Lecture 27: Synchronization

CS 2110

April 30, 2026

Today's Learning Outcomes

106. Explain the semantics of locks in Java and write code involving synchronization.

107. Describe the conditions that can cause deadlock in a concurrent program and how it can be avoided.

108. Explain how condition variables can be used to synchronize modifications of a shared resource.

Where We Left Off: Race Conditions

Multiple threads access same variables during same time window, and at least one writes to the variable.

Different interleavings of machine instructions lead to different results.

```
class SharedInt { int x = 0; }

static void increment(SharedInt i) {  i.x++; }

public static void main(String[] args) throws ... {
   SharedInt s = new SharedInt(); // shared variable
   Thread t1 = new Thread(() -> increment(s));
   Thread t2 = new Thread(() -> increment(s));

   t1.start();   t2.start();  // start the threads
   t1.join();  t2.join();     // wait for threads to stop

   System.out.println("Final value of s.x: " + s.x);
}
```

Critical Sections

Sometimes, multiple instructions must be executed sequentially to properly update a shared variable without the possibility of another thread messing with that variable.

Mutexes and Synchronized Blocks

Coding Demo: Synchronized Blocks

Disciplined Synchronization

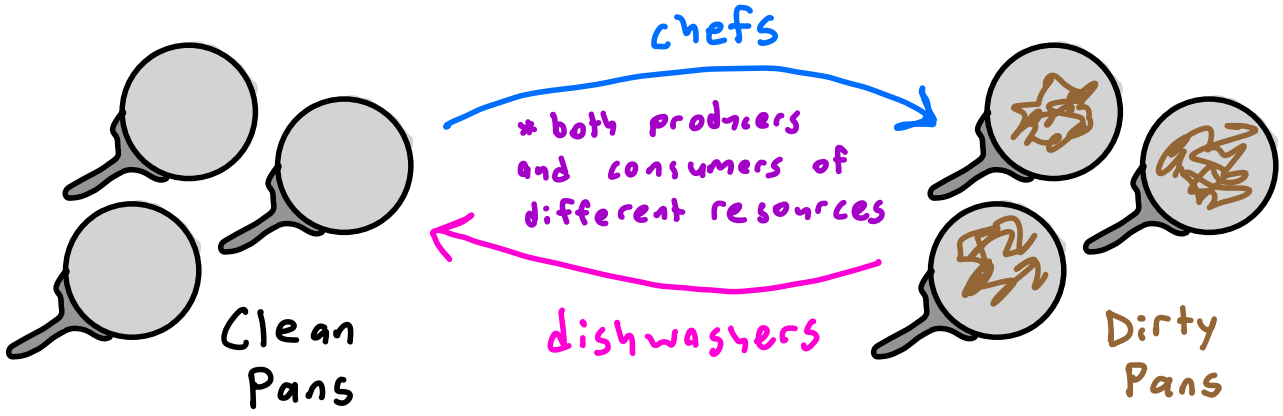
- Synchronization only works when every modification to a shared variable is synchronized

Producer-Consumer Problems

Multi-threading problem for resource processing
- some threads are producers that add resources
- some threads are consumers that remove resources

Ex. - order fulfillment on e-commerce sites
- job scheduling on computing servers

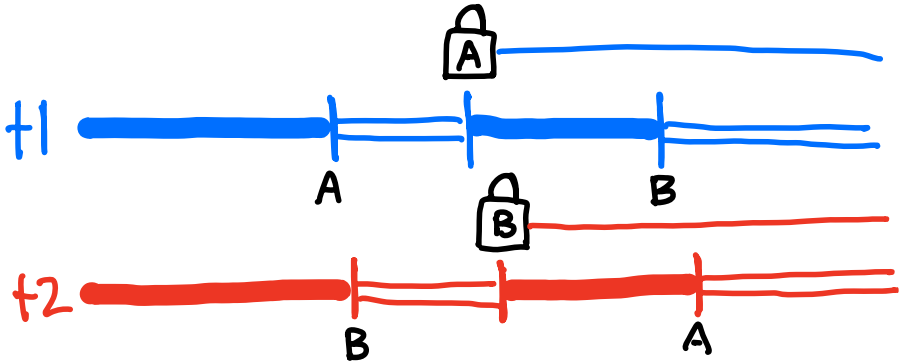
Today: Toy "Kitchen Simulation"



Coding Demo: Multiple Shared Resources

Deadlock

Concurrent code can freeze up when every thread is waiting to acquire a mutex held by another.

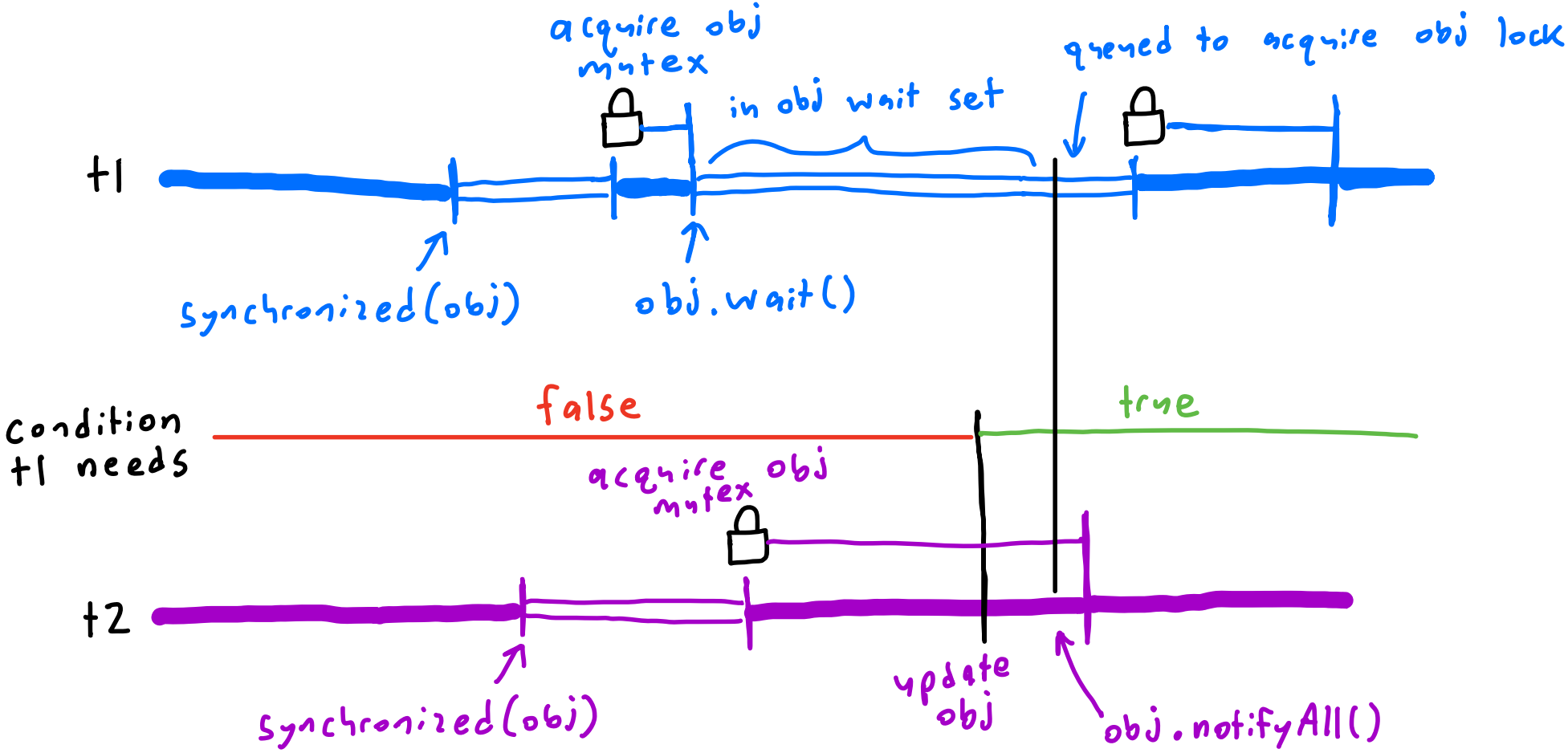


Coordinating Threads

Sometimes, threads need to pause execution to give others time to do prerequisite work.

wait() and notifyAll()

Visualizing Synchronization



Coding Demo: Synchronization

Keys for Proper Synchronization

- Always call wait() inside of a synchronized block for that object. Part of wait() is releasing the mutex
- Always call wait() within a loop checking for the condition
- Always call notifyAll() (not notify()) on a shared object after modifying its state

* Synchronization requires careful attention and discipline. Missing one rule can break everything and this may not be caught during testing.

The Monitor Pattern