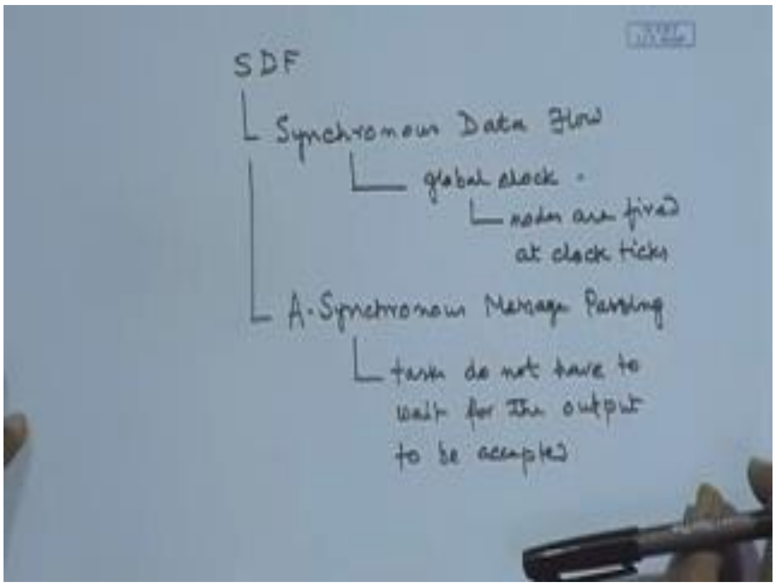
Embedded Systems Design

Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 39
Data Flow Model – II

So, in the context of data flow models, we have seen briefly the con process network right, it is difficult to analyze that, because of that buffer issue.

(Refer Slide Time: 00:40)

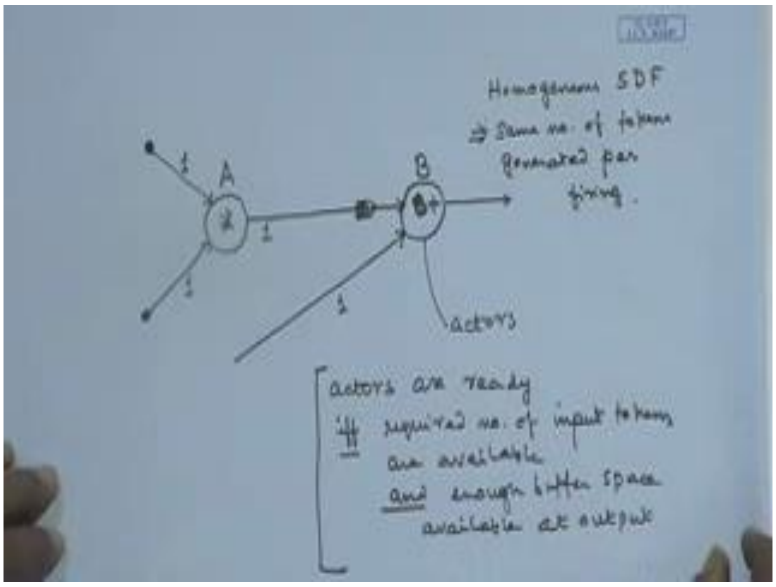


SDF
- Synchronous Data Flow
  - global clock.
    - nodes are fired at clock ticks
- A-Synchronous Message Passing
  - tasks do not have to wait for the output to be accepted

And now we will be discussing about another modelling language, which is known as SDF or synchronous data flow. That means, whenever I say synchronous; that means, there is A global clock, and A global clock ticks, and at the global clock, the nodes are fired. Nodes are fired at global clock, fired at clock ticks. At the same time, we use. Let me write it in this way; asynchronous message passing for communication. So, it may appear to be little confusing, that we are calling it synchronous dataflow, but this asynchronous. The message passing is asynchronous right. So; that means, what we mean by asynchronous message passing is that, the tasks do not have to wait, until the output is accepted. This means it gets the data; it produces some data it sends it right. At every clock it goes on sending it. Now it is not necessary that it will wait for, that to be accepted. So, now, SDF will give some examples of SDF.

Student: (Refer Time: 03:00).

(Refer Slide Time: 03:10)



Homogeneous SDF
⇒ Same no. of tokens generated per firing.

actors are ready
iff required no. of input tokens are available
and enough buffer space available at output

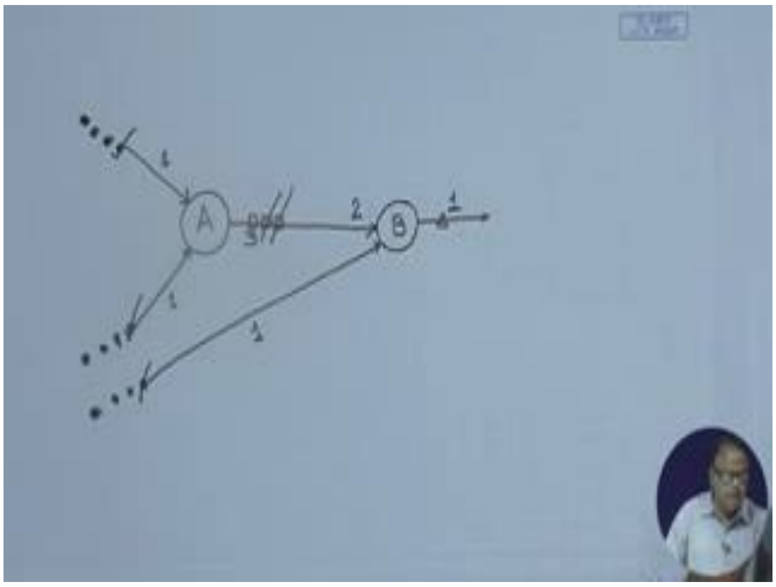
It is. So, let us look at an SDF diagram. Suppose we have got a task A, which is A multiplication, and the task B. Sorry or task B which is an addition all right. And A is getting input from two data sources, and B is getting the output of A and something else. Now, these nodes are known as actors. These nodes are called actors. Now the actors are ready, when the necessary numbers of input tokens are available, and enough buffer space is available at the output. Actors are ready if and only if, required number of input tokens, are available and enough buffer space; space is available at output.

Now we can have two types of. Now is it clear; say for example, I have got required number, say I can put in weights to this edges, what does it mean? It means that this input only one token is required at this input; at this input also one token is required. So, if I have a token here, and if I have a token here, then this is ready to fire. And suppose there is no token here, I am saying here also one; that means, one token will be produced after multiplication, and no token is there so therefore, buffer space is available, if I fire it, it will come. Now, suppose some token is available here. I am showing another instance, some token is available here, and no token is available here. Therefore, this is not firing, because then, and there is also one all right. Now this one can it fire, suppose the tokens are here; 1, 1, no it cannot fire, because there is not enough buffer space, this is not consumed all right. Now there can be homogeneous SDF and non homogeneous SDF. What is meant by homogeneous SDF?

Student: (Refer Time: 07:03).

That is what; I mean we will have to manage that we will show that, I mean that if the buffer space is not available then it would not fire at all. Homogeneous means, homogeneous SDF means, that there are same numbers of tokens. The numbers of tokens accepted at different node points are all the same. Say for example, everything is one, the same numbers of tokens are firing, and non homogeneous, this implies the same number of tokens generated per firing. So, if this fires there will be one token generated, but that is not the general case. We can also have non homogeneous and in fact, we will be looking at the non homogeneous once more.

(Refer Slide Time: 08:28)



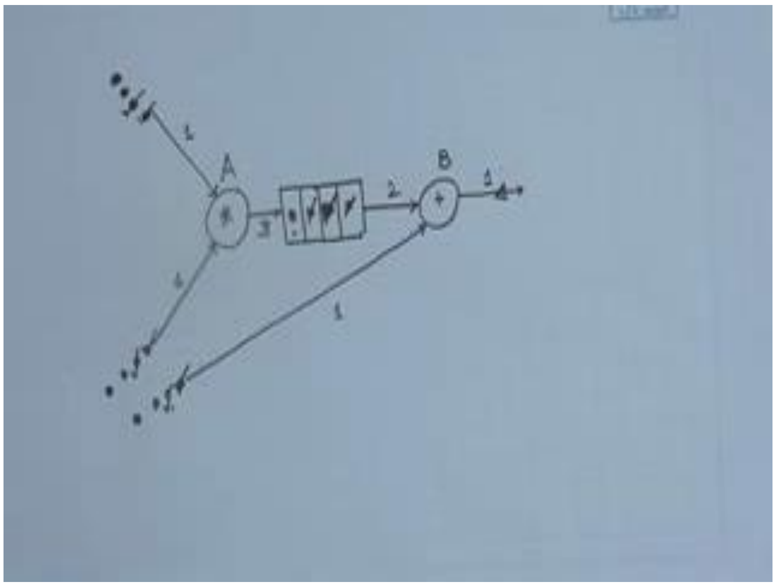
So, for same actor, I am just showing an example. say for example, this is a typical one all right, and here I have got the edges 1, 1, say 3; that means, if this one, this one fires this is A multiplier, if this one fire it will generate three tokens. Whereas, this is 1, and this one consumes 1 tokens and it produces 1 token. This is a non homogeneous scenario right.

Student: (Refer Time: 09:20).

No I have not yet given the buffer space, right now this I have not put in any buffer, let me proceed, I have not put in any buffer like. So, I am just saying that this is A non homogeneous. So, what happens? If I now put the buffer space; now suppose I put in tokens here, there are four tokens here, there are four tokens here, and four tokens here and there are four tokens here. What it means is, this has got the requisite number of tokens, why requisite number of tokens, one is required. So, as this one fires, this one is consumed, this one is consumed, and here I will produce three tokens.

Now, this one is ready, why because this needs two. So, this one will be taken up, this one will be taken up, this one will be taken up, and one token maybe you will be produced here right; that is the flow. Now two are being taken, now what will happen? This one, can this one fire, can A fire again, why can you fire again, this one is available, this one is available, but is there enough buffer space. At least three can come here so there is buffer space, at least two can come in right, but not at least two can come in, as soon as it fires, three will come in here, three will come in, because this is producing three all right.

(Refer Slide Time: 11:38)



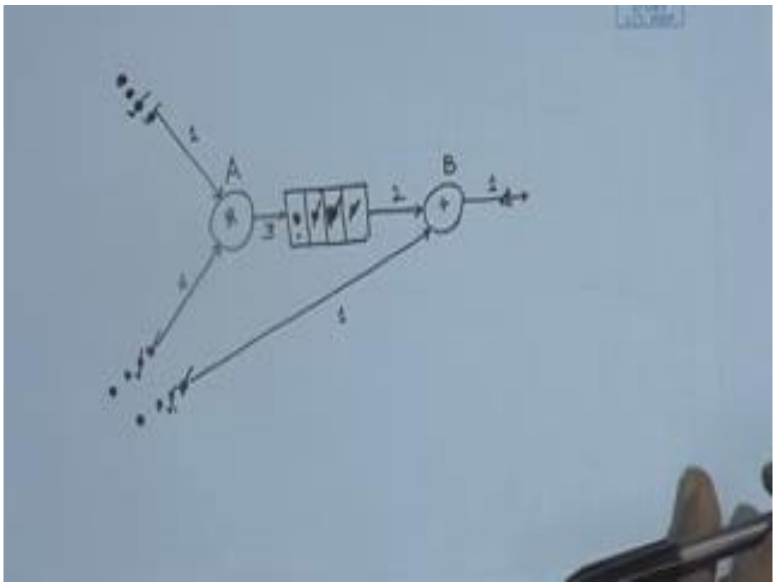
So, now let us put in A buffer at this point. So, the same thing, so I put in A buffer here. So, this is three tokens, this is one, this is one, this is one, this is two, now you understand, but here I am putting two and here it is one. So, here again I put in four, now all of you can yourself workout and animate it, to see what will happen, four here, four here, four here. Now what will happen in the first shot? In the first shot what will happen, this one is ready. So, here this buffer is completely empty; therefore, this one will be consumed here at four. So, one will be consumed, this one will be consumed, and

I will have three tokens here, coming here. And since I have three tokens coming here, this one will fire, this will consume one, and it will consume two from here, and one token will be produced here right.

Next this one will fire again, A will fire again, B is ready and B can fire, so B has fired already. Now here B cannot fire right now, although A has got, but here I do not have two; therefore, again this fires, this one first, sorry this one also fires, and I put in three here. So, let us put in, again three is there, this one is not there all right, this is not there, so three are there. So, again these fires this goes up and two will be consumed from this. So, this goes this goes, and one will again be produced here, in that way it goes on all right. Ultimately in this way as it goes on, ultimately what will happen, ultimately there will be a situation when the buffer is empty, everything is being consumed.

Now, you see the buffer is not being full, because if there is no buffer space, if there will be four; suppose all the four have come here, then I am not firing this any longer right. So, I am not being able to fire this any longer, because the necessary condition is, that the space must be available in the buffer space. So, if you animate this repeatedly, then you will find ultimately, say for example, at a particular point.

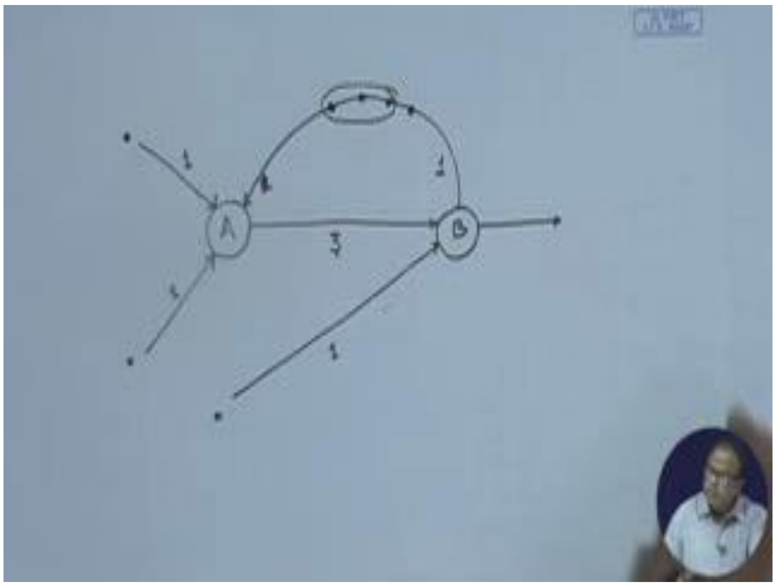
(Refer Slide Time: 14:31)



We will have to see this situation will come, that we have got three tokens left here; one token here, one token here, and four tokens here. At that point, so this is 3, this is one, this is 1, this is 2, this is 1, this is 1. At that point, who is ready to fire; A is not ready to fire, B is ready to fire right. So, as B fires this one will go. So, this is ready to fire, this one will go, and two go out, as B fires. Now, at that point can A fire. A cannot fire, because there is no space. Can B fire, B can fire. So, B will fire here, these two will be consumed, and this. So, that is a complete period that the buffer has become empty; that is how typically SDF works. So, as one period is complete we will again start.

Now the question is, there is an interesting point here, that how do you model the buffer, are you going to draw the buffer. So, if you just think A little bit, it will be immediately clear to you, that we can very nicely model the buffer in this way.

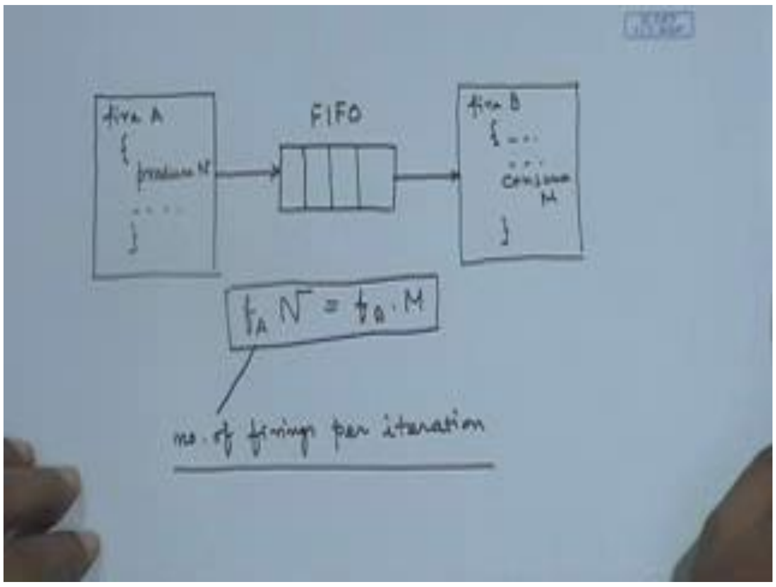
(Refer Slide Time: 16:12)



Say my buffer space was four. So, I have A token here, I have A token here, and I have A token here and. So, I put in the weights 1, 1, 1, 3. Now, if I put a reverse edge here, and put in four tokens at this point, what happens? Because, my buffer space was four, so at any point of time, now what does this mean? This means whenever this fires, A token, this will fire, when this one, one will be consumed, and one will be consumed from here. What are the input data points? 1, 2, 3 now; this signifies that as this one fires, one of the buffered space is consumed. These tokens are actually showing how many buffer slots are available empty. So, this will be there A fires.

As B fire it will produce a token here, it will also produce a token here; that means, as B fires, it is creating an empty space these by default is 1, is by default is 1, this is also 1. Will this be 1, this will be 3, this is 3 that is why I am not showing it, this is 3. So, when this one is firing, actually I am consuming three tokens from here 1 1, I am producing 3 tokens; that means, three buffer spaces are gone right and then, I have got 3 here I have got 1 here, and B can produce, and as B produces B puts in B consumes two right, B was consuming two. So, then two it returns again here all right. In that way we can model the buffer.

(Refer Slide Time: 18:49)



Now, one of the conditions, now let us look at this diagram; say I have got two processes, one is A and B I was showing. So, I am showing fire A. I module it as A producer consumer problem, and B is fire B, B is consuming right. So, consume something M, is producing N it is consuming M all right. You are aware of the producer consumer problem typically, and here is A channel, and channel is nothing, but a FIFO buffer. Now it is synchronous, but again these are the two processes. Now they can be multi date, there can be a multi rate model; that means the number of tokens being produced by this and the number of tokens consumed by this, since it is asynchronous, they may be different right. Therefore, in order to have a balance in this data flow, what is the condition that we need to meet?

The condition is, if I have, say f A, if by f A I denote the number of tokens produced as. Sorry, N is the number of tokens produced all right, here. And f A is the number of firings per iteration, then I am producing so many right. This must be equal to f B M. If this condition is satisfied then there will be a balance. So, this is a very important condition for SDF, and the advantage is, it is decidable. I can find out what is the buffer memory requirement, by looking at the multi rate, I can find it out much more easily, which was not that explicit in Kahn’s models ok.

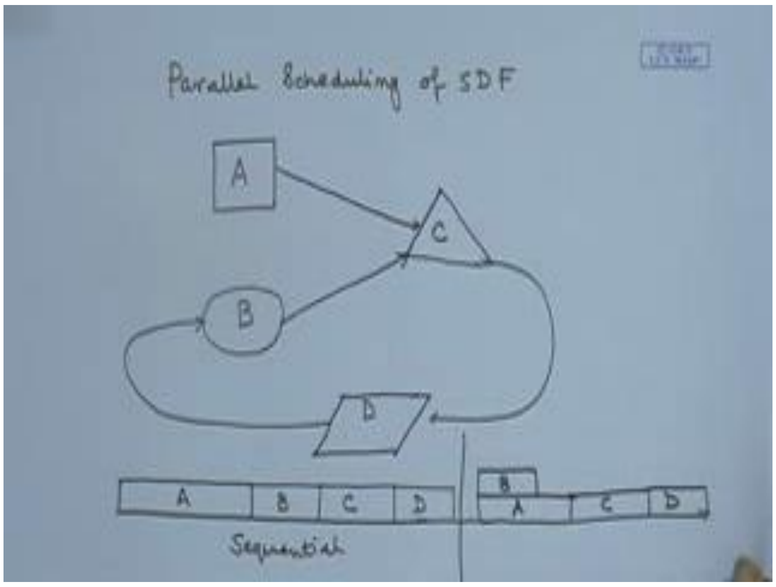
Student: (Refer Time: 21:44).

Which one?

Student: (Refer Time: 21:48).

Yes this is a balance, absolute balance. So, as long as it will less than equal to it will run fine, it is a perfect balance equation. So, last I will conclude SDF with an example of the possibility of parallel scheduling in SDF models.

(Refer Slide Time: 22:11)



Say for example, I have got a task A all right. I have got another task B; I am denoting in different shapes, another task is C all right. and the data flow is like this; C depends on A and B, and C produces something to maybe another one, which is D, and D again feeds to B all right. Now, I want to schedule it, you see once I get the data flow, I can schedule it in this way that I can first schedule A. So, maybe I draw the schedule part here, here is the time. So, I can schedule A, then I can schedule B, then I can schedule C, and; obviously, D can be scheduled after C, so maybe D; that is A sequential schedule, this is A absolutely sequential schedule.

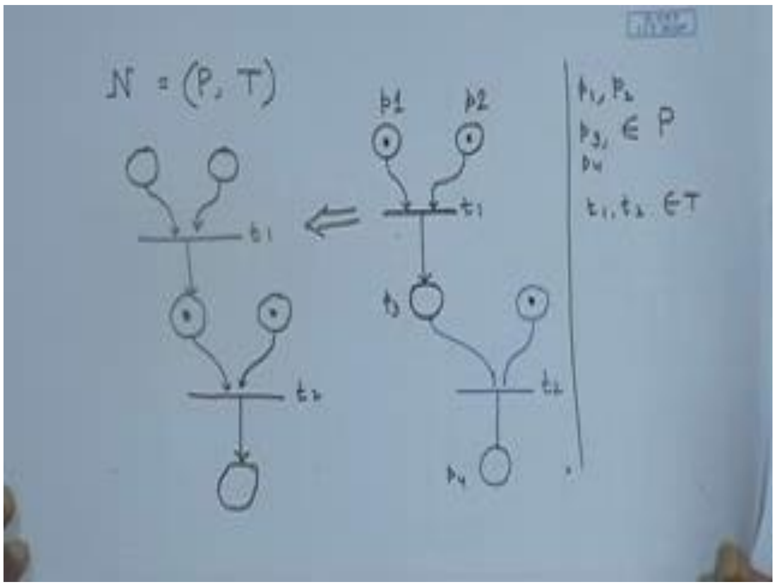
But my data flow graph tells me about the dependence, and therefore, I could have arrived at a parallel schedule, where I could have scheduled A and while A is being done I could have schedule B, and A or B are both finished by this time. So, I next schedule C, and then I have to schedule D. Thereby I can save the total execution time, and that is very much clear from this data flow diagram.

Student: (Refer Time: 24:46).

No initially the first time, first time, initially these. I am starting from this point right. So, I am starting with A and B, some data is already there, because D has not yet executed otherwise it will not start. So, A and B starts and after D is, next round D can only start, B can only start after D has completed, then the condition is satisfied. So, SDF is A little easier to analyze, it is analyzability is better; it is implementation efficiency is better, but it is expression power is very restricted. And we are always have, we will see that, we have to make a balance between these two.

Now, before concluding this part, I will just introduce you to the notion of Petri nets, and later on if there is scope I will discuss about their application of Petri nets, but here I am utilizing about 5 10 minutes, just to tell you about what this Petri net is, because you can find it handy in many other things.

(Refer Slide Time: 26:18)



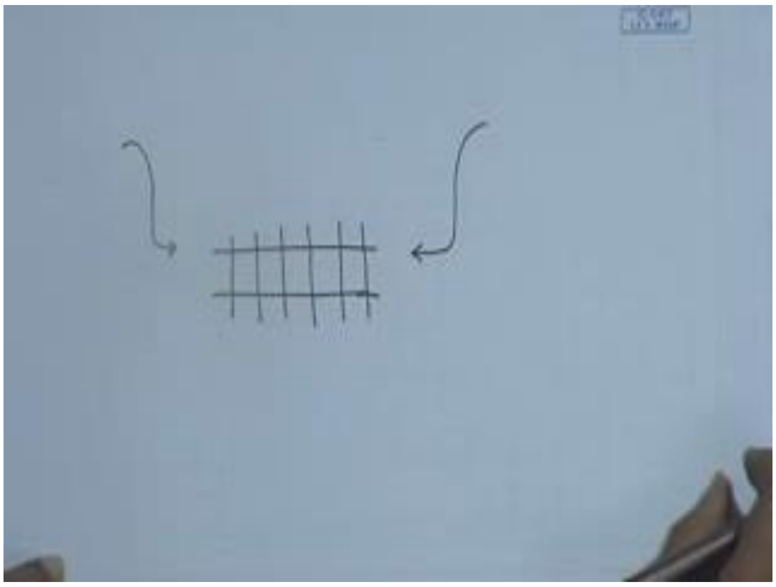
A Petri net, I am very informally telling you Petri net, Petri net is A net, consisting of places and transitions all right. There are different forms of Petri nets, but let us start with this, let us see for example, these are two places p 1 and p 2, so p 1 and p 2, and maybe p 3 also will come, now belonging to the set p all right, and there are some transitions. Just like A graph I have got some transitions say; t 1 t 2 belonging to t, t 1 is A transition that fires when, there are tokens available at both its inputs p 1 and p 2, and one variety of Petri nets says that this transition t 2 will fire, or will be enabled, if the output place, these are the input places of t 1, this is the output place of t 1, if the output place is empty.

But there are other varieties of Petri net, which at associates some capacity with each of these places; that means, how many tokens can be there. This is very similar to our buffer, thing that we are thinking of. So, simply speaking right now let us assume that, by default it is capacity is one. Therefore, this transition can fire, only when both these inputs are, both these input places have token, and this one does not have a token, so this will fire. As it fires, this will come to that these tokens are removed, and the token will come here.

So, this will be transformed to as soon as this firing is done, the scenario will be this. And maybe in this transition we had, something like another transition t 2 which is also in t, and there is A output place here. So, p 1 p 2 p 3, this was p 3, and this is p 4. Now, as this fired, then what will happen. This, suppose this one already had A token here, but this could not proceed. Now as this fired, there was A token already here. So, this transition will also fire, and as this transition will fire, will have these two tokens removed and it will come here, and this is completely asynchronous. There is no need of A clock as soon as the tokens are available this transitions may fire.

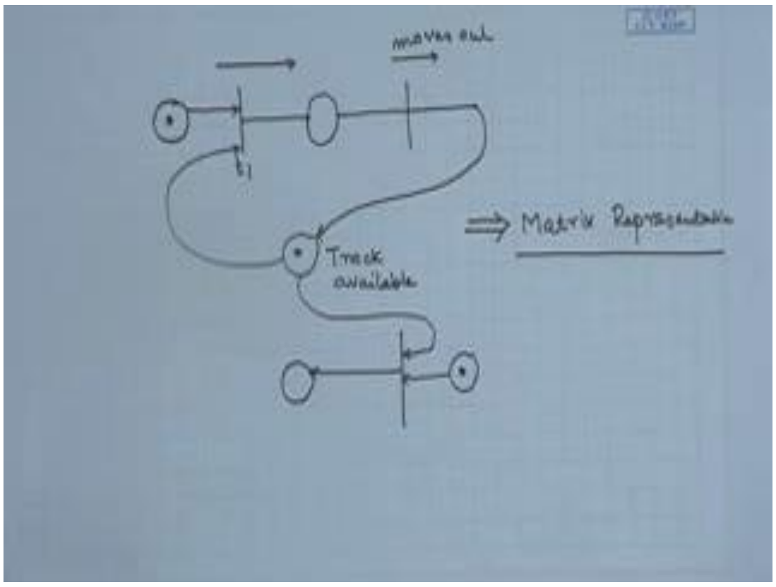
Now there are different varieties of Petri net, which says that along with these transitions I can put in some conditions, that even if these are available.

(Refer Slide Time: 30:35)



Now, say for example, there is a situation, where there is a single railway track, and trains can come, anyone train can come through this track; obviously. Either a train can come from this or a train can come from this, and you are designing a signalling system. So, that there is no accident or something like that. Now this, now you have designed something a system. Now that system or the requirement can be nicely specified using Petri net. For example let us try to do it, and see if we can. Have you got this problem, all of you have got this problem.

(Refer Slide Time: 31:29)



So, I can have one place, saying track available. Suppose this is a place which says track available, and there is a place, and there is a transition; a train has come, and this transition t 1 means the train will move to the right all right. This is the meaning of this transition. Now, I will allow the train to move to this direction, only if the track is available. So, if I modulate like this, then the train can move. If both the conditions are true, and as soon as the train moves across the signal, it comes here, then this token is consumed the track is no longer available. So, now, if at the same time, another train is coming on this side, and trying to come on this side, it is also connected here, and therefore, you can see that cannot proceed, because once one train is moving in this direction, it cannot proceed. Now when it goes out of the junction, then now this has been consumed now as it moves out of the junction; that is moves out. Then it can return the token here, and then this one can proceed.

So, in this way I can represent this scenario. Now connect this scenario to your embedded system scenario, that you can have resources. Or in operating system, I was mentioning about operating systems time and again now. In operating systems also we are using resources. So, will there be a deadlock situation. Or is it a live situation that ultimately the trains will pass. How is my design? It does my design allow that the track will be ultimately available for all the trains to pass, is it the case that all the processes will get the necessary sources, so that it can go through this, so that they can run to completion that is the aliveness property. So, all these can be analyzed very nicely using Petri net representation.

And once we do this, there are. I am not discussing that in detail in this here that this can, this has got different varieties with the number of tokens that each place can have and all those. Now it can be translated to a matrix representation, where we have got the places on their transitions, and through that matrix representation we can have a matrix representation, and using linear algebra we can do a lot of analysis on this, and that is why this comes in very handy. We can carry out a number of analysis and prove many properties of a system.

Student: Is it different from SDF?

This is asynchronous, absolutely asynchronous, SDF is synchronous and in SDF we are it is similar, but this one Petri net is asynchronous completely, and this transitions and all those there is no explicit condition of in a buffer, it is in the capacity and all this.