MODULE III

BY
Elizabeth Isaac
Department of Computer Science and Engg
MA College of Engineering

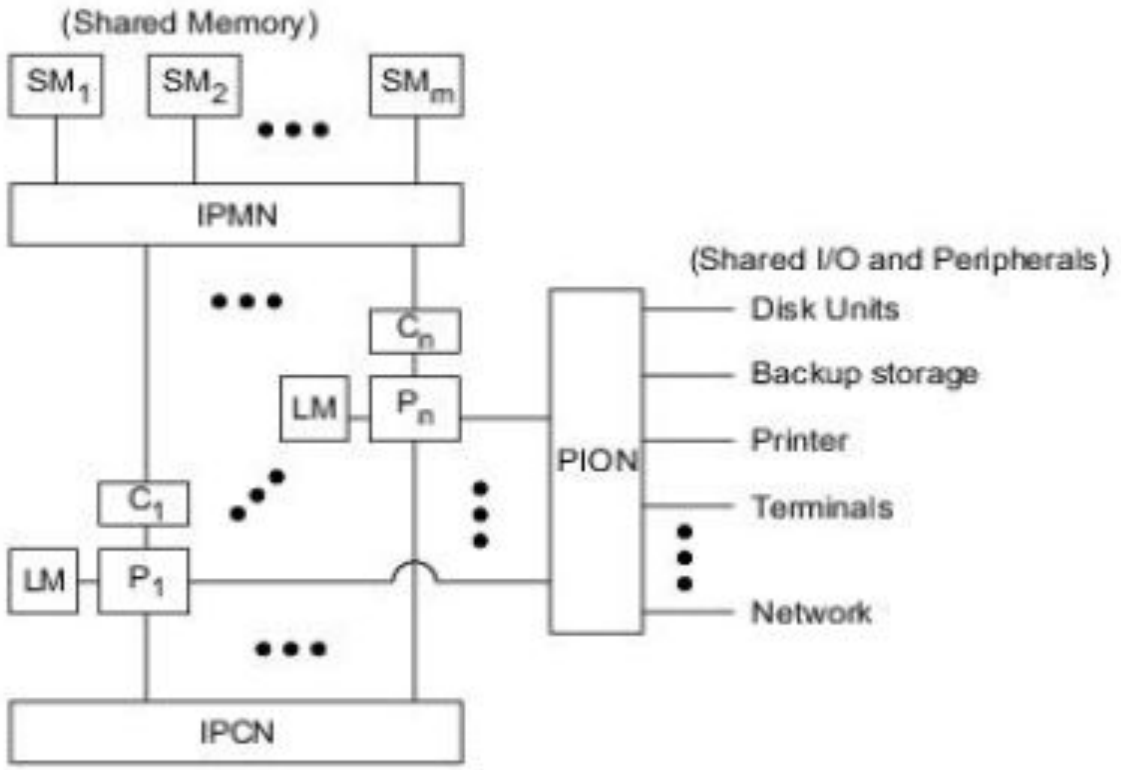
MODULE III

- **Multiprocessors system interconnects**
  - Hierarchical bus systems
  - Cross bar switch and multiport memory
  - Multistage and combining networks
- Cache Coherence and Synchronization Mechanisms
  - Cache Coherence Problem
  - Snoopy Bus Protocol
  - Directory Based Protocol
  - Hardware Synchronization Problem

Multiprocessor System Interconnects

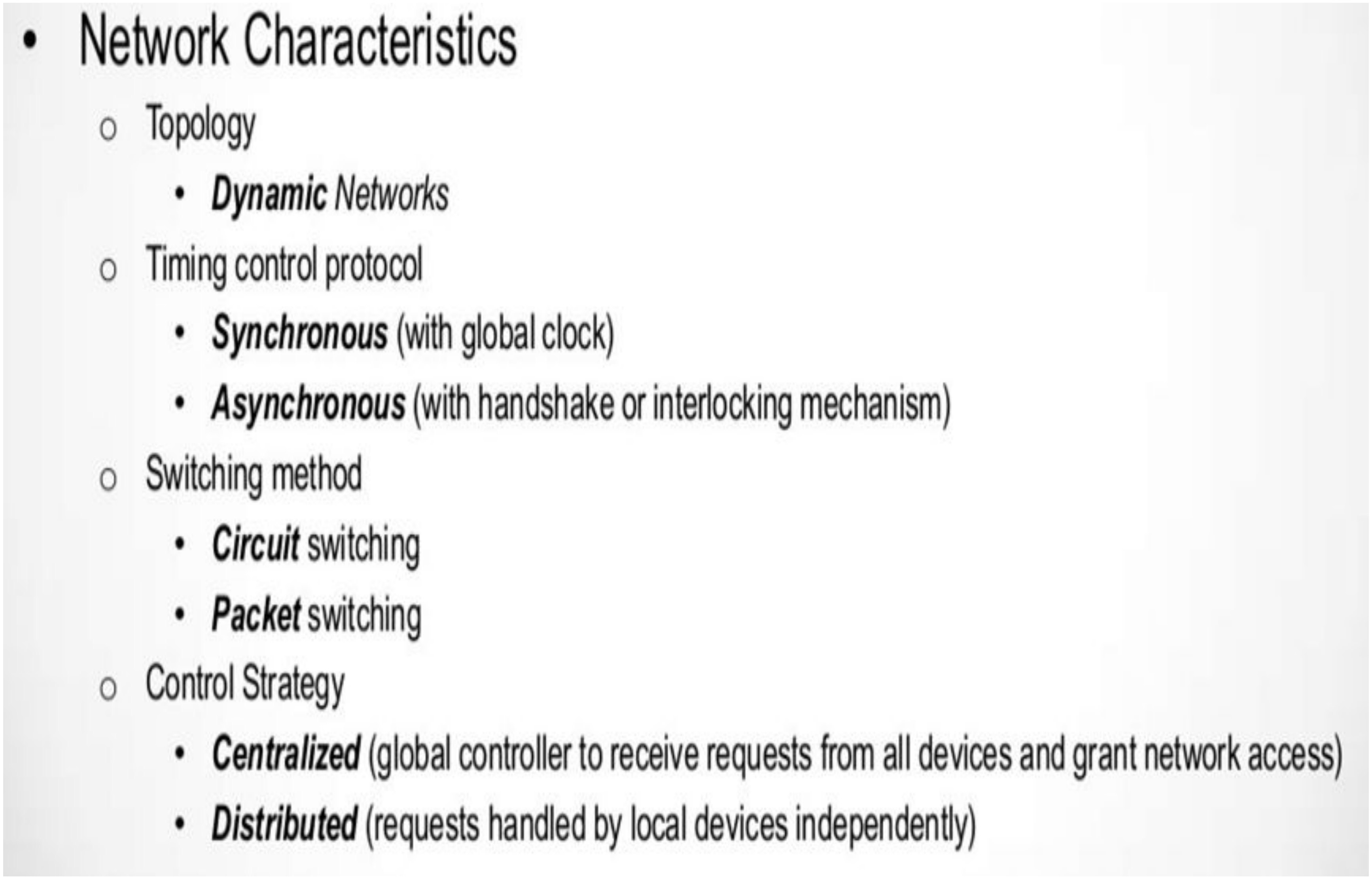
- Parallel processing - efficient system interconnects for fast communication among multiple processors and shared memory, I/0, and peripheral devices
- Hierarchical buses, crossbar switches, and multistage networks

Generalized Multiprocessor System



Legends:	IPMN	(Inter-Processor-Memory Network)
	PION	(Processor- I/O Network)
	IPCN	(Inter-Processor Communication Network)
	P	(Processor)
	C	(Cache)
	SM	(Shared Memory)
	LM	(Local Memory)

Fig. 7.1 Interconnection structures in a generalized multiprocessor system with local memory, private caches, shared memory, and shared peripherals



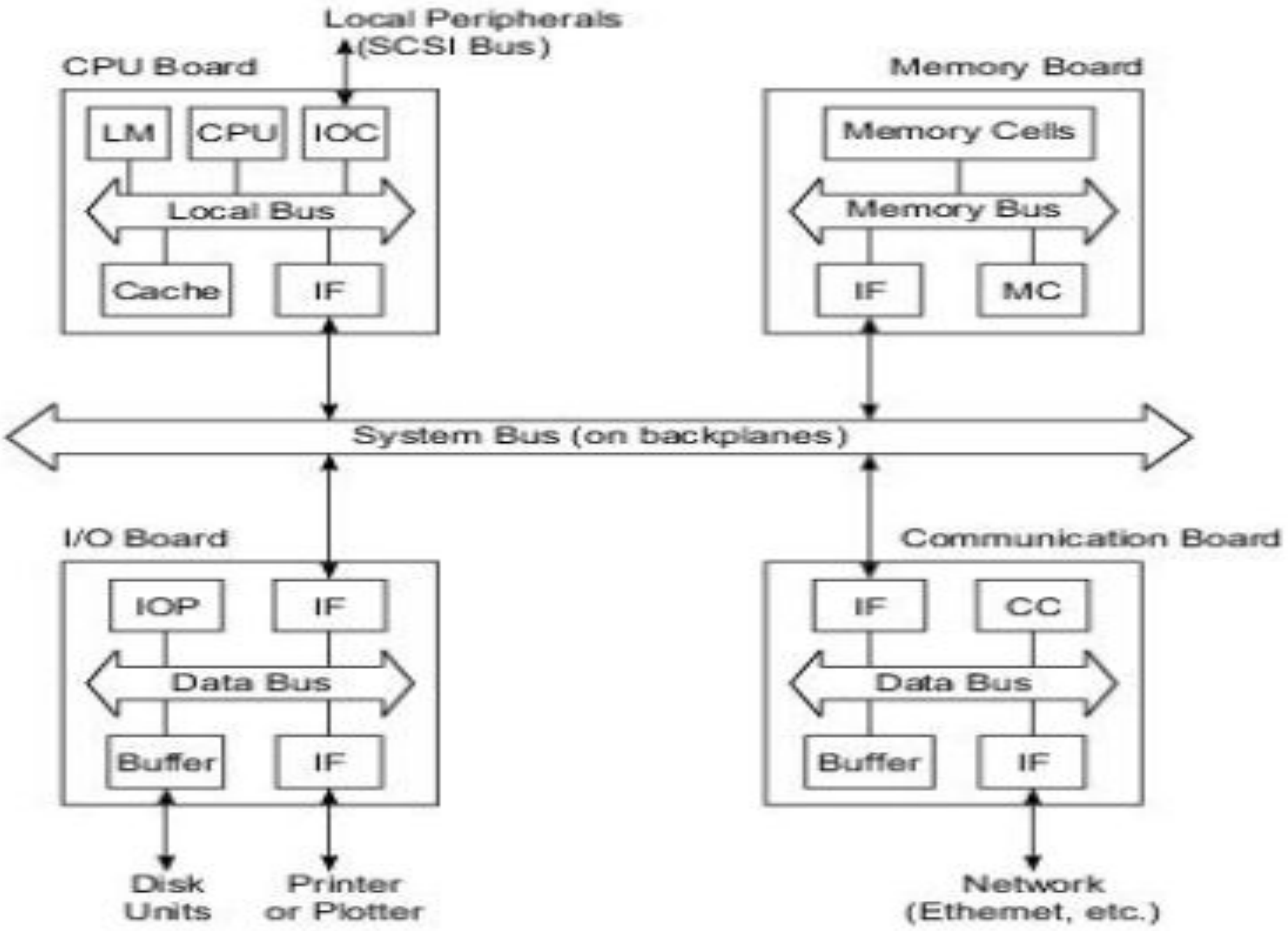
- Network Characteristics
  - Topology
    - ***Dynamic** Networks*
  - Timing control protocol
    - ***Synchronous*** (with global clock)
    - ***Asynchronous*** (with handshake or interlocking mechanism)
  - Switching method
    - ***Circuit*** switching
    - ***Packet*** switching
  - Control Strategy
    - ***Centralized*** (global controller to receive requests from all devices and grant network access)
    - ***Distributed*** (requests handled by local devices independently)

MODULE III

- Multiprocessors system interconnects
    - **Hierarchical bus systems**
    - Cross bar switch and multiport memory
    - Multistage and combining networks
- Cache Coherence and Synchronization Mechanisms
    - Cache Coherence Problem
    - Snoopy Bus Protocol
    - Directory Based Protocol
    - Hardware Synchronization Problem

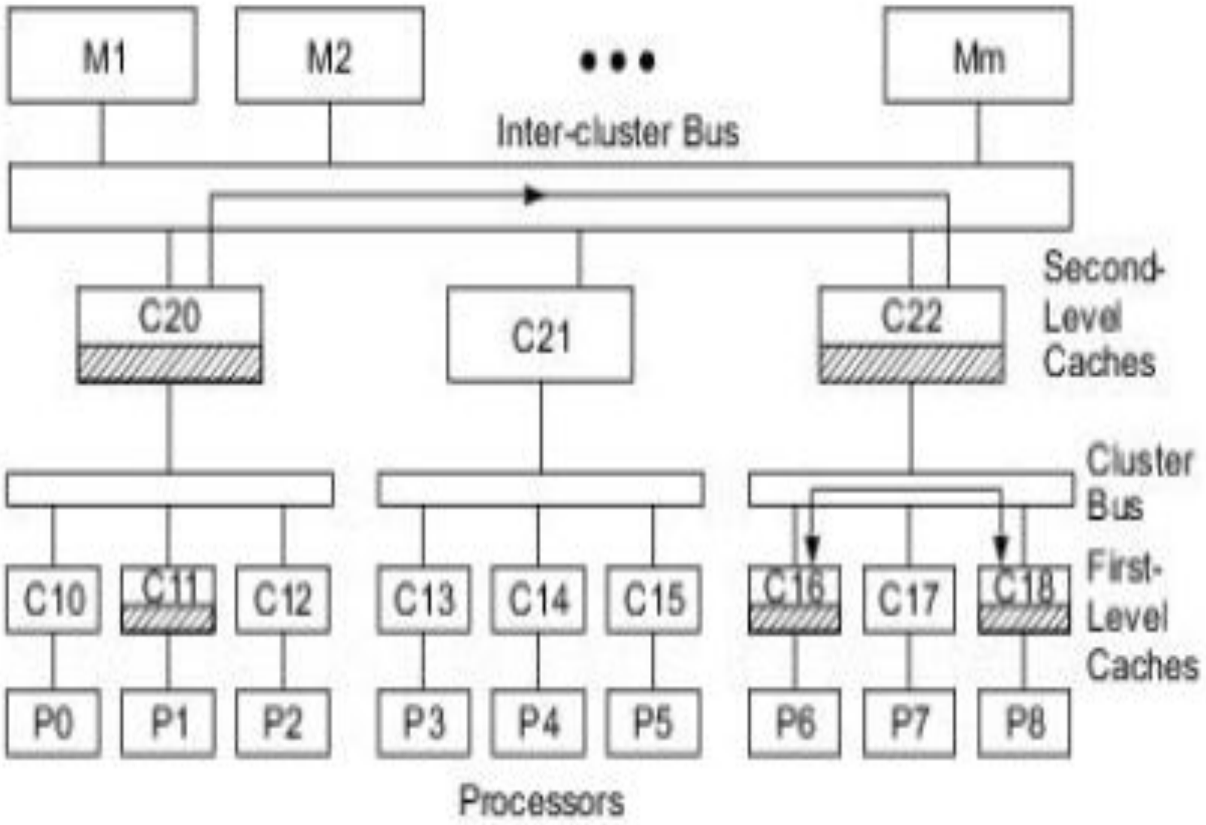
Hierarchical Bus Systems

- Bus System: hierarchy of buses connecting various system and subsystem components in a computer
- Formed with a number of signal, control, and power lines
- Different buses - different interconnection functions



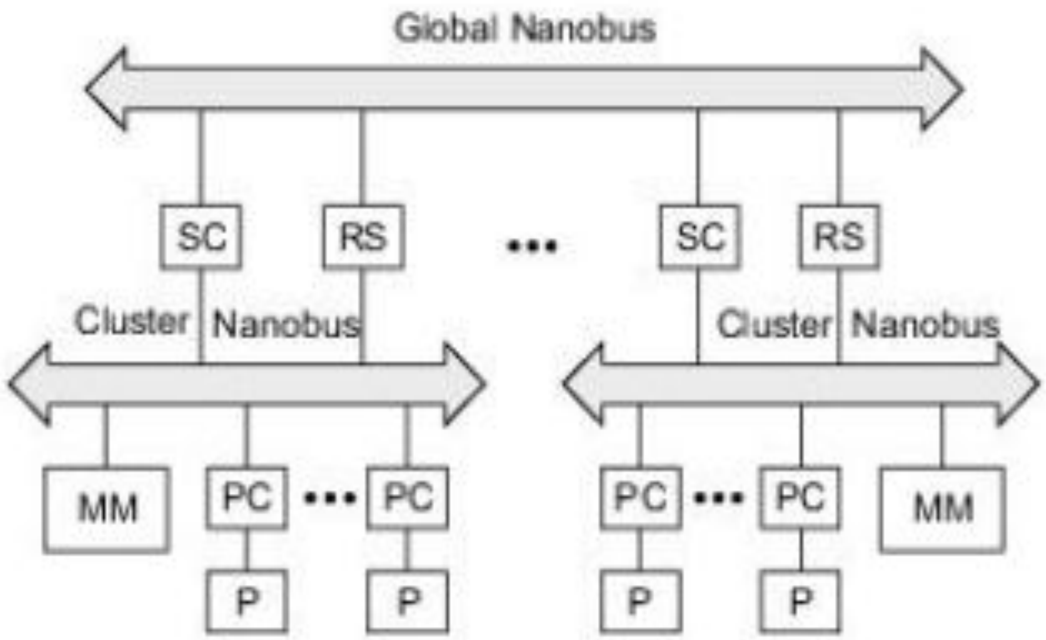
Legends: IF (Interface logic), LM (Local Memory)
IOC (I/O Controller), MC (Memory Controller)
IOP (I/O Processor), CC (Communication Controller)

Fig. 7.2 Bus systems at board level, backplane level, and I/O level



A hierarchical cache/bus architecture for designing a scalable multiprocessor (Courtesy of Wilson; reprinted from *Proc. of Annual Int. Symp. on Computer Architecture, 1987*)

Example 1.1 Encore Ultramax Multiprocessor Architecture

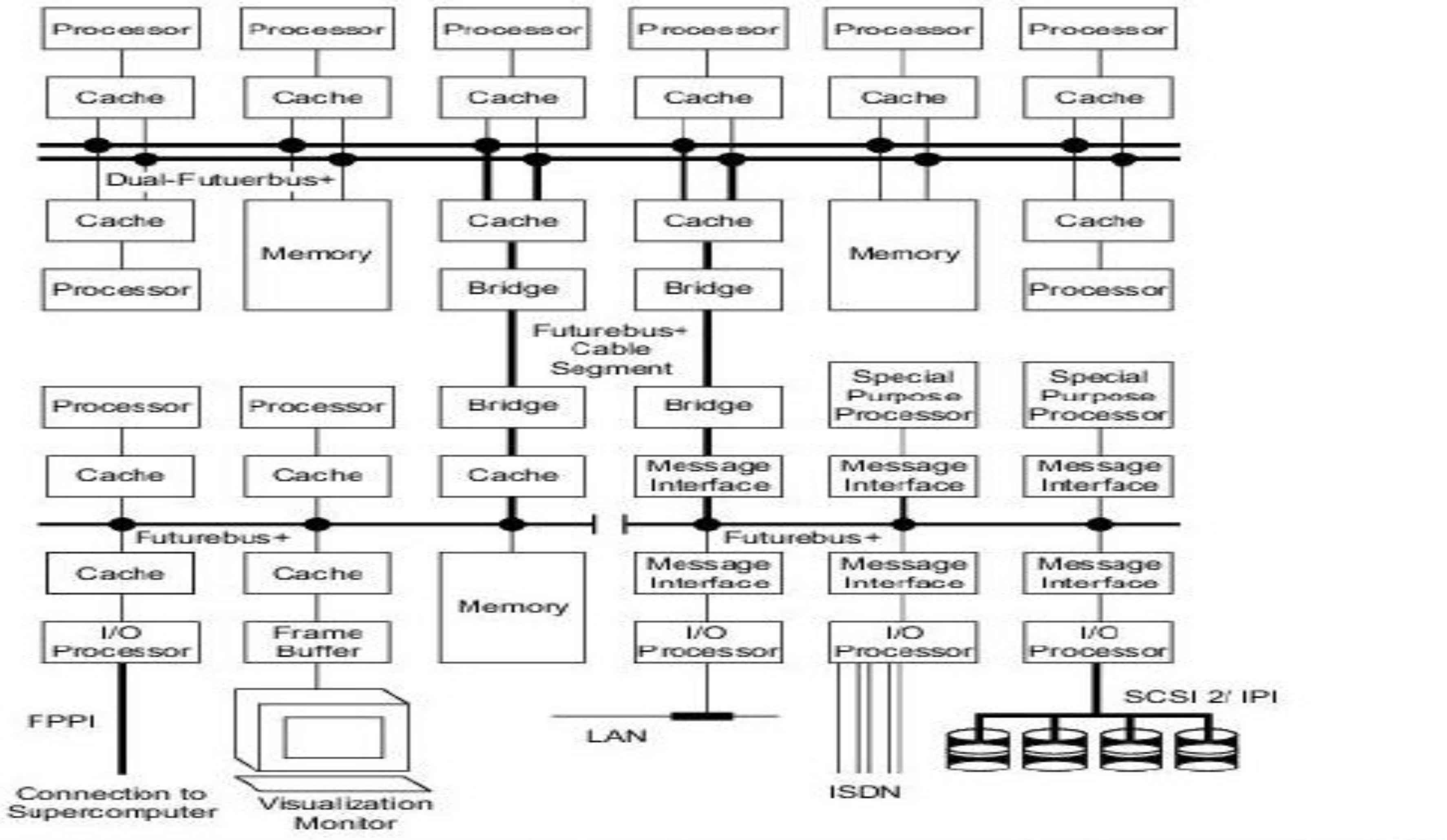


Legends: P = Processor
PC = Private Cache
MM = Main Memory
SC = Shared Cache
RS = Route Switch

The Ultramax multiprocessor architecture using hierarchical buses with multiple clusters (Courtesy of Encore Computer Corporation, 1987)

- Two-level hierarchical-bus architecture
- The shared memories were distributed to all clusters instead of being connected to the intercluster bus
- When an access request reached the top bus, it would be routed down to the cluster memory that matched it with the reference address

- Idea of using bridges between multiprocessor clusters is to allow transactions initiated on a local bus to be completed on a remote bus



A multiprocessor system using multiple Futurebus+ segments (Reprinted with permission from IEEE Standard 896.1-1991, copyright © 1991 by IEEE, Inc.)

MODULE III

- Multiprocessors system interconnects
  - Hierarchical bus systems
  - **Cross bar switch and multiport memory**
  - Multistage and combining networks
- Cache Coherence and Synchronization Mechanisms
  - Cache Coherence Problem
  - Snoopy Bus Protocol
  - Directory Based Protocol
  - Hardware Synchronization Problem

Crossbar Switch and Multiport Memory

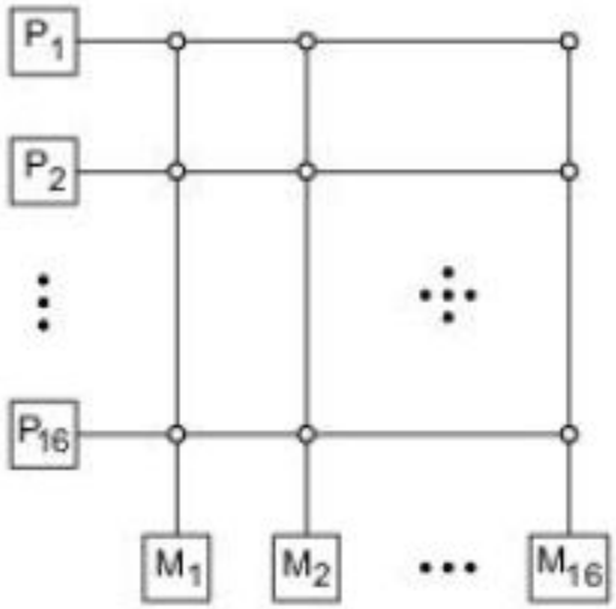
- Switched networks provide dynamic interconnections between the inputs and outputs
- **Network Stage:** Single-stage network is also called a recirculating network
  - cheaper to build, but multiple passes may be needed to establish certain connections

Multi-stage netwok

- **Blocking**
  - eg: Omega, Baseline and Banyan
- **Nonblocking Networks**:
  - Benes netwoks

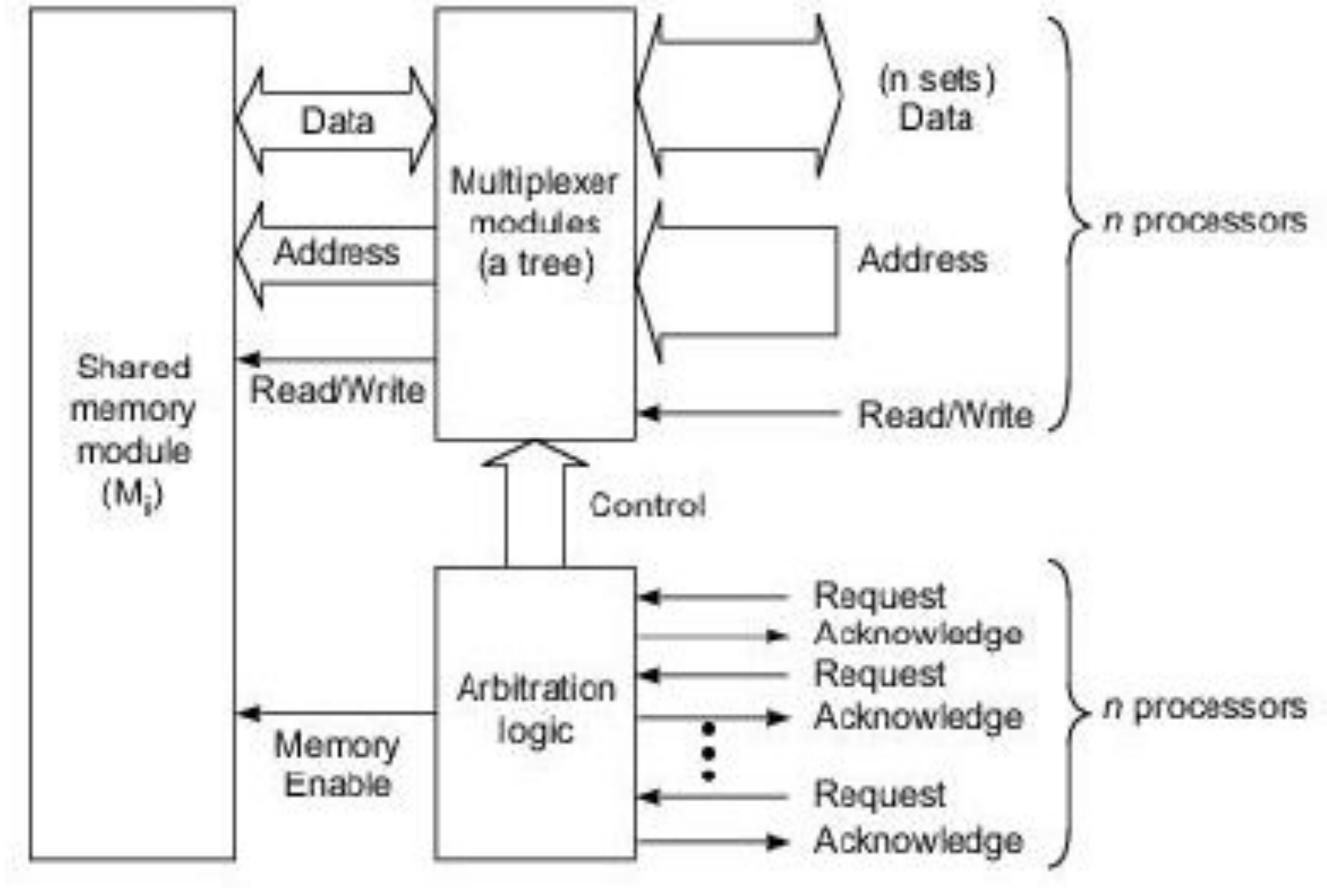
- **Crossbar Networks** : every input port is connected to a free output port through a crosspoint switch without blocking
  - A crossbar network is a single-stage network built with unary switches at the crosspoints
  - Once the data is read from the memory. its value is returned to the requesting processor along the same crosspoint switch

- **Crosspoint Switch Design** : Out of n crosspoint switches in each columnof an n*m or crossbar mesh, only one can be connected at a time
  - To resolve the contention for each memory module, each crosspoint switch must he designed with extra hardware

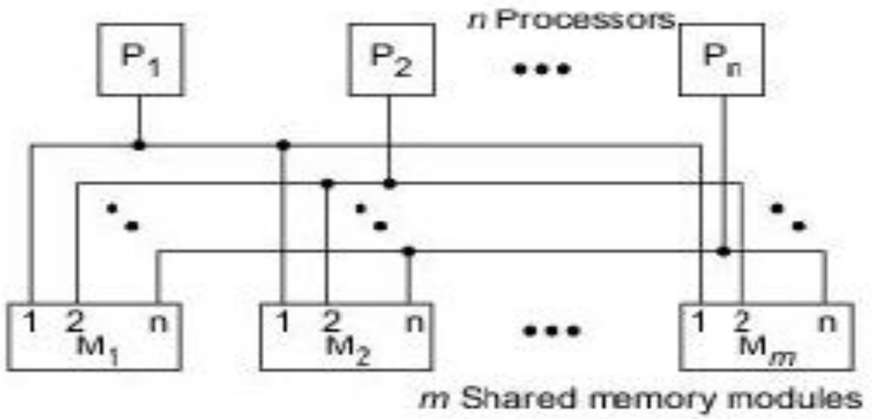


(a) Interprocessor-memory crossbar network built in the C.mmp multiprocessor at Carnegie-Mellon University (1972)

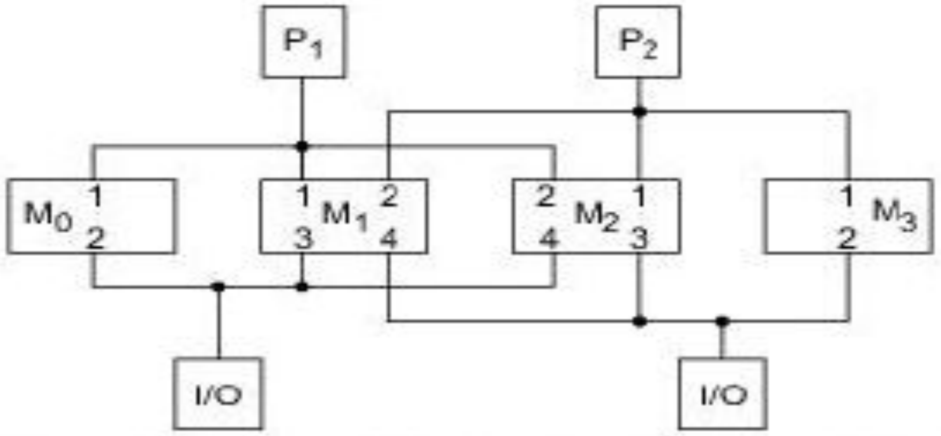
- Figure shows the schematic design of a row of crosspoint switches in a single crossbar network



- **Crossbar Limitations :** single processor can send many requests to multiple memory modules
  - cost-effective only for small multiprocessors with a few processors accessing a few memory modules
- **Multiport Memory** : memory module becomes more expensive due to the added access ports and associated logic.
  - Only one of n processor requests can be honored at a time



(a) n-port memory modules used



(b) Memory ports prioritized or privileged in each module by numbers

Multiport memory organizations for multiprocessor systems (Courtesy of P. H. Enslow, *ACM Computing Surveys*, March 1977)

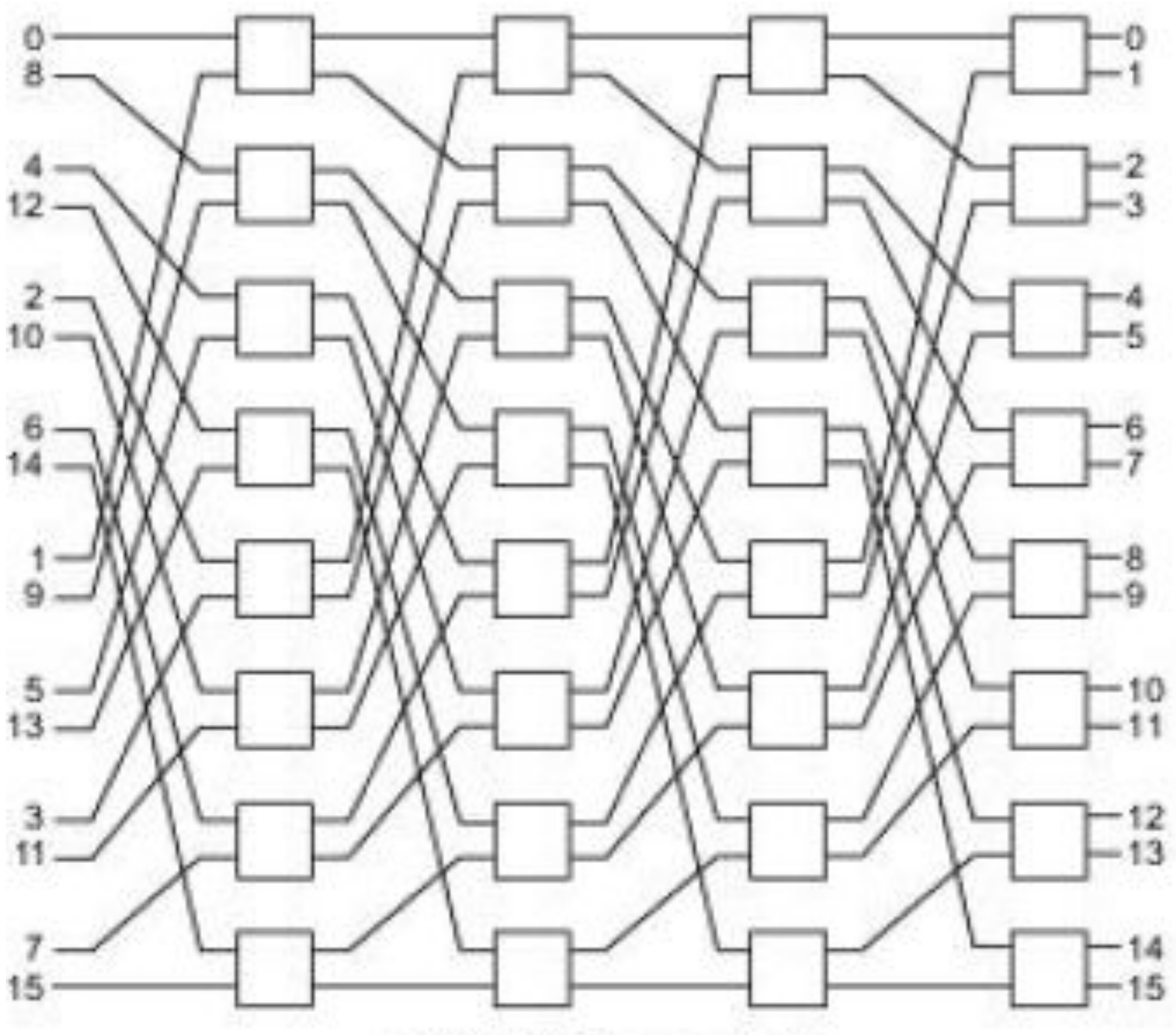
MODULE III

- Multiprocessors system interconnects
  - Hierarchical bus systems
  - Cross bar switch and multiport memory
  - **Multistage and combining networks**
- Cache Coherence and Synchronization Mechanisms
  - Cache Coherence Problem
  - Snoopy Bus Protocol
  - Directory Based Protocol
  - Hardware Synchronization Problem

Multistage and combining networks

- Multistage networks are used to build larger multiprocessor systems.
- Consists of more than one stage of switch boxes
- Perfect shuffle, butterfly, multiway shuffle , crossbar..
- multistage networks are the Omega network and the Butterfly network

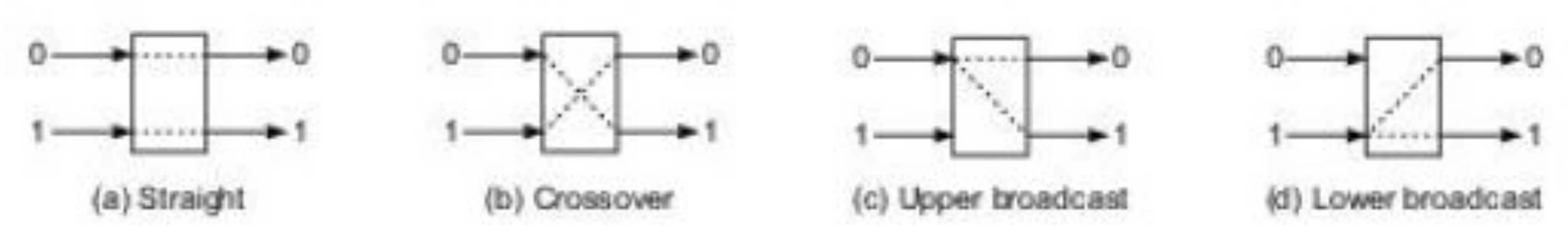
Multistage and combining networks



(e) 16 × 16 Omega network

Properties - Multistage Interconnection Network (MIN).

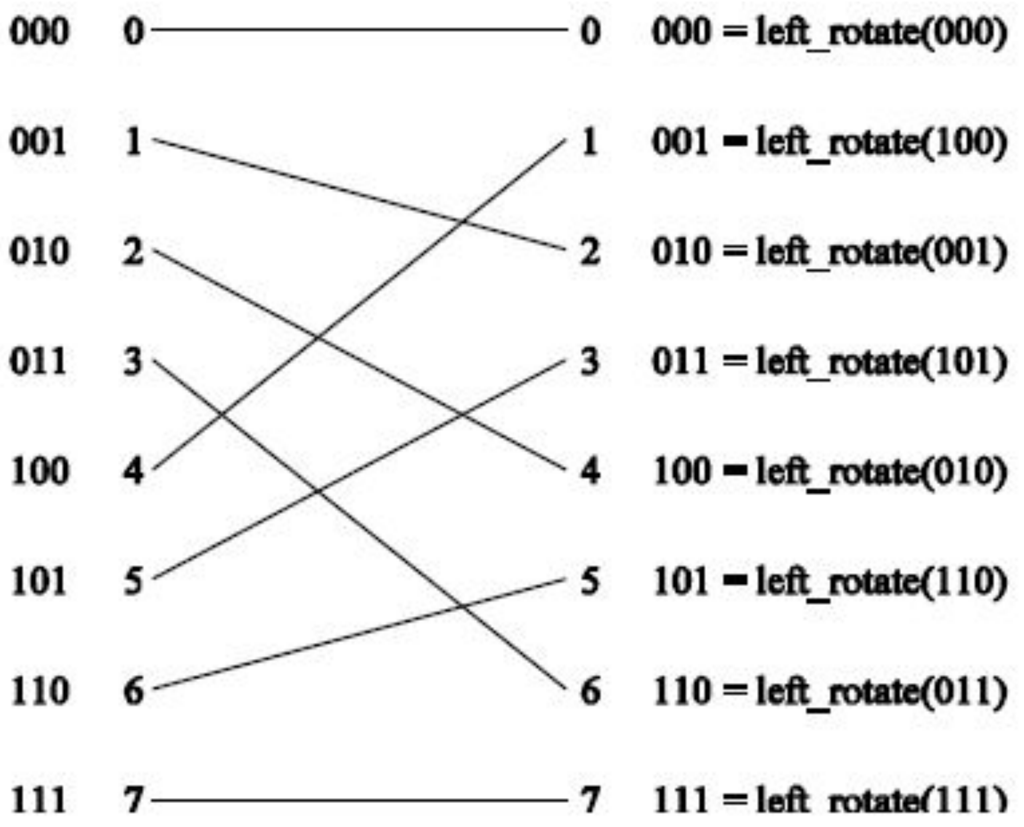
- Consists of $\log_2 N$ stages of 2x2 switches
- Has N/2 number of 2x2 switches per stage
- Stages are labeled from 0 to $\log_2 N-1$
- There are 4 types of switch connection

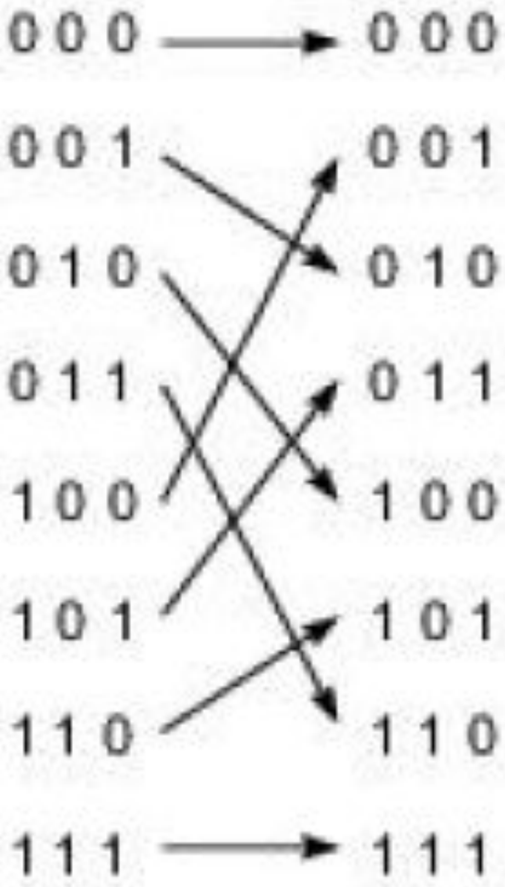


Multistage and Combining Networks

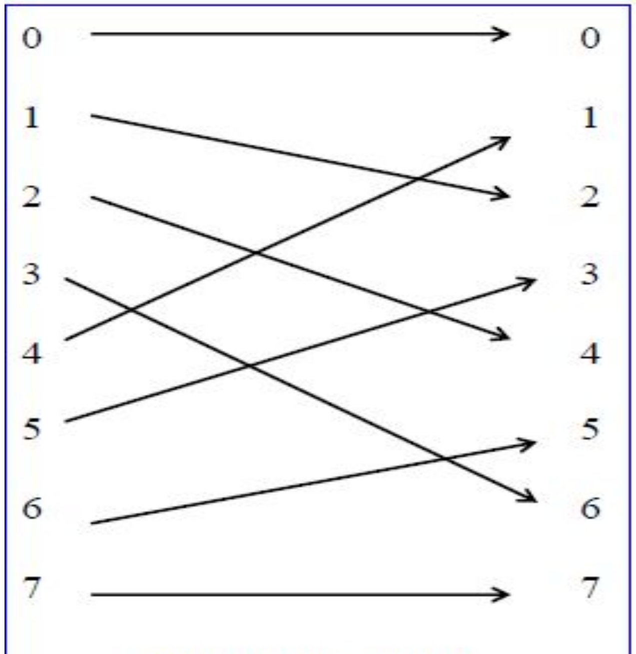
shuffle interconnection => rotate left

$S(a_{n-1}\, a_{n-2} \ldots a_1\, a_0) = (a_{n-2}\, a_{n-3} \ldots a_0\, a_{n-1})$

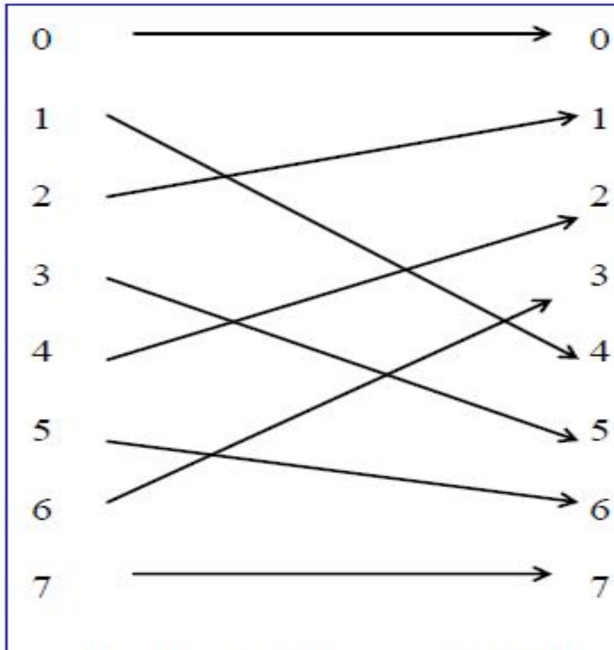




(a) Perfect shuffle



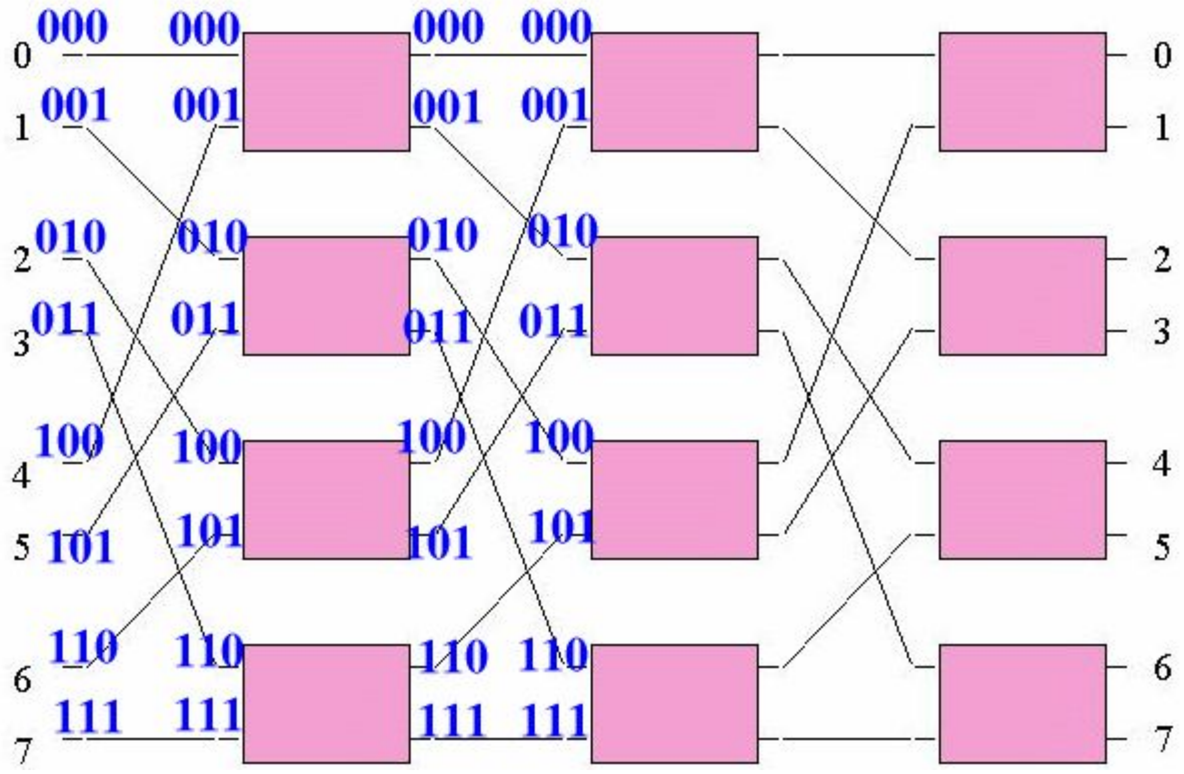
(a) Perfect Shuffle



(b) Inverse Perfect Shuffle

```
000 ---> 000 ---> 000 ---> 000
001 ---> 010 ---> 100 ---> 001
010 ---> 100 ---> 001 ---> 010
011 ---> 110 ---> 101 ---> 011
100 ---> 001 ---> 010 ---> 100
101 ---> 011 ---> 110 ---> 101
110 ---> 101 ---> 011 ---> 110
111 ---> 111 ---> 111 ---> 111
```

8 x 8 OMEGA NETWORK

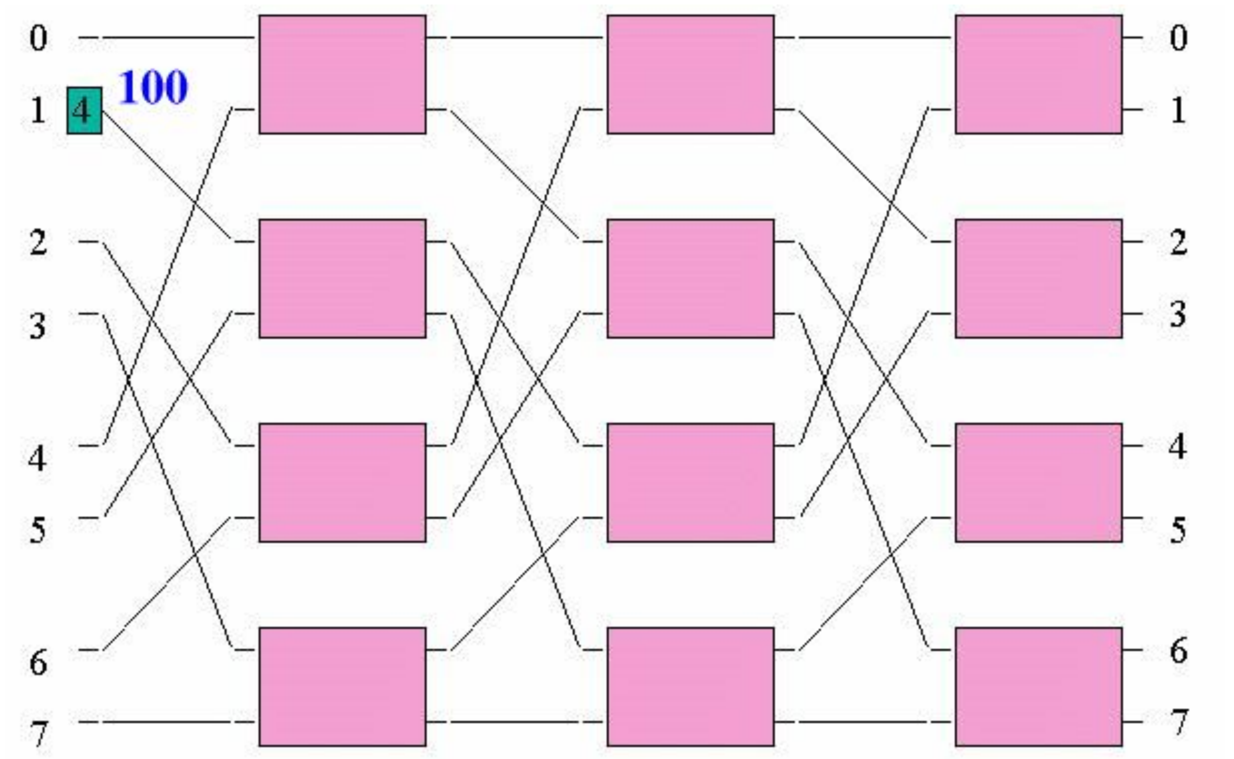


Self Routing in Omega networks

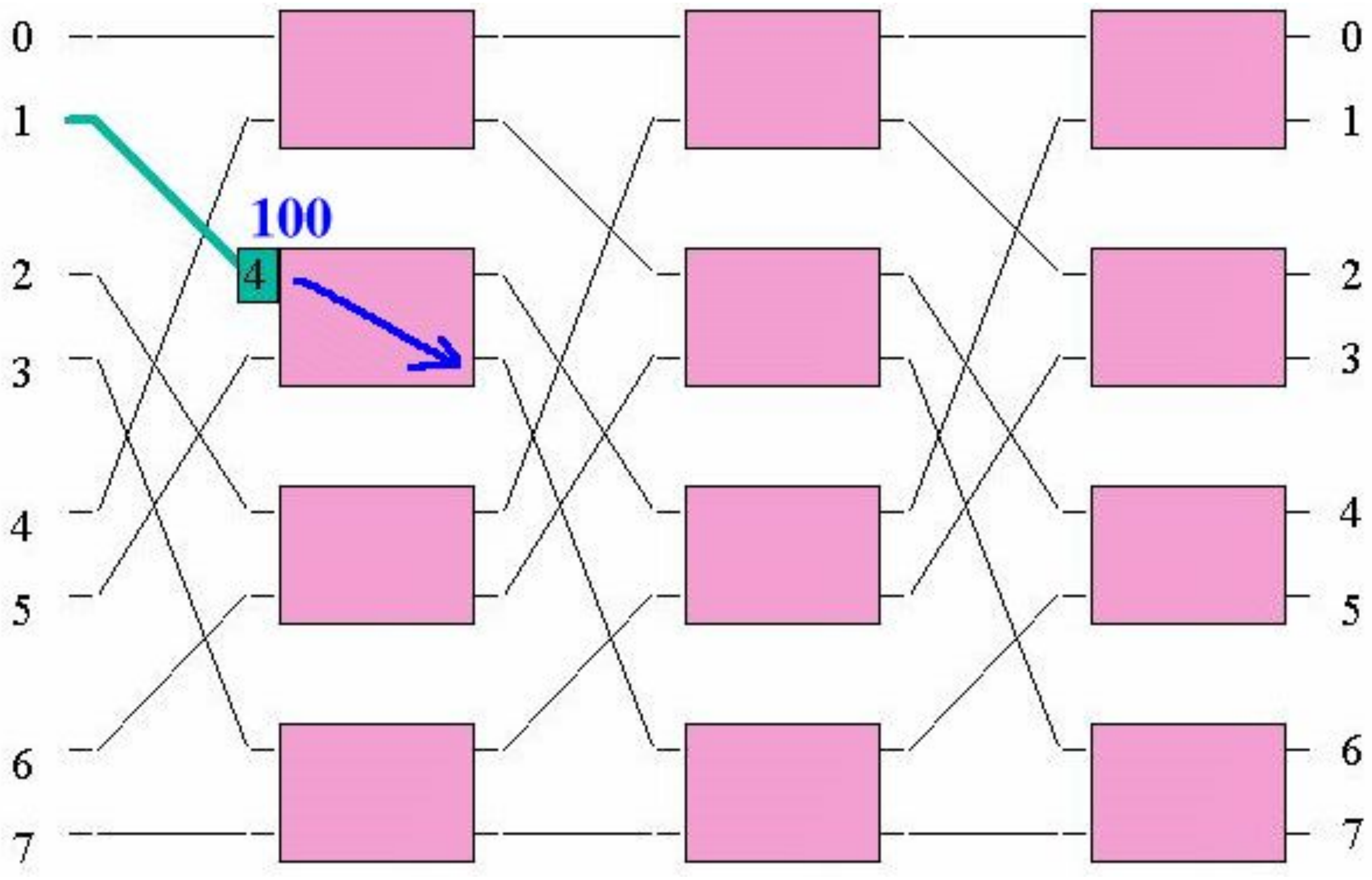
- Destination Tag Routing
- The source node generates a tag, which is binary equivalent of the destination.
- At each switch, the corresponding tag bit of destination is checked.
- If the bit is 0, the input is connected to the upper output.
- If bit is 1, the input is connected to the lower output.
- If both inputs have either 0 or 1, it is a switch conflict. One of them is connected. The other one is rejected or buffered at the switch

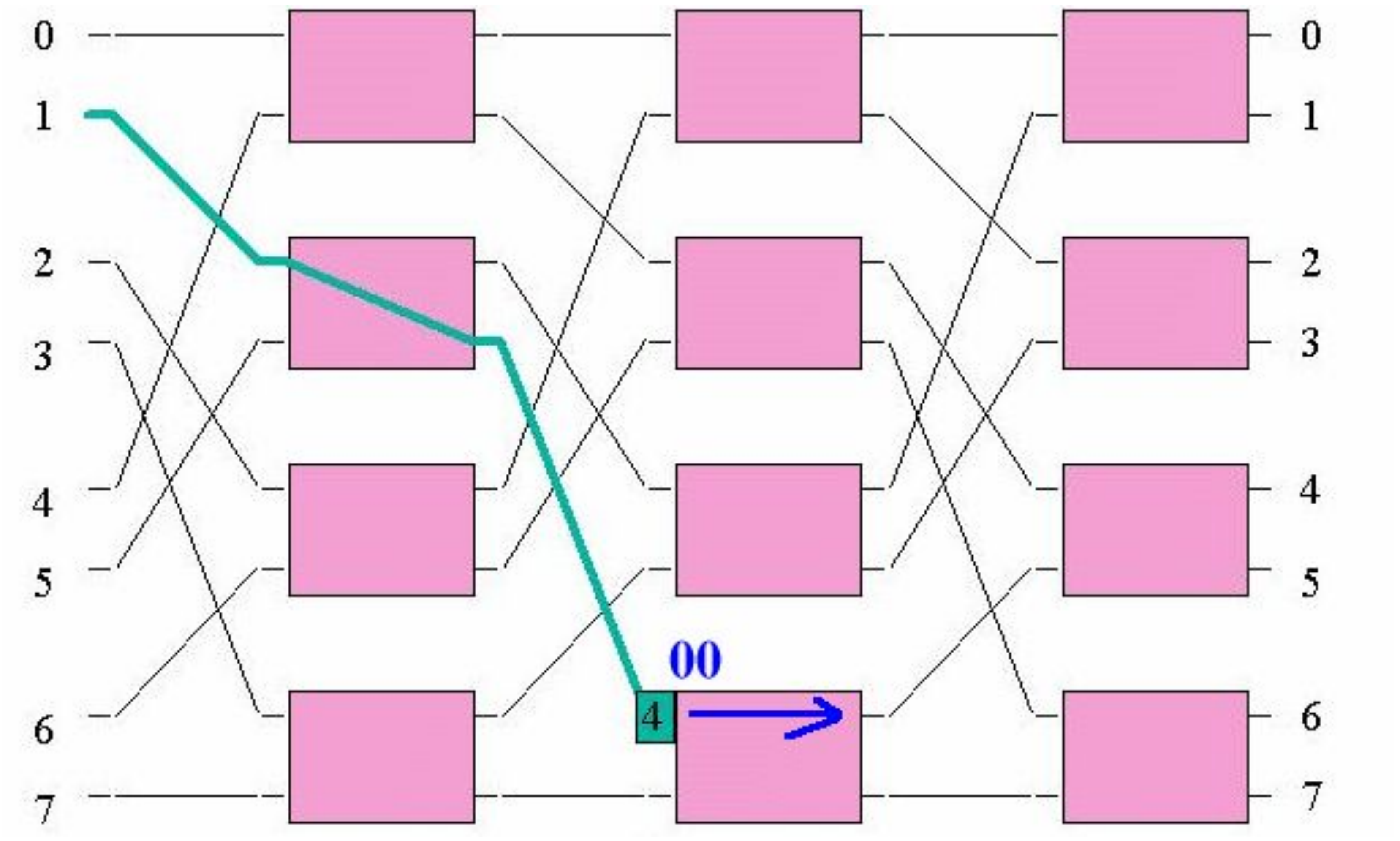
Routing in Omega networks

- Suppose 1 to 4



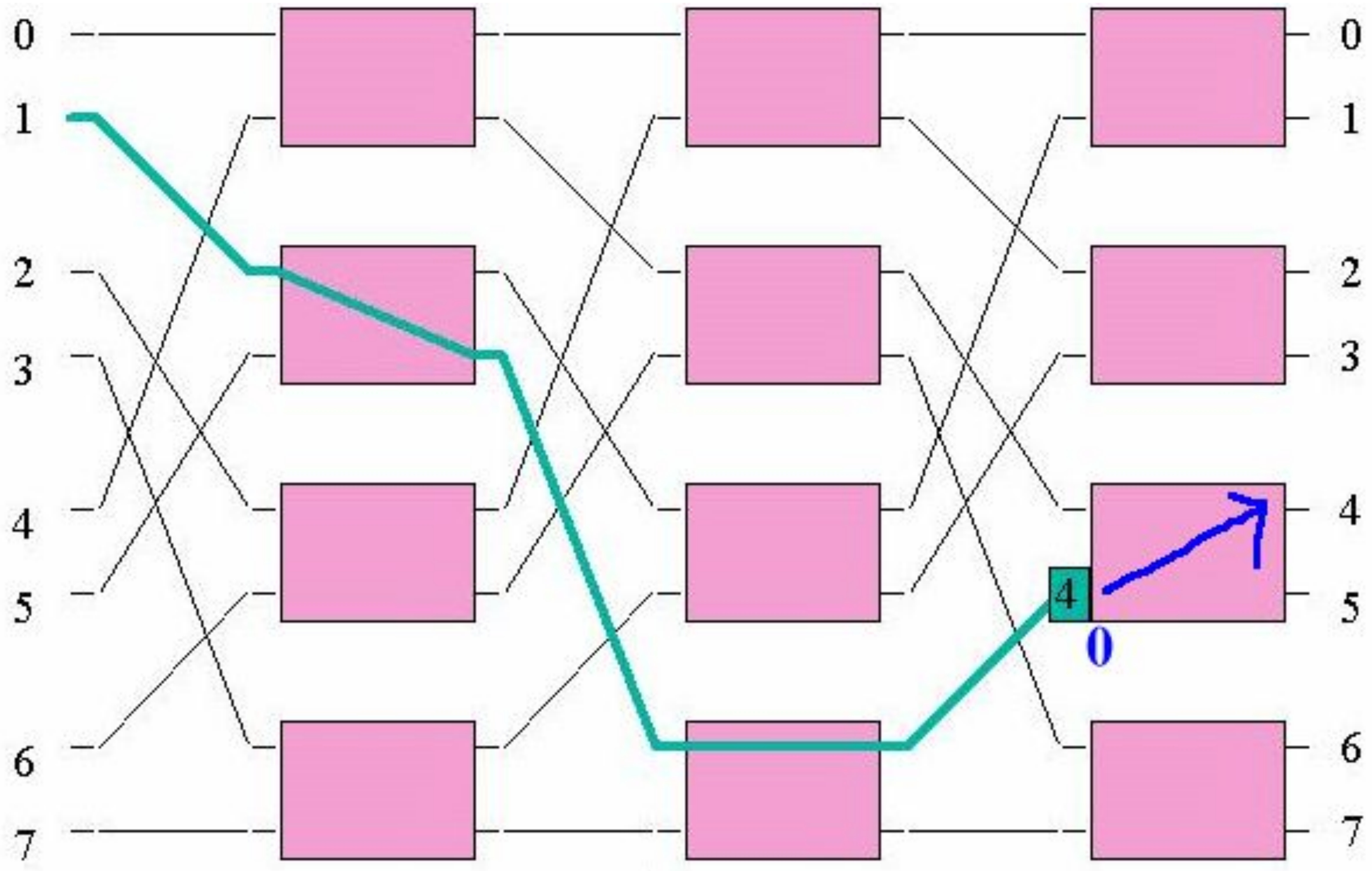
Routing in Omega networks





0
1
2
3
4
5
6
7
00
4
0
1
2
3
4
5
6
7

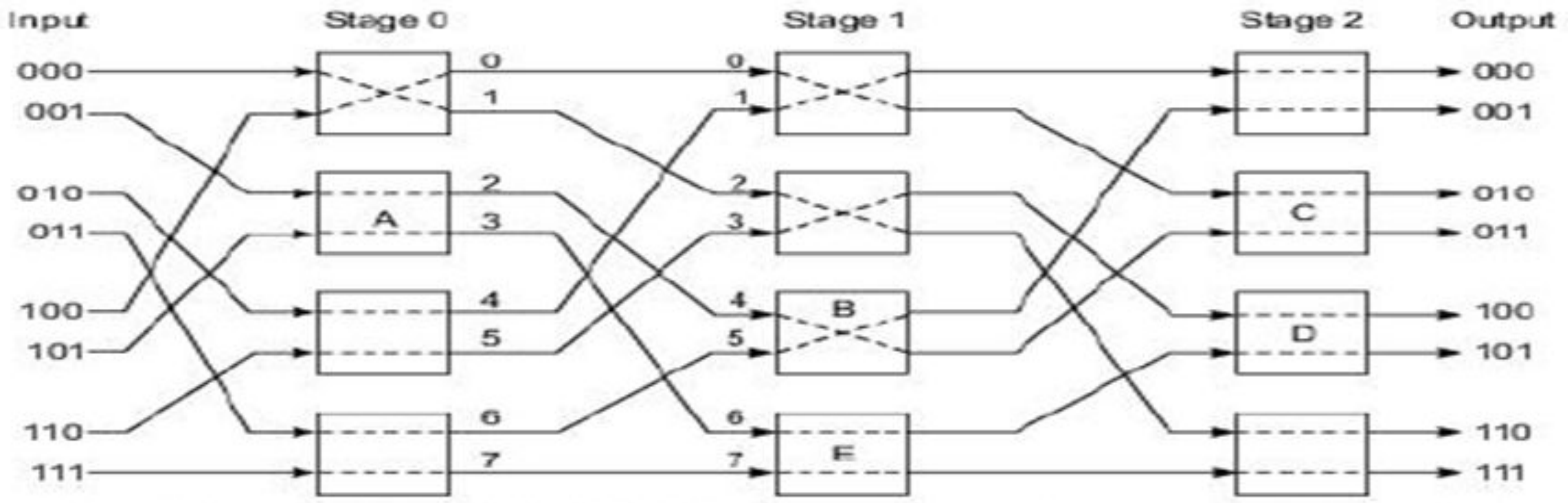
Routing in Omega networks



Routing in Omega Networks

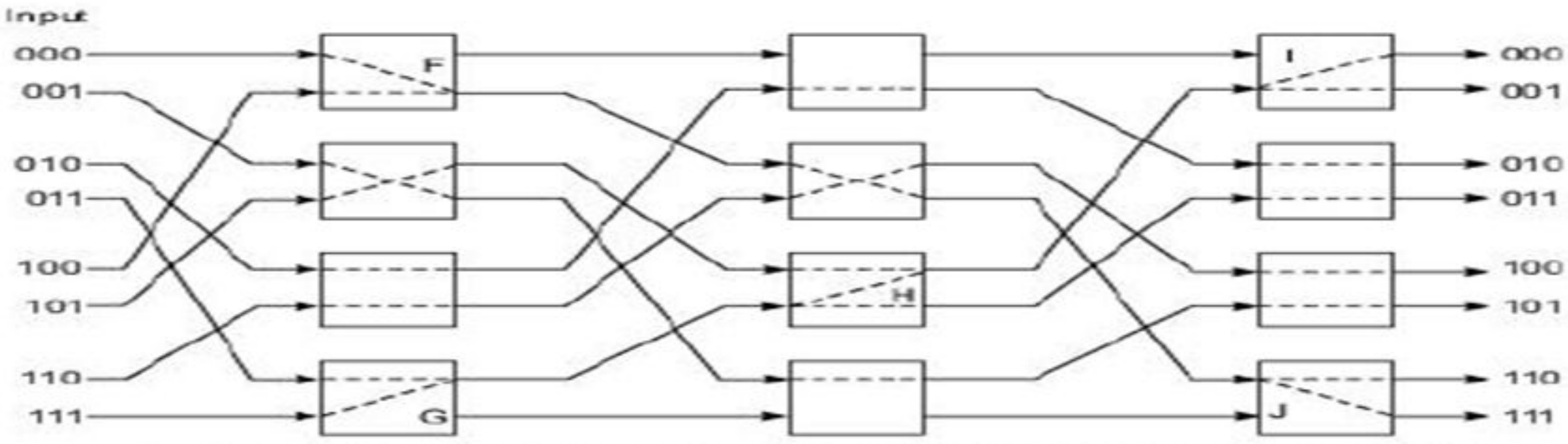
- permutations $\prod_1$ = (0, 7, 6,4, 2) (1, 3) (5)
- $\prod_2$= (0, 6, 4, 7, 3) (1, 5) (2) respectively

- Two switch setting of an 8*8 Omega network built with 2*2 switches



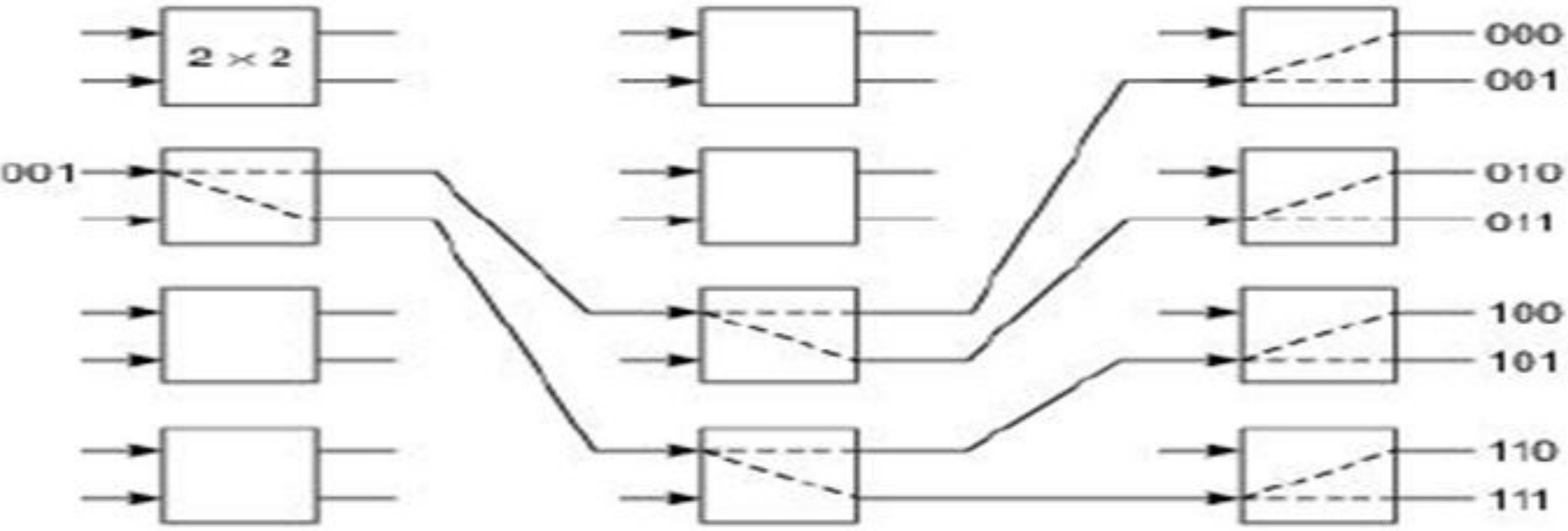
(a) Permutation $\pi_1 = (0, 7, 6, 4, 2)\ (1, 3)\ (5)$ implemented on an Omega network without blocking

Omega network

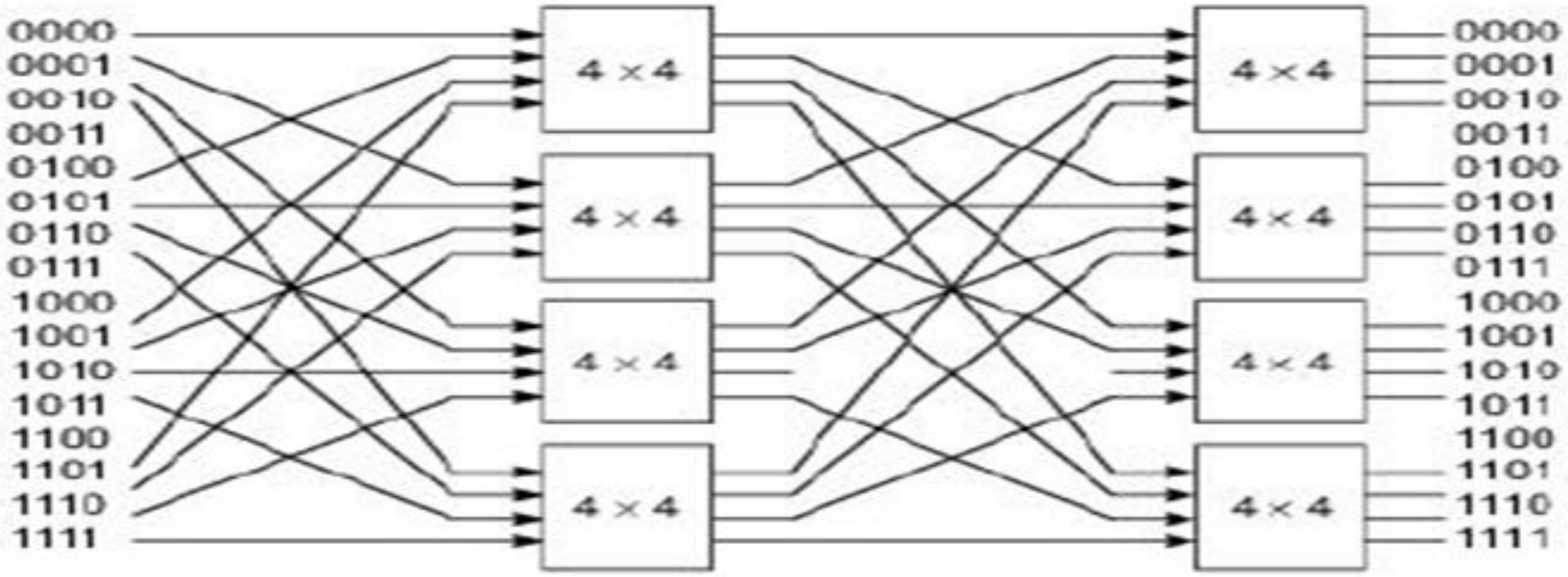


(b) Permutation $\pi_2 = (0, 6, 4, 7, 3)\,(1, 5)\,(2)$ blocked at switches marked F, G, and H

- The Omega network can also-be used to broadcast data from one source to many destinations, as exemplified in Figure

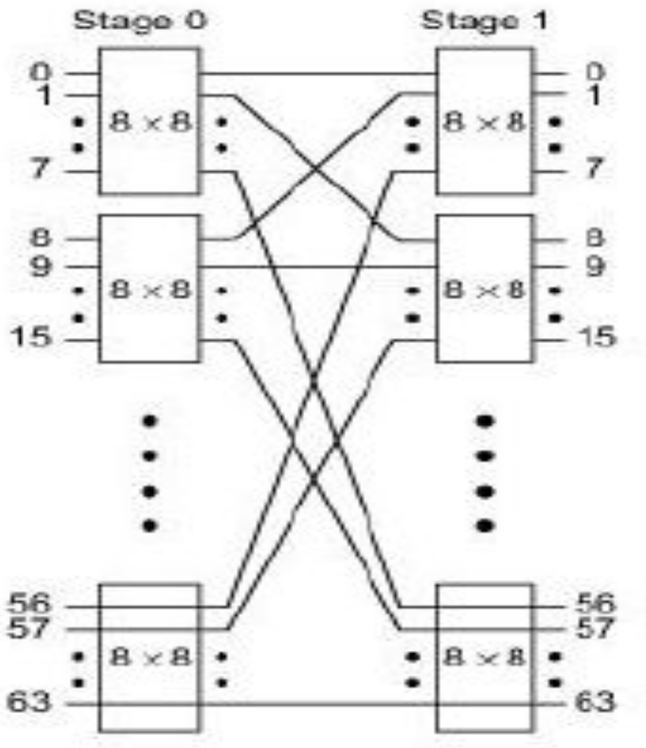


(a) Broadcast connections



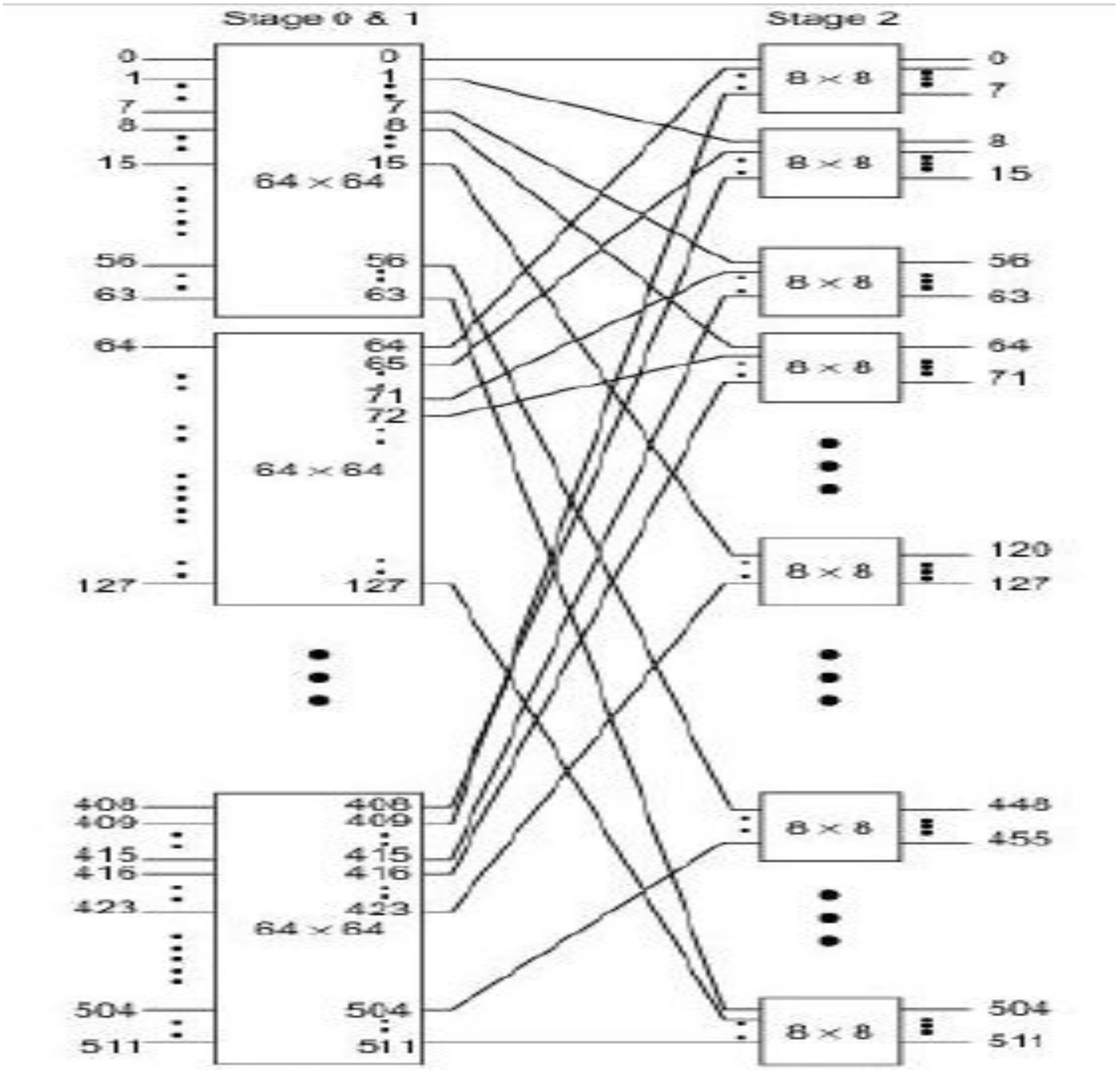
(b) Using four-way shuffle interstage connections

- **Routing In Butterfly Networks :** constructed with crossbar switches as building blocks



(a) Atwo-stage 64 ×64 Butterfly switch network built with 16 8 × 8 crossbar switches and eight-way shufle interstage connections

Modular construction of Butterfly switch networks with 8 X 8 crossbar switches



(b) A three-stage 512 × 512 Butterfly switch network built with 192 8 × 8 crossbar switches

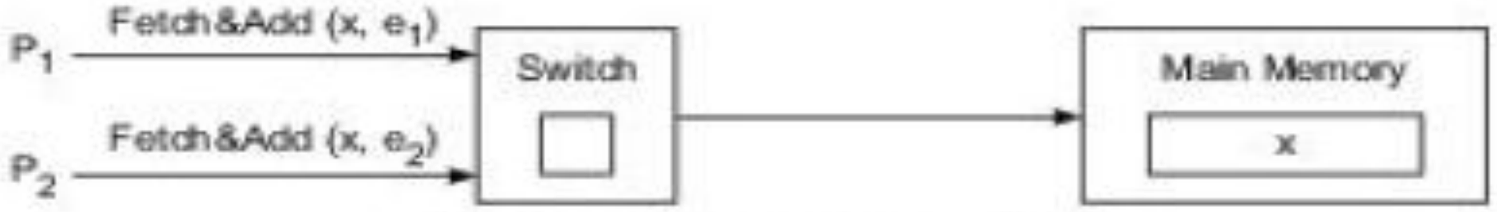
Fetch & add

- **The Hot-Spot Problem** : may appear corresponding to a certain memory module being excessively accessed by many processors at the same time the network performance significantly
- **Fetch & add** : This atomic memory operation is effective in implementing an N-way synchronization with a complexity independent of N.
  - In a Fetch & Add (x, e) operation , x is an
  - integer variable in shared memory
  - e is an integer increment

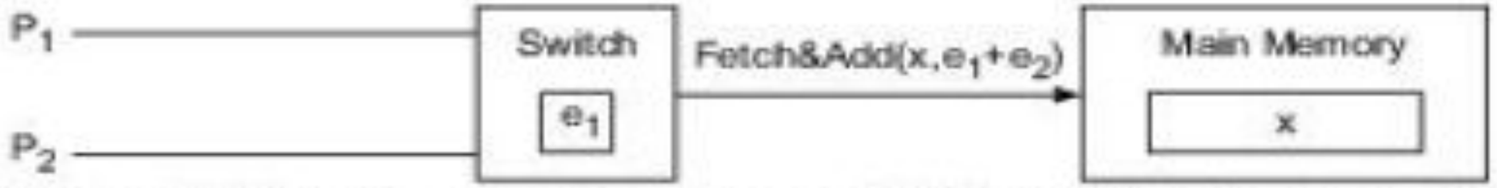
Fetch & add

```
Fetch&Add ($x$, $e$)
        {$temp$   $\leftarrow$   $x$;
             $x$   $\leftarrow$   $temp + e$;
        return   $temp$}
```

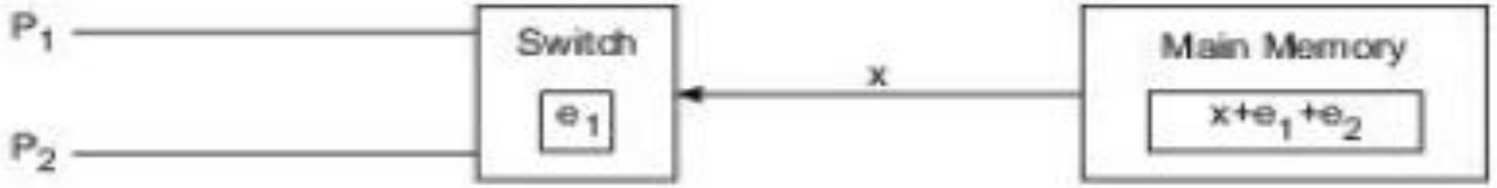
Fetch & add



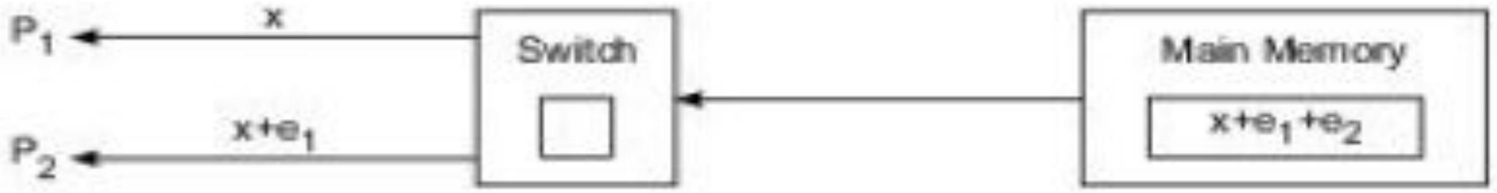
(a) Two requests meet at a switch



(b) The switch forms the sum $e_1 + e_2$, stores e_1 in buffer, and transmits the combined request to memory



(c) The original value stored in *x* is returned to switch, and the new value $x + e_1 + e_2$ is stored in memory



(d) The values *x* and $x + e_1$ are returned to P_1 and P_2 respectively

- Applications and Drawbacks
- Multistage Networks in Real System

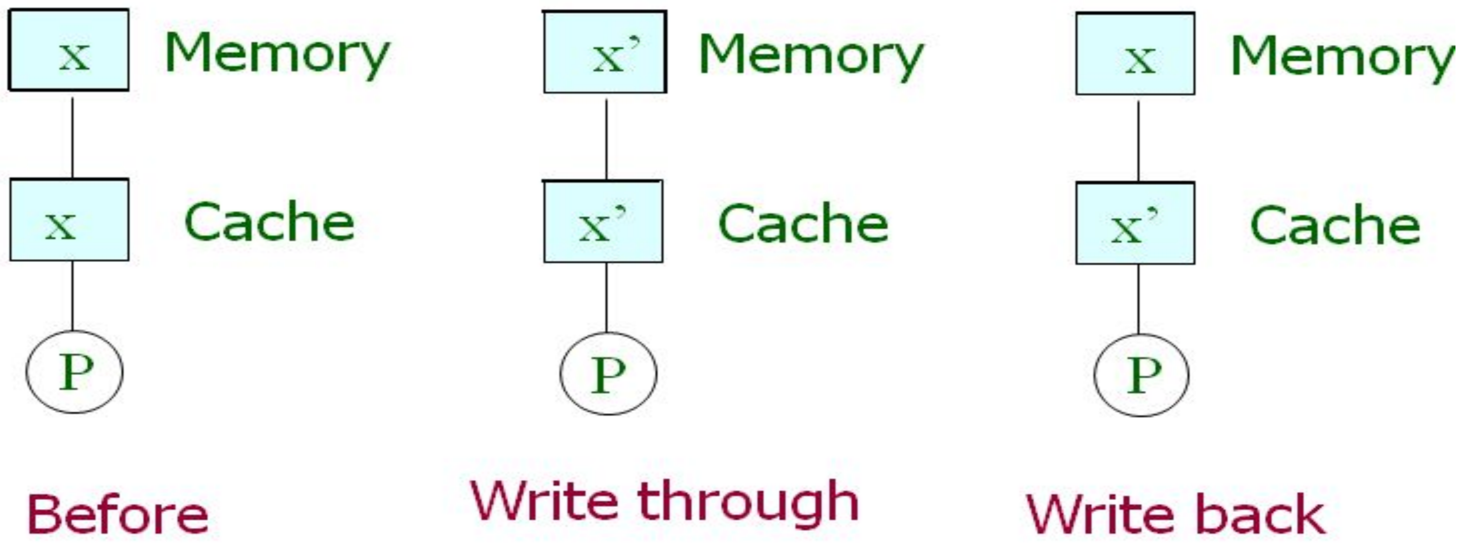
Cache Coherence And Synchronization Mechanisms

1. The Cache Coherence Problem
2. Directory-Based Protocols
3. Snoopy Bus Protocols
4. Hardware Synchronization Mechanisms

The Cache Coherence Problem

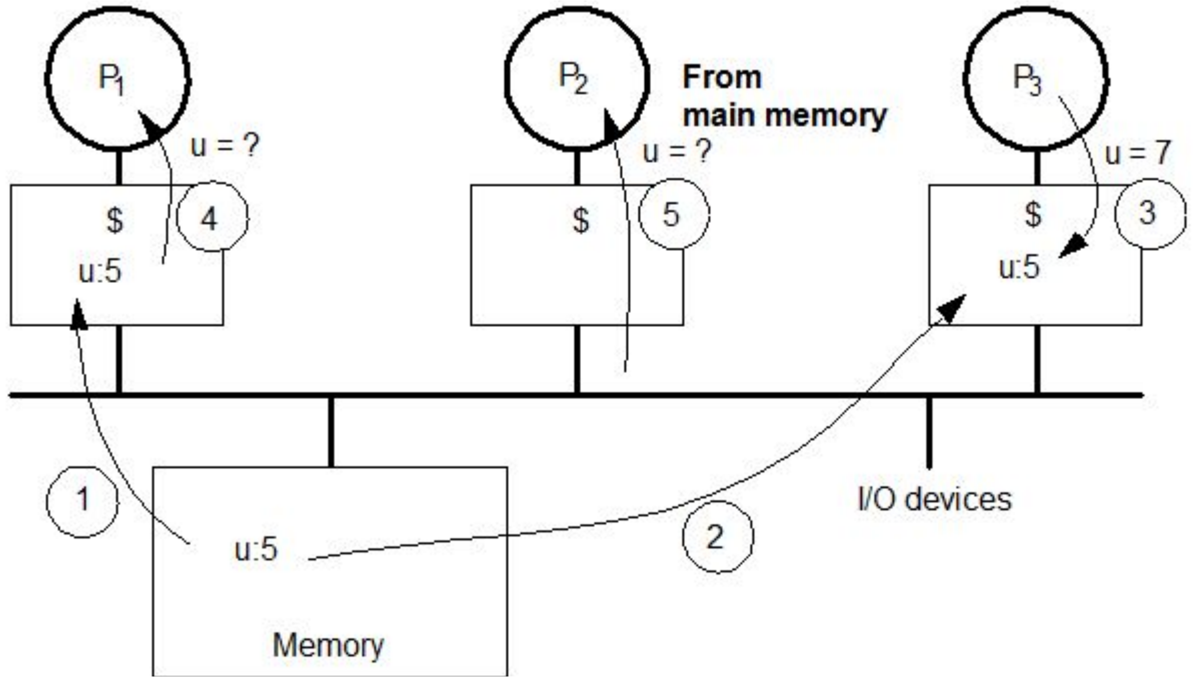
- In a memory hierarchy for a multiprocessor system, data inconsistency may occur between adjacent levels or within the same level
- Caches in a multiprocessor environment introduce the Cache Coherence Problem
- Prevent the problem by maintaining a uniform state for each cached block of data

Condition for Cache Incoherence



- This condition arise when the processor need to share the writable data.
- In both policy write back and write through
- Write back: Write operations are usually made only to the cache. Main memory is only updated when the corresponding cache line is flushed from the cache.
- Write through: All write operations are made to main memory as well as to the cache, ensuring that main memory is always valid

Cache Coherence Problem Example



CACHE COHERENCE & SYNCHRONIZATION MECHANISMS

- Cache protocols with the coherence for coping multi cache inconsistency problem
- Snoopy protocols are designed for bus-connected systems.
- Directory-based protocols apply to network-connected systems.
- Finally, hardware support for fast synchronization.

Inconsistency in Data Sharing

In general, three sources of the problem are identified

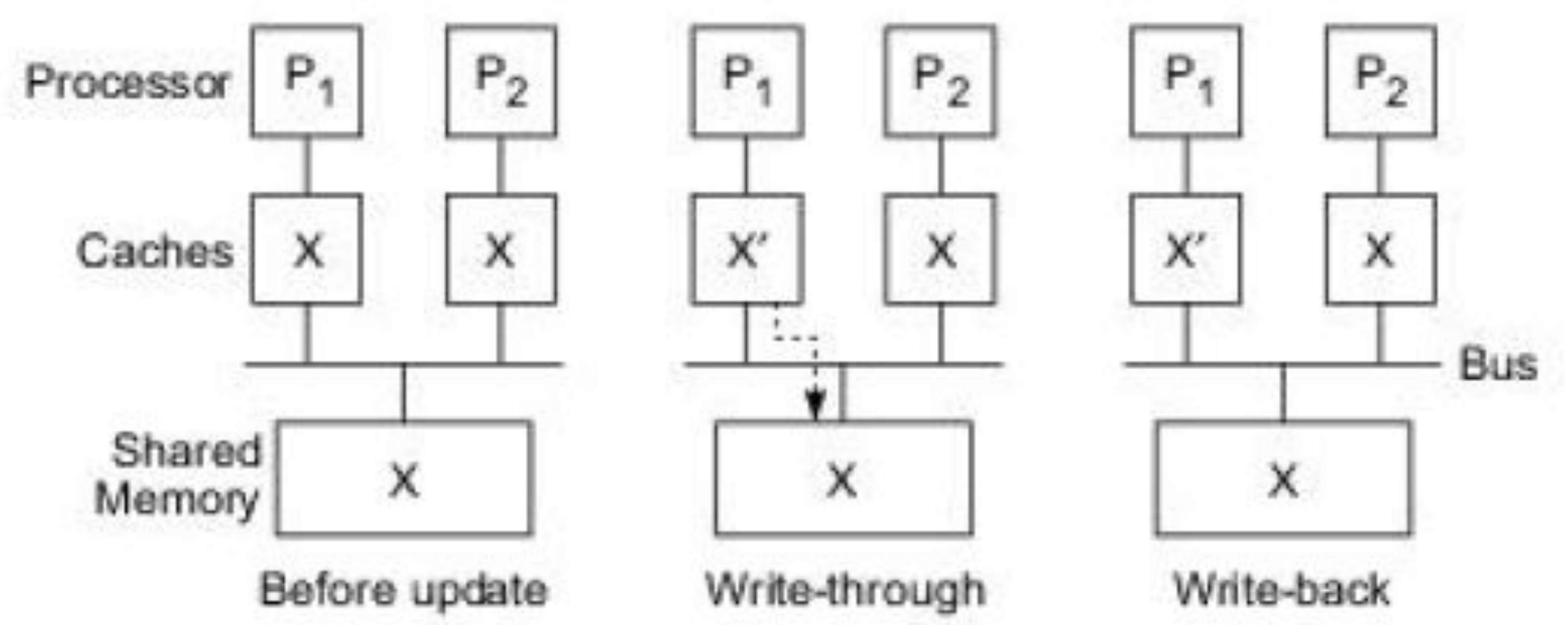
1. Inconsistency in Data sharing
2. Process Migration and I/O
3. Two Protocol Approaches

1. Inconsistency in Data sharing

- The cache inconsistency problem occurs only when multiple private caches are used.
- In general, three sources of the problem are identified:
  - sharing of writable data
  - process migration
  - I/O activity
- inconsistency may also occur when a write-back policy is used
- The main memory will be eventually updated when the modified data in the cache are replaced or invalidated

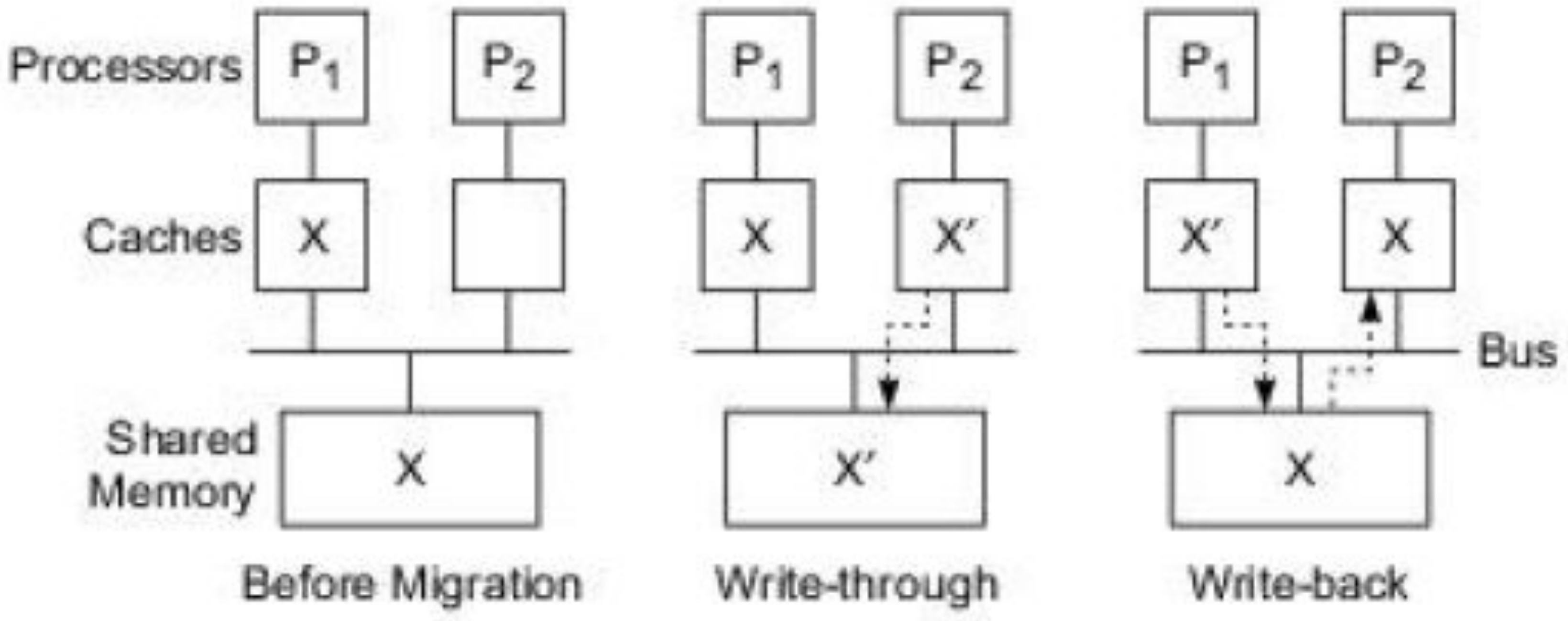
1. Inconsistency in Data sharing

- write-back policy
- Write-through policy

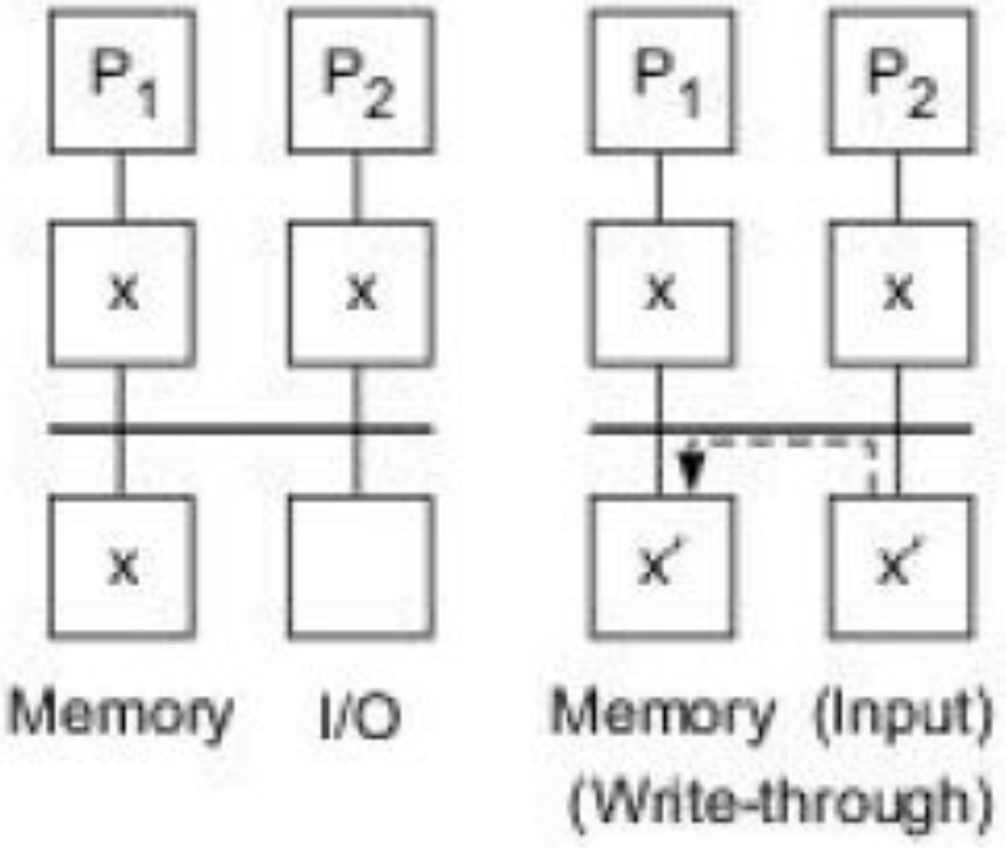


(a) Inconsistency in sharing of writable data

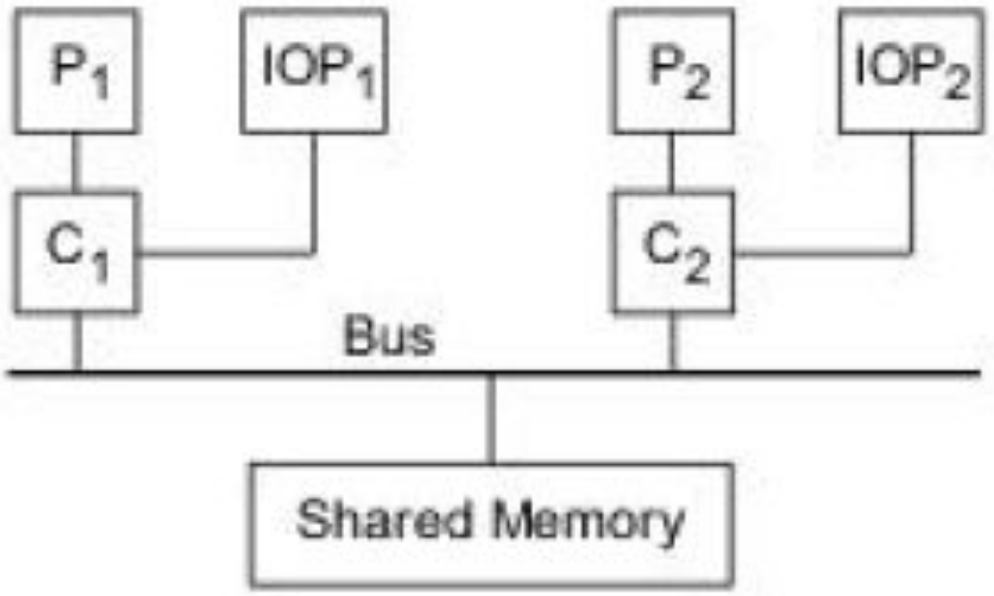
2. Process Migration and I/O



(b) Inconsistency after process migration



(a) I/O operations bypassing the cache



Legends:
P_i (processor i)
IOP_i (I/O Processor i)
C_i (Cache i)

(b) A possible solution

- One possible solution to the I/O inconsistency problem is to
  - attach the I/O processors (IOP1, and IOP2) to the private caches (C1 and C1)

3. Two Protocol Approaches

- Many of the early commercially available multiprocessors used bus-based memory systems.
- A bus is a convenient device for ensuring cache coherence

Directory-Based Protocols

1. Directory Structures
2. Full-Map Directories
3. Limited Directories
4. Chained Directories
5. Cache Design Alternative
6. Shared Cache
7. Non-cacheable Data
8. Cache Flushing

1. Directory Structures

- store information on where copies of cache blocks reside.
- Various directory-based protocols differ mainly in how the directory maintains information and what information it stores
- Central directory containing duplicates of all cache directories

Directory Structures

- This central directory, providing
  - all the information needed to enforce consistency,
  - Is usually very large
  - must be associatively searched.
  - like the individual cache directories

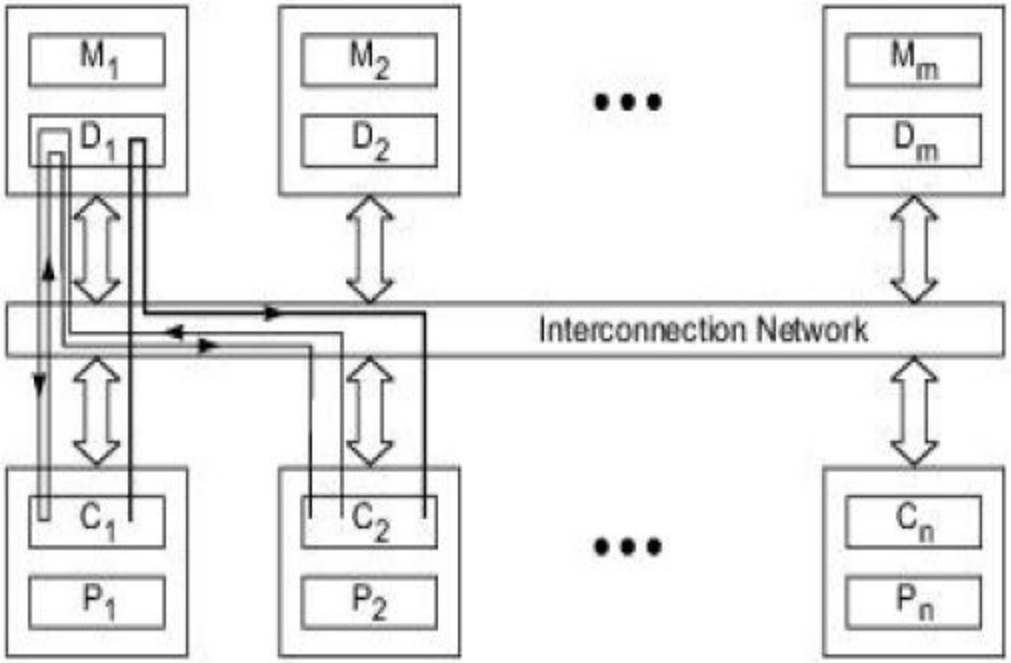
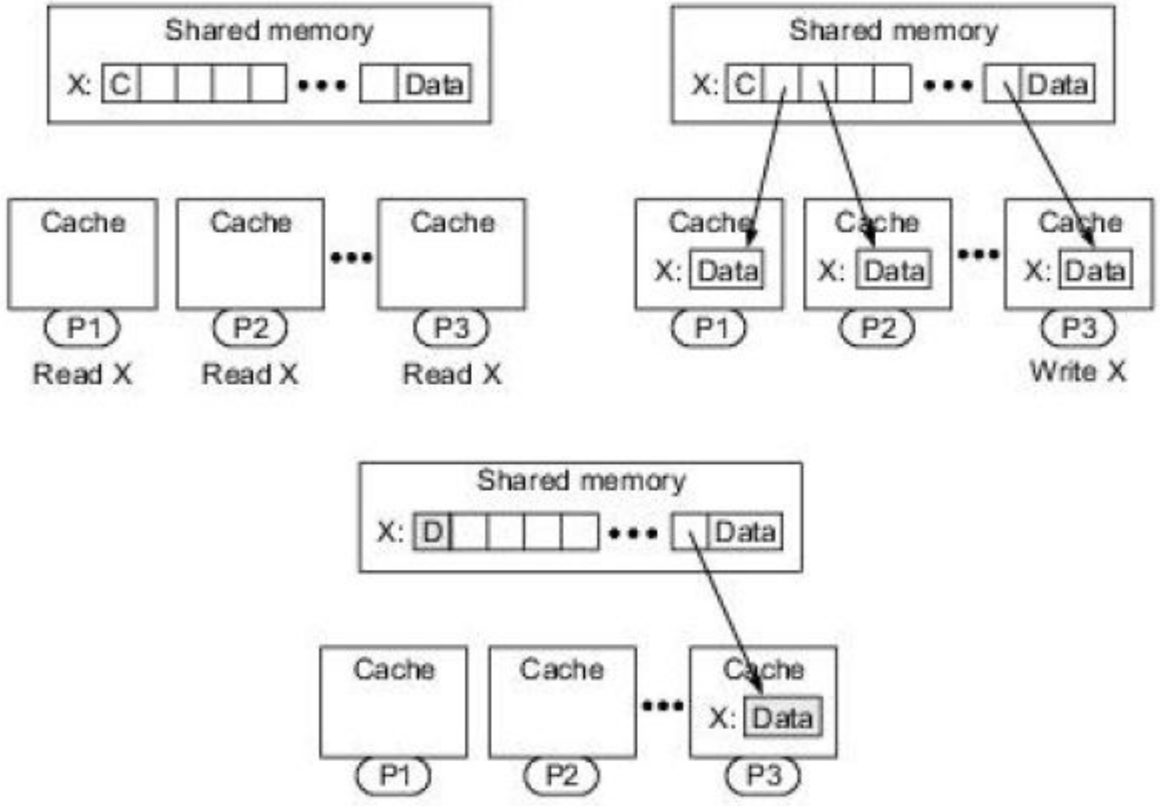


Fig. 7.17 Basic concept of a directory-based cache coherence scheme (Courtesy of Censier and Feautrier, *IEEE Trans. Computers*, Dec. 1978)

2. Full-Map Directories

- implements directory entries with one bit per processor and a dirty bit
- Each bit represents the status of the block in the corresponding processor's cache
  - present or absent
- If the dirty bit is set,
  - then one and only one processor's bit is set and that processor can write into the block.

Full-Map Directories



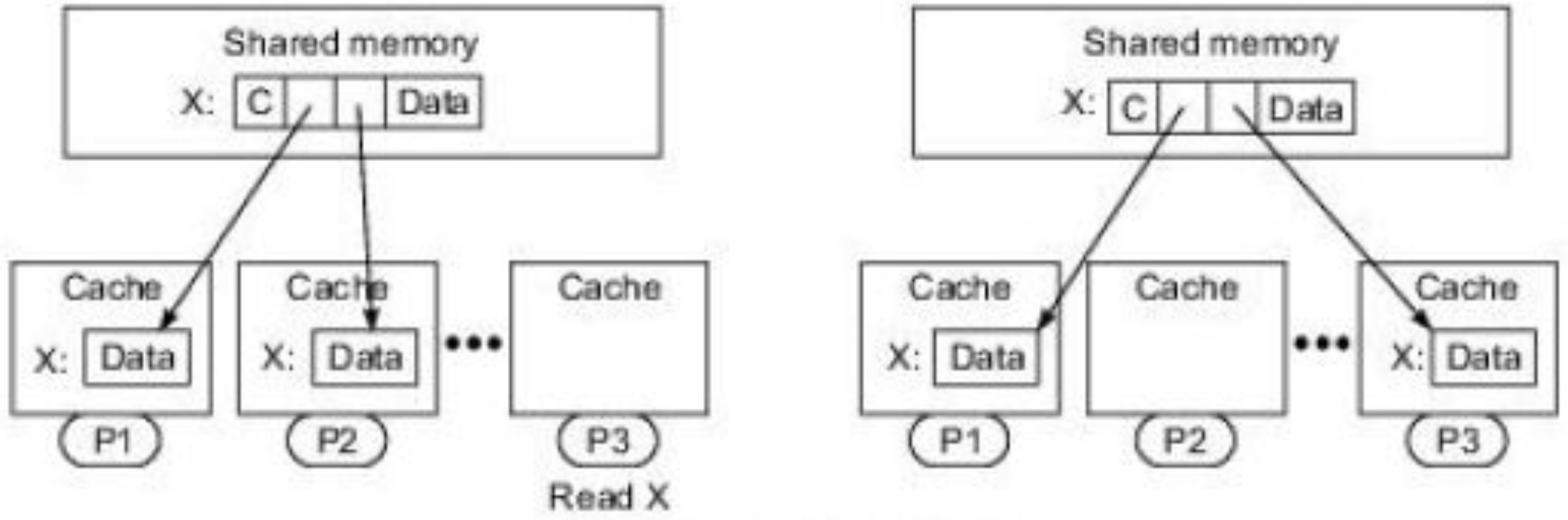
(a) Three states of a full-map directory

- In the first state, location X is missing in all of the caches in the system.
- The second state results from 3 caches requesting X.
- Three pointers (processor bits) are set

3. Limited Directories

- designed to solve the directory size problem
- Restricting the number of simultaneously cached copies of any particular block of data
- A directory protocol can be classified as Dir_i X
    - i is the number of pointers
    - X is NB for a scheme with no broadcast
    - Dir_N NB is full- map scheme without broadcast

Limited Directories

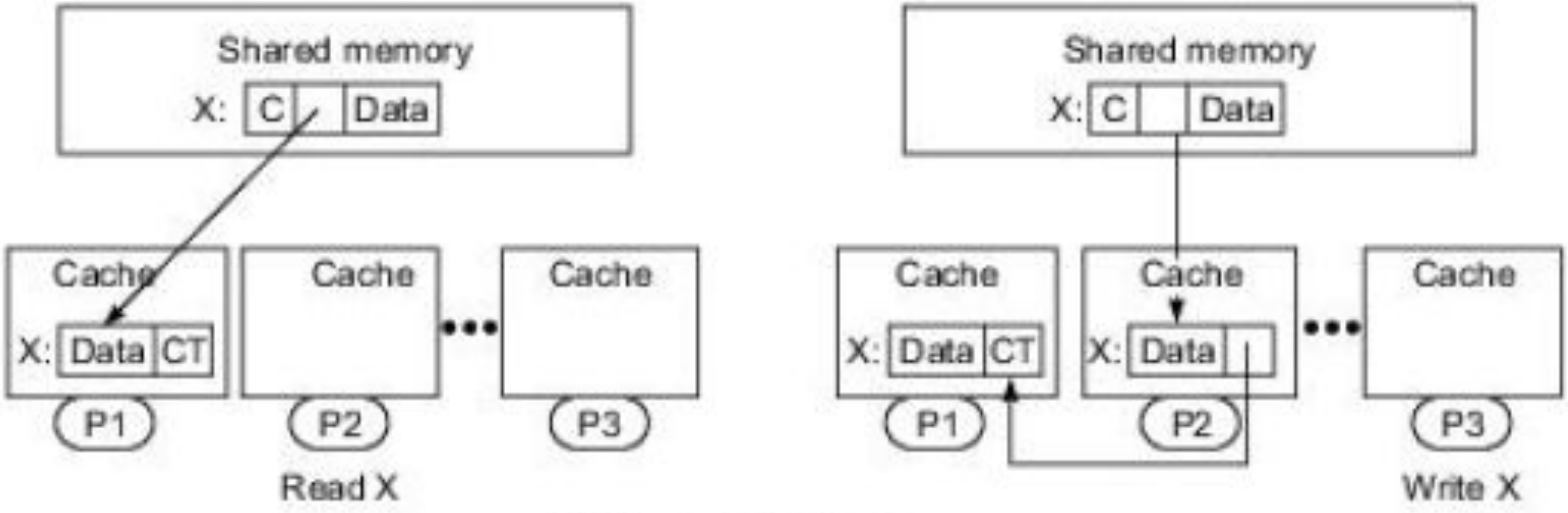


(b) Eviction in a limited directory

4. Chained Directories

- realize the scalability of limited directories without restricting the number of shared copies of data blocks.
- This type of cache coherence scheme is called a chained scheme
  - because it keeps track of shared copies of data by maintaining a chain of directory pointers

Chained Directories



(c) The chained directory

5. Cache Design Alternative

- Each of the design alternatives has its own advantages and shortcomings
  - Shared Cache
  - Non-cacheable Data: The compiler must tag data as either Cacheable or non cacheable
  - Cache Flushing

Hardware Synchronization Mechanisms

- Synchronization is a special form of communication in which control information is exchanged.
- use hardware mechanisms to implement low-level or primitive synchronization operations.
- or use software level synchronization mechanisms
- Semaphares or monitors

Atomic Operation

- Most multiprocessors are equipped with hardware mechanisms for enforcing atomic operations
- Operations- memory read, write, or read-modify-write
- some interprocessor interrupts can be used for synchronization purposes.

- For example, the synchronization primitives, Test & Set (lock) and Reset (lock), are defined below:

```
Test & Set (lock)
    temp <— lock; lock <— l;
return temp
    Reset (lock)
    lock <- 0
```

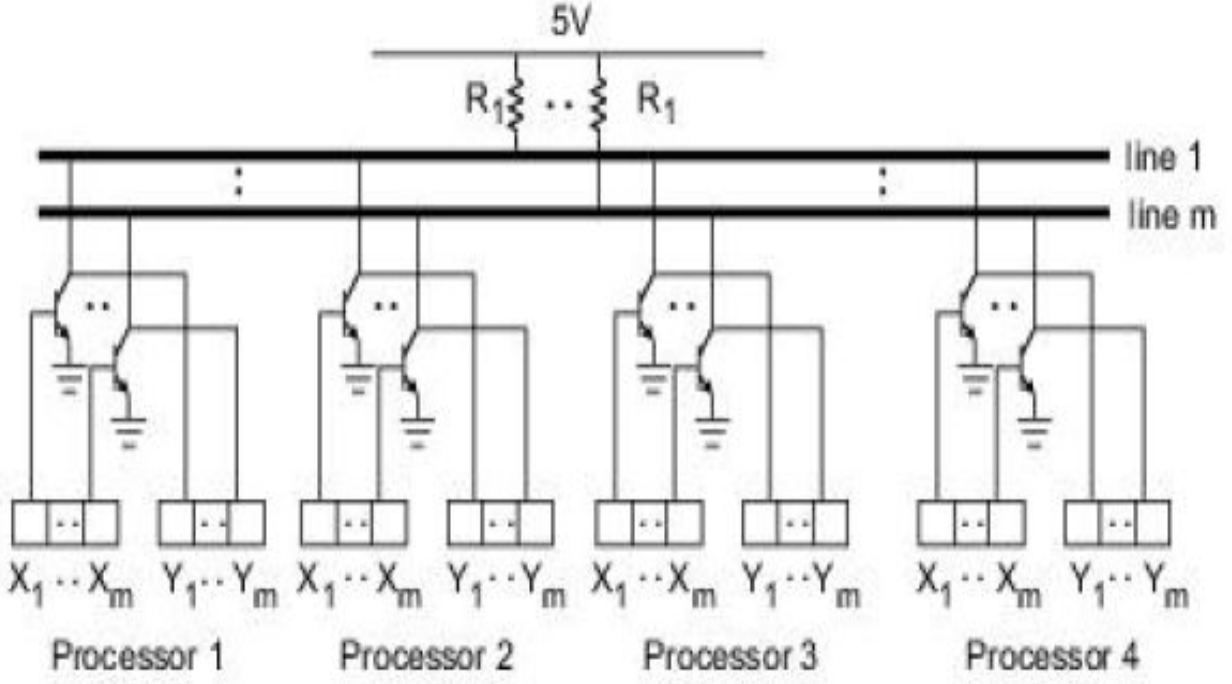
- Test & Set is implemented with atomic read-modify-write memory operations.
- To synchronize concurrent processes, the software may repeat Test & Set until the returned value (temp) becomes 0.
- This synchronization primitive may tie up some bus cycles while a processor enters busy-waiting on the spin lock.

- When the lock is open, it signals all waiting processors through an interrupt.
- Concurrent processes residing in different processors can be synchronized using barriers

Wired Barrier Synchronization

- A wired-NOR logic for implementing a barrier mechanism for fast synchronization
- control vector X= (X1,X2….Xm)
- Monitor vector Y = (Y1,Y2,…,Ym)
- m- barrier lines

- The number of barrier lines depends on the multiprogramming degree and the size of the multiprocessor system.
- Each control bit Xi, is connected to the base (input) of a probing transistor
- The monitor bit Yi checks the collector voltage (output) of the transistor.

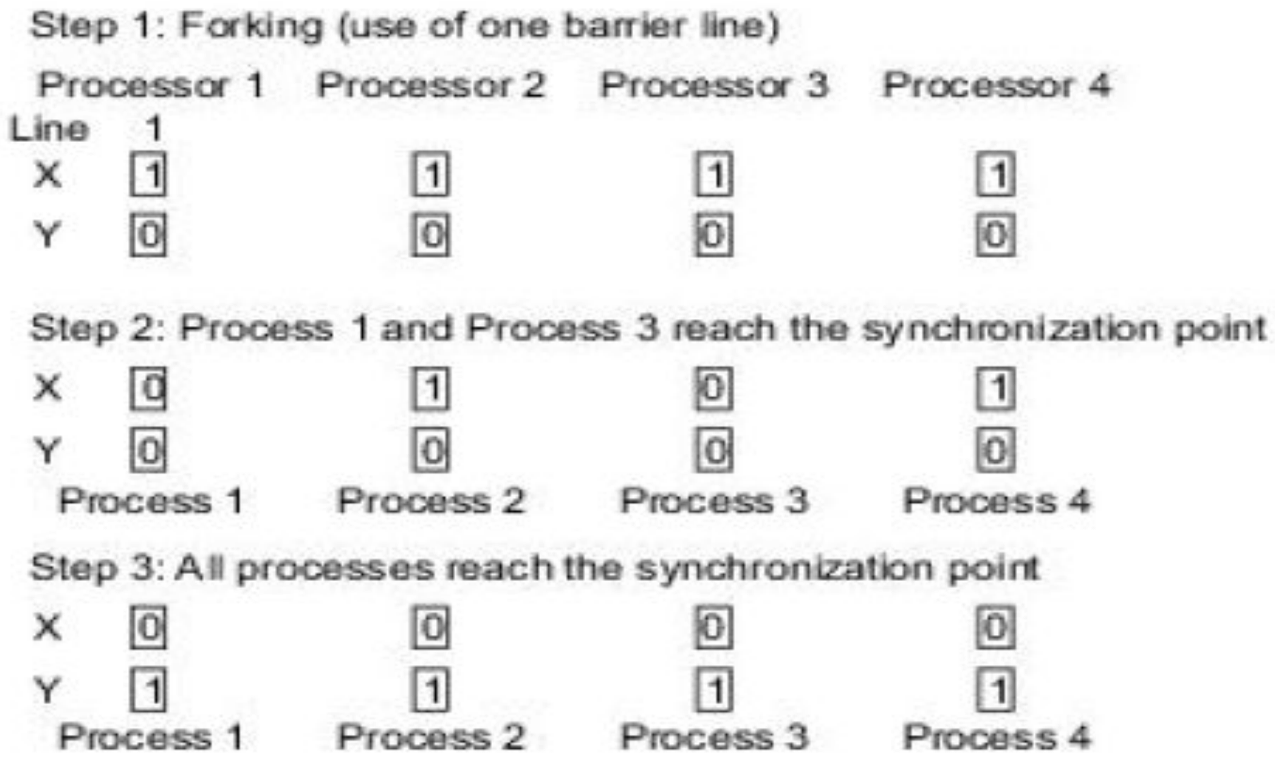


(a) Barrier lines and interface logic

- Each barrier line is wired-NOR to n transistors from n processors.
- Whenever bit Xi is raised to high (1),
- the corresponding transistor is closed, pulling down (0) the level of barrier line i.
- The wired-NOR connection implies that line i will be high (1) only if control bits Xi, from all processors are low (0).

- The bit Xi is set to 1 when a process is initiated and reset to 0 when the process finishes its execution
- all processes finish their jobs,
  - the Xi bits from the participating processors are all set to 0
  - the barrier line is then raised to high (1)
  - signaling the synchronization barrier has been crossed

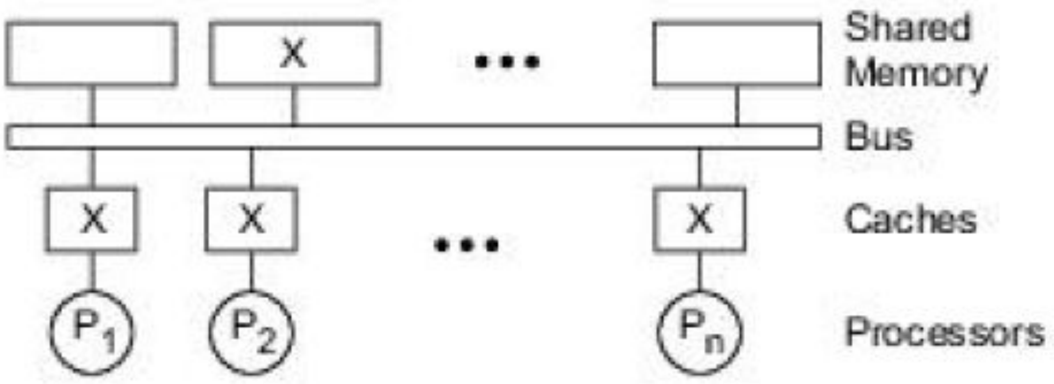
- only one barrier line is needed to monitor the initiation and completion of a single synchronization



(b) Synchronization steps

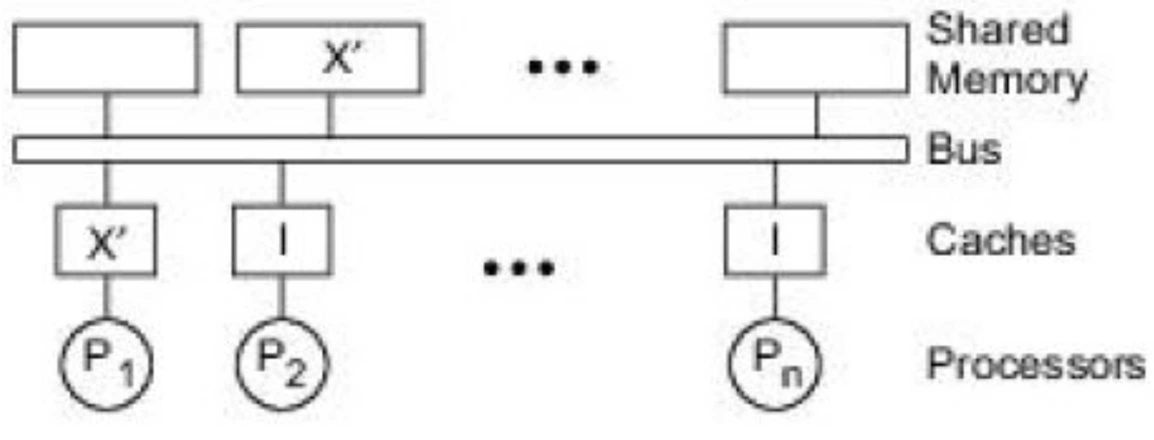
Snoopy Bus Protocols

- Write-invalidate
- Write-update

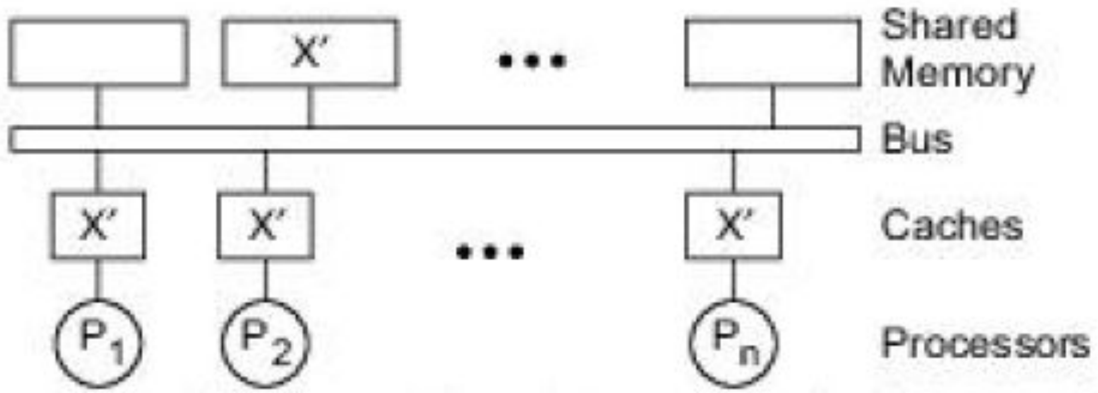


(a) Consistent copies of block X are in shared memory and three processor caches

Snoopy Bus Protocols

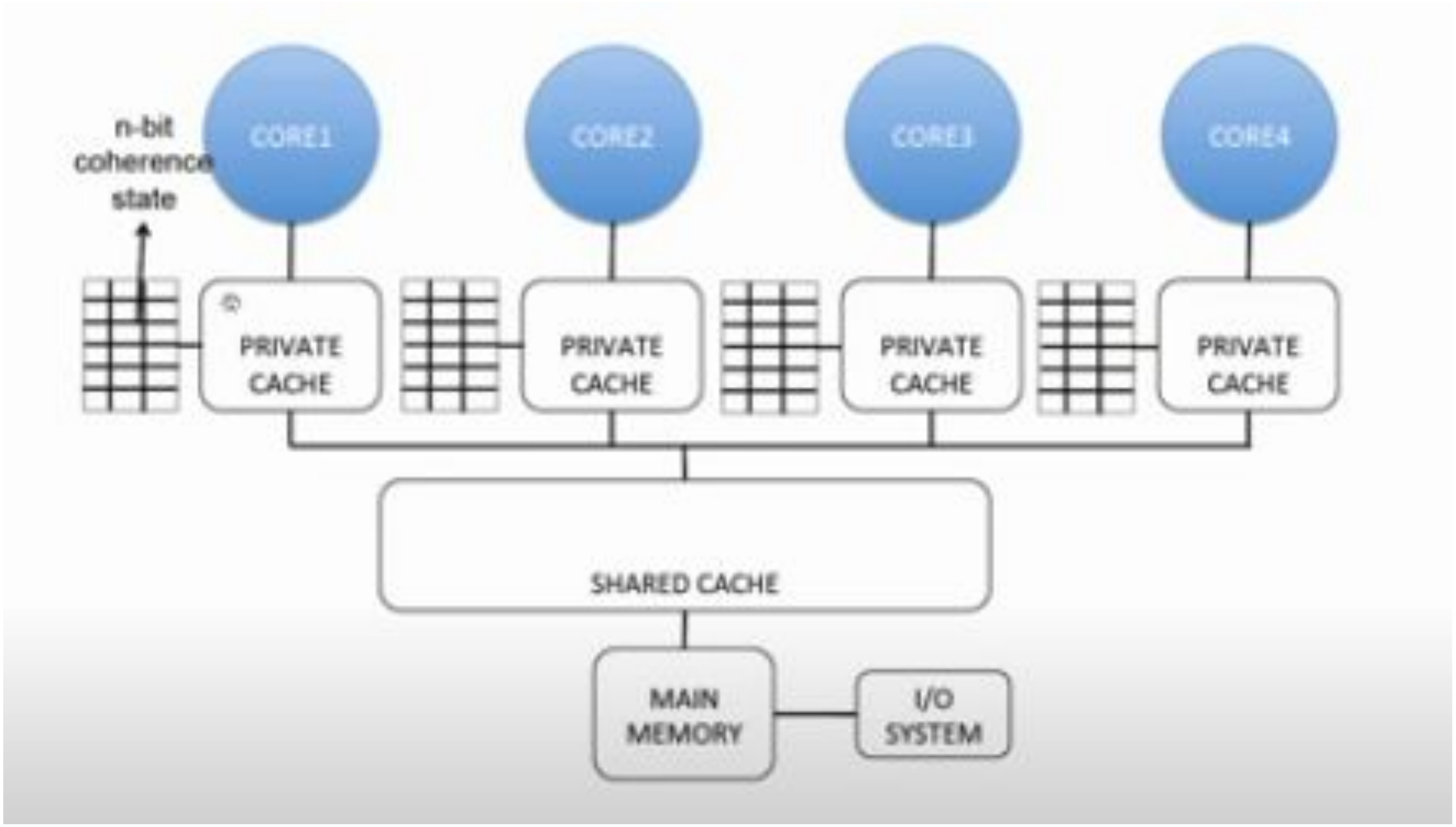


(b) After a write-invalidate operation by P_1

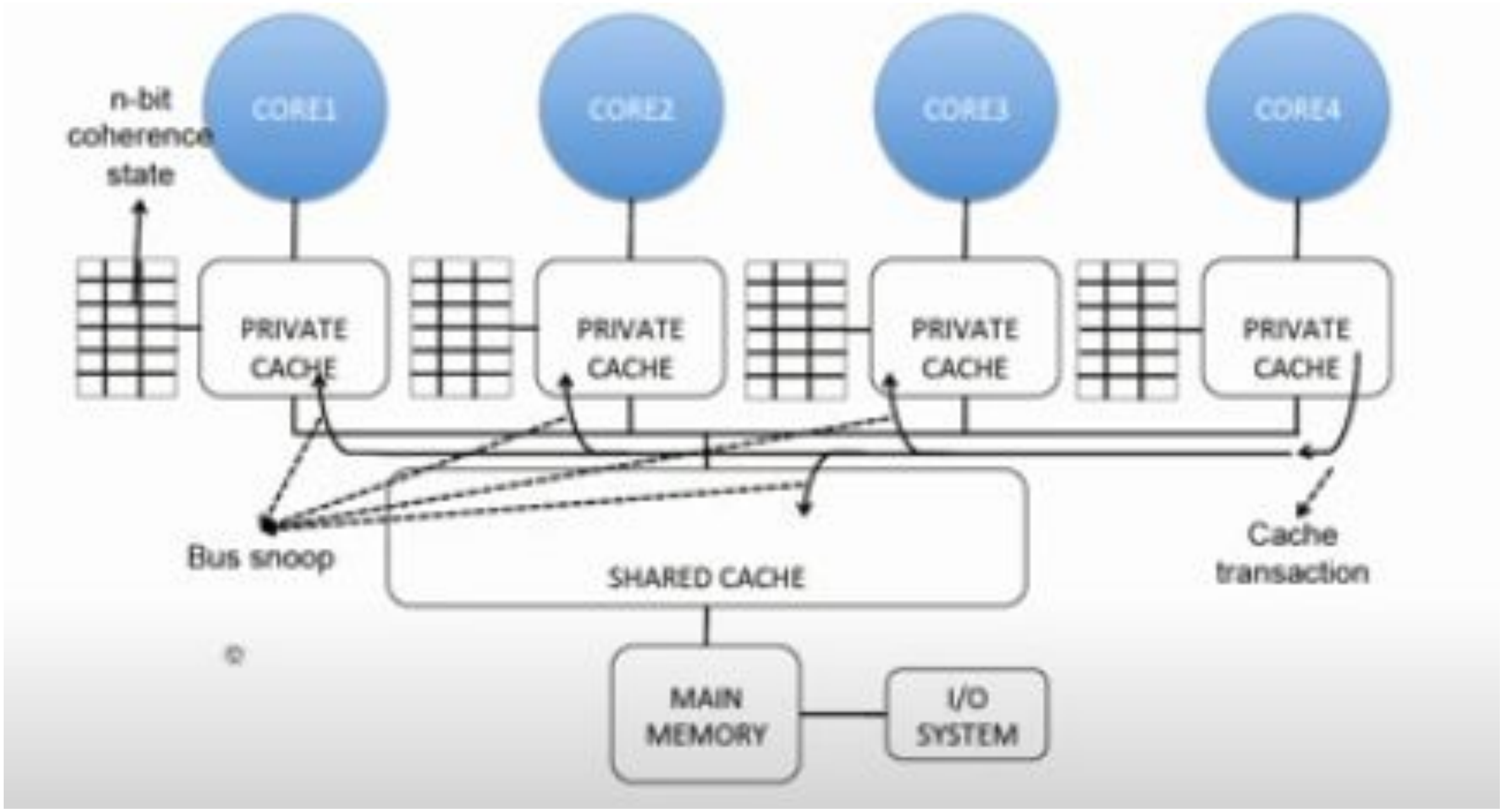


(c) After a write-update operation by P_1

Snoopy Bus Protocols



Snoopy Bus Protocols

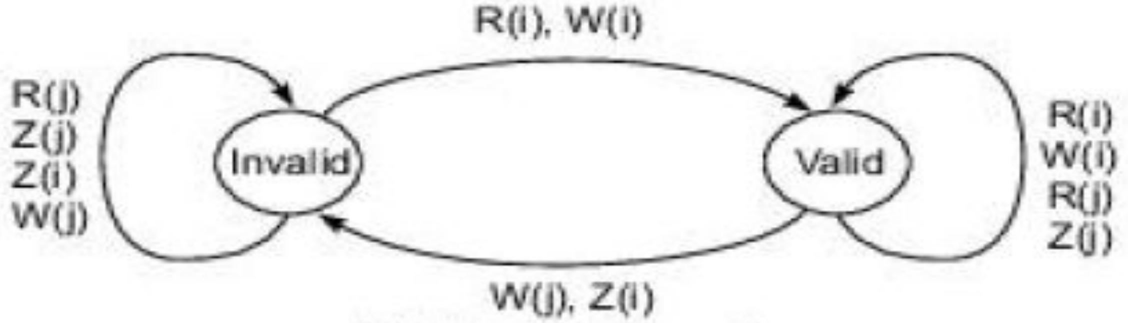


Snoopy Bus Protocols

- Write-Back Caches
- Write-Through Caches
- Write-once Protocol

Write-Through Caches

- The states of a cache block copy change with respect to read , write and replacement operations in the cache.
- A block copy of a write through cache i attached to processor can assume one of two possible cache states: valid or invalid



(a) Write-through cache

W(i) = Write to block by processor *i*.
R(i) = Read block by processor *i*.
Z(i) = Replace block in cache *i*.

W(j) = Write to block copy in cache *j* by processor $j \neq i$.
R(j) = Read block copy in cache *j* by processor $j \neq i$.
Z(j) = Replace block copy in cache $j \neq i$.

- In a *valid* state all processors can read *(R(i), R(j))* safely.
- Local processor i can also write *(W (F1)* safely in a valid state.
- The in valid state corresponds to the case of the block either being invalidated or being replaced *(Z(i) or Z(j)).*

Write-Back Caches

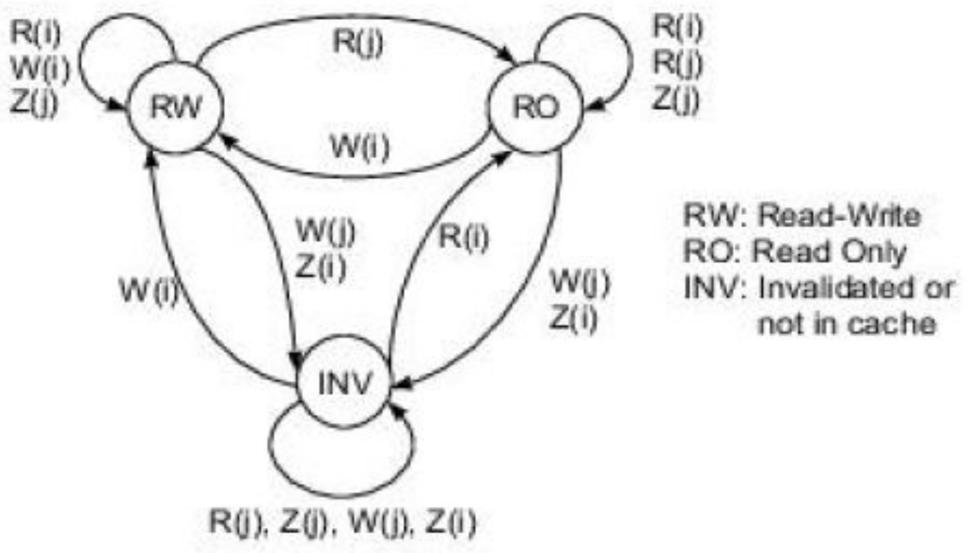
three-state coherence scheme

RW (Read-Write)

RO (Read-Only)

The INV (invalidated or not-in-cache) cache state

Write-Back Caches



W(i) = Write to block by processor *i*.
R(i) = Read block by processor *i*.
Z(i) = Replace block in cache *i*.

W(j) = Write to block copy in cache *j* by processor $j \neq i$.
R(j) = Read block copy in cache *j* by processor $j \neq i$.
Z(j) = Replace block copy in cache $j \neq i$.

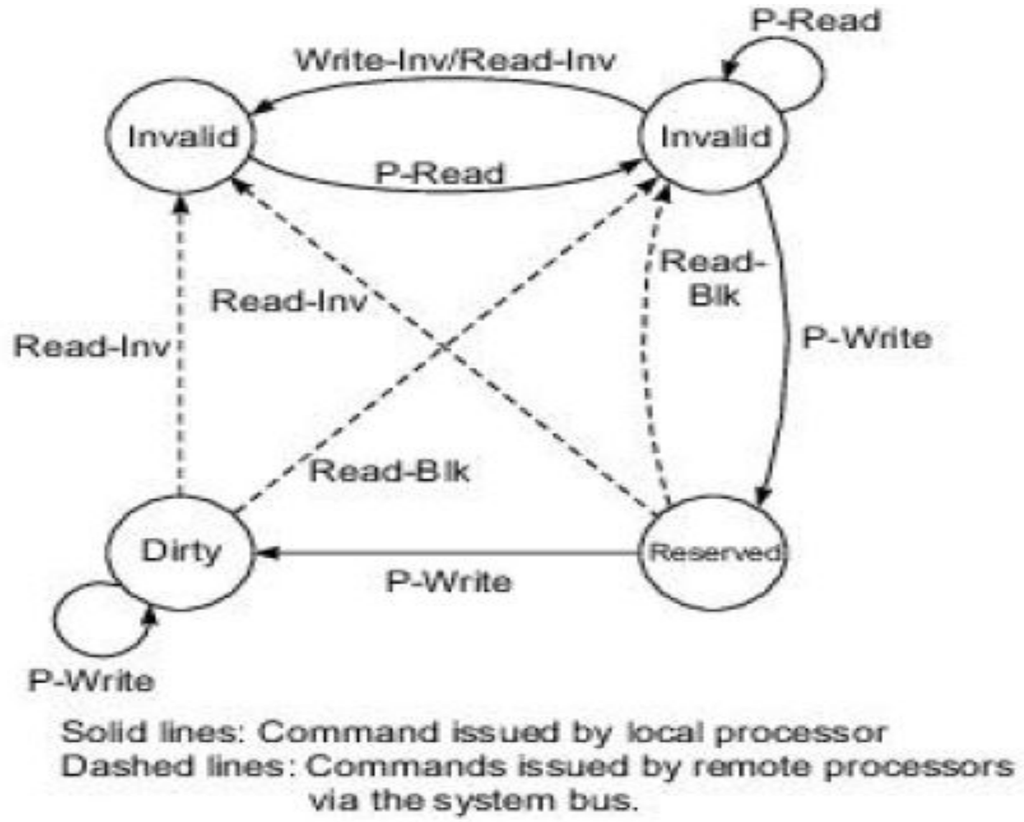
(b) Write-back cache

State-transition graph for a cache block using write –invalidate snoopy protocol

Write-once Protocol

- combines the advantages of both write-through and write-back invalidations.
- In order to reduce bus traffic, the very first write of a cache block uses a write-through policy.
- This will result in a consistent memory copy while all other cache copies are invalidated.
- After the first write, shared memory is updated using a write—back policy.

- Four cache states are given here:


P-Read
Write-Inv/Read-Inv
Invalid
Invalid
P-Read
Read-Inv
Read-Inv
Read-Blk
P-Write
Read-Blk
Dirty
Reserved
P-Write
P-Write
Solid lines: Command issued by local processor
Dashed lines: Commands issued by remote processors via the system bus.

Goodman's write-once cache coherence protocol using the write invalidate policy on write-back caches (Adapted from James Goodman 1983, reprinted from Stenstrom, *IEEE Computer*, June 1990)

- Valid :The cache block, which is consistent with the memory copy, has been read from shared memory and has not been modified.
- Invalid :The block is not found in the cache or is inconsistent with the memory copy.
- Reserved :Data has been written exactly once since being rend' from shared memory.
  - The cache copy is consistent with the memory copy, which is the only other copy.

- Dirty: The cache block has been modified (written) more than once, and the cache copy is the only one in the system
  - thus inconsistent with all other copies

4.Cache Event and Action

- The memory-access and invalidation commands trigger the following events and actions:
  - Read-miss: When a processor wants to read a block that is not in the cache
  - Write-hit: If the copy is in the dirty or reserved state, the write can be carried out locally

- Write-miss: When a processor Fails to write in a local cache, the copy must come either from the main memory or from a remote cache with a dirty block.
- Read-hit: Read-hits can always be performed in a local cache without causing a state transition or using the snoopy bus for invalidation.
- Block Replacement.‘ If a copy is dirty it has to be written back to main memory by block replacement.

Multilevel Cache Coherence

- To maintain consistency among cache copies at various levels
- Consistency of caches at different levels

Protocol Performance issue

- depends heavily on the workload patterns and implementation efficiency.
  - is to reduce bus traffic,
  - reducing the effective memory-access time
- The block size is very sensitive

- For a uniprocessor system
  - bus traffic and memory-access time are mainly contributed by cache misses.
- For larger caches
  - the data pollution point appears at a larger block sire.
- For a system requiring extensive process migration or synchronization.
- the write-invalidate protocol will perform better