Multiprocessors

Introduction

Multiprocessor : Two or more CPUs communicate with memory and I/O through interconnection network / structure.

Types of Multiprocessors

- Categories into 2 types: Tightly coupled and loosely coupled multiprocessor system.
- The degree of coupling is high in the tightly coupled system, whereas the degree of coupling is low n loosely coupled system.
- **Tightly coupled** → Multiprocessor system has shared memory.
- It is classified into 3 categories

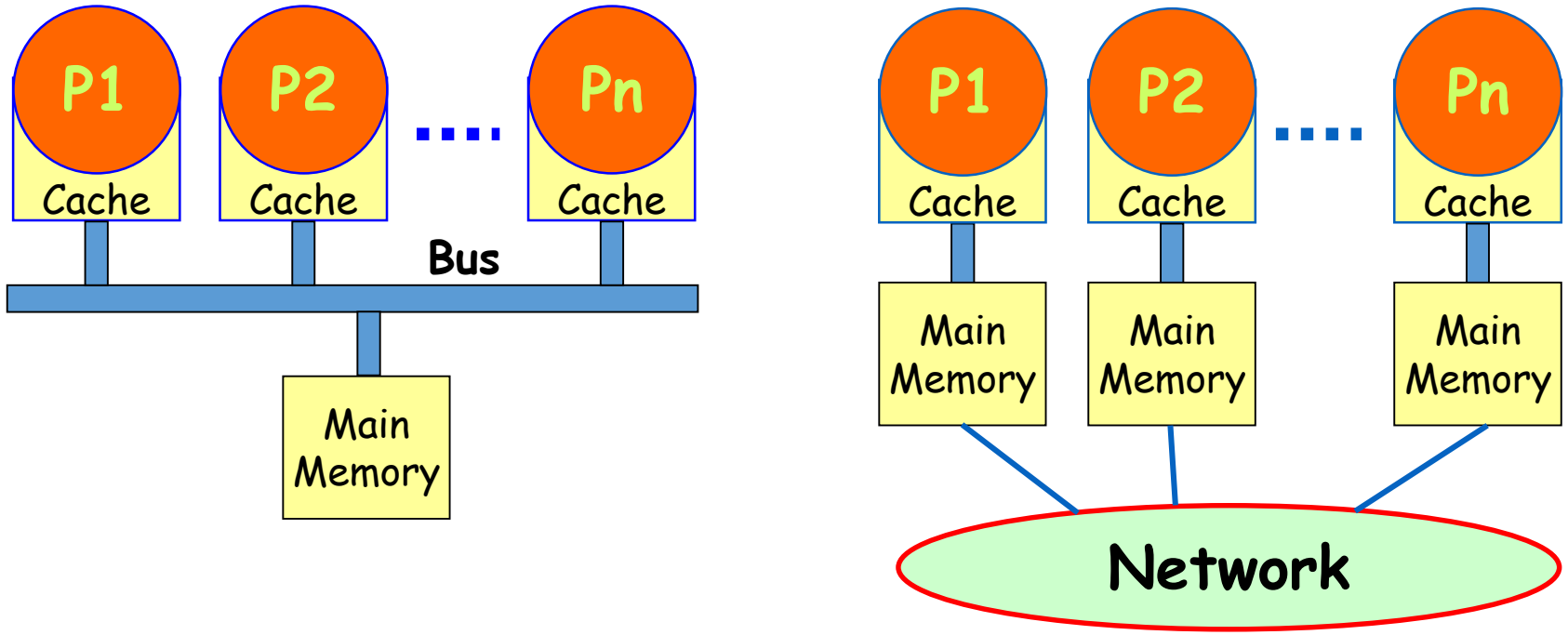
  UMA(Uniform Memory Architecture)

  NUMA(Non Uniform Memory Architecture)

  COMA(Cache Only Memory Architecture)

- **Loosely coupled**→ Multiprocessor system has distributed memory. Each processor has its own memory

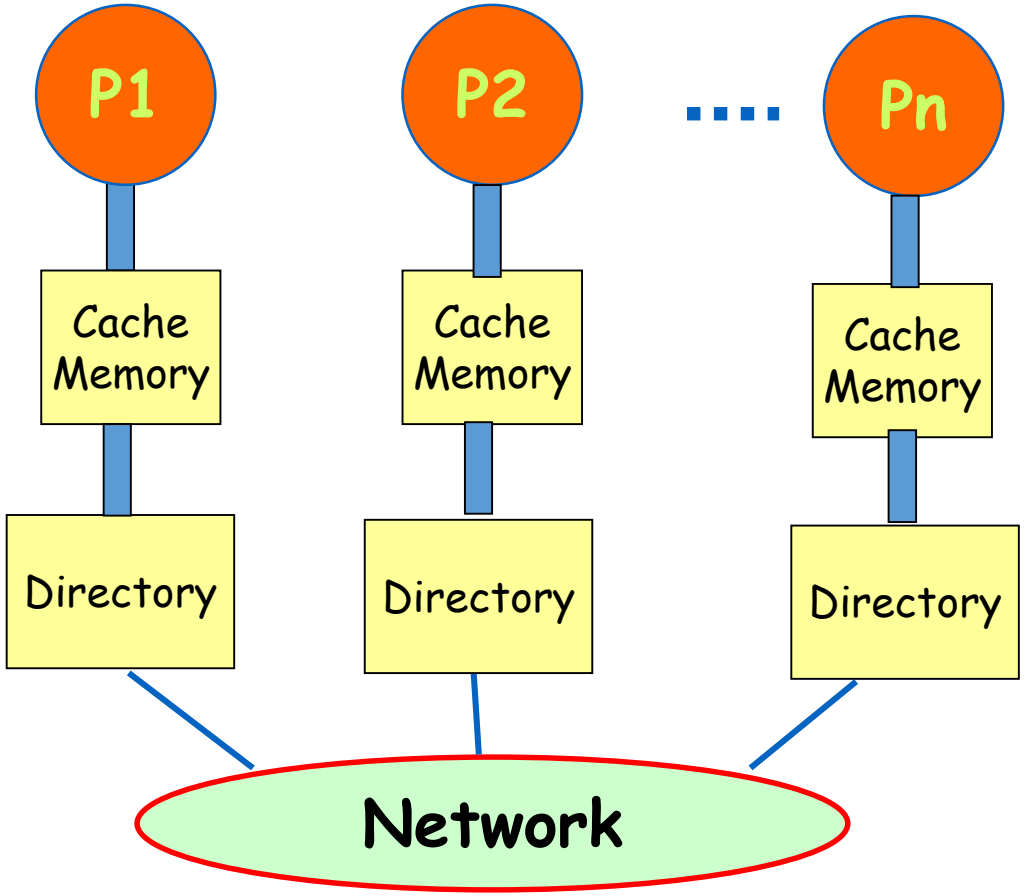
UMA and NUMA Computers



(a) UMA Model

(b) NUMA Model

COMA Computers



(c) COMA Model

UMA (Uniform Memory Architecture)

- In UMA model of multiprocessor, physical memory is uniformly shared by all the processors.
- All processors have equal access time to all memory words, so it is called UMA.
- It is also called Symmetric Shared Memory Architecture.
- Here peripherals (I/O) are also shared in same fashion.
- It is also called as tightly coupled systems due to the high degree of resource sharing.
- UMA is two types: Symmetric Multiprocessor and Asymmetric Multiprocessor.
- In Symmetric UMA, all processors have equal access time to all peripheral devices.
- In asymmetric UMA, only one or subset of processors are capable to execute the peripheral device. A master processor can execute the OS and handle I/O. Attached processor has no I/O capability and execute user codes under the supervision of master processor.

NUMA (Non-Uniform Memory Architecture)

- In NUMA model, access time varies with the location of the memory word.
- It is also called Distributed Shared-memory (DSM) architectures
- Local memories: shared memory is physically distributed among all the processors.
- Global address space is formed by collection of all local memories that is accessible by all processors
- Faster access to a local memory with a local processor
- Slow access to remote memory attached to other processors due to added delay through the interconnection network.

COMA (Cache Only Memory Architecture)

- Coma model is a special case of the NUMA model. Here all the distributed main memories are converted into cache memories.
- Cache form Global address space.
- Remote cache access by distributed cache directories.

Distributed Memory Computers

- Distributed memory computers use:
  - Message Passing Model
- Explicit message send and receive instructions have to be written by the programmer.
  - Send: specifies local buffer + receiving process (id) on remote computer (address).
  - Receive: specifies sending process on remote computer + local buffer to place data.

Advantages of Message-Passing Communication

- Hardware for communication and synchronization are much simpler:
  - Compared to communication in a shared memory model.
- Explicit communication:
  - Programs simpler to understand, helps to reduce maintenance and development costs.
- Synchronization is implicit:
  - Naturally associated with sending/receiving messages.
  - Easier to debug.

Disadvantages of Message-Passing Communication

- Programmer has to write explicit message passing constructs.
  - Also, precisely identify the processes (or threads) with which communication is to occur.
- Explicit calls to operating system:
  - Higher overhead.

Distributed Shared-Memory (DSM)

- Physically separate memories are accessed as one logical address space.
- Processors running on a multi-computer system share their memory.
  - Implemented by operating system.
- DSM multiprocessors are NUMA:
  - Access time depends on the exact location of the data.

Distributed Shared-Memory Architecture (DSM)

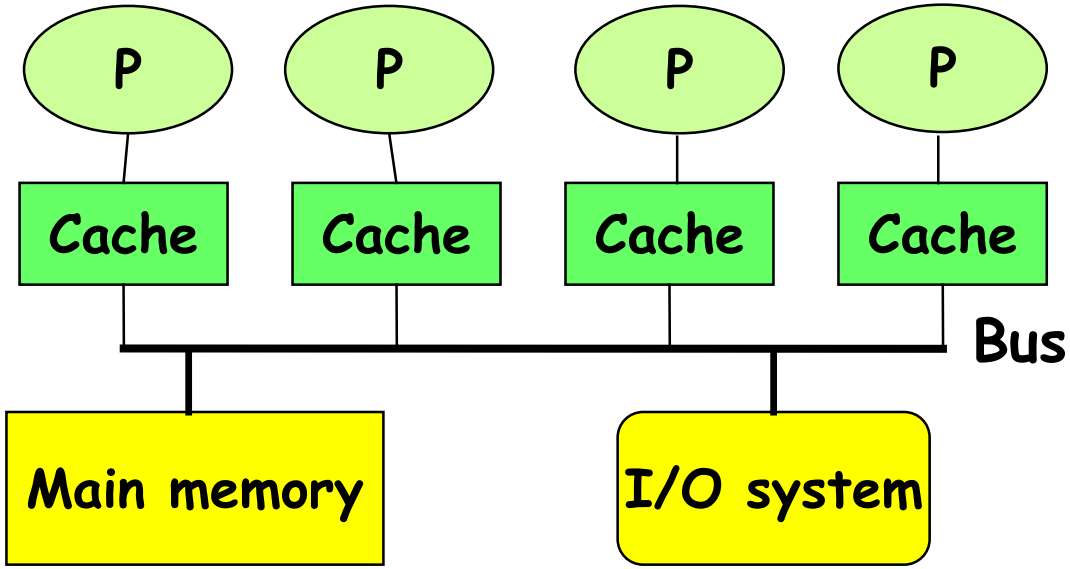
- Underlying mechanism is message passing:
  - Shared memory convenience provided to the programmer by the operating system.
  - Basically, an operating system facility takes care of message passing implicitly.
- Advantage of DSM:
  - Ease of programming

Disadvantage of DSM

- High communication cost:
  - A program not specifically optimized for DSM by the programmer shall perform extremely poorly.
  - Data (variables) accessed by specific program segments have to be collocated.
  - Useful only for process-level (coarse-grained) parallelism.

Symmetric Multiprocessors (SMPs)

- SMPs are a popular shared memory multiprocessor architecture:
  - Processors share Memory and I/O
  - Bus based: access time for all memory locations is equal --- “Symmetric MP”



SMPs: Some Insights

- In any multiprocessor, main memory access is a bottleneck:
  - Multilevel caches reduce the memory demand of a processor.
  - Multilevel caches in fact make it possible for more than one processor to meaningfully share the memory bus.
  - Hence multilevel caches are a must in a multiprocessor!

Different SMP Organizations

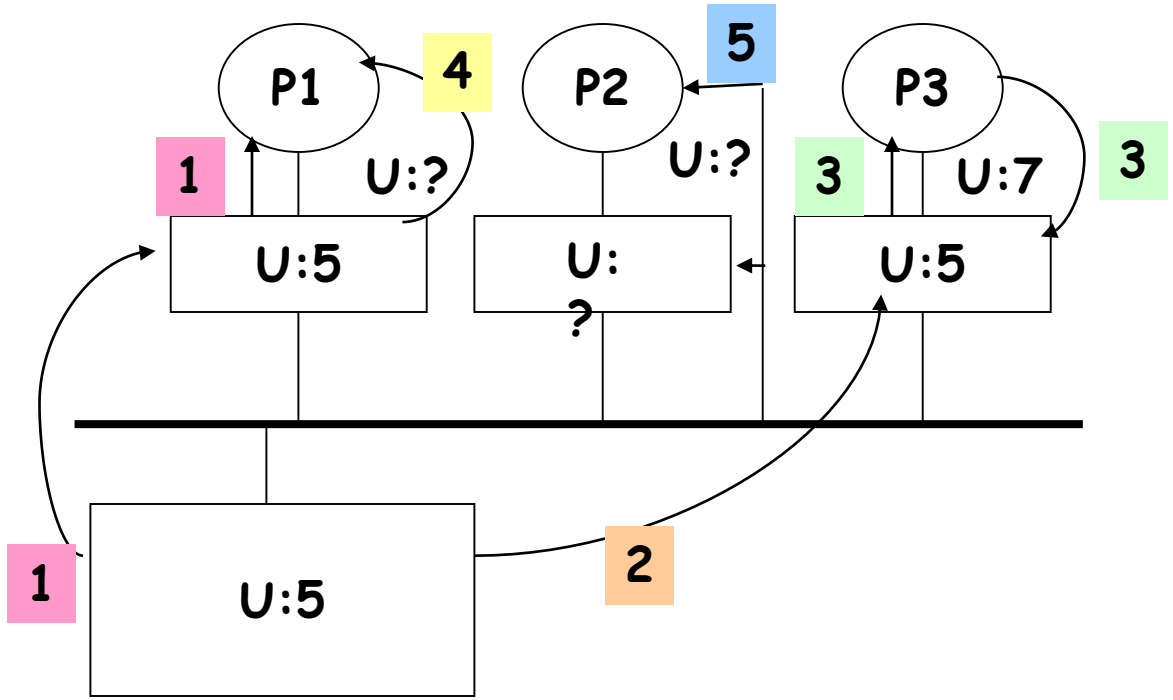
- Processor and cache on separate extension boards (1980s):
  - Plugged on to the backplane.
- Integrated on the main board (1990s):
  - 4 or 6 processors placed per board.
- Integrated on the same chip (multi-core) (2000s):
  - Dual core (IBM, Intel, AMD)
  - Quad core

- Pros of SMPs
- Ease of programming:
  - Especially when communication patterns are complex or vary dynamically during execution.
- Cons of SMPs
- As the number of processors increases, contention for the bus increases.
  - Scalability of the SMP model restricted.
  - One way out may be to use switches (crossbar, multistage networks, etc.) instead of a bus.
  - Switches set up parallel point-to-point connections.
  - Again switches are not without any disadvantages: make implementation of cache coherence difficult.

Cache Coherence Protocols

A memory system is coherent if any read of a data item returns the mostly recently written value of the data item.

The Cache Coherence Problem:

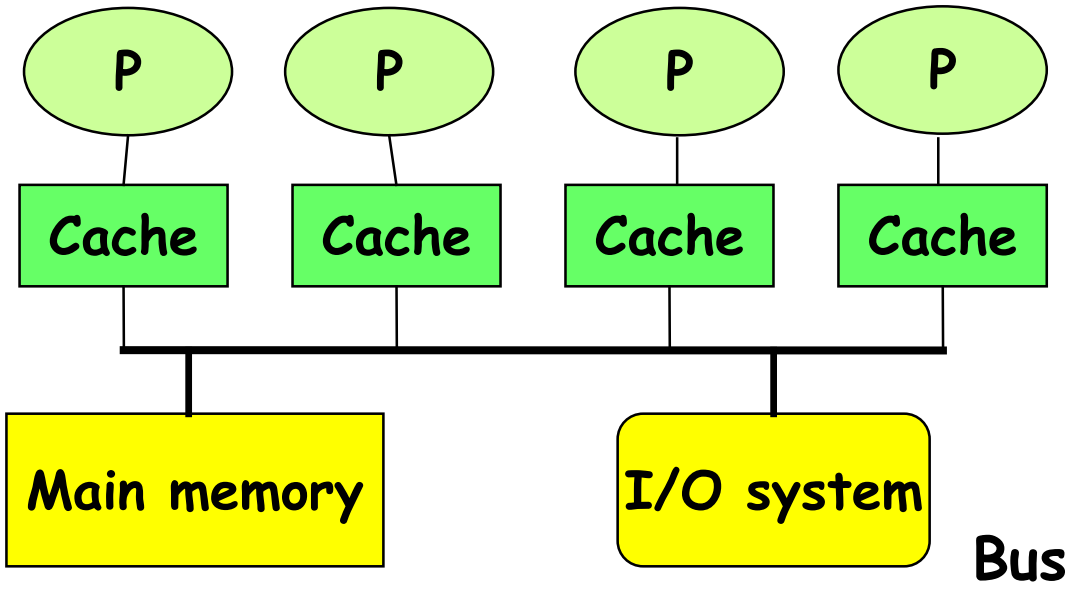


What value will P1 and P2 read?

Cache Coherence Solutions (Protocols)

- The key to maintain cache coherence:
  - Track the state of sharing of every data block.
- Based on this idea, following can be an overall solution:
  - Dynamically recognize any potential inconsistency at run-time and carry out preventive action.

Basic Idea Behind Cache Coherency Protocols



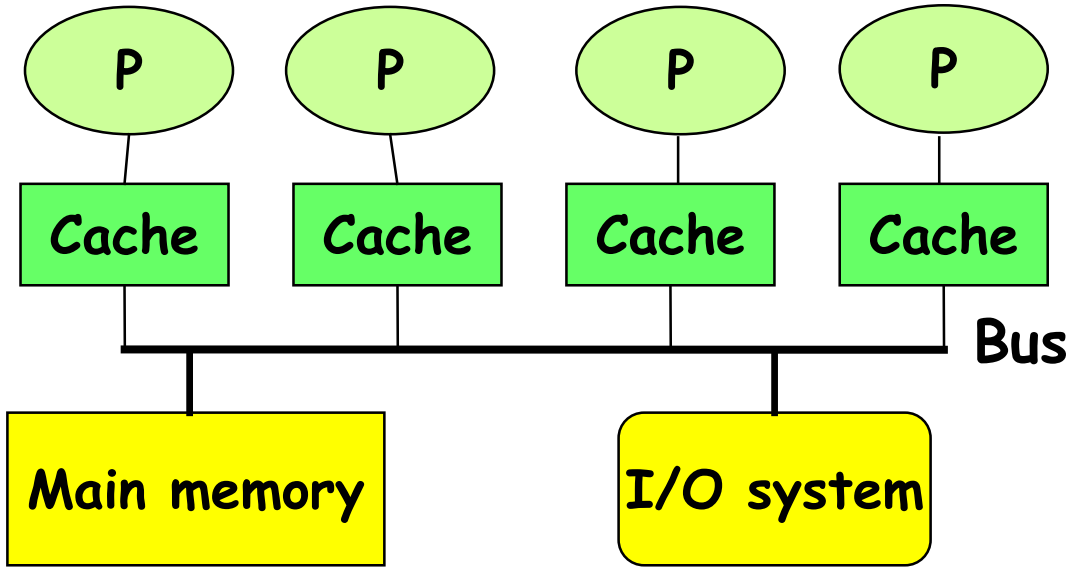
Pros and Cons of the Solution

- Pro:
  - Consistency maintenance becomes transparent to programmers, compilers, as well as to the operating system.
- Con:
  - Increased hardware complexity .

Two Important Cache Coherency Protocols

- Snooping protocol:
  - Each cache “snoops” the bus to find out which data is being used by whom.
- Directory-based protocol:
  - Keep track of the sharing state of each data block using a directory.
  - A directory is a centralized register for all memory blocks.
  - Allows coherency protocol to avoid broadcasts.

Snoopy and Directory-Based Protocols



Snooping vs. Directory-based Protocols

- Snooping protocol reduces memory traffic.
  - More efficient.
- Snooping protocol requires broadcasts:
  - Can meaningfully be implemented only when there is a shared bus.
  - Even when there is a shared bus, scalability is a problem.
  - Some work arounds have been tried: Sun Enterprise server has up to 4 buses.

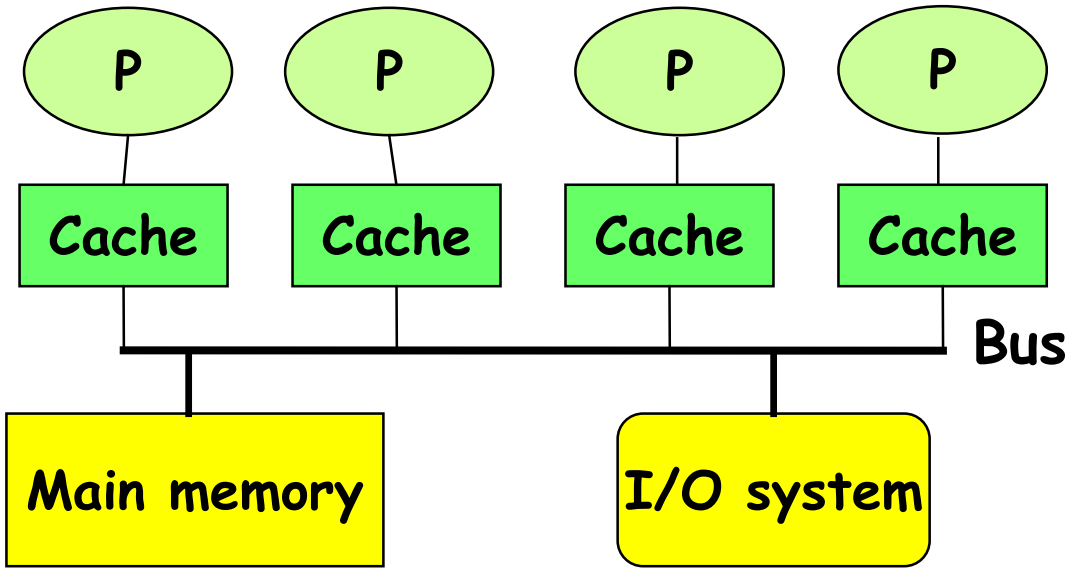
Snooping Protocol

- As soon as a request for any data block by a processor is put out on the bus:
  - Other processors “snoop” to check if they have a copy and respond accordingly.
- Works well with bus interconnection:
  - All transmissions on a bus are essentially broadcast:
    - Snooping is therefore effortless.
  - Dominates almost all small scale machines.

Categories of Snoopy Protocols

- Essentially two types:
  - Write Invalidate Protocol
  - Write Broadcast Protocol
- Write invalidate protocol:
  - When one processor writes to its cache, all other processors having a copy of that data block invalidate that block.
- Write broadcast:
  - When one processor writes to its cache, all other processors having a copy of that data block update that block with the recent written value.

Write Invalidate Vs. Write Update Protocols



Write Invalidate Protocol

- Handling a write to shared data:
  - An invalidate command is sent on bus --- all caches snoop and invalidate any copies they have.
- Handling a read Miss:
  - Write-through: memory is always up-to-date.
  - Write-back: snooping finds most recent copy.

Write Invalidate in Write Through Caches

- Simple implementation.
- Writes:
  - Write to shared data: broadcast on bus, processors snoop, and update any copies.
  - Read miss: memory is always up-to-date.
- Concurrent writes:
  - Write serialization automatically achieved since bus serializes requests.
  - Bus provides the basic arbitration support.

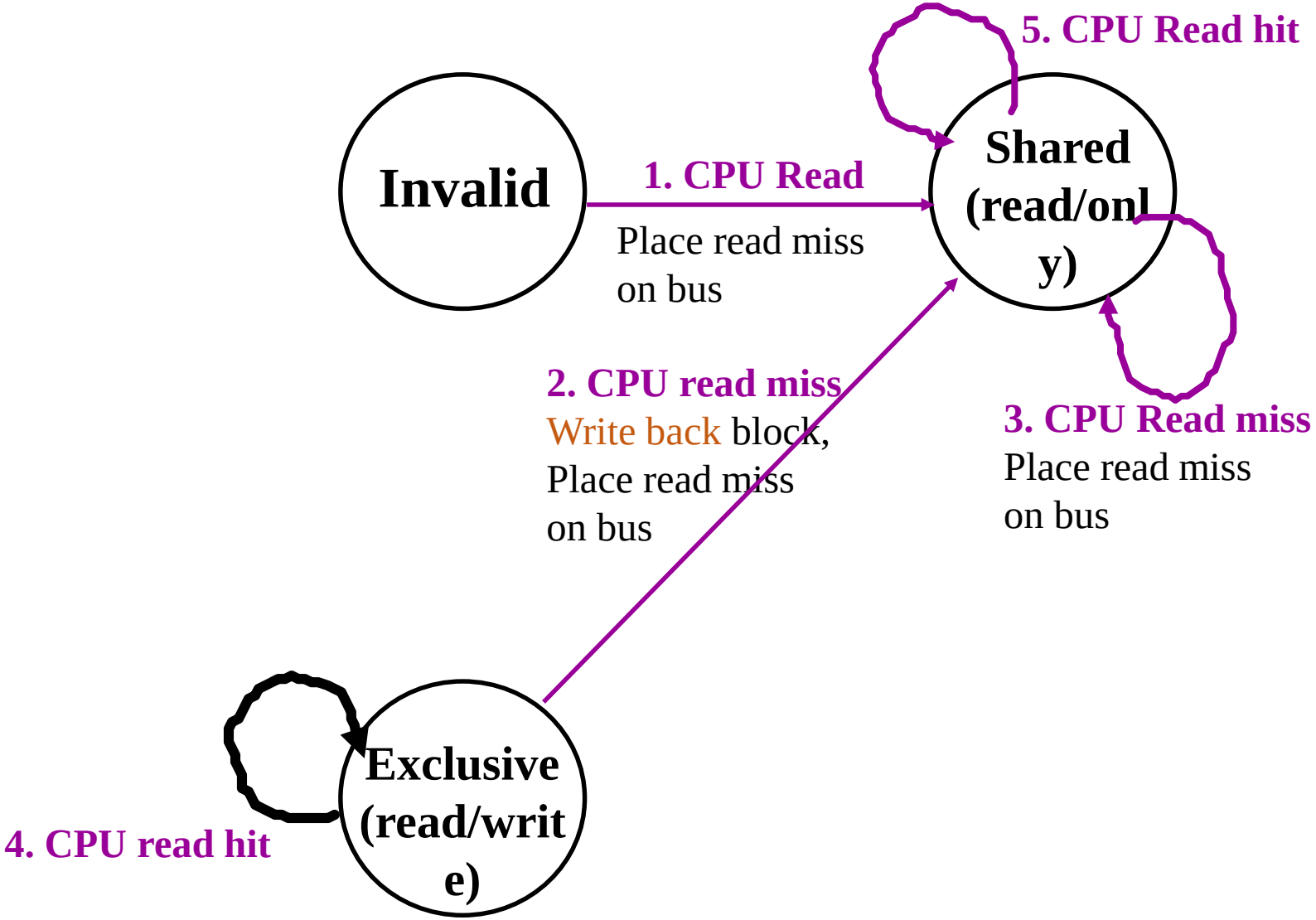
Write Invalidate Vs Broadcast

- Invalidate exploits spatial locality:
  - Only one bus transaction for any number of writes to the same block.
  - Obviously, more efficient.
- Broadcast has lower latency for writes and reads:
  - As compared to invalidate.

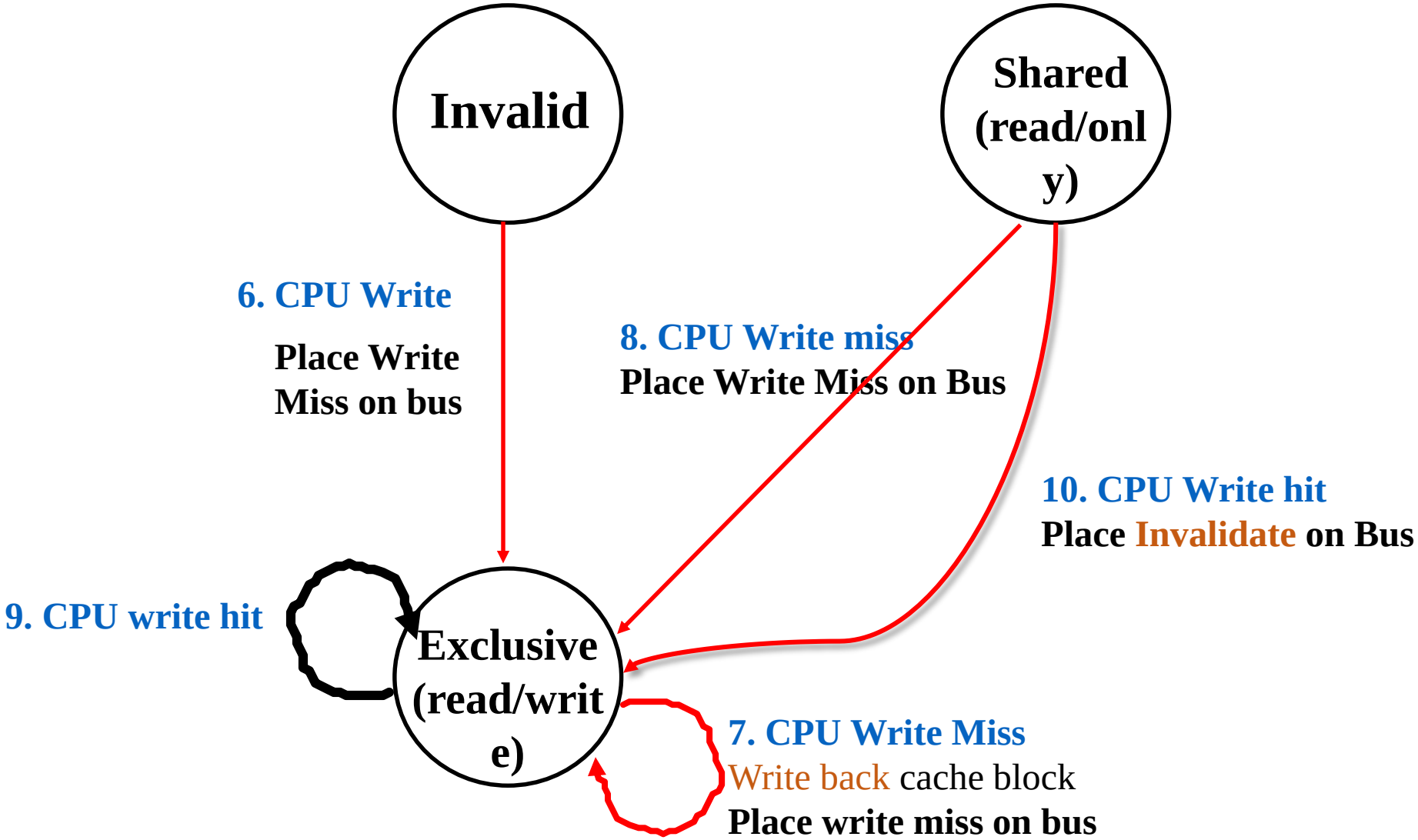
An Example Snoopy Protocol

- Assume:
  - Invalidation protocol, write-back cache.
- Each block of memory is in one of the following states:
  - Shared: Clean in all caches and up-to-date in memory, block can be read.
  - Exclusive: cache has the only copy, it is writeable, and dirty.
  - Invalid: Data present in the block obsolete, cannot be used.

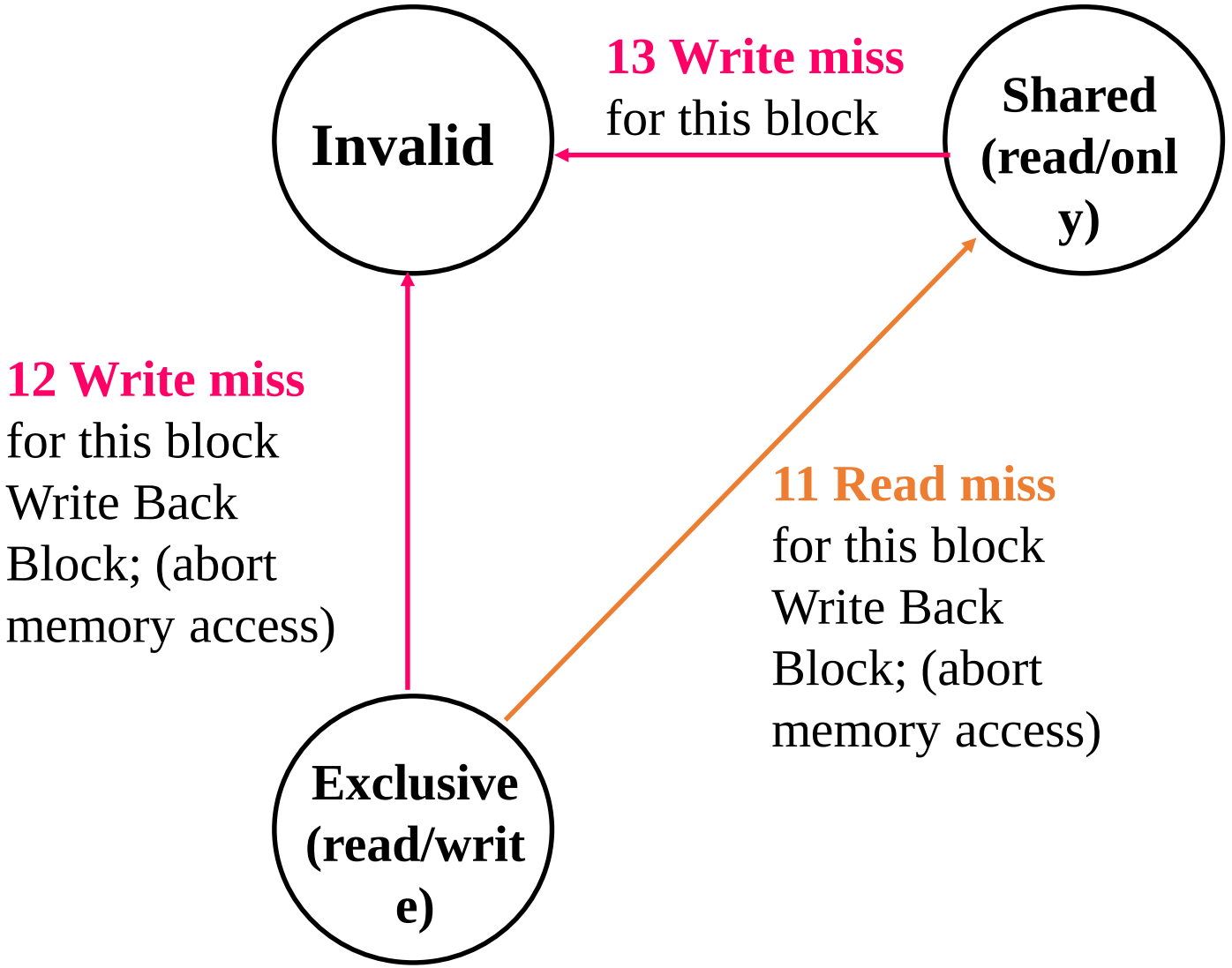
- State machine considering only CPU read requests on each cache block.



- State machine considering only CPU write requests on each cache block.

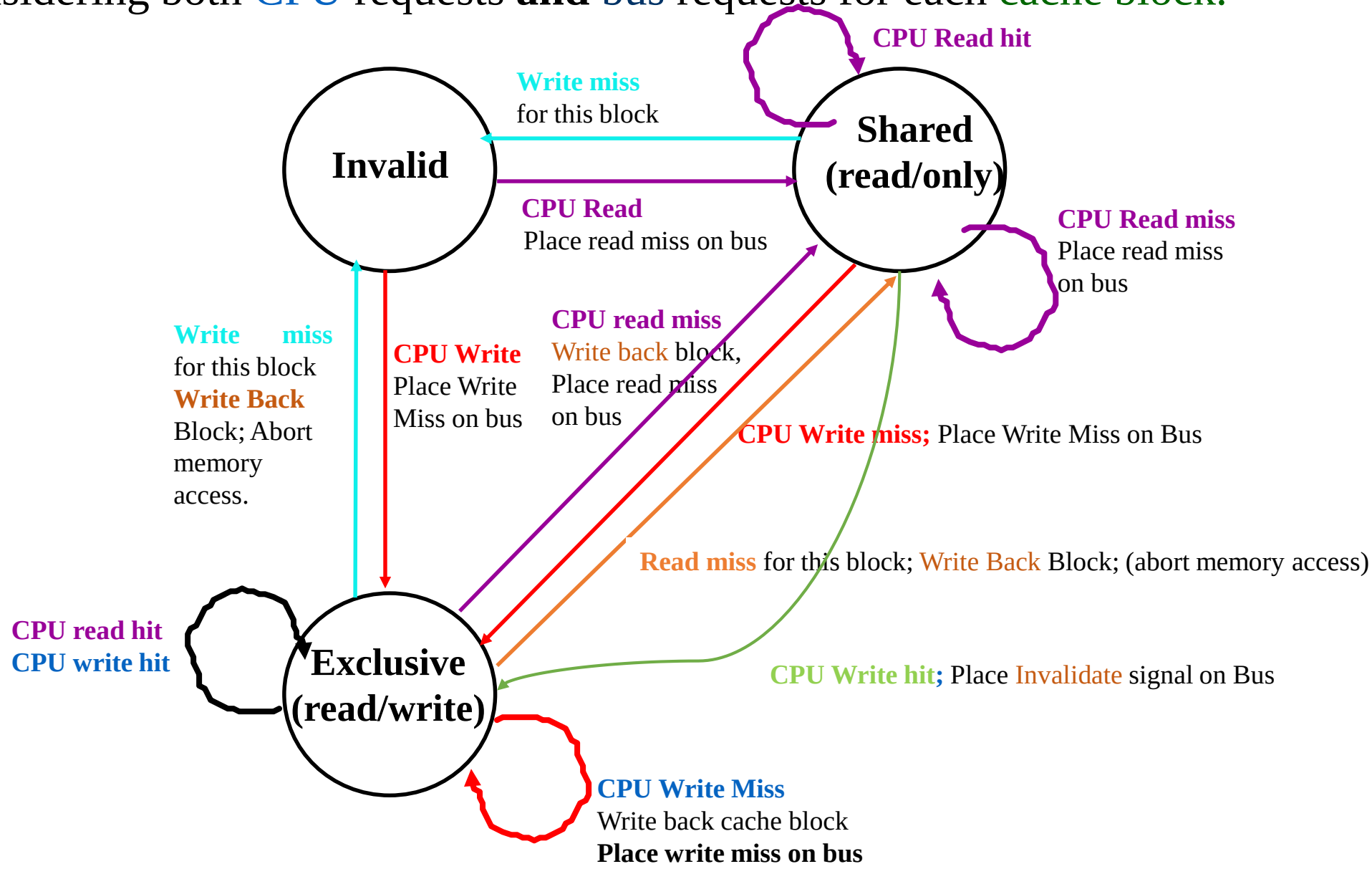


- State machine considering only bus requests for each cache block.



Combined Snoopy-Cache State Machine

- State machine considering both CPU requests **and** bus requests for each cache block.



Directory-based Solution

- In NUMA computers:
  - Messages have long latency.
  - Also, broadcast is inefficient --- all messages have explicit responses.
- Main memory controller to keep track of:
  - Which processors are having cached copies of which memory locations.
- On a write,
  - Only need to inform users, not everyone
- On a dirty read,
  - Forward to owner

Directory Protocol

- Three states as in Snoopy Protocol
  - Shared: 1 or more processors have data, memory is up-to-date.
  - Uncached: No processor has the block.
  - Exclusive: 1 processor (owner) has the block.
- In addition to cache state,
  - Must track which processors have data when in the shared state.
  - Usually implemented using bit vector, 1 if processor has copy.

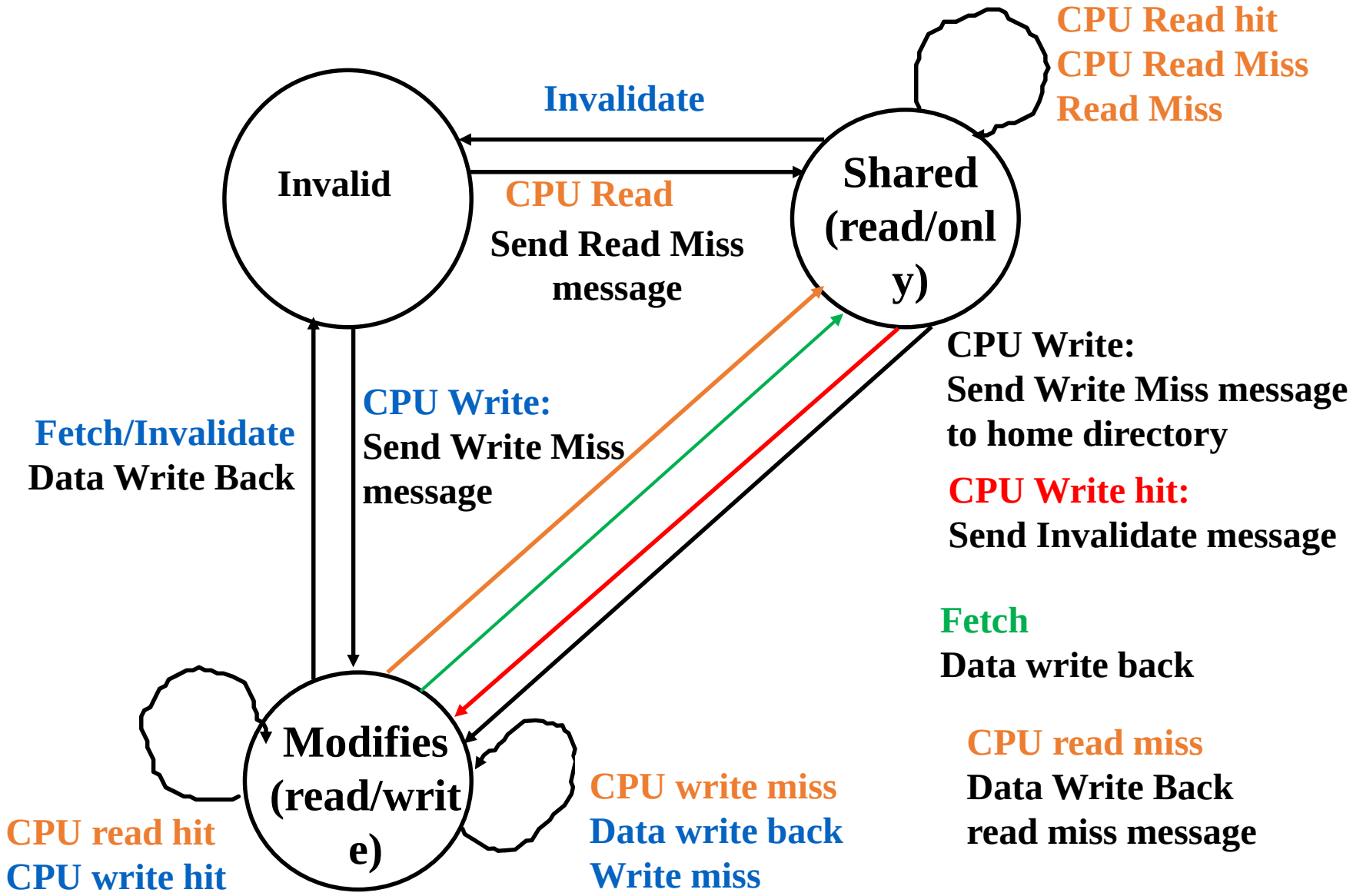
Directory Behavior

- On a read:
  - Unused:
    - give (exclusive) copy to requester
    - record owner
  - Exclusive or shared:
    - send share message to current exclusive owner
    - record owner
    - return value
  - Exclusive dirty:
    - forward read request to exclusive owner.

Directory Behavior

- On Write
  - Send invalidate messages to all hosts caching values.
- On Write-Thru/Write-back
  - Update value.

- State machine for an individual cache block in a directory based system



State Transition Diagram for the Directory

- Tracks all copies of memory block.
- Same states as the transition diagram for an individual cache.
- Memory controller actions:
  - Update of directory state
  - Send messages to satisfy requests.
  - Also indicates an action that updates the sharing set, Sharers, as well as sending a message.

State machine for Directory based protocol