18-747 Lecture 23: Shared Memory Multiprocessors

James C. Hoe
Dept of ECE, CMU
November 26, 2001

Reading Assignments: *Two papers below*

Announcements: Office hour cancelled, come to my Talk at 4:30 instead
On Wednesday
Exam Review
Guest Lecture by Prof. D. Marculescu on
Power-Aware Processor Design

Handouts: *"Shared Memory Consistency Models: A Tutorial"*,
Adve and Gharachorloo
"Using Cache Memory to Reduce Processor-Memory Traffic",
Jim Goodman

Uniprocessor Load and Store Semantics

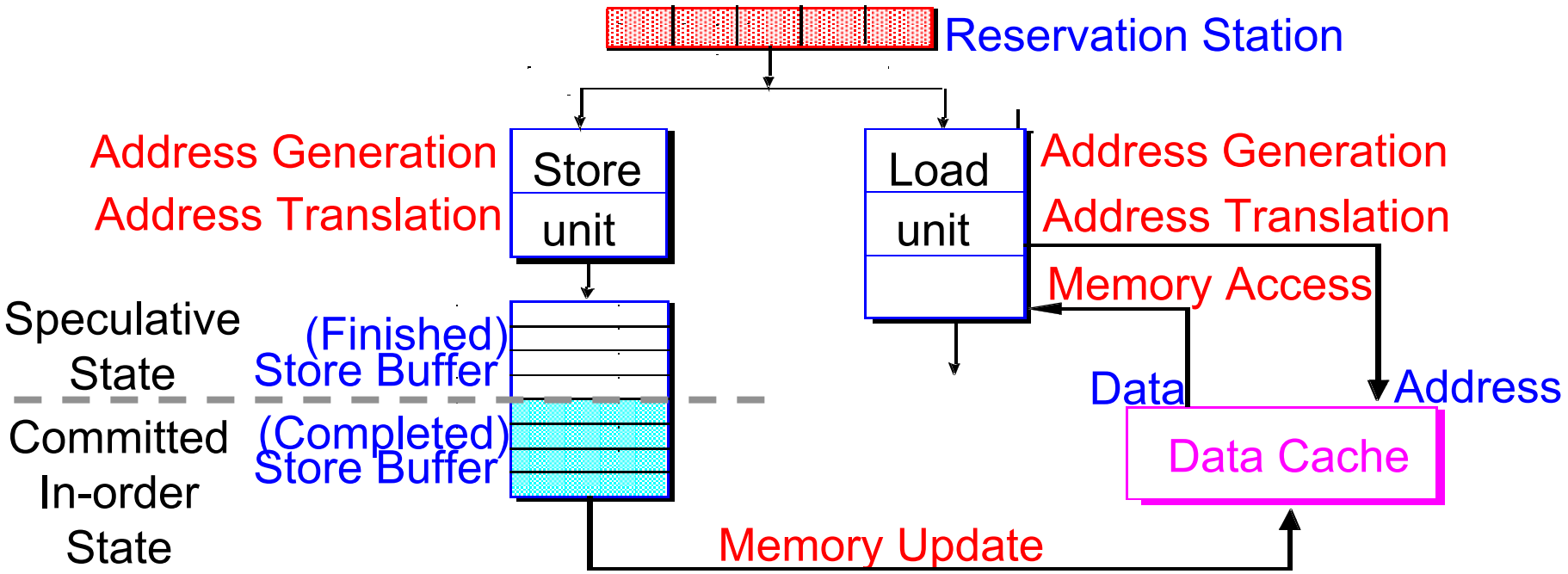
- The programmer (or compiler) believes memory reads and writes are going to be executed in program order
- Given $W_i(a, v) << R_j(a)$ *("<<" means precedes)*

  $R_j(a)$ must return v if there does not exist another W_k such that

  $$W_i(a, v) << W_k(a, v') << R_j(a)$$

  In short, a read should return the value of the "last" write to the same memory address
- Processors can guaranteed this semantics by obeying the ordering of memory data dependent operations
  - RAW: $W(a, v) << R(a)$
  - WAW: $W(a, v') << W(a, v)$
  - WAR: $R(a) << W(a, v')$

Uniprocessor Reordering of Loads & Stores



- Reordering of memory Op's to different addresses
- Buffered stores *(some stores may never show up in memory)*
- Load forwarding and load bypassing

Note: one should not be able to write a program to differentiate this from a true program-ordered execution!

Memory Ordering for Shared Memory Multiprocessors

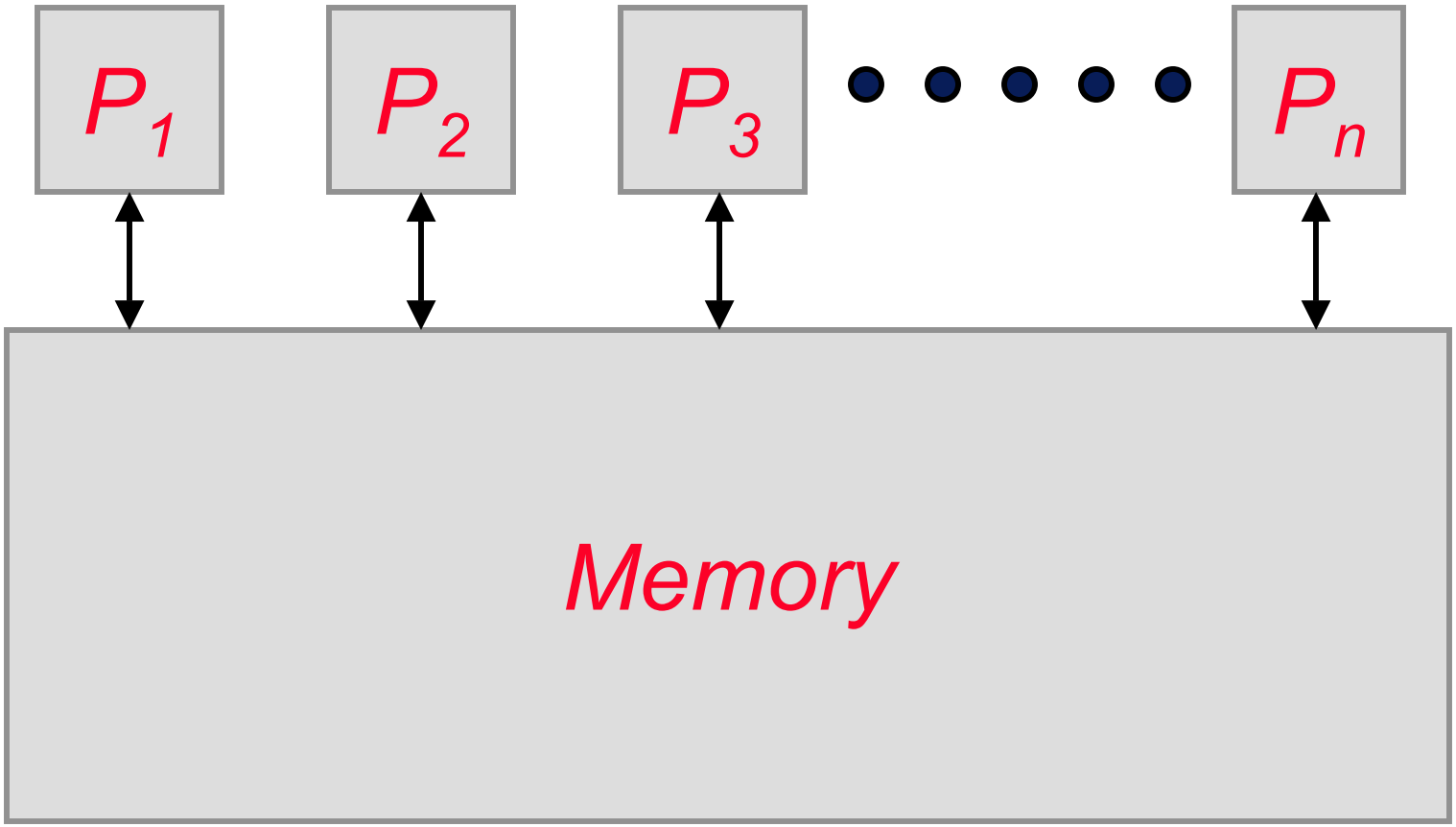
- Consider these two programs running on two processors that communicate via shared memory locations X and Y

```
Proc A:                          Proc B:
    Y is initially 1                 ……
    ……                               do {
    compute V                                lock=Load Y
    Store (X, V)                     } while (lock)
    Store (Y, 0)                     data = Load X
    ……                               ……
```

- Can the orders of Loads and Stores be swapped during dynamic execution?

Can the two programs perceive different order of events?

Simplified Multiprocessor/Memory Arch.



Ignore caches for now

Multiprocessor Memory Consistency

- A memory consistency model tells the programmer for each load which store operation bound the value to be returned
- Intuition: *a load should return the value of the "last" store to the same memory address*
- In multiprocessor, each processor performs a stream of reads and writes

. . . . . $W_{P1}(x)$ . . . . . .

. . . . . $W_{P2}(x)$,$W_{P2}(y)$, $R_{P2}(x)$, $R_{P2}(y)$ . . . . . .

. . . . . $W_{P3}(x)$. . . . $W_{P3}(y)$. . . . . $W_{P3}(x)$ . . . . . .

Who performed the last write to x before R(x) by P2?

- *How do you establish a global ordering of memory operations? Do you need a global ordering?*

Sequential Consistency (SC)

- What if every one can agree on a single point of serialization, for example at the memory bus?
- Sequential Consistency *[Lamport]*
  - a thread on a processor perceives its own memory ops in program order
  - memory ops from different processors can be interleaved arbitrarily (different interleaving are allowed on different runs)
  - For each run, all threads on all processors must agree on the same total ordering
  - *i.e. execution of a parallel program appear as some interleaving of the execution of parallel processes on a sequential machine"*

Example: Concurrent tasks T1 and T2 and shared variables X and Y (initially X = 0, Y = 0)

T1:	…….	T2:	…….
	Store(X, 1);		Y' = Load(Y);
	Store(Y, 1);		X' = Load(X);

SC says ⇒ Y' and X' may be assigned different values from run to run, but if Y' is 1 then X' cannot be 0

Implementation Implications of SC

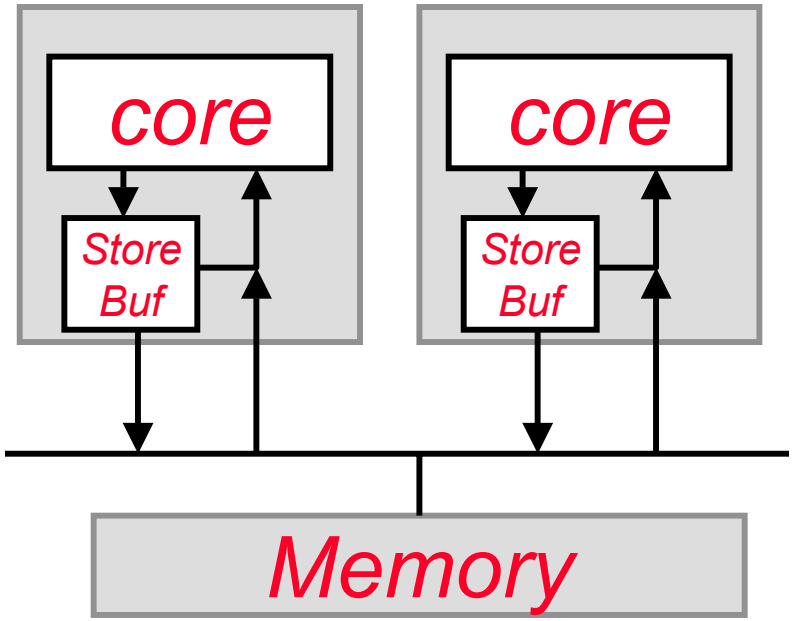
- SC requires a processor to preserve the following program-specified orderings at the common point of serialization

$R_i(x) < R_j(x)$
$R_i(x) < R_j(y)$

$R_i(x) < W_j(x)$ RAW
$R_i(x) < W_j(y)$

$W_i(x) < R_j(x)$ WAR
$W_i(x) < R_j(y)$

$W_i(x) < W_j(x)$ WAW
$W_i(x) < W_j(y)$



Shared location X and Y initially 0

T1: Store(X, 1) *T2: Store(Y, 1)*

print X Y *print X Y*

Is it possible that T1 prints "1 0" but T2 prints "0 1"?

Performance Implications of SC

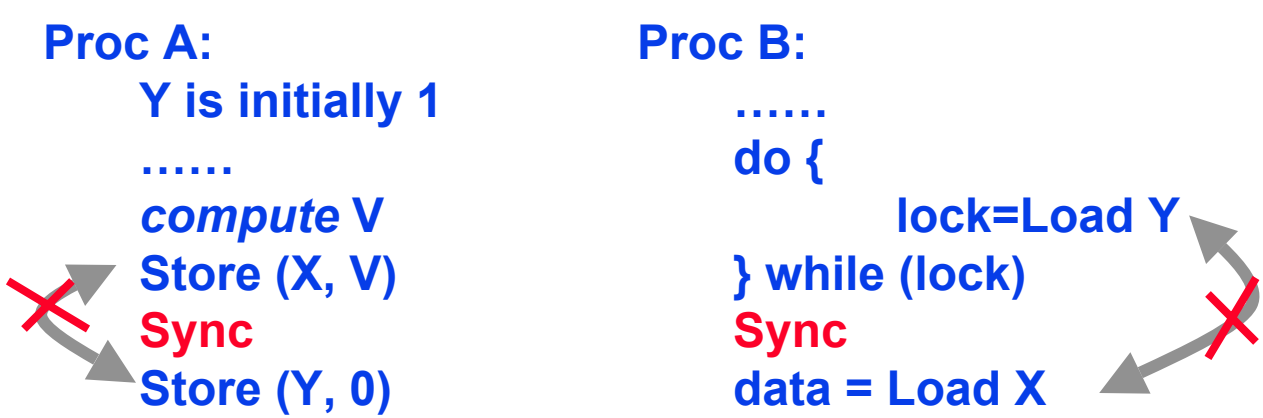
- SC memory model places sever restrictions on the applicability of high-performance memory flow techniques from Lecture 12
- Solutions are
  - disallow reordering of memory operations *(not good enough)*
  - speculatively reorder memory operations and repair if it made an "observable" difference *(MIPS R10000)*

    e.g. allow a load to issue as early as possible without violating uniprocessor dependence. If no other processors issue any memory operations between the time of advanced load and the would be time of the in-order load, then no problem, else "rewind and reload"
  - support weaker memory models *(PowerPC)*
    - Only a small minority of the processors sold will be used in shared-memory systems
    - SC is not always needed even in parallel applications

Weak Consistency (WC)

- By default, WC processors only obey basic uniprocessor memory dependence $R_i(x)<W_j(x)$, $W_i(x)<R_j(x)$, $W_i(x)<W_j(x)$
- A special barrier instruction lets SW explicitly serialize memory operations when it matters

$B_i < R_j(y)$
$B_i < W_j(y)$
$B_i < B_j$
$R_i(x) < B_j$
$W_i(x) < B_j$



- You rarely need to serialize so we can use a low-cost (low-performance) implementation, i.e. on a sync, stop all instructions from issuing until all earlier instructions have finished

Other models, between SC and WC, have been used

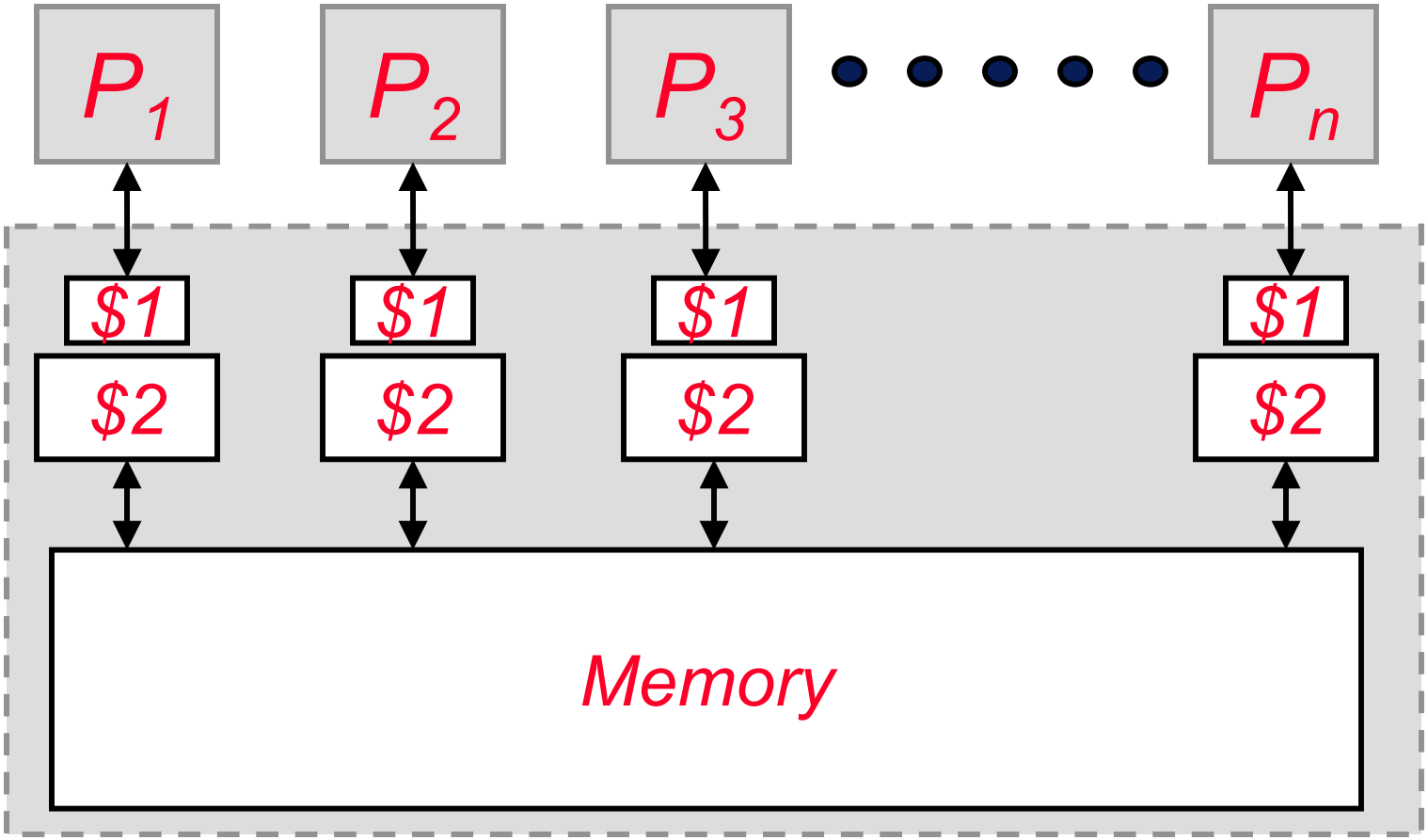
Memory Coherence

- If P1 writes to X, "later" P2 reads X, and no one else writes to X in between, P2 should read the value written by P1
- Example:

```
P1:                          P2:
    Y is initially 1             ……
    ……                           do {
    compute V                            lock=Load Y
    Store (X, V)                 while (lock)
    Sync                         Sync
    Store (Y, 0)                 data = Load X
    ……                           ……
```

- Under WC interpretations, Load X by P2 occurs after Store(X, V) by P1, P2 should get V
- This is absolutely unambiguous if there is only one place where the value of memory location X is kept

Multiprocessor/Cache/Memory Arch



The Goal of Cache Coherence is to make all the processors believe they are connected to the same memory directly?

(warning: slightly oversimplified statement)

Extreme Solutions to Cache Coherence

- Disallow caching of shared variables
- Only allow only one copy of a mem location at a time
  - If location X is cached in one cache then it is not valid in memory or another cache
  - Another processor must have a way to find out who has location X and take over ownership before reading or writing
  - thus, can only have one reader/writer per location
- Allow multiple copies, but make sure they all have the same value at all time
  - update to one copy must be visible to all copies where ever they might be (memory and all of the caches)
  - thus, can have multiple readers and writers at once

A cache coherence protocol is the "rules of conduct" between caches to enforce a particular policy

CC Protocol for Bus-based Systems

- Bus is a broadcast medium, bus "snooping" allows every cache to see what everyone else wants to do
- A cache can even intervene in another cache's bus transaction, e.g. a cache might ask another cache to "retry" the transaction later or respond in place of the memory
- Besides the usual status bits, additional information might have to be recorded with each cache line, aka *cache coherence states*, e.g.
  - Invalid: cache line does not have valid data
  - Modified: cache line has been written to since it was brought in
  - Shared: valid line, but other caches may have copies
    (presumably all identical and unchanged from memory)
  - Exclusive: valid line, unchanged from memory but no other cache has a copy

Example: Multiple Identical Copies

- A cache line can be either Valid or Invalid
- Based on a write-through scheme
  - a cache issues a read transaction on a read or write miss
  - a cache issues a write transaction to memory whenever the cache line is changed by the processor
  - a cache do not need to write back when a line is displaced
- All writes are write-through so the writer's cache is coherent with memory
- All caches "snoop" the bus for other's write transactions
  - Check if the write is to a currently cached location
  - If a write goes to a cached location, overwrite the old (aka stale) value with the new snooped value
  - else do nothing
- A read miss can fetch directly from memory (always current)

Example: One Copy at All Time

- A cache line can be either Valid or Invalid
- Based on a write-back scheme
  - a cache issues a read transaction on a read or write miss
  - a cache issues a writes-back to memory when a line is displaced
- All caches "snoop" the bus for other's read transactions
  - If a cache observes a request to a currently cached line then respond with a value in place of memory, mark its own copy Invalid
  - Alternatively, a cache can also ask the requester to retry later and, in the meanwhile, write-back its copy to memory

Why don't caches need to snoop for write-back transactions?

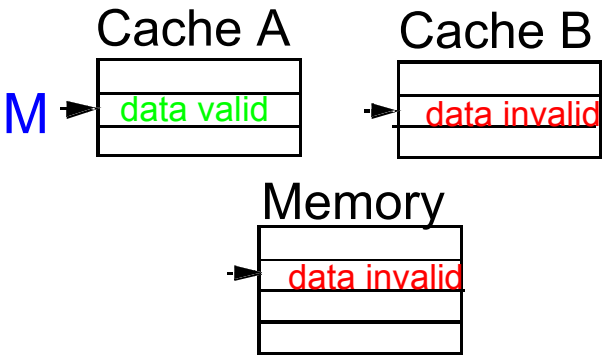
MESI Cache Coherence Protocol

- An efficient policy for single-writer/multi-reader usage
  - Allow multiple read-only copies (all identical) *(Shared)*
  - Allow only a single writable copy *(Exclusive, Modified)*
  - Minimizes the number of bus transactions
- Based on a write-back scheme
  - On a read miss, issue a read transaction for a read-only copy
  - On a write miss, issue a "read-with-intent-to-modify" for an exclusive copy
  - On a write hit to a read-only copy, issue a "invalidate" transaction
  - When displacing a "clean" line, do nothing
  - When displacing a Modified line, write the dirty value back to memory
- All caches "snoop" the bus for other caches' read, RWITM and invalidate transactions

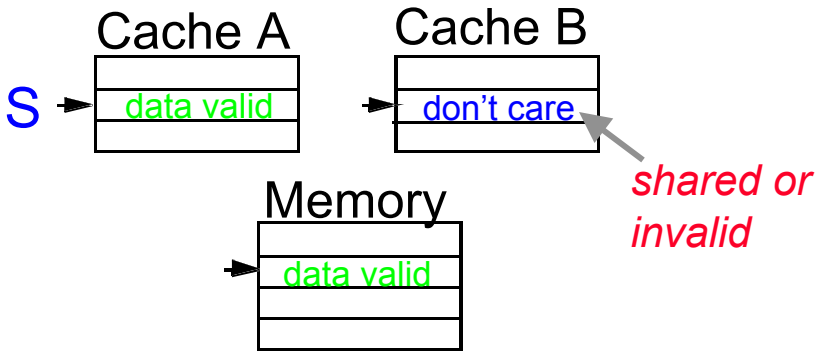
What happens on a snoop hit is kind of complicated to describe in words ...

MESI States

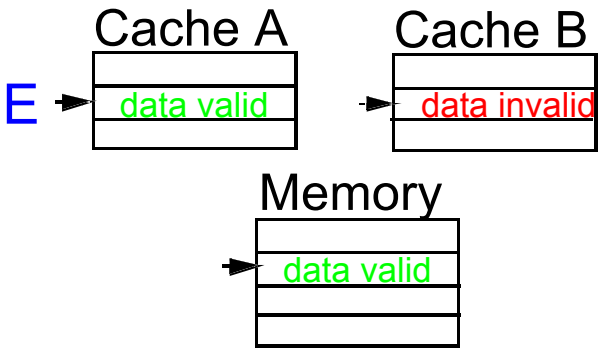
Modified in Cache A



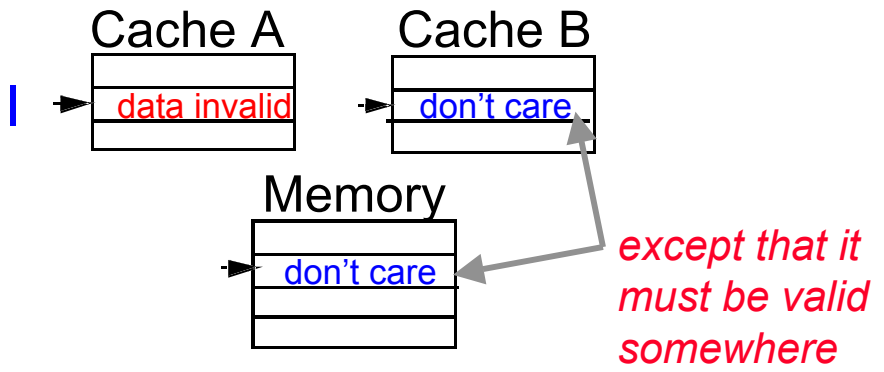
Shared in Cache A



Exclusive in Cache A



Invalid in Cache A



Given the state of an address in one cache, what can one infer about the possible state of the same address else where?