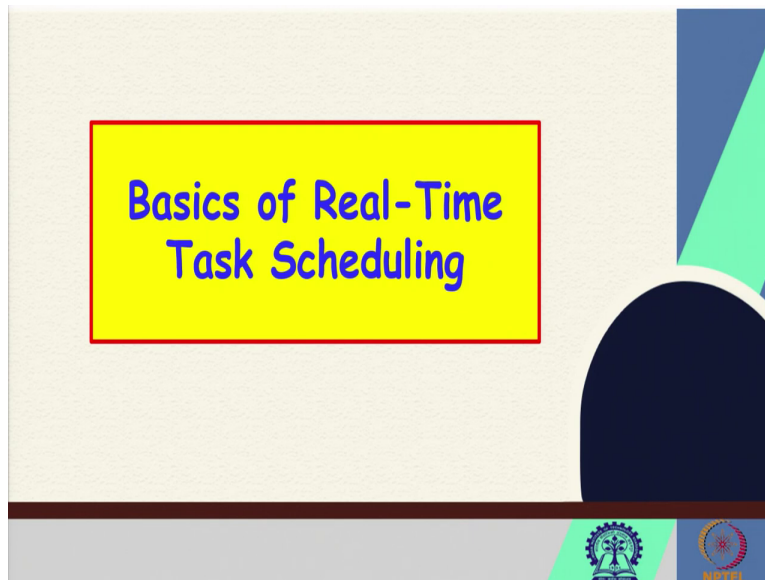


Real Time Systems
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 08
Basics of Real Time Task Scheduling

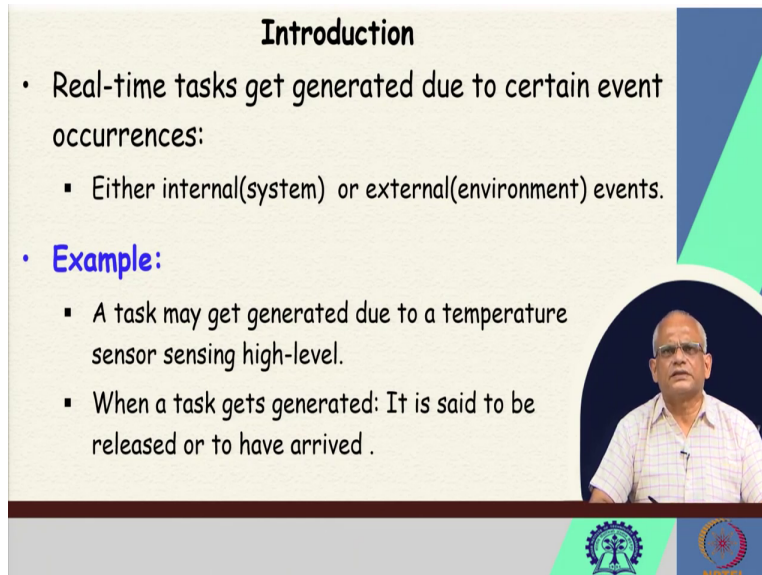
Welcome to this lecture. In the last lecture, we had discussed about timing constraints and how to model a time constraint, we had seen a simple FSM based modeling of time constraints. And we had said that there are many usages of modeling a time constraint and it is also not difficult. Now, let us start with a new topic, look at the different events, how to model them, classification and so on.

(Refer Slide Time: 0:45)



And now, let us start with a new topic, which is Real Time Task Scheduling. We had said that the task scheduling is a very important function of a real time operating system it is the primary mechanism by which the different real time tasks can meet their deadlines. So, this is a very important topic about the task schedulers in a real time system.

(Refer Slide Time: 01:26)



Introduction

- Real-time tasks get generated due to certain event occurrences:
 - Either internal(system) or external(environment) events.
- **Example:**
 - A task may get generated due to a temperature sensor sensing high-level.
 - When a task gets generated: It is said to be released or to have arrived .

The slide features a circular inset of a man in a white shirt speaking. At the bottom, there are logos for a university and a research center.

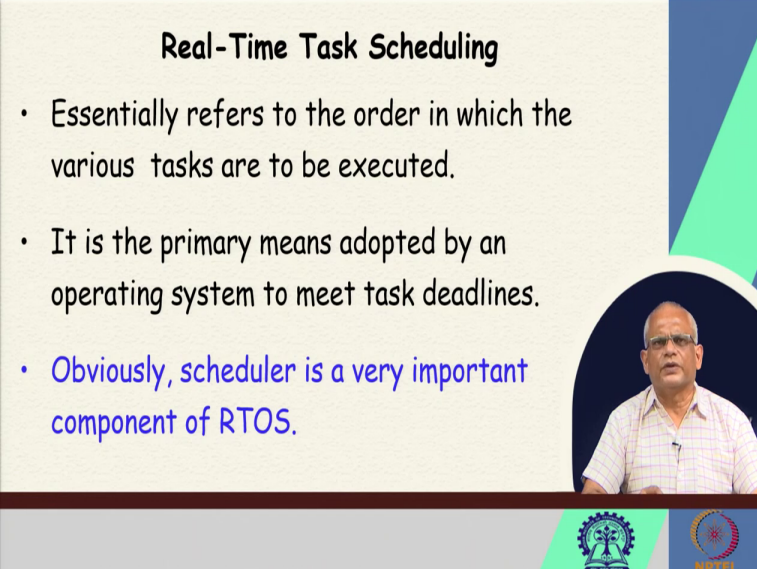
Now, let us again start with some very basic thing about task schedulers. This is a rather large topic, the task scheduling and to understand the real time system well, especially a real time operating system, we need to understand the real time task schedulers well and we will see that a lot of work has been, lot of results, lot of tools etc. are available for the task schedulers, let us get acquainted with them.

The real time tasks they get generated when events occur, the events as we had said that the events can be caused by the environment or by the system. These we call as the internal event which are produced by the system itself for example, a timer alarm occurred. So, there may be some event and then there will be a task generated based on this event. On the other hand, there can be external events which are generated by the environment.

Maybe the threshold of the chemical concentration exceeded, the chemical concentration exceeded the threshold and then there is a event that is generated and the task would be created to handle this situation. Or another example, maybe that the temperature sensor sensed the high level and then it will generate an event, this is the external event and then the system would generate an event when this event, the system will generate a task when this event occurs.

And in the real time operating system terminology, we say that based on the system event of the temperature exceeding a threshold a task is released or a task has arrived. We will use both these terminologies, we will say that a task has been released or we might say that a task has arrived.

(Refer Slide Time: 4:23)



Real-Time Task Scheduling

- Essentially refers to the order in which the various tasks are to be executed.
- It is the primary means adopted by an operating system to meet task deadlines.
- Obviously, scheduler is a very important component of RTOS.

The slide features a video inset of a man in a checkered shirt speaking. The background is light beige with a blue and green geometric design on the right. Logos for institutions are visible at the bottom.

Now, the main thing about the real time schedulers is that they decide which tasks to run when that is the order in which the different tasks that are present in the system will be run. And as we have said that the scheduler is the one which helps all tasks to meet their respective deadlines. The scheduler takes care the one which has sorted deadline, somehow it will make it, meet its deadline, and the other one may be delayed a little bit.

Since the deadline is farther away, we will see how it achieves. There are some standard ways in which this is done. And needless to say, that the scheduler part is possibly the most important component of a real time operating system or RTOS.

(Refer Slide Time: 5:33)

Real-Time Workload

- **Job:**
 - A task instance
 - A unit of work
 - A computation, a file read, a message transmission, etc
- **Task:** a sequence of similar jobs

The diagram shows a horizontal timeline. A red line marks the start as 'Released'. A blue bar represents the 'Execution time'. A red arrow labeled '50msec' indicates the interval between releases. A red arrow labeled 'Relative deadline' points to the end of the execution time. A red arrow labeled 'Absolute deadline' points to a later point on the timeline. A small inset video shows a man speaking.

Now, let us get familiar with one or two terminology. One terminology is about a job. A job is a unit of work, which is to be performed by the system; a job is a unit of work which is performed by a system. And it may be a task instance. For example, let us say periodically, the temperature is sensed, and then some action is being taken.

Let us say every 50 milliseconds the temperature is sensed. So, after every 50 milliseconds a new job or a new unit of work occurs, which we call it a task instance. The task is the temperature sensing, and it has different instances created every 50 milliseconds. And each of these instances we will call a job and in this task instance or job, the workload on the system may be to read some data, to do some computation to transmit a message etc.

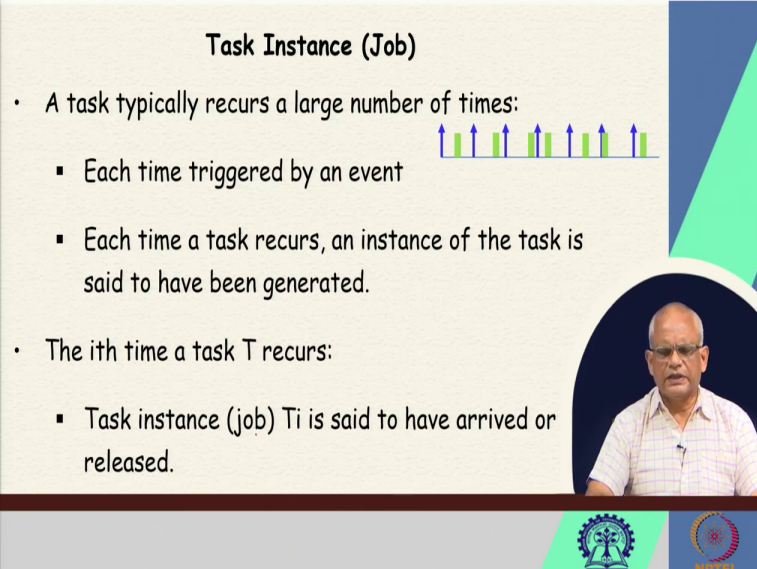
Now, a sequence of jobs or task instances they form of a task, temperature sensing is a task which is sensed every 50 millisecond and each time every 50 millisecond, we have an instance of the temperature sensing task. So, the task is a sequence of similar jobs. And see here that once the temperature is sensed, the temperature is sensed here, we say that the task is released or the task has arrived, but the task does not start execute immediately.

It will start executing based on the scheduler action, which will define when the task will actually execute. And the scheduler let the task execute at this time instant. And then it executes

for some time. And the task might have a deadline. So, the relative deadline is from the instance the task is released to the deadline. That is the relative deadline. On the other hand, the absolute deadline is from the time 0.

Let us say we have zero time here. So, zero time to the deadline is called as absolute deadline. Maybe the relative deadline let us say the task was released at 500 millisecond and the deadline is 100 millisecond from there, which is a 600 millisecond. So, 600 millisecond is the absolute deadline. But the relative deadline is 100 milliseconds. That is the terminology we will use.

(Refer Slide Time: 9:27)



Task Instance (Job)

- A task typically recurs a large number of times:
 - Each time triggered by an event
 - Each time a task recurs, an instance of the task is said to have been generated.
- The i th time a task T recurs:
 - Task instance (job) T_i is said to have arrived or released.

The slide includes a diagram showing a horizontal timeline with several vertical bars of varying heights, representing task instances. A circular inset in the bottom right corner shows a man speaking. Logos for IIT Bombay and NPTIL are visible at the bottom.

Now, we had said that instance of a task is called as a job and the task typically recurs many times. And each time the task is triggered by an event. The event maybe a periodic event, which is usually the case and this can be aperiodic or sporadic and each time a task recurs we say that instance of task has arrived or generated and this is a periodic task here. So, each time there is a clock interrupt, a task instance has been generated, but it may execute depending on when it is scheduled by the scheduler.

It may execute immediately after it is released, or maybe it will execute sometime after it was released. So, for different instances of the task or different jobs, they might execute at different

time after it has been released. And the i^{th} instance of the task T , we represent it by T_i . So, once it occurs, we say that T_i has arrived or T_i is released.

(Refer Slide Time: 11:15)

Relative and Absolute Deadlines

- **Absolute deadline:**
 - Counted from time 0.
- **Relative deadline:**
 - Counted from time of occurrence of task.

The diagram shows a horizontal timeline starting at 0. A task instance T_i occurs at time t_e . The absolute deadline is marked as 'deadline' at time d . The relative deadline is marked as 'relative' at time d . Handwritten red annotations show '900 mSec' for the time from 0 to t_e , '100 mSec' for the time from t_e to 'relative', and '1000 mSec' for the time from 0 to 'deadline'. A small inset video shows a man speaking.

We had just discussed a little bit about the relative and absolute deadline, let us just recapture what we discussed. We said that the absolute deadline occurs with respect to time 0. So, if this is 1000 second or let me just write millisecond, then the absolute deadline of that task instance T_i is 1000 millisecond.

But if this is the enabling event t_e , based on who is that task T_i occurred, then from this instance, let us say this instance was 900 millisecond, then the T_i , the i^{th} instance of tasks T is the enabling event which occurred at 900 millisecond, then the relative deadline is 100 millisecond, whereas the absolute deadline for task T_i is 1000 milliseconds. Very simple terminology. Absolute deadline is counted from 0, whereas the relative deadline is counted from the time of the occurrence of that task.

(Refer Slide Time: 12:39)

Response Time

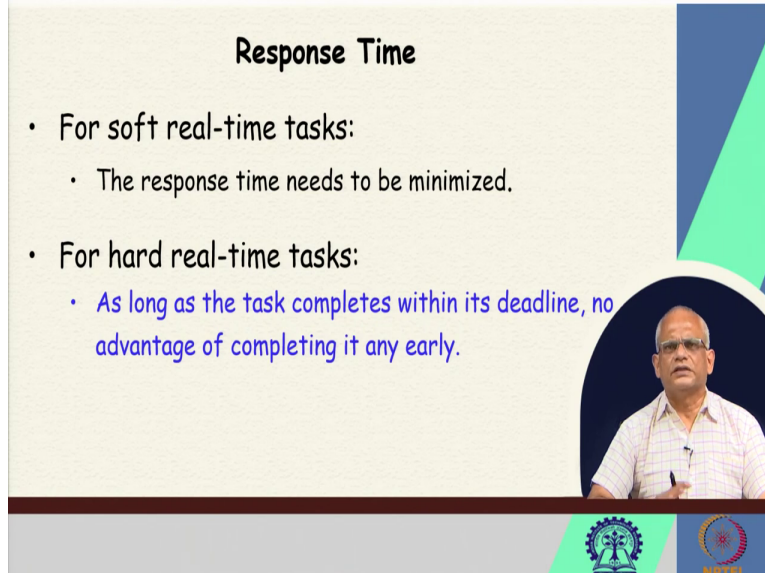
- It is the time between release time and completion time.
- Release time
 - Time of occurrence of the event generating the task.
- Completion time
 - Time at which result is produced by the task

0 e t

Now, let us define another term, which is the response time. The response time is the time from the event based on which the task arrived or the task arrival time to the task completion time, we said that the task takes some time, let us say Δt . And it may get scheduled at different times depending on the scheduler. The workload on the system, the scheduler might schedule at different times, it might schedule it here, or it might schedule it here or it might schedule it somewhere here.

So, if it has scheduled it here, and Δt is the execution time, then the response time is from the arrival time of the task to the completion time of the task. So, that is our definition of the response time. It is a time between the release time or arrival time of a task to the completion time. The release time is the arrival time or the event generation, the time at which the corresponding event occurred. And the completion time is at which the task completes and the result is produced. And typically, the result is produced at the end. So, you can say that the result produced by the task, or the task completes.

(Refer Slide Time: 14:36)



Response Time

- For soft real-time tasks:
 - The response time needs to be minimized.
- For hard real-time tasks:
 - As long as the task completes within its deadline, no advantage of completing it any early.

The slide features a speaker inset in the bottom right corner showing a man in a checkered shirt. The background has a blue and green geometric design. At the bottom, there are logos for institutions, including one with a tree and another with a gear.

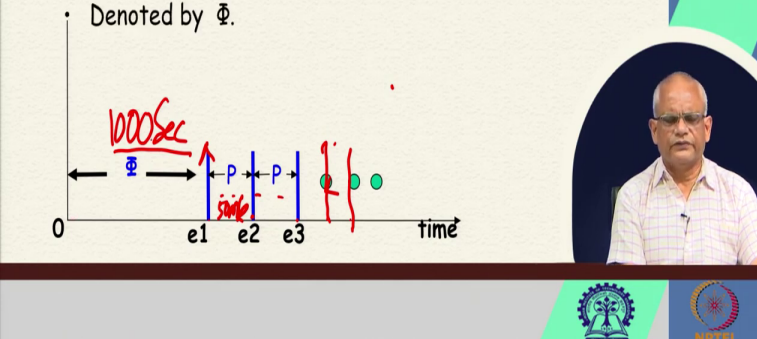
Now let us, for the response time let us just look for soft real time tasks. Here, one objective of the operating system is to minimize the response time of the soft real time tasks in a real time system, there may be many type of tasks, hard real time tasks, soft real time tasks, firm real time tasks and so on. For soft real time tasks, the response time needs to be minimized. Example of a soft real time task is the user requests the current system readings, the current system health parameters, let us say this is a soft real time task, it is a request from the user, and the system must show it as early as it can.

And therefore, for soft real time tasks, the response time needs to be minimized. But for hard real time tasks, the objective is to meet the deadline. As long as it is produced within the deadline, it does not matter whether it was done early or late or so on. As long as the deadline is met, it is okay. So that is the main difference with respect to the soft real time task and hard real time task, on the response time behavior, soft real time tasks, the objective of the operating system is to minimize the response time. And for hard real time task, the objective for the operating system is to meet the deadline. And there is no advantage in completing the hard real time tasks as early as possible.

(Refer Slide Time: 16:41)

Phase of a Periodic Task

- Phase for a periodic task:
 - The time from 0 till the occurrence of the first instance of the task.
 - Denoted by Φ .



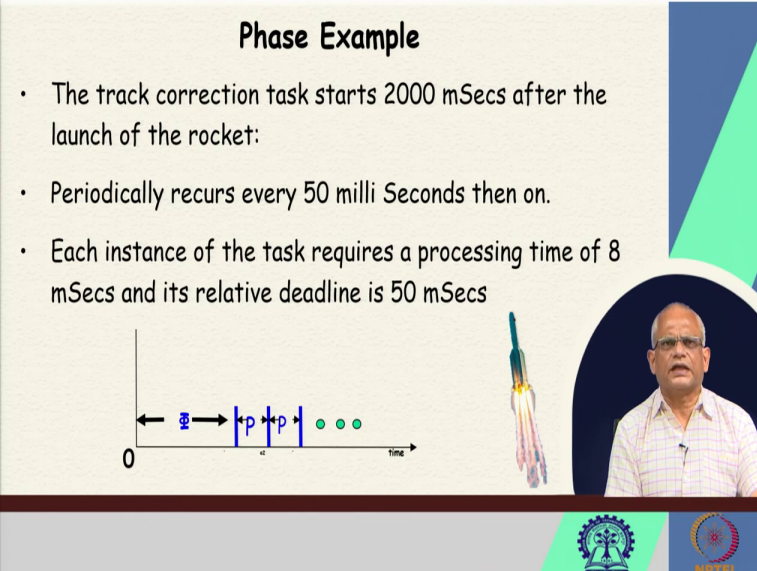
Now, here is another term. We will use this term as we proceed through this course, this is called as a phase of a periodic task, a periodic task recurs or repeats based on a timer alarm, a timer event, let us say e_1, e_2, e_3, e_4, e_5 , etc. These are the timer events. And but the first event occurs after let us say some delay. Let us say after 1000 seconds, the first event occurs and from then on, it just keeps on repeating after every 50 milliseconds.

The first event occurs after 1000 second and after that, the task repeats every 50 milliseconds. So, the phase of the task is 1000 second. Typically represented by Φ the phase of a task is the time from 0 till the first occurrence of that task that we call is the phase, if the first occurrence of the task occurs at time 0, and then we say phase is 0. And we denote the phase by Φ .

(Refer Slide Time: 18:28)

Phase Example

- The track correction task starts 2000 mSecs after the launch of the rocket:
- Periodically recurs every 50 milli Seconds then on.
- Each instance of the task requires a processing time of 8 mSecs and its relative deadline is 50 mSecs



Let us look at an example of a phase with respect to a rocket. Now, let us say once the rocket was fired, initially it accelerates at a very large rate, and after 2000 milliseconds, that is 2 seconds of the launch of the rocket. The track corrections task recurs every 50 milliseconds from 2000 seconds. So, initially when the acceleration is very high, the task correction task does not take place.

But then, after every 50 millisecond. The first task correction track corrections task occurs after 2000 milliseconds. And from then on, it occurs every 50 milliseconds. So, the phase of the track correction task is 2000 milliseconds. So, this is the definition of the phase of a task.

(Refer Slide Time: 19:51)



A Few Important Scheduling Terminologies

- **Valid Schedule:**
 - At most one task is assigned to a processor at any time.
 - No task is scheduled before it is ready.
 - Precedence and resource constraints of all tasks are satisfied.
- **Feasible Schedule:**
 - A valid schedule in which all tasks meet their respective time constraints

Now, let us just get used to a few important terms, which we will use again and again. With respect to the different schedulers that we will discuss, one term we will use is a valid schedule, a valid schedule is one where at most one task is assigned to the processor, if you do not assign any task to the processor, like the processor is idle, no problem, but you cannot assign two tasks to one processor that becomes invalid.

And no task can be scheduled by the scheduler before it arrives, very natural cannot run a task even before the enabling event occurs. And other constraints in the task have been satisfied that it should follow some other task, it should have some critical resources etc., and then only it can run. So, this is the definition of a valid schedule that it does not violate the basic requirement that at any time a single task is assigned to the processor the task does not run before it arrives.

And also, the task has satisfied its precedence constraints like which tasks are complete and then it is enabled and then the resource constraints that if it needs some critical section, critical data then it should have that data. Now, another term is a feasible schedule, a feasible schedule is a valid schedule in which all the tasks meet their time constraints, in the valid schedule some tasks may not meet their time constraints, we just had the basic thing that one task is assigned to a processor, task is not run before it arrives and the precedence constraints etc. are satisfied. But

then the valid schedule becomes a feasible schedule if all the tasks meet their respective deadlines.

(Refer Slide Time: 22:35)

A Few Important Scheduling Terminologies

- **Proficient Scheduler:**
 - A scheduler S_1 is as proficient as another Scheduler S_2 :
 - If whichever task sets that S_2 can feasibly schedule so can S_1 , but not vice versa.
- **Equally proficient schedulers:**
 - If a task set scheduled by one can also be scheduled by the other and vice versa

The slide includes a Venn diagram with a yellow circle labeled S_1 and a blue circle labeled S_2 inside it, illustrating that S_1 is more proficient than S_2 . Another Venn diagram shows two overlapping circles, both labeled S_2 , illustrating that two schedulers are equally proficient. A video inset shows a man speaking at a podium.

Now, we call a scheduler, a proficient scheduler, then another scheduler, more proficient which say that a scheduler is one is as proficient as another scheduler is to, if given a task set, if S_2 can feasibly schedule S_1 then S_1 can feasibly schedule, if S_2 can schedule it, then S_1 can schedule it, but not vice versa. So, the green one that you see here that is S_2 is scheduling all these tasks set each point here is some instance of a task set.

So, for the task set that is scheduled by S_2 , S_1 also can feasibly schedule them, it can find schedules in which their deadlines are met. But there may be some tasks sets which S_1 can feasibly schedule but S_2 cannot, but for all tasks set that S_2 can feasibly schedule so can S_1 then we say that S_1 is more proficient than S_2 or S_1 is at least as proficient as S_2 , so, we can use these terms.

And when two schedulers will look at various types of scheduler and we will say that, let us say the EDF is more proficient than let us say the rate monotonic schedule. Then what do we mean is that whatever tasks can be feasible scheduled by the rate monotonic scheduler, EDF can schedule

them. But maybe there will be some tasks set which EDF can schedule, not the rate monitor. Two schedulers are equally proficient.

If a task set is scheduled by one scheduler, then the other scheduler also can schedule it and vice versa. So, it becomes, if this is the our S2, then that is all tasks sets, each task set, we just represented the point here, then S1 also can schedule all the tasks set that is to come, then we say them they are equally proficient schedulers.

(Refer Slide Time: 25:37)

A Few Important Scheduling Terminologies

- **Optimal Scheduler:**
 - An optimal scheduler can feasibly schedule any task set that can be scheduled by any other scheduler.

The slide features a diagram of a large yellow circle labeled 'S4' which contains three smaller circles labeled 'S1', 'S2', and 'S3'. To the right of the diagram is a circular inset showing a man in a white shirt. At the bottom of the slide are two logos: a gear-like logo on the left and a circular logo on the right.

Now, we will discuss the concept of an optimal scheduler, an optimal scheduler is one which can physically schedule any task set which can be feasibly scheduled by any other scheduler. For example, we have schedulers S1 which can schedule some tasks feasibly, scheduler S2 which can schedule some other tasks sets, S3 it can schedule all these tasks sets.

Now, we say that S4 is an optimal scheduler because it can feasibly schedule all the tasks set that S1, S2, S3, which are the possible schedulers that are available, S4 is the optimal scheduler. So, that is the terminology we will use that an optimal scheduler can schedule any tasks set which can be scheduled by feasibly scheduled by any other schedulers.

(Refer Slide Time: 26:55)

Scheduling Points

- **At these points on time line:**
 - Scheduler makes decision regarding which task to be run next.
- **Clock-driven:**
 - Scheduling points are defined by interrupts from a periodic timer.
- **Event-driven:**
 - Scheduling points defined by task completion and generation events

Now, we will discuss a very important thing here, which is about scheduling points. The scheduler is basically a piece of code, it runs and then it finds out which tasks to run next and then makes it run. But this scheduler code, it does not run continuously all the time, because on the CPU other tasks would run, the scheduler cannot run all the time. So, the scheduler runs at only certain points of time. That we call as the scheduling points.

In a clock driven scheduler, the time points at which the scheduler will be invoked, which will start running that is defined by interrupts received from a periodic timer like the scheduler runs here, here, here for some time it takes runs very less time maybe just a millisecond or something. But at this point, the once the interrupt received from the timer it runs under the hand in an event driven scheduler only when certain event occurs, the scheduler starts running.

And typically, in event driven scheduler, the events are task completion and task generation. So, once a task is generated, the scheduler runs to find out if these tasks would run immediately or some of the tasks would run or the existing tasks will continue. And if a task completes, it again runs to find out which tasks to run next. So, in the event driven scheduler, typically the scheduling points are the task arrival event at that time the scheduler runs and the task completion when the task completes, runs and decides which tasks to run next.

So, that is the very basic concept about the scheduling point. It is a very important thing. When we discuss about the schedulers, different types of scheduler, we will invariably look at the scheduling points of that scheduler. We are at the end of this lecture. We will stop here and we will continue in the next lecture. Thank you.