SWEN430 - Compiler Engineering

Lecture 21 - Memory Models I

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Overview

Thread 1	Thread 2
`x = 1;`	`while(done == 0) { ... }`
`done = 1;`	`print(x);`

- **Q)** Can this program print `0` ?
- In **Java**, what else could affect this?

Terminology

Thread 1	Thread 2
`x = 1;`	`while(done == 0) { ... }`
`done = 1;`	`print(x);`

- **Shared Variables**. Denoted by `x`, `y`, `done`, etc.
- **Local Variables**. Denoted by `r1`, `r2`, `r3`, etc.

Sequential Consistency

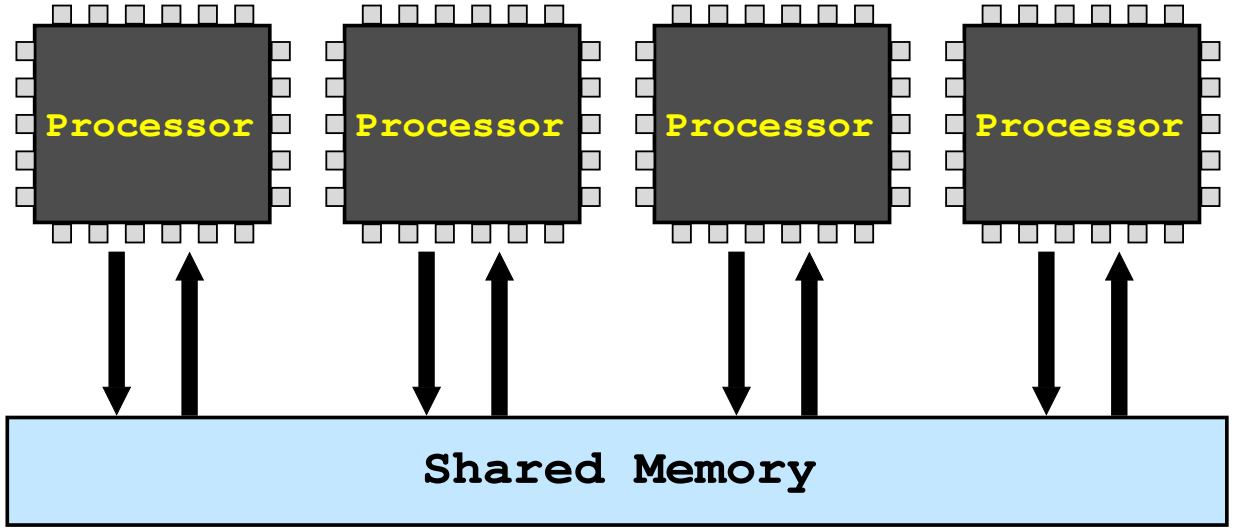
Sequential Consistency

“ ... the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.” – Lamport’79.

Thread 1	Thread 2
`x = 1;`	`r1 = y;`
`y = 1;`	`r2 = x;`

- Can this program ever see `r1=1` and `r2=0` at the same time?

Mental Model



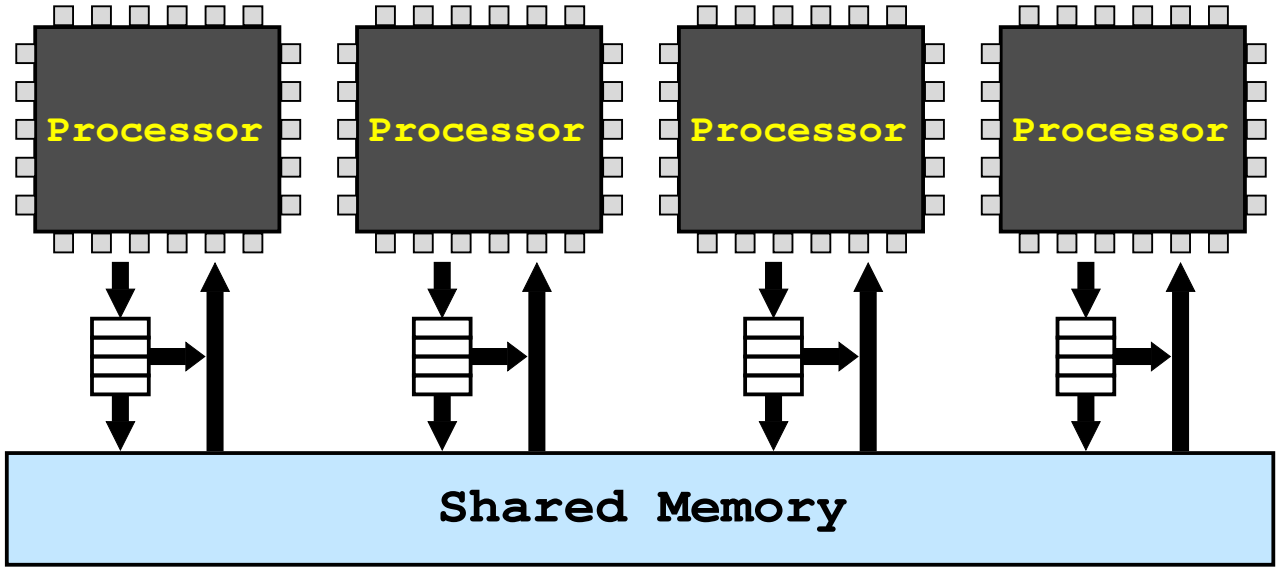
- A model for sequentially consistent machine (but not only one).
- Processors connected to **single shared memory**.
- Processors read / write **directly** from shared memory

Interleavings

Thread 1	Thread 2
x = 1;	r1 = y;
y = 1;	r2 = x;

x=1 y=1 r1=y (1) r2=x (1)	x=1 r1=y (0) y=1 r2=x (1)	x=1 r1=y (0) r2=x (1) y=1
r1=y (0) x=1 y=1 r2=x (1)	r1=y (0) x=1 r2=x (1) y=1	r1=y (0) r2=x (0) x=1 y=1

Total Store Order (x86)



- Processors connected to **single shared memory**.
- Each processor writes to **local queue** (FIFO).
- Processor reads consult local queue **before** share memory.
- Whenever write reaches shared memory **all future reads** see it.

TSO Litmus Tests

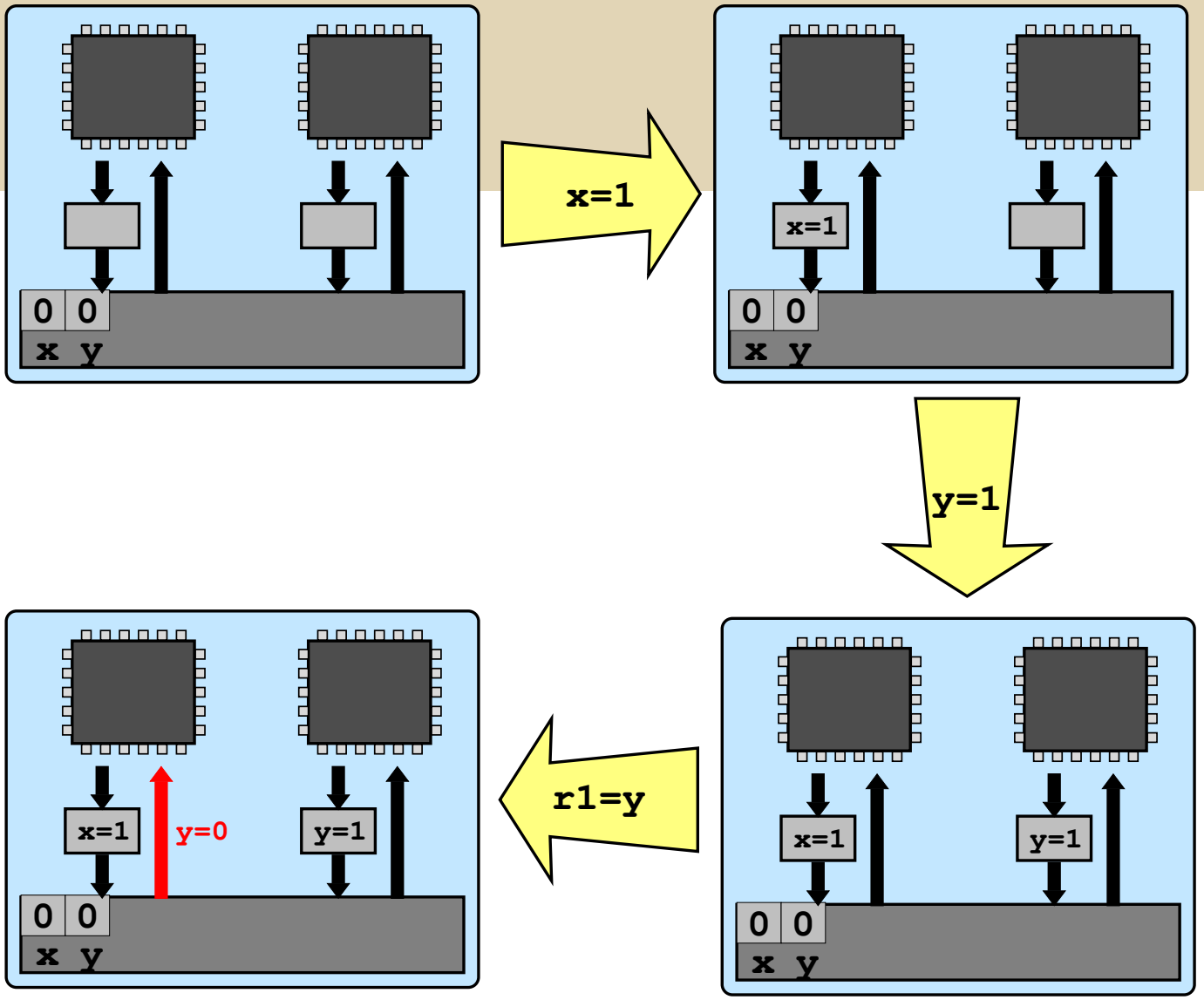
Thread 1	Thread 2
`x = 1;`	`r1 = y;`
`y = 1;`	`r2 = x;`

- Cannot see `r1=1, r2=0` under TSO (and under sequential consistency).

Thread 1	Thread 2
`x = 1;`	`y = 1;`
`r1 = y;`	`r2 = x;`

- Can see `r1=0, r2=0` under TSO (but **not** under sequential consistency).

TSO Litmus



Thread 1	Thread 2
`x = 1;`	`y = 1;`
`r1 = y;`	`r2 = x;`

Memory Barriers

"In short, because reordering memory references allows much better performance, and so **memory barriers** *are needed to force ordering in things like synchronization primitives whose correct operations depends on ordered memory references"*

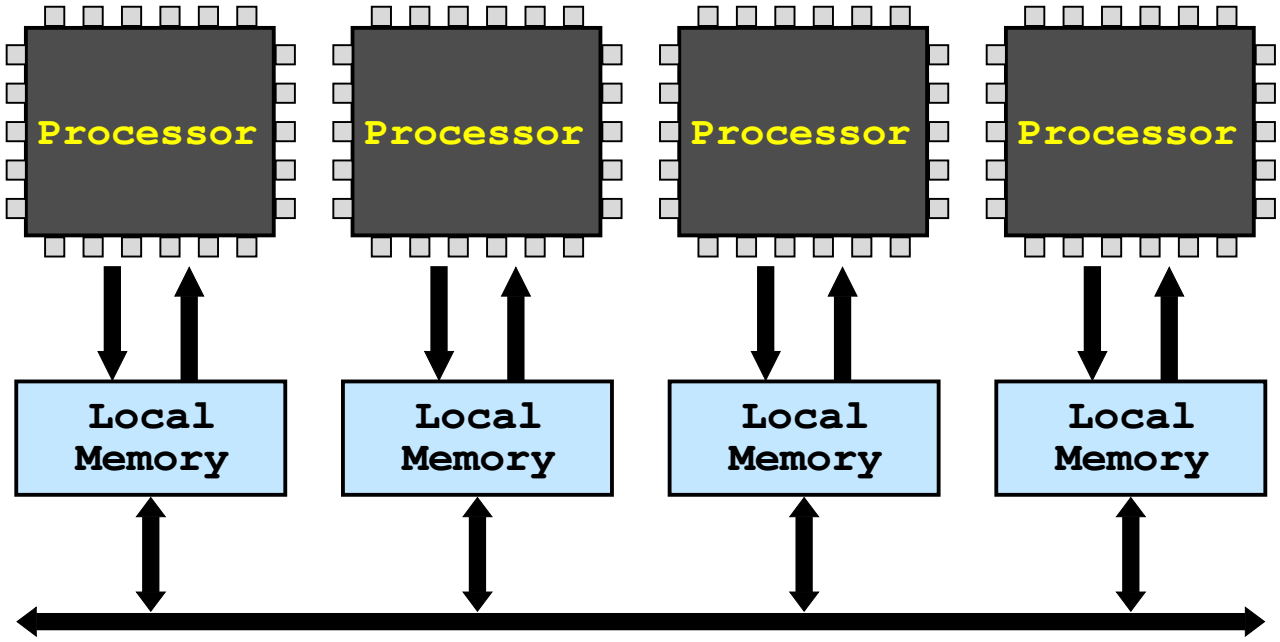
– McKenney '09

Memory Barriers

Thread 1	Thread 2
`x = 1;`	`y = 1;`
`barrier;`	`barrier;`
`r1 = y;`	`r2 = x;`

- Barriers (or fences) are explicit instructions to flush local queue.
- Using barriers above means cannot see `r1=0, r2=0`.
- On X86, have `LFENCE`, `SFENCE` and `MFENCE` instructions.

Relaxed Memory Model (ARM/POWER)



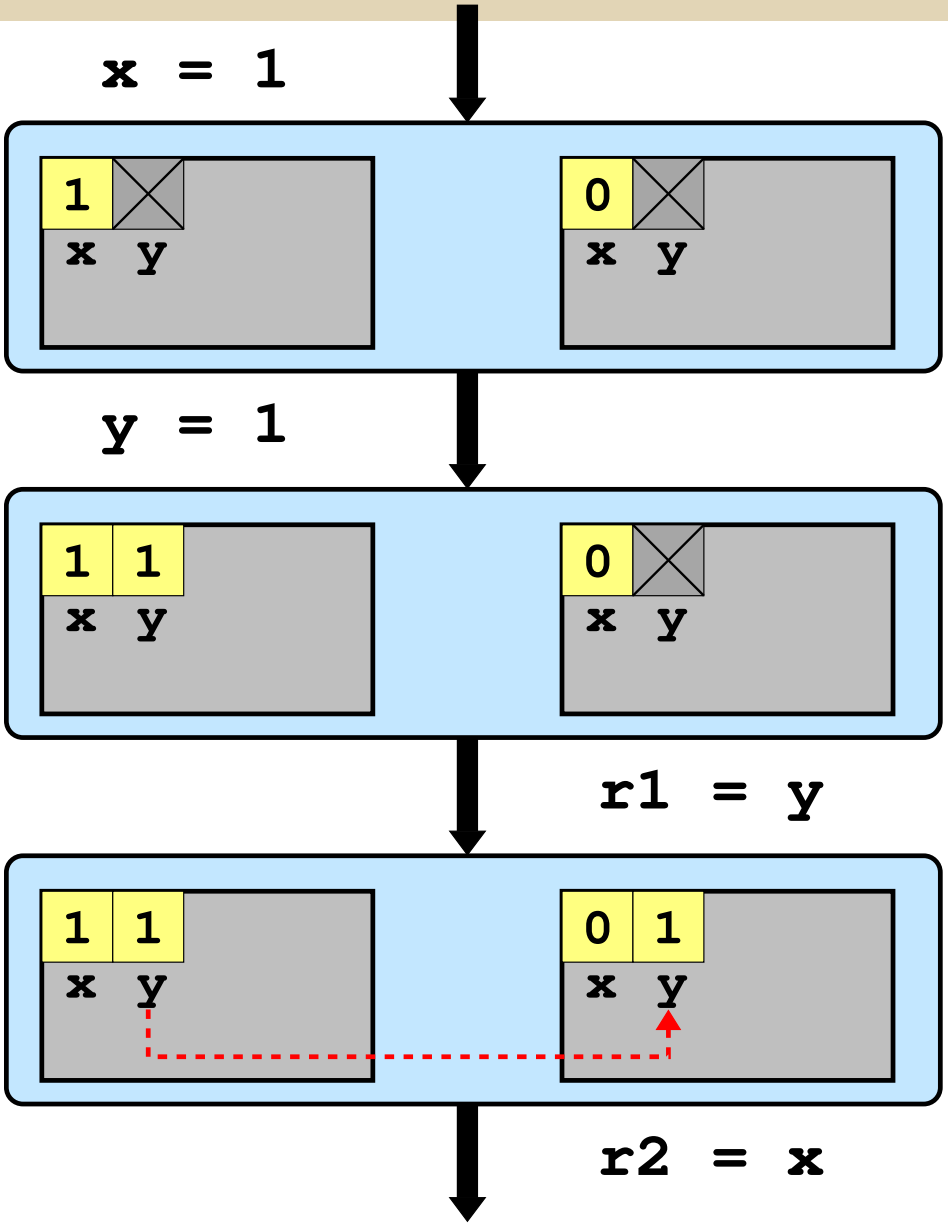
- Processor maintains **complete copy** of memory.
- Writes propagate to other processors **independently**.
- Reads can be **delayed arbitrarily** (i.e. are not always up to date).

ARM/POWER Litmus Tests

Thread 1	Thread 2
`x = 1;`	`r1 = y;`
`y = 1;`	`r2 = x;`

- Cannot see `r1=1, r2=0` under TSO or sequential consistency.
- But **can** see `r1=1, r2=0` under ARM/POWER!

ARM/POWER Litmus Tests (Cont'd)



References

- **Hardware Memory Models**, Russ Cox, 2021. `https://research.swtch.com/hwmm`
- **How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs**, Leslie Lamport, "IEEE Trans. Comput., 690-691, 1979.
- **Memory Barriers: a Hardware View for Software Hackers**, Paul E. MecKenney, Linux Technology Center, 2009.