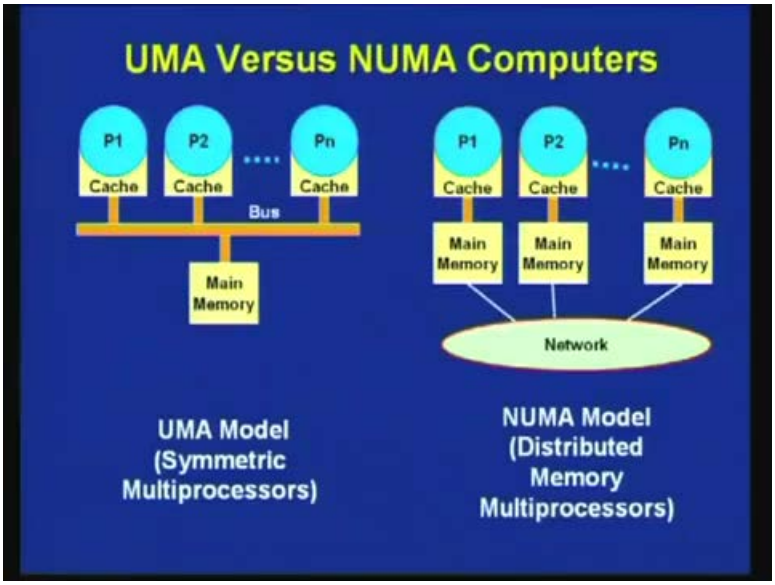
High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 39
Symmetric Multiprocessors

Hello and welcome today's lecture on Symmetric Multiprocessors. Actually, it should be read as symmetric shared memory multiprocessors, but shared memory is not mentioned although always, we use shared memory in a symmetric multiprocessors.

(Refer Slide Time: 00:55)



If you look at the different types of multiprocessor systems, they can be broadly categorized into two types. In the first case, we have a number of processors, along with their tri-fit cache memories and there is a shared main memory, through which the processors may exchange information. And this is known as symmetric multiprocessors or symmetric shared memory multiprocessors.

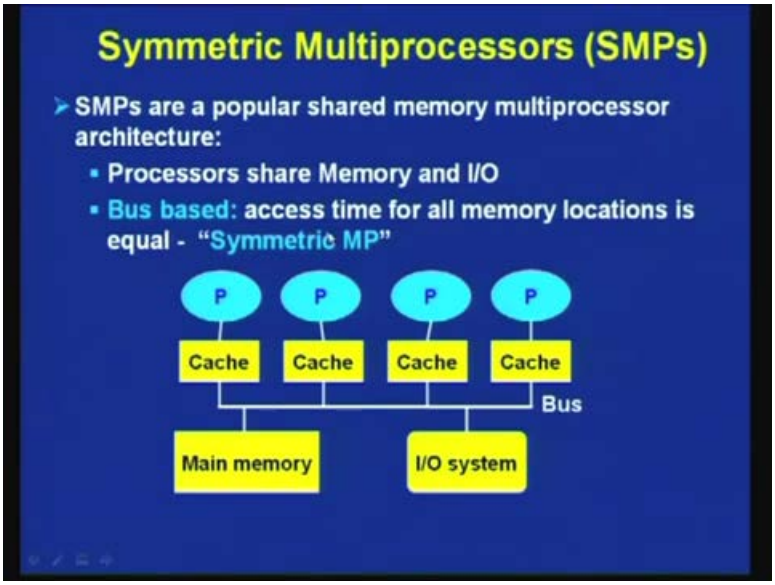
On the other hand you have another model, where you have got the processors with on chip cache memories. Then, they have their own private main memory and they are connected through a network. So, this is known as distributed memory multiprocessors or there are other names as well, the symmetric multiprocessors also called as uniform memory access model.

Because, for each of these processor the access time to the main memory is same it does not change, because they are connected through a bus. So, irrespective of which processor is accessing the main memory, the access time remains the same. On the other hand in your distributed memory multiprocessors, which is known as non-uniform memory access model.

Where as you can see, since each processor has it is own private main memory, whenever or in other words the main memory is distributed among all the processors. So, if a particular processor p 1 access it is own main memory, then the access time is smaller, compared to if it accesses another main memory which is attached to the processor p 2.

So, that is the reason why it is called non-uniform memory access model or also it is called distributed memory multiprocessors. So, we shall focus on symmetric multiprocessors based on UMA model in this lecture, various aspects of it various challenges, that is faced by this type of uniform memory access model.

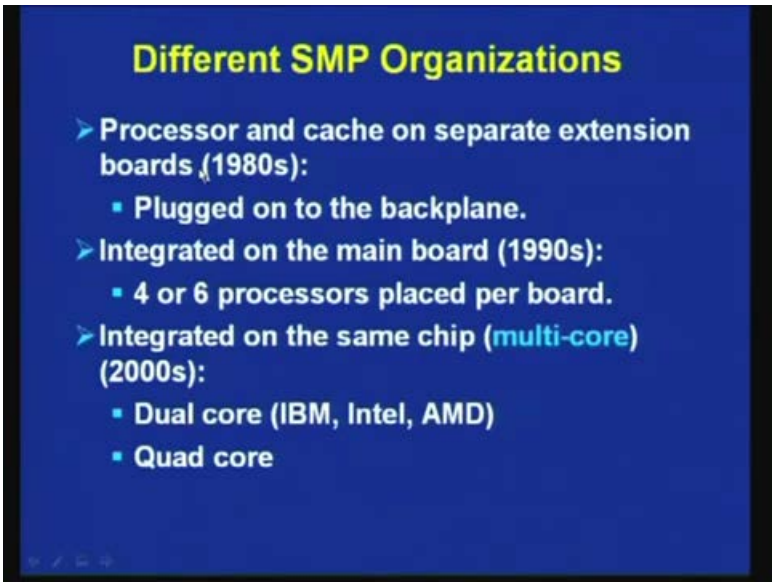
(Refer Slide Time: 03:13)



And these symmetric multiprocessors are very popular shared memory multiprocessor architecture. The processors share memory and also the input output devices, various input output devices, that are used in the system are also shared along with the main memory. And as I have already mentioned. We have omitted the shared bus apart from the symmetric processors, but usually all the access is through bus.

So, it is bus based access time for all memory locations is equal that is the reason the name symmetric multiprocessor.

(Refer Slide Time: 03:52)



So, over the years the multiprocessors symmetric multiprocessor systems have evolved as you can see in 1980 processors and cache memories were on separate extension boards. So, there was a backplane and the different cards along with the cache memory can be connected and build a complete system.

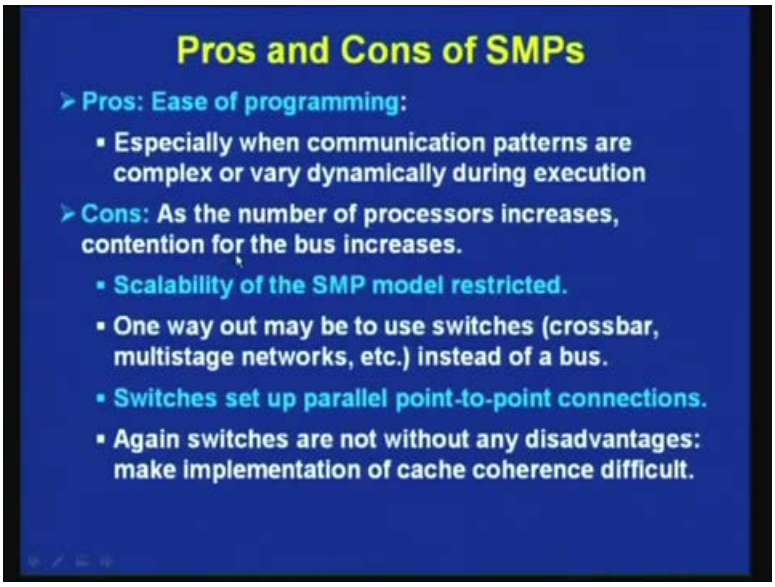
So, that was in 80's that means on a single printed circuit board, the processor and cache memories were put and they were plugged on the backplane, to link them together or that connected to the common bus. On the other hand in the 1990 they were integrated on the main board that means, all the processors were integrated on the main board and you can have 4 or 6 processors placed per board that means…

Of course, if you want to extend it beyond 4 or 6 processors then again you have to use the concept of backplane and multiple processors can be interfaced. And then in 2000 you know in this century integrated on this same chip that means, multi core. Now it is not on different boards, it is not on single board, but on a single chip.

So, single die or chip you can say, so core multiple processors are fabricated and we got dual core, quad core and the number of cores are increasing and processors multi core

processors have been manufactured by IBM, Intel, AMD. So, those who are cheap manufacturers all of them have joined the race of producing multi core processors.

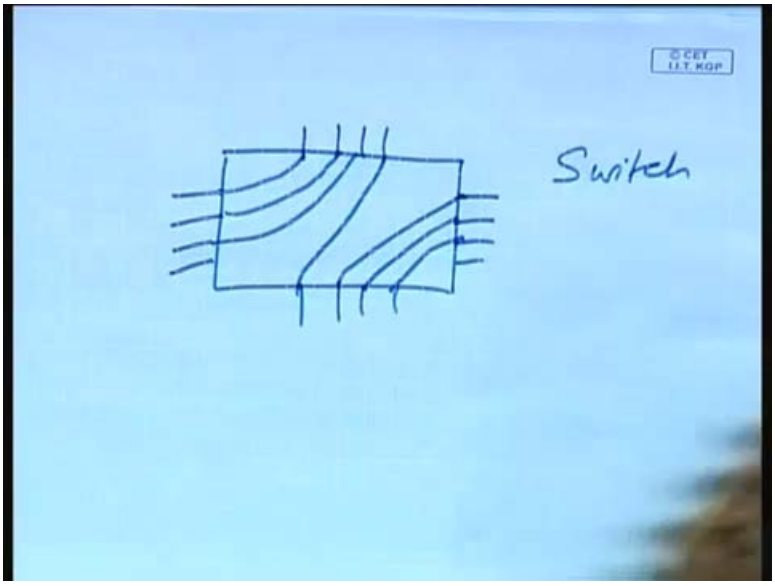
(Refer Slide Time: 05:50)



Now, let us look at the advantages and disadvantages of symmetric multiprocessor systems that means, the main advantage is ease of programming. Especially when communication patterns are complex or vary dynamically or vary dynamically during execution. So, in such a situation the programming is relatively easy in symmetric multiprocessors of course, there are many other disadvantages as you can see as the number of processors increases the contention on the bus increases.

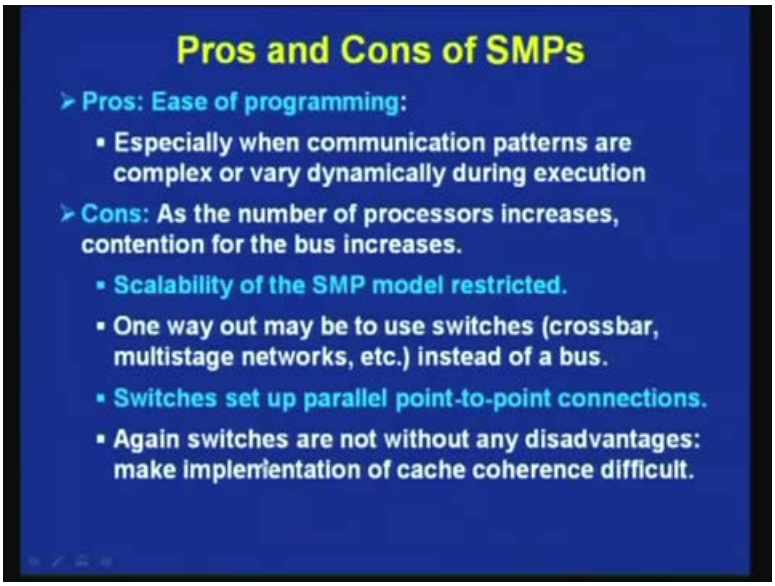
So, since all the exchange of information takes place through a bus shared bus obviously the traffic on the bus will keep on increasing as you keep on adding the number of processors on the bus. And as a consequence scalability of these symmetric multiprocessor model is restricted you cannot have large number of processors in a single system. Of course, you can try to overcome the problem, by removing the bus with the help of the switches. So, one way to maybe use switches; switch means you can have a switch.

(Refer Slide Time: 07:03)



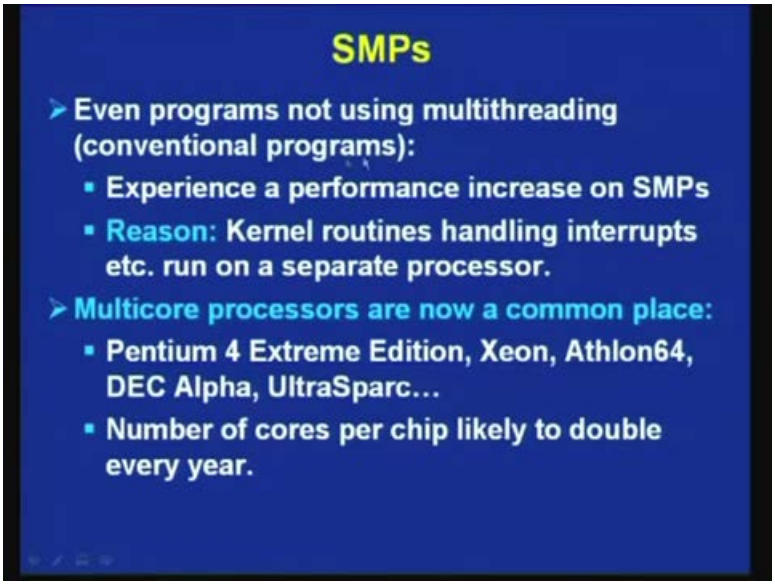
See, 16 port switch, you can have 4, 16 port switch, in this case what can be done any pair of processors can be connected in this switch and parallel exchange of information can be done. So, this is a switch or you can have multi stage interconnection network that is also possible instead of a bus, however whenever you do this. Switches set up parallel point to point connections, as I have shown in this diagram. So we have got point-to-point parallel connections and that complicates the situation.

(Refer Slide Time: 07:50)



So, because if you have got a shared memory and whenever you have got parallel point to point connection, then this will lead to other problems, so again switches are not without any disadvantages make implementation of cache coherence difficult. As we shall see even on a simple dos-based system cache coherence problem is difficult to implement it can be implemented but, it is quite complex. And whenever you go for either a switch or a multistage interconnection network then, it adds to the complexity and obviously, it is disadvantageous.

(Refer Slide Time: 08:28)

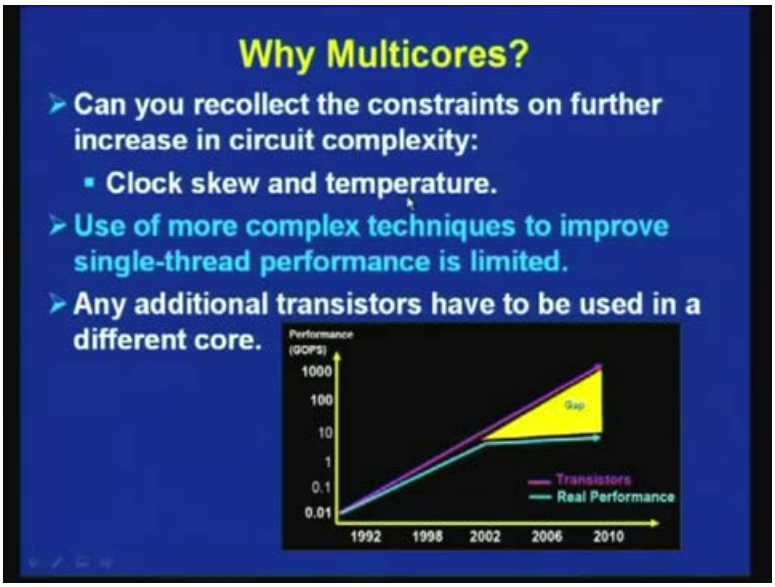


So, in programs not using multithreading that if you go for single threading even then, experience a performance increase on symmetric multiprocessors. That means, even if you do not use multithreading but, go for multiprocessor systems then there will be performance increase, the reason for that is kernel routines handling interrupts etc run on separate processors.

So, what is happening here application program will run on one processor and kernel operating system routines will run on another processor. So, that way performance is improve. So, multi core processors are now a commonplace as I have already told Pentium 4 extreme edition Xeon, Athlon64, DEC alpha, Ultra SPARC. So, for all these processors we have got multi core versions and it has been I mean, there is a forecast that the number of cores per chip is likely to double every year.

So, again we are trying to I mean increase the number of processors for every year; however, programming is a big constraint. So, the forecast which has been told may not be successful that means, the number of cores may not increase twice per year may not double every year, but in any case it will keep on increasing.

(Refer Slide Time: 09:58)

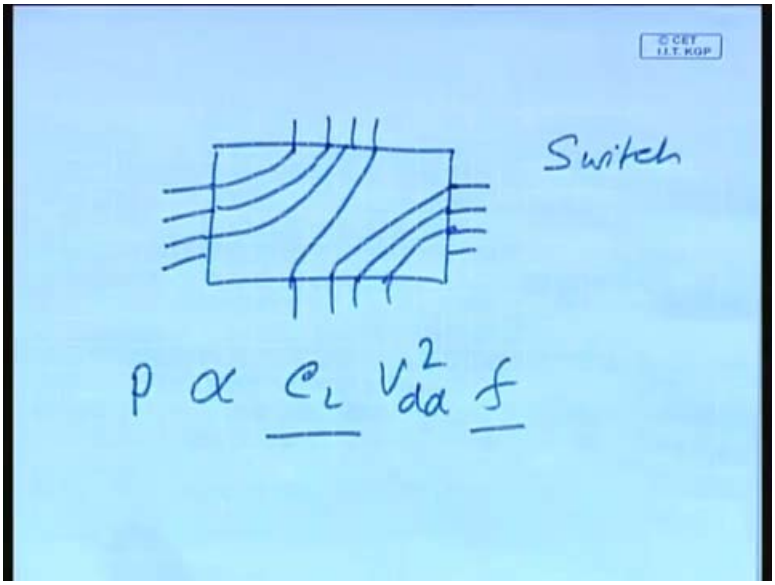


Now, question naturally arises, why you have gone for multi core instead of increasing the complexity of a single processor, putting more and more transistors, making it more and more powerful, increasing the clock frequency and so on. So, that the performance is increased, as we know performances related to clock frequency, performance is related to complex functions, that you put in the chip and so on.

So, why not, why we are going for multi core instead of trying to improve the performance, the reason for that is clock skew and temperature. So, as you increase the clock frequency there is a problem on known clock skew that means different points of a chip. I mean circuit will receive the clock at different instance of time and that is known as clock skew.

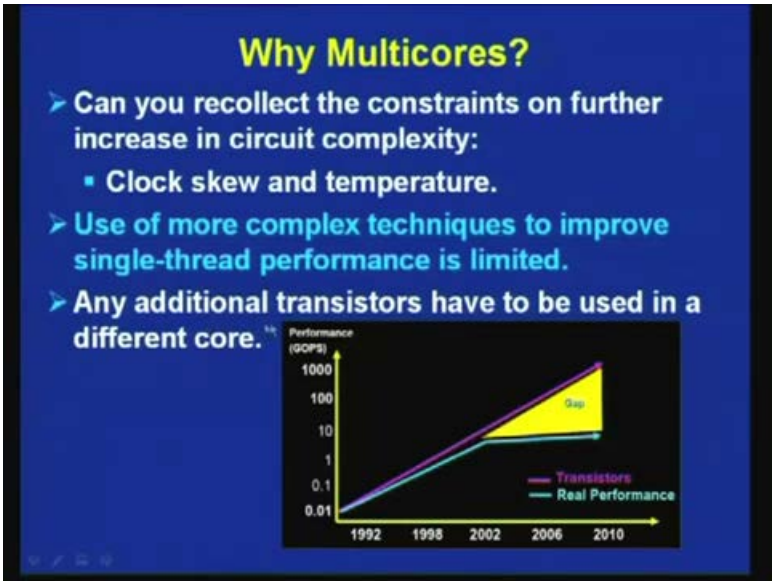
Because, of the delay in different parts of the circuits clock will not reach at the same instant and that is known as clock skew problem. And the temperature because power dissipation keeps on increasing as the frequency increases as you put on more and more number of transistors.

(Refer Slide Time: 11:18)



As we know the power dissipation is proportional to c l v d d square f; that means, c l the capacitance of the circuit will keep on increase, as you increase the number of transistors and also as you increase the frequency power dissipation will increase. So, both the factors will keep on increasing, if you keep increasing, if you increase the number of transistors.

(Refer Slide Time: 11:42)

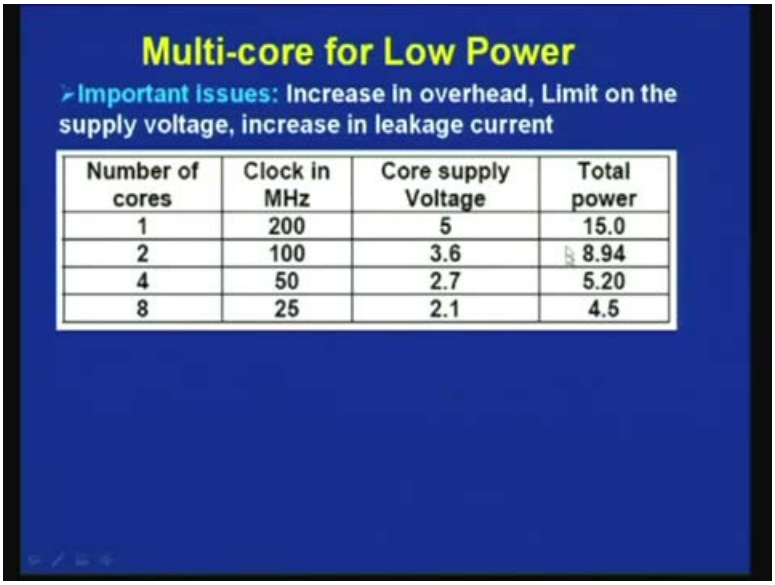


So instead of that, I mean whenever this happens then you can see any additional transistors have to be used in different core. So, use of more complex techniques to

improve single thread performance is very limited and as a consequence as you can see as you put more and more number of transistors.

The number of transistor you may keep on increasing linearly, but the performance will not increase linearly. And it has been found that beyond 2002 the performance gap, I mean is increasing real performance with respect to the number of transistors is keeping on increasing.

(Refer Slide Time: 12:23)



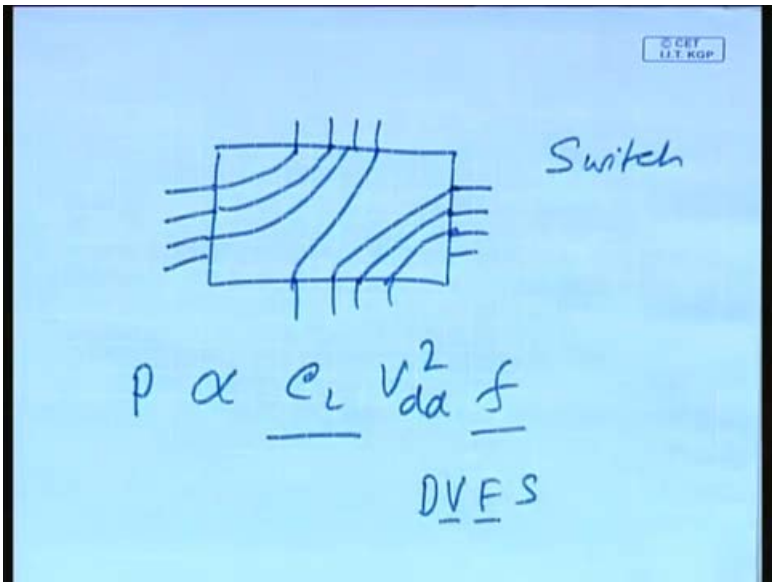
If you look at this particular table we find that instead of increasing the number of I mean increasing the clock frequency of a single core. If you reduce increase the number of cores and reduce the clock frequency to maintain the same performance why here we are comparing for the same performance. So, you can keep on increasing the cores assuming that the task is evenly distributed in all cores.

So, parallel computation will take place and to achieve the same performance we can reduce the clock frequency correspondingly. And so, when you have got 1 core clock frequency is 200 mega hertz you got 2 cores hundred megahertz and so on. So, in this way you can as the clock frequencies reduced.

It is possible to reduce the supply voltage as well because, it has been found that in a v l s I circuit that frequency delay is related to supply voltage. So, if you reduce the frequency

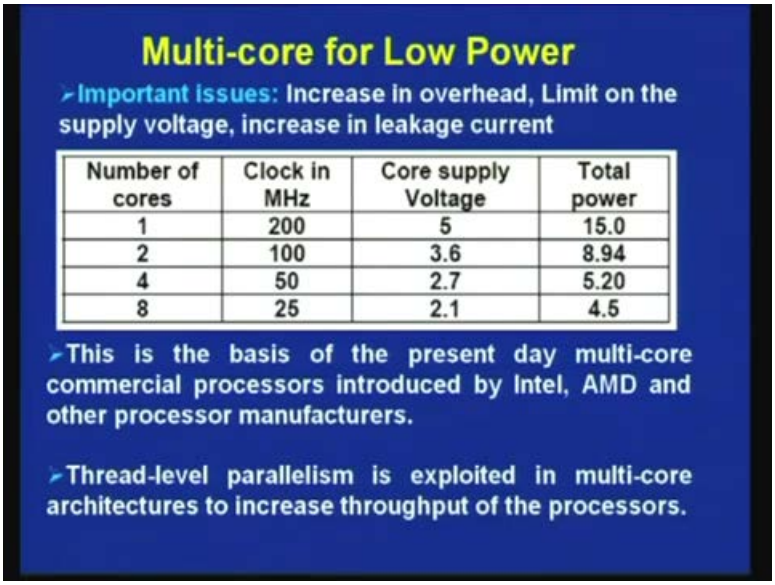
it is possible to operate the circuit at a lower voltage. So, that has led to the concept of Dynamic Voltage and Frequency Scaling D V F S.

(Refer Slide Time: 13:41)



So, you can dynamically reduced voltage and frequency I mean in some cases.

(Refer Slide Time: 13:48)

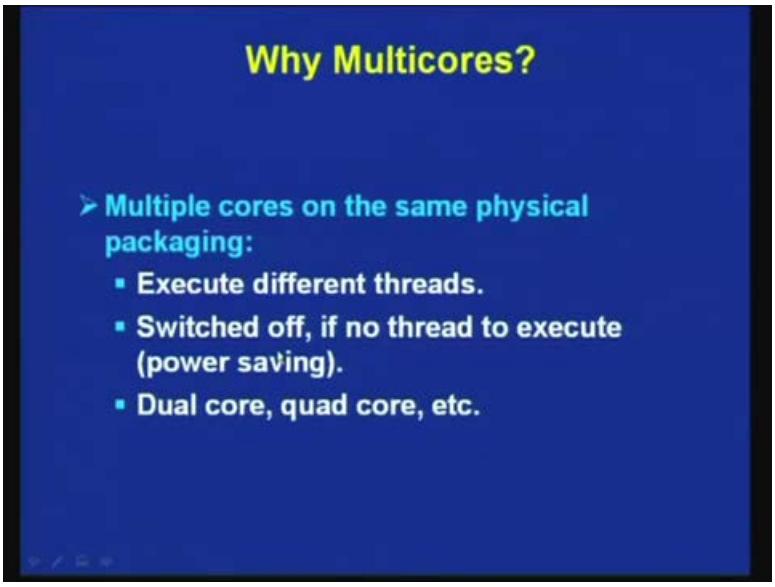


But, in this particular case we are reducing the frequency, we are reducing the voltage and as we can see for same performance. The power dissipation can reduce if you go from 1 core to 8 core. So, this is this is you can say an extreme case in ideal situation it

will be a little less. But, definitely performance you will get lesser power dissipation same performance, if you go for multi-core circuit.

So, this is the basis for present-day multi-core commercial processors introduced by Intel, AMD and so on and another thing is thread level parallelism exploited in multi-core architecture to increase the throughput of the processors. We have already discussed that at length in my last lecture. How you can use different types of thread level parallelism in a multi-core system.

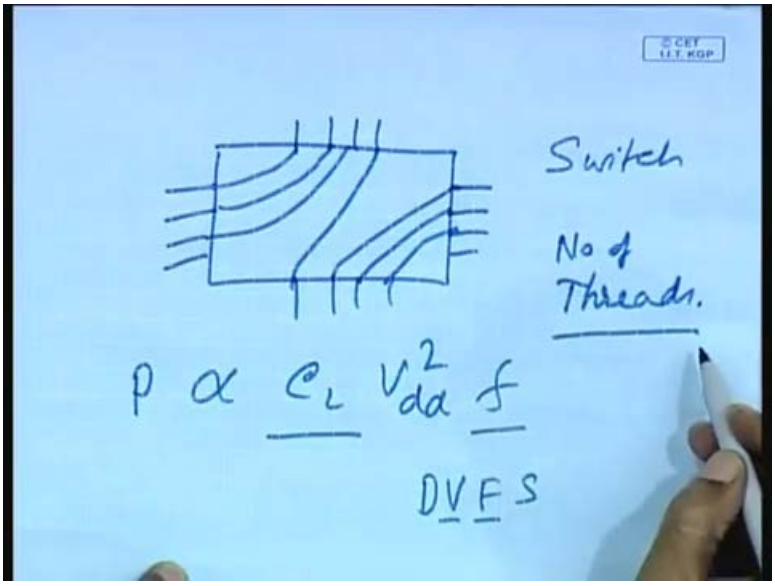
(Refer Slide Time: 14:43)



Now, you can say that whenever you are having multiple threads. So, multiple cores on the same physical packaging, what you can do you can execute different threads on different processors. Now suppose the number of threads reduces. Because, in application we will decide how many threads a particular application can have rather an application can be considered as a process.

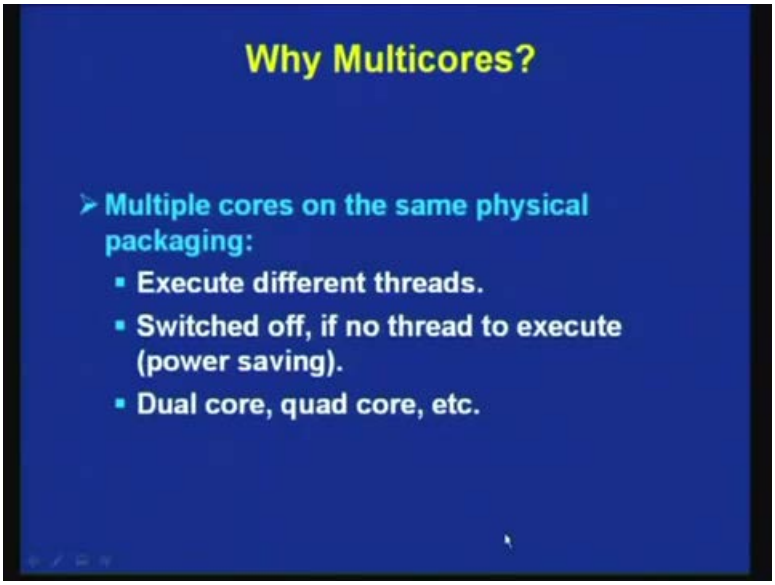
A process will have some number of threads and the number of threads cannot be increased. Beyond certain limit it depends on the process. So, what you can do whenever the number of thread is less you can switch off some of the cores. So, if no thread to execute that means, what you can do you can do various things like depending on.

(Refer Slide Time: 15:43)



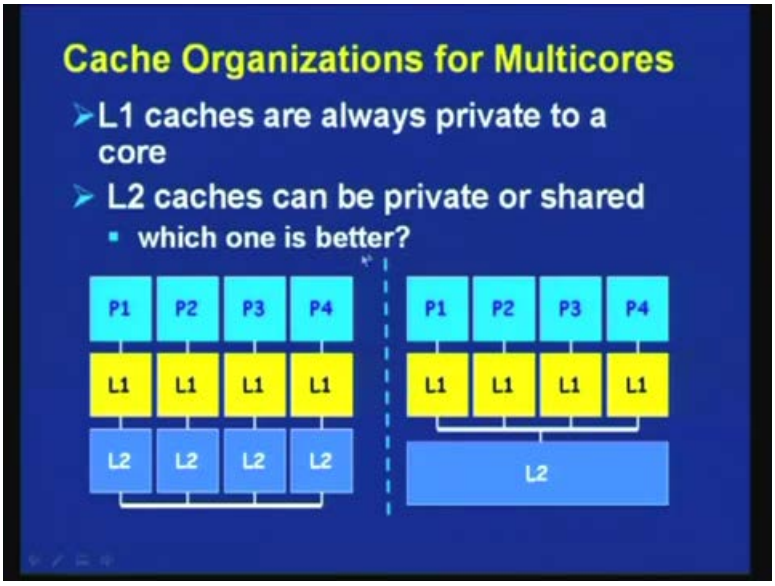
Number of threads you can keep number of cores active. So, number of threads will decide. How many cores will remain active and not only that another alternative is you can do kind of load balancing. You can do thread level, thread switching from one core to the another core and in this way you can evenly distribute the power dissipation of different cores these are all possible.

(Refer Slide Time: 16:16)



And as i have mentioned the number of cores is increasing

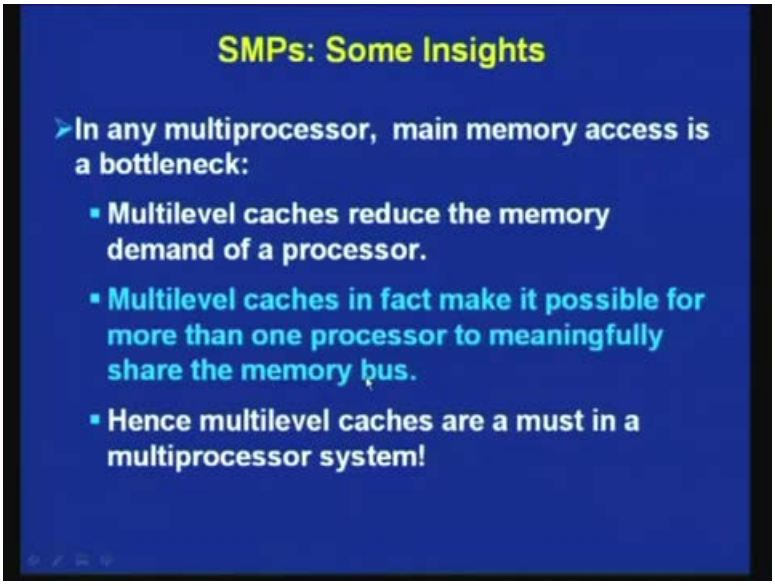
(Refer Slide Time: 16:21)



Now, let us focus on the cache organizations for multi cores. So, what kind of cache memory is suitable in a multi core architecture. So, one possibility is that you can have two levels of cache memories L 1 cache is available with the core. So, you can say I mean it is private to each of the processors L 1 cache memory and you can also have L 2 cache memory and for each of them you can have a private L 2 cache memory and then of course, you can have a bus which is connected to the main memory.

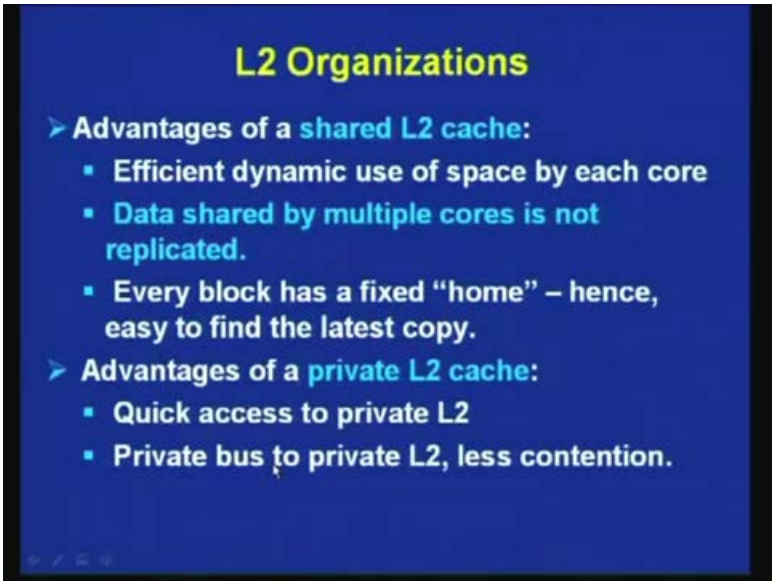
This is one possibility another possibility is that each of the processors can have its own private L 1 cache memory L 2 cache memory can be shared. So, L 2 cache can be private or shared now question arises in multi core situation. where you're doing symmetric multiprocessing what type of configuration is more suitable.

(Refer Slide Time: 17:32)



It has been found that, in any multiprocessor main memory access is a bottleneck. So, that is the reason why multilevel cache reduces the memory demand of a processor. So, multilevel cache memory is proposed in case of processors however, when you go for multilevel cache memory. In fact, makes it possible for more than one person to meaningfully share the memory bus.

(Refer Slide Time: 18:05)



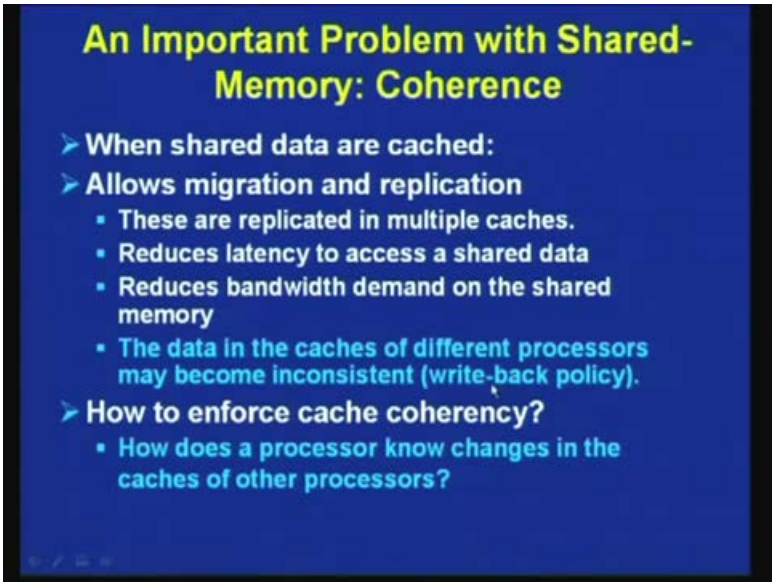
So, multilevel cache can be used and not only that you can you can go for a shared L2 cache. So, it has been found that instead of having private L 2cache. It is profitable to

have shared L 2 cache in a multi core architecture, the reason for that is efficient dynamic use of space by each core data shared by multiple cores is not replicated that means, whenever it is in the L 2 cache then it is not necessary to replicate it on earlier we have seen.

If data is shared and L 2 cache is private then it has to be replicated in each of the L 2 cache memories, but if it is shared it is not necessary to replicate and as a consequence it will save space and every block has a fixed home. Hence is easy to find latest copy very easy to find latest copy in other words cache memory management becomes easier.

So, these are the advantages of private L 2 cache of course, there are some advantages of private L 2 cache memory number one is quick access to private L 2, private bus to private L 2 less contention. So, these are there, but it has been found that shared L 2 cache is more preferable compared to private L 2 caches although it has got some advantages.

(Refer Slide Time: 19:33)



Now, we shall come to the main problem that we encounter in the shared memory multiprocessor systems. So, that known as that is known as coherence problem. So, when shared data are cached. So, a data which is being used by multiple processors, but if it is brought from the main memory to the cache then what problem arises. So, allows migration and replication they are replicated in multiple cache memories and reduces

latency to access a shared data that means whenever it is transferred to the cache memory and access obviously.

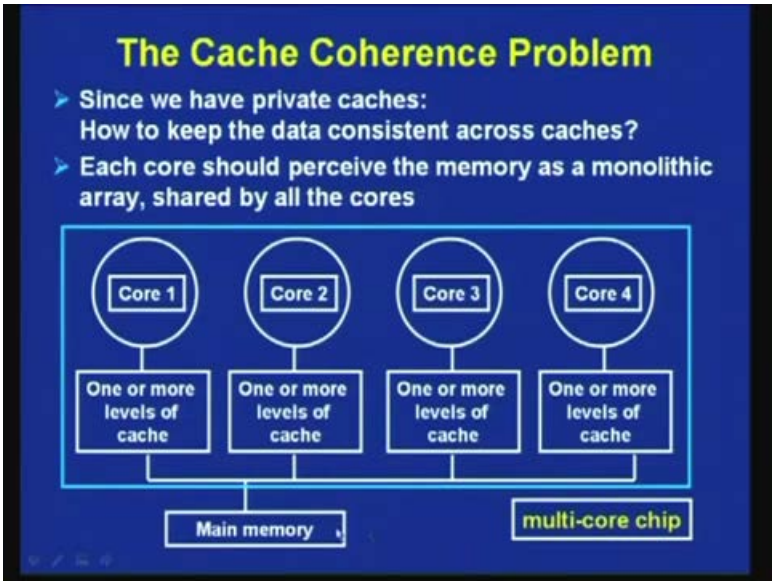
The latency will be small and the fast access can be done reduces bandwidth demand shared memory will also reduce. The traffic on the main memory. So, this is because of this reasons it is necessary to transfer for the shared data to the cache memory. So, data in the cache of different processors may become inconsistent. So, whenever you use write back policy we know that whenever data is modified in the cache it is not modified in the main memory.

So, when in that cache what happens a particular processor if shared data has been transferred main memory to the cache memory and if it modifies it then it will lead to problem. Because, in the main memory it is not updated and of course, if you use write through policy then it would be updated, but it is not done in case of a write back policy. So, you have to explicitly write in the main memory whenever you use write through write back policy. So, how to enforce cache coherency.

So, how does the processor know changes in the caches of the other processors that means, what is happening I shall show you multiple architecture. Where each processor has its own cache memories.

Now in the main memory a data has been transferred to the cache memory and it has modified and that data and main memory data is different and that is actually known as cache coherence problem and then if the data from the main memory is read by another processor obviously. It will not get the recent data so that means, it leads to inconsistency that is known as cache coherency problem.

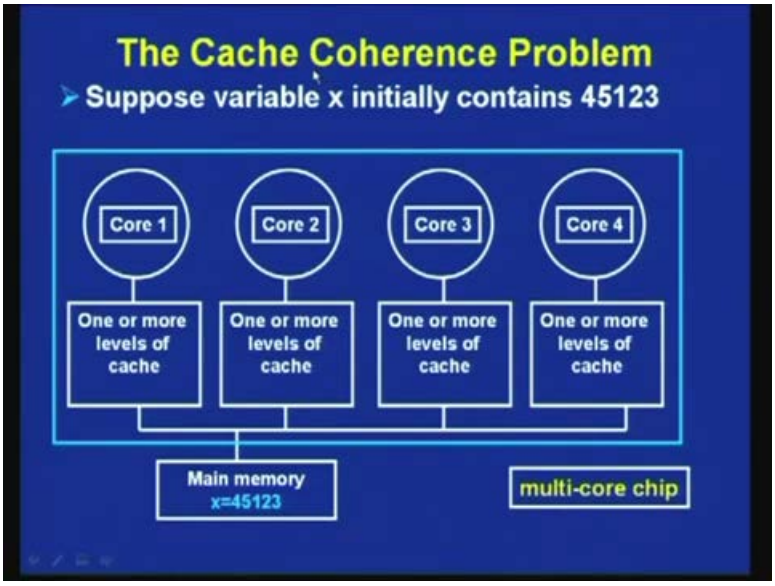
(Refer Slide Time: 22:13)



So, let me consider a multi core situation having four cores in a processor a multi core chip. Where as you can see you have got a main memory which is being shared and connected through a shared bus. So, since we have private caches question is, how to keep the data consistent across cache memories. So, each core should perceive the memory as a monolithic array shared by all cores that means, this main memory is being shared by all cores.

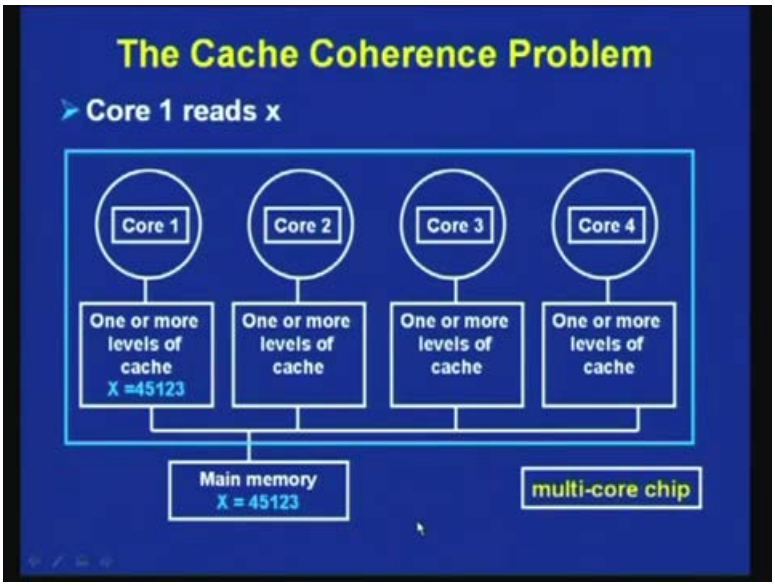
So, in such a case first we shall see how cache coherency problem is arising and how it can be then we shall discuss how it can be taken care of.

(Refer Slide Time: 23:00)



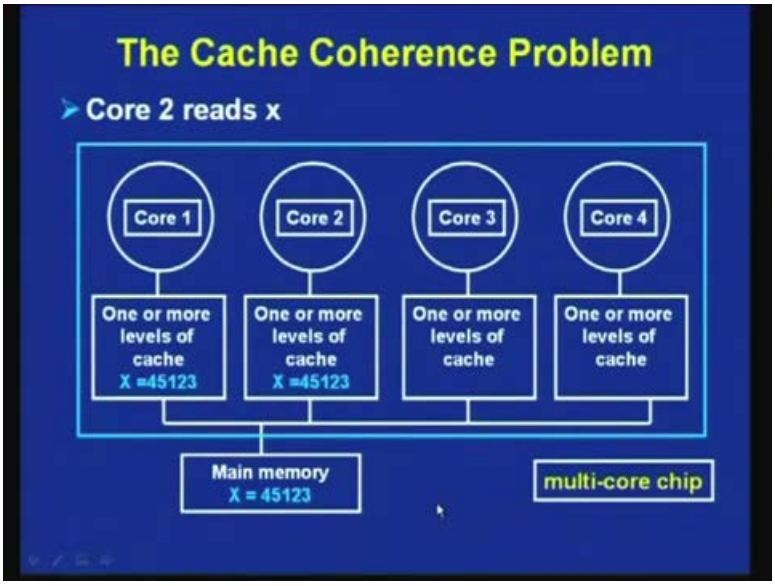
Suppose, in main memory you have got some data x is equal to 4 5 1 2 3, a particular variable x is here then a particular processor reads.

(Refer Slide Time: 23:14)



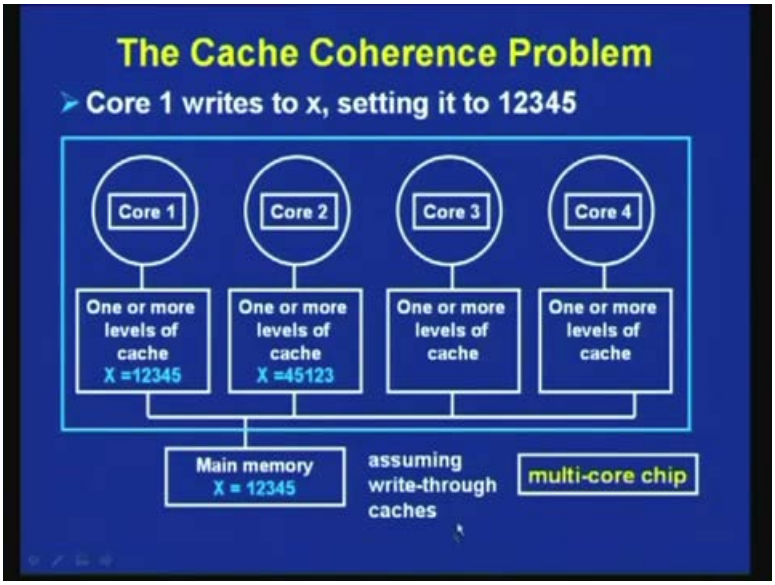
It through core 1 this processor reads it, x is equal to 4 5 1 2 3. So, from main memory it has been transferred there.

(Refer Slide Time: 23:24)



Then another processor another core feeds, it x is equal to 4 5 1 2 3.

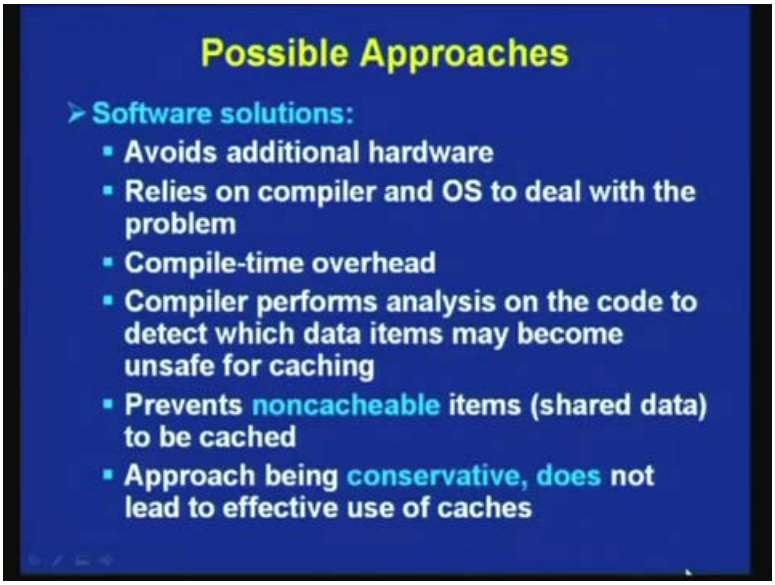
(Refer Slide Time: 23:32)



Now, suppose a particular processor I mean this is done as I mean now core 1 writes x setting. It 1 2 3 4 5. So, here what has happening this particular cache memory has been modified and assuming it is a write through cache the main memory is also updated. So, we find that this particular content of the cache memory 4 1 and the content of the main memory for the variable x is same. Because, of the write through policy that is being used.

But we find that in the cache memory of core 2 the value is different. So, you can see across all these cache memories will find, the variable x is not having the same value. So, this is your this leads to cache coherence problem.

(Refer Slide Time: 24:37)



Now, how we can overcome this there are basically we have two possible approaches one is based on hardware, another is based on software. So, first we shall see how we can maintain cache coherency problem by using a by using a software. So, whenever we go for software solution obviously, it does not require any additional hardware and it relies on compiler and operating system to deal with the problem that means, this cache coherence problem is dealt with the help of operating system and compiler.

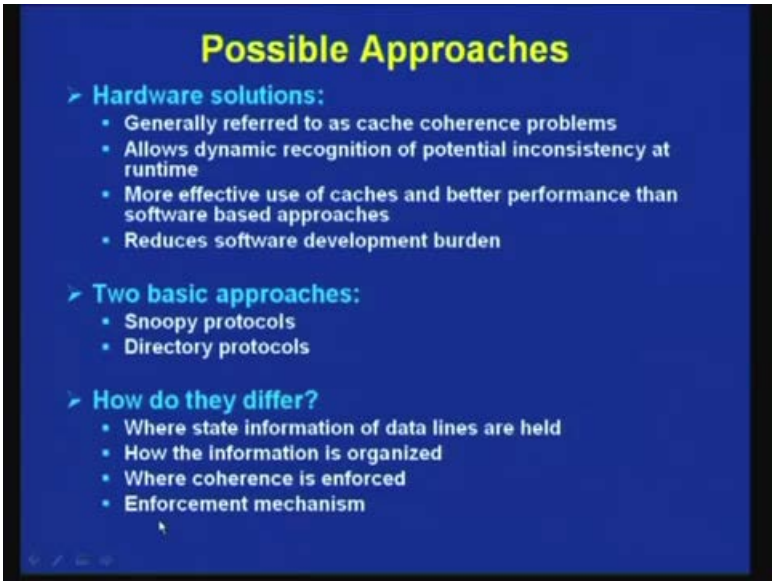
So, obviously it increases compile time overhead your compiler is no longer a simple straightforward compiler. It has to be an optimizing compiler, it will take into consideration the cache coherence problem and it will try to devise mechanism to overcome it how question is how. So, the compiler performs analysis on the code to how detect which data items may become unsafe for caching.

So, compiler has to perform analysis of data and should identify which data is vulnerable for cache coherence, which data may lead to cache coherence problem. So, then what it will do then it will prevent non cacheable items. So, that means prevents it will make the data non cacheable That means, the data which is shared by multiple cores or multiple

processors should not be taken to cache. So, it is somewhat like telling whenever you play football there is a possibility of injury.

So, do not play do not play football it is somewhat like that obviously, this solves the problem not very attractive. So, this approach being conservative does not lead effective use of cache memories you see cache memory has been used to the performance and obviously. If it is not used shared data is not forbidden to be taken to the cache memory does not serve the full purpose moreover you compiler becomes more complicated. So, this is not a very good solution although for the sake of completeness we have discussed it.

(Refer Slide Time: 27:26)



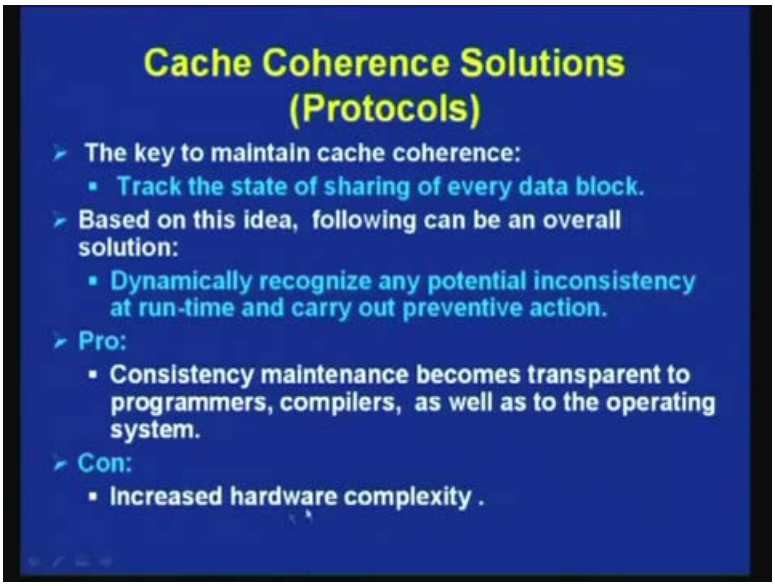
Now, we shall focus on hardware solution. So, these hardware solutions are referred as cache coherence problems. So, this is classical cache coherence problem and. In fact, the hardware solutions are referred to this refer to this, as cache coherence problem. So, this allows dynamic recognition of potential inconsistency at runtime. So, since it is done by hardware. It is not done at compiled time nor it is done by operating system.

So, it this coherency is mitigated at runtime, when the programs are running and obviously, it will lead to more effective use of caches and better performance than software based approaches and also it will reduce software development burden. That means, the programmers who are writing compilers, who are writing application programs, who are writing operating systems, their burden is reduced.

So, software designers are very happy about it now there are two basic approaches I mean hardware based approaches. One is known as snoopy protocols another is directory protocols. So, today we shall focus on snoopy protocols question is how do they differ these two techniques snoopy protocols and directory-based protocols can differ in several aspects 1 is state.

Where state information of data lines are held that means, you have to store, the state information of the data lines. So, that you can identify the cache coherence problem. So, question is where this is held, where this is stored and how the information is organized where coherence is enforced at what level then what is the enforcement mechanism. So, these are the four aspects in which these two protocol differs.

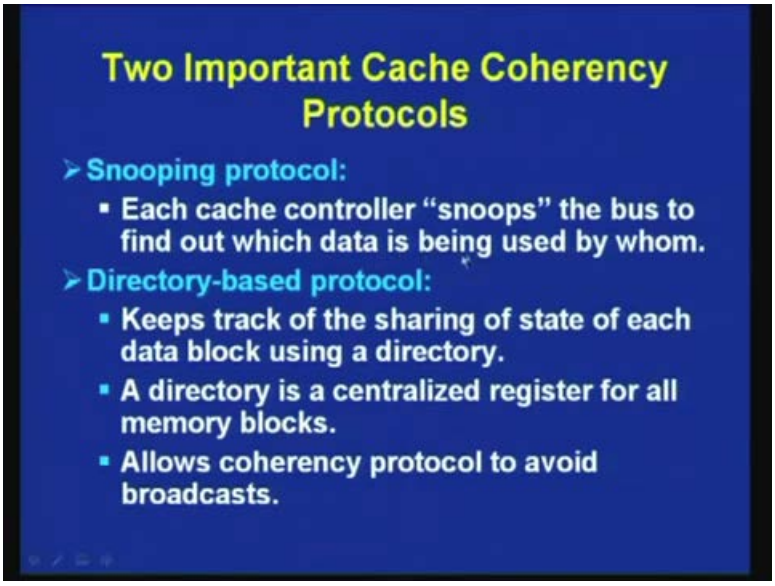
(Refer Slide Time: 29:45)



So, obviously the objective is same the key to maintain cache coherence track the state of sharing every data block. So, whenever a data is shared you have to track keep track of the state of sharing of every data block based on the idea, following can be the overall solution and dynamically recognize any potential inconsistency at runtime and carry out preventive action.

So, this is the basic philosophy that is being used by these cache coherence protocols. So, since this is done by hardware the advantage is consistency maintenance becomes transparent to the programmers, compilers as well as to the operating system and of course, it will lead to hardware complexity.

(Refer Slide Time: 30:48)

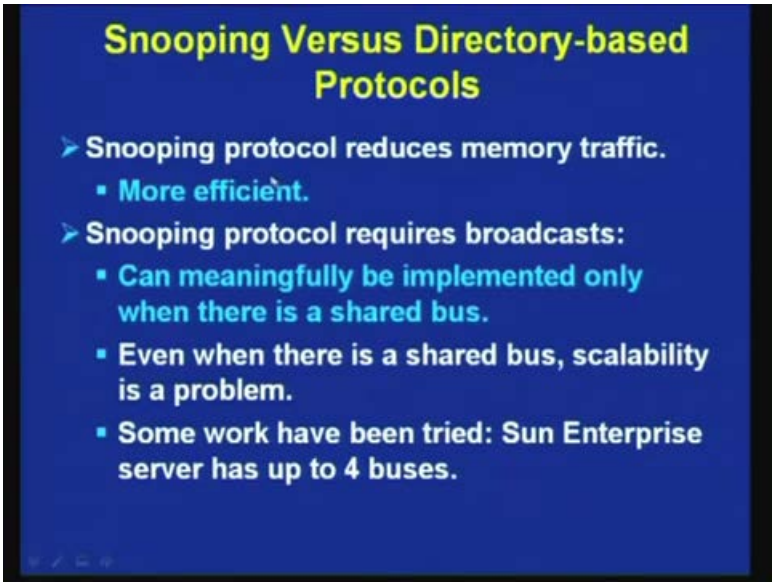


And, as I have already mentioned there are two basic approaches. One is snooping protocol, each cache controller snoops on the bus to find out which data is being used by whom.

So, here you know just like a dog keeps looping I mean wait sat the gate and keeps on observing. Who is coming through the gate who is going out it just keeps on monitoring and whenever somebody unknown enters the house it keeps on barking. So, it us somewhat idea is somewhat similar. So, each cache controller snoops the bus and keeps on monitoring and then only when it finds out that the data is being used by is being shared and others then it takes up appropriate action.

So, on the other hand in directory-based protocol it keeps track of sharing of state of each data block, using a directory that is the name that is why the name directory-based protocol. So, directory is maintained, so directory is a centralized resistor for all memory blocks. So, in this case you may say it is a centralized enforcement system on the other hand a snooping protocol is distributed enforcement system. Because each cache controller takes control of it and allows coherency protocol to avoid broadcasts. So, this is the two techniques and today we shall focus on.

(Refer Slide Time: 32:22)

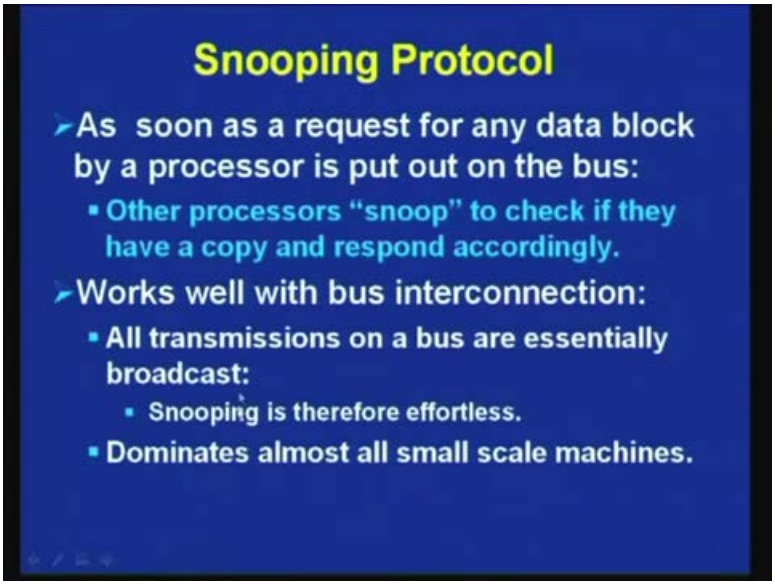


Snooping protocol because, snooping protocol reduces memory traffic is more efficient and snooping protocol requires broadcasts. Because, there is no problem about the broadcast because you have a shared bus and that is the reason, why we call it shared memory multiprocessor based system and since you have got a shared bus. The data is broadcasted on the bus and supreme protocol requires the bus and can meaningfully be implemented, only when there is a shared bus.

So, as I mentioned it cannot function without a shared bus. Even when there is a shared bus scalability is a problem as I have already mentioned. So, because as the number of processors increases on a bus traffic on the bus increases performance degrades. So, that in a single shared bus system the number of processors that you can have is limited may be one dozen two dozen at most a few dozen. If cannot have massively parallel computers. Where you can have a 1000 2000 processors.

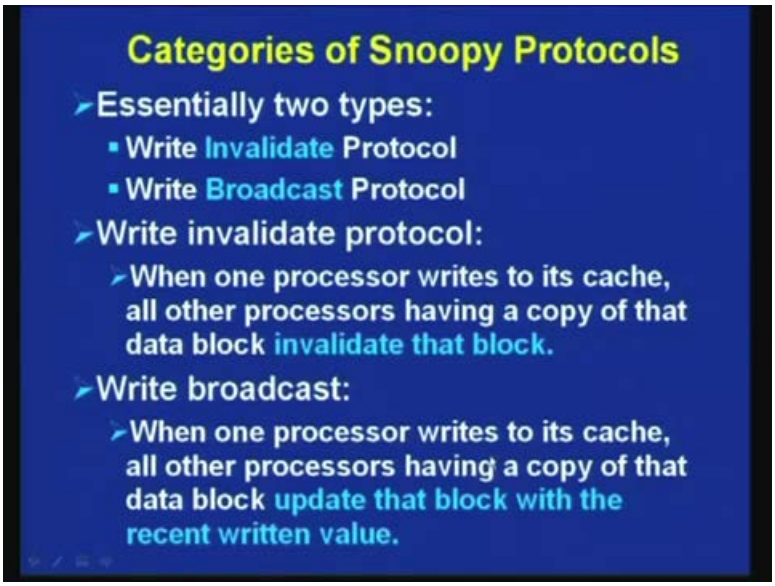
So, that you cannot have in a shared memory system. So, one alternative could be to have more than 1 buses some work has been tried sun enterprise server has up to 4 buses. So, you can increase the number of buses to reduce the traffic on the each bus. Where it has been found that it is not a very I mean a very good approach. So, some people have tried, but not very successful commercially.

(Refer Slide Time: 34:14)



So, let us go into the details of snooping protocol as soon as a request for any data block is put by a processor on the bus. Other processors snoop to check if they have a copy and respond accordingly that means, each processor that means the cache controller residing in each processor snoops to check if they have a copy and respond accordingly. So, this works well with bus interconnection all transmissions, on a bus are essentially broadcast snooping is therefore, effortless you do not have to do anything else and it dominates almost all small-scale machines as I have already told with a limited number of processors in a system.

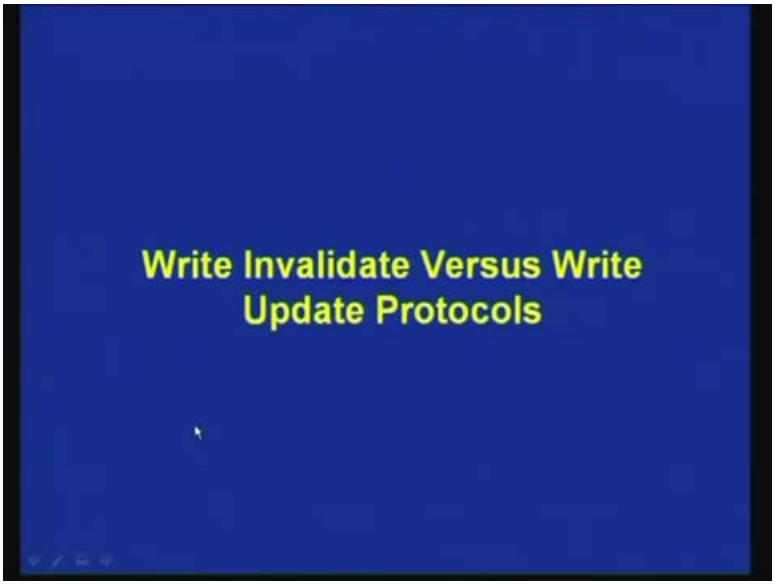
(Refer Slide Time: 35:07)



Now, let us look at the different categories of snooping protocols first one is write invalidate protocol, second one is write broadcast protocol. So, the protocols can be broadly divided into two types write invalidate write broadcast and in case of write invalidate protocol. When one processor writes to the cache all other processors having a copy of that data block invalidate that block.

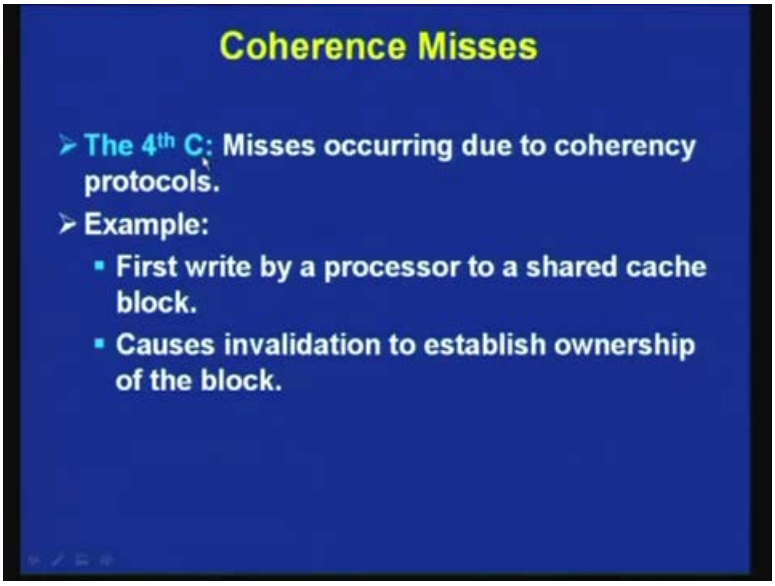
So, it is pretty simple. So, and it can be very easily implemented and on the other hand in write broadcast rights to one processor. Its cash all other processors having a copy of that data block update that block with the recent written value. So, this has happened in the case of write broadcast.

(Refer Slide Time: 35:59)



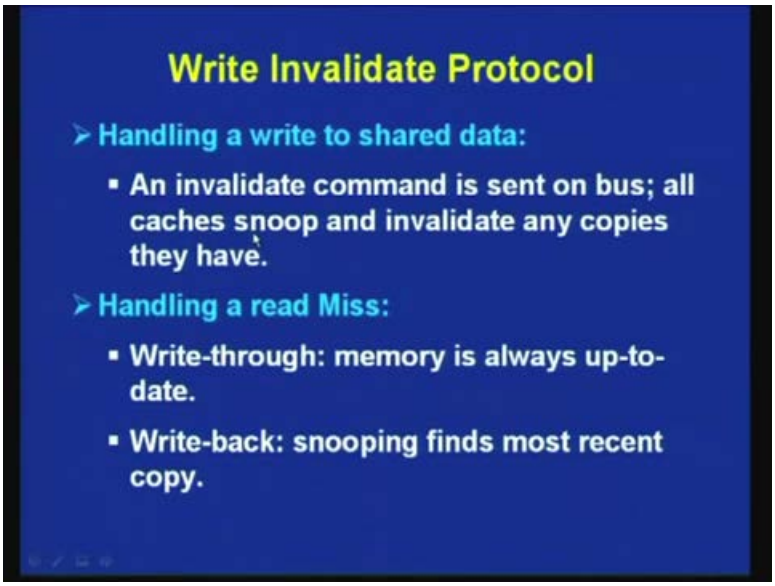
First we shall focus on write invalidate versus write update protocols.

(Refer Slide Time: 36:04)



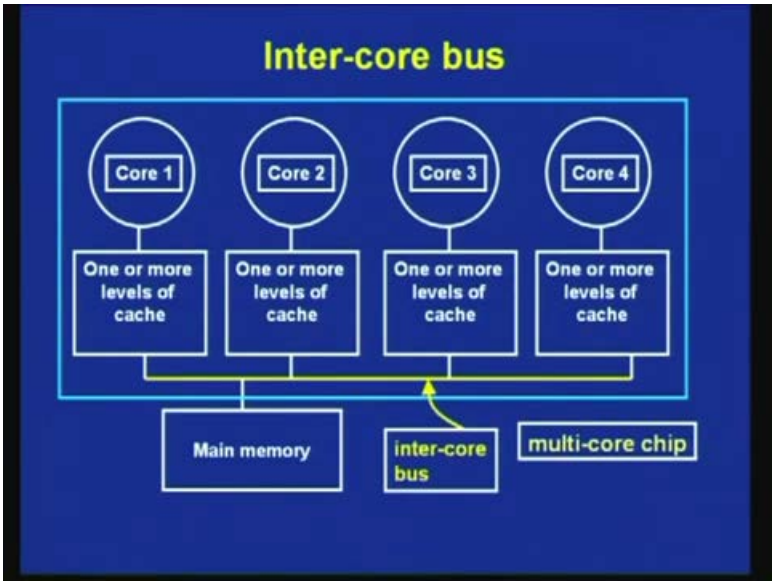
So, this is the fourth c I mention that cache coherence protocol is the fourth c that three C. I mentioned earlier and this is fourth c

(Refer Slide Time: 36:21)



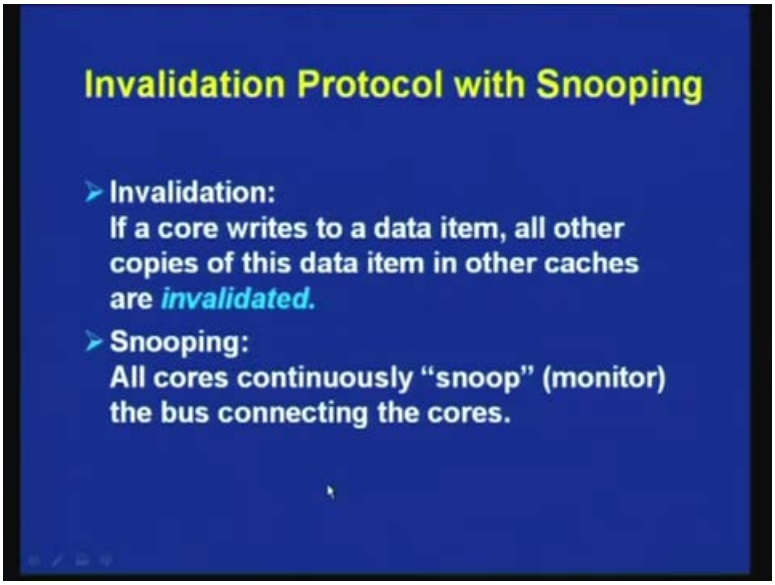
So, let us focus on the write invalidate protocol handling are write to the shared bus all invalidate command is sent on the bus. All caches snoop and invalidate any copies they have and whenever a read miss occurs write through memory is always up-to-date write back snooping finds most recent copy.

(Refer Slide Time: 36:47)



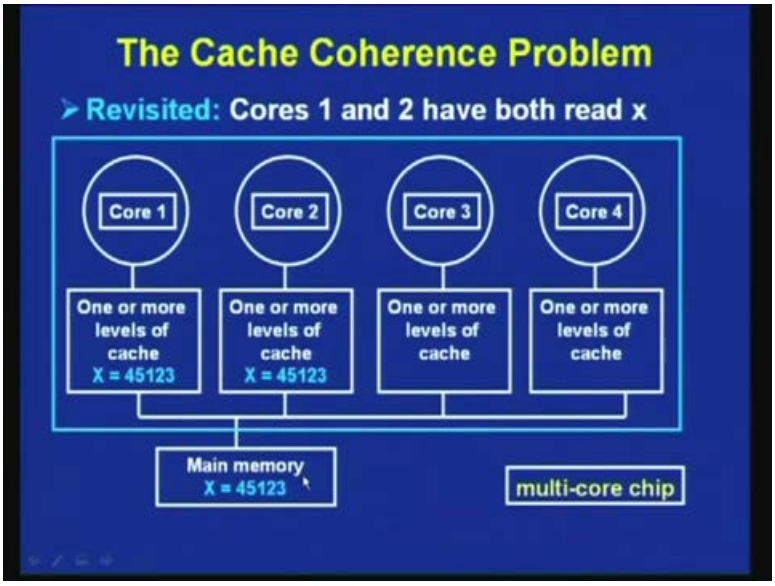
So, let us illustrate with the help of this example where you have got a multi core chip.

(Refer Slide Time: 36:55)



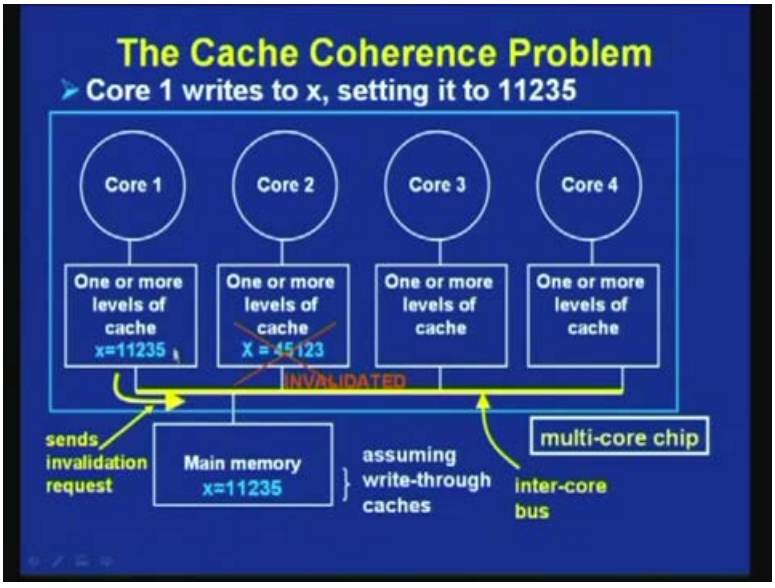
And, if a core writes to a data item all other copies of this data item in other caches are invalidated.

(Refer Slide Time: 37:07)



So, let us see this is the scenario that we have started already three I mean the main memory you have a variable x is equal to 4 5 1 2 3 and we have copies in two cache memories corresponding to core 1 and core 2 and having the same values. So, right now they are consistent data is consistent.

(Refer Slide Time: 37:32)

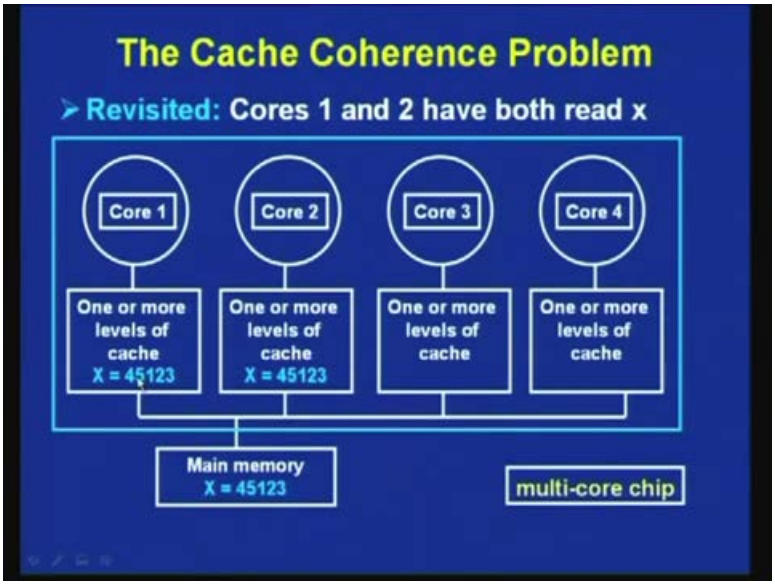


So, now suppose a this core 1 writes x setting it to 1 1 2 3 5. So, this core one has modified has written data into it and whenever data is written, then it is also sent to this bus. So, there is inter-core bus through, which that invalidation request is sent when this

data is modified it is only sending the invalidation request corresponding to this variable x.

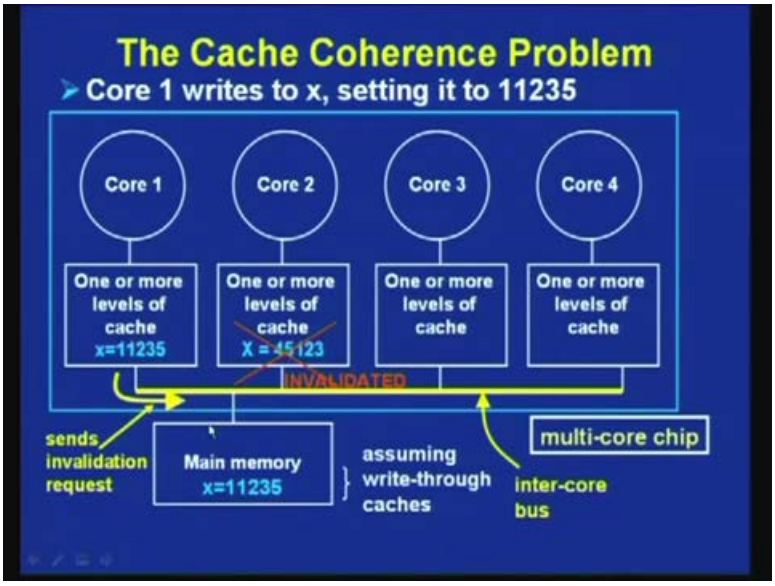
So, what happens in such a request the core which is having this variable x is equal to x. So, this will simply invalidate it. So, invalidation will take place in this of course, in the main memory. It will remain the same I mean because, of the write through policy that is being used here in the main memory it is updated.

(Refer Slide Time: 38:31)



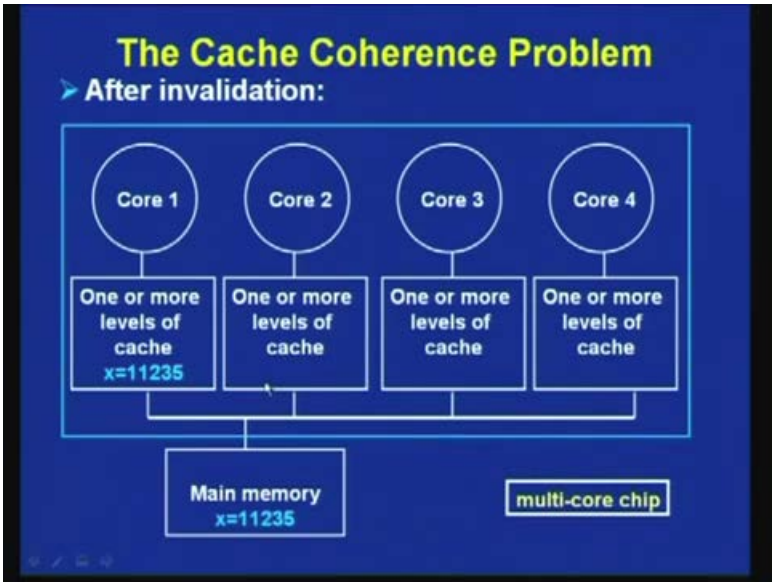
Corresponding to this you can see here earlier it was 4 5 1 2 3 in both cache memory of core 1 as well as in the main memory.

(Refer Slide Time: 38:41)



Now, in both the places it is modified because, of writing in the cache memory. It also writes in the main memory and then, whenever this writing takes place on the bus that invalidated signal is propagated request is sent and as a consequence this particular entry is invalidated what does it mean.

(Refer Slide Time: 39:04)

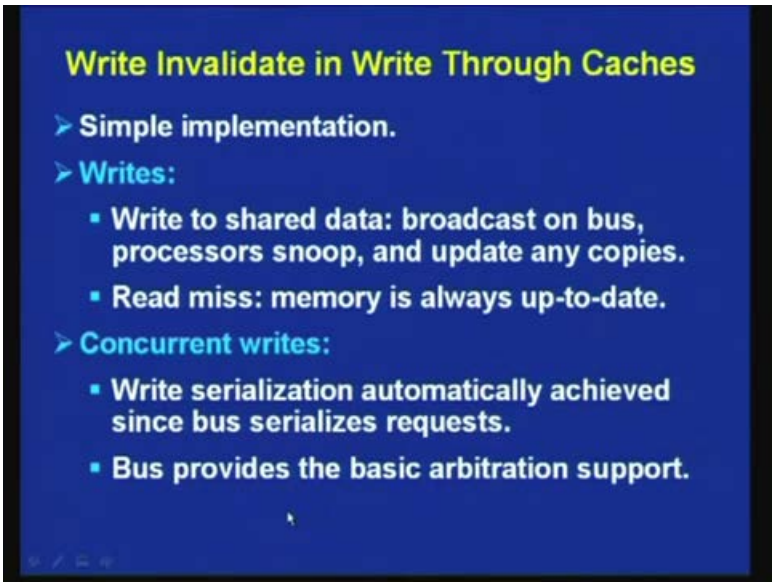


Invalidated means it is simply removed from that cache memory; that means, as if a copy of that variable is not present in this cache. So, this is what happens after that

invalidation. So, after invalidation you can find that this particular cache memory is not having a variable x. So, that is removed. So, this is your invalidation technique.

Now, So of course what can happen core 2 reads x cache miss occurs and loads the new value that means, if core 2 wants to read it I mean whenever it performs the read x operation obviously. It will encounter cache miss because, it is not present in this cache. So, on through that cache miss it will get a new copy . So, now, again it becomes consistent. So, this is how this invalidation protocol works.

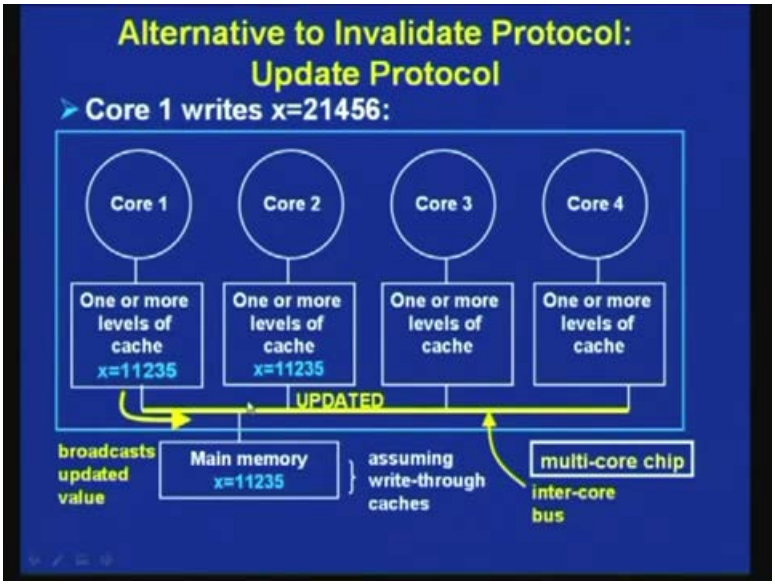
(Refer Slide Time: 40:09)



And, it is very simple to implement write to shared data broadcast on bus, processors snoop, and update any copies. So, these are things to be performed in case of writes and in case of read miss memory is always up-to-date there is no problem and whenever you do concurrent writes.

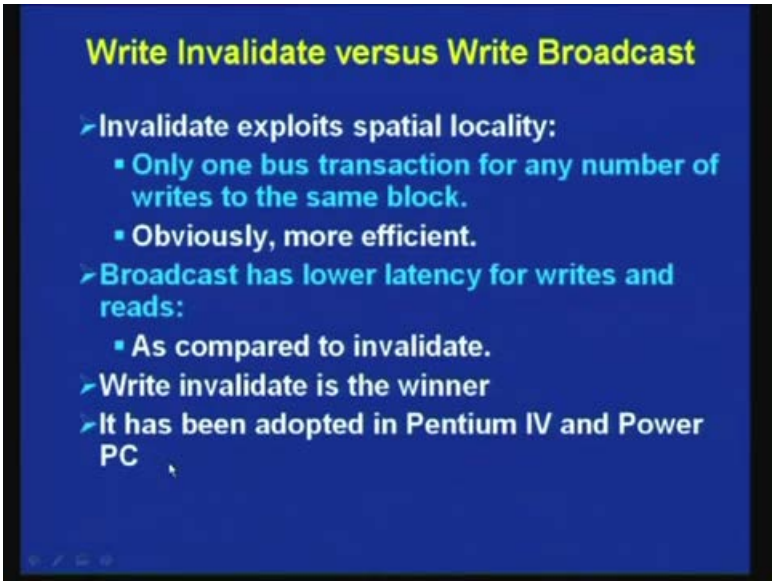
Serialization automatically achieved since bus serializes requests, so that means, you can do to concurrent writing. That means, whenever this writing occurs and then subsequently you can write it in this also, that is what is being mentioned write serialization automatically achieved. Since bus serialization request, so bus provides the basic arbitration support. So, this is these are all bus based.

(Refer Slide Time: 41:02)



Alternative to invalidate protocol is update protocol. So, in case of update protocol what is happening it has broadcasts update value on the bus instead of invalidate signal. It will send the updated value through the bus and as a consequence wherever that variable is present, that variable will be copied and I can see this is how the consistency is being maintained.

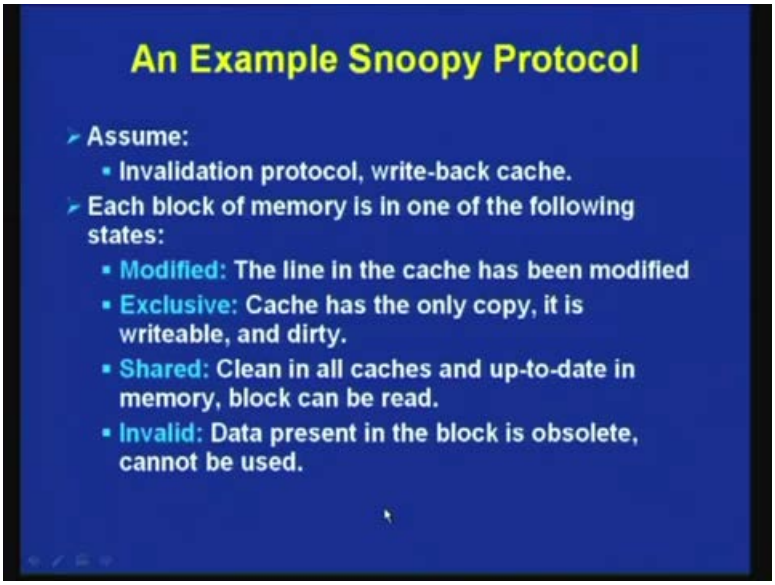
(Refer Slide Time: 41:34)



So, invalidate exploits spatial locality only one bus transaction for any number of Writes to the same block and obviously. This is more efficient and broadcast has lower latency

for writes and reads as compared to invalidate protocol. So, write invalidate is winner of these advantages and as a consequence. This has been adopted in Pentium IV and power P C, this particular protocol has been adopted.

(Refer Slide Time: 42:12)

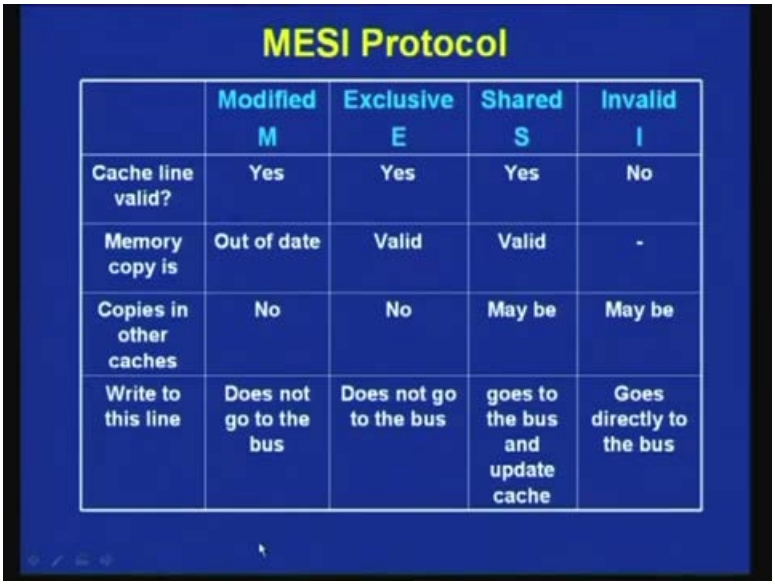


Now, let us consider the snoopy protocol. So, invalidation protocol write back cache we have to assume that invalidation occurs and we are using write back cache memory not write through. So, based on this assumption each block of memory is one of the following states. Is having one of the following states modified exclusive shared and invalid that means, that each block of the memory as you now a cache access takes place in terms of blocks.

So, we are thinking in terms of blocks and each one of them can have different states and in the for each of these blocks with the help of flag waits these states are maintained. So, modified means this line in the cache has been modified exclusive means cache has the only copy it is writable and dirty. That means It is in exclusive use of a particular core shared means it is clean in all caches and up-to-date in memory block can be read without any problem shared and invalid means data present in this block is obsolete cannot be used.

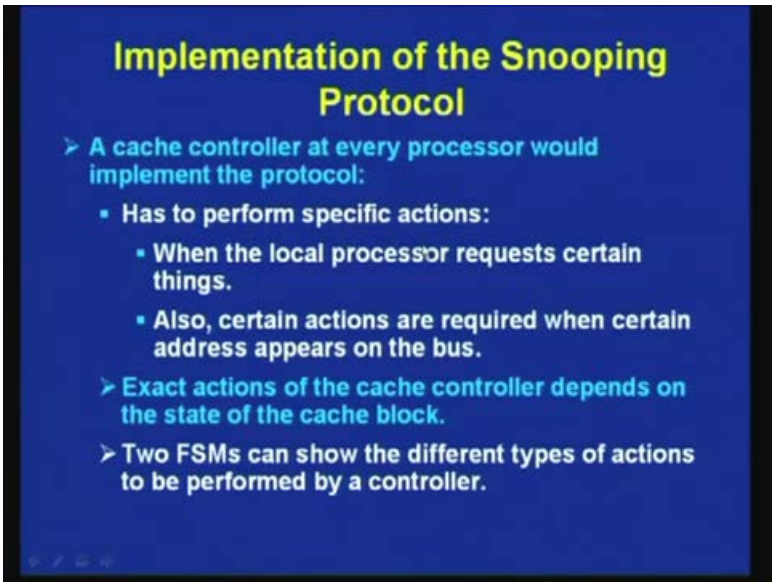
So, this is how you can perform I mean with the help of these four states you can implement this snoopy protocol.

(Refer Slide Time: 43:40)



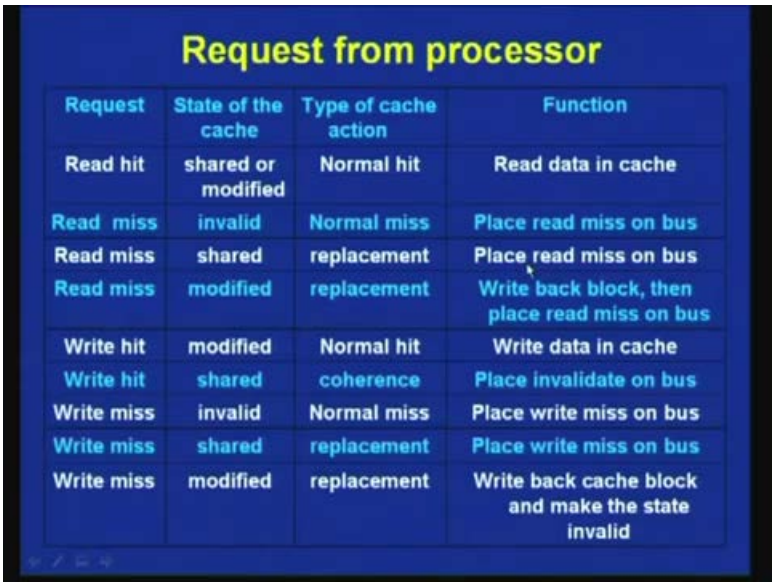
Let us see and. So, cache line valid. So, these are the flag beats mentioned memory copy is out of date. When it is modified with and exclusive E is valid, shared is valid copies in other blocks, no in this shared or may not be shared. So, these are the different cases write to this line does not go to the bus inclusive exclusive states. Shared state goes to the bus and update cache in case of shared and goes directly to the bus whenever it is invalid.

(Refer Slide Time: 44:21)



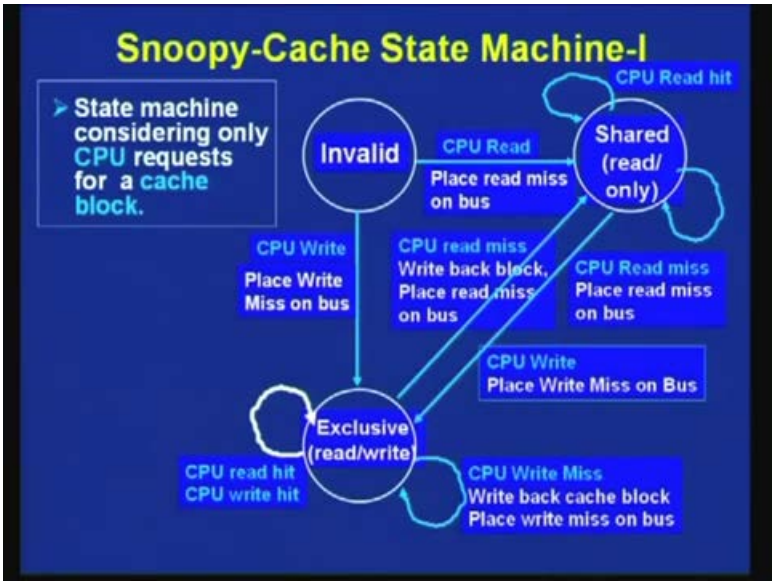
So, cache controller at every processor would implement this protocol, has to perform specific actions when local processor requests certain things and also certain actions are required when certain address appears on the bus. So, exact actions of the cache controller depends on the state of cache block. So, you require two finite state machines that can show different types of actions to be performed by the controller.

(Refer Slide Time: 44:50)



So, let us look at different actions that is being that is possibly these are the different types of requests. That can be done and that the state of the cache controller shared or modified invalid and. So, on and types of cache actions that can occur normal hit normal miss and. So, on and these are the functions to be performed for different situations. So, whenever the read hit occurs reads data, in the cache read miss occurs place the read miss on the bus, read miss occurs place read miss on the bus like that So, I think it will be better.

(Refer Slide Time: 45:30)



If we discuss it with the help of finite state machine. So, this is the state machine considering only the c p u requests for a cache block. So, C P U is requesting for a cache block to read in such a case here three states are shown invalid, shared and exclusive that state is not required, that modified is not required in this case. So, C P U read is occurring place read miss on the bus. If it is invalid means the data is not present in the cache.

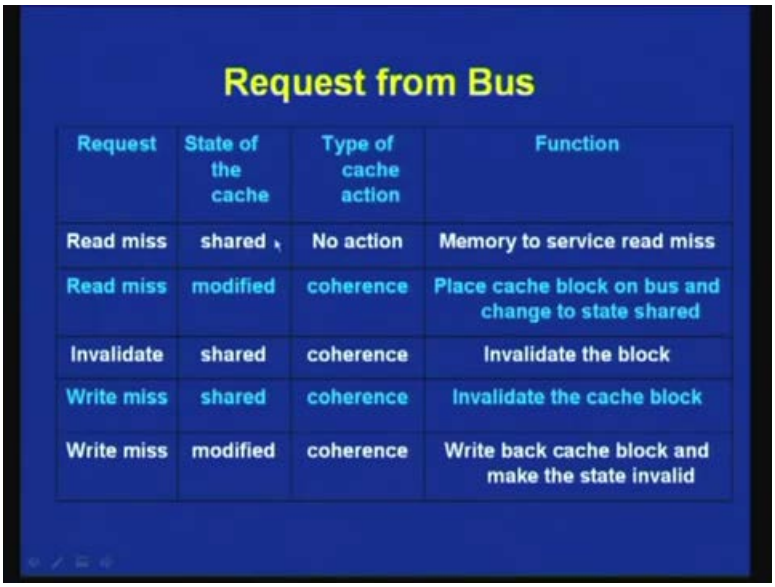
So, it will lead to read miss it will go to shared state from the invalid state to shared state. Whenever read miss occurs then the C P U read hit occurs, then it remains in the state keeps on reading it. Then a shared read miss occurs then again it will remain in the shared state plus read miss on the bus. So, cache controller will put on this read miss on the bus and then whenever C P U writes it then it will go from shared to exclusive state.

Because, now modification has been done only that particular cache controller. I mean that particular core is having exclusive right, I mean exclusive state because other cache memories. I mean old copies and then C P U read hit and C P U write hit occurs and it will remain in the exclusive state as long as read and write hit occurs I mean. It will remain in this exclusive state that cache controller c p u read miss occurs and write back.

It will also write back cache memory block place write miss on the bus remains in the exclusive state for all these cases now a c p u read miss occurs and write back block is done place read miss on bus it will again go back from exclusive to shared and finally,

whenever C P U write occurs and place write miss on the bus. It will go to invalid to this exclusive state., so whenever a c p u write occurs c p u read write all possible cases have been considered, considering the state machine and considering c p u requests and back as block.

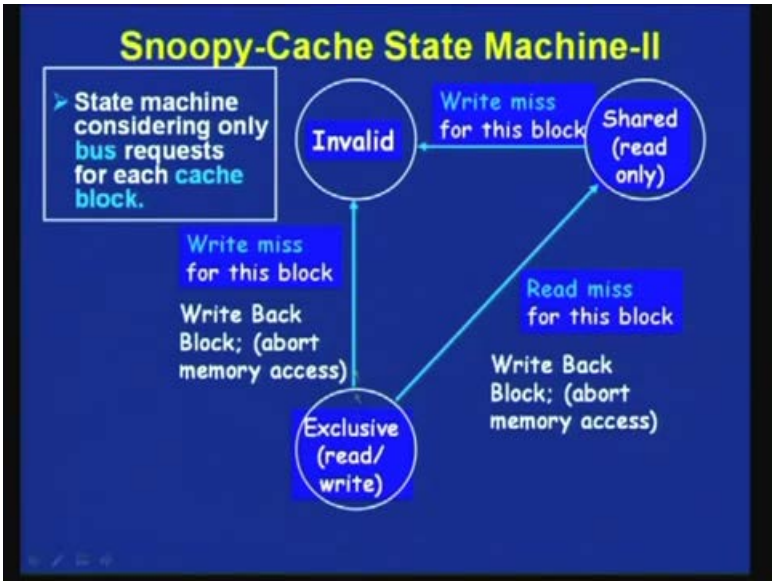
(Refer Slide Time: 48:19)



So, these are the various requests which can come from the bus read miss, read miss shared modified depending on the state. No action whenever read miss occurs it is in the shared state memory to service and read miss and modified state; that means, it will lead to cache coherence problem and place cache block on the bus and change to shared state and if the request is in the invalid state.

State of the cache is shared then again cache coherence problem invalidate the block write miss. If it is in the shared state coherence problem arises, then it will invalidate the block and write miss occurs it is in the modified state then again cache coherence problem write back cache block and make the state invalid. So, this is how this will occur.

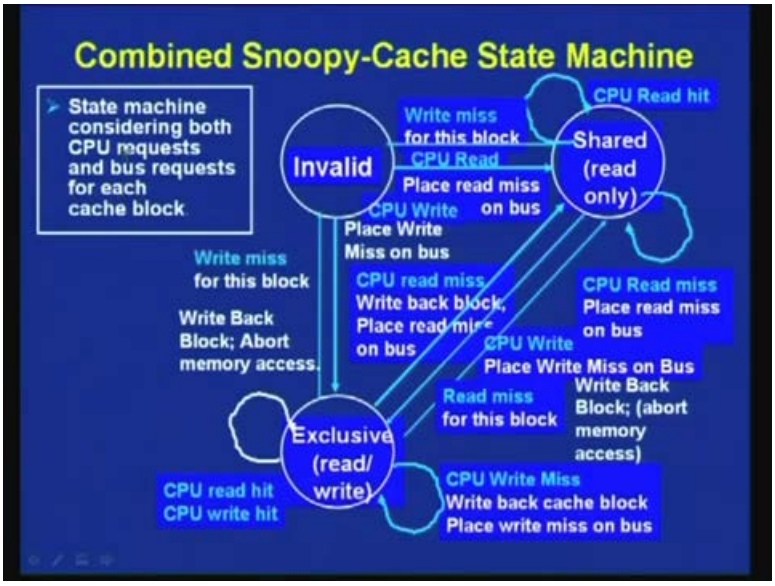
(Refer Slide Time: 49:18)



And, again it can be explained with the help of a state machine diagram. So, state machine considering only bus requests for each cache block. So, from exclusive state it will go to shared state. Whenever a read miss occurs for each block or write back policy is being used abort memory access. So, from exclusive to shared state it will go and then from invalid state. When write miss occurs it will go from shared to invalid state and write miss occurs, whenever there is a it is in the exclusive state.

And write miss is occurring for this block and write back block in this case abort memory access again it goes to invalid state. So, this is how it will occur.

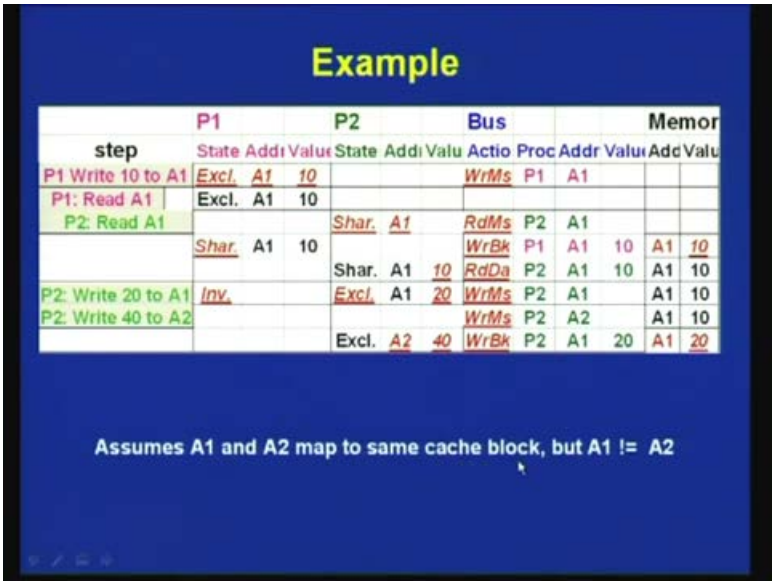
(Refer Slide Time: 50:16)



And, combining the snoopy cash state machines I mean both the considering, the state machine considering both the C P U requests and the bus request for each cache block. This is the situation I mean we have combined the both the finite state machines earlier we have discussed the operation of two different finite state machines.

Now we have combined both the finite state machines and to have a snoopy cash state machine again the states are same invalid shared exclusive and different I mean transitions have been combined to represent this I have already discussed the two finite state machines separately. So, I shall not discuss it any longer.

(Refer Slide Time: 51:08)



Now, let me illustrate this with the help of an example. So, you have got two processors p 1 and p 2. So, this state of the processor address and value. So, these are for each of these processors this is written for p 1 and p 2 and then the bus action is written here action and similarly, for the memory what is the value and various values are being mentioned. Let may illustrate this with the help of this example. So, p one write 10 to a 1.

So, it will go to exclusive state because assuming that it was in the invalid state, it will go to exclusive state and at this is A 1 and value is 10 and similarly on the bus that the write memory that action is taken and processor involved is P 1 and address is a one. So, that will be visible on the bus and that will what will happen inside the processor then p 1 reads A 1 in that case.

Since it is already in the exclusive state reading is not a problem. It will simply read this data and it will lead not go to the bus there is no note change on the bus that means, bus will not have any action. There will be no nothing will happen on the bus, now p two reads A. So, P 2 another processor is reading A, so now, it will go from exclusive to shared state. So, you can see now the state of both P 1 and P 2 cache both are changing.

So, it has been earlier it was in the exclusive state now it is in the shared state board for since P 2 is reading here it will go to shared state and here it is read miss on the bus and P 2 is the source of the bus and address is A 1 and then in response to that that P 1

processor will switch to shared state to exclusive state and that on the bus again that write back will take place that P1 A1 is equal to 10 and for the memory A1 will remain 10. So, that does not change.

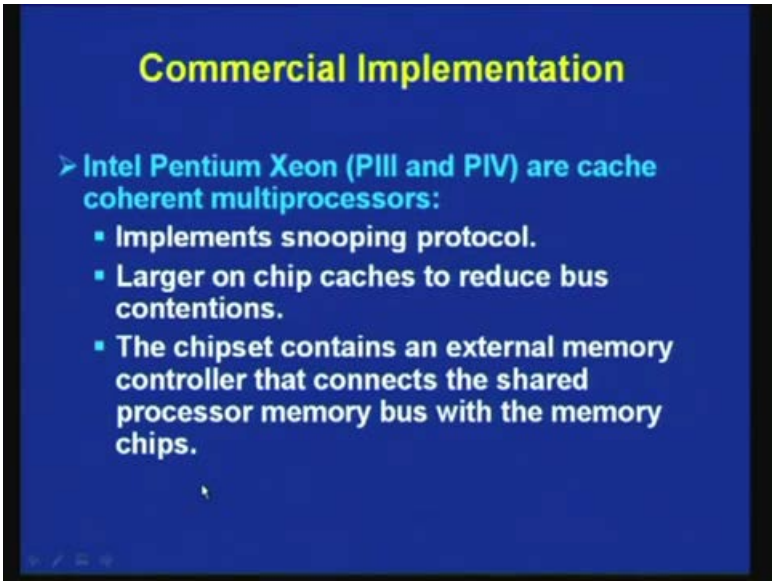
So, this is what will happen now P2 writes 20 to A2. So, in such a case since P2 is writing on the memory location 20 then P1 will invalidate that particular block. So, invalidation has taken place and it will go to the exclusive state of the processor P2 core P2. So, it will go to the exclusive state of P2 and on the bus. It will appear write miss on the bus and P2, it is originated from P2 and address is A1and memory will write address is A1 and I believe it will be 20 and, so some change is required.

Now P2 writes 40 to A2, so in this case there will be no change. So, far as the processor P1 is concerned it will remain in invalid state and assuming that A1 and A2. They belong to the same block it will go to the exclusive state, it is already in the exclusive state. So, it will remain in the exclusive state and that it will perform this write miss will occur and then a write back policy will be used and 20 P2 will write.

So, this is for the purpose on the bus P2 will generate that and write policy action is right that and for the address A1. It is 20 value is for the address it is 20, so that is 40 to A2 and. So, far as A1 is concerned it remains 20. So, we find that A2 on the memory. I believe there is some mistake here all are 10 A2 for A2 value is not shown here. A1 is remaining 10 and A1 here is 20 for A2, it is value is not given here.

Because, it is restricted to the processor it does not go to the bus, that is why it is not given here. So, assume A1 and A2 map to same cache block, but A1 is not A2.

(Refer Slide Time: 56:49)



So, these are the commercial implementations Intel Pentium Xeon P III and P IV are cache coherent multiprocessors, implement snooping protocol and larger on chip caches, to reduce bus contentions and the chipset contains an external memory controller. That connects the shared processor memory bus with memory chips. So, with this we have come to the end of our discussion on, shared memory multiprocessor systems or symmetric multiprocessor systems.

In my next lecture we shall focus on distributed memory multiprocessor systems.

Thank you.