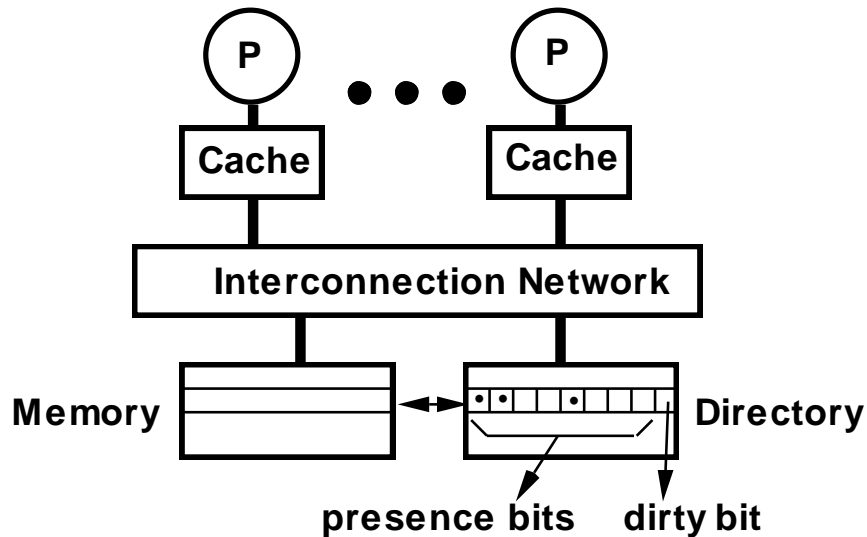


Basic DSM Cache Coherence

[§10.3] Let us start off by considering a full bit-vector approach.



For each block of memory, assuming there are k processors, it maintains at the home node of the block ...

- k presence bits $p[.]$
- 1 dirty bit D

Cache state is represented the same way as in bus-based designs (MSI, MESI, etc.).

- On a read miss by processor i , the home node reacts this way:
 - **If** ($D == 0$) { supply data; $p[i] = 1$; }
 - **else** { send intervention to owner; update home; $D = 0$; $p[i] = 1$; supply data to i ;
- On a write miss by processor i ; [tell how the home reacts](#):
 - **if** ($D == 0$) { _____ ; $D = 1$; $p[i]=1$; supply data to node i ; }
 - **else** { _____ ; $p[owner] = 0$; $p[i] = 1$; supply data to i ;

On the replacement of a dirty block by node i , the data is written back to memory and the directory is updated to turn off the dirty bit and $p[i]$.

On the replacement of a shared block, the directory may or may not be updated.

How does a directory help? It keeps track of which nodes have copies of a block, eliminating the need for _____.

Would directories be valuable if most data were shared by most of the nodes in the system?

Fortunately, the number of valid copies of data on most writes is small.

The [attached animation](#) uses the MESI protocol, with [3 block states](#) in main memory:

- EM (exclusive or modified)
- S (shared)
- U (unowned)
- ✓ Network transactions for coherence
 - Read: read request
 - ReadX: read exclusive (or write) request
 - Upgr: upgrade request
 - ReplyD: home replies with data to requestor
 - Reply: home replies to requestor with IDs of sharers
 - Inv: home asks sharer to invalidate
 - WB+Inv: home asks owner to flush and invalidate
 - WB+Int: home asks owner to flush and change to S
 - Flush: owner flushes data to home + requestor
 - InvAck: sharer/owner acks an invalidation msg

- Flush+InvAck: Flush, piggybacking an InvAck message

✓ Notation

- Transaction (Source → Destination)
- H = Home node

The following example is used in the animation:

Proc action	P1 state	P2 state	P3 state	Dir state @home	Network messages	# of hops
R1	E	–	–	EM, 100	Read (P1 → H), ReplyD (H → P1)	2
W1	M	–	–	EM, 100	—	0
R3	S	–	S	S, 101	Read (P3 → H), WB+Int (H → P1), Flush (P1 → H, P3)	3
W3	I	–	M	EM, 001	Upgr (P3 → H), Reply (H → P3) // Inv (H → P1), InvAck(P1 → P3)	3
R1	S	–	S	S, 101	Read (P1 → H), WB+Int (H → P3), Flush (P3 → H, P1)	3
R3	S	–	S	S, 101	—	0
R2	S	S	S	S, 111	Read (P2 → H), ReplyD (H → P2)	2

Scaling with number of processors

In order for directory schemes to be practical, they must scale gracefully.

- Scaling of memory and directory bandwidth
 - Centralized directory is bandwidth bottleneck, just like centralized memory.
 - How can we maintain directory information in a distributed way?
- Scaling of performance characteristics
 - traffic: # of network transactions each time protocol is invoked
 - latency: # of network transactions in critical path each time
- Scaling of directory storage requirements
 - Number of presence bits needed grows as the number of processors.

E.g., 64-byte block size and 1024 processors. How many bits in block, vs. # of bits in directory?

Directory organization affects all of these issues.

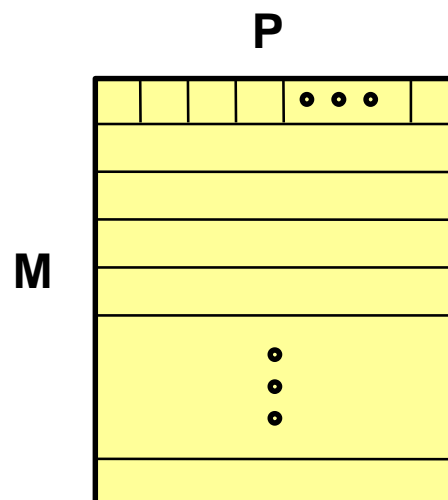
Organizing a memory-based directory scheme

All info about copies is colocated with the block itself at the home

This works just like a centralized scheme, except that it is distributed.

Scaling of performance characteristics

- Traffic on a write is proportional to number of sharers.
- Latency? Can issue invalidations in parallel.



Scaling of storage overhead? Assume representation is a full bit-vector.

[Optimizations](#) for full bit-vector schemes

- Increase (1) size (reduces storage overhead proportionally).
- Use multicore nodes (one bit per multicore node, not per processor)
- still scales as pm , but only a problem for very large machines
 - 256 procs, 4 per chip, 128B line: (2) % o'head

► Reducing “width”: addressing the p term

- Observation: most blocks are cached by only few nodes
- Instead of keeping a bit per node, make entry contain a few (3).
 - If $p = 1024$, 10-bit \Rightarrow can use 100 and still save space.
- Sharing patterns indicate a few pointers should suffice (five or so).
- We also need an overflow strategy for when there are more sharers than pointers.

► Reducing “height”: addressing the m term.

- Observation: number of memory blocks \gg number of cache lines.
- Thus, most blocks will not be cached at any particular time; therefore, most directory entries are useless at any given time
 - organize directory as a cache, rather than having one entry per memory block (key is (4), value is (5))

Organizing a cache-based directory scheme.

In a cache-based scheme, the home node only holds a pointer to the rest of the directory information.

The copies are linked together via a distributed list that weaves through caches.

Each cache tag has a pointer that points to the next cache with a copy.

- On a read, a processor adds itself to the head of the list (communication needed).
- On a write, it makes itself the head node on the list, then propagates a chain of invalidations down the list.
Each invalidation must be acknowledged.
- On a write-back, the node must delete itself from the list (and therefore communicate with the nodes before and after it).

Disadvantages: All operations require communicating with at least three nodes (node that is being operated on, previous node, and next node).

Write latency is proportional to number of sharers.

Synchronization is needed

Advantages: Directory overhead is small.

Work of performing invalidations can be distributed among sharers.

The IEEE Scalable Coherent Interface has formalized protocols for handling cache-based directory schemes.

The SSCI protocol

- SCI (Scalable Coherence Interface) protocol
 - IEEE standard, ratified in 1993
 - 7 state bits, 29 stable states + many pending states
- For illustration we will use Simple SCI (SSCI)
 - Retains similarity with full-bit vector protocol:
 - MESI states in the cache
 - U, S, EM states in the memory directory
 - Replaces the presence bits with a pointer
 - Similar features to SCI

- Overall protocol operation
- Doubly linked list
 - Many possible race conditions, which are mostly ignored in the illustration
- Additional coherence network transactions (in addition to those used in full bit-vector approach):
 - WB+Int+UpdPtr
 - UpdPtr: update next/prev/head pointers

Here is the example used in the animation.

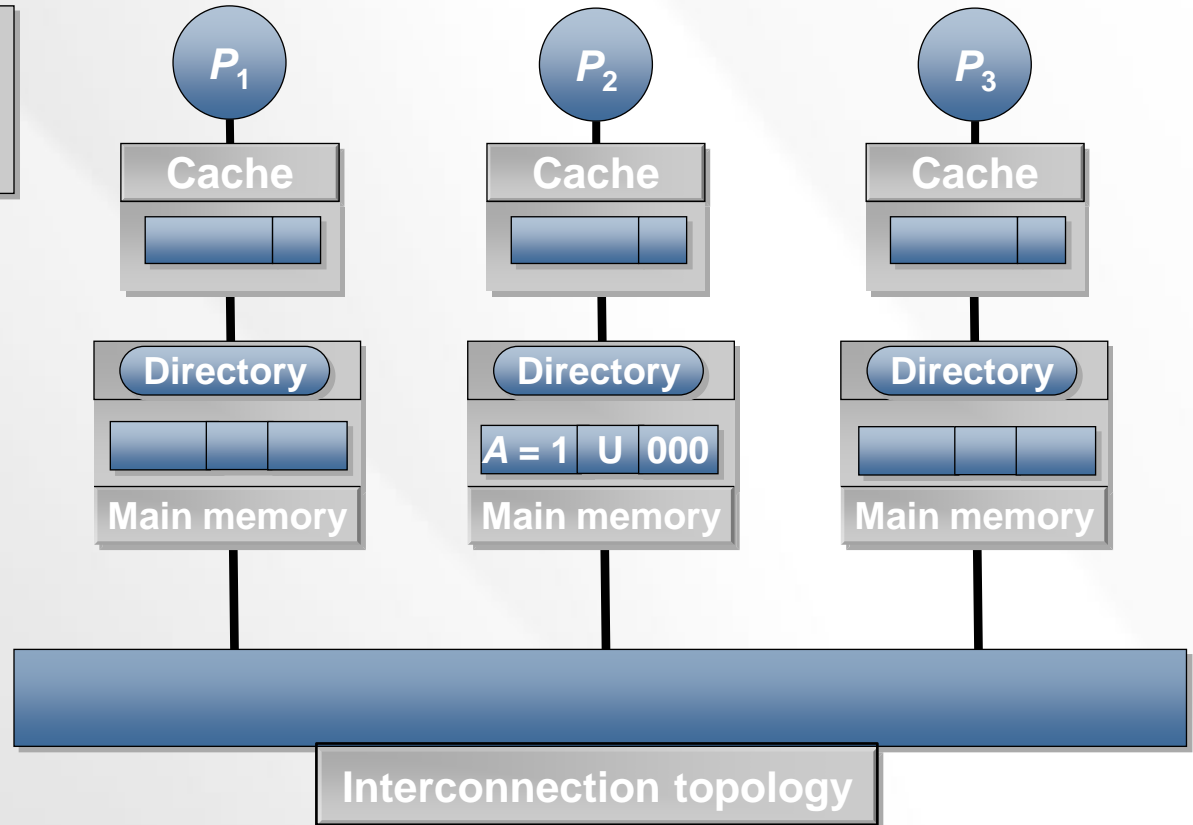
Proc action	P1 state	P2 state	P3 state	Dir state @home	Network message	# of hops
R1	E,0,0	—	—	EM, 1	Read (P1 → H), ReplyD (H → P1)	2
W1	M,0,0	—	—	EM, 1	—	0
R3	S,3,0	—	S,0,1	S, 3	Read (P3 → H), Reply (H → P3), WB+Int+UpdPtr (P3 → P1), Flush (P1 → H, P3)	4
W3	I,3,0	—	M,0,0	EM, 3	Upgr (P3 → H) // Inv (P3 → P1) InvAck(P1 → P3)	2
R1	S,0,3	—	S,1,0	S, 1	Read (P1 → H), Reply (H → P1), WB+Int+UpdPtr (P1 → P3), Flush (P3 → H, P1)	4
R3	S,0,3	—	S,1,0	S, 1	—	0
R2	S,2,3	S,0,1	S,1,0	S, 2	Read (P2 → H), ReplyD/ID (H → P2), UpdPtr (P2 → P1)	3

Full Bit-Vector Visualization – Start State

Start state. All caches empty and main memory has $A = 1$ in state U. Bit vector is 000.

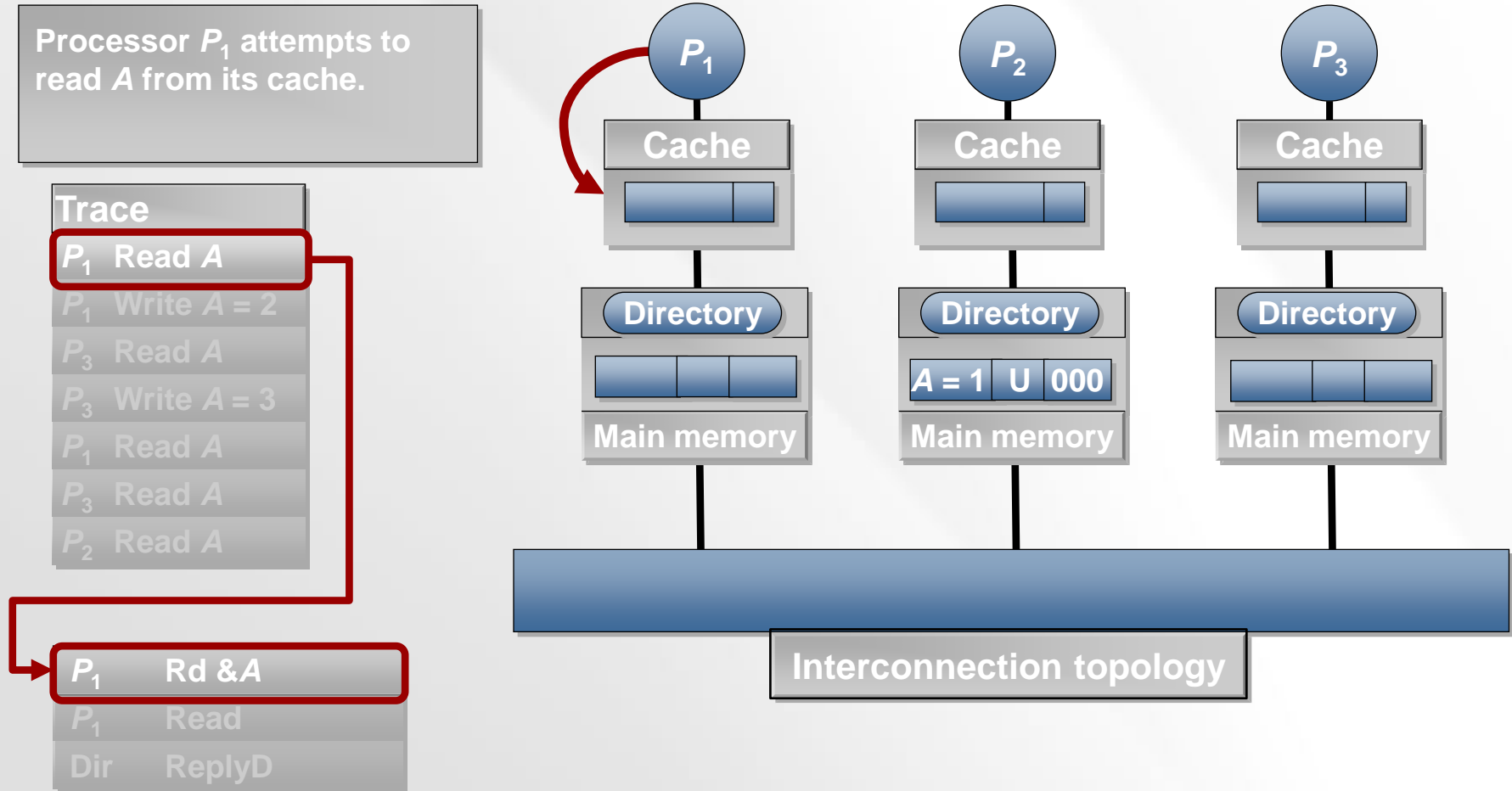
Trace

P_1 Read A
 P_1 Write $A = 2$
 P_3 Read A
 P_3 Write $A = 3$
 P_1 Read A
 P_3 Read A
 P_2 Read A

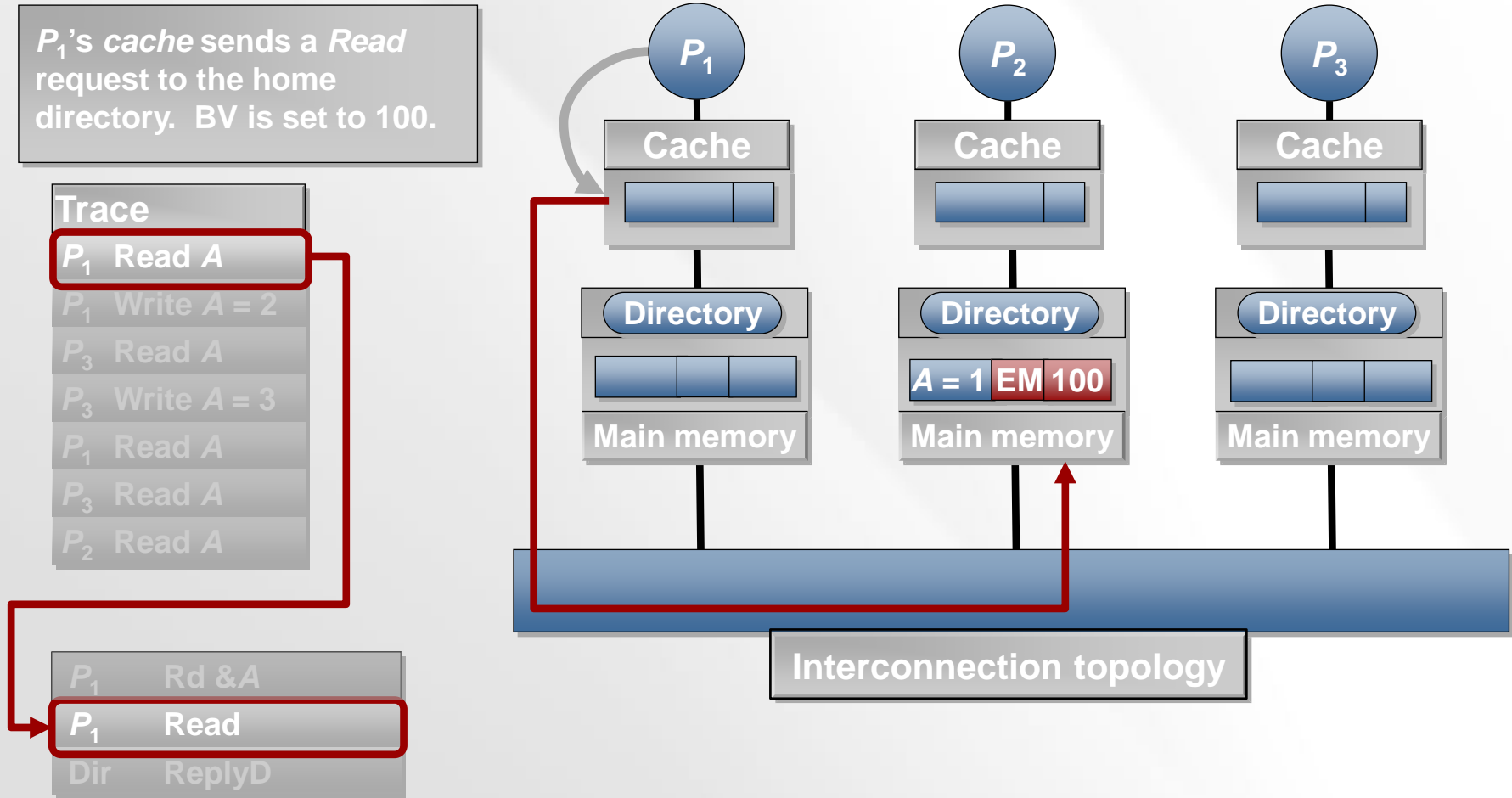


Edited by Samuel Christie and Amey Deshpande

Full Bit-Vector: Processor P_1 Reads A



Full Bit-Vector: Processor P_1 Reads A



Full Bit-Vector: Processor P_1 Reads A

Directory replies with value of A and the state of A in P_1 's cache is set to E.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

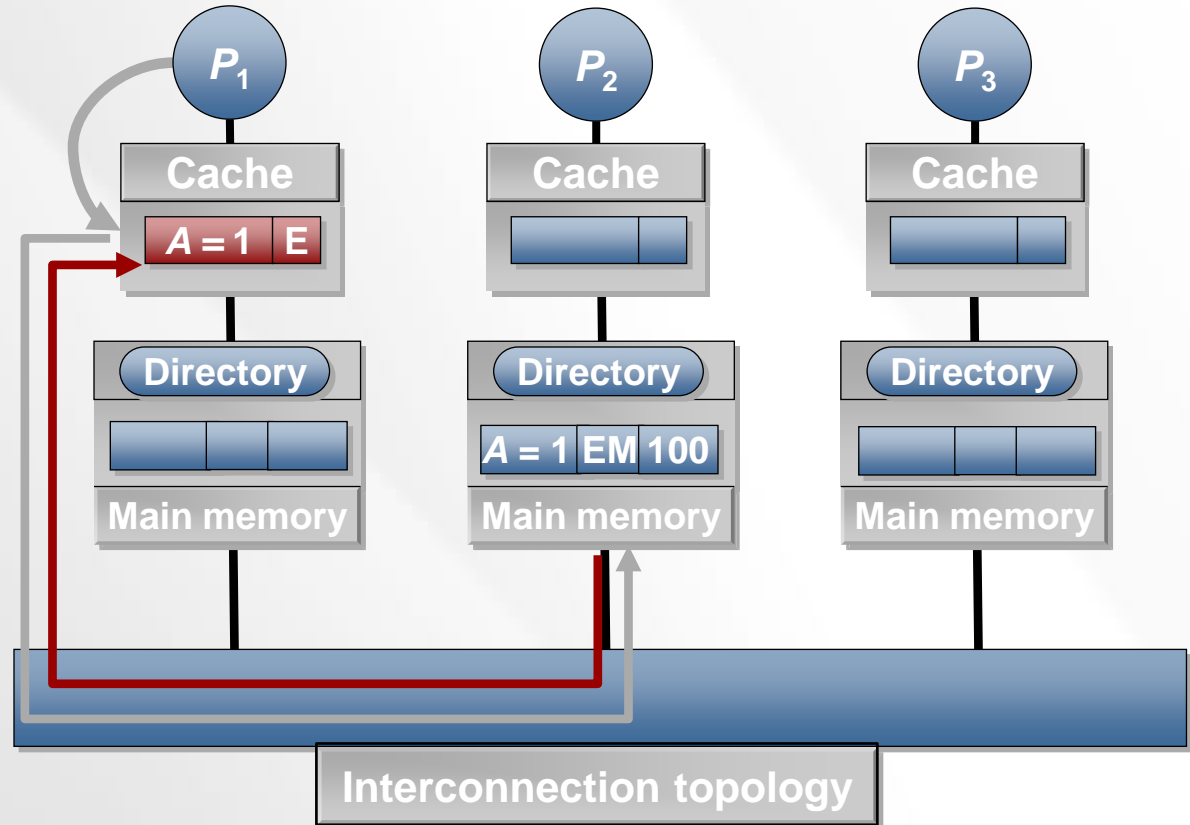
P_3 Read A

P_2 Read A

P_1 Rd &A

P_1 Read

Dir ReplyD



Full Bit-Vector: Processor P_1 Reads A

Processor P_1 's read completes.

Trace

P_1 Read A

P_1 Write A = 2

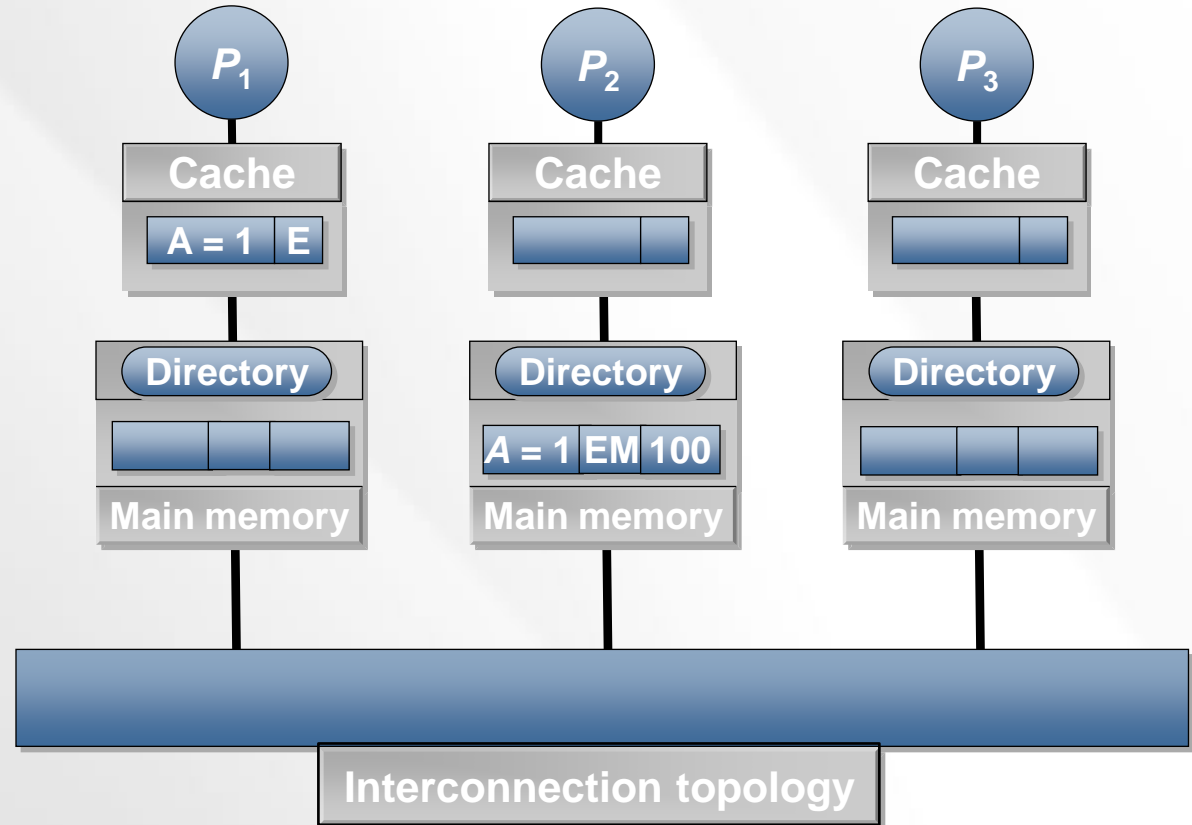
P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

P_2 Read A



Full Bit-Vector: Processor P_1 Writes A

Processor P_1 attempts to write $A=2$ in its cache. Value is modified and state is set to M.

Trace

P_1 Read A

P_1 Write $A = 2$

P_3 Read A

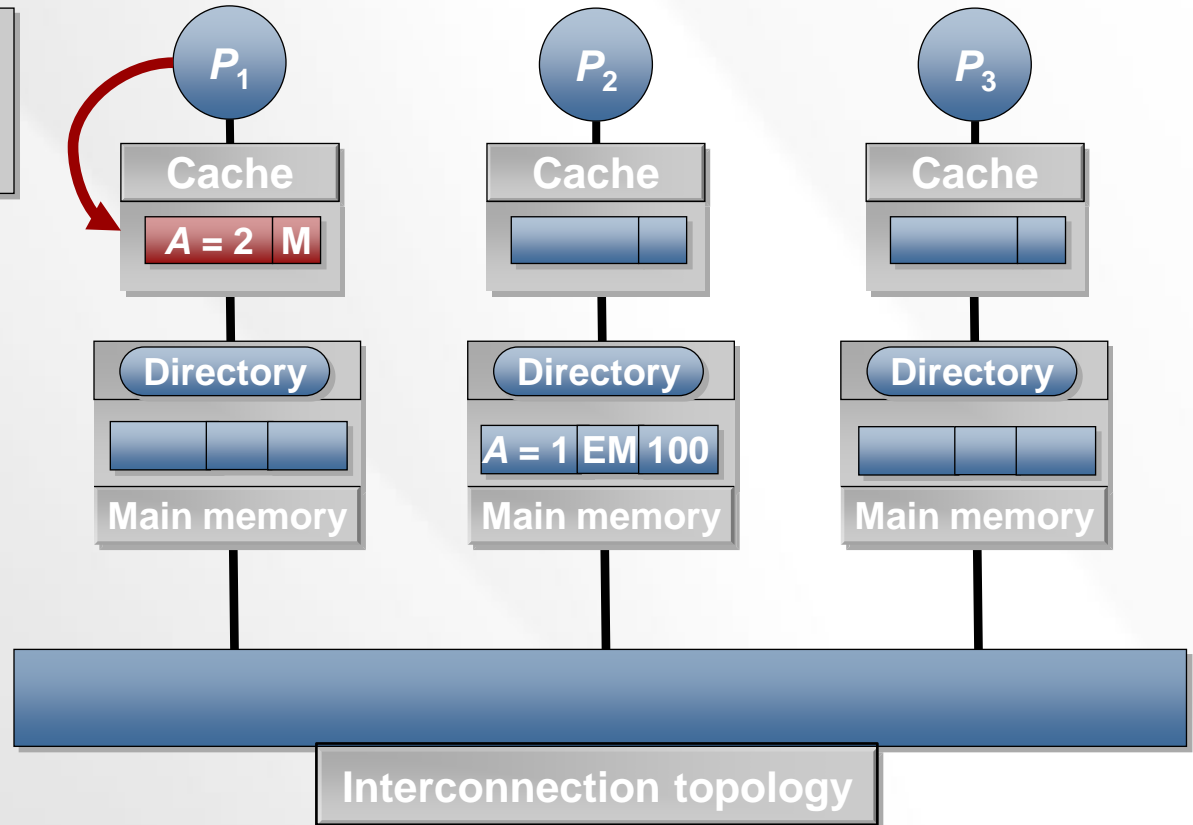
P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A

P_1 Wr $A, \#2$



Full Bit-Vector: Processor P_1 Writes A

Processor P_1 completes writing $A=2$ to its cache.

Trace

P_1 Read A

P_1 Write $A = 2$

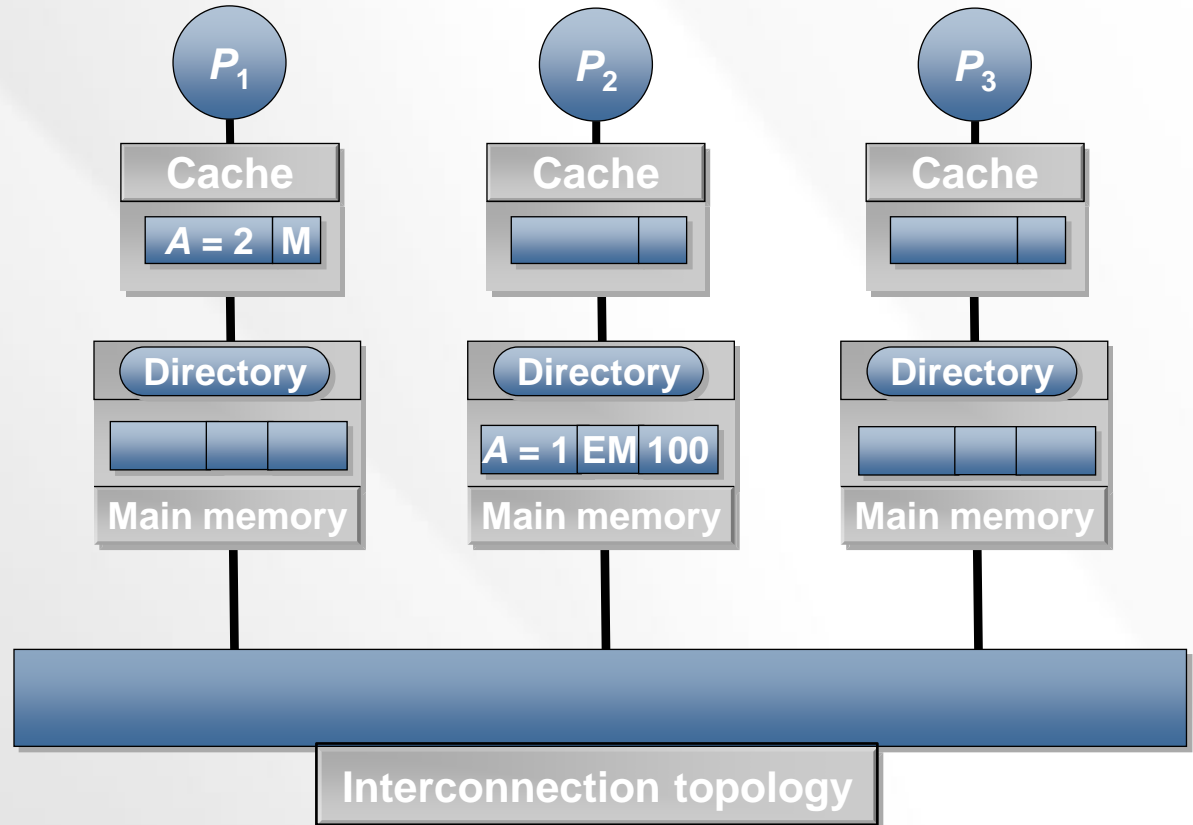
P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A



Full Bit-Vector: Processor P_3 Reads A

Processor P_3 attempts to read A from its cache, misses.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

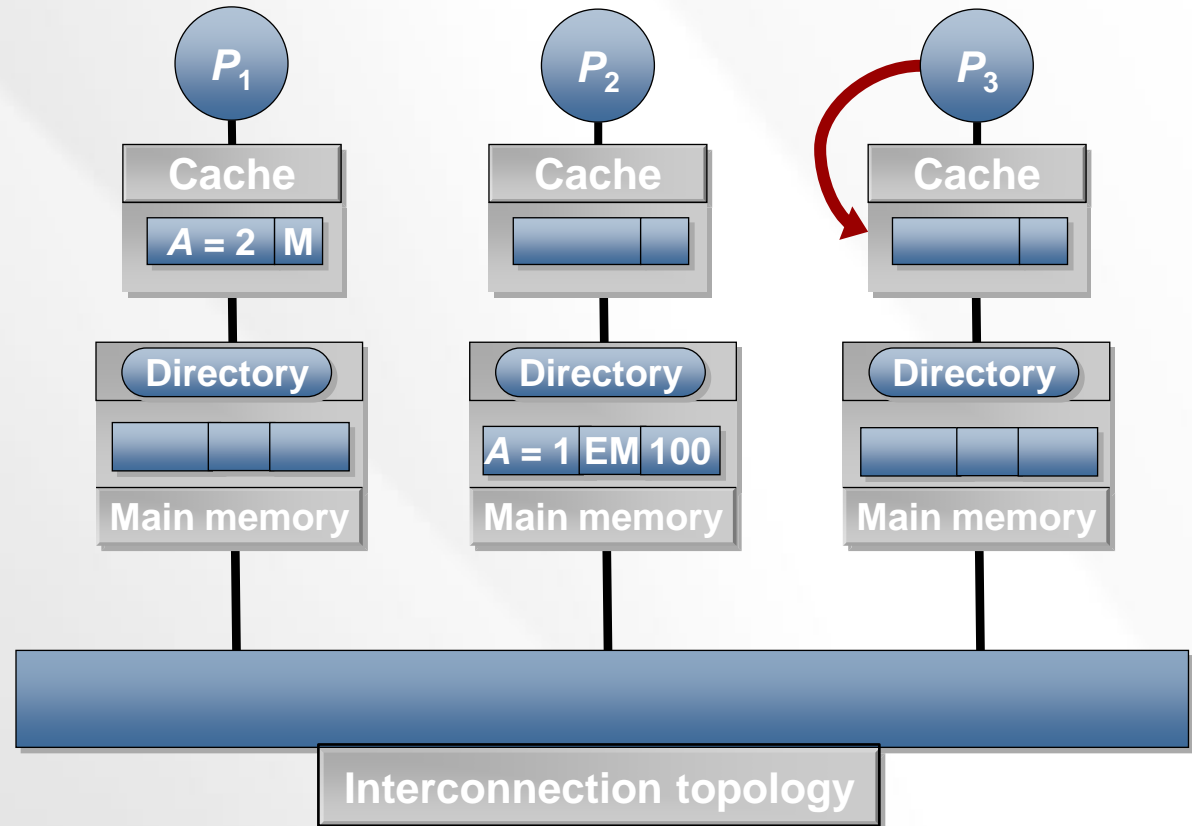
P_2 Read A

P_3 Rd &A

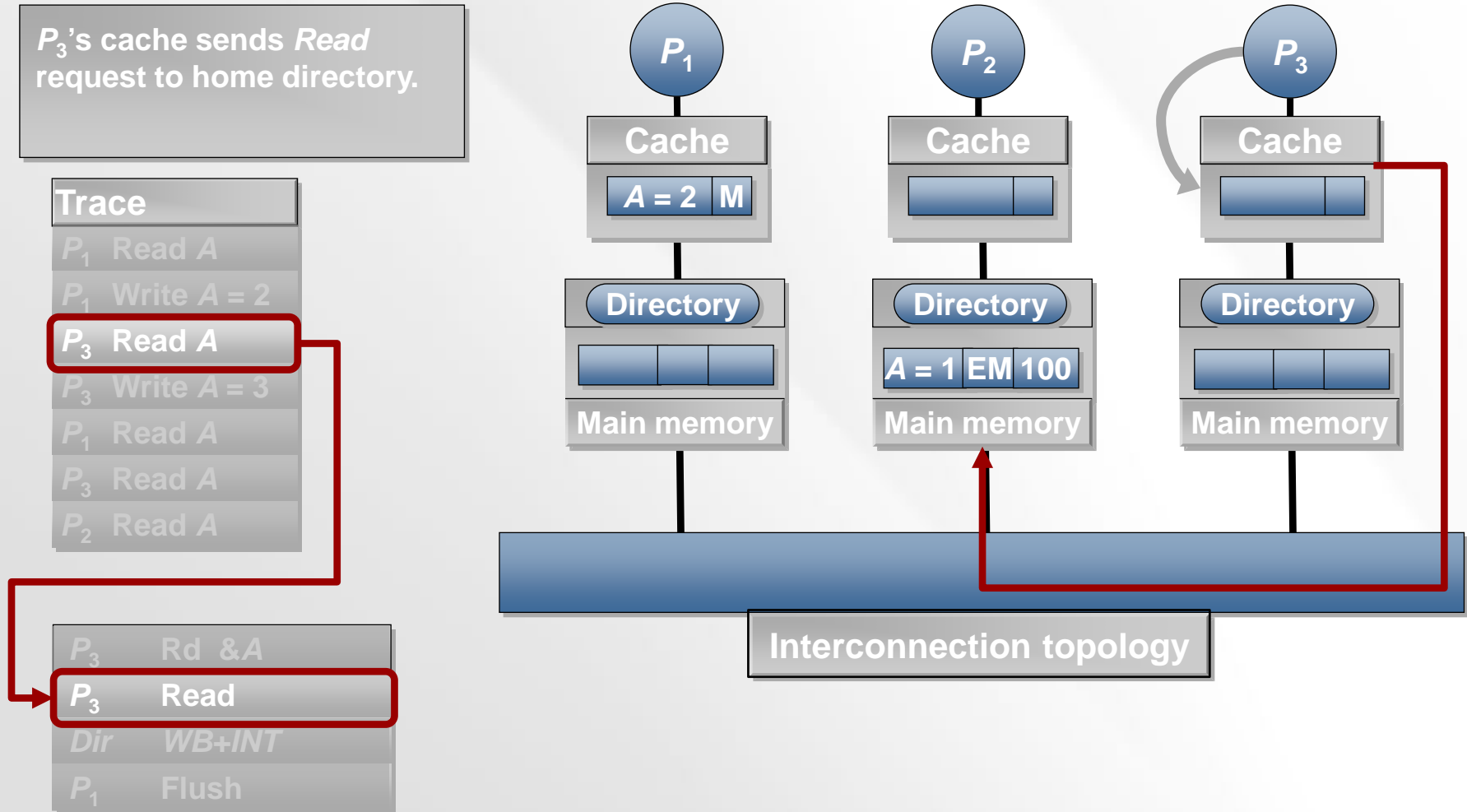
P_3 Read

Dir WB+INT

P_1 Flush



Full Bit-Vector: Processor P_3 Reads A



Full Bit-Vector: Processor P_3 Reads A

Directory sends owning cache a *WB+Int*. Owner P_1 changes state to S.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

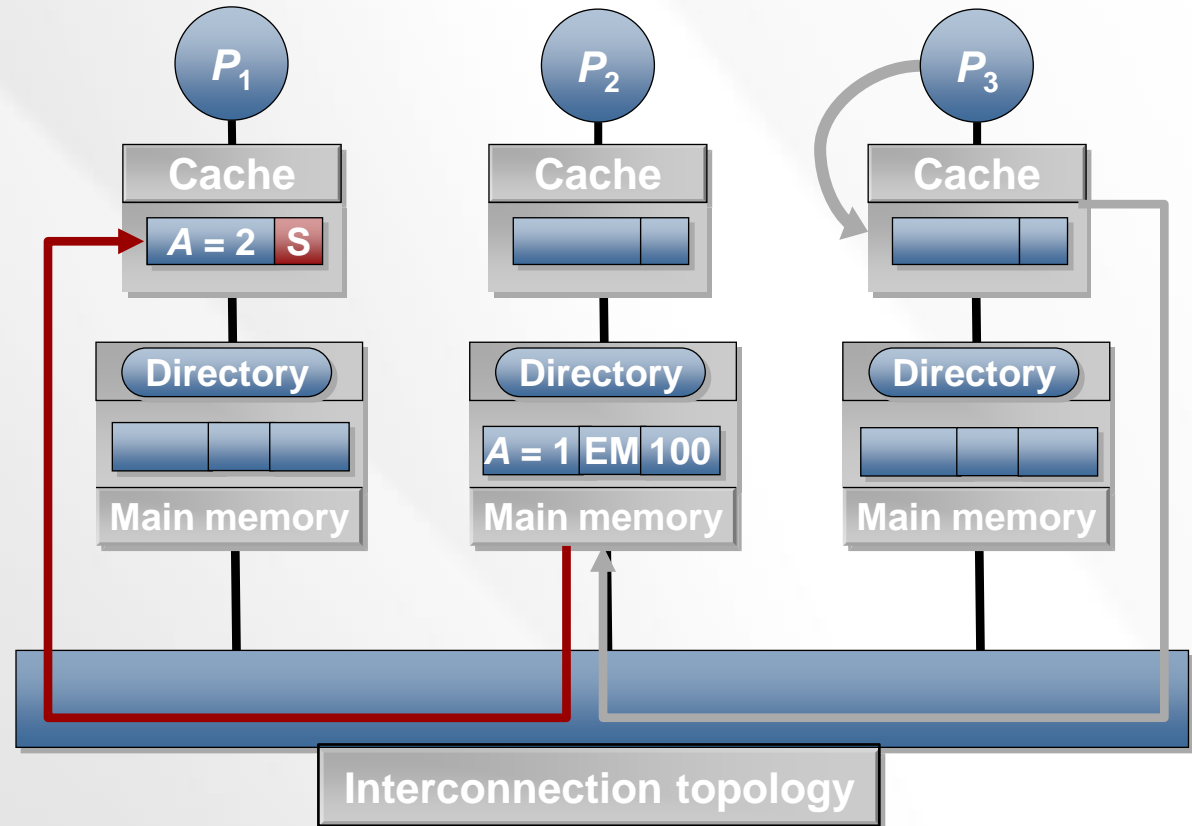
P_2 Read A

P_3 Rd &A

P_3 Read

Dir **WB+Int**

P_1 Flush



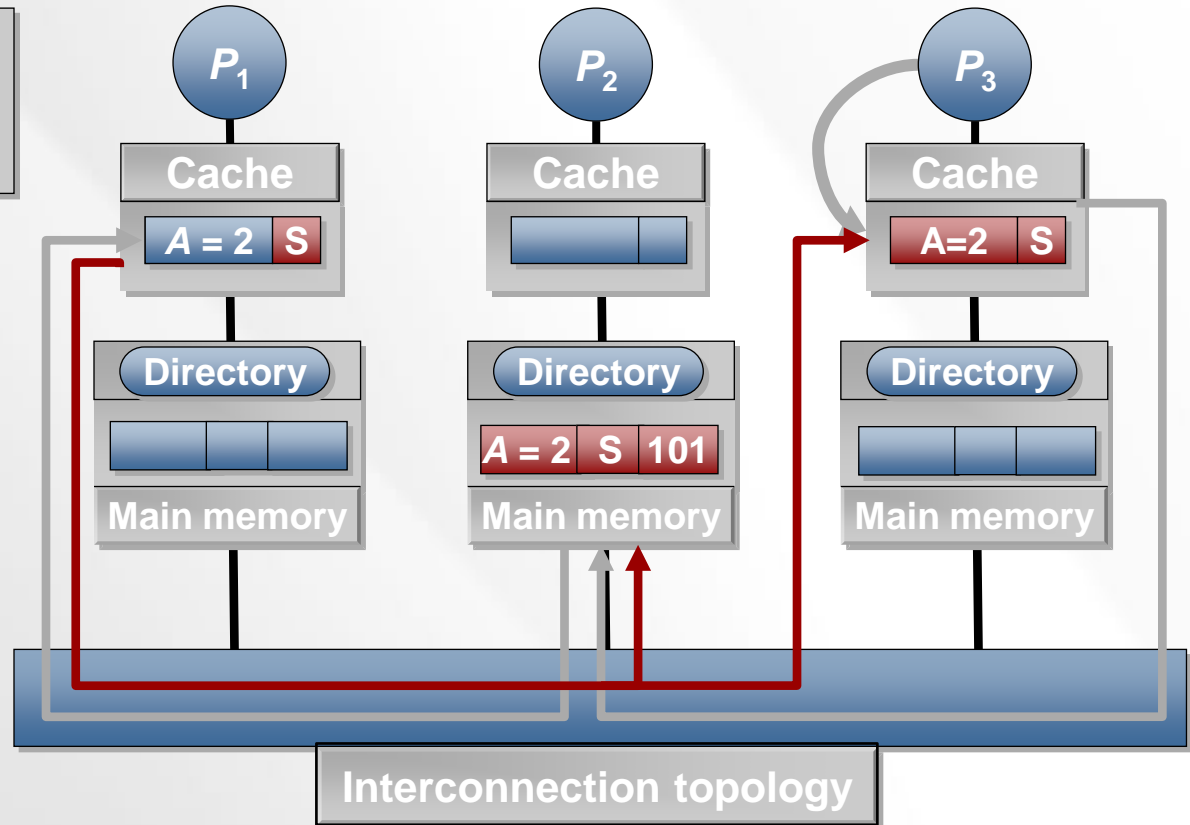
Full Bit-Vector: Processor P_3 Reads A

P_1 flushes the block. Directory and P_3 update the block and set state = S. BV is set to 101.

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_3 Rd & A
 P_3 Read
 Dir WB+INT
 P_1 Flush



Full Bit-Vector: Processor P_3 Reads A

Processor P_3 completes read operation from its cache.

Trace

P_1 Read A

P_1 Write A = 2

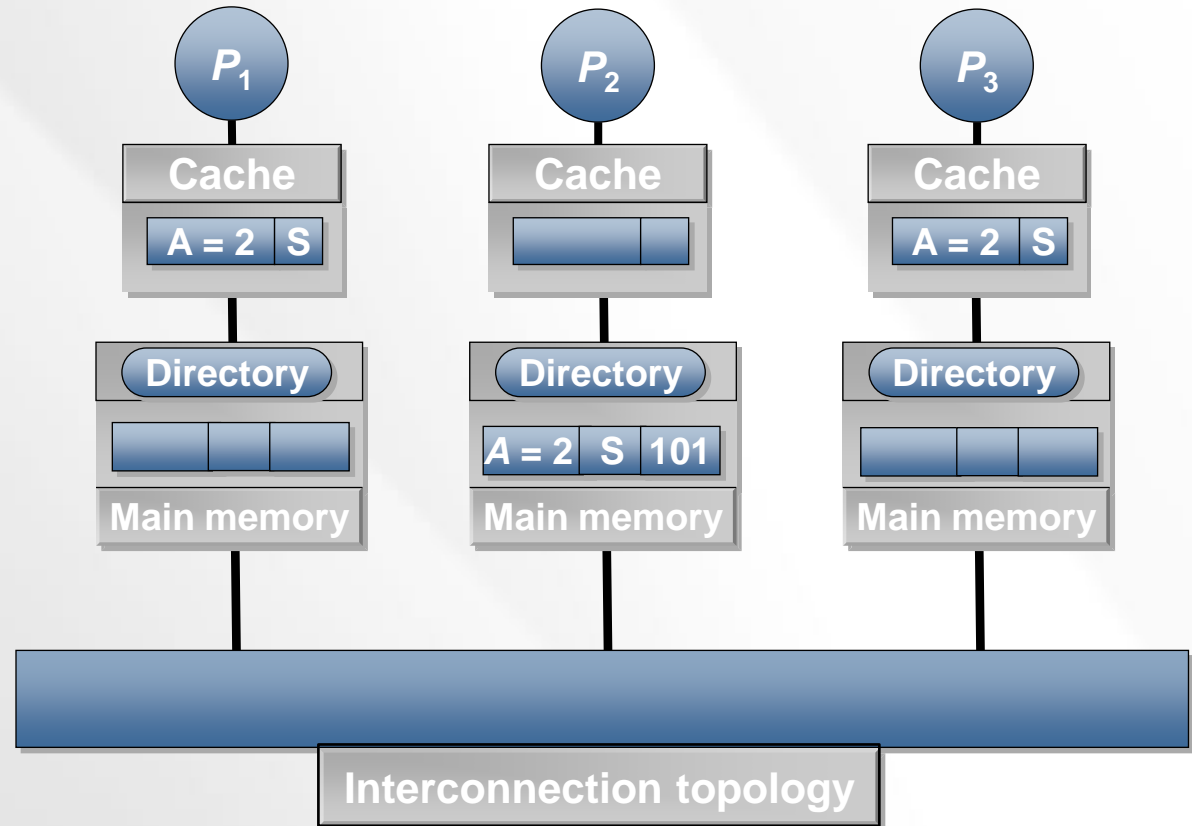
P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

P_2 Read A



Full Bit-Vector: Processor P_3 Writes A

Processor P_3 attempts to write A in its cache.

Trace

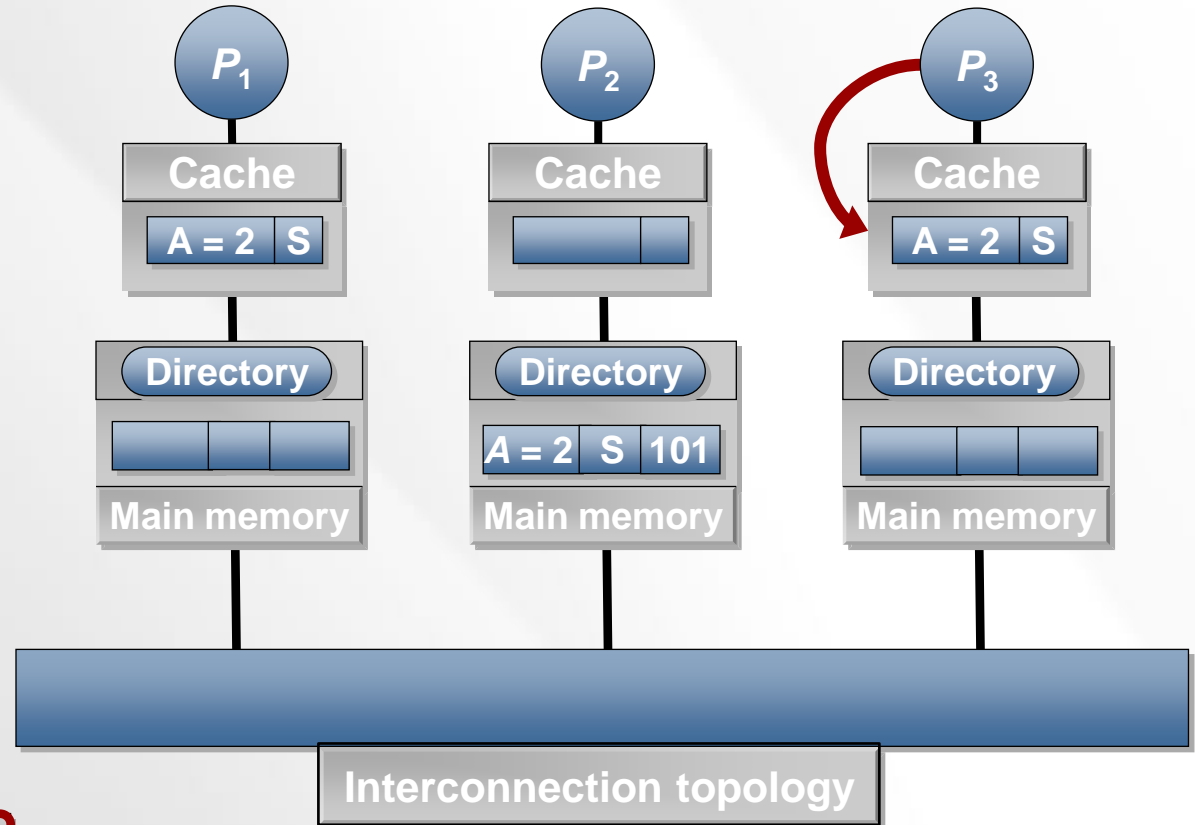
P_1 Read A
 P_1 Write $A = 2$
 P_3 Read A
 P_3 Write $A = 3$
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_3 Wr &A

P_3 Upgr

Dir Inv, Reply

P_1 InvAck



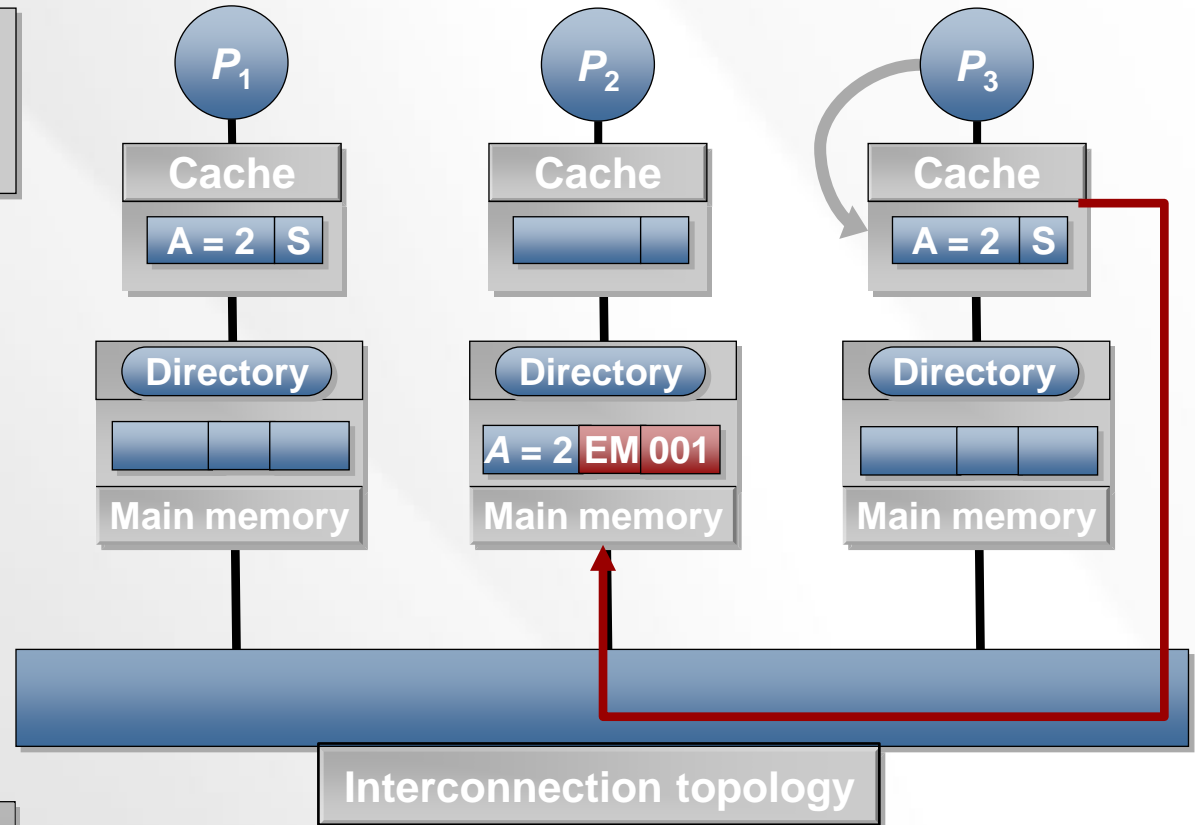
Full Bit-Vector: Processor P_3 Writes A

P_3 sends *Upgr* request to the directory. BV is set to 001, as P_3 becomes the owner.

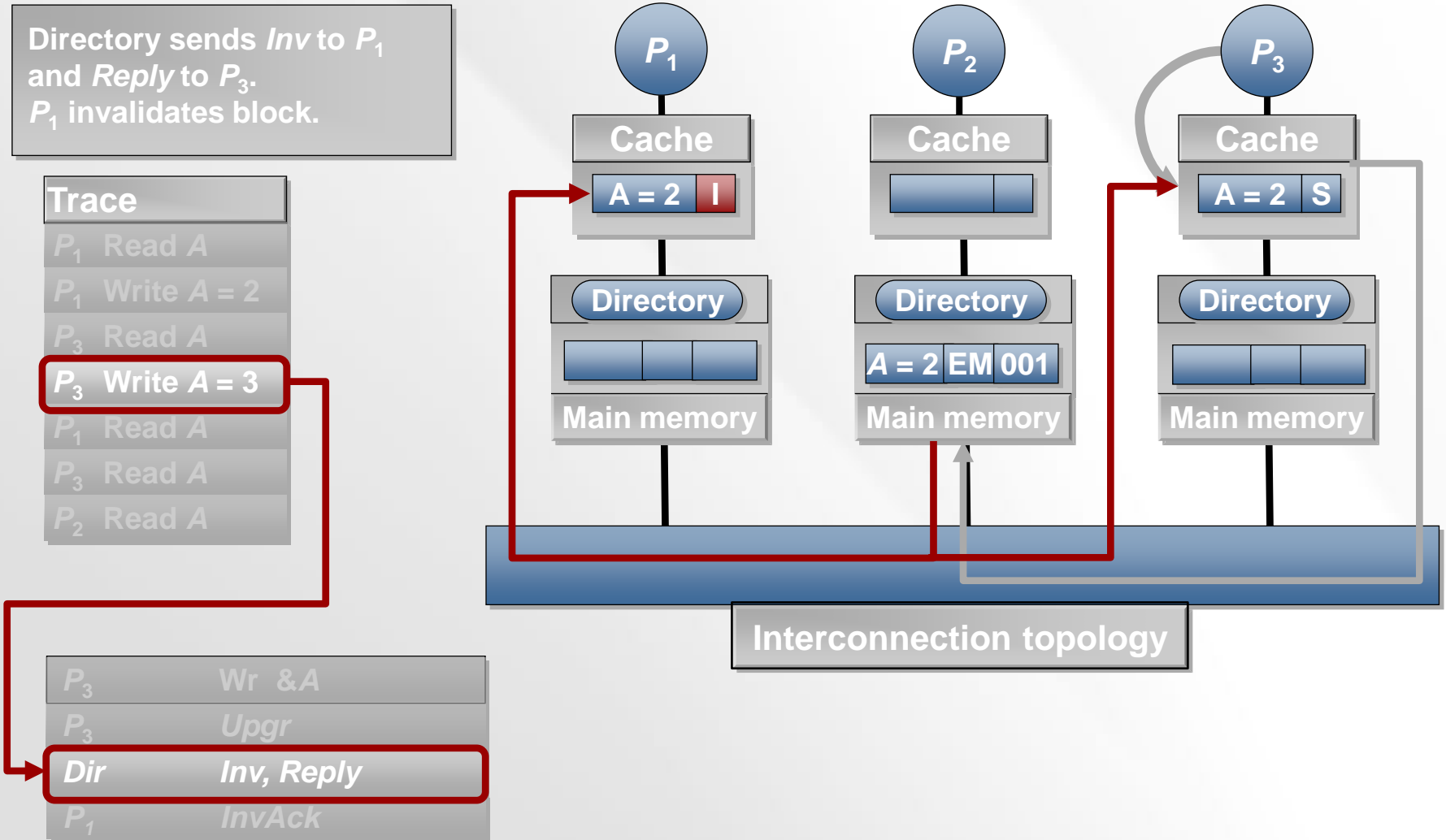
Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_3 Wr &A
 P_3 Upgr
 Dir Inv, Reply
 P1 InvAck



Full Bit-Vector: Processor P_3 Writes A

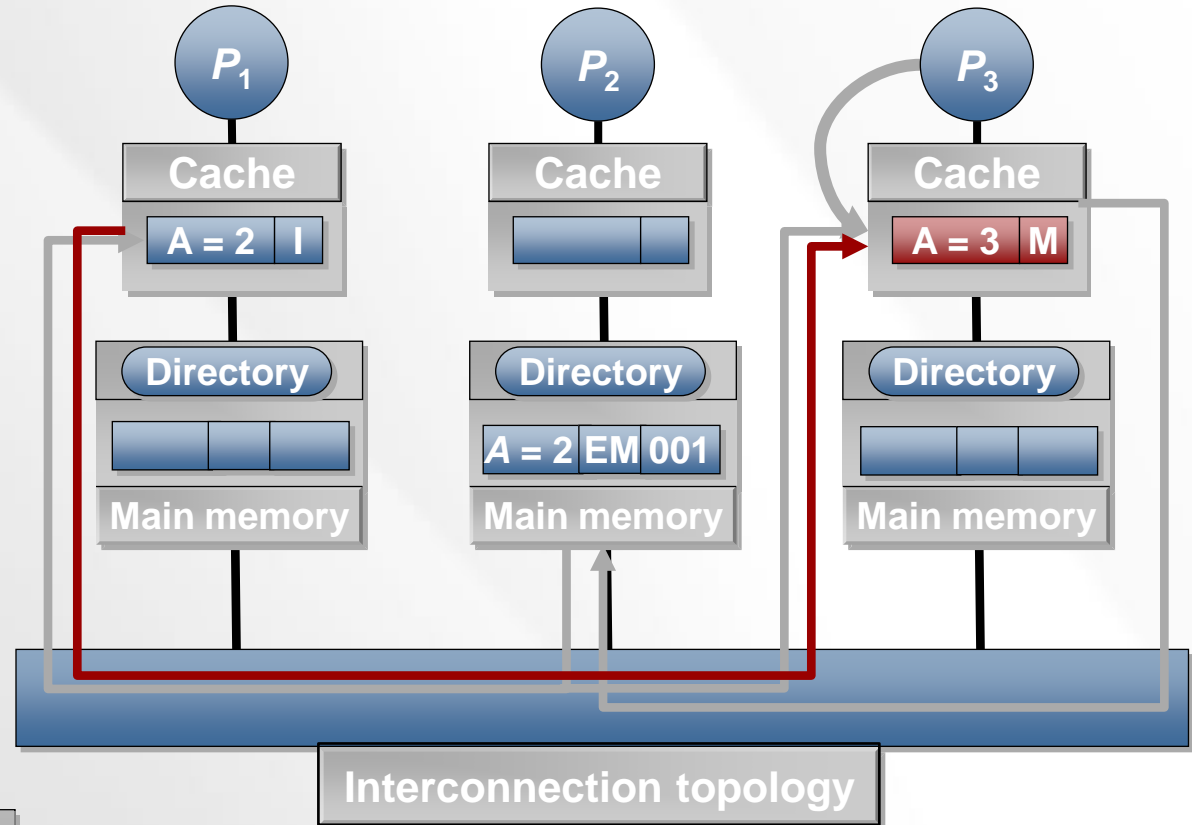


Full Bit-Vector: Processor P_3 Writes A

Processor P_1 sends *InvAck* and P_3 proceeds with the write, becomes owner.

Trace

P_1	Read A
P_1	Write A = 2
P_3	Read A
P_3	Write A = 3
P_1	Read A
P_3	Read A
P_2	Read A



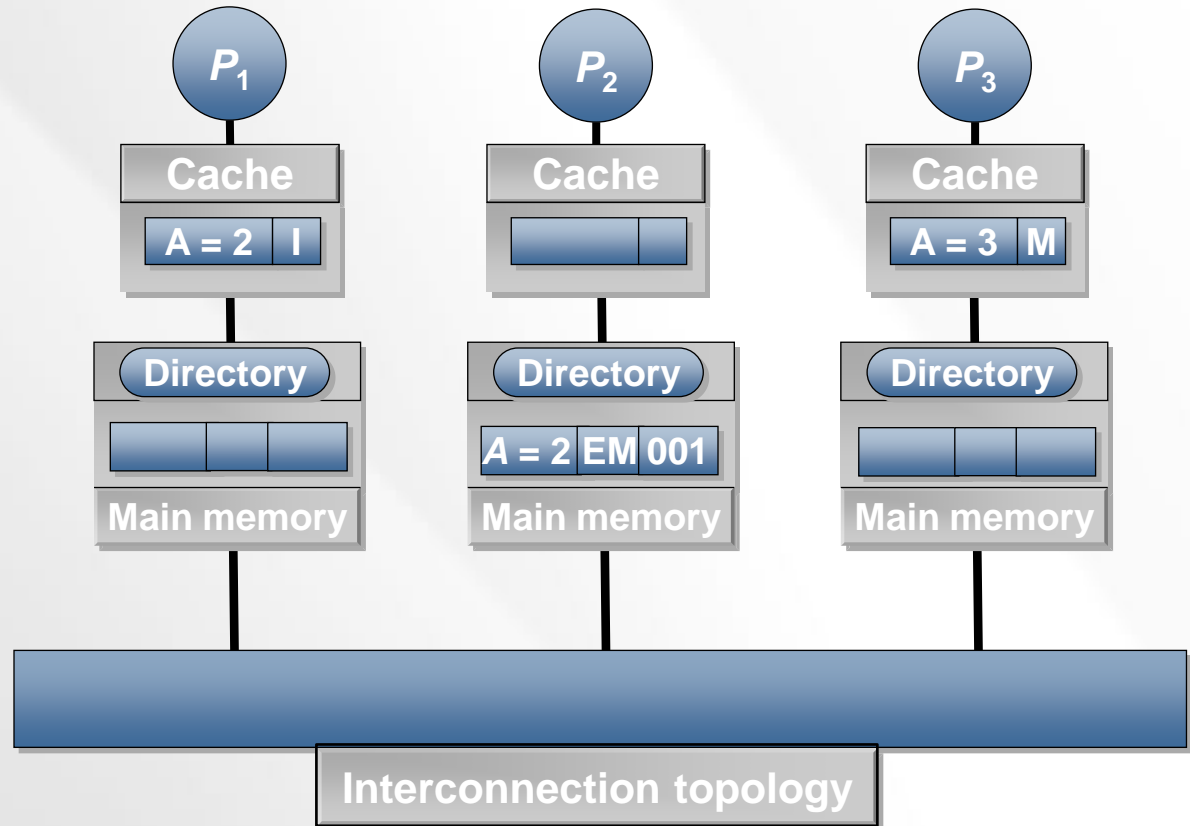
P_3	Wr &A
P_3	Upgr
Dir	Inv, Reply
P_1	<i>InvAck</i>

Full Bit-Vector: Processor P_3 Writes A

Processor P_3 completes writing $A=3$ to its cache.

Trace

P_1 Read A
 P_1 Write $A = 2$
 P_3 Read A
 P_3 Write $A = 3$
 P_1 Read A
 P_3 Read A
 P_2 Read A



Full Bit-Vector: Processor P_1 Reads A

Processor P_1 attempts to read A from its cache, misses.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

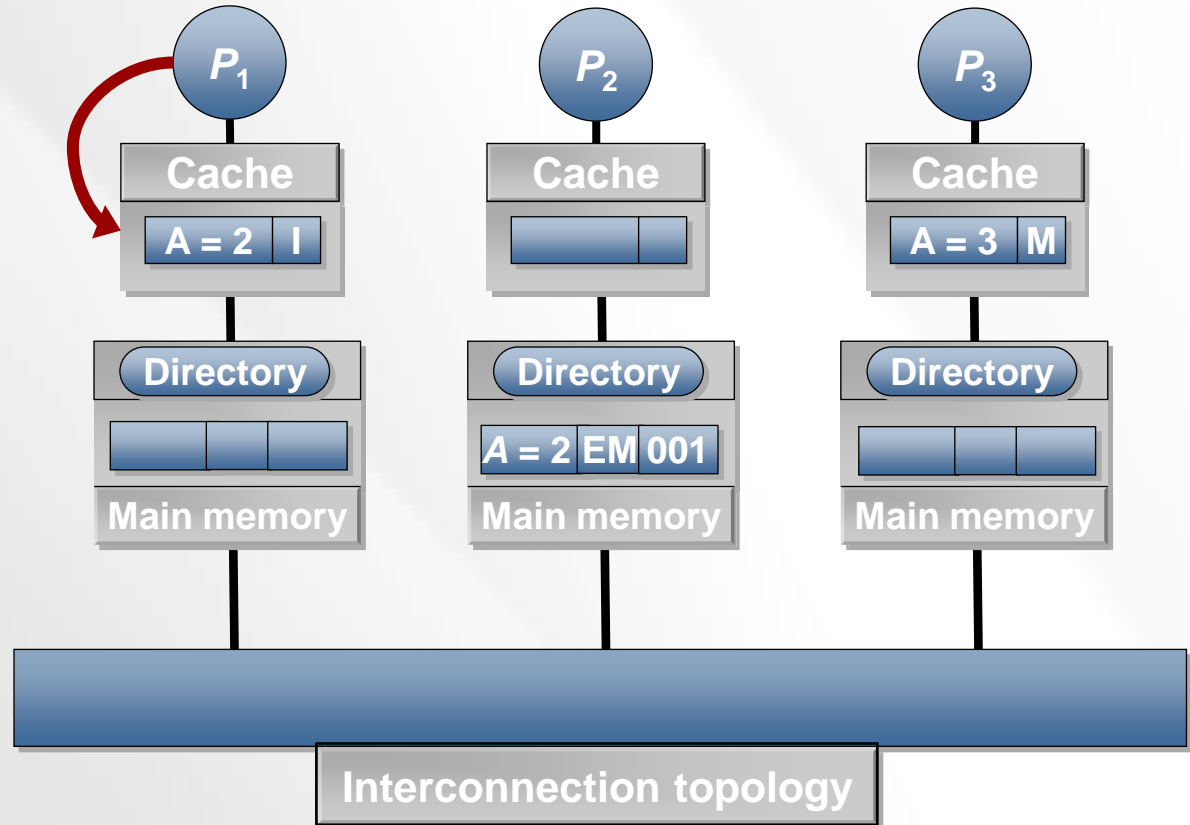
P_2 Read A

P_1 Rd &A

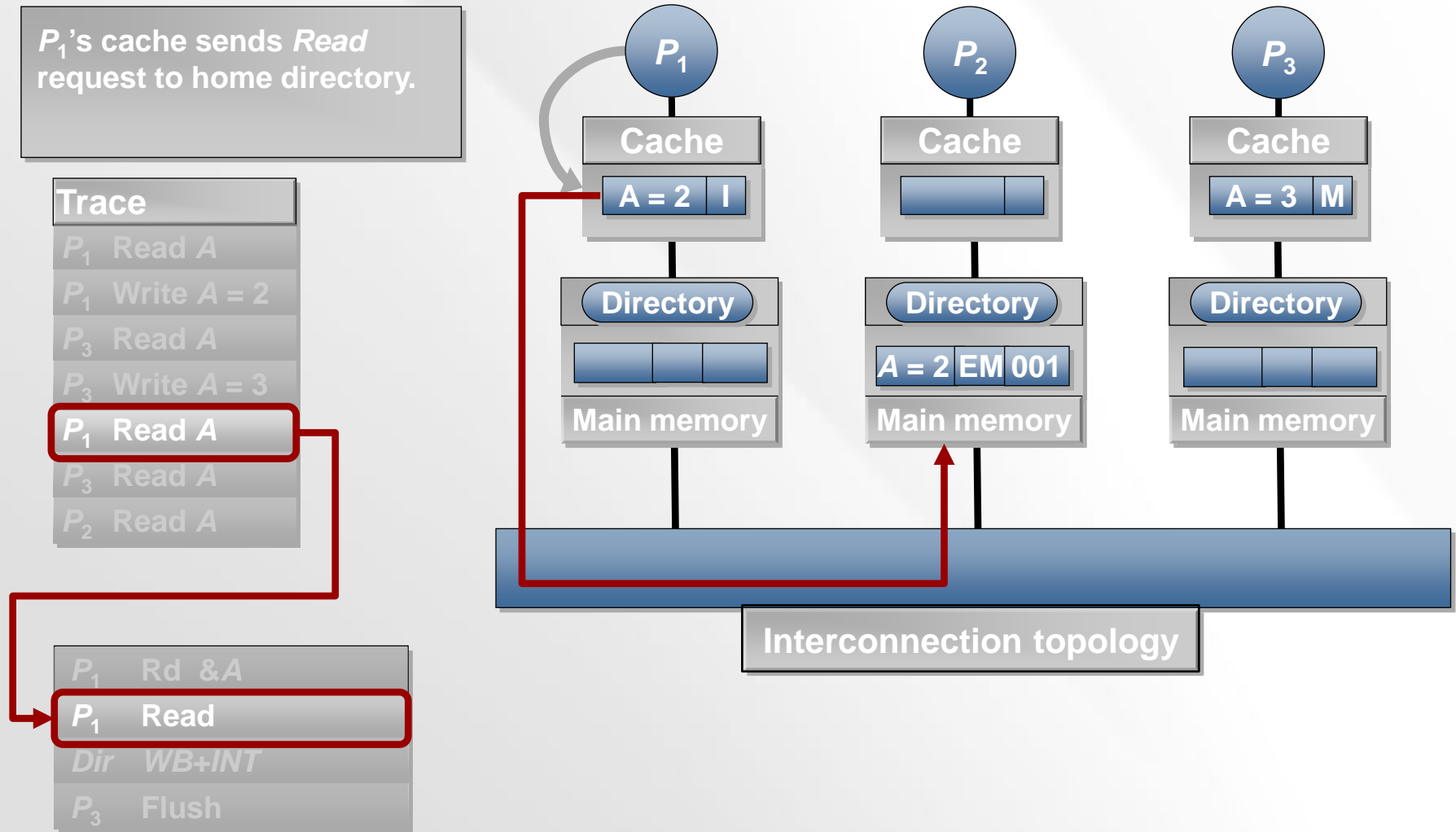
P_1 Read

Dir WB+INT

P_3 Flush



Full Bit-Vector: Processor P_1 Reads A



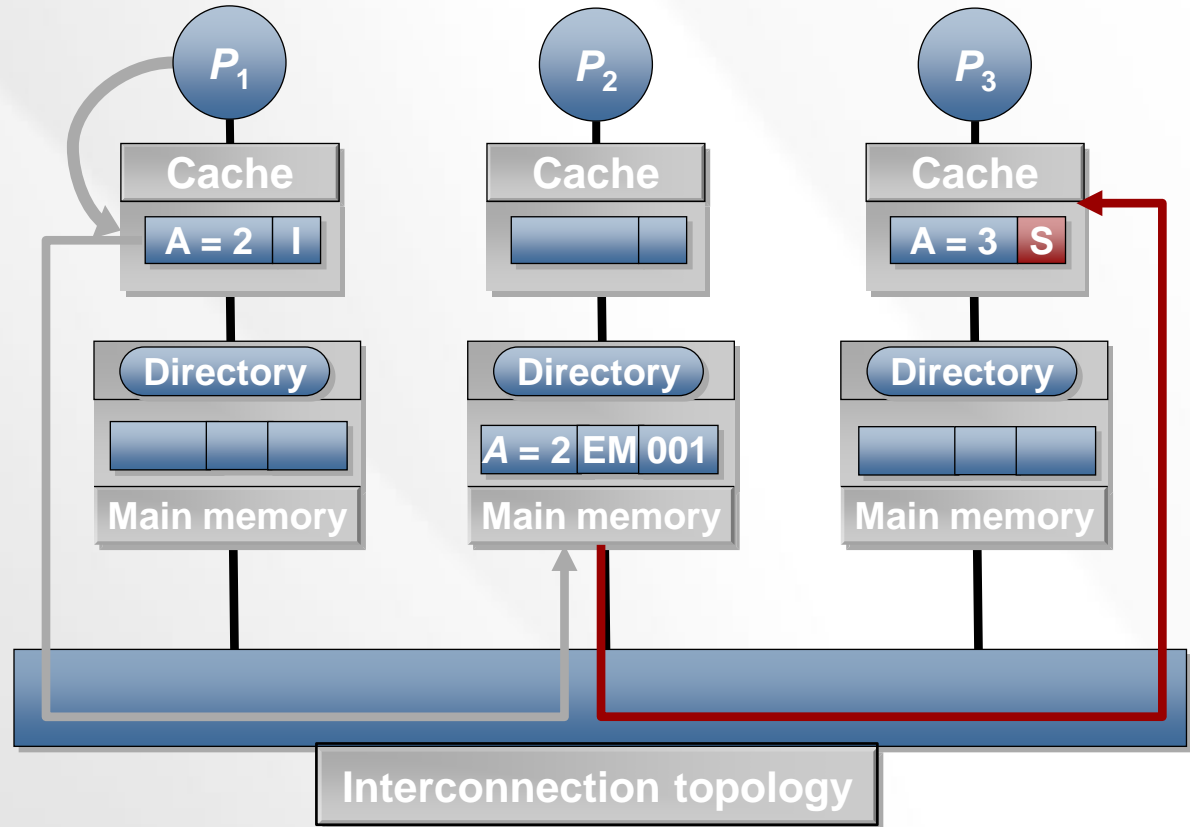
Full Bit-Vector: Processor P_1 Reads A

Directory sends *WB+Int* to the owner cache. The owner is downgraded to state S.

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_1 Rd &A
 P_1 Read
Dir WB+Int
 P_3 Flush



Full Bit-Vector: Processor P_1 Reads A

P_3 flushes the block to the directory and P_1 's cache. State is set to S. BV is set to 101.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

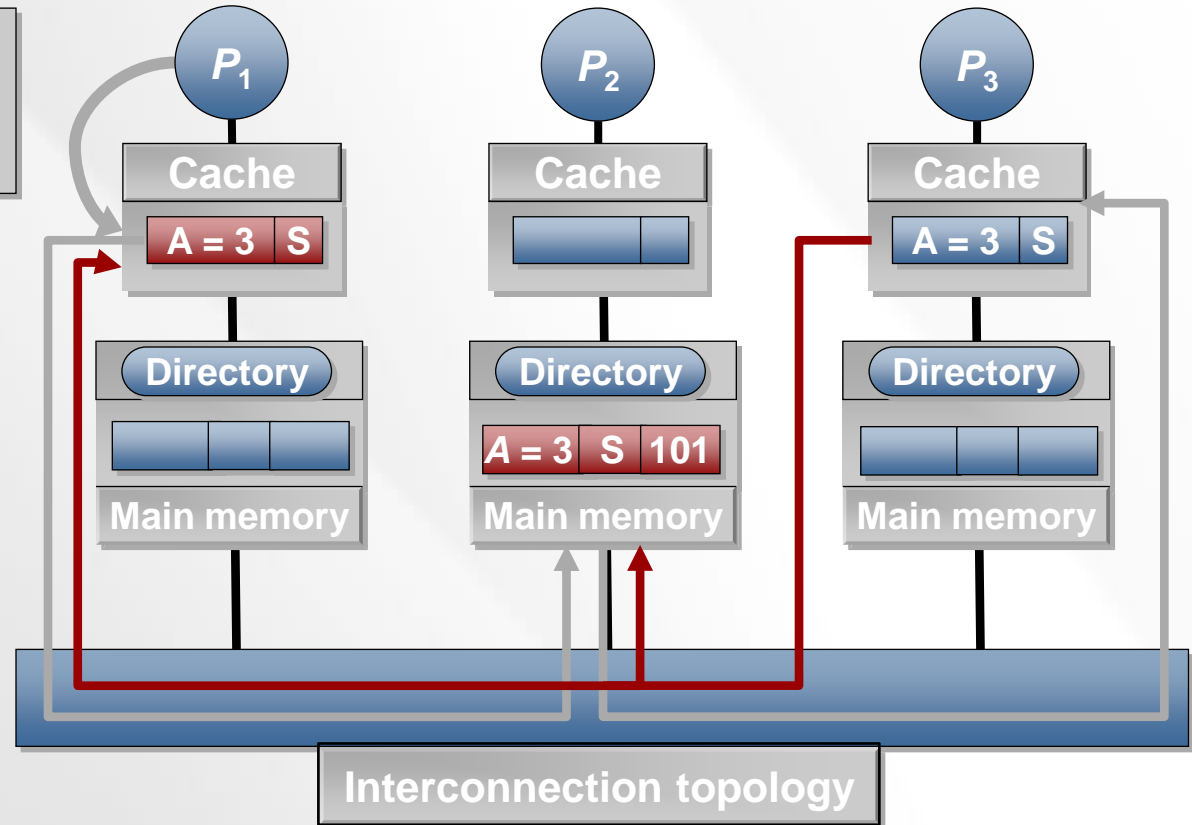
P_2 Read A

P_1 Rd &A

P_1 Read

Dir WB+INT

P_3 Flush



Full Bit-Vector: Processor P_1 Reads A

Processor P_1 completes reading A from its cache.

Trace

P_1 Read A

P_1 Write $A = 2$

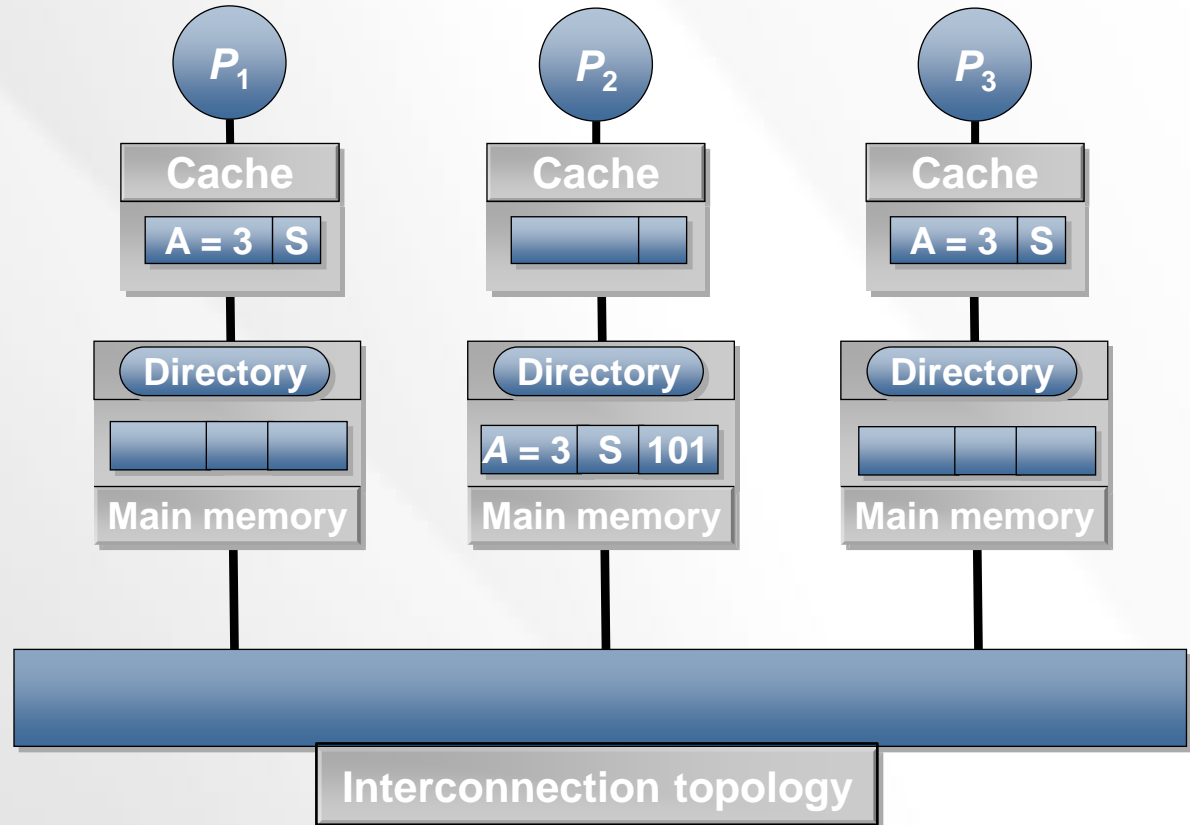
P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A



Full Bit-Vector: Processor P_3 Reads A

Processor P_3 attempts to read A from its cache, hits.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

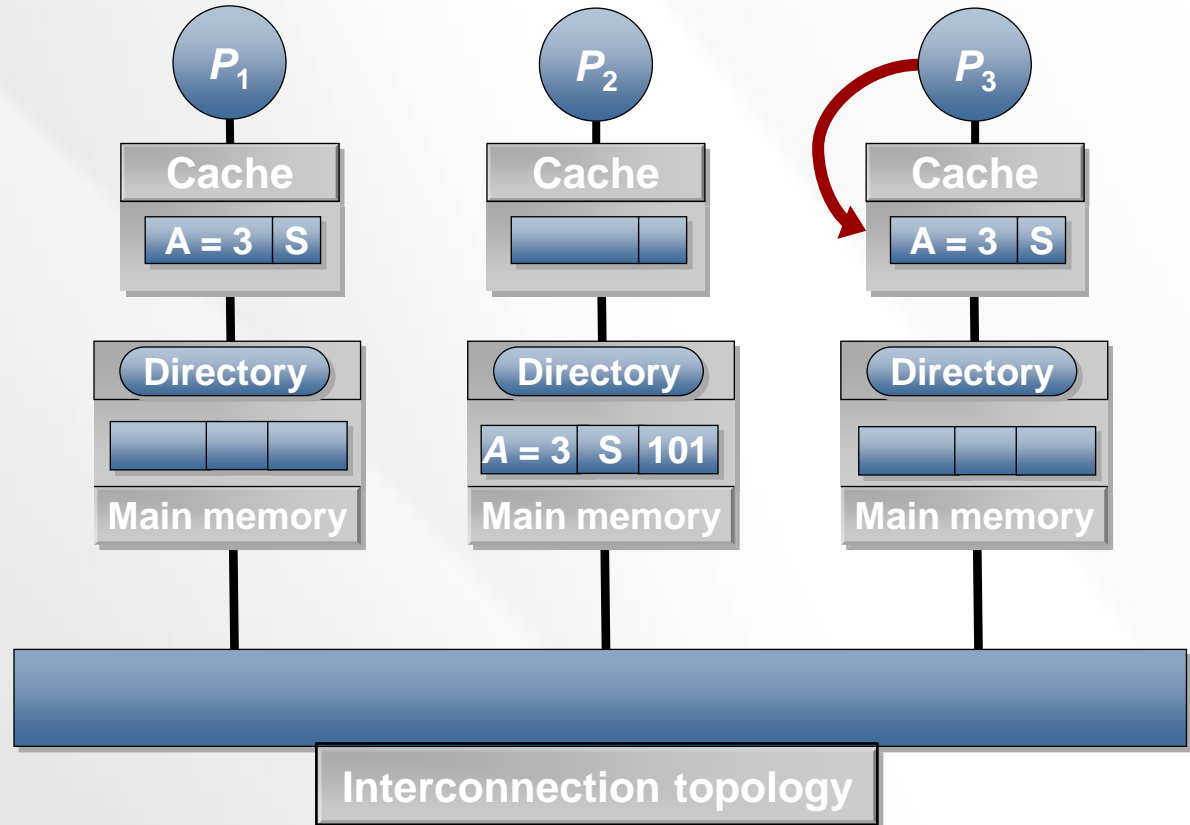
P_1 Read A

P_3 Read A

P_2 Read A

P_3 Rd &A

P_3 returns data



Full Bit-Vector: Processor P_3 Reads A

P_3 's cache returns the value of A immediately.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

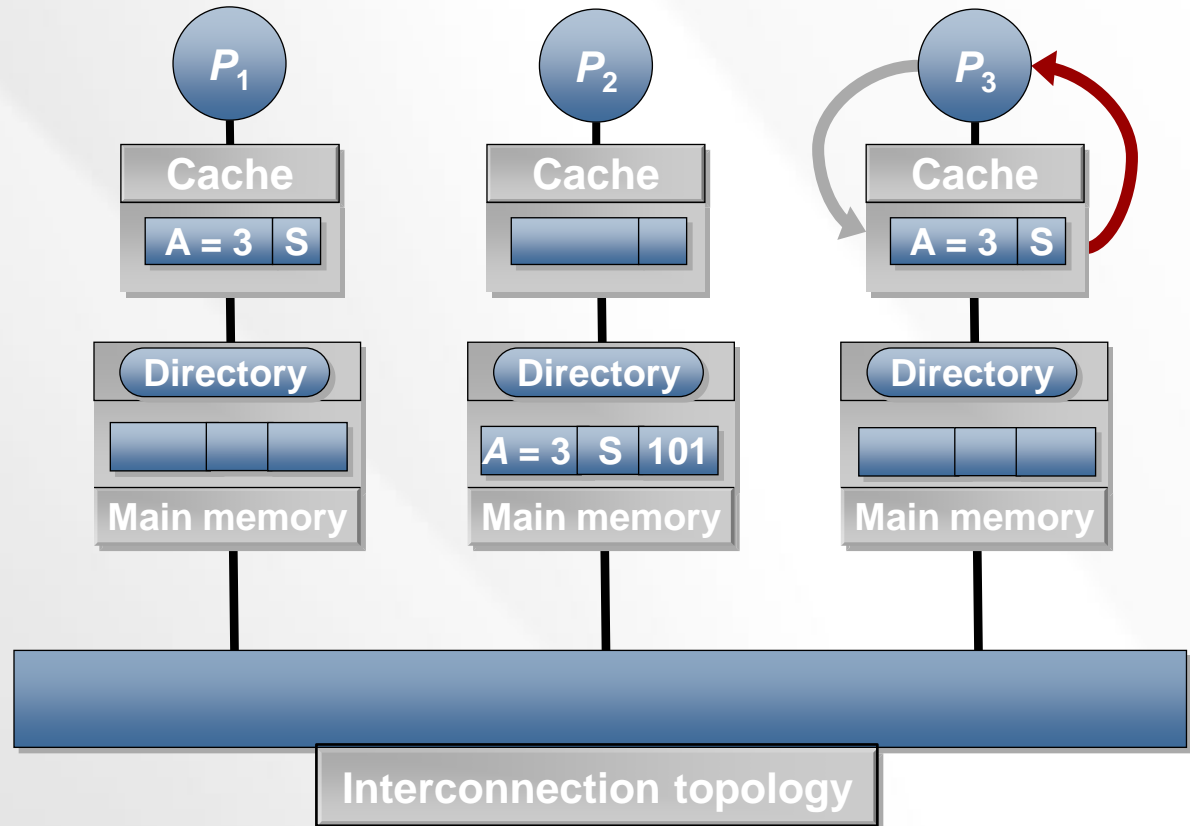
P_1 Read A

P_3 Read A

P_2 Read A

P_3 Rd &A

P_3 returns data



Full Bit-Vector: Processor P_3 Reads A

Processor P_1 completes reading A from its cache.

Trace

P_1 Read A

P_1 Write $A = 2$

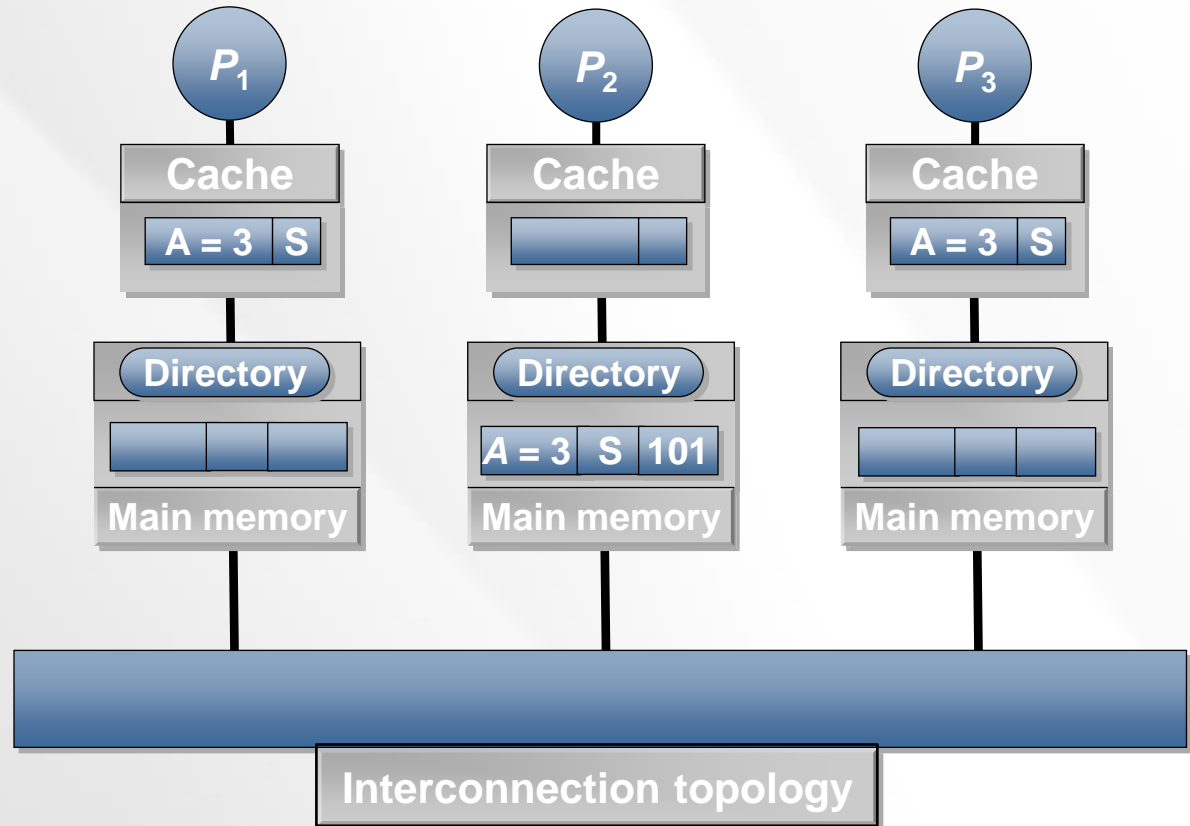
P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A



Full Bit-Vector: Processor P_2 Reads A

Processor P_2 attempts to read A from its cache.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

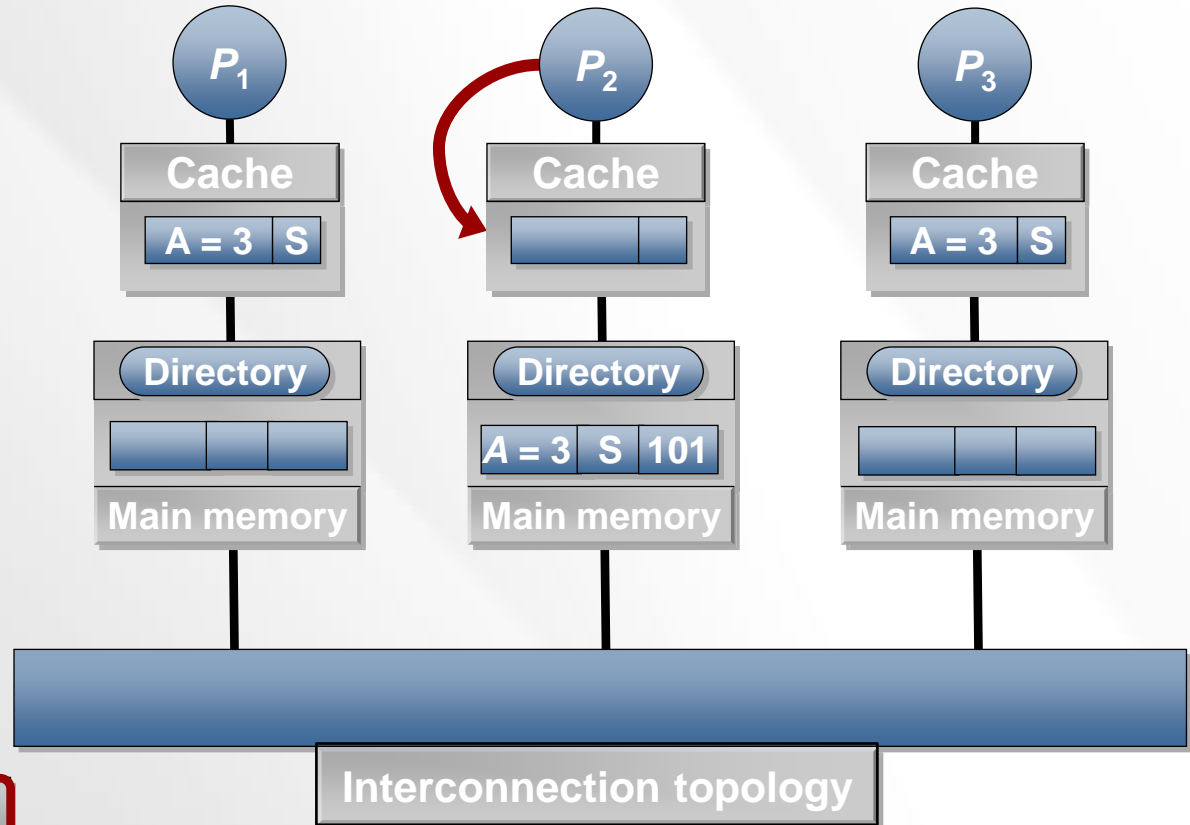
P_3 Read A

P_2 Read A

P_2 Rd &A

P_2 Read

Dir ReplyD



Full Bit-Vector: Processor P_2 Reads A

P_2 's cache sends a *Read* request to its directory. BV is set to 111.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

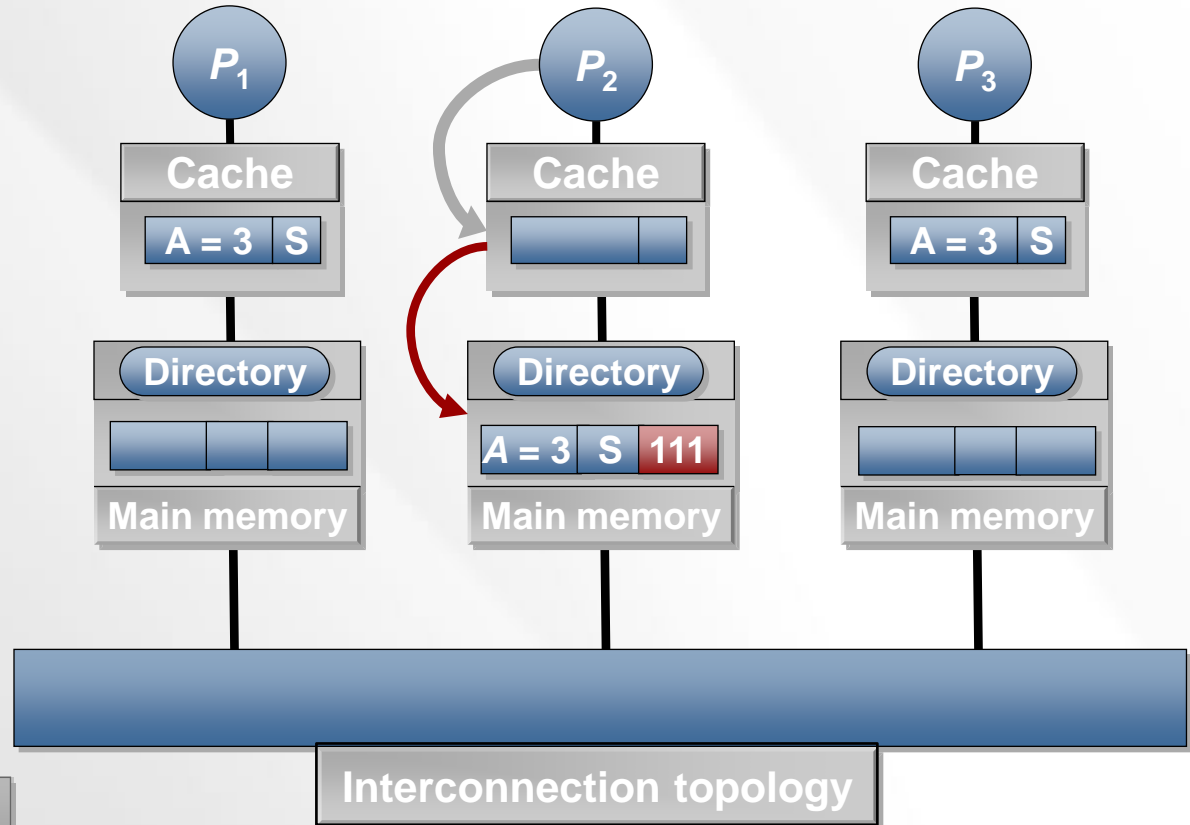
P_3 Read A

P_2 Read A

P_2 Rd &A

P_2 Read

Dir ReplyD



Full Bit-Vector: Processor P_2 Reads A

Directory returns the block and state with *ReplyD*.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

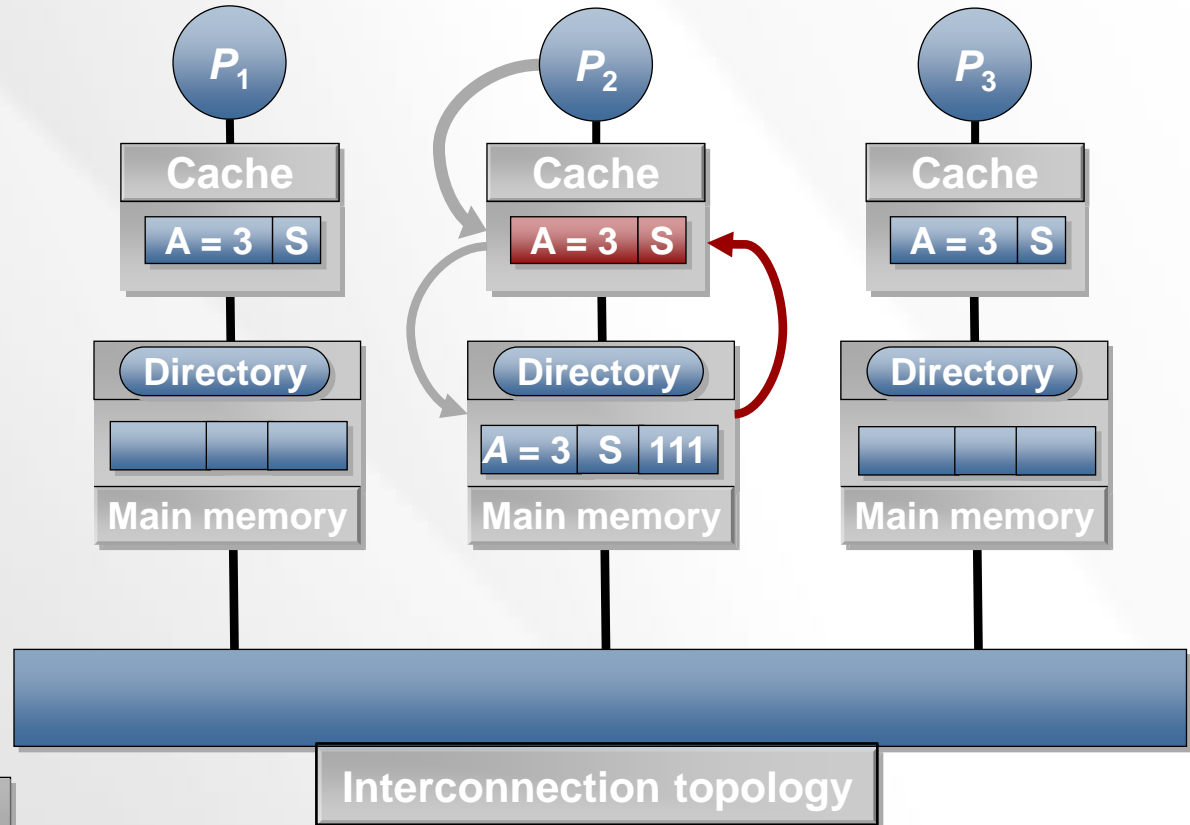
P_3 Read A

P_2 Read A

P_2 Rd &A

P_2 Read

Dir ReplyD



Full Bit-Vector: Processor P_2 Reads A

Processor P_2 finishes *Read* operation.

Trace

P_1 Read A

P_1 Write $A = 2$

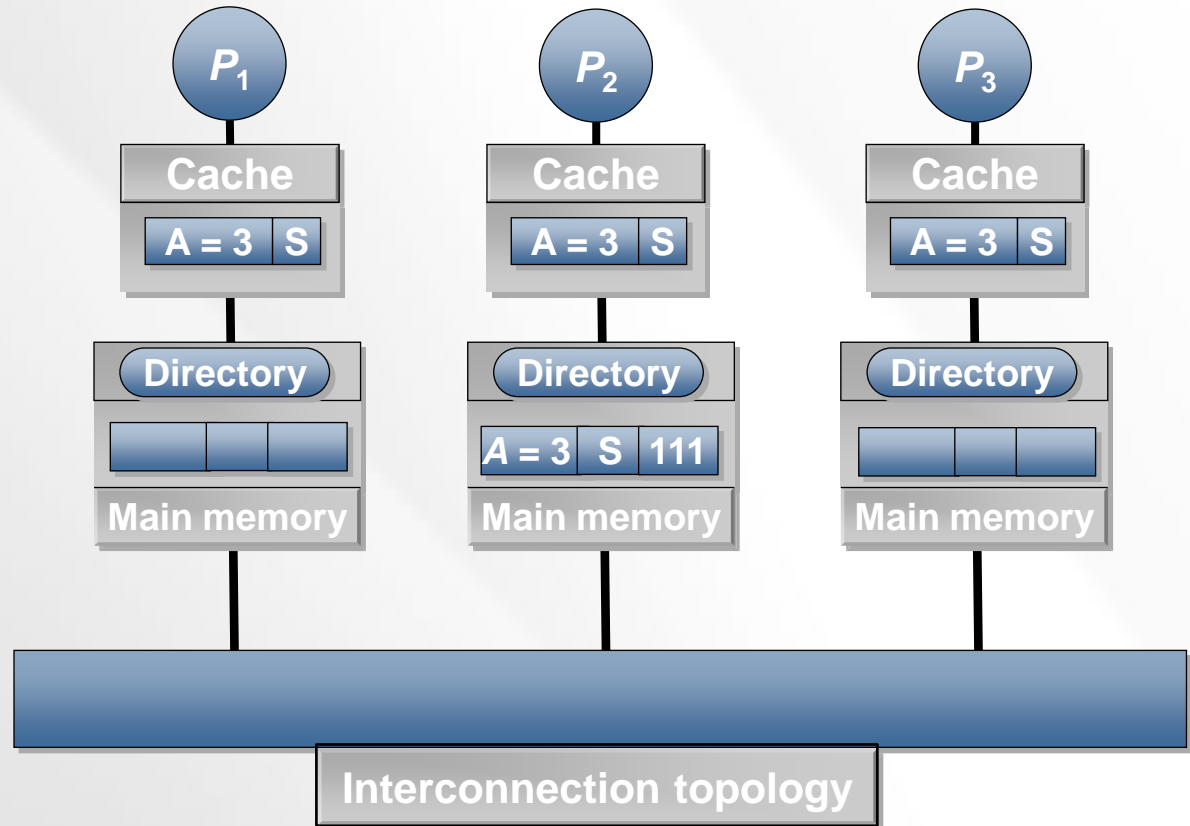
P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A

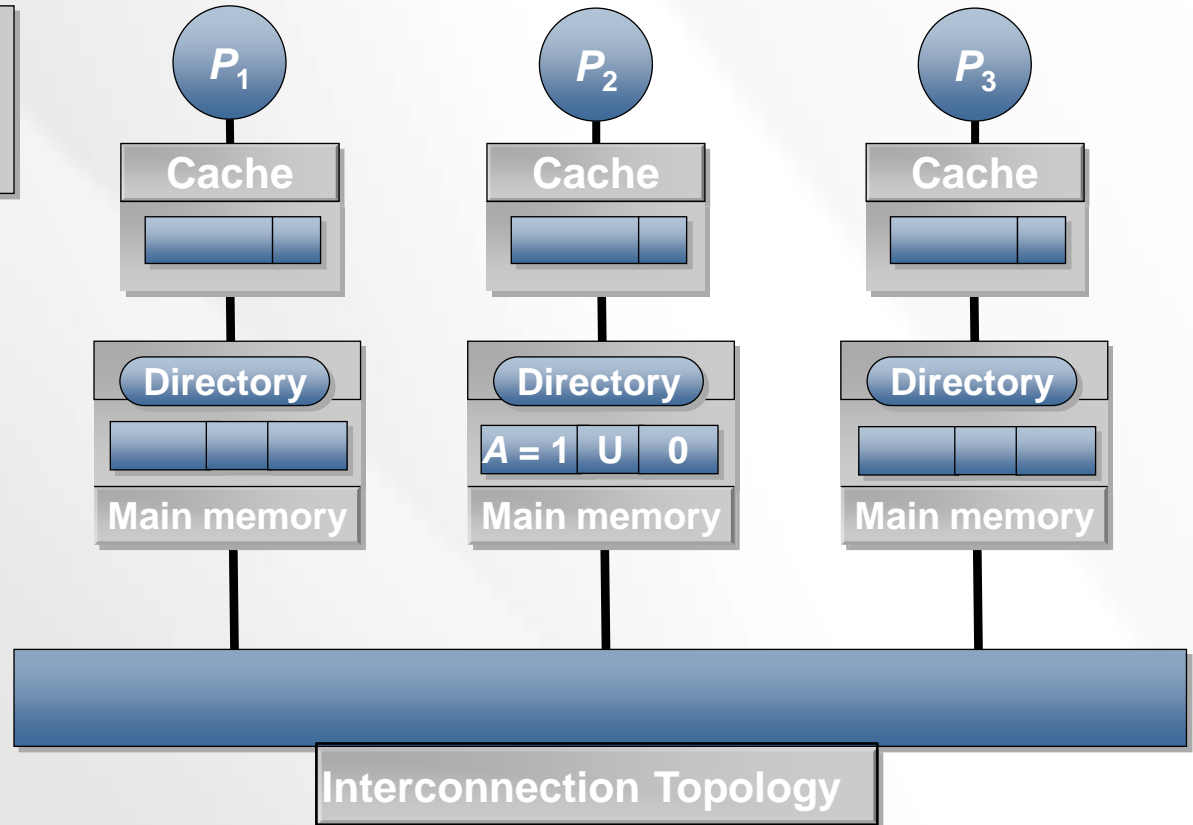


SSCI Visualization – Start State

Start state. All caches empty and main memory has $A = 1$ in state U.

Trace

P_1 Read A
 P_1 Write $A = 2$
 P_3 Read A
 P_3 Write $A = 3$
 P_1 Read A
 P_3 Read A
 P_2 Read A



SSCI: Processor P_1 Reads A

Processor P_1 attempts to read A from its cache.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

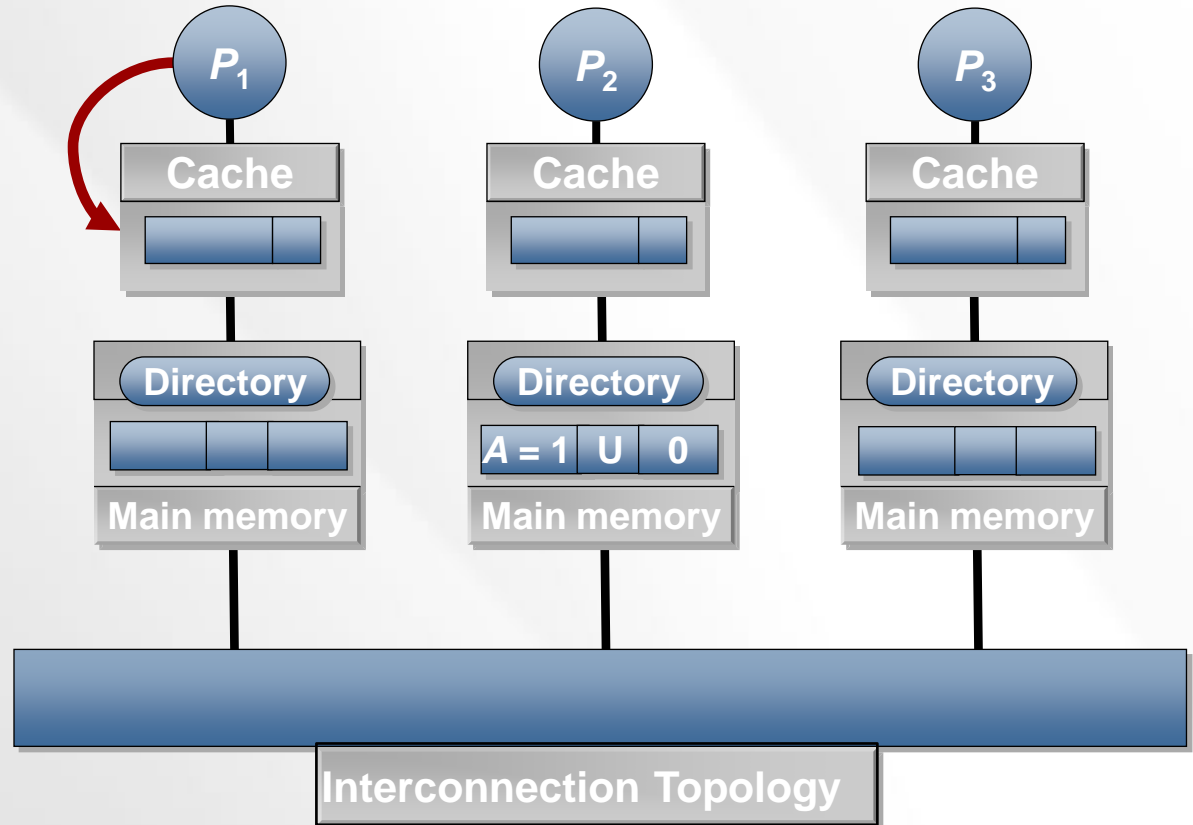
P_3 Read A

P_2 Read A

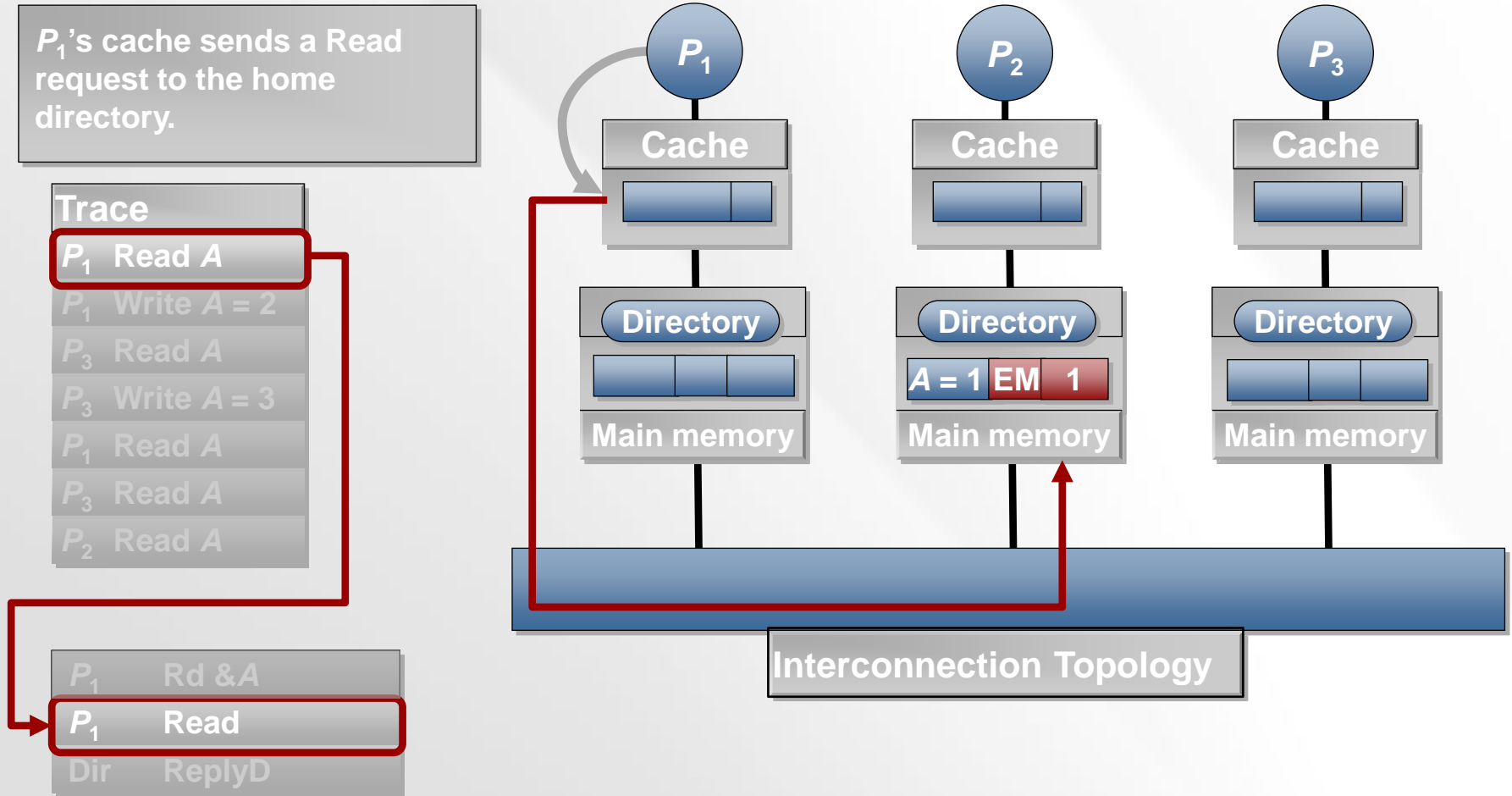
P_1 Rd &A

P_1 Read

Dir ReplyD



SSCI: Processor P_1 Reads A



SSCI: Processor P_1 Reads A

Directory replies with value of A and the state of A in P_1 's cache is set to E. The cache line contains fields *state*, *prev*, and *next*

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

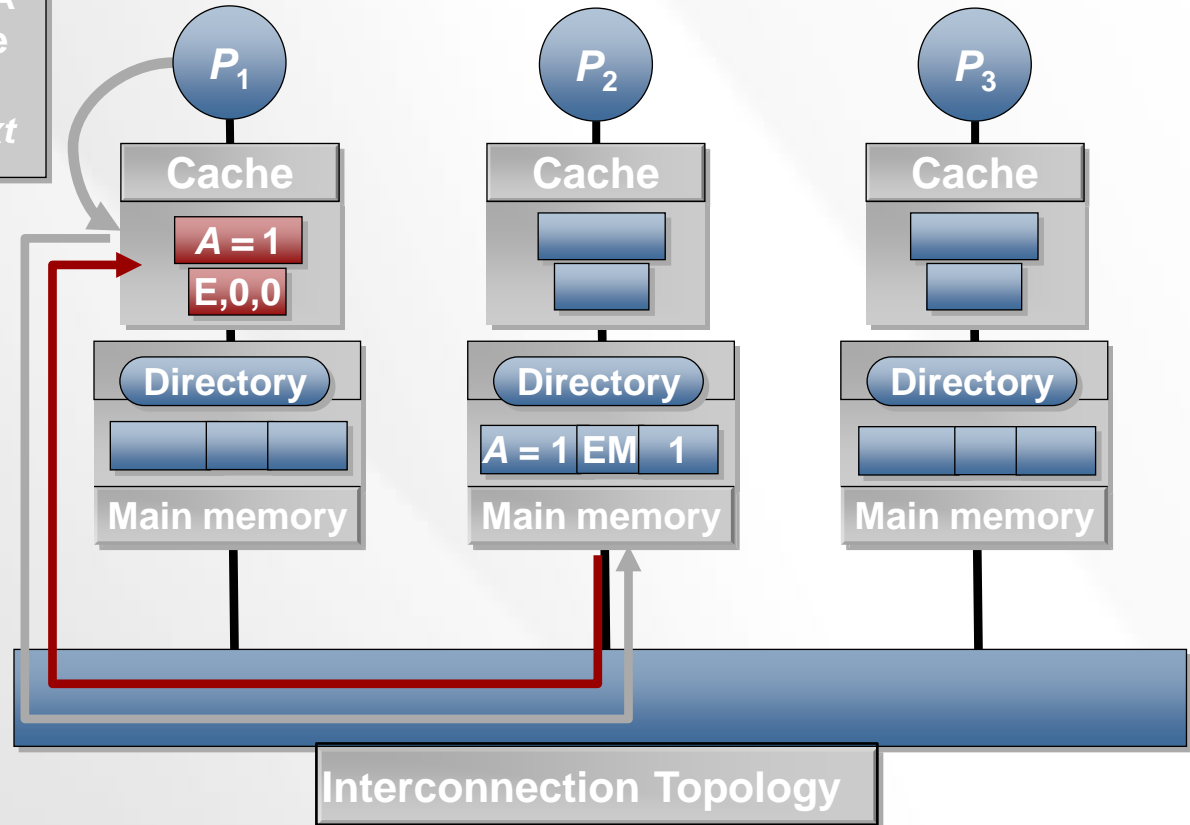
P_3 Read A

P_2 Read A

P_1 Rd & A

P_1 Read

Dir ReplyD



SSCI: Processor P_1 Reads A

Processor P_1 's read completes.

Trace

P_1 Read A

P_1 Write $A = 2$

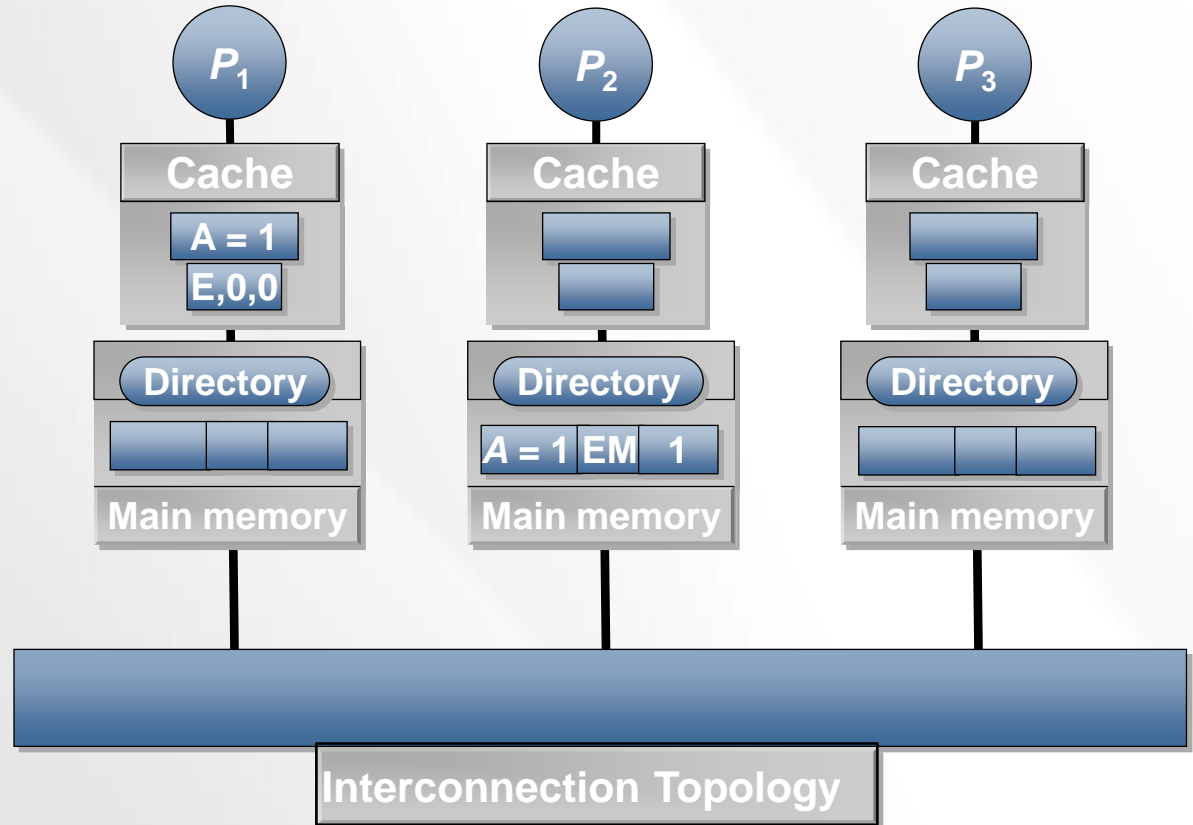
P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A



SSCI: Processor P_1 Writes A

Processor P_1 attempts to write $A=2$ in its cache. Block is modified and state is set to M.

Trace

P_1 Read A

P_1 Write $A = 2$

P_3 Read A

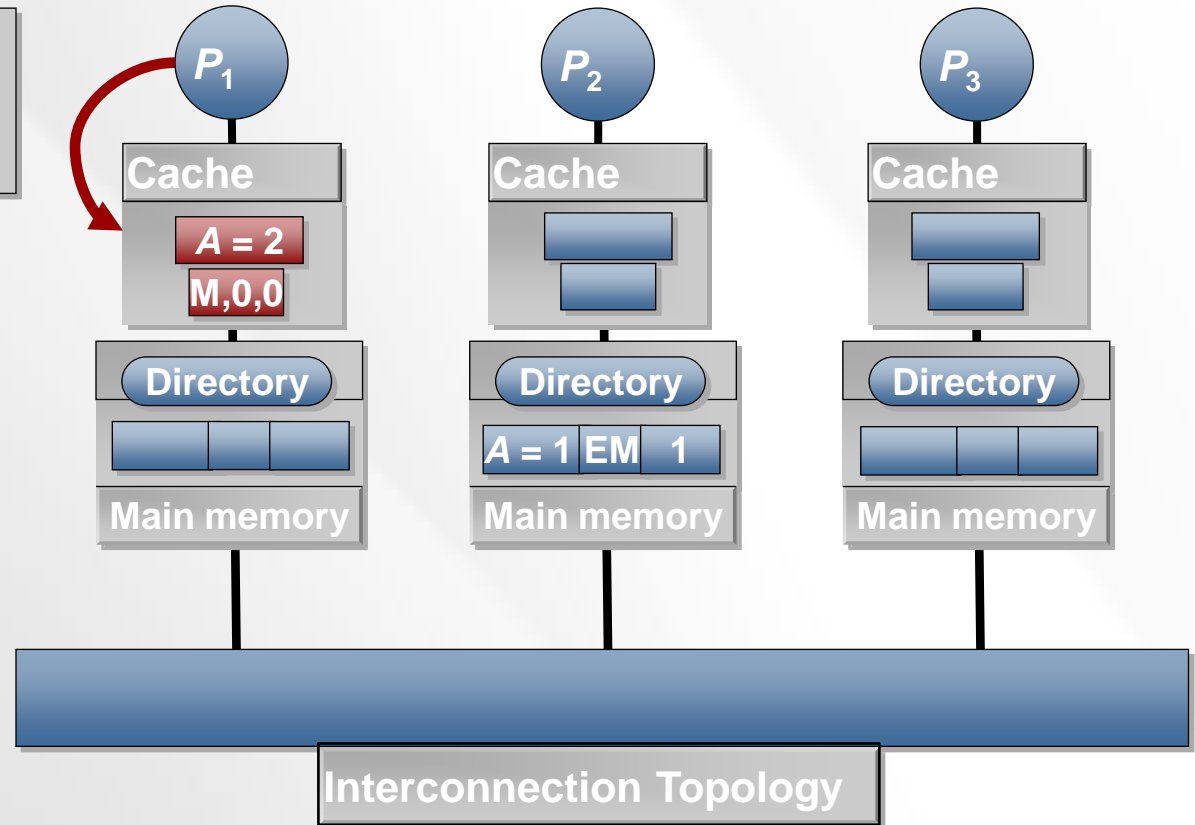
P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A

P_1 Wr $A, \#2$



SSCI: Processor P_1 Writes A

Processor P_1 completes writing $A=2$ to its cache.

Trace

P_1 Read A

P_1 Write $A = 2$

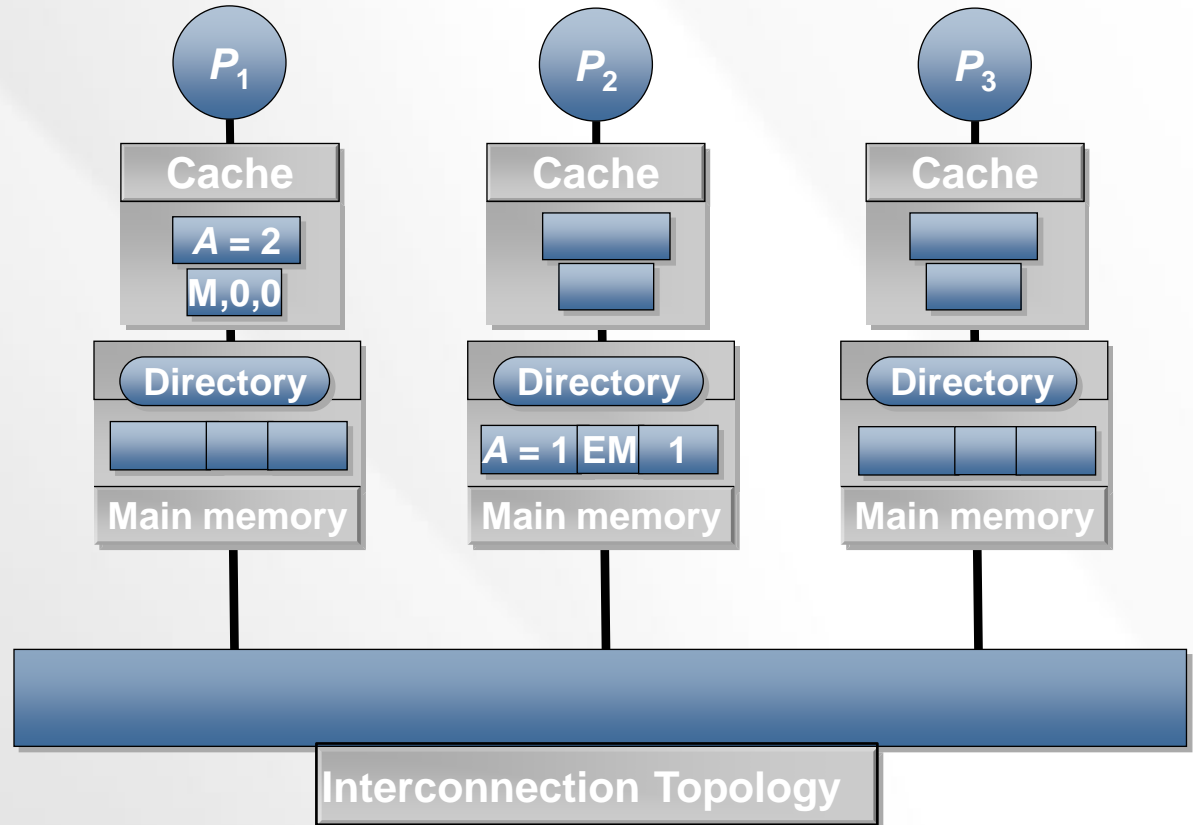
P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A



SSCI: Processor P_3 Reads A

Processor P_3 attempts to read A from its cache, misses.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

P_2 Read A

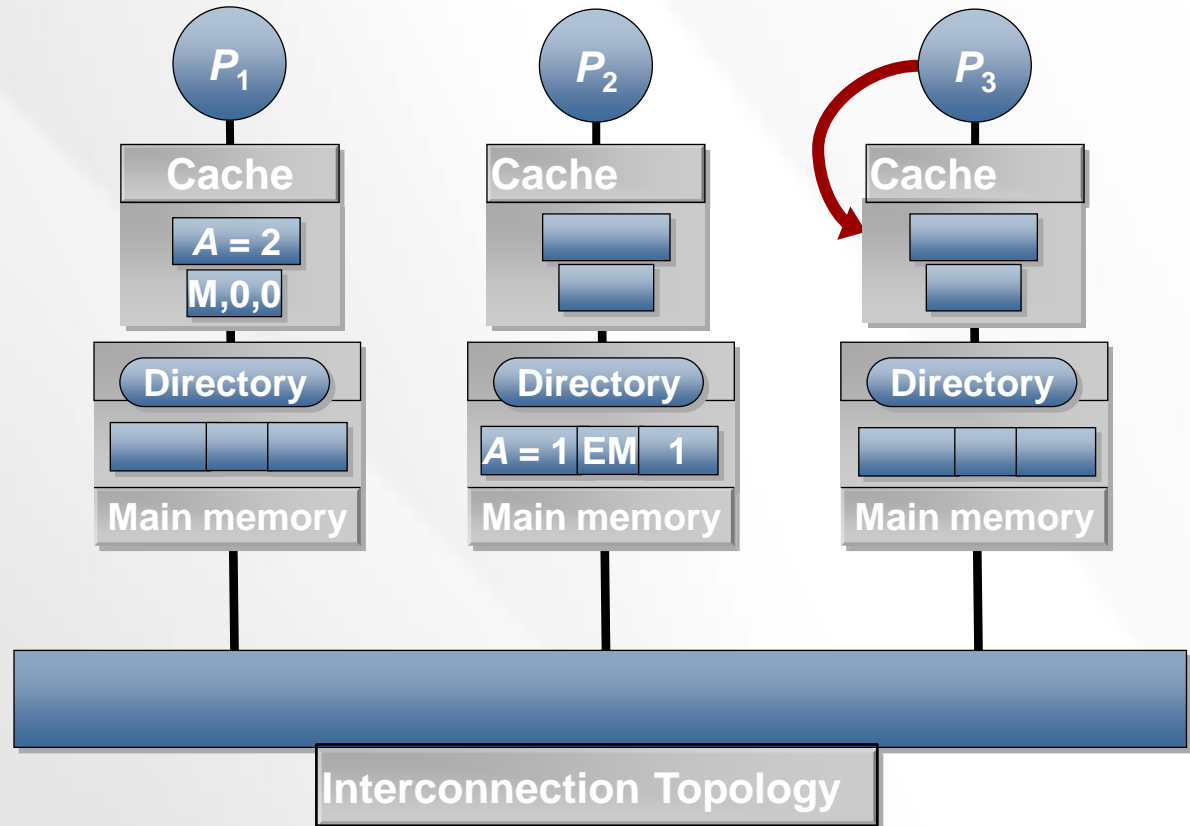
P_3 Rd &A

P_3 Read

Dir Reply

P_3 WB+INT+UpdPtr

P_1 Flush



SSCI: Processor P_3 Reads A

P_3 cache sends Read request to home directory.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

P_2 Read A

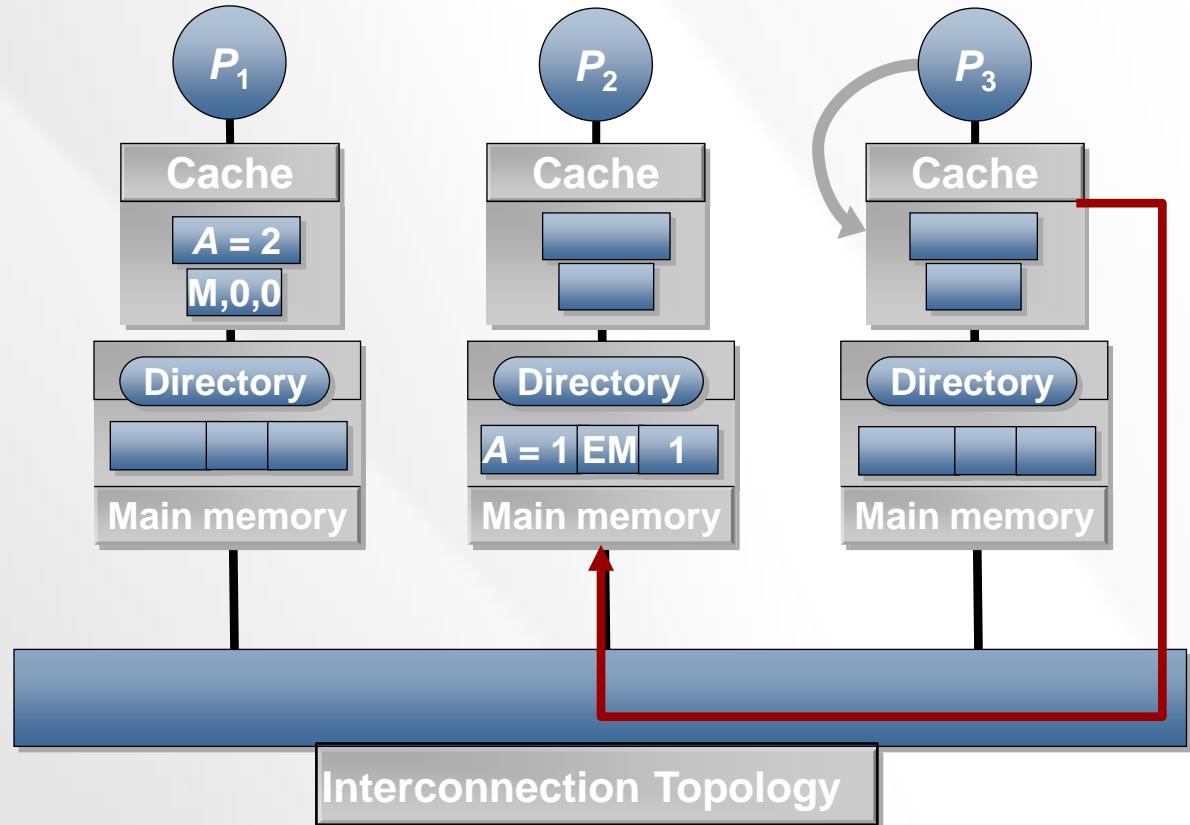
P_3 Rd &A

P_3 Read

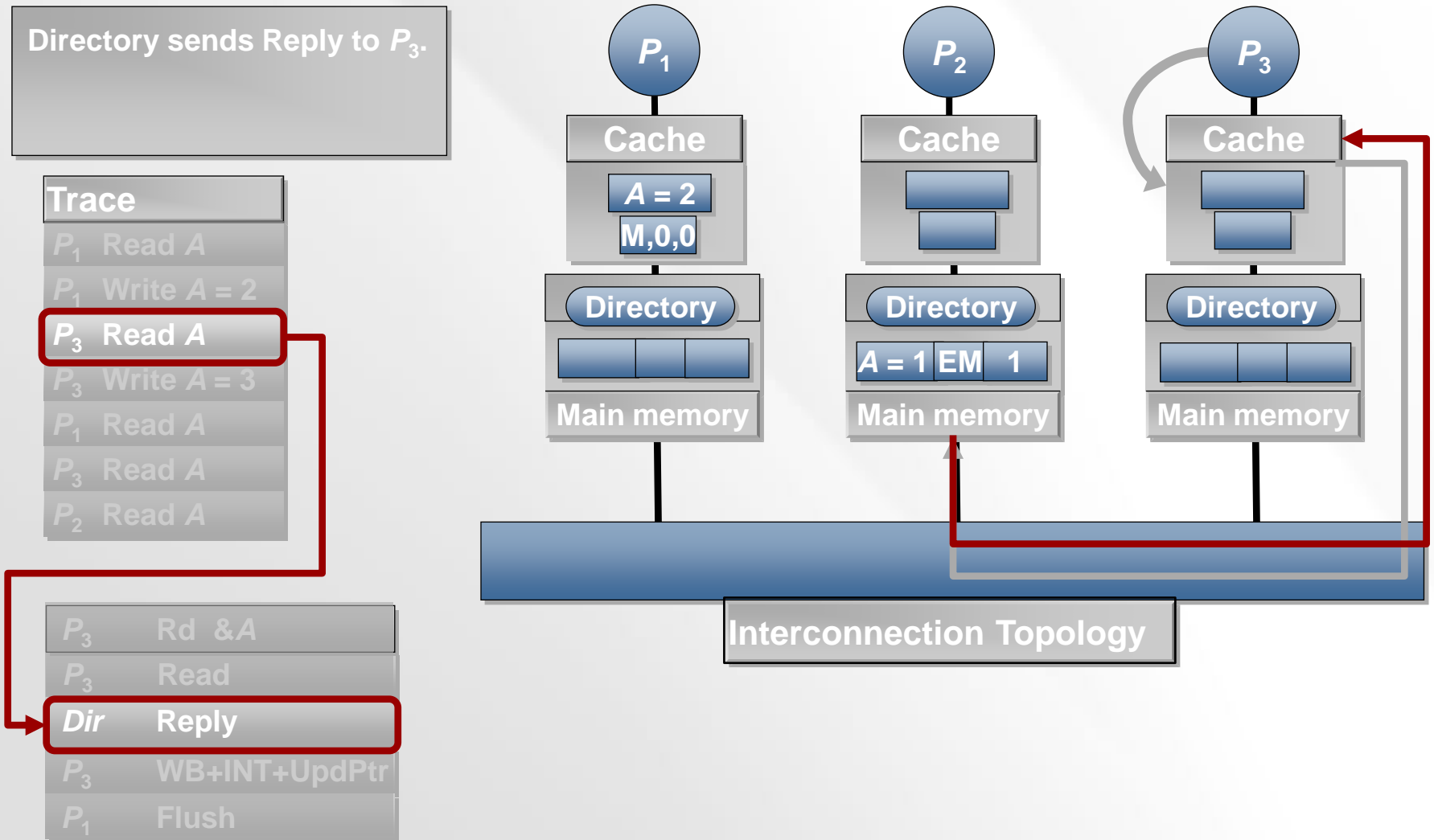
Dir Reply

P_3 WB+INT+UpdPtr

P_1 Flush



SSCI: Processor P_3 Reads A



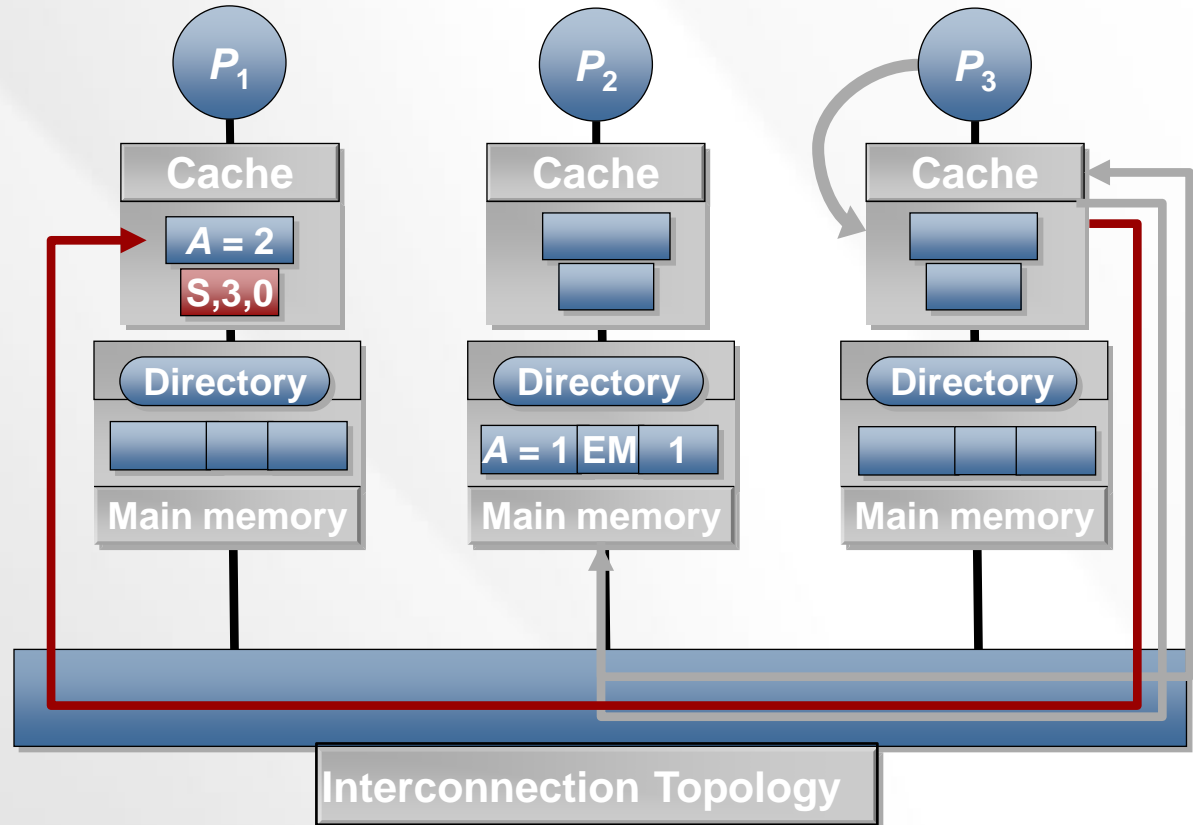
SSCI: Processor P_3 Reads A

P_3 sends owning cache a WB+Int+UpdPtr.
Owner P_1 changes state to S, with *prev* pointer to P_3

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_3 Rd &A
 P_3 Read
Dir Reply
 P_3 WB+Int+UpdPtr
 P_1 Flush



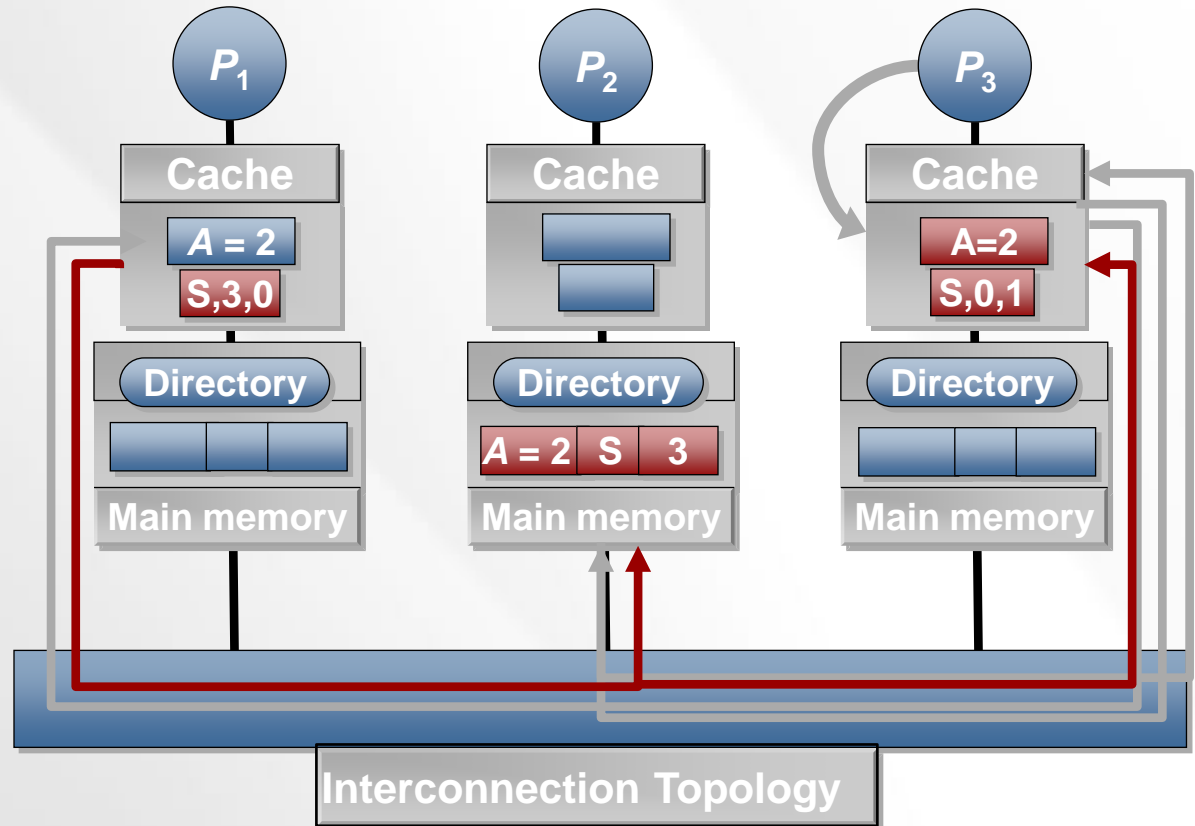
SSCI: Processor P_3 Reads A

P1 flushes the block.
Directory state changes to S,3 and P_3 's state changes to S with *next* pointing to P_1

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_3 Rd &A
 P_3 Read
Dir Reply
 P_3 WB+INT+UpdPtr
 P_1 Flush



SSCI: Processor P_3 Reads A

Processor P_3 completes read operation from its cache.

Trace

P_1 Read A

P_1 Write $A = 2$

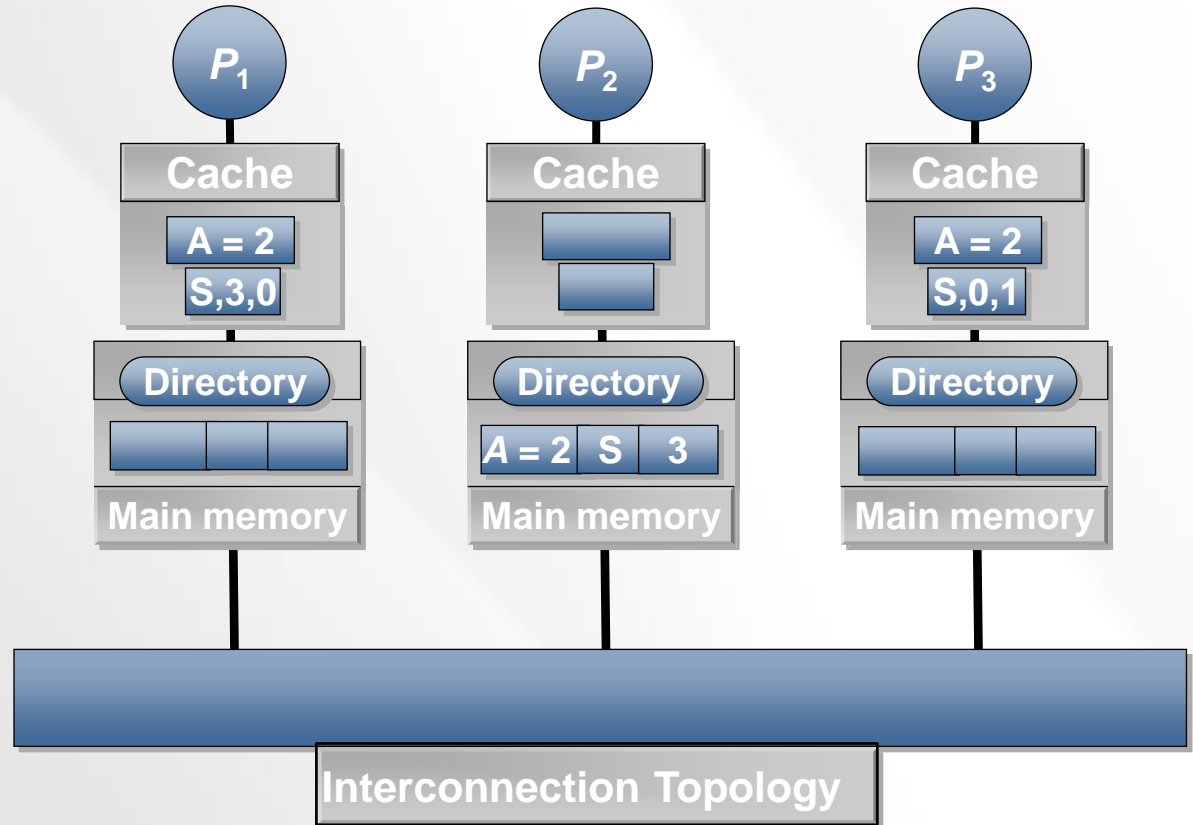
P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A



SSCI: Processor P_3 Writes A

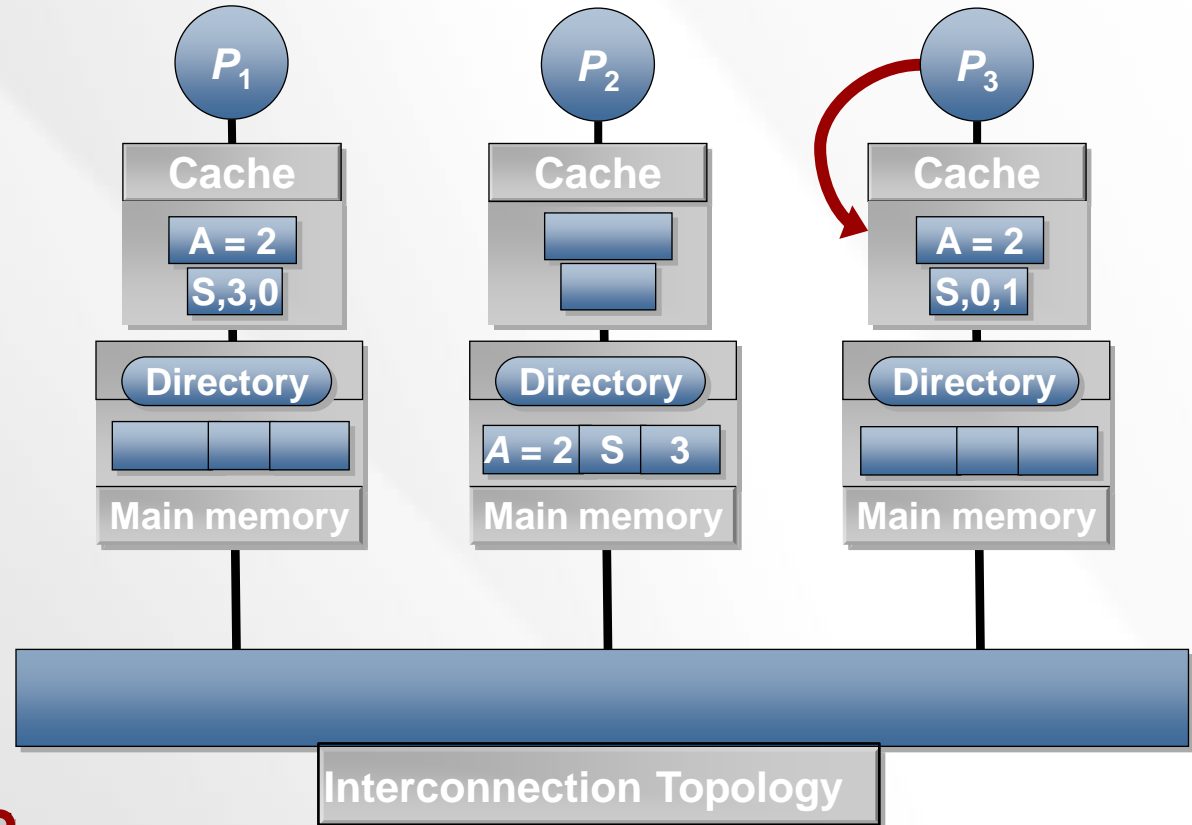
Processor P_3 attempts to write A, which is in its cache.

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_3 Wr &A

P_3 Upgr
 Dir Inv, Reply
 P_1 InvAck



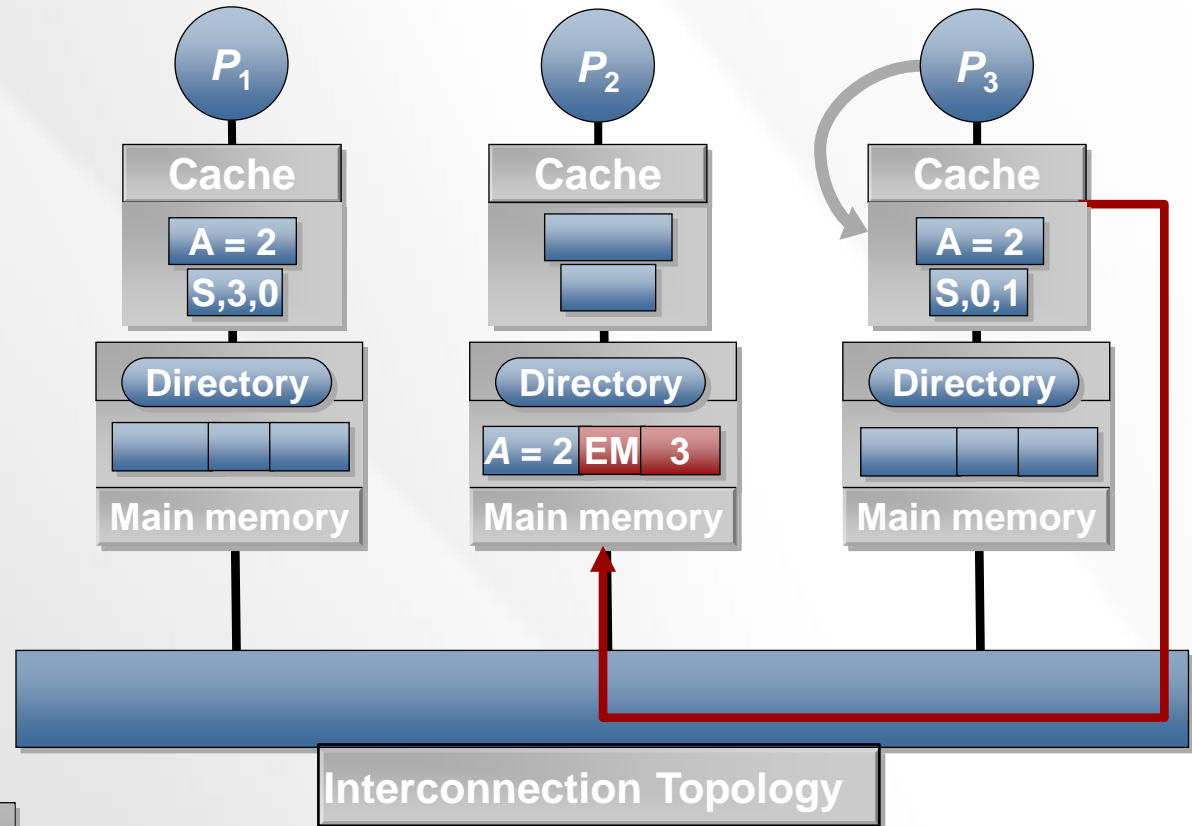
SSCI: Processor P_3 Writes A

Processor P_3 sends *Upgr* request to the directory.

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_3 Wr &A
 P_3 Upgr
 Dir Inv, Reply
 P1 InvAck



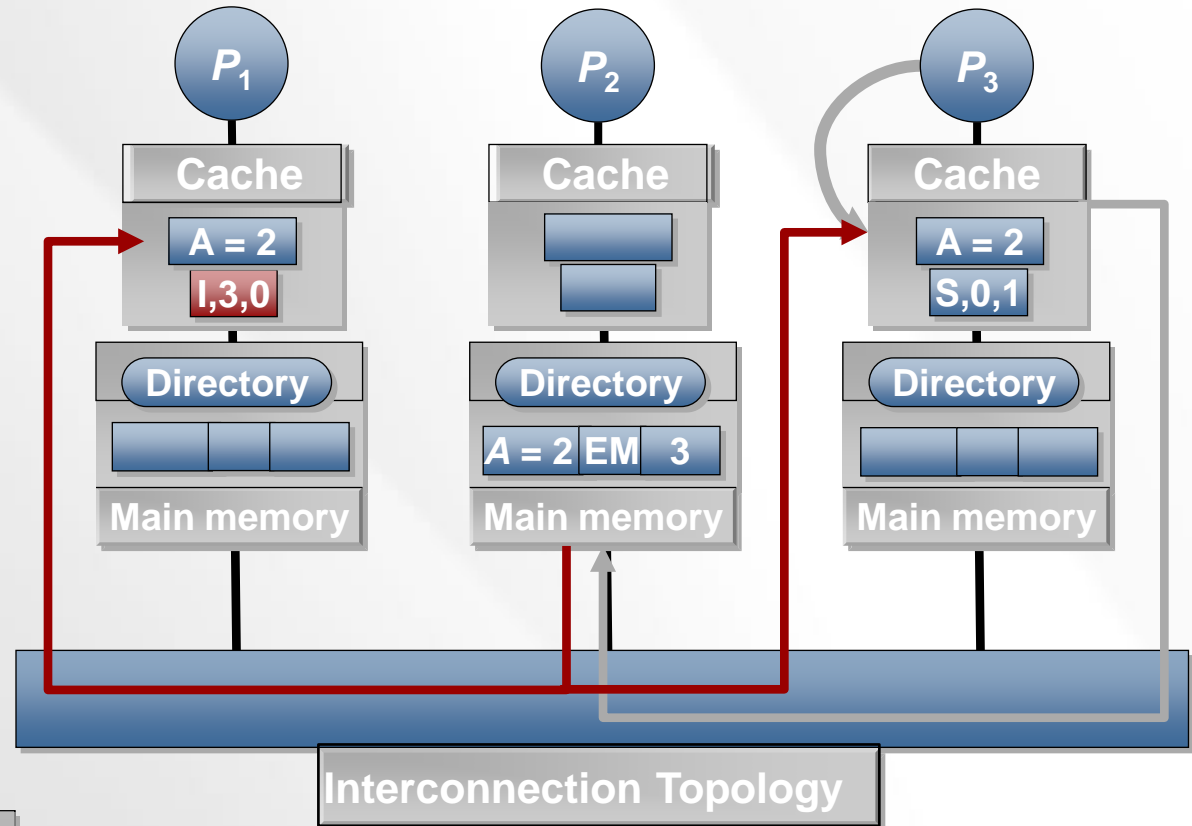
SSCI: Processor P_3 Writes A

Directory sends *Inv* to P_1
and *Reply* to P_3 .
 P_1 invalidates block.

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_3 Wr &A
 P_3 Upgr
Dir Inv, Reply
 P_1 InvAck

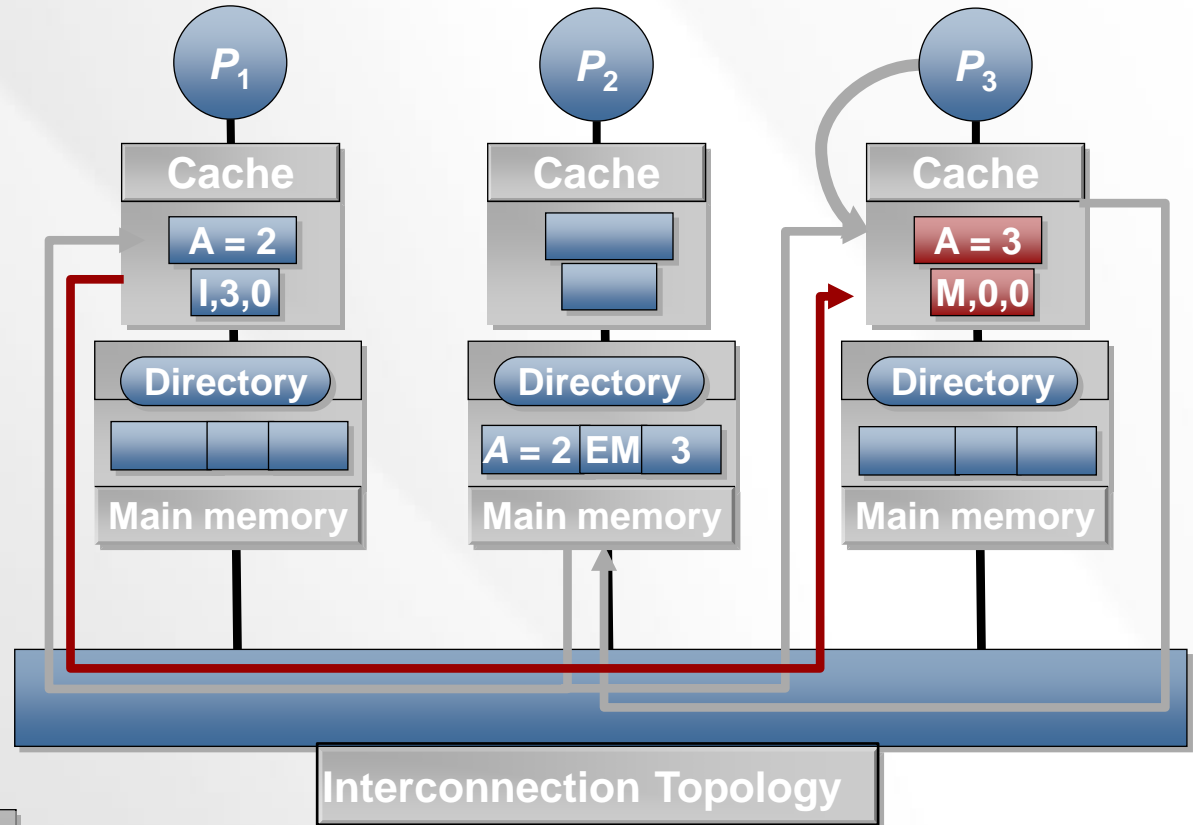


SSCI: Processor P_3 Writes A

Processor P_1 sends *InvAck* and P_3 proceeds with the write, becomes owner.

Trace

P_1	Read A
P_1	Write A = 2
P_3	Read A
P_3	Write A = 3
P_1	Read A
P_3	Read A
P_2	Read A



P_3	Wr &A
P_3	Upgr
Dir	Inv, Reply
P_1	InvAck

SSCI: Processor P_3 Writes A

Processor P_3 completes writing $A=3$ to its cache.

Trace

P_1 Read A

P_1 Write A = 2

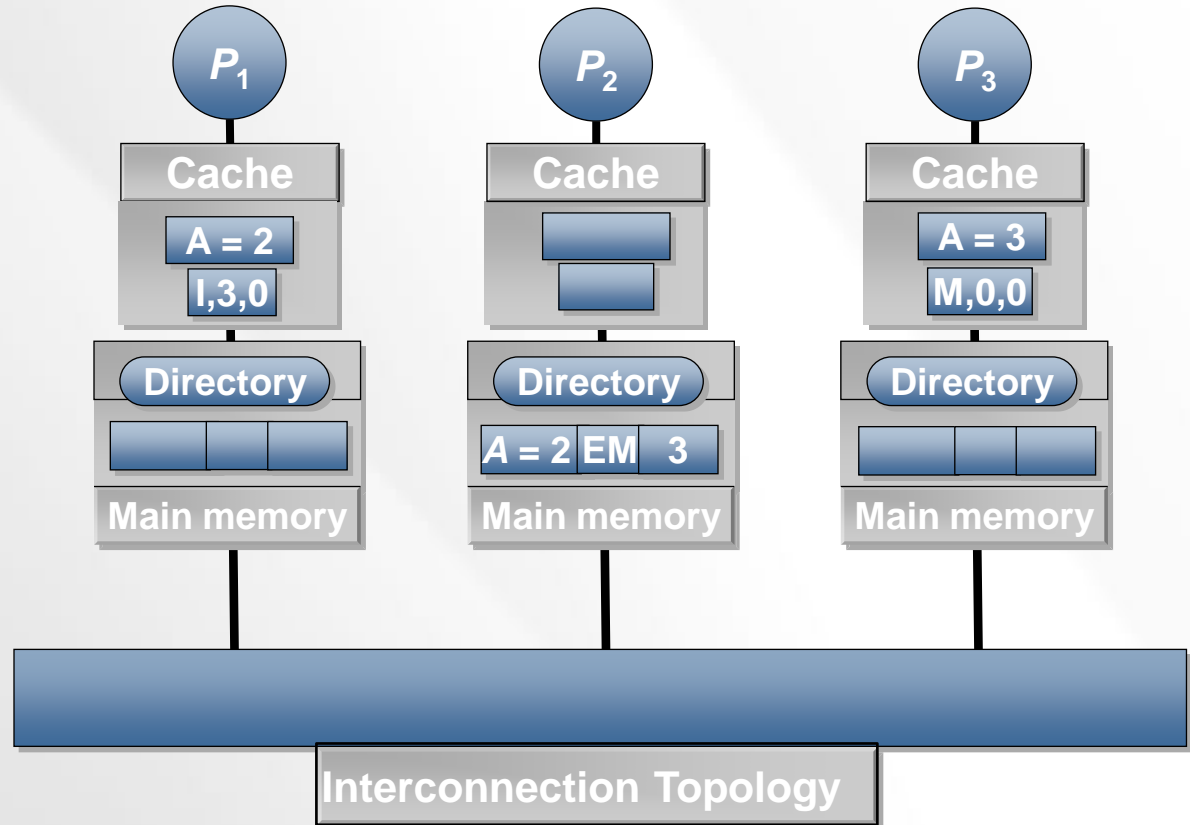
P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

P_2 Read A



SSCI: Processor P_1 Reads A

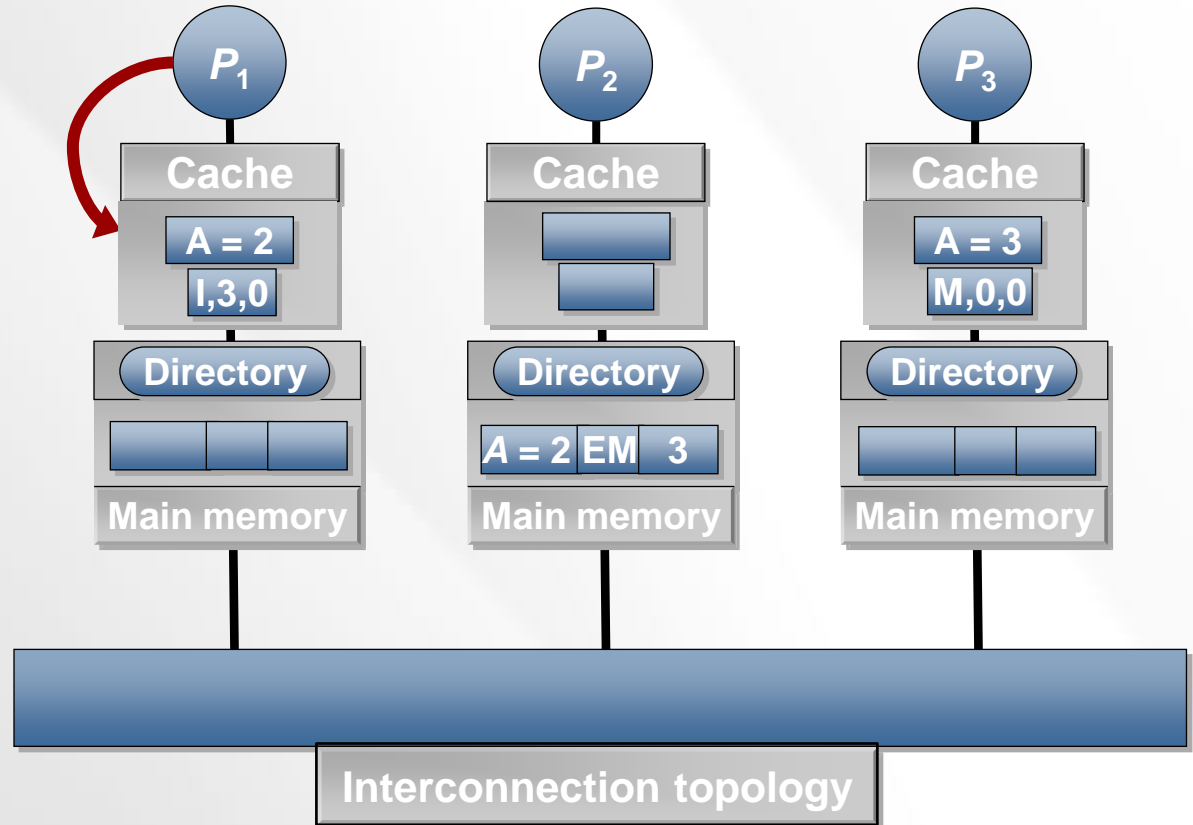
Processor P_1 attempts to read A from its cache, misses.

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_1 Rd &A

P_1 Read
 Dir Reply
 P_1 WB+INT+UpdPtr
 P_3 Flush



SSCI: Processor P_1 Reads A

P_1 cache sends *Read* request to home directory.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

P_2 Read A

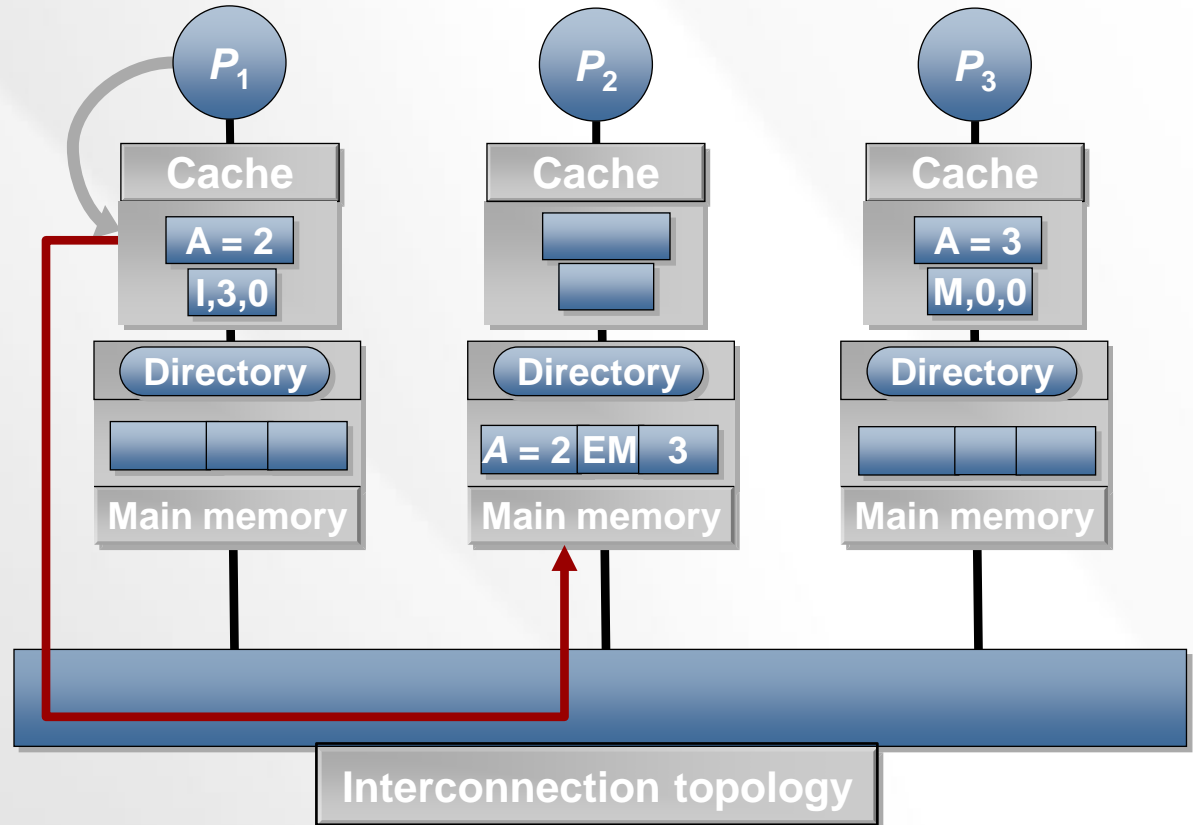
P_1 Rd &A

P_1 Read

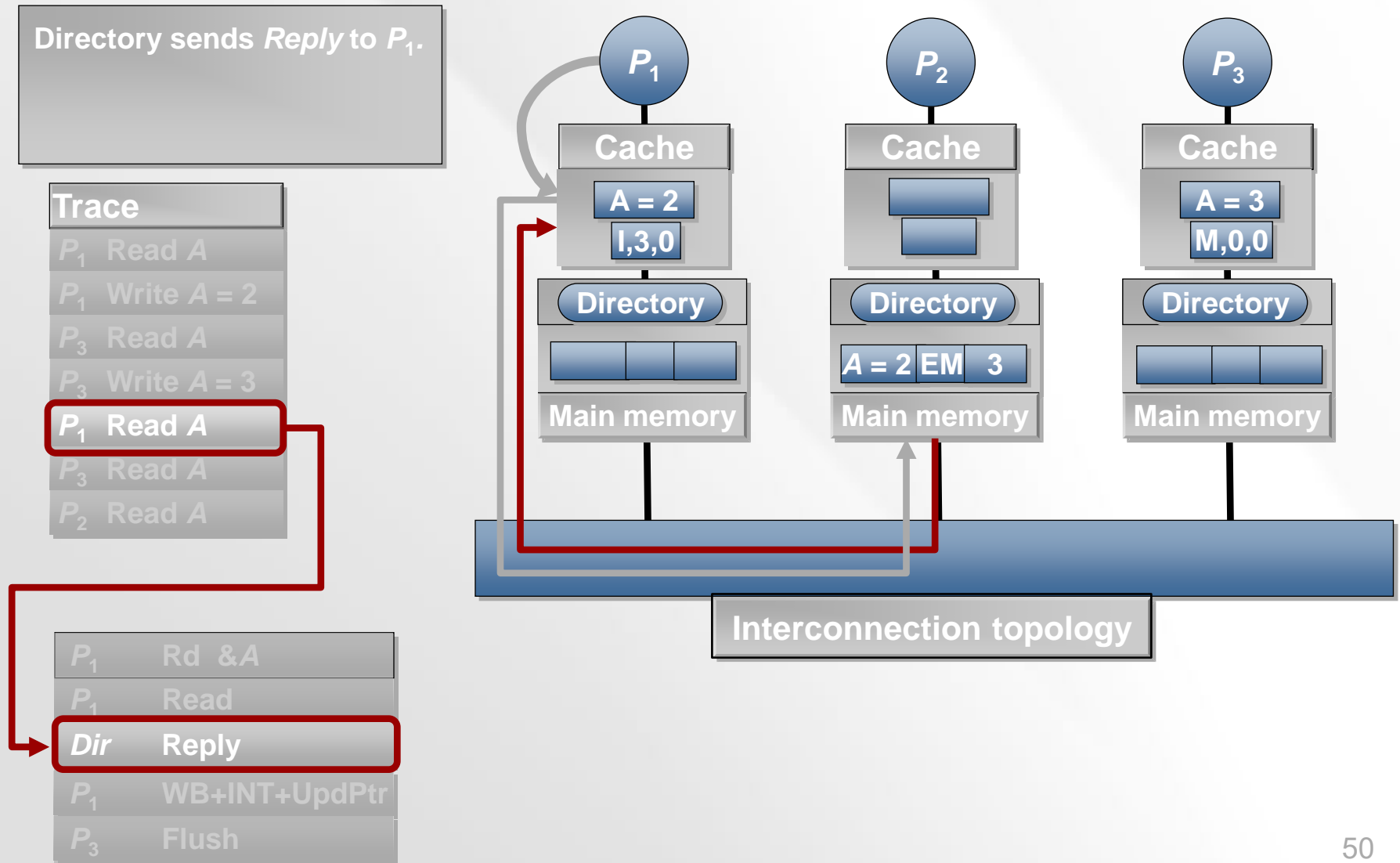
Dir Reply

P_1 WB+INT+UpdPtr

P_3 Flush



SSCI: Processor P_1 Reads A



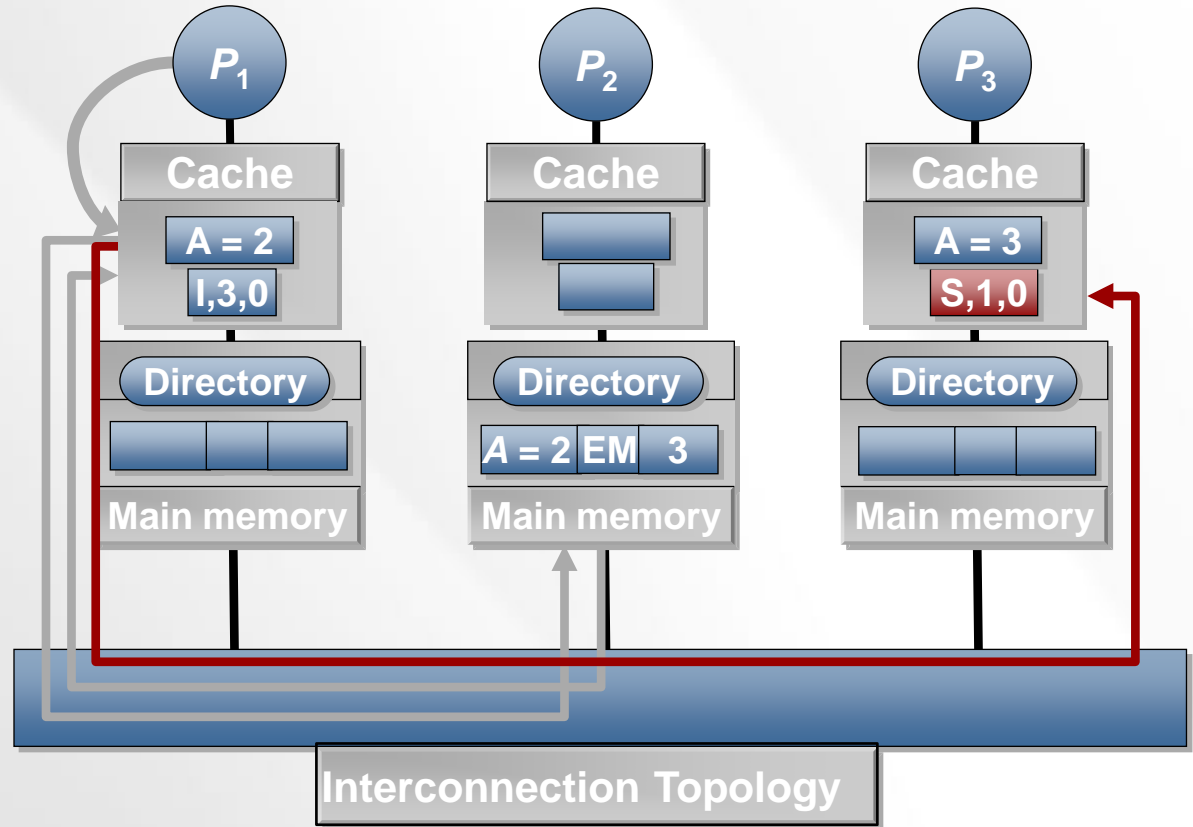
SSCI: Processor P_1 Reads A

P_1 sends *WB+Int+UpdPtr* to the owner cache.
The owner is downgraded to state S with *prev* set to P_1

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_1 Rd &A
 P_1 Read
Dir Reply
 P_1 WB+INT+UpdPtr
 P_3 Flush



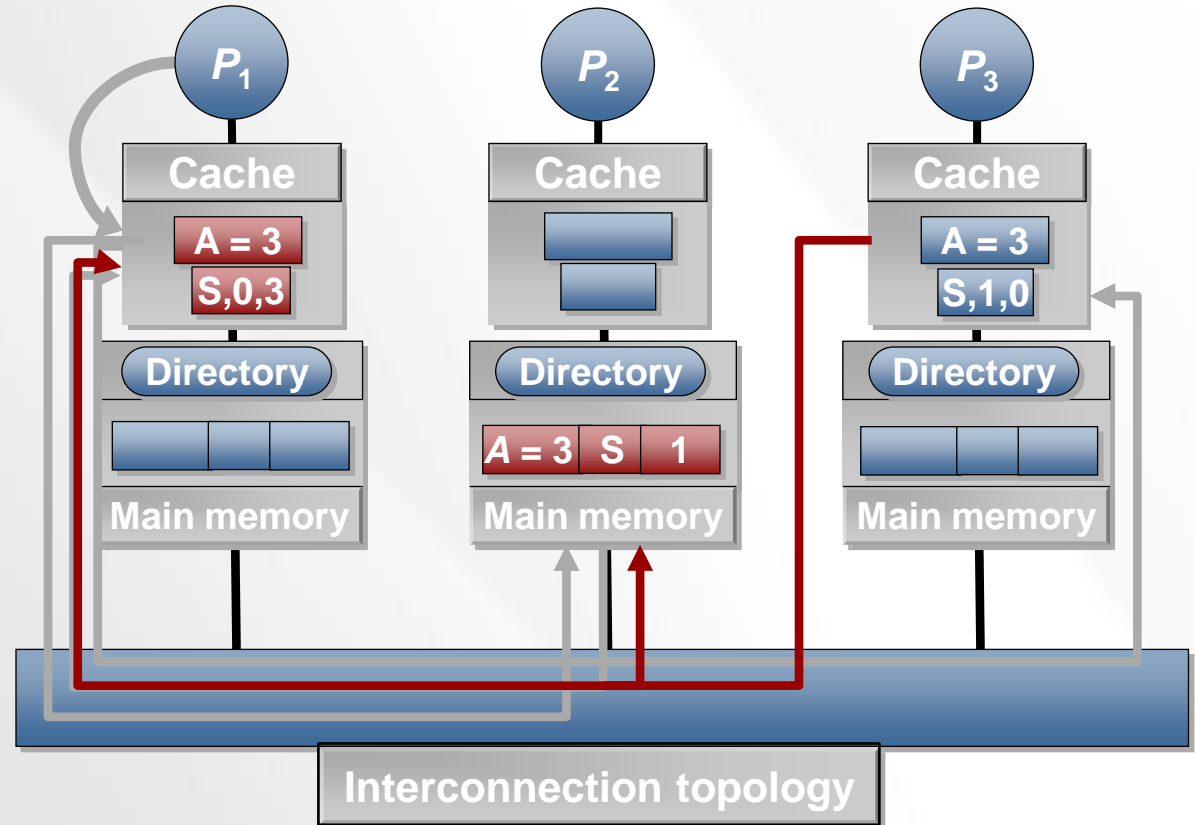
SSCI: Processor P_1 Reads A

Processor P_3 flushes the block to the directory and P_1 's cache. State is set to S with *next* pointer to P_3

Trace

P_1	Read A
P_1	Write $A = 2$
P_3	Read A
P_3	Write $A = 3$
P_1	Read A
P_3	Read A
P_2	Read A

P_1	Rd &A
P_1	Read
Dir	Reply
P_1	WB+INT+UpdPtr
P_3	Flush



SSCI: Processor P_1 Reads A

Processor P_1 completes reading A from its cache.

Trace

P_1 Read A

P_1 Write $A = 2$

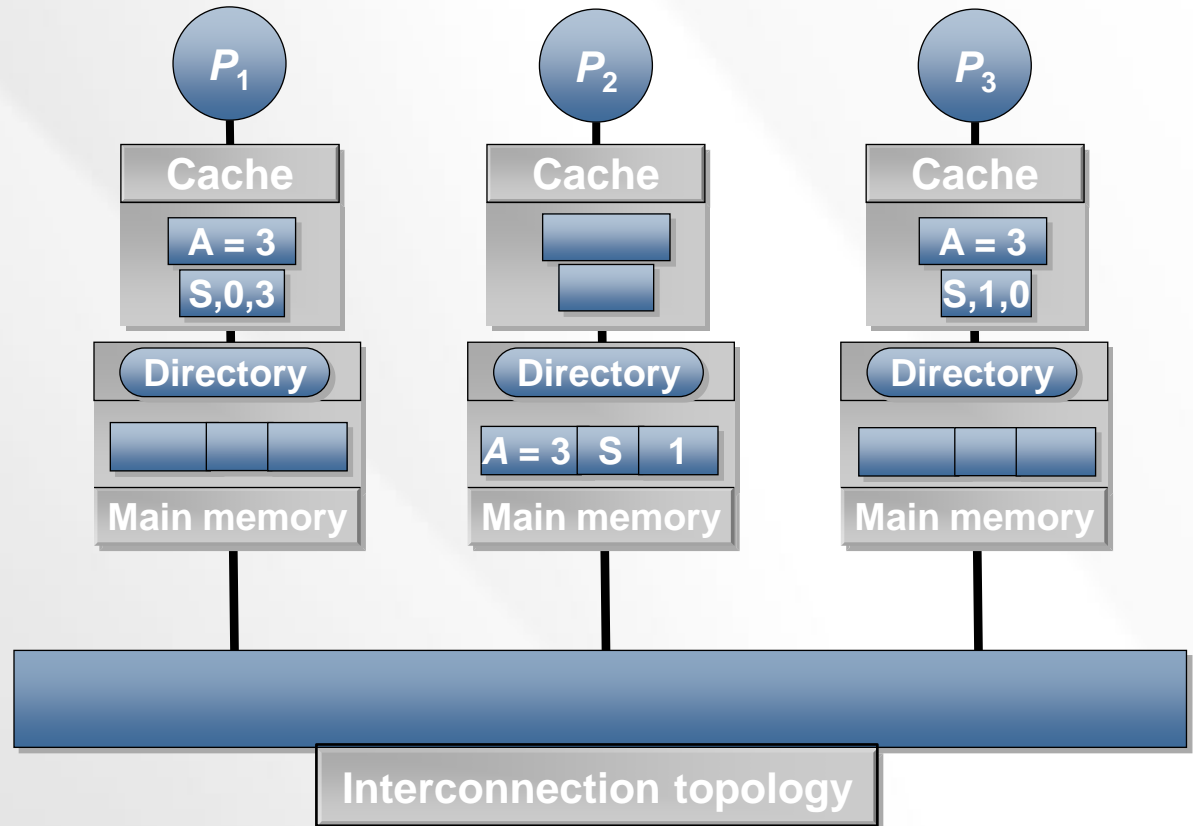
P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A



SSCI: Processor P_3 Reads A

Processor P_3 attempts to read A from its cache, hits.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

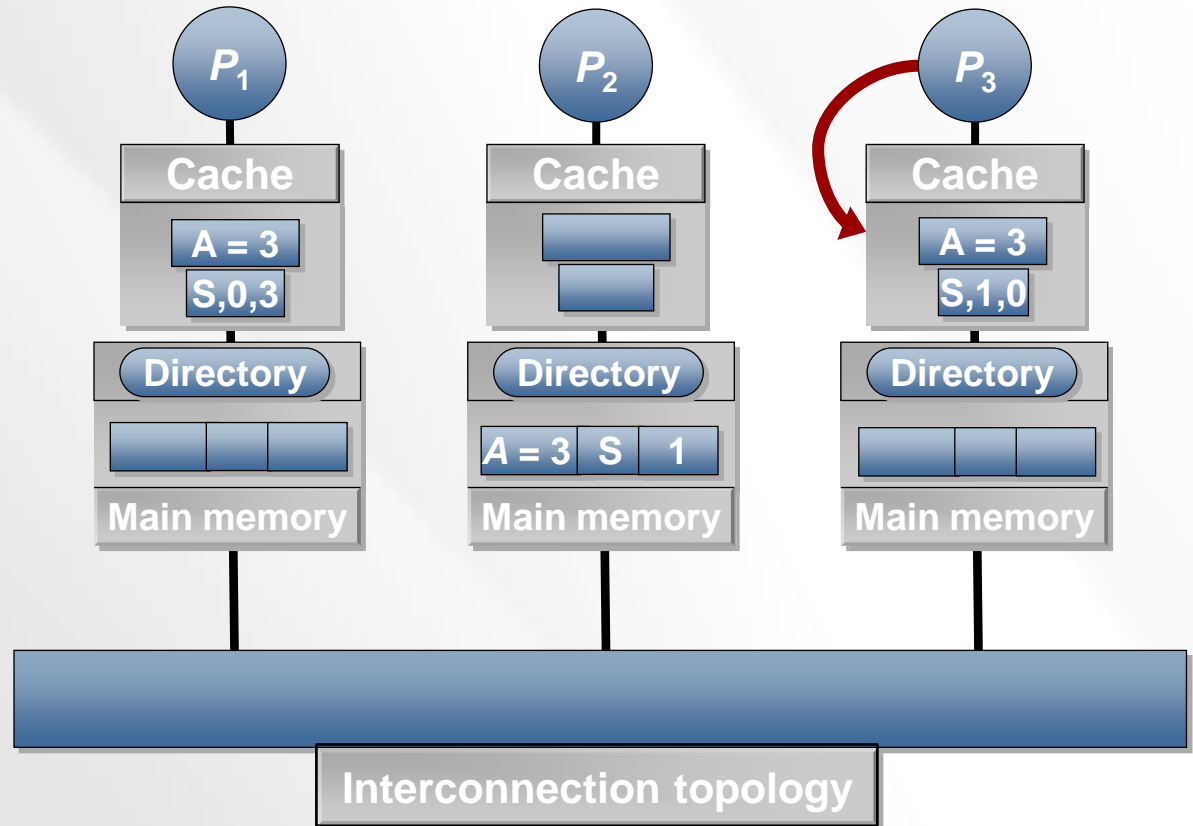
P_1 Read A

P_3 Read A

P_2 Read A

P_3 Rd &A

P_3 returns data



SSCI: Processor P_3 Reads A

P_3 's cache returns the value of A immediately.

Trace

P_1 Read A

P_1 Write $A = 2$

P_3 Read A

P_3 Write $A = 3$

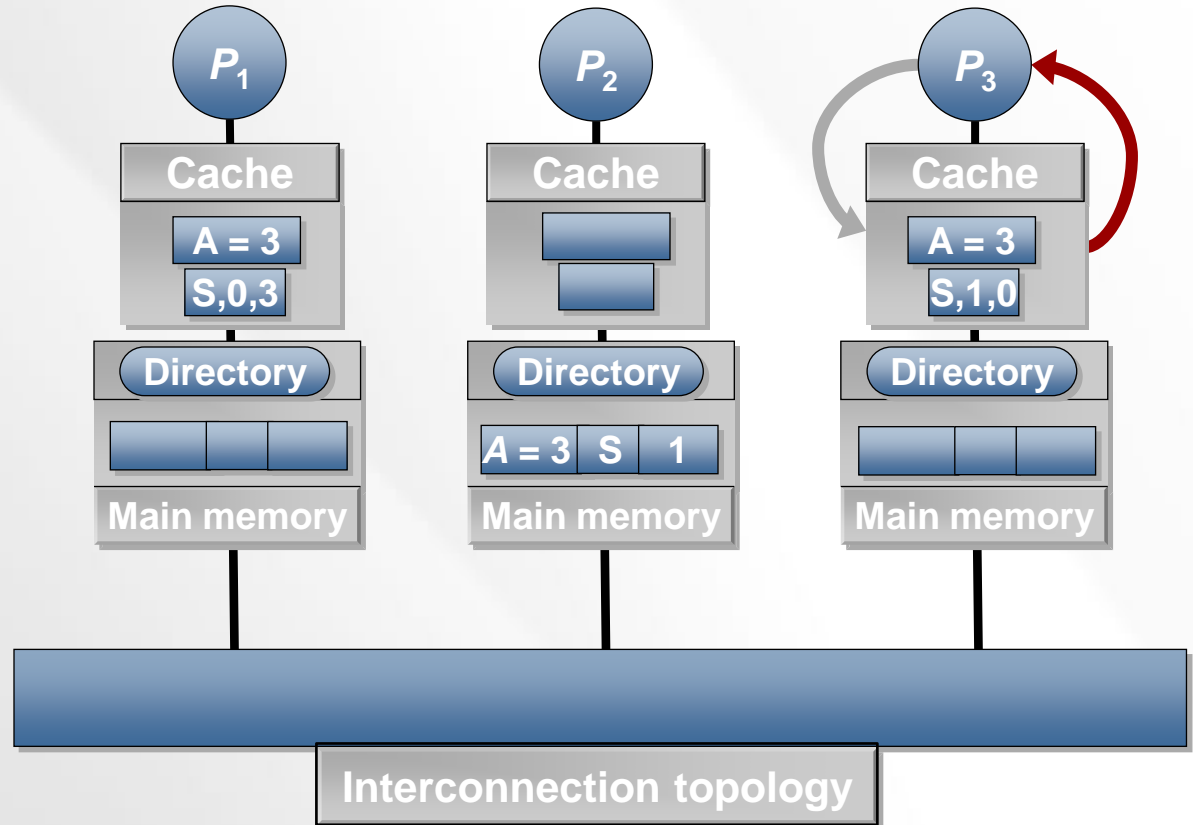
P_1 Read A

P_3 Read A

P_2 Read A

P_3 Rd & A

P_3 returns data



SSCI: Processor P_3 Reads A

Processor P_1 completes reading A from its cache.

Trace

P_1 Read A

P_1 Write $A = 2$

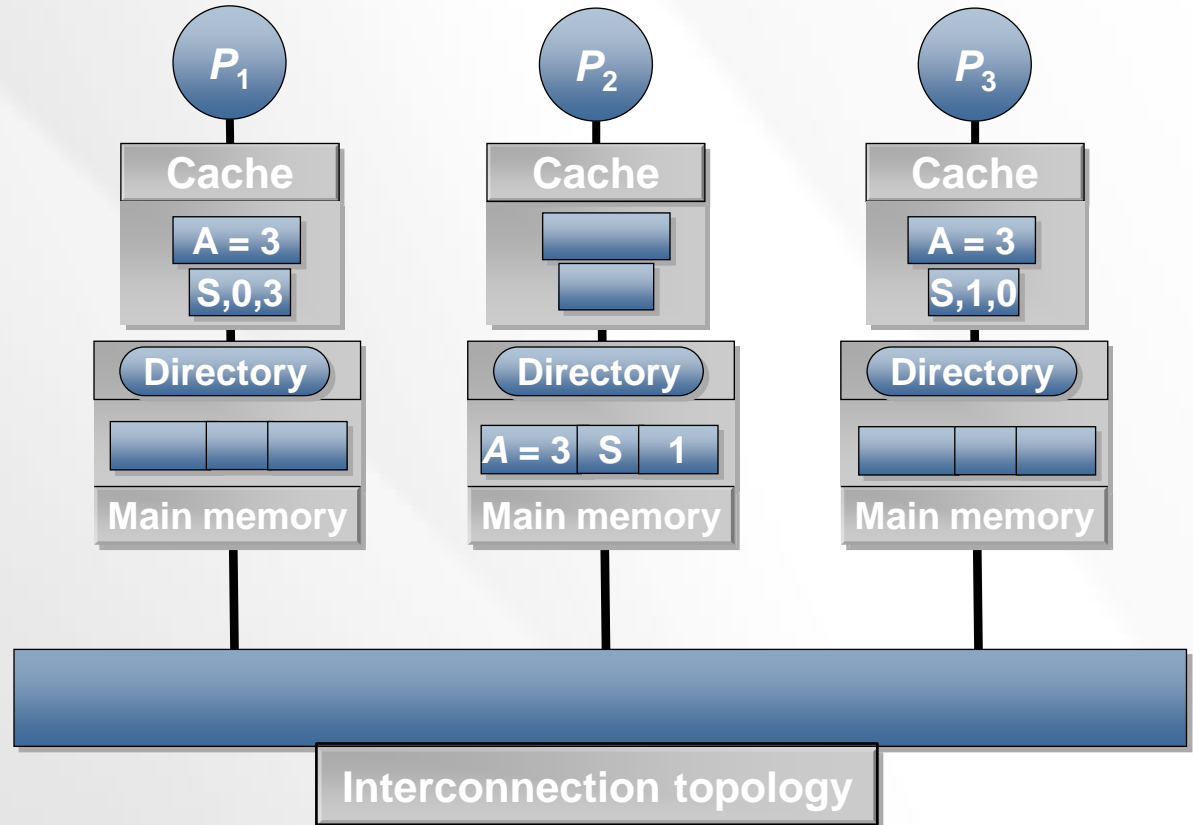
P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A



SSCI: Processor P_2 Reads A

Processor P_2 attempts to read A from its cache.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

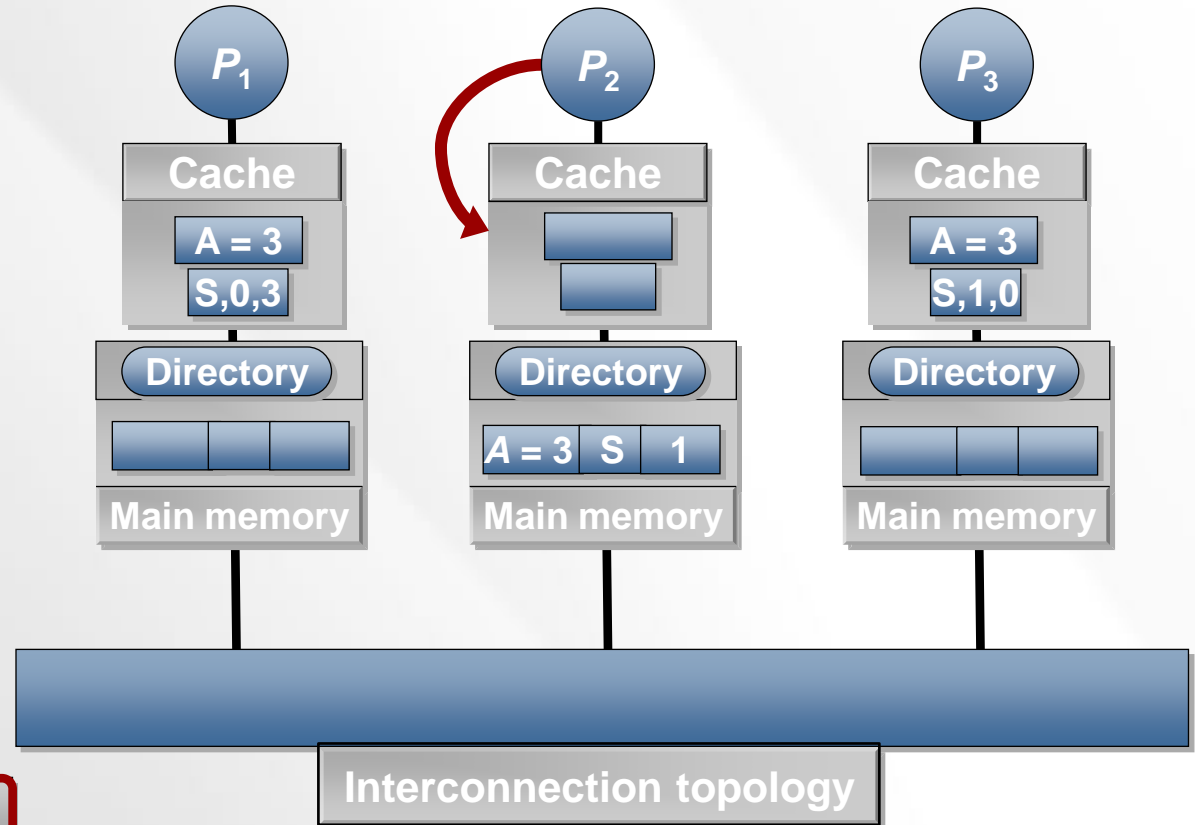
P_2 Read A

P_2 Rd &A

P_2 Read

Dir ReplyD/ID

P_2 UpdPtr



SSCI: Processor P_2 Reads A

P_2 's cache sends a *Read* request to its directory. Head in directory is updated.

Trace

P_1 Read A

P_1 Write A = 2

P_3 Read A

P_3 Write A = 3

P_1 Read A

P_3 Read A

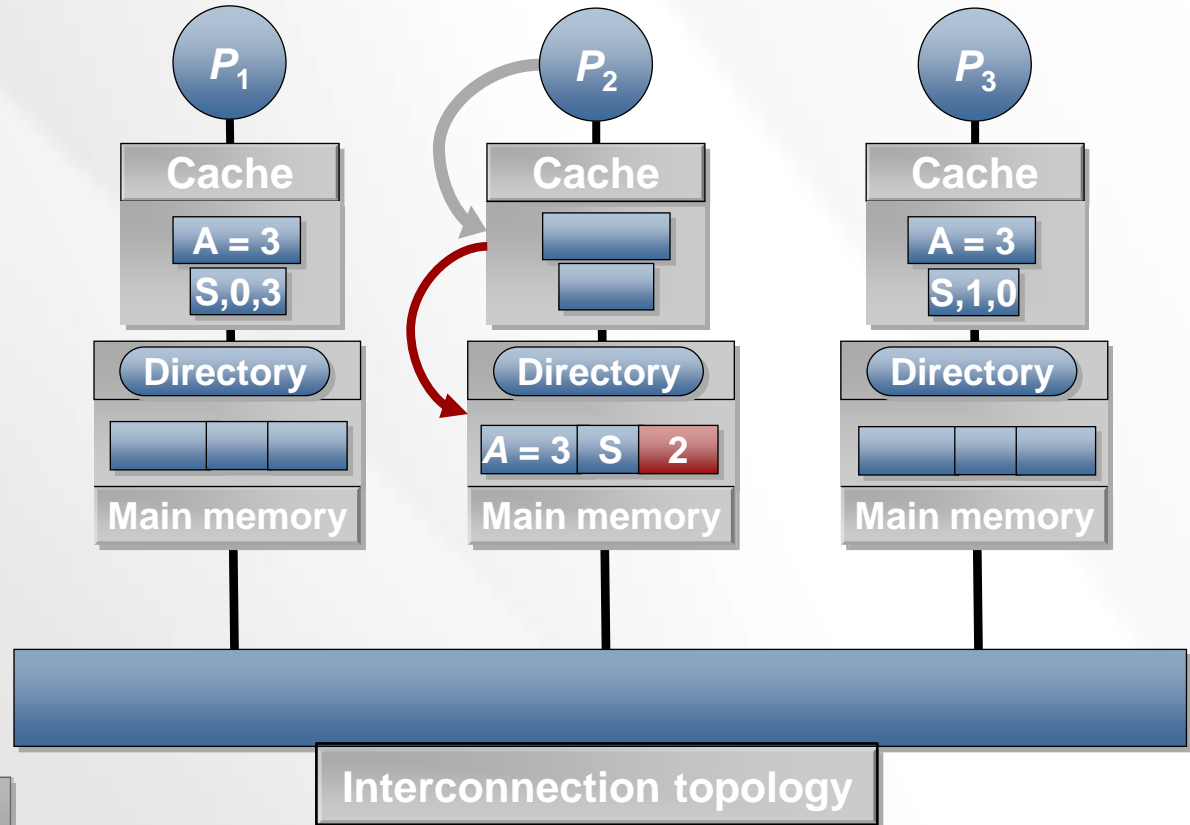
P_2 Read A

P_2 Rd &A

P_2 Read

Dir ReplyD/ID

P_2 UpdPtr



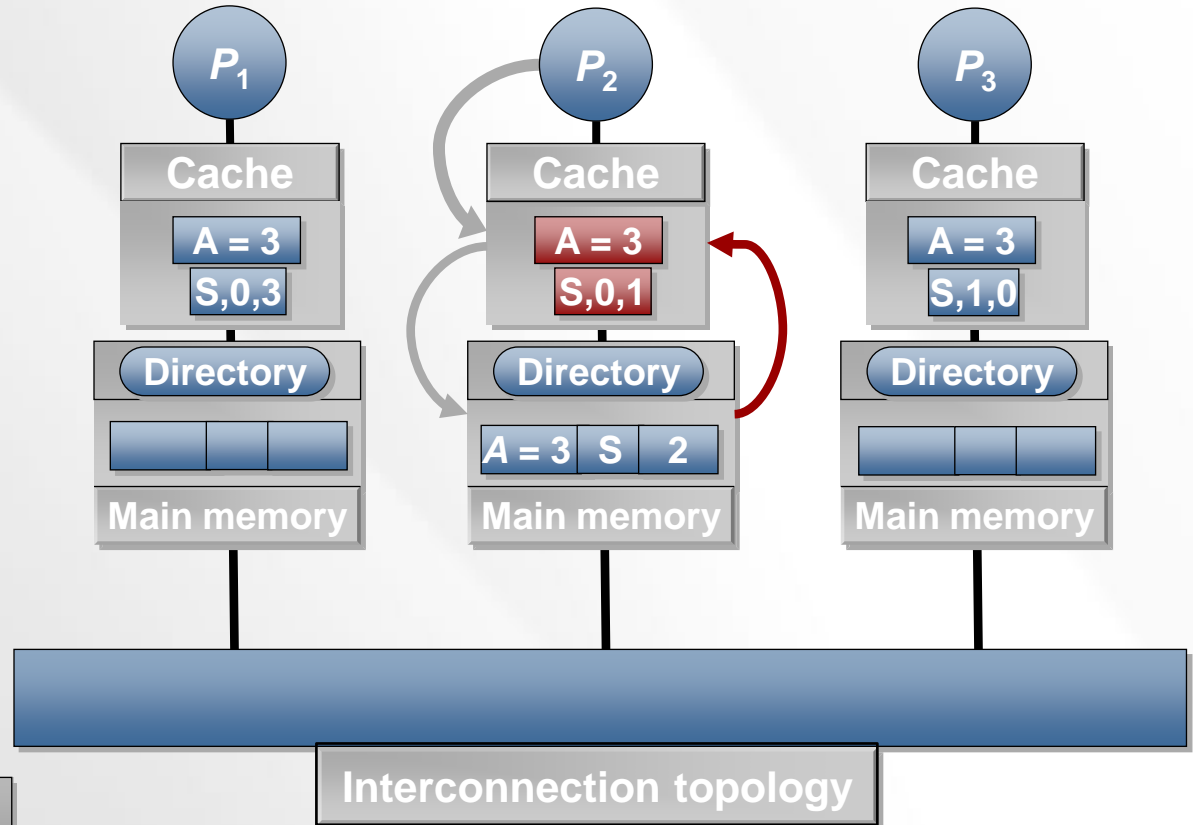
SSCI: Processor P_2 Reads A

Directory returns the block and state with ReplyD/ID. P_2 's state becomes S with *next* pointing to P_1 .

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_2 Rd &A
 P_2 Read
Dir ReplyD/ID
 P_2 UpdPtr



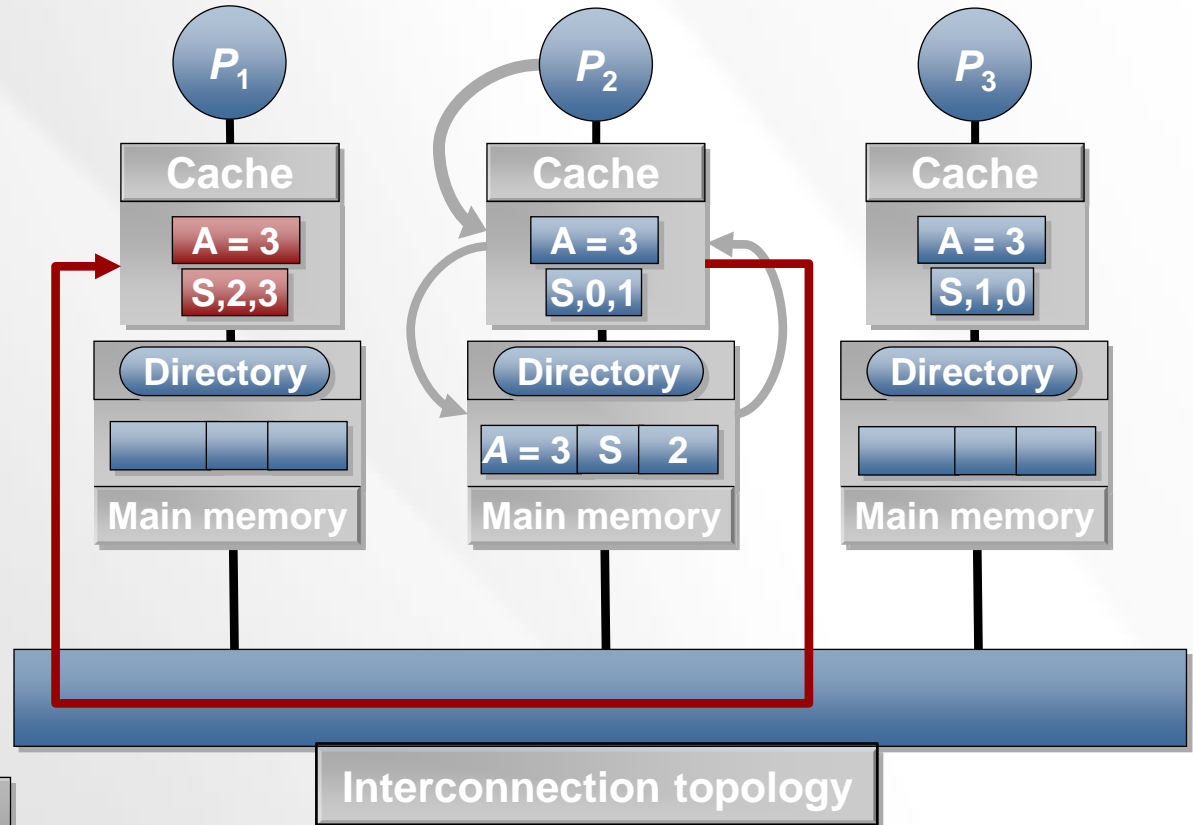
SSCI: Processor P_2 Reads A

P_2 sends UpdPtr to P_1 .
 P_1 changes Next pointer to point to P_2 .

Trace

P_1 Read A
 P_1 Write A = 2
 P_3 Read A
 P_3 Write A = 3
 P_1 Read A
 P_3 Read A
 P_2 Read A

P_2 Rd &A
 P_2 Read
Dir ReplyD/ID
 P_2 UpdPtr



SSCI: Processor P_2 Reads A

Processor P_2 finishes *Read* operation.

Trace

P_1 Read A

P_1 Write $A = 2$

P_3 Read A

P_3 Write $A = 3$

P_1 Read A

P_3 Read A

P_2 Read A

