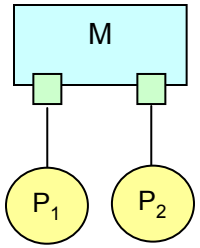
Chapter 4

Shared Memory Architecture

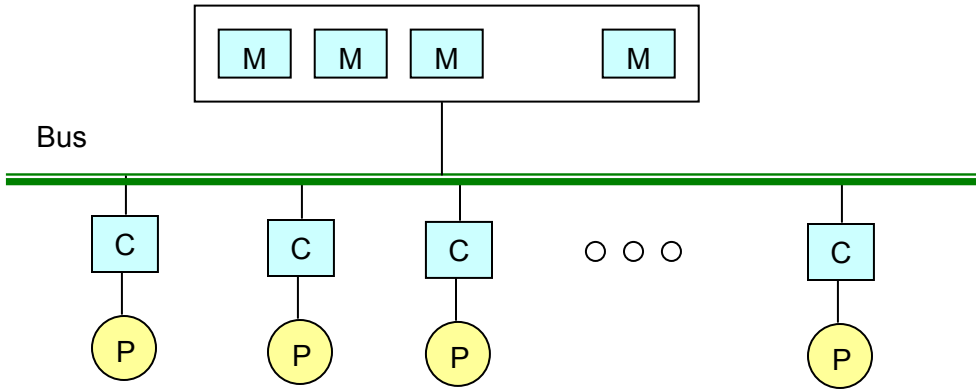
4.1 Classification of Shared Memory Systems

- Shared memory systems are multi-port and categorized as follows:
  - Uniform memory Access (UMA)
  - Non-uniform Memory Access (NUMA)
  - Cache Only Memory Architecture (COMA)



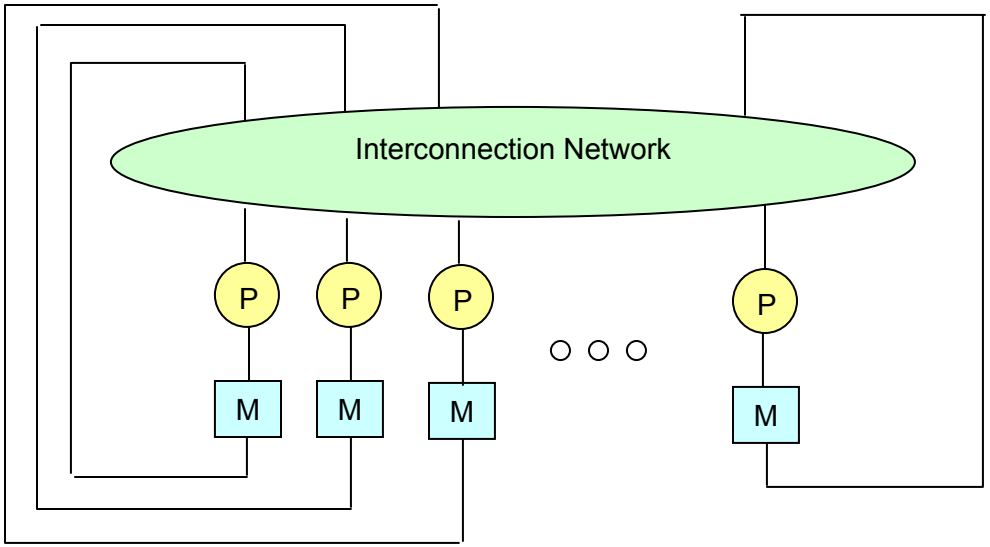
4.1 Classification of Shared Memory Systems

- Uniform Memory Access (UMA)
  - Shared memory is accessible by all processors through an interconnection network in the same way a single processor accesses its memory.
  - All processors have equal access time to any memory location.
  - Because access to the shared memory is balanced, these systems are called SMP (symmetric multiprocessor)



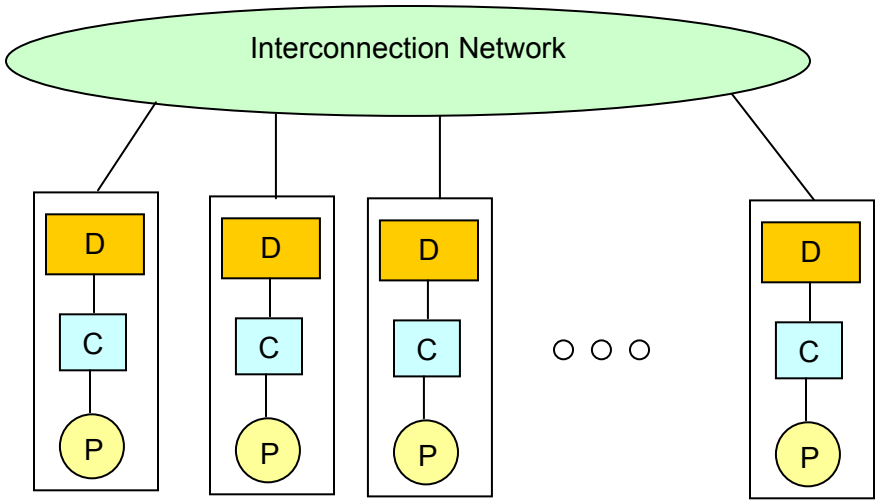
4.1 Classification of Shared Memory Systems

- Non Uniform Memory Access (NUMA)
  - Each processor has part of the shared memory attached.
  - The access time to modules depend on the distance to the processor, which results a non-uniform memory access time.



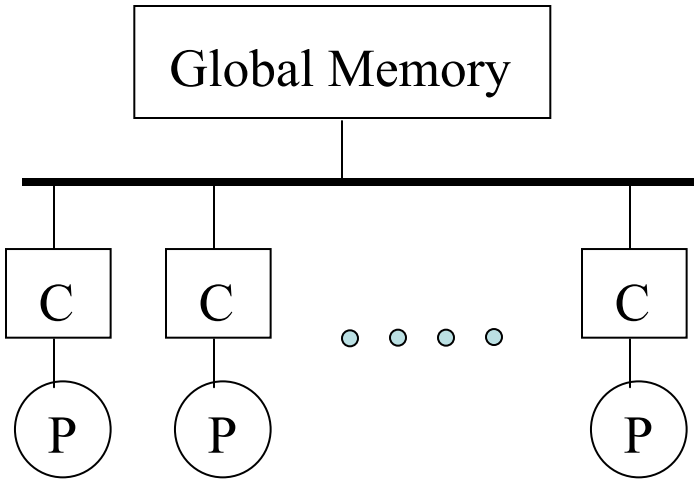
4.1 Classification of Shared Memory Systems

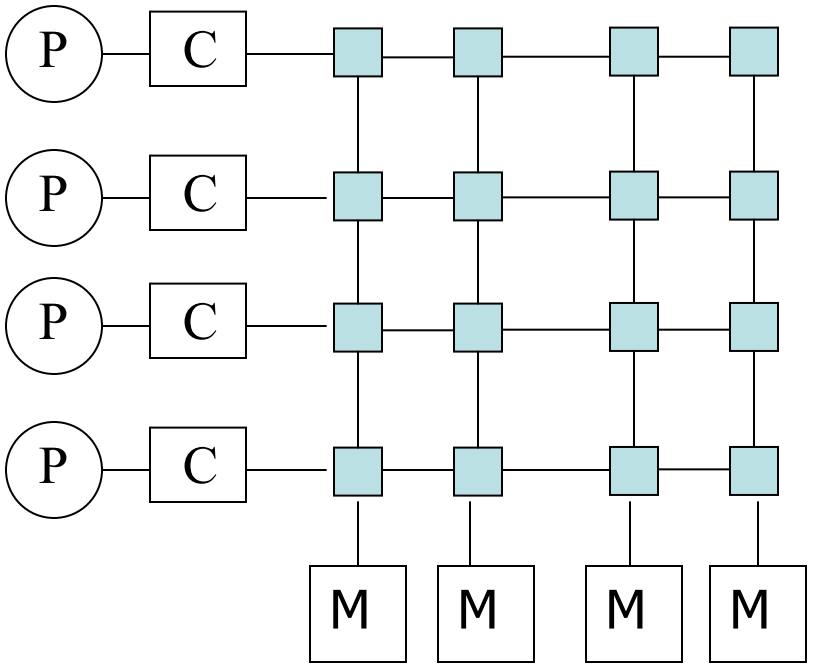
- Cache-Only Memory Architecture (COMA)
  - Like NUMA, each processor has part of the shared memory in COMA.
  - COMA requires the data to be migrated to the processor requesting it.



4.2 Bus-Based Symmetric Multiprocessors

- Shared memory systems can be designed using bus-based or switch-based interconnection networks.



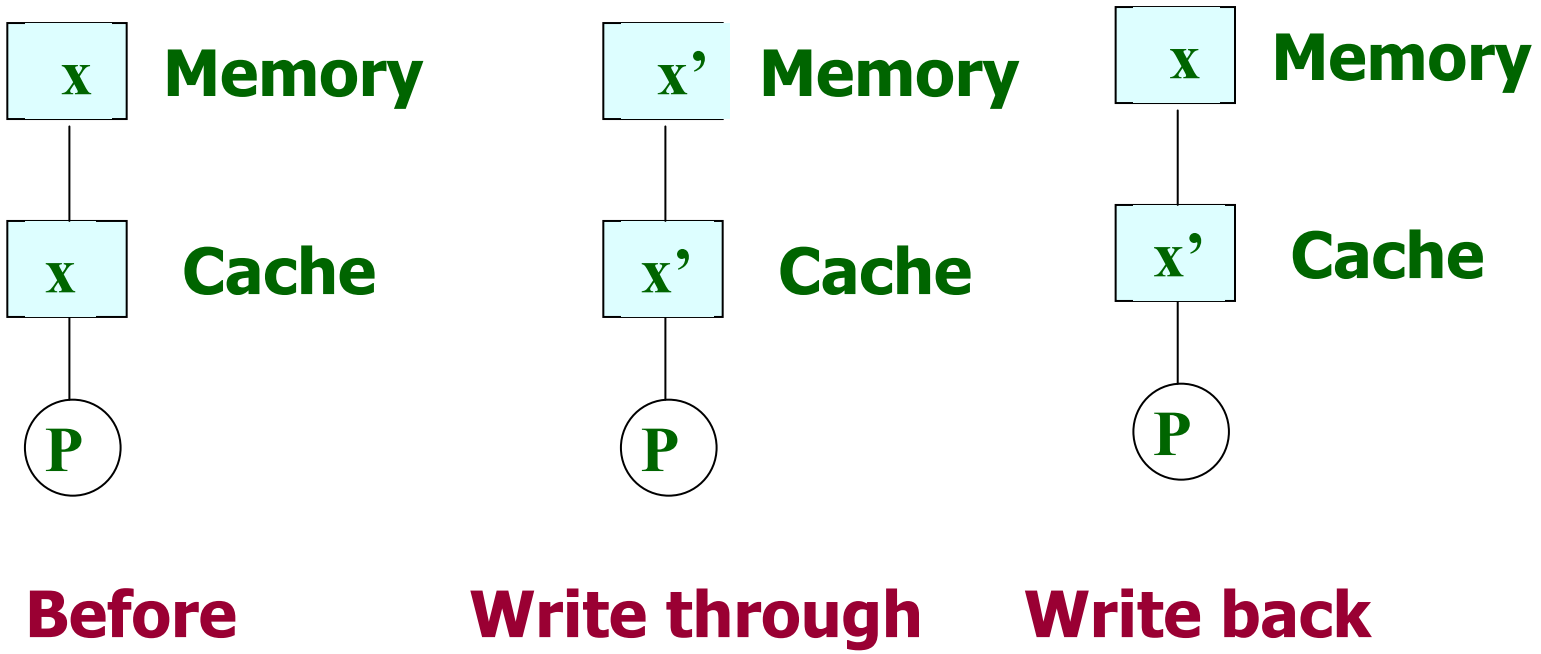


4.3 Basic Cache Coherency Methods

- Cache-memory coherence using two policies:
  - Write-Through:
    - The memory is updated every time the cache is updated.
  - Write-Back:
    - The memory is updated only when the block in the cache is being replaced.

4.3 Basic Cache Coherency Methods

- Cache-memory coherence using two policies:

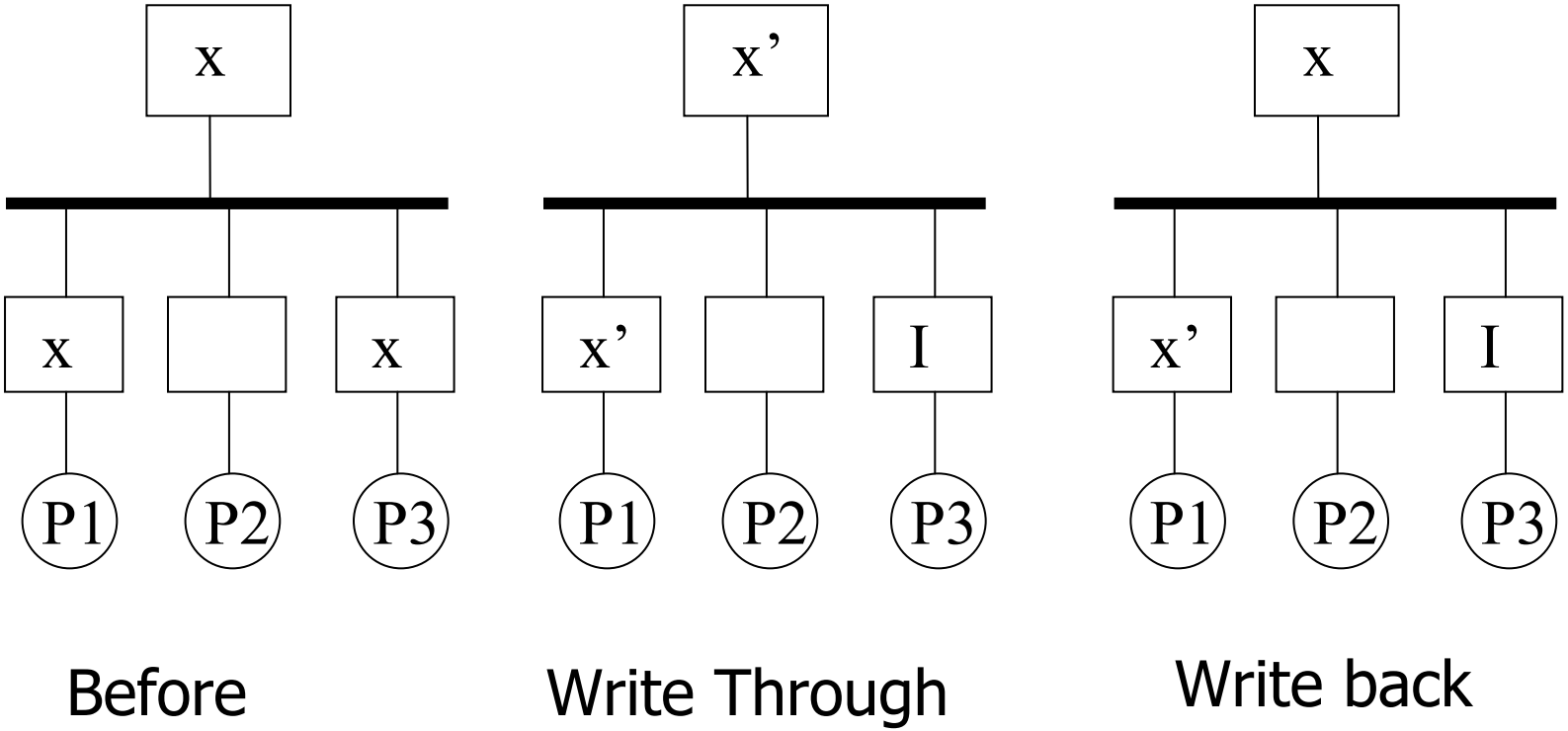


4.3 Basic Cache Coherency Methods

- Cache-cache coherence using two policies:
  - Write-Invalidate:
    - Maintains consistency by reading from local caches until a write occurs.
  - Write Update:
    - Maintains consistency by immediately updating all copies in all caches.

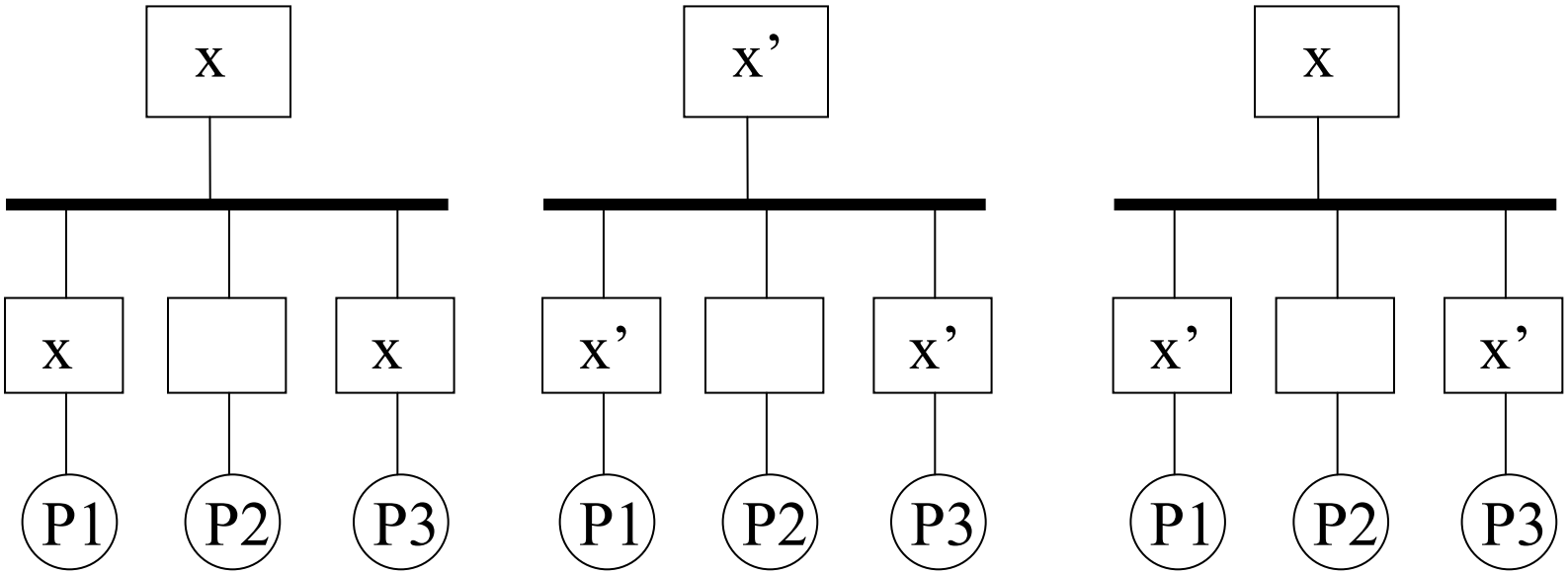
4.3 Basic Cache Coherency Methods

- Write-Invalidate:



4.3 Basic Cache Coherency Methods

- Write-Update:



Before

Write Through

Write back

4.3 Basic Cache Coherency Methods

- Shared Memory System Coherence :
  - Four combinations to maintain coherence among all caches and global memory:
    - Write-Update and Write-Through.
    - Write-Update and Write-Back.
    - Write-Invalidate and Write-Through.
    - Write-Invalidate and Write-Back.

4.4 Snooping Protocols

- Based on watching bus activities and carry out the appropriate coherency commands when necessary.
- Global memory is moved in blocks, and each block has a state associated with it, which determines what happens to the entire contents of the block.
- The state of a block might change as a result of the operations Read-Miss, Read-Hit, Write-Miss, and Write-Hit.

4.4 Snooping Protocols

- Write-Invalidate and Write-Through
  - Multiple processors can read block copies from main memory safely until one processor updates its copy.
  - At this time, all cache copies are invalidated and the memory is updated to remain consistent.

4.4 Snooping Protocols

- Write-Invalidate and Write-Through

State	Description
Valid [VALID]	**The copy is consistent with global memory**
Invalid [INV]	**The copy is inconsistent**

4.4 Snooping Protocols

- Write-Invalidate and Write-Through

Event	Actions
Read Hit	**Use the local copy from the cache.**
Read Miss	**Fetch a copy from global memory. Set the state of this copy to Valid.**
Write Hit	**Perform the write locally. Broadcast an Invalid command to all caches. Update the global memory.**
Write Miss	**Get a copy from global memory. Broadcast an invalid command to all caches. Update the global memory. Update the local copy and set its state to Valid.**
Replace	**Since memory is always consistent, no write back is needed when a block is replaced.**

4.4 Snooping Protocols

- Write-Invalidate and Write-Back (Ownership Protocol)
  - A valid block can be owned by memory and shared in multiple caches that can contain only the shared copies of the block.
  - Multiple processors can safely read these blocks from their caches until one processor updates its copy.
  - At this time, the writer becomes the only owner of the valid block and all other copies are invalidated.

4.4 Snooping Protocols

- Write-Invalidate and Write-Back (Ownership Protocol)

State	Description
Shared (Read-Only) [RO]	**Data is valid and can be read safely. Multiple copies can be in this state**
Exclusive (Read-Write) [RW]	**Only one valid cache copy exists and can be read from and written to safely. Copies in other caches are invalid**
Invalid [INV]	**The copy is inconsistent**

4.4 Snooping Protocols

- Write-Invalidate and Write-Back

Event	Action
Read Hit	**Use the local copy from the cache.**
Read Miss:	**If no Exclusive (Read-Write) copy exists, then supply a copy from global memory. Set the state of this copy to Shared (Read-Only). If an Exclusive (Read-Write) copy exists, make a copy from the cache that set the state to Exclusive (Read-Write), update global memory and local cache with the copy. Set the state to Shared (Read-Only) in both caches.**

4.4 Snooping Protocols

- Write-Invalidate and Write-Back

Write Hit	**If the copy is Exclusive (Read-Write), perform the write locally. If the state is Shared (Read-Only), then broadcast an Invalid to all caches. Set the state to Exclusive (Read-Write).**
Write Miss	**Get a copy from either a cache with an Exclusive (Read-Write) copy, or from global memory itself. Broadcast an Invalid command to all caches. Update the local copy and set its state to Exclusive (Read-Write).**
Block Replacement	**If a copy is in an Exclusive (Read-Write) state, it has to be written back to main memory if the block is being replaced. If the copy is in Invalid or Shared (Read-Only) states, no write back is needed when a block is replaced.**

4.4 Snooping Protocols

- Write-Once
  - This write-invalidate protocol, which was proposed by Goodman in 1983 uses a combination of write-through and write-back.
  - Write-through is used the very first time a block is written. Subsequent writes are performed using write back.

4.4 Snooping Protocols

- Write-Once

State	Description
Invalid [INV]	**The copy is inconsistent.**
Valid [VALID]	**The copy is consistent with global memory.**
Reserved [RES]	**Data has been written exactly once and the copy is consistent with global memory. There is only one copy of the global memory block in one local cache.**
Dirty [DIRTY]	**Data has been updated more than once and there is only one copy in one local cache. When a copy is dirty, it must be written back to global memory**

4.4 Snooping Protocols

- Write-Once

Event	Actions
Read Hit	**Use the local copy from the cache.**
Read Miss	**If no Dirty copy exists, then supply a copy from global memory. Set the state of this copy to Valid. If a dirty copy exists, make a copy from the cache that set the state to Dirty, update global memory and local cache with the copy. Set the state to VALID in both caches.**

4.4 Snooping Protocols

- Write-Once

Write Hit	**If the copy is Dirty or Reserved, perform the write locally, and set the state to Dirty. If the state is Valid, then broadcast an Invalid command to all caches. Update the global memory and set the state to Reserved.**
Write Miss	**Get a copy from either a cache with a Dirty copy or from global memory itself. Broadcast an Invalid command to all caches. Update the local copy and set its state to Dirty.**
Block Replacement	**If a copy is in a Dirty state, it has to be written back to main memory if the block is being replaced. If the copy is in Valid, Reserved, or Invalid states, no write back is needed when a block is replaced.**

4.4 Snooping Protocols

- Write-Update and Partial Write-Through
  - In this protocol an update to one cache is written to memory at the same time it is broadcast to other caches sharing the updated block.
  - These caches snoop on the bus and perform updates to their local copies.
  - There is also a special bus line, which is asserted to indicate that at least one other cache is sharing the block.

4.4 Snooping Protocols

- Write-Update and Partial Write-Through

State	Description
Valid Exclusive [VAL-X]	This is the only cache copy and is consistent with global memory
Shared [SHARE]	There are multiple caches copies shared. All copies are consistent with memory
Dirty [DIRTY]	This copy is not shared by other caches and has been updated. It is not consistent with global memory. (Copy ownership)

4.4 Snooping Protocols

- Write-Update and Partial Write-Through

Event	Action
Read Hit	**Use the local copy from the cache. State does not change**
Read Miss:	**If no other cache copy exists, then supply a copy from global memory. Set the state of this copy to Valid Exclusive. If a cache copy exists, make a copy from the cache. Set the state to Shared in both caches. If the cache copy was in a Dirty state, the value must also be written to memory.**

4.4 Snooping Protocols

- Write-Update and Partial Write-Through

Write Hit	**Perform the write locally and set the state to Dirty. If the state is Shared, then broadcast data to memory and to all caches and set the state to Shared. If other caches no longer share the block, the state changes from Shared to Valid Exclusion.**
Write Miss	**The block copy comes from either another cache or from global memory. If the block comes from another cache, perform the update and update all other caches that share the block and global memory. Set the state to Shared. If the copy comes from memory, perform the write and set the state to Dirty.**
Block Replacement	**If a copy is in a Dirty state, it has to be written back to main memory if the block is being replaced. If the copy is in Valid Exclusive or Shared states, no write back is needed when a block is replaced.**

4.4 Snooping Protocols

- Write-Update and Write-Back

  – This protocol is similar to the previous one except that instead of writing through to the memory whenever a shared block is updated, memory updates are done only when the block is being replaced.

4.4 Snooping Protocols

- Write-Update and Write-Back

State	Description
Valid Exclusive [VAL-X]	This is the only cache copy and is consistent with global memory
Shared Clean [SH-CLN]	There are multiple caches copies shared.
Shared Dirty [SH-DRT]	There are multiple shared caches copies. This is the last one being updated. (Ownership)
Dirty [DIRTY]	This copy is not shared by other caches and has been updated. It is not consistent with global memory. (Ownership)

4.4 Snooping Protocols

- Write-Update and Write-Back

Event	Action
Read Hit	**Use the local copy from the cache. State does not change**
Read Miss:	**If no other cache copy exists, then supply a copy from global memory. Set the state of this copy to Valid Exclusive. If a cache copy exists, make a copy from the cache. Set the state to Shared Clean. If the supplying cache copy was in a Valid Exclusion or Shared Clean, its new state becomes Shared Clean. If the supplying cache copy was in a Dirty or Shared Dirty state, its new state becomes Shared Dirty.**

4.4 Snooping Protocols

- Write-Update and Write-Back

Write Hit	**If the sate was Valid Exclusive or Dirty, Perform the write locally and set the state to Dirty. If the state is Shared Clean or Shared Dirty, perform update and change state to Shared Dirty. Broadcast the updated block to all other caches. These caches snoop the bus and update their copies and set their state to Shared Clean.**
Write Miss	**The block copy comes from either another cache or from global memory. If the block comes from another cache, perform the update, set the state to Shared Dirty, and broadcast the updated block to all other caches. Other caches snoop the bus, update their copies, and change their state to Shared Clean. If the copy comes from memory, perform the write and set the state to Dirty.**
Block Replacement	**If a copy is in a Dirty or Shared Dirty state, it has to be written back to main memory if the block is being replaced. If the copy is in Valid Exclusive, no write back is needed when a block is replaced.**

4.5 Directory Based Protocols

- Due to the nature of some interconnection networks and the size of the shared memory system, updating or invalidating caches using snoopy protocols might become unpractical.

- Cache coherence protocols that somehow store information on where copies of blocks reside are called directory schemes.

4.5 Directory Based Protocols

- A directory is a data structure that maintains information on the processors that share a memory block and on its state.
- The information maintained in the directory could be either centralized or distributed.
- A Central directory maintains information about all blocks in a central data structure.
- The same information can be handled in a distributed fashion by allowing each memory module to maintain a separate directory.

4.5 Directory Based Protocols

- Protocol Categorization:
  - Full Map Directories.
  - Limited Directories.
  - Chained Directories.

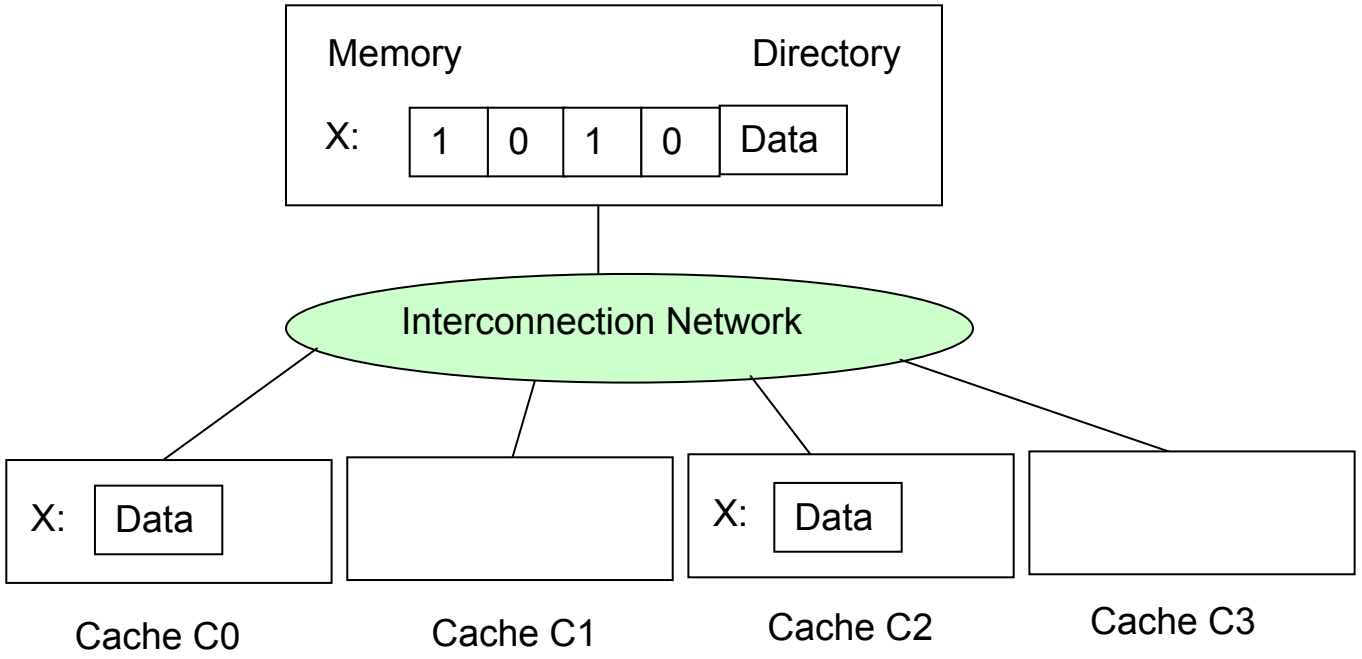
4.5 Directory Based Protocols

– Full-map Directories:

- Each directory entry contains *N* pointers, where *N* is the number of processors.
- There could be *N* cached copies of a particular block shared by all processors.
- For every memory block, an *N* bit vector is maintained, where *N* equals the number of processors in the shared memory system. Each bit in the vector corresponds to one processor.

4.5 Directory Based Protocols

– Full-map Directories:



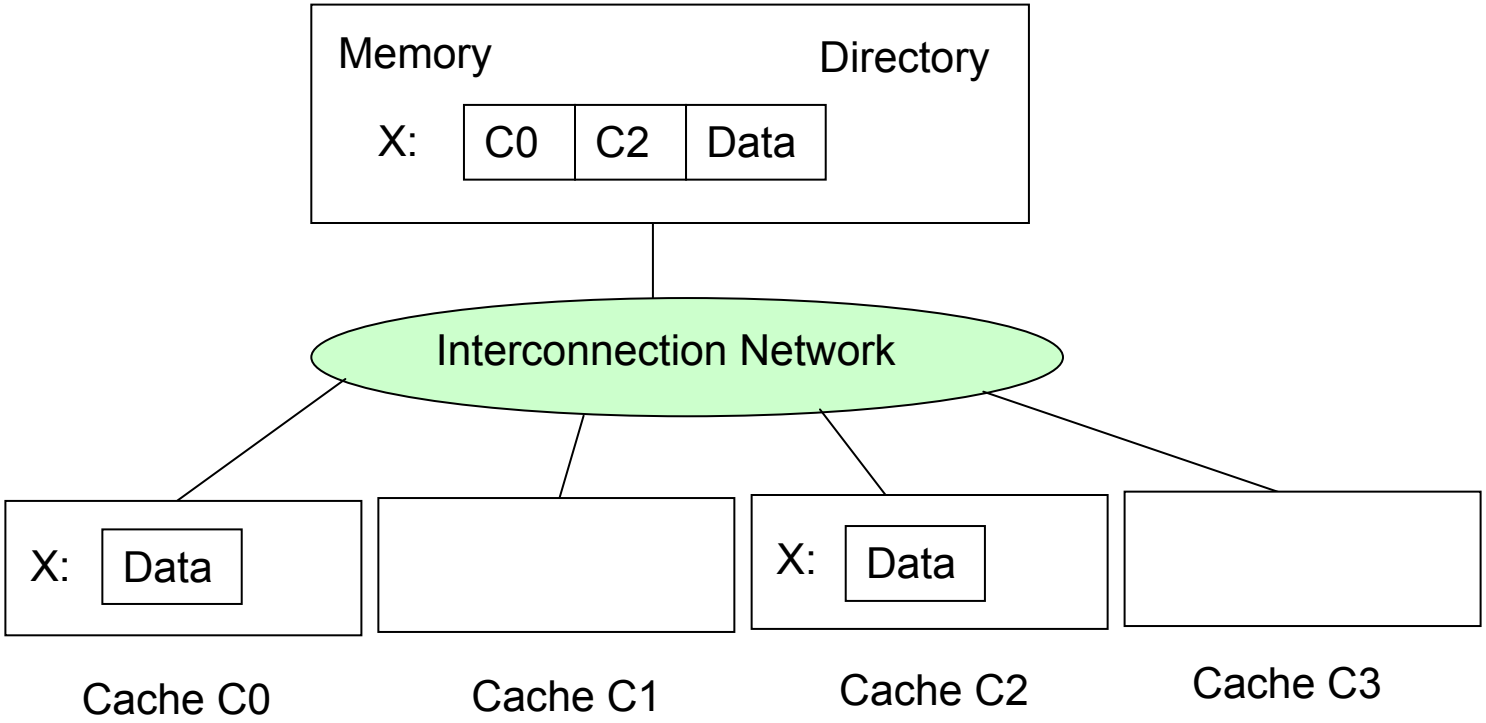
4.5 Directory Based Protocols

– Limited Directories:

- Fixed number of pointers per directory entry regardless of the number of processors.
- Restricting the number of simultaneously cached copies of any block should solve the directory size problem that might exist in full-map directories.

4.5 Directory Based Protocols

– Limited Directories:



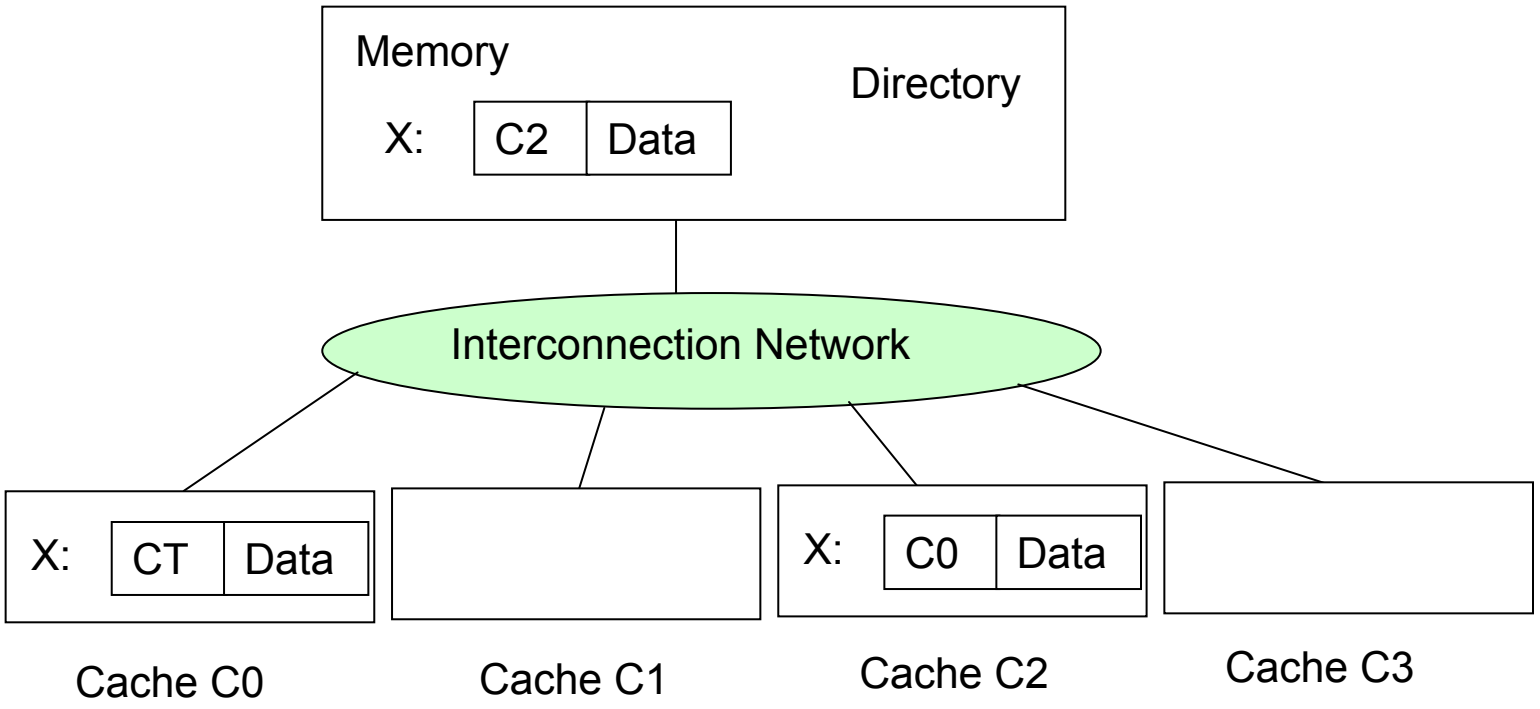
4.5 Directory Based Protocols

– Chained Directories:

- Chained directories emulate full-map by distributing the directory among the caches.
- Solving the directory size problem without restricting the number of shared block copies.
- Chained directories keep track of shared copies of a particular block by maintaining a chain of directory pointers.

4.5 Directory Based Protocols

– Chained Directories:



4.5 Directory Based Protocols

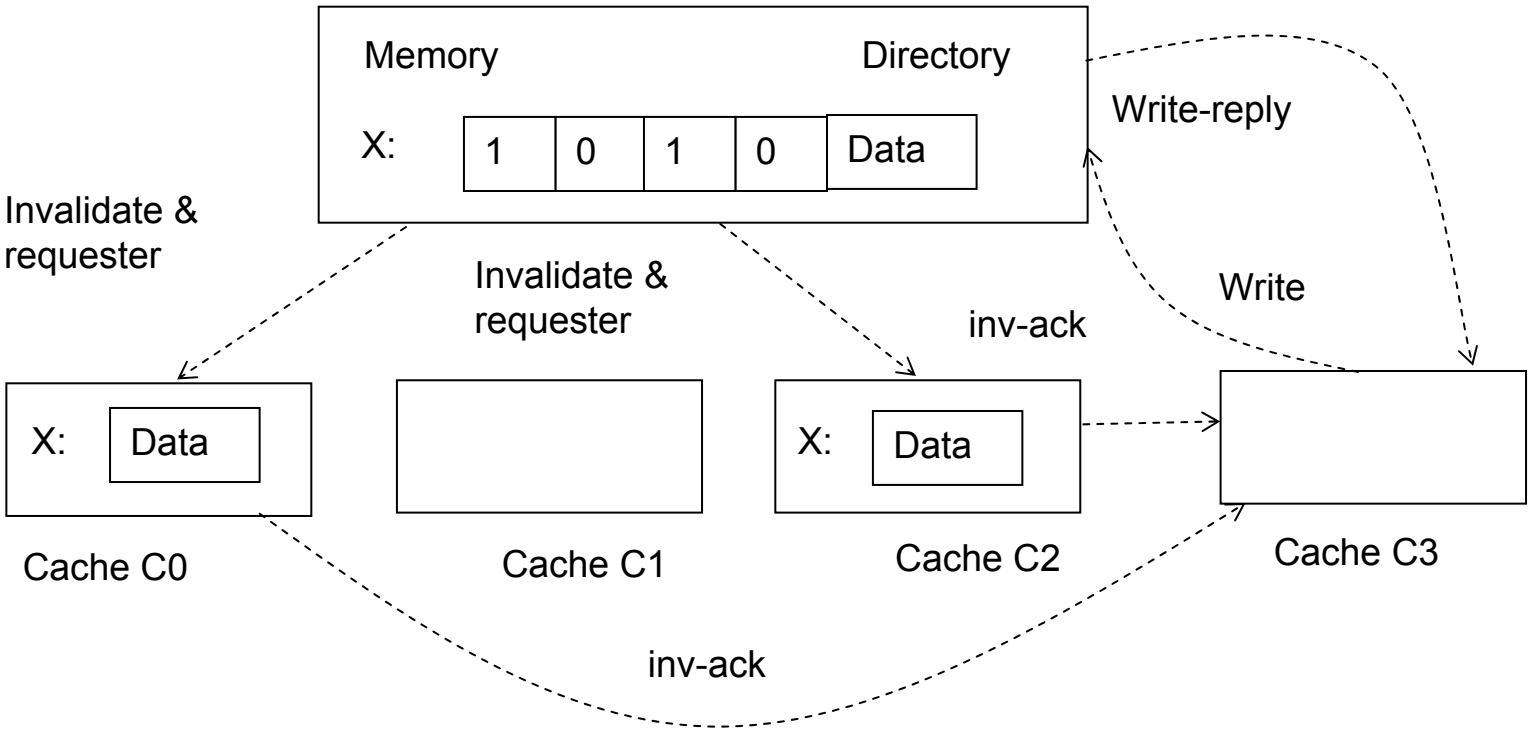
- Invalidate Protocols:
  - Centralized directory invalidate.
  - Scalable Coherent Interface (SCI).
  - Stanford Distributed Directory (SDD).

4.5 Directory Based Protocols

- Centralized Directory Invalidate
  - Invalidating signals and a pointer to the requesting processor are forwarded to all processors that have a copy of the block.
  - Each invalidated cache sends an acknowledgment to the requesting processor.
  - After the invalidation is complete, only the writing processor will have a cache with a copy of the block.

4.5 Directory Based Protocols

- Centralized Directory Invalidate – Write by P3



4.5 Directory Based Protocols

- Scalable Coherent Interface (SCI)
  - Doubly linked list of distributed directories.
  - Each cached block is entered into a list of processors sharing that block.
  - For every block address, the memory and cache entries have additional tag bits. Part of the memory tag identifies the first processor in the sharing list (the head). Part of each cache tag identifies the previous and following sharing list entries.

4.5 Directory Based Protocols

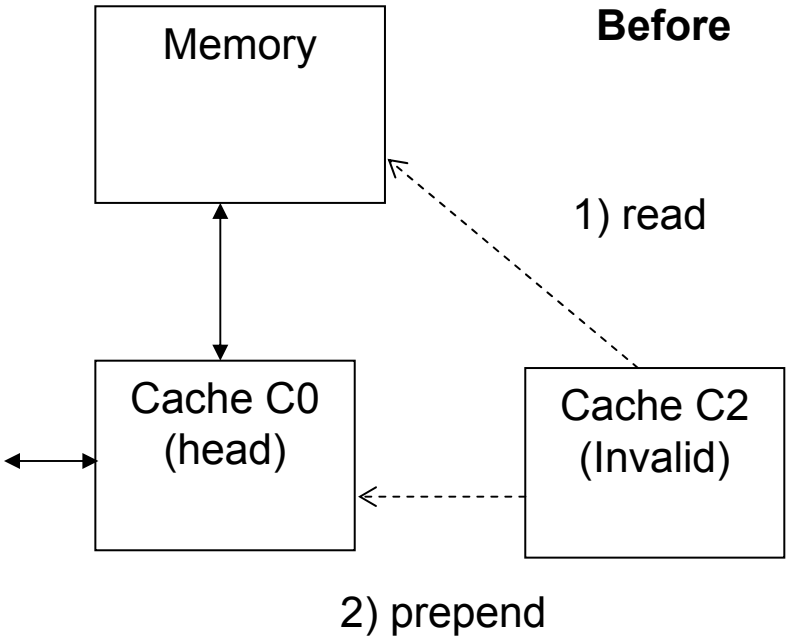
- Scalable Coherent Interface (SCI)
  - Initially memory is in the uncached state and cached copies are invalid.
  - A read request is directed from a processor to the memory controller. The requested data is returned to the requester's cache and its entry state is changed from invalid to the head state.
  - This changes the memory state from uncached to cached.

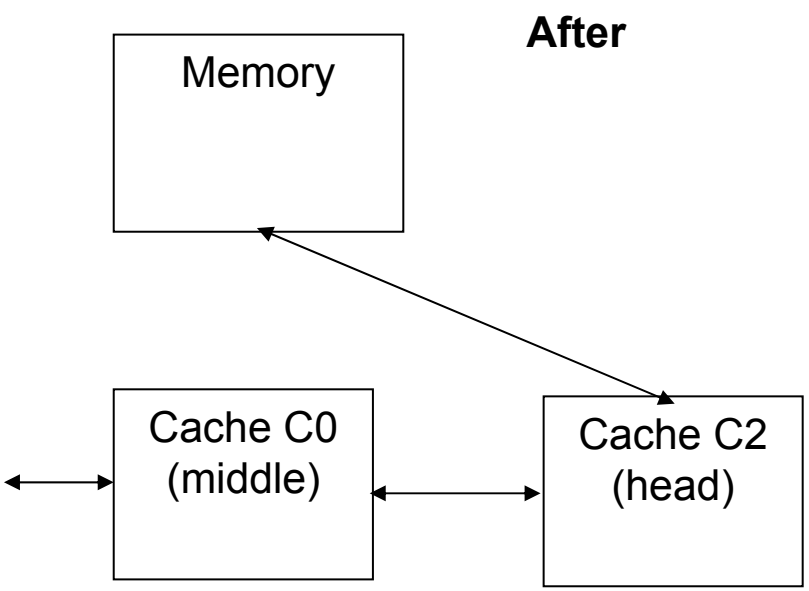
4.5 Directory Based Protocols

- Scalable Coherent Interface (SCI)
  - When a new requester directs its read request to memory, the memory returns a pointer to the head.
  - A cache-to-cache read request (called Prepend) is sent from the requester to the head cache.
  - On receiving the request, the head cache sets its backward pointer to point to the requester's cache.
  - The requested data is returned to the requester's cache and its entry state is changed to the head state.

4.5 Directory Based Protocols

- Scalable Coherent Interface (SCI)-Sharing list addition



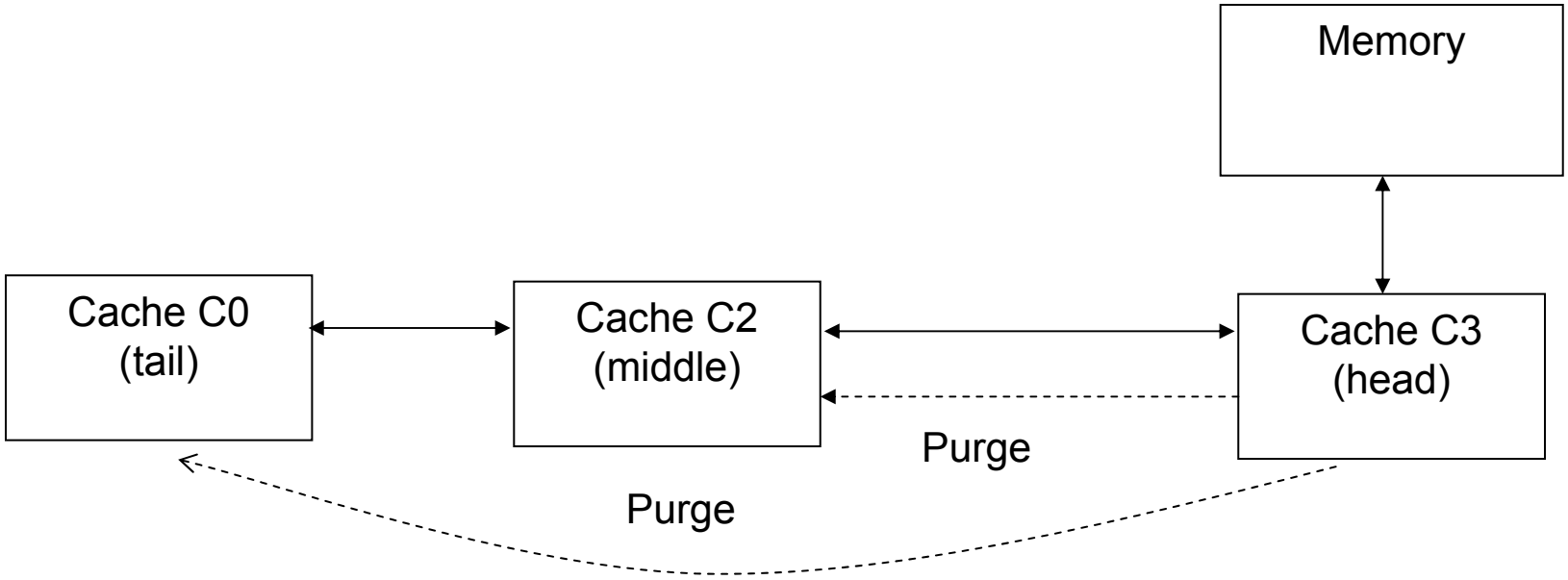


4.5 Directory Based Protocols

- Scalable Coherent Interface (SCI)
  - The head of the list has the authority to purge other entries in the list to obtain an exclusive (read-write) entry.

4.5 Directory Based Protocols

- Scalable Coherent Interface (SCI)-Head purging other entries.



4.5 Directory Based Protocols

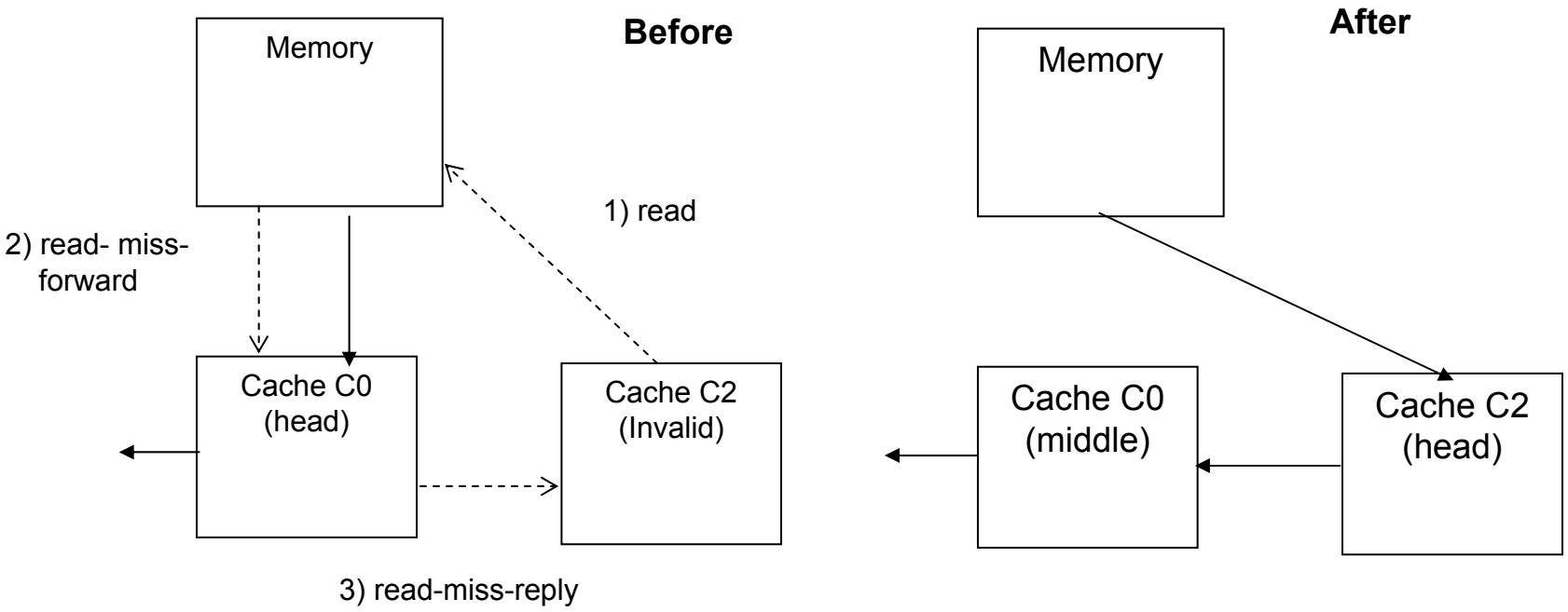
- Scalable Distributed Directory (SDD)
  - A singly linked list of distributed directories.
  - Similar to the SCI protocol, memory points to the head of the sharing list.
  - Each processor points only to its predecessor.
  - The sharing list additions and removals are handled different from the SCI protocol .

4.5 Directory Based Protocols

- Scalable Distributed Directory (SDD)
  - On a read miss, a new requester sends a read-miss message to memory.
  - The memory updates its head pointers to point to the requester and send a read-miss-forward signal to the old head.
  - On receiving the request, the old head returns the requested data along with its address as a read-miss-reply.
  - When the reply is received, at the requester's cache, the data is copied and the pointer is made to point to the old head.

4.5 Directory Based Protocols

- Scalable Distributed Directory (SDD)-List addition



4.5 Directory Based Protocols

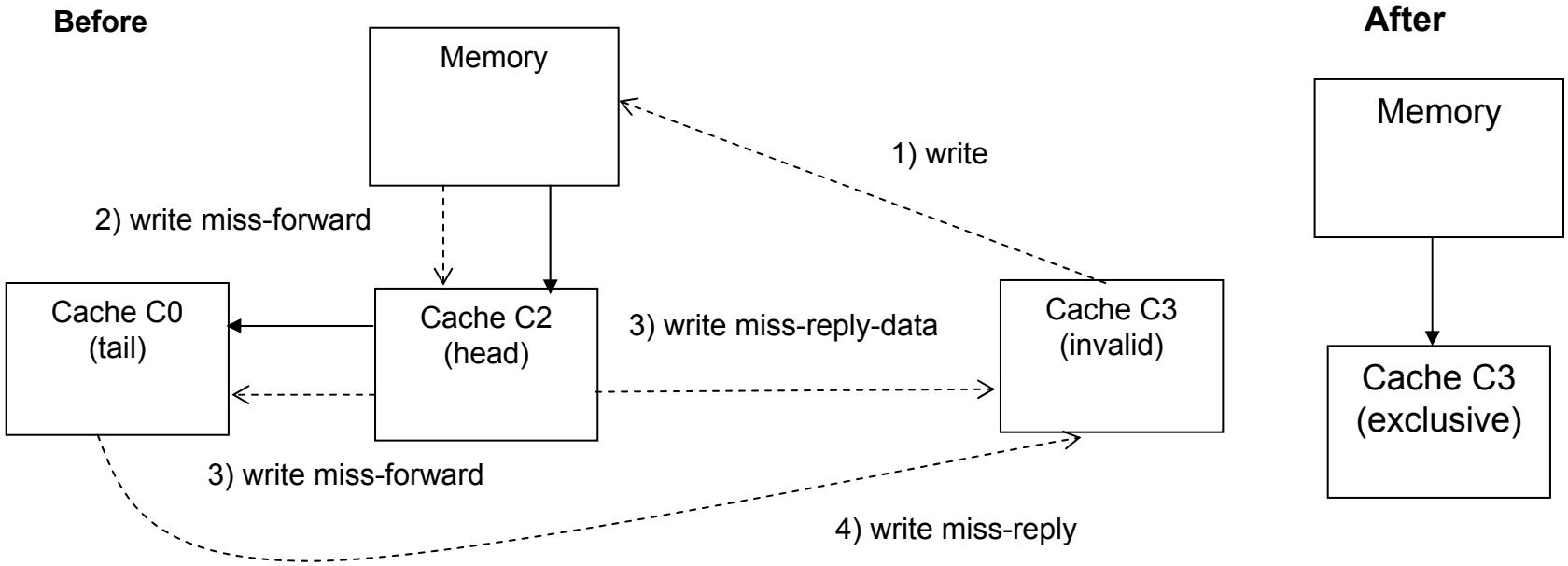
- Scalable Distributed Directory (SDD)
  - On a write miss, a requester sends a write-miss message to memory.
  - The memory updates its head pointers to point to the requester and sends a write-miss-forward signal to the old head.
  - The old head invalidates itself, returns the requested data as a write-miss-reply-data signal, and send a write-miss-forward to the next cache in the list.

4.5 Directory Based Protocols

- Scalable Distributed Directory (SDD)
  - When the next cache receives the write-miss-forward signal, it invalidates itself and sends a write-miss-forward to the next cache in the list.
  - When the write-miss-forward signal is received by the tail or by a cache that no longer has copy of the block, a write-miss-reply is sent to the requester.
  - The write is complete when the requester receives both write-miss-reply-data and write-miss-reply.

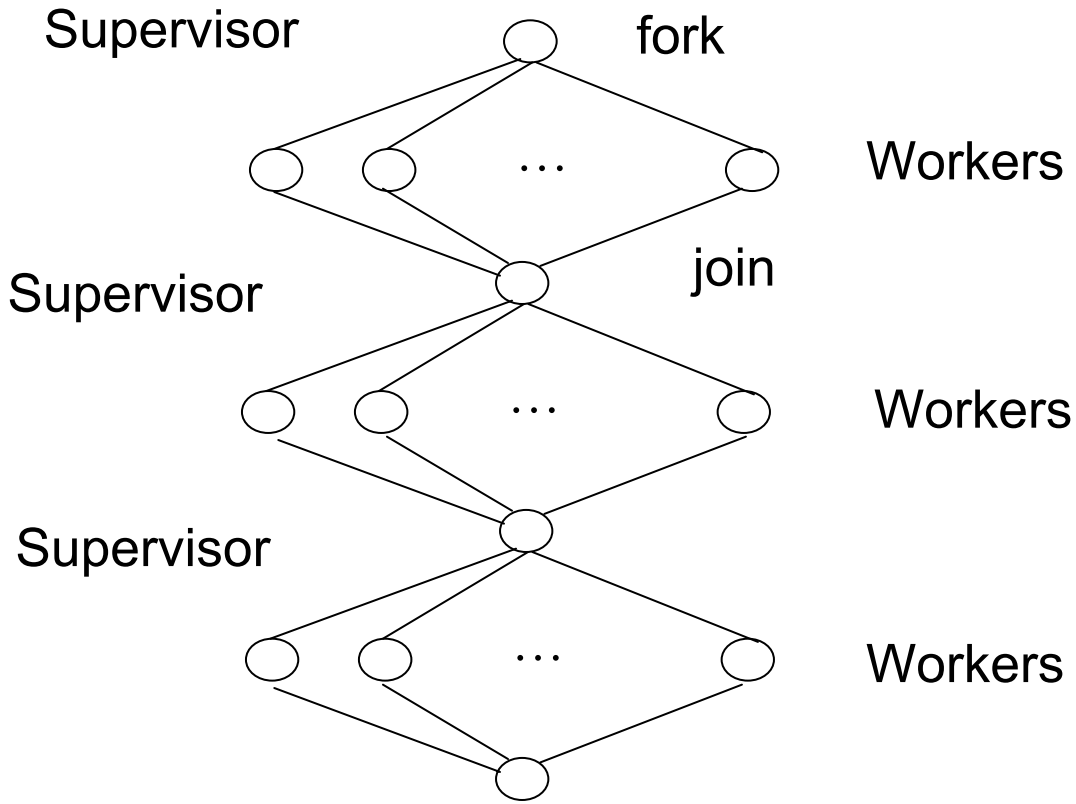
4.5 Directory Based Protocols

- Scalable Distributed Directory (SDD)- Write Miss List Removal



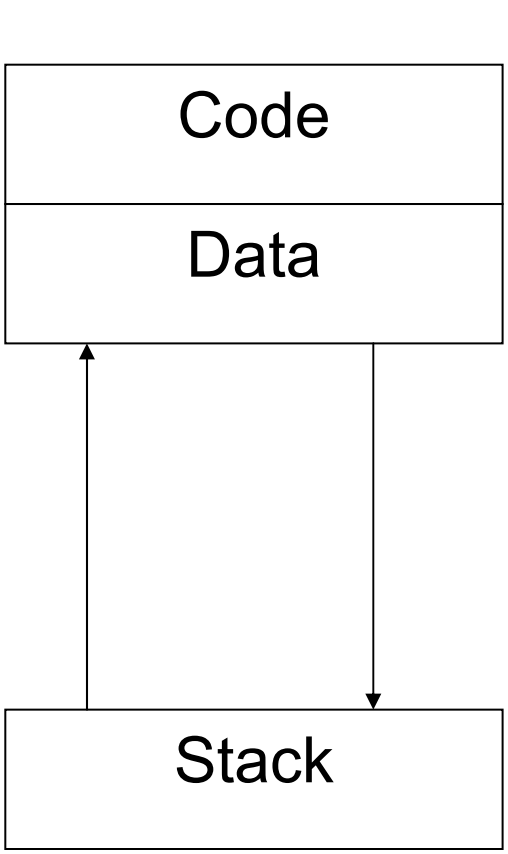
4.6 Shared Memory Programming

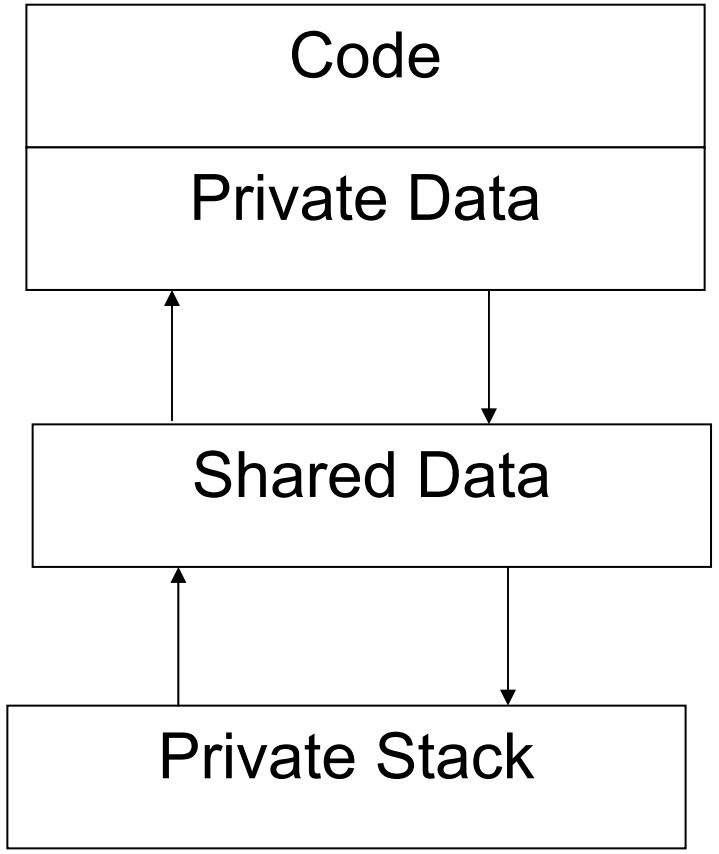
- Task Creation



4.6 Shared Memory Programming

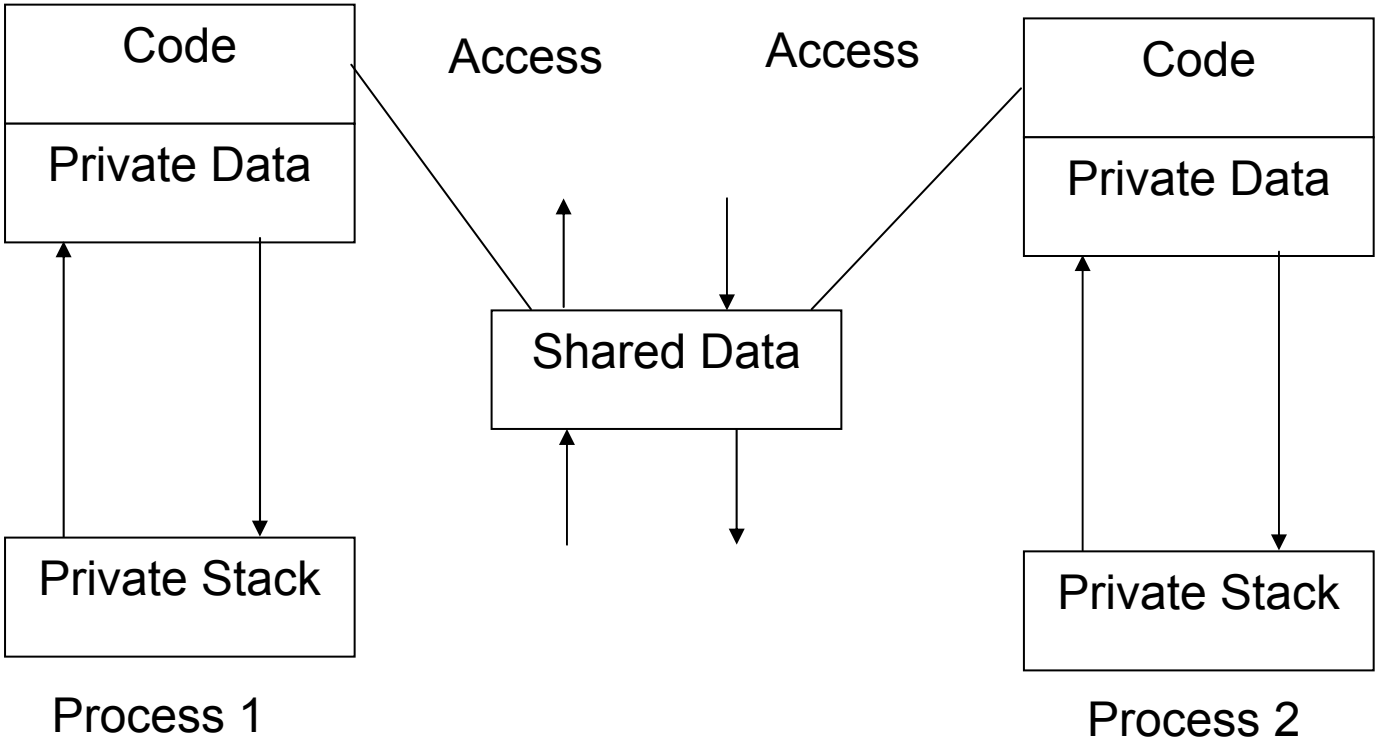
- Communication – Serial versus Parallel





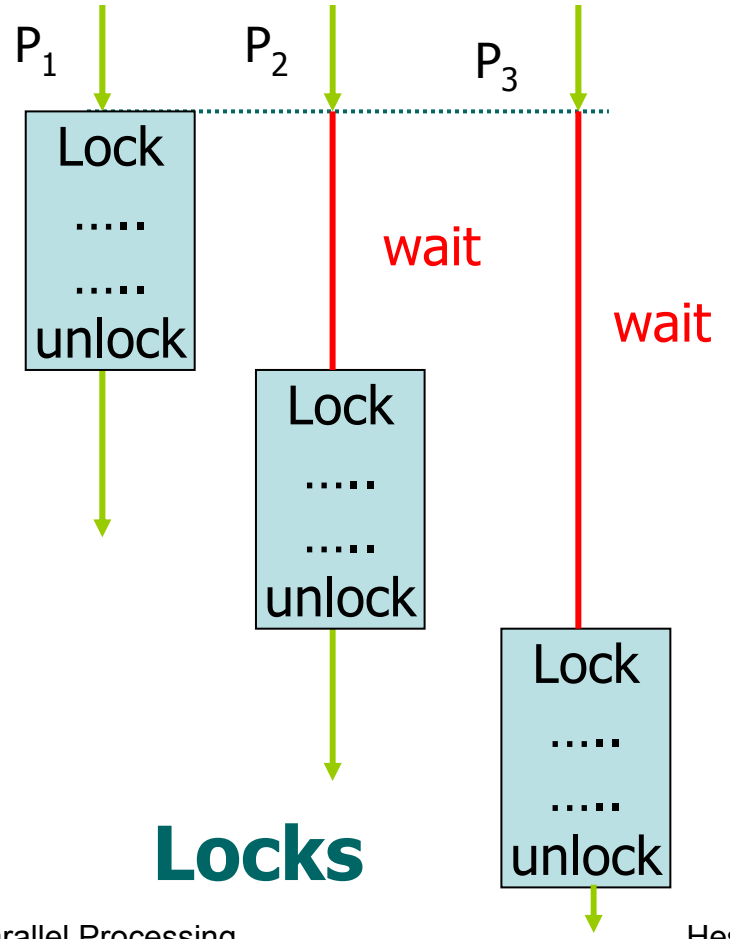
4.6 Shared Memory Programming

- Communication – Via shared data



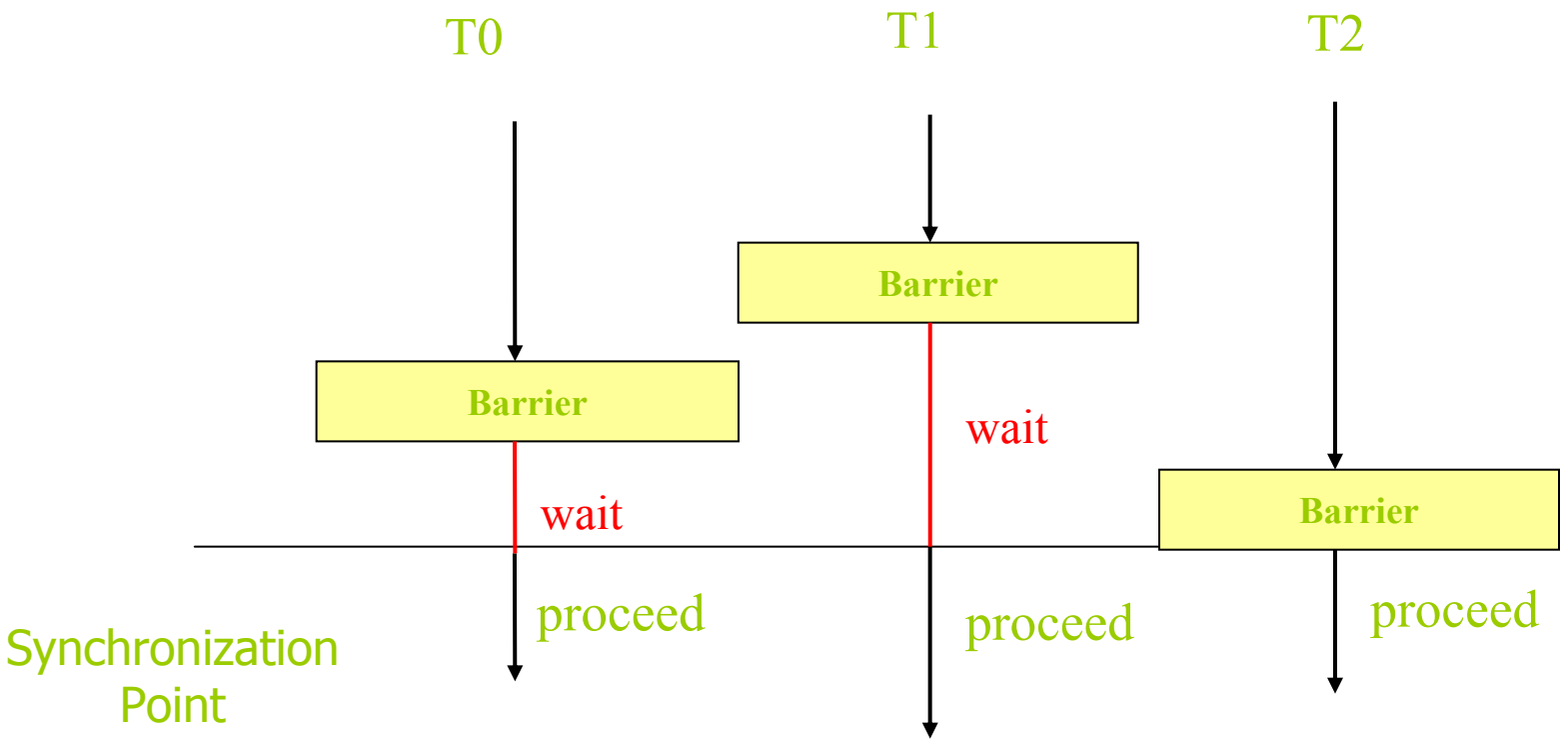
4.6 Shared Memory Programming

- Synchronization - Locks



4.6 Shared Memory Programming

- Synchronization - Barriers



4.7 Summary

- A shared memory system is made of multiple processors and memory modules that are connected via some interconnection network.
- Shared memory multiprocessors are usually bus-based or switch-based.
- Communication among processors is achieved by writing to and reading from memory.
- Synchronization among processors is achieved using locks and barriers.
- Performance of a shared memory system becomes an issue when the interconnection network connecting the processors to global memory becomes a bottleneck.

4.7 Summary

- Local caches are used to alleviate the bottleneck problem.
- But the introduction of caches created a consistency problem among caches and between memory and caches.
- Cache coherence schemes can be categorized into 2 main categories:
  - Snooping protocols.
  - Directory-based protocols.
- The programming model in shared memory systems has proven to be easier to program compared to message passing systems.