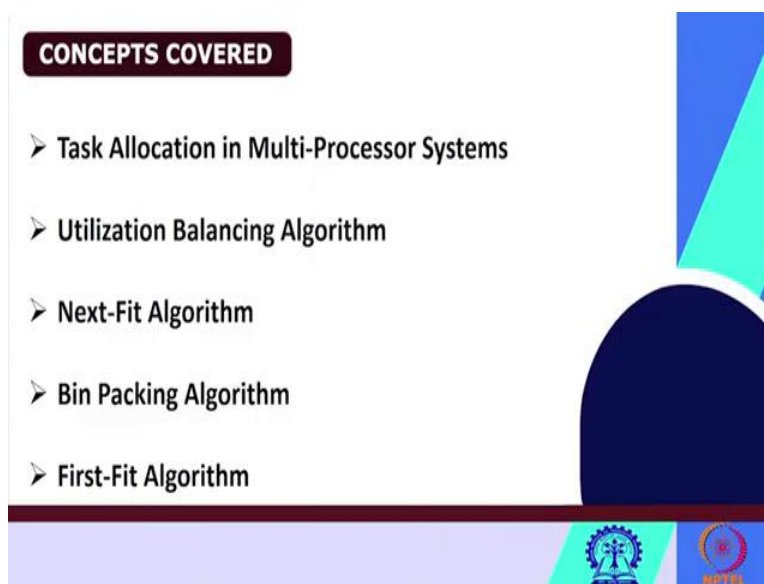


**Real Time Systems**  
**Professor Durga Prasad Mohapatra**  
**Department of Computer Science and Engineering**  
**National Institute of Technology, Rourkela**  
**Lecture 33**  
**Static Allocation of Tasks**

Good afternoon to all of you. Now, today we will discuss about this static allocation of tasks. Last class we have discussed about the different classifications of allocation of tasks. One is static, another is dynamic. So, today we will mainly focus on the static allocation of these tasks.

(Refer Slide Time: 0:00:39)



So, basically today we will cover how the task allocation can be made, can be done in multiprocessor systems. Different static allocation techniques, such as utilization balancing algorithm, next fit algorithm, bin packing algorithm you will see, and under bin packing algorithm we will see a special algorithm called a first fit algorithm.

(Refer Slide Time: 0:01:01)

**KEYWORDS**

- RMA
- EDF
- Utilization Balancing
- Next-Fit Algorithm
- Bin-Packing Algorithm

The slide features a dark blue 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 two logos: a circular one on the left and a rectangular one on the right.

So, today the main keywords that will be used are RMA that is rate monotonic algorithm, EDF, earliest deadline fast, utilization balancing algorithm, next fit algorithm, bin packing algorithm, et cetera.

(Refer Slide Time: 0:01:14)

**TASK ALLOCATION IN MULTI-PROCESSOR SYSTEMS**

- **Static Priority Scheduling:**
  - Allocation is made before run-time, i.e., the tasks are pre-allocated to processors.
  - Allocation remains valid throughout full run of the system.
  - Each task is assigned a fixed priority that does not change during its lifetime.
  - Allocation algorithms are centralized.
  - **No overhead during run time since tasks are permanently assigned to processors.**

The slide has a dark blue header with the title 'TASK ALLOCATION IN MULTI-PROCESSOR SYSTEMS' in white. Below the header, there is a blue bullet point followed by the text 'Static Priority Scheduling:'. Underneath this, there are five sub-bullets, each preceded by a square bullet point. The last sub-bullet is highlighted in green. The background and footer are identical to the previous slide.

So, let us see about the first the difference between static priority or static allocation of static priority scheduling and dynamic priority scheduling then we will take up the individual static priority scheduling algorithms. So, in static allocation, how the allocation is made? The allocation

is made before the run time. So, before the run time the allocation is made that is what happens that the tasks are pre-allocated to the processors.

We know the details of the tasks, we know the details of the processors in advance, beforehand and accordingly the tasks are pre-allocated, they are allocated before the run time to the different processors, and this allocation it remains valid, throughout the full run of the system. It will not change. The allocation remains valid, that means which tasks is assigned to which processor. This allocation remains valid throughout the full run of the system.

So, here each task is assigned a fixed priority. So, every task, it is assigned a fix priority and this priority does not change during its lifetime. So, normally these static allocation algorithms, they are centralized, we are using a centralized algorithm here, so basically the static algorithm, the static allocation algorithms are centralized in nature.

So, one of the advantage of the static priority scheduling is that, there is no overhead, why? So, there are no overheads during run time as the tasks are permanently assigned to the processors. I have already told you here, the tasks are pre-allocated and this allocation, it remains valid, if this allocation does not change, these fixed priorities, it does not change during its lifetime. So, since the tasks are permanently assigned to the processors so there is no extra overhead during the run time. This is one of the advantages of static priority scheduling.

(Refer Slide Time: 0:03:11)

**TASK ALLOCATION IN MULTI-PROCESSOR SYSTEMS**

- **Examples:**
  - **Utilization Balancing Algorithm**
  - **Next-fit algorithm for Rate Monotonic Algorithm (RMA)**
  - **Bin packing algorithm for Earliest Deadline First (EDF)**

The slide features a video inset of a speaker in the bottom right corner and two logos at the bottom center.

Now, which algorithms are coming under static priority or static allocation algorithms? The following algorithms will come under static allocation, like utilization balancing algorithm, next fit algorithm for RMA, bin packing algorithm for EDF. So, today we will discuss these three algorithms, but before that let us quickly see about the drawbacks or let us say the difference between static algorithm or static allocation algorithm and the dynamic allocation algorithm, then we will come back to these three static allocation algorithms.

(Refer Slide Time: 0:03:47)

**MULTIPROCESSOR TASK ALLOCATION (cont ...)**

- **Dynamic Priority Scheduling:**
  - In many applications tasks arrive sporadically at different nodes, so there is a need of dynamic scheduling algorithms.
  - The tasks are assigned to processors as and when they arrive.
  - The priority assigned to a task may change during its lifetime.
  - Task allocation to nodes is made based on instantaneous load position of other nodes.

So, let us compare how this dynamic priority scheduling it is different than the static priority scheduling or static allocation. You know that what problem we will suffer in case of static priority scheduling, normally we have to know in advance what are the tasks. The task details we have to know, but in several applications these tasks arrive sporadically at different nodes. Simultaneously we will see different tasks they are arriving.

The tasks arrive sporadically at different nodes and hence the static algorithms they will not be fruitful, they will not be suitable for these types of tasks, so the need of having some dynamic scheduling algorithms. In case of dynamic priority scheduling or in case of the dynamic allocation how the tasks are assigned? The tasks are assigned to the processors as and when they arrive. So, as and when the task arrives, then each task will be assigned to the processors accordingly.

So, here in dynamic allocation the tasks are assigned to the processors as an when they arrive, and this priority that is assigned to any task it may change during its lifetime. The priority assigned to

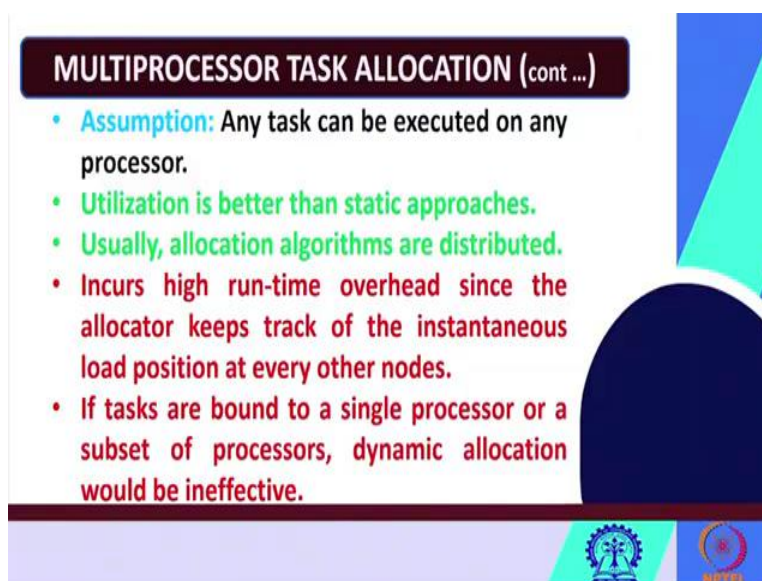
a task during this dynamic priority scheduling, it may change during its lifetime, but in static allocation what we have seen? Once the priority is assigned to a task that cannot change. That will remain throughout the lifetime.

But in dynamic priority scheduling, the priority, once assigned to a task, it may change during its lifetime, this is one of the differences between the static allocation algorithms. Here the task allocation to the different nodes is made based on what? Based on the instantaneous load position of other nodes.

So, other nodes you can consider was other computers or other processors. The nodes may be considered as the other processors. So, the task allocation to the nodes, how it will be met. The task allocation to the nodes is made on, is made based on the instantaneous load position of other loads, that means each node, it has to know what is the node position of the other nodes.

Based on that the tasks will be assigned. So, here we will take examples, here the task allocation will be made based on the instantaneous load position of other nodes, what is the load position at other processors, at other nodes, based on that the task allocation to the current node will be met.

(Refer Slide Time: 0:06:21)



**MULTIPROCESSOR TASK ALLOCATION (cont ...)**

- **Assumption:** Any task can be executed on any processor.
- **Utilization is better than static approaches.**
- **Usually, allocation algorithms are distributed.**
- **Incurs high run-time overhead since the allocator keeps track of the instantaneous load position at every other nodes.**
- **If tasks are bound to a single processor or a subset of processors, dynamic allocation would be ineffective.**

The slide features a dark blue header with white text, a list of five bullet points with varying colors (blue, green, red), and a footer with two logos: a tree logo on the left and a circular logo on the right.

## MULTIPROCESSOR TASK ALLOCATION (cont ...)

- **Dynamic Priority Scheduling:**
  - In many applications tasks arrive sporadically at different nodes, so there is a need of dynamic scheduling algorithms.
  - The tasks are assigned to processors as and when they arrive.
  - The priority assigned to a task may change during its lifetime.
  - Task allocation to nodes is made based on instantaneous load position of other nodes.



So, in this dynamic scheduling or this dynamic allocation algorithm one assumption we are taking that any task can be executed on any processor. There is no restriction. Any task, it can be executed in any processor. Now, let us say the advantages of this dynamic scheduling. So, advantage is that the utilization is better than these static approaches. Why?

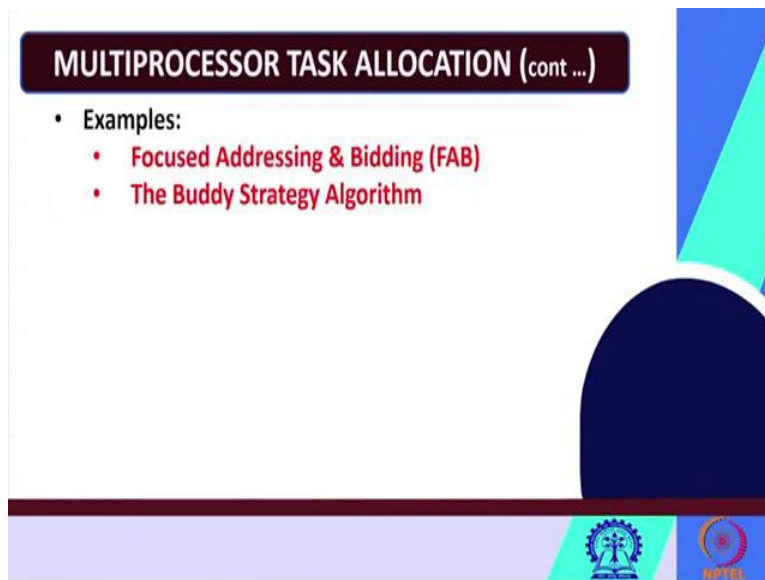
Because the task allocation is made based on the instantaneous load position of the other nodes. So, since every node, what is the instantaneous node position of other nodes, so it is expected that the utilization in case of the dynamic algorithms, that will be better than the static approaches and also usually the allocation, this dynamic allocation algorithms are distributed in nature.

We have seen in case of static allocation, the static allocation algorithms are centralized in nature, whereas usually, most of the times many of the dynamic allocation algorithms, they are distributed in nature. Let us go to the drawbacks of dynamic scheduling. So, these dynamic allocation algorithms they incur high run time over it, why?

Because the allocator, it has to keep track of the instantaneous load position at every other node. So, since the allocator, it has to keep the track of the load position at other nodes, before making, before allocating a task to a particular node, the allocator has to know what is the load position, what is the instantaneous load position at the other nodes. Then only it will, whether it will take a decision whether the task will be allocated to the current node or not.

So, due to this keeping of the track of the instantaneous load position at every other node, so dynamic allocation algorithms they incur high run time overhead. So, another disadvantage is that if the tasks are bound to a single processor. So, what tasks they will be assigned, or the tasks those will be allocated. If those tasks are bound to a single processor or a subset of processors, then what will happen? The dynamic allocation would be ineffective. Here the dynamic allocation will not be so much effective, rather we should go for may be the static allocation.

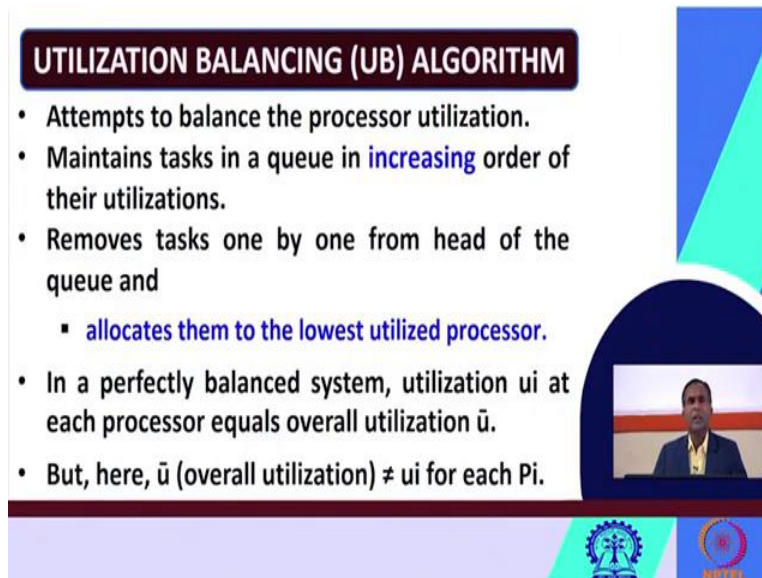
(Refer Slide Time: 0:08:36)



So, what algorithms are coming under this dynamic allocation? So, two important allocation we will discuss under dynamic allocation. One the focus addressing and bidding algorithm, another is the buddy strategy algorithm, or simply we can say that buddy algorithm. So, these two things we will see under this one.



(Refer Slide Time: 0:08:55)



**UTILIZATION BALANCING (UB) ALGORITHM**

- Attempts to balance the processor utilization.
- Maintains tasks in a queue in **increasing** order of their utilizations.
- Removes tasks one by one from head of the queue and
  - **allocates them to the lowest utilized processor.**
- In a perfectly balanced system, utilization  $u_i$  at each processor equals overall utilization  $\bar{u}$ .
- But, here,  $\bar{u}$  (overall utilization)  $\neq u_i$  for each  $P_i$ .

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

Now, let us see first about this utilization balancing algorithm. Now, I am migrating or moving to the first discussing the static allocation algorithms. So, the first static allocation algorithm, the simplest static allocation algorithm is the utilization balancing algorithm. So, here these algorithms they attempt to balance the processor utilization.

So, in these algorithms they try to allocate the tasks in such a way that the processor utilization will be balanced. So, these algorithms they attempt to balance the processor utilization that means the processor utilization at the different nodes, it will be balanced. So, not that one processor will get very highly utilization and another processor will be very negligible utilization. These things should not occur.

So, all the processors, they will, their utilization will be balanced. So, these algorithms attempt to balance the processor utilization, then these algorithms what do they do, they maintain the task in the queue. So, how the task allocation is made in utilization balancing algorithm? First step, so these algorithms, they maintain a queue. They maintain the task in the queue, in which order? In increasing order of their utilization.

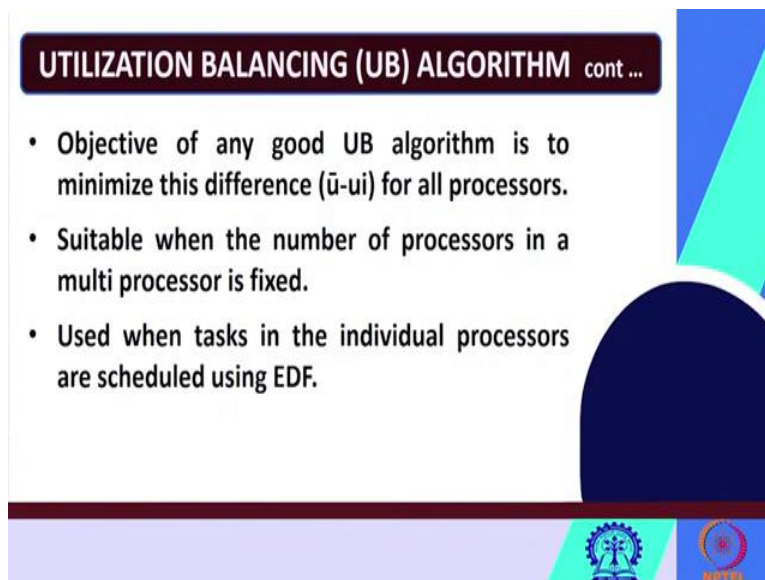
So, first this algorithm, it what puts all these tasks in a queue, in the increasing order of their utilizations, then what it does, it gradually, it removes the task one by one. In the second step this utilization balancing algorithm, it removes the task one by one from where? From the head of the queue and then it allocates. How it allocates?



It allocates these tasks to what? To the lowest utilized processor. So, each time it will remove one task from the head of the queue, then it will try to find out which processor is having lowest utilization, then this algorithm, it will allocate, this task to the lowest utilized processor with this very simple algorithm. So, always this algorithm will try to extract, to remove the tasks one by one from the head and then it will find out which processor is lowest utilized and it will allocate the task to that lowest utilized processor.

But if you will see at a perfectly balanced system, in case of a perfectly balanced system the utilization  $u_i$  at a particular processor, normally it should be equal to the overall utilization or the average utilization which you denote as  $\bar{u}$ , but usually you will see, this is very much difficult to make equal this overall utilization, and with this utilization  $u_i$  at a particular processor. So, in this, usually in these utilization balancing algorithms, these overall utilizations may not be equal to the utilization, the individual utilization or the utilization  $u_i$  are the easy processor. It is very much difficult to get it. it is very much difficult to make the overall utilization equal with the utilization for each of these processors.

(Refer Slide Time: 0:12:15)



**UTILIZATION BALANCING (UB) ALGORITHM cont ...**

- Objective of any good UB algorithm is to minimize this difference ( $\bar{u}-u_i$ ) for all processors.
- Suitable when the number of processors in a multi processor is fixed.
- Used when tasks in the individual processors are scheduled using EDF.

The slide features a decorative background with a blue and green geometric shape on the right side and a dark blue semi-circle at the bottom right. At the bottom, there are two logos: a circular logo on the left and a rectangular logo on the right.

## UTILIZATION BALANCING (UB) ALGORITHM

- Attempts to balance the processor utilization.
- Maintains tasks in a queue in **increasing** order of their utilizations.
- Removes tasks one by one from head of the queue and
  - **allocates them to the lowest utilized processor.**
- In a perfectly balanced system, utilization  $u_i$  at each processor equals overall utilization  $\bar{u}$ .
- But, here,  $\bar{u}$  (overall utilization)  $\neq u_i$  for each  $P_i$ .



So, what should be objective of any good utilization balancing algorithm? The objective of any good utilization balancing algorithm, should be, how to minimize these differences between this overall utilization and the utilization at an individual processor  $i$ , so then you can take the summation because we have to consider all the processors and we have to minimize this.

Because as I have already told you that normally that normally this  $\bar{u} \neq u_i$  for each processor. So, that is why what our objective should be? Our objective should be to minimize this difference. So, the objective of any good utilization balancing is to minimize this difference  $\bar{u} - u_i$  for all processors.

Then you will take the summations, suppose there are  $n$  processors so take this value, like

$$\sum_{n=1}^n (\bar{u} - u_i)$$

and your objective should be to minimize this value. So, let us say when utilization balancing algorithm will be suitable. So, utilization balancing algorithm is suitable when the number of processors in a multiprocessor is fixed. So, when the number of processors, we know in advance, and the number of processors in a multiprocessor is fixed, say  $n = 10$ , there are only 10 number of processors. So, in those case utilization balancing algorithm is very much suitable. Similarly, when it is used, so this utilization balancing algorithm is used when the tasks in the individual processors are scheduled using EDF. So, when the tasks in the individual processors they can be scheduled or they are scheduled using this EDF, earliest deadline first, in those case utilization balancing

algorithm is normally used. So, this is about utilization balancing algorithm. The simplest static allocation algorithm.

(Refer Slide Time: 0:14:08)

**NEXT FIT ALGORITHM FOR RMA**

- There is a utilization based allocation heuristics which is can be used in conjunction with RMA.
- A task set is partitioned so that each partition is scheduled on a uniprocessor using RMA.
- Attempts to use as few processors as possible.
- Unlike UB algorithm, it does not require the number of processors of the system to be predetermined and given before hand.
- It classifies the different tasks into a few classes based on the utilization of the task.

**UTILIZATION BALANCING (UB) ALGORITHM cont ...**

- Objective of any good UB algorithm is to minimize this difference ( $\bar{u}-u_i$ ) for all processors.
- Suitable when the number of processors in a multi processor is fixed.
- Used when tasks in the individual processors are scheduled using EDF.

Now, we will go to the next one. The next one is this next fit algorithm. And normally that is used for RMA. We have already seen that utilization balancing algorithm is used when the individual processors, when the tasks in individual processors are scheduled using EDF, but when we will see another next algorithm, this next fit algorithm. That is normally used for RMA.

Let us first say that already we have seen this utilization balancing algorithm. So, there is a utilization based allocation heuristics which can be used in conjunction with RMA. So, because this utilization balancing we have seen. Normally this is used when the tasks in individual processors they are scheduled using EDF.

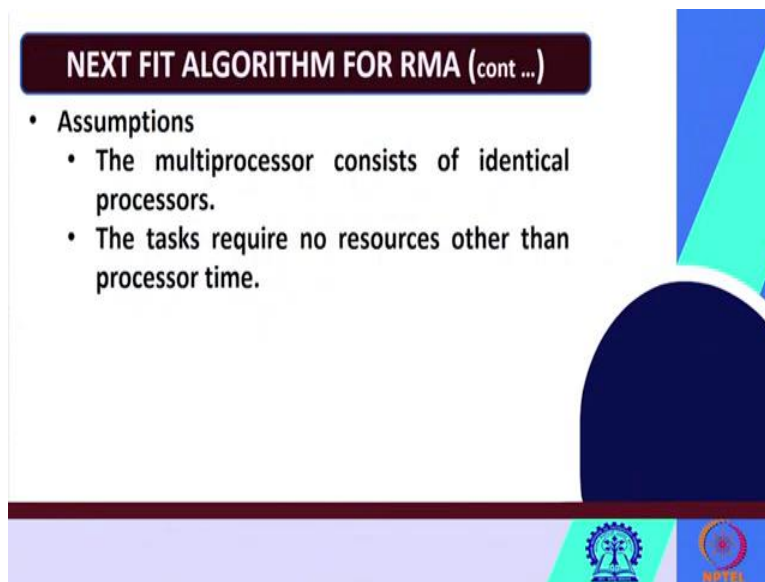
Similarly, there is also a utilization based allocation heuristics, which can be used in conjunction with RMA and in this next fit algorithm how the tasks are allocated, let us say, so first a task set is partitioned, a task set is partitioned in to a number of classes. So, that each partition is scheduled on a unique processor using RMA.

So, first a task set is partitioned into a number of classes so that each partition. It can be scheduled on a uni processor using RMA and this algorithm, next fit algorithm it tries to attempt to use as few processors as possible. So, next fit algorithm it attempts to use as less processors as possible. So, let us see one of the difference between the utilization balancing that we have discussed now on the next fit algorithm.

Unlike utilization balancing algorithm, this next fit algorithm it does not require the number of processors of the system to be predetermined and given beforehand. So, we have seen in case of utilization balancing algorithm, we have to know the number of processors in advance, because the number of processor is fixed but in next fit algorithm, this algorithm does not require the number of processors of the system to be predetermined, to be known and it should be given beforehand that this requirement is not there in case of next fit algorithm.

So, let us see how does this algorithm works. So, first I have already told you, this algorithm classifies the different task or a given task set into a few classes, based on what? Based on the utilization of the task, this next fit algorithm, it categorizes the different task into a few classes. Based on what? Based on the utilization of the tasks.

(Refer Slide Time: 0:16:42)



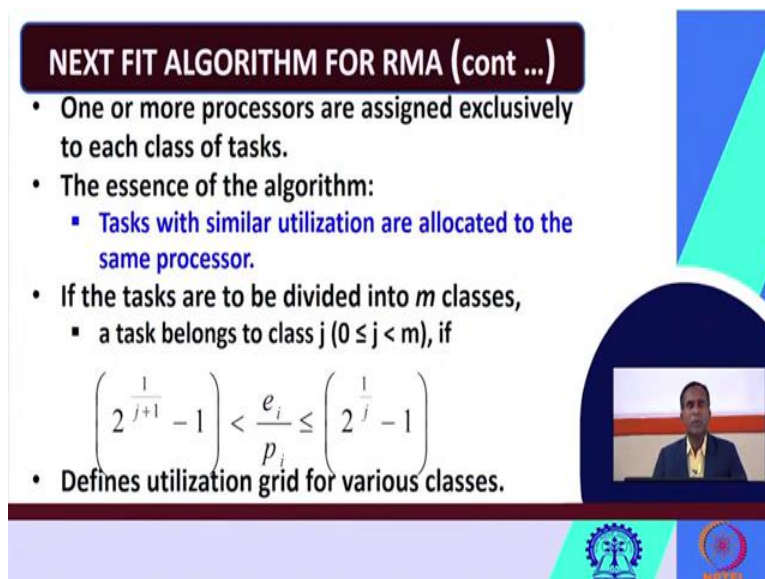
**NEXT FIT ALGORITHM FOR RMA (cont ...)**

- Assumptions
  - The multiprocessor consists of identical processors.
  - The tasks require no resources other than processor time.

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. At the bottom, there are logos for a university and NPTEL.

So, some of the assumptions that will be used in next fit algorithm are as follows, like the multiprocessor consist of identical processor. So, the multiprocessor system we are discussing or that will be in use. The multiprocessor system, it will consist of identical processors and the tasks, they will be allocated, these tasks require no resources other than processor time. So, the tasks will require only processor's time as the resources. No other resources will be required by the tasks. These are the two assumptions we will use for the nest fit algorithm.

(Refer Slide Time: 0:17:13)



**NEXT FIT ALGORITHM FOR RMA (cont ...)**

- One or more processors are assigned exclusively to each class of tasks.
- The essence of the algorithm:
  - Tasks with similar utilization are allocated to the same processor.
- If the tasks are to be divided into  $m$  classes,
  - a task belongs to class  $j$  ( $0 \leq j < m$ ), if

$$\left(2^{\frac{1}{j+1}} - 1\right) < \frac{e_i}{p_i} \leq \left(2^{\frac{1}{j}} - 1\right)$$

- Defines utilization grid for various classes.

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 speaking. At the bottom, there are logos for a university and NPTEL.

Now, let us see in detail how does this work. One or more processors are assigned exclusively to each class of task. So, here in next fit algorithm, after classifying the tasks into a number of classes, what we will do? We will do the followings. One or more processors, they are assigned exclusively to each class of the task. So, after categorizing or after classifying the task set in to a number of classes, then we have to assign one or more processors to each of the class of the task.

Now, what is the essence of this algorithm? The essence of the algorithm is that the task with similar utilization, they are allocated to the same processor. The objective of the algorithm, the essence of the algorithm is that the task having similar utilization, they are allocated towards, they are allocated to the same processor. Now, let us see how this allocation will be met.

Now, if the tasks are to be divided into m classes, suppose we want to divide the task set into m number of classes, then how to decide a task will belong to which class? A task belongs to class j, a task will belong to class j if and only if the following formula holds good. What is the formula?

$$2^{\left(\frac{1}{j+1}\right)} < \frac{e_i}{p_i} \leq (2^{\frac{1}{j}} - 1)$$

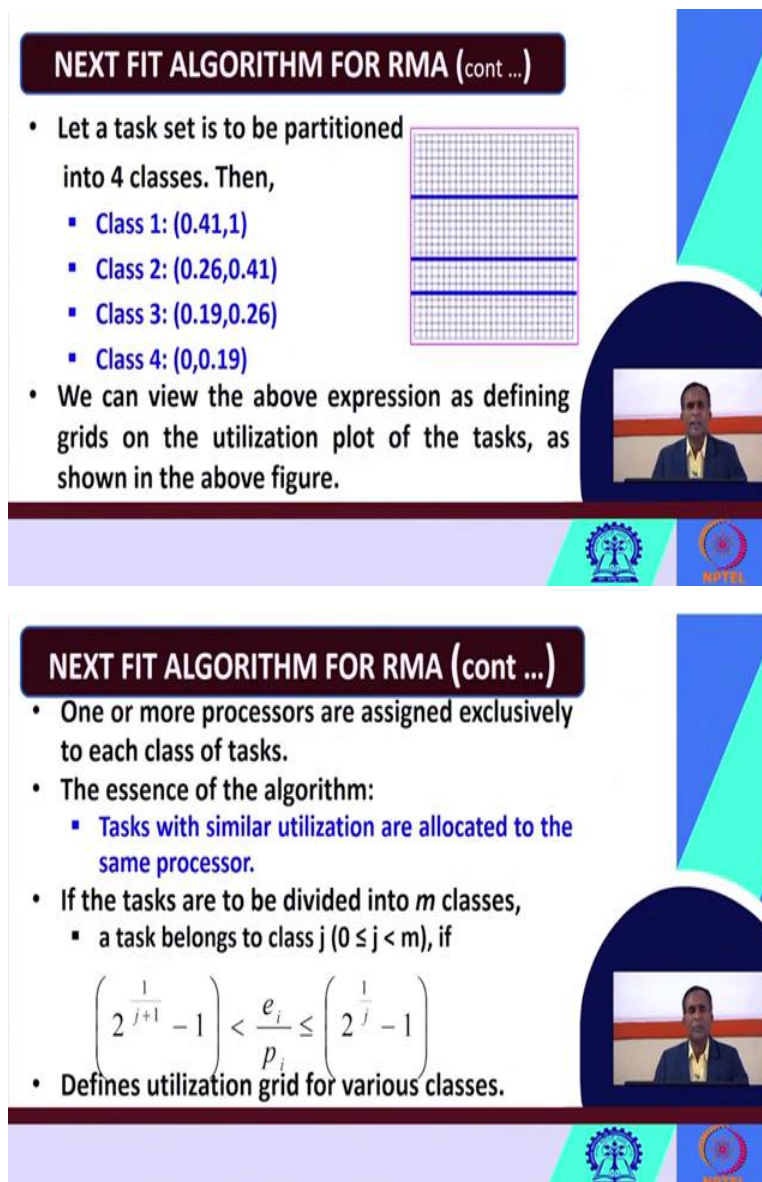
So, if we have, you want to divide the given task set into m classes then the task may belong to class j, if and only if the following equation holds good. I have already told you the classification of the task will be made based on the utilization.

And here you can see, here utilization they are  $e_i/p_i$  is nothing but the  $u_i$ , you have already known, utilization is equal to  $e_i/p_i$  where  $e_i$  is the execution time of task i and  $p_i$  is the period of task i, and you have already known that the value of j will lie between 0 to m where m is the number of classes we want to divide the task set into.

So, we require that the task set will be divided into m classes so I am repeating again, this is very much important, because the classes will be defined based on this utilization bound. So, if the tasks are to be divided into m classes then a task will belong to class j where j lies between 0 and m if and only if the following condition holds.

So, if we will do, it will assign the classes like this or if a task will belong to the class using this formula, this given formula then this will define the utilization grid for various classes. Let us see how this, we can get a graph like, we will call our utilization grid for the various classes.

(Refer Slide Time: 0:20:43)



**NEXT FIT ALGORITHM FOR RMA (cont ...)**

- Let a task set is to be partitioned into 4 classes. Then,
  - Class 1: (0.41,1)
  - Class 2: (0.26,0.41)
  - Class 3: (0.19,0.26)
  - Class 4: (0,0.19)
- We can view the above expression as defining grids on the utilization plot of the tasks, as shown in the above figure.

**NEXT FIT ALGORITHM FOR RMA (cont ...)**

- One or more processors are assigned exclusively to each class of tasks.
- The essence of the algorithm:
  - Tasks with similar utilization are allocated to the same processor.
- If the tasks are to be divided into  $m$  classes,
  - a task belongs to class  $j$  ( $0 \leq j < m$ ), if

$$\left( 2^{\frac{1}{j+1}} - 1 \right) < \frac{e_i}{p_i} \leq \left( 2^{\frac{1}{j}} - 1 \right)$$

- Defines utilization grid for various classes.

Now, what I will do, first suppose, no I am explaining this formula that we have used. Suppose, let us say that we want to divide a class into, a task set into four classes. So, please remember that  $j$  should be greater than 3. So, I have taken here now the value of  $m$  is equal to 4, so that means I require that the task should be divided into four classes.

Now, let us see, what would be bounds, what will the utilization bounds for those four classes. So, number 1, let a task set is to be partitioned into four classes, then we will have four, what four classes, class 1, class 2, class 3, class 4, for class 1 where I will get the value? You can go here. In this equation put the value of  $j$  is equal to 1, because this is the first class.



So, the bound it will say, it will simplify this thing, then you can say that the utilization will lie in between two values. So, what are the two values? The left hand side value will be 0.41 and right hand side value is 1. I have already shown you.

Put just the value of 1 here, right hand side, obviously it is becoming 1, because  $2^1 - 1$ , it is 1. Similarly, the left hand side of this equation it will be 0.41, so that means if we will use this equation the utilization bound for class 1 will lie in the range 0.41 to 1. Similarly, put the value of  $j = 2$  in the equation. You will find out the utilization bound for class 2 that will between 0.26 to 0.41. Put the value of  $j = 3$  in this equation, then you can see the utilization bound for class 3 will be 0.19 to 0.26. And put the value of  $j = 4$ , you can see that the value of the utilization bounds will lie in between 0 to 0.19.

So, in this way we have seen the utilization bounds for these different four classes. Now, what we can do? We can view the above expression as, so this given expression. Now, we can put in the form of a graph, because the values for the four classes we have already shown you, and let us see how does it look like.

It will look like this. There are only four classes, you can look at, there are four classes. I have divided them by putting three blue color lines. So, you can view the above expression here as defining as the grids on the utilization plot of the tasks, which is shown in the above figure. You can see this is just looking like a utilization plot of the task. Or you can say that this is the, this graph looks like the grids on the utilization plot of the task. And there are four tasks, four classes and hence you can see there are four regions I have shown separated by the three blue colored lines.

(Refer Slide Time: 0:24:02)

**NEXT FIT ALGORITHM FOR RMA (cont ...)**

- A task is assigned to a grid depending on its utilization.
- It may be observed that the size of the grids at higher task utilization values are coarser compared to that at low task utilization values.
- E.g., grid size of class 1 tasks= $1-0.41=0.59$ , while grid size of class 3 tasks= $0.26-0.19=0.07$
- Simulation studies shows that:
  - next fit algorithm requires at most 2.34 times the optimum number of processors.

**NEXT FIT ALGORITHM FOR RMA (cont ...)**

- Let a task set is to be partitioned into 4 classes. Then,
  - Class 1: (0.41,1)
  - Class 2: (0.26,0.41)
  - Class 3: (0.19,0.26)
  - Class 4: (0,0.19)
- We can view the above expression as defining grids on the utilization plot of the tasks, as shown in the above figure.

So, one observation you can make from this graph is that. And you can observe that the size of the grids at higher task utilization values are normally coarser compared to that at low utilization values. I am repeating again, you can observe that the size of the grids at higher task utilization. Higher task utilization where it is you can see at class 1 it is higher utilization because it lies between 0.41 to 1, and lowest is coming to be 0 and 0.19, so class 4.

So, here you can see that the size of the grids are the higher task utilization values, they are normally coarser compared to what? Compared to that at the low task utilization values, and now

the important thing I have left that is how the task will be assigned. So, a task is assigned to a grid depending on what? Depending on the utilization.

So, we have seen this is the grid structure a task will be assigned to the grid, a task will be assigned to a grid, depending on what? Depending on its utilization, and this I have already told you that the size of the grids, you can look at the graph, you can see that the size of the grids at higher task utilization value they are normally coarser as compared to that of the task, that are the low task utilization values.

For example, you can see class 1 task, how it is coarser? This range is, how much you can see, it is 0.41 to 1. So, that means its coarser is 1 minus 0.41, that is coming to be 0.59. So, grid size of class 1 tasks are how much it is 0.59, but if we will come to the class 3 task, then you can see how, what value, the grid size, 0.26 minus 0.19, it is coming to be 0.07.


So, we can conclude that the size of the grids are at higher task utilization values. Normally they are coarser as compared to that at the low task utilization values. So, then you will see about its performance. The simulation studies shows that, the next fit algorithm it requires at most 2.34 times the optimum number of processors. So, this algorithm, it requires, so if optimum number of processor required is  $x$  then next fit algorithm requires  $2.34 \times$  times the optimum number of processors. Regarding performance we can say that, the simulation studies show that the next fit algorithm requires utmost 2.34 times of the optimum number of processors. Now, we will take quickly a small example and illustrate how does it work.

(Refer Slide Time: 0:26:32)

### NEXT FIT ALGORITHM FOR RMA - EXAMPLE

- The table below shows the execution time (in milliseconds) and periods (in milliseconds) of a set of 10 periodic real-time tasks.
- Suppose, the tasks need to run on a multiprocessor with four processors.

Task	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>	T <sub>9</sub>	T <sub>10</sub>	T <sub>11</sub>
e <sub>i</sub>	5	7	3	1	10	16	1	3	9	17	20
p <sub>i</sub>	10	21	22	24	30	40	50	55	70	90	100



So, the example is like this. the following table, it shows the execution time of 11 tasks, and this execution time is in millisecond and of a set up, it is actually a typing mistake it should not be 10, it should be 11, 11 periodic real times tasks are there, you can see in the below table, there are 11 periodic tasks and suppose the tasks there need to run on a multiprocessor, with four processors. So, in your multiprocessor system there are four processors there. No, I think, let us see, there might be again a typing mistake here you will see.


(Refer Slide Time: 0:27:06)

### NEXT FIT ALGORITHM FOR RMA - EXAMPLE

- Allocate the tasks using next-fit algorithm.
- Assume that the individual processors are to be scheduled using RMA.

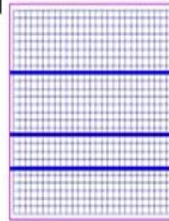
Solution

Task	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>	T <sub>9</sub>	T <sub>10</sub>	T <sub>11</sub>
e <sub>i</sub>	5	7	3	1	10	16	1	3	9	17	20
p <sub>i</sub>	10	21	22	24	30	40	50	55	70	90	100
u <sub>i</sub>	0.5	0.33	0.14	0.04	0.33	0.4	0.02	0.05	0.13	0.19	0.20
class	1	2	4	4	2	2	4	4	4	4	3



### NEXT FIT ALGORITHM FOR RMA (cont ...)

- Let a task set is to be partitioned into 4 classes. Then,
  - Class 1: (0.41,1)
  - Class 2: (0.26,0.41)
  - Class 3: (0.19,0.26)
  - Class 4: (0,0.19)
- We can view the above expression as defining grids on the utilization plot of the tasks, as shown in the above figure.



### NEXT FIT ALGORITHM FOR RMA - EXAMPLE

- The table below shows the execution time (in milliseconds) and periods (in milliseconds) of a set of 10 periodic real-time tasks.
- Suppose, the tasks need to run on a multiprocessor with four processors.

Task	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>	T <sub>9</sub>	T <sub>10</sub>	T <sub>11</sub>
e <sub>i</sub>	5	7	3	1	10	16	1	3	9	17	20
p <sub>i</sub>	10	21	22	24	30	40	50	55	70	90	100



So, now we have to, what is the problem? Problem is allocate the task using the next fit algorithm, assume that the individual processors are to be scheduled using RMA, because I have already told you next fit algorithm will work for this RMA so assume that the individual processors are to be scheduled using RMA.

The solution I have made very simple. First what I have done? I have this, e<sub>i</sub> is giving execution time, p<sub>i</sub> is also given. First what I have to do? I have to find out the value of e<sub>i</sub> / p<sub>i</sub>, that is utilization, find out the value of u<sub>i</sub>, which is equal to e<sub>i</sub> / p<sub>i</sub>. So, every case I have found out like 5 by 10 is 0.5, 7 by 21 is 0.31, like that found out the u<sub>i</sub>.

Then I have to find out what? It will belong to which class, and then next find out that interval, so suppose for example task 1, what is the value of utilization 0.5, so which class it will fit in. let us see. 0.5, obviously it will fit to class 1, because class 1, for class 1 grid size the value lies in between 0.41 to 1.

So, since for the first task it is 0.5, so obviously it will fit in to which class? Class 1, because 0.5 lies in between 0.41 to 1. Similarly find out for task 2, which class it will belong to, 0.33. 0.33 is coming where? You can see 0.33 it lies in between 0.36 to 0.41, which is in class 2. So, this T2 will belong to class 2.




Like this you assign, you divide the task into some classes. So, how many task are there, we have done? We have seen that these 11 tasks are now divided into four classes. So, now we are having four classes. So, this is divided into four classes. Class 1, class 4 and I have shown which task will belong to which class, that I have shown in this table. Now, we will do what? We will find out which task will be assigned to which processor how to do it? Now, let us see.

(Refer Slide Time: 0:29:06)

**NEXT FIT ALGORITHM FOR RMA - EXAMPLE**

- Let us start by earmarking one processor for each class.
- It may be noted that (T2,T5,T6) is not RMA schedulable on the same processor. So, T6 is assigned to an additional processor p6.

Processor	Tasks
p1	T1
p2	T2, T5
p3	T11
p4	T3,T4,T7,T8,T9,T10
p5	T6



## NEXT FIT ALGORITHM FOR RMA - EXAMPLE

- Allocate the tasks using next-fit algorithm.
- Assume that the individual processors are to be scheduled using RMA.

Solution

Task	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>	T <sub>9</sub>	T <sub>10</sub>	T <sub>11</sub>
e <sub>i</sub>	5	7	3	1	10	16	1	3	9	17	20
p <sub>i</sub>	10	21	22	24	30	40	50	55	70	90	100
u <sub>i</sub>	0.5	0.33	0.14	0.04	0.33	0.4	0.02	0.05	0.13	0.19	0.20
class	1	2	4	4	2	2	4	4	4	4	3

So, let us start by here marking one processor for each class. Now, it may be noted that, so first will go toward T<sub>1</sub>, first will go to T<sub>1</sub>. So, T<sub>1</sub> you can see easily it belongs to class 1 and class 1 utilization is 0.5, no problem. So, it will be assigned to processor 1. So, T<sub>1</sub> is assigned to processor 1.

Next T<sub>2</sub> it will belong to which class? It is class 2, what is the utilization, 0.33. Please remember for RMA the maximum utilization can be 1, this is the necessary condition and there is a sufficient condition for RMA, that formula you should remember that is  $n$  to, the utilization should be less than  $n$  into 2 to the power  $n$  minus 1, that you must have read earlier.

So, here, now for T<sub>2</sub>, which class is there? Its utilization 0.33, so I can assign it to processor 2, no problem. Now, when we will come to T<sub>3</sub>, T<sub>3</sub> is in where? It is in class 4. So, class 4 means it will be assigned to processor 4, no problem. Similarly, T<sub>4</sub> it is class 4 so I can assign to processor 4. So, now what is the utilization of processor 4? so 0.14 plus 0.04, that means how much? 0.18. So, no problem. We can go ahead.

Now, coming to T<sub>5</sub>. T<sub>5</sub> belongs to which class? Class 2. So, for class 2 I have to use processor 2 and what is the utilization of this class 2. It is now 0.33, again now 0.33, that is 0.66. It is okay, less than 1. No problem. So, T<sub>5</sub> can also be assigned to processor 2. that I have shown here. So, T<sub>2</sub>, T<sub>5</sub>, they can also be assigned to processor 2. Similarly, it will go ahead.



Now, let us go to next, T3. T3 I have already told you it will be assigned to processor 4. T4 to processor 4. T5 I have already told you processor 2. Now, let us come to T6, this is a problem. T6 you will assign to which one? Class 2. Class 2 means we are using processor 2, but by this time let us see what is the utilization. So, already for class 2 utilization is how much? 0.33, this is T2, next is T5, is 0.33, that means 0.66.

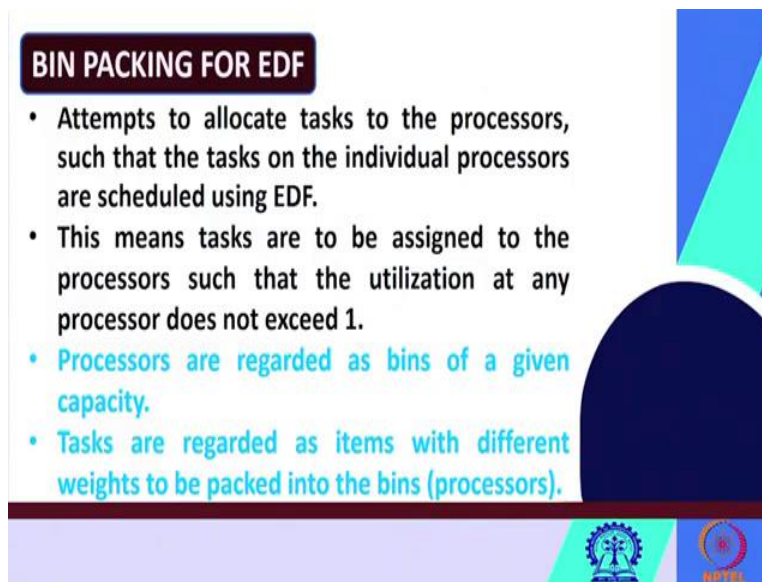
Now, if I will assign again T6 to that processor the utilization will be how much? 0.66 plus 0.4, that means it will be greater than 1. So, I cannot assign T6 to processor 2, because it will not satisfy the schedulability criteria for RMA and hence these three tasks T2, T5, T6, this task set, it is not RMA schedulable, it will not be RMA schedulable on the same processor.

What is the processor? P2. So, I will be forced to assign T6 to one new processor. So, T6 is assigned to an additional processor. Not P6 I am sorry this will be P5, I am sorry this should be P5. So, this P5. So, T6 is assigned to P5, because already I am having P1, P2, P3, P4 for the four classes. So, not T6 will be assigned to the next processor P five.

So, similarly you see the other. Next is T7. T7, it will go to what? Class 4. Till now what is the utilization of this class 4? That means 0.14 plus 0.04 that is 0.18 and this 0.02, that means 0.2. again less than 1, no problem. Similarly, T8 can be assigned again to P4, Because this 0.2 plus 0.05, 0.25, still possible.

Then T9, so 0.25 plus 0.13, because again this is class 4. It is again 0.38, again less than 1 and T10, again 0.15, so still it is there RMA schedulable because they are not exceeding the maximum utilization. So, hence they can be assigned it to, which one? They can be assigned it to those all the tasks, which tasks? T3, T4, T7, T8, T9, T10, they all can be assigned to P4, because the total utilization is not exceeding the maximum utilization, and T11 it is the obvious, it is 3, so this is assigned to processor 3. In this way the different tasks are assigned to the different processors. So, this is how the next fit algorithm, it works. So, let us quickly we will finish the bin packing algorithm for EDF.

(Refer Slide Time: 0:33:46)



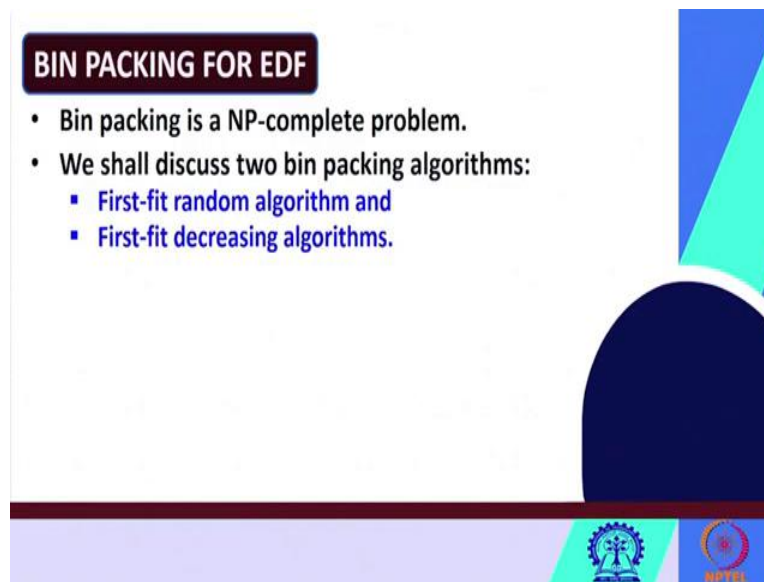
**BIN PACKING FOR EDF**

- Attempts to allocate tasks to the processors, such that the tasks on the individual processors are scheduled using EDF.
- This means tasks are to be assigned to the processors such that the utilization at any processor does not exceed 1.
- Processors are regarded as bins of a given capacity.
- Tasks are regarded as items with different weights to be packed into the bins (processors).

The slide features a dark blue header with the title 'BIN PACKING FOR EDF' in white. The main content is a list of four bullet points. The last two points are highlighted in light blue. The slide is decorated with a large dark blue semi-circle on the right side and a vertical bar with blue and cyan segments on the far right. At the bottom, there are two logos: a circular emblem on the left and a red circular logo with the word 'NPTCL' on the right.

So, this algorithm attempts to allocate tasks to the processors so that the tasks on individual processors are scheduled using EDF. So, this algorithm, it allocates the task to the processors so that the task on the individual processors, they can be scheduled using EDF. This means that the task which are to be assigned to the processors, they will be assigned in such a way that the utilization at any processor does not exceed 1 and the advantage is that the processors, so now let us see this is we are saying bin packing, so what do you mean by bin here, what will you consider bin? So, processors are regarded as bins of a given capacity and the tasks, they are regarded as the items with different weights which need to be packed into the bins. So, bins means the processors are treated as the bins and the tasks are treated as the items.

(Refer Slide Time: 0:34:32)



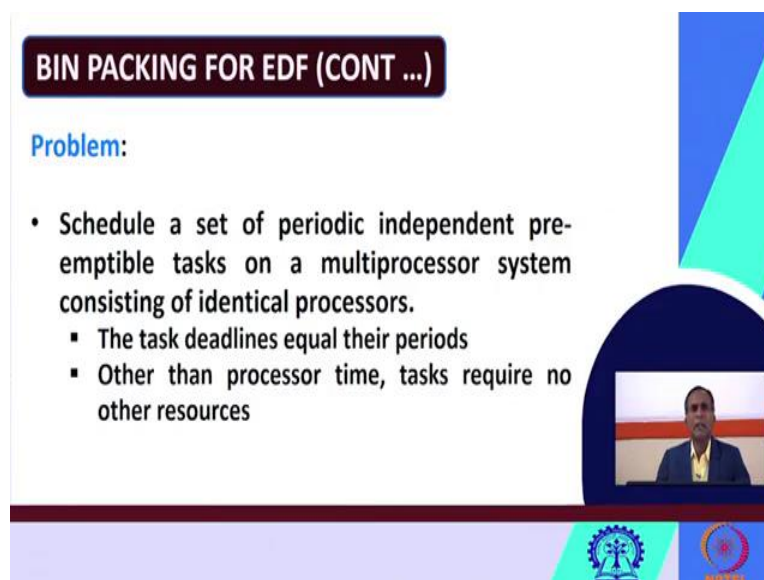
**BIN PACKING FOR EDF**

- Bin packing is a NP-complete problem.
- We shall discuss two bin packing algorithms:
  - First-fit random algorithm and
  - First-fit decreasing algorithms.

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. At the bottom, there are logos for a university and NPTEL.

So, the bin packing normally it is NP complete problem, we will quickly discuss two bin packing algorithms, first fit random algorithm and first decreasing algorithm.

(Refer Slide Time: 0:34:41)



**BIN PACKING FOR EDF (CONT ...)**

**Problem:**

- Schedule a set of periodic independent pre-emptible tasks on a multiprocessor system consisting of identical processors.
  - The task deadlines equal their periods
  - Other than processor time, tasks require no other resources

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 speaking. At the bottom, there are logos for a university and NPTEL.

In bin packing algorithm the problem is like this. We have to schedule a set of periodic independent pre-emptible tasks on a multiprocessor system, consisting of identical processors and the task deadlines here normally equal to their periods and I have already told you that one assumption. The only resource required is the processor time. So, other than the processor time, the task, they do not require any other resources.

(Refer Slide Time: 0:35:08)

**BIN PACKING FOR EDF (CONT ...)**

**Solution:**

- EDF-scheduling on a processor: if  $U < 1$  (for the task set assigned to the processor)  $\Rightarrow$  the set is schedulable on the processor.
- The problem reduces to making task assignments to processors with the property that  $U < 1$ .

And the solution is that, so EDF scheduling, we have to schedule the tasks on a processor using EDF. You know that EDF scheduling, for EDF scheduling on a processor that if  $U < 1$ , the utilization  $< 1$  then you will see that the task set is assigned to that processor. It means the task set is scheduleable on the processor, and the problem, it reduces to making the task assignments to processors with the property that  $U < 1$ . The total utilization should be  $< 1$ .

(Refer Slide Time: 0:35:39)

**FIRST-FIT RANDOM ALGORITHM**

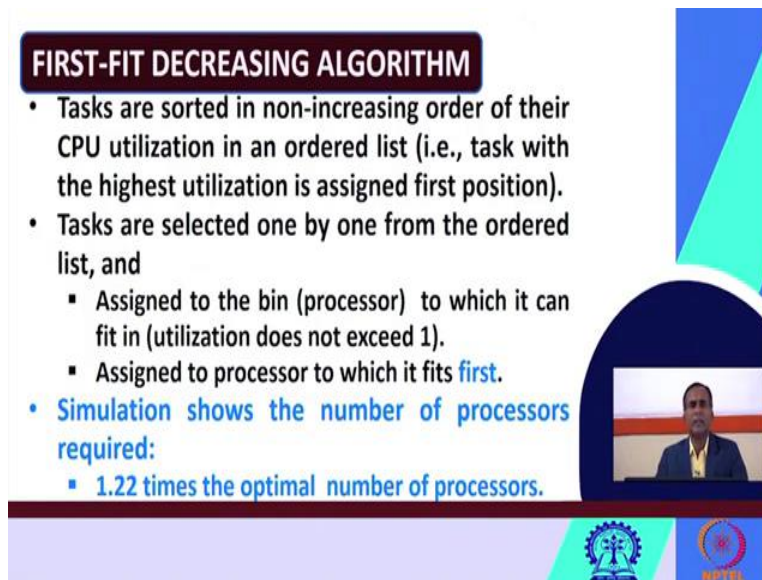
- Tasks are selected randomly and assigned to processors arbitrarily,
  - as long as the utilization of a processor does not exceed 1.
- **Performance:**
  - At most 1.7 times the optimum number of processors are required.

We will quickly see that the first fit random algorithm, here the tasks are selected randomly. That is why the name is first fit random algorithm. So, here the tasks, they are selected randomly and

they are assigned to the processors arbitrarily, again the tasks are assigned to the processors randomly or arbitrarily, as long as the utilization of the processor does not exceed here.

So, till or as long as the utilization of a processor is not exceeding 1 randomly you can assign the task to those processors, it is very simple, that is why the name is first fit random algorithm, and simulation is also that that regarding its performance it is said that the followings, simulation is also that at most 1.7 times the optimum number of processors are required. So, if in optimum number of, if the optimum number of processors required is  $x$ , then first fit random algorithm it will require  $1.7x$  times of the processor. So, it will require at most 1.7 times the optimum number of processors.

(Refer Slide Time: 0:36:40)



**FIRST-FIT DECREASING ALGORITHM**

- Tasks are sorted in non-increasing order of their CPU utilization in an ordered list (i.e., task with the highest utilization is assigned first position).
- Tasks are selected one by one from the ordered list, and
  - Assigned to the bin (processor) to which it can fit in (utilization does not exceed 1).
  - Assigned to processor to which it fits **first**.
- Simulation shows the number of processors required:
  - 1.22 times the optimal number of processors.

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

## FIRST-FIT RANDOM ALGORITHM

- Tasks are selected randomly and assigned to processors arbitrarily,
  - as long as the utilization of a processor does not exceed 1.
- Performance:
  - At most 1.7 times the optimum number of processors are required.

Then next is first fit decreasing algorithm. So, as its name suggests, first fit decreasing algorithm, here the tasks are sorted in non-increasing order, maybe in the decreasing order of their CPU utilization, in an ordered list. That means what we can say? The task with the highest utilization is assigned the first position and so on.

Then what will happen, then the tasks are selected one by one from the ordered list, please select the tasks or extract the tasks one by one from the ordered list. Then you assign the task to the bins. You assign the tasks you have extracted, the tasks you have removed, the tasks you have removed you assign them, you assign it to the bin, bin means to the processor to which it can fit in.

And how can we know whether it can fit in or not? We can add the task, we can assign the task as long as the utilization does not exceed 1, and another thing is that how can assign the task? We can assign the task to the processor to which it fits first. So, we have to assign the task to the processor in such way that it should fit the first.

So, assign the processor. Assign to processor or assign the task to the processor to which it fits first and similarly they are adding the performance, the simulation shows that the number of processors required is 1.22 times the optimal number of the processors. So, first fit random algorithm requires 1.7 times the optimum number of processors, whereas first fit decreasing algorithm requires 1.22 times the optimal number of processors.

(Refer Slide Time: 0:38:11)

## FIRST-FIT DECREASING ALGORITHM - EXAMPLE

- Suppose there are  $n_T$  tasks to be assigned.
- Prepare a sorted list  $L$  of the tasks so that their utilizations (i.e.,  $u_i = e_i/p_i$ ) are in decreasing order.

Task	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$	$T_8$	$T_9$	$T_{10}$	$T_{11}$
$e_i$	5	7	3	1	10	16	1	3	9	17	21
$p_i$	10	21	22	24	30	40	50	55	70	100	95
$u_i$	0.5	0.33	0.14	0.04	0.33	0.4	0.02	0.05	0.13	0.17	0.22

So, we will quickly take a small example. Suppose there are  $n_T$  tasks to be assigned, first we have to prepare a sorted list  $L$  of the tasks. So, their utilization, so that their utilizations, they are in decreasing order. Utilization means you know  $e_i / p_i$ , so 11 tasks are given. Their execution times and periods are given. I have computed  $u_i$  as using the formula  $e_i / p_i$  so these are the  $u_i$  utilizations.

(Refer Slide Time: 0:38:41)

## FIRST-FIT DECREASING ALGORITHM - EXAMPLE

### Solution

- The ordered list  $L$  is  $L = (T_1, T_6, T_2, T_5, T_{11}, T_{10}, T_3, T_9, T_8, T_4, T_7)$ .

Step	1	2	3	4	5	6	7	8	9	10	11
$T_i$	$T_1$	$T_6$	$T_2$	$T_5$	$T_{11}$	$T_{10}$	$T_3$	$T_9$	$T_8$	$T_4$	$T_7$
$u_i$	0.5	0.4	0.33	0.33	0.22	0.18	0.14	0.13	0.06	0.04	0.02
Assign	p1	p1	p2	p2	p2	p3	p3	P3	p1	p1	P2
Post-Assignm-ent U vector	0.5	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.96	1	1
			0.33	0.66	0.88	0.88	0.88	0.88	0.88	0.88	0.9
						0.18	0.32	0.45	0.45	0.45	0.45



## FIRST-FIT DECREASING ALGORITHM - EXAMPLE

- Suppose there are  $n_T$  tasks to be assigned.
- Prepare a sorted list  $L$  of the tasks so that their utilizations (i.e.,  $u_i = e_i/p_i$ ) are in decreasing order.

Task	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$	$T_8$	$T_9$	$T_{10}$	$T_{11}$
$e_i$	5	7	3	1	10	16	1	3	9	17	21
$p_i$	10	21	22	24	30	40	50	55	70	100	95
$u_i$	0.5	0.33	0.14	0.04	0.33	0.4	0.02	0.05	0.13	0.17	0.22

And then what we have done? We have ordered them, how? In the decreasing order. So, first will must come, what? You can say 0.5, this is  $T_1$ . After that which one will come, you can say that next will come  $T_6$ . This is the utilization, we have arranged the utilization. First one is  $T_1$ , then the utilization we have already computed here.

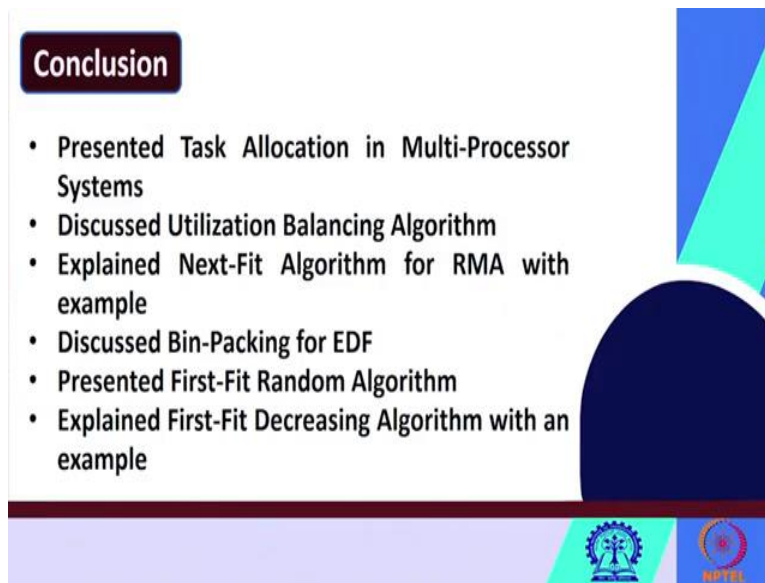
Next one is  $T_6$ , because after 0.5, then I have to put 0.4.  $T_6$ , then it is  $T_2$ , this is 0.33, and in case of tie, 0.33, 0.33 any one you can put early. So, like this I have created this ordered list, and then to which processor I will assign, first one. 0.5 I can assign to  $P_1$ , processor 1, and what is this, what utilization, total utilization by this point 0.5, no problem.

Then second task  $T_6$ , again assigned to  $P_1$ , because 0.5 plus 0.4 is 0.9, still less than 1, no problem, but while  $T_2$  will come I cannot add that this  $T_2$  to  $P_1$ , because 0.9 plus 0.33 it will be greater than 1, so I have to add a new processor  $P_2$ . Like that,  $T_5$ , again I can, I assign to  $P_2$ , because the total utilization is now 0.66.  $T_{11}$  also can be assigned to  $P_2$ , because the total utilization is 0.88.

But  $T_{10}$  cannot be assigned to  $P_2$ , because if you will add 0.88, with 0.18, it will be greater than 1. So, I have to use another processor  $P_3$ . But  $T_3$ ,  $T_9$ , they can be assigned to  $P_3$ , because this value is less 0.45 is less than 1. So, when  $T_8$  will come, 0.06, please see I am using first fit, I have to see 0.06, now it can be assigned to the first processor, because  $P_1$ , it is utilization is 0.9, it can accommodate. I can add 0.06, still it is 0.96, less than 1, no problem. So,  $T_8$  can be assigned to  $P_1$ .

And while coming to T4, so it is 0.04, so add 0.96 plus 0.04, it is exactly 1, still it can be assigned to P1, but the other T7, I cannot assign to P1, because it is already 1. let us see it can be assigned to P2, yes, it is 0.88, add 0.02, it will be 0.9, which is less than 1. So, it can be assigned to P2, like this. in this way we can apply first fit decreasing algorithm for allocating the tasks.

(Refer Slide Time: 0:41:06)



**Conclusion**

- Presented Task Allocation in Multi-Processor Systems
- Discussed Utilization Balancing Algorithm
- Explained Next-Fit Algorithm for RMA with example
- Discussed Bin-Packing for EDF
- Presented First-Fit Random Algorithm
- Explained First-Fit Decreasing Algorithm with an example

So, today I have discussed the task allocation in multiprocessor systems, we have explained the static allocation algorithms such as my utilization balancing algorithm, next fit algorithm for RMA, bin packing algorithm for EDF and two special cases of bin packing algorithm, we have seen first fit random algorithm and first fit decreasing algorithm, with the examples. We have seen their performances, their simulation results we have also seen. For each of the algorithm we have seen their performances.

(Refer Slide Time: 0:41:34)



We have taken from these books, the contents what we have discussed today. These two books we have used. Thank you very much.