# Multicores and Cache coherence

Arkaprava Basu

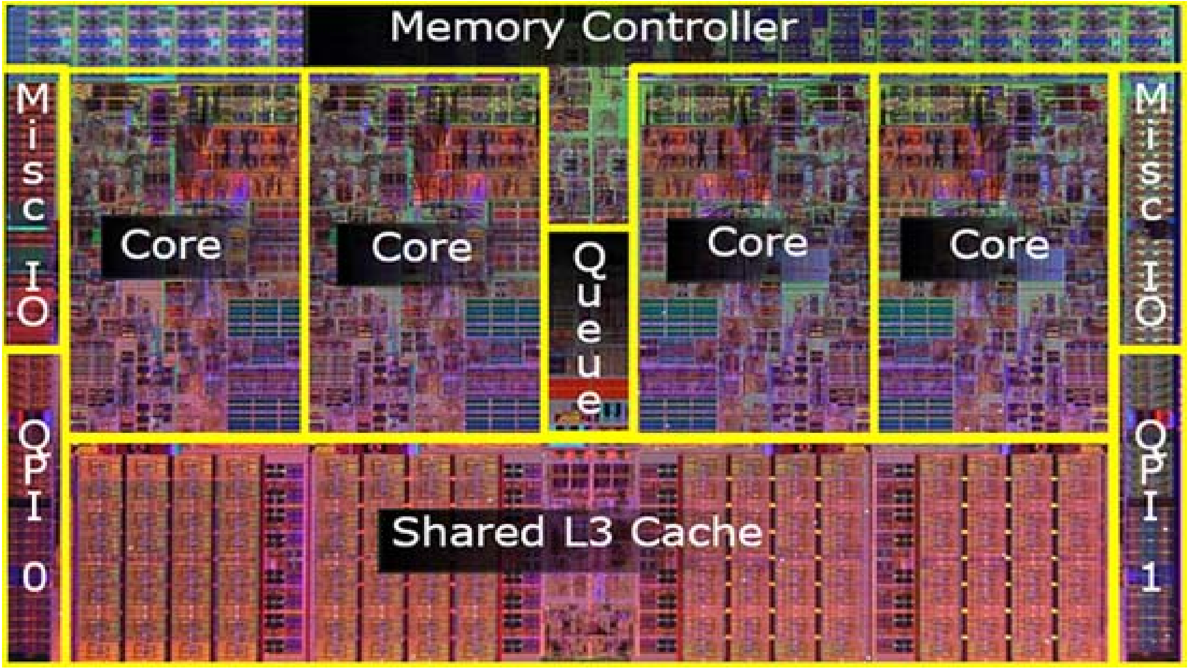
https://www.csa.iisc.ac.in/~arkapravab/

# Acknowledgements

- Some of the slides in the deck were provided by Luis Ceze (Washington), Nima Horanmand (Stony Brook), Mark Hill, David Wood, Karu Sankaralingam (Wisconsin), Abhishek Bhattacharjee(Yale).

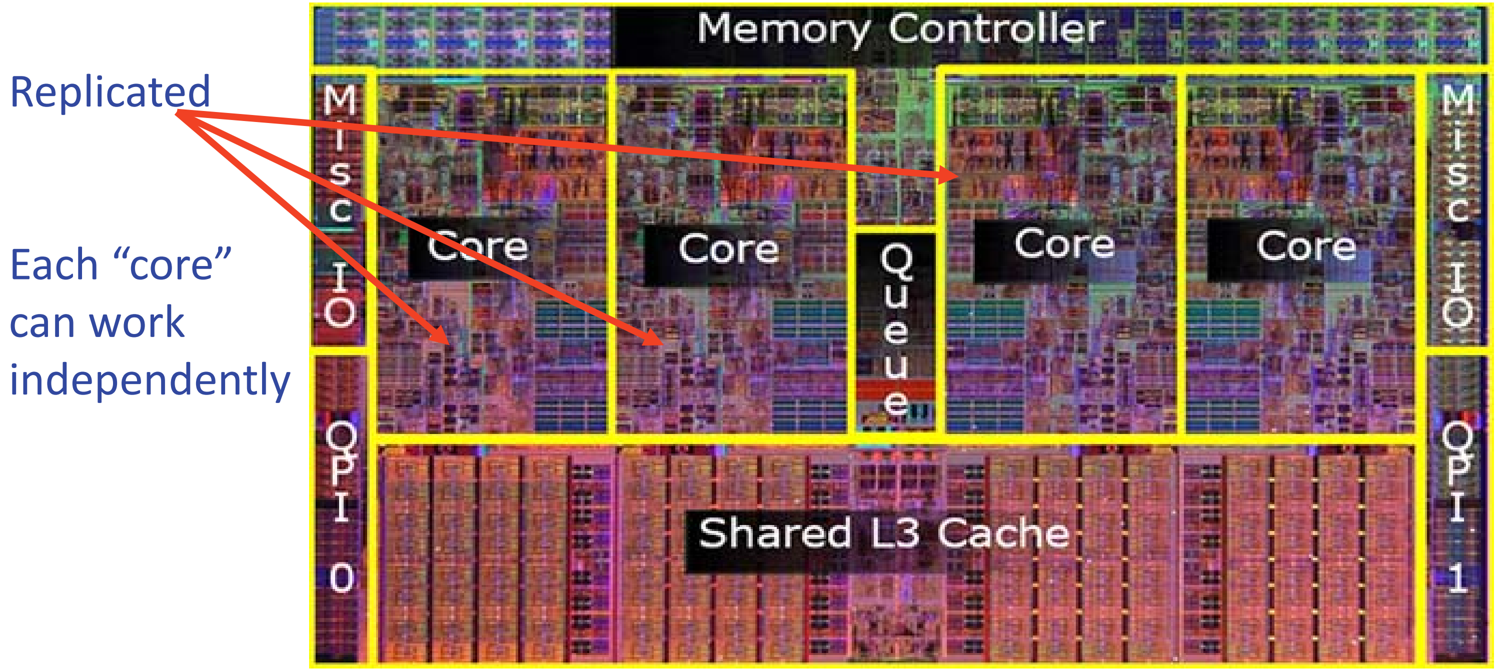
# What are Multicore chips?

- Multiple replicated cores on a single chip



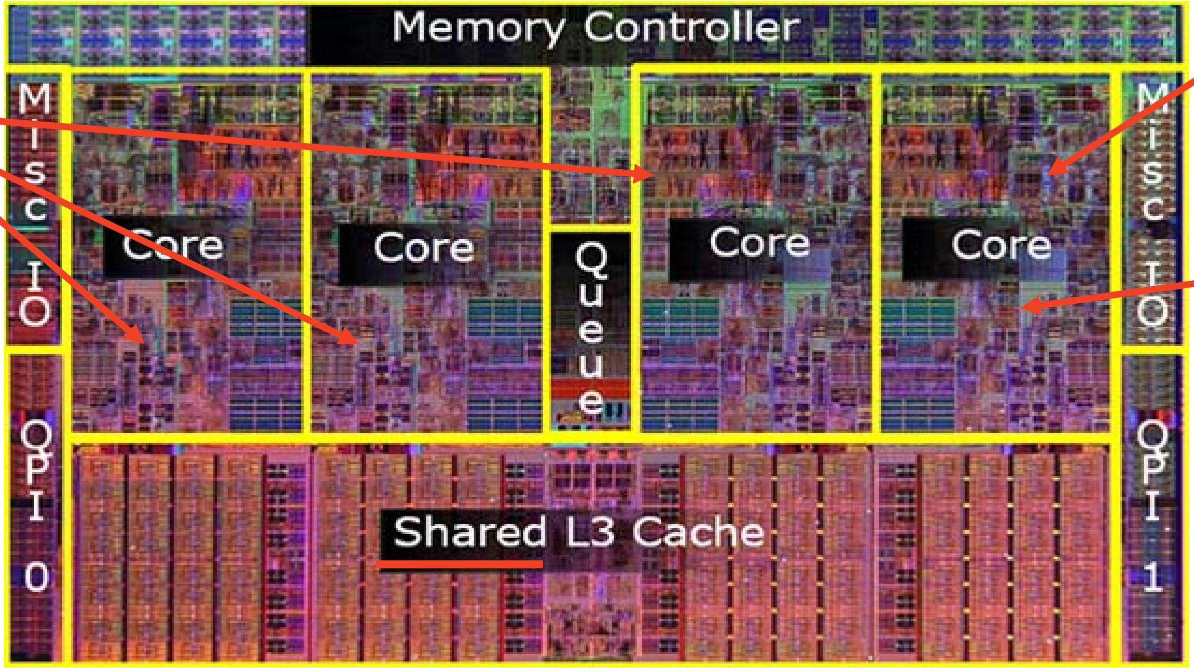
# What are Multicore chips?

- Multiple replicated cores on a single chip



# What are Multicore chips?

- Multiple replicated cores on a single chip



Replicated

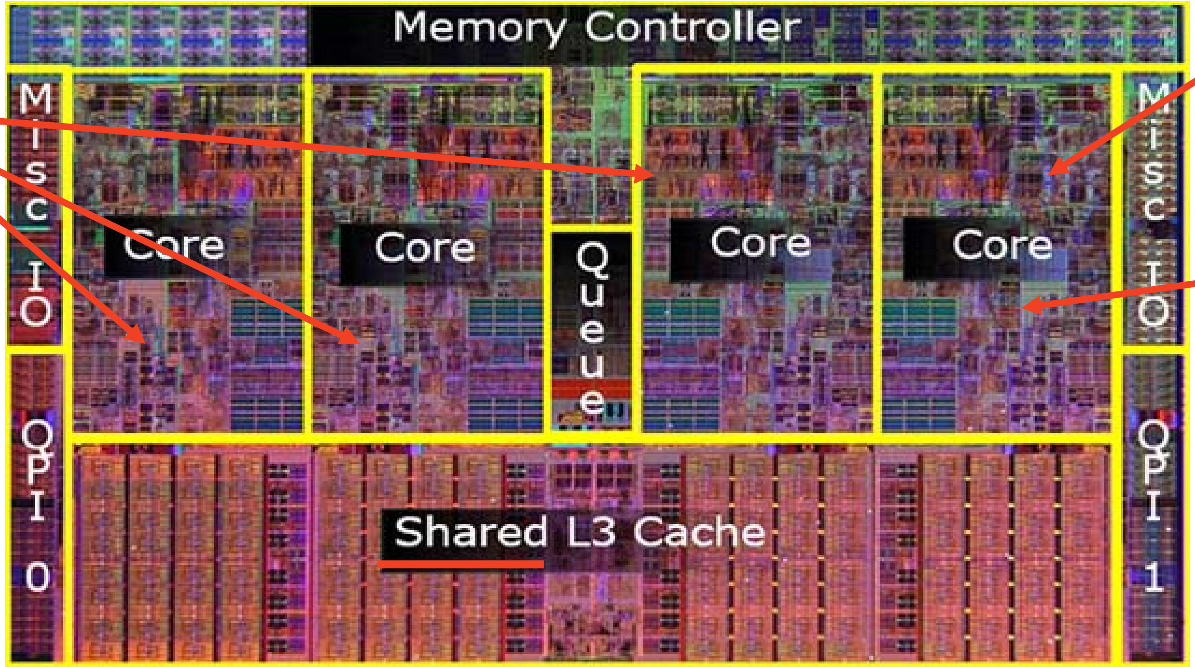
Each "core" can work independently

Pipeline, execution units

**Private** L1/L2 caches

# What are Multicore chips?

- Multiple replicated cores on a single chip



Replicated

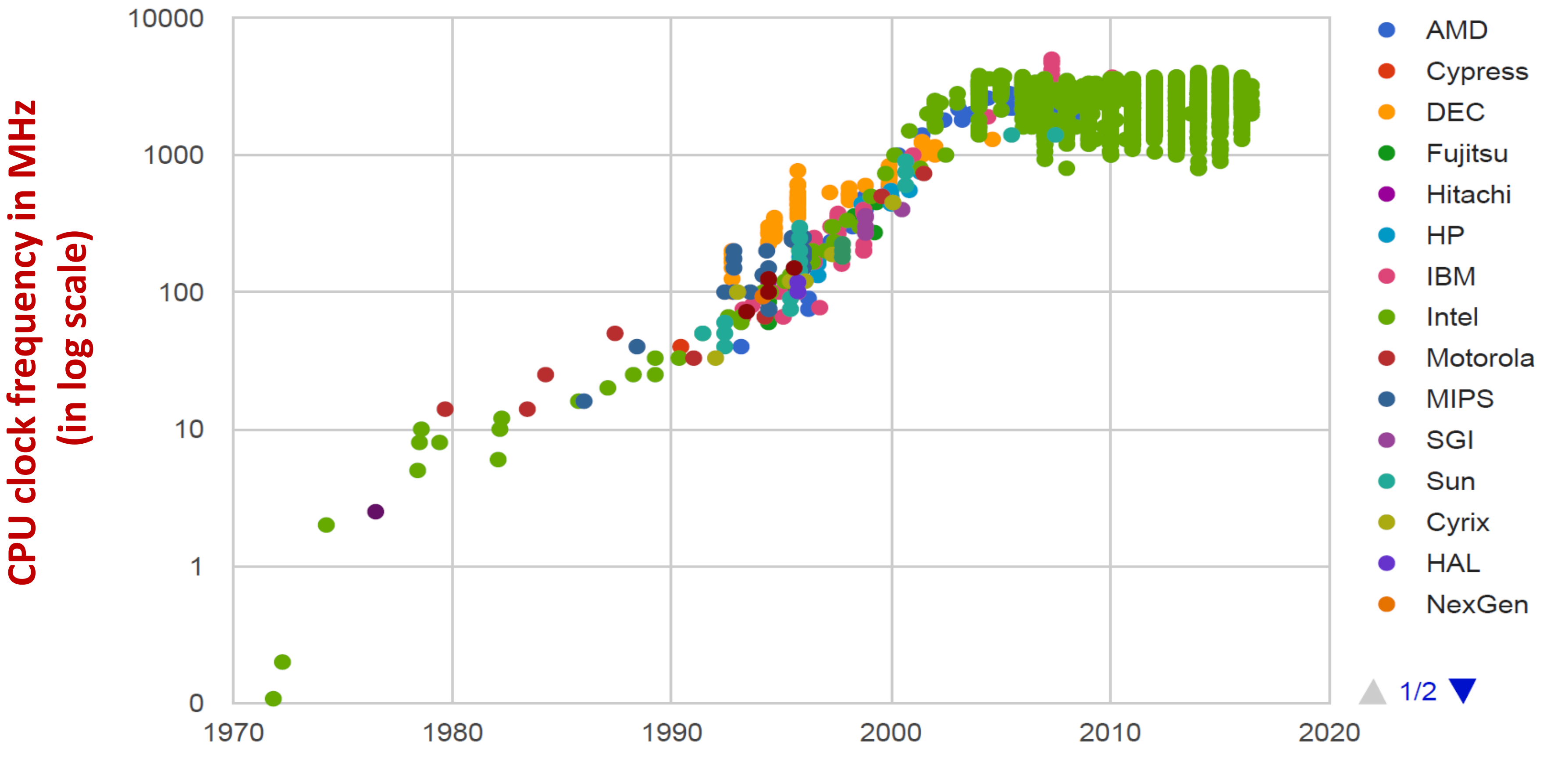
Each “core” can work independently

Pipeline, execution units

**Private** L1/L2 caches

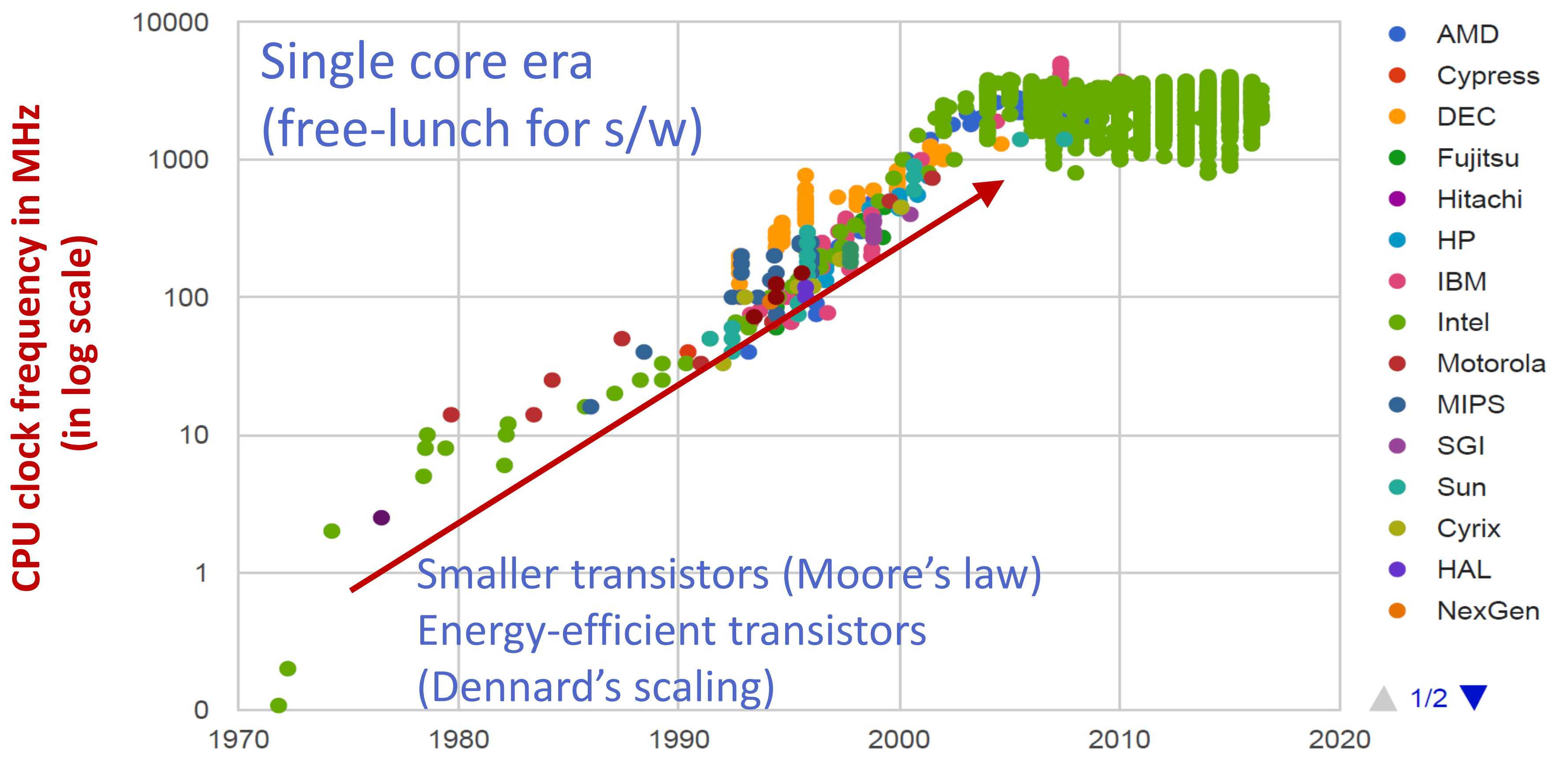
Examples: Apple A14 (iPhone 12) 6 cores
Intel i7-10700T (desktop class) 8 cores
AMD Epyc gen 2 (server class) 64 cores

# Why have Multicores?



Historical data from Stanford CUPDB

# Why have Multicores?

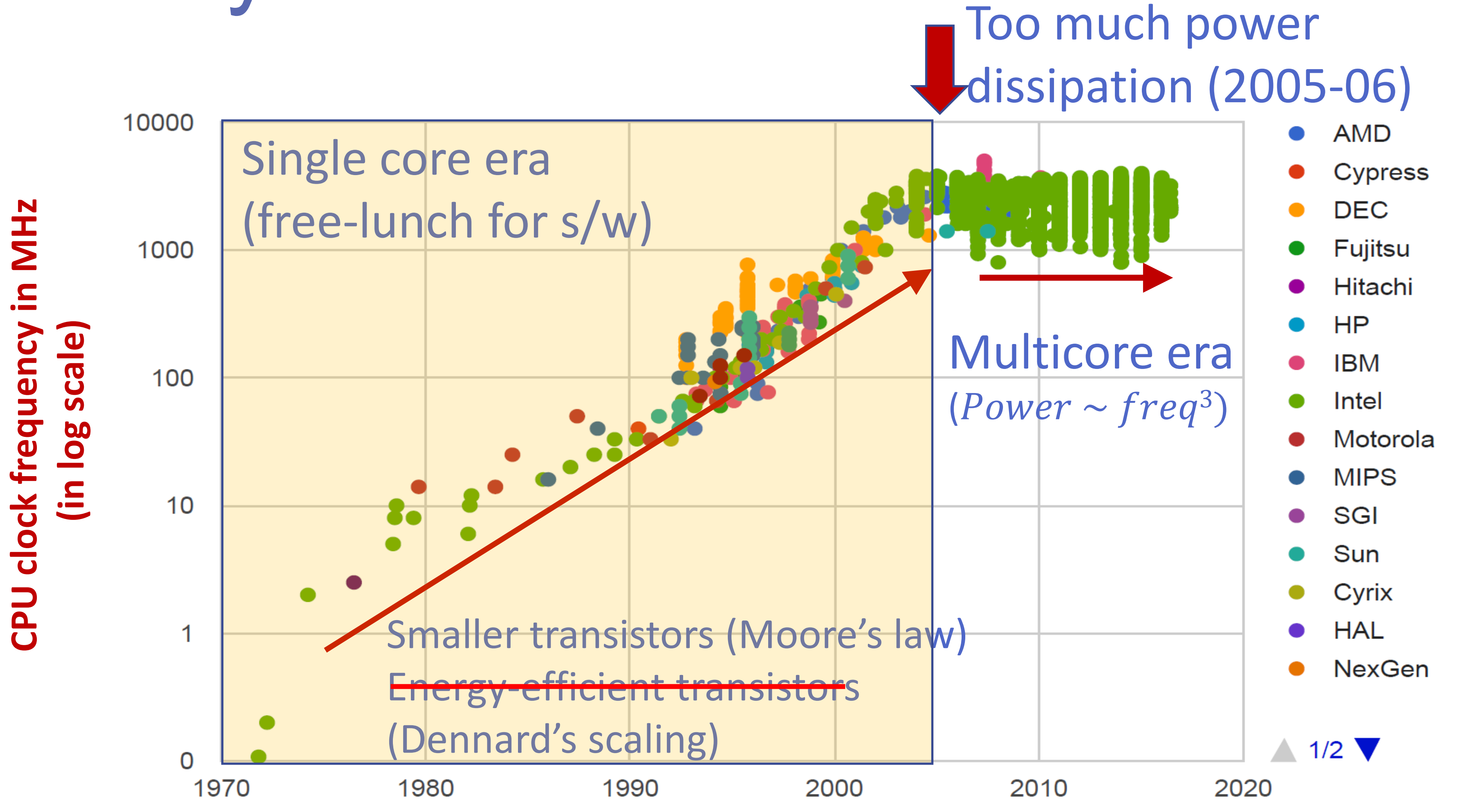


Single core era
(free-lunch for s/w)

Smaller transistors (Moore's law)
Energy-efficient transistors
(Dennard's scaling)

Historical data from Stanford CUPDB

# Why have Multicores?



Too much power dissipation (2005-06)

Single core era (free-lunch for s/w)

Multicore era ($Power \sim freq^3$)

Smaller transistors (Moore's law)

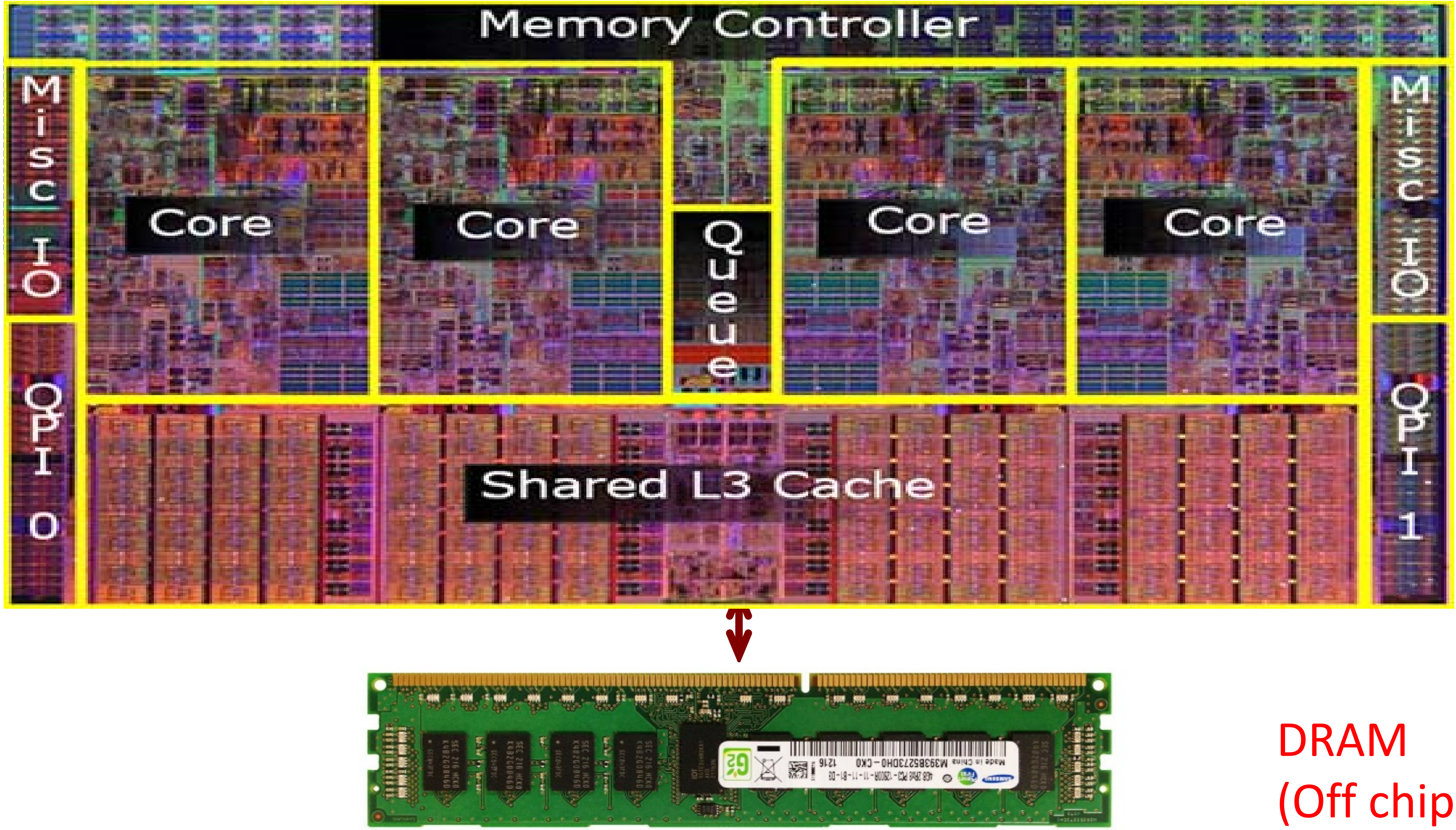
~~Energy-efficient transistors~~

(Dennard's scaling)

Historical data from Stanford CUPDB

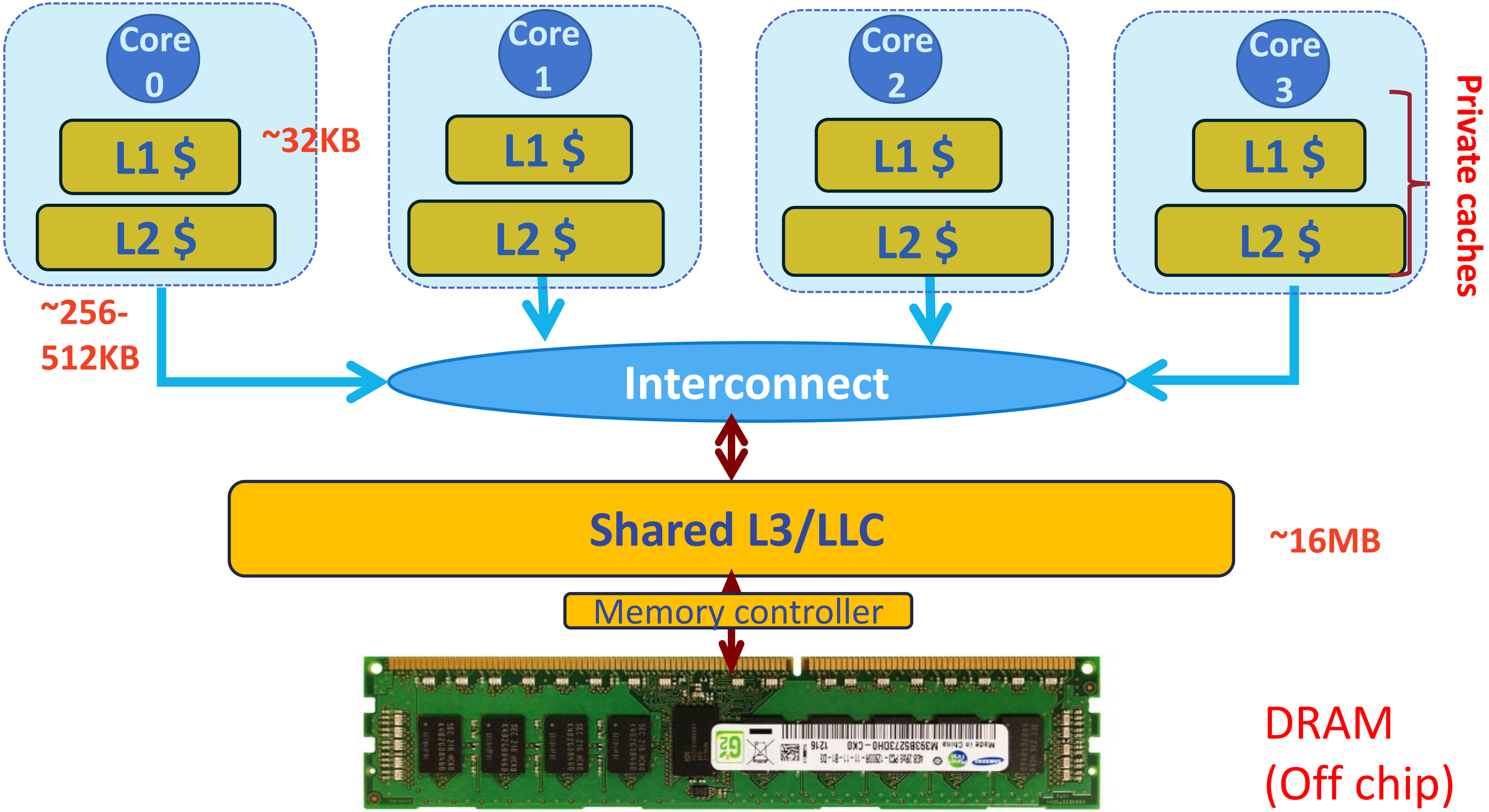
Multicores was a compulsion and not a choice

# Schematic: Physical connection

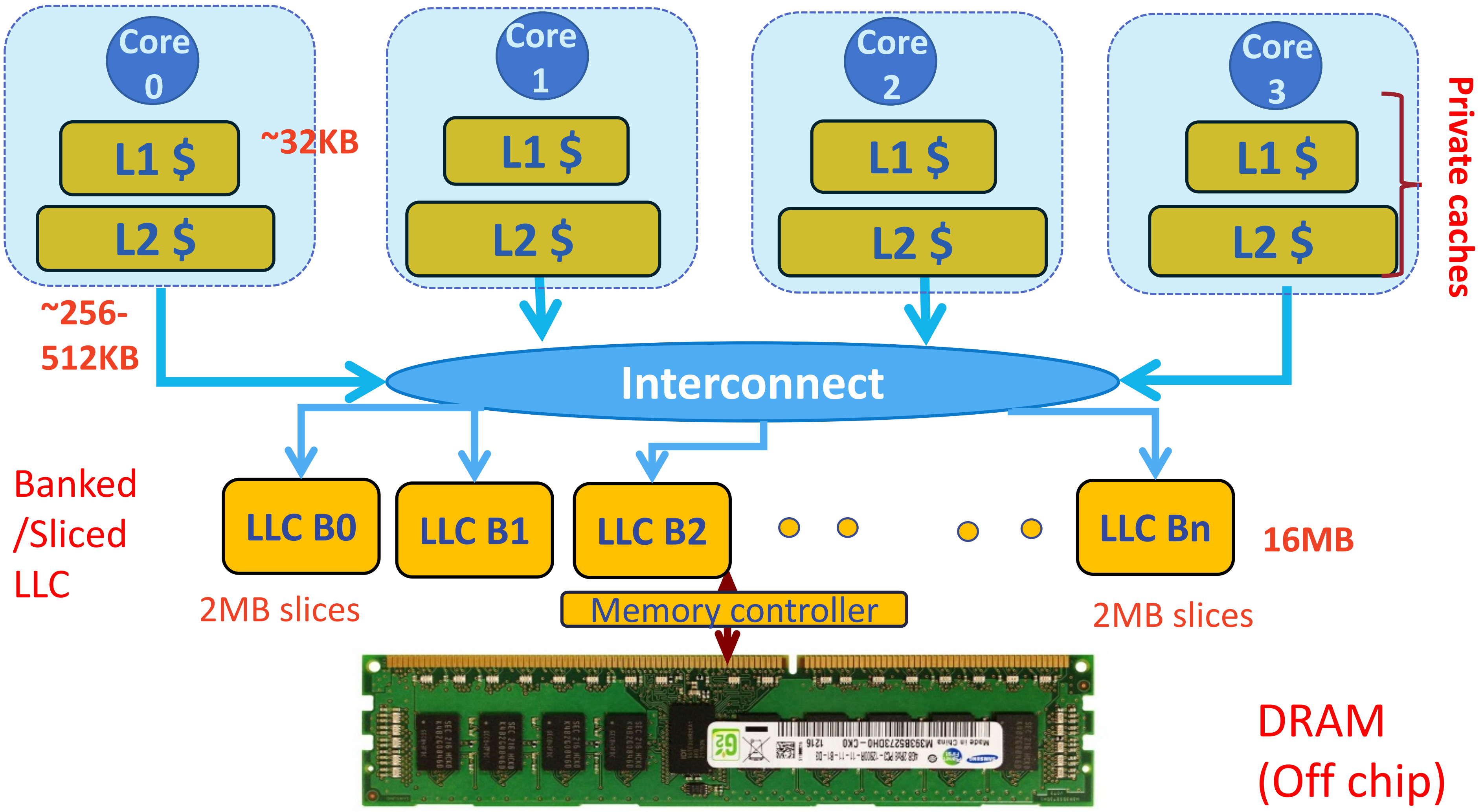


DRAM
(Off chip)

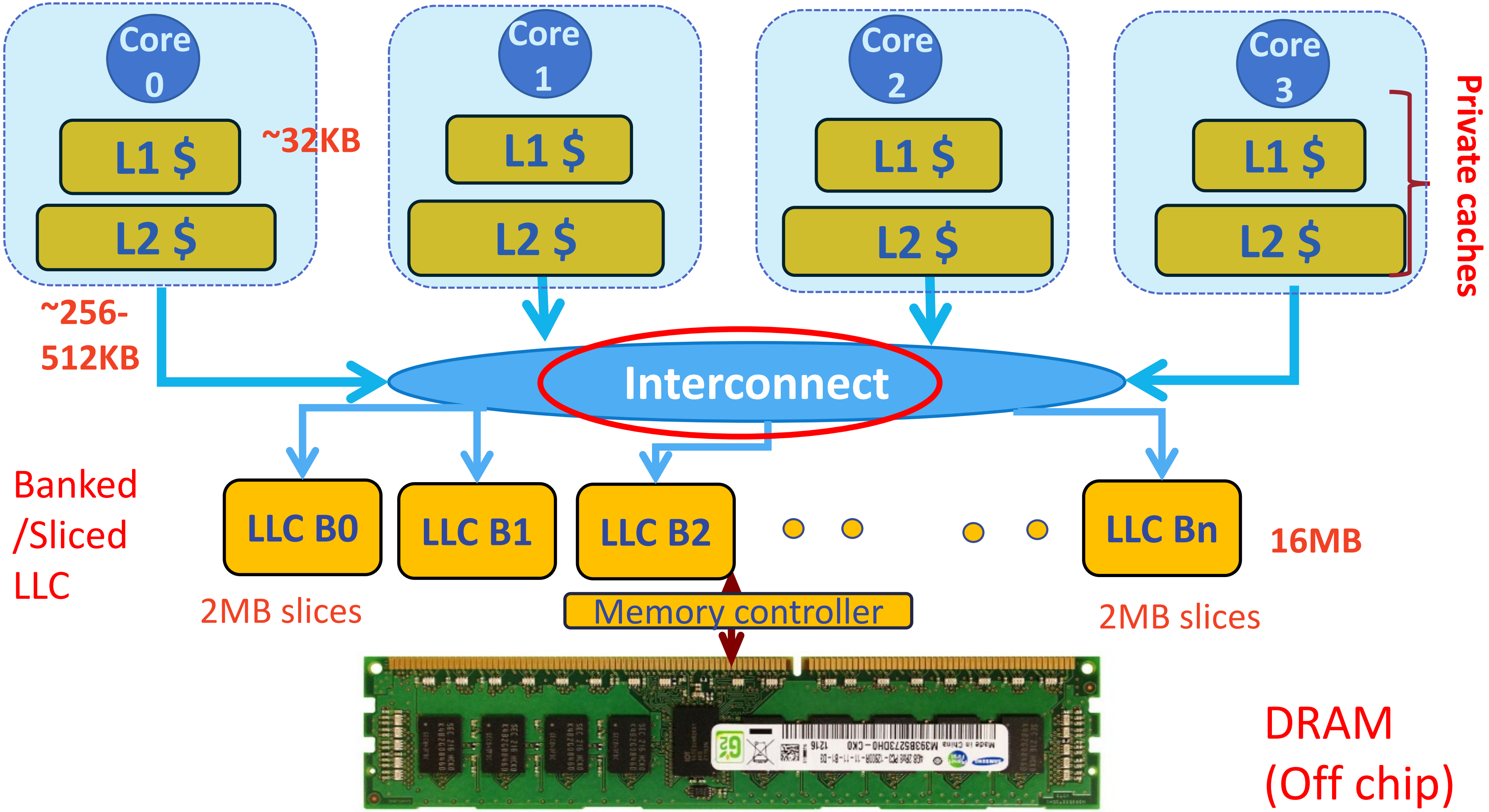
# Schematic: Physical connection



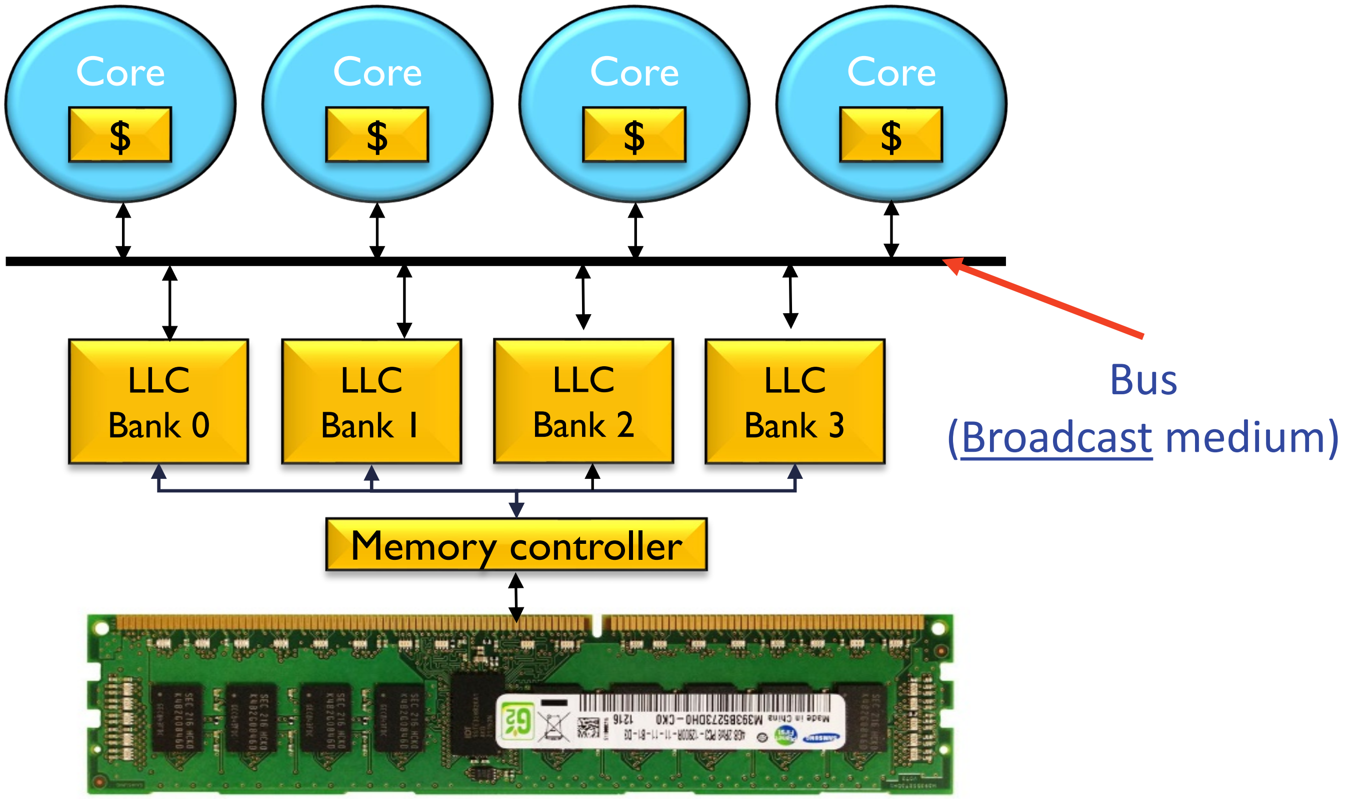
# Schematic: Physical connection



# Schematic: Physical connection



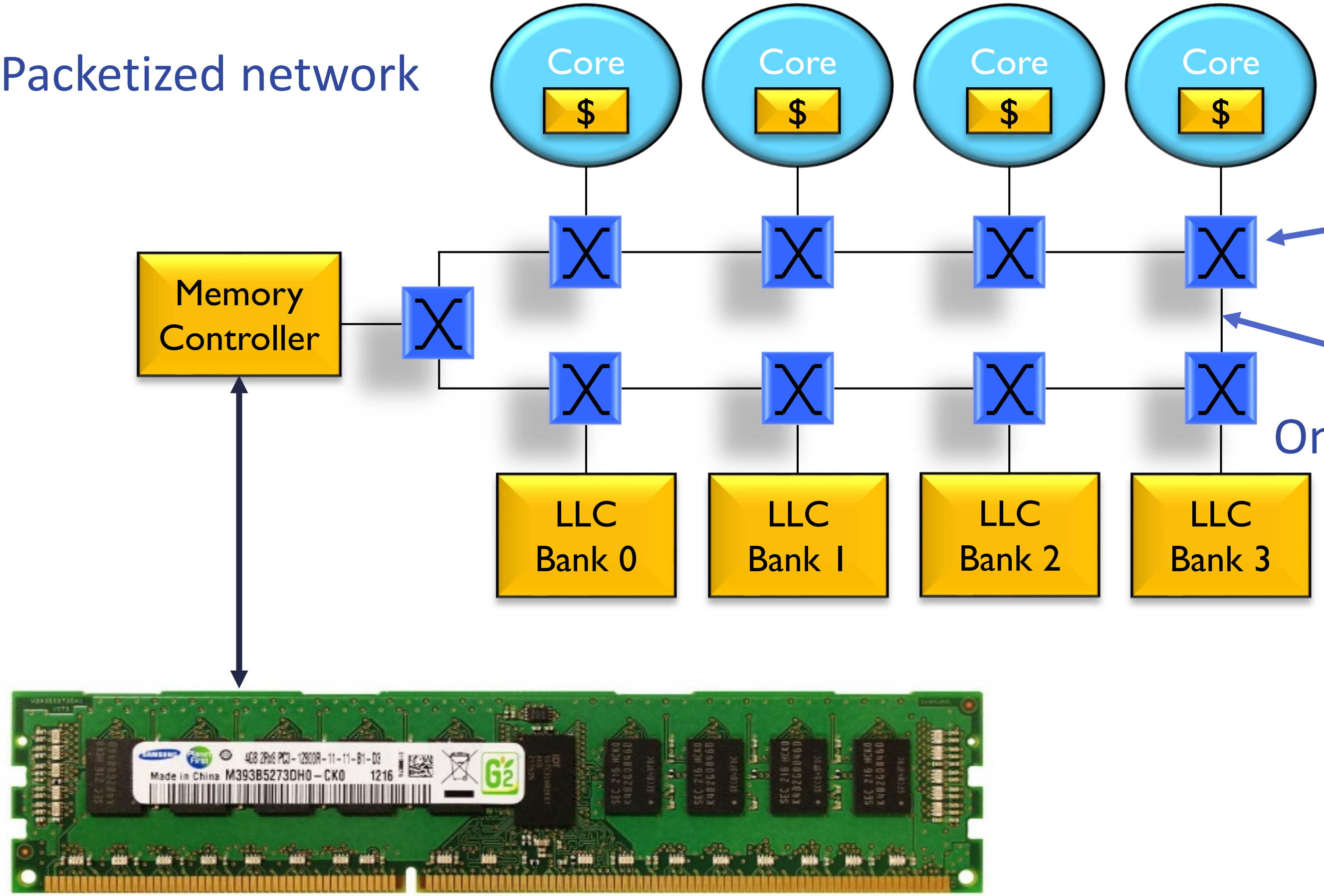
# Choice of Interconnects: Bus



Good for small number of cores

# Choice of Interconnects: Ring

Packetized network

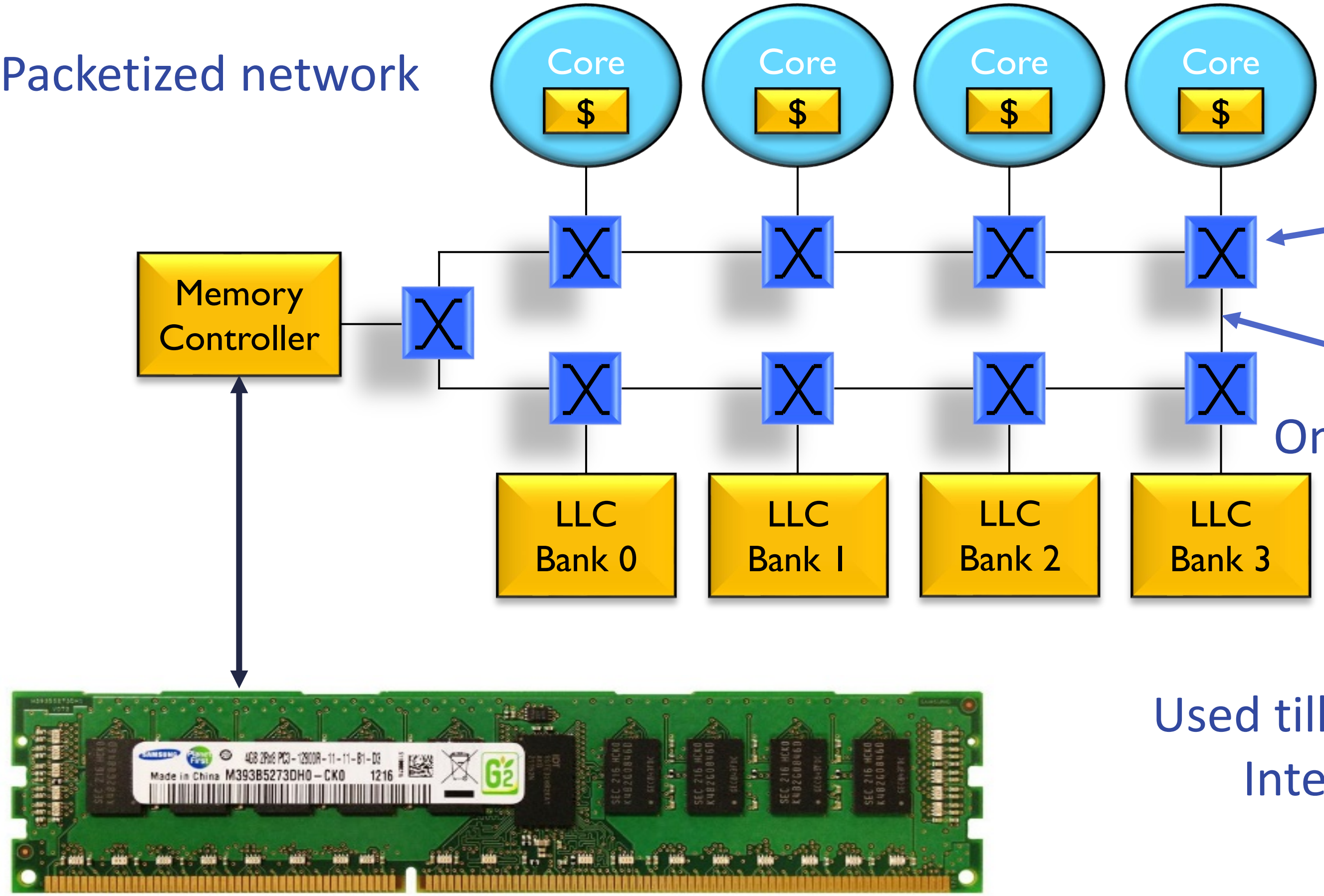


Routers/ switches

Omni-/bi-directional ring

# Choice of Interconnects: Ring

Packetized network



Core $

Core $

Core $

Core $

Memory Controller

X

LLC Bank 0

LLC Bank 1

LLC Bank 2

LLC Bank 3

Routers/ switches

Omni-/bi-directional ring

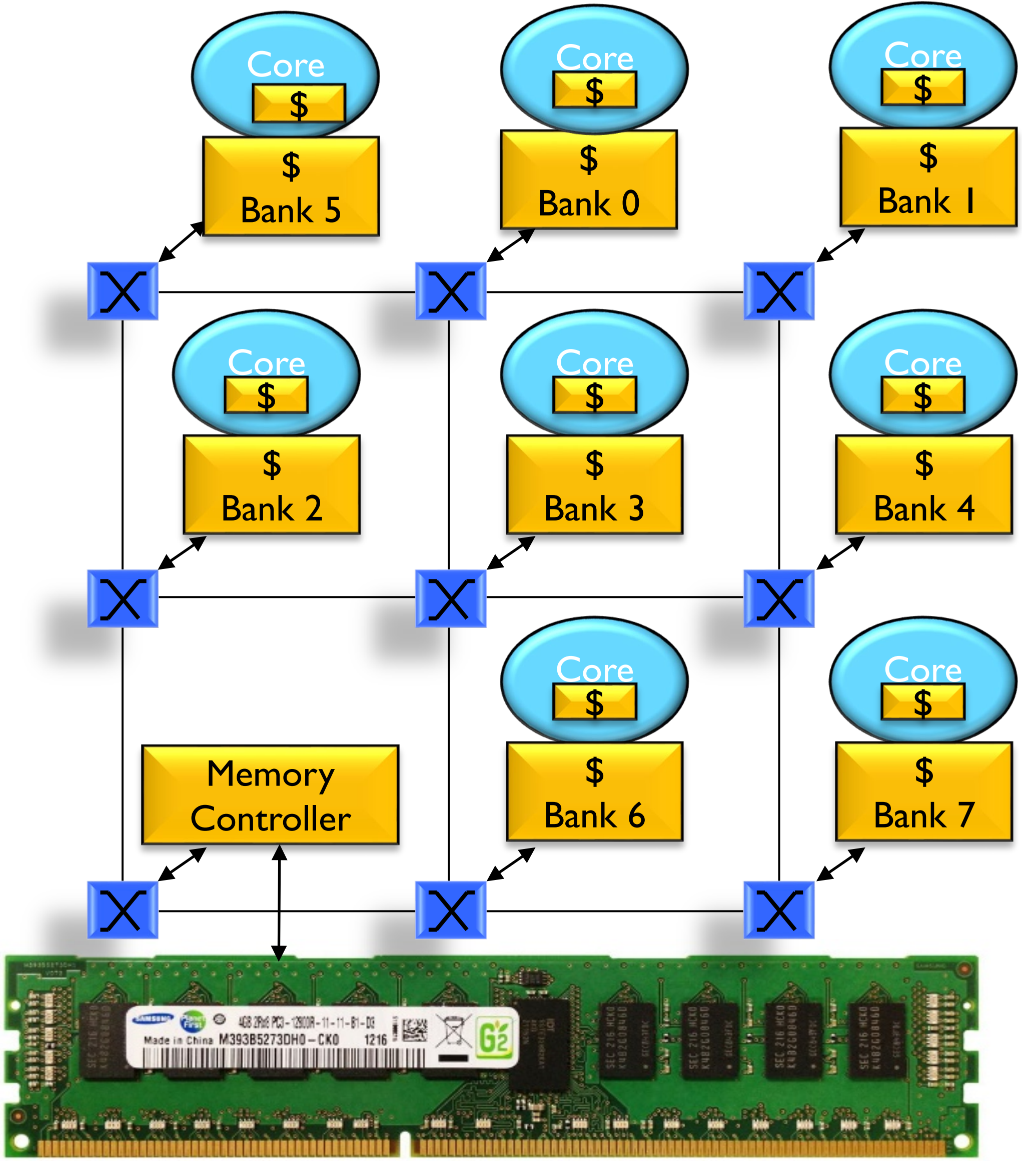
Used till very recently in Intel processors

Can be used as broadcast medium; good for moderate number of cores

# Choice of Interconnects: Mesh

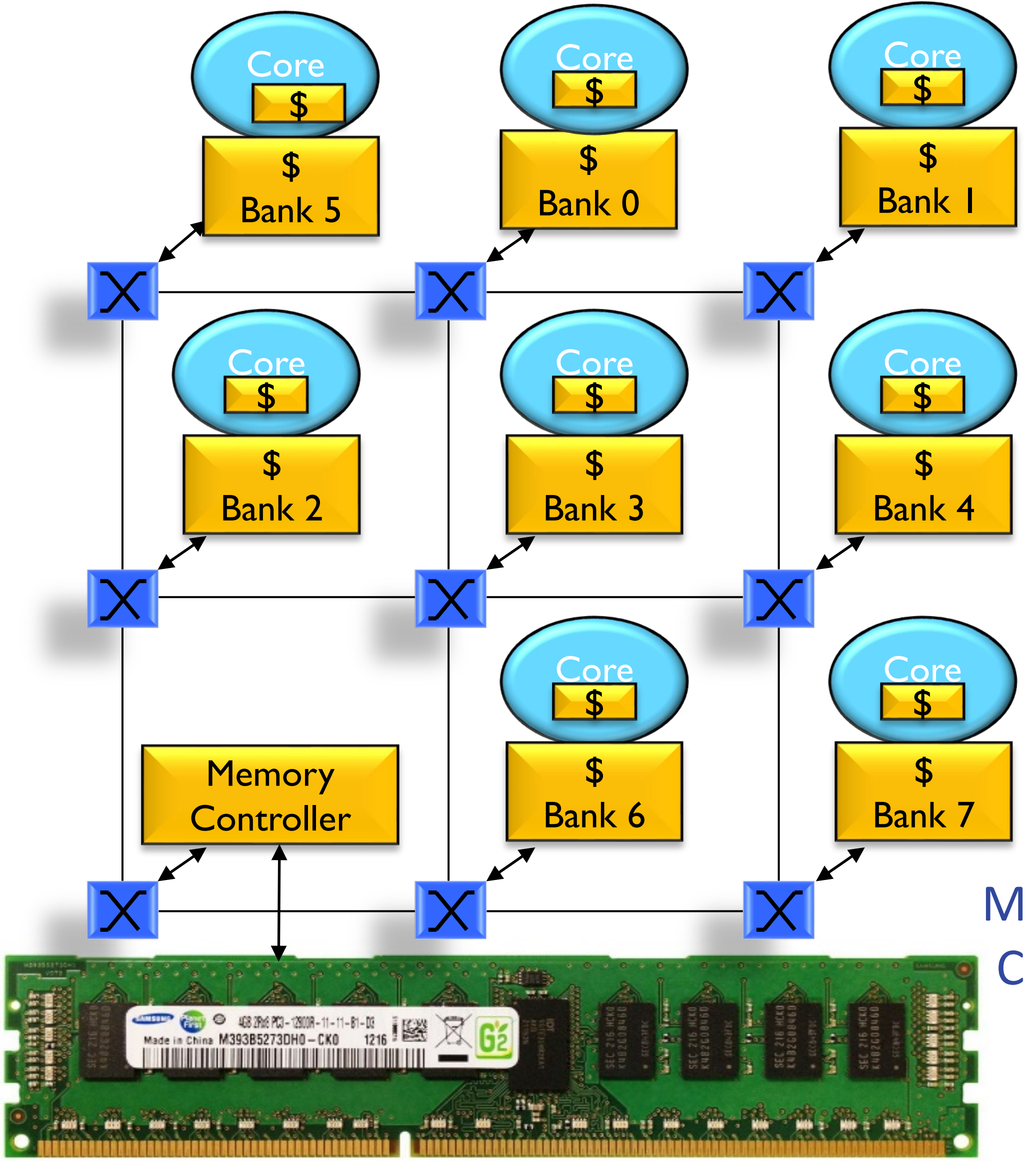
Scalable,
point-to-point,

Better for larger
number f cores



Latest Intel
server processors
uses mesh

# Choice of Interconnects: Mesh



Scalable,
point-to-point,

Better for larger
number f cores

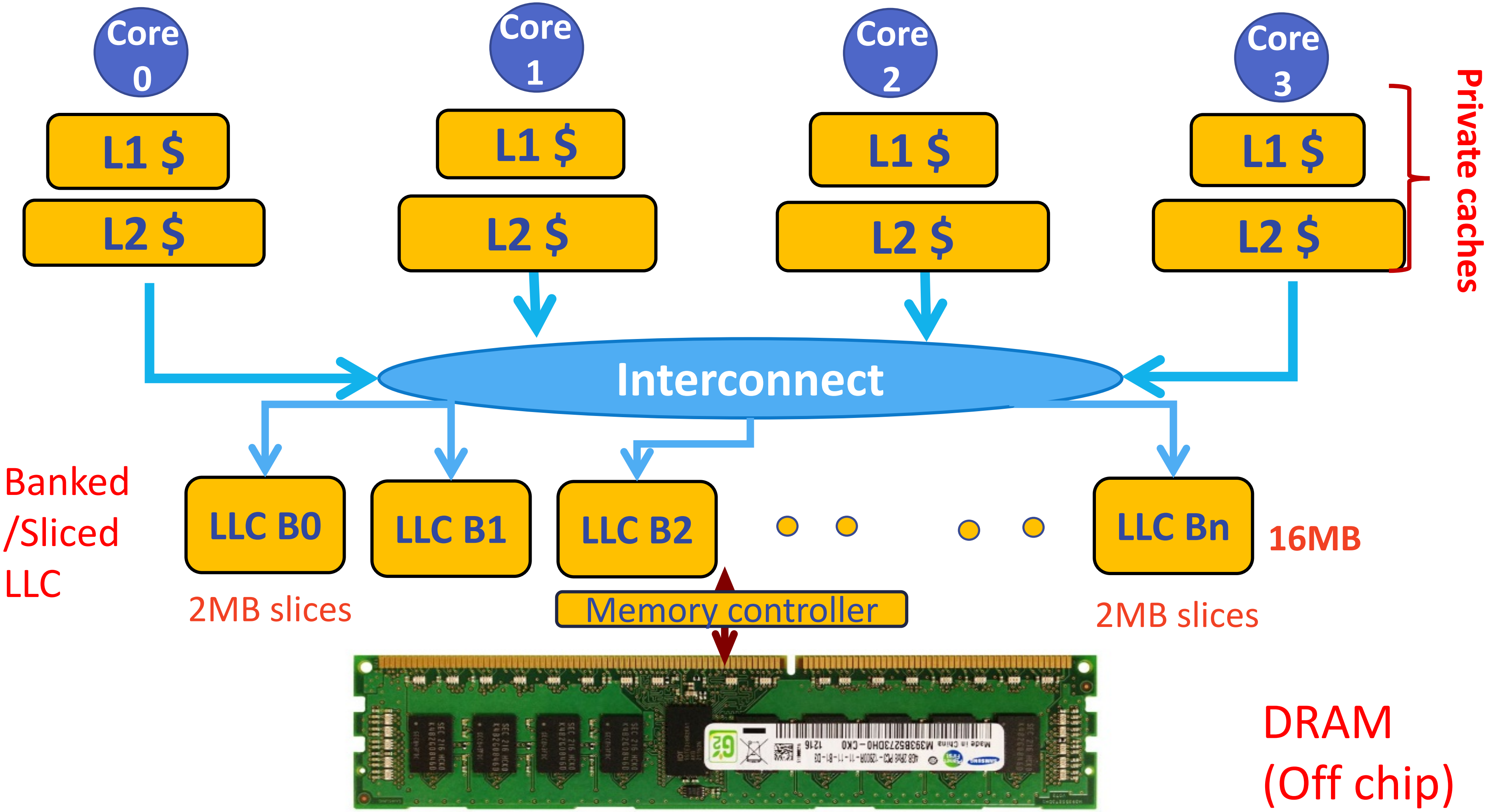
Latest Intel
server processors
uses mesh

Many other options:
Crossbar, torus etc.

# Software's view of multicore

Need for cache coherence

# Software's view of multicore



Core 0

Core 1

Core 2

Core 3

L1 $

L2 $

Private caches

Interconnect

Banked /Sliced LLC

LLC B0

LLC B1

LLC B2

LLC Bn

16MB

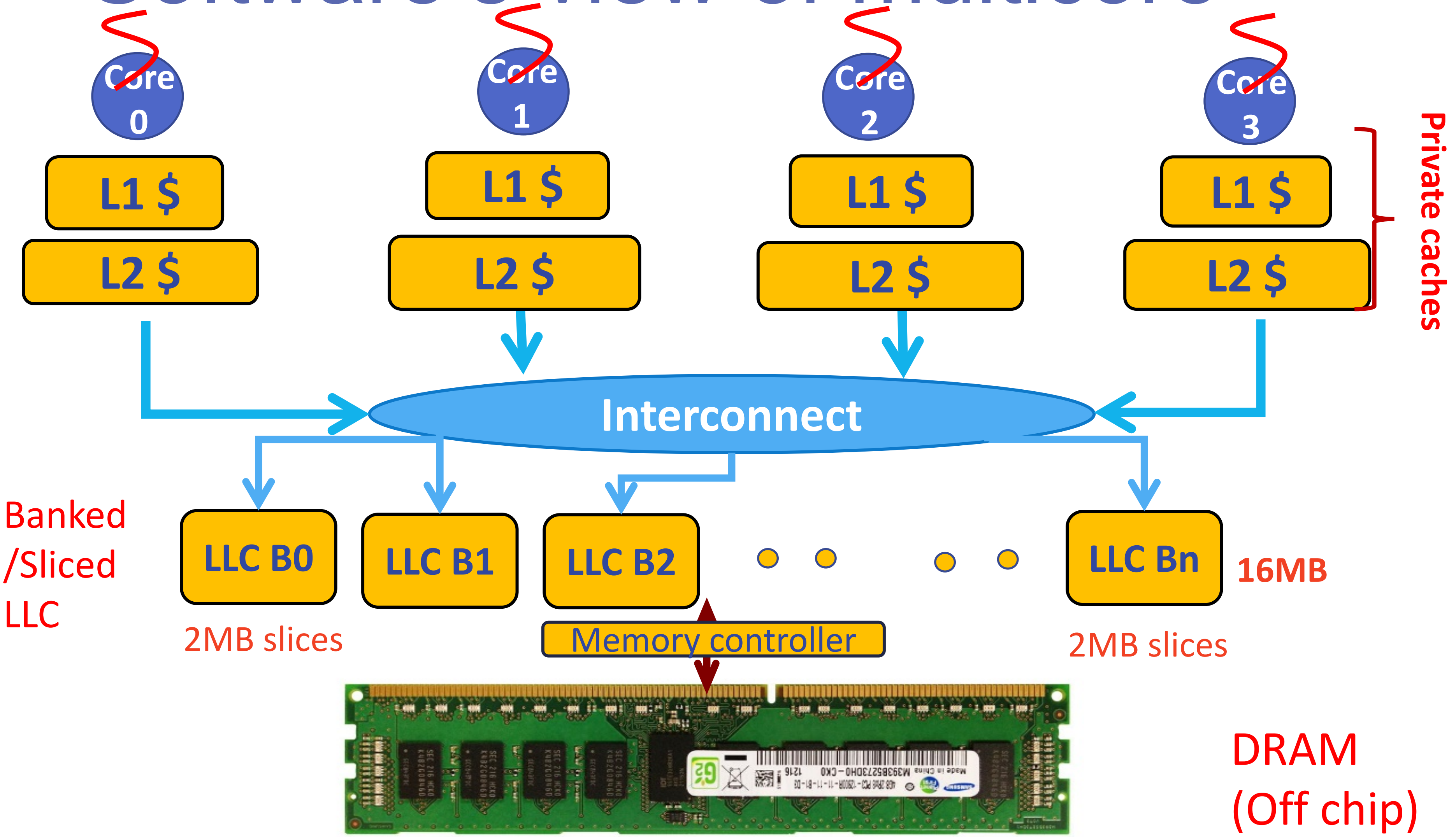
2MB slices

2MB slices

Memory controller

DRAM (Off chip)

# Software's view of multicore



Core 0

Core 1

Core 2

Core 3

L1 $

L2 $

Private caches

Interconnect

Banked /Sliced LLC

LLC B0

LLC B1

LLC B2

LLC Bn

16MB

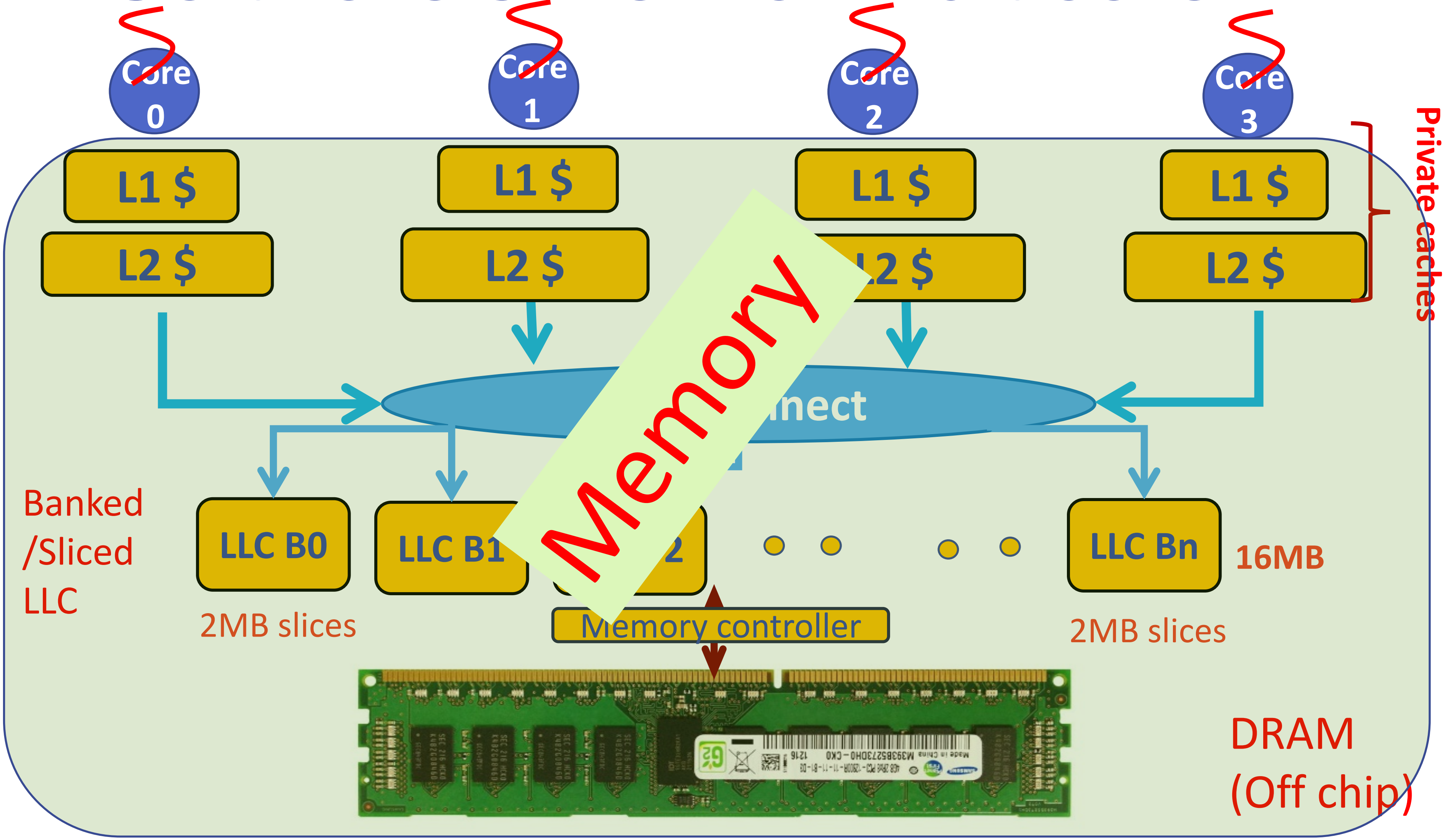
2MB slices

Memory controller

2MB slices

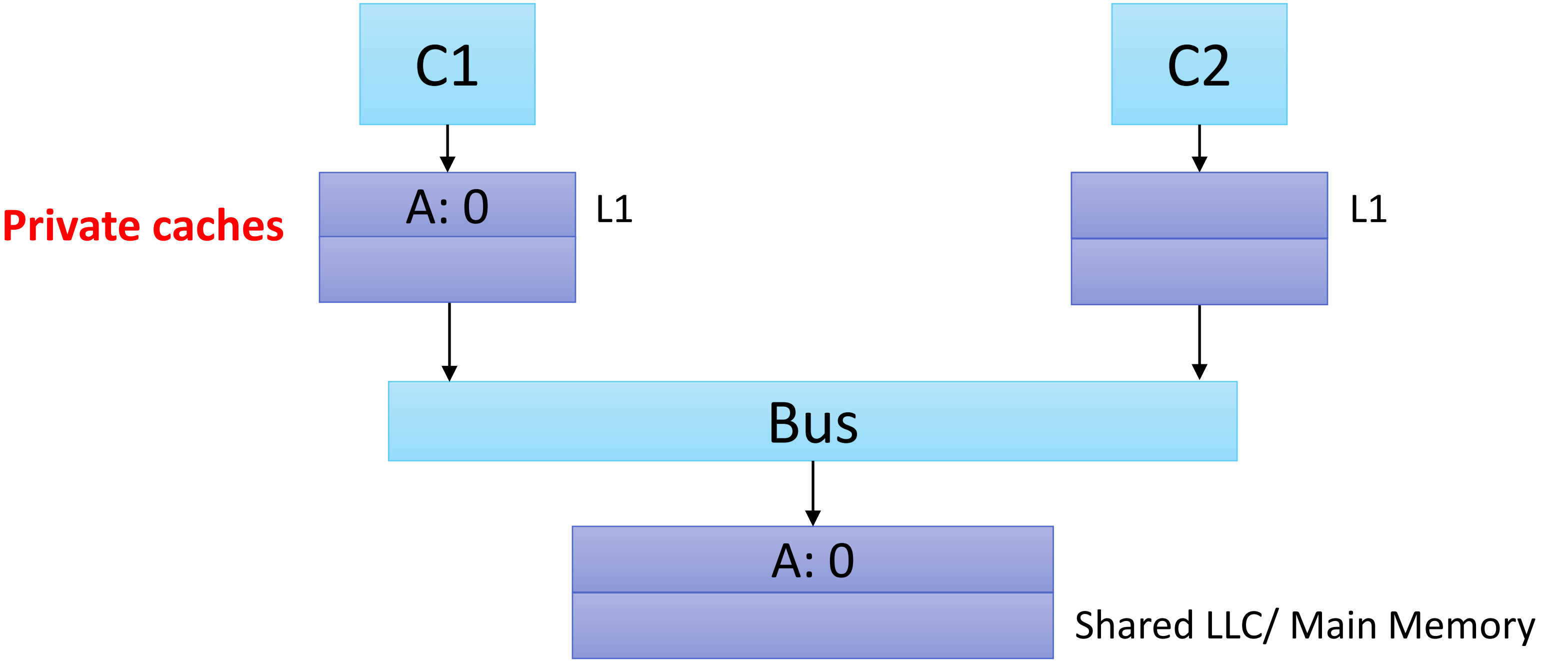
DRAM (Off chip)

# Software's view of multicore



# Cache In-coherence problem

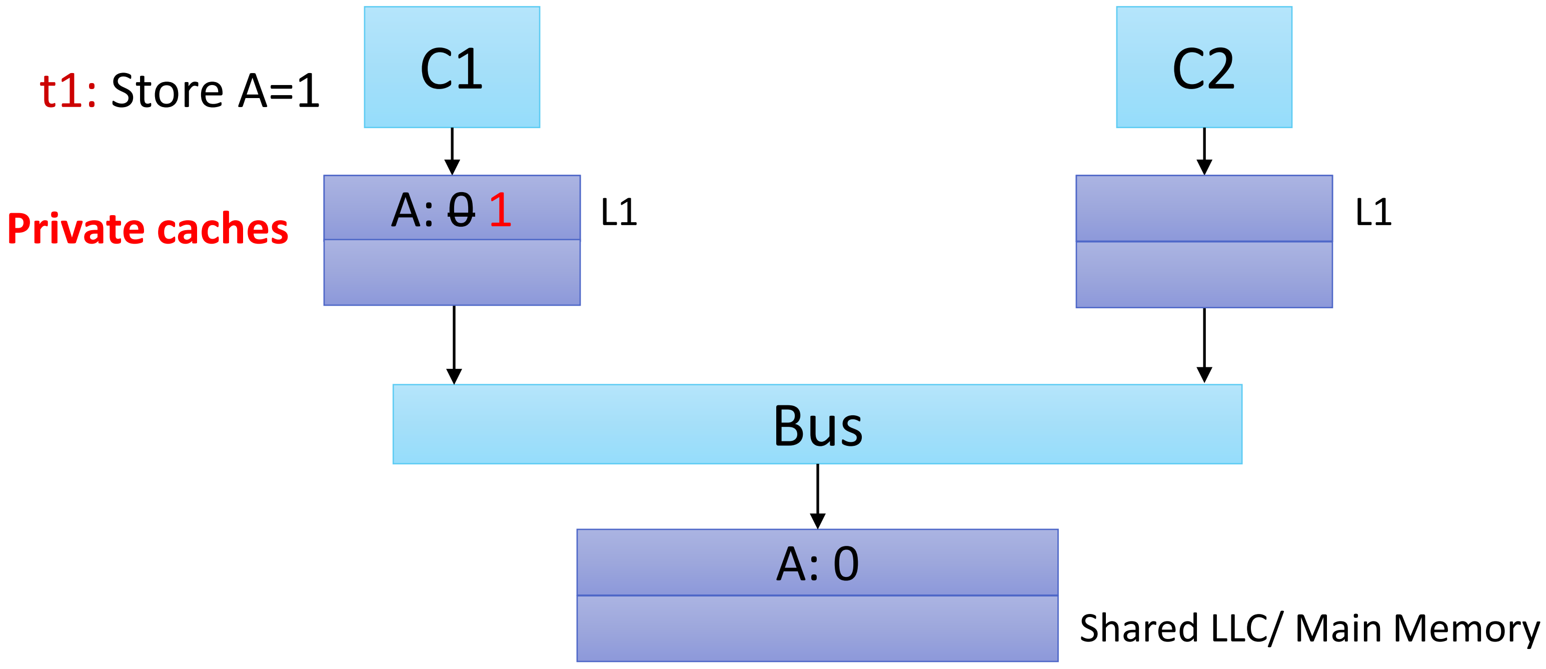
- Variable A initially has value 0
- C1 stores value 1 into A
- C2 loads A from memory and sees old value 0



Simplified view of a multicore

# Cache In-coherence problem

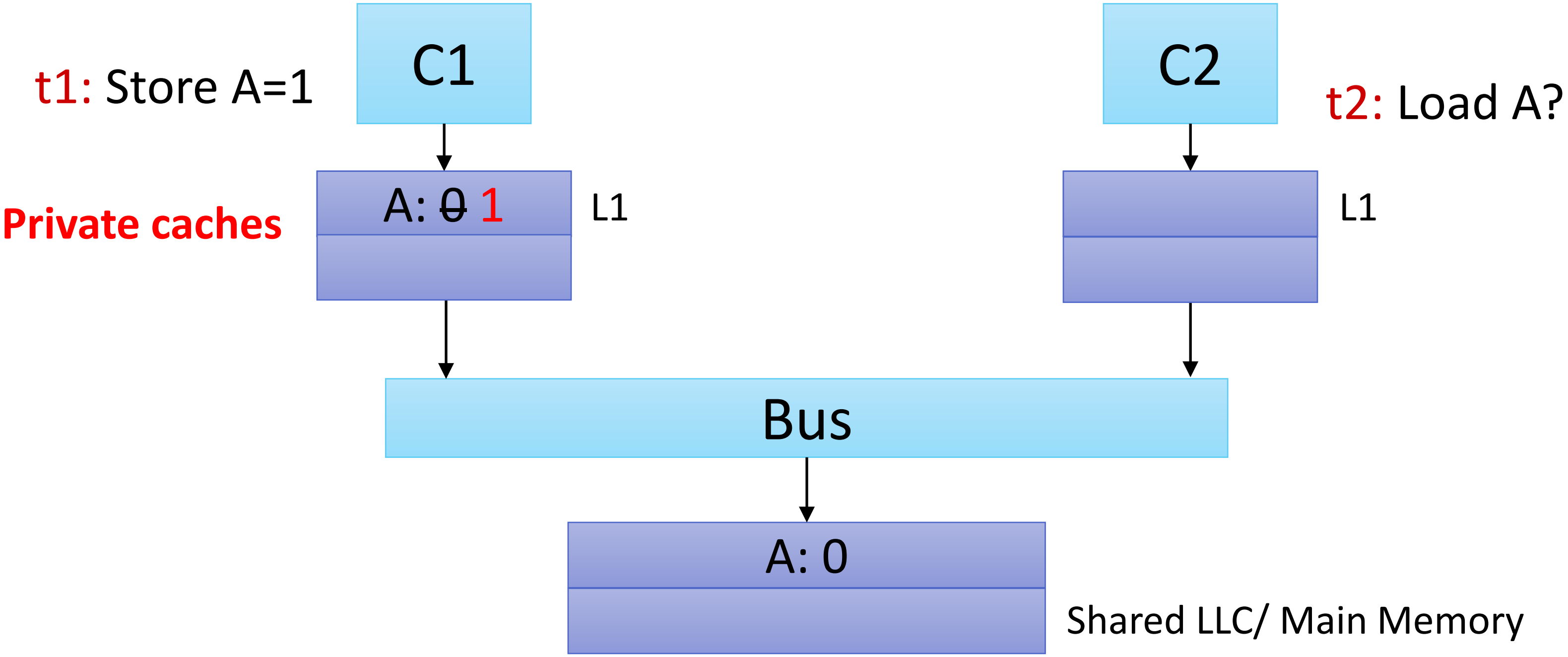
- Variable A initially has value 0
- C1 stores value 1 into A
- C2 loads A from memory and sees old value 0



Simplified view of a multicore

# Cache In-coherence problem

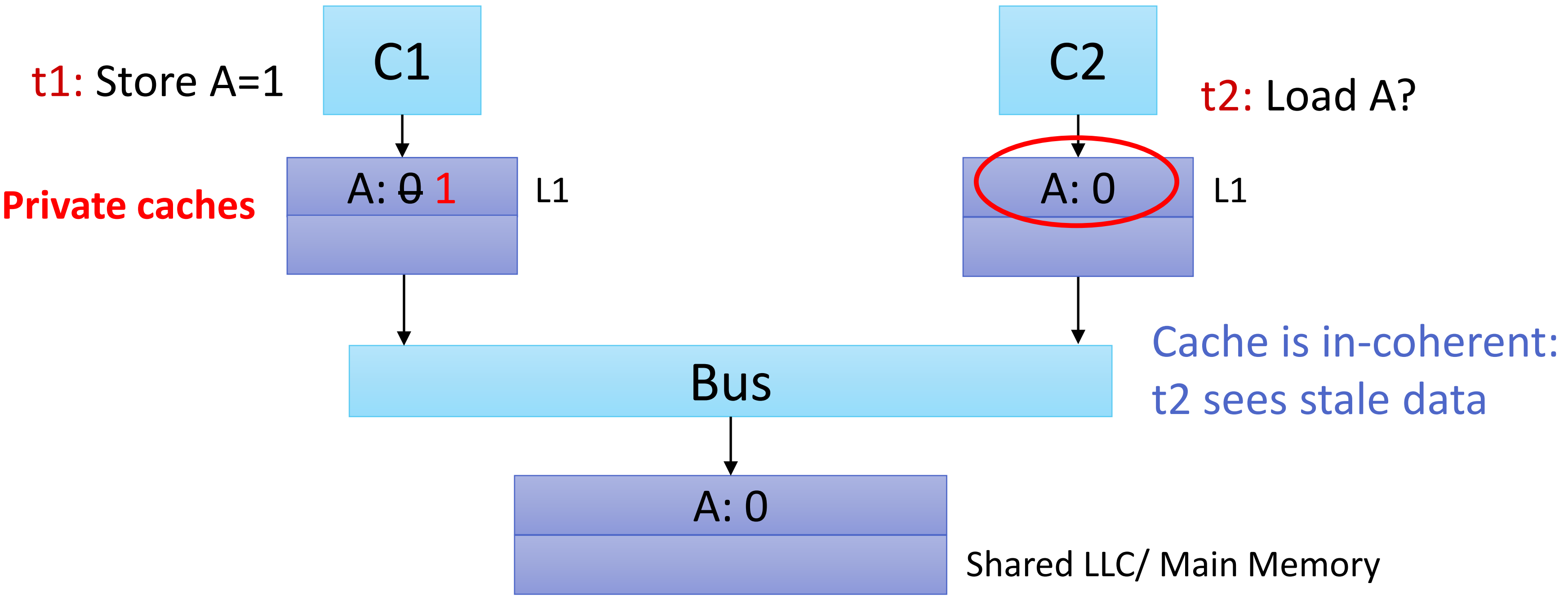
- Variable A initially has value 0
- C1 stores value 1 into A
- C2 loads A from memory and sees old value 0



Simplified view of a multicore

# Cache In-coherence problem

- Variable A initially has value 0
- C1 stores value 1 into A
- C2 loads A from memory and sees old value 0



t1: Store A=1

C1

A: ~~0~~ 1

L1

Private caches

C2

t2: Load A?

A: 0

L1

Cache is in-coherent: t2 sees stale data

Bus

A: 0

Shared LLC/ Main Memory

Simplified view of a multicore

# Goal of Cache coherence

- Keep contents of private caches coherent
  - Software should get the “latest” data

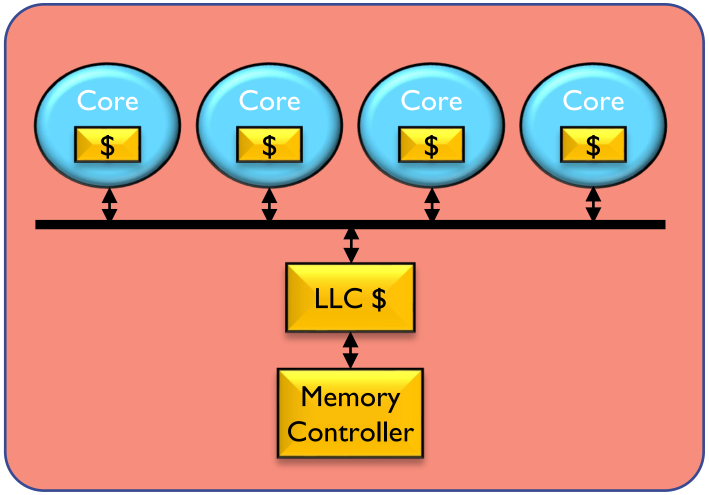
- Cache coherence subsystem orders (total order) all writes to a given memory address/location
  - It does it for all memory address/location

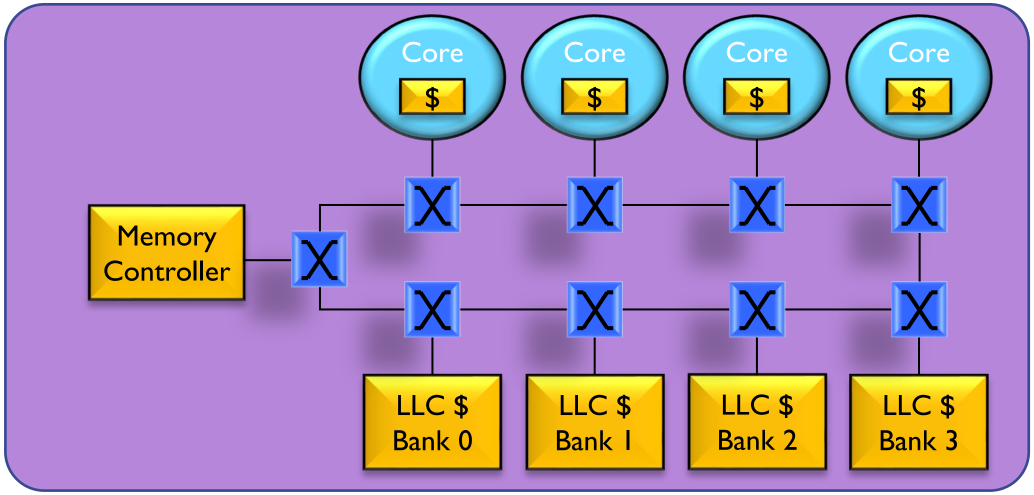
# Goal of Cache coherence

- Keep contents of private caches coherent
  - Software should get the “latest” data
- Cache coherence subsystem orders (total order) all writes to a given memory address/location
  - It does it for all memory address/location
- Two design classes of cache coherence subsystems
  - Snoopy coherence protocol ←
  - Directory coherence protocol (if time permits)

# Snoopy cache coherence

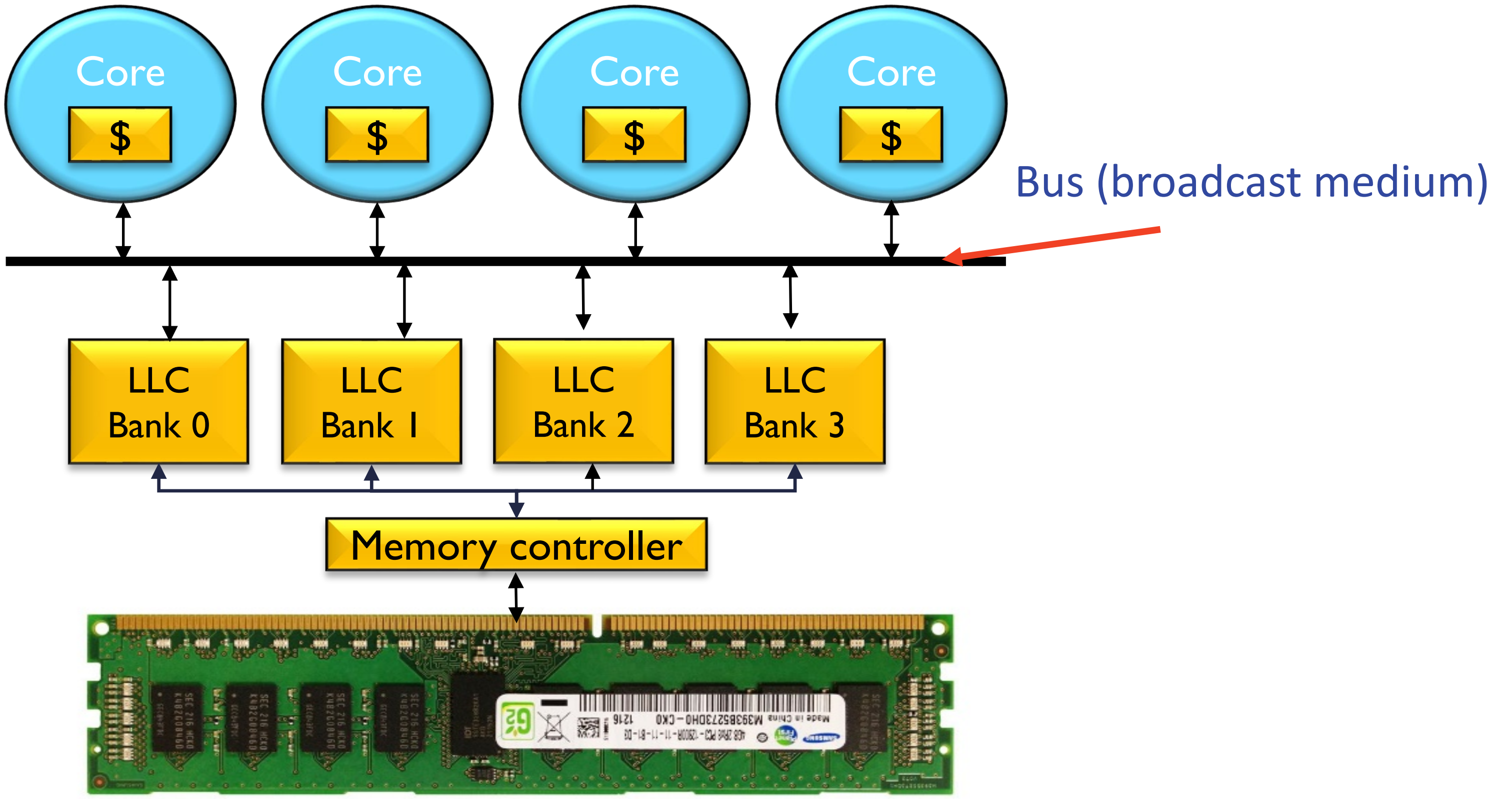
- Relies on broadcast-based interconnection network
  - Typically Bus or Ring





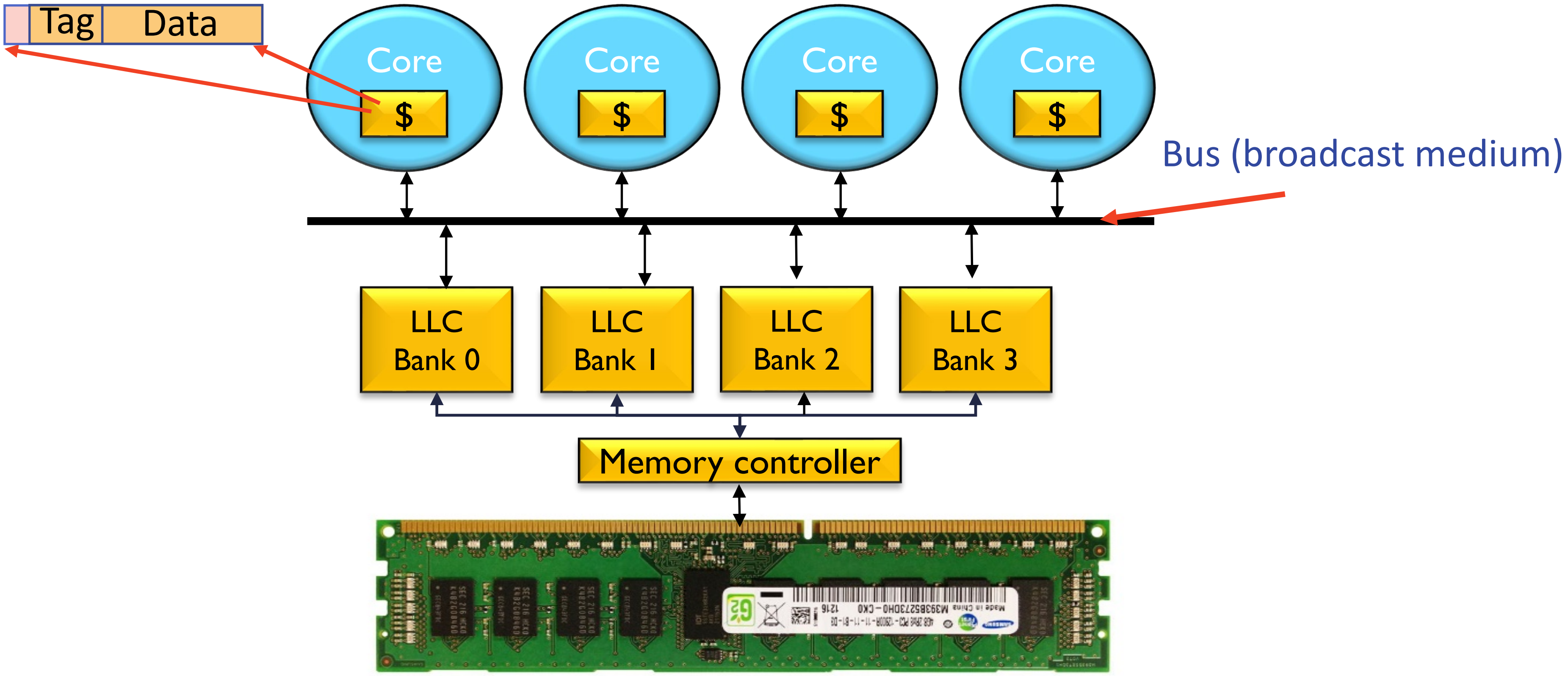
- All caches must monitor (aka "*snoop*") all traffic
  - And keep track of cache line *states* based on the observed traffic

# Snoopy cache coherence



Example: MSI coherence protocol

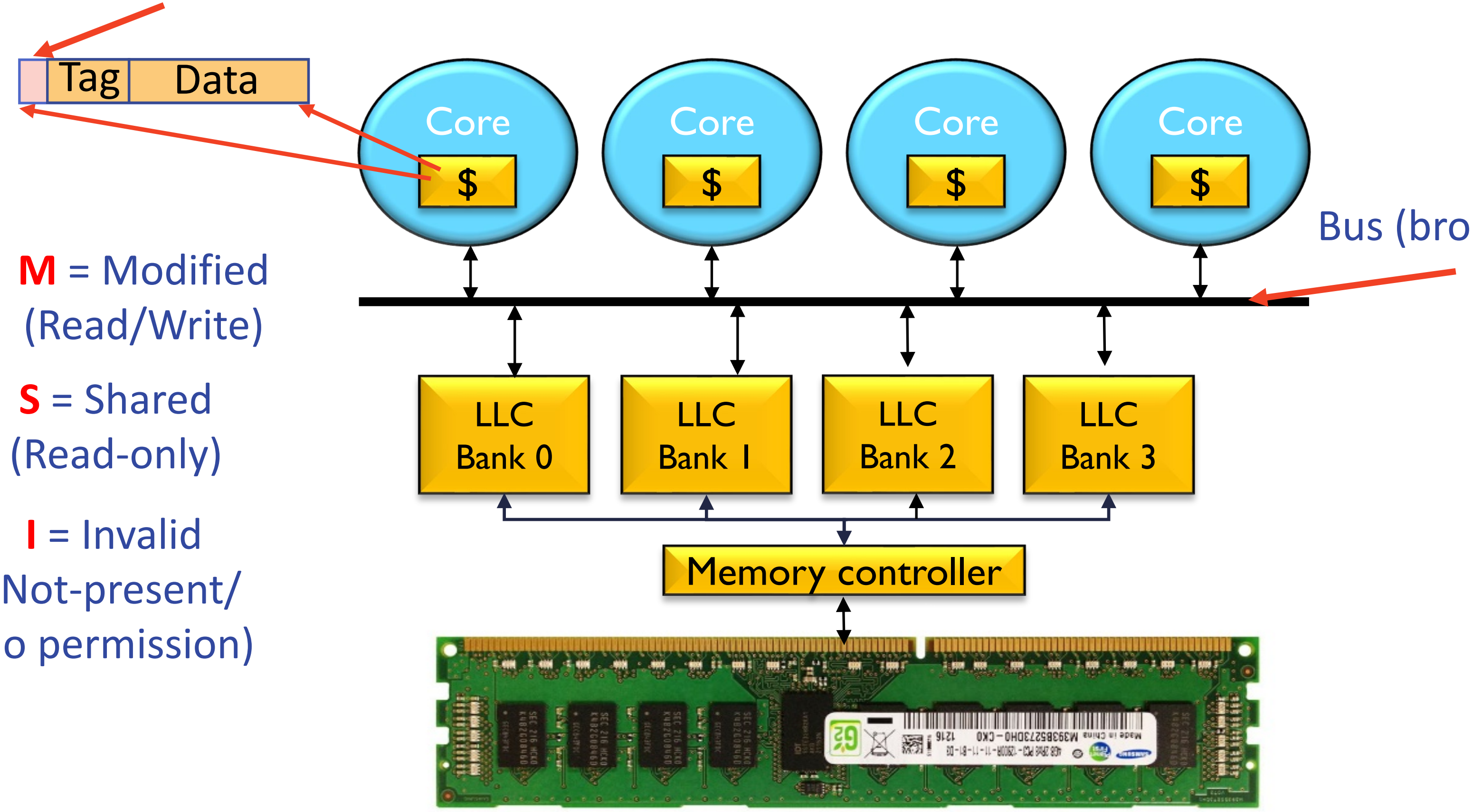
# Snoopy cache coherence



Example: MSI coherence protocol

# Snoopy cache coherence

Coherence state of the cache block



Bus (broadcast medium)

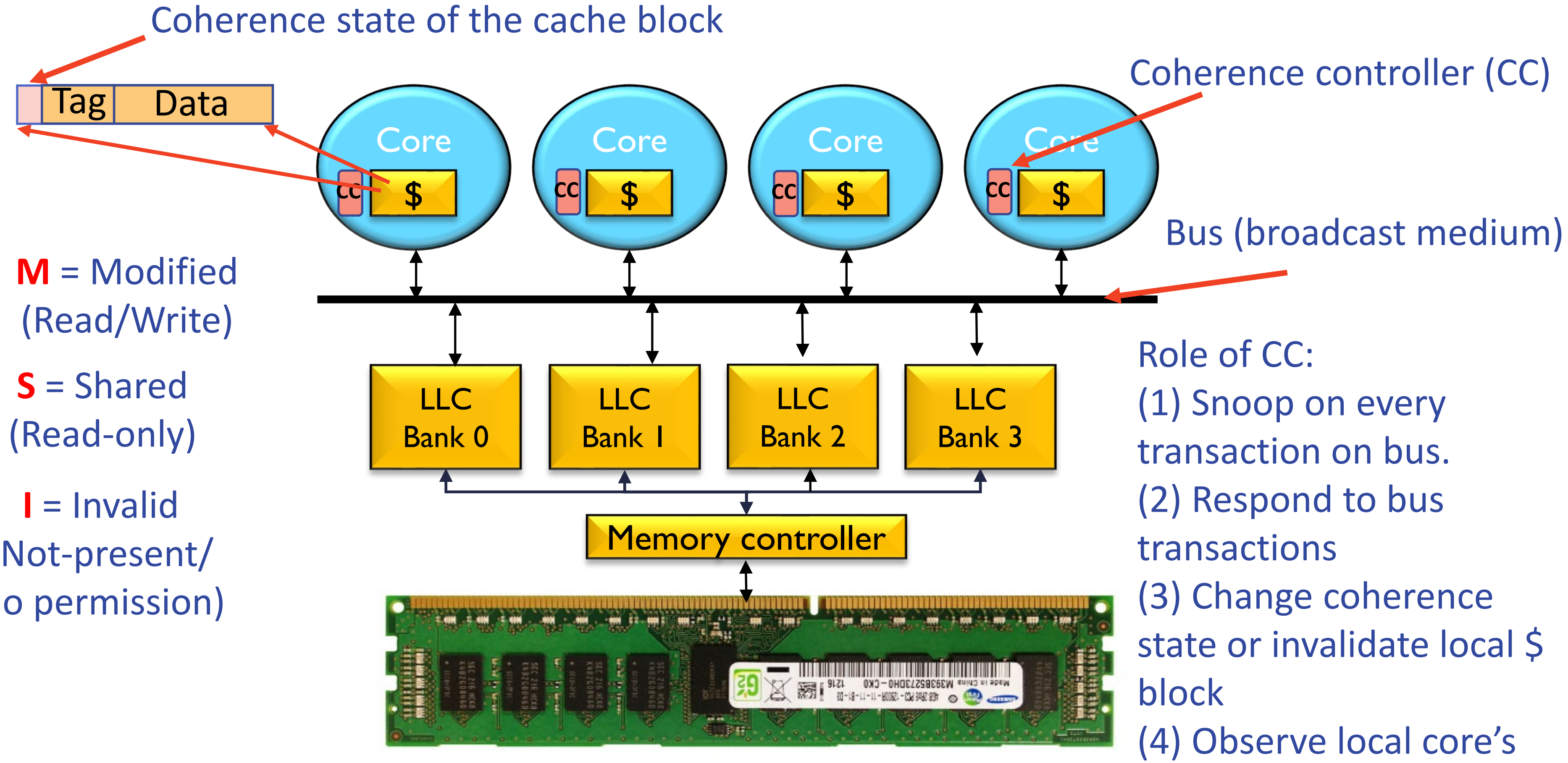
**M** = Modified (Read/Write)

**S** = Shared (Read-only)

**I** = Invalid (Not-present/ No permission)

Example: MSI coherence protocol

# Snoopy cache coherence



**M** = Modified (Read/Write)

**S** = Shared (Read-only)

**I** = Invalid (Not-present/ No permission)

Role of CC:

(1) Snoop on every transaction on bus.

(2) Respond to bus transactions

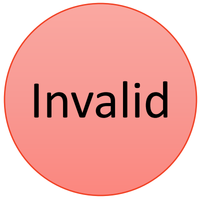
(3) Change coherence state or invalidate local $ block

(4) Observe local core's ld/st and (possibly) initiate bus transactions

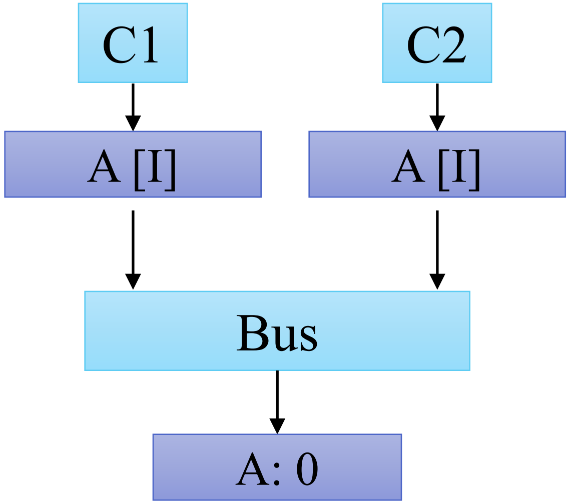
[Implements an FSM]

Example: MSI coherence protocol

# Ex: MSI protocol in action (1)

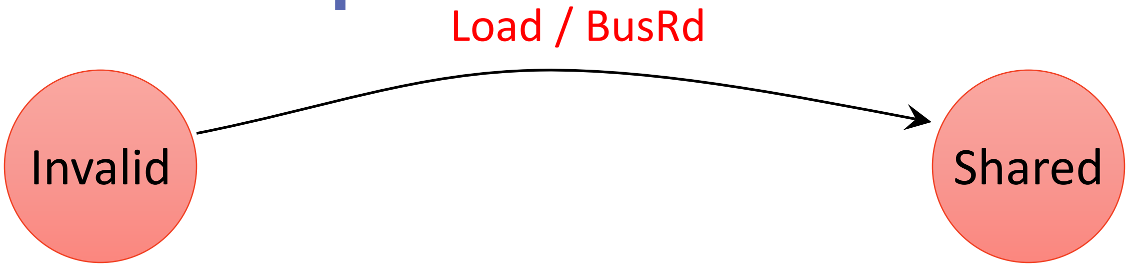


→ **Transition caused by local action**

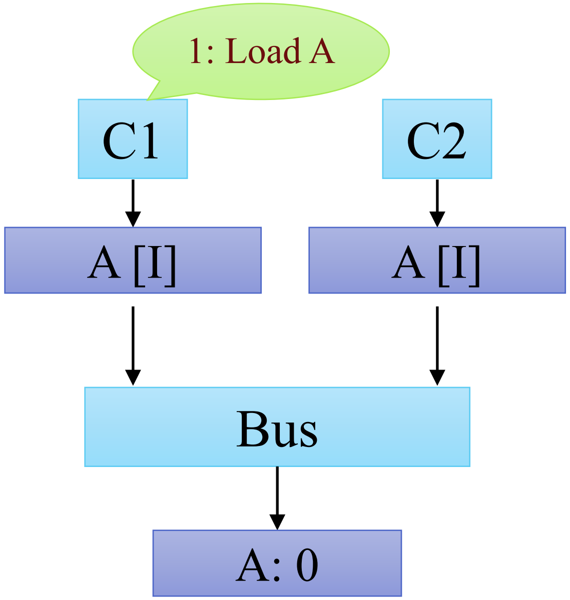


Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (1)

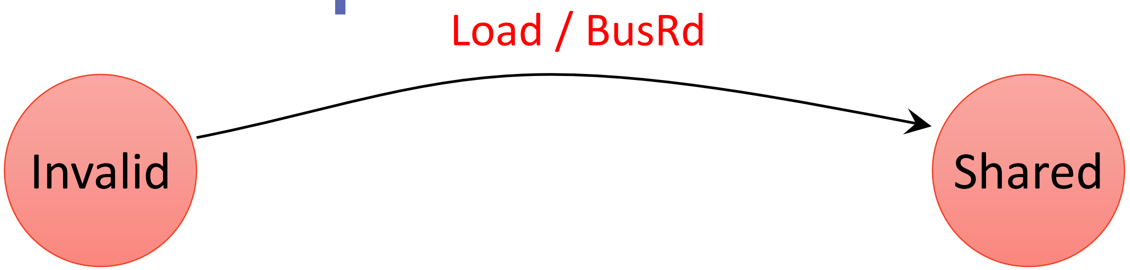


→ **Transition caused by local action**

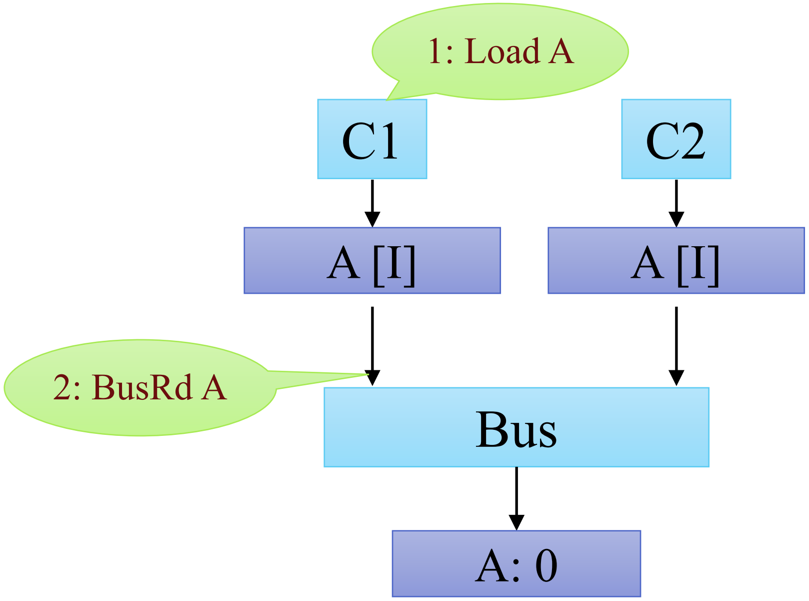


Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (1)

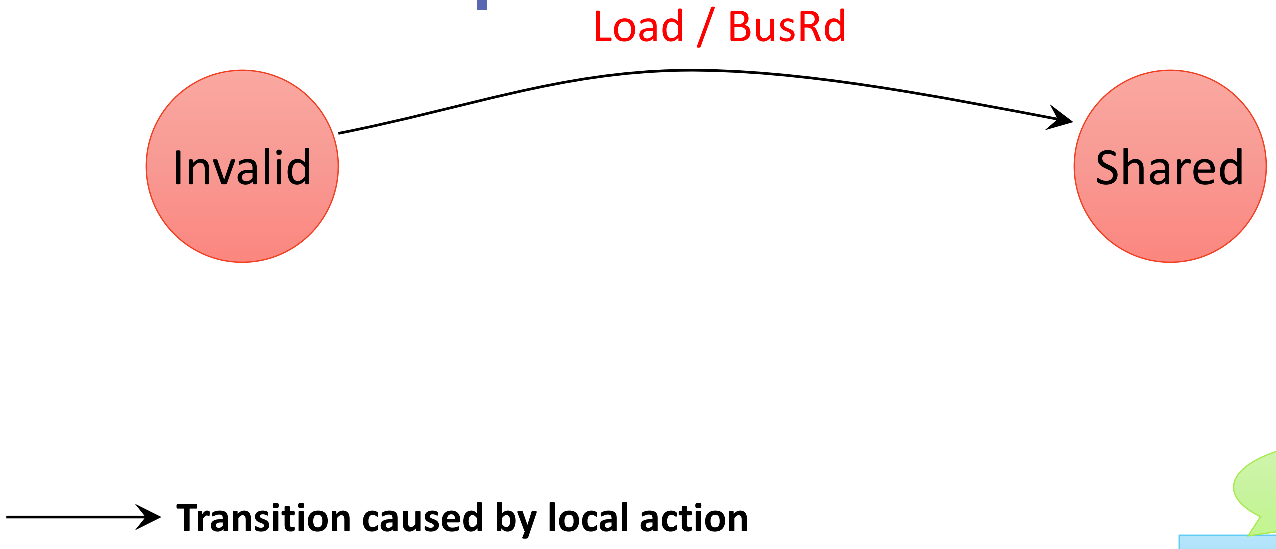


⟶ **Transition caused by local action**



Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (1)

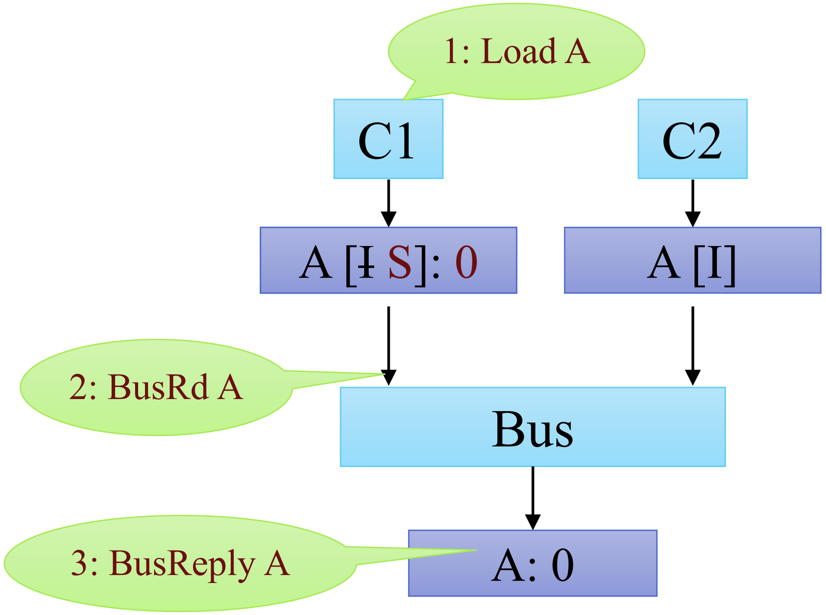


Load / BusRd

Invalid

Shared

Transition caused by local action



1: Load A

C1

C2

A [~~I~~ S]: 0

A [I]

2: BusRd A

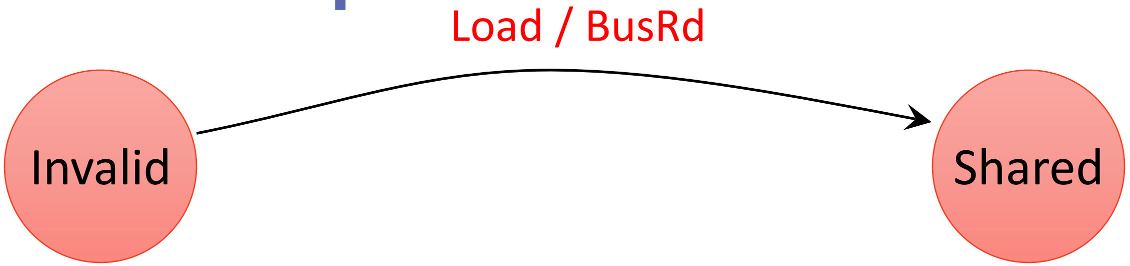
Bus

3: BusReply A

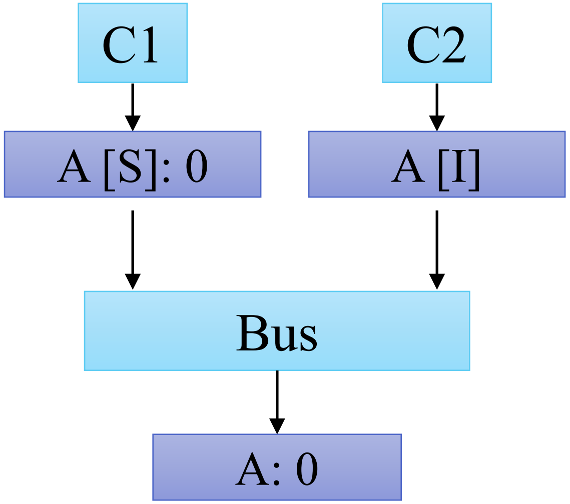
A: 0

Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (2)

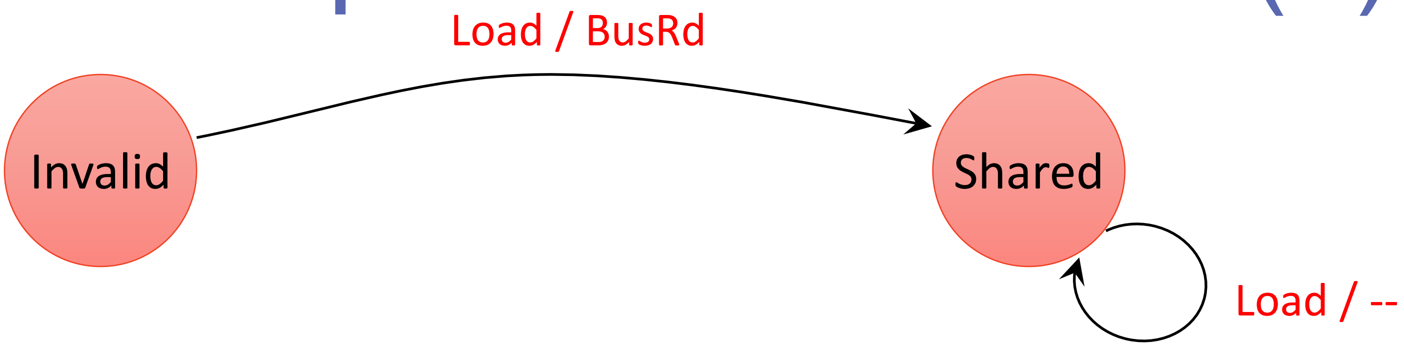


Transition caused by local action
Transition caused by bus message



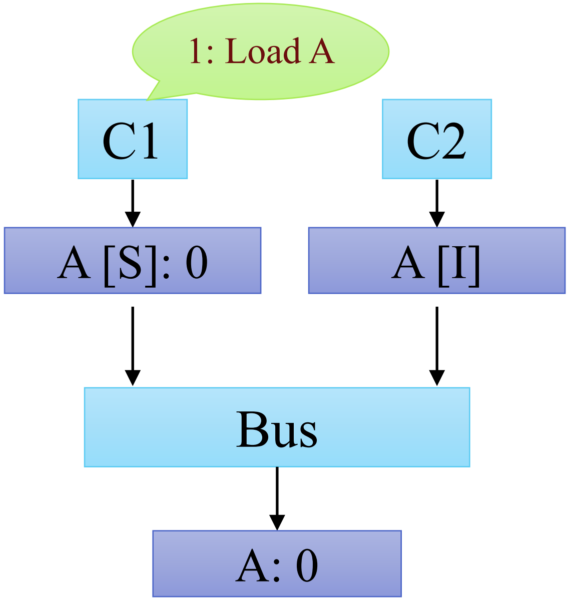
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (2)



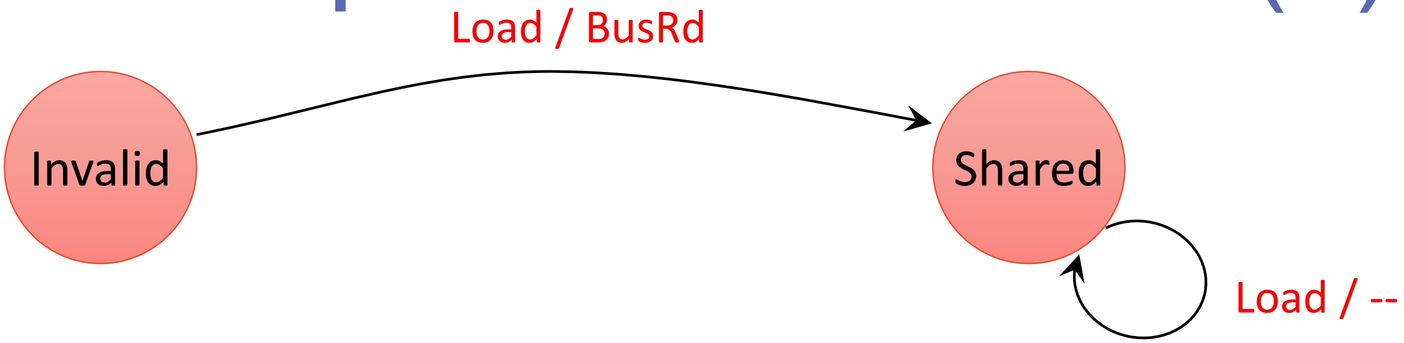
→ **Transition caused by local action**

- - - - - - -→ **Transition caused by bus message**



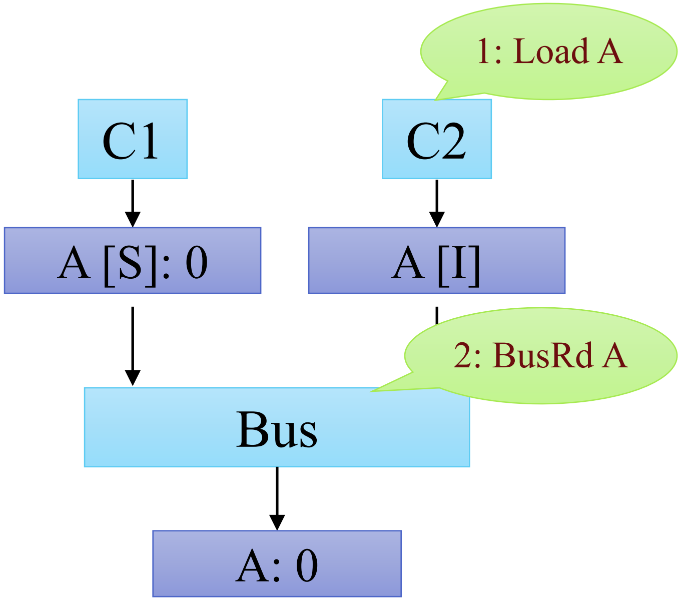
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (2)



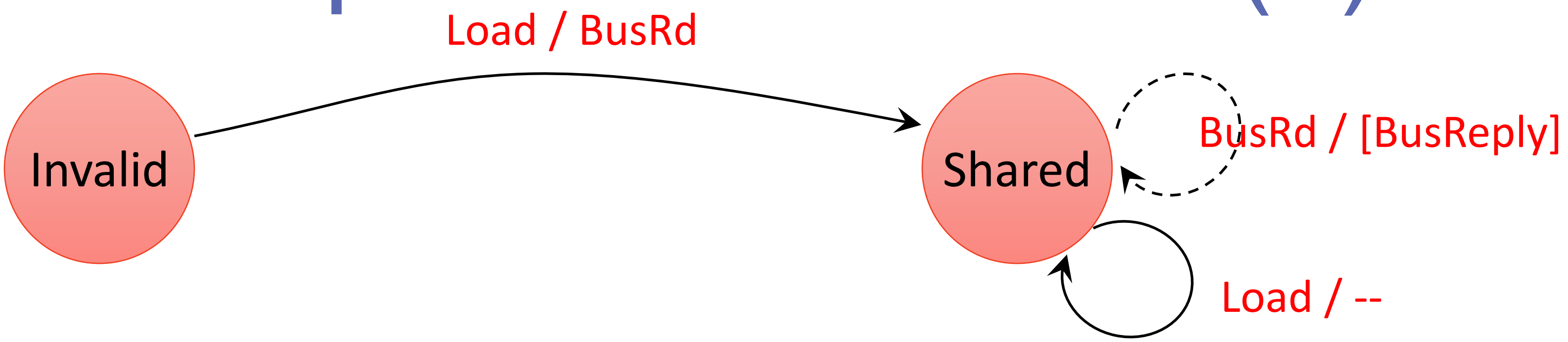
→ **Transition caused by local action**

- - - - - - -→ **Transition caused by bus message**



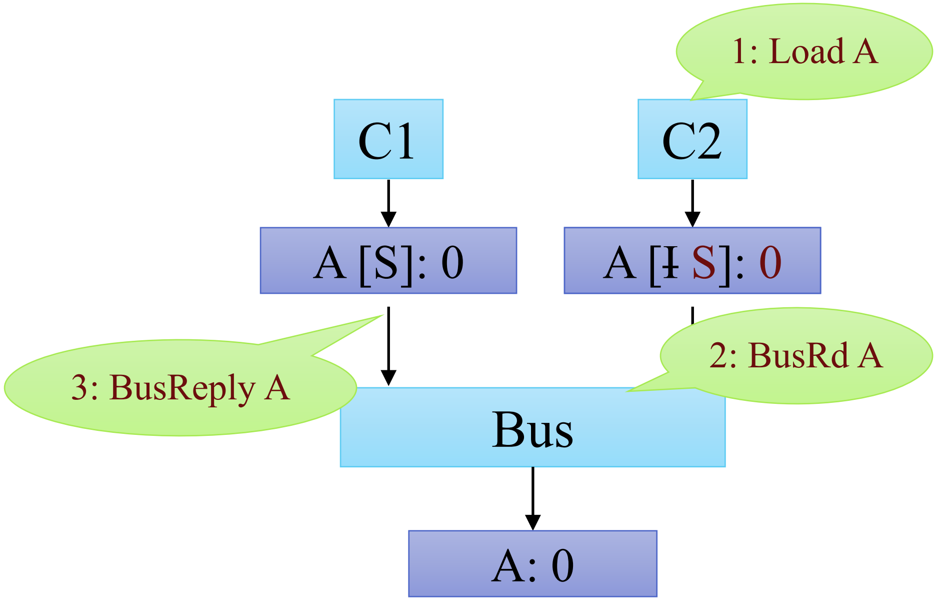
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (2)



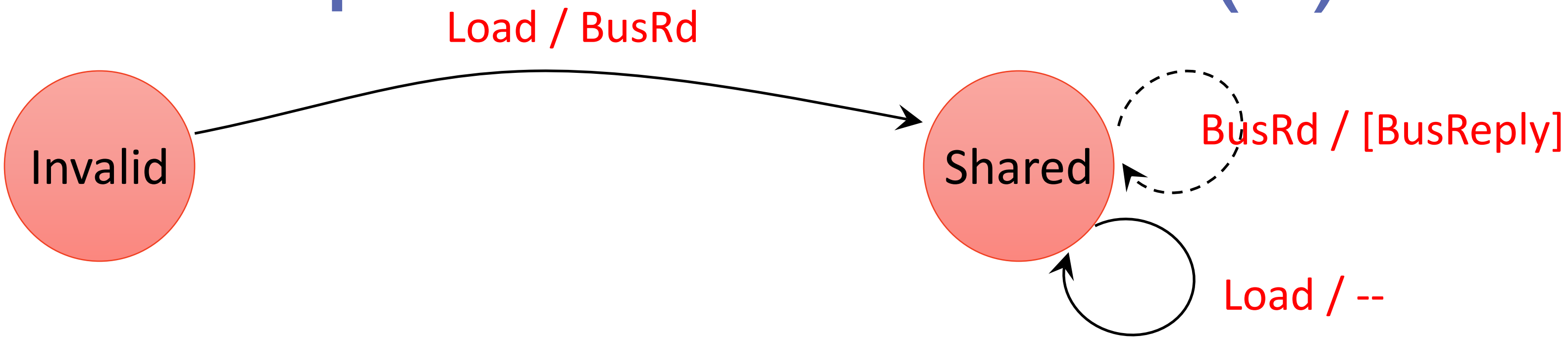
⟶ **Transition caused by local action**

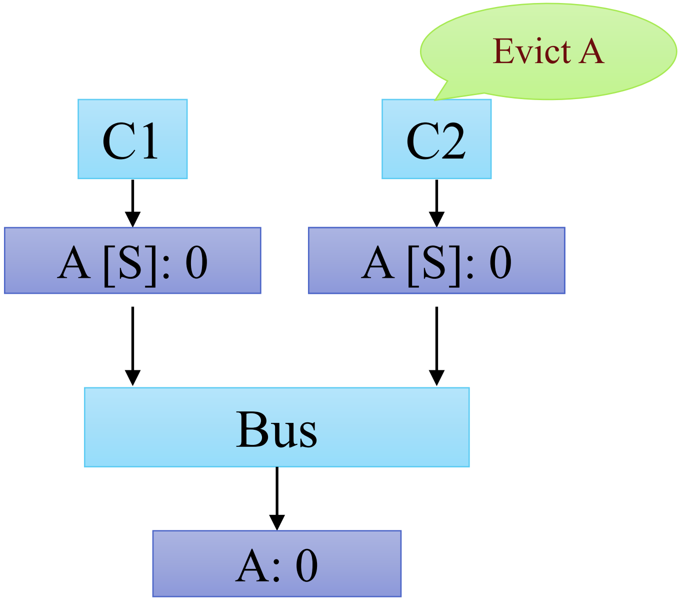
- - - - - - -➤ **Transition caused by bus message**



Invariant: Only one writer (M) per block; Many sharers (S) okay

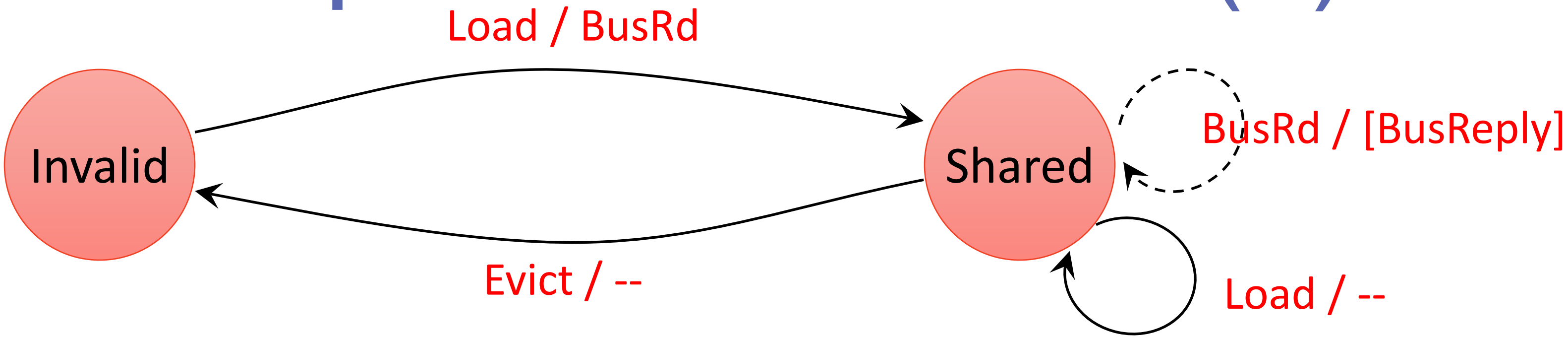
# Ex: MSI protocol in action(3)

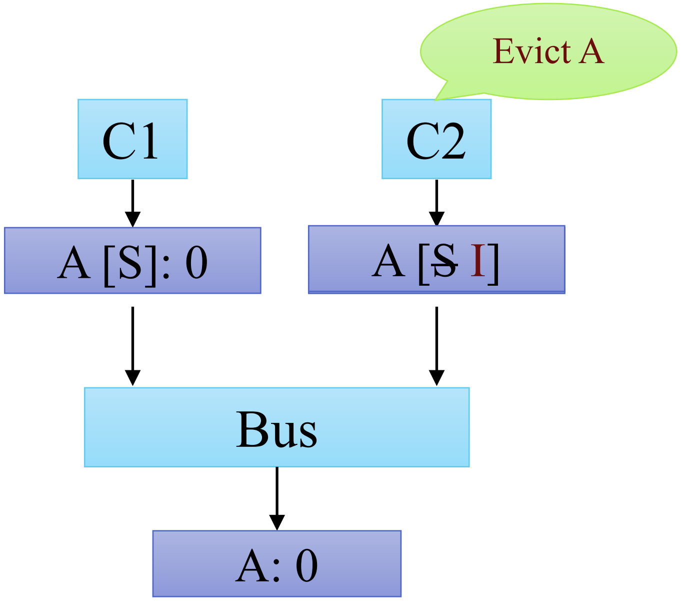




Invariant: Only one writer (M) per block; Many sharers (S) okay

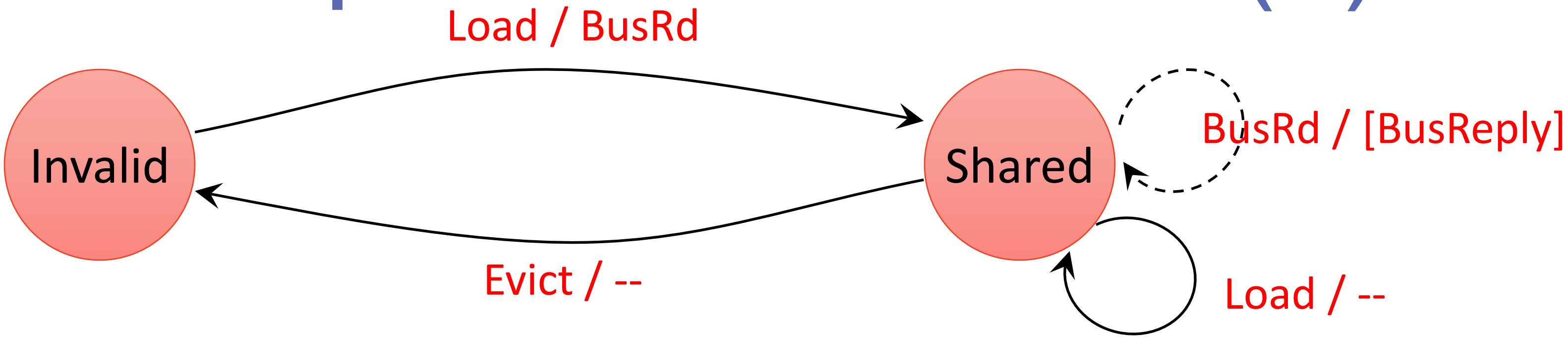
# Ex: MSI protocol in action(3)

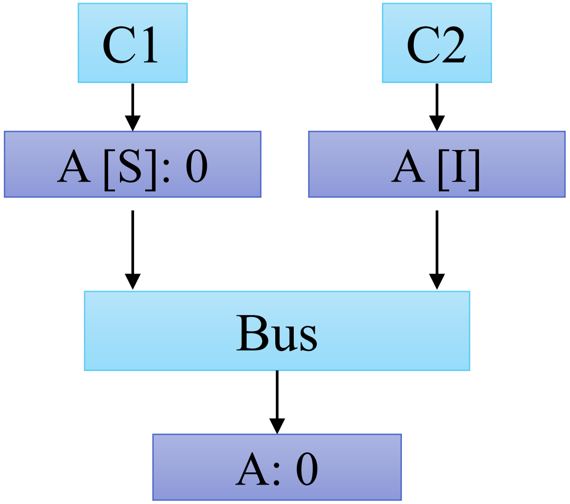




Invariant: Only one writer (M) per block; Many sharers (S) okay

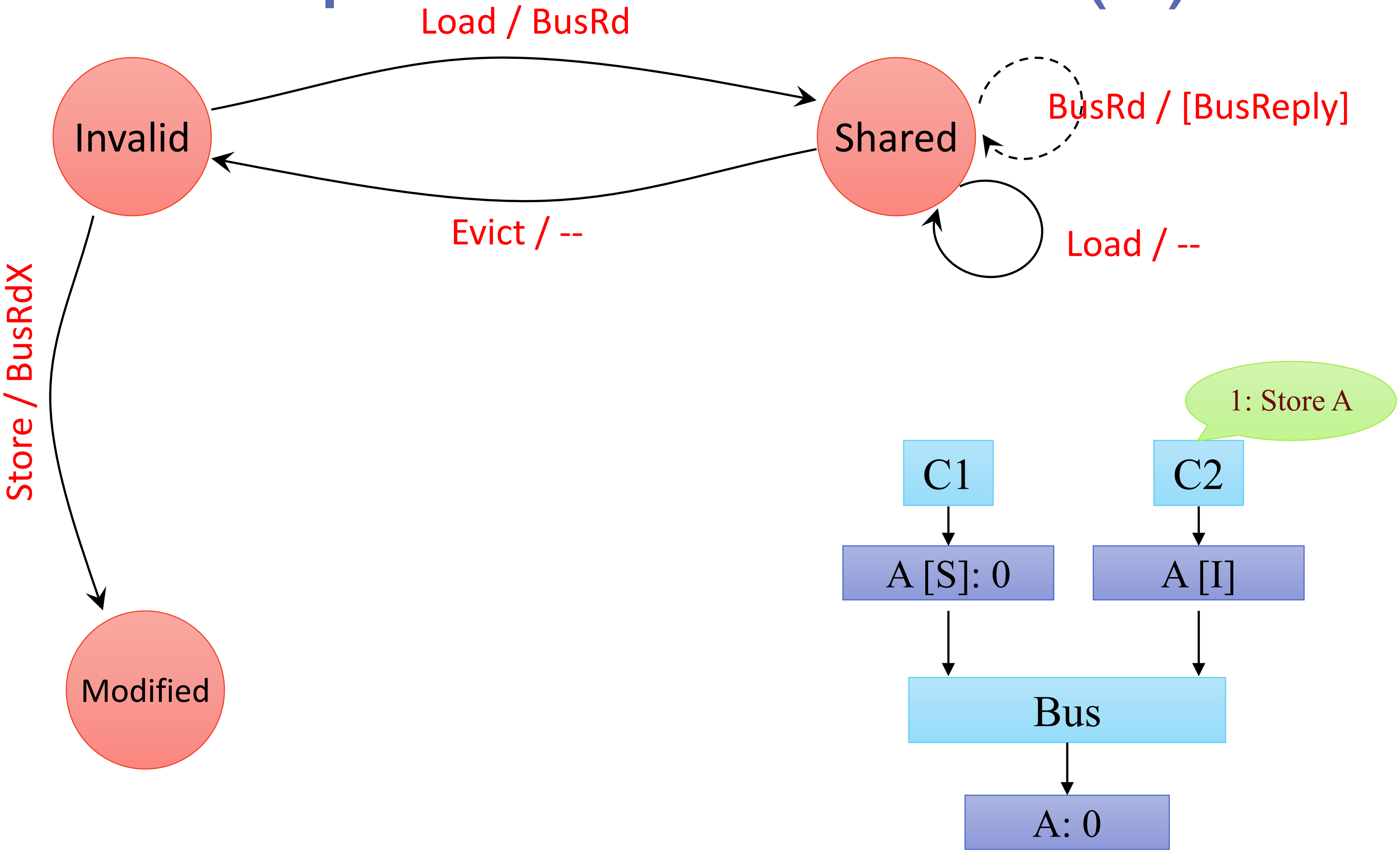
# Ex: MSI protocol in action (4)





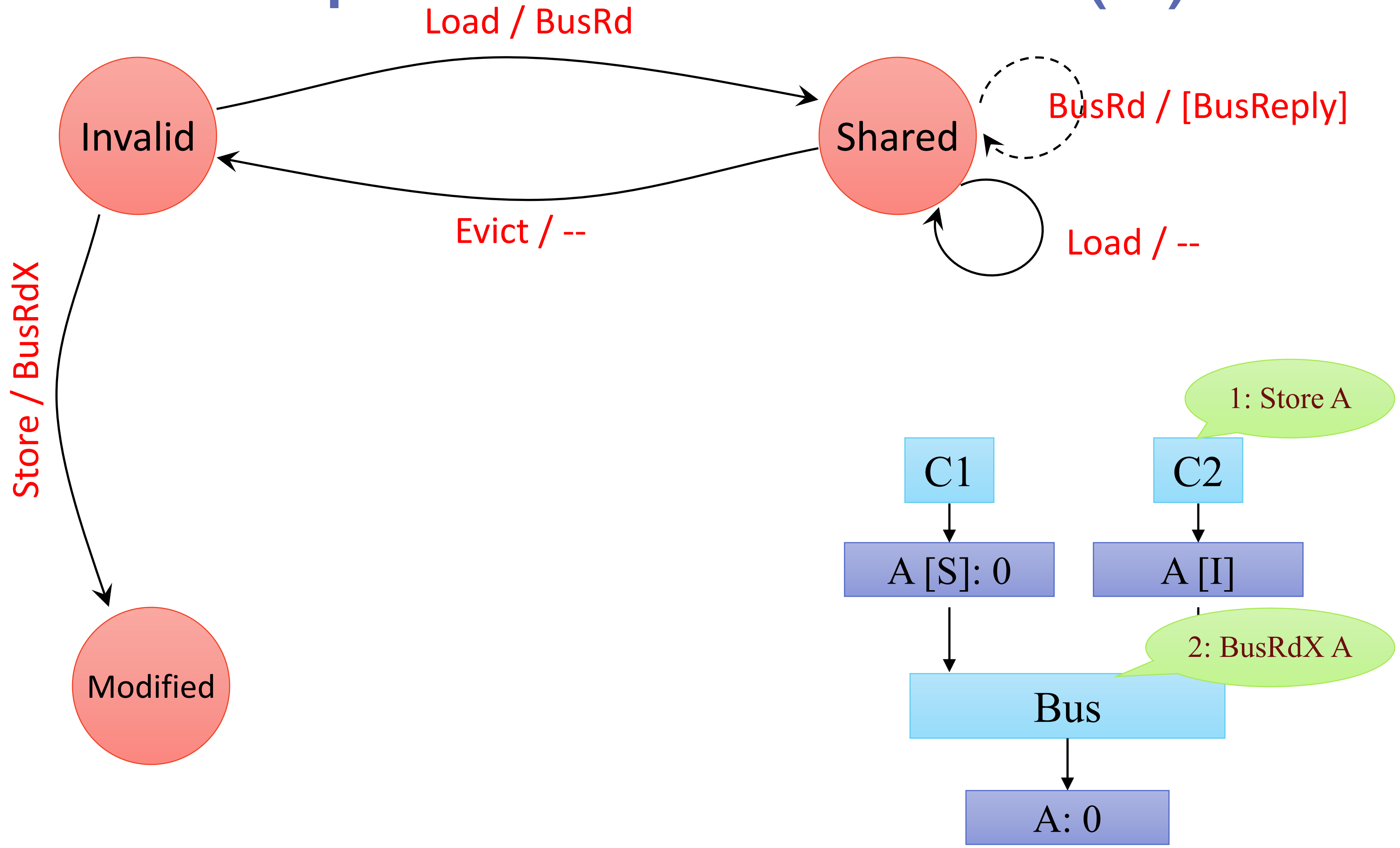
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (4)



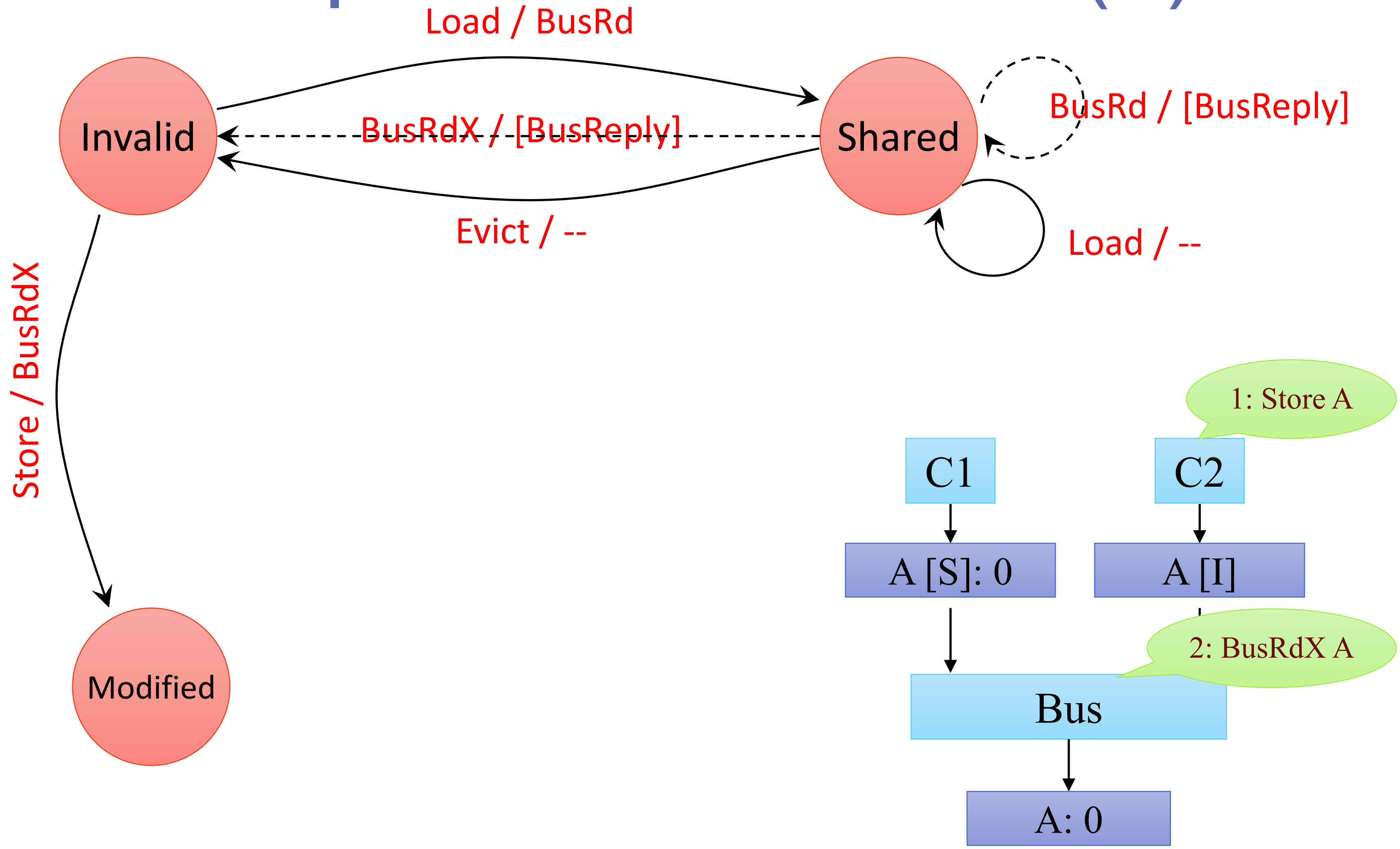
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (4)



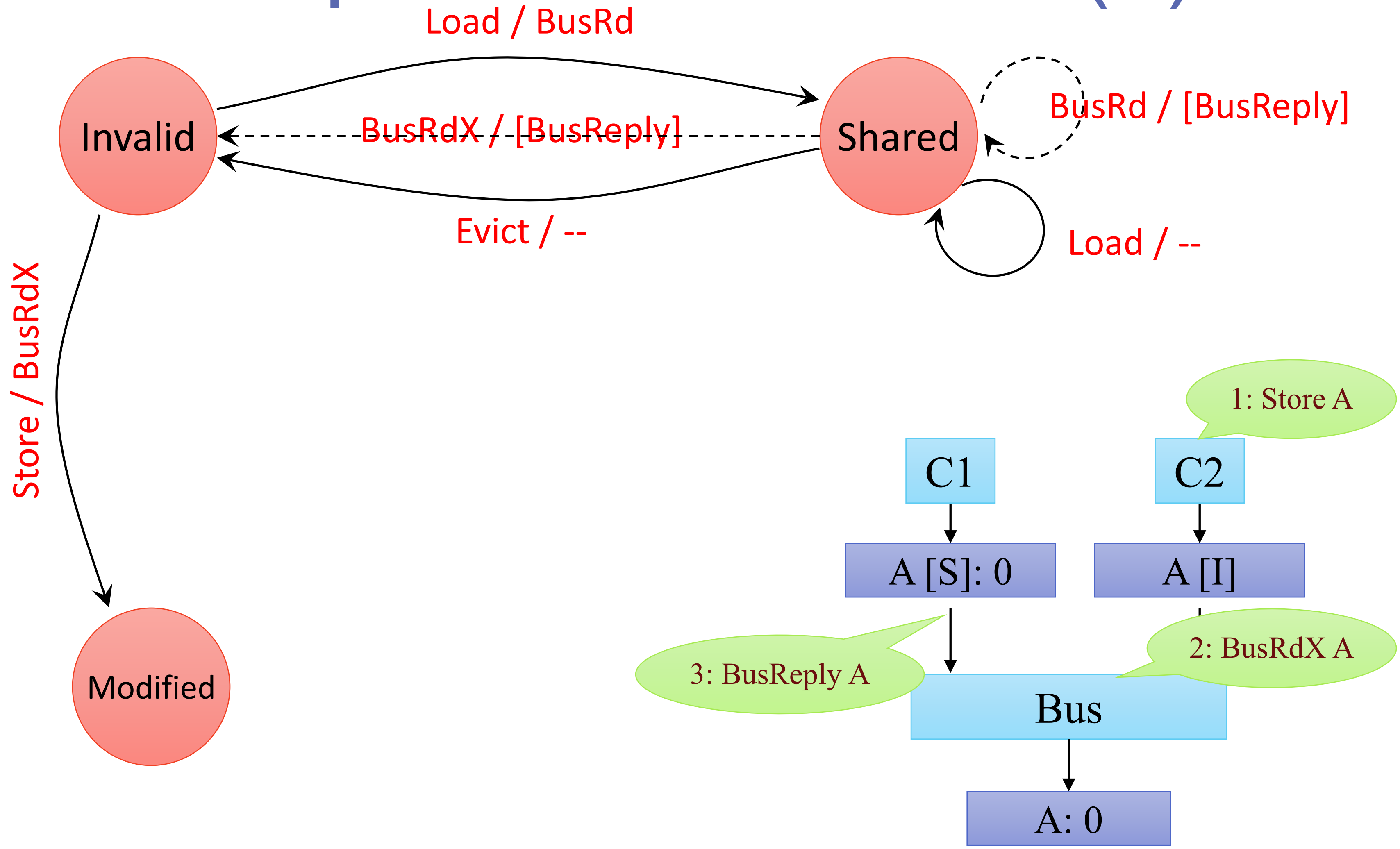
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (4)



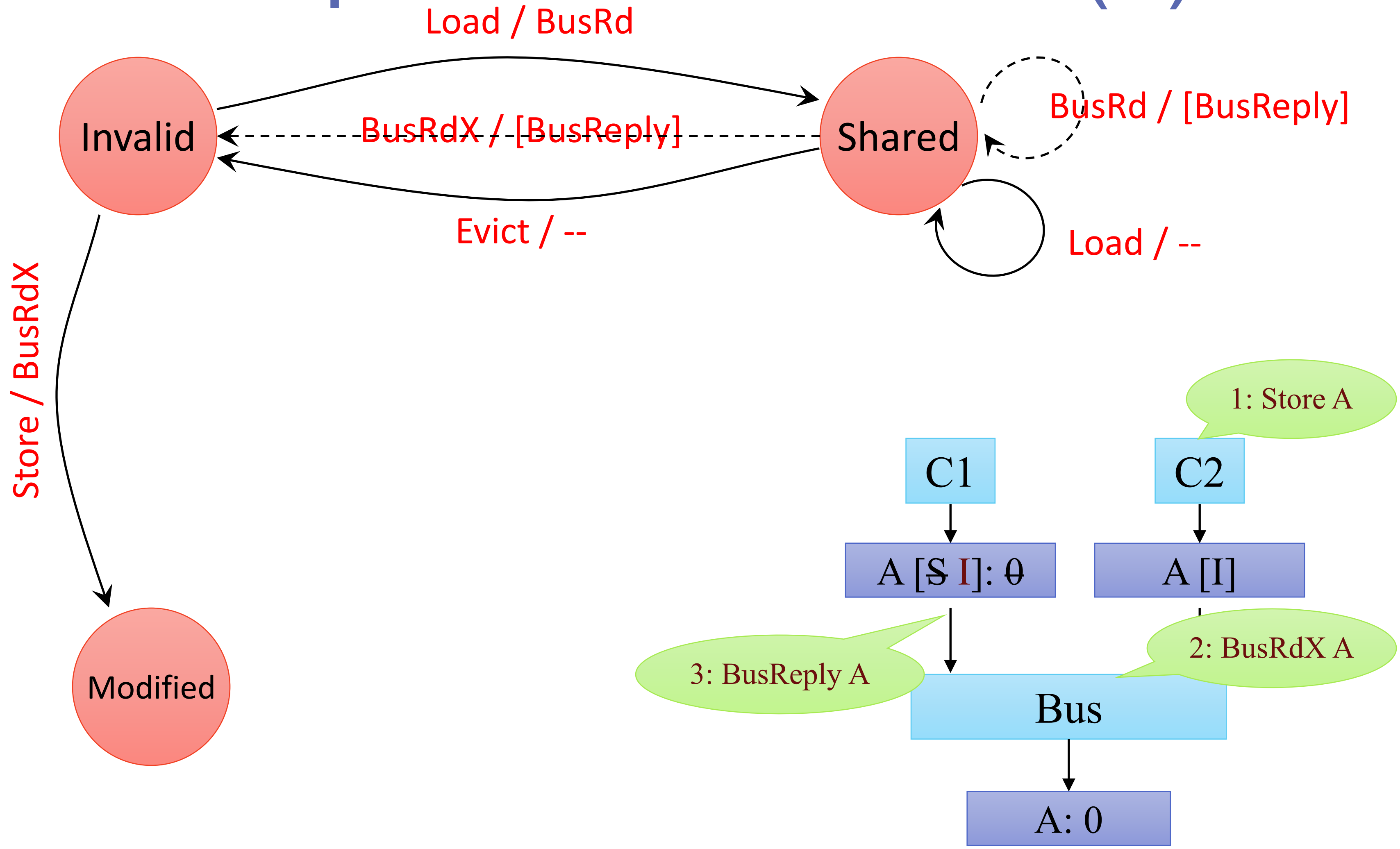
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (4)



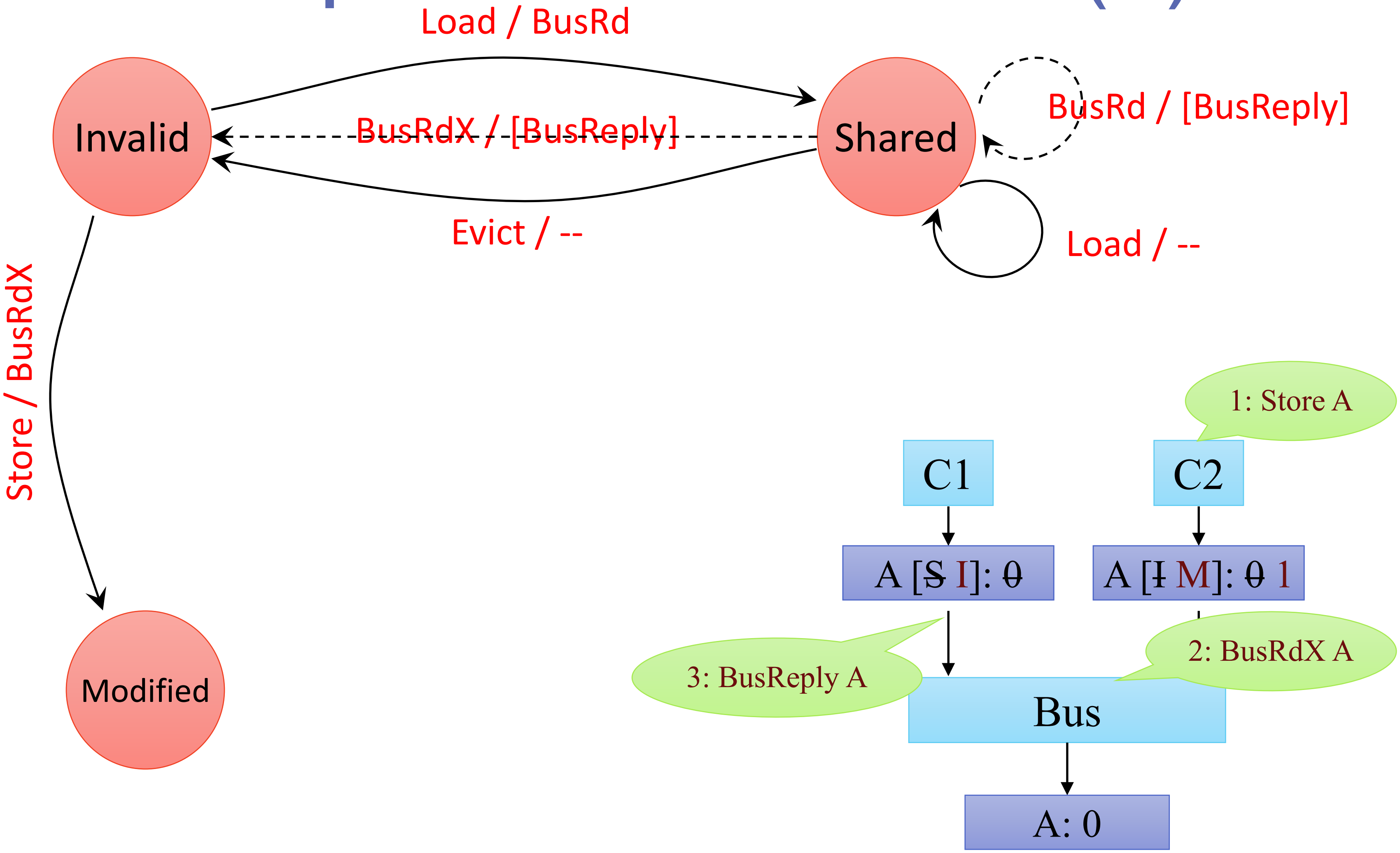
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (4)



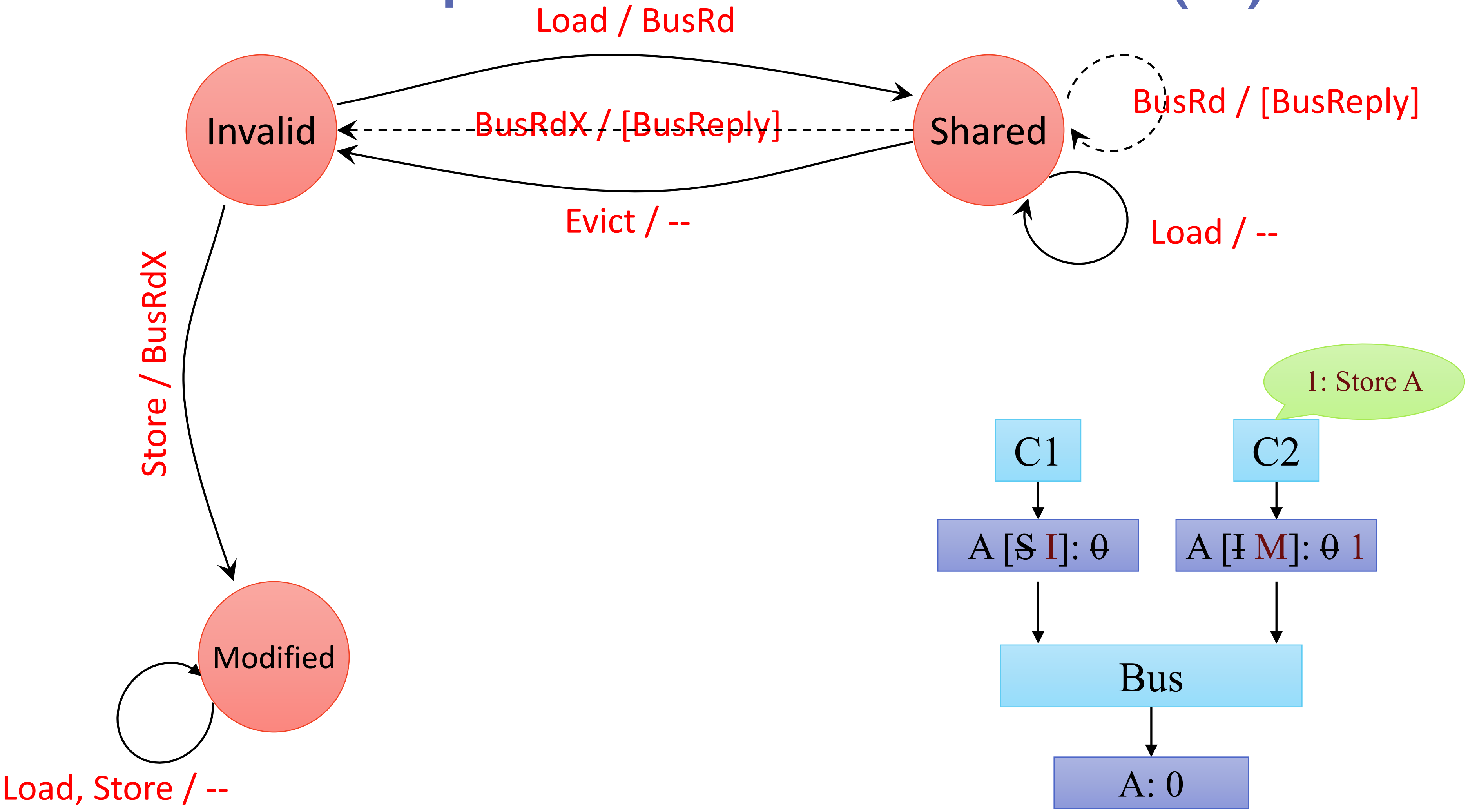
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (4)



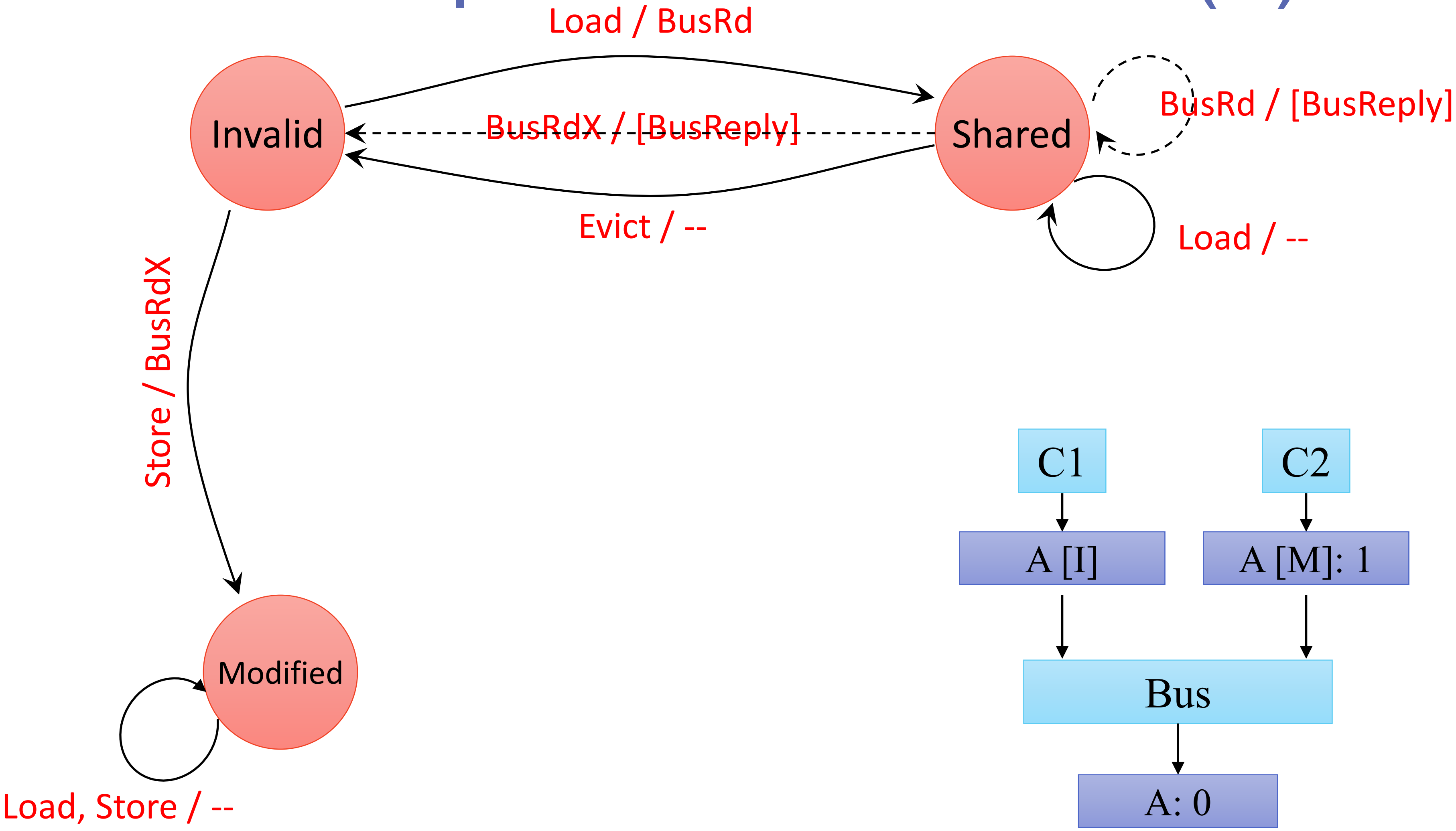
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (4)



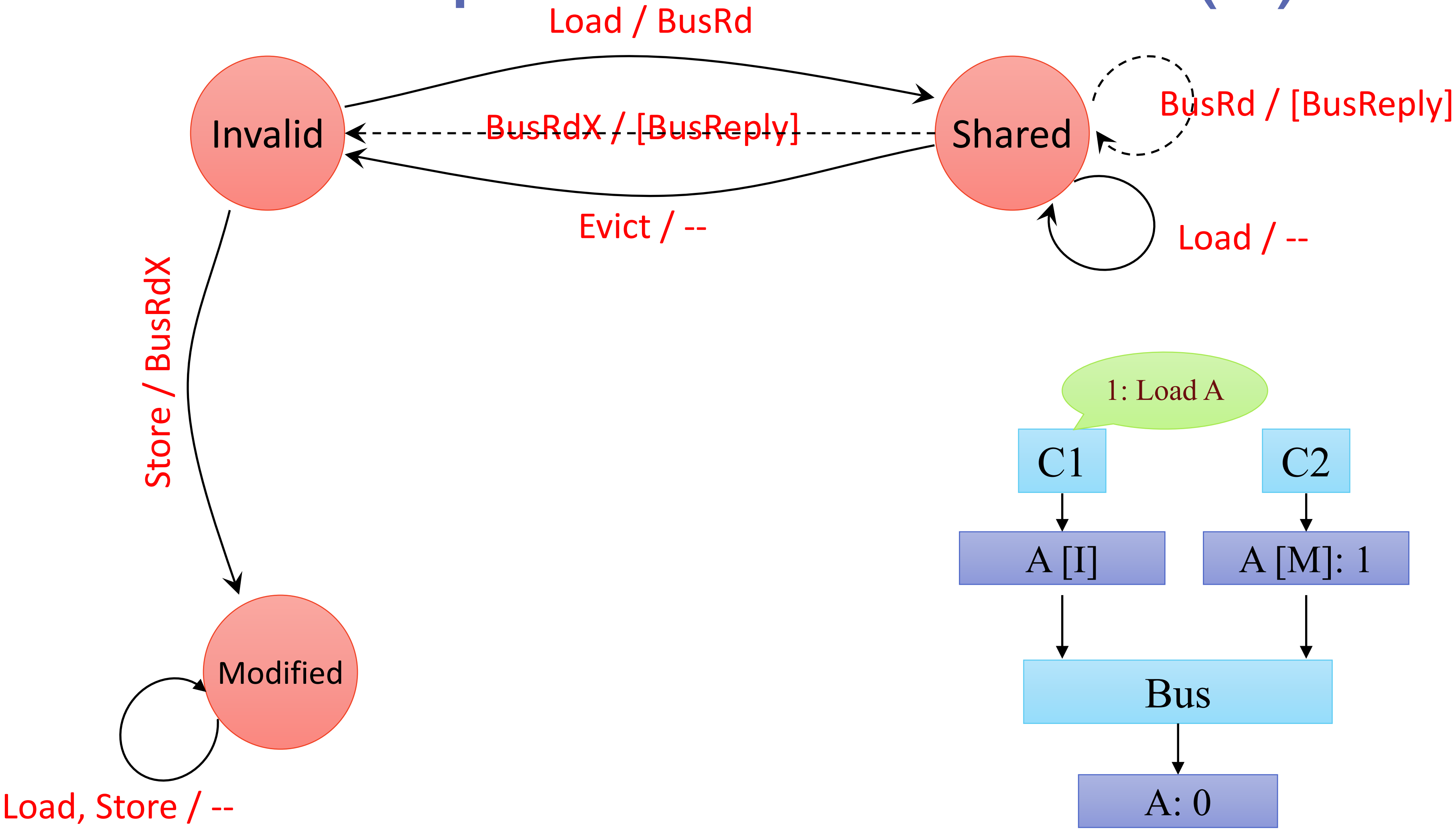
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (5)



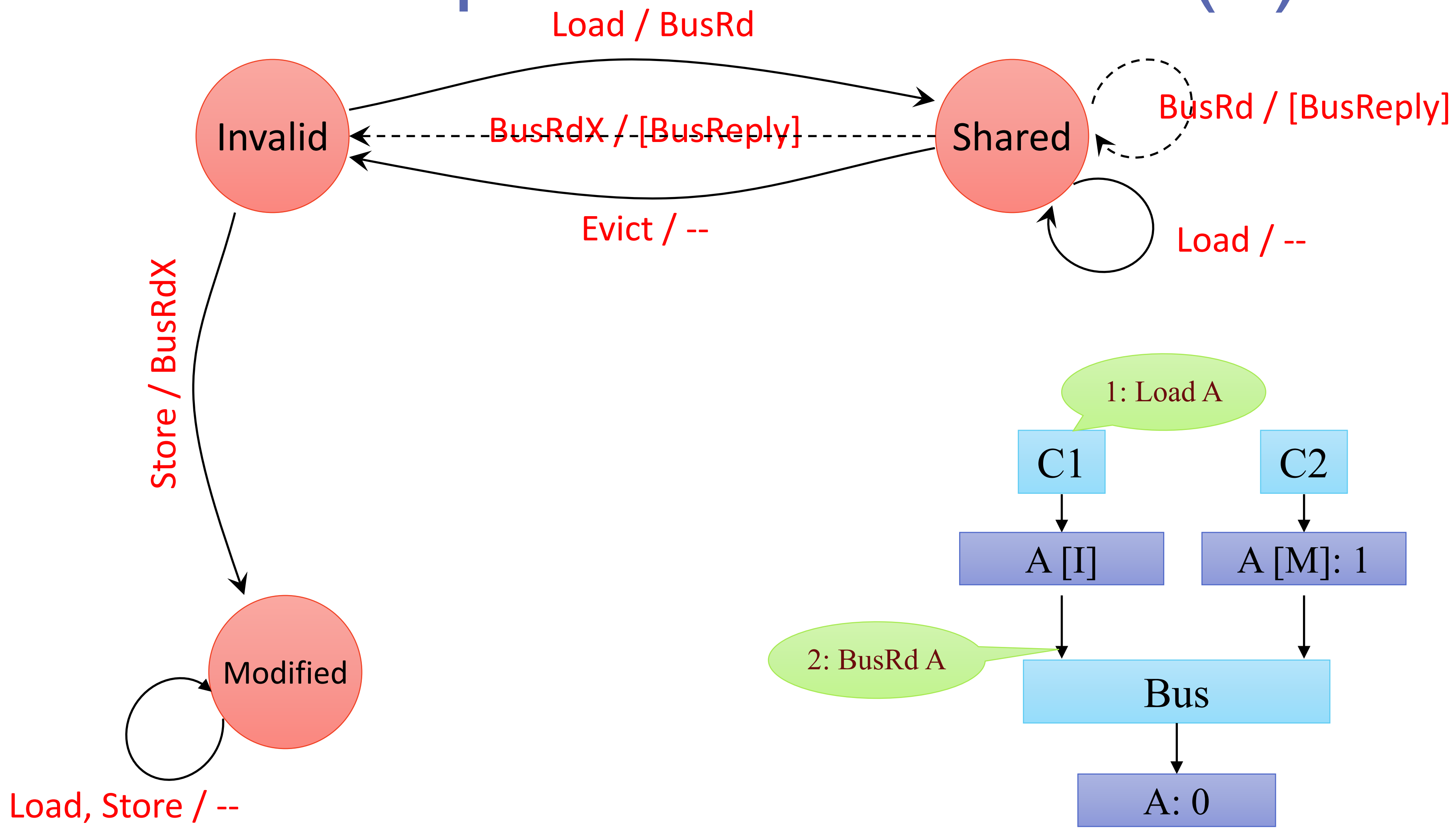
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (5)



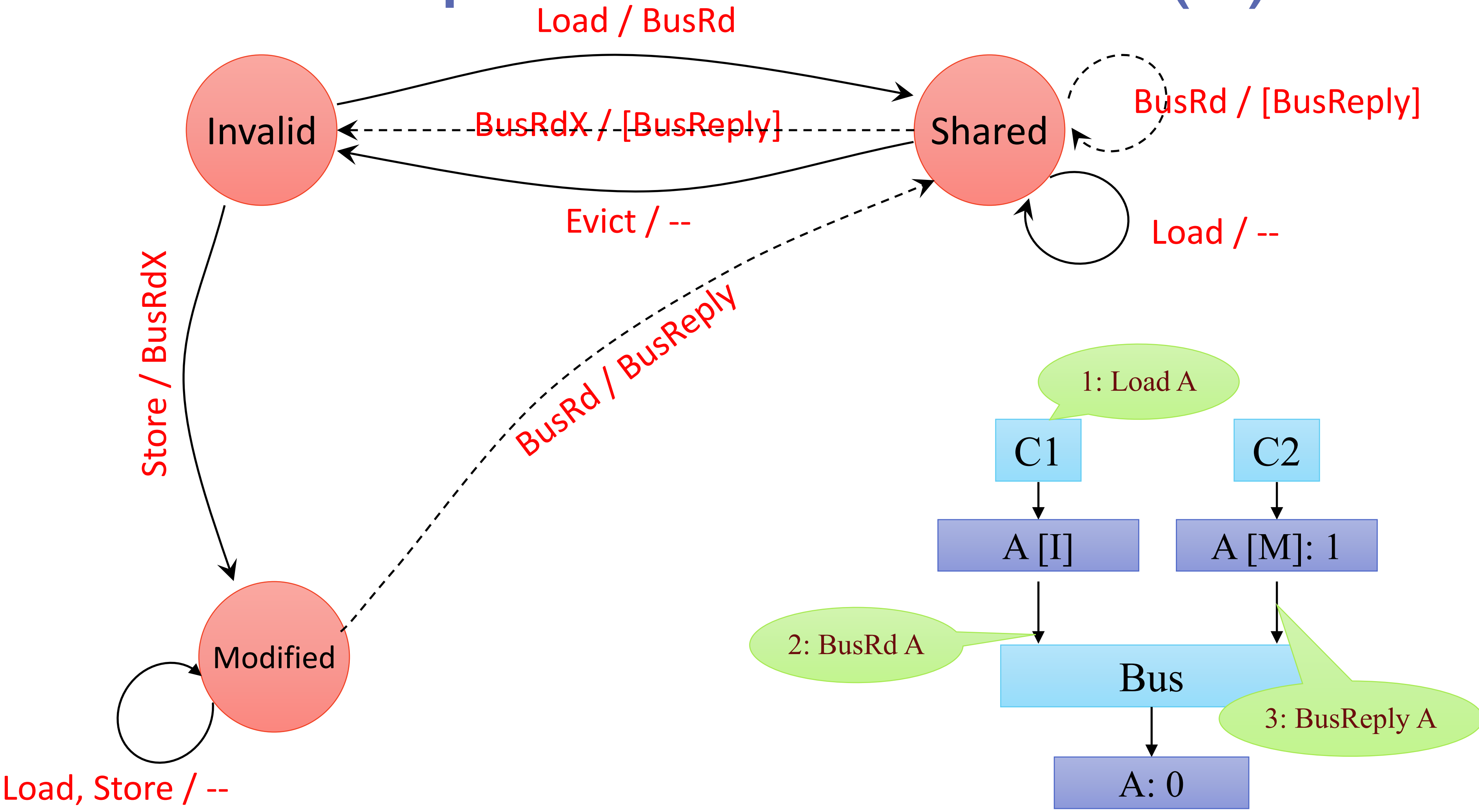
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (5)



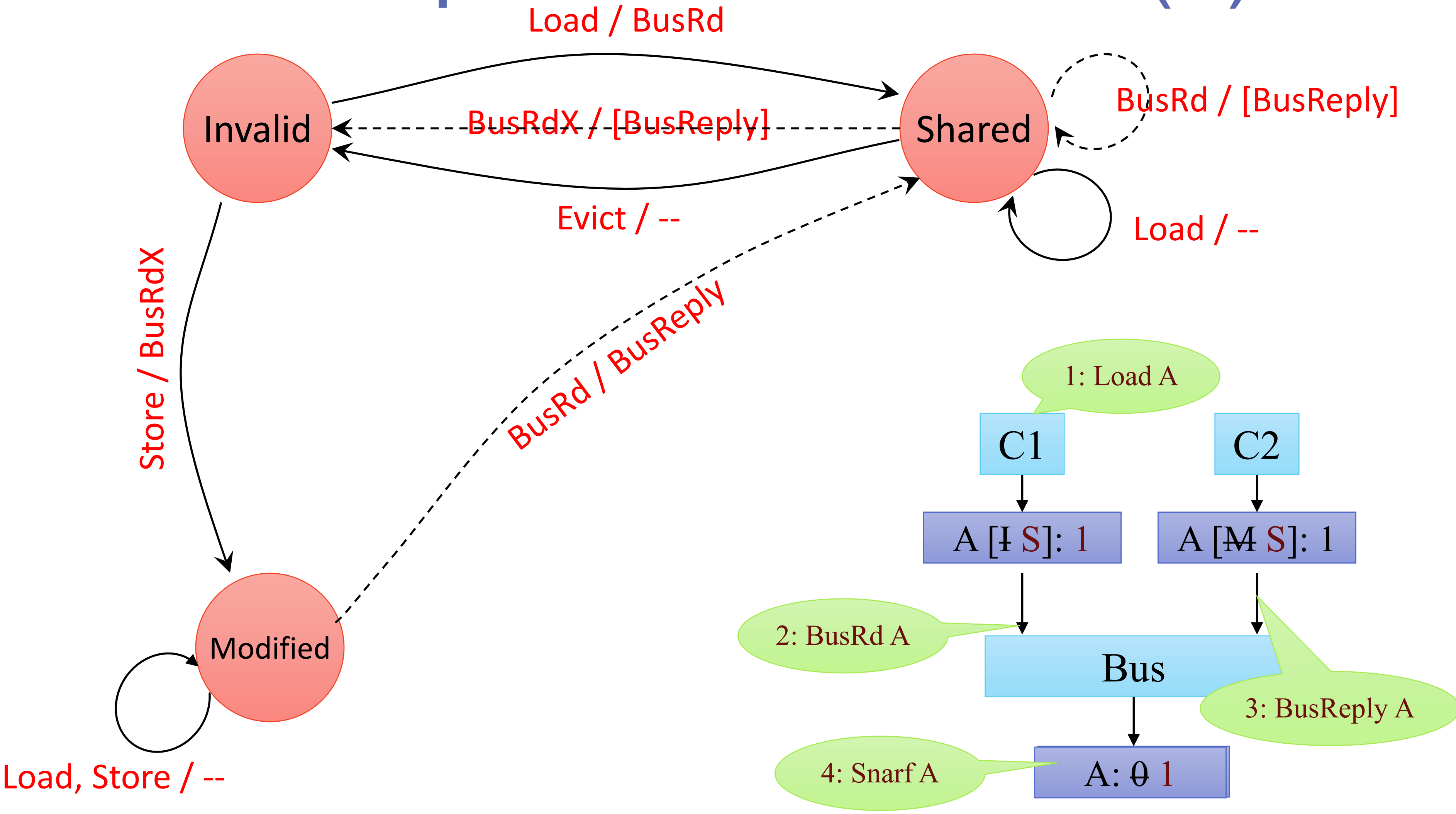
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (5)



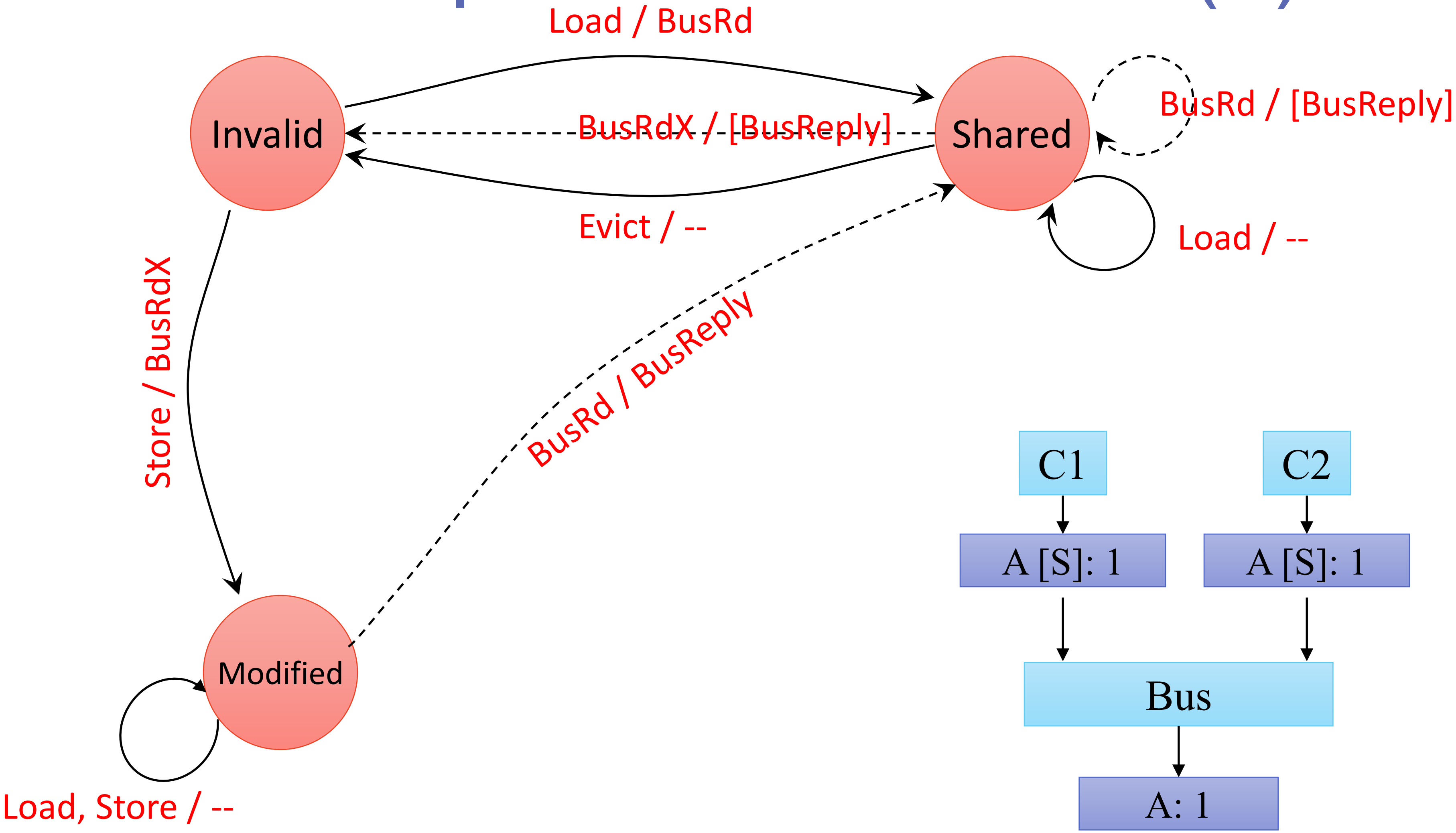
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (5)



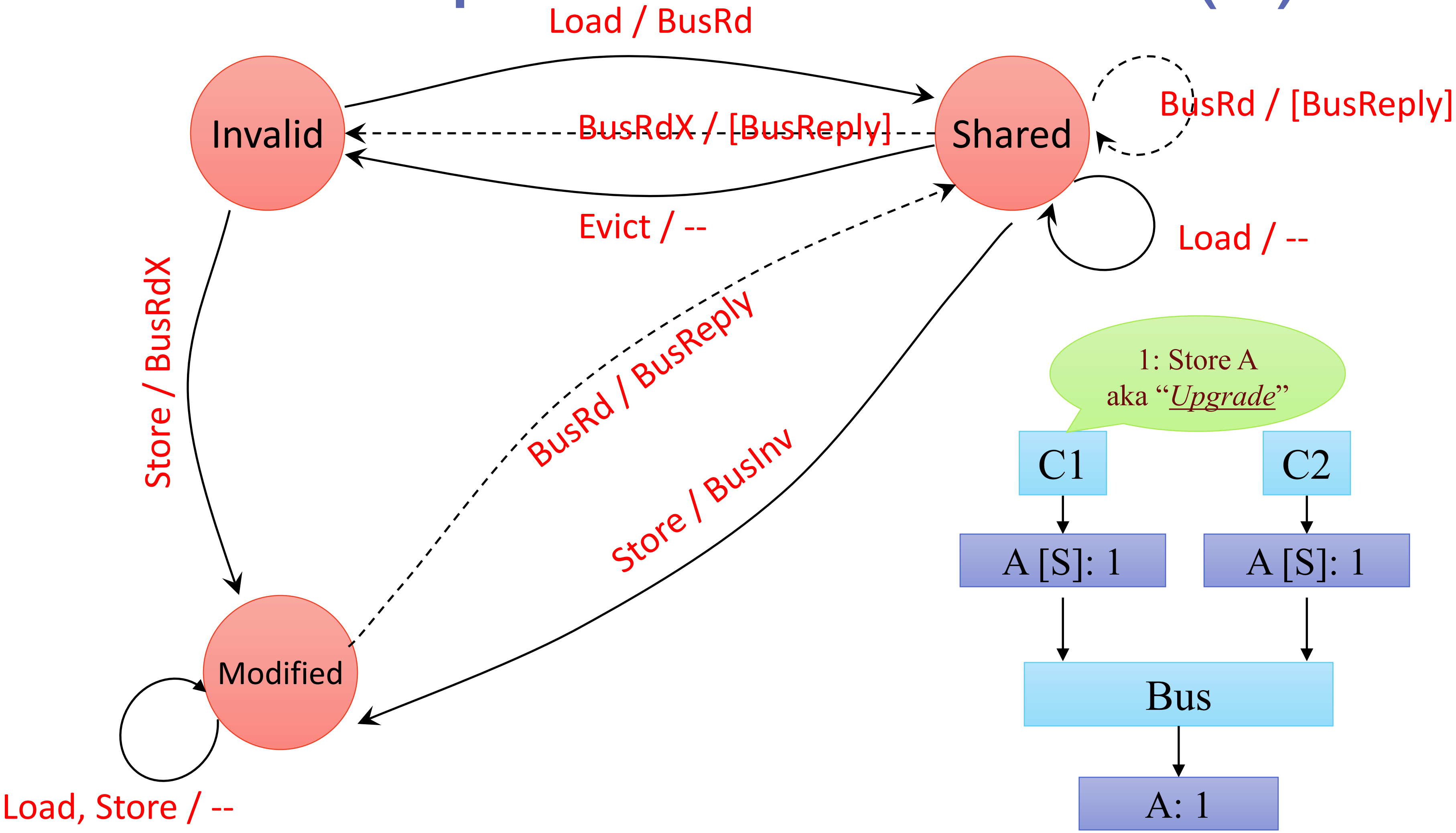
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (6)



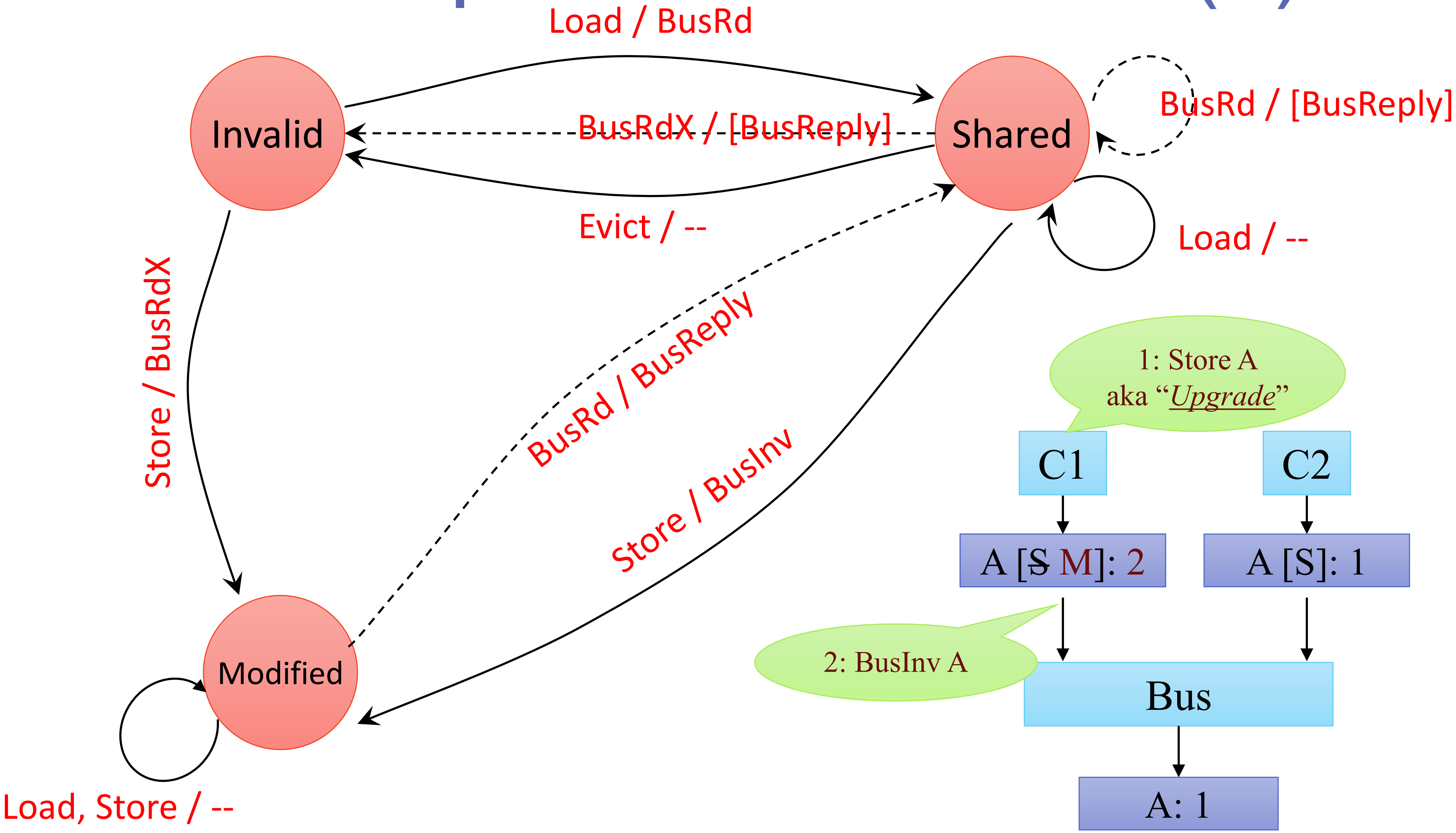
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (6)



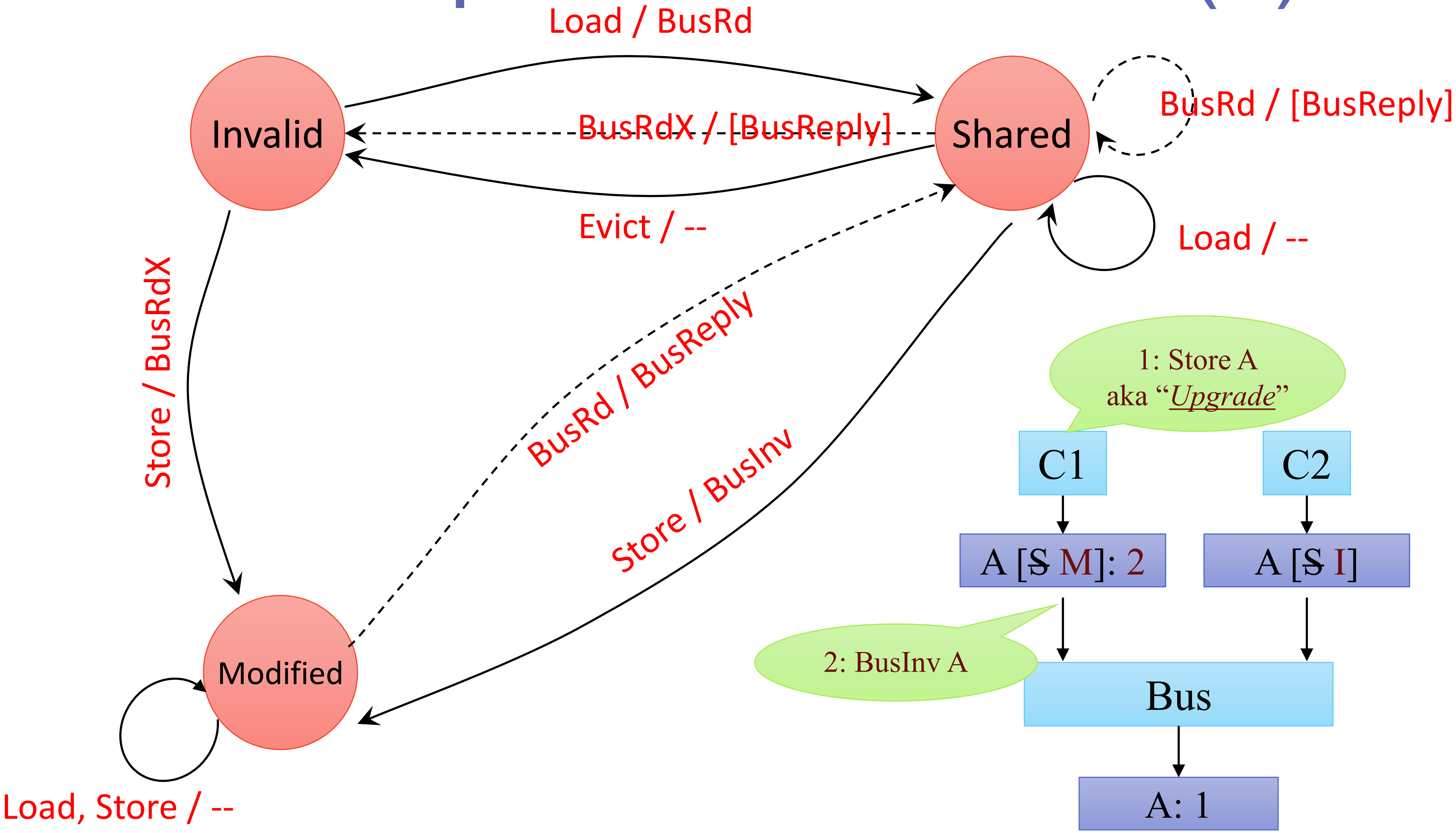
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (6)



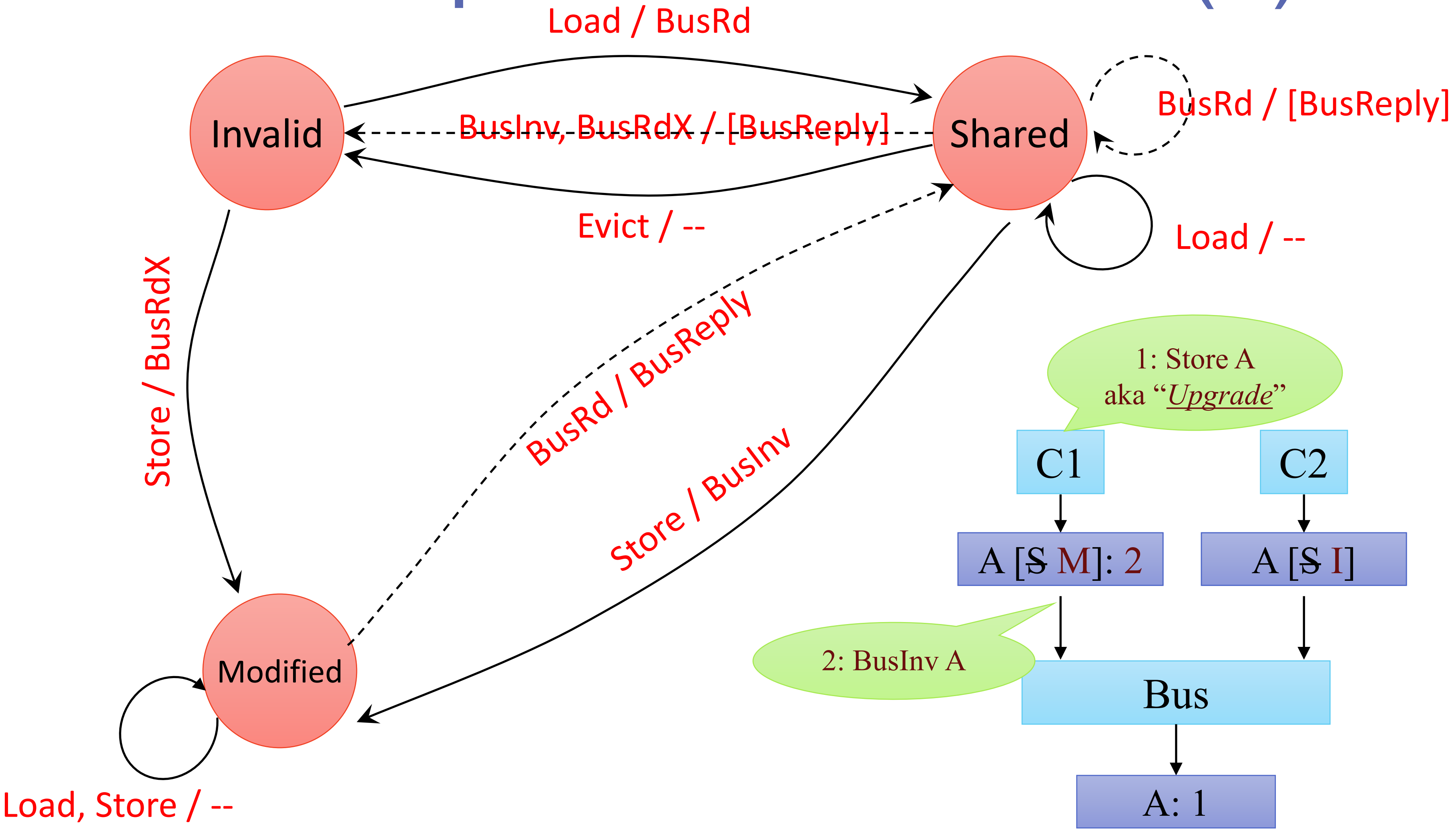
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (6)



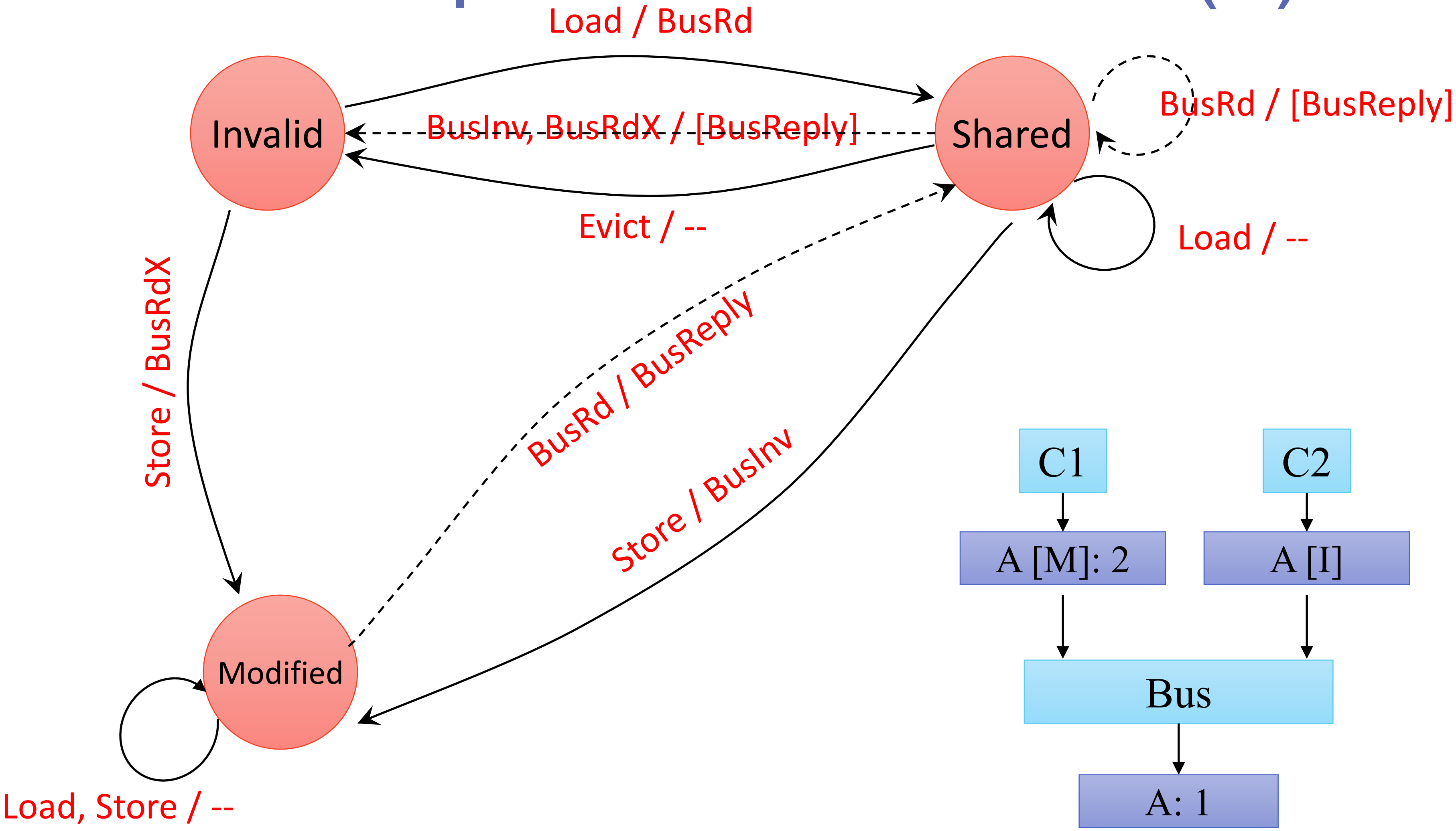
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (6)



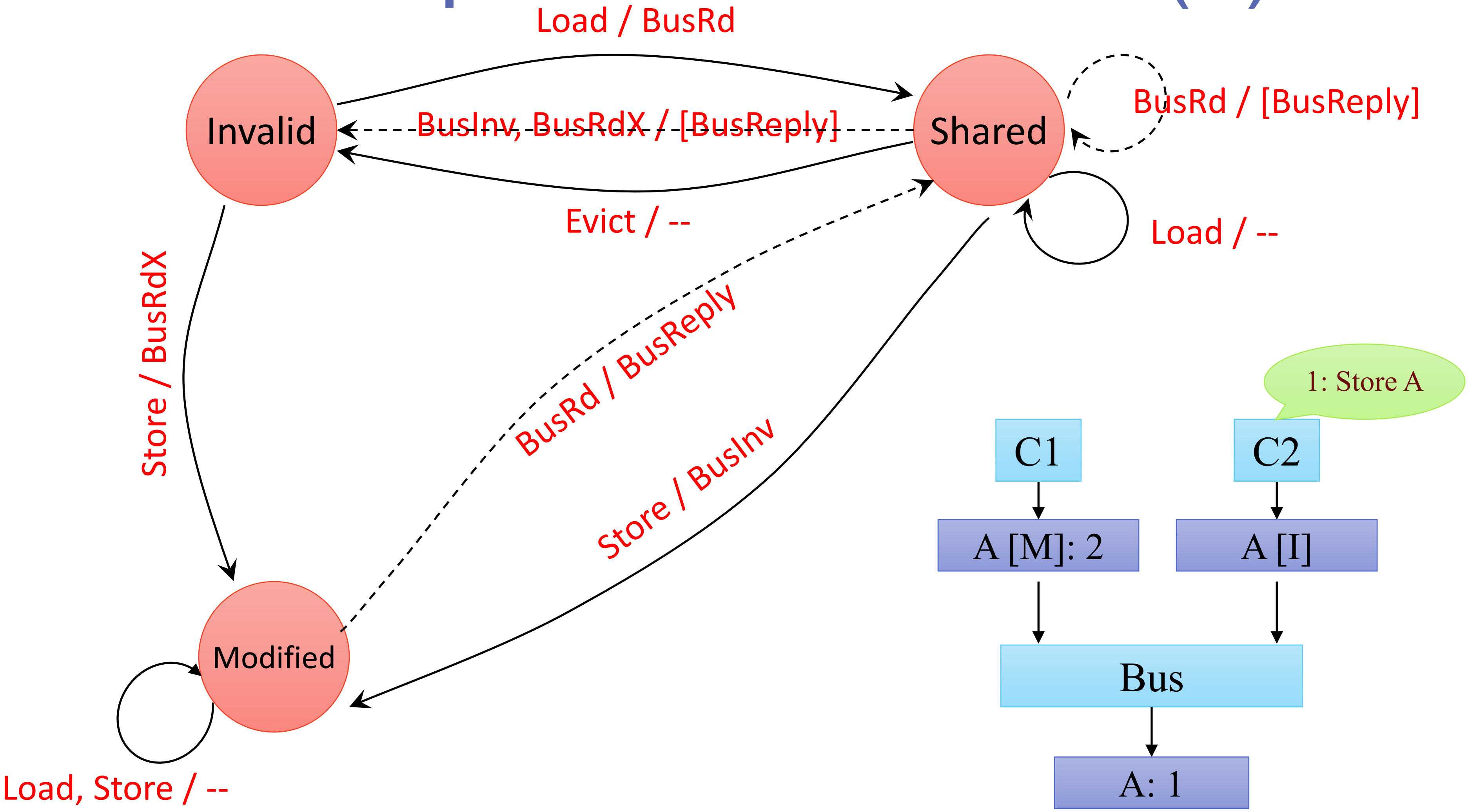
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (7)



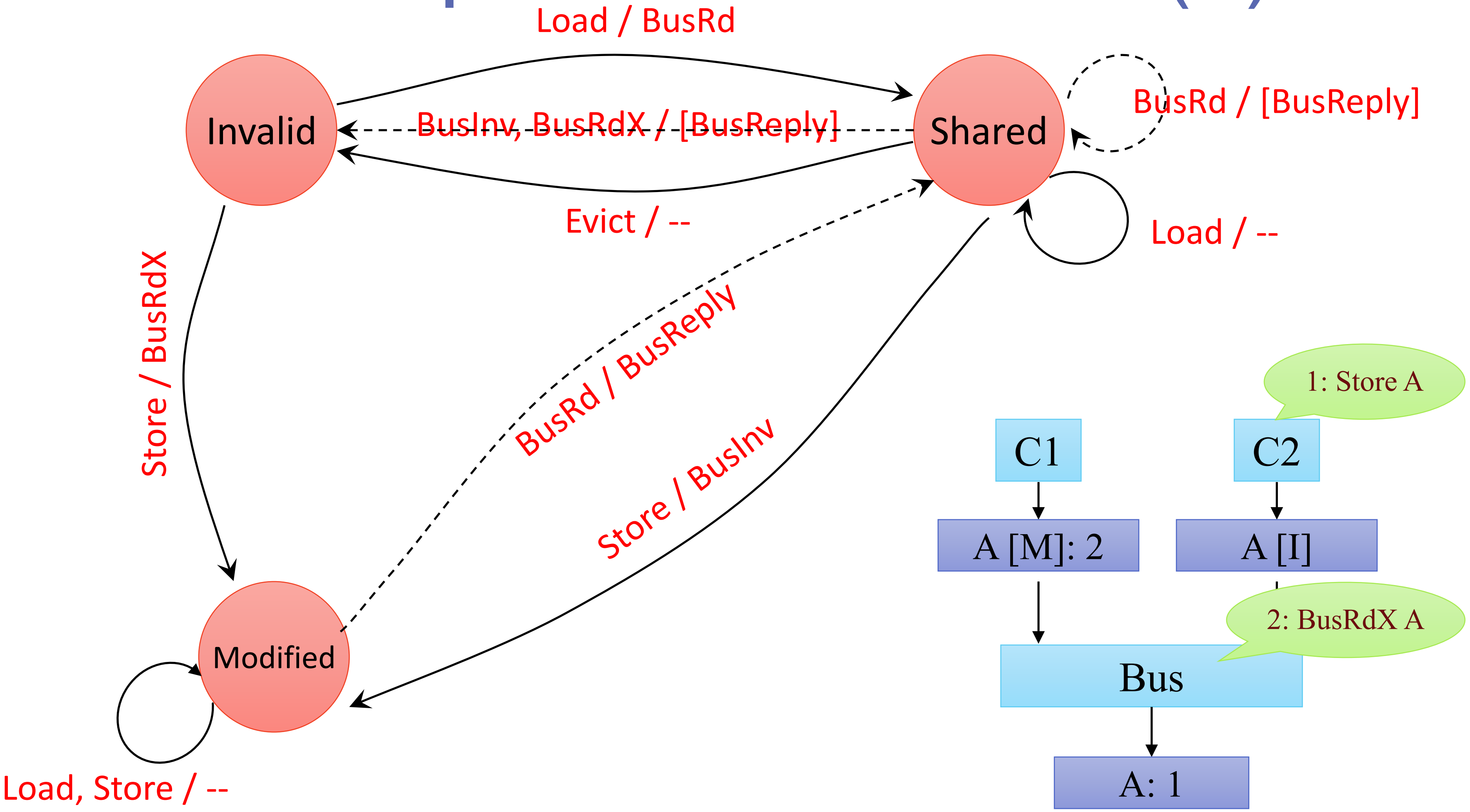
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (7)



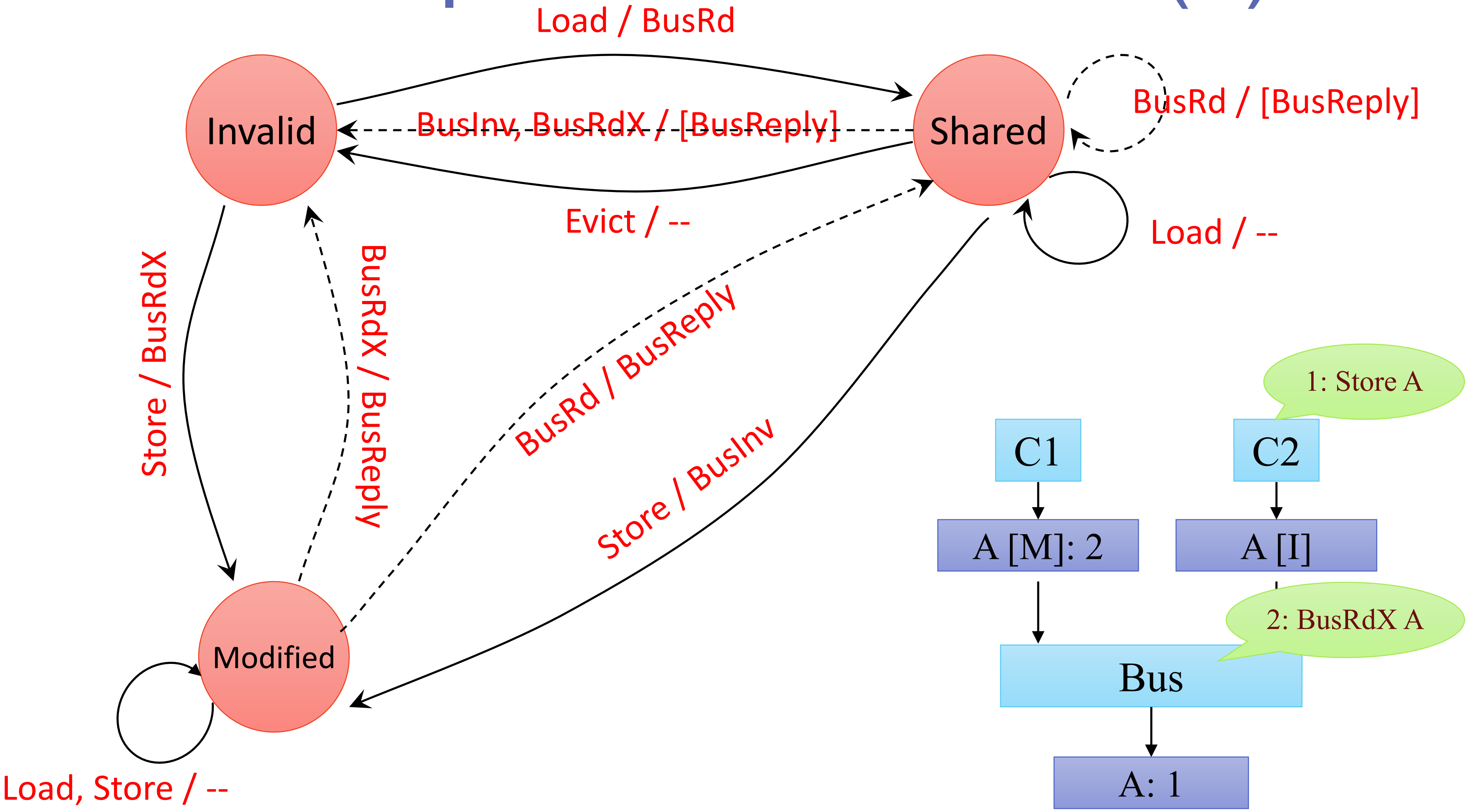
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (7)



Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (7)



Load / BusRd

Invalid

Shared

BusRd / [BusReply]

BusInv, BusRdX / [BusReply]

Evict / --

Load / --

Store / BusRdX

BusRdX / BusReply

BusRd / BusReply

Store / BusInv

Modified

Load, Store / --

1: Store A

C1

C2

A [M]: 2

A [I]

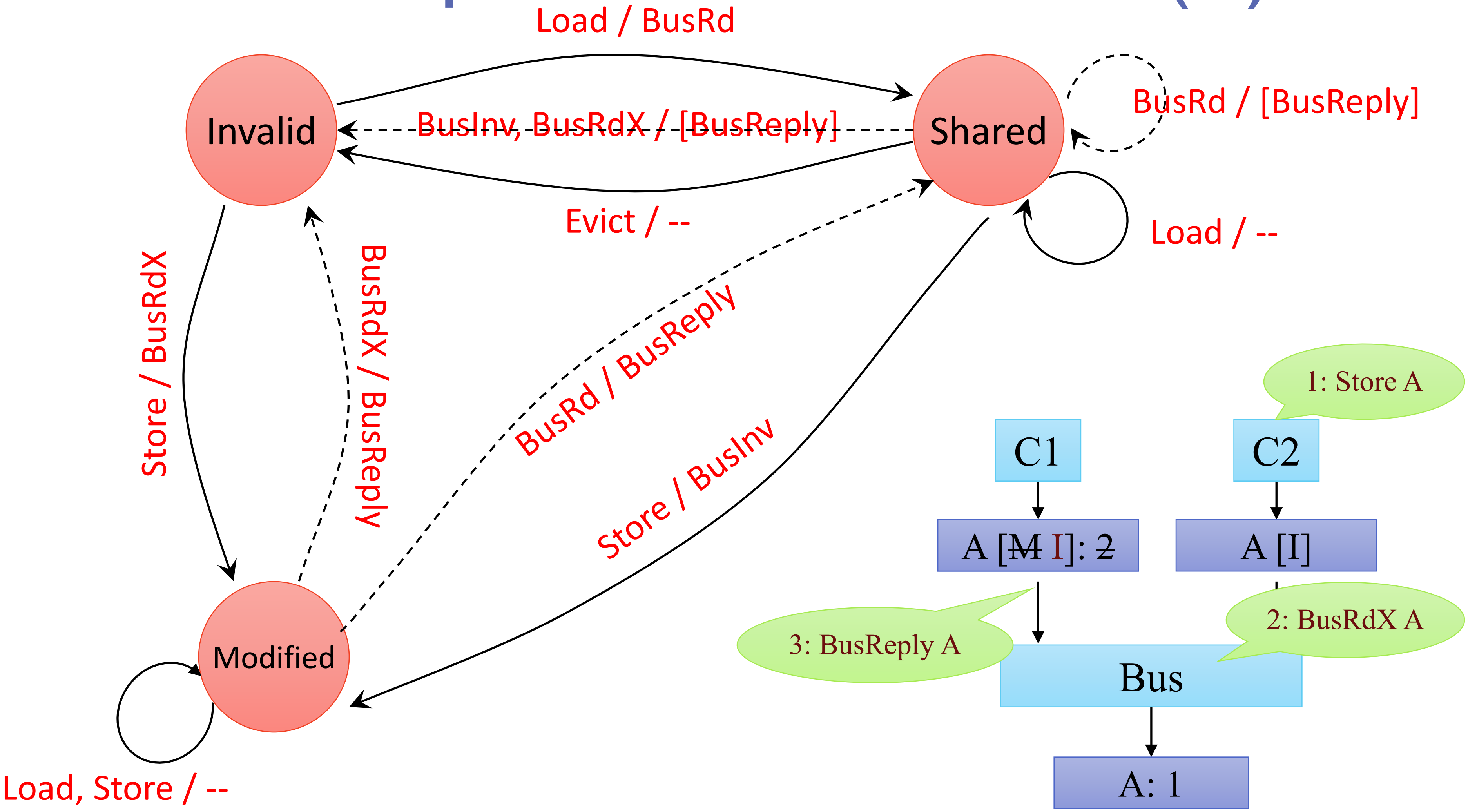
2: BusRdX A

Bus

A: 1

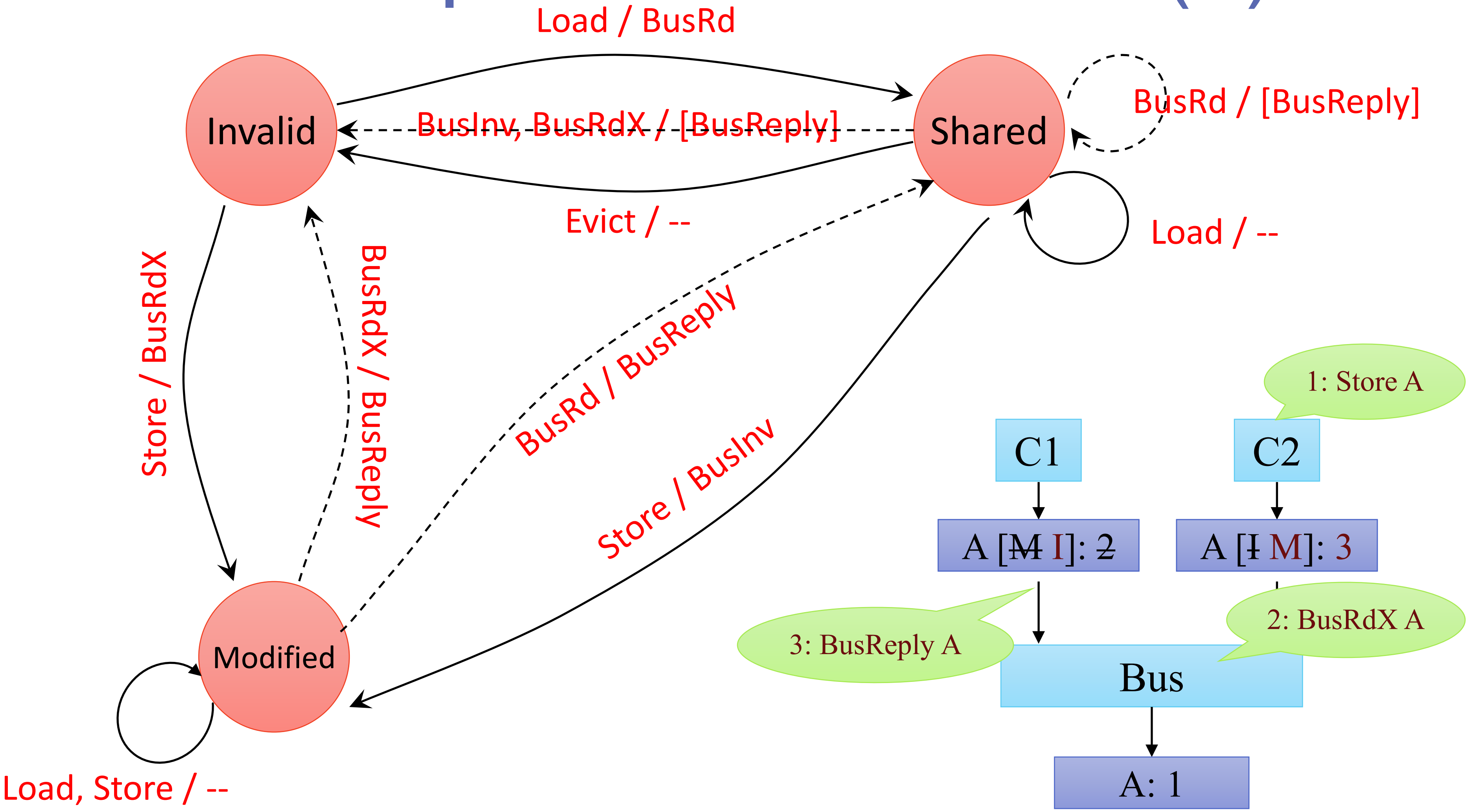
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (7)



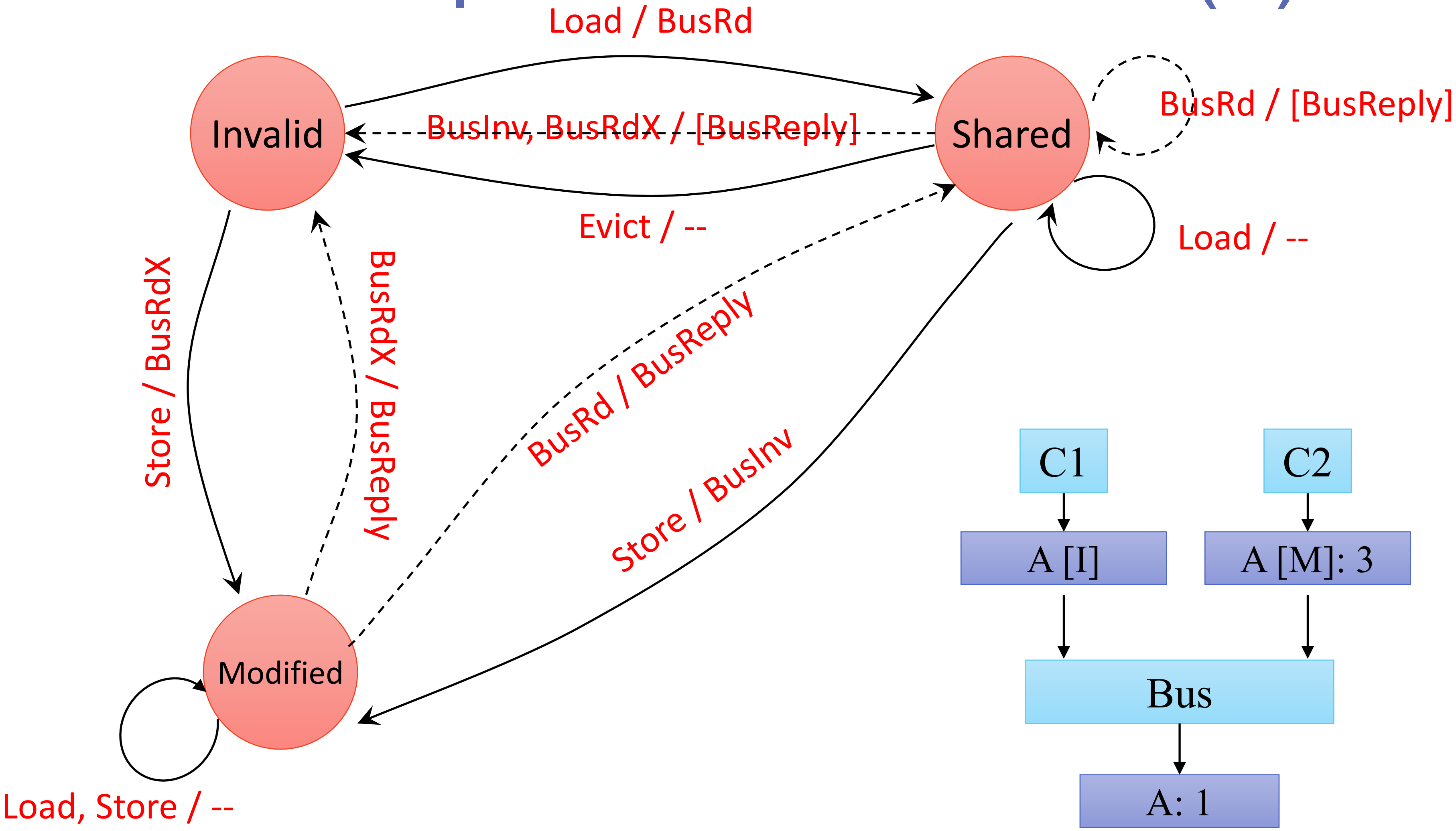
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (7)



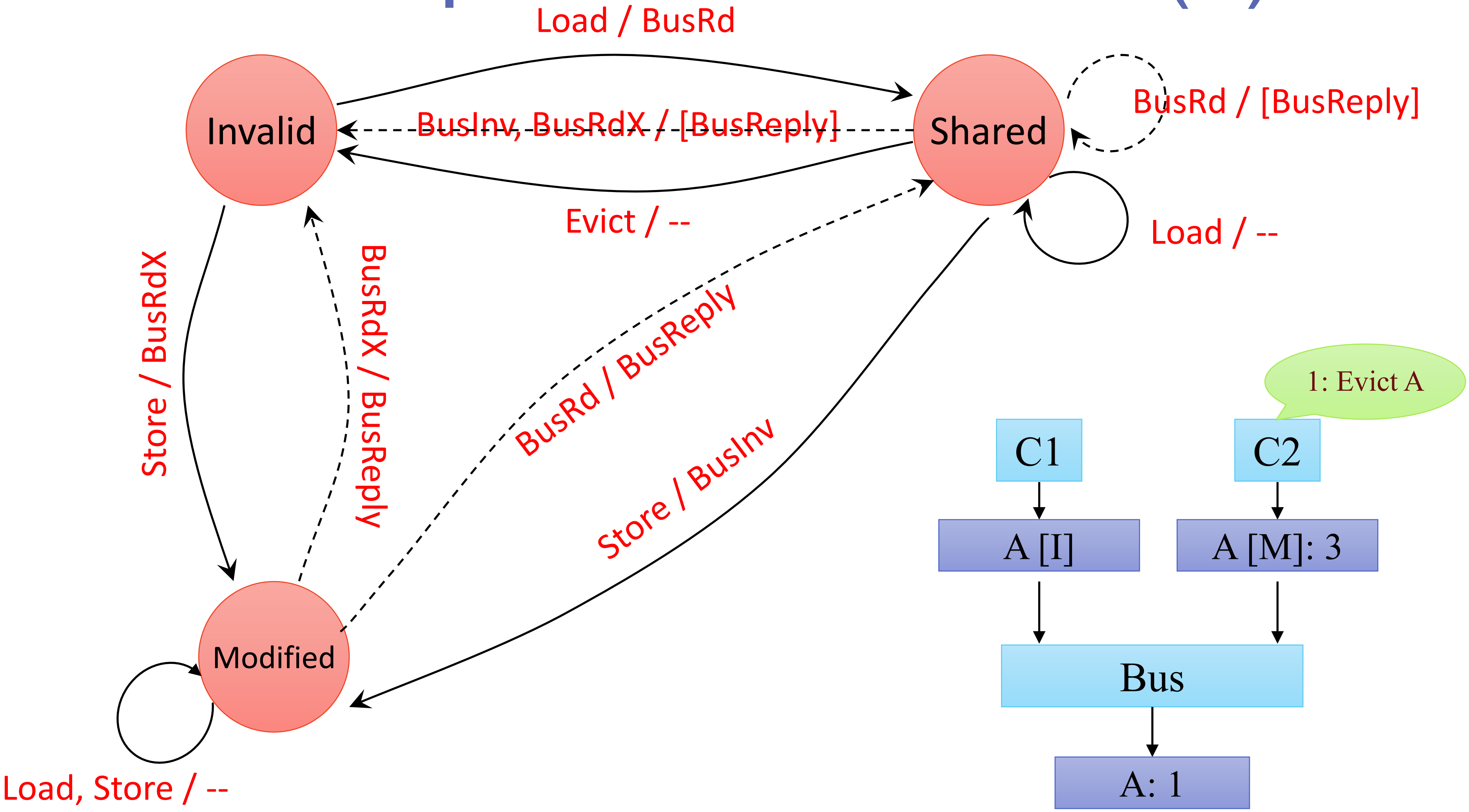
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (8)



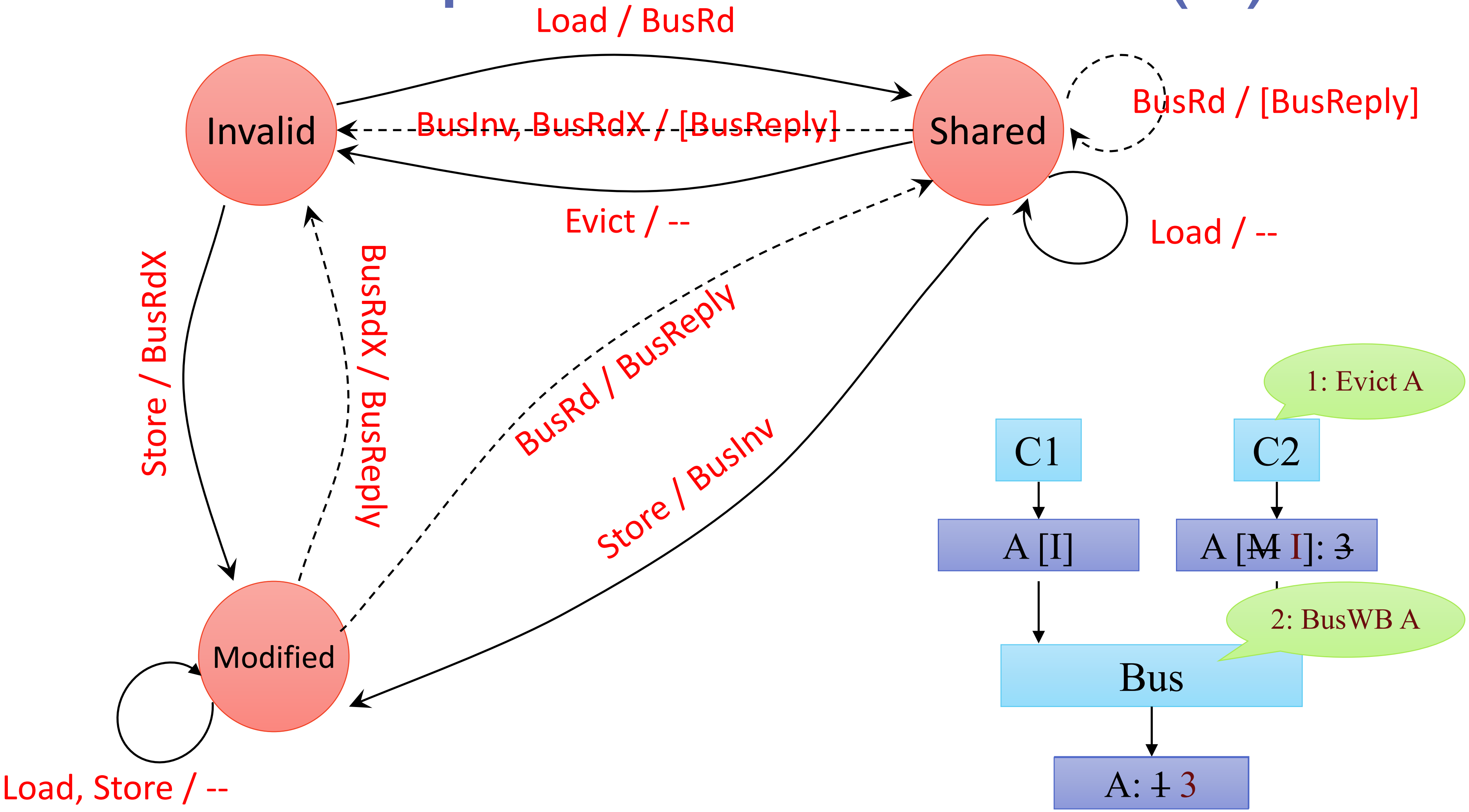
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (8)



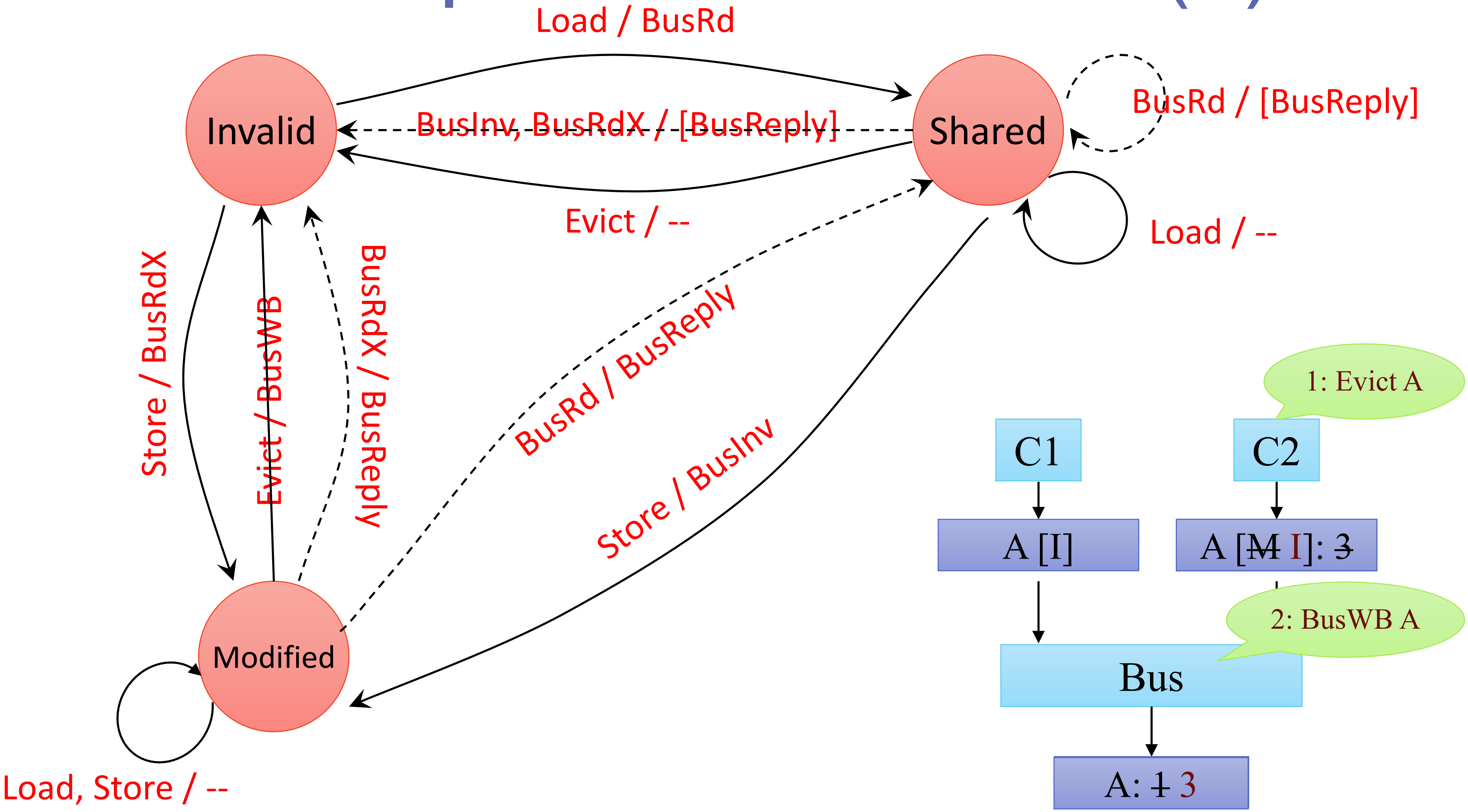
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (8)



Invariant: Only one writer (M) per block; Many sharers (S) okay

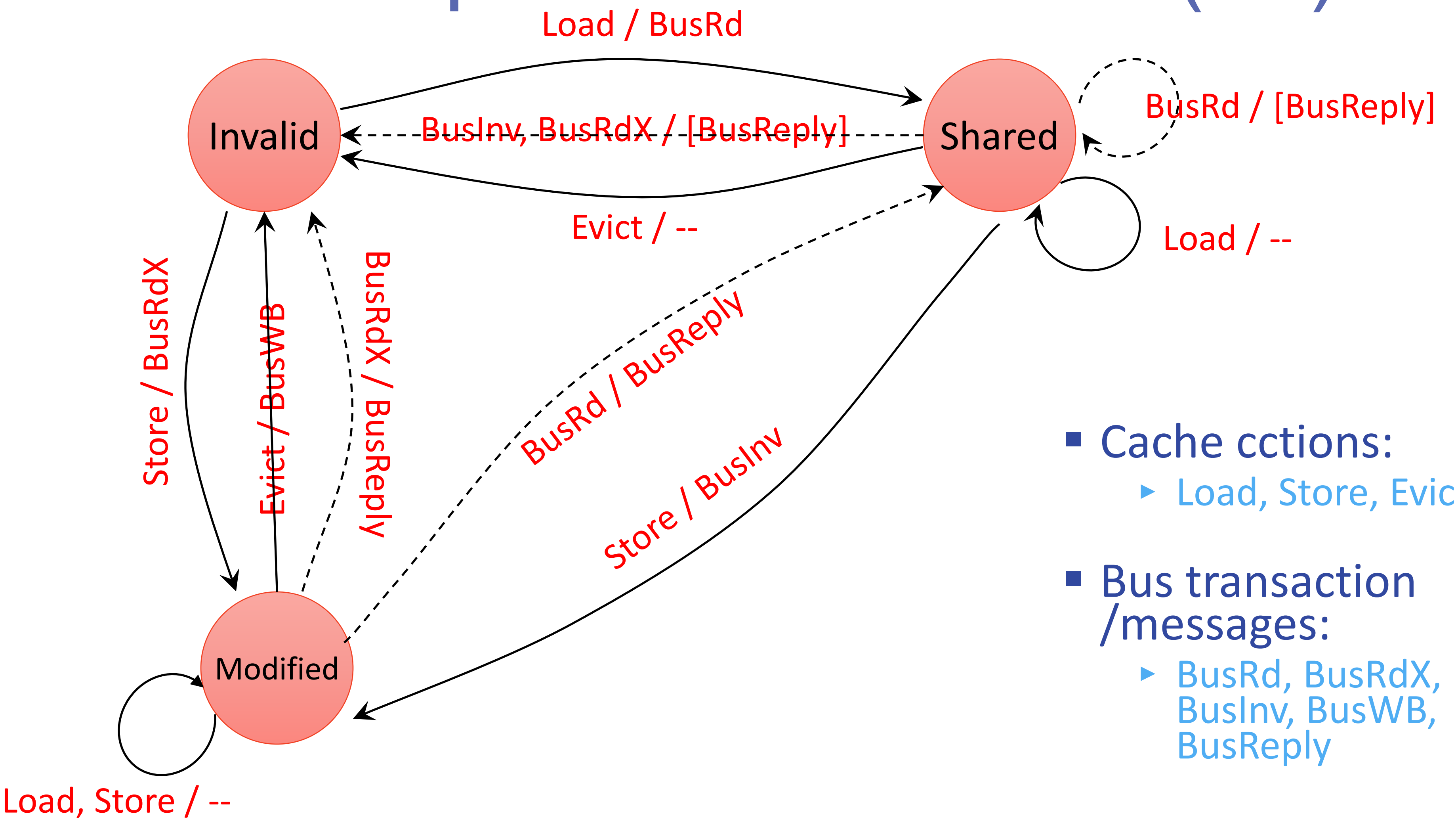
# Ex: MSI protocol in action (8)



Invariant: Only one writer (M) per block; Many sharers (S) okay

Indian Institute of Science (IISc), Bangalore, India

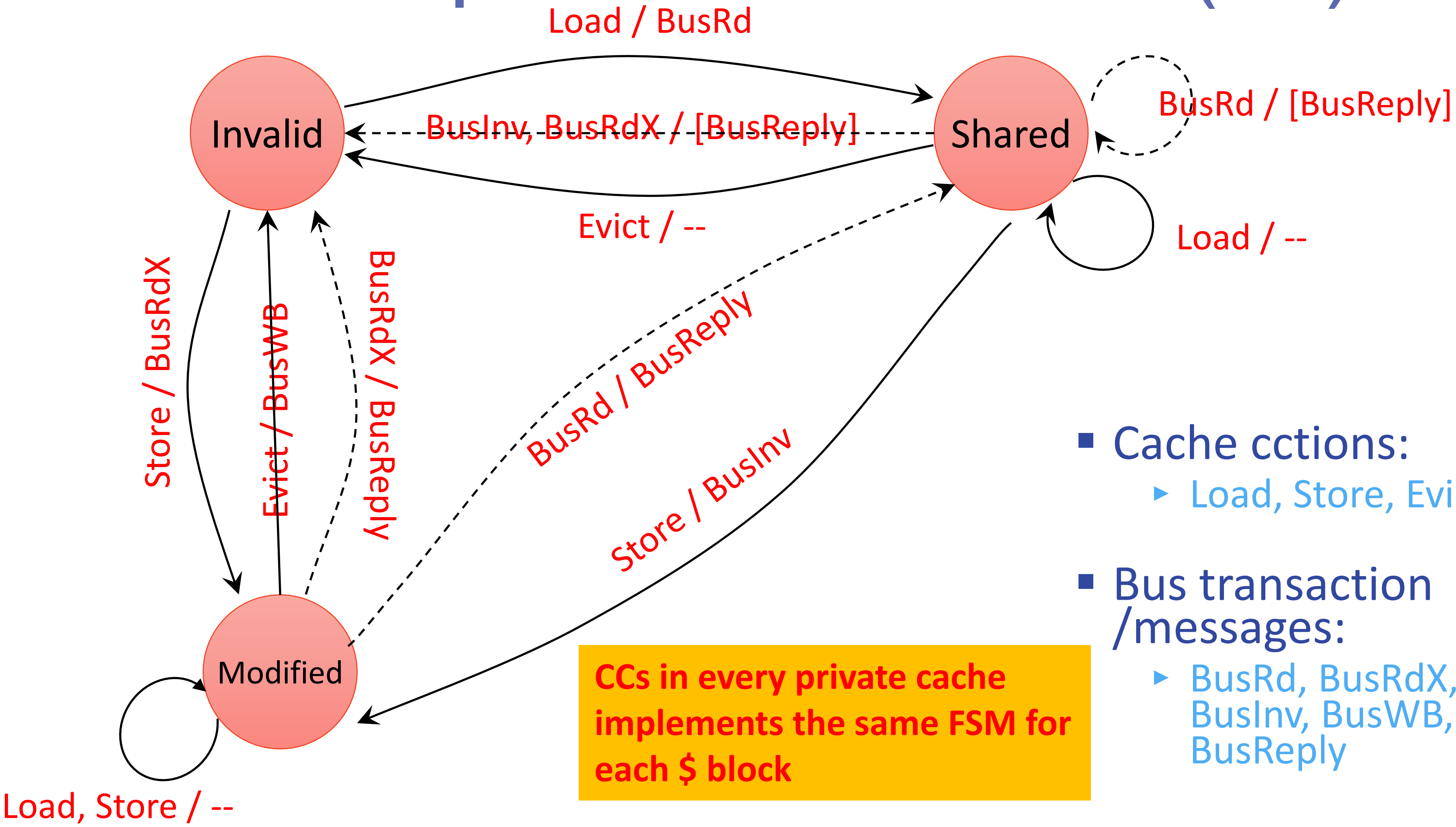
# Ex: MSI protocol in action (10)



- Cache cctions:
  - Load, Store, Evict
- Bus transaction /messages:
  - BusRd, BusRdX, BusInv, BusWB, BusReply

Invariant: Only one writer (M) per block; Many sharers (S) okay

# Ex: MSI protocol in action (10)

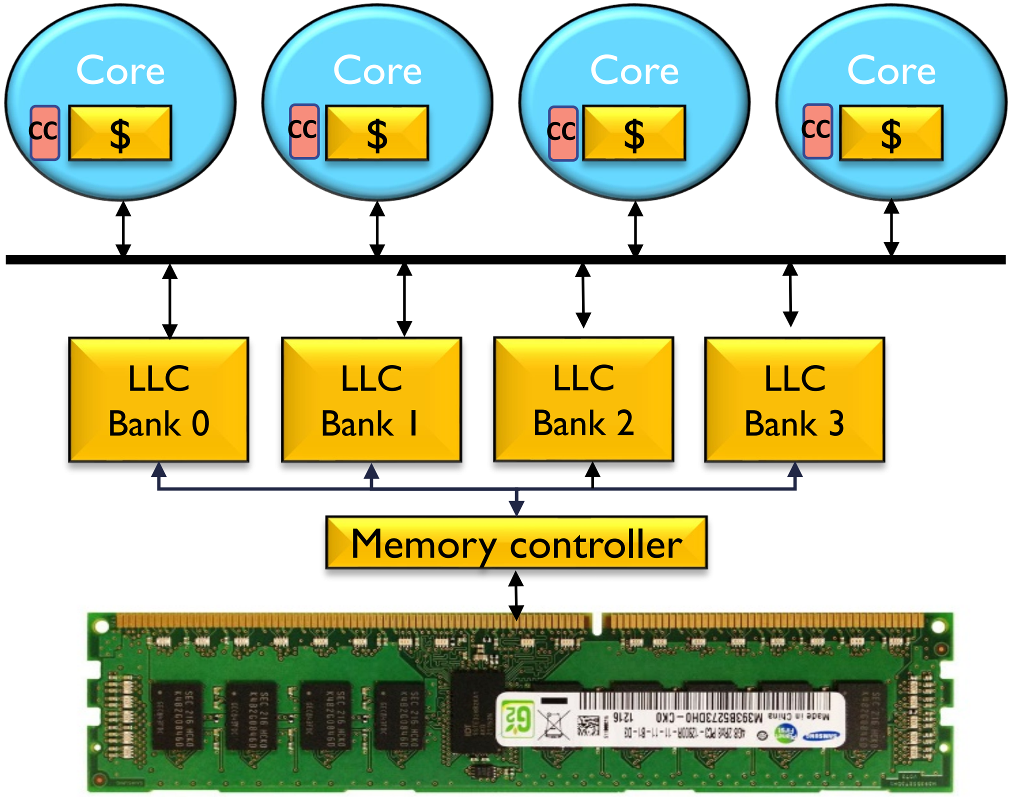


**CCs in every private cache implements the same FSM for each $ block**

- Cache cctions:
  - Load, Store, Evict
- Bus transaction /messages:
  - BusRd, BusRdX, BusInv, BusWB, BusReply

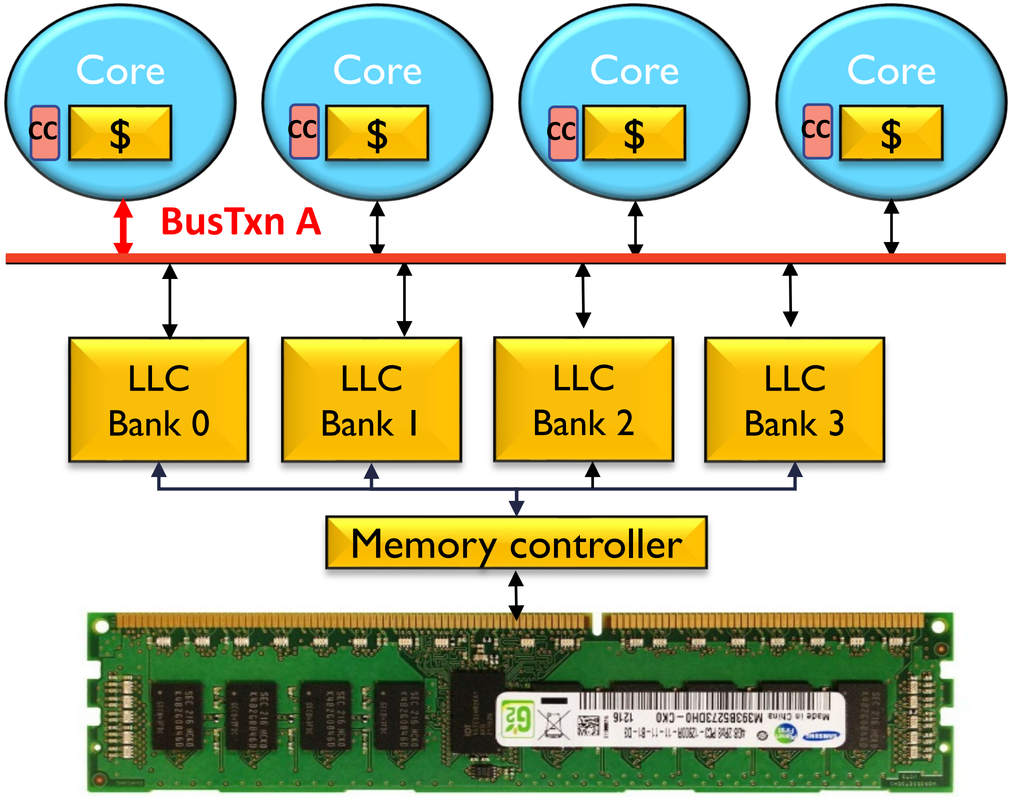
Invariant: Only one writer (M) per block; Many sharers (S) okay

# Putting it together: Coherence



Example: MSI coherence protocol

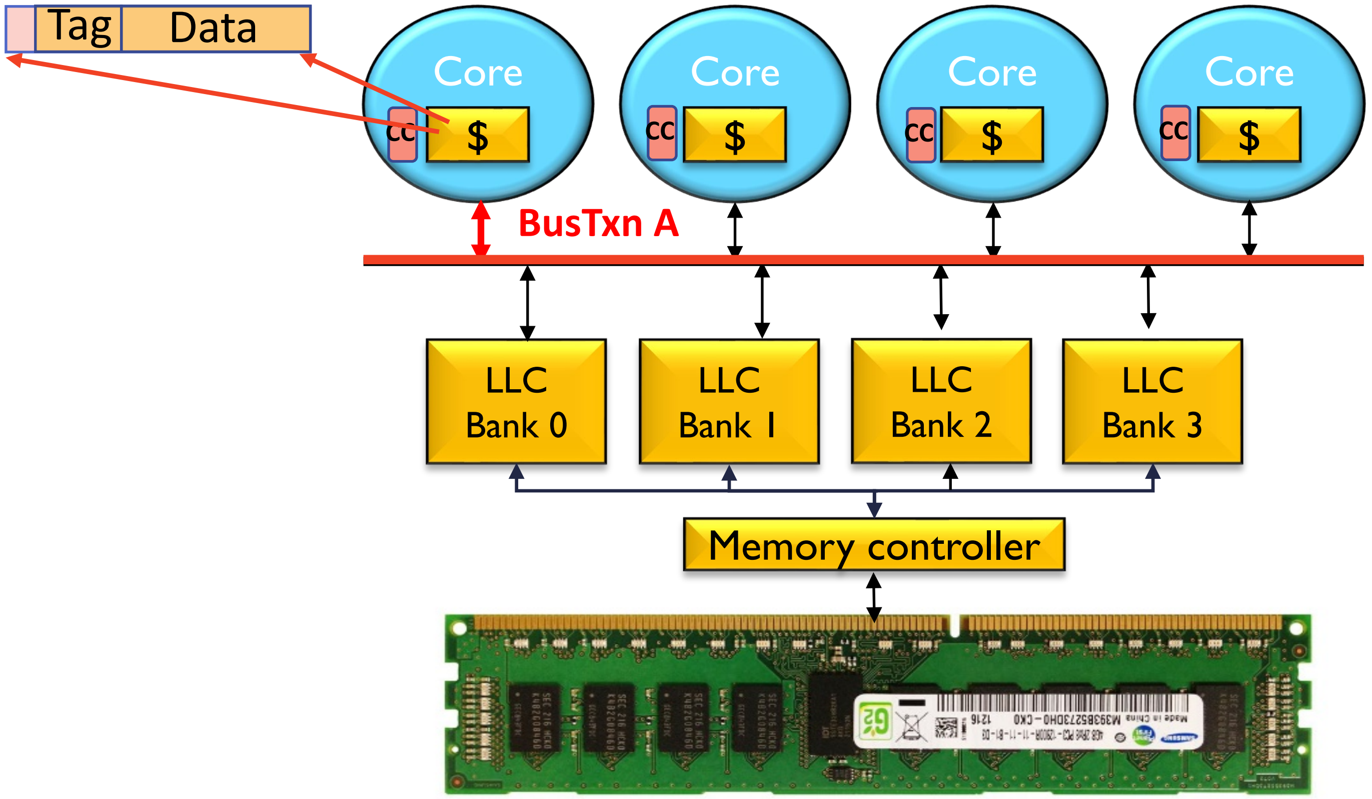
# Putting it together: Coherence



Example: MSI coherence protocol

# Putting it together: Coherence

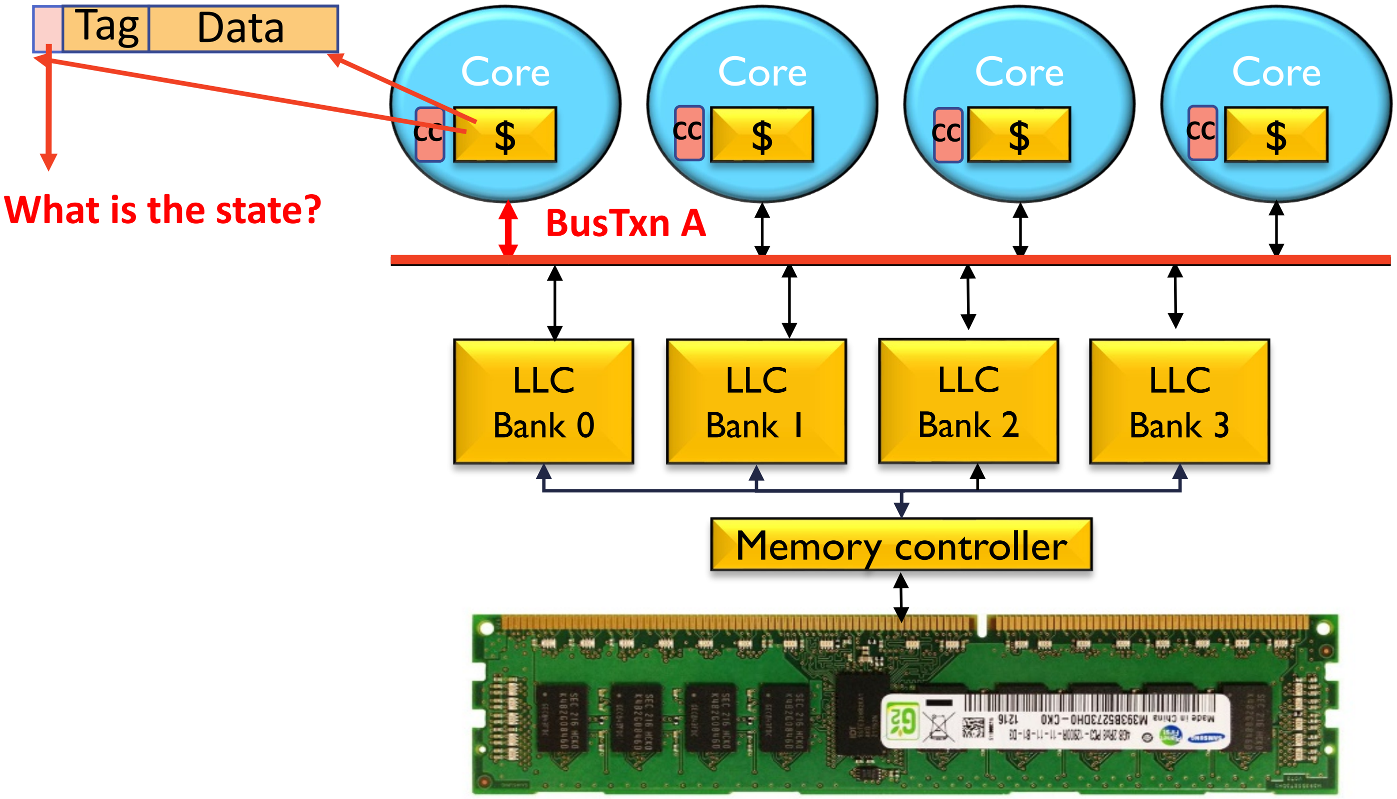
$ A (if present, "I" state otherwise)



Example: MSI coherence protocol

# Putting it together: Coherence

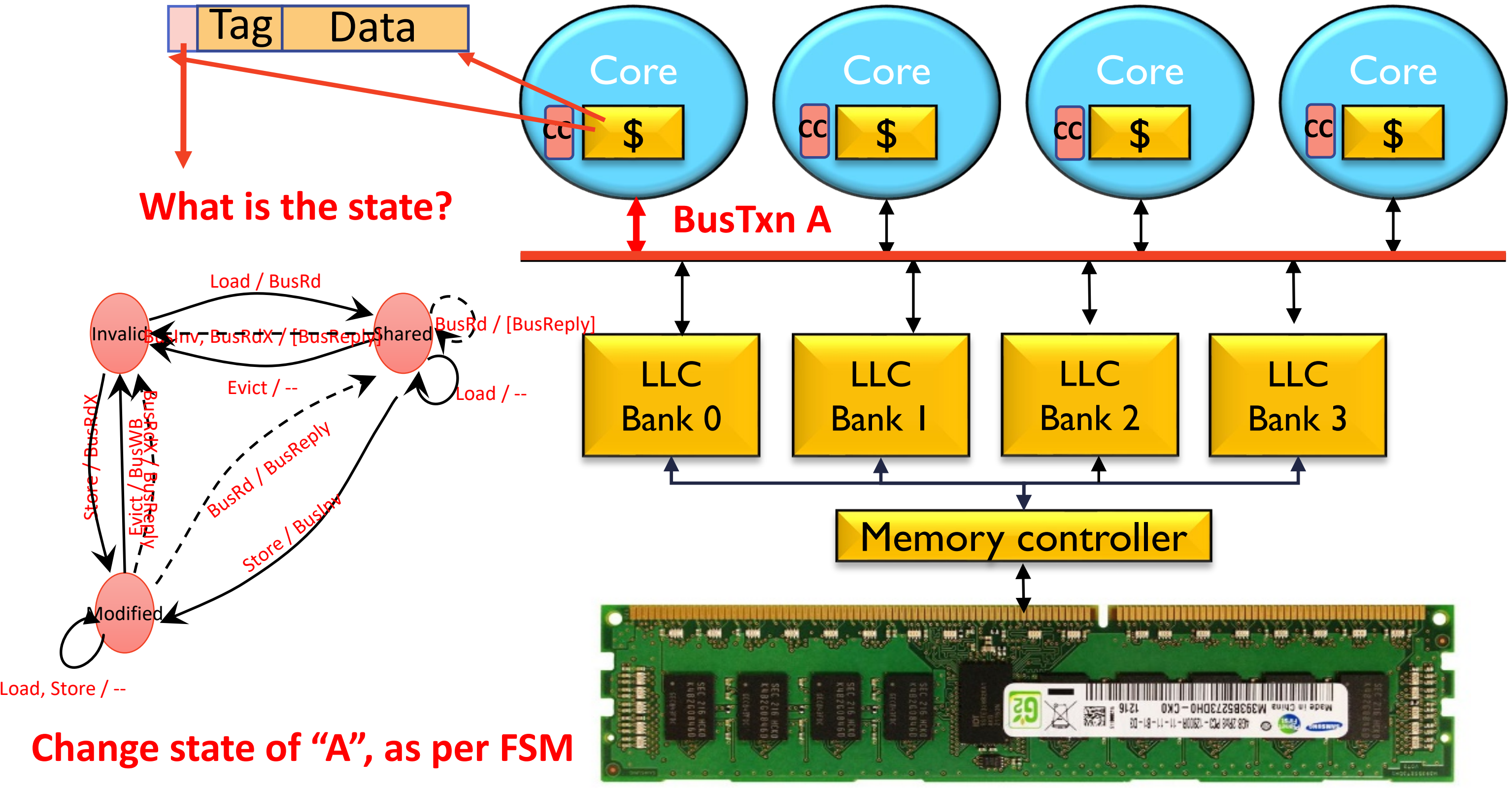
$ A (if present, "I" state otherwise)



Example: MSI coherence protocol

# Putting it together: Coherence

$ A (if present, “I” state otherwise)

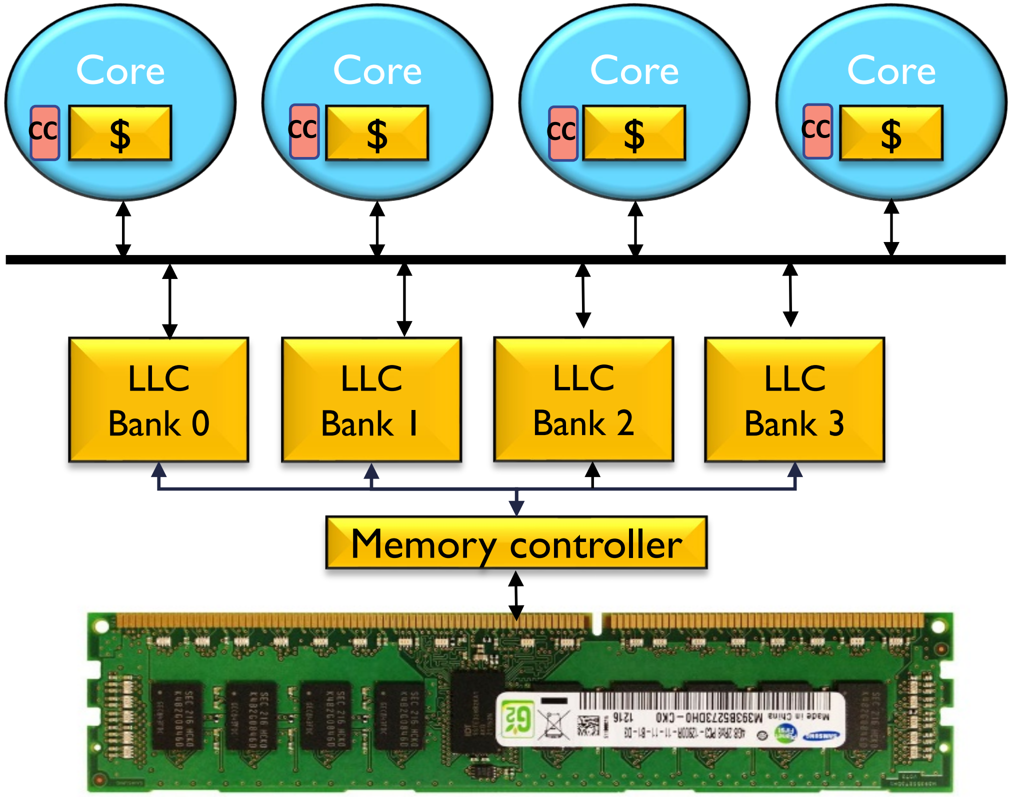


What is the state?

Change state of “A”, as per FSM

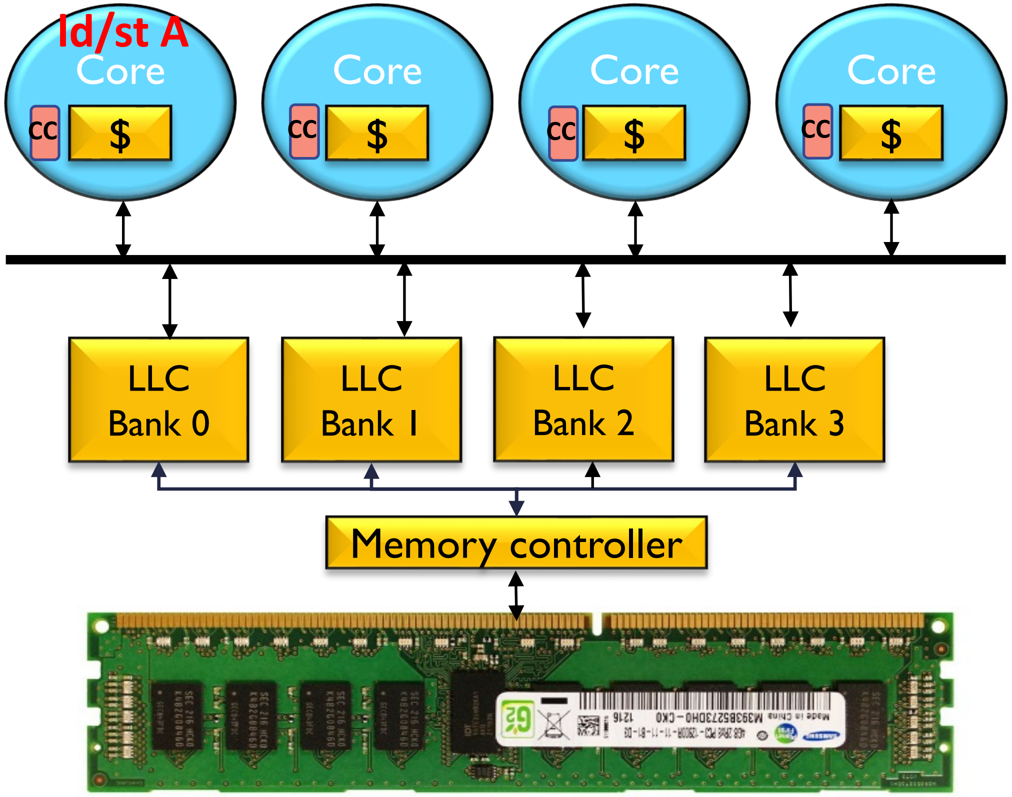
Example: MSI coherence protocol

# Putting it together: Coherence



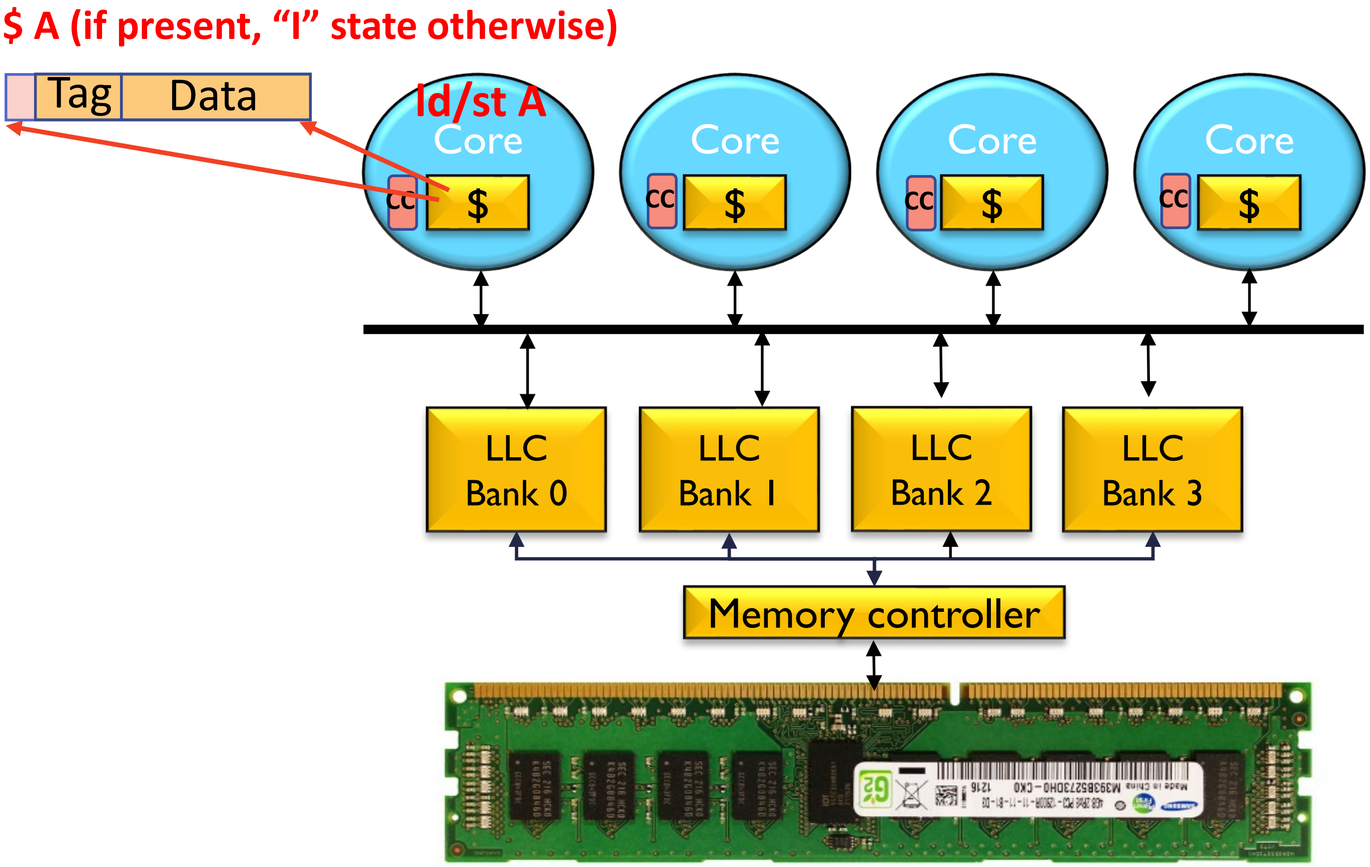
Example: MSI coherence protocol

# Putting it together: Coherence



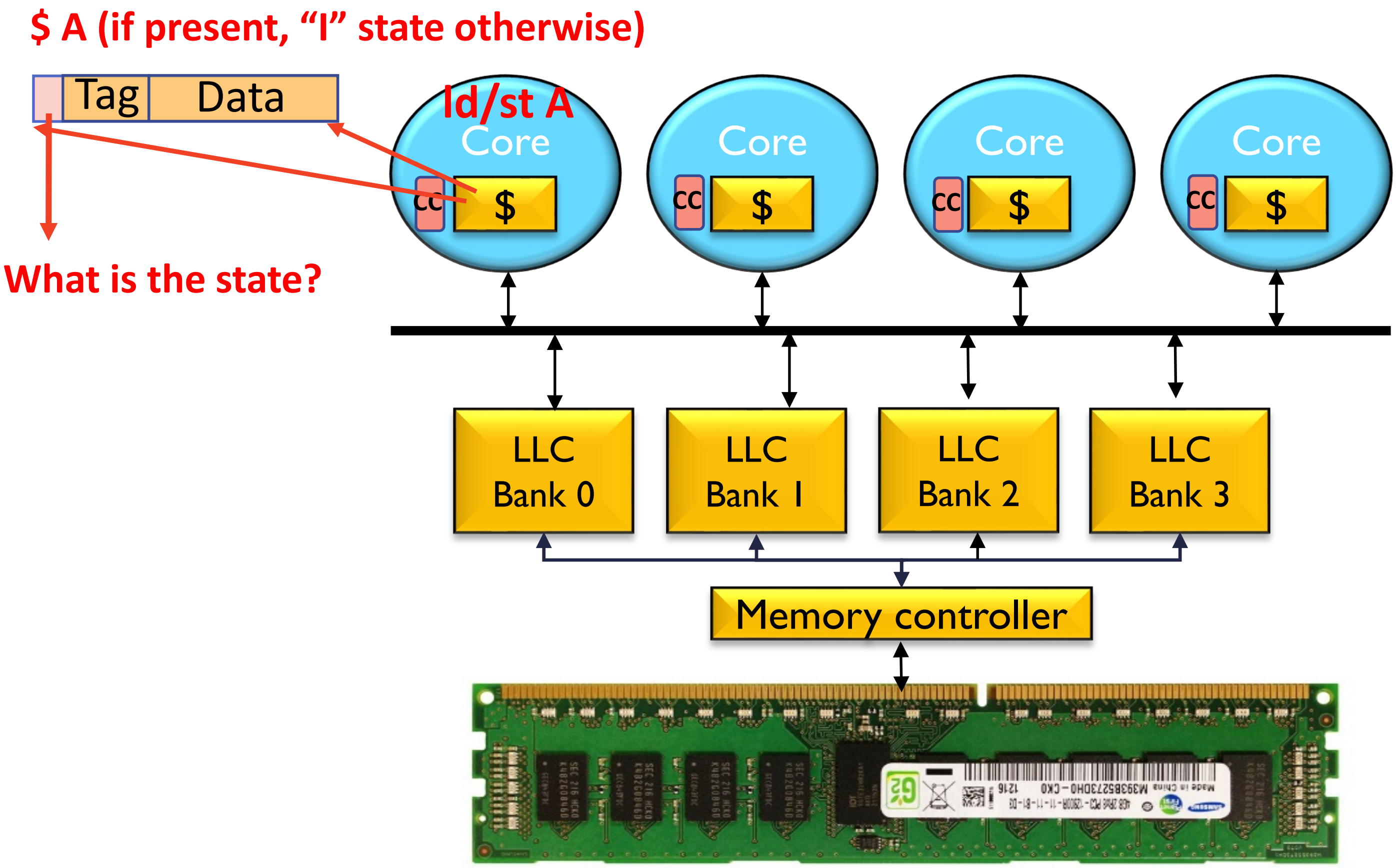
Example: MSI coherence protocol

# Putting it together: Coherence



Example: MSI coherence protocol

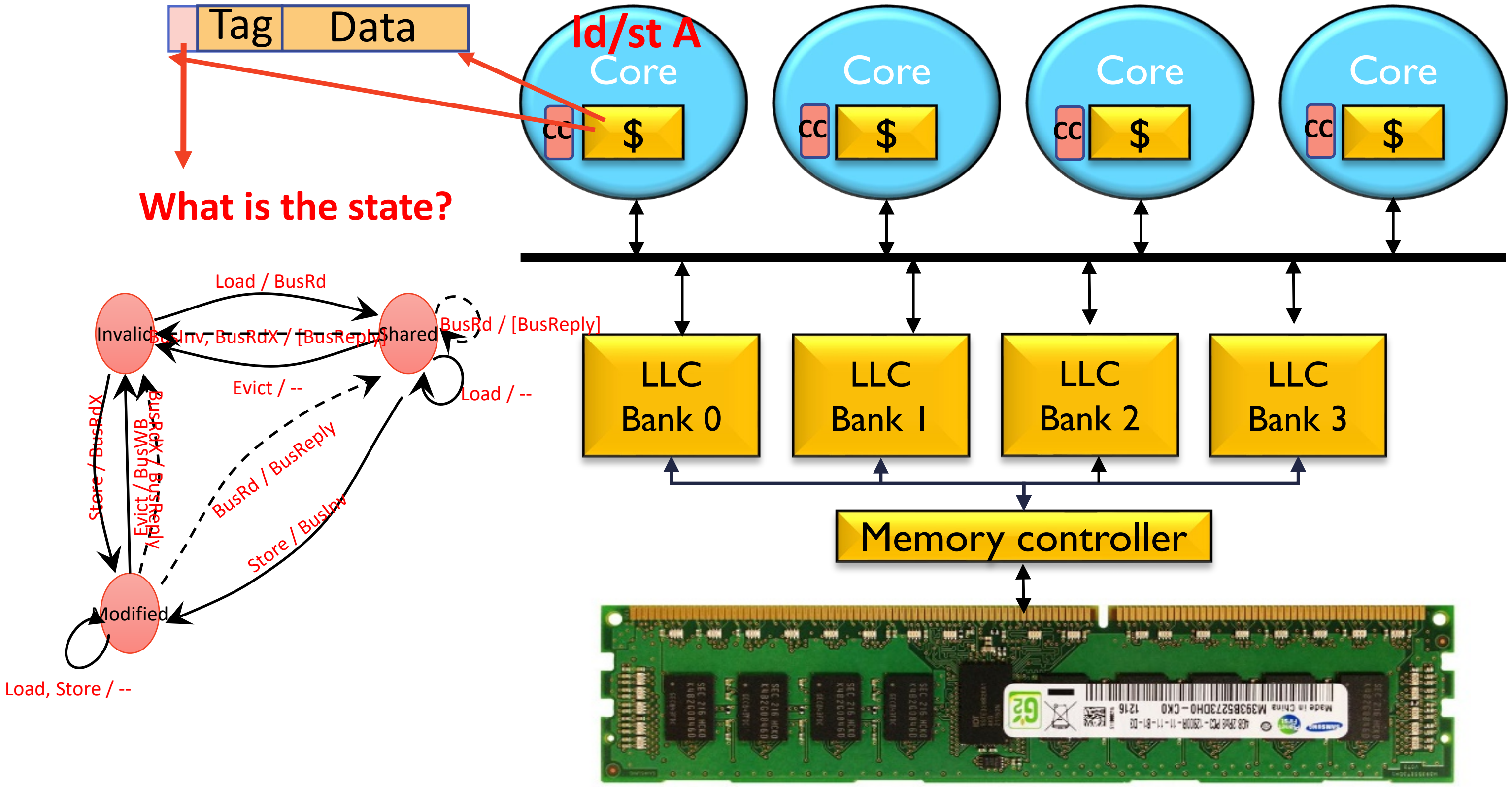
# Putting it together: Coherence



Example: MSI coherence protocol

# Putting it together: Coherence

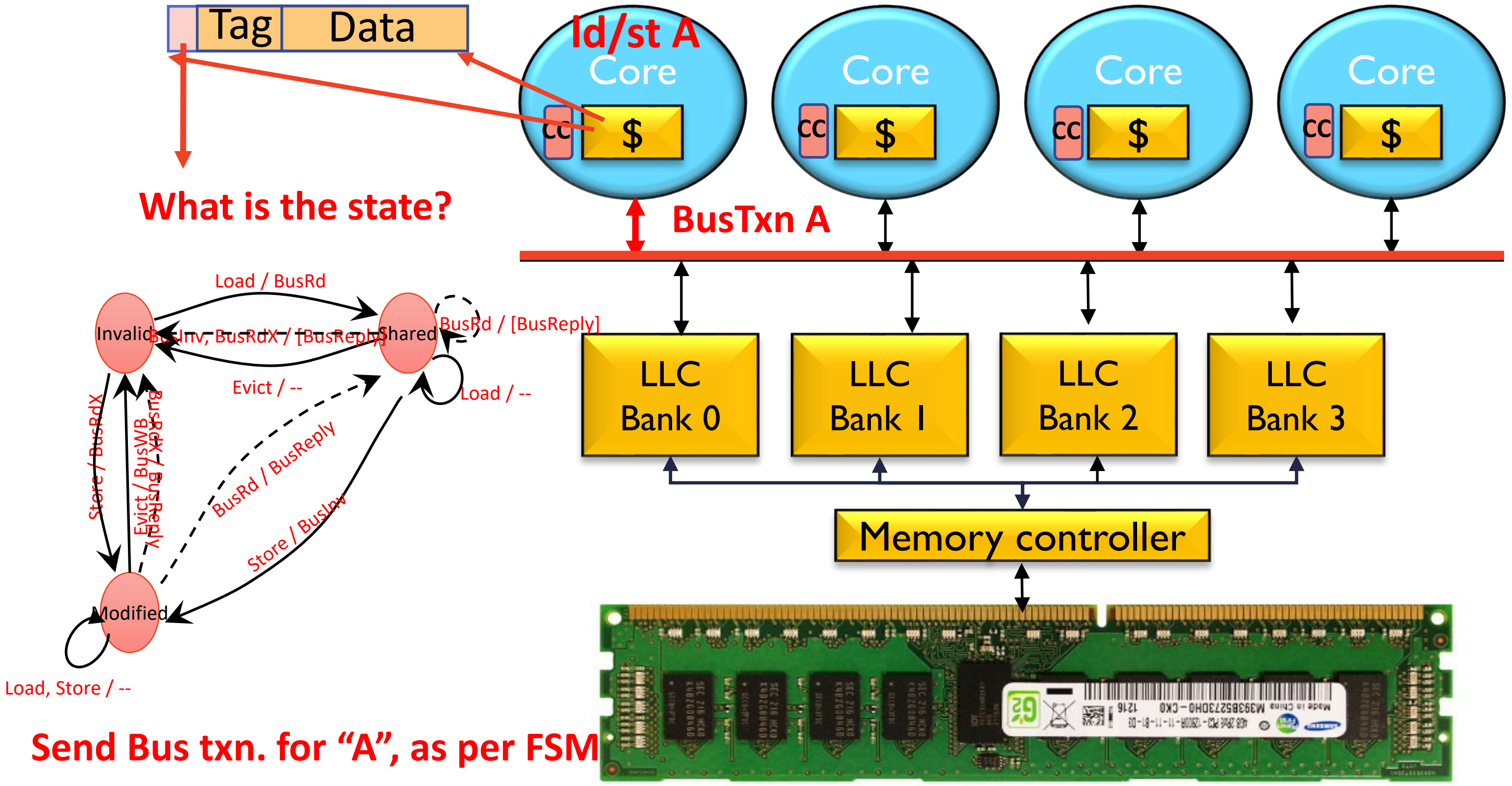
$ A (if present, "I" state otherwise)



What is the state?

Example: MSI coherence protocol

# Putting it together: Coherence



$ A (if present, "I" state otherwise)

What is the state?

Send Bus txn. for "A", as per FSM

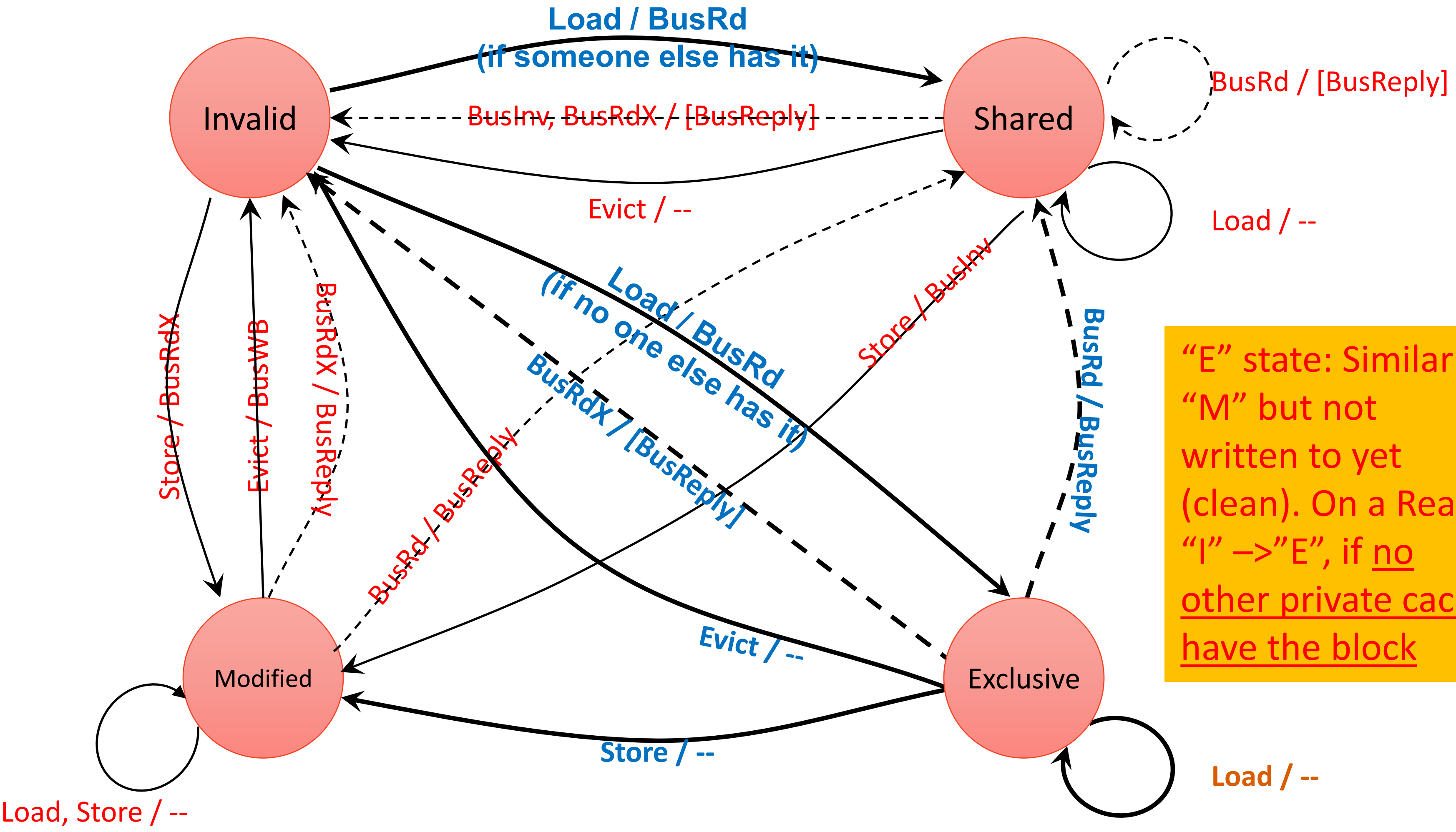
Example: MSI coherence protocol

# Many possible protocols

- MSI protocol is not the only possibility
  - It’s a basic protocol

- MESI protocol optimizes for access pattern where the same data is read and then immediately written by a thread

# FSM for MESI protocol



"E" state: Similar to "M" but not written to yet (clean). On a Read, "I" –>"E", if no other private cache have the block

# Many possible protocols

- MSI protocol is not the only possibility
  - It’s a basic protocol

- MESI protocol optimizes for access pattern where the same data is read and then immediately written by a thread

- MOESI further optimizes for producer-consumer access pattern in applications

# Take away

- Multicores: Multiple replicated cores, along with own private caches that typically shares a larger LLC
- Cache coherence protocols keeps the private caches coherent
- Many possible ways to implement coherence

# Directory coherence protocols

Scalable cache coherence

# Problems w/ Snoopy Coherence

1) Interconnect bandwidth
   - Problem: Bus and Ring are not scalable interconnects
     - Limited bandwidth
     - Cannot support more than a dozen or so processors
   - Solution: Replace non-scalable interconnect (ring or bus) with a scalable one (e.g., mesh)

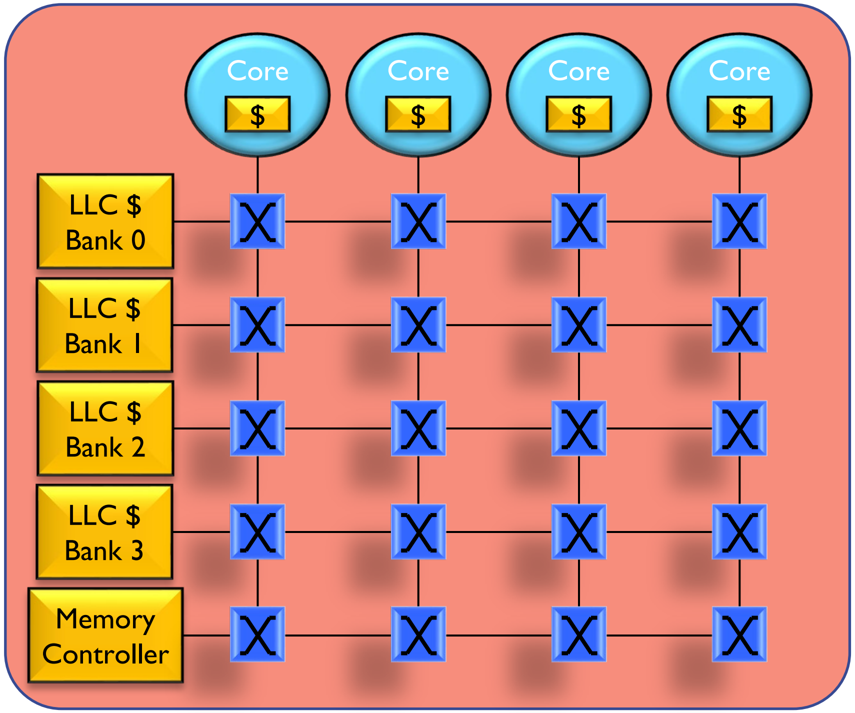
# Problems w/ Snoopy Coherence

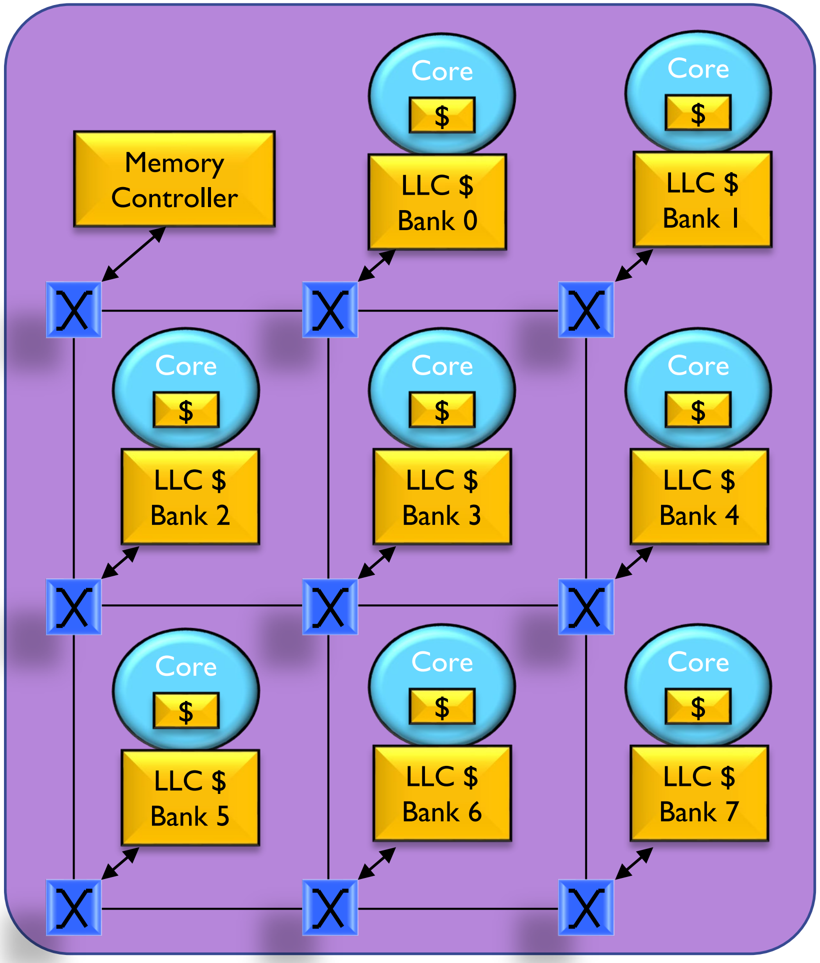
1) Interconnect bandwidth
   - Problem: Bus and Ring are not scalable interconnects
     - Limited bandwidth
     - Cannot support more than a dozen or so processors
   - Solution: Replace non-scalable interconnect (ring or bus) with a scalable one (e.g., mesh)

2) Cache snooping bandwidth
   - Problem: All caches must monitor all bus traffic; most snoops result in no action
   - Solution: Replace non-scalable broadcast protocol (spam everyone) with scalable directory protocol (notify cores that care)
     - The “directory” keeps track of “sharers”

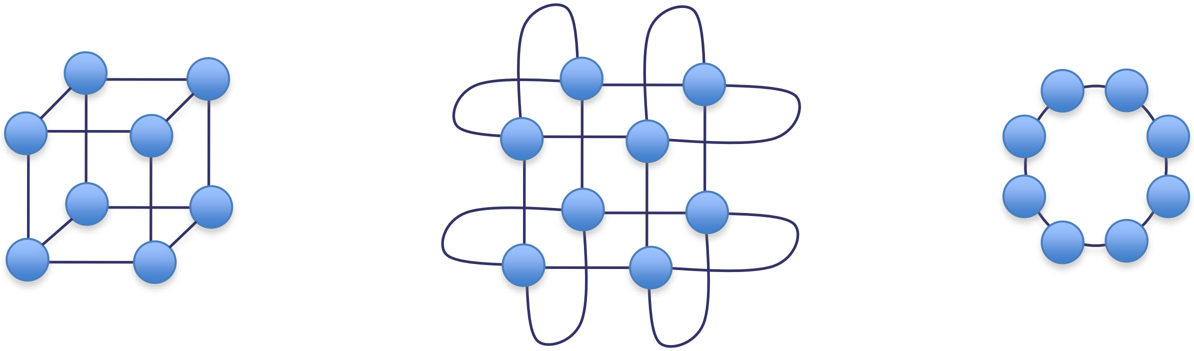
# Directory Coherence Protocols

- Typically use point-to-point scalable networks
  - Such as Crossbar or Mesh





# Issues with point-to-point networks



Advantages:

- better electrical behavior (shorter wires)
- coherence transaction parallelism

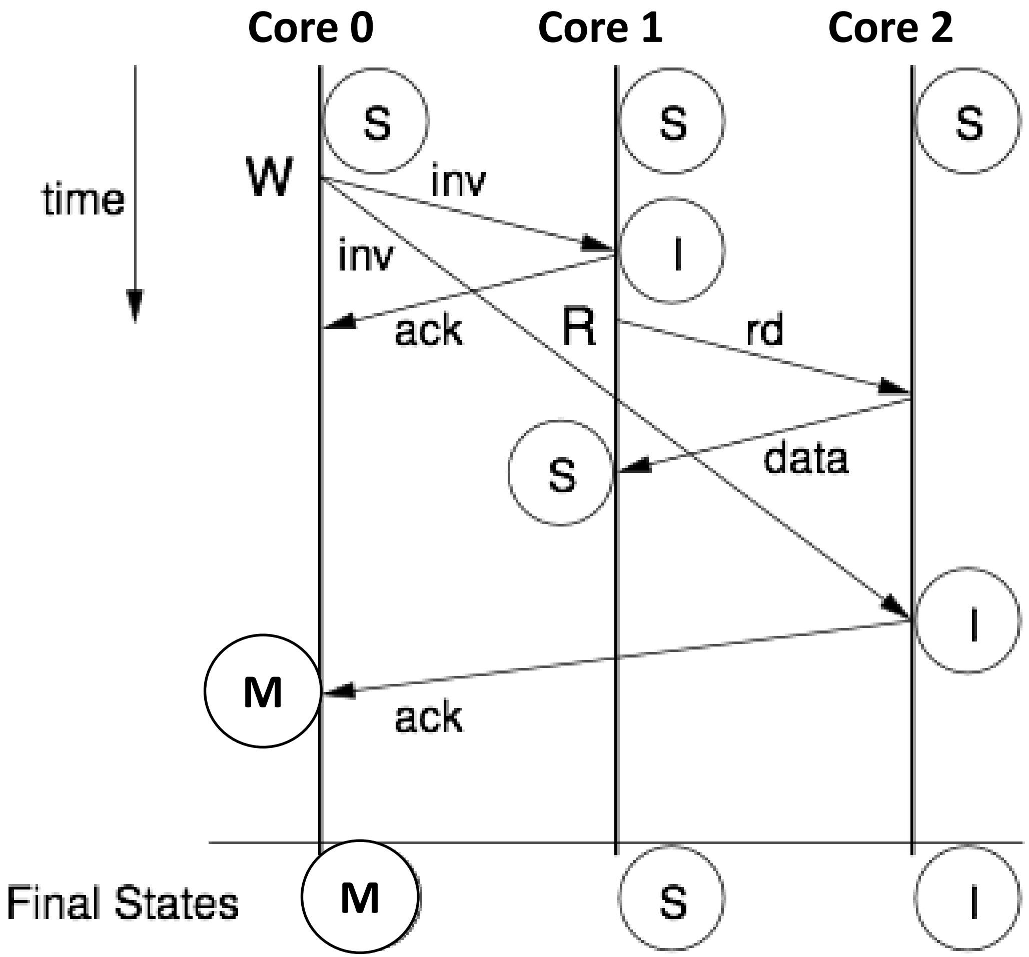
Problem: unordered network

Nodes may observe messages in different orders

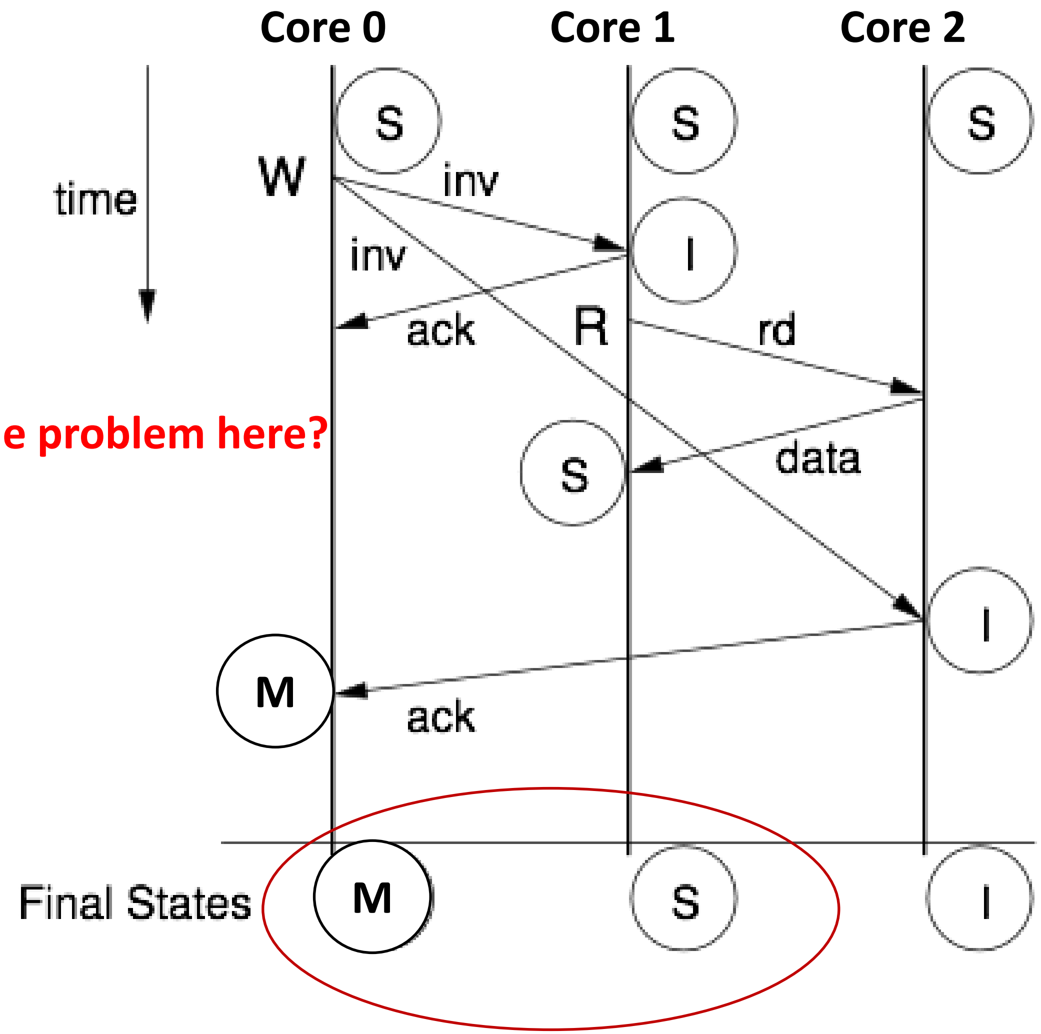
Is this a problem?

(May break the write propagation constraint)

# Ordering – What is wrong here?

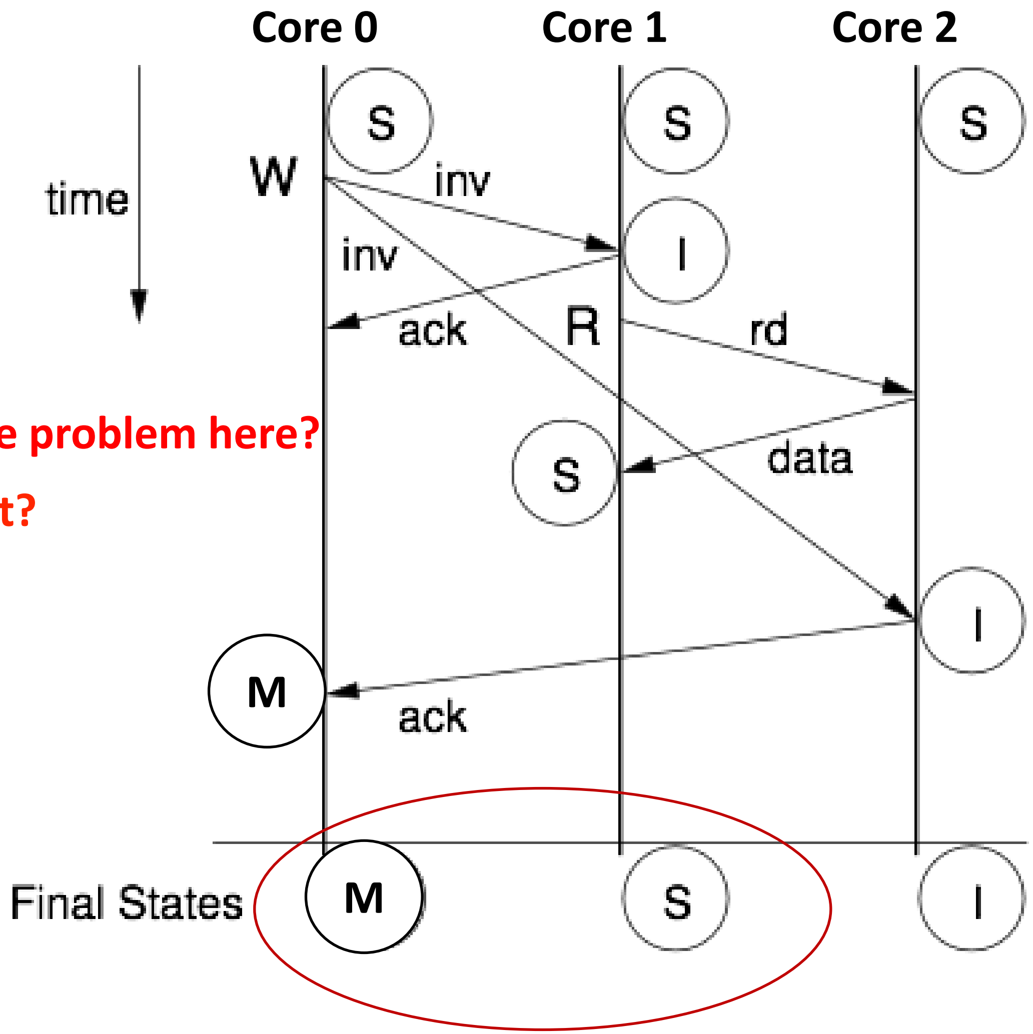


# Ordering – What is wrong here?



What caused the problem here?

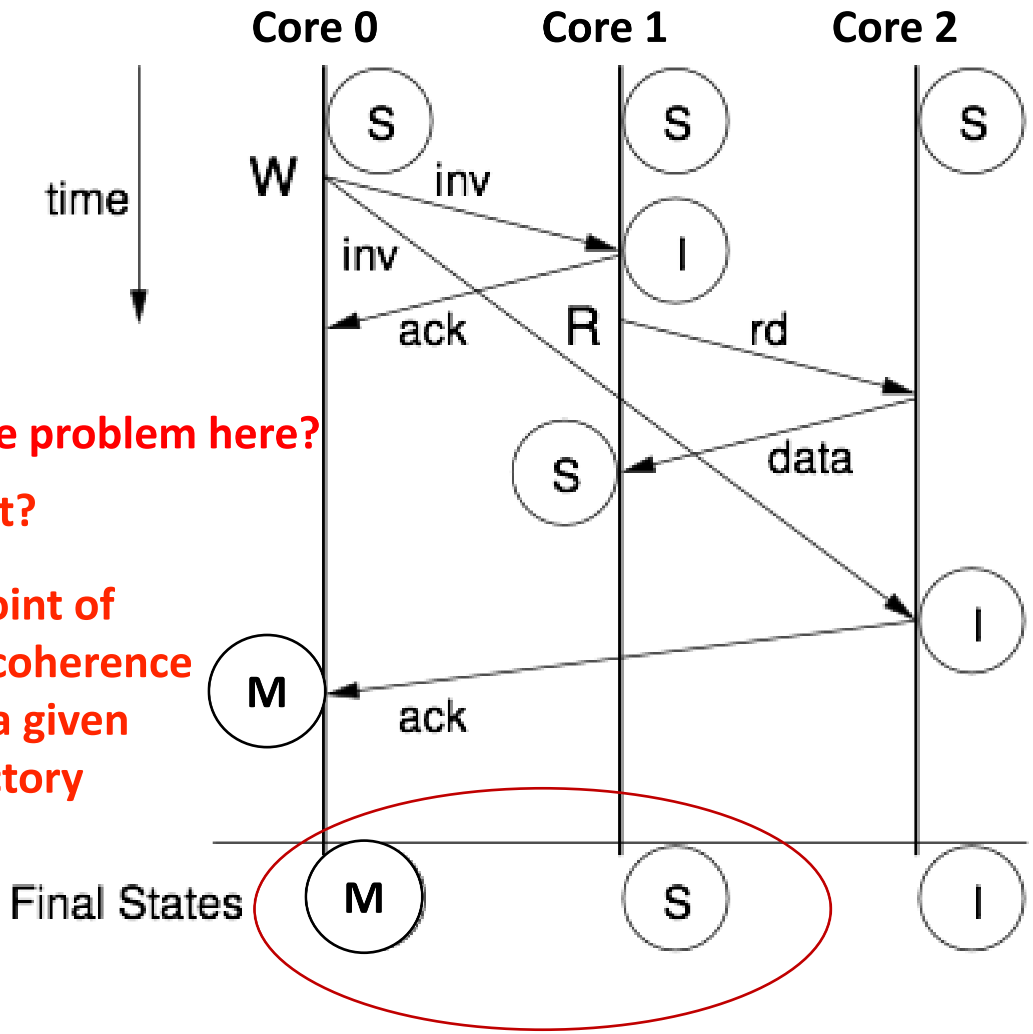
# Ordering – What is wrong here?



What caused the problem here?

How do we fix it?

# Ordering – What is wrong here?



What caused the problem here?

How do we fix it?

Need a single point of ordering for all coherence transactions to a given address → directory

# Directory Coherence Protocols

- Each physical cache line has a ***home node/core/contoller***
- Extend memory controller (or LLC bank) to track caching information for cache lines for which it is home
  - Information kept in a hardware structure called ***Directory***

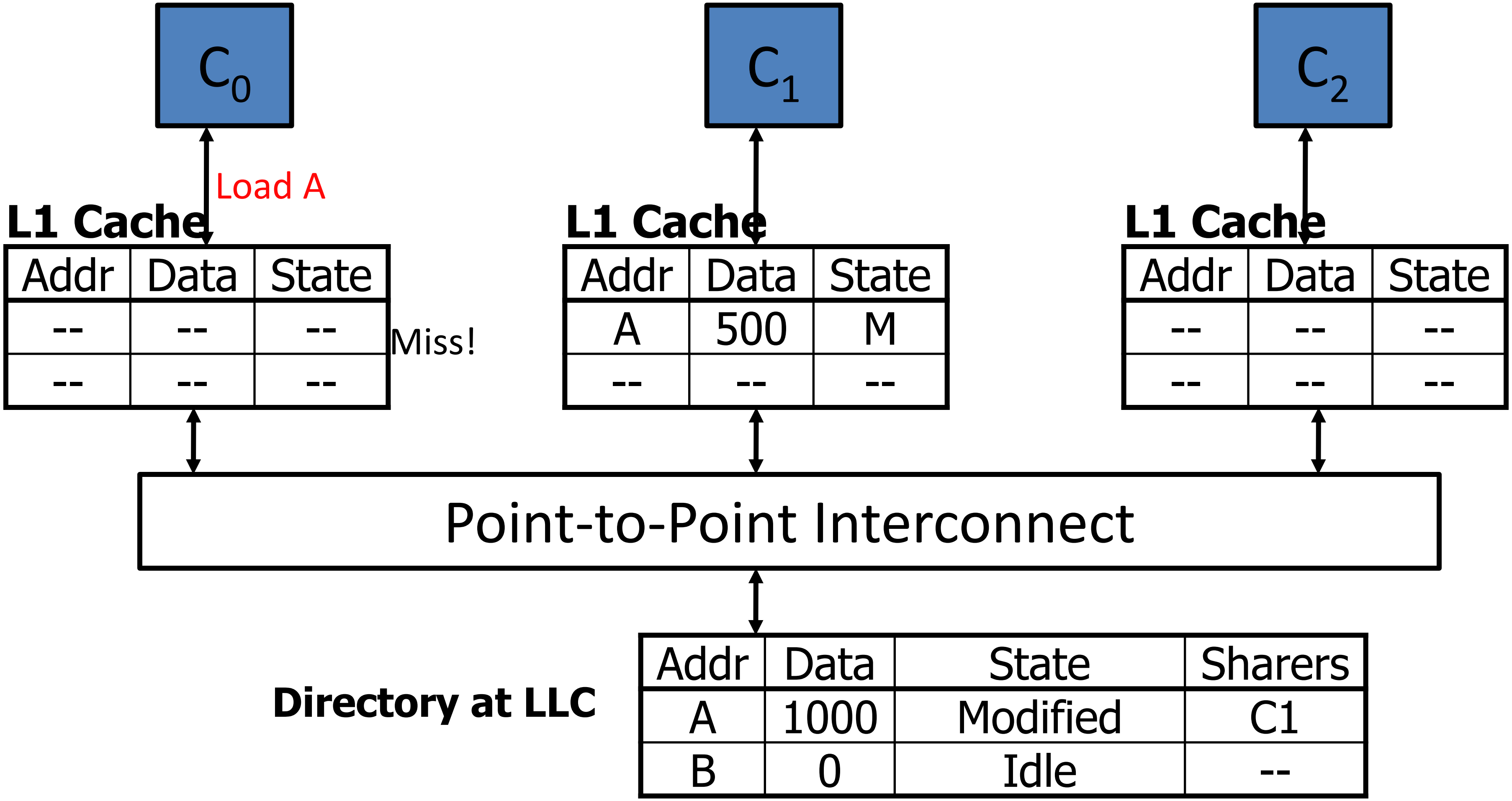
# Directory Coherence Protocols

- Each physical cache line has a ***home node/core/contoller***
- Extend memory controller (or LLC bank) to track caching information for cache lines for which it is home
  - Information kept in a hardware structure called ***Directory***
- For each physical cache line, a ***home directory*** tracks:
  - ***Owner***: core that has a dirty copy (i.e., M state)
  - ***Sharers***: cores that have clean copies (i.e., S state)

# Directory Coherence Protocols

- Each physical cache line has a ***home node/core/contoller***
- Extend memory controller (or LLC bank) to track caching information for cache lines for which it is home
  - Information kept in a hardware structure called ***Directory***
- For each physical cache line, a ***home directory*** tracks:
  - ***Owner***: core that has a dirty copy (i.e., M state)
  - ***Sharers***: cores that have clean copies (i.e., S state)
- Cores send coherence requests to home directory
- Home directory forwards messages only to cores that "care" (i.e., cores that might have a copy of the line)

# MSI Directory Example: Step #1



$C_0$

Load A

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

Miss!

$C_1$

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | 500 | M |
| -- | -- | -- |

$C_2$

**L1 Cache**

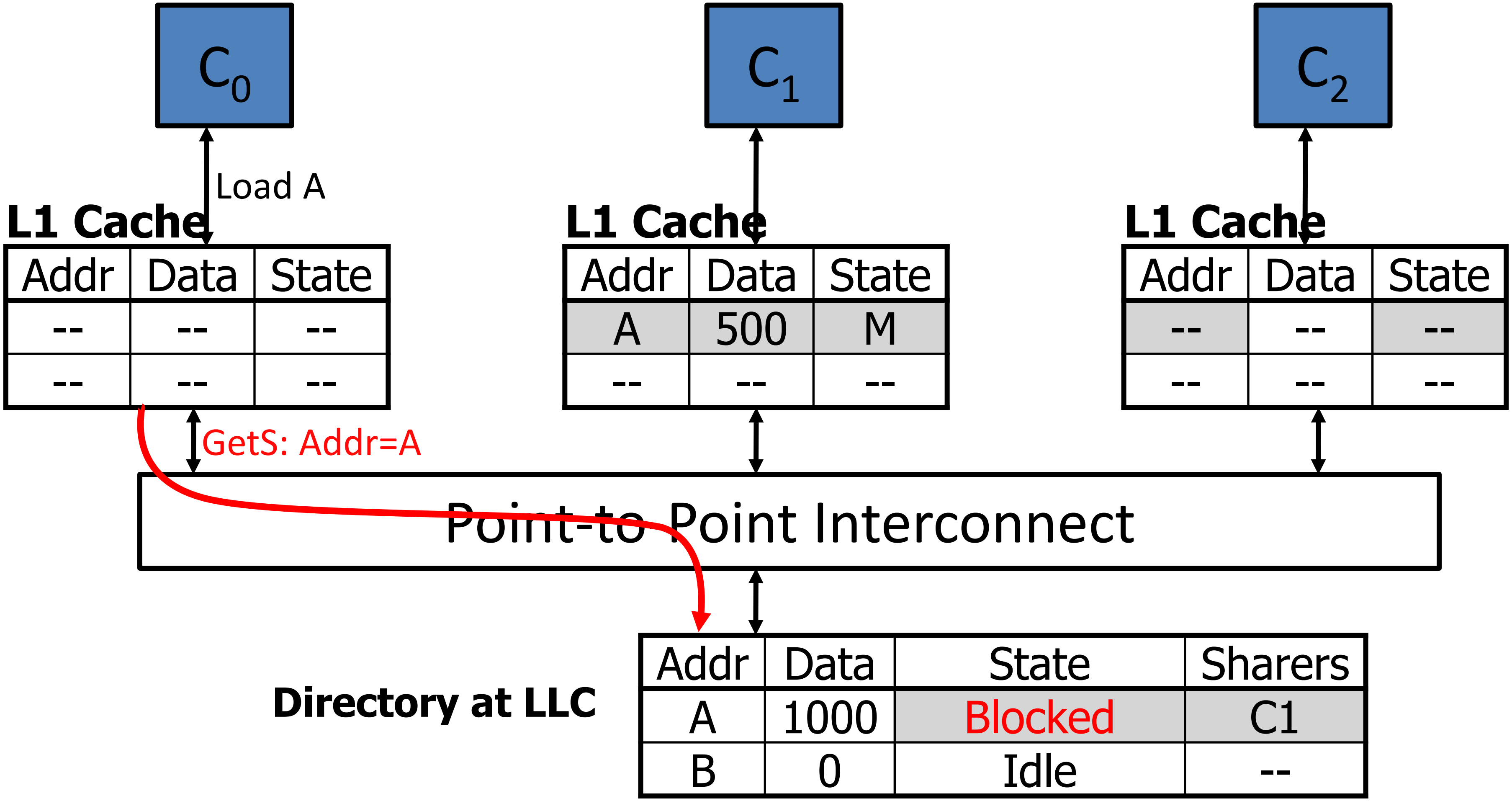
| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

Point-to-Point Interconnect

**Directory at LLC**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 1000 | Modified | C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #2



L1 Cache ($C_0$)

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

L1 Cache ($C_1$)

| Addr | Data | State |
|---|---|---|
| A | 500 | M |
| -- | -- | -- |

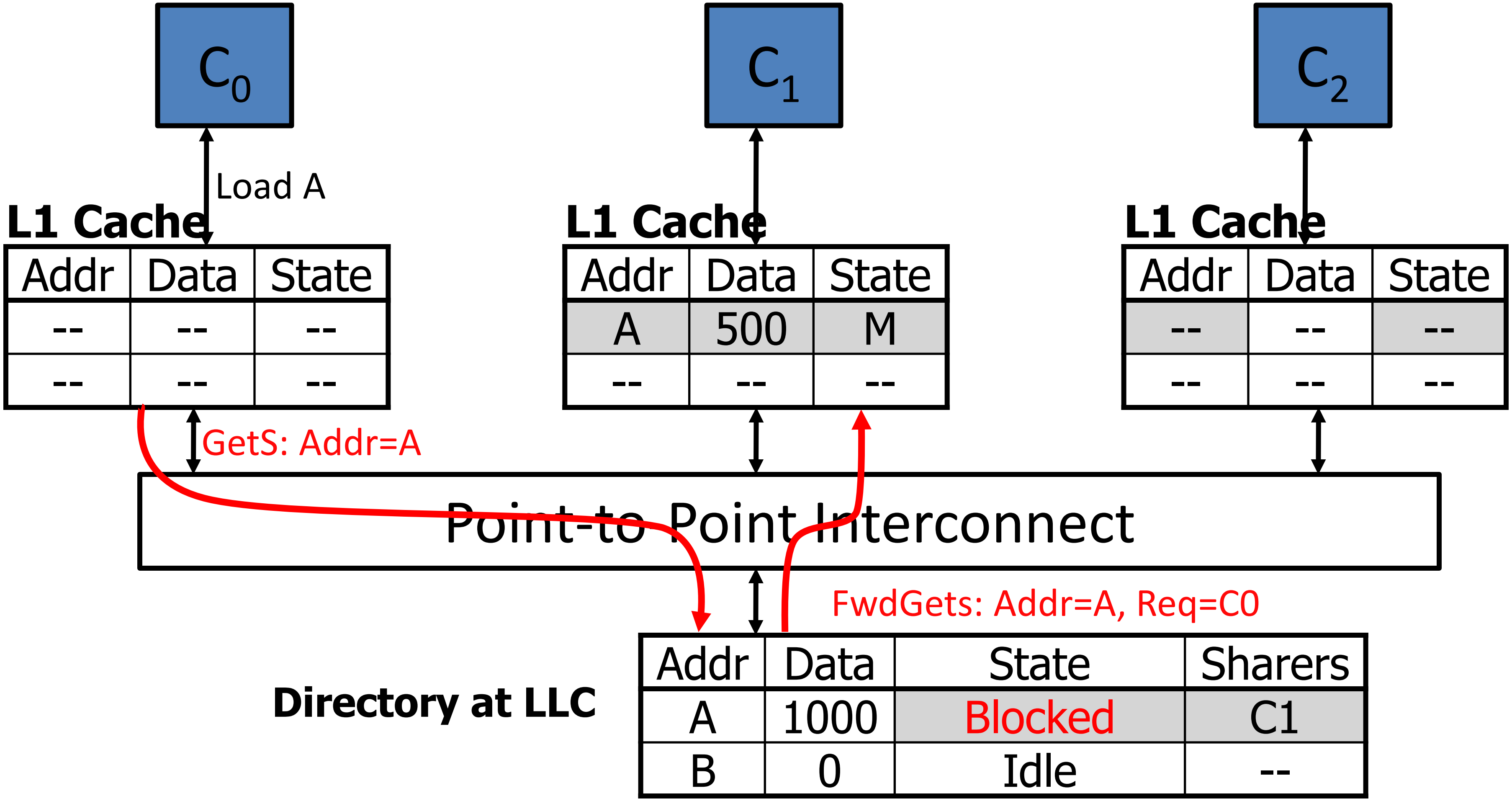
L1 Cache ($C_2$)

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

Directory at LLC

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 1000 | Blocked | C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #2



$C_0$

Load A

L1 Cache

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

$C_1$

L1 Cache

| Addr | Data | State |
|---|---|---|
| A | 500 | M |
| -- | -- | -- |

$C_2$

L1 Cache

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

GetS: Addr=A

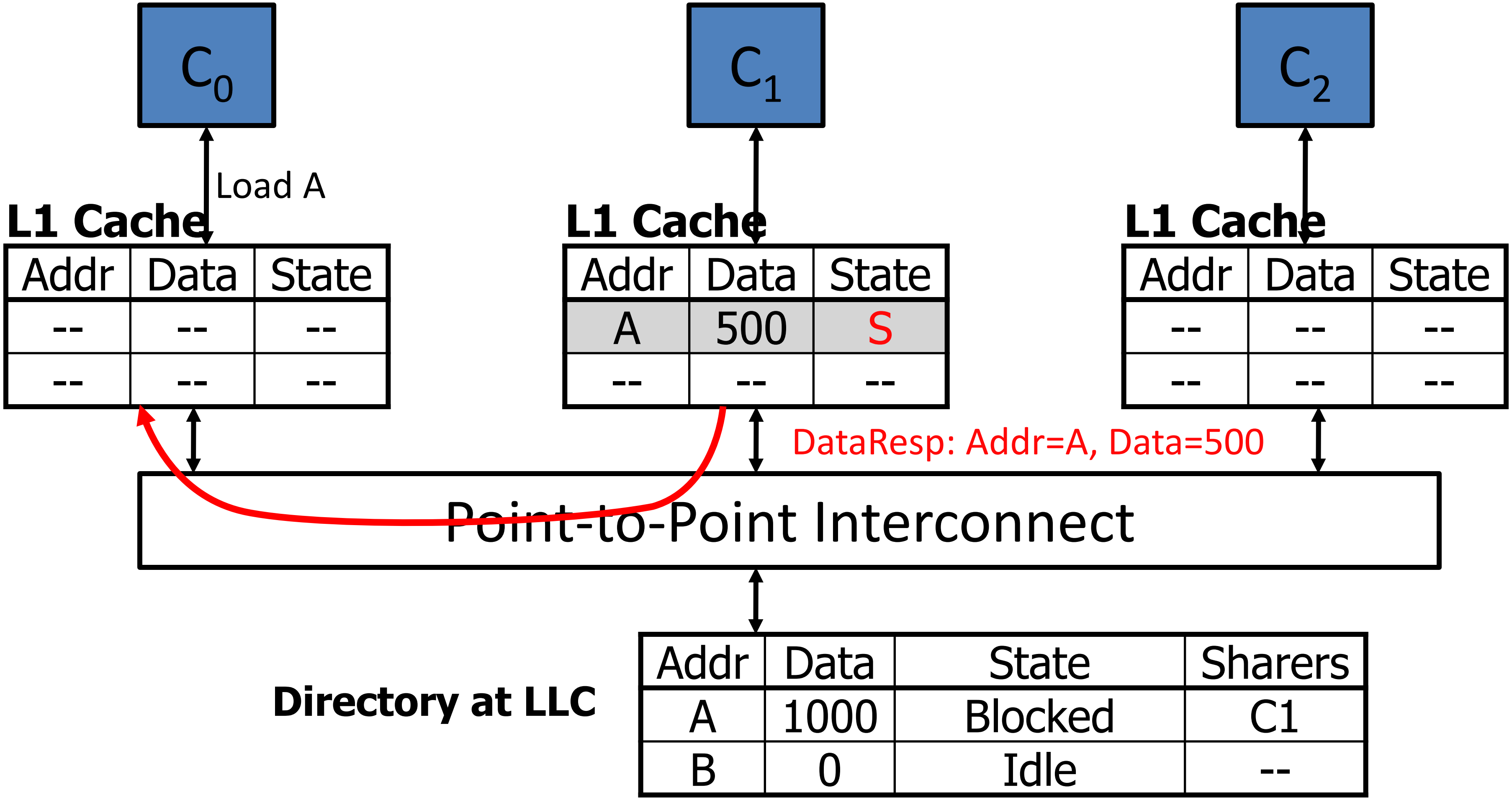
Point-to-Point Interconnect

FwdGets: Addr=A, Req=C0

Directory at LLC

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 1000 | Blocked | C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #3



DataResp: Addr=A, Data=500

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

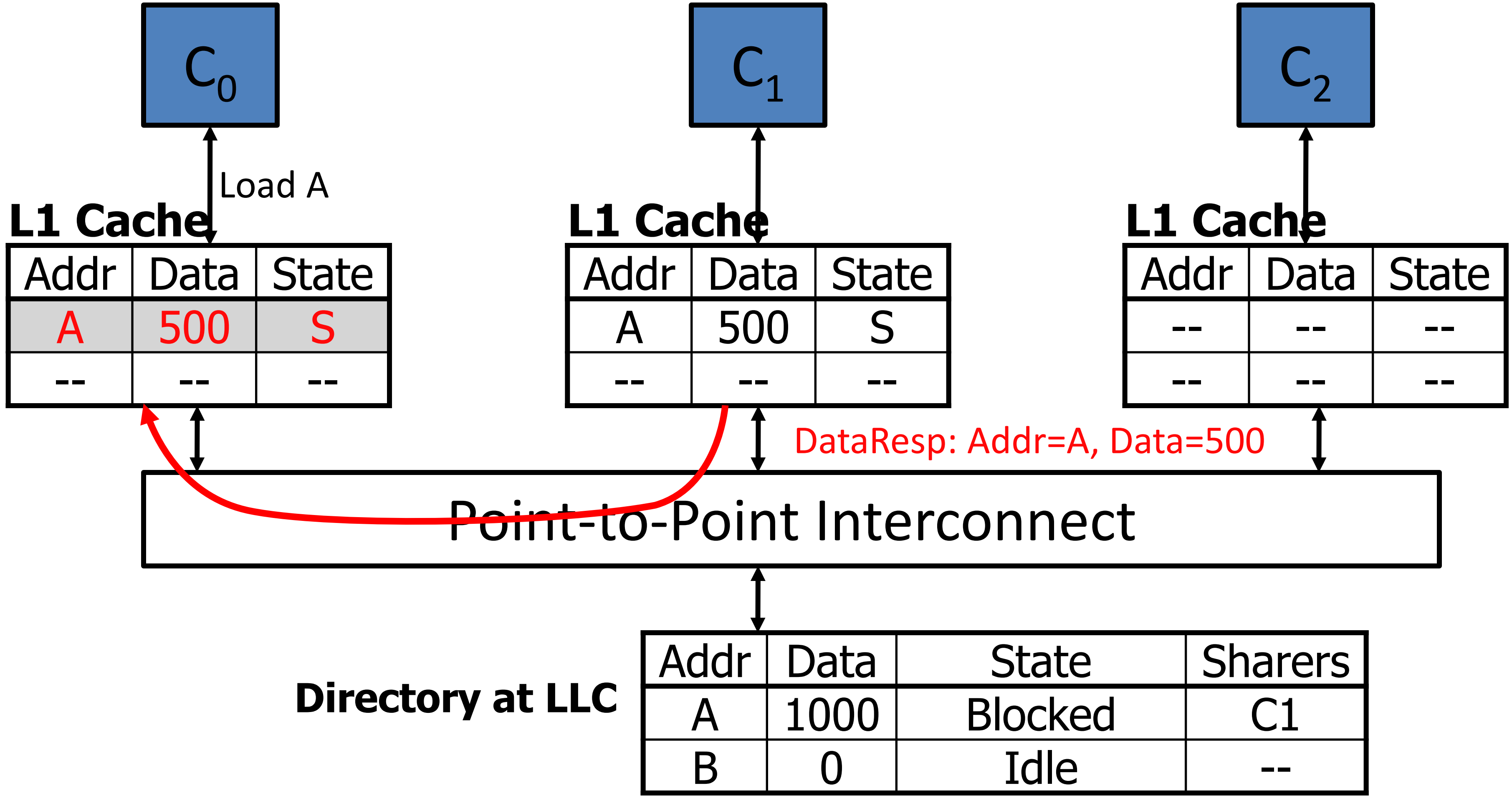
**L1 Cache**

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

**Directory at LLC**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 1000 | Blocked | C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #4



$C_0$

Load A

L1 Cache

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

$C_1$

L1 Cache

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

$C_2$

L1 Cache

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

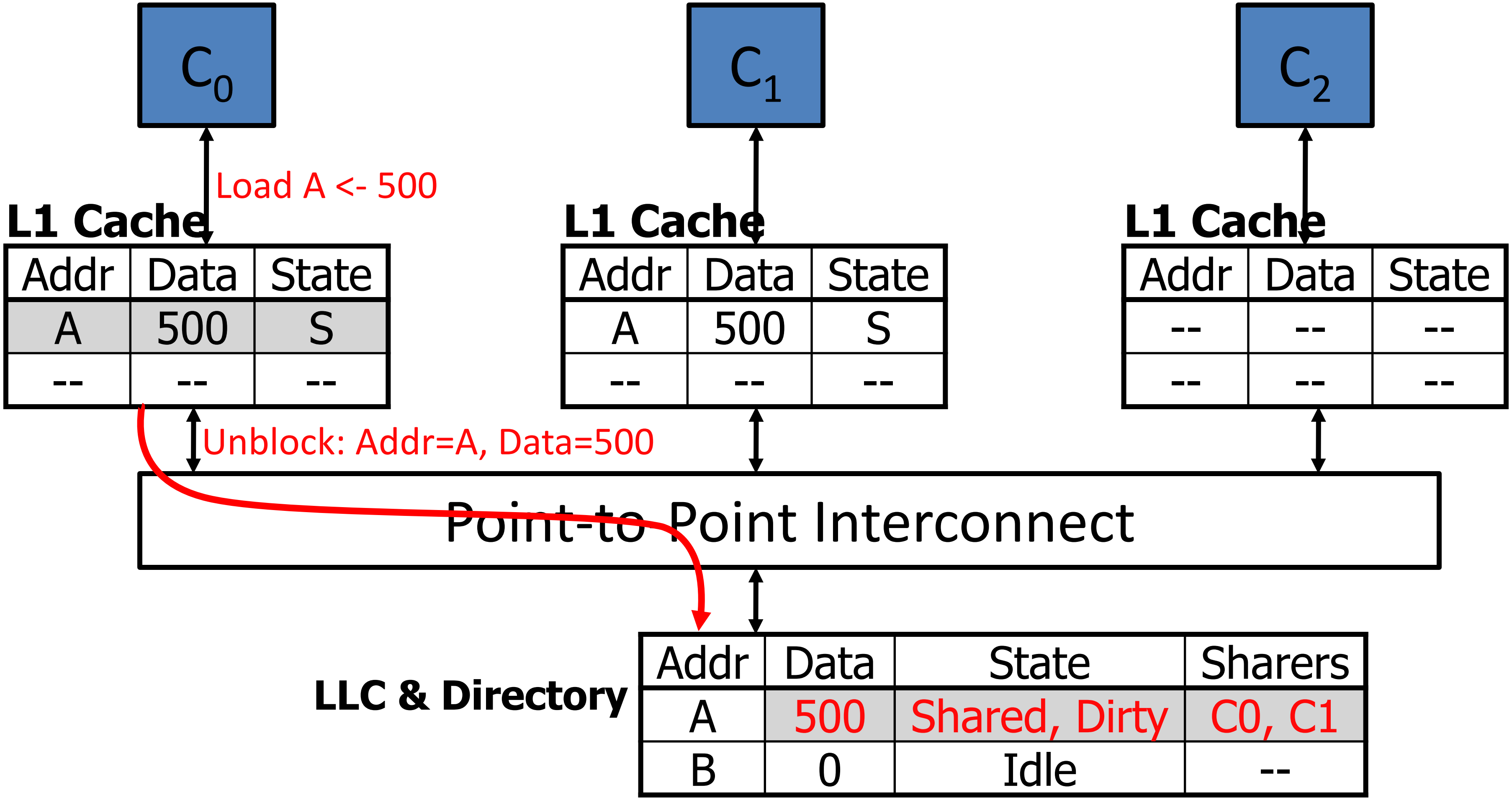
DataResp: Addr=A, Data=500

Point-to-Point Interconnect

Directory at LLC

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 1000 | Blocked | C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #5



**L1 Cache** ($C_0$)

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

**L1 Cache** ($C_1$)

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

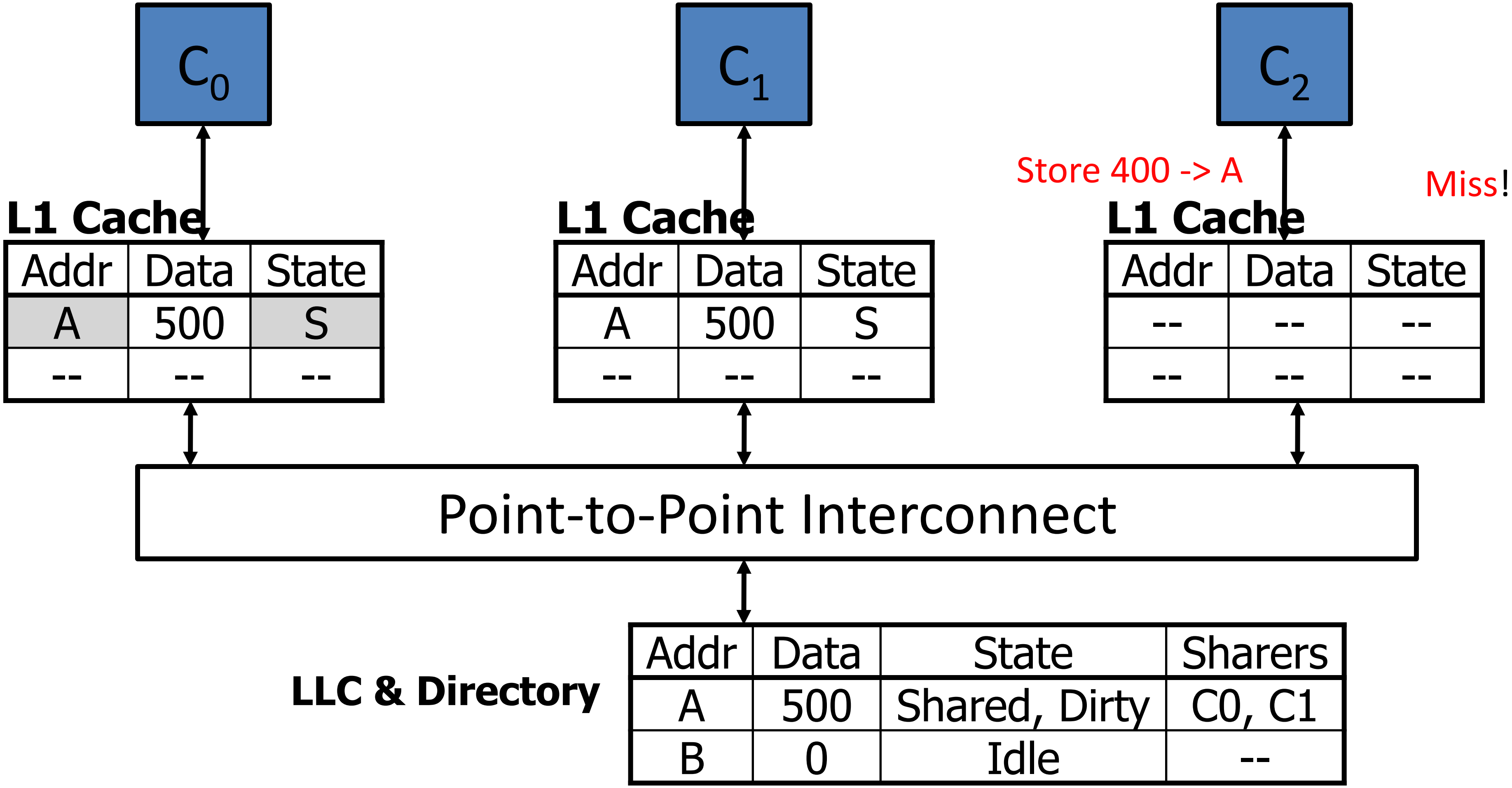
**L1 Cache** ($C_2$)

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

**LLC & Directory**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Shared, Dirty | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #6



L1 Cache

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

L1 Cache

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

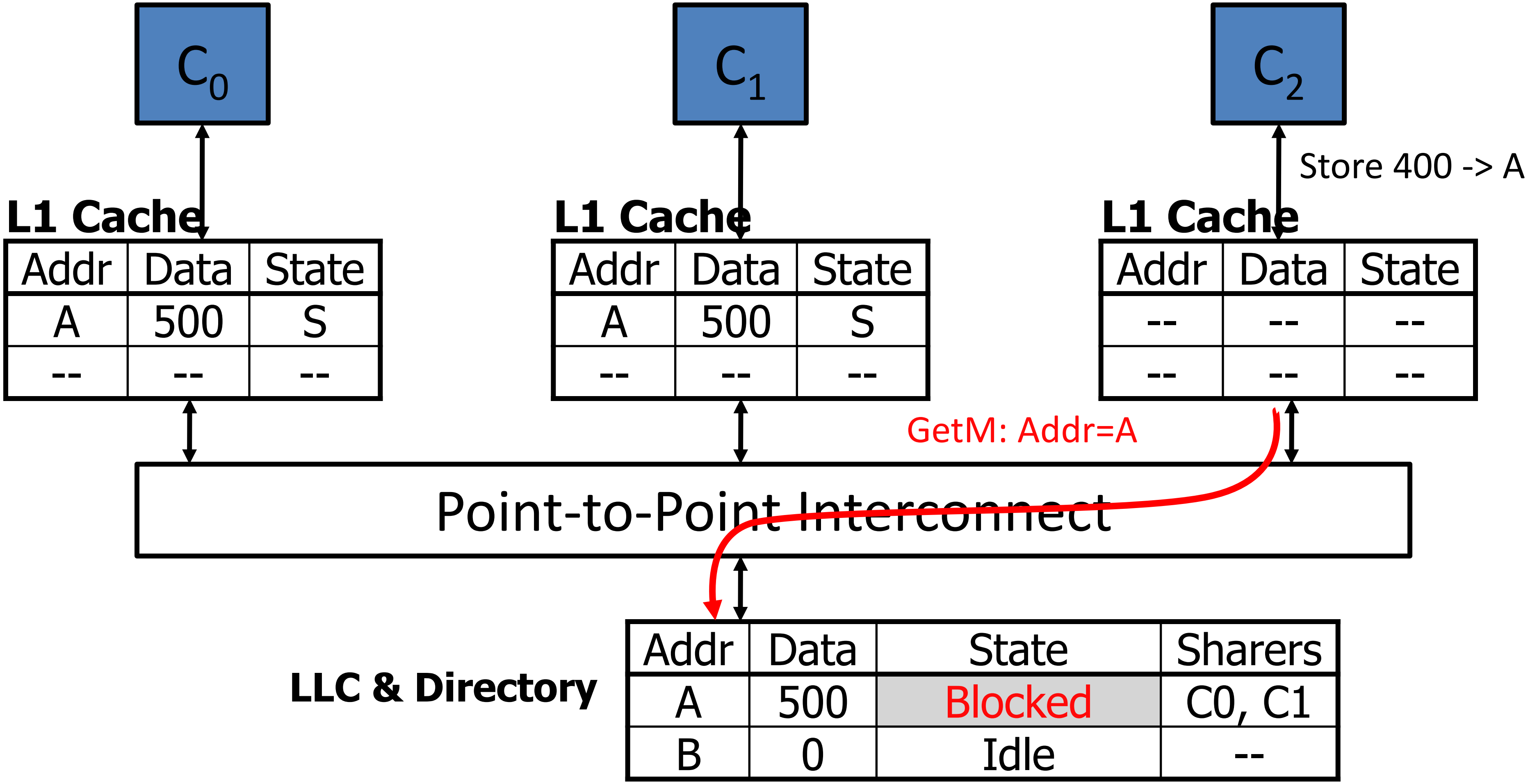
L1 Cache

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

**LLC & Directory**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Shared, Dirty | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #7



L1 Cache (C0)

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

L1 Cache (C1)

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

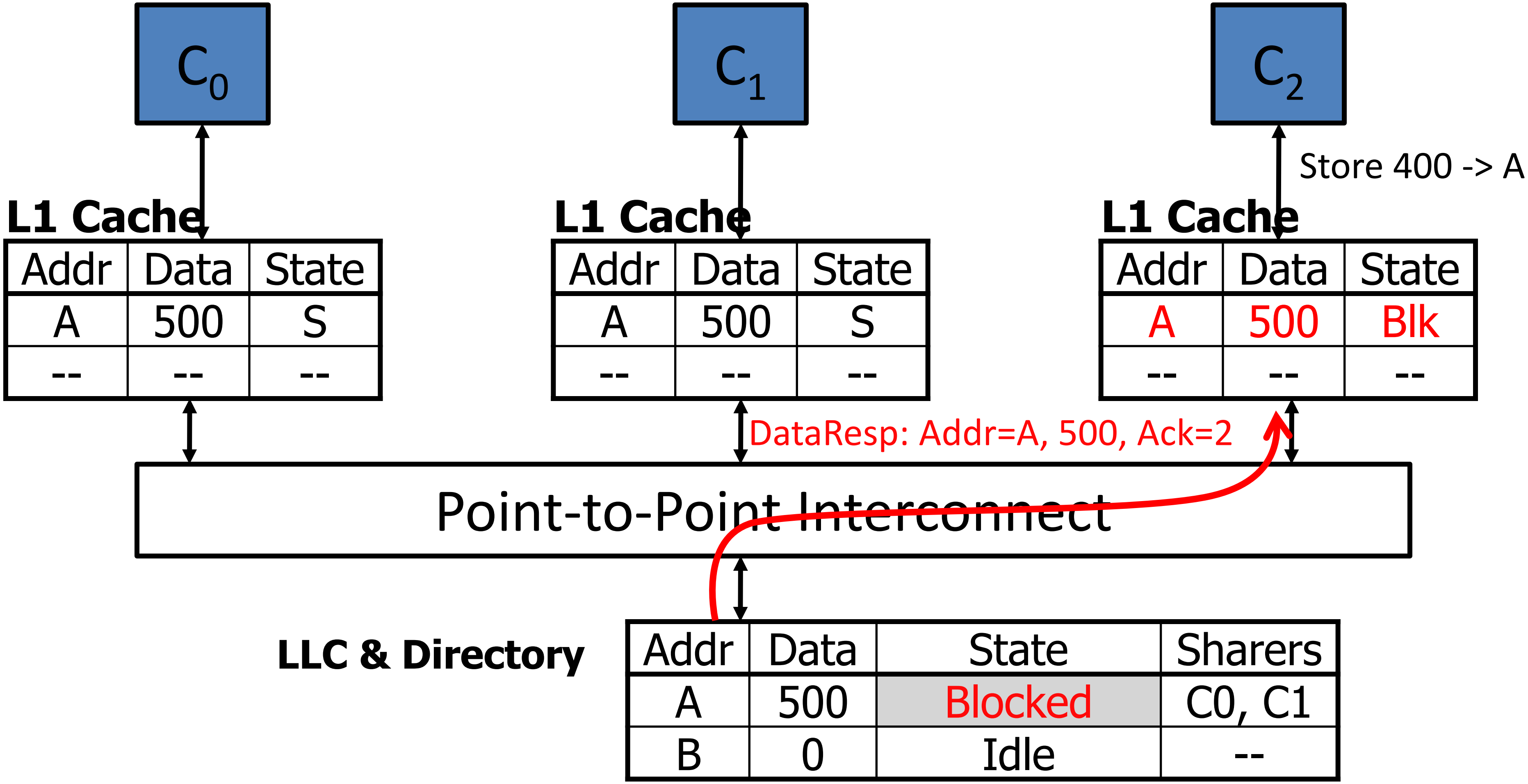
L1 Cache (C2)

| Addr | Data | State |
|---|---|---|
| -- | -- | -- |
| -- | -- | -- |

**LLC & Directory**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Blocked | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #8



$C_0$

$C_1$

$C_2$

Store 400 -> A

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | 500 | Blk |
| -- | -- | -- |

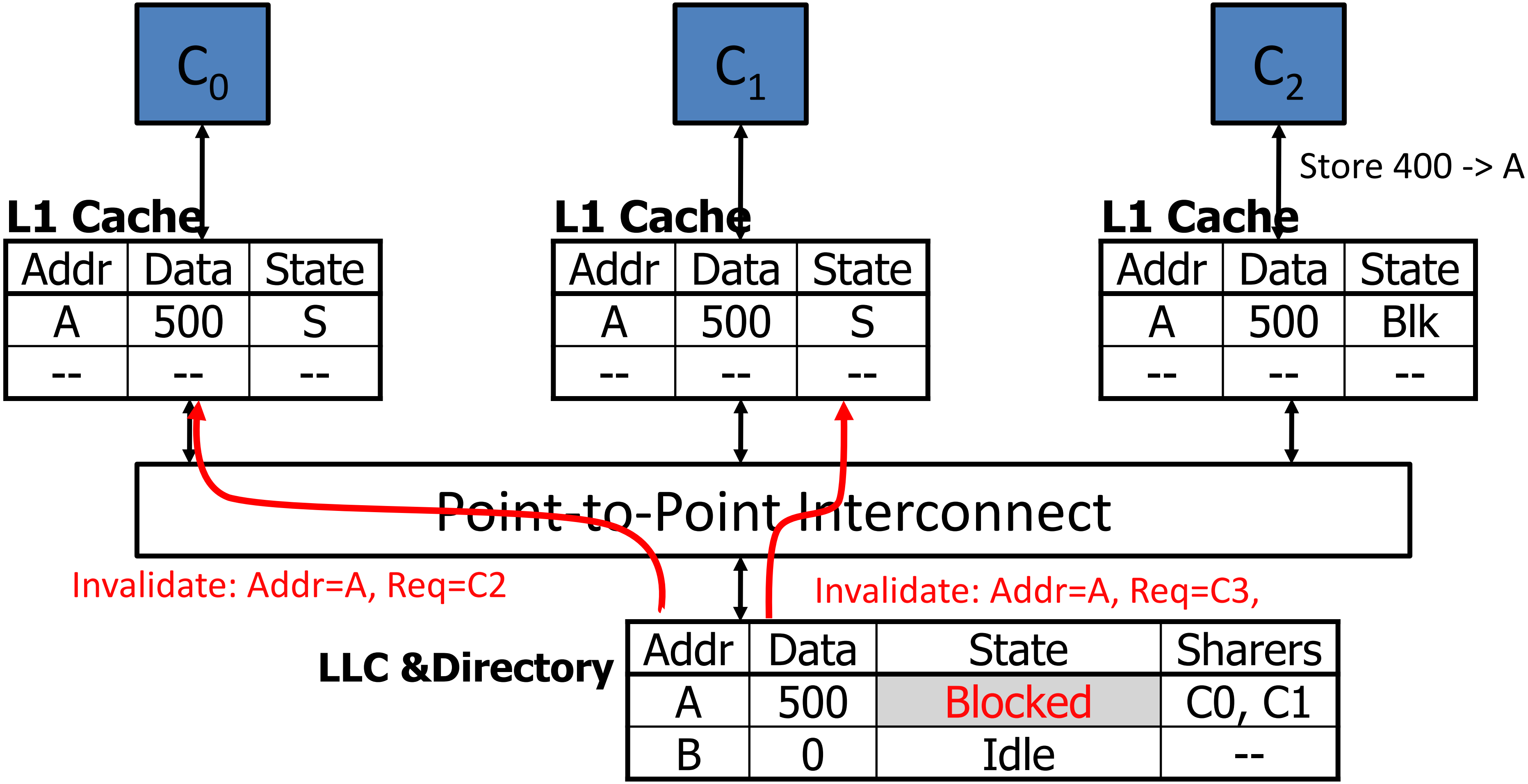
DataResp: Addr=A, 500, Ack=2

Point-to-Point Interconnect

**LLC & Directory**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Blocked | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #9



**L1 Cache** ($C_0$)

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

**L1 Cache** ($C_1$)

| Addr | Data | State |
|---|---|---|
| A | 500 | S |
| -- | -- | -- |

**L1 Cache** ($C_2$)

| Addr | Data | State |
|---|---|---|
| A | 500 | Blk |
| -- | -- | -- |

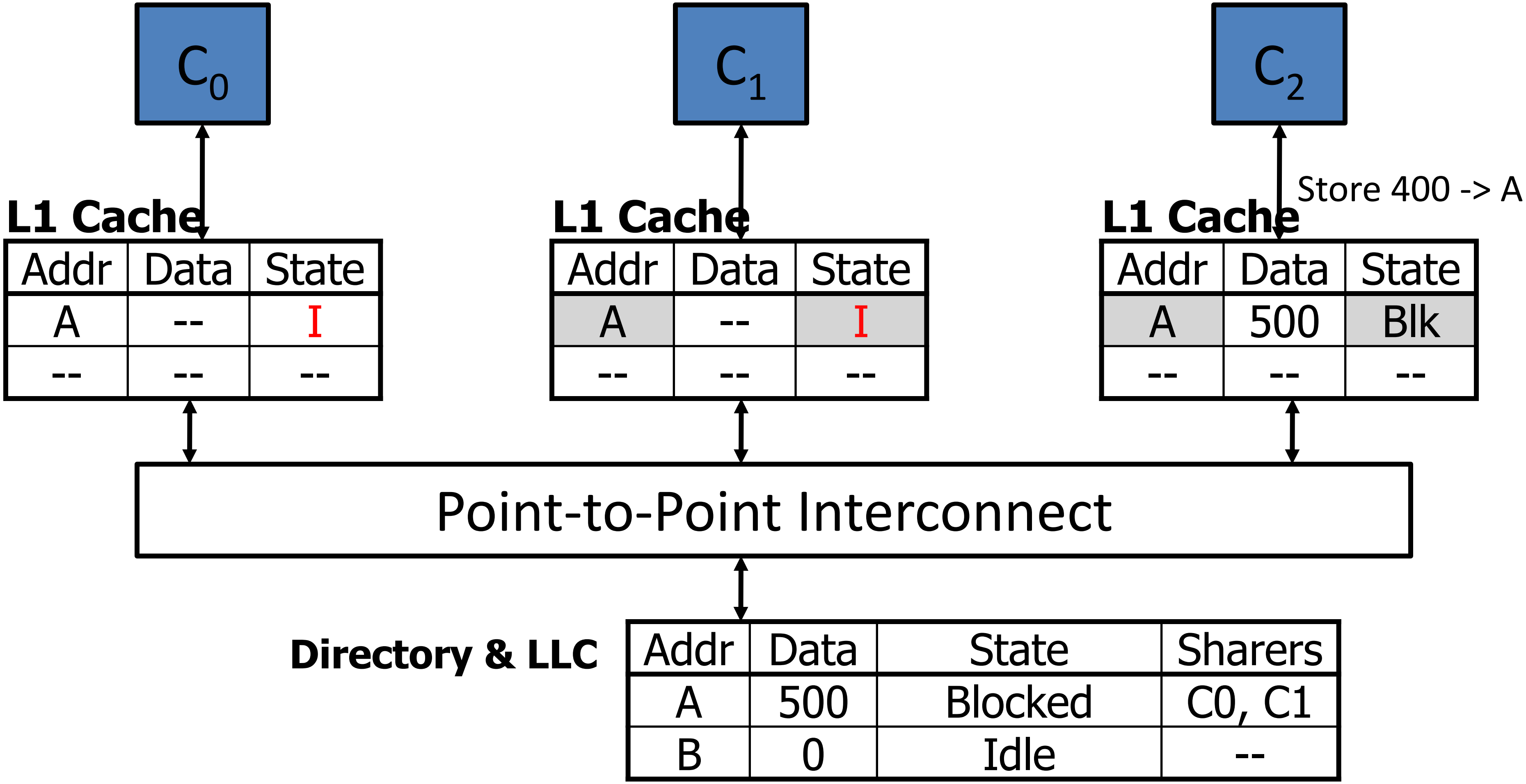
Invalidate: Addr=A, Req=C2

Invalidate: Addr=A, Req=C3,

**LLC &Directory**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Blocked | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #10



$C_0$

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

$C_1$

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

$C_2$

Store 400 -> A

**L1 Cache**

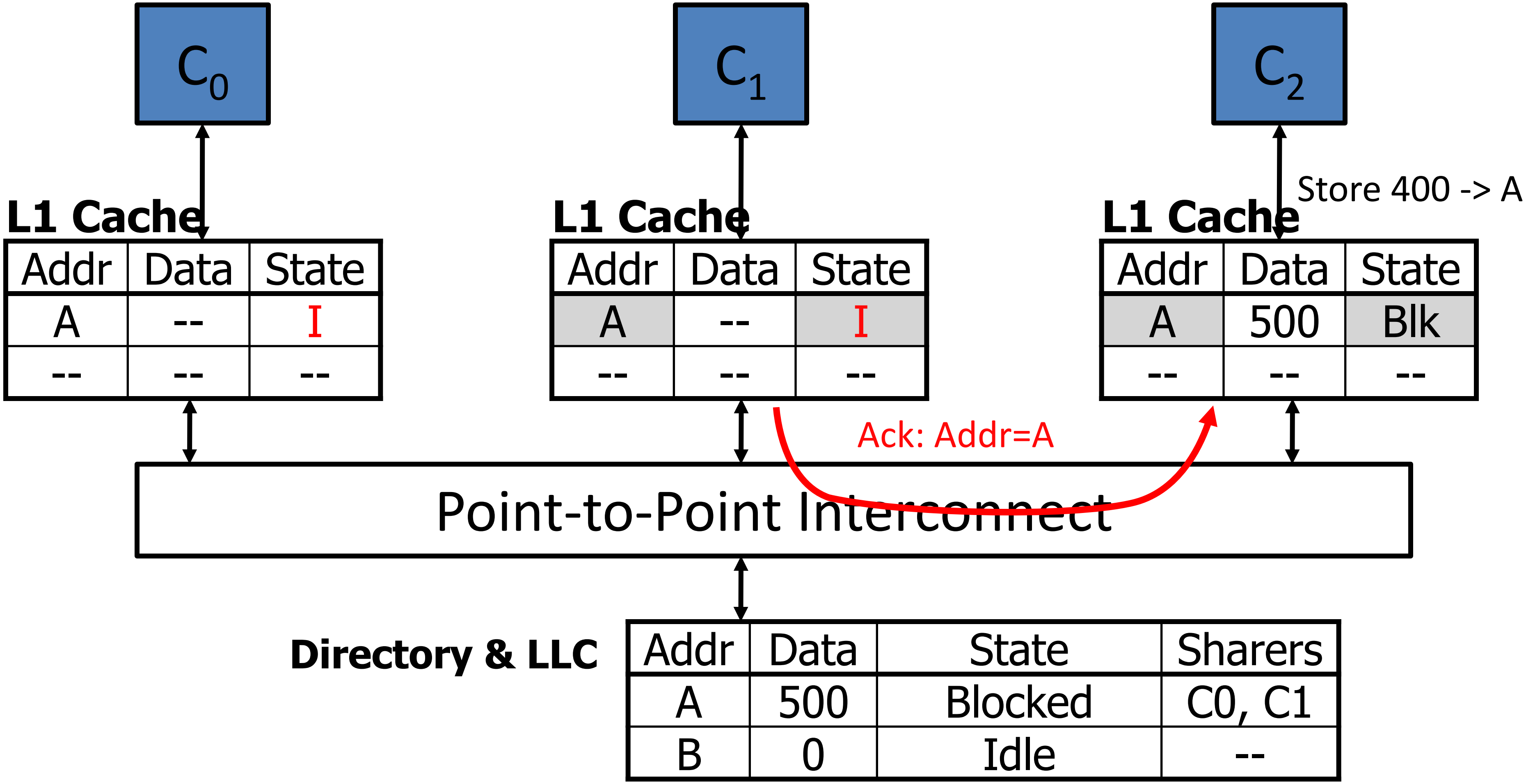
| Addr | Data | State |
|---|---|---|
| A | 500 | Blk |
| -- | -- | -- |

Point-to-Point Interconnect

**Directory & LLC**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Blocked | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #10



L1 Cache ($C_0$)

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

L1 Cache ($C_1$)

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

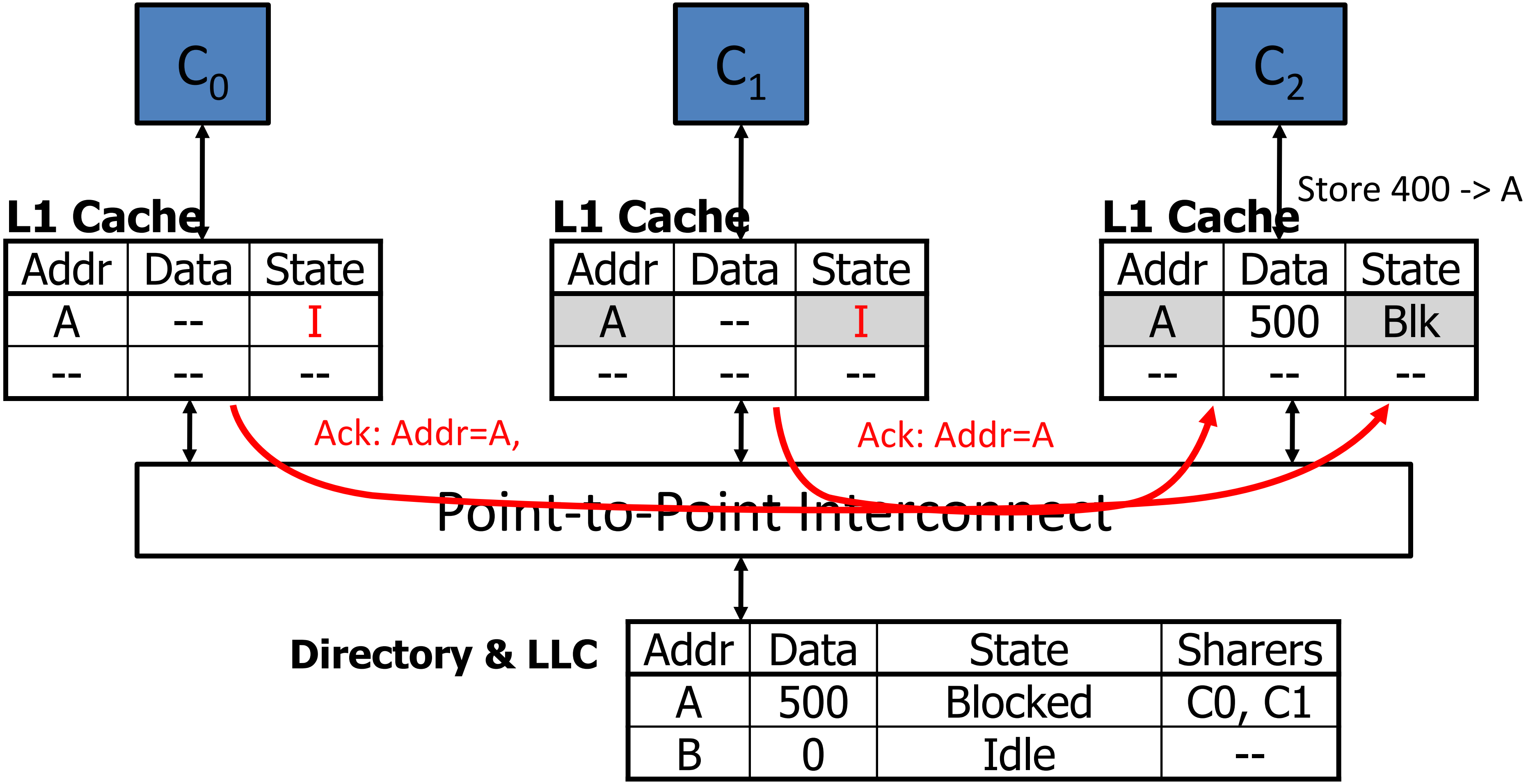
L1 Cache ($C_2$)

| Addr | Data | State |
|---|---|---|
| A | 500 | Blk |
| -- | -- | -- |

Directory & LLC

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Blocked | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #10



**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

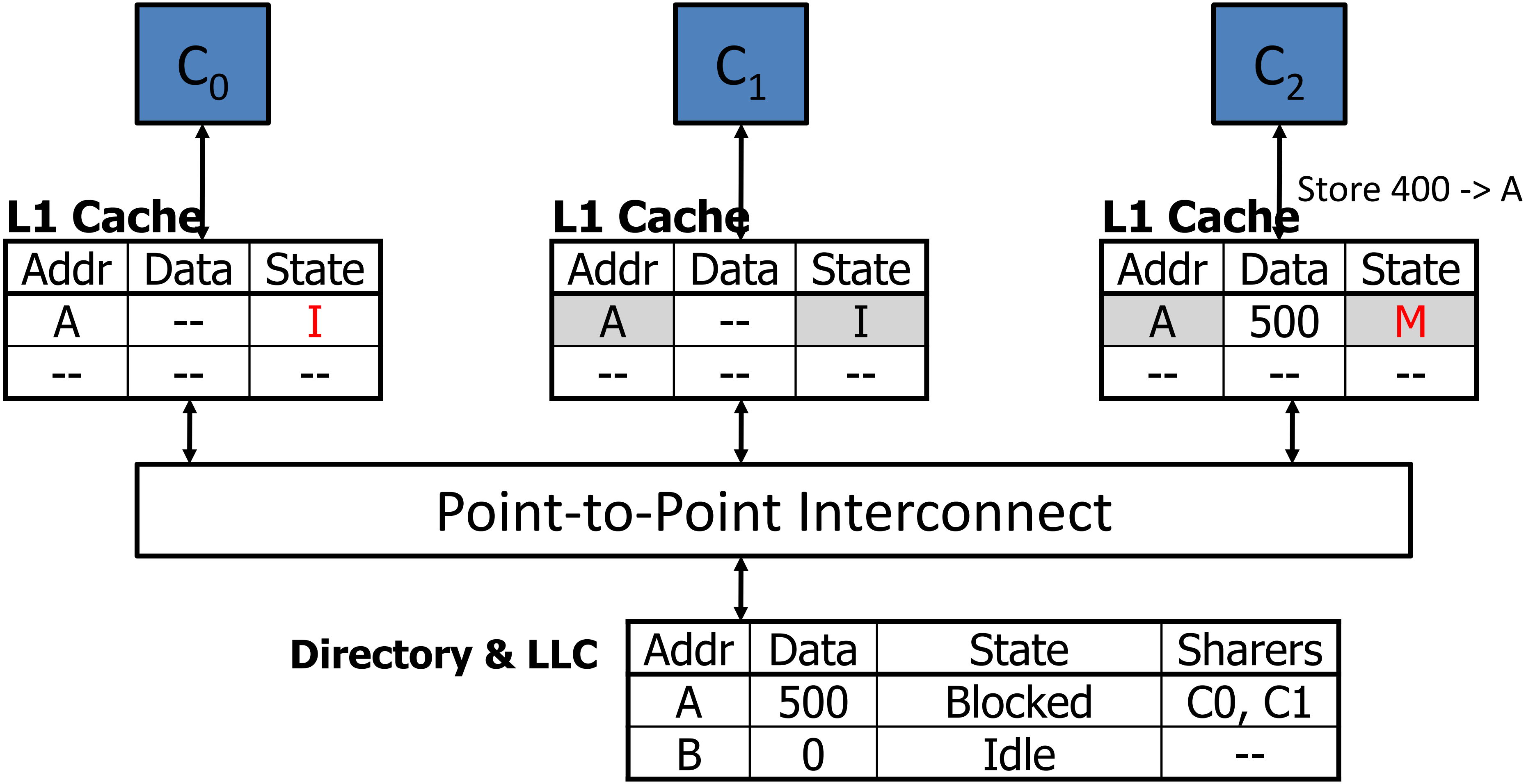
**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | 500 | Blk |
| -- | -- | -- |

**Directory & LLC**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Blocked | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #11



**L1 Cache** ($C_0$)

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

**L1 Cache** ($C_1$)

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

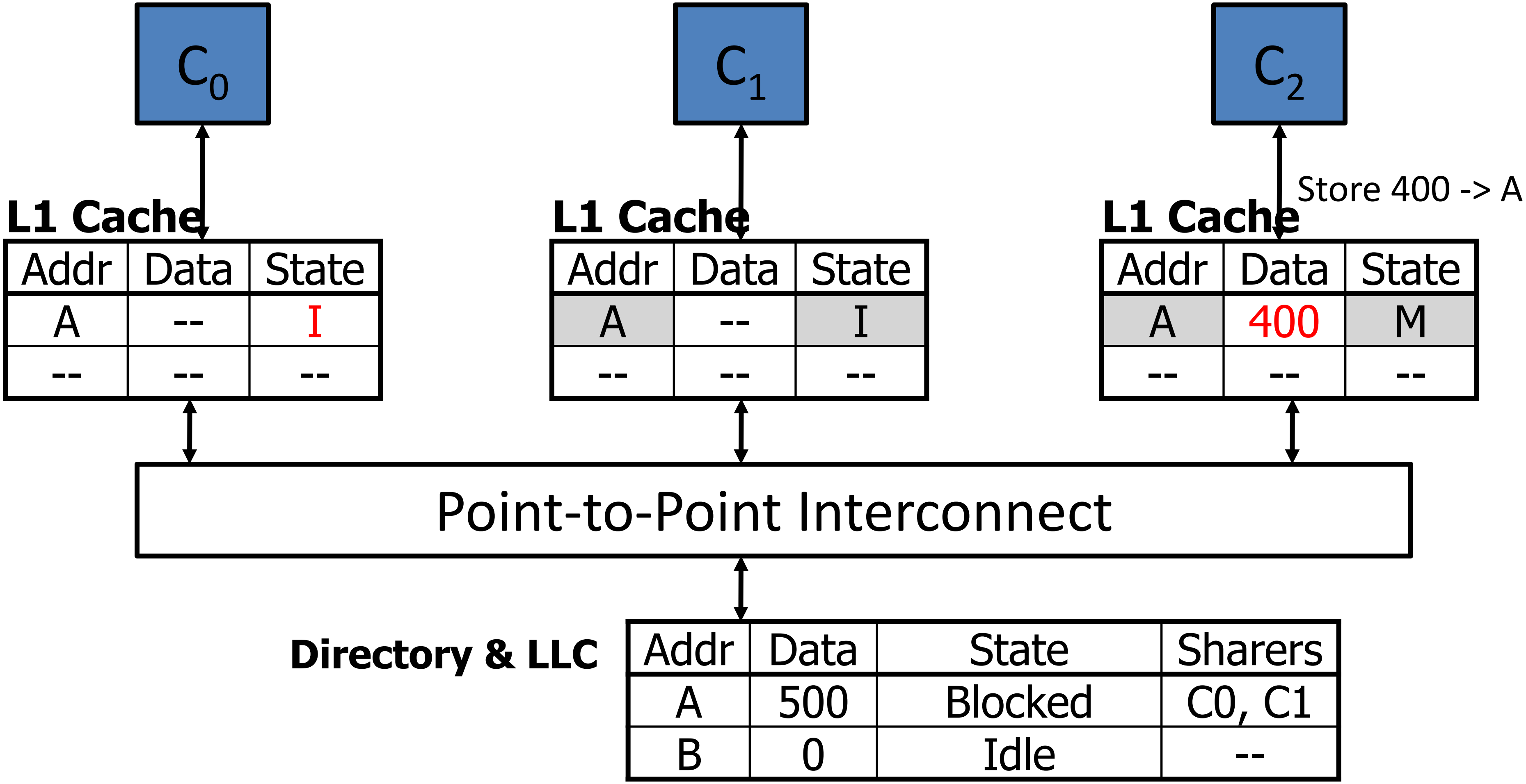
**L1 Cache** ($C_2$)

| Addr | Data | State |
|---|---|---|
| A | 500 | M |
| -- | -- | -- |

**Directory & LLC**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Blocked | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #12



Store 400 -> A

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

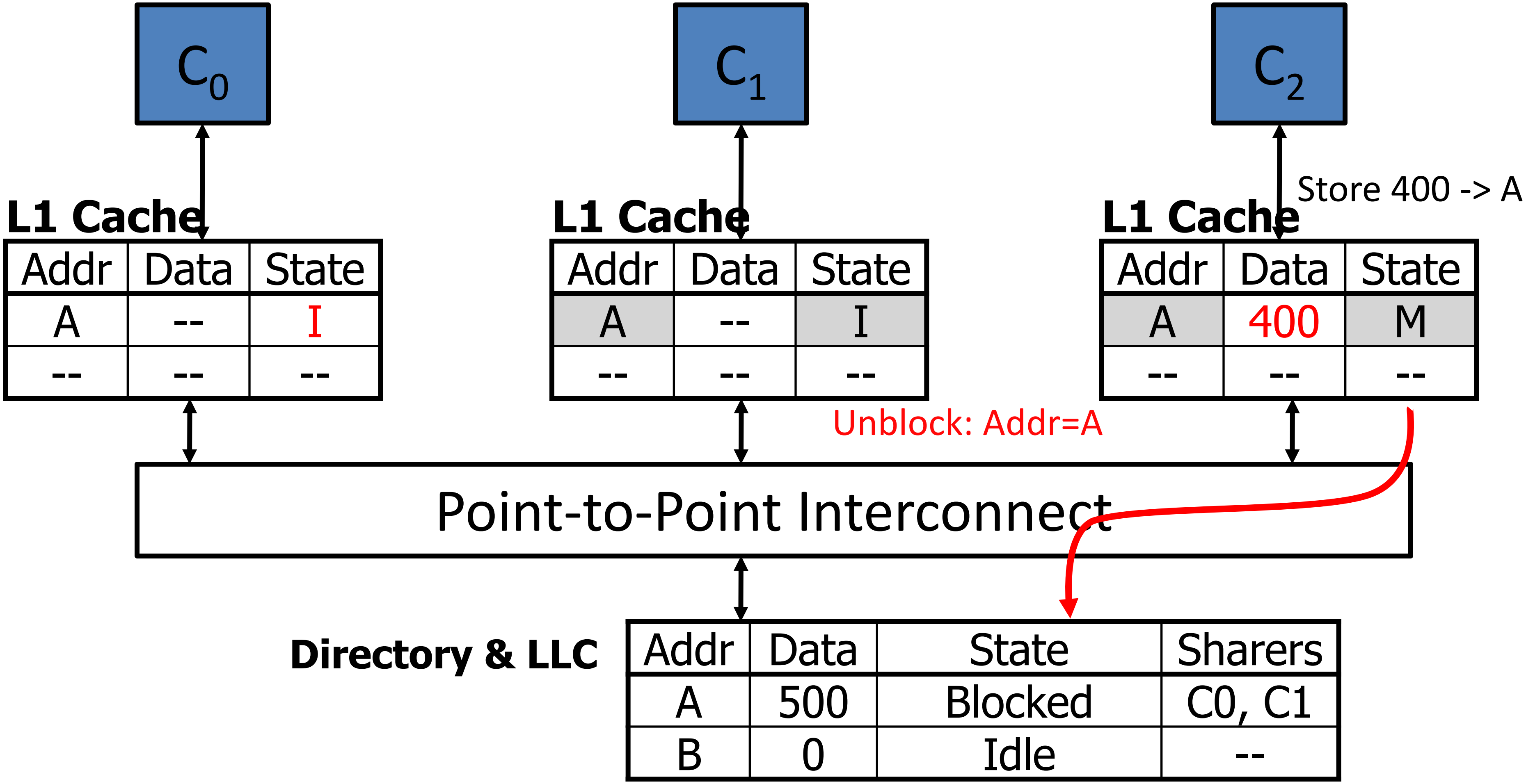
**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | 400 | M |
| -- | -- | -- |

**Directory & LLC**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Blocked | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #12



$C_0$

$C_1$

$C_2$

Store 400 -> A

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

**L1 Cache**

| Addr | Data | State |
|---|---|---|
| A | 400 | M |
| -- | -- | -- |

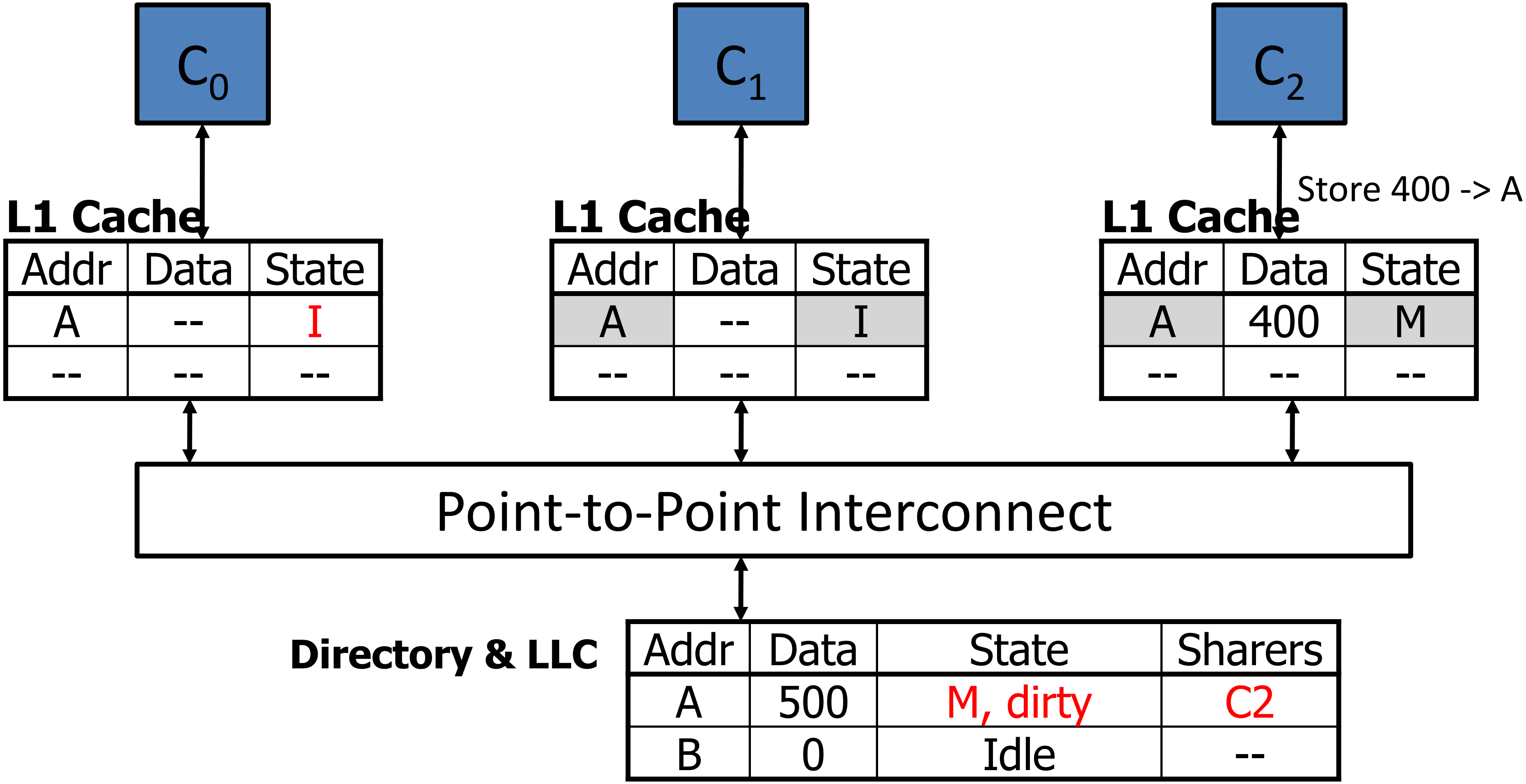
Unblock: Addr=A

Point-to-Point Interconnect

**Directory & LLC**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | Blocked | C0, C1 |
| B | 0 | Idle | -- |

# MSI Directory Example: Step #13



**L1 Cache** ($C_0$)

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

**L1 Cache** ($C_1$)

| Addr | Data | State |
|---|---|---|
| A | -- | I |
| -- | -- | -- |

**L1 Cache** ($C_2$)

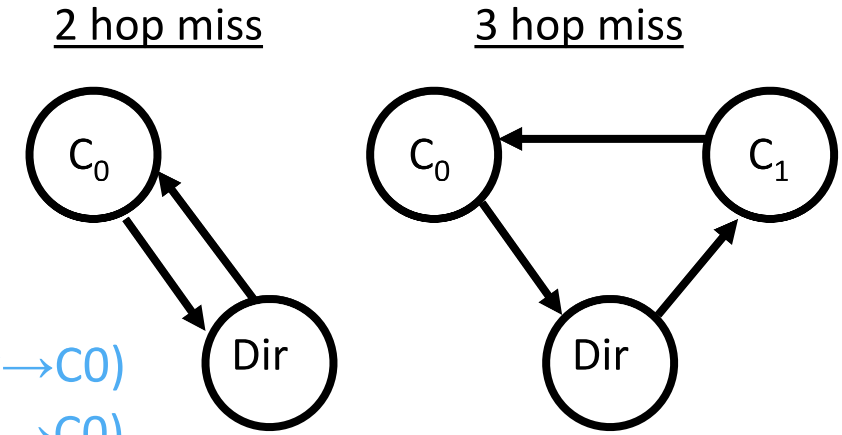
| Addr | Data | State |
|---|---|---|
| A | 400 | M |
| -- | -- | -- |

**Directory & LLC**

| Addr | Data | State | Sharers |
|---|---|---|---|
| A | 500 | M, dirty | C2 |
| B | 0 | Idle | -- |

# Directory Flip Side: Latency

- Directory protocols
  + Lower bandwidth consumption → more scalable
  - Longer latencies



- Two read miss situations
- Unshared: get data from memory
  - Snooping: 2 hops (C0→LLC/memory→C0)
  - Directory: 2 hops (C0→LLC/memory→C0)
- Shared or exclusive: get data from other processor (P1)
  - Assume cache-to-cache transfer optimization
  - Snooping: 2 hops (C0→C1→C0)
  - Directory: **3 hops** (C0→Dir/LLC→C1→C0)
  - Common, with many processors high probability someone has it

# Directory Flip Side: Complexity

- Latency not the only issue for directories
  - Subtle correctness issues as well
  - Stem from unordered nature of underlying inter-connect
- Individual requests to single cache must be ordered
  - Point-to-point network: requests may arrive in different orders
    - Directory has to enforce ordering explicitly
    - Cannot initiate actions on request B…
    - Until all relevant processors have completed actions on request A
    - Requires directory to collect acks, queue requests, etc.
- Directory protocols
  - Obvious in principle
  - Complicated in practice
  - **State space explosion due to unordered network**
  - **Need to consider various possible coherence message races**