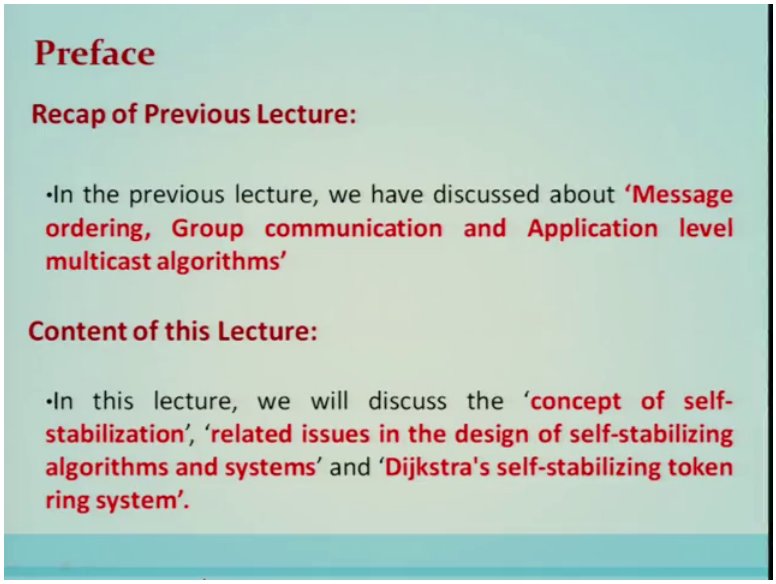
Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 17
Self-stabilization

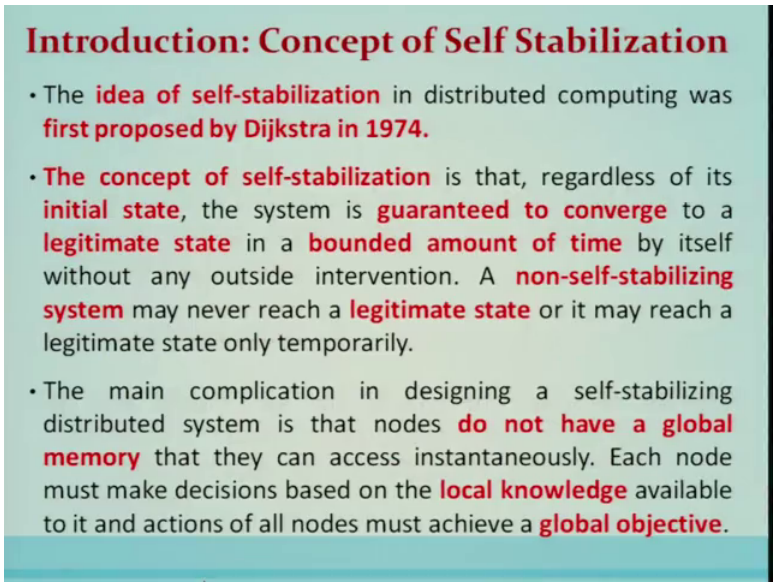
Lecture 17: Self-stabilization. Preface: recap of previous lecture.

(Refer Slide Time: 00:22)



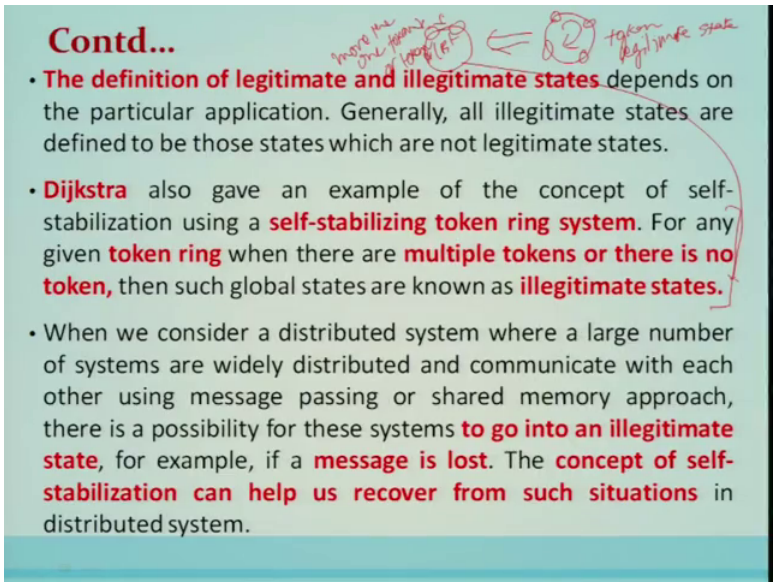
In previous lecture we have discussed about message ordering, group communication and application level multicast algorithms. Content of this lecture: in this lecture we will discuss the concept of self-stabilization, related issues in the design or self-stabilizing distributed algorithms and systems, and Dijkstra's self-stabilizing token ring system.

(Refer Slide Time: 00:50)



Introduction: concept of self-stabilization: the idea of self-stabilization in distributed computing was first proposed by Dijkstra in 1974. The concept of self-stabilization is that, regardless of its initial state the system is guaranteed to converge to a legitimate state in a bounded amount of time by itself without any outside intervention. So, the non-self-stabilizing system, may never reach the legitimate state, or it may reach a legitimate state only temporarily. The main complication designing self-stabilizing distributed system is that, nodes do not have the global memory that they can access instantaneously. Each node must make decision based on the local knowledge available to it, and actions of all the node must achieve the global objective.

(Refer Slide Time: 01:54)



The definition of legitimate and illegitimate state depends upon the particular application. Generally all illegitimate states are defined to be those states which are not legitimate. Dijkstra also give an example of the concept self-stabilization, using self-stabilizing token ring system; that is called Dijkstra's self-stabilizing token ring system.

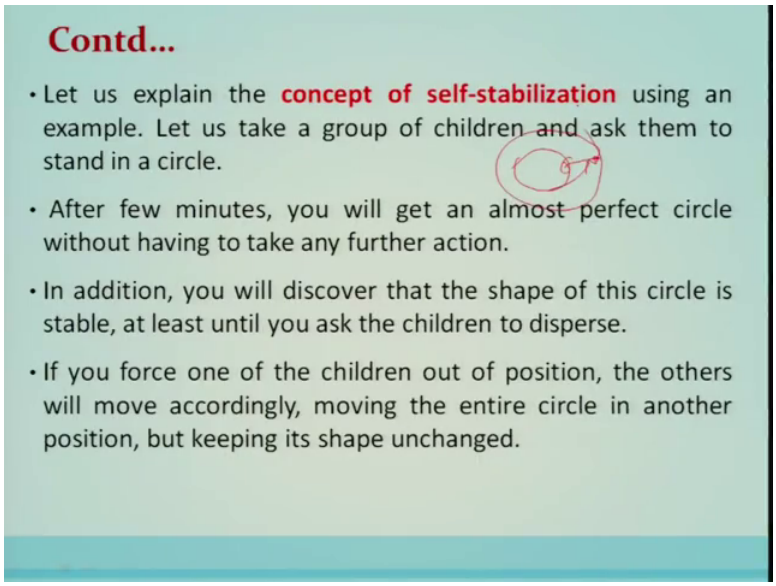
So, for any token ring, when there are multiple tokens or there is no token, then such a global state are known as illegitimate state. When we consider distributed systems, where a large number of systems are widely distributed and communicate with each other using message passing or shared memory approach, there is a possibility for these systems to go into an illegitimate state. For example, if a message is lost, the concept of self-stabilization can help us recover from such situation in a distributed system; so again before going ahead. So, Dijkstra gave an example of a token ring system. In a token ring system you know that, only one token is called privilege which circulates, and if it is circulates then it is legitimated state in the system.

Now, in contrast to this, if this particular token has, this particular token ring has two privileges, or two tokens, or more than one token, or it does not have a token at all, or a token is lost. This particular situation is illegitimate state as far as this definition is concerned. So, this particular state which is illegitimate state, if it goes to a legitimate state automatically without external intervention, then it is called a self-stabilizing system. If it is a token ring system, then it is called self-stabilizing token ring system.

In distributed systems you are seeing that lots of processors are connected through the communication network, and they exchange through the message communication. So, there is a possibility that nodes may fail down or the messages may lost; obviously, these conditions will basically lead to a illegitimate state in the distributed system. How automatically, how the self-stabilizing distributed system will recover? Automatically to a legitimate state is basically a design issue which Dijkstra has opened through this particular system. That is called a self-stabilizing system.

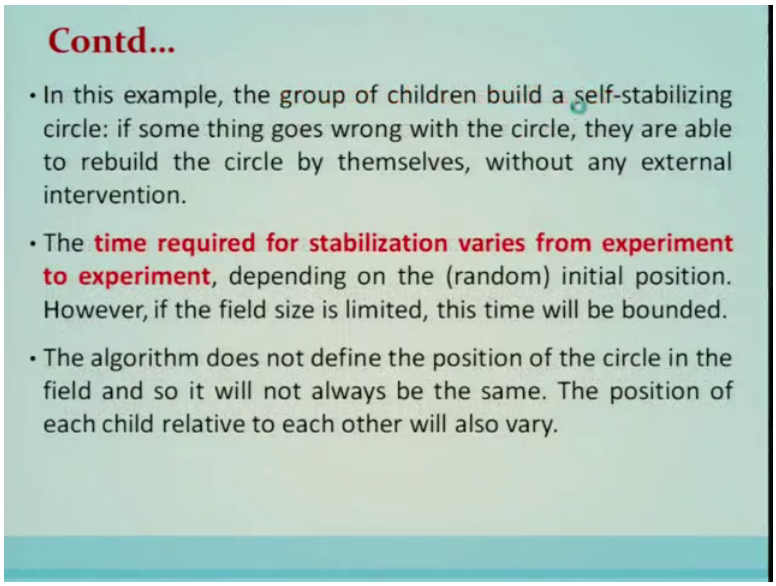
So, the concept of self-stabilizing is very useful to understand about the self-stabilizing system, and the design for different distributed systems, based on this particular concept.

(Refer Slide Time: 05:01)



So, let us explain the concept of self-stabilization using an example. Let us take a group of children and ask them to stand in a circle. So, after a few minutes you will see almost a perfect circle without having to take any further action. In addition you will also discover that the shape of this circle is stable, at least until you asked the children to disperse. If you force one of the children's out of this particular position, the others will move accordingly and they will form a bigger circle.

(Refer Slide Time: 05:44)

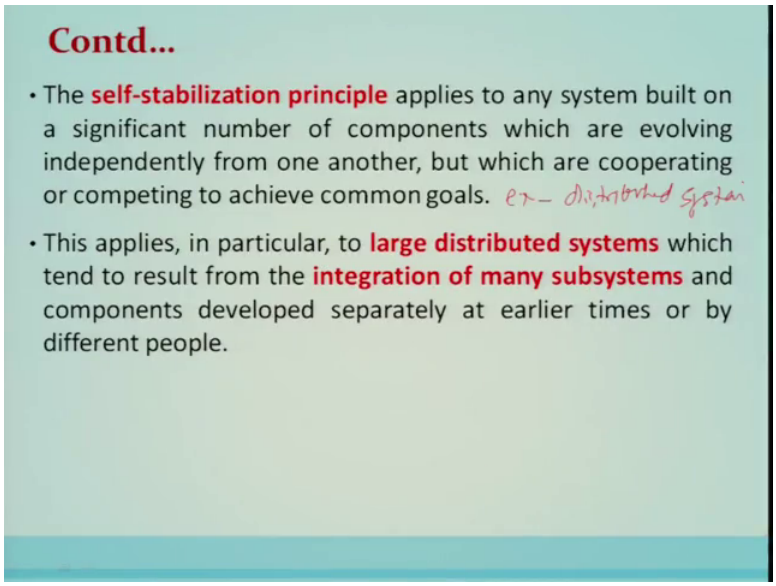


So, keeping the shape of the circle unchanged in all the cases. So, in this example the group of children's, children build a self-stabilizing circle. So, if something goes wrong with the circle, they are able to rebuild the circle by themselves without any external intervention. So, we have seen a example of a self-stabilizing circle, and by a group of children, and the legitimate state and illegitimate state, all these things we have seen.

So, this particular example motivates us to understand about self-stabilization in different type of systems, especially in a distributed system. Now the time required for stabilization varies from experiment to experiment, depending on random or the initial position; however, if the field size is limited in the case of, the children building a circle is limited this particular time will be bounded.

The algorithm does not define the position of the circle in the field. So, it will not always be the same. The position of each child relative to each other will also vary.

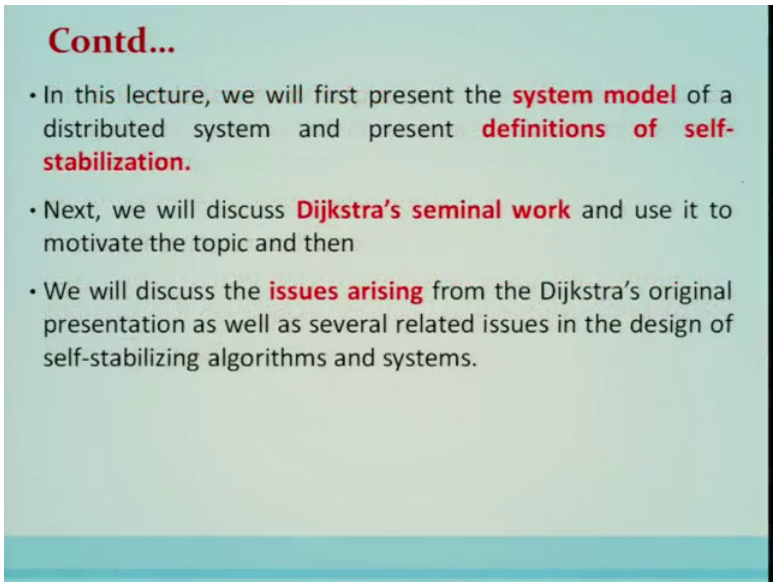
(Refer Slide Time: 07:06)



So, there are several factors, meaning to say that particular factors are going to basically take how much time the system is going to stabilize. So, the self-stabilization, principle applies to any system build on a significant number of components, which are evolving independently from one another, but which are cooperating or competing to achieve a common goal and example is the distributed system.

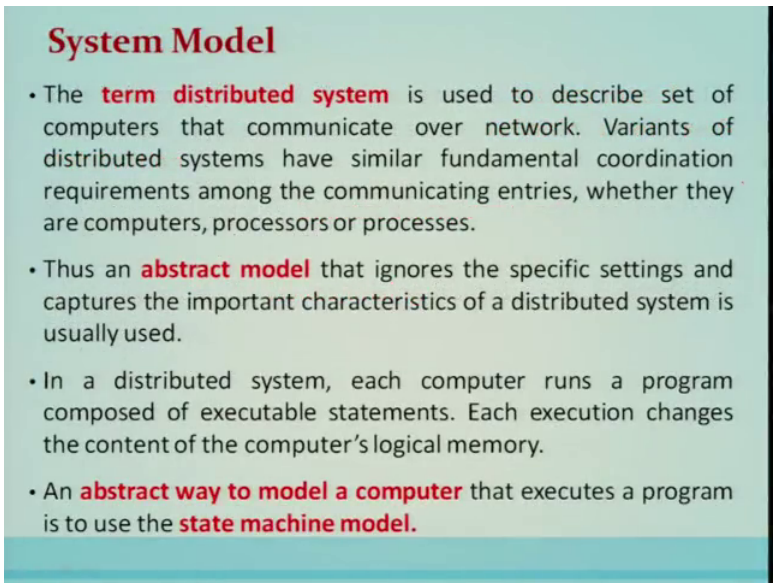
So, this applies in particular to large distributed systems, which tend to result from integration of many subsystems, and components developed separately, at the earlier times or by different people.

(Refer Slide Time: 07:54)



So, in this lecture we will first present the system model of distributed system and present the definitions of self-stabilization. Next we will discuss Dijkstra's seminal work and use it to motivate, the topic. And then we will discuss by issues arising from Dijkstra's original presentation, as well as several related issues in the design of self-stabilizing algorithms and systems.

(Refer Slide Time: 08:20)

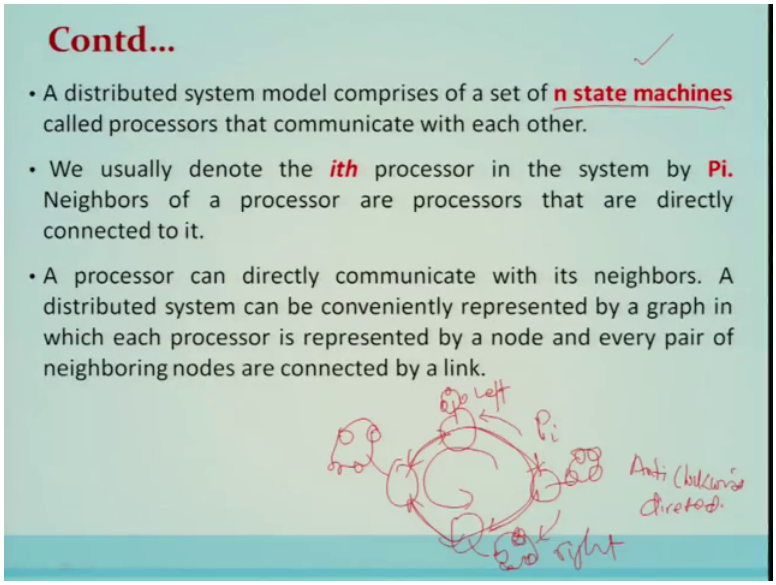


Let us see the system model. The term distributed system is used to describe the set of computers that communicate over the network, variants of distributed systems have

familiar fundamental coordination requirements among communicating entities, whether they are computer processor or processes.

Thus an abstract model that ignores the specific setting, and captures the important characteristics of a distributed system is usually used. In a distributed system each computer run a program composed of executable statement. Each execution changes the content of the computers logical memory. And abstract way to model a computer that executes a program is to use, state machine model.

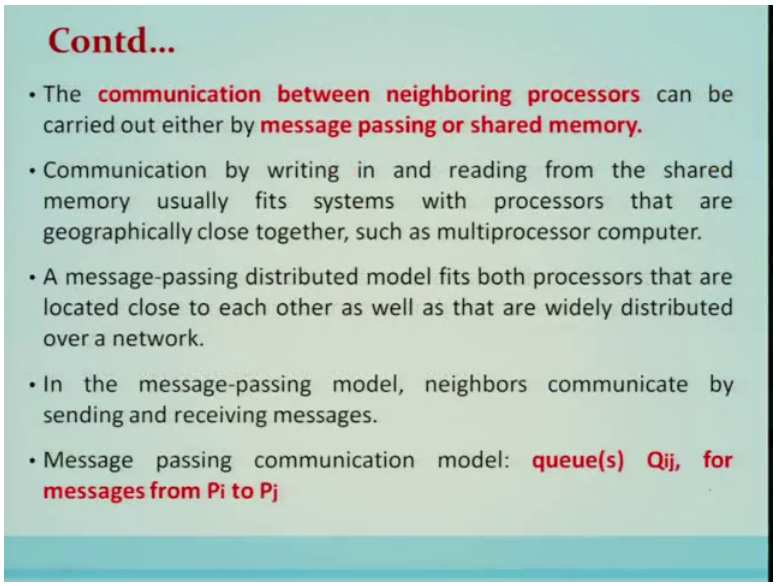
(Refer Slide Time: 09:06)



A distributed system model comprises of n state machines called processors that communicate with each other. So, each processor is nothing, but a state machine, and these particular processors will communicate with each other also, and these processor will make a transition among these states. So, the example here is basically comprising a set of n state machines, called processors which will communicating with each other. Usually denoted i-th processor denoted by P I, the neighbor of processor are the processors. So, here we are calling it as, the left and the right neighbors, and the system we are considering to be in a anticlockwise directed.

A processor can directly communicate with its neighbors, a distributed system can be conveniently represented by a graph, in which each processor is represented by a node, and every pair of neighboring nodes are connected by a link.

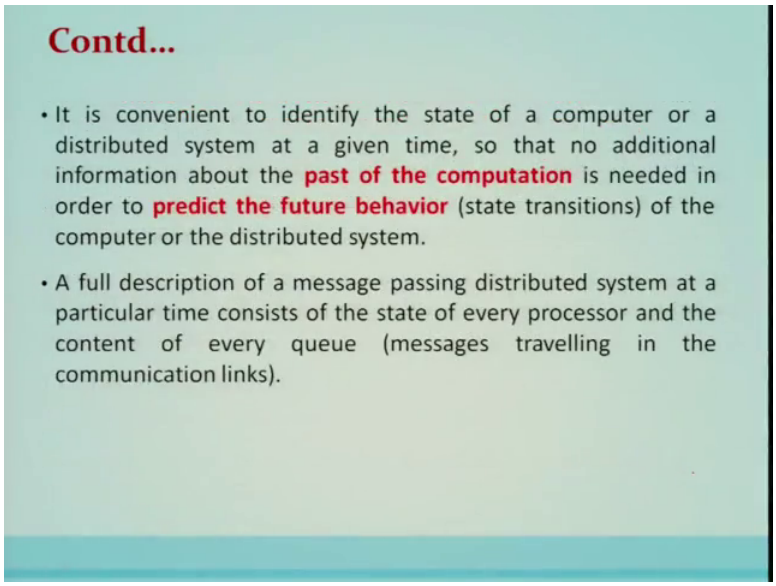
(Refer Slide Time: 10:46)



That I have shown in the figure. The communication between neighboring processors can be carried out, either by message passing or shared memory, communication by writing in and reading from the shared memory usually fits the system with processors that are geographically close together; such as multiprocessor computers.

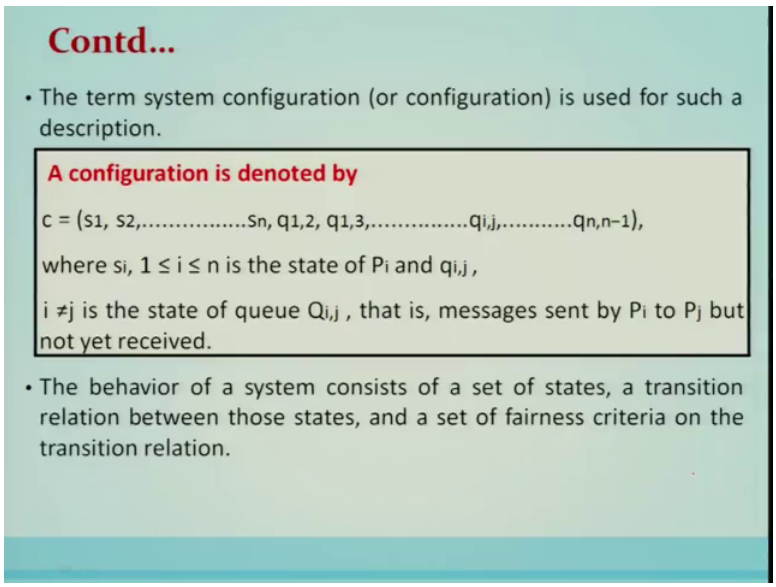
Message passing distributed system, distributed model fits both processors that are located close to each other, as well as they are widely geographically distributed, and they are connected over a network. In the message passing model neighbors communicate by sending and receiving messages. Message passing communication model will should contain a queue, and which is represented as Qij for the messages from Pi to pj.

(Refer Slide Time: 11:36)



It is convenient to identify the state of a computer or a distributed system at a given time. So, that no additional information about the past of the computation is needed in order to predict the future behavior, of the computer or a distributed system. A full description of the message passing distributed system at a particular time consists of the state of every processor and content of every queue.

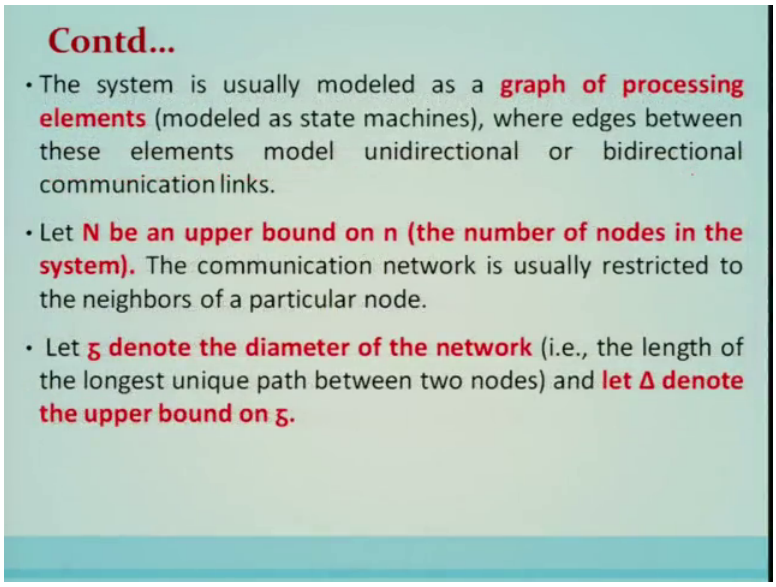
(Refer Slide Time: 12:01)



The term configuration, or a configuration is uses that; such a description configuration is noted by a set that is c, which contains the states where Si is the state of a P i and Q i j,

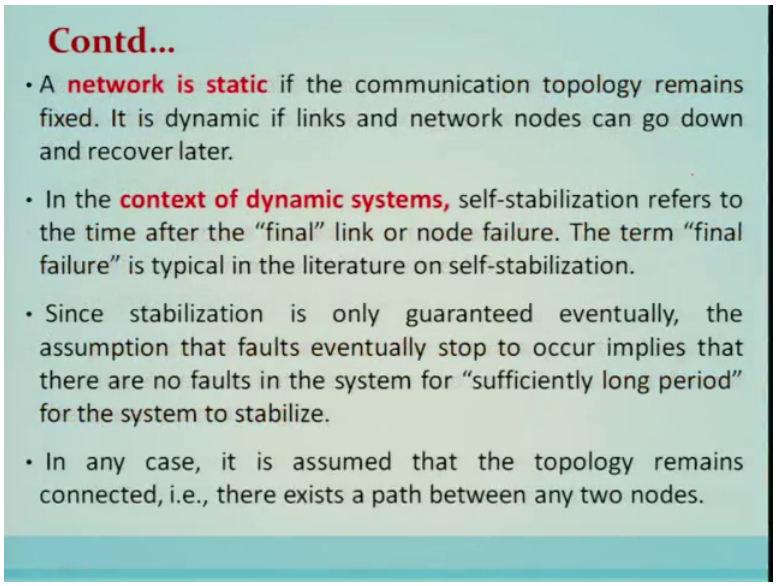
where i is not equal to j is a state of a Q; that is the messages sent by Pi to Pj, but not yet received. The behavior of a system consists of a set of states, a transition relation between those states and a set of fairness criteria on the transition relation.

(Refer Slide Time: 12:34)



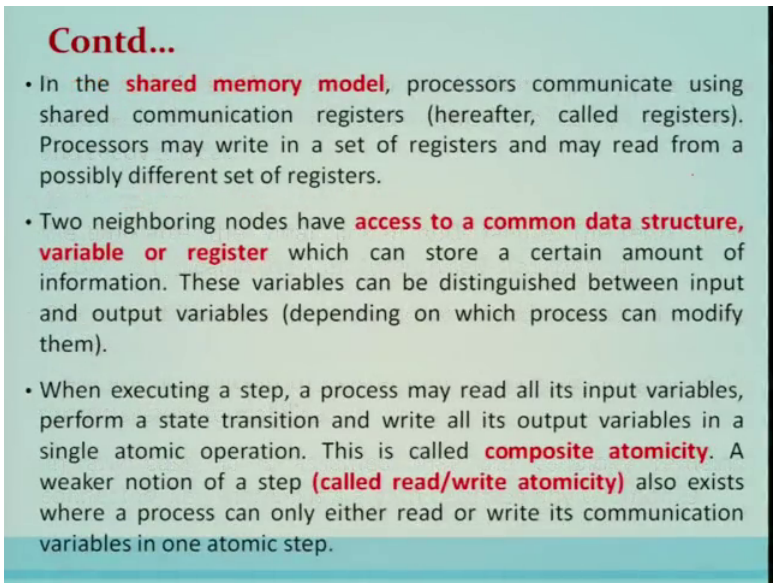
The system is usually modeled as the graph of processing elements, where edges between the elements model the unidirectional or bidirectional communication links that I have already explained. Let N be the upper bound on n that is the number of nodes in the system, the communication network is usually restricted, to the neighbors of the particular node. So, here the diameter and the big delta, denotes the upper bound on that diameter.

(Refer Slide Time: 13:04)



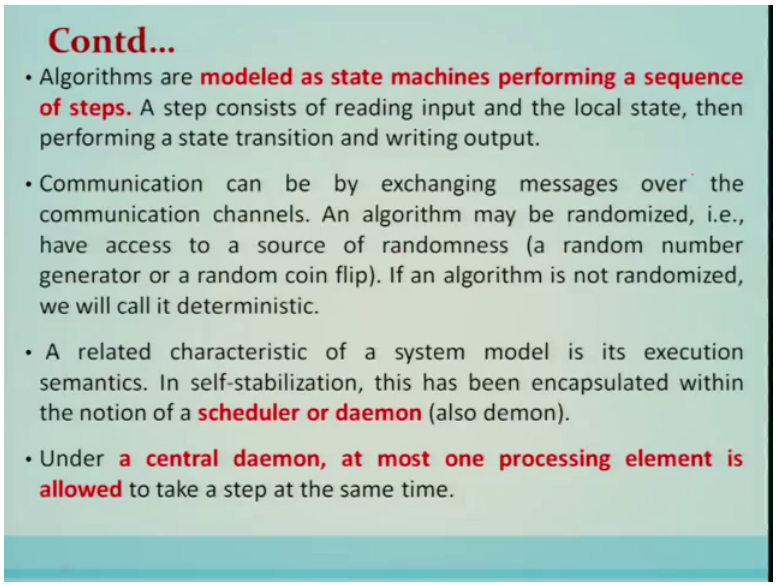
A network is static, if the topology remains fixed, dynamic the links and networks can go down and recover later on. So, self-stabilization is guaranteed eventually, in spite of all the faults. Shared memory model, is basically we are not going to consider two neighboring nodes, having access to a common data structure variable is not possible in distributed systems.

(Refer Slide Time: 13:23)



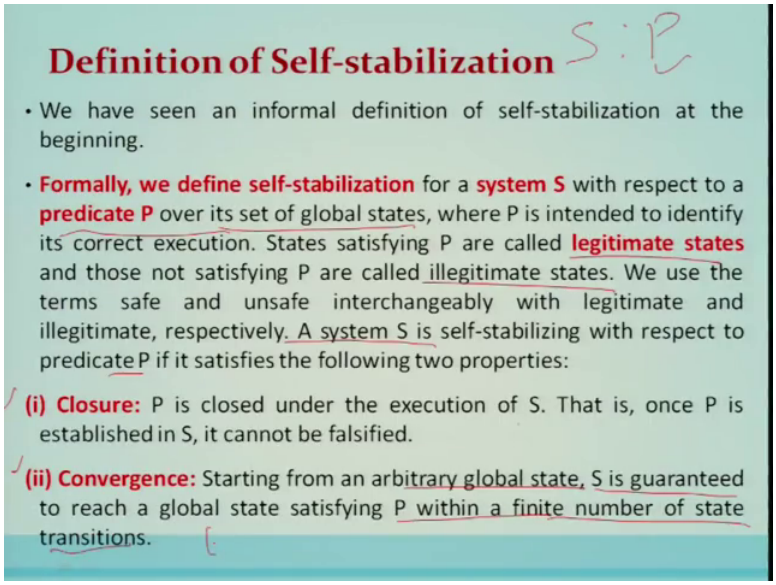
So, the algorithms are modeled as the state machine, performing the sequence of steps. A step consists of reading input and the local state.

(Refer Slide Time: 13:35)



And then performing the state transition and writing output. Communication can be by exchanging messages over a communication channel. So, a related characteristics of a system model is, the execution semantics, if the self-stabilization. This has encapsulated within the notion of scheduler or a daemon, also called demon one under a central daemon at most one processing element is allowed to take i step at a point of time.

(Refer Slide Time: 14:11)

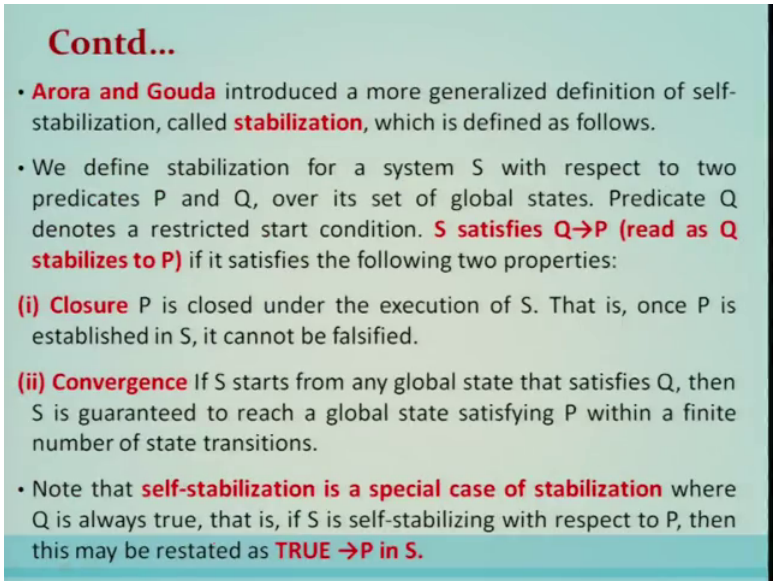


So, we will see that, these particular assumptions are basically well defined for a particular self-stabilizing system, definition of self is stabilization. We have seen an

informal definition of a self-stabilization at the beginning. Formally, we define self-stabilization for a system S with respect to a predicate P, or its set of global states, where P is intended to identify its correct execution. So, the states satisfying the predicate, P is called the legitimate state, and those not satisfying the predicate p are called illegitimate state. We use the term safe and unsafe interchangeably with the legitimate and illegitimate respectively. So, a system S is self-stabilizing with respect to the predicate P, if it satisfied the following two properties, which are most important.

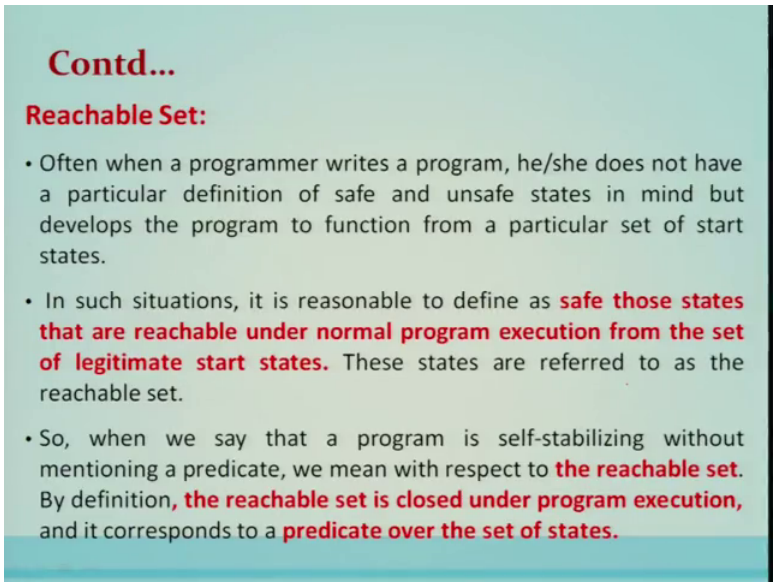
So, the first property is called the closure property; says that P is closed under the execution of S; that is once P is established in S, it cannot be falsified. Second one is called convergence starting from an arbitrary global state, the predicated system defined by P; that is called S is guaranteed to reach a global state satisfying P within a finite state of transitions. So, the closure and convergence are two important properties in that self-stabilization.

(Refer Slide Time: 15:36)



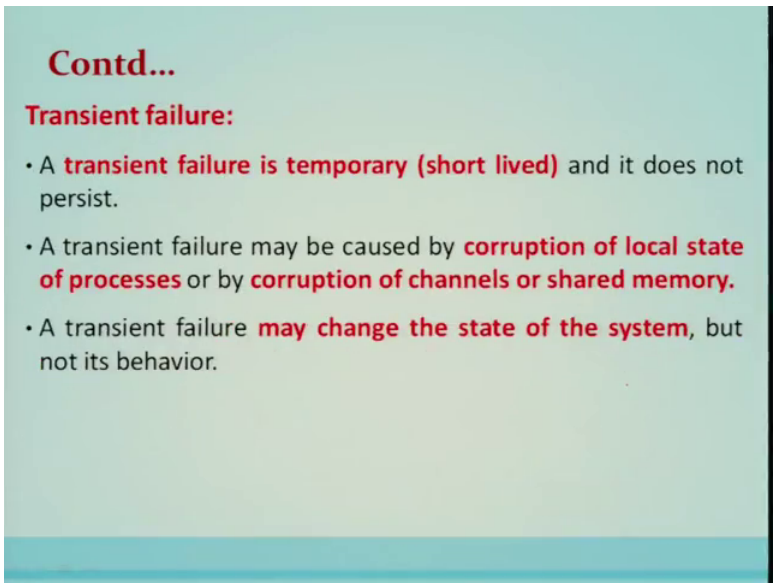
So, we define the self-stabilization or stabilization for a system S, and self-stabilization is basically a part of the stabilization. So, self-stabilization is a special case of stabilization.

(Refer Slide Time: 15:54)



Now, then there is a reachable set, often when the programmer writes a program, he she does not have a particular definition of a safe and unsafe states in mind, but develops a program to function from a particular set of start states. Such situation it is reasonable to define, as states those states that are reachable under normal program execution from the set of legitimate start states.

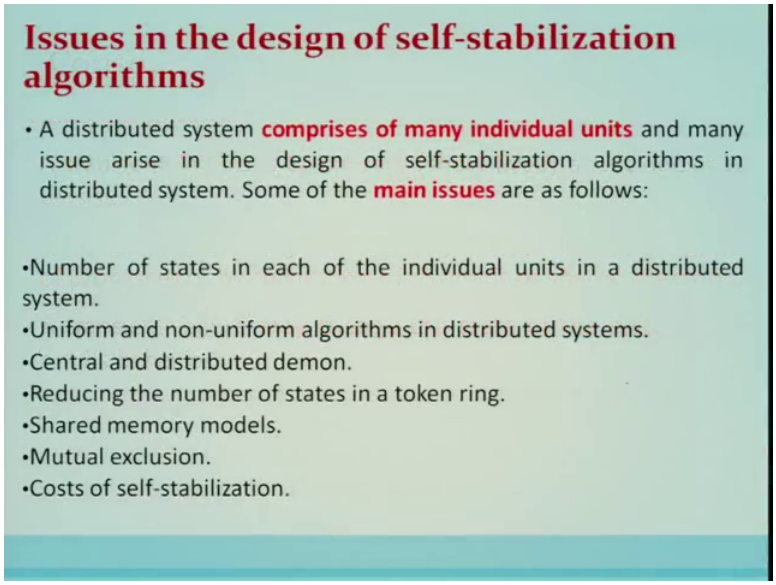
(Refer Slide Time: 16:21)



These are referred to as reachable sets. Transient failure set. Transient failure is temporary or short lived and does not persist. The transient failure may be caused by

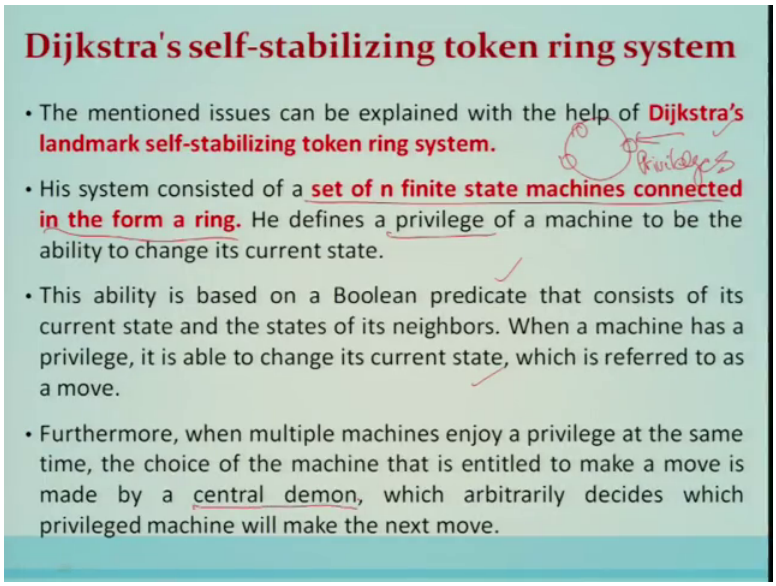
corruption of local state processes, or by corruption of channel or shared memory. Transient failures may change the state of a system, but not its behavior.

(Refer Slide Time: 16:35)



Issues in design of self-stabilization algorithms: a distributed system comprises of many individual units, and many issue arise in the design of self-stabilization, algorithms in distributed system some of the main issues, are number of states in which each of the individual unit in a distributed system, uniform and non-uniform algorithms, central and distributed demon, reducing the number of states in a token ring, shared memory models mutual exclusion and cost of self-exploration.

(Refer Slide Time: 17:10)



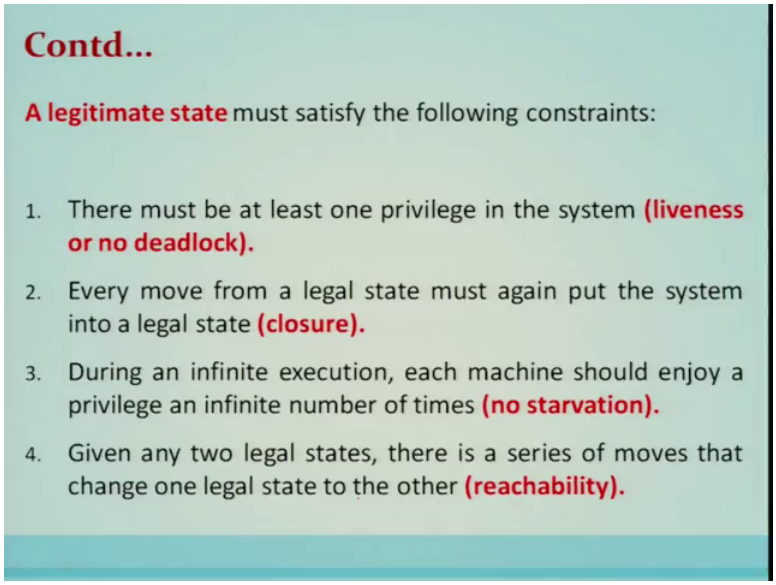
The mentioned issues can be explained with the help of Dijkstra's landmark self-stabilizing token ring system. So, his token ring system consisted of set of n finite state machine connected in a form of a ring. I told you previously, and he defines the privilege of a machine, to be, the ability to change the, to change its state. So, that particular node, which is called a privilege node, has the ability to change its state, and Dijkstra assumes initially that there is only one privilege at a particular point of time, but later on its change the model, and we are going to see all both models in this part of the discussion.

So, this ability is based on the Boolean predicate that consists of the current state and the state of its neighbor. So, when a machine has a privilege, it is able to change its current state, which is referred to as a move. So, furthermore when multiple machines enjoy a privilege at a same time, the choice of the machine that is entitled to make a move is made by a central demon, which arbitrarily decides which privilege machine will make a move. So, these are the important concepts which I will again highlight before going ahead.

So, first thing is, in a given self-stabilizing in a system, there must be some set of privileges, and these privileges are subject to the Boolean predicate, and it will change the state from illegitimate state to the legitimate state, following two rules which we have seen; closure and the convergence. So, furthermore when multiple machines, they enjoy more than one machines having a privilege at the same point of time, then the central

demon will come, and this will decide, among many privileges, one of these privileges will be activated, or will be allowed to make a boom at particular point of time.

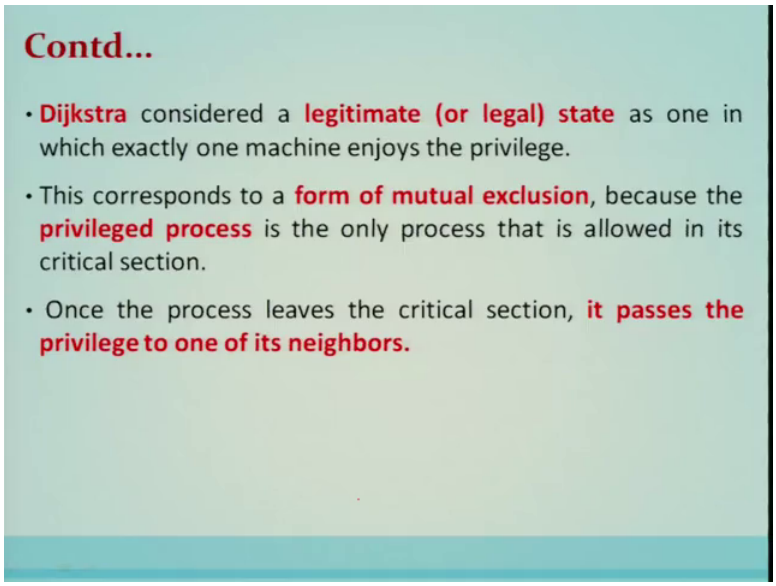
(Refer Slide Time: 19:38)



A legitimate state must satisfy the following constraints. There must be at least one privilege in the system; that is liveness or no deadlock. Every move from legal state must again put the system into a legal state.

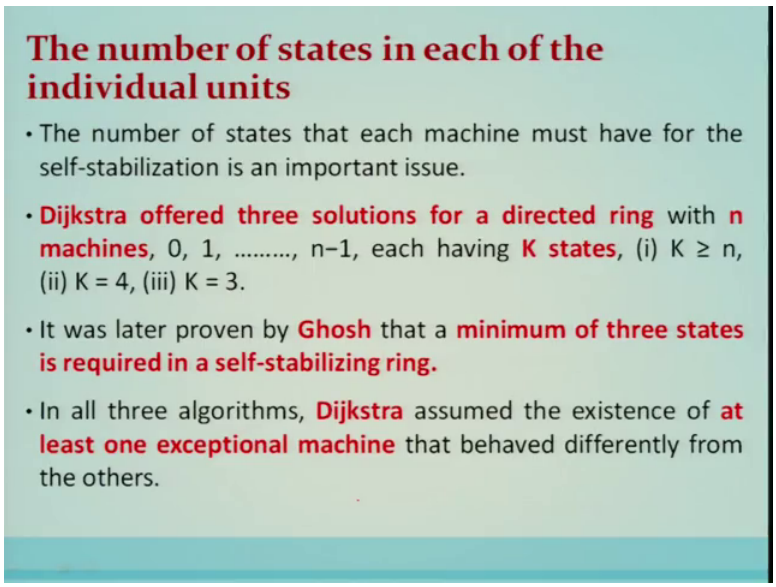
So, from one legal state the system will make a move, and go to another legal move; that is called a closure property. So, during an infinite execution, each machine should enjoy a privilege an infinite number of times.

(Refer Slide Time: 20:21)



So, that is the no starvation condition. So, given any two legal states, there is a series of moves that change one legal state to another; that is called the reachability. So, Dijkstra considered, a legitimate or illegal state as one, which exactly one machine enjoys the privilege as I told you that only one privilege is initially considered by Dijkstra. This corresponds to a form of mutual exclusion, because privilege process is the only process that is allowed in the critical section. Once the process leaves the critical section it passes the privilege to the other nodes; the number of states in each of the individual units.

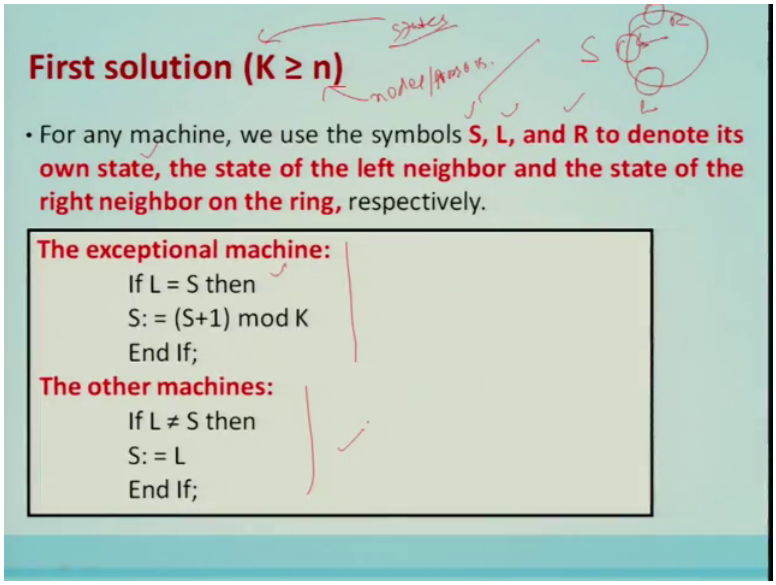
(Refer Slide Time: 20:51)



So, the number of states that each machine must have for the self-stabilization, is an important issue. Not only important issue, but it is a design issue as well. So, in the previously we have seen how to minimize this number of states, and that will be the one of the most important design issue in self-stabilization. So, Dijkstra offered three solutions for a directed ring, with n machines, each having K states. So, the three solutions, where in the first one assumes that the number of states is same as n, or more than that; so greater than or equal to n. The second solution assumes, the number of states is equal to 4, and then you will see that k is equal to 3. That means, the Ghosh has proved later on that, with the number of states is equal to 3, it is possible to design a self-stabilizing system.

So, Ghosh proved that minimum of three-states is required in a self-stabilizing ring. So, in all three algorithms by Dijkstra's assume the existence, of at least one exceptional machine, that behaved differently from others.

(Refer Slide Time: 22:23)

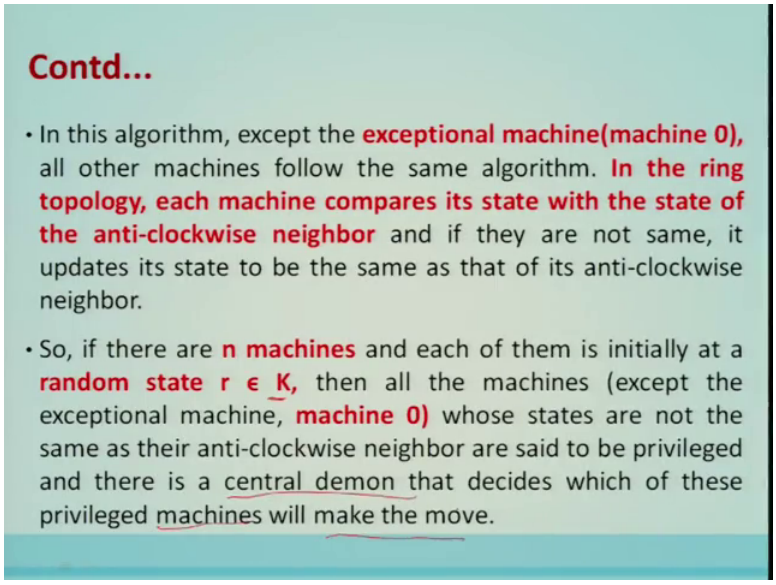


Let us see the first solution where the number of states, is assumed to be greater than or equal to the number of nodes, or the processors in the system. So, this is very loose bound on the number of states. So, for any machine we use the symbols S L R, S means its own state, and L is by state of its left neighbor, and R is the state of its right neighbor on the ring respectively. So, meaning to say, that if this is the ring and this is by state. So,

if this is the current state. So, this is S, its left neighbor is L, and its right neighbor is R respectively.

Now, Dijkstra's assumed one machine which is called an exceptional machine, the code of exceptional machine is like this. If l is equal to s; that is the state of left is equal to the state of the current state, then S the state of S, or the current state will be modified as S plus 1 mod K; the other machines which are not an exceptional machine. So, there if the state; that is left is not equal to the current state, then the current state will be same as the left state.

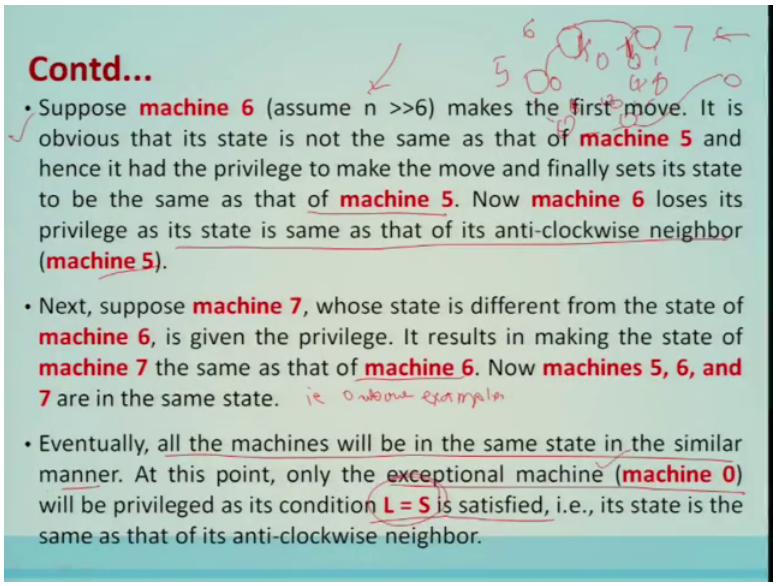
(Refer Slide Time: 24:06)



Now, in this algorithm except the exceptional machine 0 all other machines follow the same algorithm in the ring topology, each machine compares its state with the state of its the anti-clockwise neighbor and if they are not same, it updates its state to be the same, as that of its anti-clockwise neighbor.

So, if there are n machines, and each of them is initially at a random state, drawn from possible set of the states, then all machines except the exceptional machine; that is machine 0 whose states are not the same as their anti-clockwise neighbor are said to be privileged, and there is a central demon which will decide, which among those privileged machine will be allowed by the system to make a move.

(Refer Slide Time: 25:00)



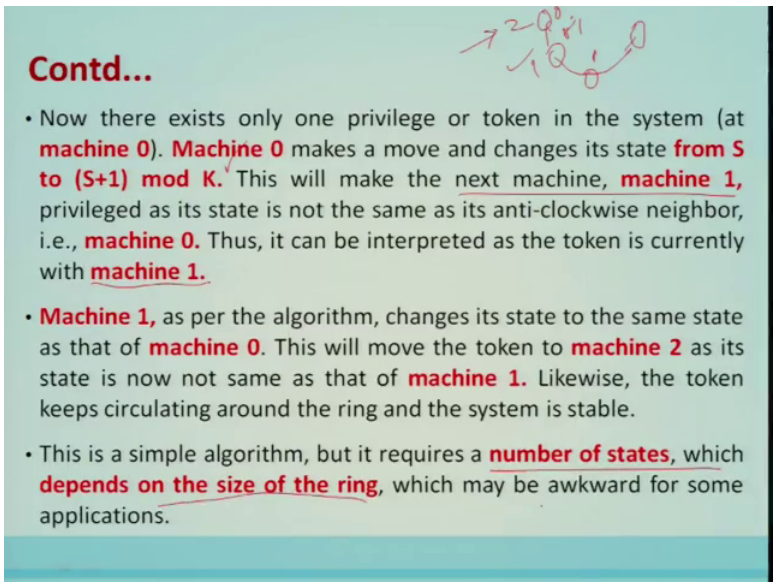
Suppose machine 6 in a system which has the number of mod or more than 6, makes the first move. It is obvious that it state is not the same as that of the machine 5, and hence it has, it had the privilege to make the move. And finally, sets, its state to be the same as that of machine 5. Now machine 6 loses the privilege, as its state is same as that of its anticlockwise neighbor machine.

So, let us see the example of this particular description right over here, before we take it. So, if this machine 6 is going to make a move. So, its neighbor is 5 which is at the left. If it is going to make a move then, the current state of 6 is not equal to current state of 5. So, it will change by state accordingly, let us say that the state of 5 is let us say 0, and the state of 6 is 1 . So, now, the state of 6 will be changed to 0. So, finally, sets the state to be the same as that of machine 6, that we have done here. Now machine 6 loses its privilege, as its state is same as that of anti-clockwise neighbor 5, that I have explained.

Now, next, suppose machine 7 whose state is different from the state of machine 6. Let us say its state is 1, is given the privilege. It is having a privilege why, because its left neighbor is not same as its current a state; that is 0 and one they are different. So, hence it will be a privileged. So, it results in making the state of the machine 7 as that of machine number 6. So, it will become 0 in this case. Now machine 5 6 7, they are in the same state; that is equal to 0 in the above example.

So, eventually if you see the progress, all the machines eventually will basically make a transition, and will have the same state; that is 0, then what will happen. So, eventually all the machines will be in the same state in the similar manner. So, at this point, only an exceptional machine that is a machine 0 will be the privileged, as its condition L is equal to S will be satisfied. So, there exists a machine 0, whose left and right, whose left and the current state both are same here. Then according to the exceptional machine code, if they are same then then they have to make a move, and if state is the same as that of anti-clockwise neighbor.

(Refer Slide Time: 28:23)

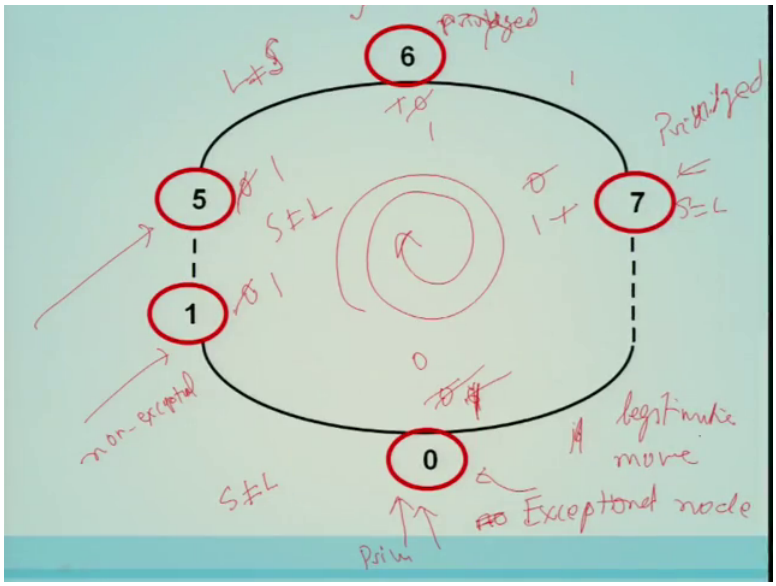


Now, there exists only one privilege or a token in the system; that is machine 0 makes a move and change its state from S to S plus 1 mod K. So, that machine in that case, 0 will make a move and its state will become 1, according to this particular formula. And this will trigger, the next state which is left which is 0. So, they are not same and so on. This particular way the token circulates, or the moon circulates around the ring.

So, this will make the next machine 1, here as I shown you, is privileged as its state is not same as in the anti-clockwise neighbor; that is here. Thus it can be interpreted as a token currently with the machine 1. So, machine 1 will change it to 1 and so on. So, now, it will go to the 2, 2 will is having 0; it will become a privileged and so on. So, the moves will move in this manner, the clockwise way. Although the ring is basically identified in a anti clockwise manner.

So, machine 1, as per the algorithm changing state, to the same state of that machine 0 and move to the machine 2 and so on. So, this is a simple algorithm, but requires the number of states which depends on the size of the ring, which may be awkward for some applications.

(Refer Slide Time: 30:02)



So, I explained you that, if let us say mod 6 is going to make a move, move this if it is a privileged, and privileged means its left, is not equal to the current state that is S . So, let us assume that it is having 0, state is 0, and this state is 1, so they are not equal. If they are not equal, then S will be assigned to L. So; that means, one will be changed to 0, and both will be now having the same state.

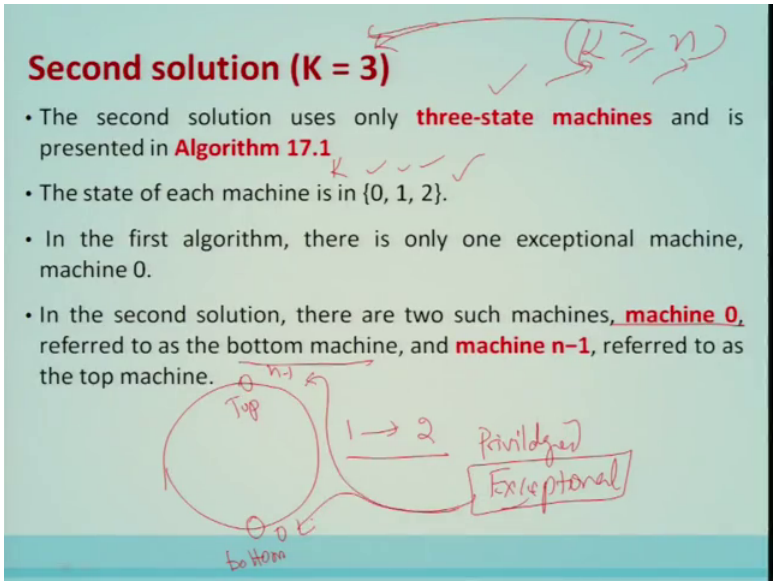
Now, let us assume 7, which is having the state 1. Now these two states are not same. So, this will become privileged. This is no longer privileged because its left is, same as the current state. So, this will become a privileged. So, if it is privileged, then again it will set its state to the same as the left one so; that means, here S will be assigned to the left, left is 0. So, this way eventually all the nodes will be able to change the state, and that is the same state, but there will be an exception

So, according to the convention, there is a non or there is an exceptional node. So, this let us say 0 is an exceptional node, and exceptional node says that, the code of a exceptional node says that, if this particular left S is equal to left, S is not equal to left. Then basically this will change to 1. So, exceptional machine says that if both are equal, then S will be

incremented by 1. So, if both are equal to 1. So, it will be changed to 1 in this case. Now this a non-exceptional node, a non-exceptional node, will see that its left neighbor is 1. So, it is going to change, and it will become, it will change its a state, and it is no longer privileged. So, the privilege will be here why, because both S is not equal to L and so on.

So, just see that all the states will now being rotated, and change to 1, and again finally, come back and found out that this is an exceptional node, and it is now privileged, and it will change again from 1 to 0 and so on. So, keeps on rotating, the token in this particular manner, and this is the legitimate state. So, you just see that whenever there is a move, it will goes from one legitimate state to another legitimate state according to the programs which are defined.

(Refer Slide Time: 33:29)

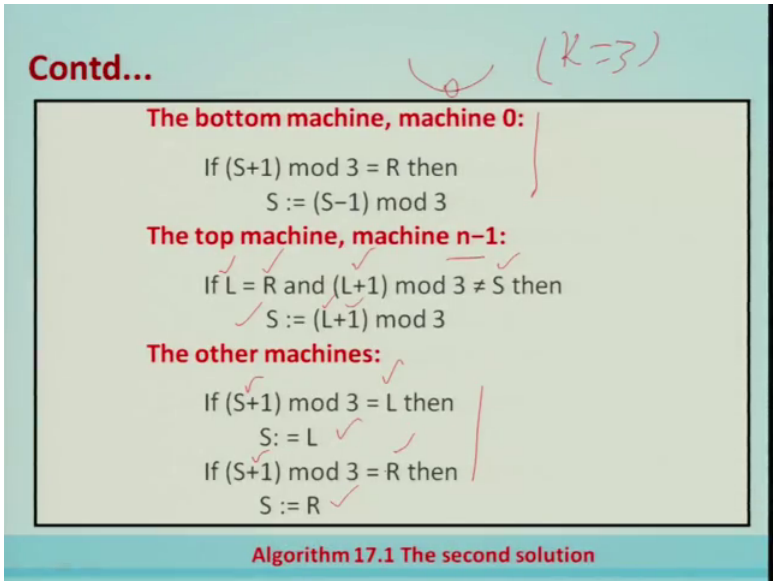


Now there is another solution which says that the previous solution was has considered that K is greater than or equal to n; that means, the numbers of states are too much, depending upon the number of nodes which are there in the system.

Now, the second solution says that, let us assume K is equal to 3. So, the states for the K is equal to 3. So, the states for the K will be 0 1 and 2 which are assigned to a three-state machine, to every node of a system. So, in the first algorithm, there is only one exceptional machine; so by reducing the number of states from n to 3. Now the number of privileged nodes also will be increased from 1 to 2 over here.

So, there will be two machines which are privileged machine, or exceptional machine code. So, these are called machine 0, and which is machine 0 is also called a bottom machine. So, here if you see a ring structure; let us see that this is a 0, so it is a bottom machine and its number is 0 and here this will be a machine number n minus 1, and it is called a top machine, and they are two machines which are exceptional machines.

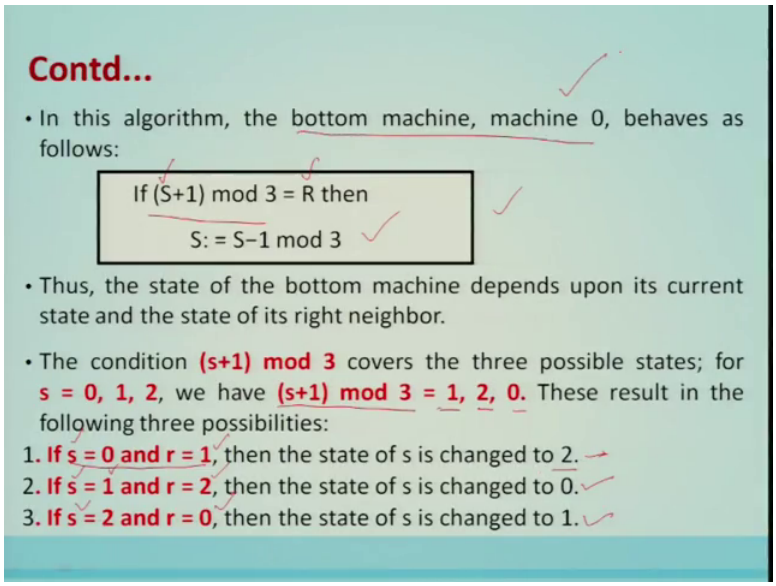
(Refer Slide Time: 35:02)



So, let us see the program, in this particular problem setting where K is equal to 3. So, the bottom machine as I explained you will have this kind of code, if S plus 1 mod 3 is equal to R, R means the neighbor on the right side, then S is equal to S minus 1 mod three, and the top machine which is the machine number n minus 1. If left neighbor is equal to right neighbor, and the left plus 1 mod 3 is not equal to the current state of the machine, then current of the state of the machine is increment, is basically the left plus one.

And all other machines in this case; either of these two rules are applied whichever is correct is being applicable; if the current state plus 1 is equal to the left neighbor state, then state will be same as the left neighbor. Otherwise if S plus 1 mod 3 is equal to the right, then basically S is equal to right.

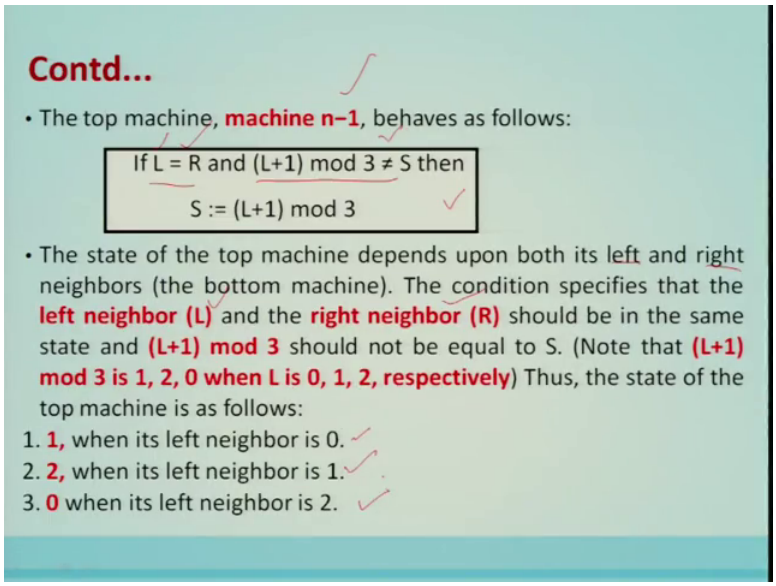
(Refer Slide Time: 36:13)



So, in this algorithm the bottom machine 0 behaves as follows; that we have seen. Thus the state of the bottom machine depends upon the current state, and the state of its right neighbor. The condition S plus 1 mod 3 covers three possible states, for S is equal to 0 1 and 2. Thus we have S plus 1 mod 3, is equal to 1 2 and 0; these results in the following three possibilities. So, when S plus 1 mod 3 is equal to 1, when S plus 3 mod 3 is equal to 2, and when it is 3, and it is same as R, then S will be 0 1 and 2.

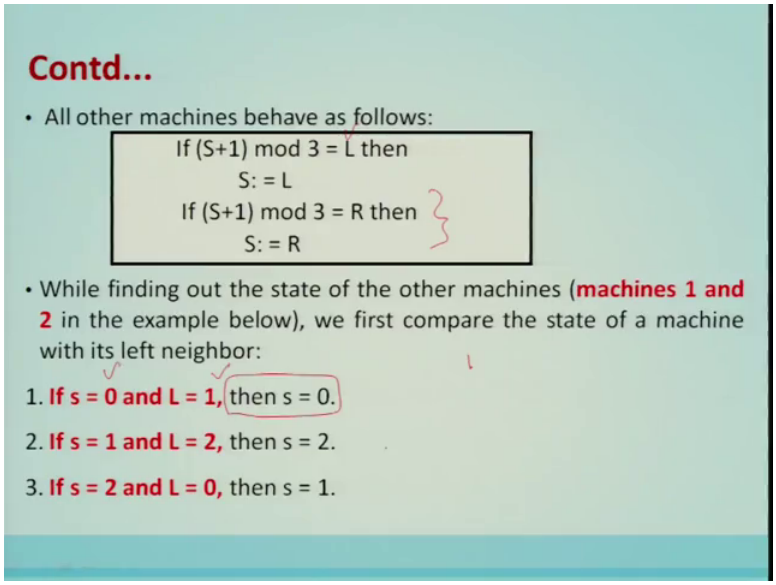
So, let us see that when S is equal to 0. Here if you see S is equal to 0, and R is equal to 1, then the state of S will be changed to S minus 1 that is 2, when S is equal to 1 here, and R is equal to 2 then S will be changed to 0, because S minus 1, S minus 1 1 minus 1 will become 0. Similarly in this condition S minus 1 means 1. So, this particular case is considered for the bottom machine; that is machine number 0 will behave in this manner.

(Refer Slide Time: 37:48)



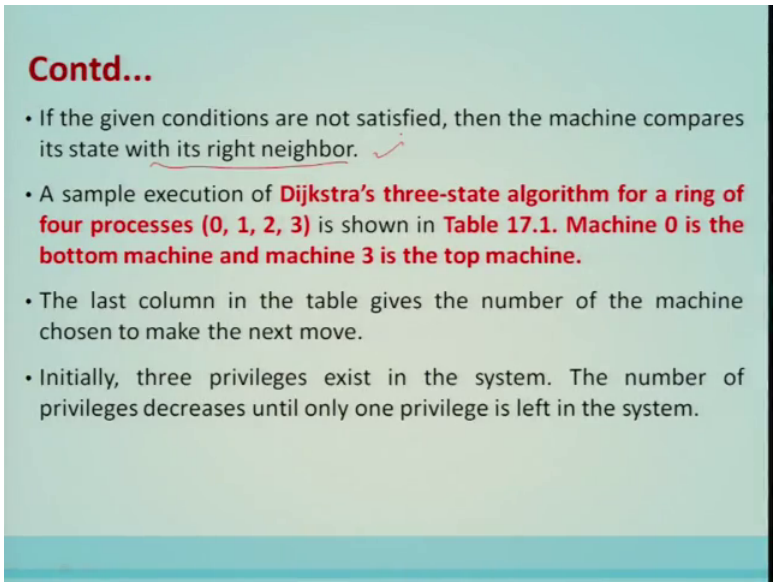
Similarly we can see about the top machine; that is the machine number n minus 1 will behave according to this rule. So, the top machine depends upon both its left and right neighbors. The condition specifies that the left neighbor L and the right neighbor R should be in the same state here. And L plus 1 mod 3 should not be equal to 2 S, the other condition connected by an end. Note that L plus 1 mod 3 is 1 2 and 0, when L is equal to 0 1 and 2 respectively. Thus the state of the top machine will be assigned to 0, and it will be assigned to 0, and it will be assigned to 0 and it will be assigned 2, from 1 to n 0 according to this particular rule.

(Refer Slide Time: 38:46)



Now, all other machines behave as follows, as I explained you, while finding out the states of the other machines machine 1 and machine 2 let us for example, below we first compare the state of, state with its left. So, when S is equal to 0, and L is equal to 1. So, S is equal to 0, this will become 1, and L is also 1, both are equal, then S will be assigned to L. If it is not assigned to L, then basically the second rule will be basically considered and so on.

(Refer Slide Time: 39:30)



So, if the given conditions are not satisfied, then the machine compares its state with its right neighbor that I explained in the previous example. So, the same execution of Dijkstra's three-state algorithm for a ring of 4 processors is shown in the next table. Machine 0 is the bottom machine, and machine 3 is the top machine. The last column the table gives the number of machines chosen to make a move, initially 3 privileges exist in the machine, the number of privileges decreases only 1 privilege is in the left.

(Refer Slide Time: 39:59)

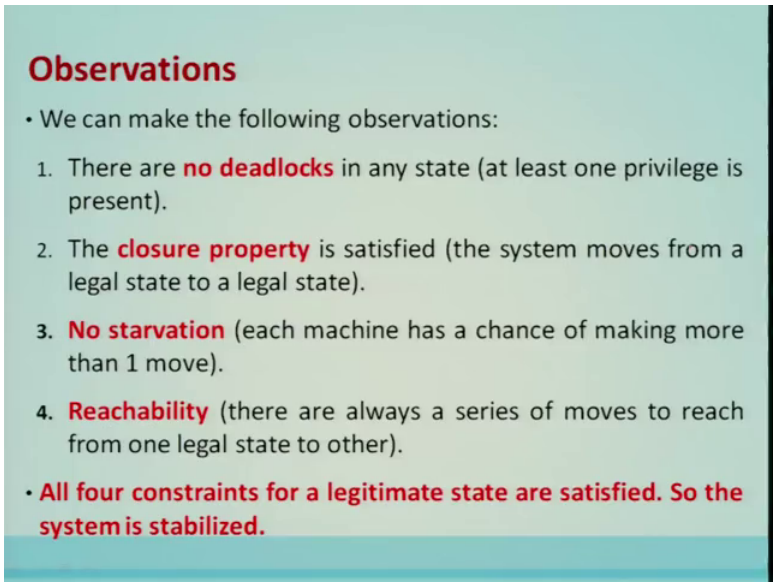
Table 17.1 An example execution of Dijkstra's three-state algorithm

State of machine 0	State of machine 1	State of machine 2	State of machine 3	Privileged machines	Machine to make move
0	1	0	2	0, 2, 3	0
2	1	0	2	1, 2	1
2	2	0	2	1	1
2	0	0	2	0	0
1	0	0	2	1	1
1	1	0	2	2	2
1	1	1	2	2	2
1	1	2	2	1	1
1	2	2	2	0	0
0	2	2	2	1	1
0	0	2	2	2	2
0	0	0	2	3	3
0	0	0	1	2	2

So, initially you see that there are three privileged machine are there, and then it will be moved to two privileges, and finally, one privilege at a time onwards. Now, as far as the machine is concerned. So, let us go back again. So, machine 0 is the bottom machine, and machine 3 is the top machine. So, this is the bottom machine, and machine 3 is the top machine. These two codes are different, and these are the other machines which will follow a different code.

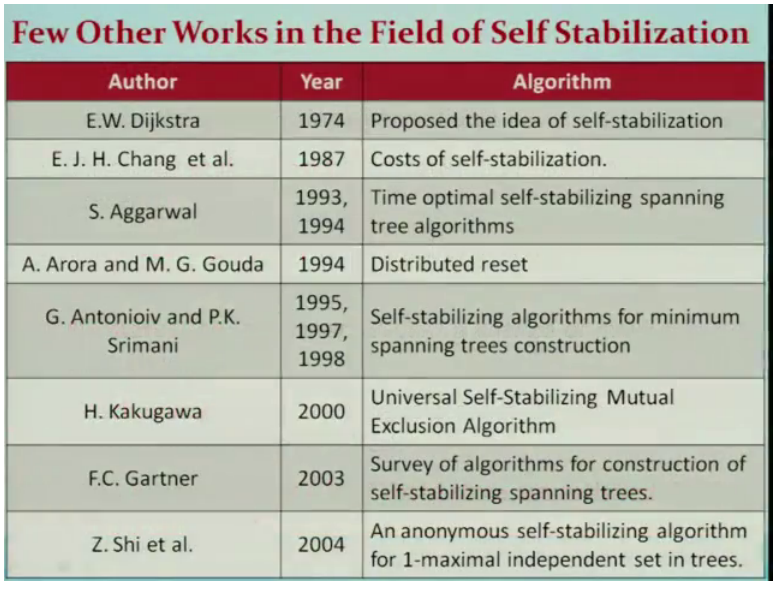
So, on the last column, here you can see that the machine to make a move is basically mentioned over here; that means, these are the privileged machines. If more than one privileges are there, then central demon will select one, and it is allowed to make a move. So, here out of two, this is allowed to make a move in all other cases, the same machines.

(Refer Slide Time: 41:08)



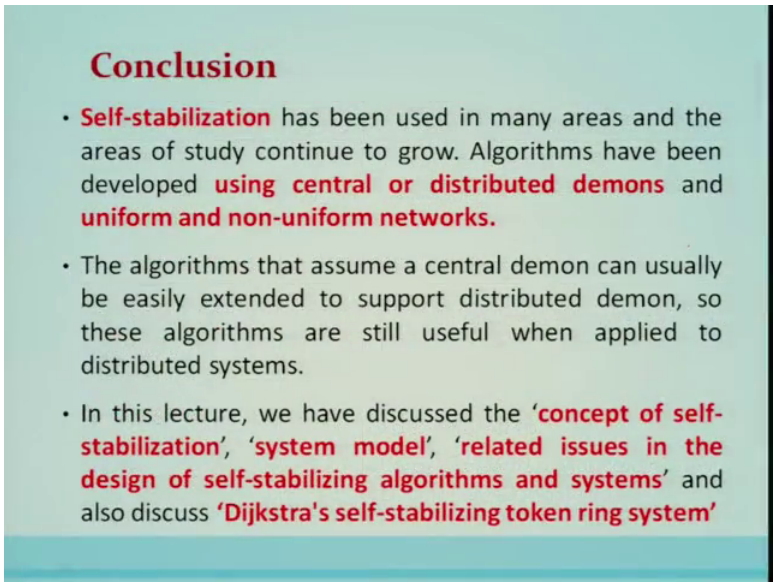
Same previous node will be allowed to make a move in this particular diagram. Observations: so we can make the following observations, there are no deadlocks in any state. The closure property satisfied, no starvation is also satisfied, reachability is also there.

(Refer Slide Time: 41:24)



So, all four constraints for a legitimate state are satisfied. So, the system is stabilized. Few other works in the field of self-stabilization are mentioned here for further reading.

(Refer Slide Time: 41:33)



Conclusion self-stabilization has been used in many areas, and the areas of study continues to grow. Algorithms have been developed using central and distributed demons and uniform and non-uniform algorithms. The algorithm that assumed the central demon can usually be easily extended to support distributed demon.

So, these algorithms are still useful when applied to the distributed system. In this lecture we have discussed the concept of self-stabilization, system model, related issues in the design of self-stabilizing algorithms and systems, and also discussed Dijkstra's self-stabilizing token ring system.

Thank you.