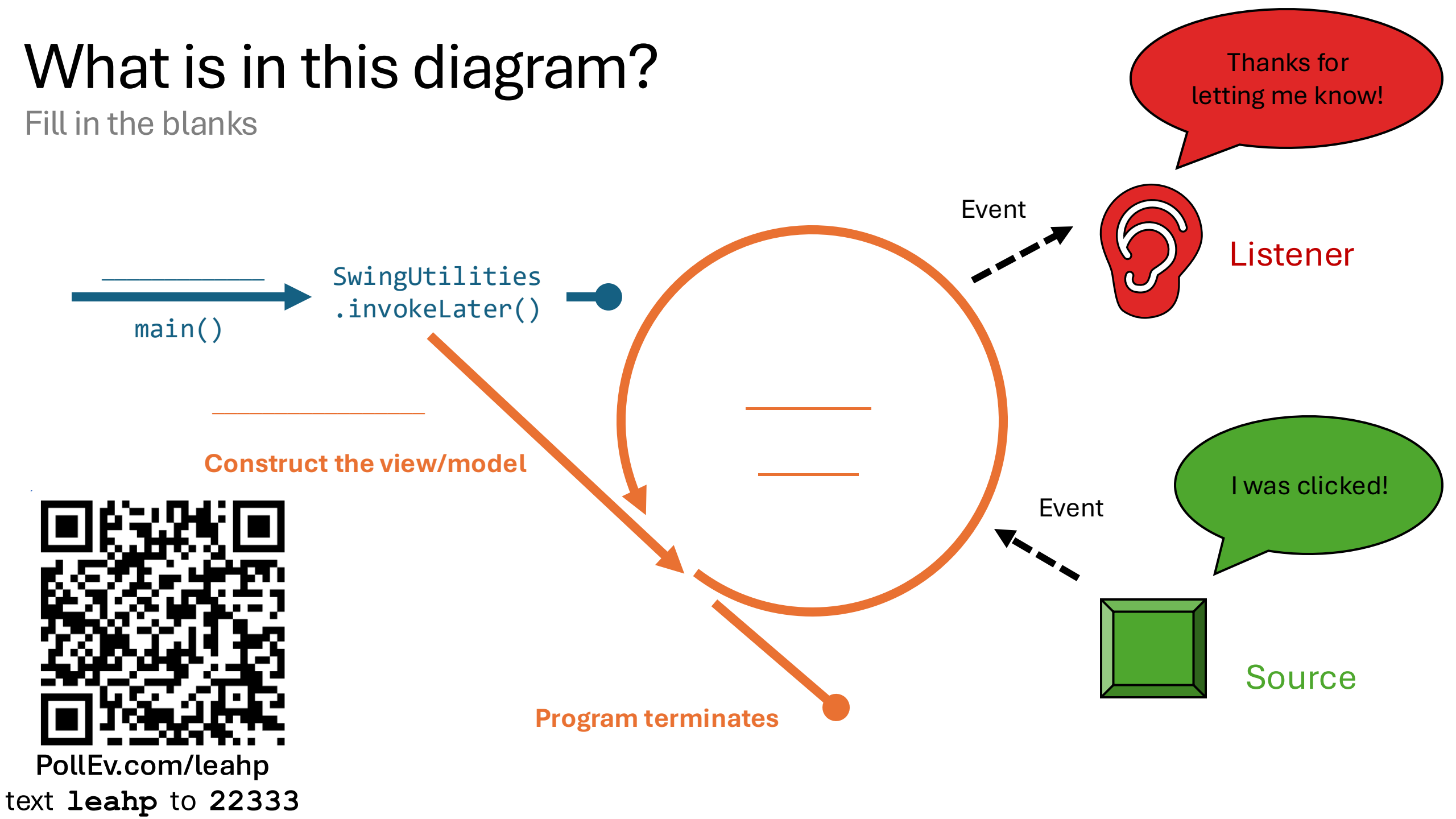
What is in this diagram?

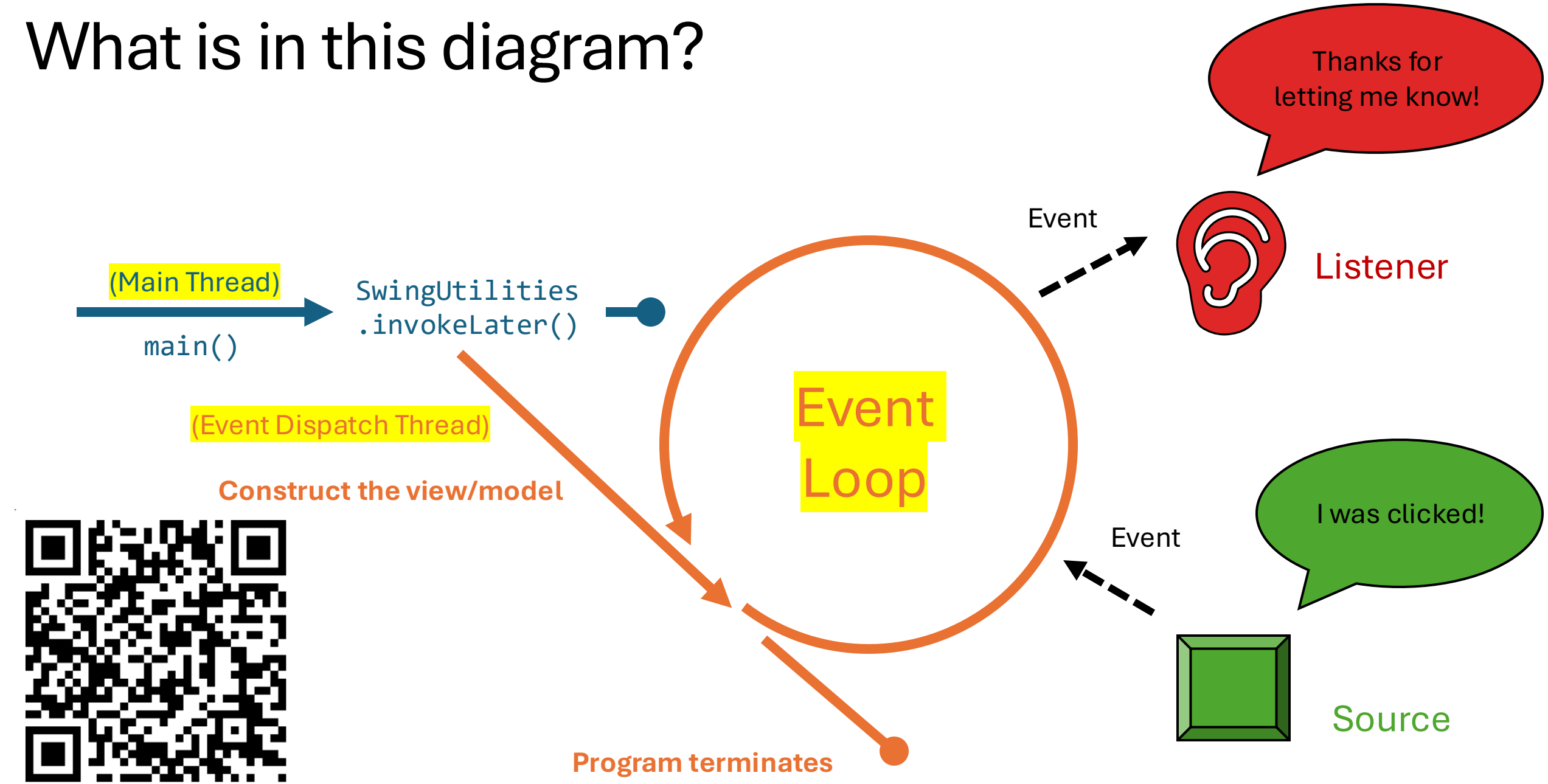
Fill in the blanks



PollEv.com/leahp

text **leahp** to **22333**

What is in this diagram?



PollEv.com/leahp

text **leahp** to **22333**

Lecture 26: Concurrency

CS 2110, Matt Eichhorn and Leah Perlmutter

November 25, 2025

Roadmap

Java, Complexity, OOP

- start– 9/30

ADTs I

- List, Stack, Queue, Iteration
  - 10/2 – 10/16

ADTs II

- Trees, Set, Map, Hash Table, Graph
  - Tues 10/21- 11/13

Beyond ADTs

- Graphical User Interfaces & Event-Driven Programming
  - 11/18, 11/20
- **Parallel Programming**
  - 11/25, 12/2
- Data Structures and Social Implications
  - 12/4

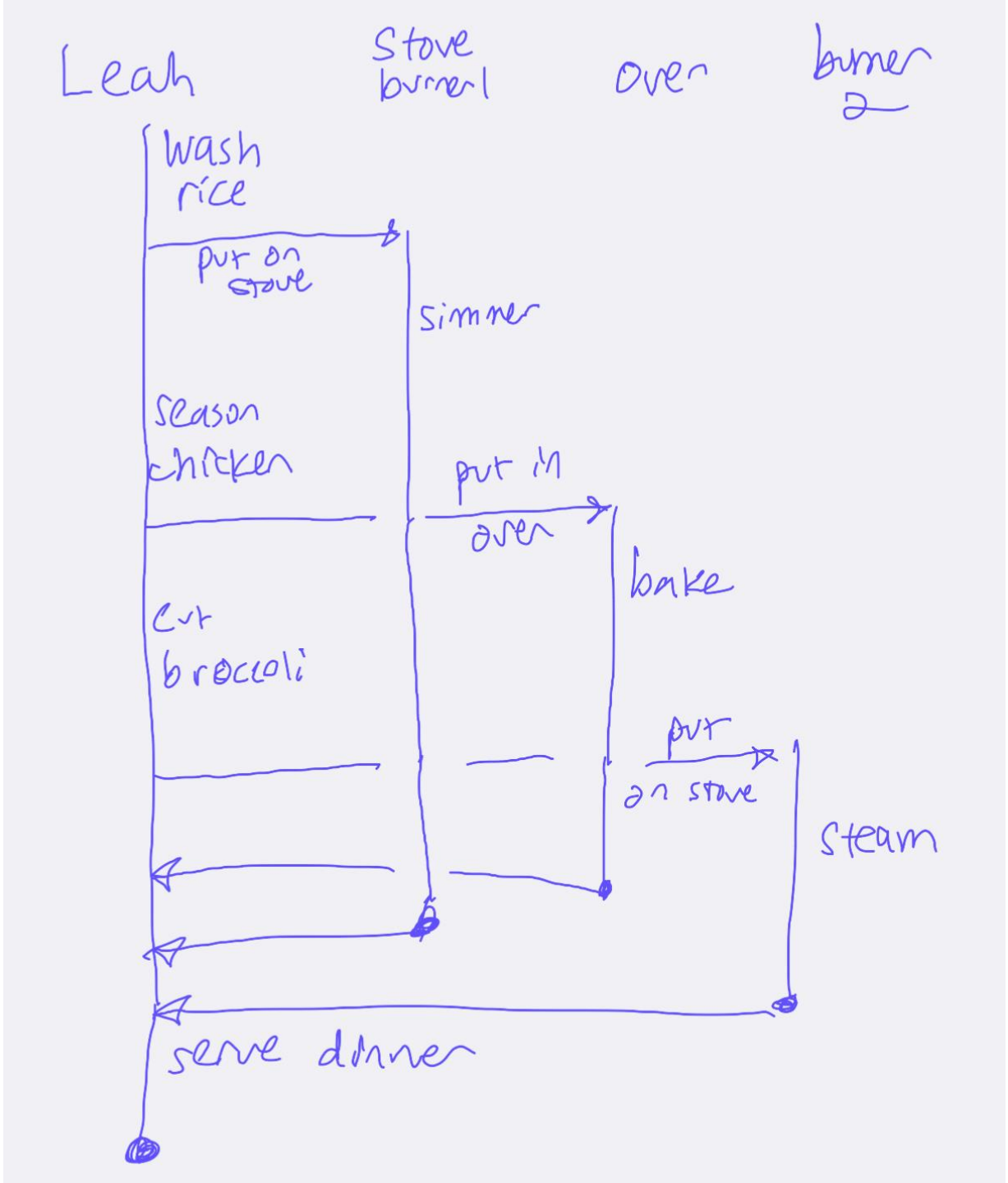
Overview of today

- Concurrent Tasks and the Operating System
- The Thread Class
  - Thread()
  - run() and start()
  - sleep()
  - join()
- Race Conditions
- Data Structures and Thread Safety
- Tomorrow: Safely Coordinating Threads (synchronization)

Concurrent Tasks and the Operating System

Concurrent tasks and the operating system

- **concurrency** – ability to carry out multiple procedures at the same time (WHITEBOARD: cooking)

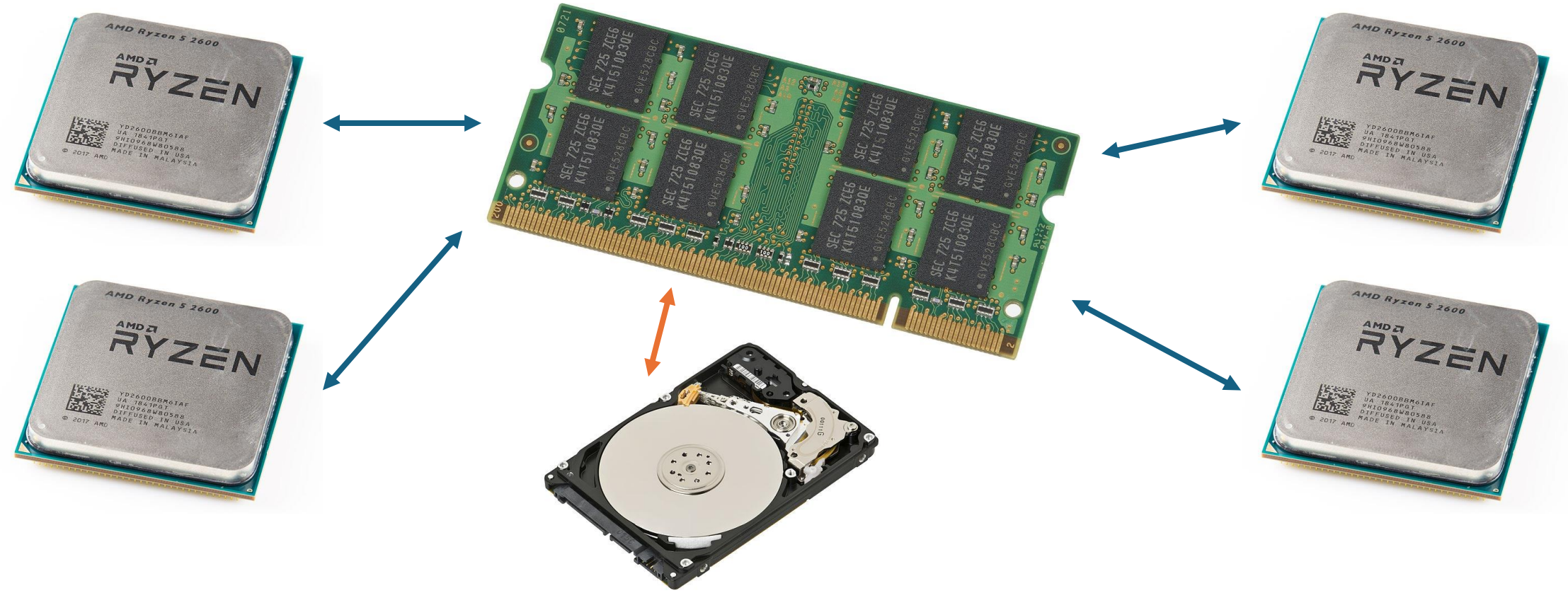


WHITEBOARD: a diagram that represents concurrent tasks while cooking. Time progresses from top to bottom. Vertical lines parallel to each other represent activities going on at the same time. For instance, Leah cut broccoli at the same time as stove burner 1 simmered and oven baked.

Note: the labels are labeling *lines* not regions.

How computers work

- Computer = Memory + CPU cores + I/O peripherals



Concurrent tasks and the operating system

- **operating system** – program that begins executing when you turn on your computer and manages the execution of all other programs
- **process** – a running program, a sequence of instructions with its own memory
- **thread** – part of a running program, a sequence of instructions with access to memory shared between other threads of the same process

Computer = Memory + CPU cores + I/O peripherals

Concurrent tasks and the operating system

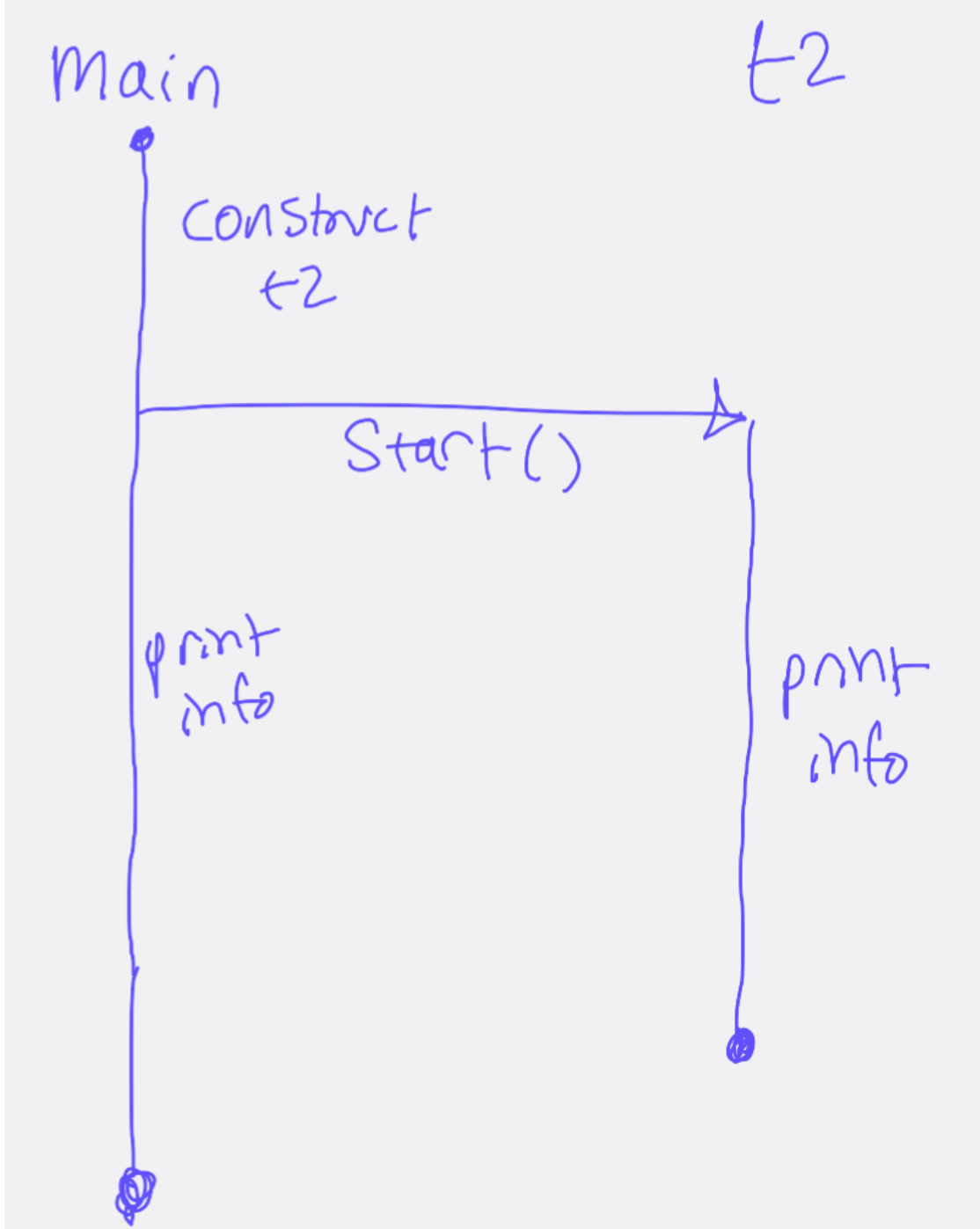
- **scheduling** – given multiple threads and processes that must run concurrently, decide which to run on which pieces of hardware at any given time (operating system does this)
- **time slicing** – describes switching between actively working on multiple different procedures at the same time
- **parallelism** – the ability to simultaneously carry out multiple procedures at an instantaneous point in time, achievable by dividing the work among multiple entities, or *processors*

Computer = Memory + CPU cores + I/O peripherals

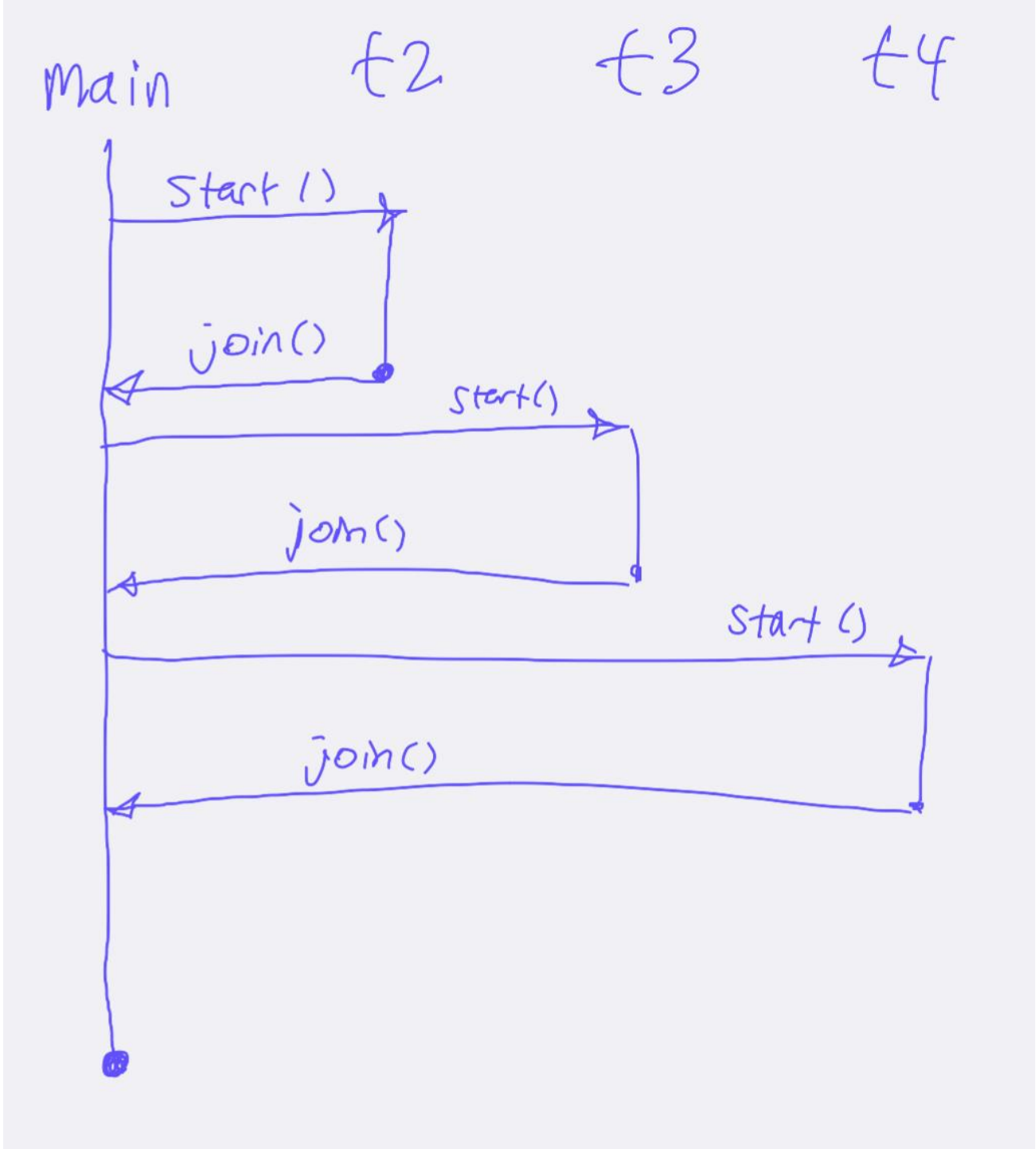
Threads in Java

Thread Basics

- **Code demo**
  - Thread class and basic attributes
  - Constructing and running new threads
- **Whiteboard**
  - Thread diagram for 2-thread program
- **Code demo**
  - Indeterminate order of execution
  - sleep() and interruptions
  - join() and sequencing
- **Whiteboard**
  - four threads running sequentially with join



WHITEBOARD: a thread diagram that represents the execution of ThreadDemo.simpleDemo() in the lecture code.



WHITEBOARD: a thread diagram that represents the execution of ThreadDemo.fourThreads() in the lecture code.

Shared Data

- **Code demo**
  - counting to 2 million
- **Poll**
  - what will be the final value of shared.x?
    - exactly 2,000,000 every time
    - exactly 1,000,000 every time
    - almost 2,000,000 most of the time
    - just over 1,000,000 most of the time
    - None of the above

PollEv.com/leahp
text `leahp` to `22333`

Shared Data

- **Code demo**
  - counting to 2 million
- **Poll**
  - what will be the final value of shared.x?
    - exactly 2,000,000 every time
    - exactly 1,000,000 every time
    - almost 2,000,000 most of the time
    - just over 1,000,000 most of the time
    - **None of the above –** When we ran the experiment, the outcomes were all over the place between 1 and 2 million.

PollEv.com/leahp

text `leahp` to `22333`

Race Conditions

(the need for synchronization)

Race Conditions

- **atomic –** describes an operation that executes fully or not at all, whose parts cannot be separated
- `shared.x++` is actually 3 steps!
  - LOAD: Load the value of shared.x from RAM into Register_1
  - INCREMENT: Add 1 to Register_1
  - STORE: Store Register_1 into RAM at the address shared.x
- Why? Compiled languages
  - Source code is compiled into machine code which runs on the CPU
  - Machine code instructions are much simpler and less powerful than lines of code in Java
  - A line of Java code is not necessarily atomic

Machine Code and Hardware

- The CPU has several registers that can store data directly on the CPU
- Machine code CANNOT
  - do arithmetic on data stored in memory (RAM)
- Machine code CAN
  - perform arithmetic on data stored in registers
  - load data from RAM into a register
  - store data from a register into RAM

Example: shared primitive

Thread 1:
```
shared.x++;
```

Thread 2:
```
shared.x++;
```

Scenario 1

1. T1 LOAD ($reg_1 \leftarrow 0$)
2. T1 INC ($reg_1 \leftarrow 1$)
3. T1 STORE ($x \leftarrow 1$)
4. T2 LOAD ($reg_2 \leftarrow 1$)
5. T2 INC ($reg_2 \leftarrow 2$)
6. T2 STORE ($x \leftarrow 2$)

Scenario 2

1. T1 LOAD ($reg_1 \leftarrow 0$)
2. T2 LOAD ($reg_2 \leftarrow 0$)
3. T2 INC ($reg_2 \leftarrow 1$)
4. T2 STORE ($x \leftarrow 1$)
5. T1 INC ($reg_1 \leftarrow 1$)
6. T1 STORE ($x \leftarrow 1$)

Racing for the Critical Section

- **critical section** - bit of code that accesses a shared data structure
- **race condition** - situation where the result of the program depends on which thread accesses the shared data structure first
- How can we coordinate threads to safely access the critical section?
  - … Synchronization! (coming soon to an auditorium near you)

Data Structures and Thread Safety

- **Code Demo**
  - Naive shared ArrayList
- **Poll**
  - Make a prediction: What do you think the program will output?

PollEv.com/leahp

text `leahp` to `22333`

Data Structures and Thread Safety

- **Code Demo**
  - Naive shared ArrayList
- **Poll**
  - Make a prediction: What do you think the program will output?
  - **Answer:** If thread safe, it should output an array containing five 1's and five 2's in indeterminate order. Our experiment showed that it did this most of the time, but running it hundreds of times revealed a few unexpected outputs such as an array containing five 1's and no 2's.

PollEv.com/leahp
text **leahp** to **22333**

Data Structures and Thread Safety

- Non-thread-safe data structures might behave correctly 99% of the time and have wildly unexpected behavior the other 1% of the time
- Why?
  - Race conditions!
- What do you think is the critical section of the code for an ArrayList?

Data Structures and Thread Safety

- Suppose the fields of the ArrayList include **size**, which indicates how many elements there are and **data**, the array backing the ArrayList.
- Here is some pseudocode that represents a possible implementation of the add() method.
  - idx = this.size
  - data[idx] = 0
  - this.size++ // Actually 3 separate machine instructions!
    - load this.size into a register
    - increment that register
    - store that register back to this.size

Metacognition

- Take 1 minute to write down a brief summary of what you have learned today
- indeterminate -
- atomic -
- critical section -
- race condition -

Thanks and have a great break!