CS6354: Snooping Cache Coherency

7 October 2016

To read more...

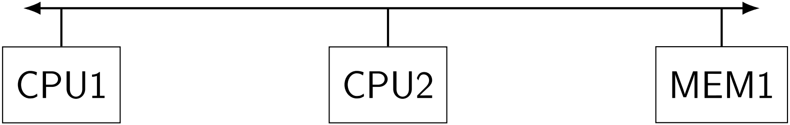
This day's papers:

Goodman, "Using cache memory to reduce processor-memory traffic"
Archibald and Baer, "Cache Coherence Models: Evaluation Using a Multiprocessor Simulation Model"

Supplementary readings:

Hennessy and Patterson, section 5.3

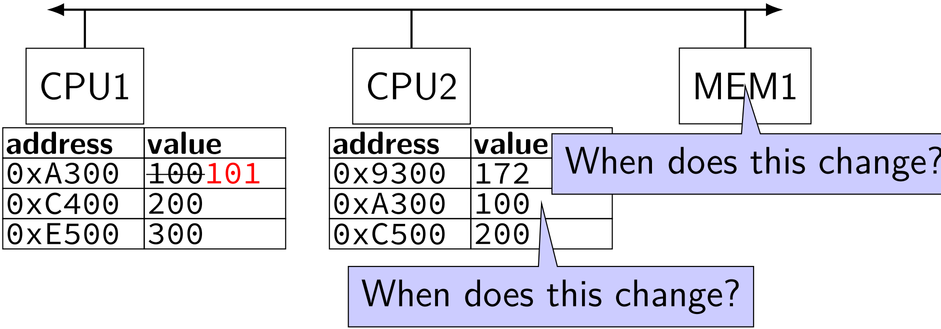
caching shared memories



address	value
0xA300	100
0xC400	200
0xE500	300

address	value
0x9300	172
0xA300	100
0xC500	200

caching shared memories



address	value
0xA300	~~100~~101
0xC400	200
0xE500	300

address	value
0x9300	172
0xA300	100
0xC500	200

When does this change?

When does this change?

CPU1 writes 101 to 0xA300?

cache coherency states

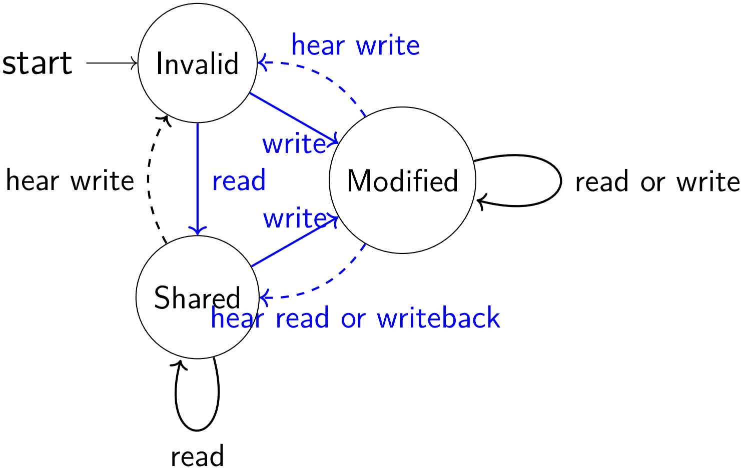
extra information for each cache block

- overlaps with valid, dirty bits

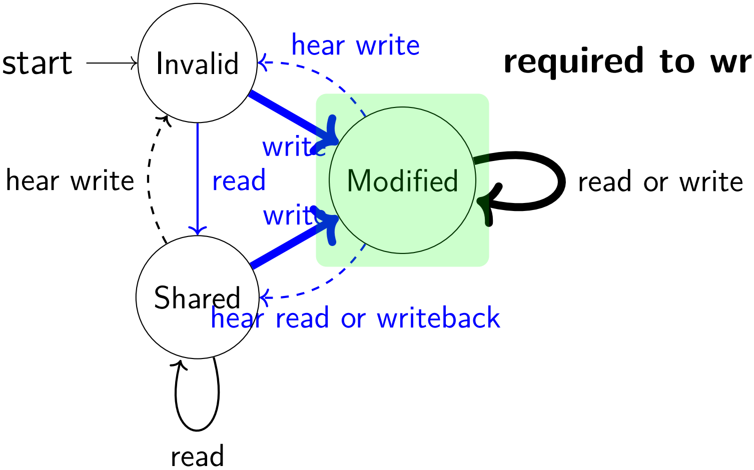
stored in each cache

different caches may have different states for same block

scheme 1: MSI

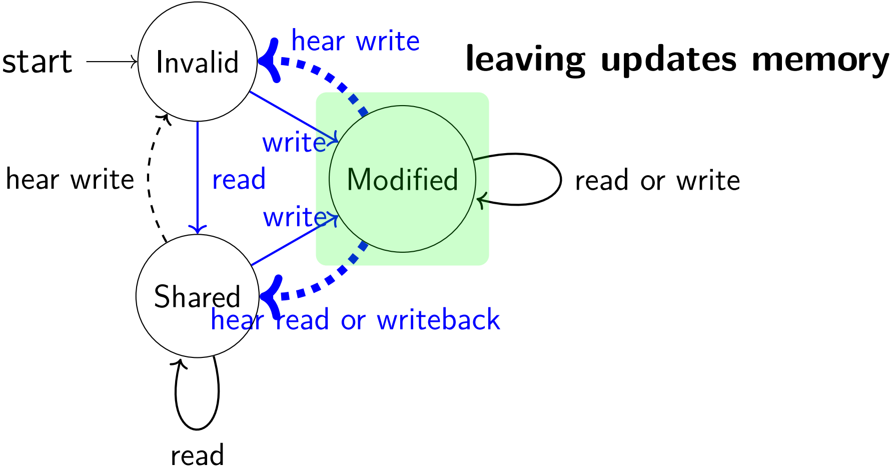

start
Invalid
hear write
write
Modified
read or write
hear write
read
write
Shared
hear read or writeback
read

scheme 1: MSI



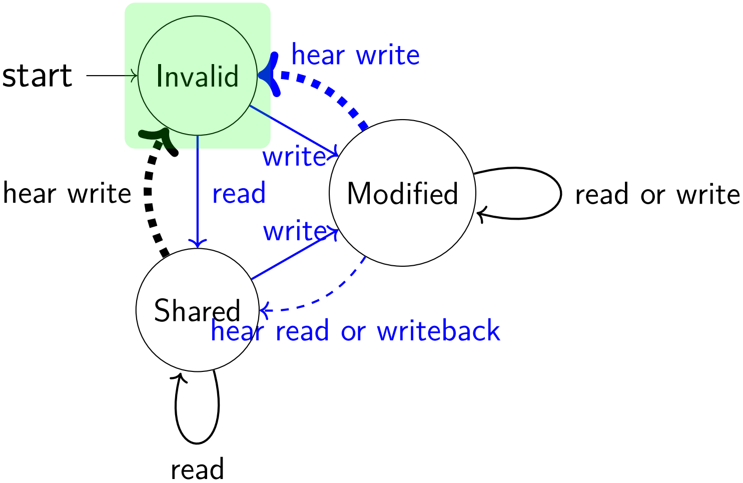
required to write

scheme 1: MSI


hear write
leaving updates memory
start
Invalid
write
hear write
read
read or write
Modified
write
Shared
hear read or writeback
read

scheme 1: MSI

triggered by others writing

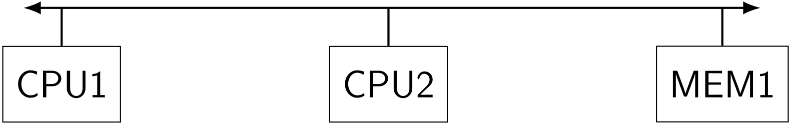


scheme 1: MSI

State	hear read	hear write	read	write
Invalid	—	—	Shared	Modified
Shared	—	to Invalid	Modified	
Modified	Shared	Invalid	—	—

blue: transition sends bus signal

MSI example



CPU1

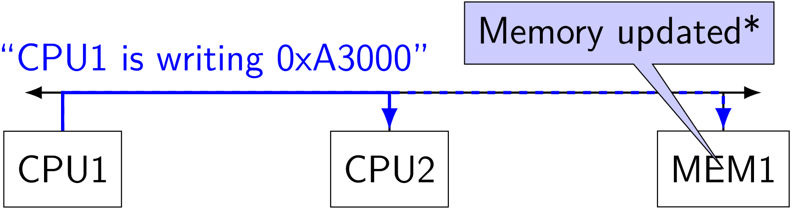
CPU2

MEM1

address	value	state
0xA300	100	Shared
0xC400	200	Shared
0xE500	300	Shared

address	value	state
0x9300	172	Shared
0xA300	100	Shared
0xC500	200	Shared

MSI example



address	value	state
0xA300	~~100~~101	Modified
0xC400	200	Shared
0xE500	300	Shared

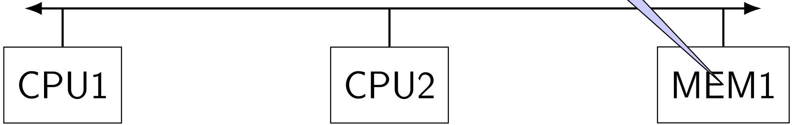
address	value	state
0x9300	172	Shared
~~0xA300~~	~~100~~	Invalid
0xC500	200	Shared

Cache sees write: invalidate 0xA300

CPU1 writes 101 to 0xA300

MSI example

Nothing changed yet (writeback)



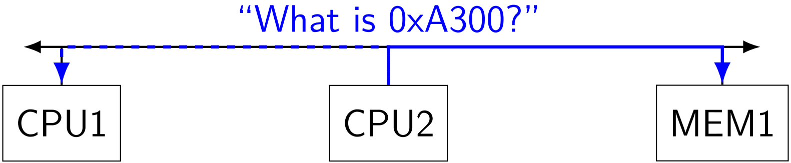
address	value	state
0xA300	~~101~~102	Modified
0xC400	200	Shared
0x		

address	value	state
0x9300	172	Shared
~~0xA300~~	~~100~~	Invalid
		Shared

Modified state — nothing communicated!

CPU1 writes 102 to 0xA300

MSI example



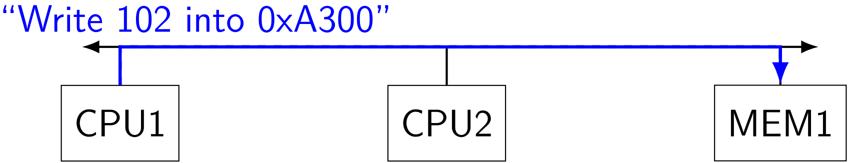
address	value	state
0xA300	102	
0xC400	200	Shared
0		

address	value	state
0x9300	172	Shared
~~0xA300~~	~~100~~	Invalid
		Shared

Modified state — must update for CPU2!

CPU2 reads 0xA300

MSI example



address	value	state
0xA300	~~102~~	Shared
0xC400	200	Shared
0xE		

address	value	state
0x9300	172	Shared
~~0xA300~~	~~100~~	Invalid
		Shared

Written back to memory early
(could also become Invalid at CPU1)

CPU2 reads 0xA300

update memory

to write value (enter modified state), only need to invalidate others

more efficient: shorter bus message

on cache replacement/writeback

still happens — e.g. want to store something else

changes state to invalid

requires writeback if modified (= dirty bit)

scheme 1: MSI

Modified value is different than memory *and*
I am the only one who has it

Shared value is the same as memory

Invalid I don't have the value; I will need
to ask for it

MSI complaints

modifying (read then write then write) a value often three messages:

initial read from memory

invalidate other caches (and maybe write to memory) on initial write

final writeback

scheme 2: MESI

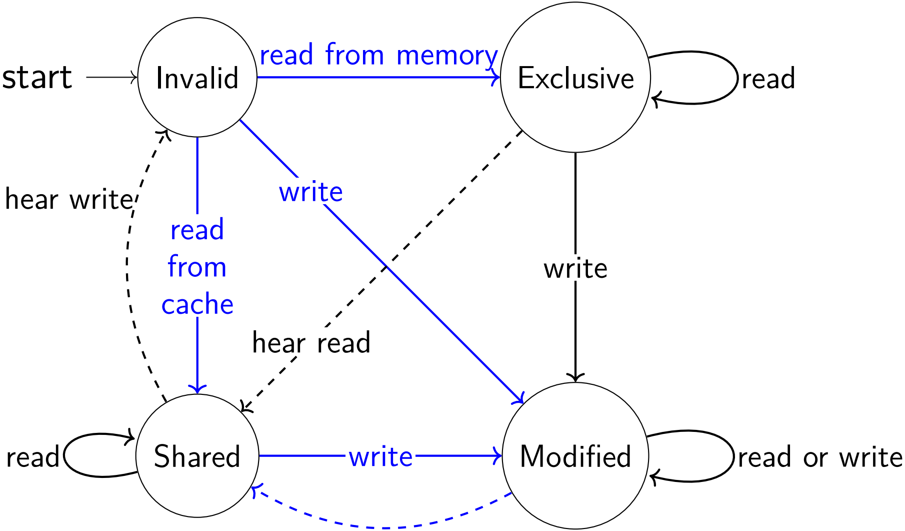
Modified value is different than memory *and* I am the only one who has it

Exclusive value is same as memory *and* I am the only one who has it

Shared value is the same as memory

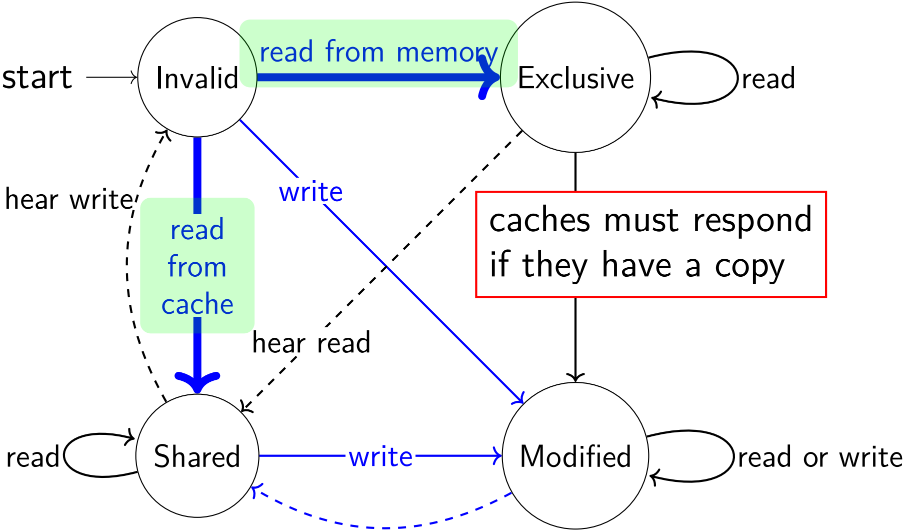
Invalid I don't have the value; I will need to ask for it

scheme 2: MESI



blue = message sent

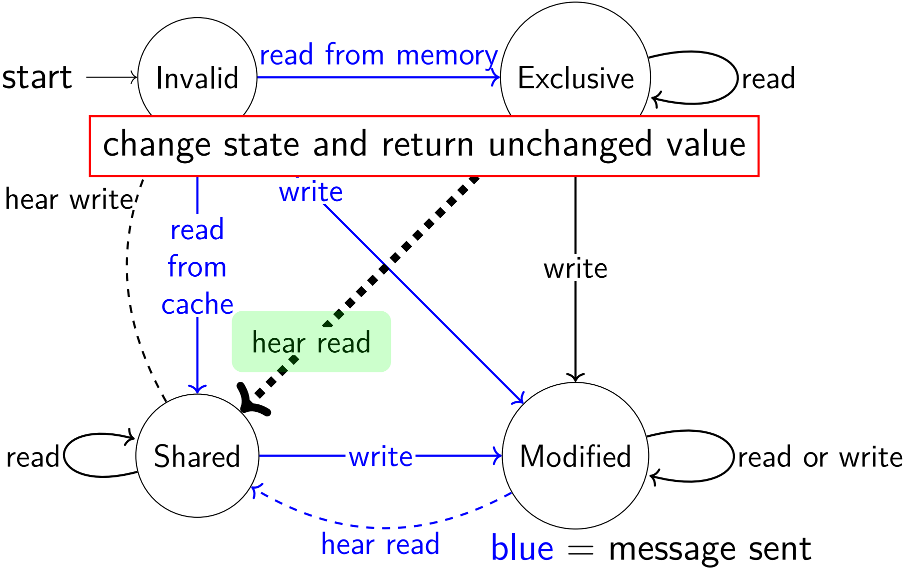
scheme 2: MESI



caches must respond if they have a copy

blue = message sent

scheme 2: MESI



change state and return unchanged value

start

Invalid

read from memory

Exclusive

read

hear write

read from cache

write

write

hear read

Shared

write

Modified

read

read or write

hear read

blue = message sent

read for ownership

reading to modify a value soon?

read into Exclusive state even if reading from cache

invalidate and read

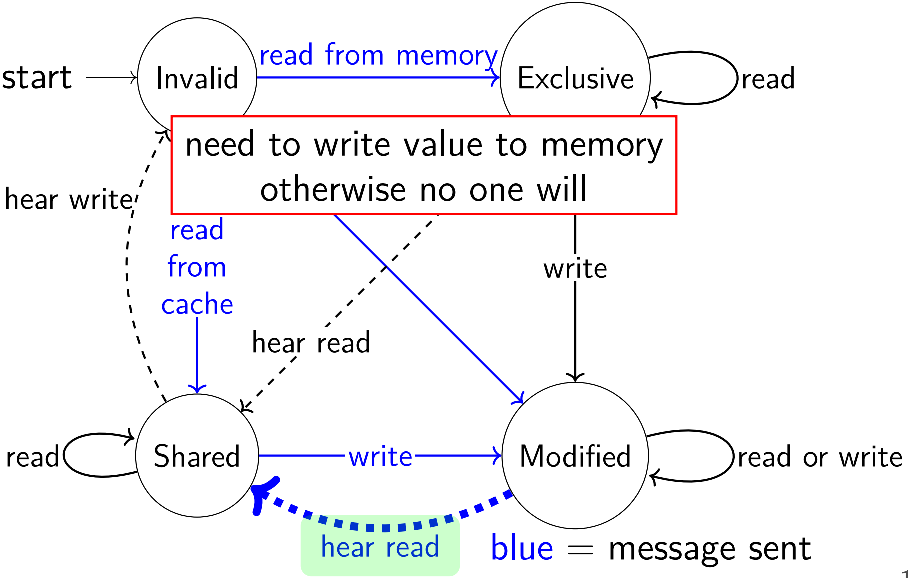
second way to enter exclusive state

MESI complaints

have to update memory to share a modified value …
even though caches read from other caches

read from which cache?

scheme 2: MESI



need to write value to memory
otherwise no one will

blue = message sent

scheme 3: MOESI

Modified value is different than memory *and* I am the only one who has it

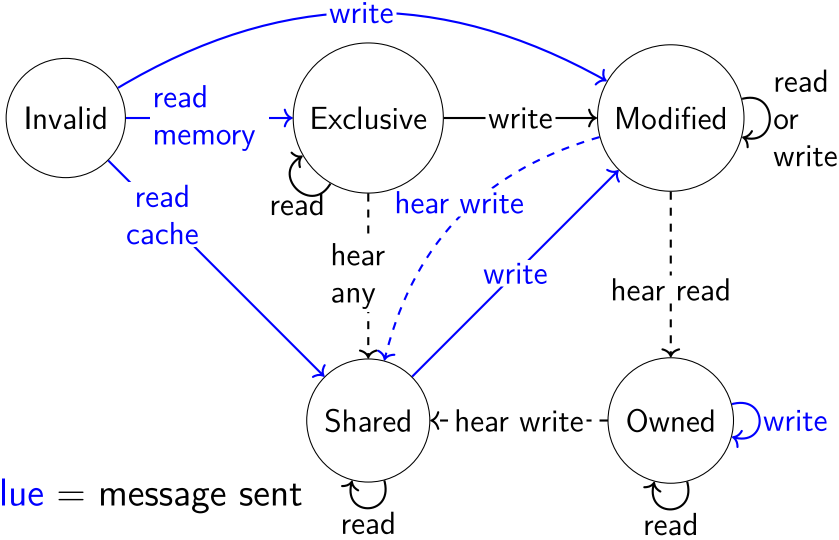
Owned value is different than memory *and* I must update memory

Exclusive value is same as memory *and* I am the only one who has it

Shared value is same as memory **or cache in Owned state**

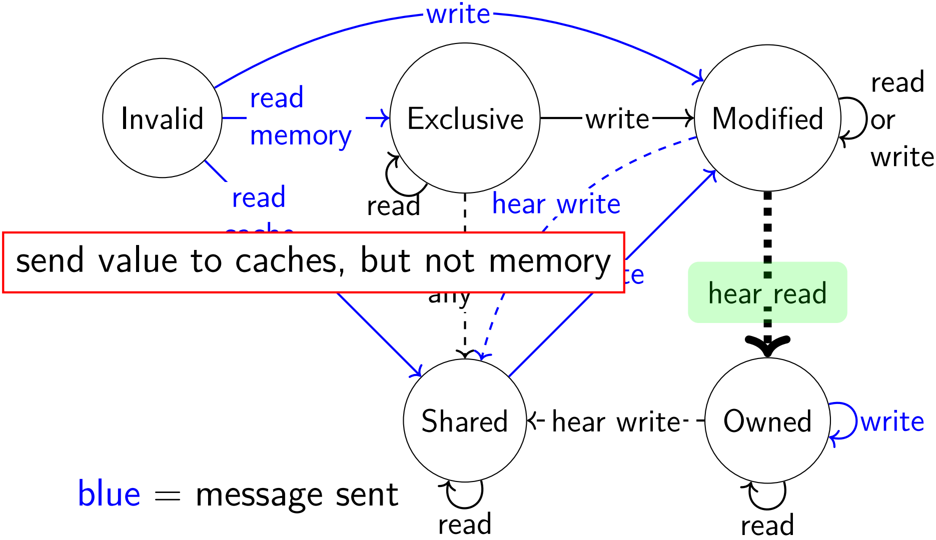
Invalid I don’t have the value

scheme 3: MOESI



blue = message sent

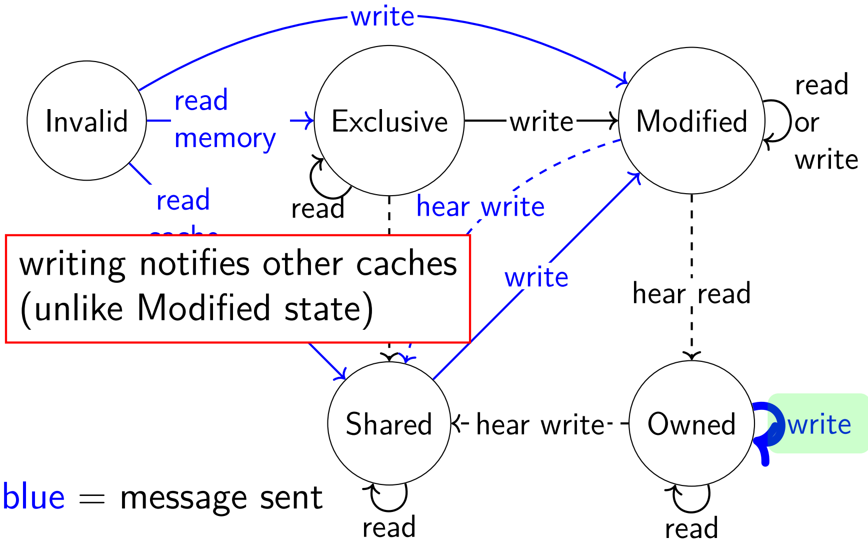
scheme 3: MOESI



send value to caches, but not memory

blue = message sent

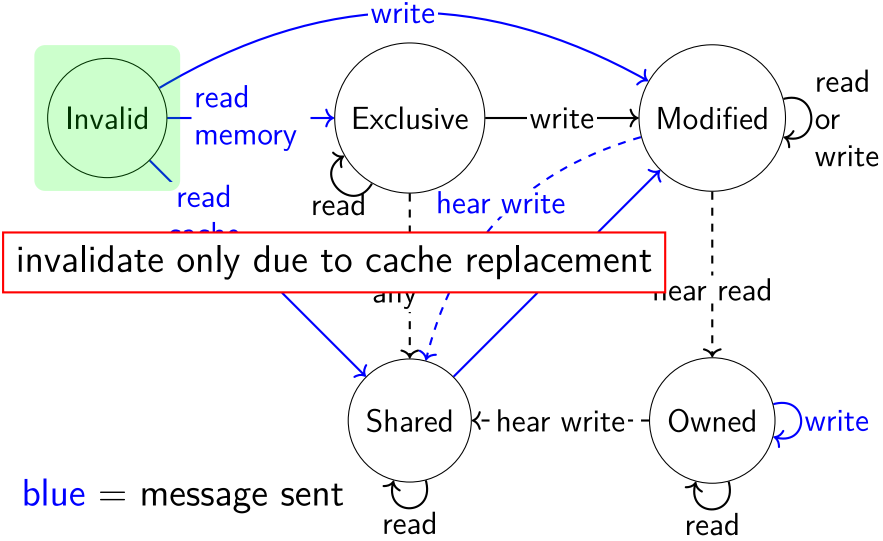
scheme 3: MOESI



writing notifies other caches
(unlike Modified state)

blue = message sent

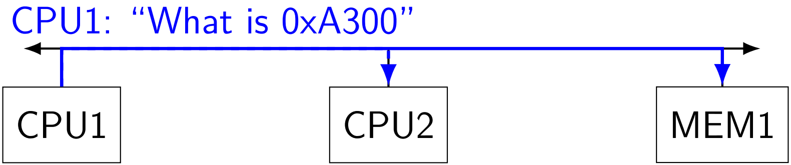
scheme 3: MOESI



invalidate only due to cache replacement

blue = message sent

MOESI example



CPU1: “What is 0xA300”

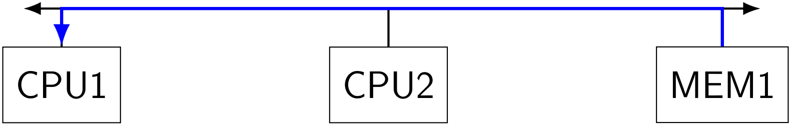
address	value	state

address	value	state

CPU1: read 0xA300
CPU1: write 0xA300
CPU1: read 0xA300
CPU2: read 0xA300
CPU2: write 0xA300

MOESI example

Memory: “0

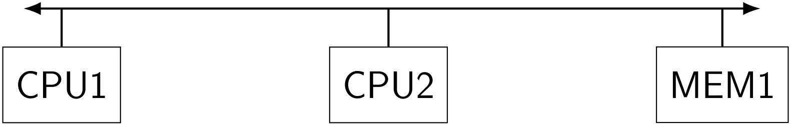


address	value	state
`0xA300`	`100`	Exclusive

address	value	state

CPU1: read 0xA300
CPU1: write 0xA300
CPU1: read 0xA300
CPU2: read 0xA300
CPU2: write 0xA300

MOESI example

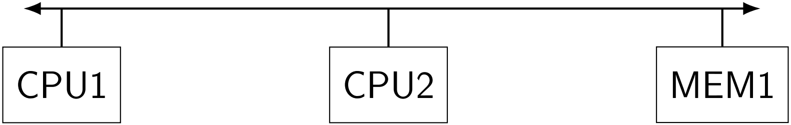


address	value	state
0xA300	~~100~~101	Modified

address	value	state

CPU1: read 0xA300
CPU1: write 0xA300
CPU1: read 0xA300
CPU2: read 0xA300
CPU2: write 0xA300

MOESI example

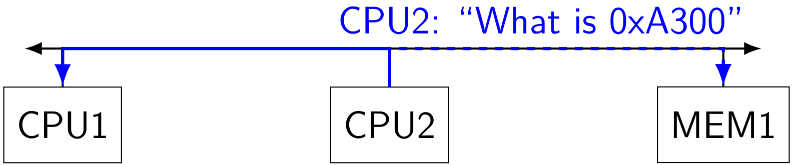


address	value	state
0xA300	101	Modified

address	value	state

CPU1: read 0xA300
CPU1: write 0xA300
CPU1: read 0xA300
CPU2: read 0xA300
CPU2: write 0xA300

MOESI example

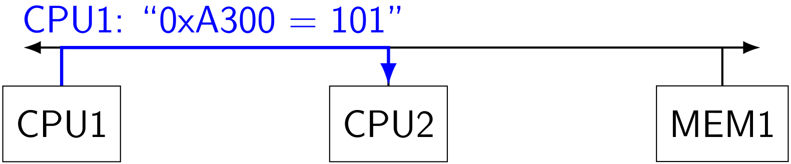


address	value	state
0xA300	101	Modified

address	value	state

CPU1: read 0xA300
CPU1: write 0xA300
CPU1: read 0xA300
CPU2: read 0xA300
CPU2: write 0xA300

MOESI example

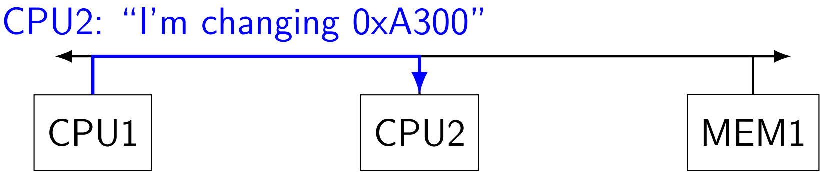


address	value	state
0xA300	101	Owned

address	value	state
0xA300	101	Shared

CPU1: read 0xA300
CPU1: write 0xA300
CPU1: read 0xA300
CPU2: read 0xA300
CPU2: write 0xA300

MOESI example


CPU2: “I’m changing 0xA300”

address	value	state
0xA300	~~101~~	Invalid

address	value	state
0xA300	~~101~~102	Modified

CPU1: read 0xA300
CPU1: write 0xA300
CPU1: read 0xA300
CPU2: read 0xA300
CPU2: write 0xA300

MSI versus MESI versus MOESI

CPU1:	read	0xA300	
CPU1:	write	0xA300	MSI: invalidate
CPU1:	read	0xA300	
CPU2:	read	0xA300	MSI/MESI: memory write
CPU2:	write	0xA300	MSI: invalidate

Other cache coherency options

can invalidate instead of updating other caches on write

invalidation message faster to send than new value

tradeoff: faster if other cache won't use value

Dropping states from MOESI

Modified value is different than memory *and* I am the only one who has it

Owned value is different than memory *and* I must update memory

Exclusive value is same as memory *and* I am the only one who has it

Shared value is same as memory or cache in Owned state

Invalid I don't have the value

Dropping states from MOESI

Modified value is different than memory *and* I am the only one who has it

Owned value is different than memory *and* I must update memory

~~**Exclusive**~~ ~~value is same as memory *and* I am the only one who has it~~

Shared value is same as memory or cache in Owned state

Invalid I don't have the value

Mapping to the paper

MSI + reread to get in Modified: Synapse

MESI + full-write-to-invalidate: write-once

MOSI + forward-on-write: Berkeley

MESI + forward-on-write: Illinois

MESI + invalidate-on-write: Firefly

MOESI + forward-on-write: Dragon

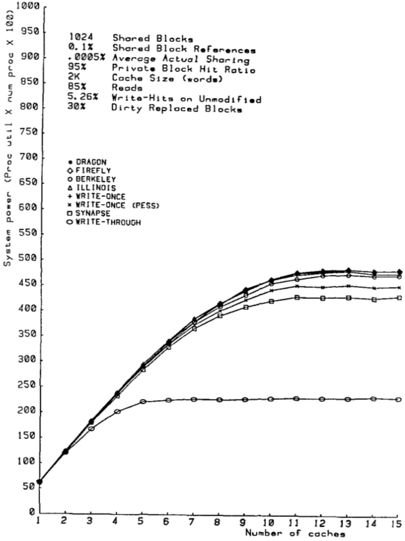
“System Power”

sum of processor utilizations

how much time are CPUs spending waiting for bus

what about overlapping cache accesses and computation??

overhead if almost no shared data



overheads without sharing data

sending invalidation signals no other cache needs

reloading value from memory no cache needs (Synapse)

simulation caveats

workloads?

variation in hardware?

false sharing

cache blocks are shared even if you are accessing different parts

huge performance problem with writes

Present-day snooping cache coherency

AMD processors use MOESI

Intel uses something called MESIF

plus some techniques we'll talk about next time

MESIF states

Modified value is different than memory *and* I am the only one who has it

Exclusive value is same as memory *and* I am the only one who has it

Shared value is same as memory

Invalid I don't have the value

Forwarding value is same as memory *and* I should provide it if requested

Forwarding state: lower traffic

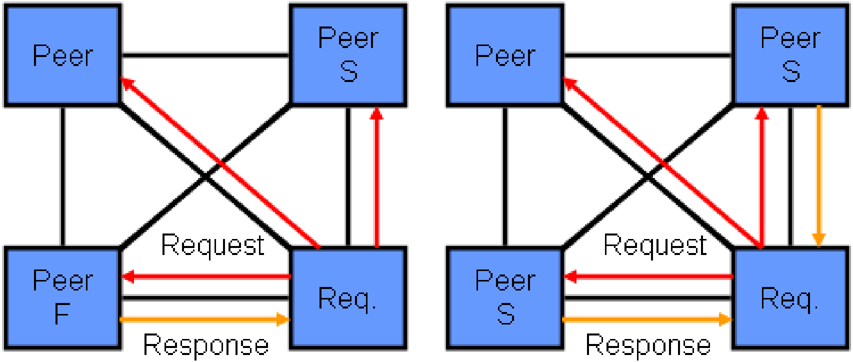


Image from Kanter, “The Common System Interface: Intel’s Future Interconnect”
`http://www.realworldtech.com/common-system-interface/5/`

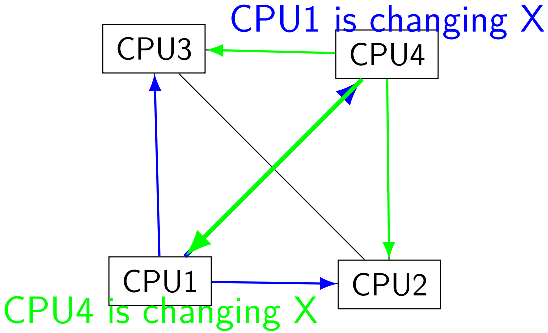
Non-bus topologies

necessary to connect large numbers of caches

higher bandwidth — if you don't broadcast everything

next time: avoiding broadcast

timing trickiness



compare-and-swap

```
compare-and-swap(address, expect-old-value, new-va
    atomically {
        if (expect-old-value == memory[address])
            memory[address] = new-value
        }
    }
}
```

Implementing compare-and-swap

get block into Exclusive or Modified state

- read from memory/cache if necessary
- invalidate other caches if necessary

compare, if value matches, do write (Modified state)

Coherency

common property: single ‘responsible’ cache for possibly changed values

- Owned, Exclusive, Modified states

responsible cache must reply to reads of address

variation:

- when is responsibility acquired? (only on write?)
- when is it relinguished? (only on other’s write?)