SNOOPING PROTOCOLS

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

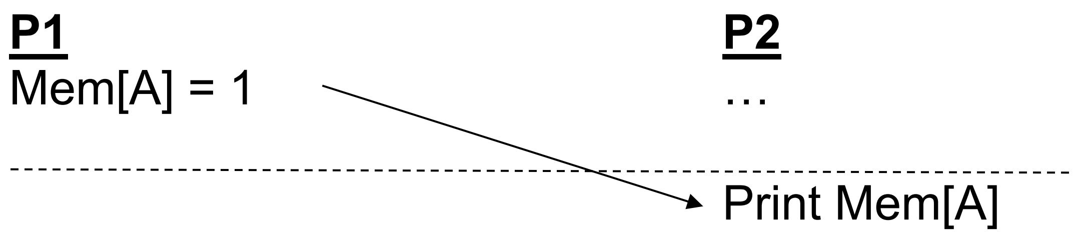
- Upcoming deadline
  - Mar. 8th: homework assignment release
  - Multiple questions from the list of papers suggested for reading until 11:59PM on Mar. 8th

Overview

- This lecture
  - Coherence basics
  - Update vs. Invalidate
  - A simple protocol
  - Illinois protocol
  - MESI protocol
  - MOESI optimization
  - Implementation issues

Recall: Shared Memory Model

- Goal: parallel programs communicate through shared memory system
- Example: a write from P1 is followed by a read from P2 to the same memory location (A)



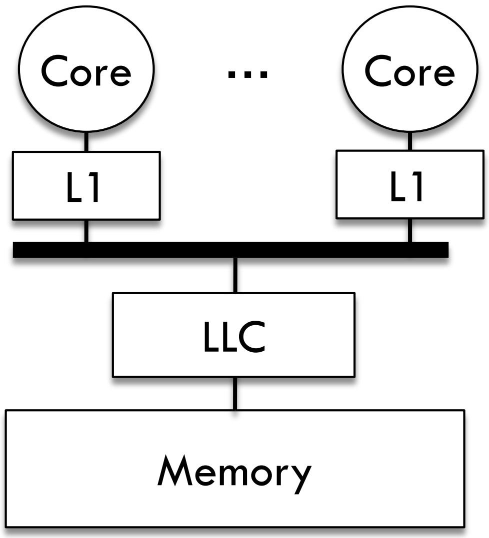
- Problem: what if Mem[A] was cached by P1 or P2?
  - Writable vs. read-only data

Cache Coherence Protocol

- Guarantee that all processors see a consistent value for the same memory location
- Provide the followings
  - Write propagation that sends updates to other caches
  - Write serialization that provide a consistent global order seen by all processors
- A global point of serialization is needed for ordering store instructions

Bus Snooping

- Relies on a broadcast infrastructure among caches
- Every cache monitors (snoops) the traffic to keep the states of the cache block up to date
  - All communication can be seen by all
- More scalable solution: ‘directory based’ schemes

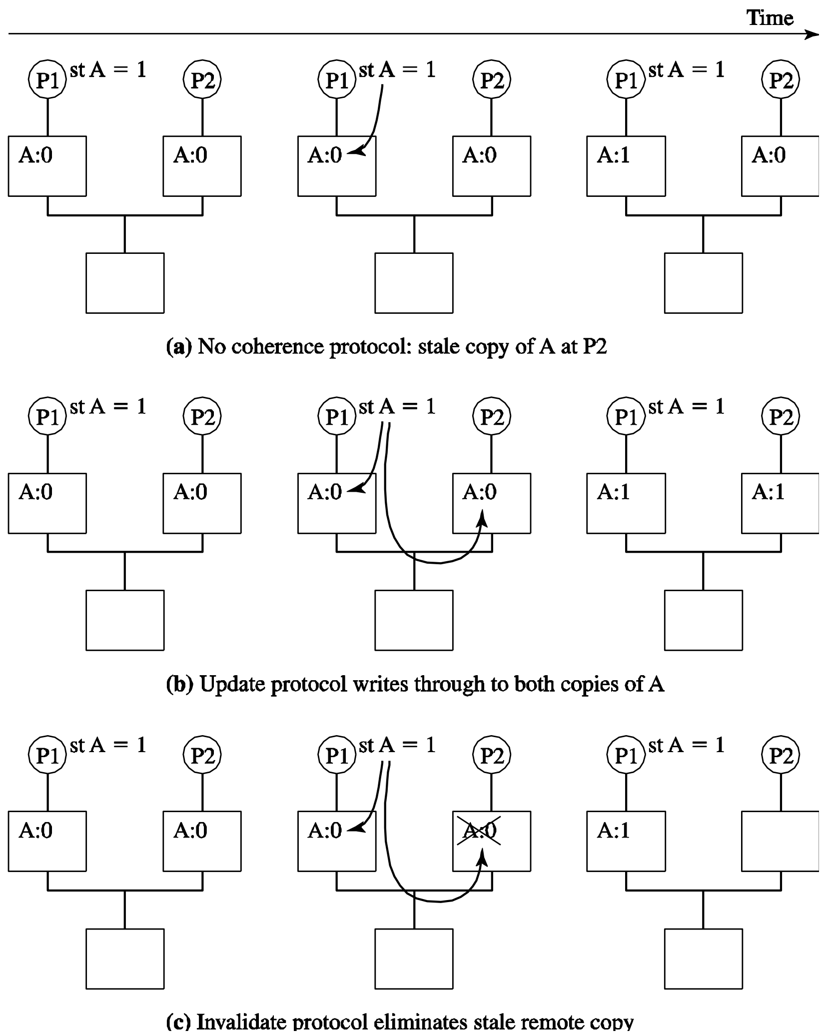


[Goodman’83]

Write Propagation

- Invalidate signal
  - Keep a single copy of the data after a write
- Update message
  - Update all of the replicas

Which one is better?



[slide ref.: Lipasti]

Invalidate vs. Update

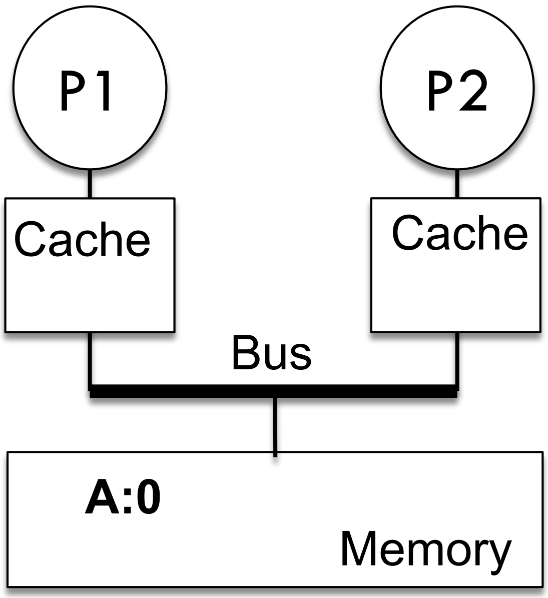
- Invalidate signal
  - Exclusive access rights for a single copy after every invalidation
  - May lead to rapid invalidation and reacquire of cache blocks (ping-ponging)
- Update message
  - Can alleviate the cost of ping-ponging; useful for infrequent updates
  - Unnecessary cost paid for updating blocks that will not be read
  - Consumes significant bus bandwidth and energy
- In general, invalidate based protocols are better

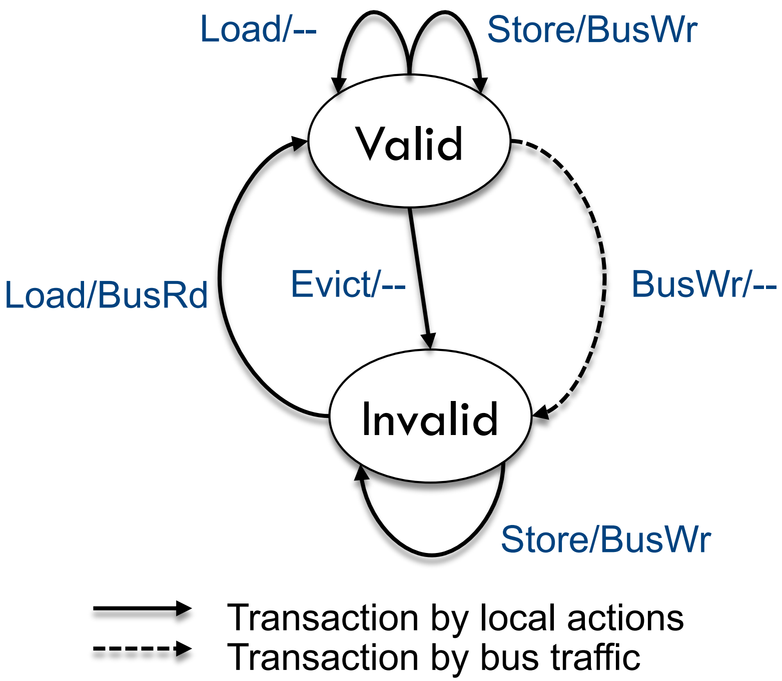
Implementation Tips

- Avoid sending any messages if no other copies of the cache block is used by other processors
- Depending on the cache write policy, the memory copy may be not up to date
  - Write through vs. write back
  - Write allocate vs. write no-allocate
- We need a protocol to handle all this

Simple Snooping Protocol

- Relies on write-through, write no-allocate cache
- Multiple readers are allowed
  - Writes invalidate replicas
- Employs a simple state machine for each cache unit



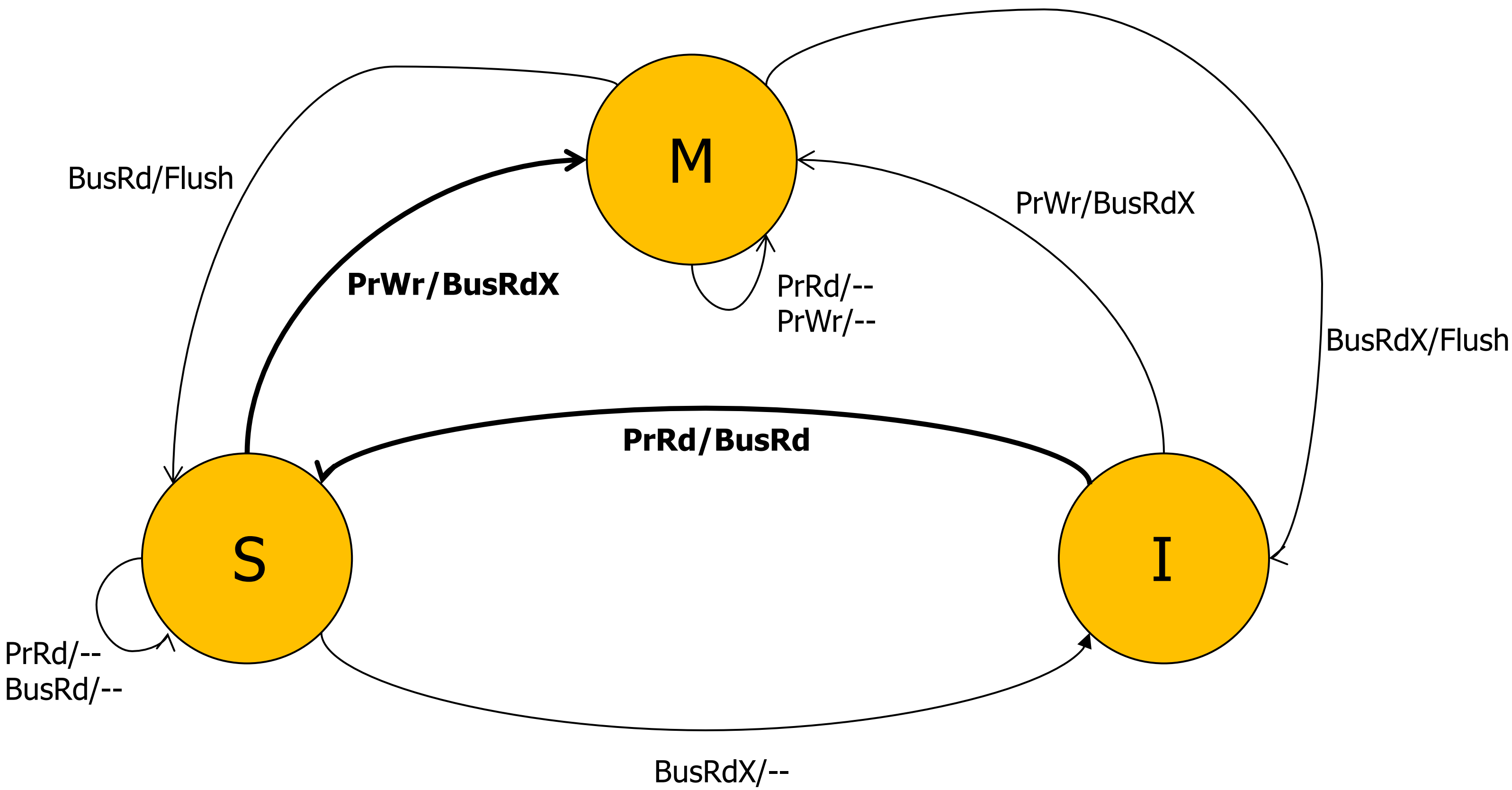


MSI: A Three State Protocol

- Instead of a single valid bit, more bits to represent
  - Modified (M): cache line is the only copy and is dirty
  - Shared (S): cache line is one of possibly many copies
  - Invalid (I): cache line is missing
- Read miss makes a *Read* request on bus, transitions to S
- Write miss makes a *ReadEx* request, transitions to M state
- When a processor snoops *ReadEx* from another writer, it must invalidate its own copy (if any)
- Upgrading S to M needs no reading data from memory

MSI: State Machine

ObservedEvent/Action



[Culler/Singh96]

MSI: Challenges

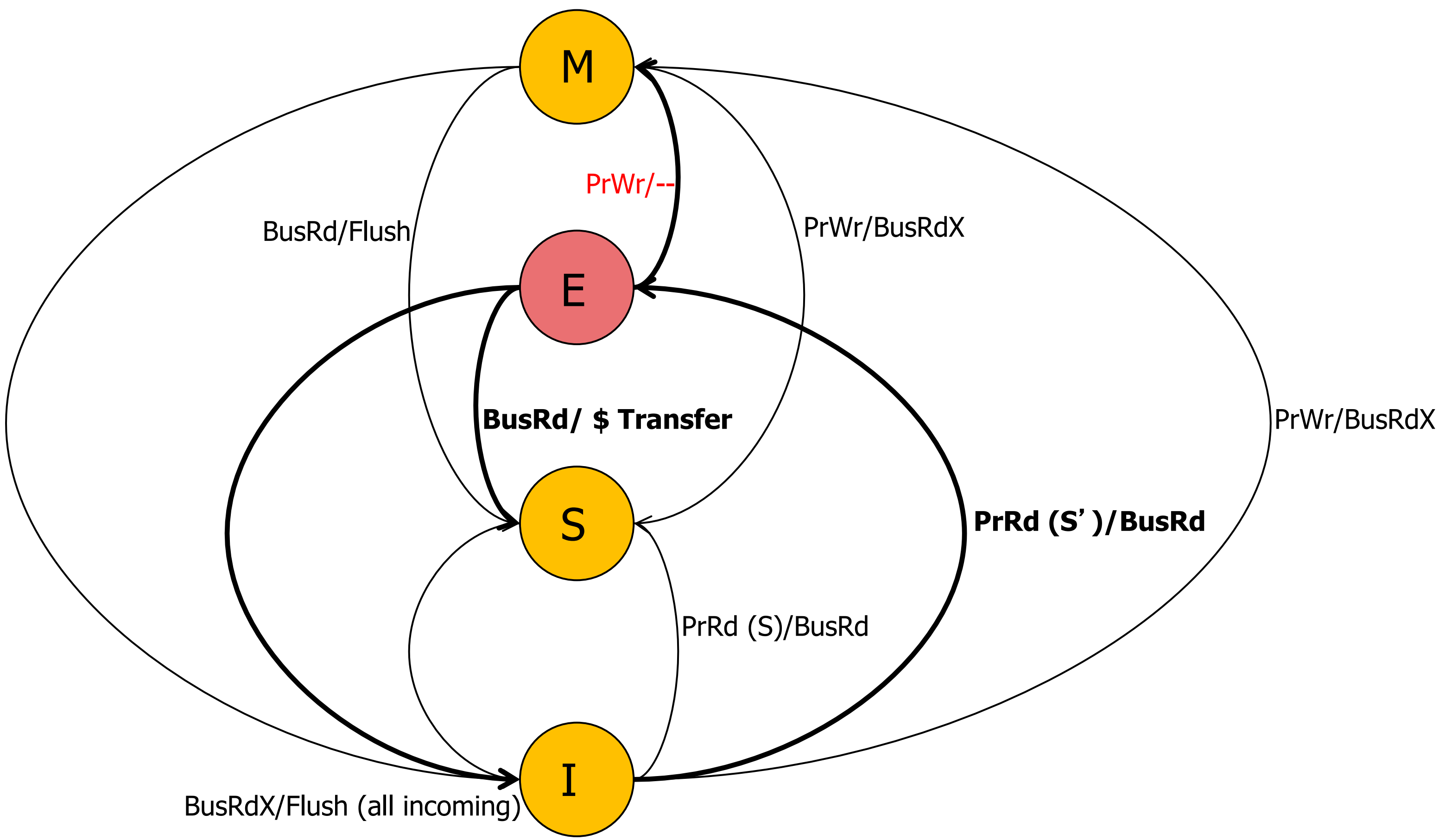
- Observation: on a read, the block immediately goes to "Shared" state although it may be the only copy to be cached and no other processor will cache it
  - A processor reads a block and wants to write to the same block
- Problem: we need to broadcast "invalidate" even for single copy cache blocks
- Solution: skip broadcasting "invalidate" signal
  - If the cache knew it had the only cached copy in the system, it could have written to the block without notifying any other cache
  - Save energy and time

MESI: A Four State Protocol

- Idea: Add another state indicating that this is the only cached copy and it is clean
  - *Exclusive* state
- How: block is placed into the *exclusive* state if, during *BusRd*, no other cache had it
  - Wired-OR "shared" signal on bus can determine this
    - snooping caches assert the signal if they also have a copy
- Result: silent transition E to M is possible on write

[Papamarcos'84]

MESI: State Machine



[Culler/Singh96]

MESI: Challenges

- Shared state requires the data to be clean
  - All caches that have the block have the up-to-date copy and so does the memory
- Observation: Need to write the block to memory when BusRd happens when the block is in Modified state
- Problem: Memory may be updated unnecessarily
  - Other processor may want to write to the block again while it is cached
  - Memory accesses consume significant time and energy

MESI: Challenges

- Solution 1: do not transition from M to S on a BusRd
  - Invalidate the copy and supply the modified block to the requesting processor directly without updating memory

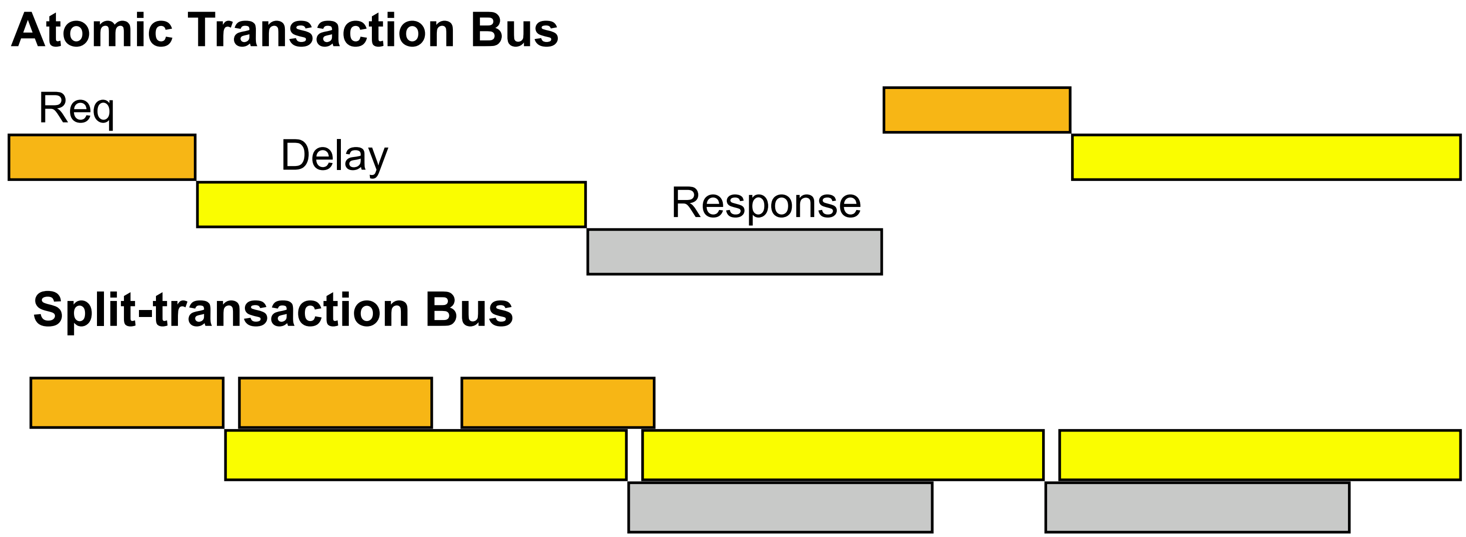
- Solution 2: transition from M to S, but designate one cache as the owner (O), who will write the block back when it is evicted
  - Now "Shared" means "Shared and potentially dirty"
  - This is a version of the MOESI protocol

Ownership Optimization

- Observation: shared ownership prevents cache-to-cache transfer, causes unnecessary memory read
  - Add O (owner) state to protocol: MOSI/MOESI
  - Last requestor becomes the owner
  - Avoid writeback (to memory) of dirty data
  - Also called shared-dirty state, since memory is stale
- Used in AMD Opteron

Implementation Challenges

- Multi-layer cache architecture
- Uncertain memory delay
- Non-atomic bus transactions

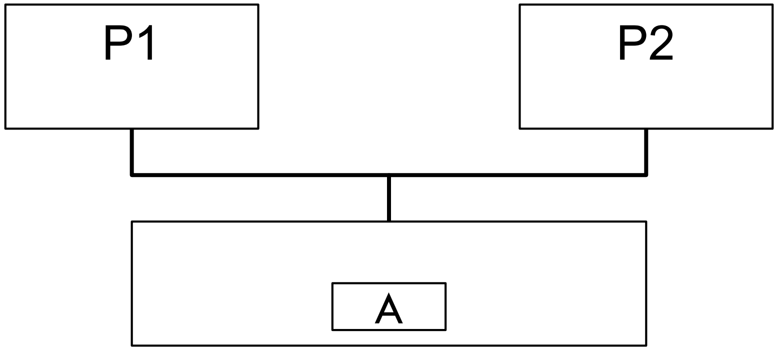


Implementation Challenges

- Deadlock
  - All system activity ceases
  - Cycle of resource dependences
- Livelock
  - No processor makes forward progress
  - Constant on-going transactions at hardware level
  - E.g. simultaneous writes in invalidation-based protocol
- Starvation
  - Some processors make no forward progress
  - E.g. interleaved memory system with NACK on bank busy

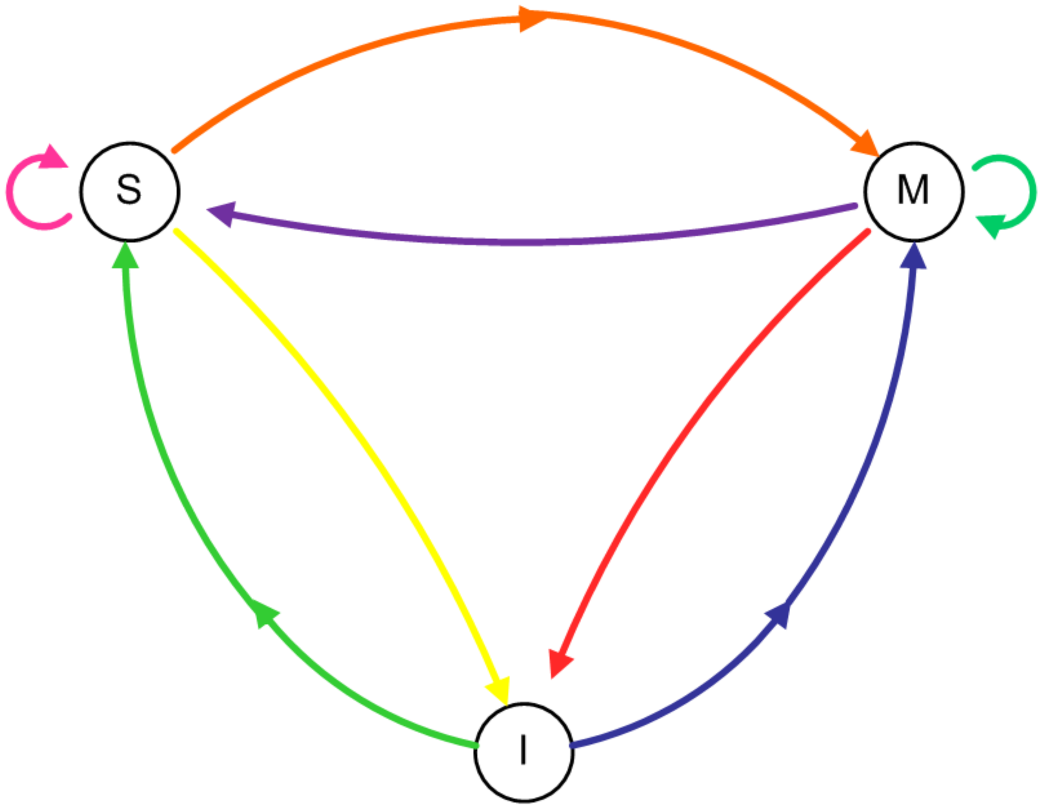
Recall: Cache Coherence

- Definition of coherence
  - Write propagation
    - Write ate visible to other processors
  - Write serialization
    - All write to the same location are seen in the same order by all processes



Implementation Challenges

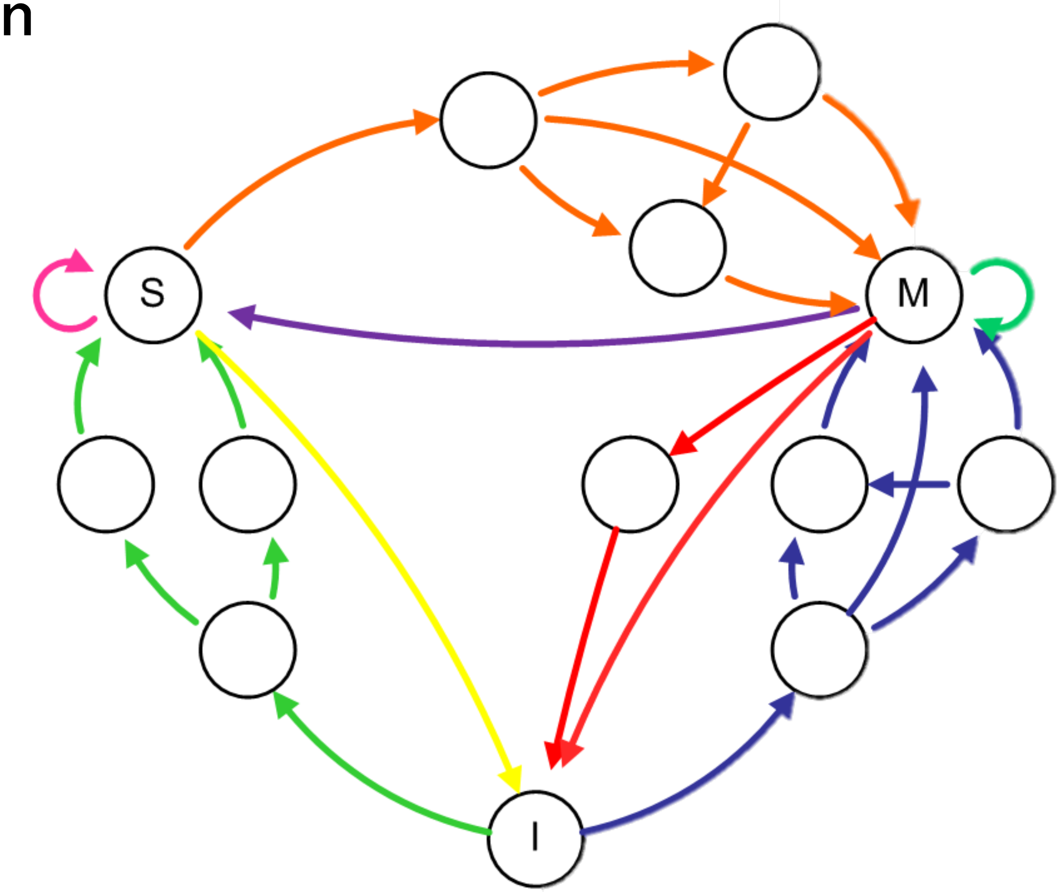
- MSI implementation
  - Stable States



[Vantrease'11]

Implementation Challenges

- MSI implementation
  - Stable States
  - Busy states

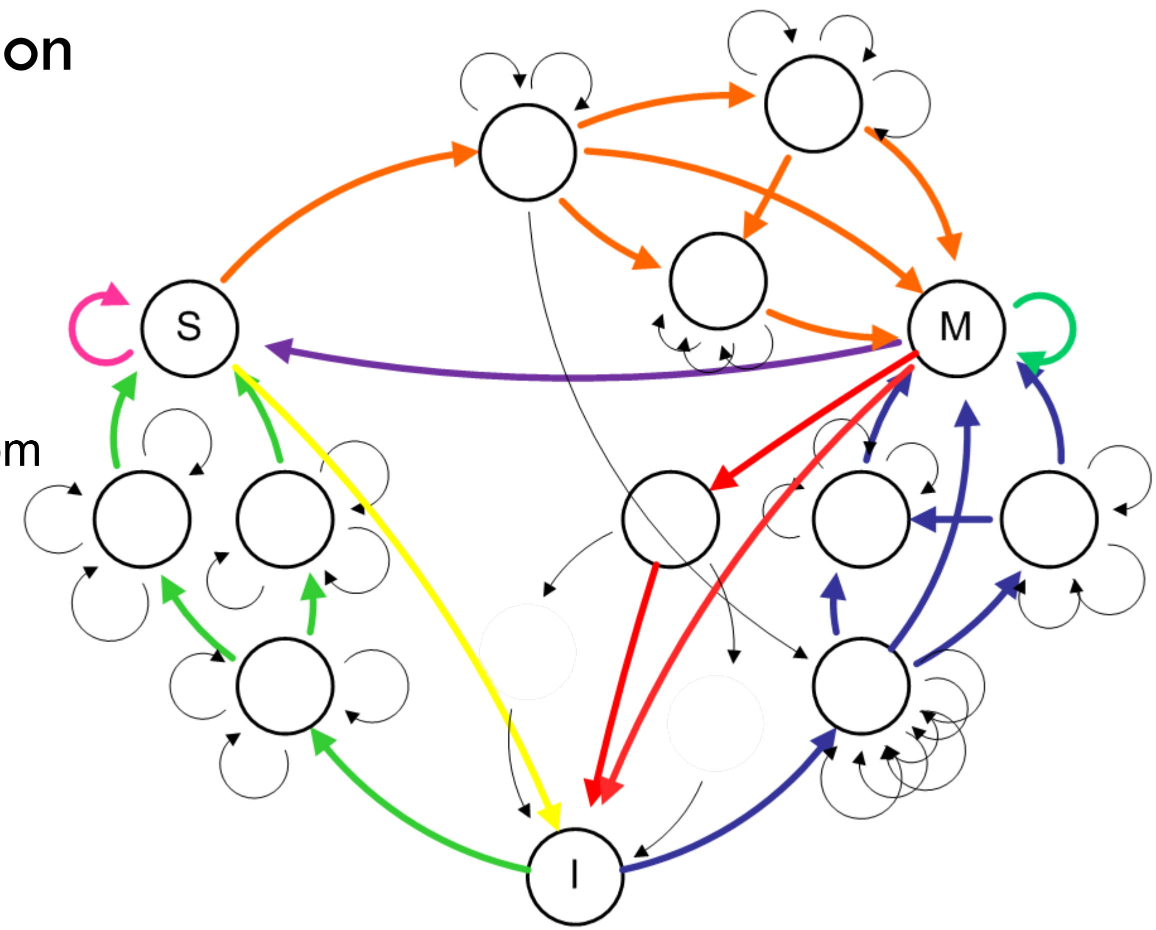


[Vantrease'11]

Implementation Challenges

- MSI implementation
  - Stable States
  - Busy states
  - Races

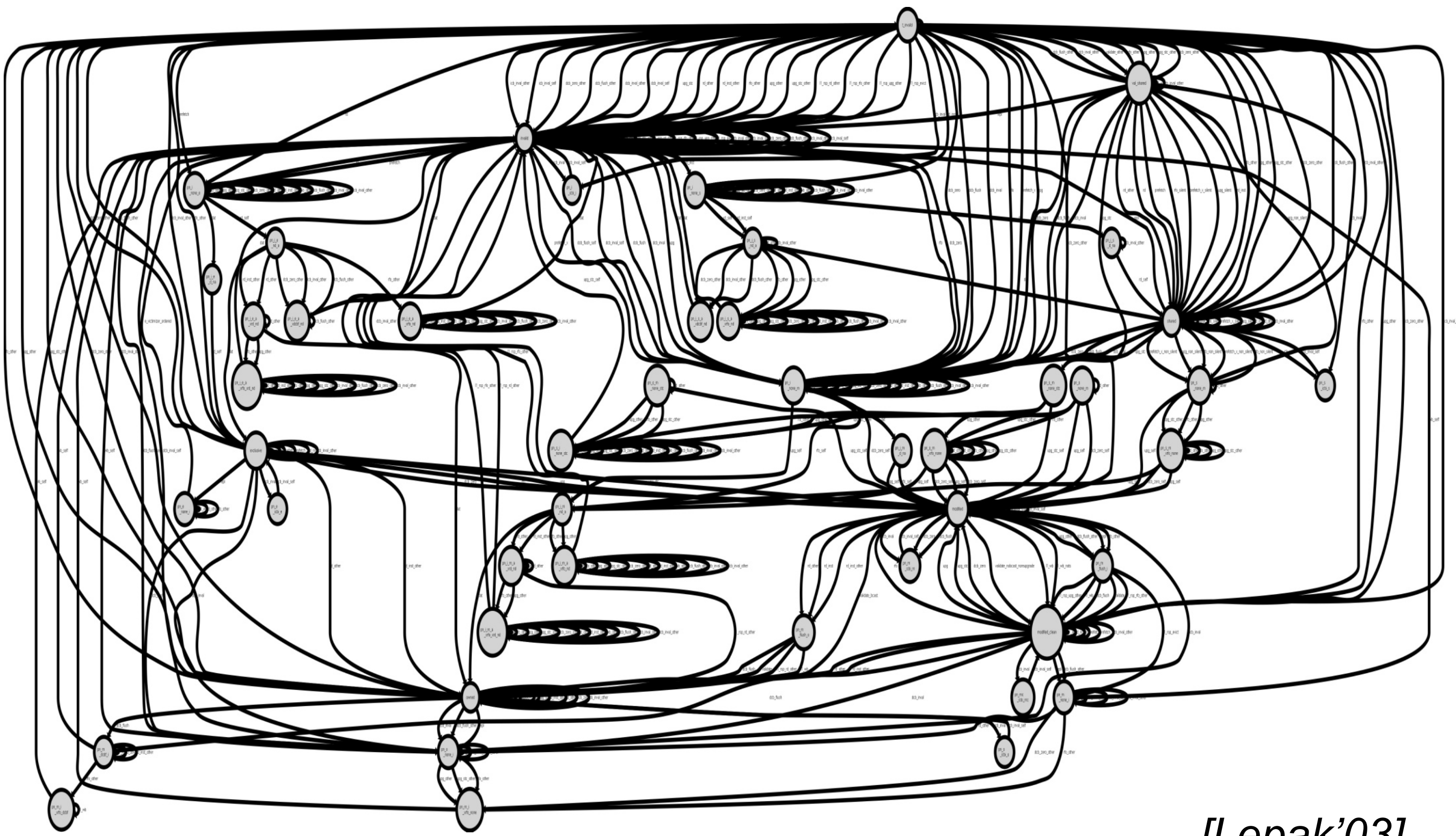
    Unexpected events from concurrent requests to same block



[Vantrease’11]

Cache Coherence Complexity

- A broadcast snooping bus (L2 MOETSI)



[Lepak'03]

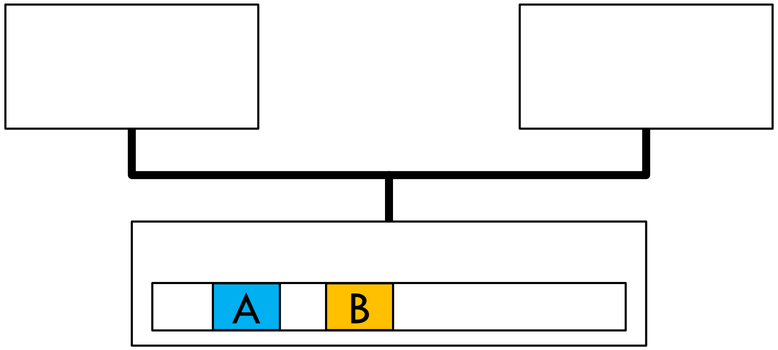
Implementation Tradeoffs

- Reduce unnecessary invalidates and transfers of blocks
  - Optimize the protocol with more states and prediction mechanisms

- Adding more states and optimizations
  - Difficult to design and verify
    - lead to more cases to take care of
    - race conditions
  - Gained benefit may be less than costs (diminishing returns)

Coherence Cache Miss

- Recall: cache miss classification
  - Cold (compulsory): first access to block
  - Capacity: due to limited capacity
  - Conflict: many blocks are mapped to the same set
- New class: misses due to sharing
  - True vs. false sharing



Summary of Snooping Protocols

- Advantages
  - Short miss latency
  - Shared bus provides global point of serialization
  - Simple implementation based on buses in uniprocessors

- Disadvantages
  - Must broadcast messages to preserve the order
  - The global point of serialization is not scalable
    - It needs a virtual bus (or a totally-ordered interconnect)

Scalable Coherence Protocols

- Problem: shared interconnect is not scalable
- Solution: make explicit requests for blocks
- Directory-based coherence: every cache block has additional information
  - To track of copies of cached blocks and their states
  - To track ownership for each block
  - To coordinate invalidation appropriately