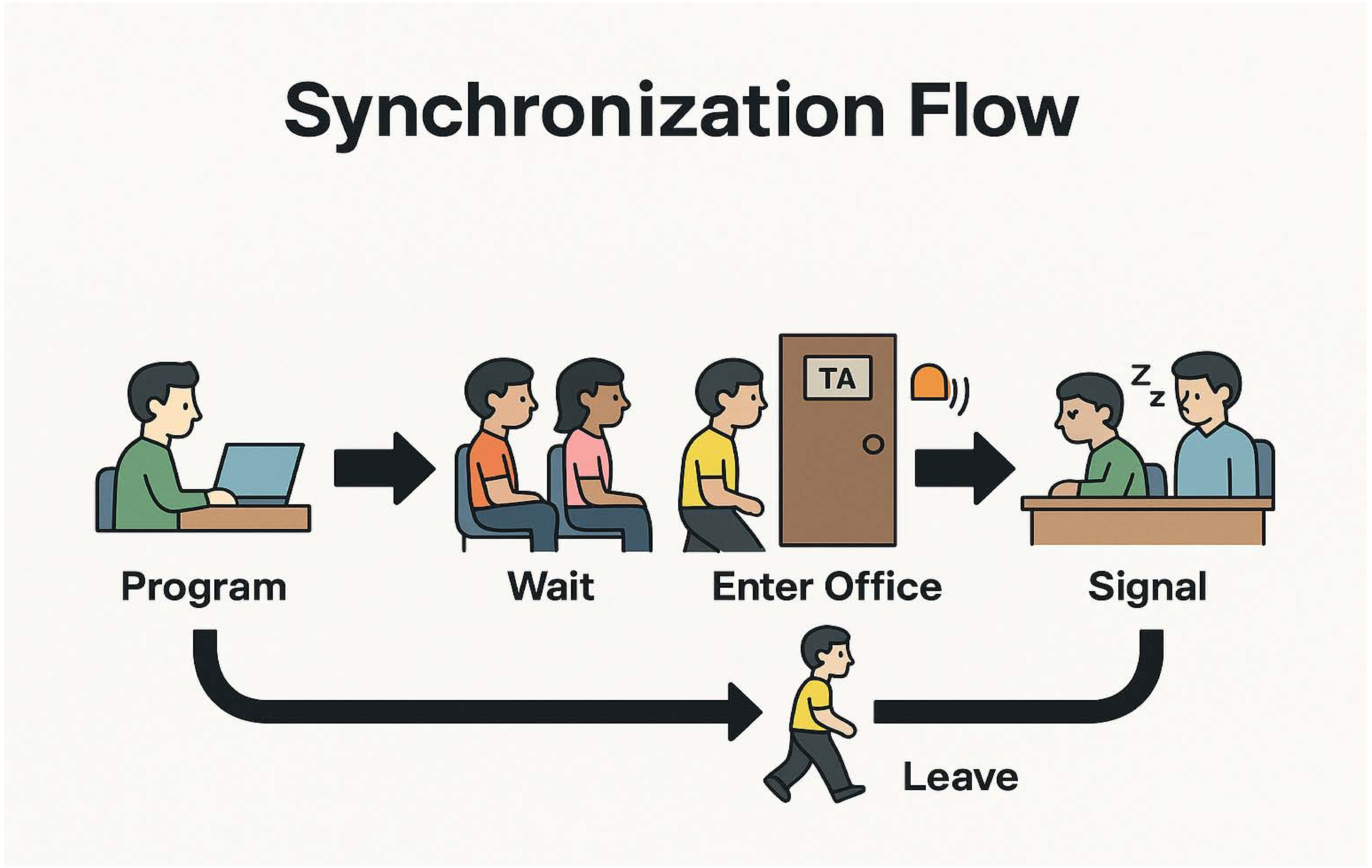
Analogy: A single door key.
Tool: `SemaphoreSlim(1)`

3. The "TA's Nap"

Analogy: A doorbell.
Tool: `SemaphoreSlim(0)`

4. The "Session Over" Signal

Analogy: A tap on the shoulder.
Tool: `SemaphoreSlim(0)`



Code: Defining the "Managers"

C# SemaphoreSlim

We create a "manager" for each rule. A semaphore is a thread-safe counter.

- `_hallwayChairs`: Starts at **3**. Manages the 3 chairs.
- `_taMutex`: Starts at **1**. Acts as a "key" for the office (a mutex).
- `_studentsReady`: Starts at **0**. The TA "sleeps" (Waits) on this.
- `_taDoneHelping`: Starts at **0**. The Student "sleeps" (Waits) on this.

```
// 1. The Hallway Chairs (3 tokens)
private static SemaphoreSlim _hallwayChairs =
    new SemaphoreSlim(3, 3);

// 2. The TA's Office (1 key)
private static SemaphoreSlim _taMutex =
    new SemaphoreSlim(1, 1);

// 3. The TA's "Doorbell" (TA sleeps on this)
private static SemaphoreSlim _studentsReady =
    new SemaphoreSlim(0);

// 4. The "Session Over" Signal
private static SemaphoreSlim _taDoneHelping =
    new SemaphoreSlim(0);
```

Code: The TA's Logic

TaProcess()

The TA runs in an infinite loop.

1. **Sleep:** Calls _studentsReady.WaitAsync(). This "blocks" the TA, making them "nap" until a student signals (Releases).
2. **Help:** Simulates work with a Task.Delay.
3. **Signal:** Calls _taDoneHelping.Release(). This "wakes up" the waiting student, telling them the session is over.

```csharp
private static async Task TaProcess()
{
    while (true)
    {
        Console.WriteLine("TA is napping... ZzzZzz...");

        // --- 1. Sleep ---
        await _studentsReady.WaitAsync();

        // --- 2. Wake Up and Help ---
        Console.WriteLine("TA is awake and helping.");
        await Task.Delay(Rng.Next(3000, 6000));
        Console.WriteLine("TA is finished helping.");

        // --- 3. Signal Session Over ---
        _taDoneHelping.Release();
    }
}
```

Code: Student Logic (Part 1/2)

`StudentProcess()` - The Big Check

The student first "programs" (a delay).

Then, they try to get a chair using `_hallwayChairs.Wait(0)`.

- `Wait(0)` is a **non-blocking** call. It means "try to decrement the counter, but don't wait if it's already 0."
- If it returns `false` (no chairs), the student gives up, logs a message, and loops back to programming.

```
private static async Task StudentProcess(int id)
{
    while (true)
    {
        // --- 1. Program ---
        Console.WriteLine($"Student {id} is programming.");
        await Task.Delay(Rng.Next(5000, 15000));

        Console.WriteLine($"Student {id} needs help.");

        // --- 2. Try to Get a Chair ---
        if (_hallwayChairs.Wait(0))
        {
            Console.WriteLine($"Student {id} is waiting...");
            // Got a chair! Continue in Part 2...
            // (Code from next slide goes here)
        }
        else
        {
            // --- 7. No Chairs Available ---
            Console.WriteLine($"Student {id} found no chairs.");
            // Loop back to programming
        }
```

Code: Student Logic (Part 2/2)

Inside the `if` block...

The student has a chair and is now in line for the TA's office.

1. **Wait for TA:** `await _taMutex.WaitAsync()`. The student waits for the "office key."
2. **Enter Office:** Once they have the key, they give up their chair: `_hallwayChairs.Release()`.
3. **Ring Bell:** `_studentsReady.Release()` signals the napping TA.
4. **Wait for Finish:** `await _taDoneHelping.WaitAsync()`. The student "sleeps" until the TA is done.
5. **Leave:** `_taMutex.Release()`. The student returns the "office key".

```
// (Inside the if-block from Part 1)

// --- 4. Wait for TA (Get the key) ---
await _taMutex.WaitAsync();

// --- 5. See the TA ---
Console.WriteLine($"Student {id} is getting help.");

// Leave the hallway chair (return the token)
_hallwayChairs.Release();

// Ring the "doorbell" to wake the TA
_studentsReady.Release();

// Wait for the TA to signal session is over
await _taDoneHelping.WaitAsync();

// --- 6. Leave Office (Return the key) ---
_taMutex.Release();

Console.WriteLine($"Student {id} is done.");
```

Questions?

Thank You