Advanced Topics in Multicore Architecture & Software Systems

Cache Coherence

Administration

- HW#2 out tonight, due **December 11th**
    - 1 question on this lecture
    - 1 question on lecture #4
    - Max 3 pages
    - **Can be submitted in pairs**

Orientation

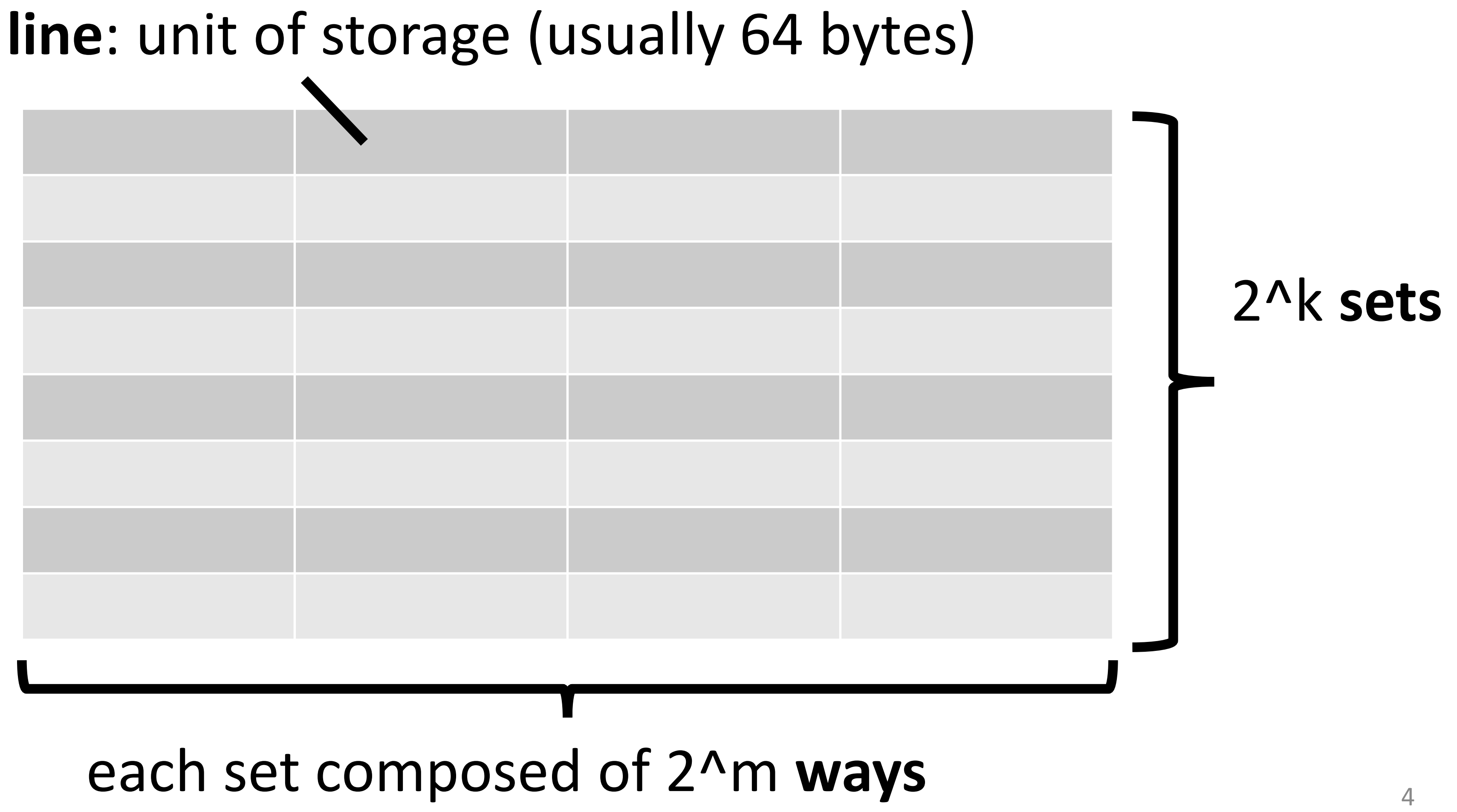
First part of the course: background

- Processors and OoO/speculative execution
- Linearizability
- **Cache coherence**

  $\Rightarrow$ Next week: Algorithmic interlude
- Memory consistency (PL & processors)

Background: CPU cache

Set-associative caches

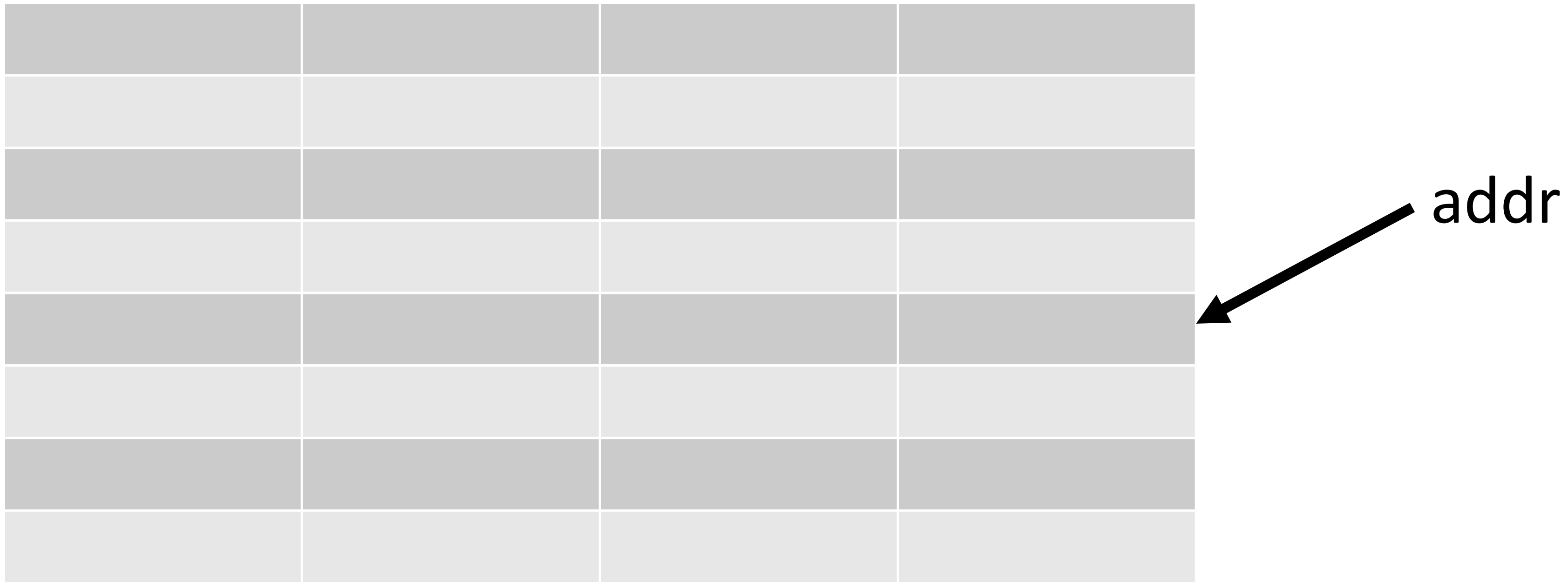

line: unit of storage (usually 64 bytes)
2^k **sets**
each set composed of 2^m **ways**

Background: CPU cache

physical (physically indexed cache)

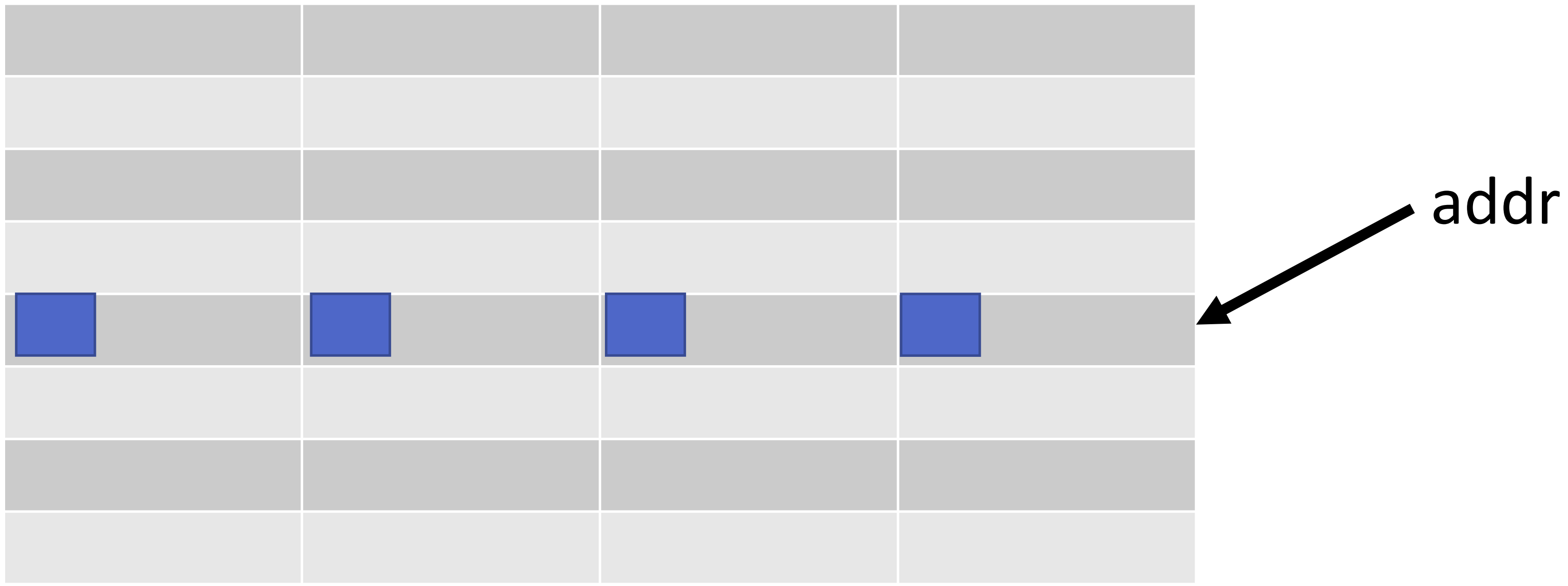
Memory op: addr maps to set

Typically: (addr >> log(line size)) *mod* #sets



Background: CPU cache

Tags in each way searched in parallel to figure out if addr is stored in cache or not

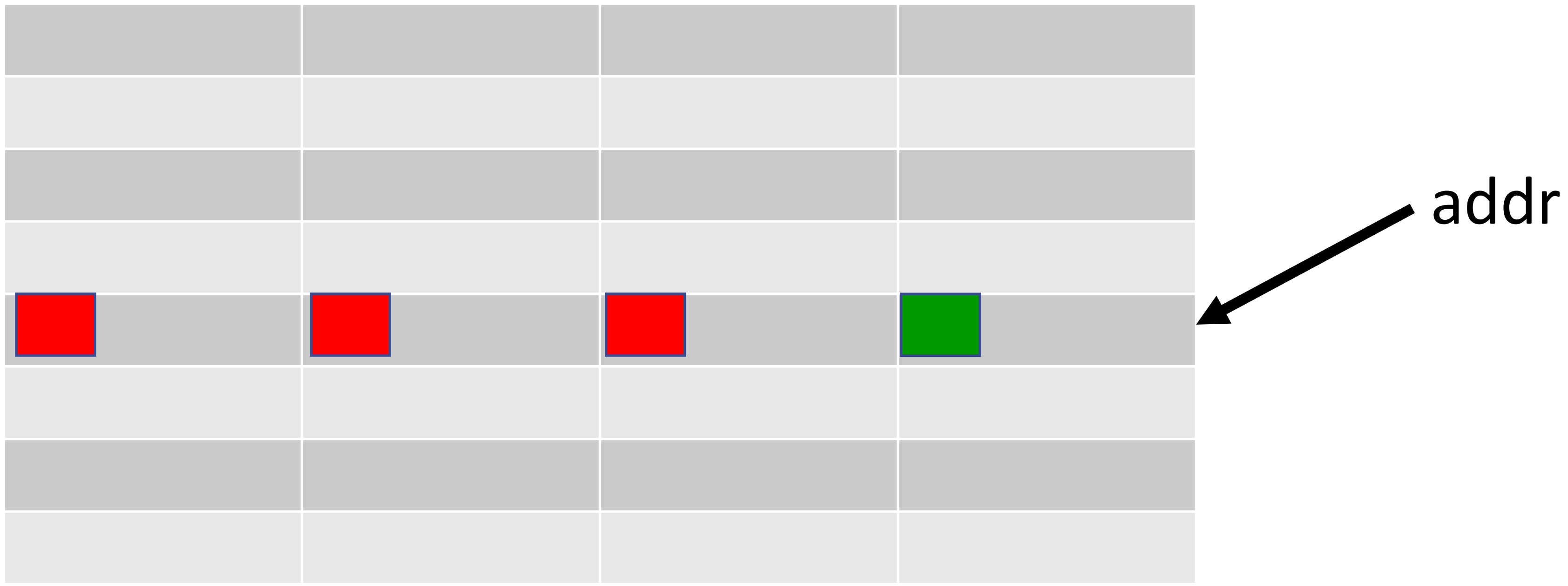


Tag contains high addr bits (addr >> log(line size))

Again, assume physical (physically tagged)

Background: CPU cache

Hit: Line containing address in cache

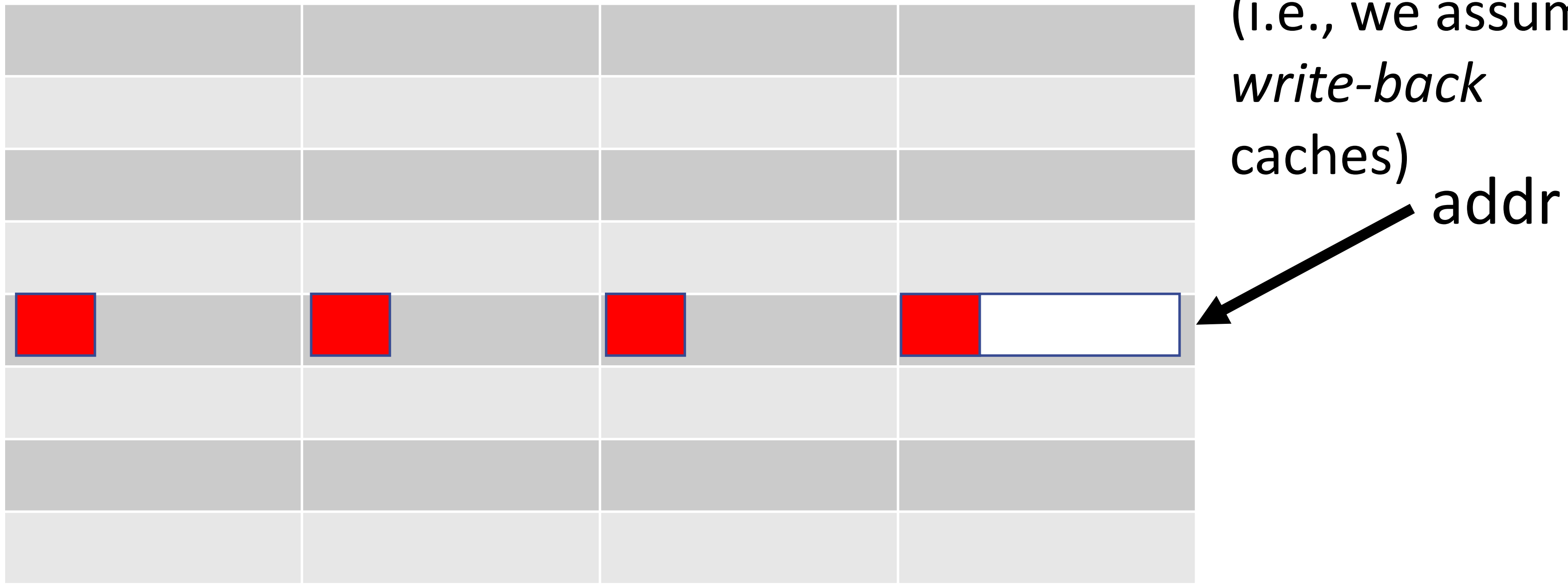


Background: CPU cache

Miss: No line containing address in cache

Cache allocates way for data (even for writes)

(i.e., we assume *write-back* caches)

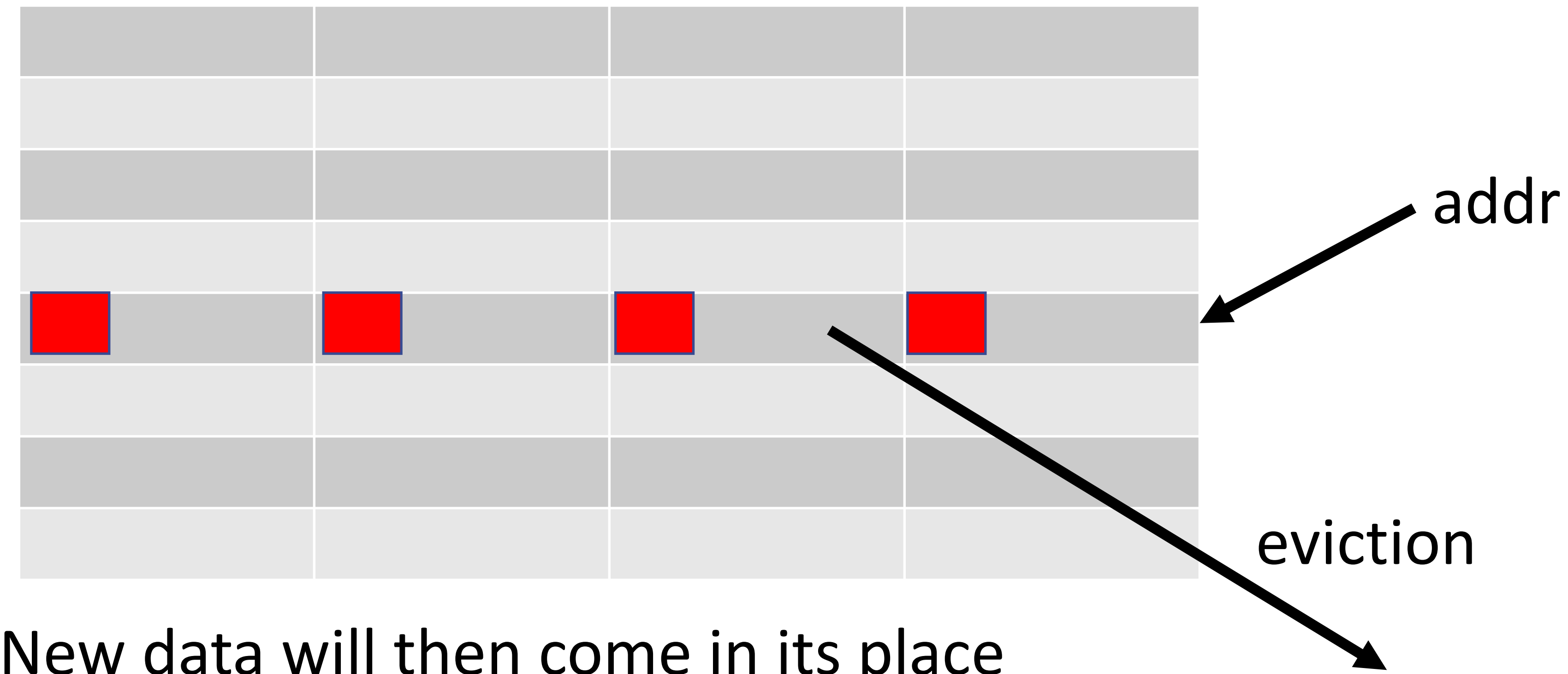


If some way in set is empty, data will go in there

Background: CPU cache

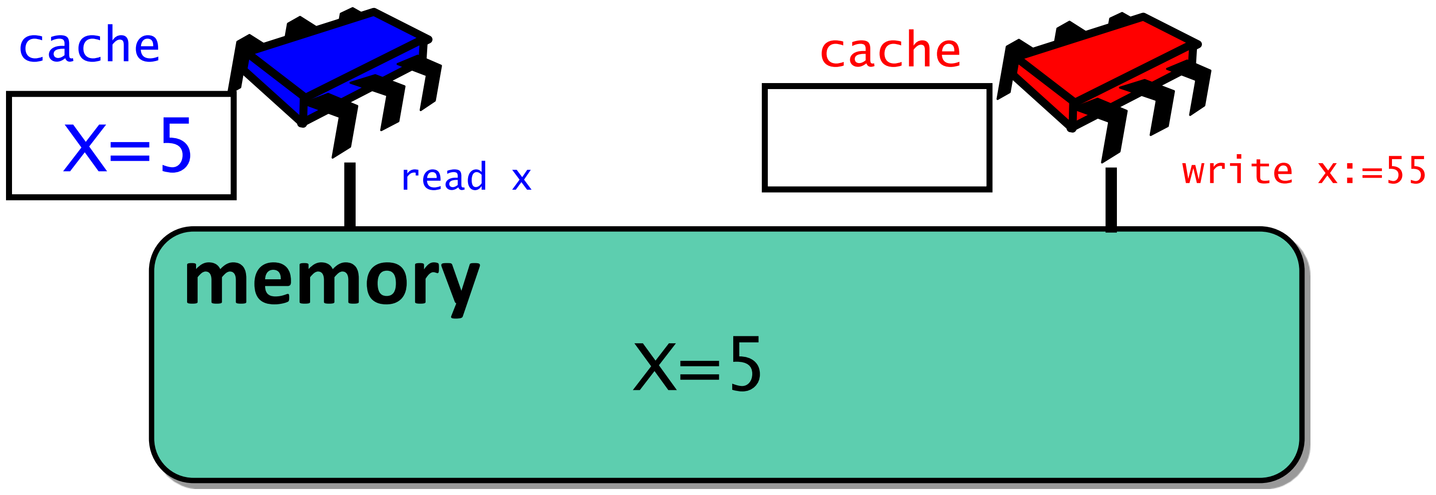
Typically, all ways have data

Replacement policy picks a line and *evicts* it

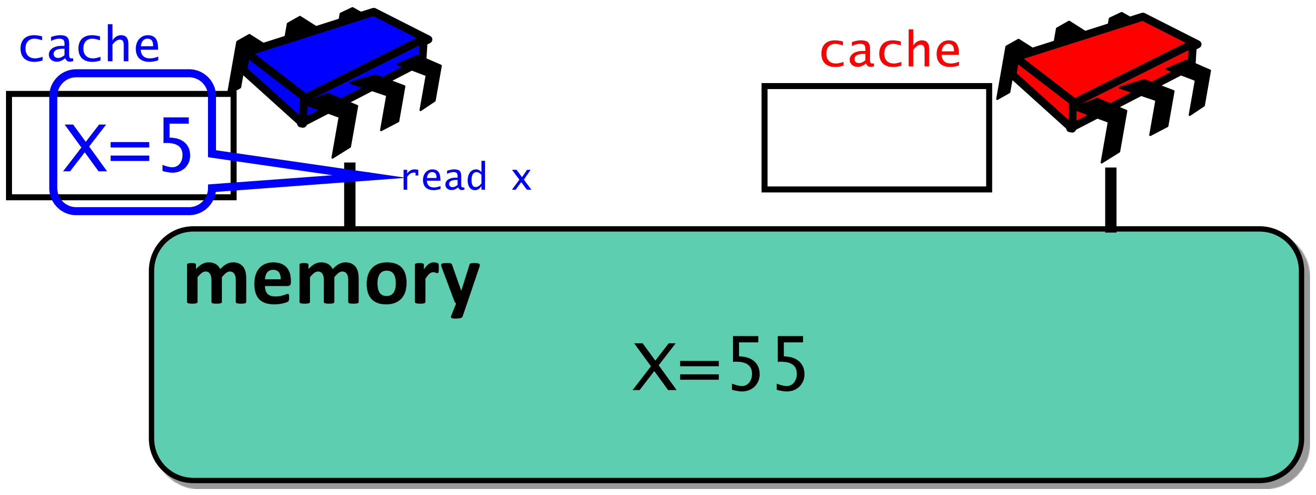


New data will then come in its place

Cache incoherence



Cache incoherence



Cache coherence

Cache coherence ("לכידות", קשר הגיוני ורציף):

- When multiple caches exist, want to prevent access to stale data

Applies to any type of caching system!

Cache (concept):

- Component storing data so that future requests for data can be served faster

Coherence abstraction

- In a cache coherent system, *caches are invisible*
- Any execution observed could have also been observed with just memory
- Example take-away: once a write leaves CPU (e.g., store buffer) it is *logically in memory*
  - Don’t really care whether it’s physically in DRAM or in some cache
  - Based on observing results of reads, no thread can distinguish the cases

Defining cache coherence

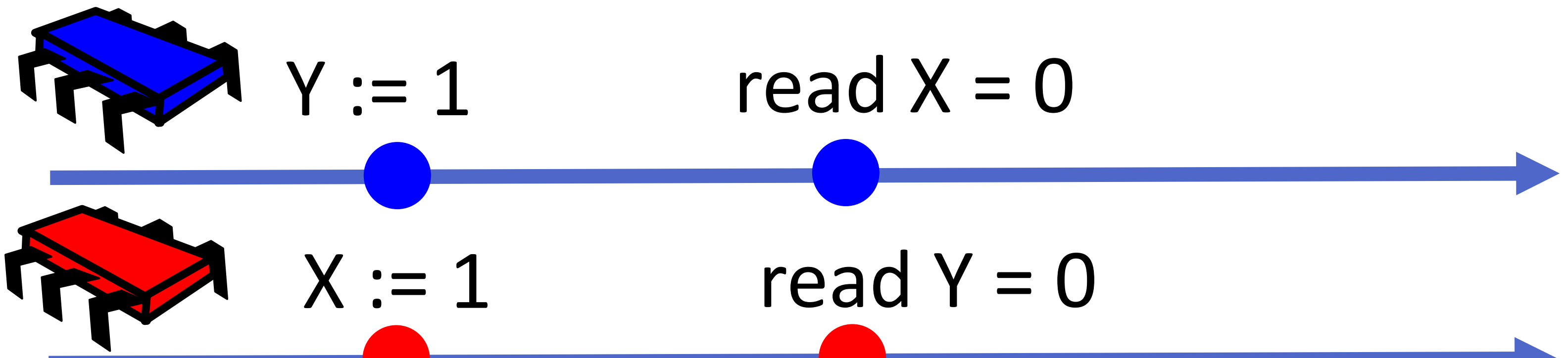
Important: cache coherence is defined per-location

A *consistency model* defines interactions between distinct locations

Defining cache coherence

One option:

- A coherent system must appear to execute all threads' loads and stores to a single memory location in a total order that respects the program order of each thread
- Like SC, but for a *single* location at a time
- Does *not* imply SC

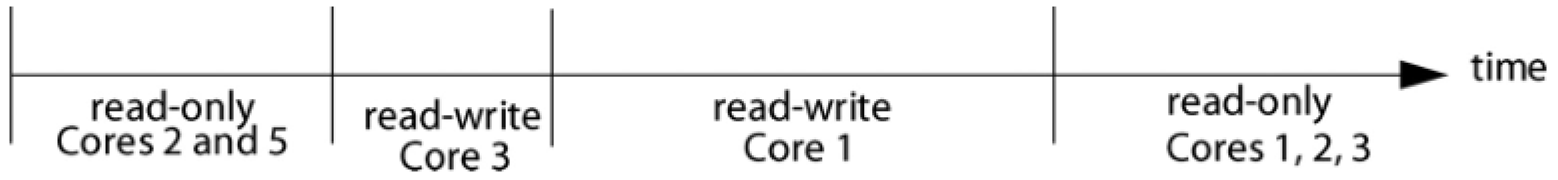


Defining cache coherence

Another option:

Single-writer-multi-reader. For any location, at any time, there is either a single core that may write it (and read it), or some number of cores that may read it

Equiv: memory location lifetime consists of *read/write epochs*



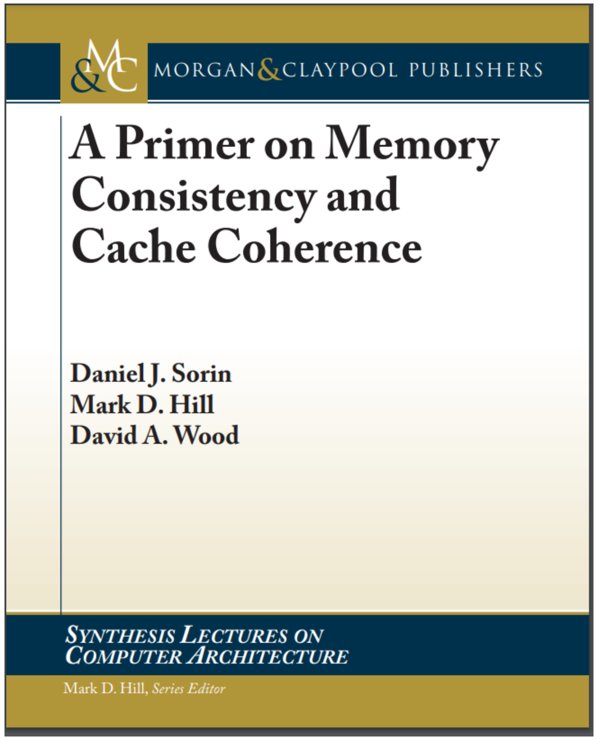
Value read at read epoch *e* is the value written in the prior write epoch

Lecture goals

- Understand multi-core cache coherence
- See details of a distributed algorithm
  - Multi-threaded algorithms have a similar taste

- Material based on book

(see link on moodle)



MORGAN & CLAYPOOL PUBLISHERS

A Primer on Memory Consistency and Cache Coherence

Daniel J. Sorin
Mark D. Hill
David A. Wood

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE

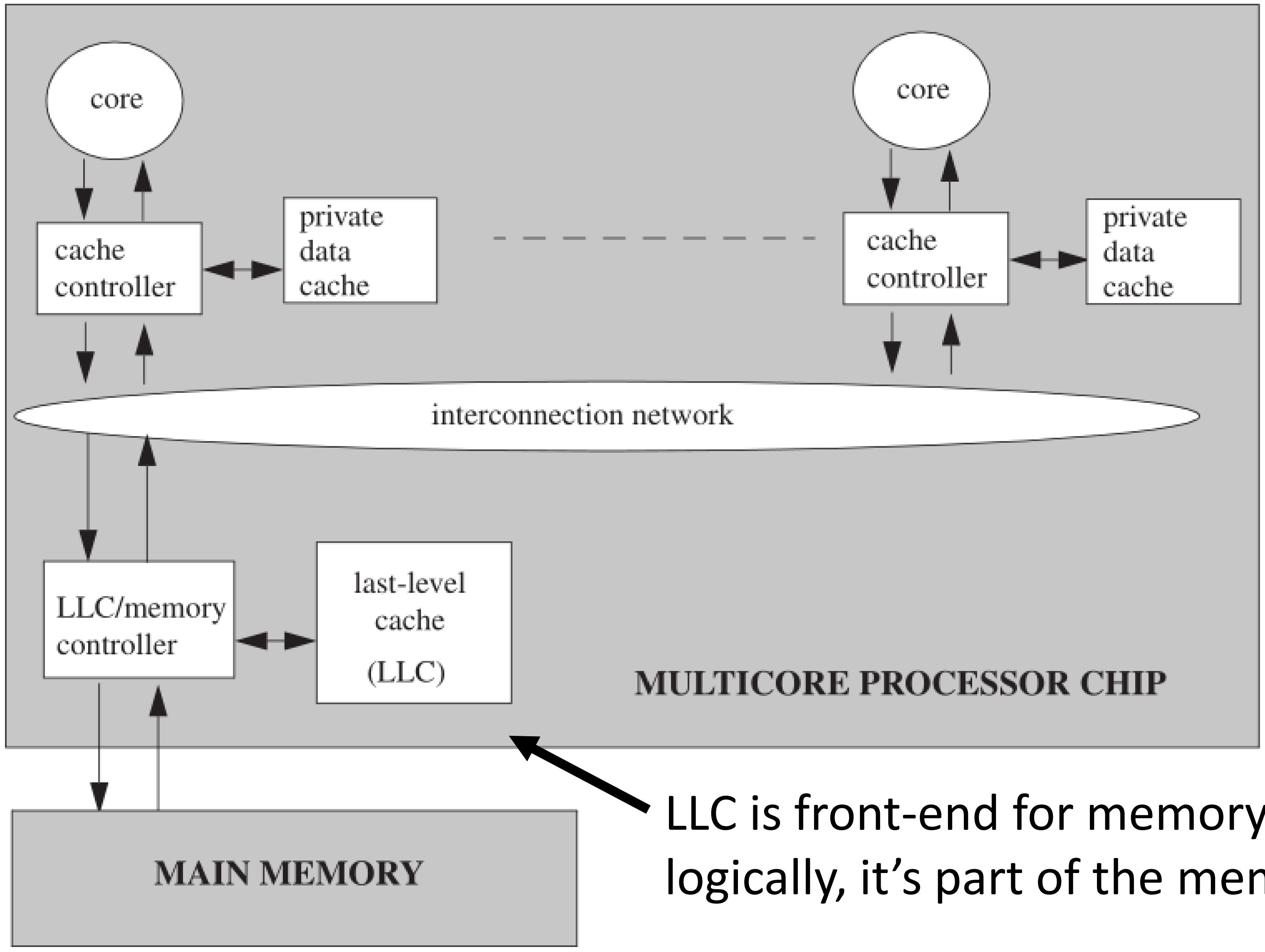
Mark D. Hill, *Series Editor*

Coherence protocols

Coherence protocol

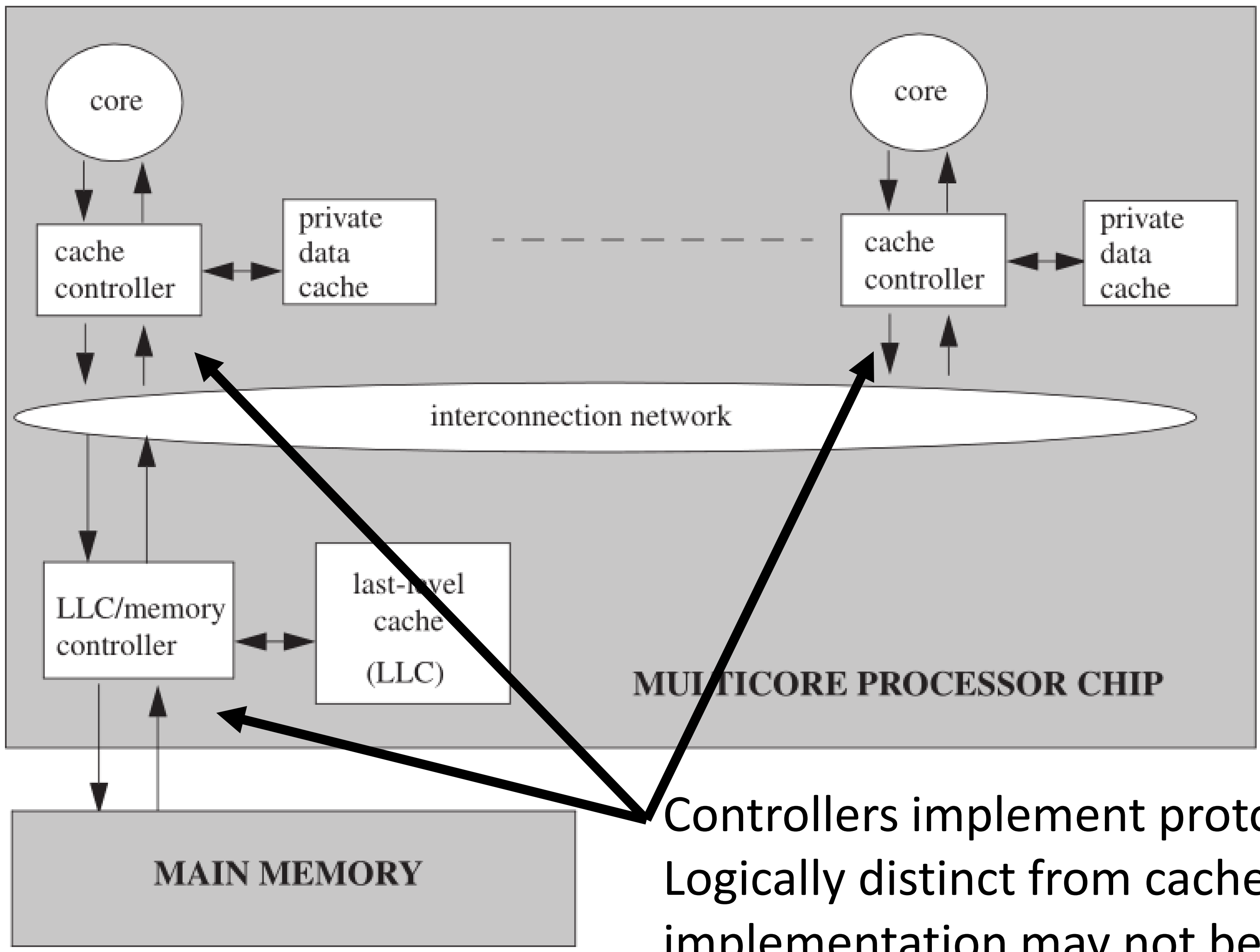
A set of rules implemented by the distributed set of caches in the system; if followed, provides cache coherence

System model



LLC is front-end for memory; logically, it’s part of the memory

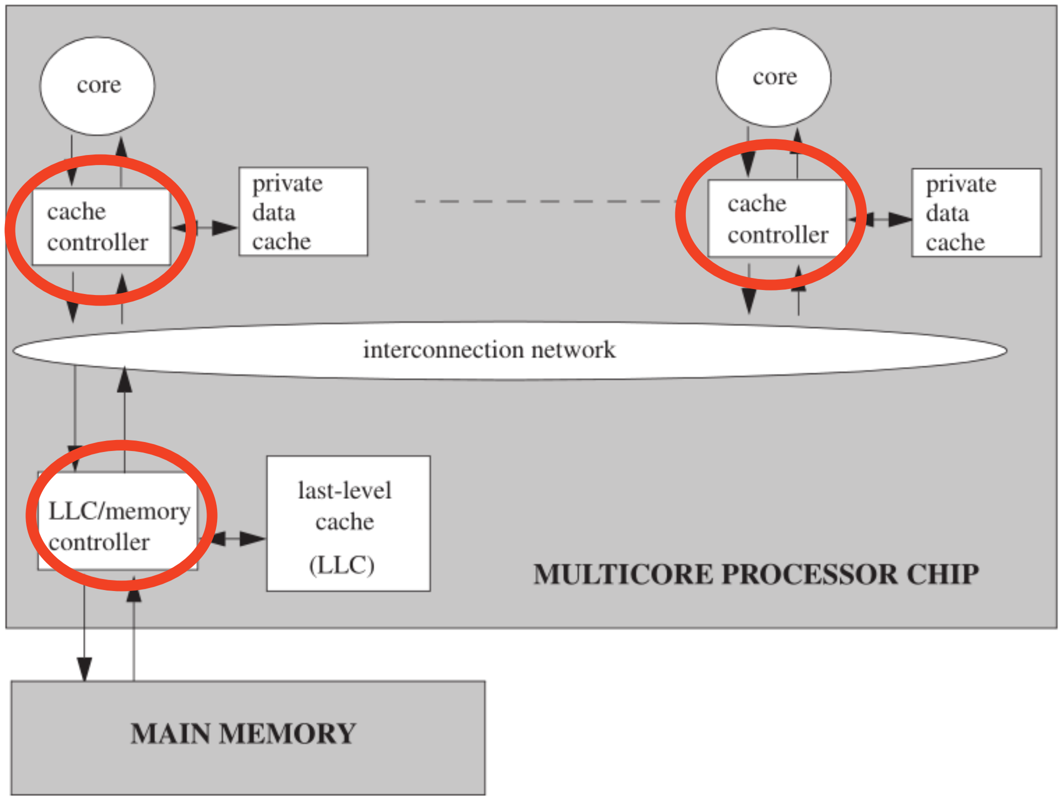
System model



Controllers implement protocol
Logically distinct from cache, but implementation may not be

Coherence protocol players

- Cache controllers
- Memory controllers



Cache controller

- Accepts loads/stores from core; returns loaded value to core
  - Cache miss issues a *coherence request* for the line
  - The request and all messages involved in satisfying it are called a *coherence transaction*
- Receives coherence requests and responses on the network

Memory controller

- Only has network connection
- Receives coherence requests on the network
    - Doesn’t issue coherence requests
- We assume inclusive LLC
    - If line in private cache, LLC will also have state for it

Coherence protocol players

Cache controllers, memory controllers

- Implemented as state machine:
  given state *S* and event *E*, take *actions* and
  change state to *S'*

- Notice: state is *per-line*
  (i.e., multiple state machines running concurrently)

Logical attributes of a line

Possible attributes of a line with respect to cache/LLC:

- *Valid* (cache has up to date value)
- *Exclusive* (cache has only privately cached line, but data may also be cached at the LLC)
- *Owned* (cache responsible for responding to coherence request; cannot be evicted without passing ownership)
- *Dirty* (up to date but different from value in memory/LLC; cache is responsible for updating them eventually)

Notice: These attributes are **not** mutually exclusive

MSI protocol

Possible cache controller states:

- M(odified)
- S(hared)
- I(invalid)

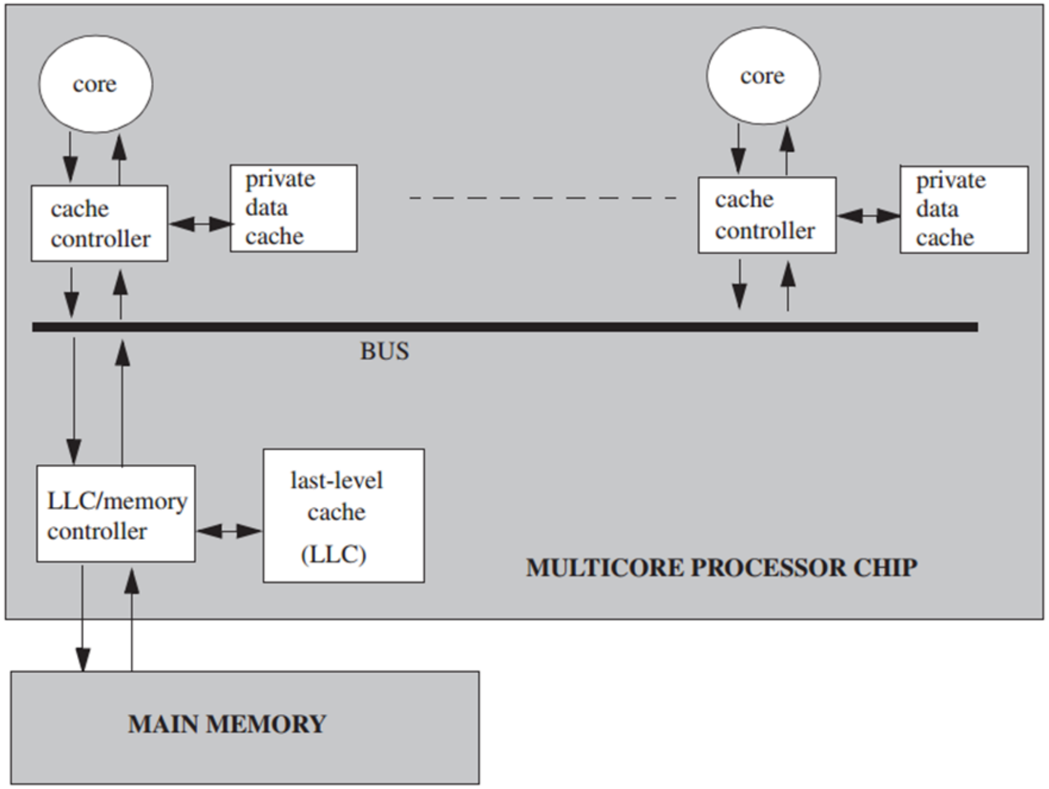
MSI protocol

Cache controller state:

- M(odified): valid, exclusive, owned, maybe dirty
- S(hared): valid, not exclusive, not dirty, not owned
    - Cache has read-only copy
    - LLC responsible for answering coherence requests
- I(nvalid): not valid
    - Cache doesn’t have line or has a stale version, which it may not read or write

MSI snooping protocol

All controllers observe (*snoop*) coherence requests in the same order and can thus maintain coherence



MSI snooping protocol

- Coherence requests broadcast on *totally ordered* shared bus; observed in same order by all controllers (including sender)
- Controller sends and listens until seeing its request on the bus: may observe other requests in the meantime
- Bus acts as a serialization point
  - **Coherence transactions logically occurs when the request is serialized**
  - Responses don’t need to go over shared bus

Shared bus analogy

Classroom with teacher	
Raise hand:	*Local event (send)*
Permission to speak:	*Ordered on bus*

- Total order over all statements
- You hear other statements while waiting for your turn

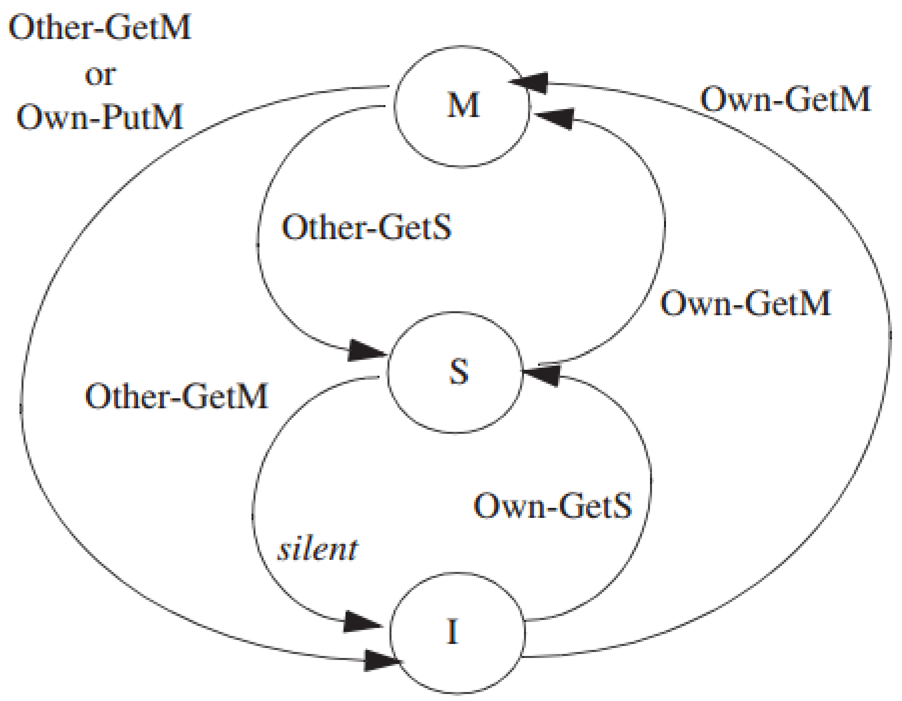
- Everyone sees same events, so all agree about the state of the system and how it evolves

Simple protocol

Simplifying assumptions on system:

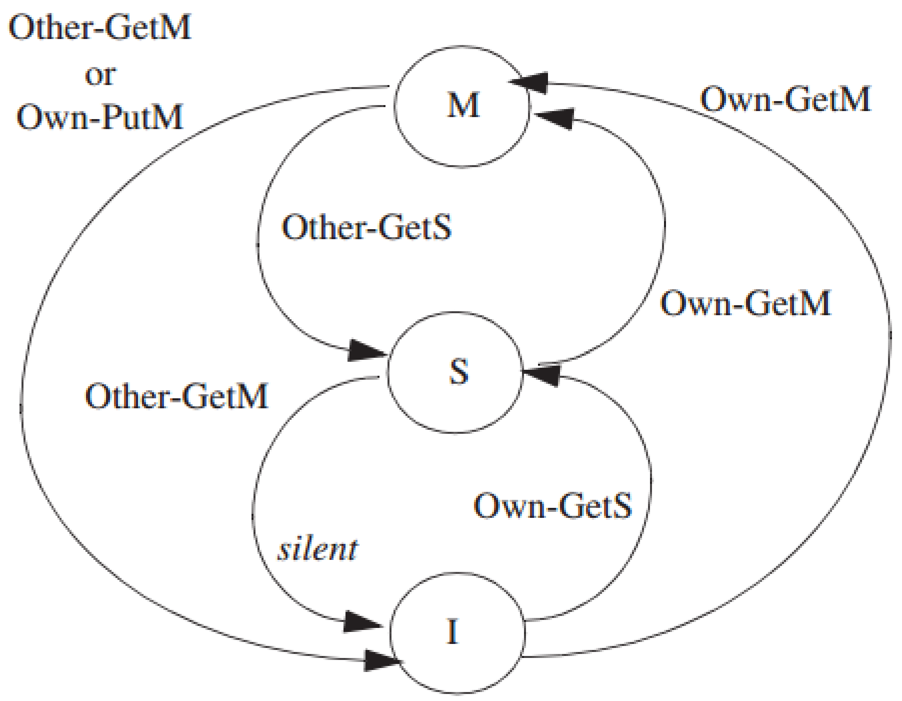
- *Atomic requests*: request ordered immediately
  - "Permission to speak immediately upon raising hand"
- *Atomic transactions*: a subsequent request for the same line cannot appear on the bus until the current transaction completes

MSI snooping protocol

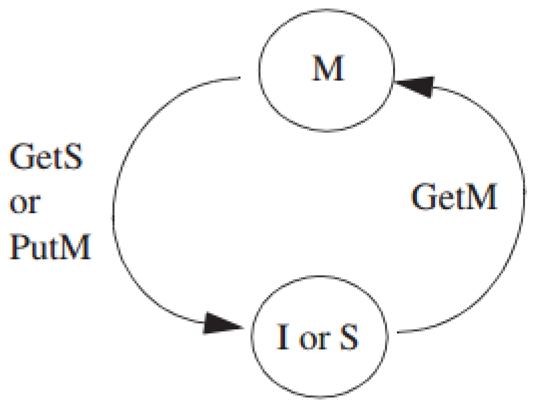


cache controller

MSI snooping protocol



cache controller



memory controller

(state names reflect status of line in caches)

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I										

can happen any time

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS / ?									

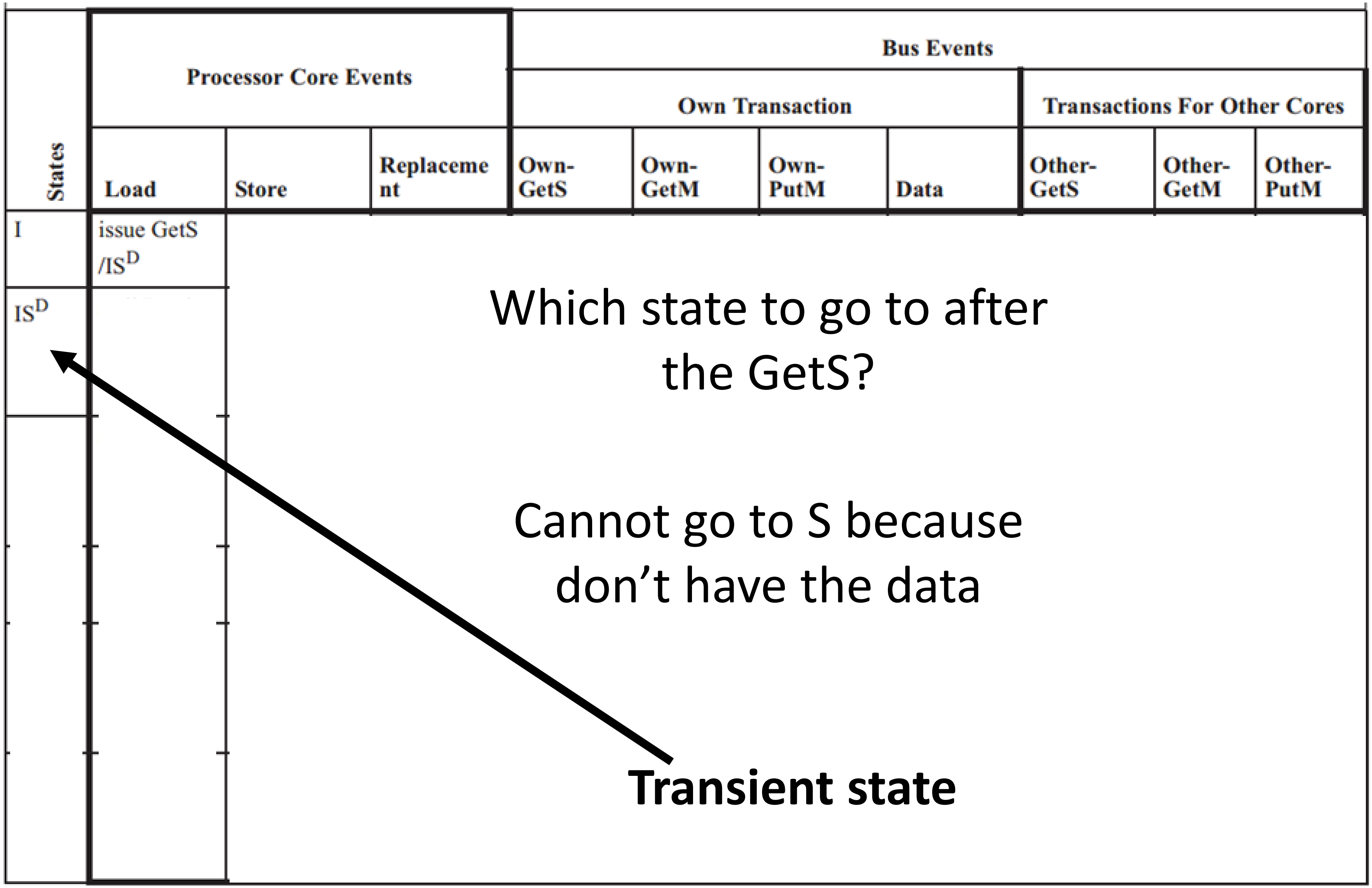
Which state to go to after the GetS?

$$\begin{tikzcd}
M \arrow[d, bend right, "\text{Other-GetS}"'] \arrow[dd, bend right=80, "\text{Other-GetM or Own-PutM}"'] \\
S \arrow[u, bend right, "\text{Own-GetM}"'] \arrow[d, bend right, "\text{Other-GetM} \quad \textit{silent}"'] \\
I \arrow[u, bend right, "\text{Own-GetS}"'] \arrow[uu, bend right=80, "\text{Own-GetM}"']
\end{tikzcd}$$

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS / ?									

Which state to go to after the GetS?

Cannot go to S because don’t have the data



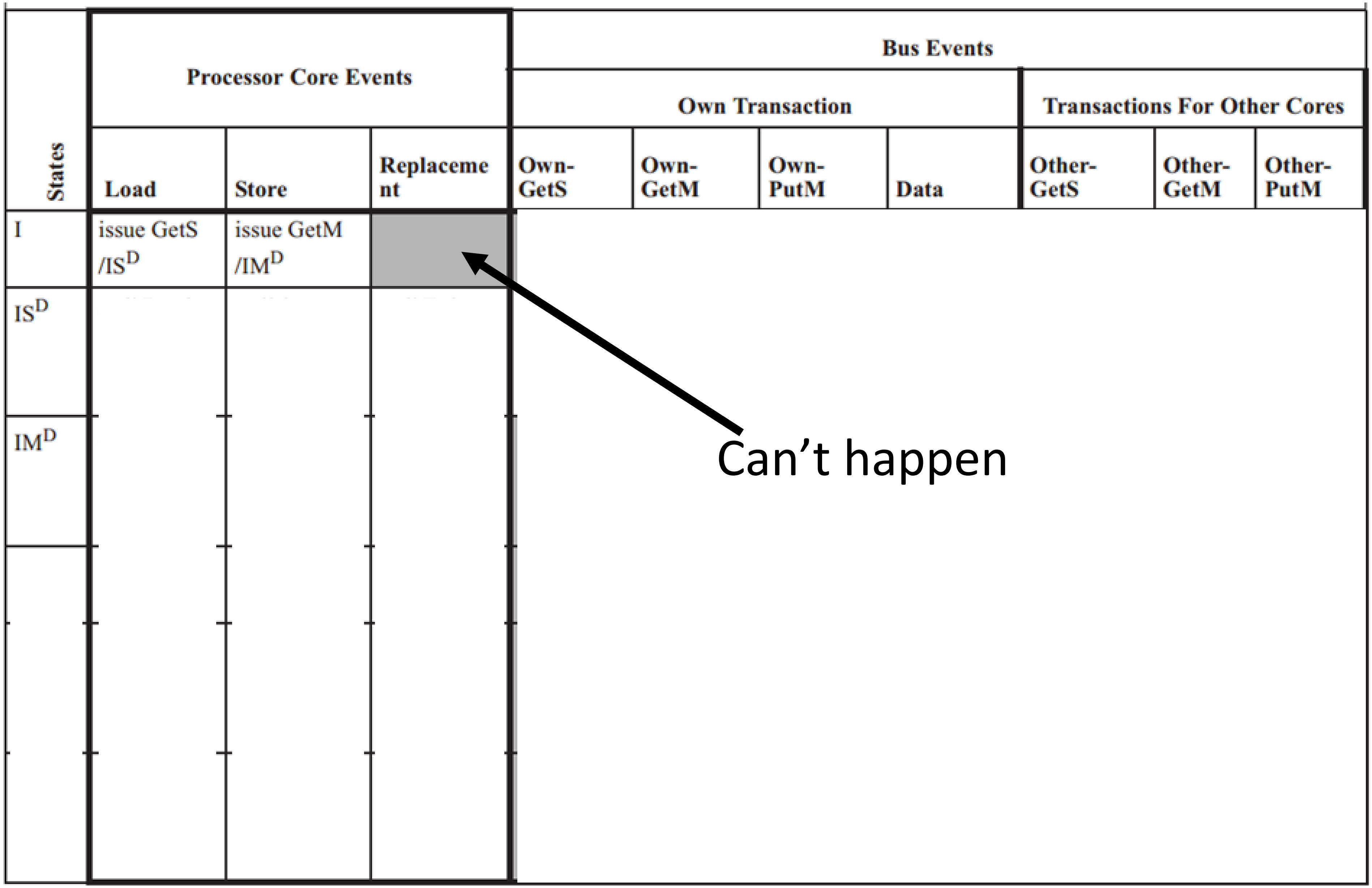
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D									
IS^D										

Which state to go to after the GetS?

Cannot go to S because don’t have the data

Transient state

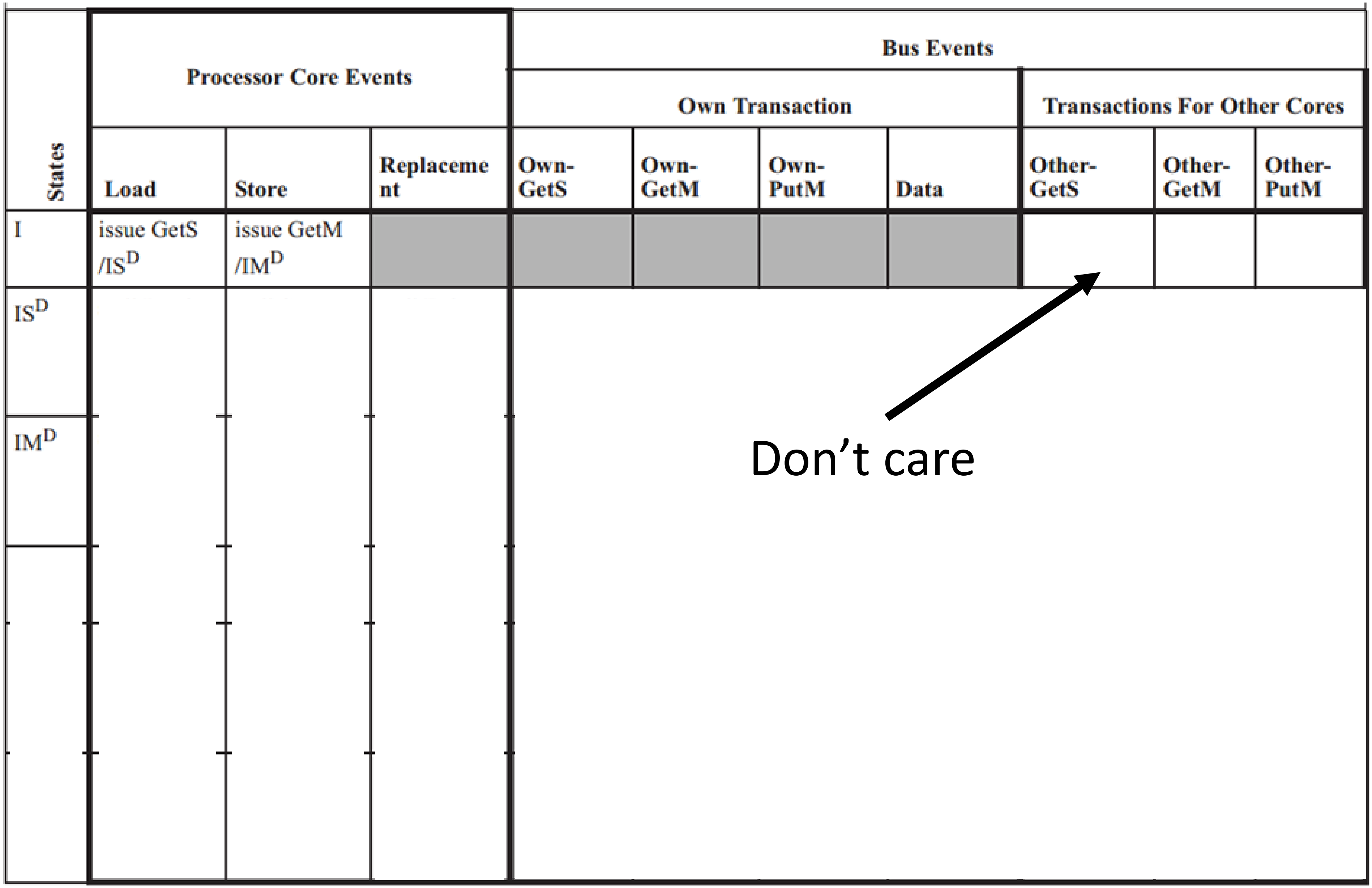
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D								
IS^D										
IM^D										



States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨							
IS^D										
IM^D										

Can’t happen

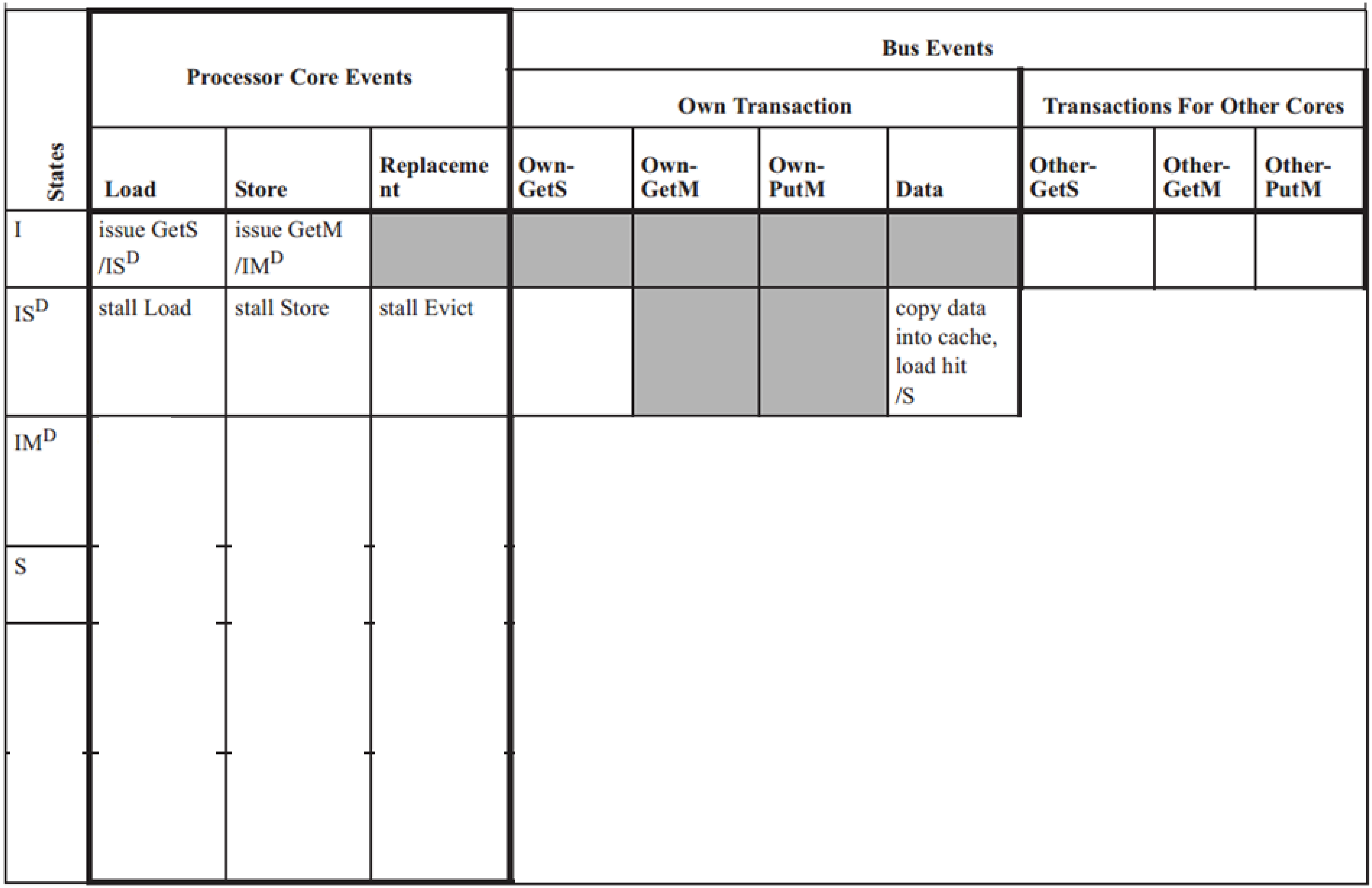
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	**Store**	**Replacement**	**Own-GetS**	**Own-GetM**	**Own-PutM**	**Data**	**Other-GetS**	**Other-GetM**	**Other-PutM**
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D										
IM^D										



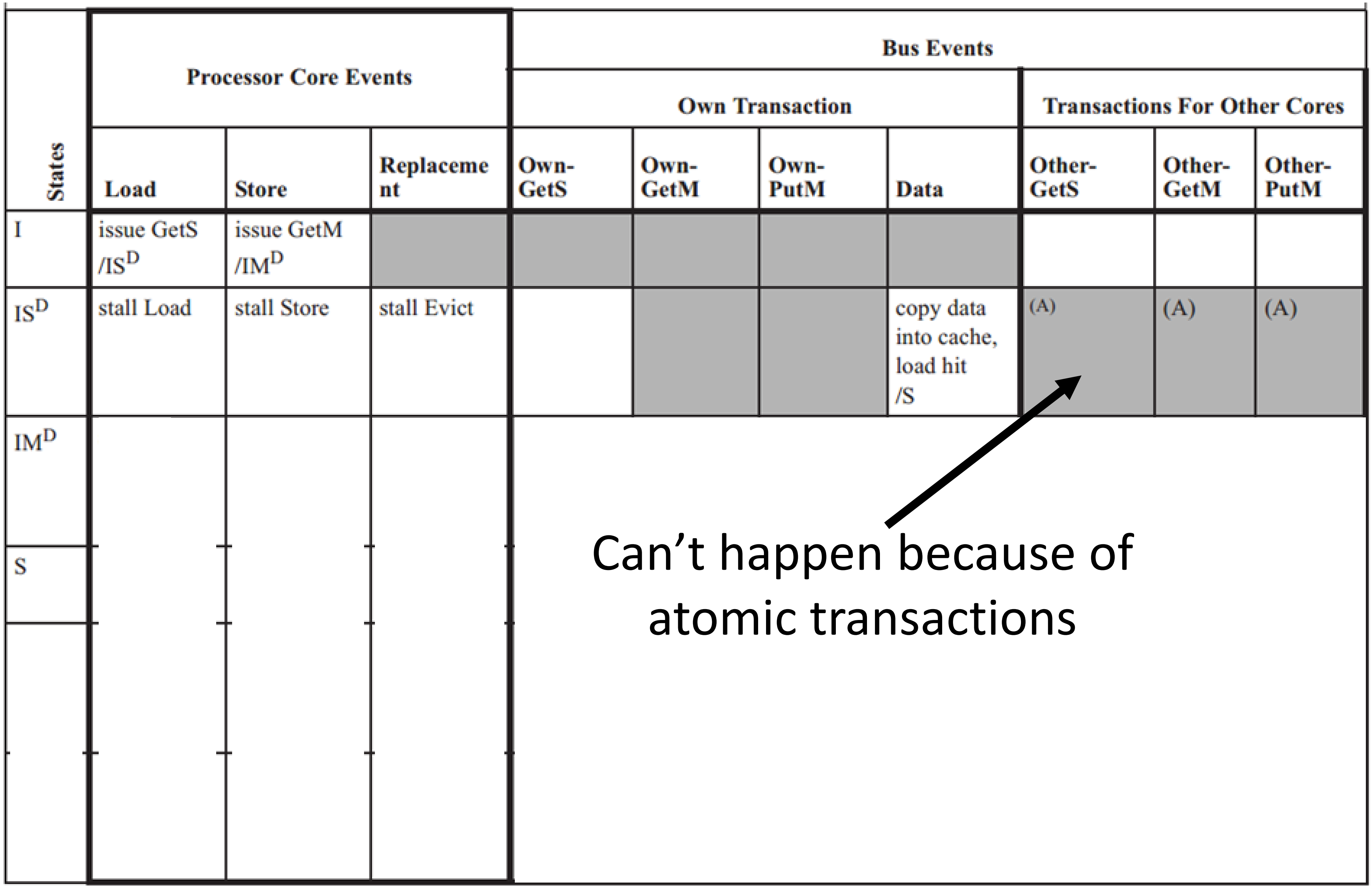
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D										
IM^D										

Don't care

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict							
IM^D										



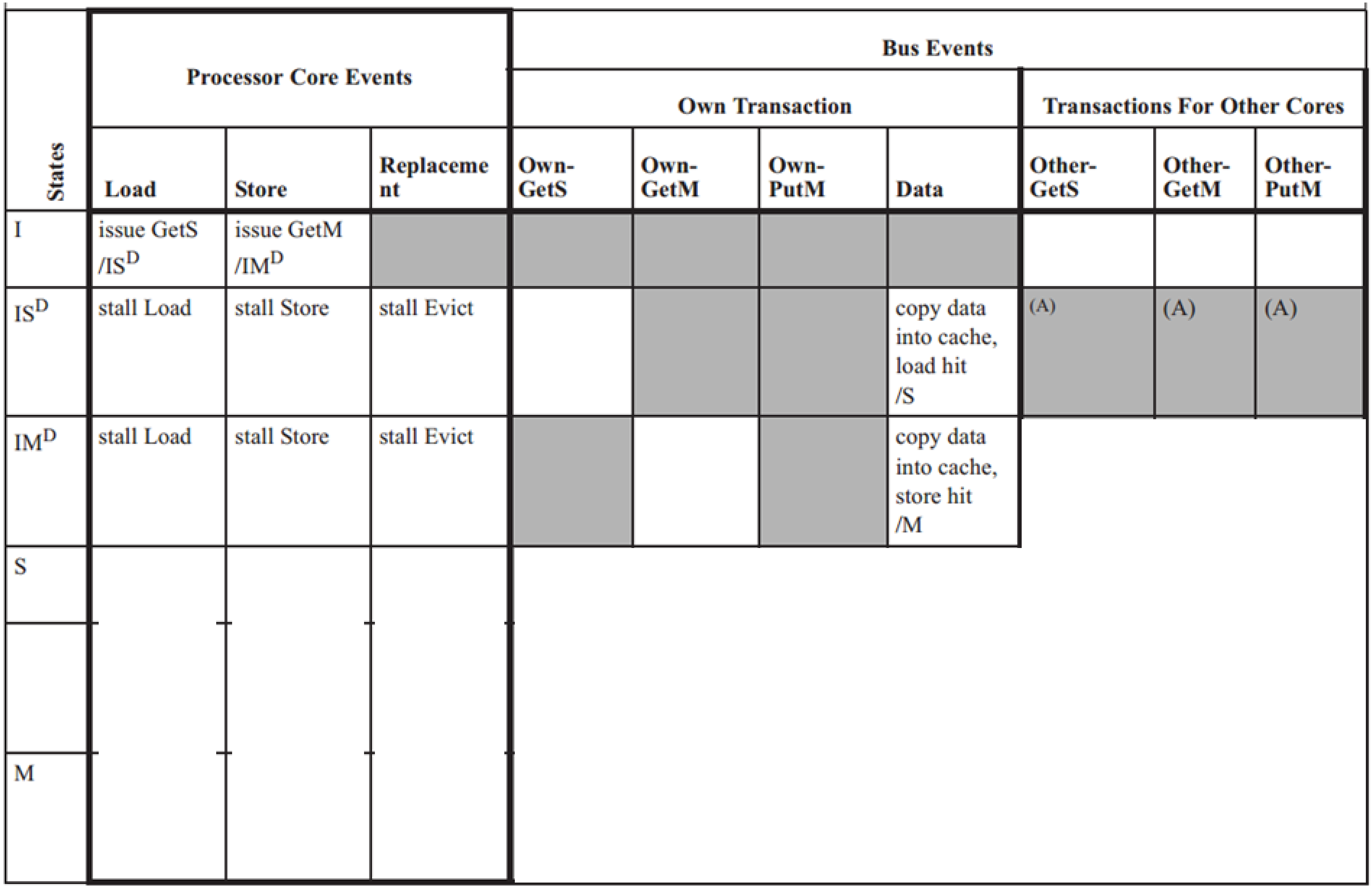
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S			
IM^D										
S										



States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D										
S										

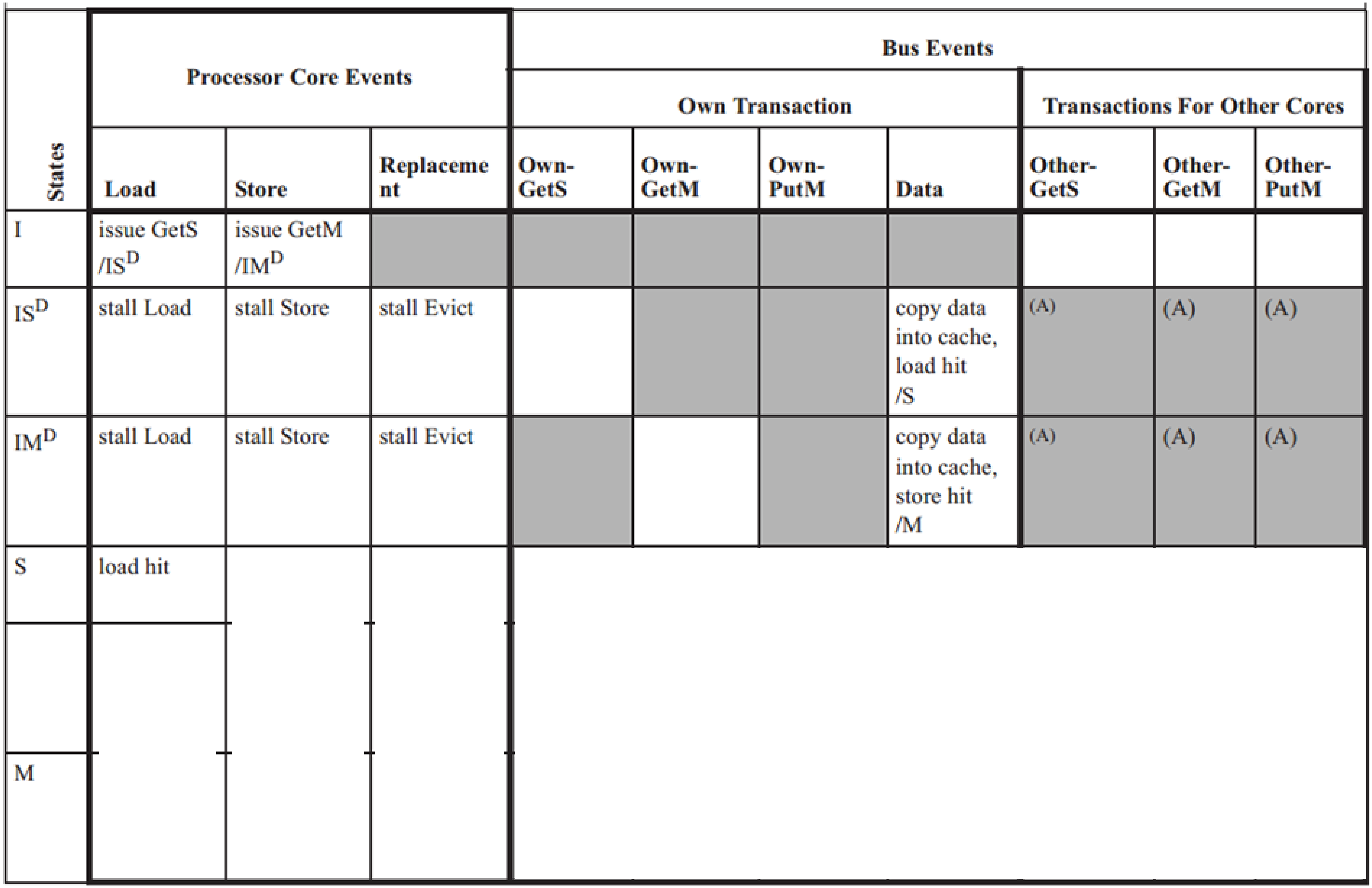
Can’t happen because of atomic transactions

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	**Store**	**Replacement**	**Own-GetS**	**Own-GetM**	**Own-PutM**	**Data**	**Other-GetS**	**Other-GetM**	**Other-PutM**
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	▨ (A)	▨ (A)	▨ (A)
IM^D	stall Load	stall Store	stall Evict							
S										

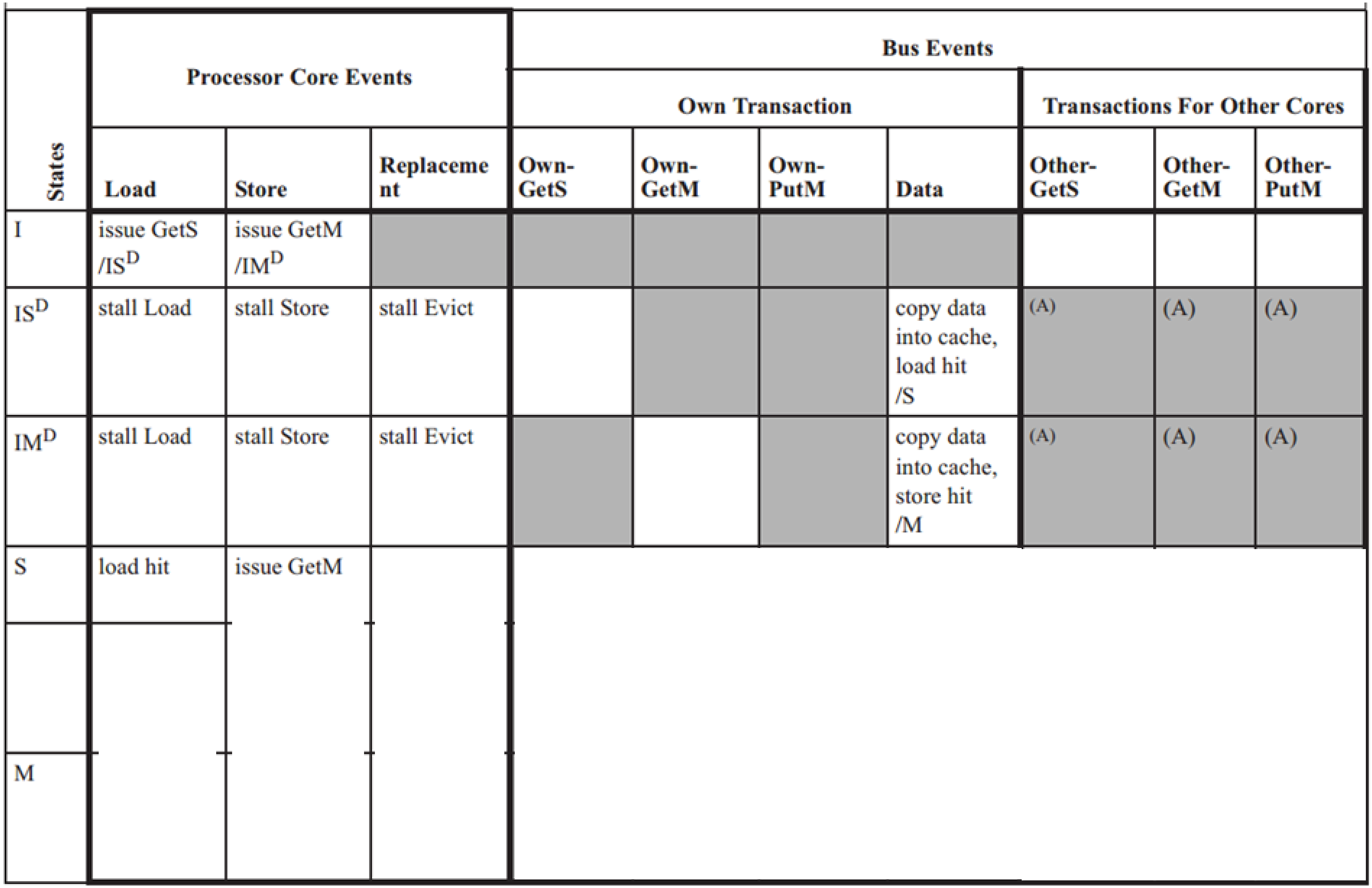


States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	▨ (A)	▨ (A)	▨ (A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M			
S										
M										

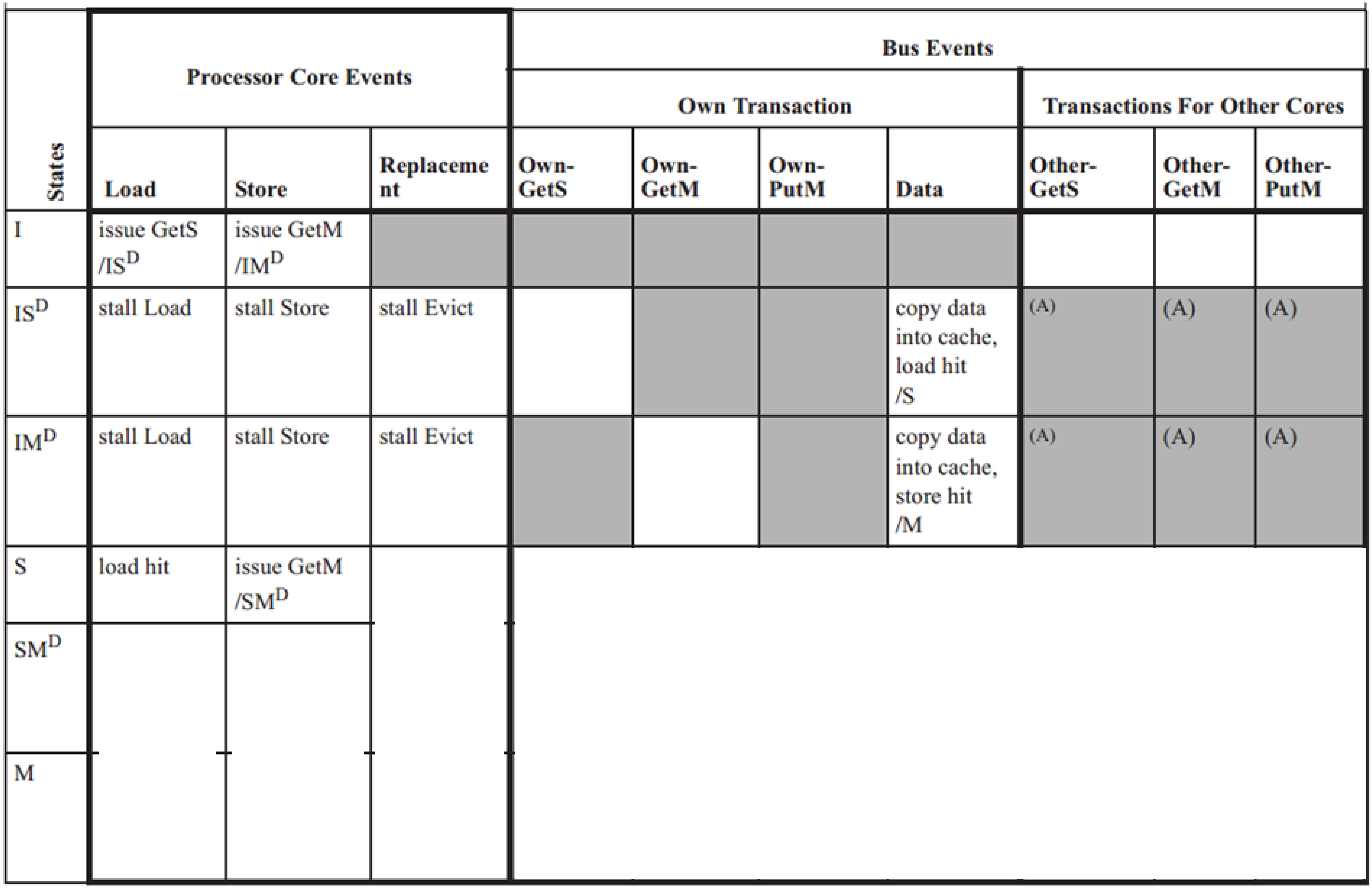
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	▨ (A)	▨ (A)	▨ (A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	▨ (A)	▨ (A)	▨ (A)
S										
M										



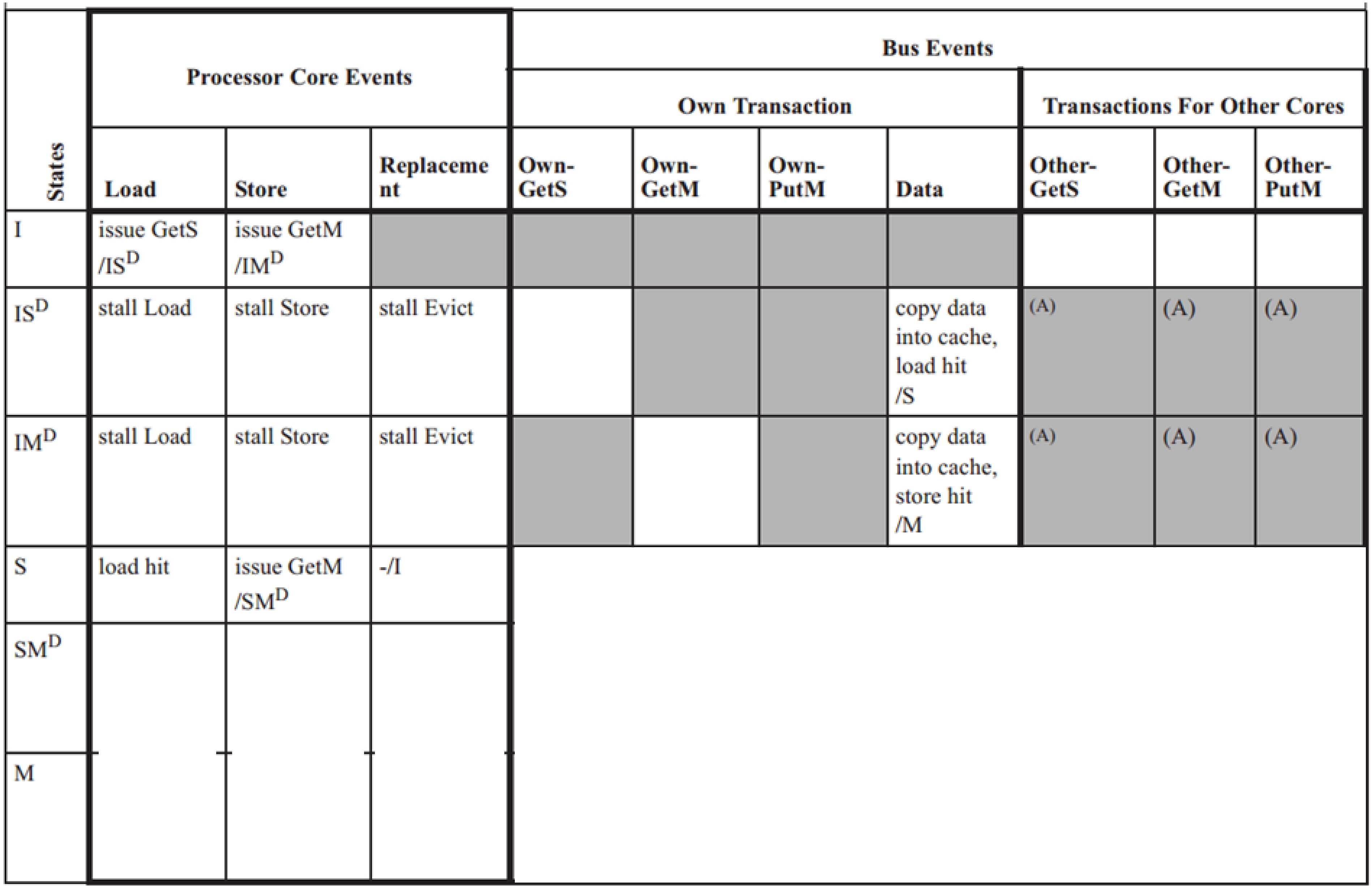
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	▨ (A)	▨ (A)	▨ (A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	▨ (A)	▨ (A)	▨ (A)
S	load hit									
M										



States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM								
M										



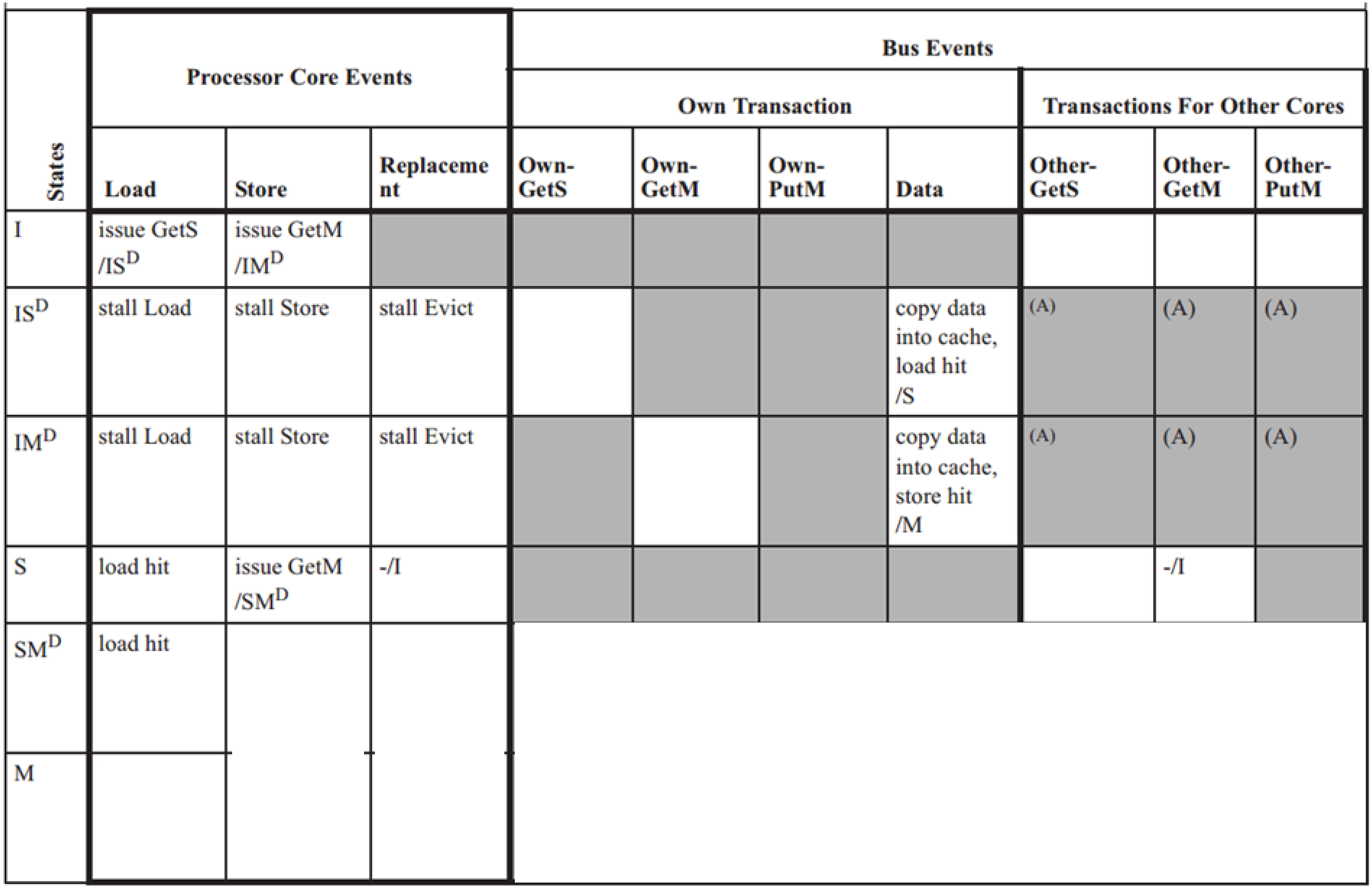
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	▨ (A)	▨ (A)	▨ (A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	▨ (A)	▨ (A)	▨ (A)
S	load hit	issue GetM /SM^D								
SM^D										
M										



States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I							
SM^D										
M										

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨			
SM^D										
M										

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D										
M										



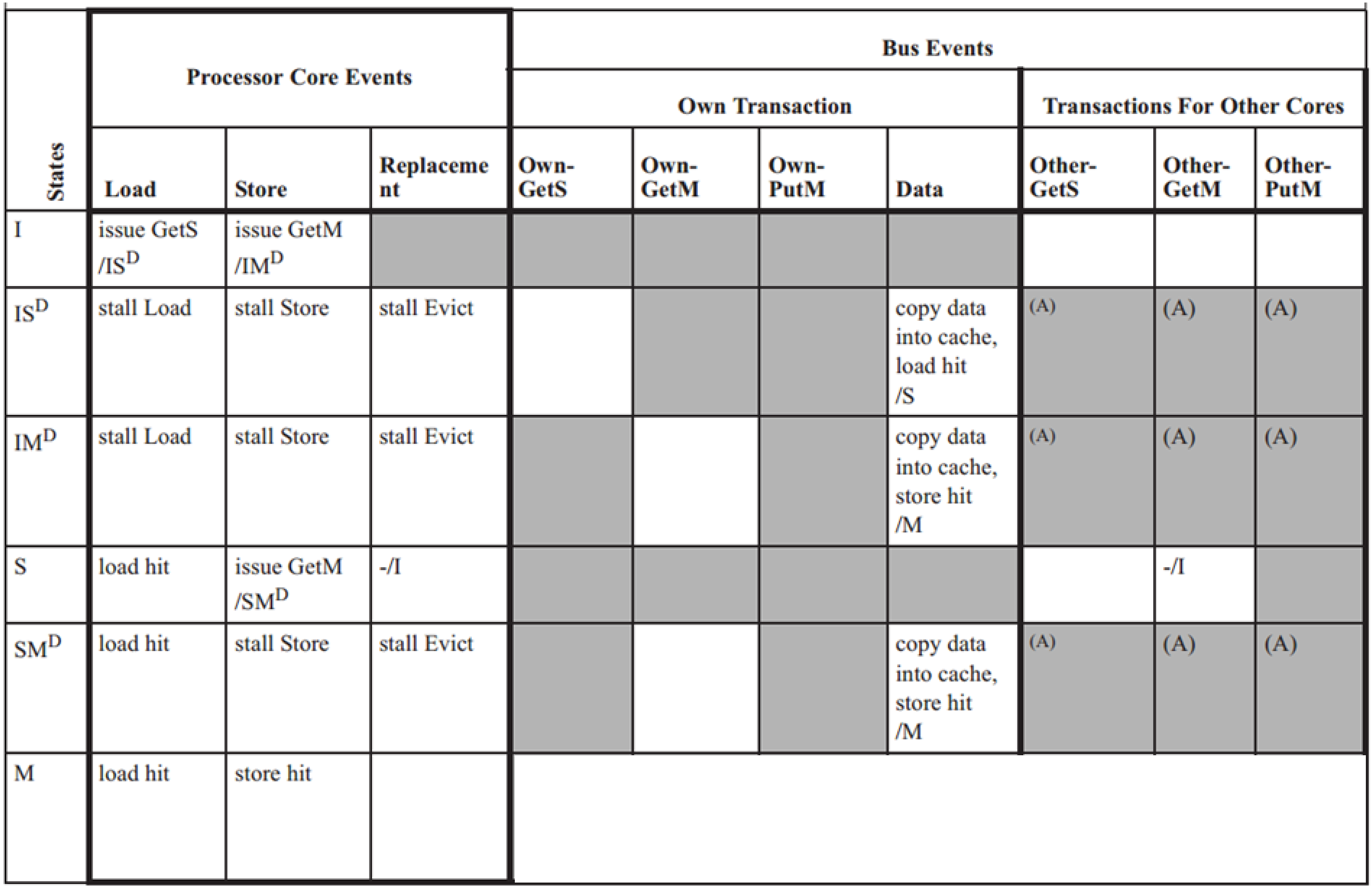
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	**Store**	**Replaceme nt**	**Own-GetS**	**Own-GetM**	**Own-PutM**	**Data**	**Other-GetS**	**Other-GetM**	**Other-PutM**
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	▨ (A)	▨ (A)	▨ (A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	▨ (A)	▨ (A)	▨ (A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit									
M										

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit	stall Store	stall Evict							
M										

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M			
M										

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
M										

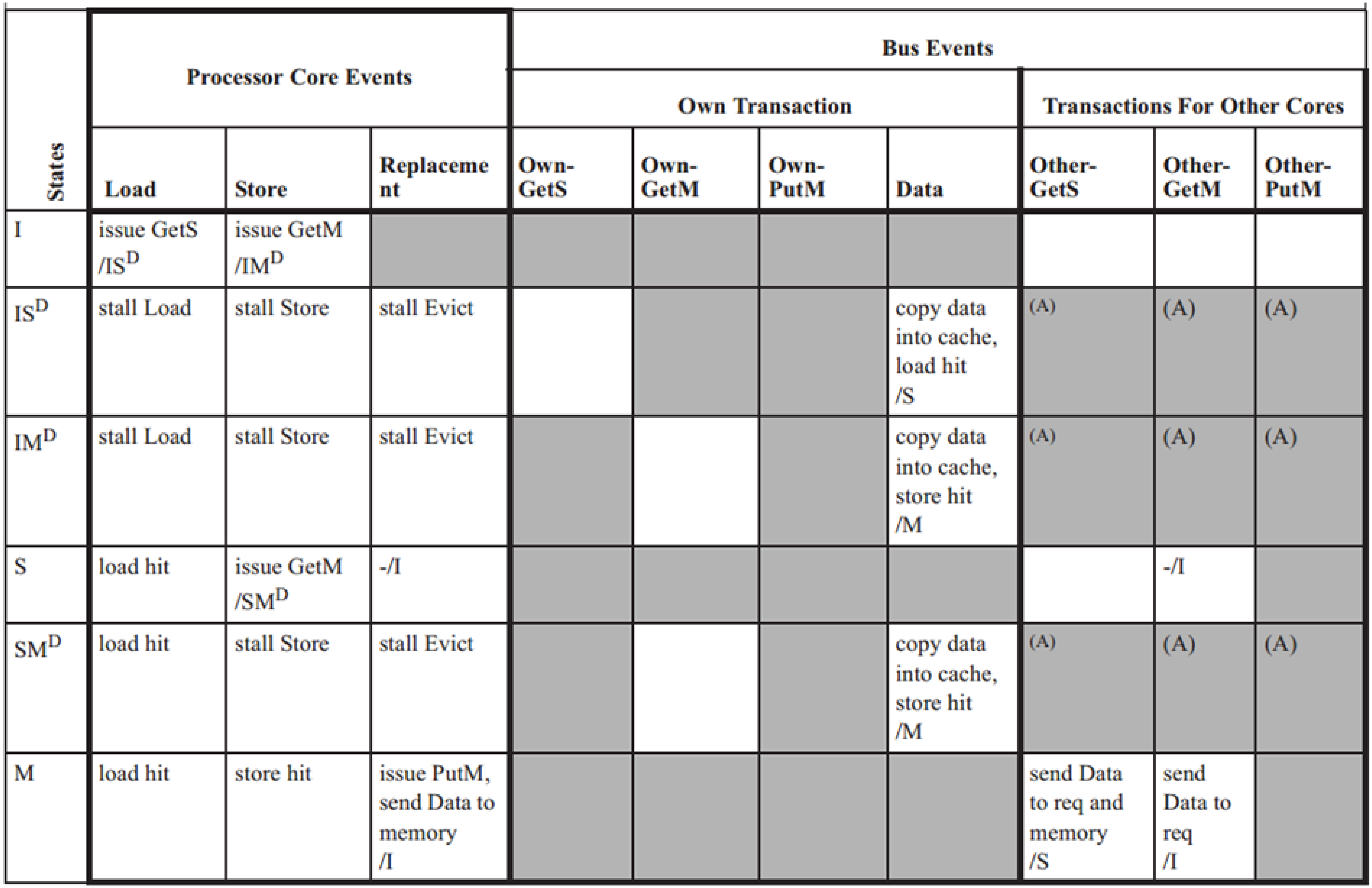
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	▨ (A)	▨ (A)	▨ (A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	▨ (A)	▨ (A)	▨ (A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	▨ (A)	▨ (A)	▨ (A)
M	load hit									



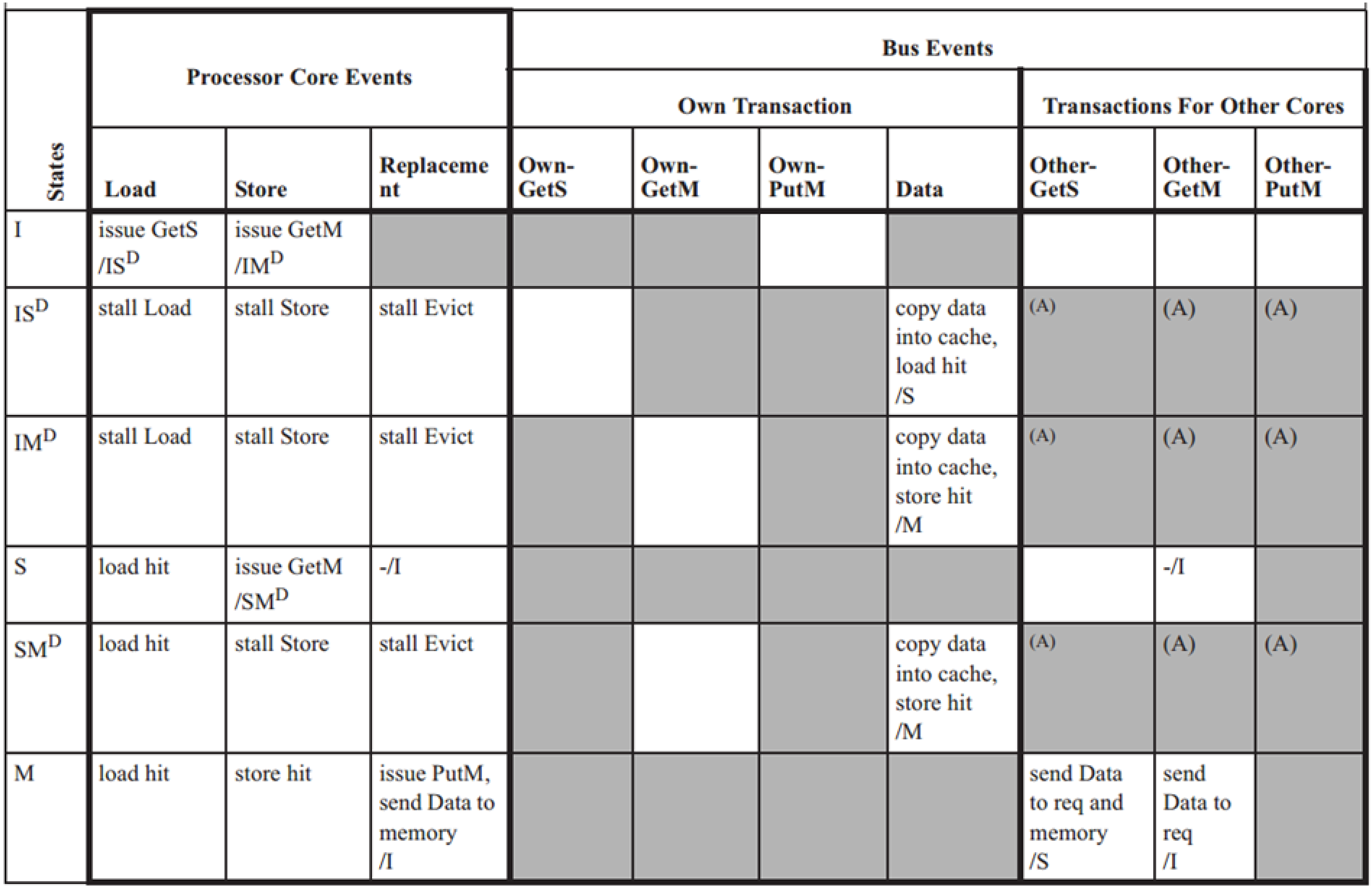
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
M	load hit	store hit								

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
M	load hit	store hit	issue PutM, send Data to memory /I							

States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replacement	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
M	load hit	store hit	issue PutM, send Data to memory /I	▨	▨	▨	▨			



States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	**Store**	**Replaceme nt**	**Own-GetS**	**Own-GetM**	**Own-PutM**	**Data**	**Other-GetS**	**Other-GetM**	**Other-PutM**
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨	▨	▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
M	load hit	store hit	issue PutM, send Data to memory /I	▨	▨	▨	▨	send Data to req and memory /S	send Data to req /I	▨



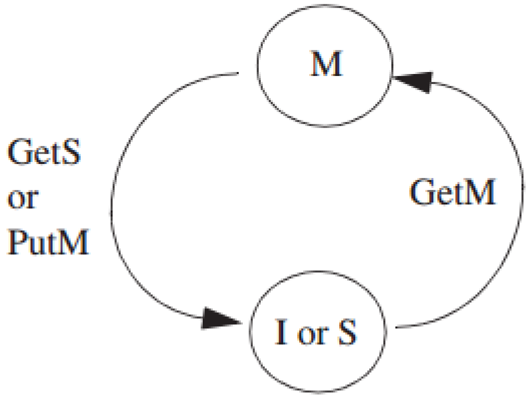
States	Processor Core Events			Bus Events						
				Own Transaction				Transactions For Other Cores		
	Load	Store	Replaceme nt	Own-GetS	Own-GetM	Own-PutM	Data	Other-GetS	Other-GetM	Other-PutM
I	issue GetS /IS^D	issue GetM /IM^D	▨	▨	▨		▨			
IS^D	stall Load	stall Store	stall Evict		▨	▨	copy data into cache, load hit /S	(A)	(A)	(A)
IM^D	stall Load	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
S	load hit	issue GetM /SM^D	-/I	▨	▨	▨	▨		-/I	▨
SM^D	load hit	stall Store	stall Evict	▨		▨	copy data into cache, store hit /M	(A)	(A)	(A)
M	load hit	store hit	issue PutM, send Data to memory /I	▨	▨	▨	▨	send Data to req and memory /S	send Data to req /I	▨

What's wrong with this table?

Memory controller

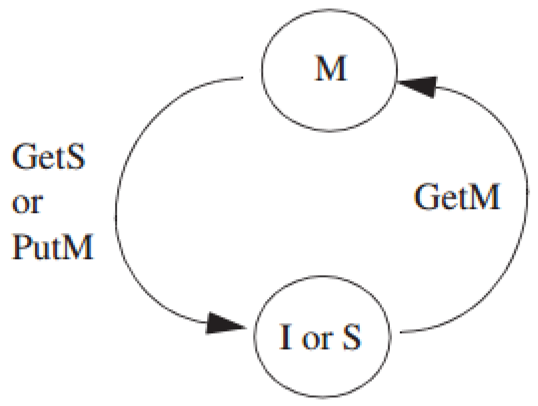
Memory controller

	Bus Events			
State	GetS	GetM	PutM	Data from Owner
IorS				
M				



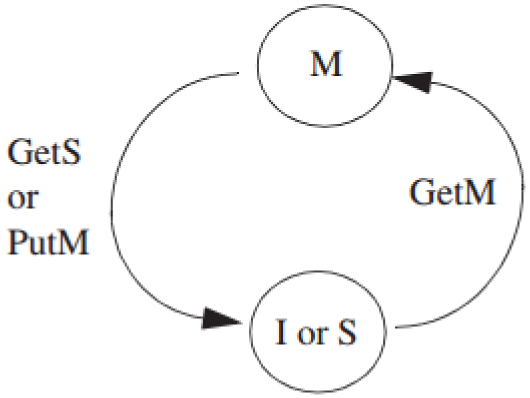
Memory controller

State	Bus Events			
	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS			
M				



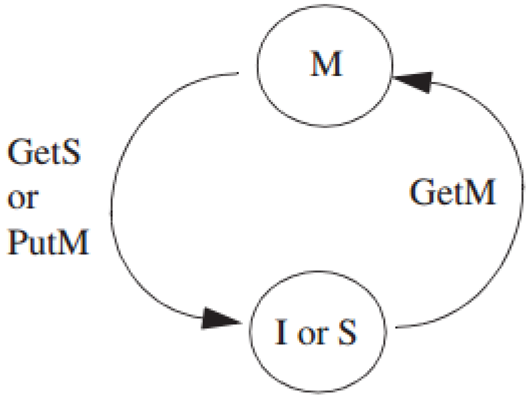
Memory controller

	Bus Events			
State	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS	send data block in Data message to requestor/M		
M				


$$\begin{tikzcd}
M \arrow[d, bend right, "\begin{array}{l}\text{GetS}\\\text{or}\\\text{PutM}\end{array}"'] \\
\text{I or S} \arrow[u, bend right, "\text{GetM}"']
\end{tikzcd}$$

Memory controller

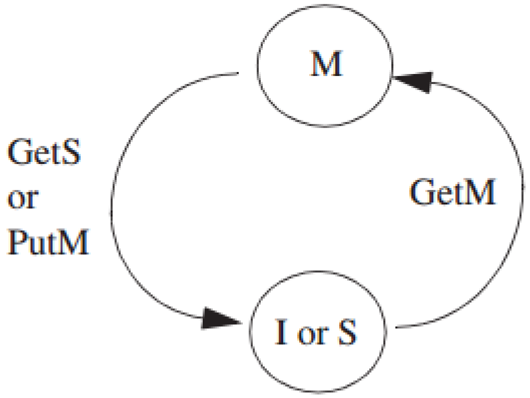
State	Bus Events			
	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS	send data block in Data message to requestor/M	▨	▨
M				



$$\begin{tikzcd}
M \arrow[d, bend right, "GetS\ or\ PutM"'] \\
I\ or\ S \arrow[u, bend right, "GetM"']
\end{tikzcd}$$

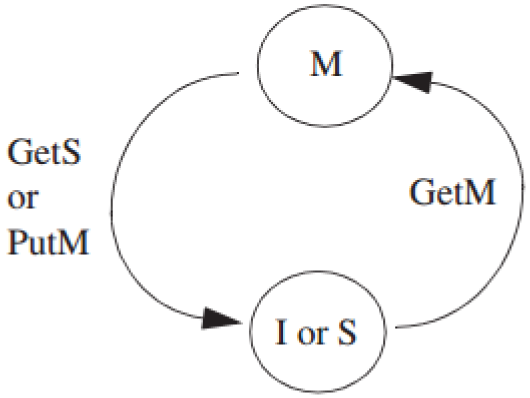
Memory controller

	Bus Events			
State	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS	send data block in Data message to requestor/M	▨	▨
M	-/ ?			



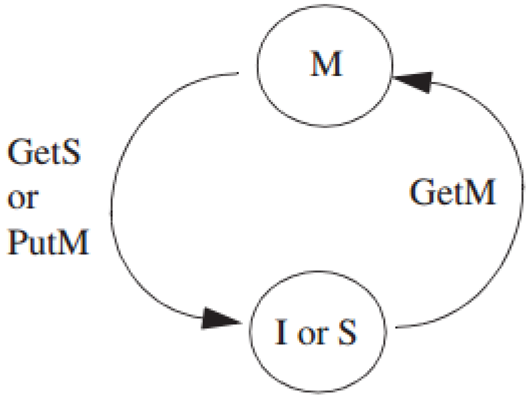
Memory controller

State	Bus Events			
	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS	send data block in Data message to requestor/M	▨	▨
$IorS^D$				
M	$-/IorS^D$			



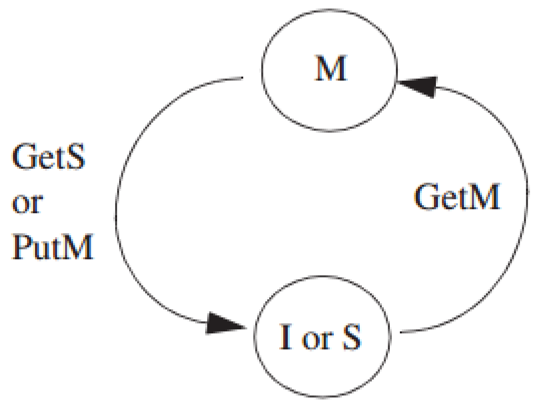
Memory controller

State	Bus Events			
	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS	send data block in Data message to requestor/M	▨	▨
IorS^D	(A)	(A)	▨	
M	-/IorS^D			



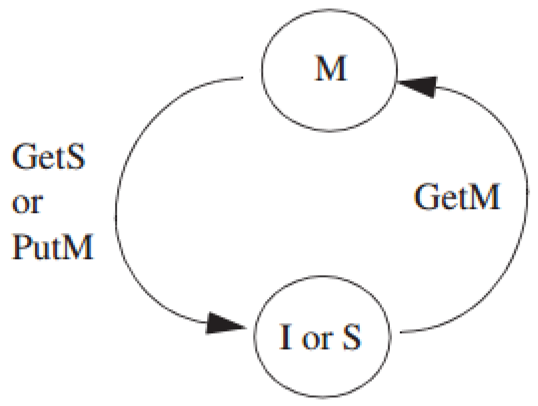
Memory controller

State	Bus Events			
	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS	send data block in Data message to requestor/M	▨	▨
IorS^{D}	(A)	(A)	▨	update data block in memory/IorS
M	-/IorS^{D}			


$$\begin{tikzcd}
M \arrow[d, bend right=60, "\text{GetS or PutM}"'] \\
\text{I or S} \arrow[u, bend right=60, "\text{GetM}"']
\end{tikzcd}$$

Memory controller

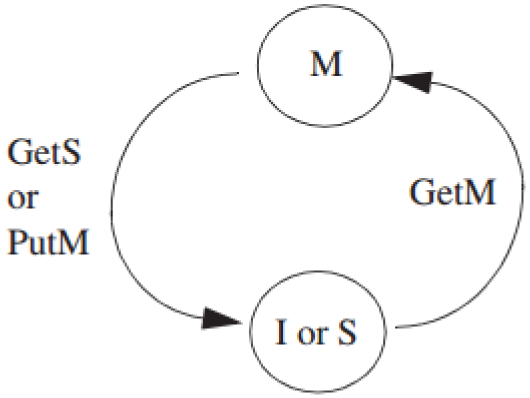
State	Bus Events			
	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS	send data block in Data message to requestor/M	▨	▨
$IorS^D$	(A)	(A)	▨	update data block in memory/IorS
M	-/$IorS^D$			



$$\begin{tikzcd}
M \arrow[d, bend right=60, "GetS\ or\ PutM"'] \\
I\ or\ S \arrow[u, bend right=60, "GetM"']
\end{tikzcd}$$

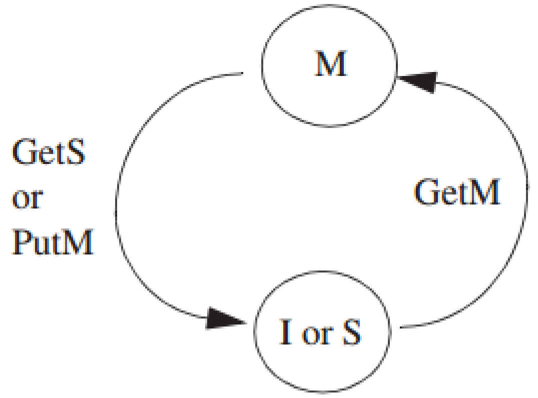
Memory controller

State	Bus Events			
	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS	send data block in Data message to requestor/M	▨	▨
IorS^D	▨ (A)	▨ (A)	▨	update data block in memory/IorS
M	-/IorS^D		-/IorS^D	


$$\begin{tikzcd}
M \arrow[d, bend right=60, "\text{GetS or PutM}"'] \\
\text{I or S} \arrow[u, bend right=60, "\text{GetM}"']
\end{tikzcd}$$

Memory controller

State	Bus Events			
	GetS	GetM	PutM	Data from Owner
IorS	send data block in Data message to requestor/IorS	send data block in Data message to requestor/M		
IorS^D	(A)	(A)		update data block in memory/IorS
M	-/IorS^D		-/IorS^D	



Non-atomic requests

Time can pass (and transactions can get ordered) between making a request and seeing it on the bus

Non-atomic requests

Time can pass (and transactions can get ordered) between making a request and seeing it on the bus

I->S and I->M need another transient state:

- IS^{AD}: Logically still in I mode, waiting to be ordered
- IS^{D}: Logically in S mode, just don't have the data

	load	store	replacement	OwnGetS	OwnGetM	OwnPutM	OtherGetS	OtherGetM	OtherPutM	Own Data response
I	issue GetS/IS^{AD}	issue GetM/IM^{AD}					-	-	-	
IS^{AD}	stall	stall	stall	-/IS^{D}			-	-	-	
IS^{D}	stall	stall	stall				(A)	(A)		-/S
IM^{AD}	stall	stall	stall		-/IM^{D}		-	-	-	
IM^{D}	stall	stall	stall				(A)	(A)		-/M
S										
M										

Non-atomic requests

Time can pass (and transactions can get ordered) between making a request and seeing it on the bus

I->S and I->M need another transient state:

- IS^{AD}: Logically still in I mode, waiting to be ordered
- IS^{D}: Logically in S mode, just don't have the data

S->M more complex:

- $$\begin{tikzcd} S \arrow[r, dashed, "store"'] & SM^{AD} \arrow[r, dashed, "Other\text{-}GetM"'] & IM^{AD} \end{tikzcd}$$

Can satisfy loads

Can't satisfy loads

	load	store	replacement	OwnGetS	OwnGetM	OwnPutM	OtherGetS	OtherGetM	OtherPutM	Own Data response
I	issue GetS/IS^{AD}	issue GetM/IM^{AD}					-	-	-	
IS^{AD}	stall	stall	stall	-/IS^{D}			-	-	-	
IS^{D}	stall	stall	stall				(A)	(A)		-/S
IM^{AD}	stall	stall	stall		-/IM^{D}		-	-	-	
IM^{D}	stall	stall	stall				(A)	(A)		-/M
S	hit	issue GetM/SM^{AD}	-/I				-	-/I	-	
SM^{AD}	hit	stall	stall		-/SM^{D}		-	-/IM^{AD}	-	
SM^{D}	hit	stall	stall				(A)	(A)		-/M
M										

Non-atomic requests

Time can pass (and transactions can get ordered) between making a request and seeing it on the bus

I->S and I->M need another transient state:

S->M more complex:

- $$\begin{tikzcd} S \arrow[r, "\text{store}"'] & SM^{AD} \arrow[r, "\text{Other-GetM}"'] & IM^{AD} \end{tikzcd}$$

  loads OK   loads not OK

M->I even more complex:

- $$\begin{tikzcd} M \arrow[r] & MI^{A} \arrow[r, "\text{Other-GetM/ Send data to requestor}"'] & II^{A} \arrow[r, "\text{Own-PutM/ Send NoData msg to memory}"'] & I \end{tikzcd}$$

  data still owned

	load	store	replacement	OwnGetS	OwnGetM	OwnPutM	OtherGetS	OtherGetM	OtherPutM	Own Data response
I	issue GetS/IS^{AD}	issue GetM/IM^{AD}	🟦	🟦	🟦	🟦	-	-	-	🟦
IS^{AD}	stall	stall	stall	-/IS^{D}	🟦	🟦	-	-	-	🟦
IS^{D}	stall	stall	stall	🟦	🟦	🟦	(A)	(A)	🟦	-/S
IM^{AD}	stall	stall	stall	🟦	-/IM^{D}	🟦	-	-	-	🟦
IM^{D}	stall	stall	stall	🟦	🟦	🟦	(A)	(A)	🟦	-/M
S	hit	issue GetM/SM^{AD}	-/I	🟦	🟦	🟦	-	-/I	-	🟦
SM^{AD}	hit	stall	stall	🟦	-/SM^{D}	🟦	-	-/IM^{AD}	-	🟦
SM^{D}	hit	stall	stall	🟦	🟦	🟦	(A)	(A)	🟦	-/M
M	hit	hit	issue PutM/MI^{A}	🟦	🟦	🟦	send data to requestor and to memory/S	send data to requestor/I	-	🟦
MI^{A}										
II^{A}										

	load	store	replacement	OwnGetS	OwnGetM	OwnPutM	OtherGetS	OtherGetM	OtherPutM	Own Data response
I	issue GetS/IS^{AD}	issue GetM/IM^{AD}					-	-	-	
IS^{AD}	stall	stall	stall	-/IS^{D}			-	-	-	
IS^{D}	stall	stall	stall				(A)	(A)		-/S
IM^{AD}	stall	stall	stall		-/IM^{D}		-	-	-	
IM^{D}	stall	stall	stall				(A)	(A)		-/M
S	hit	issue GetM/SM^{AD}	-/I				-	-/I	-	
SM^{AD}	hit	stall	stall		-/SM^{D}		-	-/IM^{AD}	-	
SM^{D}	hit	stall	stall				(A)	(A)		-/M
M	hit	hit	issue PutM/MI^{A}				send data to requestor and to memory/S	send data to requestor/I	-	
MI^{A}	hit	hit	stall			send data to memory/I	send data to requestor and to memory/II^{A}	send data to requestor/II^{A}		
II^{A}										

	load	store	replacement	OwnGetS	OwnGetM	OwnPutM	OtherGetS	OtherGetM	OtherPutM	Own Data response
I	issue GetS/IS^{AD}	issue GetM/IM^{AD}					-	-	-	
IS^{AD}	stall	stall	stall	-/IS^{D}			-	-	-	
IS^{D}	stall	stall	stall				(A)	(A)		-/S
IM^{AD}	stall	stall	stall		-/IM^{D}		-	-	-	
IM^{D}	stall	stall	stall				(A)	(A)		-/M
S	hit	issue GetM/SM^{AD}	-/I				-	-/I	-	
SM^{AD}	hit	stall	stall		-/SM^{D}		-	-/IM^{AD}	-	
SM^{D}	hit	stall	stall				(A)	(A)		-/M
M	hit	hit	issue PutM/MI^{A}				send data to requestor and to memory/S	send data to requestor/I	-	
MI^{A}	hit	hit	stall			send data to memory/I	send data to requestor and to memory/II^{A}	send data to requestor/II^{A}		
II^{A}	stall	stall	stall			send NoData to memory/I	-	-		

Why send to memory controller?

- It observes the PutM, so it waits for data. (It has no way of knowing when core issued PutM; it only sees it on the bus.)
- On the other hand, can't send real data, because ours might be stale by now!

	load	store	replacement	OwnGetS	OwnGetM	OwnPutM	OtherGetS	OtherGetM	OtherPutM	Own Data response
II^A	stall	stall	stall			send NoData to memory/I	-	-		

Memory controller

	GetS	GetM	PutM	Data From Owner	NoData
IorS	send data to requestor	send data to requestor/M	-/IorS^D		
IorS^D	(A)	(A)		write data to LLC/memory /IorS	-/IorS
M	-/IorS^D		-/M^D		
M^D					

Memory controller

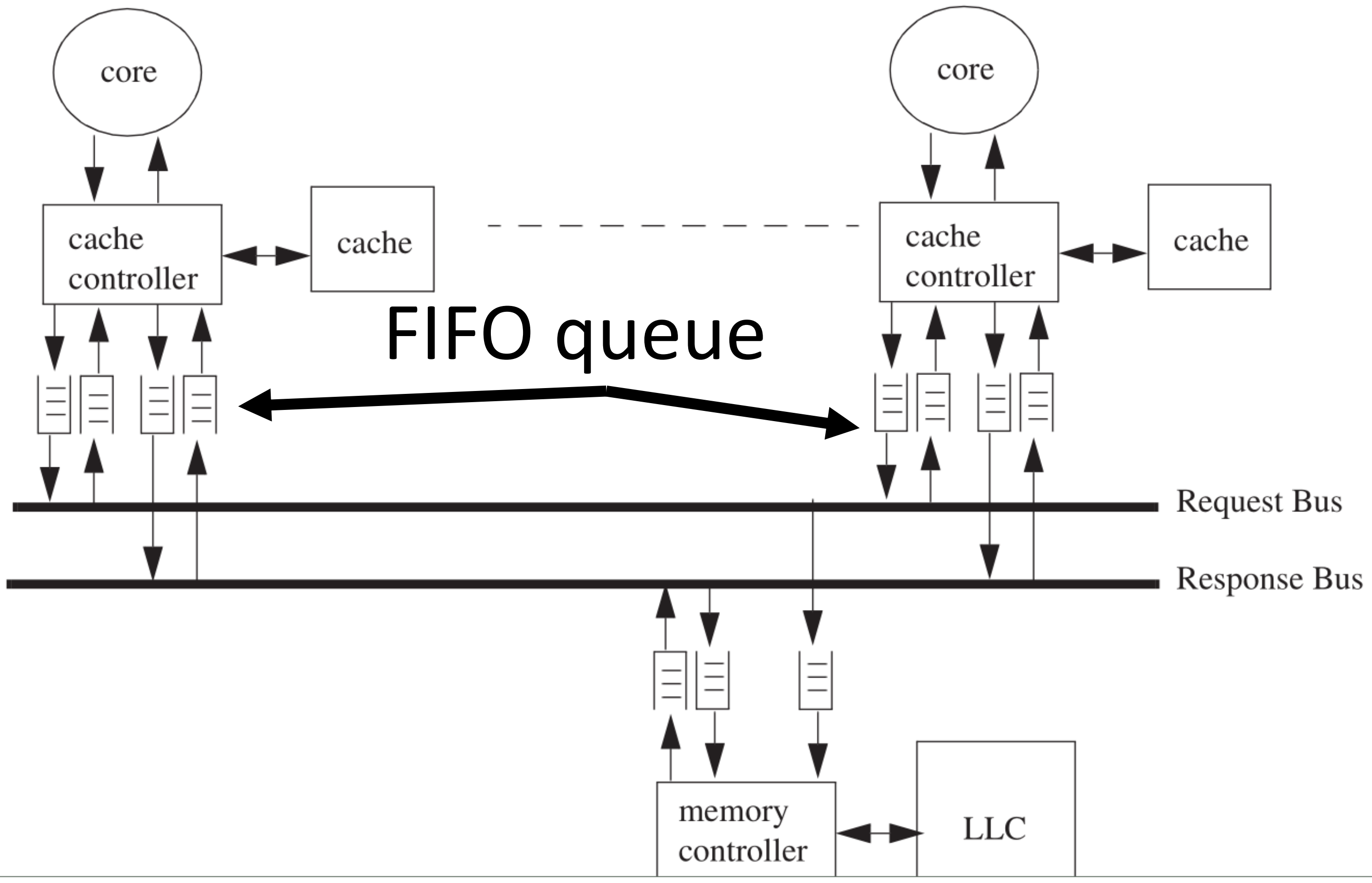
	GetS	GetM	PutM	Data From Owner	NoData
IorS	send data to requestor	send data to requestor/M	-/IorS^D		
IorS^D	(A)	(A)		write data to LLC/memory /IorS	-/IorS
M	-/IorS^D		-/M^D		
M^D	(A)	(A)		write data to LLC/IorS	-/M

Upgrade transaction

- Current S->M issues a GetM, which waits for the data from the LLC
- But we have the data: an *upgrade* transaction would invalidate other sharers and then go to M state.
- Complicated by possibility of state changing before transaction gets ordered; see book

Non-atomic transactions

Can observe another transaction before getting response to mine



Non-atomic transactions

Example problem:

- Issued (& observed) GetM: line is logically mine
- Observe a GetS (but I don’t have the data)

Non-atomic transactions

Example problem:

- Issued (& observed) GetM: line is logically mine
- Observe a GetS (but I don’t have the data)

Simple solution:

- Stall response (buffer request and respond only when data arrives)

Non-atomic transactions

Example problem:

- Issued (& observed) GetM: line is logically mine
- Observe a GetS (but I don’t have the data)

Simple solution:

- Stall response (buffer request and respond only when data arrives)
- Deadlock? No, because my request was already ordered, now just waiting for data

Non-stalling non-atomic txns

Instead of stalling requests, can add new transient states:

IM^{D}S: in I, going to M, waiting for data, and when
data arrives will go to S

IS^{D}I, IM^{D}I, IM^{D}SI

To avoid livelock, must perform the instruction that triggered the original request (e.g., store for IM^{D}S) when data arrives, before switching to the final state

- (Otherwise, in the IM^{D}S case, we'd end up in S state with a store pending, which would trigger another GetM request… which can go on forever)

MESI protocol

- Optimizes for a common case of write-after-read; used in commercial processors
- MSI: GetS followed by GetM
- MESI: GetS can get line in E(xclusive) state, if no other cache has line in S state. Then, later store can silently upgrade from E to M
  - Requires: knowing that no other cache has line in S state
  - LLC can track this; hard to do precisely due to silent evictions, but “LLC state == I” is good enough

Atomic RMW instructions

No need to “lock the bus”

- Get line in M state
- Stall incoming coherence requests
- Complete instruction
- Resume processing incoming coherence transactions

Directories

Scalability problems with snooping:

- Bandwidth
- Having to process unrelated coherence requests
- Enforcing global total order

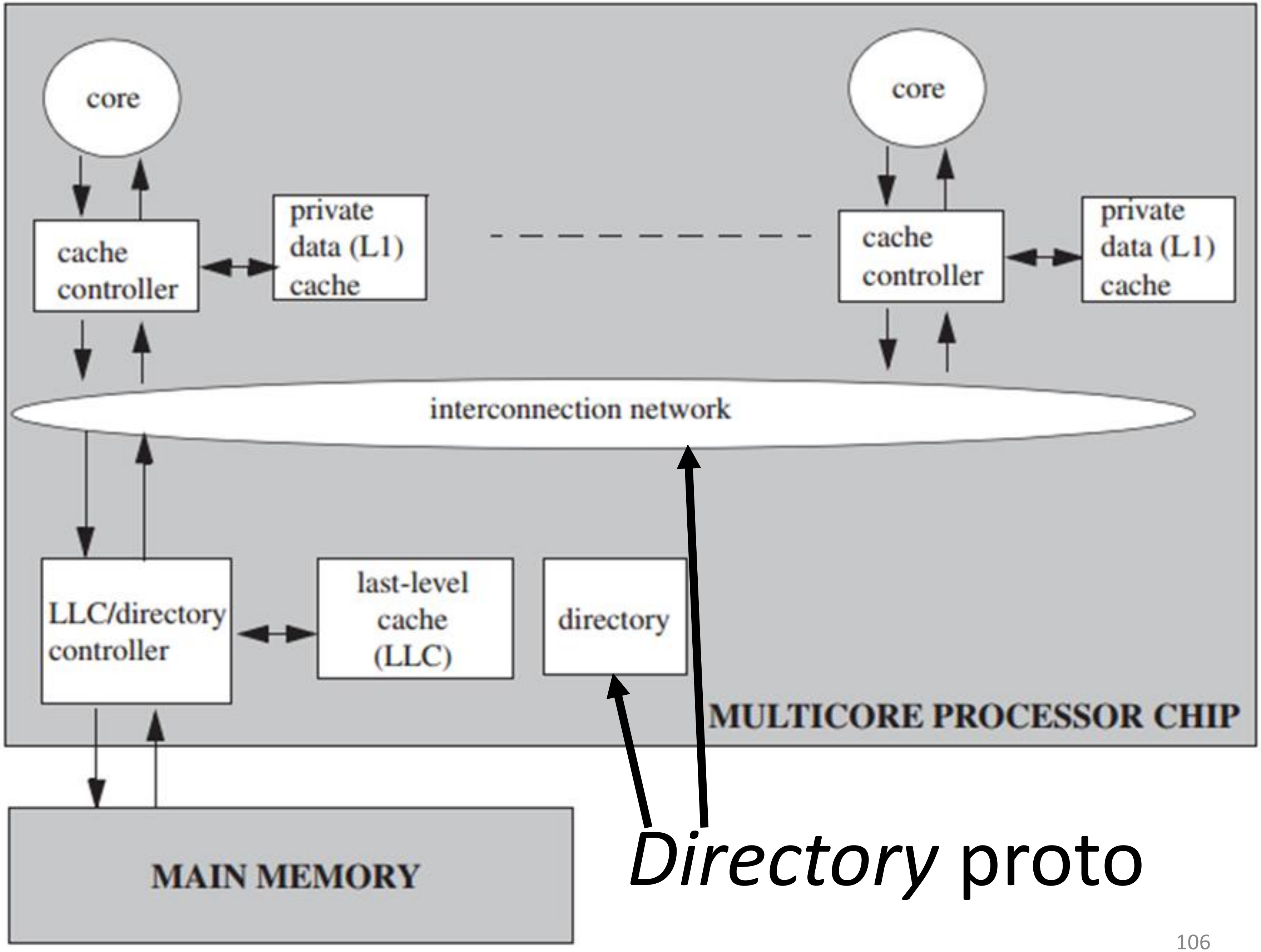
Directories

Scalability problems with snooping:

- Bandwidth
- Having to process unrelated coherence requests
- Enforcing global total order

Directory:

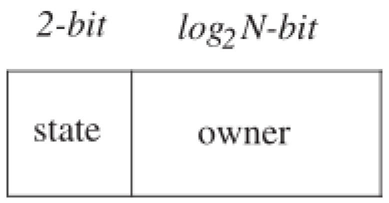
- Maintains global view of coherence state instead of a distributed view
- Requests go to it rather than being broadcast
- Trade-off: latency, need for acks



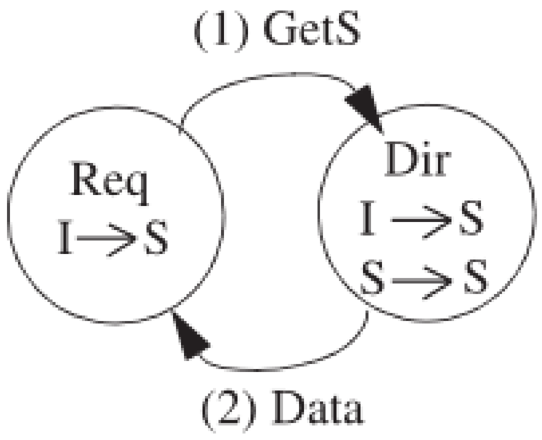
Directory proto

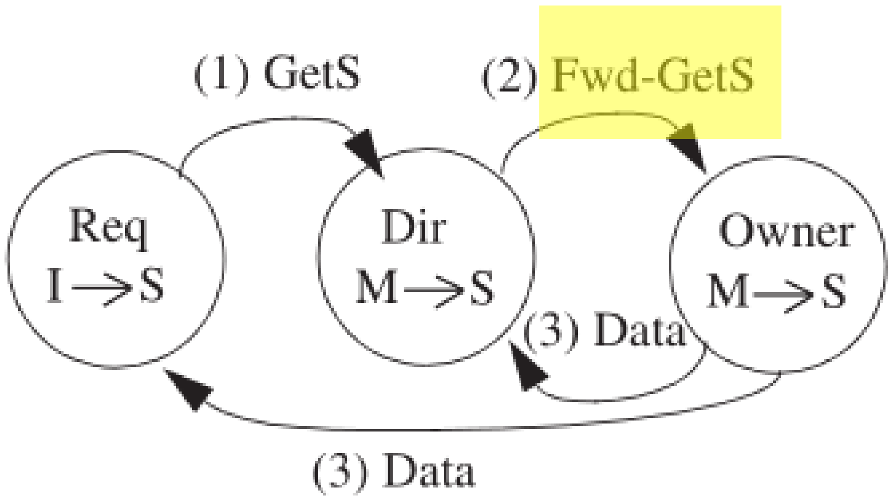
Directory state

- Line is owned by directory (i.e., will respond with data) unless it's in M state at some core



I -> S flow



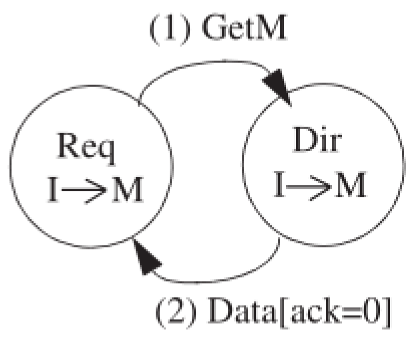


line owned by dir

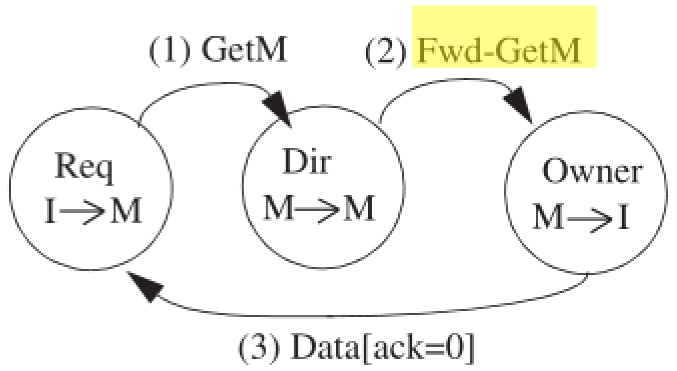
line owned by core *Owner*

new message types

I -> M flow



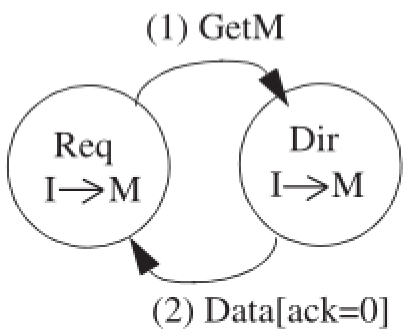
line only in mem



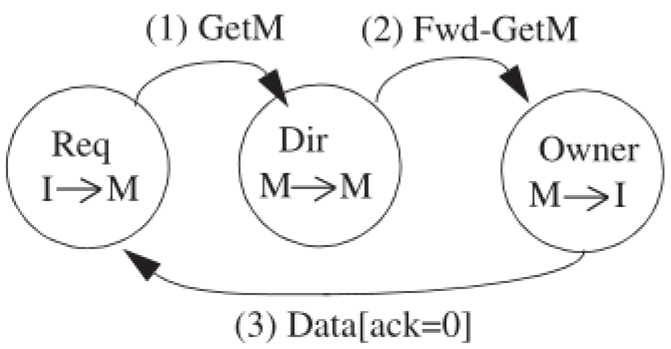
line M at *Owner*

new message types

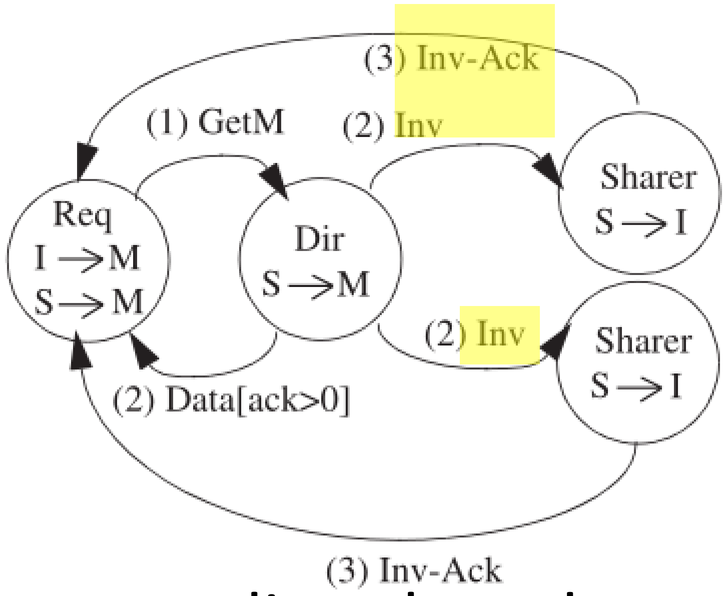
I -> M flow



line only in mem



line M at *Owner*



line shared

if *Req* is only sharer, Data[ack=0]

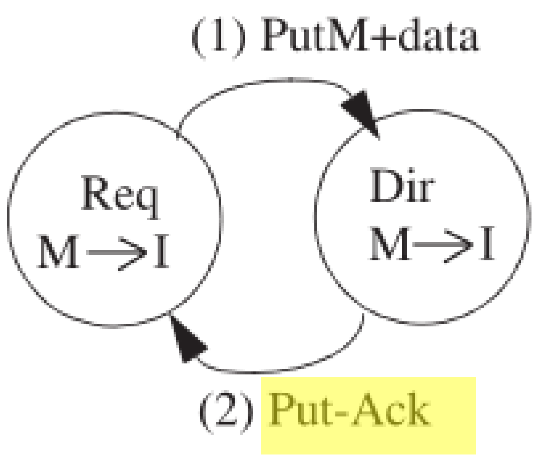
new message types

Directory state

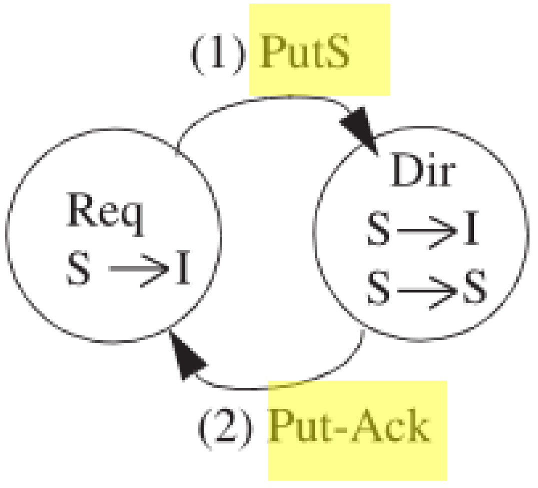
- Line is owned by directory (i.e., will respond with data) unless it's in M state at some core
- **Directory needs to know who has line in S state**
  - In snooping, sharers observed all requests; now they don't, and directory needs to forward requests to them

2-bit	*$log_2 N$-bit*	*N-bit*
state	owner	sharer list (one-hot bit vector)

Evictions



line M at *Req*



line shared

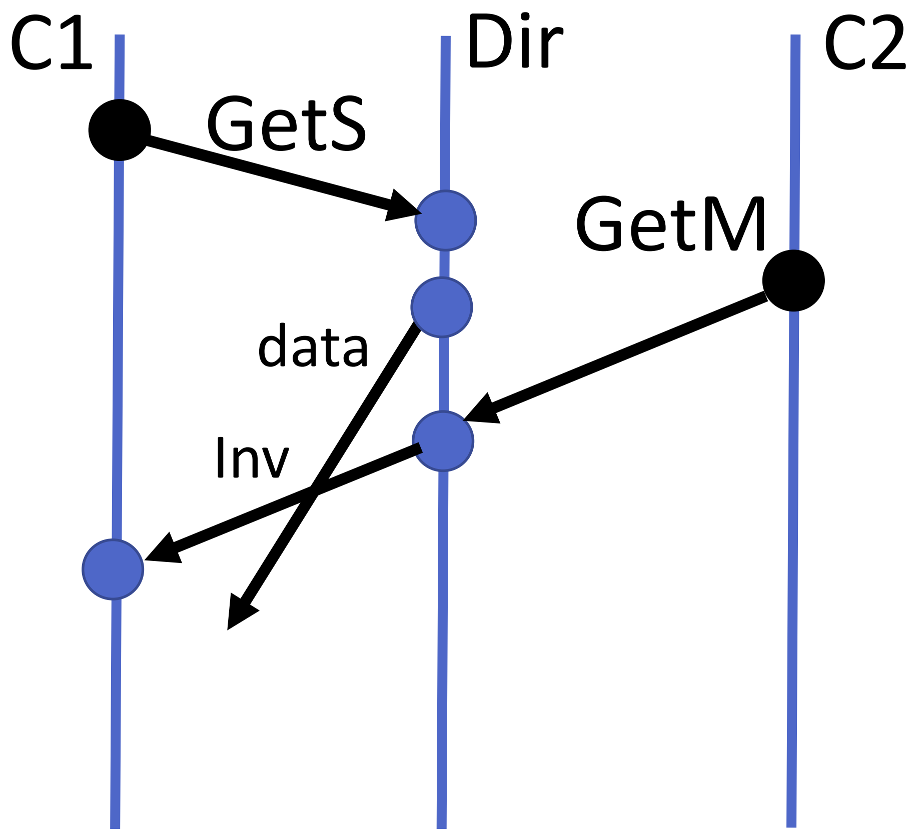
new message types

Evictions

- PutM now includes data
  - In snooping, sent two messages in parallel
- Evictions from S no longer silent
  - Directory needs to keep track of sharers
  - Avoids a race:
    - Cache silently evicts, then does a GetS
    - Now gets an Inv; was this Inv sent for the prior S state or the next one?

Races

- Multiple transactions in progress for a line
- Example:

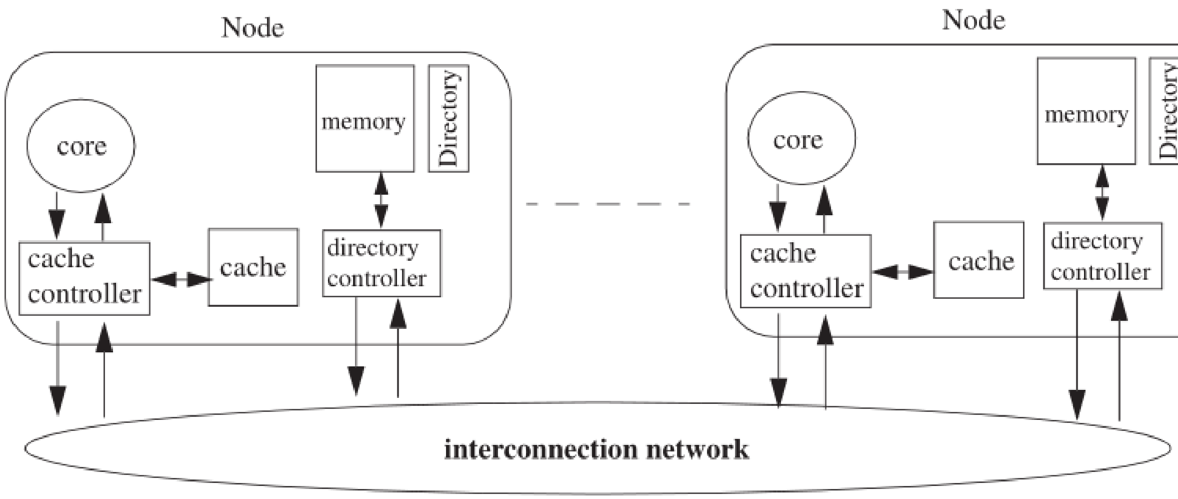


Races

- Multiple transactions in progress for a line
- Solutions:
  - Stall racy requests (costs performance)
  - More transient states

Distributed directories

- Everything works (and scales better) if we have several directories, and hash the line to obtain its directory
- Example: NUMA system



- Same idea applies within chip (banked LLC and directory)

Summary

- Cache coherence
- MSI & MESI
  - Commercial protocols are based on these
- Snooping and directory-based protocols
- **Important:** state machine way of looking at an algorithm