# CS 2110

Lecture 26
Concurrency

April 28th, 2026

1

## Agenda

- Concurrency
- Threads
- Race Conditions

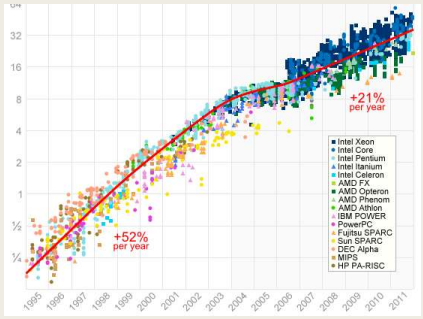
## Learning Outcomes

103. Describe advantages and disadvantages of concurrent code.
104. Use Java's Thread class to write concurrent code.
105. Identify race conditions in a given piece of concurrent code and list the possible states the program can be in after it executes.

2

# CONCURRENCY

3




Single-Threaded Integer Performance, based on SPECint results
Source: Jeff Preshing, preshing.com

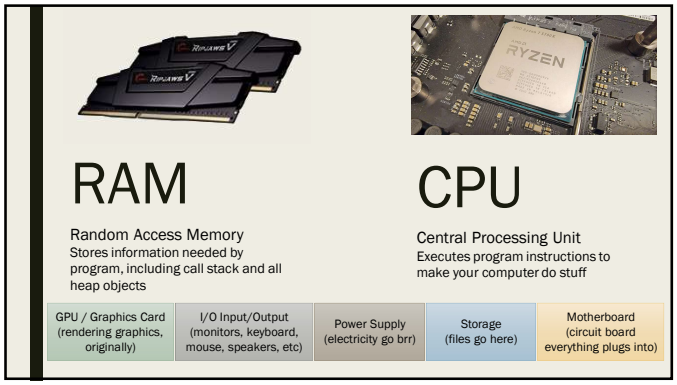
4


5

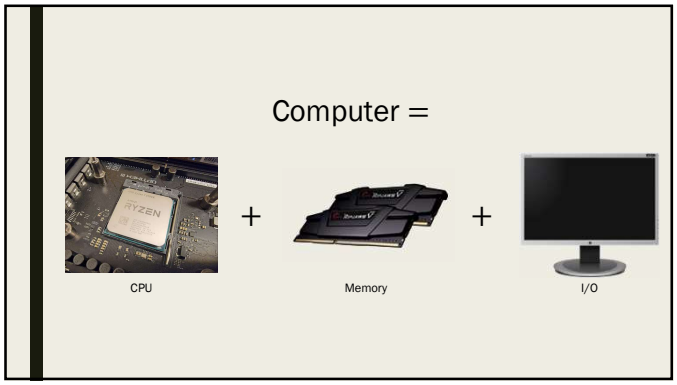
# Parallelism

Carrying out multiple tasks
at the exact same time

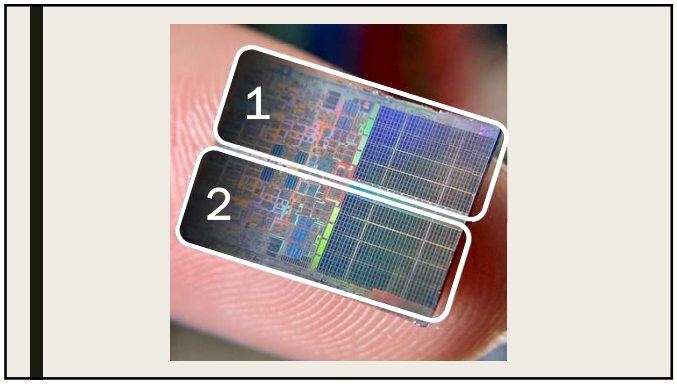
6


RAM
Random Access Memory
Stores information needed by program, including call stack and all heap objects
CPU
Central Processing Unit
Executes program instructions to make your computer do stuff
GPU / Graphics Card (rendering graphics, originally)
I/O Input/Output (monitors, keyboard, mouse, speakers, etc)
Power Supply (electricity go brr)
Storage (files go here)
Motherboard (circuit board everything plugs into)

7


Computer =
CPU
Memory
I/O

8


1
2

9

```
task1();
task2();
task3();
task4();
task5();
task6();
```

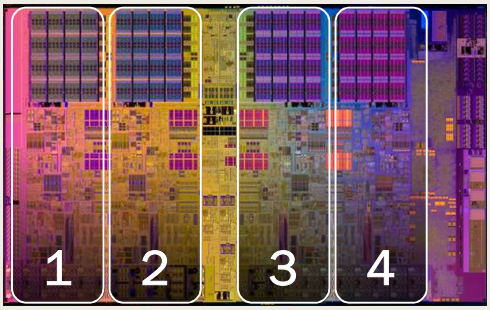
NOT real Java code

10

```
run(core1) {
    task1();
    task2();
    task3();
}
run(core2) {
    task4();
    task5();
    task6();
}
```

NOT real Java code

11



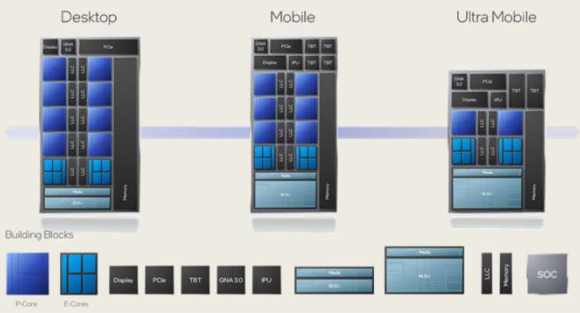

12

```
run(core1) {
    task1();
    task2();
}
run(core2) {
    task4();
    task5();
}
```

```
run(core3) {
    task3();
}
run(core4) {
    task6();
}
```

**NOT** real Java code

13




14

# Problems

1. Different computers have different cores

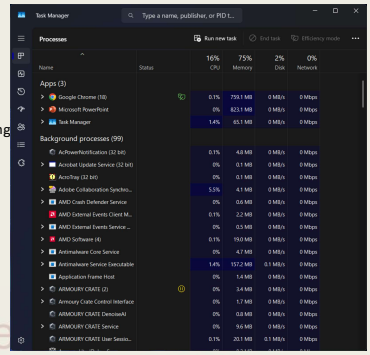
15

```
for (int i = 0; i < tasks.length; i++) {
  run(core(i % numCores)) {
    task i();
  }
}
```

NOT real Java code

16

```
for (int i = 0; i < tasks.leng
   run(core(i % numCores)) {
      task i();
   }
}
```



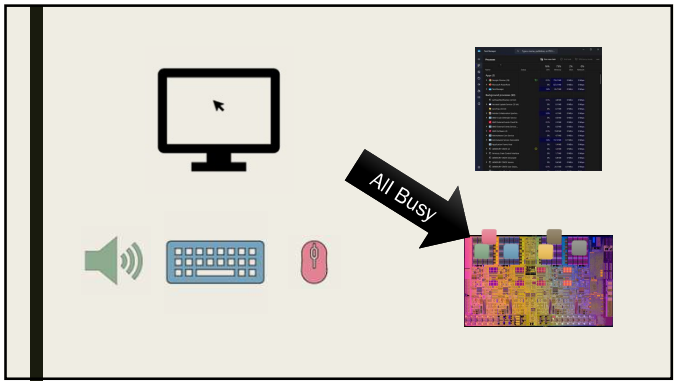

NOT re

17

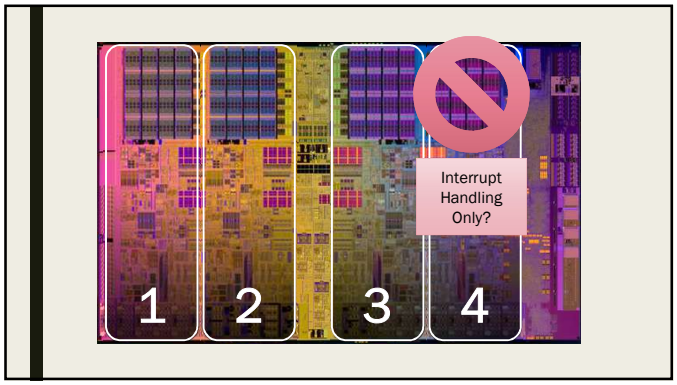
# Problems

1. Different computers have different cores
2. Multiple programs run at the same time

18




19




20

## Problems

1. Different computers have different cores
2. Multiple programs run at the same time
3. Rare but high priority events cannot wait

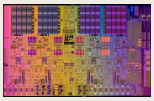
21

# THREADS

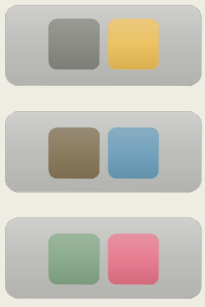
22

# Abstraction

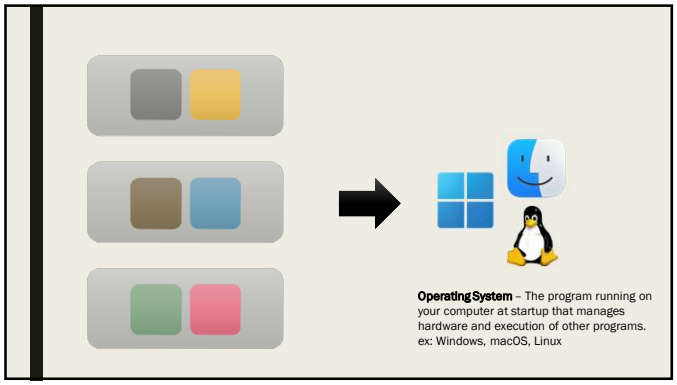
Abstraction Barrier



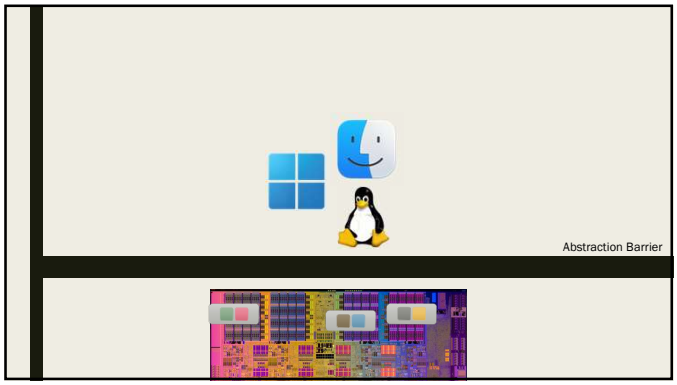
23



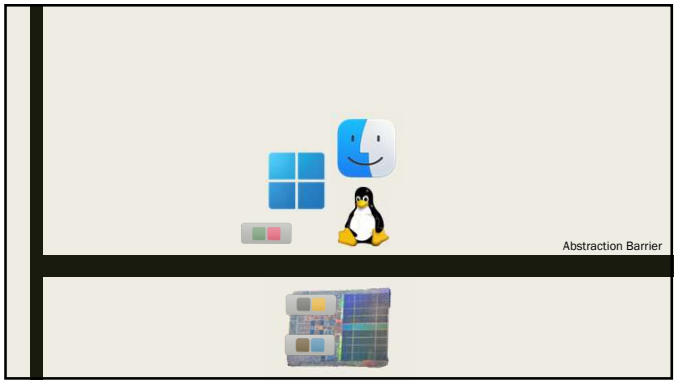
24


**Operating System** – The program running on your computer at startup that manages hardware and execution of other programs.
ex: Windows, macOS, Linux

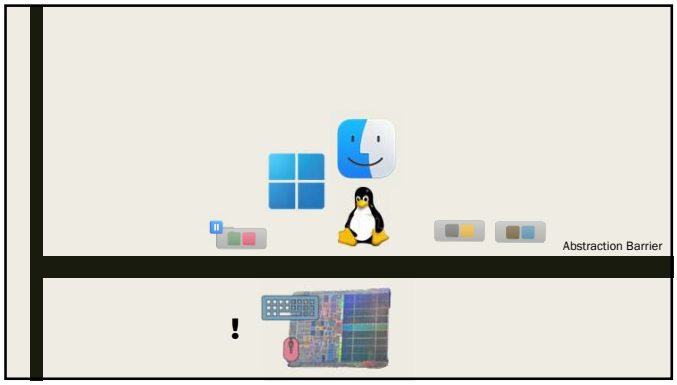
25


Abstraction Barrier

26


Abstraction Barrier

27




28

# Thread

A model of a single, sequential execution of instructions that performs work (within a process)

A running program, w/ exclusive chunk of memory from OS

29

# Problems

1. Different computers have different cores
2. Multiple programs run at the same time
3. Rare but high priority events cannot wait

30

# Solutions

1. Different computers have different cores
2. Multiple programs run at the same time

   Unlimited threads (and processes) scheduled by operating system onto cores
3. Rare but high priority events cannot wait

   OS can preempt thread to run higher priority tasks

31

# Drawbacks

Unlimited threads (and processes) scheduled by operating system onto cores →
- No guarantee when they execute
- No guarantee what order they run
- No guarantee how fast they run

OS can preempt thread to run higher priority tasks →
- No guarantee when they're paused

32

# Parallelism

Carrying out multiple tasks at the exact same time

# Concurrency

Having multiple independent tasks in progress at the same time

33

# JAVA THREADS

34

## Thread Class

Encapsulates operating system calls to create a thread.

Constructor:

```
public Thread(Runnable r);
```

Functional interface: run() method takes no arguments and returns void

Use:

```
Thread t = new Thread(() -> { ... });
```

35

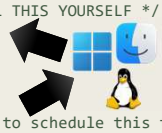
## Thread Methods

```
/** Get the current Thread */
static Thread currentThread();

/** The method with the code for this thread;
 *  DO NOT CALL THIS YOURSELF */
void run();

/** Ask the OS to schedule this thread for execution;
 *  Call this to start the thread */
void start();
```

36

# Thread Methods

```
/** The method with the code for this thread;
 *  DO NOT CALL THIS YOURSELF */
void run();
```



```
/** Ask the OS to schedule this thread for execution;
 *  Call this to start the thread */
void start();
```

You

37

# Thread Methods

```
/** Make the current thread sleep for the given time */
static void sleep(long milliseconds);

/** Wait for the thread to terminate */
void join();

/** Make the thread a daemon thread */
void setDaemon(boolean on);
```

Program will not wait for thread to finish before exiting

38

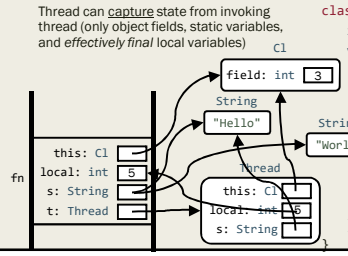
# NERD CORNER

An in-depth discussion of lambda capture

39

# Capturing State

Thread can capture state from invoking thread (only object fields, static variables, and *effectively final* local variables)




```
class Cl {
    int field = 3;
    void fn() {
        int local = 5;
        String s = "Hello";
        Thread t = new Thread(() -> {
            System.out.println(field);
            System.out.println(local);
            System.out.println(s);
        });
        t.start();
        s = "World";
    }
}
```

40

```
int[] x = { 0 };
```

| Thread 1: x[0]++ | Thread 2: x[0]-- |
|---|---|
| load x[0] | |
| add 1 | |
| store x[0] | |
| | load x[0] |
| | subtract 1 |
| | store x[0] |

41

```
int[] x = { 0 };
```

| Thread 1: x[0]++ | Thread 2: x[0]-- |
|---|---|
| load x[0] | |
| add 1 | |
| | load x[0] |
| | subtract 1 |
| | store x[0] |
| store x[0] | |

42

```
int[] x = { 0 };
```

| Thread 1: x[0]++ | Thread 2: x[0]-- |
|---|---|
| | load x[0] |
| load x[0] | |
| add 1 | |
| store x[0] | |
| | subtract 1 |
| | store x[0] |

43

# Race Condition

When the system's behavior depends on the timing of uncontrollable events.

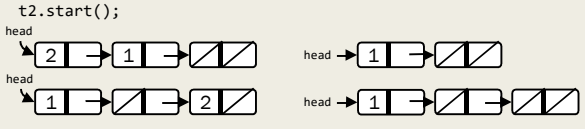
44

What sequence of operations results in each of the following states?

Which of the following is **not** a possible result of running this code?

```
CS2110List<Integer> list =
   new SinglyLinkedList<>();
Thread t1 = new Thread(() -> list.add(1));
Thread t2 = new Thread(() -> list.add(2));
t1.start();
t2.start();
```

```
public void add(T elem) {
  tail.next =
    new Node<>(null, null);
  tail.data = elem;
  tail = tail.next;
  size++;
}
```




45

# Race Conditions

Race conditions occur because order of instructions between threads is not guaranteed.

Within a single thread, instructions are guaranteed to run sequentially.

46

# Recap

- Parallelism lets us do multiple things at the same time
- Split code into concurrent threads (abstraction over cores)
- Give up control of execution order and speed to OS
- Race conditions if multiple threads write to same data

47

# CS 2110

48