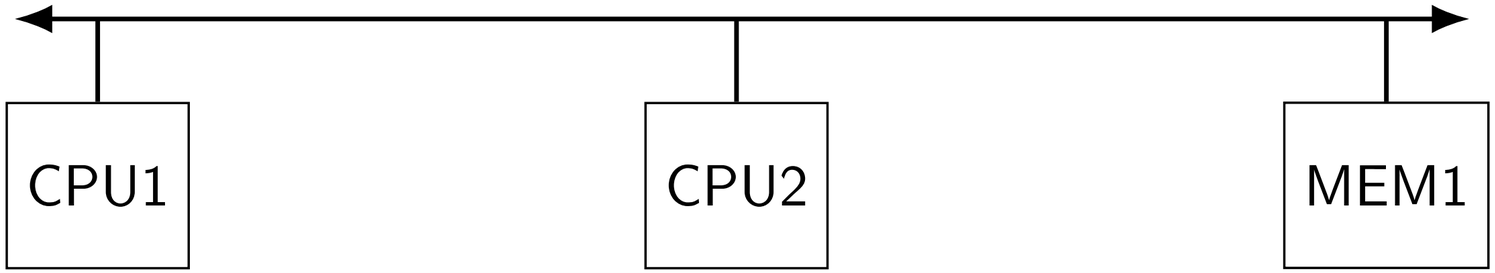
sync-cc

the cache coherency problem

the cache coherency problem



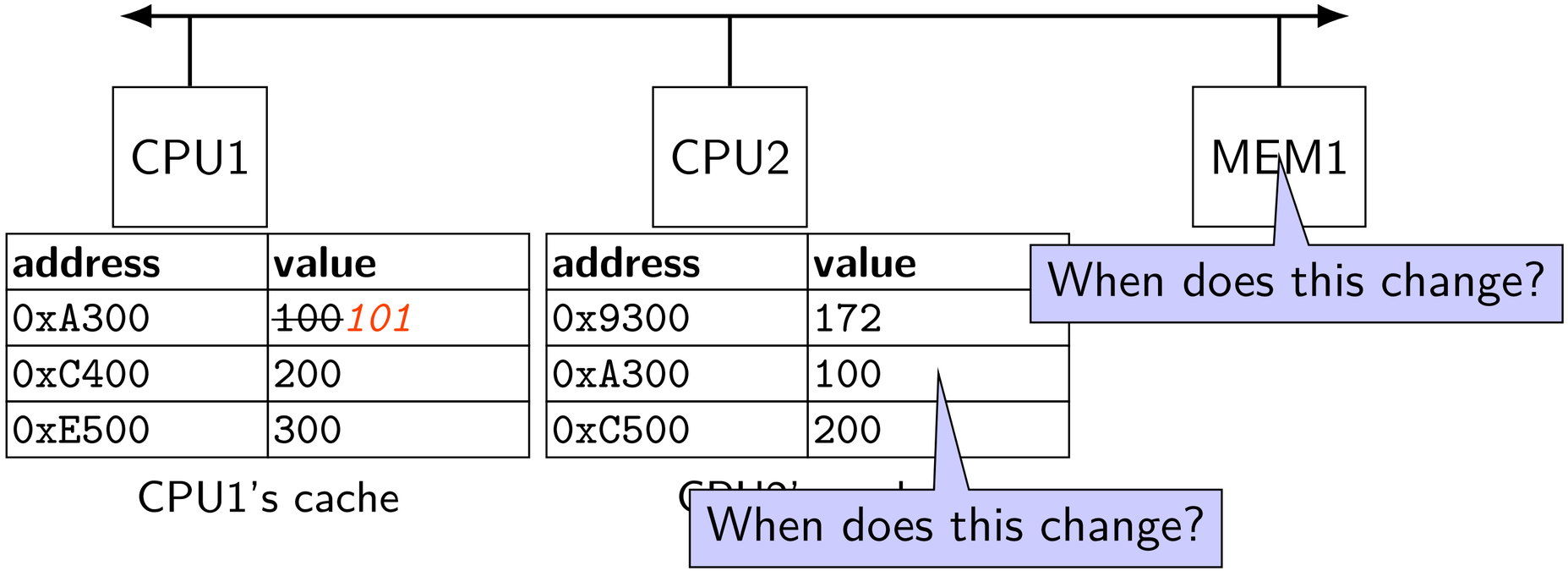
address	value
0xA300	100
0xC400	200
0xE500	300

CPU1's cache

address	value
0x9300	172
0xA300	100
0xC500	200

CPU2's cache

the cache coherency problem



CPU1's cache

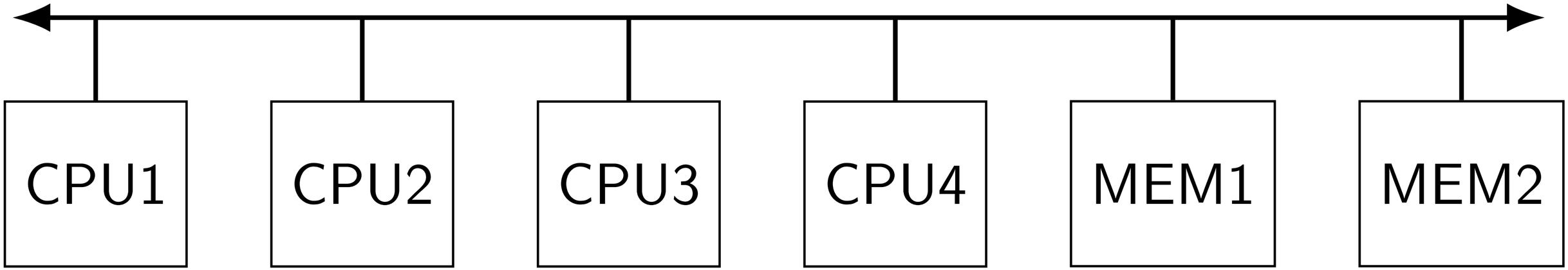
CPU1 writes 101 to 0xA300?

connecting CPUs and memory

multiple processors, common memory

how do processors communicate with memory?

shared bus



one possible design

- we'll revisit later when we talk about I/O

tagged messages — everyone gets everything, filters

contention if multiple communicators

- some hardware enforces only one at a time

shared buses and scaling

shared buses perform poorly with “too many” CPUs

so, there are other designs

we’ll gloss over these for now

shared buses and caches

remember caches?

memory is *pretty slow*

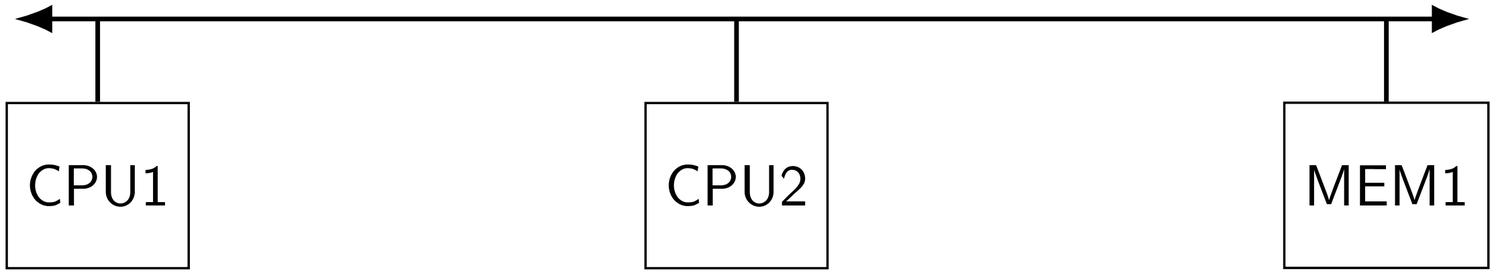
each CPU wants to keep local copies of memory

what happens when multiple CPUs cache same memory?

cache coherency high-level

the cache coherency problem

the cache coherency problem



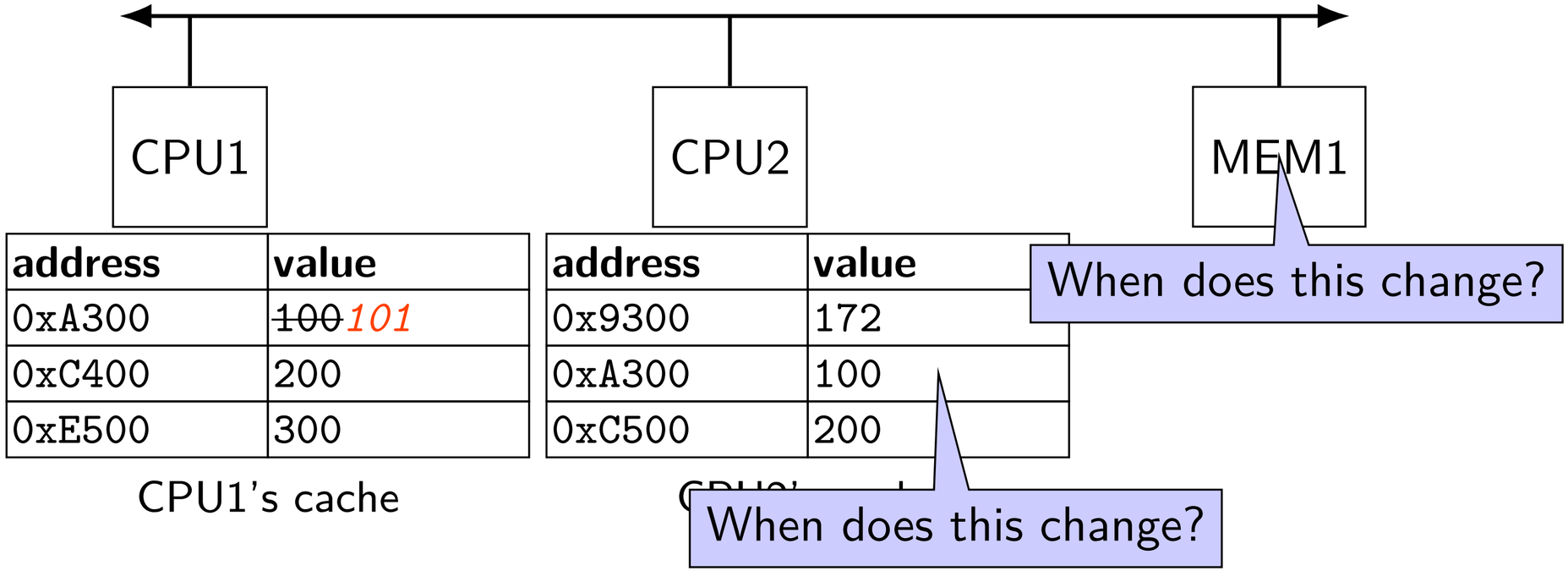
address	value
0xA300	100
0xC400	200
0xE500	300

CPU1's cache

address	value
0x9300	172
0xA300	100
0xC500	200

CPU2's cache

the cache coherency problem



CPU1's cache

CPU1 writes 101 to 0xA300?

"snooping" the bus

every processor already *receives every read/write to memory*

take advantage of this to update caches

idea: use messages to clean up "bad" cache entries

cache coherency states

extra information for *each cache block*

- overlaps with/replaces valid, dirty bits

stored in *each cache*

update states based on reads, writes *and heard messages on bus*

different caches may have different states for same block

sample states:

- Modified: cache has updated value
- Shared: cache is only reading, has same as memory/others
- Invalid

scheme 1: MSI

from state	hear read	hear write	read	write
\hline{} Invalid	—	—	blueto Shared	blueto Modified
Shared	—	to Invalid	—	blueto Modified
Modified	blueto Shared	blueto Invalid	—	—

blueblue: transition requires sending message on bus

example: write while Shared

- must send write — inform others with Shared state
- then change to Modified

example: hear write while Shared

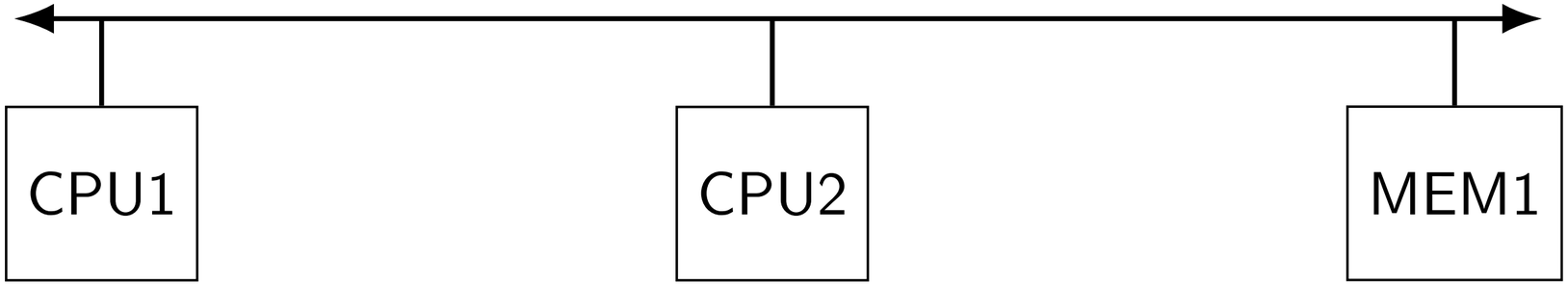
- change to Invalid
- can send read later to get value from writer

example: write while Modified

- nothing to do — no other CPU can have a copy

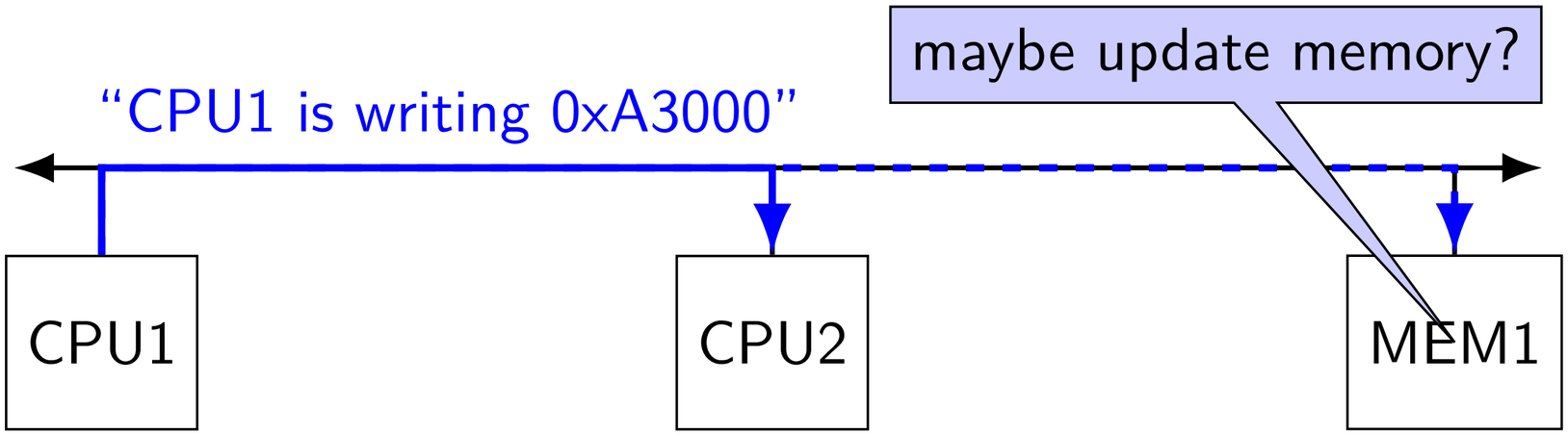
MSI example

MSI example



address	value	state	address	value	state
0xA300	100	Shared	0x9300	172	Shared
0xC400	200	Shared	0xA300	100	Shared
0xE500	300	Shared	0xC500	200	Shared

MSI example

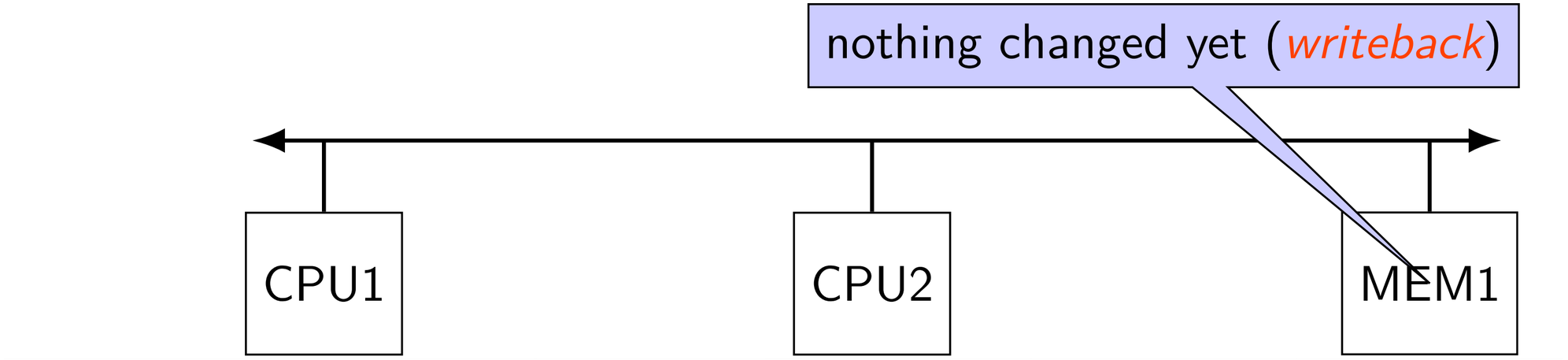


address	value	state	address	value	state
0xA300	~~100~~*101*	*Modified*	0x9300	172	Shared
0xC400	200	Shared	~~0xA300~~	~~100~~	*Invalid*
0xE500	300	Shared	0xC500	200	Shared

cache sees write:
invalidate 0xA300

CPU1 writes 101 to 0xA300

MSI example

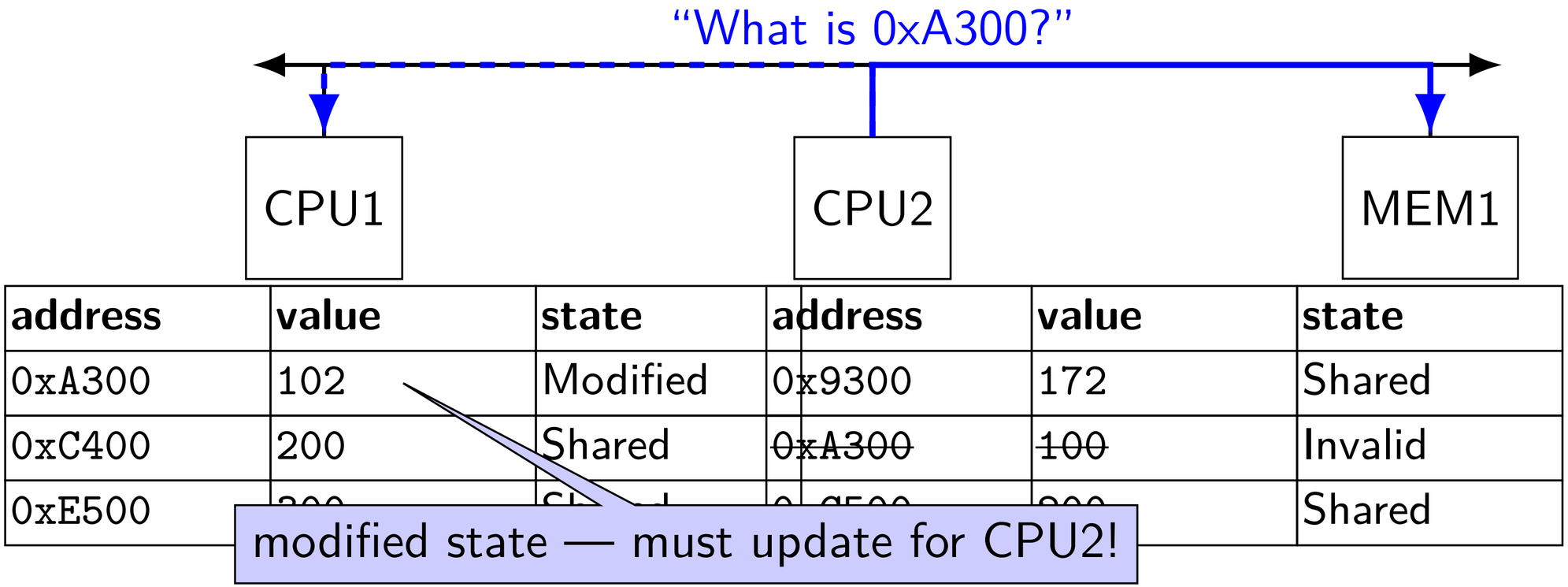


address	value	state	address	value	state
0xA300	~~101~~*102*	Modified	0x9300	172	Shared
0xC400	200	Shared	~~0xA300~~	~~100~~	Invalid
0xE500					Shared

modified state — nothing communicated!
will "fix" later if there's a read

CPU1 writes 102 to 0xA300

MSI example

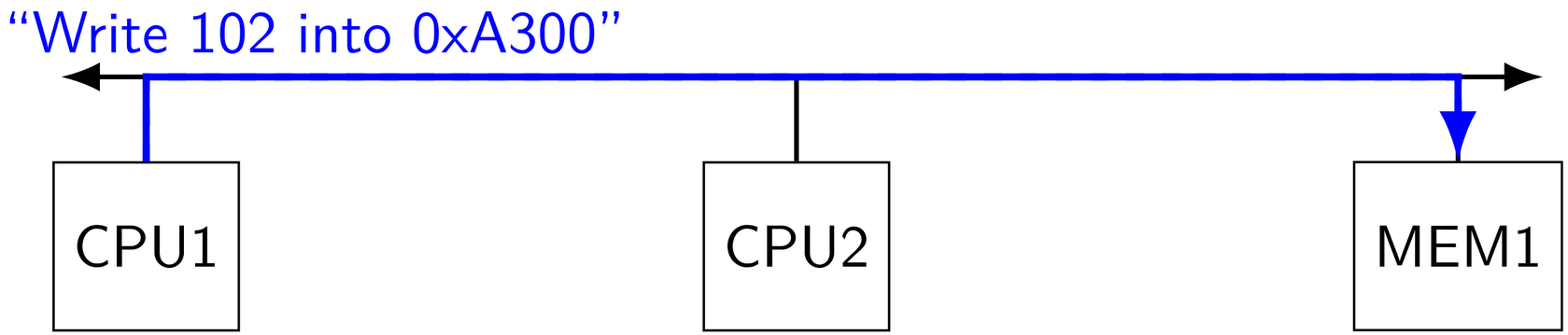


address	value	state	address	value	state
0xA300	102	Modified	0x9300	172	Shared
0xC400	200	Shared	~~0xA300~~	~~100~~	Invalid
0xE500	300	Shared	0xE500	300	Shared

modified state — must update for CPU2!

CPU2 reads 0xA300

MSI example

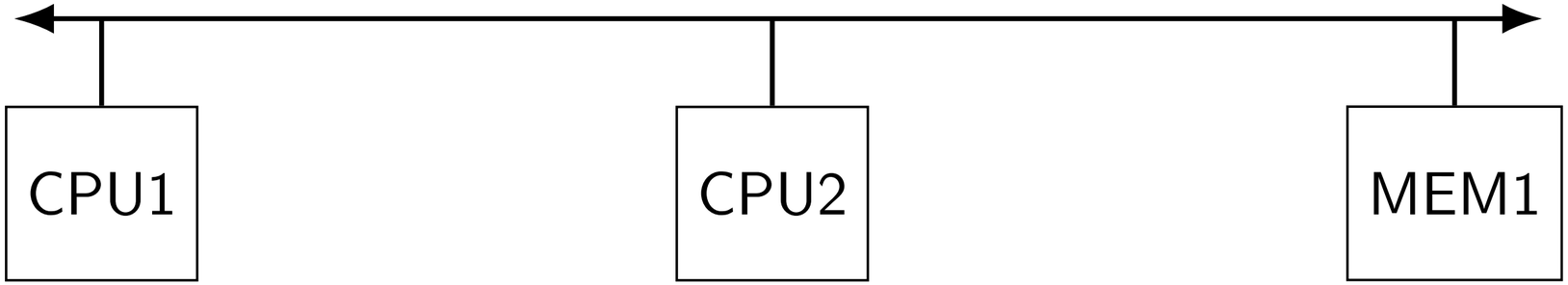


address	value	state	address	value	state
0xA300	*102*	*Shared*	0x9300	172	Shared
0xC400	200	Shared	~~0xA300~~	~~100~~	Invalid
0xE500	[illegible]	[illegible]	[illegible]	[illegible]	Shared

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

CPU2 reads 0xA300

MSI example



address	value	state	address	value	state
0xA300	102	Shared	0x9300	172	Shared
0xC400	200	Shared	~~0xA300~~	~~100~~ *102*	*Shared*
0xE500	300	Shared	0xC500	200	Shared

MSI: update memory

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

can avoid sending actual value (shorter message/faster)

“I am writing address X” versus ``I am writing Y to address X”

MSI: on cache replacement/writeback

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

changes state to *invalid*

requires writeback if modified (= dirty bit)

MSI state summary

Modified	value may be *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 extensions

extra states for *unmodified* values where no other cache has a copy

- avoid sending "I am writing" message later

allow values to be sent directly between caches

- (MSI: value needs to go to memory first)

support not sending invalidate/etc. messages to *all* cores

- requires some tracking of what cores have each address
- only makes sense with non-shared-bus design

using a shared the bus

want to change a value other processors might have?

use bus to tell them “get rid of your copy”

want to start using value other processor might have reserved?

use bus to say “I’d like to use this value now”

cache coherency states

extra information for *each cache block*

overlaps with/replaces valid, dirty bits

stored in *each cache*

update states based on reads, writes *and heard messages on bus*

different caches may have different states for same block

MSI state summary

Modified value may be *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 scheme

from state	hear read	hear write	read	write
\hline{} Invalid	—	—	blueto Shared	blueto Modified
Shared	—	to Invalid	—	blueto Modified
Modified	blueto Shared	blueto Invalid	—	—

blueblue: transition requires sending message on bus

example: write while Shared

- must send write — inform others with Shared state
- then change to Modified

example: hear write while Shared

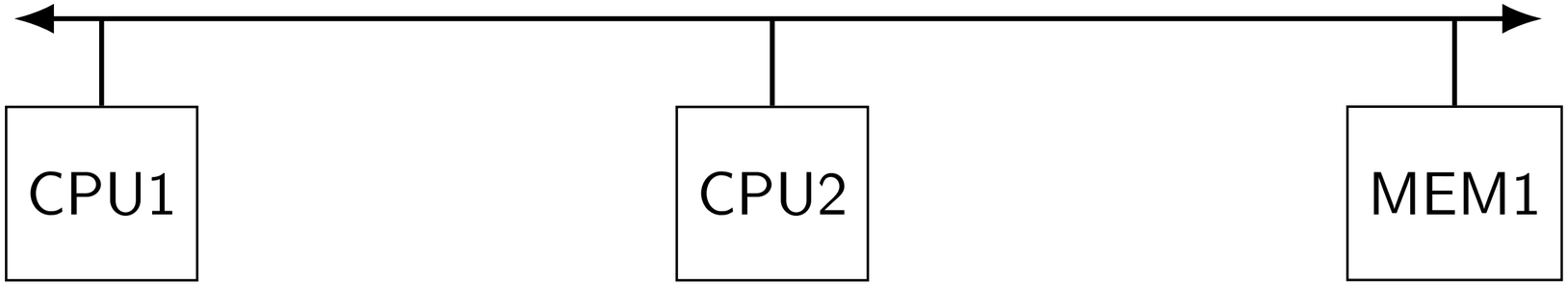
- change to Invalid
- can send read later to get value from writer

example: write while Modified

- nothing to do — no other CPU can have a copy

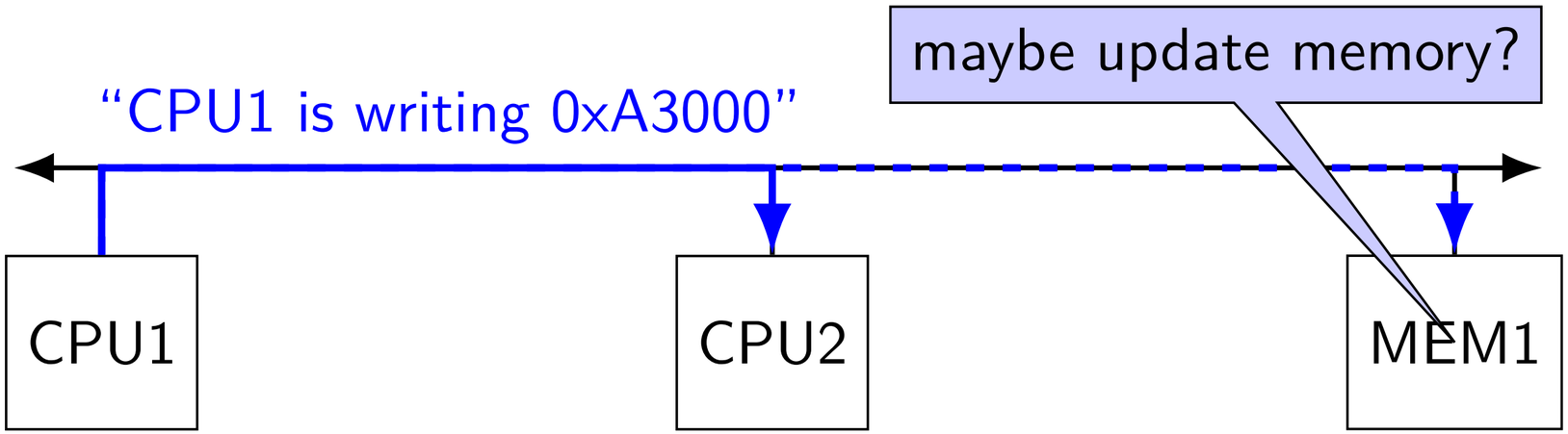
MSI example

MSI example



address	value	state	address	value	state
0xA300	100	Shared	0x9300	172	Shared
0xC400	200	Shared	0xA300	100	Shared
0xE500	300	Shared	0xC500	200	Shared

MSI example

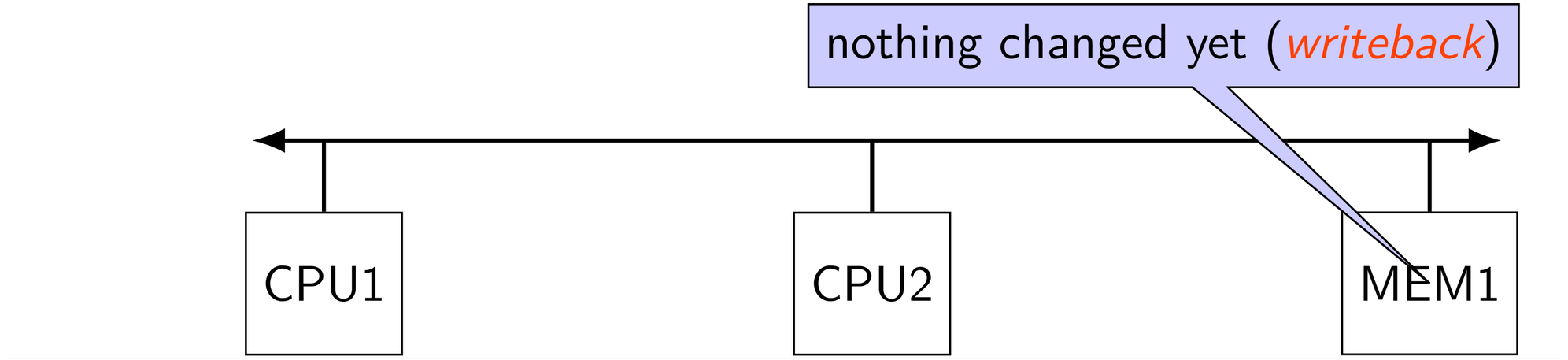


address	value	state	address	value	state
0xA300	~~100~~*101*	*Modified*	0x9300	172	Shared
0xC400	200	Shared	~~0xA300~~	~~100~~	*Invalid*
0xE500	300	Shared	0xC500	200	Shared

cache sees write:
invalidate 0xA300

CPU1 writes 101 to 0xA300

MSI example

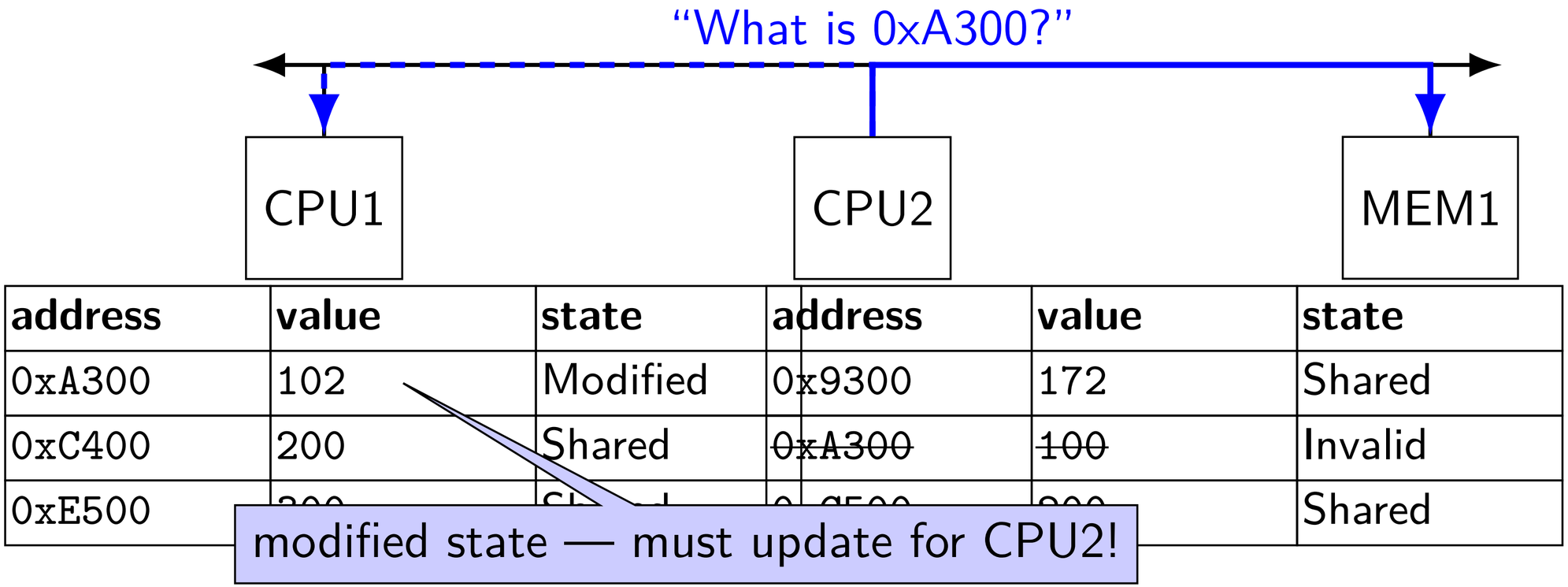


address	value	state	address	value	state
0xA300	~~101~~*102*	Modified	0x9300	172	Shared
0xC400	200	Shared	~~0xA300~~	~~100~~	Invalid
0xE500					Shared

modified state — nothing communicated!
will "fix" later if there's a read

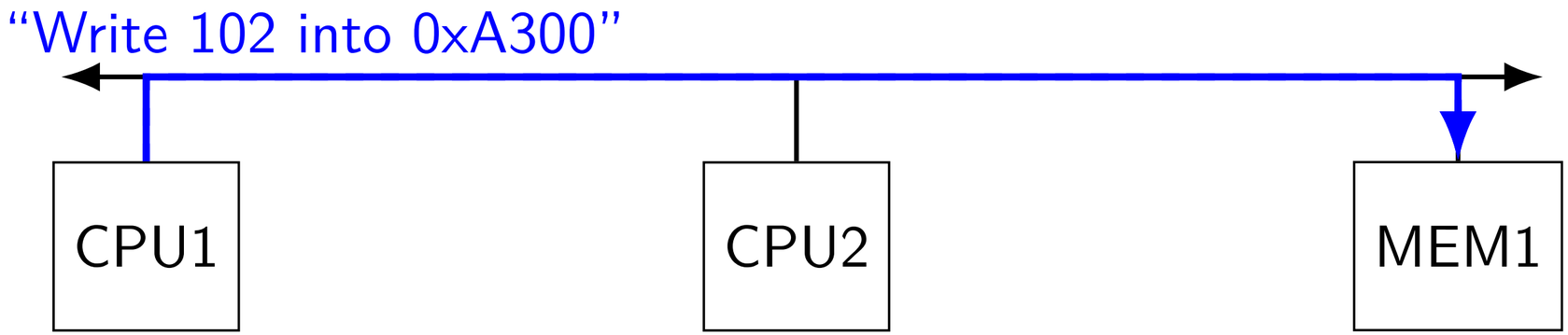
CPU1 writes 102 to 0xA300

MSI example



CPU2 reads 0xA300

MSI example

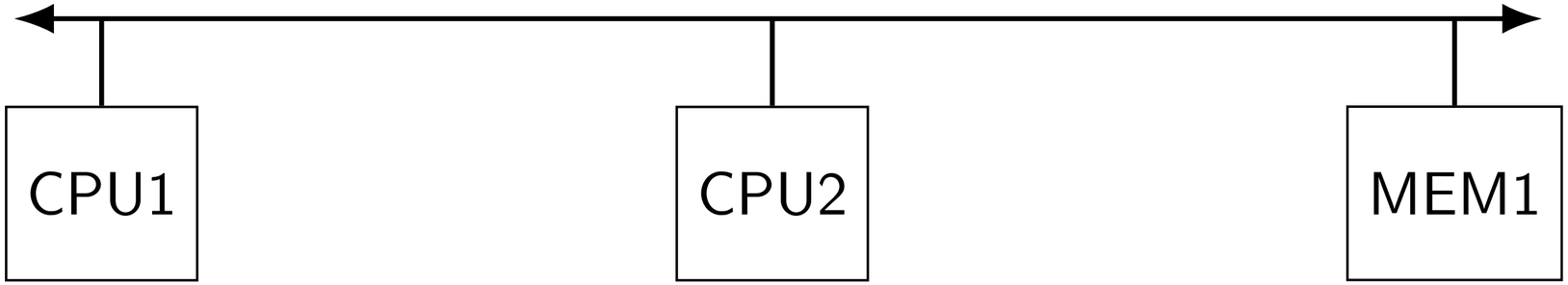


address	value	state	address	value	state
0xA300	*102*	*Shared*	0x9300	172	Shared
0xC400	200	Shared	~~0xA300~~	~~100~~	Invalid
0xE500	[illegible]	[illegible]	[illegible]	[illegible]	Shared

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

CPU2 reads 0xA300

MSI example



address	value	state	address	value	state
0xA300	102	Shared	0x9300	172	Shared
0xC400	200	Shared	~~0xA300~~	~~100~~ *102*	*Shared*
0xE500	300	Shared	0xC500	200	Shared

MSI: update memory

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

can avoid sending actual value (shorter message/faster)

“I am writing address X” versus ``I am writing Y to address X”

MSI: on cache replacement/writeback

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

changes state to *invalid*

requires writeback if modified (= dirty bit)

cache coherency exercise

modified/shared/invalid; all initially invalid; 32B blocks, 8B read/writes

CPU 1: read 0x1000
CPU 2: read 0x1000
CPU 1: write 0x1000
CPU 1: read 0x2000
CPU 2: read 0x1000
CPU 2: write 0x2008
CPU 3: read 0x1008

Q1: final state of 0x1000 in caches?

Modified/Shared/Invalid for CPU 1/2/3
CPU 1: CPU 2: CPU 3:

Q2: final state of 0x2000 in caches?

Modified/Shared/Invalid for CPU 1/2/3
CPU 1: CPU 2: CPU 3:

cache coherency exercise solution

	\multicolumn{3c| 0x1000-0x101f}	\multicolumn{3c 0x2000-0x201f}				
action	CPU 1	CPU 2	CPU 3	CPU 1	CPU 2	CPU 3
\hline{}	I	I	I	I	I	I
CPU 1: read 0x1000	*S*	I	I	I	I	I
CPU 2: read 0x1000	S	*S*	I	I	I	I
CPU 1: write 0x1000	*M*	*I*	I	I	I	I

againframe(cacheCoherenceExSoln)

MSI extensions

real cache coherency protocols sometimes more complex:

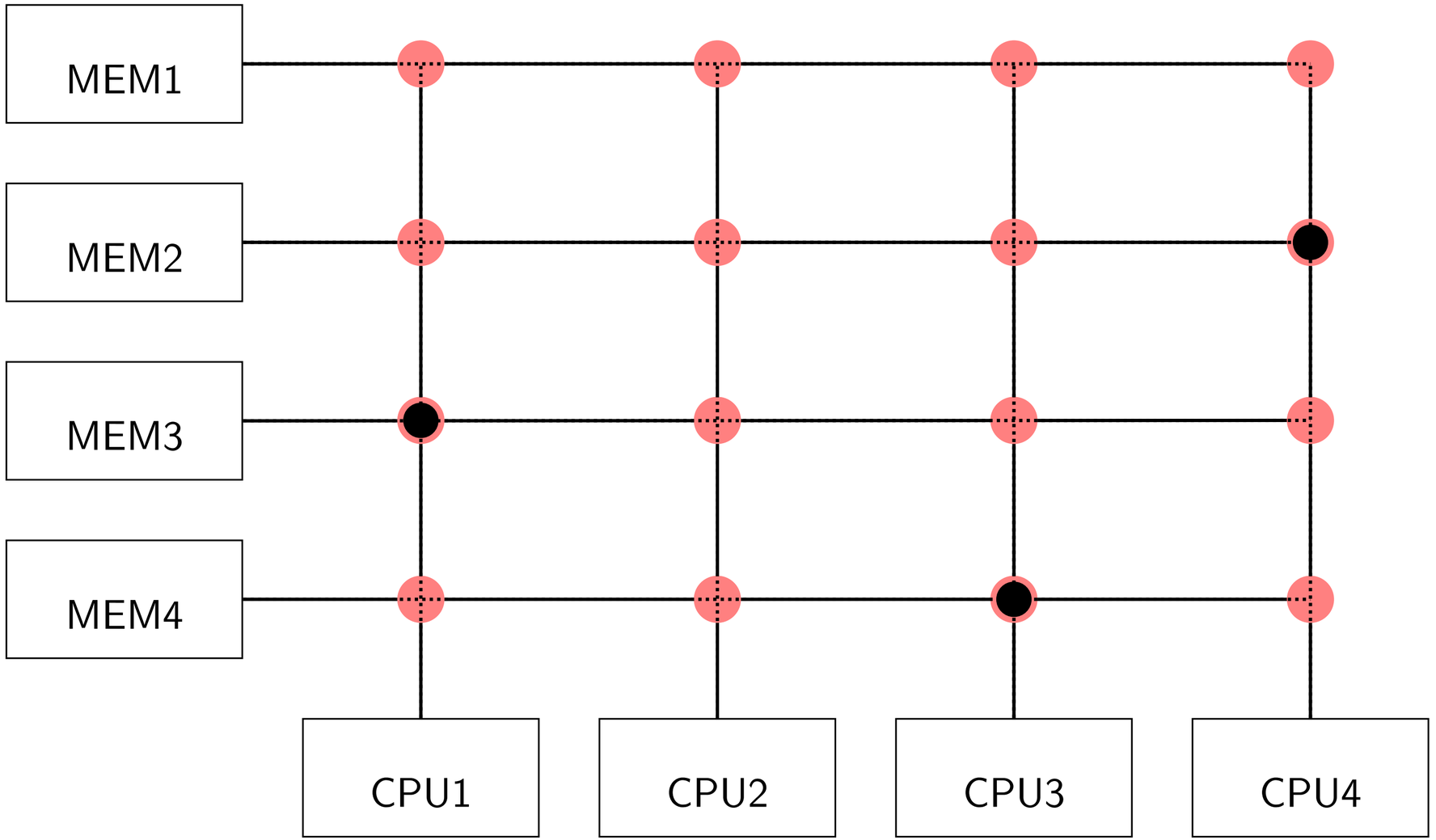
separate tracking modifications from whether other caches have copy

send values directly between caches (maybe skip write to memory)

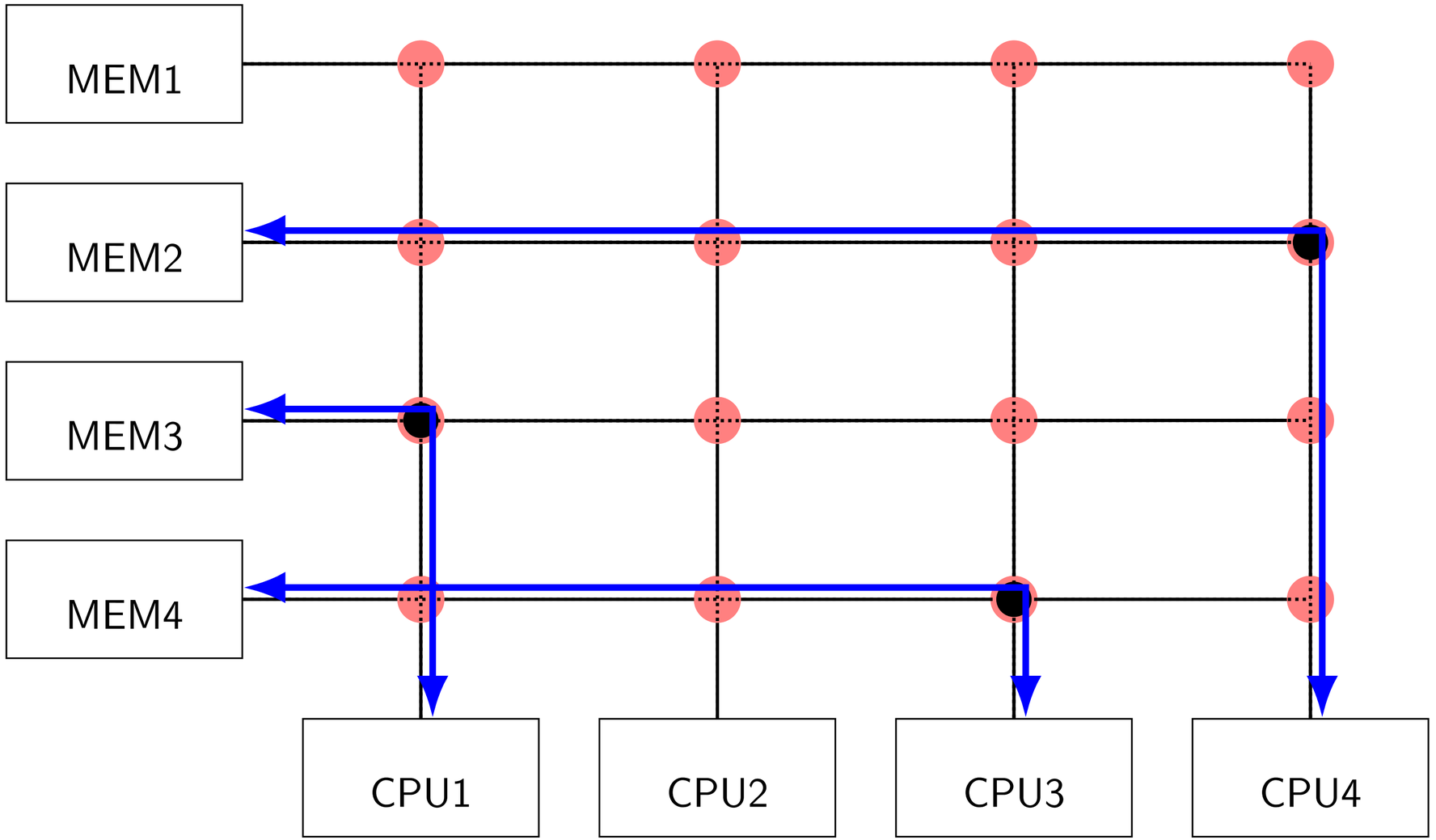
send messages only to cores which might care (no shared bus)

alternative to buses: crossbar

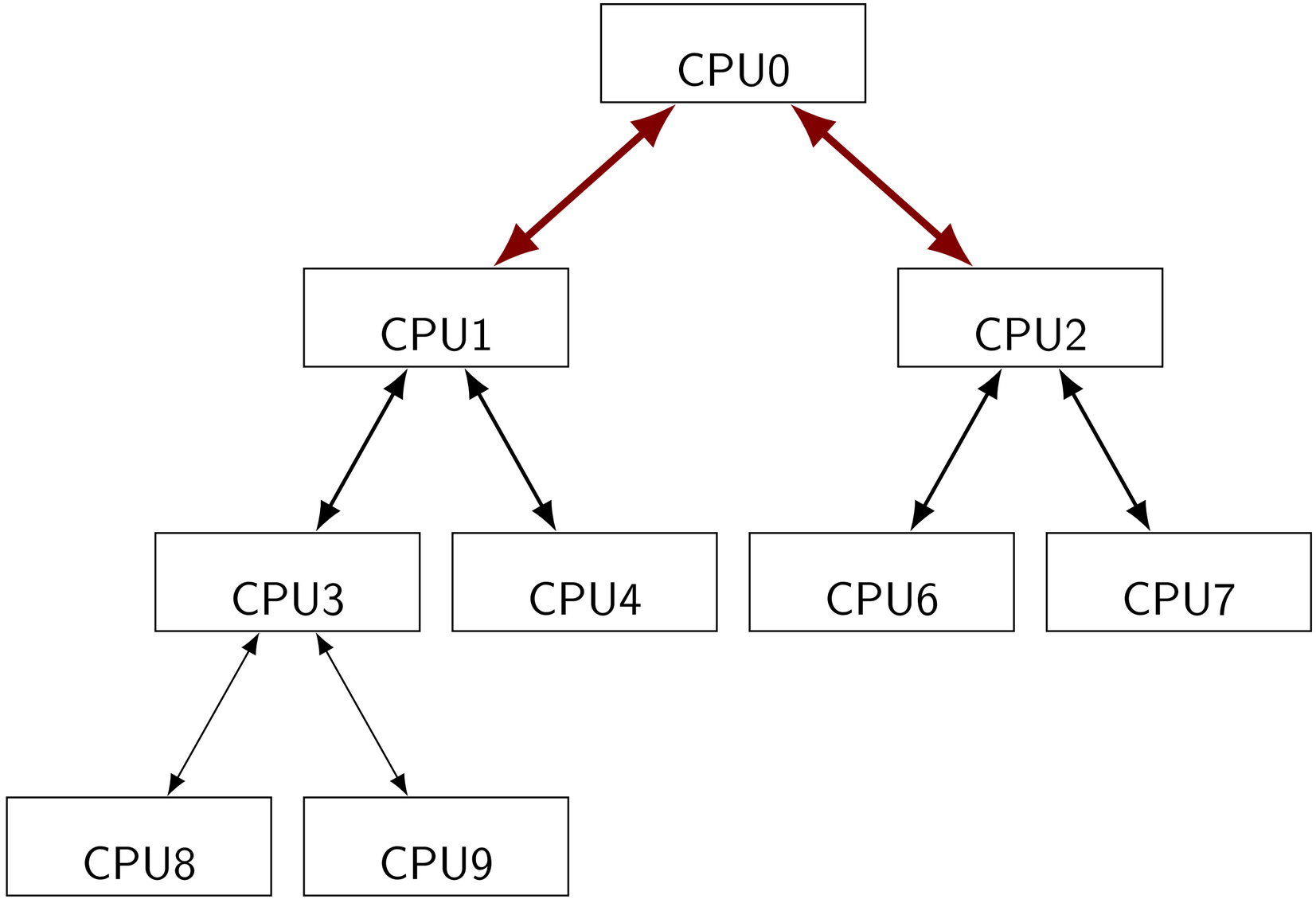
alternative to buses: crossbar



alternative to buses: crossbar



alternatives to buses: trees



MESI / MOESI

MSI complaints

modifying (read 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

read for ownership

reading to modify a value soon?

read into Exclusive state even if reading from cache

invalidate and read

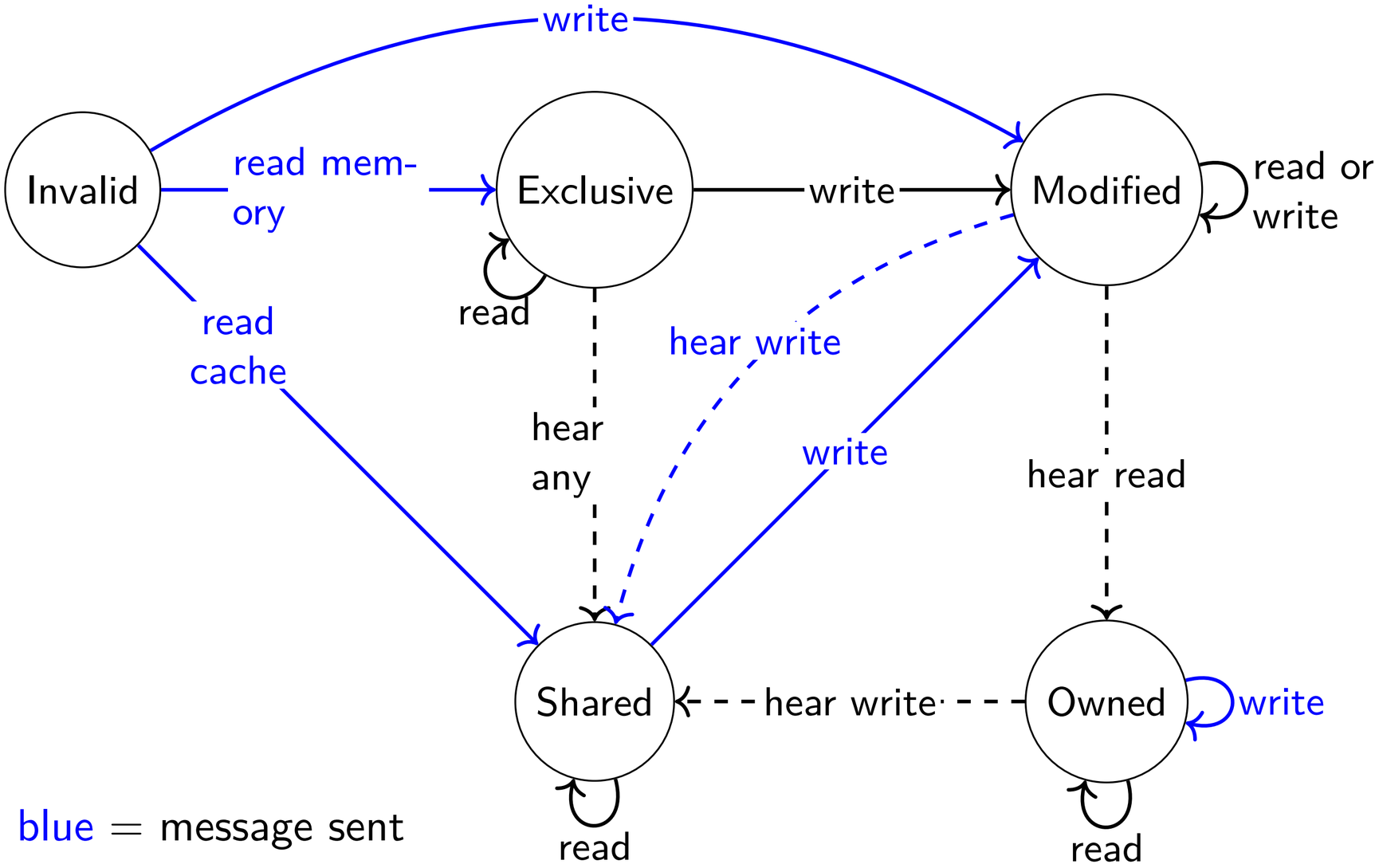
second way to enter exclusive state

MESI complaints

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

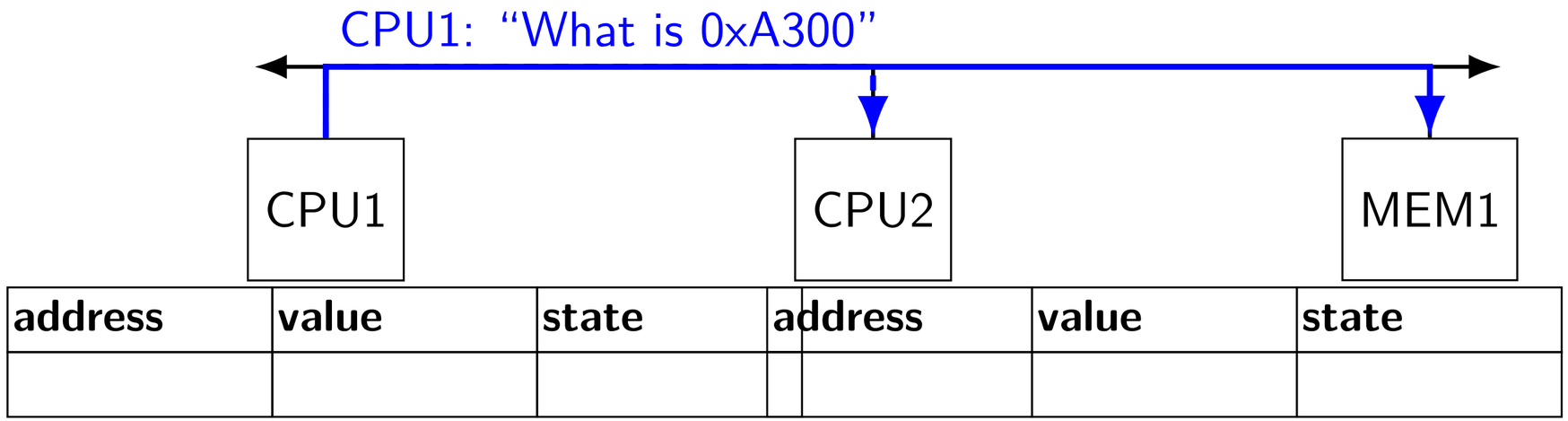
read from which cache?

scheme 3: MOESI



MOESI example

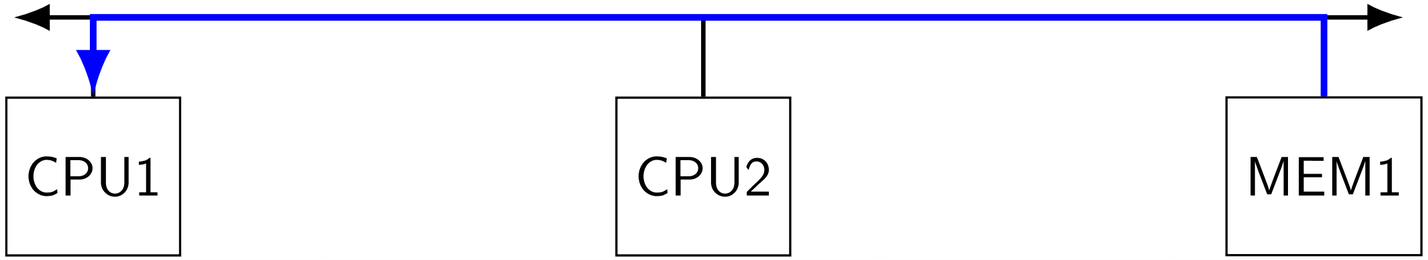
MOESI example



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

MOESI example

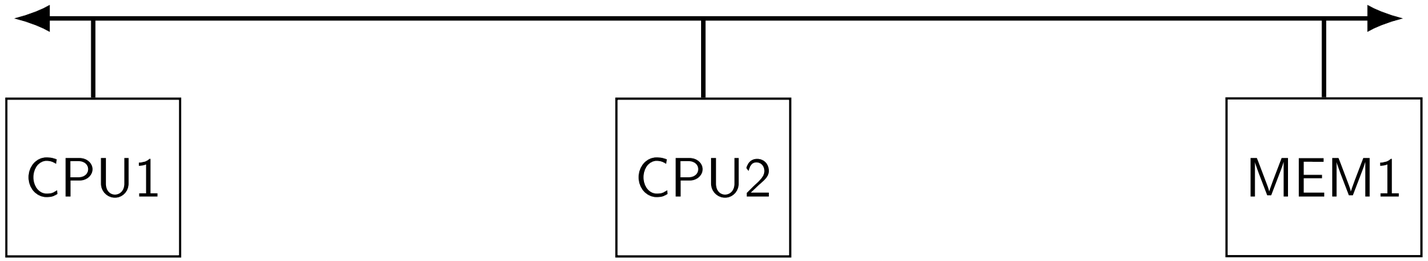
Memory: "0xA300 = 100"



address	value	state	address	value	state
0xA300	100	*Exclusive*			

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

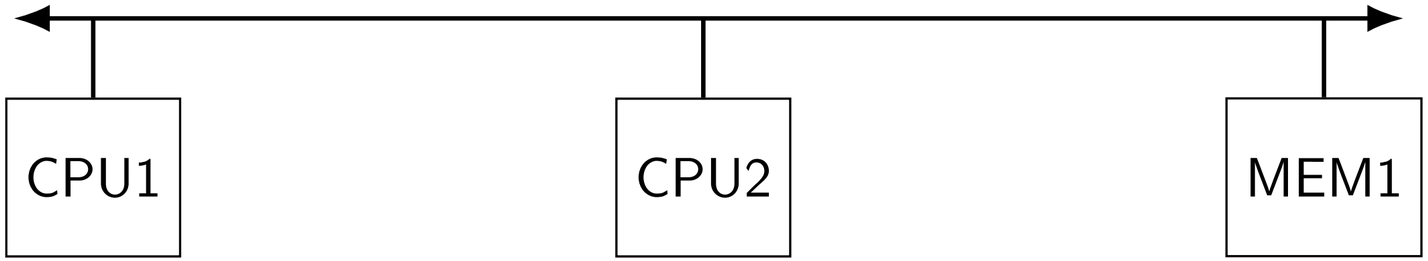
MOESI example



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

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

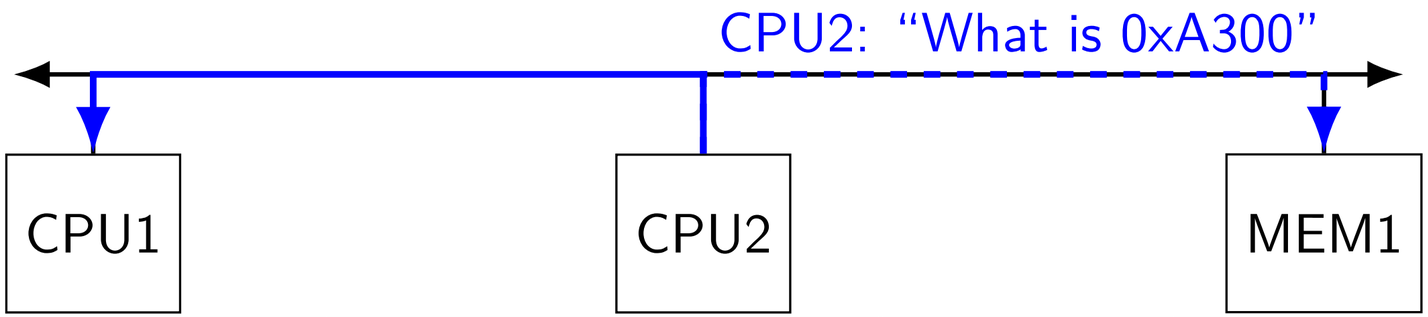
MOESI example



address	value	state	address	value	state
0xA300	101	Modified			

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

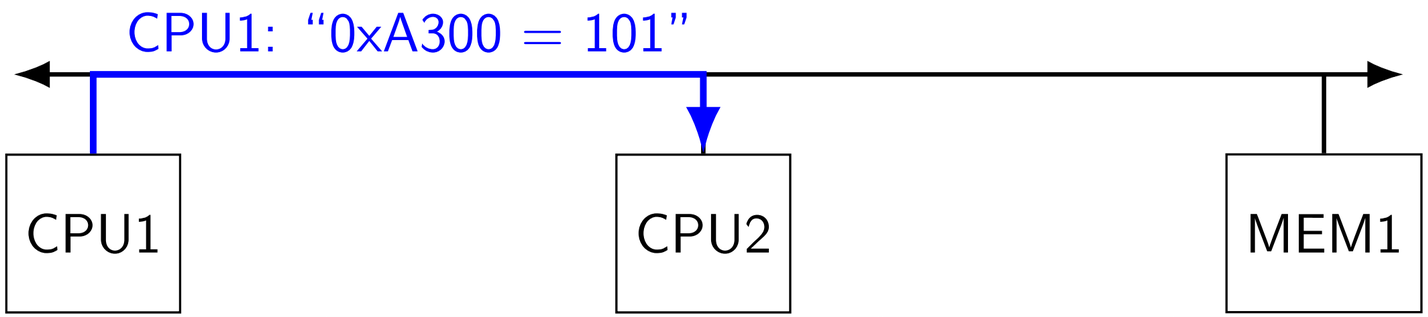
MOESI example



address	value	state	address	value	state
0xA300	101	Modified			

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

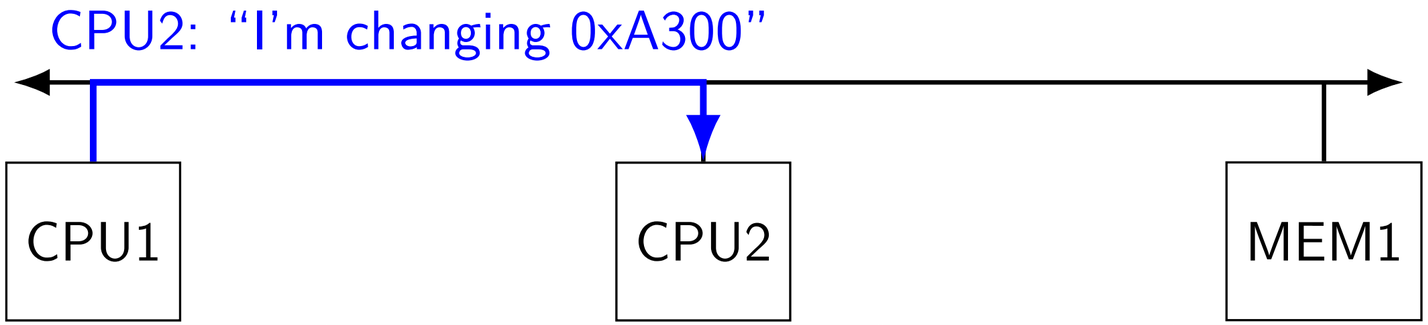
MOESI example



address	value	state	address	value	state
0xA300	101	*Owned*	0xA300	*101*	*Shared*

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

MOESI example



address	value	state	address	value	state
0xA300	~~101~~	*Invalid*	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

Backup slides