Parallel Computer Architecture
Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 07
Lecture - 40

Lec 40: Introduction to Directory Cache Coherence

Hello everyone. We are starting module 5 now. The topic of the module is scalable shared memory systems. This is lecture number 1, introduction to the directory cache coherence. Right, so quick recap that we started this subject with understanding what a parallel computer looks like and this slide shows you a very big general purpose parallel machine which is connected over a scalable interconnect and every node in the system could be a multiprocessor node or a single core processor connected through a network interface. Every node has a processor, its own cache, maybe some memory associated with it. Okay.

So this is how a general machine would look like. You have the interconnect and here I have shown three nodes, also the disk is connected, maybe more I/O devices and so on. Okay. And we have been learning snooping cache coherence up to the previous module where we discussed that all these nodes would snoop on the interconnect to understand what activities are going on, what read-writes are going on and whether each node should take any action. Okay. And a quick recap, we have been seeing these layers of abstraction where the applications sit at the top and then we have the different programming models, mainly multiprogramming, shared address space, message passing and data parallel.

All of them at the end run on a common hardware platform with the help of compiling libraries and operating system support. Right. So a quick recap to set the context of the topic and at the bottom you can see the same picture of a generic parallel architecture where we have the processor, a local cache, a slice of the shared global memory and a scalable interconnect. So this is the interconnect. And these memories which you can see, they are providing a global address space, that is it is not that only this processor can access this memory whereas it is actually available to all the modules. For example, if I have 100 addresses here, I will have 101 to 200 number of addresses here and so on. Okay.

So the addresses are distributed and globally you will get four times the capacity of this one particular module. Okay. So that is the architecture we are thinking about. Right. So

if you move even or zoom into the system where I have just shown four processor nodes in this picture and the interconnect is just shown as a small box. If we zoom into it, that interconnect would become a very big scalable network which would itself be made up of several switches. Right. It is not a single network or a single switch, it is made up of multiple switches. Each switch is connected to several nodes and so on. Right.

So several nodes are connected to this scalable interconnect. And what does this word scalable mean? It essentially says that all the transactions can happen simultaneously. Right. So because it is a network, you are not dependent on only one transaction happening on the bus like in a swooping based bus protocol, we had this bus and at a time a single transaction could flow and only then the next one could happen. Although we had split transaction bus which overlapped transactions, but at a time only one sender and receiver was able to use that bus as a resource. Whereas when you have a scalable network, you can have multiple senders and receivers using that interconnect. Okay.

So each node connected to this scalable network looks like this. It has a processor and here I have shown that there are multiple processors in this node. Every processor has got its own cache, then a slice of the distributed memory, again several banks could be placed here. Then you have CA which is the communication assist and the network interface, the black one. Okay, so every processor has caches and caches help us to replicate data, that is I can bring data items from another memory maybe in this node or maybe a memory from this node. Right.

So even this memory can be accessed by this particular node. So I can bring data from different memories and cache in my local cache here, that is helping me to naturally replicate the data. So this replication creates the problem of coherence and hence we need to address the coherence problem because the particular data item could be cached in multiple such nodes and multiple caches. Then we have this scalable distributed memory, scalable because as I add nodes we can continue to add memory modules and increase the capacity available to the complete system. Then we have the communication assist and the network interface.

These two together help us to execute all the network transactions and in fact are the foundation for executing a scalable coherence protocol. Why? Because they help us with an efficient node to network interface. Efficient because these processor nodes which you can see here, they can be considered as commodity devices. You just take any processor and you are able to plug this into a scalable network because of these two entities which help to transfer the request from this node onto the interconnect. Because, suppose this processor node was just looking like this, it was able to handle its own request connected to a single memory module and this could be connected over a bus or

a scalable network. Right.

So how do these two communicate or how does the processor communicate to another memory module? All these things are dependent on the interconnect because we want to keep a division between the capabilities of the processor IC and the overall network. So that I can use the same processor IC in different types of interconnects. Okay. So this communication assist and the network interface help to have a node to network interface. It is able to interpret the transactions which will flow through the network and it saves the processor from understanding the language of the network. So processor simply sends read /write requests and it is the job of the CA and the NI to translate it to the relevant protocol and send it outside to the network. Okay. So such a physical network which you are talking about can have 100 to 1000 nodes and every node can be a single processor or a multiprocessor node. But we still need to solve the cache coherence problem.

Here, you could send that as a, for example we were doing snooping in a bus based system. So you can do snoop so you can use messages instead of bus transaction because now we have a scalable network. Okay. So that is the scale of problem we are talking about and in this scale if you have been able to visualize how big the network is, we are expected to maintain cache coherence. Recollect the snooping based topic where we were discussing a bus with 10 to 20 processor nodes and they snoop each other and we had 4 different protocols which we discussed and we know how it is implemented in snooping. But now that whole idea has to be brought into this big size system. Right. So now we need to maintain cache coherence among this big system where you have several processor nodes connected to each other.

Now what does a cache coherence system actually demand? Right. It demands that when you are designing a coherent system, you need to know when should you invoke the coherence protocol, one. Then you should know what are the states of this particular block across all the caches. Then which are the locations of these copies, the third requirement and the fourth is we have to communicate with these copies and then do a necessary action either update them, invalidate them or whatever. So we have a set of actions. Okay, so when do I invoke a protocol? Whenever there is a access fault.

Access fault is there is a cache miss or a read or write comes and the cache doesn't have the block in the proper state. This is when we are going to invoke the cache coherence protocol. So invoking the protocol is same across snooping or a large scale system. But how do we implement the other points, the other three things, that is state of the block, how do I locate the copies and how do I communicate with those copies will decide the different varieties or designs which we will discuss. Right. So we are, to do cache

coherence, I need to do these four actions, zero is same for everybody whereas the different approaches are distinguished by A, B and C. Okay. Now A, B and C in a bus based cache coherence.

Now how is this happening? You can recollect that you have a bus which is a broadcast medium and the faulting processor sends a request on this bus. The nodes which are connected to this bus snoop onto the bus and the snooping implicitly tells them where the blocks are rather the nodes which snoop and they would know that they have the block and they need to take appropriate action. They can raise the wired-OR shared signal, the dirty signal and so on. So there is a procedure where the nodes which have the data come forward and implicitly tell that they have the information. Okay, so this is how locating the copies is implemented in a bus based snooping protocol.

Now if I want to implement the same idea of snooping in a big scalable network which we discussed, so how will you implement this idea in a big network? I will give you an analogy. Suppose you are speaking to somebody or showing a picture to somebody, a group of your friends. Suppose you have 5 friends and when you say a sentence, they all hear it or if you show a picture they are able to see that picture. Now you want the same thing to be done to 500 people or in a big classroom. So you are either not audible or the picture you are showing is not visible to the person sitting at the far off corner in the hall.

So how would you do that? Okay, so overall you have to either shout aloud or you have to broadcast a message because you cannot shout aloud beyond a limit because if I say 500 you are able to reach up to 500. If I make that 500 to 5000, you might not have systems or volume which can reach to everybody. So what do we do? We slowly transfer the information. So I will transfer the information to a set of people. From there it moves on and on.

So overall we will slowly broadcast the information from 5 friends to 500 and to 5000 people. Okay. So this broadcast if I do then everybody will be able to see or hear what you are going to communicate. Now same idea, in snooping, if I say that all these 5000 nodes want to snoop about what is happening, then you need to send this information through some way or some way of broadcasting to everybody. Then all these 5000 nodes will know about it, they will take appropriate actions and then act upon it. And then again the actions will have to be reverse broadcast to the originator of the message. Okay.

If you imagine the scenario it is working but not scalable and it is going to take a lot of time. So a bus based cache coherence though it is implementable, it is not practical in a large scale system. Right. So if I have a scalable network I can broadcast to all the

processors and wait for them to respond. This is simple as we said, Yes, it is simple to do, it is implementable but it does not scale with P. So if I have 5 friends then 5 become 500 that becomes 5000 you imagine the scale of problem which is increasing and it becomes more difficult.

So as your number of processors increase, I cannot very easily implement this broadcast idea. So bus does not scale and whenever I have a scalable network, every fault, every access fault would lead to P network transactions. So here I had 5 network transactions which will go up to 5000 network transactions which is not a practical idea. So how do I build scalable coherence is my objective. So overall cache state is same , that is either it is shared, invalid, modified. Okay.

So all those states remain same. The state transition diagrams also remain same according to the protocols we have been discussing but the difference lies in how do I manage the protocol. So protocol management will change the cache state and the state transition diagrams of the protocol will not change. Okay. Now same reasons again listed here, why snooping does not scale because snooping has limitations related to broadcasting because I am going to use lots of bus bandwidth. Every transaction again uses the controller gets involved for a longer time not only the bus but also the controllers at every node will be involved.

It is good for small to medium size systems and say up to 128 nodes, you can still manage snooping but beyond that it becomes difficult. So it will become a main bottleneck snooping that is broadcasting the information would become a impractical idea to implement beyond certain number of processes. So how do I overcome this bottleneck? I can do this if I get away with the idea of broadcasting or a logical bus. So I do not want to use broadcast. I do not want to assume a logical bus.

So how will you do this? I hope the problem is now clear that we do not want to broadcast. You still want to reach 5000 of your friends but you do not want to send a message which reaches all 5000 in a broadcast manner. So you will say that, well, if I cannot establish a connection with every one of these 5000 nodes can I establish connection with 10 of those and those 10 establish connection to other 10 and so on. So can I have a hierarchy of this snooping idea? Okay. So let me stick to the snooping idea and see if I can build a hierarchical snooping protocol and will that help me. So that is what we are going to discuss now.

So one approach is hierarchical snooping. Here we can say that let me talk to, say my 5 of my friends and so 5 at a time suppose and then these 5 speak to other 5 and so on. So I can have a ring where nodes are connected, another ring where nodes are connected.

Here I can use snooping within these ellipses at the leaf nodes and here again I have another bus. So all these circles are buses which work on the snooping protocol.

So I have a snooping protocol here, a snooping protocol here, another snooping protocol here and maybe bigger one here which is again connected to several such. So you can imagine the scale of this network or the tree which I am building called the hierarchical snooping idea. So now the processors are connected to buses or rings and each such ring could be a single processor or a multiprocessor which are the leaves of this whole structure. The memory can be centralized at the root. So if I say my memory sits here this is my root then every access will have to go all the way to the memory to read so there will be contention at that node or you can have the memory distributed with the leaf.

So this is good for close-by accesses but suppose this node wants to access it will have to go through the hierarchy to reach this particular node. Okay. So even the distributed idea would have challenges. This is memory access and how do you broadcast information? So how do you transfer or transmit information through this network? Right. So if this node wants to do a write, it will send a snoop onto this bus, then if any of these want to satisfy good, if they do not, then it will have to go here. From here again a snoop happens, if the snoop is satisfied by this level ,it is responded back, otherwise it either travels up. The other option is that suppose this particular node says I have the dat,a then it will go down here, do a local snoop at this level, get the information and then come back.

So it is you go up one level and then each level will decide whether to fetch data from the sibling nodes or go to the parent node. Okay. So you will have to propagate up and down the hierarchy based on the snoop results and most of the time if it satisfied here it is okay, otherwise you might have to go all the way to the root and then go to some remote node at the other end of the tree. Now we try to stick to the snooping idea, but you see that this problem becomes even more tough to manage. You are going to have higher latency traveling up and down the tree and even the local snoops in every node is going to take a lot of time and then there is going to be a bottleneck at the root node. Okay. So this idea is not popularly used today.Okay. So now we need to start discussing or thinking how do I build a scalable cache coherence protocol all these things I want to do in the hardware. Right.

I want to implement it in hardware and we know that the memory is physically distributed, it is not logically distributed, it is physically distributed and I cannot take the advantage of globally snoopable interconnect. We are not going to use the snooping idea where somehow through a tree of networks or hierarchy or any medium, I am able to

snoop. So this snooping idea is not provided. Okay. So we want to do it in hardware on a distributed memory which is physically very far off plus I do not have snoopable interconnect given to me. Now when I have a snooping interconnect what did we have? A state was implicitly told to us, that is a state of the cache block was known implicit because of the shared bus.

The shared bus helped to give this information, that is if I have these nodes and a bus, a write request goes from this node, that particular node says yes, I have the data and it takes appropriate action. So this processor P1 does not need to know whether P3 has the data or memory has the data. So it does not need to bother who will give it the data. It simply puts the request onto the bus and somebody else in the system takes care of the response. So that is why we say snooping becomes an implicit type of a protocol where we know the cache state is implicitly derived by the system.

The P1 does not need to know what is the state of the block across P2, P3 or memory. Whereas now that we do not have this snooping we have to make the cache state explicitly known to others. That is P1 should know, what is the cache state of the block and who has this particular block. Whether the P3 has the block, memory has the block or any other processor has the block or nobody has the block. Right. So P1 should have information about the cache block, if I do away with snooping.

So for scalable coherence we are going to use the idea of a directory. As the word says, directory is a one point lookup table or information or storage of information where you can go and check out what do you want to have. Right. Because now we are going to implement an explicit type of a protocol rather than an implicit type snooping protocol. So there is not going to be any broadcast because broadcast has its own problems. So we want to avoid broadcast. So how will I do this? So now I have a directory kept somewhere in this big network.

That directory has information about which processor has cached what type of data blocks. So when processor wants, suppose P1 wants to write to a data block, it needs permission to write to change its state from I to M or S to M. So it goes to the directory. The directory knows whether this particular block is cached by others in the whole network. Because it has the list of who all have cached this data and then the directory takes appropriate action. The directory will contact all the nodes who have cached this data.

It will tell those nodes to invalidate the block and then once it receives all the acknowledgments, it will then tell the new requester that now you are free to start writing to this data because I have invalidated others. So that's about writing. If a new

read happens in the system, the reading processor would start reading but then because it is now going to cache a shared data item, it has to record its presence with the directory. Because directory should know that, it is reading this data, hence this processor also goes to the directory tells that, now I have started reading. So directory makes a note that okay, now processor P1 is reading block B. So this directory is just a big list of all the memory blocks with the processors which are caching the data item. It also has the responsibility of contacting all these sharers and telling them about actions like invalidation, updates or whatever is demanded by the protocol.

So how do I implement this? We had a memory block. So now I have with every processor node we have processor cache and a memory. With every block I can associate the directory information. It keeps track of the copies that is which all nodes have cached that particular block and what is the state of that block, that is if three processors have it and they are in the shared state or a particular single processor has it in a dirty state and so on. So it needs to keep track of the states. Okay. Then the directory also sends messages to the cached copies and all these messages go as network transactions over that scalable network.

So I don't need a hierarchical snooping or anything. I need a good quality scalable network where the directory can send messages which reach very quickly to the desired destinations. Okay. So overall how does the directory function? Whenever there is a cache miss, the cache controller of that particular local node has to send a message to the directory. Now it is not implicit message that, you put a message onto the bus that I want to do a BusRd or a BusRdX. You just can't send those messages.

You have to send an explicit message to the directory. It has to be targeted to a destination which is directory. Directory determines what action has to be taken, whether to invalidate the sharers, if it is an update based protocol to send the updated data items to those sharers and so on. Then the directory also waits for the protocol actions to finish and then responds to the original requester. Suppose you send invalidations to three of them, whether they have been validated. In snooping we assumed that they invalidated though it took some time but before the next bus transaction on that particular data item, it was known that this node would have invalidated this data item.

But in a scalable network, we are not sure of this. Because there are multiple transactions happening parallely into this scalable network. Hence, we need to wait for the protocol to finish all its actions, only then the original request can be then restarted. That is the requester is told that, now all have invalidated you can start writing to the data block. Okay. So this is how directory would be associated.

We have the memory. Now I have shown this memory outside but it could be associated even here. Okay. You can just imagine this memory is similar and with this memory we have this directory associated with every memory block. I would say this is equal to one cache block size of data and with that a directory or a bit vector type of information is maintained. We will discuss this in detail. So directory is associated with the memory and it keeps track of the blocks across all the caches by using this bit vector.

When a cache miss occurs, here it will go to the directory. It will go like this to the directory controller. Directory controller knows information who all have the data block. It might send invalidations outside. Then it will receive acknowledgments, only then it will send an answer to the requester. So this is how overall the directory helps to communicate. Now in a scalable multiprocessor, we have memory modules distributed across nodes and every such distributed memory module has an associated directory structure.

In this picture we had a centralized memory with some limited number of nodes. Here you have a ,even larger system where every node has a slice of the memory. Okay. So are there other options to do that because should we do away with snooping altogether? So what are the other options. Because snooping does not scale. Can I have a hierarchy of snooping, that is, have some way of transmitting this information up and down the hierarchy? and we have discussed that having larger depths of the, tree that is traveling from leaf nodes to intermediate nodes and several intermediate nodes before we reach the root node. Right. If you recollect that diagram we had this, then we had this.

This would become very difficult to scale but that was not a bad idea. If I still want to implement that, I would stick to doing that only up to a two level hierarchy then it becomes manageable. If your depth of the tree is very high, the latency and the disadvantages are huge. But if I limit it to a two level hierarchy, I would still be able to use my old snooping idea for a big network. So two level hierarchy does work very well where individual nodes are now multiprocessor nodes, caches in this nodes are kept coherent by an inner protocol. So I have outside, two level means I have this and I have this.

So this two levels. So we have leaf nodes and the root,that is all. So that is node 1, node 2, node 3. Now these every node how does n1 look like? n1 is equal to several processor nodes again and each of these processor nodes have their own snooping. So within this, I have a inner protocol and outside in this, I have the outer protocol. So we have two level hierarchy, every level has its own protocol, one the inner protocol, the second is the outer protocol. For the outer protocol, how many nodes does it need to handle? Suppose each of them have got four processors, I have total 12 in the system but

this particular processor will only say, I have got three nodes, I need to handle three nodes because for it every leaf node is a single node. And once that information reaches to a leaf node it is then handled by the inner protocol to inform the internal four nodes. Okay.

So if I take a quick example, a the black one is the top level and the green ones are the bottom level. Right. So here I had a node 1, 2, 3. Similarly the green boxes are my nodes where each node has got say, four processors, their respective caches and so on. So I have multiple processors in every green box. Inside this green box is the inner protocol, around this black ring is the outer protocol. So we have two protocols. For this outer protocol there are only four nodes 1, 2, 3 sorry, five nodes. There are five nodes. It needs to know about and it only knows that these are the five node,s I have to handle. Each of these five nodes has other four nodes. For the inner protocol, there are only four nodes, outer one has only five nodes.

So overall I have lots of nodes in my system, but this is the demarcation. When the outer protocol talks with any of these nodes, it only communicates with the communication assist. It only talks with this controller and for the outer protocol, all these five caches do not matter. It only says that there is a single cache because it deals with a single node. So this is treated as a logical node. There is a logical single node connected and the communication assist or you can say there is a third level cache if you add. So final level cache or the communication assist is going to represent this cluster. So I need a representative which will talk on my behalf to the outer protocol. And it hides that how many processors are there within the small cluster. Okay. So now each node is represented by the communication assist or a tertiary cache to the outer protocol and it shields the inner information. It says that this is the cache, this is how I am going to do, and once the outer protocol says anything it is handled by the inner protocol for further actions. Now within the node, that is within this green box, you can either have a snooping protocol, if it is a small enough network or you can also have a directory based protocol.

But for doing all this, you need a good interface and the controllers have to be designed properly. Okay. So when I have two level, of course you have four options, both can be snooping, you can have the inner protocol snooping, the outer protocol snooping, one snooping, the other directory, right. So we have four options and depending on that your interconnect would change. If you have a directory then you need a scalable interconnect if you have snooping you can deal with a bus. So this is the picture of that same idea where the first one is showing snooping, snooping, that is, this is the outer protocol and this is the inner protocol. The outer protocol is a bus. It is handling snooping, this snoops with only say, four nodes and within this it will deal with two nodes because there are

two nodes one and two and there could be multiple nodes for B2. But B2 is not aware that every node has got multiple processes. Going to the second one, we have snooping then the directory, so inside I have a bus. So this is a snoop and outside it is a directory.

So here I have a directory protocol because I am using a network and when I say directory, here, through the network there will be a directory maintained and this directory structure will now keep information of the nodes and not the processes. Okay. So when I have two level hierarchy, always remember outer deals with only nodes as a single entity. Fine, so these are straightforward ,directory directory and directory snooping. Right. So advantages of using small machines to build larger ones. Now I, when I said two level hierarchy, we have small small clusters connected to form larger systems. So what are the advantages? Yes, you can have ready-made such clusters plugged into the system. Then per node cost reduces overall your bandwidth is also improved because if most of the requests are locally satisfied by the local nodes, that is local memory as well as the local caches. You might not have to go to the outer protocol and when a particular processor in the node fetches a data, suppose it comes from another remote node, from another leaf node say, now when you bring this data to your local node, the other processors in your node will readily get that data item.

So because it acts like a prefetching for the other nodes. The other advantages are that because I have a hierarchy and a node is sending a request up to the parent and the parent is sending the request down to the tree to another path, I can combine the requests and go up. Right. You don't need to send single requests but all the requests from a particular node can be combined and sent then sent to the node above. Okay. The advantages are potential cost and performance improves because the per node costs are amortized now, because I can have commodity SMPs connected to the system. Then the outer protocol, if it is a directory, it has to keep track of lesser number of nodes because every node is a cluster now and you can have the idea of prefetching data for blocks which get readily removed from a remote node to your local node. The other advantages are we can combine the requests when they are moving through the hierarchy.

We can even share caches, that is, within the node the caches can be used more effectively, if I have related data items, that is how do I map my working set to a particular node. So it is good idea that your application runs on a particular node, so that all the data items get housed in that node and hence you don't need to go out of your inner protocol more number of times. Then I can take benefit of the sharing pattern and the mapping of the data can be done nicely. If it is done nicely, this idea helps. Okay. And for applications which work with data items which are closely related this would help. But if there are applications which want to communicate with all nodes, definitely, it will have its own problems of scalability. But overall there are several advantages of

having smaller machines used to build larger ones.

Now what are the disadvantages? Disadvantages that the bandwidth gets shared among the nodes because every node, node, that is every processor node when it goes to the outer ring, the outer ring is shared by all of them. So the bandwidth sharing happens and when there is sharing, there is drop in the bandwidth available to every node. Then the bus within the node or outside any level of the protocol which uses a bus is going to increase the latency to the local memory. Because you have to go on the bus to the local memory node to fetch the data and then send it to the new requester. We have to normally wait for the local snoop to happen. You have this and here, a snoop protocol is there. Suppose a request has come inside you have to wait for the local snoop inside this ring to finish before you can send a response back to the outer.

So this is going to add to the delay. So this increases the delays because of the local snoops and if the sharing patterns are not properly mapped or the applications are such that they don't share data in a disciplined manner then you might have to go across one node to the other node several times, in adding up to the waiting time. So disadvantages are small advantages are more, so we can still go for a two level such hierarchy and build systems using smaller nodes. So what are we going to consider? We are going to consider directory protocols in this topic because we already know snooping directory across nodes. So every node could be a multiprocessor node, that node could be using a snooping or a directory. So we don't bother right now about it. We are only saying, I am going to study directory protocols for nodes and every node could be anything inside.

Then coherence is maintained across memory blocks. So there are two options. If I have a block B kept in this and this processor accesses that block B, should I bring the block B in the local cache or can I bring it in the memory, my local memory bank? Even this is possible.Bbut if I know this I need to maintain coherence across these memory banks because the memory will serve also as a global storage as well as housing the remote data items. Okay. Coherence can be maintained across the memory blocks, if I can bring the remote blocks to the local memory. Here, the caches can directly read. Now cache doesn't need to think where the B will come from.

For all the caches the data will always come from its local memory node. So there is no cache level coherence, but we will have to maintain a memory bank level coherence here. So there has to be a coherence across these memory modules. Okay. This is a possible idea but we are not going to consider this. We are only going to consider coherence across caches.

So for caches we are going to bring the data within the cache. So the block B, which is

sitting in this memory, will be brought to the cache of the processor and not brought to this memory bank. Okay. So B will come from that fourth memory module to the respective caches. The architecture is going to be cache coherent. Where is the memory? Memory is still remote, so non-uniform memory access.

So it is called a CC-NUMA. Cache coherent non-uniform memory access is the architecture we are going to deal with. All of these ideas are going to have a shared address space because all these four memory modules are going to give you a big physically distributed memory. It allows coherent replication of data across the caches and hence it is called distributed shared memory. The memory is distributed, memory is shared, still we are able to maintain coherence.

Hence it is distributed shared memory system. With this idea, now my serializing point, if you remember it was the bus and the snooping based protocol, now the directory becomes my new serialization point. Because all the processors first go to the directory, directory takes appropriate action and then replies. So the order in which I reach the directory will be the order of my serialization. Okay. So from the bus as a serializing point, we have moved to the directory as a serializing point.

So this was an overview of the directories. We will do more details in the future lectures. Thank you so much.