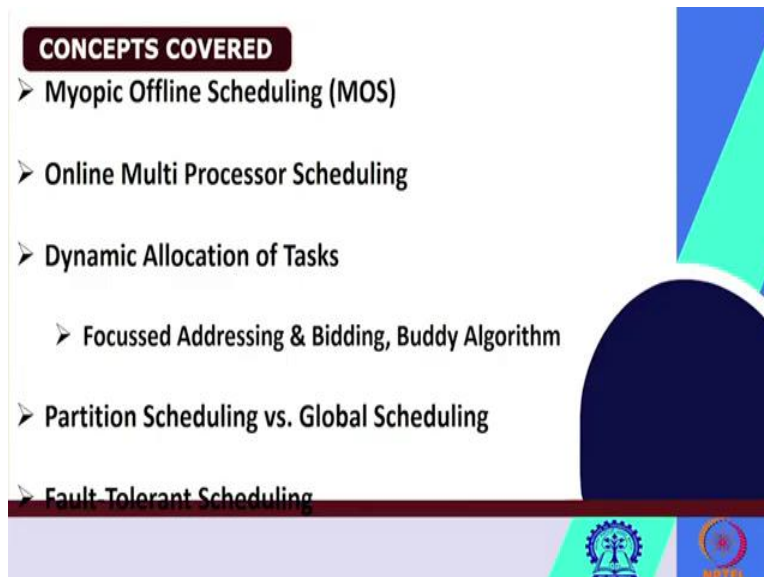


**Real Time Systems**  
**Professor Durga Prasad Mohapatra**  
**Department of Computer Science and Engineering**  
**National Institute of Technology Rourkela**

**Lecture 34**  
**Dynamic Allocation of Tasks**

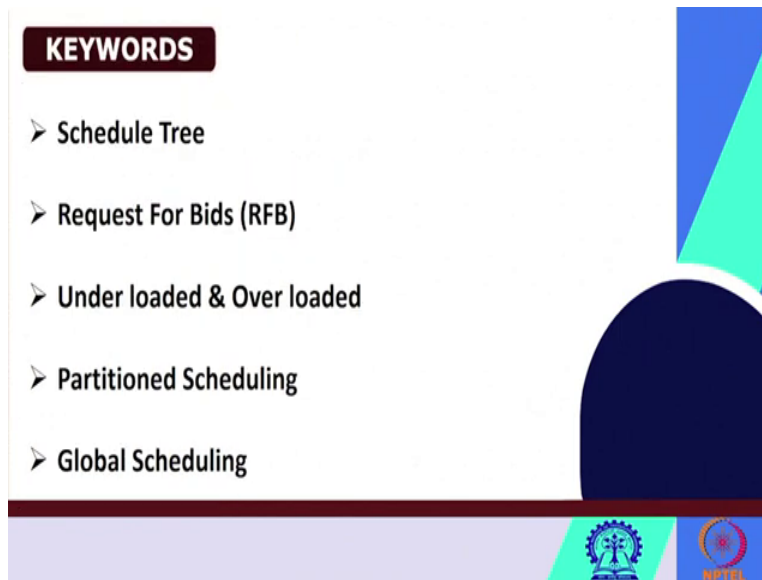
Good morning to all of you. So, in last class we have discussed about static allocation of tasks. We have seen about the different algorithms such as utilization balancing algorithm, next fit algorithm and then this bin packing algorithm. So, today we will discuss about the dynamic allocation of tasks, last class also I have highlighted the differences between the static allocation of tasks and dynamic allocation of tasks. So, before going to the dynamic allocation of tasks, one more this offline allocation is remaining, let us cover first that one.

(Refer Slide Time: 0:00:56)



So, that is myopic offline scheduling. So, before going to the dynamic allocation of tasks, we will first discuss about myopic offline scheduling, then we will discuss about a little bit online multi processor scheduling, then we will go to our actual topic, dynamic allocation of tasks under this you will see two important algorithms, that is focused addressing and bidding and buddy algorithm. We will also discuss about another classification of scheduling or task scheduling, task allocation in multi processor systems that is partition scheduling and global scheduling. Finally, we will briefly discuss about the fault tolerant scheduling.

(Refer Slide Time: 0:01:33)



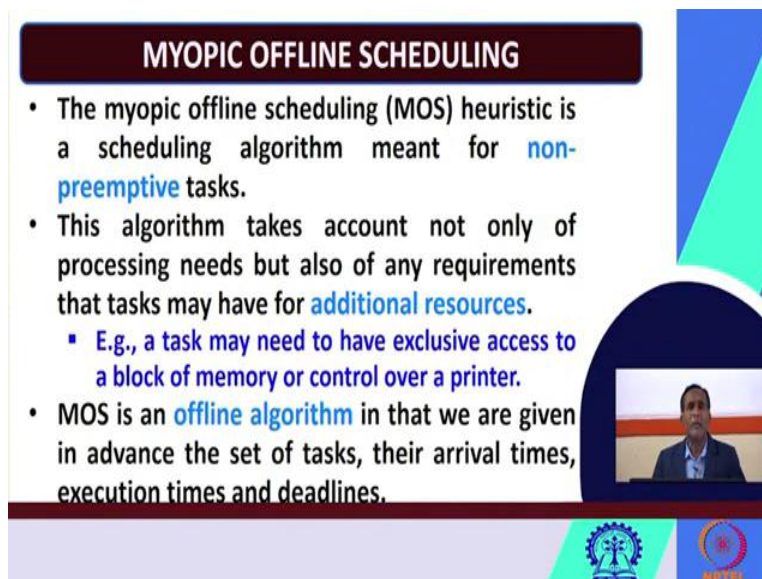
**KEYWORDS**

- Schedule Tree
- Request For Bids (RFB)
- Under loaded & Over loaded
- Partitioned Scheduling
- Global Scheduling

The slide features a dark red header with the word 'KEYWORDS' in white. Below the header, five bullet points are listed, each preceded by a right-pointing arrowhead. The background of the slide is white with a decorative blue and green geometric shape on the right side. At the bottom, there are logos for a university and NPTEL.

Some of the keywords we will use are schedule tree in this myopic offline scheduling, then request for bids in this focused on addressing technique, under loaded and over loaded in this buddy algorithm, then partition scheduling and global scheduling.

(Refer Slide Time: 0:01:50)



**MYOPIC OFFLINE SCHEDULING**

- The myopic offline scheduling (MOS) heuristic is a scheduling algorithm meant for **non-preemptive** tasks.
- This algorithm takes account not only of processing needs but also of any requirements that tasks may have for **additional resources**.
  - E.g., a task may need to have exclusive access to a block of memory or control over a printer.
- MOS is an **offline algorithm** in that we are given in advance the set of tasks, their arrival times, execution times and deadlines.

The slide features a dark red header with the title 'MYOPIC OFFLINE SCHEDULING' in white. Below the header, three bullet points describe the MOS heuristic. The second bullet point includes a sub-bullet point. A small video inset of a speaker is visible on the right side of the slide. The background is white with a decorative blue and green geometric shape on the right side. At the bottom, there are logos for a university and NPTEL.

Let us start with this myopic offline scheduling, previously what algorithms we have seen meant the bin packing algorithm, they are normally made for the preemptive tasks. So, what is about non-preemptive tasks, is there any algorithm exists, if there any algorithm available for scheduling of what non-preemptive tasks is, so that algorithm is myopic offline scheduling.

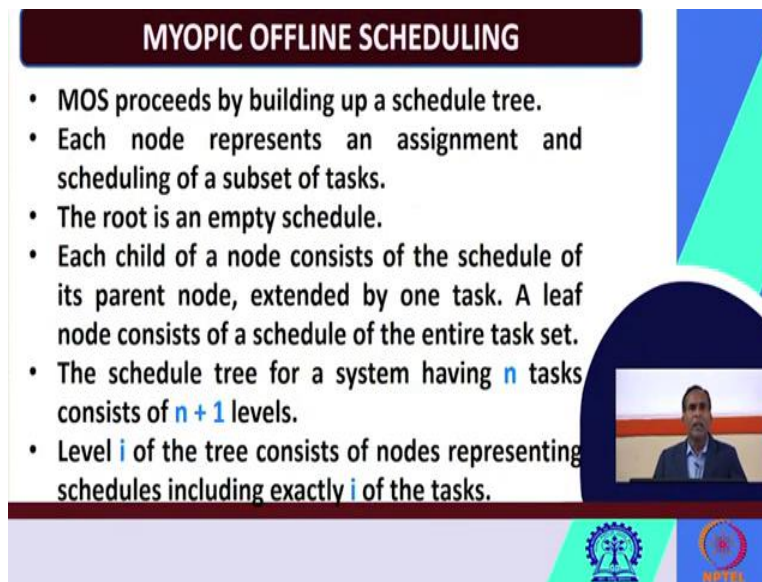
So, the myopic offline scheduling heuristic is a scheduling algorithm which is meant explicitly for the non-preemptive task. This algorithm what does it do? So, because see, please remember in my previous class I have told like in this bin packing algorithm, et cetera, one of the constraint is that the task that require only one resource, that is the processor time. The task that do not require any other additional resources except the processor time.

But here this myopic offline scheduling algorithm it takes account not only of the processing needs, but also of any requirements that the task may have for the additional resources. So, it also handles additional resources which was not we are considering in these previous algorithms such as bin packing algorithm.

So, what could be example of this additional, these tasks, that they may have for some requirement of additional resources? Let us take a small example. A task may need to have exclusive access to a block up memory or it may need to have control over a printer so these kinds of additional resources that can be considered in myopic offline scheduling.

So, this MOS, this Myopic Offline Scheduling, why you are calling it as an offline scheduling algorithm? Because we are here giving, we are giving with, in advance, the set of tasks, their arrival times, and the execution times and deadlines. So, since the set of tasks their, arrival times, execution times and deadlines, they are given in advance to us, so that is why this algorithm is known as an offline algorithm.

(Refer Slide Time: 0:03:55)



**MYOPIC OFFLINE SCHEDULING**

- MOS proceeds by building up a schedule tree.
- Each node represents an assignment and scheduling of a subset of tasks.
- The root is an empty schedule.
- Each child of a node consists of the schedule of its parent node, extended by one task. A leaf node consists of a schedule of the entire task set.
- The schedule tree for a system having  $n$  tasks consists of  $n + 1$  levels.
- Level  $i$  of the tree consists of nodes representing schedules including exactly  $i$  of the tasks.

The slide features a dark red header with the title in white. The main content is a list of six bullet points. To the right of the text is a video inset showing a man in a suit speaking. At the bottom right, there are two logos: one for a university and another for 'NOTES'.

Now, let us see some of the process how this myopic offline scheduling takes place. This myopic offline scheduling, it proceeds by constructing a schedule tree, a tree called as a schedule tree. So, what do you mean by schedule tree? Obviously schedule tree is a tree, where some nodes are there, so what does the node represent?

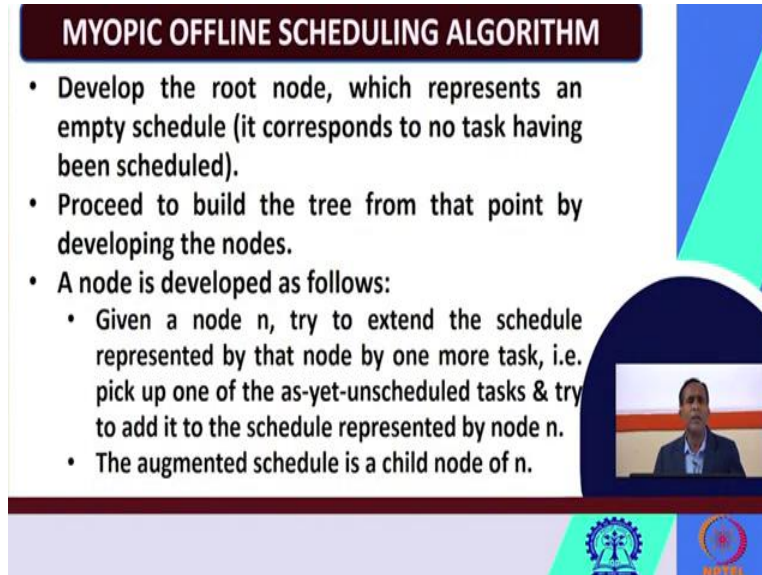
So, each node represents an assignment and scheduling of a sub set of tasks. So, if it is a tree there must be root node, what does it represent? The root node is just an empty schedule. The root node, it represents an empty schedule, so what do the children nodes represent? Each child of a node consists of the schedule of its parent node, extended by one task. So, each child of a node it consists of what? It consists of the schedule of its parent note, extended by one task.

So, now we will see in a tree some leaf nodes are there, so what do the leaf nodes represent? A leaf node consists of a schedule of the entire task set. So, whenever we are coming down to the bottom of the tree, the leaf nodes are present. So, what do they represent? A leaf node, it consists of a schedule of the entire set of tasks or the entire task set.

So, if there are  $n$  tasks the let us see how many  $n$  plus 1, how many level will be there, very basic data structure concept. The schedule tree for a system having  $n$  number of task, it consists of  $n$  plus 1 levels. So, now let us see suppose there is a level  $i$ , what does it represent? So, level  $i$  of the tree, it consists of the nodes, representing the schedules including exactly the  $i$  of the task.

So, when we are considering level  $i$ , so level  $i$  of the tree, it consists of the nodes, representing the schedules, including exactly  $i$  of the task. So, if you are saying the level 2, so level 2 of the tree it consists of the nodes representing the schedules including exactly 2 of the task.

(Refer Slide Time: 0:06:04)



**MYOPIC OFFLINE SCHEDULING ALGORITHM**

- Develop the root node, which represents an empty schedule (it corresponds to no task having been scheduled).
- Proceed to build the tree from that point by developing the nodes.
- A node is developed as follows:
  - Given a node  $n$ , try to extend the schedule represented by that node by one more task, i.e. pick up one of the as-yet-unscheduled tasks & try to add it to the schedule represented by node  $n$ .
  - The augmented schedule is a child node of  $n$ .

The slide features a dark red header with the title in white. The main content is a list of bullet points on a white background. A small video inset in the bottom right shows a man speaking. At the bottom, there are two logos: a blue gear-like logo on the left and a red circular logo with the text 'NPTEL' on the right.

Now, let us see the algorithm for myopic offline scheduling. So, first you develop the root node. I have already told you the root node represents an empty schedule that means it corresponds to no task having been scheduled, because it is the beginning. So, no tasks having been scheduled so far. So, this root node it will correspond to no task having been scheduled.

Then gradually we will proceed, constructing the tree, so next step is proceed to build the tree from where? From that point by developing the other nodes. So, how you can develop the other nodes? A node is developed as per the following process. So, given a node  $n$ , suppose you are given a node  $n$ , then what you should do? Try to extend the schedule represented by that node, how? By one more task.

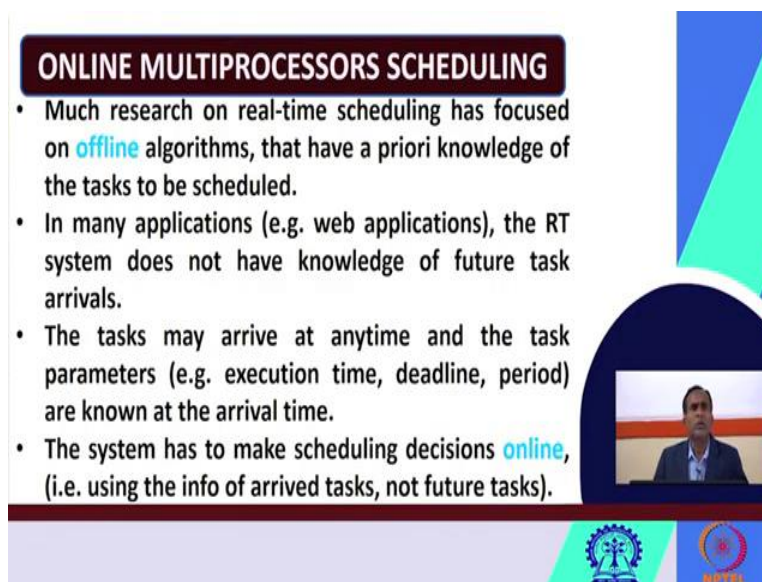
So, if a node  $n$  is given, try to extend the schedule represented by that node, by one or more tasks, that is you have to pick up one of the as-yet-unscheduled task. So, there are so many tasks, they are unscheduled, you pick up one of the as-yet-unscheduled tasks and then try to add it to the schedule represented by node  $n$ .

So, pick up one of the as-yet-unscheduled tasks, the tasks which are so far not scheduled and try to add it to the schedule represented by the node  $n$ . So, now the question is how to pick up this one of the as-yet-unscheduled tasks, here you can use any heuristic, those heuristic may be shall like the period, deadline, et cetera, you may consider.

Some of the heuristic might be like that, you will take the next task having what the lowest period, or the task having the earliest deadline, like that. So, these kinds of heuristics you may use to pick up for any one of the as-yet-unscheduled tasks from the total number of unscheduled task. You can pick up one by using any heuristics such as say largest execution time or least period or earliest deadline, et cetera.

Any of these heuristics you can consider for selecting the node or selecting the task from the total number of unscheduled tasks. The augmented schedule is normally a child of node of  $n$ . So, finally the augmented schedule is a child node of  $n$ . So, in this way we have seen how this myopic offline scheduling algorithm, it works.

(Refer Slide Time: 0:08:34)





**ONLINE MULTIPROCESSORS SCHEDULING**

- Much research on real-time scheduling has focused on **offline** algorithms, that have a priori knowledge of the tasks to be scheduled.
- In many applications (e.g. web applications), the RT system does not have knowledge of future task arrivals.
- The tasks may arrive at anytime and the task parameters (e.g. execution time, deadline, period) are known at the arrival time.
- The system has to make scheduling decisions **online**, (i.e. using the info of arrived tasks, not future tasks).

The slide features a video inset of a speaker in the bottom right corner. At the bottom, there are two logos: a tree logo on the left and a circular logo with the text 'MPPS' on the right.

## MYOPIC OFFLINE SCHEDULING

- The myopic offline scheduling (MOS) heuristic is a scheduling algorithm meant for **non-preemptive** tasks.
- This algorithm takes account not only of processing needs but also of any requirements that tasks may have for **additional resources**.
  - E.g., a task may need to have exclusive access to a block of memory or control over a printer.
- MOS is an **offline algorithm** in that we are given in advance the set of tasks, their arrival times, execution times and deadlines.



So, now we will see, so far the algorithms we have seen like bin packing algorithm, myopic algorithm they are all offline algorithms, because, why they are offline? I have already told you, we consider them as offline because we are given advanced set of tasks, their arrival times, execution times and deadlines, et cetera.

But this offline scheduling is always, it is not actually, it may not be applied to every circumstances, but much research till now, on real time scheduling, it has focused on offline algorithms and offline algorithms what we are doing? So, in offline algorithms that have a prior knowledge of the tasks to be scheduled, the detailed knowledge about maybe the task set, the execution time, deadline period, et cetera, they are known in advance.

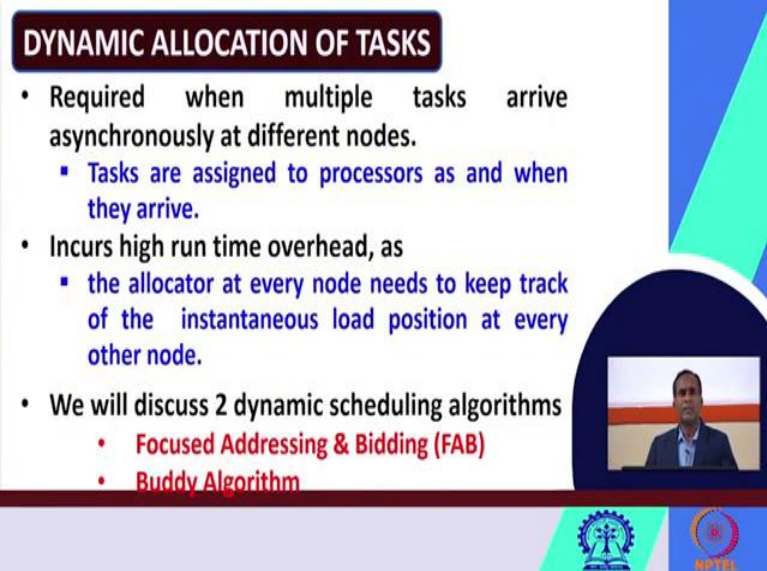
But in many applications, for example the web servers you can take, web applications. So, there the real time system does not have the knowledge of the future task arrivals when which task will arrive. Those information you do not have. You take the example of a web application, a web server, when a task will come, when a task will arrive, that you cannot say in advance.

So, in many applications like web applications or web servers, the real time system, it does not have the knowledge of the future task arrivals. So, these tasks may arrive at anytime and the task parameters, such as the execution time, deadline period, et cetera. They are known only at the arrival time. They are not known prior, they are not known earlier. They can be known only at the arrival time.

So, since these tasks. They will be, or these parameters, they will be only known are these what arrival time, so the system has to make the scheduling decisions online. In those cases, offline algorithms like bin packing algorithm, myopic algorithm, they cannot work. The system has to make the scheduling decisions online. How? By using the information of the current task, by using the information of the tasks, those who have arrived so far, but not by using the information of the future tasks.

So, that is why the system has to make the scheduling decisions online by using the current information of the tasks which have arrived, not based on the information of the future tasks. So, we will not discuss much on this online multiprocessor scheduling, but during the subsequent slides you can say some of the algorithms they can be used, they are actually online in nature. They are online algorithms.

(Refer Slide Time: 0:11:13)



**DYNAMIC ALLOCATION OF TASKS**

- Required when multiple tasks arrive asynchronously at different nodes.
  - Tasks are assigned to processors as and when they arrive.
- Incurs high run time overhead, as
  - the allocator at every node needs to keep track of the instantaneous load position at every other node.
- We will discuss 2 dynamic scheduling algorithms
  - Focused Addressing & Bidding (FAB)
  - Buddy Algorithm

The slide features a video inset of a man in a suit on the right side. At the bottom, there are logos for IIT Bombay and NPTEL.

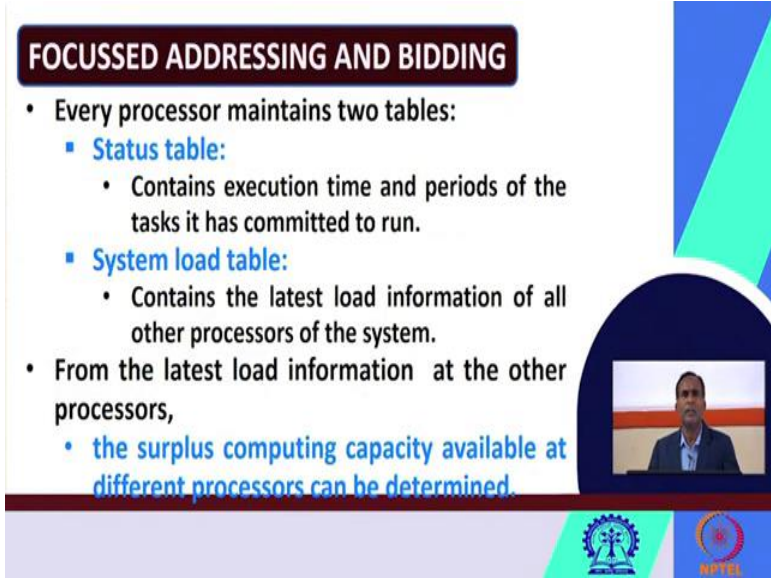
So, now let us go to our original topic, that is the dynamic allocation of tasks. I have already told you last class, the difference between static allocation of task and dynamic allocation of task. These dynamic allocations of tasks, they are required when? They are required when multiple tasks, they arrive asynchronously at different nodes and here the tasks are assigned to processors as and when they arrive, so it is not that you can say that what this is not offline.

You can say that these are also examples of online. Here the tasks are assigned to the processors as and when the tasks arrive. So, this I have already told in the last class, that the dynamic allocation



of tasks, they incur high run time overhead because the allocator at every node, it keeps track of the instantaneous load position at every other node, that is why these dynamic allocation algorithms, they require high run time overhead in comparison to the static algorithms. So, in this class, we will discuss two important dynamic scheduling algorithms, that is focused addressing and bidding and buddy algorithm.

(Refer Slide Time: 0:12:16)



**FOCUSED ADDRESSING AND BIDDING**

- Every processor maintains two tables:
  - **Status table:**
    - Contains execution time and periods of the tasks it has committed to run.
  - **System load table:**
    - Contains the latest load information of all other processors of the system.
- From the latest load information at the other processors,
  - the surplus computing capacity available at different processors can be determined.

The slide features a video inset of a man in a suit and two logos at the bottom: a gear logo and a logo with the text 'NPTU'.

First let us see about the focused addressing and bidding. So, focused addressing and bidding it works like this. In this algorithm two tables are maintained, every processor maintains two tables, one is the status table, another is the system load table. In status table what does it contain? The status table it contains the execution time,  $e_i$  and the periods  $p_i$  of the tasks, it has committed to run. So, this processor, it has committed some tasks to run. So, this processor, it maintains the information such as execution time and periods of the tasks that it has committed to run, where? In the status table.

Now, let us see what does it store in the system load table? The system load table, it contains the latest load information of all other processors of the system. So, one is the current processor, so others are the remaining processors. So, system load table of a processor, it contains the latest load information of all other processors present in the system.

So, now why it requires the latest load information? Because from the latest load information available are the other processors, the current processor, it can compute, it can find out the surplus

computing capacity. How much capacity is still there? How much amount of capacity is still available, which can be used for the different processors.

So, from the latest load information are the other processors. The surplus computing capacity available at different processors can be determined. So, why it require the latest load information? Because from the latest load information available at the other processors, the surplus computing capacity which are available at the different processors, they can be determined and using this information it can allocate the task to the different processors.

(Refer Slide Time: 0:14:14)

**FOCUSSED ADDRESSING AND BIDDING (CONT ...)**

- The time axis is divided into windows:
  - An window is an interval of fixed duration.
- At the end of each window:
  - Each processor broadcasts the fraction of its computing power in the next window that is currently free.
- Every processor on receiving a broadcast,
  - updates the system load table.

**FOCUSSED ADDRESSING AND BIDDING**

- Every processor maintains two tables:
  - **Status table:**
    - Contains execution time and periods of the tasks it has committed to run.
  - **System load table:**
    - Contains the latest load information of all other processors of the system.
- From the latest load information at the other processors,
  - the surplus computing capacity available at different processors can be determined.

Here the time axis is divided into some windows, what is a window? A window is an interval of some fixed duration. You can put the duration at 5 millisecond or 10 millisecond or whatever like that. So, now at the end of each window, what happens? So, let us see how the focus addressing it works. At the end of each window, each processor, it broadcasts the fraction of its computing power in the next window that is currently free.

So, at the end of each window, that means at the end of this fixed duration or at the end of this interval, so each processor it will send the broadcast message regarding what? The fraction of its computing power, what is its total computing power, how much it is already utilized, how much fraction of its computing power in the next window, that is currently free.

So, in the next window, how much computing power is free, which can be given to the other tasks. Suppose the total utilization, suppose is 1, and now it has already utilized 0.6, that means 0.4 amount is free, so that 0.4 can be given to other task in the next window. That is what it means. So, at the end of each window, each processor. It will send a broadcast message, containing the fraction of its computing power, in the next window that is currently free. And since broadcast message it will send to all other nodes.

Now, every processor will receive that broadcast message and what they will do? So, now on receiving the broadcast message, every processor, they will update the system load table, because every processor has two tables, system loads table and another is the status table. So, on receiving this broadcast message, every processor will update the system load table.

(Refer Slide Time: 0:16:03)

**FOCUSSED ADDRESSING AND BIDDING (CONT ...)**

- When tasks arise at a node:
  - The node first checks whether the task can be processed locally, at that node.
  - If yes, then
    - It updates its status table.
  - If not, then:
    - Examines the system load table to look for a processor to offload the task.

**FOCUSSED ADDRESSING AND BIDDING**

- Every processor maintains two tables:
  - **Status table:**
    - Contains execution time and periods of the tasks it has committed to run.
  - **System load table:**
    - Contains the latest load information of all other processors of the system.
- From the latest load information at the other processors,
  - the surplus computing capacity available at different processors can be determined.

Now, let us see how the scheduling or how the allocation takes place. Now, when task arise at a processor or at a node, what happens? The node or that processor, first checks, whether the task can be processed locally at that node, because if the utilization is suppose one, and it is currently 0.6, another 0.4, but the task that has arrived, it is having the utilization 0.3. Then easily it can be fit in there, because 0.4 available, the task that is arriving, its utilization 0.3, then it can be allocated, it can processed in that node or in that processor itself.

But now if the task that has arrived if it is having utilization of 0.5, then what will happen? It cannot be processed by that node, because already 0.6 is already utilized, only 0.4 is left, but the

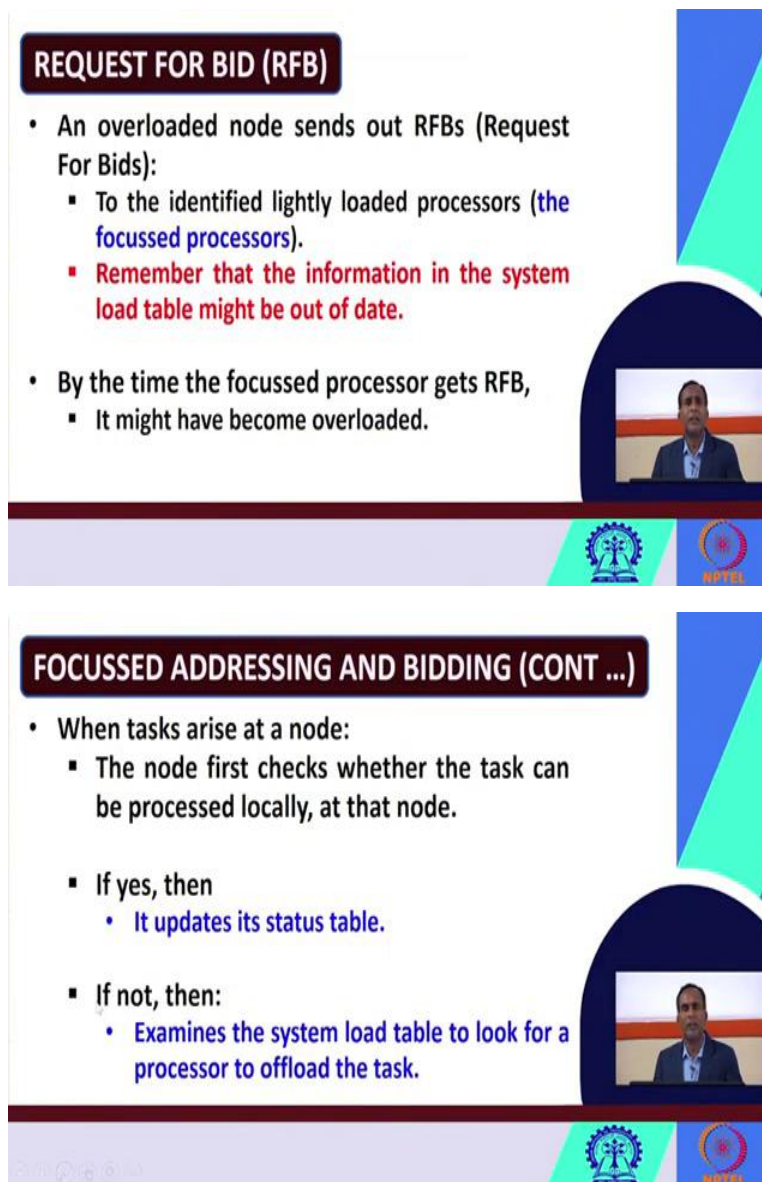
task which is arriving its utilization is 0.5. it is greater than 0.4. So, what will happen? It cannot be processed at that node. So, then what will happen? Let us see what will happen.

So, first whenever any task arrives or arises at a node or a processor, the node or the processor it first checks whether that task can be processed locally in that node or not. If yes, then it is fine, it will update its status table, but if no, that means if that utilization of the task is greater than the currently available utilization, the task cannot be processed locally at that node, in that case what will happen?

If not, then this load, this node or processor, it will examine the system load table. Why it will examine the system load table? Because I have already told you the system load table, it contains what? It contains the latest load information of all other processors of the system. So, by examining the system load table it can get the latest load information of other processors. So, then it can see, okay, processor x is now a little bit free, it is lightly loaded, so accordingly these nodes, these tasks, it will pass this task to that node, say y, which is having sufficient utilization free, which is having, which is lightly loaded. So, then this task, this node x can offload the task, can pass the task to the node y, because it is having what sufficient capacity available.

So, that is what I am saying, I am summarizing quickly what happens in focused addressing mode. So, when any task is arising at a node, the node or the processor it will first check whether it can be processed there or not. If the utilization is less than its free time or free utilization then it can be accommodated there, it can be processed there and then it will update the status table, if the utilization of the arising task is greater than the free available utilization then it cannot be processed locally at that node and hence it will examine the system load table in order to know the latest information of all other processors and it will find out which one is lightly loaded. Then it will upload the task to that processor. It will pass the task to that processor, may be y.

(Refer Slide Time: 0:19:28)



The image shows two presentation slides. The top slide is titled "REQUEST FOR BID (RFB)" and contains a bulleted list. The bottom slide is titled "FOCUSED ADDRESSING AND BIDDING (CONT ...)" and also contains a bulleted list. Both slides feature a small video inset of a man in a suit on the right side and logos for IIT Bombay and NPTEL at the bottom.

### REQUEST FOR BID (RFB)

- An overloaded node sends out RFBs (Request For Bids):
  - To the identified lightly loaded processors (the focussed processors).
  - Remember that the information in the system load table might be out of date.
- By the time the focussed processor gets RFB,
  - It might have become overloaded.

### FOCUSED ADDRESSING AND BIDDING (CONT ...)

- When tasks arise at a node:
  - The node first checks whether the task can be processed locally, at that node.
  - If yes, then
    - It updates its status table.
  - If not, then:
    - Examines the system load table to look for a processor to offload the task.

Now, another step, it is involved in this algorithm it is RFB or request for bid, so now let us say suppose the second case, that means it is not possible here, because it is already overloaded and it is having less amount of computing, what capacity, then it is searching for another available processor in that processor set. So, on an overloaded node, what does it do? It sends out RFBs.

RFB is what? Request For Bid. So, if a processor, suppose processor  $x$  it is already overloaded, it cannot process anymore task, then this overloaded node or processor, it sends out RFBs to whom? To the identified lightly loaded processors. It will check what? The system load table and from the

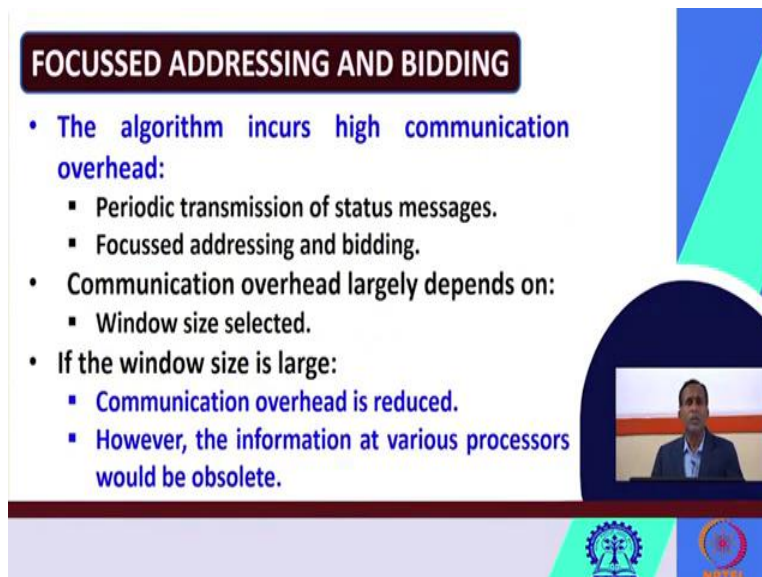
system load table it can determine which processors are little bit lightly loaded, sufficient computing capacity is there, then it will pass that task to that processor which is lightly loaded.

So, an overloaded node, it sends out RFBs to some identified, lightly loaded processors, by looking at this system, by looking at the system load table. So, these identified, lightly loaded processors as known as the focused processors, that is why this is known as focused addressing, but please remember that while the information in this system load table, it is consulting, it is examining the processor, the overloaded processor is examining the system load table and then it is sending the message, these RFBs.

By that time, the information in the system table, might be out of that, because this communication takes some time and by that time this information it may be out of date, that may be obsolete. So, remember that, the information in the system load table might be out of date, might be obsolete due to these communication delays.

By the time the focused processor or the slightly loaded processor gets the RFB, by that time it might have become overloaded, because that information will be obsolete, by that time that lightly loaded processor, it might have accepted some other tasks, and it will become overloaded. So, this problem will happen, in case of this, what focused addressing skills.

(Refer Slide Time: 0:21:57)



**FOCUSED ADDRESSING AND BIDDING**

- The algorithm incurs high communication overhead:
  - Periodic transmission of status messages.
  - Focussed addressing and bidding.
- Communication overhead largely depends on:
  - Window size selected.
- If the window size is large:
  - Communication overhead is reduced.
  - However, the information at various processors would be obsolete.

The slide features a video inset of a man in a suit speaking. At the bottom, there are logos for a university and NPTEL.

Now, let us see the drawbacks of this algorithm, this focused addressing and bidding algorithm, it incurs high communication overhead, because there will be several periodic transmission of status messages, several broadcast messages will be there and also you have to identify which are lightly loaded processors that is focused addressing and bidding.

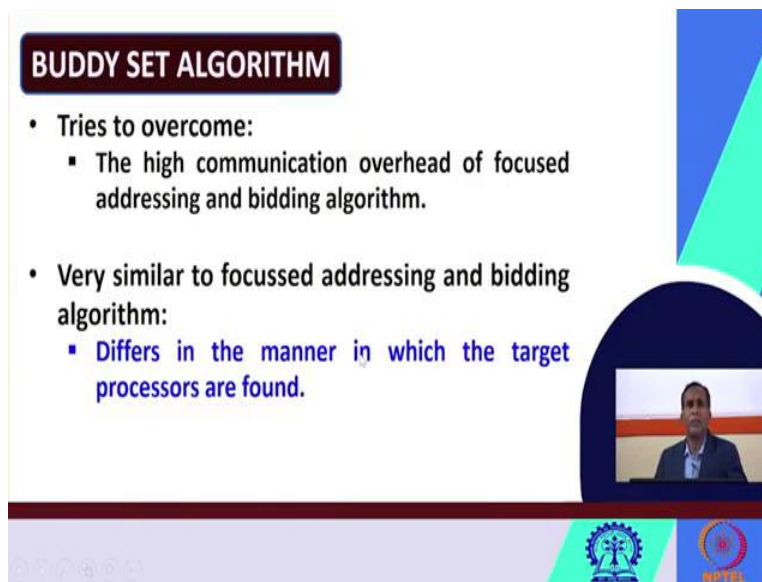
So, due to these two processes, the algorithm incurs high communication overhead. So, the communication overhead largely depends on the window size. I have already told you what time axis it is, it contains some windows. So, windows means interval of fixed duration. So, this communication overhead largely depends upon the window size selected.

If the window size is very large, what will happen? The communication overhead can be reduced, because the window size is large. So, you have to, you may have to do less number of broadcasting, but what will be the drawback, if the window size is large, the information at various processors, by that time, because the time is already a huge gap.

So, by that time the processor receives the message, the information at various processors they will become irrelevant, they will become obsolete. On the other hand if you will make the window size less what will happen? The information may be up to date, but the communication overhead will be very high. So, these are the pros and cons of making the window size large or small, but the communication overhead, it largely depends upon the window size selected.



(Refer Slide Time: 0:23:26)



**BUDDY SET ALGORITHM**

- Tries to overcome:
  - The high communication overhead of focused addressing and bidding algorithm.
- Very similar to focussed addressing and bidding algorithm:
  - Differs in the manner in which the target processors are found.

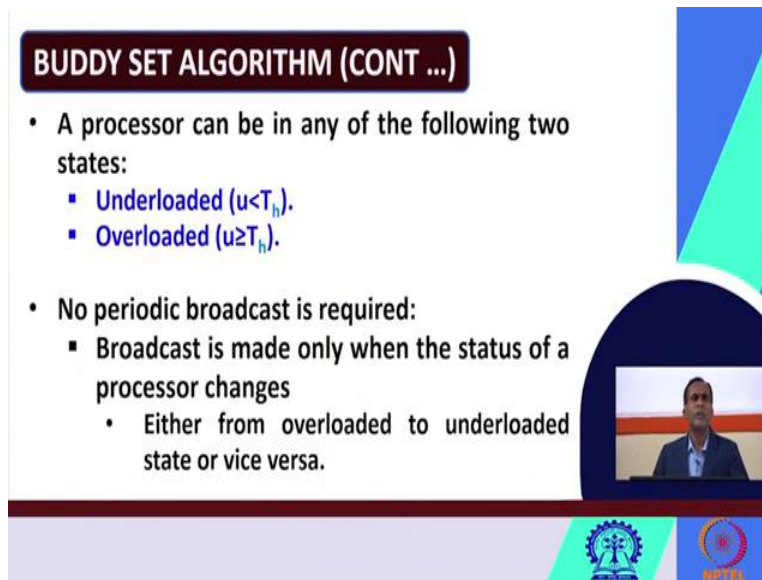
The slide features a dark red header with the title 'BUDDY SET ALGORITHM' in white. Below the title, there are two main bullet points. The first bullet point is 'Tries to overcome:' followed by a sub-bullet 'The high communication overhead of focused addressing and bidding algorithm.' The second bullet point is 'Very similar to focussed addressing and bidding algorithm:' followed by a sub-bullet 'Differs in the manner in which the target processors are found.' To the right of the text, there is a video inset showing a man in a suit. The slide also has a decorative background with blue and green geometric shapes and logos at the bottom, including a tree logo and a 'HOTEL' logo.

Now, let us quickly look at the other algorithm that is the buddy set algorithm. So, I have already told you one of the drawback of focus addressing method is that that it incurs high communication overhead, so this buddy set algorithm, it tries to overcome the high communication overhead of this focused addressing and bidding algorithm.

It is very much similar to the focused addressing and bidding algorithm. Only differs in one way, let us say how does it differ. It differs in the manner in which the target processors are found. So, how that target processors are found? Because in the other case, in focused addressing it broadcast the message to all, then it finds out which one is the lightly loaded or which are the focused processors.

But here the target processors are little bit, we will see it is less, it is not all the, what other processors. So, in that it will be different, so these algorithms differs in the manner, in which the target processor is found. So, not like this focus addressing. It will not broadcast the message to all processors. It will send the message to only some specific processors. So, that is why this algorithm differs in the manner, in which the targets processors are found.

(Refer Slide Time: 0:24:38)



**BUDDY SET ALGORITHM (CONT ...)**

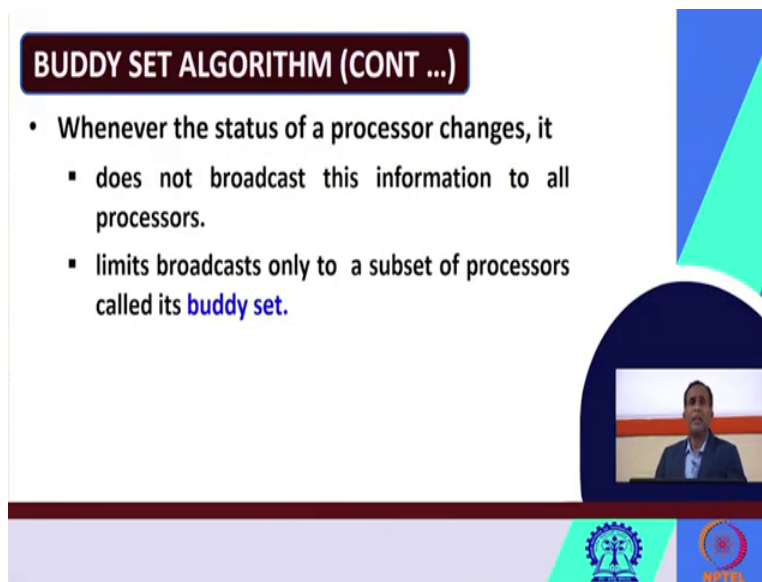
- A processor can be in any of the following two states:
  - Underloaded ( $u < T_h$ ).
  - Overloaded ( $u \geq T_h$ ).
- No periodic broadcast is required:
  - Broadcast is made only when the status of a processor changes
    - Either from overloaded to underloaded state or vice versa.

The slide features a dark red header with the title in white. The main content is on a white background with blue and green accents. A small video inset on the right shows a man in a suit. At the bottom, there are logos for a university and a research center.

Now, a processor in buddy set algorithm may be neither of the two states, either it may be overloaded or under loaded, so if the utilization is less than some threshold value we say that the processor is under loaded, if the utilization is greater than or equal to some threshold value, we call it as over loaded. So, here no periodic broadcast is required. I have already told you the difference.

So, here like in focused bidding periodically you have to send the broadcast message, so here no periodic broadcast is required. So, when you will broadcast the message? So, broadcast will be done, so broadcast is made only when the status of the processor changes. So, when the status of the processor changes either from under loaded to over loaded or from over loaded to under loaded, then only a message will be broadcasted, then only broadcasting will be done.

(Refer Slide Time: 0:25:31)



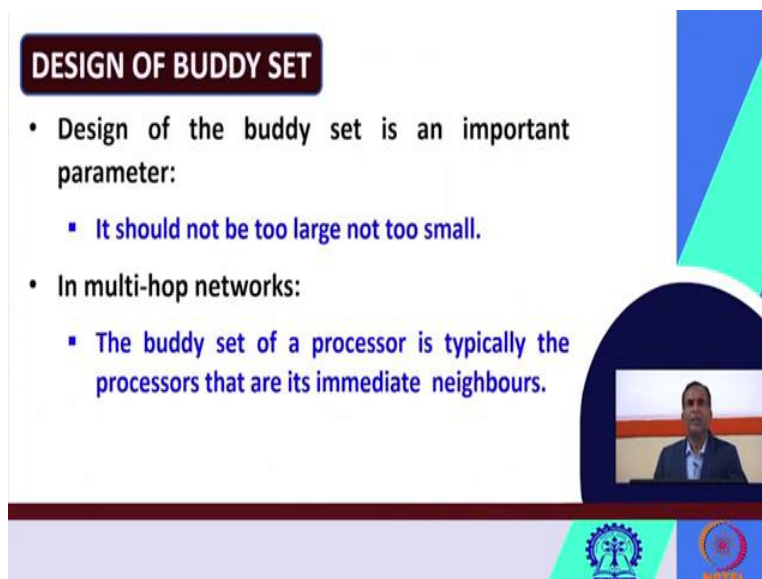
**BUDDY SET ALGORITHM (CONT ...)**

- Whenever the status of a processor changes, it
  - does not broadcast this information to all processors.
  - limits broadcasts only to a subset of processors called its **buddy set**.

The slide features a dark blue header with the title in white. The main content is on a white background with a dark blue footer containing logos for IIT Bombay and NPTEL. A video inset in the bottom right shows a man in a suit speaking.

Now, similarly, whenever the status of a processor changes, as I have already told you one difference, this buddy algorithm does not broadcast this information to all the processors. To where it sends? It limits the broadcast only to a subset of processors. So, instead of sending, because the communication overhead is becoming very much high. So, instead of sending the broadcast message to all the processors, it broadcasts only to a subset of processors and that subset is known as the buddy set.

(Refer Slide Time: 0:25:57)



**DESIGN OF BUDDY SET**

- Design of the buddy set is an important parameter:
  - It should not be too large not too small.
- In multi-hop networks:
  - The buddy set of a processor is typically the processors that are its immediate neighbours.

The slide features a dark blue header with the title in white. The main content is on a white background with a dark blue footer containing logos for IIT Bombay and NPTEL. A video inset in the bottom right shows a man in a suit speaking.

Now, let us see what is the important feature here. So, designing of the buddy set is an important parameter. How to design that buddy set, how to construct that buddy set design, important parameter. These buddy sets should not be too large or should not be too small. So, let us take a small example, how you can find out the buddy set.

If you know different network topologies in multi-hub networks what you can do, the buddy set of a processor, it is typically the processors that are its immediate neighbors, not all the neighbors will consider, which are the immediate neighbors of a processor, they will be served as the buddy set. So, in multi-hub networks the buddy set of a processor is typically the processors which are its immediate neighbors.

(Refer Slide Time: 0:26:41)

**HIGHLIGHTS OF BUDDY SET ALGORITHM**

- **Node State:** Threshold-based:
  - Underloaded (U),
  - Overloaded
- **Selection policy:**
  - The tasks that fail admission test at the local node are considered for transfer.

So, highlights of the buddy set algorithm let us quickly see. So, I have already told you here the nodes that, node can have two states, over loaded or under loaded, based on the threshold, the selection policy here is like follows, the task which fail admission test at the local node are considered for transfer. So, if a task cannot be processed at a particular processor that means it fails the admission test, are that particular local node, then they will be considered, then that will be considered for transferring to another processor.

(Refer Slide Time: 0:27:15)

**BUDDY SET ALGORITHM**

- **Information policy**
  - Based on buddy set.
  - When a node makes transition into or out of a state, it informs its buddies.
- **Transfer policy**
  - One of the buddies is chosen as the receiver, based on the load information available.

The slide includes a video inset of a man in a suit speaking, and logos for IIT Bombay and NPTEL at the bottom.

The information policy is based on the buddy sets, when a node makes a transition into or out of a state, that means either from over loaded to under loaded or under loaded to over loaded. Then it informs, it broadcasts to its buddies or to its buddy sets. The transfer policy is like that, one of the buddies is chosen as the receiver. So, in the buddy set, maybe there are a few numbers of a processors. Out of those few number of processors one of the processor, one of the buddies that means one of the processors is chosen as the receiver based on the local information available.

Like which is having, so what maybe more available computing facility, that buddy or that processor may be chosen as the target processors, may be chosen as the receiver processor to whose? To which the task may be transferred for processing. This is how the buddy set algorithm works.

(Refer Slide Time: 0:28:07)

**MULTIPROCESSOR SCHEDULING ALGORITHMS**

- In addition to static and dynamic-priority algorithms, multiprocessor scheduling algorithms can be classified according to
  - how the tasks are allocated to processors.
- Two categories:
  - **Partitioned Scheduling**
  - **Non-partitioned or Global scheduling**

The slide features a dark blue header with the title in white. The main content is on a white background with a blue and green geometric design on the right. A small video inset shows a man in a suit. Logos for IIT Bombay and NPTEL are at the bottom.

Now, let us quickly look at this another important aspect. So, previously I have already told you for, in the second class we have discussed that one classification of multiprocessor scheduling algorithms is static and dynamic priority algorithms. So, besides these static and dynamic priority algorithms, multiprocessor scheduling algorithms can also be classified according to how the tasks are allocated to the processor. So, based on how the tasks are allocated to the processors, we can make another classification. So, two such categories we will see here, one is partitioned scheduling, another is non-partitioned or global scheduling.

(Refer Slide Time: 0:28:42)

**MULTIPROCESSOR SCHEDULING ALGORITHMS**

- In **partitioned scheduling**,
  - all jobs generated by a task are required to execute on the same processor.
- In **non-partitioned or global scheduling**,
  - **task migration** is allowed, i.e.
  - different jobs of the same task may execute on different processors.
  - **Job migration** is also allowed, i.e. a job preempted on a processor may resume execution on the same or a different processor.

The slide features a dark blue header with the title in white. The main content is on a white background with a blue and green geometric design on the right. A small video inset shows a man in a suit. Logos for IIT Bombay and NPTEL are at the bottom.

So, in partitioned scheduling what is happening? All the jobs generated by a task, they are required to execute on the same processor. In partition scheduling, all the jobs which are generated by a task, they are required to be executed, they are required to be processed on the same processor. This is called partitioned scheduling.

So, the algorithms, like last class we have seen the bin packing algorithms et cetera, they are coming under this partitioned scheduling and let us see what is happening in non-partitioned scheduling or global scheduling. As the name suggests global scheduling so here task migration is allowed. Task migration means what?

The different jobs of the same task, the different jobs raised or the different jobs which are arriving out of the same task, the different jobs of the same task, they may execute on different processor. That is why I am saying that task migration is allowed in global scheduling, similarly job migration is also allowed.

What do you mean by job migration? A job is preempted on a processor. So, supposed a job, it is preempted on a processor, maybe processor x, it may resume execution on the same processor x or a different processor y. That is known as job migration. So, in partitioned scheduling task migration and job migration is not allowed, all the jobs generated by a task are required to execute or processed on the same processor. Whereas in non-partitioned or global scheduling, both, task migration and job migrations are allowed.

(Refer Slide Time: 0:30:09)

## MULTIPROCESSOR SCHEDULING ALGORITHMS

- The above 2 classification schemes give rise to 4 general classes of multiprocessor scheduling algorithms:
  - Partitioned static-priority scheduling
  - Partitioned dynamic-priority scheduling
  - Global static-priority scheduling
  - Global dynamic-priority scheduling



## MULTIPROCESSOR SCHEDULING ALGORITHMS

- In addition to static and dynamic-priority algorithms, multiprocessor scheduling algorithms can be classified according to
  - how the tasks are allocated to processors.
- Two categories:
  - Partitioned Scheduling
  - Non-partitioned or Global scheduling





## MULTIPROCESSOR SCHEDULING ALGORITHMS

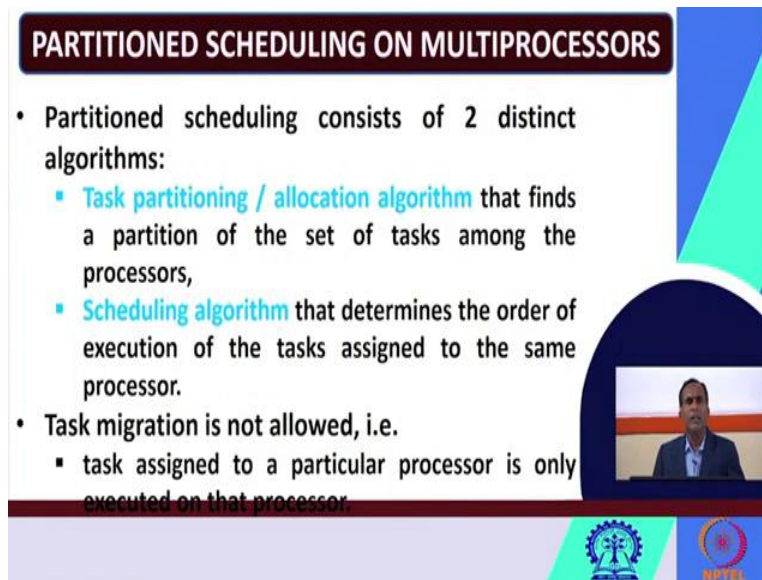
- In **partitioned scheduling**,
  - all jobs generated by a task are required to execute on the same processor.
- In **non-partitioned or global scheduling**,
  - **task migration** is allowed, i.e.
  - different jobs of the same task may execute on different processors.
  - **Job migration** is also allowed, i.e. a job preempted on a processor may resume execution on the same or a different processor.



Now, I have already told you, there are two things, one is static priority scheduling and dynamic priority scheduling, then here we have seen partitioned scheduling and global scheduling. So, based on these we can make the permutations tables the above two classification schemes that is static versus dynamic priority and partitioned versus global algorithm.

So, the above two classification schemes give rise to four general classes of multiprocessor scheduling algorithms. Like partitioned static priority scheduling, partitioned dynamic priority scheduling. So, when I am saying static priority scheduling, I can use RMA. When I am saying dynamic priority scheduling I can use EDF. So, similarly global static priority scheduling and global dynamic priority scheduling. Let us quickly see about, these are all a little bit deeper things. We will not go into deeper. We will discuss only what is partition scheduling and what is global scheduling.

(Refer Slide Time: 0:31:04)



**PARTITIONED SCHEDULING ON MULTIPROCESSORS**

- Partitioned scheduling consists of 2 distinct algorithms:
  - **Task partitioning / allocation algorithm** that finds a partition of the set of tasks among the processors,
  - **Scheduling algorithm** that determines the order of execution of the tasks assigned to the same processor.
- Task migration is not allowed, i.e.
  - task assigned to a particular processor is only executed on that processor.

The slide features a dark blue header with the title in white. The main content is on a white background with a blue and green geometric design on the right. A small video inset shows a man in a suit. At the bottom, there are logos for a university and a research center.

The partition scheduling consists of two distinct algorithms, these we have already told you, because I have already told you the example is bin packing algorithm, and in bin packing algorithm. I have already told you that these algorithm consists of two important approaches. One is task partitioning or allocation algorithm another is scheduling algorithm.

In task partitioning or task allocation algorithm it finds the partition of the task, the classification of the task or the division of the set of tasks, among the processors and then once this partition is made, we can apply any scheduling algorithm. The scheduling algorithm, it determines the order of the execution, the sequence of the executions of the tasks assigned to the same processor.

I have already told you, here in partition scheduling, task migration or job migration, they are not allowed. So, task migration not allowed means, task assigned to a particular processor. It is only executed on that processor. It cannot be what transferred, it cannot be processed at some other different processor.

(Refer Slide Time: 0:32:06)

**PARTITIONED SCHEDULING ON MULTIPROCESSORS**

- **Advantages:**
  - Incurs less runtime overhead because task partitioning can be performed before runtime.
  - Once tasks are assigned to processors, well-known scheduling algorithms with efficient uniprocessor schedulability tests (such as RM and EDF) can be used.

The slide features a dark blue header with the title in white. The main content is on a white background with blue and green accents on the right. A small video inset of a speaker is in the bottom right. Logos for a university and NPTEL are at the bottom.

The advantages of partition scheduling like that, it incurs less run time overhead, because task partitioning it can be performed before run time and once the task are assigned to the processors any well known scheduling algorithm with efficient uni processors schedule-ability test can be used such as you can use RMA or EDF, et cetera.

(Refer Slide Time: 0:32:26)

**PARTITIONED SCHEDULING ON MULTIPROCESSORS**

- The task partitioning problem is analogous to the **bin-packing problem**.
- Among bin-packing heuristics, 4 algorithms for task allocation in partitioned multiprocessor scheduling are popular.
  - First Fit (FF) Algorithm
  - Next Fit (NF) Algorithm
  - Best Fit (BF) Algorithm
  - Worst Fit (WF) Algorithm

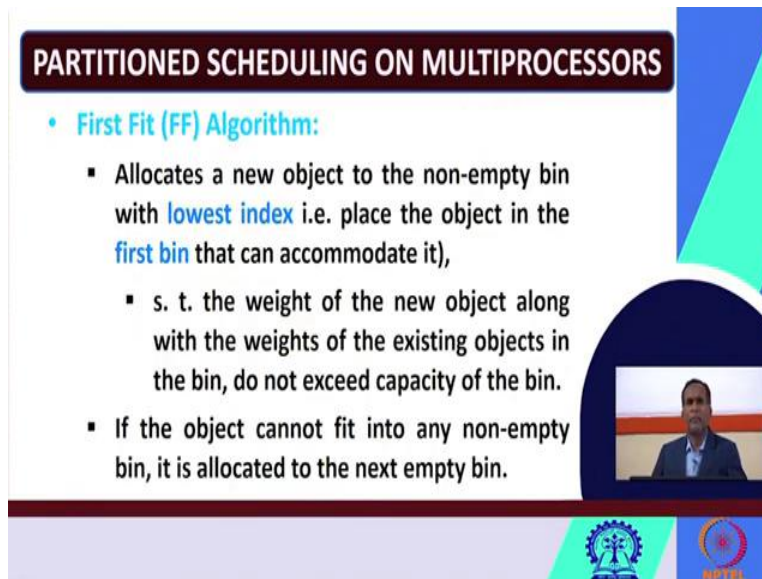
The slide features a dark blue header with the title in white. The main content is on a white background with blue and green accents on the right. A small video inset of a speaker is in the bottom right. Logos for a university and NPTEL are at the bottom.

Now, let us say the possible, some examples of this partitioned scheduling. I have already told you that this bin packing problem, bin packing algorithm that is coming under this particular

classification, particular class. The task, so in partitioned scheduling on multiprocessors, the task partitioning problem, it is very much analogous, it is very much similar to to bin packing problem.

Bin packing problem already we have discussed in the previous class. So, among the bin packing heuristics, four popular algorithms are used. Those are first fit algorithm, next fit algorithm, best fit algorithm and worst fit algorithm. In the last class we have already discussed first fit algorithm, two varieties we have seen, first fit random algorithm and first fit decreasing algorithm. So, let us quickly just look at, I have also taken some examples in the last class. The same examples you please extend to these things.

(Refer Slide Time: 0:33:23)



**PARTITIONED SCHEDULING ON MULTIPROCESSORS**

- **First Fit (FF) Algorithm:**
  - Allocates a new object to the non-empty bin with **lowest index** i.e. place the object in the **first bin** that can accommodate it),
    - s. t. the weight of the new object along with the weights of the existing objects in the bin, do not exceed capacity of the bin.
  - If the object cannot fit into any non-empty bin, it is allocated to the next empty bin.

The slide features a dark red header with the title in white. The main content is in black text with blue highlights for 'lowest index' and 'first bin'. A small video inset shows a man in a suit. At the bottom, there are two logos: a blue gear-like emblem on the left and a red circular emblem on the right.

So, let us quickly look at, we have already discussed first fit algorithm in last class. So, I will just skip it. so in this scheme or this algorithm, it allocates a new object to the non-empty bin with lowest index, that means you have to place the object in the first bin, which can accommodate it, in such a way that the weight of the new object along with the weights of the existing objects in the bin, it do not exceed the capacity of the bin.

Here capacity you can considered as the utilization. So, in this case we will try in such that the utilization should not be greater then maybe 1. If the object cannot fit into any non-empty bin, it is allocated to the next empty bin. The example already I have taken in the last class. So, please see yourself that example. That is for the first fit algorithm.

(Refer Slide Time: 0:34:10)

**PARTITIONED SCHEDULING ON MULTIPROCESSORS**

- **Next Fit (NF) Algorithm:**
  - Maintains a pointer to the “current” bin, which is the bin to which the last object was allocated.
  - A new object is allocated to the current bin if it fits into that bin,
    - Else it is allocated to the next empty bin.
  - This algorithm does not revisit the previous bins, as possible in FF algorithm.

The slide features a dark red header with the title in white. The main content is in a white box with a blue and green geometric background on the right. A small video inset shows a man in a suit. At the bottom, there are logos for a university and a 'NPTEL' logo.

Similarly, next fit algorithm, as its name suggests, next, it maintains a pointer to the current bin, so what is current bin? A current bin is the bin, to which the last object was allocated, and new object is allocated to the current bin if it fits into that bin. So, whenever any new object or a new task is arriving it can be allocated to the current bin or the current processor, if it fits into that bin or processor.

Else what will happen? Else the task is allocated to the next empty bin, the next empty processor, that is why this is known as next fit algorithm. Please remember that in first fit algorithm there is a possibility that we can revisit again, processor 1, processor 2, processor 3, if a task is coming, it having a very less utilization, again we can go to processor 1. But in next fit algorithm this next fit algorithm does not revisit the previous bins, previous processors which are possible in first fit algorithm.

(Refer Slide Time: 0:35:07)

**PARTITIONED SCHEDULING ON MULTIPROCESSORS**

- **Best fit (BF) Algorithm:**
  - Allocates the new object to the non-empty bin with the **smallest remaining capacity** into which it can fit.
  - If an object cannot be allocated to a non-empty bin, it is allocated to the next empty bin.
- **Worst fit (WF) Algorithm:**
  - Similar to BF, except that it allocates a new object to the bin with the **largest remaining capacity** into which it can fit.

**PARTITIONED SCHEDULING ON MULTIPROCESSORS**

- **Next Fit (NF) Algorithm:**
  - Maintains a pointer to the “current” bin, which is the bin to which the last object was allocated.
  - A new object is allocated to the current bin if it fits into that bin,
    - Else it is allocated to the next empty bin.
  - This algorithm does not revisit the previous bins, as possible in FF algorithm.

Next it best fit algorithm, so as its name suggest, best fit, this algorithm allocates the new object, or the new task to the non-empty bin or the non-empty processor with the smallest remaining capacity into which it can fit. So, supposed there are two bins, two processors, one is the remaining capacity, 0.4, another is remaining capacity 0.3 and the task it is arriving it is having a utilization of 0.2, so obviously it will assign to whom? The processor having the remaining capacity 0.3, because this is the smallest one, in between 0.4 and 0.3 utilization. So, in this case the task will be allocated to the non-empty bin having the smallest remaining capacity. If an object cannot be allocated to a non-empty bin, what will happen, it is allocated to the next empty bin. Then there is

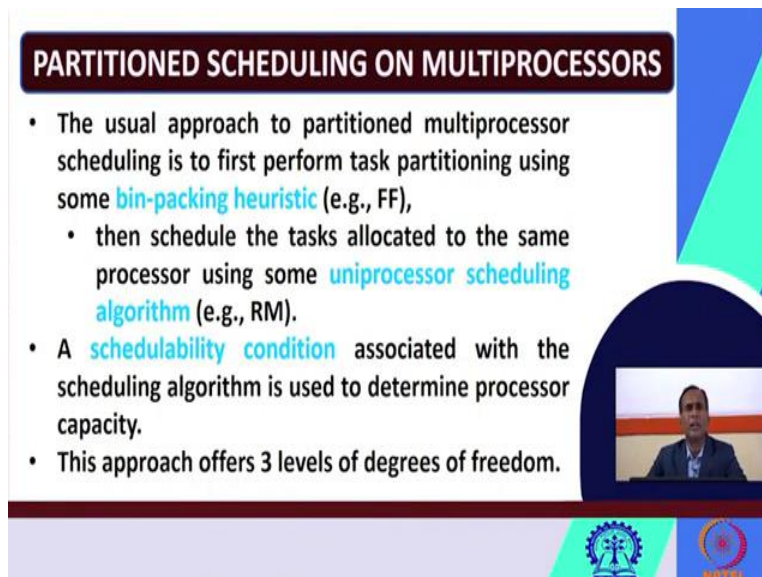
worst fit algorithm. It is very much similar to the best fit algorithm, only one difference is there, it allocates a new object or the new task to the bin or to the processor with the largest remaining capacity to which it can fit in.

So, here we are considering smallest remaining capacity, so the example I have already given, two bins are there, suppose in one the remaining capacity or remaining utilization is 0.4, another is 0.3, the task it is arriving, its utilization is 0.2, it will be fitted to where? To the second one, because it is having 0.3 utilization, it is less.

But had it been used worst fit algorithm, it will be allocated to this bin having the largest remaining capacity. That means out of 0.4 and 0.3, if the task requires, 0.2, it is having 0.2 utilization, it will be allocated to the first processor which is having utilization, remaining utilization 0.4, because this 0.4 is larger than 0.3.

So, this is the only difference, in this case the task will be assigned to the smallest bin having, to the bin having smallest remaining capacity but in worst fit algorithm the task will be allocated to the bin having the largest remaining capacity. This is different. So, last class I have taken one example for first fit algorithm. Please exercise yourselves and take that same example, apply next algorithm, best fit algorithm and worst fit algorithm. This is the assignment for you.

(Refer Slide Time: 0:37:26)



**PARTITIONED SCHEDULING ON MULTIPROCESSORS**

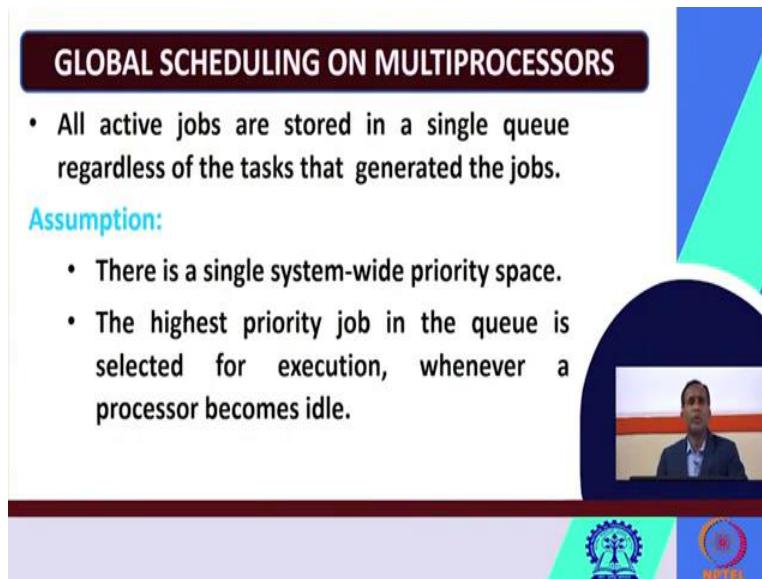
- The usual approach to partitioned multiprocessor scheduling is to first perform task partitioning using some **bin-packing heuristic** (e.g., FF),
  - then schedule the tasks allocated to the same processor using some **uniprocessor scheduling algorithm** (e.g., RM).
- A **schedulability condition** associated with the scheduling algorithm is used to determine processor capacity.
- This approach offers 3 levels of degrees of freedom.

The slide features a video inset of a man in a suit speaking. At the bottom, there are two logos: a tree logo on the left and a circular logo with 'MPT' on the right.

Let us see the common approach for this partitioned multiprocessor scheduling, the common approach to the partitioned multiprocessor scheduling is, that first you have to perform the task partitioning using some bin packing heuristic, maybe first fit or next fit, et cetera. Then you schedule the task allocated to the same processor, using some uni processor scheduling algorithm, maybe RMA or EDF.

Then a scheduling condition associated with the scheduling algorithm it is used to determined the processor capacity. You can see that this approach offers three levels of degrees of freedom, first using bin packing heuristic, second using a uni processor scheduling algorithm like RMA or EDF, third a schedule-ability condition associated with the scheduling algorithm. So, this approach offers three levels of degrees of freedom, to design a partitioned scheduling on multiprocessor systems.

(Refer Slide Time: 0:38:18)



**GLOBAL SCHEDULING ON MULTIPROCESSORS**

- All active jobs are stored in a single queue regardless of the tasks that generated the jobs.

**Assumption:**

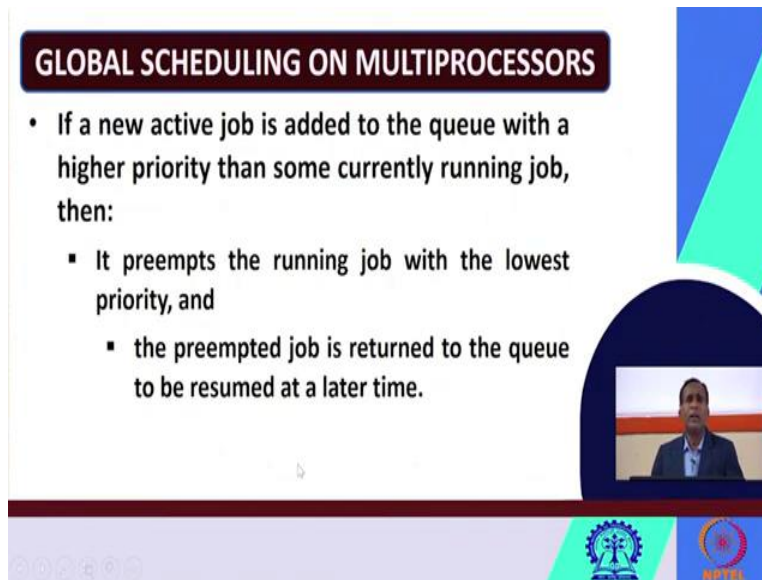
- There is a single system-wide priority space.
- The highest priority job in the queue is selected for execution, whenever a processor becomes idle.

The slide features a dark red header with the title in white. The main content is on a white background with a blue and green geometric design on the right. A small video inset shows a man in a suit. At the bottom, there are logos for a university and NPTL.

We will quickly look at this global scheduling on multiprocessors. I have already told you the difference between partition scheduling and global scheduling. Here, all active jobs, they are stored in a single queue, regardless of tasks, that generated the jobs. The assumption here is taken like this. there is a single system wide priority space and the highest priority job in the queue is selected for execution whenever a processor becomes idle. So, whenever a processor, it becomes idle, the highest priority job in the queue is selected for execution and it is allocated to that idle processor.



(Refer Slide Time: 0:38:57)

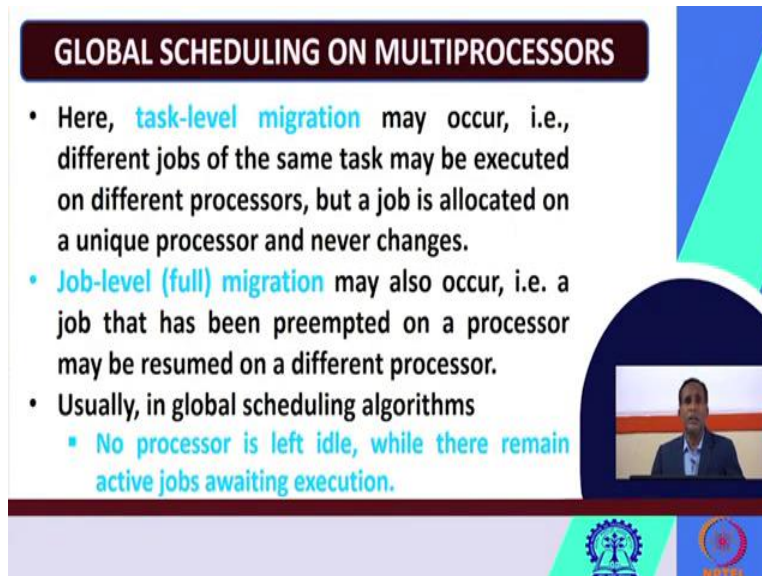


**GLOBAL SCHEDULING ON MULTIPROCESSORS**

- If a new active job is added to the queue with a higher priority than some currently running job, then:
  - It preempts the running job with the lowest priority, and
    - the preempted job is returned to the queue to be resumed at a later time.

So, if a new active job is added to the queue then what will happen? If a new active job is added to the queue with a higher priority than some currently running job, then what will happen? It will preempt the running job with the lowest priority and the preempted job is returned to where? The preempted job is returned to the queue, which has to be resumed at a later point of time.

(Refer Slide Time: 0:39:22)



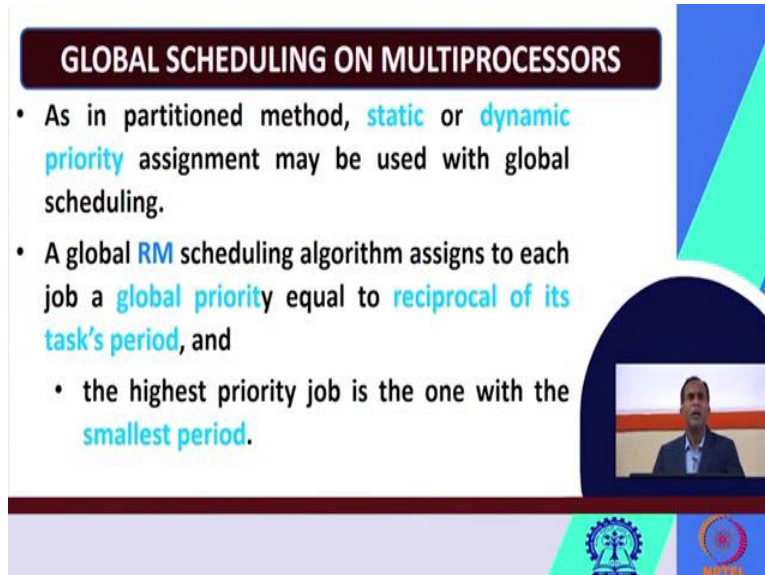
**GLOBAL SCHEDULING ON MULTIPROCESSORS**

- Here, **task-level migration** may occur, i.e., different jobs of the same task may be executed on different processors, but a job is allocated on a unique processor and never changes.
- **Job-level (full) migration** may also occur, i.e. a job that has been preempted on a processor may be resumed on a different processor.
- Usually, in global scheduling algorithms
  - **No processor is left idle, while there remain active jobs awaiting execution.**

I have already told you, in global scheduling task level migration may occur as well as job level migration may occur, this I have already explained earlier, job level migration is also known as

full migration. Usually in the global scheduling algorithms no processor is left idle while the remaining, the active jobs awaiting for execution, this is possible in global scheduling algorithms.

(Refer Slide Time: 0:39:47)



**GLOBAL SCHEDULING ON MULTIPROCESSORS**

- As in partitioned method, **static** or **dynamic priority** assignment may be used with global scheduling.
- A global **RM** scheduling algorithm assigns to each job a **global priority** equal to **reciprocal of its task's period**, and
  - the highest priority job is the one with the **smallest period**.

The slide features a dark blue header with the title in white. The main content is on a white background with blue and green accents. A small video inset shows a man in a suit. Logos for institutions are visible at the bottom.

As in partitioned method, here also we can use static priority assignment or dynamic priority assignment, as in partitioned method static or dynamic priority assignment may be used with the global scheduling approach. So, what is the example of static priority assignment? That is RMA, example of dynamic priority assignment is EDF, now let us see, a global RM, that is rate monitoring scheduling algorithm, it assigns to each job a global priority.

Its name is global scheduling algorithm, obviously it will assign a global priority, how? A global rate monitoring scheduling algorithm, assigns to each job a global priority equal to what? Equal to the reciprocal of its task period. That means if the period is  $P$ , reciprocal mean  $1$  by  $P$ , and the higher priority job is the one with the smallest period. So, which one will be the highest priority job, we have already known RMA in the previous classes. The highest priority job is the one with the smallest period.

(Refer Slide Time: 0:40:47)

**GLOBAL SCHEDULING MULTIPROCESSORS**

- A global **EDF** scheduling algorithm assigns to each job a **global priority** equal to **its absolute deadline**, and
  - the highest priority job is the one with the **earliest absolute deadline**.
- All jobs are considered for execution in accordance with their **global priorities**.

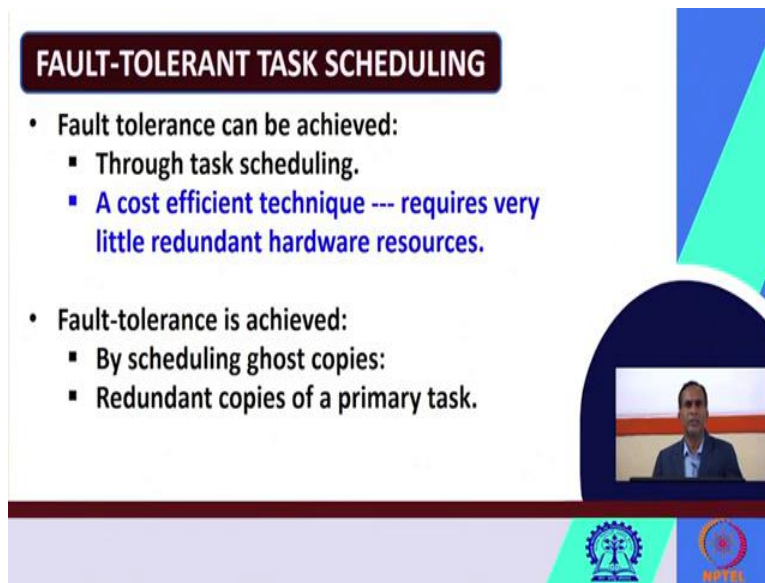
**GLOBAL SCHEDULING ON MULTIPROCESSORS**

- As in partitioned method, **static** or **dynamic priority** assignment may be used with global scheduling.
- A global **RM** scheduling algorithm assigns to each job a **global priority** equal to **reciprocal of its task's period**, and
  - the highest priority job is the one with the **smallest period**.

Similarly, what will happen for EDF? A global EDF scheduling algorithm, it assigns to each job a global priority like RMA, a global EDF scheduling algorithm also assigns to each job a global priority, equal to what? So, in case of RMA, the global priority is equal to the reciprocal of its, or inverse of its period, inverse of its task period. But in a global EDF scheduling algorithm it assigns to each job a global priority, which is equal to what? Which is equal to its absolute deadline, and what is the highest priorities job? The highest priority job is the one with the earlier absolute deadline.

I hope you have already read earlier the relative deadline, absolute deadline, you have read earlier. Then all jobs are considered for execution in accordance with their global priorities. Then all the jobs they are considered for execution. They will be executed in accordance with their global priority. So, this is something about the global scheduling multiprocessors.

(Refer Slide Time: 0:41:45)



**FAULT-TOLERANT TASK SCHEDULING**

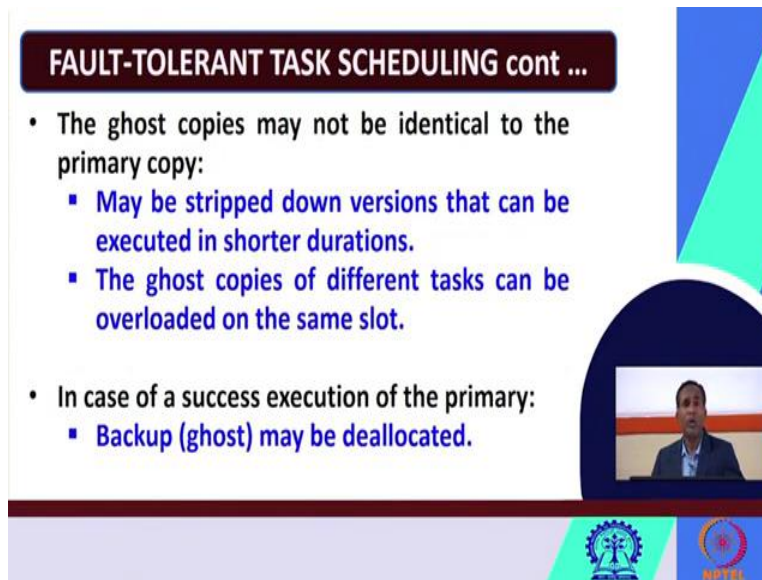
- Fault tolerance can be achieved:
  - Through task scheduling.
  - A cost efficient technique --- requires very little redundant hardware resources.
- Fault-tolerance is achieved:
  - By scheduling ghost copies:
  - Redundant copies of a primary task.

The slide features a dark blue header with the title in white. The main content is on a white background with blue and green geometric shapes on the right. A small video inset shows a man in a suit. At the bottom, there are logos for a university and a research center.

Just few minutes you will take on this fault tolerant task scheduling, how your real time systems can be made fault tolerant. Fault tolerance in real time systems can be achieved through task scheduling. This is a cost-efficient technique. This requires very little redundant hardware resources, why this is cost efficient technique? This is a cost-efficient technique because it requires very less, very little redundant hardware resources.

Now, let us see how fault tolerance can be achieved in real time systems. The fault tolerance can be achieved in real time systems, by scheduling ghost copies, what do you mean by ghost copy? Ghost copy means redundant copies of a primary task. So, by scheduling the ghost copies or redundant copies of a primary task fault tolerance can be achieved in real time systems.

(Refer Slide Time: 0:42:37)



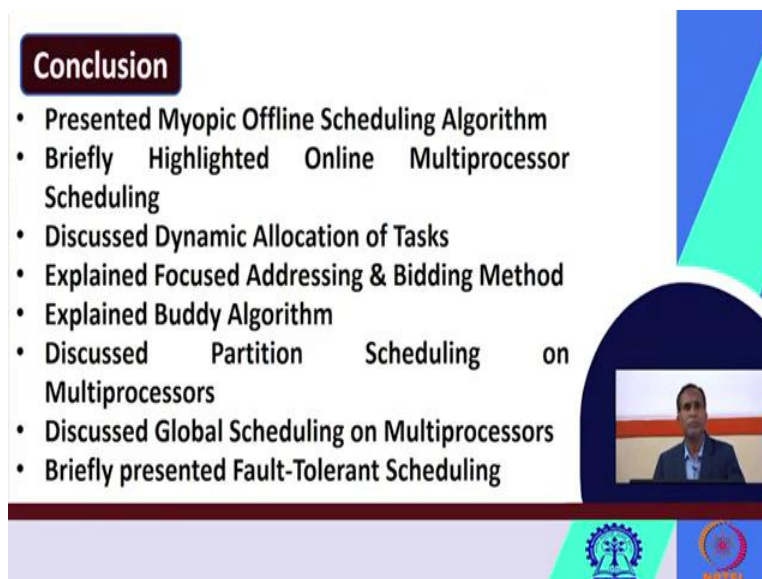
**FAULT-TOLERANT TASK SCHEDULING cont ...**

- The ghost copies may not be identical to the primary copy:
  - May be stripped down versions that can be executed in shorter durations.
  - The ghost copies of different tasks can be overloaded on the same slot.
- In case of a success execution of the primary:
  - Backup (ghost) may be deallocated.

The slide features a dark red header with the title in white. The main content is a list of bullet points in black and blue. A small video inset on the right shows a man in a suit. At the bottom, there are two logos: a blue gear-like logo on the left and a red circular logo with 'NPTEL' on the right.

These ghost copies may not be exactly identical to the primary copy, the ghost copies may be stripped down versions so that can be executed in shorter durations. The ghost copies of different tasks can be over loaded on the same slot. So, if there are multiple numbers of ghost copies, n number of ghost copies they can be over loaded in the same slot. In case of a success execution of the primary, the backup copy or the ghost copy may be deallocated, so if the primary copy is successfully executed then the backup copy or the ghost copy may be deallocated.

(Refer Slide Time: 0:43:08)



**Conclusion**

- Presented Myopic Offline Scheduling Algorithm
- Briefly Highlighted Online Multiprocessor Scheduling
- Discussed Dynamic Allocation of Tasks
- Explained Focused Addressing & Bidding Method
- Explained Buddy Algorithm
- Discussed Partition Scheduling on Multiprocessors
- Discussed Global Scheduling on Multiprocessors
- Briefly presented Fault-Tolerant Scheduling

The slide features a dark red header with the title in white. The main content is a list of bullet points in black. A small video inset on the right shows a man in a suit. At the bottom, there are two logos: a blue gear-like logo on the left and a red circular logo with 'NPTEL' on the right.

So, today we have discussed, first, the myopic offline scheduling algorithm, then we have briefly highlighted the online multiprocessor scheduling, we have discussed the fundamentals of dynamic allocation of task and how do they differ from static allocation of tasks. We have explained the two dynamic allocation algorithms, such as focus addressing and bidding and buddy algorithm. We have also discussed partition scheduling on multiprocessors and global scheduling on multiprocessors. Finally, we have briefly presented something about or the basic concepts of fault tolerant scheduling techniques.

(Refer Slide Time: 0:43:42)



**REFERENCES**

1. Rajib Mall, Real-Time Systems: Theory and Practice, 1st Edition, 2007, Pearson Education
2. C. M. Krishna & K. G. Shin, Real-Time Systems, 2017, Tata McGraw Hill Education

The slide includes a video inset of a man in a suit speaking, and logos for IIT Bombay and NPTEL at the bottom.

We have taken these things from these two books, thank you very much.