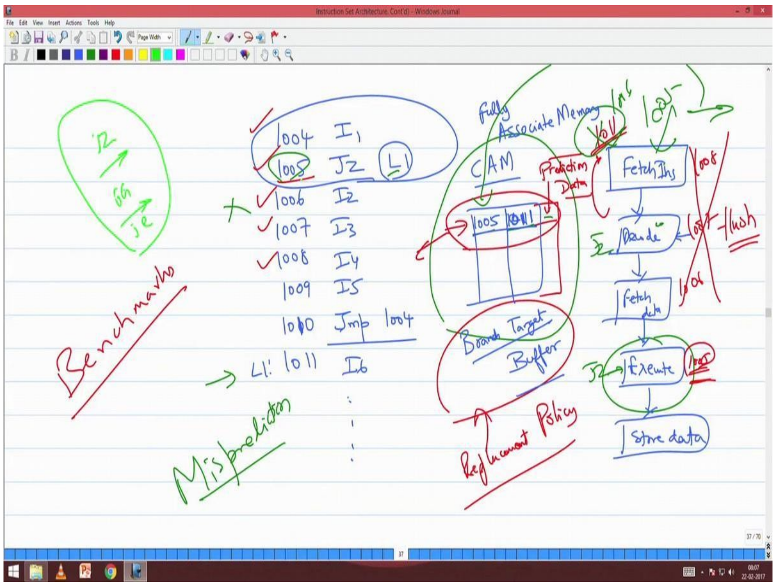
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 23
Branch prediction

(Refer Slide Time: 00:26)



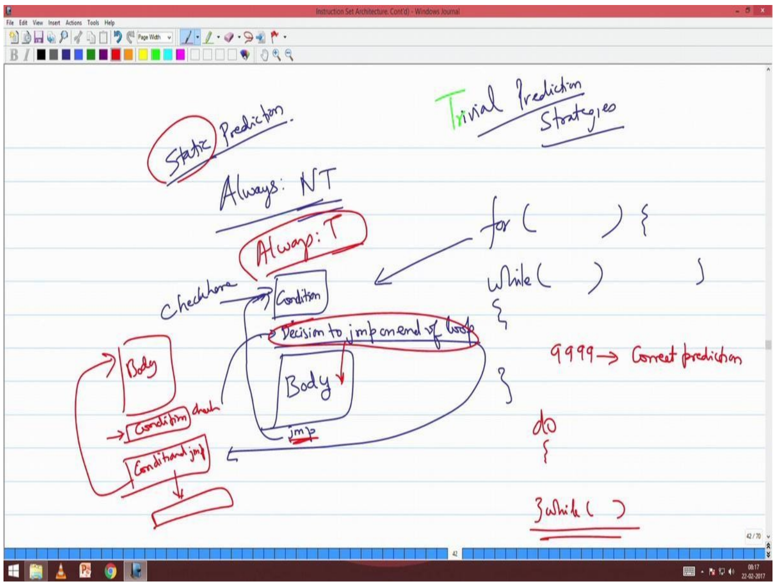
So, I basically we predict whether a branch is to be taken or not taken at the fetch stage correct. So, at the fetch stage I give the address of the you know, the particular PC. If that address is actually stored in your branch target buffer then that is a conditional jump. I have that target address of that conditional jump along with this. I also have the prediction data. From the prediction data if I find out yes it is going to be taken, then immediately I will note this target address. If it is not taken I just leave it off, it goes to the next address.

So, by the time I come to the execute stage, that is the point where I realize that the jump is, what is going to actually happen to the jump. So, far I predicted. So, if my prediction is wrong. I have say predicted it will be taken and actually it is not taken then I go and flush the pipeline. All these 3, 3 whatever I fetch the other instructions and decoder and data fetch whatever has happened further is have to go and flush it out right. Now please note that in the first 3 stages I am not changing the state of any systems, any hardware.

Fetching data is only reading, decoding the instruction does not mean I am changing any register or anything. And fetching instruction fetching data also I am only fetching the data I am not changing anything. So, all these 3 stages are read only stages. I read something I process something, but I am not updating some the context of the process. I am not updating any general purpose register nor am I updating anything. So, what does this imply? My flushing operation is going to be very, very simple. I have to just erase and put, what do you put there we put no ops, right. We have already seen what a NOP is, right. I just put a NOP there for all the 3 stages and it just progresses.

So, please note that branch prediction also crucially depends upon the fact that I can afford to make a miss prediction and quickly recover from that miss prediction. And the reason being that the 3 stages before execute are read only stage. Read only in the sense that I am not going to go and update or change the context of the process. I am not going to update the you know general purpose register and I am not going to update anything, flag registers correct. So, this is one point which makes these types of prediction mechanism possible. Now what is more pending in branch prediction is how do we arrive at the prediction data.

(Refer Slide Time: 03:14)


Instruction Set Architecture, Cont'd) - Windows Journal
File Edit View Insert Actions Tools Help
Trivial Prediction Strategies
Static Prediction.
Always: NT
Always: T
for ( ) {
while ( )
{
}
Check here
Condition
Decision to jmp on end of loop
Body
jmp
Body
Condition check
Conditional jmp
9999 → Correct prediction
do
{
} while ( )

So, let us take some trivial prediction strategies. Suppose I say always it is not taken. What will happen? Who will benefit? Always not taken.

Student: When will be there no flushing?

Ah?

Student: When there will be no flushing of input.

No, no that flushing out has nothing to do with prediction. If we miss predict then you will flush out.

Student: It is hiding problem then this in miss prediction, because taken is better than not taken.

Taken is better than not taken ah.

Student: (Refer Time: 04:06).

Think yeah think.

Always I say not taken, no problem. This just, this is what we call as a static predictor. Who will benefit? How will even think about an answer? Suppose I asked this in the quiz no, how will you think of an answer for this?

Student: In the loops.

Loops yeah correct. So, very good. Next, which loop will benefit from always not taken.

Student: Entry control.

Entry, which entry, entry control, entry control in the sense.

So what, can you give me a example of an entry control loop? For loop. So, if I have a for loop or a while loop. What you do is in the for loop the way, see you can translate it in any way, the correct way of translation would be that I check here. Check what for a condition, right. If the condition so this is checking would be something like a compare instruction etcetera. And here I make a decision to jump on end of loop. I will make a decision to jump on end of loop here. And here would be the body of the loop and your jump actually will jump here also.

So, here there will be an unconditional jump which will take you back to this checking condition. So, this is the structure of how while loop or a for loop. So, there will be some condition check to start with and then a decision to jump, so this will be the conditional jump here, will be the no conditional jump here correct? And then I have the body of the loop and then yeah unconditional jump which will again take at the end of this. So, this is how a for loop can be translated. One of the ways by which for loop gets for and while loops.

Now what you see here? The interesting things, so you have done the translation, right. You have done generated code in your previous course in that. So, this is how we did it you remember [FL] now assuming that the for loop is going to execute for say 10,000 times in 10,000 times 9,999 times you are going to have correct prediction. Because 9,999 times this jump will not be taken, the body will be executing. There is exactly one time where the, there is exactly once that is at the end of loop where the, this condition becomes true and I jump out. The remaining 9,999 times I will not be taken. So, my prediction is correct for 9999.

Suppose I say always taken, that is true when I have something like you know, do while that is I want the body of the loop to execute once and then take do while. So, if I do while what happens. So, I will have the body and I will have a condition check here and then I will have the conditional jump here which will basically take back. So, for majority of the time what happens I will be taking this, the branch will be taking I will go back to the body only once I will come back. Do you follow this? So if I have an always taken type of a criteria then what happens? Then this works. If I take always not taken then for and while loop works. And these are major part of your program, right. For while and do are major. Other than that there could be some internal if and else statement or switch statements, right, which are different.

Now, so I have taken not taken, so I, So these are basically static prediction because I do not change the prediction at all. I do not change the prediction, at the regular interval. So, I just do this, right. I just fix the prediction. So, that is why I call it as a static prediction. It is fixed in the architectural and every time it is not taken or taken. So now, how will you decide? Suppose you are saying that I do not want to So much on branch prediction. I do not want to complicate the hardware.

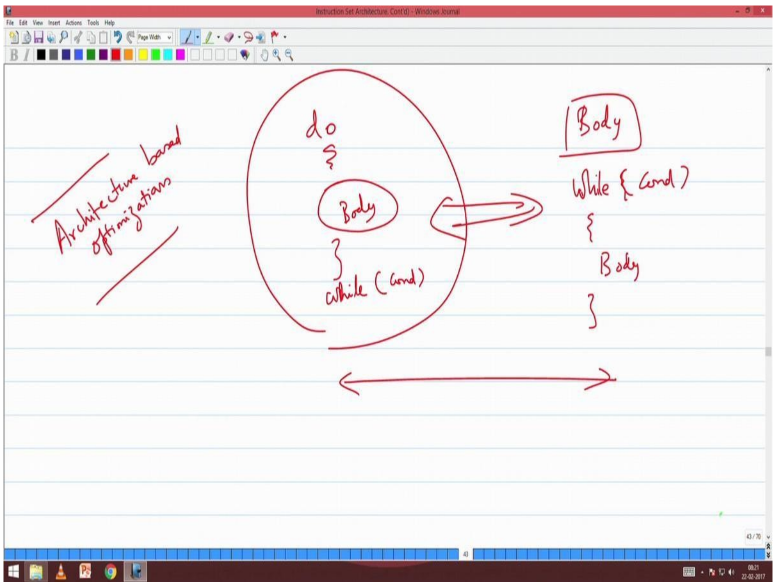
Now you just, so you are given an mandate to do either static either not taken or taken as a static prediction, right. Now how will you decide whether to put not taken or how will you decide how will you put taken? And how will you interact with the compiler guy? So, there is one compiler person then you have the architect, now you are given the mandate you should only do static prediction correct. So, either you have to decide I should put not taken or I should decide that I have to put taken as a prediction. Now you have to interact with compiler guy, now how will you make this decision? How both of them together will sit and make the prediction? Got my question? Yes give me the answer. For what program?

Student: (Refer Time: 10:32).

Correct. So, that is a interesting question. So, the compiler guy will tell you decide and tell me, to whom? To the architect. You decide and tell me what you want to do. So, the architect will say always not taken it is preferable. Then the compiler should translate every do while statement to a while do statement, correct? Or vice versa I will say always taken, he is very happy with always taken then he has to convert every for and while statement to a do wile type of (Refer Time: 11:15) so, how do you convert a do while, which is easier? For while for and while construct to do 2 while construct to for while?

Student: (Refer Time: 11:30).

(Refer Slide Time: 11:33)



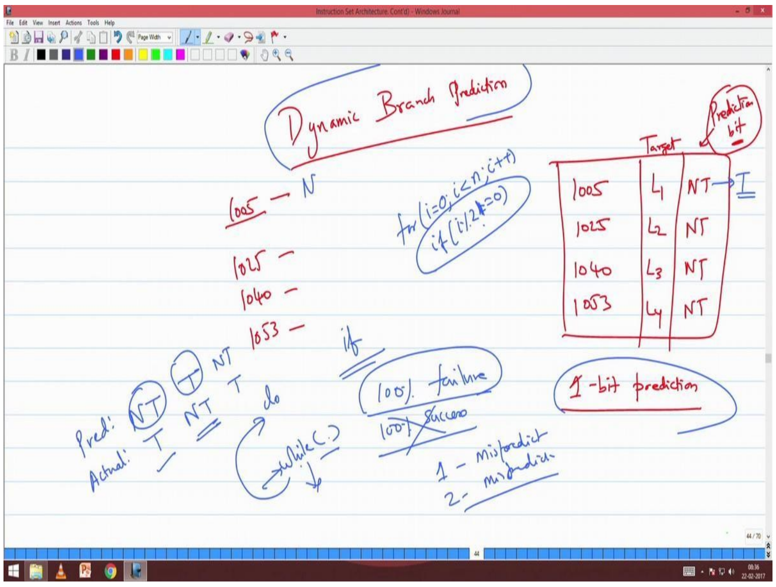
So, how do you do it? How will you construct, convert do while to while do? So, that initial condition one code you make up then put while condition then body. So, this is a transformation that you do. I will move one version of that body I will put for the first 2 first condition, that I will put in above, then I will put while body. So, this is how I can convert a do while to a while, but the other ways slightly tricky. Now there is some other if branch.

So, so what do you get here? So, so what I am, this is the first thing that we are starting off on what you call as something more on architecture based optimisations. I have this so, I say I have an ARM architecture, I have a Intel architectural, I have some RISK-V architecture, right. I have the Shakthi architecture from IIT Madras. I am writing a compiler, I want to optimize it what does it mean to optimize something for an architecture, right. This is one example there are (Refer Time: 13:14) number of examples, but this is one thing that we could start off.

Now, this is all about static prediction where I fix it for once and I do not change. And that will work for 99 percent of the case, now whatever we are going to talk further is just about one percent of the case, right. In my opinion we can just throw it off and still it will work. You will not get some big you know hit there, right. And for every prediction mechanism I could get a counter example which kill it, but it will not be the real case right.

So, it in normal programming environment you may not get something that complex. So, I specifically write a program to kill an architecture that is that will not happen in practice. The serious stuff about branch prediction NCR. If the compiler does this and all these do while for loop. So, if you take any programming language you will learn these types of constructs right. So, if you so we can generally characterise as entry check or exit check right. So, if everything is going to be entry check then I will take not taken it will work, if everything is going to be exit check if I take if I put taken as a static prediction it will work. But what beyond this? Now let us go to something called the dynamic branch prediction.

(Refer Slide Time: 14:43)



So, what is dynamic branch prediction? I keep changing my prediction for every branch. Let us say in a program I have 1005, 1025 1040, 1053. So, let me say there are branches at these points. So, I am so in your branch target buffer I will have these entries 1005, 1025, 1040 and 1053. I will have some target address namely L 1 L 2 some address to which it can branch. And I will have a prediction bit. Normally we assume that initially they are all not taken, right. Now since I said prediction bit this is a 1-bit prediction 1-bit prediction. Now what will happen is always the prediction bit will be adjusted or will be changed to what happened in the previous instance. If it is always so if in the previous instance if I am executing 1005 initially it is not taken, if it is going to be not taken it will be continuing in not taken, but if it becomes taken then you will go and change it change this to this will become taken. So, whatever happened in the previous execution of that branch will be the prediction. So, this is what we call as a 1-bit prediction ok.

Whatever happened in the previous execution of this branch will be taken as the prediction. So, it can become taken next time it can become not taken. So, so this type of a dynamic branch prediction will be extremely useful when you are having these type of this if statements, if else statements. So, can you give me; so for a 1-bit predictor. So, how will I, so can give me an instance where this will have 100 percent failure? And an instance where it will have 100 percent success, a 100 percent success is trivial, just get a branch it will give always 100 if 1 equal to 0 or something like that. So I can, but can you give me something where it will have 100 percent failure. So, initially this branch is not taken. So, this is the prediction and this is the actual action. So, this becomes taken. So, essentially next time when this branch is executed it will say taken, but actually it should be not taken next time when it comes it will say not taken, but actually it will be taken.

Can you give me a realistic example for this?

Student: (Refer Time: 18:22) if we take up any number which is divisible by 2.

Exactly good beautiful.

So, if I have if I have a for loop. So, so little more practical way of handling your answer is absolutely correct, but still you can answer like this. And I go and say if i mod 2 is equal to 0 something I try. And use ensure that the code that is generates generated will take if will take if i mod 2 is 0 we will take this condition is satisfy. Then what will happen? So, I can say i mod 2 is equal to 0. So, started 0, it will predict not taken, but

actually it will be taken next time when i equal to 1 it will be actually it will predict taken, but this will be not taken. So, this is the 1-bit prediction and this will basically, there is an example there is an trivial example which can kill this prediction. Now, but where will this 1-bit prediction be very useful can you give me an example where one. So, why I teach 1-bit prediction if it is where will be.

Student: Both the loops and compiler both the loops.

Both the loops and compiler. Both the loops that is entry check and exit check loops exist and the compiler does not do any optimisation ok.

So, where will where will you get why should the compiler not do such a thing? Today you sell a software, right. It execute for next 5, 10 years, correct? And what would have happened, just imagine you assume not taken and you released a software for a nice architecture. And suddenly you start fighting with that architecture fellow like this company and that company has a huge fight. Or somebody else bribed that company to kill you he has to not do anything that next time he creates the architecture you assume not taken, right. I will put taken there that is all and if the architecture actually becomes very powerful, I am talking political things here if the architecture actually becomes very powerful then your software receives to, assume the architecture is ARM and you are a simple you know one app, people carry damn for it. So, you are completely gone. You have made a compiled executable and now you say you have you have made that you know not taken as this.

And then finally, if you compile that and put here then this fellow changes the strategy then you have to go and redo your app and release it again, right. And if it is a commercial app, right. You may not be in a position to immediately redo. So, lot of issues come. So, if your app is going to run damn slow for say next one week, right. People will discard your app and take another app. So, if I am an architecture company and you are an application and you are trying to dominate over me. So, what should I do? I should have a software team which can write a software as good as your and then I will go and change some architectural parameter. So, that your app gets good, so enough.

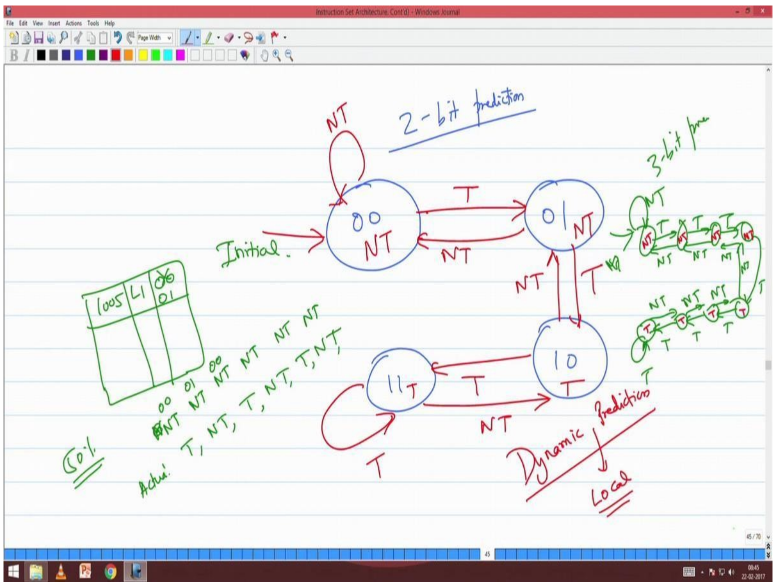
So, traditionally people will not depend upon an architecture they will make it architecture independent right. So, if you study the x86 operating system next time. You will see that none of the features none of the great features of you know x86 are used in that Linux. For example, segmentation is not used at all. There will some segment, but it is not used it in full glory that what we are talked off, right. For example, there will be segments that are merged for example, code segments tag segment everything will start from a same address and we leave it to the operating system to distinguish between where the code should be and where the tag should be.

So, I declare the entire 4 GB space as you can store code, you can store data, you can read write on it, you can store stack everything. And then finally, you know the operating system will demarcate this area and it will manage it. By this what you will lose? You lose entire security that is offered by the Intel architecture. But why did people write such a code? Because they do not want it to be Intel specific, right. They want it to be they want to quickly port this o s on to a sun spark or into a ARM or into a another architecture. So, there has been a constant threat that if the architecture dies. Or if the architecture guy becomes hostile to me, what will happen to may software? What will happen to my operating system? Right. If it is open source operating system effort then I care a damn for the architecture because if that fellow dies my entire effort will go off. So, I do not want to be specific to that architecture.

If I am a close source architecture I have a commercial interest in that, right. So, I want to make one software which will run on all the architectures that were in the past currently executing that may be in the future. So, I am looking at things like that right. So, so when you come, So when you come get into these type of issues please note that you know compilers will not do something it is very, very specific for an architecture, right. We will talk more about this as we proceed in the course, but that I have given this background can you tell me an example where 1-bit prediction can help? So, the simple thing, right. If I have a do while statement right. So, if I have a for and while statement then what will happen all the not taken if I say not taken is default, then the 1-bit prediction will work beautifully 99, 0.999 percent it will correctly and there will be only one miss prediction.

But in the case of a do while statement, what will happen? The first time it will take it will miss predict. First time there will be a miss predict, because it will say not taken it will say at this stage it will say not taken, but actually it will be taken. Then from the next time onwards this will become taken it will continue to be taken. And only at the last time again it will say taken, but it will fall down which is not taken. So, there will be 2 miss predictions totally in the case of a do while loop. While in the case of a in a for and while loop there will be one miss prediction correct. So, in that context I need not do static, I need not depend upon the compiler to do any optimisation for me, still I can basically get things working. So, 1-bit predictor assumes a dumb compiler and still can get very good predictions for both for while and do while. In the case of for and while I will have one miss prediction, but in the case of do while I will have 2 miss predictions. Are you able to follow? So, that is why 1-bit prediction becomes interesting.

(Refer Slide Time: 26:29)



Now let us go into 2-bit prediction. So, what do you mean by 1-bit prediction here I have 1-bit here in this, right. Now we will have 2-bits here. So, 2-bit prediction essentially means, what are the 2 possible bits values of the 2 possible bits? I got 0 0 0 0 1 0 1 1. 0 0, 0 1, 1 0, 1 1. Now we will do a state machine for this. Now let us do the state machine for this. So, initially I will start this is not taken. So, the prediction here is not taken the prediction here also not taken, taken, taken. If I am in not taken 0 0 not taken state, if I again if actually there is a not taken there, if that branch again it is not taken I will maintain 0 0. If it is taken from this state I will go to not taken, again 0 1 not taken. Again if I get not taken I go back here, but if I get taken I come here. And here if I get taken I go here if I get not taken I go back here. And as long as I am taken I will be here not taken.

So what I have written inside the circle is the prediction. What I have written as the labels of the edges is actually what is happening to a particular branch, are you able to follow? Right. So, this is a 2-bit prediction. Now what happens, initially I will say not taken. So, this is the initial condition. So, initially the branches not taken. So, I will predict for this, is for every branch please note that is for every branch. So, say 1005 there is a branch, target address is L 1 and initially it is not taken. If the branch is not taken again it will remain at 0 0. So, I will predict not taken first if the prediction is true then I will remain at 0 0, but if my prediction becomes false I will go to the state 0 1, but there also I will predict only not taken.

Again if my prediction is wrong that is it is taken then I will go to the taken state. Then I will and again if I get not taken I will come back to this not taken state. Again if there is taken I will go to this taken state right. So, this is how. So, at least 2 times I have to see not taken. So, that I decide on not taken, right where at least 2 times I have to see taken before I decide on taken. In the previous case I just saw ones if it is taken I agree with him, now I say let me do twice then I will agree with him.

So, can you give me an example where 1-bit prediction becomes 2-bit, but 2-bit prediction becomes better? So, can you give me an instance where 1-bit predictions screws up while 2-bit prediction works. So, what happens last day even example we had the actual thing was taken, not taken, taken, not taken, taken, not taken. This is actual now what would have happened? We started off it not taken, then it was not taken please note that this succeeded. And again I it went from taken so it was 0 0 and since it was taken this became 0 1 and I predicted not taken. Again I get taken so I predict not taken oh again I get not taken I again go back to 0 0. So, this is not taken. So, this is not taken. So, this will go on like this. So, 50 percent of the time I will succeed where 100 percent failure so I bought it down by off. So, my program will work with respect to branches and all it will work twice faster ok.

Can you give me an example where it will fail 100 percent? Yeah you should adjust this modulo no? Instead of modulo 2 if I make modulo 3, modulo 4 we can give it so. So now, the next thing is I can now go to a 3 bit predictor. Which will have 8 1, 2, 3, 4, 5, 6, 7, 8 initially it will be not taken to start with. So, every time I get taken I go to the next stage. And every time I take not taken I go back. And here not taken I remain here taken I remain here all these will predict taken while all these will predict not taken all, right. Now so, there is one of the papers which basically says that a 2-bit predictor is as good as k-bit predictor. We need not go through 3 bit 4 bit and all I can keep on extending each one will give me an another paper no question, but then somebody put a stop to the somebody got failed let us put a stop to this.

2-bit predictor is best it is as good or better than k-bit predictor k greater than 2. So, that was the end of the story. Now this is what this is dynamic prediction. Now but this is dynamic local prediction. Why local? Why that adjective local? It is the decision is made rather than the way the decision is made based on the behaviour of that branch. The prediction decision for a branch is made based on the behaviour of that branch and not any other branch, right. I have 1005, how do I predict for 1005? I take the history of how 1005 behaved in the previous instances.

The decision for 1005 is not influenced by anybody else. Decision for 1005 is not influenced by anybody else or is not influenced by the decision of other branches. And that is why I make this that is how I make decision. So, that is why we call it as dynamic local prediction, right. Right so, that means in Mondays next Mondays class we will have dynamic global prediction right. So, the moment we introduce an adjective there is something about that adjective. So, so what I want you to think out is why there is a necessity for a global prediction.