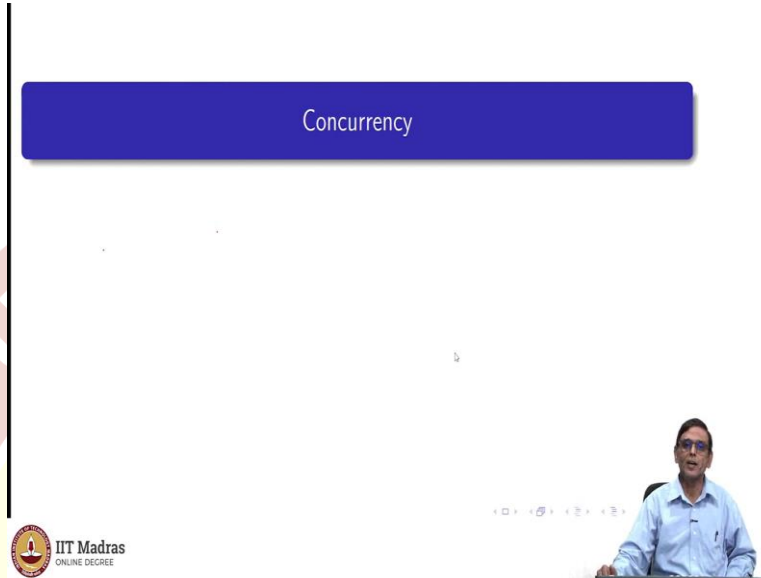


IIT Madras

ONLINE DEGREE

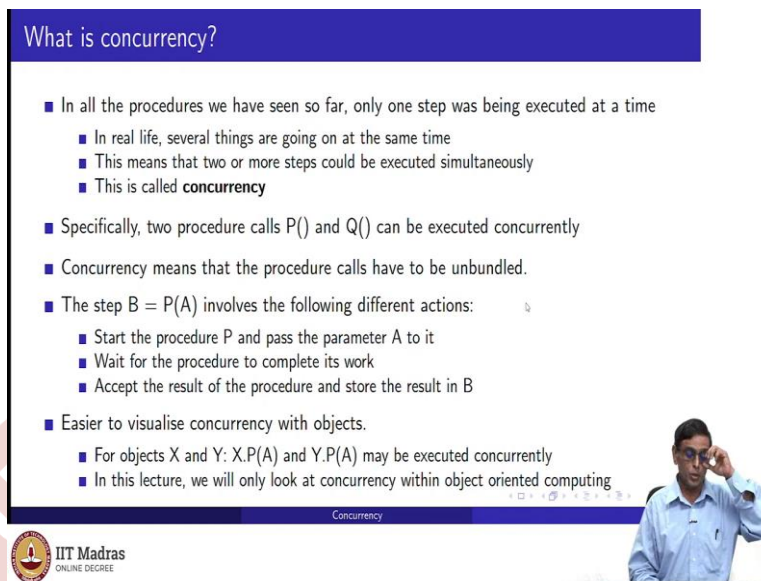
Computational Thinking
Professor G. Venkatesh
Indian Institute of Technology, Madras
Formalized notations and summary of concurrency

(Refer Slide Time: 0:14)



Welcome back to the computational thinking course. This week we saw slightly different kind of thing in the course which is, we saw how the, how we can organise the programs or the code that we are writing in the way that allows us to do multiple things at the same time. So far, we have never seen that. So, this is the first time we are trying to see whether we can do things together, simultaneously. And what kind of problems that can lead to and so on. So, specifically so we are interested in understanding this idea of this notion which is called concurrency which lets you deal with handing things multiple things at the same time.

(Refer Slide Time: 0:59)



What is concurrency?

- In all the procedures we have seen so far, only one step was being executed at a time
 - In real life, several things are going on at the same time
 - This means that two or more steps could be executed simultaneously
 - This is called **concurrency**
- Specifically, two procedure calls $P()$ and $Q()$ can be executed concurrently
- Concurrency means that the procedure calls have to be unbundled.
- The step $B = P(A)$ involves the following different actions:
 - Start the procedure P and pass the parameter A to it
 - Wait for the procedure to complete its work
 - Accept the result of the procedure and store the result in B
- Easier to visualise concurrency with objects.
 - For objects X and Y : $X.P(A)$ and $Y.P(A)$ may be executed concurrently
 - In this lecture, we will only look at concurrency within object oriented computing

Concurrency

IIT Madras
ONLINE DEGREE

So, specifically the procedures we have seen so far only once that so procedure contains a sequence of steps, the steps can be iterated, it can repeated, you can do loops and if-else and so on. But at any given point a time we ask where are you in the code, you can point exactly to one line in the code and say, this is where I am. So, at any given point in time, you are at one step and that is the only step that will be executed. Once that step is executed, it might move to another step based on how the code is structured. But any given point time, only one step is executed at a time.

But, in real life that it is not how it works because you have seen basically that when multiple people are doing things, I am here, I am doing something, you are sitting, you are doing your thing. Meanwhile, somebody else is doing something else and so. So, different people are doing things concurrently, simultaneously. So, in real life, many things are going on at the same time which means that multiple steps could be happening simultaneously. And when you have multiple steps happening simultaneously, this thing is called concurrency.

Now, we want to have a simple way of representing concurrency, putting it down in some kind of codified form and to manage it in a way that it does not create problem for us. So, what kind of issues can come up because of concurrency and how do you manage those is what we are going to discuss or what we have been discussing in this week. The lectures that you have already seen this week and this one we will actually expand on that little bit more to understand the concepts by fixing them in some kind of codified form on which you can basically then work on problems based on this.

So, specifically we can consider that there are 2 procedure calls P and Q which can be called concurrently. They can call a procedure P and then we can call another procedure Q. P and Q are executing together. So, P is at some step, Q is at another step and P proceeds to one step, Q proceeds to another step, they are going on together. So, how do you do this? How do you call procedures P and Q concurrently?

Now, to start with this is not possible currently in the way we written things because when the caller calls the procedures P, it will call P, P is basically and you passes some parameter to it. P executes, it finishes and comes back, meanwhile the caller is waiting. The fellow who called P is just waiting and P finishes its computation, it returns the value back to the caller and then only the caller continue.

So, somehow we must allow the caller to start P or call P and then not wait, do something else. So, it means that we have to unbundle this procedure call. Procedure call is doing too many things right now. So, we have to unbundle it and break it. What are the things, let us consider, so we know, it is a beautiful, beautiful thing that procedures do for us. It allows you to write things like this. $B = P(A)$. So, $B = P(A)$ basically means that we are basically calling procedure P with the argument A and then basically they, procedure P executes and then it finds the result and then returns the result and then this result is coded.

All of that, all these things because of calling procedure P, passing it a parameter A and P doing its work, then it computes its result and then it sends the result back and then the result is put in B. So, what written in one line $B = P(A)$, it is very neat cute way of writing thing. But the problem with this $B = P(A)$ is bundling too many things in it. So, we cannot do concurrency.

So, let us unbundle it. What are the 3 things that we are doing here, we are first starting the procedure P and passing the parameter A. That is at first step. We are passing procedure P, passing the parameter A into it. Then, what are we doing? Then we are saying caller is waiting for a procedure to complete its work, this is another thing. While procedure P is working that means, it has taken the parameter A and it is doing its work, the caller is waiting. So, the caller is waiting with what, wait, caller is waiting for what? Caller is waiting for procedure to complete its work.

So, we have to know whether a procedure has completed as well. has to be able to tell that it is complete as well. So, that is another thing. Nowadays, the procedure when it finishes completing its work. It has a result, it must send the result back to the caller and then the

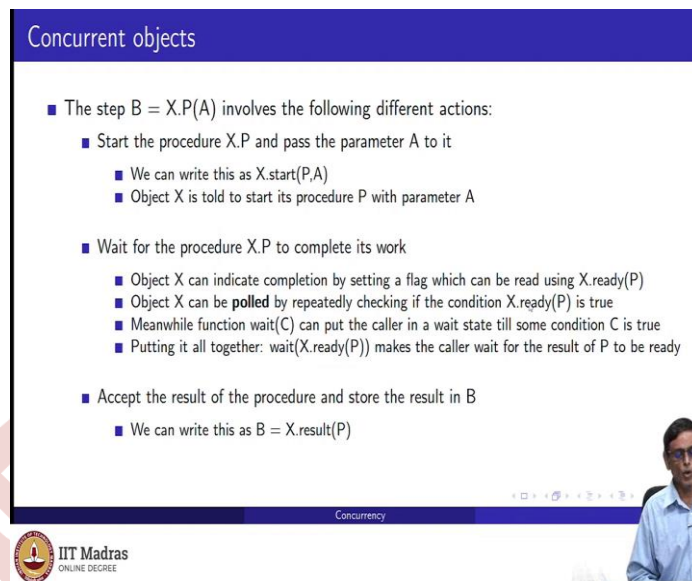
result must be store into. So, there are 3 different things going on and unfortunately when we wrote B equal to P of A, it is a cute nice way of writing, compact way of writing thing, but that compact way of writing thing is good for sequential programing.

In concurrent programing it basically comes in the way. So, we want unbundled, so we want to break it down into 3 different in some sense calls, one which will start the procedure P and pass parameter to A, another which will wait and the third which will accept the result, 3 different calls. Now, usually concurrency is nicer to visualize concurrency in the (con) context of object oriented programing. Actually, we have discussed last week object oriented programing.

So, in the context object oriented programing is very easy to understand because just like I told you I am doing something, you are doing something and so on, I and you we are all object in some sense. So, if a two objects X and Y, X can be executing its procedure P A, while Y is also executing its own procedure P A. So, Maths teacher can be computing the average, while Physics teacher is computing average, Maths teacher is computing average, then Chemistry teacher is computing average. All 4 are computing average. So, 4 can be working concurrently.

So, for objects X and Y, they can call their own procedures concurrently. We can call the procedures, the procedures maybe running on right. So, it is a nice that to think about concurrency together with objects. In fact, in this lecture we are only going to talk about concurrency in the context of object oriented programing. There is concurrency in the general context of procedure, normal procedure programming, but we are not going to discuss that. We are discussing it in the context of object oriented.

(Refer Slide Time: 6:53)



Concurrent objects

- The step $B = X.P(A)$ involves the following different actions:
 - Start the procedure $X.P$ and pass the parameter A to it
 - We can write this as $X.start(P,A)$
 - Object X is told to start its procedure P with parameter A
 - Wait for the procedure $X.P$ to complete its work
 - Object X can indicate completion by setting a flag which can be read using $X.ready(P)$
 - Object X can be **polled** by repeatedly checking if the condition $X.ready(P)$ is true
 - Meanwhile function $wait(C)$ can put the caller in a wait state till some condition C is true
 - Putting it all together: $wait(X.ready(P))$ makes the caller wait for the result of P to be ready
 - Accept the result of the procedure and store the result in B
 - We can write this as $B = X.result(P)$

IIT Madras
ONLINE DEGREE

So, we said we want to unbundle a procedure call into 3 part. So, the step B equal to $X \cdot P$, if you are writing $P \ A$ as $X \cdot P$, remember $P \ A$ is a procedure which is a public procedure of object X , so I can call it and then I can find it and it returns the result and I can store the result in B . The B equal to $X \cdot P$ is, it is a cute nice way of compact way of representing thing, you would not unbundle that into 3 part.

The first part is start, the procedure $X \cdot P$ and pass the parameter to A to it. I must have a way of writing it. So, let us say I write it as $X \cdot start$. It means that X , object X implements a generic procedure called start to which you give a procedure name and a parameter and it will start the procedure with that parameter. Then, I want to wait for the procedure $X \cdot P$ to complete its work. So, the $X \cdot P$ has to be able to tell me it has completed its work. So, how does it do that?

Let us say object X , it need not be P will say, object X if it tells the thing. So, let us say object X has within it a flag which, it sets when P is finished, when P is finished, it will set the flag and object X basically can have a procedure called ready which you can call, anybody can call ready and what ready will do basically is, it will check all the flag corresponding to P as we said. We give it P as a procedure, I will check whether P is flag, that means P completion flag whether it has been set or not.

If it has been set, then $X \cdot ready \ P$ will come, return to, if it is not been set, then $X \cdot ready \ P$ will be returning call. Obviously, when you start $X \cdot P$, the first thing that when you start $X \cdot P$, that means when you call $X \cdot start \ P \ A$, the first thing the start thing has to do basically is to set this completion flag to false. Whereas, basically if somebody ask, are

you ready? It should say false. So, you can check whether or not the object is ready by calling X dot ready, but when do you check?

So, maybe the caller can keep checking, so object X can poll, it is called polling. Polling means you get repeatedly keep calling and checking, neat checking. So, it can check repeatedly check, put it in the while loop and check whether or not P is ready. So, whether X dot ready P is true or not you can keep on checking by using polling. So, we put it in a while loop and we need to wait. I can put the while loop first, what does the while loop doing other than checking? It has to wait for some time.

So, a much better way is to write a nice function called wait and what wait will do basically is, and wait for condition, wait field and what wait does is it put the caller in a wait state and will wake up the caller once he becomes true. Now of course, how this is implemented is the very interesting concept in the operating system, when you get into operating system, must ignore in, you might want to think about how this is implement, things like this are implemented.

But usually systems have a very (of) waking up things from sleep and so on and we also would use one of those method. So, let us say that the caller has put to sleep and you waken up when the conditions becomes true. So, wait here basically makes the caller to wait till the condition P becomes true. So, we can put these 3 things together. We say this object X has implemented a procedure called ready for each procedure which will tell you whether or not that procedure is finished or not. Then the caller basically will keep checking whether every P is true and it does that by actually using this wait condition, wait for.

Not only wait for, what we see in this case, where what should it waited for, usually it should wait for X dot ready P. So, the simplest thing for the caller to do basically is to wait simply for X dot ready P. So, as when does X dot ready P become true? X dot ready P become true when procedure P finishes and then procedure P set the flag to true and that basically makes that X dot ready P call come out of true. So, wait X dot ready P is very basically become true at that point of time.

So, we saw two things, one is how to start a procedure, you start a procedure by calling X dot start, give it a procedure and a parameter or bunch of parameters. The second is to check whether or not the procedure is completed, if not you do that by waiting for X dot ready P. And now we need the third thing which is basically assuming that the procedure is ready, it

has completed its work, I must pick up the result. So, you have to call a way of picking up the result.

So, we can write a third function that X implements which is called result. If you remember we have written already two functions that X implements, one is called start, another is called ready. Now this is the third function result. So, X implements the function called result, you can call X dot result and it, pass it the procedure name as the parameter and it will return back the result of it.

(Refer Slide Time: 11:47)

Example: Classroom dataset

- Consider two objects MaT and PhT of the ClassAve datatype
 - We wish to find MaT.average() and PhT.average() concurrently
- Recall that ClassAve had the following:
 - private fields marksList and aValue
 - public procedures average() and addStudent(newMark)
- To make average concurrent:
 - We need to implement the procedures ready(average) and result(average)
 - But we already have the field aValue which does this !
 - So ready(average) can just return the boolean value (aValue ≥ 0)
 - Similarly, result(average) can just return aValue

Concurrency

IIT Madras
ONLINE DEGREE

So, the code for concurrency therefore, consists of unbundling the procedure call into these 3 procedures. Let us look at it with the with an example. easier to understand with the example. So, let us go back to our classroom dataset. So, we saw basically the class average is the data type that we define which found the average of different things. The class teacher for example could find the total average, maths teacher could find the maths average, physics teacher could find the physics average and so on.

So, let us say we want to find the maths and physics average n parallel. How do you do that? We create a maths teacher object, we create a physics teacher average and we want to call maths teacher average and physics teacher average is parallel, we want to call them concurrent. So, we call that class average has the following fields. It has maths list and A value as the field and here we type public procedures average and add Student mark, add Student, two public procedures.

So, to make average concurrent, the class average procedure has to implement start, ready and result. It has to implement start, ready and result. I will write here, we should implement the procedure start average, ready average and result average. Now, for start average, we have to implement it, I mean there is nothing we can do. So, assume basically it is the procedure that basically starts the procedure, basically starts P A will just call P of A that is what it should do.

So, it is relatively straight forward, but we have to implement it. Now, assume that as what about ready average? Now we have already a field aValue with fields to do it because remember, how did we write the codes for average, the average code for class average set basically that check whether aValue is minus 1, if it is minus 1, compute the average and put the average result in aValue. So, if aValue is not minus 1, it basically means that average is computed, finish computing, it got its result, aValue as the result. So, it is nothing to do.

So, aValue is already in some sense the flag for checking whether the procedure is completed its work or not. If aValue is minus 1, it means it has not completed. If aValue is positive, it means it has completed, it finished its work. So, aValue is already the flag that we need. So, all we need basically for ready is to return aValue greater than equal to 0, the Boolean condition aValue greater than equal to 0, you can return that.


So, if aValue is greater than equal to 0, one ready will return true, if aValue is not greater, it could return aValue is minus 1, then ready will return false which is all we need. What about result? Now, the result of computational average is aValue. So, result just needs to return aValue. So, previous, so start of P A will just call P A. What does ready average do? It checks aValue is greater than equal to 0 and returns it. What does result average do? It just returns aValue. Very simple procedure, all these procedures are extremely simple. And by using these 3 procedures we are now doing in some sense our average procedure concurrently.

(Refer Slide Time: 15:00)


Example: Classroom dataset

- So to execute `MaT.average()` and `PhT.average()` concurrently:

```
MaT.start(average)
PhT.start(average)
wait(MaT.ready(average) and PhT.ready(average))
AveMaths = MaT.result(average)
AvePhysics = PhT.result(average)
```
- Note that:
 - `start` is called with only `average`, as it has no parameters
 - caller waits for both `MaT` and `PhT` to be ready



IIT Madras
ONLINE DEGREE



Concurrency

So, suppose we want to do maths and physics averages in parallel, this is the code for it, what we will do? We will call maths teacher's average you want to start, now they are not passing any parameter here because does not take a parameter. So, we say `MaT dot start average` it means the starting the average for maths, then physics teacher dot start average. So, we started the average physics. Now we are all waiting for average to compute.

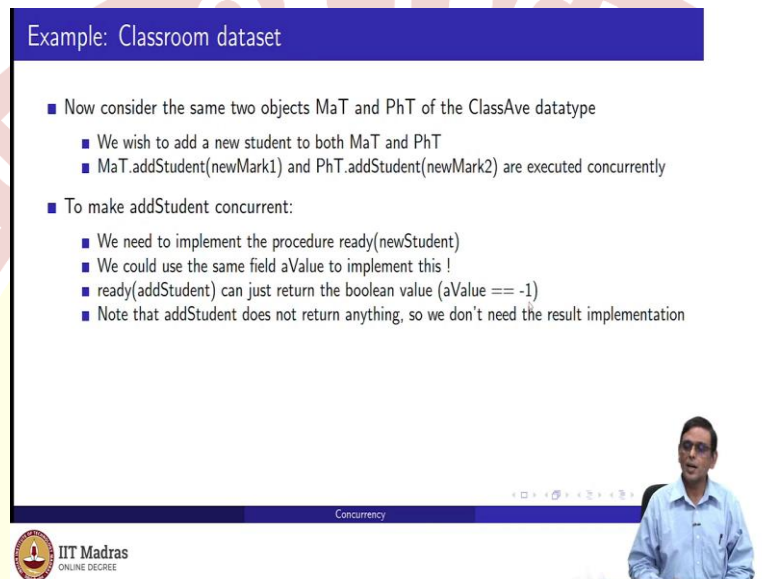
We started average for maths, and then proceeded to start physics average and then proceeded. Now both average maths and physics are running now. We do not know what is going on, they are running. Now, what do we wait for, you have to wait for both of them to finish. How do you do that? Remember that `ready` is the Boolean so we can do `add`, do maths waiting it ready, maths teacher dot ready average and physics teacher dot ready. Add it for both of them to become true.

When both are true at that point in time the wait condition is true so it comes out. So, when it comes out basically at this point both of them are ready, now I need to read the result. How do I read the result? I call `math MaT dot result average` so that give me the maths average and I store it in average maths and I call physics teacher dot result average, I get the result for physics and store it in average.

So, I have, as you can see here, I have unbundle it because of unbundle is a call, because unbundle is a call therefore I can do this start, I can do the starts and then proceed without waiting, I can start physics, I can start maths, and I can proceed and then I can wait whenever I want I can wait for both of them to become ready, then I can read the result and store it. So, this unbundling allows me to do things concurrently.

Of course, this code is much looking much more complicated and involved by just writing average maths equal to MaT dot average. So, this is a cute code. And average physics equal to PhT dot average but that means basically it cannot be concurrent. You want to do concurrency, it is the cost in terms you have to do something a cost in terms of coding that you have to do in order to. So, these are the value. Now, note that the caller is waiting for both maths and physics which has to become ready through the Boolean AND which is true.

(Refer Slide Time: 17:13)



Example: Classroom dataset

- Now consider the same two objects MaT and PhT of the ClassAve datatype
 - We wish to add a new student to both MaT and PhT
 - MaT.addStudent(newMark1) and PhT.addStudent(newMark2) are executed concurrently
- To make addStudent concurrent:
 - We need to implement the procedure ready(newStudent)
 - We could use the same field aValue to implement this !
 - ready(addStudent) can just return the boolean value (aValue == -1)
 - Note that addStudent does not return anything, so we don't need the result implementation

IIT Madras
ONLINE DEGREE

Now, consider that we have two objects maths teacher and physics teacher and, we want now to do not only average, we do want to do add student also concurrently where maths teacher and physics teacher object and they have add student procedures, why only average, let us also make the add student also more concurrent. How did we do that? For that again same way, we need to implement ready newStudent, not newStudent, it is addStudent. It should say ready addStudent and how do you implement ready addStudent?

Ready addStudent basically, then again you see aValue because we know that the result when addStudent finishes, aValue is set to minus 1, so how do we know actual it has finished its work? You know as soon as finished its job, when aValue is minus 1. So, just look at aValue and you know whether addStudent is in aValue, then also act as a flag to see whether or not addStudent has finished.

So, ready addStudent it just return the Boolean value, aValue equal to minus 1. And what is the result? We do not need a result because addStudent is a procedure which has no result, it is just adding a student to the marks list and setting aValue to minus 1 otherwise, it does not

return anything. So, addStudent does not return anything so we do not need to implement the result. Very simple. So, we have now made both average as well as addStudent concurrently.

(Refer Slide Time: 18:43)

Example: Classroom dataset


- So to execute `MaT.addStudent(newMark1)` and `PhT.addStudent(newMark2)` concurrently:

```
MaT.start(addStudent, newMark1)
PhT.start(addStudent, newMark2)
wait(MaT.ready(addStudent) and PhT.ready(addStudent))
```

- Note that:

 - start is called with parameters
 - caller waits for both MaT and PhT to be ready

IIT Madras
ONLINE DEGREE



Now, I want to take you to some contiguous I am going to see how things can get exhibit concurrently. So, let us say for example now I want to do, I will show you how to, we will talk about the code. So, to basically do the code concurrently, addStudent concurrently for physics and maths, we do `MaT dot start addStudent newMark 1` with this teacher dot start `addStudent newMark 2` and then you wait for both maths and physics students to be ready, that is what you do.

(Refer Slide Time: 19:22)

Example: Classroom dataset

- Now consider the situation where while `MaT.average()` is executing, a new student is added to MaT:

```
MaT.start(average)
MaT.start(addStudent, newMark)
wait(MaT.ready(average) and MaT.ready(addStudent))
AveMaths = MaT.result(average)
```


- There is potential for conflict here !

 - average will set `aValue` to 0 or a positive value
 - addStudent will set `aValue` to -1
 - So both cannot be ready at the same time !
 - `wait(MaT.ready(average) and MaT.ready(addStudent))` will just wait forever !!

- To prevent this, we can use explicit `aveReady` and `addReady` fields:

 - `addReady` is set to false when `addStudent` starts and to true when it finishes
 - `ready(addStudent)` returns the value of `addReady`
 - Similarly for `aveReady` which is set after average completes

IIT Madras
ONLINE DEGREE



Now, I want to do basically something much more interesting, what do I want to do? While average is executing, I want to add a new student, while average is executing, I want to add a new student, what does this do? So, I start average and I start addStudent in parallel I want to do both of them because average is concurrent, addStudent is concurrent, so in principle I can also call addStudent concurrently. That is what I want to type. And I want to wait for both average and addStudent to complete and then I want to take the average of average resultant put it in average maths.

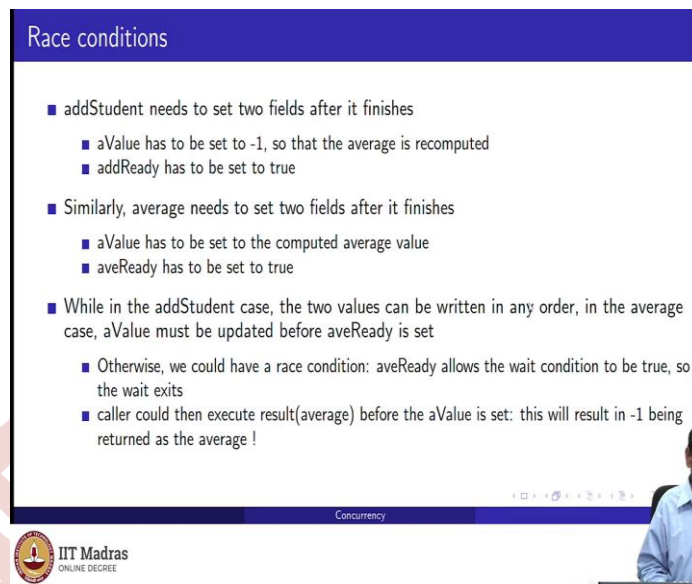
So, I am calling the same object. If it was different objects maybe things would work out okay, but the same object maths I want to call basically for maths, physics I want to call it average and addStudent student at the same time. We already see that this is going to create problem, but let us see what kind of problem it create. First problem we see is that there is a conflict in aValue itself. Remember that average will set aValue to 0 or a positive value at end, addStudent will set aValue to minus 1 at the end. So, either aValue is 0 or a positive value or aValue is minus 1. It cannot be both.

So, remember that for ready for ready average we said just check whether aValue is non minus 1 and for ready addStudent you just check whether aValue is minus 1. Now, both of them are not possible. So, it is not possible for aValue to be both minus 1 and not minus 1. So, which means that they wait, no in those case if you look at that wait, you can see that ready average and MaT dot ready addStudent look very innocent. It look as if.

The correct thing, you are saying wait for average to become ready, and wait for addStudent to become ready. Correct thing to say, but you see a way implemented it. It never possible for both average as well as addStudent. So, the wait will wait forever. It is never going to win. So, it is actually a purposable, it is wrong code. We have made a mistake in the code. What is the mistake? Mistake was trying to use too much two way too much work with this value. It is the same way as we are using as a flag using it to show us the result, we are doing many things with it or using it. so, let us say I do not want to be sogreedy as any number of fields where my.

Let us add some more fields. So, let us say we explicitly add two field, one is called average ready, another is average. Average ready will tell you that there is an average, the procedure average is completed its work or not and addReady will tell you whether or not the procedure addStudent has complete or not. So, when addStudent finishes, it will set addReady to true, similarly when average finishes it will set average to true. So, that should take care of it.

(Refer Slide Time: 21:58)



Race conditions

- addStudent needs to set two fields after it finishes
 - aValue has to be set to -1, so that the average is recomputed
 - addReady has to be set to true
- Similarly, average needs to set two fields after it finishes
 - aValue has to be set to the computed average value
 - aveReady has to be set to true
- While in the addStudent case, the two values can be written in any order, in the average case, aValue must be updated before aveReady is set
 - Otherwise, we could have a race condition: aveReady allows the wait condition to be true, so the wait exits
 - caller could then execute result(average) before the aValue is set: this will result in -1 being returned as the average !

IIT Madras
ONLINE DEGREE

Now, addStudent has two things to do, one is it has to set aValue to minus 1 and then it has to set addReady to true. Strictly speaking, it can do this in any order, it really does not seem to matter. Similarly, average also can set, let us see. The average also should set two fields to operate finishes. One is aValue and another is average ready, both of them should be 0. So, in the case of addStudent, it looks like we can set the aValue and addReady in any order, we could set aValue first and then set addReady or addReady and then aValue, nothing we go wrong. It looks like that. We will have to see.

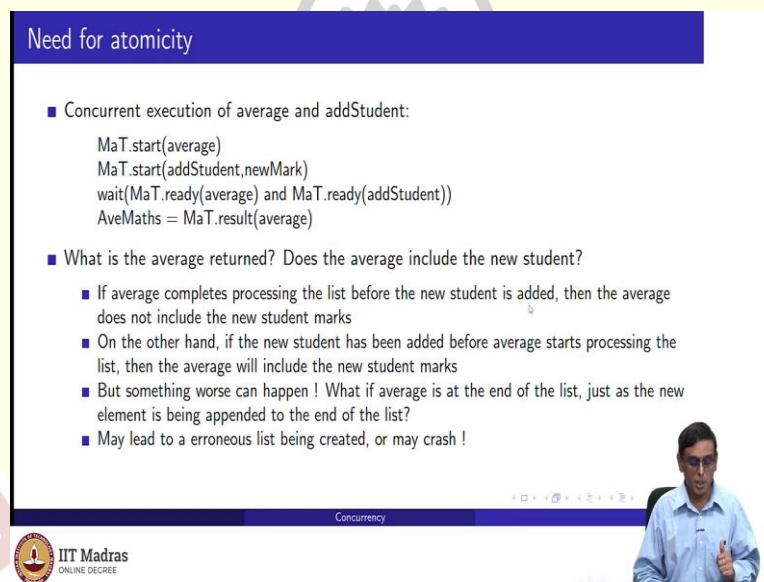
But definitely there is a problem with average because assume for a moment that when you are writing the procedure for average, at the end of average you set aveReady, average ready to true and then after that you are going to set aValue to minus 1. Now you are going to set aValue to be average value. So, it is not minus 1 value. So, but the minute you set average ready to true, it may turn out that just as you did that, the caller basically check for the condition ready and he finds that basically average is ready.

And because he found average is ready, why, how does he find out average is ready? He found it average is ready because the ready return true and when did ready return true? Ready return true because average ready is set to true, and which you already set, so it return true. So, the caller basically it found that the procedure is ready so it immediately calls for the result. So, it calls for the result and we just saw happen, like of course, if the answers are very low because there are lot of things happening from outside, this is happening inside, so it will setting the average ready to true and calling aValue occurs at only a few microsecond may take place.

So, it may happen, actually nothing may happen, but this is a odd change, this is a possibility that there may be some delay between getting aveReady and aValue to minus 1. In that delay it may happen basically that ready is rend by the caller, caller thinks that average is ready so calls for the result. And what will happen? This guy will resend, return the result. What is the result? Result in aValue. It will return minus 1.

So, possible that there is a race condition, very small chance, but there is a possibility that the race condition, you ask for the result and actually get nonsense result as minus 1. So, therefore, you must set aValue first and then set addReady. There are like this when you do concurrency, you will find basically that you must respect sequence in which things are done much more carefully than you would do it normal procedure programming because for outside any order things may come to you.

(Refer Slide Time: 24:47)



Need for atomicity

- Concurrent execution of average and addStudent:

```
MaT.start(average)
MaT.start(addStudent,newMark)
wait(MaT.ready(average) and MaT.ready(addStudent))
AveMaths = MaT.result(average)
```
- What is the average returned? Does the average include the new student?
 - If average completes processing the list before the new student is added, then the average does not include the new student marks
 - On the other hand, if the new student has been added before average starts processing the list, then the average will include the new student marks
 - But something worse can happen ! What if average is at the end of the list, just as the new element is being appended to the end of the list?
 - May lead to a erroneous list being created, or may crash !

IIT Madras
ONLINE DEGREE

Now, this quietly more interesting or more complicated problem will comes because of concurrency. In this case what happens basically is that you call average and you call addStudent let us say in parallel concurrent execution, we already saw basically that if you try to do it with aValue, that ready average and ready addStudent will basically look for all. So, we took care of that. What did we do? We kept a separate flag called average ready, and addStudent ready, addReady and will search for that and so at least now you know that wait is not looking forever, it will cover all the thing.

But still we can ask is when we read the result of average what is this result? What result is coming out? This result including the new student we have just been added or is this result before the new student has added, we do not know because we call both of them together. We

called average and new student together. So, did the average it computed before the newStudent were added or did it get computed after the new student was added?

Now, if it so turned out that the average had computed finished processing the entire list and after that when new student was added, then it will return the old value of average, the average without considering this student. Or if it turned out that addStudent was faster and it finished appending the student to the enter the list, and then average started computing. Then it will find basically that the average will include the new student. So, average could either be including the new student , sometimes or sometimes may be not including the new student. So, it , but, something worst can happen.

What happens if average is processing the list, it has come to the last element in that list and it going to that last element is indeed the last element, it may just connect to the end of the last element of the list. At this point, addStudent starts inserting the element into the list. So, while one guy is trying to read the end of the list, another guy is trying to change the end of the list, what will happen here we do not know. I mean, something crazy can happen.


It is possible that the list is completely corrupted as a result of this operation, many times system may crash and all that, erroneous list say, a list can be corrupted and subsequent calls will all crash and all this can happen. Therefore, we must be very careful about calling two things concurrently when they access the same to make a list and so on.

(Refer Slide Time: 27:11)


Need for atomicity

- Concurrent execution of average and addStudent:

```
MaT.start(average)
MaT.start(addStudent,newMark)
wait(MaT.ready(average) and MaT.ready(addStudent))
AveMaths = MaT.result(average)
```
- To prevent corruption of the list, we have to make sure that we do only one list operation at a time. The list operations cannot be concurrent. We say that the list field is **atomic**.
 - If we are doing append on the list, then any first, rest, ... operation will have to wait
 - Reverse also holds: append will have to wait for any other list operation to complete



IIT Madras
ONLINE DEGREE



Concurrency

If you want to prevent 2 people from accessing the list at the same time, how will you do that? you do that by say making the list operations what are called atomic. You say the list field is atomic, atomic means that only one person can enter it at one time, two people cannot access the list operation at the same time. If one list operation is running like for example, append is going on, all the other list operations are made to wait.

So, if you are trying to do a first with the list, then every other operation that is waiting to do things with the list may crash, append and so on, they will all have to wait. So, only one at a time, single thing at a time that is why it is called atomic. So, if you make the list atomic, the list field atomic, it will operation for the list that are atomic, then you can prevent this kind of race conditions and you view these things.


(Refer Slide Time: 27:55)

Need for atomicity


- We could argue: why do we need to execute average and addStudent concurrently. Just do them in sequence. For instance:

```
MaT.start(addStudent,newMark)
wait(MaT.ready(addStudent))
MaT.start(average)
wait(MaT.ready(average))
AveMaths = MaT.result(average)
```
- Will clearly return the average including the new student.
- The issue is while designing the concurrent object of datatype ClassAve, we cannot control who will call the procedures and in what order
 - One caller X may call addStudent, and a different caller Y may call average concurrently
 - It is very difficult to have X and Y co-ordinate on their use of the shared object MaT

Concurency



IIT Madras
ONLINE DEGREE



So, one can argue now I mean, if point is that this looks very artificial one can say, nobody is going to call addStudent and average at the same time. Why would the caller call addStudent and average at the same time? You know this do not add a student, why does it just adds a student and then call average like what I have written here. Call addStudent, wait for ready, then call average, wait for ready, then find there is a average, even if you want to do it concurrently, do it like that. So, do it sequentially in a sense, do not try to do it concurrently and you going to do both of them, mat will do it correctly in right order.

The answer is it may not be the case that the same guy is calling both the procedure, you might have one object X which is calling addStudent and different caller while might be calling average concurrently. So, two people may be calling the same object, two procedures at the same time and it is not possible for X to talk to Y and find out, are you going to call, it is asking why, are you going to call, if so then I will wait, you finish your call and then I will call, how do you coordinate like that? It is very difficult for X and Y to coordinate.


So, because this very hard for X and Y to coordinate, therefore and we do not know, we cannot prevent in fact because both of we are public procedures, we cannot prevent anybody from calling and at any time, therefore, it is necessary for us to plan for concurrency. We have to plan that if there are two procedures, it is possibility that both of them are called at the same time and therefore we cannot access the list and the list can get corrupted. So, we have to think when we are doing concurrency, we have to think about all these possible cases that can come up and protect all these kind of cases by making things atomic.

(Refer Slide Time: 29:34)


The Producer-Consumer model

- So far, the caller started the procedures and then just waited for them to finish (polling).
 - What if the caller also does something concurrently with the procedures ?
 - Then the called procedure will have to wait till the caller is ready before it can return the result. This can become quite complicated
- There are two ways this can be remedied:
 - The procedure when it finishes execution can **pre-empt** the caller and return its value. We will not discuss this method further here.
 - The procedure when it finishes places the result in a pre-agreed place (called a **buffer**). This gives rise to a model called the **producer-consumer** model of concurrency,
- In the producer-consumer model, the caller is the consumer and the object whose procedure is called is the producer.
 - The consumer may also queue up tasks (procedure calls) for the producer to execute one after the other.
 - To ensure that the producer knows where to write the results for each of these tasks, the consumer can pass the result buffer as an argument in the procedure call.

Concurrency



IIT Madras
ONLINE DEGREE



Now, I want to finally come to another model which is called the Producer-Consumer model. So far what we saw basically is that the caller started the procedures and then just waited for them to finish, to which what we call as polling. Now, what if the caller does not want to wait, wants to do also something? So, wants to do concurrently something. So, it may turn out basically that the caller calls the procedure, here maths teacher and ask it find average, the caller is also doing some work.

So, now the maths teacher when it finishes it publishes ready and it is got it result and so on but this result it has to hold indefinitely because it does not know when this guy is going to call, it may in the earlier case here the polling I know that many type of put ready I know that that guy is waiting or need to become ready so as soon as it becomes ready you will take my result. But now I do not know I do not know when he is going to by this holding this result so I just prevent other people from calling me till I have asked from the result what should I do?

So, this is quite complicated this complicated. So, there are two ways to deal with it. One is the procedure can create the column here I do not care you are the one who started me so I have finished my work, whether you are doing something else or it is not my business I stop you and I will give you the result, if you are in the classroom teaching I will interrupt you in the class and give you the result, something like that. So, this is called pre-emption and in order to implement pre-emption we need a number of things like interruption.

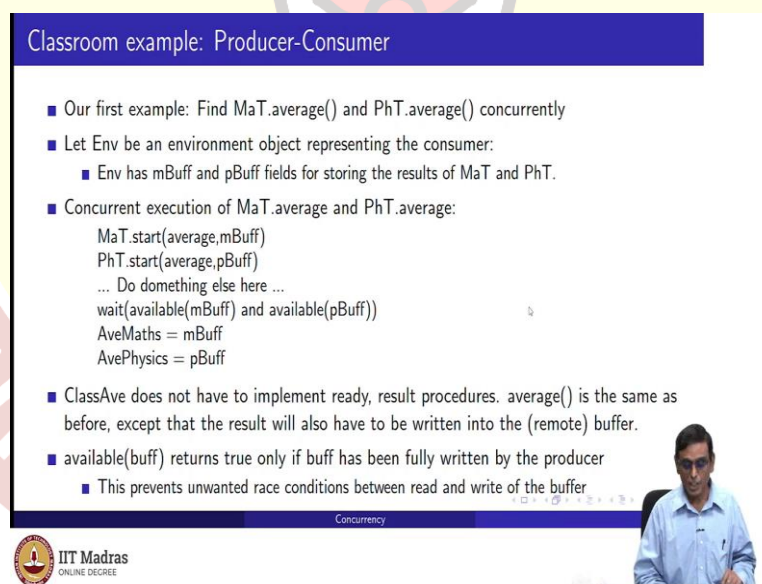
We are not going to discuss this method for that here. We are going to discuss another method which is the much easier method which is called the producer-consumer model. This is what happen with me, is that the caller says look I want to do something while I am calling

you so therefore, what I will do is, I call you now I really do not want to know whether you are finished or not you just when you finish, whenever you finish it. You put your result in a buffer. So, how do I so I pass the buffer to you, I give a buffer and tell you or say whatever it is we called in the message.

And I tell you basically that you put your result in this buffer and I will know that you are finish because I can see that the value has come in this buffer. This is the way it is done. So, in the producer-consumer model the caller is the consumer and the object whose procedure is being called is the producer because that is the guy who is doing the work for me. Worker is the producer as a sub-contractor is a producer and the contractor they were asking for the work to be done is the consumer.

The consumer can also of queue up task friends the producer but we are not looking at this particular kind of area. So, to ensure that the producer knows where to write the results I pass the buffer which I am basically wanting the consumer to write into producer to write into I pass that buffer as an argument in the producer call.

(Refer Slide Time: 32:16)



Classroom example: Producer-Consumer

- Our first example: Find `MaT.average()` and `PhT.average()` concurrently
- Let `Env` be an environment object representing the consumer:
 - `Env` has `mBuff` and `pBuff` fields for storing the results of `MaT` and `PhT`.
- Concurrent execution of `MaT.average` and `PhT.average`:

```
MaT.start(average,mBuff)
PhT.start(average,pBuff)
... Do something else here ...
wait(available(mBuff) and available(pBuff))
AveMaths = mBuff
AvePhysics = pBuff
```
- `ClassAve` does not have to implement `ready`, `result` procedures. `average()` is the same as before, except that the result will also have to be written into the (remote) buffer.
- `available(buff)` returns true only if `buff` has been fully written by the producer
 - This prevents unwanted race conditions between read and write of the buffer

Concurrency

IIT Madras
ONLINE DEGREE

So, let us take our old example back again. Now first example was finding, doing physics and chemistry, physics and maths are this together. So, we saw the polling model for it we started maths, we started the maths average, we started the physics average we waited for both of them to be ready and then we take other example that is what game. That was the polling model. What do we do in the producer consumer model?

In the producer consumer model we need buffers and where are the buffers, the buffers are in the environment. So, we do not want to leave this buffer hanging around somewhere so it is nice to think about the producer as a consumer as a object, the guy who is going to consume the result of this average, we like to think about him also as an object. So, let us create an object which is called environment, environment Env, let us say environment as 2 fields called mBuff and pBuff that is supposed to hold the results of the average computation of maths, physics and chemistry.

So, the concurrent execution of maths teacher and physics teacher average then goes like that. Maths teacher starts average, now you remember, average does not normally have a parameter but because we have fed in producer-consumer model I will pass the buffer to them as argument so, therefore I pass mBuff as an argument to this function. Physics teacher has start with an average and pBuff as the argument so I am telling you basically physics teacher, you compute the average and you finish computing the average you put the result in pBuff I am telling the maths teacher when you look finish your computational average you put your result in mBuff.

So, the two different buffers so both there is no danger that physics teacher write into the maths location and maths teacher writes into physics, there are two different boxes. Once I have started it I can do something else so the caller can go to something else after he has finished doing something else whenever he has finished doing something else, we can check whether mBuff and pBuff are available.

Available is a function now what we have to write who writes available who is implementing available? Environment has to implement available where environment will check whether mBuff is fully return into, pBuff is fully written into and if both of them are fully written into and then the and condition, then I get out of that way so I am waiting till so very lightly that if I spend a lot of time doing something else by then both of them finish and put result in mBuff so this is available will it and say it is done and I can wait will not take any time. It just come out.

It is also possible that they are doing something else quite take very little time I finished it quickly and I come here and I find meanwhile that guy has not finished, although it is just still computing in which case it is very much like the polling case. So, now I am waiting for that guy to finish it is fine there is no problem. But it is also possible that I have lot of work to do so I did all that work and came meanwhile and those guys has finished they tool the result

in tray and they went on and did something else, they can attend to other customer they can do average for another customer.

So, once I have the complete pack, one for both mBuff and pBuff ready, I can just take the value of mBuff and put it in average maths, pBuff and put it in average physics and done. I do not need to go and fetch for results to now anymore from the object because the object have already gave me the result, we put it in the mBuff. So, we see basically that the owners now is on environment, owners is not so much on class average. Class average does not have to implement this ready and result procedures because we are not checking for ready, there is no need for ready to be implemented.

There is no result procedure because it is just reading mBuff, so no result required. So, average just the same as before except that at the end of average, after computing average, instead of setting from flag to true so that ready can basically we call to find out whether or not it finished, instead of doing that basically what average is doing now is that it is writing its result in a buffer. Now remember this buffer is in the environment, so it is writing, average is writing the result in a remote buff in a buffer that is far away, it will write it there.

So, that work it has to do, that is the extra work it has. that it is a much easier job for class average. And what does available do? Available basically returns true only if the buffer whose it argument is basically fully written into that producer. Remember when we have this discussion, we were talking about what happens, this idli is being served with chutney and before the chutney you have put on the tray, I took that idli and all that, you can issues.

How do I know that what are written as being written fully, before I wrote the full data, somebody could start reading the data and that nonsense. So, the available function basically make sure that buffer has been fully return to by the producer and only after that it says it is available. So, this basically a ensure that the read and write operation in the buffer are atomic. Remember what the term is, it is atomic. So, therefore it basically make sure that you cannot do read and write in the same time, that is done by available.


(Refer Slide Time: 37:03)

Classroom example: Producer-Consumer


- The next example: `MaT.average()` and `MaT.addStudent(newMark)` concurrently
- Let `Env` be an environment object representing the consumer:
 - `Env` has `aveBuff` field for average, and `addBuff` field which is just a boolean flag
- Concurrent execution of `MaT.average` and `MaT.addStudent`:

```
MaT.start(average,aveBuff)
MaT.start(addStudent,newMark,addBuff)
... Do something else here ...
wait(available(aveBuff) and available(addBuff))
AveMaths = aveBuff
```
- `addStudent` will have to set `addBuff` to true after it finishes
- There is no need for the two flags `aveReady` and `addReady` anymore. The write to `aveBuff` and `addBuff` serve as their equivalents
- The race conditions with the list can still occur within `MaT`, so the need for the list field to be atomic continues

Concurrency



IIT Madras
ONLINE DEGREE



So, now if I want to do average and add student concurrently, it is the same. I have a buffer for average and I have another buffer filed for, well we can have for addStudent. After all, addStudent never returns the result. addStudent is not returning any result, so why do I need any buffer? Average was returning result so I need a buffer, but average is the, the problem is that now we are not, we do not have the ready function, so we do not have a way of checking the work has finished or not.

So, we need to implement a buffer for addStudent but it can just be a Boolean flag and when addStudent finishes its work, what it does is basically it sets the remote Boolean flag, that is all it does. So, addBuff is just a Boolean flag, there is no result, but it makes this flag simply to know it is finished. So, the concurrent execution says is basically start a average passing a carriage buffer and for addStudent you have to pass it its parameter which is newMark and also pass this addBuffer, then you can do whatever you want. and after you have finished, you check whether both average buffer and add buffer are available, means both have been return into and then you read the average mark into average buffer.


But the problem is of atomicity of that list what we saw that both, the addStudent and average work on the same list. So, one can be reading, another can be writing into the list and we need to take care of that, that problem still remains. So, we have to basically that the atomicity condition of the list is being represent. Other than that we have saved the border of writing ready functions and so on, so in some sense this producer-consumer model is a little bit more incurred and nicer and we have to understand than the polling model. But one can use either, one can use this or that.

(Refer Slide Time: 38:46)


Summary

- Concurrency allows several steps to be executed simultaneously
 - To keep control of this, we only considered concurrent execution of procedures within objects
- In the polling model, the caller starts all the procedures to be executed concurrently, and then waits for them to complete
 - The object can be polled to check if it has finished. The result can then be retrieved from the object (we used `ready` and `result` for this).
- In the producer-consumer model, the caller object starts all the procedures to be executed concurrently, and passes result buffers to them to write into when they finish
- In both cases:
 - caller used `wait(C)` to wait for the boolean condition `C` to become true
 - In our polling model, `C` used `ready()` to check if the object has finished its task
 - In our producer-consumer model, `C` called `available(B)` to check if `B` has been fully written
- In both cases, race conditions have to be handled by ensuring that access to shared objects (such as lists) are made atomic (i.e. non-concurrent).

Concurrency



IIT Madras
ONLINE DEGREE



So, in summary, concurrency basically allows us to do several things simultaneously, we keep control of it by in some sense encapsulating the concurrency within objects and then in the object, the objects implement basically procedures to unbundle the procedure call, they basically implement the start procedure which starts the procedure; ready procedure which tells you whether or not the if the finished, procedure has finished and the result procedure will return the result. That is the polling mode.

In the producer-consumer mode, on other hand, the consumer will pass, create a buffer in consumer object will create a buffer and pass this buffer to the producer, the producer when it finishes writes the value into its buffer, consumer can check whether the buffer is available and if the buffer is available, then it can. So, in both cases, we are calling condition wait for a condition to check to wait for something some. In both cases there is wait `C` that is going on.

In polling model we use `ready` to check whereas, in the producer-consumer we will use `available` to check. So, that is kind of similar so, and in both cases basically we had this issue of atomacy. We have to make sure that list operations two things cannot be going on at the same time. One guy is writing the list, another is reading the list, it does not happen simultaneously.

So, hope you enjoy this lecture on concurrency. We will see you all next week where we are going to look at yet another different type of programming paradigm which is the bottom of computing, which we call bottom of computing. So, we will see you next week.