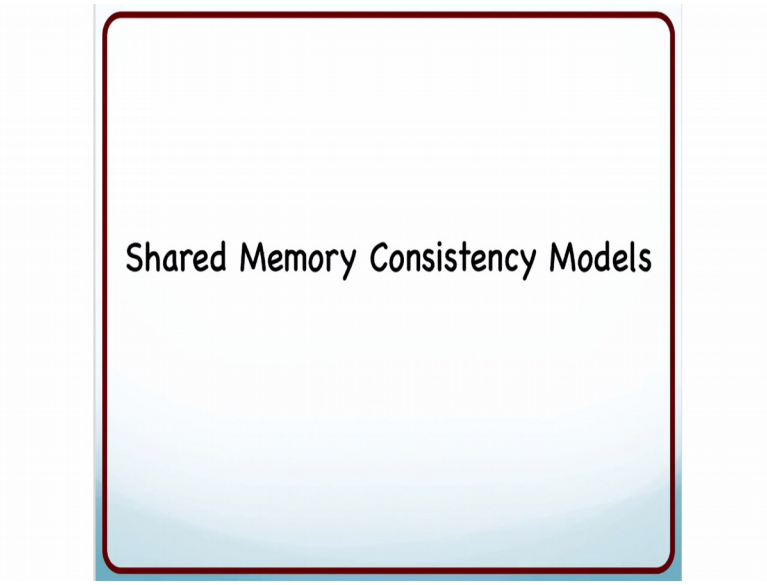
Introduction to Parallel Programming in OpenMP

Dr. Yogish Sabharwal
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

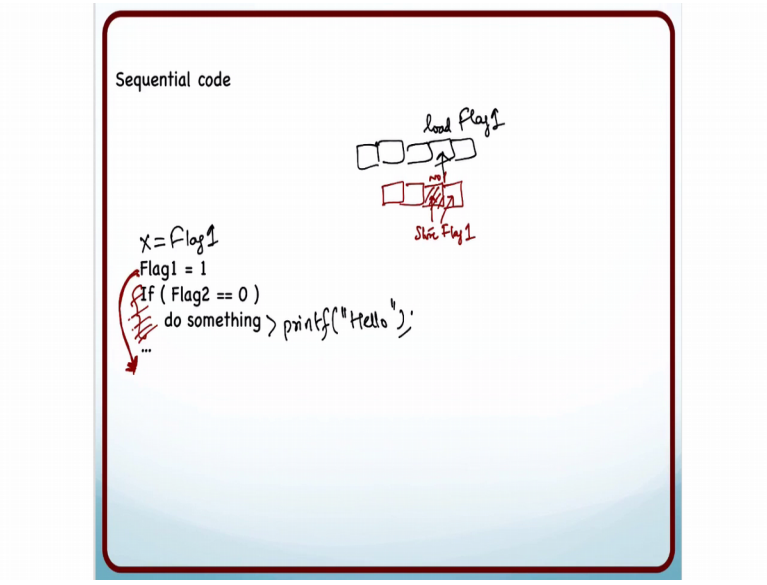
Lecture – 17
Shared Memory Consistency Models and the Sequential Consistency Model

(Refer Slide Time: 00:01)



Today we are going to talk about shared memory consistency models.

(Refer Slide Time: 00:06)



And just to motivate why we need these consistency models consider the sequential code right. It sets flag 1 equal to 1 and then it takes a flag 2 is equal to 0, then it does something right. Let us say prints hello right. Now when you give this to the compiler right when this code is compiled one of the things that the compiler generally does is reorganize the code right to make it more efficient.

So, let us say that you set x equal to flag 1 just before this instruction and then you execute the flag 1 equal to one. So, what will the compiler do? So, over here as you recall right typically instructions are pipelined. So, what is it going to do it is going to load the value of flag 1 at some point in time right, and store it into x right, but a load has to happen flag 1 has to be loaded? And now in the next instruction we are saying that flag 1 is equal to one right and now over here it cannot change the value of flag 1 until that previous value of flag 1 has not been assigned to x, right I cannot change the value of flag 1.

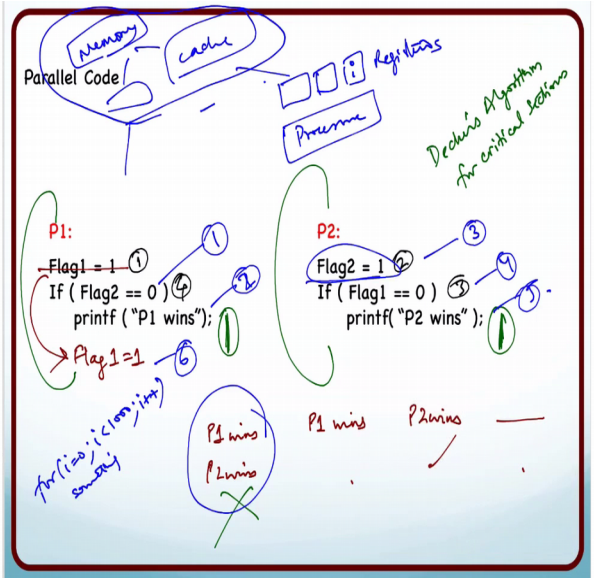
So, there are various ways of dealing with this, but one of the things that the compiler will typically do in such a scenario is it will moves this instruction later right how does that help. So, the way it helps is the suppose that this is where in the pipeline, this is where you were going to store the value one to flag 1 right. Now you cannot execute this because flag 1 has not been yet copied out right. So, maybe you insert a nop over here, you do not do anything in that cycle, and then in the next cycle you end up doing the store right. So, that the previous value has been taken right has been put somewhere else.

So, there are various of scenarios which occur and the crux is that the compiler wants to reorganize the code right. So, the compiler realizes that this flag 1 is not being used in these instructions right a flag 2 equal to 0 printf hello. So, flag 1 is not being used. So, it can safely move this instruction over here, it would not change the output of the code right. So, typically when you write some code and when you compile the code or the compiled instructions will not match exactly with the sequence of the instructions that you are specified. The code will be rearranged by the compiler that is a very normal optimization that all compilers do.

And you need these kind of optimizations otherwise your code will run very slow. So, these optimizations help your code run much faster. So, there are loads of things that they do they do loop unrolling and various other stuff right. So, as far as sequential code is

concerned, I can safely move this flag 1 equal to one instruction later and nothing will be disturbed right the output will still be the same of the program, all right.

(Refer Slide Time: 03:21)



What happens in our parallel code?

So, now let us say that I have the following code. So, it is a race between 2 processors P 1 and P 2 right. So, we want to see who reaches the finishing line first. So, P 1 declares flag 1 equal to 1 saying that I have reached the finish line, then it checks a flag 2 is equal to 0; that means, that the other processor has not reached the finishing line then it declares itself to be the winner right. So, the code is clear? Similarly what the professor to does is it sets flag 2 equal to one, and then if flag 1 is equal to 0, it prints P 2 is the winner it declares itself as the winner.

So, what are the possible outcomes of this code is. Let us say I was executing it on a single processor. So, these instructions may be interleaved in some way and let us say there is no compiler reordering of the code, how would this code get executed, what could be the possible outcomes? Either P 1 wins write this is one possible outcome P 1 wins another possible outcome is P 2 wins, any other outcome that is possible. So, one other possible outcome is

Student: (Refer Time: 04:44) no output.

No output right no output at all and is these a possible output P 1 wins P 2 wins both of them getting printed. No there is no possible interleaving of these instructions that you can come up with such that you know both of them will declare themselves to be the winner is this point clear. So, this is not going to happen that is how you are designing this code? This is part of the code of dickers algorithm for critical sections. So, will come across this again, but essentially this allows you to put a critical section code where you are doing the printf what does critical section mean? Only one thread is allowed to execute at a time right.

So, you can put that code over here where you see this printf. As long as what is happening that the moment you set flag 1 equal to one the other processor must also see flag 1 equal to one. The moment you set flag 2 equal to one P 1 must also see flag 2 equal to 1. So, as long as you can ensure that right then this algorithm works for critical sections. So, you put your critical section code over here, processor to put the critical section code over here, but in critical section you always want to make progress. So, you can always put this in a loop right. So, that is fine if you fail you go back and try again right. So, that that is a simple.

All right let us come back to the question that on a uni-processor how can it happen that, nobody prints anything. So, here is how it can happen? So, this is the first instruction to be executed flag 1 equal to 1 right and then the program gets swapped out and the other program gets scheduled right. So, the other program execute the statement next and then this statement.

Student: These 2 codes will be in a single mainstream code.

No, no, no, they are 2 threads running on the same processor right. So, they will get swapped in swapped out or even if their processes process gets swapped in swapped out right here we are talking about shared memory. So, let us stick to threads right. So, if there are threads then professor one will get swapped out at some point in time processor 2 will get swapped in and so on right. So, then what will happen? So, professor 2 it will it checks a flag 1 is equal to 0 and what is it find it is equal to one right. So, it cannot enter this condition.

Similarly, now control goes back here and now processor 1 checks a flag 2 equal to 0 again it finds it to be one and therefore, none of them print. So, now, the question is that I

understand this is the behavior I want, but is the parallel processing system going to promise me that. Now I am suddenly I am asking the system to follow certain rules why because if I just compile the code which is running on P 1 if I compile it separately remember it will see flag 1 is equal to 1 if flag 2 is 0 print P 1 wins. So, I am not using flag 1 again. So, it might end up moving this flag 12 later. What happens if it moves flag 1 equal to 1 2 later, what happens if a reorganizes the code. Now like we discussed earlier right compilers reorganizes code, what happens if it reorganizes code now?

Student: P 2 wins.

P 2 wins

Student: (Refer Time: 08:28).

That is an expected output. So, if any of these 3 occurs I am I mean this is expected output, what is more serious is that this also may happen right that is a serious issue how can that happen?

Student: (Refer Time: 08:45).

First this instruction gets executed right if flag 2 equal to 0 is 0. So, it is going to enter and print P 1 wins right and then it gets scheduled out, this program comes in it sets flag 2 equal to 1, but the damage has already been done right, it takes flag 1 equal to 0 yes it is 0 it is not been set to 1 here. So, it comes in prints this right and then finally, maybe at some point of time this program execute flag 1 equal to 1 right. So, you ended up printing both P 1 wins P 2 wins, when I am in a multiprocessor system I need to be more careful when I am writing programs how is the system going to interpret these programs.

A lot of things happen right for instance there are lots of optimizations, there is getting into architecture, but still the compiler decides that what variables am I going to keep it registers. Remember there are these registers which are right next to the processor and then there is the cache and then there is the main memory right I am going all over the place. The other cache is on this memory subsystem together they form your memory system which is consistent.

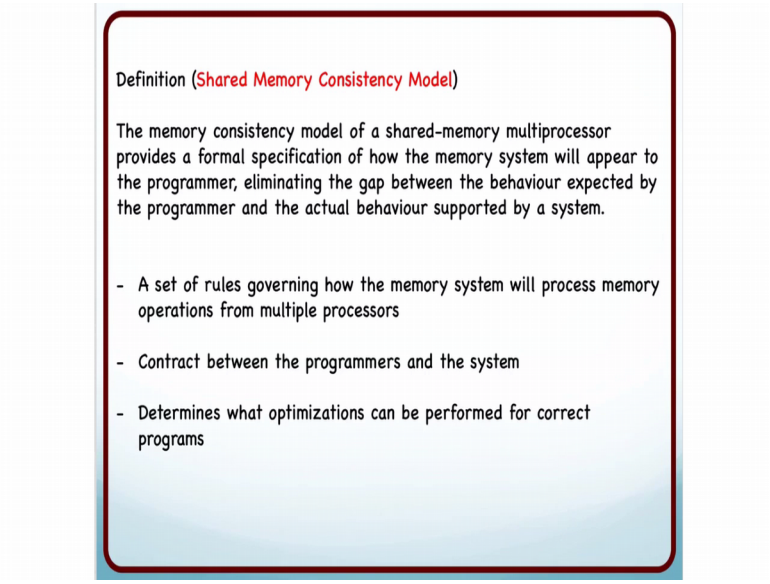
So, now what does the compiler do? If a variable is being used very often suppose I have a loop variable for i equal to 0, i is less than 1000 i plus plus, right, I do something. So,

in this case what does the compiler do? Every time it increments i it is not going to go and write it back to the memory subsystem it is not going to update it in the memory every time right it costs to go and update the memory subsystem. So, in that case what does the compiler do it keeps it in the register right that is where all the operations are happening and it knows that has to do lot of operations with this variable I it has to increment it so, often right. So, it is not going to put it back to the memory or the cache, right.

So, it is going to keep it in the register. So, now, the compiler decides what are the variables that is going to keep in the registers. What if the compiler decides to keeps flag 1 in the register let us go back to the original code or flag 2 right over here, what if it decides to keep flag 2 in the register. When it sets flag 2 equal to one, it is not writing it back to the memory right and if it does not write it back to the memory then the other process is not going to see that value. So, again this code is going to be incorrect, but again keeping variables in the registers is something you know that the compiler decides then compiler optimizes how can we take that away from the compiler. I cannot say that just because I am running on a parallel system right you are not allowed to keep variable then the registers anymore because some other process may access it right it is a big pain.

And then there are other features also for instance there are update buffers. When you write something back to the memory the processor has update buffers it may not immediately go and update it, it may get updated after some time right again. Now if you try to ensure that the update happens before I execute the next instruction that is a big penalty right you are going to lose performance. So, all of these are optimizations that the compiler and the system do for you right and if you try to enforce that in a parallel system just because you are writing a variable and now you expect somebody else might read it. So, you cannot execute any other instruction you are putting a lot of instruction on the system right the compiler and the runtime system.

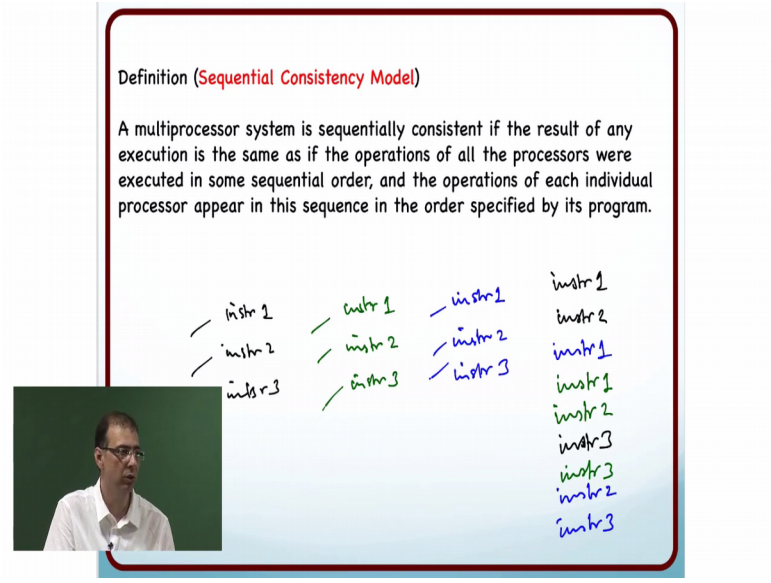
(Refer Slide Time: 12:41).



So, that is why you need shared memory consistency models. So, what is the shared memory consistency model? The memory consistency model of a shared memory multiprocessor that provides a specification of how the memory system will appear to the programmer; it basically eliminates the gap between the behavior expected by the programmer and the actual behavior supported by the system right, which kind of like an agreement between the programmer and the system that this is how I want you to behave, and the system promising you that this is how I am going to behave, right.

So, it is essentially a set of rules that govern how the memory system is going to process the memory operations from multiple processes right. So, it is essentially as we said a contract between the programmer and the system right an agreement. It is a set of rule using which then you can determine whether you are writing a correct program or not. It also basically determined that what are the optimizations that can be performed for correct programs like; what is the compiler allowed to do what can be done at runtime right all of that is decided by these set of rules.

(Refer Slide Time: 13:48)



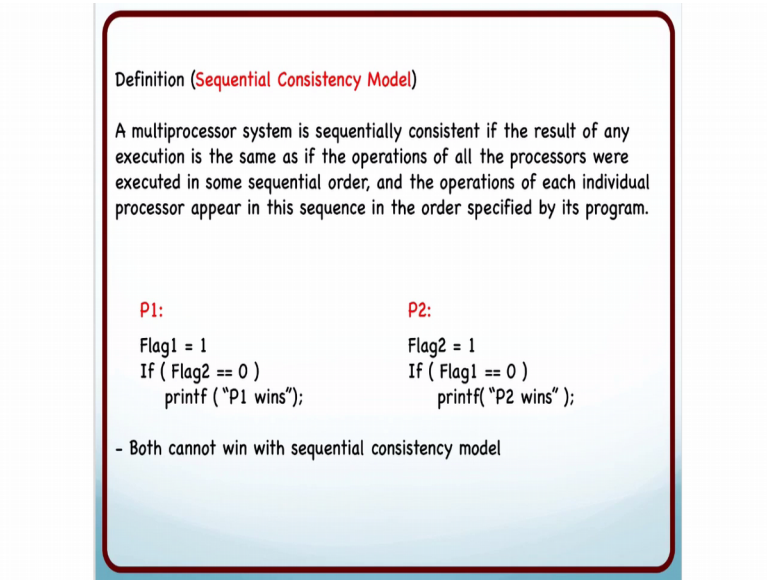
So, this is one of the most basic consistency models is called the sequential consistency model and it is very very stringent. So, what does it say? It says that a multiprocessor system is sequentially consistent, if the result of any execution is the same as if the operations of all the processors were executed in some sequential order and the operations of each individual processor appear in this sequence in the order specified by it is program.

What is it mean? It basically means that if this is my instruction 1, instructions 2, instruction 3 rate of one processor and instruction 1, instruction 2, instruction 3right of one processor and instruction 1, instruction 2, instruction 3 of a second processor and let us say instruction 1 instruction 2 instruction 3 of a third processor the output that I see should be one of the outputs of inter-leavings of these instructions with the order of each program being maintained.

So, what are the possible orders? One possible order is that instruction 1 instruction 2 instruction 3 get executed and then instruction 1 instruction 2 instruction 3 get executed and then these 3 get executed that is one order, but there are lots of orderings right another ordering is that first all the green one is then blue one in then black one is. Another possible ordering is that first the instruction 1 and instruction 2 get executed then maybe instruction 1 of the third processor gets executed, and then instruction 1 of the second processor gets executed, instruction 2 of the second processor gets executed and then instruction 3 and then instruction 2 instruction 3, right; it is just an interleaving of the instructions of the 3 programs with their order being maintained, right.

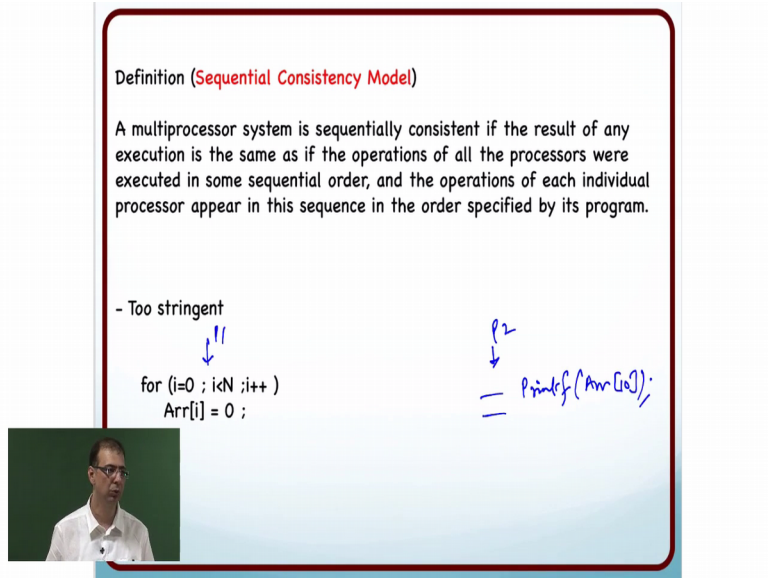
So, this is the sequential consistency model. So, as you can see there are lots of orderings they may produce different outputs, but the programmer has to understand that all these outputs are valid outputs.

(Refer Slide Time: 16:14)



So, as we just saw right. So, with the sequential consistency model what is not possible? Both P 1 and P 2 both of them winning is not a possibility with the sequential consistency model right because that is not a valid interleaving of the instructions as you see over here right. There is no way you can interleave this instruction and get that output and why is it stringent? Because the moment I write flag 2 equal to one, I wanted to appear to be visible to the other process of the moment I write flag 1 equal to one I want it to be visible to the other processor right I want any update should be immediately visible because the other processor may access them right it leaves no scope for the compiler to do optimization right very little scope.

(Refer Slide Time: 17:00)



So there is one problem why is it so stringent. So, this is a for loop for i equal to 0, i is less than n i plus plus and you are just initializing and i write to 0 there are I equal to 0 at simple piece of code. So, as I said right one very common optimization that compilers do is keep certain variables in the registers right whichever ones are access very very often they just keep them in the registers because it is too costly to go and keep on updating the memory fetching them from memory and so on right. So, you want to keep it in the resistor.

So, what is the problem here with sequential consistency model it would not even allow you to maintain i in a register why is that? Let us say this is getting executed on P 1 can you give me some instructions, which if I execute on P 2 I may get an inconsistent result which is not corresponding to the sequential order of execution of all the threads right a valid interleaving of all the instructions of the different programs.

Student: Print.

Print what?

Student: Print any value of the array.

Print any value of the array ok.

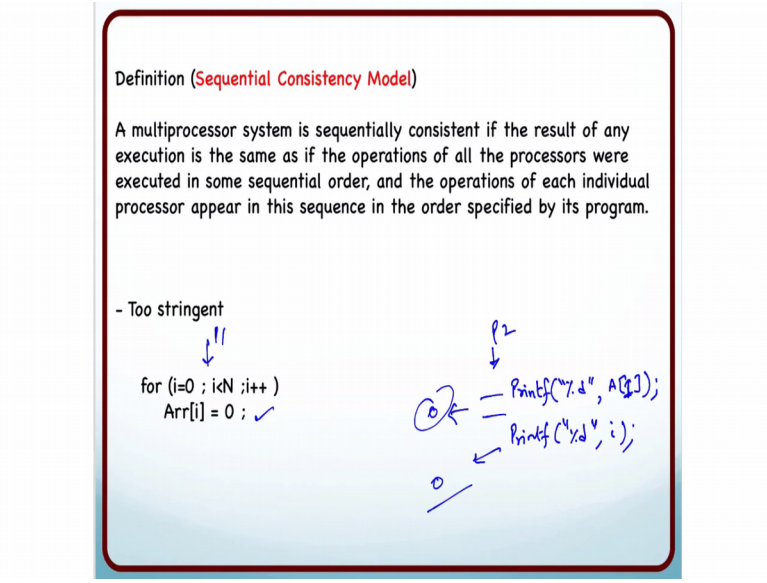
Student: So, if (Refer Time: 18:25) variable updated it will (Refer Time: 18:26) 0.

Ok.

Student: For else it will (Refer Time: 18:27) garbage value.

But that is fine right that is a valid sequential interleaving if this code on P 2 execute before the code on P 1, that is what you will see garbage. If this executes after you P 1 then that is what you will see 0, both of them are valid outputs they correspond to some valid inter-leavings of the 2 codes.

(Refer Slide Time: 18:47)



Student: You print the value of array of I (Refer Time: 18:52) memory.

Right.

Student: Or it may be some (Refer Time: 18:56) uninitialized value (Refer Time: 18:58).

But that is fine right because if suppose the interleaving is such that P 2 gets scheduled before P 1, it will print some garbage which is fine because that is a valid interleaving P 2 instructions getting executed before P 1 instructions.

So, let us see a simple example. So, let us say I first print the value of let us say this is an integer ai or not even A i let us say a or 1 right I print the value of a 1, and then I print the value of i that is it how can this go wrong? I might end up seeing 0 over here which means that this instruction has been executed and when I print i

Student: (Refer Time: 20:02).

I might get garbage or 0 why because what has processor one done, it fetch these instructions it kept i in the register. So, whatever it was updating on i it is not updating the memory, right.

So, I is garbage in the memory or maybe it would let us say it was initialized to 0 in the memory, such value in the memory is still 0 right where as it has gone ahead and initialize Arr 0, Arr 1, Arr 2 and so on right it has initialized all of these things, but the value of I has not been updated in the memory right. Now these Arr i is are not going to stay in the registers this is a million values you do not have a million register. So, these it has to keep on updating the in the memory, it has to write these back to the memory, right.

So, these array values are written back to the memory and now what does the other processor see? It see is that when it prints a one it gets the value 0 which means that it has already executed Arr 1 equal to 0, which means I must have already taken the value 1, but when it prints the value of i it finds out to be 0 is this point clear. So, what is the crux again the sequential consistency model is a very very stringent model it is not going to allow the compiler and the runtime system to make a lot of optimizations to the code to move code around right, they cannot even store loop variables in the register. So, sequential consistency model though they are very good theoretically they are not used in practice right you cannot build systems your sequential consistency models or you cannot build good performing systems, right.