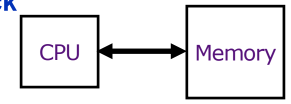
CSE 486/586 Distributed Systems
Cache Coherence

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Storage to Memory

- We've looked at storage consistency.
- The same consistency models are equally applicable to memory.
  - Think multiple threads accessing the same memory addresses
- But a memory system can have another form of consistency mainly for managing caches. We'll look at this today.
  - In a multi-core system, there are many caches, and they need to be synchronized.

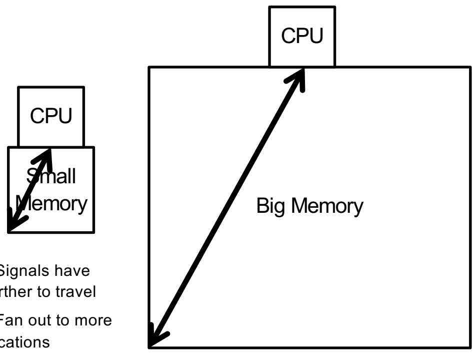
Caching Basics: CPU-Memory Bottleneck



Performance of high-speed computers is usually limited by memory *bandwidth* & *latency*

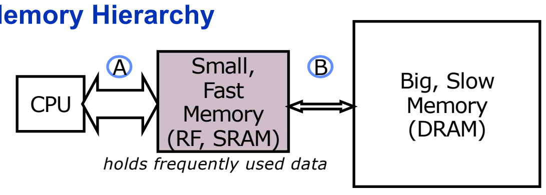
- Latency (time for a single access)
  Memory access time >> Processor cycle time
  Problematic
- Bandwidth (number of accesses per unit time)
  Increase the bus size, etc.
  Usually OK

Physical Size Affects Latency



- Signals have further to travel
- Fan out to more locations

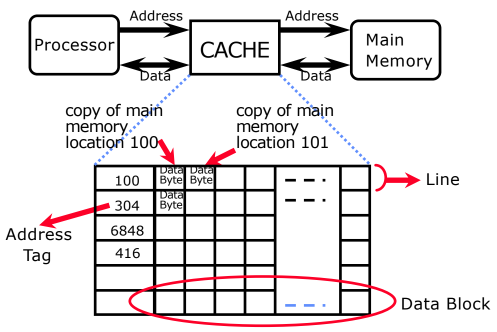
Memory Hierarchy



- *capacity*: Register << SRAM << DRAM *(cost)*
- *latency:* Register << SRAM << DRAM *(size)*
- *bandwidth:* on-chip >> off-chip *(delays)*

On a data access:
if data ∈ fast memory ⇒ low latency access *(SRAM)*
If data ∉ fast memory ⇒ long latency access *(DRAM)*

Inside a Cache



Cache Read

Look at Processor Address, search cache tags to find match. Then either

Found in cache
a.k.a. HIT

Return copy of data from cache

Not in cache
a.k.a. MISS

Read block of data from Main Memory

Wait …

Return data to processor and update cache
(Use a replacement algorithm to select a line to replace)

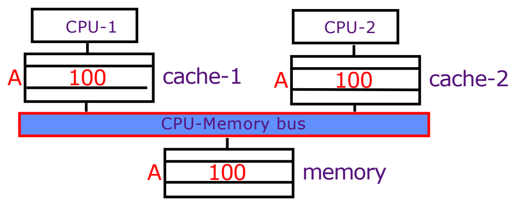
Cache Write

- Cache hit:
  - ***write through:*** write both cache & memory
  - ***write back:*** write cache only, memory is written only when the entry is evicted
- Cache miss:
  - ***no write allocate:*** only write to main memory
  - ***write allocate*** *(aka fetch on write):* fetch into cache
- Common combinations:
  - write through and no write allocate
  - write back with write allocate

Administrivia

- PA3 grading still going on
- This Friday, no recitation, undergrad office hours from 2 pm – 4 pm & general office hours from 4 pm – 5 pm

Memory Coherence in SMPs



Suppose CPU-1 updates A to 200.
write-back: memory and cache-2 have stale values
write-through: cache-2 has a stale value

Do these stale values matter?
What kind of guarantee do you get?

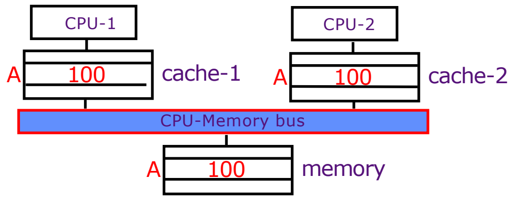
Cache Coherence

- A cache coherence protocol ensures that all writes by one processor are eventually visible to other processors
  - i.e., updates are not lost
  - You can consider this a hardware-based update propagation mechanism for distributed caches.
- Hardware support is required such that
  - only one processor at a time has write permission for a location
  - no processor can load a stale copy of the location after a write

Cache Coherence

- A memory system is coherent if:
- A read by a processor P to a location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P.
- A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.
- Writes to the same location are serialized; that is, two writes to the same location by any two processors are seen in the same order by all processors.
- (Coherence provides per-location sequential consistency).

One Design: Snoopy Cache



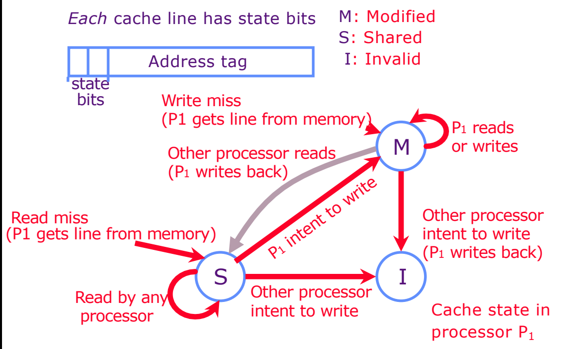
- Cache controllers work together to maintain cache coherence.
- Each cache controller snoops on the bus traffic and "do the right thing."

Snoopy Cache Coherence Protocol

- For a read operation, a cache line is shared across multiple caches.
- For a write operation, a cache line is not shared.
- Each cache line has a state:
  - M (modified): the processor has written to it.
  - S (shared): other caches have a copy as well.
  - I (invalid): the data is no longer valid.
- Writing to a cache line
  - If it's M, the cache controller does the write.
  - If it is not M, it sends an invalidation request to other caches, switches the state to M, and does the write.
  - Other cache controllers switch the state to I.
- Reading a memory address
  - If it's a hit, read it.
  - If it's not a hit, read it from memory, and other cache controllers switch the state to S.

Cache State Transition Diagram

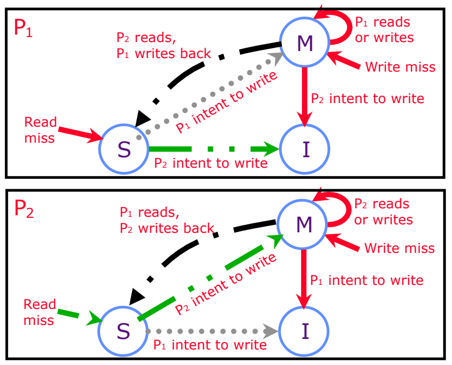
The MSI protocol


Each cache line has state bits
M: Modified
S: Shared
I: Invalid
Address tag
state bits
Write miss (P1 gets line from memory)
P1 reads or writes
Other processor reads (P1 writes back)
M
P1 intent to write
Read miss (P1 gets line from memory)
Other processor intent to write (P1 writes back)
S
I
Read by any processor
Other processor intent to write
Cache state in processor P1

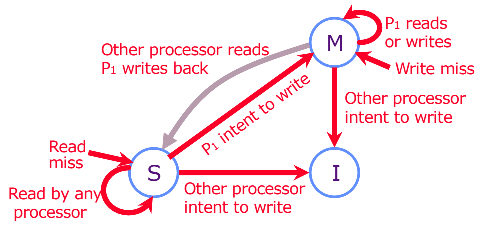
Two Processor Example

(Reading and writing the same cache line)

P_1 reads
P_1 writes
P_2 reads
P_2 writes
P_1 reads
P_1 writes
P_2 writes
P_1 writes



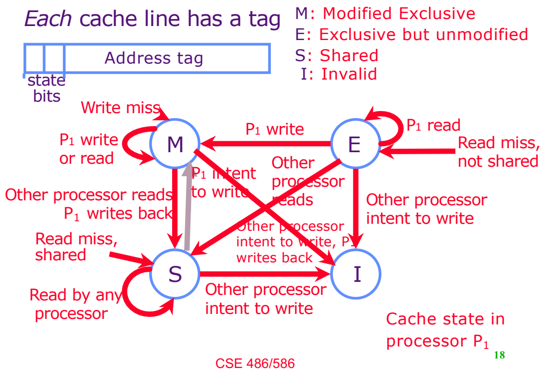
Observation


M
P1 reads or writes
Other processor reads P1 writes back
Write miss
P1 intent to write
Other processor intent to write
Read miss
S
I
Read by any processor
Other processor intent to write

- If a line is in the M state then no other cache can have a copy of the line!
  - Memory stays coherent, multiple differing copies cannot exist

MESI: An Enhanced MSI protocol

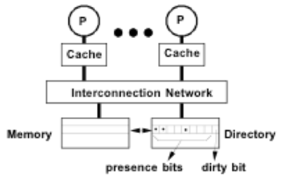
increased performance for private data


Each cache line has a tag
M: Modified Exclusive
E: Exclusive but unmodified
S: Shared
I: Invalid
Address tag
state bits
Write miss
P1 write or read
M
P1 write
E
P1 read
Read miss, not shared
P1 intent to write
Other processor reads
Other processor reads P1 writes back
Other processor intent to write
Other processor intent to write, P1 writes back
Read miss, shared
S
I
Read by any processor
Other processor intent to write
Cache state in processor P1

Scalable Approach: Directories

- Every memory block has associated directory information
  - keeps track of copies of cached blocks and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - in scalable networks, communication with directory and copies is through network transactions
- Many alternatives for organizing directory information

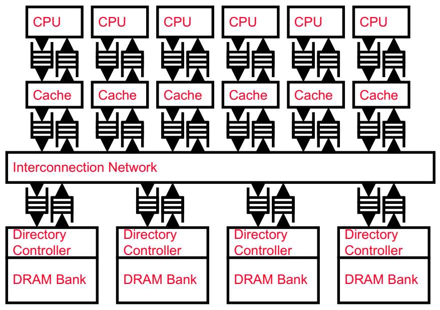
Basic Operation of Directory



- k processors.
- With each cache-block in memory: k presence-bits, 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

- Read from main memory by processor i:
  - If dirty-bit OFF then { read from main memory; turn p[i] ON; }
  - if dirty-bit ON then { recall line from dirty proc (downgrade cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i;}
- Write to main memory by processor i:
  - If dirty-bit OFF then {send invalidations to all caches that have the block; turn dirty-bit ON; supply data to i; turn p[i] ON; ... }

Directory Cache Protocol



- Assumptions: Reliable network, FIFO message delivery between any given source-destination pair

Cache States

For each cache line, there are 4 possible states:

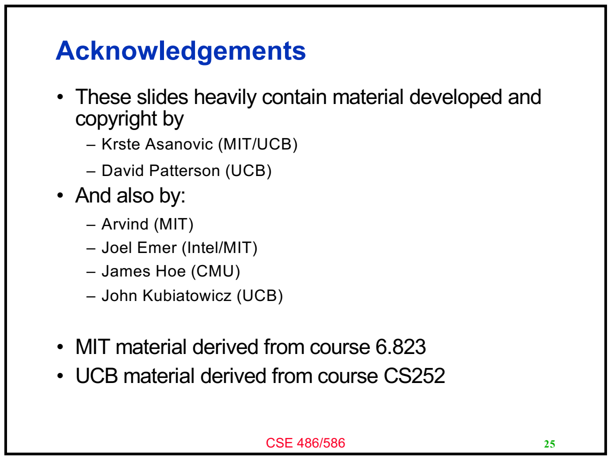
- C-invalid (= Nothing): The accessed data is not resident in the cache.
- C-shared (= Sh): The accessed data is resident in the cache, and possibly also cached at other sites. The data in memory is valid.
- C-modified (= Ex): The accessed data is exclusively resident in this cache, and has been modified. Memory does not have the most up-to-date data.
- C-transient (= Pending): The accessed data is in a *transient* state (for example, the site has just issued a protocol request, but has not received the corresponding protocol reply).

Home directory states

- For each memory block, there are 4 possible states:
  - R(dir): The memory block is shared by the sites specified in dir (dir is a set of sites). The data in memory is valid in this state. If dir is empty (i.e., dir = ε), the memory block is not cached by any site.
  - W(id): The memory block is exclusively cached at site id, and has been modified at that site. Memory does not have the most up-to-date data.
  - TR(dir): The memory block is in a transient state waiting for the acknowledgements to the invalidation requests that the home site has issued.
  - TW(id): The memory block is in a transient state waiting for a block exclusively cached at site id (i.e., in C-modified state) to make the memory block at the home site up-to-date.

Summary

- Cache coherence
  - Making sure that caches do not contain stale copies.
- Snoopy cache coherence
  - MSI
  - MESI
- Directory-based
  - Uses a directory per memory bank



Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252