CPSC313: Computer Hardware and Operating Systems

Unit 3: Caching

Multiple cores and processors: MESI
(Communicating… as little as we can!)

Thanks to J.P. Shen of CMU for inspiration for the slides on snoopy caches.

Administration

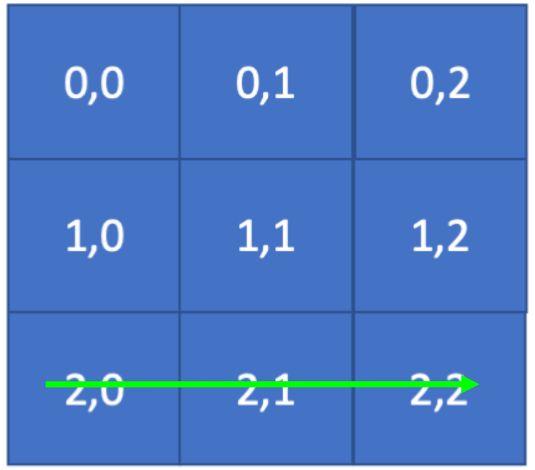
- Quiz 3:
  - You should already have made a reservation!
  - Get to know the provided material with the Quiz 3 Introduction/Information
  - Practice with the Quiz 3 Practice and our in-/pre-class exercises, labs, tutorials, lectures, and the textbook.
  - Class Friday cancelled (as usual for a quiz), but come if you have questions!
- Lab 7 is due on Sunday and is fairly challenging (hopefully less-so than Lab 5).

Today

- Intuition behind matrix multiply exercise
- Learning Outcomes for Today
  - Motivate *why* we want to use MSI, MESI, or other “cache coherency” approaches
  - Identify shortcomings in the MSI protocol
  - Derive the state transitions for the MESI protocol

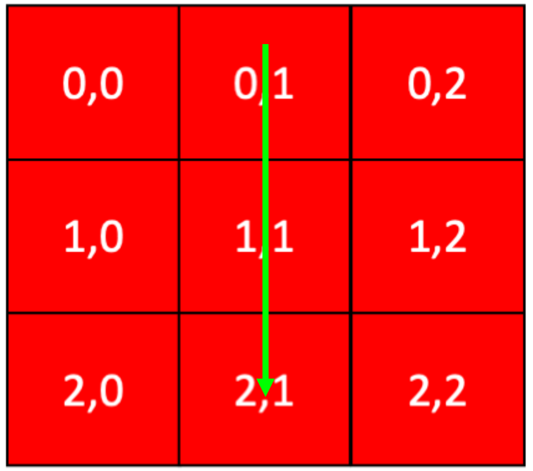
Matrix Multiply from Last Class

IJK: MELT(c, sz, i, j)+=MELT(a, sz, i, k) * MELT(b, sz, k, j)



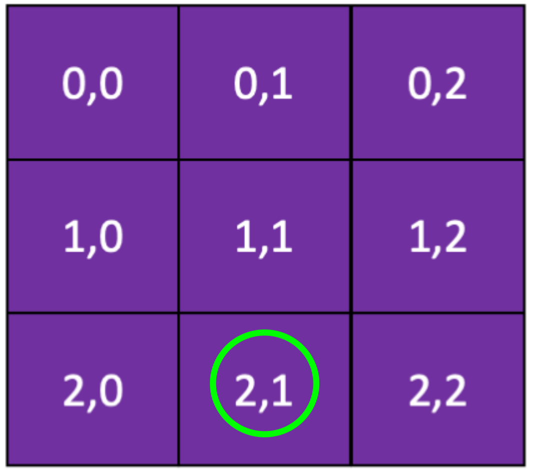
A

A(i,0), A(i,1), A(i,2)...



B

B(0,j), B(1,j), B(2,j)...

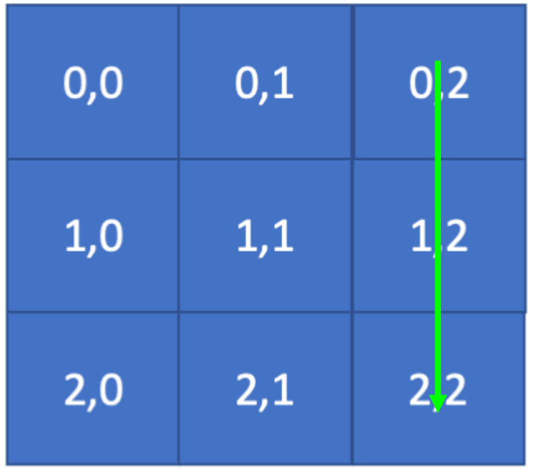


C

C(i,j), C(i,j), C(i,j)...

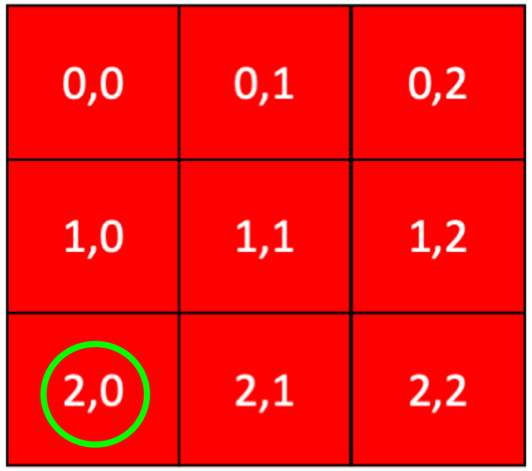
Matrix Multiply from Last Class

KJI MELT(c, sz, i, j)+=MELT(a, sz, i, k) * MELT(b, sz, k, j)



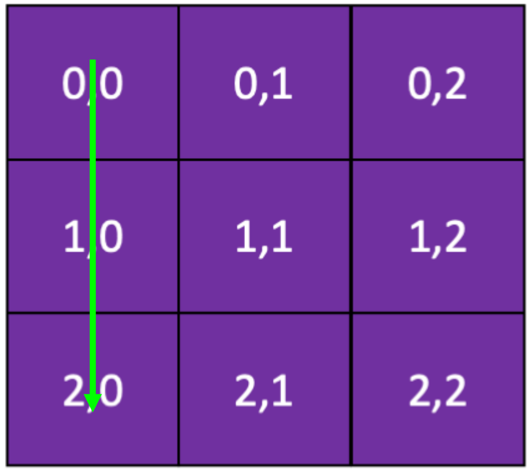
A

A(0,k), A(1,k), A(2,k)...



B

B(k,j), B(k,j), B(k,j)...

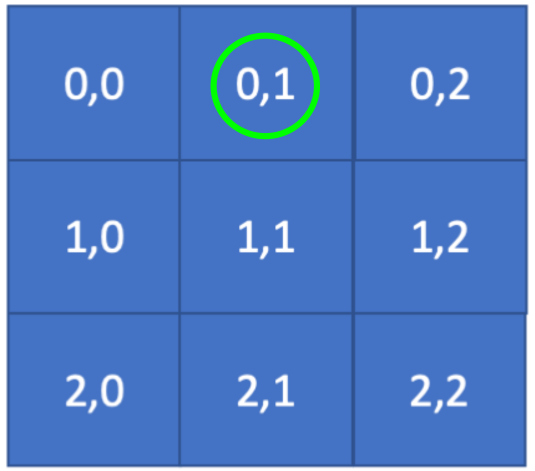


C

C(0,j), C(1,j), C(2,j)...

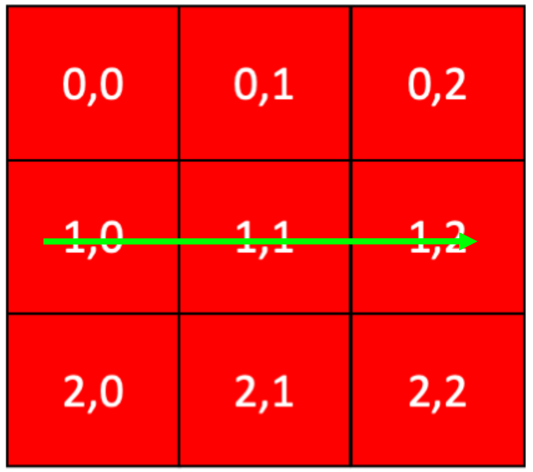
Matrix Multiply from Last Class

KIJ MELT(c, sz, i, j)+=MELT(a, sz, i, k) * MELT(b, sz, k, j)



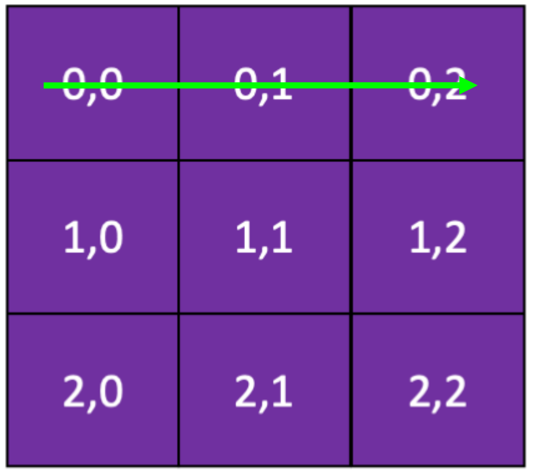
A

A(i,k), A(i,k), A(i,k)...



B

B(k,0), B(k,1), B(k,2)...



C

C(i,0), C(i,1), C(i,2)...

On to Cache Coherence: What is costly in parallelism?

Communication!!!

For Each Action: Is communication needed? (with the data source or other caches)

Scenario #1:

- A single-core computer
- A single, write-back, write-allocate, L1 cache

Actions:

- Read hit: **NO**
- Read miss: **YES**
- Write hit: **NO**
- Write miss: **YES**
- Evict a *clean* line: **NO**
- Evict a *dirty* line: **YES**

We want to **minimize** communication.
So, we must know when communication happens!
We can't do better than our single-core case. It is our "goal".

For Each Action: Is communication needed? (with the data source or other caches)

Scenario #2:

- A multi-core computer
- Each has a write-back, write-allocate L1 cache
- All share a single L2 cache
- We maintain coherency based on the valid/dirty bits

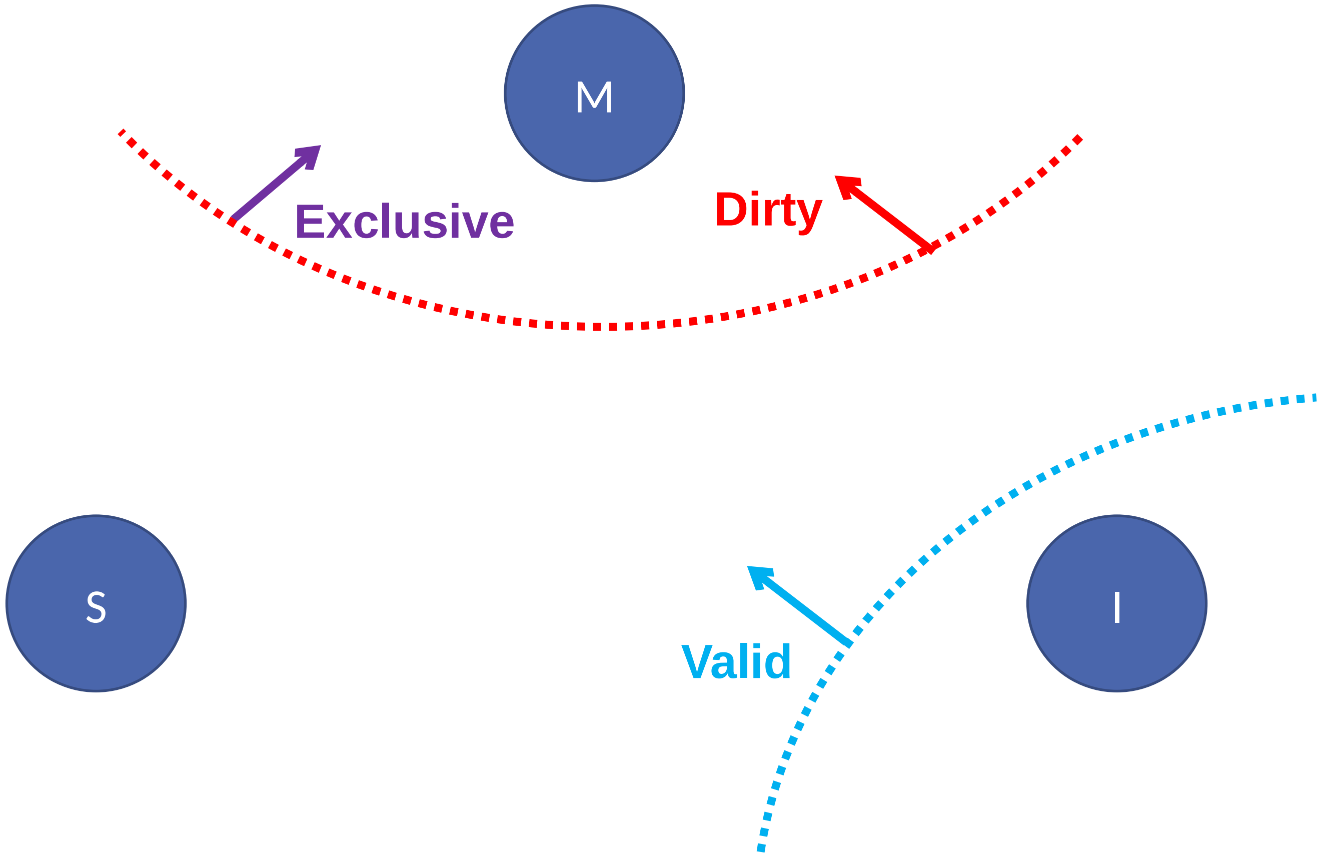
Actions:

- Read hit: **NO**
- Read miss: **YES**
- Write hit: **YES**
- Write miss: **YES**
- Evict a *clean* line: **NO**
- Evict a *dirty* line: **YES**

Remember: all the cores should agree on the value of any given cache line.

MSI Adds “Exclusivity” to “Valid, Dirty”

M: means “modified” but also means “exclusively **mine!**”



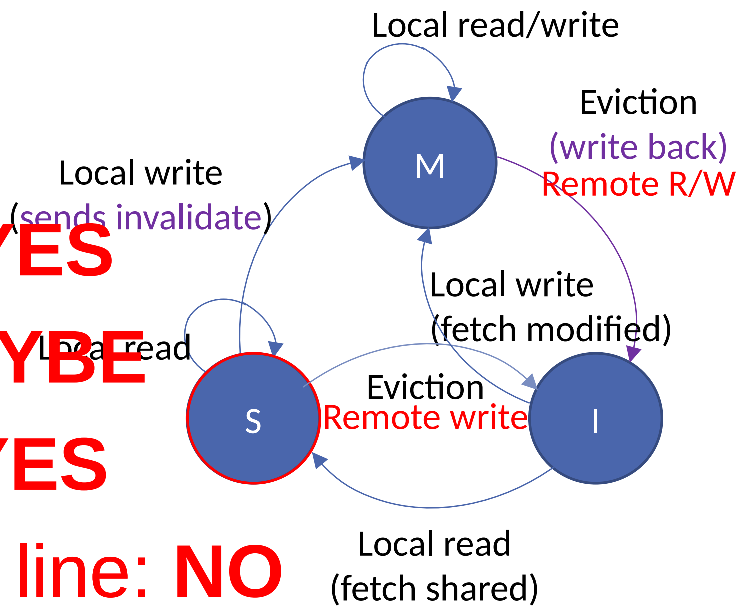
For Each Action: Is communication needed?
(with the data source or other caches)

Scenario #3:

- A multi-core computer
- Each has a write-back, write-allocate L1 cache
- All share a single L2 cache
- L1 caches synchronize with MSI

Actions:

- Read hit: **NO**
- Read miss: **YES**
- Write hit: **MAYBE**
- Write miss: **YES**
- Evict a *clean* line: **NO**
- Evict a *dirty* line: **YES**



Reminder: state of **M**, **S**, or **I** replaces both the valid and dirty bits.
M/modified: dirty, solely "owned" by this cache
S/shared: clean, may be in multiple caches
I/invalid: not in this cache

The big picture...

- Each cache line is in some state for each cache (though most are “I”!)
- The metadata for each slot in the cache now contains a state.
  - One of M, S, I (requires 2 bits)
  - These provide MORE information than just “valid”, “dirty”

The big picture (continued)…

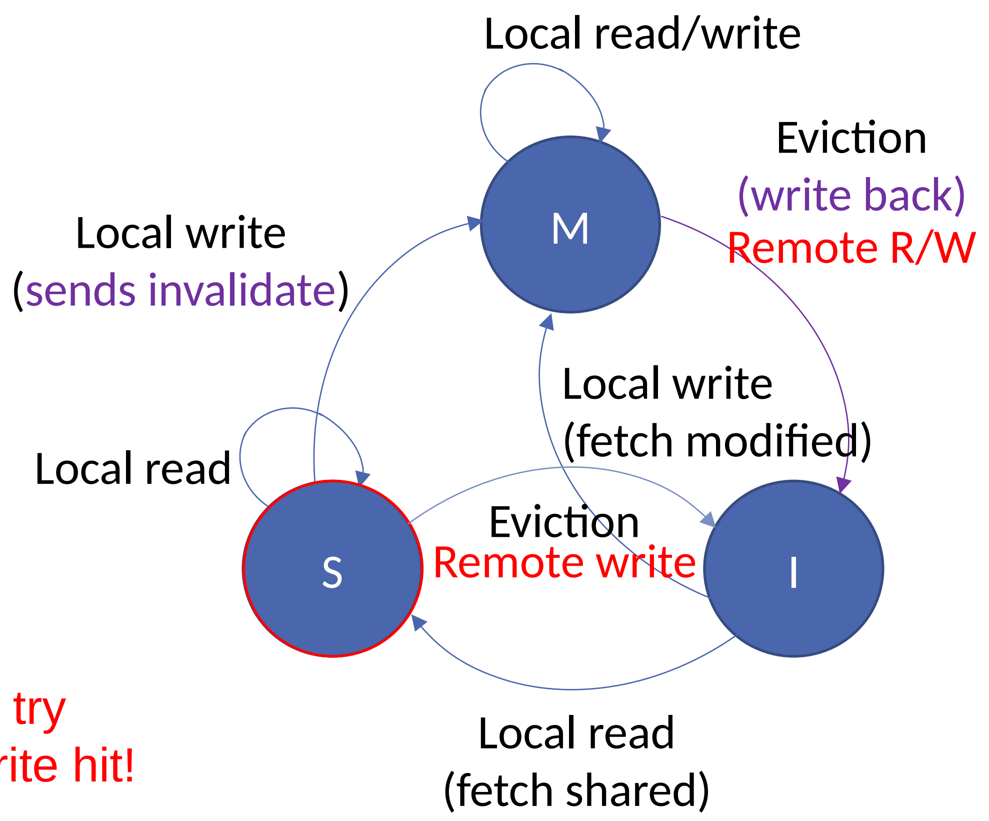
- A particular state is specific to a particular line in a particular cache
  - The state of the cache line at 0xffff00 is unrelated to that of 0xc0aa00!
- Each message sent includes an address’ index and tag bits and possibly more.
- A cache operation can fetch data, set a cache line’s state, evict a line, and send an “invalidate” request to make other caches evict lines

But wait …

- In the **common case**, I am the only one who has a particular line cached.
  - *Algorithms that behave otherwise **share a lot of memory** and so **communicate a lot** and thus do a **bad job of parallelism***
- But sometimes on a write hit, I have to communicate?! Let's try it:

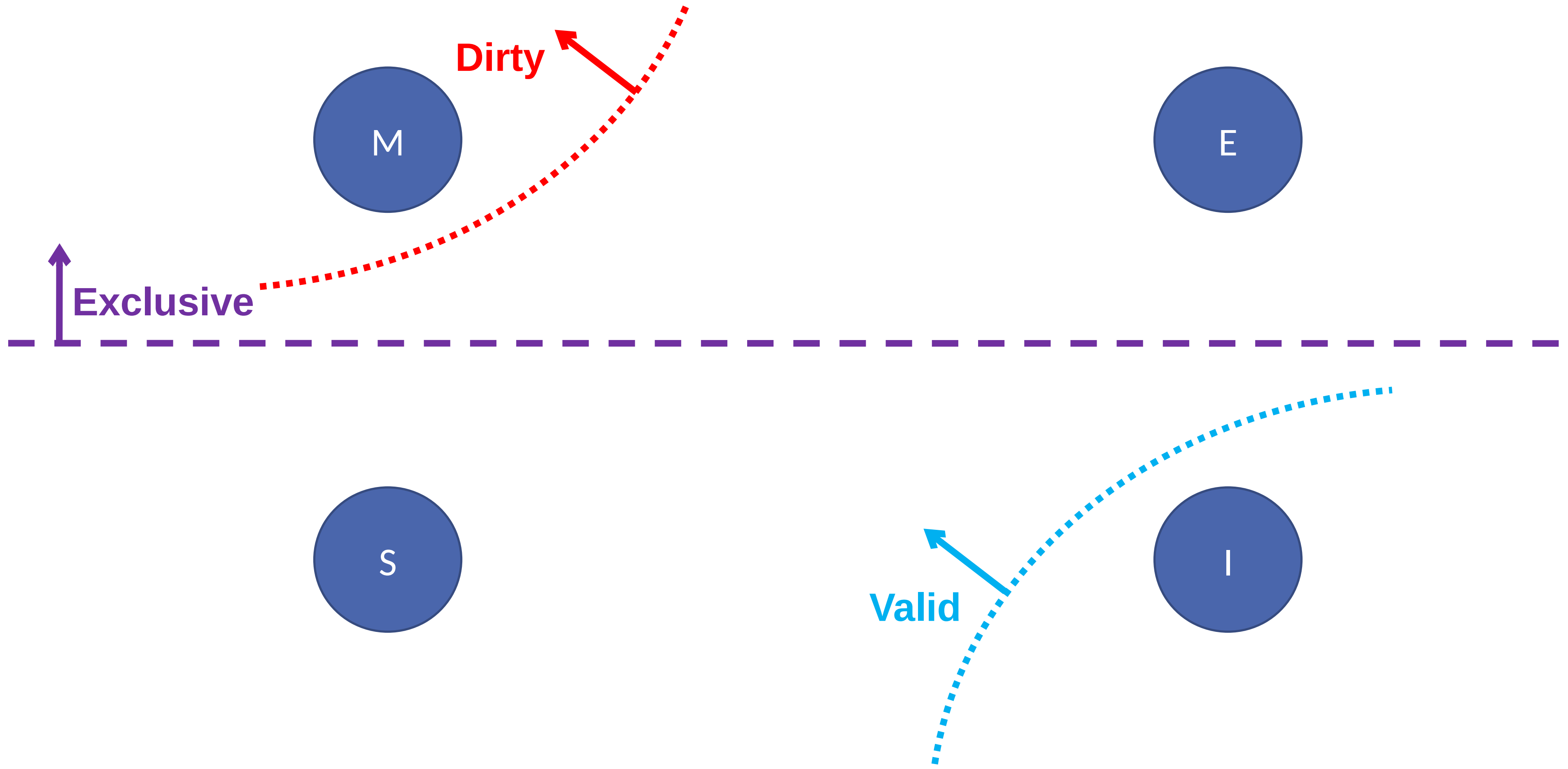
Read 0x8000	Read miss: communicates; OK!
Write 0x8000	Write hit: communicates; NO?!

Let's add another state "E" for "exclusive, clean" and try to use it to avoid communication on that particular write hit!



Let’s add another state! MESI (0)

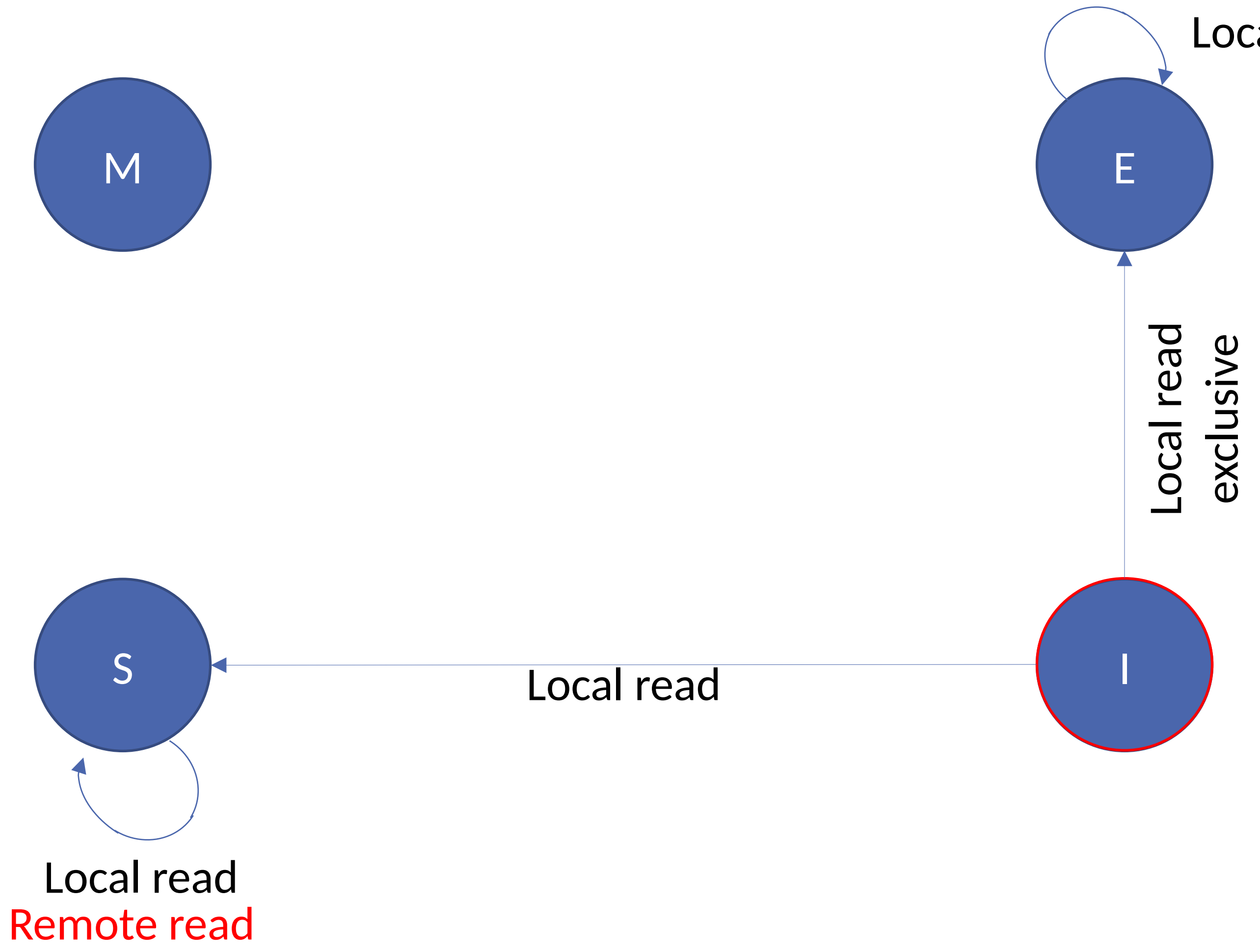
- E: Exclusive: The data does not live in any other cache



Legend:
BLACK: local actions
RED: remote actions
BLUE: messages I send

MESI (1)

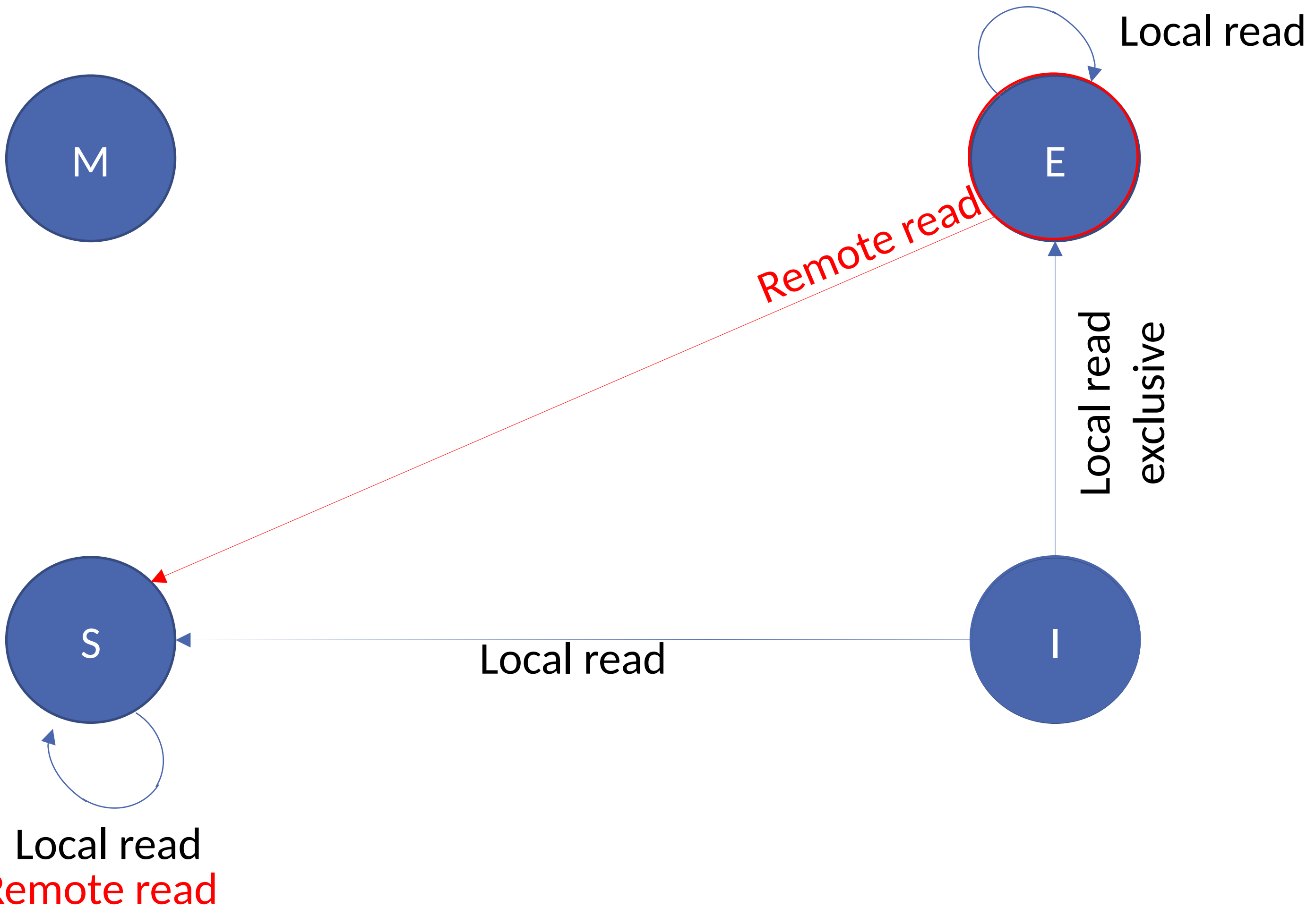
- E: Exclusive: The data does not live in any other cache



Legend:
BLACK: local actions
RED: remote actions
BLUE: messages I send

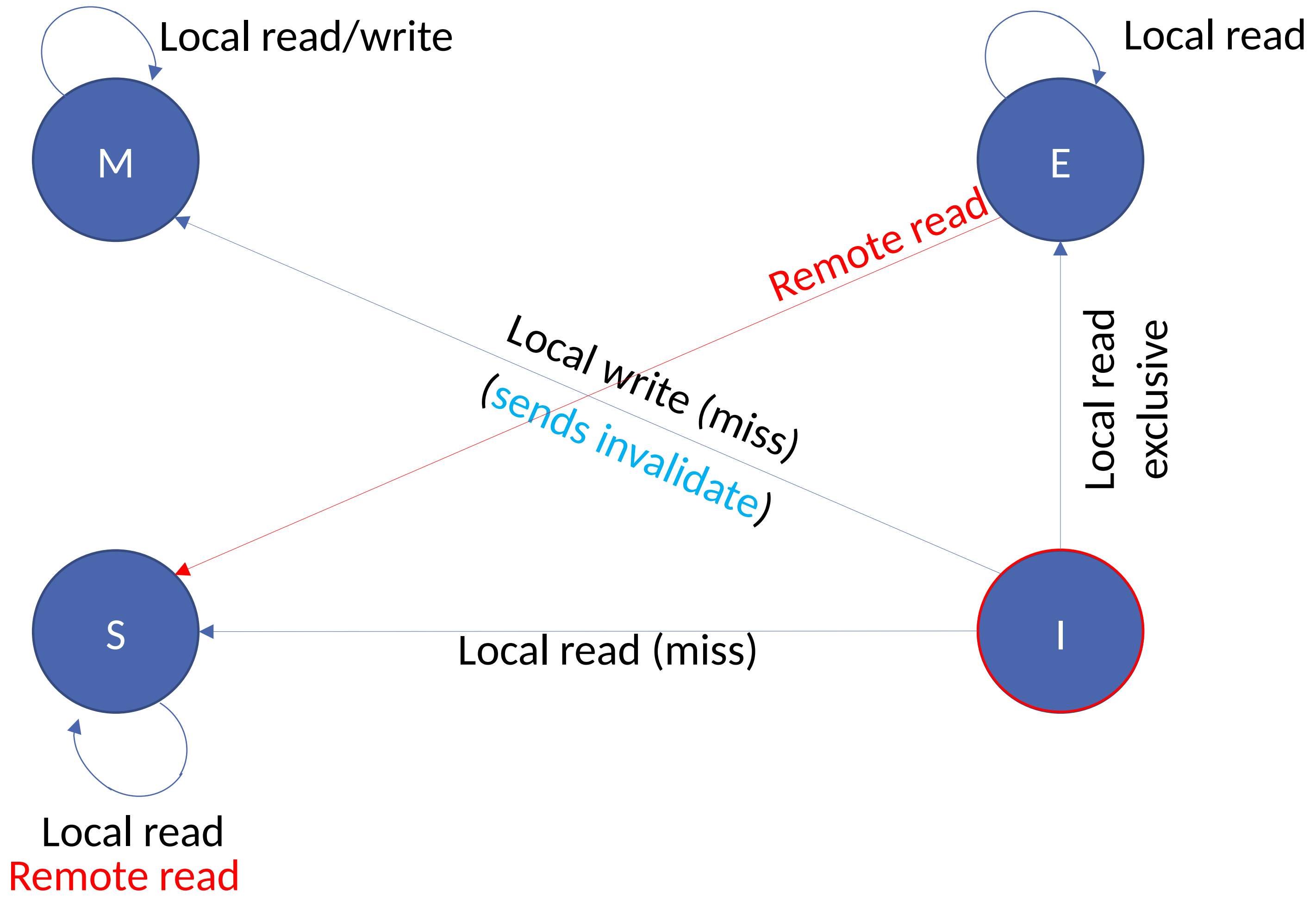
MESI (2)

- E: Exclusive: The data does not live in any other cache



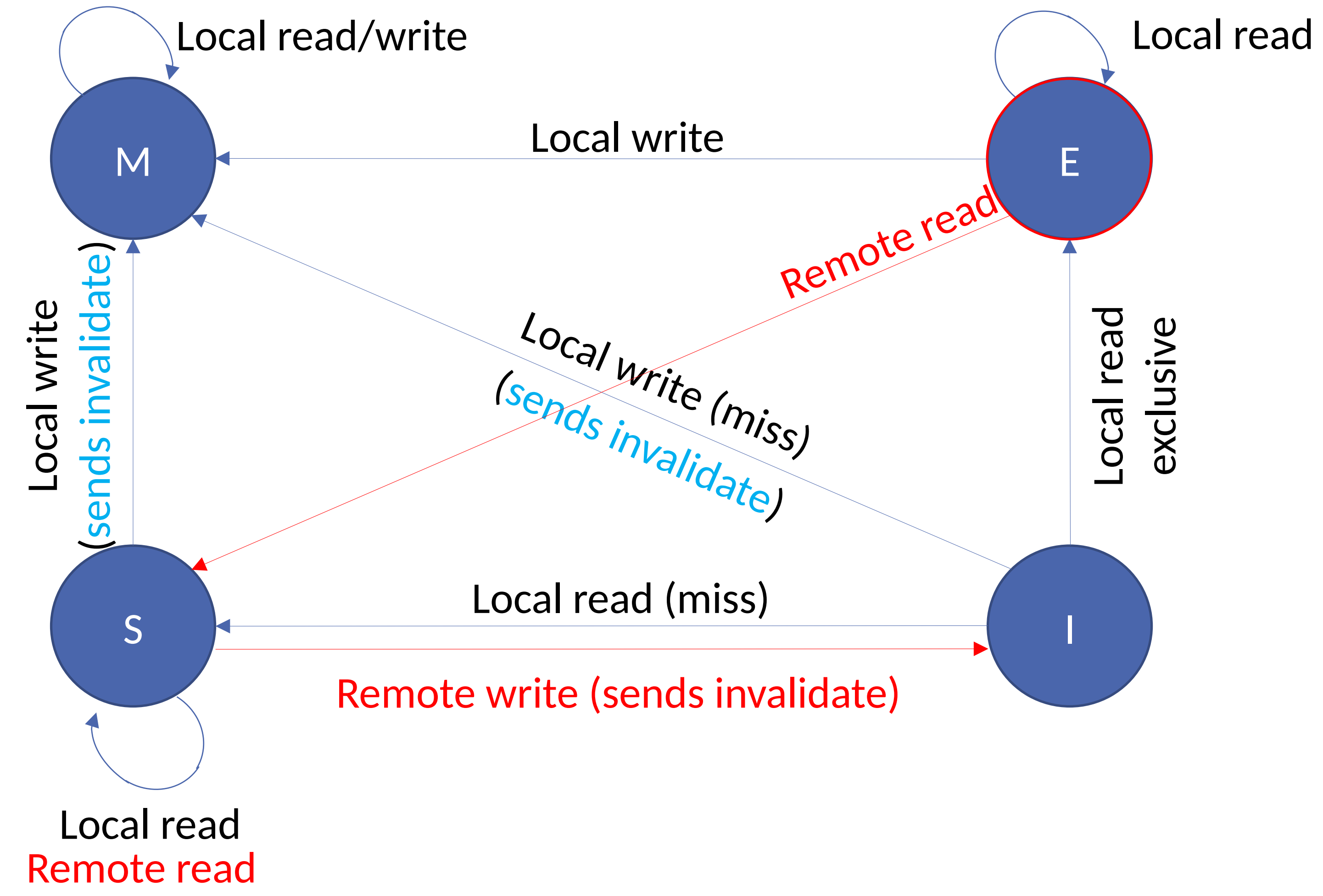
MESI (3)

Legend:
BLACK: local actions
RED: remote actions
BLUE: messages I send



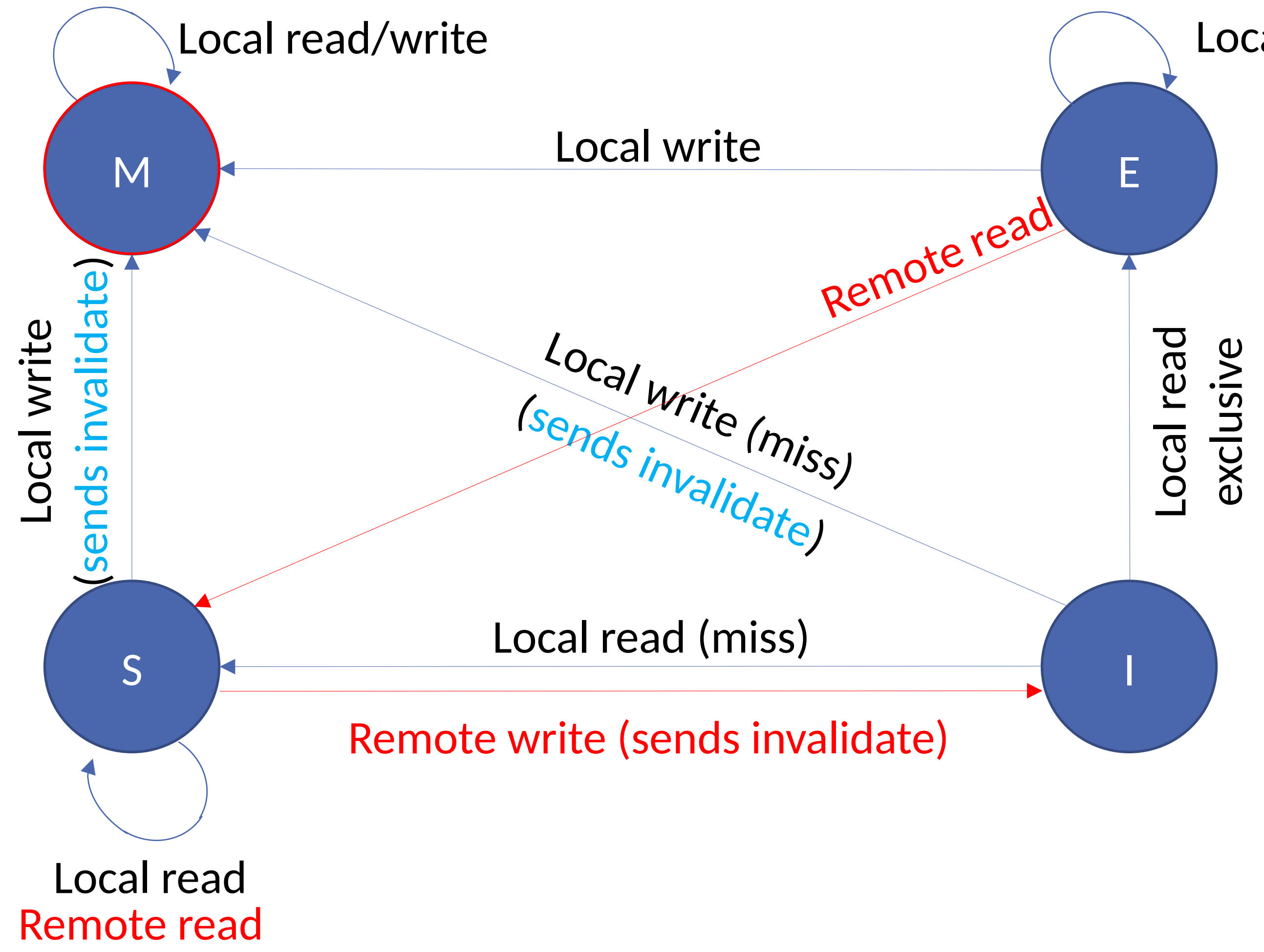
Legend:
BLACK: local actions
RED: remote actions
BLUE: messages I send

MESI (4)



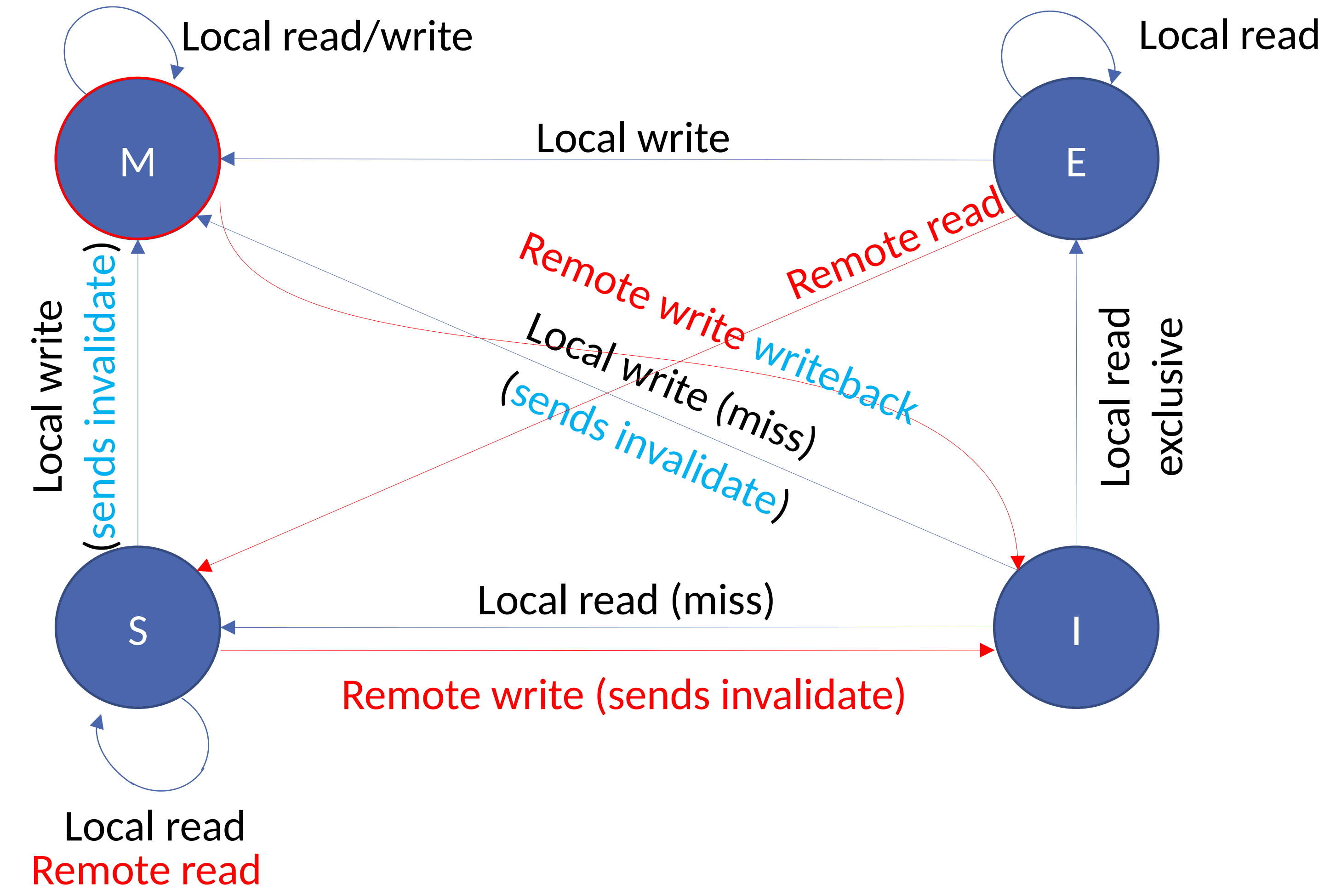
MESI (5)

Legend:
BLACK: local actions
RED: remote actions
BLUE: messages I send



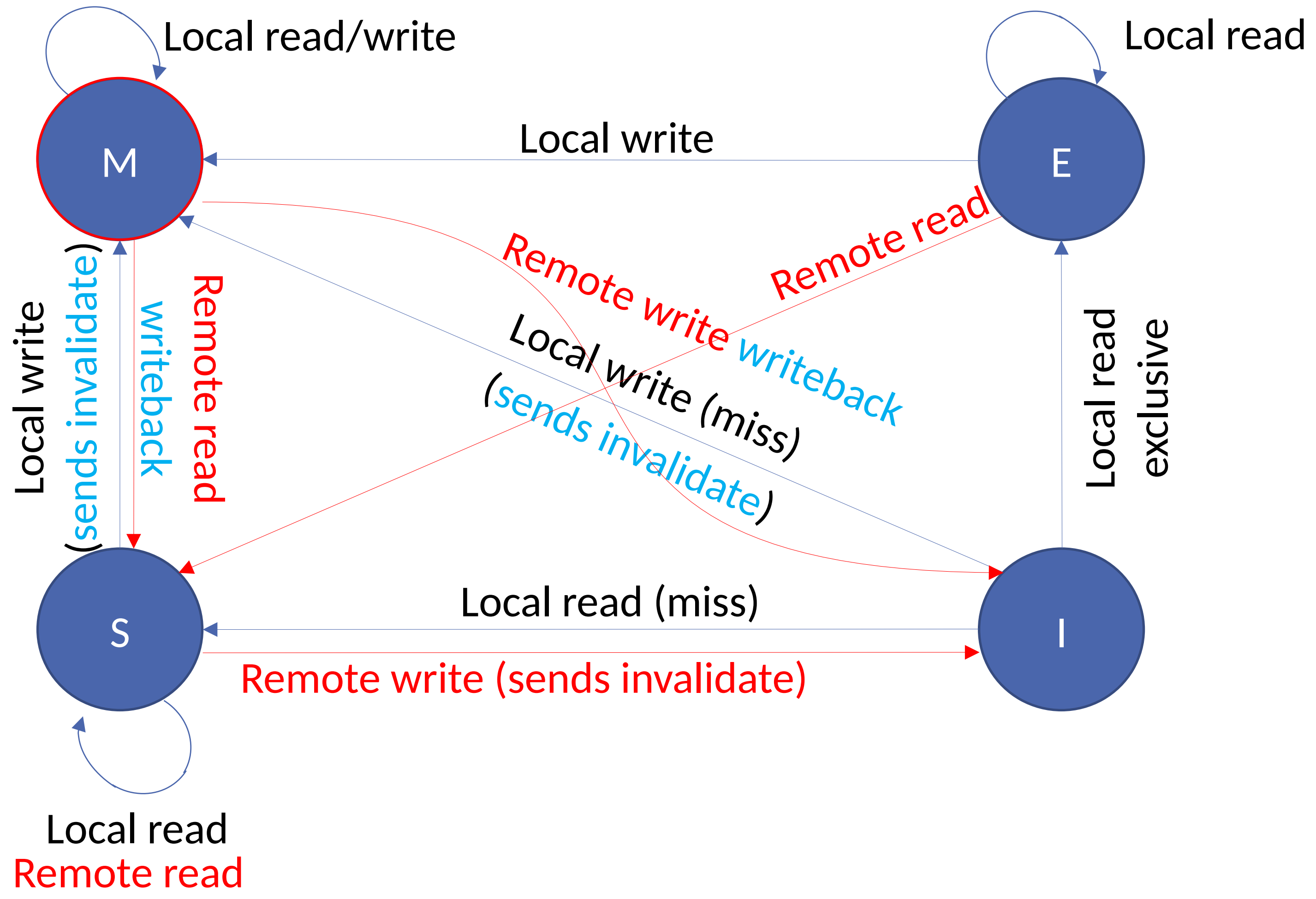
MESI (6)

Legend:
BLACK: local actions
RED: remote actions
BLUE: messages I send



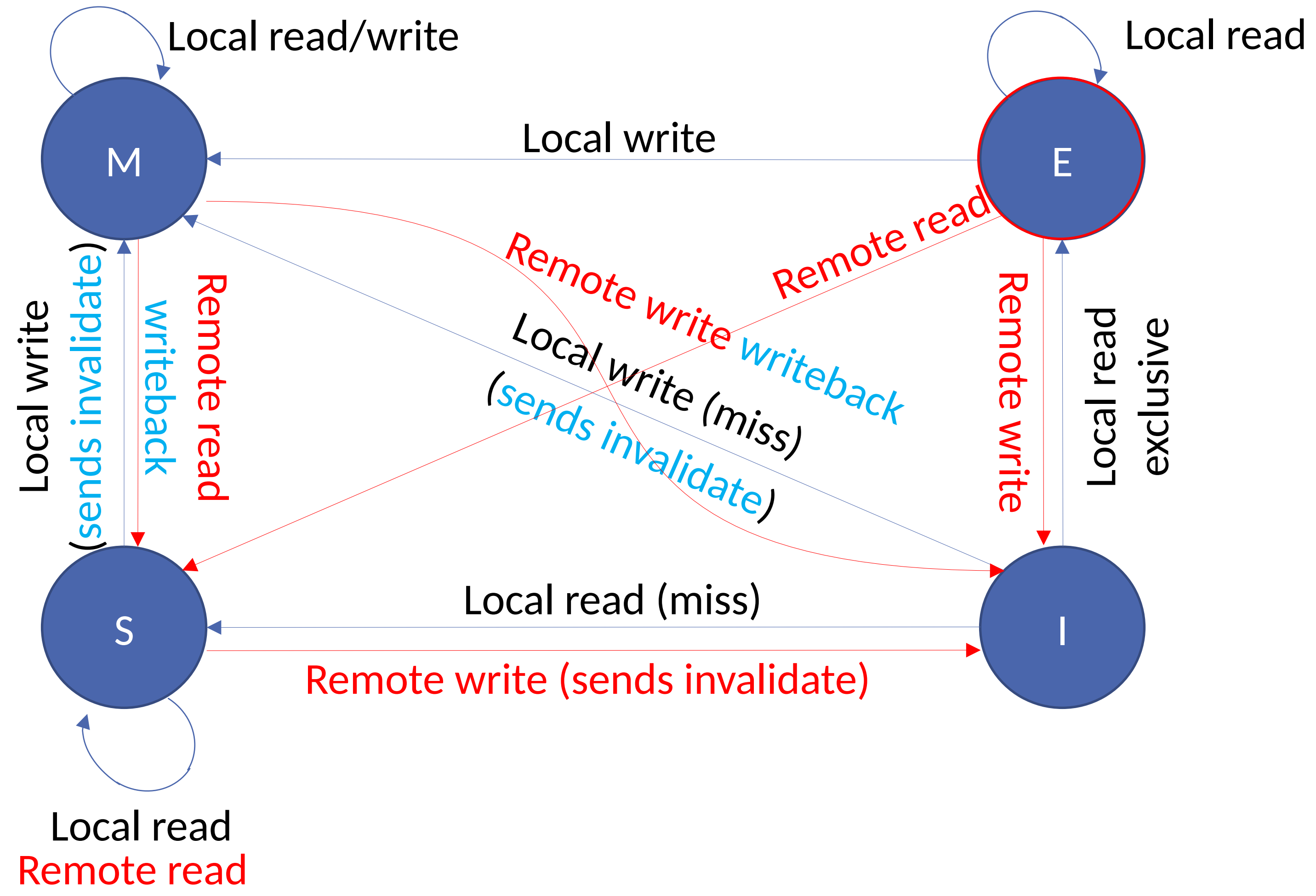
MESI (7)

Legend:
BLACK: local actions
RED: remote actions
BLUE: messages I send



Legend:
BLACK: local actions
RED: remote actions
BLUE: messages I send

MESI (8)



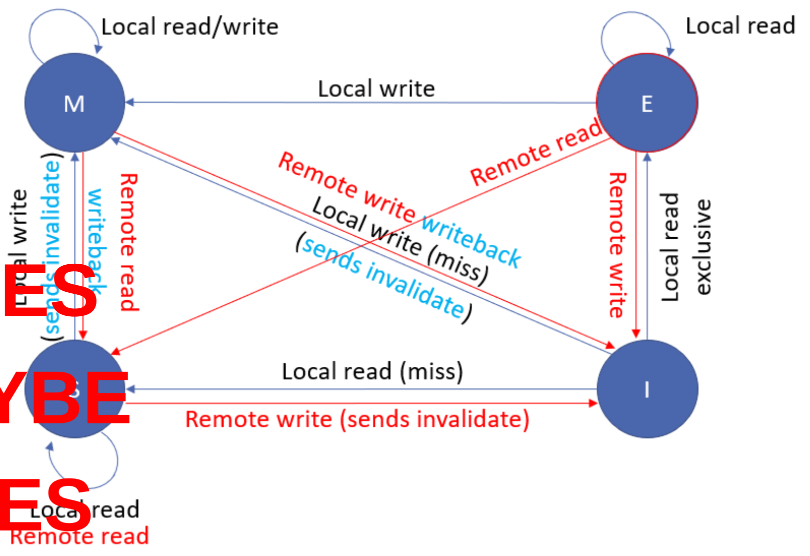
On what actions does communication happen?

Scenario #4:

- A multi-core computer
- Each has a write-back L1 cache
- All share a single L2 cache
- L1 caches synchronize with MSI

Actions:

- Read hit: **NO**
- Read miss: **YES**
- Write hit: **MAYBE**
- Write miss: **YES**
- Evict a *clean* line: **NO**
- Evict a *dirty* line: **YES**



Reminder: state of **M**, **E**, **S**, or **I** replaces both the valid and dirty bits.
M/modified: dirty, solely “owned” by this cache
E/exclusive: clean, only in this cache
S/shared: clean, may be in multiple caches
I/invalid: not in this cache

The Alternative: Directory Based Coherence

- A (centralized) directory stores all the state information
- Processors consult the directory before updating caches
  - Information in the directory is similar to the cache states
    - Shared: Lives in more than one cache
    - Uncached: Lives in 0 caches
    - Exclusive: Lives in exactly 1 cache
  - Must also track which processor(s) have the data
- Rather than broadcasting on the bus, a core consults the directory and sends messages only to affected cores.

End of Caching

- Next Up: File Systems
  - What happens when things no longer fit in memory?
  - What happens when you want data to persist across a reboot?