Cache Coherence in SMP

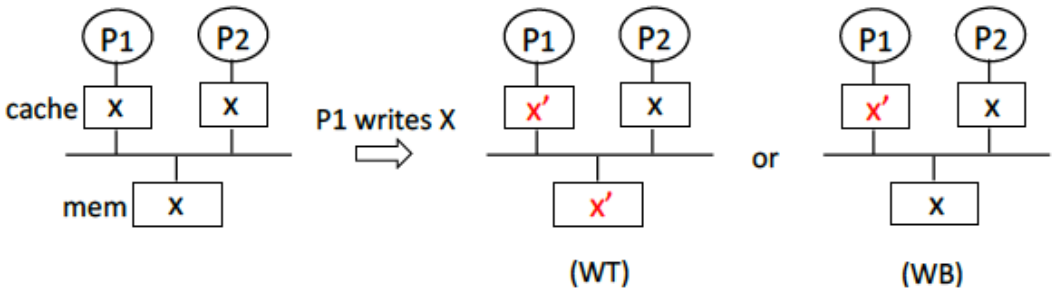
Processors may see different values through their caches.
Copies of the same information item should be consistent at different memory levels.

ex) with WT cache:

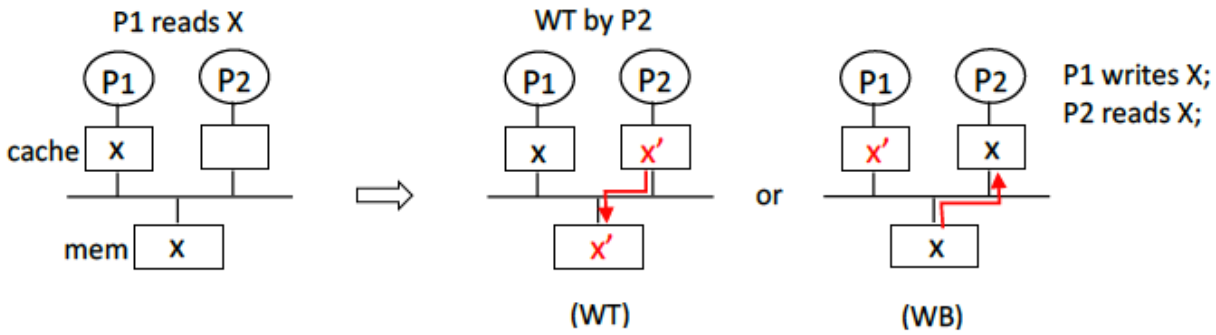
Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

All reads by any processor must return the most recently written value.

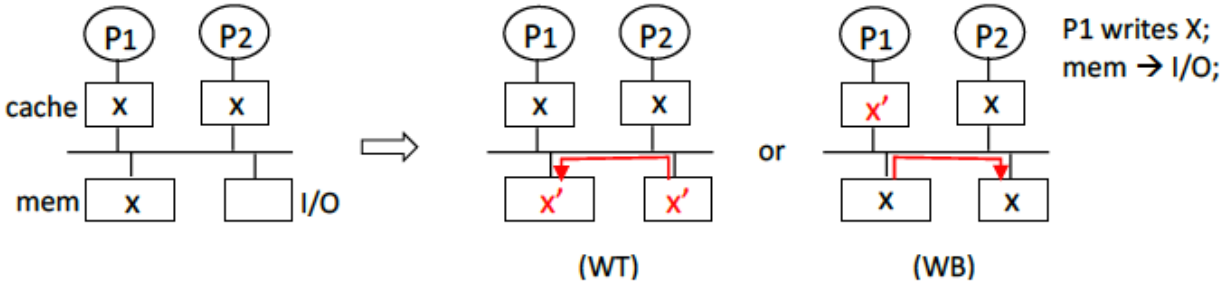
- Sources of cache incoherence:
  1. sharing of writable data (written by different processors asynchronously);



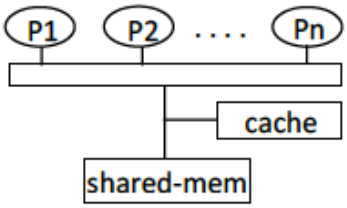
  2. process migration (e.g., P1 → P2);



3. by I/O operation bypassing the cache (e.g., DMA);



- Solutions to cache incoherence:
  - ➢ common cache – simplest, but limited cache size, no private cache;



  - ➢ snoopy bus – HW support with snoopy cache controller;

    based on I/s seen on the bus, controls status of data in the cache;

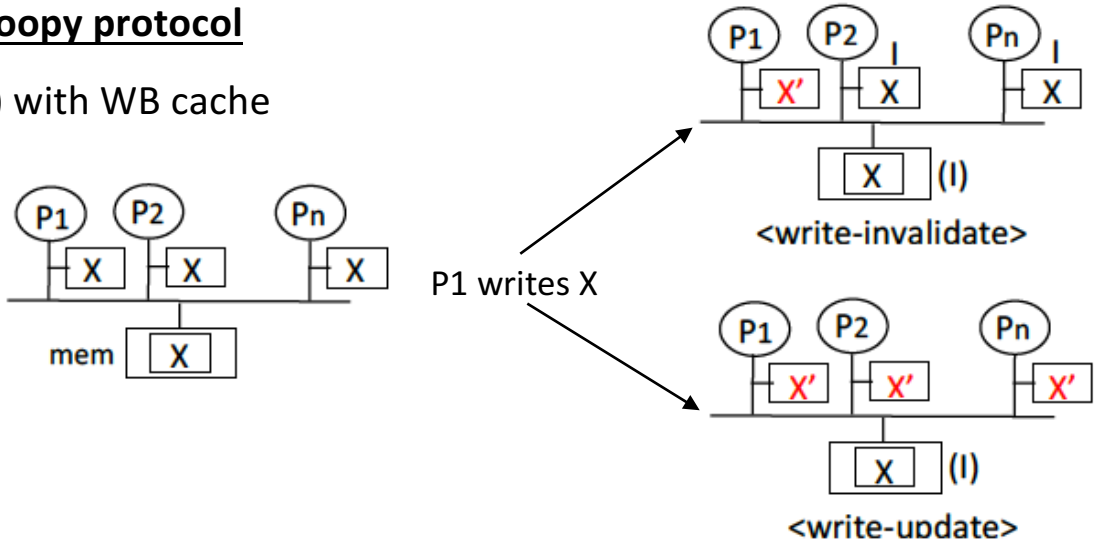
    two variations: write-invalidate – commonly used, low cost;

    write-update – high degree of coherence, expensive;

  - ➢ directory-based (DSM)

Snoopy protocol

ex) with WB cache



MESI snoopy protocol

a write-invalidate method

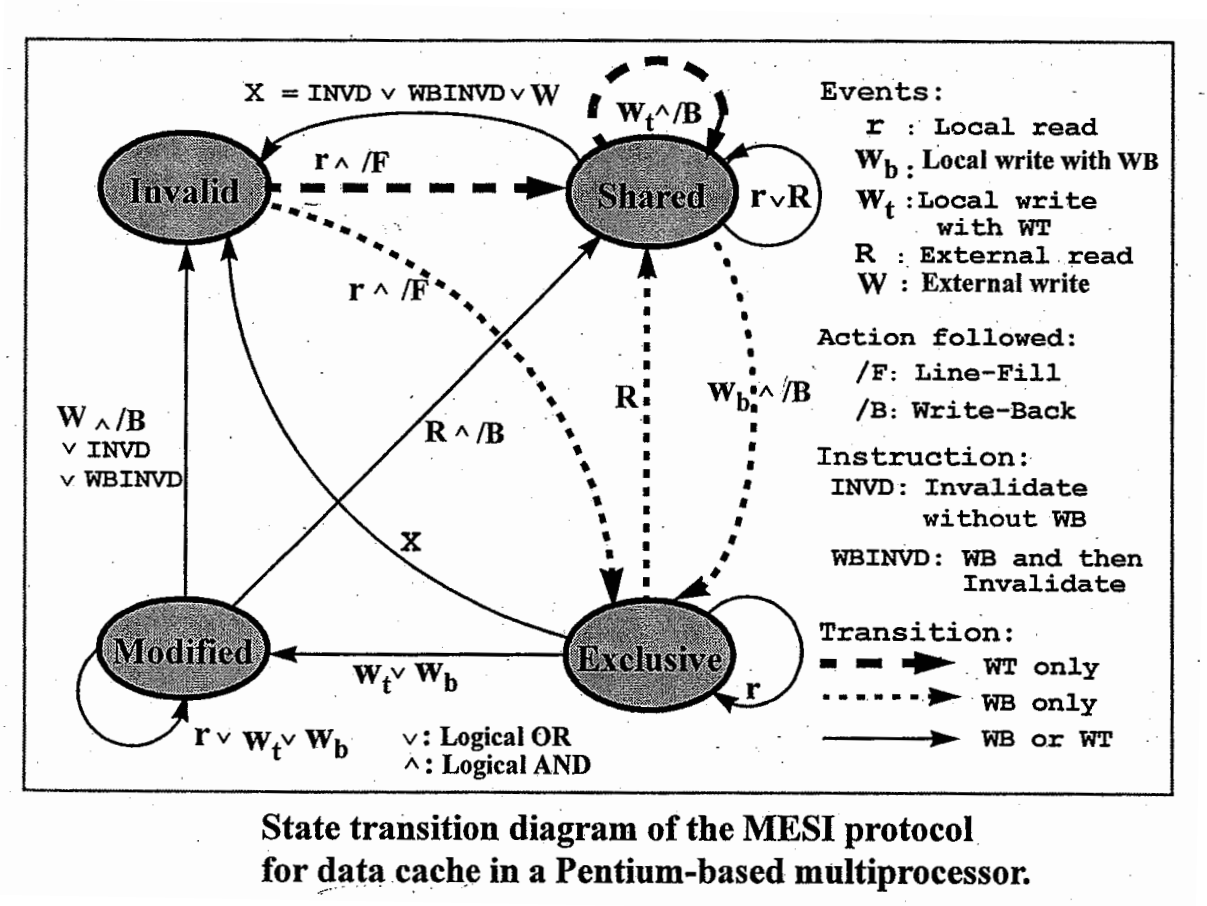
on a write miss, uses no-write-allocate

each cache line (block) can be

- M (modified) state
- E (exclusive) state
- S (shared) state
- I (invalid) state

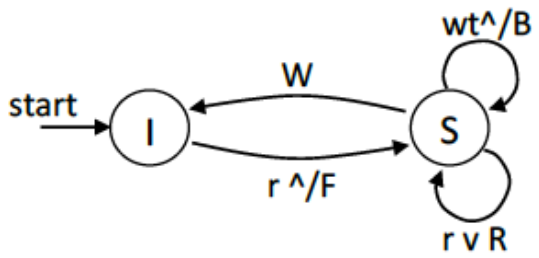
WT cache – SI protocol (using only S and I);

WB cache – MESI protocol;



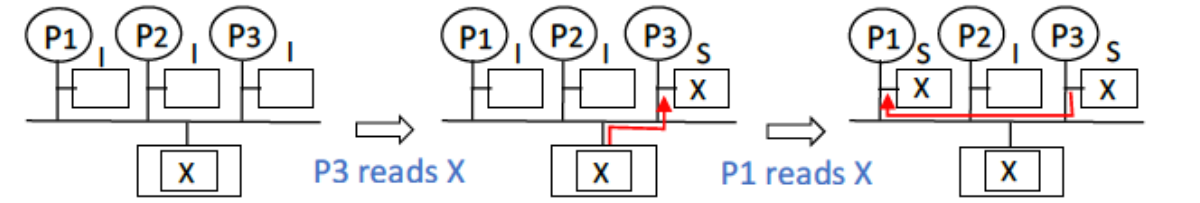
State transition diagram of the MESI protocol for data cache in a Pentium-based multiprocessor.

SI protocol (WT cache only)



W – external write
wt – local write-through
R – external read
r – local read
/B – write back (update mem)
/F – cache line fill

ex)

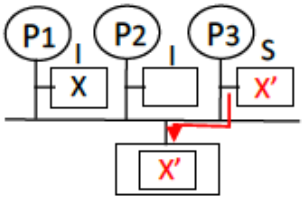


initially, all caches are vacant (I);

X in C3 – shared with mem;

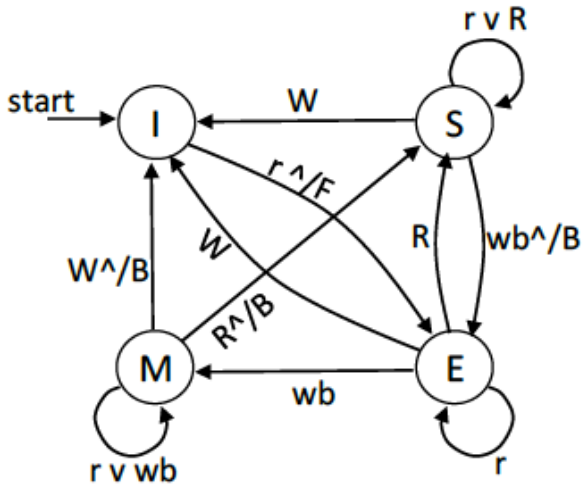
C1 gets X from C3; X in C1 – shared;

P3 writes X



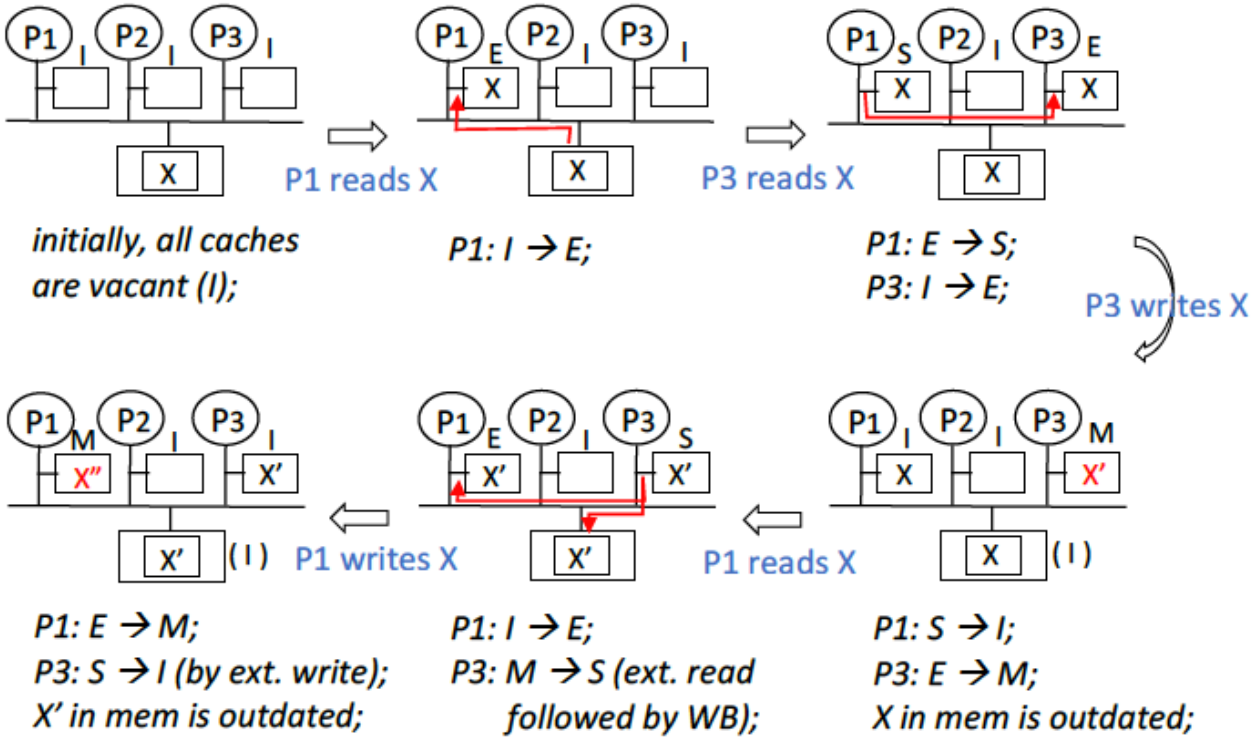
X in C3 – shared with mem;
X in C1 – invalid by C3's external-W;

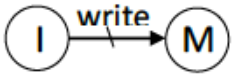
MESI protocol (WB cache only)



W – external write
wt – local write-through
R – external read
r – local read
/B – write back (update mem)
/F – cache line fill

ex)



- ∄ direct transition 

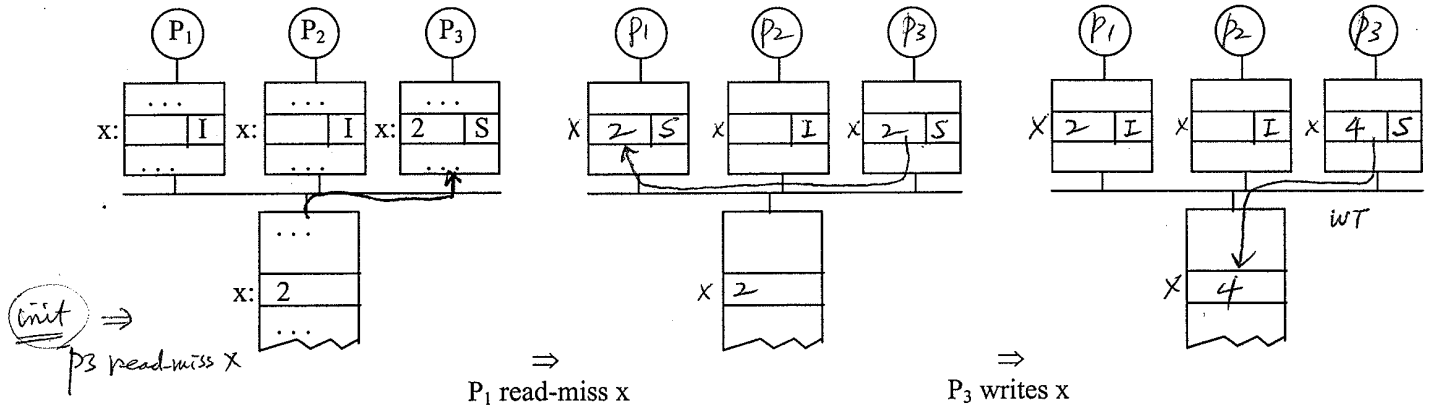
in MESI protocol.

→ when write-miss, no-write-allocate is used; so, no cache update from I.

ref) P1 reads X (C1: E); then, P1 writes X (C1: M – write hit);

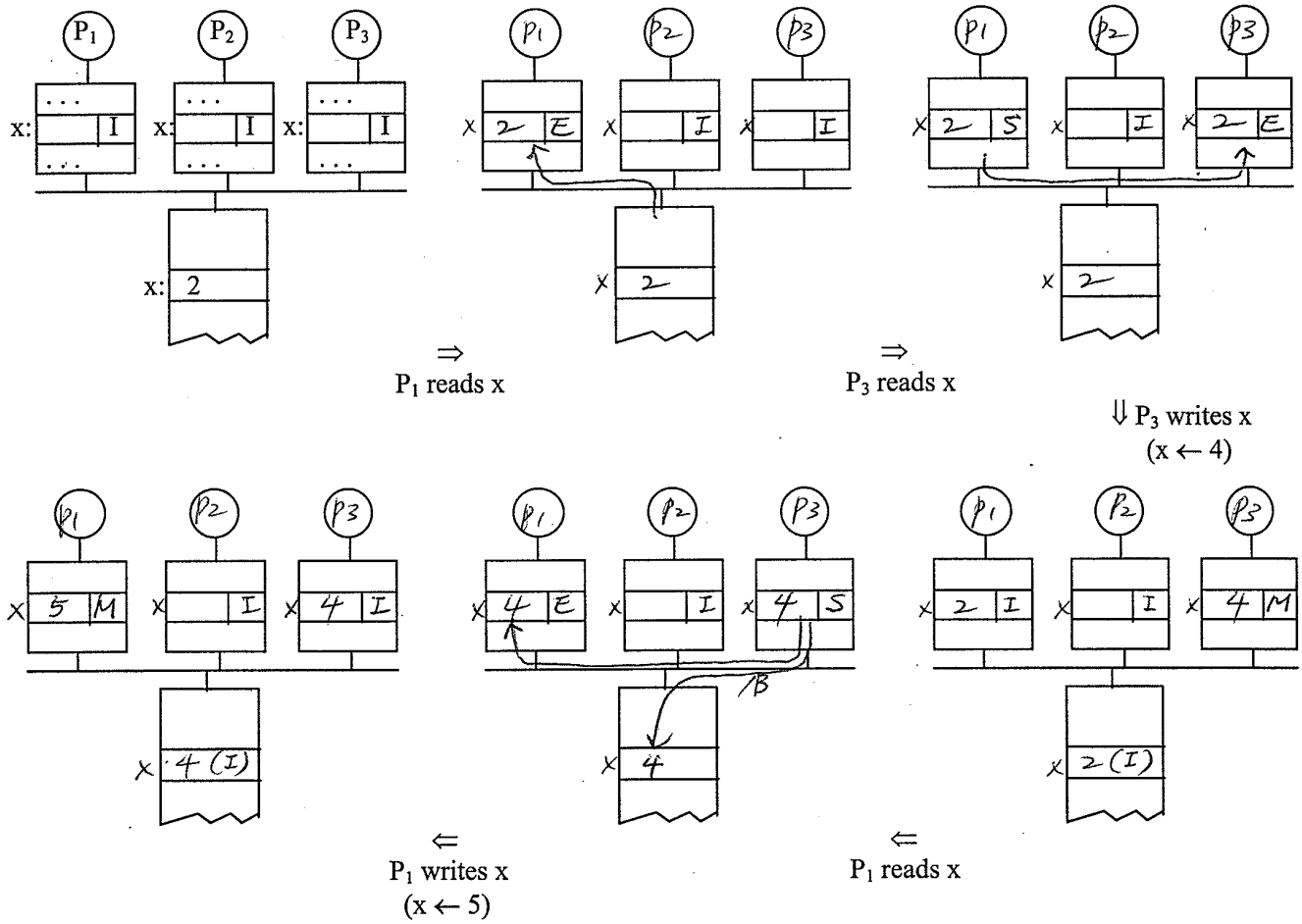
MESI Snoopy protocol

1. with Write-Through cache (SI protocol)



---

2. with Write-Back cache (MESI protocol)
(assume: on a write miss, no-write-allocate)

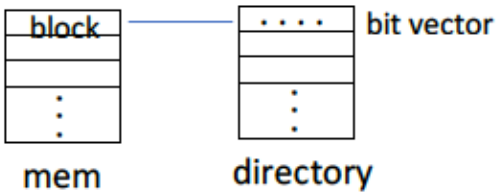


Directory-based cache coherence control in DSM

Directory keeps track of every block:

which caches have each block;

dirty status of each block;



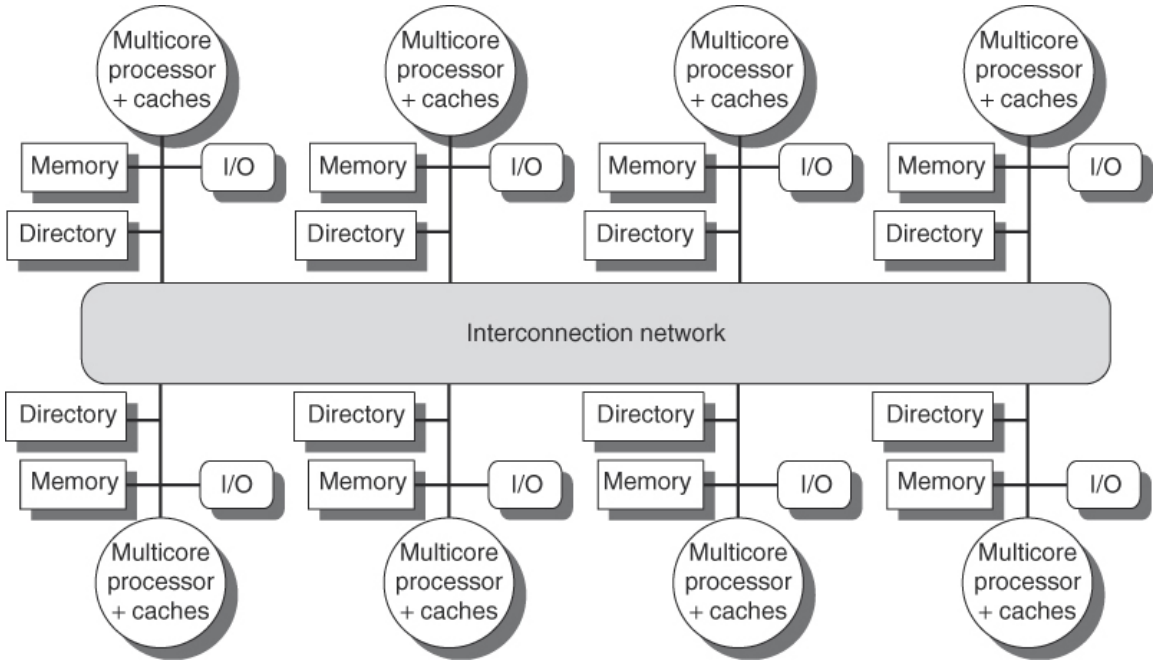


Figure 5.20 A directory is added to each node to implement cache coherence in a distributed-memory multiprocessor. In this case, a node is shown as a single multicore chip, and the directory information for the associated memory may reside either on or off the multicore. Each directory is responsible for tracking the caches that share the memory addresses of the portion of memory in the node. The coherence mechanism would handle both the maintenance of the directory information and any coherence actions needed within the multicore node.

Distributed directory (a directory for each node)

- sharing status of a (memory) block is always in a single known location;
- directory is used for tracing the state of each (memory) block;

state can be
- uncached – no processor has a copy;
  memory is up-to-date;
- shared – one or more processors have copy;
  memory is up-to-date;
- modified – exactly one processor has a copy;
  memory is out-of-date;

In the directory, a bit vector is used for each (memory) block.

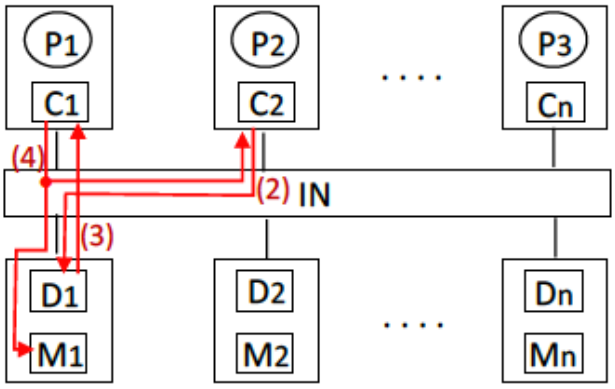
ex) 0 1 1 0 0 1 0 . . . .

shared *P0 has copy* *P3 has copy*

A node is:

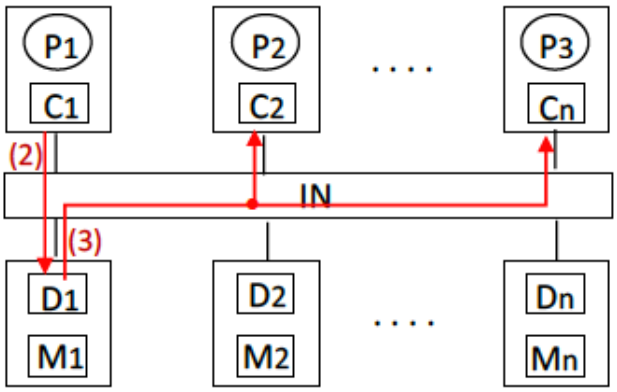
- local node – requesting (r/w) node (can be home node);
- remote node – a third node having copy;
- home node – having the memory block;

Basic idea of directory-based cache coherence scheme



ex) Read-miss case

1. Assume: read-miss at C2;
2. goto D1 (since M1 has data);
3. D1 indicates that C1 has a clean copy;
4. C1 (actually, mem. controller for C1) updates M1; sends copy to C2;



ex) Write-miss case

1. Assume: write C1;
2. mem. controller searches D1 (since M1 has data);
3. invalidate all marked caches in D1;

- Advantages of directory-based protocol:

  no broadcasting (snooping is used) – only caches having copy are invalidated;

  works on any IN for DSM;

  forwarding requests to other nodes;