Parallel Computer Architecture
Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 09
Lecture - 48

Lec 48: Working of protocol

Hello everyone. We are doing module 6 on directory coherence case studies. This is lecture number 2 in which we are going to look at the working of the protocol of the SGI Origin architecture. Now what are we going to see in this lecture? We have seen that there is a node which has a hub and two processors. Through the hub it goes to the Xbow router and the router connects to other routers and other nodes, forming a 1024 node type size system. Right. So, when we have such a big size system where there could be multiple caches sharing various data items, they want to do a coherent read and write accesses to this data, the directory has to handle all of this.

It is a flat memory based protocol. Hence, the directory information is kept with the main memory associated with that particular node. We are going to especially see how a read miss is handled, how a write miss is handled, how a write back or a replacement is handled, in this lecture. In the next one we are going to look at the correctness issues.

Alright, so starting with the read miss. Going back to the cache we have a processor, it sends a request to the local cache. The cache incurs a read miss because the request is for read and the block is not with this cache. Now this read miss has to go to the main memory. In a normal single node system, the last level cache will send a request to the main memory, main memory responds with the block.

Whereas in a snooping coherence protocol it goes to the snooping coherence controller at the bus which broadcasts that request on the snoop and then eventually it is going to get the data. In a directory coherence protocol, the cache miss moves out from the processor or the last level cache and it goes to the hub because in my architecture, we have the hub node which is going to handle your directory coherence. So it goes to the hub. Now hub sees that address and it has to analyze whether this address is belonging to this particular node. That is, is the address range mapping to the memory bank of this particular node.

If it is, then it serves it locally. If it is not, it has to identify to which node in the network should it send this request. If it is a remote node, it makes a message and then

sends it through the network to the remote node. Now when it goes to the remote node, the hub at the remote node is going to take this message and then take particular actions. So all the actions which are happening at the home node, home node could be the local node or home node could be the remote node.

So these actions are going to be the same. Only thing is if the cache is remote, we have to send a network message. So what happens at the home node when we get the read miss? So we need to read the data from the memory and also see that if there are any other sharers or not because we need to update information in the bit vector. So we are going to do a parallel memory access as well as, also take actions related to the directory to save on the time. So start the directory search in parallel to the memory read.

If you recollect, we had a 16 bit vector and extended 48 bit directory bit vector as well as the data storage. So read the data, read the directory in parallel. Definitely the directory is going to finish quickly because it is a smaller memory compared to the big main memory. So going to do these two things in parallel. And once we do a directory search, we will come to know whether the block is in one of these states.

What could be the states? The block could be in a uncached that is unowned state. Nobody is sharing this block or the block could be exclusive with some node or the directory could be in a busy state. So we need to handle these three cases. One is either the block is unowned or it is shared. The block is exclusive with one particular node or the directory says that I am busy handling some other request.

We are going to see these three cases. We look at the first one where the directory state could be shared or unowned. If it is shared, that is there are already other sharers, we have the presence bit vector already built up. So we, the bit vector is existing and the directory state is S. So what we will do, we will go to the particular processor ID sending the read request and simply turn this bit to 1.

We do this. Once this is done, the data which we have read will be sent to the requester by turning this bit on and the action completes. So the speculative read, we call it speculative because when we read the data block, we are not sure whether this data will be used or not. Because in case there is a dirty copy in the system that copy will get utilized. So, in the current state where the state of the directory is shared or unowned, the speculative read is successful. In case the state is unowned, unowned there are no sharers. So what will we do? We need to either construct the bit vector or set up a pointer.

So our decision is we are going to use the pointer format and make the directory state as

exclusive because exactly one sharer is there and put a pointer to the requester. So this pointer points to the processor. The bit vector bits point to the nodes. Alright. So once we have done this, send a reply, that is send the data block to the requester using a strict request response manner because there is no intervention to be done in this case. So when a request comes, we are going to do a parallel search on the directory and the main memory.

So here the data comes out and that data is then sent to the requester. So this speculative data gets transferred always and it is up to the receiver or the requester to either use it or not depending on the situation. Okay. The next directory state is busy. If the directory state is busy, a request comes. Why is the directory busy? Because it is already servicing an existing request and to manage serialization, the directory says I will do one at a time and hence it goes into the busy state.

This is the easiest case to handle. Simply send a negative acknowledgement, so that the requester will then retry the request in future. The advantage of using NACKs is we don't need to hold up a lot of buffer space for all these pending requests. Okay, so the third case is when the directory is in exclusive state. Exclusive means there is a single sharer of this data block in the system.

The directory is not having the information whether this data block is clean or dirty, but because there is a sharer, it is a possibility that the block would be dirty and hence the directory has to possibly go to the current owner and fetch the dirty data block. So when a read miss reaches the directory which is in exclusive state, it cannot do a strict request response because it is not sure whether its own copy is clean. So it does reply forwarding. Here it will send a request to the current owner which is holding the block in exclusive state. This owner of the data block will then send the data to the requester, if the data was dirty and then it will also update the information at the directory.

So this is the reply forwarding type of information. So at the directory, that is the home node, when this request comes, should we change the state? That is the next question. Should I reply? Because the speculative data is anyway going to go? Should I update the bit vector? Should I change the directory state to busy? So all these questions we have to answer. Additionally, what happens at the owner, the exclusively holding block processor? So here the processor has to see whether the block is dirty or not. If it is dirty, it has to send the data.

If it is not dirty, should it send the data or not? So it says that anyway the directory has sent the speculative data, why should I send the data? So these decisions are to be taken. So we will see what origin strategy is happening. Okay. So in an exclusive state, we are

going to do a reply forwarding, where I send a request to the owner, owner replies directly to the requester, sends a revision to the home node and once all this is over, only then we change the state to shared because until then our state was exclusive. Okay. When a read request comes, okay, first become busy. Why busy? Because if future requests keep coming, we do not want to buffer them, hence they will be easily nacked if the state is busy.

So from an exclusive state, go to the busy state and we are not going into the shared state yet. Why? Because the data with the memory may or may not be the latest data. In case the exclusive sharer has already changed it. And we also cannot keep it E, because then other requests will be chasing this particular request. As I am doing reply forwarding, any request keeps going to the owner and if I remain in the state E, then we will end up forwarding several requests to the owner. Hence we cannot remain in E, we cannot go to S, hence we go into the busy state. Okay.

Once this is done, at the end of this complete reply forwarding, we will change the bit vector and send a speculative reply. So these are the actions which happen at the home node, which we discussed in the previous slide, do not change to shared, do not remain in exclusive, become in the busy state, add the new requester to the presence bit vector and then forward the request to the owner. What happens at the owner? At the owner when this intervention message reaches, it has to check whether the block with it, is clean or dirty. In case the block is dirty, it has to send the data to the requester. So it will send the data to the requester.

It has to send a revision message to the home node and in this revision message you are going to send the updated copy, so that the main memory gets updated. In case the block is clean, then we have to reply to the requester and to the home, but here we have to decide should we send the data or should we not send the data. So Origin architecture decides that I am not going to send the data. We will assume that the speculative data which was sent by the directory, it is the valid copy and let the requester use that copy. Okay. So we have listed the actions here.

If the block is dirty, that is the local state is modified, then we have to send the data reply to the requester and the revision message to the home. This data reply is going to overwrite the speculatively sent data, because speculative data may not reach early, but if it has already reached, it is going to be overwritten by the data reply from the owner. The revision message which goes to the home is called a sharing write back message because it is a write back, that is the latest copy is being written back to the main memory. At the same time, the owner is still going to hold the data block. So it is not writing back purely, it is giving the data plus keeping the data.

The second case is exclusive where the data is clean and hence, we simply send an acknowledgement to the requester, do not send the data, send a downgrade revision message to the home, do not send the data, so that the home switches from E to S state. Once all these actions are done, then the home will change from busy to shared. Because now we have two sharers, the original previous owner and the new requester. This information will be now kept in a bit vector format. So we will change from an exclusive pointer format to a bit vector format at the end of all these actions. Okay. So now the next question is about speculative replies.

We sent a speculative reply from the directory to the requester and the owner was also informed about the message. So the owner was supposed to take an action and owner has a data copy, so why not rely on the owner to give the data? Because the requester has to anyway wait for the owner's answer. See either the owner sends a data reply or it sends an act. So until this event or until this event is complete, one of these is complete, the requester cannot proceed. So it has to wait for this, so why not allow the owner to send the data and the directory can avoid that data transfer.

So for this there are multiple reasons. Definitely there is no latency saving in doing this because we are not either adding more latency or saving much because everything is happening in the background. But we are sending speculative reply because the way the processor is designed and how the protocol optimizations are going to help each other. The first reason is the processor design where this R10000 processor's cache controller, it is designed in such a way that it says, if my copy is clean I am not going to share that copy. Okay. So if the copy is clean it does not give the data and hence home has to send the data. If we had not sent speculative answer, what would happen? Once this owner would say that I am not giving the data then home has to later fetch the data and send this would add to the latency.

Hence it is always good to send it in advance. Okay. The second reason for speculative reply is that it is going to help in my protocol optimization for write backs. So when this owner, which is Pi it replaces this block which it had. It was Pi was the single owner, it had the data block. In case it decides to evict this block, it can evict it locally and need not inform the directory.

Now how does this help? Now Pi has already deleted the data block and memory still knows that Pi has the data because Pi did not inform that it had the data. So Pi had the data, the directory knows that Pi has the data because it is pointing to this. Pi deleted the data block and it would not inform the directory. It need not, because the block was anyway clean, the data was already up to date in the main memory. And this will not

affect the protocol because in case a new processor Pj goes to the directory asking for the data item, what would happen? It would get the speculative reply from the directory and intervention will happen with Pi.

Pi will simply ACK Pj because it did not have the data and accordingly then inform the directory about its eviction. So this information of block replacement can be informed to the directory later in time and not at the moment when the block is deleted. So this is how the protocol optimization forces us to send a speculative reply because that is always required. Because Pi the current owner may keep the block, may not keep the block and would not inform the directory. Next is the handling write miss. Write miss that happens when a processor wants to write the data.

It may have the block, it may not have the block. If it does not have the block then the request is called read exclusive. If it has the block, it is called a upgrade request. Both these requests will go to the hub or eventually to the directory controller of the home node.

Here the directory states could be multiple. One is it could be busy, the second could be unowned, that is there are no sharers, third is there could be multiple sharers and the fourth is exclusive. So we will have these four directory states which we have to take care of. The first one is very easy, if the directory state is busy then what to do? A request has come, simply NACK the request, it will be retried later in future. So the busy state is taken care of very easily.

We will do the next one, unowned. In the case of unowned, there are no sharers in the system. The current request, if it is a ReadX request, it is the first request, hence this will be the only sharer in the system, we can go for a pointer data format and make the directory state to exclusive. So, make the directory state from unowned to exclusive, write the pointer to the requester in the bitvector. Okay. Second case is, if the request, that is the write miss request is an upgrade request. This will come when the block was already with a processor in a read only manner.

It wants to now get permission for writing, so it sends an upgrade request to the directory. Would this happen? Because the current directory state is unowned. See this is, we have this P1 which had sent an upgrade to the block A because P1 already had this data block A. So it sent this data blocks upgrade message and right now, in the current timestamp the directory state is unowned which is indicating there is no sharer of this data block. Whereas we just now said that P1 sent an upgrade which means P1 had the data block.

So what is happening here? So, this is happening because we can say that this is a mismatching request, because the upgrade which P1 had sent probably, this is a very old message which got lost in the network and has right now reached the home node. So it got late into the network and when it reached the home node, home had already assumed that P1 has deleted the block. Consider the scenario that in the meanwhile, a request for some other road came to the home, there was another write request, probably that write request was satisfied, the block was replaced and eventually the effect was that this block with P1 was replaced by the home node for some of the other reason and when it is replaced, it removes that P1 from the bit vector. So this blue actions have already taken place and the pink action of upgrade block A, this message got lost somewhere in the network and in the meanwhile, all the blue part was over and now this request reaches to the home node. So this is why we call that upgrade request to a data block in unowned state is a mismatch because this is some older request which has reached late to the directory.

Why does it reach late? Because we do not assume point to point network, you can imagine the 1024 processor big network and the message can go through variety of routers, take different paths and probably take longer depending on the various scenarios. So we are not assuming a point to point data order or message order, hence messages can reach in any order. So now the block is unowned, means directory had already replaced that block from P1's cache. So what should we do? We have got a request which is of a wrong type and hence we send a NACK to this request, so that later it will be retried. So when this NACK goes back for an upgrade in future when P1 asks for the data, P1 already does not have the block, so it receives this NACK, it will say oh well a NACK has come for block A but I do not have the block A, I want to write to this block in future, whenever such a request comes it will send it as a read exclusive request for the future time.

Now we will consider when the directory state is shared, shared that is there are multiple owners, two or more sharers of this data block. In this scenario we have to send invalidations to all the sharers, we are going to use reply forwarding, where an intervention is sent to the sharer, sharers will directly send an acknowledgement to the requester. So an ACK message will go and another revise or revision message will go to the home node. So we use reply forwarding to reduce on the latency of the protocol, home has to send all the invalidations. How do I manage multiple sharers? If there was one sharer then P1 or this requester will know that it has to wait for one acknowledgement from one sharer. But right now the requester does not know how many sharers are there. Okay.

So how is this solved? So right now let us see the scenario, P1 sends a write request to

the home node, home has the list that is the bit vector of the sharers, so it sends an intervention message to all the sharers. In parallel it is going to send an acknowledgement, a speculative data to the requester and along with it, it is going to send a number of sharers because why is this number sent? So that P1 knows how many acts it has to receive. In parallel H is going to send invalidation messages to all the sharers, multiple sharers in the system, along with it, it is going to give the identity of the new requester. When these invalidations go to the sharers, they will invalidate the block, they will send a reply or acknowledgement to the home, in addition, they will send an acknowledgement to the requester. Because they know the identity of P1, they send ACK to P1 and to home. What should P1 do? P1 has to wait for all the acknowledgments, so you say S1's ACK has come, S2 ACK has come, 3 has come and so on.

So it knows the number of acts it has to wait for. So once it receives all the 3 or 5 acknowledgments, it can say that my request is complete. Okay. So we have listed the same thing here in a textual format. What happens at the home node? The home, currently if you see here, the home state was shared. So what should be the state now that a new request has come? Because a write request has come, there will be exactly one sharer, so the new state of the directory will become exclusive. So we first set this directory state to exclusive and then we have to put the pointer to the requester Pi.

So pardon me, we have to put the pointer and not the presence bit vector. So this will ensure that any future request will go to the new owner Pi. If the request was read exclusive, then we have to send the data and invalidations pending. So send the speculative data, send the number of sharers to the requester. The requester gets the data, the number of sharers and it waits for those many invalidation acknowledgments.

In case the request was upgrade, upgrade means we already had the data block, we want a writing permission. In this case, no data is sent, speculative data is not sent, only the invalidations list is sent to the requester. After this, the sharers will send the invalidations and ACK to the requester. What happens at the requester Pi? Pi gets the speculative data, if it comes from the directory and it waits for all the acknowledgments from the sharers. In the meanwhile, if a new requester Pj comes, okay, see this case here, we have the home, which initially was in shared state.

When Pi sent a write request, home went to the exclusive state and it is doing all the work in the background of invalidations. While this is happening, what is Pi doing? Pi is still waiting for all the invalidation acknowledgments. Once all these are done, then Pi's work will get over. This is the current scenario, Pi is still waiting. In the meanwhile, another node Pj sends a request to home for the same block.

Now what should happen? H is in exclusive state, recollect what we did for the exclusive state situation. We first go into the busy state and then forward the request to the owner. Who is the owner? Owner is Pi. So, H goes into the state busy and then forwards that intervention message to Pi. This actually means that, when Pj wants the data, home says Pi, you give the data to Pj.

But when this message reaches to Pi, should Pi take care of this request? You would say not yet, because we are still waiting for the previous request to get completed. So the request from Pj, when it reaches here, it will simply put it in a queue and it will, it will keep it pending. Once all the ACKs are done, only then the request for Pj will be handled by Pi. So here, why does it do this? Because we need to guarantee serialization. Home is serializing it properly by sending NACKs and sending correct interventions and the owner also has to guarantee serialization by saying that let me first finish my work, the pending request, only then we will start serving newer requests.

Now the last point, a new request, another say Pk comes. A new request Pk goes to the home, what will happen? Right now the home state is busy, so it will get NACKed and retried later. So there were multiple steps and multiple scenarios. So you can pause the video intermittently to check your understanding and probably again run through how the protocol is functioning. Okay, now our next directory state is exclusive.

Now directory state exclusive means that there is a single sharer. Now this sharer can immediately start writing to the data block without informing the directory, that is in the MESI protocol, it can go from the E to M state without telling the directory. So exclusive state of the directory means that the block with the sharer could be clean or it could be dirty. Okay, what is the request? The request could be either read exclusive or upgrade. If it is an upgrade request, will this request be a valid request? You can pause and think, directory is an exclusive state that is the node which is currently sharing the data block, this node has direct permission to write to the data block. So it will start writing without telling the directory and when it doesn't tell the directory, the directory state will remain exclusive and the data with the directory will be what it already had.

Okay, so if I am going to send an upgrade request to the directory and the directory state is exclusive, do you think this is a matching request or a mismatch request? We sent an upgrade because we had the data block. I have the data block and I go to the directory asking permission for writing and the directory state says I am exclusive. What does this mean that I had the data block? In exclusive mode, we could directly start writing without telling the directory. Now we have gone to the directory for permission, hence I would say that this is a mismatch request. Why mismatch? Because when this upgrade went, the one who sent the upgrade had the data block in shared state, not in exclusive

state.

It went there for asking permission for writing. Now the directory in the meanwhile has turned exclusive. So something else has happened in the middle because our upgrade request got lost in the or got delayed in the network. Okay. So let us see this nicely. P1 sent an upgrade to the block A.

Now this is going to take time through the network. Why did it send an upgrade? Because it had the block in the shared state. In the meanwhile, another P2 goes to the home node asking for read exclusive access. Okay. It says I want the block for writing. Effect of this is, home is going to remove P1 as a sharer because P1 was in S state, it was invalidated and P2 will be given the exclusive permission for writing to this data block.

Now this upgrade reaches to the home node. This is an old message which has reached late. Now because it reaches late, we have to simply NACK the request, because it is again a mismatch scenario. The upgrade request has reached late to the home node and in the meanwhile, home has already removed the data block from P1, given it to P2, hence home is in exclusive state. We have seen directory exclusive, request upgrade is a mismatch.

Next is the request being ReadEx. This is a valid request and valid because ReadEx means, the processor did not have the data block, it has sent a request. Directory is in exclusive state. So directory will go into the busy state, update the presence vector, send a speculative reply and then send an intervention to the owner, so that the owner can send the data to the requester.

That is the overall flow of the protocol. Now let us see it slowly in detail. P1 sends a write request to home node. This is not an upgrade request, this is a read exclusive request. Home sends a speculative reply to P1 and it sends an invalidation to the current owner with the identity of P1. The current owner has to now decide whether the data block is clean or dirty.

If it is clean, it will simply send acknowledgments without the data. In that case, the speculative data will get used. In case the owner has the block in dirty state, it has to send the data to the requester and also send a ownership transfer, that is the revision message to the home node. So this is what will happen. Once all this is complete, what is the new state of the directory? It will go from exclusive state to the shared state and now P1 and the previous owner will have their presence bits set in the bit vector. Okay. So at the owner, if the block is dirty, we send the ownership transfer revision message to the home.

We do not send the data to the home. Why are we not sending the data in this case? Because the new requester is anyway going to change the data. So don't send the data to the home. The data will get changed by P1 again. We send response with data to the requester.

We have to give data to the requester, don't give data to the home node. This data from the owner is going to override the speculative reply. So the speculative data will not be used, but this dirty data will get used, because the block was dirty with the requester. If the block was clean exclusive, we know that the owner is not going to give the data. Hence the speculative data will get used.

Owner simply sends ownership transfer revision message to the home. Again, it does not send the data because the data is up to date with the memory. Fine. So we have seen how write misses the handle. The next is how write backs are handled. Write back is when a node is having the data as a dirty copy, it wants to delete this block from its cache. So when it deletes the block, it has to send the data back to the main memory, update the directory status, and so on.

So very straightforward, but this particular action can have some raised conditions which we need to take care of. So we'll see them one by one. So easiest thing is that, data block is removed. So this is successful in sending the block to the main memory, that is the home node. Home node updates the data, deletes the presence bit vector of this particular processor, and the action finishes.

But now if a write back is going to the directory, and in the meanwhile, there is another processor which has sent a request for the same block to the directory node. So these two messages are happening together in the networks. Okay. So when a request comes to the directory and there is a dirty copy in the system, what does the directory do? Directory forwards that request to the node having the dirty copy. This node having the dirty copy has to send the data block to the new requestor, and also reply back to the home directory. In the current scenario, the owner, that is the node having the dirty copy, is already deleting the data block, so it has evicted it from its cache, and the data block is on its way to the home node.

In the meanwhile, somebody else has gone to the home node asking for the same data block, and home would have forwarded the intervention to the dirty node, so these messages have crossed each other in the network. So what should you do? Okay, so that is the scenario we have to handle. Let us do this write back request in more detail. I'm assuming that node x is holding the dirty copy to be written back. In the current scenario,

what are the possible directory states? Can the directory be unowned? You would say definitely not, there is already a dirty copy in the system.

Can the directory be shared? Again, yes, it cannot be shared because when there is a dirty copy in the system, there is exactly a single owner, so there can be multiple owners. So therefore, the directory state has to be exclusive, okay? In case the directory state was shared or unowned, and another request y would have come to the directory which will change the state to shared, would it happen? Currently, we have the directory state as E. If a new reader comes to the directory when it is in exclusive state, it would have forwarded this to the current owner and the state would have been busy. The state would never remain shared because we will, to handle this new request, the directory will go into the shared state only after finishing the complete transactions and not before that, okay? So these are the reasons why we do not prematurely change the state when a new request comes.

When a new request comes, we will only change the state after completing the operation. So in case the read request y would have come to this one, we would have forwarded it to the dirty node and the state of the directory would have been busy and never shared or unowned. So these states are never possible. The possible state is exclusive, because the directory state is exclusive as there's a single dirty copy in the system. What should we do? Update the data which is being written back, send an acknowledgement to the requester, that is the block doing the write back, send an ACK.

Now your new state will become unowned. From exclusive, we will go to the unowned case because there is no more sharer in the system. So this is a straightforward case. Next we will see when there is a additional request coming and possibly handle a raise condition. Okay, why does this happen? We have this node x which had the dirty copy.

This was the home node and we have a new requester y. This is a new node in the system who wants to do a read or write. So this request goes to the home node for that particular block. Home is in state exclusive. There's a single data copy.

Once this request 1 reaches, it will become busy and it will forward this as an intervention to x. Okay, so this intervention from I would just put a y here because this is getting forwarded on behalf of y. When this intervention reaches x, in a normal scenario, x would send the data to y and reply to home node. So this is what will happen.

But right now, x has already started deleting the block. So its write back request is already on its way. So the write back is coming in this direction, whereas the intervention is going in the up direction. Okay, so this is the scenario we want to tackle.

Would you say that I will NACK the write back, because right now, the directory state is busy and you get a write back request or a request from node x and in normal scenario, when the directory is busy, we NACK the request. But in case this request, if I send a NACK, it will not be retried because the block is already removed from the cache of x.

X doesn't have the block. It has already sent the block to the home node and this is the only valid copy of that data block. We cannot lose that copy. Okay. So what do we do in this case? Okay, in addition, H is also going to send a speculative reply to y with whatever data it already had. But we need to handle this case when an intervention is going towards the owner and the owner is writing back the data block.

So you can pause the video and think how you will handle this case. Okay. So for doing this, the request which made the directory state as busy and it gets forwarded to y as an intervention. So this node x, which is sending the data block back, we cannot NACK that request. Because they've crossed each other, that's the only copy. And the solution for this is, we are going to, okay, let me use this picture here. This written back data which has come to the home node, when the home was busy, is not going to be NACKed, but I'm going to use this particular data and give it to the new requester.

So we've already sent the speculative reply, but as a third step, we are going to send the data coming from our back to the new requester, because we cannot lose the single available copy of that data item. So we cannot drop the data coming back from x, otherwise our only valid copy will get lost. We cannot NACK that request because that block is already deleted. And if we NACK that request, then probably the speculative data which we have sent to y would be used, but that is an old copy.

Hence, we cannot even do this. We cannot drop the data, we cannot NACK the request. So the solution is, just take the data which x is sending and give it to y. Okay. So this race condition is solved by combining the two operations. So what does the home do? Home forwards the write back coming from x to y. So you simply forward the data to y and send an acknowledgement to x.

Tell x that yes, I have taken your data, whatever was the data, give it to y. Okay, you would say now that this to be intervention, it is on its way to R, what do we do with that? So what should R do? That is the dirty data holding previous processor. What should it do when the intervention reaches? This intervention is for a data block which the owner has already deleted. So owner says, I do not have the data block, I cannot handle your request, so it will simply NACK that request. Okay, so later when x receives the intervention on behalf of y, it will simply ignore it.

That is it may send the knack because it does not have the data block. So with respect to x, the operation finishes when it gets a write back acknowledgement from the home node. y finishes its work when it gets the data from the home node. At the home node, what should be the state of the data block? Initially it was exclusive, intermediately it became busy, now from busy it has to either become exclusive or to shared. It will become exclusive, if y had asked for writing and it will become shared, if y had asked for a read only copy.

So this is the solution. We are going to forward the data coming from the dirty to the home node. All right, now we are going to replace a shared data block. The previous one was replacing a dirty data block. When we delete a shared data block, that is the block is clean. And when the data block is clean and we give this information, in case we give the information to the directory, it is going to remove the particular node from the sharers list.

It may or may not send this message. That was a point to be noted because it's not compulsory to inform about deleting clean copies. But in case you inform, then the replacement hint will remove the node from the sharers list. Doing this is going to avoid future invalidation messages, but it is not going to reduce traffic. Because either we send a replacement hint now or you send an extra invalidation message later.

So overall, the number of messages in the system is going to be the same. So the Origin protocol says we are not going to send replacement hints. We will simply delete the block from our cache and not inform the home about it, okay? So we are not going to see more details of several of these transactions. But overall, we had nine request types, six invalidations and different types of responses. We've seen an overview, but every response will be at minute level of different kinds.

Sometimes you send data, sometimes only acknowledgements, sometimes ownership transfer, right? So variety of responses. So this is the total number of transaction types in the Origin protocol, okay? So with this, we've seen the working of the Origin directory coherence protocol. Thank you so much.