Lecture 26: Multiprocessors

- Today's topics:
  - Snooping-based coherence
  - Synchronization
  - Consistency

HW 10 due Friday

2 lectures this week

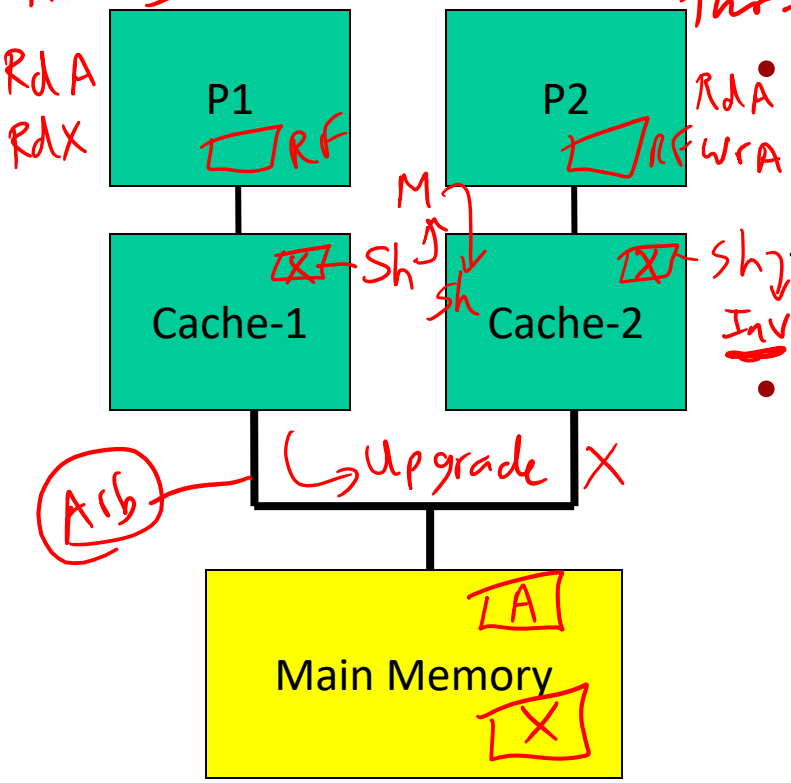
Next Tues: review session

Friday 8-10am : Final exam

Practice Final + Solutions posted on Canvas

Example

- P1 reads X: not found in cache-1, request sent on bus, memory responds, X is placed in cache-1 in shared state
- P2 reads X: not found in cache-2, request sent on bus, everyone snoops this request, cache-1does nothing because this is just a read request, memory responds, X is placed in cache-2 in shared state



- P1 writes X: cache-1 has data in shared state (shared only provides read perms), request sent on bus, cache-2 snoops and then invalidates its copy of X, cache-1 moves its state to modified
- P2 reads X: cache-2 has data in invalid state, request sent on bus, cache-1 snoops and realizes it has the only valid copy, so it downgrades itself to shared state and responds with data, X is placed in cache-2 in shared state, memory is also updated

Example

Request	Cache Hit/Miss	Request on the bus	Who responds	State in Cache 1	State in Cache 2	State in Cache 3	State in Cache 4
				Inv	Inv	Inv	Inv
P1: Rd X	Rd Miss	Rd X	Memory	S	Inv	Inv	Inv
P2: Rd X	Rd Miss	Rd X	Memory	S	S	Inv	Inv
P2: Wr X	Perms Miss	Upgrade X	No response. Other caches invalidate.	Inv	M	Inv	Inv
P3: Wr X	Wr Miss	Wr X	P2 responds (No mem wrtblc)	Inv	Inv	M	Inv
P3: Rd X	Rd Hit	-	-	Inv	Inv	M	Inv
P4: Rd X	Rd Miss	Rd X	P3 responds. Mem wrtbk	Inv	Inv	S	S

Cache Coherence Protocols

- Directory-based: A single location (directory) keeps track of the sharing status of a block of memory

- Snooping: Every cache block is accompanied by the sharing status of that block – all cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary

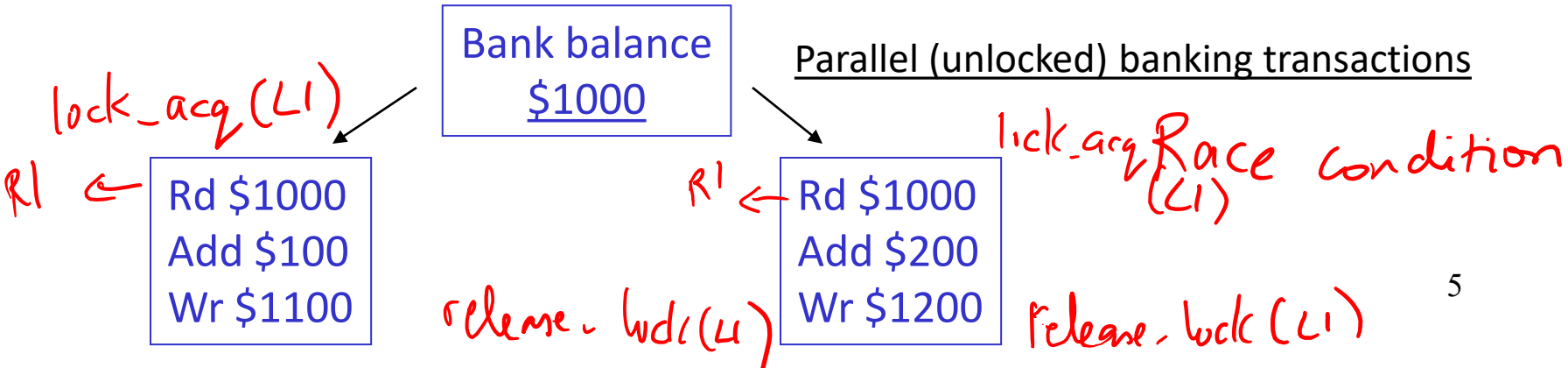
what we just described

  - Write-invalidate: a processor gains exclusive access of a block before writing by invalidating all other copies
  - Write-update: when a processor writes, it updates other shared copies of that block

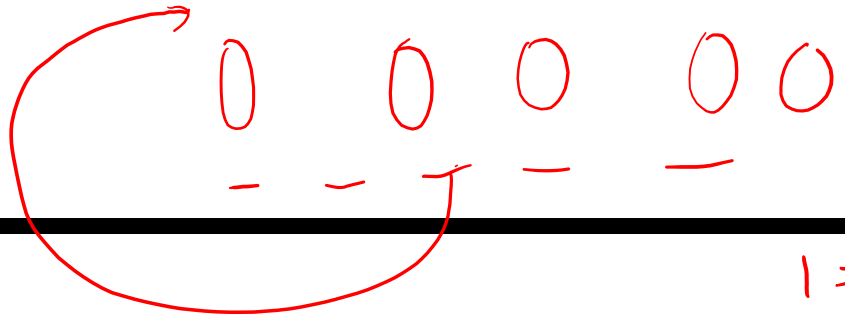
MOESI

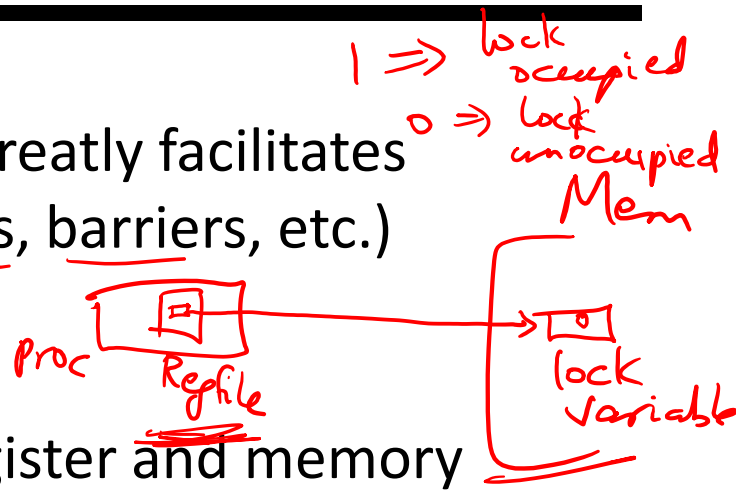
Constructing Locks

- Applications have phases (consisting of many instructions) that must be executed atomically, without other parallel processes modifying the data
- A lock surrounding the data/code ensures that only one program can be in a critical section at a time
- The hardware must provide some basic primitives that allow us to construct locks with different properties



Synchronization





- The simplest hardware primitive that greatly facilitates synchronization implementations (locks, barriers, etc.) is an atomic read-modify-write
- Atomic exchange: swap contents of register and memory
- Special case of atomic exchange: test & set: transfer memory location into register and write 1 into memory (if memory has 0, lock is free)

tas $1, 8($3)

```
lock:   t&s   register, location
        bnz   register, lock
        CS
        st    location, #0
```

lock acquire

critical section

lock release

When multiple parallel threads execute this code, only one will be able to enter CS

Coherence Vs. Consistency

- Coherence guarantees (i) write propagation (a write will eventually be seen by other processors), and (ii) write serialization (all processors see writes to the same location in the same order)

pertains to 1 variable X

- The consistency model defines the ordering of writes and reads to different memory locations – the hardware guarantees a certain consistency model and the programmer attempts to write correct programs with those assumptions

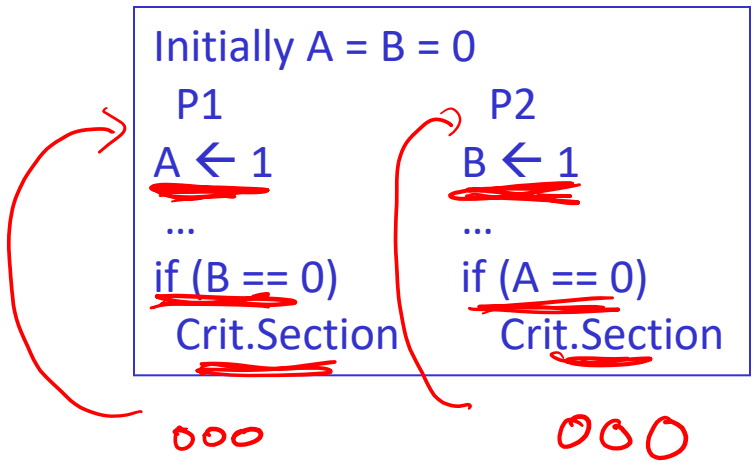
pertains to x and y

Consistency Example

deadlock, lockloc, starvation

- Consider a multiprocessor with bus-based snooping cache coherence

Surprising beh only caused if your code has race conditions



We program with a hw abstraction

⇒ sequential consistency model

Correctness & easy prog model

HW does re-orderings (for performance)

Consistency Example

- Consider a multiprocessor with bus-based snooping cache coherence

```
Initially A = B = 0
  P1                  P2
A ← 1               B ← 1
 …                   …
if (B == 0)         if (A == 0)
  Crit.Section        Crit.Section
```

The programmer expected the above code to implement a lock – because of ooo, both processors can enter the critical section

(mutual exclusion)

The consistency model lets the programmer know what assumptions they can make about the hardware's reordering capabilities

Sequential Consistency

- A multiprocessor is sequentially consistent if the result of the execution is achieveable by maintaining program order within a processor and interleaving accesses by different processors in an arbitrary fashion

- The multiprocessor in the previous example is not sequentially consistent

- Can implement sequential consistency by requiring the following: program order, write serialization, everyone has seen an update before a value is read – very intuitive for the programmer, but extremely slow

Relaxed Consistency

- Sequential consistency is very slow
- The programming complications/surprises are caused when the program has race conditions (two threads dealing with same data and at least one of the threads is modifying the data)
- If programmers are disciplined and enforce mutual exclusion with locks when dealing with shared data, we can allow some re-orderings and higher performance
- This is effective at balancing performance & programming effort