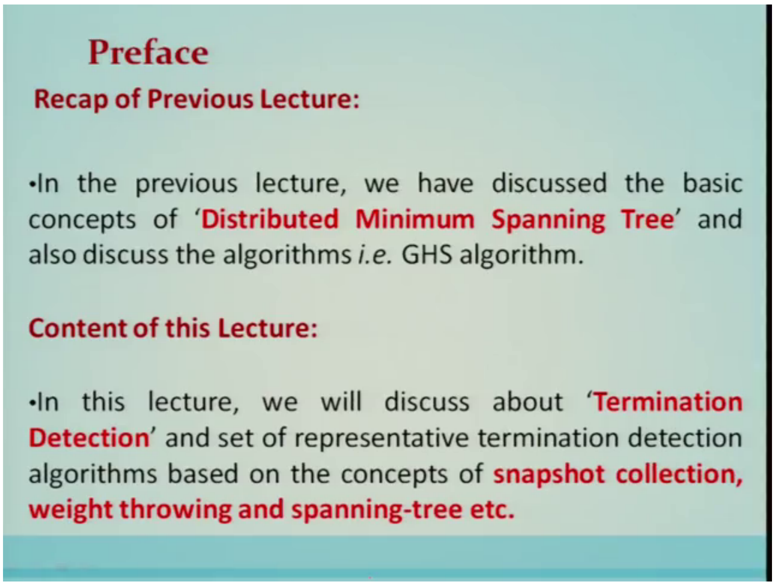
Distributed Systems

Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 15
Termination Detection

Lecture 15 Termination Detection preface recap of previous lecture.

(Refer Slide Time: 00:19)



Preface

Recap of Previous Lecture:

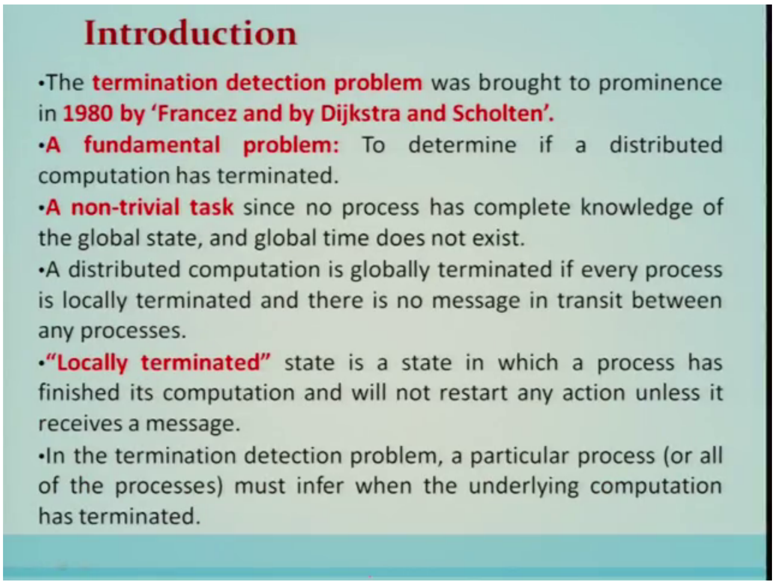
•In the previous lecture, we have discussed the basic concepts of '**Distributed Minimum Spanning Tree**' and also discuss the algorithms *i.e.* GHS algorithm.

Content of this Lecture:

•In this lecture, we will discuss about '**Termination Detection**' and set of representative termination detection algorithms based on the concepts of **snapshot collection, weight throwing and spanning-tree etc.**

Previous lecture we have discussed the basic concepts of distributed minimum spanning trees and also discussed the algorithms that is GHS algorithm, content of this lecture. In this lecture we will discuss about 'Termination Detection' and a set of representative algorithms for termination detection. Termination detection based on concepts of snapshot based on weight throwing and his spanning-trees etcetera these different algorithms which we are going to see in this lecture.

(Refer Slide Time: 00:55)



Introduction

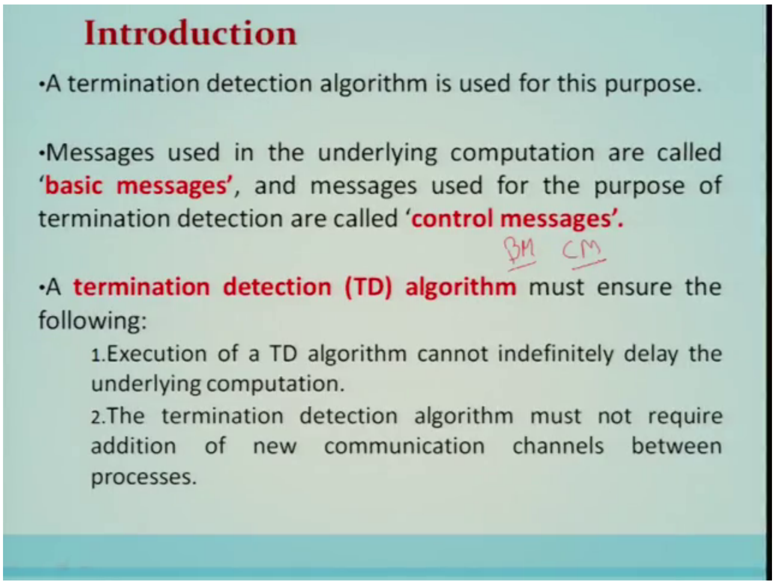
- The **termination detection problem** was brought to prominence in **1980 by 'Francez and by Dijkstra and Scholten'.**
- **A fundamental problem:** To determine if a distributed computation has terminated.
- **A non-trivial task** since no process has complete knowledge of the global state, and global time does not exist.
- A distributed computation is globally terminated if every process is locally terminated and there is no message in transit between any processes.
- **"Locally terminated"** state is a state in which a process has finished its computation and will not restart any action unless it receives a message.
- In the termination detection problem, a particular process (or all of the processes) must infer when the underlying computation has terminated.

Introduction termination detection problem was brought to the prominence in 1980 by the famous people, a fundamental problem to determine if the distributed computation has terminated, meaning in the sense that the distributed computation consists of the cooperation of many processors and the corresponding their communication channel.

So, the distributed application which runs on different processors required to know when the entire computation is completed this is a fundamental problem as far as distributed computation is concerned this task of finding the termination detection is a non-trivial task.

Because there does not exist a global time and also there does not exist a global state and the processes no process has the complete knowledge of the global state. In this particular setting finding out the termination of a distributed application is a non-trivial task. A distributed computation is globally terminated if every processes locally terminated and there is no message in transit between any processes, given this definition locally terminated is state is a is state in which a process has finished it is execution and will not restart any action unless it receives a message. In termination detection problem a particular process must infer when the underlying computation has terminated and it has a different application it has wider application in the distributed system applications.

(Refer Slide Time: 03:03)



Introduction

- A termination detection algorithm is used for this purpose.
- Messages used in the underlying computation are called '**basic messages**', and messages used for the purpose of termination detection are called '**control messages**'.

BM CM

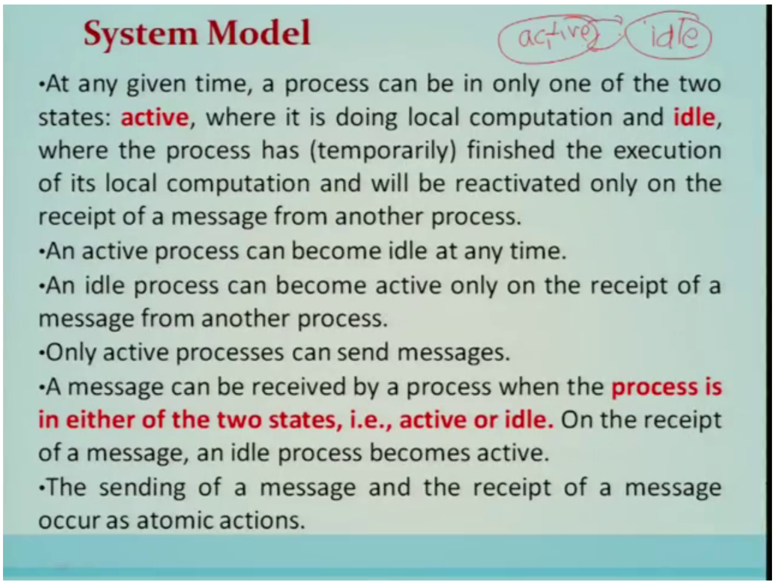
- A **termination detection (TD) algorithm** must ensure the following:
  1. Execution of a TD algorithm cannot indefinitely delay the underlying computation.
  2. The termination detection algorithm must not require addition of new communication channels between processes.

So, Introduction termination detection algorithm is used for this particular purpose that is the messages used in the underlying computation are called basic messages and the messages used for the purpose of termination detection are called control messages.

So, in termination detection algorithm we are going to use 2 different type of messages. So, the messages which are normal computational messages called 'basic messages' and the messages which will basically lead to the termination detection messages are called the 'control messages'.

Termination detection algorithm must ensure the following that is first is the execution of termination detection algorithm cannot indefinitely delay the underlying computation. Second is the termination detection algorithm must not require additional new communication channels between the process so; that means, without disturbing the underlying computation how the termination detection algorithm works and how the termination is detected is going to be an important issue for designing such algorithms.

(Refer Slide Time: 04:22)



System Model

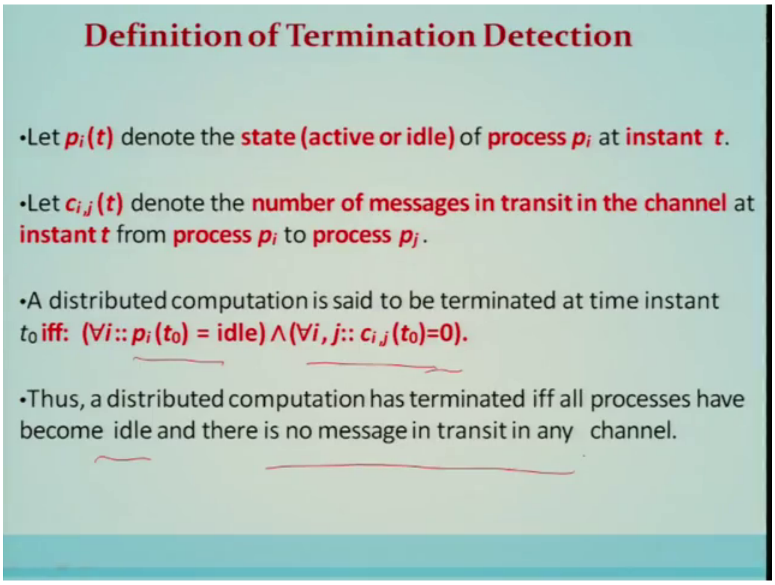
- At any given time, a process can be in only one of the two states: **active**, where it is doing local computation and **idle**, where the process has (temporarily) finished the execution of its local computation and will be reactivated only on the receipt of a message from another process.
- An active process can become idle at any time.
- An idle process can become active only on the receipt of a message from another process.
- Only active processes can send messages.
- A message can be received by a process when the **process is in either of the two states, i.e., active or idle.** On the receipt of a message, an idle process becomes active.
- The sending of a message and the receipt of a message occur as atomic actions.

Now, the system model which is going to be assumed in our discussion for designing the transmission detection algorithm, at any given time a process can be in one of the 2 states that is the active, and the idle state. So, if it is in active state; that means, it is doing the local computation and idle means where the process has temporarily finished the execution of it is local computation and will be reactivated only on the receipt of the message from another process. So, these states basically are not fixed they are changing.

So, an active process can become idle at any point of time and idle process can become active only on the receipt of a message from any other process only active process can send the messages. Now a message can be received by a process when the process is either active or idle. So, on the receipt of a message the idle process can become a active that is shown here.

So, the sending of a message and the receipt of a message occur as atomic actions.

(Refer Slide Time: 05:48)



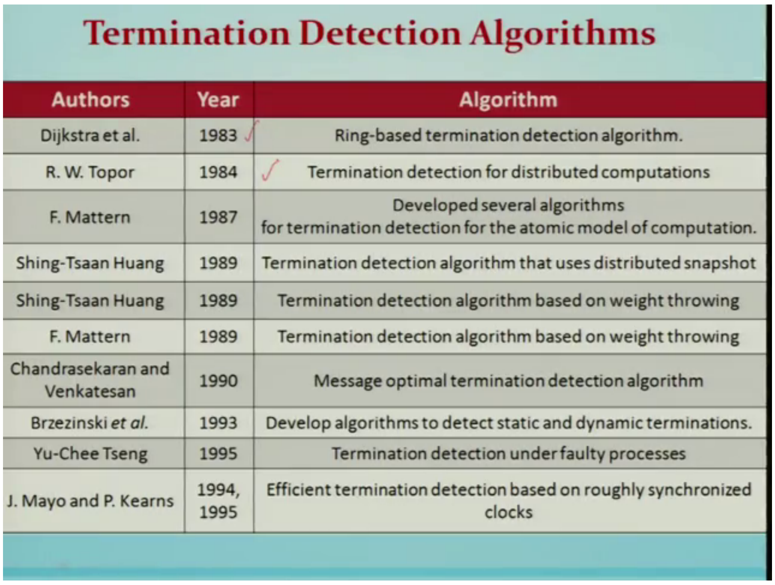
Definition of Termination Detection

- Let $p_i(t)$ denote the **state (active or idle)** of **process p_i** at **instant t**.
- Let $c_{i,j}(t)$ denote the **number of messages in transit in the channel** at **instant t** from **process p_i** to **process p_j**.
- A distributed computation is said to be terminated at time instant t_0 **iff: $(\forall i :: p_i(t_0) = \text{idle}) \wedge (\forall i, j :: c_{i,j}(t_0)=0)$.**
- Thus, a distributed computation has terminated iff all processes have become idle and there is no message in transit in any channel.

Now we are going to define the termination detection formally let p i t denote the state that is either active or idle of a process i at the time instant t. Let c i j at time t denotes the number of messages in transit in the channel at the instant t from pi to pj.

A distributed computation is said to be terminated at any time instant let us say t 0 if and only if for all processes at time t 0 by the state is idle and for all the channels between i and j at time instants t 0 is having not having any messages; that means, it is an empty channel or there is no message in the channel thus the distributed computation is terminated if and only if all the processes have become idle and there is no message in transit in any of these communication channel then the state of the global state is basically defined as the termination detection or is a termination state.

(Refer Slide Time: 07:06)



Termination Detection Algorithms

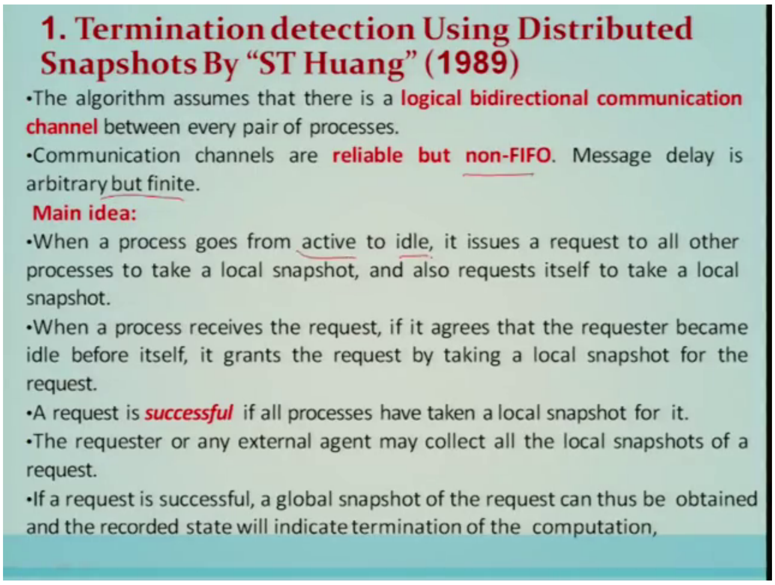
Authors	Year	Algorithm
Dijkstra et al.	1983 ✓	Ring-based termination detection algorithm.
R. W. Topor	1984	✓ Termination detection for distributed computations
F. Mattern	1987	Developed several algorithms for termination detection for the atomic model of computation.
Shing-Tsaan Huang	1989	Termination detection algorithm that uses distributed snapshot
Shing-Tsaan Huang	1989	Termination detection algorithm based on weight throwing
F. Mattern	1989	Termination detection algorithm based on weight throwing
Chandrasekaran and Venkatesan	1990	Message optimal termination detection algorithm
Brzezinski *et al.*	1993	Develop algorithms to detect static and dynamic terminations.
Yu-Chee Tseng	1995	Termination detection under faulty processes
J. Mayo and P. Kearns	1994, 1995	Efficient termination detection based on roughly synchronized clocks

So, Termination Detection Algorithm there are various termination detection algorithms available in the literature. So, these algorithms are different based on what topology they are going to assume and also how they are going to basically address the basic and the control message flows and without the destruction of the basic computation how the termination is detected.

So, Dijkstras algorithm is ring based termination detection algorithm that is given first in 1983 after that Topor has given the termination detection algorithm for the distributed computation that we are going to study.

So, there are different kind of algorithms and Huang also has given a transmission detection algorithm based on weight throwing that we are going to cover in this part of the lecture.

(Refer Slide Time: 08:03)



1. Termination detection Using Distributed Snapshots By "ST Huang" (1989)

- The algorithm assumes that there is a logical bidirectional communication channel between every pair of processes.
- Communication channels are reliable but non-FIFO. Message delay is arbitrary but finite.

Main idea:

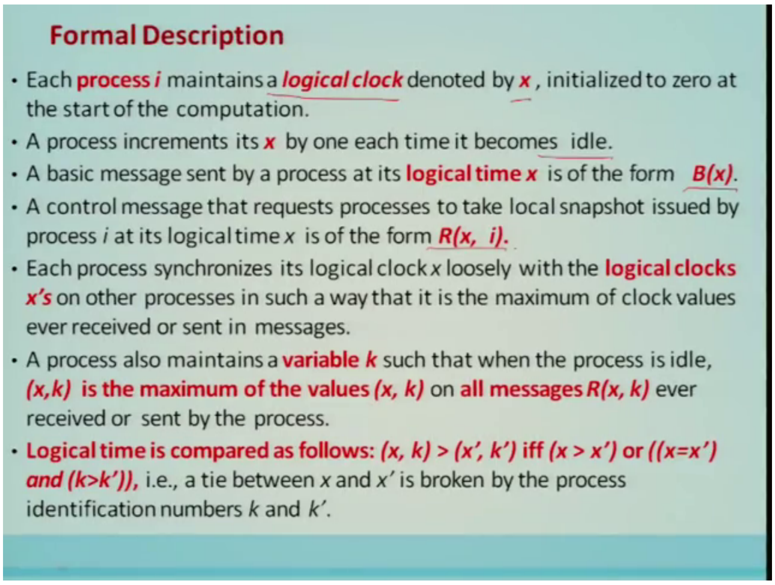
- When a process goes from active to idle, it issues a request to all other processes to take a local snapshot, and also requests itself to take a local snapshot.
- When a process receives the request, if it agrees that the requester became idle before itself, it grants the request by taking a local snapshot for the request.
- A request is *successful* if all processes have taken a local snapshot for it.
- The requester or any external agent may collect all the local snapshots of a request.
- If a request is successful, a global snapshot of the request can thus be obtained and the recorded state will indicate termination of the computation,

Termination detection algorithm using distributed snapshot by Huang in 1989 that we are going to understand first the algorithm assumes that there is a logical bidirectional communication channel between any every process. Communication channels are reliable, but non-FIFO. So, message delays are arbitrary, but they are finite main idea of this algorithm is that when a process goes from active to the idle it issues a request to all the other process to take the local snapshot and also request itself to take the local snapshot.

So, when a process receives this request if it agrees that the requester became idle before it itself then it grants the request by taking a local snapshot for the request a request is successful if all the processes have taken a local snapshot for it the requester or the external aj agent may collect all the local snapshot of the request.

If a request is successful global snapshot of the request can thus be obtained and recorded global state will indicate the termination of the computation.

(Refer Slide Time: 09:15)



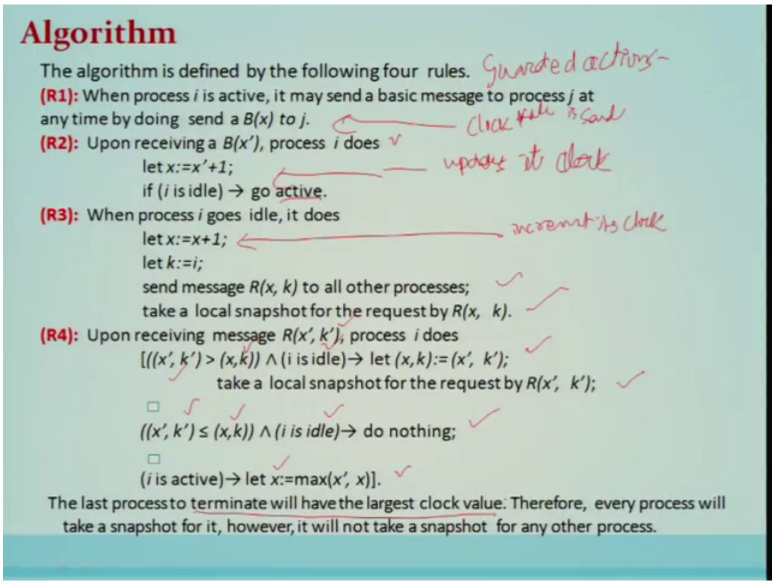
Formal Description

- Each **process *i*** maintains a ***logical clock*** denoted by ***x***, initialized to zero at the start of the computation.
- A process increments its ***x*** by one each time it becomes idle.
- A basic message sent by a process at its **logical time *x*** is of the form ***B(x)***.
- A control message that requests processes to take local snapshot issued by process *i* at its logical time *x* is of the form ***R(x, i)***.
- Each process synchronizes its logical clock *x* loosely with the **logical clocks *x's*** on other processes in such a way that it is the maximum of clock values ever received or sent in messages.
- A process also maintains a **variable *k*** such that when the process is idle, ***(x,k)* is the maximum of the values *(x, k)*** on **all messages *R(x, k)*** ever received or sent by the process.
- **Logical time is compared as follows: *(x, k) > (x', k')* iff *(x > x')* or *((x=x')* *and (k>k'))*,** i.e., a tie between *x* and *x'* is broken by the process identification numbers *k* and *k'*.

So, formal description of this algorithm goes like this every process i maintains a logical clock and which is denoted by x which is initialized to 0 at the start of the competition process increments it is x by 1 each time it becomes idle. So, the basic message sent by the process at it is logical time x is represented by B of x. A control message that requests the processes to take the local snapshot should by your process i at a logical time x is represented by R x of i, each process synchronizes it is logical clock s loosely with the logical clocks x's on the other processes in such a way that it is the maximum of the clock values ever received or sent in the message.

A process also maintains a variable cases that when process is idle to x k pair is maximum of the values xk on all the messages that is R x k ever received or sent by the logical time is compared as follows x comma k is greater than x prime comma k prime if and only x is greater than x prime or x is equal to x prime and k is greater than k prime that is if there is a tie between x and x and x bar here then the process ids are used to break this particular tie in this comparison.

(Refer Slide Time: 10:57)



Algorithm

The algorithm is defined by the following four rules. Guarded actions

(R1): When process *i* is active, it may send a basic message to process *j* at any time by doing send a *B(x) to j*. clock value is send

(R2): Upon receiving a *B(x')*, process *i* does

```
        let x:=x'+1;          update its clock
        if (i is idle) → go active.
```

(R3): When process *i* goes idle, it does

```
        let x:=x+1;           increment its clock
        let k:=i;
        send message R(x, k) to all other processes;
        take a local snapshot for the request by R(x, k).
```

(R4): Upon receiving message *R(x', k')*, process *i* does

```
        [((x', k') > (x,k)) ∧ (i is idle)→ let (x,k):=(x', k');
                take a local snapshot for the request by R(x', k');
        □
        ((x', k') ≤ (x,k)) ∧ (i is idle)→ do nothing;
        □
        (i is active)→ let x:=max(x', x)].
```

The last process to terminate will have the largest clock value. Therefore, every process will take a snapshot for it, however, it will not take a snapshot for any other process.

Now, the algorithm has 4 different rules they are Guarded actions which basically will be activated any one of them at a particular time depending on the situation when these rules are qualifying.

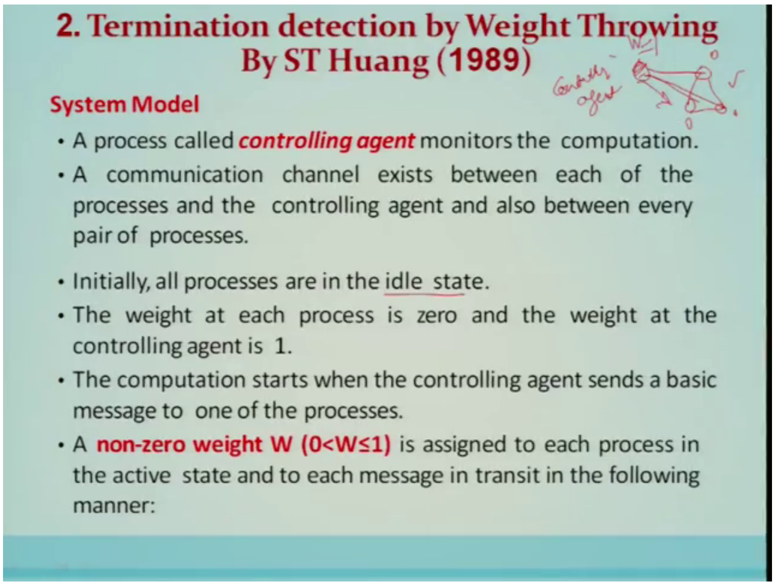
So, the rule 1 says that when a process i is active it will send a basic message to process j at any time by sending be x to j on receiving this particular message the x prime process i does it basically updates it is clock here it will send the clock value. Now if process i is idle then on receiving this message it will become an active here in rule 2.

Rule 3 says that when process i goes idle it does 2 things it will increase or it will increment it is clock and then it will send the message R x comma k to all other process and also it will take the local snapshot for the request x comma k. Rule 4 says that when this particular local snapshot taking message that is a control message is received at the process i it does find out whether the message whether the timestamp of the incoming message is having higher time than the process i and also process i is idle if it is then it will update it is clock and also it will take the local snapshot for the request which is received by R x prime and x prime.

Now, if the clock value which is basically piggy bagged or which is coming inside the message if it is having lesser value then and i is idle then it will not do anything why because the existing process that is i is finished or is idle or is terminated after the receipt of message so nothing has to be done.

Now, third point here in are 4 rule says that if i is active and on receiver of this particular message then it will not or it is not terminated then basically it will only in or it will only update it is clock why because it is still active not terminated . So, the last process to terminate will have the largest clock value therefore, every process will take a snapshot for it; however, it will not take the snapshot off for any other process.

(Refer Slide Time: 14:43)


Controlling agent
W=1
0
0

2. Termination detection by Weight Throwing By ST Huang (1989)

System Model

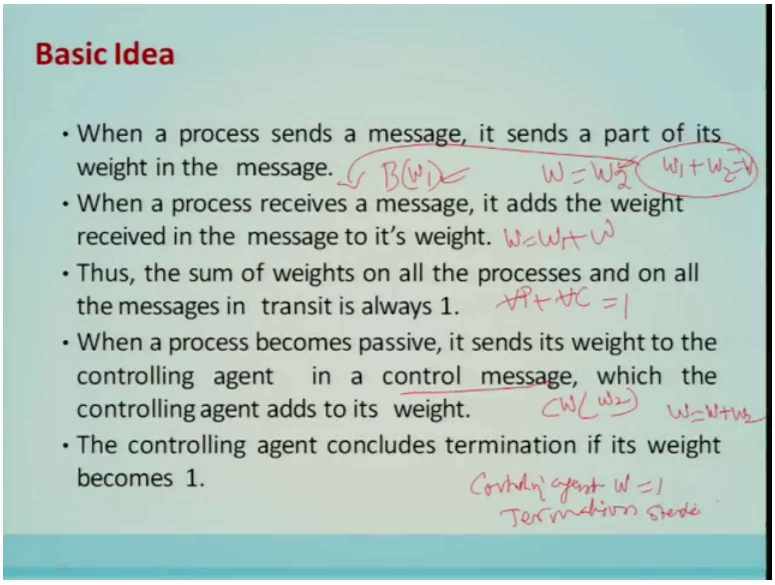
- A process called ***controlling agent*** monitors the computation.
- A communication channel exists between each of the processes and the controlling agent and also between every pair of processes.
- Initially, all processes are in the idle state.
- The weight at each process is zero and the weight at the controlling agent is 1.
- The computation starts when the controlling agent sends a basic message to one of the processes.
- A **non-zero weight W ($0<W\leq1$)** is assigned to each process in the active state and to each message in transit in the following manner:

The second algorithm for termination detection is known as the weight throwing algorithm and is given by Huang in 1989.

So, we will first understand the system model which is used in this algorithm a process called controlling agent monitors the computation. So, if let us say that this is the computation model and this is the controlling process. So, it should have the connection with all other processes and this is called a controlling agent which will monitor the computation.

A communication channel exists between each process and the controlling agent and also between every pair of process that we have seen in this figure initially all the processes are in idle state the weight at each process is 0 and the weight at the controlling is 1 whereas, here the weights are all 0. The computation starts when controlling agent sends a basic message to one of the process here the non-zero weight w from between 0 and one is assigned to each process in the active state and to each message in the transit in the following manner.

(Refer Slide Time: 16:05)



Basic Idea

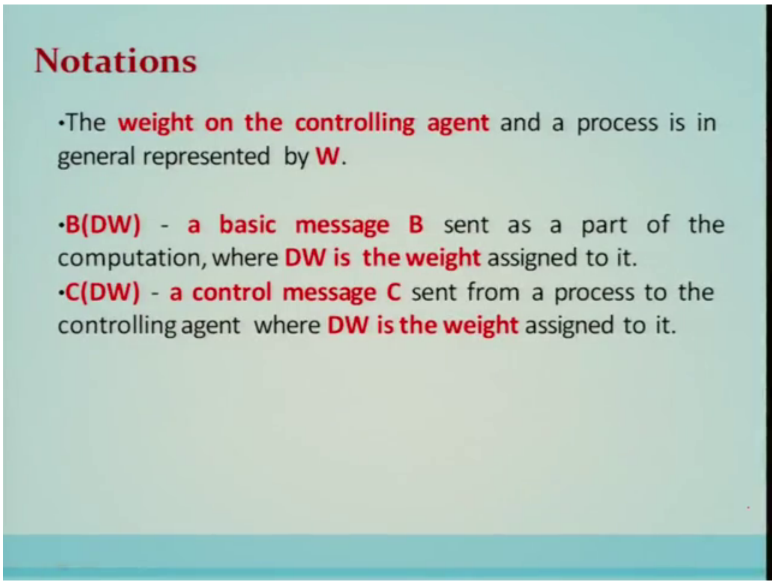
- When a process sends a message, it sends a part of its weight in the message. $B(W_1)$ $W = W_2$ $W_1 + W_2 = W$
- When a process receives a message, it adds the weight received in the message to it's weight. $W = W_1 + W$
- Thus, the sum of weights on all the processes and on all the messages in transit is always 1. $\forall P + \forall C = 1$
- When a process becomes passive, it sends its weight to the controlling agent in a control message, which the controlling agent adds to its weight. $CW(W_2)$ $W = W + W_2$
- The controlling agent concludes termination if its weight becomes 1. Controlling agent $W = 1$ Termination state

So, when a process sends a message it sends a part of it is weight in the message, when a process receives a message it adds the weight received in the message to it is weight. So, for example, if B w 1 is basically carried on the message so w will be updated as w 2 where w 1 plus w 2 was equal to w initially.

So, w was broken into 2 parts and w 2 will be retained is it is current weight and w 1 will be sent. So, the process receiving this particular message it adds it is weight in the message to it is own weight thus the sum of the weights on all the processes and on the message in transit is always 1. So, if we can sum all the weights of all the processes for all the process and weights of all the channels that is equal to 1. So, when a process becomes passive it sends a weight to the controlling agent in the control message. So, that is called CW and this particular weight will be sent.

Whatever to the controlling agent the controlling agent, which controlling is an x to it is weight? So, controlling agent w will adds w plus this w 2, now if the controlling agent concludes the termination if the weight is the controlling agent weight will become 1 then it will conclude that termination state of the algorithm.

(Refer Slide Time: 18:19)



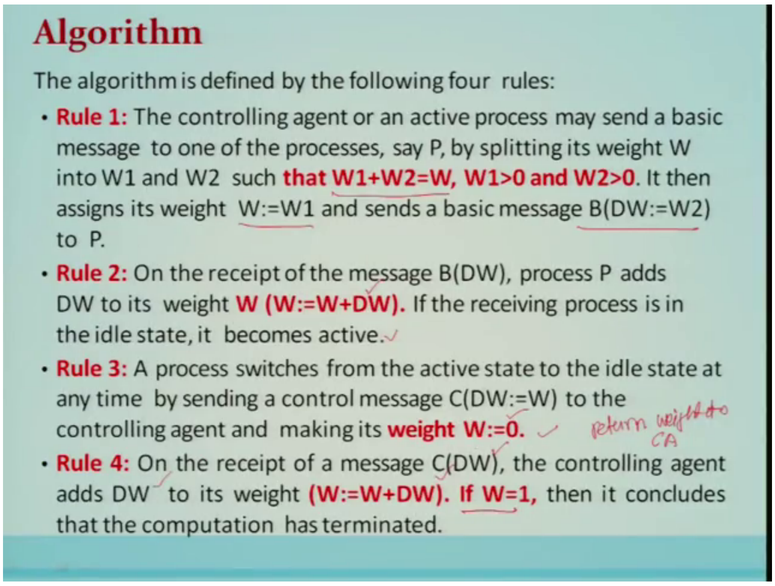
Notations

- The **weight on the controlling agent** and a process is in general represented by **W**.

- **B(DW)** - **a basic message B** sent as a part of the computation, where **DW is the weight** assigned to it.
- **C(DW)** - **a control message C** sent from a process to the controlling agent where **DW is the weight** assigned to it.

Notations the weight on the controlling edge and the process is represented by W here B W is the basic message which is carrying a weight and CW is the control message which is carrying a weight from a process to the controlling agent.

(Refer Slide Time: 18:34)



Algorithm

The algorithm is defined by the following four rules:

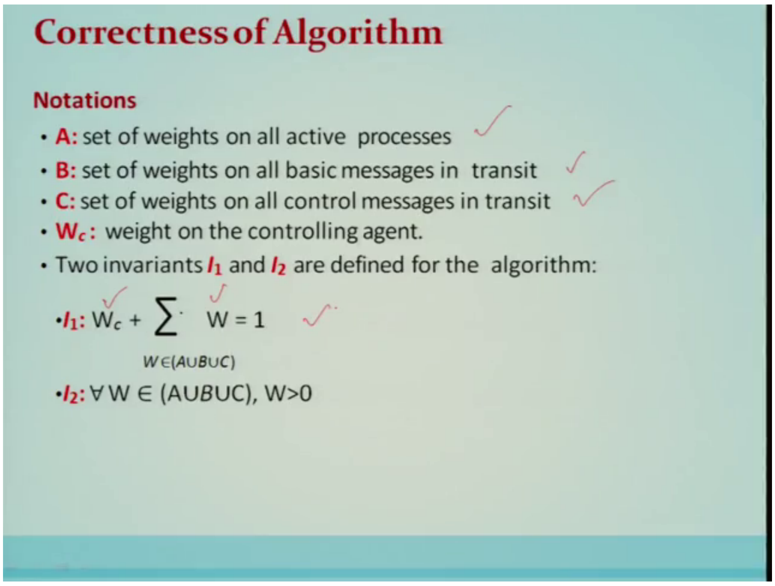
- **Rule 1:** The controlling agent or an active process may send a basic message to one of the processes, say P, by splitting its weight W into W1 and W2 such **that W1+W2=W, W1>0 and W2>0**. It then assigns its weight W:=W1 and sends a basic message B(DW:=W2) to P.
- **Rule 2:** On the receipt of the message B(DW), process P adds DW to its weight **W (W:=W+DW).** If the receiving process is in the idle state, it becomes active.✓
- **Rule 3:** A process switches from the active state to the idle state at any time by sending a control message C(DW:=W) to the controlling agent and making its **weight W:=0.** ✓ return weight to CA
- **Rule 4:** On the receipt of a message C(DW), the controlling agent adds DW to its weight **(W:=W+DW). If W=1,** then it concludes that the computation has terminated.

So, this is the algorithm. So, the algorithm is defined in 4 different rules here Rule 1 says that the controlling agent or an active process may send the basic message to one of the processes, say P, by splitting it is weight W into W 1 and W 2 that I told. W 1 and W 2 they are non-zero if then assigns it is weight W is equal to W 1 and sends the basic message with the weight assigned as the other part that is W 2 P on receipt of this particular message the process P S DW to it is weight if the receiving process is in idle state it will become an active.

Rule 3 says that the process switches from active state to the idle state at any time by sending the control message see DW is equal to W to the controlling agent and making it is way to W; that means, it will return the weight to the controlling to the controlling agent and put and basically put it is weight as 0.

So, on received of this such messages which is shown in Rule number 3 the controlling agent will add DW which is coming from different processes to it is weight now if after getting the weights come from all the processes if summation of the weight is equal to 1 then the controlling agent will conclude that the computation is has terminated.

(Refer Slide Time: 20:19)



Correctness of Algorithm

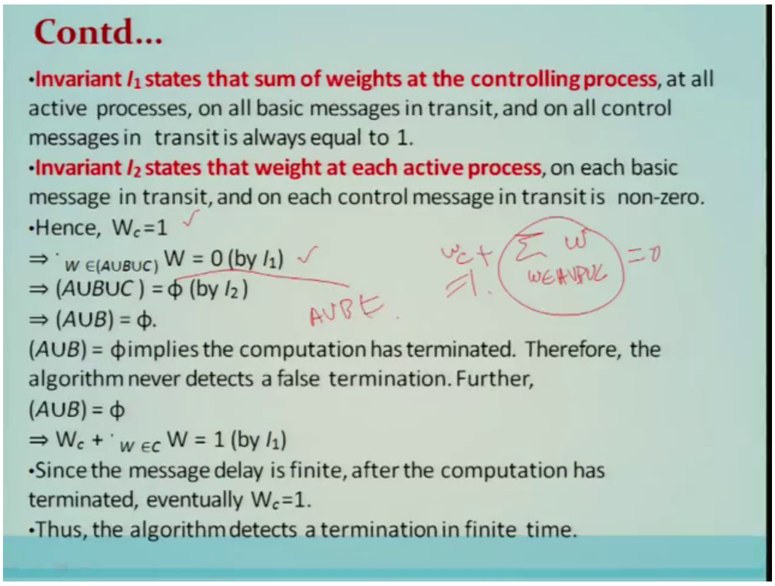
Notations

- **A:** set of weights on all active processes
- **B:** set of weights on all basic messages in transit
- **C:** set of weights on all control messages in transit
- **W_c:** weight on the controlling agent.
- Two invariants I_1 and I_2 are defined for the algorithm:
  - I_1: $W_c + \sum_{W \in (A \cup B \cup C)} W = 1$
  - I_2: $\forall W \in (A \cup B \cup C), W > 0$

Now, the Correctness of this Algorithm let capital A be the set of all active processes, capital B with the set of all basic messages in concept, Capital C is the set of weights on all the control messages in the transit and Wc is the weight on the controlling agent, there are 2 invariants I 1 and I 2 which are defined for the algorithm like this.

So, variant I 1 says that the weight of weight on the controlling agent plus, the sum of the weights on a that is all the active processes and the weights on all the basic messages in transit and some of all the control messages in transit is equal to 1 ; that means, entire weight on the system is equal to 1. And second invariance is says that that for all W. So, a union B union C and W is greater than 0.

(Refer Slide Time: 21:19)



Contd...

•**Invariant I_1 states that sum of weights at the controlling process**, at all active processes, on all basic messages in transit, and on all control messages in transit is always equal to 1.

•**Invariant I_2 states that weight at each active process**, on each basic message in transit, and on each control message in transit is non-zero.

•Hence, $W_c=1$ ✓

$\Rightarrow {}^{\cdot}{}_{W \in (A \cup B \cup C)} W = 0$ (by I_1) ✓

$\Rightarrow (A \cup B \cup C) = \phi$ (by I_2)

$\Rightarrow (A \cup B) = \phi$.

$(A \cup B) = \phi$ implies the computation has terminated. Therefore, the algorithm never detects a false termination. Further,

$(A \cup B) = \phi$

$\Rightarrow W_c + {}^{\cdot}{}_{W \in C} W = 1$ (by I_1)

•Since the message delay is finite, after the computation has terminated, eventually $W_c=1$.

•Thus, the algorithm detects a termination in finite time.

$W_c + \boxed{\sum_{W \in A \cup B \cup C} W} = 0$

$=1.$

$A \cup B \in .$

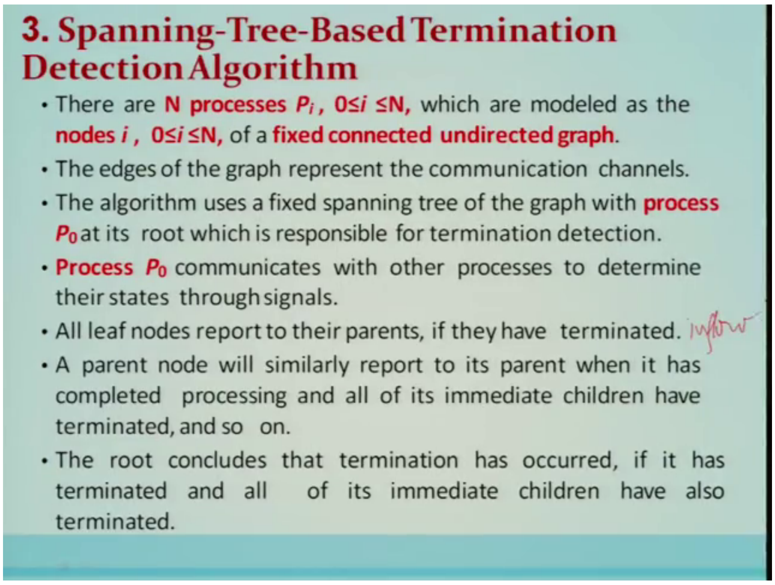
So, in variant I 1 states that the sum of the weights at the controlling process at all the active processes on all the basic messages and all the control message in transit is always equal to 1 in variant to state says the weight at, each active process on each basic message in transit and on each control messages in transit is non-zero.

Hence the weight on controlling agent is equal to 1 this will imply by the invariant 1 that the other part that is Wc plus the sum of the weights of the union A union, B union, C this part is equal to 0 why because this is equal to 1.

Now this will imply that A union B union C is equal to null by I 2 because all are non-zero. So, hence now since the communication channel is empty. So, that is why A union B is equal to 5 this implies that the computation has terminated therefore, the algorithm never detects a false termination further A union B is equal to phi that is Wc and Wc is equal to 1 by I 1, since the message delay is finite that is after the consideration estimated eventually Wc will become equal to 1.

Thus the algorithm permitted in a finite amount of time meaning to say that the control messages which carries the weights form the terminated process eventually reach to the controlling agent and the sum of all the weights will become 1 within a finite time.

(Refer Slide Time: 23:18)



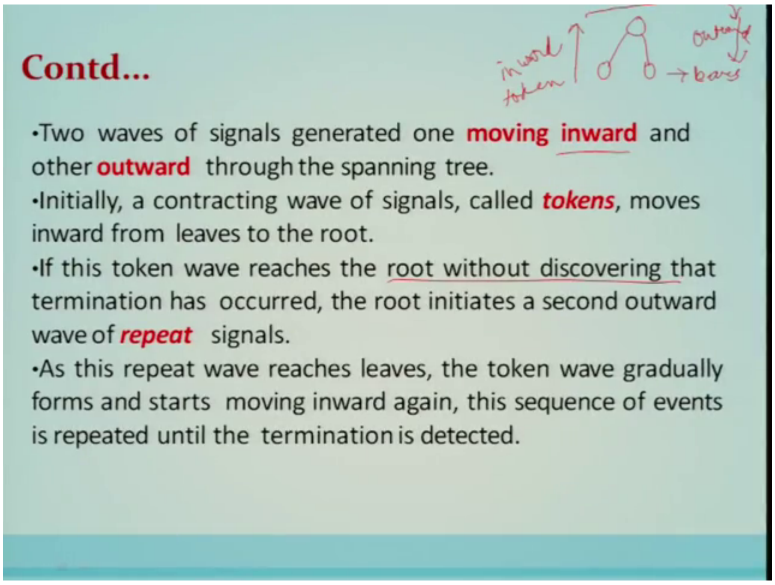
3. Spanning-Tree-Based Termination Detection Algorithm

- There are **N processes P_i, $0 \le i \le N$,** which are modeled as the **nodes i, $0 \le i \le N$,** of a **fixed connected undirected graph**.
- The edges of the graph represent the communication channels.
- The algorithm uses a fixed spanning tree of the graph with **process P_0** at its root which is responsible for termination detection.
- **Process P_0** communicates with other processes to determine their states through signals.
- All leaf nodes report to their parents, if they have terminated. inflow
- A parent node will similarly report to its parent when it has completed processing and all of its immediate children have terminated, and so on.
- The root concludes that termination has occurred, if it has terminated and all of its immediate children have also terminated.

Third Algorithm the Spanning-Tree-Based Termination Detection Algorithm, now we consider in this algorithm that there are N processes P i is which are modeled as the nodes of a fixed connected undirected graph. The edges of the graph represents the communication channel the algorithm use fixed the spanning tree of the graph with the process P 0 at the root which is responsible for the termination detection process P 0 communicates with the other process to determine their states through signals.

All the leaf node reports to their parents if they have terminated and this is called inflow, the parent node will similarly report to the parent when it has completed processing and all of it is immediate children are terminated and so on. The root concludes that termination has occurred if it has terminated and all of it is immediate children have been terminated.

(Refer Slide Time: 24:23)


inward
token
outward
→ leafs

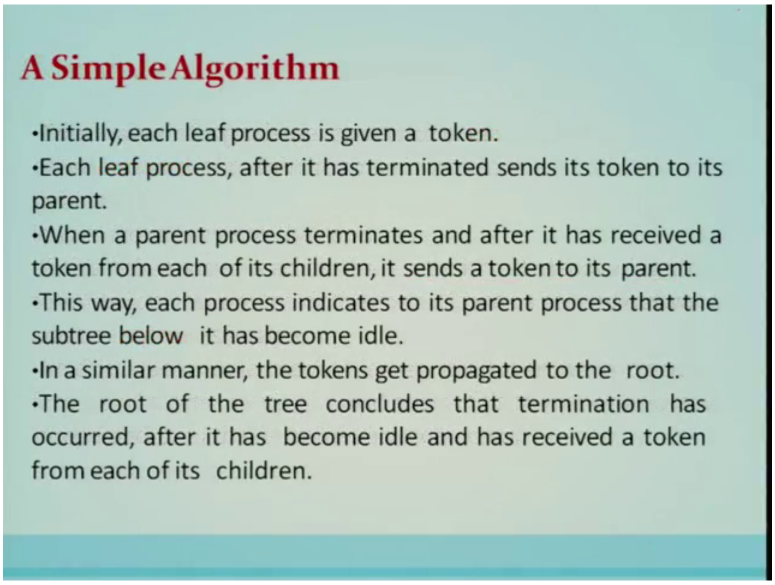
Contd...

- •Two waves of signals generated one **moving inward** and other **outward** through the spanning tree.
- •Initially, a contracting wave of signals, called ***tokens***, moves inward from leaves to the root.
- •If this token wave reaches the root without discovering that termination has occurred, the root initiates a second outward wave of ***repeat*** signals.
- •As this repeat wave reaches leaves, the token wave gradually forms and starts moving inward again, this sequence of events is repeated until the termination is detected.

Now, 2 waves of the signal generated one on moving inwards moving inwards in the sense this is a tree structure the leafs this is called leafs and when they terminated they will send the message and this is this direction is called moving inwards and moving outwards; that means, if the root sends the message down to the leaf then it is called outward initially a contracting wave of signals called token moves inwards from leaf to the root here.

Now, if this token wave reaches the root without discovering that the termination has occurred the root initiates the second the root initiates the second outward wave of request signals, at this repeat wave reaches the leaf and the token wave gradually forms and starts moving inwards again this sequence of event is repeated until the termination is detected.

(Refer Slide Time: 25:44)



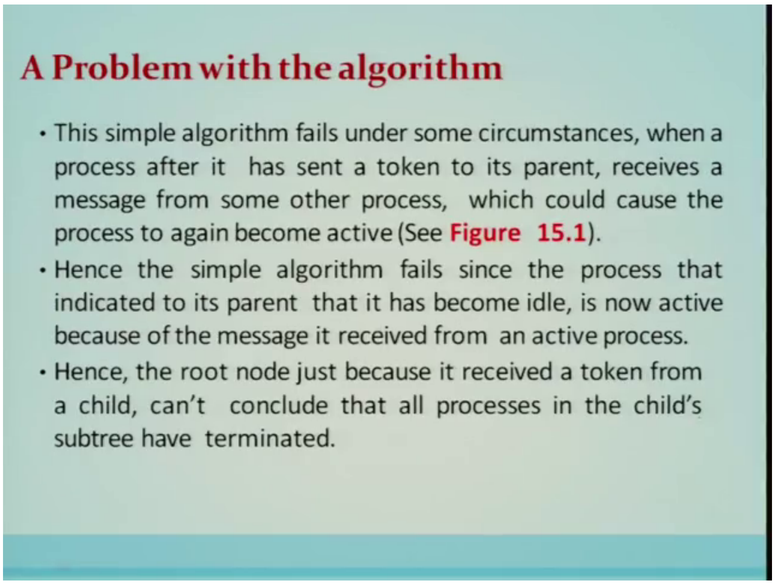
A Simple Algorithm

- Initially, each leaf process is given a token.
- Each leaf process, after it has terminated sends its token to its parent.
- When a parent process terminates and after it has received a token from each of its children, it sends a token to its parent.
- This way, each process indicates to its parent process that the subtree below it has become idle.
- In a similar manner, the tokens get propagated to the root.
- The root of the tree concludes that termination has occurred, after it has become idle and has received a token from each of its children.

Simple Algorithm which we will first understand and then we will see the problems and we will using these problems we will see the final algorithm.

So, the simple strategy is that initially the leaf nodes are given the token. So, they are given token let us say t 1 and t 2. Each leaf process, after termination sends it is token to it is parent, where the parent process terminates and after it has received the token from each of the children; it sends their token to it is parent. This way each process indicates to exponent that the sub tree below is idle. In similar manner the tokens get propagated to the root. So, when it gets propagated to the root the root will contain that particular token and this concludes that the entire computation has terminated after it has become idle and has received the token from each of it is children.

(Refer Slide Time: 26:56)



A Problem with the algorithm

- This simple algorithm fails under some circumstances, when a process after it has sent a token to its parent, receives a message from some other process, which could cause the process to again become active (See **Figure 15.1**).
- Hence the simple algorithm fails since the process that indicated to its parent that it has become idle, is now active because of the message it received from an active process.
- Hence, the root node just because it received a token from a child, can't conclude that all processes in the child's subtree have terminated.

Now, the problem exists in this particular algorithm the simple algorithm fails under the some situation when the process after it has sent a token to it is parent which indicates that the that that particular process is idle, but again receives a message from some other process.

So, once it has received a message from it is process then the idle state again will go to a tournament state, but it is parent is knowing that that particular process is idle which could cause the process to again become idle and that is shown in figure 15.1.

(Refer Slide Time: 27:36)

An Example of the Problem

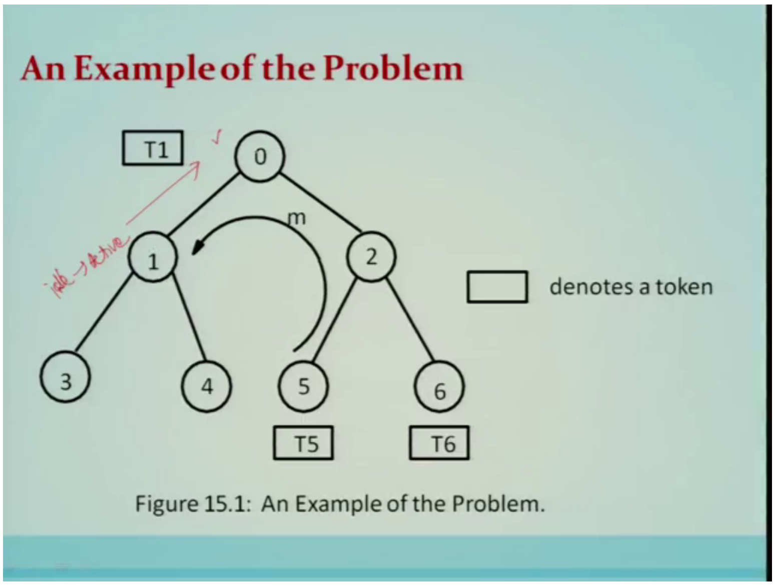
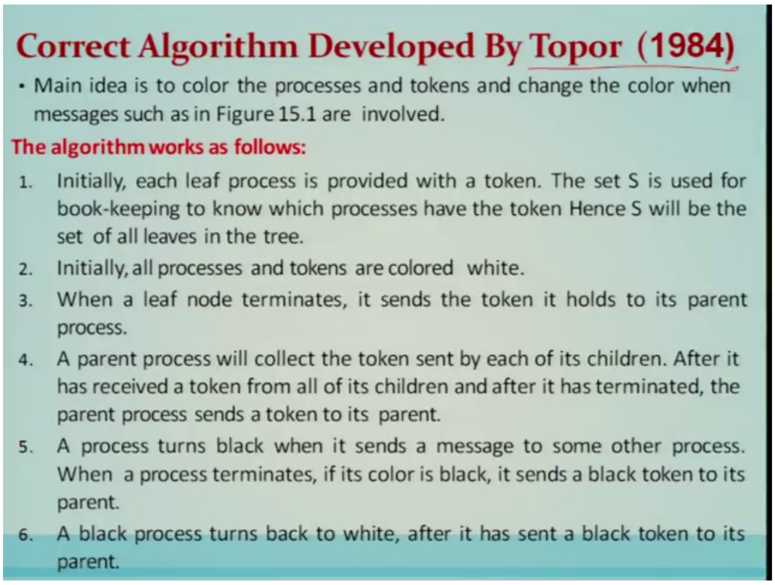


Figure 15.1: An Example of the Problem.

In 15.1 you can see this particular situation that after 1 has given it is token 2 the 2 it is parent the goal number 5 has sent a message back to 1. So, 1 will again become from idle it will become an active, but this parent knows that 1 is idle which is going to make the contradiction.

(Refer Slide Time: 28:17)



Correct Algorithm Developed By Topor (1984)

- Main idea is to color the processes and tokens and change the color when messages such as in Figure 15.1 are involved.

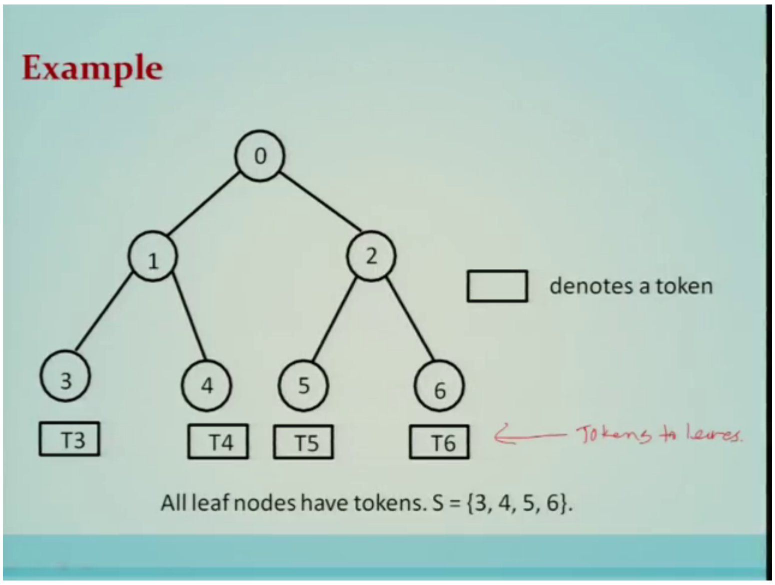
The algorithm works as follows:

1. Initially, each leaf process is provided with a token. The set S is used for book-keeping to know which processes have the token Hence S will be the set of all leaves in the tree.
2. Initially, all processes and tokens are colored white.
3. When a leaf node terminates, it sends the token it holds to its parent process.
4. A parent process will collect the token sent by each of its children. After it has received a token from all of its children and after it has terminated, the parent process sends a token to its parent.
5. A process turns black when it sends a message to some other process. When a process terminates, if its color is black, it sends a black token to its parent.
6. A black process turns back to white, after it has sent a black token to its parent.

So, this kind of problems is now corrected and the final algorithm is given by the Topor in 1984. So, the main idea is to color the processes and the tokens and change the color when the messages such as we have shown in figure 15.1 are involved the algorithm works as follows initially each leaf is provided with the token the set S is used for the book keeping to know which processes have the token hence S will be the set of all leads in the tree. Initially, all processes and the token are colored white when a leaf terminates it sends the token it holds to the parent process

A parent process will collect the token sent by each of it is children after it has received token from all of it is children and after it has terminated the parent process since a token to it is parent. A process turns black when it sends the message to some other process when a process terminates it if it is color is black it sends black token to it is parent black process turns white after it has sent a black to it is parent.

(Refer Slide Time: 29:31)



Example

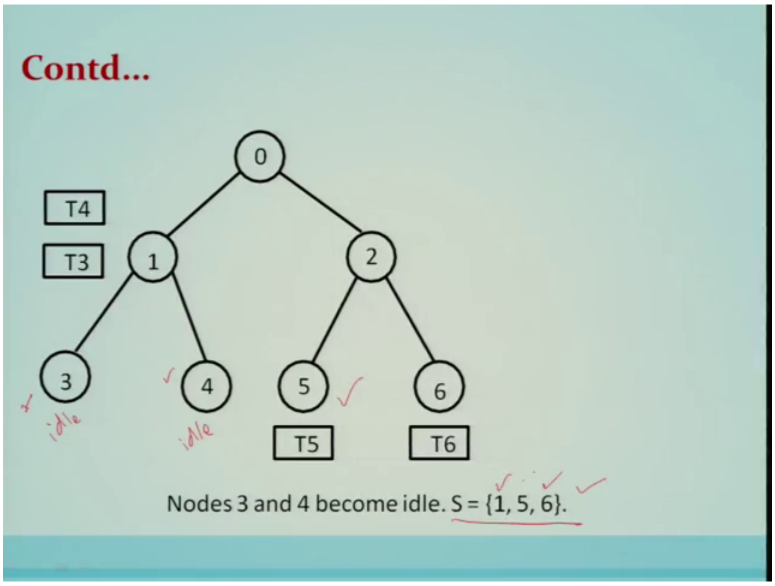
□ denotes a token

Tokens to leaves.

All leaf nodes have tokens. S = {3, 4, 5, 6}.

So, let us understand the entire algorithm through this particular example. Now as in the algorithm you have seen that all the leaf nodes are given the tokens initially Token to the leaps now when these leafs finishes their computation they will send the token to the parent.

(Refer Slide Time: 30:11)



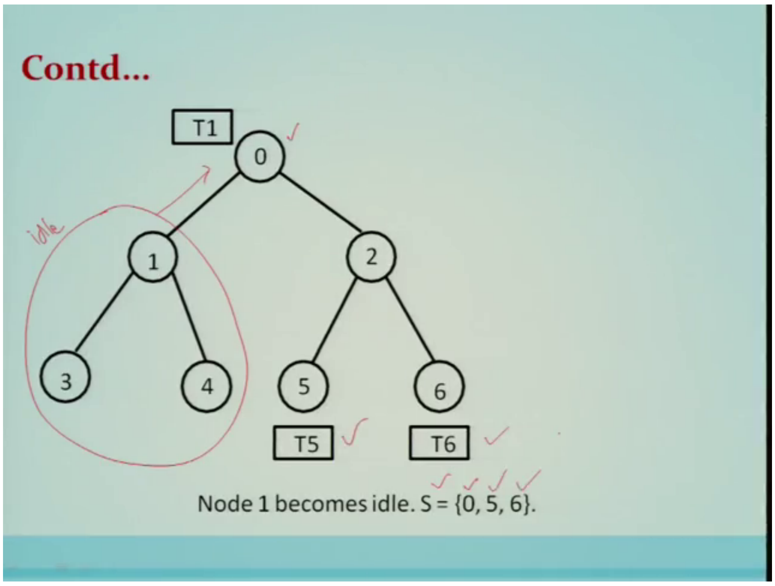
Contd...

Nodes 3 and 4 become idle. S = {1, 5, 6}.

So, here you can see that this particular node and this particular leaf they have finished their computation. So, they have sent their token to it is parent. So, this will become an idle.

And the set S initially which was having 3 4 5 6 because they were having token now the active set of processes will be the process 1 then 5 because it is having token and then 6 and Node 3 and 4 they have become idle as far as the working of the console algorithm.

(Refer Slide Time: 30:43)

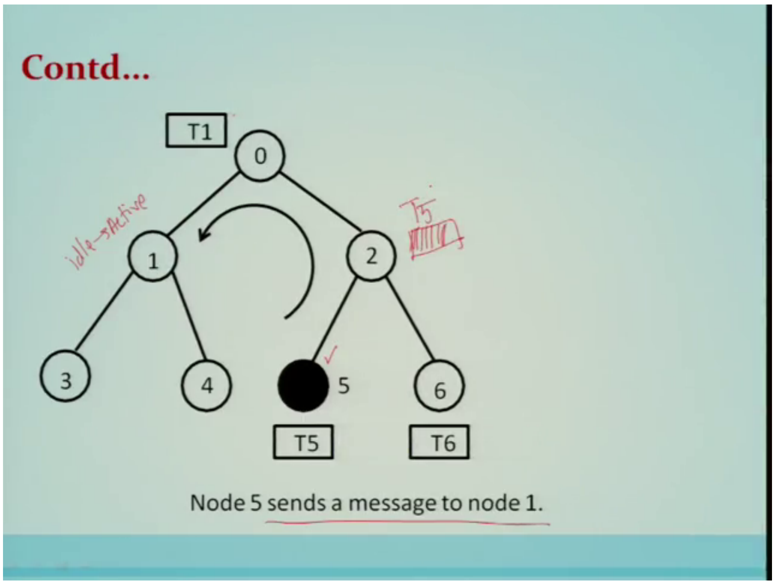


Contd...

Node 1 becomes idle. S = {0, 5, 6}.

Now this one has received the token from both of his children. So, it will send this particular token to it is parent, t 1 will send it is token t 1 to the parent. So, now, the parent will know that the underlying sub tree is terminated, that is it is idol and the set of active states s will be now the Node 0, Node 5 which is having token and node 6 now the algorithm will monitor them.

(Refer Slide Time: 31:12)



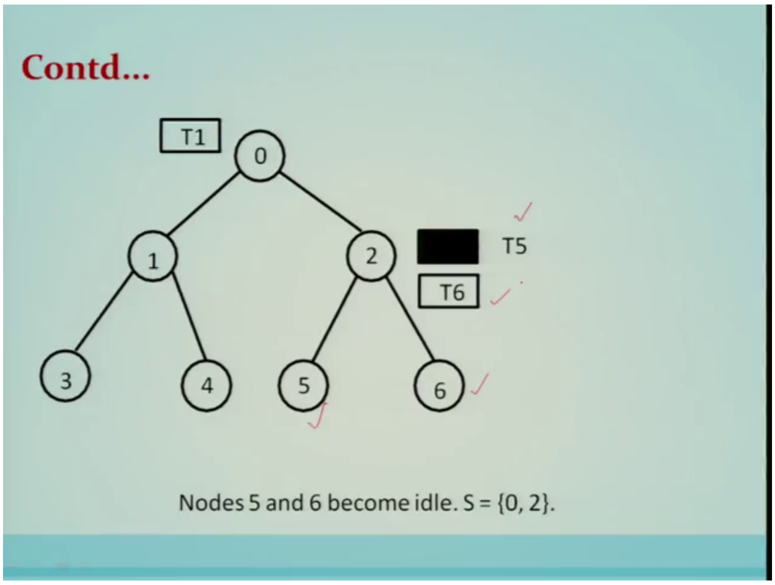
Contd...

Node 5 sends a message to node 1.

Meanwhile Node 5 has generated a message for the Node 1, now Node 1 from idle state it will become an active state after receiving this particular message although this token represents that 1 is active.

So, the node which sends this particular message that node will become black in this case and once this particular node which is black finishes it is execution and it is idle then it will send the black token to it is parent and become white then in that case.

(Refer Slide Time: 32:01)

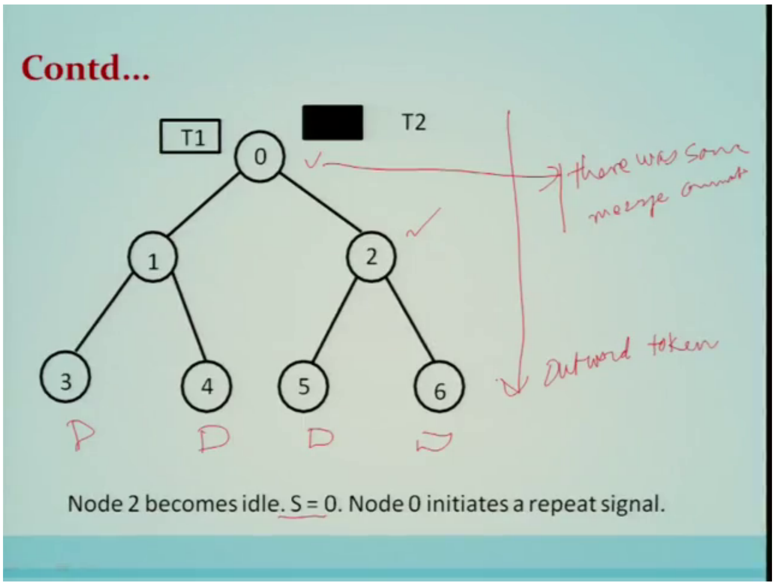


Contd...

Nodes 5 and 6 become idle. S = {0, 2}.

So, here the Node 5 becomes white, but it has send a token that is T 5 is a black token similarly 6 has send a s token to this parent. Now the active states are 0 and 2 here in this case.

(Refer Slide Time: 32:23)



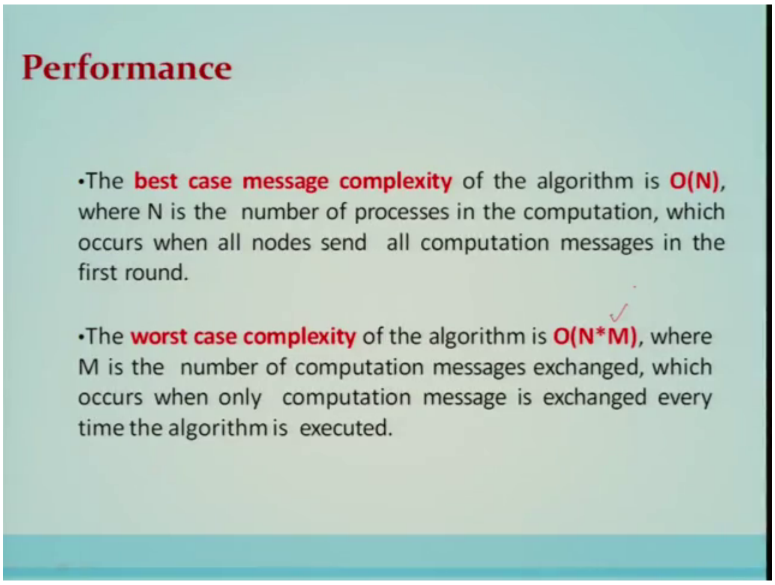
Contd...

Node 2 becomes idle. S = 0. Node 0 initiates a repeat signal.

Now, in this black token finally, reaches to the Node 0 and Node 2 become idle now S the activates states is only S. Now Node S will Node 0 will understand by receiving the black token that that there is some message communication. Hence the distributor computation is not terminated. So, by taking this black token Node 0 or a root will again send the outward token and this outward token will propagate the token down to the leafs and restart the entire process again.

So, in the second round if let us say that if it is if the computation is terminated without any message exchange the root may get all white tokens back and this will be the terminated state. If in the second round also if it receives a black token then again this particular process iterates till it receives a white token and that is the indication that the mess that the algorithm is terminated.

(Refer Slide Time: 34:02)



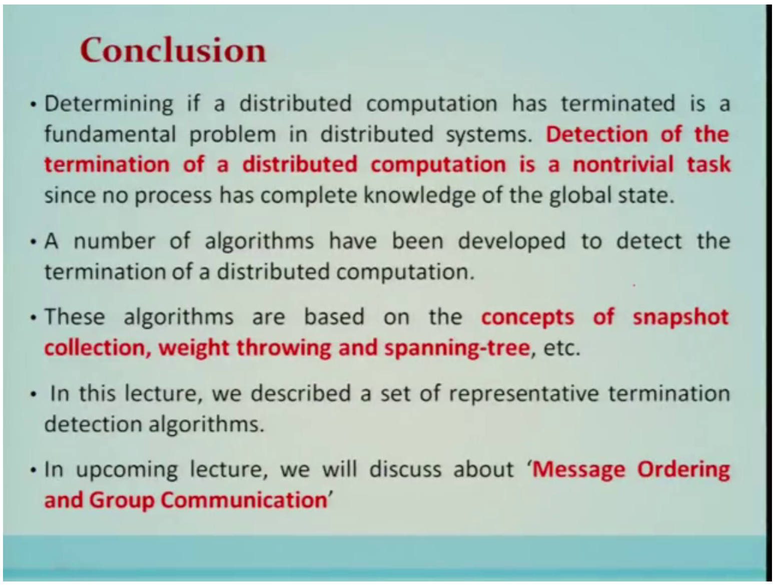
Performance

- The **best case message complexity** of the algorithm is **O(N)**, where N is the number of processes in the computation, which occurs when all nodes send all computation messages in the first round.

- The **worst case complexity** of the algorithm is **O(N*M)**, where M is the number of computation messages exchanged, which occurs when only computation message is exchanged every time the algorithm is executed.

Now, the performance the best case message complexity of this algorithm is of the order N, where N is the number of processes in the computation. Which occurs when all the nodes sends all the computation messages in the first round worst case complexity of this algorithm is of the order N times M. M is the number of computation messages exchanged. So, this if only one message is exchanged then basically it will 2 times 3 times and so on. So, number of times this algorithm iterates outwards inwards it depends upon the number of messages.

(Refer Slide Time: 34:45)



Conclusion

- Determining if a distributed computation has terminated is a fundamental problem in distributed systems. **Detection of the termination of a distributed computation is a nontrivial task** since no process has complete knowledge of the global state.
- A number of algorithms have been developed to detect the termination of a distributed computation.
- These algorithms are based on the **concepts of snapshot collection, weight throwing and spanning-tree**, etc.
- In this lecture, we described a set of representative termination detection algorithms.
- In upcoming lecture, we will discuss about '**Message Ordering and Group Communication**'

Conclusion determining if the distributed computation has terminated is a fundamental problem in distributed systems. The detection of termination of a distributed computation is a nontrivial task since no process has the complete knowledge of the global state and also the distributed system does not have common clock. So, a number of algorithms we have seen have been developed to detect the termination of a distributed computation why because it is going to be used in many applications these algorithms are based on the concepts which we have seen here the representative algorithms based on these concepts of snapshot collection weight throwing and a spanning tree.

So, in this lecture we have described a set of representative termination detection algorithm in the upcoming lecture we will discuss about message ordering and group communication.