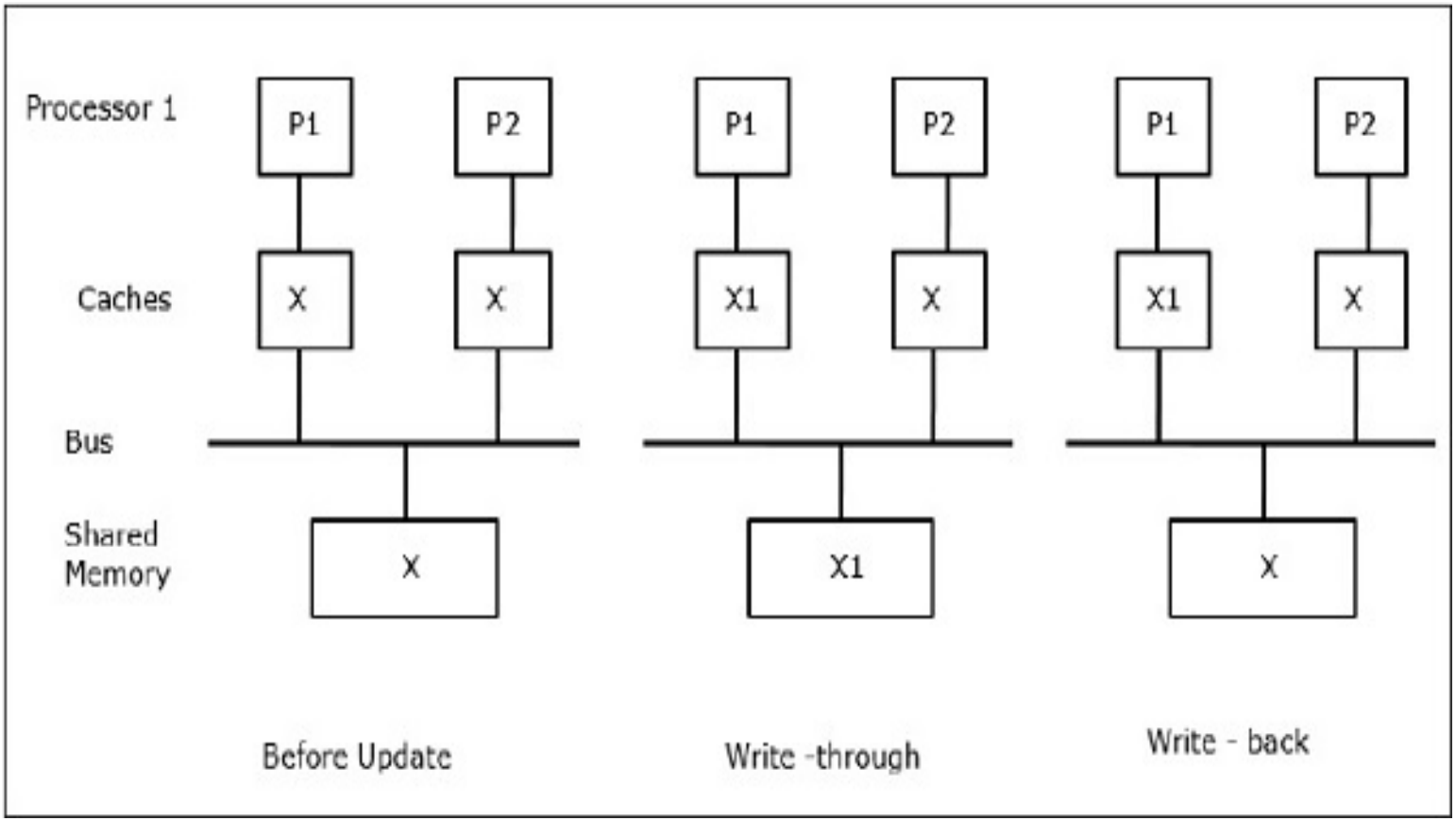
CACHE COHERENCE

In computer architecture, cache coherence is the uniformity of shared resource data that ends up stored in multiple local caches. When clients in a system maintain caches of a common memory resource, problems may arise with incoherent data, which is particularly the case with CPUs in a multiprocessing system.

The Cache Coherence Problem

- As multiple processors operate in parallel, and independently multiple caches may possess different copies of the same memory block, this creates cache coherence problem. Cache coherence schemes help to avoid this problem by maintaining a uniform state for each cached block of data.


Processor 1
P1
P2
Caches
X
X
Bus
Shared Memory
X
Before Update
P1
P2
X1
X
X1
Write -through
P1
P2
X1
X
X
Write - back

- Let X be an element of shared data which has been referenced by two processors, P1 and P2. In the beginning, three copies of X are consistent. If the processor P1 writes a new data X1 into the cache, by using **write-through policy**, the same copy will be written immediately into the shared memory. In this case, inconsistency occurs between cache memory and the main memory. When a **write-back policy** is used, the main memory will be updated when the modified data in the cache is replaced or invalidated.

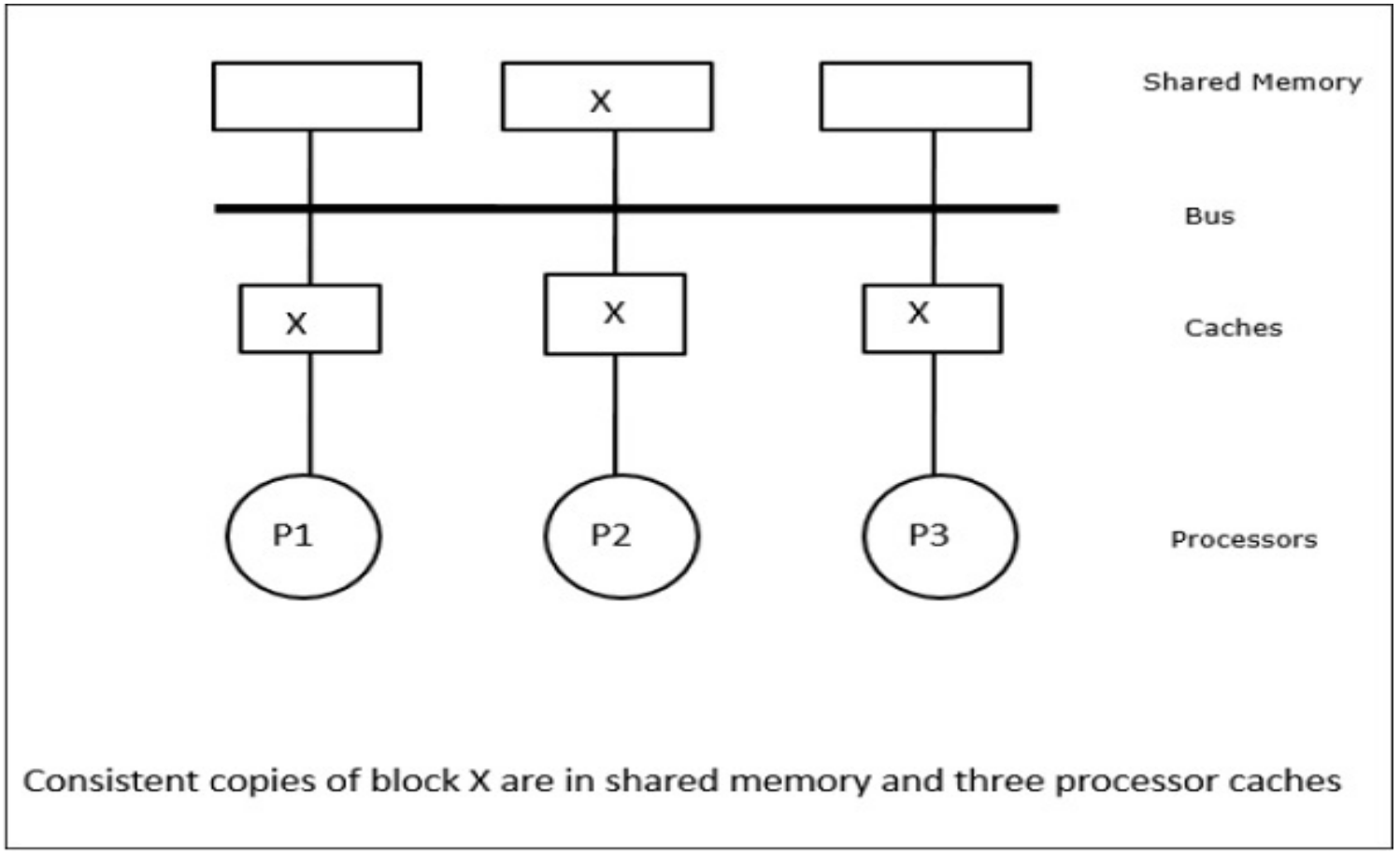
INCONSISTENCY PROBLEMS

- In general, there are three sources of inconsistency problem
- Sharing of writable data
- Process migration
- I/O activity

Snoopy Bus Protocols

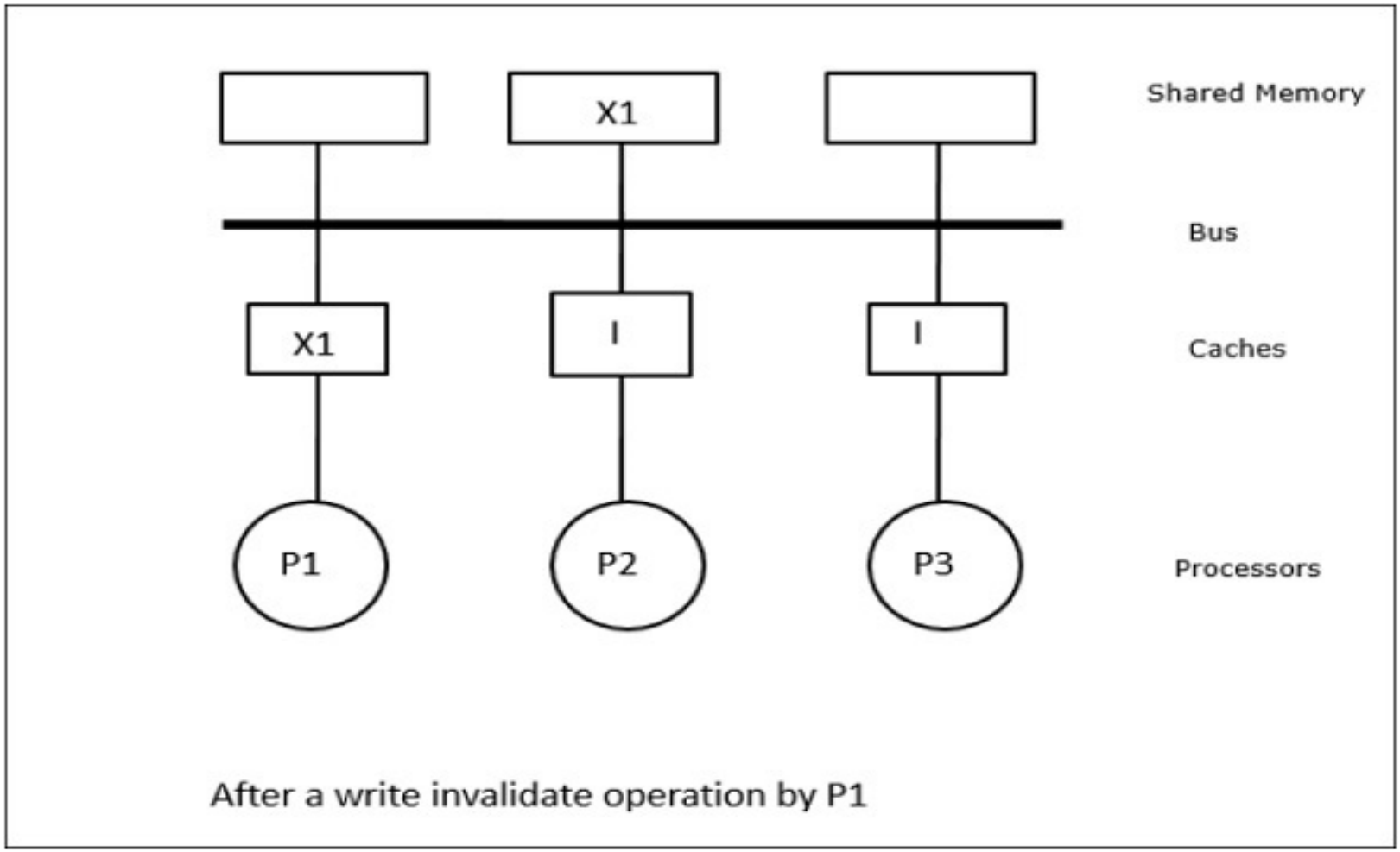
- Snoopy protocols achieve data consistency between the cache memory and the shared memory through a bus-based memory system. Write-invalidate and write-update policies are used for maintaining cache consistency.

(Fig: a)

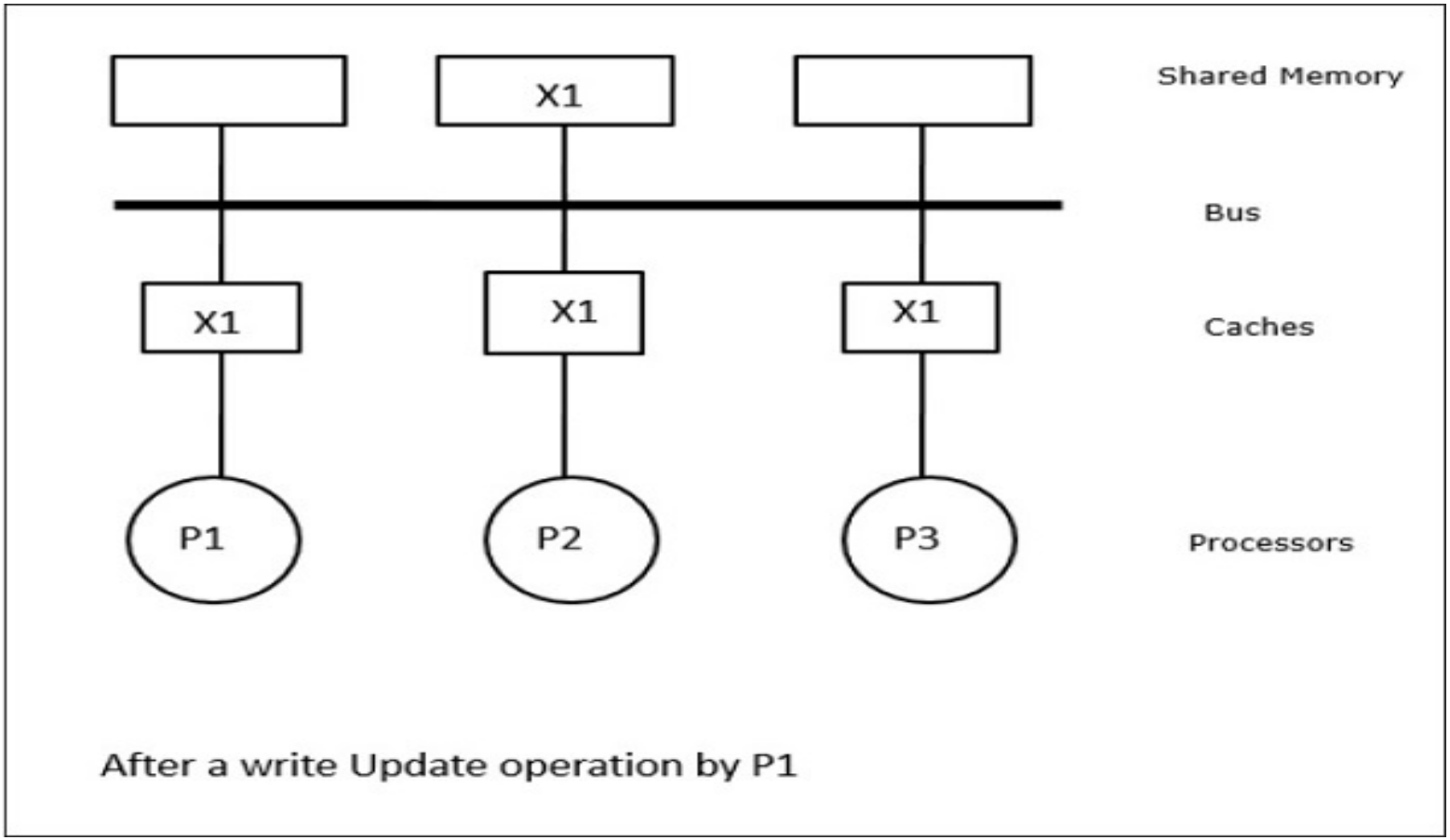


Consistent copies of block X are in shared memory and three processor caches

(Fig: b)


X1
Shared Memory
Bus
X1
I
I
Caches
P1
P2
P3
Processors
After a write invalidate operation by P1

(Fig: c)


X1
Shared Memory
Bus
X1
X1
X1
Caches
P1
P2
P3
Processors
After a write Update operation by P1

- In this case, we have three processors P1, P2, and P3 having a consistent copy of data element ‘X’ in their local cache memory and in the shared memory (Figure-a). Processor P1 writes X1 in its cache memory using **write-invalidate protocol**. So, all other copies are invalidated via the bus. It is denoted by ‘I’ (Figure-b). Invalidated blocks are also known as **dirty**, i.e. they should not be used. The **write-update protocol** updates all the cache copies via the bus. By using **write back cache**, the memory copy is also updated (Figure-c).

Directory-Based Protocols

- By using a multistage network for building a large multiprocessor with hundreds of processors, the snoopy cache protocols need to be modified to suit the network capabilities. Broadcasting being very expensive to perform in a multistage network, the consistency commands is sent only to those caches that keep a copy of the block. This is the reason for development of directory-based protocols for network-connected multiprocessors.

Directory-Based Protocols

- In a directory-based protocols system, data to be shared are placed in a common directory that maintains the coherence among the caches. Here, the directory acts as a filter where the processors ask permission to load an entry from the primary memory to its cache memory. If an entry is changed the directory either updates it or invalidates the other caches with that entry.

Hardware Synchronization Mechanisms

- Synchronization is a special form of communication where instead of data control, information is exchanged between communicating processes residing in the same or different processors.
- Multiprocessor systems use hardware mechanisms to implement low-level synchronization operations. Most multiprocessors have hardware mechanisms to impose atomic operations such as memory read, write or read-modify-write operations to implement some synchronization primitives. Other than atomic memory operations, some inter-processor interrupts are also used for synchronization purposes.

Cache Coherency in Shared Memory Machines

Maintaining cache coherency is a problem in multiprocessor system when the processors contain local cache memory. Data inconsistency between different caches easily occurs in this system.

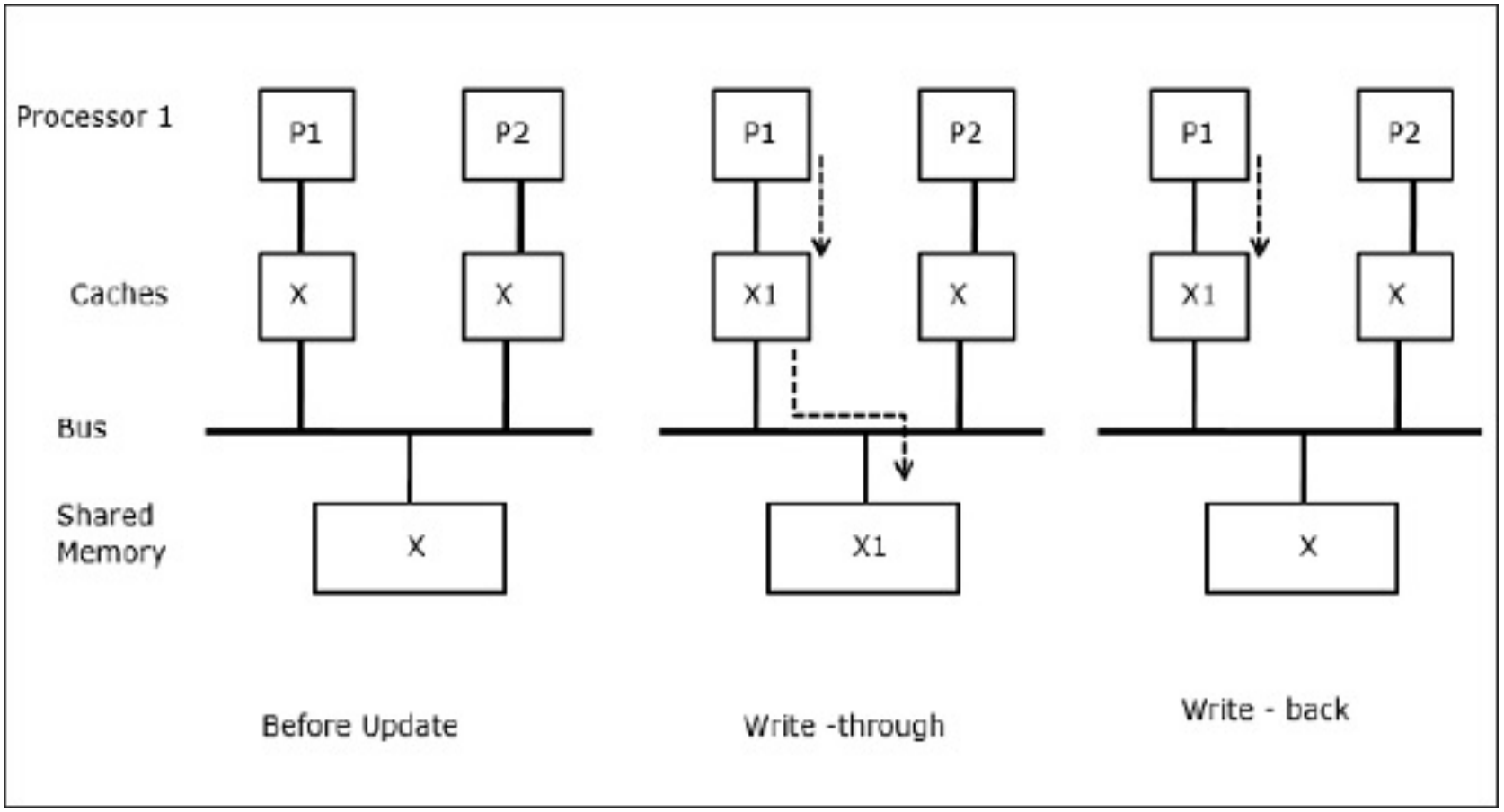
The major concern areas are

- Sharing of writable data
- Process migration
- I/O activity

Sharing of writable data

- (In fig. d) when two processors (P1 and P2) have same data element (X) in their local caches and one process (P1) writes to the data element (X), as the caches are write-through local cache of P1, the main memory is also updated. Now when P2 tries to read data element (X), it does not find X because the data element in the cache of P2 has become outdated.

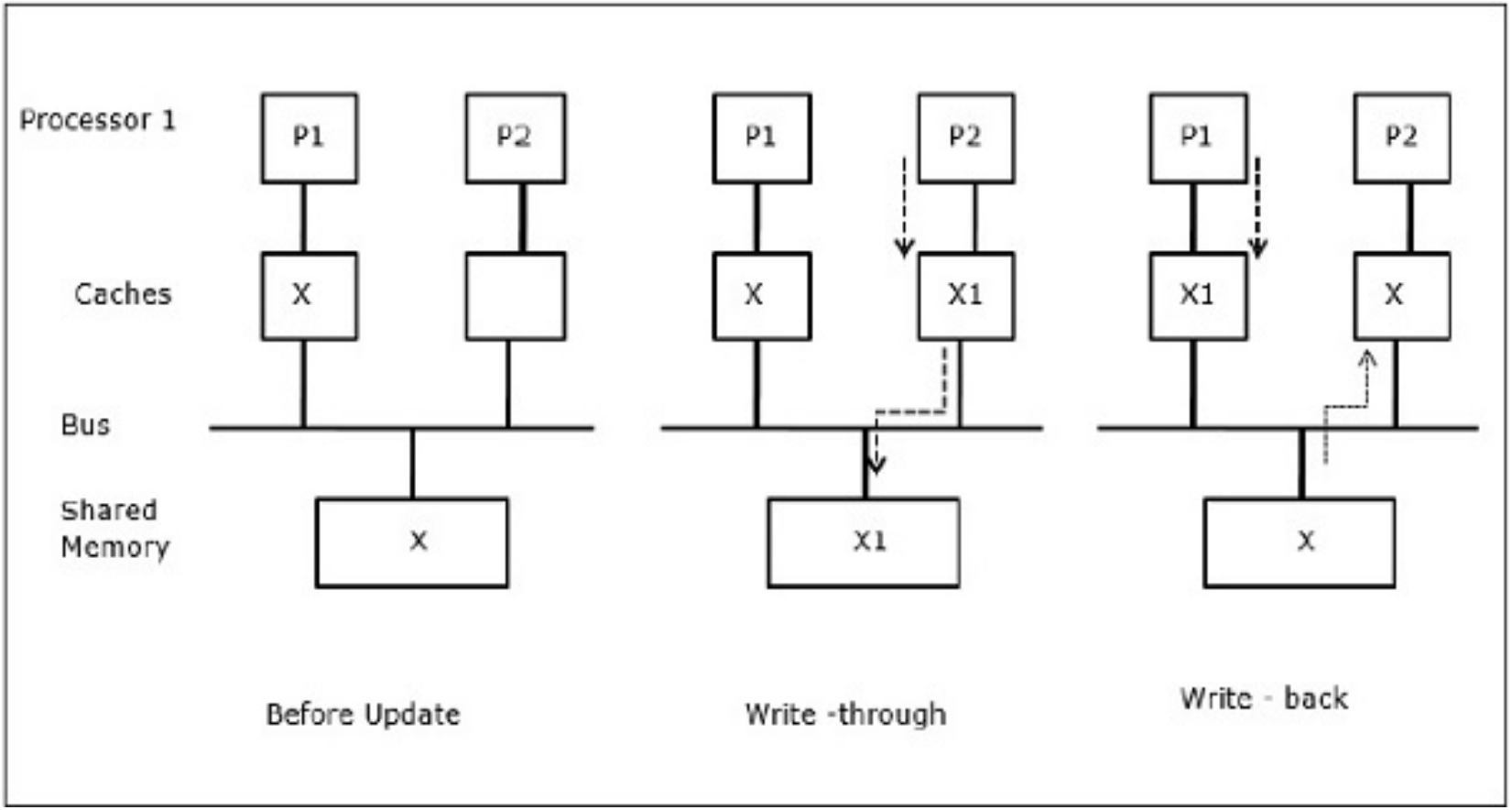
Sharing of writable data (fig.d)



Process migration

- In the first stage, cache of P1 has data element X, whereas P2 does not have anything. A process on P2 first writes on X and then migrates to P1. Now, the process starts reading data element X, but as the processor P1 has outdated data the process cannot read it. So, a process on P1 writes to the data element X and then migrates to P2. After migration, a process on P2 starts reading the data element X but it finds an outdated version of X in the main memory (fig. e).

Process migration (fig. e)



I/O activity

- an I/O device is added to the bus in a two-processor multiprocessor architecture. In the beginning, both the caches contain the data element X. When the I/O device receives a new element X, it stores the new element directly in the main memory. Now, when either P1 or P2 (assume P1) tries to read element X it gets an outdated copy. So, P1 writes to element X. Now, if I/O device tries to transmit X it gets an outdated copy (fig. f).

I/O activity (fig. f)