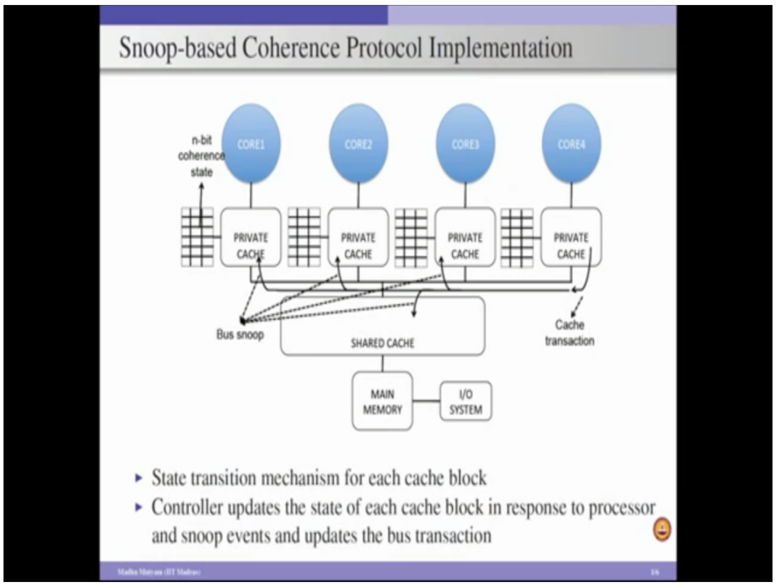
Computer Architecture
Prof. Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 08
Lecture – 30
Cache Coherence Protocol Design

So, in this module we are going to discuss cache coherence protocol design for snoop based systems as well as directory based systems. And also we discuss as a case study 3 state invalidation based protocol.

(Refer Time Slide: 00:28)



Snoop-based Coherence Protocol Implementation

- State transition mechanism for each cache block
- Controller updates the state of each cache block in response to processor and snoop events and updates the bus transaction

So, first we start with snoop based design. So, when we have the number of cores in our multicore system is 4 or 8 then we can consider these cores through a bus based communication mechanism. So, for example, in this particular figure we can clearly see here it is a 4 core system and each of these cores are connected through a common bus. And here each core has a private cache and there is a shared cached, this shared cache is shared by all the cores and after that there is a memory and I/O system.

And in order to implement a cache coherence protocol. So, for each of this private caches we are going to have separate table. And this table is going to give state information associated with each of the blocks that is present in this private cache. So, effectively there is a one on one match between each block in the private cache with entry in this table. So, here each entry is providing n bit coherence state. So, this is going to specify the state in which the block corresponding block is present at that particular point of time.

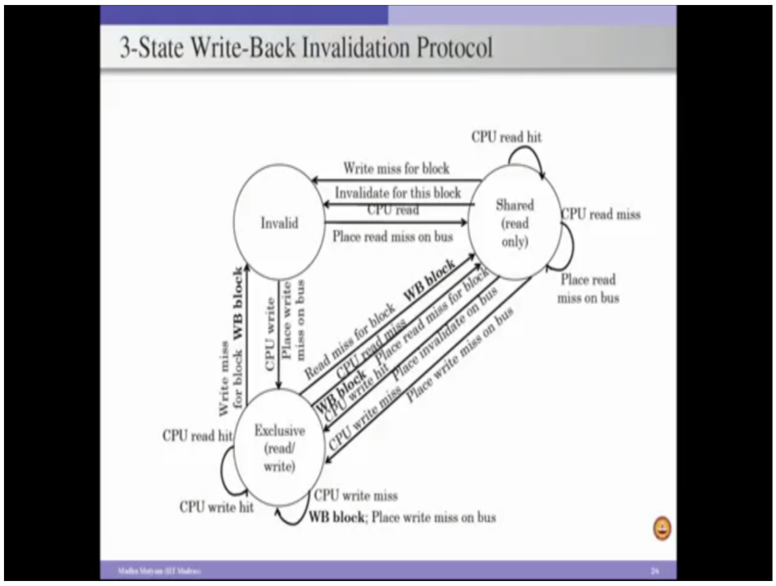
If you are considering a 4 state cache coherence protocol, then we are going to have 2 bit information associated with each of the blocks present in this private cache. So, given this multicore system, now when we want to implement snoop based cache coherence protocol. So, what typically happens is, whenever a core wants to update or whenever a core wants to read some data and if it finds a miss in it private cache, then it is going to place a transaction on the bus.

For example, see here core 4 wants to access some data and it finds a miss in the private cache. So, it is going to place a transaction on the bus. And as soon as a transaction is placed on the bus all other cache controllers associated with our multicore system will snoop on the bus and they take this transaction and using the address specified in this transaction they are going to search in their private caches or the associated shared cache. So, as a second step these cache controllers will snoop the bus and they will search in their caches.

And if any cache finds a match for this request, then that cache is going to respond to this particular action, either by supplying the data or by invalidating the data or updating the data and so on that depends on the type of protocol what we are going to design. So, effectively in the snoop based system the cores are connected by a common bus and whenever any core wants to initiate any transaction, it is going to place the transaction on the bus. And all other cache controllers will snoop on the bus and they will take the appropriate action in their caches.

So, here we are going to maintain state information for each of the blocks and there is a state transition diagram associated with each block. And depending on the transaction that is placed on the bus. So, the state of a block can be changed by the cache controller. So, effectively the cache controller updates the state of each block in response to processor and snoop events that are there on the bus. And also it updates the bus transactions. So, this is the overall idea of snoop based cache coherence protocol. Now, we are going to discuss a 3 state cache coherence protocol for write back caches and we are going to use invalidation based mechanism

(Refer Time Slide: 04:26)



3-State Write-Back Invalidation Protocol

So, in 3 state write back invalidation based protocol, we have invalid state, shared state and exclusive or modify state. So, here as I mentioned previously, so we have state information for each of the blocks present in the cache. So, now this state transition is going to happen based on the protocol whatever we are designing. And now we will see how the state transitions happen when we consider invalidation based protocol.

So, now consider a scenario where a core 4 wants to read some data. So, it is going to issue a CPU read request and unfortunately the cache associated with core 4 finds that the request is a miss. And wherever core 4 incurs a CPU read miss and it asks the cache replacement policy to select a victim block. So, that it can keep the new data into that victim block location. And assume that the victim block location is in a state invalid. Now, we have to change the state of this location from invalid to shared. So, that is what is represented by this transition.

And also because now core 4 incurs a read miss. So, the cache controller associated with core 4 is going to place a transaction on the bus and transaction is read miss transaction for this particular address. And now consider another scenario - core 4 generated a read request and which is a miss in the cache. Now, the cache replacement policy identifies a victim block which is in a location whose state is exclusive then what is going to happen? Whenever the victim block is in a location exclusive, in that case also for a CPU read miss we are going to change the state of that location from exclusive to shared and also we are going to place a transaction on the bus that is read miss transaction for this block.

And compared to this invalid to shared transition, when we are moving from exclusive to shared we are also having another operation that is write back block operation because our victim block is in an exclusive state. That indicates that the victim block is having the dirty data and we have to write this dirty data back to the lower level cache. So, in order to update the lower level cache with this dirty data we have to write this data back and that is indicated by this write back block operation.

So, as a result whenever we are moving from exclusive state to shared state, we have to write whatever the data that is present in this exclusive state location to the lower level cache. So, that is what we have performed here. Now, consider the third scenario. Again core 4 initiated a read miss and cache replacement policy identifies a victim block, but now the victim block is in a location whose state is shared. Now, what is going to happen?

When the victim block is in a location whose state is shared then we are not going to change the state of this location and we retain the state as it is, but similar to all other transitions, here also we are going to place read miss transaction on the bus. When we are moving invalid to shared or exclusive to shared, we placed a read miss transaction on a bus for CPU read miss. And similarly, here also we are going to place read miss transaction on the bus. So, effectively in this 3 state protocol we have 3 states and for each of these states when there is a CPU read miss we are going to place read miss transaction on the bus.

Now, when we place read miss transaction on the bus all the cache controllers associated with our multicore system will snoop on the bus and they take this read miss transaction and they search in their caches and if they find the matching data then they have to respond provided they have the data in exclusive state. So, that is what is indicated by this. So, whenever let us say core 3 is having the requested data in modified state. Then core 3 cache controller will snoop on the bus and it finds that there is a transaction for a read miss for a particular block and it has the block in the modified state.

Then core 3 is going to write back this data by placing this data on the bus. So, that is what is indicated by this. And now after core 3 places this block of data on the bus, now core 4 is going to get this block of data in the appropriate location. And as a result now core 4 and core 3 both are going to have the same data, but now according to the definition of exclusive or modified state at any point of time only one instance of a block will be there in multicore system in a modified state.

When a block is in a modified state then automatically we have to know that only one instance of that block will be there in modified state, in all the caches in our system. And once a block is in a modified state even the shared cache is having a stale copy, but now because core 3 and core 4 both have the block of data which is same then as a result we have to change the state of this block in core 3. Previously the block state is in exclusive. Now, we have to change the state of this block in the core 3 by changing it from exclusive to shared.

Effectively at the end of this operation now core 3 and core 4 both have this block of data in the shared state. Now, we will consider write request. So, core 4 generated a CPU write request and it finds that this write request is a miss in the cache. Now, again the cache replacement policy associated with the private cache of core 4 is going to select a victim block and it finds a victim block which is in a location whose state is invalid. Now, what is that we are going to do?

When it finds block which is in an invalid state, now for this write miss request the cache controller associated with core 4 is going to place a transaction on the bus. And the transaction is write miss transaction for that particular block. So, whenever core 4 places this write miss transaction, it also changes the state of this location from invalid to exclusive because once the core 4 is going to get the data, either from some other cache or from the shared cache. Then it is going to have exclusive permission on this block to perform whatever the operation it requires because this is a write request and as a result we have to change the state from invalid to exclusive. That is what we have done.

And we will consider another scenario where core 4 generated a write request and again the cache replacement policy is going to select a victim block, but the victim block state is now exclusive. So, that means a victim block is in a location whose state is exclusive, then we are not going to change the state of this victim block location and we keep the state as it is. And also because this is a write request we are going to place a write miss transaction on the bus similar to whatever we have done earlier, but whenever we are evicting a dirty block, we have to write this data back to the shared cache.

So, that is what is performed by this write back block operation. So, other than this, the remaining things are same. So, in both the cases there is a write miss and as a result we are going to place write miss transaction on the bus. Now, consider the third scenario, if the cache replacement policy selects a victim block which is in a state shared. Now, what is that we are going to do? Whenever our victim block is in a location whose state is shared, now we are just simply changing the state of that location from shared to exclusive. And we are going to place the write miss transaction.

Note that here the victim block is in a shared state. So, as a result we do not have to write this data back to the shared cache because this is a shared state so, automatically our shared cache is actually having the up to data and as a result we do not have to write the data back. Note that only when we are evicting the dirty block we have to write the data to the lower level cache. In all other scenarios we would not have to write the data back to the lower level caches. So, now here when there is a write miss if the replacement policy selects an invalid block location, then we have this transition.

And when the cache replacement policy selects victim block which is in an exclusive location then this is the transition. And similarly, if the cache replacement policy selects victim block which is in a location shared, then this is the state transition. So, clearly we can see here, when there is a write miss transaction then we are actually going to exclusive state at the end. Whereas if there is a read request we are actually going to the shared state and now for this write request we place the write miss transaction and now we have to see what happens to this bus transaction because all the cache controllers will snoop on the bus.

And they now find that there is a write miss transaction on the bus. And now if they have the data they have to perform appropriate actions to their corresponding blocks. So, let us say core 3 is having this requested data in the exclusive state then first thing it has to do is it has to place this data on the bus by performing a write back block operation. And also it has to invalidate its copy because we are going perform a write operation and core 4 is now going to have the exclusive permission on this particular block. So, as a result even though core 3 is having this data and it has to invalidate it copy.

So, that is what is represented by this state transition from exclusive to invalid and because this is a dirty data it has to place this block on the bus. So, that the shared cache is going to update its copy and also the requesting core that is core 4 is also going to get the data. And now consider another scenario where only core 2 is having the data and that too in a shared state. And now what is going to happen for write miss transaction on the bus?

So, when there is a write miss transaction on the bus and now core 2 is going to invalidate its copy and without placing this data on the bus. As I mentioned earlier, for the shared blocks we do not have to write the data back to the lower level caches. So, we just simply invalidate. So, as a result at the end we can clearly see for a write operation, if any other caches have the data either in the shared state or an exclusive state they have to invalidate their copies.

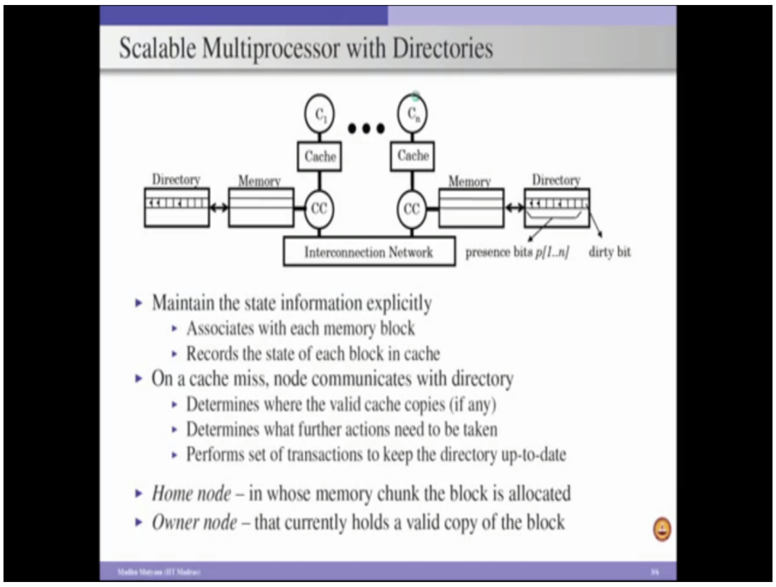
So, that the core which initiated this write request is going to get the exclusive permission on that particular block. And whereas, when we are performing a read operation then so the resulting state will be the shared state and even if any other cache has the data in the exclusive state, it has to change the state from exclusive to shared by placing the data on the bus. And now we will consider another scenario where core 4 is generating a read request and read is a hit in the local cache and this hit happens to a location where the state of that location is shared.

So, that indicates that core 4 is actually having the block of data which is in a shared state and core 4 generated a request to that particular block. So, as a result it is a read hit and once it is a read hit and because it is a read request we do not have to change the state from shared state. So, as a result there is a self-loop here. Now, what happens if core 4 generated read request, but now this time, this is a hit to a block which is in the exclusive state.

So, in the case of the exclusive also if there is a CPU read request hit, then we are not going to change the state of that location and we retain the state as it is. And similarly, even when there is a write request and this write request is a hit in a block whose state is the exclusive then also we are not going to change the state. Now, we will see what happens if there is a CPU write request which is a hit to a block which is in a location whose state is shared. Whenever we find a CPU write hit to a block which is in a state shared, then we have to change the state of that location from shared to exclusive because this is a write request.

As a result this requesting core needs to get the exclusive permission for that particular block. So, as a result it is going to place an invalidate signal on the bus. So, by looking at this invalidate signal, if any other caches have that data in the shared state, then they have to invalidate their copies. So, that is what is given by this. Now, let us say core 4 generated a write request which is a hit in its private cache and core 2 and core 3 also having this data in the shared state, then core 2 and core 3 have to invalidate their copies. So, that is what is indicated by this state transition. So, effectively, at the end of this write hit operation only core 4 is going to have the block in the exclusive state and core 2 and core 3 are going to invalidate their copies. So, this is about the 3 state write back invalidation protocol and it ensures that coherence is maintained across multiple private caches in our multicore system.

(Refer Time Slide: 19:28)



Scalable Multiprocessor with Directories

- Maintain the state information explicitly
  - Associates with each memory block
  - Records the state of each block in cache
- On a cache miss, node communicates with directory
  - Determines where the valid cache copies (if any)
  - Determines what further actions need to be taken
  - Performs set of transactions to keep the directory up-to-date
- *Home node* – in whose memory chunk the block is allocated
- *Owner node* – that currently holds a valid copy of the block

But this snoop based system will be realizable as long as the number of cores in our system is limited. Let us say if the number of cores is 4 or 8 we can go for snoop based system or a bus based system for our multicore system design but once the number of cores increase beyond 8, then it may not be a good idea to interconnect these number of cores by using a common bus. In such scenarios, we have to go for another type of communication mechanism to inter connect these cores and we have to use interconnection networks.

And this interconnection network can be ring type of topology, it can be a mesh based topology or torus topology or different type of topologies. So, as a result when we want to scale up the number of cores in our system, then we have to interconnect these cores by using some interconnection network. And also once we have more number of cores it is not a good idea to keep our shared cache in one particular location. And we have to distribute our shared cache or the shared memory across all these cores. We see here there are multiple cores and each core has its private cache and there is a cache controller and there is shared distributed memory across all these cores.

We can treat this as a shared distributed memory or also we can consider this as a shared distributed cache. So, here this memory is distributed and shared across all. So, as a result an application running on this core can share the data that is associated with this last core or similarly the application running on this core can share the data that is available with this memory and so on. And because this is shared and distributed memory so as a result at any point of time a given address will be uniquely identified only in one particular memory chunk. And now because we are distributing the memory across all the cores, now in order to maintain the cache coherence in this many core system what we have to do is we have to consider a directory for each of this memory chunk.

So, there is a one on one map between directory and the memory. So, that for each block in the memory there is an entry in the directory. And this entry in the directory is going to specify what are all the cores that are sharing the corresponding block of memory? And also this directory entry specifies whether this block is in a modified state or whether it is shared by multiple cores or it is not shared by any core and so on. So, that is what is given by this. We can clearly see here so there are presence bits. So, n bits are there and each bit is corresponding to each of the cores and also in addition to that there is one extra bit that is called as a dirty bit.

And this dirty bit is going to specify whether this block is dirty. And if it dirty then it is also going to specify which one is having this dirty block in their private caches. And in addition to this directory we also have the state transition diagram for each of the blocks that are present in the private caches. So, as a result when you are designing a directory based cache coherence protocol, we have to consider a directory for each of this shared memory. And we have to consider state transition diagram for each of the blocks present in the private caches. So, here in this directory based cache coherence protocols we maintain the state information for each of the blocks explicitly.

So, here that is what is mentioned. So, for each block in the memory we are maintaining the directory entry. So, we associate state information for each of the memory blocks, in this shared memory. And also we record the state of each of the blocks in the cache. So, this state is going to specify what is the state in which that particular block is present? Whether it is in modify or invalid or the shared state and so on if we are consider a 3 state invalidation based protocol.

So, whenever there is a cache miss from a particular note or a core, then the corresponding core is going to communicate this information with the home directory. So, that means for example, core n is generating a request and which finds a miss in the cache. This cache is the private to this core and now it sends the request to this cache controller and cache controller will look at the address of this request and accordingly it is going to identify the home node for this particular request. And it sends this request on the interconnection network and it reaches the home node. And once it goes to the home node then it determines what is the state of that particular block and so on.

So, that is what is given here. So, it determines whether where the valid block is available? and what are the further actions we have to take on this particular block? and also it performs a set of transactions to keep the directory upto date. For each operation whatever we are going to perform on cache blocks present in our, the private caches. So, in order to deal with this many core system here we use naming convention for each of the nodes and we call the node as a home node, a requesting node or owner node.

So, whenever some core generates a request and given this request address, we know in which memory chunk this particular block is allocated and that particular node is called as the home node. For example, when we send a request from this core and this request address is allocated in this memory chunk then we call c1 as the home node for this particular request and as I mentioned earlier because we are sharing and distributing the memory across all the cores.

As a result for each memory address we can identify only one memory chunk. We cannot replicate the same memory block in multiple memory chunks. So, we can uniquely identify a memory chunk for given address and wherever we find this particular address the corresponding core is called as the home node. And we also have owner node. Here the owner node indicates the node which actually holds the valid copy of the data. This valid copy can be like the dirty block or it can be a shared block and so on. For example, core 4 generated a read request and right now core 2 is actually having that particular block then core 2 is called as the owner node and core 4 is called as the requesting node.

A requesting node is nothing but the node which actually generates a request. Given these definitions now we will see how we design a directory based cache coherence protocol. As I mentioned earlier, so in the cache coherence protocol for scalable multicore processors, we need to consider the state transition diagrams associated with the directories as well as we need to consider state transition diagrams for each of the individual cache lines, that present in our private caches. And the directory is maintained at the shared cache or the shared memory and as a result we have to consider the state transition diagram for each entry for this directory separately. And because we are going to perform operations on our individual blocks present in the private cache, then we need to maintain a state transition diagram for each of the individual cache lines that are present in the private caches.

And this is a state transition diagram and right now here we are assuming in this directory based protocol also we are implementing the 3 state write back invalidation based cache coherency protocol. So, if we see this state transition diagram, this is similar to the state transition diagram whatever we discussed for snoop based systems. In both the things the state transitions and the transactions we place on the bus are same. The only difference is previously in the snoop based systems whenever there is a CPU read miss or CPU write miss, we are going to place a transaction that is a read miss transaction or write miss transaction on the bus, but now here in this scalable multicore systems.

So, we have interconnection network which is not a simple bus. So, as a result we have to send this read miss or write miss transactions as messages in our interconnection network. So, that is what is represented here. So, whenever there is a CPU read miss and if our cache replacement policy identifies a victim block which is in a location whose state is invalid, then we are going to send a transaction on to the interconnection network that is the send read miss message.

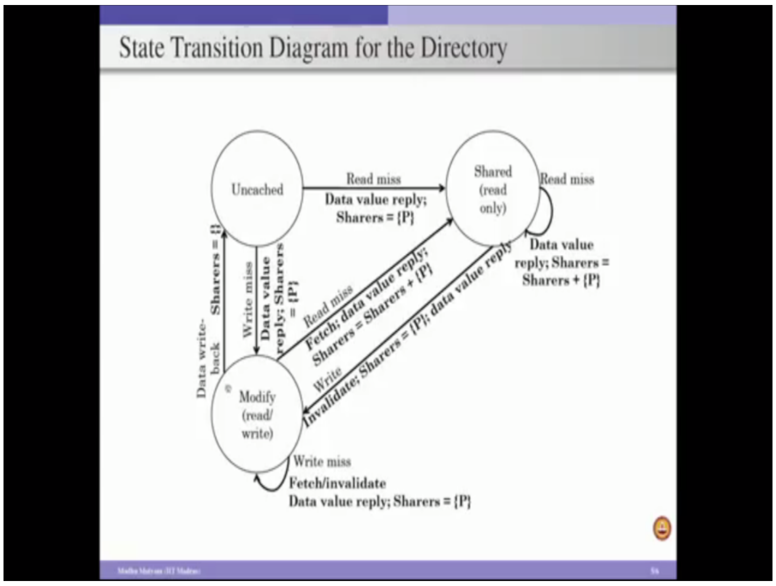
So, we are going to place a read miss message on the interconnection networks. So that this read miss message will go on the traverse on the interconnection network and finally, it reach the home node. And once it reaches the home node then based on our directory based protocol, we are going to see what operations are going to happen and so on and that we are going to discuss in the next foil. And similarly, if there is a write miss request and our replacement policy selects victim block which is in a location whose state is shared.

Then we are going to have a state transition from shared to modify and we are going to place write miss message transaction on the interconnection network. So, that this write miss message will be traversed on the interconnection network and it will reach its home node because for every memory address we have a unique memory chunk that is going to specify the details about the sharer’s information of that particular block.

So, as result whenever there is a message sent from any of the requesting cores, this message has to traverse on the interconnection network and it has to reach the home node. And the home node is the one which is going to specify what is the state of this particular block, whether this block is shared by some other cores or whether the block is owned by some other core and so on. So, the state transition diagram for the individual cache lines in our private cache is same as the transition diagram whatever we discussed for a snoop based cache coherence protocol.

The only difference is we have to come up with the state transition diagram for each of the directory entry, for each of the blocks that are present in our shared memory or shared cache. And that we are going to see now. Now, we are going to see the state transition diagram for each of the blocks that are present in our shared memory or shared cache.

(Refer Time Slide: 31:00)


State Transition Diagram for the Directory
Uncached
Shared (read only)
Modify (read/ write)
Read miss
Data value reply; Sharers = {P}
Read miss
Data value reply; Sharers = Sharers + {P}
Data write-back
Sharers = {}
Write miss
Data value reply; Sharers = {P}
Read miss
Fetch; data value reply; Sharers = Sharers + {P}
Write
Invalidate; Sharers = {P}; data value reply
Write miss
Fetch/invalidate
Data value reply; Sharers = {P}

So, once a read miss or a write miss message from the requesting core reaches the home node and home node will find the state of that particular block in one of these 3 states - either the block is not cached, the block is shared or the block is in a modified state. So, let us assume that core 1 sends a read miss transaction on the interconnection network and core 4 is the home node for this particular block and now core 4 receives that particular read miss message. And it identifies that this particular block is not cached anywhere. That means this particular block is not present in any of the private caches of our multicore system.

Now, what is the transaction we are going to perform? When there is a read miss from core 1, core 4 is now going to supply the data because the block state is uncached at this point of time. That means it is not in any of the private caches of our multicore systems. So, as a result it is the responsibility of the home node to supply the data. So, as a result core 4 is supplying the data by responding to this data value reply. And also it is going to update the sharer's information in the corresponding directory entry by placing the core 1 id.

Now, what happens if core 1 is generating a write request and which is a miss. So, there is a write miss message and now this write miss message is traverse on the interconnection network and it finally reached core 4 which is a home node. And core 4 identifies that this block is cached in any of the private caches associated with our multicore system. So, that means the state is uncached, but this is a write miss request now what is going to happen?

Whenever there is a write miss and the block is not cached anywhere earlier, now we have to change the state of that block from uncached to modify in the corresponding directory entry in the home node. And because this is a write miss now it is the home nodes responsibility to supply the data back to core 1. So, it is going to supply the data by using this data value reply. And also because now core 1 is going to have this data in its private cache, so we have to change the sharer's information and now sharer's information is equal to p or core 1 id.

So, this is about handling a read miss or a write miss from a requesting core at the home node and the state of this block is in the uncached state. Now, we will see what happens if there is a read miss or a write miss and if the block in the home node is in a shared state. Whenever there is a read miss from a core 1 and this core 1 sends the read miss message on the interconnection network and now core 4 receives this particular read miss message because core 4 is the home node for this particular request address. And now it identifies that this block is in the shared state.

So, now once it is in the shared state because it is a read request we are not going to change the state of this block. So, we will retain the state as it is, but because we have to supply the data to core 1 because the core 1 incurs a read miss. Now, this home node is going to supply the data by using this data value reply and also it is going to update the sharer's information. Previously let us say core 2 and core 3 are sharing this particular block now core 1 is also requesting this block for read operation.

So, as a result now we have to add core 1 also to the sharer's list. Effectively now after this operation now core 1, core 2, core 3 are sharers for this particular block and core 4 is the home node for this particular the block address. So, this is about a read miss transaction and the block is in the shared state in the home node. Now, what happens if there is a write miss? Whenever there is a write request, whether it is a write miss or a write hit, now what we have to do is this write transaction or write message will come on the interconnection network and it reaches the home node.

And now home node identifies that this block is in the shared state. And when it identifies that this is a write request automatically it has to change the state of this block from shared to exclusive or modified because this is a write request. So, as a result only the requesting core needs to have exclusive permission on that particular block.

So, as result invalidation signal will be sent to all the sharers because sharers information is already available with the home node in the corresponding directory entry. So, it is going to send invalidate signals to all the sharers. So, that they will invalidate their copies and at the end only the requesting core is going to have this data and that too in modify or exclusive state because all other sharers are invalidated their copies. So, as a result now the sharers list will be updated with only the requesting core id.

And also if this is a write miss request from the requesting core then automatically this home node has to supply the data back to the requesting node. That is done by the data value reply. So, this is about handling read miss and write request write hit or write miss to a block which is in a shared state in the home node. Now, we will see what happens when we have read miss or a write miss to a block which in the modified state. So, whenever there is a read miss from core 1. So, this core 1 is going to send this read miss message on the interconnection network. So, that the home node that is core 4 receives this message and core 4 identifies that the block is in the modified state.

And also it knows which core is actually owning this particular block in the modified state. So, as a result first it sends a request to the owner node to fetch the data that is by using this fetch operation. So, as a result now if core 3 is actually owning this particular block of data then core 3 has to send this data to the home node that is core 4. And after that core 4 is going to supply this data to the requesting core that is core 1 by using this data value reply.

And after that because this is a read request now we have to update the sharers information with the previous sharer as well as the new sharer. Previous sharer is actually core 3 which was having the data in a modified state, but because core 1 is also requesting data for read operation. So, as a result core 3 is going to change the state of this particular block from modified to the shared that will be reflected in the directory entry in the home node. Now, the home node is going to change the state of this block from modified to shared and also it adds core 3 and core 1 to this sharers list.

So, that is what is indicated by this. So, this is about read miss to a block modified state in the home node. Now, what happens if it is a write miss? If it is a write miss and the block in the home node is in a modified state, then we are not going to change the state of this block, we still retain the state modify as it is, but previously let us say core 3 is actually having this data in a modified state and now core 1 wants to write to that particular block.

So, as a result, first step we have to do is we have to get the data from core 3 to home node that is core 4 by using this fetch operation and after that core 3 has to invalidate its copy because core 1 is actually requesting this data for write operation. So, as a result we have to invalidate by sending this invalidate signal to core 3. So, that core 3 will invalidate its copy. And now core 1 is actually having this data in exclusive state. So, as a result we have to update the sharer's information as the sharers equal to the core 1 id and because core 1 is actually incurred a write miss.

So, we have to supply the data from the home node to the core 1. So, that is by using this data value reply. So, this is the whole process that is going to take place when we are dealing with the read miss, write miss for a block which is uncached or shared or modify in our home directory entries. And also we have to consider another scenario where let us say if core 3 is having a block of data in the modified state. When a block is in a modified state we know that only one cache can hold this particular block of data in a modified state and none other caches have the data.

Now, if core 3's cache replacement policy evicts this particular block now what is going to happen? Whenever core 3 evicts this modified block or the dirty block then we have to write this data to the home node. And once we write the data to the home node now this particular block is not cached by any of the caches in our multicore system. So, that is indicated by this particular state transition. So, core 3 previously having this block of data in the modified state, but there is a write back operation because the cache replacement policy associated with core 3 is actually evicting this block.

So, we have to write this data back and when we are performing this write back operation. So, as a result core 3 now is going to invalidate its copy. So, as a result we do not have this block in any of the private caches in our multicore system. So, effectively sharers list will be empty. So, that is what is indicated here and we have to change the state of this particular block in the home node from modify to the uncached state.

In addition to this the state transition diagram for each of the directory entries. For each of the cache blocks that are present in the shared cache of shared memory, we also have state transition diagrams for each of the blocks that present in our private caches. And that is a reason why when we are dealing with the scalable multicore systems and when we are designing a directory based cache coherence protocols. We have to design cache coherence protocol for each of the blocks that are present in the private caches associated with all our multicore systems.

As well as we have to come up with this state transition diagram for each of the blocks that are present in the shared memory or the shared cache. So, with that this directory based cache coherence protocol design is completed. And I am concluding this cache coherence protocol design module.

Thank you.