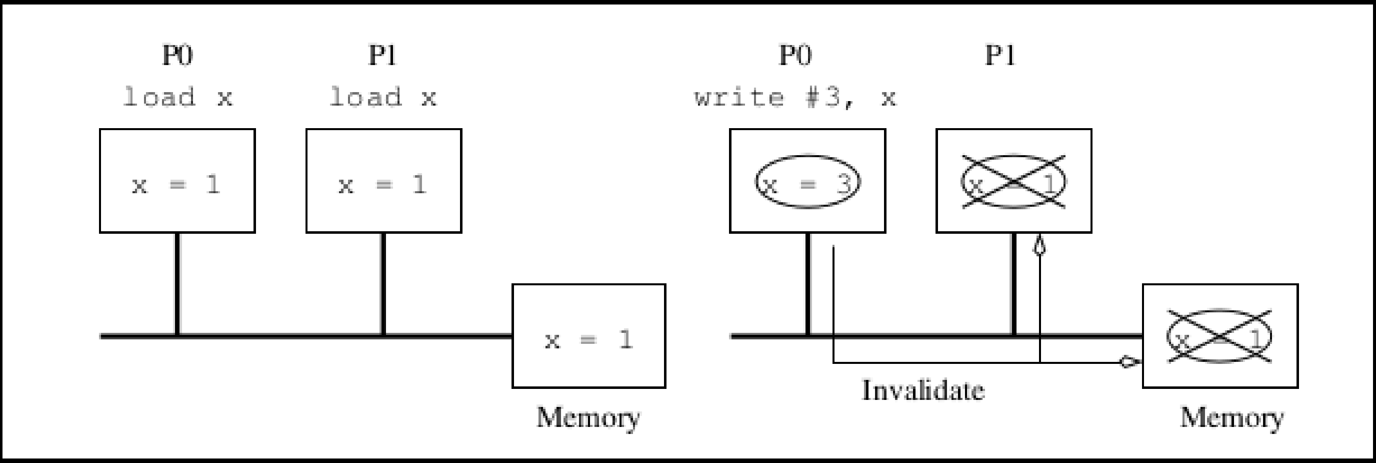
Sai

Cache Coherence in Multiprocessor Systems

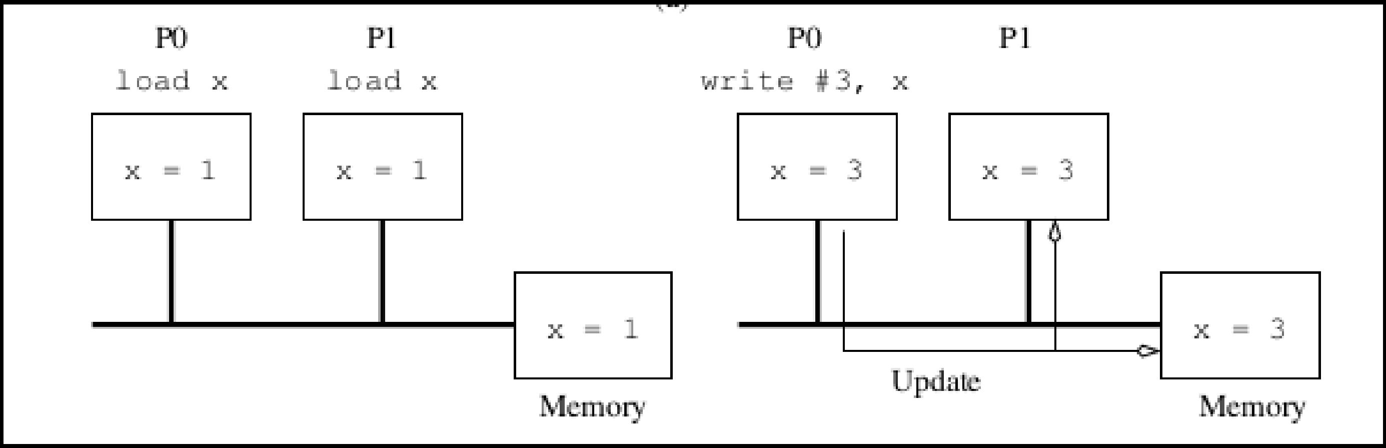
- Issue in Shared Address Space model
- Additional hardware needed to keep multiple copies of data consistent with each other
- Local Caches + Memory
- Cache Coherence to provide a guarantee of Serializability → ∃ some serial order of instruction execution corresponding to the Parallel Schedule
- If multiple copies of a variable are loaded, and one of them modifies its copy, either invalidate the other copies or update the other copies
- Invalidate & Update protocols

→ INVALIDATE



- Invalidates the data item on first update at a remote processor and subsequent updates are not necessary.

→ UPDATE



- Whenever a data item is written, all of its copies are updated

- If a processor reads a data item once and never uses it, subsequent updates to this item at other processors cause excess overhead

→ Both suffer from False Sharing ←

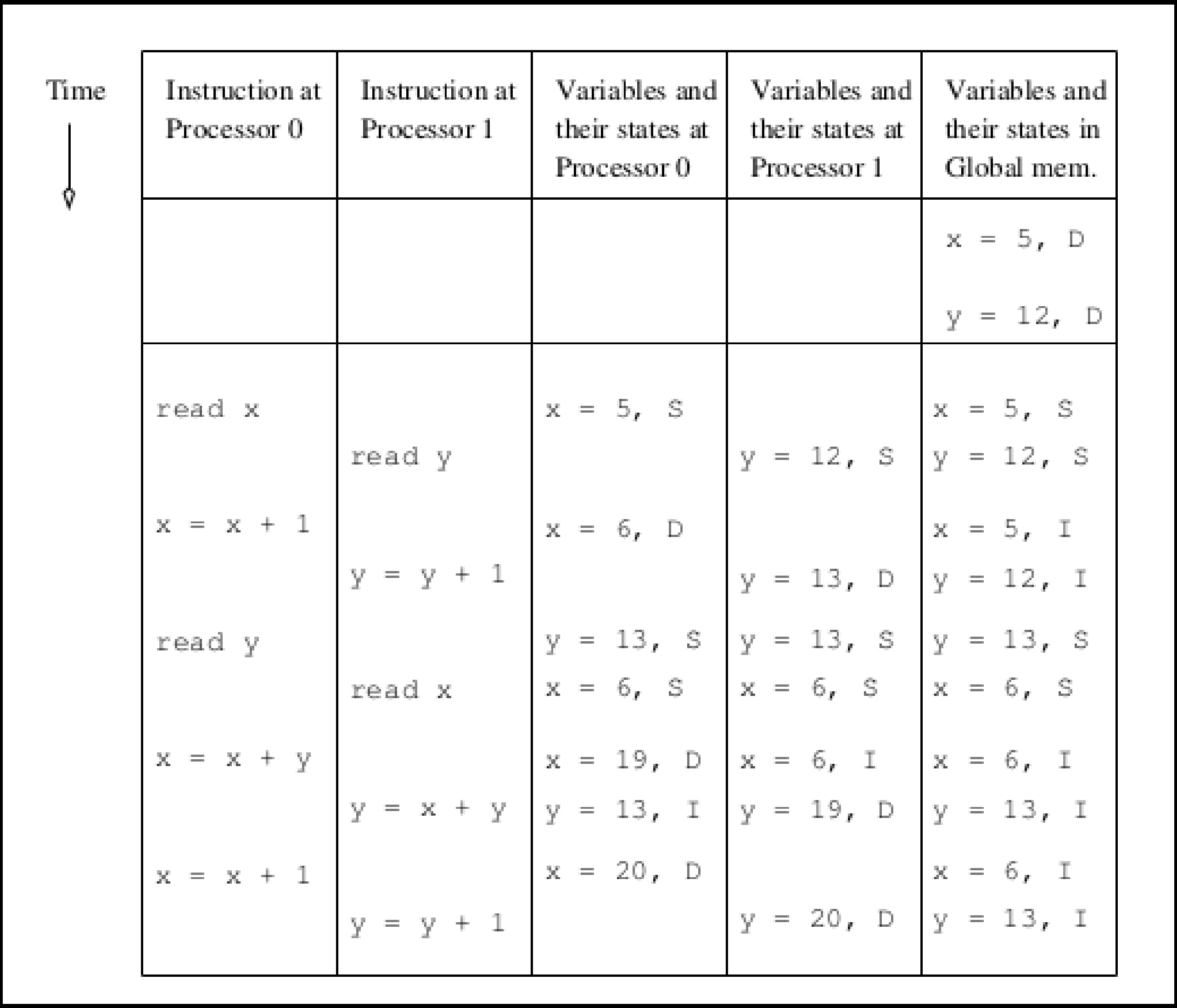
Three-State Protocol (Invalidate)

State of Copy of a Variable (Cache line)

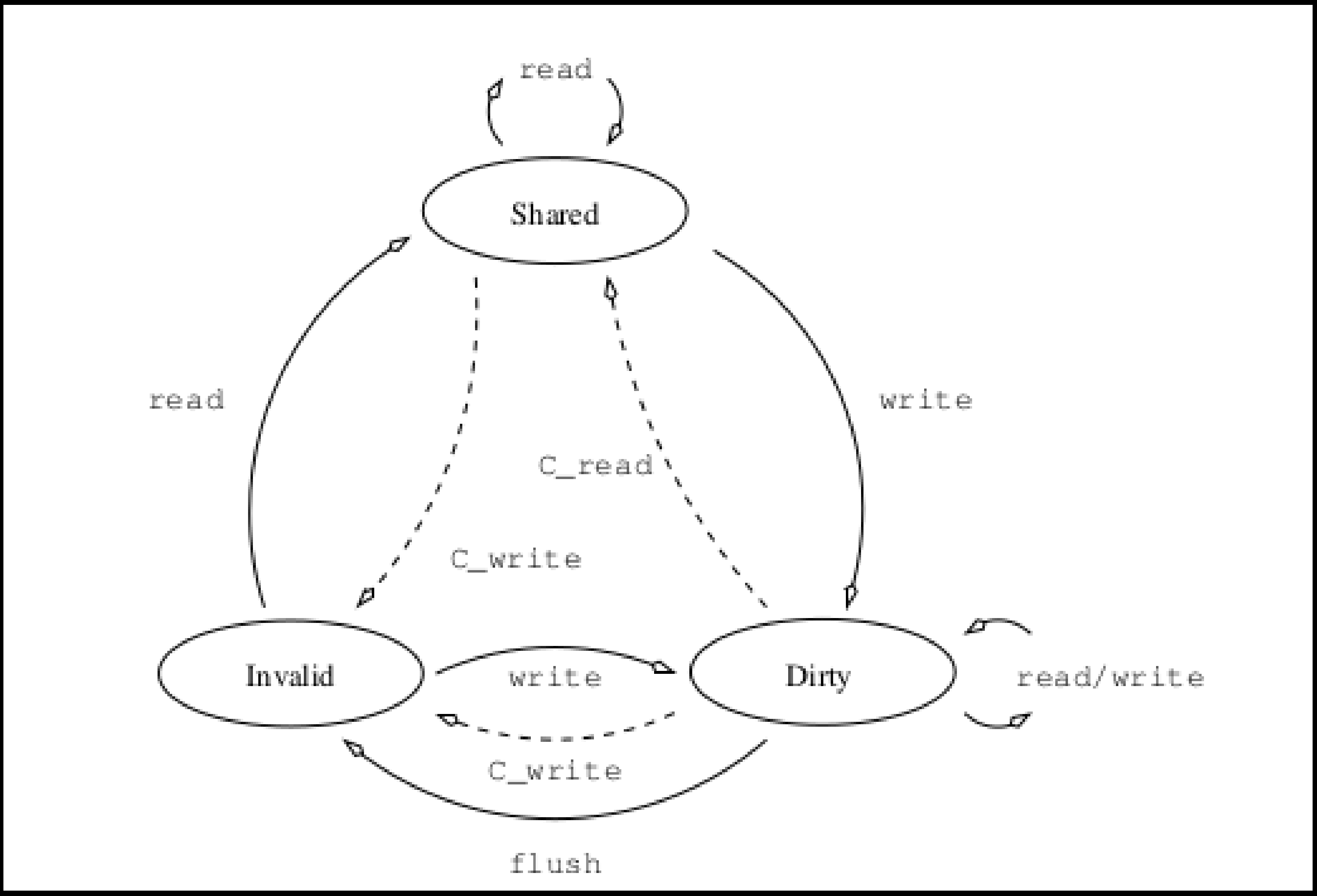
→ Shared : Multiple valid Copies of the variable exist

→ Dirty : This copy of the variable holds the correct value and must be used to service future load requests

→ Invalid : This copy of the variable is not valid



Time	Instruction at Processor 0	Instruction at Processor 1	Variables and their states at Processor 0	Variables and their states at Processor 1	Variables and their states in Global mem.
					x = 5, D y = 12, D
	read x		x = 5, S		x = 5, S
		read y		y = 12, S	y = 12, S
	x = x + 1		x = 6, D		x = 5, I
		y = y + 1		y = 13, D	y = 12, I
	read y		y = 13, S	y = 13, S	y = 13, S
		read x	x = 6, S	x = 6, S	x = 6, S
	x = x + y		x = 19, D	x = 6, I	x = 6, I
		y = x + y	y = 13, I	y = 19, D	y = 13, I
	x = x + 1		x = 20, D		x = 6, I
↓		y = y + 1		y = 20, D	y = 13, I



Implementation :

I. Snoopy Cache Systems

→ Based on broadcast interconnection like bus or ring

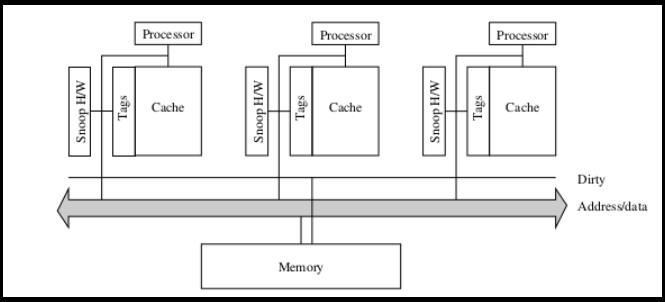
→ All processors snoop the bus for transactions

→ Each Processor's cache has a set of tag bits storing the state of the cache line

→ Tags are updated according to the Coherence protocol

→ e.g.: If there is a read x from P_1, a copy of which, is present in the cache line of P_0, with tag = D (state), then P_0 asserts control of the bus and puts the data out.

If P_0 detects a write on the snoop hardware to a cache line it has a copy of, it invalidates the cache line.



→ Easy to implement as it is bus based

Performance advantage due to -

- If different processors operate on different data, and cached as dirty, all subsequent operations can be performed locally without generating any traffic.

- If a data item is read by multiple processors, and if it moves to the shared state, then all subsequent reads are local.

Bottleneck due to -

- If multiple processors read and update the same data item, they generate coherence functions across the processors.

Shared bus has finite bandwidth and hence, only a constant number of such coherence functions can be executed in a time unit.

To improve performance,

Why should all processors snoop all transactions?
Broadcasting all memory operations to all processors is not a scalable solution.

∴ Propogate coherence operations only to those processors that need to participate

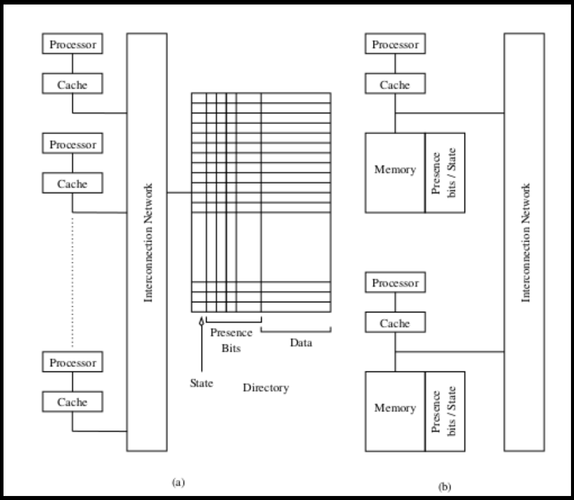
⇒ Track copies of data items & states on processors

II. Directory based systems

→ Global Memory is augmented with a directory that maintains a bitmap representing memory blocks and the processors at which they are cached (presence bits)

→ State of the block (Shared, Invalid or Dirty) is also maintained in the directory

→ Only those processors that cache a particular memory block participate in state transitions due to coherence operations.



e.g: Let X be Dirty ⇒ Memory block state is D.

If P_0 & P_1 read X ⇒ Memory block state is S and Presence bits of P_0 & P_1 is 1

If P_0 writes to X ⇒ Memory block state is D
Presence bit of P_0 is 1
& " " " P_1 is 0

All subsequent reads of X at P_0 are local.

If P_1 reads X, directory notices "D" state, uses presence bits (P_0 is 1), directs the request to P_0.
P_0 updates block in memory and sends it to P_1
State of memory block changes to "S"
and presence bits of P_0 & P_1 are set to 1.

⇒ Better than Snoopy, as Processors that Cache a memory block participate in coherence operations and not all

⇒ But causes contention in the directory as all coherence operations update state in the directory that can service only a bounded number of read / write operations in a time unit

⇒ Memory requirement grows as $O(mp)$, where m is the number of memory blocks

Larger memory blocks → lesser memory in directory
but leads to False Sharing

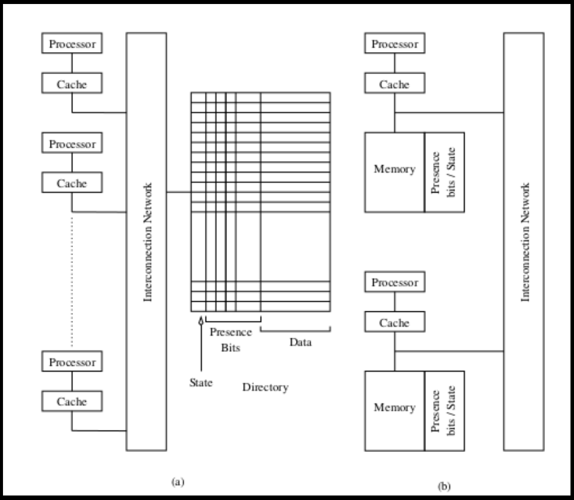
∴ To break the contention, break up the task of maintaining coherence across multiple processors.

III. Distributed Directory Systems

⇒ Assume a physical / logical partitioning of memory blocks across processors

⇒ Each processor to own a set of blocks and maintain coherence of its blocks

⇒ Every memory block owner can be computed easily and hence knowledge of owner is known to all processors implicitly



⇒ When a Processor reads a block for the first time, it requests the owner for the block

⇒ Owner directs the request based on State and Presence bits locally available

⇒ When a processor writes to a memory block, it propagates an Invalidate to the owner, which in turn forwards the invalidate to all Processors that have a Cached Copy of the block

∴ Contention alleviated

⇒ Can permit $O(p)$ Simultaneous Coherence operations

∴ More Scalable than Snoopy or Centralized Directory Systems.