Lecture 03: RAM, PRAM, and Threads

COSC 273: Parallel and Distributed Computing

Spring 2023

Announcement

Programming Assignment 1 posted soon

- due next Friday
- use HPC cluster

Last Time

A `CounterExample` Mystery!

What happens when multiples threads call `increment()`?

```
public void increment () {
    ++count;
}
```

Today

1. Random Access Machines (RAM)
2. Parallel Random Access Machines (PRAM)
3. Reasoning about all possible executions

Terminology

Program: sequence of instructions to be carried out by a computer

- specified by programmer through code

Process: a "computing entity" that can carry out instructions specified in a program

- e.g., CPU or CPU core

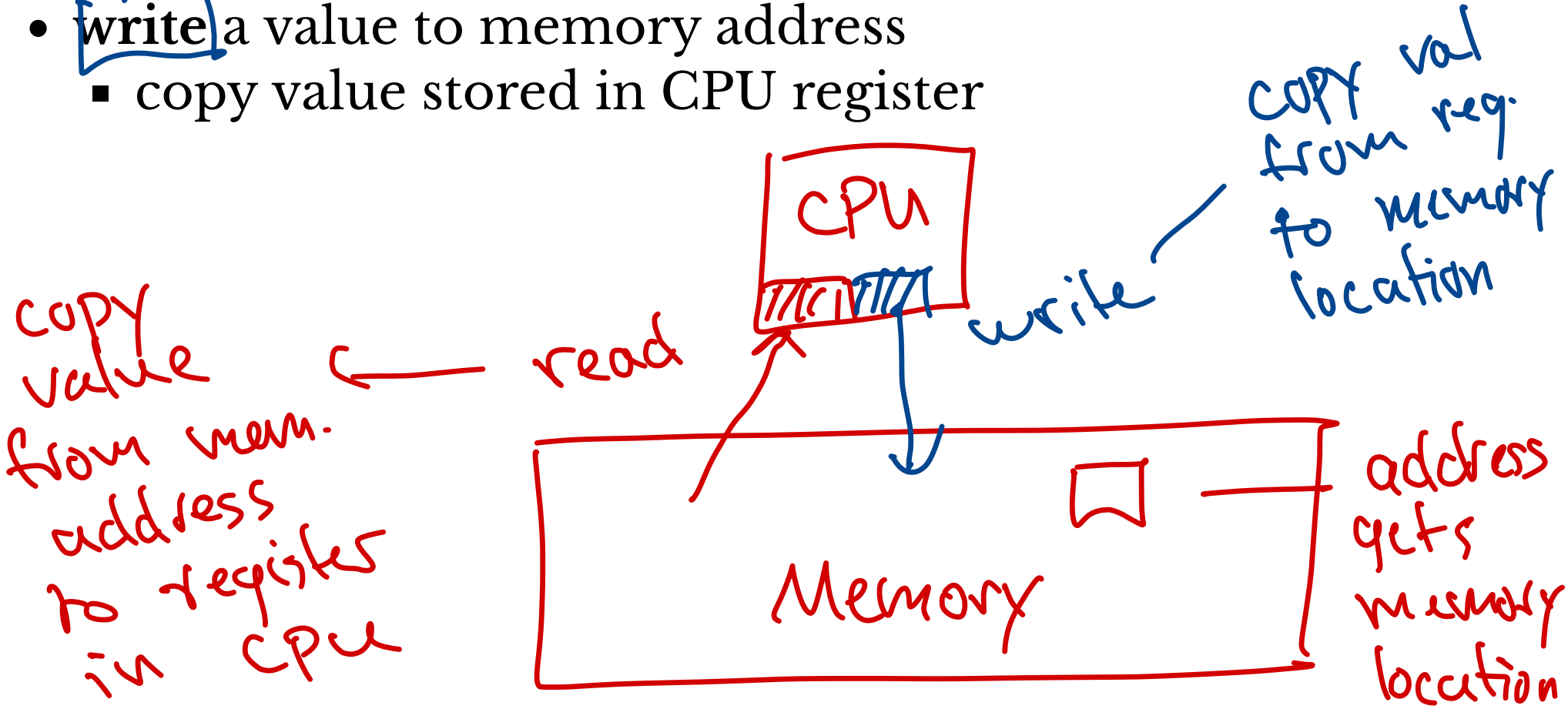
Execution: a sequence of operations performed by a set of processes

- accounts for *interactions* between processes
- specifies actual order in which operations are performed

CPU/Memory Interactions

Random Access Machine (RAM) model interactions:

- **read** a value from memory address
  - load value into CPU register
- **write** a value to memory address
  - copy value stored in CPU register



Counter Example, 1 thread

- `Counter` object is stored in memory
  - `Counter` stores a value `count`
- `CountThread` instructions stored in memory
- When `CounterThread` is executed, it follows these instructions

```
for (long i = 0; i < times; i++) {
     counter.increment();
}
```

- In turn:

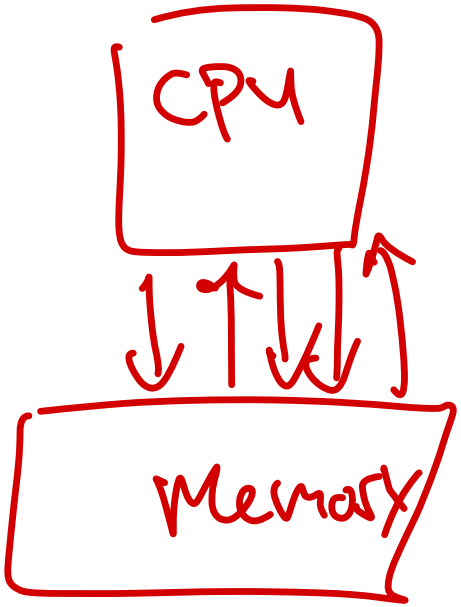
```
public void increment () { ++count; }
```

Question

What are CPU/Memory interactions when `counter.increment()` is executed?

```
public void increment () { ++count; }
```

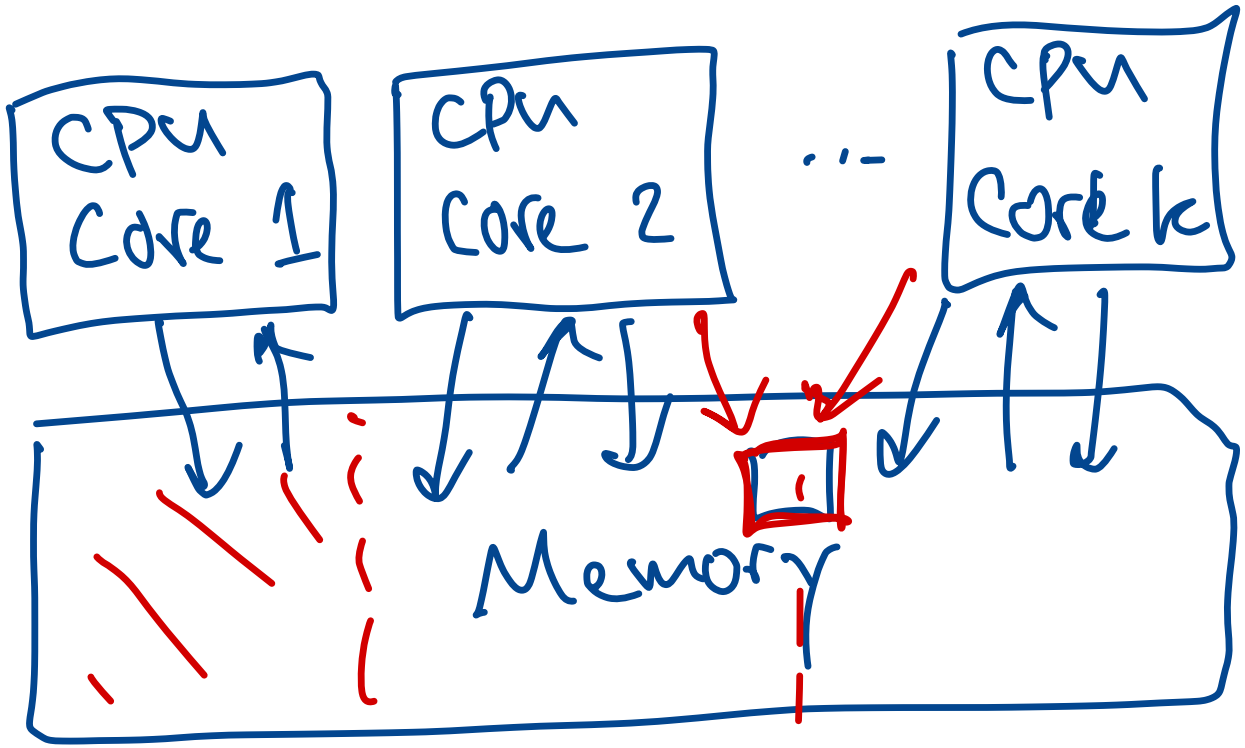
- read count from memory
- increment just CPU
- write new val to memory



Multicore Architecture

Modern computers:

- multiple cores
  - think of them as separate, independent CPUs
  - different cores *can* execute different threads simultaneously
- **shared memory**



PRAM model

Parallel Random Access Machine (PRAM)

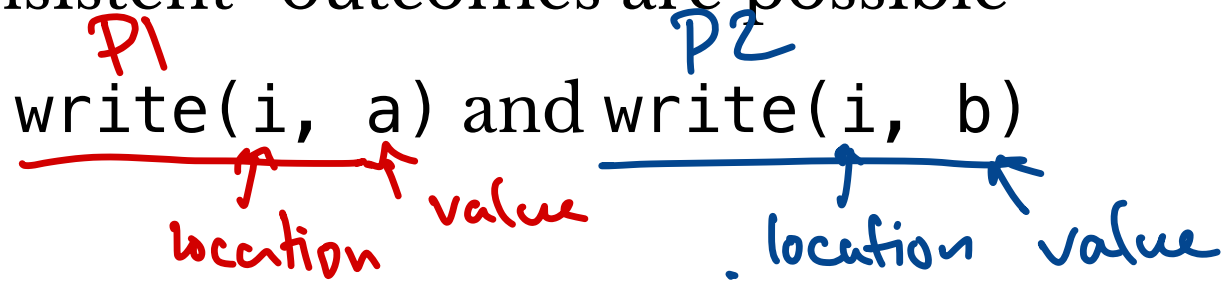
- Abstract model for parallel computing
- Shared memory: cells w/ addresses
  - think one giant array
- Multiple processors access memory
  - basic operations are `read(i)` and `write(i, val)`

PRAM Assumptions

- `read`/`write` operations are **atomic**

Nondeterminism:

- if multiple threads access same memory location concurrently all "consistent" outcomes are possible
  - two processes call `write(i, a)` and `write(i, b)`



  - one process calls `read(i)` another `write(i, a)`

After concurrent read/write

- Stored value is a
- read value could be b or a

Multicore Counter Example

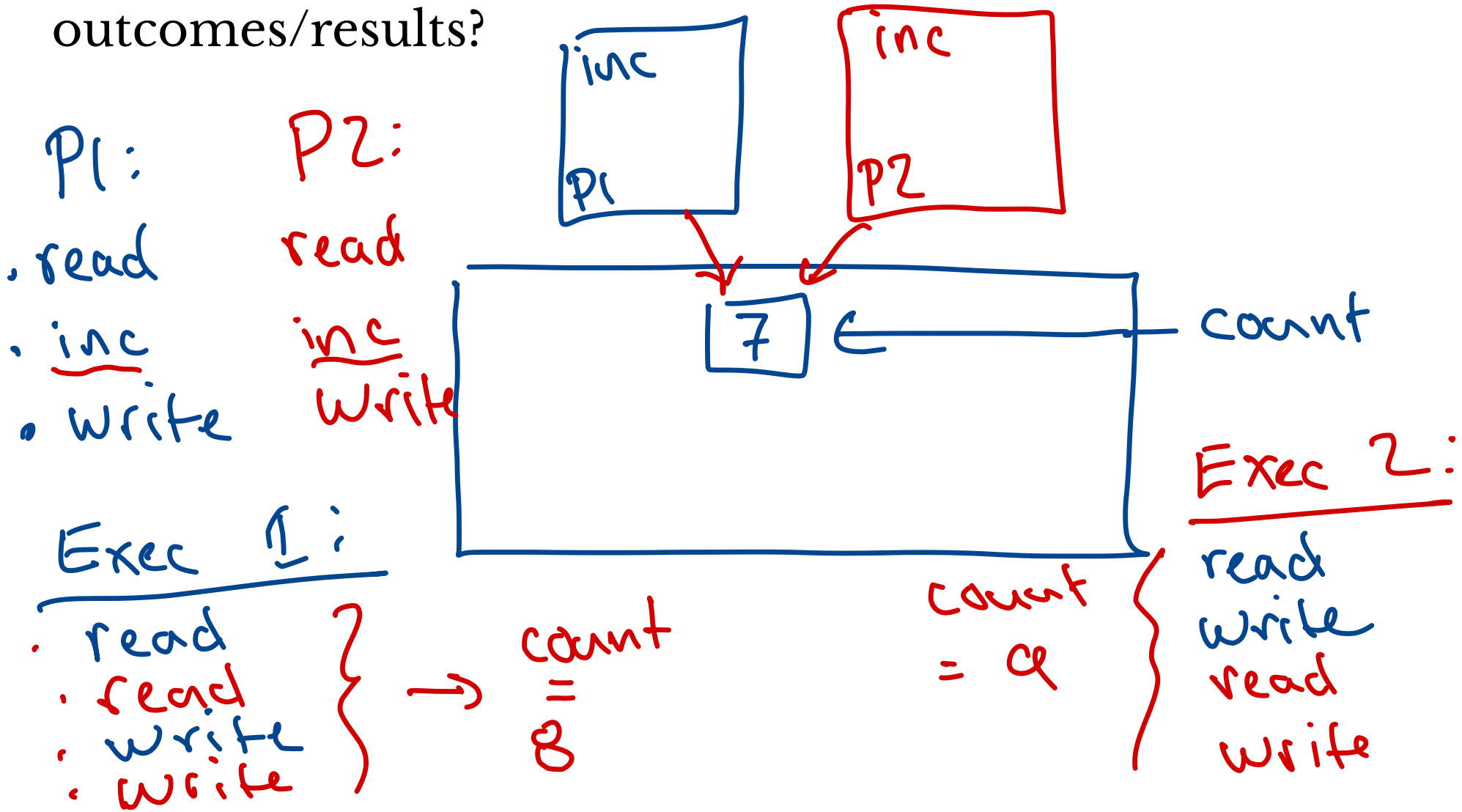
- two threads perform `increment` operation
- threads both try to `increment` **same** `Counter` concurrently

```
public void increment () { ++count; }
```

Question

Suppose: `count = 7` & two threads both call `increment()` concurrently

What are the possible executions? What are possible outcomes/results?



PRAM and Threads

PRAM model allows for all processes to access/modify all memory

- can choose to partition/allocate memory to individual processes as well
- shared memory used only when necessary
  - i.e., processes must interact/communicate

Thread-local variables

Each thread can have variables that only it accesses

- these are **thread-local variables**

```
public class CounterThread implements Runnable {
    private Counter counter;  private long times;
    public CounterThread (Counter counter, long times) {
        this.counter = counter; this.times = times;
    }

    public void run () {
        for (long i = 0; i < times; i++) {
            counter.increment();
        }
    }
}
```

Lecture 03 Activity

```
void increment(int[] a) {
    int i = 0;
    while (i < a.length) {
        a[i] = a[i] + 1;
        i = i + 1;
    }
}
```

Question 1

If `a = [0, 0, 0, 0]` and two threads, what are possible outcomes?

```
void increment(int[] a) {
    int i = 0;
    while (i < a.length) {
        a[i] = a[i] + 1;
        i = i + 1;
    }
}
```

Question 2

If `a = [0, 0, 0, 0]` and k threads, what are possible outcomes?

```
void increment(int[] a) {
    int i = 0;
    while (i < a.length) {
        a[i] = a[i] + 1;
        i = i + 1;
    }
}
```

Back to Counter

How could we **fix** the problem of mis-counting?

- Want every increment to count!

Next Week

1. Embarrassingly parallel computation
   - Programming assignment 01
2. Limits of Parallelism
3. Mutual Exclusion