CS/ECE 5381/7381 Computer Architecture Spring 2023

Dr. Manikas

Computer Science

Lecture 20: Apr. 13, 2023

Assignments

- Quiz 9 – due Sat., Apr. 15 (11:59 pm)
  - Covers concepts from Module 11 (this week)

Quiz 9 Details

- The quiz is open book and open notes.
- You are allowed 90 minutes to take this quiz.
- You are allowed 2 attempts to take this quiz - your highest score will be kept.
  - Note that some questions (e.g., fill in the blank) will need to be graded manually
- Quiz answers will be made available 24 hours after the quiz due date.

Thread-Level Parallelism

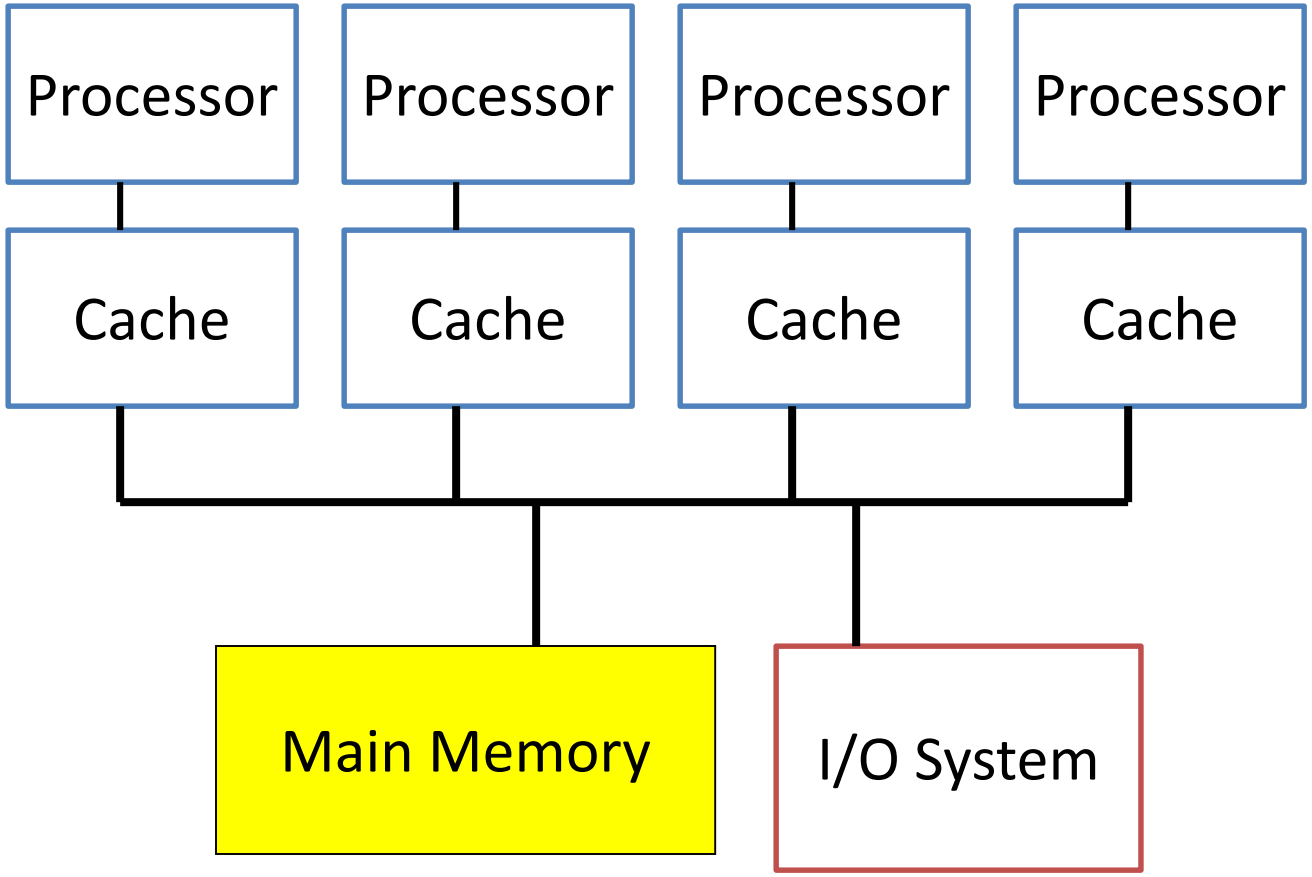
(Chapter 5, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Outline

- 5.1 Introduction
- 5.2 Centralized Shared-Memory Architectures

Centralized Shared-Memory



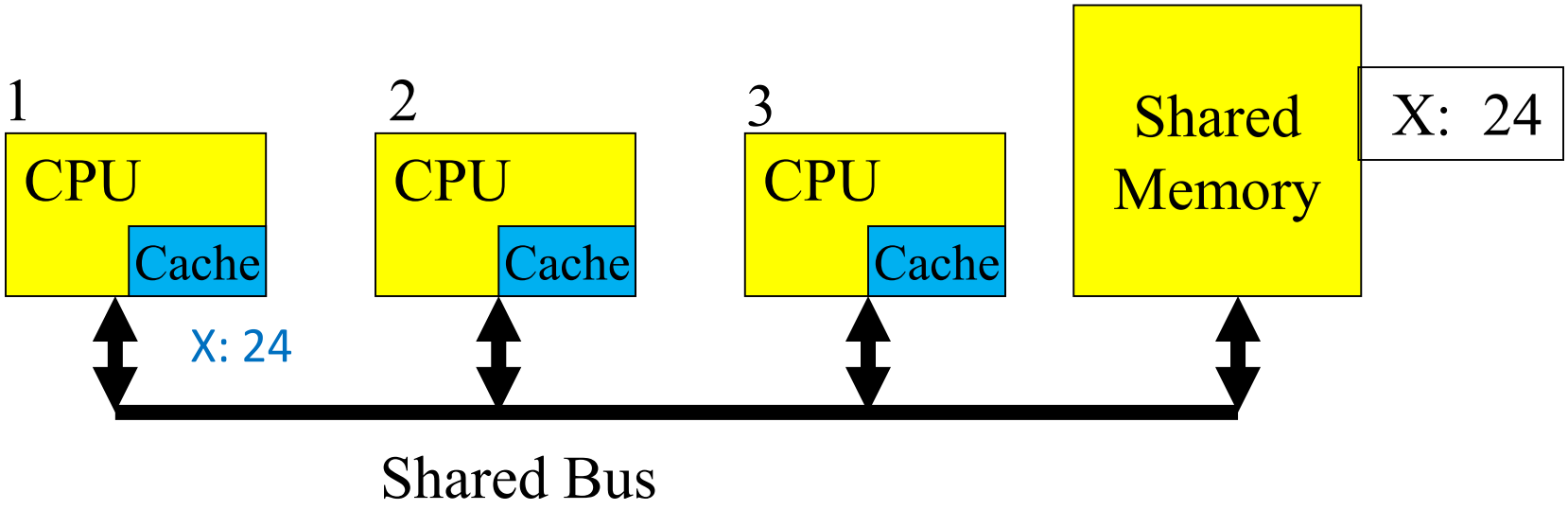
Multiprocessor Cache Coherence

- Recall that a processor's view of main memory is through the cache
- With multiple processors, each processor has its own cache
- How to make sure that the values in each cache match?
  - This is the cache coherence problem

Problem of Memory Coherence

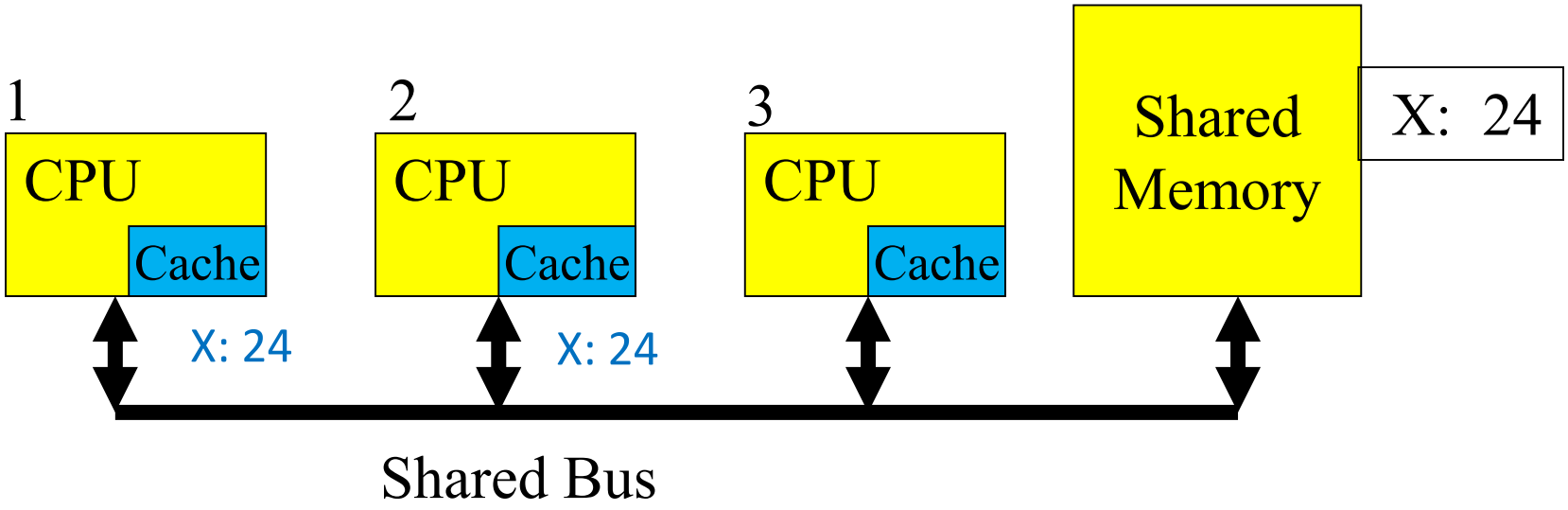
- Assume just single level caches and main memory
- Processor writes to location in its cache
- Other caches may hold shared copies - these will be out of date
- Updating main memory alone is not enough

Example



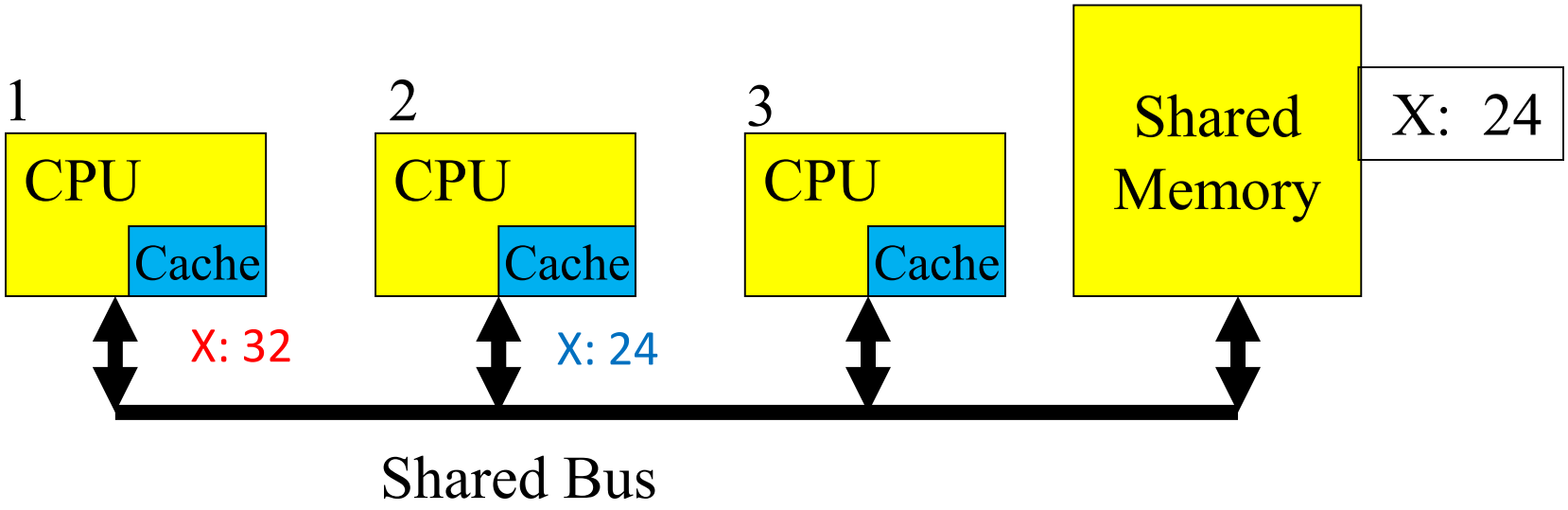
Processor 1 reads X: obtains 24 from memory and caches it

Example



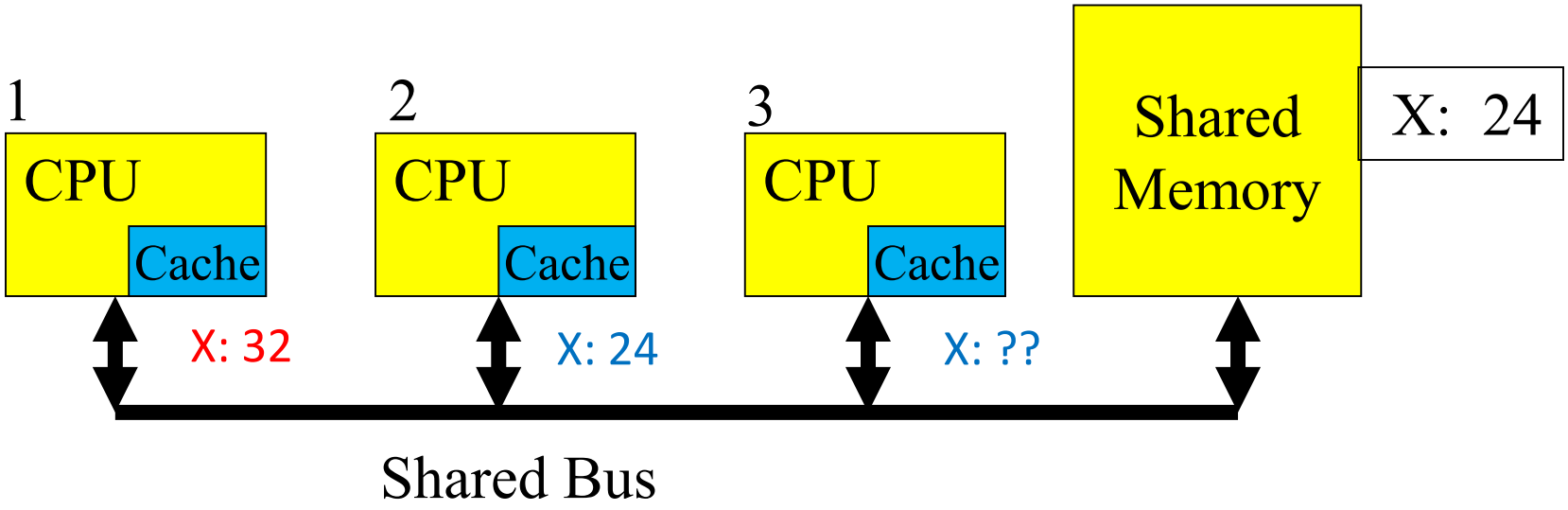
Processor 2 reads X: obtains 24 from memory and caches it

Example



Processor 1 writes 32 to X: its locally cached copy is updated

Example



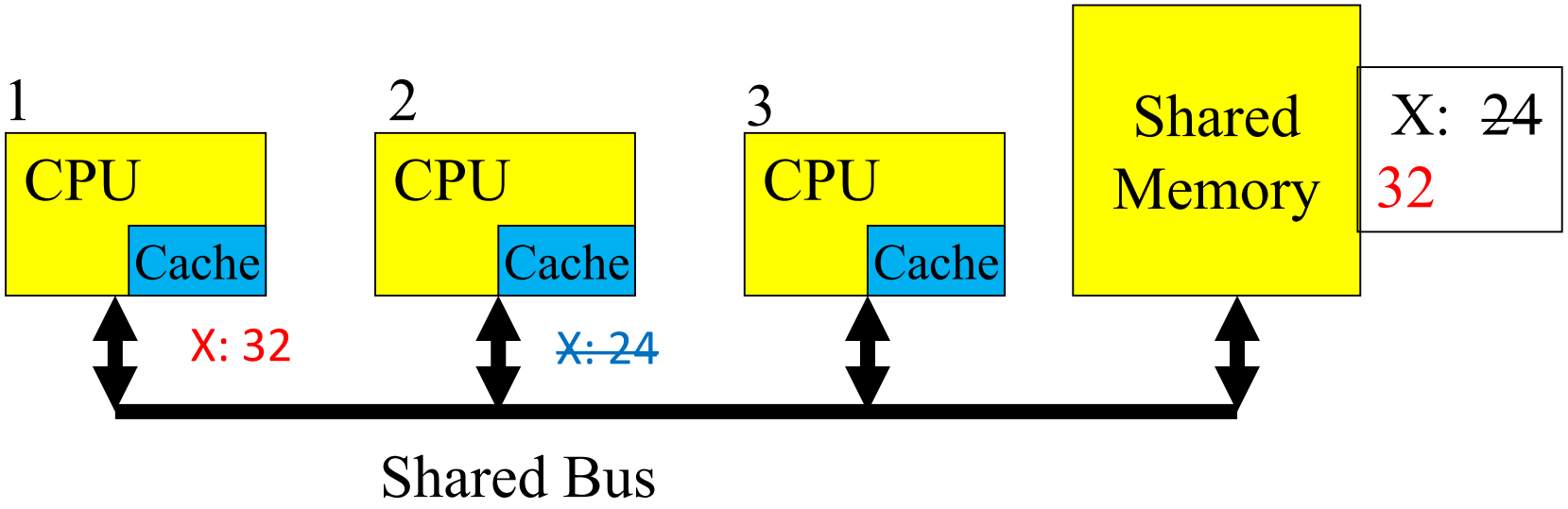
Processor 3 reads X: what value should it get?

Memory and processor 2 think it is 24

Processor 1 thinks it is 32

Cache Coherence Protocols

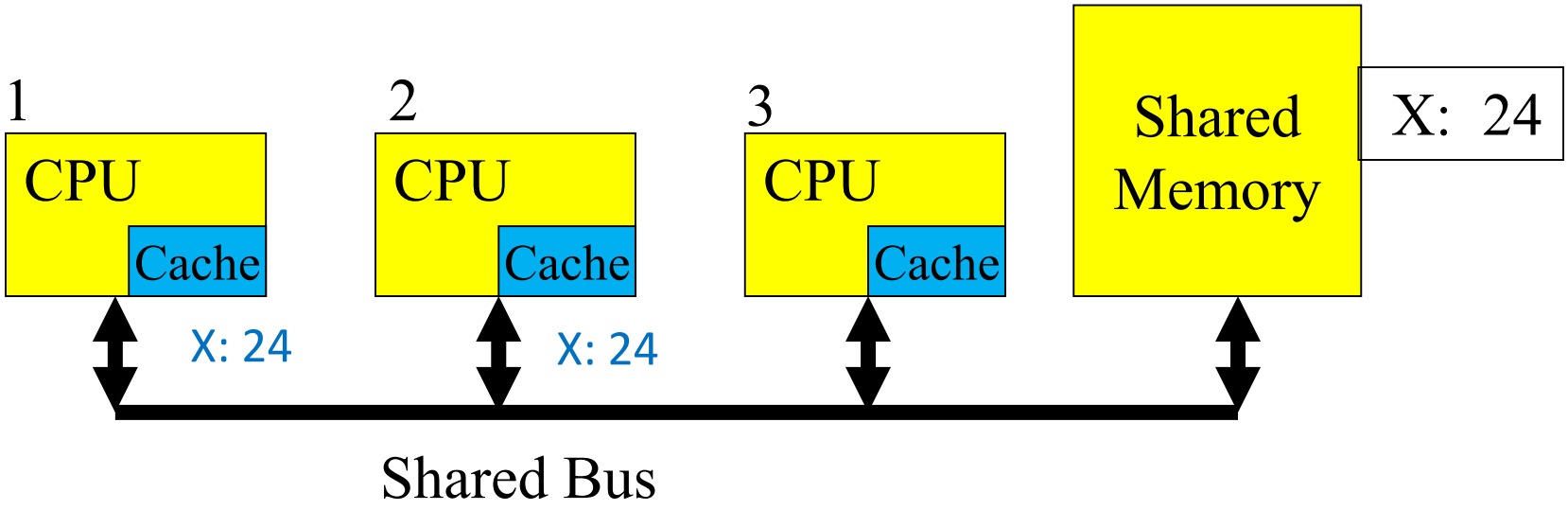
- Directory based — Sharing status of a block of physical memory is kept in just one location, the directory
- Snooping — Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept
  - All caches are accessible via some broadcast medium (a bus or switch)
  - All cache controllers monitor or snoop on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access



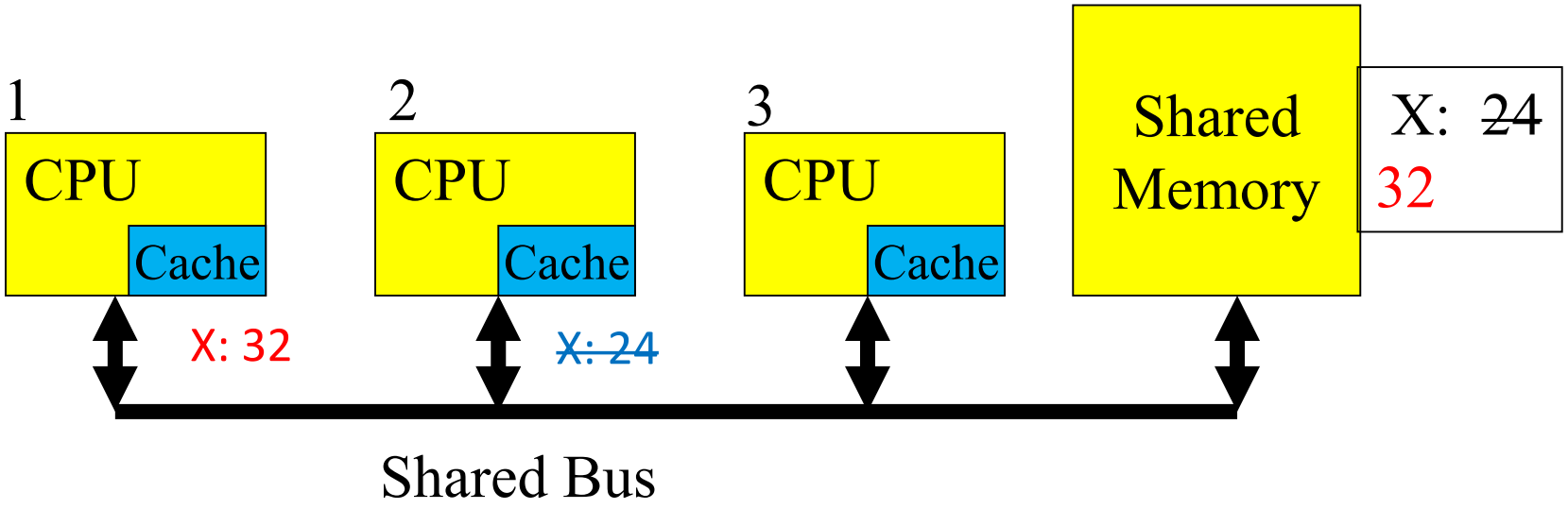
1. Processor 1 writes 32 to X: its locally cached copy is updated
2. All snooping caches *invalidate* their copy of X
3. CPU 1 writes cached copy to shared memory (new X value)
4. Any shared reads of X for other CPUs will now *miss* in local caches – must re-fetch new data from shared memory (X now is 32)

MESI Protocol

- A practical multiprocessor invalidate protocol which attempts to minimize bus usage
- Extension of usual cache bits
  - Valid bit – data is valid in cache
  - Dirty bit – data is old, needs to be updated
- Allows usage of a ‘write back’ scheme - i.e. main memory not updated until ‘dirty’ cache line is displaced



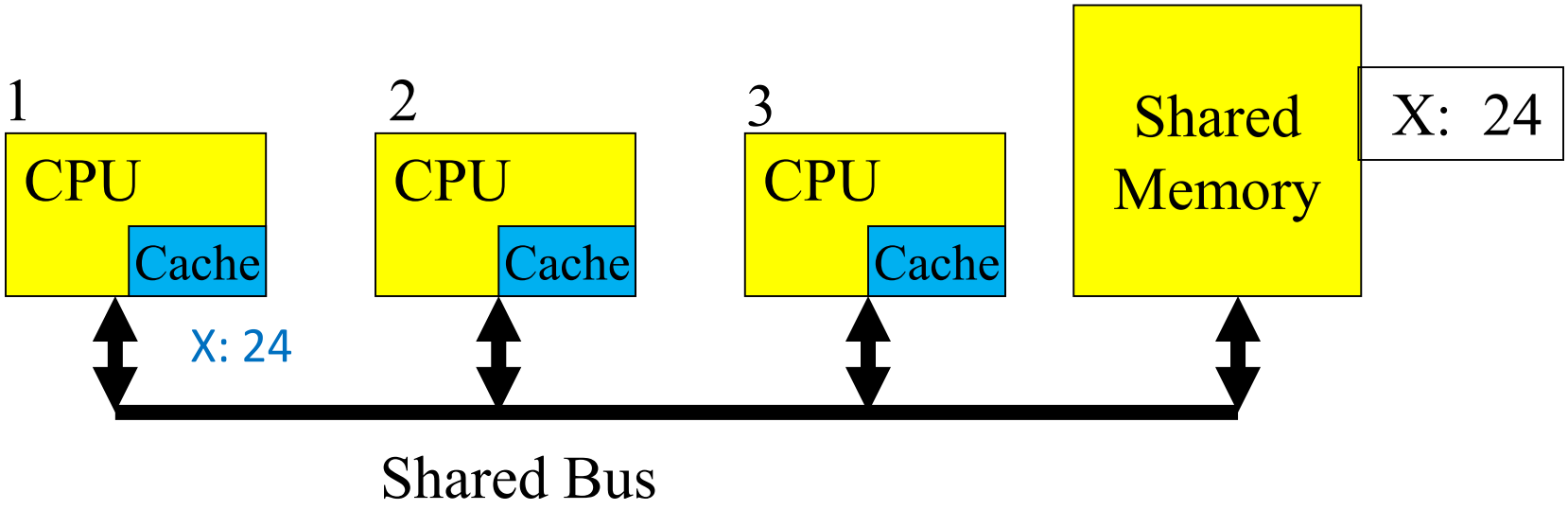
Cache	Valid	Dirty	Data
1	1	0	24
2	1	0	24
3	0	0	(empty)



Cache	Valid	Dirty	Data
1	1	0	32
2	1	1	24
3	0	0	(empty)

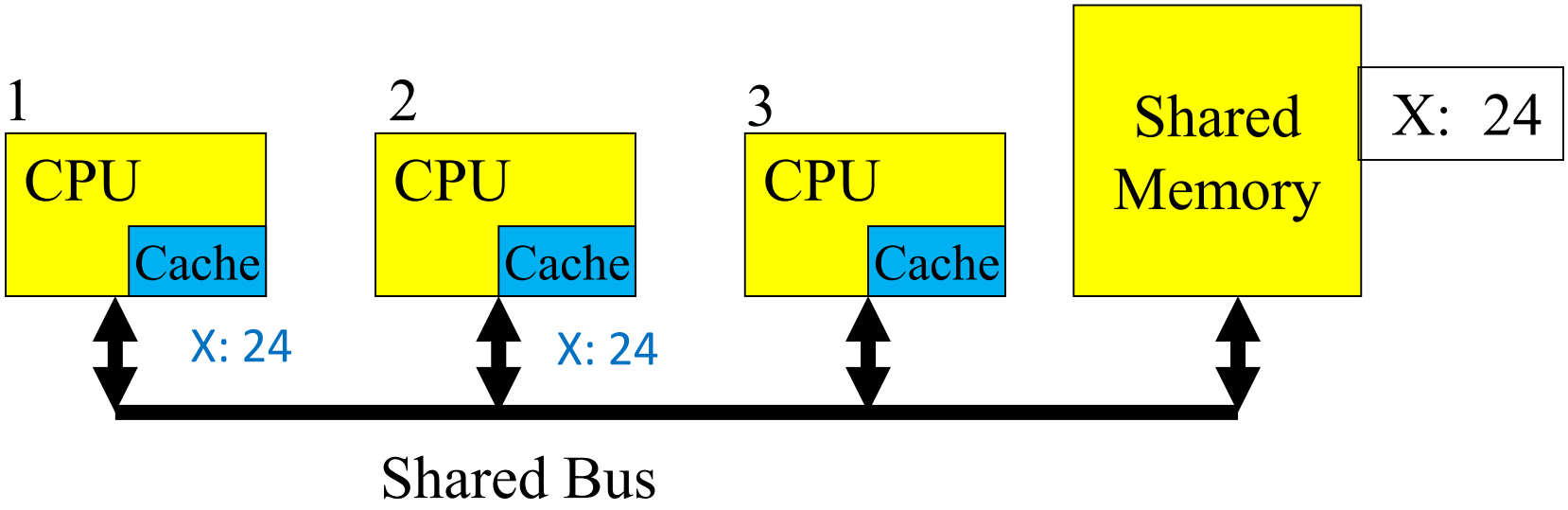
MESI Protocol – Cache Line States

- **(M) Modified** - cache line has been modified, is different from main memory - is the only cached copy. (multiprocessor ‘dirty’)
- **(E) Exclusive** - cache line is the same as main memory and is the only cached copy
- **(S) Shared** - Same as main memory but copies may exist in other caches.
- **(I) Invalid** - Line data is not valid (as in simple cache)



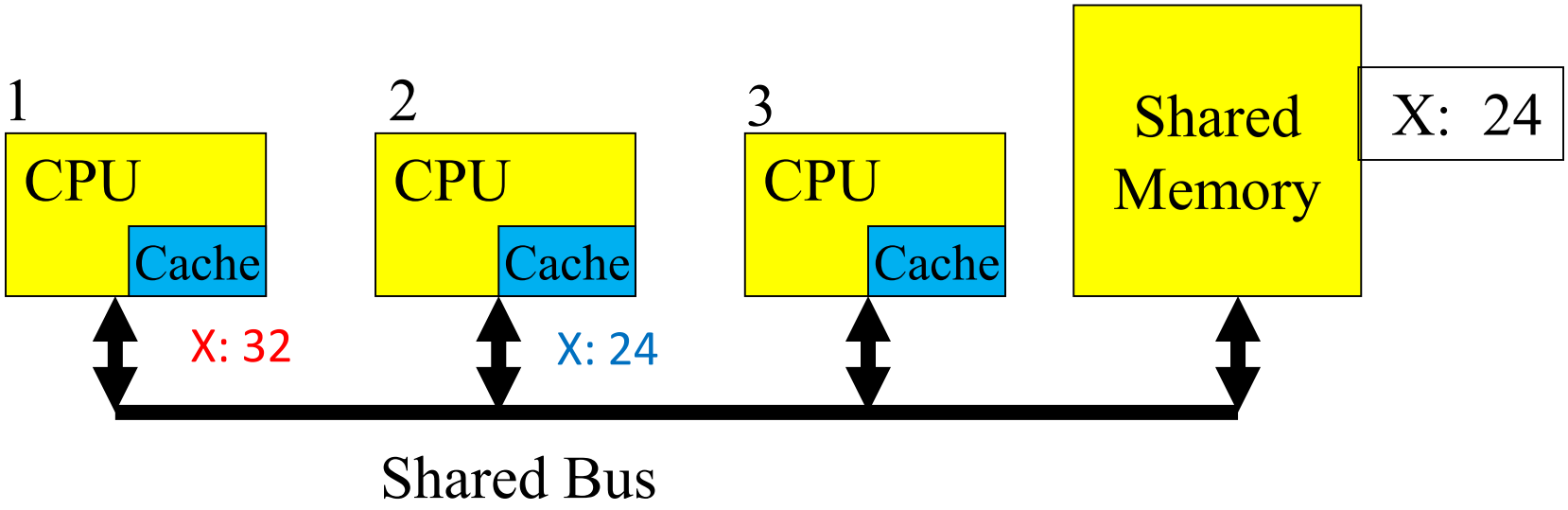
Cache 1 line is in Exclusive (E) state: same value as main (shared) memory for address X

Cache 2 and Cache 3 lines are in Invalid (I) state: no copies of data for address X



Cache 1 line is in Shared (S) state: same value as main memory for address X, but copies exist in other caches (Cache 2)

Note that Cache 2 line is also in Shared (S) state



Cache 1 line is in Modified (M) state: cache line value has been modified and is *different* from main memory

Cache Line State Changes

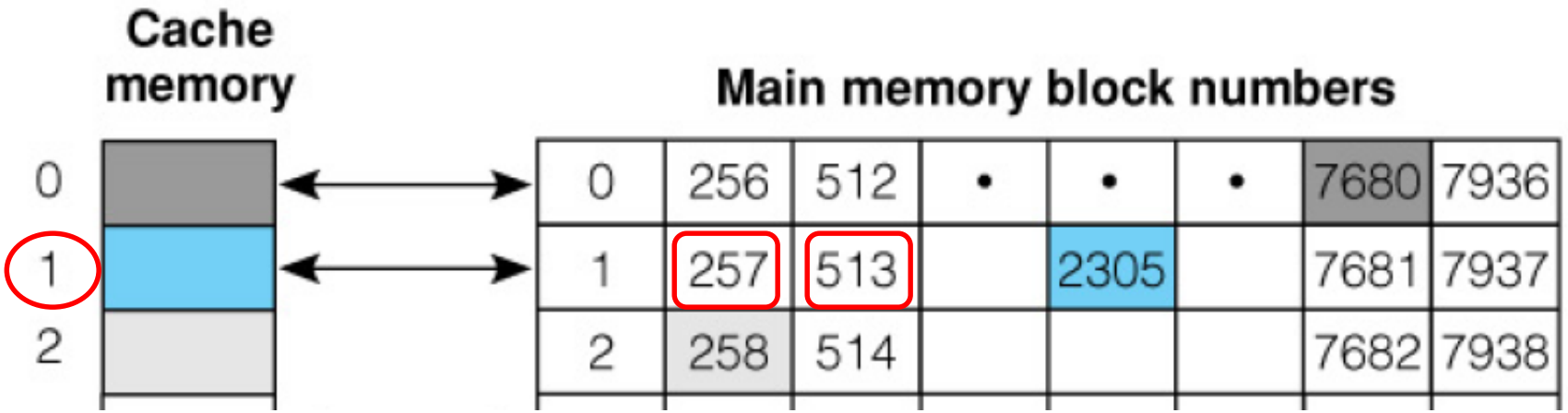
- Cache line changes state as a function of memory access events.
- Event may be either
  - Due to local processor activity (i.e. cache access)
  - Due to bus activity - as a result of snooping
- Cache line has its own state affected only if address matches

MESI Protocol Operation

- Operation can be described informally by looking at action in local processor
  A. Read Hit
  B. Read Miss
  C. Write Hit
  D. Write Miss
- More formally by state transition diagram
- We will use a working example to illustrate

Assumptions for Example

- Caches are *direct-mapped*
  - Main memory blocks A1 and A2 map to same cache block, but A1 ≠ A2
    - E.g., A1 = 257 and A2 = 513: both map to cache block 1



Cache Actions

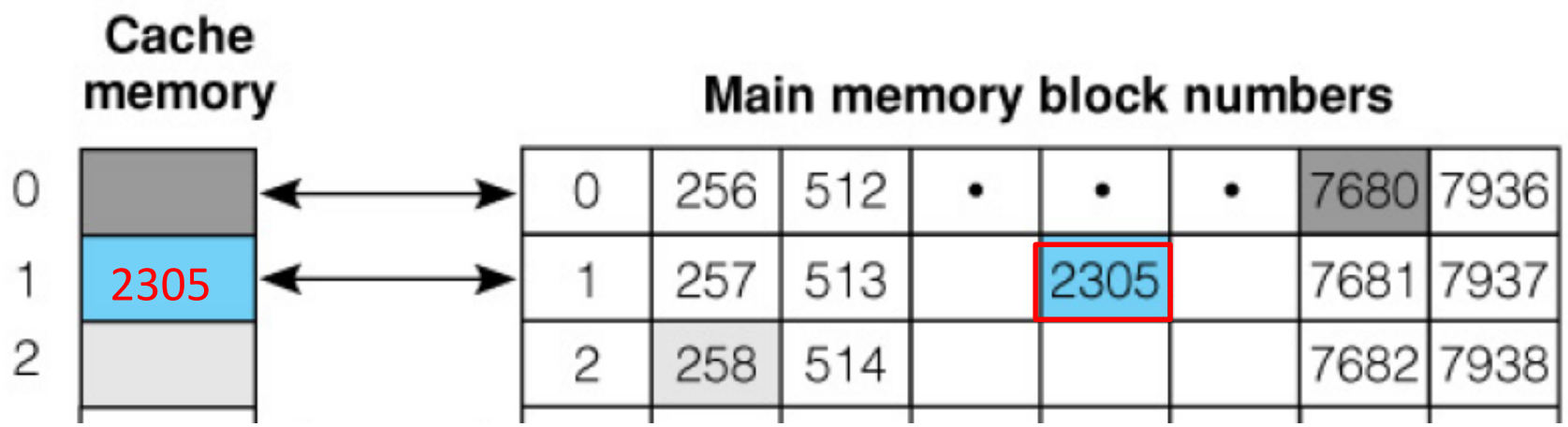
- HIT – copy of item found in cache
- MISS – copy of item NOT found in cache

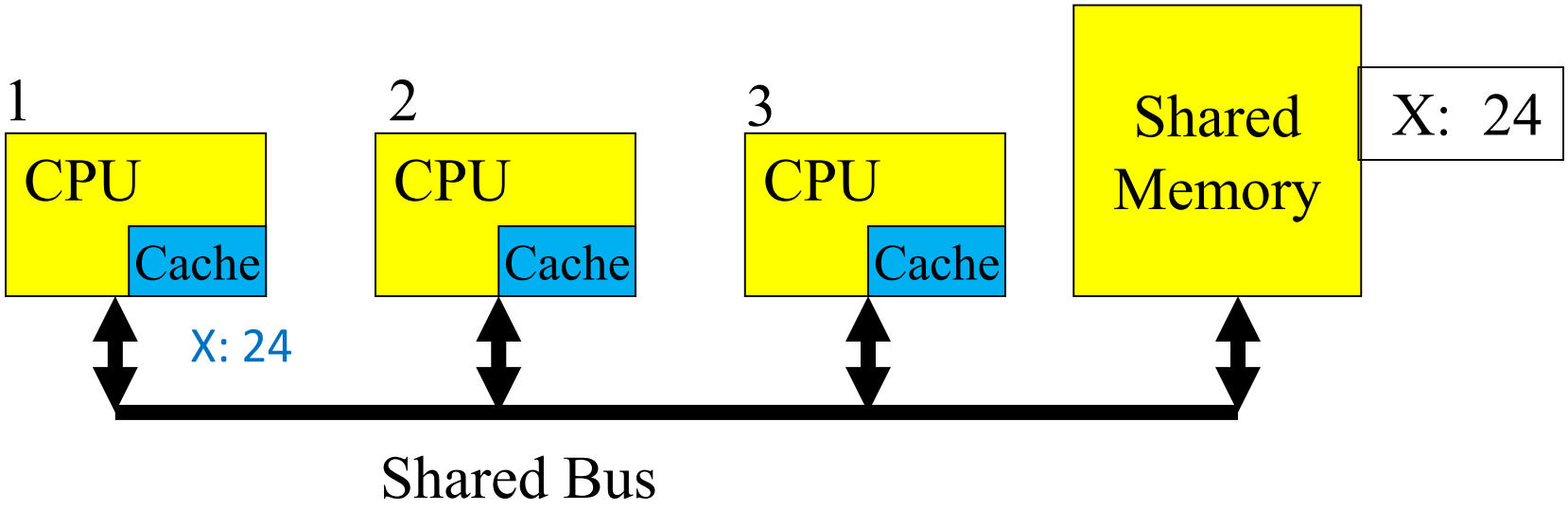
CPU Read/Write

- READ: CPU wants to load memory item X into its registers
  - READ HIT: copy of item X exists in cache
  - READ MISS: copy of X does NOT exist in cache
- WRITE: CPU wants to store value from a register into memory item X
  - WRITE HIT: copy of item X exists in cache
  - WRITE MISS: copy of X does NOT exist in cache

A. Local Read Hit

- Read Hit: value found in cache for memory address X
  - Copy of address X in cache
- E.g., processor requests value in block 2305

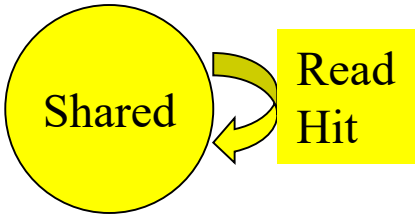


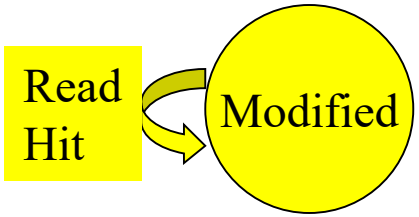


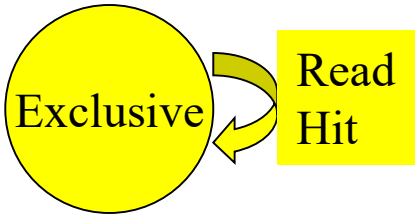
CPU 1 requests value for X

Valid copy of X is in cache 1: local read hit

Read Hit State Diagram

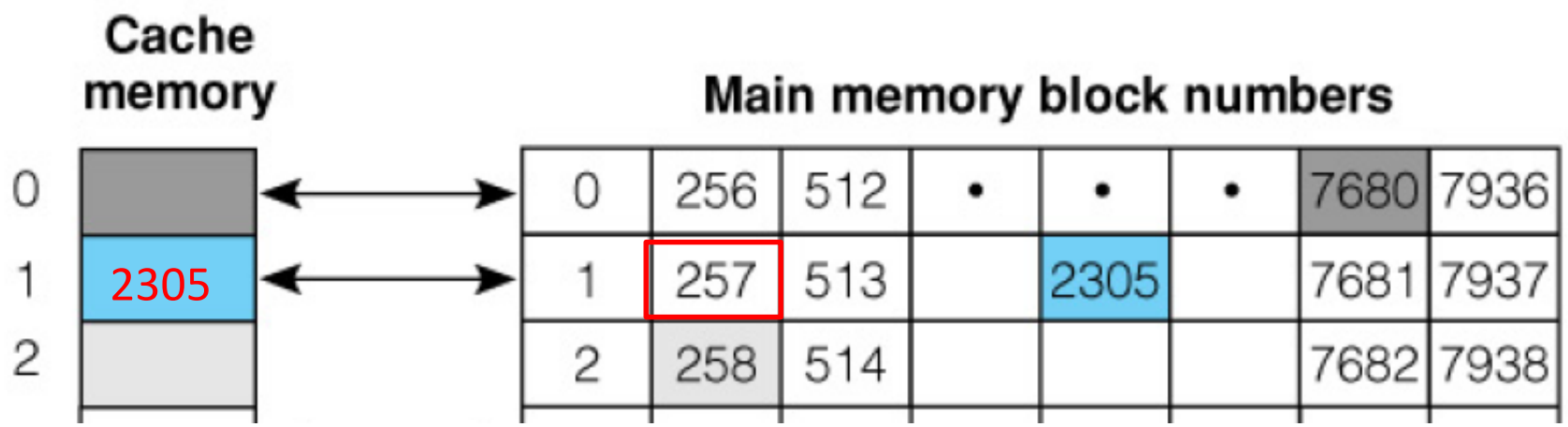






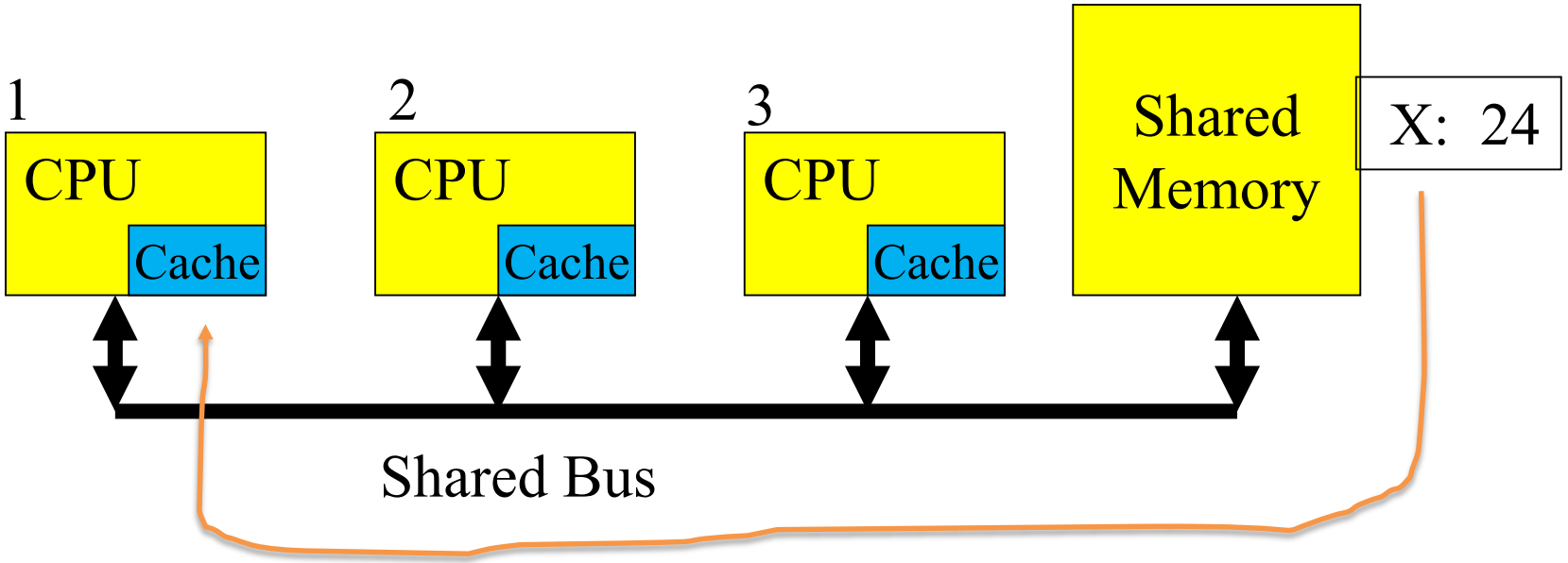
B. Local Read Miss

- Read Miss: value not found in cache
  - Cache line state is I (Invalid)
- E.g., processor requests value in block 257
  - Not in cache line



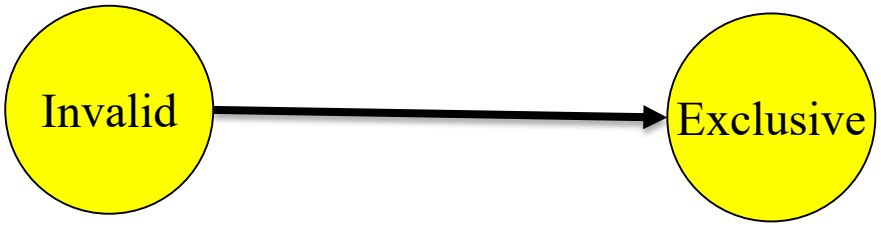
Local Read Miss (1)

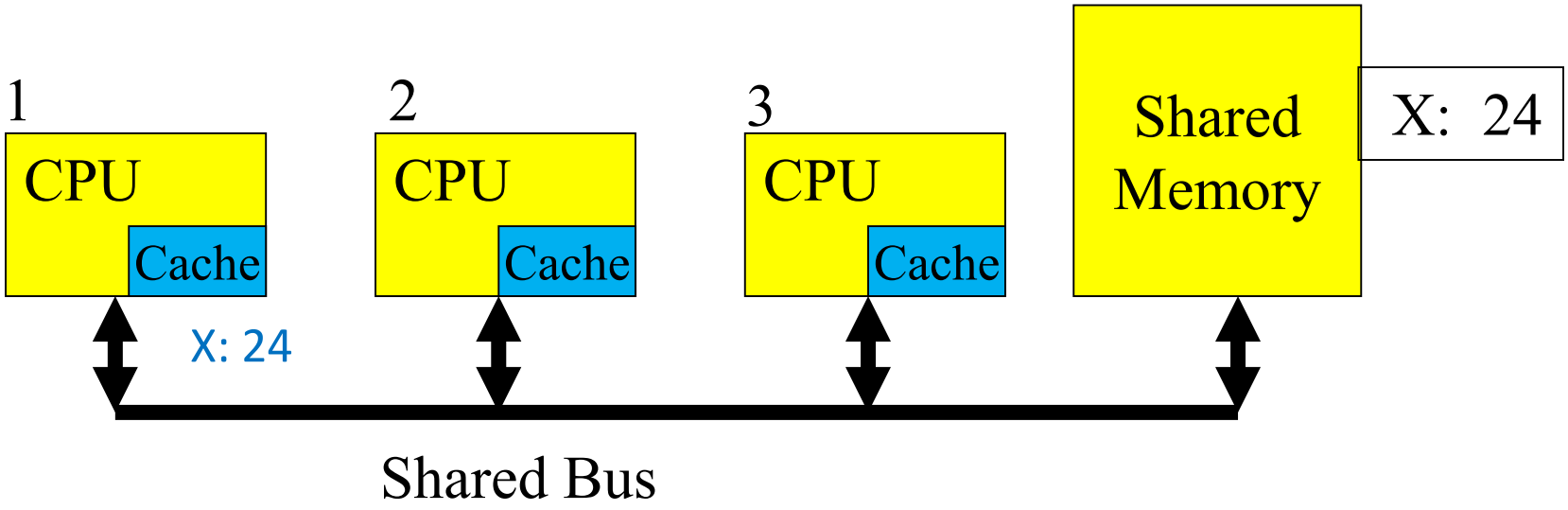
- **If no copies in other caches**
  - Processor must make a bus request to main memory
  - Value read to local cache
  - Cache line state is now E (Exclusive)
    - Cache line same as main memory and is only cached copy



Read miss for Cache 1: get X from shared (main) memory and read into appropriate cache line for cache 1 (assume X is in block 257)

Cache 1 MESI state diagram

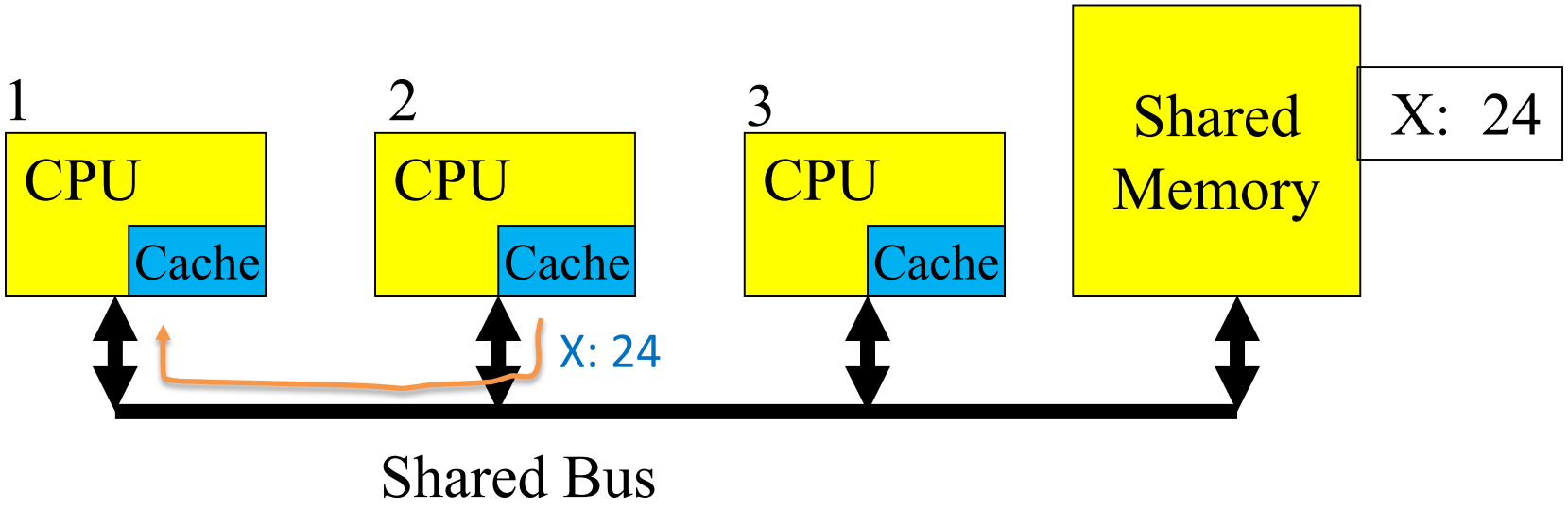




Cache 1 line is now in Exclusive (E) state: same value as main (shared) memory for address X

Local Read Miss (2)

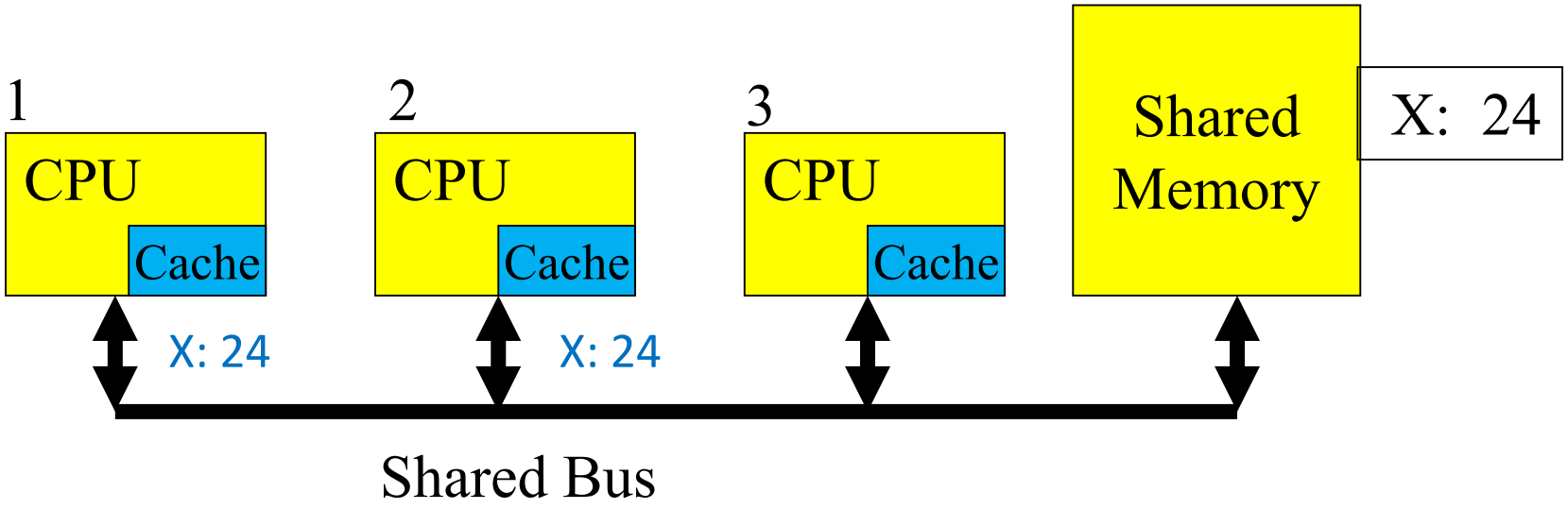
- **Else, if another cache has an exclusive copy of item (state E)**
  - Processor makes a bus request to main memory
  - Snooping cache puts copy of item on bus
  - Value read to local cache
  - Both cache line states are now S (Shared)
    - Both cache lines now have same value as main memory



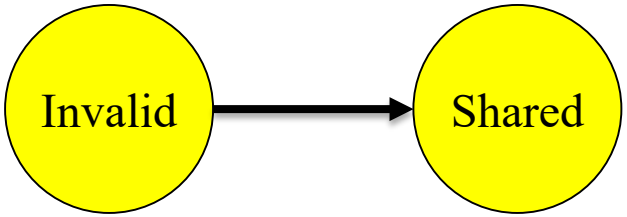
Cache 2 has copy of X – need to copy to Cache 1

Cache 2 is "snooping" the bus: Cache 2 sees a request for X, so Cache 2 puts a copy on the bus

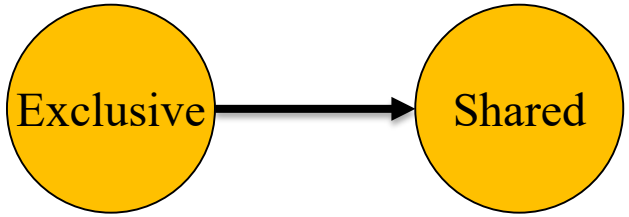
Copy of X is read into Cache 1



Cache 1 MESI state diagram

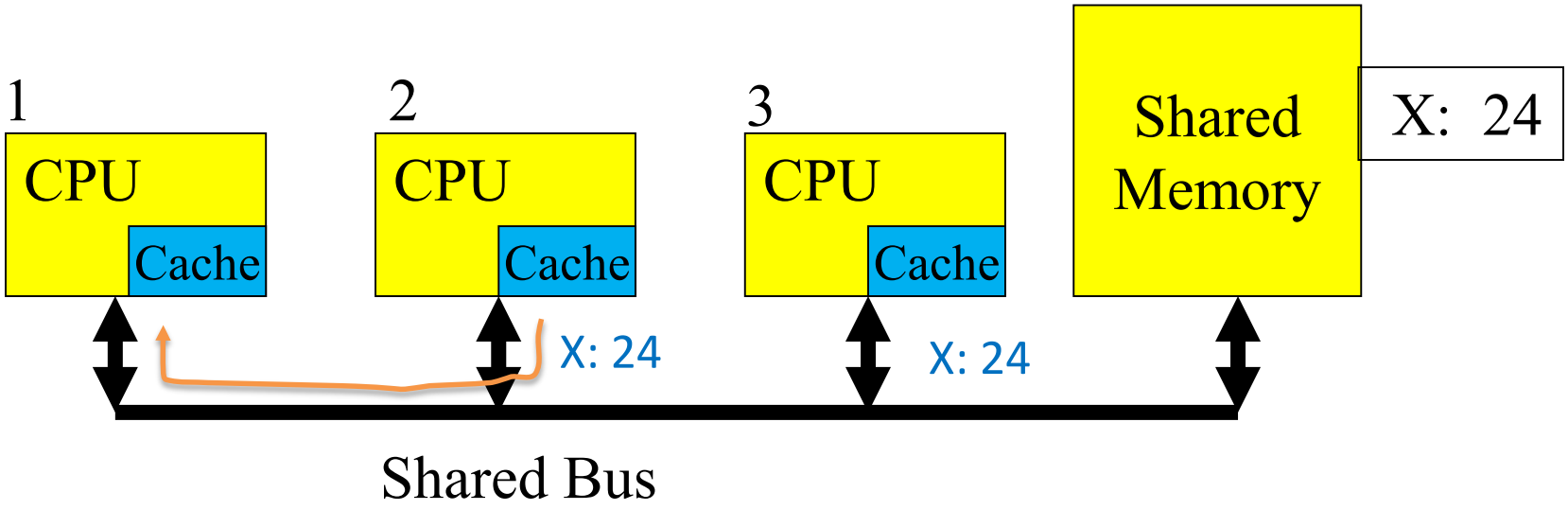


Cache 2 MESI state diagram



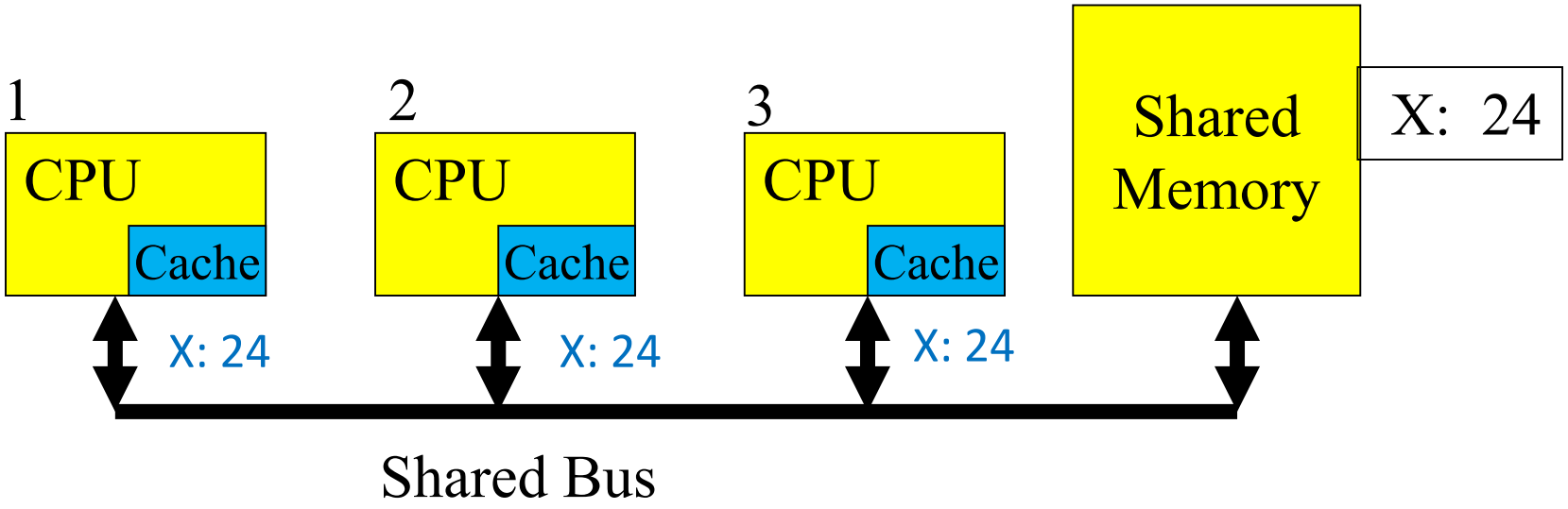
Local Read Miss (3)

- **Else, if other caches have a shared copy of item (state S)**
  - Processor makes a bus request to main memory
  - **One cache** puts copy of item on bus
  - Value read to local cache
  - Local cache state is now S
  - Other caches with copies remain at state S

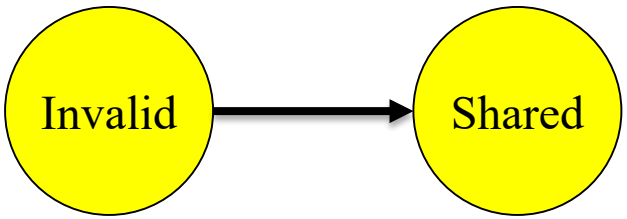


Caches 2 and 3 both have copy of X - one of them will put a copy on the bus

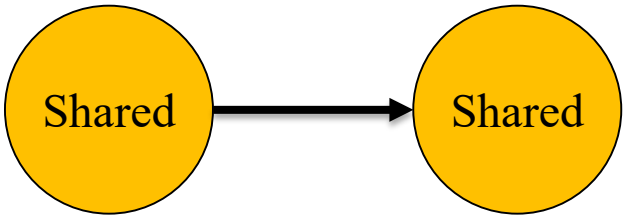
Copy of X is read into Cache 1



Cache 1 MESI state diagram

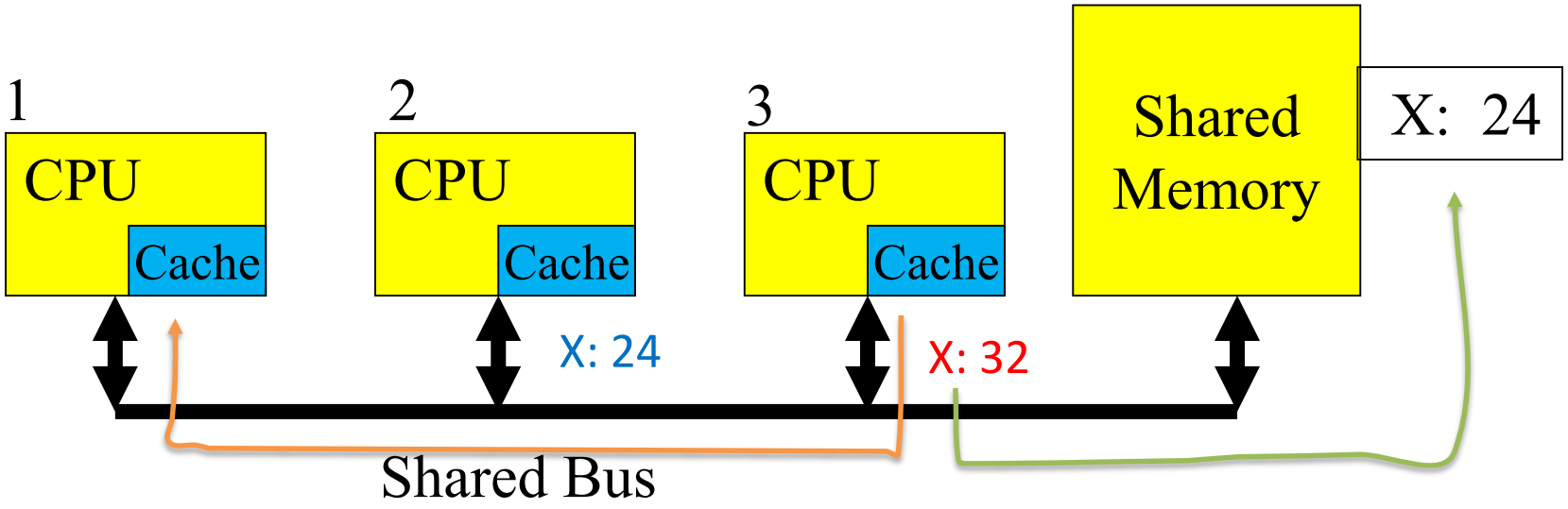


Cache 2 and 3 MESI state diagram



Local Read Miss (4)

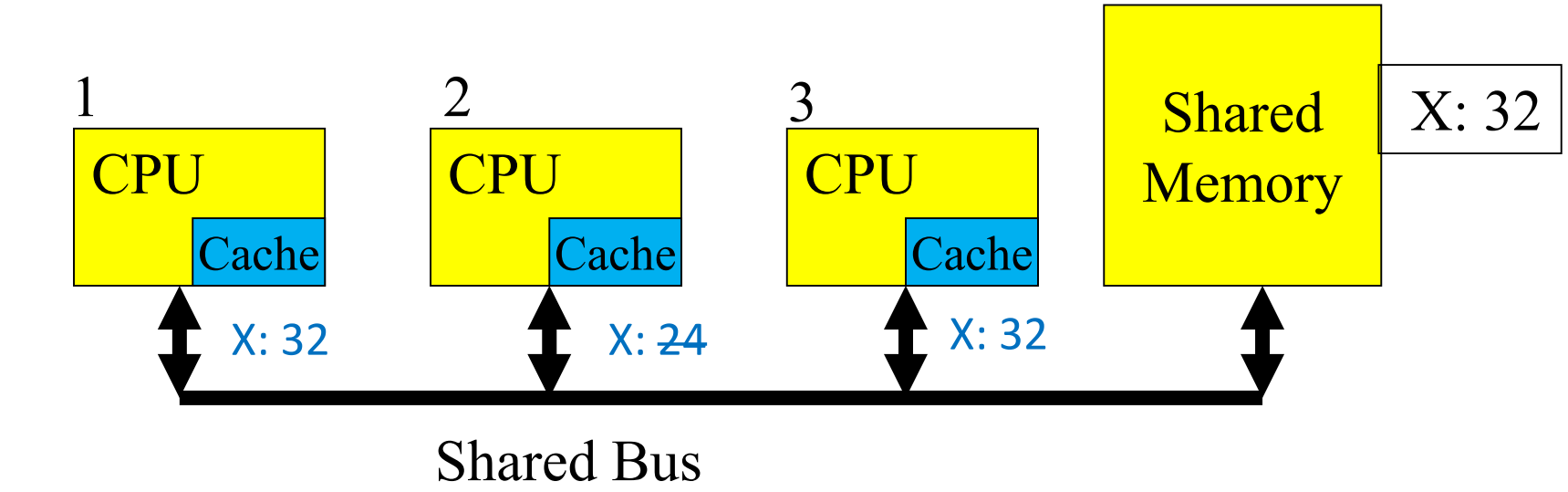
- **Else, if one cache has a *modified* copy of item (state M)**
  - Processor makes a bus request to main memory
  - Cache puts copy of item on bus
  - Value read to local cache
  - Local cache state is now S
  - M cache copies value to main memory
    - State changes from M to S

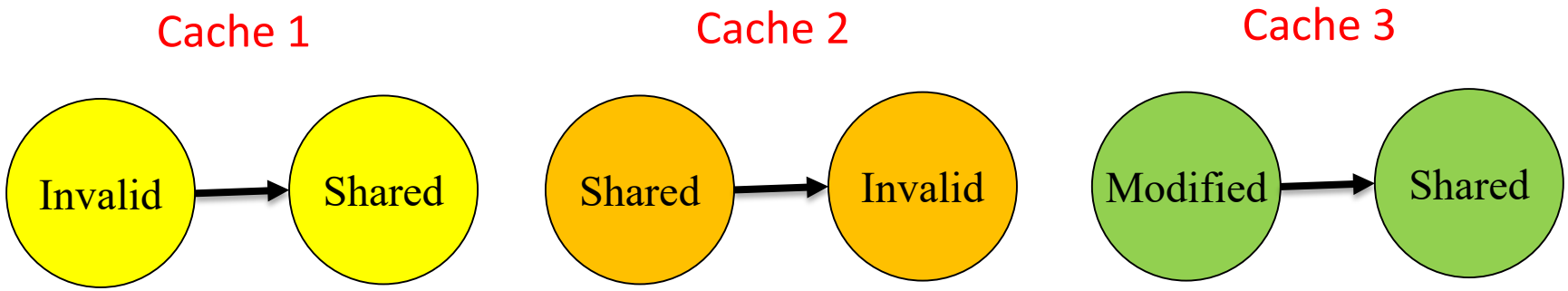


Cache 3 has a *modified* copy of X.

Modified value of X is put on bus, and copied to both Cache 1 and shared (main) memory

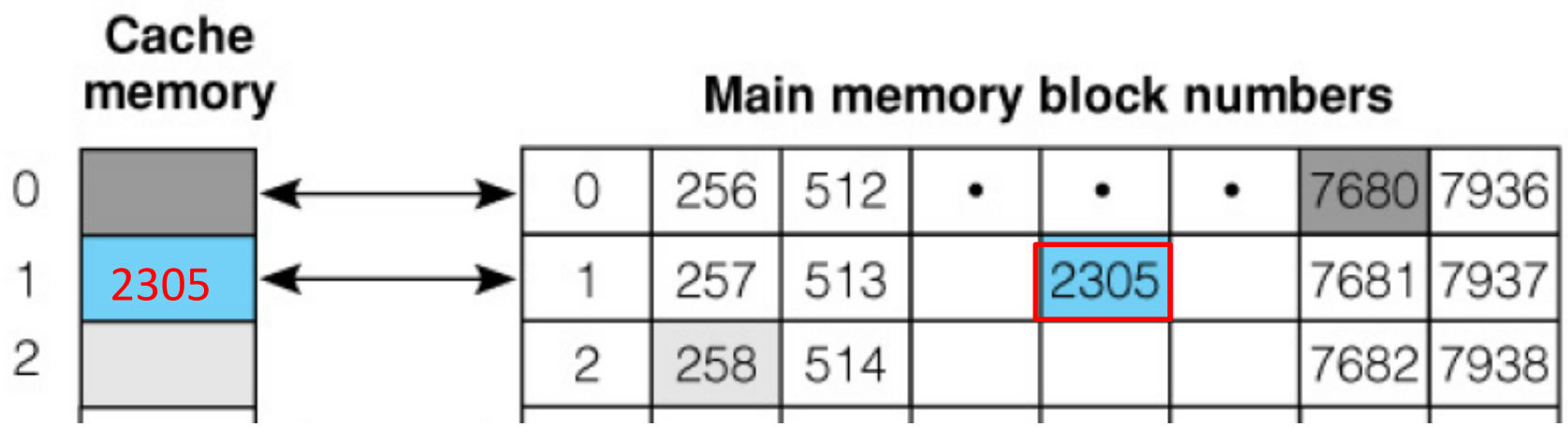
Cache 2 copy of X is old, so is *invalidated* (dirty bit set)


1
CPU
Cache
2
CPU
Cache
3
CPU
Cache
Shared Memory
X: 32
X: 32
X: 24
X: 32
Shared Bus


Cache 1
Invalid
Shared
Cache 2
Shared
Invalid
Cache 3
Modified
Shared

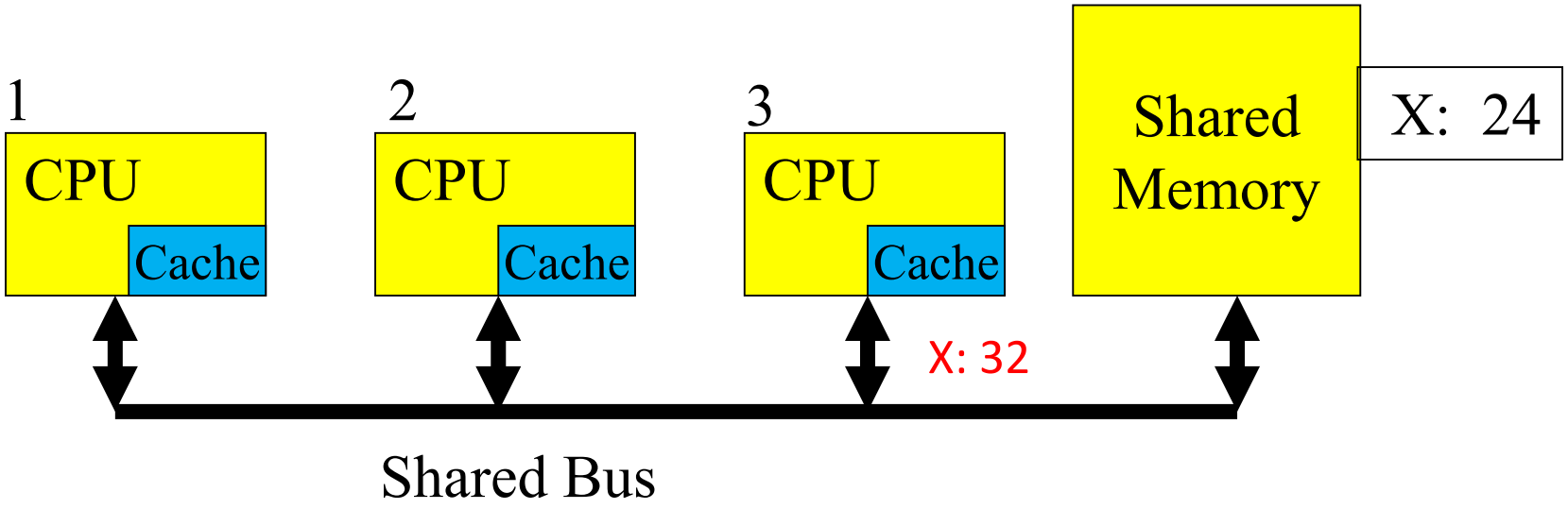
C. Local Write Hit

- Write Hit: CPU wishes to update value for memory address X
  - Copy of address X found in cache
- E.g., processor requests value in block 2305



Local Write Hit (M)

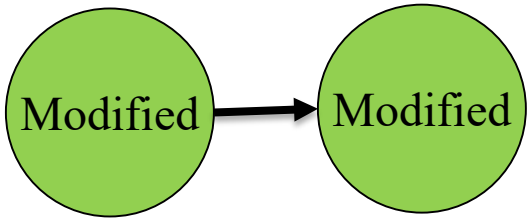
- **Cache line state M: cache line was modified but main memory not yet updated**
  - Cache line is exclusive and already “dirty” (old)
  - Update local cache value
  - No state change



Cache 3 has a *modified* copy of X.

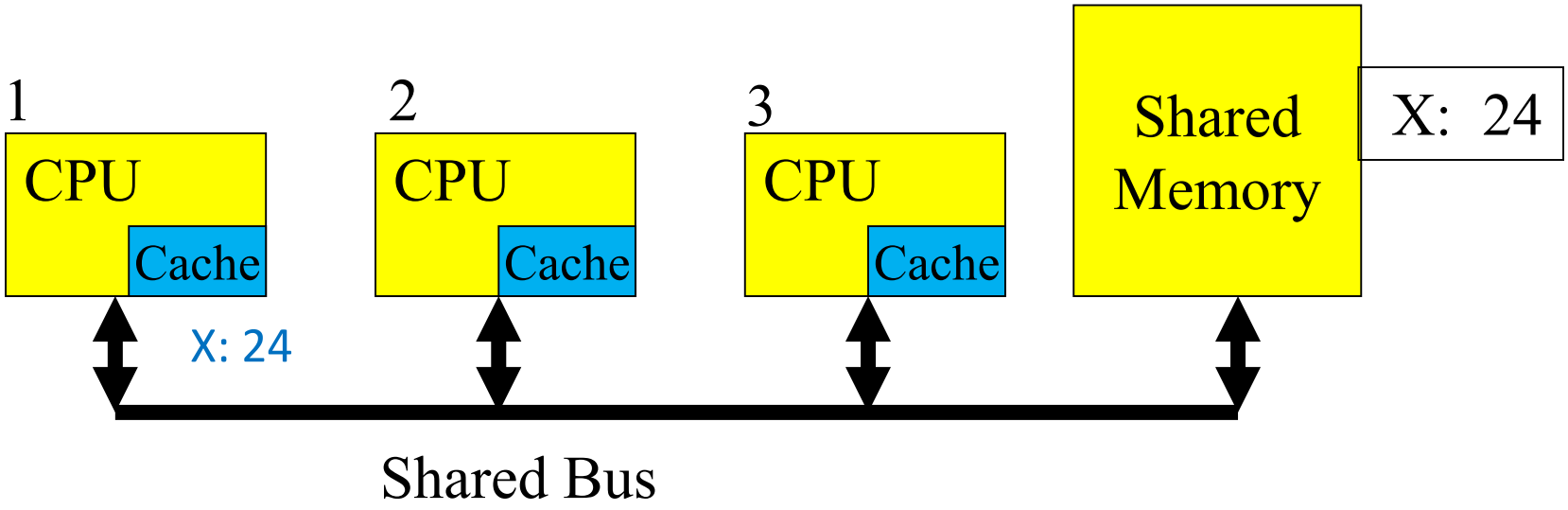
Update Cache 3 line with new value of X.

Cache 3



Local Write Hit (E)

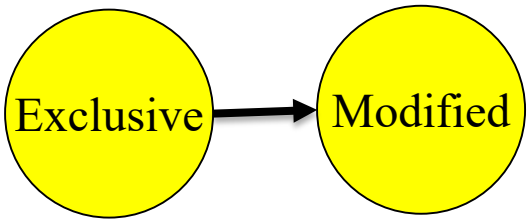
- **Cache line state E: cache line same as main memory and is the only cached copy**
  - Update local cache value
  - Cache line state changes to M since we are modifying value



Cache 1 line is in Exclusive (E) state: same value as main (shared) memory for address X

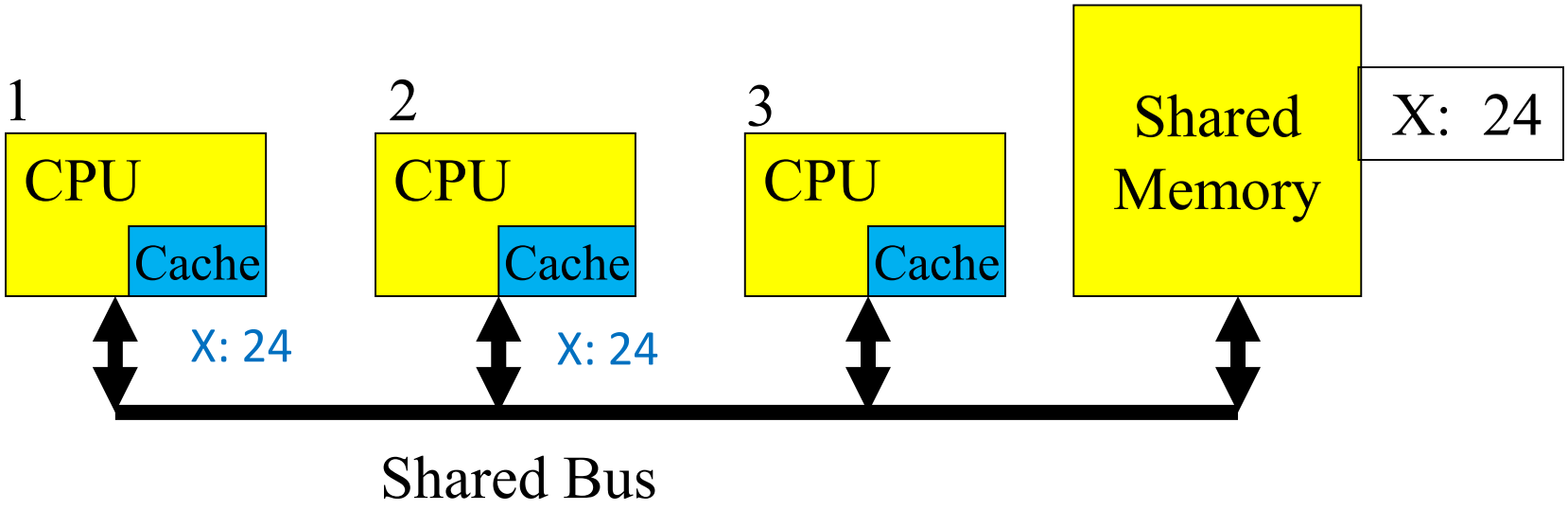
Update Cache 1 line with new value of X: Modified (M) state

Cache 1



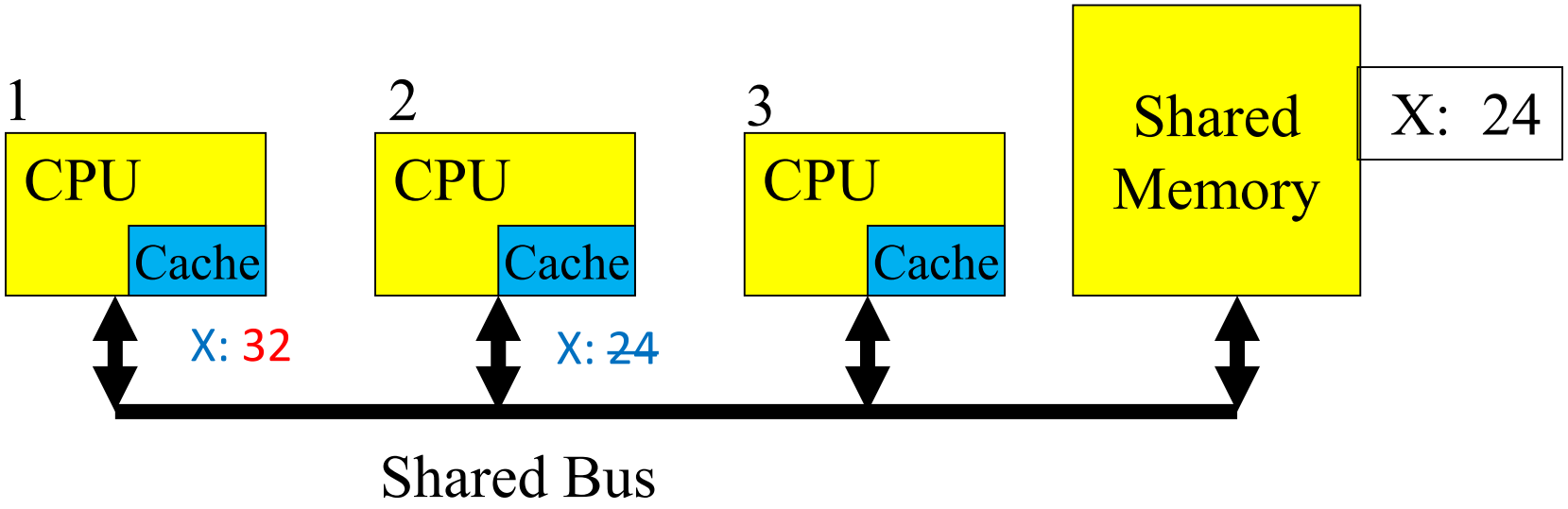
Local Write Hit (S)

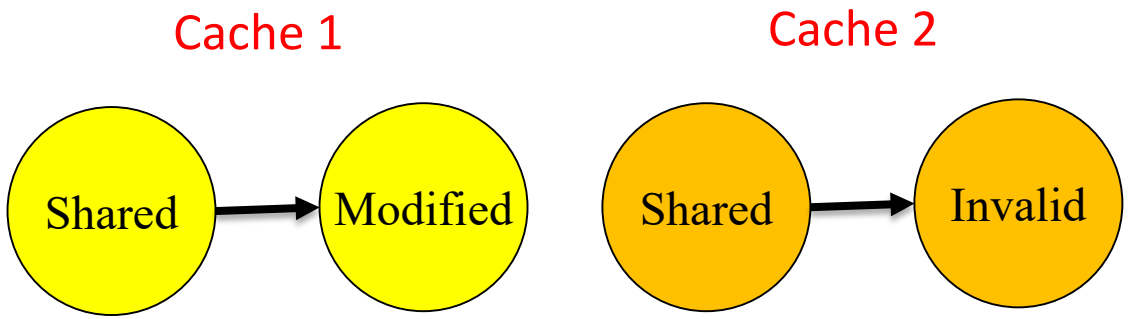
- **Cache line state S: cache line is same as main memory and there may be copies in other caches**
  - Processor broadcasts “invalidate” on bus
  - Snooping processors with S caches change cache line state to I (Invalid)
  - Local cache value is updated
  - Local cache line state changes to M (Modified)



Both Cache 1 and 2 have copies of X (Shared state S)

Cache 1 changes it value of X, so must "invalidate" copy of X in Cache 2


1
CPU
Cache
2
CPU
Cache
3
CPU
Cache
Shared Memory
X: 24
X: 32
X: 24
Shared Bus

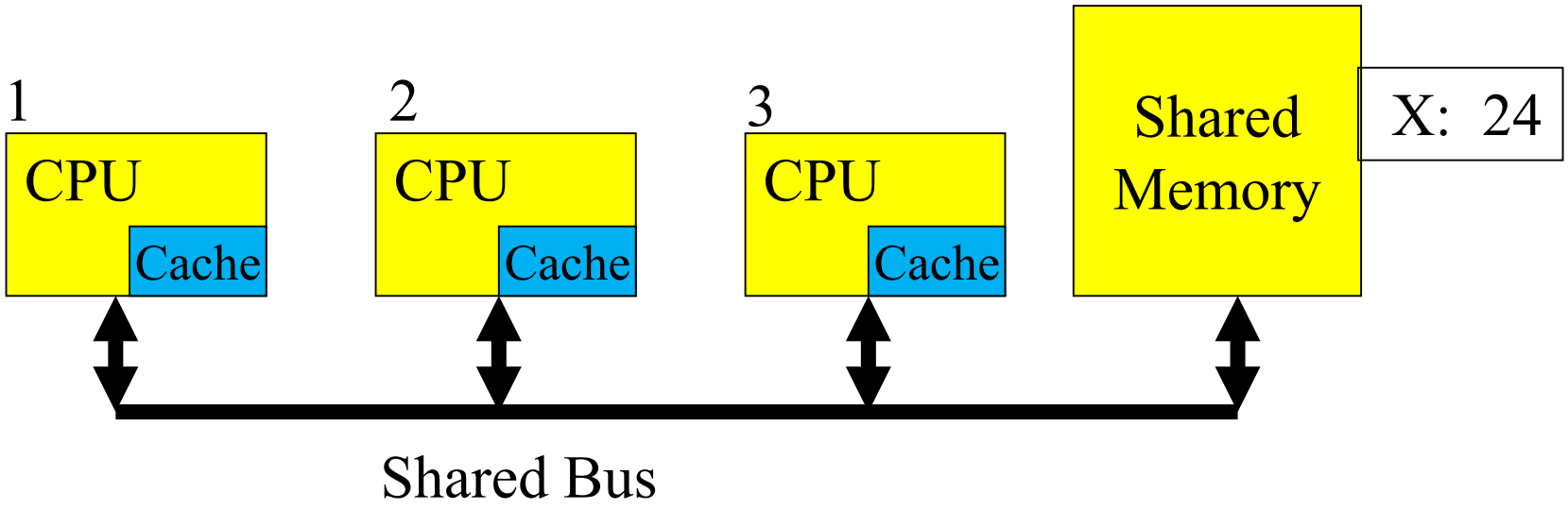

Cache 1
Shared
Modified
Cache 2
Shared
Invalid

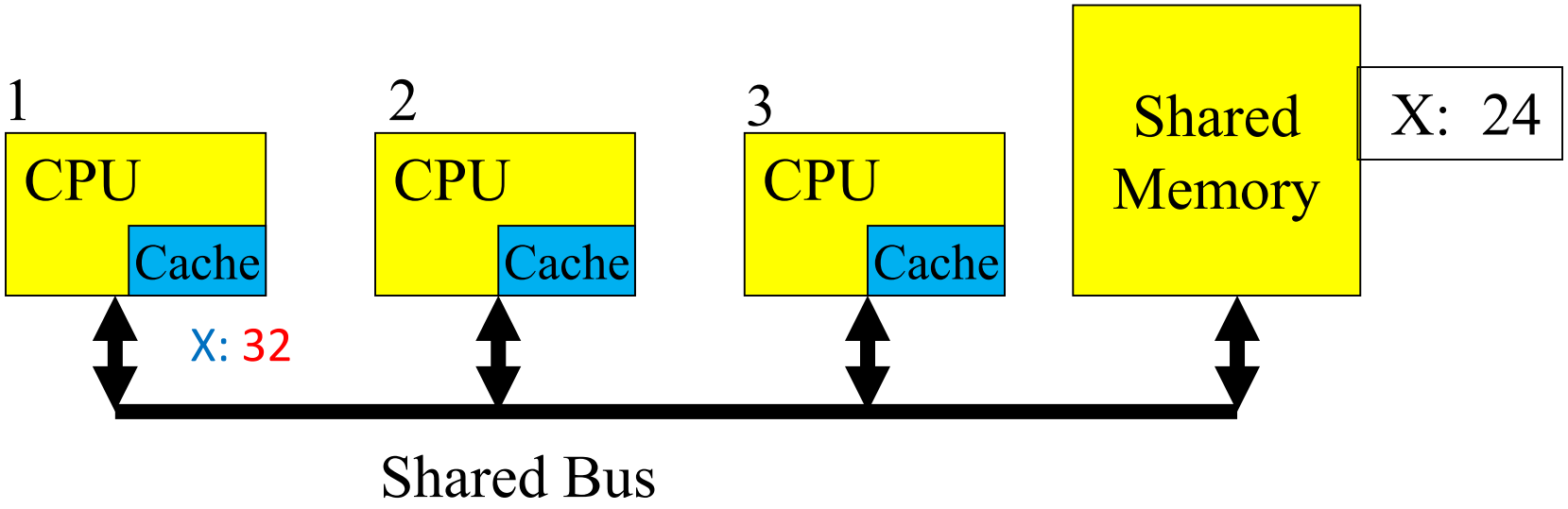
D. Local Write Miss

- Write Miss: address not found in cache
  - Cache line state is I (Invalid)
- Action depends on copies of X in other caches, if they exist

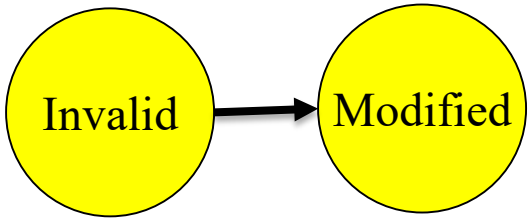
Local Write Miss (1)

- **If no copies of X exist in other caches**
  - Update value of X in local cache line
  - Set local cache state to M (modified)



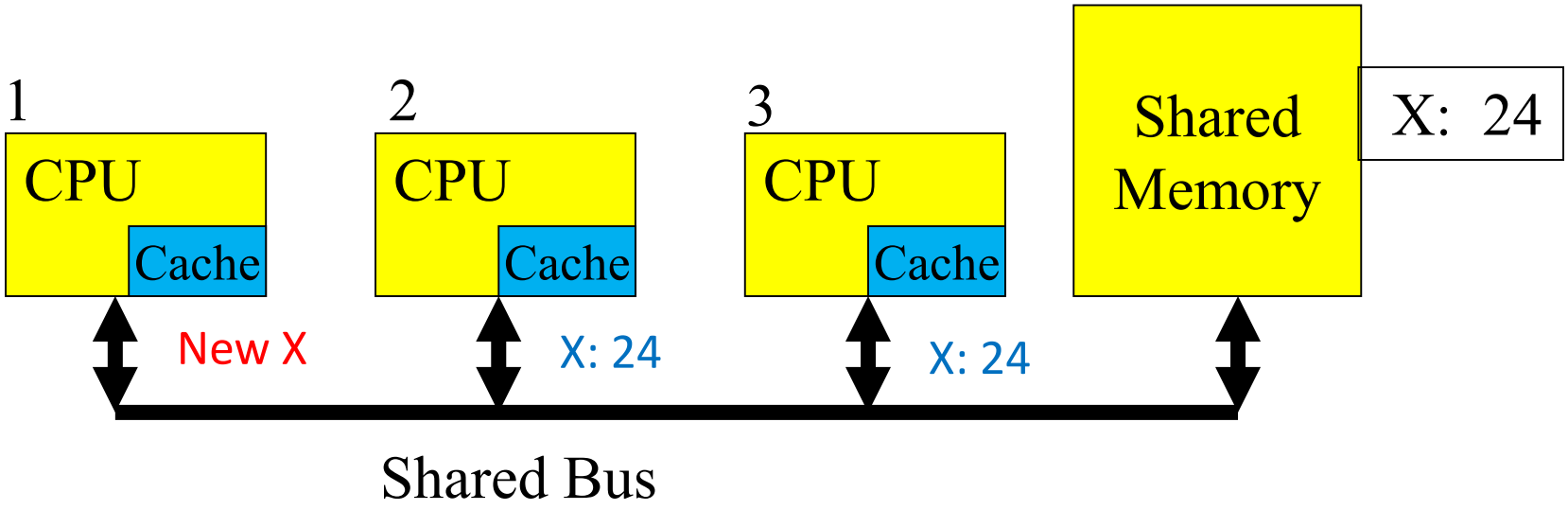

1
CPU
Cache
2
CPU
Cache
3
CPU
Cache
Shared Memory
X: 24
X: 32
Shared Bus

Cache 1


Invalid
Modified

Local Write Miss (2)

- **Else, if other copies, either one in state E or more in state S**
  - Value read from memory to local cache - bus transaction marked RWITM (read with intent to modify)
  - Snooping processors see this and set their copy state to I
  - Local copy updated & state set to M

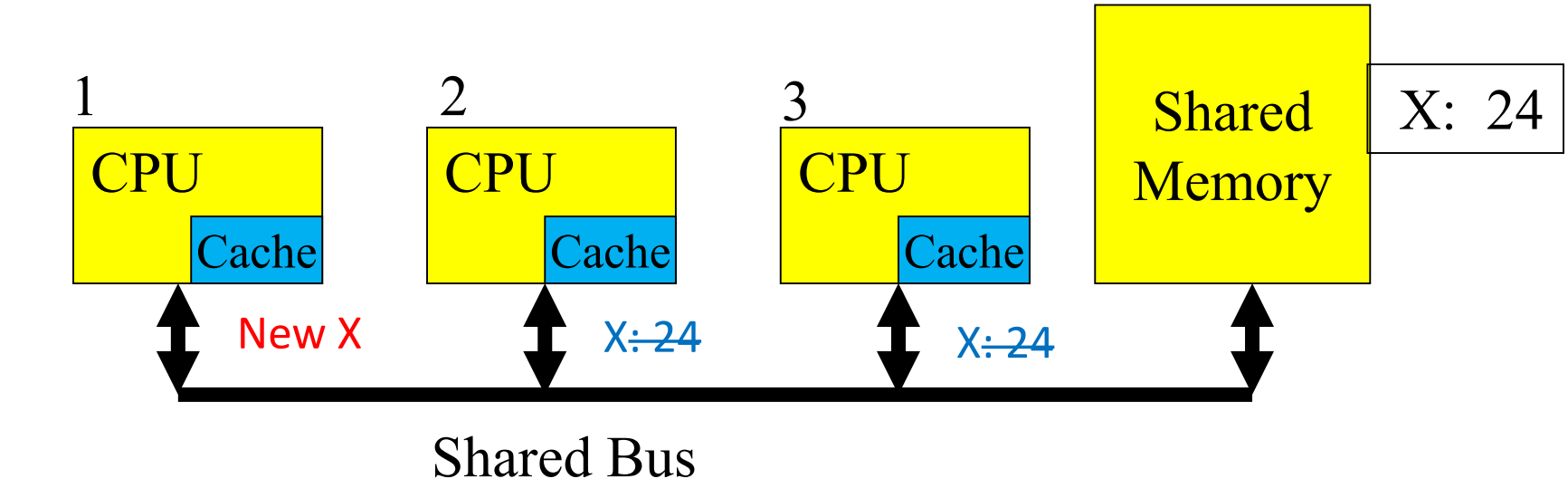


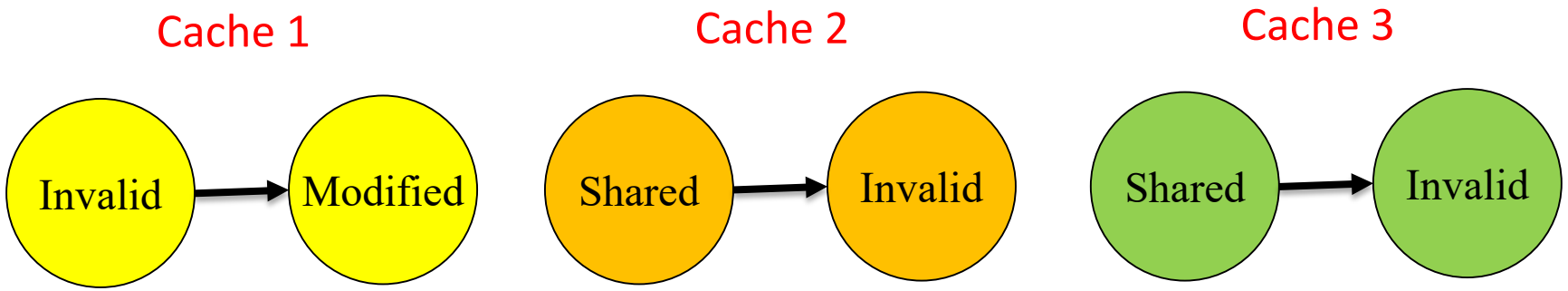
Caches 2 and 3 both have copy of current value of X

Cache 1 does NOT have copy of X, but wants to modify X. CPU 1 sends RWITM signal to bus

Caches 2 and 3 set their copies of X to "invalid"

Cache 1 modifies X


1
CPU
Cache
2
CPU
Cache
3
CPU
Cache
Shared Memory
X: 24
New X
X: 24
X: 24
Shared Bus


Cache 1
Invalid
Modified
Cache 2
Shared
Invalid
Cache 3
Shared
Invalid