Multiprocessors Motivations:

- To increase computing power
- advanced single processors are reaching diminishing returns
- Improve reliabi;ity of systems

Different Models:

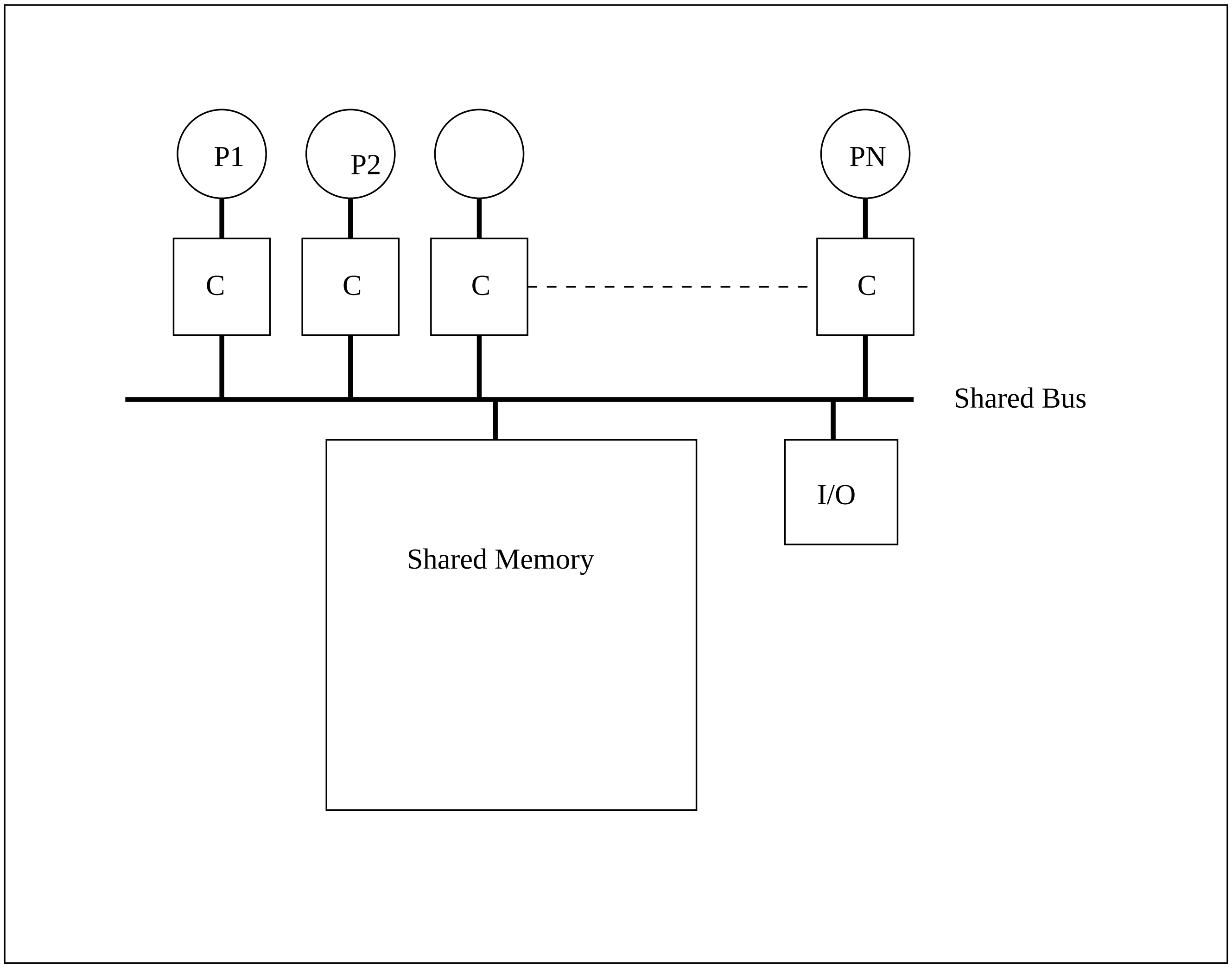
- **SISD:** Single Intruction Single Data Stream. This is the single processor.
- **MISD:** Multiple Instruction Streams, Single Data Streams. No machine of this type.
- **MIMD:** Multiple Instruction Streams, Multiple Data Streams. Uses multilple of single processors
- **SIMD:** Single Instruction Stream, Multiple Data Streams. Uses vector operations with one instruction is fed to multiple processors using different data streams.

MIMD

-Can use off the shelf processors , can function as single user machine or high performance as in mutiprogrammed machines

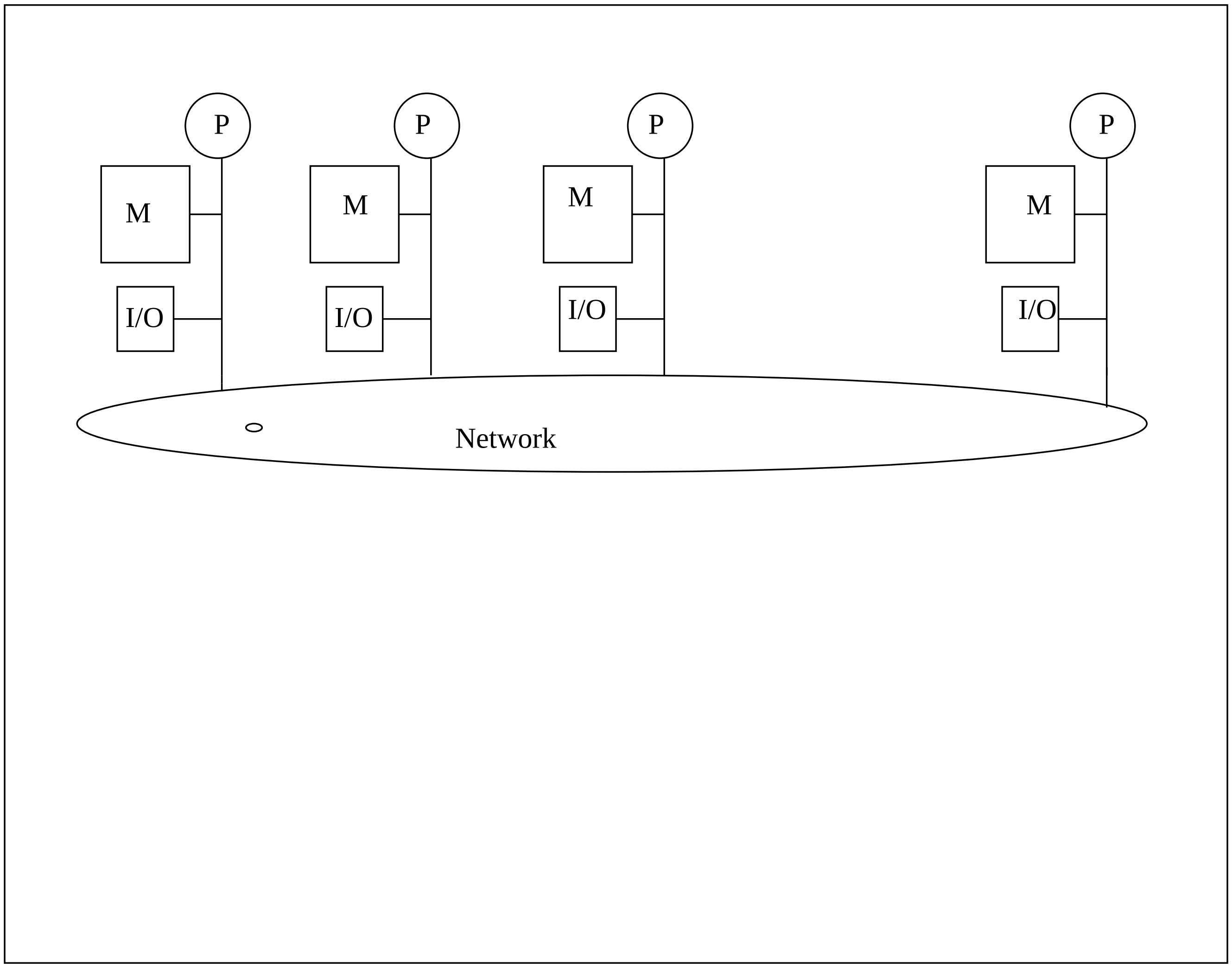
Types of MIMD:

1-Shared Memory Architecture. **UMA:** Uniform Memory Access uses centeralized shared memory.



2-NUMA: Non uniform Memory Access. Used in Distributed Memory Processors.

Distributed Memory Systems



It consists of individual nodes and an Interconnection Network.

Each node has processor, cache, local memory and I/O.

Has the following advantages-

- Cost effective to scale memory bandwidth
- reduces latency of memory accesses to local memory

the disadvantages:-

- communication between processors is more complex

Models for Communication and Memory Architectures

1-Shared Memory:
Processors Communicate with shared address space.
Easy for small scale machines
Advanrages:-

- Model of choice for single processors, and small scale multiprocessors
- low latency
- easy to program
- easy to use cache

2- Message Passing:
Processors communicate with messages and have private memories.
advantages:

- less hardware and easier to design
- scale better

Performance Metrics for Communication Mechanisms

1- Bandwidth:

-Limited by processor, memory and interconnection bandwidth.

-most likely is limited by the communication mechanism

2-Latency:

-affects performance and programming of multiprocessors

-processor might have t wait

-must hide latency (overlap message with computation).

latency hiding needs software support and depends on the application (effectivness)

Advantages of Shared Memory Communications

- Compatibility
- Easy to programm and simllify compiler design
- Low overhead for communication (not using operating system)
- Ability to use caching to reduce latency

Advantages of Message Passing

- Simpler hardware (no cache coherency)
- Communication is explicit forcing programmers and compilers to pay attention to it.

Challanges of Parallel Processing

- **1-Limited parallelism in programs**
  Amdahl's law: performance improvements is limited by small part that cannot be executed in parallel.
  Example: what fraction of original computation can be sequential for having 80 speed up of 100 processors.
  $speed_up = 1 \div (fraction_enhanced \div (seed_upenhanced) -$
  $(1 - fraction_enhanced))$
  $80 = 1 \div (n \div 100 + (1 - n))$
  n=.9975, only .25% is allowed to be sequential.

- **Communication Overhead**
  Example: Assume 32 processor machine that has 2000 ns communication overhead. Processor cycle time = 10 ns, if base CPI=1, find how much faster if the machine has no communication overhead versus .5%.
  No over head CPI =1
  with communication $= 10 + 2000 \times .005 = 20$ ns, two times faster

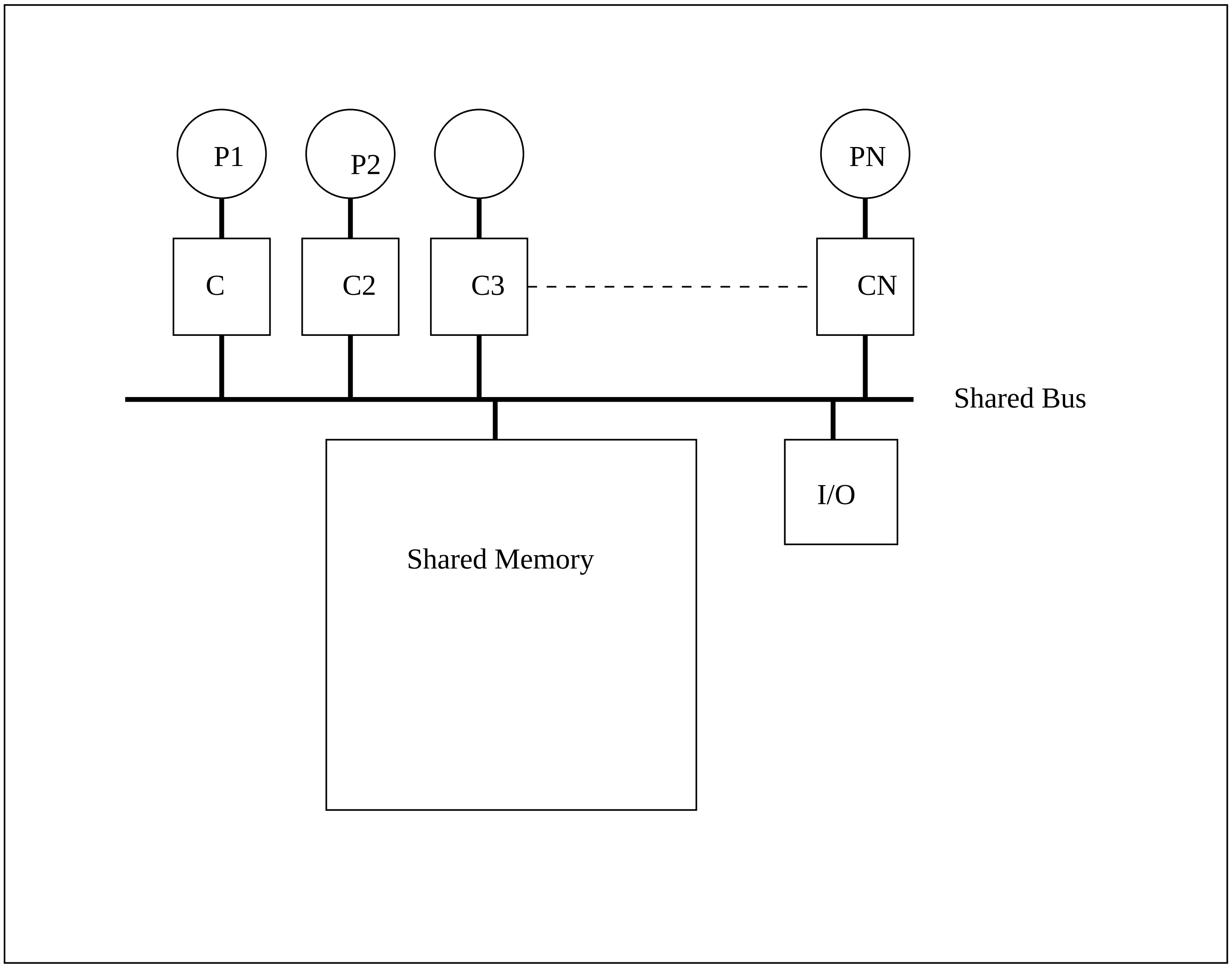
Centralized Shared Memory Architectures

Feaures:

-Small scale multiprocessor system

-Using one bus

-Need large caches to reduce bus requirements

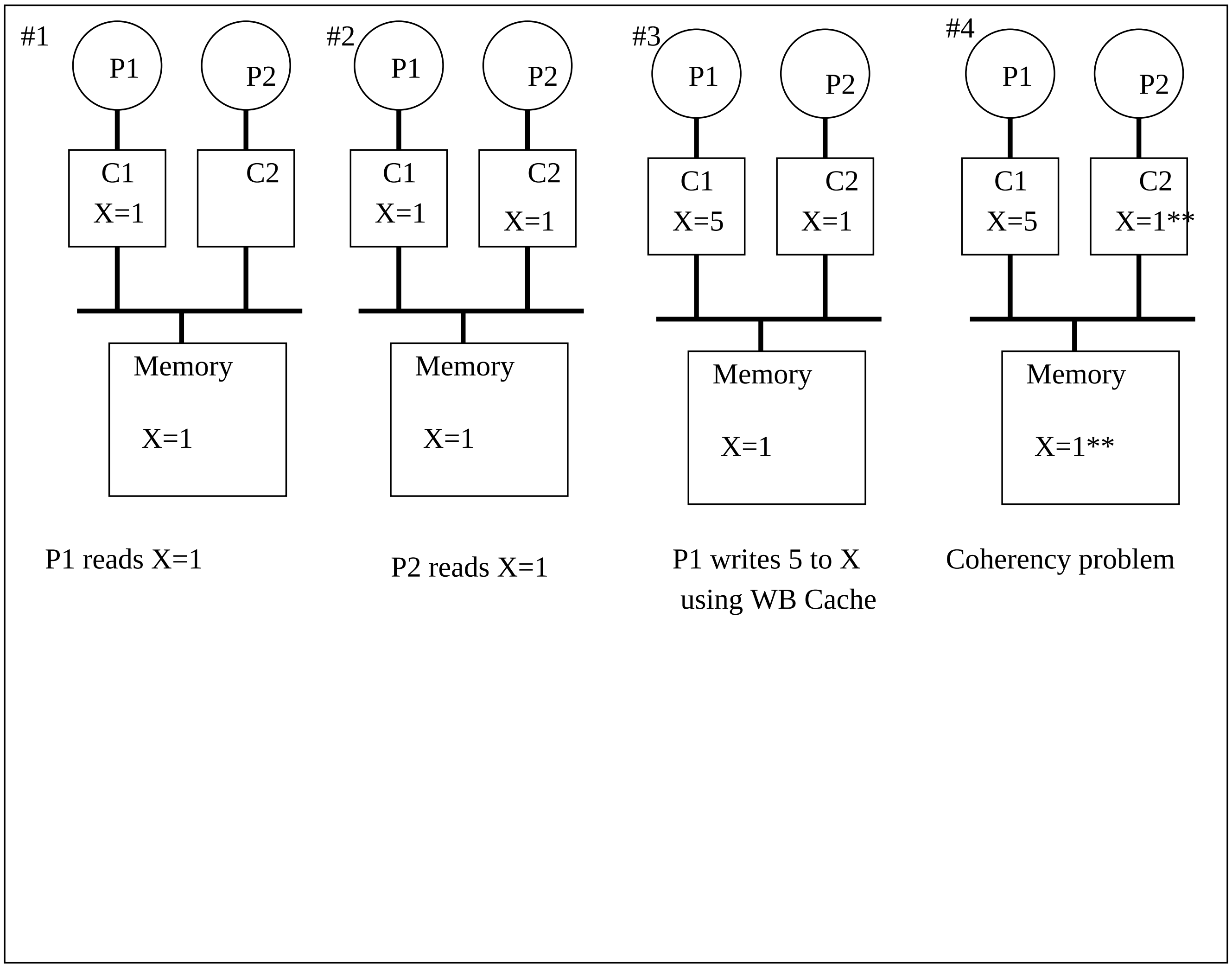


Data types in Multiprocessor system

- 1-**Shared Data:** Used by multiple processors to communicate (readwrite)
- **2-Private Data:** Used by a single processor (readwrite)

Private and shared data can be cached by local processors.
Cache helps to reduce demand for bus, and reduce latency to access memory.
Processor uses private data in its own cache the same way as a single processor uses its data on cache (dirty,..)
Shared data caching should be handeled differently.
Multiple copies of a shared data item could exist on multiple caches. This reduces bus contention and reduces access time.

Data Types:


#1
P1
P2
C1
X=1
C2
Memory
X=1
P1 reads X=1
#2
P1
P2
C1
X=1
C2
X=1
Memory
X=1
P2 reads X=1
#3
P1
P2
C1
X=5
C2
X=1
Memory
X=1
P1 writes 5 to X using WB Cache
#4
P1
P2
C1
X=5
C2
X=1**
Memory
X=1**
Coherency problem

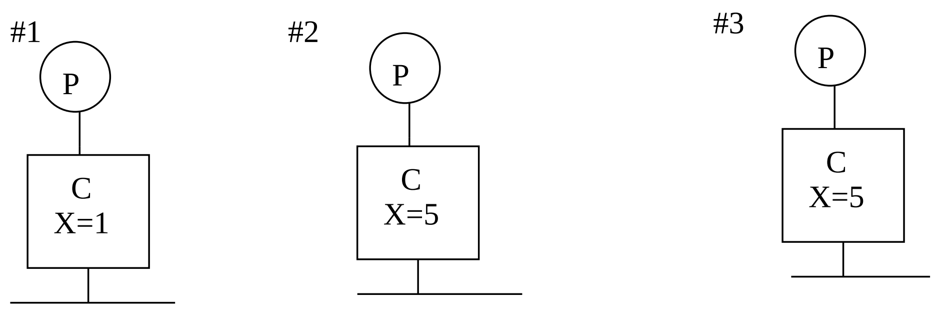
Memory Coherency Requirements

Memory is coherent if any read will return the most recent written value of data.
The following operations satisfy coherency:

- 1-read by a processor, follows a write by same processor to same location , with no writes by other processors to that location.
  last write is from same processor

- If the write to X is from other processor, Processor must read X=5 for the memry to be coherent. (processor must read value of last write even if it occurs at other processor cache)

Memory Coherency Requirements



P reads X

P writes 5 to X

P readS X=5 is coherent

Memory Coherency Requirements

- Writes to same location must be serialized. If P1 writes 5 to X, tthen P5 writes 7 to X, then P1 should read X=7 (not 5). [If P1 misses cache, P5 hits , we must still have X=7 as final value]

BASIC SCHEMES FOR COHERENCY

Need a protocol to maintain coherence for multiple processors. This is needed to track the state of data shared between different processors.

Types of Coherence Protocols

Directory Based:

Status of blocks is kept in the directory (which cache has the most recent value) it is centralized

Snooping:

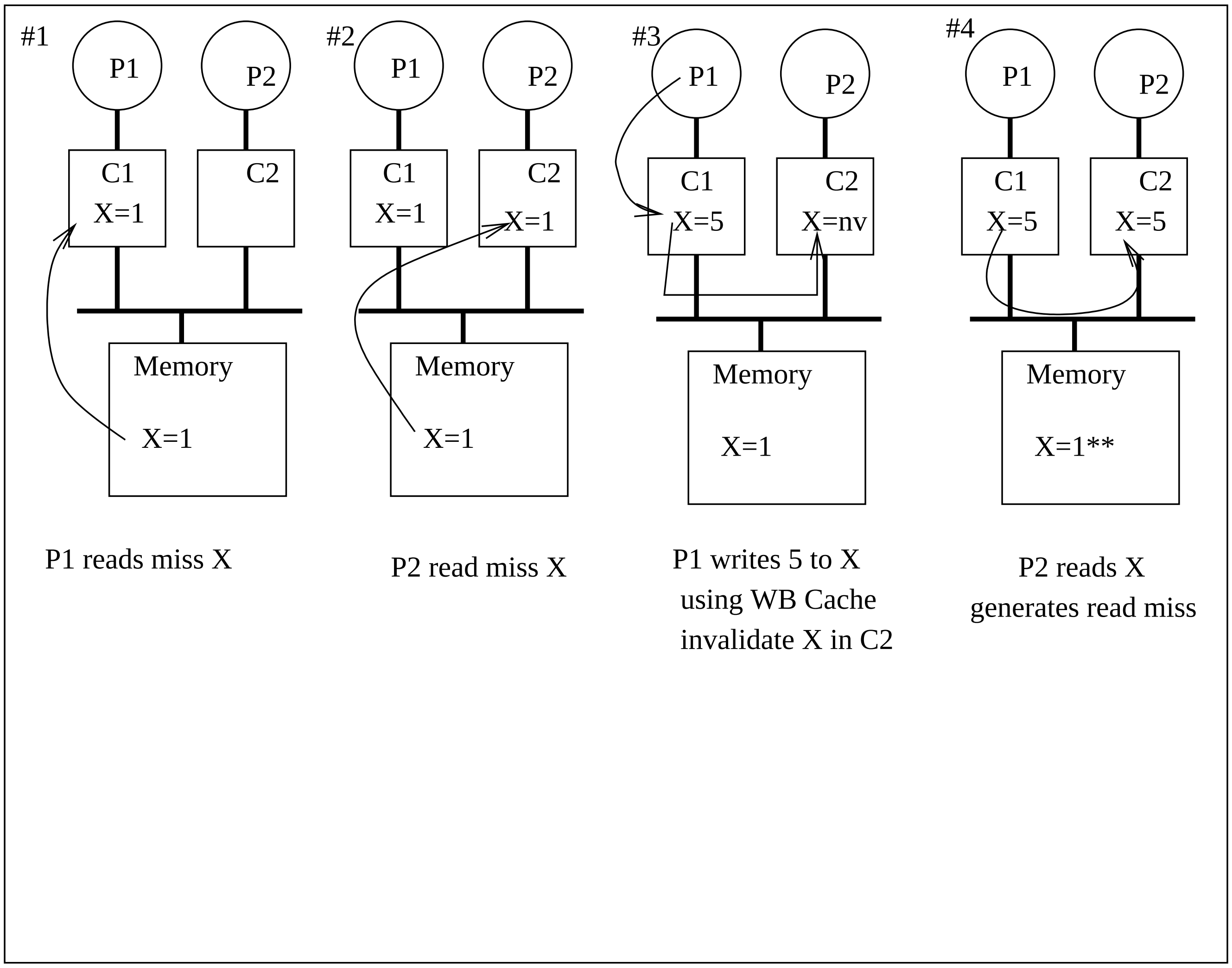
Every cache has status of shared data block. All caches monitor or snoop on the bus to determine whether or not they have a copy of the block.

Two Ways to maintain coherence in caches

- 1-**Write Invalidate:** The protocol maintain an exclusive access to a data item through invalidating all copies on other caches.

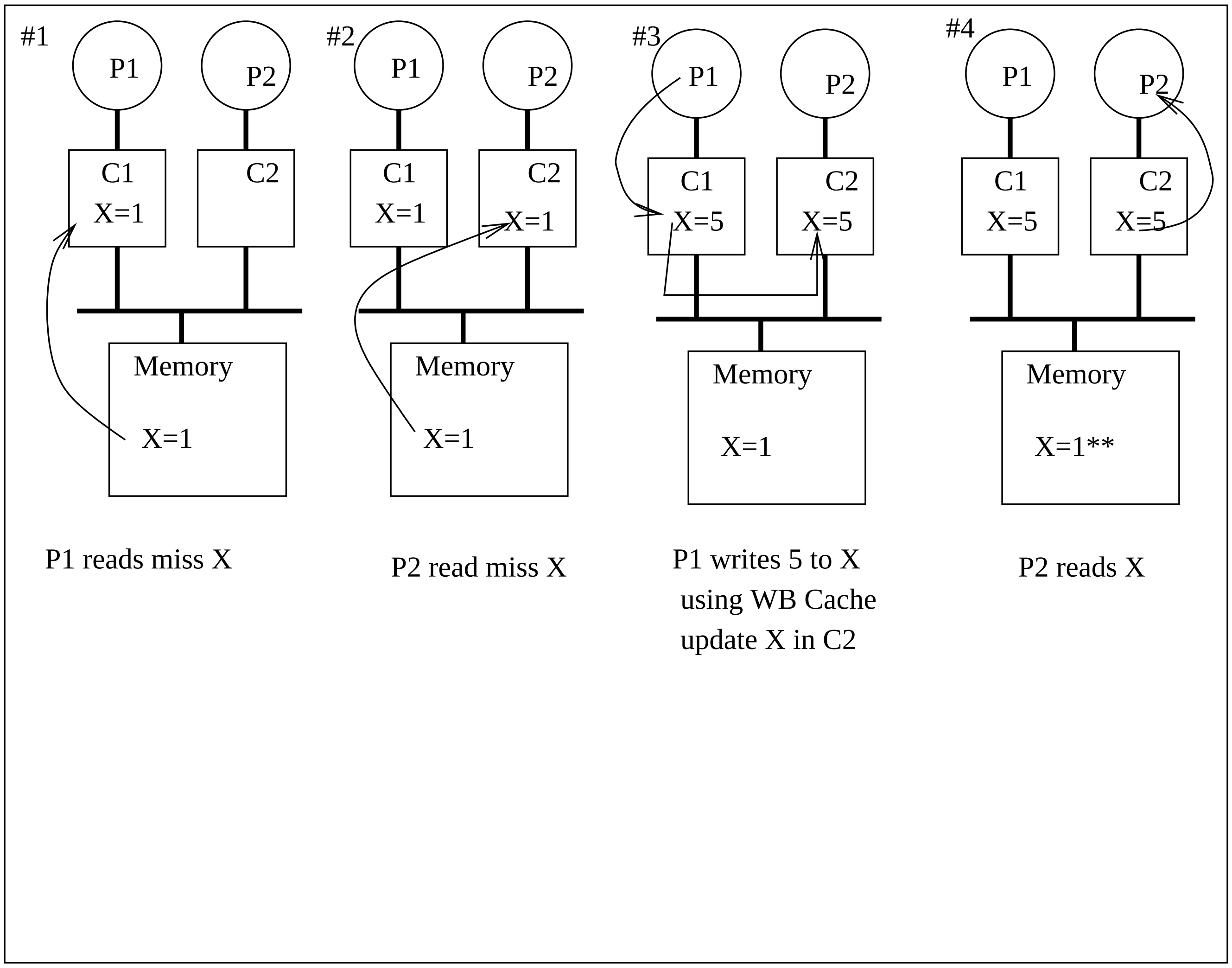
Example for write invalidate protocol

Write Invalidate:



- 2-**Write Update or write Brodcast:** To keep most update copy in cache.
  It requires higher bandwidth as every write will be transfered to all caches.
  It must isolate shared data from private data and should not brodcast writes to private data

Write Update



Performance of Write Invalidate Compared to Write Update

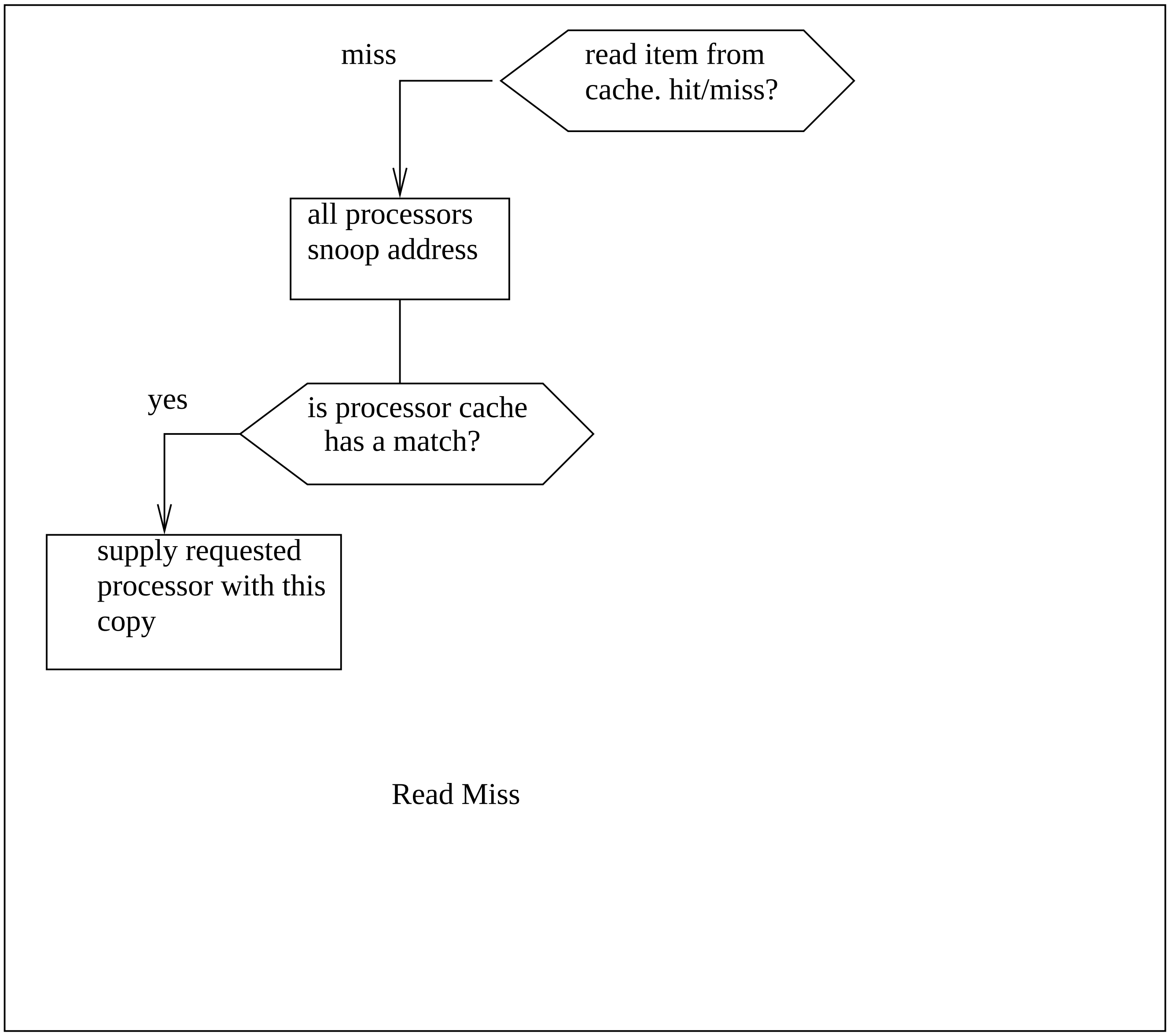
The differences are:-

- 1-Multiple writes to same data requires multiple write updates (bus bandwidth), on write update protocol, but only one invalidate for invalidate protocol.
- Multiword blocks in cache requires multiple transfer of all words on update protocol.
- delay between write followed by a read is usually less in write update. (no cache read miss "invalidate").
  because bus bandwidth is performance bottelneck, write invalidate has become the protocol of choice.

Implementation Techniques for Invalidate Protocol

- Using shared bus for Invalidation.
- All processors continuously snoop on bus for addresses that matches cache addresses.
- If there is a match, the block is invalidated
- Processor must obtain bus access (arbitrate) to write and must have cache misses
  with write back cache, if processor has read miss, and processors snoop bus and find that it has a dirty copy of it, then it must supply it to other processor that has requested it.
- For writes: If shared, then invalidate all other copies.
  with write back: Each block should be marked dirty with a write/hit
  For writes to not shared data, we need not to send invalidate signal.

Read Miss

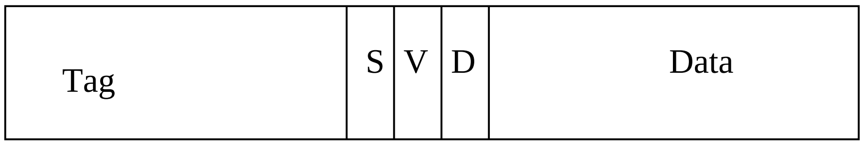

miss
read item from cache. hit/miss?
all processors snoop address
yes
is processor cache has a match?
supply requested processor with this copy
Read Miss

Status of Cache Block

It uses 3 bits

- **Dirty:** write hit makes block dirty
- **Invalid:** write invalidate signal from a write to same block in different cache
- bf SharedPrivate: when write invalidate all other copies, the local copy becomes not shared or private.
  If private block is requested by other procesor, it will be made shared.

Status of Ccahe Block



s: shared
D: Dirty
V: valid

Coherency in Multi-Level Cache

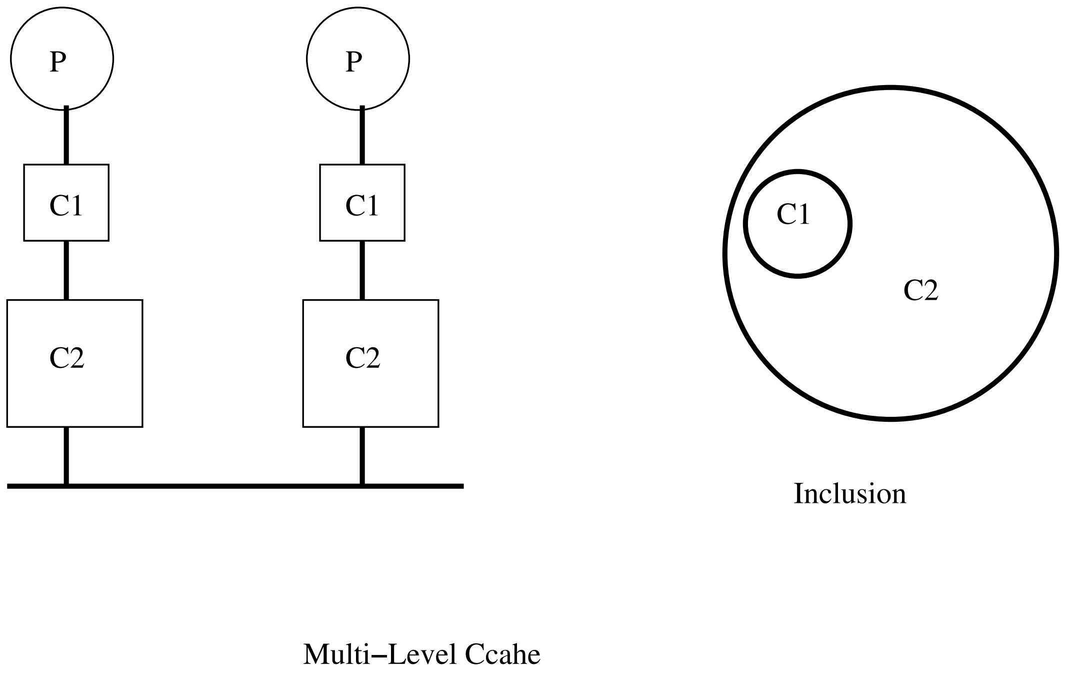
The shared bus is connected to one level only.

Must keep coherence between data in both levels, but only one level is connected to the bus?? (for invalidate)

Solution: Inclusion

Level closer to processor (C1) are a subset of those further away (C2).
If C2 cache has to invalidate or change status of a block, primary cache C1 must update block status.

Coherency in Multi-Level Cache



Cache Coherency Protocols

Write Through Cache Coherency Protocol

Two States: Valid and Invalid
Requests are from:
1- local processor i for a read or write
2-remote processor j for read or write through the bus
If state is Invalid and there is another processor reading or writing to it, will result in no change in state
If processor needs to read or write to a block and the state is invalid, processor will change block state to Valid
On state Valid, processor can read and write to block with no change in the state, but if other processor writes to the block, the state change to Invalid.

Write Through Coherency Protocol

$$\begin{tikzcd}
\text{Invalid} \arrow[loop left] \arrow[rr, bend left, "{R(i), W(i)}"] & & \text{Valid} \arrow[loop right, "{R(i), W(i), R(j)}"] \arrow[ll, bend left, "{W(j)}"]
\end{tikzcd}$$

Write Through Protocol

Write Invalidate Coherence Protocol

Must deal with:-
1-Processor ReadWrite to its cache
2-Bus (other processor's ReadWrite)
Coherency Protocol has three states:

- **Invalid:** This copy has been changed by another processor write to it. Only bus write causes block to be invalid
- **Shared (Read Only):** Processor reads a variable that was not in its cache. A read miss generates this state.
- **Exclusive (privatenot shared):** Processor writes to a variable, causes the block to be labeled Exclusive, as other copies in other processors caches are invalidated.

Write Back Coherency Protocol

Processor actions are:

1-Read hit

2-Read miss

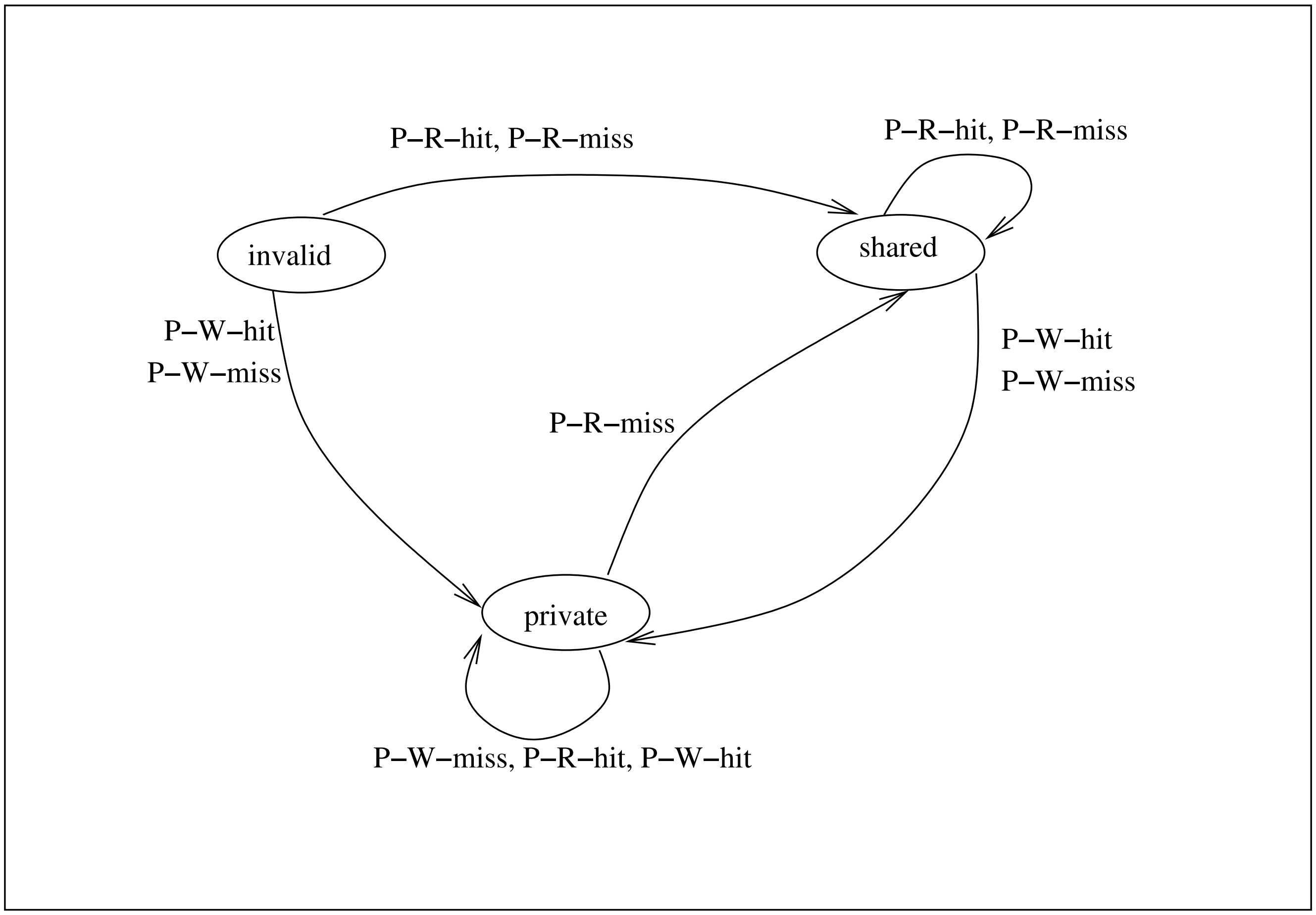
3-Write hit

4-Write miss

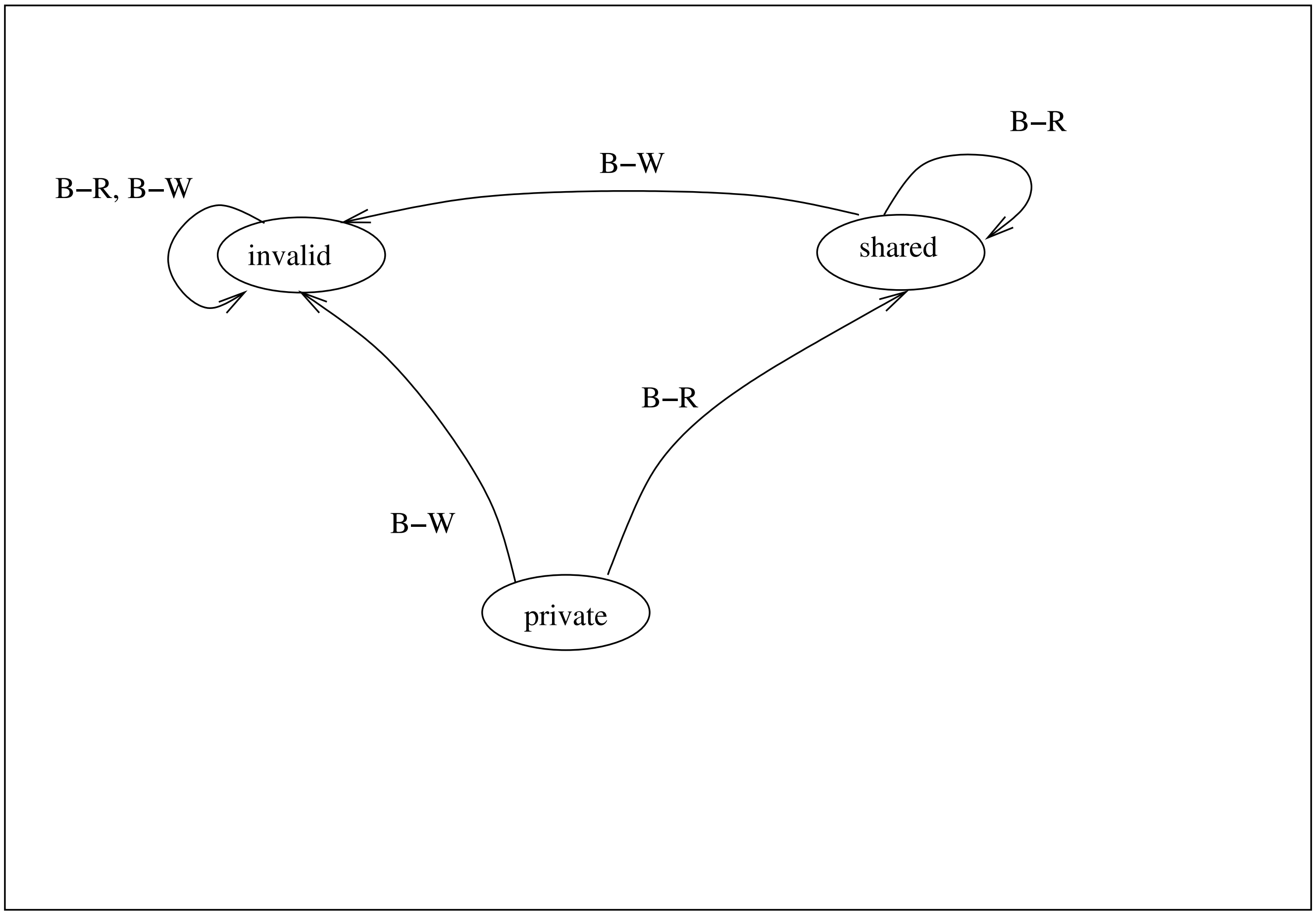
Using three state protocol

- 1-Processor writes (hitmiss) makes block private
- 2-Processor Read miss makes block shared
- 3-Processor Read hit no change if it is shared or private but if it is invalid it makes it shared
- 4-bus write makes block invalid
- 5-bus read makes block shared except if block is invalid it will stay invalid

Coherence Protocol for Processor Actions

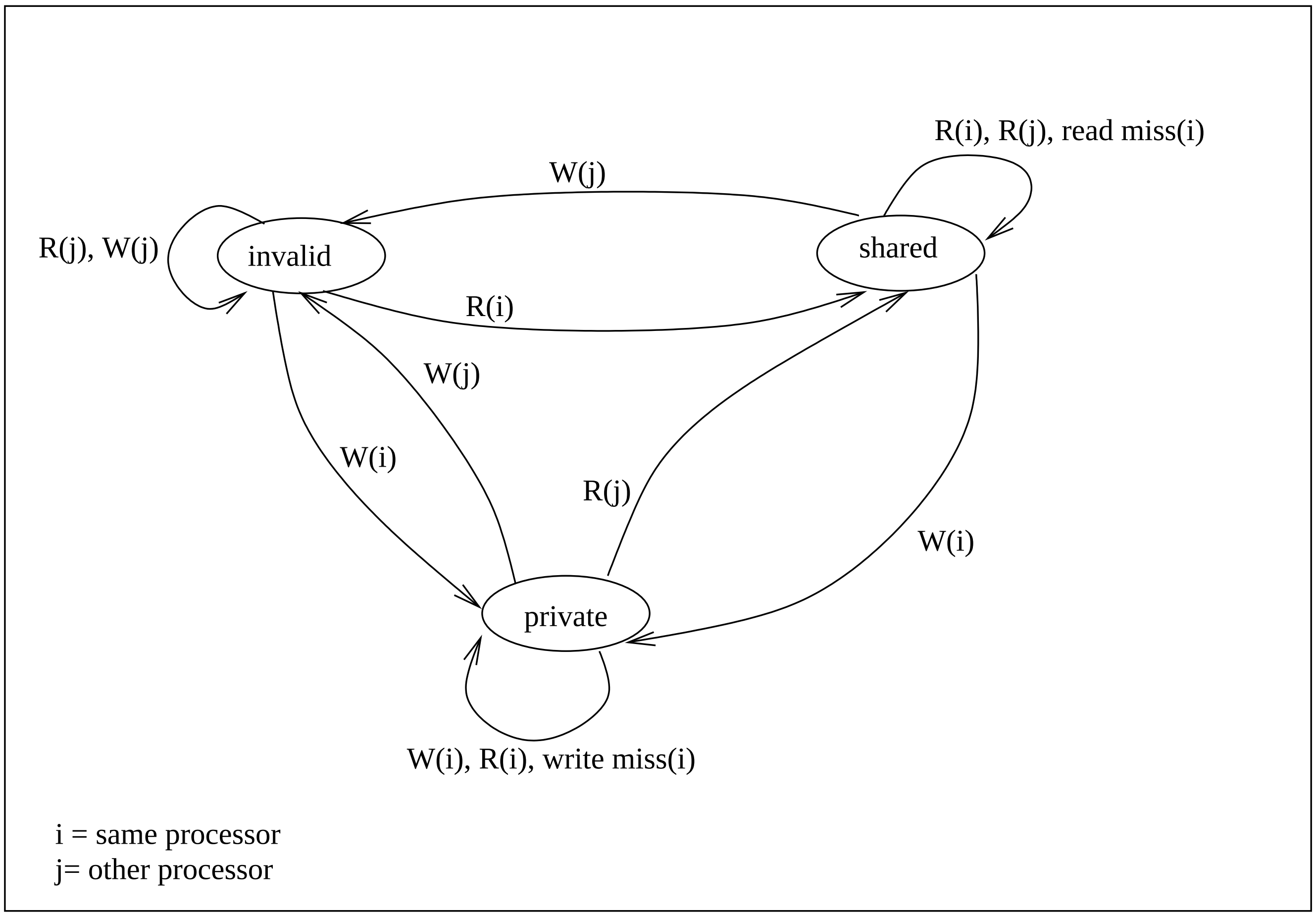


Coherence Protocol for Bus Actions



Complete Write Back Coherence Protocol

It includes Processor and Bus actions.



Multiprocessor Model

Model multiprocessor system by the average of multiplying the probability of each action times its cost in cycles.

Cost of processor read hit is= 1 cycle

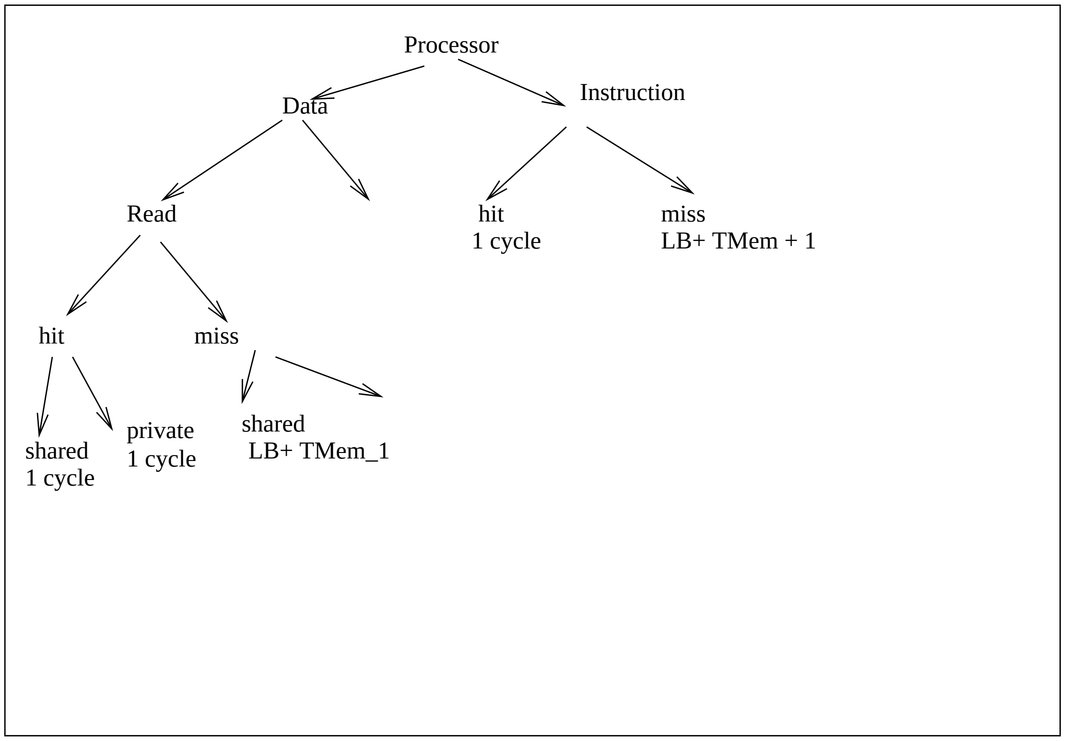
cost of processor write hit to private block = 1 cycle

cost of processor write hit to shared block = 1 cycle + bus invalidation

cost of read miss = bus latency + memory latency (DRAM or other processor cache) + 1 cycle

cost of write miss = bus latency + invalidate + 1 cycle

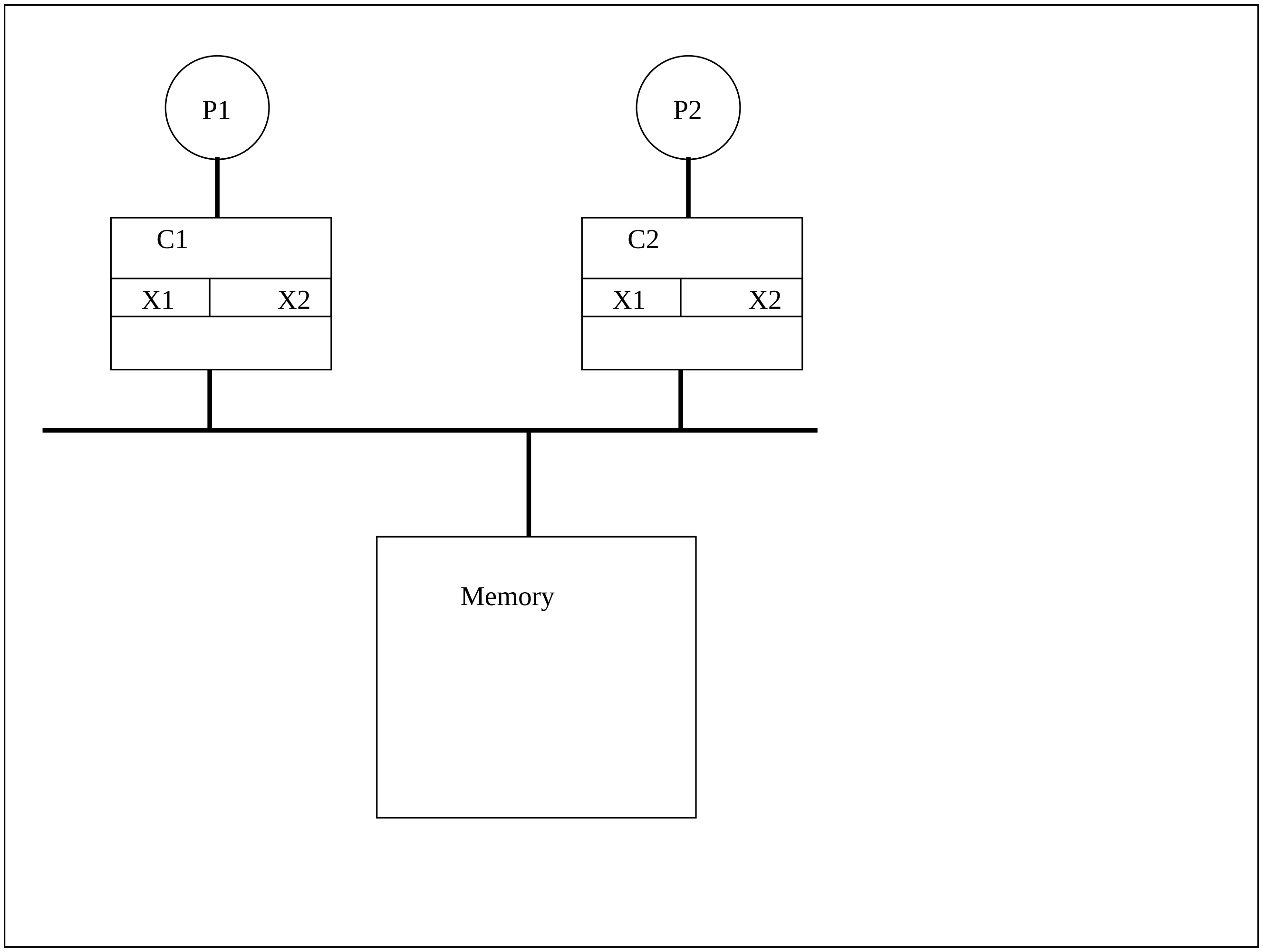
Example: Find performance of 8 processors assuming balanced load is executed on each one and read=20%, writ=10%, sharing=5%, and invalid = 3%. cost of bus latency = 30 cycles, memory latency=60 cycles, Inst miss = 5%, read miss= 7%, write miss= 8 %.


Processor
Data
Instruction
Read
hit
1 cycle
miss
LB+ TMem + 1
hit
miss
shared
1 cycle
private
1 cycle
shared
LB+ TMem_1

False Sharing

If block size is greater than word size, two elements could be mapped to same block, and both of them will be invalidated even if writing to one element only.

If P1 writes to X1, X2 will be also invalidated in P2 cache. P2 has to perform read miss to get X2 from P1 cache, although X2 value has not changed.



Example: Assume X1, X2 are read by P1 and P2 and share the same block. Find true and false sharing in the following sequence:
1-P1 writes to X1, causes P2 to invalidate X1, X2 in cache. X1 was read by P2 at time 0, so this is true sharing for X1
2-P2 reads X2, invalid because P1 writing to X1 in step 1, P2 has to miss ; this is false sharing
3-P1 writes to X1 , causes P2 to invalidate X1, X2. X1 was not read by P2, false sharing (P2 read X2 only).
4-P2 writes to X2, P1 invalidate X1, X2. P1 did not read X2 , False sharing.
5-P1 reads X2, has to miss and get it from P2 cache, P2 was last to write to X2, It is true sharing.

False Sharing

Time	P1	P2	comments
1	Write to X1		makes X2 invalid
2		Read X2	makes X1 shared
3	Write to X1		invalidate X2
4		Write to X2	makes X private, invalid at P1
5	Read X2		

Performance of coherence Protocol

1-Increasing number of processors affect miss rate (coherence miss rate increases as N increases).

2-Increasing cache size improves both coherence miss and capacity miss.

3-Increasing block size improves capacity miss but might incease coherence miss due to false sharing.

Synchronization

-Need synchronization to know when it is safe to use shared data

-Need special type of instructions with hardware capability.

-For large scale machines, sychronization could be the performance bottelneck.

Special Type of Instructions with Atomic Operations

Atomic operation: Uninterruptable operation to retreive and change the value in memory.

Need to read and modify memory location atomically.

Typical atomic operations:

- **atomic exchange:** interchange value in a register for a value in memory
- **test and set:** tests a value and sets it if value passes the test.
- **fetch and increment:** returns the value of a memory and atomically increments it.

User Level Synchronization Operation
User uses atomic operations for synchronization.
1-Using atomic exchange operation:-
Use a lock in memory for the variable. If the lock value=0, the lock is free and processor can access the variable. If the lock value=1, lock is unavailable.
Processor sets a value in a register=1, and uses the exchange instruction
if the register returns 0, the lock is available and now is set to 1 by the value of register exchange (this for other processors not to access it).
if register returns 1, the lock is used by another processor.

Spin Locks

processor continuously tries to acquire lock, spinning around a loop

```
            li      R2, #1    ; R2=1
lockit:     exch    R2, 0(R1) ;atomic exchange
            bnez    R2, lockit;
```

Problem with spin lock; the processor is tied up waiting in a loop.
Solution: can cache the locks and use coherence to maintain lock value.
Advantages:
-Processor spinning in its local cache (no memory or bus)
Locality in lock accesses suggest that processor that used the lock last will use it again soon.
Problem: On local cache, the exch instruction involves a write to the cache copy, needs to invalidate all other copies. This causes a lot of bus traffic.
Solution: repeat just read variable, and only change it if the value of lock=0 (when available).

example of synchronization using spin lock

```
lockit: lw   R2, 0(R1)  ; read lock
        bnez R2, lockit ; not available
        li   R2, #1     ; Set R2=1
        exch R2, 0(R1)  ; now can swap
        bnez R2, lockit ; branch if lock
                          was not 0
```

Example of Cache Coherence and Synchronization

step	P0	P1	P2	Lock state	bus
1	has lock	waiting lock=0	waiting lock=0	shared	none
2	set lock=0	invalid received	invalid received	exclusive	write invalida from P0
3		cache miss	cache miss	shared	cache mi service

	wait	lock=0	shared	miss sat for P2
5		exch	shared	P2 write invalida

6		lock=1
		return 0
			exclusive	write ba
				P2

7	reads lock		shared	
8	wait for lock=0			

Coherence and Synchronization Example

Processor P0 has lock in its cache

P1,P2 Simultaneously require the lock

P0 release it, write invalidate

P1,P2 read miss trying to get lock.

P2 is faster)first), reads lock=0

P2 acquire the lock and sets it to 1 and use write invalidate.

P1 read miss will return lock=1, and P1 must wait until P2 releases the lock and lock value=0

Other Instructions for Synchronization

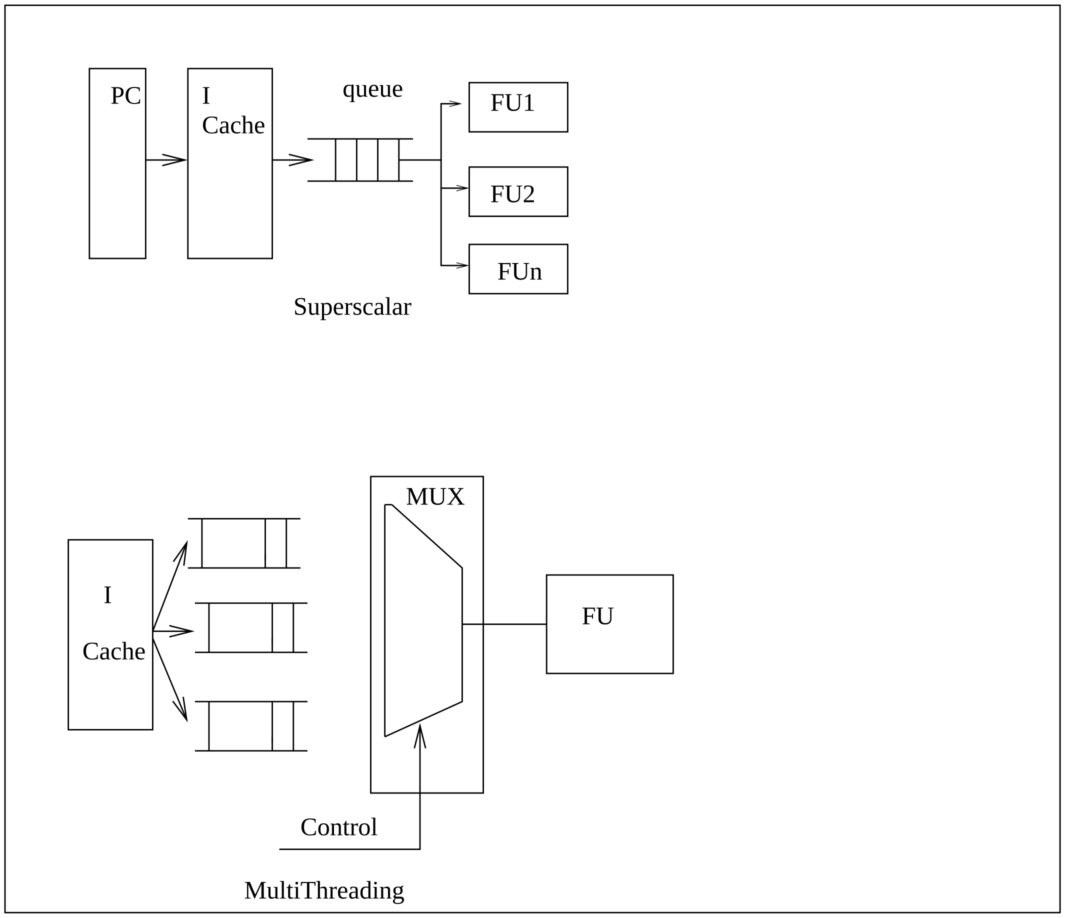
1-Load Linked or load locked + store conditional
-load returns intial value
-store conditional returns 1 if it succeeds
two operations that are separated (readwrite)
load linked does not use bus.

```
lockit: li    R2, 0(R1); load linked
        bnez  R2, lockit; not aval
        li    R2, #1     ; R2=1
        SC    R2, 0(R1)  ; store
        beqz  R2,lockit  ; if store fails
```

Need to implement synchronization at hardware level

Multithreading In superscalar, it issues multiple instructions dynamically and extract ILP from loop unrolling. It uses multiple function units to execute instructions in parallel. All instructions are from the same stream (thread).

Multiple threads each has its own PC and different registers, different page table.
All threads share same Function Units and other resources by switching threads.
Advantages:
When one thread stalls, others can execute and hide stall latency.



Types of Multithreading:

- 1-Coarse grain: Switches on L2 stalls, not effective in hiding shorter stalls because of pipeline startup latency
- 2-Fine grain: switches on every cycle and could slow the execution of each individual threads as it will be interrupted by the other threads.

Simultaneous Multithreading It combine multiple issue instructions and multithreading together.
TLP + ILP at the same time.
multiple instructions from independent threads could be issued simultaneously.

Superscalar	MT	SMT
1-T1 X X	T1 X X	T1,T2 X X Y Y
2-T1 X	T1 X	T1,T2 X Y
3-T1 X X X	T1 X X X	T1,T2 X X X Y
4-T1 X X		T1,T2 Y Y Y
5-	T2 Y Y Y	
6-T1 X	T2 Y Y Y	
7-T1 X X X	T2 Y	