Lecture 19-21: September 20 and 22, 2021
Computer Architecture and Organization-II
Biplab K Sikdar

Example MSI

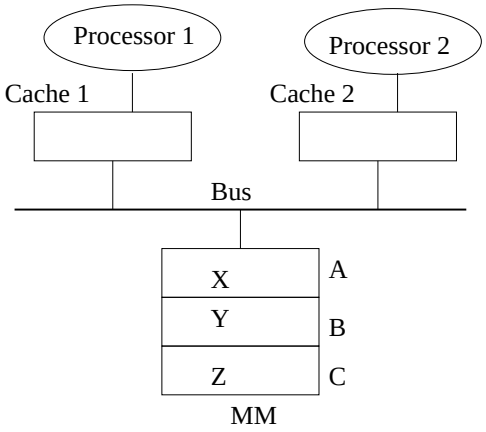


Figure 51: Example MSI

Figure 51: With two processors - follows write-invalidate scheme and MSI model.

Each processor cache can fit only one block at a time.

Shared memory MM in divided into three blocks A, B and C.

A contains variable X, Y is in B, and Z is in C.

Initial values of variables in MM are

X	Y	Z
10	20	40

Assume caches are initially empty and following events are occurred.

(1) P_1 reads X, (2) P_2 reads X, (3) P_2 updates X=X+50,
(4) P_1 reads Y, (5) P_2 reads X, (6) P_1 updates X=X-20

a) Show state of cached copy of A after each event (operation).

—

b) Find cache misses for block A after each event.

—

Limitations

In MSI, modifying B require 2 bus transactions even there is no other sharer of B.

Example: Assume a system with processor P_1 and P_2 (Figure 52).

Initially there is no cached copy of B.

Let P_1 modifies B.

Now, following seq events are occurred.

1. *P_1 tries to read B*: It is a read miss. B is transferred from MM to C_1.

   This is a bus transaction. State of B at C_1 is modified from I to S.

2. *P_1 writes to B*: It is a write hit. State of B at C_1 is modified from S to M.

   P_1's cache controller sends invalidation signal to P_2.

   It is also a bus transaction although P_2 does not have valid cached copy of B.

   This bus transaction is unnecessary.

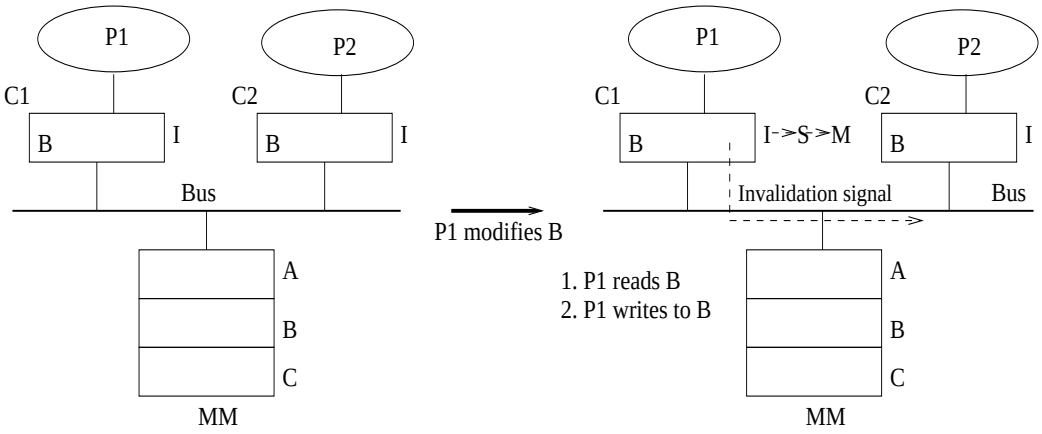


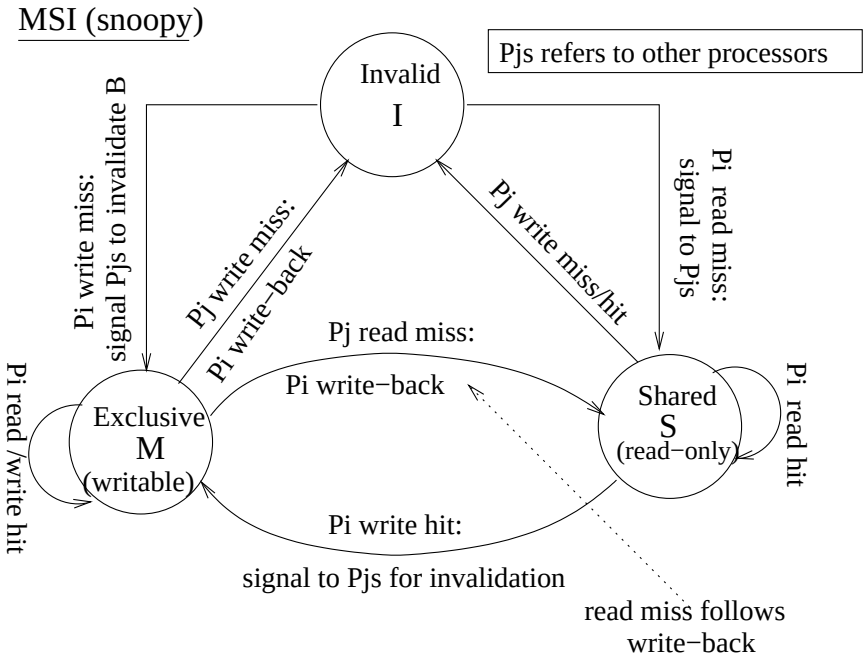
Figure 52: Bus transactions in MSI for modifying a block

Limitation of MSI is due to - state S does not have write privilege.

That is, a write to cached copy of B in S state is followed by state transition S→M.

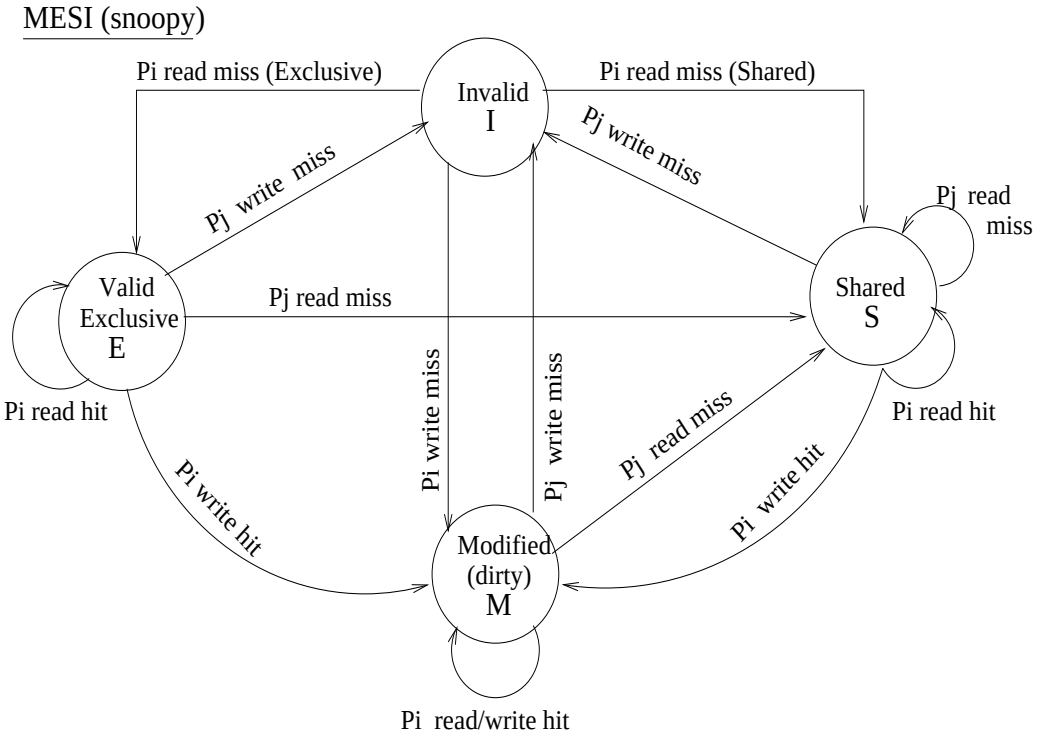
This transition is followed by invalidation signal even if there is no other sharer.

To address this, a fourth state, E (exclusive) is defined in 4-state MESI model.



0.9.2 MESI 4-state

The states in MESI model are - M, E (Exclusive), S, and I.



I: B is in I state signifies - other cached copies can be either in M/E/S/I state.

M: B is in M signifies - only valid (dirty) copy and different from MM copy.

S: B is in S indicates - this is a valid copy and at par with MM copy.

Other cached copies of B can either be in S/I state.

E: B is in E defines - this is the only cached copy and at par with MM copy.

Other cached copies of B are declared as I.

Cache block in E is free to modify without a bus transaction.

(That is, no invalidation message is generated).

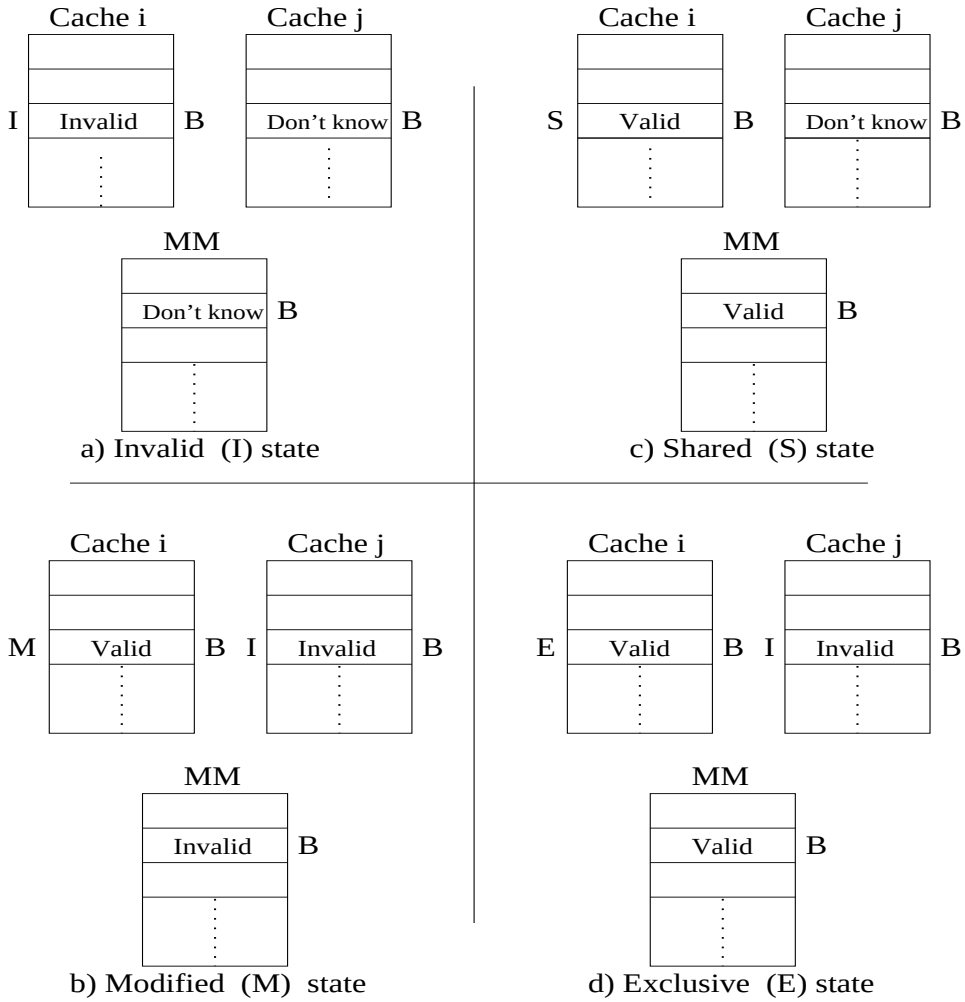


Figure 53: MESI states

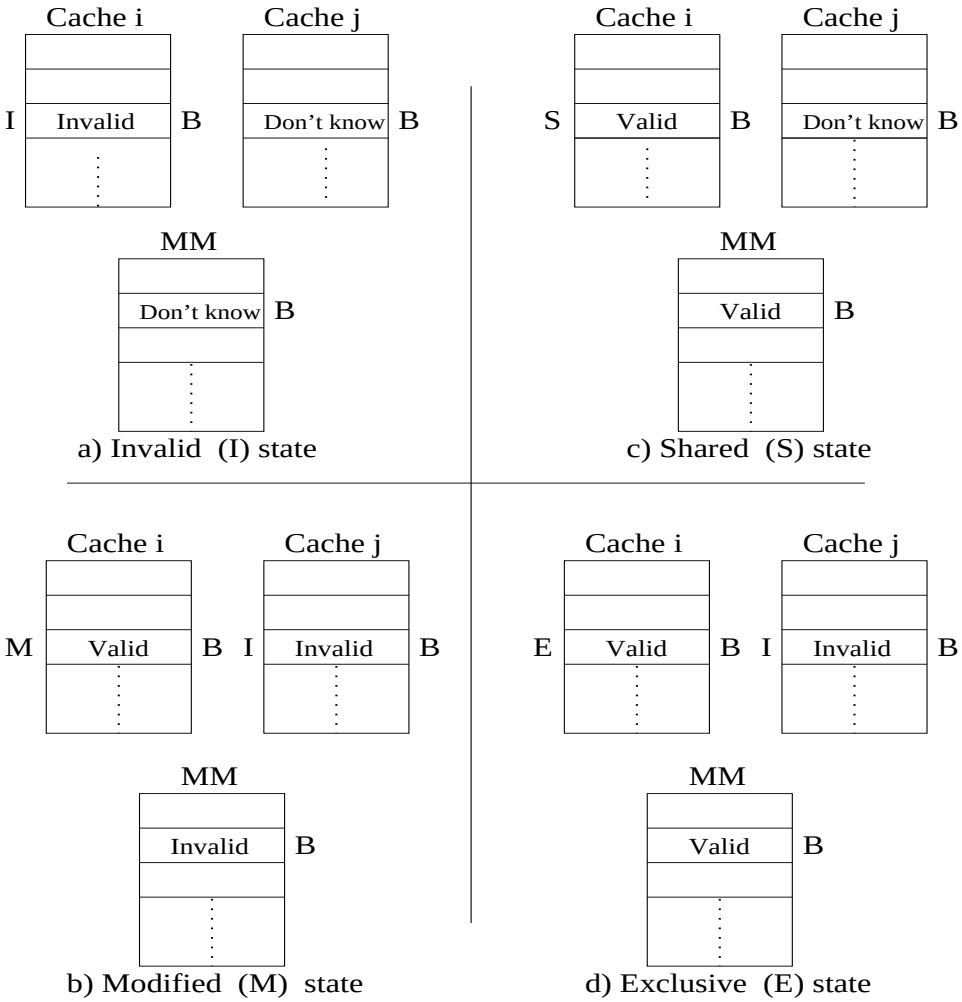
I: Cached copy of B at C_i (of P_i) is in I state signifies

The other cached copies of B can be either in M/E/S/I state (Figure 53 (a)).

B in MM can be either a valid or invalid copy.

M: Copy of B in C_i is in M signifies - it is the only valid (dirty) copy of B.

It is different from MM copy (Figure 53 (b)).



S: A cache line B in C_i is in S indicates - this is a valid copy of B.

It is at par with MM copy of B.

Other cached copies of B can either be in S/I state (Figure 53 (c)).

E: Cache line B in C_i is in E defines - this is the only cached copy of B.

It is at par with MM copy of B (Figure 53 (d)).

Other cached copies of B are declared as I.

Cache block in E is free to modify without a bus transaction.

(That is, no invalidation message is generated).

The summary:

For a pair of caches C_i (of P_i) and C_j (of P_j), following relations hold.

$B\ in\ C_i$	$B\ in\ C_j$	$B\ in\ MM$
M	I	$invalid$
E	I	$valid$
S	$S\ or\ I$	$valid$
I	$M/E/S/I$	$valid/invalid$

Example MESI

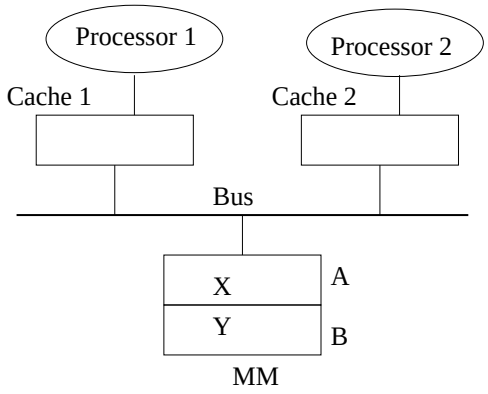


Figure 54: Example MESI

Consider Figure 54. It follows write-invalidate scheme and MESI model.

Each processor cache can fit only one block at a time.

MM in divided into two blocks A and B. A contains variable X and Y is in B.

Assume caches are initially empty and the following events are occurred.

(1) P_1 reads X, (2) P_2 reads X, (3) P_2 updates X,

(4) P_1 reads Y, (5) P_2 reads X, (6) P_1 updates X

a) Show state of cached copy of A after each event (operation).

___

b) Find the cache misses for block A after each event.

___

The details of 4-state MESI state transitions are shown in Figure 55.

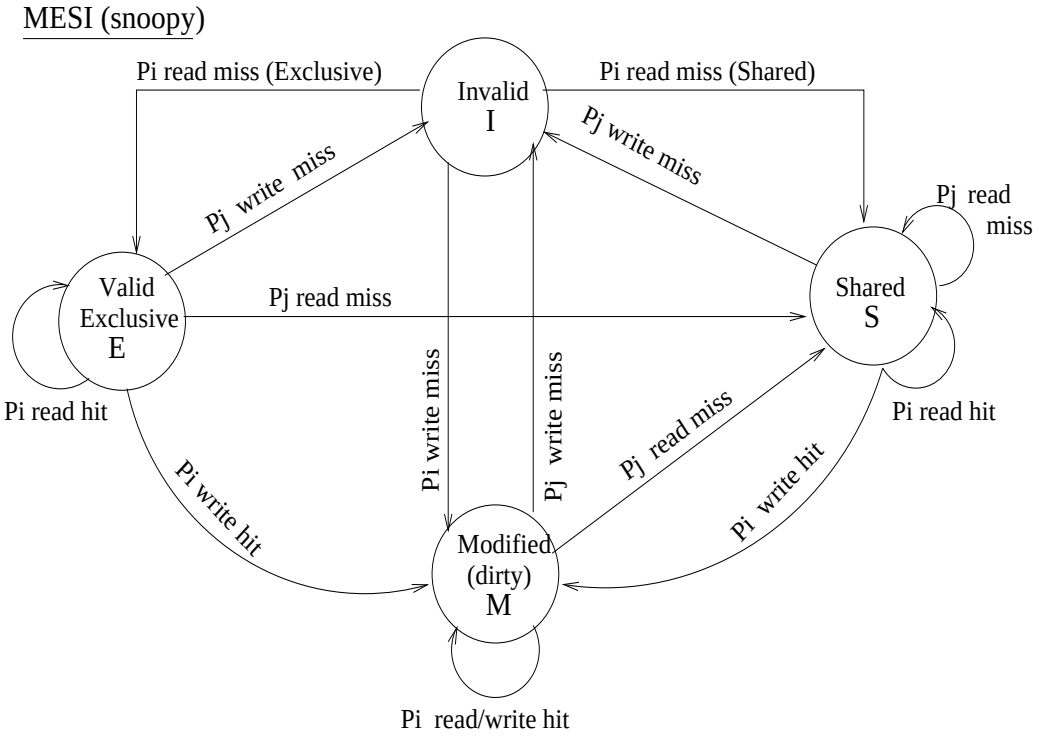


Figure 55: MESI state transitions for block B in processor P_i's cache

Cached block B can change its state from I to E on a read miss for block B,

When B at some other processor's cache C_j is not in S/M/E state.

On the other hand, a state change from I to S can occur on read miss,

When some other processor's cache has a copy of block B in S/M/E state.

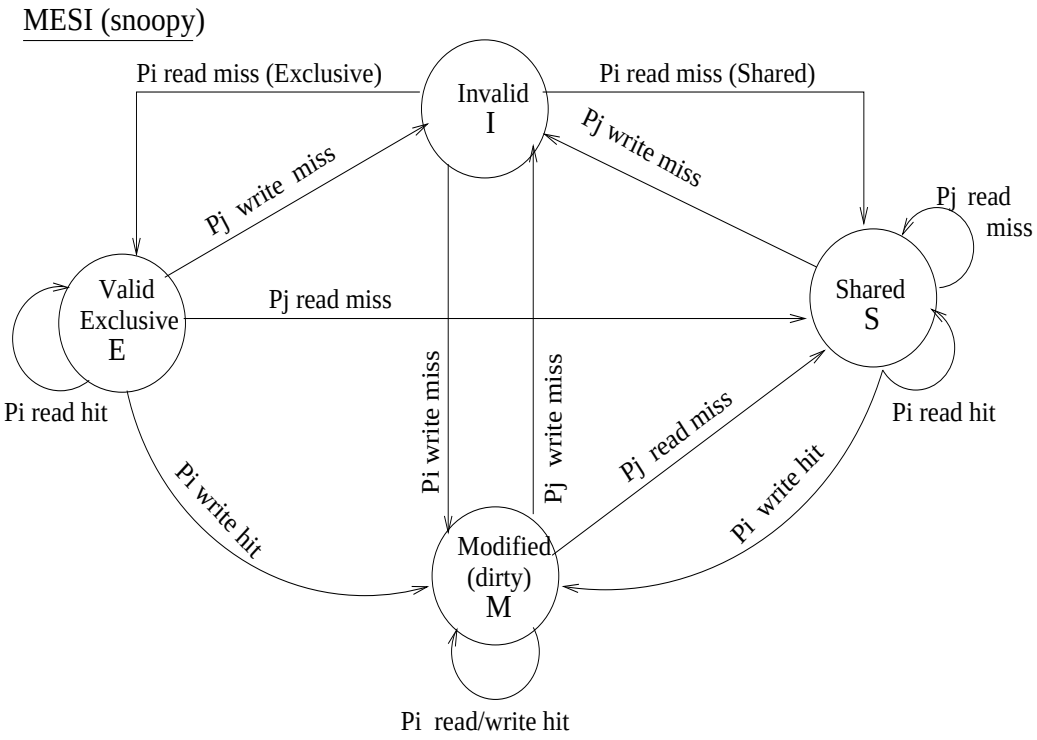
State M/E of other processor's cached copy is then changed to S.

MESI is popularly implemented cache model.

Limitation of MESI model:

In MESI, modifying a copy (B) in S requires bus transaction as in MSI.

It implements write operation of dirty cached copy of B, in M state at C_i of P_i,
to MM when a processor (P_j) tries to read B.



This is resolved in MOESI through introduction of *owned* state.

0.9.3 MOESI cache model

Owned state

B in C_i is in *owned state* (O) means C_i has most recent and valid copy of B.

Other caches C_js can also have same (most recent and valid) copy of B.

These are in shared state (S).

That is, it allows dirty sharing.

MM copy can be stale (invalid).

Only one processor can hold the data in the owned state.

In MOESI, state S is marginally different than that it is MSI/MESI.

If cache C_j holds B in O state and C_i holds it in S state, then MM copy can be stale.

Following relations hold for a block B in MOESI model.

$B\ in\ C_i$	$B\ in\ C_j$	$B\ in\ MM$
O	S/I	$invalid/valid$
M	I	$invalid$
E	I	$valid$
S	$O/S/I$	$valid/invalid$
I	$O/M/E/S/I$	$valid/invalid$

If P_k demands B (not in C_k), C_i with B in state O only responds (snoops).

Owned state allows P_i to supply B directly to P_k (cache to cache transfer).

This results in fewer write-back.

Limitation

If C_i's copy of B is in shared state S and is clean (that is, MM is a valid copy),

It cannot be quickly sent to C_k in MOESI (it is as in MESI).

Beacause when C_k demands B, it gets B from MM.

That is, cache-to-cache transfer is lost out.

This drawback of MOESI is avoided in MESIF.

0.9.4 MESIF

I skip.

0.10 Directory Based Protocol

Cache read/write at P_i affects status in cache at P_j, placed even at a long distance.

In snoopy, update is done through broadcasting.

Broadcasting demands heavy network bandwidth.

In a system with large number of processors (CMPs), broadcasting is expensive.

A directory based protocol

Issues coherence commands only to cached copies affected by an update.

A directory consists of a set of vectors (*sharing vectors*).

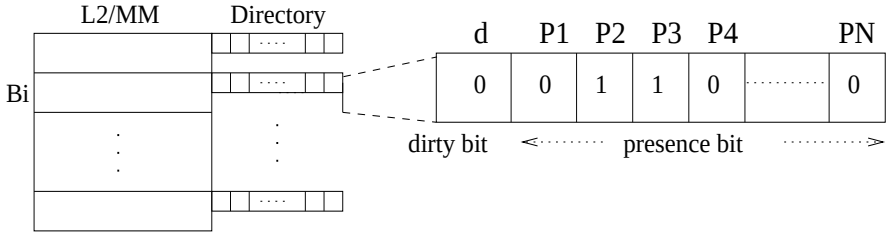


Figure 56: Sharing vector

For each block B_i in memory (MM), there is one sharing vector V_i (Figure 56).

Length of it is N+1, where N is the number processors in the system.

First bit (d) is dirty bit.

d = 1 implies - there is a cached copy of B_i with latest update, MM copy is stale.

Each of rest bits V_{ij} corresponds to processor P_j.

V_{ij} = 1 implies - P_j has cached copy of B_i.

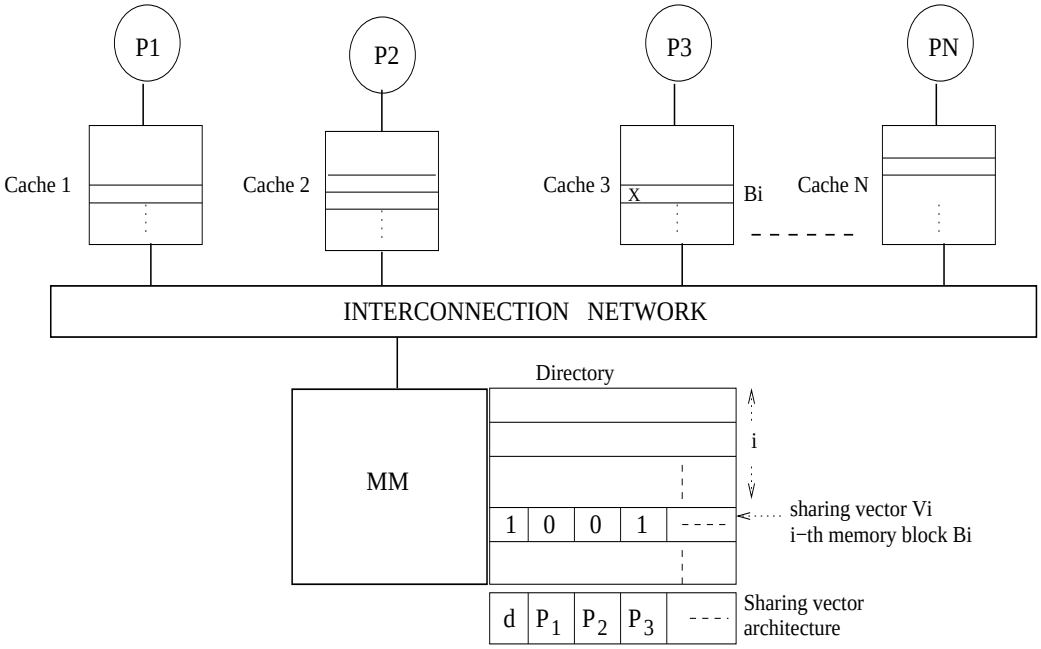


Figure 57: Directory based protocol: full directory

Directory based system architecture is shown in Figure 57.

If P_1 writes to B_i in its cache (Cache 1), V_i is checked.

Here, B_i is in processor P_3's cache (Cache 3) only.

First bit of V_i is 1 (that is, d = 1). That is, MM copy of B_i is stale.

4^{th} bit is 1 - signifies owner of B_i is P_3.

Read miss

Figure 58 describes steps performed for a read miss.

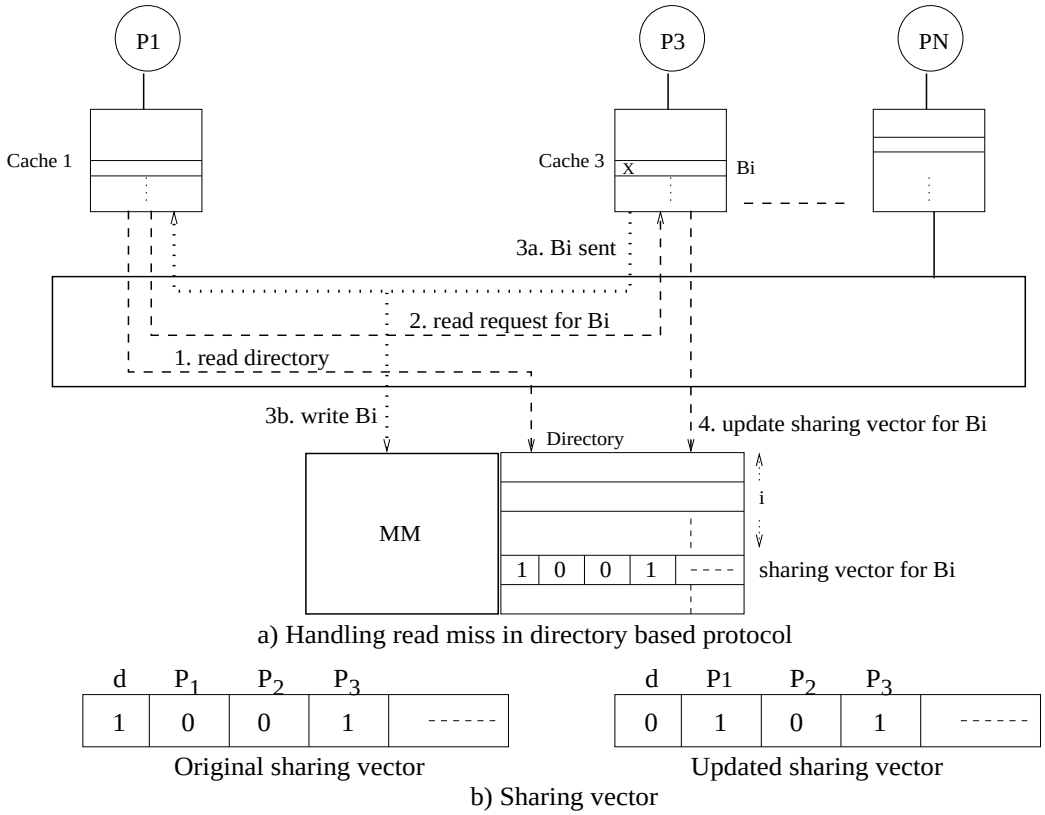


Figure 58: Directory based protocol: full directory: read miss

1. If P_1 has read miss on B_i, it sends read request for directory and reads V_i.
2. If d of V_i is 1 (MM copy is not valid copy) & P_k (P_3) = 1 in V_i,

   Then read request for B_i goes to P_k (P_3).
3. P_3 (a) sends B_i to P_1 and (b) writes back B_i to MM.
4. P_3 updates V_i (Figure 58(b)). It clears dirty bit d and sets P_1 as 1.

In step 2, if d = 0, then P_1 gets B_i from MM. V_{i1} is set to 1.

Directory based system of Figure 58 maintains central directory.

Sharing status of all MM blocks is maintained in one location. This is a bottleneck.

0.10.1 Distributed directory

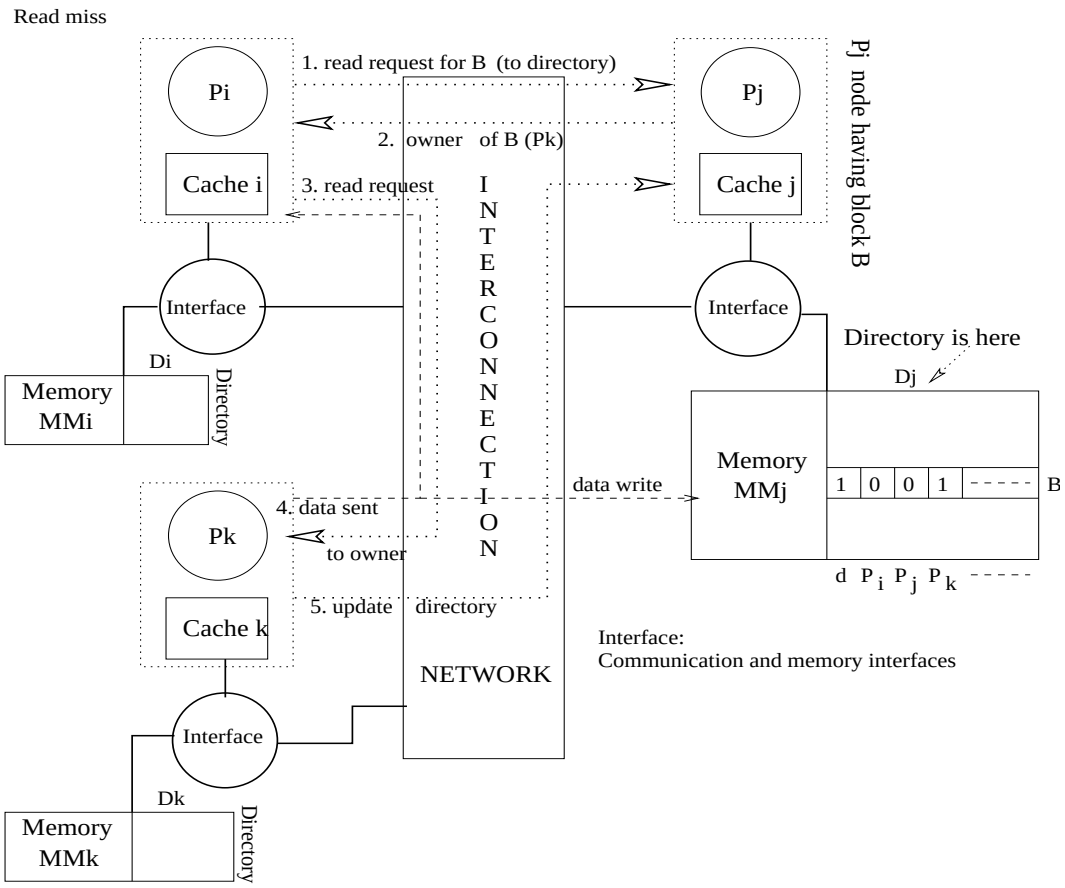


Figure 59: Distributed directory based protocol: read miss to a block

For each memory module MM_i, local to P_i has a directory D_i.

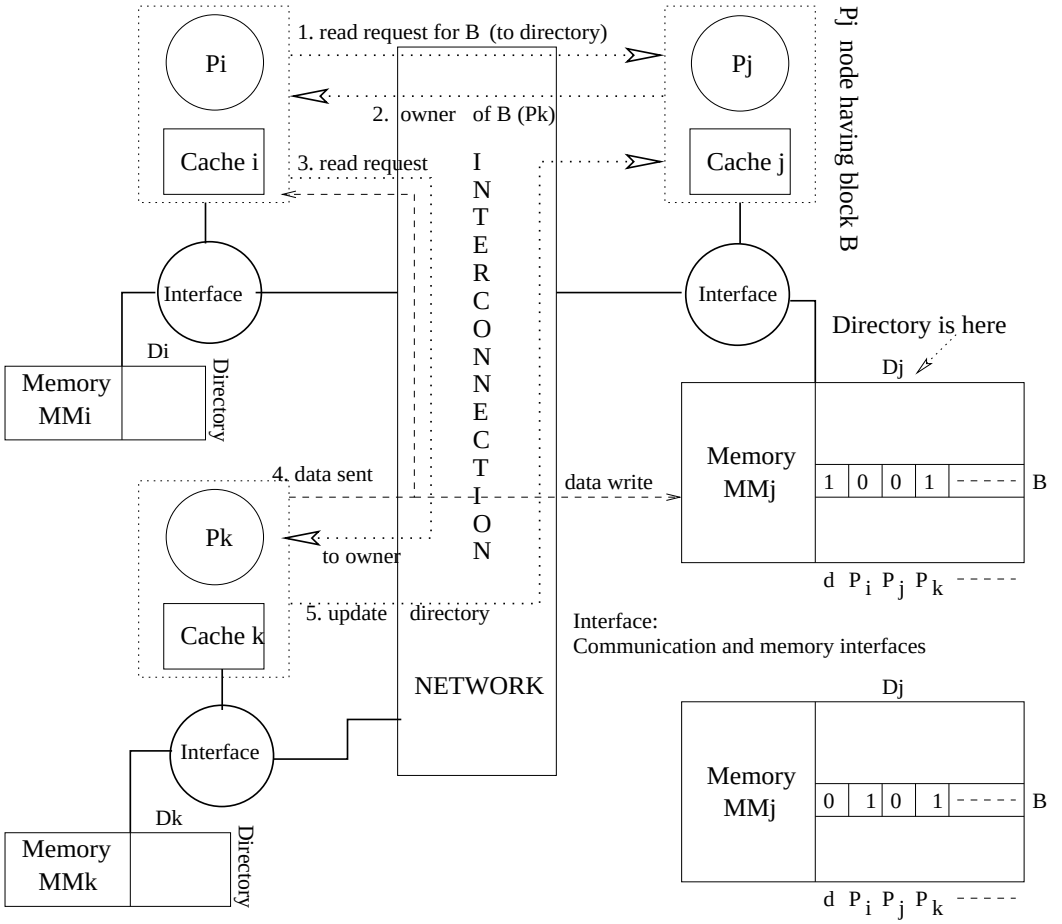
D_i maintains sharing vector for each block B of MM_i.

Figure 59 describes distributed directory based protocol.

It has processors P_i, P_j, P_k, ... with memory modules MM_i, MM_j, MM_k, ...

Distributed directories D_i, D_j, D_k, ... are local to P_i, P_j, P_k, ...

Read miss cont.



(a) Read miss to a block B (b) Updated directory at Pj for block B

If P_i has a read miss on block B non local to P_i (not in MM_i),

Consults its communication unit and finds home of B (say P_j -that is, MM_j).

The request goes to P_j's site. D_j at P_j is consulted.

If in sharing vector V of B (stored in D_j) d = 0, P_j retrieves B and sends it to P_i.

Then updates directory D_j.

If d = 1, and P_k is having dirty copy (shown in figure), P_k sends B to P_i.

Also writes to MM_j. Updates directory D_j.

Write miss

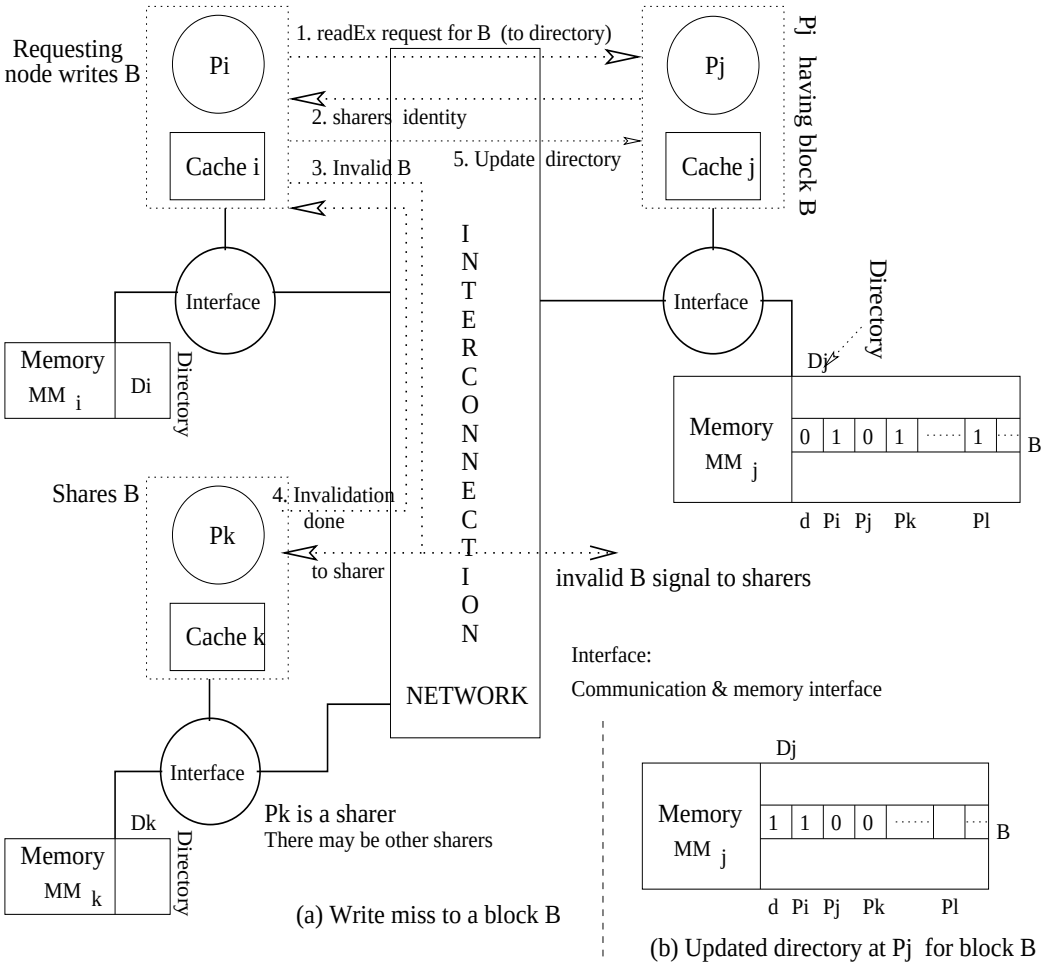


Figure 60: Distributed directory based protocol: write miss to a block

1. P_i issues write request for B. It then request for directory (D_j, home of B).
2. Vector for B, from D_j, is sent to P_i. Sharers are P_k/others (Figure 60(a)).
3. On receiving sharers identity, P_i sends invalidation signal to sharers (P_ks).
4. On receipt of invalidation message each sharer (P_k) invalidates

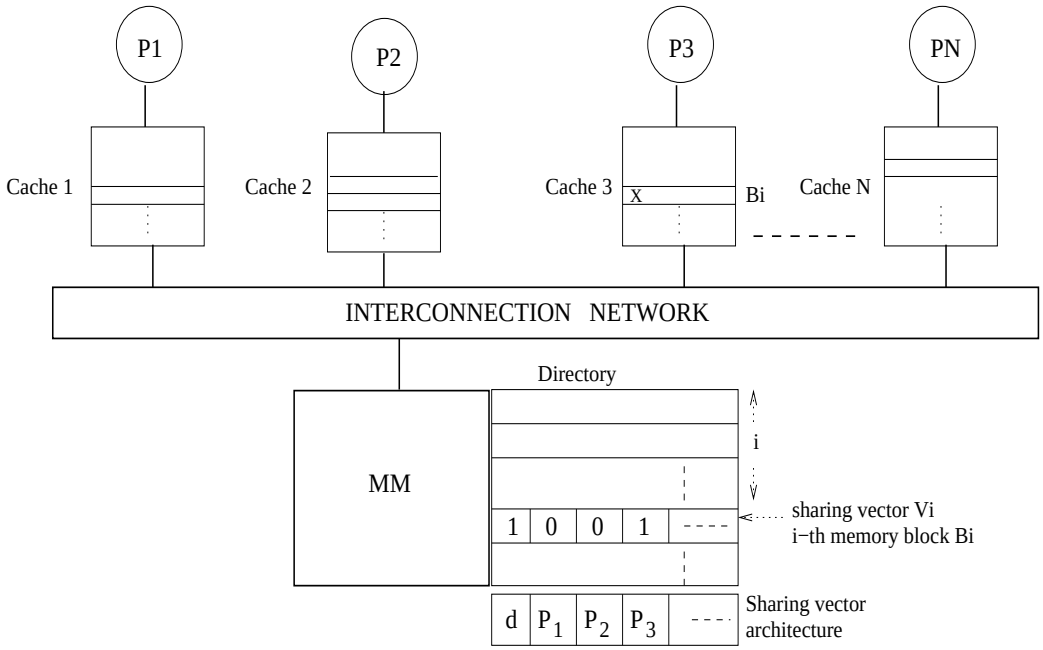
Its cached copy of B and sends ack signal ”invalidation done” to P_i.

5. P_i then requests P_j to update sharing vector for B at D_j (Figure 60(b)).

0.10.2 Directory based protocol category

Directories (for example in Figure 57) consume large storage.

These store full directory.



Directory based system

Information stored in a full directory is proportional to

total number of memory blocks in the system $\times$ *number of processors*

Ignoring the dirty bit.

For a system with thousands of processors, full directory is not cost effective.

Following are the different structures of directory.

1. Full directory,
2. Limited directory,
3. Chained (linked) directory.

0.10.3 Full directory

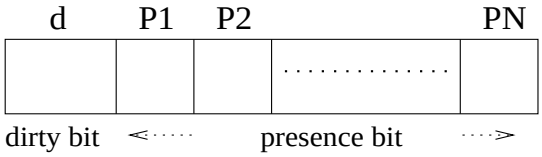


Figure 61: Full directory

Space in directory is under utilized in full directory when sharer of block is few.

Full directory needs changes in directory size/bus width for expansion of system.

That is, full directory is not scalable.

0.10.4 Limited directory

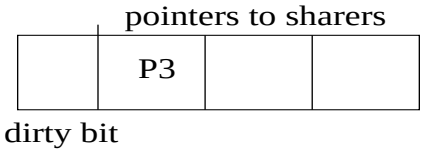


Figure 62: Directory entry of block in limited directory

Sharing vector for block B, in a limited directory, is in Figure 62.

Number of pointers to processors is assumed 3 (pointers to sharing processors).

That is, a block can be shared by at most 3 caches at an instant of time.

Content of Figure 62 indicates that block B has a copy in Cache 3.

Each vector needs $3 \times log_2(number\ of\ processors)$ bits for pointers.

Total storage requirement for directory (ignoring the dirty bit) is

total number of memory blocks in the system $\times\ 3 \times log_2(number\ of\ processors)$.

0.10.5 Chained directory

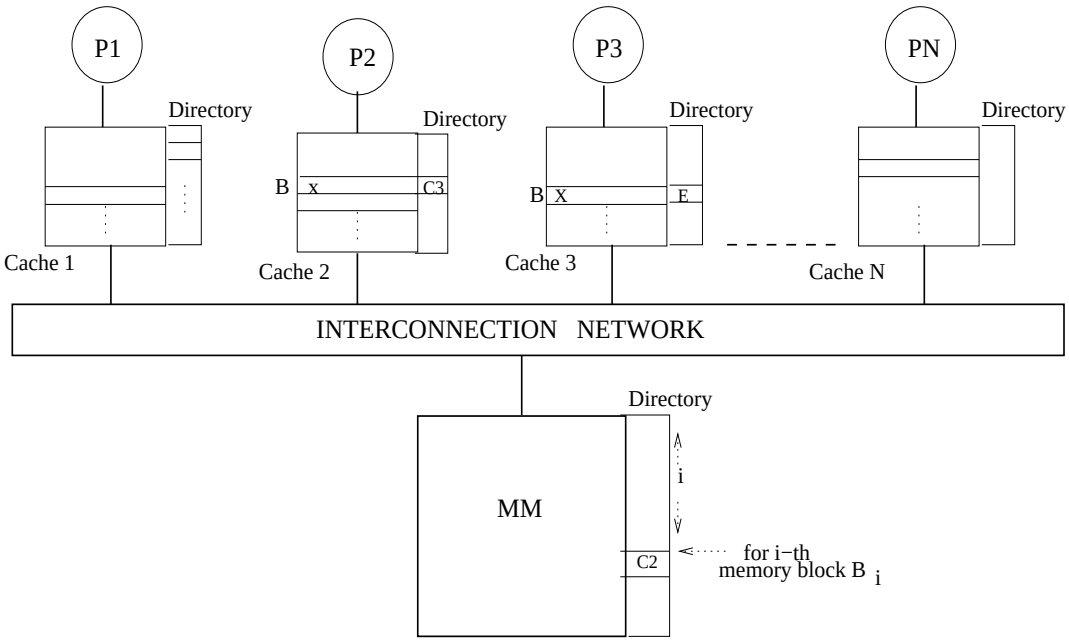


Figure 63: Directory based protocol: chained directory

Figure 63 describes a chained directory architecture.

In chained directory, the directory is distributed among processor caches.

When number of sharers of block B at an instant of time is few but not limited

Then chained directory architecture offers better design.

Entry C_2 in MM directory signifies - recent copy is in Cache 2.

Sharing vector entry for B_i at C_2 points to sharer Cache 3.

E in Cache 3's directory denotes - there is no other sharer of B_i.

An entry in sharing vector (pointer to the processor) of chained directory needs

$$log_2(number\ of\ cachesorprocessors)$$

Therefore, in a chained directory based system, total number of required bits is

(number of MM blocks + number of cache lines in all processors) $\times log_2(number\ of\ processors)$

0.10.6 Scalable coherent interface protocols

I skip.

0.11 Cache coherence in CMPs

I skip.