

Lecture 26: Concurrency

CS 2110

November 24, 2026

Today's Learning Outcomes

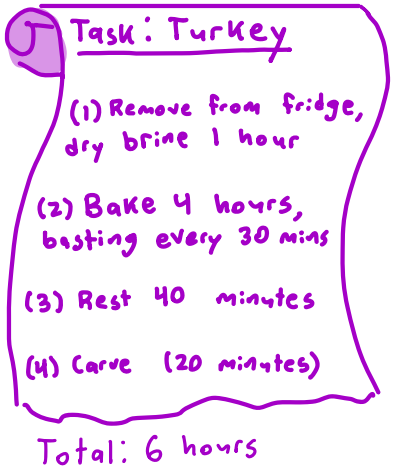
93. Describe advantages and disadvantages of concurrent code.

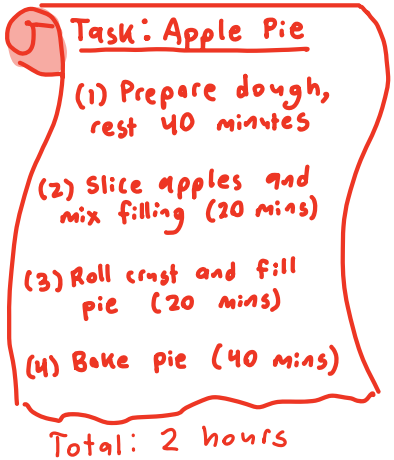
94. Use Java's Thread class to write concurrent code.

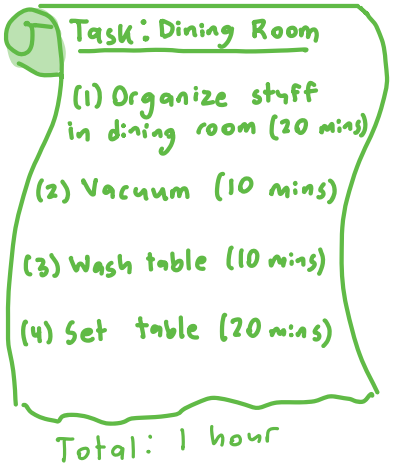
95. Identify race conditions in a given piece of concurrent code and list the possible states the program can be in after it executes.

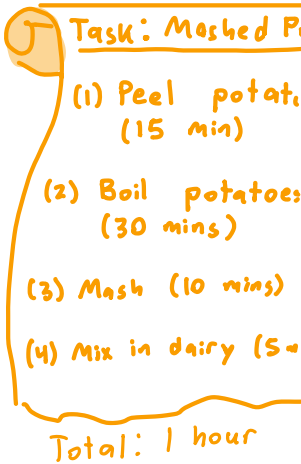
Motivation: Making a Thanksgiving Dinner

Preparing and hosting the meal requires completing a lot of separate tasks:









How can we take care of them efficiently?

Concurrency

= The ability to have multiple independent procedures in progress at the same time.

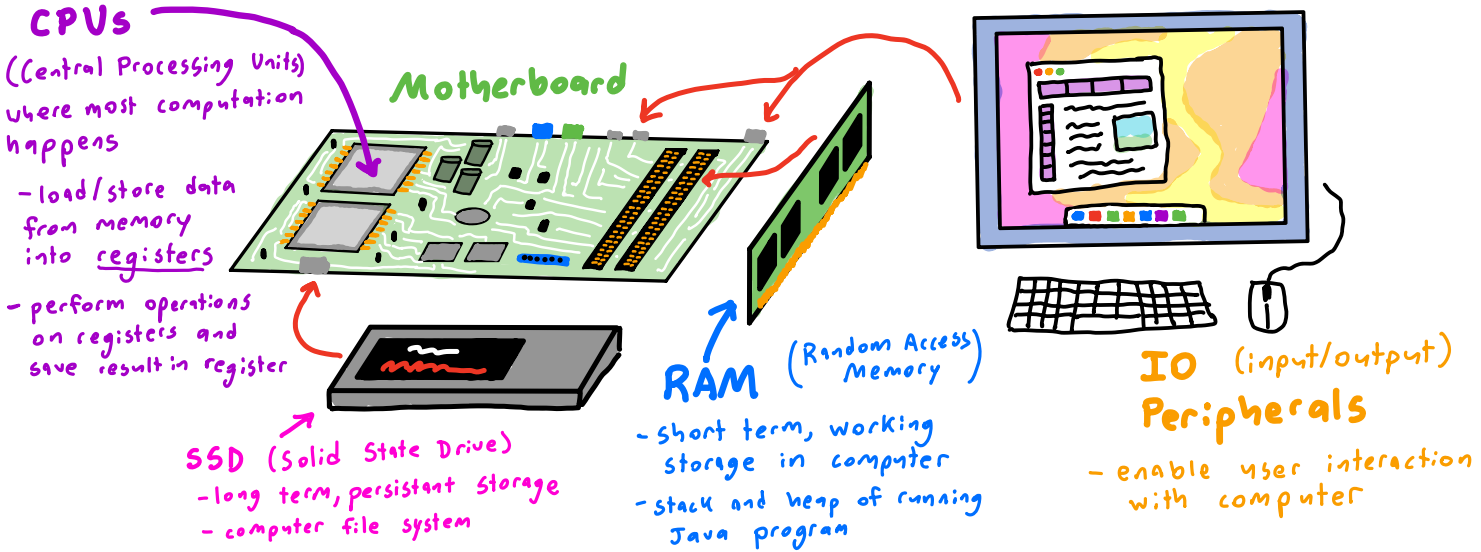
We can achieve concurrency in two different ways:

1. Time Slicing

2. Parallelism

Computer Hardware (High Level)

Various hardware components serve different functions in your computer and must work together to enable computation.



Operating Systems and Processes

A **process** is a running program on your computer.

The **operating system** is the main program that is responsible for managing the execution of all other processes.

Threads

= an abstraction that models a single, sequential execution of instructions that performs work within a process.



Coding Demo: Java's Thread Class



Thread Methods: `run()` and `start()`

Threads are constructed with a `Runnable` argument, a functional interface that models a sequence instructions.

The thread exposes two methods to carry out these instructions:

1. **`run()`**

2. **`start()`**

Thread Methods: `sleep()` and `join()`

Sometimes, we may wish to temporarily pause the execution of instructions on one thread. Java offers many methods for doing this:

`sleep()`

`join()`

Aside: Variable Capture

We often define the `Runnable` parameter inline within the `Thread` constructor using a lambda expression.

The `Runnable.run()` method has no parameters. How can we share/pass information to this new thread?

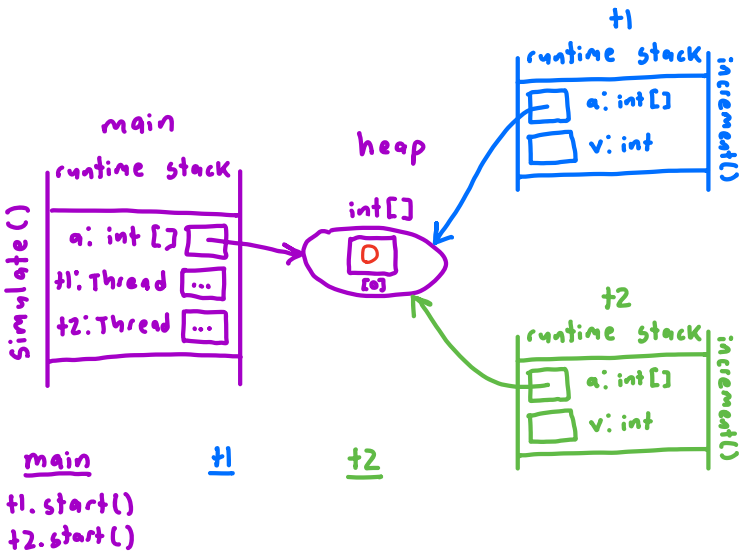
Sharing State Between Threads

```
static void increment(int[] a) {
  int v = a[0];
  a[0] = v + 1;
}

void simulate() throws InterruptedException {
  int[] a = {0}; // shared mutable variable

  Thread t1 = new Thread(() -> increment(a));
  Thread t2 = new Thread(() -> increment(a));

  t1.start(); t2.start();
  t1.join();  t2.join();
  System.out.println(a[0]);
}
```



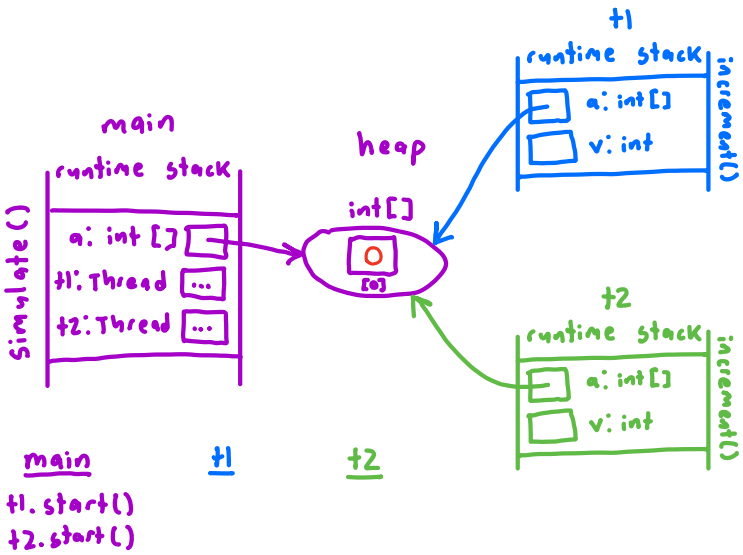
Sharing State Between Threads (v2)

```
static void increment(int[] a) {
  int v = a[0];
  a[0] = v + 1;
}

void simulate() throws InterruptedException {
  int[] a = {0}; // shared mutable variable

  Thread t1 = new Thread(() -> increment(a));
  Thread t2 = new Thread(() -> increment(a));

  t1.start(); t2.start();
  t1.join();  t2.join();
  System.out.println(a[0]);
}
```



Race Conditions

= a phomenon in multi-threaded code where different post-conditions are reached depending on the specific order in which the instructions across multiple threads are executed.

Preview: Synchronization

= the process of coordinating the execution of multiple threads, particularly their access and modification of shared state, within a piece of concurrent code.

Sometimes, we'll need to carefully time (i.e., strategically pause) work on different threads to avoid race conditions and achieve the desired behavior.

We'll see how to get this working when we return next week.

For now, we'll strategically pause so you can enjoy your Thanksgiving!