Computer Architecture

Spring 2016

Lecture 20: Shared-Memory Multiprocessors & Cache Coherence

Shuai Wang

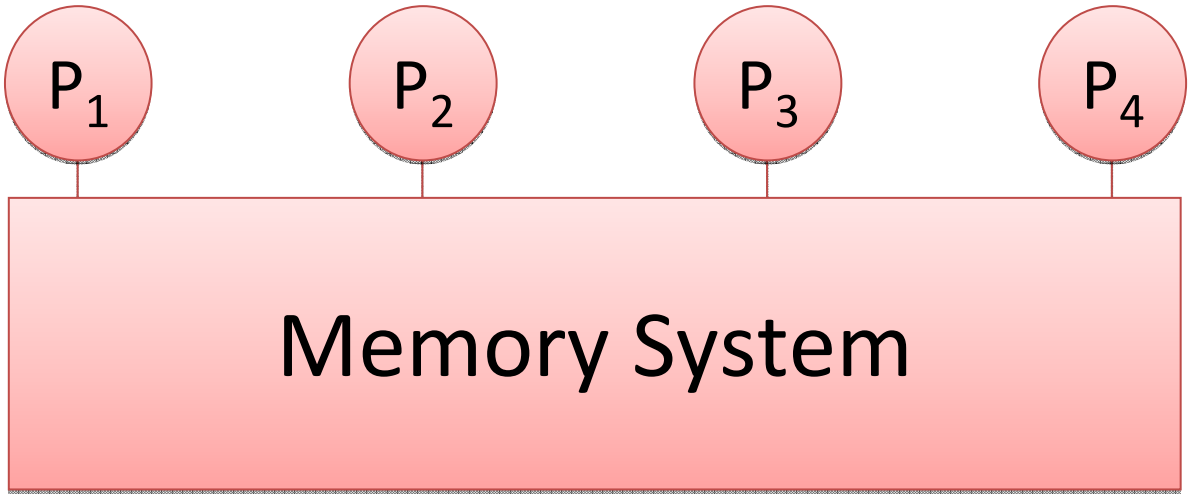
Department of Computer Science and Technology

Nanjing University

[Slides adapted from CSE 502 Stony Brook University]

Shared-Memory Multiprocessors

- Multiple threads use *shared memory* (address space)
  - "SysV Shared Memory" or "Threads" in software
- Communication implicit via loads and stores
  - Opposite of explicit *message-passing multiprocessors*
- Theoretical foundation: *PRAM model*

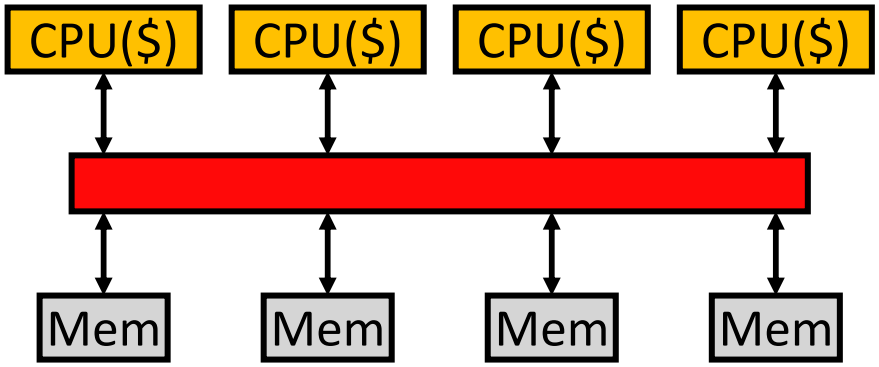


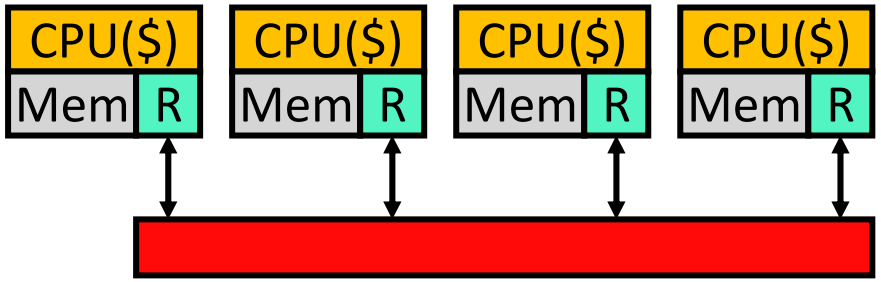
Why Shared Memory?

- Pluses
  - App sees multitasking uniprocessor
  - OS needs only evolutionary extensions
  - Communication happens without OS
- Minuses
  - Synchronization is complex
  - Communication is implicit (hard to optimize)
  - Hard to implement (in hardware)
- Result
  - SMPs and CMPs are most successful machines to date
  - First with multi-billion-dollar markets

Paired vs. Separate Processor/Memory?

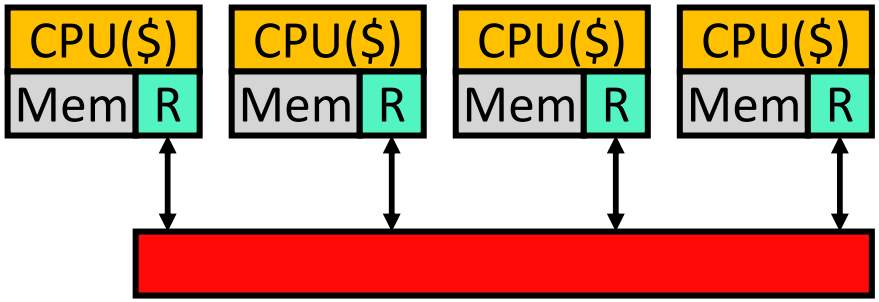
- Separate CPU/memory
  - *Uniform memory access* (*UMA*)
    - Equal latency to memory
  - Low peak performance
- Paired CPU/memory
  - *Non-uniform memory access* (*NUMA*)
    - Faster local memory
    - Data placement matters
  - High peak performance

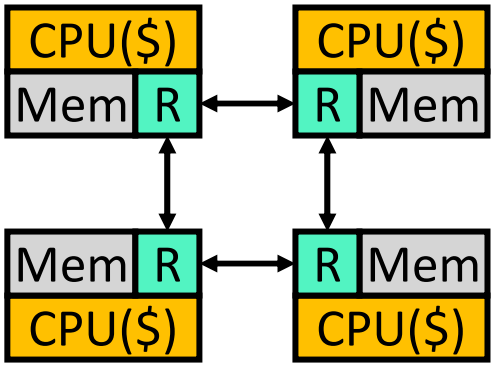




Shared vs. Point-to-Point Networks

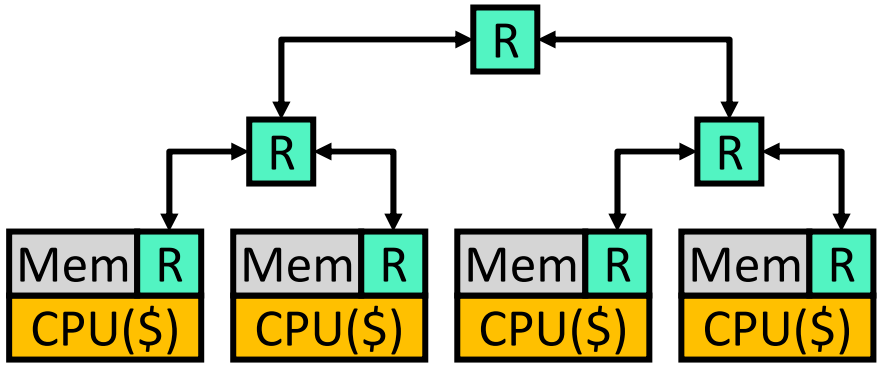
- Shared network
  - Example: bus
  - Low latency
  - Low bandwidth
    - Doesn't scale >~16 cores
  - Simple cache coherence
- Point-to-point network:
  - Example: mesh
  - High latency (many "*hops*")
  - Higher bandwidth
    - Scales to 1000s of cores
  - Complex cache coherence

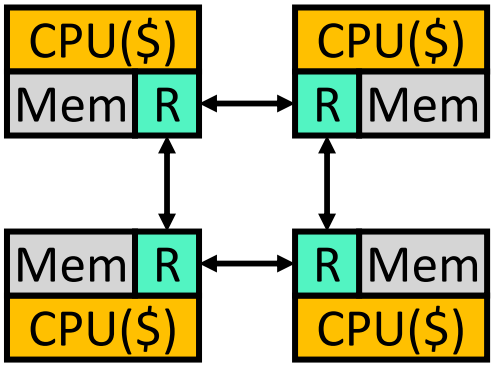




Organizing Point-To-Point Networks

- *Network topology*: organization of network
  - Trade off perf. (connectivity, latency, bandwidth) ↔ cost
- Router chips
  - Networks w/separate router chips are *indirect*
  - Networks w/ processor/memory/router in chip are *direct*
    - Fewer components, "*Glueless MP*"



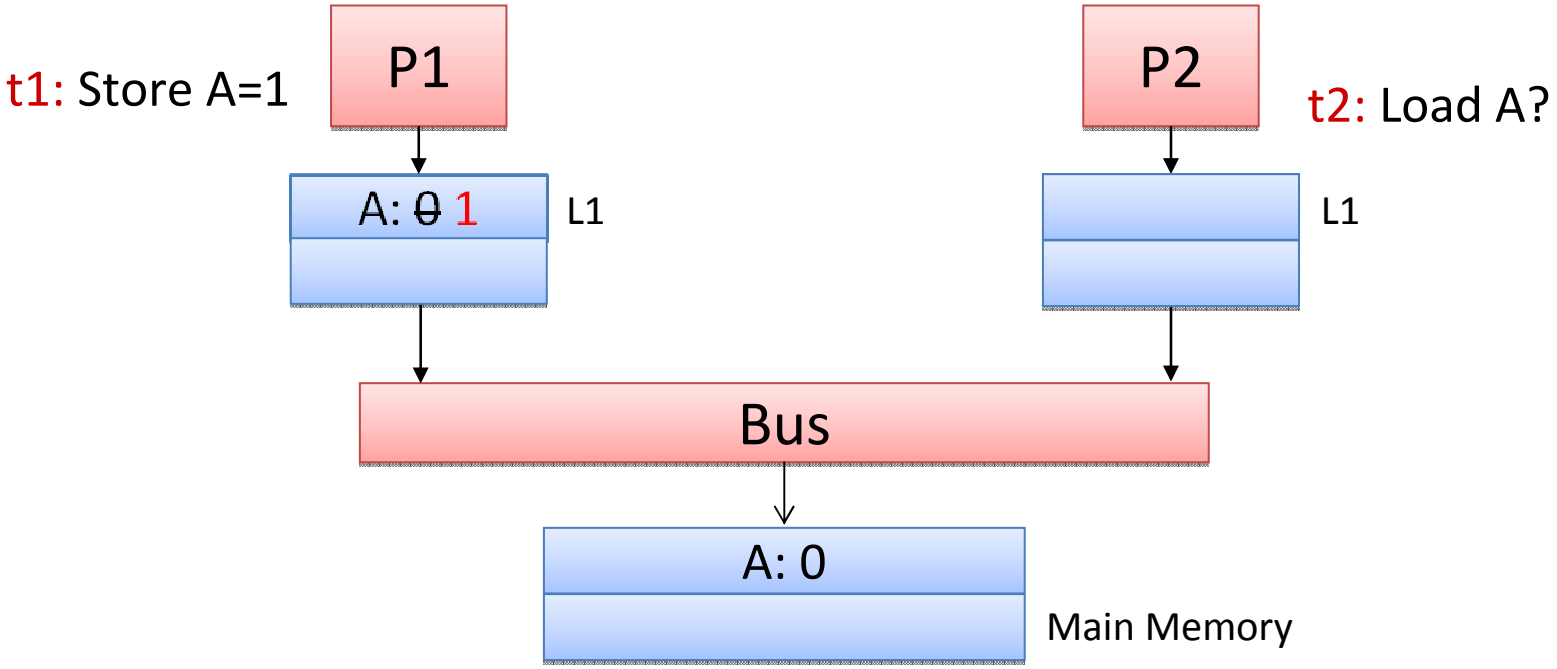


Issues for Shared Memory Systems

- Two big ones
  - Cache coherence
  - Memory consistency model
- Closely related
- *Often confused*

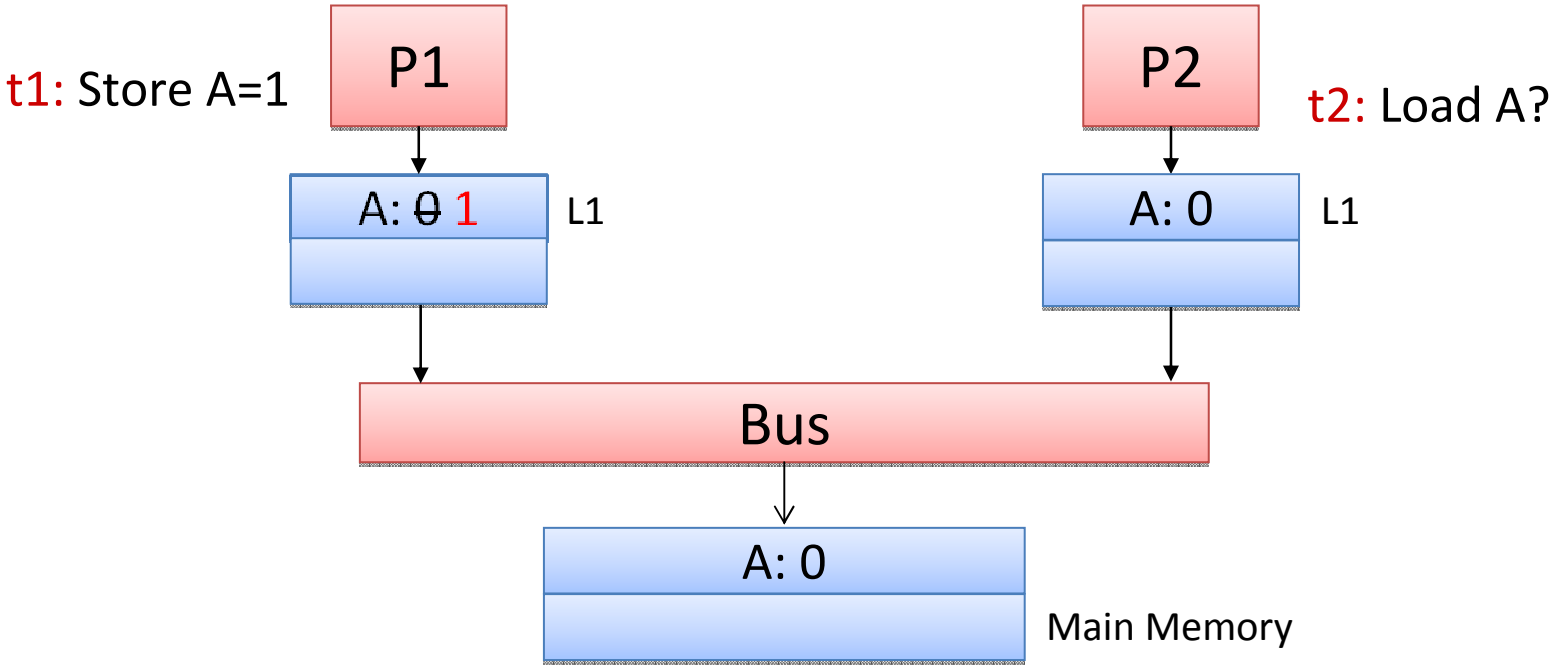
Cache Coherence: The Problem (1/2)

- Variable A initially has value 0
- P1 stores value 1 into A
- P2 loads A from memory and sees old value 0



Cache Coherence: The Problem (2/2)

- P1 and P2 have variable A (value 0) in their caches
- P1 stores value 1 into A
- P2 loads A from its cache and sees old value 0



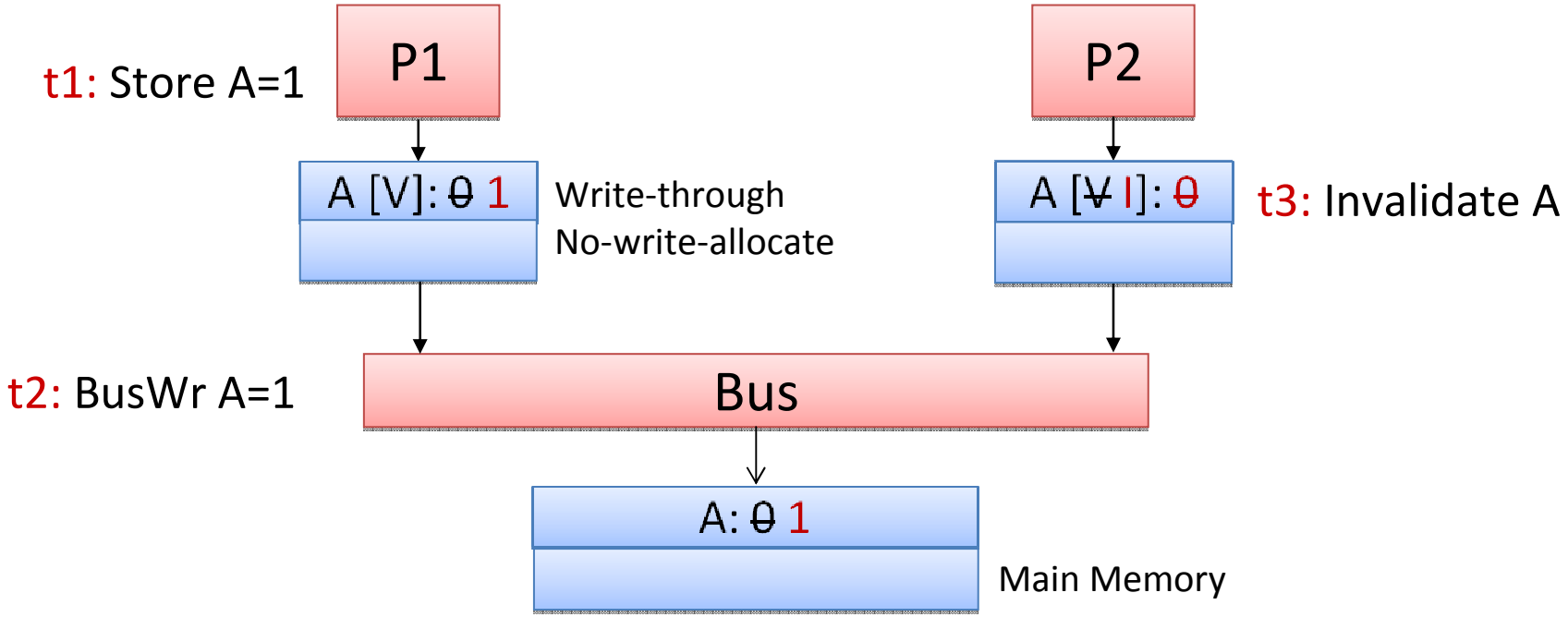
Approaches to Cache Coherence

- Software-based solutions
  - Mechanisms:
    - Mark cache blocks/memory pages as cacheable/non-cacheable
    - Add “Flush” and “Invalidate” instructions
  - Could be done by compiler or run-time system
  - Difficult to get perfect (e.g., what about memory aliasing?)

- Hardware solutions are far more common
  - System ensures everyone always sees the latest value

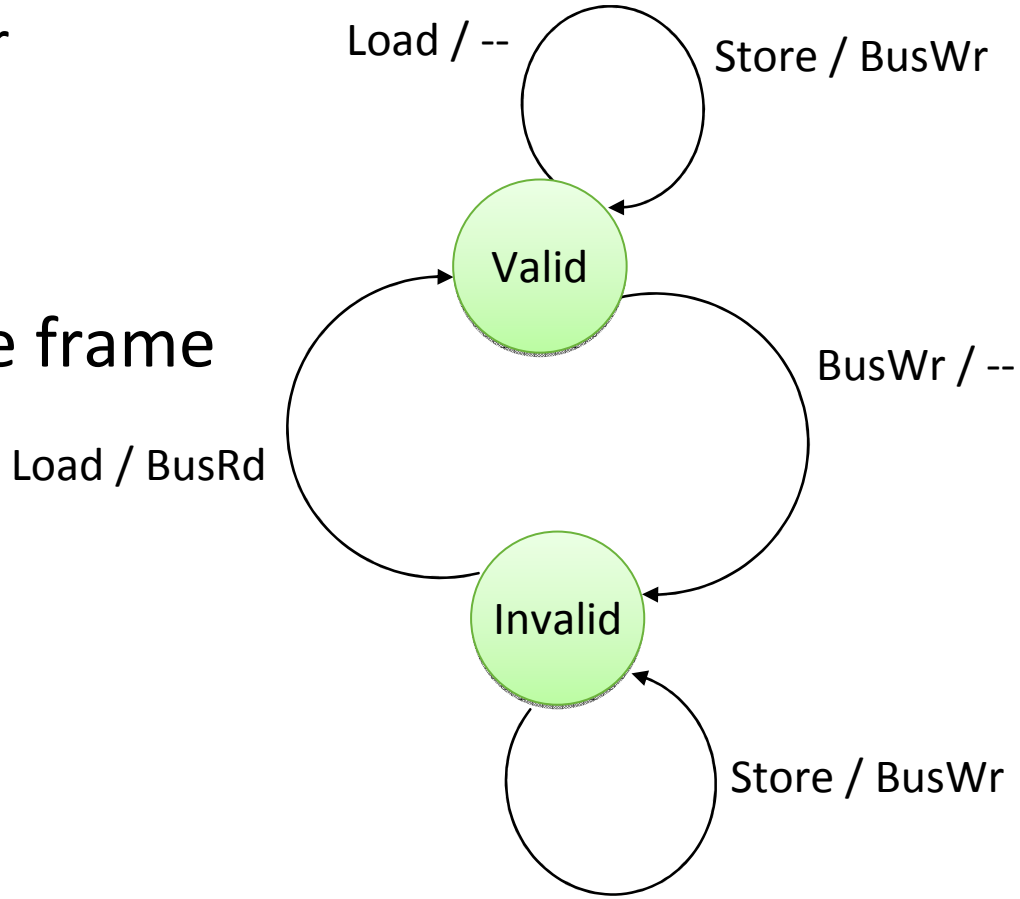
Coherence with Write-through Caches

- Allows multiple readers, but writes through to bus
  - Requires Write-through, no-write-allocate cache
- All caches must monitor (aka "*snoop*") all bus traffic
  - Simple state machine for each cache frame



Valid-Invalid Snooping Protocol

- Processor Actions
  - Ld, St, BusRd, BusWr
- Bus Messages
  - BusRd, BusWr
- Track 1 bit per cache frame
  - Valid/Invalid



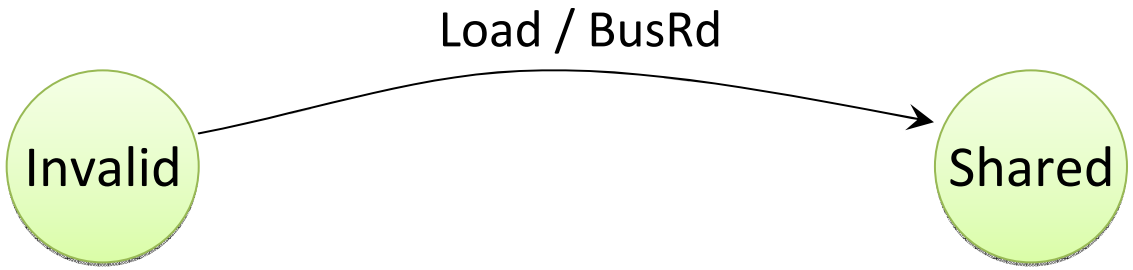
Supporting Write-Back Caches

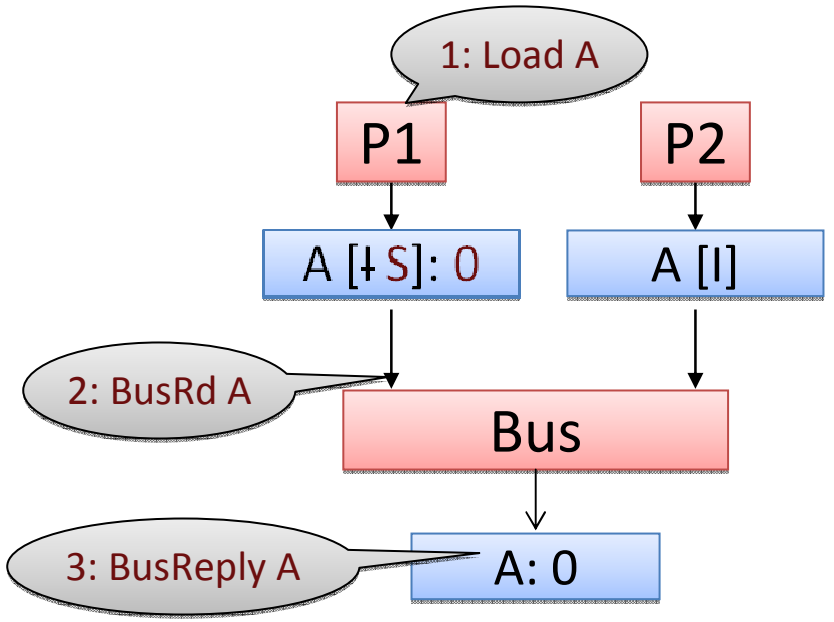
- Write-back caches are good
  - Drastically reduce bus write bandwidth
- Add notion of "*ownership*" to Valid-Invalid
  - When "*owner*" has only replica of a cache block
    - Update it freely
  - Multiple readers are ok
    - Not allowed to write without gaining ownership
  - On a read, system must check if there is an owner
    - If yes, take away ownership

Modified-Shared-Invalid (MSI) States

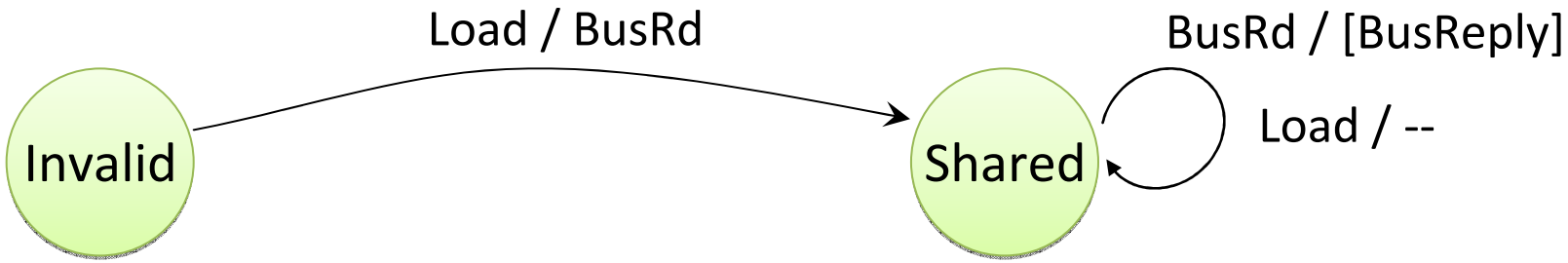
- Processor Actions
  - Load, Store, Evict
- Bus Messages
  - BusRd, BusRdX, BusInv, BusWB, BusReply
    (Here for simplicity, some messages can be combined)
- Track 3 states per cache frame
  - *Invalid*: cache does not have a copy
  - *Shared*: cache has a read-only copy; clean
    - Clean: memory (or later caches) is up to date
  - *Modified*: cache has the only valid copy; writable; dirty
    - Dirty: memory (or later caches) is out of date

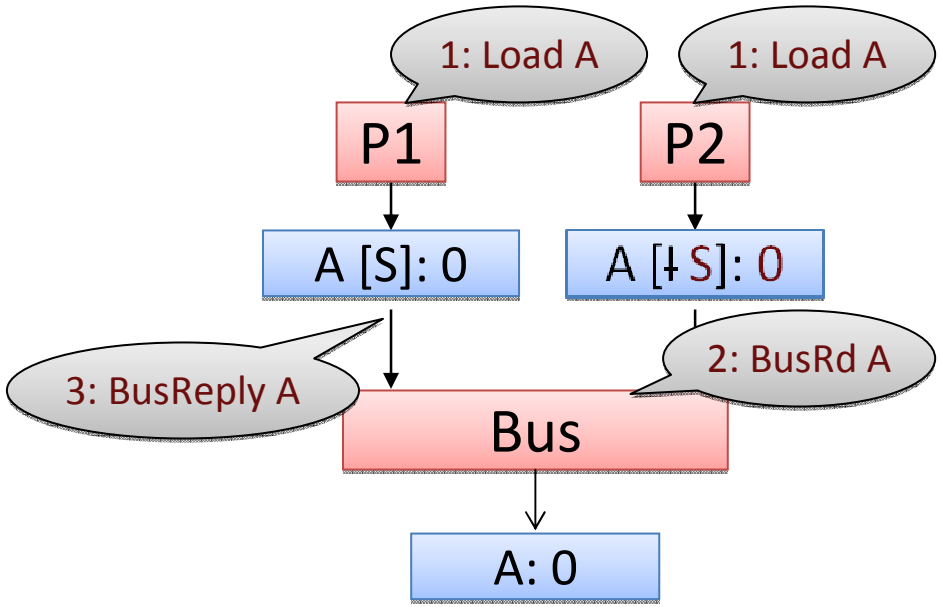
Simple MSI Protocol (1/9)



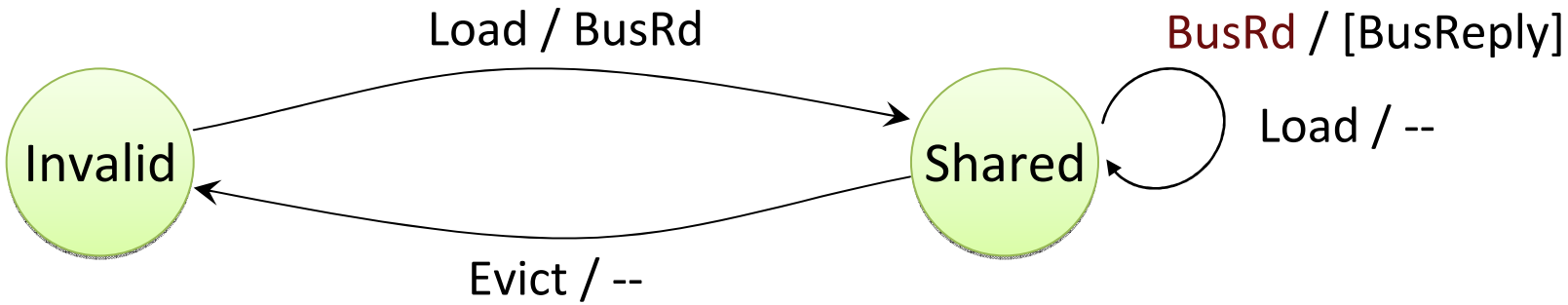


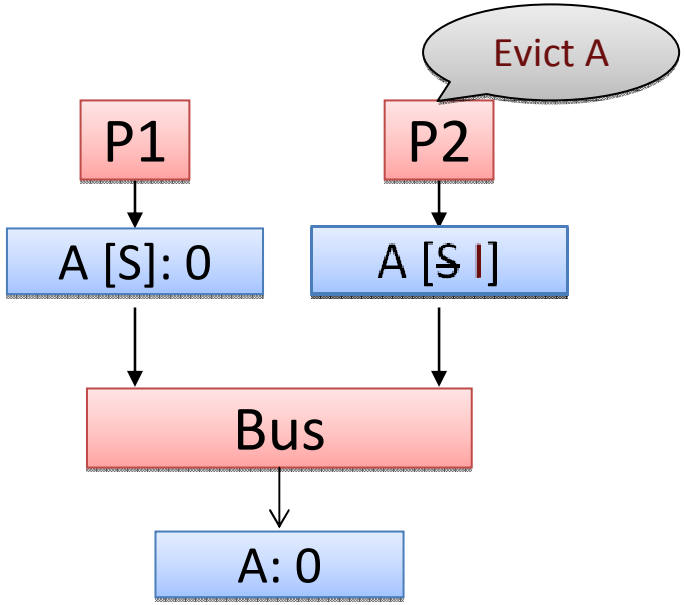
Simple MSI Protocol (2/9)



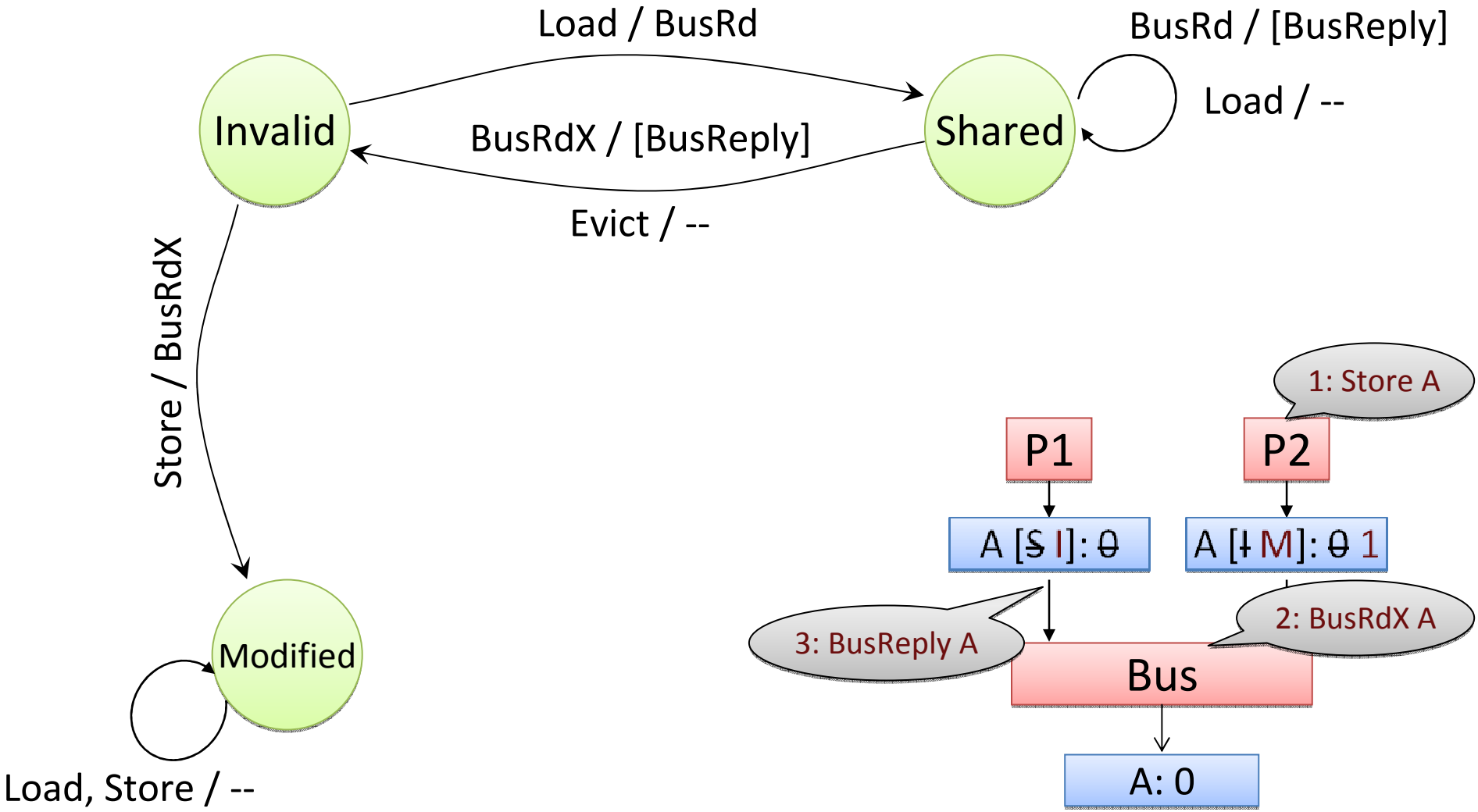


Simple MSI Protocol (3/9)

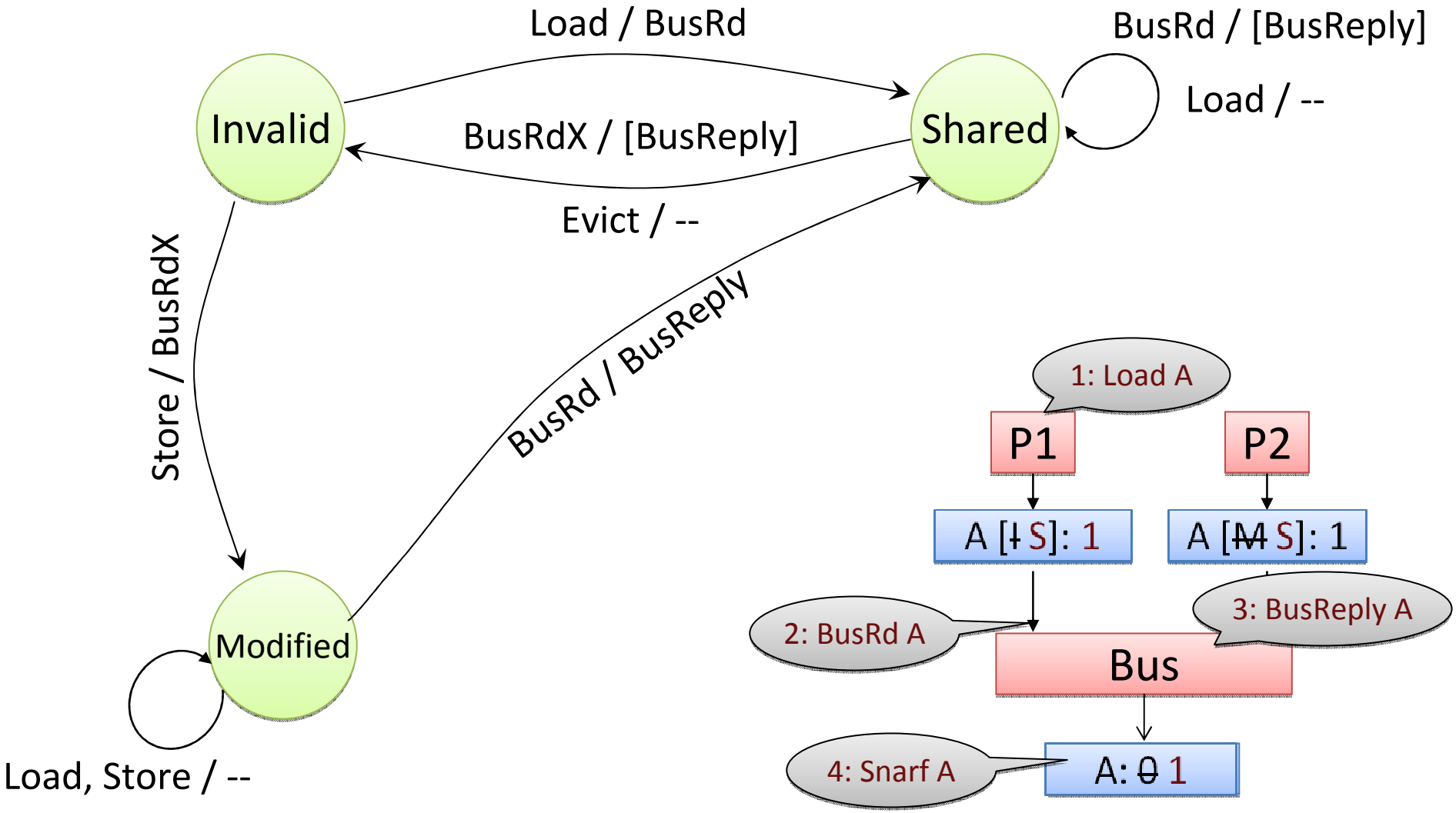




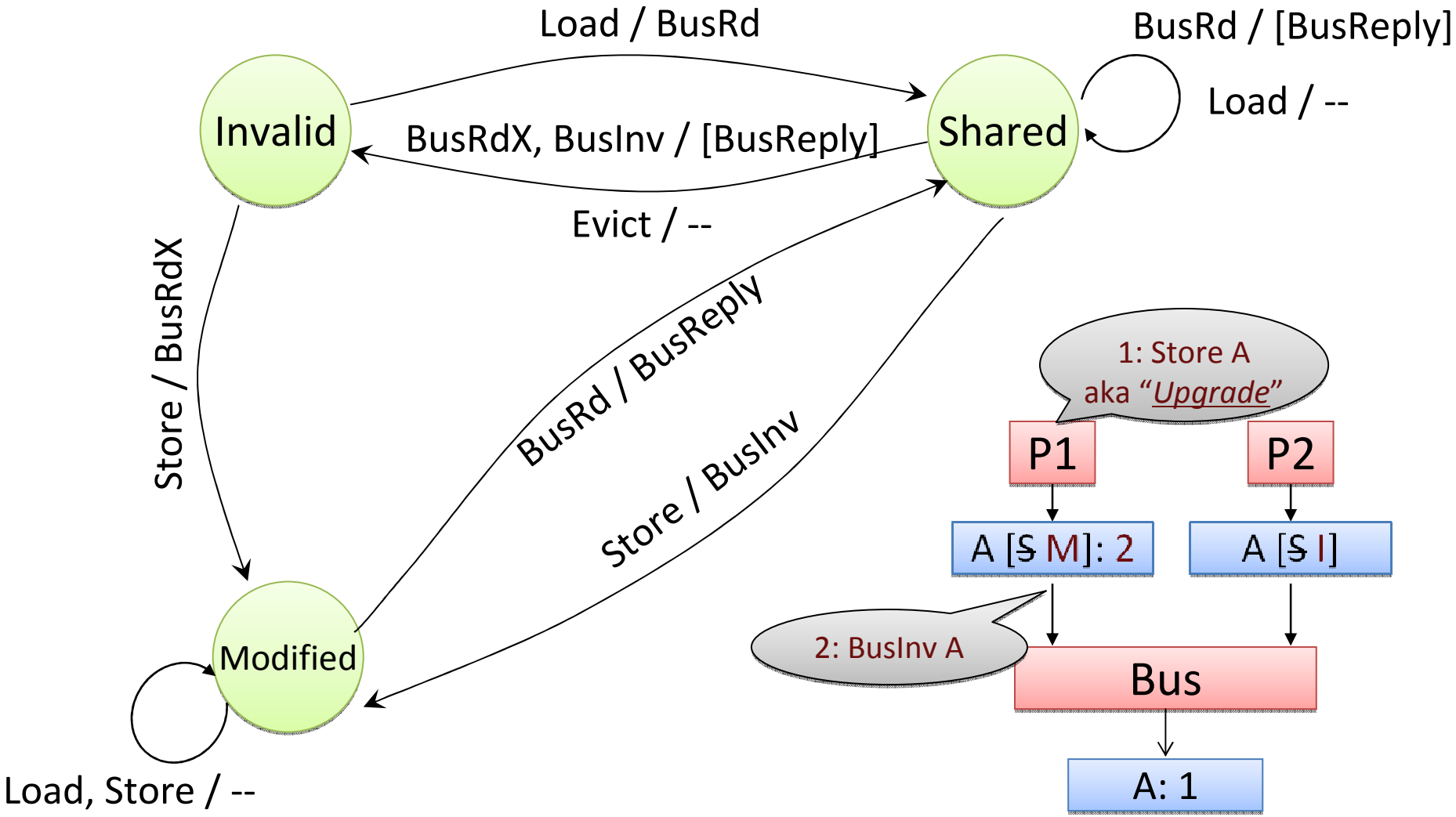
Simple MSI Protocol (4/9)



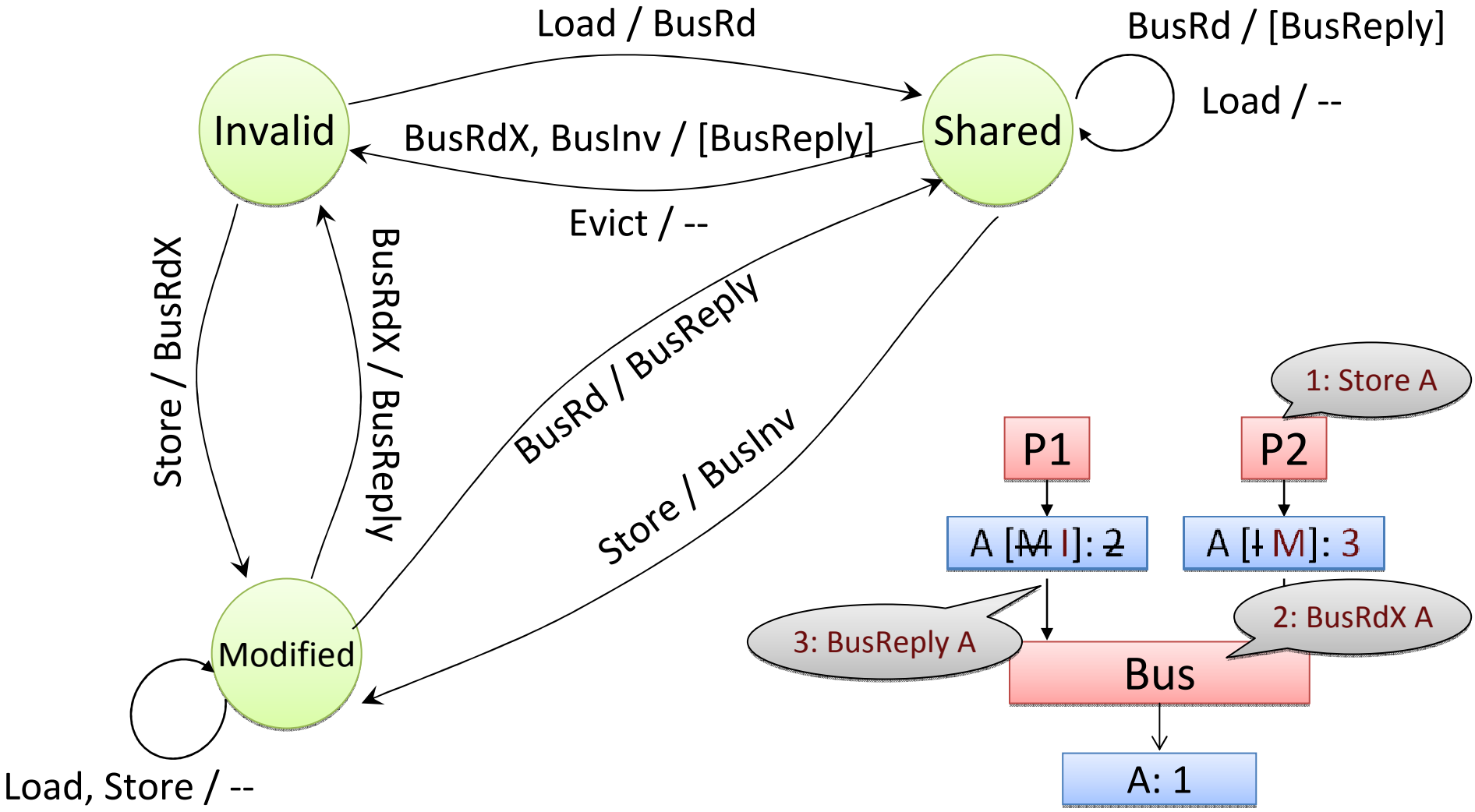
Simple MSI Protocol (5/9)



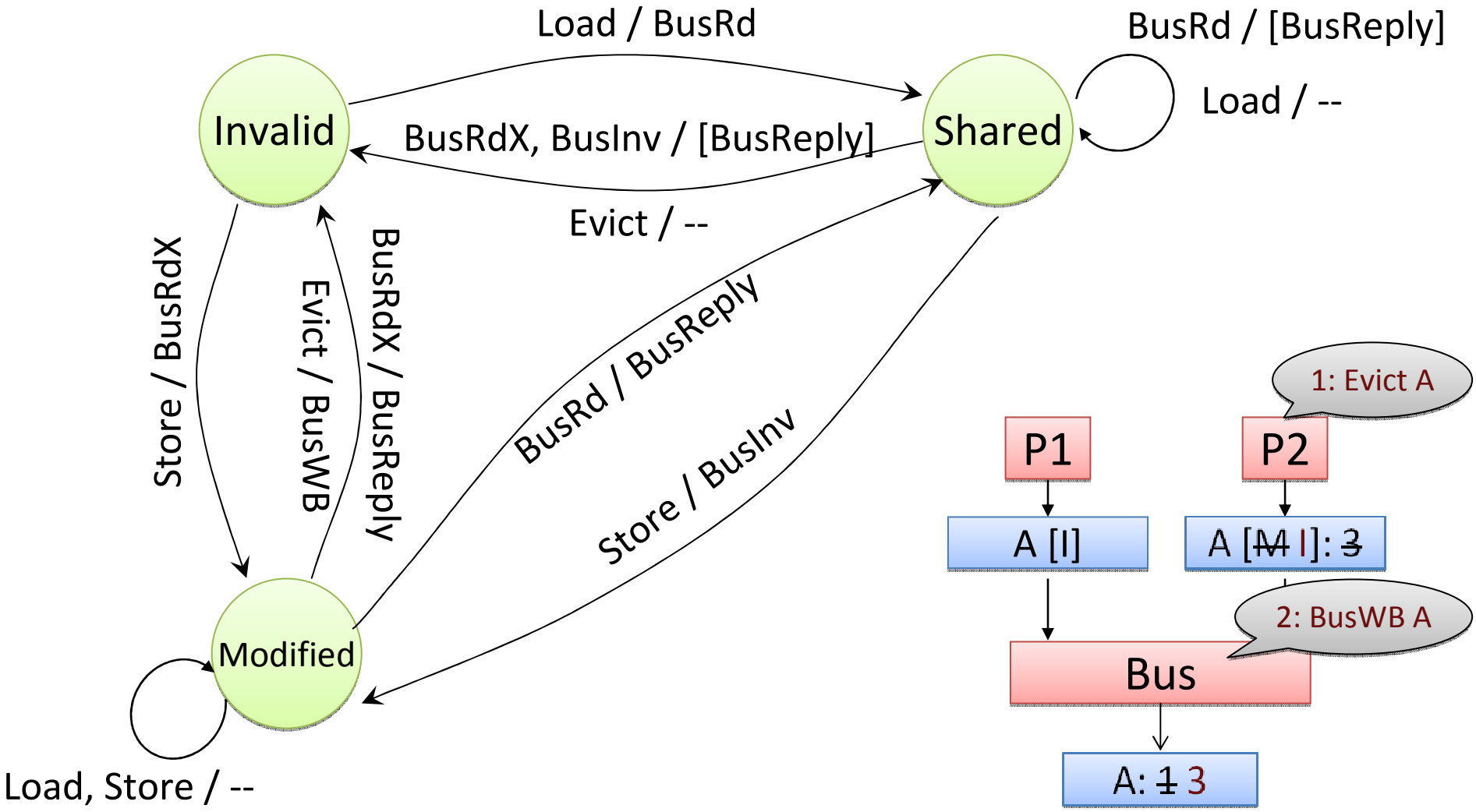
Simple MSI Protocol (6/9)



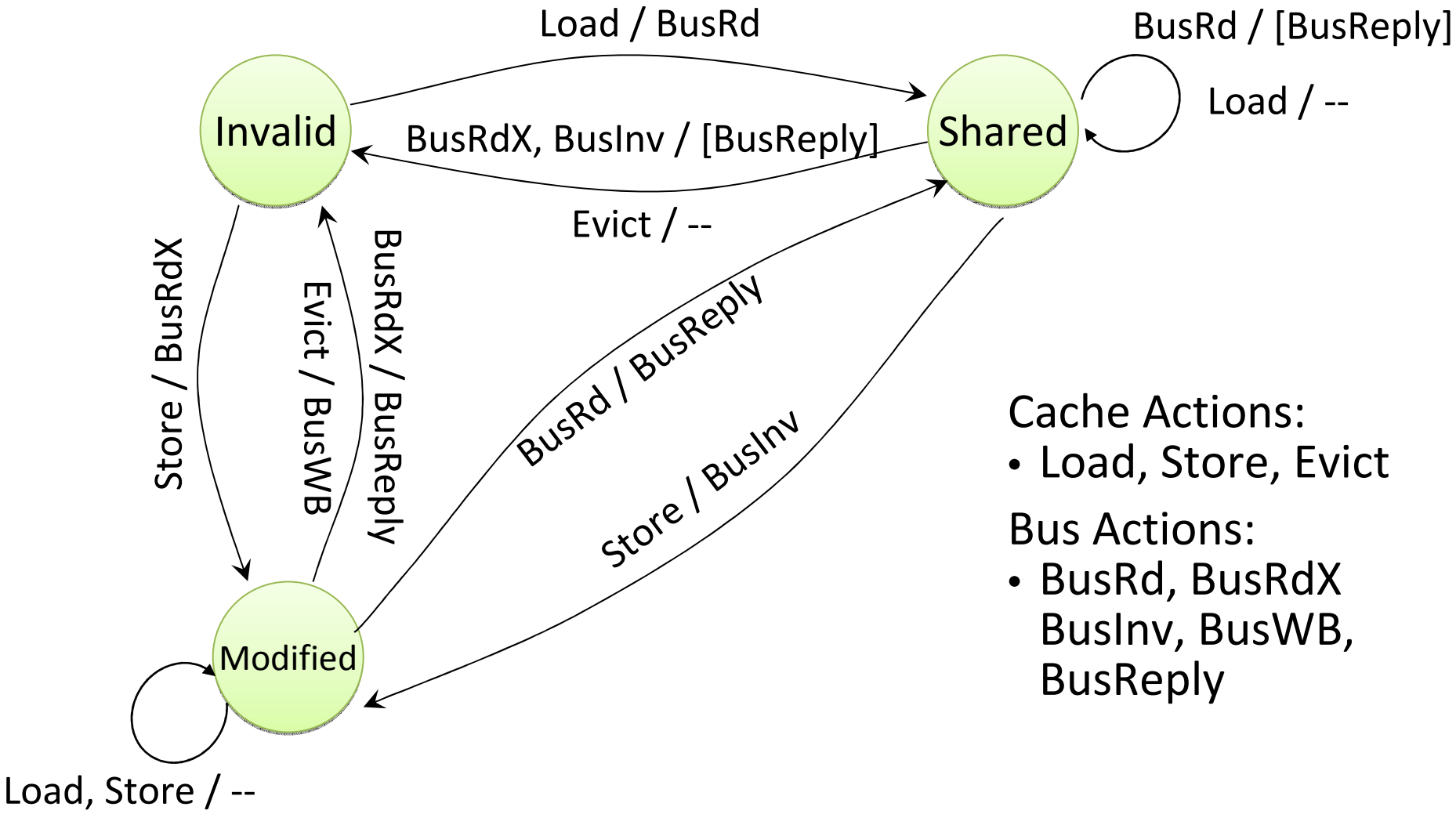
Simple MSI Protocol (7/9)



Simple MSI Protocol (8/9)



Simple MSI Protocol (9/9)



Cache Actions:

- Load, Store, Evict

Bus Actions:

- BusRd, BusRdX BusInv, BusWB, BusReply

Scalable Cache Coherence

- Part I: bus bandwidth
  - Replace non-scalable bandwidth substrate (bus)
    …with scalable-bandwidth one (e.g., mesh)
- Part II: processor snooping bandwidth
  - Most snoops result in no action
  - Replace non-scalable broadcast protocol (spam everyone)
    …with scalable directory protocol (spam cores that care)