Lecture 17-18: September 15, 2020

Computer Architecture and Organization-II

Biplab K Sikdar

The fourth C

0.6 Cache Coherence

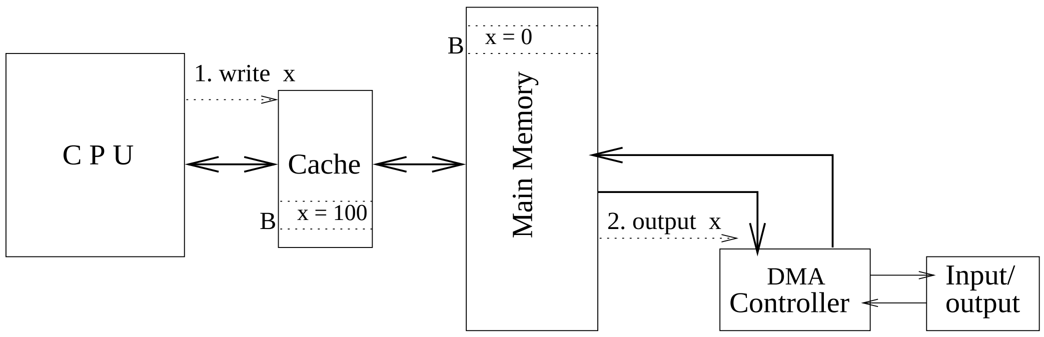


Figure 39: Cache data inconsistency

Figure 39: A single processor system with cache, MM, and DMA controller.

Let assume cache write-back policy.

If CPU writes x = 100 to cache and output device reads x through DMA controller,

Then value received for x at the output device is 0 (old/stale value of x).

This causes data inconsistencies in memory system.

However, if cache write policy is write-through,

Then there will be no such data inconsistency.

For CPU read x and DMA modifies x in MM prior to that??

In a multiprocessor system (Figure 40), each processor has its own/private cache.

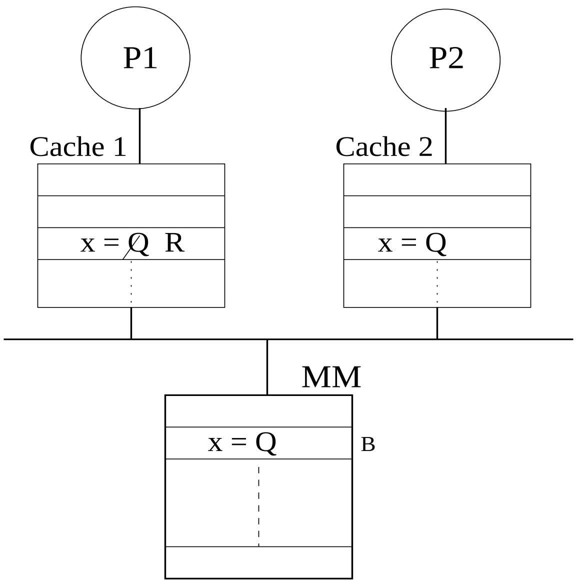


Figure 40: Multiprocessor system

More than one processor (P_1 and P_2) may share item x (say x = Q) of block B.

If P_1 updates x = R in its cache,

P_2's cache may contain old/stale copy of x = Q.

This results in inconsistency in cached copies of block B.

Definition 0.1 *Cache coherence defines what values can be returned by a read operation generated by a processor i.e, how do other processors see a memory update?*

Coherence assures values written by one processor are read by other processors.

But it says nothing about when writes will become visible.

Multiprocessor system should ensure: all cached copies of block B are coherent.

0.6.1 Cache coherence problem

In a multiprocessor, only one copy of X exists in MM.

But each processor's cache stores its own copy of X.

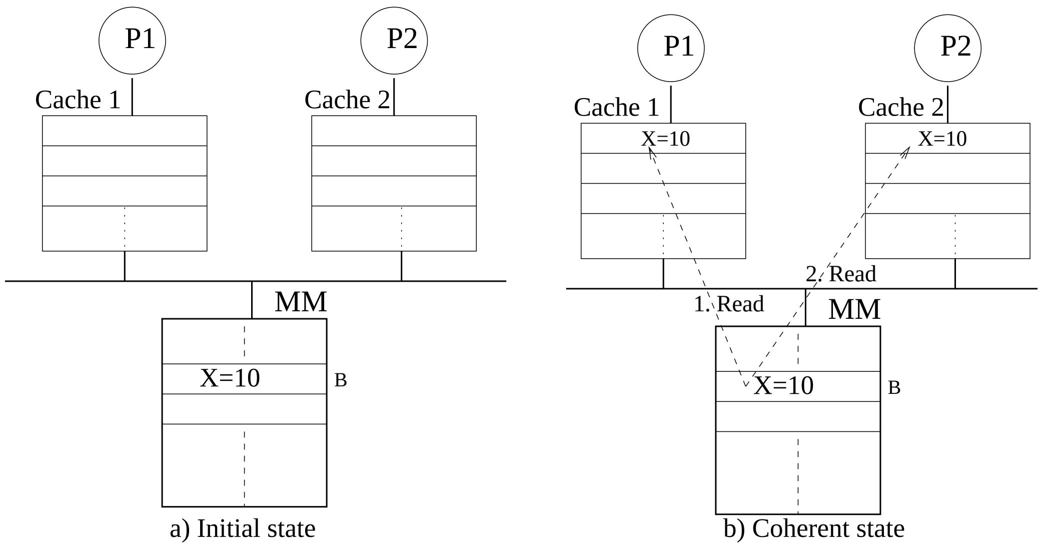


Figure 41: Cache coherence

In Figure 41(a), initially, caches were empty and in MM $X = 10$.

Now if

1. P_1 reads X, it results in a cache miss.

Block B containing X is then read from MM to cache 1

-that is, in cache 1, $X = 10$.

2. P_2 reads X, it also results in cache miss.

Block B is then read from MM to cache 2. Then in cache 2, $X = 10$.

States of cached copies and MM copy are shown in Figure 41(b).

In caches and in MM, $X = 10$. Therefore, it is a coherent state.

Now let

3. P_2 updates $X = X + 5$ -it is a hit for X.

Write operation done at cache 2 and in cache 2, $X = 15$.

But in cache 1 $X = 10$, and in cache 2 $X = 15$, and in MM $X = 10$ for write-back.

That is, caches are incoherent (Figure 42(b)).

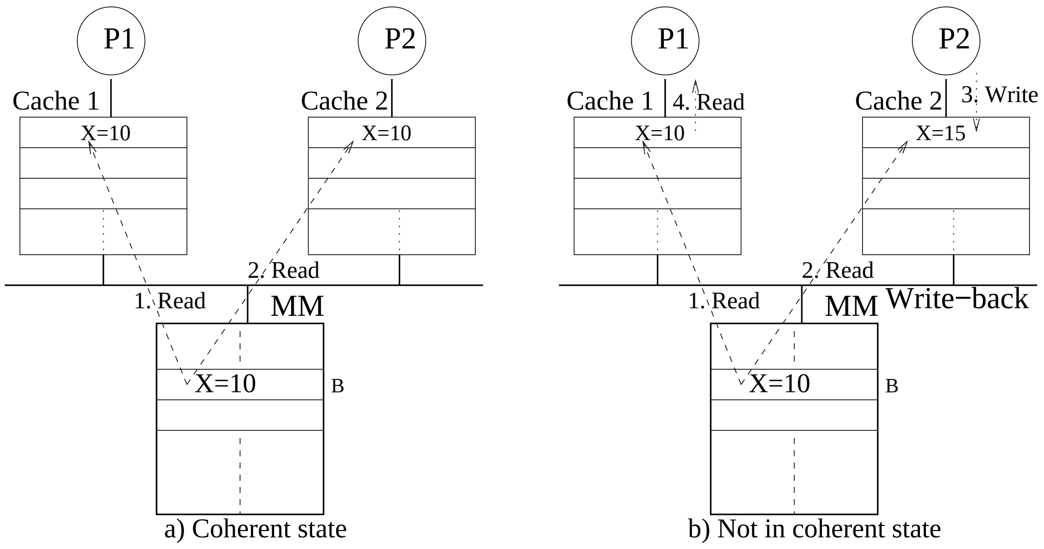


Figure 42: Incoherent state

4. If P_1 now reads X.

It is a hit for X as block B containing X already is in cache 1.

That is, P_1 receives stale value of $X = 10$ (Figure 42(b)).

0.7 Enforcing Cache Coherence

For enforcing cache coherence, a number of schemes/protocols/hardwires are there.

Following schemes are considered for enforcing cache coherence.

(i) Write-invalidate

(ii) Write-update

(iii) Write-once

0.7.1 Write-invalidate

In write-invalidate scheme, a write to shared data X/B by processor P_1 -

1. Invalidates all other cached copies of B (but not MM copy, Figure 43(a)). However, for write-through, X in MM will be modified (Figure 43(b)).

2. A subsequent request for B, by other processor than P_2, is treated as miss.

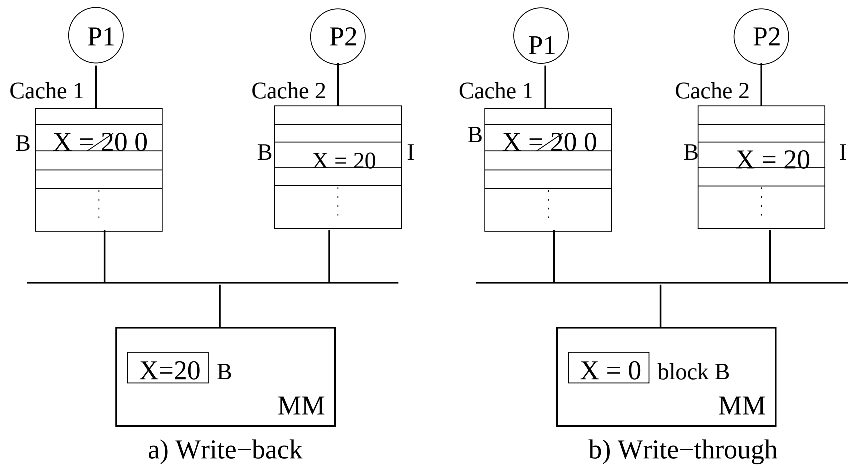


Figure 43: Write-invalidate cache coherence scheme

Activities in write-invalidate

Consider Figure 44.

Cache 1 initially holds block B. Assume initially X = 20 in B.

Cache write policy is write-back.

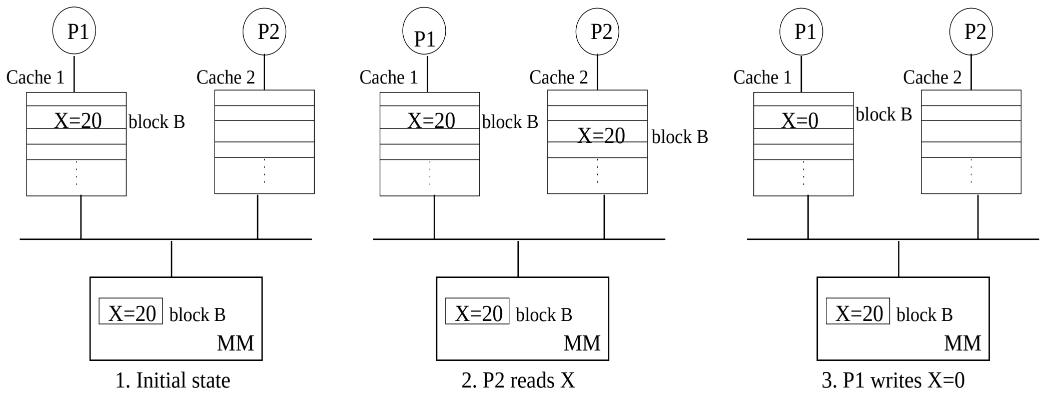


Figure 44: Write-invalidate cache coherence example

Following points to X in cache 1, cache 2, and MM after different activities.

Activity	*X at*		
	cache 1	*cache* 2	*MM*
1. *Initial state*	20		20
2. P_2 *reads* X, *cache* 2 *miss*	20	20	20
3. P_1 *writes* $X = 0$, *cache* 1 *hit*, *Invalidate all cached copies*	0		20

Now, say (Step 4) P_2 reads X after P_1 invalidates B (Step 3), it is a cache 2 miss.

How P_2 will get latest copy of B??

Follow example of write invalidate scheme working on snooping bus (WAIT).

0.7.2 Write-update

Also referred to as write-broadcast scheme.

An updation in block B of P$_i$'s cache, upgates all cached copies of B and MM copy.

Any subsequent request for B from any processor is treated as hit.

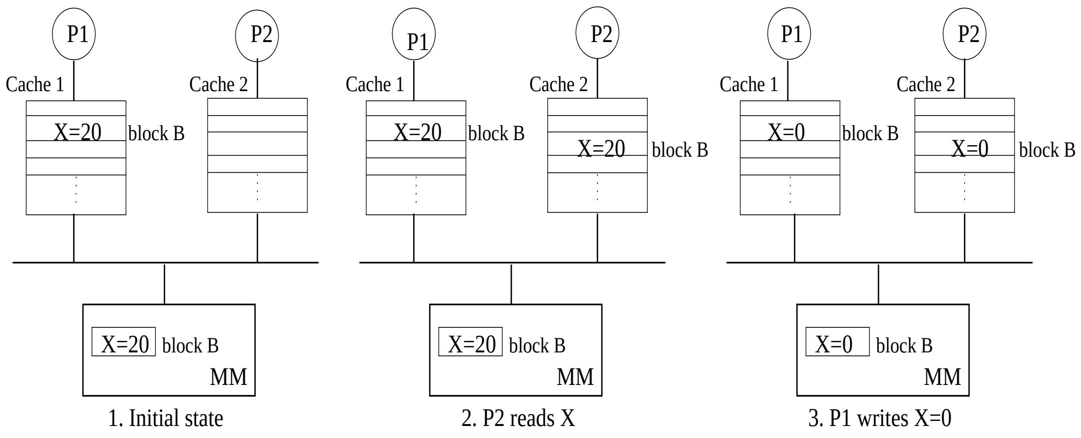


Figure 45: Write-update cache coherence scheme

Consider Figure 45. Cache 1 initially holds block B.

Assume X in B is initially 20.

Following points to X in cache 1, cache 2, and MM after different activities.

Policy here is write-back (however, suppressed by write-through of write-update).

$Activity$	$X\ at$		
	$cache\ 1$	$cache\ 2$	MM
1. $Initial\ state$	20		20
2. $P_2\ reads\ X,\ cache\ 2\ miss$	20	20	20
3. $P_1\ writes\ X = 0,\ cache\ 1\ hit,\ updates$ $shared\ copies\ and\ MM\ copy$	0	0	0
4. $P_2\ reads\ X,\ cache\ 2\ hit$	0	0	0

Write update scheme requires high bandwidth as it requires broadcasting.

In write-update, each write is followed by an update. Therefore,

Consecutive writes to same block, with no read, require multiple write broadcast.

On the other hand, in write-invalidate, invalidation messages is to be controlled.

For a shared block B,

If write to word is followed by number of read B by different processors, it needs
Only one write-broadcast in a write-update scheme.

On the other hand, for such case,

Write-invalidate scheme requires much time to serve read requests.
(As each read request is a read miss)

In reality, a hybrid scheme (write-update and write-invalidate scheme) is used.

0.7.3 Write-once

The principle of write-once scheme is:

1. If a block B in cache C_i at processor P_i is written for first time,

   All cached copies of B are declared invalid (I) and MM copy is updated.

2. If next time B is updated in C_i, no invalidation message is generated.

   MM write follows - write-back policy.

3. When P_j tries to access B, B is supplied to C_j from C_i.

   MM copy is also updated.

Write-once reduces overall bus traffic for consecutive writes by a processor to B.

The X in cache 1, cache 2, and MM after different activities in write-once follows.

$Activity$	$X\ at$		
	$cache\ 1$	$cache\ 2$	MM
$0.\ Initial\ state$			20
$1.\ P_1\ reads\ X,\ cache\ 1\ miss$	20		20
$2.\ P_2\ reads\ X, cache\ 2\ miss$	20	20	20
$3.\ P_1\ writes\ X = 0,\ cache\ 1\ hit, write\ through\ MM,$			
$Invalidate\ other\ cached\ copies$	0		0
$4.\ P_1\ writes\ X = 5,\ cache\ 1\ hit, updates\ cache\ 1\ only$	5		0
$5.\ P_2\ reads\ X, cache\ 2\ miss,\ P_1\ cache\ 1\ intervenes$	5	5	5

0.7.4 False sharing miss

In cache coherence, write-invalidate scheme shows better performance.

However, for large block size, write-invalidate scheme reports *false sharing miss*.

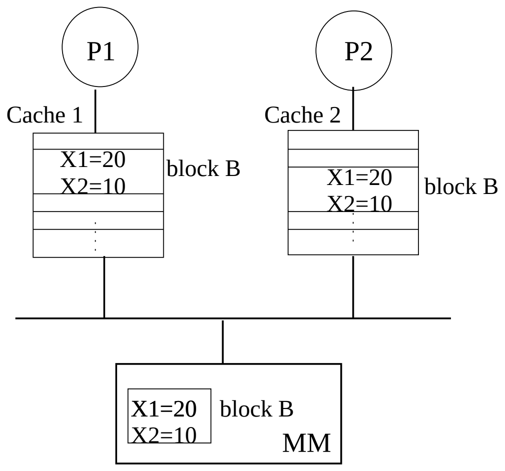


Figure 46: False sharing miss in write-invalidate scheme

Consider Figure 46. Variables X_1 and X_2 are in same block B.

Block B is in cache 1 and cache 2 at time 0.

Assume following activities, in sequence.

$Time$	P_1	P_2	$Remark$
1.	$write\ X_1$	$--$	$Invalidate\ block\ B\ in\ cache\ 2$
2.	$--$	$read\ X_2$	$False\ sharing\ read\ miss$
3.	$--$	$write\ X_2$	$Invalidate\ block\ B\ in\ cache\ 1$
4.	$write\ X_1$	$--$	$False\ sharing\ write\ miss$

False sharing miss is identified at time 2 and 4.

It is due to writes to two different words (X_1 and X_2) of B by two processors P_1/P_2.

For small-scale multiprocessors, we adopt a hardware solution for cache coherence.

Two major hardware based coherence solutions are:

(i) Snoopy protocol, and
(ii) Directory based protocol.

0.8 Snoopy Protocol

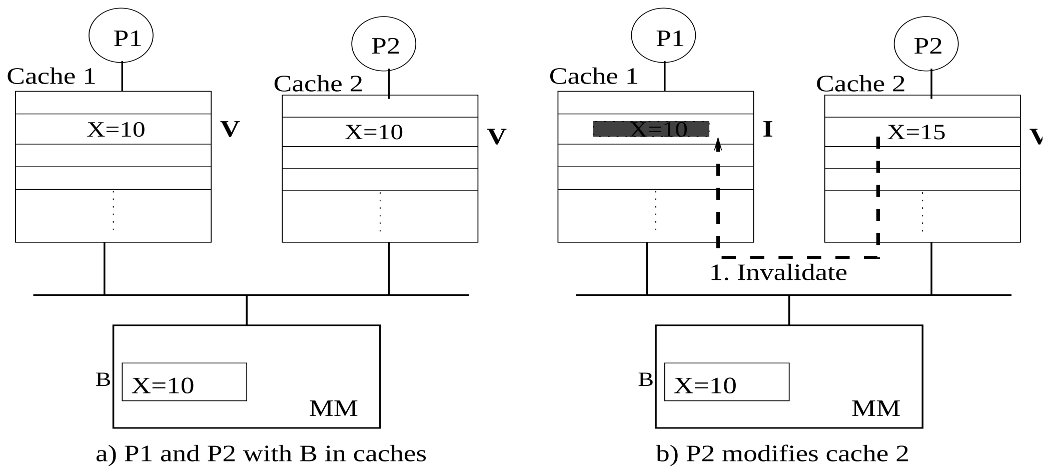


Figure 47: Snoopy protocol

Snoopy protocol is effective in a system with broadcast network (like bus).

Following figures describe snoopy protocol based system with two processors P_1/P_2.

Initially, (Figure 47(a)) both caches of P_1 and P_2 are having B (X = 10).
Then P_2 modifies X to 15 (Figure 47(b)).

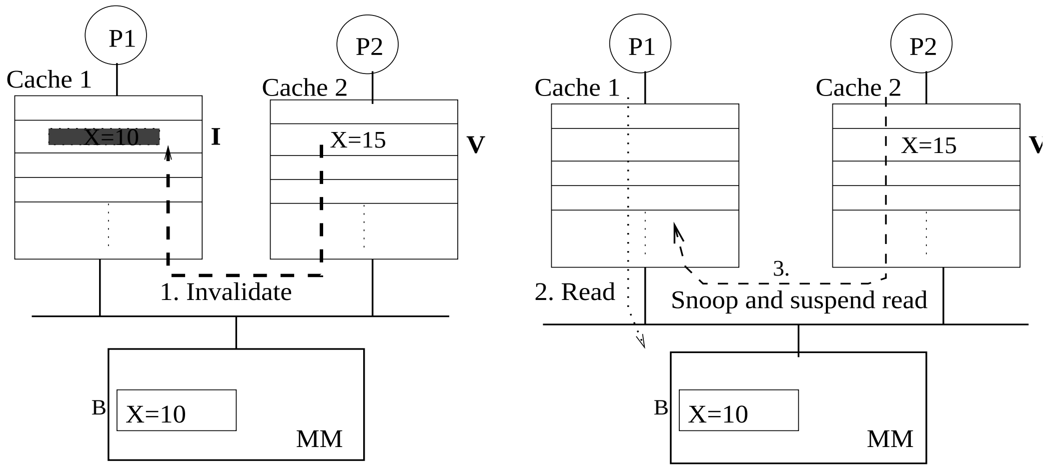
In snoopy based system, with write-invalidate scheme, the events occurred are:

1. P_2 generates an invalidate transaction on bus (Cache 2 has B in modified state).
Other processors check (snoop) invalidate X transaction.
If a processor (P_1) has a cached copy of B, P_1 marks its copy as invalid (I).
This is shown as signal 1 in Figure 47(b).

2. Now, read X by a processor (P_1) causes a cache read miss.

It then initiates a bus transaction to read B (Figure 48(b)) from MM (signal 2).

Handling of read request depends on cache write policy.



a) P2 modifies cache 2 b) P1 searches for exclusive copy

Figure 48: Snoopy protocol

For write-through, MM has latest update. So, P_1 reads B from MM.

For write back, MM has stale copy of B and updated copy is in cache 2, therefore,

3. P_2 always keeps vigilance on bus transaction through snoopy hardware.

It detects MM read for B by P_1 and generates suspend signal 3 of Figure 48(b).

4. P_2 then issues a write-back signal (signal 4 of Figure 49(b)). T

Modified B is written to MM. P_1 is informed to continue with the read B.

5. P_1 retries read (signal 5 of Figure 49(c)) for B and gets updated B from MM.

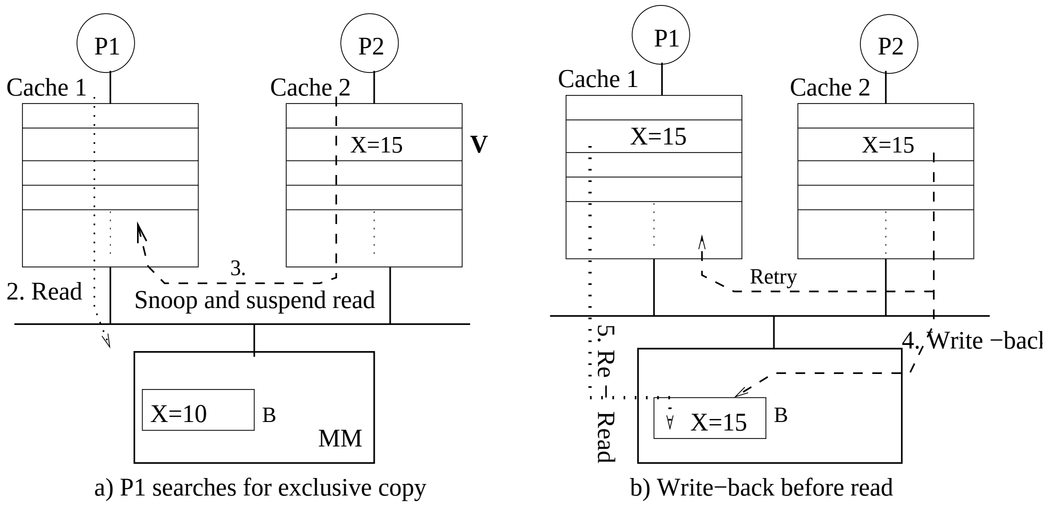


Figure 49: Snoopy protocol

This process of snoopy protocol is too slow.

It requires two memory latencies (Step 4 and 5) to move data from P_2 to P_1.

Following steps describe a better solution.

When P_1 encounters a cache miss for block B,

(i) P_2 indicates that it will supply B to P_1.

(ii) MM system decides not to supply B to P_1 and waits for P_2's data.

(iii) P_2 sends B on bus for write-back. P_1 snoops and gets B. MM is also updated.

A snoopy protocol based system can adopt any one of the following techniques.

Write-update or write-broadcast (WU)
Write-invalidate (WI)
Competitive-update (CU)

0.9 Cache Models

To update all cache blocks in write-update requires high bandwidth data bus.

On the other hand, write-invalidate scheme demands status updates of cached blocks.

If copy of block B in cache C$_i$ (of P$_i$) is in *valid or invalid* state and P$_i$ writes to B,

Then an updation message (in write-update) or

Invalidation message (in write-invalidate)

Is sent to bus even there may not be any taker.

That is, all other processors may not have a copy of B in their caches.

Different cache models are considered to avoid such unproductive update overhead.

A few among those are the MSI/MESI/MOSI/MOESI/ ...etc.

0.9.1 MSI

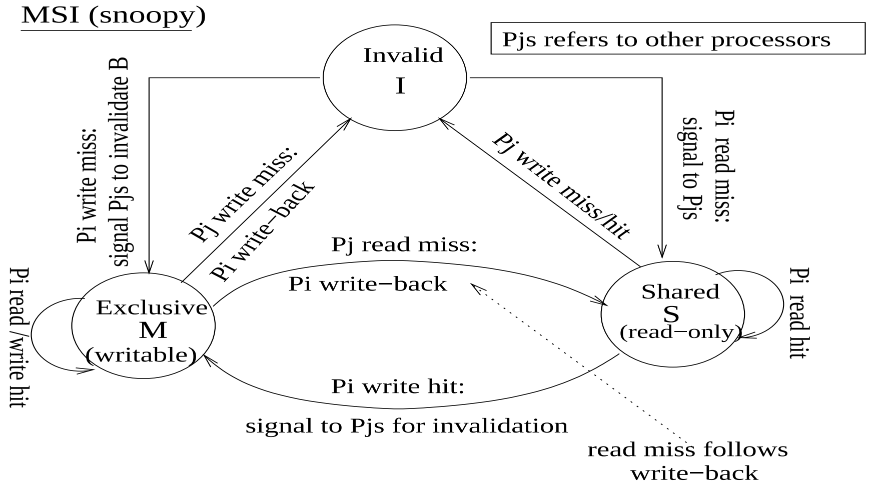
In MSI, each cached block B holds a cache state such as modified(M)/shared(S)/invalid(I).

A state defines how does cached block participate during cache read/write operation.

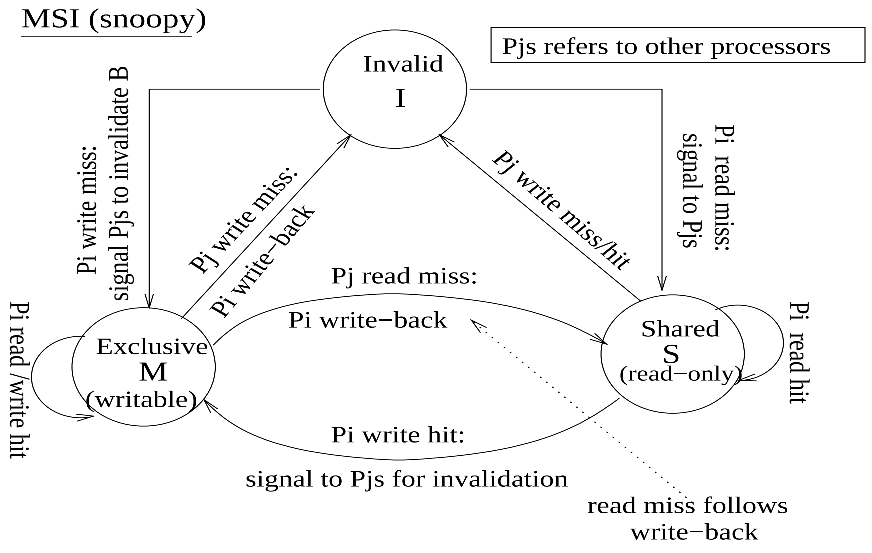
I: Invalid - one or more cached copies of B can be in I state simultaneously.

S: Shared - one or more cached copies of B can be in S state simultaneously.

M: Modified or dirty - only one cached copy of B can be in M state.


MSI (snoopy)
Pjs refers to other processors
Invalid
I
Pi write miss: signal Pjs to invalidate B
Pi read miss: signal to Pjs
Pj write miss: Pi write−back
Pj write miss/hit
Pj read miss: Pi write−back
Exclusive
M
(writable)
Shared
S
(read−only)
Pi read /write hit
Pi read hit
Pi write hit: signal to Pjs for invalidation
read miss follows write−back

MSI



MSI (snoopy)
Pjs refers to other processors
Invalid
I
Pi write miss: signal Pjs to invalidate B
Pi read miss: signal to Pjs
Pj write miss: Pi write−back
Pj write miss/hit
Pj read miss: Pi write−back
Exclusive
M
(writable)
Shared
S
(read−only)
Pi read /write hit
Pi read hit
Pi write hit: signal to Pjs for invalidation
read miss follows write−back

Figure 50: MSI state transitions for block B in processor Pi's cache

Figure 50 describes the state transitions of B in C_i in MSI.

State of B in C_i can be changed due to activities in P_j's cache on B.

Example

If cached copy of B at C_i is in S, then a write to B at C_j changes status of B at C_i.

QUIZ