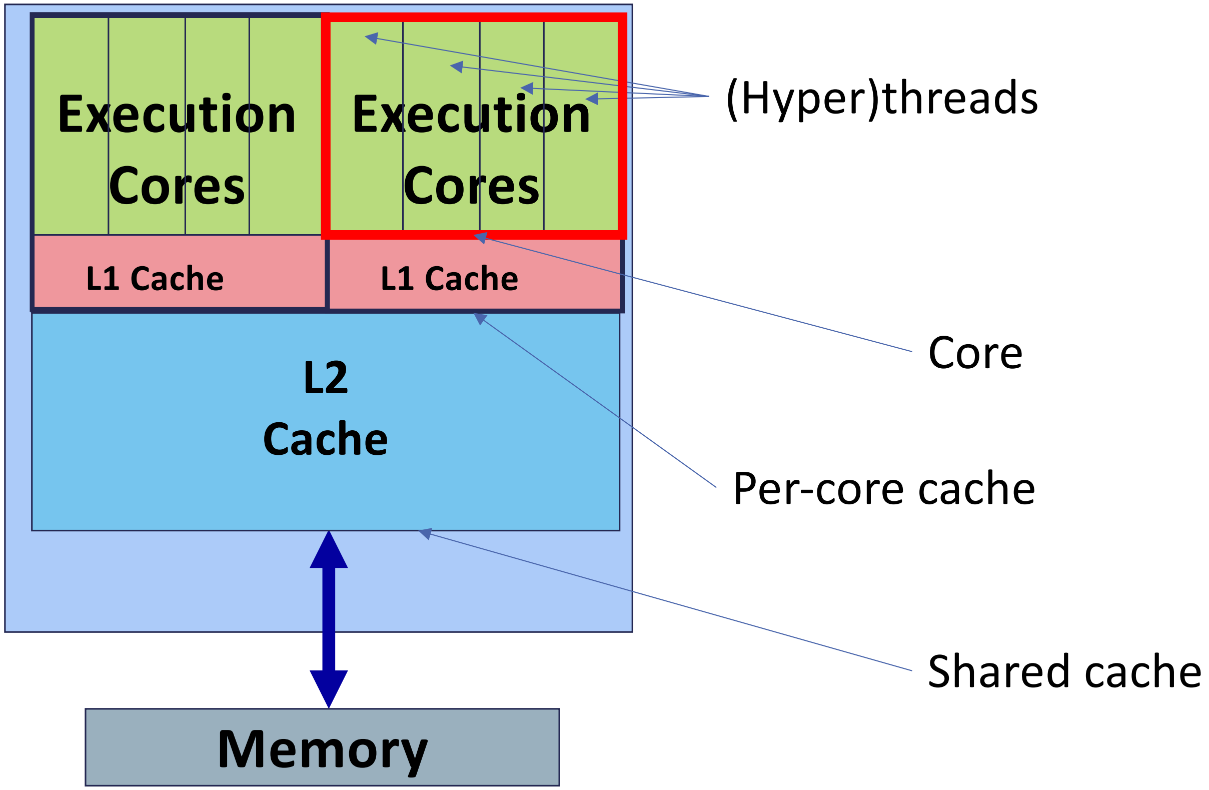
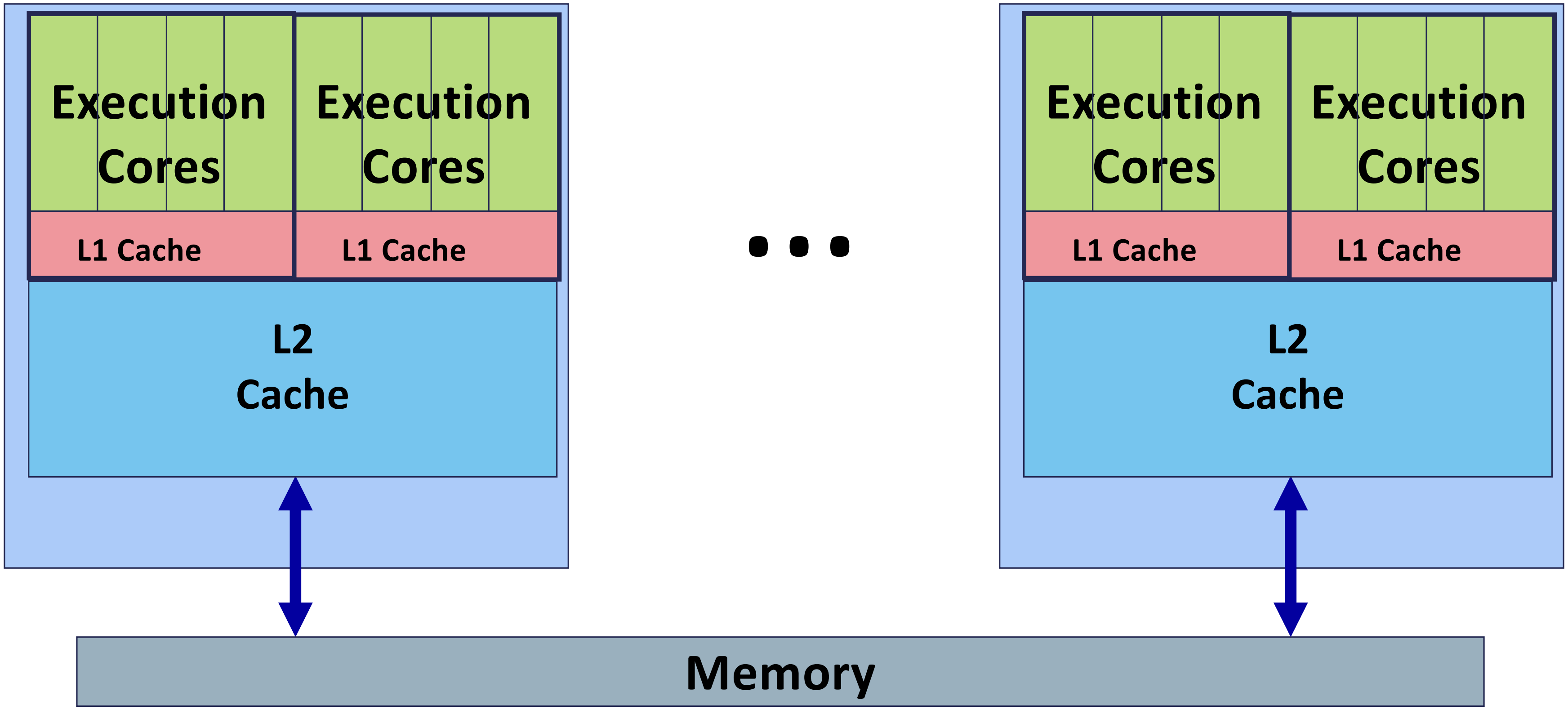
Today

- Learning Outcomes for Today
  - Explain the challenges that arise in hardware caches in the presence of multiple cores and/or processors.
  - Define each of the states in the MSI protocols
  - Relate multi-core cache behavior to concurrency control in multi-threaded programs.
- This is not in the book, but you can find good writeups in Wikipedia and elsewhere.

Recall:


Execution Cores
Execution Cores
L1 Cache
L1 Cache
L2 Cache
Memory
(Hyper)threads
Core
Per-core cache
Shared cache

Recall:


Execution Cores
Execution Cores
L1 Cache
L1 Cache
L2 Cache
Execution Cores
Execution Cores
L1 Cache
L1 Cache
L2 Cache
…
Memory

Let’s assume all caches are write-back

- What problems do you see that we have to solve when we have caches in the presence of multiple processors (or cores)?

Let’s assume all caches are write-back

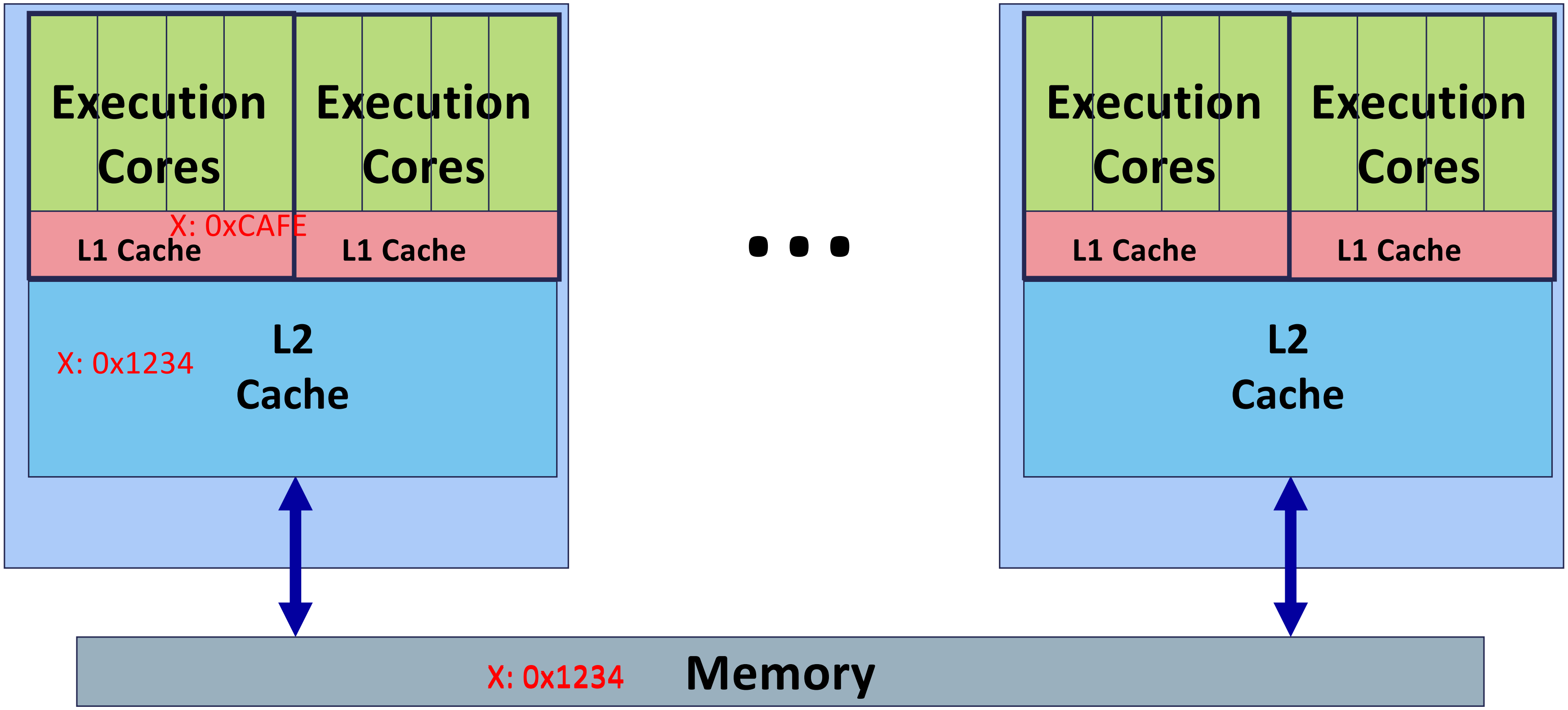
- What problems do you see that we have to solve when we have caches in the presence of multiple processors (or cores)?
    - Two cores/processors might access the same data – that could result in two copies of the same cache line present in two different caches.
    - Core 1 might wish to access data that is dirty in Core 2’s cache.
    - Processor 2 might wish to read data that is dirty in a cache on Processor 1.

Let’s assume all caches are write-back

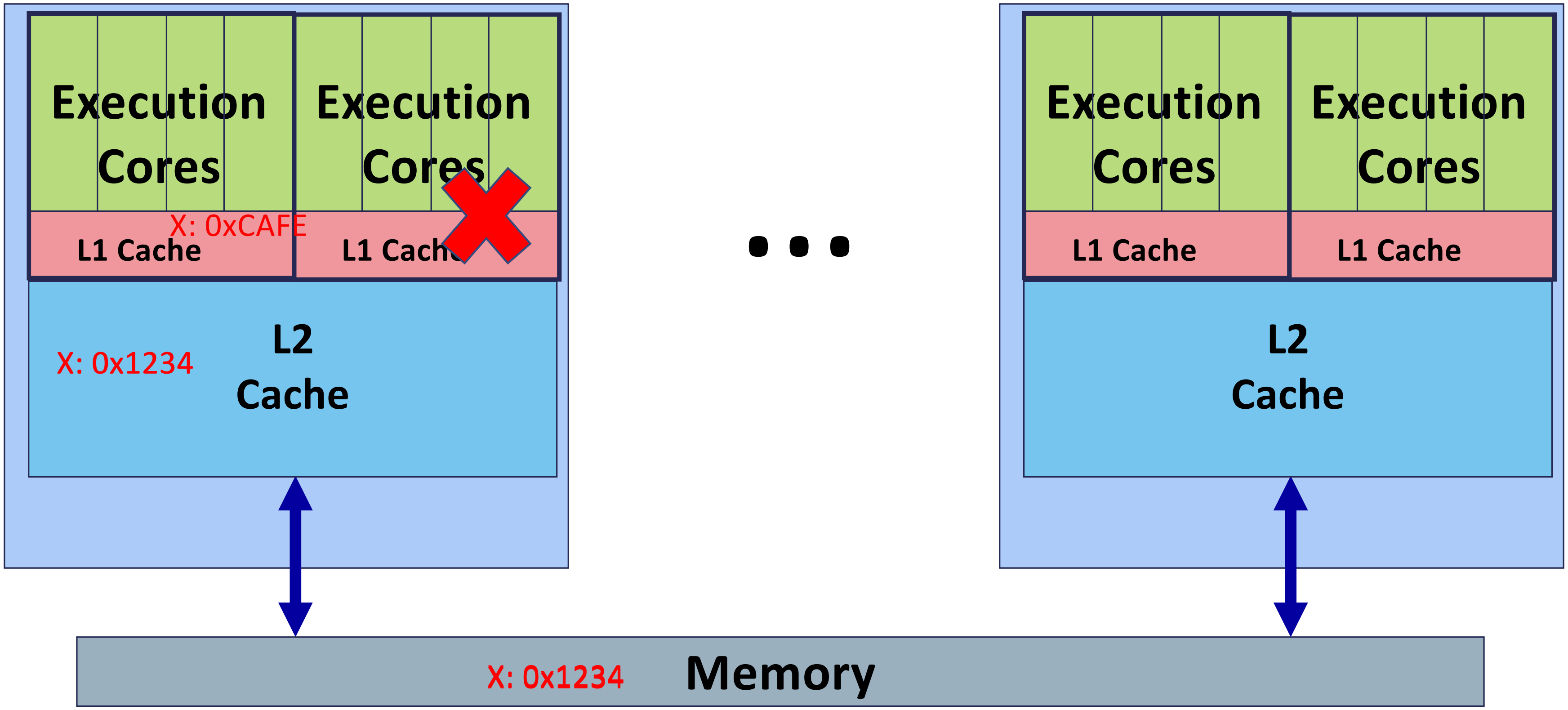
- What problems do you see that we have to solve when we have caches in the presence of multiple processors (or cores)?
  - Two cores/processors might access the same data – that could result in two copies of the same cache line present in two different caches.
  - Core 1 might wish to access data that is dirty in Core 2’s cache.
  - Processor 2 might wish to read data that is dirty in a cache on Processor 1.

If the hardware doesn’t do anything to avoid it, cores/processors can see incorrect data.

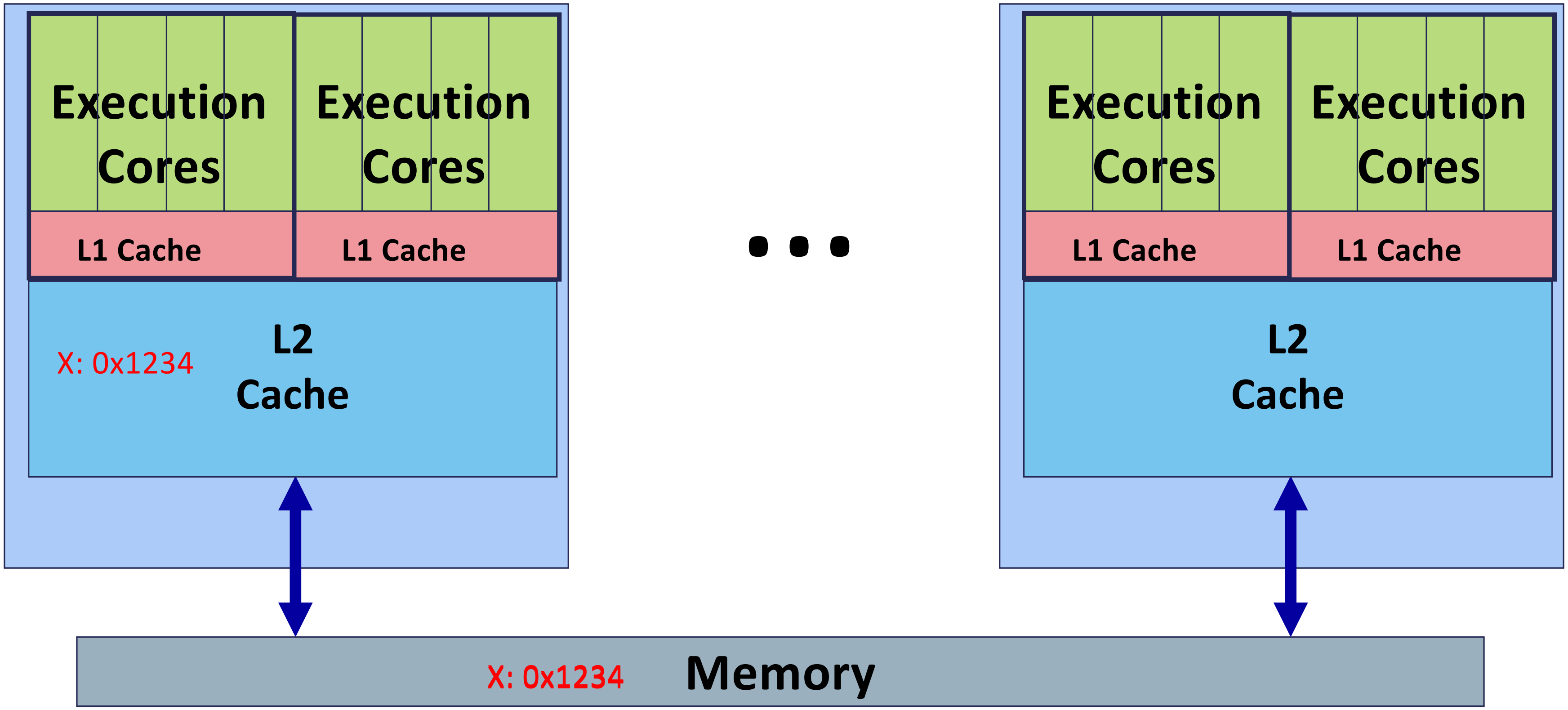
Core 1 reads X Core 1 writes X



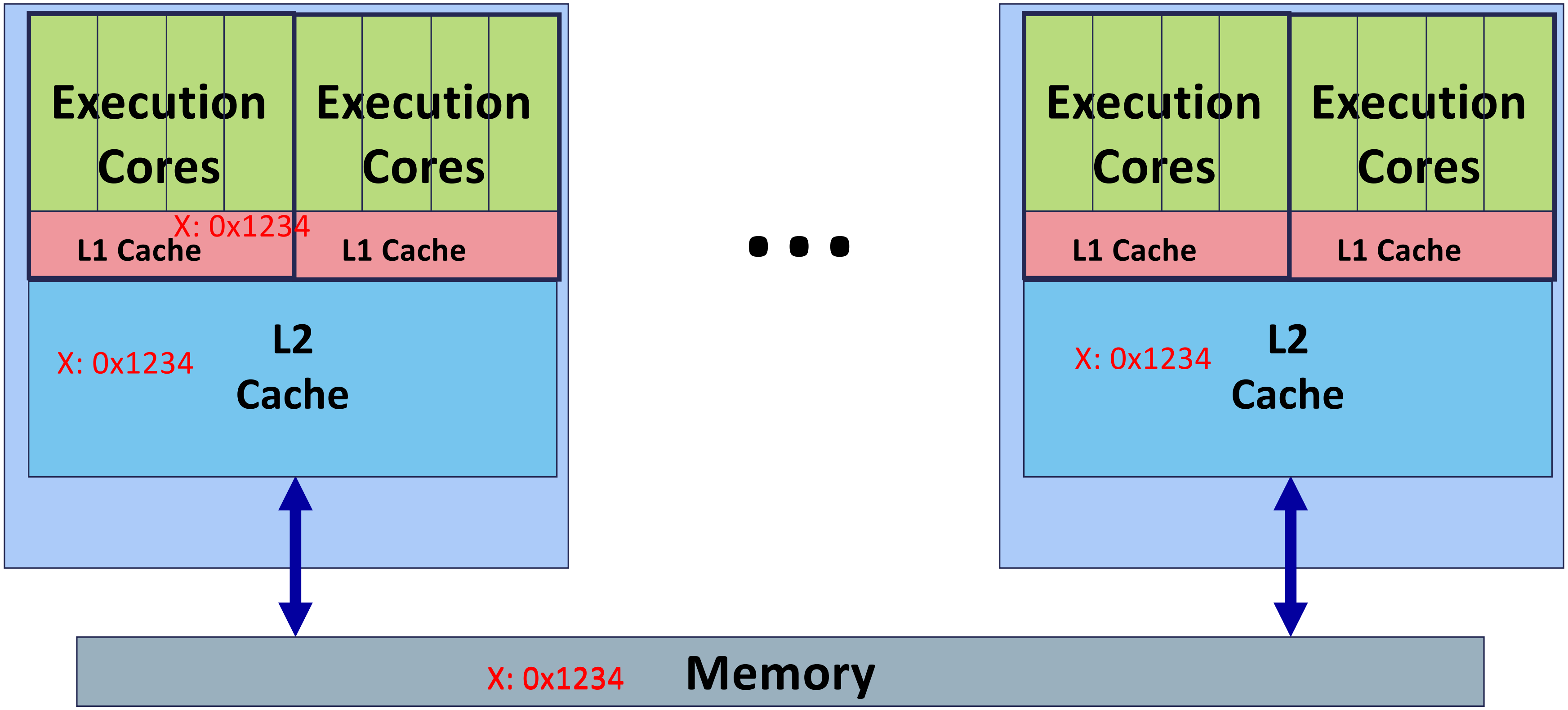
Core 2 reads X



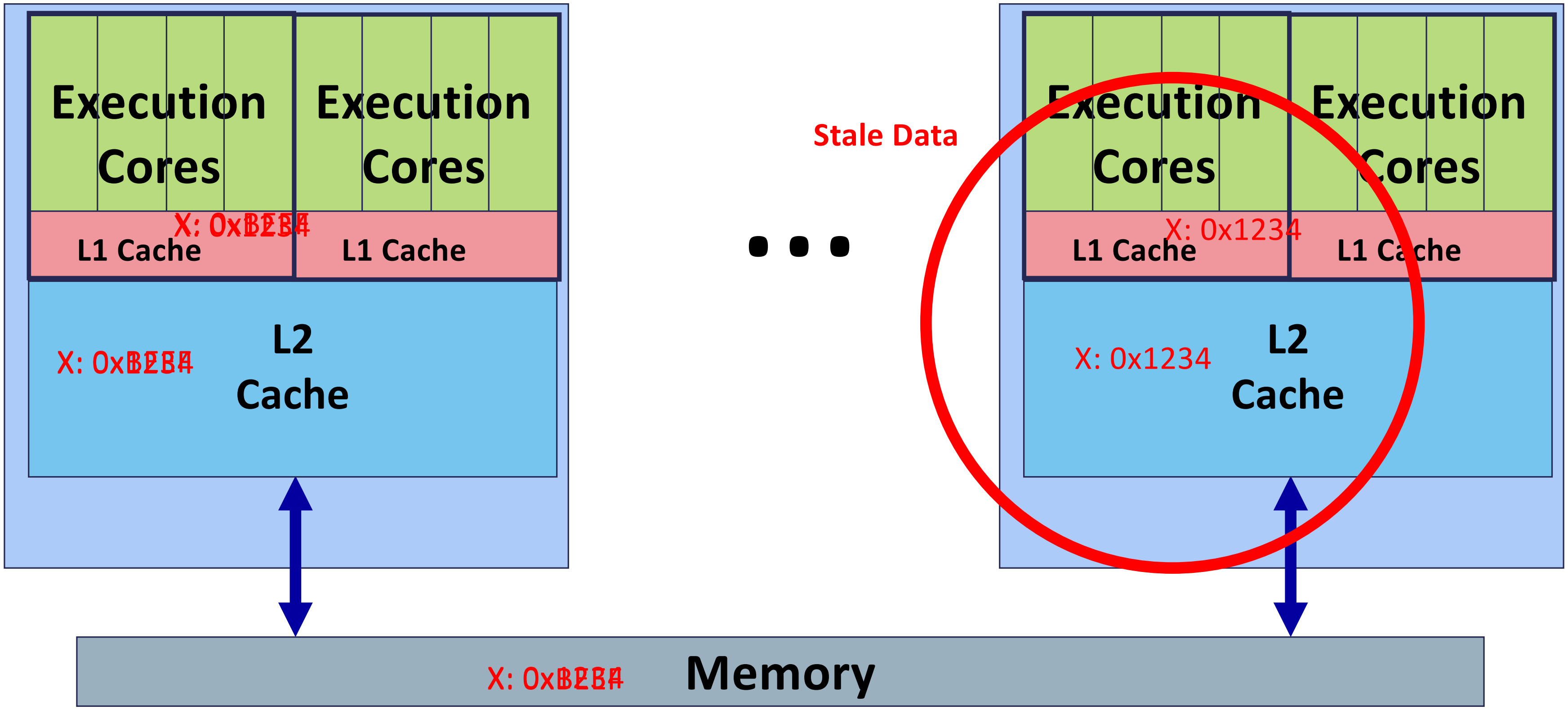
Processor 1/Core 1 reads X



Processor 2/Core 1 reads X



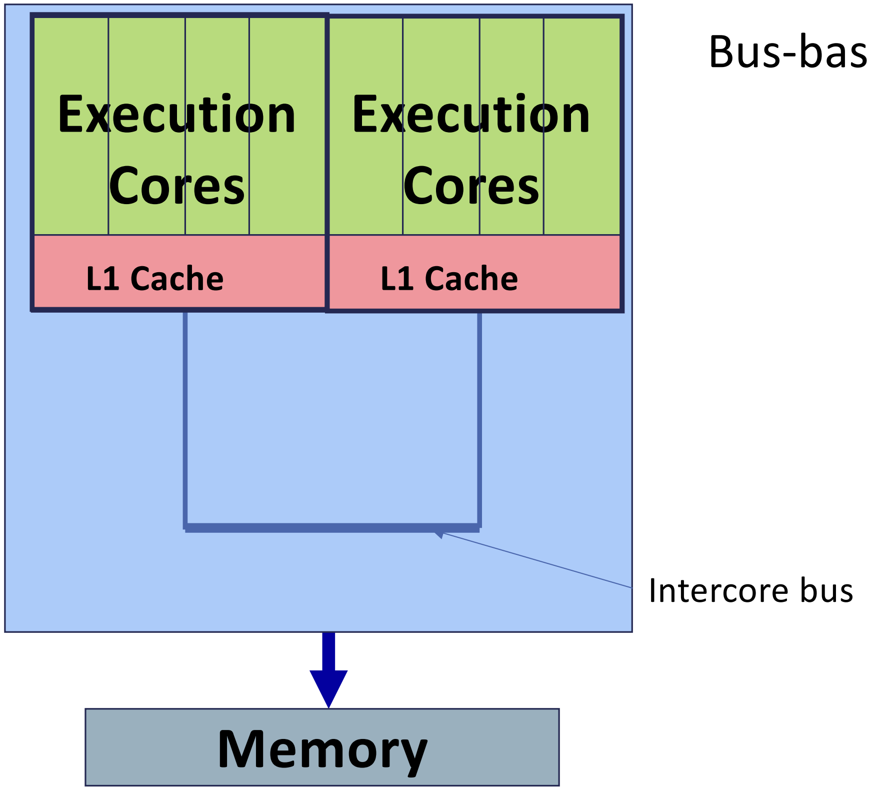
Processor 1/Core 1 writes X



Cache Coherence

- Typically, it is the hardware's job to make sure that programs cannot see such inconsistencies.
- Two parts to a solution:
  - Use hardware to ensure that loads from all cores will return the value of the latest store to that memory location.
  - Use cache metadata to track the state of cached data.
- There are two major approaches to addressing this problem.
  - Snoopy caches
  - Directory based coherence

Let’s Simplify Our Problem



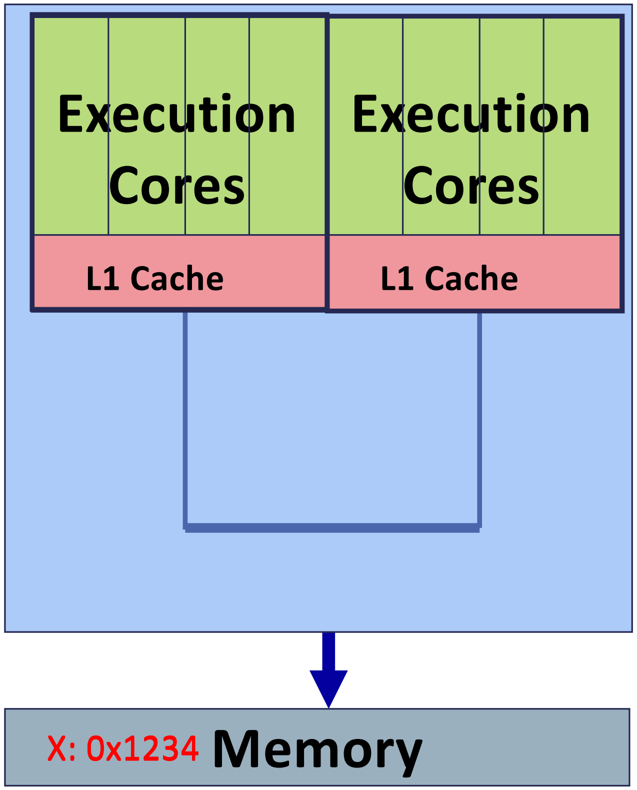
Bus-based “snooping” multicore processor

Invalidation Protocol with Snooping

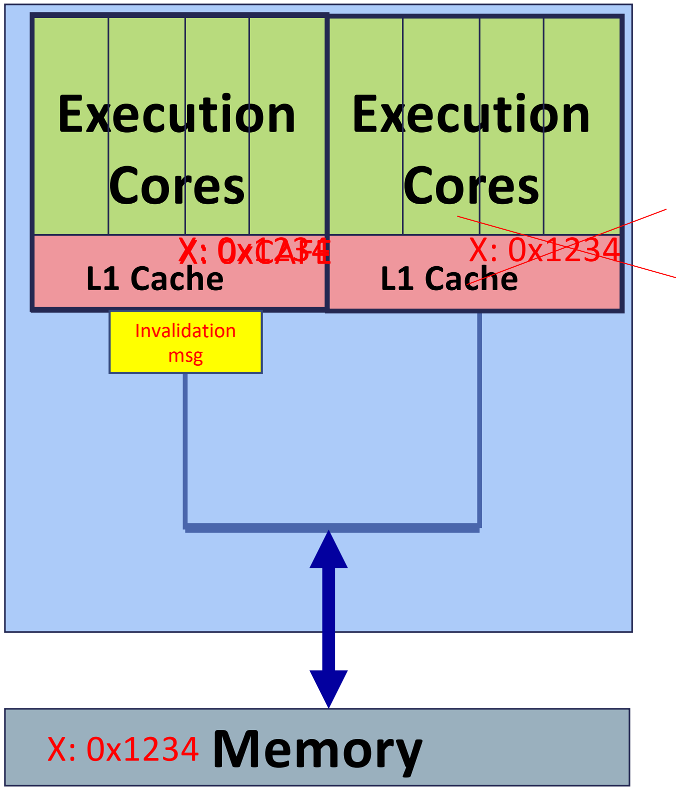
- Invalidation
  - If a core writes to a data item, all other copies of this data item in other caches are invalidated.
- Snooping:
  - All cores continuously snoop (monitor) the bus connecting the cores.

Let’s have Core 1 read X

Core 2 reads X



What Happens this time if Core 1 writes X?

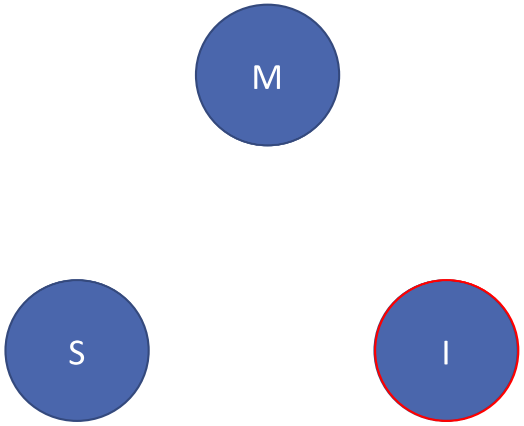


Enhancing Cache Metadata

- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
  - Change state
  - Send invalidate requests
  - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
  - Assumes a writeback cache
  - M: Cache line is modified (i.e., dirty)
  - S: Cache line is shared; appears in multiple caches.
  - I: Cache line is invalid (i.e., contains no valid data)

MSI Protocol (1)

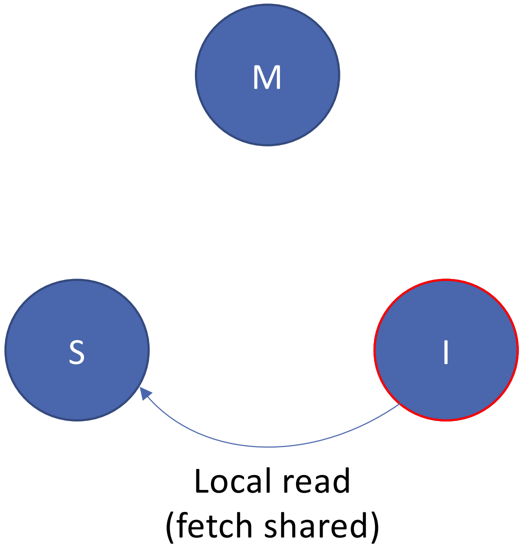
- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
  - Change state
  - Send invalidate requests
  - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
  - Assumes a writeback cache
  - M: Cache line is modified (i.e., dirty)
  - S: Cache line is shared; appears in multiple caches.
  - I: Cache line is invalid (i.e., contains no valid data)



Local Read

MSI Protocol (2)

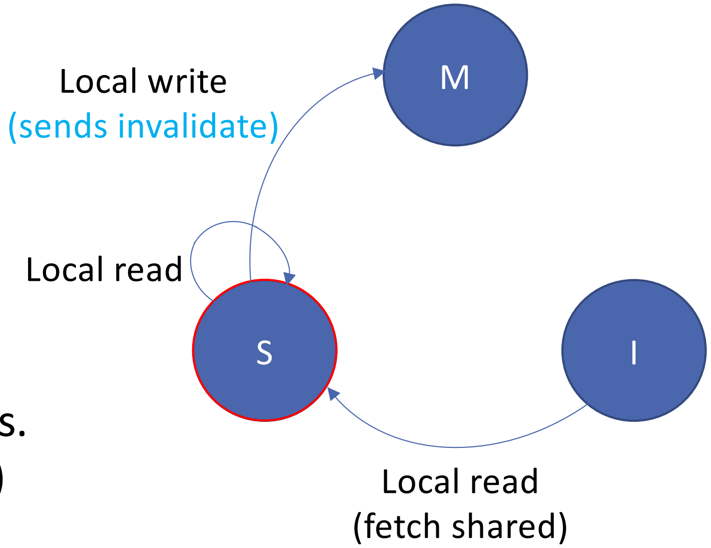
- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
  - Change state
  - Send invalidate requests
  - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
  - Assumes a writeback cache
  - M: Cache line is modified (i.e., dirty)
  - S: Cache line is shared; appears in multiple caches.
  - I: Cache line is invalid (i.e., contains no valid data)



Local Write

MSI Protocol (3)

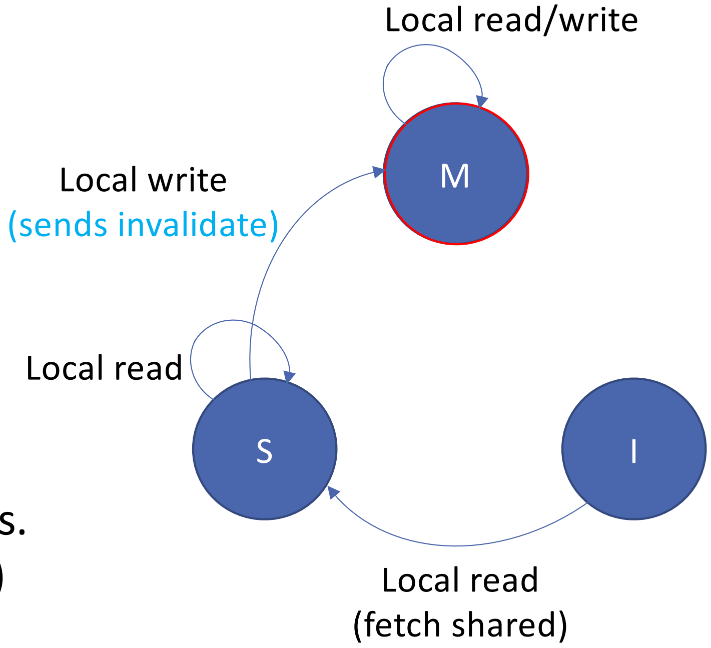
- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
  - Change state
  - Send invalidate requests
  - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
  - Assumes a writeback cache
  - M: Cache line is modified (i.e., dirty)
  - S: Cache line is shared; appears in multiple caches.
  - I: Cache line is invalid (i.e., contains no valid data)



Local Read/Write

MSI Protocol (4)

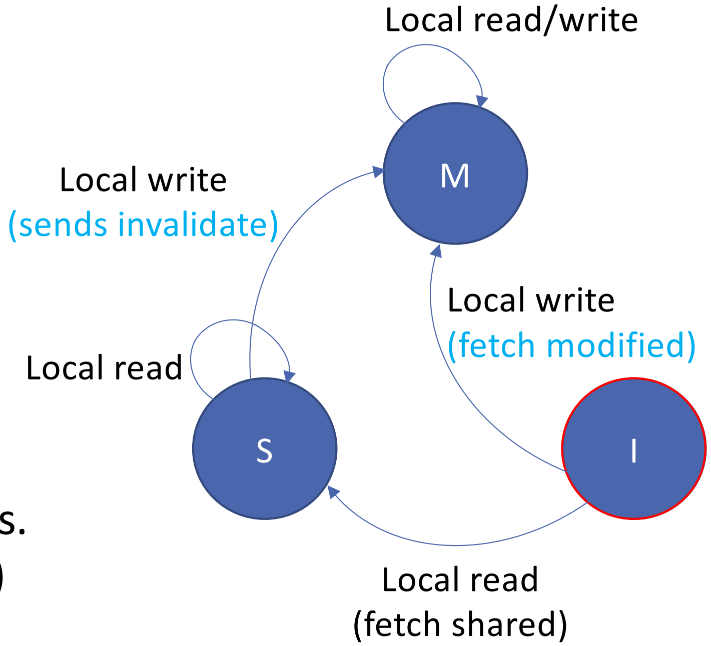
- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
    - Change state
    - Send invalidate requests
    - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
    - Assumes a writeback cache
    - M: Cache line is modified (i.e., dirty)
    - S: Cache line is shared; appears in multiple caches.
    - I: Cache line is invalid (i.e., contains no valid data)



Local Write Miss

MSI Protocol (5)

- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
  - Change state
  - Send invalidate requests
  - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
  - Assumes a writeback cache
  - M: Cache line is modified (i.e., dirty)
  - S: Cache line is shared; appears in multiple caches.
  - I: Cache line is invalid (i.e., contains no valid data)



Local read/write

Local write
(sends invalidate)

Local read

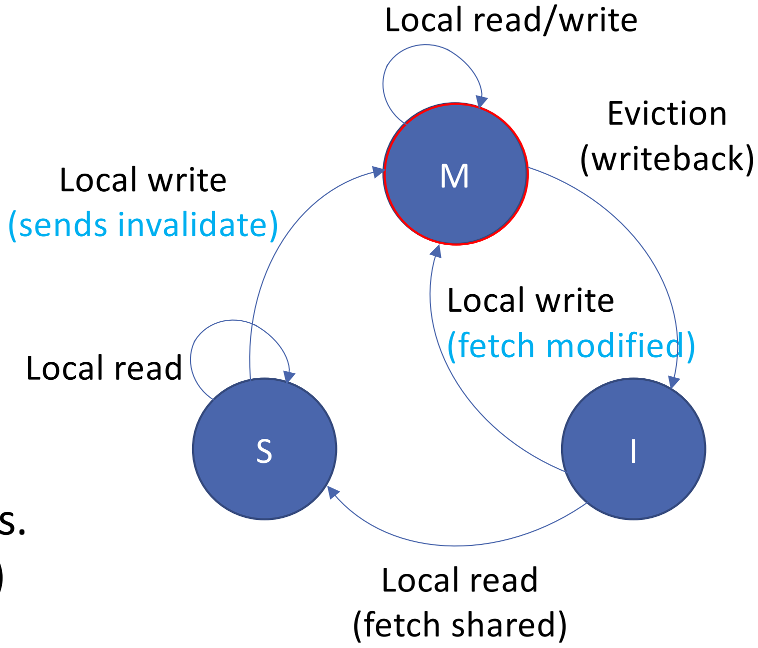
Local write
(fetch modified)

Local read
(fetch shared)

Eviction

MSI Protocol (6)

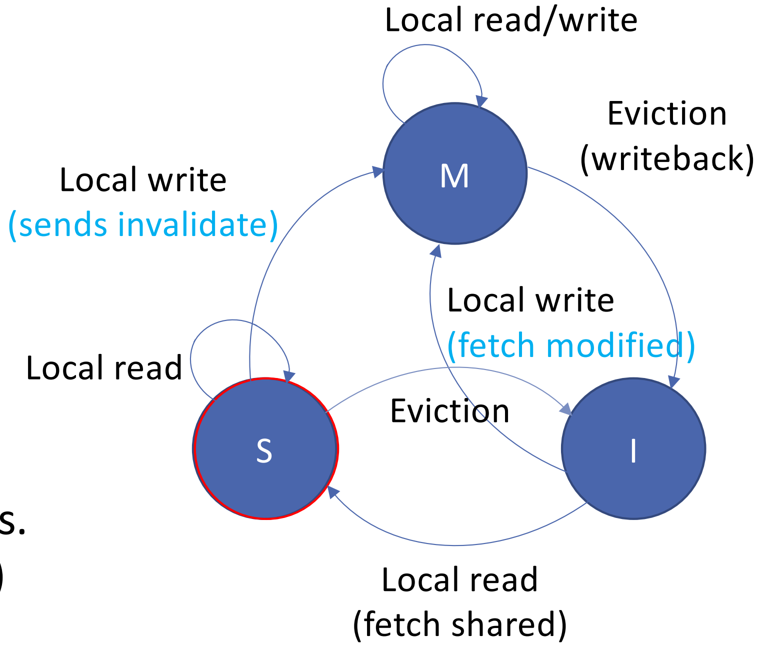
- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
  - Change state
  - Send invalidate requests
  - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
  - Assumes a writeback cache
  - M: Cache line is modified (i.e., dirty)
  - S: Cache line is shared; appears in multiple caches.
  - I: Cache line is invalid (i.e., contains no valid data)



Eviction

MSI Protocol (7)

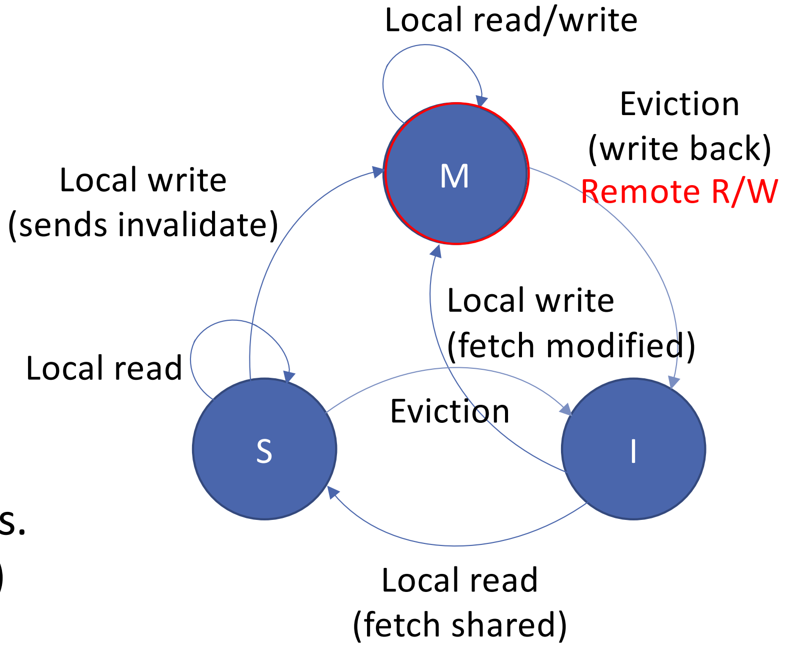
- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
  - Change state
  - Send invalidate requests
  - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
  - Assumes a writeback cache
  - M: Cache line is modified (i.e., dirty)
  - S: Cache line is shared; appears in multiple caches.
  - I: Cache line is invalid (i.e., contains no valid data)



MSI Protocol (8)

Invalidate (from remote)

- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
  - Change state
  - Send invalidate requests
  - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
  - Assumes a writeback cache
  - M: Cache line is modified (i.e., dirty)
  - S: Cache line is shared; appears in multiple caches.
  - I: Cache line is invalid (i.e., contains no valid data)


Local read/write

Eviction
(write back)
Remote R/W

Local write
(sends invalidate)

Local write
(fetch modified)

Local read

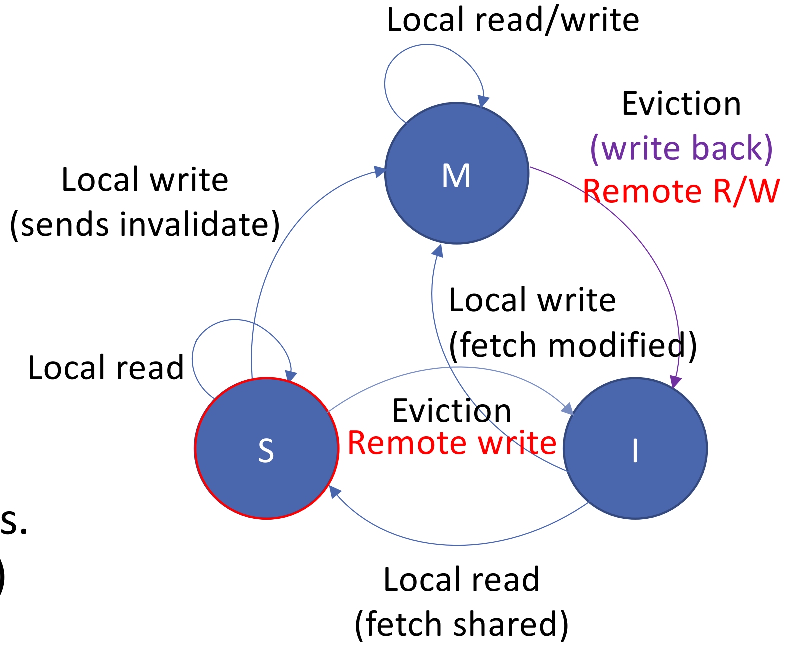
Eviction

Local read
(fetch shared)

Invalidate (from remote)

MSI Protocol (9)

- When we have caches accessible from multiple cores and we have a bus-based coherence protocol, we need to add extend our valid bit to a set of states.
- Cache operations can:
  - Change state
  - Send invalidate requests
  - Request cache lines in a particular state (fetch)
- A minimal set of states (MSI Model)
  - Assumes a writeback cache
  - M: Cache line is modified (i.e., dirty)
  - S: Cache line is shared; appears in multiple caches.
  - I: Cache line is invalid (i.e., contains no valid data)


Local read/write
M
Eviction
(write back)
Remote R/W
Local write
(sends invalidate)
Local write
(fetch modified)
Local read
S
Eviction
Remote write
I
Local read
(fetch shared)