

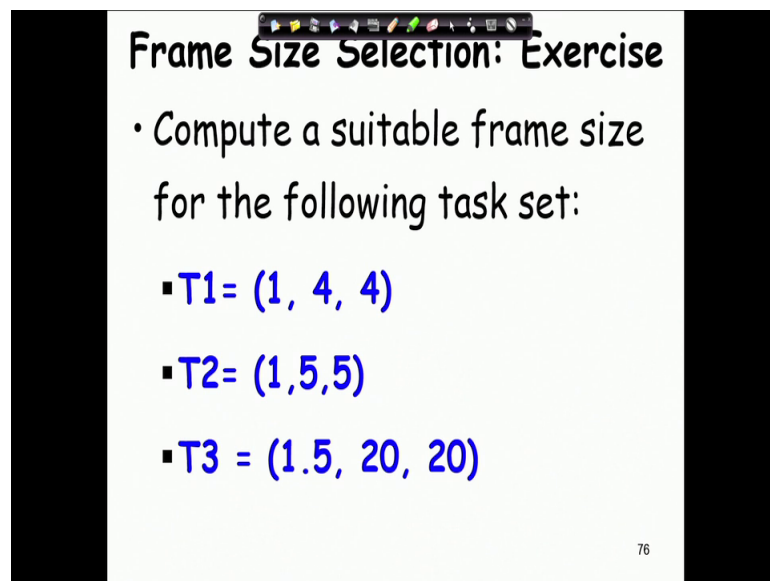
**Real Time Operating System**  
**Prof. Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 06**  
**Exercises on Frame Size Selection**

Welcome to this lecture. So, far we have been looking at one of the very important, but simple real time task scheduler known as the cyclic scheduler. We had seen that the design parameters that need to be decided by the programmer is one is the major cycle which is easy to set which is LCM of the task periods of the periodic tasks and the minor cycle also called as the frame. And we had seen that the frame needs to satisfy some constraints and we had done some examples. I hope you have understood this examples.

Now, let us do one exercise please work out on your pen and paper let us look at that exercise. So, here in this problem we have 3 periodic tasks.

(Refer Slide Time: 01:07)



**Frame Size Selection: Exercise**

- Compute a suitable frame size for the following task set:
  - **T1= (1, 4, 4)**
  - **T2= (1, 5, 5)**
  - **T3 = (1.5, 20, 20)**

76

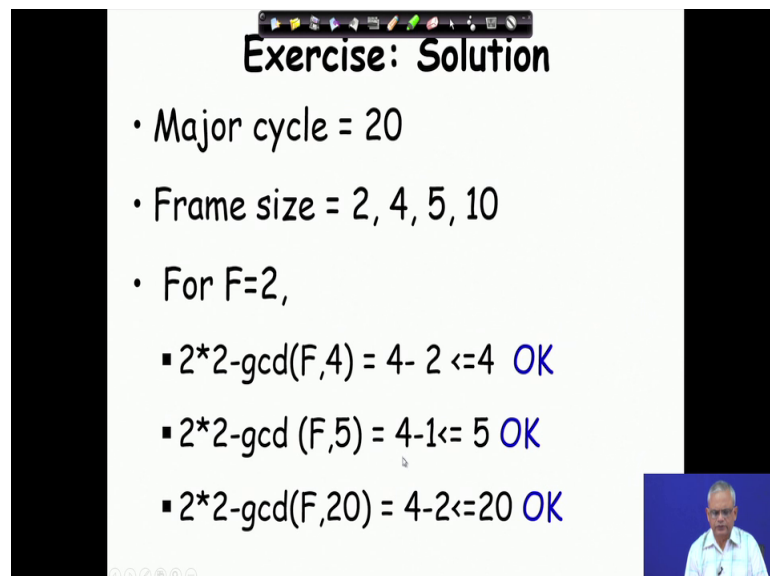
T 1 execution time is 1 and the period is 4 deadline is 4, task 2 execution time is one and period and deadline are 5 units, tasks 3 execution time is 1.5 units the task period and deadline are 20 units and we need to derive the schedule for this and once we derive the schedule it can be stored as the schedule for one major cycle and then the cyclic executive will keep on fetching the next task on each frame interrupt. So, just asking what will be the major cycle for this 3 tasks T 1, T 2, T 3 we have the task periods as 4, 5

and 20. We know that the major cycle is the LCM of T 1, T 2 and T 3. So, that would get 20 as the major cycle I hope everybody is getting that.

Now, the next thing is to select the frame size and we have to see that it satisfies 3 constraints. The first constraint is that the frame size must divide the major cycle and major cycle is 20 and that gives us some discrete values. So, the frame sizes that are possible are I hope you are also getting the same thing that 1 2 4 5 10 and of course, 20. The second constraint will put a lower bound on the frame size which is that the execution time of each task must be accommodated in one frame and that gives us that the frame size must be greater than each of the task execution times. So, 1; 1; 1.5, the execution time has to be at least 2. So, 1 is ruled out. So, now, we have to choose between 2 4 5 10 and 20.

Now, the third constraint which says that there must be a clear frame between the arrival of a task and the deadline and that if we ensure for the worst case then it will hold for all cases. So, for each task we have check  $2 \text{ into } F \text{ minus GCD } F \text{ comma } p_i$  should be less than equal to  $d_i$ , so that we need to do for all the 3 tasks for the different frame sizes. Let us start with the frame size 2. Please try.

(Refer Slide Time: 05:20)



**Exercise: Solution**

- Major cycle = 20
- Frame size = 2, 4, 5, 10
- For F=2,
  - $2*2 - \text{gcd}(F,4) = 4 - 2 \leq 4$  OK
  - $2*2 - \text{gcd}(F,5) = 4 - 1 \leq 5$  OK
  - $2*2 - \text{gcd}(F,20) = 4 - 2 \leq 20$  OK

So, this is what I have written down here. So, the major cycle is 20 that everybody would have got and the frame sizes are 2 4 5 10 because 1 is ruled out that will not hold the task executions.

So, now, let us try the frame size 2 for the third constraint,  $2 \leq F - \text{GCD}(F, p_i)$  one the period of the task one is 4 and therefore,  $4 - 2$  which is 2 is less than equal to the  $d_1$  the deadline for task one is 4 which is holds ok. We check for the second task with frame size 2 we do the similar thing here  $\text{GCD}(F, 5)$  and  $F$  is 2, so  $4 - 1$  which is also less than equal to 5. So, this is and similarly we check for the third task  $\text{GCD}(F, 20)$ . So,  $F$  is 2 and therefore, the GCD is 2 and again this holds. So, 2 is a possible frame size, but what about the number of frames that are required by the schedule.

So, the frame sizes if you see here are 4, 5 and 20. So, in one major cycle the third task will occur once the second task will occur 4 times and the first task will occur 5 times. So,  $5 + 4 + 1 + 5 + 4 + 9 + 1$  is 10. So, we need at least 10 frames and therefore, this just holds  $F$  equal to 2.

(Refer Slide Time: 07:45)

**Example 3**

- For  $F=4$ ,
  - $2*4 - \text{gcd}(F,4) = 8 - 4 \leq 4$  OK
  - $2*4 - \text{gcd}(F,5) = 8 - 1 \leq 5$  Not OK
  - $2*4 - \text{gcd}(F,20) = 8 - 4 \leq 20$  OK
  - **Frame size=2 can be chosen**

Now, let us see for frame size 4. The frame size 4 we again look at the deadline constraint that there must be a clear frame between the task arrival and the deadline and that is given by the expression the worst case is given by the expression  $2 \leq F - \text{GCD}(F, p_i)$  and we see here for the second task it does not hold.

And also if we think how many frames will available in one major cycle we see that there are 5 frames because frame size is 4 and therefore, there will be 5 frames that will be available and here for the 3 task you need 10 frames. So,  $F$  equal to 4 is not suitable first is it violates the deadline constraint and not enough frames are available to develop

the schedule and therefore, no point in looking for frame size 5 10 etcetera because the number of frames available will be still less. So, frame size 4 is sorry frame size equal to 2 is the one that can be chosen and once we have the frame size 2 then we can easily the schedule which frame in which frame which task will execute.

(Refer Slide Time: 09:25)

**What If No Frame Size is Suitable?**

- Possible culprit is a task with large execution time e.g. (20,100,100):
  - Prevents a smaller frame size to be chosen
  - Try splitting task with large execution time into two or three sub-tasks.
  - Example:  $T_1=(20,100,100)$  can be split into (10,100,100) and (10,100,100).

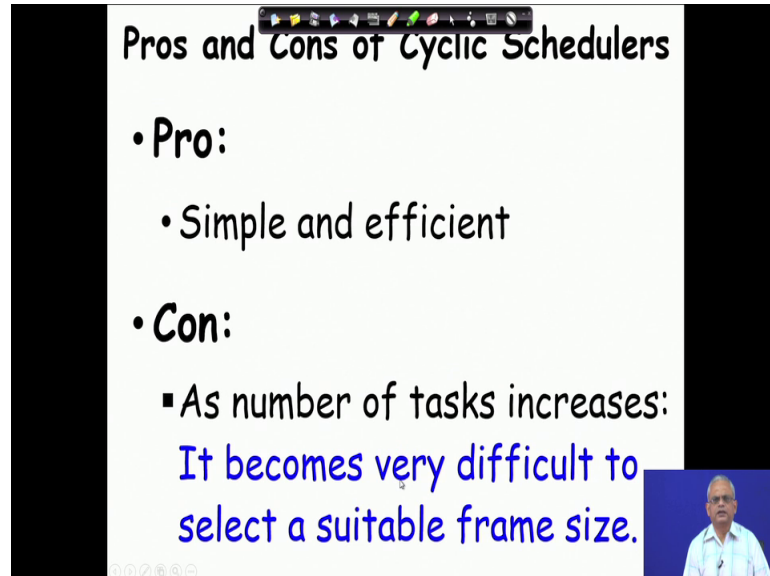
But what happens if we try out all possible frame sizes and find that none of the frame size is suitable. If we think of it the major culprit in this case is a task with execution time for example, if we have many tasks with execution time of 3 4 5 etcetera and there is one with 20, 100, 100. Obviously, the frame size need to be large here 20 at least and the number of frames will become less, only let us say the major cycle is 100 number of frames become 5 and not only that the deadline constraint will also be violated. So, how do we get about the when we have a large task with large execution time.

We need to bit of rewrite our program and we need to split this task into smaller tasks. So, let us assume that we split it into let us say 10 and 10, we look at the code and then decide what can be a possible smaller task size. So, the same task you need to split in 2 and; obviously, we must ensure that end of the task let us say we make the  $T_1$  is  $T_{1a}$  and  $T_{1b}$ . So,  $T_{1a}$  the end of  $T_{1a}$  it must leave the system with a consistent state.

So, that even if other tasks they execute they should not get inconsistent values similar with  $T_2$  sorry  $T_{1b}$ . So, whenever we cannot really get any feasible frame size we need to split the tasks with longer execution times. We will not go into details of that. Just

given the indication and based on that it will not be really very difficult to again rework on the frame size and see that if a feasible schedule can be developed.

(Refer Slide Time: 11:46)



**Pros and Cons of Cyclic Schedulers**

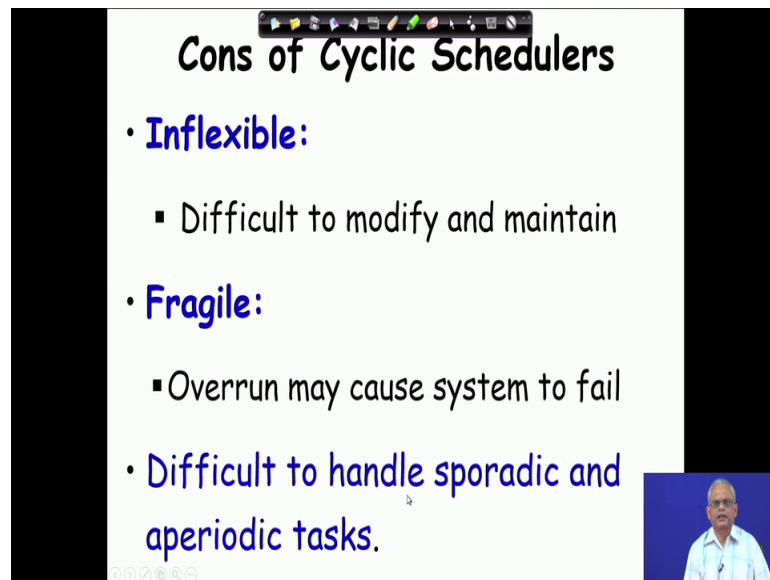
- **Pro:**
  - Simple and efficient
- **Con:**
  - As number of tasks increases:  
It becomes very difficult to select a suitable frame size.

So, as you can see that this is iterative process sometimes we do not get correct frame size that we can use and in that case we need to restructure the tasks and again look for feasible frame size. So, now, we conclude on this cyclic scheduler we look at more sophisticated scheduler, but before that let us just recapitulate on what are strong points and weak points of this cyclic scheduler.

The strong point is that it is a simple and efficient task scheduler. The executive program is very small just needs to consult the schedule table and each time it gets frame interrupt it just runs the task that is there on the schedule table. It is a more efficient even than a table driven scheduler because repeatedly setting a timer is not required here the timer is set only at initialization time.

Now let us look at the negative points for the cyclic scheduler. The first point is that how do we really develop the schedule when the number of tasks become large like slightly a non trivial application where the number of tasks are let us say 20 or 15. Manually looking at all developing a major cycle finding out candidate frame cycles and then trying to check every constraint is actually a non trivial and that too it is a derivative task. So, as the number of tasks increases it may be become very difficult for the designer to find the frame size that is one problem with this scheduler.

(Refer Slide Time: 14:10)



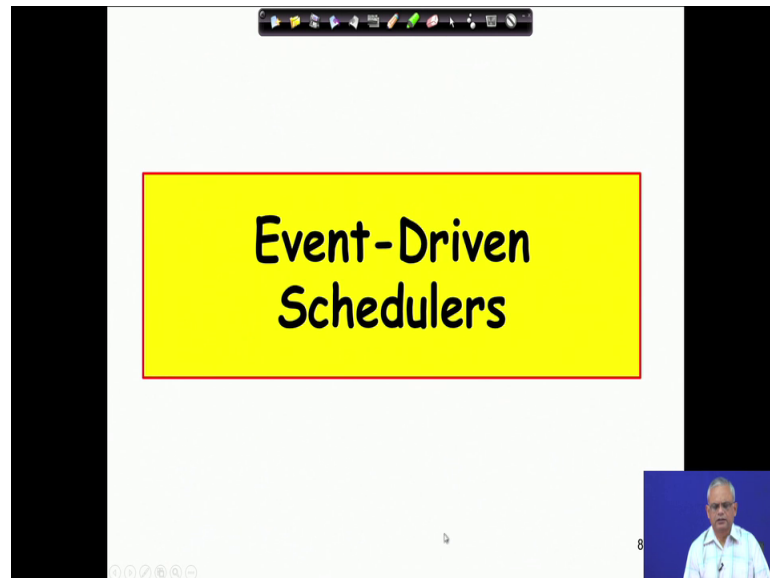
**Cons of Cyclic Schedulers**

- **Inflexible:**
  - Difficult to modify and maintain
- **Fragile:**
  - Overrun may cause system to fail
- **Difficult to handle sporadic and aperiodic tasks.**

The second one is that it is difficult to modify and maintain what if we need to just incorporate a new device and therefore, the execution time of one of the tasks become longer. So, in those case we have entirely rework the schedule it is not just that we change the schedule for that task we need to rework for the entire schedule has to be reworked new frame size and so on.

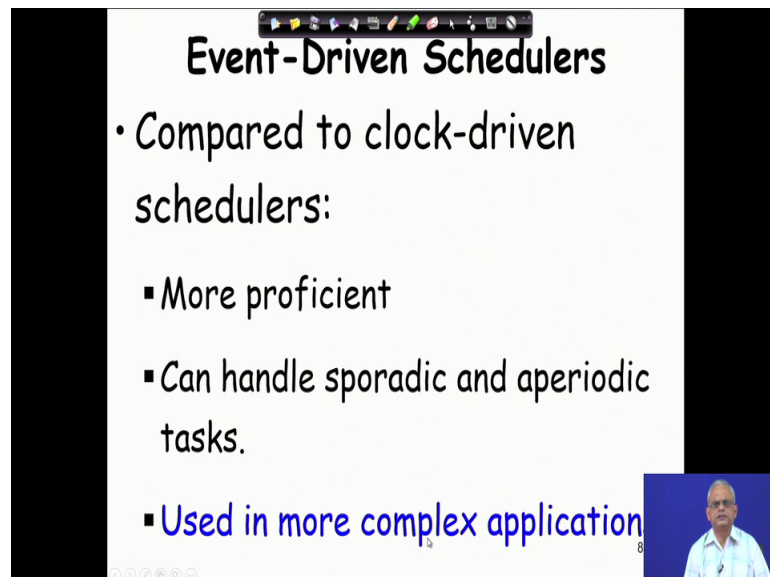
These are fragile because if for any reason one of the task takes longer than it may leave the system in inconsistent state and it the system may fail. Also in many applications we have sporadic and aperiodic tasks and there is no clear way in which you can handle the sporadic and aperiodic tasks. So, these are the weak points of the cycling scheduler we must remember that.

(Refer Slide Time: 15:23)



Now, let us look at more sophisticated schedulers the clock driven schedulers are simple used in simple applications, but as the applications becomes more sophisticated we need to deploy the event driven schedulers. Let us look at the event driven schedulers the major classes of the event driven schedulers and the issues that need to be handled.

(Refer Slide Time: 15:54)



Compared to the clock driven schedulers these are more proficient. What does that mean? It means that for a given task set even if a clock driven scheduler cannot find a schedule it can be easily executed at the feasible schedule can be obtained using the

event driven scheduler. So, these are more proficient. This can handle the sporadic and aperiodic tasks and naturally these are used more complex applications, but these are not as efficient as the clock driven scheduler both in terms of the code size of the schedulers and also the execution time of the schedulers.

(Refer Slide Time: 16:50)

**Event-Driven Scheduling**

- **Scheduling points:**
  - Defined by task completion and arrival events.
- **Preemptive schedulers:**
  - On arrival of a higher priority task, the running task may be preempted.
- **Simplest event-driven scheduler:**
  - **Foreground-Background Scheduler**

But then for event driven schedulers what will be the scheduling points if you recollect we had said that as long as we are discussing about tasks schedulers, one of the very fundamental concept is the scheduling point the scheduling point are the times at which the task scheduler becomes active starts running and decides which task to dispatch or start execution next.

So, what will be scheduling points for event driven scheduling. So, there are two scheduling points, two types of scheduling points one is task completion. So, whenever a task is running and it completes that vex of the scheduler that defines a scheduling point and the scheduler starts running and tries to decide which task to run the next.

The other scheduling point other type of scheduling point are the arrival events when a task arrives you know that the task arrives due to an interrupt may be a periodic task can arrive due to a clock interrupt a periodic timer interrupt. So, each time the periodic interrupt occurs the corresponding instance of that task which is called as the job is released or it becomes ready to execute. So, the arrival of every task sorry every instance of a task that is a job defines a scheduling point. So, as the job arrives the scheduler code



starts running and checks whether this is a task which has arrived needs to be executed by preempting the already executing task. So, the event schedulers are also called as preemptive schedulers because whenever a task arrives and it has a higher priority, then the task that is already executing needs to be pre-empted.

The simplest event driven scheduler goes by the name foreground background scheduler. So, this you might have already been become familiar in your first level operating system course the basic operating system course the foreground background scheduler is a popular scheduler even in non real time applications.

(Refer Slide Time: 20:08)

### Foreground-Background Scheduler

- Real-time tasks are run as foreground tasks.
- Sporadic, aperiodic, and non-real-time tasks are run as background tasks.
- Among the foreground tasks, at every scheduling point:
  - The highest priority foreground task is scheduled.
- A background task can run:
  - When no foreground task is ready.

If you recollect the basic ideas in a foreground background scheduler here we have a mixture of periodic, aperiodic, and sporadic tasks. The periodic tasks run in the foreground; that means, that they are given high priority compared to the background tasks. Whenever the foreground tasks do not exist, the background tasks start running, and in a real-time situation, all the tasks having a real-time deadline are run as the foreground tasks. So, these are the periodic tasks which have a deadline, just shown one example in this diagram. If you see here that this line here defines the point at which the periodic task, real-time periodic task, recurs, and since this is the real-time task, it is the foreground task. So, it starts running, displacing all other background tasks, so as it completes, then the background tasks start running. And again, as the periodic timer interrupt comes, the

periodic task again becomes it arrives or is becomes ready and as soon as it becomes ready it starts running.

So, this is the first instance of the periodic task and as it started running the background task have to wait and after its completion at this point the background task start running until the clock interrupts comes the periodic timer interrupts at which the second instance of the periodic task starts running and it completes only then the other aperiodic and sporadic tasks start running. So, it is a simple scheduler where as long as there is a foreground task it is it runs. And in more complicated cases there might be multiple types of foreground tasks, in that case the scheduler will choose the highest priority foreground task and only when none of the foreground tasks exists then the background task will run. So, this is a simple scheduler we will just do a problem based on this.

(Refer Slide Time: 23:28)

**Background Task Completion Time**

- Let  $T_B$  be the only background task
  - Its processing time be  $e_B$
  - The time for it to complete would be:

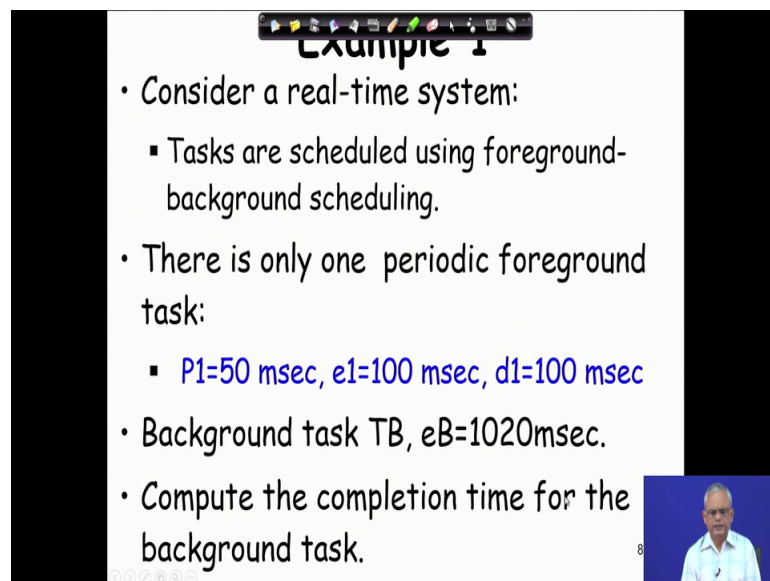
$$ct_B = \frac{e_B}{1 - \sum \frac{e_i}{p_i}}$$

So, before we do the problem let me just give you a mathematical expression if  $T_b$  is the background task and processing time is  $e_B$ . So, the background task we do not consider them having a deadline. So, we just give indicate the background by its execution time. Whereas, the foreground task is  $p_1$  and its execution time is  $e_1$  or we might have a set of foreground tasks we indicate the execution time by  $e_i$  and the period  $p_i$ . So, in that case how long will the background task take to complete.

The formula is simple here if you look at the formula  $e_B$  is the execution time of the background task and each time the task  $e_i$ ,  $t_i$  starts executing it will run for  $e_i$  and out

of every  $p_i$  units the task  $t_i$  will execute for  $e_i$ . So, the fraction of time it will run is  $e_i$  by  $p_i$  if we consider all such periodic tasks you can write  $\sum e_i$  by  $p_i$ . So, the fraction of time that is left put for the background task to run can be given by  $1 - \sum e_i$  by  $p_i$  and therefore, the completion time  $F$  the background task can be expressed as  $e_B$  is execution time of the background task divided by one minus  $\sum e_i$  by  $p_i$ . So, let us do one exercise.

(Refer Slide Time: 25:41)



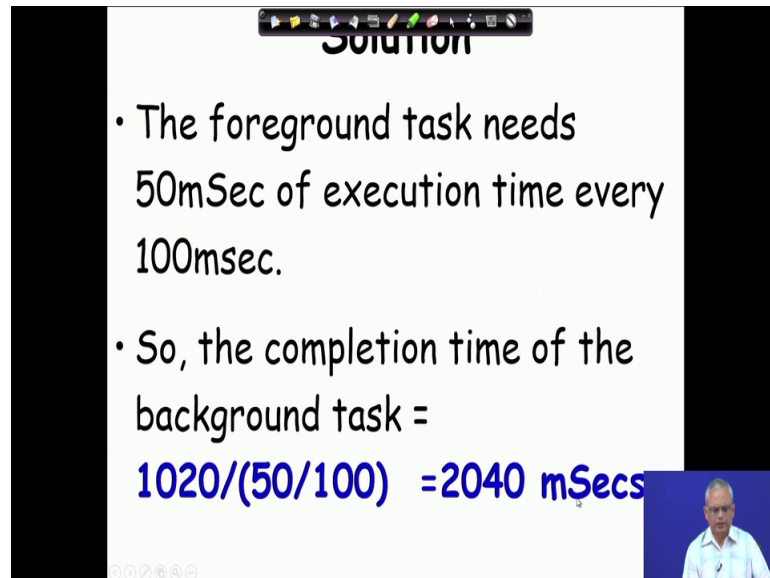
**Example 1**

- Consider a real-time system:
  - Tasks are scheduled using foreground-background scheduling.
- There is only one periodic foreground task:
  - $P_1=50$  msec,  $e_1=100$  msec,  $d_1=100$  msec
- Background task TB,  $e_B=1020$ msec.
- Compute the completion time for the background task.

Let us consider there is very simple case where there is only one real time task which is periodic the period is 50 sorry, the execution time is 50 the period is 100 and the deadline is 100 I just have written here  $p_1$  it should be  $e_1$  and this should be  $e_1$ . So, execution time is 50 period is 100 and deadline is 100, and the background task the execution time is 1020 millisecond.

So, what will be the completion time of the background task that is the question. The answer to this is that every time the foreground task runs it runs for 50 millisecond and it runs once every 100 millisecond. So, the rest of the 50 millisecond is available for the background task to run. So, the utilization due to the foreground is 50 by 100 or 0.5. So, the time fraction of time available for the background task is 0.5. So, you can just substitute that in an expression  $1020$  by  $50$  by  $100$  which is  $2040$  millisecond. So, the background task will complete in  $2040$  millisecond.

(Refer Slide Time: 27:12)



**Solution**

- The foreground task needs 50mSec of execution time every 100msec.
- So, the completion time of the background task =  
 $1020 / (50 / 100) = 2040 \text{ mSecs}$

So, the background task the foreground background scheduler is a very simple scheduler even covered in a first level operating system course you just reviewed because it has relevance for real time applications, but we will not discuss too much on this.

We will look at more sophisticated schedulers the rate monotonic scheduler and the earliest deadline first scheduler, but we will take that up in the next lecture. Now for this lecture we will stop here.