EN2910A: Advanced Computer Architecture
Topic 05: Coherency of Memory Hierarchy

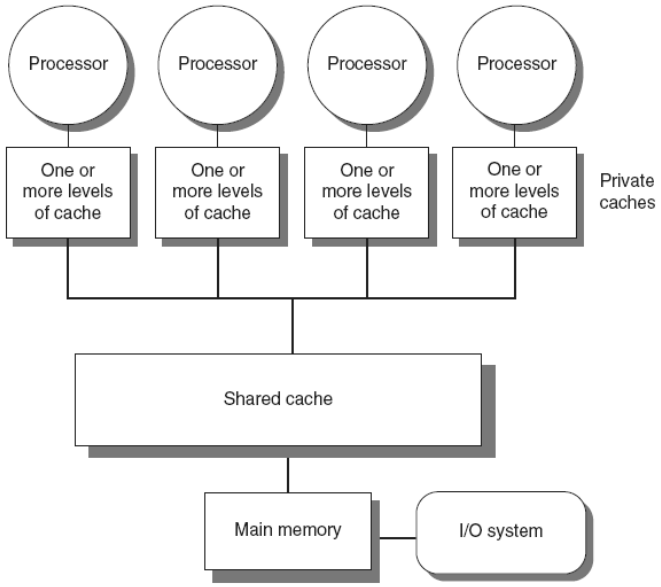
Prof. Sherief Reda
School of Engineering
Brown University

Material from:

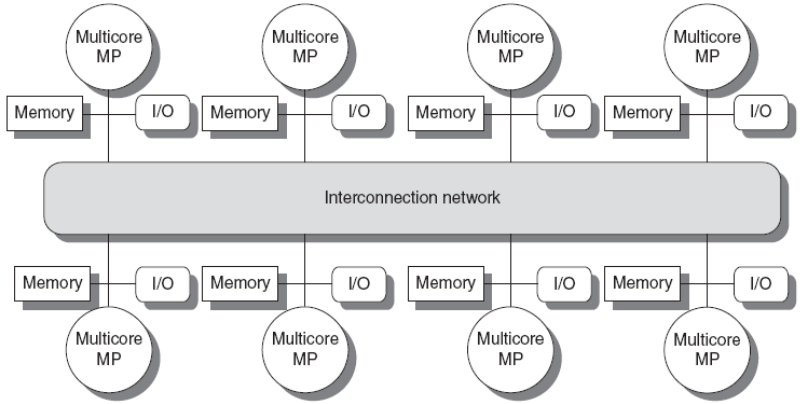
- *Parallel Computer Organization and Design* by Debois, Annavaram and Stenstrom
- *Computer Architecture: A Quantitative Approach* by Hennessy and Patterson

Cache coherency in computer systems

- All reads by any processor must return the most recently written value
- Writes to the same location by any two processors are seen in the same order by all processors



Multi-core systems

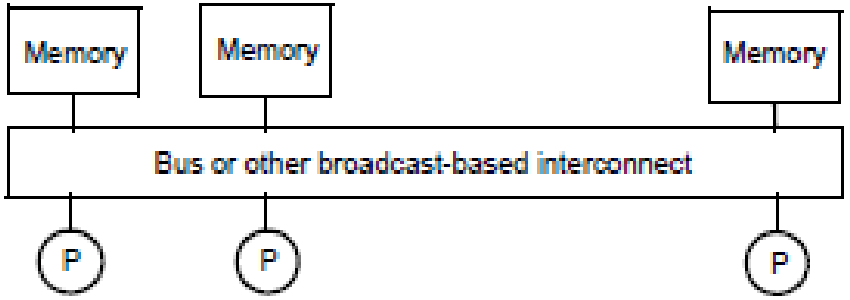


Multi-core multiprocessor systems (NUMA)

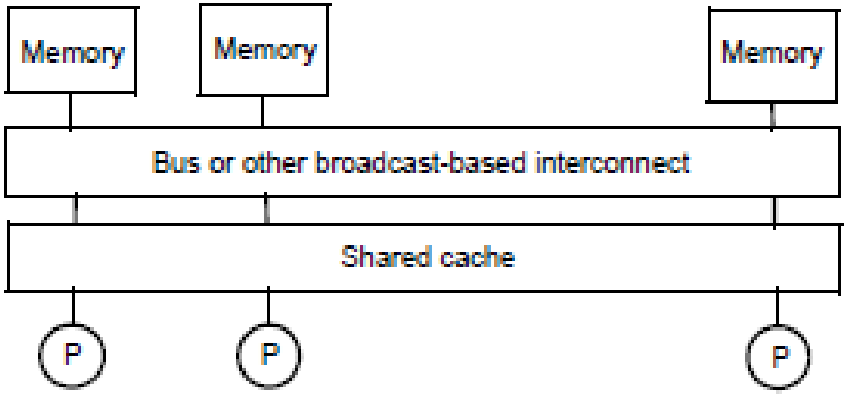
Enforcing cache coherency

- Coherent caches provide:
  - *Migration*: movement of data
  - *Replication*: multiple copies of data
- Cache coherence protocols
  1. Snooping for bus-based systems: each core tracks sharing status of each block
  2. Directory based: Sharing status of each block kept in one location
- Enforcing cache coherency is a necessity that comes at the expense of increased HW complexity, increase cache access time and reduce cpu-memory traffic BW.

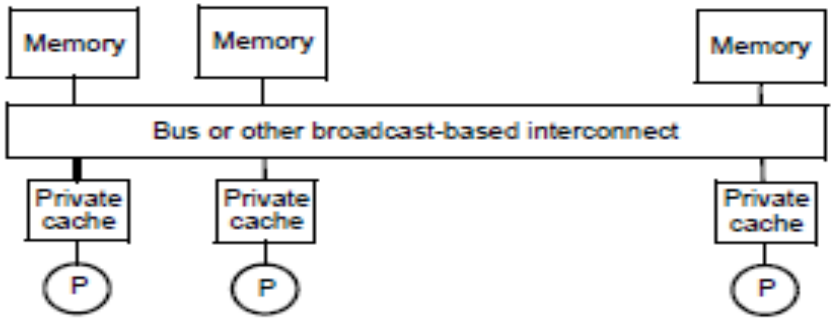
1. Bus-based shared memory systems



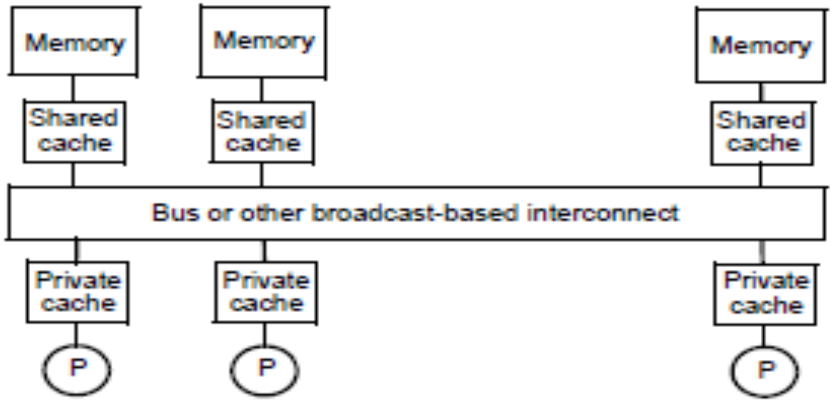
(a) Dance-hall multiprocessor architecture or SMP



(b) SMP with shared level 1 cache



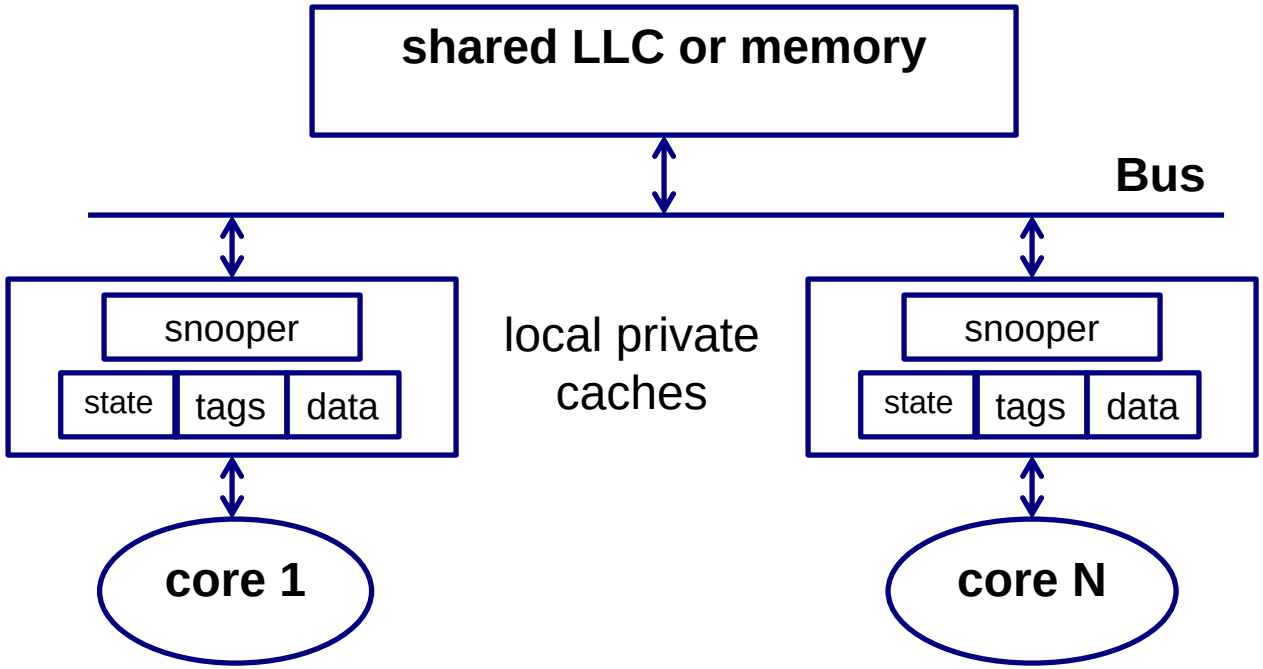
(c) SMP with private caches



(d) SMP with private caches and shared Level 2 cache

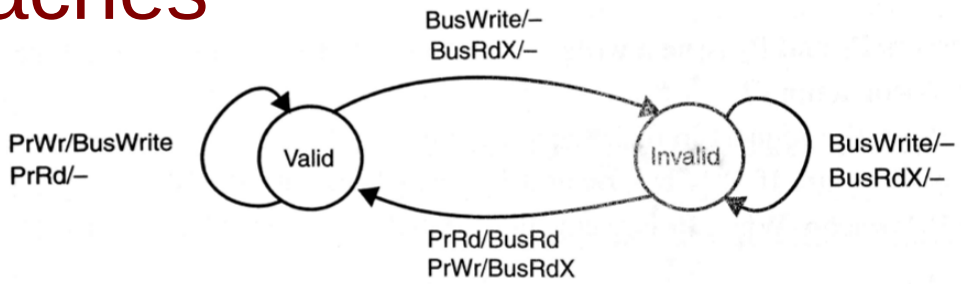
Key requirement: ability to broadcast-based interconnect

Snooping-based cache coherence



- Transactions on the bus are “visible” to all processors.
- Bus interface can “snoop” (monitor) the bus traffic and take action when required.
- To take action the “snooper” must check the status of the cache.
- Snoop controller interferes with CPU access → sometimes cache tags are duplicated to enable concurrent accesses.

A simple coherency protocol for write-through caches



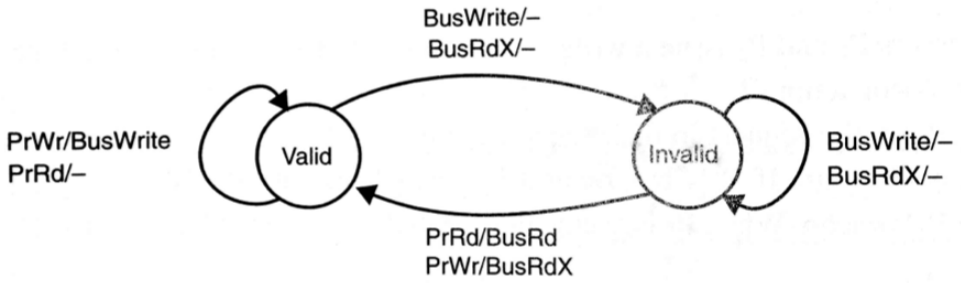
Cache controller receives requests from:

1. Processor (Pr): Read (PrRd) and Write (PrWr)
2. Bus (Bus): Read (BusRd), Read Exclusive (BusRdX) and write (BusWrite)

What happens in the following scenarios?

1. A read miss: controller acquires the bus; BusRd request placed; a copy of the block is returned from memory; release the bus
2. A write hit: acquire the bus; place BusWr → updates the main memory (or shared LLC) and invalidate matching remote caches; update local cache; release the bus
3. A write miss: acquire the bus; place BusRdX → block brought from memory (or shared LLC) and invalidate other cache copies; write to block and main memory (or shared LLC); release bus

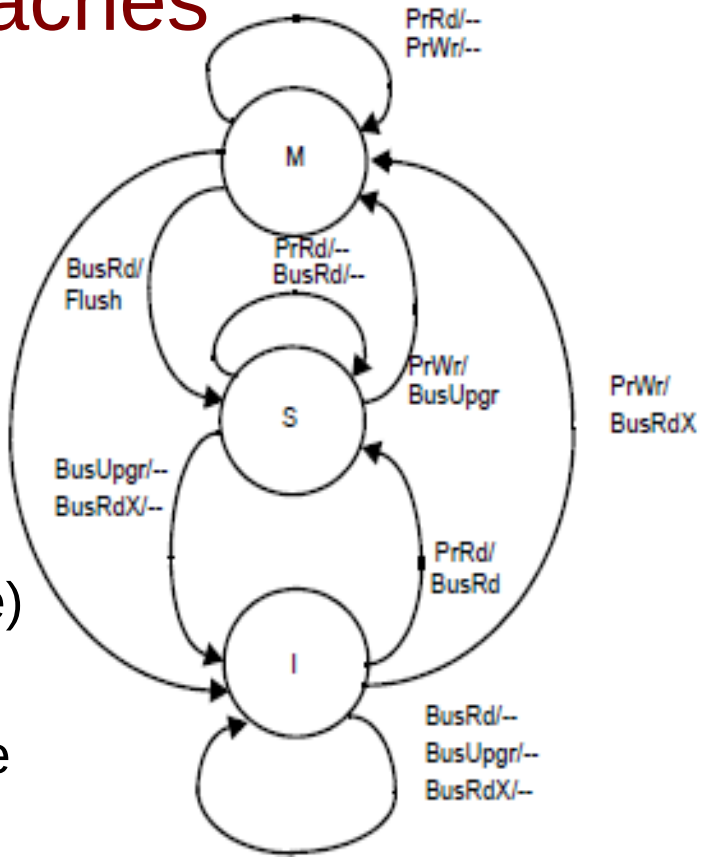
Simple protocol subtleties



- Suppose that two processors P1 and P2 issue a write request to a block A that is resident in both caches. What are the protocol actions?
- What is impact of the protocol on cache memory access time?
- Write through generally creates large bus traffic
- Coherency protocol consume additional unnecessary bus traffic by broadcasting invalidations for writes to unshared blocks.

MSI protocol for write-back caches

- Block states:
  - Invalid (I);
  - Shared (S): one copy or more, memory is clean;
  - Modified (M) or dirty: one copy, memory is stale
- Processor requests: same like before
- Bus transactions: BusRd, BusRdX (as before)
  - BusUpgr: invalidate remote copies
  - Flush: supply a block to a requesting cache

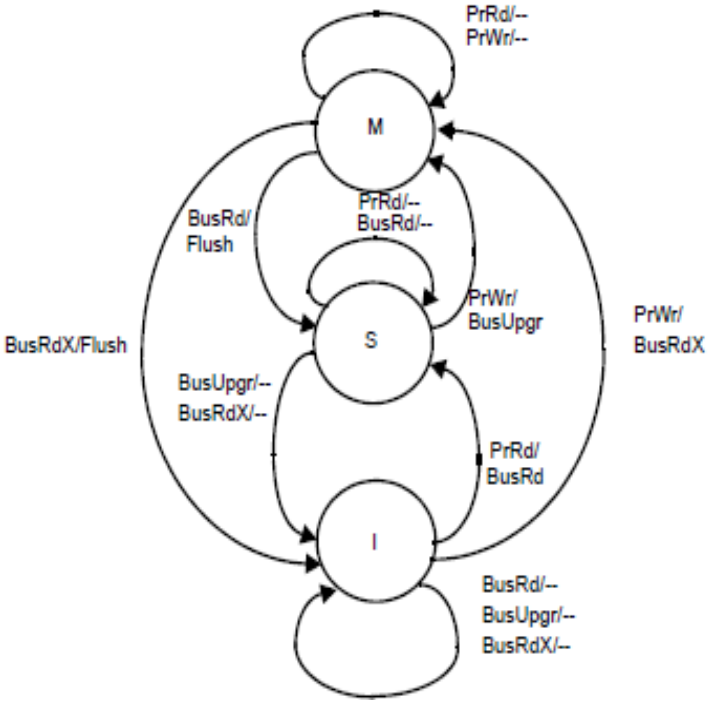


Key differences from simple protocol:

- Consider a read followed by a subsequent write, how are subsequent writes handled?
- Consider a read following by a write of block X by P1. What happens if P2 reads block X?

Issues with MSI protocol

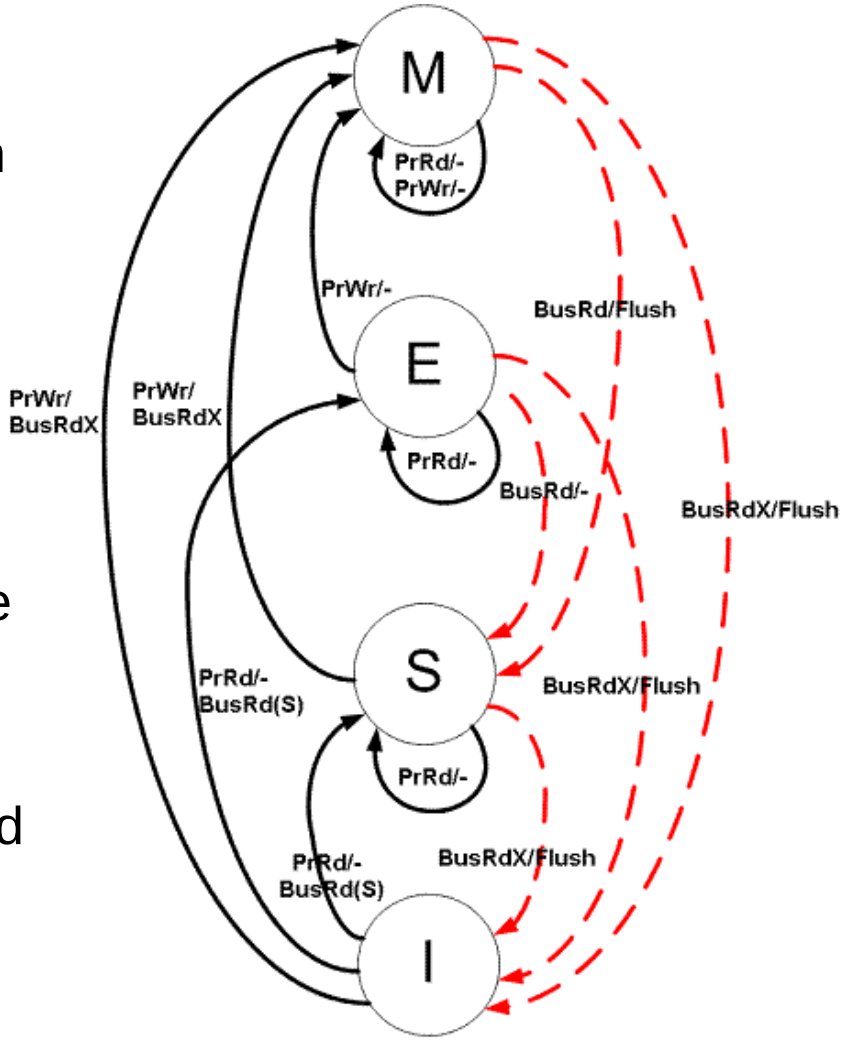
Processor 1	Processor 2	Processor 3
R_A		
W_A		
W_A		
	R_A	
		R_A



- Who satisfies the read by processor 3? How is the handling of the read misses of processor 2 and processor 3 different?
- Was the invalidation caused by the bus upgrade of the first W_A necessary? Can we eliminate it?
- Can we send updates instead of invalidates? What are pros and cons?

MESI Protocol

- Adds an exclusive state (E) → one copy and memory is clean.
- On a miss, go to E or S depending on sharing status.
- Notice no need for invalidation when transitioning from E to M.
- Assume Block X is shared in two caches, what would be its status in cache 1 if it is in state M or E in cache 2?
- When a block is in M or E and moves to S, memory assumes ownership and supplies future reads → could be faster if we continue to flush from cache → who's job?



True and false sharing

- True sharing arises from the communication of data through the cache coherence mechanism.
- False sharing arises from the use of invalidation-based coherent algorithms because of the use of a single valid bit per cache block where the block has multiple words.
- Example: assume words x1 and x2 are in the same cache block, which is shared in the caches of both P1 and P2. Identify each miss as a true sharing miss or a false share miss.

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

Synchronization

- Two processors sharing an area of memory
  - P1 writes, then P2 reads
  - Data ***race*** if P1 and P2 don't synchronize
    - Result depends of order of accesses
- Example: two threads want to increment a global variable

- Mutual exclusion (i.e., *mutex*) algorithms are used in parallel programming to prevent the simultaneous use of a common resource, such as a global variable, by code called ***critical sections.***

Hardware support for mutex

- Hardware support required
  - Atomic read/write memory operation
  - No other access to the location allowed between the read and write
- Could be a single instruction
  - E.g., atomic swap of register ↔ memory
  - Or an atomic pair of instructions

ISA and HW support for mutex

X86 exchange with a single instruction

- Uses a single instruction to conditionally exchange a register and memory location
- Memory controller must initiate read-modify-write as one transaction to lock memory from use by other controllers.

```
        mov     eax, 1
try:    exch    eax, lock_loc
        cmp     eax, 0
        jnz     try
```

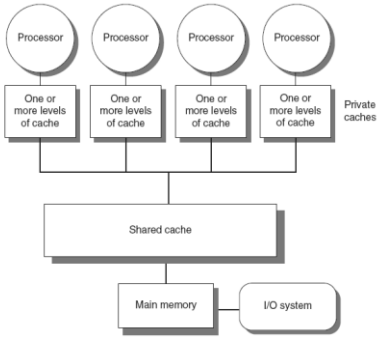
Spinning locks

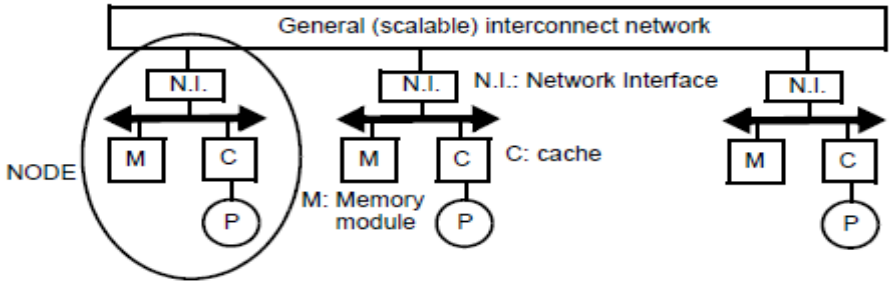
- Spin lock

```
lockit:   MOV     R2,#1
          EXCH    R2,0(R1)    ;atomic exchange
          BNEZ    R2,lockit   ;already locked?

          // critical section here

          MOV     R2, #0
          EXCH    R2, 0(R1)
          RET
```

Summary

- Memory hierarchy coherency
  - Bus-based snooping protocols (for bus)





- Synchronization in multi-threaded processors