CS 554 – Advanced Topics in Concurrent Computing Systems

William Mansky

Questions

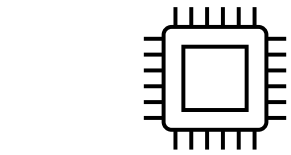
Nobody has responded yet.

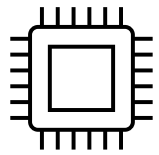
Hang tight! Responses are coming in.

Concurrent programming with caches

```
int val = *x;
*x = 0 + 1;
```

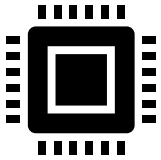
```
int val = *y;
*y = 0 + 1;
```





x	y
1	0

x	y
0	0



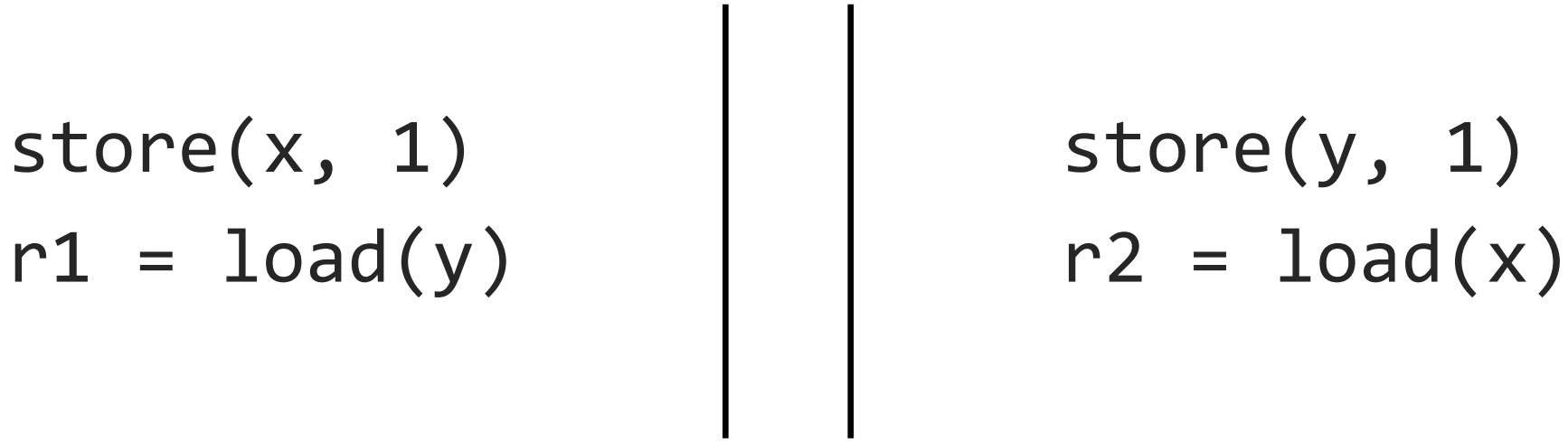
x	y
0	0

Memory

Hardware memory models

- Easiest to start at the processor level
- We'll still sometimes write C-like pseudocode, but imagine it's in your preferred assembly language
- Every processor architecture comes with a *specification* explaining its memory model – on this chip, what can you expect?

Sequential Consistency



- Sequential consistency (SC): programs execute as if every operation happens in a single total order

Sequential Consistency

```
1   store(x, 1)        |   2   store(y, 1)
3   r1 = load(y)       |   4   r2 = load(x)
```

- Sequential consistency (SC): programs execute as if every operation happens in a single total order

Sequential Consistency

```
1   store(x, 1)        |   2   store(y, 1)
4   r1 = load(y)       |   3   r2 = load(x)
```

- Sequential consistency (SC): programs execute as if every operation happens in a single total order

Sequential Consistency

```
1   store(x, 1)        |   3   store(y, 1)
2   r1 = load(y)       |   4   r2 = load(x)
```

- Sequential consistency (SC): programs execute as if every operation happens in a single total order
- This matches the one-memory, no-caches picture (but note the “as if”!)

Sequential Consistency

```
1   store(x, 1)
2   r1 = load(y)
```

```
3   store(y, 1)
4   r2 = load(x)
```

- Exercise: What are the possible outcomes of this program?

Sequential Consistency and beyond

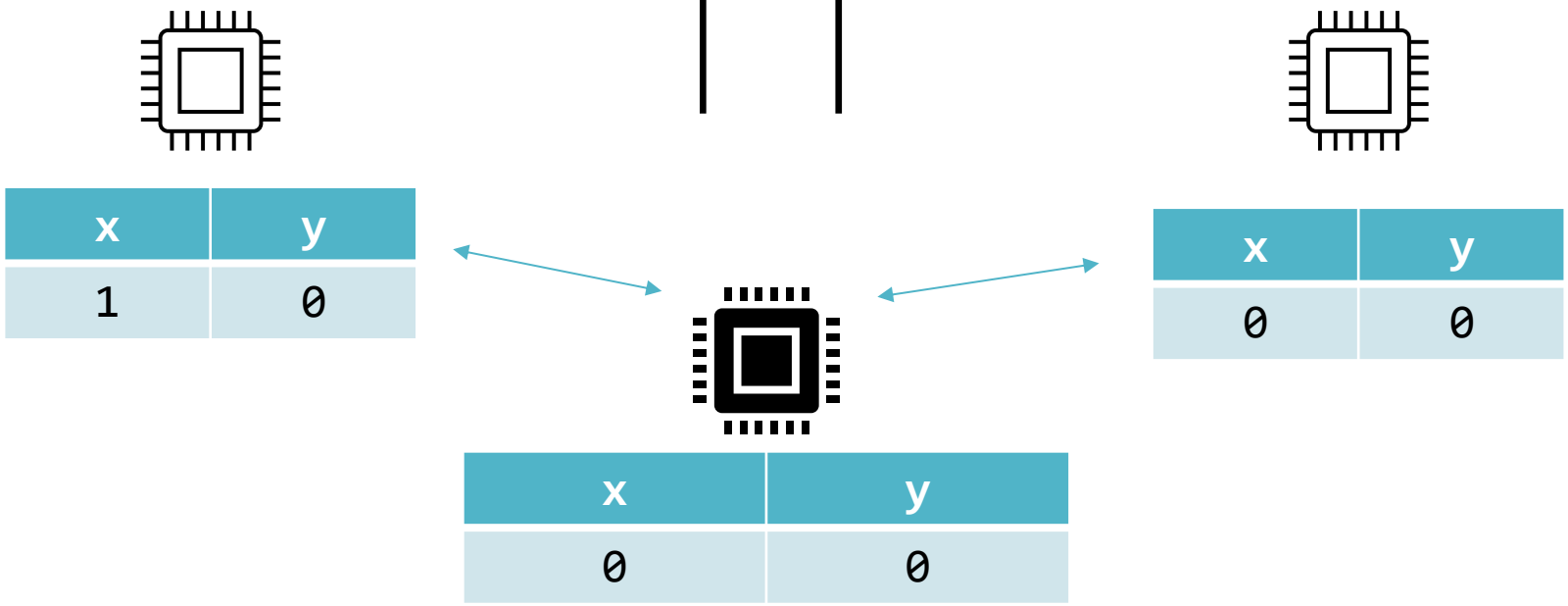
```
1   store(x, 1)        |   3   store(y, 1)
2   r1 = load(y)       |   4   r2 = load(x)
```

- A *weak* or *relaxed* memory model is one that allows non-SC behavior
- E.g., almost every model other than SC allows
  `r1 = 0, r2 = 0`

Concurrent programming with caches

```
int val = *x;
*x = 0 + 1;
```

```
int val = *y;
*y = 0 + 1;
```



Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Weak MM #1: Total Store Order (TSO)

```
store(x, 1)          |  store(y, 1)
r1 = load(y)         |  r2 = load(x)
```

- Sequential consistency (SC): programs execute as if every operation happens in a single total order
- Total store order (TSO): stores are totally ordered, but loads can read anywhere in that order (subject to program order)

Weak MM #1: Total Store Order (TSO)

```
store(x, 1)          |  store(y, 1)
r1 = load(y)         |  r2 = load(x)
```

- Total store order (TSO): stores are totally ordered, but loads can read anywhere in that order (subject to program order)
- Or equivalently: writes are buffered, propagate to other threads eventually

Weak MM #1: Total Store Order (TSO)

```
r1 = load(y)        |   r2 = load(x)
store(x, 1)         |   store(y, 1)
```

- Exercise: can we get `r1 = r2 = 1`? Why or why not?

Weak MM #1: Total Store Order (TSO)

```
// Thread 1   // Thread 2 // Thread 3 // Thread 4
x = 1         y = 1       r1 = x      r3 = y
                          r2 = y      r4 = x
```

- Total store order (TSO): stores are totally ordered, but loads can read anywhere in that order (subject to program order)
- Or equivalently: writes are buffered, propagate to other threads eventually

Weak MM #1: Total Store Order (TSO)

```
store(x, 1)          |          store(y, 1)
r1 = load(y)         |          r2 = load(x)
```

- If we don't want weak behavior, we can add *fences*

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Weak MM #2: Partial Store Order (PSO)

```
store(x, 1)          |  store(y, 1)
r1 = load(y)         |  r2 = load(x)
```

- Total store order (TSO): stores are totally ordered, but loads can read anywhere in that order (subject to program order)
- Partial store order (PSO): stores *to each location* are totally ordered (subject to program order)

Weak MM #2: Partial Store Order (PSO)

```
store(x, 1)          |  |          store(y, 1)
r1 = load(y)         |  |          r2 = load(x)
```

- Partial store order (PSO): stores *to each location* are totally ordered (subject to program order)
- Or equivalently: writes are buffered, with a separate buffer for each location

Weak MM #2: Partial Store Order (PSO)

```
r1 = load(y)        ||        r2 = load(x)
store(x, 1)         ||        store(y, 1)
```

- Partial store order (PSO): stores *to each location* are totally ordered (subject to program order)
- Or equivalently: writes are buffered, with a separate buffer for each location

Weak MM #2: Partial Store Order (PSO)

```
store(x, 1)  ||  r1 = load(y)
store(y, 1)  ||  r2 = load(x)
```

- Partial store order (PSO): stores *to each location* are totally ordered (subject to program order)
- Or equivalently: writes are buffered, with a separate buffer for each location

Litmus Tests

```
// Thread 1  // Thread 2 // Thread 3 // Thread 4
x = 1        y = 1       r1 = x      r3 = y
                         r2 = y      r4 = x
```

- Write buffering, read buffering, independent reads of independent writes, …
- A good practical characterization, but not very expressive – there are always more variations

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.