Lecture 7
Cache Coherence

Daniel Sanchez and Joel Emer

6.888 Parallel and Heterogeneous Computer Architecture
Spring 2013

Massachusetts Institute of Technology

Coherence & Consistency

- Shared memory systems:
  - Have multiple private caches for performance reasons
  - Need to provide the illusion of a single shared memory

- Intuition: A read should return the most recently written value
  - What is most recent?

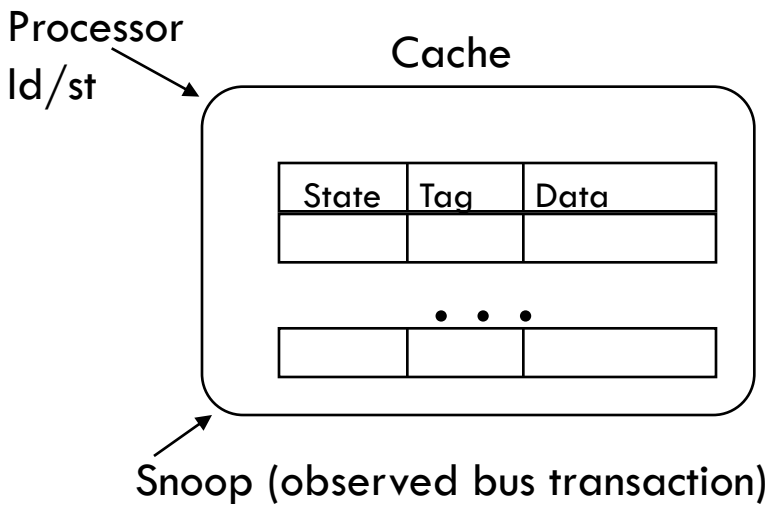
- Formally:
  - Coherence: What values can a read return?
    - Concerns reads/writes to a single memory location
  - Consistency: When do writes become visible to reads?
    - Concerns reads/writes to multiple memory locations

Coherence Rules

- Writes eventually become visible to all processors
- Writes to the same location are serialized

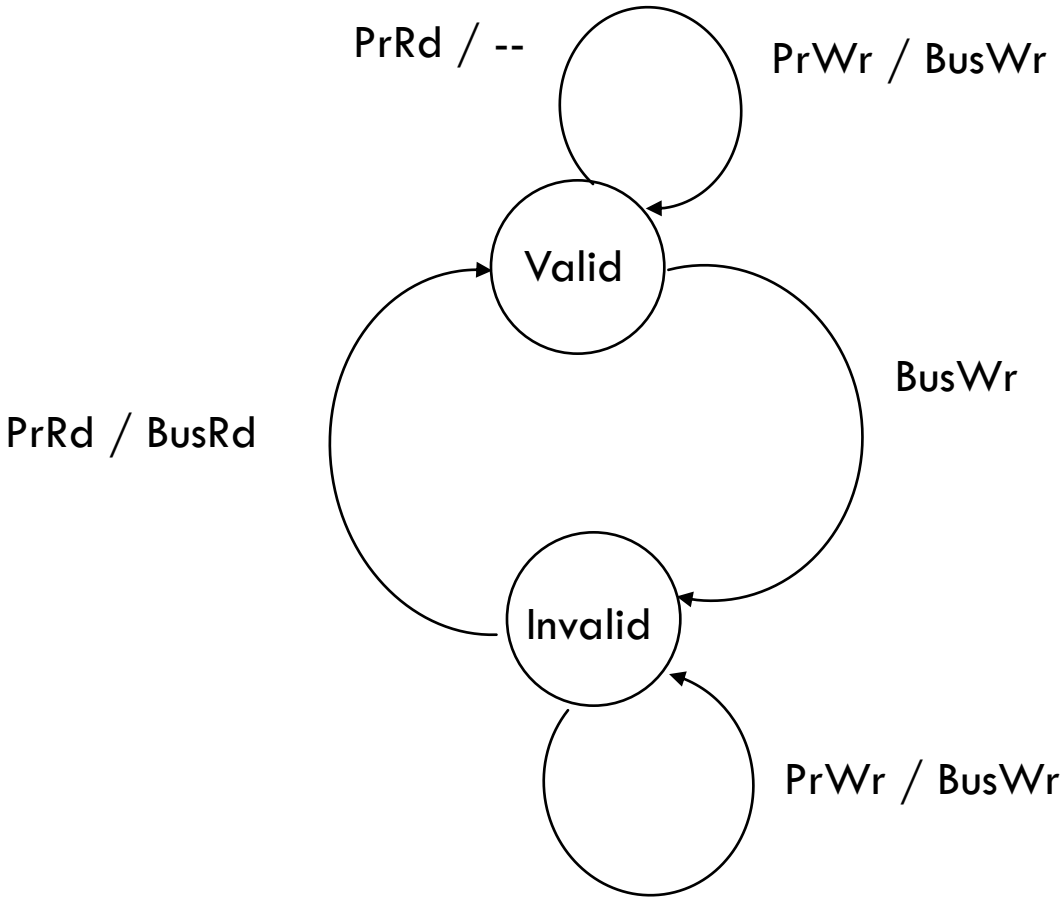
Snoopy Coherence Protocols

- Bus provides serialization point
  - Broadcast, totally ordered
  - Each cache controller "snoops" all bus transactions
  - Controller updates state of cache in response to processor and snoop events and generates bus transactions
- Snoopy protocol (FSM)
  - State-transition diagram
  - Actions
- Handling writes:
  - Write-invalidate
  - Write-update



[adapted from Olukotun & Kozyrakis, CS316 lecture notes, 2012]

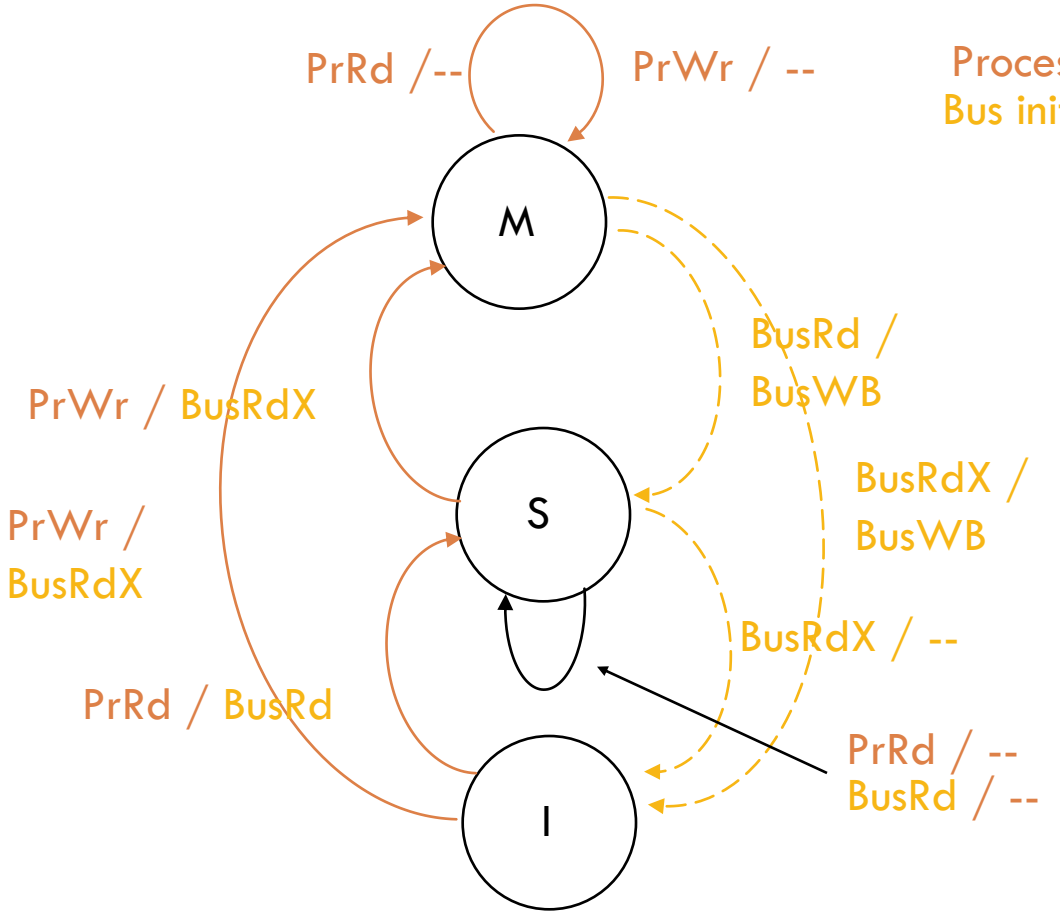
Valid/Invalid (VI) Protocol



- Write-through, no-write-allocate cache

Action	Abbreviation
Processor Read	PrRd
Processor Write	PrWr
Bus Read	BusRd
Bus Write	BusWr

MSI State Diagram



Abbreviation	Action
PrRd	Processor Read
PrWr	Processor Write
BusRd	Bus Read
BusRdX	Bus Read Exclusive
BusWB	Bus Writeback

Exclusive State

- Observation: Doing read-modify-write sequences on private data is common
  - What's the problem with MSI?

- Solution: E state (exclusive, clean)
  - If no other sharers, a read acquires line in E
  - Writes silently cause E→M (exclusive, dirty)
  - Does everything get faster?

Owner State

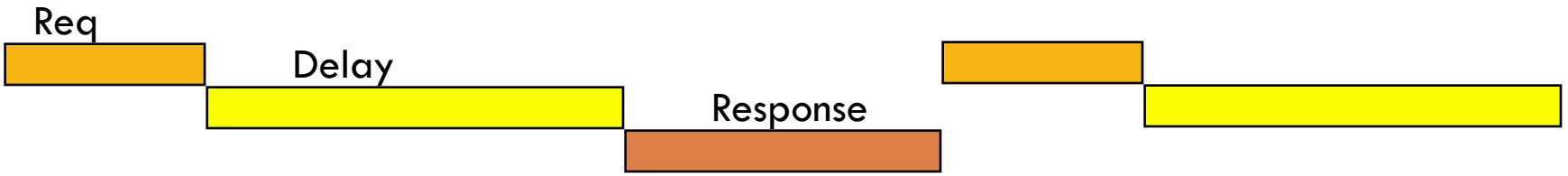
- Observation: On M→S transitions, must write back line!
  - What happens with frequent read-write sharing?
  - Can we defer the write after S?

- Solution: O state (Owner)
  - O = S + responsibility to write back
  - On M→S transition, one sharer (typically the one who had the line in M) retains the line in O instead of S
  - On eviction, O writes back line (or other sharer does S→O)

- MSI, MESI, MOSI, MOESI…
  - Typically E if private read-write >> read-shared (common)
  - Typically O only if writebacks are expensive (main mem vs L3)

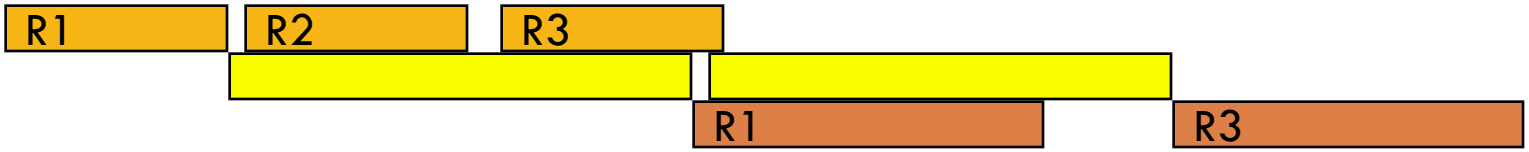
Split-Transaction and Pipelined Buses

Atomic Transaction Bus



- Simple, but low throughput!

Split-Transaction Bus

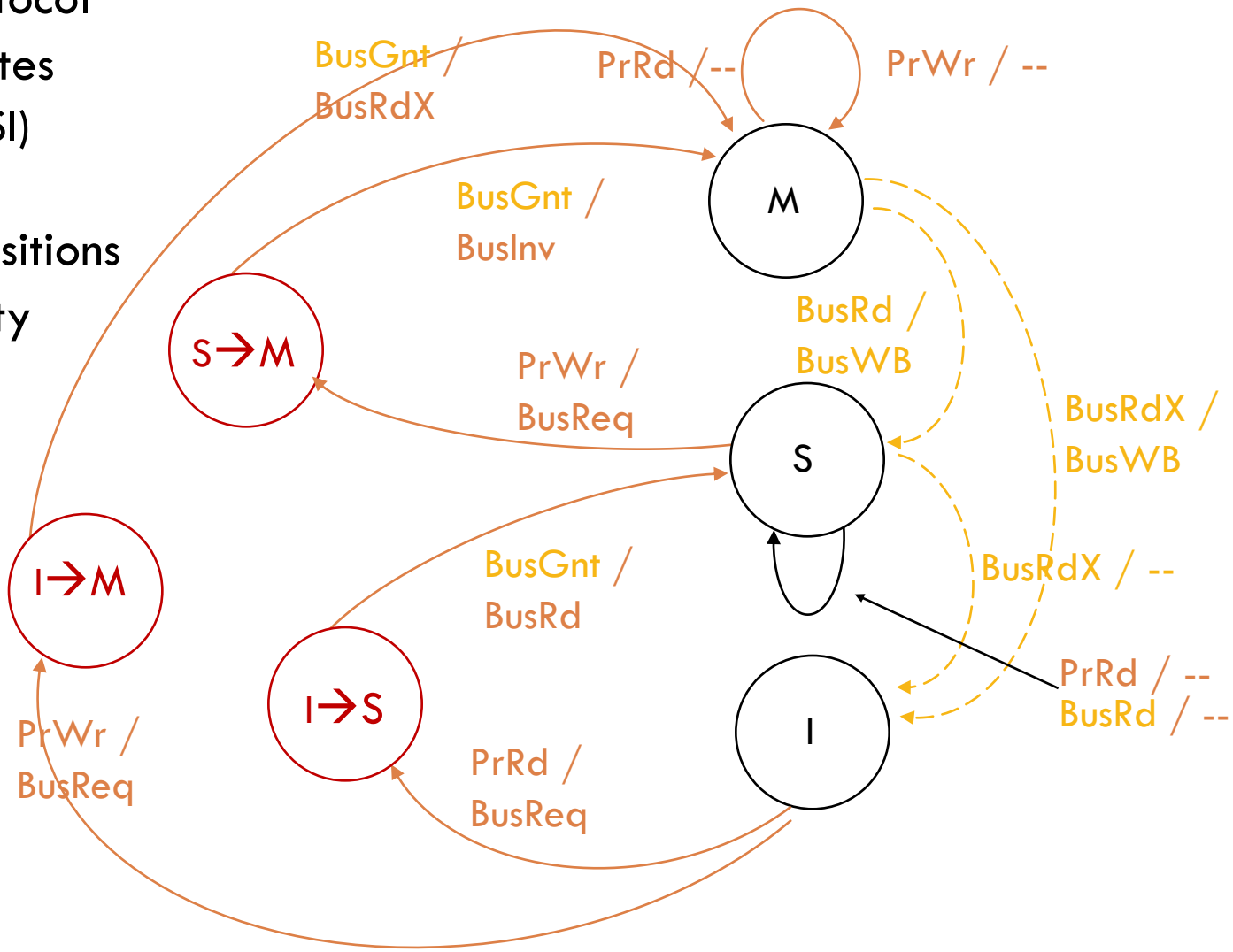


- Supports multiple simultaneous transactions
  - Higher throughput
  - Responses may be OOO
- Often implemented as multiple buses (req+resp)
- What happens to coherence?

Non-Atomicity → Transient States

- Must extend protocol
- Two types of states
  - Stable (e.g. MSI)
  - Transient
- Split + race transitions
- Higher complexity

Action	Abbr.
Bus Request	BusReq
Bus Grant	BusGnt



Complex Protocols → More Races

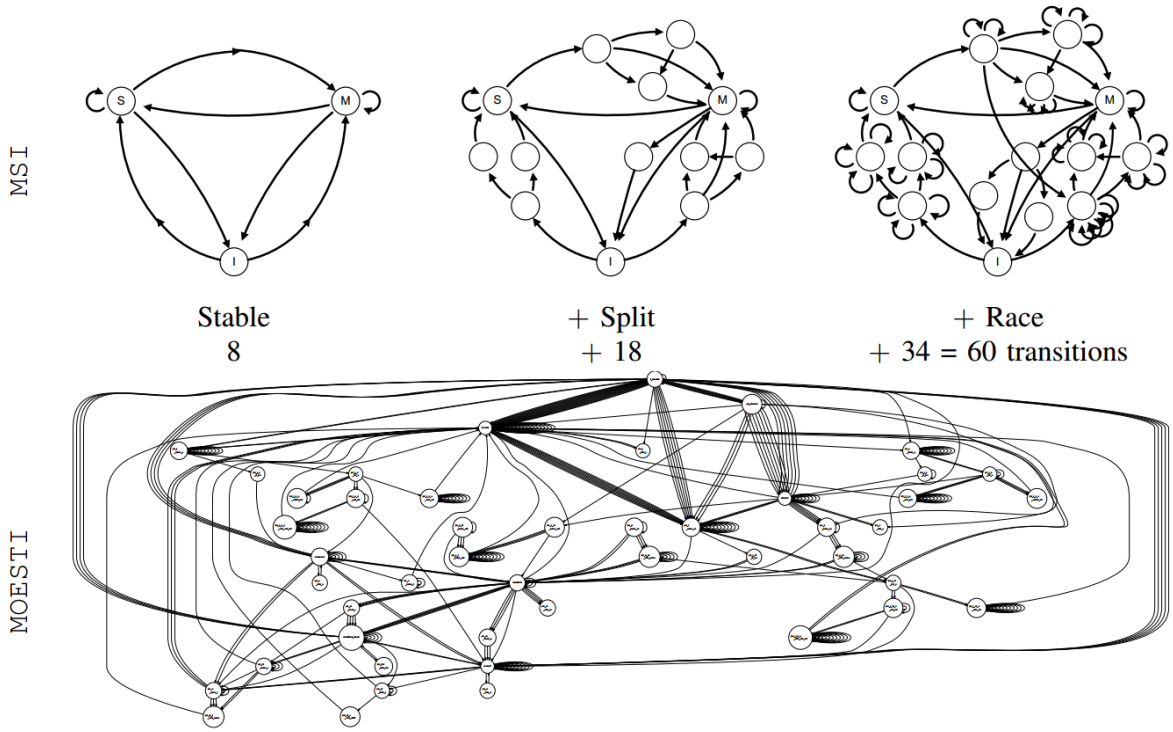


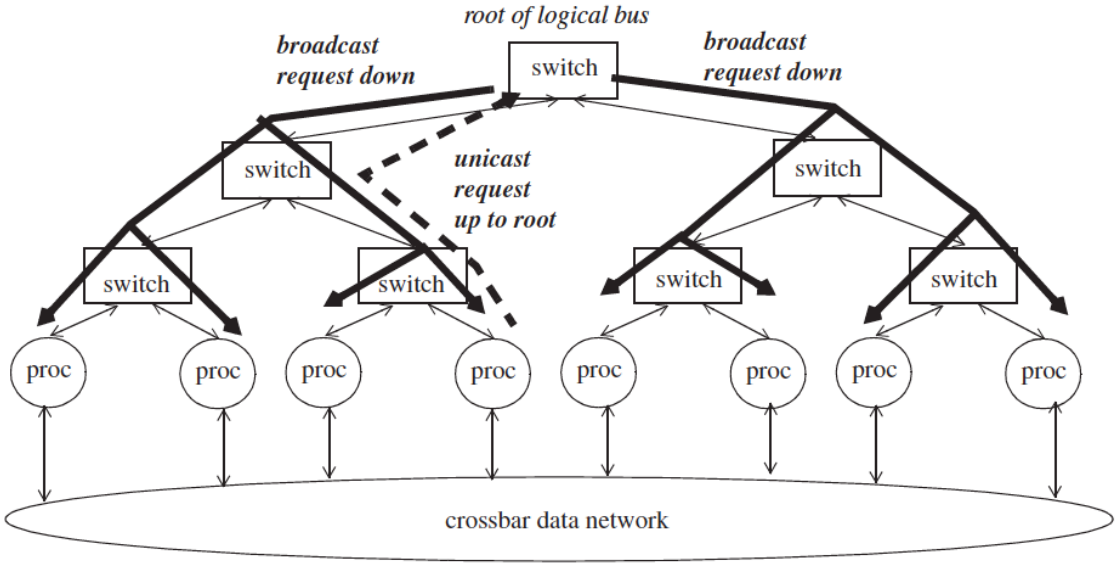
Fig. 1: L2 Transitions for Two Protocols.

[Vantrease et al., "Atomic Coherence", HPCA 2011]

- How to ensure the protocol works?
  - Preserve coherence invariants
  - Deadlock, livelock, starvation-free

Scaling Cache Coherence

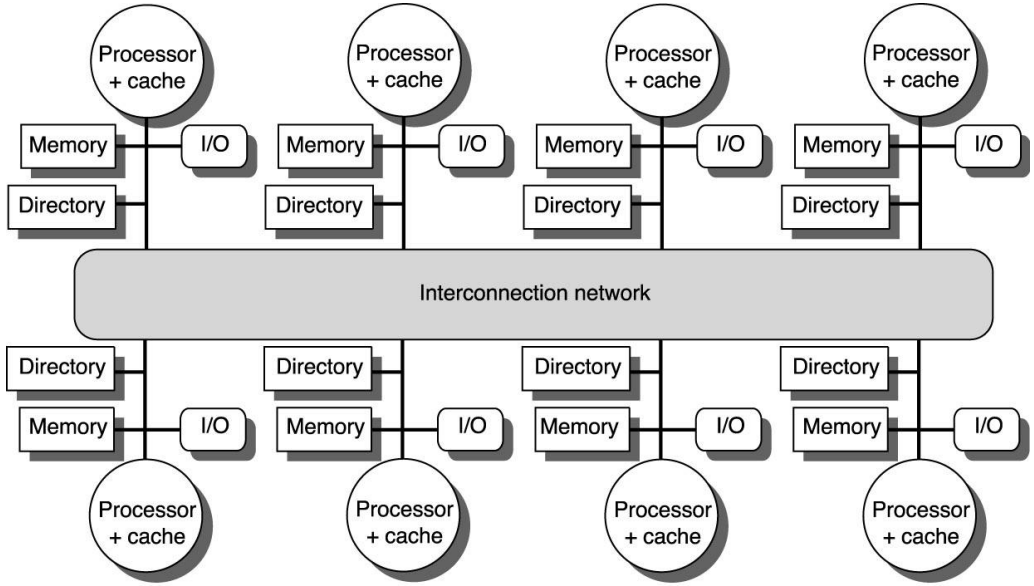
- Can implement more scalable ordered interconnects…



Starfire E10000 (drawn with only eight processors for clarity). A coherence request is unicast up to the root, where it is serialized, before being broadcast down to all processors

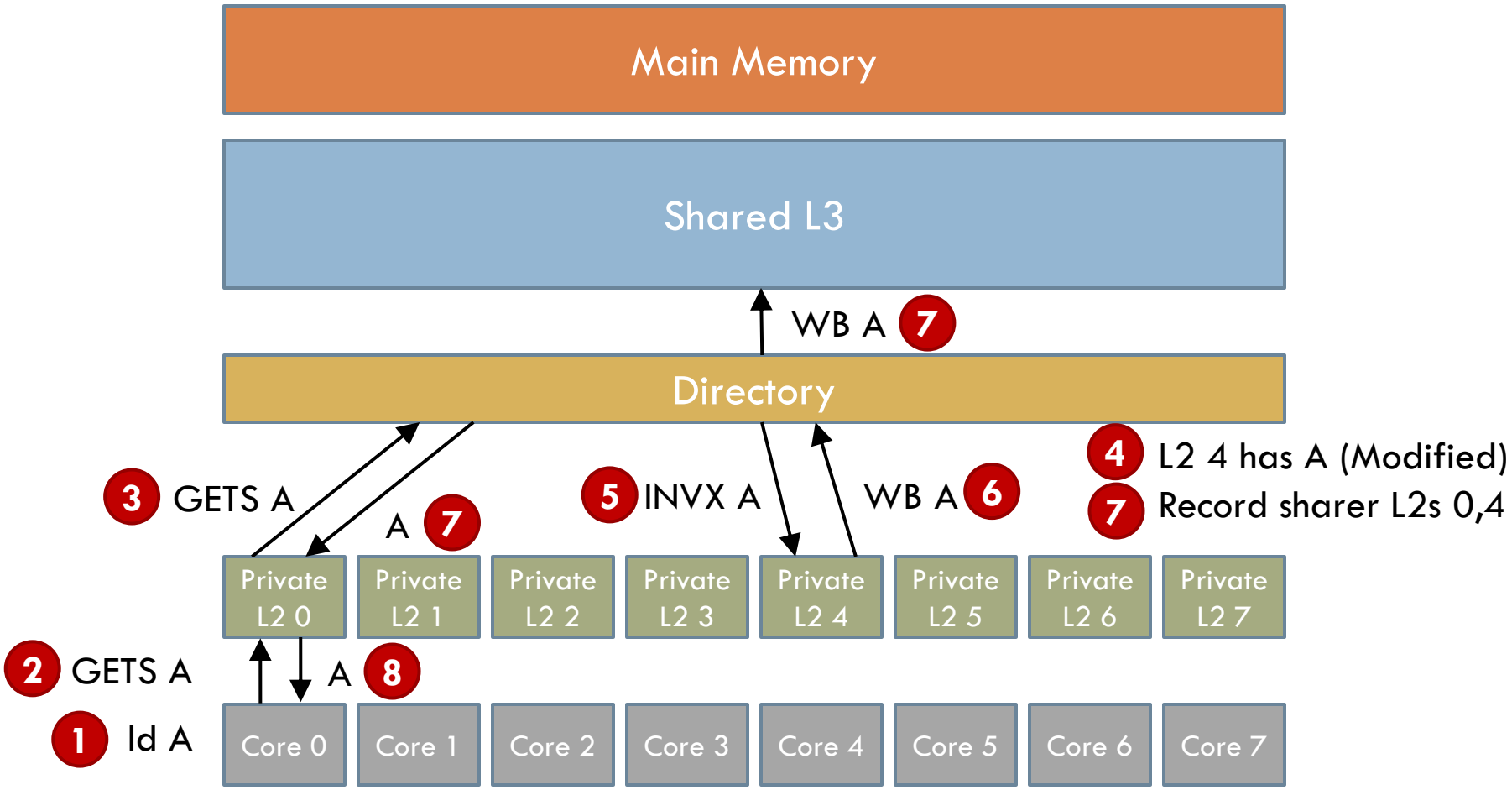
- … but broadcast is fundamentally unscalable
  - Bandwidth, energy of transactions with 1K cache snoops?

Directory-Based Coherence



- Route all coherence transactions through a directory
  - Tracks contents of private caches → No broadcasts
  - Serves as ordering point for conflicting requests → Unordered networks

Example: Shared Cache Line Read



Directory Taxonomy & Scalability

- Duplicate tags
- Full-map
- Sparse
  - Full bit-vectors
  - Coarse-grain bit-vectors
  - Limited-pointers
- In-cache
- Hierarchical sparse

Readings for Monday

- Read BulkSC
- Skim Consistency Tutorial