Advanced Computer Architecture

COMP 5123

Fall 2016

Computer Science Department

Prairie View A&M University

Mary Heejin Kim (aka Heejin Lim), Ph.D.

Chapter 17 Parallel Processing

- Cache Coherence

Cache Coherence

Software Solutions

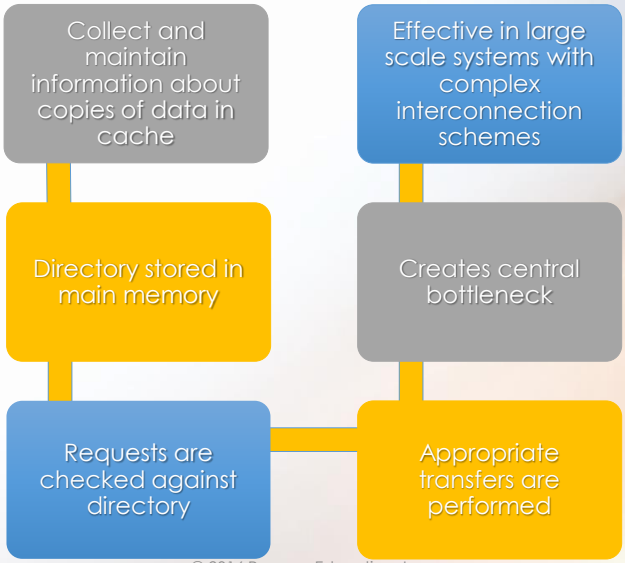
- Attempt to avoid the need for additional hardware circuitry and logic by relying on the compiler and operating system to deal with the problem
- Attractive because the overhead of detecting potential problems is transferred from run time to compile time, and the design complexity is transferred from hardware to software
  - However, compile-time software approaches generally must make conservative decisions, leading to inefficient cache utilization

Cache Coherence

Hardware-Based Solutions

- Generally referred to as cache coherence protocols
- These solutions provide dynamic recognition at run time of potential inconsistency conditions
- Because the problem is only dealt with when it actually arises there is more effective use of caches, leading to improved performance over a software approach
- Approaches are transparent to the programmer and the compiler, reducing the software development burden
- Can be divided into two categories:
  - Directory protocols
  - Snoopy protocols

Directory Protocols



Snoopy Protocols

- Distribute the responsibility for maintaining cache coherence among all of the cache controllers in a multiprocessor
  - A cache must recognize when a line that it holds is shared with other caches
  - When updates are performed on a shared cache line, it must be announced to other caches by a broadcast mechanism
  - Each cache controller is able to "snoop" on the network to observe these broadcast notifications and react accordingly
- Suited to bus-based multiprocessor because the shared bus provides a simple means for broadcasting and snooping
  - Care must be taken that the increased bus traffic required for broadcasting and snooping does not cancel out the gains from the use of local caches
- Two basic approaches have been explored:
  - Write invalidate
  - Write update (or write broadcast)

Write Invalidate

- Multiple readers, but only one writer at a time
- When a write is required, all other caches of the line are invalidated
- Writing processor then has exclusive (cheap) access until line is required by another processor
- Most widely used in commercial multiprocessor systems such as the x86 architecture
- State of every line is marked as modified, exclusive, shared or invalid
  - For this reason the write-invalidate protocol is called *MESI*

Write Update

Can be multiple readers and writers

When a processor wishes to update a shared line the word to be updated is distributed to all others and caches containing that line can update it

Some systems use an adaptive mixture of both write-invalidate and write-update mechanisms

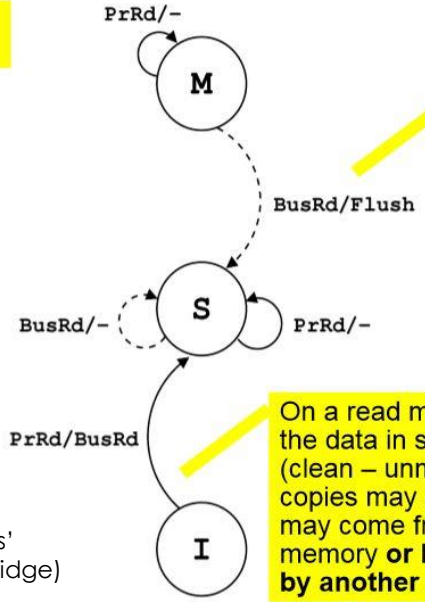
MESI Protocol

To provide cache consistency on an SMP the data cache supports a protocol known as MESI:

- Modified
  - The line in the cache has been modified and is available only in this cache
- Exclusive
  - The line in the cache is the same as that in main memory and is not present in any other cache
- Shared
  - The line in the cache is the same as that in main memory and may be present in another cache
- Invalid
  - The line in the cache does not contain valid data

MSI write-back invalidate protocol

Let's build the protocol up in stages...



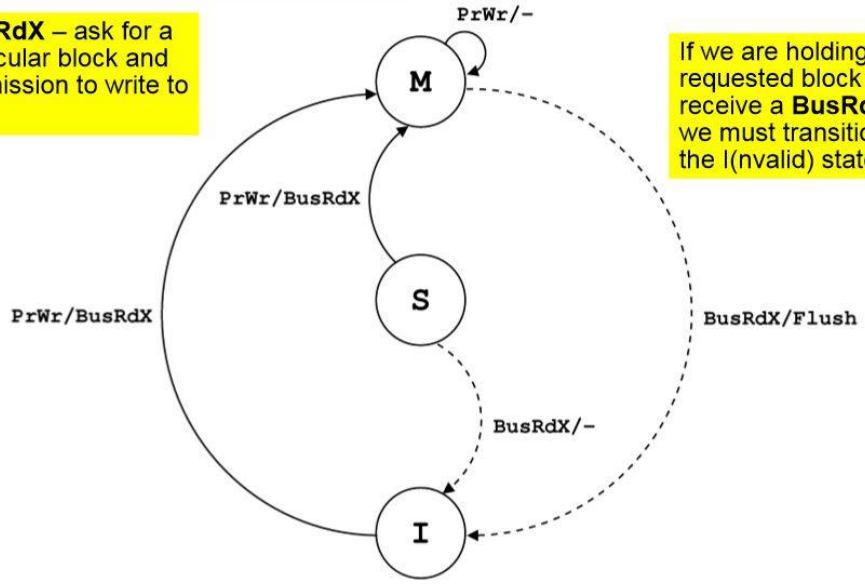
If another cache needs the block that we have in state M (the block is dirty/modified), we must flush the block to memory and provide the data to the requesting cache (over the bus)

On a read miss, we get the data in state S (clean – unmodified, other copies may exist). Data may come from main memory **or be provided by another cache**

Ref: Dr. Robert Mullins' slides (Univ.of Cambridge)

MSI write-back invalidate protocol

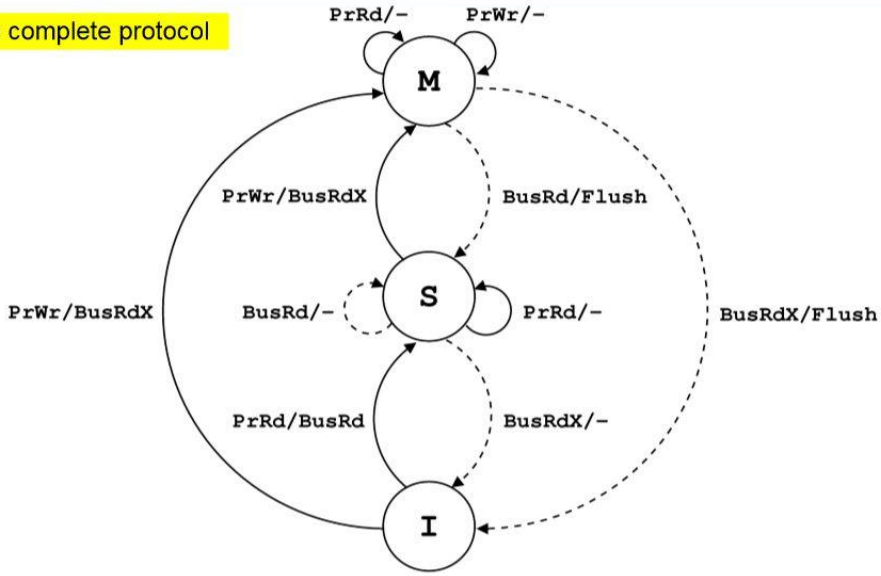
BusRdX – ask for a particular block and permission to write to it.



If we are holding the requested block and receive a **BusRdX**, we must transition to the I(nvalid) state

MSI write-back invalidate protocol

The complete protocol

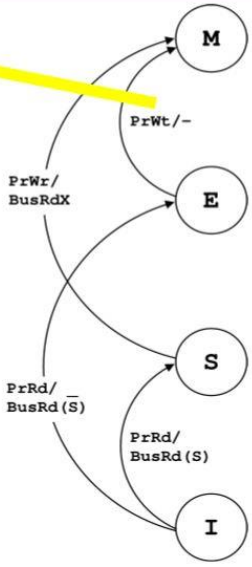


Suppose we have three processors which have its own cache and have a shared main memory. Each processor is reading/writing same value from the main memory as a given stream below. Please show the status changes of each cache and show the last states when this system uses MSI as its cache coherence protocol.

R2 R1 W3 R1 W2 R1 R3

MESI protocol

If we are in state E and need to write, we require no bus transaction to take place

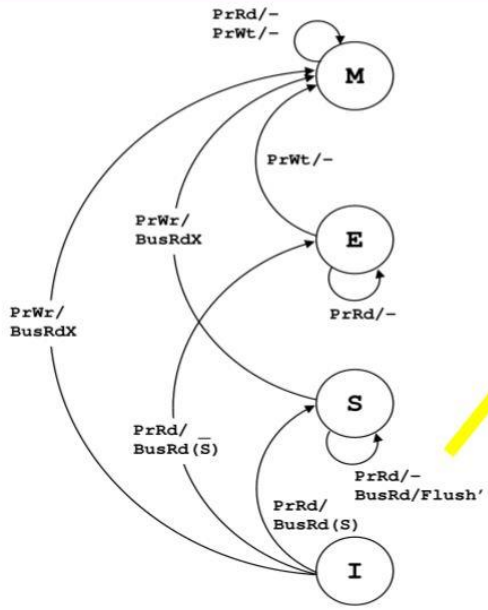


The shared signal (S) is used to determine if any caches currently hold the requested data on a PrRd.

BusRd(S) means the bus read transaction caused the shared signal to be asserted (another cache has a copy of the data). **BusRd(Not S)** means no cache has the data.

This signal is used to decide if we transition from state I to S or E

MESI protocol

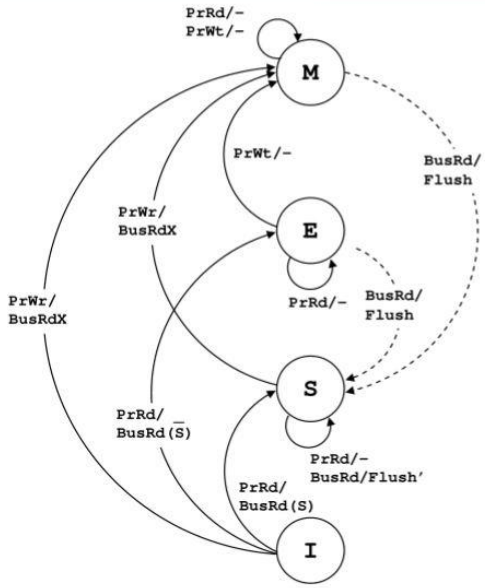


BusRd/Flush'

To enable cache-to-cache sharing we must be sure to select only one cache to provide the required block. Flush' is used to indicate that only **one** cache will flush the data (in order to supply it to the requesting cache). This is easy in a bus-based system of course as only one transaction can take place at a time.

Of course a cache will normally be able to provide the block faster than main memory.

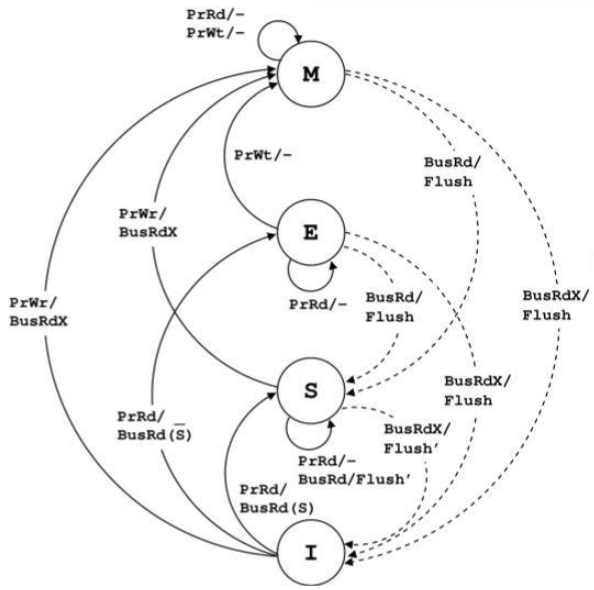
MESI protocol



BusRd/Flush

If we are in state M or E and other cache requests the block, we provide the data (we have the only copy) and move to state S (clean & zero or more copies)

MESI protocol

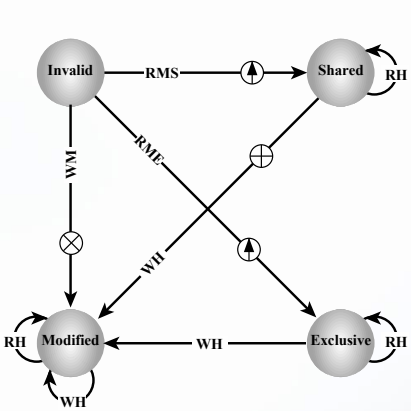


BusRdX/Flush

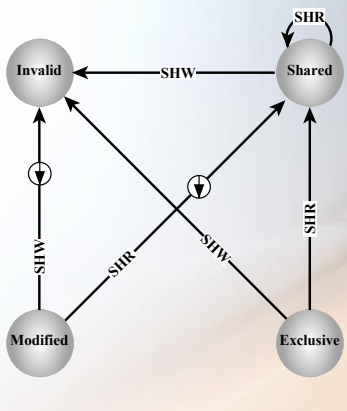
If we are in states M, E or S and receive a BusRdX we simply flush our data and move to state I

Table 17.1
MESI Cache Line States

	M **Modified**	**E** **Exclusive**	**S** **Shared**	**I** **Invalid**
This cache line valid?	Yes	Yes	Yes	No
The memory copy is…	out of date	valid	valid	—
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line…	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus



(a) Line in cache at initiating processor



(b) Line in snooping cache

RH	Read hit		Dirty line copyback
RMS	Read miss, shared		
RME	Read miss, exclusive		Invalidate transaction
WH	Write hit		
WM	Write miss		Read-with-intent-to-modify
SHR	Snoop hit on read		
SHW	Snoop hit on write or read-with-intent-to-modify		Cache line fill

Figure 17.6 MESI State Transition Diagram