Computer Architecture: Memory Coherence

Berk Sunar and Thomas Eisenbarth

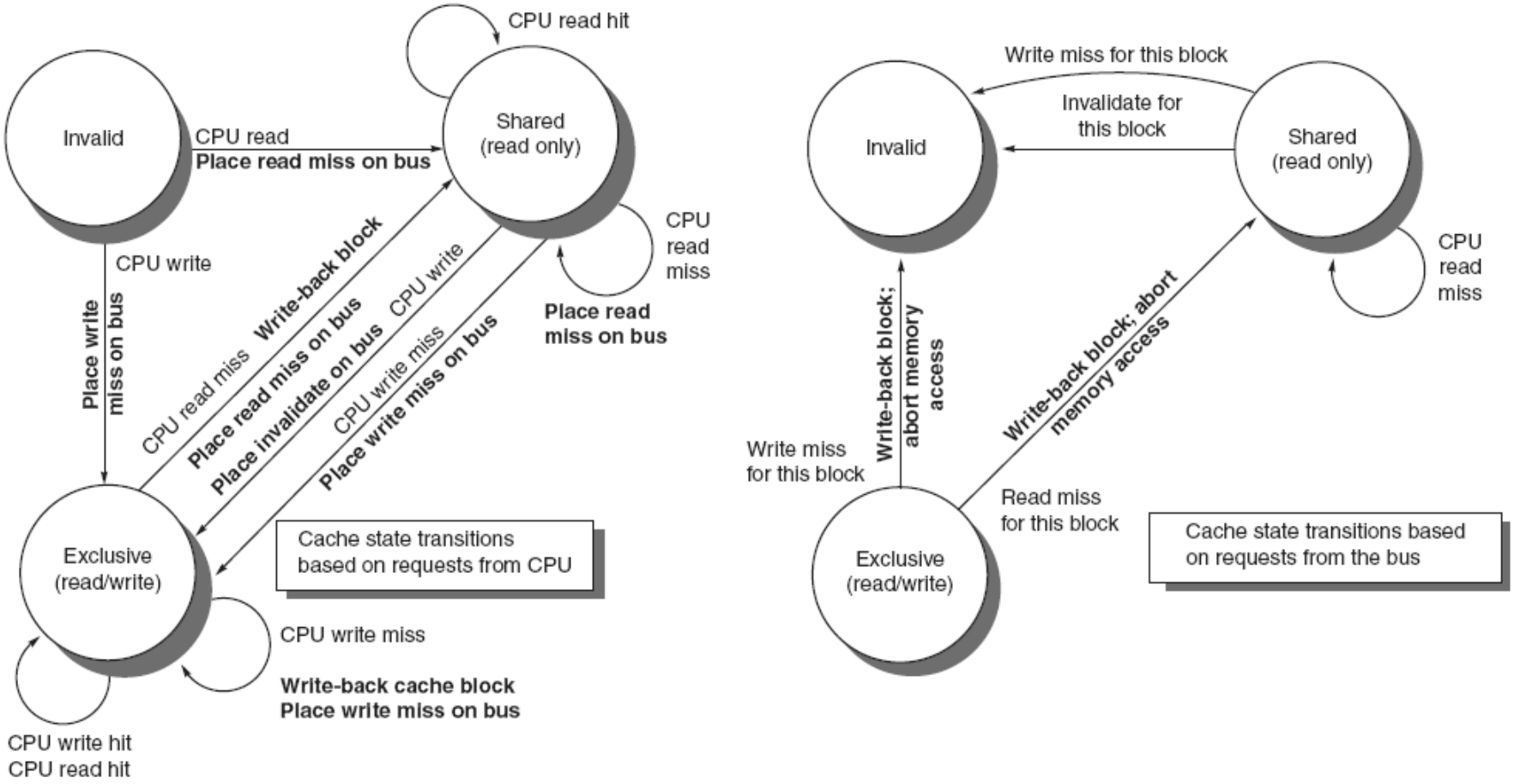
ECE 505

WPI

Snoopy Coherence Protocol Implementation: Example

- Each core has a FSM to implement snooping
  - Interacts with core and bus
- Both regular cache and snoop need to check cache address tags: duplication of checking HW
- Protocol with three states: invalid, shared, modified (latter implies private/exclusive)

Snoopy Coherence Protocol Implementation: Example



Simplifying assumptions

- Protocol is *atomic*: no intervening operations can occur during operation execution
  - In reality: write miss detected, then acquire bus, then receive response
  - Nonatomic operations can result in deadlock:
    - Dining Philosophers problem
- Protocol only solves on-chip communication
  - Many modern chips support connection of several multiprocessor chips. That connection uses different protocol

Coherence Protocol Extensions

Basic Protocol: MSI (for Modified, Shared, Invalid)

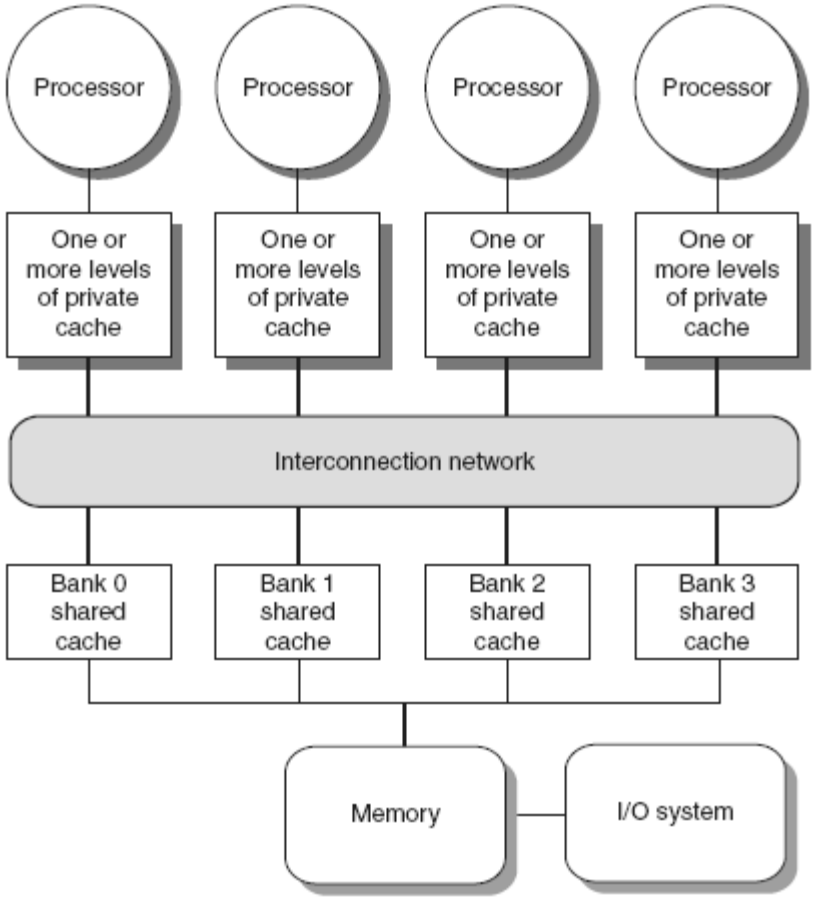
1. MESI: New Exclusive state if block only in one cache and clean→ write without invalidate
   - i7 uses MESIF: Forward state indicates which processor of shared state should respond
2. MOESI: Owned state indicates cache *owns block*, i.e. outdated in memory (writeback required on miss); used by AMD Opteron

Coherence Protocols: Extensions

Shared memory bus and snooping bandwidth are bottleneck for larger symmetric multiprocessors

Solutions:

- Place directory in outermost cache (i7)
- Use crossbars or point-to-point networks with banked memory



Implementing Snooping Cache Coherence

- Bus is usually used to make *write misses* atomic
  - Core blocks bus until other cores received invalidates
- Larger multicores (>8) use interconnect instead of shared bus
  - Coherence must be ensured without bus-enforced serialization (miss must be atomic)
  - Races (two cores writing simultaneously) must be handled by protocol (only one winner allowed)
- Snooping and directories are sometimes combined, especially one within chip and the other between chips

Performance

Coherence influences cache miss rates

- Coherence misses (fourth C)
  1. True sharing misses: directly arise from sharing
     - Write to shared block (transmission of invalidation)
     - Read an invalidated block
  2. False sharing misses: due to invalidation of block, though word unchanged
     - Read an unmodified word in an invalidated block
     - Occur only due to block size

Coherence Misses: Example

- Words x1 and x2 are in same cache block, shared state, in caches of core P1 and P2

Time	P1	P2
1	Write x1	
2		Read x2
3	Write x1	
4		Write x2
5	Read x2	

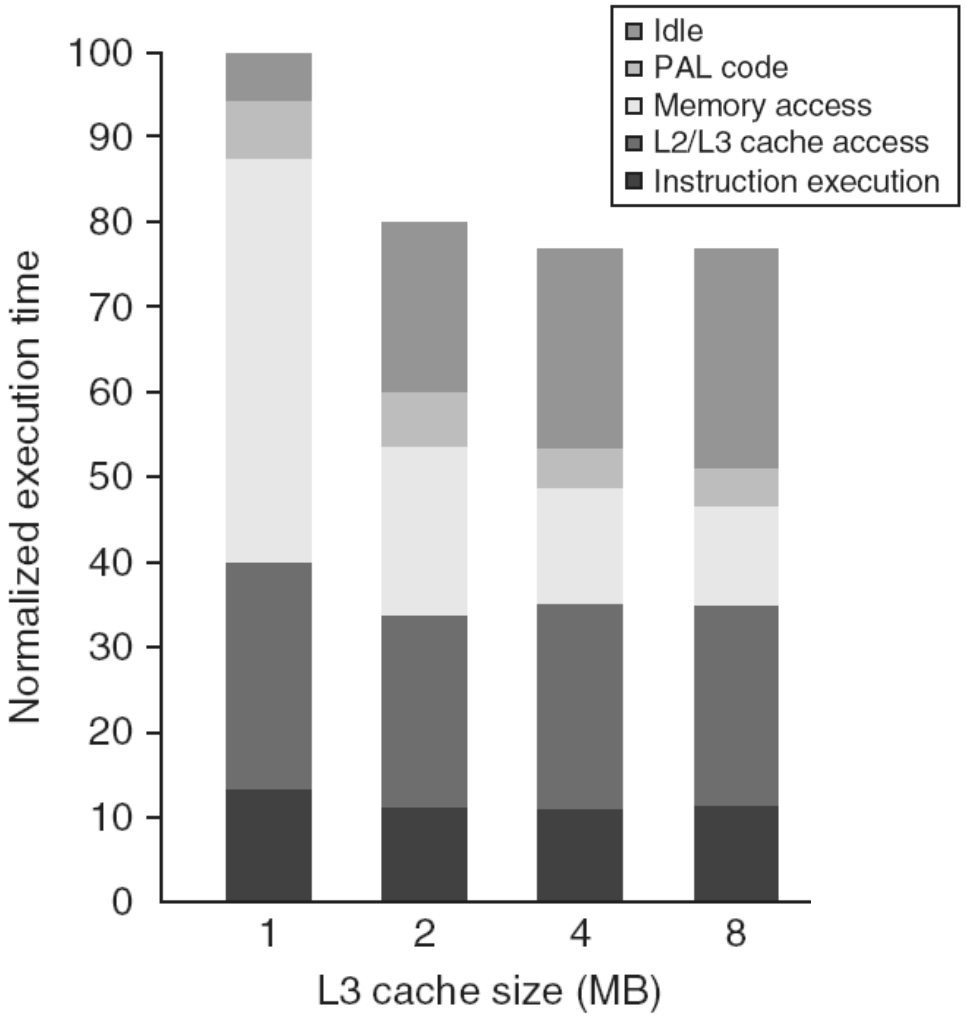
1. True sharing miss: x1 in P2's $→needs to be invalidated by P2
2. False sharing miss: x2 invalidated by prev. write, though unchanged
3. False sharing miss: x1 marked shared due to prev. read. Write miss gives exclusive access
4. False sharing miss: as 3.
5. True sharing miss: read value was written by P2

Performance Study: Commercial Workload

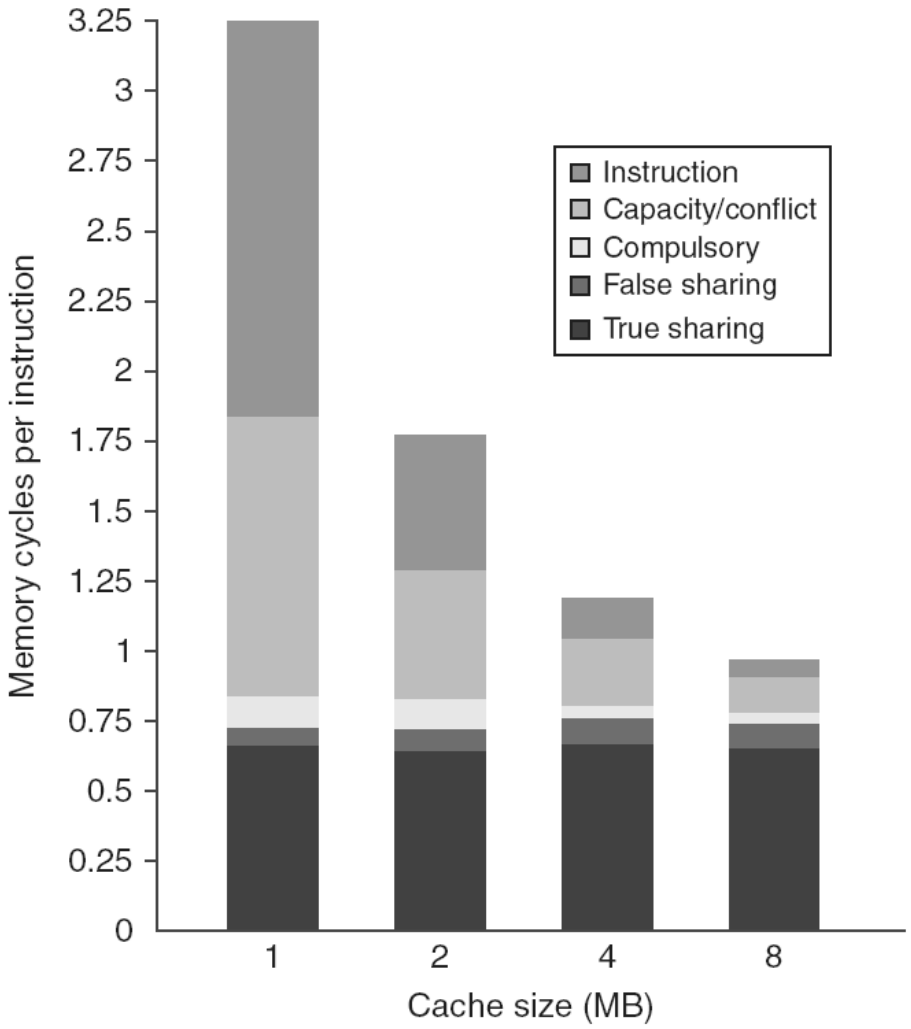
Scenario:

- 4-core SMP running general-purpose commercial workload
- Alpha system, done in 1998 (4 IPC per core @300MHz, 3-L $)

- Why the jump from 1 to 2 MB of L3 $?

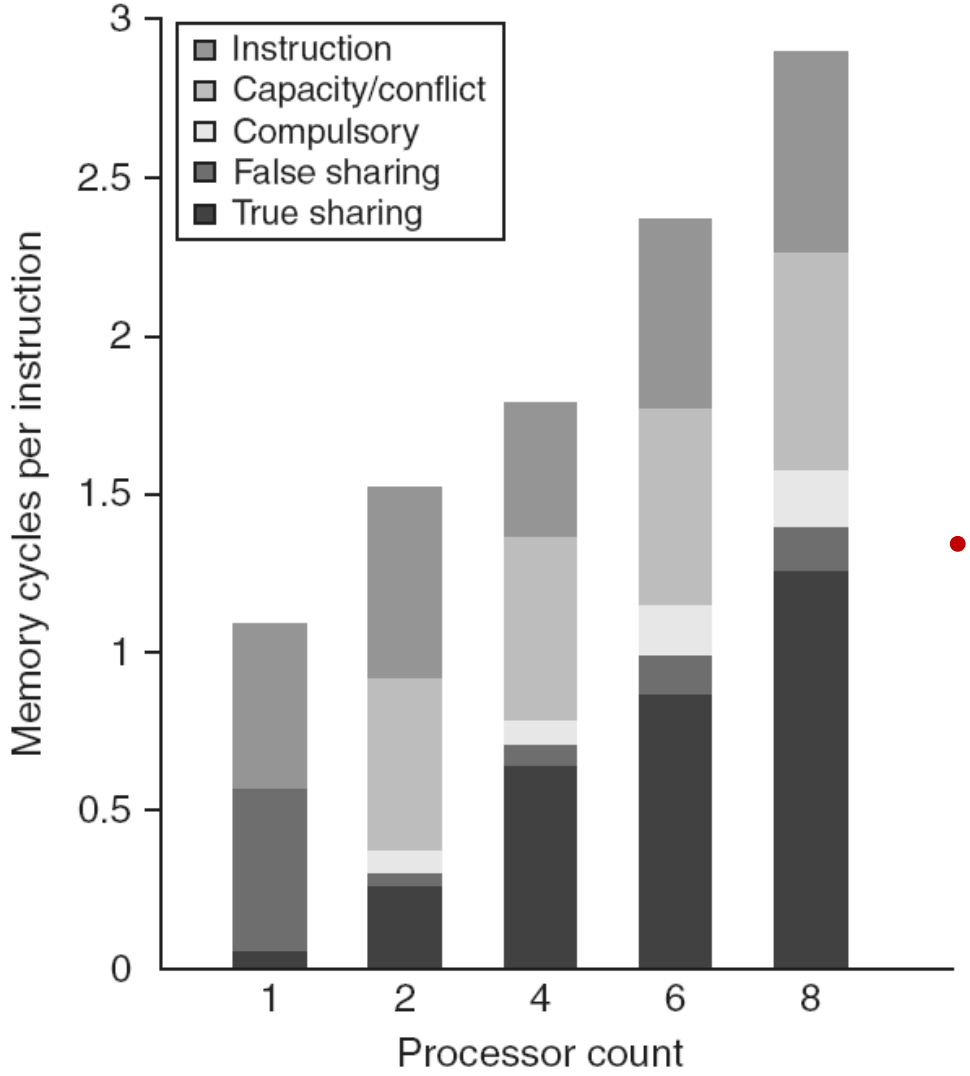


Performance Study: Commercial Workload



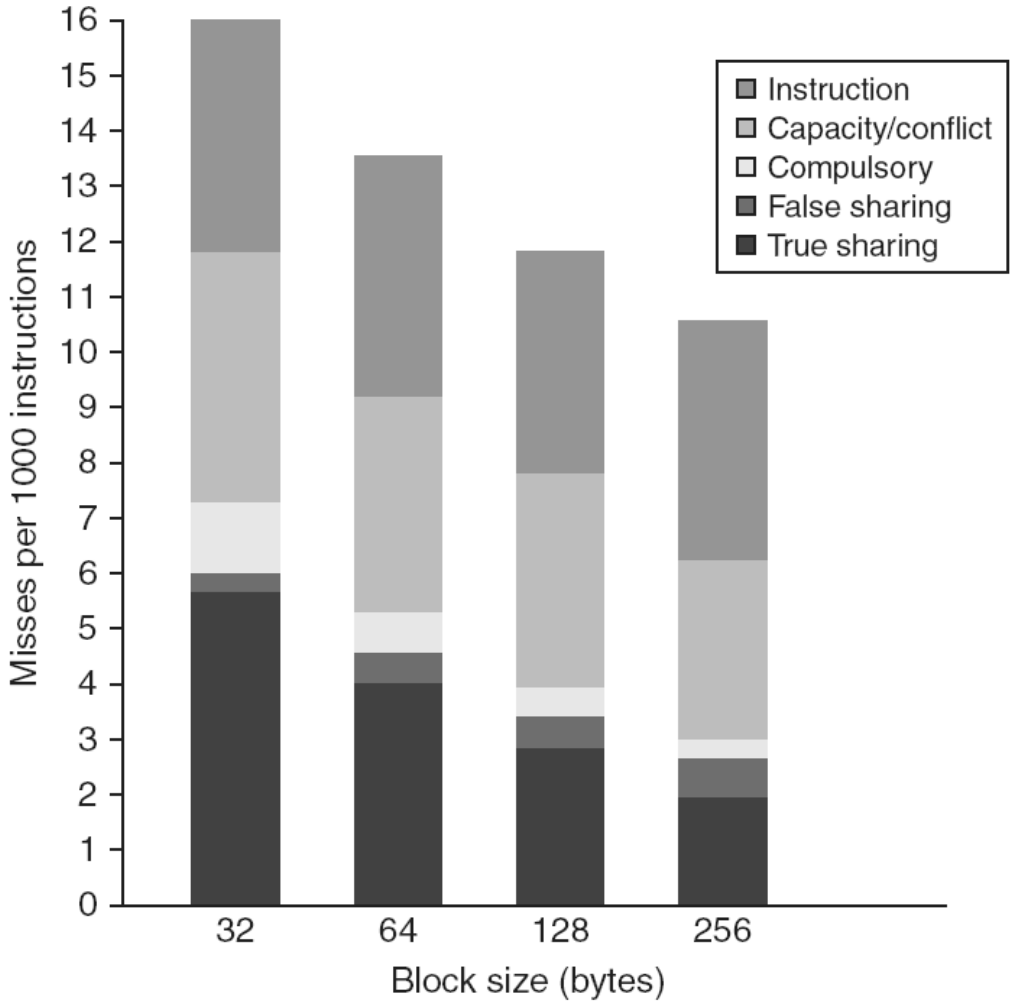
- Increasing cache size decreases Uniprocessor misses, but not multiprocessor misses

Performance Study: Commercial Workload



- Increasing processor count increases multiprocessor misses (mainly true sharing misses)

Performance Study: Commercial Workload



Larger block size:

- Decreases true sharing miss rate significantly
- Compulsory misses also significantly decreased
- Conflict/capacity slightly decreased: spatial locality in L3 not high any more
- False sharing misses increase significantly (but still small number)

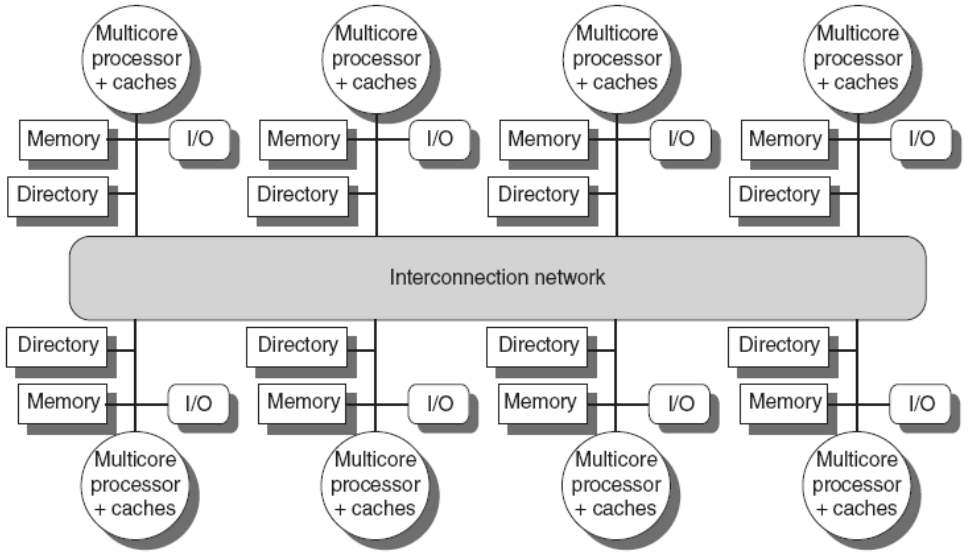
Directory Coherence: Motivation

- Snoopy protocols require broadcast for every cache miss
  - Requires large bus bandwidth, O(N)
  - Requires large cache snooping bandwidth
- Directory protocols enable further scaling
  - Directory can track all caches holding a memory block and use point-to-point messages to maintain coherence
  - Communication done via scalable point-to-point interconnect

Directory Protocols

- Directory keeps track of every block
  - Which caches have each block
  - Dirty status of each block
- Implement in shared L3 cache (e.g. in i7)
  - Keep bit vector of size = # cores for each block in L3
  - Send invalidates only to cores that share block
  - Not scalable beyond shared L3
- Solution: Implement in a distributed fashion:
  - Sharing status in single known location (e.g. with memory)
  - Directory size: # blocks x # directories
  - ➢ Avoids broadcast

Might not be sufficient for Supercomputers with 1000s of chips



Directory Protocols: Basics

- Primary operations: handle
  - read miss;
  - write to shared clean block
- For each block, maintain state (cf. MSI protocol):
  - Shared: One or more nodes have the block cached, value in memory is up-to-date;
    - Set of node IDs
  - Uncached: not in any cache
  - Modified: Exactly one node has a copy of the cache block, value in memory is out-of-date
    - Owner node ID
- Directory maintains block states and sends invalidation messages
  - Physical address space is statically distributed: directory and node containing data identified by address

Messages

Local Node to Home Dir

Local Dir to Remote Node

Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Node P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Node P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	Local cache	Home directory	A	Request to send invalidates to all remote caches that are caching the block at address A.
Invalidate	Home directory	Remote cache	A	Invalidate a shared copy of data at address A.
Fetch	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
Fetch/invalidate	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache.
Data value reply	Home directory	Local cache	D	Return a data value from the home memory.
Data write-back	Remote cache	Home directory	A, D	Write-back a data value for address A.

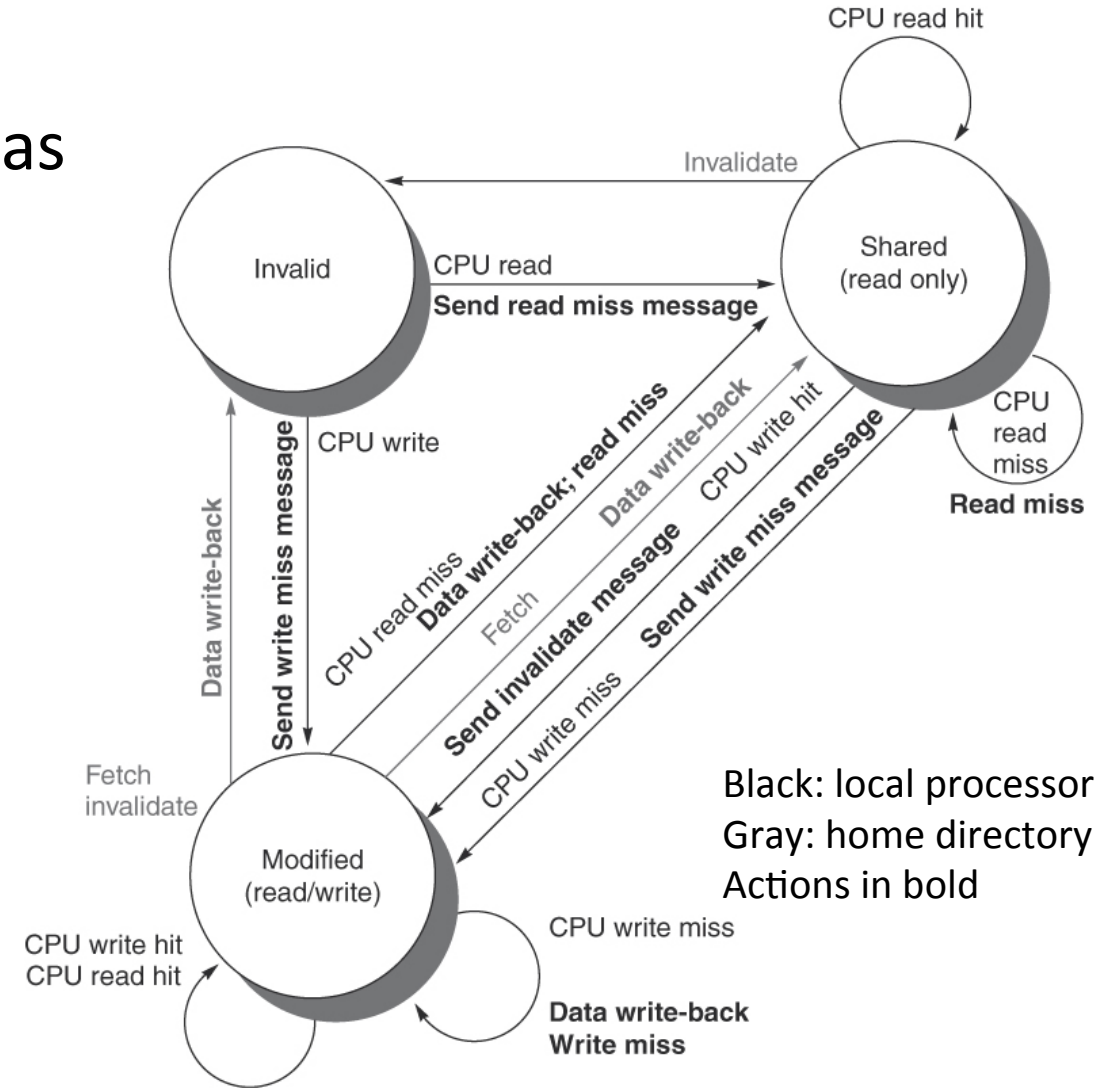
Request originates from *local node*
Memory entry and directory reside at *home node*
Copies may exist in third nodes: *remote node*

P: requesting node number
A: requested address
D: data contents

Cache State Transition Diagram

- Each cache block has state bits:
  - M: Modified
  - S: Shared
  - I: Invalid

state bits



Basic Directory State

State	Sharers/Owner
S	10110100
U	xxxxxxxx
E	00010000
S	00100100
…	

- State: state of the cache block (Shared, Uncached, Exclusive)
- Sharers/Owner:
  either bit vector of all nodes that share copy of block (if shared) or
  which node has exclusive copy (if Exclusive)

Directory Operations

- For uncached block:
  - Read miss
    - Requesting node is sent the requested data and is made the only sharing node, block is now shared
  - Write miss
    - The requesting node is sent the requested data and becomes the sharing node, block is now exclusive
- For shared block:
  - Read miss
    - The requesting node is sent the requested data from memory, node is added to sharing set (kept by bit vector)
  - Write miss
    - The requesting node is sent the value, all nodes in the sharing set are sent invalidate messages, sharing set only contains requesting node, block is now exclusive

Directory Operations (2)

- For exclusive block:
  - Read miss
    - The owner is sent a data fetch message, block becomes shared, owner sends data to the directory, data written back to memory, sharers set contains old owner and requestor
  - Data write back
    - Block becomes uncached, sharer set is empty
  - Write miss
    - Message is sent to old owner to invalidate and send the value to the directory, requestor becomes new owner, block remains exclusive

Directory State Transition Diagram